**Orange**

November 2024

Khiops 10.2 Guide

- **Khiops Desktop**
  - Optimal data preparation based on discretization and value grouping
  - Scoring models for classification and regression
  - Correlation analysis between pairs of variables
  - Automatic variable construction for multi-table relational mining

- **Khiops Visualization**
  - Analysis of Khiops results using an interactive visualization tool

- **Khiops Coclustering**
  - Correlation analysis of two or more variables using a hierarchical coclustering model
  - Prediction models for cluster assignment

- **Khiops Covisualization**
  - Exploratory analysis of Khiops Coclustering results using an interactive visualization tool

This guide is about the Khiops component.

Abstract

Khiops is an automatic data preparation and scoring tool for supervised learning and unsupervised learning in the case of large multi-tables databases.

It allows to perform univariate and bivariate descriptive statistics, to evaluate the predictive importance of explanatory variables, to discretize numerical variables, to group the values of categorical variables, to recode input data according to these discretizations and value groupings. It allows to perform multi-table relational mining with automatic variable construction. Khiops also produces a scoring model for supervised learning tasks, according to a Selective Naive Bayes approach extended with trees, either for classification or for regression.

Khiops run on multi-cores machines under multiple platforms. It is available both in user interface mode and in batch mode, such that it can easily be embedded as a software component in a data mining deployment project.

This paper describes the parameters of Khiops and all its functionalities, the management of the data dictionaries and the list of derivation rules that enable to construct new variables, and finally the analysis and evaluation reports. The user interface is presented as well, together with a quick start of the tool.

Summary

7. Appendix: data preparation and modeling derivation rules 74

7.1. Technical structures 74

7.2. Vectors 75

7.3. Hash maps 76

7.4. Data preparation 76

7.5. Recoding 77

1. Presentation

Khiops is an automatic data preparation and scoring tool for supervised learning and unsupervised learning in the case of large multi-tables databases.

Khiops allows to quickly perform the descriptive and explanatory phases in a Data Mining project. The database must be formatted according to a text file format, with a line per record, one header line containing the variable names and a field separator (tabulation by default).

The first step is the specification of the data dictionary, which is the choice of the variable types (Categorical, Numerical, Date, Time or Timestamp) in the database to analyse. This dictionary is automatically built by Khiops owing to a parsing of the database file. The built dictionary is saved in a dictionary file, which basic syntax allows easy modifications. The Data Miner must then validate the variable types in the built dictionary, and eventually specify which variables to ignore in the analysis or construct new variables owing the derivation rule language.

The second step is to check the correctness of the database file. In this step, Khiops parses the database file and completely checks formatting or variable type errors.

The third step, the most important, is to analyse the predictive value of the explanatory variables or pairs of variables. In supervised analysis, when a target variable is specified, Khiops evaluates the predictive importance of any numerical or categorical explanatory variable, and of any pair of explanatory variable. In unsupervised analysis, when no target variable is specified, Khiops evaluates the correlation between any pair of variables. The evaluation is based on discretization for numerical variables, value grouping for categorical variables, and bivariate discretization/grouping for pairs of variables. Two reports, for univariate and bivariate analysis, are produced at the end of the data analysis, based on the train data set. They summarize the information contained in each analysed variable or pair of variables.

In the case of supervised tasks, a scoring model is computed as well, based on a Selective Naive Bayes predictor (predictors). To leverage the limitation of the Selective Naive Bayes that considers the input variables independently, trees can be constructed in the case of classification. A modeling report summarizes the features of the built classifier or regressor. Two evaluation reports, based on the train and test data, evaluate the performance of the scoring model. New dictionaries, recoding dictionary and scoring dictionary are produced, allowing a deployment of the data preparation recoding specifications or of the scoring model.

The fourth step is the deployment step. This is done by applying the recoding dictionary or the scoring dictionary on new data, in order to compute recoding variables or score variables. This functionality can also be used to construct any new variable, described using the derivation rule language.

Khiops can also be used for multi-table relational mining, with star schemas or snowflake schemas. For example, data can be structured with one root entity and several sub-entities in 0-1 or 0-n relationship (e.g.: Customer and list of Sales per customer). Khiops is able to produce an instances * variables flat table using automatic variable construction. Throughout this paper, all Khiops features related to multi-table relational mining are identified using the thumbnail in the left margin. The related sections or paragraphs can be skipped when single table Khiops features fulfil the requirements.

All analysis Khiops results can be visualized using the Khiops visualization and covisualization tools. They are available as text files (.xls) and can be opened using a spreadsheet tool. They can also be exported as JSON files and can be exploited from any programming language (such as python) for an easy integration in information systems.

A Khiops session can be registered in a scenario file, which can be replayed by Khiops in batch mode. This allows to automatize data preparation, modelling and deployment in a Data Mining project and to easily integrate the process in any information system. Khiops libraries are also available: KNI (Khiops Native Interface) for online deployment of models, and khiops-python to drive the whole data mining process from python.

2. Detailed functionalities

This section is organized as the user interface of Khiops. The subtitles correspond to the sub-windows or submenus of Khiops main window. The options described in each subsection correspond to the fields or menu actions available in the interface.

2.1. Dictionary file
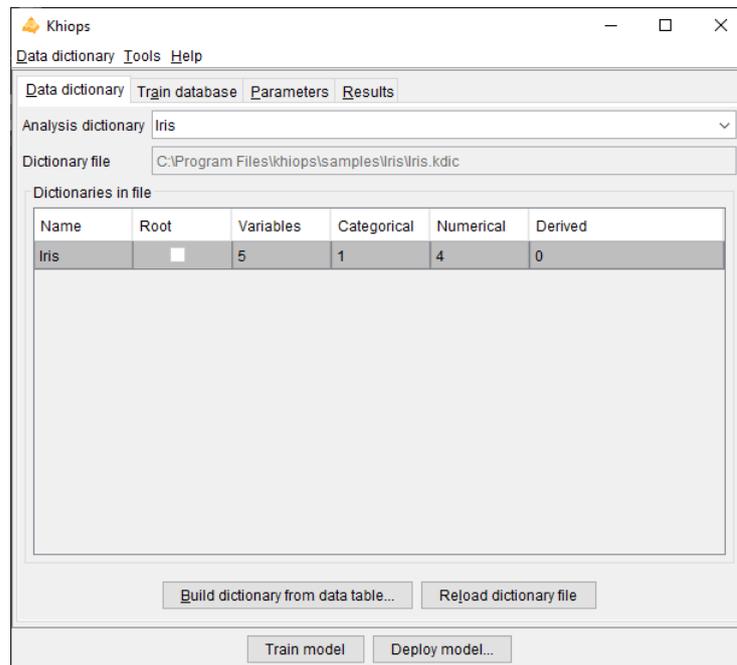
A dictionary file is a text file with the extension .kdic. It contains the definition of one or several dictionaries, each one describing the set of variables to use in a data analysis. The dictionaries can be automatically built from the data table file owing to a file parsing, automatically enriched during data preparation or modeling, or manually modified by the Data Miner using a text editor (for example Notepad). The dictionary files built by Khiops use tabulations as field separators, which allows direct copy-paste interactions with Excel. This provides a way to quickly sort, select and modify large numbers of variable definitions.

Khiops allows to **Open** a **Dictionary file**. Opening a dictionary file amounts to loading its dictionaries into memory and making them available for data analysis. The **Save** and **Save as** actions write dictionaries to a dictionary file, whereas the **Close** action cleans the memory. The **Build dictionary from file** action builds dictionaries from data files and save them in a dictionary file. The **Reload dictionary file** action reads again a dictionary file, which may have been modified using an external text editor. The list of available dictionaries can be browsed using Khiops.

Analysis dictionary: name of the dictionary related to the data to analyze. Mandatory field.

Dictionary file: name of the dictionary file related to the data to analyse. Read-only field that shows the name of the current dictionary file.

Dictionaries in file: list of available dictionaries, with statistics describing the used variables (Name, Variables, Categorical, Numerical, Derived).

Dictionary files are opened only during the execution of the actions. They are then loaded into memory and available everywhere inside Khiops.

2.1.1. Build dictionary from data table
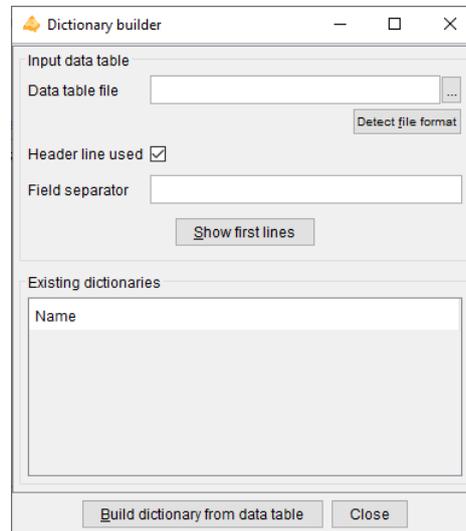
This action opens a dialog box that allows to build dictionaries from data tables, then saves them in a dictionary file.

Data table file: name of the data table file to analyse. Mandatory field.

Detect file format: heuristic help that scans the first few lines to guess the file format. The header line and field separator are updated on success, with a warning or an error in the log window only if necessary. Note that the heuristic may fail in some cases: for example, a header line is only detected if all labels in the first line are distinct and contain at least one alphabetical character.

Header line used: (default: true). If the file has a header line, Khiops will use the header line fields as variables names; otherwise, the variables will be names Var1, Var2...

Field separator: by default, if nothing is specified, the tabulation is used as the field separator.

Show first lines: shows first lines of data table in log window.

Build dictionary from data table: starts the analysis of the data table file to build a dictionary. The first lines of the file are analyzed in order to determine the type of the variables: Categorical, Numerical, Date, Time or Timestamp. After analysis, the user can choose the name of the dictionary.

Close: closes the window. If dictionaries have been built, proposes to save them in a dictionary file

The values in the data table file are parsed in order to guess their type.

- values with format YYYY-MM-DD (e.g. 2014-01-15) are recognized as Date variables,
- values with format HH:MM:SS (e.g. 11:35:20) are recognized as Time variables,
- values with format YYYY-MM-DD HH:MM:SS are recognized as Timestamp variables,

- values with format YYYY-MM-DD HH:MM:SS.zzzzzz are recognized as TimestampTZ variables, with time zone information,

- for other Date, Time, Timestamp or TimestampTZ formats (e.g. Date format DD/MM/YYYY), a specific meta-data value is used (see paragraph 3.2.2. Dictionary) to specify the format used for the variable,

    o DateFormat: see section 6.7. Date rules

    o TimeFormat: see section 6.8. Time rules

    o TimestampFormat: see section 6.9. Timestamp rules

    o TimestampTZFormat: see section 6.10. TimestampTZ rules

- other values with numerical format are recognized as Numerical values,

- other values are recognized as Categorical values.

Dictionaries are built automatically for convenience, but they should be checked carefully by the data miner. For example, zip codes are made of digits and recognized as Numerical variables, whereas they are Categorical variables. Values such as 20101123 or 20030127 are recognized as Date with format YYYYMMDD, whereas they could be Numerical.

The Date, Time or Timestamp formats can be erroneous (example: 2010-10-10 is ambiguous w.r.t. the format: "YYYY-MM-DD" or "YYYY-DD-MM"). The meta-data must be corrected directly in the dictionary file is necessary. In some cases, a date, time or timestamp variable may have a format not recognized by Khiops. This the case for example for formats where the century is not specified (e.g: "YY-MM-DD"). In that case, the corresponding variable should be declared as Categorical (and not used), and a new variable can be built using derivation rules, as illustrated below.

```
Unused      Categorical MyDate ;          // Date with unrecognized format "YY-MM-DD"
            Date        MyCorrectDate = AsDate(Concat("20", MyDate), "YYYY-MM-DD");      // Correction if all centuries are 20th
Unused      Numerical   MyCentury = If(LE(AsNumerical(Left(MyDate, 2)), 15), 20, 19);    // 20th for year below 15, 19th otherwise
            Date        MyCorrectDate2 = AsDate(
                            Concat(AsCategorical(MyCentury), MyDate), "YYYY-MM-DD");  // Correction using MyCentury
```

In case of multi-table databases, after building and checking the resulting dictionary file, the data miner has to modify the dictionary file using a text editor in order to specify the relations between the dictionaries of the multi-table database. See section 3.2.3. Multi-table dictionary.

2.1.2. Reload dictionary file

Reload the current dictionary file into memory.

This action allows to modify the dictionary file using an external text editor (Notepad for example), to save the modifications, and to take them into account into Khiops by reloading the dictionary file.

In case of invalid dictionary file, the current dictionaries are kept in memory.

2.1.3. Dictionary file menu
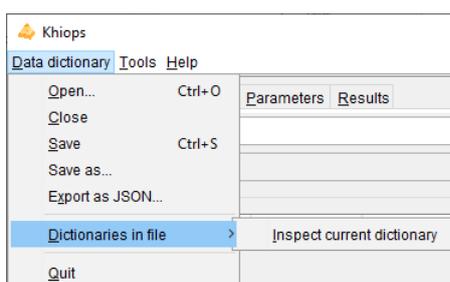
2.1.3.1. Open

An open dialog box asks the name of the dictionary file to open.

In case of invalid dictionary file, the current dictionaries are kept in memory.

2.1.3.2. Close

The dictionaries are removed (from memory only). The potential pending modifications are lost if they have not been saved.

2.1.3.3. Save

The memory dictionaries are saved under the current dictionary file.

2.1.3.4. Save as

A save dialog box asks the name of the dictionary file to save.

2.1.3.5. Export as JSON

A save dialog box asks the name of the JSON file to export the dictionaries under a JSON format, with a .kdicj extension.

2.1.3.6. Dictionaries in file/Inspect current dictionary
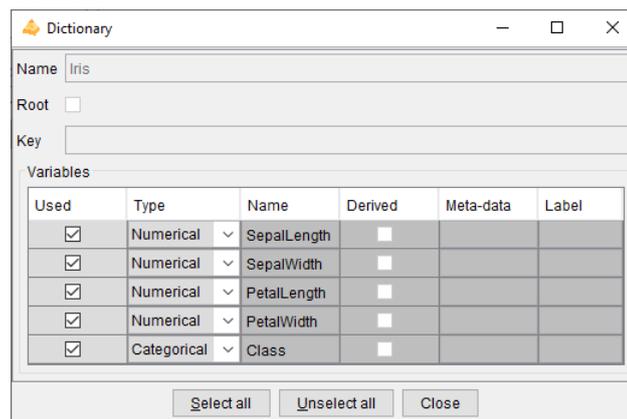
Allows to inspect and partly modify a dictionary chosen among the list of available dictionaries. The dictionary to inspect must be selected among the dictionaries in file.

The action is available both from the menu and using a right click button on the selected dictionary.

During the inspection of a dictionary, the list of its variables can be browsed into a sub-window. For each variable, the following properties are displayed: **Used**, **Type**, **Name**, **Derived**, **Meta-data** and **Label**.

The Data Miner can choose whether to keep or not the variable for data analysis, using the **Used** property. The **Select all** and **Unselect all** buttons allow to choose all or no variables.

It is also possible to change the **Type** of variables: Numerical, Categorical, Date, Time or Timestamp.

Remark: for complex or large scale modifications in a dictionary, it is preferable to update the dictionary file using an external text editor (Notepad, WordPad…), to save the file with the external editor, and then to reload the dictionary.

For multi-table dictionaries, the **Root** property indicates whether the dictionary relates to a Root entity, and the **Key** field recall the key variables of the dictionary. The type of variables is extended to store the relationships among entities: **Entity** for 0 to 1 relationship and **Table** for 0 to n relationship.

2.1.3.7. Quit

Quits the application.

2.2. Train database
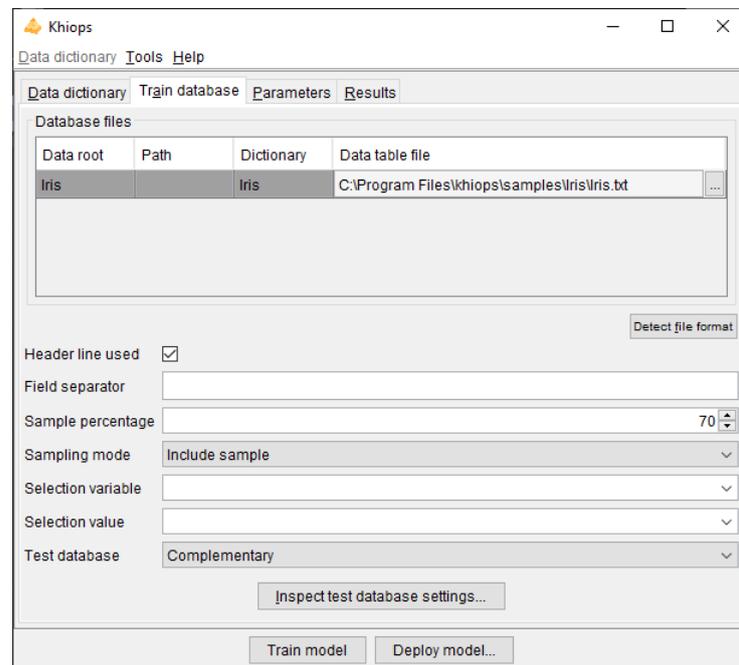
Database files: name of the database files to analyse.

Data table file: name of the data table file. Mandatory field.

Detect file format: heuristic help that scans the first few lines to guess the file format. The header line and field separator are updated on success, with a warning or an error in the log window only if necessary.

Header line used: (default: true). If the file does not have a header line, Khiops considers the variables in the dictionary to analyse the fields in the file.

Field separator: Character used as field separator in the file. It can be space (S), semi-colon (;), comma (,) or any character. By default, if nothing is specified, the tabulation is used as the field separator.

Khiops can be used to extract a subpart (or its exact complementary) of the records in a database file. This sampling is specified with a sample percentage of the records (to keep or to discard). The sampling is a random sampling, but is reproducible (the random seed is always the same).

Sample percentage: percentage of the samples (default: 70%)

Sampling mode: to include or exclude the records of the sample (default: include sample). This allows to extract a train sample and its exact complementary as a test sample (if the same sample percentage is used both in train and test, in include sample mode in train and exclude sample mode in test).

Another way to build train or test samples is to use a selection variable and a selection value.

Selection variable: when nothing is specified, all the records are analysed. When a selection variable is specified, the records are selected when the value of their selection variable is equal to the selection value.

Selection value: used only when a selection variable is specified. In that case, the value must be a correct value (numerical value if the selection variable is a numerical variable).

Test database: specification of the test database, according to one of the following choices:

- Complementary (default): same as the train database, with 'Sampling mode' inverted in the test database, in order to get test samples that are the exact complementary of the train samples,

- Specific: specific parameters for the test database,

- None: no test database is used.

For multi-table databases, there are potentially several lines in the array of database files. The data path (Data root, Path) represents the semantic path of the table, that is, its source dictionary followed by the chain of variable names leading to it. All these names separated by a backquote " ` " form the data path. In the GUI, data paths are automatically build, and one data table file must be specified per data path in the multi-table dictionary. Each data table file must be sorted by key. In root tables, keys play for role of identifiers, so that root entities must be unique per key. This is not the case for sub-entities in zero to many relationship with the root entity.

2.2.1. Inspect test database settings

This action allows to inspect the test database parameters.

The test parameters are editable only in the case of a specific test database.
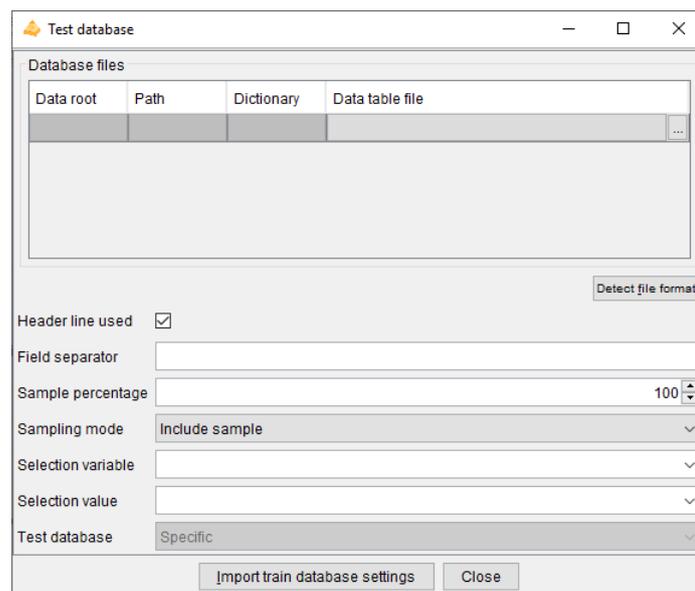
The test database is defined exactly in the same way as the train database.

In the case of a specific database, there is an additional button to import the train database parameters. It allows to fill all the test database fields, by copying them from the 'Train database' pane. The only change is the 'Sampling mode' which value is inverted in the test parameters, in order to get a test sample that is the exact complementary of the train sample.

2.3. Parameters
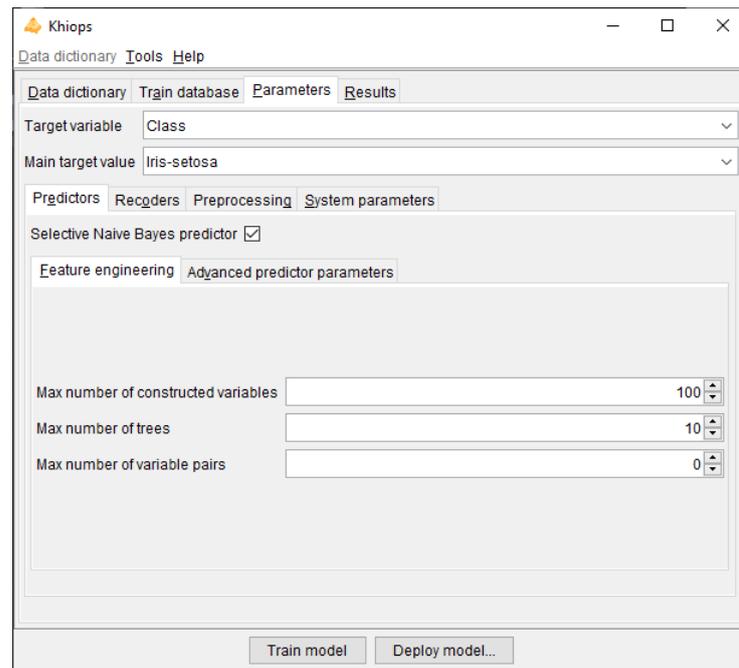
Target variable: name of the target variable. The learning task is classification if the target variable is categorical, regression if it is numerical. If the target variable is not specified, the task is unsupervised learning.

Main target value: value of the target variable in case of classification, for the lift curves in the evaluation reports.

2.3.1. Predictors

Khiops builds predictors only in case of a supervised learning task.

Selective Naive Bayes predictor: builds a Selective Naive Bayes predictor (default: true).

The Selective Naive Bayes predictor performs a "soft" variable selection by directly optimizing the variable weights. This improves the accuracy, the interpretability and the deployment time, with few selected variables.

The variable weights are reported in the evaluation report. The overall training time is O(NKlog(NK)) where N is the number of instances and K the number of variables. As the algorithm is parallelized, it efficiently benefits from multi-core machines.

2.3.1.1. Feature engineering

Khiops performs automatic feature engineering by constructing variables from multi-table schema, building trees and analysing pairs of variables.

Max number of constructed variables: max number of variables to construct (default: 100). The constructed variables allow to extract numerical or categorical values resulting from computing formula applied to existing variable (e.g. YearDay of a Date variable, Mean of a Numerical Variable from a Table Variable).

Max number of trees: max number of trees to construct. The constructed trees allow to combine variables, either native or constructed (default: 10). Construction of trees is not available in regression analysis.

Max number of pairs of variables: max number of pairs of variables to analyze during data preparation (default: 0). The pairs of variables are preprocessed using a bivariate discretization method. Pairs of variables are not available in regression analysis.

By default, few features are constructed to get a good trade-off between accuracy, interpretability and deployment speed. Maximum interpretability and deployment speed can be achieved by choosing no feature to be built. Conversely, choosing more feature to construct allows to train more accurate predictors at the cost of computation time and loss of interpretability.

- variable construction

   o allows to exploit a multi-table schema, by automatically flattening the schema into an analysis table that summarizes the information in the schema

   o being automatic, robust and scalable, accelerates the data mining process to obtain accurate predictors

   o constructed variables remain understandable by the mean of human readable names, in the limit of their complexity

   o recommendation: increase incrementally with 10, 100, 1000, 10000… variables to construct, to find a good trade-off between computation time and accuracy

- trees

   o allows to leverage the assumption of the Selective Naive Bayes classifier that considers the input variables independently, to improve accuracy

   o combines natives or constructed variables to extract complex information

   o tree-based variables are categorical variables, which values are the identifiers of the leaves of a tree; they are pre-processed like other categorical variables (although they do not appear in the preparation report), then used by the classifier, for potentially improved classification performance

   o tree based variables are black-boxes, with potentially improved accuracy at the expense of loss of understandability

   o recommendation: increase incrementally with 10, 20, 50, 100… trees to construct, to find a good trade-off between computation time and accuracy

- variables pairs

   o allows to understand the correlation between variables either in supervised classification (not available in regression) or unsupervised learning tasks

   o the analysis of pairs of variables is a time consuming operation

   o in the supervised case, a variable pair is considered to be informative if it brings more information than both variables individually (see criterion DeltaLevel); non informative pairs are summarized in analysis reports, but not used in predictors.

   o recommendation: build variable pairs for exploratory analysis of correlations rather than to improve the accuracy of predictors

Constructed variables are stored in the output dictionaries (recoding or modeling dictionary), with formula that allow to compute their values during model deployment.

2.3.1.2. Advanced predictor parameters
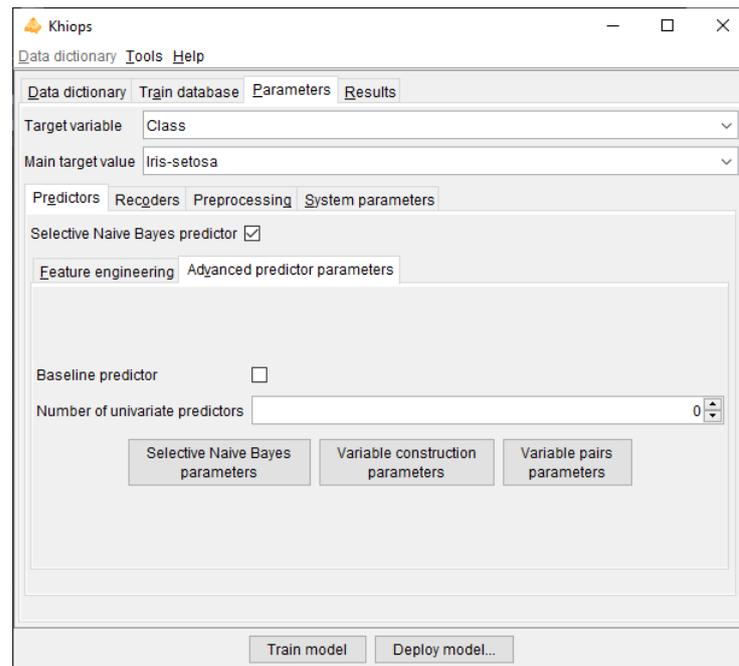
Baseline predictor: builds a baseline model (default: false).

The baseline classifier predicts the train majority class whereas the baseline regressor predicts the train mean of the target variable.

Number of univariate predictors: number of univariate predictors, built and evaluated.

The univariate predictors are chosen according to their predictive importance, which is assessed during the analysis of the train database. Each evaluation of a univariate predictor requires a read pass of the test database (default: 0).

2.3.1.3. Selective Naive Bayes parameters

These parameters are user constraints that allow to control the variable selection process. Their use might decrease the performance, compared to the default mode (without user constraints).
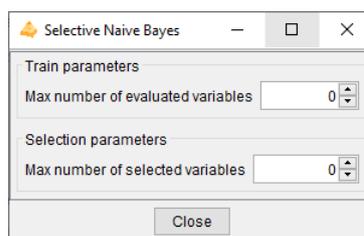
Train parameters

Max number of evaluated variables: max number of variables originating from the data preparation, to use as input variables in the multivariate selection of the Selective Naive Bayes predictor. The evaluated variables are those having the highest predictive importance (Level). This parameter allows to simplify and speed up the training phase (default: 0, means that all the variables are evaluated).

Selection parameters

Max number of selected variables: max number of variables originating from the multivariate variable selection, to use in the final Selective Naive Bayes predictor or MAP Naive Bayes predictor. The selected variables are those with the largest importance in the multivariate selection. This parameter allows to simplify and speed up the deployment phase (default: 0, means that all the necessary variables are selected).

2.3.1.4. Variable construction parameters

New variables can be constructed manually by the data miner using the construction language described in Appendix exploits efficiently a subset of construction rules, defined below.

Automatic variable construction exploits the set of construction rules specified in the **Variable construction parameters** window.

The construction rules applied to Date, Time or Timestamps variables allow to extract numerical values at different periodicities (e.g. year day, month day or week day from a Date variable). By default, these date and time rules are not selected. They are interesting for exploratory analysis. For supervised analysis, they should be used with caution as the deployment period may be different from the training period.

The **Select all** and **Unselect all** buttons allow to choose all or no construction rules, and the **Default** button restores the initial selection. The **Used** checkboxes allow to select construction rules one by one.

In case of multi-table databases, construction rules can be applied to Entity or Table variables. They allow to extract values from sub-entities, such as the mean of the costs of sales of a customer. The TableSelection rule can be combined with other rules in order to be applied on a subset of the sub-tables (e.g. mean cost of sales of a customer for sales related to a given category of products). Given the combinatorial number of potential selection formula, thousands of variables can be constructed automatically.

Several heuristics are applied during the variable construction process, whenever possible:

- Constructed variable do not exploit key variables of secondary tables in multi-table dictionaries, since they are mostly redundant with the key variables of the main table.

- Constructed variables that exploit the same derivation rules as existing initial variables (used or unused) would be redundant. They are not constructed and new variables are constructed instead.

- Constructed variables that exploit different parts of the same partition of a secondary table (via the TableSelection construction rule) may be grouped in a sparse variable block.

- To optimize the overall computation time, temporary variables are also constructed, since they can be reused as operands of several constructed variables. Still, if a temporary variable exploits the same derivation rule as an existing initial variable, the temporary variable is not constructed and the initial variable is used instead.

2.3.1.5. Variable pairs parameters

These parameters allow you to choose to analyze all potential variables pair, or to select individual variable pairs or families of variable pairs involving certain variables to analyze first.
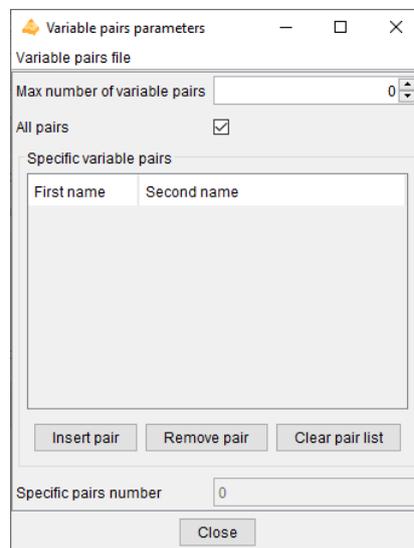
Max number of pairs of variables: max number of pairs of variables to analyze during data preparation (default: 0). The pairs of variables are preprocessed using a bivariate discretization method. Pairs of variables are not available in regression analysis.

If the number of pairs specified is greater than this maximum value, the pairs are chosen first for the specific pairs, then for the pairs involving the variables with the highest level in the supervised case and by alphabetic order otherwise.

All pairs: Analyzes all possible variable pairs.

Specific variables pairs: Allows to specify a specific list of variables pairs. A variable pair can be specified with a single variable to indicate that all pairs involving that variable should be analyzed.

- **Insert pair**: Adds a variable pair

- **Remove pair**: Removes a variable pair

- **Insert pair**: Removes all specific variable pairs

The list of specific variable pairs is cleaned up on closure, removing pairs with syntactically invalid names and redundant pairs.
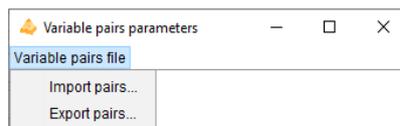
Variable pairs file

- **Import pairs…**: Imports a list of variable pairs from a tabular text file with two columns. Invalid or redundant pairs are ignored during import.

- **Export pairs…**: Exports the list of variable pairs to a tabular text file with two columns. Only valid and distinct pair are exported.

2.3.2. Recoders

Khiops builds recoders in case of a supervised or unsupervised learning task, to enable the recoding of an input database. The recoded database may then be exploited outside the tool to build alternative predictors while benefiting from Khiops' preprocessing.

Build recoder: Builds a recoding dictionary that recodes the input database with a subset of initial or preprocessed variables.
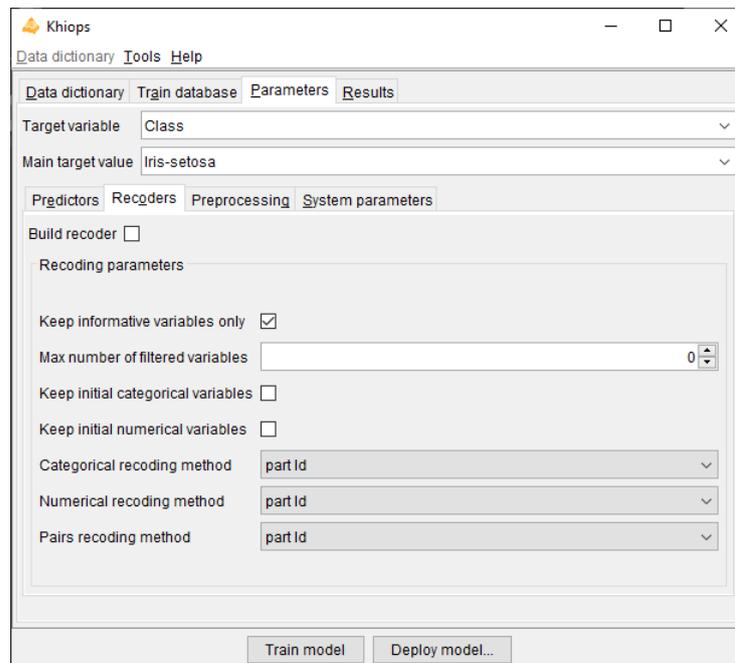
Keep informative variables only: if true, all the noninformative variables are discarded, in their initial or recoded representation (default: true).

Max number of filtered variables: max number of variables originating from the univariate data preparation (discretizations and value groupings), to keep at the end the data preparation. The filtered variables are the ones having the highest univariate predictive importance, aka Level. (default: 0, means that all the variables are kept).

Keep initial categorical variables: (default: false) keep the initial categorical variables before preprocessing.

Keep initial numerical variables: (default: false) keep the initial numerical variables before preprocessing.

Categorical recoding method: (default: part Id)

- part Id: identifier of the part (interval, group of values or cell in case of bivariate recoded variable)

- part label: comprehensible label of the part, like in reports

- 0-1 binarization: binarization of the part (generates as many Boolean variables as number of parts)

- conditional info: negative log of the conditional probability of the source variable given the target variable (-log(p(X|Y)). Potentially a good representation for distance based classifiers, such as k-nearest neighbours or support vector machines

- none: do not recode the variable

Numerical recoding method: (default: part id)

- part Id

- part label

- 0-1 binarization

- conditional info

- center-reduction: (X – Mean(X)) / StdDev(X)

- 0-1 normalization: (X – Min(X)) / (Max(X) – Min(X))

- rank normalization: mean normalized rank (rank between 0 and 1) of the instances

- none

Pairs recoding method: (default: part Id)

- part Id

- part label

- 0-1 binarization

- conditional info

- none

2.3.3. Preprocessing
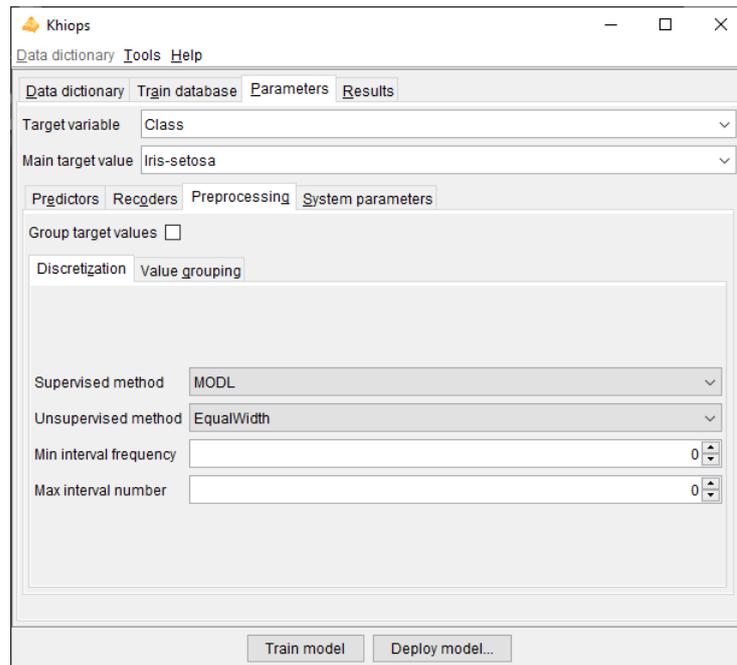
Group target values: in case of classification task, indicates that the preprocessing methods should consider building discretization by partitioning both the input values (in intervals or groups of values) and the target values into groups. This is potentially useful in case of classification tasks with numerous target values, by automatically and optimally reducing the number of target values using groups.

2.3.3.1. Discretization

Supervised method: name of the discretization method in case of classification or regression (default: MODL).

Unsupervised method: name of the discretization method in case of unsupervised analysis (default: EqualFrequency).

Min interval frequency: (default: 0, automatically set). Min number of instances in each interval. When this user constraint is active, it has priority over the criterion of the discretization method.

Max interval number: (default: 0, automatically set). Max number of intervals produced by the discretization. When this user constraint is active, it has priority over the criterion of the discretization method. By default (value 0), the MODL methods chose the optimal interval number automatically, whereas the unsupervised methods (EqualWidth and EqualFrequency) build discretizations with at most 10 intervals.

The user constraints help improving the comprehensibility of the discretization intervals, but may decrease their statistical quality.

Name	Min freq.	Max nb.	Comment
MODL	✓	✓	Bayes optimal discretization method: the criterion allows to find the most probable discretization given the data. Bottom-up greedy heuristic, with deep post-optimizations.
EqualFrequency		✓	EqualFrequency discretization method, which builds intervals having the same frequency (by default: 10 intervals)
EqualWidth		✓	EqualWidth discretization method, which builds intervals having the same width (by default: 10 intervals)

All the methods are available for classification tasks. For regression tasks, the only available discretization method is MODL: it discretizes both the input variable and the target variable. In unsupervised tasks, the only available methods are EqualFrequency and EqualWith.

Note: Missing values are treated as a special value (minus infinity) which is smaller than any actual value. If missing values are both informative and numerous, the discretization algorithm builds a special interval containing all missing values. Otherwise, missing values are included in the first built interval, containing the smallest actual values.

2.3.3.2. Value grouping

Supervised method: name of the value grouping method in case of classification or regression (default: MODL).

Unsupervised method: name of the value grouping method in case of unsupervised analysis (default: BasicGrouping).

Min group frequency: (default: 0, automatically set). When this user constraint is active, all the explanatory values with frequency below the threshold are unconditionally grouped in a "garbage" group.

Max group number: (default: 0, automatically set). When this constraint is active, the value grouping method does not stop the grouping until the required number of group is reached. By default (value 0), the MODL methods choose the optimal group number automatically, whereas the unsupervised method (BasicGrouping) build at most 10 groups.

The user constraints help improving the comprehensibility of the value groups, but may decrease their statistical quality.

Name	Min freq.	Max nb.	Comment
MODL	✓	✓	Bayes optimal value grouping method: the criterion allows to find the most probable value grouping given the data. A "garbage" group is used to unconditionally group the infrequent values (the frequency threshold is automatically adjusted). Bottom-up greedy heuristic, with deep post-optimizations.
BasicGrouping	✓	✓	Basic value grouping method that builds one group for each frequent explanatory values. The infrequent values (below the frequency threshold) are unconditionally grouped in a "garbage" group. The remaining infrequent values are also grouped until the required number of groups is reached. When no user constraint is specified, the min frequency is set to 2 and the max number of groups is set to 10. Remark: when the max number of groups is set to 2, the value grouping method builds two groups, the first one containing the mode (most frequent value) and the second one containing all the other values. Remark: using the min frequency set to 1 and max group number set to a large number (e.g. 10 000) is a way of collecting the statistics on all the values of a variable.

All the methods are available for classification tasks. For regression tasks, the only available value grouping method is MODL: it groups the values of the input variable and discretizes the target variable. In unsupervised tasks, the only available method is BasicGrouping.

2.3.4. System parameters

Max number of items in reports: allows to control the size of reports, by limiting the number of reported items, such as the number of lines or rows in contingency tables, the number of detailed groups of values or the number of values in each detailed group (default: 1 000 000).

Max number of error messages in log: allows to control the size of the log, by limiting the number of messages, warning or errors (default: 20).

Memory limit in MB: allows to specify the max amount of memory available for the data analysis algorithms. By default, this parameter is set to the limit of the available RAM. This parameter can be decreased in order to keep memory for the other applications.

Max number of processor cores: allows to specify the max number of processor cores to use.

Temp file directory: name of the directory to use for temporary files (default: none, the system default temp file directory is then used).

The resources fields related to memory, processor cores and temp file directory allow the user to upper-bound the system resources used by Khiops. Given this, Khiops automatically manages the available system resources to perform at best the data mining tasks.

The following tasks of Khiops benefit from a parallel implementation on multi-core machines:

- sort functionality to sort large tables,

- extraction of an identifier table from a sorted log table,

- check of database,

- deployment for single or multi-tables schemas,

- univariate preprocessing: discretization and value grouping,

- bivariate preprocessing,

- learning of a Selective Naïve Bayes predictor

- evaluation of a predictor.

The learning of trees will be parallelized in a future version of the tool.

2.4. Results
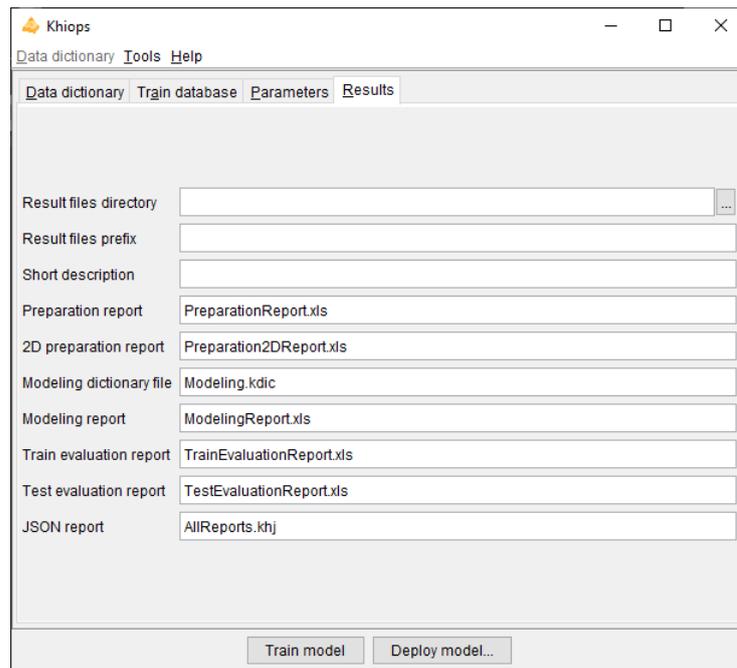
Result files directory: name of the directory where the results files are stored (default: empty). By default, the results files are stored in the directory of the train database. If a result directory is specified, it can be:

- an absolute path (example "c:\project\scenario1"): the results files are stored in this directory

- a local path (example "scenario1"): the results files are stored in a sub-directory of the train database directory

- a relative path (example ".\scenario1"): the results files are stored in a sub-directory of current directory (Khiops executable start directory)

Result files prefix: (default: empty). This prefix is added before the name of each result file.

Short description: (default: empty). Brief description to summarize the current analysis, which will be included in the reports.

The following result file names allow to specify the name of each report or model resulting from an analysis. When a file name is missing, the corresponding report is not produced.

Preparation report: name of the data report file produced after the univariate data analysis on the train database (default: PreparationReport.xls).

2D preparation report: name of the report file produced after the bivariate data analysis on the train database (default: Preparation2DReport.xls).

Modeling dictionary file: name of the dictionary file that contains the trained predictors (default: Modeling.kdic). This dictionary file can then be used to deploy the predictors on new data.

Modeling report: name of the report file produced once the predictors have been trained (default: ModelingReport.xls).

Train evaluation report: name of the report file produced after the evaluation of the predictors on the train database (default: TrainEvaluationReport.xls).

Test evaluation report: name of the report file produced after the evaluation of the predictors on the test database (default: TestEvaluationReport.xls).

JSON report: name of the JSON file that contains the results of all the reports (default: AllReports.khj). The JSON file is useful to inspect the modeling results from any external tool. For example, the khiops-python library enables to exploit any Khiops modeling results from python.

2.5. Tools menu
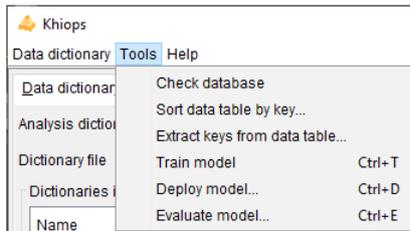
2.5.1. Check database

Prerequisite: the train database must be specified, and the dictionary related to the train database must be loaded in memory.

This action checks the integrity of the train database.

This action reads each line of the train database to perform the integrity checks. During formatting checks, the number of fields in the train database is compared to the number of native variables in the dictionary. Data conversion checks are performed for the fields corresponding to numerical, date, time and timestamp variables. Error messages are displayed in the message log window.

Remark: errors during database checking are always displayed, but they are autocorrected (empty or erroneous numerical values are replaced by a system missing value, superfluous values are discarded). Therefore, a data analysis can always be performed, even though it might not be reliable in case of database errors.

In case of multi-tables database, data tables must be sorted by key. Sort errors are reported but cannot be corrected, and therefore, no data analysis can be performed.

2.5.2. Sort data table by key

Prerequisite: the dictionary of the input data table must be loaded in memory.
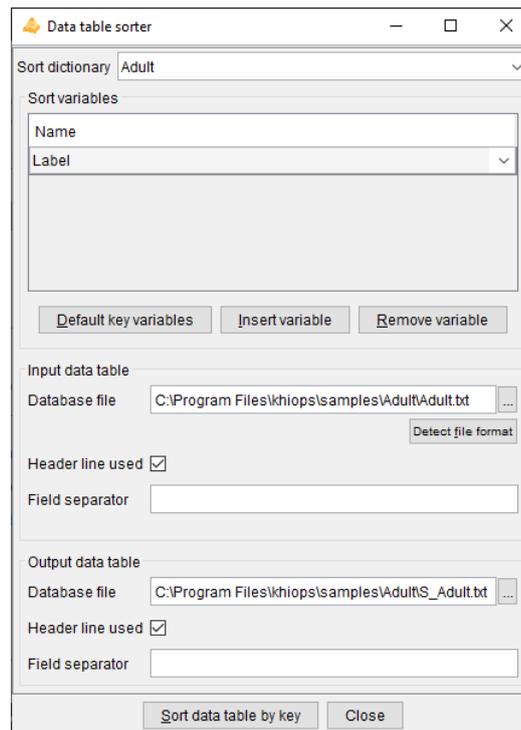
This action allows to sort a data table according to sort variables. It is dedicated to the preparation of multi-table databases, which requires all data table files to be sorted by key, for efficiency reasons.

The parameters of the dialog box are the following.

Sort dictionary: dictionary that describes all native of the database file. Native variables are the variables stored in date files, of type Numerical, Categorical, Date, Time or Timestamp, and not derived using a formula.

Sort variables: must be native (not derived) and Categorical

- **Default key variables**: the sort variables are the key variables retrieved from the sort dictionary

- **Insert variable**: inserts a variable in the list of sort variables

- **Remove variable**: removes a variable from the list of sort variables

Input data table:

- Database file

    o Detect file format

- Header line used

- Field separator

Output database:

- Database file

- Header line used

- Field separator

The "**Sort data table by key**" action reads the input data, sorts the lines by key, and writes the sorted output data.

All native variables (either used or not in the sort dictionary) are written in the output database, whereas derived variables are ignored: the output database has the same content as the input database, except that the lines are now sorted by key.

Note that this feature is very low-level and performs only minimal checks. For example, even the header line can be invalid w.r.t. all native variables defined in the dictionary, provided that the mandatory key fields for sorting are correct.

2.5.3. Extract keys from data table

Prerequisite: the dictionary of the input data table must be loaded in memory and the input data table must be sorted by the keys of its dictionary.
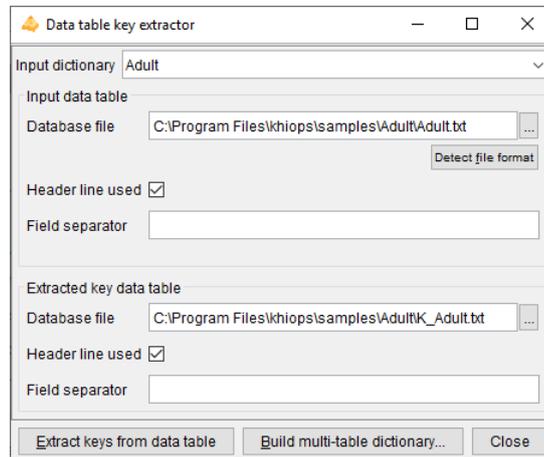
This action allows to extract keys from a sorted input data table. It is dedicated to the preparation of multi-table databases, where a root entity has to be extracted from a detailed 0-n entity. For example, in case of a web log file with cookies, page, timestamp in each log, extracting keys allow to build a table with unique cookies from the table of logs.

The parameters of the dialog box are the following.

Input dictionary: dictionary that describes the content of the input data table.

Input data table:

- Database file
  - o Detect file format
- Header line used
- Field separator

Extracted key data table:

- Database file
- Header line used
- Field separator

The "**Extract keys from data table**" action reads the input data, remove duplicate keys and store the unique resulting keys in the output data table.

The "**Build multi-table dictionary**" action builds a root dictionary with a Table variable based on the input dictionary, then saves the dictionary file.

2.5.4. Train model

Prerequisite: the train database must be specified, and the dictionary related to the train database must be loaded in memory.

If the name of the class variable is missing, the data analysis is restricted to unsupervised descriptive statistics. Otherwise, the learning task is classification or regression according to the type of the target variable (categorical or numerical).

The analysis starts by loading the database into memory. Data chunks are potentially used in order to be consistent with the available RAM memory. Khiops then performs discretizations for numerical variables, value groupings for categorical variables. Variables are also constructed according to whether the feature engineering options are activated. Finally, the requested predictors are trained and evaluated.

A data preparation report describing the univariate statistics (discretizations and value groupings) is then produced, as well as new dictionaries related to data preparation or to the built predictors. The built dictionaries (available according to the activated options) are:

- R_<Dic>: dictionary containing the recoding variables (discretizations and value groupings),
- SNB_<Dic>: dictionary containing the prediction and prediction score formulae for a Selective Naive Bayes predictor,
- BU_<Dic>_<variable>: dictionary containing the prediction and prediction score formulae for the kth univariate predictor, corresponding to variable <variable>.

A modeling report summarizes the features of the built predictors.

An evaluation report is also produced on the train and test datasets.

The predictor dictionaries

At the end of the data analysis, Khiops builds predictors and saves them by means of dictionaries including variables dedicated to prediction. The formulae used to compute prediction variables are stored in the dictionaries, enabling the deployment of prediction scores on unseen data. The data miner can select or unselect variables to deploy using the "Unused" keyword in the modeling dictionary. For example, to produce a score file, the data mining can select a key variable, in order to enable joins in databases, and the variable related to the probability of the class value of interest.

The main output variables in a classification dictionary with a target variable named <class> are:

- Predicted<class>: predicted value
- Prob<class><value>: probability of each target value named <value>

The main output variables in a regression dictionary with a target variable named <target> are:

- M<target>: predicted mean of the target
- SD<target>: predicted standard deviation of the target

Other interesting variables are related to the prediction of the normalized rank of the target value (rank between 0 and 1):

- MR<target>: predicted mean of the target rank
- SDR<target>: predicted standard deviation of the target rank
- CPR<i><target>: cumulative probability of the target rank, i.e. probability that the target rank is below i

It is noteworthy that in case of regression, Khiops is able to predict the full conditional distribution of the target values. For example, the regression variables TargetRankCumulativeProbAt(rank) available in the regression dictionary enable to predict the conditional probability of the target variable for any interval of values (more precisely interval of normalized target ranks, i.e. target partile).

In case of multi-table database, automatic variable construction allows to explore complex representation spaces from multiple tables by creating many variables in the root entity to summarize the content of the sub-entities.

2.5.5. Deploy model

Prerequisite: at least one dictionary must be loaded in memory.

This action opens a dialog box allowing to specify an input and output database, and a dictionary describing the variables to keep, discard or derive. This allows to recode a database or to deploy a predictor on new data. The parameters of the dialog box are the following.

Deployment dictionary: dictionary used to select or derive new variables.

Input database:

- Data table file
  - Detect file format
- Header line used
- Field separator
- Sample percentage
- Discard mode
- Selection variable
- Selection value

Output database:

- Data table file
- Header line used
- Field separator
- Output format: tabular (default): standard tabular format; sparse: extended tabular format, with sparse fields in case of blocks of variables

The "**Deploy model**" action reads the input data, applies the deployment dictionary to select all or part of the variables and add derived variables, and writes the output data.

This action can be used to generate a data preparation file, containing the recoded variables. It also can be used to deploy a scoring model, owing to the prediction variables contained in the predictor dictionaries.

The "**Build deployment dictionary**" action creates an output dictionary that enables to read and analyses the output files: it contains the deployed variables only, without any derivation rule in the dictionary.

For multi-table databases, there are potentially several lines in the array of the input and output database files. One input data table file must be specified for each entity in the multi-table dictionary. For the output database, all or some of the data table file can be specified: the deployment writes only the output data table files that are specified (and not "Unused" in the multi-table dictionary).

2.5.6. Evaluate model

Prerequisite: at least one dictionary must be loaded in memory, and correspond to a predictor dictionary.
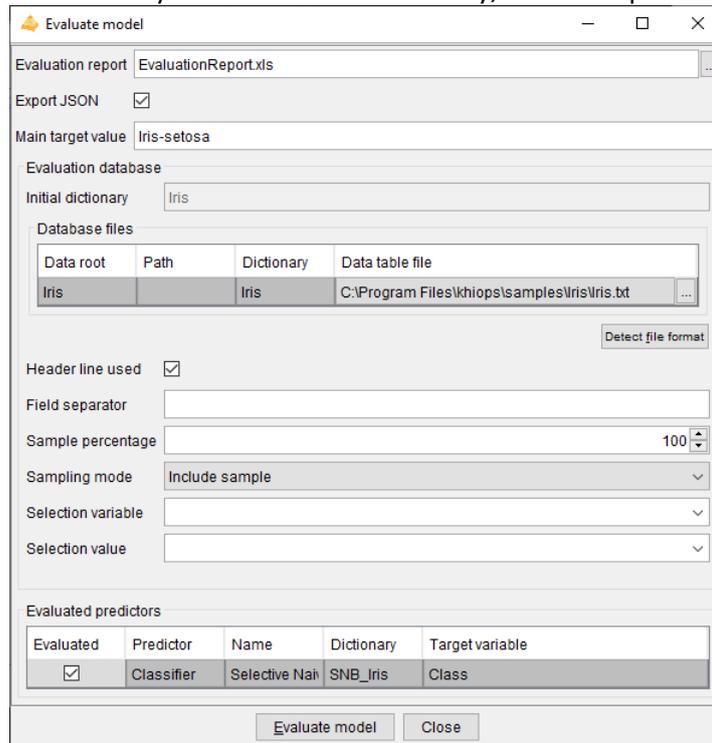
This action opens a dialog box allowing to specify an evaluation report, an evaluation database and to choose the predictor(s) to evaluate. The parameters of the dialog box are the following.

Evaluation report: name of the evaluation report file

Export JSON: exports the evaluation report under a JSON format. The exported JSON file has the same name as the evaluation report file, with a .khj extension. The JSON file is useful to inspect the evaluation results from any external tool.

Main target modality: value of the target variable in case of classification, for the lift curves in the evaluation reports.

Evaluation database:

- Data table file
    - Detect file format
- Header line used
- Field separator

- Sample percentage

- Discard mode

- Selection variable

- Selection value

Evaluated predictors: List of the predictor dictionaries, which are dictionaries among the loaded dictionaries that are recognized as predictor dictionaries. This array allows to choose (parameter "Evaluated" which predictor to evaluate).

- Evaluated: to choose whether to evaluate the predictor

- Predictor: Classifier or Regressor

- Name: label of the predictor

- Dictionary: name of the predictor dictionary

- Target variable: name of the target variable (for classification or regression)

The "**Evaluate model**" applies the evaluated predictors on the evaluation database and writes an evaluation report. Whereas the "Train model " action trains predictors and evaluates them immediately on the train and test databases, the "Evaluate model" action allows a differed evaluation of previously trained predictors.

For multi-table databases, there are potentially several lines in the array of the evaluation database files.

2.6. Help menu
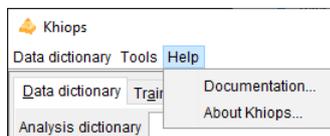
The actions available from the help menu are
- **Documentation…**
  Shows a summary of the available documentation and other resources.
- **About Khiops…**

3. Inputs-outputs

3.1. Format of the database files

A database file is a text file, containing one line per record. By default, the first line contains the names of the variables. If no header line is used, the fields in the database file must appear in the same order as the variables in the related dictionary.

The values of the variables are separated by a field separator. The field separator is tabulation by default (empty), and can be space (S), semi-colon (;), comma (,) or any character.

Fields can contain separator chars provided that they are surrounded by double-quotes:

- any field can be surrounded by surrounded by double-quotes (e.g. *"city"* for *city*),

- for fields which content is surrounded by double-quotes:

- the separator char can be used inside the field (e.g. *"NY, city" for NY, city*)

- the double-quote can be used if it is paired (e.g. *"""NY"", city"* for *"NY", city*),

- the end of line character cannot be used inside a field (multiple-line fields are not allowed).

The numerical values may use the scientific notation (for example: 1.3E7). The decimal separator can be either the dot or the comma (the commas are recoded into dots). Missing or erroneous numerical values are replaced by a missing system value (–infinity, to avoid collision with any valid value).

Tabulations inside categorical values are replaced by blank characters, since they raise problem in visualisation tools that are based on text files with tabulation separated fields. The special char Ctrl-Z (ascii 26) is also replaced by a blank character. Space characters are discarded at the beginning and end of categorical values.

Date values are stored using the *YYYY-MM-DD* format, Time values using the *HH:MM:SS.* format and Timestamp values using the *YYYY-MM-DD HH:MM:SS.* format. Numerous other formats are available (see appendix). For these formats, the variable must be declared with a meta-data (with key *DateFormat*, *TimeFormat* or *TimestampFormat*) to specify the external format (see Paragraph 3.2.2. Dictionary).

Note that Khiops also exploits an extended tabular format with sparse fields. This advanced feature, used internally by Khiops for the management of sparse data, is detailed in section 8. Appendix: variable blocks and sparse data management.

For multi-table databases, database files must be sorted by key for efficiency reasons.

3.2. Format of the dictionary files

3.2.1. Dictionary file

A dictionary file is a text file with extension .kdic, containing the definition of one or many dictionaries.

3.2.2. Dictionary

A dictionary allows to define the name and type of native variables in a data table file, as well as the constructed variables described by means of derivation rules.

> *Dictionary name [meta-data] [// label]*
> *{*
> * {Variable definition}**
> *};*

A dictionary is defined by its name and by the list of its variables, and optionally meta-data and a label.

Variables within a dictionary can be organized into *variables blocks*. This advanced feature, used internally by Khiops for the management of sparse data, is detailed in section 8. Appendix: variable blocks and sparse data management.

Meta-data is a list of keys or key value pairs (*<key>* or *<key=value>* for numerical or categorical constant values). Meta-data is used internally by Khiops to store information related to dictionaries or variables (to annotate the results of analysis). It is also used to store the external format of Date, Time and Timestamp variables, in case where the default format is not used. In the following example, the three predefined meta-data keys *DateFormat*, *TimeFormat, TimestampFormat* and *TimestampTZFormat* are used to specify the input and output format of the related variables:

```
Date MyDate      ; <DateFormat="DD/MM/YYYY">
Time MyTime      ; <TimeFormat="HH.MM">
Timestamp     MyTimestamp        ; <TimestampFormat="YYYY-MM-DD_HH:MM:SS">
TimestampTZ   MyTimestampTZ      ; <TimestampTZFormat="YYYY-MM-DD_HH:MM:SS.zzzzz">
```

Each variable is defined by its type, its name and other optional information.

[Unused] type name [= derivation rule]; [meta-data] [// label]

A variable can be ignored in the data processing (memory loading, modeling, deployment) if the keyword *Unused* is specified before the variable definition. Even though, Khiops is still aware of the variable, which allows to construct new variables derived from the ignored variable.

The types are Categorical, Numerical, Date, Time or Timestamp for native variables.

The names are case sensitive and are limited to 128 characters. In the case where they use characters other than alphanumeric characters, they must be surrounded by back-quotes. Tabulations are not allowed inside variables names (replace by blank characters). Back-quotes inside variable names must be doubled.

Derivation rules are formulas that allow to compute the value of variable from other values coming from other variables, rules, or constants.

Each line in the definition of a dictionary can be commented, using "//" as a prefix.

Some technical types are used by Khiops to specify prepocessing or modeling methods: for example Structure(DataGrid), Structure(Classifier).

Example: dictionary file Iris.kdic with a constructed variable PetalArea

```
Dictionary Iris
{
   Unused   Numerical          SepalLength;
            Numerical          SepalWidth;
            Numerical          PetalLength;
            Numerical          PetalWidth;
            Numerical          PetalArea   = Product(PetalLength, PetalWidth);
            Categorical        Class;             // Class variable
};
```

3.2.3. Multi-table dictionary

Whereas most data mining tools work on instances*variables flat tables, real data often have a structure coming from databases. Khiops allows to analyse multi-table databases, where the data come from several tables, with zero to one or zero to many relation between the tables.

To analyse multi-table databases, Khiops relies on:

- an extension of the dictionaries, to describe multi-tables schemas, (this section)
- databases that are stored in one data file per table in a multi-table schema (cf. .section 2.2. Train database),
- automatic feature construction to build a flat analysis table(cf. Section 2.3.1.4. Variable construction parameters).

In this section, we present star schema, snowflake schemas, external tables, then give a summary.

3.2.3.1. Star schema

For each dictionary, one or several key fields have to be specified in the first line of the dictionary definition, using parenthesis (e.g. *Dictionary Customer (id_customer)*). In case of multiple key fields, they must be separated by commas (e.g. *Dictionary Customer (id_country, id_customer)*). The key fields must be chosen among the Categorical variables and must not be derived from a rule.

One of the dictionaries has to be chosen as the Root dictionary, which represents the entities to analyze, using the Root flag (e.g. *Root Dictionary Customer (id_customer)*).

The relation between the dictionaries has to be specified by creating new Entity or Table relational variables

- e.g. in *Dictionary Customer*, an *Entity(Address) Address* variable for a 0-1 relationship between a customer and its address (where *Address* is the dictionary of the sub-entity).

- e.g. in *Dictionary Customer*, a *Table(Usage) Usages* variable for a 0-n relationship between a customer and its usages (where *Usage* is the dictionary of the sub-entity).

The keys in the dictionaries of the sub-entities must have at least the same number of fields as in the root dictionary, but these key fields do not need to have the same names.

There must be one table file per table use in the schema. All tables must be sorted by key, and as for the root table, each record must have a unique key.

```
         Customer                  0  Usages
 id_customer  Categorical          0  Address                    Usage
 Name:        Categorical                          N   id_customer  Categorical
                                                       Product:     Categorical
                                                       Time         Timestamp
                                                       Duration     Numerical


                                                              Address
                                                   1   id_customer
                                                       StreetNumber  Numerical
                                                       StreeName     Categorical
                                                       City          Categorical
```
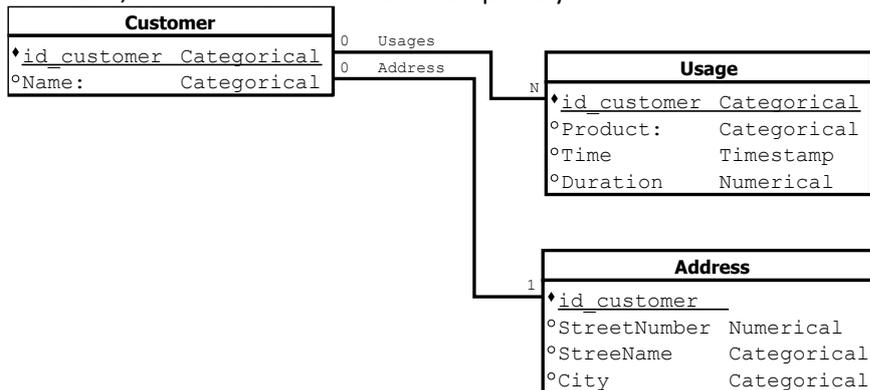
Example: dictionary file Customer.kdic with a root dictionary *Customer* and a 0-1 relation with address and a 0-n relation with sales. A multi-table database related to this multi-table dictionary consists of three data table files, sorted by their key fields.

```
Root Dictionary Customer (id_customer)
{
    Categorical         id_customer;
    Categorical         Name;
    Entity(Address)     Address;    // 0-1 relationship
    Table(Usage)        Usages;              // 0-n relationship
};


Dictionary Address (id_customer)
{
    Categorical         id_customer;
    Numerical           StreetNumber;
    Categorical         StreetName;
    Categorical         City;
};


Dictionary Usage (id_customer)
{
    Categorical         id_customer;
    Categorical         Product
    Timestamp           Time;
    Numerical           Duration
};
```

3.2.3.2. Snowflake schema

The example in the preceding section illustrates the case of a star schema, with the customer in a root table and its address and sales in secondary tables. Secondary tables can themselves be in relation to sub-entities, leading to a snowflake schema. In this case, the number of key fields must increase with the depth of the schema (but not necessarily at the last depth).
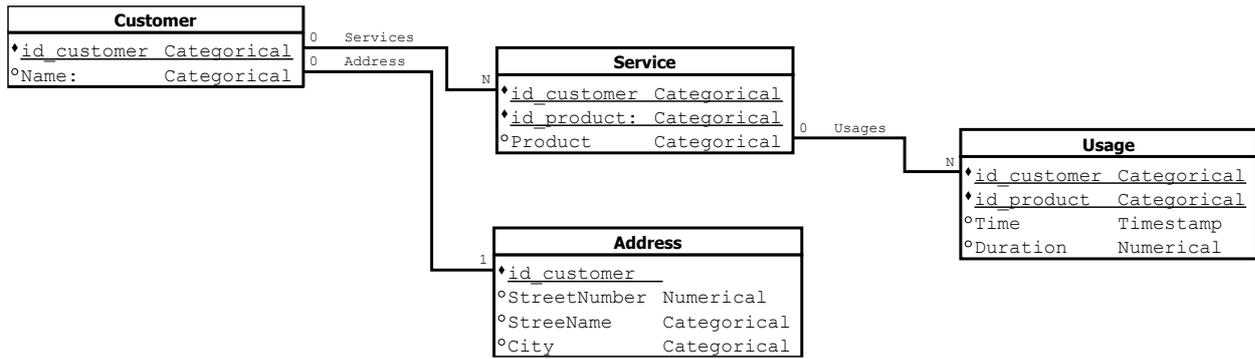
3.2.3.3. External tables

External tables can also be used, to reuse common tables that are shared by all the analysis entities. In the following schema, the products can be referenced from the services of a customer.
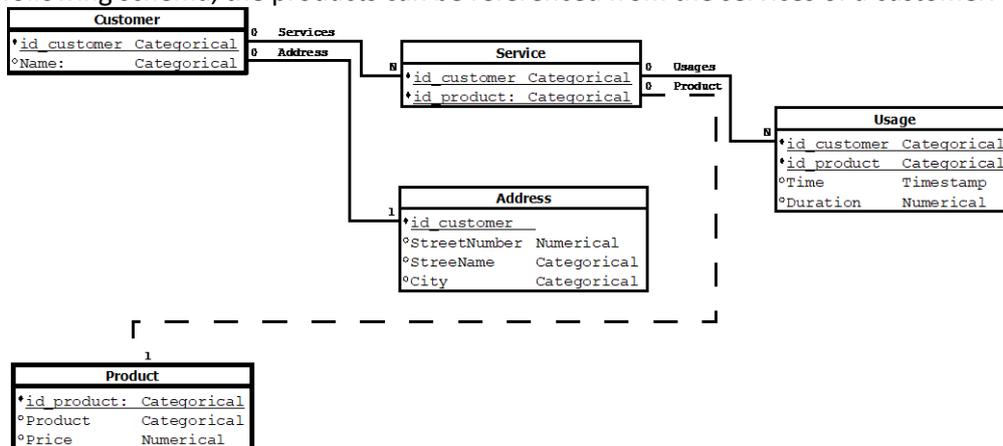
Whereas the sub-entities of root entity Customer are all ***included*** in the customer ***folder*** (the address, services and usages per service belong to the folder), the products are ***referenced*** by the services.

The dictionary of an external table must be a root dictionary, with a unique key.

The related table file will be fully loaded in memory for efficient direct access, whereas the entities of each folder can be loaded one at a time, for scalability reasons.

Whereas the joins between the tables of the same folder are implicit, on the basis of the table keys, the join with an external table must be explicit in the dictionary, using a key (into brackets) from the referencing entity.

```
Dictionary          Service      (id_customer, id_product)
{
        Categorical          id_customer                    ;
        Categorical          id_product           ;
        Entity(Product)      Product    [id_product]        ;
        Table(Usage)         Usages               ;
};

Root     Dictionary Product    (id_product)
{
        Categorical          id_product           ;
        Categorical          Name                 ;
        Numerical Price                   ;
};
```

Examples of datasets with multi-table schemas and external tables, are given in the "samples" directory of the Khiops package (%PUBLIC%\khiops_data\samples in windows, $HOME/khiops_data/samples in Linux) .

3.2.3.4. Summary

Khiops allow to analyse multi-table databases, from standard mono-table to complex schema.

Mono-table	

• standard representation
 	
Fields types
  • Numerical
  • Categorical
  • Date
  • Time
  • Timestamps	

Star schema
  • Multi-table extension
  • Each table must have a key
  • The root table is the main entity
 	
Additional field types in the root table
  • Entity: 0-1 relation ship
  • Table: 0-n relationship	
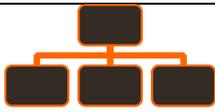
Snowflake schema
  • Extended star schema
  • Each table must have a key
  • The root table is the main entity
 	
Additional field types in any table of the schema
  • Entity: 0-1 relation ship
  • Table: 0-n relationship	
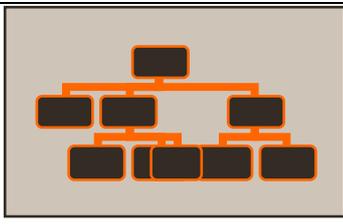
External tables
  • External tables allow to reuse common tables referenced by all entities
  • Must be root tables
  • Must be referenced explicitly, using keys from the referencing entities	
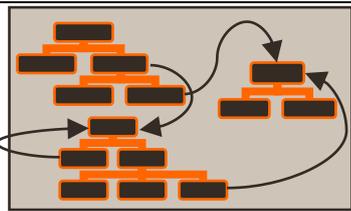
3.2.4. Edition of dictionary files by means of Excel

The dictionary files, which are text files with tabulation separators, could be easily edited using Excel. Unfortunately, the use of derivation rules or categorical constants (surrounded by double quotes) is error prone in Excel (due to automatic data conversion in Excel). However, Excel can be used safely with the following process:

- Open the dictionary file using a basic text editor (for example: Notepad),

- Copy-past all or part of the defined variables to Excel,

- Edit the variables in Excel (select, modify, sort…),

- Copy-paste the edited variables back to the text editor.

Editing the variables using Excel allows to display the variables properties (Unused keyword, User type, Type, Name, Derivation rule, Comment, Level…) in Excel columns. This is then easy to perform sorts and modify the definition of variables.

3.2.5. Derivation rules

The derivation rules allow to construct new variables in a dictionary. The operands in a derivation rule can be a variable (specified by its name), a constant numerical or categorical value, or the result of another derivation rule. The derivation rules can be used recursively.

A constant categorical value must be surrounded by double quotes. A double quote character inside a categorical value must be doubled. When the length of a categorical value is too important, the value can be split into sub-values, concatenated using '+' characters.

A constant numerical value can be specified using scientific notation (for example: 1.3E7). The decimal separator is the dot. The missing value is represented as #Missing when used in a derivation rule.

There are no Date, Time Timestamp constants, but they can be produced using conversion rules (see appendix: e.g. *AsDate("2014-01-15", "YYYY-MM-DD")*);

The list of available derivation rules is given in appendix.

3.2.5.1. Derivation rules for multi-table schemas

Derivation rules can be used to extract information from other tables in a multi-table schema. In this case, they use variables of different scopes:

- First operand of type Entity or Table, in the current dictionary scope (ex: DNA),

- Next operands, in the scope of the secondary table (ex: Pos, Char).

In the following example, the "MeanPos" and "MostFrequentChar" extract information from a DNA sequence in the secondary table. The derivation rules (TableMean and TableMode) have a first operand that is a Table variable in the scope of SpliceJunction, while their second operand is in the scope of SpliceJunctionDNA.

```
Root Dictionary SpliceJunction(SampleId)
{
  Categorical          SampleId;
  Categorical          Class;
  Table(SpliceJunctionDNA)      DNA;
  Numerical            MeanPos = TableMean(DNA, Pos);          // Mean position in the DNA sequence
  Categorical          MostFrequentChar = TableMode(DNA, Char); // Most frequent char in the DNA sequence
  };

Dictionary SpliceJunctionDNA(SampleId)
{
  Categorical          SampleId;
  Numerical            Pos;
  Categorical          Char;
};
```

3.2.5.2. Derivation rules with multiple scope operands

For operands in the scope of a secondary table, it is possible to use variables from the scope of the current dictionary, which is in the "upper" scope of the secondary table. In this case, the scope operator "." must be used.

In the following example, the "FrequentDNA" selects the record of the "DNA" table, where the Char variable (in secondary table) is equal to the "MostFrequentChar" variable (with the scope operator ".", as it in the scope of the current dictionary. And the "MostFrequentCharFrequency" computes the frequency of this selected sub-table.

```
Root Dictionary SpliceJunction(SampleId)
{
  Categorical          SampleId;
  Categorical          Class;
  Table(SpliceJunctionDNA)      DNA;
  Categorical          MostFrequentChar = TableMode(DNA, Char);
  Table(SpliceJunctionDNA)      FrequentDNA = TableSelection(DNA, EQc(Char, .MostFrequentChar));
  Numerical MostFrequentCharFrequency = TableCount(FrequentDNA);
  };
```

Note that the resulting "MostFrequentCharFrequency" could be computed using one single formula:

<p style="color:red">*Numerical MostFrequentCharFrequency = TableCount(*</p>
<p style="color:red">*TableSelection(DNA,*</p>
<p style="color:red">*EQc(Char,*</p>
<p style="color:red">*.TableMode(DNA, Char))));*</p>

3.3. Reports

Depending on the type of analysis, Khiops can produce 4 reports whose default names are :
- Preparation reports : one report or two reports in case of pairs of variables
- Modeling report : available if predictors have been trained
- Evaluation reports : available if a train or test database has been specified and if predictors have been trained.

The reports can be consulted in three ways :
- Using the Khiops Vizualisation tool to open the AllReports.khj file
- Using a text editor to directly open the tabulated text of the reports with extension .xls
- Using the khiops-python Python API

3.3.1. Preparation report

The preparation report is a tabulated text file with file extension .xls, directly editable using Excel, Word, Notepad...

It contains in a header a recall of the main parameters of the data analysis problem, general statistics related to the train database, and a synthetic array of univariate statistics for the categorical and numerical variables, sorted by decreasing predictive importance.

The remainder of the report consists of contingency tables for the discretized or grouped variables, also sorted by decreasing predictive performance. These detailed statistics are reported for informative variables only.

The fields in the synthetic array of the univariate statistics are defined below.

General evaluation of the variable	
Rank	Rank of the variable, sorted by decreasing importance. This rank is also a convenient identifier, which eases search operations in report files.
Name	Name of the variable
Level	Evaluation of the predictive importance of the variable. The Level is a value between 0 (variable without predictive interest) and 1 (variable with optimal predictive importance).
Groups/Intervals	Number of groups/intervals resulting from the discretization/grouping preprocessing of the variable.
Values	Number of initial values of the variable.
Fields specific to categorical variables	
Mode	Most frequent initial value.
Mode coverage	Coverage of the mode.
Fields specific to numerical variables	
Min	Min value of the variable.
Max	Max value of the variable.
Mean	Mean value.
Std dev	Standard deviation.
Missing number	Number of missing values

Preparation results on variables (uncommented: internal use)	
Constr.cost	
Prep. cost	
Data cost	
User information on variables (optional fields)	
Derivation rule	Derivation rule used to compute the variable. User defined, or automatically constructed by the variable construction functionality of Khiops.

3.3.2. Bivariate preparation report

This report is available if the field "*Max number of pairs of variables*" is not zero.

It contains in a header a recall of the main parameters of the data analysis problem, general statistics related to the train database, and a synthetic array of bivariate statistics for the pairs of variables, sorted by decreasing predictive importance.

The remainder of the report consists of detailed statistics for each pair of variables, also sorted by decreasing predictive performance. These detailed statistics are reported for informative pairs of variables only.

The fields in the synthetic array of bivariate statistics are defined below.

General evaluation of the variable	
Rank	Rank of the pair of variables, sorted by decreasing importance. This rank is also a convenient identifier, which eases search operations in report files.
Name 1	Name of the first variable in the pair
Name 2	Name of the second variable in the pair
DeltaLevel	Evaluation of the relative predictive importance of the pair of variables. DeltaLevel = Level – Level1 – Level2 The DeltaLevel is strictly positive if and only if the variable pair is informative, that is if it brings more information than both variables taken individually.
Level	Evaluation of the total predictive importance of the pair of variables. The Level is a value between 0 (variable without predictive interest) and 1 (variable with optimal predictive importance).
Level 1	Recalls the univariate predictive importance of the first variable
Level 2	Recalls the univariate predictive importance of the second variable
Variables	Number of active variables in the pair: 0: there is no information in the pair 1: the information in the pair reduces to the information in one variable 2: the two variables are jointly informative
Parts 1	Number of intervals or groups of values in the first variable
Parts 2	Number of intervals or groups of values in the second variable
Cells	Number of non-empty cells in the data grid resulting from the cross-product of the two univariate partitions
Constr.cost	Uncommented: internal use
Prep. cost	Uncommented: internal use
Data cost	Uncommented: internal use

Note: in case of unsupervised analysis, the fields DeltaLevel, Level1 and Level2 are not relevant and thus not reported.

3.3.3. Modeling report

This report is available if predictors have been trained.

It summarizes the list of trained predictors, with their name and number of used variables.

Predictor details are reported as well.

For the Selective Naive Bayes predictor, each selected variable is described using the following criteria

- Level: univariate evaluation

    Evaluation of the predictive importance of the variable taken individually. The Level is a value between 0 (variable without predictive interest) and 1 (variable with optimal predictive importance).

- Weight: multivariate evaluation

    Evaluation of the predictive importance of the variable taken relative to all the variables used by the predictor. Many Selective Naive Bayes predictors are built with different variables subsets, and weighted according their predictive interest. Averaging many predictors by predictors weights amounts to building one single predictor with variable weights. The Weight is a value between 0 (variable used in no predictors) and 1 (variable used in all good predictors). A variable (even with slight predictive interest and thus a small Level) that is used by many good predictors can have a high weight. On the opposite, a variable with a high Level can have a small weight in case of redundant variables: the Weight is shared by the redundant variables.

- Importance: overall evaluation

    The Importance criterion is the geometric mean of the Level and the Weight.

$$Importance = \sqrt{\{Level \times Weight\}}$$

3.3.4. Evaluation report

This report is available if a train or test database has been specified and if predictors have been trained.

<u>Classification</u>

Each classifier is evaluated using the following criteria:

- Accuracy: evaluates the proportion of correct prediction

- Compression: evaluates the predicted target probabilities using a negative log likelihood approach and is normalized (between 0 and 1) using the baseline predictor

- AUC: area under the ROC curve (AUC); evaluates the ordering of the predicted scores per target value. Note that the value calculated is the AUC ROC, not the AUC PR.

In case of multiple class values, the AUC is computed for each class value against all the others, then the results are weighted by the proportions of each class value.

A confusion matrix is reported for each classifier, to compare the predicted values (prefixed by $) and the actual values.

If a main class value is specified, lift curves are built. They plot the fraction of actual correct prediction on the y-axis for each fraction of the dataset on the-x-axis, when the dataset is sorted by decreasing predicted probability of the main target value.

For univariate predictors, the contingency table between the intervals/groups of the preprocessed variable and the class variable is also displayed.

<u>Regression</u>

Each classifier is evaluated using the following criteria:

- RMSE: root mean square error between the predicted value and the actual value

- MAE: mean absolute error

- NLPD: negative log predictive density

- RankRMSE: root mean square error between the predicted value rank and the actual value rank

- RankMAE: mean absolute error in the rank domain

- RankNLPD: negative log predictive density in the rank domain

REC (Regression Error Characteristic) curves are also built, in the rank domain. They plot the error tolerance on the x-axis versus the percentage of points predicted within the tolerance on the y-axis.

3.3.5. Visualization report

The content of the reports detailed in this section is fully available under the JSON format, in the JSON report with file extension .khj (by default: AllReports.khj). This JSON report file is exploited by the Khiops visualization tool to inspect all modeling results and by the khiops-python library to access all modelling results from python scripts.

4. Integration in information systems

4.1. Batch mode

4.1.1. Register and replay a batch scenario

For a recurrent use of Khiops, it is interesting to register a scenario in a command file. Such a command file can easily be modified using a text editor, and then replayed by Khiops in batch mode.

To run Khiops, open a Shell khiops using the shortcut available in the Khiops installation directory.

, then type khiops (-h for batch mode options with ability to record or replay a scenario).

To register a scenario, use the –o option on the command line.

khiops –o scenario._kh

All the actions or modifications of field values in the graphical user interface are registered in the command file scenario._kh.

It is a good practise to create a working directory for each Data Mining project. This directory contains all the files for databases, dictionaries, scenarios and reports.

Remark: by default, Khiops keeps a command file named scenario._kh for the last use of the tool, in the "khiops_data/lastrun" sub-directory in your user directory (%USERPROFILE% in Windows, $HOME in Linux).

The following scenario corresponds to a use of Khiops with the Iris sample. The scenario opens the Iris dictionary and performs a classification analysis of the database. To replay this scenario:

- Copy the scenario file (named iris._kh) in the Iris directory

- Open a Shell Khiops and go to Iris directory

- Start Khiops using –i on the command line

khiops –i iris._kh

Scenario file iris._kh

```
// -> Khiops
ClassManagement.OpenFile       // Open...

// -> Open
ClassFileName C:\Program Files\khiops\Samples\Iris\Iris.kdic // Dictionary file
OK                 // Open
// <- Open

TrainDatabase.DatabaseFiles.List.Key Iris       // List item selection
TrainDatabase.DatabaseFiles.DataTableName C:\Program Files\khiops\Samples\Iris\iris.txt   // Data table file
AnalysisSpec.TargetAttributeName Class   // Target variable
AnalysisResults.ResultFilesDirectory ClassificationAnalysis // Result files directory
ComputeStats            // Train model
Exit               // Close
// <- Khiops

// -> Khiops
OK                 // Close
// <- Khiops
```

Note: In parameters files, comments start by "//". More precisely, any line starting with "//" is a comment, and otherwise, only the last part of the line starting by "//" is a comment. This allows to have parameter values that include the "//" characters, provided that the end of the line is a comment.

Note: The two last "Close" actions must be commented to replay the scenario and remain in the graphical user interface of Khiops (otherwise, all the Khiops session is replayed in batch mode, even the close action that exits from Khiops).

4.1.2. Integration in a program to industrialize a data analysis process

Khiops can easily be called from a program written in any computer language, such as C, C++, Java, or Python, in order to build an industrial data analysis process.

You first have to prepare the input data, dictionary file and datasets, write a scenario file in which you set up all needed parameters, mainly the location of the input and output data files and the data analysis operations to be performed, and finally call khiops (in bin directory) by program using the command line options.

4.1.3. List of command line options

Usage: khiops [OPTIONS]
Examples:
  khiops -e log.txt
  khiops -o scenario.txt
  khiops -i scenario.txt -r less:more -r 70:90

In the first example all the logs are stored in the file log.txt
In the second example, khiops records all user interactions in the file scenario.txt
In the last example, khiops replays all user interactions stored in the
file scenario.txt after having replaced 'less' by 'more' and '70' by '90'

Available options are:
  -e <file>   stores logs in the file
  -b   batch mode, with no GUI
  -i <file>   replays commands stored in the file
  -o <file>   records commands in the file

```
(-r <string>:<string>)...    searchs and replaces in the command file
 -p <file>                    stores last progression messages
 -v                           prints version
 -h                           prints help
```

4.2. Khiops Native Interface

The purpose of Khiops Native Interface (KNI) is to allow a deep integration of Khiops in information systems, by the means of any programming language, using a dynamic link library (DLL).

This relates especially to the problem of model deployment, which otherwise requires the use of input and output data files when using directly the Khiops tool in batch mode.

The Khiops deployment features are thus made public through an API with a DLL.

Therefore, a Khiops model can be deployed directly from any programming language, such as C, C++, Java, Python, Matlab...

This enables model deployment in real time application (e.g. scoring in a marketing context, targeted advertising on the web) without the overhead of temporary data files or launching executables.

The KNI package can be downloaded from the khiops.org web site.

4.3. Khiops Python library

The khiops-python library is available at khiops.org .

It allows to perform any Khiops task (data management, data preparation, modelling, evaluation…) and to access any Khiops analysis result from a Python program.

The core modules inside the khiops-python library are:
- core.api: to perform Khiops and Khiops Coclustering tasks
- core.dictionary: Classes to manipulate Khiops dictionaries (kdic or kdicj files)
- core.analysis_results: Classes to inspect Khiops analysis results (khj reports files)
- core.coclustering_results: Classes to inspect Khiops Coclustering results (khc or khcj files)

4.4. JSON file exports

Khiops can export dictionaries as well as any analysis result in files under the JSON format.

The khiops-python modules contain python classes with a simple access to all Khiops JSON reports. These classes completely exemplify the use of the Khiops JSON reports, and provide a pedagogic example of the JSON report manipulation. The coding style in khiops-python is neither "pythonic" nor optimized but rather generic and verbose so it can be easily translated to other structured programming languages.

The structure of the Khiops JSON reports is self-documented:
- In order to be human-readable, the files are in "beautified" form, that is, with a rather comfortable spacing and indentation.
- Most information is available as key-value pairs, where the keys resemble the labels used in Khiops report files (Khiops text reports with extension ".xls") and in the Khiops visualization tools.

4.5. Technical limitations

4.5.1. Supported platforms

Khiops is available on windows and linux, 64 bits: see the khiops.org for the list of downloads. For other platforms (android…), please contact us.

4.5.2. Executable return code

The Khiops executable returns 1 in case of fatal error, 2 in case of error(s), 0 otherwise.

4.5.3. Numerical precision

Numerical precision:

- in memory, the numerical values are stored on 8 bytes, with an exponent between 10^{-100} and 10^{100}, and a mantissa of 10 significant digits,

- risk of loss of numerical precision when using some derivation rules (for example, adding a very small value and a very large value),

- risk of loss of numerical precision during a database deployment if input values use a mantissa of more than 10 digits.

4.5.4. End of line encoding

Ends of line are encoded using CR (Carriage Return) LF (Line Feed) on Windows, LF on linux. Both formats are supported.

But Mac OS Classic format (now deprecated, since Mac OS X in 2001) that encoded ends of lines using CR (without LF) is not supported.

4.5.5. Character encodings

Management of categorical values:

- in memory, the categorical values are stored using null-terminated strings, which is compatible with ANSI files, UTF-8 files, but not with double-bytes character sets such as Unicode that may contain null bytes insides strings,

- categorical values are kept at most once in memory,

- for example, a variable "address" of average length 100 characters and containing 10,000 different values will requires about 1 MB RAM,

- risk of lacking from RAM in case of categorical variables containing large numbers of different values of large size (for example: managing 100,000 different values of average length 10,000 characters requires about 1 GB RAM).

Graphical Interface Issues:

- some data dependent parameters (e.g. variable names with UTF8 characters) may produce errors when modifying them with the Khiops GUI because of inconsistent encodings between Java and input data. Note that this is not a problem in batch mode.

File names issues:

- File names with non ANSI characters are not supported.

Data sets and categorical values:

- when a categorical value is read from a data set, it is trimmed and its tab characters are replaced by blanks,

- this means for example that "Mr", "<space>Mr" or "Mr<space>", "<space<tab>Mr" are treated as the same value,

- however, non-trimmed values can be created using derivation rules (ex: Concat("Mr", "<space>"): this may be convenient for data transformation, but this results in an inconsistent behaviour in the data preparation and modelling processes, with preprocessing results based on non-trimmed values and deployment based on trimmed values

Data files and BOM:

- some tools add a BOM (byte order mark) at the beginning text files in the case of UTF8, UTF16, UTF32 encodings. BOM are encoded with special characters that are invisible but can cause problems, for example when recognizing field names in a header line. BOM are not accepted as valid formats by Khiops, which raises an error in this case.

Character encoding of JSON files:

- JSON files are used to export analysis results, in .khj files for Khiops, .khcj for Khiops coclustering; dictionary files (.kdic) can also be exported to JSON (.kdicj)

- standard-compliant JSON files require the use of UTF8 characters. However, UTF8 is incompatible with still widely used encodings such as Windows-1252 ("ANSI") and ISO-8859-1. To address this incompatibility, Khiops produces valid UTF8 JSON files with the following rules:

  o The ASCII range (0 to 127) is UTF8 compatible so they are written directly

  o The ANSI-only range (or "extended ASCII", 128 to 255) is encoded using a slightly modified version of the Unicode translation table for Windows-1252 that encodes even undefined ANSI characters. We refer to the resulting character set as ANSI-as-UTF8.

  o Valid UTF8 characters are written directly

- note that if your data or dictionary files have characters in the ANSI-only range but were interpreted with a code page other than Windows-1252 the resulting JSON file will not have the correct locale.

- the Khiops JSON files contain the necessary information to detect and diagnose any character encoding problem :

  o The field "khiops_encoding" at the top level of the JSON file allows to know which character sets are present in it. The following table summarizes the field values

Character set present / "khiops_encoding" value	ASCII	ANSI-as-UTF8 (ANSI origin)	ANSI-as-UTF8 (UTF8 origin)	Other UTF8
ascii	Yes			
ansi	Yes	Yes		
utf8	Yes		Yes	Maybe
mixed_ansi_utf8	Yes	Yes		Yes
colliding_ansi_utf8	Yes	Yes	Yes	Maybe

  o All cases can be unambiguously encoded back to ANSI except for the last, which is rare. In the latter case two extra fields provide additional diagnostic information:

    ▪ "ansi_chars": The list of ANSI-only characters present in the JSON file (as UTF8). They are written with Unicode escape sequences "\uXXXX" using the aforementioned Windows-1252 to Unicode translation.

    ▪ "colliding_utf8_chars": The list of UTF-8 characters that collide with the Unicode representation some characters in "ansi_chars". They are written as normal UTF-8 characters. Note that this field may be absent, if no character from the ANSI-as-UTF8 set with UTF-8 origin collides with an ansi character.

4.5.6. Scalability limits

Let N be the number of instances C the number of class values and K the number of variables. The RAM (in bytes) required for data analysis is about:

- Min memory = N.(50+8.C) + K.(1000+250.C)

- Optimal memory = N.(50+8.C+8.K) + K.(1000+250.C)

Thus, a database containing 100 000 instances, 1000 variables and 2 class values can be processed on a PC with 512 MB RAM. A PC with one GB RAM allows to speed up the processing, by avoiding data chunking.

Scalability tests have been performed on a PC with 2 GB RAM in the case of 2 class values, with databases containing tens of millions of records and few variables, or hundreds of thousands of records and tens of thousands of variables. Scalability tests on database files have been performed up to files with tens of GB.

Computation time for data preparation:

- K.C.N.log(N)

- about 5 to 10 min on a PC 2 Ghz, for a database containing 100 000 records and 1000 variables, or 1000 records and 100 000 variables.

Computation time to train the Selective Naive Bayes predictor:

- K.N.C.log(K.N)

- about 5 to 10 min on a PC 2 Ghz, for a database containing 100 000 records and 1000 variables, or 1000 records and 100 000 variables.

In regression analysis, the limitations are similar to those of classification analysis, with an equivalent of C~SquareRoot(N) class values during data preparation, and up to N class values during modeling. Thus, regression analysis may require much more memory and computation time than classification analysis. A potential trade-off between accuracy and computational efficiency is to preprocess the numerical target variable using an equal frequency method (with for example at most SquareRoot(N) intervals).

Limitation of database deployment:

- no known limitation: the databases are processed one record at a time.

4.5.7. Temporary files

Khiops uses temporary files for various internal tasks and stores them in the environment's temporary directory (usually '\Users\{username}\AppData\Local\Temp' on Windows and '/tmp' on Linux).
Khiops prefix its temporary file names with a tilde ('~') and stores them in a sub-directory prefixed by '~Khiops'.

If Khiops exits successfully, it deletes all temporary files generated in the session. In case the application is forcibly killed or another uncontrollable event occurs (such as a power or disk failure), Khiops might not remove these files. When this happens, the following Khiops sessions will search for the '~~anchor~~' file in old temporary directories, check the expiration date stored there and delete them if this date is before one day. If the undeleted files are too large and the user needs to free the space immediately, he can delete them manually.

In very rare cases, errors in the Khiops execution will corrupt the temporary files and the tool will not work properly. If this is the case, the easiest solution is to exit Khiops and restart it.

4.5.8. Known problems

Memory overflow: in spite of conservative evaluation of required memory, Khiops may crash down with memory overflow. In this case, a "memory overflow" message is present in the tool log file. Asking Khiops to use less memory (see "System parameters") is likely to solve this problem.

Some file with UTF-8 encoding may cause problems in rare and unusual cases. Actually, the encoding of the characters differs between the data files and the user interface of Khiops, written in Java. In this case, use Khiops in batch mode rather than in user interface mode.

The user interface, written in Java, may crash in some rare cases (sometimes in case of Java known bugs like screen savers with some graphic cards). In this case, use Khiops in batch mode rather than in user interface mode.

Khiops can be started many times simultaneously on the same machine. However, it is recommended to use Khiops in batch mode when tens of instances of Khiops are used in the same time, to avoid potential problems caused by too many opened java virtual machines.

Under Windows, Khiops can be started using the "Execute Khiops Script" context menu, by a right-click on Khiops script file ("._kh"). However, this does not work for UNC paths (ex: \\server\filename), since they are not supported by Windows cmd.exe.

5. Quick start

5.1. Running Khiops

Khiops can be opened by clicking on the shortcut from the desktop.

A shortcut is also available from the Windows start menu or from the Khiops installation directory.

5.2. Starting sample: classification

Several samples are delivered in the sample directory of the installation directory.

The Iris sample is a small size sample, convenient for a quick start of Khiops.

It contains the following text file database:

File Iris.txt

SepalLength	SepalWidth	PetalLength	PetalWidth	Class
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3.0	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5.0	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5.0	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
…				

The objective is to predict the class of Iris plants (3 class values: Iris-setosa, Iris-versicolor and Iris-virginica), based on measures of the plants (length and width of the petals and sepals). The database contains 150 instances.

Click on the shortcut from the desktop to start Khiops. The Iris sample can be analysed using the following steps.

Build the dictionary

The first step is to define the structure of the data:

- click on button "Build dictionary from data table..." in the "Dictionary file" pane; Khiops opens a dialog box for building dictionaries:

    - fill the path of the Iris file (path of file *iris.txt*) in the "Data table file" field,

    - click on "Build dictionary from data table",

    - fill the dictionary name (*Iris* for example) in the "Dictionary name" field and click on "OK"; Khiops builds the dictionary

    - click on button "Close"; Khiops opens a dialog box to save the build dictionary in a dictionary file,

    - fill the "Dictionary file" field (by default: *Iris.kdic*) and click on "Save".

Validation and extension of the dictionary

The built dictionary must be carefully checked, especially for the type of the variables. This can be done using a text editor (Notepad for example). In the case of the Iris database, the built dictionary is correct. However, new variables can be constructed using derivation rules, in order to evaluate their predictive importance.

File Iris.kdic

```
Dictionary Iris
{
    Numerical      SepalLength;
    Numerical      SepalWidth;
    Numerical      PetalLength;
    Numerical      PetalWidth;
    Categorical    Class;
};
```

Train model

The learning problem can now be specified:

- specify the train database in "Train database" pane:

    - fill the path of the Iris database in the "Data table file" field,

    - fill 70% in the "Sample percentage" field,

- specify the modeling parameters in the "Parameters" pane;

    - fill *Class* in the "Target variable" field,

- start the data analysis, by clicking on button "Train model":

    - Khiops discretizes the numerical variables and groups the values of categorical variables, then sorts the variables by decrease predictive importance; an analysis report named *PreparartionReport.xls* is produced and stored in the directory of the train dataset,

    - Khiops trains one predictor (Selective Naive Bayes), saved in a dictionary file named *Modeling.kdic* containing a classifier dictionary named SNB_Iris; a modeling report named ModelingReport.xls and two evaluation reports named *TrainEvaluationReportl.xls* and *TestEvaluationReportl.xls.*

    - All the report files are referenced in a visualization report named VisualizationReport.khj, which can be opened by the **Khiops Visualization** tool.

PreparationReport.xls

Descriptive statistics

Problem description
Short description

Dictionary	Iris	
Variables		
	Categorical	1
	Numerical	4
	Total	5

Database	C:\Program Files\khiops\Samples\Iris\iris.txt
Sample percentage	70
Sampling mode	Include sample
Selection variable	
Selection value	
Instances	99

Learning task   Classification analysis

Target variable   Categorical   Class

Target variable stats

Value	Frequency	Coverage
Iris-setosa	31	0.313131
Iris-versicolor	37	0.373737
Iris-virginica	31	0.313131

Evaluated variables	4
Informative variables	4

Max number of constructed variables	0
Max number of trees	0
Max number of variable pairs	0
Discretization	MODL
Value grouping	MODL

Null model

	Const. cost	0.693147
	Prep. Cost	8.52714
	Data cost	103.619

Numerical variables statistics

Rank	Name	Level	Intervals	Values	Min	Max	Mean	Std dev	Missing number
R1	PetalLength	0.644632	3	36	1	6.9	3.8010101	1.712137	0
R2	PetalWidth	0.610094	3	20	0.1	2.5	1.21818182	0.74986378	0
R3	SepalLength	0.290065	3	30	4.3	7.7	5.84848485	0.80658447	0
R4	SepalWidth	0.121532	2	22	2	4.4	3.04242424	0.4422374	0

Variables detailed statistics

Rank   R1
Variable   Numerical   PetalLength

% target values

Interval	Iris-setosa	Iris-versicolor	Iris-virginica	Interest	Frequency	Coverage
]-inf;2.4]	1	0	0	0.374224	31	0.313131
]2.4;4.95]	0	0.947368	0.0526316	0.311045	38	0.383838
]4.95;+inf[	0	0.0333333	0.966667	0.314731	30	0.30303
Total	0.313131	0.373737	0.313131	1	99	1

Rank   R2
Variable   Numerical   PetalWidth
…

Modeling.xls

Modeling report

Dictionary   Iris
Target variable   Class
Database   C:\Program Files\khiops\Samples\Iris\iris.txt
Sample percentage   70
Sampling mode   Include sample
Selection variable
Selection value

Trained predictors

Predictor	Variables
Selective Naive Bayes	2

Trained predictors details

Predictor   Selective Naive Bayes

Selected variables

Prepared name	Name	Level	Weight	MAP
PPetalLength	PetalLength	0.644632	0.523438	0.580883
PPetalWidth	PetalWidth	0.610094	0.476562	0.539211

TestEvaluationReport.xls

Test evaluation report

Dictionary   Iris
Target variable   Categorical   Class
Database   C:\Program Files\khiops\Samples\Iris\iris.txt
Sample percentage   70
Sampling mode   Exclude sample
Selection variable
Selection value
Instances   51

```
Predictors performance

Classifier                    Accuracy      Compression        AUC
Selective Naive Bayes         0.96.784       0.853844        0.987584


-----------------------------------------------------------------------------------


Predictors detailed performance


Classifier                Selective Naive Bayes
Accuracy                  0.96.784
Compression               0.853844
AUC                       0.987584


Confusion matrix
                          Iris-setosa      Iris-versicolor     Iris-virginica
$Iris-setosa                  19                 0                  0
$Iris-versicolor               0                12                  4
$Iris-virginica                0                 1                 15
```

Deploy model

New data can be scored using a predictor, owing to the "deploy model" functionality. For example, if you want to deploy the Selective Naive Bayes prediction and prediction score on the Iris database, first load the deployment dictionary file named *Modeling.kdic* using menu "Dictionary file"-"Open …", then open the "Deploy model" dialog box by clicking on menu "Tools"-"Deploy model", then:

- select the deployment dictionary using the "Deployment dictionary" field (for example: *SNB_Iris*),

- fill the "Data table file" fields in the "Input database" panel (for example: *Iris.txt*),

- fill the "Data table file" fields in the "Output database" panel (for example: *T_Iris.txt*),

- click on button "Deploy model".

All the used variables of the deployed dictionary are computed for each record of the input database and written to the output database.

In the following sample, the file *T_Iris.txt* file corresponds to the deployment of the three used variables coming from the predictor dictionary *SNB_Iris*: *Class*, *PredictedClass* and *ScoreClass*.

Deployed file T_Iris.txt

Class	PredictedClass	ProbClassIris-setosa	ProbClassIris-versicolor	ProbClassIris-virginica
Iris-setosa	Iris-setosa	0.996041869	0.001979091	0.00197904
Iris-setosa	Iris-setosa	0.996041869	0.001979091	0.00197904
Iris-setosa	Iris-setosa	0.996041869	0.001979091	0.00197904
Iris-setosa	Iris-setosa	0.996041869	0.001979091	0.00197904
…				

5.3. Other starting samples

5.3.1. Regression

The classification starting sample can be reused for regression, by choosing for example the PetalLength variable as a target variable.

- specify the modeling parameters in the "Parameters" pane;

        o   fill *PetalLength* in the "Target variable" field,

- start the data analysis, by clicking on button "Train model":

When you train a model, the main difference compared to classification is that a new report is produced for correlation analysis. The *level* of each pair of variables, between 0 and 1, evaluates the correlation, and the detailed part of the report presents the joint discretization of the pairs of variables, in order to make the correlation understandable.

Rank	R1					
Variable	Categorical	Class				
Variable stats						
Group	Size	Value list				
{Iris-versicolor}	1	Iris-versicolor				
{Iris-setosa}	1	Iris-setosa				
{Iris-virginica}	1	Iris-virginica	*			
% target values						
Group	]-inf;2.45]	]2.45;4.85]	]4.85;+inf[	Interest	Frequency	Coverage
{Iris-setosa}	1	0	0	0.366329	31	0.313131
{Iris-virginica}	0	0	1	0.337186	31	0.313131
{Iris-versicolor}	0	0.918919	0.081081	0.296484	37	0.373737
Total	0.313131	0.343434	0.343434	1	99	1
...						
--						
Rank	R3					
Variable	Numerical	SepalLength				
% target values						
Interval	]-inf;3.95]	]3.95;4.55]	]4.55;+inf[	Interest	Frequency	Coverage
]-inf;5.45]	1	0	0	0.478600	31	0.313131
]5.45;5.75]	0.375	0.56250	0.06250	0.114846	16	0.161616
]5.75;+inf[	0	0.21153	0.788462	0.406554	52	0.525253
Total	0.373737	0.20202	0.424242	1	99	1
...						

5.3.2. Correlation analysis

The classification starting sample can be reused for correlation analysis, by leaving thetarget variable empty and asking for the analysis of pairs of variables.

- specify the modeling parameters in the " Parameters" pane;

        o   leave the "Target variable" field empty,

        o   in the "Variable construction"sub-pane, choose the "Max number of pairs of variable" to analyse (for example 100)

- start the data analysis, by clicking on button "Train model":

When you train a model, a new data preparation report is produced. In the extract of the report below, the pairs of variables are presented by decreasing level of correlation, and one example of correlation between two numerical variables (PetalLength and PetalWidth) is reported. Each variable is discretized, and the joint contingency table enlightens the correlation.

PreparationReport2D.xls

Descriptive statistics

Problem description

Dictionary	Iris	
Variables		
	Categorical	1
	Numerical	4
	Total	5

Database	C:\Program Files\khiops\Samples\Iris\iris.txt
Instances	99

Learning task   Unsupervised analysis

Variables pairs statistics

Rank	Name 1	Name 2	Level	Variables	Parts 1	Parts 2	Cells
R01	Class	PetalLength	0.145055	2	3	3	4
R02	Class	PetalWidth	0.142386	2	3	3	5
R03	PetalLength	PetalWidth	0.079284	2	3	3	4
R04	Class	SepalLength	0.056410	2	3	3	7
R05	PetalLength	SepalLength	0.041482	2	3	3	5
R06	PetalWidth	SepalLength	0.035258	2	3	3	8
R07	Class	SepalWidth	0.016106	2	2	2	4
R08	PetalLength	SepalWidth	0.005975	2	2	2	4
R09	PetalWidth	SepalWidth	0.005975	2	2	2	4
R10	SepalLength	SepalWidth	0	0	1	1	1

Variables pairs detailed statistics

(Pairs with two jointly informative variables)

...

Rank	R03
Variables	

Type	Name
Numerical	PetalLength
Numerical	PetalWidth

Variables stats

PetalLength

Interval	Frequency	Coverage
]-inf;2.45]	31	0.313131
]2.45;4.75]	32	0.323232
]4.75;+inf[	36	0.363636

PetalWidth

Interval	Frequency	Coverage
]-inf;0.7]	31	0.313131
]0.7;1.65]	37	0.373737
]1.65;+inf[	31	0.313131

Cell frequencies

	PetalWidth				
PetalLength	]-inf;0.7]	]0.7;1.65]	]1.65;+inf[	Total	Coverage
]-inf;2.45]	31	0	0	31	0.313131

]2.45;4.75]	0	32	0	32	0.323232
]4.75;+inf[	0	5	31	36	0.363636
Total	31	37	31	99	
Coverage	0.313131	0.373737	0.313131		

Cells	4			
Cell Id	PetalLength	PetalWidth	Frequency	Coverage
C5	]2.45;4.75]	]0.7;1.65]	32	0.323232
C1	]-inf;2.45]	]-inf;0.7]	31	0.313131
C9	]4.75;+inf[	]1.65;+inf[	31	0.313131
C6	]4.75;+inf[	]0.7;1.65]	5	0.050505
	Total		99	1

...

6. Appendix: variable construction language

This section introduces a list of derivation rules than can be used to constructed new variables from the initial representation.

6.1. Comparison between numerical values

These rules return boolean values encoded as 0 or 1 numerical values. Note that the missing value is treated as a value that is inferior to any valid value in comparisons.

Numerical EQ (Numerical value1, Numerical value2)

Equality test between two numerical values.

Numerical NEQ(Numerical value1, Numerical value2)

Inequality test between two numerical values.

Numerical G(Numerical value1, Numerical value2)

Greater than test between two numerical values.

Numerical GE(Numerical value1, Numerical value2)

Greater than or equal test between two numerical values.

Numerical L(Numerical value1, Numerical value2)

Less than test between two numerical values.

Numerical LE(Numerical value1, Numerical value2)

Less than or equal test between two numerical values.

6.2. Comparison between categorical values

These rules return boolean values encoded as 0 or 1 numerical values.

Numerical EQc(Categorical value1, Categorical value2)

Equality test between two categorical values.

Numerical NEQc(Categorical value1, Categorical value2)

Inequality test between two categorical values.

Numerical Gc(Categorical value1, Categorical value2)

Greater than test between two categorical values.

Numerical GEc(Categorical value1, Categorical value2)

Greater than or equal test between two categorical values.

Numerical Lc(Categorical value1, Categorical value2)

Less than test between two categorical values.

Numerical LEc(Categorical value1, Categorical value2)

Less than or equal test between two categorical values.

6.3. Logical operators

These rules use boolean operands (numerical values with 0 for false and not 0 for true) and return boolean values encoded as 0 or 1 numerical values.

Numerical And(Numerical boolean1, ...)

And logical operator.

Numerical Or(Numerical boolean1, ...)

Or logical operator.

Numerical Not(Numerical boolean)

Not logical operator.

Numerical If(Numerical test, Numerical valueTrue, Numerical valueFalse)

Ternary operator returning second operand (true) or third operand (false) according to the condition in first operand.

Categorical IfC(Numerical test, Categorical valueTrue, Categorical valueFalse)

Ternary operator returning second operand (true) or third operand (false) according to the condition in first operand.

Date IfD(Numerical test, Date valueTrue, Date valueFalse)

Ternary operator returning second operand (true) or third operand (false) according to the condition in first operand.

Time IfT(Numerical test, Time valueTrue, Time valueFalse)

Ternary operator returning second operand (true) or third operand (false) according to the condition in first operand.

Timestamp IfTS(Numerical test, Timestamp valueTrue, Timestamp valueFalse)

Ternary operator returning second operand (true) or third operand (false) according to the condition in first operand.

TimestampTZ IfTSTZ(Numerical test, TimestampTZ valueTrue, TimestampTZ valueFalse)

Ternary operator returning second operand (true) or third operand (false) according to the condition in first operand.

Numerical Switch(Numerical test, Numerical valueDefault, Numerical value1,…, Numerical valueK)

Switch operator that returns the numerical value corresponding to the index given by the test operand if it is between 1 and K. The default value is returned if the index if outside the bounds.

Categorical SwitchC(Numerical test, Categorical valueDefault, Categorical value1,…, Categorical valueK)

Switch operator that returns the categorical value corresponding to the index given by the test operand if it is between 1 and K. The default value is returned if the index if outside the bounds.

6.4. Copy and data conversion

Numerical Copy (Numerical value)

Copy of a numerical value. Allows to rename a variable.

Categorical CopyC(Categorical value)

Copy of a categorical value.

Date CopyD(Date value)

Copy of a date value.

Time CopyT(Time value)

Copy of a time value.

Timestamp CopyTS(Timestamp value)

Copy of a timestamp value.

TimestampTZ CopyTSTZ(Timestamp value)

Copy of a timestampTZ value.

Numerical AsNumerical(Categorical value)

Conversion of a categorical value to a numerical value. If the value to be converted is a numerical value, the rule returns the converted value. If the input value is missing or is not a numerical value, the method returns the system missing value.

Categorical AsNumericalError(Categorical value)

Label of a conversion error when converting a categorical value to a numerical value. This rule allows to analyse the missing or erroneous values of a numerical variable. This can be done by using a categorical type for the numerical variable to analyse, then by creating a derived variable with the current derivation rules. Thus, statistics on missing or erroneous values can easily be collected.

List of conversion errors:

        Unconverted end of string

        Underflow

        Overflow -inf

        Overflow +inf

Conversion OK

Numerical RecodeMissing(Numerical inputValue, Numerical replaceValue)

Returns the input value if it is different from the missing value and the replace value otherwise.

Categorical AsCategorical(Numerical value)

Conversion of a numerical value to a categorical value. This allows to process the input numerical values as unordered categorical values, and thus to analyse the variable using a value grouping method rather than a discretization method.

6.5. Character strings

If a missing value is used as operand for a rule returning a Categorical value, the return value is the empty string.

Numerical Length(Categorical value)

Length in chars of a categorical value.

Categorical Left(Categorical value, Numerical charNumber)

Extraction of the left substring of a categorical value.

If charNumber is less than 0, returns an empty value.

If charNumber is beyond the value length, returns the input value.

Categorical Right(Categorical value, Numerical charNumber)

Extraction of the right substring of a categorical value

If charNumber is less than 0, returns an empty value.

If charNumber is beyond the value length, returns the input value.

Categorical Middle(Categorical value, Numerical startChar, Numerical charNumber)

Extraction of the middle substring of a categorical value

If startChar is not valid (must start at 1), returns and empty value.

If charNumber is less than 0, returns an empty value.

If the end of the extraction is beyond the value length, returns the end of the input value.

Numerical TokenLength(Categorical value, Categorical separators)

Length in tokens of a categorical value.

A token is a non-empty substring that does not contain any separator character. The tokens are separated by one or many separator characters, which definition is given in the separator parameter.

If the separator parameter is empty, there is at most one token in the input value.

Example:

Using separators " ," (blank and comma), the categorical value " Numbers: 1, 2, 3.14, 4,5" contains exactly six tokens: 'Numbers:' '1' '2' '3.14' '4' '5'.

Categorical TokenLeft(Categorical value, Categorical separators, Numerical tokenNumber)

Extraction of the left tokens in a categorical value.

If several tokens are extracted, they remain separated by the initial separator characters used in the input value.

If the tokenNumber is less than 0, returns an empty value.

If the number of tokens is beyond the token length, returns the input value (cleaned from its begin and end separators).

Categorical TokenRight(Categorical value, Categorical separators, Numerical tokenNumber)

Extraction of the right tokens in a categorical value.

If several tokens are extracted, they remain separated by the initial separator characters used in the input value.

If the tokenNumber is less than 0, returns an empty value.

If the number of tokens is beyond the token length, returns the input value (cleaned from its begin and end separators).

Categorical TokenMiddle(Categorical value,Categorical separators, Numerical startToken, Numerical tokenNumber)

Extraction of the middle tokens in a categorical value.

If startToken is not valid (must start at 1), returns and empty value.

If several tokens are extracted, they remain separated by the initial separator characters used in the input value.

If the tokenNumber is less than 0, returns an empty value.

If the number of tokens is beyond the token length, returns the input value (cleaned from its begin and end separators).

Categorical Translate(Categorical value, Structure(VectorC) searchValues, Structure(VectorC) replaceValues)

Replace substrings in a categorical value. The replacement is performed in sequence with each search value in the first parameter vector replaced by its corresponding value in the second parameter vector.

Example:

The following rule allows to replace accented characters with regular characters:

Translate(inputValue, VectorC("é", "è", "ê", "à", "ï", "ç"), VectorC("e", "e", "e", "a", "i", "c"))

Numerical Search(Categorical value, Numerical startChar, Categorical searchValue)

Searches the position of a substring in a categorical value.

If startChar is not valid (must start at 1), returns -1.

If the substring is not found, returns -1.

Categorical Replace(Categorical value, Numerical startChar, Categorical searchValue, Categorical replaceValue)

Replaces a substring in a categorical value.

If startChar is not valid (must start at 1), returns the input value.

If the substring is not found, returns the input value, otherwise returns the modified value.

Categorical ReplaceAll(Categorical value, Numerical startChar, Categorical searchValue, Categorical replaceValue)

Replaces all substring in a categorical value.

It is the same as the Replace rule, except that Replace applies only to the first found searched values, whereas ReplaceAll applies to all found searched values

Numerical RegexMatch(Categorical value, Categorical regexValue)

Returns 1 if the entire value matches the regex, 0 otherwise.

The syntax for regular expressions is that of ECMAScript syntax (JavaScript).

See http://www.cplusplus.com/reference/regex/ECMAScript/ for the reference.

Numerical RegexSearch(Categorical value, Numerical startChar, Categorical regexValue)

Searches the position of a regular expression in a categorical value.

If startChar is not valid (must start at 1), returns -1.

If the regular expression is not found, returns -1.

Categorical RegexReplace(Categorical value, Numerical startChar, Categorical regexValue, Categorical replaceValue)

Replaces a regular expression in a categorical value.

If startChar is not valid (must start at 1), returns the input value.

If the regular expression is not found, returns the input value, otherwise returns the modified value.

Categorical RegexReplaceAll(Categorical value, Numerical startChar, Categorical regexValue, Categorical replaceValue)

Replaces all found regular expression in a categorical value.

It is the same as the ReplaceRegex rule, except that ReplaceRegex applies only to the first found searched values, whereas ReplaceAllRegex applies to all found searched values

Categorical ToUpper(Categorical value)

Conversion to upper case of a categorical value.

Categorical ToLower(Categorical value)

Conversion to lower case of a categorical value.

Categorical Concat(Categorical value1,...)

Concatenation of categorical values.

Numerical Hash(Categorical value, Numerical max)

Computes a hash value of a categorical value, between 0 and max-1.

Categorical Encrypt(Categorical value, Categorical key)

Encryption of a categorical value using an encryption key.

The encryption method used a "randomized" version of the input value. This is not a public encryption method, and it is convenient for basic use such as making the data anonymous. The encrypted value contains only alphanumeric characters. No reverse encryption method is provided.

Warning: Non printable characters are first replaced by blank characters, prior to encryption.

6.6. Math rules

Most of the math rules return a numerical value from operators having one or many numerical operands. When one operand is missing or invalid, the result is missing.

Categorical FormatNumerical(Numerical value, Numerical width, Numerical precision)

Returns a string formatted version of a numerical value, with given minimum digit number before separator and given precision after separator.

Example:

FormatNumerical (3.141592, 0, 8) -> 3.14159200

FormatNumerical (3.141592, 2, 5) -> 03.14159

FormatNumerical (3.141592, 0, 0) -> 3

Numerical Sum(Numerical value1, …)

Sum of numerical values.

Numerical Minus(Numerical value)

Opposite value.

Numerical Diff(Numerical value1, Numerical value2)

Difference between two numerical values.

Numerical Product(Numerical value1, …)

Product of numerical values.

Numerical Divide(Numerical value1, Numerical value2)

Ratio of two numerical values.

Numerical Index()

Integer index related to the line number of the current record from a data file (start at 1).

Numerical Random()

Random number between 0 and 1.

Each time a database is read, the random seed is forced to the same value, such that the sequence of random numbers will be the same.

Note that the Random rule can be used several times in dictionaries, resulting in independent sequences of random numbers.

Numerical Round(Numerical value)

Closest integer value.

Numerical Floor(Numerical value)

Largest previous integer value.

Numerical Ceil(Numerical value)

   Smallest following integer value.

Numerical Abs(Numerical value)

   Absolute value.

Numerical Sign(Numerical value)

   Sign of a numerical value.

   Returns 1 for values greater or equal than 0, -1 for values less than 0.

Numerical Mod(Numerical value1, Numerical value2)

   Returns value1 mod value2 = value1 – value2 * Floor(value1/value2).

Numerical Log(Numerical value)

   Natural logarithm.

Numerical Exp(Numerical value)

   Exponential value.

Numerical Power(Numerical value1, Numerical value2)

   Power function.

Numerical Sqrt(Numerical value)

   Square root function.

Numerical Sin(Numerical value)

   Sine function.

Numerical Cos(Numerical value)

   Cosine function.

Numerical Tan(Numerical value)

   Tangent function.

Numerical ASin(Numerical value)

   Arc-sine function.

Numerical ACos(Numerical value)

   Arc-cosine function.

Numerical ATan(Numerical value)

   Arc-tangent function.

Numerical Pi()

   Pi constant.

Numerical Mean(Numerical value1, …)

   Mean of numerical values.

Numerical StdDev(Numerical value1, …)

Standard deviation of numerical values.

Numerical Min(Numerical value1, …)

Min of numerical values (among non-missing values).

Numerical Max(Numerical value1, …)

Max of numerical values (among non-missing values).

Numerical ArgMin(Numerical value1, …)

Index of the min value in a numerical series. The index starts at 1 and relates to the first value that is equal to the min.

Numerical ArgMax(Numerical value1, …)

Index of the max value in a numerical series. The index starts at 1 and relates to the first value that is equal to the max.

6.7. Date rules

Date values are encoded in data table files using Khiops native format **YYYY-MM-DD**. Other formats are available, that allow to convert categorical values to date values.

YYYY-MM-DD	YYYY/MM/DD	YYYY.MM.DD	YYYYMMDD
DD-MM-YYYY	DD/MM/YYYY	DD.MM.YYYY	DDMMYYYY
YYYY-DD-MM	YYYY/DD/MM	YYYY.DD.MM	YYYYDDMM
MM-DD-YYYY	MM/DD/YYYY	MM.DD.YYYY	MMDDYYYY

Valid dates range from 0001-01-01 to 4000-12-31, with validity according to Gregorian calendar. Date rules return a missing value when their date operand is not valid or when a numerical operand is missing.

Categorical FormatDate(Date value, Categorical dateFormat)

Format a date into a categorical value using a date format. Date format is a categorical constant value among the available date formats (for example: "YYYY-MM-DD").

Date AsDate(Categorical dateString, Categorical dateFormat)

Recode a categorical value into a date using a date format.
Example: AsDate("2014-01-15", "YYYY-MM-DD").

Numerical Year(Date value)

Year in a date value.

Numerical Month(Date value)

Month in a date value.

Numerical Day(Date value)

Day in a date value.

Numerical YearDay(Date value)

Day in year in a date value.

Numerical WeekDay(Date value)

Day in week in a date value.

Returns 1 for monday, 2 for tuesday…, 7 for sunday.

Numerical DecimalYear(Date value)

Year in a date value, with decimal part for day in year.

Numerical AbsoluteDay(Date value)

Total elapsed days since 2000-01-01.

Numerical DiffDate(Date value1, Date value2)

Difference in days between two date values.

Date AddDays(Date value, Numerical dayNumber)

Adds a number of days to a date value.

Numerical IsDateValid(Date value)

Checks if a date value is valid.

Date BuildDate(Numerical year, Numerical month, Numerical day)

Builds a date value from year, month and day.

The year must be between 1 and 9999. The month must be between 1 and 12. The day must be between 1 and 31, and consistent with the month and year.

6.8. Time rules

Time values are encoded in data table files using Khiops native format **HH:MM:SS.**. Other formats are available, that allow to convert categorical values to time values. The dot at the end of the format means that fractions of seconds are optional. The use of parenthesis in time formats means that the corresponding digit is optional when it is null (e.g. 9:30:8 with format (H)H:(M)M:(S)S corresponds to 09:30:08 with format HH:MM:SS).

HH:MM:SS	HH:MM:SS.	HH:MM
HH.MM.SS	HH.MM.SS.	HH.MM
(H)H:(M)M:(S)S	(H)H:(M)M:(S)S.	(H)H:(M)M
(H)H.(M)M.(S)S	(H)H.(M)M.(S)S.	(H)H.(M)M
HHMMSS	HHMMSS.	HHMM

Valid times range from 00:00:00 to 23:59:59, with optional fractions of seconds up to 1/10000 of a second. Time rules return a missing value when their time operand is not valid or when a numerical operand is missing.

Categorical FormatTime(Time value, Categorical timeFormat)

Formats a time into a categorical value using a time format. Time format is a categorical constant value among the available time formats (for example: "HH:MM:SS").

Time AsTime(Categorical timeString, Categorical timeFormat)

Recodes a categorical value into a time using a time format.
Example: AsTime("18:25:00", "HH:MM:SS").

Numerical Hour(Time value)

Hour in a time value.

Numerical Minute(Time value)

Minute in a time value.

Numerical Second(Time value)

Second in a time value.

Numerical DaySecond(Time value)

Total second in a time value, since 00:00:00.

Numerical DecimalTime(Time value)

Decimal day in a time value, between 0.0 and 23.9999.

Precisely, DecimalTime=DaySecond/(24*60*60).

Numerical DiffTime(Time value1, Time value2)

Difference in seconds between two time values.

Numerical IsTimeValid(Time value)

Checks if a time value is valid.

Time BuildTime(Numerical hour, Numerical minute, Numerical second)

Builds a time value from hour, minute and second.

The rule uses the floor value of hour and minute, and the full value of second. The hour must be between 0 and 23. The minute must be between 0 and 59. The second must be between 0 and 59, with optional fraction of second.

6.9. Timestamp rules

Timestamp values are encoded in data table files using Khiops native format **YYYY-MM-DD HH:MM:SS.**. Other formats are available, that allow to convert categorical values to timestamp values.

<Date format> <Time format>: a date format followed by a blank and a time format

<Date format>-<Time format>: a date format followed by a dash and a time format

<Date format>_<Time format>: a date format followed by an underscore and a time format

<Date format>T<Time format>: a date format followed by a 'T' and a time format

<Date format><Time format>: a date format directly followed by a time format, only in case of date and time formats with not separator character (e.g. YYYYMMDDHHMMSS).

Timestamp rules return a missing value when their timestamp operand is not valid or when a numerical operand is missing.

Categorical FormatTimestamp(Timestamp value, Categorical timestampFormat)

Formats a timestamp into a categorical value using a timestamp format. Timestamp format is a categorical constant value among the available timestamp formats (for example: "YYYY-MM-DD HH:MM:SS").

Timestamp AsTimestamp(Categorical timestampString, Categorical timestampFormat)

Recodes a categorical value into a timestamp value using a timestamp format.

Example: AsTimestamp("2014-01-15 18:25:00", "YYYY-MM-DD HH:MM:SS").

Date GetDate(Timestamp value)

Date in a timestamp value.

Time GetTime(Timestamp value)

Time in a timestamp value.

Numerical DecimalYearTS(Timestamp value)

Year in a timestamp value, with decimal part for day in year, at a timestamp precision.

Numerical AbsoluteSecond(Timestamp value)

Total elapsed seconds since 2000-01-01 00:00:00.

Numerical DecimalWeekDay(Timestamp value)

Week day of the date of the timestamp value, plus decimal day of the time.

Precisely, DecimalWeekDay = WeekDay(date) + DecimalTime(time)/24.

Numerical DiffTimestamp(Timestamp value1, Timestamp value2)

Difference in seconds between two timestamp values.

Timestamp AddSeconds(Timestamp value, Numerical secondNumber)

Adds a number of seconds to a timestamp value.

Numerical IsTimestampValid(Timestamp value)

Checks if a timestamp value is valid.

Timestamp BuildTimestamp(Date dateValue, Time timeValue)

Builds a timestamp from a date and time values.

6.10. TimestampTZ rules

TimestampTZ values are encoded in data table files using Khiops native format **YYYY-MM-DD HH:MM:SS.zzzzzz**.

TimestampTZ values consist of a local timestamp value together with time zone information, using the ISO 8601 time zone format:

<Timestamp format>zzzzz: basic time zone format (Z or +hhmm or –hhmm)

<Timestamp format>zzzzzz: extended time zone format (Z or +hh:mm or –hh:mm), with hours and minutes separated by ':'

TimestampTZ values are time zone-aware whereas Timestamp values are not.

TimestampTZ values can be transformed to Timestamp values using either the LocalTimestamp or UtcTimestamp rules. Then, the Timestamp rules that extract information can be used (ex: GetDate, GetTime, DecimalYearTS, DecimalWeekDay, AbsoluteSecond).

TimestampTZ rules return a missing value when their timestampTZ operand is not valid or when a numerical operand is missing or invalid.

Categorical FormatTimestampTZ(TimestampTZ value, Categorical timestampTZFormat)

Formats a timestampTZ value into a categorical value using a timestampTZ format. TimestampTZ format is a categorical constant value among the available timestampTZ formats (for example: "YYYY-MM-DD HH:MM:SSzzzzzz").

TimestampTZ AsTimestampTZ(Categorical timestampTZString, Categorical timestampFormat)

Recodes a categorical value into a timestampTZ value using a timestampTZ format.

Example: AsTimestampTZ("2014-01-15 18:25:00+02:00", "YYYY-MM-DD HH:MM:SSzzzzzz").

Timestamp UtcTimestamp(TimestampTZ value)

Builds a timestamp value from a timestampTZ value after conversion to UTC time zone.

Example: applying the UtcTimestamp rule to timestampTZ "2020-03-21 12:15:30+02:00" returns the timestamp "2020-03-21 10:15:30".

Timestamp LocalTimestamp(TimestampTZ value)

Builds a timestamp value from a timestampTZ value after conversion to local time zone.

Example: applying the LocalTimestamp rule to timestampTZ "2020-03-21 12:15:30+02:00" returns the timestamp "2020-03-21 12:15:30".

TimestampTZ SetTimeZoneMinutes(Timestamp value, Numerical minutes)

Modify the time zone information of a timestampTZ value. The minutes must be between -12*60 and +14*60.

Example: applying the SetTimestampMinutes rule to timestampTZ "2020-03-21 12:15:30-03:00" with 120 for the minutes operand returns timestampTZ "2020-03-21 12:15:30+02:00".

Numerical GetTimeZoneMinutes(TimestampTZ value)

Returns the total minutes of the time zone (+- (hh * 60 + mm)) from a timestampTZ value.

Example: applying the GetTimestampMinutes rule to timestampTZ "2020-03-21 12:15:30+02:00" returns 120.

Numerical DiffTimestampTZ(TimestampTZ value1, TimestampTZ value2)

Difference in seconds between two timestampTZ values.

Timestamp AddSecondsTSTZ(TimestampTZ value, Numerical secondNumber)

Adds a number of seconds to a timestampTZ value.

Numerical IsTimestampTZValid(TimestampTZ value)

Checks if a timestampTZ value is valid.

TimestampTZ BuildTimestampTZ(Timestamp timestampValue, Time timeValue)

Builds a timestampTZ value from a timestamp value and time zone information in minutes. The minutes must be between -12*60 and +14*60.

6.11. Multi-table rules

The multi-table rules allow to extract values from entities in 0-1 relationship (Entity) or 0-n relationship (Table) with their owner.

```
Root Dictionary Customer (customer_id)
{
    Categorical            customer_id;
    Numerical              age;
    Categorical            sex;
    Entity(Address)        customerAddress;     // 0-1 relationship
    Table(Sale)            sales;               // 0-n relationship
};

Dictionary Address (customer_id)
{
    Categorical            customer_id;
    Categorical            street;
    Categorical            city;
    Categorical            zipcode;
    Categorical            State;
};

Dictionary Sale (customer_id)
{
    Categorical            customer_id;
    Categorical            product;
    Numerical              cost;
    Date                   purchaseDate;
};
```

The multi-table dictionaries beyond, with customers having one address and several sales, are used to illustrate the multi-table rules, with variables added in the root dictionary to extract information from the address and sales.

6.11.1. Entity rules

The entity rules return an empty (categorical) or missing (numerical) value when the entity does not exist.

Numerical Exist(Entity entityValue)

Checks is an entity exists. Returns 0 or 1.

Example:
```
Numerical   ExistingAddress = Exist( customerAddress );  // Check is address exists
```

Numerical GetValue(Entity entityValue, Numerical value)

Access to a numerical value of an entity. Returns missing value if the entity does not exist.

Example:
```
Numerical   streetNameLength = GetValue( customerAddress, Length(street) );          // Length of the street name
```

Categorical GetValueC(Entity entityValue, Categorical value)

Access to a categorical value of an entity. Returns empty value if the entity does not exist.

Example:
```
Categorical  city = GetValueC( customerAddress, city );    // City from address
```

Date GetValueD(Entity entityValue, Date value)

Access to a date value of an entity. Returns empty date if the entity does not exist.

Time GetValueT(Entity entityValue, Time value)

Access to a time value of an entity. Returns empty time if the entity does not exist.

Timestamp GetValueTS(Entity entityValue, Timestamp value)

Access to a timestamp value of an entity. Returns empty timestamp if the entity does not exist.

TimestampTZ GetValueTSTZ(Entity entityValue, TimestampTZ value)

Access to a timestampTZ value of an entity. Returns empty timestamp if the entity does not exist.

Entity GetEntity(Entity entityValue, Entity value)

Access to an entity value of an entity.

Table GetTable(Entity entityValue, Table value)

Access to a table value of an entity.

6.11.2. Table rules

Table rules return an empty (categorical) or missing (numerical) value when the table is empty.

Numerical TableCount(Table table)

Size of a table.

Example:
> *Numerical saleNumber =TableCount(sales); // Number of sales of the customer*

Numerical TableCountDistinct(Table table, Categorical value)

Number of distinct values for a categorical value in a table. A missing value is considered as a special value (empty) and counted as well.

Example:
> *Numerical saleProductNumber =TableCountDistinct(sales, product); // Number of different products in sales*

Numerical TableEntropy(Table table, Categorical value)

Entropy of a categorical value in a table. The entropy of a categorical value is analogous to the variance of a numerical value. It is large in case of all values having the same frequency in the table, and small in the case of few frequent values.

Example:
> *Numerical saleProductEntropy =TableEntropy(sales, product); // Entropy of the different products in sales*

Categorical TableMode(Table table, Categorical value)

Most frequent value for a categorical value in a table. In case of ties in frequency, the method returns the first value by lexicographic order.

Example:
> *Categorical saleMainProduct =TableMode(sales, product); // Most frequent product in sales*

Categorical TableModeAt(Table table, Categorical value, Numerical rank)

I[th] most frequent value for a categorical value in a table. Returns empty value for ranks beyond the number of different values.

Example:
> *Categorical saleSecondMainProduct =TableModeAt(sales, product, 2); // Second most frequent product in sales*

Numerical TableMean(Table table, Numerical value)

Mean of numerical values in a table.

This rule (and the other similar ones) takes only the non missing values into account. Its returns missing if the table is empty or if all the values are missing.

Example:

Numerical   saleMeanCost =TableMean(sales, cost);   // Mean of product costs in sales

Numerical TableStdDev(Table table, Numerical value)

Standard deviation of numerical values in a table.

Example:

Numerical   saleStdDevCost =TableStdDev(sales, cost);   // Standard deviation of product costs in sales

Numerical TableMedian(Table table, Numerical value)

Median of numerical values in a table.

Example:

Numerical   saleMedianCost =TableMedian(sales, cost);   // Median of product costs in sales

Numerical TableMin(Table table, Numerical value)

Min of numerical values in a table.

Example:

Numerical   saleMinCost =TableMin(sales, cost);   // Min of product costs in sales

Numerical TableMax(Table table, Numerical value)

Max of numerical values in a table.

Example:

Numerical   saleMaxCost =TableMax(sales, cost);   // Max of product costs in sales
Numerical   saleLastYearDay =TableMax(sales, YearDay(purchaseDate));   // YearDay of last purchase in sales

Numerical TableSum(Table table, Numerical value)

Sum of numerical values in a table.

Example:

Numerical   saleTotalCost =TableSum(sales, cost);   // Total cost of products in sales

6.11.3. Table management rules

```
Root Dictionary Customer (customer_id)
{
    Categorical        customer_id;
    Numerical          age;
    Categorical        sex;
    Entity(Address)    customerAddress;    // 0-1 relationship
    Table(Sale)        salesJanuary;       // 0-n relationship
    Table(Sale)        salesFebruary;      // 0-n relationship
    Table(Sale)        salesMarch;         // 0-n relationship
};
```

The multi-table dictionary beyond, with sale tables per month, is used to illustrate the table management rules.

Entity TableAt(Table table, Numerical rank)

Extraction of an entity of a table at a given rank.

Example:

Entity(Sale) firstJanuarySale =TableAt(salesJanuary, 1); // First sale of january

Entity TableAtKey(Table table, Categorical keyField1, Categorical keyField2, …)

Extraction of an entity of a table at a given key. The number of key fields must match that of the dictionary of the table.

Table TableExtraction(Table table, Numerical firstRank, Numerical lastRank)

Extraction of a sub-table containing the entities of the table between firstRank and lastRank. Ranks below 1 or beyond the size of the table are ignored.

Example:

Table(Sale) firstTenJanuarySales =TableExtraction(salesJanuary, 1, 10); // First 10 sales of january

Table TableSelection(Table table, Numerical selectionCriterion)

Selection of a sub-table containing the entities of the table that meet the selection criterion.

Example:

Table(Sale) januaryProductFooSales =TableSelection(salesJanuary, EQc(product,"Foo")); // January sales for product Foo
Table(Sale) week1Sales =TableSelection(salesJanuary, LE(YearDay(purchaseDate), 7))); // Sales for first week of year

Entity TableSelectFirst(Table table, Numerical selectionCriterion)

Return the first entity of the table that meet the selection criterion. It is equivalent to combining the TableSelection and TableAt rules (at first rank).

Example:

Entity(Sale) firstWeek1Sales =TableSelectFirst(salesJanuary, LE(YearDay(purchaseDate), 7)));
// First sale in table among the sales of first week of year

Table TableSort(Table table, simpleType sortValue1, simpleType sortValue2, …)

Sort a table by increasing order according to a list of one to many sort values. Each sort value is of type Numerical, Categorical, Time, Date, Timestamp or TimestampTZ, and can be either of variable of the table or the result of a rule computed from the rules of the table.

Table EntitySet(Entity entity1, Entity entity2, …)

Build a table from a set of entities. All the entities in the operands must be of the same type (Dictionary), and the result table will also be of the same type.

Example:

Table(Sale) salesSamples =EntitySet(TableAt(salesJanuary, 1), TableAt(salesFebruary, 1) , TableAt(salesMarch, 1));
// Table of sales for quarter 1

Table TableUnion(Table table1, Table table2, …)

Union of a set of tables. The union table contains the entities that belong to one of the table operands.

Example:

Table(Sale) salesQuarter1 =TableUnion(salesJanuary, salesFebruary, salesMarch); // Table of sales for quarter 1

Table TableIntersection(Table table1, Table table2, …)

Intersection of a set of tables. The intersection table contains the entities that belong to all the table operands.

Example:

Table(Sale) salesQuarter1 =TableUnion(salesJanuary, salesFebruary, salesMarch);   // Table of sales for quarter 1
Table(Sale) salesMonth1 =TableIntersection(salesJanuary, salesQuarter1);   // Same as salesJanuary

Table TableDifference(Table table1, Table table2)

Difference between two tables. The difference table contains the entities that belong to either of the two table operands, but not to their intersection.

Example:

```
Table(Sale)   salesMonth1and2 =TableUnion( salesJanuary, salesFebruary );          // Table of sales for months 1 and 2
Table(Sale)   salesMonth2and3 =TableUnion( salesFebruary, salesMarch );            // Table of sales for months 2 and 3
Table(Sale)   salesQuarter1 =TableUnion( salesMonth1and2, salesMonth1and2 );       // Table of sales for months 1, 2 and 3
Table(Sale)   salesMonth2 =TableIntersection( salesMonth1and2, salesMonth1and2 );  // Table of sales for month 2
Table(Sale)   salesMonth2and3 =TableDifference( salesMonth1and2, salesMonth1and2 ); // Table of sales for months 1 and 3
```

Table TableSubUnion(Table table, Table subTable)

Union of the sub-tables of a table. This applies in the case of a snowflake schema, such as for example:

- Customer: root table
    - Table(Service) services; // 0-1 relation with sub-table Service
- Service: secondary table used by Customer
    - Table(Usage) usages; // 0-1 relation with sub-table Usage
- Usage: secondary table used by Service

In the following example, the rule constructs a table that is the union of all the usages of all the services.

Example:

```
Table(Usage)            allUsages =TableSubUnion( services, usages );  // Table of all usages for all services
```

Table TableSubIntersection(Table table, Table subTable)

Intersection of the sub-tables of a table.

7. Appendix: data preparation and modeling derivation rules

This section summarizes the technical derivation rules that are used internally by Khiops to store data preparation and predictor models in dictionary files, and to compute any model output values such as variable recoding, prediction, density estimation. The aim of this appendix is to enable a quick understanding of the rules generated by Khiops: it is not intended to be usable by the data miner to constructed new variables.

7.1. Technical structures

Khiops exploits several technical types to store the results of data preparation and modeling into dictionaries:

- Vector
    - Structure(VectorC): vector of categorical values
    - Structure(Vector): vector of numerical values
- Hash map
    - Structure(HashMapC): hash map of categorical values
    - Structure(HashMap): hash map of numerical values
- Data preparation
    - Structure(DataGrid): a data grid is defined by its input partitions (discretizations or value groupings) and by the frequencies of the cells of the cross-product of the input partitions

- Partitions

  - Structure(IntervalBounds): numerical partition, defined by a sorted list of interval bounds

  - Structure(ValueGroups): categorical partition, defined by a set of exclusive groups of values (Structure(ValueGroup))

  - Structure(ValueGroup): set of exclusive categorical values

  - Structure(ValueSet): numerical partition, defined by a set of exclusive numerical values

  - Structure(ValueSetC): categorical partition,   defined by a set of exclusive categorical values

  - Structure(Frequencies): vector of frequencies

- Data preparation stats

  - Structure(DataGridStats): the data grid stats structure is defined by a data grid and a set a variables related to the input partitions of the data grid; the stats are computed with respect to the data grid cell related to the input values

- Predictors

  - Structure(Classifier): specification of a classifier

  - Structure(Regressor): specification of a regressor

  - Structure(RankRegressor): specification of a rank regressor

- Deployment of a coclustering

  - Structure(DataGridDeployment): the data grid deployment structure is defined by a data grid, the index of the deployed variable and a set a vectors of values (Vector and VectorC) for each input partitions of the data grid except the deployed variable, plus an optional vector of frequencies; this structure provides predictions related to the deployed variable, given the input distribution of the other variables

7.2. Vectors

Structure(VectorC) VectorC(Categorical value1, …)

Builds a vector of categorical values. The operands must be constants values (no variables or rules).

Structure(VectorC) TableVectorC(Table table, Categorical value)

Builds a vector of categorical values from values in a table.

Categorical ValueAtC(Structure(VectorC) vector, Numerical index)

Returns a value of a categorical vector at given index (index starts at 1). Returns "" if index out of bounds.

Structure(Vector) Vector(Numerical value1, …)

Builds a vector a numerical values. The operands must be constants values (no variables or rules).

Structure(Vector) TableVector(Table table, Numerical value)

Builds a vector of numerical values from values in a table.

Numerical ValueAt(Structure(Vector) vector, Numerical index)

Returns a value of a numerical vector at given index (index starts at 1). Returns missing value if index out of bounds.

7.3. Hash maps

Structure(HashMapC) HashMapC(Structure(VectorC) keyVector, Structure(VectorC) valueVector)

Builds a hash map of categorical values indexed by keys. The operands must come from VectorC rules of the same size, with unique keys in the vector of keys.

Structure(HashMapC) TableHashMapC(Table table, Categorical key, Categorical value)

Builds a hash map of categorical values from keys and values in a table. In case of duplicate keys in the table, the first matching value is kept.

Categorical ValueAtKeyC(Structure(HashMapC) hashMap, Categorical key)

Returns a value of a categorical hash map at given key. Returns "" is not found. It allows recoding efficiently a categorical value into another categorical value.

Example:
```
Categorical   Sex;
Categorical   Gender = ValueAtKeyC(HashMapC(VectorC("male", "female"), VectorC("Mr", "Mrs")), Sex);
```

Structure(HashMap) HashMap(Structure(VectorC) keyVector, Structure(Vector) valueVector)

Builds a hash map of numerical values indexed by keys. The operands must come from VectorC and Vector rules of the same size, with unique keys in the vector of keys.

Structure(HashMap) TableHashMap(Table table, Categorical key, Numerical value)

Builds a hash map of numerical values from keys and values in a table. In case of duplicate keys in the table, the first matching value is kept.

Numerical ValueAtKey(Structure(HashMap) hashMap, Categorical key)

Returns a value of a numerical hash map at given key. Returns missing value is not found. It allows recoding efficiently a categorical value into a numerical value.

Example:
```
Categorical   Sex;
Numerical     Gender = ValueAtKeyC(HashMapC(VectorC("male", "female"), VectorC(0, 1)), Sex);
```

7.4. Data preparation

Structure(DataGrid) DataGrid(Structure(<partition>) partition1, ..., Structure(Frequencies))

Builds a data grid structure.

The first parameters are partitions, chosen among IntervalBounds, ValueGroups, ValueSetC or ValueSet). The last argument represents the frequencies of the cells of the data grid.

For example, the following contingency table (a two-dimensionnal data grid)

Interval	Iris-setosa	Iris-versicolor	Iris-virginica
]-inf;0.75]	38	0	0
]0.75;1.55]	0	33	0
]1.55;+inf[	0	3	34

is encoded in the dictionary using a constructed variable:

```
Structure(DataGrid)    PPetalWidth    = DataGrid(IntervalBounds(0.75, 1.55),
                                        ValueSetC("Iris-setosa", "Iris-versicolor", "Iris-virginica"),
                                        Frequencies(38, 0, 0, 0, 33, 3, 0, 0, 34));
```

The cells of a contingency table are indexed according to the following example.

Interval	Iris-setosa	Iris-versicolor	Iris-virginica
]-inf;0.75]	1	4	7
]0.75;1.55]	2	5	8
]1.55;+inf[	3	6	9

Several derivation rules exploit a data grid structure to retrieve a cell given a vector of input values. For example, the vector of values (1.1, Iris-versicolor) falls into the cell which index is 5.

The vector of values can be partially complete. In this case, the other dimension has default values corresponding to the first part of their dimension. For example, the single value vector (1.1) falls into the cell which index is 2.

The type of the values can be either numerical or categorical according to the type of the input partitions of the data grid. In the following, this will be indicated using SimpleType for such parameter types.

Structure(IntervalBounds) IntervalBounds(Numerical bound1 …)

Builds a partition into interval.

Structure(ValueGroup) ValueGroup(Categorical value1 …)

Builds a group of values. The special value " * " cannot correspond to a value extracted from datasets (these ones are trimmed before being processed). This special value can be used as a "garbage value" to match any value that are not explicitly defined elsewere.

Structure(ValueGroups) ValueGroups(Structure(ValueGroup) valueGroup1 …)

Builds a partition into groups of values. The special value " * " must be defined in exactly one value group, that will be assigned to unknown values .

Structure(ValueSetC) ValueSetC(Categorical value1…)

Builds a partition into categorical values.

Structure(ValueSet) ValueSet(Numerical value1…)

Builds a partition into numerical values.

Structure(Frequencies) Frequencies(Numerical frequency1, …)

Builds a vector of frequencies.

7.5. Recoding

Many recoding methods take a partition as a first parameter and a value of the same type (numerical or categorical) as a second parameter. They retrieve the part in the partition containing the value, and output an index (like 1), which is an integer value ranging from 1 to S, where S is the size of the partition, or an identifier (like "I1"), which is a generic categorical value, or a label (like "]-∞; 2]"), which is a comprehensible categorical value.

Numerical InInterval(Structure(IntervalBounds) interval, Numerical inputValue)

Returns 1 if the input value belongs to the interval of values, 0 otherwise. The interval bounds must contain exactly two bounds, for intervals of type]lowerBound; upperBound]. For left-open or right-open intervals, use comparison derivation rules, such as G, GE, L or LE.

Numerical InGroup(Structure(ValueGroup) valueGroup, Categorical inputValue)

Returns 1 if the input value belongs to the group of values, 0 otherwise.

Numerical CellIndex(Structure(DataGrid) dataGrid, SimpleType inputValue1, …)

Computes the numerical cell index of a list of values given a data grid. The list of values are either numerical (Numerical) or categorical (Categorical) according to the type of the input partitions of the data grid.

Categorical CellId(Structure(DataGrid) dataGrid, SimpleType inputValue1, …)

Computes the categorical cell identifier of a list of values given a data grid.

Categorical CellLabel(Structure(DataGrid) dataGrid, SimpleType inputValue1, …)

Computes the cell label of a list of values given a data grid.

Numerical ValueIndexDG(Structure(DataGrid) dataGrid, SimpleType inputValue)

Computes the numerical index of a value given a univariate data grid.

Numerical PartIndexAt(Structure(DataGrid) dataGrid, Numerical index, SimpleType inputValue)

Computes the numerical part index of a value given a data grid and an index of dimension in the data grid. The index must be between 1 and the number of dimension in the data grid, and the value of the type (Numerical or Categorical) relative to the given dimension of the data grid.

Categorical PartIdAt(Structure(DataGrid) dataGrid, Numerical index, SimpleType inputValue)

Computes the categorical part identifier of a value given a data grid and an index of dimension in the data grid.

Numerical ValueRank(Structure(DataGrid) dataGrid, Numerical inputValue)

Computes the averaged normalized rank of a numerical value given a univariate numerical data grid.

Numerical InverseValueRank(Structure(DataGrid) dataGrid, Numerical inputRank)

Computes the averaged value related to a normalized rank given a univariate numerical data grid.

Structure(DataGridStats) DataGridStats(Structure(DataGrid) dataGrid, SimpleType inputValue1, …)

Computes statistics (conditional probabilities) for a list of values given a data grid. The number of input values correspond of the inputs variables, which might be inferior to the number of dimensions in that data grid. The remaining dimensions in the data grid correspond to the output variables.

Numerical SourceConditionalInfo(Structure(DataGridStats), Numerical outputIndex)

Computes the source conditional info (negative log of the conditional probability) for the input cell related to the data grids stats and for the index of the target cell of the data grid given as a parameter (index starts at 1).

Categorical IntervalId(Structure(IntervalBounds) intervalBounds, Numerical value)

Computes the categorical part identifier of a numerical value given a partition into intervals.

Categorical ValueId(Structure(ValueSet) values, Numerical value)

Computes the categorical part identifier of a numerical value given a partition into a set of numerical values.

Categorical GroupId(Structure(ValueGroups) valueGroups, Categorical value)

Computes the categorical part identifier of a categorical value given a partition into a set of groups of categorical values.

Categorical ValueIdC(Structure(ValueSetC) values, Categorical value)

Computes the categorical part identifier of a categorical value given a partition into a set of categorical values. If value is not found in the value set, returns the part identifier of the special value " * " if it is defined, 1 otherwise.

Numerical IntervalIndex(Structure(IntervalBounds) intervalBounds, Numerical value)

Computes the numerical part index of a numerical value given a partition into intervals.

Numerical ValueIndex(Structure(ValueSet) values, Numerical value)

Computes the numerical part index of a numerical value given a partition into a set of numerical values.

Numerical GroupIndex(Structure(ValueGroups) valueGroups, Categorical value)

Computes the numerical part index of a categorical value given a partition into a set of groups of categorical values.

Numerical ValueIndexC(Structure(ValueSetC) values, Categorical value)

Computes the numerical part index of a categorical value given a partition into a set of categorical values. If value is not found in the value set, returns the index of the special value " * " if it is defined, 1 otherwise.

7.6. Predictors

Structure(Classifier) NBClassifier(Structure(DataGridStats) dataGridStats1,…)

Builds a naive Bayes classifier structure from a set of data grids stats, that is from a set of conditional probabilities. Each data grid stats results from a preparation model (data grid) and input values, which allows the computation of conditional probabilities.

Structure(Classifier) SNBClassifier(Structure(Vector) variableWeights, Structure(DataGridStats) dataGridStats1,…)

Builds an averaged selective naive Bayes classifier. The first parameter is a vector of variables weights. The other parameters are the same as for the NBClassifier rule.

Structure(RankRegressor) NBRankRegressor(Structure(DataGridStats) dataGridStats1,…)

Builds a naive Bayes rank regressor structure from a set of data grids stats.

Structure(RankRegressor) SNBRankRegressor(Structure(Vector) variableWeights, Structure(DataGridStats) dataGridStats1,…)

Builds an averaged selective naive Bayes rank regressor.

Structure(Regressor) NBRegressor(Structure(RankRegressor) nbRankRegressor, Structure(DataGrid) targetValues)

Builds a naive Bayes regressor structure. The first parameter is a naive Bayes rank regressor. The second parameter is the distribution of the numerical target values, encoded as a univariate numerical data grid based on a vector of values partition.

Structure(Regressor) SNBRegressor(Structure(RankRegressor) snbRankRegressor,
    Structure(DataGrid) targetValues)

Builds a naive Bayes regressor structure.

7.7. Classifier prediction

The following derivation rules take a classifier structure as a first parameter and compute the outputs of the classifier.

Categorical TargetValue(Structure(Classifier) classifier)

Computes the most probable target value.

Numerical TargetProb(Structure(Classifier) classifier)

Computes the probability of the most probable target value.

Numerical TargetProbAt(Structure(Classifier) classifier, Categorical targetValue)

Computes the probability (score) of a given target value.

Categorical BiasedTargetValue(Structure(Classifier) classifier, Structure(Vector) biasValues)

Computes the target value with the highest score, after adding a bias to each initial target value score.

7.8. Rank regressor prediction

The following derivation rules take a rank regressor structure as a first parameter and compute the outputs of the rank regressor.

Numerical TargetRankMean(Structure(RankRegressor))

Computes the mean of the target rank.

Numerical TargetRankStandardDeviation(Structure(RankRegressor))

Computes the standard deviation of the target rank.

Numerical TargetRankDensityAt(Structure(RankRegressor), Numerical rank)

Computes the density of the target rank for a given normalized rank (between 0 and 1).

Numerical TargetRankCumulativeProbAt(Structure(RankRegressor), Numerical rank)

Computes the probability that the target rank is below a given normalized rank.

7.9. Regressor prediction

The following derivation rules take a regressor structure as a first parameter and compute the outputs of the regressor.

Numerical TargetMean(Structure(Regressor))

Computes the mean of the target value.

Numerical TargetStandardDeviation(Structure(Regressor))

Computes the standard deviation of the target value.

Numerical TargetDensityAt(Structure(Regressor), Numerical value)

Computes the density of the target for a given value.

7.10. Coclustering

A coclustering model is stored using a Structure(DataGrid). Using a Structure(DataGridDeployment), it can be deployed on a given deployed variable, for a distribution of the values on the other variables of the coclustering. Several deployment rules can then be used to get the closest part for the deployed variable, to get the distance to all the other parts on the deployed variable, and to compute aggregated frequencies on the parts of all the input variables.

Structure(DataGridDeployment) DataGridDeployment (Structure(DataGrid) dataGrid, Continuous
  deployedVariableIndex, Vector inputValues1, Vector inputValues2,…, [Vector
  inputFrequencies])

Builds a deployment structure for a given deployment variable of a data grid. The input values (Vector for Numerical and VectorC for Categorical values) correspond to the distribution of the input values. The frequency vector is optional.

Numerical PredictedPartIndex(Structure(DataGridDeployment) DataGridDeployment)

Computes the index of the closest part of the deployed variable.

Structure(Vector)PredictedPartDistances(Structure(DataGridDeployment) DataGridDeployment)

Computes the distance to all the parts of the deployed variable.

Structure(Vector) PredictedPartFrequenciesAt (Structure(DataGridDeployment)
  DataGridDeployment, Numerical inputVariableIndex)

Computes the frequency of all the parts of a given input variable.

8. Appendix: variable blocks and sparse data management

Sparse data often appears in data mining, in the case of many missing or null values in a large instances x variables data matrix. This is the standard case in text mining. It also occurs in the case of a multi-table schema, when many variables are constructed to extract information from secondary tables. For example, in the marketing domain, you might construct features such as the number of items purchased by a customer, per item type, by day of the week and by time of day, which represents potentially tens of thousands of variables, most of which contain the value 0.

Data mining algorithms can leverage sparse data to improve their efficiency, from data management, feature construction, data preparation, modeling to model deployment. Khiops automatically exploits sparse data throughout the data mining process, using variable blocks within dictionaries, sparse data format for storing data files and specific derivation rules to efficiently exploit variable blocks.

The aim of this appendix is to enable a quick understanding of the rules generated by Khiops: it is not intended to be usable by the data miner to constructed new variables.

8.1. Dictionaries and blocks of variables

The aim of Khiops dictionaries is to specify the variables, mainly with their type, name and derivations rules (see section 3.2.2. Dictionary). Variables within a dictionary can be organized in variable blocks.

A variable block is defined as following:

- A variable block is a group of variables delimited by { and } in a dictionary,

- All variables in a block must have the same type: Numerical, Categorical or Table,

- A variable block has a name, that is an identifier among all the variables and variable blocks within its dictionary,

- A variable block can be computed from a block derivation rule, but variables within a block cannot be individually computed with their own derivation rule,

- Each variable within a variable block has "VarKey", defined using meta-data
    - it is an identifier of the variable locally to its block
    - it can be either an integer, starting from 1, or an alpha-numerical value
    - VarKeys are no necessary contiguous nor ordered

Dictionaries can contain zero to many variables blocks. All dictionary variables can be used exactly the same way throughout the data mining process, regardless of whether they belong to a variable block or not:

- *Unused* if it must be ignored during data analysis,

- chosen to be the target variable, or the selection variable,

- exploited as an input of a derivation rule.

Example of one variable block with numerical VarKeys

```
Dictionary Document
{
  Categorical        DocId      ;
  Categorical        Label      ;
  {
  Numerical archive    ; <VarKey=1>
  Numerical name       ; <VarKey=2>
  …
  Numerical etrbom     ; <VarKey=61188>
  }        Words     ;
};
```

Example of one variable block with categorical VarKeys

```
Dictionary Document
{
  Categorical        DocId      ;
  Categorical        Label      ;
  {
  Numerical archive    ; <VarKey="archive">
  Numerical name       ; <VarKey="name">
  …
  Numerical etrbom     ; <VarKey="etrbom">
  }        Words     ;
};
```

8.1.1. Variable blocks and derivation rules

A variable block can be computed from a derivation rule.

The derivation rule must then be able to generate a set of (VarKey, value) pairs. Each variable of the block is initialized with the value corresponding to its VarKey, so that the number of variables having an actual value can be much smaller than the number of variables of the block. We then obtain a block of values than may be sparse.

Variables in a block can be set as *Unused* or removed from a block, which is equivalent. In both cases, the values will not be stored without error, even if they are available as output of the derivation rule.

A variable block can be used as input of a derivation rule, provided that it is fully declared in the dictionary with all its detailed specification, including all it VarKeys.

8.2. Data files with sparse data format

Khiops sparse format is documented in this appendix, but it is not intended to be usable outside of Khiops.

Khiops exploits temporary data files during the data mining process, for example when data does not fit in RAM or to efficiently handle parallel processing. In the case of variable blocks, Khiops exploits a proprietary sparse data format to exploit the potential sparsity of variable blocks.

As a reminder, in the standard case, Khiops exploits a tabular format, with or without a header line and using a field separator. Fields can contain the separator character provided that they are surrounded by double-quotes.

This is extended to the sparse format by considering each numerical or categorical variable block as a sparse field. The name of the variable block is used in the header line, and for each record, all sparse values in the same block are stored in the same field as a list of key-value pairs. Sparse fields are managed as following:

- list of pairs '*VarKey:value*' separated by the blank character,
- special cases:
  - *'Varkey:'* for a numerical variable with missing value or a categorical variable with empty value
  - *'VarKey'* for a numerical variable with value 1 or a categorical variable with value "1"
- constraints on VarKeys within a sparse field:
  - must be unique within the field
  - must be sorted in the case of a numerical VarKey
  - value associated to VarKeys that do not exist in the dictionary are ignored without error message,
- VarKeys are stored
  - as is if they are alpha-numerical
  - between simple quotes otherwise
- values are stored
  - as is if they are alpha-numerical
  - between simple quotes otherwise

8.2.1. Examples

Examples of numerical sparse fields:

- with numerical VarKeys
  ```
  8:4965 1123:350 3069:6795 3972:7531 4100:7603
  1:0 2:3.7 3:5.2
  ```

- with categorical VarKeys
  ```
  mon:1 nom:1.0 est:1 personne:1
  mon nom est personne
  'aujourdh''hui' il:1 fait 'très': beau
  ```

Examples of categorical sparse fields:

- with numerical VarKeys
    - 1:bleu 2:blanc 3:rouge

- with categorical VarKeys
    - un:bleu deux:blanc trois:rouge
    - 10:rouge 20:bleu trente:'vert pommes' 40:rose

Example of sparse files with two fields, one is a categorical variable named Class and the other is a numerical variable block named Words, which contains counts per word, with one sparse variable per word declared in the dictionary:

- with numerical VarKeys and a default value (1) per sparse variable

Class	Words
0	191 367 614 634 711 1202 1220 1311 1472 1730 2281 2572 2602 2611 2824 2855 2940 3149 3313 3560 3568 …
0	118 307 367 478 505 512 807 878 939 1024 1095 1836 1915 1961 2261 2474 2521 2633 2673 2969 3143 3193 …
0	10 184 284 297 320 375 445 588 658 1108 1411 1471 1684 1787 1878 1889 1958 1986 2133 2208 2432 2460 …
1	87 149 433 704 711 892 988 1056 1070 1234 1246 1289 1642 1669 1861 1924 1956 2081 2150 2909 3038 307 …
1	84 118 279 316 435 505 584 629 849 1029 1082 1176 1324 1327 1472 1504 1849 2004 2005 2240 2519 2568 …

- with numerical VarKeys

Class	Words
atheism	3:1 10:1 12:8 17:1 23:8 27:1 29:6 30:7 33:10 42:12 48:2 51:2 52:3 60:4 67:3 73:2 81:6 83:1 99:1 100: …
atheism	23:2 27:1 29:3 30:1 33:5 42:3 48:1 51:1 60:7 71:1 72:1 81:1 122:4 131:1 144:2 291:1 297:1 335:1 368: …
atheism	10:1 12:8 17:2 23:9 27:8 28:1 29:15 30:8 33:10 42:5 44:4 46:1 48:2 49:1 51:3 52:2 60:6 67:1 72:1 81: …
graphics	297:1 27:2 29:3 30:3 33:1 80:2 131:1 233:1 235:2 304:1 355:1 365:1 474:1 475:1 770:2 775:2 778:2 813 …
politics	29:4 30:1 33:1 35:1 51:1 143:1 245:1 388:1 466:1 467:1 511:1 747:1 748:1 770:1 775:2 778:1 850:1 23: …
politics	9:2 12:3 23:1 27:1 29:3 30:2 33:4 42:1 48:1 51:1 54:1 60:1 72:1 73:1 76:1 81:3 122:2 137:1 144:1 1 …
politics	8:1 12:3 23:5 29:3 30:1 33:6 42:6 44:1 51:1 52:3 60:4 72:2 81:1 100:2 104:1 122:3 142:1 144:2 179: …

- with categorical VarKeys

Class	Words
atheism	archive:4 name:2 atheism:10 resources:4 alt:2 last:1 modified:1 december:1 version:3 atheist:9 addre …
atheism	of:7 other:1 are:7 the:19 in:7 to:13 it:3 like:1 on:2 but:7 is:16 people:2 get:1 com:1 so:2 one:4 su …
atheism	of:6 and:7 are:2 the:13 in:5 us:1 to:4 it:6 on:1 but:2 is:3 people:2 can:2 for:3 one:1 or:1 society: …
graphics	of:1 and:2 the:9 in:1 to:3 like:1 on:1 but:1 with:2 is:5 people:1 for:7 who:1 one:1 address:1 black: …
politics	of:5 freedom:1 from:1 the:8 in:2 us:1 to:5 it:1 but:1 with:2 is:1 can:1 com:1 go:1 so:1 one:1 east:1 …
politics	of:11 from:4 and:21 are:5 the:38 in:10 to:17 it:17 like:1 on:2 their:1 but:2 with:4 is:18 north:1 pe …
politics	of:5 and:3 are:3 the:5 in:4 us:1 to:3 it:2 on:1 is:2 ca:2 people:2 for:3 who:1 by:3 see:3 they:2 tha …

8.2.2. Khiops versus SVMLight sparse format

The SVMLight tool exploits a sparse format that mainly consists of the target value followed by a list of key-value pairs with integer keys:

```
<target> <feature>:<value> <feature>:<value> ... <feature>:<value> # <info>
```

Khiops can handle any number of fields, either for dense variables or for blocks of sparse variables. In the case of one dense field followed by one block of sparse variables with numerical VarKeys, Khiops format is very close from SVMLight format:

```
<target> SEP <feature>:<value> <feature>:<value> ... <feature>:<value>
```

The main difference is that there must be a separator character (SEP) between the two fields. This separator character can be a tab or ';' for example. If the separator character is the space character, the second field that contains the block of sparse variables must be surrounded by double-quotes, as it contains the space character.

The other difference is that the dictionary must explicitly specify all the variables in the block, with their names and VarKeys.

8.3. Variable block derivation rules

8.3.1. Basic variable block rules

The following variable block rules allow to copy a block of variable within a dictionary (like Copy and CopyC), or to obtain a copy of a block of variable defined in an Entity variable (like GetValue and GeValueC).

Block(Numerical) CopyBlock(Block(numerical) valueBlock)

Copy of a block of numerical variables.

Block(Categorical) CopyBlockC(Block(Categorical) valueBlock)

Copy of a block of categorical variables.

In the following example, the block of variables named *Items* is copied to a new block named *Words*. Each variable has a unique name within the whole dictionary, and a unique VarKey within its block. The variables of the input block are copied to those of the output block having the same VarKey.

```
Dictionary Document
{
    Categorical         DocId      ;
    Categorical         Label      ;
    {
    Numerical item1     ; <VarKey=1>
    Numerical item2     ; <VarKey=2>
    …
    }         Items      ;
    {
    Numerical archive    ; <VarKey=1>
    Numerical name       ; <VarKey=2>
    …
    }         Words = CopyBlock(Items)      ;
};
```

Block(Numerical) GetBlock(Block(numerical) valueBlock)

Access to a block of numerical variables of an entity.

Block(Categorical) GetBlockC(Block(Categorical) valueBlock)

Access to a block of categorical variables of an entity.

In the following example, the block of variables named *Items* in the Entity(Document) variable *CurriculumVitae* is accessed in a new block named *CVWords*. In the main entity.

```
Root Dictionary Applicant(id_applicant)
{
   Categorical           id_ applicant        ;
   Categorical           Name       ;
   Entity(Document)      CurriculumVitae      ;
   Entity(Document)      MotivationLetter     ;
   {
   Numerical archive     ; <VarKey=1>
   Numerical name        ; <VarKey=2>

   …
   }         CVWords = GetBlock(CurriculumVitae, items)          ;
};

Dictionary Document(id_ applicant)
{
   Categorical           id_ applicant        ;
   {
   Numerical item1       ; <VarKey=1>
   Numerical item2       ; <VarKey=2>

   …
   }         Items     ;
};
```

8.3.2. Sparse partition of a secondary Table

Table variables are used in the multi-table format of Khiops to specify a 0-n relationship between two entities, for example between a customer and its usages in a secondary table.

The following rules allow to partition a secondary table into a set of part, using a Partition rule that specifies how to partition the secondary table and a TablePartitionRule that produces a block of Table variables from a secondary table and the partition specification.

Structure(Partition) Partition(Structure(<partition>) partition1, …)

Builds a partition structure, which is a cross-product of univariate partitions. The parameters are univariate partitions, chosen among IntervalBounds, ValueGroups, ValueSetC or ValueSet.

For example, the following bivariate partition exploits a ValueSetC rule to partition categorical values into three groups and a IntervalBounds rule to partition numerical rules into two intervals

Structure(Partition)   partitionServiceDuration   = Partition(ValueSetC("Mobile", "Tel", " * "), IntervalBounds(5.5));

The resulting bivariate partition consists of 6 parts, with index from 1 to 6.

	Mobile	Tel	*
]-inf;5.5]	1	2	3
]5.5;+inf[	4	5	6

Block(Table) TablePartition(Table table, Structure(Partition) partition)

Builds a block of Table parts from a secondary Table and the specification of a partition. Note that the block variable is potentially sparse, as only the non-empty parts are managed.

For example, in the following dictionary, the *usagesByServiceDuration* block of variables is computed from a TablePartition rule that divides the secondary table *Usages* into a set of sub-parts according to the partition specified in the variable *partitionServiceDuration*. Among the 6 potential parts, 4 are described in the block of variables and related to their part index using their VarKey. The other 2 parts are simply ignored in the dictionary.

```
Root Dictionary    Customer(id_customer)
{
        Categorical             id_customer;
        Categorical             Name;
        Table(Usage)            Usages;
        Structure(Partition)    partitionServiceDuration = Partition(ValueSetC("Mobile", "Tel", " * "), IntervalBounds(5.5));
        {
        Table(Usage)            MobileSmallDuration; <VarKey=1>
        Table(Usage)            TelSmallDuration;      <VarKey=2>
        Table(Usage)            MobileLargeDuration; <VarKey=4>
        Table(Usage)            TelLargeDuration;      <VarKey=5>
        }                       usagesByServiceDuration = TablePartition(Usages, partitionServiceDuration, Service, Duration);
};


Dictionary         Usage(id_customer)
{
        Categorical             id_customer;
        Categorical             Service;
        Numerical               Duration;
        Numerical               Price;
};
```

8.3.3. Computing statistics from blocks of Table variables

Blocks of Table parts can be used to produce block of values by computing the statistics of a given secondary variable on each part defined in the block.

In the following example, the TablePartitionMean rule computes the mean value of the secondary variable *Price* for each Table part in the block of Table variables *usagesByServiceDuration*.

```
Root Dictionary    Customer(id_customer)
{
        Categorical             id_customer;
        Categorical             Name;
        Table(Usage)            Usages;
        Structure(Partition)    partitionServiceDuration = Partition(ValueSetC("Mobile", "Tel", " * "), IntervalBounds(5.5));
        {
        Table(Usage)            MobileSmallDuration; <VarKey=1>
        Table(Usage)            TelSmallDuration;      <VarKey=2>
        Table(Usage)            MobileLargeDuration; <VarKey=4>
        Table(Usage)            TelLargeDuration;      <VarKey=5>
        }                       usagesByServiceDuration = TablePartition(Usages, partitionServiceDuration, Service, Duration);
        {
        Numerical               MobileSmallDurationMeanPrice; <VarKey=1>
        Numerical               TelSmallDurationMeanPrice;      <VarKey=2>
        Numerical               MobileLargeDurationMeanPrice; <VarKey=4>
        Numerical               TelLargeDurationMeanPrice;      <VarKey=5>
        }                       usagesMeanPriceByServiceDuration = TablePartitionMean(usagesByServiceDuration, Price);
};


Dictionary         Usage(id_customer)
{
        Categorical             id_customer;
        Categorical             Service;
        Numerical               Duration;
        Numerical               Price;
};
```

The following sparse rules allow to compute various statistic indicators from a block of Table parts.

Block(Numerical) TablePartitionCount(Block(Table) tableParts) tablePartition)

Number of records per part.

Example:

usagesCountsByServiceDuration = TablePartitionCount(usagesByServiceDuration);

Block(Numerical) TablePartitionCountDistinct(Block(Table) tableParts, Categorical value)

Number of distinct values per part for a given categorical variable defined in the secondary Table.

Block(Numerical) TablePartitionEntropy(Block(Table) tableParts, Categorical value)

Entropy of the distribution of the values per part for a given categorical variable defined in the secondary Table. For a part containing k distinct values with probabilities p_1, p_2... p_k, the entropy is defined as entropy = $-p_1 \log(p_1)$ $-p_2 \log(p_2)$ ... $-p_k \log(p_k)$. It can be seen as a measure of variance in the case of a categorical variable.

Block(Categorical) TablePartitionMode(Block(Table) tableParts, Categorical value)

Most frequent value per part for a given categorical variable defined in the secondary Table.

Block(Categorical) TablePartitionModeAt(Block(Table) tableParts, Categorical value, Numerical rank)

I^{th} most frequent value per part for a given categorical variable defined in the secondary Table.

Block(Numerical) TablePartitionMean(Block(Table) tableParts, Numerical value)

Mean value per part for a given numerical variable defined in the secondary Table.

Block(Numerical) TablePartitionStdDev(Block(Table) tableParts, Numerical value)

Standard deviation per part for a given numerical variable defined in the secondary Table.

Block(Numerical) TablePartitionMedian(Block(Table) tableParts, Numerical value)

Median value per part for a given numerical variable defined in the secondary Table.

Block(Numerical) TablePartitionMin(Block(Table) tableParts, Numerical value)

Min value per part for a given numerical variable defined in the secondary Table.

Block(Numerical) TablePartitionMax(Block(Table) tableParts, Numerical value)

Max value per part for a given numerical variable defined in the secondary Table.

Block(Numerical) TablePartitionSum(Block(Table) tableParts, Numerical value)

Sum of values per part for a given numerical variable defined in the secondary Table.

8.3.4. Computing statistics from blocks of values in secondary tables

A block of values in a secondary table corresponds to a list of secondary variables managed in the same block. Table rules such as TableMean, TableMode, TableStandard deviation are then extended to the case of a secondary block of variables to compute a block of values for all variables in the secondary block.

In the following example, the TableBlockSum rule computes the sum of the values of each variable in the block of variables defined in the secondary table.

```
Root Dictionary Applicant(id_applicant)
{
    Categorical          id_ applicant        ;
    Categorical          Name        ;
    Entity(Document)     CurriculumVitae      ;
    Entity(Document)     MotivationLetter     ;
    Table(Document)      RecommandationLetters           ;
    {
    Numerical archive    ; <VarKey=1>
    Numerical name       ; <VarKey=2>

    …
    }        RecommandationWords = TableBlockSum(RecommandationLetters, items)  ;
};

Dictionary Document(id_ applicant)
{
    Categorical          id_ applicant        ;
    {
    Numerical item1      ; <VarKey=1>
    Numerical item2      ; <VarKey=2>

    …
    }        Items      ;
};
```

Block(Numerical) TableBlockCountDistinct(Table table, Block(Categorical) valueBlock)

Number of distinct values per variable of a categorical block defined in the secondary Table.

Block(Numerical) TableBlockEntropy(Table table, Block(Categorical) valueBlock)

Entropy of the value distribution per variable of a categorical block defined in the secondary Table.

Block(Categorical) TableBlockMode(Table table, Block(Categorical) valueBlock)

Most frequent value per variable of a categorical block defined in the secondary Table.

Block(Numerical) TableBlockMean(Table table, Block(Numerical) valueBlock)

Mean value per variable of a numerical block defined in the secondary Table.

Block(Numerical) TableBlockStdDev(Table table, Block(Numerical) valueBlock)

Standard deviation per variable of a numerical block defined in the secondary Table.

Block(Numerical) TableBlockMedian(Table table, Block(Numerical) valueBlock)

Median value per variable of a numerical block defined in the secondary Table.

Block(Numerical) TableBlockMin(Table table, Block(Numerical) valueBlock)

Min value per variable of a numerical block defined in the secondary Table.

Block(Numerical) TableBlockMax(Table table, Block(Numerical) valueBlock)

Max value per variable of a numerical block defined in the secondary Table.

Block(Numerical) TableBlockSum(Table table, Block(Numerical) valueBlock)

Sum of values per variable of a numerical block defined in the secondary Table.