Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky
Katedra informačního a znalostního inženýrství

Studijní program: Aplikovaná informatika
Obor: Znalostní technologie

Automation of a data mining process by the LISp-Miner system

DIPLOMOVÁ PRÁCE

Student : Bc. Zuzana Ochodnická
Vedoucí : doc. Ing. Milan Šimůnek, Ph.D.

2014

Prehlásenie:

Prehlasujem, že som diplomovú prácu spracovala samostatne a že som uviedla všetky použité pramene a literatúru, z ktorej som čerpala.

V Praze dne .... 2014 . . . . . .

Zuzana Ochodnická

Poďakovanie:

Na tomto mieste by som rada poďakovala vedúcemu diplomovej práce doc. Ing. Milanovi Šimůnkovi Ph.D., za veľkú pomoc pri písaní tejto práce, za jeho rady, ochotu, trpezlivosť a pevné nervy.

Vrelý dík tiež patrí Ing. Lubošovi Dufkovi za textovú a jazykovú korektúru.

Taktiež veľmi ďakujem mojej rodine a priateľom za mnohostrannú podporu.

Abstrakt

Táto práca je zameraná na oblasť automatického data miningu. Jej cieľom je popísať oblasť automatického data miningu, vytvoriť návrh procesu automatického vytvárania data miningových úloh pre verifikáciu zadaných doménových znalostí a pre hľadanie nových znalostí a tiež implementácia verifikácie zadaných doménových znalostí s typom závislosti *influence* a prispôsobovaním prehľadávaného priestoru. Jazyk implementácie je LMCL, ktorý umožňuje použitie funkcionality systému LISp-Miner automatizovaným spôsobom. Pre tieto analýzy dát boli použité data z monitorovania znečistenia ovzdušia. Návrh aj implementácia boli úspešné a vytvorené skripty by mohli byť použité (s manuálnymi zmenami vstupných parametrov) aj pre analýzu ďalších dát.

Kľúčové slová

Automatický data mining, LMCL, LISp-Miner, prispôsobenie hľadaného priestoru

Abstract

This thesis is focused on the area of automated data mining. The aim of this thesis is a description of the area of automated data mining, creation of a design of an automated data mining tasks creation process for verification of set domain knowledge and new knowledge search, and also an implementation of verification of set domain knowledge of attribute dependency type *influence* with search space adjustments. The implementation language is the LMCL language that enables usage of the LISp-Miner system's functionality in an automated way. These data analyses were performed on data from air pollution monitoring. The design and implementation were successful and the created scripts could be used (with some manual changes in initial parameters) for analyses of another dataset as well.

Keywords

Automated data mining, LMCL, LISp-Miner, search space adjustment

Content

1 Introduction 8

2 Automated data mining 11

3 The LISp-Miner system 18

3.1 KL-Miner 19

3.2 CF-Miner 20

3.3 4ft-Miner 20

3.4 Domain knowledge 21

3.5 LMCL 22

3.6 EverMinerSimple demo 23

4 Description of data and domain 26

5 Automation assignment 28

5.1 Business and data understanding 29

5.2 Data preparation 29

5.3 Modelling and evaluation 29

5.4 Deployment 30

6 Overall design 31

6.1 Import 33

6.2 Explore 33

6.3 Preprocess 33

6.4 Domain 35

6.5 Tasks 35

6.6 Results 39

7 Tasks design 40

7.1 Domain knowledge verification 40

7.2 New knowledge search-specialized process 43

7.3 New knowledge search- AllToAll process 46

8 Domain knowledge verification 47

8.1 Attributes setting 48

8.2 Run 51

8.3 Domain knowledge verification search space adjustments 53

8.3.1 Reduction of a search space 54

8.3.2 Enlargement of a search space 58

9 Implementation 60

9.1 XochzStart 61
9.2 XochzImport 61
9.3 XochzExplore 62
9.4 XochzPreprocess 62
9.5 XochzDomain 62
9.6 XochzTasks 63
9.7 XochzDomainKnowledgeVerification 63
9.8 XochzKLMiner 64
9.9 XochzKLSpaceAdjustment 64
9.10 XochzNewKnowledgeSpecialized and XochzAllToAllProcess 65
9.11 XochzResults 65
9.12 Metabase back-ups 66

10 Testing 67
10.1 Total number of iterations 67
10.2 Run time 68
10.3 Number of found hypotheses without condition 68
10.4 Number of found hypotheses with condition 68
10.5 Comments 68

11 Conclusion 69
11.1 Literature search of the automated data mining area 69
11.2 Gain of practical knowledge of the LISp-Miner system and the LMCL scripting language 70
11.3 Creation of a design of an automated data mining tasks creation process for verification of set domain knowledge and new knowledge search 70
11.4 Implementation of verification of set domain knowledge of attribute dependency type *influence* 71
11.5 General conclusion 71

12 References 72

1 Introduction

This thesis is focused on the area of automated data mining which is a relatively new area whose aim is to automate all the steps necessary to complete an analysis of data from given domain. This idea encounters many issues, such as choosing the right problems to be solved by data mining of given data, specific data preparation of various attributes data, model creation for data of various domains, presentation of the results that is understandable for data owners and many others.

However, the benefits that the automation produces are undeniable. First, the automation enables non-expert users to operate advanced data mining analyses that demand the work of expert users in non-automated data mining analyses. Consequently, the automation makes data mining solutions available for subjects that cannot afford expensive expert data mining projects. Moreover, the automations saves time and labour of expert users who would not have to perform time consuming repeated data mining activities.

This thesis provides a theoretical insight of the automated data mining area by literature search of the area as well as an implementation of a part of suggested automated data mining process design created in this thesis by the LMCL scripting language and the LISp-Miner system's functionality.

The goals of this thesis are:

1) Literature search of the automated data mining area
2) Gain of practical knowledge of the LISp-Miner system and the LMCL scripting language
3) Creation of a design of an automated data mining tasks creation process for verification of set domain knowledge and new knowledge search
4) Implementation of verification of set domain knowledge of attribute dependency type *influence*

The text is organized as follows:

The second chapter, *Automated data mining* contains information about the automated data mining area gained by literature search.

The third chapter *The LISp-Miner system* and its numbered paragraphs describe the system whose functionality was used for designing and implementation of the automated data mining task creation process created in this thesis. It also contains information about the LMCL language which is the language of implementation in this thesis, as well as information about EverMinerSimple demo, a simplified demo version tool of an automated data mining process.

In the fourth chapter *Description of data and domain* there are some brief descriptions of the used data and the domain of the data. This step of data understanding is important for any analysis.

The fifth chapter *Automation assignment* contains description of detailed assignment and requirements for the practical part of this thesis – design of automated tasks creation process and implementation of domain knowledge verification of dependency type *influence*.

In the sixth chapter *Overall design*, the overall design of the created automated process of this thesis is illustrated. Next chapters are more and more detailed descriptions of the parts of the whole process that will be implemented and that are in the main aims of this thesis.

The seventh chapter *Tasks design* describes more detailed design of the automated tasks creation process, which is one the goals of this thesis.

In the eighth chapter *Domain knowledge verification*, there is a detailed description of a part of the tasks creation process, the domain knowledge verification of dependency type *influence*. This chapter also contains design of the search space adjustment process for the mentioned type of verification. This is also one of the main goals of this thesis and this designed part will also be implemented by the LMCL language.

The ninth chapter *Implementation* contains information about the implementation of the whole automated data mining process described in this thesis with implementation of the domain knowledge verification of dependency type *influence* as the tasks creation process, which is one branch of the designed domain knowledge verification process.

The tenth chapter *Testing* describes testing of the implemented processes designed in previous chapters.

In the eleventh chapter *Conclusion*, there is an assessment of the work and goals as well as suggestions for further development.

The last numbered chapter, the twelfth chapter *References*, contains a list of used articles and literature sourced used in this thesis.

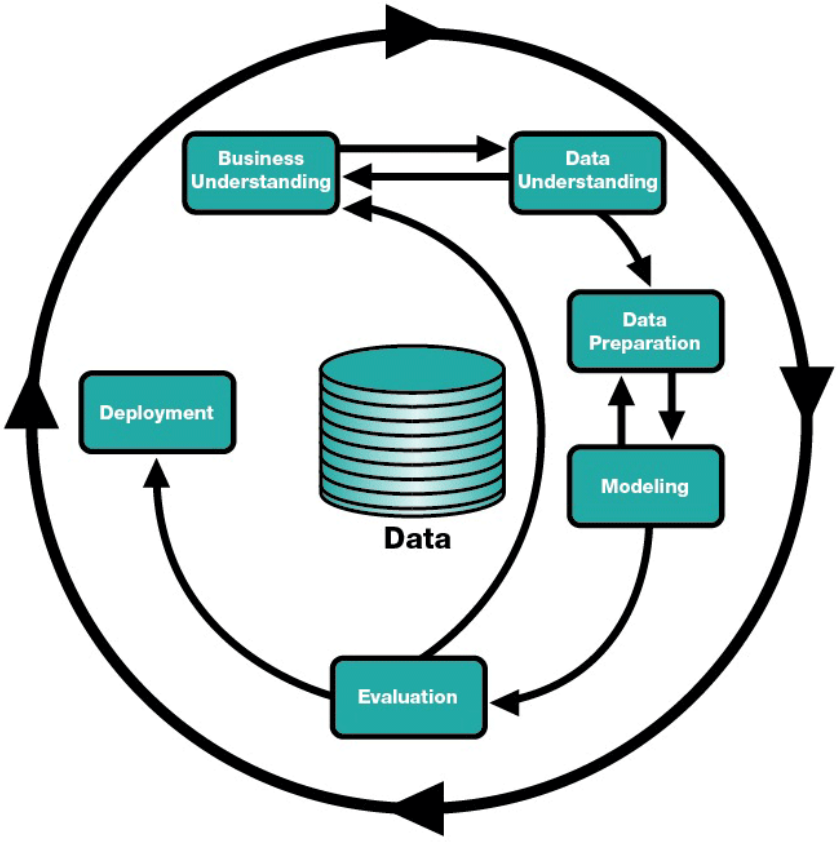
Finally, the thesis ends with appendixes of the

Picture number 16, *KL-Miner with condition space adjustment concept*, a log example, a final report example and the scripts.

2 Automated data mining

This starter chapter contains a description of problems with using and implementing automated data mining, explains why it is even interesting to automate data mining and shows examples of how automated data mining has been implemented and used.

Data mining is a complex process to gain previously unknown knowledge from data. There are several data mining methodologies, CRISP-DM being still the most used (KDNuggets, 2011). This methodology consists of processes of 1) Business understanding, 2) Data understanding, 3) Data preparation, 4) Modelling, 5) Evaluation and 6) Deployment (see *Picture number 1, the CRISP-DM model*). (Smart Vision Europe, 2011)



Picture number 1, the CRISP-DM model, (Smart Vision Europe, 2011)

All these processes require vast knowledge and are time and labour consuming. Automating of the data mining process (meaning all the methodology steps done automatically without or with only initial user interference) implies saving considerable number of resources and bringing data mining to non expert users for their own benefit.

However, automating of the data mining process produces lots of problems. Some of them were mentioned in the article *10 Challenging Problems in Data Mining Research* (Yang et al., 2006). In the chapter 8, the authors express their concerns about automating data pre-processing, namely the data cleaning part, also, they state that there is an issue in combining visual interaction and automatic data mining techniques together because in many applications, data mining goals and tasks cannot be fully specified, especially in exploratory data analysis, so visualization is necessary. The last issue mentioned is a theory behind interactive exploration of large/complex datasets and its description to users.

Additionally, there are even more problems concerning automating data mining. Even the first steps of business and understanding seemed to be strictly human domains. Consequently, there appeared discussions if it was even possible for data mining to be automated.

There was a discussion about whether it was even possible to automate the process of data mining in 2010 (Saitta, 2013), e.g. to create a "black box" data mining tool with data as an input and results as output. Consequently, the search for an answer led to analyzing which aspects of data mining could be automated and which would be very difficult or impossible.

In (Saitta, 2013) there is a description of all the CRISP-DM (Smart Vision Europe, 2011) steps with opinions about their automating. The author claims that business and data understanding are steps that cannot be automated, but the steps of data preparation, modelling and evaluation can be and already are made automatic (with problems such as for example missing values and outliers, business dependent model evaluation and large enough data) and the deployment part that is already somehow automated, but still needs manual intervention.

In *Data Mining Automation* (Coppock, 2002), the author suggests that the parts of data mining that cannot be automated are "[…] *choosing a methodology to match a business problem, selecting a data set, quality checking and preparing the data for analysis, choosing among the available options within the analysis process, and interpreting and presenting the results*." The author also declares that a data mining process can only be automated when: 1) using only data from a familiar source, 2) the needed analysis were used the same context before, 3) the included variables have been used before in the same type of analysis and 4) results will be interpreted and used in an established manner. Moreover, the author claims that automated data mining tools could produce wrong results in the hands of a non expert.

However, in spite of all the problems joined with automated data mining, there are good reasons to continue in the effort. Automated data mining enables fast and deep analysis for non expert users who otherwise would not be able to perform these analyses of their data without work of experts, which is costly.

In (Franks et al., 2010), the authors show why automated data mining is useful on practical examples. They claim that in spite of general concerns, more automated data mining processes will not make data analysts redundant, but it will allow them to focus on other important parts of the process, such as creating and testing more models. The automated process is also much faster, which is essential especially in fast-changing markets.

Furthermore, there is a lot of human labour necessary for a classical data mining project. The human resources needed for a data mining project are shown in the article *The Involvement of Human Resources in Large Scale Data Mining Projects* (Hofmann et al., 2003) which describes 8 types of roles that are needed in order to successfully complete a data mining project: Business Analyst, Data Analyst, Data Engineer, Domain Expert, Data Miner, Knowledge Engineer, Strategic Manager and Project Manager. The paper also specifies each role's responsibilities and importance as well as the needed cooperation between the roles in each stage of a data mining project. The paper proves that traditional data mining projects require lots of human labour which is unaffordable for smaller businesses. Automating of the data mining process significantly reduces the need of specialists, so that data mining analyses can be available for more users.

Consequently, there has been many attempts to automate or at least semi-automate the data mining process. One of the first steps towards automating data mining is in the patent *Method and system for simplifying the use of data mining in domain-specific analytic applications by packaging predefined data mining models* (Vishnubhotla, 2004). The inventors tried to separate data mining roles into an end user role (analytic product user) and an analytic product developer role. The analytic product developer's responsibility is to define domain areas and create proper domain specific data mining models to use in each area. The end user then is free to use only the models that have been already created by analytic product developer experts, which saves time and allows end users to benefit from data mining capabilities without needing data mining expertise.

In the present day, there are many implementation of automated or semi-automated data mining. However, most of them seem very data and area specific and do not have universal use.

One of the area specific implementations of automated data mining is described in a document *Automated and perceptual data mining of stock market data* (Nesbitt, 2003). The author writes that there are two approaches how to use and gain knowledge from large datasets: perceptual data mining tools which bring data to their users by presenting them to their senses, and auto-

mated data mining tools which allow computers to search the data and find knowledge. The author claims that these two approaches should be connected – human perceptual data mining should provide patterns for automated data mining. The automated data mining should be implemented as an agent system according to the case study performed in the document where human subjects' steps to create predictions of the stock market development should be the basis for an automated data mining tool to create automated predictions. The paper also contains a simple proposed agent model. This shows that there does not have to be a problem in combining visual interaction and automatic data mining techniques as mentioned in (Yang et al., 2006), but that there can be joined use of both approaches.

Also, the document *The Virtual Analyst Program: Automated Data Mining, Error Analysis, and Reporting*, (Moser et al., 2005) shows that combining visual interaction and automatic data mining techniques does not bring unsolvable issues. In the document, there is a focus on automating reporting process for the forest inventory and analysis program of the U.S. Department of Agriculture Forest Service. The document contains diagrams of displaying the path from database to the final report by their Virtual Analyst program. The VA has been successfully run on an example dataset and it is another example of an area specific application.

Another data mining automation attempt has been made in the document *NetFCM: A Semi-Automated Web-Based Method for Flow Cytometry Data Analysis* (Frederiksen et al., 2014). It describes a semi-automated strategy for gating (creating desirable subsets of cellular populations). There is an online clustering tool where the users can choose between Principle Component Analysis (PCA) for outliers, K-means and comparison of cells' proportions for extreme value detection. The tool was tested on its ability to detect relevant T cell populations (a type of lymphocyte) in a group of HIV-infected individuals. The test showed that the semi-automatic method identifies relevant cell populations that were obtained by a manual analysis as well in most cases. It is a tool that provides lots of data specific algorithms at one place. However, this tool is focused on a rather smaller area and cannot be used for any kind of data.

Another application using automated data mining is described in the paper *Semi-Automatic Image Data Analysis* (Burget et al., 2010). The author writes about an extension for image processing to the RapidMiner system. Two use cases are illustrated: tissue identification of mammograph pictures and sky recognition in ordinary photos. The process may be divided into three steps: feature extraction, model learning (provided standard RapidMiner operators) and results visualization. The results of both use cases were very good with only small amount of incorrectly identified pixels and the extension seems to be usable for most types of images.

One more implementation of automated data mining is illustrated in the patent *Automated data mining runs* (Dill et al., 2004). There is a description of an automated data mining process including "[…] *replicating transaction data from a source system into a data Warehouse, trigger-*

ing a data mining procedure (such as a training or a prediction procedure) that enriches the data with new attributes, and triggering the upload of the enriched data back into the data Warehouse."(Dill et al., 2004) The data mining process uses predictive modelling to determine the probability of a customer buying more products. The process automatically creates a predictive model and uses it to define customer loyalty for further data sources. The automated process is triggered when new data enter a data warehouse environment. As a result, the data mining process creates new attributes (new column) with the calculated probability rate. However, there is no detailed description of the automated data mining algorithms.

Next, in *Development of Automated Data Mining System for Quality Control in Manufacturing* (Hideyuki et al., 2001), automated data mining is used to help engineers obtain knowledge of a production process. The paper describes automated data mining system for quality control with three characteristics: "[…] *periodical-analysis, storing the result and extracting temporal-variances of the result.*" Results in this paper showed that the created system was helpful in recovering from issues in the production process.

Another system that claims to provide automated data mining is Katana Analytics Engine (Ninja Metrics, 2014), a system focused on analytics of games. On their pages the authors claim that the system is capable of providing churn predictions, segmentation and cohort analysis (segments of users that started playing the analysed game at different time). In the demo video on the webpage, there is shown that the system focuses on social analytics, finding the relationships between users and the influences of a user to spending decisions of other users.

In *Automated Data Mining from Web Servers Using Perl Script* (Neeli et al., 2008) the authors describe rather automated data extraction from web sites then automated data mining. However, in the first part of the document there is a favourable summary of steps describing how data mining of web data is performed. Then the document illustrates how authors' web data extraction functions using regular expressions to extract only the needed information from web pages.

In spite of the title of the paper *Automatic data analysis of real-time song and locomotor activity in zebra finches* (Cappendijk et al., 2013), there is no automated data mining. The authors only created models that automatically recognize different songs in sound recording data from male zebra finches as well as their locomotor activity. This study was run in order to test impact of nicotine on the birds' ability to learn new songs and also to examine the side effects – locomotor activity, changes in appetite and water intake. The team tested several data mining algorithms and compared the results with manually detected number of songs with the best result of the Euclid high-frequency boost model. All models were created in the R system. However, this study was merely a data mining project that resulted in finding of a detection model, not a project with automated data mining process.

Another misleading title is in the document *Automated Data Mining: An Innovative and Efficient Web-Based Approach to Maintaining Resident Case Logs* (Bhattacharya, 2010). In this document the authors describe a way of automatically creating resident case logs (a way of documenting all the cases the residents have done so far in their education) from residents' recording. There was a need for automation because of time consuming and incorrect manual creation of case logs. The authors describe a way how to automatically extract needed information from the unstructured recordings and create a structured Microsoft Excel spreadsheet. However, even though the described tool, Healthcare SmartGrid, is promised to be capable of performing data mining tasks, this article illustrates only data transformations from unstructured recordings to structured spreadsheets.

Furthermore, there are also systems using automated data mining that can operate with many areas of data, so that it makes them more universal. Usually the user only has to initially specify the area or a kind of task he or she would like to perform on the data and the system automatically generates the results.

One of those kinds of more universal systems in described in the patent, *Method and system for data mining automation in domain-specific analytic applications* (Vishnubhotla, 2003) that uses predefined data mining models described in the patent (Vishnubhotla, 2004). (The years of the patents in references are publication dates, not the filing dates). This way the end users do not need to possess vast data mining knowledge. The system automatically uploads production data, transfers it into a data analysis suitable form, runs predefined domain specific data mining models and uploads the results for further use. The automation is time-saving and enables performing fast analysis, making a step toward near real-time analytic reporting.

A very good example of automated data mining that can be used universally for any kind of data is illustrated in the paper *Data-Centric Automated Data mining* (Campos et al., 2010). The goal of this paper is to describe an approach that would help non expert users to do data mining without deeper studying. The paper describers 2 applications: the Oracle Database 10g Release 2 Predictive Analytics package (OPA) and the Oracle Spreadsheet Add-In for Predictive Analytics (SPA). In the document the authors try to make data mining data centric, such as queries into databases or report creating, so that data mining is easier to use for BI users, making models and complex methodologies invisible and deleting supporting objects or adding them into data sources in order to avoid problems with matching data and models. In other words, the authors wanted to automate the data mining process using only simple orders with initial parameters. The application creates tasks such as EXPLAIN, PREDICT, GROUP etc. with hidden implementation, so that the operations are similar to queries in databases. Methods EXPLAIN and PREDICT are described more in detail in the paper and both of them automatically perform all data mining methodology (most probably the SEMMA methodology) steps. The paper also contains suggest-

ed steps for implementing automated data mining methodologies in general. Moreover, the add-in SPA creates spreadsheets to present results in a more user friendly way. This approach seems to be very easy to use especially for people who work with the SQL language, because the syntax to run a data mining task is very similar. Also, the construction and initial parameters that need to be set in order to run the task seem to be very understandable even for a non data mining expert.

Another system offering universal automated data mining is IBM SPSS Analytic Catalyst (IBM, 2014). The system helps identify key drivers from big data and generally bring analysis to business users without deep data mining knowledge. The system automates data cleansing and modelling, automatically interprets results with natural language explanations, and presents analyses with interactive visuals and natural language. The role of the user is then reduced to uploading the data to analyse, select the field to predict and click start. (KDNuggets, 2013) In the tutorial video (IBM Business Analytics, 2013), there were shown the three steps needed by a user to receive results and a presentation of the results. The results were described in natural language and the whole process seemed very user friendly.

All these implementations of automated data mining are focused on a specific area or they require the user to specify the domain or the task. This thesis's goal is design of automatic tasks creation process and implementation of domain knowledge verification of dependency type *influence* with the initially parameters set to the particular used data (see chapter 4 *Description of data and domain*), however, these parameters can be manually customized for another data.

3 The LISp-Miner system

This chapter provides information about a data mining system whose functionality was used in this thesis. The chapter contains some facts about the system's history of development as well as some information about its modules and procedures. Comprehension of this system is necessary in order to use the LMCL language (Šimůnek, 2014) which will be used for implementation in this thesis. A LISp-Miner version of 24.18.00 of 23 November 2014 was used in this thesis.

The LISp-Miner system is an academic project of data mining tool for knowledge discovery in databases, constructed to support research and teaching, but also used as a tool for real life mid-size data mining projects. (LISp-Miner, 2014)

Development of the LISp-Miner system started between 1995 and 1996 at the University of Economics, Prague. Consequently, more and more people became involved in the development as well as more and more procedures and functions were added. (Šimůnek, 2010)

The LISp-Miner system is one of implementations of the GUHA (Generalized unary hypotheses automaton) method, or rather an implementation of a few GUHA procedures. A GUHA procedure is an algorithm that implements GUHA method and whose goal is to offer interesting hypotheses resulting from an analyzed dataset (Rauch, 2013). A GUHA procedure is a procedure for discovering interesting hypotheses in analyzed data usually in a form of "φ relates with ψ", where φ and ψ are logical combinations of attributes that relate together with some type of a quantifier (Hájek, 2002). Inputs of GUHA procedures are data and a definition of potentially interesting relations and output is the set of all prime patterns (hypotheses), with a pattern being prime when it is relevant, true in the analyzed data and when it cannot be logically derived from another simpler pattern that is already a part of the output (Rauch, 2013).

For this thesis practical part, the two used components were LISp-Miner.Core obtaining all the data mining modules as well as user interface for manual data mining, and LM Exec for running automated data mining processes written in a LMCL (LISp-Miner Control Language) script, described in the chapter *3.5 LMCL*. Both these components may be downloaded from (Download, 2014).

Further description is not in the focus of this thesis, but can be found in (Šimůnek, 2010), (Rauch, 2013) and (LISp-Miner, 2014).

In the next paragraphs, there is a description of the LISp-Miner procedures used for the automated data mining process designed in this thesis (see chapter 5 *Automation assignment*).

3.1 KL-Miner

KL-Miner is one of the procedures of the LISp-Miner system and it will be used later in design and implementation in this thesis (see chapter 8 *Domain knowledge verification*).

This procedure searches for interesting relationships between two attributes value frequency count. If creates hypotheses in a form of $Attribute_K$ x $Attribute_L$ / Condition (Šimůnek, 2010) or in a form of $R \approx C/\gamma$ (Rauch, 2013) which is the same form written slightly differently. The relationships can be represented by a KxL table, where rows represent categories of K attribute and columns are represented by categories of the second attribute L (*Picture number 2, KxL table in KL-Miner*). Thanks to this composition, a single cell represents the frequency count of values from a given row category of attribute K and a given column category of attribute L when they appear together in the chosen dataset.

	Rain (L)				
	categories	0-5 mm	5-10 mm	10-15 mm	15-20 mm
Dust (K)	0-10 µg/m^3	12	2	5	149
	10-20 µg/m^3	13	21	136	5
	20-30 µg/m^3	13	173	14	17
	30-40 µg/m^3	120	4	9	15

Picture number 2, KxL table in KL-Miner

For example, in the *Picture number 2, KxL table in KL-Miner*, the first number 12 means, that there are 12 rows in the source dataset that contain values from the interval 0-10 µg/m^3 for column Dust and the same 12 rows contain values from the interval 0-5 mm for column Rain.

If there are higher numbers on either of the diagonals, it means reciprocal proportion when the numbers are higher on the diagonal as shown in the *Picture number 2, KxL table in KL-Miner*, and direct proportion when the numbers are higher on the opposite diagonal. (Šimůnek, 2010) (Ochodnicka, 2012)

The quantifier used in the implementation of this procedure by the LMCL in chapter number 8 *Domain knowledge verification* , is *Kendall's TauB coefficient*, in this thesis referred to as the *Kendall's coefficient*. The *Kendall's TauB coefficient* is a statistic used to measure the association between two measured quantities (Wikipedia, 2014). The value of this coefficient can be from -1 to 1. The further the value is from 0, the larger the dependency between the chosen row and column attributes. Additionally, negative values of the *Kendall's coefficient* represent reciprocal proportion and positive numbers represent direct proportion. (Šimůnek, 2010)

3.2 CF-Miner

The CF-Miner procedure is another procedure of the LISp-Miner system and will be used in the design of the automated data mining process designed in this thesis in the chapter 7 *Tasks design*.

This procedure describes interesting frequency count tendencies for attributes. The hypotheses resulting from this procedure have the form of Attribute / Condition. In setting of each task there has to be also definition of at least one quantifier that determines the desirable frequency count tendency. The quantifiers for the CF-Miner procedure are STEPS-UP for finding increasing tendency of frequency count of an attribute's categories and STEPS-DOWN for finding decreasing tendency of frequency count of an attribute's categories. These quantifiers can also be combined, so that it is possible to search for almost any frequency count tendencies. (Ochodnicka, 2012) (Šimůnek, 2010)

3.3 4ft-Miner

The 4ft-Miner is another LISp-Miner system's procedure used in the design of the automated data mining process created in this thesis in the chapter 7 *Tasks design*.

This procedure searches for relations in a form of Antecedent ≈ Succedent / Condition. The 4ft-Miner procedure is probably the most described and used procedure, so there is no need for further description. However, more information about this procedure can be found in (Šimůnek, 2010) and (Rauch, 2013).

3.4 Domain knowledge

Domain knowledge declaration is one of the important steps to be taken to successfully run the scripts created in this thesis (see chapter 8 *Domain knowledge verification*). The LISp-Miner system provides means to do that.

The domain knowledge declaration in this thesis means to set relations and their types between a pair of attributes. There are 12 types of possible relations between attributes in the LISp-Miner system:

1. *Positive influence* -if the row attribute increases then the column attribute increases, too
2. *Negative influence* -if the row attribute increases then the column attribute decreases
3. *Some influence* -there is some not yet examined influence (negative or positive)
4. *Positive frequency* -if the row attribute increases then the relative frequency of objects satisfying column attribute increases
5. *Negative frequency* -if the row attribute increases then the relative frequency of objects satisfying column attribute decreases
6. *Positive Boolean* -if truthfulness of the row attribute increases then relative frequency of true values of the column attribute increases
7. *Negative Boolean* -if truthfulness of the row attribute increases then relative frequency of true values of the column attribute decreases
8. *Functional* -the dependency between the row and the column attribute has characteristics of a function
9. *None* -no dependency
10. *Do not care* -there might be a dependency between the row and the column attribute, but it is not important
11. *Unknown* -there are no details known if there is any dependency
12. *Not set* -the dependency is not set yet

The design of the automated task creation described in this thesis (see chapter 7 *Tasks* design) calculates with all types of dependencies, however, the implemented part (see chapter 8 *Domain knowledge verification*) uses only the *negative influence* type of dependency, because this type of relation was declared true for the domain of the used data by an expert (see chapter 4 *Description of data and domain*).

3.5 LMCL

The LMCL (LISp-Miner Control Language) language is a scripting language designed to control functions of the LISp-Miner system. LMCL is used in this thesis for the implementation part in the chapter 9 *Implementation* and the scripts are in the appendix at the end of this thesis and on the enclosed CD.

The purpose of this language is to provide a way of automating the data mining steps that would otherwise have to be taken manually through the user interface of the LISp-Miner system – the Workspace module. LMCL allows for calling of LISp-Miner internal functions and accessing user data stored in metabase. The language was built upon the LISp-Miner system (see chapter 3 *The LISp-Miner system*) and Lua scripting language (Lua, 2014) whose syntax conventions are followed in LMCL. (Šimůnek, 2014)

"The main goal is to provide a script-like mean to import data, to preprocess them, to formulate reasonable analytical tasks, to process those tasks and finally to digest results (found patterns) and to report only the interesting ones to the user." (Šimůnek, 2014)

All the LISp-Miner related classes and methods are placed in several namespaces, all of them also belonging to the *lm* namespace. There are 8 main namespaces (LISp-Miner Control Language Reference, 2014):

lm -base namespace with log, directory and sleep (for a script suspense) functions

lm.analysis -for Principal component analysis – PCA, used for dimension reduction (Raschka, 2014), not used in this thesis

lm.data -analyzed data database related functions (such as import function – see chapter *9.2 XochzImport*)

lm.domain -domain knowledge related classes and functions, for example functions for declaring domain relations between attributes (see chapter *9.5 XochzDomain*)

lm.explore -analyzed data exploration related classes and functions, initiating data tables (see chapter *9.3 XochzExplore*)

lm.metabase -LM Metabase related functions, such as opening, closing and updating a metabase (see chapter *9.2 XochzImport*)

lm.prepro -data preprocessing related classes and functions, for example attribute and categories creation (see chapter *9.4 XochzXochzPreprocess*)

lm.tasks -analytical tasks related classes and functions, such as creating and running of tasks, contains two more namespaces lm.tasks.results (namespace for task result classes) and lm.tasks.settings (for task settings classes), (see chapters *9.6 XochzTasks* and *9.8 XochzKLMiner*)

All of these namespaces (except the lm.analysis namespace) were used in the implementation part of this thesis. Further documentation description is not in the focus of this thesis, however, the whole documentation with detailed description of all the LMCL namespaces, classes and functions can be found in (LISp-Miner Control Language Reference, 2014).

The LMCL scripts are executed through the LM Exec module, which provides a means to run a chosen script. It also contains an execution log screen so that the user is always informed simultaneously about the progress of an execution.

3.6 EverMinerSimple demo

EverMinerSimple demo is an example of an implemented automated data mining tool whose main purpose is proving that the LMCL is really able to automate the KDD process. Some of the designed and implemented modules created in this thesis were strongly inspired by this demo version and some of them actually use the modules from EverMinerSimple demo with only few changes, such as the *Results* module (see chapter *9.11 XochzResults*).

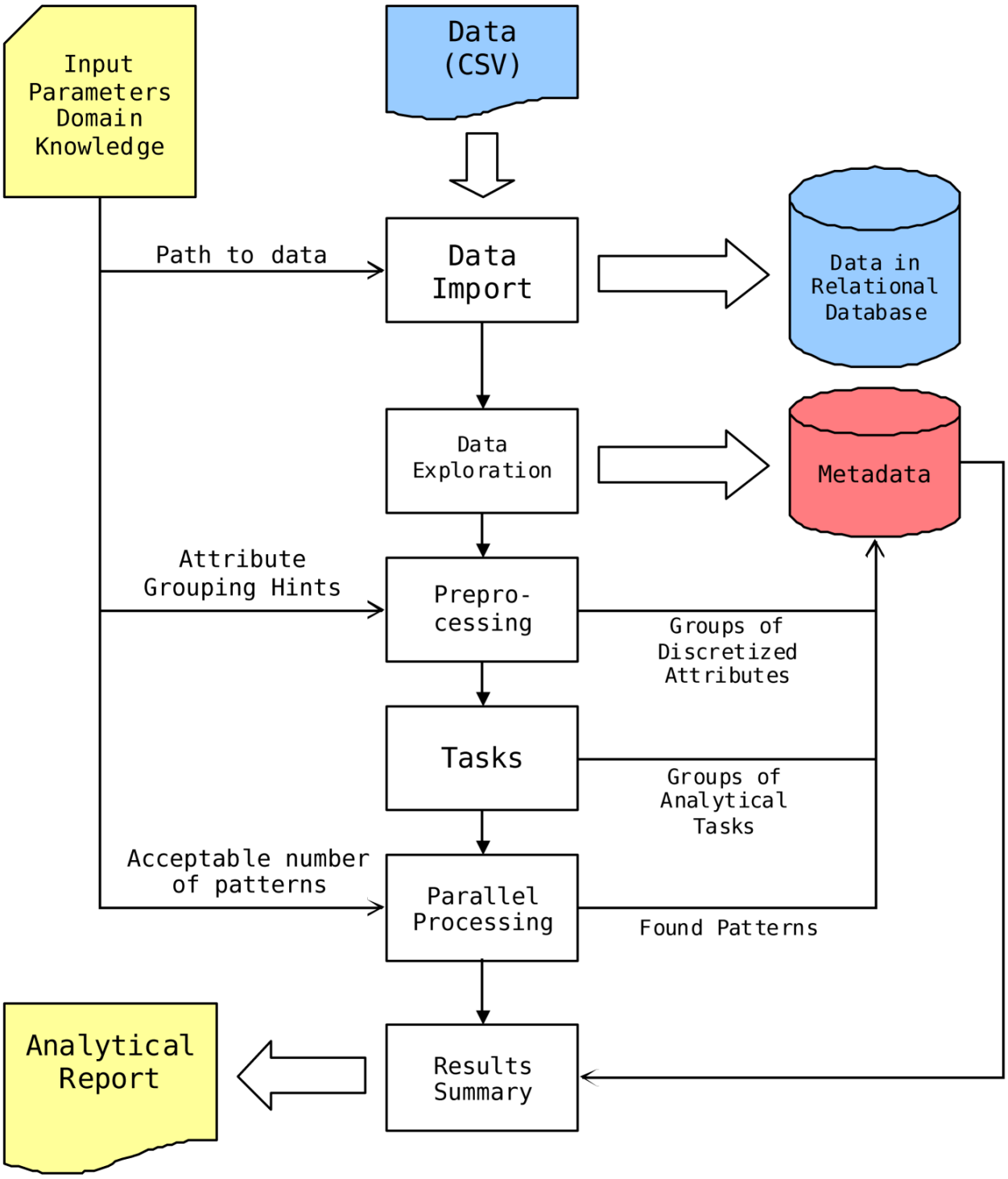
The EverMinerSimple demo is a simplified implementation of the EverMiner project for automating the data mining process (Šimůnek et al., 2011). This demo version automated process imports data from a text file, preprocess the data by creating attribute groups and attributes and their categories, creates a 4ft-Miner procedure task (see chapter *3.3 4ft-Miner*), runs it in iterations to find an acceptable number of hypotheses and finally creates a report. (Šimůnek, 2014)

The whole concept of the EverMinerSimple demo can be seen in the *Picture number 3, EverMinerSimple demo* from (Šimůnek, 2014) below.

The main difference between the EverMinerSimple demo and the tool created in this thesis (see *Picture number 4, the overall concept of the automatic data mining task creation process designed in this thesis*) is in creating the tasks provided by the EverMinerSimple demo's module *EMSTasks* and search space adjustments (see chapter 5 *Automation assignment*) which are provided by the EverMinerSimple demo's module *EMSIterations.* Another difference is that the tool created in this thesis also domain knowledge (see chapter *9.5 XochzDomain*).

In the module *EMSTasks* all the tasks are 4ft-Miner procedure's tasks and the tasks are created by using attributes from each one group as Succendent and all attributes from all the other groups as Antecedent. In this thesis design part, the tasks are created according to the additional information about the data, e.g. tasks will be created differently if there is some domain knowledge set or if there is or is not the main group of attributes set (see chapter *6.5 Tasks* and 7 *Tasks design*).

In the module *EMSIterations*, the search space adjustment is created by increasing or decreasing the used quantifier value by a certain number, while the search space adjustment designed and implemented in this thesis sets the new values as the middle value of a quantifier's value that proved to be too high and the one that proved to be too low (see chapter *8.3 Domain knowledge verification search space adjustments*). Also, there is a difference in preventing infinite loops – in the *EMSIterations* module it is done by checking a certain number of iterations that cannot be crossed as well as preventing infinite loops created by constant increasing and then decreasing (or the other way round) the quantifier's value by the same amount. In the space adjustment module created in this thesis (see chapter *9.9 XochzKLSpaceAdjustment*), the infinite loops are prevented by checking the difference in the quantifier's value last and current adjustment change. If the change is too insignificant, the process stops. Moreover, there is no need to prevent infinite loops created by constant increasing and then decreasing the quantifier's value by the same amount, since the quantifier is never increased and then decreased (or the other way round) by the same amount (see chapter *8.3 Domain knowledge verification search space adjustments*).



Picture number 3, EverMinerSimple demo (Šimůnek, 2014)

4 Description of data and domain

This chapter describes the data used for implementation and testing of the tool created for this thesis. The script modules for this thesis use some data specific knowledge set in initial parameters (such as group division of attributes) and domain knowledge part (setting of relations between attributes) as can be seen in chapters *6.3 Preprocess* and *6.4 Domain*, and that is why it is important to show the content of the data to make users easily understand the settings in the mentioned parts of the script. However, this description is only very brief as the data understanding and preparation are not in the main focus of this thesis.

The data used in this thesis are data from air quality measurements. These measurements contain values of one measurement year from each hour of measuring (to be precise, average value from at least 40 minutes of measuring – if there was less information from any measuring hour, the final value was discarded due to low confidence).

The data set is anonymous because the data owner does not wish to be known. Because of this, names of the locations where the measurements were performed are not known and will be referenced to only as place 1 or place 2.

The whole dataset is 1 121 KB and contains 17 519 lines.

The dataset' columns can be divided into three groups:

1) Columns with information about the amount of pollutants in the air

   - Values of SO_2 in $\mu g/m^3$

   - Values of NO in $\mu g/m^3$

   - Values of NO_2 in $\mu g/m^3$

   - Values of NO_X in $\mu g/m^3$

   - Values of dust particles PM10 (less than 10 micrometers in diameter) in $\mu g/m^3$

2) Columns with information about the weather conditions

   -Air temperature in °C

-Humidity in %

-Pressure in hPa

-Wind velocity in m/s

-Wind direction in degrees

-Indicator whether earth radiates or absorbs heat

3) Columns with information about time and place

-Time in DD.MM.YYYY format

-Place number 1 or number 2

Moreover, the data owners' expert stated domain knowledge that small wind velocity causes higher values of the air pollutants and this knowledge will be verified if true in the used data in the next chapters.

Additionally, the primary reason why the owner collects the data is to analyse if and when the owner's company produces air pollution concentrations higher than the limit values. However, in the used data there is too little air pollutant values that are over the limits, so an analysis of this kind would bring only unreliable results.

5 Automation assignment

This chapter defines the design and implementation assignments of this thesis.

The aim of the design part is to describe the structure of a proposed automated data mining task creation (defined below on this page) process that will verify initially set domain knowledge as well as it will search for new knowledge in the used data. Results (found hypotheses) of this designed process would most probably be interesting to a data owner.

The aim of the implementation part is to create a script that would implement a part of the whole proposed task creation process, which will be able to automatically, or at least semi-automatically perform verification of set domain knowledge relations (dependencies) between attributes with the dependency type *influence* (see chapter *3.4 Domain knowledge*) in the used data as well as perform verification if these relations are true under given conditions in the used data. The script will try to find an initially set amount of hypotheses for each declared *influence* dependency by a search space adjustment process (defined below on this page).

By task creation is meant choosing LISp-Miner procedures and their settings of attributes, conditions and quantifiers in order to gain desirable interesting hypotheses. There will be designed three task creation processes in this thesis and one of them will also be implemented (the verification of set domain knowledge relations with the dependency type *influence*, as mentioned above in this chapter). More on task creation is in the chapters *6.5 Tasks* and *5.3 Modelling and evaluation*.

By search space adjustment is meant adjusting the parameters of a task in order to gain an initially set ideal number of hypotheses. A search space in this thesis refers to a set of patterns (hypotheses) that are created according to the pattern given in the setting of a task. This set is then searched for those hypotheses that are true given the task's parameters. The search space can be reduced (used in case that the task settings are too loose: the consequence of overly loose task settings results in too many found hypotheses) or the search space can be enlarged (used in case that the task settings are too strict: the consequence of overly strict task settings results in too few found hypotheses).

Ideally, users of the implemented tool will only have to provide the data and set some initial parameters, run the script and wait for the final results report in an HTML file.

The next paragraphs contain requirements for the overall automated data mining task creation process in this thesis. The requirements are structured step by step according to the CRISP-DM methodology.

5.1 Business and data understanding

The steps of business and data understanding in general are very data specific and are not in the main focus of this thesis; however, there is a brief description of the used dataset in previous chapter 4 *Description of data and domain*, because understanding of dataset is crucial for any data analysis.

5.2 Data preparation

The data preparation part of the script should include data import to LISp-Miner, creation of an MDB database of the data, creation of a metabase and association of the database and metabase.

When the import is successful and database and metabase are successfully constructed and associated, groups of attributes, attributes and each attribute's categories may be generated for further use in the data mining tasks.

Also, known domain knowledge relations should be declared in this part. The domain knowledge should consist of information of which attributes are in a relationship and what kind of a relation it is. It will be calculated with all kinds of dependency relationships in the design part, however, only the dependency type *influence* will be fully implemented as mentioned at the beginning of this chapter.

5.3 Modelling and evaluation

In the modelling part of the CRISP-DM methodology, tasks are supposed to be generated and run. There is a need to create tasks and use procedures that will find the desirable knowledge (domain knowledge verification or search for new knowledge) and will be fast as well. The modelling part represents the task creation process mentioned at the beginning of this chapter.

In the evaluation part, the found hypotheses of previously created and run tasks will be tested whether they are sufficient. If not, the tasks will be modified by changing their parameters, run and tested again. The sufficiency of run tasks will be set as a number of found hypotheses which proved to be the most useful indicator of hypotheses' interestingness. The evaluation part represents the search space adjustment process mentioned at the beginning of this chapter.

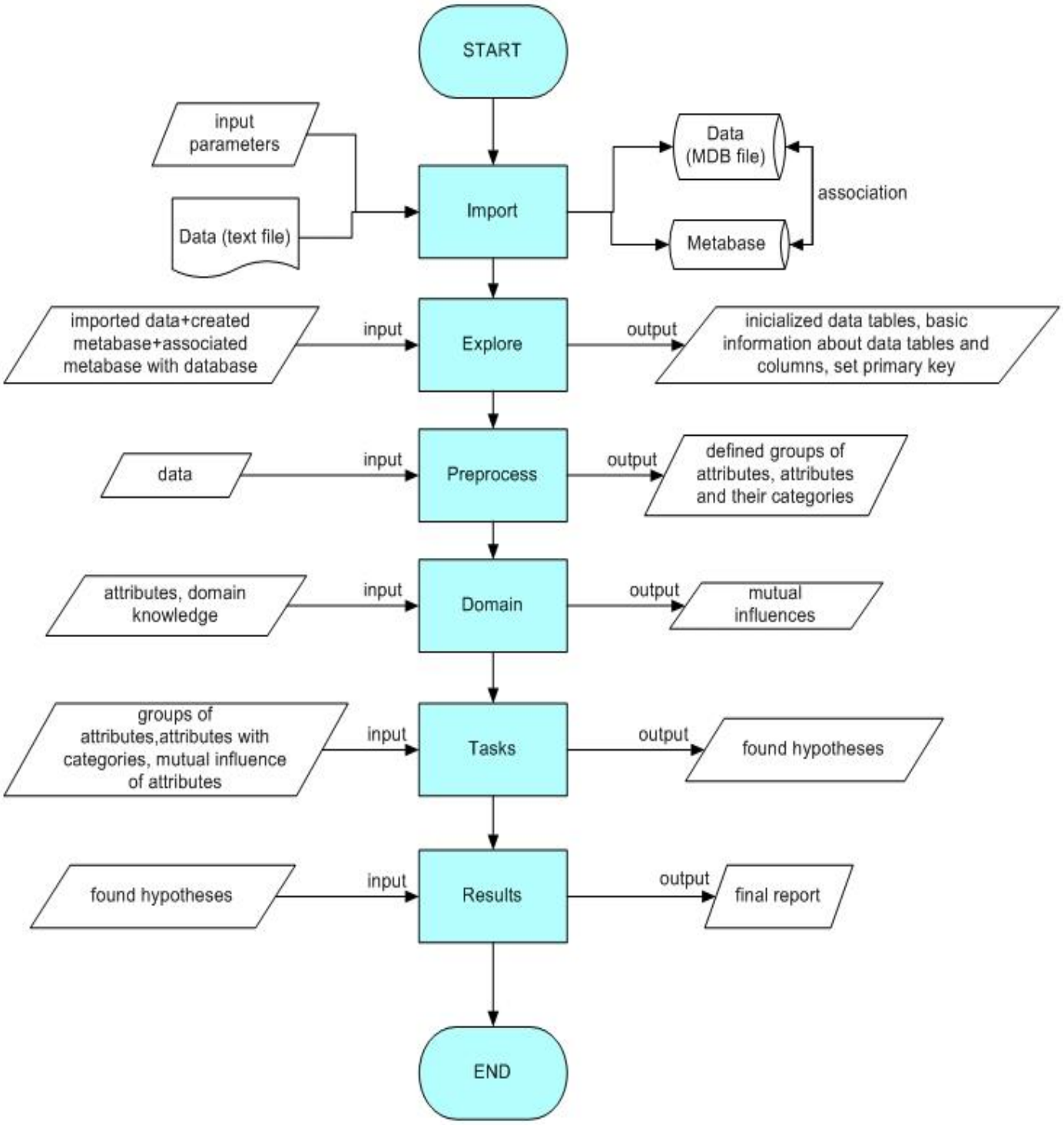
5.4 Deployment

In this thesis, the deployment should consist of a report of all found interesting hypothesis for each run task creation sub-process. It should also contain general information about the dataset, attributes and their categories.

6 Overall design

This chapter contains description of the design of the overall automated data mining process created in this thesis. Each numbered paragraph describes one step in the overall process shown below in the *Picture number 4, the overall concept of the automatic data mining task creation process designed in this thesis*. The process contains all the steps necessary to automatically run a data mining analysis designed in this thesis for the used data (described in chapter number 4 *Description of data and domain)* with the LISp-Miner system.

The design was strongly inspired by EverMiner Simple Demo (Šimůnek, 2014) as you can see on the *Picture number 3, EverMinerSimple demo* and consists of these sub-processes: import, data exploration, pre-processing to create groups of attributes, attributes and their categories, domain knowledge declaration, task modelling process and a results sub-process for final report creation. The steps are in compliance with the CRISP-DM methodology (Smart Vision Europe, 2011) for data mining and were constructed to fulfil the requirements for each step in the previous chapter (5 *Automation assignment*).



Picture number 4, the overall concept of the automatic data mining task creation process designed in this thesis

6.1 Import

The first step of the whole process is based on data and their import. The data are obliged to be in the text format (.txt) with missing values represented by an empty space (not a character made by a space bar). Additionally, names of columns should not contain any SQL key words.

When a text file is ready, the import of data may be performed. The import sub-process provides a creation of a MDB database file from the text data file as well as a creation of a metabase and the association of the metabase with the database. The metabase is a means for not changing the actual database – metabase stores all the changes and transactions, so that the real data may remain unaltered. The metabase needs to be updated after every change performed on the data in order to preserve the changes.

6.2 Explore

The exploration part sub-process calculates the basic statistics of each column, calculating maximums, minimums and averages as well as frequencies for each column and value. Exploring the data helps determine what categories should be created for each attribute and also which tasks may most likely produce interesting hypotheses. This sub-process also secures that there has been a primary key set.

However, automated data preparation is not the primary focus of this thesis, so the script will calculate the categories by automated creation of categories built in the LISp-Miner system, as described in the next chapter, (*6.3 Preprocess*).

6.3 Preprocess

The pre-processing sub-process shapes data into an acceptable form. Basically, during this process LISp- Miner creates attributes groups and attributes with their categories. The basics of dividing into categories depend on the type of data in each column.

Attribute groups will be defined in the initial parameters, as well as attributes that belong to each group. It is out of reach of this thesis to determine that automatically. There will be three attribute groups defined in compliance with the used data characteristics (see chapter number 4 *De-*

scription of data and domain: a main group (with attributes that data owners wish to be described, for example in the used data in this thesis, the main group contains air pollutant attributes), descriptive group (with attributes that are supposed to describe the main group of attributes, for example for this thesis there are weather condition attributes in the descriptive group) and a time_place group (with attributes of time and place).

For each column one or more attributes may be created, depending on its data type.

For numerical columns, the most important feature is the number of values that the column contains.

If the number of values is relatively small (for example up to around 20 distinct values) and the data type is discrete, one may create categories as enumeration-each value one category.

Columns with more than 20 distinct values should be treated as intervals as well as columns with continuous numeric data.

For columns with many distinct values it is always better to create more attributes with different scale of categories to make sure that users will not lose any knowledge due to excessively small or vast categories scale. For example, if one creates too many categories for a column containing information about rainfall (such as interval categories with length 0,1), there might not be enough data to support a hypothesis that would be true if the length of the categories were 0,2 (in 4ft cedent this, however, could be solved by proper setting of coefficient type). Similarly, if one creates too little categories (e.g. 3 interval categories-no_rain, low_rain, high_rain), the users might lose hypotheses that are bound to smaller categories scale.

Because of this, in this process of automation, for each data column there will be attributes created with these categories:

<u>For text, Boolean and numerical discrete columns with up to 20 distinct values:</u>

-enumeration categories

<u>For numerical discrete columns with more than 20 distinct values and numerical continuous columns:</u>

-equidistant and equifrequent intervals made automatically by the LISp-Miner system autocreate categories methods

<u>For date and time columns:</u>

-enumeration categories

Some attributes may need more domain knowledge for category creation, for example there are commonly used scales of wind velocity (Beaufort scale) or rainfall. However, in this thesis the categories will be created automatically for each attribute based only on data type and number of distinct values, although for example an algorithm recognizing the name of columns might be developed, it is out of the focus of this thesis.

6.4 Domain

In this part of the process, domain knowledge of dependencies between two attributes will be established according to expert declarations. The types of relations between two attributes can be *influence* (positive, negative, some), *frequency* (positive, negative), *Boolean* (positive, negative), *functional, none, do not care, unknown* and *not set* (see chapter *3.4 Domain knowledge*).

Since the implementation part of this thesis focuses only on the *influence* dependency type, the domain knowledge relations between attributes in this thesis will be set manually to negative influence between all attributes of main attribute group (see chapter *6.3 Preprocess*) and wind velocity attribute, according to general knowledge about the used data described in the chapter 4 *Description of data and domain.*

6.5 Tasks

When the attributes are prepared and ready, the process continues with the modelling part. In this part various procedures can be used in order to obtain interesting hypotheses. However, the aim is also to consume as little time as possible. Therefore, domain knowledge and additional information about attributes will be needed. The whole concept of tasks design is shown below, in the *Picture number 5, Concept of the tasks sub-process*.

There will be three task creation sub-processes constructed to search for hypotheses that might be interesting for data owners. The three sub-processes together create a complex process that is supposed to ensure that areas of a data owner's interest will be analyzed.

1) The first task creation sub-process will be focused on verification of the initially declared domain knowledge to explore whether the set domain knowledge relations are true in the used dataset. The Domain knowledge verification sub-process of the dependency type *influence* will also be implemented by the LMCL language.

2) The second task creation sub-process will search for new knowledge in the dataset. This sub-process will be designed according to previous manual experience with the dataset through the LISp-Miner user interface. The goal is to design a sequence of LISp-Miner procedures and their tasks that would find most interesting hypotheses. This specialized sub-process will only be designed, not implemented, as it is out of focus of this thesis.

3) The third task creation sub-process will be a process when there is not enough initial information to run the 2), specialized process, or when a user wants to perform more general data mining. This sub-process will create universal sequence of LISp-Miner procedures and their tasks, using all attributes for all attributes and condition settings. This sub-process, however, can be very time-demanding, but, on the other hand, it would be capable of finding the largest number of new interesting hypotheses. This sub-process will only be designed, not implemented, as it is out this thesis focus.

Detailed design of task creation processes and the search space adjustment process is described in the next chapters 7 *Tasks design* and *8.3 Domain knowledge verification search space adjustments*.

At the beginning of the tasks sub-process, it will be checked whether there was a setting of a main group of attributes, i.e. the group of attributes the users would like to be described by other attribute groups and also, whether users have established any domain knowledge (i.e. any mutual dependencies between attributes). In the created script, there will be manually set both main attribute group attributes as well as domain knowledge (see paragraph *6.4 Domain*).

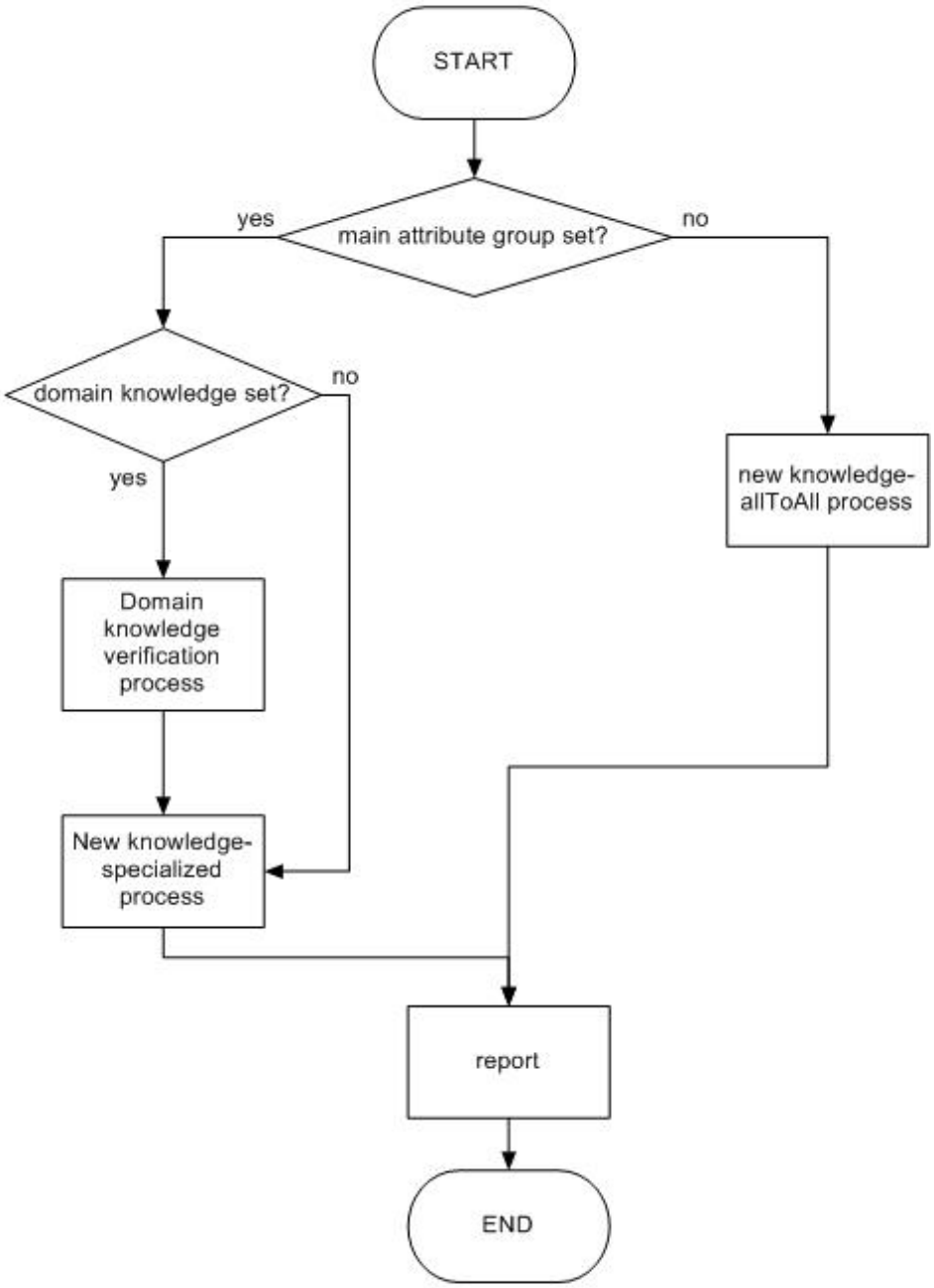
If both checks result in being true (as they should be, since the parameters have been initialized manually), the sub-process will continue with the first step of domain knowledge verification of the identified domain dependencies to discover if they are true for the used data. Then the process continues with determining whether rules found in the previous step are true under conditions and new knowledge specialized search, which is a sub-process with steps that are likely to discover new knowledge. However, the new knowledge specialized search will not be implemented as it is out of the focus of this thesis.

If there is no domain knowledge set but the main group of attributes has been set, the process should continue with a new knowledge specialized search.

If the users set no main group of attributes, then the next step is to undergo more time demanding sub-process of finding new knowledge with the AllToAll process, iterating through all attribute groups, giving all attributes as antecedents, succedents and conditions at some point, so that the

system will iterate all the possible variants of a dataset. However, implementation of the All-ToAll process is out of focus of this thesis.

The whole concept of tasks module may be seen below in *Picture number 5, Concept of the tasks sub-process*. Additionally, each step will be described in detail in the following chapter number 7 *Tasks design*.



Picture number 5, Concept of the tasks sub-process

6.6 Results

The results sub-process, will create a report that would state all the found interesting hypotheses as well as basic information about the dataset and its attributes. It would also list attributes categories.

The results will be stored in an HTML file.

This part is, however, out of the main focus of this thesis, so the results module will use the EverMiner Simple Demo (chapter *3.6 EverMinerSimple demo*) results module with slight alterations to fit this thesis script.

7 Tasks design

In this chapter, there is a description of the design of suggested ways of creating tasks as mentioned in the previous chapter in the paragraph number *6.5 Tasks*. Each paragraph illustrates each of the three tasks sub-processes and their usage. This is a more detailed design description of the sub-process *tasks* as described in the overall automated data mining process created in this thesis in the chapter number 6 *Overall design*.

7.1 Domain knowledge verification

This paragraph describes a design of one branch of the automated task creation process, the branch of domain knowledge verification. This sub-process should verify if the set domain knowledge dependencies declared in the previous step (see chapter *6.4 Domain*) are true in the used data and if they are true under given condition in the used data. The design in this paragraph considers domain knowledge verification with the LISp-Miner system in general (all possible dependency types of mutual influence in the LISp-Miner system), however, only domain knowledge verification of the dependency type *influence* will be implemented in this thesis as implementation of other types is not in the focus of this thesis.

In general, the outcome of this sub-process are tested domain knowledge relationships between attributes that have been set earlier by an expert in the *domain* sub-process (see chapter *6.4 Domain*) and found data specific exceptions against this general domain knowledge.

At the beginning, the sub-process asks for the type of the established dependency between two attributes.

If the type of dependency is *influence* (with sub-types *positive, negative* or *some*), the next step is to run KL-Miner procedure's tasks to discover whether the influence is true in the used data, if it has the same or opposite proportion as set by an expert and determine the influence in the undefined state *some influence*. The domain knowledge verification of this dependency type will also be implemented in this thesis.

For the dependency type *frequency*, the process should continue with CF-Miner procedure's tasks to explore if the defined dependencies between the two attributes are true, different or false.

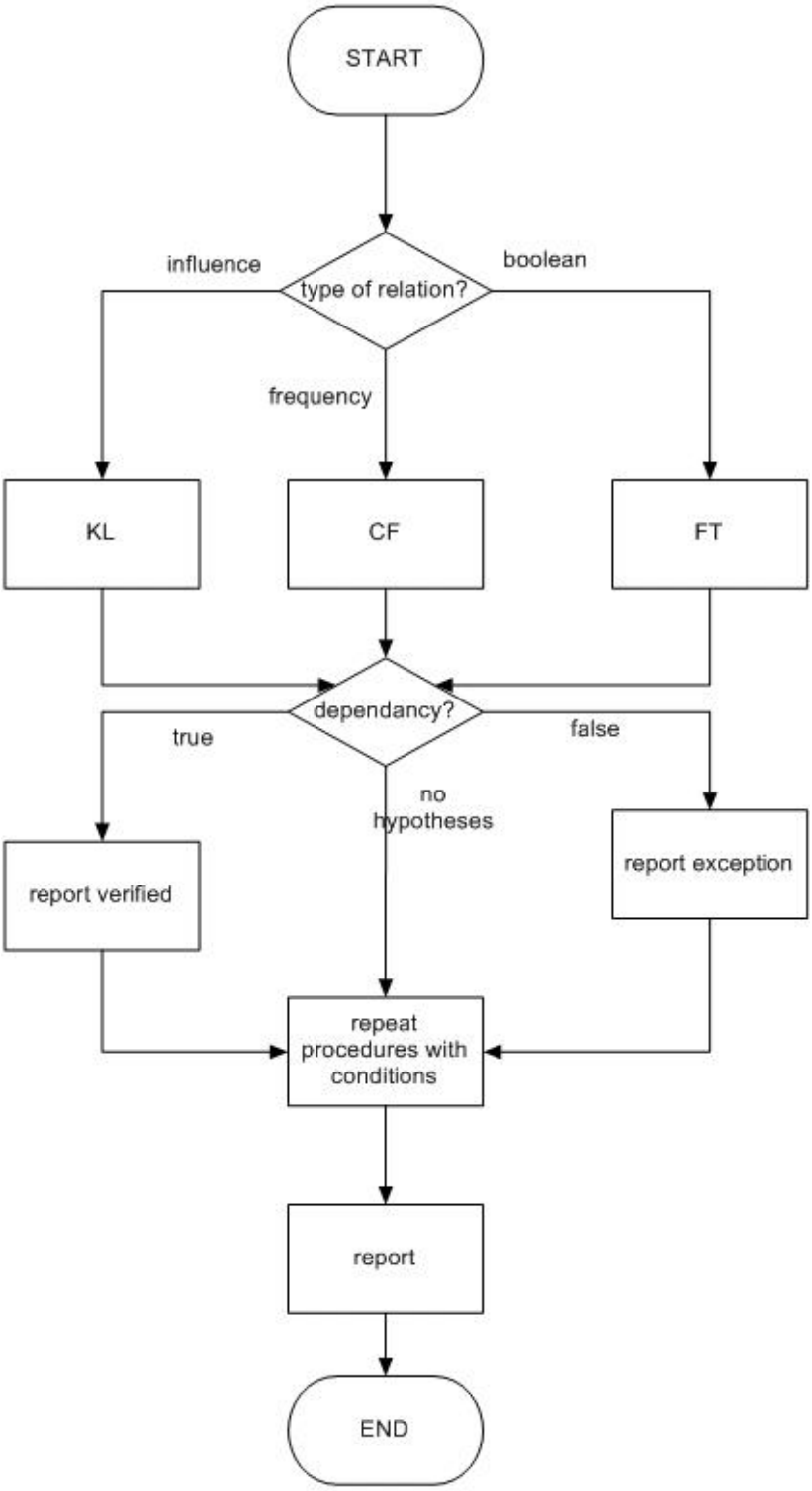
The last dependency type is *Boolean*, and for this type of dependency the process should use the 4ft-Miner procedure's tasks.

Concerning the other types of dependencies set in the domain knowledge part of the process, the attribute couples with dependencies *not set* and *unknown* should be operated in each of the domain knowledge tasks sub-processes (KL, CF and 4ft- Miner) to determine the unidentified dependency, as well as the dependency type *none* will be tested if true in all of the domain knowledge tasks sub-processes.

The attribute couples with dependencies *do not care* will not be further explored in any steps, since users do not wish to know more about relations between those two attributes. Furthermore, the attributes with the dependency type *functional* will not be further operated, since this is a very specific type of dependency.

If there are no results gained from any of the procedures, the certain procedure will be performed again with the focus on whether there are any conditions within which the dependency will be true. In order to prove the generality of the dependencies and to search for exceptions, each true and false dependency resulted in previous procedures will be run again with conditions.

At the end, the results of the domain knowledge verification process will be recorded for further use in a final report. The concept of this sub-process is shown below, in the *Picture number 6, general domain knowledge verification process*.



Picture number 6, general domain knowledge verification process design

7.2 New knowledge search-specialized process

In this step, the process will try to find new knowledge (hypotheses) from the given data.

This sub-process starts with the CF-Miner procedure because it is together with the KL-Miner procedure rather general procedure. CF-Miner was chosen to be the first one because it appears that general KL-Miner dependencies might be already known to the data owner (such as the bigger the wind velocity, the lower the pollutant values) even if they were or were not already declared and tested in the previous step (see chapter *7.1 Domain knowledge verification*). Because of this, it would be advisable to use the KL-Miner procedure only with conditions and these conditions should not be already used in the Domain knowledge verification sub-process. Also, in order to use the most interesting conditions, it is better to use the CF-Miner procedure first to gain supplementary information.

The CF-Miner procedure will help find the shape of frequencies of an attribute given a condition. The procedure will search for increasing form of frequencies (STEPS-UP quantifier), decreasing form of frequencies (STEPS-DOWN quantifier), increasing and then decreasing form (STEPS-UP then STEPS-UP quantifiers) and decreasing and then increasing form (STEPS-DOWN and then STEPS-UP quantifiers).

CF-Miner might reveal interesting behaviour of categories' frequencies of an attribute with given condition.

An important input parameter for this procedure is the identified main group of attributes (attributes that users would like to learn more about) and also results from the previous phase (see chapter *7.1 Domain knowledge verification*) to determine interesting hypotheses and reduce search space (the process will not perform the same tasks that has already been solved). The main group of attributes will be used as the attributes for CF-Miner and attributes from other groups will be used as conditions.

The following step of the sub-process depends on the results from the CF-Miner procedure part.

For the found interesting hypotheses with increasing frequencies of categories (STEPS-UP quantifier), the process will continue with the 4ft-Miner procedure task where antecedent will be attributes from non main groups, succedent will be the left cut of categories of the attribute from the CF-Miner task and condition will be the same condition as in CF-Miner. This setting of cedents is created this way because users will most probably be interested in what causes the main group attribute category values given the condition.

For found interesting hypotheses with decreasing frequencies of categories (STEPS-DOWN quantifier), the process will continue with the 4ft-Miner procedure task where antecedent will be

attributes from non main groups, succedent will be the right cut of categories of the main group attribute from the CF-Miner task and condition will be the same condition as in CF-Miner.

For found interesting hypotheses with firstly increasing and then decreasing frequencies of categories (STEPS-UP then STEPS-DOWN quantifiers), the process will continue with 4ft-Miner task where antecedent will be attributes from non main groups, succedent will be both right and left cut of categories of the main group attribute from CF-Miner task and condition will be the same condition as in CF-Miner.

And finally, for found interesting hypotheses with decreasing and then increasing frequencies of categories (STEPS-DOWN and then STEPS-UP quantifier), the process will continue with 4ft-Miner task where antecedent will be attributes from non main groups, succedent will be the middle categories of the main group attribute from the CF-Miner task and condition will be the same condition as in CF-Miner.

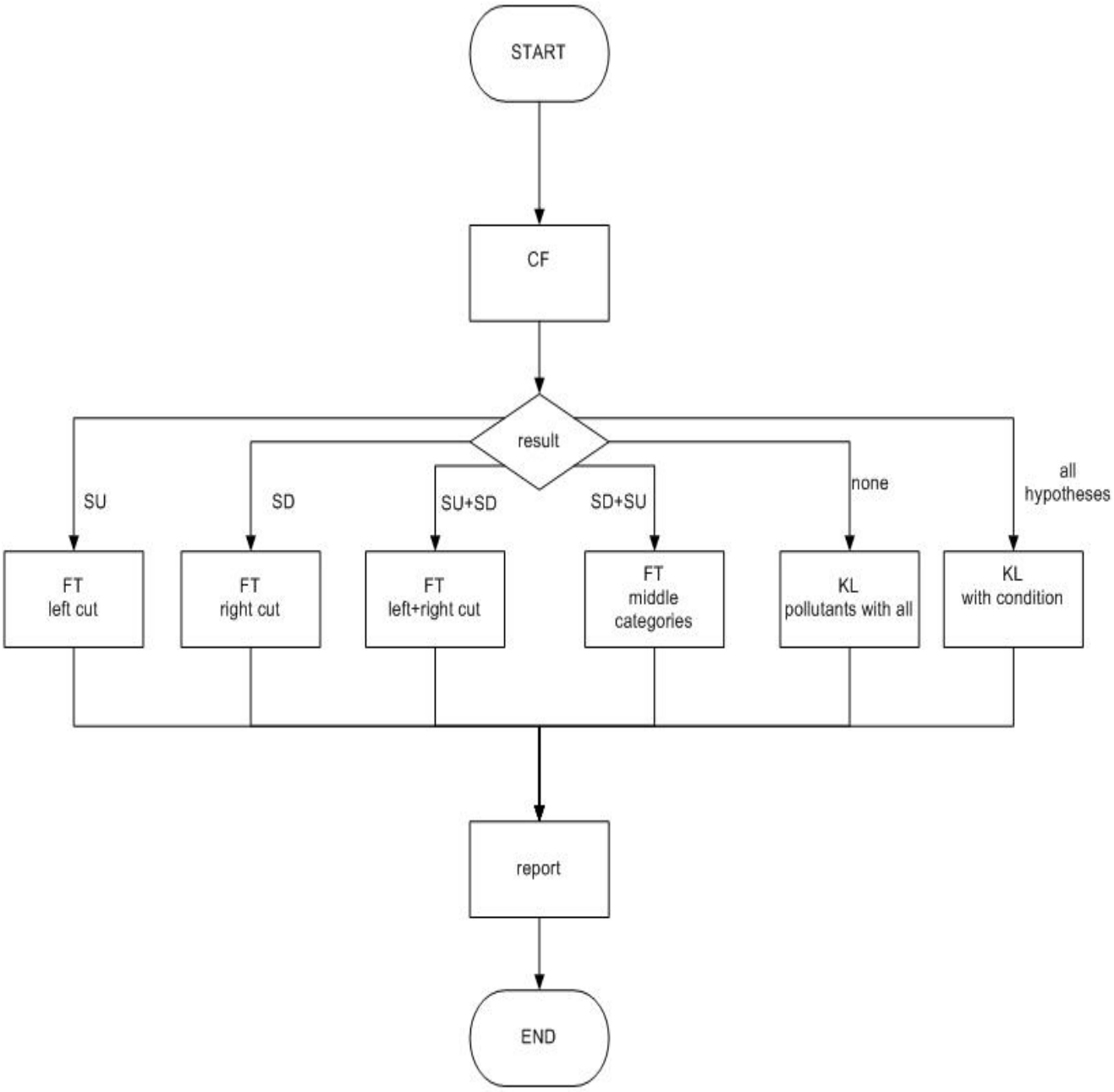
Additionally, all found hypotheses from CF-Miner will be used in the following KL-Miner procedure, where row attributes will be set as main group attributes from the relevant CF-miner task and column attributes will be attributes from other attribute groups. Condition will be the same as used in the found CF-Miner hypotheses.

Moreover, if no hypotheses are found in the CF-Miner procedure, the process will continue with the KL-Miner procedure where row attributes will be set as attributes from the main attribute group and column attributes will be attributes from other attribute groups.

At the end, all results will be kept for further use in a final report.

Before creating tasks in each procedure, the process will ensure that the tasks are unique, e.g. they have not been run yet.

The concept of this sub-process is shown below, in *Picture number 7, new knowledge search – specialized process*.



Picture number 7, new knowledge search – specialized process

7.3 New knowledge search- AllToAll process

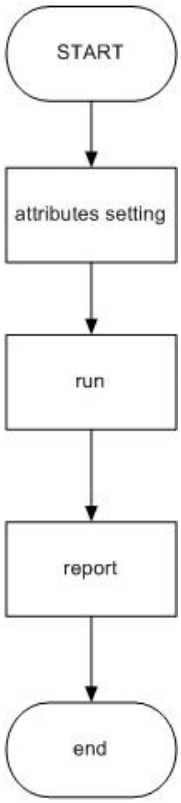
This sub-process is used when there is not enough information about the analyzed data. It is based on the iterations of all attribute groups with all attribute groups (in case it is not know what the main group is or that are no attribute groups set). This process might be very time consuming, but it should also provide largest number of interesting hypotheses.

The AllToAll process uses all the three mentioned procedures of the LISp-Miner system. Firstly, the KL-Miner procedure is used and every attribute is used both as a column and a row attribute with prevention against repetition of pairs of attributes. Then, the CF-Miner procedure will be run with all the attributes used as Attribute and all attributes used as condition. Finally, the 4ft-Miner procedure is run, using also all the attributes as antecedent, all the attributes as succedent and all the attributes as condition.

8 Domain knowledge verification

This chapter contains detailed design of the process of domain knowledge verification with dependency type *influence*, which is a branch of the automated task creation process designed in this thesis and described in the previous chapter 7 *Tasks design*. This branch verifies the set domain knowledge (chapter number *6.4 Domain*) of dependency between two attributes with the dependency type *influence*.

In order to verify the dependency type *influence in the used data*, there is a need to use the KL-Miner procedure of the LISp-Miner system (see chapter number *3.1 KL-Miner*). The KL-Miner procedure in the Domain knowledge verification sub-process starts with creation of tasks and setting attributes, initial parameters and quantifiers for the tasks. Then each task is run and iterated with changes in quantifier settings until a desirable number of hypotheses is found or there cannot be any more changes made to achieve the desirable number of hypotheses (see chapter 8.3 *Domain knowledge verification search space adjustments*). The results are recorded to be used in a final report. The concept can be seen below, in the *Picture number 8, the KL-Miner procedure in Domain knowledge verification*, and each step is described in following numbered paragraphs.



Picture number 8, the KL-Miner procedure in Domain knowledge verification

8.1 Attributes setting

At the beginning of this sub-process, the proper attributes will be chosen. The attributes used for the KL-Miner procedure should contain bigger amount of categories, so that the KxL table is big enough to determine possible dependencies and the results are unbiased. However, in the implementation this is not addressed, because the attributes used in this process have all 30 categories (see chapter number *6.3 Preprocess*).

The script will set attributes to use as column, row and condition attributes, choosing the pairs of attributes with the dependency type *influence* as claimed by users or experts in the previous step (see chapter *6.4 Domain*). In the used data, attributes from the main group of attributes (air pollutant attributes) will be used as row attributes and the wind velocity attribute will be used as the

column attribute (see chapter *6.4 Domain*). Finally, the script will assign initial parameters and quantifiers for the tasks.

In the KL-Miner procedure for Domain knowledge verification process there will be two different approaches used: the first one will be used for the KL-Miner procedure without condition (the case where the system verifies the declared domain knowledge *influence* dependencies that are supposed to be true in the whole dataset) and the second KL-Miner approach will try to verify the set *influence* dependencies with condition, with the focus on finding if the declared dependencies stay true given a condition.

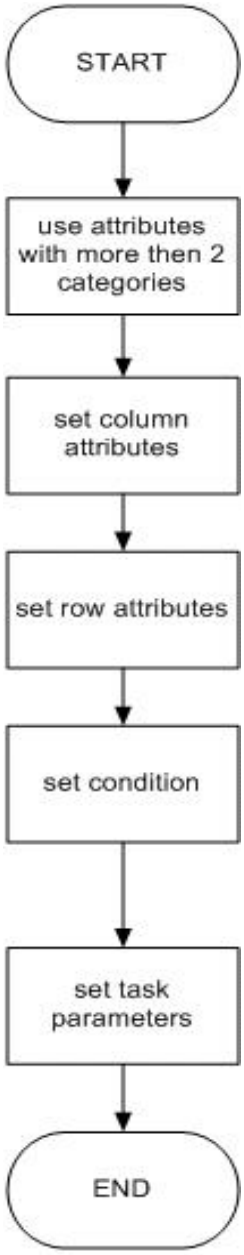
The difference in these approaches is that the first one does not use any search space adjustments, as the result can produce only one hypothesis, and the second approach uses search space adjustments in order to find the most interesting hypotheses, defined by the ideal number of hypotheses to be found. These search space adjustments will use the parameter *Kendall threshold value* (see chapter *3.1 KL-Miner*) to change the search space so that the adjustment (change of the task) is in the strength of the given *influence* dependency.

The initial *Kendall threshold value* will be set to the absolute value of 0.1 for the Domain knowledge verification without any condition scenario (which is very low value, so that if there is any dependency, the process will find it) and the absolute value of 0.5 in the Domain knowledge verification with conditions scenario, so that the tasks start with mild strength dependency and the dependency strength can be easily increased or decreased in the search space adjustment process (*8.3 Domain knowledge verification search space adjustments*).

Kendall's 0.5 dependency is rather small to be considered for a reliable factor, however, when verifying domain knowledge, it might be useful for the user to see how big (or small) the declared general domain knowledge is under given conditions. For example, it might be interesting for the user to discover that his/her general knowledge of a rule: higher wind velocity the lower amount of CO in the air, might be true only to -0.2 of *Kendall threshold value* given the condition of a wind direction 100 degrees, which would mean that it is a very insignificant dependency and it might be interesting to investigate why is it so in the used dataset.

However, the *Kendall threshold value* of 0.5 is too small for considering a dependency to be true, which is the reason why the script should not use lower values of the *Kendall threshold value* for finding new knowledge.

The Attributes setting sub-process diagram is shown below, in the *Picture number 9, Attributes setting sub-process*.



Picture number 9, Attributes setting sub-process

8.2 Run

In the Run sub-process in the KL-Miner procedure for Domain knowledge verification with dependency type *influence*, the first step is to run the tasks set in the previous sub-process for both verifications with and without condition (see *8.1 Attributes setting*).

Then, the verification without condition can result in finding a hypothesis or not finding any (since there is only one row and one column attribute set with no condition). The parameters for this sub-process are intentionally very low, allowing almost the whole search space to be examined, so that if no hypothesis is found, there is no point in adjusting the search space furthermore. As a result, all the tasks in this sub-process end after the first run, so the further lines in this chapter will consider only Domain knowledge verification with dependency type *influence* with condition.

For verification with condition, there is a need to check if there are enough hypotheses found (the desirable number of hypotheses is initially declared).

If the number of found hypotheses is the ideal number of hypotheses to be found, set initially, the results are recorded to be used in a final report. However, if the number of found hypotheses is too low or too high, the process of the search space adjustment starts.

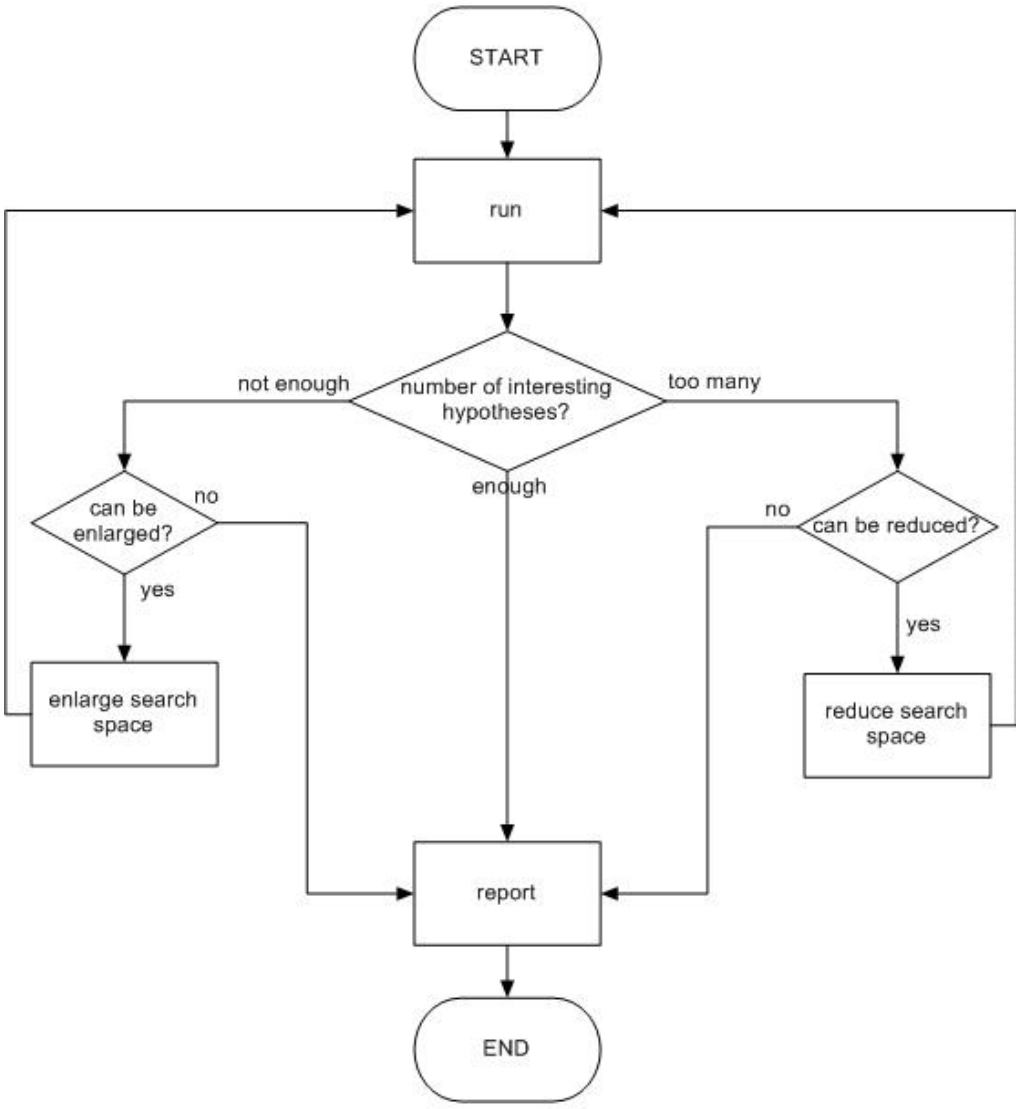
For too many found hypotheses, there will be a sub-process run in order to reduce search space, i.e. limit the sought hypotheses by making the task parameters more restricting, and, for too few hypotheses, there will be a sub-process run in order to enlarge the search space, i.e. expand the sought hypotheses by loosening the task's parameters. Consequently, after enlargement or reduction of a search space, the task will be run and tested again. This way, with some iterations, this sub-process is equipped to find the desirable number of hypotheses.

Additionally, before the actual enlargement or reduction of the search space, there is a test to determine whether it is still possible to change the search space, so that there are no infinite loops created. The diagram of the KL-run sub-process may be seen below in the picture *Picture number 10, Run sub-process*. This part, however, changed in the implementation to only checking if the adjustment change is significant enough to receive different results (see chapter *8.3 Domain knowledge verification search space adjustments*), and this check was implemented inside both enlarge search space and reduce search space functions, not before them as is shown it the *Picture number 10, Run sub-process*.

Moreover, another concept has been considered without division between enlarging and reducing the search space and creating only one adjusting process with both enlargement and reduction, since the two processes are considerably similar. However, for the script to be clear and understandable, it was preferable to create it this way. Also, without division to enlargement and re-

duction sub-processes, it would be harder to determine in which phase the process is and setting of the parameters would also be more challenging.

In the next numbered paragraphs, there will be a detailed description the of search space adjustment process, which are boxes *enlarge search space* and *reduce search space* shown below, in the *Picture number 10, Run sub-process*.



Picture number 10, Run sub-process

8.3 Domain knowledge verification search space adjustments

The concept of adjusting a KL-Miner search space is based on experiences of manual adjusting of KL task's parameters. In order to create a suitable algorithm, there was a need of understanding how the increasing (reduction) and decreasing (enlargement) of *Kendall threshold value* (see chapter *3.1 KL-Miner*) works: when and how it is needed to use these adjustments. The whole tree of possibilities may be seen below, in the *Picture number 11, Search space adjustment tree for KL-Miner*, and the larger version can be seen in the appendix part,

Picture number *16*, KL-Miner with condition space adjustment concept. The tree shows how the *Kendall threshold value* changes according to the number of found hypotheses.

The initial *Kendall threshold value* starts with 0.5, reasons for this particular setting are explained in *8.1 Attributes setting.* The principle is to find the middle value between the value that proved to be too high (found too few hypotheses) and the one that was too low (found too many hypotheses).

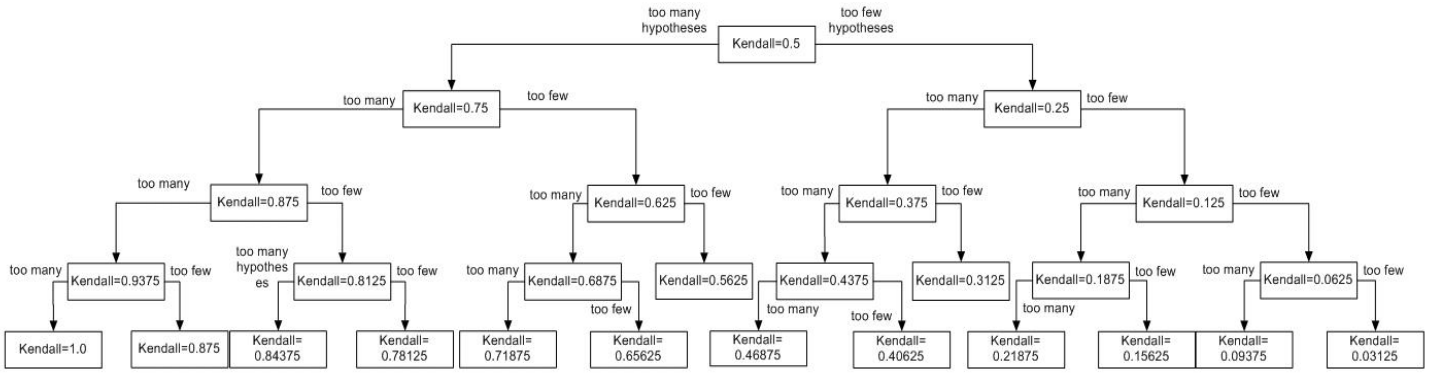
If the initial 0.5 value gives too many hypotheses, the script finds a value that is the middle between the maximal value 1.0 and value 0.5.

If the initial value 0.5 gives too few hypotheses, the script finds a middle value as well, but this time the border values are 0.5 and 0.0.

If there were only reductions, the formula for the new *Kendall threshold value* is (1.0+ value of the *Kendall threshold value* from previous task)/2.

If there were only enlargements, the formula for the new *Kendall threshold value* is value of the *Kendall threshold value* from previous task/2.

In any other case, the formula for the new *Kendall threshold value* is (*Kendall threshold value* of the turning point task that was followed by two different search space adjustments + value of the *Kendall threshold value* from previous task)/2.



Picture number 11, Search space adjustment tree for KL-Miner

8.3.1 Reduction of a search space

This description of the reduction of a search space might contain some implementation parts, however, in order to understand the whole sub-process, it is necessary to include some of the implementation instruments, such as the parameters needed in the script.

The parameters needed for this sub-process are:

K= *Kendall threshold value* used for current task

lastK= *Kendall threshold value* used for previous run task

preLastK= *Kendall threshold value* used two runs ago, i.e. the *Kendall threshold value* used in the task before previous one

turnK= *Kendall threshold value* used in the turning point task, where its run was followed by two different search space adjustments

lastSearch= parameter with the information whether the previous search was reduction or enlargement

allEnlargement= Boolean parameter, true if all previous search space adjustments were enlargements

allReduction= Boolean parameter, true if all previous search space adjustments were reductions

The main function of this sub-process is to set the right value of the *Kendall threshold value* (see chapter *3.1 KL-Miner*) for the lastly run task, so that there are less hypotheses found then in the previous run. At the beginning the script asks whether all previous search space adjustments were reductions.

If that is true, the process can easily increase the value of the *Kendall threshold value* (in order to limit the search space, the *Kendall threshold value* must be increased) by dividing the sum of previously used *Kendall threshold value* and 1.0 which is the maximum value for the *Kendall threshold value.* Since the *Kendall threshold value* is set to be in the absolute value, we can be sure that the reciprocal proportion with negative values is not a problem.

However, if all the previous search space adjustments were not reductions, the script needs to determine what kind of search space adjustment was the previous adjustment.

If it was reduction as well, then the *Kendall threshold value* will be calculated as the previous *Kendall threshold value* plus turning point *Kendall threshold value* divided by 2.

However, if the previous search space adjustment was not reduction, the script needs to set a new turning point *Kendall threshold value* to the *Kendall threshold value* used in the task before previous task and then the *Kendall threshold value* for the next task will be calculated as the previous *Kendall threshold value* plus the newly set turning point *Kendall threshold value* divided by 2.

In order not to create infinite loops and to make sure that the adjusted tasks will produce better results than the previous ones, the script checks if the difference between the last used *Kendall threshold value* and current *Kendall threshold value* is greater than 0.04.

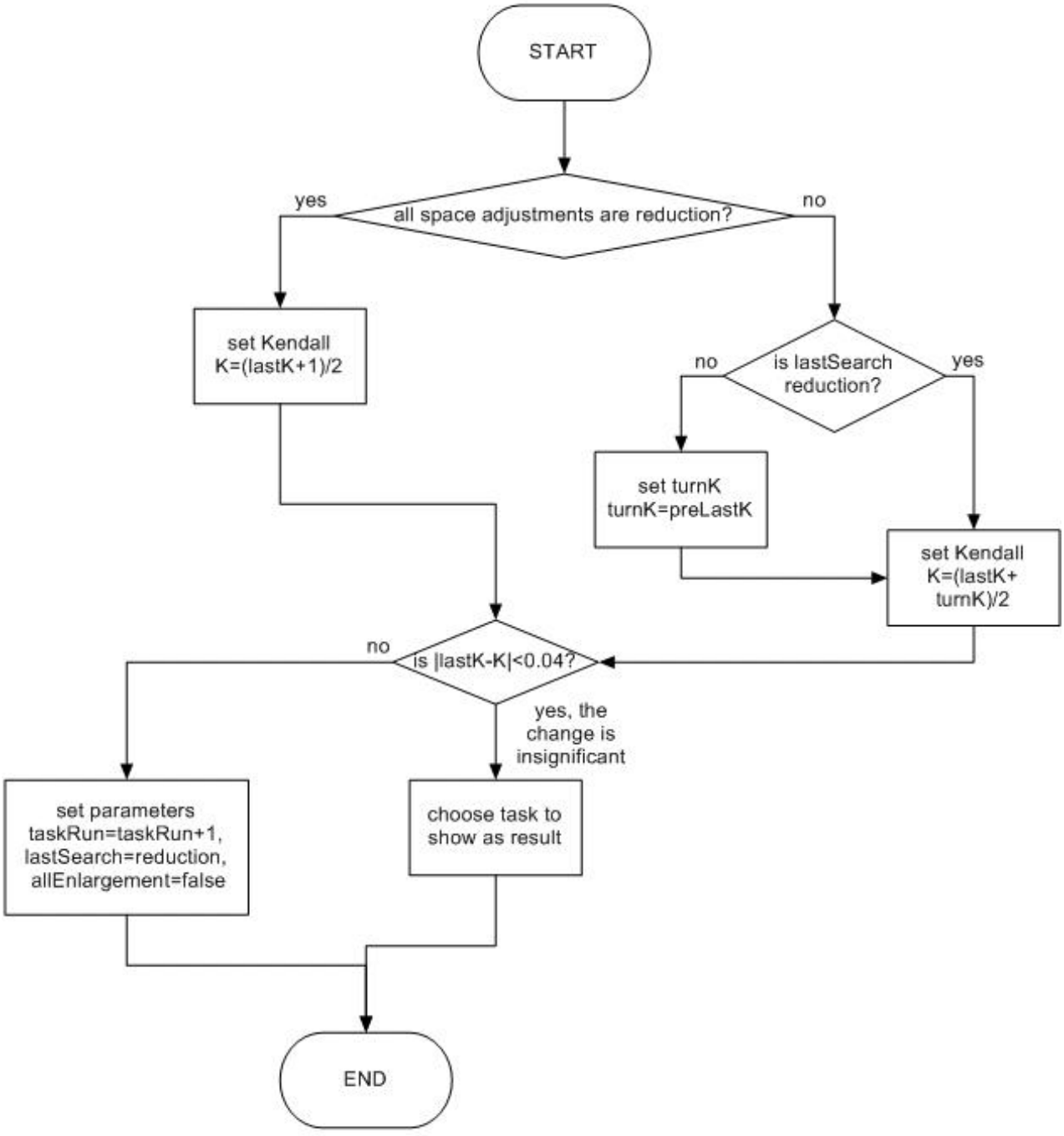
If false, it means that the difference between the previous and adjusted task is small and there will probably be only a little change in results, so the script will not run the adjusted task and will choose some of the previous tasks' hypotheses to be recorded in the final report. This way the script ensures that the *Kendall threshold value* is not increased forever and that the new tasks can produce significantly better results.

On the other hand, if the difference between previous and new *Kendall threshold value* is higher than 0.04, then the parameters are changed to fit for the newly created task and the task is ready to be run.

This sub-process also needs to update its parameters. The parameters *preLastK* and *lastK* as well as – if it is the first time a task has been run – the parameters *allEnlargement* and *allReduction* are changed at the beginning of the process, when deciding which adjustment to use so that the enlargement or reduction process can work with the updated parameter values. All other parame-

ters, including the parameters *allEnlargement* and *allReduction* when it is not the first time a task has been run, are then changed after calculating the new K parameter value.

The whole diagram of this sub-process is shown below, in the *Picture number 12, Reduction of a search space*.



Picture number 12, Reduction of a search space

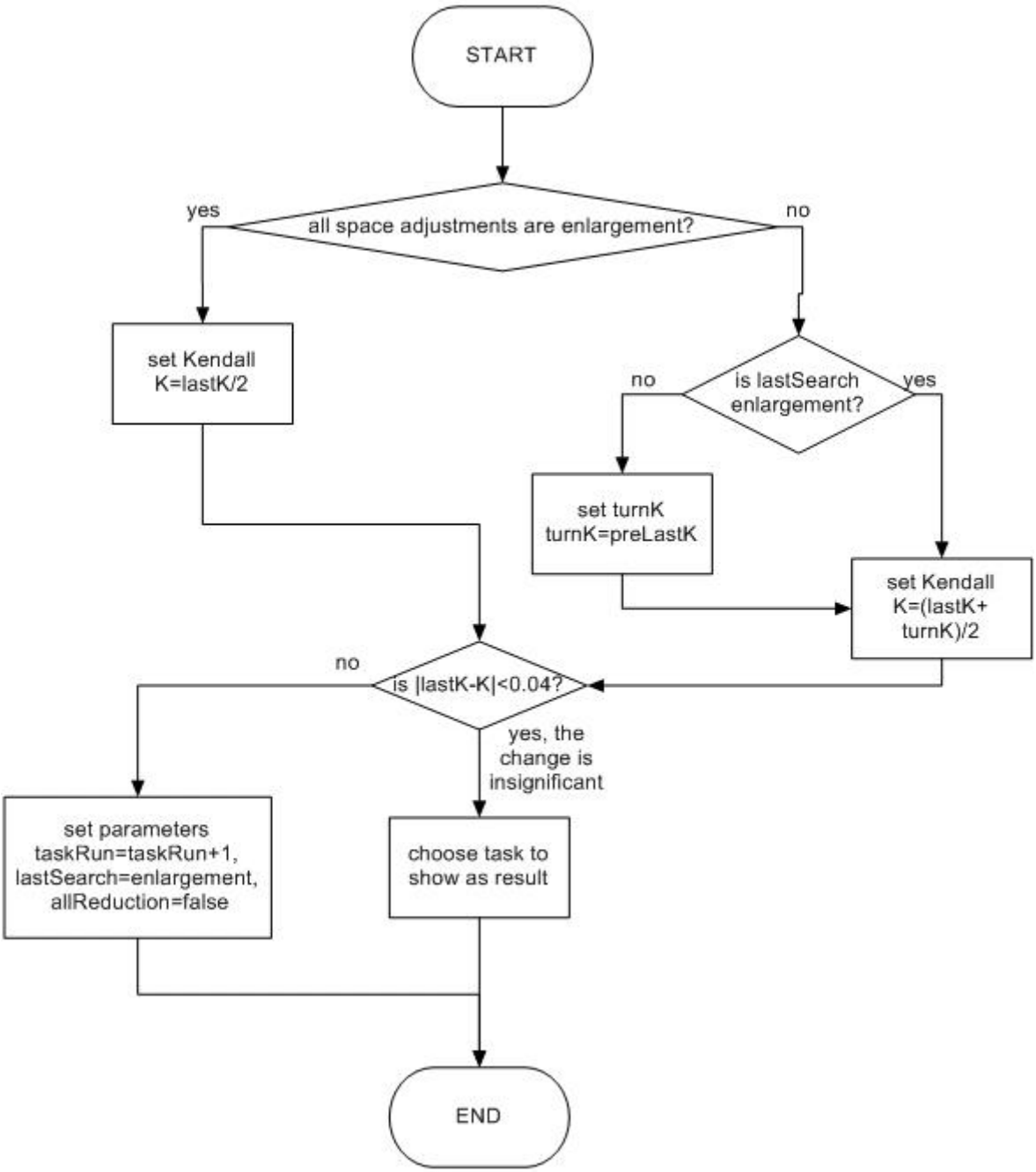
8.3.2 Enlargement of a search space

This sub-process is very similar to the sub-process in previous chapter (*8.3.1 Reduction of a search space*). There are only two differences:

If all the previous search space adjustments were enlargements, the calculation for the new *Kendall threshold value* is previous *Kendall threshold value* divided by 2.

If all the previous search space adjustments were not enlargements, then the next step asks if the previous search space adjustment was enlargement.

The diagram of this process can be seen below, in the *Picture number 13, Enlargement of a search space*.

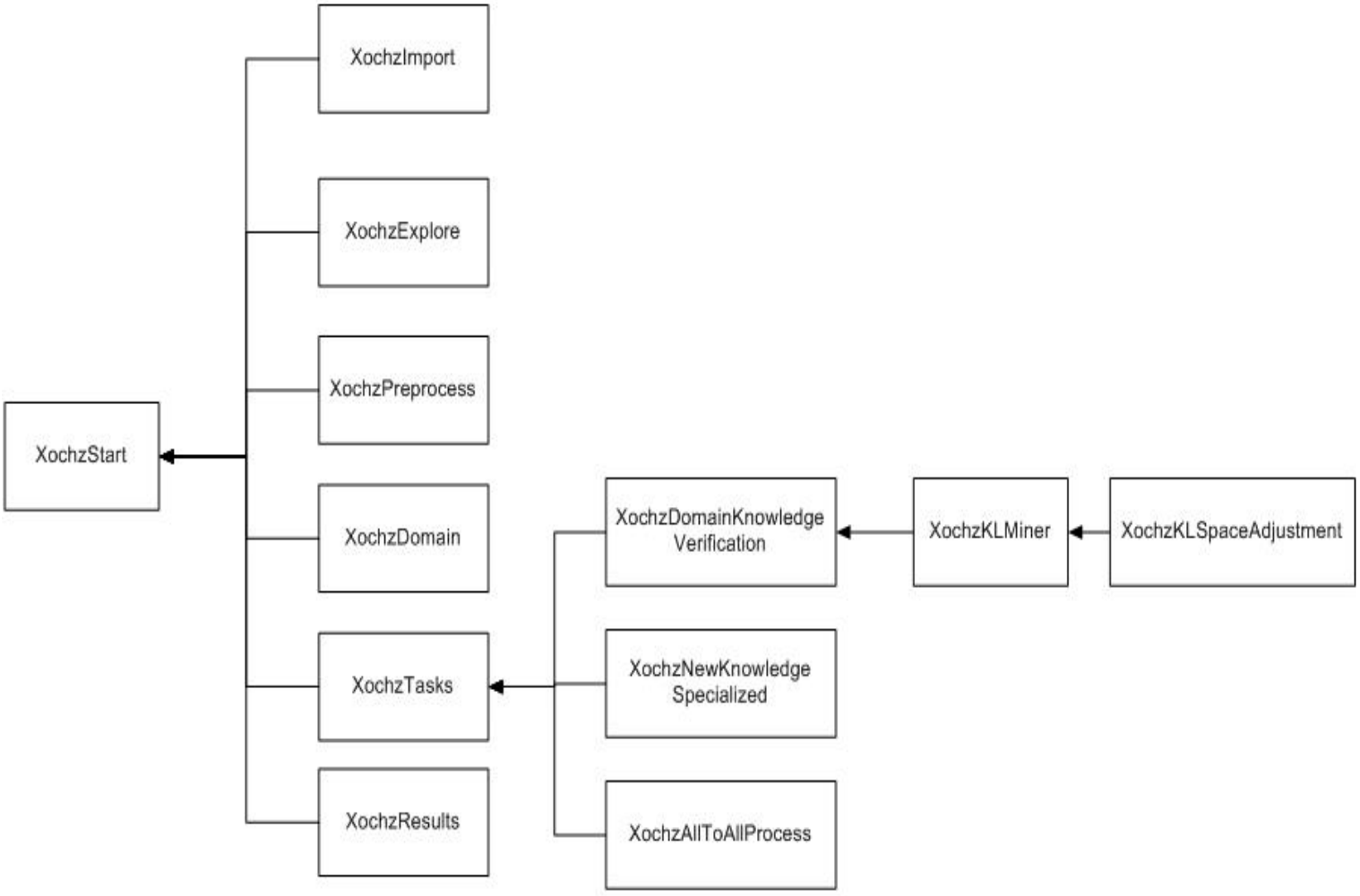


Picture number 13, Enlargement of a search space

9 Implementation

This chapter describes the implementation part of the automated data mining process created in this thesis – the verification of set domain knowledge of attribute dependency type *influence*. The numbered paragraphs below contain description of each implemented module.

The implementation consists of 12 modules, each stored in a separate Lua text file. The connections between the modules may be seen below, in the *Picture number 14, Implementation modules*. All the modules were declared in the first module, *XochzStart*. The script should be run through LMExec.exe file and the user needs to open and run only the *XochzStart* module.



Picture number 14, Implementation modules

The implementation uses full names convention, so that the names of the parameters and functions describe what the parameter of function does. This convention might, however, create longer names, but when creating the script, the understandability was more important.

All the created scripts are available in the appendix part as well as on the enclosed CD.

9.1 XochzStart

The *XochzStart* module is the base module. All the main methods from modules are called one by one after completing previous method. The main methods are called as follows:

xochz.import.creation(inputParams); --Import of a data TXT file and metabase creation

xochz.explore.exploration(); --Exploring data

xochz.preprocess.createAttributes(inputParams); --Data preprocessing

xochz.domain.createDomain(inputParams); --Domain knowledge declaration

xochz.tasks.tasksprocess(inputParams); --Tasks creation, run and search space adjustments

xochz.results.exportReport(inputParams); --Report creation

Additionally, input parameters are initialized in this module.

At the end, a metabase is updated and closed and the script prints a log line to inform that it has ended.

9.2 XochzImport

The *XochzImport* module is responsible for importing the data from a text file with the command lm.data.importTXT(importParams). This command also creates an MDB database from the text file. If the data import was successful, the next command lm.metabase.createAndAssociateWithDataMDB({}) creates a metabase and associates it with the previously created MDB database. Next, the metabase is opened, updated and closed and a back-up copy of the metabase is created to be used for the next module.

9.3 XochzExplore

The *XochzExplore* module's only method xochz.explore.exploration() called in the *XochzStart* module, opens a metabase that was created in the *XochzImport* module, initializes all data tables in the used data (in the data used for this thesis it was only one data table) and sets the primary key, if it has not been already set. Then the script informs about the number of rows in each data table, updates the metabase, closes the metabase and creates a back-up copy with the updated content.

9.4 XochzPreprocess

The *XochzPreprocess* module is responsible for attribute groups, attributes and categories creation. Its only method xochz.preprocess.createAttributes(inputParams) is called in the *XochzStart* module and it opens a metabase copy that was created in the *XochzExplore* module. Then, the method creates attribute groups with names defined in input parameters in the *XochzStart* module. Next, for each data table (in the data used in this thesis, for only one data table) all the columns are found and paired with their attribute group. If no attribute group is declared for an attribute, the attribute will be paired with the root attribute group. Moreover, all date and time attributes were placed in the root attribute group and not used in the analyses. This decision has been made because of the used laptop low performance (see the description in the chapter *10 Testing*) and the fact that data preparation is not in the focus of this thesis, so not including these attributes does not interfere with fulfilment of this thesis' goals.

Then, if an attribute group called *mainGroup* has been used, a parameter called *isMainAttributeGroupUsed* is set to 1, so that the *XochzTasks* module used later will know, that this attribute group is not empty, which will help decide which branch to use in the *XochzTasks* module (for further information why it is needed, see the design part of the script in the chapter number 7 *Tasks design*). Finally, attributes' categories are created as described earlier in the chapter number *6.3 Preprocess*. Then the method updates the metabase, closes it and creates a back-up copy with the updated content.

9.5 XochzDomain

This module's only method *xochz.domain.createDomain(inputParams)* declares dependency relations between attributes as designed in the chapter number *6.4 Domain*. The attributes that

are in the relations are manually listed in the script, since the domain knowledge declaration is not in the primary focus of this thesis. At the end, the method updates the metabase, closes it and creates a back-up copy with the updated content.

9.6 XochzTasks

The *XochzTasks* module determines which of the modules for tasks creation will be run. According to the design described in the chapter *6.5 Tasks*, if there are no attributes in the main attribute group, the *XochzAllToAll* module's method will be called. If there are some attributes that belong to the main attribute group and there is no declared domain knowledge, then the method from the *NewKnowledgeSpecialized* module will be called. Finally, if the main attribute group is not empty and some domain knowledge has been declared in the *XochzDomain* module, then the module *XochzDomainKnowledgeVerification*'s script will be executed and then the *XochzNewKnowledgeSpecialized* module as well.

At the end of the *XochzTasks* script, there is a method *lm.sleep()* called, so that the script execution can wait for the xxPooler to close. There is also another way possible – forced shut down of the xxPooler.

Finally, the method updates the metabase, closes it and creates a back-up copy with the updated content.

9.7 XochzDomainKnowledgeVerification

The *XochzDomainKnowledgeVerification* module decides which LISp-Miner procedure will be used to verify the set domain knowledge relations. The procedure to run will be determined according to the design in the chapter number *7.1 Domain knowledge verification*.

If the domain knowledge dependency type is *influence*, the KL-Miner procedure with and without condition will be run for each declared pair of attributes with this dependency type, if the dependency is *frequency*, then the CF-Miner procedure will be run and if the dependency type is *Boolean*, then the 4ft-Miner procedure will be run. The dependency types *unknown* and *notSet* will trigger all the three procedures in order to identify the dependency and other dependency types will skip the domain knowledge verification and start the *XochzNewKnowledgeSpecialized* module methods.

9.8 XochzKLMiner

There are two functions in the *XochzKLMiner* module. The first one, *xochz.klMiner.DomainKnowledgeVerification(rowAttribute,columnAttribute)* creates a KL-Miner task and sets its attributes and quantifiers without using any condition and the other one, *xochz.klMiner.DomainKnowledgeVerificationCondition(rowAttribute,columnAttribute,inputParams)* creates a KL-Miner task and sets its attributes and quantifiers with a condition.

The function *xochz.klMiner.DomainKnowledgeVerification(rowAttribute,columnAttribute)* firstly created a new KL-Miner task, then sets the KL-Miner quantifier to Kendall's coefficient and sets its threshold value to absolute value of 0,1. (Reasons setting the KL-Miner task this way are explained in the chapter number *8.1 Attributes setting*. Then the partial cedents are declared as cedent types KLAttributeCol and KLAttributeRow and an empty condition is declared as well. Finally, the function declares column and row attributes, runs the task and places it into the finalTask group for report purposes.

Xochz.klMiner.DomainKnowledgeVerificationCondition(rowAttribute,columnAttribute,inputParams) if the second function used in this module and it is very similar to the previous one, apart from a few differences.

One of the differences is that it uses condition, so that the condition is not empty and contains attributes. These attributes are from all attribute groups except the main attribute group and the root attribute group. All conditions created in LMCL are 4ft-cendents, so they must be set through classes FTPartialCedentSetting and FTLiteralSetting.

The second difference is that there are initialized parameters that will be needed for the *XochzKLSpaceAdjustment* module. The initial values of these parameters are described in the chapter number *8.3 Domain knowledge verification search space adjustments*. Moreover, after the created KL-Miner task with condition is run, the process of search space adjustment of *XochzKLSpaceAdjustment* module is triggered to reach ideal number of found hypotheses.

9.9 XochzKLSpaceAdjustment

The *XochzKLSpaceAdjustment* module starts after a KL-Miner task with condition has been run. It contains 3 functions:

Xochz.KLspaceAdjustment.run(taskDomainKnowledgeKLCondition,KLAdjustmentParams) function determines whether to enlarge or reduce the search space according the number of hypothe-

ses found in the last run of the task. If there are less hypotheses found then the initially set ideal number, then the parameters containing information about the last and prelast value of Kendall's coefficient are updated and if it is the first time the task has been run, also the parameters containing information about whether all the previous search space adjustments were only reductions or only enlargements are updated. Then the space search enlargement function is run. If there are more hypotheses found then the initially set ideal number, the process is the same, but at the end the search space reduction function is triggered. If the number of found hypotheses fits the initially set ideal number, the lastly run task is added to the final task group for report creation and the search space adjustment for this task is ended.

The actual search space adjustment functions calculate new Kendall quantifier threshold value (more details in the design part in the chapters number *8.3.1 Reduction of a search space* and *8.3.2 Enlargement of a search space*). If the difference between the newly created Kendall quantifier threshold value and the last Kendall quantifier threshold value used for the same task is insignificant, the search space adjustment process is ended for this task and the lastly run task is used in the final report. If the difference is significant, then the function parameters are updated a clone of the lastly run task is created and set with the new Kendall quantifier threshold value. Then this task is run and the search space adjustment process is run again.

9.10 XochzNewKnowledgeSpecialized and XochzAllToAllProcess

These modules have not yet been implemented and their implementation is out of this thesis focus. However, the proposed design of these modules can be seen in chapters *7.2 New knowledge search-specialized process* and *7.3 New knowledge search- AllToAll process.*

9.11 XochzResults

For the *XochzResults* module, a module from the EverMiner Simple Demo (see chapter *3.6 EverMinerSimple demo*), the *EMSResults* module has been used with only some small changes. This module creates the final report in form of an HTML file.

The changes to the *EMSResults* module were the use the final task group as the task group whose tasks are reported and changing the name of the initially set ideal number of hypotheses.

9.12 Metabase back-ups

The script uses a system for creating back-up copies of the metabases after running any module that is linked to the *XochzStart* module. This system was created in EverMiner Simple Demo (Šimůnek, 2014). Each of the modules linked to the *XochzStart* module starts with opening a copy of metabase that was created in the previously run module. This way it is ensured, that the right metabase will be used in the right module, even without running the process more time without deleting the already created metabase copies.

10 Testing

The testing was constructed to test the time needed for various amounts of data and to search for any abnormalities that may occur during real use of the application. The testing also investigated how the number of total iterations and the number of final hypotheses change with a dataset size.

The testing was performed on laptop Lenovo SL500, with OS Windows XP and processor Intel® Core™2 Duo CPU T6670 (2.20 GHz).

There were 4 dataset sizes, each of them tested in 4 iterations to check for differences in total run time and created reports (see *Picture number 15, the testing table*).

size of data	total number of iterations	1.	2.	3.	4.	number of found hypotheses without condition	number of found hypotheses with condition	comments
140 KB	15	3m 21s	3m 22s	3m 41s	3m 27s	4	48	same reports
280 KB	19	4m 18s	4m 23s	4m 11s	4m 20s	3	52	same reports
560 KB	19	4m 33s	4m 49s	4m 26s	4m 26s	3	49	same reports
1121 KB	14	4m 52s	4m 32s	4m 32s	4m 23s	3	49	same reports

Picture number 15, the testing table

10.1 Total number of iterations

The total number of iterations is a number calculated only for the KL-Miner Knowledge verification with condition process because only there are search space adjustments used. The number represents how many search space adjustments were made in the process. The smaller number the better because each adjustment is time consuming. Surprisingly, the largest dataset size used the smallest number of iterations. However, this indicator is very data specific since the changing parameter – Kendall's threshold value (see chapter *3.1 KL-Miner*) – is always set the same way.

10.2 Run time

The run time was measured 4 times in 4 repetitions. For the same dataset size, the run time seems to be more or less the same. Small changes in the times might be caused by background processes of the laptop on which the tests were run (despite efforts to eliminate them). However, there seem to be no huge changes in the run times between the smallest dataset and 8 times larger dataset. This might be caused by the fact that even the largest used dataset is in data mining terms very small, so the changes of run time might not be very significant for a data mining tool that is designed to work with larger datasets.

10.3 Number of found hypotheses without condition

The number of found hypotheses without condition was different in the smallest dataset. This is probably a consequence of the data sample – the sampled smallest dataset probably contained data supporting more hypotheses.

10.4 Number of found hypotheses with condition

The ideal number of found hypotheses was set to be 10. So, ideally, each run task would provide 10 found hypotheses. Because there were 5 domain knowledge relations set (which means 5 domain knowledge verification tasks), each tested dataset should ideally provide 50 hypotheses. *Picture number 15, the testing table* shows that the results from all tested datasets provided results very close to 50. With the number of found hypotheses being the primary indicator for choosing the right hypotheses in this thesis, one could admit that the script accomplished its role sufficiently.

10.5 Comments

No problems occurred during the testing, final reports were the same in each testing run for the same dataset size and logs proved that the space search adjustments worked as they were supposed to. An example log file of one of the runs can be seen in Appendix.

11 Conclusion

This thesis had four goals:

1) Literature search of the automated data mining area
2) Gain of practical knowledge of the LISp-Miner system and the LMCL scripting language
3) Creation of a design of an automated data mining tasks creation process for verification of set domain knowledge and new knowledge search
4) Implementation of verification of set domain knowledge of attribute dependency type *influence*

Assessment and results of each of these goals are described in the following paragraphs.

11.1 Literature search of the automated data mining area

The literature search is described in the chapter number 2 *Automated data mining*. Overall, it can be said that in spite of the debate if it is even possible to automate the data mining process, there are many implemented automated, or at least semi-automated tools capable of advanced data analyses. Most of them are, however, designed for very specific tasks, but there are a few systems that claim to be able to perform automated data mining with vast range of models on various data domains.

On the other hand, some found articles with automated data mining in their title proved to be only regular data mining manual projects. This could happen due to wrong used terminology or due to an effort to make the articles more interesting.

Moreover, there was found no literature with automated data mining with usage of environmental data.

To sum up, the literature search in this thesis shows that automated data mining is not just an idea, but there are many successful implementations for various data mining analyses providing many benefits.

11.2 Gain of practical knowledge of the LISp-Miner system and the LMCL scripting language

Practical knowledge of the LISp-Miner system can be seen throughout the chapters *5 Automation assignment* to *9 Implementation*. It was necessary to understand the functions and procedures of the system to design the overall process of automated data mining created in this thesis (see chapter 6 *Overall design*). Moreover, in order to use the LMCL language, it is also necessary to have practical knowledge of the system. The proof of practical knowledge of the LMCL language can be seen in chapter 9 *Implementation* as well as in the scripts in Appendix and the enclosed CD with created LMCL modules.

Overall, the LMCL language seems to be very efficient tool for automating data mining steps in the LISp-Miner system. The language has good documentation and is quite capable of using the system's functionality. Moreover, for example comparing to the R language, the LMCL language is more user friendly and there is no need to study many packages and new syntax. The LMCL syntax is quite intuitive and the user should only have some knowledge of the LISp-Miner functions prior using the LMCL language.

11.3 Creation of a design of an automated data mining tasks creation process for verification of set domain knowledge and new knowledge search

The creation of both designs is described in chapters from 6 *Overall design* to 8 *Domain knowledge verification*. The designs are based on the manual experience with the LISp-Miner system as well as the knowledge of the used data domain and the system's functionality. The processes were designed in order to find hypotheses that the users would be interested in in a timely manner.

Generally, there were designed three processes for tasks creation: domain knowledge verification, new knowledge specialized process and new knowledge AllToAll process (see chapter 7 *Tasks design*). These processes are capable of finding new interesting knowledge and verifying set domain knowledge relations between attributes.

11.4 Implementation of verification of set domain knowledge of attribute dependency type *influence*

The implementation of the verification proved to be successful together with the implemented search space adjustments (see chapters from 8 *Domain knowledge verification* to *10 Testing*). The testing proved that the scripts run as they were supposed to and there were no major issues during the run. The implementation tried to be more general, not data specific, so that another data (with prior manual changes in the initial parameters) could be used for this implemented algorithms as well.

11.5 General conclusion

This thesis contribution lies in the description of the automated data mining area, in design of an automated data mining process using the LISp-Miner system functionality and implementing a part of this design with the LMCL language.

This thesis is one of the first implementations using the LMCL language together with the EverMinerSimple demo (see chapter *3.6 EverMinerSimple demo*). The implementation contains an automatic tasks creation process for the used data as well as a search space adjustment algorithm to automatically change the setting of a task.

There is a lot of space for further development, firstly, implementing the rest of the designed processes in order to create a complex tool. Secondly, there is a possibility for creation of different tasks creation scenarios as well as different search space adjustment processes. Any of these further developments would contribute to establish an improved version of the tool developed in this thesis.

To conclude, this thesis is another attempt to make the data mining process more automated. The benefits the automation brings are significant and for this reason (if the attempts will continue) there might be one day a fully automated universal data mining tool available for both experts and non-experts.

12 References

1. **Bhattacharya, 2010.** BHATTACHARYA, Pratik, Renee VAN STAVERN a Ramesh MADHAVAN. Automated Data Mining: An Innovative and Efficient Web-Based Approach to Maintaining Resident Case Logs. In: Journal of Graduate Medical Education. 2010, s. 566-570. ISSN 1949-8349. DOI: 10.4300/JGME-D-10-00025.1. Available from: http://www.jgme.org/doi/abs/10.4300/JGME-D-10-00025.1

2. **Burget et al., 2010.** BURGET, Radim, Martin ZUKAL, Václav UHER a Jan KARÁSEK. Semi-Automatic Image Data Analysis. Elektrorevue [online]. 2010 [cit. 2014-12-11]. Available from: http://www.elektrorevue.cz/en/articles/analogue-technics/0/semi-automatic-image-data-analysis/

3. **Campos et al., 2010.** CAMPOS, Marcos M, Peter J. STENGARD a Boriana L. MILENOVA. Data-Centric Automated Data Mining. In: Oracle.com [online]. 2010 [cit. 2014-12-11]. Available from: http://www.oracle.com/technetwork/testcontent/automated-data-mining-paper-1205-128874.pdf

4. **Cappendijk et al., 2013.** CAPPENDIJK, Susanne L. T., Geoffery L. MILLER, Patrick L. YOUNT a Robert A. Van ENGELEN. Automatic data analysis of real-time song and locomotor activity in zebra finches. In: International Journal of Bioinformatics Research and Applications. 2013, s. 91-108. ISSN 1744-5485. DOI: 10.1504/IJBRA.2013.050652. Available from: http://www.inderscience.com/link.php?id=50652

5. **Coppock, 2002.** COPPOCK, David S. Data Mining Automation. Information Management [online]. 25. 01. 2002 [cit. 2014-11-29]. Available from: http://www.information-management.com/news/4584-1.html

6. **Dill et al., 2004.** DILL, Marcus, Harish MAHABAL a Jens WEIDNER. Automated data mining runs [patent]. United States. US 2004/0215656 A1. 28. 10. 2004. Available from: http://www.google.com/patents/US20040215656

7. **Download, 2014.** Download. LISp-Miner [online]. 2014 [cit. 2014-12-04]. Available from: http://lispminer.vse.cz/download/index.php

8. **Franks et al., 2010.** FRANKS, Bill a Scott VANVALKENBURGH. When Automating Analytics Works- And When It Doesn't. Information Management [online]. 07. 10. 2010

[cit. 2014-11-29]. Available from: http://www.information-management.com/infodirect/2009_179/automation_analytics-10018899-1.html

9. **Frederiksen et al., 2014.** FREDERIKSEN, Juliet, Marcus BUGGERT, Annika C. KARLSSON a Ole LUND. NetFCM: A semi-automated web-based method for flow cytometry data analysis. In: Cytometry Part A. International Society for Advancement of Cytometry, 2014, s. 969-977. ISSN 15524922. DOI: 10.1002/cyto.a.22510. Available from: http://doi.wiley.com/10.1002/cyto.a.22510

10. **Hájek, 2002.** HÁJEK, Petr. Metoda GUHA - současný stav. In: JČMF, Zorganizované odbornou skupinou pro výpočetní statistiku při MVS a Gejza Dohnal USPOŘÁDALI JAROMÍR ANTOCH. ROBUST 2002: sborník prací dvanácté zimní školy JČMF ve dnech 21. -25. ledna 2002 v Hejnicích. V Praze: Jednota českých matematiků a fyziků, 2002, s. 133-135. ISBN 80-7015-900-6

11. **Hideyuki et al., 2001.** MAKI, Hideyuki a Yuko TERANISHI. Development of Automated Data Mining System for Quality Control in Manufacturing. In: Data Warehousing and Knowledge Discovery. Munich, Germany: Springer Berlin Heidelberg, 2001, s. 93-100. ISBN 978-3-540-44801-3. DOI: 10.1007/3-540-44801-2_10. Available from: http://link.springer.com/10.1007/3-540-44801-2_10

12. **Hofmann et al., 2003.** HOFMANN, Markus a Brendan TIERNEY. The involvement of human resources in large scale data mining projects. In: ISICT '03 Proceedings of the 1st international symposium on Information and communication technologies. Dublin: Trinity College Dublin, 2003, s. 103-109.

13. **IBM Business Analytics, 2013.** IBM BUSINESS ANALYTICS. IBM SPSS Analytic Catalyst. In: YouTube [online]. 2013 [cit. 2014-12-11]. Available from: https://www.youtube.com/watch?v=GtlgE_RSuP4

14. **IBM, 2014.** SPSS Analytic Catalyst. IBM [online]. 2014 [cit. 2014-12-11]. Available from: http://www-01.ibm.com/software/analytics/solutions/big-data/predictive-analytics/analytic-catalyst/

15. **KDNuggets, 2011.** What main methodology are you using for your analytics, data mining, or data science projects? Poll. KDnuggets [online]. 2014 [cit. 2014-11-29]. Available from: http://www.kdnuggets.com/polls/2014/analytics-data-mining-data-science-methodology.html

16. **KDNuggets, 2013.** Automated Data Scientist? IBM SPSS Analytic Catalyst. KDnuggets [online]. 2013 [cit. 2014-12-11]. Available from:

http://www.kdnuggets.com/2013/06/automated-statistician-ibm-spss-analytic-catalyst.html

17. **LISp-Miner Control Language reference, 2014.** LISp-Miner Control Language Reference. LISp-Miner [online]. 2014 [cit. 2014-12-04]. Available from: http://lispminer.vse.cz/lmcl/index.html

18. **LISp-Miner, 2014.** LISp-Miner: The official site of the LISp-Miner project [online]. 2014 [cit. 2014-11-29]. Available from: http://lispminer.vse.cz/index.html

19. **Lua, 2014.** Lua [online]. 2014 [cit. 2014-12-11]. Available from: http://www.lua.org/

20. **Moser et al., 2005.** MOSER, W. K., M.H. HANSEN, P. MILES, B. JOHNSON a R. MCROBERTS. The Virtual Analyst Program: A Small Scale Data-Mining, Error-Analysis and Reporting Function. In: 16th International Workshop on Database and Expert Systems Applications (DEXA'05). Washington DC: IEEE, 2005, s. 691-695. ISBN 0-7695-2424-9ISSN 1529-4188. DOI: 10.1109/DEXA.2005.185. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1508353

21. **Neeli et al., 2008.** NEELI, Sandeep, Kannan GOVINDASAMY, Bogdan M. WILAMOWSKI a Aleksander MALINOWSKI. Automated Data Mining from Web Servers Using Perl Script. In: 2008 International Conference on Intelligent Engineering Systems. Miami, Florida: IEEE, 2008, s. 191-196. ISBN 978-1-4244-2082-7. DOI: 10.1109/INES.2008.4481293. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4481293

22. **Nesbitt, 2003.** NESBITT, Keith Vincent. Automated and Perceptual Data Mining of Stock Market Data. In: ANZIIS 2003, Proceedings of the Eight Australian and New Zealand Intelligent Information Systems Conference. Brisbane, Australia: Queensland University of Technology, 2003, s. 145-150. ISBN 1741970392.

23. **Ninja Metrics, 2014.** Automated Predictive Analytics. Ninja Metrics [online]. 2014 [cit. 2014-12-11]. Available from: http://www.ninjametrics.com/automated-predictive-analytics

24. **Ochodnicka, 2012.** OCHODNICKA, Zuzana. Analýza reálnych dat CRM pomocí systému LISp-Miner. Praha, 2012. Bachelor thesis. Vysoká škola ekonomická v Praze

25. **Raschka, 2014.** RASCHKA, Sebastian. Implementing a Principal Component Analysis (PCA) in Python step by step. Sebastian Raschka [online]. April 13, 2014 [cit. 2014-12-04]. Available from: http://sebastianraschka.com/Articles/2014_pca_step_by_step.html

26. **Rauch, 2013.** RAUCH, Jan. Observational calculi and association rules. Heidelberg: Springer-Verlag, 2013. ISBN 978-364-2117-374

27. **Saitta, 2013.** SAITTA, Sandro. Can we automate data mining?. Data Mining Research [online]. 14. 04. 2013 [cit. 2014-11-29]. Available from: http://www.dataminingblog.com/can-we-automate-data-mining/

28. **Šimůnek et al., 2011.** ŠIMŮNEK, M. a J. RAUCH. EverMiner - towards Fully Automated KDD Process. In: New Fundamental Technologies in Data Mining. InTech, 2011-01-21, s. 221-240. ISBN 978-953-307-547-1. DOI: 10.5772/13998. Available from: http://www.intechopen.com/books/new-fundamental-technologies-in-data-mining/everminer-towards-fully-automated-kdd-process

29. **Šimůnek, 2010.** ŠIMŮNEK, Milan. Systém LISp-Miner: akademický systém pro dobývání znalostí z databází : historie vývoje a popis ovládání. Vyd. 1. Praha: Oeconomica, 2010, 106 s. ISBN 978-80-245-1699-8

30. **Šimůnek, 2014.** ŠIMŮNEK, Milan. LISp-Miner Control Language description of scripting language implementation. In: Journal of Systems Integration. 2014, s. 28-42. ISSN 1804-2724.

31. **Smart Vision Europe, 2011.** What is the CRISP-DM methodology?. Smart Vision Europe [online]. 2011 [cit. 2014-11-29]. Available from: http://www.sv-europe.com/crisp-dm-methodology/

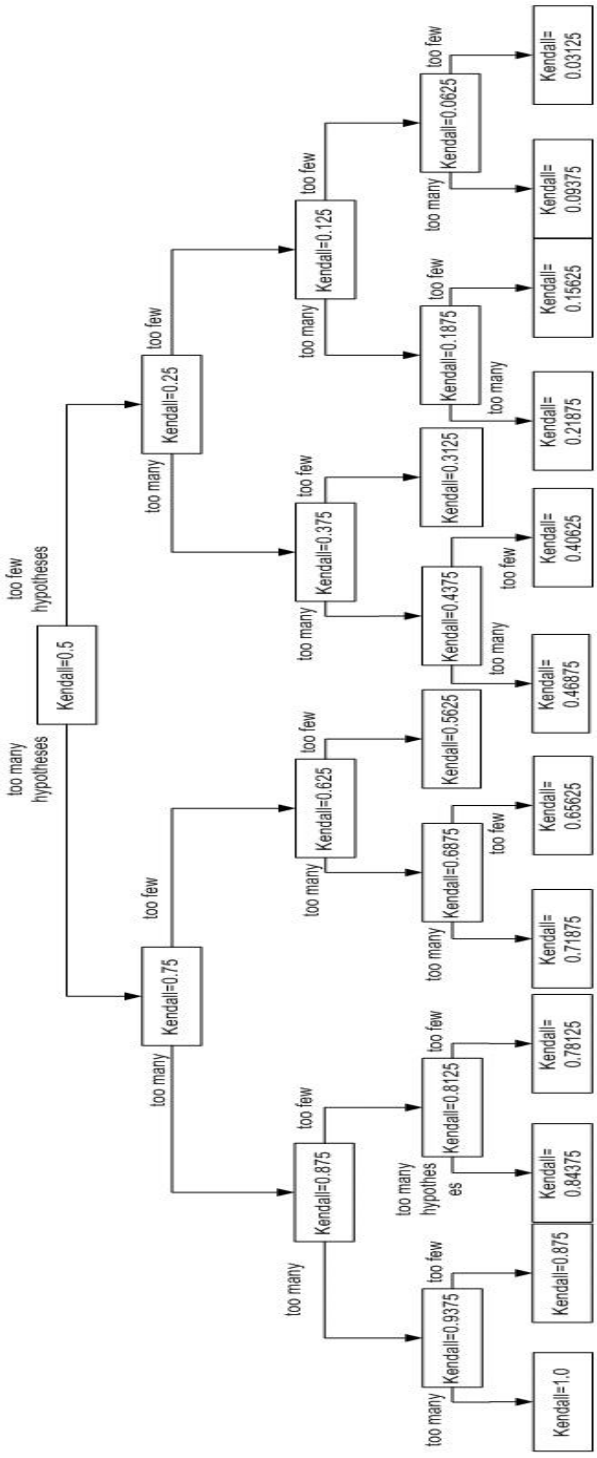
32. **Vishnubhotla, 2003.** VISHNUBHOTLA, Prasad R. Method and system for data mining automation in domain-specific analytic applications [patent]. United States. US 6636860 B2. 21. 10. 2003. Available from: http://www.google.com/patents/US6636860

33. **Vishnubhotla, 2004.** VISHNUBHOTLA, Prasad R. Method and system for simplifying the use of data mining in domain-specific analytic applications by packaging predefined data mining models [patent]. United States. US 6820089 B2. 16. 11. 2004.

34. **Wikipedia, 2014.** Kendall tau rank correlation coefficient. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2014-12-04]. Available from: http://en.wikipedia.org/wiki/Kendall_tau_rank_correlation_coefficient

35. **Yang et al., 2006.** YANG, QIANG a XINDONG WU. 10 CHALLENGING PROBLEMS IN DATA MINING RESEARCH. In: International Journal of Information Technology [online]. 2006, s. 597-604 [cit. 2014-11-29]. ISSN 0219-6220. DOI: 10.1142/S0219622006002258. Available from: http://www.worldscientific.com/doi/abs/10.1142/S0219622006002258

Appendix 1- KL space adjustment concept



Picture number 16, KL-Miner with condition space adjustment concept

Appendix 2, Execution log

```
ScriptExec - start C:\diplo\automation_aktual\start.lua
  LISp-Miner version: 24.18.00 of 23 Nov 2014
  Lua interpreter version: Lua 5.2
  TimeStamp: 24.11.2014 22:51:48
LMLuaBind registration
Load file C:\diplo\automation_aktual\start.lua
Script execution (pcall)
  -->Input parameters recorded
  -->Start of data import and metabase creation
  lm.data.importTXT
     pathNameSrc= 'C:\diplo\automation_aktual\CompletData2.txt'
     pathNameDest=
'C:\diplo\automation_aktual\CompletData2.DB.mdb'
     tableName= 'Meranie'
  -->Data import succesful
  lm.metabase.createAndAssociateWithDataMDB
     pathNameMetabase=
'C:\diplo\automation_aktual\CompletData2.LM.mdb'
    lm.metabase.associateWithDataMDB
       pathNameMetabase=
'C:\diplo\automation_aktual\CompletData2.LM.mdb'
       pathNameData=
'C:\diplo\automation_aktual\CompletData2.DB.mdb'
       dsnBase= 'Exec Ochodnicka CD2'
  Data and metabase succesfuly associated.
  lm.metabase.open
     dataSourceName= 'LM Exec Ochodnicka CD2 MB'
  -->Metabase was opened.
  lm.metabase.updateMetadata
  -->Metabase was updated.
  lm.metabase.close
  lm.metabase.backupMDB
     pathNameSrc=
'C:\diplo\automation_aktual\CompletData2.LM.mdb'
     pathNameDest=
'C:\diplo\automation_aktual\CompletData2.LM.mdb_bkup.import.
mdb'
  -->Start of exploration
  lm.metabase.restoreMDB
     pathNameSrc=
'C:\diplo\automation_aktual\CompletData2.LM.mdb_bkup.import.
mdb'
     pathNameDest=
'C:\diplo\automation_aktual\CompletData2.LM.mdb'
  lm.metabase.open
     dataSourceName= 'LM Exec Ochodnicka CD2 MB'
  -->Array of data tables prepared
  -->Initializing data table Meranie
  lm.explore.DataTable.init: 'Meranie' (DataTable, ID= 1)
     columnName= 'ID_LM'
  -->Number of records in the table Meranie is 17519
  lm.metabase.updateMetadata
  -->Primary key has been defined
  lm.metabase.close
  lm.metabase.backupMDB
     pathNameSrc=
'C:\diplo\automation_aktual\CompletData2.LM.mdb'
     pathNameDest=
'C:\diplo\automation_aktual\CompletData2.LM.mdb_bkup.explore
.mdb'
  -->Start of data preprocessing
  lm.metabase.restoreMDB
     pathNameSrc=
'C:\diplo\automation_aktual\CompletData2.LM.mdb_bkup.explore
.mdb'
     pathNameDest=
'C:\diplo\automation_aktual\CompletData2.LM.mdb'
  lm.metabase.open
     dataSourceName= 'LM Exec Ochodnicka CD2 MB'
  lm.metabase.clearLocalDataCache
  -->Starting attribute groups creation
  lm.prepro.AttributeGroup.create
     name= 'descriptiveGroup1'
     pParentGroup= 'Root group of attributes' (AttributeGroup, ID=
1)
  -->Attribute group named descriptiveGroup1 was created.
  lm.prepro.AttributeGroup.create
     name= 'mainGroup'
     pParentGroup= 'Root group of attributes' (AttributeGroup, ID=
1)
  -->Attribute group named mainGroup was created.
  lm.prepro.AttributeGroup.create
     name= 'time_place'
     pParentGroup= 'Root group of attributes' (AttributeGroup, ID=
1)
  -->Attribute group named time_place was created.
  -->All attribute groups created
  -->Start of attributes creation.
  -->Creating attributes for the table Meranie
     name= 'descriptiveGroup1'
  -->I have found a column named BIL with data type Decimal
number
  lm.explore.DataColumn.getDistinctValueCount: 'BIL' (DataCol-
umn, ID= 11)
  lm.prepro.Attribute.create
     name= 'BIL_equidistant'
     pAttributeGroup= 'descriptiveGroup1' (AttributeGroup, ID= 2)
     pDataColumn= 'BIL' (DataColumn, ID= 11)
  lm.prepro.Attribute.autoCreateIntervalEquidistant:
'BIL_equidistant' (Attribute, ID= 1)
     nCount= 30
  lm.prepro.Attribute.create
     name= 'BIL_equifrequent'
     pAttributeGroup= 'descriptiveGroup1' (AttributeGroup, ID= 2)
     pDataColumn= 'BIL' (DataColumn, ID= 11)
  lm.prepro.Attribute.autoCreateIntervalEquifrequency:
'BIL_equifrequent' (Attribute, ID= 2)
     nCount= 30
     name= 'descriptiveGroup1'
  -->I have found a column named direc with data type Integer
number
  lm.explore.DataColumn.getDistinctValueCount: 'direc' (DataCol-
umn, ID= 10)
  lm.prepro.Attribute.create
     name= 'direc_equidistant'
     pAttributeGroup= 'descriptiveGroup1' (AttributeGroup, ID= 2)
     pDataColumn= 'direc' (DataColumn, ID= 10)
  lm.prepro.Attribute.autoCreateIntervalEquidistant:
'direc_equidistant' (Attribute, ID= 3)
     nCount= 30
  lm.prepro.Attribute.create
     name= 'direc_equifrequent'
     pAttributeGroup= 'descriptiveGroup1' (AttributeGroup, ID= 2)
     pDataColumn= 'direc' (DataColumn, ID= 10)
  lm.prepro.Attribute.autoCreateIntervalEquifrequency:
'direc_equifrequent' (Attribute, ID= 4)
     nCount= 30
```

```
    name= 'mainGroup'
 -->I have found a column named dust with data type Integer number
 lm.explore.DataColumn.getDistinctValueCount: 'dust' (DataColumn, ID= 5)
 lm.prepro.Attribute.create
    name= 'dust_equidistant'
    pAttributeGroup= 'mainGroup' (AttributeGroup, ID= 3)
    pDataColumn= 'dust' (DataColumn, ID= 5)
 lm.prepro.Attribute.autoCreateIntervalEquidistant: 'dust_equidistant' (Attribute, ID= 5)
    nCount= 30
 lm.prepro.Attribute.create
    name= 'dust_equifrequent'
    pAttributeGroup= 'mainGroup' (AttributeGroup, ID= 3)
    pDataColumn= 'dust' (DataColumn, ID= 5)
 lm.prepro.Attribute.autoCreateIntervalEquifrequency: 'dust_equifrequent' (Attribute, ID= 6)
    nCount= 30
    name= 'descriptiveGroup1'
 -->I have found a column named humid with data type Integer number
 lm.explore.DataColumn.getDistinctValueCount: 'humid' (DataColumn, ID= 7)
 lm.prepro.Attribute.create
    name= 'humid_equidistant'
    pAttributeGroup= 'descriptiveGroup1' (AttributeGroup, ID= 2)
    pDataColumn= 'humid' (DataColumn, ID= 7)
 lm.prepro.Attribute.autoCreateIntervalEquidistant: 'humid_equidistant' (Attribute, ID= 7)
    nCount= 30
 lm.prepro.Attribute.create
    name= 'humid_equifrequent'
    pAttributeGroup= 'descriptiveGroup1' (AttributeGroup, ID= 2)
    pDataColumn= 'humid' (DataColumn, ID= 7)
 lm.prepro.Attribute.autoCreateIntervalEquifrequency: 'humid_equifrequent' (Attribute, ID= 8)
    nCount= 30
 -->I have found a column named ID_LM with data type Integer number
 lm.explore.DataColumn.getDistinctValueCount: 'ID_LM' (DataColumn, ID= 26)
    name= 'mainGroup'
 -->I have found a column named oxNO with data type Decimal number
 lm.explore.DataColumn.getDistinctValueCount: 'oxNO' (DataColumn, ID= 2)
 lm.prepro.Attribute.create
    name= 'oxNO_equidistant'
    pAttributeGroup= 'mainGroup' (AttributeGroup, ID= 3)
    pDataColumn= 'oxNO' (DataColumn, ID= 2)
 lm.prepro.Attribute.autoCreateIntervalEquidistant: 'oxNO_equidistant' (Attribute, ID= 9)
    nCount= 30
 lm.prepro.Attribute.create
    name= 'oxNO_equifrequent'
    pAttributeGroup= 'mainGroup' (AttributeGroup, ID= 3)
    pDataColumn= 'oxNO' (DataColumn, ID= 2)
 lm.prepro.Attribute.autoCreateIntervalEquifrequency: 'oxNO_equifrequent' (Attribute, ID= 10)
    nCount= 30
 -->Could not find the group for attribute oxNO2
 -->I have found a column named oxNO2 with data type Decimal number
 lm.explore.DataColumn.getDistinctValueCount: 'oxNO2' (DataColumn, ID= 3)
 lm.prepro.Attribute.create
    name= 'oxNO2_equidistant'
    pAttributeGroup= 'Root group of attributes' (AttributeGroup, ID= 1)
    pDataColumn= 'oxNO2' (DataColumn, ID= 3)
 lm.prepro.Attribute.autoCreateIntervalEquidistant: 'oxNO2_equidistant' (Attribute, ID= 11)
    nCount= 30
 lm.prepro.Attribute.create
    name= 'oxNO2_equifrequent'
    pAttributeGroup= 'Root group of attributes' (AttributeGroup, ID= 1)
    pDataColumn= 'oxNO2' (DataColumn, ID= 3)
 lm.prepro.Attribute.autoCreateIntervalEquifrequency: 'oxNO2_equifrequent' (Attribute, ID= 12)
    nCount= 30
    name= 'mainGroup'
 -->I have found a column named oxNOX with data type Decimal number
 lm.explore.DataColumn.getDistinctValueCount: 'oxNOX' (DataColumn, ID= 4)
 lm.prepro.Attribute.create
    name= 'oxNOX_equidistant'
    pAttributeGroup= 'mainGroup' (AttributeGroup, ID= 3)
    pDataColumn= 'oxNOX' (DataColumn, ID= 4)
 lm.prepro.Attribute.autoCreateIntervalEquidistant: 'oxNOX_equidistant' (Attribute, ID= 13)
    nCount= 30
 lm.prepro.Attribute.create
    name= 'oxNOX_equifrequent'
    pAttributeGroup= 'mainGroup' (AttributeGroup, ID= 3)
    pDataColumn= 'oxNOX' (DataColumn, ID= 4)
 lm.prepro.Attribute.autoCreateIntervalEquifrequency: 'oxNOX_equifrequent' (Attribute, ID= 14)
    nCount= 30
    name= 'mainGroup'
 -->I have found a column named oxSO2 with data type Decimal number
 lm.explore.DataColumn.getDistinctValueCount: 'oxSO2' (DataColumn, ID= 1)
 lm.prepro.Attribute.create
    name= 'oxSO2_equidistant'
    pAttributeGroup= 'mainGroup' (AttributeGroup, ID= 3)
    pDataColumn= 'oxSO2' (DataColumn, ID= 1)
 lm.prepro.Attribute.autoCreateIntervalEquidistant: 'oxSO2_equidistant' (Attribute, ID= 15)
    nCount= 30
 lm.prepro.Attribute.create
    name= 'oxSO2_equifrequent'
    pAttributeGroup= 'mainGroup' (AttributeGroup, ID= 3)
    pDataColumn= 'oxSO2' (DataColumn, ID= 1)
 lm.prepro.Attribute.autoCreateIntervalEquifrequency: 'oxSO2_equifrequent' (Attribute, ID= 16)
    nCount= 30
    name= 'time_place'
 -->I have found a column named place with data type Integer number
 lm.explore.DataColumn.getDistinctValueCount: 'place' (DataColumn, ID= 13)
 lm.prepro.Attribute.create
    name= 'place'
    pAttributeGroup= 'time_place' (AttributeGroup, ID= 4)
    pDataColumn= 'place' (DataColumn, ID= 13)
 lm.prepro.Attribute.autoCreateEnumeration: 'place' (Attribute, ID= 17)
    name= 'descriptiveGroup1'
 -->I have found a column named pres with data type Decimal number
 lm.explore.DataColumn.getDistinctValueCount: 'pres' (DataColumn, ID= 8)
```

```
  lm.prepro.Attribute.create
     name= 'pres_equidistant'
     pAttributeGroup= 'descriptiveGroup1' (AttributeGroup, ID= 2)
     pDataColumn= 'pres' (DataColumn, ID= 8)
  lm.prepro.Attribute.autoCreateIntervalEquidistant:
'pres_equidistant' (Attribute, ID= 18)
     nCount= 30
  lm.prepro.Attribute.create
     name= 'pres_equifrequent'
     pAttributeGroup= 'descriptiveGroup1' (AttributeGroup, ID= 2)
     pDataColumn= 'pres' (DataColumn, ID= 8)
  lm.prepro.Attribute.autoCreateIntervalEquifrequency:
'pres_equifrequent' (Attribute, ID= 19)
     nCount= 30
     name= 'descriptiveGroup1'
  -->I have found a column named rain with data type Decimal
number
  lm.explore.DataColumn.getDistinctValueCount: 'rain' (DataCol-
umn, ID= 12)
  lm.prepro.Attribute.create
     name= 'rain_equidistant'
     pAttributeGroup= 'descriptiveGroup1' (AttributeGroup, ID= 2)
     pDataColumn= 'rain' (DataColumn, ID= 12)
  lm.prepro.Attribute.autoCreateIntervalEquidistant:
'rain_equidistant' (Attribute, ID= 20)
     nCount= 30
  lm.prepro.Attribute.create
     name= 'rain_equifrequent'
     pAttributeGroup= 'descriptiveGroup1' (AttributeGroup, ID= 2)
     pDataColumn= 'rain' (DataColumn, ID= 12)
  lm.prepro.Attribute.autoCreateIntervalEquifrequency:
'rain_equifrequent' (Attribute, ID= 21)
     nCount= 30
     name= 'descriptiveGroup1'
  -->I have found a column named temp_ with data type Decimal
number
  lm.explore.DataColumn.getDistinctValueCount:  'temp_'
(DataColumn, ID= 6)
  lm.prepro.Attribute.create
     name= 'temp__equidistant'
     pAttributeGroup= 'descriptiveGroup1' (AttributeGroup, ID= 2)
     pDataColumn= 'temp_' (DataColumn, ID= 6)
  lm.prepro.Attribute.autoCreateIntervalEquidistant:
'temp__equidistant' (Attribute, ID= 22)
     nCount= 30
  lm.prepro.Attribute.create
     name= 'temp__equifrequent'
     pAttributeGroup= 'descriptiveGroup1' (AttributeGroup, ID= 2)
     pDataColumn= 'temp_' (DataColumn, ID= 6)
  lm.prepro.Attribute.autoCreateIntervalEquifrequency:
'temp__equifrequent' (Attribute, ID= 23)
     nCount= 30
  -->I have found a column named time_date with data type
Date/Time
  lm.explore.DataColumn.getDistinctValueCount:  'time_date'
(DataColumn, ID= 14)
  -->Could not find the group for attribute time_date.Day
  -->I have found a column named time_date.Day with data type
Integer number
  lm.explore.DataColumn.getDistinctValueCount:  'time_date.Day'
(DataColumn, ID= 17)
  lm.prepro.Attribute.create
     name= 'time_date.Day_equidistant'
     pAttributeGroup= 'Root group of attributes' (AttributeGroup,
ID= 1)
     pDataColumn= 'time_date.Day' (DataColumn, ID= 17)
  lm.prepro.Attribute.autoCreateIntervalEquidistant:
'time_date.Day_equidistant' (Attribute, ID= 24)
     nCount= 30
  lm.prepro.Attribute.create
     name= 'time_date.Day_equifrequent'
     pAttributeGroup= 'Root group of attributes' (AttributeGroup,
ID= 1)
     pDataColumn= 'time_date.Day' (DataColumn, ID= 17)
  lm.prepro.Attribute.autoCreateIntervalEquifrequency:
'time_date.Day_equifrequent' (Attribute, ID= 25)
     nCount= 30
  -->Could not find the group for attribute time_date.DayOfRange
  -->I have found a column named time_date.DayOfRange with
data type Integer number
  lm.explore.DataColumn.getDistinctValueCount:
'time_date.DayOfRange' (DataColumn, ID= 25)
  lm.prepro.Attribute.create
     name= 'time_date.DayOfRange_equidistant'
     pAttributeGroup= 'Root group of attributes' (AttributeGroup,
ID= 1)
     pDataColumn= 'time_date.DayOfRange' (DataColumn, ID=
25)
  lm.prepro.Attribute.autoCreateIntervalEquidistant:
'time_date.DayOfRange_equidistant' (Attribute, ID= 26)
     nCount= 30
  lm.prepro.Attribute.create
     name= 'time_date.DayOfRange_equifrequent'
     pAttributeGroup= 'Root group of attributes' (AttributeGroup,
ID= 1)
     pDataColumn= 'time_date.DayOfRange' (DataColumn, ID=
25)
  lm.prepro.Attribute.autoCreateIntervalEquifrequency:
'time_date.DayOfRange_equifrequent' (Attribute, ID= 27)
     nCount= 30
  -->Could not find the group for attribute time_date.DayOfWeek
  -->I have found a column named time_date.DayOfWeek with
data type Integer number
  lm.explore.DataColumn.getDistinctValueCount:
'time_date.DayOfWeek' (DataColumn, ID= 21)
  lm.prepro.Attribute.create
     name= 'time_date.DayOfWeek'
     pAttributeGroup= 'Root group of attributes' (AttributeGroup,
ID= 1)
     pDataColumn= 'time_date.DayOfWeek' (DataColumn, ID= 21)
  lm.prepro.Attribute.autoCreateEnumeration:
'time_date.DayOfWeek' (Attribute, ID= 28)
  -->Could not find the group for attribute time_date.DayOfYear
  -->I have found a column named time_date.DayOfYear with data
type Integer number
  lm.explore.DataColumn.getDistinctValueCount:
'time_date.DayOfYear' (DataColumn, ID= 22)
  lm.prepro.Attribute.create
     name= 'time_date.DayOfYear_equidistant'
     pAttributeGroup= 'Root group of attributes' (AttributeGroup,
ID= 1)
     pDataColumn= 'time_date.DayOfYear' (DataColumn, ID= 22)
  lm.prepro.Attribute.autoCreateIntervalEquidistant:
'time_date.DayOfYear_equidistant' (Attribute, ID= 29)
     nCount= 30
  lm.prepro.Attribute.create
     name= 'time_date.DayOfYear_equifrequent'
     pAttributeGroup= 'Root group of attributes' (AttributeGroup,
ID= 1)
     pDataColumn= 'time_date.DayOfYear' (DataColumn, ID= 22)
  lm.prepro.Attribute.autoCreateIntervalEquifrequency:
'time_date.DayOfYear_equifrequent' (Attribute, ID= 30)
     nCount= 30
  -->Could not find the group for attribute time_date.Hour
  -->I have found a column named time_date.Hour with data type
Integer number
```

```
 lm.explore.DataColumn.getDistinctValueCount: 'time_date.Hour'
(DataColumn, ID= 18)
 lm.prepro.Attribute.create
    name= 'time_date.Hour_equidistant'
    pAttributeGroup= 'Root group of attributes' (AttributeGroup,
ID= 1)
    pDataColumn= 'time_date.Hour' (DataColumn, ID= 18)
 lm.prepro.Attribute.autoCreateIntervalEquidistant:
'time_date.Hour_equidistant' (Attribute, ID= 31)
    nCount= 30
 lm.prepro.Attribute.create
    name= 'time_date.Hour_equifrequent'
    pAttributeGroup= 'Root group of attributes' (AttributeGroup,
ID= 1)
    pDataColumn= 'time_date.Hour' (DataColumn, ID= 18)
 lm.prepro.Attribute.autoCreateIntervalEquifrequency:
'time_date.Hour_equifrequent' (Attribute, ID= 32)
    nCount= 30
 -->Could not find the group for attribute time_date.Min
 -->I have found a column named time_date.Min with data type
Integer number
 lm.explore.DataColumn.getDistinctValueCount: 'time_date.Min'
(DataColumn, ID= 19)
 lm.prepro.Attribute.create
    name= 'time_date.Min'
    pAttributeGroup= 'Root group of attributes' (AttributeGroup,
ID= 1)
    pDataColumn= 'time_date.Min' (DataColumn, ID= 19)
 lm.prepro.Attribute.autoCreateEnumeration: 'time_date.Min'
(Attribute, ID= 33)
 -->Could not find the group for attribute time_date.Month
 -->I have found a column named time_date.Month with data type
Integer number
 lm.explore.DataColumn.getDistinctValueCount:
'time_date.Month' (DataColumn, ID= 16)
 lm.prepro.Attribute.create
    name= 'time_date.Month'
    pAttributeGroup= 'Root group of attributes' (AttributeGroup,
ID= 1)
    pDataColumn= 'time_date.Month' (DataColumn, ID= 16)
 lm.prepro.Attribute.autoCreateEnumeration: 'time_date.Month'
(Attribute, ID= 34)
 -->Could not find the group for attribute time_date.Quarter
 -->I have found a column named time_date.Quarter with data
type Integer number
 lm.explore.DataColumn.getDistinctValueCount:
'time_date.Quarter' (DataColumn, ID= 24)
 lm.prepro.Attribute.create
    name= 'time_date.Quarter'
    pAttributeGroup= 'Root group of attributes' (AttributeGroup,
ID= 1)
    pDataColumn= 'time_date.Quarter' (DataColumn, ID= 24)
 lm.prepro.Attribute.autoCreateEnumeration: 'time_date.Quarter'
(Attribute, ID= 35)
 -->Could not find the group for attribute time_date.Sec
 -->I have found a column named time_date.Sec with data type
Integer number
 lm.explore.DataColumn.getDistinctValueCount: 'time_date.Sec'
(DataColumn, ID= 20)
 lm.prepro.Attribute.create
    name= 'time_date.Sec'
    pAttributeGroup= 'Root group of attributes' (AttributeGroup,
ID= 1)
    pDataColumn= 'time_date.Sec' (DataColumn, ID= 20)
 lm.prepro.Attribute.autoCreateEnumeration: 'time_date.Sec'
(Attribute, ID= 36)
 -->Could not find the group for attribute time_date.WeekOfYear
 -->I have found a column named time_date.WeekOfYear with
data type Integer number
 lm.explore.DataColumn.getDistinctValueCount:
'time_date.WeekOfYear' (DataColumn, ID= 23)
 lm.prepro.Attribute.create
    name= 'time_date.WeekOfYear_equidistant'
    pAttributeGroup= 'Root group of attributes' (AttributeGroup,
ID= 1)
    pDataColumn= 'time_date.WeekOfYear' (DataColumn, ID=
23)
 lm.prepro.Attribute.autoCreateIntervalEquidistant:
'time_date.WeekOfYear_equidistant' (Attribute, ID= 37)
    nCount= 30
 lm.prepro.Attribute.create
    name= 'time_date.WeekOfYear_equifrequent'
    pAttributeGroup= 'Root group of attributes' (AttributeGroup,
ID= 1)
    pDataColumn= 'time_date.WeekOfYear' (DataColumn, ID=
23)
 lm.prepro.Attribute.autoCreateIntervalEquifrequency:
'time_date.WeekOfYear_equifrequent' (Attribute, ID= 38)
    nCount= 30
 -->Could not find the group for attribute time_date.Year
 -->I have found a column named time_date.Year with data type
Integer number
 lm.explore.DataColumn.getDistinctValueCount: 'time_date.Year'
(DataColumn, ID= 15)
 lm.prepro.Attribute.create
    name= 'time_date.Year'
    pAttributeGroup= 'Root group of attributes' (AttributeGroup,
ID= 1)
    pDataColumn= 'time_date.Year' (DataColumn, ID= 15)
 lm.prepro.Attribute.autoCreateEnumeration: 'time_date.Year'
(Attribute, ID= 39)
    name= 'descriptiveGroup1'
 -->I have found a column named velo with data type Decimal
number
 lm.explore.DataColumn.getDistinctValueCount: 'velo' (DataCol-
umn, ID= 9)
 lm.prepro.Attribute.create
    name= 'velo_equidistant'
    pAttributeGroup= 'descriptiveGroup1' (AttributeGroup, ID= 2)
    pDataColumn= 'velo' (DataColumn, ID= 9)
 lm.prepro.Attribute.autoCreateIntervalEquidistant: 've-
lo_equidistant' (Attribute, ID= 40)
    nCount= 30
 lm.prepro.Attribute.create
    name= 'velo_equifrequent'
    pAttributeGroup= 'descriptiveGroup1' (AttributeGroup, ID= 2)
    pDataColumn= 'velo' (DataColumn, ID= 9)
 lm.prepro.Attribute.autoCreateIntervalEquifrequency: 've-
lo_equifrequent' (Attribute, ID= 41)
    nCount= 30
 lm.metabase.updateMetadata
 lm.metabase.close
 lm.metabase.backupMDB
    pathNameSrc=
'C:\diplo\automation_aktual\CompletData2.LM.mdb'
    pathNameDest=
'C:\diplo\automation_aktual\CompletData2.LM.mdb_bkup.preproc
ess.mdb'
 -->Start of domain knowledge creation
 lm.metabase.restoreMDB
    pathNameSrc=
'C:\diplo\automation_aktual\CompletData2.LM.mdb_bkup.preproc
ess.mdb'
    pathNameDest=
'C:\diplo\automation_aktual\CompletData2.LM.mdb'
```

```
 lm.metabase.open
    dataSourceName= 'LM Exec Ochodnicka CD2 MB'
    name= 'velo_equifrequent'
    name= 'oxSO2_equifrequent'
 lm.domain.MutualInfluence.create
    pAttributeRow= 'oxSO2_equifrequent' (Attribute, ID= 16)
    pAttributeCol= 'velo_equifrequent' (Attribute, ID= 41)
 lm.domain.MutualInfluence.autoCreateDiagonaleNegative:      '-'
(MutualInfluence, ID= 1)
    name= 'oxNO_equifrequent'
 lm.domain.MutualInfluence.create
    pAttributeRow= 'oxNO_equifrequent' (Attribute, ID= 10)
    pAttributeCol= 'velo_equifrequent' (Attribute, ID= 41)
 lm.domain.MutualInfluence.autoCreateDiagonaleNegative:      '-'
(MutualInfluence, ID= 2)
    name= 'oxNO2_equifrequent'
 lm.domain.MutualInfluence.create
    pAttributeRow= 'oxNO2_equifrequent' (Attribute, ID= 12)
    pAttributeCol= 'velo_equifrequent' (Attribute, ID= 41)
 lm.domain.MutualInfluence.autoCreateDiagonaleNegative:      '-'
(MutualInfluence, ID= 3)
    name= 'oxNOX_equifrequent'
 lm.domain.MutualInfluence.create
    pAttributeRow= 'oxNOX_equifrequent' (Attribute, ID= 14)
    pAttributeCol= 'velo_equifrequent' (Attribute, ID= 41)
 lm.domain.MutualInfluence.autoCreateDiagonaleNegative:      '-'
(MutualInfluence, ID= 4)
    name= 'dust_equifrequent'
 lm.domain.MutualInfluence.create
    pAttributeRow= 'dust_equifrequent' (Attribute, ID= 6)
    pAttributeCol= 'velo_equifrequent' (Attribute, ID= 41)
 lm.domain.MutualInfluence.autoCreateDiagonaleNegative:      '-'
(MutualInfluence, ID= 5)
 lm.metabase.updateMetadata
 lm.metabase.close
 lm.metabase.backupMDB
    pathNameSrc=
'C:\diplo\automation_aktual\CompletData2.LM.mdb'
    pathNameDest=
'C:\diplo\automation_aktual\CompletData2.LM.mdb_bkup.domain
.mdb'
 -->Start of tasks creation and run
 lm.metabase.restoreMDB
    pathNameSrc=
'C:\diplo\automation_aktual\CompletData2.LM.mdb_bkup.domain
.mdb'
    pathNameDest=
'C:\diplo\automation_aktual\CompletData2.LM.mdb'
 lm.metabase.open
    dataSourceName= 'LM Exec Ochodnicka CD2 MB'
 lm.tasks.TaskGroup.create
    name= 'Final'
 lm.tasks.TaskGroup.create
    name= 'DomainKnowledgeVerification'
 lm.tasks.TaskGroup.create
    name= 'DomainKnowledgeVerificationWithCondition'
    name= 'Meranie'
 lm.tasks.TaskKL.create
    name=
'kl_DomainKnowledgeVerification_oxSO2_equifrequent_velo_eq
uifrequent'
    pTaskGroup=  'DomainKnowledgeVerification'  (TaskGroup,
ID= 3)
    pDataTable= 'Meranie' (DataTable, ID= 1)
 lm.tasks.settings.KLQuantifierSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerification_oxSO2_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 1)
    nKLQuantifierTypeCode= 14
    dThresholdValue= 0.100
 lm.tasks.settings.KLPartialGroupSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerification_oxSO2_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 1)
    nCedentTypeCode= 11
 lm.tasks.settings.KLPartialGroupSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerification_oxSO2_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 1)
    nCedentTypeCode= 10
 lm.tasks.settings.FTPartialCedentSetting.create
    pTask=
'kl_DomainKnowledgeVerification_oxSO2_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 1)
    nCedentTypeCode= 4
 lm.tasks.settings.KLAttributeSetting.create
    pKLPartialGroupSetting= KLPartialGroupSetting (ID= 2)
    pAttribute= 'oxSO2_equifrequent' (Attribute, ID= 16)
 lm.tasks.settings.KLAttributeSetting.create
    pKLPartialGroupSetting= KLPartialGroupSetting (ID= 1)
    pAttribute= 'velo_equifrequent' (Attribute, ID= 41)
 lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerification_oxSO2_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 1)
   lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerification_oxSO2_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 1)
   lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerification_oxSO2_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 1)
    name= 'Meranie'
 lm.tasks.TaskKL.create
    name=
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifre
quent-->velo_equifrequent'
    pTaskGroup=  'DomainKnowledgeVerificationWithCondition'
(TaskGroup, ID= 4)
    pDataTable= 'Meranie' (DataTable, ID= 1)
 -->A new task
kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifre
quent-->velo_equifrequent created
 lm.tasks.settings.KLQuantifierSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifre
quent-->velo_equifrequent' (TaskKL, ID= 2)
    nKLQuantifierTypeCode= 14
    dThresholdValue= 0.500
 lm.tasks.settings.KLPartialGroupSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifre
quent-->velo_equifrequent' (TaskKL, ID= 2)
    nCedentTypeCode= 11
 lm.tasks.settings.KLPartialGroupSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifre
quent-->velo_equifrequent' (TaskKL, ID= 2)
    nCedentTypeCode= 10
 lm.tasks.settings.FTPartialCedentSetting.create
    pTask=
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifre
quent-->velo_equifrequent' (TaskKL, ID= 2)
    nCedentTypeCode= 4
 lm.tasks.settings.KLAttributeSetting.create
    pKLPartialGroupSetting= KLPartialGroupSetting (ID= 4)
    pAttribute= 'oxSO2_equifrequent' (Attribute, ID= 16)
 lm.tasks.settings.KLAttributeSetting.create
```

pKLPartialGroupSetting= KLPartialGroupSetting (ID= 3)
pAttribute= 'velo_equifrequent' (Attribute, ID= 41)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 2)
pAttribute= 'BIL_equidistant' (Attribute, ID= 1)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 2)
pAttribute= 'BIL_equifrequent' (Attribute, ID= 2)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 2)
pAttribute= 'direc_equidistant' (Attribute, ID= 3)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 2)
pAttribute= 'direc_equifrequent' (Attribute, ID= 4)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 2)
pAttribute= 'humid_equidistant' (Attribute, ID= 7)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 2)
pAttribute= 'humid_equifrequent' (Attribute, ID= 8)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 2)
pAttribute= 'pres_equidistant' (Attribute, ID= 18)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 2)
pAttribute= 'pres_equifrequent' (Attribute, ID= 19)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 2)
pAttribute= 'rain_equidistant' (Attribute, ID= 20)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 2)
pAttribute= 'rain_equifrequent' (Attribute, ID= 21)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 2)
pAttribute= 'temp__equidistant' (Attribute, ID= 22)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 2)
pAttribute= 'temp__equifrequent' (Attribute, ID= 23)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 2)
pAttribute= 'velo_equidistant' (Attribute, ID= 40)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 2)
pAttribute= 'velo_equifrequent' (Attribute, ID= 41)
--> Found main group, but it is useless to use it in condition
--> No groups for condition
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 2)
pAttribute= 'place' (Attribute, ID= 17)
lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifrequent-->velo_equifrequent' (TaskKL, ID= 2)
lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifrequent-->velo_equifrequent' (TaskKL, ID= 2)
lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifrequent-->velo_equifrequent' (TaskKL, ID= 2)
lm.metabase.reloadResults
-->Number of found hypothesis is 6
-->There is less hypotheses then needed, I will enlarge the search space
-->I am enlarging the search space
lm.tasks.Task.clone:
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifrequent-->velo_equifrequent' (TaskKL, ID= 2)
nKLQuantifierTypeCode= 14
-->The new K is 0.25
lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifrequent-->velo_equifrequent (01) 2' (TaskKL, ID= 3)
lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifrequent-->velo_equifrequent (01) 2' (TaskKL, ID= 3)
lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifrequent-->velo_equifrequent (01) 2' (TaskKL, ID= 3)
lm.metabase.reloadResults
-->Number of found hypothesis is 11
-->There is more hypotheses then needed, i will reduce the search space
-->I am reducing the search space
-->The new K is 0.375
lm.tasks.Task.clone:
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifrequent-->velo_equifrequent (01) 2' (TaskKL, ID= 3)
nKLQuantifierTypeCode= 14
lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifrequent-->velo_equifrequent (01) 2 (01) 3' (TaskKL, ID= 4)
lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifrequent-->velo_equifrequent (01) 2 (01) 3' (TaskKL, ID= 4)
lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifrequent-->velo_equifrequent (01) 2 (01) 3' (TaskKL, ID= 4)
lm.metabase.reloadResults
-->Number of found hypothesis is 11
-->There is more hypotheses then needed, i will reduce the search space
-->I am reducing the search space
-->The new K is 0.4375
lm.tasks.Task.clone:
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifrequent-->velo_equifrequent (01) 2 (01) 3' (TaskKL, ID= 4)
nKLQuantifierTypeCode= 14
lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifrequent-->velo_equifrequent (01) 2 (01) 3 (01) 4' (TaskKL, ID= 5)
lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifrequent-->velo_equifrequent (01) 2 (01) 3 (01) 4' (TaskKL, ID= 5)
lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifrequent-->velo_equifrequent (01) 2 (01) 3 (01) 4' (TaskKL, ID= 5)
lm.metabase.reloadResults
-->The number of found hypotheses is 9
-->Number of found hypothesis is 9
-->There is less hypotheses then needed, I will enlarge the search space
-->I am enlarging the search space
-->Change in task parameters is insignificant
End of task kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifrequent-->velo_equifrequent (01) 2 (01) 3 (01) 4
name= 'Meranie'
lm.tasks.TaskKL.create
name= 'kl_DomainKnowledgeVerification_oxNO_equifrequent_velo_equifrequent'
pTaskGroup= 'DomainKnowledgeVerification' (TaskGroup, ID= 3)
pDataTable= 'Meranie' (DataTable, ID= 1)
lm.tasks.settings.KLQuantifierSetting.create

```
    pTaskKL=
'kl_DomainKnowledgeVerification_oxNO_equifrequent_velo_equ
ifrequent' (TaskKL, ID= 6)
    nKLQuantifierTypeCode= 14
    dThresholdValue= 0.100
 lm.tasks.settings.KLPartialGroupSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerification_oxNO_equifrequent_velo_equ
ifrequent' (TaskKL, ID= 6)
    nCedentTypeCode= 11
 lm.tasks.settings.KLPartialGroupSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerification_oxNO_equifrequent_velo_equ
ifrequent' (TaskKL, ID= 6)
    nCedentTypeCode= 10
 lm.tasks.settings.FTPartialCedentSetting.create
    pTask=
'kl_DomainKnowledgeVerification_oxNO_equifrequent_velo_equ
ifrequent' (TaskKL, ID= 6)
    nCedentTypeCode= 4
 lm.tasks.settings.KLAttributeSetting.create
    pKLPartialGroupSetting= KLPartialGroupSetting (ID= 12)
    pAttribute= 'oxNO_equifrequent' (Attribute, ID= 10)
 lm.tasks.settings.KLAttributeSetting.create
    pKLPartialGroupSetting= KLPartialGroupSetting (ID= 11)
    pAttribute= 'velo_equifrequent' (Attribute, ID= 41)
 lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerification_oxNO_equifrequent_velo_equ
ifrequent' (TaskKL, ID= 6)
   lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerification_oxNO_equifrequent_velo_equ
ifrequent' (TaskKL, ID= 6)
   lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerification_oxNO_equifrequent_velo_equ
ifrequent' (TaskKL, ID= 6)
    name= 'Meranie'
 lm.tasks.TaskKL.create
    name=
'kl_DomainKnowledgeVerificationWithCondition_oxNO_equifreq
uent-->velo_equifrequent'
    pTaskGroup=  'DomainKnowledgeVerificationWithCondition'
(TaskGroup, ID= 4)
    pDataTable= 'Meranie' (DataTable, ID= 1)
 -->A                new                task
kl_DomainKnowledgeVerificationWithCondition_oxNO_equifreq
uent-->velo_equifrequent created
 lm.tasks.settings.KLQuantifierSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerificationWithCondition_oxNO_equifreq
uent-->velo_equifrequent' (TaskKL, ID= 7)
    nKLQuantifierTypeCode= 14
    dThresholdValue= 0.500
 lm.tasks.settings.KLPartialGroupSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerificationWithCondition_oxNO_equifreq
uent-->velo_equifrequent' (TaskKL, ID= 7)
    nCedentTypeCode= 11
 lm.tasks.settings.KLPartialGroupSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerificationWithCondition_oxNO_equifreq
uent-->velo_equifrequent' (TaskKL, ID= 7)
    nCedentTypeCode= 10
 lm.tasks.settings.FTPartialCedentSetting.create
    pTask=
'kl_DomainKnowledgeVerificationWithCondition_oxNO_equifreq
uent-->velo_equifrequent' (TaskKL, ID= 7)
    nCedentTypeCode= 4
 lm.tasks.settings.KLAttributeSetting.create
    pKLPartialGroupSetting= KLPartialGroupSetting (ID= 14)
    pAttribute= 'oxNO_equifrequent' (Attribute, ID= 10)
lm.tasks.settings.KLAttributeSetting.create
    pKLPartialGroupSetting= KLPartialGroupSetting (ID= 13)
    pAttribute= 'velo_equifrequent' (Attribute, ID= 41)
lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 7)
    pAttribute= 'BIL_equidistant' (Attribute, ID= 1)
lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 7)
    pAttribute= 'BIL_equifrequent' (Attribute, ID= 2)
lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 7)
    pAttribute= 'direc_equidistant' (Attribute, ID= 3)
lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 7)
    pAttribute= 'direc_equifrequent' (Attribute, ID= 4)
lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 7)
    pAttribute= 'humid_equidistant' (Attribute, ID= 7)
lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 7)
    pAttribute= 'humid_equifrequent' (Attribute, ID= 8)
lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 7)
    pAttribute= 'pres_equidistant' (Attribute, ID= 18)
lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 7)
    pAttribute= 'pres_equifrequent' (Attribute, ID= 19)
lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 7)
    pAttribute= 'rain_equidistant' (Attribute, ID= 20)
lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 7)
    pAttribute= 'rain_equifrequent' (Attribute, ID= 21)
lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 7)
    pAttribute= 'temp__equidistant' (Attribute, ID= 22)
lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 7)
    pAttribute= 'temp__equifrequent' (Attribute, ID= 23)
lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 7)
    pAttribute= 'velo_equidistant' (Attribute, ID= 40)
lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 7)
    pAttribute= 'velo_equifrequent' (Attribute, ID= 41)
 --> Found main group, but it is useless to use it in condition
 --> No groups for condition
lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 7)
    pAttribute= 'place' (Attribute, ID= 17)
 lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxNO_equifreq
uent-->velo_equifrequent' (TaskKL, ID= 7)
   lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerificationWithCondition_oxNO_equifreq
uent-->velo_equifrequent' (TaskKL, ID= 7)
   lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxNO_equifreq
uent-->velo_equifrequent' (TaskKL, ID= 7)
 lm.metabase.reloadResults
 -->Number of found hypothesis is 10
 Ideal number of hypotheses!
 End                of                task
kl_DomainKnowledgeVerificationWithCondition_oxNO_equifreq
uent-->velo_equifrequent
    name= 'Meranie'
```

```
 lm.tasks.TaskKL.create
    name=
'kl_DomainKnowledgeVerification_oxNO2_equifrequent_velo_eq
uifrequent'
    pTaskGroup= 'DomainKnowledgeVerification' (TaskGroup,
ID= 3)
    pDataTable= 'Meranie' (DataTable, ID= 1)
 lm.tasks.settings.KLQuantifierSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerification_oxNO2_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 8)
    nKLQuantifierTypeCode= 14
    dThresholdValue= 0.100
 lm.tasks.settings.KLPartialGroupSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerification_oxNO2_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 8)
    nCedentTypeCode= 11
 lm.tasks.settings.KLPartialGroupSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerification_oxNO2_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 8)
    nCedentTypeCode= 10
 lm.tasks.settings.FTPartialCedentSetting.create
    pTask=
'kl_DomainKnowledgeVerification_oxNO2_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 8)
    nCedentTypeCode= 4
 lm.tasks.settings.KLAttributeSetting.create
    pKLPartialGroupSetting= KLPartialGroupSetting (ID= 16)
    pAttribute= 'oxNO2_equifrequent' (Attribute, ID= 12)
 lm.tasks.settings.KLAttributeSetting.create
    pKLPartialGroupSetting= KLPartialGroupSetting (ID= 15)
    pAttribute= 'velo_equifrequent' (Attribute, ID= 41)
 lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerification_oxNO2_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 8)
   lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerification_oxNO2_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 8)
   lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerification_oxNO2_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 8)
    name= 'Meranie'
 lm.tasks.TaskKL.create
    name=
'kl_DomainKnowledgeVerificationWithCondition_oxNO2_equifre
quent-->velo_equifrequent'
    pTaskGroup= 'DomainKnowledgeVerificationWithCondition'
(TaskGroup, ID= 4)
    pDataTable= 'Meranie' (DataTable, ID= 1)
 -->A new task
kl_DomainKnowledgeVerificationWithCondition_oxNO2_equifre
quent-->velo_equifrequent created
 lm.tasks.settings.KLQuantifierSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerificationWithCondition_oxNO2_equifre
quent-->velo_equifrequent' (TaskKL, ID= 9)
    nKLQuantifierTypeCode= 14
    dThresholdValue= 0.500
 lm.tasks.settings.KLPartialGroupSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerificationWithCondition_oxNO2_equifre
quent-->velo_equifrequent' (TaskKL, ID= 9)
    nCedentTypeCode= 11
 lm.tasks.settings.KLPartialGroupSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerificationWithCondition_oxNO2_equifre
quent-->velo_equifrequent' (TaskKL, ID= 9)
    nCedentTypeCode= 10
 lm.tasks.settings.FTPartialCedentSetting.create
    pTask=
'kl_DomainKnowledgeVerificationWithCondition_oxNO2_equifre
quent-->velo_equifrequent' (TaskKL, ID= 9)
    nCedentTypeCode= 4
 lm.tasks.settings.KLAttributeSetting.create
    pKLPartialGroupSetting= KLPartialGroupSetting (ID= 18)
    pAttribute= 'oxNO2_equifrequent' (Attribute, ID= 12)
 lm.tasks.settings.KLAttributeSetting.create
    pKLPartialGroupSetting= KLPartialGroupSetting (ID= 17)
    pAttribute= 'velo_equifrequent' (Attribute, ID= 41)
 lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 9)
    pAttribute= 'BIL_equidistant' (Attribute, ID= 1)
 lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 9)
    pAttribute= 'BIL_equifrequent' (Attribute, ID= 2)
 lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 9)
    pAttribute= 'direc_equidistant' (Attribute, ID= 3)
 lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 9)
    pAttribute= 'direc_equifrequent' (Attribute, ID= 4)
 lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 9)
    pAttribute= 'humid_equidistant' (Attribute, ID= 7)
 lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 9)
    pAttribute= 'humid_equifrequent' (Attribute, ID= 8)
 lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 9)
    pAttribute= 'pres_equidistant' (Attribute, ID= 18)
 lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 9)
    pAttribute= 'pres_equifrequent' (Attribute, ID= 19)
 lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 9)
    pAttribute= 'rain_equidistant' (Attribute, ID= 20)
 lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 9)
    pAttribute= 'rain_equifrequent' (Attribute, ID= 21)
 lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 9)
    pAttribute= 'temp__equidistant' (Attribute, ID= 22)
 lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 9)
    pAttribute= 'temp__equifrequent' (Attribute, ID= 23)
 lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 9)
    pAttribute= 'velo_equidistant' (Attribute, ID= 40)
 lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 9)
    pAttribute= 'velo_equifrequent' (Attribute, ID= 41)
 --> Found main group, but it is useless to use it in condition
 --> No groups for condition
 lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 9)
    pAttribute= 'place' (Attribute, ID= 17)
 lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxNO2_equifre
quent-->velo_equifrequent' (TaskKL, ID= 9)
   lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerificationWithCondition_oxNO2_equifre
quent-->velo_equifrequent' (TaskKL, ID= 9)
```

```
    lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxNO2_equifre
quent-->velo_equifrequent' (TaskKL, ID= 9)
  lm.metabase.reloadResults
  -->Number of found hypothesis is 11
  -->There is more hypotheses then needed, i will reduce the search
space
  -->I am reducing the search space
  -->The new K is 0.75
  lm.tasks.Task.clone:
'kl_DomainKnowledgeVerificationWithCondition_oxNO2_equifre
quent-->velo_equifrequent' (TaskKL, ID= 9)
      nKLQuantifierTypeCode= 14
  lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxNO2_equifre
quent-->velo_equifrequent (01) 2' (TaskKL, ID= 10)
    lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerificationWithCondition_oxNO2_equifre
quent-->velo_equifrequent (01) 2' (TaskKL, ID= 10)
    lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxNO2_equifre
quent-->velo_equifrequent (01) 2' (TaskKL, ID= 10)
  lm.metabase.reloadResults
  -->The number of found hypotheses is 10
  -->Number of found hypothesis is 10
  Ideal number of hypotheses!
  End                    of                    task
kl_DomainKnowledgeVerificationWithCondition_oxNO2_equifre
quent-->velo_equifrequent (01) 2
      name= 'Meranie'
  lm.tasks.TaskKL.create
      name=
'kl_DomainKnowledgeVerification_oxNOX_equifrequent_velo_eq
uifrequent'
      pTaskGroup=  'DomainKnowledgeVerification'  (TaskGroup,
ID= 3)
      pDataTable= 'Meranie' (DataTable, ID= 1)
  lm.tasks.settings.KLQuantifierSetting.create
      pTaskKL=
'kl_DomainKnowledgeVerification_oxNOX_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 11)
      nKLQuantifierTypeCode= 14
      dThresholdValue= 0.100
  lm.tasks.settings.KLPartialGroupSetting.create
      pTaskKL=
'kl_DomainKnowledgeVerification_oxNOX_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 11)
      nCedentTypeCode= 11
  lm.tasks.settings.KLPartialGroupSetting.create
      pTaskKL=
'kl_DomainKnowledgeVerification_oxNOX_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 11)
      nCedentTypeCode= 10
  lm.tasks.settings.FTPartialCedentSetting.create
      pTask=
'kl_DomainKnowledgeVerification_oxNOX_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 11)
      nCedentTypeCode= 4
  lm.tasks.settings.KLAttributeSetting.create
      pKLPartialGroupSetting= KLPartialGroupSetting (ID= 22)
      pAttribute= 'oxNOX_equifrequent' (Attribute, ID= 14)
  lm.tasks.settings.KLAttributeSetting.create
      pKLPartialGroupSetting= KLPartialGroupSetting (ID= 21)
      pAttribute= 'velo_equifrequent' (Attribute, ID= 41)
  lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerification_oxNOX_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 11)
    lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerification_oxNOX_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 11)
    lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerification_oxNOX_equifrequent_velo_eq
uifrequent' (TaskKL, ID= 11)
      name= 'Meranie'
  lm.tasks.TaskKL.create
      name=
'kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifr
equent-->velo_equifrequent'
      pTaskGroup=  'DomainKnowledgeVerificationWithCondition'
(TaskGroup, ID= 4)
      pDataTable= 'Meranie' (DataTable, ID= 1)
  -->A                    new                    task
kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifre
quent-->velo_equifrequent created
  lm.tasks.settings.KLQuantifierSetting.create
      pTaskKL=
'kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifr
equent-->velo_equifrequent' (TaskKL, ID= 12)
      nKLQuantifierTypeCode= 14
      dThresholdValue= 0.500
  lm.tasks.settings.KLPartialGroupSetting.create
      pTaskKL=
'kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifr
equent-->velo_equifrequent' (TaskKL, ID= 12)
      nCedentTypeCode= 11
  lm.tasks.settings.KLPartialGroupSetting.create
      pTaskKL=
'kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifr
equent-->velo_equifrequent' (TaskKL, ID= 12)
      nCedentTypeCode= 10
  lm.tasks.settings.FTPartialCedentSetting.create
      pTask=
'kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifr
equent-->velo_equifrequent' (TaskKL, ID= 12)
      nCedentTypeCode= 4
  lm.tasks.settings.KLAttributeSetting.create
      pKLPartialGroupSetting= KLPartialGroupSetting (ID= 24)
      pAttribute= 'oxNOX_equifrequent' (Attribute, ID= 14)
  lm.tasks.settings.KLAttributeSetting.create
      pKLPartialGroupSetting= KLPartialGroupSetting (ID= 23)
      pAttribute= 'velo_equifrequent' (Attribute, ID= 41)
  lm.tasks.settings.FTLiteralSetting.create
      pFTPartialCedentSetting= FTPartialCedentSetting (ID= 12)
      pAttribute= 'BIL_equidistant' (Attribute, ID= 1)
  lm.tasks.settings.FTLiteralSetting.create
      pFTPartialCedentSetting= FTPartialCedentSetting (ID= 12)
      pAttribute= 'BIL_equifrequent' (Attribute, ID= 2)
  lm.tasks.settings.FTLiteralSetting.create
      pFTPartialCedentSetting= FTPartialCedentSetting (ID= 12)
      pAttribute= 'direc_equidistant' (Attribute, ID= 3)
  lm.tasks.settings.FTLiteralSetting.create
      pFTPartialCedentSetting= FTPartialCedentSetting (ID= 12)
      pAttribute= 'direc_equifrequent' (Attribute, ID= 4)
  lm.tasks.settings.FTLiteralSetting.create
      pFTPartialCedentSetting= FTPartialCedentSetting (ID= 12)
      pAttribute= 'humid_equidistant' (Attribute, ID= 7)
  lm.tasks.settings.FTLiteralSetting.create
      pFTPartialCedentSetting= FTPartialCedentSetting (ID= 12)
      pAttribute= 'humid_equifrequent' (Attribute, ID= 8)
  lm.tasks.settings.FTLiteralSetting.create
      pFTPartialCedentSetting= FTPartialCedentSetting (ID= 12)
      pAttribute= 'pres_equidistant' (Attribute, ID= 18)
  lm.tasks.settings.FTLiteralSetting.create
      pFTPartialCedentSetting= FTPartialCedentSetting (ID= 12)
      pAttribute= 'pres_equifrequent' (Attribute, ID= 19)
```

lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 12)
pAttribute= 'rain_equidistant' (Attribute, ID= 20)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 12)
pAttribute= 'rain_equifrequent' (Attribute, ID= 21)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 12)
pAttribute= 'temp__equidistant' (Attribute, ID= 22)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 12)
pAttribute= 'temp__equifrequent' (Attribute, ID= 23)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 12)
pAttribute= 'velo_equidistant' (Attribute, ID= 40)
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 12)
pAttribute= 'velo_equifrequent' (Attribute, ID= 41)
--> Found main group, but it is useless to use it in condition
--> No groups for condition
lm.tasks.settings.FTLiteralSetting.create
pFTPartialCedentSetting= FTPartialCedentSetting (ID= 12)
pAttribute= 'place' (Attribute, ID= 17)
lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifrequent-->velo_equifrequent' (TaskKL, ID= 12)
lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifrequent-->velo_equifrequent' (TaskKL, ID= 12)
lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifrequent-->velo_equifrequent' (TaskKL, ID= 12)
lm.metabase.reloadResults
-->Number of found hypothesis is 11
-->There is more hypotheses then needed, i will reduce the search space
-->I am reducing the search space
-->The new K is 0.75
lm.tasks.Task.clone:
'kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifrequent-->velo_equifrequent' (TaskKL, ID= 12)
nKLQuantifierTypeCode= 14
lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifrequent-->velo_equifrequent (01) 2' (TaskKL, ID= 13)
lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifrequent-->velo_equifrequent (01) 2' (TaskKL, ID= 13)
lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifrequent-->velo_equifrequent (01) 2' (TaskKL, ID= 13)
lm.metabase.reloadResults
-->The number of found hypotheses is 8
-->Number of found hypothesis is 8
-->There is less hypotheses then needed, I will enlarge the search space
-->I am enlarging the search space
lm.tasks.Task.clone:
'kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifrequent-->velo_equifrequent (01) 2' (TaskKL, ID= 13)
nKLQuantifierTypeCode= 14
-->The new K is 0.625
lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifrequent-->velo_equifrequent (01) 2 (01) 3' (TaskKL, ID= 14)
lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifrequent-->velo_equifrequent (01) 2 (01) 3' (TaskKL, ID= 14)
lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifrequent-->velo_equifrequent (01) 2 (01) 3' (TaskKL, ID= 14)
lm.metabase.reloadResults
-->Number of found hypothesis is 10
Ideal number of hypotheses!
End of task kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifrequent-->velo_equifrequent (01) 2 (01) 3
name= 'Meranie'
lm.tasks.TaskKL.create
name= 'kl_DomainKnowledgeVerification_dust_equifrequent_velo_equifrequent'
pTaskGroup= 'DomainKnowledgeVerification' (TaskGroup, ID= 3)
pDataTable= 'Meranie' (DataTable, ID= 1)
lm.tasks.settings.KLQuantifierSetting.create
pTaskKL= 'kl_DomainKnowledgeVerification_dust_equifrequent_velo_equifrequent' (TaskKL, ID= 15)
nKLQuantifierTypeCode= 14
dThresholdValue= 0.100
lm.tasks.settings.KLPartialGroupSetting.create
pTaskKL= 'kl_DomainKnowledgeVerification_dust_equifrequent_velo_equifrequent' (TaskKL, ID= 15)
nCedentTypeCode= 11
lm.tasks.settings.KLPartialGroupSetting.create
pTaskKL= 'kl_DomainKnowledgeVerification_dust_equifrequent_velo_equifrequent' (TaskKL, ID= 15)
nCedentTypeCode= 10
lm.tasks.settings.FTPartialCedentSetting.create
pTask= 'kl_DomainKnowledgeVerification_dust_equifrequent_velo_equifrequent' (TaskKL, ID= 15)
nCedentTypeCode= 4
lm.tasks.settings.KLAttributeSetting.create
pKLPartialGroupSetting= KLPartialGroupSetting (ID= 30)
pAttribute= 'dust_equifrequent' (Attribute, ID= 6)
lm.tasks.settings.KLAttributeSetting.create
pKLPartialGroupSetting= KLPartialGroupSetting (ID= 29)
pAttribute= 'velo_equifrequent' (Attribute, ID= 41)
lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerification_dust_equifrequent_velo_equifrequent' (TaskKL, ID= 15)
lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerification_dust_equifrequent_velo_equifrequent' (TaskKL, ID= 15)
lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerification_dust_equifrequent_velo_equifrequent' (TaskKL, ID= 15)
name= 'Meranie'
lm.tasks.TaskKL.create
name= 'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequent-->velo_equifrequent'
pTaskGroup= 'DomainKnowledgeVerificationWithCondition' (TaskGroup, ID= 4)
pDataTable= 'Meranie' (DataTable, ID= 1)
-->A new task kl_DomainKnowledgeVerificationWithCondition_dust_equifrequent-->velo_equifrequent created
lm.tasks.settings.KLQuantifierSetting.create
pTaskKL= 'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequent-->velo_equifrequent' (TaskKL, ID= 16)

```
    nKLQuantifierTypeCode= 14
    dThresholdValue= 0.500
  lm.tasks.settings.KLPartialGroupSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent' (TaskKL, ID= 16)
    nCedentTypeCode= 11
  lm.tasks.settings.KLPartialGroupSetting.create
    pTaskKL=
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent' (TaskKL, ID= 16)
    nCedentTypeCode= 10
  lm.tasks.settings.FTPartialCedentSetting.create
    pTask=
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent' (TaskKL, ID= 16)
    nCedentTypeCode= 4
  lm.tasks.settings.KLAttributeSetting.create
    pKLPartialGroupSetting= KLPartialGroupSetting (ID= 32)
    pAttribute= 'dust_equifrequent' (Attribute, ID= 6)
  lm.tasks.settings.KLAttributeSetting.create
    pKLPartialGroupSetting= KLPartialGroupSetting (ID= 31)
    pAttribute= 'velo_equifrequent' (Attribute, ID= 41)
  lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 16)
    pAttribute= 'BIL_equidistant' (Attribute, ID= 1)
  lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 16)
    pAttribute= 'BIL_equifrequent' (Attribute, ID= 2)
  lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 16)
    pAttribute= 'direc_equidistant' (Attribute, ID= 3)
  lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 16)
    pAttribute= 'direc_equifrequent' (Attribute, ID= 4)
  lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 16)
    pAttribute= 'humid_equidistant' (Attribute, ID= 7)
  lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 16)
    pAttribute= 'humid_equifrequent' (Attribute, ID= 8)
  lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 16)
    pAttribute= 'pres_equidistant' (Attribute, ID= 18)
  lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 16)
    pAttribute= 'pres_equifrequent' (Attribute, ID= 19)
  lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 16)
    pAttribute= 'rain_equidistant' (Attribute, ID= 20)
  lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 16)
    pAttribute= 'rain_equifrequent' (Attribute, ID= 21)
  lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 16)
    pAttribute= 'temp__equidistant' (Attribute, ID= 22)
  lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 16)
    pAttribute= 'temp__equifrequent' (Attribute, ID= 23)
  lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 16)
    pAttribute= 'velo_equidistant' (Attribute, ID= 40)
  lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 16)
    pAttribute= 'velo_equifrequent' (Attribute, ID= 41)
  --> Found main group, but it is useless to use it in condition
  --> No groups for condition
  lm.tasks.settings.FTLiteralSetting.create
    pFTPartialCedentSetting= FTPartialCedentSetting (ID= 16)
    pAttribute= 'place' (Attribute, ID= 17)
  lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent' (TaskKL, ID= 16)
    lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent' (TaskKL, ID= 16)
    lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent' (TaskKL, ID= 16)
  lm.metabase.reloadResults
  -->Number of found hypothesis is 11
  -->There is more hypotheses then needed, i will reduce the search
space
  -->I am reducing the search space
  -->The new K is 0.75
  lm.tasks.Task.clone:
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent' (TaskKL, ID= 16)
    nKLQuantifierTypeCode= 14
  lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent (01) 2' (TaskKL, ID= 17)
    lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent (01) 2' (TaskKL, ID= 17)
    lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent (01) 2' (TaskKL, ID= 17)
  lm.metabase.reloadResults
  -->The number of found hypotheses is 7
  -->Number of found hypothesis is 7
  -->There is less hypotheses then needed, I will enlarge the search
space
  -->I am enlarging the search space
  lm.tasks.Task.clone:
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent (01) 2' (TaskKL, ID= 17)
    nKLQuantifierTypeCode= 14
  -->The new K is 0.625
  lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent (01) 2 (01) 3' (TaskKL, ID= 18)
    lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent (01) 2 (01) 3' (TaskKL, ID= 18)
    lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent (01) 2 (01) 3' (TaskKL, ID= 18)
  lm.metabase.reloadResults
  -->Number of found hypothesis is 11
  -->There is more hypotheses then needed, i will reduce the search
space
  -->I am reducing the search space
  -->The new K is 0.6875
  lm.tasks.Task.clone:
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent (01) 2 (01) 3' (TaskKL, ID= 18)
    nKLQuantifierTypeCode= 14
  lm.tasks.Task.runAndWaitForResults:
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent (01) 2 (01) 3 (01) 4' (TaskKL, ID= 19)
    lm.tasks.Task.runAsync:
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent (01) 2 (01) 3 (01) 4' (TaskKL, ID= 19)
```

```
    lm.tasks.Task.waitForResults:
'kl_DomainKnowledgeVerificationWithCondition_dust_equifrequ
ent-->velo_equifrequent (01) 2 (01) 3 (01) 4' (TaskKL, ID= 19)
  lm.metabase.reloadResults
  -->The number of found hypotheses is 10
  -->Number of found hypothesis is 10
  Ideal number of hypotheses!
  End                    of                    task
kl_DomainKnowledgeVerificationWithCondition_dust_equifreque
nt-->velo_equifrequent (01) 2 (01) 3 (01) 4
  lm.metabase.updateMetadata
  lm.metabase.close
  lm.metabase.backupMDB
      pathNameSrc=
'C:\diplo\automation_aktual\CompletData2.LM.mdb'
      pathNameDest=
'C:\diplo\automation_aktual\CompletData2.LM.mdb_bkup.tasks.m
db'
  -->Start of report creation
  lm.metabase.restoreMDB
      pathNameSrc=
'C:\diplo\automation_aktual\CompletData2.LM.mdb_bkup.tasks.m
db'
      pathNameDest=
'C:\diplo\automation_aktual\CompletData2.LM.mdb'
  lm.metabase.open
      dataSourceName= 'LM Exec Ochodnicka CD2 MB'
  Creating analytical report
  Metabase was closed.
  End of script.
ScriptExec-end
```

Appendix 3, Final report

Automation Analytical Report

Input parameters

Source data file: C:\diplo\automation_aktual\XochzData.txt
Domain knowledge (attribute groups): descriptiveGroup1, mainGroup, time_place
Preferred number of patterns to be found: 10
Report file destination: C:\diplo\automation_aktual\report.html

Data Exploration

Meranie

Original columns:

- BIL: Decimal number
- direc: Integer number
- dust: Integer number
- humid: Integer number
- oxNO: Decimal number
- oxNO2: Decimal number
- oxNOX: Decimal number
- oxSO2: Decimal number
- place: Integer number
- pres: Decimal number
- rain: Decimal number
- temp_: Decimal number
- time_date: Date/Time
- velo: Decimal number

Derived columns:

- time_date.Day: Integer number
- time_date.DayOfRange: Integer number
- time_date.DayOfWeek: Integer number
- time_date.DayOfYear: Integer number
- time_date.Hour: Integer number
- time_date.Min: Integer number
- time_date.Month: Integer number
- time_date.Quarter: Integer number
- time_date.Sec: Integer number
- time_date.WeekOfYear: Integer number

- time_date.Year: Integer number

Data Preprocessing

descriptiveGroup1

BIL_equidistant: <-384;-347.32333>, (-347.32333;-310.64667>, (-310.64667;-273.97>, (-273.97;-237.29333>, (-237.29333;-200.61667>, (-200.61667;-163.94>, (-163.94;-127.26334>, (-127.26334;-90.58667>, (-90.58667;-53.910004>, (-53.910004;-17.233337>, (-17.233337;19.443329>, (19.443329;56.119995>, (56.119995;92.796661>, (92.796661;129.47333>, (129.47333;166.14999>...

BIL_equifrequent: <-384;-66.2), <-66.2;-61.2), <-61.2;-57.5), <-57.5;-54), <-54;-50.6), <-50.6;-46.9), <-46.9;-42.7), <-42.7;-38), <-38;-32.5), <-32.5;-27.7), <-27.7;-23.1), <-23.1;-19.2), <-19.2;-15.8), <-15.8;-12.7), <-12.7;-9.8)...

direc_equidistant: <0;11>, (12;23>, (24;35>, (36;47>, (48;59>, (60;71>, (72;83>, (84;95>, (96;107>, (108;119>, (120;131>, (132;143>, (144;155>, (156;167>, (168;179>...

direc_equifrequent: <0;24), <24;42), <42;54), <54;63), <63;73), <73;81), <81;90), <90;100), <100;110), <110;119), <119;133), <133;144), <144;154), <154;165), <165;181)...

humid_equidistant: <19;21>, (22;24>, (25;27>, (28;30>, (31;33>, (34;36>, (37;39>, (40;42>, (43;45>, (46;48>, (49;51>, (52;54>, (55;57>, (58;60>, (61;63>...

humid_equifrequent: <19;39), <39;44), <44;49), <49;53), <53;58), <58;62), <62;65), <65;68), <68;71), <71;73), <73;75), <75;77), <77;79), <79;81), <81;83)...

pres_equidistant: <860.88;866.424>, (866.424;871.968>, (871.968;877.51199>, (877.51199;883.05599>, (883.05599;888.59999>, (888.59999;894.14399>, (894.14399;899.68799>, (899.68799;905.23199>, (905.23199;910.77598>, (910.77598;916.31998>, (916.31998;921.86398>, (921.86398;927.40798>, (927.40798;932.95198>, (932.95198;938.49597>, (938.49597;944.03997>...

pres_equifrequent: <860.88;969.8), <969.8;975.7), <975.7;979.5), <979.5;981.9), <981.9;983.76), <983.76;985), <985;986.2), <986.2;987.3), <987.3;988.22), <988.22;989.2), <989.2;990), <990;990.8), <990.8;991.7), <991.7;992.88), <992.88;994.6)...

rain_equidistant: <0;1.546667>, (1.546667;3.093333>, (3.093333;4.64>, (4.64;6.186667>, (6.186667;7.733334>, (7.733334;9.28>, (9.28;10.826667>, (10.826667;12.373334>, (12.373334;13.92>, (13.92;15.466667>, (15.466667;17.013334>, (17.013334;18.560001>, (18.560001;20.106667>, (20.106667;21.653334>, (21.653334;23.200001>...

rain_equifrequent: <0;0.2), <0.2;0.4), <0.4;0.6), <0.6;0.8), <0.8;1), <1;1.2), <1.2;1.4), <1.4;1.6), <1.6;1.8), <1.8;2), <2;2.2), <2.2;2.4), <2.4;2.6), <2.6;2.8), <2.8;3)...

temp__equidistant: <-14.8;-13.036667>, (-13.036667;-11.273333>, (-11.273333;-9.51>, (-9.51;-7.746667>, (-7.746667;-5.983334>, (-5.983334;-4.22>, (-4.22;-2.456667>, (-2.456667;-0.693334>, (-0.693334;1.07>, (1.07;2.833333>, (2.833333;4.596666>, (4.596666;6.359999>, (6.359999;8.123333>, (8.123333;9.886666>, (9.886666;11.649999>...

temp__equifrequent: <-14.8;-3.7), <-3.7;-2.2), <-2.2;-1.2), <-1.2;0), <0;1), <1;1.6), <1.6;2.4), <2.4;3.3), <3.3;4.5), <4.5;5.7), <5.7;7.2), <7.2;8.1), <8.1;9.2), <9.2;10.2), <10.2;11)...
velo_equidistant: <0;0.87>, (0.87;1.74>, (1.74;2.61>, (2.61;3.48>, (3.48;4.35>, (4.35;5.22>, (5.22;6.09>, (6.09;6.96>, (6.96;7.83>, (7.83;8.7>, (8.7;9.57>, (9.57;10.44>, (10.44;11.31>, (11.31;12.18>, (12.18;13.05>...
velo_equifrequent: <0;0.2), <0.2;0.3), <0.3;0.4), <0.4;0.5), <0.5;0.6), <0.6;0.7), <0.7;0.8), <0.8;0.9), <0.9;1), <1;1.1), <1.1;1.2), <1.2;1.3), <1.3;1.4), <1.4;1.5), <1.5;1.6)...

mainGroup

dust_equidistant: <-4;3>, (4;11>, (12;19>, (20;27>, (28;35>, (36;43>, (44;51>, (52;59>, (60;67>, (68;75>, (76;83>, (84;91>, (92;99>, (100;107>, (108;115>...
dust_equifrequent: <-4;4), <4;6), <6;8), <8;9), <9;11), <11;12), <12;13), <13;14), <14;15), <15;16), <16;17), <17;18), <18;19), <19;20), <20;21)...
oxNO_equidistant: <0;1.72>, (1.72;3.44>, (3.44;5.16>, (5.16;6.88>, (6.88;8.6>, (8.6;10.32>, (10.32;12.04>, (12.04;13.76>, (13.76;15.48>, (15.48;17.199999>, (17.199999;18.919999>, (18.919999;20.639999>, (20.639999;22.359999>, (22.359999;24.079999>, (24.079999;25.799999>...
oxNO_equifrequent: <0;0.2), <0.2;0.3), <0.3;0.4), <0.4;0.5), <0.5;0.6), <0.6;0.7), <0.7;0.8), <0.8;0.9), <0.9;1), <1;1.1), <1.1;1.2), <1.2;1.3), <1.3;1.4), <1.4;1.5), <1.5;1.6)...
oxNOX_equidistant: <0;4.786667>, (4.786667;9.573334>, (9.573334;14.360001>, (14.360001;19.146667>, (19.146667;23.933334>, (23.933334;28.720001>, (28.720001;33.506668>, (33.506668;38.293335>, (38.293335;43.080002>, (43.080002;47.866669>, (47.866669;52.653336>, (52.653336;57.440002>, (57.440002;62.226669>, (62.226669;67.013336>, (67.013336;71.800003>...
oxNOX_equifrequent: <0;2.8), <2.8;3.6), <3.6;4.1), <4.1;4.6), <4.6;5.1), <5.1;5.6), <5.6;6), <6;6.5), <6.5;6.9), <6.9;7.4), <7.4;7.8), <7.8;8.2), <8.2;8.7), <8.7;9.2), <9.2;9.7)...
oxSO2_equidistant: <0;16.163333>, (16.163333;32.326666>, (32.326666;48.489999>, (48.489999;64.653333>, (64.653333;80.816666>, (80.816666;96.979999>, (96.979999;113.14333>, (113.14333;129.30667>, (129.30667;145.47>, (145.47;161.63333>, (161.63333;177.79666>, (177.79666;193.96>, (193.96;210.12333>, (210.12333;226.28666>, (226.28666;242.45>...
oxSO2_equifrequent: <0;0.3), <0.3;0.7), <0.7;0.8), <0.8;1), <1;1.1), <1.1;1.2), <1.2;1.3), <1.3;1.5), <1.5;1.6), <1.6;1.7), <1.7;1.8), <1.8;2), <2;2.2), <2.2;2.4), <2.4;2.6)...

time_place

place: 1, 2

Tasks

kl_DomainKnowledgeVerification_dust_equifrequent_velo_equifrequent

Task finished but an acceptable number of patterns has not been achieved
Number of iterations: 1
Found patterns: 1
The most interesting ones:

- **dust_equifrequent** × **velo_equifrequent**

kl_DomainKnowledgeVerification_oxNO_equifrequent_velo_equifrequent

Task finished but an acceptable number of patterns has not been achieved
Number of iterations: 1
No interesting results found.

kl_DomainKnowledgeVerification_oxNO2_equifrequent_velo_equifrequent

Task finished but an acceptable number of patterns has not been achieved
Number of iterations: 1
Found patterns: 1
The most interesting ones:

- **oxNO2_equifrequent** × **velo_equifrequent**

kl_DomainKnowledgeVerification_oxNOX_equifrequent_velo_equifrequent

Task finished but an acceptable number of patterns has not been achieved
Number of iterations: 1
Found patterns: 1
The most interesting ones:

- **oxNOX_equifrequent** × **velo_equifrequent**

kl_DomainKnowledgeVerification_oxSO2_equifrequent_velo_equifrequent

Task finished but an acceptable number of patterns has not been achieved
Number of iterations: 1
No interesting results found.

kl_DomainKnowledgeVerificationWithCondition_dust_equifrequent-->velo_equifrequent (01) 2 (01) 3 (01) 4

Task finished succesfully and an acceptable number of patterns has been found
Number of iterations: 4
Found patterns: 10
The most interesting ones:

- **dust_equifrequent** × **velo_equifrequent** / **BIL_equidistant**(*(-347.32333;-310.64667>*)
- **dust_equifrequent** × **velo_equifrequent** / **BIL_equidistant**(*(-273.97;-237.29333>*)
- **dust_equifrequent** × **velo_equifrequent** / **humid_equidistant**(<=*21*)
- **dust_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(921.86398;927.40798>*)
- **dust_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(944.03997;949.58397>*)
- **dust_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(949.58397;955.12797>*)
- **dust_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(955.12797;960.67197>*)

- **dust_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(960.67197;966.21597>*)
- **dust_equifrequent** × **velo_equifrequent** / **rain_equidistant**(*(21.653334;23.200001>*)
- **dust_equifrequent** × **velo_equifrequent** / **rain_equidistant**(>*44.853335*)

kl_DomainKnowledgeVerificationWithCondition_oxNO_equifrequent-->velo_equifrequent

Task finished succesfully and an acceptable number of patterns has been found
Number of iterations: 1
Found patterns: 10
The most interesting ones:

- **oxNO_equifrequent** × **velo_equifrequent** / **BIL_equidistant**(*(-347.32333;-310.64667>*)
- **oxNO_equifrequent** × **velo_equifrequent** / **BIL_equidistant**(*(642.94666;679.62332>*)
- **oxNO_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(871.968;877.51199>*)
- **oxNO_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(944.03997;949.58397>*)
- **oxNO_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(949.58397;955.12797>*)
- **oxNO_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(955.12797;960.67197>*)
- **oxNO_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(960.67197;966.21597>*)
- **oxNO_equifrequent** × **velo_equifrequent** / **rain_equidistant**(*(24.746667;26.293334>*)
- **oxNO_equifrequent** × **velo_equifrequent** / **rain_equidistant**(*(30.933334;32.480001>*)
- **oxNO_equifrequent** × **velo_equifrequent** / **rain_equidistant**(>*44.853335*)

kl_DomainKnowledgeVerificationWithCondition_oxNO2_equifrequent-->velo_equifrequent (01) 2

Task finished succesfully and an acceptable number of patterns has been found
Number of iterations: 2
Found patterns: 10
The most interesting ones:

- **oxNO2_equifrequent** × **velo_equifrequent** / **BIL_equidistant**(*(-347.32333;-310.64667>*)
- **oxNO2_equifrequent** × **velo_equifrequent** / **BIL_equidistant**(*(-273.97;-237.29333>*)
- **oxNO2_equifrequent** × **velo_equifrequent** / **BIL_equidistant**(*(642.94666;679.62332>*)
- **oxNO2_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(921.86398;927.40798>*)
- **oxNO2_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(944.03997;949.58397>*)
- **oxNO2_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(949.58397;955.12797>*)
- **oxNO2_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(955.12797;960.67197>*)
- **oxNO2_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(960.67197;966.21597>*)
- **oxNO2_equifrequent** × **velo_equifrequent** / **rain_equidistant**(*(30.933334;32.480001>*)

- **oxNO2_equifrequent** × **velo_equifrequent** / **rain_equidistant**(>*44.853335*)

kl_DomainKnowledgeVerificationWithCondition_oxNOX_equifrequent-->velo_equifrequent (01) 2 (01) 3

Task finished succesfully and an acceptable number of patterns has been found
Number of iterations: 3
Found patterns: 10
The most interesting ones:

- **oxNOX_equifrequent** × **velo_equifrequent** / **BIL_equidistant**(*(-347.32333;-310.64667>*)
- **oxNOX_equifrequent** × **velo_equifrequent** / **BIL_equidistant**(*(-273.97;-237.29333>*)
- **oxNOX_equifrequent** × **velo_equifrequent** / **BIL_equidistant**(*(642.94666;679.62332>*)
- **oxNOX_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(921.86398;927.40798>*)
- **oxNOX_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(944.03997;949.58397>*)
- **oxNOX_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(949.58397;955.12797>*)
- **oxNOX_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(955.12797;960.67197>*)
- **oxNOX_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(960.67197;966.21597>*)
- **oxNOX_equifrequent** × **velo_equifrequent** / **rain_equidistant**(*(30.933334;32.480001>*)
- **oxNOX_equifrequent** × **velo_equifrequent** / **rain_equidistant**(*(34.026668;35.573335>*)

kl_DomainKnowledgeVerificationWithCondition_oxSO2_equifrequent-->velo_equifrequent (01) 2 (01) 3 (01) 4

Task finished but an acceptable number of patterns has not been achieved
Number of iterations: 4
Found patterns: 9
The most interesting ones:

- **oxSO2_equifrequent** × **velo_equifrequent** / **BIL_equidistant**(*(642.94666;679.62332>*)
- **oxSO2_equifrequent** × **velo_equifrequent** / **BIL_equidistant**(>*679.62332*)
- **oxSO2_equifrequent** × **velo_equifrequent** / **humid_equidistant**(<=*21*)
- **oxSO2_equifrequent** × **velo_equifrequent** / **humid_equidistant**(*(31;33>*)
- **oxSO2_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(944.03997;949.58397>*)
- **oxSO2_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(949.58397;955.12797>*)
- **oxSO2_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(955.12797;960.67197>*)
- **oxSO2_equifrequent** × **velo_equifrequent** / **pres_equidistant**(*(960.67197;966.21597>*)
- **oxSO2_equifrequent** × **velo_equifrequent** / **rain_equidistant**(>*44.853335*)

Apendix 4, Scripts

XochzStart MODULE

```
--This is a start module of the script. This module calls all other main sub-processes:
import,
--explore, preprocess, domain, tasks and results and inicializes all other modules.
--Also, initial parametes are declared here. At the end of this module, changes are submit-
ted to a metabase and then the metabse is close and the whole process is ended.

--Import of libraries
local lm= require( "Exec/Lib/LMGlobal");

--Look-up packages also in the script directory
package.path= package.path..";"..lm.getScriptDirectory().."?.lua";

require( "xochz");
require( "XochzPreprocess");
require ("XochzImport");
require ("XochzExplore");
require ("XochzDomain");
require ("XochzTasks");
require ("XochzDomainKnowledgeVerification");
require ("XochzNewKnowledgeSpecialized");
require ("XochzKLMiner");
require ("XochzKLspaceAdjustment");
require ("XochzAllToAllProcess");
require ("XochzResults");

--Import parameters
inputParams= {

--path to the txt file with data
pathNameDataSrc= lm.getScriptDirectory().."XochzData.txt",

--path and filename of the database to be created from the text data file
pathNameDataDest= lm.getScriptDirectory().."XochzData.DB.mdb",

--path and filename of the metabase to be created
pathNameMetabase= lm.getScriptDirectory().."XochzData.LM.mdb",

--name of the database table to be created
tableName= "Meranie",

--base ODBC DataSourceName for both metabase and data
dsnBase= "Exec Ochodnicka CD2",

--path and filename of the backup copy of a metabase
pathNameBkup= lm.getScriptDirectory().."XochzData.LM.mdb_bkup.mdb",

--list of groups the data can be divided to
domainAttributeGroups= {"descriptiveGroup1", "mainGroup", "time_place"},

--pairing attributes with their attribute group
domainAttributeToGroup={
time_date="time_place",
oxSO2="mainGroup",
oxNO="mainGroup",
oxNOX="mainGroup",
dust="mainGroup",
temp_="descriptiveGroup1",
humid="descriptiveGroup1",
pres="descriptiveGroup1",
velo="descriptiveGroup1",
direc="descriptiveGroup1",
BIL="descriptiveGroup1",
rain="descriptiveGroup1",
place="time_place",
},

--desirable number of hypotheses that a user wishes to find
idealHypothesesCount=10,
```

```
--path and name of the final report file
pathNameReportOutput=lm.getScriptDirectory().."report.html"
};



--Log to inform about recording parameters
lm.log ("-->Input parameters recorded");

--Declaration of parameters needed for data import
importParams= {
pathNameSrc= inputParams.pathNameDataSrc,
pathNameDest= inputParams.pathNameDataDest,
tableName= inputParams.tableName,

};

--Import of a data TXT file and metabase creation
lm.log("-->Start of data import and metabase creation");
xochz.import.creation(inputParams);

--Exploring data
lm.log("-->Start of exploration");
xochz.explore.exploration();


--Data preprocessing
lm.log("-->Start of data preprocessing");
xochz.preprocess.createAttributes(inputParams);

--Domain knowledge declaration
lm.log("-->Start of domain knowledge creation");
xochz.domain.createDomain(inputParams);

--Tasks creation and run
lm.log("-->Start of tasks creation and run");
xochz.tasks.tasksprocess(inputParams);

--Report creation
lm.log("-->Start of report creation");
xochz.results.exportReport( inputParams);


--Updating metabase

if (lm.metabase.isOpen()== true) then

lm.metabase.updateMetadata();

end;

--Closing metabase
lm.metabase.close();
lm.log("Metabase was closed.");

--Log to inform about the end of the script.


lm.log("End of script.");
```

XochzImport MODULE

```
--The import module imports the data from a txt file, creates an mdb database from the text
data
--file, as well as creates a metabase and associates it with the mdb database from the text
data.
--Then it opens the metabase, updates it with all the changes, closes it and creates a
back-up
--metabase copy to be used in the Explore module.
```

```
xochz.import = {};


function xochz.import.creation(inputParams)

--Import of the data from the text file and creation of an mdb database from the data file
lm.data.importTXT(importParams);

--Log information if the import was succesful
lm.log ("-->Data import succesful");

--Creation of a metabase and associating it with the previously made database from the used
data
lm.metabase.createAndAssociateWithDataMDB({
       pathNameMetabase=inputParams.pathNameMetabase,
       pathNameData=inputParams.pathNameDataDest,
       dsnBase=inputParams.dsnBase} );

----Log information if the association was succesful
lm.log("Data and metabase succesfuly associated.");

--Opening of the metabase in order to write there
lm.metabase.open({dataSourceName= "LM "..inputParams.dsnBase.." MB"}) ;
lm.log("-->Metabase was opened.");

--Updating metabase
lm.metabase.updateMetadata();
lm.log("-->Metabase was updated.");

--Closing metabase
lm.metabase.close();

--Creation of a back-up copy of metabase
lm.metabase.backupMDB({
pathNameSrc= inputParams.pathNameMetabase,
pathNameDest= inputParams.pathNameMetabase.."_bkup.import.mdb"
});

end;

function xochz.import.getMetabaseDSN (inputParams)
return "LM "..inputParams.dsnBase.." MB";
end;

return xochz.import;
```

XochExplore MODULE

```
--Explore module used to initialize all data tables in the used data as well as setting ---
the primary key

xochz.explore = {};


function xochz.explore.exploration()

--Opening a metabase that was created in the DataImportAndMetabase process
bOpened= false;
       if ( lm.metabase.isOpen()== false) then

                   lm.metabase.restoreMDB({
                   pathNameSrc= inputParams.pathNameMetabase.."_bkup.import.mdb",
                   pathNameDest= inputParams.pathNameMetabase
               });

               lm.metabase.open({
                    dataSourceName= xochz.import.getMetabaseDSN( inputParams)});
               bOpened= true;
        end;

--Collecting all data tables from the used data
```

```
local dataTableArray= lm.explore.prepareDataTableArray();
lm.log("-->Array of data tables prepared");

--Initialization of data tables
for i, dataTable in ipairs( dataTableArray) do
lm.log( "-->Initializing data table ".. dataTable.Name);
dataTable.init();

--Defining primary key if not already defined
if ( not dataTable.isPrimaryKeyDefined()) then
        dataTable.markPrimaryKey({
                columnName= lm.data.IDColumnNameDefault
                    })
end;

dataTable.LocalDataCacheFlag= true;

--Information about the data size
lm.log( "-->Number of records in the table ".. dataTable.Name.." is
"..dataTable.RecordCount);
end;

lm.metabase.updateMetadata();

lm.log("-->Primary key has been defined");

lm.metabase.close();

--Create a backup metabase copy of explore process
lm.metabase.backupMDB({
      pathNameSrc= inputParams.pathNameMetabase,
      pathNameDest= inputParams.pathNameMetabase.."_bkup.explore.mdb"
     });



end;



return xochz.explore;
```

XochzPreprocess MODULE

```
--Data preprocessing: first, groups for attributes are created, then columns are found for
each --data table and for each column name (if it is not part of primary key or is not in
"date"
--format) an atribute group where it belongs is found thanks to the declared list of column
names -and the given attribute groups. Then the specific attributes are created for each
column given
--the column and the attribute group where it belongs.

xochz.preprocess={};



--Attributes groups creation

function xochz.preprocess.createAttributes(inputParams)

--Opening and using the metabase that was created in the explore process
bOpened= false;
if ( lm.metabase.isOpen()== false) then

     lm.metabase.restoreMDB({
             pathNameSrc= inputParams.pathNameMetabase.."_bkup.explore.mdb",
             pathNameDest= inputParams.pathNameMetabase
          });

     lm.metabase.open({
     dataSourceName= xochz.import.getMetabaseDSN( inputParams)});
     bOpened= true;
end;
```

```
--Clearing to assure clean cache for start
lm.metabase.clearLocalDataCache();

listOfAttributes={};
isMainAttributeGroupUsed=nil;

--Creating of attribute groups
lm.log("-->Starting attribute groups creation");

rootAttributeGroup= lm.prepro.getRootAttributeGroup();

--Creating attribute groups with name as defined in the input parameters in Start molude
for i, attributeGroupName in ipairs(inputParams.domainAttributeGroups)do

   attributeGroup= lm.prepro.AttributeGroup({
       name= attributeGroupName,
       pParentGroup= rootAttributeGroup
      });
   lm.log("-->Attribute group named "..attributeGroup.getName().." was created.");
end;
lm.log("-->All attribute groups created");

--Attributes creation
lm.log("-->Start of attributes creation.");
rootAttributeGroup= lm.prepro.getRootAttributeGroup();

--Finding data tables
local dataTableArray= lm.explore.prepareDataTableArray();


--For each data table finds its columns
for j, dataTable in ipairs (dataTableArray) do

     if ( dataTable.isInitialized()) then
     lm.log( "-->Creating attributes for the table "..dataTable.Name);
     local dataColumnArray= dataTable.prepareDataColumnArray();

--For each column, an attribute group that it belongs to is found thanks to
--the declared list domainAttributeToGroup that pairs columns with attribute groups
          for k, column in ipairs( dataColumnArray) do
            attribute= nil;


            if ( (not column.isPrimaryKeyPart()) and
              (column.getValueSubTypeCode() ~= lm.codes.ValueSubType.DateTime)) then


                attributeGroup= lm.prepro.findAttributeGroup({
                   name= inputParams.domainAttributeToGroup[ column.getName()]
                 });


            end;


            if ( attributeGroup ~= nil) then

--Change of the isMainAttributeGroupUsed parameter status in case the main attribute group
--has been declared
                attributeGroupName=attributeGroup.getName();

                if (attributeGroupName == "mainGroup") then
                   isMainAttributeGroupUsed=1;
                end;
            end;

--If no attribute groups that belong to used columns are found then use the root attribute
group
--instead
            if ( attributeGroup== nil) then

               lm.log( "-->Could not find the group for attribute "..column.Name);
               attributeGroup= rootAttributeGroup;

            end;
```

```
            lm.log("-->I have found a column named "..column.getName().." with data type
"..column.getValueSubTypeName());



--Creating attributes for each specific column

--Creating enumerations (each value onecategory) for columns with text values and boolean
values,
--also columns with distinct count of values less then 21 and columns containing date/time
values

            if (column.getValueSubTypeName()== "Text" or
               column.getValueSubTypeName()== "Boolean" or
               column.getDistinctValueCount() < 21 ) then

                   local attribute = lm.prepro.Attribute({
                        name= (column.getName()),
                        pAttributeGroup= attributeGroup,
                        pDataColumn= column
                   });

           attribute.autoCreateEnumeration({});
           listOfAttributes["attribute "..column.getName()]=attribute;
           attribute=nil;


           elseif  (column.getValueSubTypeName()~= "Date/Time") and
                   (column.isPrimaryKeyPart()== false)  then
                       local attribute = lm.prepro.Attribute({
                          name= (column.getName().."_equidistant"),
                          pAttributeGroup= attributeGroup,
                          pDataColumn= column
                       });

                      attribute.autoCreateIntervalEquidistant({
                      nCount= 30
                         });
                listOfAttributes["attribute_equidistant "..column.getName()]=attribute;


                attribute=nil;

                attribute = lm.prepro.Attribute({
                      name= (column.getName().."_equifrequent"),
                      pAttributeGroup= attributeGroup,
                      pDataColumn= column
                     });

               attribute.autoCreateIntervalEquifrequency({
                      nCount= 30
                    });
               listOfAttributes["attribute_equifrequent "..column.getName()]=attribute;

               attribute=nil;

               elseif  (column.getName() == "time_date.&.*")  then
                       attribute = lm.prepro.Attribute({
                               name= (column.getName()),
                               pAttributeGroup= attributeGroup,
                               pDataColumn= column
                             });


               end;


         end;
end;
end;


lm.metabase.updateMetadata();
lm.metabase.close();

--Create a backup copy of metabase updated in this method for further usage
lm.metabase.backupMDB({
```

```
          pathNameSrc= inputParams.pathNameMetabase,
          pathNameDest= inputParams.pathNameMetabase.."_bkup.preprocess.mdb"
});

end;

return xochz.preprocess;
```

XochzDomain MODULE

```
--This domain module is responsible for declaring dependencies and their type between 2
--attributes.
--This module is data specific and creates only dependency type "influence" between attrib-
utes
--that are air pollutants in the used data set and wind velocity.

xochz.domain={};


function xochz.domain.createDomain(inputParams)

--Opening and using the metabase that was created in the proprocessing process
bOpened= false;
if ( lm.metabase.isOpen()== false) then

lm.metabase.restoreMDB({
pathNameSrc= inputParams.pathNameMetabase.."_bkup.preprocess.mdb",
pathNameDest= inputParams.pathNameMetabase
});

lm.metabase.open({
dataSourceName= xochz.import.getMetabaseDSN( inputParams)});
bOpened= true;
end;
--List of names of attributes that will be used as row attributes for "influence" dependen-
cy type
rowAttributesNames={
"oxSO2_equifrequent",
"oxNO_equifrequent",
"oxNO2_equifrequent",
"oxNOX_equifrequent",
"dust_equifrequent"
};

--Declaration of an attribute that will be used as the column attribute for "influence"
--dependency type
local columnAttribute= lm.prepro.findAttribute({

name="velo_equifrequent"
});

--Creation of dependencies between row and column attributes,
--the dependency type being "negative influence"
for i, attributeName in ipairs (rowAttributesNames) do


local rowAttribute= lm.prepro.findAttribute({

name=attributeName
});

dependency= lm.domain.MutualInfluence({

pAttributeRow =rowAttribute,
pAttributeCol = columnAttribute
});

dependency.setMutualInfluenceTypeCode(
lm.codes.MutualInfluenceType.NegativeInfluence);

--Creating atomic rules
dependency.autoCreateDiagonaleNegative();
```

```
end;

lm.metabase.updateMetadata();
lm.metabase.close();

--Create a backup copy for further use
lm.metabase.backupMDB({
pathNameSrc= inputParams.pathNameMetabase,
pathNameDest= inputParams.pathNameMetabase.."_bkup.domain.mdb"
});



end;

return xochz.domain;
```

XochzTasks MODULE

```
--Tasks module is for determining which of the task creation branches will be started.

xochz.tasks={};


function xochz.tasks.tasksprocess(inputParams)

--Opening and using the metabase that was created in the proprocessing process
bOpened= false;
if ( lm.metabase.isOpen()== false) then

lm.metabase.restoreMDB({
pathNameSrc= inputParams.pathNameMetabase.."_bkup.domain.mdb",
pathNameDest= inputParams.pathNameMetabase
});

lm.metabase.open({
dataSourceName= xochz.import.getMetabaseDSN( inputParams)});
bOpened= true;
end;

--Creation of final task group-group, where all the best set tasks will be stored
finalTasks=lm.tasks.TaskGroup({
name="Final"
});

influenceArray= lm.domain.prepareMutualInfluenceArray();


--used only to fasten the process- without prepro and import part
--isMainAttributeGroupUsed=1;

--Check if the main attribute group is not empty
if (isMainAttributeGroupUsed==1) then


--Check if there was any domain knowldge declared
if (influenceArray ~= nil) then

--If both checks are true, the domain knowledge verification module will be run
xochz.DomainKnowledgeVerification.run(inputParams);

end;
--If there is no domain knowledge set, but the main attribute group is not empty, the new
--knowledge
--specialized module will be run
if (influenceArray == nil) then

xochz.NewKnowledgeSpecialized.run();

end;

--If the main attribute group is empty, the the AllToAll module will be run
else
```

```
xochz.allToAllProcess.createAllToAllProcess();

end;

--Waiting for xxPooler to shutdown
lm.sleep( 1000);
lm.metabase.updateMetadata();
lm.metabase.close();
lm.sleep( 1000*( lm.tasks.getPoolerShutdownDelay()+ 2));

--Create a backup copy for debug
lm.metabase.backupMDB({
pathNameSrc= inputParams.pathNameMetabase,
pathNameDest= inputParams.pathNameMetabase.."_bkup.tasks.mdb"
});

end;

return xochz.tasks;
```

XochzDomainKnowledgeVerification MODULE

```
--Process to decide which LISp-Miner procedure to use in the next step of domain knowledge
verification


xochz.DomainKnowledgeVerification = {};


function xochz.DomainKnowledgeVerification.run(inputParams)



--Creating task groups for verifying domain knowledge
DomainKnowledgeVerificationTasks=lm.tasks.TaskGroup({
name="DomainKnowledgeVerification"
});

DomainKnowledgeVerificationConditionTasks=lm.tasks.TaskGroup({
name="DomainKnowledgeVerificationWithCondition"
});

--Creating a table of all mutual dependencies declared in the previous - domain part
local influenceArray= lm.domain.prepareMutualInfluenceArray();

for i, mutualInfluence in ipairs (influenceArray) do

influenceType= mutualInfluence.getMutualInfluenceTypeCode();

--If the declared influence type is any type of influence, KL-Miner procedure will be run
if( (influenceType==lm.codes.MutualInfluenceType.PositiveInfluence) or
(influenceType==lm.codes.MutualInfluenceType.NegativeInfluence) or
(influenceType==lm.codes.MutualInfluenceType.SomeInfluence))then

local rowAttribute= mutualInfluence.getAttributeRow()
local columnAttribute=mutualInfluence.getAttributeCol();
xochz.klMiner.DomainKnowledgeVerification(rowAttribute,columnAttribute);
xochz.klMiner.DomainKnowledgeVerificationCondition(rowAttribute,columnAttribute, input-
Params);

--If the declared influence type is any type of frequency, CF-Miner procedure will be run
elseif( (influenceType==lm.codes.MutualInfluenceType.PositiveFrequency) or
(influenceType==lm.codes.MutualInfluenceType.NegativeFrequency) )then

xochz.cfMiner.DomainKnowledgeVerification();

--If the declared influence type is the Boolean type, 4ft-Miner procedure will be run
elseif( (influenceType==lm.codes.MutualInfluenceType.PositiveBoolean) or
(influenceType==lm.codes.MutualInfluenceType.NegativeBoolean) )then

xochz.ftMiner.DomainKnowledgeVerification();

--If the declared influence type is not set yet, all KL, CF and 4ft-Miner procedures will
```

```
be run
--in order to establish the dependency
elseif( (influenceType==lm.codes.MutualInfluenceType.NotSet) or
(influenceType==lm.codes.MutualInfluenceType.Unknown) )then

local rowAttribute= mutualInfluence.getAttributeRow()
local columnAttribute=mutualInfluence.getAttributeCol();
xochz.klMiner.DomainKnowledgeVerification(rowAttribute,columnAttribute);
xochz.klMiner.DomainKnowledgeVerificationCondition(rowAttribute,columnAttribute, input-
Params);
xochz.cfMiner.DomainKnowledgeVerification();
xochz.ftMiner.DomainKnowledgeVerification();

--If the declared influence type is any other type, new knowledge specialized process will
be run
else
lm.log("--> Not interested in these influences, no need for verification");
xochz.NewKnowledgeSpecialized.run();

end;


end;

end;

return xochz.DomainKnowledgeVerification;
```

XochzKLMiner MODULE

```
--The klMiner module contains 2 functions: the first one to set and run a KL-Miner task
without
--condition and the other one to set and run a KL-Miner task with conditions.

xochz.klMiner={};

--A function to set and run a KL-Miner task without condition
function  xochz.klMiner.DomainKnowledgeVerification(rowAttribute,columnAttribute)


local dataTable= lm.explore.findDataTable({
name= inputParams.tableName
});
--Creation of a new KL task
taskDomainKnowledgeKL=lm.tasks.TaskKL({

name="kl_DomainKnowledgeVerification_"..rowAttribute.Name.."_"..columnAttribute.Name,
pTaskGroup= DomainKnowledgeVerificationTasks,
pDataTable= dataTable
});

lm.log("-->A new task "..taskDomainKnowledgeKL.Name.." created");

--Setting of a KL-Miner quantifier
klQuantifierSetting=lm.tasks.settings.KLQuantifierSetting({

pTaskKL=taskDomainKnowledgeKL,
nKLQuantifierTypeCode= lm.codes.KLQuantifierType.Kendall,
dThresholdValue= 0.1

});
klQuantifierSetting.setFromCol (-1);
klQuantifierSetting.setFromRow (-1);
klQuantifierSetting.setToCol (-100);
klQuantifierSetting.setToRow (-100);
--For using only absolute values of Kendall's coefficient
klQuantifierSetting.KendallTauBAbsValueFlag= true;


--Declaration of KL column partial cedent
klColSetting=lm.tasks.settings.KLPartialGroupSetting({
```

```
pTaskKL=taskDomainKnowledgeKL,
nCedentTypeCode= lm.codes.CedentType.KLAttributeCol

});


--Declaration of KL row partial cedent
klRowSetting=lm.tasks.settings.KLPartialGroupSetting({

pTaskKL=taskDomainKnowledgeKL,
nCedentTypeCode= lm.codes.CedentType.KLAttributeRow

});
--Declaration of an empty condition
klConditionSetting=lm.tasks.settings.FTPartialCedentSetting({

pTask=taskDomainKnowledgeKL,
nCedentTypeCode= lm.codes.CedentType.Condition

});


--Creation of row attribute
klRowAttributeSetting=lm.tasks.settings.KLAttributeSetting({

pKLPartialGroupSetting=klRowSetting,
pAttribute=rowAttribute

});

--Creation of column attribute
klColAttributeSetting=lm.tasks.settings.KLAttributeSetting({

pKLPartialGroupSetting=klColSetting,
pAttribute=columnAttribute

});


--Running the task
taskDomainKnowledgeKL.runAndWaitForResults ({});

--Placing the task into final task group to by displayed in the final report
taskDomainKnowledgeKL.setTaskGroup(finalTasks );


end;


--Domain knowledge verification with condition tasks setting
function  xo-
chz.klMiner.DomainKnowledgeVerificationCondition(rowAttribute,columnAttribute,inputParams)

--Task parameters inicialization
KLAdjustmentParams={
--Kendall threshold value used for current task

K=0.5,

--Kendall threshold value used for previous run task
lastK=nil,

--Kendall threshold value used two runs ago, i.e. the Kendall threshold value used in the
task
--before previous one
preLastK=nil,

--Kendall threshold value used in the turning point task, where its run was followed by two
--different search space adjustments
turnK=0.5,

--parameter with the information whether the previous search was reduction or enlargement

lastSearch=nil,

--Boolean parameter, true if all previous search space adjustments were enlargements
allEnlargement=nil,
```

```
--Boolean parameter, true if all previous search space adjustments were reductions
allReduction=nil,
--paratemer with information how many times a task was turn- 1 means a task without space
--adjustments, 2 means there has been one space adjustment and so on
taskRun=1

};
local dataTable= lm.explore.findDataTable({
name= inputParams.tableName
});

--Creation of a new KL task
taskDomainKnowledgeKLCondition=lm.tasks.TaskKL({

name="kl_DomainKnowledgeVerificationWithCondition_"..rowAttribute.Name.."--
>"..columnAttribute.Name,
pTaskGroup= DomainKnowledgeVerificationConditionTasks,
pDataTable= dataTable
});
lm.log("-->A new task "..taskDomainKnowledgeKLCondition.Name.." created");

--Setting of maximal number of found hypotheses, generation stops if exceeded. It is set to
--number of idealHypotheses parameter plus one more, so that the script knows, that there
was
--more then needed hypotheses found
taskDomainKnowledgeKLCondition.setHypothesisCountMax(inputParams.idealHypothesesCount +1);

--Setting of a KL-Miner quantifier
klConditionQuantifierSetting=lm.tasks.settings.KLQuantifierSetting({

pTaskKL=taskDomainKnowledgeKLCondition,
nKLQuantifierTypeCode= lm.codes.KLQuantifierType.Kendall,
dThresholdValue= 0.5

});
klConditionQuantifierSetting.setFromCol (-1);
klConditionQuantifierSetting.setFromRow (-1);
klConditionQuantifierSetting.setToCol (-100);
klConditionQuantifierSetting.setToRow (-100);

--For using only absolute values of Kendall's coefficient
klConditionQuantifierSetting.KendallTauBAbsValueFlag= true;


--Declaration of KL column partial cedent
klConditionColSetting=lm.tasks.settings.KLPartialGroupSetting({


pTaskKL=taskDomainKnowledgeKLCondition,
nCedentTypeCode= lm.codes.CedentType.KLAttributeCol

});


--Declaration of KL row partial cedent
klConditionRowSetting=lm.tasks.settings.KLPartialGroupSetting({

pTaskKL=taskDomainKnowledgeKLCondition,
nCedentTypeCode= lm.codes.CedentType.KLAttributeRow

});

--Declaration of KL condition

klCConditionSetting=lm.tasks.settings.FTPartialCedentSetting({

pTask=taskDomainKnowledgeKLCondition,
nCedentTypeCode= lm.codes.CedentType.Condition

});
klCConditionSetting.setMaxLen (1);
klCConditionSetting.setMinLen (1);


--Creation of row attribute
klConditionRowAttributeSetting=lm.tasks.settings.KLAttributeSetting({
```

```
pKLPartialGroupSetting=klConditionRowSetting,
pAttribute=rowAttribute

});


--Creation of column attribute
klConditionColAttributeSetting=lm.tasks.settings.KLAttributeSetting({

pKLPartialGroupSetting=klConditionColSetting,
pAttribute=columnAttribute

});

--Creation of condition attributes
local attributeGroups=lm.prepro.prepareAttributeGroupArray();

if (attributeGroups~= nil) then
for i, attributeGroup in ipairs (attributeGroups) do

if (attributeGroup.Name~="mainGroup"  and attributeGroup.Name ~=
lm.prepro.getRootAttributeGroup().Name) then
conditionAttributes=attributeGroup.prepareAttributeArray ();

for j, conditionAttribute in ipairs (conditionAttributes) do

--Condition is always a 4ft-cedent, so it has to be set with FTLiteralSetting

klCConAttSetting=lm.tasks.settings.FTLiteralSetting({

pFTPartialCedentSetting=klCConditionSetting,
pAttribute=conditionAttribute

});

end;

elseif (attributeGroup.Name=="mainGroup") then
lm.log("--> Found main group attribute, but it is useless to use it in condition");

else
lm.log("--> This attribute group is not suitable for condition");
end;

end;
else
lm.log("--> No attribute groups found.");
end;


--Running the task
taskDomainKnowledgeKLCondition.runAndWaitForResults ({});

--Partially reloads the metabase with results to preserve other objects (tasks, prepro)
--already used in the Lua script
lm.metabase.reloadResults();

--Running of search space adjustments
xochz.KLspaceAdjustment.run(taskDomainKnowledgeKLCondition, KLAdjustmentParams);


end;
return xochz.klMiner;
```

XochzKLSpaceAdjustment MODULE

```
--Functions to adjust search space of KL-Miner knowledge verification tasks

xochz.KLspaceAdjustment={};

--Function to set adjusting paramaters and determine whether to enlarge or reduce the
search
--space
function xochz.KLspaceAdjustment.run(taskDomainKnowledgeKLCondition, KLAdjustmentParams)
```

```
--Finding how many hypotheses were found in the last run of the task
taskDKKLCcount=taskDomainKnowledgeKLCondition.getHypothesisCount();
lm.log("-->Number of found hypothesis is "..taskDKKLCcount);

--If there were less hypotheses found then needed
if(taskDKKLCcount < inputParams.idealHypothesesCount ) then


--Changing of parameters, so that the space enlargement or reduction functions can work
with
--current values of parameters
KLAdjustmentParams.preLastK=KLAdjustmentParams.lastK;
KLAdjustmentParams.lastK=KLAdjustmentParams.K;

lm.log("-->There is less hypotheses then needed, I will enlarge the search space");


--If it was the first time the task has been run, then these two parameters has to be
changed in
--this function, otherwise they are changed in the space enlargement or space reduction
functions
if (KLAdjustmentParams.taskRun==1) then

--So far all search space adjustments are enlargements (including the one that is only go-
ing to
--happen)
KLAdjustmentParams.allEnlargement=true;

--There is already going to be an enlargement, so not all the adjustments will be reduc-
tions
KLAdjustmentParams.allReduction=false;

end;

--Start of the search space enlargement
xochz.KLspaceAdjustment.enlargeSearchSpace();


--If there were more hypotheses found then needed
elseif (taskDKKLCcount > inputParams.idealHypothesesCount )then


--Changing of parameters, so that the space enlargement or reduction functions can work
with
--Current values of parameters
KLAdjustmentParams.preLastK=KLAdjustmentParams.lastK;
KLAdjustmentParams.lastK=KLAdjustmentParams.K;

lm.log("-->There is more hypotheses then needed, i will reduce the search space");

--If it was the first time the task has been run, then these two parameters has to be
changed in
--this function, otherwise they are changed in the space enlargement or space reduction
functions
if (KLAdjustmentParams.taskRun==1) then

--There is already going to be a reduction, so not all the adjustments will be reductions
KLAdjustmentParams.allEnlargement=false;

--So far all search space adjustments are reductions (including the one that is only going
to
--happen)
KLAdjustmentParams.allReduction=true;

end;

--Start of a search space reduction
xochz.KLspaceAdjustment.reduceSearchSpace();

--If there is an ideal number of hypothese found
elseif (taskDKKLCcount== inputParams.idealHypothesesCount) then

lm.log("Ideal number of hypotheses!");

--The task is added to the final task group for report creation and the task search space
--adjustment is ended.
```

```
taskDomainKnowledgeKLCondition.setTaskGroup(finalTasks);
lm.log("End of task "..taskDomainKnowledgeKLCondition.Name);
end;



end;

--Function to enlarge the search space
function xochz.KLspaceAdjustment.enlargeSearchSpace()

lm.log("-->I am enlarging the search space");

--If all adjustments so far were enlargements...
if (KLAdjustmentParams.allEnlargement==true) then

--...then use this formula to calculate new K parameter value
KLAdjustmentParams.K=KLAdjustmentParams.lastK / 2;
end;

--If all adjustments so far were not enlargements...
if (KLAdjustmentParams.allEnlargement==false) then

--...and the last adjustment was enlargement...
if (KLAdjustmentParams.lastSearch == "enlargement") then

--...then use this formula to calculate the new K parameter value
KLAdjustmentParams.K=(KLAdjustmentParams.lastK + KLAdjustmentParams.turnK)/ 2;

--If last search space adjustment was reduction...
elseif (KLAdjustmentParams.lastSearch == "reduction") then

--... then change the parameter turnK...
KLAdjustmentParams.turnK= KLAdjustmentParams.preLastK;

--...and calculate the new K parameter value
KLAdjustmentParams.K=(KLAdjustmentParams.lastK + KLAdjustmentParams.turnK)/ 2;

end;
end;

--If the difference between lastly used K parameter value and current new K parameter value
is
--less than 0.04...
if (math.abs(KLAdjustmentParams.lastK-KLAdjustmentParams.K)<0.04) then

--...then inform that the change is too insignificant, use the lastly run task for final
report
--and end adjustments of the current task
lm.log("-->Change in task parameters is insignificant");


--Placing the task into final task group to by displayed in the final report
taskDomainKnowledgeKLCondition.setTaskGroup(finalTasks );
lm.log("End of task "..taskDomainKnowledgeKLCondition.Name);

--If the If the difference between lastly used K parameter value and current new K parame-
ter
--value is not insignificant...
else

--...update the parameters...
KLAdjustmentParams.taskRun=KLAdjustmentParams.taskRun + 1;
KLAdjustmentParams.allReduction=false;
KLAdjustmentParams.lastSearch="enlargement";
--... create a clone of the task with a number of the current run in its name...
taskDomainKnowledgeKLCondition=taskDomainKnowledgeKLCondition.clone();
taskDomainKnowledgeKLCondition.setName (taskDomainKnowledgeKLCondition.Name.."
"..KLAdjustmentParams.taskRun);

--...and set the new K parameter as Kendall quantifier value for the new task
local DKKLCsetting=taskDomainKnowledgeKLCondition.findKLQuantifierSetting ({

nKLQuantifierTypeCode=lm.codes.KLQuantifierType.Kendall
});
lm.log("-->The new K is "..KLAdjustmentParams.K);
```

```
DKKLCsetting.setThresholdValue(KLAdjustmentParams.K);

--Run the newly created task
taskDomainKnowledgeKLCondition.runAndWaitForResults ({});

--Partially reloads the metabase with results to preserve other objects (tasks, prepro)
--already used in the Lua script
lm.metabase.reloadResults();

--Run the space adjustment process again for the lastly run task
xochz.KLspaceAdjustment.run(taskDomainKnowledgeKLCondition, KLAdjustmentParams);
end;


end;


--Function to reduce the search space
function xochz.KLspaceAdjustment.reduceSearchSpace()

lm.log("-->I am reducing the search space");

--If all adjustments so far were reductions...
if (KLAdjustmentParams.allReduction==true) then

--then use this formula to calculate the new K parameter
KLAdjustmentParams.K=(KLAdjustmentParams.lastK + 1) / 2;

end;

--If all adjustments so far were not reductions...
if (KLAdjustmentParams.allReduction==false) then

--...and the lastly run adjustment was reduction...
if (KLAdjustmentParams.lastSearch == "reduction") then

--...then use this formula to calculate the new K parameter value
KLAdjustmentParams.K=(KLAdjustmentParams.lastK + KLAdjustmentParams.turnK)/ 2;

--If last search space adjustment was enlargement...
elseif (KLAdjustmentParams.lastSearch == "enlargement") then

--... then change the parameter turnK...
KLAdjustmentParams.turnK= KLAdjustmentParams.preLastK;

--...and calculate the new value for the K parameter
KLAdjustmentParams.K=(KLAdjustmentParams.lastK + KLAdjustmentParams.turnK)/ 2;

end;
end;


--If the difference between lastly used K parameter value and current new K parameter value
is
--less than 0.04...
Kdifference=math.abs(KLAdjustmentParams.lastK - KLAdjustmentParams.K);

if (Kdifference <0.04) then

--...then inform that the change is too insignificant, use the lastly run task for final
report
--and end adjustments of the current task
lm.log("-->Change in task parameters is insignificant");

--Placing the task into final task group to by displayed in the final report
taskDomainKnowledgeKLCondition.setTaskGroup(finalTasks );
lm.log("End of task "..taskDomainKnowledgeKLCondition.Name);

--If the If the difference between lastly used K parameter value and current new K parame-
ter
--value is not insignificant...
else
lm.log("-->The new K is "..KLAdjustmentParams.K);

--...update the parameters...
KLAdjustmentParams.taskRun=KLAdjustmentParams.taskRun + 1;
KLAdjustmentParams.allEnlargement=false;
```

```
KLAdjustmentParams.lastSearch="reduction";

--... create a clone of the task with a number of the current run in its name...
taskDomainKnowledgeKLCondition=taskDomainKnowledgeKLCondition.clone();
taskDomainKnowledgeKLCondition.setName (taskDomainKnowledgeKLCondition.Name.."
"..KLAdjustmentParams.taskRun);

--...and set the new K parameter as Kendall quantifier value for the new task
local DKKLCsetting=taskDomainKnowledgeKLCondition.findKLQuantifierSetting ({

nKLQuantifierTypeCode=lm.codes.KLQuantifierType.Kendall
});
DKKLCsetting.setThresholdValue(KLAdjustmentParams.K);

--Run the newly created task
taskDomainKnowledgeKLCondition.runAndWaitForResults ({});

--Partially reloads the metabase with results to preserve other objects (tasks, prepro)
--already used in the Lua script
lm.metabase.reloadResults();


--Run the space adjustment process again for the lastly run task
xochz.KLspaceAdjustment.run(taskDomainKnowledgeKLCondition, KLAdjustmentParams);

end;


end;


return xochz.KLspaceAdjustment;
```

XochzResults MODULE

```
--This module is resposible for creating a final report in an HTML file. This module was
taken ----from the
--EverMiner project's module EMSResults and only some slight adjustments has been made.

xochz.results = {};

--Local functions

function xochz.results.getAttributeGroupStr( inputParams)

str= table.concat( inputParams.domainAttributeGroups, ", ");
return str;

end;

function xochz.results.getAttributeList( attributeGroup)

attributeArray= attributeGroup.prepareAttributeArray({
pDataTable= dataTable
});

str= "";
for i, attribute in ipairs( attributeArray) do

if ( i > 1) then str= str..", "; end;
str= str.."<code>";
str= str..attribute.Name;
str= str.."</code>";

end;

return str;
end;

function xochz.results.getCategoryList( attribute)

categoryArray= attribute.prepareCategoryArray();

str= "";
```

```
for i, category in ipairs( categoryArray) do

if ( i > 15) then
--Too many categories, report just the first few
str= str.."...";
break;
end;

if ( i > 1) then str= str..", "; end;
str= str.."<code>";
str= str..category.NameHTML;
str= str.."</code>";

end;

return str;
end;

function xochz.results.exportReport( inputParams)
--Export an analytical report with results

bOpened= false;
if ( not lm.metabase.isOpen()) then

lm.metabase.restoreMDB({
pathNameSrc= inputParams.pathNameMetabase.."_bkup.tasks.mdb",
pathNameDest= inputParams.pathNameMetabase
});

lm.metabase.open({
dataSourceName= xochz.import.getMetabaseDSN( inputParams)});
bOpened= true;
end;

lm.setLogVerbosityLevel( lm.codes.LogVerbosityLevel.Normal);
lm.setIsLogFunctionParameterValues( false);

lm.log( "Creating analytical report");
lm.logIndentUp();

outputFile= io.open( inputParams.pathNameReportOutput, "w");

outputFile:write( "<!DOCTYPE html>\n");
outputFile:write( "<html lang=\"en\">\n");
outputFile:write( "<head><link rel=\"stylesheet\" href=\"ems.css\" type=\"text/css\"
/></head>\n");

outputFile:write( "<body>\n");

dataTable= lm.explore.findDataTable({
name= inputParams.tableName
});
assert( dataTable, "Database table not found!");

outputFile:write( "<center><font size=\"+3\">Automation Analytical Re-
port</font></center>\n");
--outputFile:write( " <p> \n");

--Input parameters

outputFile:write( "<h1>Input parameters</h1>\n");

outputFile:write( "<p>Source data file:
<code>"..inputParams.pathNameDataSrc.."</code><br>\n");
outputFile:write( "<p>Domain knowledge (attribute groups):
<code>"..xochz.results.getAttributeGroupStr( inputParams).."</code><br>\n");
outputFile:write( "<p>Preferred number of patterns to be found:
<code>"..inputParams.idealHypothesesCount.."</code><br>\n");
outputFile:write( "<p>Report file destination:
<code>"..inputParams.pathNameReportOutput.."</code><br>\n");

--Database Tables

outputFile:write( "<h1>Data Exploration</h1>\n");

line= string.format( "<h2>%s</h2>\n",
dataTable.Name
```

```
);
outputFile:write( line);

--outputFile:write( "<h2>Original columns</h2>\n");
outputFile:write( "<p>\n");
outputFile:write( "Original columns:<br><ul>\n");

dataColumnArray= dataTable.prepareDataColumnArray();

for i, dataColumn in ipairs( dataColumnArray) do

if ( (dataColumn.getDataColumnSubTypeCode() == lm.codes.DataColumnSubType.Ordinary) and
(not dataColumn.isPrimaryKeyPart())) then

line= string.format( "<li>%s: %s</li>\n",
dataColumn.Name,
dataColumn.getValueSubTypeName()

);
outputFile:write( line);

end;
end;

outputFile:write( "</ul>\n");

--outputFile:write( "<h2>Original columns</h2>\n");
outputFile:write( " <br><p>\n");
outputFile:write( "Derived columns:<br><ul>\n");

dataColumnArray= dataTable.prepareDataColumnArray();

for i, dataColumn in ipairs( dataColumnArray) do

if ( dataColumn.getDataColumnSubTypeCode() ~= lm.codes.DataColumnSubType.Ordinary) then

line= string.format( "<li>%s: %s</li>\n",
dataColumn.Name,
dataColumn.getValueSubTypeName()

);
outputFile:write( line);

end;
end;

outputFile:write( "</ul>\n");

--Attribute groups

outputFile:write( "<h1>Data Preprocessing</h1>\n");

rootAttributeGroup= lm.prepro.getRootAttributeGroup();

attributeGroupArray= rootAttributeGroup.prepareSubAttributeGroupArray();

for i, attributeGroup in ipairs( attributeGroupArray) do

--attributeList= ems.results.getAttributeList( attributeGroup);

line= string.format( "<h2>%s</h2>\n",
attributeGroup.Name
);
outputFile:write( line);

--list of attributes

attributeArray= attributeGroup.prepareAttributeArray({
pDataTable= dataTable
});

str= "";
for i, attribute in ipairs( attributeArray) do

categoryList= xochz.results.getCategoryList( attribute);

line= string.format( "<p>%s: %s<br>\n",
```

```
attribute.Name,
categoryList
);
outputFile:write( line);

end;

end;

--Tasks

outputFile:write( "<h1>Tasks</h1>\n");

taskArray= lm.tasks.prepareTaskArray({
pDataTable= dataTable
});

for i, task in ipairs( taskArray) do

--finalHypothesisGroup= task.findHypothesisGroup({
--name= "Final"
--});

if ( task.TaskGroup.getName() == "Final") then
--final task

--reportTaskName= string.match (task.getName(),"%D+" );

line= string.format( "<h2>%s</h2>\n",
task.getName()
);
outputFile:write( line);

nHypoTotalCount= task.getHypothesisCount();
if (nHypoTotalCount == inputParams.idealHypothesesCount)  then

outputFile:write( "<p>Task finished succesfully and an acceptable number of patterns has
been found<br>\n");

else

outputFile:write( "<p>Task finished but an acceptable number of patterns has not been
achieved<br>\n");

if ( task.Note ~= "-") then
outputFile:write( "<p>"..task.Note.."<br>\n");
end;

end;


iterationCount = string.sub(task.getName(), -1, -1 );    --string.sub(s, 2, -2)
if (string. match (iterationCount,"%D")~=nil) then
iteCount=1;

else
iteCount=iterationCount;
end;


--%D will match all non-digit characters.



--iterationStart, iterationEnd= string.find( task.Name, "%(%*d%)");
--if ( iterationStart~=nil) then

--iterationCountStr= string.sub( task.Name, iterationStart+ 1, iterationEnd- 1);
--iterationCount= tonumber( iterationCountStr);

outputFile:write( "<p>Number of iterations: "..(iteCount).."<br>\n");
--end;

nHypoCount= task.getHypothesisCount();
if ( nHypoCount > 0) then
```

```
line= string.format( "<p>Found patterns: %d<br>\n",
task.getHypothesisCount()
);
outputFile:write( line);

outputFile:write( "<p>\n");
outputFile:write( "The most interesting ones:<br><ul>\n");

hypothesisArray= task.prepareHypothesisArray();

for j, hypothesis in ipairs( hypothesisArray) do

--task with results title
line= string.format( "<li>%s</li>\n",
hypothesis.TextHTML
);
outputFile:write( line);

end;

outputFile:write( "</ul>\n");

else

outputFile:write( "<p>No interesting results found.<br>\n");

end;

end;
end;


if ( bOpened) then

lm.metabase.close();

--Create a backup copy for debug
--lm.metabase.backupMDB({
--pathNameSrc= inputParams.pathNameMetabase,
--pathNameDest= inputParams.pathNameMetabase.."_bkup.results.mdb"
--});

end;

outputFile:write( "</body>\n");
outputFile:write( "</html>\n");

io.close( outputFile);

lm.logIndentDown();

end;

return xochz.results;
```