12

EverMiner – Towards Fully Automated KDD Process

M. Šimůnek and J. Rauch
Faculty of Informatics and Statistics, University of Economics, Prague, Czech Republic

1. Introduction

A man-controlled data-mining process has its limits – there is a limited number of data mining tasks an user is capable to create, a limited number of results he or she is able to digest, a limited number of task parameters changes he or she is able to try to get better results and so on. Significantly improved results from data mining could be in contrast obtained from huge amount of data-mining tasks run automatically in iteration steps with changing task parameters. These changes are based on results from previous runs combined with background knowledge about given domain. Not only task parameters but also types of patterns the automated process is looking for could be influenced by previous results and changed during iterations.
This text specifies further and in more details the thoughts published in previous works of [Rauch & Šimůnek, 2005a], [Rauch & Šimůnek, 2007], [Rauch & Šimůnek, 2009a] and mainly in the paper [Rauch, 2010] where a formal architecture of the automated data-mining process was firstly proposed.
The text is organized as follows. There are main research-goals presented in the next section together with references to the work that has been already done, mainly in form of the academic KDD system LISp-Miner (see [Šimůnek, 2003], [LISp-Miner]) and tightly related project SEWEBAR for dealing with background knowledge that is necessary for every successful data mining process (see [Kliegr et. al., 2009], [SEWEBAR]). The third section introduces the EverMiner system and its global concept of two loops and its phases. These particular phases are described then in details in the fourth section of this chapter. Conclusions and plans of future work are outlined in the fifth section. The text ends with acknowledgements and references of cited works.

2. Prerequisites

The KDD research on the UEP started in the year 1995 and we are developing since the LISp-Miner system – an academic system for KDD, see [LISp-Miner]. It consists now of eight data mining procedures mining for syntactically rich patterns with much higher possibilities to express relations in analysed data than the simple *association rules* proposed by [Agrawal et. al., 1993]. Instead, those procedures are based on an original Czech data-mining method called GUHA (see [Hájek & Havránek, 1978]) with a deep theoretical background and history of development since 1966. Thus, the LISp-Miner system incorporates a core data-

mining algorithm with greater capabilities than the *a-priori* algorithm proposed by [Agrawal et. al., 1996]. For details about alternative approach to mine for patterns with a rich syntax see [Rauch & Šimůnek, 2005b] and [Hájek et. al., 2010]. A new feature to run also data-mining tasks remotely on a computer grid was implemented recently into all the procedures of the LISp-Miner system.

We concentrate now in our research on the significant problems of today's KDD, mainly to offer solutions for:

- to present results of data mining to data owners in a readable form, especially in the form of analytical reports;
- to incorporate background/domain knowledge to (a) formulate reasonable *Local Analytical Questions* (LAQ, see section 4.2 later in this chapter) to be answered by data mining tasks and (b) to prune results of trivial or of already-known facts;
- to automate the whole process of KDD, mainly its three phases – *Data Preprocessing*, *Data Analysis* and *Results Interpretation*. The EverMiner project (see the next section) has been started to deal with this problem especially.

For the whole automation to be feasible there are some necessary prerequisites that had to be accomplished first:

1. Long-time experiences with solving different types of data-mining tasks and a large set of heuristics and hints how to pre-process data and how to fine-tune task parameters to obtain suitable amount of valuable results.
2. Theoretical background of mathematical logic with a language to describe properties of mined syntactically rich patterns and with logically valid rules for deduction and induction of new knowledge based on already known facts and newly mined patterns.
3. Implemented portfolio of data-mining procedures mining for different kinds of patterns. The most suitable one of these procedures will be chosen in each step of iteration to answer given *Local Analytical Question*.
4. Implemented computer grid feature to solve many tasks simultaneously (and possibly very fast) on a computer grid (dedicated one or an one consisting of ordinary PCs linked together).
5. A *Knowledgebase* where all the above-mentioned heuristics, rules etc. are stored together with the already known domain background knowledge collected previously from domain specialists.
6. Good approach to present the mined results in a human readable form to domain specialists (and in a way suitable for them).

Eight data mining procedures were implemented already in the LISp-Miner system and they are proved through many years of using in data-mining analysis, both the real world and academic ones in teaching courses (see e.g. [Lín et. al., 2004], [Rauch et. all., 2005], [Rauch & Šimůnek, 2005b], [Rauch & Šimůnek, 2005c]). Sets of heuristics and rules for data-mining process automation were proposed and a theoretical logical backgrounds were established (see e.g. [Rauch, 2005], [Rauch, 2009]). Distributed solving of data mining tasks using the computer grid was implemented (see [Šimůnek & Tammisto, 2010]). The first version of the *Knowledgebase* for storing domain knowledge has been built within the LISp-Miner *LM DataSource* module (see [Rauch & Šimůnek, 2008]) and is now refined further in cooperation with the SEWEBAR project (see e.g. [Kliegr et. al., 2009]). Functions were implemented for an automated export of mined results into the SEWEBAR to be published in form of analytical reports. So the logical step now is to combine all the already existing parts together and to start truly automated KDD process.

The above mentioned complex patterns and rich syntax offered by those already implemented data-mining procedures of the LISp-Miner system have a good potential for fine-tuning of the task parameters and for types of mined patterns to be chosen from in each step. We see a great benefit of allowing these DM procedures to crawl automatically through the analysed data and to digest true nuggets by several iterations of answering analytical questions and provide newly found knowledge to domain specialists.
But we admit also that fulfilling this goal is not an easy task to do and there are several related problems that had to be solved. Among others, it is to provide results to the domain specialist in an understandable form. A tightly related SEWEBAR project (see section 3.5 later in this text) is aiming at this problem and is already delivering the first results in improving communication with domain experts in both directions (i.e. gathering already known facts from specialists and delivering results using analytical reports in the opposite direction).

3. EverMiner

The EverMiner project aims at developing such a system that would automatically run many data-mining tasks in several iterations and will possibly find interesting results without any user interaction. The main goals of the EverMiner project are:

1. To mine automatically in the data for all the hidden patterns which were not discovered yet and for which it is a great potential that they will be of some interests for domain specialists (i.e. not to mine obvious or already-known facts nor their consequences).
2. To free data-miners from majority of the necessary work and time spent during the KDD process. And to allow even the domain specialists themselves to do successful data-mining analysis without any need to have the analytical know-how that is necessary to discover some really useful new knowledge but which an ordinary domain specialist is not willing to learn (or has no time to learn to work with a data-mining tool).

Let us remind that a similar project has been proposed already under the name of GUHA80 and mentioned in [Hájek & Havránek, 1982] and [Hájek & Ivánek, 1982] but has been never realized. The project presented here differs completely from the GUHA80, albeit it is based on the GUHA method too.
The global concept of the EverMiner is in Fig. 1. Analysed data provide input for the data-mining process although they need to be pre-processed first. The most important property of the proposed automated data mining is its cyclic nature. Immediately after the *Data Preprocessing* phase, a main cycle of data-mining process begins (the *Outer Loop*) with an inner loop inside for a fine-tuning of the currently processed task parameters to obtain a reasonable amount of patterns (at least some, but not too many).
A brief description follows of all the phases presented in Fig 1. A detailed explanation is in the section 4.

3.1 Analysed data

Generally, any empirical data obtained through experiments, observations or transactional databases that we want to analyse and to uncover interesting relationships in them. In this text we suppose that they represent a finite many properties of finite many objects and that they are store in a table of a relational database. We expect also that the data concern some given domain where a set of typically involved properties could be identified and formally

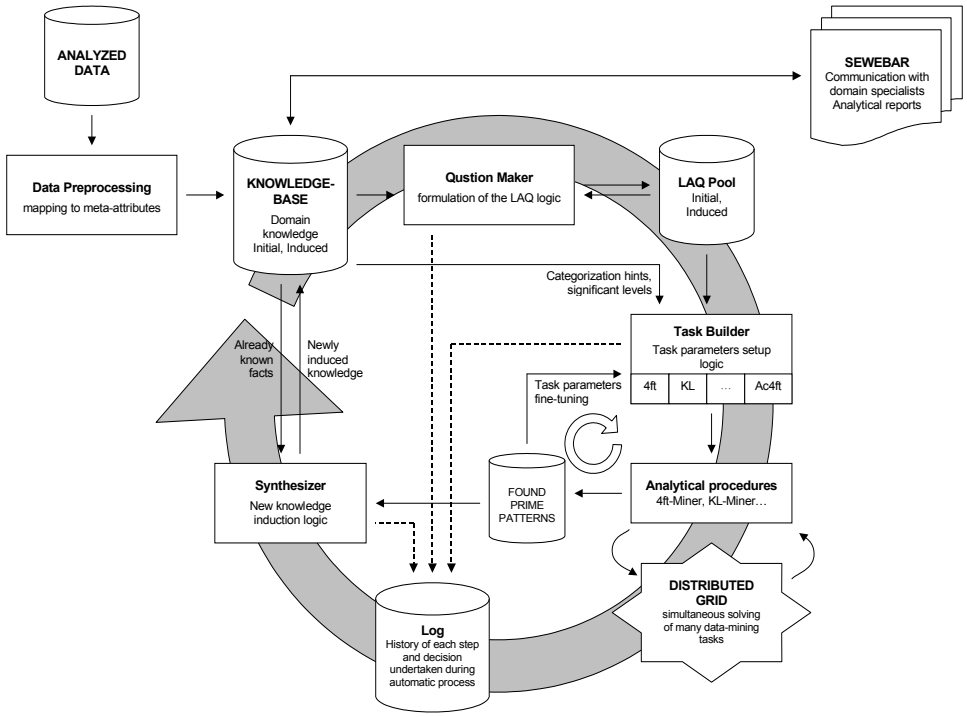


Fig. 1. Global concept of the EverMiner

described. Examples of analysed data are transactions on bank accounts, polling results, standardized records of patients etc.

3.2 Data preprocessing

Data Preprocessing phase is equally important for an automated process as it is for a manually done data-mining analysis. Understanding of analysed data structures and suitable transformations of data have significant influence on quality of the data-mining analysis results. The most important parts are mapping of analysed data columns to meta-attributes defined in the *Knowledgebase* and categorization of values based on the hints – for details see section 4.1 further in this text.

This phase is not part of the main cycle so user-feedback could be requested in case of an ambiguity – e.g. a column good be mapped to more than one meta-attribute or two possible categorization hints could be applied.

3.3 Outer loop

The *Outer Loop* is inspired by the main phases of the KDD process as described e.g. in the CRISP-DM methodology [CRISP-DM] – i.e. the *Domain Understanding*, *Data Transformation*, *Analytical Procedures* and *Results Interpretation* – and its cyclical nature.

Each of iterations of this cycle takes into account current state of the domain knowledge (stored in the *Knowledgebase*) and tries to mine new knowledge mimicking steps of the man-controlled data-mining analysis done by a user:

The *Question Maker* module formulates set of *Local Analytical Questions* (LAQ) based on the current level of knowledge. *LAQs* are stored into the *LAQ Pool* where they wait to be processed. (There could be also an initial set of *LAQs* provided manually.) The *Task Builder* module takes the *LAQs* one by one from the *LAQ Pool* and creates a data-mining task(s) to answer them in the same way as would be case in a man-controlled data-mining. Results in form of found prime patterns are handed to the *Synthetizer* module. Here, they are examined if they do not follow from some already known facts. Only those really novel patterns are added into the *Knowledgebase* and eventually reported to domain specialist in form of *Analytical Reports* via the SEWEBAR. Changes to the *Knowledgebase* make possible another set of *LAQs* to be formulated by the *Question Maker* module (e.g. "*Are they any exceptions to newly found patterns?*") and these new *LAQs* are again stored into the *LAQ Pool*. The *Task Builder* module periodically checks the *LAQ Pool* for not-yet-answered *LAQs* and the whole process is repeated. The *Outer Loop* ends if there are no further *LAQs* in the *LAQ Pool*.
There could be more than one data-mining task necessary to answer a single *LAQ*. Actual number of tasks and types of patterns it is looking for will depend on structure and complexity of the processed *LAQ*. Generally, the automated data mining proposed here will lead to a very large number of data-mining tasks to be solved by different GUHA-procedures for each step of the *Outer Loop*, in contrast to much lower number of data-mining tasks ever created by users during some data-mining analysis done manually.

3.4 Inner loop

Initial task parameters setting could be done only roughly, as is the case even for a man-controlled data-mining analysis. The precise settings could be chosen only after task is solved for many times and the data-miner could see number and quality of found patterns. It has therefore an iterative nature.
The *Inner Loop* takes care of this iterative nature while looking for the right task parameters. Results of every single task run are checked and task parameters could be tweaked if there are no patterns found (parameters are too strict) or too many of them are found (parameters are too loose). It is important to say that the background logic of implemented GUHA-procedures already filters found patterns to be prime only – not easily deducted from other already found patters (for details see e.g. [Rauch, 2005]).

3.5 SEWEBAR

The aim of the SEWEBAR project is to develop a framework of the same name that acts as a platform to illicit the background knowledge from domain specialists and – in an opposite direction – to present mined results in an understandable way (in form of structured analytical reports). There has been developed a specialised BKEF XML schema for storing background knowledge and an open-source content management system (CMS) called *Joomla* is used for preparation of structured analytical reports using created authoring tools.
The SEWEBAR project is tightly related to the KDD research at Department of Knowledge Engineering of the University of Economics Prague, but is independent of the LISp-Miner and the EverMiner projects. For more details about the SEWEBAR project see [SEWEBAR], [Kliegr et. al., 2009] and [Balhar et. al., 2010].
There is a two-way communication established between the EverMiner and the SEWEBAR framework. All the necessary knowledge will be provided automatically by the SEWEBAR to the EverMiner at request. Simultaneously, the EverMiner will send all the results and all the updates of the *Knowledgebase* (newly found knowledge about given domain) to the

SEWEBAR to be presented to domain specialists in easily understandable form of analytical reports and of graphical visualisations of the *Knowledgebase*.

4. Detailed description of the proposed phases

Each of the above-mentioned proposed phases is described in the following sub-sections.

4.1 Domain knowledge and data preprocessing

The *Knowledgebase* (i.e. the *knowledge repository*) contains all the available knowledge about given domain that was either provided by domain specialists (as a results of the domain specialists communication with the SEWEBAR before the beginning of the analysis) or induced from the results of automated iterations of the EverMiner's *Outer Loop*.
There are several levels of knowledge present in the *Knowledgebase* and described in the following paragraphs (see also [Rauch and Šimůnek, 2009], [Rauch, 2010]).

4.1.1 Meta-attributes

The basic level of knowledge consists of meta-attributes. The meta-attributes are typical properties (characteristics) of objects for a given domain. They are identified in advance by domain specialists and contain the following knowledge:

- Name (of list of possible typical names) of the concerned attribute in future analysed data.
- Cardinality type of values stored (*nominal*, *ordinal*, *cardinal*) and expected *Data type* of values (*integer*, *float*, *text*, *date*, *Boolean*).
- Categorization hints – describing either enumeration of typical values (e.g. *Sun*, *Mon*, *Tue*, …, *Sat*) or allowed range for numerical values and proposed lengths of intervals to categorize them. There could be several categorization hints provided for a single meta-attribute to differentiate among different goals of analysis.
- Important values (i.e. significant levels) that have some special meaning (e.g. 100 °C as a threshold for boiling water). They could be used for categorization or setting-up data-mining tasks (e.g. "*Are there any exceptions for X when Temperature is below 100)?*") because domain specialists are used to them and interpretation of results with such thresholds is easily understandable for them.

All this information about meta-attributes is used then to map (automatically) database columns from the analysed data to meta-attributes. It is necessary to find an appropriate meta-attribute for each database column to do a proper pre-processing of database column values – i.e. to categorize them accordingly to the categorization hints provided.
Mapping columns to meta-attributes is difficult given availability of typical names and data-types of concerned meta-attributes only. There will be lots of ambiguities and missing definitions, especially for a newly created *Knowledgebase*. Fortunately, the *Data Preprocessing* phase is not part of the *Outer Loop* so it is possible to ask user for feedback to resolve such an ambiguity or to update definition of meta-attributes if there is no appropriate meta-attribute to map a database column to.

4.1.2 Hierarchy of meta-attributes

Defined meta-attributes are associates with meta-attribute *groups*. Each meta-attribute must have its *Basic group* associated. A meta-attribute could moreover belong to an unlimited number of other meta-attribute *groups* too. Therefore it is possible to group meta-attributes

according to several criterions at once. Attributes created from database columns of the analysed data inherit associations to *groups* from its master meta-attribute on behalf of which they were created.
Groups are organized in a hierarchy with the *Root-Group* at the top of it and each subsequent *group* having its *Parent group* defined to be either the *Root-Group* or any other *group* already present in the hierarchy. This hierarchical structure allows for a clever using of *groups* in further processing (e.g. while formulating the *LAQs* – see the next section). So every mention of a *group* covers simultaneously all the attributes belonging directly to the *group* plus others in any *Child-group* having the given *group* as its *Parent group*.

4.1.3 Mutual dependency

Mutual dependency aims at visualizing relationships among meta-attributes in an easily and understandable way for domain experts. Not only known dependencies among meta-attributes are recorded but also clues for what are users interested in to be answered (and for what not). For any pair of meta-attributes (or sometimes for the whole *groups* of attributes) we could specify:

- *Type of dependency* … describes (based on the best-available knowledge of domain experts) possible dependency between values of the first meta-attribute and values of the second meta-attribute. Examples are:
  - $\uparrow\uparrow$ … a positive influence $\Rightarrow$ the higher values of the first meta-attribute the higher values of the second meta-attribute; applicable for cardinal or ordinal meta-attributes only.
  - $\uparrow\downarrow$ … a negative influence $\Rightarrow$ the higher values of the first meta-attribute the lower values of the second meta-attribute; applicable for cardinal or ordinal meta-attributes only.
  - $\mathcal{F}$ … a function-like dependency, e.g. "*velocity*" is defined as $\frac{1}{2}\, a\, t^2$ where "a" is acceleration and "t" is time. So "velocity" is dependent on "*acceleration*" and "*time*" in a function-like manner; applicable for cardinal attributes only.
  - $\uparrow+$ … a positive increase of relative frequencies $\Rightarrow$ the higher values of the first meta-attribute (cardinal or ordinal) the higher probability of occurrence of some property described by the second meta-attribute (Boolean).
  - ? … an unknown influence, need to be specified by future analysis.
  - – … no influence at all.
  - $\approx$ … there is some influence but is not specified further (yet).
  - $\otimes$ … we are not interested in determining type of influence between this pair of *meta-attributes*.
- *Scope* … whether such dependency has a global scope (has been observed in the whole domain) or is supposed to be specific only to the current sample of analysed data. Options are:
  - Data specific
  - Domain specific
- *Validity* … whether the knowledge is supported by the results of the data-mining analysis. This allows to track progress in changes of the domain knowledge. Options are:
  - Proven
  - Rejected

- Unknown

Several two-dimensional grids containing selected subsets of attributes could be constructed where graphical symbols do express possible options of mutual dependencies among them – see example in Fig. 2.

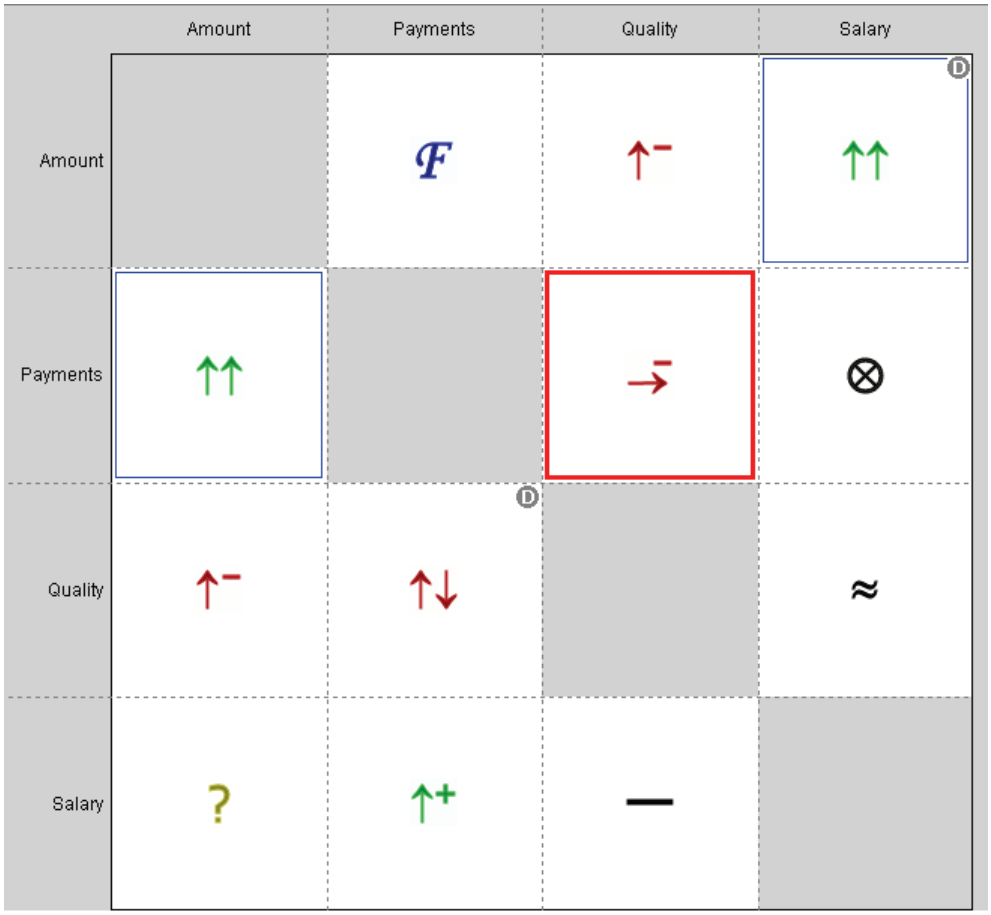


Fig. 2. Grid of mutual dependencies among subset of meta-attributes (from the *LM DataSource* module)

Proven dependencies are framed in blue, rejected dependencies in red. Dependencies with a limited scope of validity for the analysed data only are marked with the (D) sign – i.e. dependencies that do not hold globally but only for given sample of the analysed data (e.g. due to conditions how data were collected or pre-processed).

This visual presentation helps to gather as much information about given domain as possible because it seems that domain experts are keen with expressing their knowledge this way and they are happy even to input data themselves in a suitable (internet) application. It is especially important while presenting a newly found knowledge when experts are expected to approve it and to give a necessary feedback.

Although the above-mentioned *Knowledgebase* was implement first in the *LM DataSource* and the *LM KnowledgeSource* modules of the LISp-Miner system (see [Rauch & Šimůnek, 2008] and [LISp-Miner]) its future development was moved now into the SEWEBAR project.

4.1.4 Meta-attribute distribution

Another type of knowledge – about every single meta-attribute – could be included in the *Knowledgebase* too. This kind of knowledge represents conditional distributions of frequencies of values or big differences of types of distributions between two subsets of analysed data. For example, the *Education level* could be of a *Gaussian* distribution among the whole population but could be expected to be skewed towards higher levels of education in the capital/university city where highly educated people are of high demand. This type of knowledge could be used again both for the formulation of the *LAQs* and for the pruning results of well-known facts.

This type of knowledge is related to the *CF-Miner* and *SDCF-Miners* data-mining procedures of the LISp-Miner system and needs to be investigated further.

4.2 Question maker and local analytical questions

The *Question Maker* module formulates new *Local Analytical Questions* (*LAQs*) based on the current level of domain knowledge and newly mined knowledge in previous cycles. Any *LAQ* describes a question user wants to answer in a formalized way but using plain language so the *LAQ* is understandable to domain experts. Examples of *LAQs* are:

- "Are there any relations between characteristics of *Body-Mass-Index* and *Success of therapy* in analysed data?"
- "Are there any exceptions to the patterns found for the previous question depending on level of *Physical activity* when these patterns do not hold?"

Not only single attributes could be used to formulate a *LAQ* – the whole groups of meta-attributes could be instead to formulate more general analytical questions:

- "Are there any relations between *Social characteristics* of patients and *Cardiovascular risk factors*?"
- "Are there any exceptions to the patterns found for the previous question depending on *Physical activities* when these patterns do not hold?"

4.2.1 LAQ templates

After several years of experiences with data mining analyses we have observed that there is only a limited number of *LAQ* types that are typically formulated. There is of course infinite number of particular groups or even attributes that could appear in *LAQs* but the skeleton of the *LAQ* remains still the same. Thus, it is possible to prepare reasonably small number of so called *LAQ Templates* with active positions prepared where a particular group of attributes could be inserted to formulate a proper *LAQ*. Examples of such *LAQ Templates* are:

- "Are there some strong relationships in sense of <*interest measure*> between patients characteristics given by <*group of attributes A*> and by <*group of attributes S*> in subsets of data given by Boolean conditions based on <*group of attributes C*>? We are however interested in already known and proven facts."
- "Are there any exceptions to the patterns found for the previous question depending on <*group of attributes C*> when these patterns do not hold?"

It is easy then to formulate new *LAQs* given the pre-defined *LAQ Templates*, list of possible *groups of attributes* (available from the background knowledge of concerned domain) and list of available *interest measures* to describe type of looked-for relationships in data.
List of available *LAQ Templates* is not strictly closed and a new template could be added if a new type of typical user's questions emerges. List of available *groups of attributes* is provided by the *Knowledgebase* and it contains additional information about each attribute (derived from its associated master meta-attribute) – e.g. its *cardinality type* (whether its values are *nominal*, *ordinal* or *cardinal*). This could restrict positions in *LAQ templates* where such attribute could be used.
A newly created *LAQ* should be checked against already accumulated domain knowledge (both the initial and induced). It is not necessary to proceed to data-mining phase if the *LAQ* could be answered (either directly or by a deduction) from the already known facts in the *Knowledgebase*. Otherwise, a new data-mining task will be created based on it and solved.
Initially, a pool of unanswered *LAQs* will consists of the *LAQs* prepared in advance by the user, if they are such. Next, the *Question Maker* will create as many further *LAQs* as possible given the actual content of the *Knowledgebase*. Together they will be sent to the *Task Builder* module (see the next sub-section) to find answers to them. Found answers to given *LAQs* will eventually lead to a new knowledge and these new facts will be added into the *Knowledgebase*. This might offer a new possibility for the *Question Maker* module to formulated another *LAQ(s)* – e.g. using the "*are there any exceptions to newly found patterns*?" *LAQ Template*. Therefore, the actual number of unanswered *LAQs* in the *Pool* fluctuates as a *LAQ* is removed when corresponding data-mining task(s) is/are created and simultaneously as new *LAQs* are formulated as a reaction to a fresh knowledge in the *Knowledgebase* being newly induced from the results of just solved data-mining tasks.

4.2.2 LAQ grid

Again, a grid could be constructed to keep track of already processed *LAQs* – grid of combinations of available group of attributes (or its subset) – see Fig. 3.

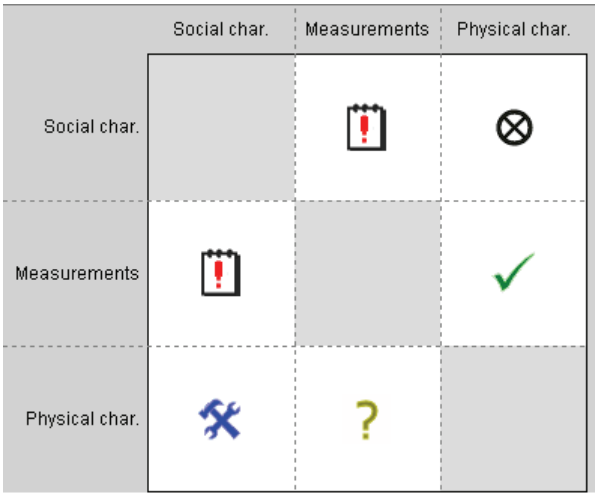


Fig. 3. *LAQ-template* Grid (from the *LM LAQManager* module)

So the user continually knows which *LAQs* are already solved (marked with ✓), which are just being processed now (marked with 🛠), which ones are waiting to be processed (marked with [!]) or we are not concerned in this *LAQ* (marked with ⊗).

4.3 Task builder

Once a *LAQ* is formulated it needs to be answered by results from solved data-mining task(s). One *LAQ* could be answered with a single task for a single analytical procedure or more tasks for different analytical procedures might be necessary to answer it.

Set-up of the task depends on structure of the *LAQ* it is supposed to answer and its structure is in turn influenced by the *LAQ Template* that was used to formulate the *LAQ* in the first place. There are prepared rules in advance how to set-up task(s) for given *LAQ Template*.

There are many task parameters available that influence significantly the size and the quality of mined results. These parameters include minimal and maximal lengths of Boolean attributes involved in constructing of patterns. Task parameteres include also criterions based on *interest measure(s)* to specify types and tightness of relationships inside to be found patterns.

First version of heuristics was prepared to guide the *Task Builder* module while setting of initial parameters for a new task. The heuristics are based on number of attributes and number of theirs categories and on experiences with expressing the most typical kinds of relationships inside patters – i.e. types of quantifiers used and suitable levels for theirs parameters – for details see the *Analytical procedures* section or [Rauch & Šimůnek, 2005b].

Once a task is created it is dispatched to a corresponding analytical procedure to be solved.

The *Task Builder* role is not limited to setting up tasks based on the given *LAQs*. It is also involved in the *Inner Cycle* of the EverMiner process – to fine-tune task parameters to get a reasonable number of results from the *Analytical procedures* phase to answer the given *LAQs*.

It is very often that either too many patterns are found given the initial settings of task parameters or that no pattern is found at all. This is typical even for man-prepared task parameters because of an unknown character of the analysed data and because of complexity of possible parameters settings. Thus, the data mining process has to be done iteratively so long as the right values of task parameters are found. The *Task Builder* module is equipped with another set of heuristics to be able to restrict searched solution-space (i.e. to decrease number of found patterns) or to enlarge it (i.e. to increase number of found patterns). The *Inner Loop* could be described in detail as visible in Fig. 4.

When the number of found patterns is too small (or no patterns found at all), task parameters are changed to be looser (to allow for not so strong patterns) or to search through an enlarged search-space (to prove more possible combinations). When the number of found patterns is too big, task parameters are made stricter so a smaller number of strong patterns will be found in the next round. This *Inner Loop* is repeated as many times as needed. There are of course limits to fine-tuning of parameters, especially when no patterns had been found, because it makes no sense to make task parameters very loose. Therefore, it is possible that no patterns for a given *LAQ* are supported by the analysed data and this particular *LAQ* is rejected.

It depends on the underlying *LAQ*, used analytical procedure and found types of patterns what is assumed to be an "acceptable" number of found patterns. Some kinds of relationships in the analysed data (e.g. of the $\uparrow\uparrow$ or the $\mathcal{F}$ mutual dependency) could be expressed with large number of *4ft-associational rules* in the *4ft-Miner* procedure compared to a single *KL-pattern* as a result of the *KL-Miner procedure*. As a rule of thumb an "acceptable" number of patterns is more than none and less than two hundred.

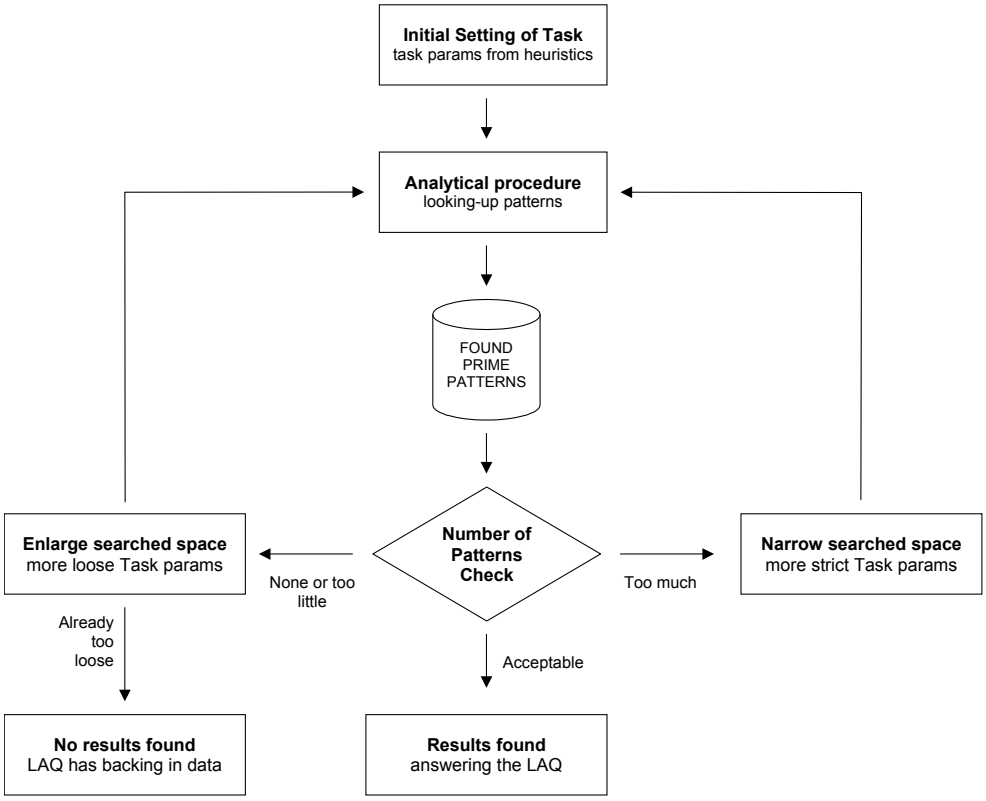


Fig. 4. Detail Description of the Inner Loop

4.4 Analytical procedures

There are seven analytical procedures implemented already in the LISp-Miner system:

- *4ft-Miner* procedure expressing found patterns as (conditional) *4ft-associational rules* with a rich syntax;
- *SD4ft-Miner* procedure looking for *SD4ft-patterns* – comparing two sub-sets in the analysed data in sense of two (conditional) *4ft-associational rules*;
- *Ac4ft-Miner* procedure looking for *4ft-action rules* describing some kind of action resulting in a change of characteristics in objects from the analysed data;
- *KL-Miner* procedure looking for KL-patterns in form of K×L contingency tables of two multi-categorical attributes and under some (richly defined) condition;
- *SDKL-Miner* procedure comparing two sub-sets in the analysed data in sense of two *KL-patterns*;
- *CF-Miner* procedure looking for *CF-patterns* in form of distribution of frequencies for a single multi-categorical attribute and under some (richly defined) condition;
- *SDCF-Miner* procedure comparing two sub-sets in the analysed data in sense of two *CF-patterns*.

All the implemented procedures are so called GUHA-procedures (in sense of [Hájek & Havránek, 1978]). Theirs input consists of the analysed data and a simple definition of a large space of potentially interesting patterns. And theirs output is set of all interesting patterns that are supported by the analysed data. For more detail see [Rauch & Šimůnek, 2005a], [Hájek et. al., 2010].

The most important feature is a really rich syntax of looked-up patterns that could be defined in relatively simple way. Each implemented data-mining procedure offers a rich syntax how to describe potentially interesting pattern we are looking for. They mine not for associational rules only but for an enhanced version called *4ft-asociational rules* (see [Rauch & Šimůnek, 2005a]) and for other types of patterns – e.g. conditional frequencies, K×L conditional frequency tables ([Lín et. al., 2004]), *Set-differs-from-set* (SD) rules or for *4ft-actional rules* (see [Ras & Wieczorkowska, 2000], [Rauch & Šimůnek, 2009b]). This rich syntax makes possible to involve semantically features of logical reasoning and deduction ([Rauch, 2009]).

4.4.1 Optimisation

Number of patterns in the search-space each analytical procedure has to walk-through is enormous, especially because of the rich syntax of mined-for patterns. Several optimisation techniques were incorporated therefore into analytical procedures implementations.

As an example of such an optimisation we could mention the *bit-string* structures for very fast computing of frequencies of derived *Boolean attributes* to construct contingency tables (for details see e.g. [Rauch & Šimůnek, 2005b]). For simplicity reasons we would discuss only *4ft-association rule* syntax that is used in the *4ft-Miner* procedure. A conditional *4ft-association rule* has form of:

$$\varphi \approx \psi \,/\, \chi$$

Where φ, ψ and χ are derived *Boolean attributes* automatically derived from basic *Boolean attributes* as theirs *conjunctions, disjunctions* and *negations*. The symbol $\approx$ is called *4ft-quantifier*. To compute frequencies from contingency table we need to know frequencies of each derived *Boolean attribute* and hence frequencies of concerned basic *Boolean attributes*. Values of each *Boolean attribute* in the analysed data are represented with binary arrays of zeros and ones, which allow an easy compounding with binary arrays of another *Boolean attribute* by bit-wise operations of AND, OR and NOT. Thus a bit-string representation of any derived *Boolean attribute* could be quickly prepared from involved basic *Boolean attributes*. Moreover bit-string representations of partial derived Boolean attributes are cached during walking-through the search-space, so a new derived *Boolean attribute* representation have not to be prepare from scratch.

Another technique of skips over the search-space is implemented that significantly reduces number of patterns that have to be constructed. The whole branches of the search-space are skipped when there is no chance of verified patterns could be present there based on logic of associational rules and actual data-mining task parameters.

4.4.2 Filtering of found patterns

There are usually many patterns found, so users could easily get overwhelmed and get lost without a chance to spot really interesting results. There were therefore implemented means to decrease number of patterns without losing any information.

Found true patterns are filtered according to theirs logical properties – only so called prime-patterns are included into results. Thus only the patterns that do not easily follow from (a more simple ones) already presented in results are included. For details see e.g. [Rauch, 2005]. This technique has also inspired the filtering of truly novel patterns in the *Synthesizer* module – see section 4.6 later.

4.5 Distributed solving of tasks using grid

For the whole automated data-mining process to be feasible, there must be a way to compute each step of the *Inner Loop* iteration very quickly. Although there are several optimisation techniques already incorporated (see the previous section), there are clear limits for shortening solution times on a single computer. One possible solution how to significantly increase the computing power and hence to decrease solution times is to use computer grid to divide solution of single task among many grid nodes.

4.5.1 Grid type chosen

There were two possible options regarding the type of grid used – the dedicated grid or the PC-Grid consisting of ordinary PCs linked together as clients of the grid server. The dedicated-grid main advantage is its huge processing power and constant availability of this power because it has nothing else to do than to wait for assigned task to be solved. Meanwhile, computers in the PC-Grid are obliged to serve their primary users first and only the remaining computing power is available for the grid. On the other hand the main advantage of the PC-Grid are low initial costs and its easy scalability – just another PCs are registered. Being an academic institution where money funds are often scarce we opt for the PC-Grid.

The Techila PC-Grid [Techila] was successfully installed on the University of Economics computer network and now we would like to increase number of participation grid nodes by registering more PCs from offices, computer labs and even dormitories.

4.5.2 Core data mining algorithm overhaul

The main problem that had to be addressed while incorporating grid features into the LISp-Miner system was how to divide data-mining task into sub-tasks that could be solved (in parallel) on particular grid nodes. The goal was to find a general solution that could be used in all the implemented GUHA procedures within the LISp-Miner system, although they mine for different patterns and theirs core data-mining algorithm is different. Different strategies of task partitioning could lead to different complexity of each atomic sub-task and therefore to very different total computing times on the grid.

4.5.3 Implementation

Two strategies were chosen and already implemented – for details see [Šimůnek & Tammisto, 2010]. The most important feature of this solution is that no changes to the original optimised core data-mining algorithms are needed and this solution is applicable for all the GUHA-procedures implemented in the LISp-Miner system so far.

All the necessary communication with the grid using provided API was implemented on the user side and new modules for solving sub-tasks on grid nodes were created (one module for each of the eight data-mining procedures). This new modules use the same program code as is used for local solving of tasks. There is no change from the point of view of user –

only a new dialog window appears to decide whether to solve the concerned task locally or by the grid. Communications links and the grid access-privileges were secured by certificates.
A real data analysis was undertaken using distributed grid verification and results were compared to solving-times of the same task on a single computer. Two procedures were selected for benchmark tests – the *4ft-Miner* procedure as the currently most used one and the *Acft-Miner* procedure (looking up the *4ft-action-rules*) where the grid potential is even higher due to a more complex pattern syntax. A significant improvement in solution times was observed right from the first tasks. The grid overhead (due to division of a task into sub-tasks, to uploading all the necessary data to the grid and then to downloading the results) is reasonably small and its relative importance decreases by growing complexity of tasks. There was observed a near linear dependency between number of grid nodes involved and reduction in task-solution times. For example a task running for more than 30 hours on a single PC was solved within 6 hours on grid consisting from just 5 grid nodes. And the same task was solved in just one hour using 24 grid nodes (albeit more powerful ones, in 1,2 factor approximately).
The undertaken experiments proved that the implemented grid feature brings significant improvements to solutions times and is easily up-scaled for even better times by simply registering more PCs as grid nodes.

4.6 Synthesizer and inducing new knowledge

After new results are mined they are handed to the *Synthesizer* module to be confronted with the existing knowledge already stored in the *Knowledgebase* and possibly to induce a new knowledge. Remember, please, that new results have been already pruned of logically dependent patterns and only so called prime-patterns are passed to this phase.

4.6.1 What to not report

Even the prime-patterns could describe many kinds of true but (from the point of view of domain experts) completely worthless facts. Examples of such "gems" are: "*There is at least a 99% probability that a person giving birth to a child will be a women*" or "*Body temperature of patients is in range from 34 to 40 °C in more than 90 % of cases*". There is many more such statements that are certainly true but will irritate domain experts and maybe they will even break they faith in results of analysis. Thus, it is very important to automatically filter out as many of such statements as possible.
There is no sense too in reporting the same patterns over and over if they were already presented to users in previous rounds of analysis. So every found pattern has to be checked against knowledge already present in the *Knowledgebase* and only really novel facts will be append there are reported in analytical reports.

4.6.2 Filtering of already-known knowledge

A technique similar to prime-rule testing is proposed for comparing newly found patterns with knowledge already in the *Knowledgebase*.
It is possible to translate any kind of *Mutual Dependency* knowledge stored in the *Knowledgebase* to one (or more) patterns looked-up by one of analytical procedures. For example, the dependency of *Education* $\uparrow\downarrow$ *BMI* (stating that a higher level of education leads generally to a lower level of the *Body-Mass-Index*) could be translated into *4ft-asociational rules* in form of:

$$\text{Education}(\alpha_{\text{right\_cut\_n}}) \Rightarrow_{p,B} \text{BMI}(\beta_{\text{left\_cut\_m}})$$

where

- $\alpha_{\text{right\_cut\_n}}$ is so-called *right cut* of categories of the attribute *Education* with the length of *n* (i.e. *n* of highest categories of the concerned ordinal attribute)
- $\beta_{\text{left\_cut\_m}}$ is so-called *left cut* of categories of the attribute BMI with the length of *n* (i.e. *n* of lowest categories of the concerned ordinal attribute)
- $\Rightarrow_{p,B}$ is the *4ft-quantifier* of *Found implication* based on the confidence value of *a/(a+b)*.

Similar translations are available for remaining types of *Mutual Dependency* using possibly another types of quantifiers or whole analytical procedures and their patterns (*KL-patterns*, *CF-patterns* and even *SD4ft-*, *SDKL-* and *SDCF-patterns*) – for more details see [Rauch, 2010].

This translation of *Mutual dependencies* needs to be done only once (either before the EverMiner analysis begins or after each change of the *Knowledgebase*). When a new pattern is mined and sent to the *Synthesizer* module it will be checked against subsets of this translated patterns (only those for the same analytical procedure). What we want to resolve is whether it (logically) follows from some (simpler) pattern already present in the *Knowledgebase*. So the same approach could be used as for the prime-rule testing described above. But this time the set of patterns it is checked against the one translated from the *Knowledgebase*.

If deduction rules prove that the newly found pattern logically follows from a pattern representing a *Mutual dependency* already present in the *Knowledgebase*, it could be either filtered-out or this *Mutual dependency* could be flagged that it is supported by the analysed data.

4.6.3 Inducing new knowledge

If a newly found pattern meets test of novelty it need to be added into the *Knowledgebase* in form of the new *Mutual dependency* knowledge. Again, there are translations-rules available for each analytical procedure (and its type of patterns) how to construct a new *Mutual dependency* based on the found pattern.

When a new *Mutual dependency* is created and inserted into the *Knowledgebase*, it is compared to already existing *Mutual dependencies* for the same pair of (meta-) attributes. If they are two different types of *Mutual dependencies* now in the *Knowledgebase*, it should be investigated further whether they are complementary or contradictory. Complementary dependencies could coexist e.g. in case of:

$$\text{Education} \uparrow\downarrow \text{BMI and Education} \downarrow\uparrow \text{BMI}$$

In this special case, a tighter Mutual dependency of $\mathcal{F}$(Education, BMI) stating that there is a strict function-like dependency between the level of education and value of the BMI could be used to formulate a new LAQ (and eventually to prove it).

On the other hand, the contradictory *Mutual dependencies*, e.g. in case of:

$$\text{Education} \uparrow\uparrow \text{BMI and Education} \uparrow\downarrow \text{BMI}$$

leads to the "*rejected in the analysed data*" flag to be set for the first mutual dependency and the found contradiction need to be highlighted in the analytical report.

4.6.4 Groups of related patterns

The same dependency in the analysed data could be expressed in more than one way by different types patterns of different analytical procedures. For example, where a single *KL-pattern* could be sufficient to describe a function-like dependency between two attributes, tens of *4ft-associational rules* could be necessary to express the same. Too many found rules make results hard to understand and complicate reaching right conclusions. It is necessary therefore to identify groups of patterns that describe a single dependency in the analysed data and possibly to use another type of the *LAQ Template* and therefore another analytical procedure to answer it. This feature is not understood well and has to be addressed in future research.

4.7 Complete history of analysis

Every decision taken during both *Loops* and even the intermediate results need to be logged so the whole history of the automated KDD process could be checked afterwards. No information of any kind is ever deleted. This will not only help during development of the EverMiner but also will allow an analytical audit of the whole reasoning behind delivered results and to prove the validity of newly induced knowledge. Types of stored information are:

- used mapping of attributes from the analysed data to the meta-attributes in the *Knowledgebase*;
- formulated *LAQs* and its parameters (used *Template*, groups of meta-attributes)
- created tasks; each task is associated to the *LAQs* it is supposed to answer;
- information about task-parameters changes during fine-tuning in the *Inner Loop*;
- found prime patterns of each task-run;
- answers to the *LAQs* derived from the found prime patterns – together with information whether they support the already known facts in the *Knowledgebase*;
- changes made to the *Mutual Dependency* type of knowledge.

The necessary infrastructure for storing this kind of information is already in place for man-controlled data-mining and could be used for the EverMiner too:

- Every created attribute has already an optional link to its master *meta-attribute*.
- Formulated *LAQs* are stored in database and their status could be monitored.
- Each data-mining task must belong to a task-group. So a task-group will be created for every *LAQ* and all data-mining tasks designed to answer it will be included in this group (remember that there could be more than one task necessary to answer a single *LAQ*).
- Already implemented feature of task-cloning will be utilized for keeping track of task-parameters evolution during the *Inner Loop* – a new clone of the task with current version of task-parameters will be created in each iteration before task-parameters are changed. The name of the newly cloned task will be the same and task will be inserted into the same task-group. Only the "iteration" index will be increased to provide information about sequence of steps in the *Inner Loop*.
- Found results are routinely stored within data-mining task data to be visible to users in man-controlled data-mining analysis. This feature allows keeping results from all iterations because each task-parameters version is stored in corresponding cloned-task and identified with its "iteration" number.

- Every newly synthesized knowledge is added in form of the *Mutual Dependency* into the *Knowledgebase*. Three important properties accompany it – whether it is created by some user (domain specialist) or by the *Synthesizer* module; the time-stamp – when it was created; and finally links to data-mining results the new knowledge is based upon. Incorporation of the time-stamp allows storing multiple instances of mutual dependence to a single pair of meta-attributes while preserving the whole evolution of who and when made any change. Thus, a mutual dependency relationship could be marked as "*proven*" when results from a data-mining task will prove them, or could be marked as "*rejected*" otherwise. And a complete "evolution graph" of the *Knowledgebase* could be constructed afterwards to provide users with deep explanation why some new knowledge was induced and based on what patterns in the analysed data.

5. Conclusion and further work

All the phases necessary to build an automated data-mining system were proposed. Some parts were already implemented and the remaining pieces have a sufficient theoretical background to be implemented in a near future.

Our goal is to proceed in partial steps and gradually build the functioning EverMiner system. Currently, we are working on the communication with the SEWEBAR project repositories to be able to gather relevant information into the EverMiner *Knowledgebase* regarding processed attributes. A first version of the *QuestionMaker* for formulating some simple kinds of *LAQs* based on the knowledge already stored in the *Knowledgebase* will be implemented then. It will allow to launch first analytical procedures tasks and to solve them using the already implemented grid feature.

After obtaining the first results we will be able then to deploy appropriate rules for the fine-tuning of the task parameters, based on the number and quality of found prime patterns. Another kinds of knowledge could be possibly stored into the Knowledge to help either during the *Data Preprocessing* phase, during formulation of *LAQs* or during pruning results of already-known facts.

6. Acknowledgements

This text was prepared with the support of "Institutional funds for support of a long-term development of science and research at the Faculty of Informatics and Statistics of The University of Economics, Prague".

7. References

Agrawal, R.; Iimielinski, T.; Swami, A. (1993). Mining associations between sets of items in massive databases. *Proceedings of the ACM-SGMOD 1993 Int. Conference on Management of Data*, pp. 207-216, 1993, Wasghington D.C.

Agrawal, R.; Manilla, H.; Srikant, R.; Toivonen, H.; Verkamo, A. (1996) *Fast Discovery of Association Rules*. In: Advances in Knowledge Discovery and Data Mining, Fayyad, U. M. et al., (Eds.), pp. 307–328, AAAI Press/The MIT Press

Balhar, T.; Kliegr, T.; Šťastný, D.; Vojíř, S. (2010). Elicitation of Background Knowledge for Data Mining. *Proceedings of Znalosti 2010*, s. 167–170, ISBN 978-80-245-1636-3, Jindřichův Hradec, February 2010, Oeconomica, Praha

CRISP-DM*: Cross Industry Standard Process for Data Mining* [online]. [cit. 18. 12. 2009], available from WWW: http://www.crisp-dm.org

Hájek, P. & Havránek, T. (1978). *Mechanising Hypothesis Formation – Mathematical Foundations for a General Theory*. Springer-Verlag, Berlin – Heidelberg – New York, 1978, 396 pp.

Hájek, P. & Havránek, T. (1982). GUHA80: An Application of Artificial Intelligence to Data Analysis. *Computers and Artificial Intelligence*, Vol. 1, 1982, pp. 107-134

Hájek, P. & Ivánek, J. (1982). Artificial Intelligence and Data Analysis, *Proceedings of COMPSTAT'82*, Caussinus H., Ettinger P., Tomassone R. (Eds.), pp. 54-60, 1982, Wien, Physica Verlag

Hájek, P.; Holeňa, M.; Rauch, J. (2010). The GUHA method and its meaning for data mining. *Journal of Computer and System Sciences*, Vol. 76, 2010, pp. 34-48, ISSN: 0022-0000

Kliegr, T.; Ralbovský, M.; Svátek, V.; Šimůnek, M.; Jirkovský, V.; Nemrava, J.; Zemánek, J. (2009). Semantic Analytical Reports: A Framework for Post-processing data Mining Results, *Proceedings of Foundations of Intelligent Systems*, pp. 88–98, ISBN 978-3-642-04124-2, ISSN 1867-8211, Praha, September 2009, Springer Verlag, Berlin

Lín, V.; Dolejší, P.; Rauch, J.; Šimůnek, M. The KL-Miner Procedure for Datamining, *Neural Network World*, Vol. 5, 2004, pp. 411–420, ISSN 1210-0552.

LISp-Miner – academic KDD system [online], [cit 2010-07-15], available from WWW http://lispminer.vse.cz

Ras, Z. & Wieczorkowska, A. (2000). Action-Rules: How to Increase Profit of a Company. *Proceedings of PKDD 2000*, Zighed, D.A., Komorowski, J., Zytkow, J.M. (Eds.), pp. 587–592, LNCS (LNAI) Vol. 1910, Springer, Heidelberg

Rauch J. (2005): Logic of Association Rules. *Applied Intelligence*, Vol. 22, 2005, pp. 9 – 28, ISSN 0924-669X

Rauch J. (2009): Considerations on Logical Calculi for Dealing with Knowledge in Data Mining, In: *Advances in Data Management*, Ras Z. W., Dardzinska A. (Eds.), pp. 177 – 202, Springer, 2009

Rauch, J. (2010). EverMiner – Consideration on a Knowledge Driven Permanent Data Mining Process, EverMiner – Consideration on a Knowledge Driven Permanent Data Mining Process, *International Journal of Data Mining, Modelling and Management*, ISSN: 1759-1171 (Online), 1759-1163 (Print), accepted for publication

Rauch, J. & Šimůnek, M. (2005a). GUHA Method and Granular Computing, *Proceedings of IEEE 2005*, HU, Xiaohua, LIU, Qing, SKOWRON, Andrzej, LIN, Tsau Young, YAGER, Ronald R., ZANG, Bo (Eds.), pp. 630–635, ISBN 0-7803-9017-2, Beijing, July 2005, IEEE, Piscataway

Rauch, J. & Šimůnek, M. (2005b). An Alternative Approach to Mining Association Rules. In: *Foundations of Data Mining and Knowledge Discovery*, LIN, Tsau Young et. al. (Eds.), pp. 211–231, ISBN 3-540-26257-1, ISSN 1860-949X, Springer, Berlin

Rauch, J. & Šimůnek, M. (2005c). New GUHA procedures in LISp-Miner system, *Proceedings of COST 274: Theory and Applications of Relational Structures as Knowledge Instruments*. pp. 73–85, Universidad de Málaga, April 2005, Málaga

Rauch, J. & Šimůnek, M. (2007). Semantic Web Presentation of Analytical Reports from Data Mining – Preliminary Considerations, *Proceedings of the WEB INTELLIGENCE*, pp. 3–7, ISBN 0-7695-3026-5, San Francisco, November 2007, IEEE Computer Society, Los Alamitos

Rauch, J. & Šimůnek, M. (2008). LAREDAM – Considerations on System of Local Analytical Reports from Data Mining, *Proceedings of Foundations of Intelligent Systems*, pp. 143–149, ISBN 978-3-540-68122-9, ISSN 0302-9743, Toronto, May 2008, Springer-Verlag, Berlin

Rauch, J. & Šimůnek, M. (2009a). Dealing with Background Knowledge in the SEWEBAR Project. In: *Knowledge Discovery Enhanced with Semantic and Social Information*, BERENDT, Bettina, MLADENIČ, Dunja, GEMMIS, Marco de, SEMERARO, Giovanni, SPILIOPOULOU, Myra, STUMME, Gerd, SVÁTEK, Vojtěch, ŽELEZNÝ, Filip (Eds.), pp. 89–106, Springer-Verlag, ISBN 978-3-642-01890-9. ISSN 1860-949X, Berlin. URL: http://www.springer.com/engineering/book/978-3-642-01890-9.

Rauch, J. & Šimůnek, M. (2009b). Action Rules and the GUHA Method: Preliminary Considerations and Results, *Proceedings of Foundations of Intelligent Systems*, pp. 76–87, ISBN 978-3-642-04124-2. ISSN 1867-8211, Praha, September, 2009, Springer Verlag, Berlin

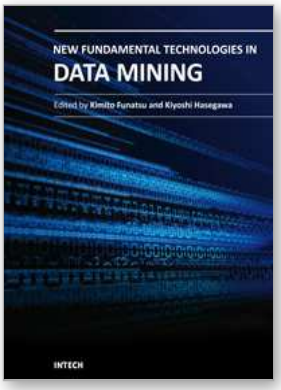
Rauch, J.; Šimůnek, M.; Lín, V. (2005). Mining for Patterns Based on Contingency Tables by KL-Miner – First Experience. In: *Foundations and Novel Approaches in Data Mining*, LIN, Tsau Young, OHSUGA, Setsuo, LIAU, C. J., HU, Xiaohua (Eds.), pp. 155–167, ISBN 3-540-28315-3. ISSN 1860-949X, Springer-Verlag, Berlin

SEWEBAR project [online], [cit 2010-17-14], available from WWW http://sewebar.vse.cz

Šimůnek, M. (2003). Academic KDD Project LISp-Miner. *Proceedings of Advances in Soft Computing – Intelligent Systems Desing and Applications*, ABRAHAM, A., FRANKE, K., KOPPEN, K. (Eds.), pp. 263–272, ISBN 3-540-40426-0, Tulsa, 2003, Springer-Verlag, Heidelberg

Šimůnek, M. & Tammisto, T. (2010). Distributed Data-Mining in the LISp-Miner System Using Techila Grid. *Proceedings of Networked Digital Technologies*, ZAVORAL, Filip, YAGHOB, Jakub, PICHAPPAN, Pit, El-QAWASMEH, Eyas (Eds.), pp. 15–21, ISSN 1865-0929, ISBN 978-3-642-14291-8, Praha, July 2010, Springer-Verlag, Berlin

Techila PC-Grid [online], [cit: 2010-07-10], see http://www.techila.fi



New Fundamental Technologies in Data Mining

Edited by Prof. Kimito Funatsu

ISBN 978-953-307-547-1
Hard cover, 584 pages
Publisher InTech
Published online 21, January, 2011
Published in print edition January, 2011

The progress of data mining technology and large public popularity establish a need for a comprehensive text on the subject. The series of books entitled by "Data Mining" address the need by presenting in-depth description of novel mining algorithms and many useful applications. In addition to understanding each section deeply, the two books present useful hints and strategies to solving problems in the following chapters. The contributing authors have highlighted many future research directions that will foster multi-disciplinary collaborations and hence will lead to significant development in the field of data mining.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

M. Šimůnek and J. Rauch (2011). EverMiner - towards Fully Automated KDD Process, New Fundamental Technologies in Data Mining, Prof. Kimito Funatsu (Ed.), ISBN: 978-953-307-547-1, InTech, Available from: http://www.intechopen.com/books/new-fundamental-technologies-in-data-mining/everminer-towards-fully-automated-kdd-process

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821