Paper

Modeling preparation for data mining processes

Timm Euler

Abstract— Today many different software tools for decision support exist; the same is true for data mining which can be seen as a particularly challenging sub-area of decision support. Choosing the most suitable tool for a particular industrial data mining application is becoming difficult, especially for industrial decision makers whose expertise is in a different field. This paper provides a conceptual analysis of crucial features of current data mining software tools, by establishing an abstract view on typical processes in data mining. Thus a common terminology is given which simplifies the comparison of tools. Based on this analysis, objective decisions for the application of decision supporting software tools in industrial practice can be made.

Keywords— data mining, data preparation, KDD process.

1. Introduction

Knowledge discovery in databases (KDD) and data mining are, in practice, complex processes whose development requires advanced skills and precise data understanding. Increasingly, however, software systems that support many aspects of data mining on a high level are becoming available, which makes the development of industrial mining applications easier even for less experienced users. Examples for such systems are SPSS Clementine, SAS Enterprise Miner or MiningMart. For industrial decision makers, the choice of the most suitable decision support software is becoming an important challenge, as many software tools are available, but each has its own strengths and weaknesses. This paper addresses the question of how the different technologies can be compared with respect to their work-saving potentials. The main thesis is that there are critical aspects of a mining process which must be explicitly supported on a conceptual level by a high-quality data mining software. The paper contributes a detailed, conceptual analysis of these critical aspects. This allows to establish a common terminology for a broad range of functionalities, and thus to easily compare different software solutions based on detailed, objective and quantitative criteria.

The general focus of the paper is data processing during data preparation and mining. For the preparation of data, the conceptual analysis presented in Section 3 yields a list of essential operators that must be available in order to be able to compute arbitrary data representations. Arbitrary data representations may be needed for a successful mining phase. During mining itself there are also central data processing tasks that must be supported, such as cross validation and parameter tuning, which is explained in Subsection 3.5. Based on this conceptual view on critical aspects of the process, criteria for existing software solutions in data mining and knowledge discovery are derived in Section 4. The criteria reveal the strengths and weaknesses of such software solutions, allowing clear and objective decisions for the application of data mining software tools in industrial practice.

2. Related work

Conceptual models of knowledge discovery processes (of which data mining is the crucial part) have mostly been developed in the context of data mining supporting tools. In particular, [4, 20] attempt to assist users during the development of discovery processes by automatically exploring various options for the process. To this end, the basic steps in a KDD process are realized by agents in [20]; meta-agents (planners) organise them to a valid process using their input and output specifications. The authors provide an ontology of KDD agents that distinguishes between three phases of the process, namely preprocessing, knowledge elicitation (modeling) and knowledge refinement. The particular choice of agents is not explicitly justified in the published articles; compare the minimal and complete list of operators in Subsection 3.3. Further, the present work includes a conceptual view on the data which is missing in [20]. The same is true for [4], where a system to systematically enumerate and rank possible KDD processes is presented, given some input data and a mining goal. These authors have developed a metamodel for KDD processes. In this respect, there system is somewhat similar to MiningMart [16], which is the system that inspired much of the present work.

A well-known standard to model the KDD process is CRISP-DM [7]. While it gives an overview of different, interdependent phases in a KDD process and defines some terminology, it is not detailed enough to model concrete instances of data preparation and modeling operations, and does not include a data model. An early sketch of a formal model of the KDD process was presented in [19]. The new predictive modeling markup language (PMML) Version 3.0[1], a standard to describe machine-learned models in extended markup language (XML) [18], includes facilities to model the data set and data transformations executed on it before modeling. However, it is not process-oriented,

[1] See http://www.dmg.org/pmml-v3-0.html

thus it does not allow to model a data flow through a complex KDD process, and the data model is restricted to one table. Other standards around data mining are Java data mining (JDM) [13], which includes web service definitions, and structured query language/multimedia extension (SQL/MM) data mining. Though extensible, they currently provide interfaces to modeling algorithms rather than to complete KDD processes.

Recently, some new research attempts to employ grid infrastructures for knowledge discovery; a good overview is given in [5]. To enable the execution of KDD processes on a grid, these processes have to be modeled independently from the machines that execute them, and heterogeneous data schemas and sources have to be modeled. In [2], a discovery process markup language (DPML) is used, based on XML, to model the complete KDD process. This language is used to formalize a conceptual view on the data mining process. Unfortunately, from the available publications it is not clear how comprehensive and detailed DPML is.

Criteria for the comparison of KDD and data mining tools have been listed in several papers [1, 8, 10, 11], but have not been linked with the conceptual works above. The criteria are therefore not consistent across publications and their selection is not justified. The present paper attempts to support the choice of criteria by a conceptual analysis of the essential data processing tasks in data mining for the first time.

3. Data preparation

This section introduces the conceptual notions that are needed to describe a data mining process. Subsection 3.1 introduces two description levels which are used to describe the data (Subsection 3.2) and the data processing (Subsections 3.3 and 3.5).

3.1. Two levels of KDD descriptions

It is generally possible to describe both the data and the preparation tasks on two different levels: a more technical one and a more KDD-related one. The technical level describes the data and any operations on the data independently of any application purpose. The higher level deals with KDD concepts: the role that the data plays, and the purpose of applying a preparation method, are seen in the context of the knowledge discovery application. This level will therefore be called *conceptual*. The differentiation of the two levels will be detailed below. One may relate the different levels to different types of users of data collections: while for example database administrators are concerned with the technical level, KDD experts and statisticians (data analysts) tend to think and work on the conceptual level, as they cannot take the application out of their focus.

One of the purposes of this work is to argue that the two levels should be explicitly supported by KDD software. This has the following advantages:

- If the higher level is made explicit, the lower one can be hidden. A software that hides the technical level can present the entire KDD process to a user in terms of familiar concepts. This eases the development of and daily work on KDD applications.
- By making the conceptual level explicit, it is automatically documented and can be stored and retrieved for later reference [16].
- Independence of the conceptual level allows to reuse parts or all of a conceptual process model on new data by simply changing the mapping to the technical level. Though this may require conceptual adaptations, it saves much effort compared to a development from scratch.
- The use of the conceptual level allows the comparison of different software tools by abstracting from technical details. Criteria for comparison can be formulated on the conceptual level, which makes their communication and application much easier (see Section 4).

3.2. Data description

Throughout the paper, the data is assumed to be in attribute-value format. On the technical level, it is common to think of *tables* which are organized in columns and rows. Conceptually, data is seen as representing objects from the real world; the objects are described in terms of their *attributes*; and each attribute has a *domain* whose *values* it can take. There can be different *sets* of data, with different attribute sets; it is common to refer to the different sets as tables even on the conceptual level, though the term *concept* will be used below. Thus there is a direct and simple mapping from attributes to columns and representation of objects to rows. Whether the columns and rows are gained from a flat file or a database system is unimportant on the conceptual level.

While attributes and concepts are used to describe the data *schema* – the organization of the data – on the conceptual level, a description of the data *contents* is also very useful on this level, since the data processing operations in a KDD process depend on both. Schema- and content-related information are usually referred to as *metadata*. During processing, both the data schema and the data contents, the data itself, change. To have the data characteristics listed below available on the conceptual level requires a data scan, which typically consumes a substantial amount of time because the data sets are large. Therefore, this analysis should be performed as rarely as possible, preferably only once, on the input data (even then, it may have to be performed on a sample of the data). Based on the characteristics of the input data, many characteristics of later, intermediate

data sets in the process can be inferred from the types of operations that were applied, rendering new data scans superfluous.

The useful metadata for modeling a KDD process includes: the number of rows in every table/concept; the minimum and maximum values of each attribute with ordered values; the list of values each discrete attribute takes; and the number of occurrences of each value of each discrete attribute.

One important task that can be solved based on this metadata is the estimation of the storage size needed for storing the data set. This is important during the declarative set-up of the KDD process, as it allows to decide *before* executing the process whether and how to store intermediate data sets (due to limited main memory), because the necessary metadata can be inferred in many cases (though not always). An intelligent administration of intermediate data processing results is important for a smooth execution of the process.

The administration of the above metadata allows not only size estimation, but also an easier declarative development of the KDD process model, as many operations depend on the values that certain attributes take. For example, *Value mapping* is an operator used to change these values, and a specification of an instance of this operator is easy if the available values can be chosen in a graphical interface, rather than be looked up elsewhere and typed in by hand.

Another useful kind of metadata is given by data *types*. On the technical level, the common data types are numbers (integer or real), strings, and calendar dates/clock times. Conceptually, however, one would distinguish types according to the way they represent real-world objects. In this work, four conceptual data types are proposed as essential because their distinction is needed during the development of a KDD process using the processing operators. These types are *date/time*, *key*, *discrete* (further divided into *binary* and *set*), and *continuous*.

In principle, every conceptual domain type can be realized by any technical data type. For example, keys can be realized by strings or by numbers; dates can be represented by strings; and so on. To hide the technical level, a flexible mapping is needed. When new data is introduced, the mapping from the technical to the conceptual level can be done automatically (by inspection of occurring values), but must also be manually manipulable to allow uncanonical mappings, like strings representing dates. The need for flexibility arises from the unpredictable ways in which data preparation operations may be combined. For example, one certain operator produces a binary output consisting of the numbers 0 and 1, where from the view of this operator the output is discrete (no ordering implied). Yet the next operator in a given application chain may compute the mean of that output, interpreting the 0s and 1s as real numbers, which is a neat way of computing the ratio of 1s. Even when such interpretation changes occur, it is still possible to hide the technical level by adjusting it automatically, determined by the kinds of manipulations that an operator defines. This is one example of how a conceptual analysis leads to objective criteria for software.

Further information about attributes (beyond conceptual data type and data characteristics) is given by the *role* it plays in the KDD process. Some attributes are used as input for learning; one or more may contain the target (the label) for learning; still others relate several tables to each other. Thus, four roles are distinguished on the conceptual level (without a correspondence on the technical level): *predictor*, *label*, *key* and *no role*.

Changing the perspective now from (domains of) attributes to whole tables, their contents, and how they relate to each other, it is easy to see that the two levels of description can be applied in a similar fashion. Data represents objects from the real world and describes them along several dimensions. Some objects share similarities, which allows to subsume them in a *class*; the science of what classes exist and how they should be described is called *ontology*. Leaving philosophical approaches aside, the word *ontology* is used in computer science as a countable noun, where an ontology is the description of a shared conceptualization of an application domain [12]. Obviously, a conceptual description of data sets could make use of ontologies. If an ontology exists for the application domain from which the data is collected, it would be very helpful to describe a KDD application on that data in terms of that ontology [6, 9]. However, realizing this idea is fraught with the difficulty that not all ontology formalisms are suitable for supporting KDD-oriented data processing. Data for KDD comes in tables, and the tables are the objects of the extensive modifications which are usual during data preparation. A useful conceptualization, from an operational point of view, should therefore introduce a concept for each table, even though some concepts from the application domain may, in a given data set, be described using several tables, or only a part of one table. The latter problem can however be remedied by the availability of data transformations in KDD to bring the tables into a suitable shape [9].

A mining process consists of a sequence of transformation operations, as explained in Section 3, and each operation introduces a new data set, or in the conceptual view, a new concept. Thus a large number of intermediate concepts is created in a large process, and the intermediate concepts are related by the data flow or process view. However, they are also related in a different way, namely by the nature of their creation: some processing operations create *subsets* of the input data, while others create *specializations*. It will be seen in Subsection 3.3 that several essential processing operations produce such relations. Further, the intermediate concepts may be related by *foreign key links*. The web of these relations allows an alternative view on the data mining process which can help the user to keep an overview of it.

3.3. Data preparation operators

Usually, data preparation is seen as the execution of basic steps, each of which applies some predefined data transformation to the output of the previous step(s), resulting

in *dependency graphs* of data preparation. The predefined data transformations are defined through *operators*, which are specified by their input, their transformation task and their output. It is important to note that input and output can be specified on the higher, conceptual description level. In [14], a list of atomic operations for data preparation was given for the first time. One main contribution of the present work is the classification of these and other operators into *primitive* and *convenience* operators. The former correspond to the technical description level. They provide basic operations without which no complex data preparation can be performed. Their computational power is examined in Subsection 3.4.

The operator *Model learning* has a special status; it is not a primitive operator for data preparation, and does not produce output in terms of the data ontology, but is indispensable for a complete KDD process.

The convenience operators describe data transformations in conceptual, KDD task-related terms; they are mere combinations or special cases of the primitives. As an example, the convenience operator *Dichotomization* takes a discrete attribute and outputs several attributes, one for each value occurring in the given attribute, where the output attributes contain a Boolean flag indicating whether the value they correspond to occurs in that row in the input. This convenience operator can be realized by a repeated application of the primitive operator *Attribute derivation*. However, for a KDD expert user, using the convenience operators where possible is more intuitive than using many primitive operators, and provides an aggregated, high-level view of the preparation graph. Thus again the claim that KDD can be extensively supported on the conceptual level is justified.

In the following, brief descriptions of all primitive and some convenience operators are given.

- *Attribute derivation (primitive)* – a very general operator to create a new attribute, usually based on values of existing attributes. To allow this, extensive date, string and numeric arithmetics must be offered by this operator. In fact, to make the list of primitive operators complete in the above sense, arbitrary computations must be allowed to derive a new attribute. This requires a computationally complete formalism such as a programming language. The input for this operator is any concept; the output is a concept that is a specialization of the input concept.

- *Attribute selection (primitive)* – this operator removes attributes from the input concept. The selection of attributes to be removed is either done by the user or, for advanced applications, automatically, using redundancy criteria or the performance of a modeling algorithm on different attribute sets. The input is any concept with at least two attributes. The output is a concept of which the input concept is a specialization.

- *Join (primitive)* – this operator joins two or more input concepts according to the values of a key attribute specified for each concept. All attributes from the input concepts occur in the output concept without duplicating keys. The input are two or more concepts, each of which has a key that relates it to one of the other input concepts. The output is a concept that is a specialization of all input concepts.

- *Model learning (special)* – this operator is a general place holder for model learning algorithms. In predictive settings, the model gives a prediction function that can be applied to other concepts in the *Attribute derivation* operator. In descriptive settings, only the model itself is produced.

- *Row selection (convenience)* – this operator copies certain rows from the input concept to the output concept, according to some criteria. The input is any concept. The output is a concept that is a subconcept of the input concept.

- *Union (convenience)* – this operator unifies two or more concepts that have the same attributes. The extension of the output concept is the union of the extensions of the input concepts. The input are two or more concepts, each with the same set of attributes. The output is a concept with again the same attributes, of which every input concept is a subconcept.

- *Aggregation (convenience)* – this operator aggregates rows of the input concept according to the values of given *Group by*-attributes. Aggregation attributes are chosen in the input concept; in the output concept, values that are aggregated over an aggregation attribute appear for each combination of values of the *Group by*-attributes. The input is any concept with at least two attributes. The output is a new concept not related to the input concept.

- *Discretization (convenience)* – this operator discretises a continuous attribute. That is, the range of values of the continuous attribute is divided into intervals, and a discrete value is given to every row according to the interval into which the continuous value falls.

- *Value mapping (convenience)* – this operator maps values of a discrete attribute to new values. In this way, different values can be mapped to a single value, thus be grouped together, if they should not be distinguished later in the process.

- *Dichotomization (convenience)* – this operator takes a discrete attribute and produces one new attribute for each of its values. Each new attribute indicates the presence or absence of the value associated with it by a binary flag.

- *Missing value replacement (convenience)* – this operator fills gaps left in an input attribute (the target attribute) by missing or empty values.

3.4. Computational power of the primitive operators

This section considers the range of convenience operators that are definable by the primitives from Subsection 3.3. Though *Attribute derivation* is assumed to use computationally complete mechanisms, it does not add tuples to its input, and it derives only one attribute. Thus it is a natural question to ask whether the three primitives can produce any output concept that is computable from given input concepts. This is a notion of computational completeness based on the given data model. The answer is that the three primitives alone cannot provide this computational completeness without iteration or recursion constructs and counting devices. However, even using the three primitives in schematic algorithms (without looping, recursion or counting) allows to realize a number of important convenience operators as demonstrated above. Below some more precise observations on the expressiveness of the primitives are given, without proof because they are easy to verify.

First, it is easy to see that the six basic operators of the relational algebra, which is used in relational databases, can be reduced to the three primitives above. For projection (*Attribute selection*) and natural join this is trivial; for the other relational operators the reduction to primitives is not difficult either, but is omitted here.

Second, the primitives are in fact more expressive than the relational algebra: it is well-known that the transitive closure of an arbitrary directed graph cannot be computed using the relational algebra [3], but this is possible using the three primitives defined here. Indeed, any function on a given set of input concepts that produces a fixed number of output concepts with known arities, and where the sizes of these output concepts are bounded in terms of the input sizes, can be computed by the three primitives, essentially by computing the cross-product of all values occurring in inputs as many times as needed to create enough entities, and using the power of *Attribute derivation* to compute the function results[2]. The number of edges in the transitive closure of a graph is of course bounded by $O(n^2)$ for n nodes and the arity of the result concept is 2. Thus the transitive closure of any graph given as a 2-ary concept can be computed by forming the cross-product of its nodes (joining the input concept with itself) and then using *Attribute derivation* to mark the relevant edges. The latter are then selected using *Row selection*.

Third, it is easy to see that given an arbitrary function on sets of input concepts, there may be no fixed schema of applying the three primitives to compute it. For example, the function that creates n copies of an input concept, where n is the number of entities in the instance of the input concept, is dependent on the input size and thus cannot be expressed on arbitrary inputs using a constant number of primitives.

From these observations it is clear that the three primitives do not provide full computational power, but they do allow to construct powerful convenience operators for practical purposes, since for many computations which are needed in practical applications, bounds on the output size can easily be given and thus no iteration or recursion constructs are needed in addition to the three primitives.

Obviously the computational power of the primitive operators stems from *Attribute derivation*, but it is a useful insight that these three conceptually simple operators, applied to instances of the intuitive modified entity-relationship (ER) data model, provide very powerful operations. So it suffices to think of data processing on instances of the modified ER metamodel in terms of attribute addition, attribute removal, and joining by keys. For many KDD applications, the full computational power of *Attribute derivation* is not even needed. It will often suffice to employ some of the simpler functions it offers. But KDD is a complex field, and users will need flexible devices to cope with very different situations in different KDD projects. Therefore a KDD workbench should offer the full computational power of *Attribute derivation* as a fall back mechanism for unforeseen situations.

3.5. Data mining

During modeling, conceptual support is mainly needed for training, testing (evaluation of models), and parameter tuning, as well as the visualization of models. Conceptual support here means again to present these tasks in suitable terms; for example, standard operations should be offered to split a data set into training set and test set, to learn, evaluate and apply a model, to automatically find optimal parameter settings, and so on. Since modeling is in itself a complex process, in fact this often leads to a separate graph of processing tasks. Following [15], trees of nestable operators are a suitable, conceptual representation for these tasks. The leaves of the trees represent operations such as the learning or application of a model, while the inner nodes correspond to more abstract, control-oriented tasks such as cross validation or meta learning. This representation provides great flexibility for the design of complex mining experiments, which are independent of the data preparation in that they take a single, fixed data table as input.

4. Criteria for data mining tools

How can the conceptual analysis from Section 3 be applied in practice? The analysis focused on data processing during preparation and mining. According to [17] and a 2003 KDnuggets poll[3], most of the efforts spent on a KDD project are consumed by data preparation. Therefore the analysis above directly concerns work-intensive areas of KDD. It provides the details for a *declarative* development of KDD processes on the conceptual level, given a system that realizes a translation to the technical level.

[2]This argument presumes that at least one input instance has more than one entity.

[3]See http://www.kdnuggets.com/polls/2003/data_preparation.htm

Various data mining systems, like Clementine, IBM Intelligent Miner, SAS Enterprise Miner or MiningMart already realize certain parts of the notions from Section 3, but no "ideal" system exists (yet) that includes all of these notions. The concepts can be directly translated to functional criteria for data mining systems that include data preparation facilities. As a simple example, *all* primitive operators from Subsection 3.3 *must* be available in such systems, otherwise the data preparation facilities are incomplete (the operator *Row derivation* is an exception). The more convenience operators are available, the better. Attribute *roles* must be supported as well as the three types of relations between intermediate concepts (see Subsection 3.2); for this, concepts (representing tables) must be explicitly represented; and so on. These criteria can be objectively and simply checked in any data mining tool. They can also be easily *quantified*, as explained in the following.

Every notion from Section 3 can be broken down into a number of *Boolean* criteria. For example, each operator from Subsection 3.3 corresponds to one Boolean flag indicating its presence or absence in a given tool. The same is true for the four attribute roles. Other ideas from Section 3 can be set up as Boolean lists as well: for example, the explicit support for conceptual data types can be present or absent; the mapping from conceptual data types to technical types may be adjusted automatically in a given tool or not; and so on. This results in a set of detailed, Boolean criteria for data mining tools.

However, while a long list of Boolean criteria is very detailed, it does not serve well to gain a quick overview of the strengths and weaknesses of a tool. To make the evaluation clearer, related criteria can be *grouped*. Assuming a group of $m > 0$ criteria, any given data mining tool will fulfill $0 \leq n \leq m$ of them. This leads to the *n-of-m* metric for evaluating KDD tools, or indeed any type of systems given functional criteria. The size of the groups is variable; each group can have an own value of m. Further, the grouping itself can be adjusted to different purposes. To gain a quick, broad overview, larger groups (larger values of m) can be used, while for detailed inspections smaller groups are recommended. So the *n-of-m* metric is adaptable to different evaluation purposes and different audiences for the presentation of evaluation results. Based on a single, detailed list of Boolean criteria, humanly comprehensible quantitative scores can be formed to compare and evaluate KDD tools.

5. Conclusions

This paper has addressed the important, time-consuming data processing phases of the KDD process, namely data preparation and data mining. It was shown how these tasks and the methods to solve them can be described on two levels, a higher, conceptual one which is independent of the realization of the KDD process, and a lower one that realizes the process. Critical aspects for declarative models of KDD processes were identified, in the area of data descriptions (data models), preparation operators (with a minimal and complete list of essential operators), and data processes around the actual mining algorithm (such as cross validation or parameter tuning). Based on these critical aspects, a methodology to set up objective and quantifiable criteria for the comparison and evaluation of KDD tools was presented. The methodology is adaptable to different evaluation purposes and audiences for the presentation of evaluation results.

References

[1] D. W. Abbott, I. P. Matkovsky, and J. F. Elder IV, "An evaluation of high-end data mining tools for fraud detection", in *IEEE Int. Conf. Syst., Man. Cybern.*, San Diego, USA, 1998.

[2] S. AlSairafi, F.-S. Emmanouil, M. Ghanem, N. Giannadakis, Y. Guo, D. Kalaitzopoulos, M. Osmond, A. Rowe, J. Syed, and P. Wendel, "The design of discovery net: towards open grid services for knowledge discovery", *High-Perform. Comput. Appl.*, vol. 17, no. 3, pp. 297–315, 2003.

[3] A. V. Aho and J. D. Ullman, "Universality of data retrieval languages", in *Proc. 6th ACM Symp. Princ. Programm. Languag.*, San Antonio, USA, 1979, pp. 110–117.

[4] A. Bernstein, S. Hill, and F. Provost, "Toward intelligent assistance for a data mining process: an ontology-based approach for cost-sensitive classification", *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 4, pp. 503–518, 2005.

[5] M. Cannataro, A. Congiusta, C. Mastroianni, A. Pugliese, D. Talia, and P. Trunfio, "Grid-based data mining and knowledge discovery", in *Intelligent Technologies for Information Analysis*, N. Zhong and J. Liu, Eds. Berlin: Springer, 2004.

[6] H. Cespivova, J. Rauch, V. Svatek, M. Kejkula, and M. Tomeckova, "Roles of medical ontology in association mining CRISP-DM cycle", in *Worksh. Knowl. Discov. Ontol. ECML/PKDD*, Pisa, Italy, 2004.

[7] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth, "CRISP–DM 1.0.", Tech. Rep., The CRISP–DM Consortium, Aug. 2000.

[8] K. W. Collier, D. Sautter, C. Marjaniemi, and B. Carey, "A methodology for evaluating and selecting data mining software", in *Proc. 32nd Hawaii Int. Conf. Syst. Sci. HICSS-32*, Hawaii, USA, 1999.

[9] T. Euler and M. Scholz, "Using ontologies in a KDD workbench", in *Worksh. Knowl. Discov. Ontol. ECML/PKDD*, Pisa, Italy, 2004.

[10] Ch. G. Giraud-Carrier and O. Povel, "Characterising data mining software", *Intell. Data Anal.*, vol. 7, no. 3, pp. 181–192, 2003.

[11] M. Goebel and L. Gruenwald, "A survey of data mining and knowledge discovery software tools", *ACM SIGKDD Explor.*, vol. 1, no. 1, pp. 20–33, 1999.

[12] T. R. Gruber, "Towards principles for the design of ontologies used for knowledge sharing", in *Formal Ontology in Conceptual Analysis and Knowledge Representation*, N. Guarino and R. Poli, Eds. Deventer: Kluwer, 1993.

[13] M. F. Hornick, H. Yoon, and S. Venkayala, "Java data mining (JSR-73): status and overview", in *Proc. Worksh. Data Min. Stand., Serv. Platf. 10th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (KDD)*, Seattle, USA, 2004, pp. 23–29.

[14] J.-U. Kietz, A. Vaduva, and R. Zücker, "Mining mart: combining case-based-reasoning and multi-strategy learning into a framework to reuse KDD-application", in *Proc. Fifth Int. Worksh. Multistrat. Learn. (MSL2000)*, Guimares, Portugal, 2000.

[15] I. Mierswa, R. Klinkberg, S. Fischer, and O. Ritthoff, "A flexible platform for knowledge discovery experiments: YALE – yet another learning environment", in *LLWA 03 Tagung. GI-Worksh. Woche Lern. Leh. Wiss. Adapt.*, Karlsruhe, Germany, 2003.

[16] K. Morik and M. Scholz, "The miningmart approach to knowledge discovery in databases", in *Intelligent Technologies for Information Analysis*, N. Zhong and J. Liu, Eds. Berlin: Springer, 2004.

[17] D. Pyle, *Data Preparation for Data Mining*. San Francisco: Morgan Kaufmann, 1999.

[18] S. Raspl, "PMML Version 3.0 – overview and status", in *Proc. Worksh. Data Min. Stand., Serv. Platf. 10th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (KDD)*, Seattle, USA, 2004, pp. 18–22.

[19] G. J. Williams and Z. Huang, "Modelling the KDD process", Tech. Rep. TR-DM-96013, Commonwealth Scientific and Industrial Research Organisation (CSIRO), DIT Data Mining, 1996.

[20] N. Zhong, C. Liu, and S. Ohsuga, "Dynamically organizing KDD processes", *Int. J. Patt. Recogn. Artif. Intell.*, vol. 15, no. 3, pp. 451–473, 2001.

Timm Euler received his dipoma in computer science from the University of Dortmund in Germany in 2001, after studies in Dortmund and Edinburgh, UK. Since then he has been a Research Assistant at the Artificial Intelligence Unit at the University of Dortmund. His research interests include natural language processing, data mining and modeling the KDD process.
e-mail: timm.euler@cs.uni-dortmund.de
Computer Science VIII
University of Dortmund
D-44221 Dortmund, Germany