Technical Report: Criteria for KDD Tool Evaluation

Timm Euler

Dortmund, December 2005

Contents

List of Figures

List of Tables

1. Evaluating Tools for KDD

This technical report lists a number detailed criteria which serve to evaluate software packages that support KDD. The criteria are based on a conceptual analysis of KDD, reported elsewhere, and detailed comparisons of available tools. The concrete criteria are given in setion 1.3. But first some related work is reviewed (section 1.1) and the methodology used is discussed in section 1.2. Section 1.5 describes a number of software packages which have been evaluated under the criteria from section 1.3; the results are presented in section 1.6.

1.1. Related work

1.1.1. General software evaluation

There are many aspects of software which can be evaluated. A useful distinction is that between the development of a software and its actual use as a *product*. The main evaluations concerning the development of software assess the quality and correctness of the source code; this is usually called *testing*. Testing is a complex issue, but this work does not involve testing a software. A good overview of software testing methods is given in (Riedemann, 1997). A higher-level type of evaluation assesses the development *process* in an institution, to see whether it follows certain standards that make the process controllable and repeatable. The software capability and maturity model (CAMM) is a major evaluation framework for development processes (Paulk et al., 1995).

The present work is concerned with software product evaluation, which addresses the central notion of software *quality*, and is defined as the assessment of software quality characteristics according to specified procedures (Punter et al., 1997). The characteristics of software quality are defined in an international standard, ISO/IEC 9126, entitled "Information Technology—Software Product Evaluation—Quality Characteristics and Guidelines for their Use", developed in 1991 and slightly modified several times afterwards. It defines six main characteristics of software quality, each with several subcharacteristics, as listed in figure 1.1. These characteristics can be the subject of an evaluation of a software product. The standard is a result of a decade of research that is mainly based on Boehm et al. (1978) and Cavano and McCall (1978). While the ISO standard 9126 aims at comprehensiveness, Kusters et al. (1997) and others have pointed out that different users of a product may have rather different quality requirements, and that it may be difficult for an organisation to determine the level and type of quality required in a specific situation.

Most of the ISO 9126 characteristics refer to external quality attributes, that is, such characteristics as can be examined when the software's source code is not available. However, at least the maintainability characteristic concerns internal aspects which are related to the code. This work considers only external characteristics; this view of software

is often subsumed under the notion *COTS* (commercial off-the-shelf) software (Maiden et al., 1997; Colombo & Guerra, 2002).

Importantly, the evaluation itself should also follow a standard procedure in order to be as objective as possible, and in particular to be reproducible. To this end another standard was published in 1999, the ISO 14598 standard, entitled "Information Technology—Software Product Evaluation". It introduces four phases that make up the evaluation process:

1. Establish evaluation requirements: The purpose of the evaluation, and the types of products to be evaluated, must be identified in this phase. Most importantly, a *quality model* is set up, which lists the characteristics that are agreed to bear an influence on the quality. The ISO 9126 quality characteristics provide a useful guide, or a checklist, for the identification of quality-related issues in a particular evaluation, but the ISO 14598 standard also allows other categorisations of quality that are more appropriate under the given circumstances. ISO 14598 explicitly states that there are no established methods for producing software quality specifications.

2. Specification of the evaluation: Since the ISO 9126 characteristics are not directly quantifiable, metrics that are correlated with them have to be established. The term "metric" is used in ISO 14598 not in the usual mathematical sense, but refers to a quantitative scale and a method which can be used for measurement. The word "measure" is used to refer to the result of a measurement (the term "score" is also used in this work). According to ISO 14598, every quantifiable feature of software that correlates with a characteristic from the quality model can be used as a metric. For every metric, a written procedure is needed that prescribes the assignment of measured values to it, to achieve objectivity.

3. Design of the evaluation process: An evaluation plan is produced that specifies the required resources, e.g. people, techniques or costs, and assigns them to the activities to be performed in the last phase.

4. Execution of the evaluation: Measurements are taken and scores computed as fixed in the evaluation plan.

In (Punter et al., 2004) a critical review and some refinements of this process can be found. In particular, the importance of establishing and prioritising the goals of an evaluation, and of involving all stakeholders of the evaluation in this, are stressed. Since the present work involves only one evaluator and has a clear, simple objective (see section 1.2), these refinements are not used here. Instead, section 1.2 describes the instantiation of the above process in the present work. Other ideas from the literature below are also used.

A new standard, ISO 25000, entitled "SQuaRE—Software Product Quality Requirements and Evaluation" is currently being developed to combine ISO 9126 and ISO 14598 (Suryn et al., 2003).

Regarding evaluation techniques, Punter (1997) argues for the use of weighted checklists, where the presence or absence of a number of agreed features is indicated and integrated into an overall score. Checklists are easy to customise and are a transparent, reproducible method of evaluation. A problem is the choice of items on the list, that is,

- Functionality—the capability of the software to provide functions which meet stated and implied needs when the software is used under specified conditions
  - Suitability—the capability of the software to provide an appropriate set of functions for specified tasks and user objectives
  - Accuracy—the capability of the software to provide right or agreed results or effects
  - Interoperability—the capability of the software to interact with one or more specified systems
  - Security—the capability of the software to prevent unintended access and resist deliberate attacks intended to gain unauthorised access to confidential information, or make unauthorised modifications to information or to the program so as to provide the attacker with some advantage or as to deny service to legitimate users
- Reliability—the capability of the software to maintain the level of performance of the system when used under specified conditions
  - Maturity—the capability of the software to avoid failure as a result of faults in the software
  - Fault tolerance—the capability of the software to maintain a specified level of performance in cases of software faults or of infringement of its specified interface
  - Recoverability—the capability of the software to re-establish is level of performance and recover the data directly affected in the case of a failure
- Usability—the capability of the software to be understood, learned, used and liked by the user, when used under specified conditions
  - Understandability—the capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use
  - Learnability—the capability of the software product to enable the user to learn its application
  - Operability—the capability of the software product to enable the user to operate and control it
  - Attractiveness—the capability of the software product to be liked by the user
- Efficiency—the capability of the software to provide the required performance, relative to the amount of resources used, under stated conditions
  - Time behaviour—the capability of the software to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions
  - Resource utilisation—the capability of the software to use appropriate resources in an appropriate time when the software performs its function under stated conditions
- Maintainability—the capability of the software to be modified
  - Analysability—the capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified
  - Changeability—the capability of the software product to enable a specified modification to be implemented
  - Stability—the capability of the software to minimise unexpected effects from modifications of the software
  - Testability—the capability of the software product to enable modified software to be validated
- Portability—the capability of the software to be transferred from one environment to another
  - Adaptability—the capability of the software to be modified for different specified environments without applying actions or means other than those provided for this purpose for the software considered
  - Installability—the capability of the software to be installed in a specified environment
  - Co-existence—the capability of the software to co-exist with other independent software in a common environment sharing common resources
  - Replaceability—the capability of the software to be used in place of other specified software in the environment of that software

Figure 1.1.: The ISO 9126 software quality characteristics and subcharacteristics, taken from (Punter et al., 1997).

the identification of the quality model. Punter argues that the only way to make this choice less subjective is to document and justify it extensively. In particular, each item on the list must be clearly related to the characteristic or aspect of the software whose quality it is supposed to indicate.

For COTS software, Carvallo et al. (2004a) and Botella et al. (2002) have suggested a process to refine the ISO 9126 standard characteristics, to arrive at a quality model for evaluation. Even a tool has been developed which supports this process and provides a formal model of the resulting quality attributes (Carvallo et al., 2004b). However, the actual identification of basic attributes is still left to the evaluator in this process.

A more empirical approach of how to arrive at a quality model (thus at items on a checklist, or at evaluation criteria) is given by Brown and Wallnau (1996). These authors suggest to identify those features of a technology that distinguish it from existing technologies. The authors call such distinctive features "technology deltas". Thus they stress that a product should be evaluated with respect to competitive products. This method ensures that no quality attributes are overlooked by the evaluators. It is particularly useful for functional criteria. Secondly, Brown and Wallnau (1996) stress that the technology deltas should be evaluated in well-defined, sharply focused usage contexts, because then the extent to which a technology delta supports a given context can be evaluated. The importance of distinctive features is supported by Maiden et al. (1997), who found that they costly evaluated some requirements which were, in the end, met by all candidate products among which they had to select. These authors also point to the usefulness of test cases, in terms of which the requirements can be stated. The present work includes a test case that can be used for that purpose, see section 1.4.

The distinction made at the beginning of this subsection, between the development of a software and its use as a product, serves the clarity of description but does not imply that there are no connections between these aspects. Obviously the quality of the source code and the development process influences the quality of the finished product; hence, some research exists that addresses these connections. For example, Punter (1997) stresses that the results of a software product evaluation are interesting for the developers of the software as well as for the potential buyers. Mayrand and Coallier (1996) and others relate the internal design of software to some external quality attributes. Similarly, April and Al-Shurougi (2000) map features that are based on the source code of a software to the ISO 9126 characteristics.

As regards metrics (see the second phase of the standard evaluation process above), obviously no internal, source-code related metrics can be used for COTS products (Colombo & Guerra, 2002). Previous research on product metrics has mainly concentrated on such internal metrics (e.g. (Mayrand & Coallier, 1996; Cartwright & Shepperd, 2000)). Research on COTS evaluations has concentrated on process-oriented aspects (Maiden et al., 1997; Carvallo et al., 2004a) but has not established quantitative metrics, except for Rangarajan et al. (2001) and Colombo and Guerra (2002). In (Rangarajan et al., 2001), rather general metrics are given, only some of which are external, but require much effort to measure (such as the percentage of design goals met by the finished product). In contrast, Colombo and Guerra (2002) seem to use a metric similar to the one developed in the present work (section 1.2.2), but no details are given, nor any examples from a concrete evaluation project.

It can be seen that the goal in software product evaluation is not to arrive at one single metric that indicates the quality of a software, as the notion of quality is too complex for this; rather, the derivation of a detailed picture involving different aspects of quality, some of which can be in conflict with each other (Barbacci et al., 1995), is recommended. Though scores from a checklist can be integrated into a single value if desired, usually this is not the goal of an evaluation. Instead, the complete checklist scores are needed to arrive at an informed opinion about a product. Section 1.2 explains how the evaluation of KDD software products was performed for the present work, in the light of the guides cited above.

1.1.2. KDD product evaluations

One of the first attempts to systematically evaluate KDD tools is (Abbott et al., 1998). This evaluation is based on a given application purpose (fraud detection). In order to handle the large number of tools then already available, the authors applied a three-stage approach. In the first stage all tools were evaluated under rather broad and simple criteria, such as support for the intended system environment, or range of algorithms provided. This stage left 10 products for the second stage, which filtered 5 products for the final examination using the additional criteria quality of technical support, and exportability of models, for example to source code. The last criterion relates to the deployment phase in KDD.

In the final stage, five tools remain and are examined under five well-discussed criteria. These are: (i) support for client server settings, which the authors deem related to scalability; (ii) automation of parameter search and documentation of experiments; (iii) range of algorithms and options offered for each algorithm; (iv) ease of use in data manipulation, mining, visualisation and technical support; and (v) accuracy of neural nets and decision trees on a dataset from the authors' application. Mainly points (ii) and (iv) are of relevance for this work. Concerning data preparation, they distinguish between loading the data and manipulating it. During data load, automatic recognition of data types and naming of attributes is an issue. This criterion is taken up in section 1.3 (criteria 14 and 19). Data manipulation is not discussed to a great extent, only the availability of built-in functions for attribute derivation is briefly discussed. This criterion shows up in section 1.3 as the extent to which primitive operators are supported by KDD tools (criterion 31).

Among their lessons learned is the requirement to define what a tool is going to be used for, in order to focus the evaluation. Reasonable though this is, it is not applicable in this work, which attempts to find application-independent criteria. More relevant is their suggestion to test a tool in the environment where it is going to be used; as all the criteria listed in section 1.3 are based on the experiences made with the different tools when implementing a model application, this requirement is fulfilled in the present work.

Another early attempt to give a systematic overview of different KDD software tools is (Gaul & Säuberlich, 1999). They consider the whole KDD process insofar as they examine only tools that offer some data preparation and deployment facilities, not only mining features. They list 16 tools and give the following features for them: manufacturer, available mining algorithms, system platform, price, year of first version, support for parallel environments, and limitations on data set size. For 12 out of the 16 tools, they give

some further information in a second table with boolean entries indicating presence or absence of certain features. Concerning data preparation, they only consider the presence or absence of the operators MISSING VALUE REPLACEMENT, ATTRIBUTE DERIVATION, ATTRIBUTE SELECTION and SCALING, plus some unexplained operator STANDARDISATION. Concerning deployment, they consider exportability and visualisability of models. The closest they come to conceptual aspects is the presence or absence of graphical user interfaces.

A more extensive list of classification features is provided by Goebel and Gruenwald (1999). These authors discuss three groups of features: general product characteristics, database connectivity, and data mining characteristics; they give tables with information for each feature for 43 tools. Of certain interest for this work is their stress of the importance of database connectivity. They claim that a KDD tool ought to be tightly integrated with database or data warehouse systems. Indeed, the large volumes of data typically involved in knowledge discovery make this issue paramount for modern KDD software. Thus they consider the data formats a tool can access, in particular certain file formats and databases, as well as data models (relational vs. single table), query options (SQL for databases, or GUI support), data types supported, and size limitations on the data set.

Concerning data mining characteristics, they distinguish between tasks such as clustering or prediction, and methods to solve the tasks. Data preparation is only considered by a single boolean flag indicating whether a tool has any preparation facilities at all.

A paper that considers data preparation features of software tools in some more detail is (Collier et al., 1999). This paper is also interesting in that it suggests a simple methodology to choose a most suitable tool from a list of tools, using a weighting scheme. While the authors do not relate their methodology to standard software product evaluation methods, see section 1.1.1, it is easy to see that their weighting scheme corresponds to the written procedure that prescribes the assignment of values for a metric, in phase 2 of the standard evaluation process according to ISO 14598. Though such a scheme is not new, the authors applied it to knowledge discovery software for the first time. The authors point out that the investigation of some effort into the choice of a suitable tool will pay off easily, considering the work saved later in the application. Indeed, the total costs of ownership (TCO) of KDD software are hardly influenced by the licence fees, but much more by how much expert work the software can save.

Collier et al. (1999) also apply tool selection in two stages, filtering the bulk of tools away in the first stage under simple but hard criteria, such as support for the intended system environment. The second stage is more refined in their approach, however. Having grouped selection criteria into five groups (performance, functionality, usability, data preparation, other), they assign weights to the criteria in each group such that the sum of weights within a group equals 1.0. The groups themselves are also assigned weights. The authors then propose to choose one of the candidate tools as *reference tool*; one could choose a personal favourite tool based on past experiences of some of the evaluators, but any candidate can be used for reference. Then, each tool is given a score in each criterion that measures its strength *relative* to the reference tool. The score is assigned by human evaluators who have some experience with the tool. The reference tool gets a medium score in all criteria. Finally, the weighted scores of all tools imply a ranking for tool

selection.

Focusing on these authors' data preparation criteria, they use mainly the presence and quality of the following data preparation operators: VALUE MAPPING, ROW SELECTION , DISCRETISATION, ATTRIBUTE DERIVATION, and MISSING VALUE REPLACEMENT. The brief discussion points out that an extensive list of functions is needed for ATTRIBUTE DERIVATION. Also, exportability of models is a criterion. Finally, one interesting criterion is called *Metadata manipulation*; it assigns a score based on the availability and manipulability of data descriptions and data types. Section 1.3 will develop rather more detailed criteria based on the ways of handling metadata supported by a tool.

A thorough study on data mining software solutions is the book by Gentsch et al. (2000), which provides detailed descriptions of 12 tools. For direct comparison, this study considers seven rather broad criteria that summarise the detailed descriptions. These are data import, data transformation (preparation), mining methods, visualisation of data and models, handling (usability), documentation, and special aspects (strengths of each tool in areas not covered by the other criteria, such as integration with other tools, code generation from models (criterion 3 below), etc.). Data import is related to the support of data types. The authors stress the importance of data preparation and mention the preparation operators that each tool provides in their detailed descriptions. They consider ATTRIBUTE DERIVATION, VALUE MAPPING, AGGREGATION, SCALING, and MISSING VALUE REPLACEMENT. However, the discussion of preparation operators is not done in a systematic way, as it is not based on a (minimal) list of operators. Because the study comprises the whole KDD process, data preparation is just one aspect and is not discussed in any detail, though its importance is pointed out clearly.

Another list of criteria is suggested by Giraud-Carrier and Povel (2003). While an evaluation based on the criteria is not included in the paper, the criteria list is rather extensive. This discussion focuses again on the criteria related to data preparation. Their criteria include the presence or absence of facilities for: reading data from flat files, databases or XML files; data characterisation by statistical measures; data visualisation; row selection; attribute selection; and data transformation, under which point any other preparation operators seem to be subsumed. Data cleaning (outlier detection) is also mentioned but not included in the final criteria list.

An example from a slightly different field is (Maier & Reinartz, 2004) which examines web mining tools. When mining data from web server logs, special preprocessing operations are needed to bring the data into attribute-value format, which is the input for data preparation as discussed in this work. The availability of some such preprocessing operations is included in the criteria list set up by Maier and Reinartz (2004).

1.2. Methodology

Several methodological deficiencies can be recognised in the previous work as discussed in section 1.1.2:

- The evaluations do not follow an accepted, standard evaluation procedure, nor do they use standard quality characteristics or concepts.

- The list of evaluation criteria is not justified in a systematic fashion, and is often rather short.
- Many approaches use boolean criteria which, on the one hand, often subsume many important aspects under one yes/no-flag, while on the other hand an overview is hard to keep if there are many criteria.
- No metric to flexibly quantify the degree to which a tool fulfils the criteria is given.
- No detailed methods how to apply the criteria to new tools are given.

This section explains the methodology used for tool evaluation in this chapter, which

- employs the conceptual level to abstract from technical details, thus allows to compare all criteria across tools and applications easily;
- follows the ISO 14598 standard of a software product evaluation process, but adds some aspects to it;
- systematically develops a list of evaluation criteria by following the notion of "technology deltas" by Brown and Wallnau (1996), see section 1.1.1;
- introduces *n-of-m* criteria as a concise, quantitative metric for complex quality characteristics, where the assignment of values can be done objectively and reproducibly;
- is adaptable to various levels of detail, thus to various audiences;
- uses all evaluation criteria found in previous work, and adds many more;
- is independent of human subjective evaluation;
- considers the complete KDD process;
- employs the list of operators from (Euler, 2005b) as another source for systematic evaluation; and
- provides a test case that allows a step-by-step evaluation of all criteria on new tools.

In the following, the methodology is developed following the four phases of the standard product evaluation process introduced in section 1.1.1. See also (Euler, 2005a).

1.2.1. Establishing evaluation requirements

The ISO 14598 standard requires the specification of the purpose of the evaluation, the type of products to be evaluated, and the quality model in this phase. The purpose of the evaluations in this chapter is to provide a detailed, yet clear picture of the strengths and weaknesses of currently available software tools that support KDD applications. It is not the purpose to test any software, nor to evaluate the tools under general software criteria such as reliability, portability or maintainability. Nor is it the purpose to select

a single best tool or to give recommendations about tools; rather, a general framework is developed that allows the evaluation of further KDD tools easily.

The evaluation is restricted to such KDD products that include strong data preparation facilities, but cover the complete KDD process, and provide at least some conceptual support. Tools that offer only mining algorithms, with little or no data preparation, are excluded.

The quality model used in this work follows the purpose of the evaluation. The strengths and weaknesses of a tool are examined in the light of the conceptual aspects developed in previous chapters, which are in fact KDD-specific. Thus *only functional* criteria are applied. Hence, all the criteria used in the quality model here, which are listed in section 1.3, bear on the quality characteristic "Functionality", in particular its subcharacteristic "Suitability", in that they are used to examine the capability of the software tools to provide the set of functions that have been found to be appropriate for KDD tasks and objectives in the previous chapters.

The development of the criteria list followed the idea of technology deltas introduced in (Brown & Wallnau, 1996). This approach is particularly useful for functional criteria. Though the present work does not use the history of a technology to identify new, distinctive features, as Brown and Wallnau have done, it compares features of different products in order to identify the distinctive ones. A feature is deemed distinctive if it is present in one or more tools, absent in one or more other tools, and considered useful in the sense that it supports some of the conceptual aspects developed in the previous chapters. In this way a list of criteria is gained that provides a maximum amount of information when comparing the tools based on them. However, some additional criteria are also introduced by a different approach, namely following the conceptual ideas from (Euler, 2005b), of which some do not appear in any currently available tool. A similar approach was taken by Romei et al. (To appear) to derive some requirements that these authors' tool had to meet: these requirements are found through a conceptual analysis of the tasks their tool was going to perform. (These requirements can also be seen as tool comparison criteria, but they are too superficial to allow rigorous evaluations like in the present work.) Thus two sources for the development of criteria are used here that cannot be expected to be available in general in software product evaluation, namely inter-product comparisons and a systematic, thorough description of the *desired* functionality.

In line with this conceptual approach, the evaluation criteria address those functionalities of a KDD tool that are explicitly supported in the user interface. For example, some tools offer a scripting language that enables the execution of a graphically modelled process from outside the tool. The power of the scripting language can sometimes be exploited to achieve some functionality that is not offered in the user interface, for example the automatic testing of parameter settings (see criterion 49 in section 1.3.3). However, in such a case, the criterion is not fulfilled because no high-level support is given for this functionality. The aim of this chapter is to provide measures for the conceptual support in KDD, that is, for the potential of a tool to save user efforts, and low-level programming is likely to require rather more than less user efforts.

1.2.2. Specification of the evaluation

Having found the quality model in the previous phase, each of its criteria is now assigned a metric, in the sense defined in ISO 14598 (see section 1.1.1). During work with the various KDD tools, most of the technology deltas identified corresponded to rather small, specific features, which are present in some tools and absent in others. A simple metric would assign a boolean value to each feature, indicating either its presence or absence. This would lead to a very long list of criteria, counteracting the evaluation goal stated in the previous phase of providing clear overviews of each tool's strengths and weaknesses. However, many small groups of features were found to be related in a rather natural way. Therefore, such naturally related features are grouped together in this work, and each group forms a criterion. The *n-of-m* metric is used to indicate the strength of a tool with respect to such a criterion: m is the number of features grouped together for this criterion, and n is the number of features that are present in the given tool. Thus each n-of-m criterion could be transformed into m boolean criteria. A simple score can be assigned to each tool under each criterion, which is the real value $0 \leq n/m \leq 1$.

This method allows much flexibility concerning the groupings of the basic features. For a quick overview or superficial comparison, only the more important features can be used, or larger inherently related groups can be formed. This corresponds to larger average values of m. For detailed surveys, like in this work, more fine-grained criteria can be used, so that the list of criteria is longer but the average value of m is lower. Thus the n-of-m method is adaptable to different granularities of detail, leading to different representations of the same evaluation scores. The different representations can be used for different audiences, like technicians or developers compared to decision makers. Section 1.6 provides two representations of the evaluation data collected for this work.

The measures for several single criteria can be combined to more integrated scores by building weighted sums, where the sum of the weight coefficients should be 1.0. For example, to assess the strength of a tool in data handling, all criteria listed under that heading in section 1.3.3 can be evaluated and combined to a single value. If desired, a single global score could be computed for every tool to get a ranking of the tools, though such a ranking would hide many aspects that the detailed score list can provide.

Some features could not be related to others and are listed as boolean criteria. These features should take one of the values 0 or 1.0 in order to be integratable with other criteria.

Though the above metrics are recommended for the type of criteria in this work because they are simple, transparent, and easily combinable, other scoring methods are applicable based on the given criteria list as well. For example, the method by Collier et al. (1999), described in section 1.1.2, can be applied as well as a simpler scoring method described in (Maier & Reinartz, 2004). Since each evaluator is likely to have their own priorities with respect to their application, the choice of the scoring method is open in this methodology. In section 1.6, which presents the results of some evaluations done for this work, the recommended metrics above are used.

The methodology described here results in objective criteria, with a written procedure that prescribes how to identify the presence or absence of each feature in a criterion. The procedures are given with each criterion in section 1.3, fulfilling the demand of objectivity and reproducibility. Further, a test case is provided in section 1.4 that provides clear

explanations about how to evaluate each criterion based on a concrete example.

Though the methodology sketched here relies on inter-product comparisons for the development of criteria (see previous phase), it provides a set of criteria that can be applied to single software products, in contrast to the method by Collier et al. (1999) which is described in section 1.1.2, and which relies on inter-product *scores*.

A limitation to this methodology may be that, when applied to a different type of software products, not all technology deltas might correspond to boolean features that can easily be grouped. Some features, such as performance-related features, require a real-valued, continuous scale. However, such metrics can be mapped to the real interval [0..1] easily, which makes them easily combinable with n-of-m metrics. A more serious limitation is that different n-of-m criteria can result in identical values when evaluated, although the respective values of n and m are different. It is not clear whether the fulfilment of 2 out of 4 features of a criterion "means" the same strength as the fulfilment of 4 out of 8 features. Further, the features within a criterion are not weighted or prioritised here, though this could be added easily. However, to compare the tools under any given criterion, the same value of m is always used, which makes the metric valid unless different criteria are compared, which makes no sense anyway in most cases.

1.2.3. Design of the evaluation process

The initial experiments for this work were done by implementing a model application in a number of tools. As the model application is based on two complex real-world applications, profound experiences could be made about a large number of issues that typically arise when realising complicated KDD processes, and about how different features of the tools support the implementation. This allowed to identify the technology deltas and develop the criteria as explained above.

However, now that a list of criteria is available, a simpler evaluation plan can be given. Section 1.4 describes a procedure to implement a test case in an arbitrary KDD tool and check various criteria in every step of the procedure. All criteria are covered. This corresponds to an evaluation plan, though elements like resource assignment are missing, as they are not applicable: the evaluation can be done by a single evaluator, and does not consume big computational resources. Hence, no team coordinations or fixed schedules are needed. The main costs are likely to be incurred if the evaluator is new to the tool to be evaluated. In this case, the average time the evaluator needs to find out whether and how the given tool supports a functionality that is examined in a certain step of the test case will dominate the overall costs. This situation can be different, though, if external stakeholders (paying clients, for instance, who impose deadlines or other resource restrictions) need to be taken into account when executing the plan.

1.2.4. Execution of the evaluation

Executing the evaluation consists of following the execution plan, taking the measurements required by the criteria, and documenting them. The results of several such evaluations performed for the present work are presented in section 1.6.

1.3. Criteria for KDD tool evaluation

This section presents the criteria that were developed following the methodology described in section 1.2. As explained there, each criterion is accompanied by a precise description of how to evaluate it in an arbitrary KDD tool. This serves not only tool selection by end users but can also provide guidelines for developers of new tools. No tool covers all aspects discussed in this section; rather, the elaborations here can be seen as describing an "ideal" tool, towards which existing solutions should be developed.

This section first discusses some criteria whose detailed examination is excluded from this work, in section 1.3.1. This is followed by a discussion of some more general criteria, in section 1.3.2, which have been found in the literature on KDD evaluations (section 1.1.2), or have been mentioned in previous chapters. The relation of these criteria to the more detailed criteria that are based on the methodology used here is explained. Those more detailed criteria are listed in section 1.3.3. They form a main contribution of this work.

For ease of reference, every criterion receives a number. The order of presentation of criteria is not significant. A list of all criteria with a reference to the page on which they are described can be found in appendix A on page 36.

1.3.1. Excluded criteria

As section 1.2.1 explains, only functional criteria are used in this work. From the perspective of the KDD process, only criteria pertaining to the more technical KDD phases data preparation, mining and deployment are developed, as conceptual support concentrates on these phases, while business and data understanding do not lend themselves so well to conceptual modelling.

One important aspect of KDD tools concerning the mining phase is obviously the range of learning algorithms they provide, as well as the range of parameters that can be set for each algorithm. In contrast to data processing operators, no minimal or complete list of algorithms, or even parameters for one algorithm, can be identified, because the sets of algorithms and parameters are open and likely to be extended by research progress in the future. Even today no single tool offers all varieties of algorithms that have already been described in the literature. Approaches to include the range of mining algorithms could perhaps be based on an ontology of mining tasks and algorithms, such as the one given in (Cannataro & Comito, 2003), but there is no accepted standard ontology yet. Therefore, the range of learning algorithms and parameters, which has often been used as an evaluation criterion in previous work (see section 1.1.2), is not used as a criterion here.

In spite of this, the methodology developed in this chapter is also applicable to the mining phase. The approach used here is to judge the extent to which a mining tool supports basic, mining-related processing and control steps such as automated parameter search, cross-validation, or ensemble learning with arbitrary base learners. However, it has to be said that not many tools offer strong coverage of such conceptual aspects of *both* the data preparation and mining phase. Therefore, separate evaluations might be appropriate for each phase.

Some studies from section 1.1.2 have used the accuracy of learned models on a given

data set as a criterion to compare KDD tools. However, a ranking of tools based on one data set is not necessarily similar on a different data set, which is why model performance related criteria are not used in this work.

An important criterion in practice is execution speed. Despite similar architectures, different tools can reveal substantial differences in terms of processing speed. Since speed is highly dependent on the hardware infrastructure used, actual performance times are of little worth, but the ranking of tools that they imply can be expected to be consistent across platforms. This criterion does not concern a conceptual, functionality-related feature, but is directly related to the ISO 9126 subcharacteristic "Time behaviour" of characteristic "Efficiency". Therefore it is not used in this work.

Rather detailed criteria might be developed concerning the visualisation of data sets, data characteristics and learned models or functions. Many tools that were examined offer some visualisation features, but they are difficult to compare as each tool has its particular emphasis on certain visualisation methods. Visualisations of models or learned functions are not comparable if a tool lacks the mining algorithm whose visualisation is the strength of another tool. For data sets with more than three attributes, any visualisation of the data must include a dimensionality reduction, which is useful for human understanding but not necessarily helpful for mining. Further, visualisation of data sets and data characteristics mainly belongs to the data understanding phase of the KDD process, while this work focuses on aspects related to data processing. For these reasons, visualisation issues are not included here. Similarly, reporting functionalities, which some KDD tools offer to ease the production of documents reporting on the results of a KDD process, are not examined in this work. This includes facilities to draw charts based on mining performance or similar, statistical data produced during a KDD application.

A set of criteria that is left out from this section concerns the general software quality characteristics which are not specific to KDD tools, in line with the purpose of this work as described in section 1.2.1. This does not mean that general software quality issues are irrelevant for KDD software, only that they are not in the focus of this work. Criteria related to these issues can be found in the literature discussed in section 1.1.1.

Finally, as a related point, recall from section 1.2.1 that any functionality listed in the criteria below can not be fulfilled by low-level constructs such as integrated programming languages, but must be explicitly provided in the user interface in order to count as conceptually supported.

1.3.2. General criteria for KDD software

This subsection lists some criteria for KDD software that can be found in the literature cited in section 1.1.2, or in the previous chapters, but are not directly included in the list of detailed criteria developed in this work. The purpose of this subsection is to relate these more general criteria to the detailed criteria where possible, in order to ease the recognition of criteria known from the literature, or known from previous chapters, as the terminology cannot always be identical. Note the remarks on excluded criteria in section 1.3.1, though. This work's detailed criteria follow in subsection 1.3.3.

1 Adaptability: Mainly the addition and deletion of conceptual metadata, as well as the *propagation* of such changes, must be supported. In general, adaptation and reuse are easiest if the software follows the two levels approach proposed in (Euler, 2005b) in the area of data sets. Therefore this criterion is related in particular to criteria 15, 17, 18, 19, 24, and 27.

2 Scalability: KDD software has to be capable of handling large data sets. Some studies in section 1.1.2 have included limits on the number of attributes or rows that a software can process as a criterion. However, the tools examined for this work do not explicitly state such limits, so that any such limitations depend on hardware resources. Other studies have stressed the importance of support for parallelisation or for client-server environments, where the server deals with large data sets and control is executed from the client. Criterion 8 is the main related criterion in this work.

3 Interoperability: If users want to use special software for some subtask, such as reporting or mining, they should be able to do so easily. In KDD, the interface to other systems is often a data set on the file system or in a database, so that this criterion is related to the data formats a tool supports (criterion 7); yet if mining results (like learned rules) are to be used outside the tool, integration with other software can be more difficult. See criteria 51 and 54.

4 Guide to KDD process: The software should offer some support to guide the user through the complex stages of a KDD process, and avoid erroneous user inputs. This support should be offered for several levels of assumed previous experiences of users. Related criteria are 25 and 53.

5 Documentation: It should be possible to add free text comments to every object involved in the KDD process. As all tools examined for this work offer facilities for this, it is not a technology delta but is obviously very important, especially for the reuse of process models.

6 Business problem: Some approaches have attempted to use the types of business problems a tool can solve as a criterion. In the absence of a theory on how business problems are related to mining tasks, they have used very simple mappings of stereotypical business problems to mining tasks, so that this criterion is related to the range of mining algorithms, which is not discussed in this work as explained in section 1.3.1.

1.3.3. Special criteria for KDD software

This subsection lists the criteria that were developed following the methodology described in section 1.2. They are categorised into a number of areas and discussed in the following under each area's heading. As section 1.2.2 explains, a criterion consists of m boolean features (questions), but in principle the number (m) and the exact grouping of questions for a criterion is flexible. Given all features, different groupings into criteria can be formed to reflect purpose-specific aspects. One may also choose to leave out some features with

low priority. Since priority is application-dependent, no weights are given to features or criteria in this work, but the grouping of features into criteria here is a recommendation based on experiences made during the implementation of the model case. Further, this particular listing of criteria allows a quantitative, detailed, yet clear comparison of KDD tools, as demonstrated in section 1.6. Finally, the test case described in section 1.4 is designed to check exactly these criteria.

Data handling

The two common types of data sources are flat files and databases. Thus one criterion is the ability to load data from at least these sources:

7 Data formats: At the very least, flat files in various common formats, such as comma-separated or sparse representations, and ODBC, the open database connection standard, must be accessible. This applies to data input and output. More precisely, the following boolean features form this criterion ($m = 6$):

- possibility to read flat files at all;
- possibility to specify any column-delimiting character when reading flat files;
- possibility to read the first line of a file as attribute names;
- possibility to read tables via ODBC;
- possibility to read sparse representations;
- applicability of all of the above to both input and output of data.

However, once the data is loaded into the KDD software, every data transformation step produces intermediate data tables. When handling large data sets, storing all intermediate tables would multiply the size needed by the original data set by a large factor (roughly corresponding to the number of data preparation steps). Thus only some intermediate results should be stored. The question of data storage is an important feature to distinguish KDD software tools. Some leave all processing to the database, so that the data never leaves the database. Others rely fully on the local file system.

Processing in databases has the advantage that structured search is possible on every intermediate concept, and that the use of views allows this essentially without consuming extra storage. Further, databases are usually installed on fast hardware with large storage devices; see also (Musick & Critchlow, 1999). However, database management systems include features such as transaction safety and concurrent access, which are not essential for KDD but may slow down processing. In contrast, using the file system might be faster, but does not allow structured search on intermediate results; further, the file system of the workstation from which the KDD application is controlled may not be sufficient to handle large volumes of data. This pertains more to data preparation and deployment than to mining, as the latter should be performed in main memory anyway.

In order to compare the two data handling approaches, a few experiments were done in the context of this work. A short data preparation chain with three attribute derivations and one attribute selection was applied to an artificial call detail records (CDR) table with 61 million records. This data preparation chain was executed using four settings:

1. inside the database, starting and ending with a materialised table (DB to DB);

Setting	Time for short processing chain in minutes	Time for complete model application in hours
DB to DB	129	39.4
DB to file	104	
File to file	29	7.8
File to DB	68	

Table 1.1.: Comparison of execution times.

2. reading from the database table, processing in main memory of the client (in batches that fit into main memory) and writing to a result file (DB to file);

3. reading from and writing to a file, processing in batches in main memory of the client (file to file); and

4. reading from a file, processing in batches in main memory of the client, and writing to a database table (file to DB).

The last setting can be relevant when data is collected from different sites to a central database, for example in distributed data mining scenarios. Depending on the application, one may want to prepare the data before combining it with data from other sources, in order to reduce the global amount of data. Thus there is some data preparation to be done on the distributed clients' file systems before loading the data to a central site.

Setting 1 was implemented in the KDD tool MiningMart which accessed an Oracle database installed on a Sun Enterprise 250 server with 1.6 GB of main memory and two 400 Mhz CPUs. For the other three settings, Clementine was used in the standalone version without a server, on a Windows client with 512 MB main memory and a Pentium 1600 Mhz CPU which was connected to the Oracle database via ODBC and a 100 Mbit/s Ethernet connection. The two KDD tools are described in section 1.5. The data table that was processed takes more than 2 GB of storage space in the database, so that processing could not take place completely in main memory in any setting.

Table 1.1 shows the execution time for each setting. In setting 2 and 4, most of the processing time is spent on reading or writing to the database, respectively. The purely file system based processing (setting 3) is fastest. This finding is repeated when the processing time of the complete data preparation part of the model use case is compared for the first and third setting. It is also consistent with the experiments reported in (Musick & Critchlow, 1999) for general data access, and (Sarawagi et al., 1998) in the data mining context, where rule mining was tested in a number of different data handling scenarios, and caching data on the file system turned out to be the fastest scenario. An interesting approach to remedy the database efficiency problem is presented by Gimbel et al. (2004), who use pipelining and a management strategy to keep the data sorted according to various indexes. This approach explicitly takes KDD-related operations into account. However, it is not yet implemented in practically used database management systems. In practice today, whether the advantages of structured search and efficient storage that

databases offer are worth the performance loss is dependent on the application. So the next criterion is obtained.

8 Data handling: Ideally, the KDD software should be able to process data both inside a given database and on the file system. If both places of processing are possible, the user must be able to specify which one to use at any point in the processing graph. This allows to distribute the computing load to the appropriate hardware. Hence, $m = 3$.

In databases, views can be employed for intermediate results, which take essentially no extra storage space. However, processing deeply nested views as resulting from a long sequence of data preparation operators is slow, so that a materialisation should take place at regular points in the data flow. Most suitable are those points in the operator dependency graph where the output of one operator is consumed by several other operators. A similar argument holds for flat file based processing: here it must be possible to state which intermediate tables of a preparation chain should be stored on the file system. There should be a mechanism for this which is independent from dedicated file input/output operators, as the latter interrupt the preparation graph inconveniently. This leads to the following criterion.

9 Caching control ($m = 2$):

- possibility to specify the points of materialisation/data storage during processing;
- independence of this option from dedicated input or output nodes.

10 Caching size estimation: The size needed for storing an intermediate table to a file, or for materialising a table, should be estimated, at least roughly, by the software before the execution of the preparation graph. How such estimations can be done based on metadata is discussed elsewhere. See also criterion 24. This criterion is boolean.

11 Automatic caching: For long preparation chains or at end points of the data flow, caching of results should be automatically done, or at least offered, by the software, so that no resource-intensive process is started whose results are inadvertently not stored. Moreover, sometimes caching is required by specific circumstances of which the user may not be aware. In two tools examined for this work, long preparation chains on big data sets had to be separated into several parts, each storing the output in a specific file for the next part to read from, because otherwise some hidden temporary files that the tools employed to store the several intermediate results (rather than only one result as when caching) would have got too large for the available disk space. Thus the need for caching must be recognised by the software. So $m = 2$:

- automatic caching at end points of a process is done;
- automatic recognition of the need for caching is done.

12 Caching transparency: The files used for caching, or the materialised tables, must be accessible, and must be clearly linked (e.g. by name) to their concept or to the operator which outputs the data stored in them. This enables the user to follow the data storage processes and arrive at own estimations of resource consumption. So $m = 2$ (accessibility and linkedness).

13 Data inspection: Intermediate data tables (the extensions of the concepts) must be inspectable from the tool. This facilitates the control of the ongoing work by the user. This criterion is boolean.

Metadata handling

The term *metadata* refers to data about data; in this work, it subsumes names of attributes, data types (technical and conceptual ones), attribute roles, and data characteristics such as the set of values occurring in an attribute, or the number and distribution of these values.

14 Attribute import: The names of attributes must be automatically recognised from database sources. For flat files, it is common to reserve the first line of the file for attribute names; if such files are read, this must be recognised by the software. Also, common formats storing attribute information in a separate file, such as ARFF, should be supported. However, attribute names must not be fixed. Thus $m = 3$:

- automatic recognition of column names, to be used as attribute names;
- possibility to edit attribute names;
- support for reading attribute information from a separate file.

15 Conceptual data types: The distinction between the actual storage type of data and the way it is used conceptually should be made. This criterion is boolean.

16 Type recognition: A strong mechanism to automatically recognise the technical as well as the conceptual data types of all attributes when importing data is a must. For large data sets, recognition can take quite some time; thus recognition based on a sample of the data, or recognition at a later point in the preparation graph, must be supported, and this must be controllable by the user. The user must have the option to specify the types by hand, to avoid long recognition processes on large data sets, or to correct wrongly recognised conceptual types. Hence $m = 5$:

- automatic recognition of technical data types;
- automatic guessing of conceptual data types;
- possibility for the user to specify when recognition takes place;
- availability of recognition based on a data sample;
- possibility to specify and change the conceptual data type by hand (at any time, but this is subsumed under the third point).

17 Flexibility of type mapping: The relation between the conceptual data type of an attribute and its technical counterpart must be transparent, flexible and controllable by the user. It must be changeable at any point in the preparation graph, not only at the beginning when data is imported. Thus $m = 3$ (transparency of mapping, changeability by user, and independence from import).

18 Robustness of type mapping: If a preparation operator uses a technical data type in a way not consistent with the conceptual data type it is currently mapped to, the mapping ought to be changed, perhaps with a warning to the user, but this should not lead to an error as it is a rather common situation. This criterion is boolean.

19 Data characteristics recognition: Similarly to type recognition (criterion 16), the set and distribution of values occurring in the data must be recognised by the software during import or at a later point, based on the entire data or on a sample, and control over this must be given to the user. For continuous attributes, the range of occurring values instead of the set of all values should be stored. Again, it must be possible to specify all of this information manually. See also criterion 35. This criterion can be extended by boolean features concerning further data characteristics, such as average values, number of unique values in an attribute, and so on. Here, $m = 6$:

- recognition of range of values of a continuous attribute;
- recognition of distribution of values of a discrete attribute;
- recognition of the number/percentage of missing values;
- possibility for the user to specify when recognition takes place;
- availability of recognition based on a data sample;
- possibility to specify and edit data characteristics by hand (at any time, but this is subsumed under the third point).

20 Data characteristics deployment: The data characteristics from criterion 19 must be available during the declaration of the KDD process in the tool. For example, for the operator VALUE MAPPING, the values of the selected input attribute (to be mapped to other values) must be selectable during operator specification. For this functionality, the availability of data characteristics in separate charts or tables is not enough. This functionality is the boolean criterion.

21 Attribute roles: Support for four roles must be given. That is, the user must be able to specify a role for each attribute. Thus $m = 4$ for the four roles *label*, *predictor*, *key* and *no role*.

22 Attribute matching: On some occasions, attributes of different concepts are mapped to each other. For example, when joining concepts, no duplicate attributes must occur in the output concept, even if the same attribute is present in more than one input concept, a situation that should be recognised by the software; at the same time the keys of the input concepts must be matched. Similarly, the UNION operator requires a matching of all attributes of the input concepts. As another example, when importing learned models

that were exported using PMML (see criterion 54), the attributes on which the model is applicable must be matched to the attributes in the concept to which it is going to be applied. The KDD software can save work by recognising matchable attributes by name and conceptual or technical data type, especially as hundreds of attributes per concept are not uncommon in some applications. However, of course the matching must be editable by the user. Hence $m = 2$ (automatic matching and editability).

23 Data type inference: When deriving a new attribute, its technical data type must be inferred. The conceptual type is never uniquely determinable but can be guessed; default types often suffice. So $m = 2$:

- inference of technical data type of derived attributes;
- guessing of conceptual data type of derived attributes;

24 Metadata inference: Inference and optimistic guessing of data characteristics are introduced elsewhere. How and when data characteristics can be inferred or guessed can be specified for each preparation operator. Inference is to be contrasted with re-recognition.

The importance of this criterion can be seen when noting that some operators, like PIVOTISATION, rely on knowledge of which values occur in an input attribute (for pivotisation, the index attribute). Tools in which this information is neither inferred nor manually editable force the user to execute all steps that lead to the operator where this information is needed, in order to then recognise the available values automatically in the input. This situation was actually encountered by the author when implementing the model use case in some tools. However, the execution may take a long time, which is unacceptable during the development of an application, when the usefulness of any preparation operation has not been established yet.

Many boolean features can be identified for this criterion based on the metadata inference information in the operator specifications. However, in the tools examined here they are not distinctive. Therefore this criterion is boolean, and is fulfilled if a tool offers any inference or guessing of data characteristics.

25 Checking wellformedness: The software must differentiate between syntactically valid and invalid preparation graphs, and support the user in finding reasons for invalidity. Invalid graphs can result from, for example, deleting attributes at one point which are needed at another point, or by changing data characteristics, through recognition or manually, which some operator's specification depends on. The boolean features are ($m = 4$):

- indication of invalid nodes in the graphical representation of the processing graph;
- indication of ill-formed derivation formulas;
- indication of well-formed derivation formulas that use non-existing attributes;
- clear error messages to hint at the reasons for invalidity.

26 Empty data sets recognition: Sometimes operators produce concepts that have an empty extension. This can happen after an inner join by key or a row selection. Not all

tools recognise this but it can be the source for errors. An error message should be given when this happens. This criterion is boolean.

27 Metadata propagation: This is one of the most important criteria for complex applications. In complex preparation graphs, many dependencies exist between attributes and concepts at different locations in the graph. A simple example is a derivation of an attribute early in the data flow; this attribute is available in every following step. If the user decides to rename the derived attribute, the new name must be propagated through the graph. Similarly, if the step deriving that attribute is deleted from the graph, all later steps and concepts must be adjusted. Some steps may become invalid in the process; this should be displayed. These adjustments must be done automatically, as they may be rather complex.

While such propagation of metadata through the graph should be as robust as possible, it must not destroy invalid metadata. For example, if the deleted attribute is used as an input to a complex derivation of another attribute, the formula for derivation must not be deleted but kept in an invalid state, as the user might wish to modify the formula to a different input attribute, for example.

The importance of this criterion, as well as that of criterion 24, was also independently recognised by AlSairafi et al. (2003).

Propagation concerns attributes and concepts here; criterion 24 lists inferences that can be used to propagate data characteristics and types. The two operations that must be supported by propagation are addition and deletion, both for attributes and concepts, so there are four boolean features that a KDD tool must fulfil. A fifth feature checks the cautious deletion of dependent information, as explained above. Thus $m = 5$.

Support for KDD development

28 Relations between concepts: Intermediate results produced during data preparation are connected in various ways. A visualisation of these connections helps to organise the work and to keep an overview of complex preparation processes. It provides an alternative perspective of the process. The visualisation should be able to exclude or to highlight specific types of interconceptual relations if several types are used. However, since the differentiation of three relation types is suggested for KDD tools for the first time in this work, its implementation is not expected in any tool, thus this criterion is boolean: it checks whether a graphical representation of any relation between the concepts involved in a processing graph is available in a tool.

29 Display intermediate results: The input to a KDD process is a number of tables. In a given line of processing, one or more of these tables are changed. Every processing step produces a new concept that replaces the input concept conceptually, but of course in the tool all concepts must be available at all times. To enable the user to view the *current* set of tables after a given processing step, there should be an option to display only this set. This criterion is boolean.

30 Operator transparency: The reason for using convenience operators is to save the work of detailed specifications of primitive operators. For example, using a discretisation operator whose input is simply the number of intervals spares the user the computation of suitable interval boundaries, because the operator does this automatically. Nevertheless, the results of such automatic specifications must be inspectable and manipulable by the user. Besides giving more control to the user, this is also very important considering that some transformations have to be reversed for deployment, which is only possible for the user if all details of the transformation are accessible (but see criterion 52). So $m = 2$ (inspectability and changeability of derivation or selection formulas that are set up by the tool rather than the user).

31 Availability of primitive operators: All primitive operators must be available in the tool, as the lack of some of these operators results in uncomputable data representations. An exception is the operator ROW DERIVATION, which in its primitive form is not useful for KDD. This criterion could be extended to use all special options of the primitive operators, but this would result in a high value of m and the criterion would subsume too many details. Instead, section 1.6 provides a detailed table about presence or absence of the special options of each operator in every tool examined for this work. Thus $m = 6$.

32 Availability of convenience operators: All convenience operators should be available in the tool. Further convenience operators can be added as well as special operators for specific types of data, like text collections or time series. Convenience operators can make detailed specifications of primitive operators unnecessary, often replacing more than one primitive operator, resulting in quicker development and simpler preparation graphs. Again, this criterion might be extended to include all special options of operators; instead, this level of detail is left to a separate comparison table in section 1.6 (compare the previous criterion 31). So $m = 10$.

33 Attribute derivation support: For attribute derivation, it must be possible to set up any formula, using basic functions. The availability and meaning of these functions must be displayed to the user during set-up of a formula. If such features are lacking, the user cannot know which functions are available and what they compute, leading to frustrating trial-and-error procedures to arrive at correct formulas. So $m = 2$ for the availability of a choice list of provided functions, and for their documentation in the interface where the formula is set up.

34 Iteration of attribute derivation: The operator ATTRIBUTE DERIVATION must be configurable to derive more than one attribute based on the same formula, using automatically a specified change in the derivation formula for each derived attribute. For example, given an attribute that contains the months of a year, one new attribute for each month might be created that contains derived values of another attribute. The KDD tool should offer to set up the formula once, with a variable that iterates over the values of the month attribute. This iteration process should create as many attributes as there are values in the month attribute. But the derivation might also iterate over several input attributes, rather than the values of one attribute, or over values outside

the data, for example an increasing counter. If an own attribute derivation operator for each new attribute had to be used instead, this would require much more work to set up the operators, and the graph structure would become unnecessarily complex. Thus $m = 3$:

- possibility to use iteration over attribute values of a given attribute, to achieve "parallel" derivation of several attributes;
- possibility to use iteration over attribute names of a given concept, for the same purpose;
- possibility to use iteration over a given value list, again for the same purpose.

35 Data independence: An operator chain is declarative, thus it ought to be possible to set it up in the absence of input data. This is required, for example, when the data has not been processed far enough yet, so that metadata inference (criterion 24) is not possible. Another scenario is grid-based data mining, in which the allocation of computational resources is independent from the declaration of the KDD process (compare (AlSairafi et al., 2003), where this criterion is also discussed). Though many operator specifications depend on metadata (see criterion 27), it was also stressed in criterion 19 that it must be possible to provide metadata by hand. This criterion is boolean.

36 Representation of data flow: The interdependences of operator instances can become rather complex in big applications, so that they must be graphically displayed. If this feature is lacking, the user has to rely on intermediate data set names to understand the connection between steps. This criterion is boolean.

37 Support for chunking: It contributes to keeping an overview of complex preparation graphs if they can be partitioned into chunks. The structure of chunks should be most flexible. More precisely, $m = 2$:

- chunks must not be restricted to atmost one input and one output concept;
- chunks must be nestable into hierarchies.

38 Graph structure: The preparation graph is, in general, a directed acyclic graph (DAG) without further restrictions. One tool evaluated for this work imposes the restriction that there can be no two different paths from a given operator to a second one. Yet such a situation is rather common. This criterion is boolean, and is fulfilled if the DAG is unrestricted.

39 Execution transparency: When a data preparation graph is executed, progress should be clearly indicated to the user. This includes ($m = 7$):

- displaying information which step is currently being executed;
- displaying the number of data rows already processed in this step;
- displaying the number of data rows yet to process in this step;
- displaying the storage space consumed for the current execution;

- displaying the storage space expected to be required for the current execution;
- displaying the time the execution has consumed so far;
- displaying an estimation of the total execution time required.

40 Execution automation: An automatic execution of preparation graphs must be possible, to automate long-time consecutive or conceptually parallel experiments. More precisely, there are three aspects to be considered ($m = 3$):

- scheduling of execution runs to particular points in time;
- the option to automatically change parameters of the processing graph for each execution, so that the same graph can be run on batches of data sets, or with some specific parameter looping through its range for each execution;
- the possibility to specify the order of execution for different parts of the graph.

41 Execution administration: An execution run of a KDD process is an experiment. Information pertaining to this experiment must be automatically stored. This helps to organise the user's work when a lot of experiments are run, or when the execution times exceed the user's memory span. In particular ($m = 7$):

- information which steps were executed in an experiment must be stored;
- the number of rows in input and output must be stored;
- start and end time and date must be stored;
- the names of any involved files or database tables must be stored;
- each experiment must be given a unique ID, which must be stored;
- each experiment must be commentable with free text;
- this information about stored experiments must be searchable.

42 Execution in background: Editing parts of the processing graph must be possible during an execution of the graph. That is, the execution should run in the background, without blocking the system. Yet edits should not pertain to the running execution. This criterion is boolean.

43 Export transparency: Obviously, a way of storing and reloading the data preparation graph with all its parameters is needed. A particular point is that the storage format should be transparent, preferably based on an XML formalism. This gives extra flexibility to the user for complex ways of editing the graphs which are not foreseen by the graphical user interface. But more importantly, it makes the graphs at least readable when the KDD tool that produced them is no longer available, making old applications usable to some degree even when the computing environment changes. This criterion is boolean (transparent storage format).

44 Editing flexibility: On each level of the KDD model (data types, parameters, operators, and chunks), copy and paste functions must be provided. All KDD tools examined in this work offer this. However, flexible editing must also be possible for formulas in attribute derivation or row selection, especially if more than one attribute is going to be

derived in unsystematic ways not supported by the derivation operator (compare criterion 34). Some tools examined for this work lacked this option, resulting in tedious extra work in the situation of deriving many attributes. This criterion is boolean (availability of copy and paste functions for all formulas).

45 Visual graph arrangement: As some applications require complex processing graphs, the KDD software should be able to automatically arrange the nodes of the graph, the operators, on the screen in a clear fashion, for example on a grid. This criterion is boolean.

Mining and deployment phase

As explained in section 1.3.1, the criteria for the mining phase in this work concern typical processing and control tasks.

46 Splitting training and test set: A facility to randomly split a data table into a training and a test set, according to a given ratio, must be available. This can be realised by the operator SEGMENTATION. This criterion is boolean.

47 Model evaluation: A facility to evaluate the performance of any model learned in the tool on a test set must be available, using typical performance measures. Such measures are not listed here because they depend on the type of mining task (examples are accuracy, support, intra-cluster density etc.). But at least one application-independent performance measure must be offered for every type of model. This criterion is boolean.

48 Cross validation: To minimise biases on the evaluation introduced by a particular split into training and test set, the process split-learn-evaluate is usually executed several times in machine learning, taking the average of the evaluations as an indication of the model performance. This is called cross validation. It should be supported by offering an operator whose input is a data set, a type of model to be learned, a performance measure, and the number of times the process is to be executed. The availability of this operator is the boolean criterion.

49 Automatic parameter tuning: Most machine learning algorithms use a number of parameters whose settings influence their learning behaviour. A facility to automatically test a specified range of each parameter, using the mining performance (criterion 47) to find the best value for the parameter, must be offered. This involves again the repeated execution of the subprocess split-learn-evaluate, but with at least one changed parameter for every execution. This criterion is related to one feature of criterion 40, but focuses on mining parameters. It is a boolean criterion.

50 Nestability of control operators: More complex machine learning experiments require the nesting of the above control operators. This criterion is inspired by the YALE system (Mierswa et al., 2003). For example, to make parameter tuning more robust, each parameter setting might be evaluated using its own cross validation. The nestability is the boolean criterion.

51 Export of models: For deployment in an actual technical or business process, functions that are learned by the tool must be exportable into source code, to be usable outside the tool in arbitrary environments. This criterion is boolean.

52 Post-processing: To enable the post-processing of data that was "encoded" for mining, the tool should offer an automatically created operator for any reversible transformation that was applied during data preparation. This automatically created operator can be applied to the predictions of a model and reverses the transformation, in order to get predictions from the original domain of the label attribute. This criterion is boolean.

Support for KDD standards

53 CRISP support: The software should support the distinction between the different phases of a KDD process, for example by providing different graphical environments for data understanding (visualisation, characteristics computation), data preparation, mining and deployment. This criterion is boolean.

54 PMML support: Models learned by the software should be exportable to files using the PMML standard (Grossman et al., 2002). Conversely, PMML files should be importable and applicable. See also criterion 22. So $m = 2$ for import and export.

There are other standards around KDD, see (Grossman et al., 2002), but they are more oriented towards KDD developers. From the conceptual point of view of a user, the two standards above are the most relevant ones.

1.4. A test case to check all criteria

In this section a test case is provided that is as small as possible but still enables to check all criteria from section 1.3.3, given a KDD tool. The test case describes a small KDD process. Its implementation is described step by step, with reference to every criterion that is tested in each step. The case can be seen as a baseline scenario whose implementation is possible using only the primitive operators, but is much simpler the more convenience operators are available. It can be implemented in a few hours, thus giving an effective method to objectively evaluate a KDD tool in practice, under the criteria given here. It corresponds to a detailed evaluation plan as explained in section 1.2.3. Since every step of the test case concerns particular criteria which are given in the description below, some steps can be omitted if the criteria tested there are known to be less important for the particular evaluation purpose.

Appendix B (page 38) provides an SQL program that realises the test case. The comparison to figure 1.2, which shows the graphical representation of the case in the KDD tool MiningMart (see section 1.5.1), illustrates some benefits of the conceptual approach to KDD.

The order of steps in this test case is based on the data flow that is modelled, rather than on the order of the criteria. An interesting alternative would be to order the test case such that those criteria which appear to be most challenging are tested last. This would

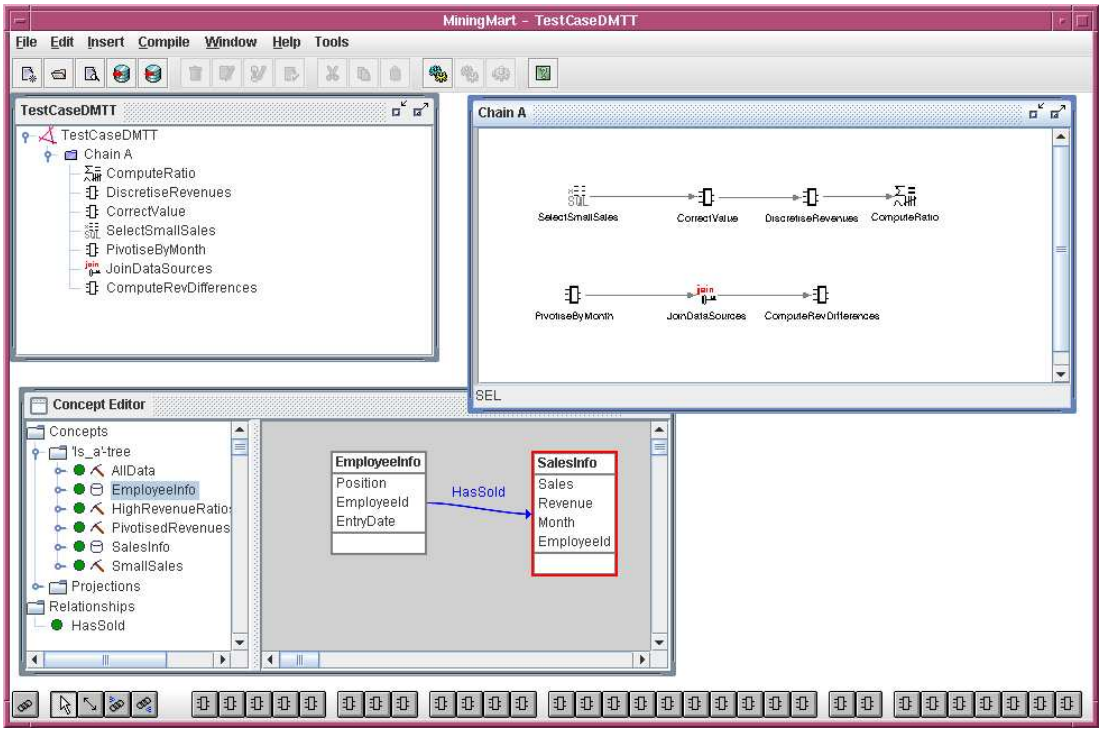

Figure 1.2.: A screenshot of the MiningMart GUI displaying the test case.

render a single score for each tool that is evaluated, namely the point in the test case at which it cannot support the tested functionality any longer. To capture the notion of difficulty, or how challenging a criterion is, the number of tools evaluated in this work that fulfill each criterion can be used for ordering the criteria. However, an implementation of the test case that follows the data flow is easier to describe, understand and realise.

The data

Figure 1.3 shows two small data sets which are the input to the test case. These data sets can be easily provided as flat files, with any file format that is deemed relevant, or as database tables. To test criterion 7, the data sets are imported into the tool (to test the output facilities, they can be written from the tool to different files/tables as well). At this point, already a number of other criteria can be checked. If the data sets can be displayed inside the tool, criterion 13 is fulfilled. Criterion 14 lists recommended facilities for attribute import. Also it can be seen whether conceptual data types are used in the tool (criterion 15), and if they are correctly recognised (criterion 16); for example, is the column `EntryDate` automatically given a correct technical and conceptual data type (e.g. *Date*)? Can the data types be changed? Can automatic recognition of types be deferred to a later point in time? The recognition of data characteristics can also be tested (criterion 19).

Table SalesData:

EmplId	Month	Sales	Profit
1	1	3	40.5
1	2	2	22.8
1	3	-1	10.0
2	1	5	54.2
2	2	7	58.6
2	3	4	41.0
3	1	-1	10.0
3	2	2	38.1
3	3	4	44.3

Table EmployeeData:

EmplId	EntryDate	Position
1	02-12-1988	Senior
2	01-06-1998	Trainee
3	01-01-1990	Senior

Figure 1.3.: Input data for the test case.

A different approach is to test criterion 35 by attempting to set up data models (concepts) without actually importing data.

Data preparation

After these preliminaries, the first processing step is a ROW SELECTION, applied to SalesData. Its output concept contains only the rows with `Sales` ≤ 5. Are the data characteristics of the output concept available without executing the operator, in particular, is the highest value of `Sales` adjusted to 5 (criterion 24)? Changing the selection condition to `Sales` > 10 can be used to test criterion 26 (empty data set recognition) after executing the operator.

The next operator corrects the values of `Sales` by replacing all occurrences of -1 (taken to be missing values or typos) with 0. The convenience operator VALUE MAPPING should be available, but if it is not, ATTRIBUTE DERIVATION can be used with an if-then-else type of derivation formula. The values of the `Sales` attribute must be available in the graphical user interface when specifying the parameters for VALUE MAPPING (criterion 20), preferably without having executed the previous operator (criterion 24).

The next operator discretises the `Profit` attribute of SalesData into two categorical values. The convenience operator DISCRETISATION (using a given number of intervals) should be available; otherwise ATTRIBUTE DERIVATION must be used. Criterion 30 tests whether the discretisation formula used by DISCRETISATION is accessible and changeable (after execution of the operator). The two categorical values created by DISCRETISATION should also be changeable: if yes, they are set to 0 and 1 now, if no, an extra VALUE MAPPING step is inserted to do so.

The fourth operator computes the ratio of the higher of the two intervals formed in the previous step. This is realised by AGGREGATION, where the *Group By* attribute is `EmplId`, and the average function is used as aggregation function, applied to the discretised attribute. This tests criterion 18 (robustness of type mapping), as the categorical values 0 and 1 created in the previous step are used as real numbers here. If this usage (and thus the criterion) fails, criterion 17 can be tested by attempting to explicitly convert the data type.

These four operators can be executed in the order given here, as each operator's input is produced by the previous operator. The four operators could be collected in a chunk, testing criterion 37 (chunking support). Criteria 39, 40, 41, and 42 (the execution-related criteria) can be tested by executing the four operators together. It can be attempted to vary the place of processing (criterion 8, data handling). Further, the caching-related criteria 9, 10, 11 and 12 can be tested, for example by checking whether the result of the execution is still available afterwards (11, automatic caching), or by trying to find out where the intermediate data was stored (12, caching transparency).

With this short chain, also the important criterion 27 about the propagation of conceptual changes can be tested. To this end, the second step is deleted. Is the attribute it creates (with the corrected values of the `Sales` attribute) automatically removed from the input and output of the following steps? If the step is added again, does the attribute show up in later steps automatically?

For further tests, a new short chain of operators is set up. The data from the SalesData table contains monthly information about the sales and profit that every employee achieved. This data is converted to single-row information about each employee by a PIVOTISATION application, where the index attribute is `Month`, the pivotisation attribute is `Profit`, the aggregation operator is summation and the *Group By* attribute is `EmplId`. If this convenience operator is missing, three ATTRIBUTE DERIVATIONs can be used, one for each month, followed by an AGGREGATION. By using the ATTRIBUTE DERIVATIONs, criterion 44 can be tested, as the three derivations are very similar. For example, in SQL, the three derivations would be `(CASE WHEN Month=`*i*` THEN Profit ELSE 0 END)`, with *i* ranging from 1 to 3, resulting in three new attributes which would be aggregated using summation.

The result of the previous step is now joined with the result of the first chunk (chain), using JOIN BY KEY and testing criterion 22 (attribute matching) when setting up this operator. The key for joining is of course the attribute `EmplId`; can its role as a key be stated explicitly (criterion 21 about attribute roles)? At this point, criterion 28 about the representation of relations between the concepts might also be tested, as the foreign key link between the two data sets should be represented.

To test criterion 34 about the iteratability of ATTRIBUTE DERIVATIONs, a final operator is added which computes the differences between the profit achieved in the third month and those achieved in the first and second month. The derivation formula should iterate over the two fields with the profit for the first and second month. A further test of criterion 27 about the propagation of changes can be done now, by deleting the pivotisation step(s). The formula for the difference in this last step should then not be automatically deleted; instead, the last step (or its derivation formula) should be clearly marked as invalid (criterion 25, checking wellformedness).

Criteria 43 about the transparency of export files, and 45 about the arrangement of operator applications in a processing graph, can be checked using the whole test case.

Criterion 38 about the unrestricted structure of the preparation graph is missing so far; it can be tested by applying two ROW SELECTIONs to the same input concept, and then applying UNION to the two results. If this is possible, it follows—together with the above—that the tool allows any directed acyclic graph.

Mining

Finally, the criteria related to mining and deployment, which are all boolean, can be tested. No particular scenario is needed to test them; in most cases, the help system or the manual will be sufficient to decide whether the criteria are fulfilled. For example, the availability of cross validation or model export facilities in a tool will surely be reflected in the documentation.

1.5. Evaluated KDD software

This section briefly describes the KDD software packages that were evaluated in this work. These tools were chosen according to their general strength in the conceptual support of data processing, and they serve well to exemplify different aspects of many criteria.

1.5.1. MiningMart

MiningMart[1] is introduced in (Morik & Scholz, 2004) as a graphical front-end to relational databases that offers a broad range of KDD-oriented data preparation operators. It leaves all processing to the underlying database system by translating the preparation graph to SQL commands. Such graphs, called *Cases* in MiningMart, can be exported and uploaded to a central web repository, where they are browsable and downloadable by anyone looking for example KDD applications. This is the only tool encountered during this work that uses an explicit representation of relations between intermediate processing results (concepts).

1.5.2. SPSS Clementine

Clementine[2] is a tool intended to support all phases of the KDD process. It includes many data preparation facilities. It was used for this examination in the standalone version, thus entirely file system based, but a client server version is also available that can delegate some data processing tasks to the database server. Version 8.1 was used in this work.

1.5.3. Prudsys Preminer

Preminer, sold by Prudsys[3], is a specialised tool for data preparation that belongs to a family of products supporting the complete KDD process. Its architecture is different from the preceding two tools in that it uses an extra data server for intermediate storage of data. This enables the user to process data from heterogeneous sources using the same front end. For example, a data set from a text file can be joined with a database table (if the keys match). The evaluation in this work was based on Version 1.3. For evaluating the mining facilities the Discoverer module version 3.2 was used.

[1] `http://mmart.cs.uni-dortmund.de`
[2] `http://www.spss.com/clementine`
[3] `http://www.prudsys.com`

1.5.4. IBM Intelligent Miner

Intelligent Miner by IBM[4] is a group of products to cover data preparation, mining and deployment based on IBM's DB2 database. The graphical front-end is the Intelligent Miner for Data, whose version 8.1 was used for the evaluation in this work. While both flat file data and database tables can be input to mining, the data preparation operators can only be applied to database tables, as they are realised by SQL views, in a way similar to MiningMart (1.5.1). Also, learned models are available as DB2 procedures, which leverages their deployment on large data sets.

1.5.5. SAS Enterprise Miner

The Enterprise Miner is one of several analysis modules available in the SAS system[5]. The SAS environment is a powerful workbench for many aspects of data analysis. It offers client-server processing distribution as well as data warehousing support. The Enterprise Miner provides several mining algorithms and many data inspection facilities, though the latter can be complemented by other SAS modules. The Enterprise Miner includes some data preparation functionality, but its focus is on the mining step and on visualisations of data sets and mining results. Therefore it lacks many of the essential primitive operators. They can be replaced by integrating small programs in the internal SAS language. However, as explained in section 1.2.1, such programming constructs do not support conceptual, high-level work, and the functionality they may offer is not seen as fulfilling any criterion. Version 4.3 of the Enterprise Miner was used in this evaluation.

1.5.6. NCR Teradata Warehouse Miner

The Warehouse Miner[6] by Teradata, a division of NCR, is a tool specifically developed to support mining Teradata databases. Apart from an ODBC interface, it can only be used on Teradata databases, from a Windows client. It leaves as much data processing as possible to the underlying database, issuing automatically created SQL statements in a way similar to MiningMart and the Intelligent Miner. It offers a number of convenience operators for processing, but relies heavily on SQL programming as far as the primitive operators are concerned (in which it resembles the Enterprise Miner by SAS). It does not use an explicit data model, nor does it display the data flow in a graph. Version 3.2 was used in this evaluation.

1.6. Evaluation results

This section provides the classification of the tools described in 1.5 following the criteria from 1.3.3. As explained in section 1.2.2, the list of criteria is amenable to several methods of scoring and weighting. In the evaluation in this section, each tool receives the score (measure) $0 \leq n/m \leq 1$, where $m > 0$ is the number of boolean features that make up a criterion, and $0 \leq n \leq m$ is the number of these features that the tools fulfils. Thus the m

[4] `http://www.ibm.com/software/data/iminer`
[5] `http://www.sas.com`
[6] `http://www.teradata.com`

No	Name	m	MM	Clem.	Prem.	IBM	SAS	NCR
1	Data Handling	14	0.43	0.79	0.57	0.43	0.64	0.36
2	Caching	7	0.71	0.43	0.43	0.71	0.43	0.71
3	Data Modelling	26	0.77	0.85	0.35	0.58	0.65	0.3
4	Process Modelling	22	0.68	0.77	0.64	0.32	0.64	0.18
5	Execution	18	0.39	0.28	0.5	0.28	0.39	0.11
6	Operators	22	0.73	0.63	0.32	0.36	0.36	0.45
7	Mining/Deployment	8	0.25	0.63	0.5	0.5	0.5	0.38

Table 1.2.: A different representation of the data in table 1.4 using larger groupings of the 117 boolean features into criteria. The groupings are shown in table 1.3.

boolean features of a criterion are not weighted (prioritised) here, as a weighting would be very dependent on the intended application and environment for the KDD tools.

Table 1.4 contains the scores for each criterion based on the list of criteria from section 1.3.3. Table 1.2 provides scores which are computed based on the same list of 117 boolean features, but a different grouping into criteria, namely into fewer criteria using higher values of m. As section 1.2.2 explains, these alternative scores are a different representation of the same data that may be more suitable for certain audiences, for example for decision makers. The criteria in table 1.2 use a different grouping into overview criteria than was indicated by section headlines in section 1.3.3 (and by horizontal lines in table 1.4). Instead, table 1.3 shows how the criteria in tables 1.2 and 1.4 are related. The fact that two rather different but meaningful groupings into overview criteria are possible illustrates that any tool evaluation methodology must be flexible and adaptable to varying circumstances or user priorities, and demonstrates that the proposed methodology achieves this.

A further table (table 1.5 on page 35) contains a detailed list of the preparation operators, that are available in each tool[7]. To illustrate the effect of the availability of many

[7]The operator ROW DERIVATION is excluded.

No	Name	Includes criteria from table 1.4	m
1	Data Handling	7, 8, 13, 14, 26	14
2	Caching	9, 10, 11, 12	7
3	Data Modelling	15–22, 28, 29, 35	26
4	Process Modelling	23–25, 27, 33, 36–38, 43, 45, 51, 53	22
5	Execution	39–42	18
6	Operators	30–32, 34, 44	22
7	Mining/Deployment	46–50, 52, 54	8

Table 1.3.: Grouping of criteria of table 1.4 into overview criteria shown in table 1.2.

convenience operators, note that the test case described in section 1.4 required—without mining—7 operator applications in MiningMart and the Teradata Warehouse Miner, 11 in Clementine, and 9 in Preminer and Intelligent Miner, respectively. In SAS Enterprise Miner the test case was only partially implemented, as this tool lacks the JOIN BY KEY operator.

No	Name	m	MM	Clem.	Prem.	IBM	SAS	NCR
7	Data formats	6	0.33	0.83	0.5	0.5	0.66	0.33
8	Data handling	3	0.33	1.0	0.66	0.66	0.66	0.33
9	Caching control	2	1.0	1.0	0.5	1.0	0	1.0
10	Caching size estimation	1	0	0	0	0	0	0
11	Automatic caching	2	0.5	0	0	0.5	0.5	0.5
12	Caching transparency	2	1.0	0.5	1.0	1.0	1.0	1.0
13	Data inspection	1	1.0	1.0	1.0	0	1.0	1.0
14	Attribute import	3	0.66	0.66	0.66	0.33	0.33	0.33
15	Conceptual data types	1	1.0	1.0	0	1.0	1.0	0
16	Type recognition	5	0.8	1.0	0.4	0.8	0.8	0.2
17	Flexibility of type mapping	3	1.0	1.0	0	0.66	1.0	0
18	Robustness of type mapping	1	1.0	1.0	1.0	1.0	0	1.0
19	Data char. recognition	6	0.66	0.66	0	0.66	0.66	0.33
20	Data char. deployment	1	0	1.0	1.0	0	1.0	0
21	Attribute roles	4	0.75	1.0	0.75	0.5	1.0	0.75
22	Attribute matching	2	1.0	1.0	1.0	0.5	0	0.5
23	Type inference	2	1.0	1.0	0.5	0.5	1.0	0
24	Metadata inference	1	1.0	0	0	0	0	0
25	Checking wellformedness	4	0.25	0.5	0.75	0	0.25	0.25
26	Empty data sets recognition	1	0	0	0	0	1.0	0
27	Metadata propagation	5	1.0	1.0	1.0	0.2	0.8	0.2
28	Relations between concepts	1	1.0	0	0	0	0	0
29	Display intermediate results	1	1.0	0	0	0	0	0
30	Operator transparency	2	0.5	0	0	0	1.0	0.5
31	Availability prim. operators	6	1.0	1.0	1.0	0.83	0.33	0.33
32	Availability conv. operators	10	0.8	0.5	0.1	0.2	0.3	0.6
33	Attribute derivation support	2	0	1.0	1.0	0.5	0.5	0
34	Iteration attribute derivation	3	0	0.33	0	0	0	0
35	Data independence	1	1.0	1.0	0	0	0	0
36	Representation of data flow	1	1.0	1.0	1.0	0	1.0	0
37	Support for chunking	2	1.0	0.5	0	1.0	1.0	0
38	Graph structure	1	1.0	1.0	0	1.0	1.0	1.0
39	Execution transparency	7	0.29	0.14	0.43	0.29	0.43	0.14
40	Execution automation	3	0	0.33	0.66	0	0	0
41	Execution administration	7	0.57	0.29	0.43	0.29	0.57	0
42	Execution in background	1	1.0	1.0	1.0	1.0	0	1.0
43	Export transparency	1	1.0	0	1.0	0	1.0	1.0
44	Editing flexibility	1	1.0	1.0	0	1.0	0	1.0
45	Visual graph arrangement	1	0	1.0	1.0	0	0	0
46	Splitting training and test set	1	1.0	1.0	1.0	1.0	1.0	1.0
47	Model evaluation	1	0	1.0	1.0	1.0	1.0	1.0
48	Cross validation	1	0	0	0	0	0	0
49	Automatic parameter tuning	1	0	0	0	0	0	0
50	Nestability of control operators	1	0	0	0	0	0	0
51	Export of models	1	1.0	1.0	0	1.0	1.0	0
52	Post-processing	1	0	0	0	0	0	0
53	CRISP support	1	0	1.0	0	0	0	0
54	PMML support	2	0	1.0	1.0	0.5	0.5	0.5

Table 1.4.: Evaluation table. $m = 1$ indicates boolean criteria.

Operator	Type	MM	Clem.	Prem.	IBM	SAS	NCR
Attribute derivation	P						
– String processing			Yes	Yes		Yes	
– Numeric arithmetics		Yes	Yes	Yes	Yes	Yes	Yes
– Date/time arithmetics		Yes	Yes	Yes	Yes	Yes	Yes
– Model application		Yes	Yes	Yes	Yes	Yes	Yes
– Value generation		PK					
– Principal components			Yes		Yes	Yes	Yes
Attribute selection	P						
– Manual selection		Yes	Yes	Yes	Yes	Yes	
– Automatic selection		Yes				Yes	
Row selection	P						
– Arbitrary filter		Yes	Yes	Yes	Yes		
– Sampling		Yes	Yes	Yes	Yes	Yes	Yes
Join by Key	P						
– Inner		Yes	Yes	Yes	Yes		Yes
– Outer			Yes	Yes			
Union	P	Yes	Yes	Yes			
Aggregation	P	Yes	Yes	Yes	Yes		
Discretisation	C						
– fixed no of intervals		Yes	Yes		Yes	Yes	Yes
– fixed width		Yes	Yes				Yes
– fixed cardinality		Yes	Yes				Yes
Scaling	C	Yes					Yes
Value mapping	C	Yes	Yes	Yes	Yes		Yes
Missing value replacement	C						
– By default value		Yes	Yes		Yes	Yes	
– By average or median		Yes	Yes			Yes	
– By learned function		Yes				Yes	
Filtering outliers	C					Yes	
Segmentation	C						
– By value		Yes					
– Randomly		Yes					Yes
– By learned clusters		Yes					
Dichotomisation	C	Yes	Yes				Yes
Pivotisation	C						
– normal		Yes					Yes
– n-fold		Yes					
– reverse					Yes		
Data generation	C		Yes				
Windowing	C	Yes					

Table 1.5.: Availability of preparation operators for each KDD tool. No entry = not available. Types are primitive (P) and convenience (C). PK = Primary key generation. Note that any convenience operator can be replaced by one or more primitives, requiring more user efforts and resulting in more complex processes.

Appendix A: List of Criteria

No	Name	Page
1	Adaptability	13
2	Scalability	14
3	Interoperability	14
4	Guide to KDD process	14
5	Documentation	14
6	Business problem	14
7	Data formats	15
8	Data handling	17
9	Caching control	17
10	Caching size estimation	17
11	Automatic caching	17
12	Caching transparency	18
13	Data inspection	18
14	Attribute import	18
15	Conceptual data types	18
16	Type recognition	18
17	Flexibility of type mapping	19
18	Robustness of type mapping	19
19	Data characteristics recognition	19
20	Data characteristics deployment	19
21	Attribute roles	19
22	Attribute matching	19
23	Data type inference	20
24	Metadata inference	20
25	Checking wellformedness	20
26	Empty data sets recognition	20
27	Metadata propagation	21
28	Relations between concepts	21
29	Display intermediate results	21
30	Operator transparency	21

Table A.1.: Overview of the criteria, part 1.

No	Name	Page
31	Availability of primitive operators	22
32	Availability of convenience operators	22
33	Attribute derivation support	22
34	Iteration of attribute derivation	22
35	Data independence	23
36	Representation of data flow	23
37	Support for chunking	23
38	Graph structure	23
39	Execution transparency	23
40	Execution automation	24
41	Execution administration	24
42	Execution in background	24
43	Export transparency	24
44	Editing flexibility	24
45	Visual graph arrangement	25
46	Splitting training and test set	25
47	Model evaluation	25
48	Cross validation	25
49	Automatic parameter tuning	25
50	Nestability of control operators	25
51	Export of models	26
52	Post-processing	26
53	CRISP support	26
54	PMML support	26

Table A.2.: Overview of the criteria, part 2.

Appendix B: SQL Implementation of Test Case

This appendix lists an SQL program that realises the test case described in section 1.4. The program can serve as a reference for an evaluation of a KDD tool, or for the functionality employed by the convenience operators involved in the test.

```
-- This SQL script creates two small tables and realises
-- some data preparation operations on them.
-- Author: Timm Euler, University of Dortmund (April 2005)

-- create tables:

DROP TABLE Saleinfo;
CREATE TABLE Saleinfo
  ( Employee NUMBER(2),
    Month NUMBER(2),
    Sales NUMBER(4),
    Revenue NUMBER
  );

DROP TABLE Employeeinfo;
CREATE TABLE Employeeinfo
  ( Employee NUMBER(2),
    Entry    DATE,
    Position VARCHAR2(10),
    CONSTRAINT EmployeePk PRIMARY KEY (Employee)
  );

-- insert some values:

DELETE FROM Saleinfo;
INSERT INTO Saleinfo VALUES (1, 1,  3, 40.5);
INSERT INTO Saleinfo VALUES (1, 2,  2, 22.8);
INSERT INTO Saleinfo VALUES (1, 3, -1, 10.0);
INSERT INTO Saleinfo VALUES (2, 1,  5, 54.2);
INSERT INTO Saleinfo VALUES (2, 2,  7, 58.6);
INSERT INTO Saleinfo VALUES (2, 3,  4, 41.0);
INSERT INTO Saleinfo VALUES (3, 1, -1, 10.0);
```

```
INSERT INTO Saleinfo VALUES (3, 2,  2, 38.1);
INSERT INTO Saleinfo VALUES (3, 3,  4, 44.3);

DELETE FROM Employeeinfo;
INSERT INTO Employeeinfo VALUES (1, to_date('02-12-1988','DD-MM-YYYY'),
                                 'Senior');
INSERT INTO Employeeinfo VALUES (2, to_date('01-06-1998','DD-MM-YYYY'),
                                 'Trainee');
INSERT INTO Employeeinfo VALUES (3, to_date('01-01-1990','DD-MM-YYYY'),
                                 'Senior');

-- chain A:

-- step A1: select rows with Sales < 5

CREATE OR REPLACE VIEW Smallsales AS
    (SELECT * FROM Saleinfo WHERE Sales < 5);

-- step A2: map -1 to 0 for the Sales column

CREATE OR REPLACE VIEW Smallsales_Corrected AS
    (SELECT Employee,
            Month,
            (CASE WHEN Sales = -1 THEN 0 ELSE Sales END) AS Sales,
            Revenue
     FROM Smallsales
    );

-- step A3: discretise Revenue column into 2 bins

CREATE OR REPLACE VIEW Binned_Revenue AS
    (SELECT Employee,
            Month,
            (CASE WHEN Revenue < 40 THEN 0 ELSE 1 END) AS Bin
     FROM Smallsales_Corrected
    );

-- step A4: compute ratio of high revenue months per employee

CREATE OR REPLACE VIEW High_Revenues_Ratio AS
    (SELECT Employee,
            AVG(Bin) AS Ratio
     FROM Binned_Revenue
     GROUP BY Employee
    );
```

```
-- chain B:

-- step B1: pivotise revenues (create 3 new columns,
            one per month)

CREATE OR REPLACE VIEW Pivotised_Data AS
    (SELECT Employee,
            SUM(CASE WHEN Month = 1 THEN Revenue ELSE 0 END)
                AS Revenue_Month1,
            SUM(CASE WHEN Month = 2 THEN Revenue ELSE 0 END)
                AS Revenue_Month2,
            SUM(CASE WHEN Month = 3 THEN Revenue ELSE 0 END)
                AS Revenue_Month3
     FROM Saleinfo
     GROUP BY employee
    );

-- step B2: join

CREATE OR REPLACE VIEW Alldata AS
    (SELECT Employeeinfo.Employee,
            Revenue_Month1,
            Revenue_Month2,
            Revenue_Month3,
            Entry,
            Position
     FROM Pivotised_Data, Employeeinfo
     WHERE Pivotised_Data.Employee = Employeeinfo.Employee
    );

-- step B3: compute changes in the revenues per month

CREATE OR REPLACE VIEW Revenue_Changes AS
    (SELECT Employee,
            (Revenue_Month3 - Revenue_Month1) AS DiffM3M1,
            (Revenue_Month3 - Revenue_Month2) AS DiffM3M2,
            Revenue_Month1,
            Revenue_Month2,
            Revenue_Month3,
            Entry,
            Position
     FROM Alldata
    );
```

Bibliography

Abbott, D. W., Matkovsky, I. P., & Elder IV, J. F. (1998). An Evaluation of High-end Data Mining Tools for Fraud Detection. *IEEE International Conference on Systems, Man, and Cybernetics*. San Diego, CA.

AlSairafi, S., Emmanouil, F.-S., Ghanem, M., Giannadakis, N., Guo, Y., Kalaitzopoulos, D., Osmond, M., Rowe, A., Syed, J., & Wendel, P. (2003). The Design of Discovery Net: Towards Open Grid Services for Knowledge Discovery. *High-Performance Computing Applications*, *17*, 297–315.

April, A. A., & Al-Shurougi, D. (2000). Software Product Measurement for Supplier Evaluation. *Proceedings of the FESMA-AEMES Software Measurement Conference*. Madrid, Spain.

Barbacci, M., Klein, M. H., Longstaff, T. A., & Weinstock, C. B. (1995). *Quality Attributes* (Technical Report CMU/SEI-95-TR-021). Software Engineering Institute.

Boehm, B. W., Brown, J., Kaspar, H., Lipow, M., McLeod, G., & Merritt, M. (1978). *Characteristics of Software Quality*. Amsterdam: North-Holland.

Botella, P., Illa, X. B., Carvallo, J. P., Franch, X., & Quer, C. (2002). Using Quality Models for Assessing COTS Selection. *Anais do WER02 - Workshop em Engenharia de Requisitos* (pp. 263–277). Valencia, Spain.

Brown, A. W., & Wallnau, K. C. (1996). A Framework for Systematic Evaluation of Software Technologies. *IEEE Software*, *13*, 39–49.

Cannataro, M., & Comito, C. (2003). A Data Mining Ontology for Grid Programming. *1st Workshop on Semantics in Peer-to-Peer and Grid Computing at the 12th Int. World Wide Web Conference*. Budapest, Hungary.

Cartwright, M., & Shepperd, M. (2000). An Empirical Investigation of an Object-Oriented Software System. *IEEE Transactions on Software Engineering*, *26*, 786–796.

Carvallo, J. P., Franch, X., Grau, G., & Quer, C. (2004a). On the Use of Quality Models for COTS Evaluation. *Proceedings of the International Workshop on Models and Processes for the Evaluation of COTS Components (MPEC)*. Edinburgh, UK.

Carvallo, J. P., Franch, X., Grau, G., & Quer, C. (2004b). QM: A Tool for Building Software Quality Models. *Proceedings of the 12th IEEE International Conference on Requirements Engineering (RE 2004)* (pp. 358–359). Kyoto, Japan: IEEE Computer Society.

Cavano, J. P., & McCall, J. A. (1978). A Framework for the Measurement of Software Quality. *Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues* (pp. 133–139).

Collier, K. W., Sautter, D., Marjaniemi, C., & Carey, B. (1999). A Methodology for Evaluating and Selecting Data Mining Software. *Proceedings of the 32nd Hawaii Int. Conference on System Sciences.*

Colombo, R., & Guerra, A. (2002). The Evaluation Method for Software Products. *Proceedings of the International Conference on Software and Systems Engineering and Their Applications (ICSSEA).* Paris, France.

Euler, T. (2005a). An Adaptable Software Product Evaluation Metric. *Proceedings of the 9th IASTED International Conference on Software Engineering and Applications (SEA).* Phoenix, Arizona, USA.

Euler, T. (2005b). Modelling Data Mining Processes on a Conceptual Level. *Proceedings of the 5th International Conference on Decision Support for Telecommunications and Information Society.* Warsaw, Poland.

Gaul, W., & Säuberlich, F. (1999). Classification and Positioning of Data Mining Tools. *Classification in the Information Age* (pp. 145–154). Springer.

Gentsch, P., Niemann, C., & Roth, M. (2000). *Data Mining: 12 Software-Lösungen im Vergleich.* Oxygon-Verlag. In German.

Gimbel, M., Klein, M., & Lockemann, P. C. (2004). Interactivity, Scalability and Resource Control for Efficient KDD Support in DBMS. In R. Meo, P. L. Lanzi and M. Klemettinen (Eds.), *Database Support for Data Mining Applications (LNAI 2682)*, 174–193. Berlin, Heidelberg: Springer.

Giraud-Carrier, C. G., & Povel, O. (2003). Characterising Data Mining Software. *Intelligent Data Analysis, 7*, 181–192.

Goebel, M., & Gruenwald, L. (1999). A Survey of Data Mining and Knowledge Discovery Software Tools. *ACM SIGKDD Explorations, 1*, 20–33.

Grossman, R. L., Hornick, M. F., & Meyer, G. (2002). Data Mining Standards Initiatives. *Communications of the ACM, 45*, 59–61.

Kusters, R., van Solingen, R., & Trienekens, J. (1997). User-Perceptions of Embedded Software Quality. *Proceedings of the STEP97 Conference.* IEEE Computer Society Press.

Maiden, N. A., Ncube, C., & Moore, A. (1997). Lessons Learned During Requirements Acquisition for COTS Systems. *Communications of the ACM, 40*, 21–25.

Maier, T., & Reinartz, T. (2004). Evaluation of Web Usage Analysis Tools. *Künstliche Intelligenz, 1*, 65–68.

Mayrand, J., & Coallier, F. (1996). System Acquisition Based On Software Product Assessment. *Proceedings of the International Conference on Software Engineering (ICSE).*

Mierswa, I., Klinkenberg, R., Fischer, S., & Ritthoff, O. (2003). A Flexible Platform for Knowledge Discovery Experiments: YALE – Yet Another Learning Environment. *LLWA 03 - Tagungsband der GI-Workshop-Woche Lernen - Lehren - Wissen - Adaptivität.*

Morik, K., & Scholz, M. (2004). The MiningMart Approach to Knowledge Discovery in Databases. In N. Zhong and J. Liu (Eds.), *Intelligent Technologies for Information Analysis*, chapter 3, 47–65. Springer.

Musick, R., & Critchlow, T. (1999). Practical Lessons in Supporting Large-scale Computational Science. *ACM SIGMOD Record, 28*, 49–57.

Paulk, M. C., Weber, C. V., Curtis, B., & Chrissis, M. B. (1995). *The Capability Maturity Model: Guidelines for Improving the Software Process.* Addison-Wesley.

Punter, T. (1997). Using Checklists to Evaluate Software Product Quality. *Proceedings of the 8th European Software Control and Metrics Conference (ESCOM).* Berlin, Germany.

Punter, T., Kusters, R., Trienekens, J., Bemelmans, T., & Brombacher, A. (2004). The W-Process for Software Product Evaluation: A Method for Goal-Oriented Implementation of the ISO 14598 Standard. *Software Quality Journal, 12*, 137–158.

Punter, T., van Solingen, R., & Trienekens, J. (1997). Software Product Evaluation. *Proceedings of the 4th Conference on Evaluation of Information Technology (EVIT).* Delft, The Netherlands.

Rangarajan, K., Swaminathan, N., Hedge, V., & Jacob, J. (2001). Product Quality Framework: A Vehicle for Focusing on Product Quality Goals. *Software Engineering Notes, 26*, 77–82.

Riedemann, E. H. (1997). *Testmethoden für sequentielle und nebenläufige Software-Systeme.* Stuttgart: Teubner.

Romei, A., Ruggieri, S., & Turini, F. (To appear). KDDML: A Middleware Language and System for Knowledge Discovery in Databases. *Data and Knowledge Engineering*, -.

Sarawagi, S., Thomas, S., & Agrawal, R. (1998). Integrating Association Rule Mining with relational Database Systems: Alternatives and Implications. *Proceedings of the ACM SIGMOD, International Conference on Management of Data* (pp. 343–354).

Suryn, W., Abran, A., & April, A. (2003). ISO/IEC SQuaRE. The Second Generation of Standards for Software Product Quality. *Proceedings of the 7th IASTED International Conference on Software Engineering Applications.* Marina del Rey, CA, USA.