Extensibility in data mining systems

Stefan Wrobel and **Dietrich Wettschereck** and **Edgar Sommer** and **Werner Emde**

GMD, FIT.KI
(Institute of Applied Information Technology, Artificial Intelligence Research Division)
Schloß Birlinghoven, 53754 Sankt Augustin, Germany
E-Mail stefan.wrobel@gmd.de

Abstract

The successful application of data mining techniques ideally requires both system support for the entire knowledge discovery process *and* the right analysis algorithms for the particular task at hand. While there are a number of successful data mining systems that support the entire mining process, they usually are limited to a fixed selection of analysis algorithms. In this paper, we argue in favor of extensibility as a key feature of data mining systems, and discuss the requirements that this entails for system architecture. We identify in which points existing data mining systems fail to meet these requirements, and then describe a new integration architecture for data mining systems that addresses these problems based on the concept of "plug-ins". KEPLER, our data mining system built according to this architecture, is presented and discussed.

Keywords: data mining, system architecture, extensibility, KEPLER

Introduction

Data Mining, or *Knowledge Discovery in Databases* (KDD) aims at finding novel, interesting, and useful information in large real-world datasets (Frawley, Piatetsky-Shapiro, & Matheus 1991; Fayyad, Piatetsky-Shapiro, & Smyth 1996). While building on parent disciplines such as Machine Learning and statistics, the field of data mining differs from these in its stronger orientation to applications on real-world databases. In Machine Learning and statistics, the focus of research tends to be mostly on the *methods* for data analysis, whereas in data mining, the *process* of using such methods to arrive at convincing application results is just as important a topic[1]. For data mining to be successful in practice, good system support for the data mining process can be just as crucial as having the right analysis methods.

Data mining researchers have responded to this challenge by creating data mining systems that combine support for all steps of the data mining process (Fayyad, Piatetsky-Shapiro, & Smyth 1996, p. 10) with a fixed selection of analysis algorithms in one integrated environment. ISL's Clementine (Integral Solutions Ltd. 1996) and Lockheed's Recon (Simoudis, Livezey, & Kerber 1996) are two commercially available examples of such systems, the former offering decision trees and neural networks, the latter also including clustering and instance-based algorithms. At the same time, however, with more and more reported applications of data mining, it is becoming increasingly clear that there can never be a fixed arsenal of data mining analysis methods that covers all problems and tasks. New methods are continually becoming available, and in many cases, algorithms are adapted or newly developed specifically for the requirements of a particular application (see e.g. (Apte & Hong 1996; Ezawa & Norton 1995)).

In this paper, we examine *extensibility*, i.e., the capability of integrating new analysis methods with as little effort as possible, as a central requirement for data mining systems to address the above problem. In the following section, we motivate the need for extensiblity with reference to the dynamic nature of the data mining process, and examine the shortcomings of existing data mining architectures in light of extensibility requirements. We then show how an architecture based on the concept of "plug-ins" can overcome these problems, and describe KEPLER, an integrated data mining system developed and implemented as a testbed for our architectural concepts. After an evaluation and discussion of related work, we conclude with pointers to future work.

Motivation and goals of extensibility

Conducting KDD in a given database or set of databases is still an art, perhaps even more so than in KDD's parent disciplines machine learning and statistics. Even in the unlikely situation that there are clear-cut goals at the outset, it is impossible even for a skilled analyst to predict just which analysis method will give the best results. More likely, the goals of data mining will not be clear beforehand, but will evolve as a result of the data mining process. In practice, this means that in many situations, several methods are tried and their results compared or combined to get the desired results. In some situations, algorithms have actually been adapted or developed for a particular application (see e.g. (Apte & Hong 1996; Ezawa & Norton 1995)).

When single-strategy data mining systems are employed, the above means that in many cases, analysts will have to switch from one system to another during the course of working on one data mining problem, incurring all the trivial but extremely time-consuming problems of adapting

[1] In fact, some authors reserve the term KDD to denote the entire process, whereas data mining is used to refer to a single analysis step (Fayyad, Piatetsky-Shapiro, & Smyth 1996).

to different user interfaces and converting data and results back and forth. While existing multi-strategy systems like RECON (Simoudis, Livezey, & Kerber 1996) or CLEMENTINE (Integral Solutions Ltd. 1996) alleviate this problem somewhat by offering multiple analysis algorithms (see related work section below), they still offer a fixed choice of algorithms, leaving the unsolved problem of what to do if the necessary method happens not to be included. Only an *extensible* system architecture can solve this problem in a fundamental way, allowing new methods to be added to the system whenever required.

Of course, in a trivial way, every system is extensible by reprogramming the system to implement a desired method, so we need to be more precise in the meaning of extensibility. An extensible system is a system into which new methods can be integrated without knowledge of system internals and without reprogramming of the system kernel, by people other than the system developers. If a system is extensible in this fashion, different users can extend the system to match the requirements of individual data mining problems. In practice, even though end users will not be able to perform such extensions since some configuration and programming will be required, the concept of extensibility is important for skilled analysts who can integrate new methods without having to switch systems, or for algorithm developers who can make their methods available to end users without having to develop a complete data mining system.

Finally, for extensibility to make sense, it is not sufficient to integrate new methods each with their own user interfaces and different ways of starting tasks and looking at results. Instead, the data mining kernel system must offer mechanisms that allow the user to manage, in a uniform way, the specification of analysis tasks and inspection of results of the different methods. Tasks and results must be first-class objects in the system kernel to allow the user to restart and modify tasks and to compare and combine results. Without integrated and uniform access to the tasks and results of new methods, a lot of the benefits of extensibility for the data mining process are lost: if each new extension has different ways of managing tasks and results, switching methods also means switching to a different user interface, resulting in an unnecessarily high learning overhead for the user.

Architecture

Within multi-strategy tool architectures, a popular distinction is to separate approaches that integrate at the *micro level* and those that integrate at the *macro level* (Emde *et al.* 1993). In macro level integration, each method to be integrated remains a separate module with its own internal representations and storage structures, but is coupled to other modules by receiving inputs and passing results back across a suitable channel. In micro-level integration, all modules directly rely on a common repository of data without transformation, and cooperate during processing, not only when they have finished. Multi-strategy data mining systems have so far mostly been realized by macro-level integration, e.g. in Recon (Simoudis, Livezey, & Kerber 1996) where several modules are linked to a data server. Micro-level integration is only beginning to be attempted, e.g. in the KESO project, where all search modules use a common hypothesis space manager and can share description generation operators (Wrobel *et al.* 1996).

Clearly, it is difficult to make a micro-level integration architecture extensible in the sense defined above, since to integrate a new method, the internals of existing methods and the system kernel must be known. The architecture we have chosen is therefore based on macro-level integration. Each tool that is part of the system is an independent software module and can be realized e.g. in different programming languages. To reach the extensibility goal defined above, we have extended the macro-level integration architecture into the *plug-in* architecture shown in Figure 1.

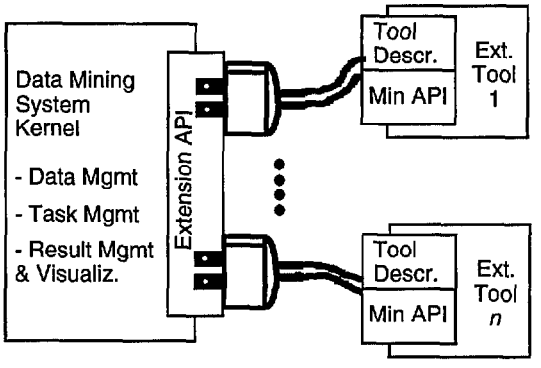

Figure 1: General plug-in architecture

The major components to realize extensibility in this architecture are the following:

- a well-defined and open extension API (application programming interface) through which extensions access data and communicate results back to the system
- declarative extension tool description containing information about the data accepted and needed by an analysis tool and the kinds of results produced
- a task and result manager that offers uniform access to tasks and results (specification, manipulation, visualization), exploiting the declarative specifications in tools
- a minimal tool API which the kernel uses to perform tool-specific functions

The extension API is the basic component towards reaching extensibility. This API must offer hooks for extension tools to access the data in the database and to communicate results back to the system. By making the definition of this API precise and open, it can be ensured that the developer of an extension need only know the API; the internals of the kernel system can be encapsulated. In domains such as image manipulation, this concept of open APIs has led to very simple and extensible systems (this is where we have borrowed the term "plug-in").

As detailed above, however, the situation in data mining is more complex since not only does the "plug-in" need to access data and return results, but the kernel also needs to

provide uniform access to the tasks and results being worked on by extensions. Since the tasks of data mining methods vary widely, ranging from classification to clustering to pattern discovery and further, there needs to be a facility with which each extension can declare the required inputs and kind of results. Furthermore, since result types also vary greatly, the extension must be able to declare in which way its results need to be visualized, choosing from the available visualization facilities in the kernel. Alternatively, the API can include visualization operations as well.

Third, the proposed architecture contains a task and result manager that makes use of the declarative information described above to ensure uniform access to tools, whether they be included with the system from the start or added later on. Ideally, based on the declarative specification in the extension, the kernel should be capable of dynamically generating an appropriate graphical user interface to allow the same comfort of usage for all tools. Tasks specifications and results are managed by the analysis task manager, allowing the user to redo and modify each task, check and interrupt tasks that are running (perhaps in parallel), and inspect, test and compare results.

Note that in general, results will not be interpretable to the task manager. This is why in such an architecture, there must be a fourth component, a minimal tool API containing functions that each tool must supply to the kernel. Besides the analysis functionality proper, each tool primarily has to offer hooks for dealing with its results, ranging from simple things like producing printed output to hooks for testing and visualizing results. To simplify this functionality, the kernel's API should include libraries of common testing and visualization functions (e.g. for common results such as decision trees).

The rest of the plug-in architecture contains the standard components that are found in most data mining systems, namely facilities for importing, exporting, selecting, and transforming data.

Kepler

At a general level, the architecture described in the preceding section contains the components that are necessary to make extensibility work: with knowledge of only the API and the tool declaration language, an extension developer can produce a tool that can be plugged in and will be fully supported by the kernel — provided that such APIs (i.e., declarations and facilities) can be designed in such a way and still support more than a narrow class of plug-in tools. We have constructed and fully implemented a data mining system termed KEPLER[2] to prove the feasibility of the proposed architecture. Given the data mining applications we are working on (market study data, 10^4 objects, ecological system analysis, 10^5 objects, protein structure prediction (Dzeroski *et al.* 1996), 10^4 objects), we decided to target KEPLER to medium-range data mining problems (10^4 to 10^6 objects). Our choice was also motivated by the fact that many other published data-mining applications fall into this class (e.g. (Apte & Hong 1996) 10^4, (Dvzeroski & Grbović 1995) 10^3, (Feelders, le Loux, & van't Zand 1995) 10^5 objects, (Li & Biswas 1995) 10^5, (Sanjeev & Zytkow 1995) 10^3, (Simoudis, Livezey, & Kerber 1995) 10^6), and by the fact that this application size, there exists a number of available ML and KDD algorithms (from our own group and others) that could be used to test the feasibility of the plug-in concept.

KEPLER's general architecture is based on the concept of a *workspace* that stores all data, tasks, and results of a data mining problem domain. Data are represented as relations (with associated key and schema information) and can be organized in different *datasets* (subsets of relations). Tuples are stored in a *data management layer* which is currently mapped to a main memory based storage scheme with disk write-through to compiled files, opening the possibilility of "swapping out" currently unused data. For the target application size, this has turned out to be a good choice. Nonetheless, in future versions of the kernel, the data management layer will map down to a database management system to gain scalability and security.

The extension API of Kepler contains only the elementary calls that are necessary for extensions to access data on a set-oriented or tuple-oriented basis, and to pass back results. When called, each extension receives a *data specification* that it passes back to the kernel whenever it wants to access data; the kernel then maps this to the actual data. This scheme protects extensions from details of data access while still allowing reasonable efficiency since even direct read-through to a database can be realized. A second set of calls is available to pass back results to the kernel as soon as they are generated.

As a tool description language, KEPLER uses parameter and result declarations. Parameter declarations come in two types: inputs and algorithm parameters. Input parameters state what kinds of inputs a tool expects in terms of supplied primitives such as "relation name" or "attribute name" or general parameter types such as "integer", "boolean" or "oneof". These are employed by the task manager in KEPLER to automatically generate the appropriate interface masks to allow the user to specify an analysis task. Similarly, algorithm parameters state the available parameters to influence algorithm behavior, indicating the allowed values in a similar fashion by using predefined parameter types. Here also, KEPLER automatically generates input masks with radio-buttons, sliders, pop-up menus etc. to let the user specify the tool parameters. Hence, KEPLER offers a uniform interface style for all extension tools, minimizing the user's effort to become familiar with a new tool (see Figure 2, next page, for an example of a task window generated from a tool description). Through the task manager, tasks can be created, edited, started and stopped. Cross-validation and other analysis scripts will be possible in the future.

As for analysis results, each tool can declare several result types. However, currently these are only utilized for user information and help. Each extension can pass back to the system kernel as many results of each type as desired; the task manager stores them on disk and allows the user to

[2] KEPLER will be demonstrated at the conference.

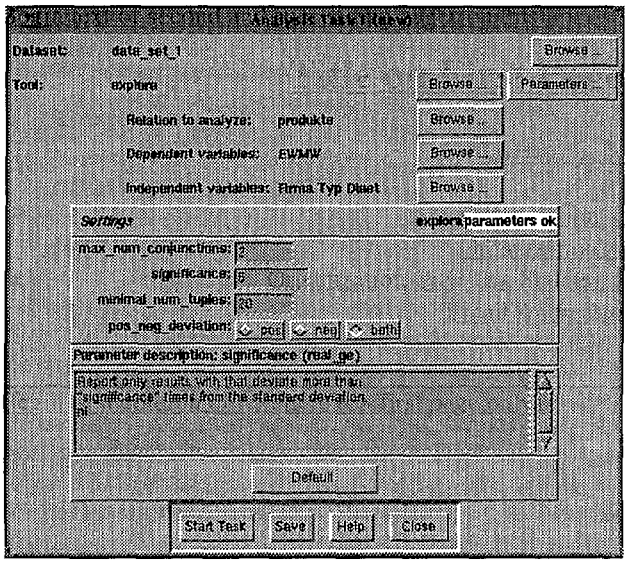

Figure 2: Task window for an EXPLORA task

select and manipulate them. It is also possible for tools to pass back a pointer to a binary results file which is from then on managed by the task manager. As indicated in the previous section, whenever a result needs to be tested on different data or needs to be visualized, the kernel calls the appropriate hooks in the extension tool which in turn can rely again on testing and visualization APIs that the kernel offers as a library.

To test the feasibility of the plug-in concept, we decided to integrate as wide a selection of plug-ins as possible, ranging not only across different methods, but also across different tasks. At first, we chose to include as plug-ins for internal use at our site:

- for classification tasks, the decision tree algorithm C4.5 in the original C version available from Ross Quinlan (Quinlan 1993), a backpropagation neural network realized using SNSS (Zell 1994), as well as our own instance-based method KNN (Wettschereck 1994) and Salzberg's NGE (Salzberg 1991; Wettschereck & Dietterich 1995)).
- for clustering tasks, the AUTOCLASS algorithm (Cheeseman & Stutz 1996) as available from the authors
- for regression tasks, the MARS (multiple adaptive regresssion spline) algorithm of (Friedman 1991), in our own implementation
- the pattern discovery algorithm EXPLORA (Klösgen 1996) developed at GMD (ported from Macintosh to Sun)

All of these are operational in KEPLER at present. To minimize programming overhead, all algorithms were taken as is (except for porting to other platforms as indicated) and used in a compiled form, implementing communication through files which is feasible for the chosen target application size. Summarizing this first experiment with the plug-in concept, we can state that except for reimplementation or porting efforts, the integration of an algorithm of one of the above types or tasks into KEPLER as a plug-in takes at most one day[3], since all that is involved is the simple parameter declaration and the writing of grammars for file creation and reading.

Given this kind of encouraging evidence, we have extended the plug-in idea to other areas of the system as well. In KEPLER, the user can plug new input formats and new data transformation operators into the system in a very similar fashion. It is therefore no problem to handle applications that require e.g. an unusual aggregation of tuples, a particular re-representation of time series, or some other preprocessing operation for a particular analysis method. For example, a specific transformation (from DINUS (Džeroski, Muggleton, & Russell 1992)) is available for transforming first-order representations (across several relations) into a manageable propositional representation in one relation. These plug-ins complement the standard ASCII input formats and predefined transformation operators like sampling and discretization.

Evaluation

KEPLER has been evaluated on three data mining applications: analysis of retail data, ecological system analysis, and protein structure prediction (Dzeroski *et al.* 1996). In the retail data application (roughly 15.000 tuples), the primary goal was discovery of interesting customer groups (carried out with EXPLORA). In addition, certain customer groups were characterized using C4.5. In the ecosystem domain (roughly 120.000 tuples), the primary goal was to derive ecological conditions for the occurence of certain plants, a secondary goal being clustering of plants into ecological groups. In the protein structure application (ca. 10.000 tuples), the goal was to predict protein structure from spectography data, mostly performed using ILP techniques (Dzeroski *et al.* 1996).

In these applications, all of which required multiple tasks to be solved from the same pool of data, we have found our speed of turning up results to be greatly increased, since all the time usually spent in preprocessing data and changing formats when the method first chosen turns out inappropriate was eliminated. Furthermore, through the automatic generation of graphical interfaces for plug-in tools, even relative newcomers to data analysis (students) produce first results fast. Thus, from an application perspective, KEPLER has reached its goals of offering multiple tools in an integrated, easy-to-use environment.

From the development perspective, in our view, KEPLER has shown that indeed it is possible to realize an extensible data mining system architecture based on the idea of plug-ins. For ourselves as system developers, the concept has resulted in integration times in the order of hours, meaning that new methods (once they are available) can be integrated very fast, resulting in a choice of methods in one system that even with the set of methods described above appears unmatched in any other system.

[3]Not including the time it takes to get to know the extension algorithm.

The limits of the present design are in scalability due to the way tools are integrated. To allow this kind of extremely rapid integration, we have used existing code (our own and code made available by others) unchanged, using operating-system level file communication. For the kinds of dataset sizes for which KEPLER is targeted, this has turned out appropriate; it will not, however, scale well much beyond 10^6 objects. To go to these scales would require algorithms to be rewritten to use the kernel's data access facilities directly. When looking at our own and published applications, however, a large number of problems seem to fall in the size range below 10^6 objects.

Related Work

The architectural concepts used here are most closely related to the idea of "external tools" that was realized in the MOBAL knowledge acquisition system (Morik *et al.* 1993; Emde *et al.* 1993). In MOBAL, a number of ILP learning algorithms could be used from within the same graphical environment. However, compared to KEPLER and the general architecture discussed here, MOBAL is lacking in important respects. Analysis tasks and results are not first class objects in MOBAL, making it very difficult for the user to keep track of what was done when and with which tool. Runs cannot be repeated. There is no declarative tool description and no general way of passing inputs and storing results, since it is implicitly assumed that all algorithms are ILP algorithms that always take the entire database as input and always produce first-order Horn clauses as output.

At the general level of multistrategy data mining, there are several tools which can be usefully compared to the work presented here. For commercial tools, there is ISL's CLEMENTINE system (Integral Solutions Ltd. 1996), integrating decision trees and neural networks. This system offers an excellent user interface, but appears limited to classification and clustering tasks. Lockheed's Recon system (Simoudis, Livezey, & Kerber 1996) addresses a wider range of problems, including also instance based methods. Both systems, however, seem to lack the extensibility that characterizes the architecture presented here, and seem to require kernel reprogramming to add new algorithms. Thinking Machine's Darwin system (Thinking Machines Corp. 1996) appears to be more a collection of tools than an integrated system.

Among other multi-strategy systems, there is DBMINER (previously DBLEARN) (Han *et al.* 1992; Han & Fu 1996) which discovers multiple kinds of knowledge, but is based on a single attribute-oriented discovery methods and is not extensible. MLC++ (Kohavi *et al.* 1994) is a collection of C++ programs designed to be configured by a user into a working Machine learning algorithm. Since the source code is available, MLC++ is an extensible system. Extensibility, however, is achieved purely at the programming and algorithm level, as MLC++ is a more a library than a system, not offering the system kernel and user interface support discussed here. The INLEN system (Michalski 1992; Ribeiro, Kaufman, & Kerschberg 1995) is related to the work presented here since it also conceptualizes data management and analysis as operators, however without a focus on extensibility and closely tied to AQ and related methods. Similarly, GLS (Zhong & Ohsuga 1995), is a multi-strategy system with four fixed analysis methods without extension facilities.

Finally, a useful comparison is with the architectural concepts of the KESO data mining project in which we are also involved (Wrobel *et al.* 1996). In KESO, the very explicit goal at the outset was to create a data mining system capable of handling the very large scale problems ($>>10^6$ objects). Consequently, a macro-level integration as was used in KEPLER was excluded, as it would not have offered the required efficiency. Instead, KESO uses a micro-level integration architecture based on a common hypothesis space manager that maintains a persistent representation of the search space in a database. Different search modules can share subcomponents like the description generator (refinement operator). Since KESO is designed for a particular class of problems (finding interesting subgroups) and extreme efficiency, extensibility across wide task ranges was not of concern. Extending KESO requires reexpressing an algorithm in KESO's framework, but of course (these are the benefits of rewriting and micro-level integration) the newly written search module may use e.g. facilities used in the construction of other methods, like the description generator or a quality computation module.

Conclusion

Based on our own experience and other reported applications, we believe extensibility to be a key feature of any data mining system to keep up with the variability of data mining tasks which does not allow to design a system once and for all that has the right methods for all situations. The key to extensibility is extensibility without system core reprogramming, which allows third parties other than developers to extend a system in their direction without knowing the system's internals. This requires carefully designed extension APIs and declarative tool descriptions so that the kernel may support extensions tools in the same fashion as possible in non-extensible systems. Due to the variety of methods and tasks that could be present in an extensible system, a task manager is a central component of such a system.

With KEPLER, we have realized an extensible system that offers a very wide range of methods and tasks for medium sized data mining problems. The experience of integrating all these different methods shows that the KEPLER's plug-in architecture indeed is a basis for a very rapid extension of the system that does not require kernel reprogramming. Even though we have not proved it in implementation, we nonetheless believe that the plug-in concept can be scaled up to even larger problems with some more effort in designing tool interfaces. Our own future work may move into this direction, but most likely will first concentrate on integrating more extension tools and further refining the system in other applications than the ones it was already tested on. For the distant future, we hope to have the system in a state that allows us to make it available to others and to publish the API specification. This would give developers of new

data mining algorithms a simple platform for delivering their methods to uses without having to worry about user interfaces or data access.

References

Apte, C., and Hong, S. 1996. Predicting equity returns from securities data. chapter 22, 542 – 560. In (Fayyad *et al.* 1996).

Cheeseman, P., and Stutz, J. 1996. Bayesian classification (AutoClass): Theory and results. chapter 6, 153 – 180. In (Fayyad *et al.* 1996).

Džeroski, S., and Grbović, J. 1995. Knowledge discovery in a water quality database. 81 – 86. In (Fayyad & Uthurusamy 1995).

Džeroski, S.; Muggleton, S.; and Russell, S. 1992. PAC-learnability of determinate logic programs. In *Proc. 5th ACM Workshop on Comput. Learning Theory*, 128–135.

Dzeroski, S.; Schulze-Kremer, S.; Heidtke, K.; Siems, K.; and Wettschereck, D. 1996. Knowledge discovery for diterpene structure elucidation from 13C NMR spectra. Proc. ECAI-96 workshop on Intelligent Data Analysis for Medicine and Pharmacology.

Emde, W.; Kietz, J.-U.; Sommer, E.; and Wrobel, S. 1993. Cooperation between internal and external learning modules in mobal: different facets of multistrategy learning. MLnet workshop on multistrategy learning, Blanes, Spain.

Ezawa, K., and Norton, S. W. 1995. Knowledge discovery in telecommunications services data using bayesian network models. 100 – 105. In (Fayyad & Uthurusamy 1995).

Fayyad, U., and Uthurusamy, R., eds. 1995. *Proc. First Int. Conf. on Knowledge Discovery and Data Mining*. Menlo Park, CA: AAAI Press.

Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R., eds. 1996. *Advances in Knowledge Discovery and Data Mining*. Cambridge, USA: AAAI/MIT Press.

Fayyad, U.; Piatetsky-Shapiro, G.; and Smyth, P. 1996. From data mining to knowledge discovery: An overview. chapter 1, 1 – 34. In (Fayyad *et al.* 1996).

Feelders, A.; le Loux, A.; and van't Zand, J. 1995. Data mining for loan evaluation at ABN AMRO: a case study. 106 – 111,. In (Fayyad & Uthurusamy 1995).

Frawley, W.; Piatetsky-Shapiro, G.; and Matheus, C. 1991. Knowledge discovery in databases: An overview. In Piatetsky-Shapiro, G., and Frawley, W., eds., *Knowledge Discovery in Databases*. Cambridge, USA: AAAI/MIT Press. chapter 1, 1 – 27.

Friedman, J. 1991. Multivariate adaptive regression splines (with discussion). *Annals of Statistics* 19(1):1–141.

Han, J., and Fu, Y. 1996. Exploration of the power of attribute-oriented induction in data mining. chapter 16, 399 – 421. In (Fayyad *et al.* 1996).

Han, J.; Cai, Y.; Cercone, N.; and Huang, Y. 1992. DBLEARN: A knowledge discovery system for databases. In *Proc. 1st Int. Conf. Inf. & Knowl. Managmt.*, 473 – 481.

Integral Solutions Ltd. 1996. Clementine data mining system: Decisions from data. WWW http://www.isl.co.uk.

Klösgen, W. 1996. Explora: A multipattern and multistrategy discovery assistant. chapter 10, 249 – 271. In (Fayyad *et al.* 1996).

Kohavi, R.; John, G.; Long, R.; Manley, D.; and Pfleger, K. 1994. MLC++: A machine learning library in C++. In *Proc. Tools with Artificial Intelligence*, 740 – 743. IEEE Computer Society Press.

Li, C., and Biswas, G. 1995. Knowledge-based scientific discovery in geological databases. 204 – 210. In (Fayyad & Uthurusamy 1995).

Michalski, R. *et. al.*. 1992. Mining for knowledge in databases: The INLEN architecture, initial implementation and first results. *J. Intell. Inf. Sys.* 1(1):85 – 113.

Morik, K.; Wrobel, S.; Kietz, J.-U.; and Emde, W. 1993. *Knowledge Acquisition and Machine Learning: Theory Methods and Applications*. London: Academic Press.

Quinlan, J. R. 1993. *C4.5 — programs for machine learning*. San Mateo, CA: Morgan Kaufman.

Ribeiro, J. S.; Kaufman, K. A.; and Kerschberg, L. 1995. Knowledge discovery from multiple databases. 240 – 245. In (Fayyad & Uthurusamy 1995).

Salzberg, S. 1991. A nearest hyperrectangle learning method. *Machine Learning* 6:277–309.

Sanjeev, A., and Zytkow, J. 1995. Discovering enrollment knowledge in university databases. 246 – 251. In (Fayyad & Uthurusamy 1995).

Simoudis, E.; Livezey, B.; and Kerber, R. 1995. Using Recon for data cleaning. 282 – 287. In (Fayyad & Uthurusamy 1995).

Simoudis, E.; Livezey, B.; and Kerber, R. 1996. Integrating inductive and deductive reasoning for data mining. chapter 14, 353 – 374. In (Fayyad *et al.* 1996).

Thinking Machines Corp. 1996. Darwin: Intelligent data mining. WWW http://www.think.com.

Wettschereck, D., and Dietterich, T. 1995. An experimental comparison of the nearest-neighbor and nearest-hyperrectangle algorithms. *Machine Learning* 19:5–28.

Wettschereck, D. 1994. *A Study of Distance-Based Machine Learning Algorithms*. Ph.D. Dissertation, Oregon State University.

Wrobel, S.; Wettschereck, D.; Verkamo, A. I.; Siebes, A.; Mannila, H.; Kwakkel, F.; and Klösgen, W. 1996. User interactivity in very large scale data mining. Contact keso-develop@cwi.nl.

Zell, A. e. 1994. SNNS user manual, version 3.2. Fakultätsbericht 6/94, IPVR, Universität Stuttgart, Germany.

Zhong, N., and Ohsuga, S. 1995. Toward a multi-strategy and cooperative discovery system. 337 – 342. In (Fayyad & Uthurusamy 1995).