A Framework to Integrate Data Mining Components

Ronaldo Cristiano Prati  
USP, Instituto de Ciências Matemáticas e de Computação  
São Carlos, Brasil, 13560-970  
prati@icmc.sc.usp.br

and

Marcos Roberto Geromini  
USP, Instituto de Ciências Matemáticas e de Computação  
São Carlos, Brasil, 13560-970  
mrgera@icmc.sc.usp.br

and

Maria Carolina Monard  
USP, Instituto de Ciências Matemáticas e de Computação  
São Carlos, Brasil, 13560-970  
mcmonard@icmc.sc.usp.br

Abstract

The amount of electronically stored data increases everyday. As the volume of data increases, our ability to understand it decreases. However, lying hidden in all this data there is potentially useful information or knowledge. This problem has given rise to new generation techniques and methods to help data analysis and knowledge discovery. These methods and tools are object of study of a relatively new research area, called Data Mining. Data Mining can be defined as the process of discovering patterns in data. In fact, the Data Mining process consists of three main phases: data pre-processing, pattern (or knowledge) discovery and pattern pos-processing. Aiming to help research in this area we are developing a free software integrated environment, which covers the tree phases of Data Mining, called DISCOVER. This paper describes the approach we are using to integrate the components in that system as well as the development of what we consider a novel user interface approach on this area.

Keywords: Data Mining, Knowledge Discovery, Software Architecture, Software Process, Component Framework, User Interface, Usability.
1 Introduction

Due to the availability of cheap disk space and automatic data collection mechanisms, huge amounts of data, often reaching terabytes, are becoming common in business and science databases. Examples include the customer’s databases of web vendors, financial and business transactions, chemistry and bioinformatics databases and remote sensing data sets. Besides being used to assist in daily transactions, such data may also contain an important kind of information concealed on data: the one that can be translated into knowledge. The aim of Data Mining (DM) is to extract useful information out of such large data collections [8].

There are several Machine Learning (ML) algorithms that can be used in DM tasks. However, it is not possible to apply such algorithms without careful data understanding and preparation, which may often dominate the actual Data Mining activity [24, 19]. In addition, evaluating, understanding, and interpreting the extracted knowledge are also hard activities.

These difficulties increase due to the fact that for a specific domain, there is not a mathematical analysis able to predict that the performance of a ML algorithm will be better than other. This means that experimentation plays an important role in ML [12].

Generally, ML algorithms use different input data formats. Furthermore, ML algorithms use different languages to express the induced knowledge. Besides that, there are some routine activities that need to be done when performing a Data Mining application. Thus, it is important to have tools to automate some of these activities, in order to help the user through the Data Mining process.

With these in mind, a group of researchers at our laboratory — Computational Intelligence Laboratory (LABIC)1 — is working on a system, called DISCOVER [2], that will be used to help our research activities in DM and ML. The aim of the system is to integrate the ML algorithms most frequently used by the community with the data and knowledge processing tools developed as the results of our work. The system can also be used as a workbench for new tools and ideas.

This paper describes the approach we are using in the development of this system and is organized as follows: Section 2 presents some other integrated systems used for DM tasks. Section 3 presents a brief introduction to Data Mining, as a motivation to our work. Section 4 describes the DISCOVER project, along with its development approach. Finally, Section 5 presents some conclusions and future work.

2 Related Work

Several efforts have been made to produce tools that provide multiple ML algorithms and tools for Data Mining. Following, we report some well-known public domain tools used by the community.

The MLC++ project, developed at Stanford University, was designed to help the use of existing ML algorithms and developing new ones. MLC++ provides interfaces to several well-known ML algorithms, defines a standard format and data conversion among those algorithms, generates performance statistics such as accuracy, learning rates and confusion matrices as well as graphical visualization of the symbolic structures learned by some of those algorithms [12].

However, the main objective of MLC++ is the accuracy obtained on the whole learning process, not giving an unified vision of the induced knowledge. In addition to that, since 1995 MLC++ has been turned into a commercial product and its public-domain version is no longer supported.

The Weka toolbox [25], developed at Waikato University, is a collection of ML algorithms re-implemented in Java and runs on almost any platform. This approach produces standard interfaces and uniform code and is also well suited for developing new tools. However, the new versions of the original algorithms might not be incorporated in the Weka because they need to be re-implemented in Java. Besides that, recoding may fail to ensure that the behaviour of the re-implemented algorithm is exactly the same as the one already implemented by its author.

YALE [20] is an environment for machine learning experiments that can be made up of a large number of arbitrarily nestable operators and their setup is described by XML files. Although it has several facilities for data processing, it does not provide enough tools for knowledge processing.

1http://labic.icmc.usp.br
A first generation of commercial tools used are DM is currently on the market. Those tools includes the Silicon Graphics’ MineSet™ 2, IBM’s Intelligent Miner for Data3, SPSS’ Clementine4 and Oracles’ Darwin5.

MineSet™ uses MLC++ as a base for ML algorithms. Over the MLC++ core, MineSet™ provides a graphical user interface (GUI) and the learned structures are shown using 3D visualization tools. Intelligent Miner for Data provides a variety of knowledge discovery algorithms for classification, association, clustering, and sequential pattern discovery. It is integrated with IBM’s DB2 database system. Clementine offers a GUI based on “graphical programming”, that can be used with drag and drop of icons and objects. This kind of interaction helps the user combining objects to produce a sequential process. Darwin is a system proposed to work with large data sets, it uses parallel algorithms and is integrated with Oracle database system.

However, commercial tools are generally developed as closed products from the end user viewpoint. Although they provide the user with exploratory features, privileging user-friendly interfaces and intuitive interaction, they overlook algorithms details and offer a limited set of methods for the different tasks of the Data Mining process. This avoids the possibility for the technical user or researcher to develop and integrate new Data Mining methods.

Furthermore, commercial systems generally are too expensive for some developing countries universities budget.

3 Data Mining

Data Mining is not an algorithmic activity but a process, i.e., there is not a recipe we should follow to obtain good results. DM can be viewed as an engineering activity, requiring experimentation, analysis and comparison of many different models in the search for useful results.

This process is inherently interactive and iterative: we cannot expect to obtain useful knowledge simply by giving a lot of data to a system. The user of a Data Mining system needs to have a solid understanding of the domain in order to select the right subsets of data, suitable pattern classes, and good criteria for interestingness of these patterns. Thus, Data Mining systems should be seen as interactive tools, not as automatic analysis systems.

The DM process consists of several phases, that can be gathered in three main groups:

1. data processing (or pre-processing);
2. discovery of patterns from data and,
3. patterns processing (or pos-processing).

These phases, shown in Figure 1, should be preceded by the domain understanding and followed by the evaluation and use of the obtained results.

Processing the data set involves selection of the data sources, integration of heterogeneous data, cleaning the data from errors, assessing noise, dealing with missing values, etc. This step can easily take up to 80% of the time needed for the whole Data Mining process [24, 19].

The pattern discovery phase is the step where the interesting and frequently occurring patterns are discovered from the data. At this step various machine learning techniques can be used, such as rule learning, decision tree induction, clustering, inductive logic programming, etc. Although ML algorithms and methods to extract knowledge from data build up the core of Data Mining, there are some differences between these areas [14].

The emphasis on the process of knowledge discovery is one of them; large part of the literature on Machine Learning concentrates only on the learning or induction step while in DM the data preparation and knowledge interpretation are as important as the learning phase.

2http://www.sgi.com/software/mineset.html
4http://www.spss.com/Clementine
5http://otn.oracle.com/products/datamining/content.html
The next difference concerns the relative role of concepts and data. In ML research, we assume that there is an underlying interesting concept or mechanism that has produced the data and the objective is to discover this underlying concept. The data may be corrupted by noise, errors, etc., but still the idea is that there is an interesting concept to be discovered. On the other hand, in knowledge discovery, we may not be interested in discovering the whole concept; useful patterns of information are sufficient.

Another difference is the amount of data. Traditionally, ML research has concentrated on data sets containing hundreds or thousands of records (examples), while Data Mining applications consider much larger data sets. Furthermore, the essential source of complexity in Data Mining is typically not the number of examples in the data set, but rather the number of attributes used to describe the examples.

The validation of knowledge is yet another difference. Most of the approaches used in ML have based their validity measures on the accuracy of the induced model, while Data Mining asks for other measures (including interestingness and surprisingness) [9, 10].

The comprehensibility and usability are also two other criteria that should be considered in the knowledge validation phase. The Data Mining process does not end up after patterns have been discovered. The user should be able to understand what has been discovered, to view the data and the patterns simultaneously, contrast the discovered patterns with his/her background knowledge, etc.

Data Mining also has a close correlation with other areas, such as statistics, database, pattern recognition, artificial intelligence, knowledge acquisition for expert systems, data visualization, and high performance computing. The unifying goal is to extract high-level knowledge from low-level data.

In particular, Data Mining has much in common with statistics, particularly exploratory data analysis methods. The statistical approach offers precise methods for quantifying the inherent uncertainty that comes out when we try to infer general patterns from a particular sample of an overall population. Data Mining systems often embed particular statistical procedures for sampling and modelling data, evaluating hypotheses, and handling noise within an overall knowledge discovery framework.

This process can take different visions depending what is focused through the DM process. From the point of view of the analyst (the person who is responsible for patterns extraction), the process can be described in terms of a hierarchical process model, as shown in Figure 2, consisting of sets of tasks described at four levels of abstraction (from general to specific): phases, generic tasks, specialized tasks and process instances [4]. In a Data Mining tool, this vision is useful when we are mapping the Data Mining process into procedures.

At the top level, the Data Mining process is organized into a number of phases; each phase consists of several second-level generic tasks. This second level is called generic, because it is intended to be general enough to cover all possible situations. The generic tasks are intended to be as complete and stable as possible. Complete means covering both the whole process of Data Mining and all possible applications. Stable means that the model should be valid for yet unforeseen developments like new modeling techniques.
The third level, the specialized task level, is the place to describe how actions in the generic tasks should be carried out in certain specific situations. For example: at the second level there might be a generic task called “data cleaning”; the third level describes how this task differs in different situations, such as cleaning numeric values versus cleaning categorical values or whether the problem type is related to descriptive or predictive modelling.

The description of phases and tasks as discrete steps performed in a specific order represents an idealized sequence of events. In practice, many of these tasks can be performed in a different order and it will often be necessary to repeatedly backtrack to previous tasks and repeat certain actions. This model does not attempt to capture all possible routes through the Data Mining process since this would require an overly complex process model.

The fourth level, the process instance, is a record of the actions, decisions and results of an actual Data Mining engagement. A process instance is organized according to the tasks defined at the higher levels, but represents what actually happens in a particular engagement, rather than what happens in general.

4 The Discover Project

As we mentioned before, Data Mining melts down several knowledge acquisition approaches, including ML, without being just a concatenation of these approaches. Integrating these methods will generate new solutions, and new research problems too. Research on this area includes the development of new tools and techniques to automate the process of extracting knowledge from data.

The DISCOVER project aims the development of an integrated environment offering functionalities and support for the research in all phases of the Data Mining and Text Mining (TM) processes using learning algorithms. Its development approach is based on a collaborative work strategy to develop a set of integrated tools to supply our, as well as other research groups, necessities.

In this environment we use learning algorithms already implemented by the ML community as well as specific tools we have developed, such as data (and text, in the case of Text Mining) processing for supervised and unsupervised learning, sampling, error and accuracy. Even though the systems cited on Section 2 offer similar features, the proposed approach has a strong focus on the understanding and evaluation of symbolic knowledge, such as rule interestingness, novelty and quality, as well as rules merging.

The use of already implemented algorithms which have been thoroughly tested by the ML community not only avoids re-implementation but also ensures the results correctness.

We intend the DISCOVER to work as a workbench in order to help Data Mining research. The advantage of the DISCOVER as a research system is the unifying vision on which the system handles the objects using the standard formats (as explained on Section 4.1) and the fact that the user has total control on the planning and execution of experiments. Such control is given by a user-centered interface, developed with usability driven methods and theories. This interface allows a totally graphical manipulation of the components of an experiment, and is better explained on Section 4.3.

The DISCOVER system is based on the scripting language Perl⁶, a very good tool for rapid code de-

⁶http://www.perl.org
development. This language provides a regular expression engine, which is a powerful sub-language that can perform complex text manipulation and data extraction tasks.

Perl can handle large amounts of data efficiently and it is flexible with regards to data types, since it is based on general lists and dictionaries (association arrays), of which the latter are implemented as very efficient hash-tables.

We are also using MySQL\(^7\), one of the most popular open source database systems, to underlie the component database systems dependents.

4.1 Standard Formats

The development of new tools, for both data and knowledge processing, is related to the handling of a set of common objects. For instance, a tool for data acquisition in a WEB system and a tool for data sampling, despite belonging to different tasks (data acquisition and data sampling), have in common the fact that they manipulate data.

Furthermore, those objects can be represented in different formats. For instance, a date can be stored as “month/day/year” in a specific base and as the number of days counted upwards from a specific data in another base. Another example is the different formats used by the learning algorithms to represent the patterns or concepts they induce.

To help the development of new tools, the DISCOVER project adopted the standard format concept to its objects. An object can be a dataset, a classifier, a rule, a measure, etc. Besides that, the use of standard formats for objects allows to have an unifying vision of those objects, facilitating their understanding.

For data representation, we adopted a standard format [3] similar to the MLC\(^*\) data format, but with extensions to Text Mining, unsupervised learning, inclusion of new data types, and the construction of new attributes, \(i.e\), attributes calculated from another (original) attributes. We have also a standard format for classification rules [16], based on the CN2 learning algorithm [5, 6] rule format, but with an addition: for each rule we have the contingency table, which is the base for most rule evaluation measures [13]. Nowadays other researchers in our laboratory are also working on the definition of new standard formats for regression and association rules.

4.2 Integrating Components

Initially, the DISCOVER consisted of only a set of scripts organized in a repository. A script is a short program that performs an atomic task. By combining these independent scripts we were able to perform more complex tasks. In this way, the DISCOVER was a set of independent tools that could be combined depending on the user necessities. However, execution time was high for complex experiments.

For this reasons, we decided to modify the initial proposal by creating an integrated environment, on which classes libraries replaced the scripts. These classes are encapsulated as components, and the components composition is done throughout a Graphical User Interface (GUI).

Although bringing some advantages, such as an easier interface with the user and a performance enhancement, this new approach raised new problems related to the system integration. In order to DISCOVER be an integrated environment, we needed to treat many issues, such as the development process management, communication between the developers and components interactions, like exception handling.

In addition to that, we have to consider some architectural tradeoffs in order to develop a system that reflects our needs. The system should be extensible such that several tools, which amount to dozens of thousands lines of Perl code\(^8\), that have been implemented by LABIC's students as part of theirs M. Sc. and Ph. D. work\(^9\), as well as future tools, can easily be incorporated into the system. Thus, the system’s architecture should impose strong interaction patterns that allow an efficient integration. In other words,

\(^7\)http://www.mysql.com


we need a method to easily integrate components and a set of process that holds and forms an integrated environment.

The idea of software components is as old as the software industry itself. All engineering fields, especially electrical and mechanical engineering, have developed a component market when the field was mature. Following this metaphor, the history of software engineering can be seen as a continuous quest for establishing such a market [21].

Components result from problem decomposition, a standard problem solving technique. In the software world, different conceptions on how systems should be organized result in different kinds of components. Thus, two systems may comprise components, but the components may have nothing more in common than the name “component”.

There is no shortage of definitions for the term component in the literature. This is not surprising, as different understandings of the problems to be solved and approaches for solving these problems invariably lead to different understandings of the constituent parts (components) of solutions to problems. We use the definition presented in [1] as the one that better defines our work:

“A Component is an opaque implementation of functionality, subject to third-party composition and in conformance with a component model.”

A component model specifies the standards and conventions imposed on developers of components. Compliance with a component model is one of the properties that distinguishes components (as we use the term) from other forms of packaged software. When we add to the component model a set of services that support or enforce a component model, we have a component framework.

Thus, a component framework should contain a set of rules to be obeyed by components in a certain environment [23]. These rules form a base on which multiple components can coexist in a single environment. The framework itself does not necessarily need to implement any functionality. Still, it should manage shared resources and provide communication between extension components.

A good way to think of a component framework is as a mini-operating system. In this analogy, components are to frameworks what processes are to operating systems. The framework manages resources shared by components, and provides the underlying mechanisms that enable communication (interaction) among components. Like operating systems, component frameworks are active and act directly upon components in order to manage a component’s life cycle or other resources, for example to start, suspend, resume, or terminate component execution.

However, unlike general-purpose operating systems such as Unix, which support a rich array of interaction mechanisms, component frameworks are specialized to support only a limited range of component types and interactions among these types. In exchange for a loss in flexibility there are improved prospects for component composition.

A system that allows the introduction of new components when they are necessary is called extensible. More formally, a system is called independently extensible, if it can cope with the late addition of extensions without requiring a global integrity check [22]. Extensibility is an important factor in Data Mining systems [26], since Data Mining is still far from being a standard process with a well defined commonly agreed how-tos.

A component framework has as its main part a set of component interface definitions (also known as contracts). The interfaces can be structured in an inheritance hierarchy. This automatically provides the system with a modular architecture.

Public and clearly specified interfaces also ease the integration of new Data Mining algorithms and methods. To introduce new methods in the framework we should guarantee that it implements one of the interfaces (fulfill one of the contracts). The more in common any new tool has with a previously integrated tool, the more concrete the interface it implements would be in the inheritance hierarchy. Different points in the interface inheritance hierarchy will allow different levels of integration. For example, let us suppose our system has a set of tools for decision trees as induction methods and another set of tools that implements functions that work with the induced trees. If we integrate a new decision tree induction method it would

10These are often referred to as inter-process communication mechanisms, and in the case of Unix includes signals, remote procedure calls, pipes, sockets, shared memory and the file system.
be integrated as a **Decision Tree Induction Method** and it would be able to use all the tools that implement functions to **Decision Tree Induction Methods**. If we integrate in the same system a Naive-Bayes classifier, the integration should be in the more general level of **Induction Method**, and the tool would only be able of taking advantage of the services that the system offers at that level.

Furthermore, with the presence of a framework component, the design of new components should consider the interoperability, *i.e.*, we expect that the design of components would be biased towards the component model, and bridges between the different models are being developed and will shortly be of common use. Thus, simply adapting one of these models will give the system an enhanced extensibility.

Another advantage is that a framework component gives a support to repair interface mismatch by integrating heterogeneous systems (which is the case of the learning algorithms developed by the community and used by DISCOVER) using wrappers. The technique consists in inserting code that mediates their interaction in a way that fixes the mismatch. The term *wrapper* implies a form of encapsulation whereby some component is encased within an alternate abstraction. It simply means that clients only access component services through an alternative interface provided by the wrapper.

A Component Framework can also incorporate some other benefits inherited from the object-oriented framework approach [7], such as:

**Modularity** Frameworks enhance modularity by encapsulating volatile implementation details behind stable interfaces. Framework modularity helps to improve software quality by localizing the impact of design and implementation changes. This localization reduces the effort required to understand and maintain existing software.

**Reusability** The stable interfaces provided by frameworks enhance reusability by defining generic components that can be reapplied to create new applications. Framework reusability leverages the domain knowledge and prior experience of developers in order to avoid re-creating and re-validating common solutions to recurring application requirements and software design challenges. Reuse of framework components can yield substantial improvements in programmer productivity, as well as enhance the quality, performance, reliability and interoperability of software.

**Extensibility** A framework enhances extensibility by providing explicit hook methods [18] that allow applications to extend its stable interfaces. Hook methods systematically decouple the stable interfaces and behaviour of an application domain from the variations required by instantiations of an application in a particular context. Framework extensibility is essential to ensure timely customization of new application services and features.

**Inversion of control** The run-time architecture of a framework is characterized by an “inversion of control”. This architecture enables canonical application processing steps to be customized by event handler objects that are invoked via the framework’s reactive dispatching mechanism. When events occur, the framework’s dispatcher reacts by invoking hook methods on pre-registered handler objects, which perform application-specific processing on the events. Inversion of control allows the framework (rather than each application) to determine which set of application-specific methods to invoke in response to external events (such as window messages arriving from end-users or packets arriving on communication ports).

As we consider all these features very appropriated for our work, we have decided to define and implement an integration framework to the DISCOVER, described next.

To construct the framework using the already implemented set of classes, we have considered three key principles: the use of independent (horizontal) and dependent (vertical) interfaces and the system dynamic structure. Most system-level designs are almost entirely vertical, making the systems noninteroperable and nonreusable.

Thus, providing a uniform way of composition aids the system interchangeability and interoperability. However, sets of classes are based almost completely on application domain analysis, which led to specialized application domain-specific interfaces (vertical interfaces). These interfaces also result from direct use of internal subsystem interfaces, proprietary product-dependent interfaces, and other sources. Vertical
interfaces have the disadvantage of being specialized to a single need, and therefore very specific to one implementation.

When each individual subsystem is considered to be part of a larger class of subsystems, it is possible to define interfaces that apply to the entire class of potential subsystem, not just one instance. These kinds of interfaces are called horizontal interfaces. Because the horizontal interfaces are not dependent upon internal implementation details, it is possible to make local changes to any subsystem without impacting the overall system. To implement the horizontal interfaces we have used a facade pattern, which provides a simplified interface to a complex system with many classes.

Vertical and horizontal interfaces define the static structure of the system. Metadata is self-descriptive information that defines the dynamic structure of the system. Metadata defers some design decisions until run-time, such as the object instantiation. The horizontal and vertical interfaces and the metadata define the Horizontal-Vertical-Metadata (HVM) design pattern.

The effectiveness of metadata is closely related to the prevalence of horizontal interfaces. The use of horizontal interfaces allows clients to integrate with a wider range of object implementations with less codification than the implementations that only support vertical interfaces. Vertical interfaces also have an important role in well-balanced HVM solutions. Vertical interfaces can capture specialized functionality and performance needs not accessible through generalized horizontal interfaces and metadata.

In order to implement the HVM pattern, we have proposed an XML-based language that represents the DM process itself [17]. This language, which stores the metadata information, acts as a translator (or interface) between the DM process user model and its real implementation within the system’s components at the integration framework level.

With such language we can represent many process steps, each one with its specificities, on an unique and uniform way. Such standardization is also an ideal way to establish the communication between the Graphical User Interface (GUI) and the framework modules or components.

On its turn, when is related on the relationship between the GUI and the framework, the framework may itself be seen as a facade pattern, underlying the complexity of the system interfaces. The framework takes a DM process represented in such language and generates a tree-based structure that will be interpreted by a Visitor pattern. One Visitor sequentially go through the tree root down to the leaves. Each node in the data structure “accepts” a Visitor, which sends a message to the Visitor which includes the node’s class. The Visitor will then execute the object for that element.

One key advantage of using the Visitor Pattern is that adding new operations to perform upon the data structure is very easy. All we have to do is create a new Visitor and define the operation there. For instance, if we want to implement a parallel or distributed interpretation instead of our sequential interpretation, we only need to implement a Visitor that could handle parallel or distributed interpretation. This is the result of the very distinct separation of variant and invariant behavior in the Visitor pattern. The data structure elements and an abstract Visitor represent the invariant behaviors. The variant behaviors are encapsulated in concrete Visitors. Figure 4 shows an overall picture of the DISCOVER system.

4.3 The Graphical User Interface

To compound the components the user uses the DISCOVER GUI [11]. It is a powerful, yet flexible, user-centered interface — a new approach for this kind of systems. This interface is being developed under rigorous usability engineering methods and theories, following the Usability Engineering Lifecicle [15].

This GUI allows the user to achieve a totally graphical manipulation of the components of an experiment, something that has only recently been used for DM, and we consider should be better explored.

In this GUI each component (e.g. datasets, inducers, data and knowledge processing, etc.) is represented as a group of possible instances that can be dragged and dropped into a working area and then instanced and linked altogether in order to set up an experiment.

Once a valid experiment has been graphically configurated by the user, the constructed process is then “translated” to the XML-based language we have defined [17], keeping all personal configurations of each step. Thus, the translated process can be saved on a file, allowing its later usage through the environment for further modification and/or execution.
5 Conclusions and Future Work

Even though there are some similar commercial and public Data Mining tools available, they do not gather in a unique tool all the characteristic and functionalities we need for our research, such as modularity, extensibility, deep knowledge of the underlying algorithms, flexibility, but also allowing the user a high level of control over all phases of the planning and execution of experiments.

We have tried to incorporate all of the above characteristics and functionalities on a single system called DISCOVER. Modularity, extensibility and other functionalities have been reached through a components integration framework, while a high level of control is given by a usability driven user centered GUI. Such level of control is possible due to the direct graphical manipulation of the components allowed by the GUI.

In order to the GUI and the framework communicate with each other, we have developed a DM process description language based on XML. This language works as a “translator” between the two modules, and also assures the standardization of the project and of the process itself.

We also consider that the fact of being a free software is another major characteristic of the DISCOVER. Ongoing work on DISCOVER includes implementations for association and regression rules as well as unsupervised Machine Learning.

References


