University of Liege
Stochastic Methods Department

ATDIDT User Guide
(versions 2.2 and 3.0)

- Draft 1.0 -

21 February 2002

Cristina Olaru

About this document

This document is confidential.

This guide intents to be a useful tool for those persons who are:

- Beginners in the Data Mining field and/or beginner users of ATDIDT software;
- Current users of ATDIDT of previous versions of the tool;
- Individuals interested in a particular method of the software;
- Peoples (organisms) interested in a software evaluation;
- People interested in trying out various data mining methods in the context of a demo loaded-in database (Tutorial version of the software).

The document covers all ATDIDT old and new functionalities, with an emphasis on operational aspects. For each data mining method practical aspects are distinguished: the proper sequence of actions to be done, parameters to choose, which are the “seen” and “unseen” effects of the method use, how to interpret the graphics/results, all completed by practical counsels. Some adjacent Lisp and Emacs tips are also included. Where possible, general features of all methods are grouped together, but still one may find redundant information in this guide, mainly for reasons of completeness at method level. The user interested in a certain Data Mining method may directly read the corresponding chapter, hyperlinks helping him to find in other chapters subsequent information if necessary.

The guide will be soon updated with a section dedicated to the user interface.

1 Introduction to the field

1.1 Data Mining

1.1.1 Definitions and objectives

'Data Mining' (DM) is a folkloric denomination of a complex activity, which aims at extracting synthesized and previously unknown information from large databases. It denotes also a multidisciplinary field of research and development of algorithms and software environments to support this activity in the context of real life problems, where often, huge amounts of data are available for mining. There is a lot of publicity in this field and also different ways to see the things. Hence, depending on the viewpoints, DM is sometimes considered as just a step in a broader overall process called Knowledge Discovery in Databases (KDD), or as a synonym of the latter as we do in this guide. Thus, according to this less purist definition DM software includes tools of automatic learning from data, such as machine learning and artificial neural networks, plus the traditional approaches to data analysis such as query-and-reporting, on-line analytical processing or relational calculus, so as to deliver the maximum benefit from data.

The general purpose of data mining is to process the information from the enormous stock of data we have or that we may generate, so as to develop better ways to handle data and support future decision-making. Sometimes, the patterns to be searched for, and the models to be extracted from data are subtle, and require complex calculus and/or significant specific domain knowledge. Or even worse, there are situations where one would like to search for patterns that humans are not well suited to find, even if they are good experts in the field. For example, in power systems related problems one is faced with high dimensional data sets that cannot be easily modeled and controlled on the whole, and therefore automatic methods capable of synthesizing structures from such data become a necessity.

By definition, data mining is the nontrivial process of extracting valid, previously unknown, comprehensible, and useful information from large databases and using it. It is an exploratory data analysis, trying to discover useful patterns in data that are not obvious to the data user. DM takes 2 forms: *verification driven data mining*, which extracts information in the process of validating a hypothesis postulated by a user, and *discovery-driven data mining*, which automatically extracts information novel for the user.

What is a database (DB)? It is a collection of objects (called tuples in the DB jargon, examples in machine learning, or transactions in some application fields), each one of which is described by a certain number of attributes, which provide detailed information about each object. Certain attributes are selected as input attributes for a problem, certain ones as outputs (i.e. the desired objective: a class, a continuous value…).

Usually, one of the first tasks of a data mining process consists of summarizing the information stored in the database, in order to understand well its content. This is done by means of statistical analysis or query-and-reporting techniques. Then more complex operations are involved such as to identify models that may be used to predict information

about future objects. The term “supervised learning” (known as “learning with a teacher”) is implied in mining data in which for each input of the learning objects, the desired output objective is known and implicated in learning. In “unsupervised learning” approaches (“learning by observation”) the output is not provided or not considered at all, and the method learns by itself only from input attribute values.

What is a data miner? - some person, usually with background in computer science or in statistics **and** in the domain of interest, or a couple of two specialists, one in data mining, one in the domain of interest, able to perform the steps of the data mining process. The miner is able to decide how much iterative to be the whole process and to interpret the visual information he gets at every sub-step.

1.1.2 The process

In general, the data mining process iterates through five basic steps:

- **Data selection.** This step consists of choosing the goal and the tools of the data mining process, identifying the data to be mined, then choosing appropriate input attributes and output information to represent the task.

- **Data transformation.** Transformation operations include organizing data in desired ways, converting one type of data to another (e.g. from symbolic to numerical) defining new attributes, reducing the dimensionality of the data, removing noise, “outliers”, normalizing, if appropriate, deciding strategies for handling missing data.

- **Data mining step per se.** The transformed data is subsequently mined, using one or more techniques to extract patterns of interest. The user can significantly aid the data mining method by correctly performing the preceding steps.

- **Result interpretation and validation.** For understanding the meaning of the synthesized knowledge and its range of validity, the data mining application tests its robustness, using established estimation methods and unseen data from the database. The extracted knowledge is also assessed (more subjectively) by comparing it with prior expertise in the application domain.

- **Incorporation of the discovered knowledge.** This consists of presenting the results to the decision maker who may check/resolve potential conflicts with previously believed or extracted knowledge and apply the new discovered patterns.

The whole data mining process is iterative, interactive, and very much a trial and error activity. DM techniques are different one form another in terms of problem representation, parameters to optimize, accuracy, complexity, run time, transparency, interpretability. The quality of the extracted knowledge is a function both of the effectiveness of the data mining techniques and the quality (often size) of the available database.

Visualization plays an important role. It may provide preliminary understanding of data, domain specific visualizations or can present the results of the mining techniques.

From the point of view of software structure, there are two types of possible implementations:

- Data mining "in place" (version 3.0 of ATDIDT): the learning system accesses the data through a data base management system (DBMS) and the user is able to interact with both the database (by means of queries) and the data mining tools. The advantage is that the approach may handle very large databases and may exploit the DBMS facilities (e.g. the handling of distributed data).
- Data mining "offline" (version 2.2 of ATDIDT): the objects are first loaded in the DM software, with a translation into a particular form, outside the database, and the user is interacting mainly with the data mining software. They may be faster but are generally limited to handle medium sized data sets that can be represented in main memory (up to several hundred Mbytes).

1.2 Main DM tasks

Depending mainly on the application domain and on the interest of the miner, one can identify several types of data mining tasks for which data mining offers possible answers. We present them in the order they are usually implied in the process.

Summarization. It aims at producing compact and characteristic descriptions for a given set of data. It can take multiple forms: numerical (simple descriptive statistical measures like means, standard deviations…), graphical forms (histograms, scatter plots…), or the form of "if-then" rules. It may provide descriptions about objects in the whole database or in selected subsets of it. *Example of summarization: "the minimum unitary price for all the transactions with energy is 70 price units".*

Clustering. A clustering problem is an unsupervised learning problem, which aims at finding in the data clusters of similar objects sharing a number of interesting properties. It may be used in data mining to evaluate similarities among data, to build a set of representative prototypes, to analyze correlations between attributes, or to automatically represent a data set by a small number of regions, preserving the topological properties of the original input space. *Example of a clustering result: "from the seller B point of view, buyers A and E are similar customers in terms of total price of the transactions done in 1998".*

Classification. A classification problem is a supervised learning problem where the output information is a discrete classification, i.e. given an object and its input attributes, the classification output is one of the possible mutually exclusive classes of the problem. The aim of the classification task is to discover some kind of relationship between the input attributes and the output class, so that the discovered knowledge can be used to predict the class of a new unknown object. *Example of a derived rule, which classifies sales made early in the day (a sale is said to be early if it was made between 6 a.m. and 12 a.m.): "if the product is energy then the sale is likely to be early".*

Regression. A regression problem is a supervised learning problem of building a more or less transparent model, where the output information is a continuous numerical value or a vector of such values rather than a discrete class. Then given an object, it is possible to predict one of its attributes by means of the other attributes, by using the built model. The prediction of numeric values may be done by classical or more advanced statistical methods and by "symbolic" methods often used in the classification task. *Example of a model derived in a regression problem: "when buyer A buys energy, there exists a linear dependence between the established unitary price and the quantity he buys:* $quantity = 170 - 1.5 * price$ *".*

Dependency modeling. A dependency modeling problem consists in discovering a model which describes significant dependencies among attributes. These dependencies are usually expressed as "if-then" rules in the form "if antecedent is true then consequent is true", where both the antecedent and the consequent of the rule may be any combination of attributes, rather than having the same output in the consequent like in the case of the classification rules. *Example: such a rule might be "if product is energy then transaction price is larger than 2000 price units".*

Deviation detection. This is the task focusing on discovering the most significant changes or deviations in the data between the actual content of the data and its expected content (previously measured) or normative values. It includes searching for temporal deviations (important changes in data with time), and searching for group deviations (unexpected differences between two subsets of data). *As an example, deviation detection could be used in order to find main differences between sales patterns in different periods of the year.*

Temporal problems. In certain applications it is useful to produce rules that take into account explicitly the role of time. There are data bases containing temporal information which may be exploited by searching for similar temporal patterns in data or learn to anticipate some abnormal situations in data. *Examples: "a customer buying energy will buy spinning reserve later on)", or "if total quantity of daily transactions is less than 100 price units during at least 1 month for a client, the client is likely to be lost".*

Causation modeling. This is a problem of discovering relationships of cause and effect among attributes. A causal rule of type "if-then" indicates not only that there is a correlation between the antecedent and the consequent of the rule, but also that the antecedent causes the consequent. *Example: "decreasing energy price will result in more sold energy daily".*

1.3 DM success factors

The success of mining some data is induced by a list of factors:

The right tools. A distinctive feature of a DM software is the quality of its algorithms, the effectiveness of the techniques, and sometimes their speed. In addition, the efficiency of using the hardware, the operating system, the database resources and the parallel computing are influencing the process. Moreover, it turns out that the particular set of tools useful in a given application are highly dependent on the practical problem. Thus at the prototyping step, it is useful to have available a broad enough set of techniques so as to identify interesting

applications. However, in the final product used for actual field implementation it is often possible to use only a small subset of the latter tools. Customizing data mining techniques to the application domain and using methods that are reliable means to the proposed goal may enhance the process of extracting useful information.

The right data. The data to be mined should contain information worth mining: consistent, cleaned, and representative for the application. Of course, it is useless to apply data mining to an invalid database with high measurement or estimation data errors, or to try to precisely estimate numerical outputs that present a priori high noise. A data mining tool ideally explains as much information as is stored in the data which is mined (a derived model is strongly dependent on the learning set used), and sometimes it is not what is in the data that matters for an application (wrong attributes, wrong selected sample).

An important part of data mining result errors are due to uncertainties in modeling and generation of objects in certain databases in discordance with the real probabilities of phenomena appearances in the system. That is why the data mining errors often do not have a meaning by themselves; they just provide a practical means to compare efficiencies of different criteria applied to the same database.

The right people. Regardless of what many producers of data mining tools claim, data mining is not (yet) an “automatic” operation with little or no human intervention. On the contrary, the human analyst plays an important role, mostly in the areas of data selection and data / knowledge interpretation. The data miner should have an understanding of the data under analysis and the domain or industry to which it pertains. It is more important for the mining process to embrace the problems of the application meant to solve, than to incorporate in the data mining software the hottest technologies.

The right application. Almost always a problem well posed is already a partially solved problem. It is important to clearly define the goals and choose the appropriate objectives so as to yield a significant impact on the underlying decision making process.

The right questions. An important issue: how does the data miner structure a data analysis problem so that the right question can be asked, knowing how easy and useless it is to give the right answer to the wrong question?

The right sense of uncertainty. Data miners are more interested in understandability than accuracy or predictability per se. Often, even the best methods of search will leave the data miner with a range of uncertainties about the correct model or the correct prediction.

2 Introduction to the software

2.1 About the software

ATDIDT software is a “Data Mining” software. It has been developed at the University of Liège for research, teaching and applications of automatic learning. The acronym stands for "Acl Top Down Induction of Decision Trees". The software is partly written in Allegro Common Lisp (ACL) and partly in GNU Emacs Lisp.

Requirements. For running ATDIDT software on your machine you need to have installed, besides GNU Emacs (version 19.29 or higher) and ACL, also the next auxiliary tools, freely distributed on the web: GUNZIP, GHOSTVIEW, XFIG (3.2 of higher), TRANSFIG (same version as XFIG), NETSCAPE and ACROBAT READER. Recent versions of Linux distributions contain all what is required.

Copyright. University of Liège owns the software. Only authorized people may use this software. Of course, if you use this software, you do it on your own responsibility.

2.2 Software organization

The software has three main parts:

Data Handling allows manipulating partially or entirely one or more databases, to prepare them off-line for loading, to load them every time one needs to explore them. Also data selection and data transformation steps of the data mining process are concerned here: a set of attributes as inputs, the task output variable, and a sample of objects (a part of the loaded objects) are selected, and new attributes are defined.

Graphics allows the visualization task: preliminary brute data visualization, customizable representation of objects, method results visualization.

Automatic Learning allows the interactive and iterative use of data mining methods. Some of these methods will produce a model, which expresses the relationships between the input attributes and the output variable. This model is added on-line to the database as a new functional attribute, which can be used in turn as input or output variable in subsequent steps of the data mining process.

Data mining process starts always with the data handling stage. Then any automatic learning method may be tried. Graphics part is needed at every intermediary stage of the process, be it data handling or automatic learning.

2.3 Available methods in ATDIDT 3.0

Table 1 synthesizes all the learning methods available in ATDIDT software. Following the commands organization within the software, we grouped them into six categories of methods. For each method the supported inputs and outputs are indicated, together with the type of the learning problem (task) it is able to accomplish.

Besides these methods, the software supports hybrid methods.

Table 1

CLASS OF METHODS	METHOD	INPUTS	OUTPUTS	DM LEARNING TASK
Graphical Tools	Dendrogram	non-constant numerical	--	Clustering
	Histogram	symbolic and numerical	--, symbolic or numerical	Summarization
	Cumulative Distribution	symbolic and numerical	--	Summarization
	Scatter -plot	symbolic and numerical	--, symbolic or numerical	Summarization
	Temporal Curves	temporal	--	Summarization
Tree Induction	Decision Tree	symbolic and numerical	symbolic	Classification
	Regression Tree	symbolic and numerical	numerical	Regression
Linear Regression	Linear Regression	non-constant numerical	numerical	Regression
	Linear Hinges Model[1]	non-constant numerical	numerical	Regression
Non Linear Regression	MLP	numerical	symbolic	Classification
	MLP	numerical	numerical	Regression
	Features Extraction	numerical	--	Clustering
Similarity	KNN	numerical	symbolic	Classification
	KNN	numerical	numerical	Regression
	K-Means	numerical	--	Clustering
Meta Learning	Regression Tree Bagging[2]	non-constant numerical	numerical	Regression
	Regression Tree Boosting	non-constant numerical	numerical	Regression

[1] Only in version 2.2 of ATDIDT
[2] Only in version 3.0 of ATDIDT

2.4 Get started. ATDIDT run and database load

Cautions. ATDIDT produces a lot of output files, temporary or permanent. The names of the files have been chosen so as to limit the risk of destroying other files. However, we recommend that you "create a new empty directory" and start the software while being in this directory, in order to make sure that none of your own files is "destroyed".

2.4.1 Start running ATDIDT

Once positioned in the directory of your choice (with rights to write on it) you tape in an `xterm` window one of the next commands:

- For **ATDIDT-2.2**:
  ```
  /sst5/soft/atdidt/bin/atdidt.script &
  ```
- For **ATDIDT-TUTORIAL-2.2**:
  ```
  /up1/lwh/gtdidt-tutorial/gtdidt-demo.script &
  ```
- For **ATDIDT-TUTORIAL-3.0**:
  ```
  /up1/lwh/gtdidt-tutorial/atdidt-tutorial.script &
  ```

where you change the path if different in your case.

2.4.2 Database load

The first thing to do in order to use the program is to load a database. Once the data is loaded, the data mining process is followed as in section 1.1.2. All the data mining techniques may be applied iteratively and independently one of each other.

ATDIDT-TUTORIAL-2.2

In the tutorial version, a database is already loaded into the program. The user skips therefore this database-loading step.

ATDIDT-2.2

This ATDIDT version is organized as an offline data mining software. The database is loaded once in the beginning and kept entirely in the main memory during the data mining process.

There are two possible ways of handling a database loading:

Long-way. It presumes that the user dispose of / create two types of files:

- *Data files* - files where explicit attribute values are stored (see example in appendix 8.1.1, long-way database load). These files are organized as tables where a column is associated with an attribute and a raw with an object. There are two possible versions

of these data files according to insertion of not of the object numbers. The user already disposes of these files when he starts the data mining process.

- *Database declaration file* – a lisp code file, providing information about what explicit attributes have to be loaded (name, short doc, type, default value (optional), possible values), in which format and what are their corresponding data files; the file may also define functional attributes. The file may load only a part of the available attributes or a part of the available objects provided in the data files. The user does not dispose of this file when he gets started. He has to create it. See in appendix 8.1.2 an example of a database declaration file. The possible attribute types may be found in section 3.2.1

This database declaration file is then loaded using the command `:LOAD_DB,` in menu `:DATA_BASE.`

Short-way. It presumes that the user dispose of / create only one type of file:

- *Data files* - files where explicit attribute values are stored (see example in appendix 8.1.1, short-way database load). They are called *javadb* files. These files are organized exactly as in the long-way case, plus the following adding:
  - The lines starting with a semicolon are comments and may appear anywhere in the file, except for the first line which is mandatory and must be exactly as in the provided example;
  - The first non-comment line must give the database name;
  - The second non-comment line contains attribute names immediately followed by their type: numerical (for ordered numerical attributes) or symbolic (for qualitative or ordered symbolic attributes);
  - Both versions 1 and 2 are possible, being not mandatory to add the object names, given that anyway if no name is specified, it will be computed on the fly based on the line number.

These data files are then loaded using the same command `:LOAD_DB,` in menu `:DATA_BASE.`

Once the database has been loaded using this short way, the user may type the following lisp command in the command line (lisp buffer):

```
(save-db)
```

The following two files are automatically created and located in the current work directory:

- Data file (database-name.att, e.g. OMIB.att) in the version 2 of short-way format
- Database declaration file (database-name.db. e.g. OMIB.db) as in the appendix example

At any new session, the database loading may be effectuated by the long-way loading of this new created database declaration file.

If the user defines new functional attributes or redefines/modifies old ones (as indicated in section 3.3) and adds them to the new created database declaration file, once re-loaded this file using the long-way loading, the new attributes will be also considered, at any new session.

Get data file in the right format. The user may not necessarily dispose of a data file in text format or *javadb* as we indicated in appendix. If data is stored in Microsoft Excel or Access format, the user has to save these files in "commas" format *data-file.cvs* and then type in an `xterm` window the command

`csv2jdb.pl –i` *`data-file.csv`* `–o` *`data-file.dat`*

in order to obtain the data in *javadb* format. *csv2jdb.pl* script is written in Pearl. It interprets the first raw of the Microsoft table as the attribute names. For every column, i.e. explicit attribute, the script asks a confirmation concerning the attribute type (numerical or symbolic) and makes a conversion type if necessary, from numerical to symbolic, by adding a prefix "S-" to numerical values.

ATDIDT-3.0

This ATDIDT version is organized as an in place data mining software. The database is accessed every time some data is necessary for a data mining manipulation. The loading process comports two stages:

- *Preparing the database.* The user disposes of a javadb data file format (called external format), applies the command `:PREPARE_DB` and gets internal format files, a compiled version of the database. This step is accomplished only once for a database, not for every new ATDIDT session. Once the internal format files obtained, the user do not need anymore the external format files. Among the generated files there is one called *DBNAME-project.xml* that represents the project description file.
- *Loading the database.* The user applies the command `:LOAD_PROJECT` in order to load the project description file *DBNAME-project.xml*. This step has to be done at each new session of the program. Presently, the software core does not manage more than one database per session and does not check for conflicts between attribute names.

In this way, the attribute values are progressively loaded in the main memory when they are needed for a data mining technique. This permits a faster manipulation of the data at the loading step. It permits to configure the memory space allocated to the data loading and to reduce the garbage collection waste of time. It allows the handling of larger and almost unlimited-size databases.

2.4.3 Restart ATDIDT

ATDIDT 2.2 command is:

- popup menu `GTDIDT`, submenu `RESTART`
- emacs menu `:RESTART.`

3 Data handling

3.1 Objects set selection

Once a database is loaded into the program, the user must select an objects subset of the loaded data, as being the current working set for the most data mining techniques and graphics. The global variable that contains the selected objects set on which the methods are trained or the graphics are drawn, is called `*learning-set*` (LS). The global variable that contains the selected objects set on which the methods are tested is called `*test-set*` (TS).

It is mandatory to a priori choose `*learning-set*` of objects before using one method. `*test-set*` must be defined only if the user needs to test its methods.

ATDIDT 2.2 commands are:

- popup menu :`Selections`, submenu `Select learning set` and `Select test set`, or
- emacs menu `:SUBSETS_SELECTION`, submenus `:LEARNING_SET` and `:TEST_SET`

Commands for learning and/or test set selection are found also in every submenu of a data mining method that uses the two variables. Both selections have the same syntax, even if they use different commands. Table 2 collects all the possible alternatives for this objects set selection. In the sequel, we will call LS the variable `*learning-set*` and TS the variable `*test-set*`.

3.2 Attributes selection

3.2.1 Types of attributes

- *Explicit attributes*: attributes values are specified explicitly for each object in some data file;
- *Functional attributes*: attributes values are computed from the values of other (explicit or functional) attributes. They are defined by the user (as lisp functions) or automatically generated by the learning methods.

3.2.2 Attributes values

- *Ordered:* numerical (integer or real valued e.g. *pu, qu, cct-sbs, pu+b*qu*) or symbolic (e.g. *cct-disk*)
- *Qualitative:* unordered symbolic (e.g. *security*)
- *Temporal*: numerical time series or sequences of events (e.g. *delta*)

- *Any lisp type:* a complex number, a scalar attribute function of a temporal one (e.g. *delta-after-fault*), etc.

Table 3 shows the correspondent type of each attribute in the database declaration file example of appendix 8.1.2.

Table 2

Command syntax	Selected objects	Examples
(first *n*) (first *n set*)	*n* first objects of the entire database *n* first objects of *set*	(first 100) (first 100 t) (first 50 *learning-set*)
(last *n)* (last *n set*)	*n* last objects of the entire database *n* last objects of *set*	(last 100) (last 100 t) (last 50 *test-set*)
(random *n)* (random *n set*)	*n* objects selected randomly from the entire database *n* objects selected randomly from *set*	(random 50) (random 500 *learning-set*)
(not-in *set*)	all the objects which are not in *set*	(not-in *learning-set*)
(member *o1 o2* …)	objects *o1, o2* …	(member op1 op250 op12)
(such-that *att cond set*)	objects from *set* for which *att* respects the condition *cond*	(suchthat pu (float 1000.0 1100.0) t) (suchthat security (member secure) *learning-set*)
(from *n1 n2*)	objects from *n1* to *n2*	(from 5001 6000)
(union *set1 set2*)	union of objects from *set1* and *set2*	(union (first 10) (last 10)) (union *learning-set* *test-set*)
(atnode *node* of *dt* in *set*)	objects from *set* which would go to node *node* of *dt* tree	(atnode “L1” of “DT1” in t)
set	objects from *set*	*learning-set* *pruning-set* *test-set* *validation-set* *last-selected-subset* *classification-errors* *knn-reference-set* t

3.2.3 Inputs selection

It is mandatory to a priori choose the list of attribute inputs before using the majority of methods. The global variable that contains the selected attributes considered as inputs is called `*candidate-attributes*`.

ATDIDT 2.2 commands are:
- popup menu `Atts`, submenu `Select Attributes`, or
- emacs menu `:ATTRIBUTES_SELECTION`, submenu `:CANDIDATE_ATTRIBUTES`

With these commands, all the available attributes are displayed and any attribute may be added or deleted from the list of inputs. Command for candidate attributes selection is found also in every submenu of a data mining method that uses this variable.

If there exists predefined lists of attributes, the user may use the command `:ATTRIBUTES_CHOICE` from menu `:ATTRIBUTES_SELECTION` in order to directly select as inputs one predefined such list or to merge the current list of inputs with a predefined one.

When applying the majority of data mining techniques, the list of `*candidate-attributes*` is expanded by replacing temporal attributes by a list of scalar ones and filtered to remove attributes which type is not handled by the technique.

Table 3

Attribute type	Database declaration type
ordered	ordonne linear-combination
qualitative	qualitatif-quinlan
temporal	(ordonne time)
lisp type	ordonne etc.

3.2.4 Output selection

It is mandatory to a priori choose the output before using the majority of methods.
The global variable `*goal-classification*` indicates the output for methods used in **classification** task and must be a symbolic type of attribute. The global variable `*goal-regression*` indicates the output for methods used in **regression** task and must be of numerical type. These goals are chosen among the available attributes in the loaded database.

ATDIDT 2.2 commands are:
- popup menu `Atts`, submenu `Select goal classification`, or
- emacs menu `:ATTRIBUTES_SELECTION`, submenus `:GOAL_CLASSIFICATION` and `:GOAL_REGRESSION`

Commands for regression/classification goal selection are found also in every submenu of a data mining method that uses these variables.

3.3 Define new attributes

New functional attributes may be defined/modified in three ways. Once defined/modified a new functional attribute, its definition may be stored in a lisp file *file.lsp* and this file loaded at any new session, or anytime during the current session, but always after the explicit attributes loading, by using one of the commands:

- (load "*file.lsp*") or
- popup menu `Load`, submenu `Load a lisp file`, or
- emacs menu `:DATA_BASE`, submenu `:LOAD_DB`, or
- (compile "*file.lsp*") together with (load "*file.fsl*") when the user wants to load the compiled version of the file

In this way, only the new defined/modified attributes have to be loaded, not all the database declaration file, fact which for large databases saves a lot of loading time.

1. Using the lisp macro def-fun-att

Example: (def-fun-att mva (object) (sqrt (+ (sqr (pu object)) (sqr (qu object)))))
See the effect with: (mva *object-name*)

2. Using an ATDIDT command

ATDIDT 2.2 command is:

- emacs menu `:ATTRIBUTES_SELECTION`, submenus `:DEFINE_ATTRIBUTE`

This command allows the user to define new attributes by using def-fun-att macro as in the above example. The definition of the new attribute is valid during the current session, and it is lost for future sessions.

3. Using the lisp macro declare-function-attributes

Example:

```
(declare-function-attributes
 OMIB
 :attributs-conserves t
 :attributs-fonctions-scalaires
 ((mva "Apparent power [MVA]"
      :type ordonee
      :valeurs (real * *)
      :fonction (sqrt (+ (sqr (pu objet)) (sqr (qu objet)))))
  (symbolic-pu "Pu>1000MW"
               :type qualitatif-quinlan
               :valeurs (member <1000 >=1000)
               :par-defaut <1000
               :fonction (if (< (pu objet) 1000.0) '<1000 '>=1000))))
```

See the effect with: (mva *object-name*)
(symbolic-pu *object-name*)

The slot `:par-defaut` is optional. The slot `:attributs-conserves` indicates the list of the newly defined attributes that will be loaded: `t` value means all the defined attributes, `nil` means none of them.

In order to visualize the function of a functional attribute *attribute-name*, at any moment the user may tape in the lisp buffer the lisp command:

(print (get *attribute-name* 'fonction))

3.4 Handling temporal attributes

The global variable `*candidate-attributes*` is specifying a list of scalar attributes. Thus, temporal attributes must be transformed into scalar ones according to a sampling strategy. This is done automatically by the software each time the `*candidate-attribute*` list is used in some option. Table 4 presents the scalar attributes created when the `*candidate-attributes*` list is activated in a command, for the temporal attribute *delta*.

Table 4

Temporal attribute in *candidate-attributes*	Function of the corresponding created scalar attribute
delta	(delta < *object*> *present-time*)
(delta 0.3)	(delta <*object*> 0.3)
(delta (time 0.0 0.2 nbsteps 5))	((delta 0.0)(delta 0.04)…(delta 0.2))
(delta (time 0.0 0.2))	((delta 0.0)(delta 0.2))
(delta (time))	(delta (time to tf *time-steps*))

ATDIDT parameters that control the temporal attributes handling are:

present-time
- Default value 0.0
- ATDIDT 2.2 command: menu :DATA_BASE, menu :ATTRIBUTES_SELECTION, command :PRESENT_TIME

time-steps
- Default value 50
- ATDIDT 2.2 command: menu :DATA_BASE, menu :ATTRIBUTES_SELECTION, command :TIME_STEPS

4 Software DM methods

4.1 Graphical Tools

All the graphical tools available in ATDIDT software use a postscript curve-drawing program called GDC (version 4.3).
All the graphics are computed on and representing the objects found in the current `*learning-set*`, and when necessary, are based on current `*goal-classification*`. When statistics are displayed below a graphic they are as follows:

- Mu – average
- Mn – minimum
- Mx – maximum
- Sd – standard deviation
- Rho – correlation coefficient between abscissa and ordinate attributes

4.1.1 Histogram

Definition. The histogram is a statistical tool that performs non-parametric density estimation. It is a frequency diagram. The 2D graphic displays the estimated number of objects for each interval of values of the chosen attribute (interval-region called bar).
Selections.

- Choose `*learning-set*` (LS)

Set parameters. ATDIDT 2.2 command: menu :DATA_BASE, menu `:GRAPHICS`, command :NUMBER_OF_BARS changes the number of intervals (bars) in which the attribute is split for computing frequencies (40 by default).
Command. ATDIDT 2.2 command: menu :DATA_BASE, menu `:GRAPHICS`, command :HISTOGRAM
The command prompts for:

- a numerical (see Figure 1) or symbolic (see Figure 2) *att* attribute

Effect.

- The created graphic is computed on LS for attribute *att*
- A postscript graphic is generated named *abs_freq-tem.ps* located in the current directory
- Statistics on LS for attribute *att* are displayed below the graphic.

4.1.2 Conditional histogram

Definition. It is a histogram colored according to the value of the `*goal-classification*`.
Selections.

- Choose `*learning-set*` (LS)
- Choose `*goal-classification*` (symbolic or numerical attribute)

Note: use lisp instruction (setf *goal-classification* '*numerical-goal-name*) for setting the goal as a numerical attribute

Set parameters. ATDIDT 2.2 command: menu :DATA_BASE, menu :GRAPHICS, command :NUMBER_OF_BARS changes the number of intervals (bars) in which the attribute is split for computing frequencies (40 by default).

Command. ATDIDT 2.2 command: menu :DATA_BASE, menu :GRAPHICS, command :COND_HISTOGRAM

The command prompts for:

- a numerical attribute (see Figure 3 for symbolic goal, see Figure 5 for numerical goal) or a symbolic attribute (see Figure 2 for symbolic goal, see Figure 4 for numerical goal) *att*

Effect.

- The created graphic is computed on LS for attribute *att* and is conditioned by *goal-classification* variable. If the goal is numerical, it is automatically split in classes of values and the graphic represents each class in a different color.
- A postscript graphic is generated named *cond_freq-tem.ps* located in the current directory
- Statistics on LS for *att* attribute are displayed below the graphic for each of the symbolic values of the *goal-classification* if the goal is symbolic, or for each of the generated classes, if the goal is numerical.

4.1.3 Multiple conditional histograms

Selections.

- Choose *learning-set* (LS)
- Choose *goal-classification* (symbolic or numerical attribute)
- Choose *candidate-attributes*

Note: use lisp instruction (setf *goal-classification* '*numerical-goal-name*) for setting the goal as a numerical attribute

Set parameters. ATDIDT 2.2 command: menu :DATA_BASE, menu :GRAPHICS, command :NUMBER_OF_BARS changes the number of intervals (bars) in which the attribute is split for computing frequencies (40 by default).

Command. ATDIDT 2.2 command: menu :DATA_BASE, menu :GRAPHICS, command :DB_STATS

Effect.

- For each attribute from *candidate-attributes*, a conditional histogram is built
- The created graphics are computed on LS and based on *goal-classification* variable
- A postscript graphic is generated named *db_stats-tem.ps* located in the current directory
- Statistics on LS for each attribute are displayed below the graphics, for each of the symbolic values of the *goal-classification* if the goal is symbolic, or for each of the generated classes, if the goal is numerical
- For a very large *candidate-attributes* list, this option will take some time.

4.1.4 Scatter-plot

Definition. It is a 2D graphic representing one attribute *yy* function of another attribute *xx*. Attributes *xx* and *yy* may be symbolic or numerical.
Selections.
- Choose **learning-set** (LS)

Command. ATDIDT 2.2 command: menu :DATA_BASE, menu :GRAPHICS, command :SCATTER_PLOT
The command prompts for:
- a numerical or symbolic attribute *xx*
- a numerical or symbolic attribute *yy*

Effect.
- A graphic (*xx*, *yy*) computed on LS is displayed (see Figure 6)
- A postscript graphic is generated named *correl-tem.ps* located in the current directory
- Statistics on LS for *xx* and *yy* attributes are displayed below the graphic.

4.1.5 Conditional scatter-plot

Definition. It is a scatter-plot colored according to the value of the **goal-classification**.
Selections.
- Choose **learning-set** (LS)
- Choose **goal-classification** (symbolic or numerical attribute)

Note: use lisp instruction (setf *goal-classification* '*numerical-goal-name*) for setting the goal as a numerical attribute
Command. ATDIDT 2.2 command: menu :DATA_BASE, menu :GRAPHICS, command :COND_SCATTER_PLOT
The command prompts for:
- a numerical or symbolic attribute *xx*
- a numerical (see Figure 7) or symbolic (see Figure 8) attribute *yy*

Effect.
- A graphic (*xx*, *yy*) computed on LS and conditioned by **goal-classification** variable is displayed. If the goal is numerical, it is automatically split in classes of values and the graphic represents each class in a different color.
- A postscript graphic is generated named *cond_correl-tem.ps* located in the current directory
- Statistics on LS for *xx* and *yy* attributes are displayed below the graphic for each of the symbolic values of the **goal-classification** if the goal is symbolic, or for each of the generated classes, if the goal is numerical.

4.1.6 Colored scatter-plot

Definition. It is a scatter-plot colored according to the value of a third attribute.
Selections.
- Choose *learning-set* (LS)

Command. ATDIDT 2.2 command: menu :DATA_BASE, menu :GRAPHICS, command :COLOUR_SCATTER_PLOT
The command prompts for:
- a numerical or symbolic attribute *xx*
- a numerical or symbolic attribute *yy*
- a numerical (see Figure 9 and Figure 10) or symbolic (see Figure 7) attribute *zz*

Effect.
- A graphic (*xx, yy*) computed on LS and conditioned by attribute *zz* is displayed. If *zz* is numerical, it is automatically split in classes of values and the graphic represents each class in a different color.
- A postscript graphic is generated named *colour_correl-tem.ps* located in the current directory
- Statistics on LS for *xx*, *yy* and *zz* attributes are displayed below the graphic.

4.1.7 Scatter-plot with values

Definition. It is a scatter-plot where each point is market on the graphic by the value of a third attribute.
Selections.
- Choose *learning-set* (LS) – a small one, for example 100 objects

Command. ATDIDT 2.2 command: menu :DATA_BASE, menu :GRAPHICS, command :SCATTER_PLOT_VAL
The command prompts for:
- a numerical attribute *xx*
- a numerical attribute *yy*
- a numerical or symbolic attribute *zz*

Effect.
- A graphic (*xx, yy*) computed on LS is displayed where values of *zz* (if numerical) or classes of *zz* (if symbolic) are market on the graphic for each point (object) of LS
- A postscript graphic is generated named *val_correl-tem.ps* located in the current directory
- Statistics on LS for *xx* and *yy* attributes are displayed below the graphic.

4.1.8 Cumulative distribution

Definition. The cumulative distribution is a statistical tool that performs a cumulative frequency diagram (the integral of the histogram). The 2D graphic displays points *(x,y)* where *x* represents the percentage of objects for which attribute value is at more *y*.
Selections.
- Choose *learning-set* (LS)

Command. ATDIDT 2.2 command: menu :DATA_BASE, menu :GRAPHICS, command :CUMULATIVE_DIST
The command prompts for:
- a numerical (see Figure 11) or symbolic (see Figure 12) *att* attribute

Effect.
- The created graphic is computed on LS for attribute *att*
- A postscript graphic is generated named *cum_freq-tem.ps* located in the current directory
- Statistics on LS for attribute *att* are displayed below the graphic
- Percentages corresponding to ordinate line are market in yellow on the graphic.

4.1.9 Dendrogram

Definition. The dendrogram is the graphical representation of a statistical tool called hierarchical agglomerative clustering. It is used to cluster attributes, the similarity between two subsets of attributes being defined as the minimum similarity of pairs of attributes of the two subsets. This tool is particularly interesting for the analysis of attribute similarities, detecting and eliminating the attributes too correlated, or detecting important correlation sbetween a goal attribute and input attributes.
Selections.
- Choose *learning-set* (LS)
- Choose *candidate-attributes*. Make sure you insert in the list all the attributes, including regression goal. Constant numerical and non-numerical attributes are not handled and excluded from the attributes list prior to dendrogram building.

Command. ATDIDT 2.2 command: menu :DATA_BASE, menu :GRAPHICS, command :DENDROGRAMS
Effect.
- The dendrogram is computed on LS
- The coefficients displayed on the graphic represent the minimum correlation coefficients between one attribute and a group of attributes or between two groups of attributes. Correlations with a coefficient more that 0.5 are depicted in red.
- A postscript graphic is generated named *dendrogram.ps* located in the current directory
- Statistics on LS for each attribute are displayed together with correlation coefficients for every two-by-two pairs of attributes.

4.1.10 Temporal curves

Definition. They represent evolution in time of temporal attributes for a given object.
Selections.
- Choose *learning-set* (LS)
- Choose *candidate-attributes*. Insert all the temporal attributes for which you wish a curve on the same graphic.

Command. ATDIDT 2.2 command: menu :DATA_BASE, menu :GRAPHICS, command :TEMPORAL_CURVES_O

The command prompts for:

- An object name *obj*

Effect.

- A curve for each temporal attribute in the list of `*candidate-attributes*` is represented on the same graphic for object *obj* (see Figure 13)
- A postscript graphic is generated named *scenario-tem.ps* located in the current directory.

4.1.11 Temporal curves for a set of objects

Definition. They represent evolution in time of one temporal attribute for a set of objects.

Selections.

- Choose `*learning-set*` (LS) – a small one, for example 50 objects

Command. ATDIDT 2.2 command: menu :DATA_BASE, menu :GRAPHICS, command :TEMPORAL_CURVES_SET

The command prompts for:

- A temporal attribute *att*

Effect.

- A curve for each object in `*learning-set*` is represented on the same graphic for attribute *att* (see Figure 14)
- A postscript graphic is generated named *scenarios-tem.ps* located in the current directory.

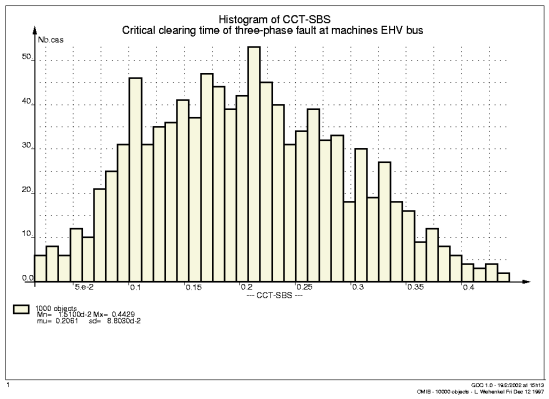


Figure 1

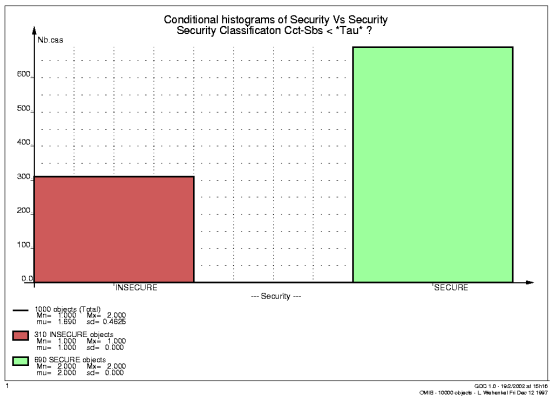


Figure 2

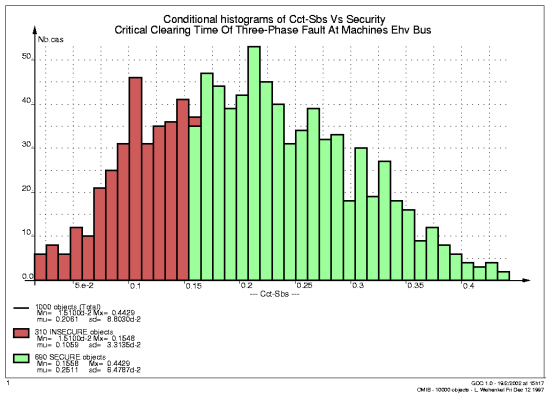


Figure 3

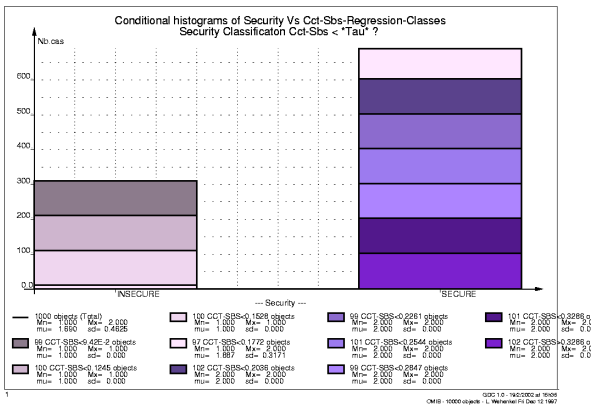


Figure 4

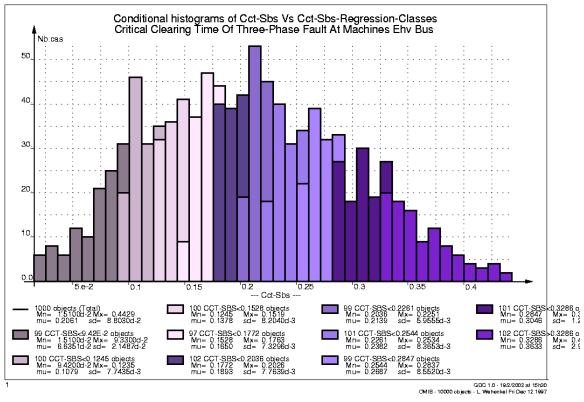


Figure 5

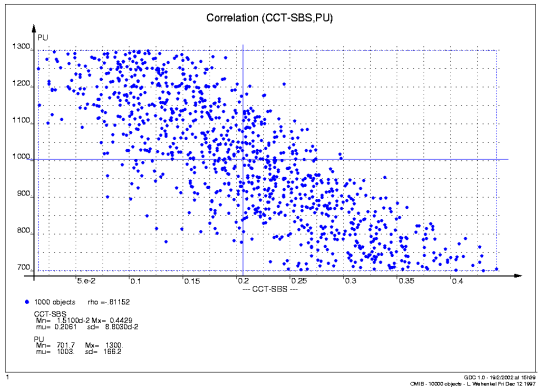


Figure 6

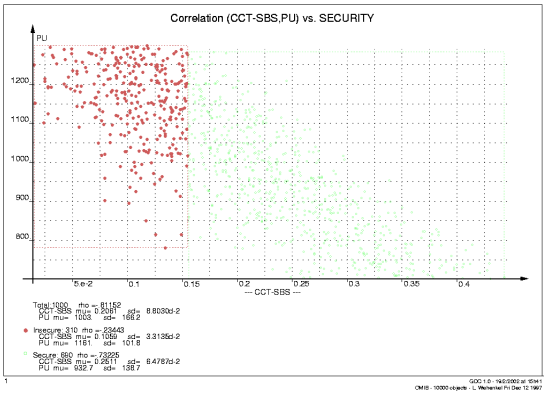


Figure 7

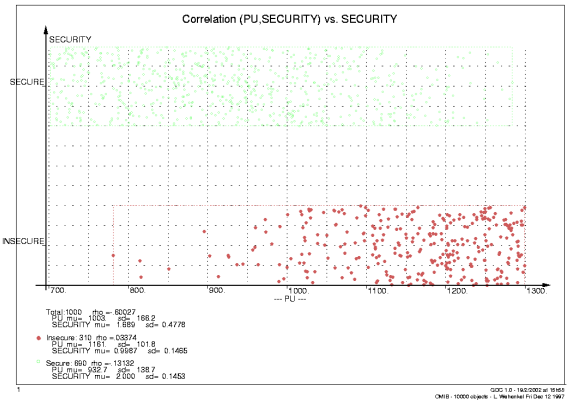


Figure 8

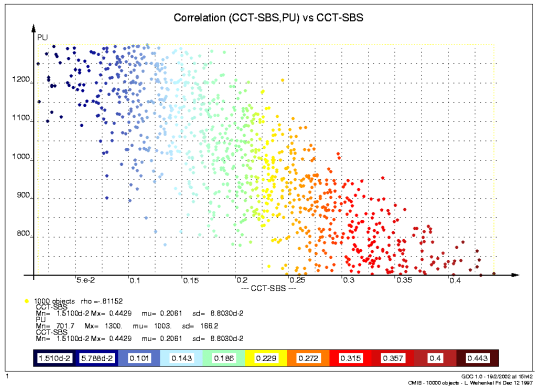


Figure 9

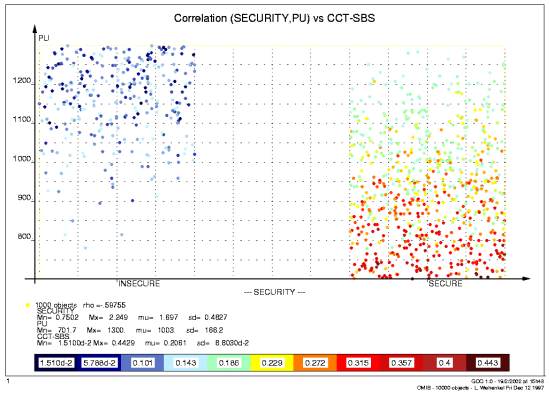


Figure 10

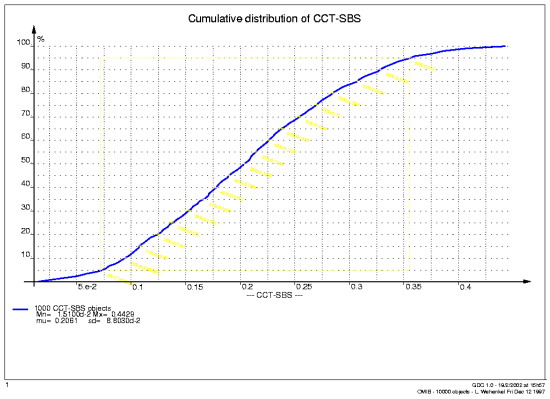


Figure 11

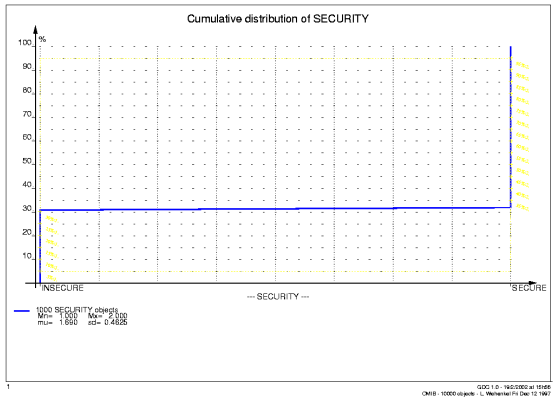


Figure 12

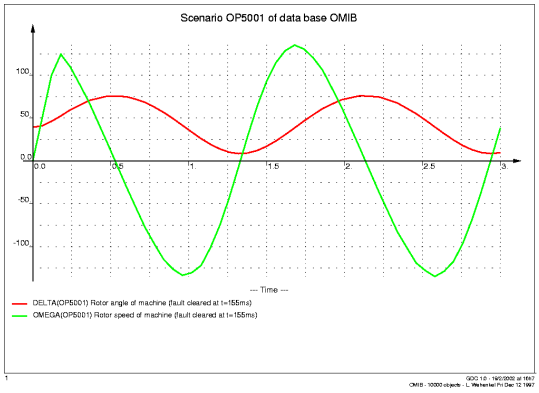


Figure 13

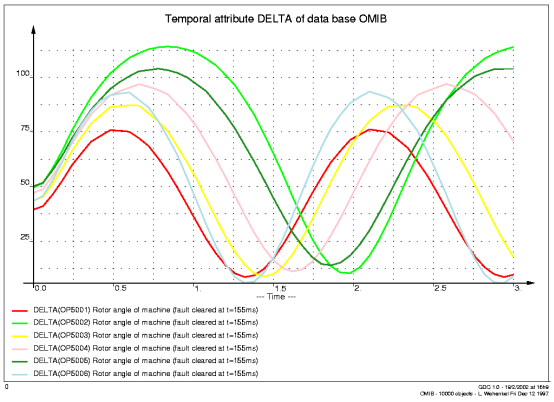


Figure 14

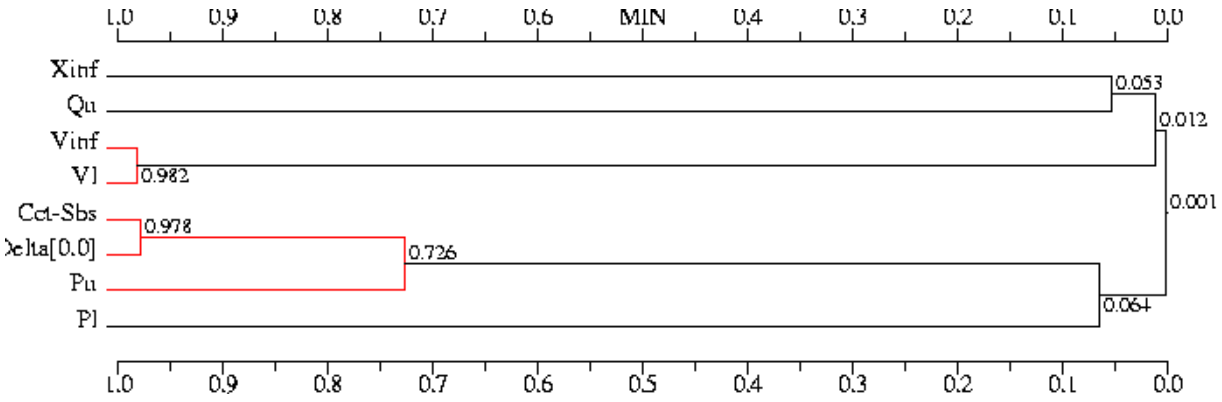


Figure 15

4.2 Decision Tree

4.2.1 What is it?

Definition. Decision trees (DT) are tools used in classification problems. They are concerned with the automatic design of if-then rules. They have a symbolic output and symbolic and/or numerical inputs.

Method characteristics. The main strength of DT is its interpretability. Another asset is the ability to identify the most relevant attributes for a problem: the model itself selects a part of the attributes from the list of candidate attributes as the model inputs. Finally, it is a computationally efficient tool. As a counter part, it is less accurate than a neural network. It may be used in association with a neural network or a KNN method in a hybrid approach.

4.2.2 Selections to make before starting

Define the problem. Choose `*goal-classification*` and choose `*candidate attributes*`. Admissible input attribute types are: “ordonee”, “linear-combination” and “qualitatif-quinlan”. Note that the model does not handle temporal attribute values, they being replaced by a list of scalar ones in `*candidate-attributes*` list prior to model building.
Select data. Choose `*learning-set*` and `*test-set*` (if you also want to test or prune the model).
Set method parameters. Method parameters are `*alfa*` and `*h-min*`.

alfa

- Necessary to detect deadens in decision trees, i.e. impure terminal nodes
- Takes values between 0.00005 and 1.0 (complete tree, maximum complexity)
- Accepted values: 1.0 0.25 0.1 0.05 0.025 0.01 0.005 0.001 0.0005 0.00025 0.0001 0.00005
- Default value 0.0001
- ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :TREE_INDUCTION, command :SET_*ALFA*

h-min

- Necessary to detect leaves in decision trees, i.e. pure terminal nodes
- Takes any real value between 0.0 (trivial tree, 0 complexity, 0 test nodes, 1 terminal node) and 10.0 (fully grown tree)
- Default value 0.028
- ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :TREE_INDUCTION, command :SET_*H-MIN*

4.2.3 Apply the method

Command. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :TREE_INDUCTION, command :BUILD_DT

Effects of the method employment. If the decision tree name is *xx* (the default name is *DT<i>*) and the *goal-classification* is *yy*:

- The new functional attribute created by default once the model is built has the name *approx-xx-of-yy*
- The created file containing information about the building process has the name *xx.log* and is located in the current directory
- The new decision tree *xx* is pushed in the global variable *decision-trees*
- The global variable *current-dt* keeps the last built (decision or regression) tree.

Interesting displayed information while building the tree: status variables, prior class probabilities in LS and at every node, LS size at every node, type of node, entropy of node, possible tests, their scores (the best score is 1.0, the worst one is 0.0) and the scores' standard deviations, chosen test, the correlation coefficient of each attribute's optimal test with the optimal test of the selected attribute, CPU time.

Interesting displayed information while describing the results: a résumé of used parameters and settings, total entropy of DT in LS (= total entropy of root node), DT complexity (number of test, leaf and deadend nodes), the percentage of the total information explained by the tree by every chosen attribute (a measure of the relevance of every attribute in the model), the name of the new created functional attribute, the name and the path of the created file containing the displayed information. All these information may be redisplayed anytime by using the command :DESCRIBE_TREE from :TREE_INDUCTION menu.

4.2.4 Test the model

Command. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :TREE_INDUCTION, command :TEST_TREE

The program detects when TS and LS are overlapping and asks the user if he wants to eliminate this overlap objects or not from TS before performing the testing.

Interesting displayed information while describing the results: non-detection costs (values between 0 and 1), confusion matrix on TS (number of objects correctly classified and misclassified), classification error rate on TS, CPU time.

Effects of the method employment. After testing, the global variable *classification-errors* contains all the misclassified objects.

4.2.5 Improve the model

Once build and tested, a tree may be pruned in order to improve the model's compromise between accuracy and complexity. The new resulted tree has less complexity than the original tree and better or comparative error rate. The pruning procedure generates a sequence of intermediary trees and based on these trees' error rates (computed on TS) the best tree is chosen following the n-standard-error-rule, i.e. the less complex tree not significantly less reliable than the best one is selected.

Note that the pruning procedure, in order to be effective, should be applied on a complete tree, i.e. a fully-grown tree. In this respect, the model's parameters should be chosen in consequence: *alfa*=1.0 (and *h-min*=0.0 evenatually).

To do before starting. For pruning a tree the user must before test the tree. The errors of the intermediary trees are computed on the global variable `*test-set*.` In order to use the cross-validation approach, the user should settle variable `*test-set*` as a set independent of LS and TS and test the original tree (on this set called usually pruning set) before pruning it.

Set method parameters. Method parameters are `*sigma-multiplier*` and `*maximum-tree-prune-complexity*.`

sigma-multiplier

- Necessary in n-standard-error-rule.
- Usual values: 0.0, 1.0, 2.0, 3.0, …
- Default value 1.0
- Examples: if `*sigma-multiplier*`=2.0, the procedure chooses the tree with the error smaller or equal to the best error plus two times its standard deviation; if `*sigma-multiplier*`=0.0, the procedure chooses the tree with the smallest error
- Example of lisp command: (setf `*sigma-multiplier*` 2.0)

maximum-tree-prune-complexity

- Settles the maximum complexity of the pruned tree
- Takes integer values between 0 and 10.000
- Default value 10.000
- Settled as a very large value this parameter has no influence on the process
- ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :TREE_INDUCTION, command :SET_*C-MAX-PRUNE*

Command. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :TREE_INDUCTION, command :PRUNE_TREE

Interesting displayed information while pruning the tree: some information concerning every intermediary tree (complexity, terminal nodes, the next node to prune, test set error rate, corresponding *alfa* parameter), information about the chosen tree.

Effects of the method employment. If the original decision tree name is *xx*, the `*goal-classification*` is *yy* and `*sigma-multiplier*` is 1.0:

- The pruned decision tree has the name *xx-BPR-1.0*

- The new functional attribute created by default once the tree is pruned has the name *approx-xx-BPR-1.0-of-yy*
- The created file containing information about the pruning process has the name *xx-BPR-1.0.log* and is located in the current directory
- The new decision tree *xx-BPR-1.0* is pushed in the global variable `*decision-trees*`
- The global variable `*current-dt*` keeps the pruned tree *xx-BPR-1.0*

ATDIDT 2.2 command `:DRW_PR_SEQ` provides a graphic of pruning sequence curves displaying the evolution of decision trees' complexity, information, test error rate and quality with parameter `*alfa*.` Two files named *xx-BPR-1.0.pruning* and *xx-BPR-1.0-pruning-seq.ps* are created in the current directory. The postscript one contains these graphics that may be visualized at any time by using GhostView tool.

4.2.6 Results visualization / interpretation

Describe tree. ATDIDT 2.2 command `:DESCRIBE_TREE` displays a résumé of the current decision tree growing and testing results (if the tree has been tested before). By current tree we understand the tree indicated by the global variable `*current-dt*,` i.e. the last built tree, or the last pruned tree, or the last tree chosen with the command :`CHOOSE_TREE`. The command may be applied at any time, once a (decision or regression) tree is stored in the variable `*current-dt*.`

Display tree. ATDIDT 2.2 command `:DISPLAY_TREE` displays the current tree on a single page. Command `:MY_DISPLAY_TREE` displays the tree on multiple pages, on the first page only the upper levels of the tree and on the other pages, the rest of the subtrees. It should be used for very complex trees (too complex to be displayed on a single sheet). Both commands generate a postscript file (named *xx.ps* for a DT called *xx*) located in the current directory that may be visualized at any time using the GhostView tool.

ATDIDT 2.2 command `:DRAW_TEST_SET` enables or disables the representation of the test results on the tree graphic. Figure 16 presents an example of a decision tree display without test results, and Figure 17, with test results.

Figure 16 and Figure 17 draw a decision tree for a `*goal-classification*` called "security" (see the attribute definition in database declaration file example of appendix), built on a learning set of 1000 objects, pruned and tested on a independent test set of 1000 objects. Figure 16 presents every node of the tree by a box area proportional to the size of the learning subset corresponding to this node (the exact size of this subset together with the name of the node are indicated above the box) and the horizontal division of each box shows the proportion of the objects from this subset in each class. In Figure 17, each node box is divided into two parts, the upper one corresponding to the learning set, the lower one to the test set. The part corresponding to the test set is horizontally divided indicating the proportion of misclassified objects in each local test set. In both figures, the test of each test node is written under the node's box and each arc leading to a successor is labeled with a possible answer to

this test (Yes and No). Above the root node, the total number of test nodes (Txx), leaves (Lxx), and deadends (Dxx) is indicated.

HTML format. ATDIDT 2.2 commands `:SAVE_TREE` and `:INSPECT_TREE` give another way of visualizing results, in html format. For a decision tree named *xx*, the first command creates a new directory called */Sav/xx/* in the current directory, and puts 7 files concerning the tree in this new directory. The second command opens a *Welcome.html* file that displays general information about the tree together with hyperlinks for all the created files:

- *xx-rules.html* – displays the IF-THEN rule base derived from the tree
- *xx-prune.lst* (for a pruned tree) or *xx-grown.lst* (for the original tree) – displays information that describe the pruning / growing processes
- *xx.dump* – outputs the internal lisp structure of the tree *xx*
- *xx.lsp* – contains the lisp function of the new created functional attribute
- *xx-mp.pdf* and *xx-sg.pdf* – are single page and multiple page displays of the tree.

Afterwards, at every new session, the ATDIDT 2.2 command `:LOAD_TREE` may load this built tree (model) based on the *xx.dump* file, thus releasing the user from building it again.

Derived rule-base. For every terminal node of a decision tree, an IF-THEN rule is generated. The file *xx-rules.html* indicates for every rule of type "if antecedent then class A", extracted from the *xx* decision tree, the next coefficients:

- support of rule – percentage of all objects in LS for which this rule is active
- cover of rule - percentage of all objects of class A in the LS for which this rule is active (the total of the covers for all the rules concluding a given class is 100%)
- certainty factor – percentage of objects of class A among those for which the rule is active
- summary – number of objects for which the rule is active counted by class.

Example of rule deducted from decision tree of Figure 16:

Rule **T3**: IF Pu > 1096.4 and Qu < 392.11 THEN class = INSECURE
Support = 22%
Cover = 63.2%
Certainty factor = 89.1%
Summary: insecure – 196, secure – 24.

Other ideas for graphics. If the decision tree name is *xx* and the `*goal-classification*` is *yy*:

- Conditional scatter-plot (*approx-xx-of-yy*, *yy*) on LS (see Figure 18) or TS
- Conditional histogram for *approx-xx-of-yy* on LS (see Figure 19), TS
- Settle LS as the objects misclassified by the tree (see command `:GET_DT-ERRORS`) and apply a conditional histogram for *approx-xx-of-yy*

4.2.7 Other possible actions

All the commands having the format :XXX_TREE are available both for regression and decision trees and are regarding the tree stored in the global variable *current-dt*.

Other useful available commands:

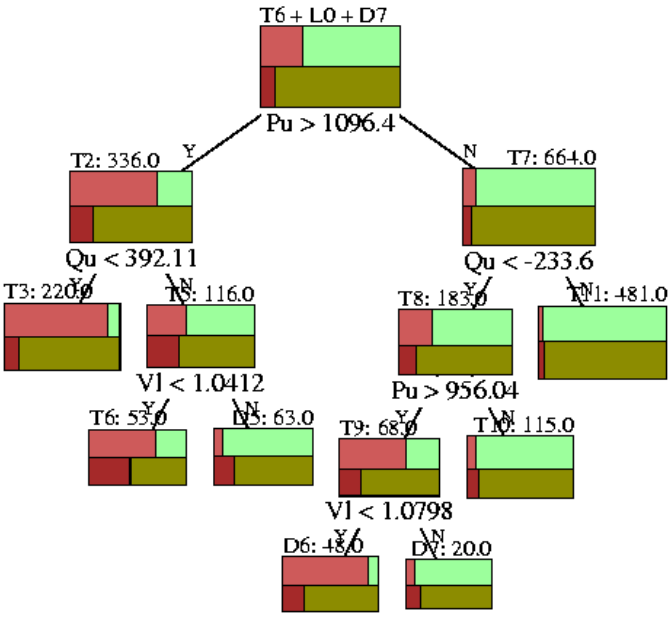
:BEST_FIRST? and :SET_*C-MAX-GROW* - concern the node development in a decision tree growing. :BEST_FIRST? allows to change the order of node development, either best first or depth first (default mode). In the case of best first strategy, command :SET_*C-MAX-GROW* fixes the global variable *maximum-tree-grow-complexity*, an upper bound on complexity, that takes integer values between 0 and 10.000, default value is 10.000.

:SELECT-DT-TEST-ATTS – settles the global variable *candidate-attributes* as the list of all the attributes chosen by the current decision tree. This command becomes very useful when decision tree technique is used in a hybrid approach together with other methods. A decision tree has the ability to reduce the input space to the relevant attributes for a given problem.

:GET_DT_ERRORS – select the global variable *learning-set* as the objects from *test-set* misclassified by the tree in the last testing and stored in the global variable *classification-errors*. If the tree has not yet been tested, the *learning-set* is settled to the empty set.


T6 + L0 + D7
Pu > 1096.4
Y
N
T2: 336.0
T7: 664.0
Qu < 392.11
Qu < -233.6
T3: 220.0
T5: 116.0
T8: 183.0
T1: 481.0
Vl < 1.0412
Pu > 956.04
T6: 53.0
D5: 63.0
T9: 68.0
T0: 115.0
Vl < 1.0798
D6: 48.0
D7: 20.0
Learning set classification (w.r.t. SECURITY)
Insecure: 310
Secure: 690
P(Insecure) = 0.3100
P(Secure) = 0.6900
OMIB - 10000 objects - L. Wehenkel Fri Dec 12 1997
DT5-BPR-1.0. N = 1000 M = 0 alfa = 0.00010 Pe = *%

Figure 16



Learning set classification (w.r.t. SECURITY)

Insecure: 310 P(Insecure) = 0.3100

Secure: 690 P(Secure) = 0.6900

Test set classification

Non detection costs : Insecure: 1.0 Secure: 1.0

Reference Security	Decision Tree Class Insecure	Secure	Total
Insecure	254	51	305
Secure	60	635	695
Total	314	686	1000

OMIB - 10000 objects - L. Wehenkel Fri Dec 12 1997

DT5-BPR-1.0. N = 1000 M = 1000 alfa = 0.00010 Pe = 11.1%

Figure 17

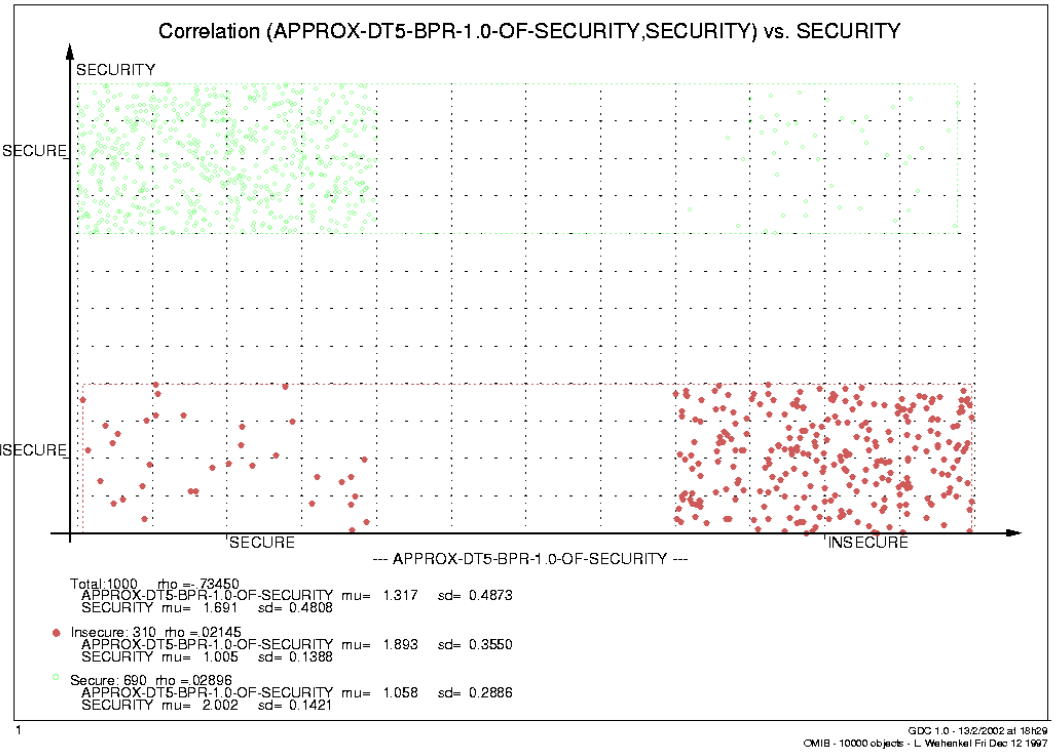


Figure 18

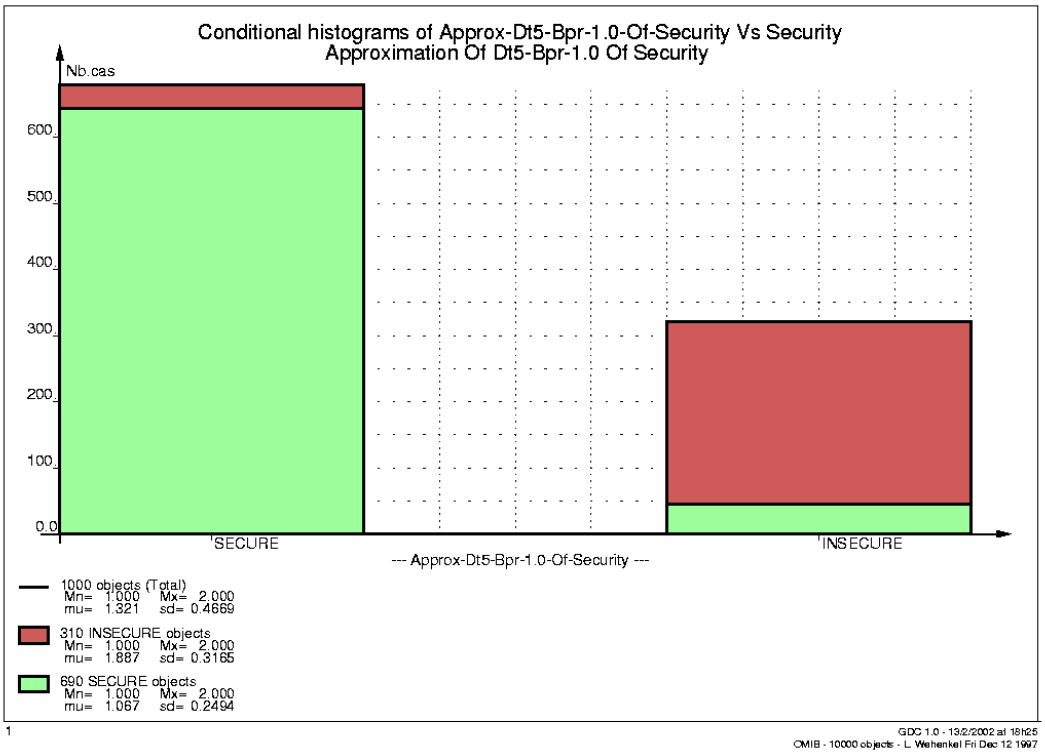


Figure 19

4.3 Regression Tree

4.3.1 What is it?

Definition. Regression trees (RT) are tools used in regression problems. They are concerned with the automatic design of if-then rules. They have a numerical output and symbolic and/or numerical inputs.

Method characteristics. The main strength of RT is its interpretability. Another asset is the ability to identify the most relevant attributes for a problem: the model itself selects a part of the attributes from the list of candidate attributes as the model inputs. It is more complex than a decision tree and thus the generated rule base is larger. It is a computationally efficient tool, comparatively fast to a decision tree and much faster than a neural network. It is less accurate than a neural network and in many cases less accurate than a linear regression technique. It may be used in association with a neural network in a hybrid approach.

4.3.2 Selections to make before starting

Define the problem. Choose `*goal-regression*` and choose `*candidate attributes*`. Admissible input attribute types are: "ordonee" and "qualitatif-quinlan". Note that the model does not handle linear-combination attribute values, they being excluded from the `*candidate-attributes*` list prior to model building. Temporal attribute values are replaced by a list of scalar ones.
Select data. Choose `*learning-set*` and `*test-set*` (if you also want to test or prune the model).
Set method parameters. Method parameters are `*alfa-rt*`, `*v-min*` and `*total-variance-min*`.

alfa-rt

- Necessary to detect deadens in regression trees, based on a Kolmogorov-Smirnov probability in node
- Takes real values between 0.0 (trivial tree) and 1.0 (full grown tree)
- Default value 0.0001
- Use value 0.000001 for strong pre-pruning
- ATDIDT 2.2 command: menu `:AUTOMATIC_LEARNING`, menu `:TREE_INDUCTION`, command `:SET_*ALFA-RT*`

v-min

- Necessary to detect leaves in regression trees, based on variance in node
- Takes any real value between 0.0 and 1.0e+11
- Default value 0.0 (complete tree)
- ATDIDT 2.2 command: menu `:AUTOMATIC_LEARNING`, menu `:TREE_INDUCTION`, command `:SET_*V-MIN*`

total-variance-min

- Necessary to detect deadens in regression trees, based on total variance in node
- Takes any real value between 0.0 and 1.0e+11
- Default value 0.0 (complete tree)
- ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :TREE_INDUCTION, command :SET_*TTVM*

4.3.3 Apply the method

Command. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :TREE_INDUCTION, command :BUILD_RT

Effects of the method employment. If the regression tree name is *xx* (the default name is *RT*<*i*>) and the *goal-regression* is *yy*:

- the new functional attribute created by default once the model is built has the name *approx-xx-of-yy*
- the created file containing information about the building process has the name *xx.log* and is located in the current directory
- The new regression tree *xx* is pushed in the global variable *decision-trees*
- The global variable *current-dt* keeps the last built (decision or regression) tree

Interesting displayed information while building the tree: status variables, LS size at every node, type of node, statistics of the output in node (mean, max, min, standard deviation), variance of node, possible tests, their scores, and the reduction of the variance each test brings, the correlation coefficient of each attribute's optimal test with the optimal test of the selected attribute, chosen test, CPU time.

Interesting displayed information while describing the results: a résumé of used parameters and settings, total variance of LS and total variance reduction realized by RT, RT complexity, the percentage of the total variance reduction explained by the tree by every chosen attribute (a measure of the relevance of every attribute in the model), the name of the new created functional attribute, the name and the path of the crated file containing the displayed information. All these information may be redisplayed anytime by using the command :DESCRIBE_TREE from :TREE_INDUCTION menu.

4.3.4 Test the model

Command. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :TREE_INDUCTION, command :TEST_TREE

The program detects when TS and LS are overlapping and asks the user if he wants to eliminate this overlap objects or not from TS before performing the testing.

Interesting displayed information while describing the results: statistics (mean, max, min, standard deviation, standard error) on: errors, absolute errors, squared errors, positive and negative errors; CPU time.

4.3.5 Improve the model

The pruning procedure generates a sequence of intermediary trees from the original complete tree and based on these trees' mean absolute errors (computed on TS) the best tree is chosen following the n-standard-error-rule, i.e. the less complex tree not significantly less reliable than the best one is selected.

Note that the pruning procedure, in order to be effective, should be applied on a complete tree, i.e. a fully-grown tree. In this respect, the model's parameters should be chosen in consequence: `*alfa-rt*`=1.0 and `*v-min*`=0.0, `*total-variance-min*`=0.0.

To do before starting. For pruning a tree the user must before test the tree. The errors of the intermediary trees are computed on the global variable `*test-set*.` In order to use the cross-validation approach, the user should settle variable `*test-set*` as a set independent of LS and TS and test the original tree (on this set called usually pruning set) before pruning it.

Set method parameters. Method parameters are `*sigma-multiplier*` and `*maximum-tree-prune-complexity*.` See pruning of decision trees for details on how to settle these parameters.

Command. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu `:TREE_INDUCTION`, command `:PRUNE_TREE`

Interesting displayed information while pruning the tree: some information concerning every intermediary tree (complexity, terminal nodes, the next node to prune, mean absolute error on TS, mean squared error on TS, corresponding `*alfa-rt*` parameter), information about the chosen tree.

Effects of the method employment. If the original regression tree name is *xx*, the `*goal-regression*` is *yy* and `*sigma-multiplier*` is 1.0:

- The pruned regression tree has the name *xx-BPR-1.0*
- The new functional attribute created by default once the tree is pruned has the name *approx-xx-BPR-1.0-of-yy*
- The created file containing information about the pruning process has the name *xx-BPR-1.0.log* and is located in the current directory
- The new regression tree *xx-BPR-1.0* is pushed in the global variable `*decision-trees*`
- The global variable `*current-dt*` keeps the pruned tree *xx-BPR-1.0*

ATDIDT 2.2 command `:DRW_PR_SEQ` provides a graphic of pruning sequence curves displaying the evolution of regression trees' complexity, variance reduction, mean absolute error and quality with parameter `*alfa*.` Two files named *xx-BPR-1.0.pruning* and *xx-*

BPR-1.0-pruning-seq.ps are created in the current directory. The postscript one contains these graphics that may be visualized at any time by using GhostView tool.

4.3.6 Results visualization / interpretation

Describe tree. ATDIDT 2.2 command `:DESCRIBE_TREE` displays a résumé of the current regression tree growing and testing results (if the tree has been tested before). By current tree we understand the tree indicated by the global variable `*current-dt*`, i.e. the last built tree, or the last pruned tree, or the last tree chosen with the command :`CHOOSE_TREE`. It may be applied at any time, once a (decision or regression) tree is stored in the variable `*current-dt*`.

Display tree. ATDIDT 2.2 command `:DISPLAY_TREE` displays the current tree on a single page. Command `:MY_DISPLAY_TREE` displays the tree on multiple pages, on the first page only the upper levels of the tree and on the other pages, the rest of the subtrees. It should be used for very complex trees (too complex to be displayed on a single sheet). Both commands generate a postscript file (named *xx.ps* for a RT called *xx*) located in the current directory that may be visualized at any time using the GhostView tool.

ATDIDT 2.2 command `:DRAW_TEST_SET` enables or disables the representation of the test results on the tree graphic. Figure 20 presents an example of a regression tree display without test results, and Figure 21, with test results.

Figure 20 and Figure 21 draw a regression tree for a `*goal-regression*` called “cct-sbs” (see the attribute definition in database declaration file example of appendix), built on a learning set of 1000 objects, pruned and tested on a independent test set of 1000 objects. Each node of the tree is represented by a box. Above the box appears the name of the node, test (T), leaf (L) or deadend (D) and the number of node learning states. The total number of different nodes is indicated above the root node. Below every test node, the corresponding test is indicated and each arc leading to a successor is labeled with a possible answer to this test (Yes and No). In Figure 20, each node’s box corresponds only to the learning set results. The node box area is proportional to the node’s local learning set size. Inside each node box, the mean value of the regression tree approximation together with its standard deviation (in brackets) computed on local LS is marked. The horizontal splits in nodes simulate this mean value plus/minus one standard deviation. In this way, the variance reduction from root node to terminal nodes becomes graphically visible. In Figure 21, the node’s box is divided into two parts, the upper one corresponding to the learning set as explained already, the lower one to the test set. Their relative heights are proportional to the relative sizes of the learning and test sets at the node. The part corresponding to the test set displays the mean value and its standard deviation for the node’ subtree absolute error computed on TS. Root node test part gives the mean value for the absolute error of the entire tree.

HTML format. ATDIDT 2.2 commands `:SAVE_TREE` and `:INSPECT_TREE` give another way of visualizing results, in html format. For a regression tree named *xx*, the first command creates a new directory called */Sav/xx/* in the current directory, and puts 7 files concerning the

tree in this new directory. The second command opens a *Welcome.html* file that displays general information about the tree together with hyperlinks for all the created files:

- *xx-rules.html* – displays the IF-THEN rule base derived from the tree
- *xx-prune.lst* (for a pruned tree) or *xx-grown.lst* (for the original tree) – displays information that describe the pruning / growing processes
- *xx.dump* – outputs the internal lisp structure of the tree *xx*
- *xx.lsp* – contains the lisp function of the new created functional attribute
- *xx-mp.pdf* and *xx-sg.pdf* – are single page and multiple page displays of the tree.

Afterwards, at every new session, the ATDIDT 2.2 command `:LOAD_TREE` may load this built tree (model) based on the *xx.dump* file, thus releasing the user from building it again.

Derived rule-base. For every terminal node of a regression tree, an IF-THEN rule is generated. The file *xx-rules.html* indicates for every rule of type "if antecedent then output=value", extracted from the *xx* decision tree, the next coefficients:

- support of rule – percentage of all objects in LS for which this rule is active
- output estimation when the rule is active.

Example of rule deducted from regression tree of Figure 20:

Rule **T4**: IF Pu > 1135.9 and Qu < -205.0 THEN CCT-SBS = 0.076139
Support = 7.4%

Other ideas for graphics. If the regression tree name is *xx*, the `*goal-regression*` is *yy* and *ww* is one input attribute:

- Scatter-plot (*approx-xx-of-yy*, *yy*) on LS (see Figure 22) or TS
- Histogram for *approx-xx-of-yy* on LS, TS
- Define a functional attribute *zz* as the error / absolute error / squared error of RT and visualize a scatter-plot for (*yy*, *zz*) or a histogram for *zz* on LS, TS
- Scatter-plot (*approx-xx-of-yy*, *ww*) on LS (see Figure 23) or TS.

4.3.7 Other possible actions

All the commands having the format :XXX_TREE are available both for regression and decision trees and are regarding the tree stored in the global variable `*current-dt*.`

Other useful available commands:

`:SELECT-DT-TEST-ATTS` – settles the global variable `*candidate-attributes*` as the list of all the attributes chosen by the current decision tree. This command becomes very useful when decision tree technique is used in a hybrid approach together with other methods. A regression tree has the ability to reduce the input space to the relevant attributes for a given problem.

`:GET_DT_ERRORS` – select the global variable `*learning-set*` as the objects from `*test-set*` for which the regression tree approximation is different from the `*goal-`

`regression*` used to build the tree. As it is often the case for regression trees, this option is not very useful for regression trees context, since often here the new LS is similar with the original LS.

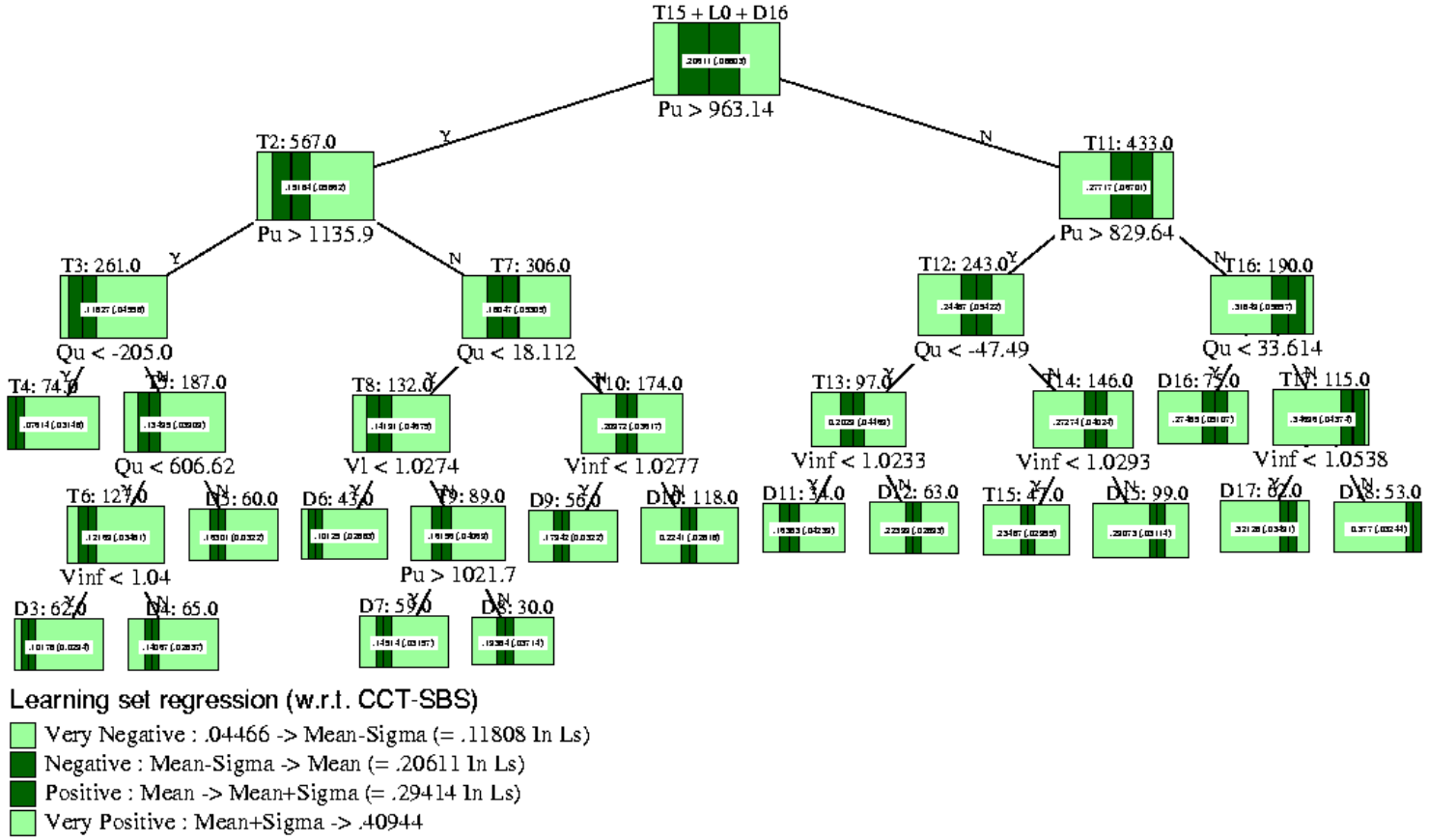
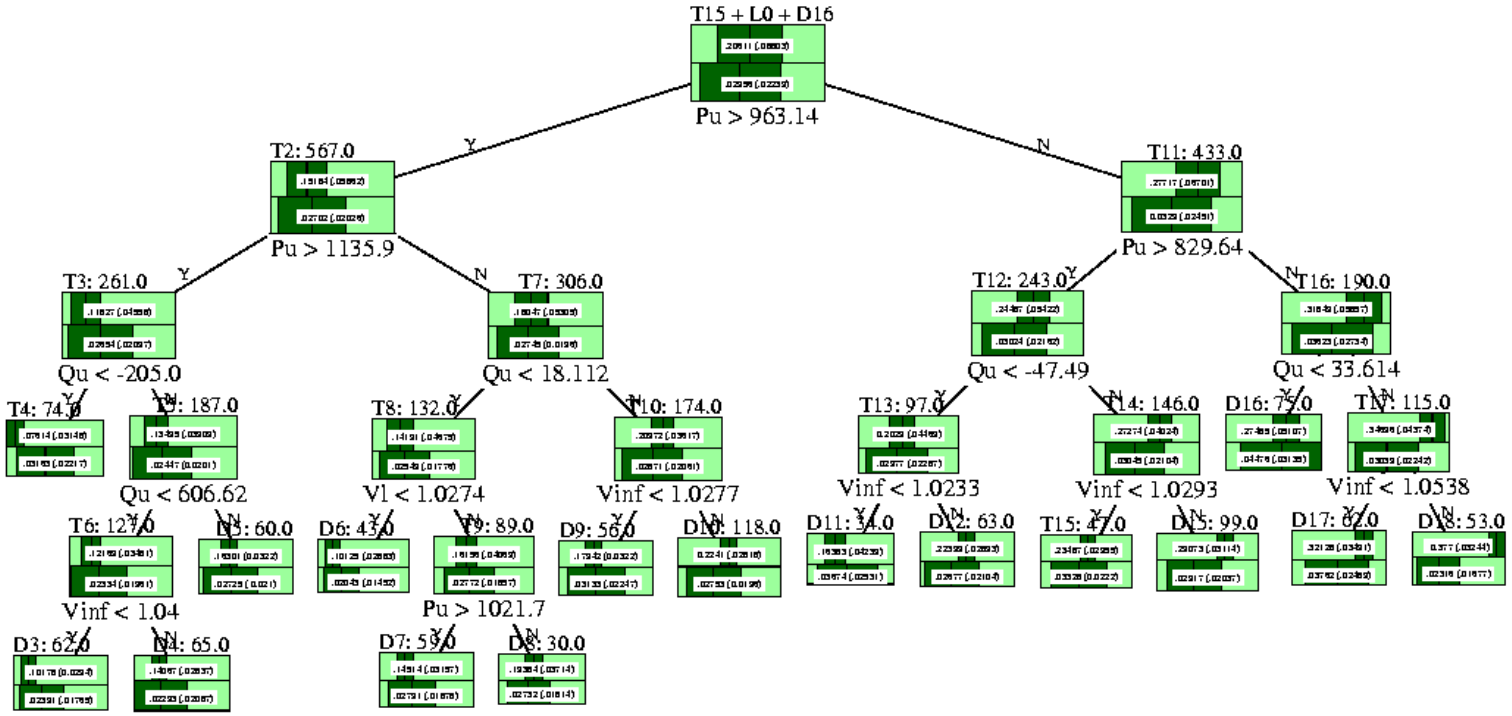

T15 + L0 + D16
Pu > 963.14
Y
N
T2: 567.0
T11: 433.0
Pu > 1135.9
Pu > 829.64
T3: 261.0
T7: 306.0
T12: 243.0
T16: 190.0
Qu < -205.0
Qu < 18.112
Qu < -47.49
Qu < 33.614
T4: 74.0
T5: 187.0
T8: 132.0
T10: 174.0
T13: 97.0
T14: 146.0
D16: 75.0
T17: 115.0
Qu < 606.62
Vl < 1.0274
Vinf < 1.0277
Vinf < 1.0233
Vinf < 1.0293
Vinf < 1.0538
T6: 127.0
D3: 60.0
D6: 43.0
D9: 89.0
D9: 56.0
D10: 118.0
D11: 34.0
D12: 63.0
T15: 47.0
D15: 99.0
D17: 82.0
D18: 53.0
Vinf < 1.04
Pu > 1021.7
D3: 62.0
D4: 65.0
D7: 59.0
D8: 30.0
Learning set regression (w.r.t. CCT-SBS)
Very Negative : .04466 -> Mean-Sigma (= .11808 ln Ls)
Negative : Mean-Sigma -> Mean (= .20611 ln Ls)
Positive : Mean -> Mean+Sigma (= .29414 ln Ls)
Very Positive : Mean+Sigma -> .40944
OMIB - 10000 objects - L. Wehenkel Fri Dec 12 1997
RT12-BPR-1.0. N = 1000 (v-min = 0.0 t-v-min = 0.0 alfa = 1.0e-7) M = 1000 (MAE = 0.029563)

Figure 20



Learning set regression (w.r.t. CCT-SBS)

- Very Negative : .04466 -> Mean-Sigma (= .11808 ln Ls)
- Negative : Mean-Sigma -> Mean (= .20611 ln Ls)
- Positive : Mean -> Mean+Sigma (= .29414 ln Ls)
- Very Positive : Mean+Sigma -> .40944

Test set regression

Error-Type	Number	Mean	Min	Max	Sigma	St.Err.
Negative	457	.027232582	.000043332	.148345339	.019944869	.000932982
Positive	543	.031524172	0.00000423	.127954659	.024079901	.001033367
Total	1000	.029562915	0.00000423	.148345339	.022387898	.000707968

OMIB - 10000 objects - L. Wehenkel Fri Dec 12 1997
RT12-BPR-1.0. N = 1000 (v-min = 0.0 t-v-min = 0.0 alfa = 1.0e-7) M = 1000 (MAE = 0.029563)

Figure 21

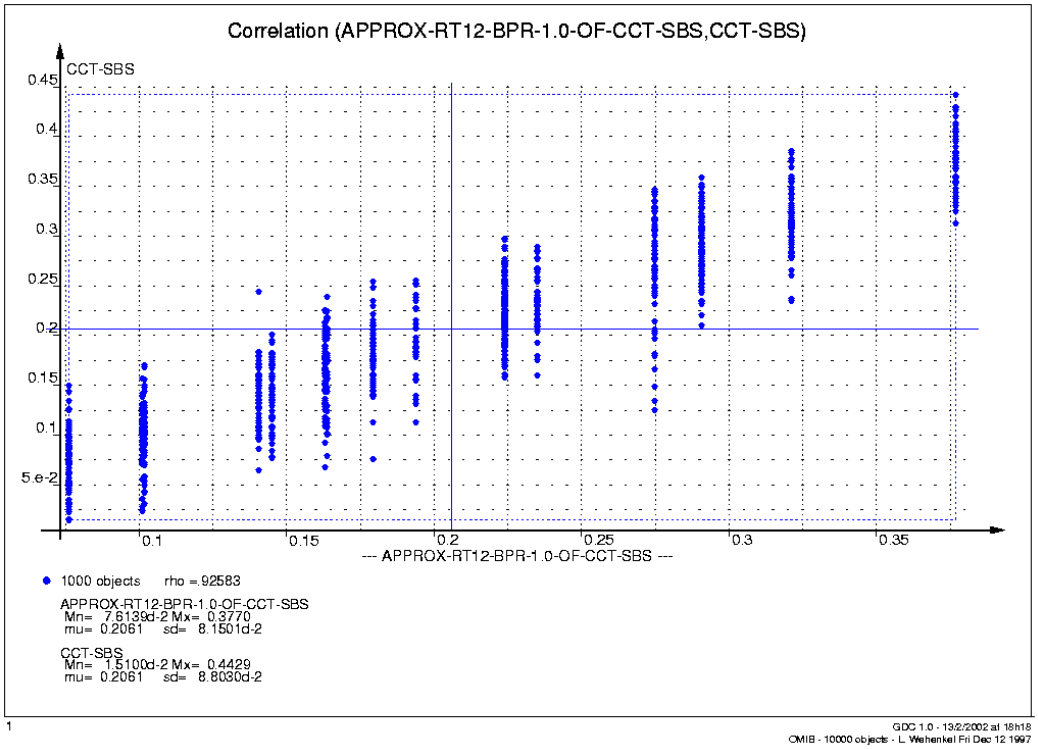


Figure 22

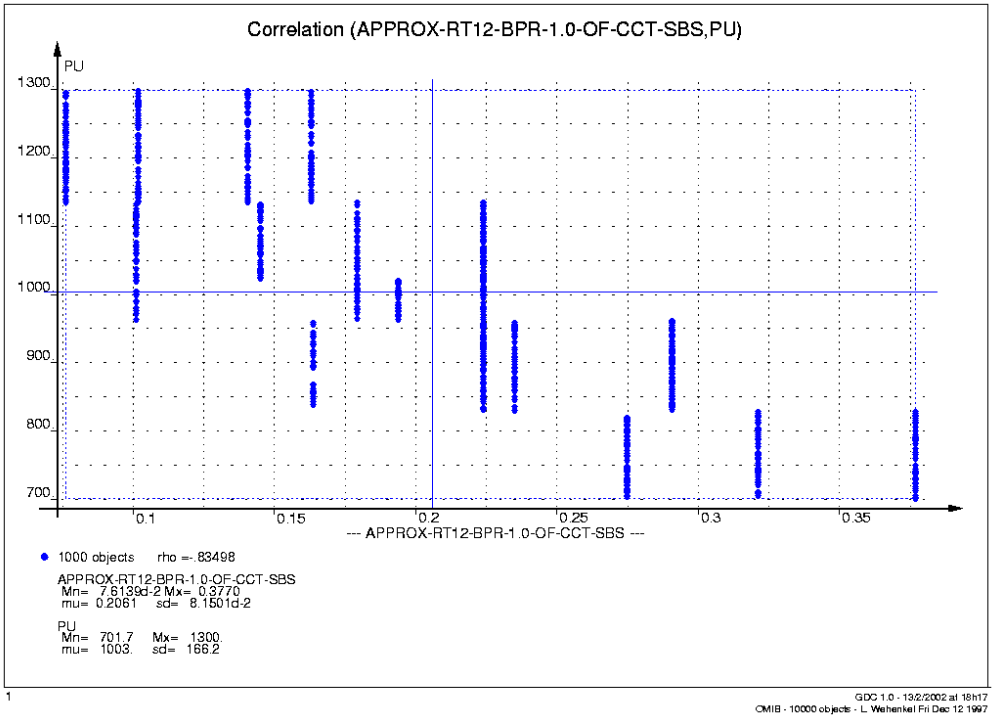


Figure 23

4.4 Linear Regression

4.4.1 What is it?

Definition. Linear regression tool is used in regression problems. The model predicts one attribute (the output) by means of other attributes (the inputs) by a linear function. It is a least squares linear combination of all the inputs with respect to the output.

$$output = c_0 + c_1 input_1 + c_2 input_2 + ...c_K input_K$$

The model has a numerical output and non-constant numerical inputs.

Method characteristics. The main strength of linear regression is its computational efficiency for reasonable sized input spaces. It is much faster then regression trees or neural networks. When the input space dimension K (number of input attributes) is high, the method is less efficient, due to a K*K matrix manipulation (inverse matrix computation) that is quadratic in K. The model complexity is given by the input space size K and the model free parameters are the K coefficients. In a hybrid approach and large sized input spaces, a dendrogram, a decision or regression tree may reduce the input space and then a linear regression technique may find the linear combination for predicting a certain output.

4.4.2 Selections to make before starting

Define the problem. Choose `*goal-regression*` and choose `*candidate attributes*`. Admissible input attribute type is "ordonee". Note that the model does not handle constant attribute values, qualitative or linear-combination attributes, all being excluded from the `*candidate-attributes*` list prior to model building. Temporal attribute values are replaced by a list of scalar ones.
Select data. Choose `*learning-set*` and `*test-set*` (if you also want to test the model).
Set method parameters. Method parameter is `*weight-decay*`.

weight-decay

- Penalization term in the "ridge-regression" model
- Takes positive real values
- Default value 0.00001 (almost no penalization)
- If the user wants to reduce the variance of the linear regression he should use larger values, say 1.0.
- ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :LINEAR_REGRESSION, command :SET_WEIGHT_DECAY

4.4.3 Apply the method

Command. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :LINEAR_REGRESSION, command :LEARN_LINEAR_REGRESSION_APPROXIMATION.

Effects of the method employment If the *goal-regression* is *yy*:

- The new functional attribute created by default once the model is built has the name *linear-regression-yy*, or any name given by the user
- The created file containing information about the model building process has the name *linear-regression-yy.log*, and is located in the current directory. If it already exists, the new information is appended to the old one in the file.

Interesting displayed information while building the model: status variables, CPU times, a description of the new created attribute, its explicit function giving the linear dependence.

4.4.4 Test the model

Command. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :LINEAR_REGRESSION, command :TEST_REGRESSION_APPROXIMATION

Interesting displayed information while describing the results: statistics (mean, max, min, standard deviation, standard error) on: errors and absolute errors; CPU time.

4.4.5 Results visualization / interpretation

Explicit function. An example of the linear model function detected by ATDIDT is:

$$cct - sbs = 0.62587 - 0.00043Pu + 0.00008Qu\,.$$

Displayed graphic. Before testing, the user is prompted for a numerical attribute name *xx* (for example the new created linear regression attribute), to be compared with the *goal-regression* *yy*. After testing, a scatter-plot is automatically displayed, (*xx*, *yy*) on TS (see Figure 24). The corresponding created postscript file is named *linear_regtst-tem.ps* and is located in the current directory.

Other ideas for graphics. If the *goal-regression* is *yy* and *xx* is one input attribute:

- Scatter-plot (*linear-regression-yy*, *yy*) on LS
- Histogram for *linear-regression-yy* on LS, TS
- Define a functional attribute *zz* as the error / absolute error / squared error of the linear model and visualize a scatter-plot for (*yy*, *zz*) or a histogram for *zz* on LS, TS
- Scatter-plot (*linear-regression-yy*, *xx*) on LS (see Figure 25) or TS.

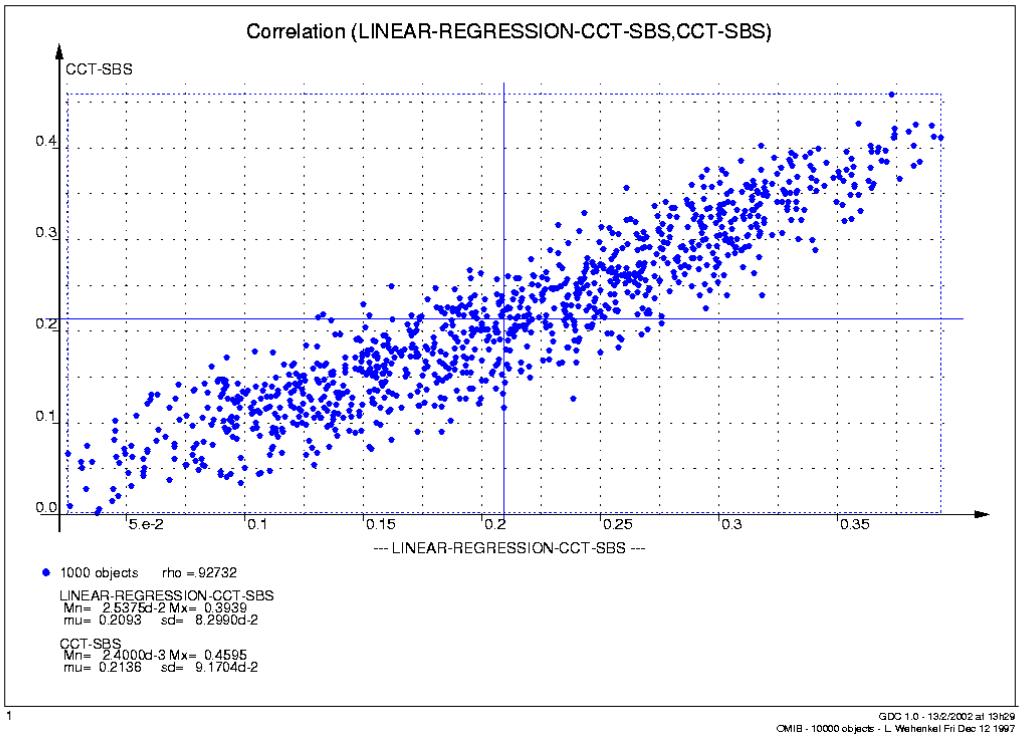


Figure 24

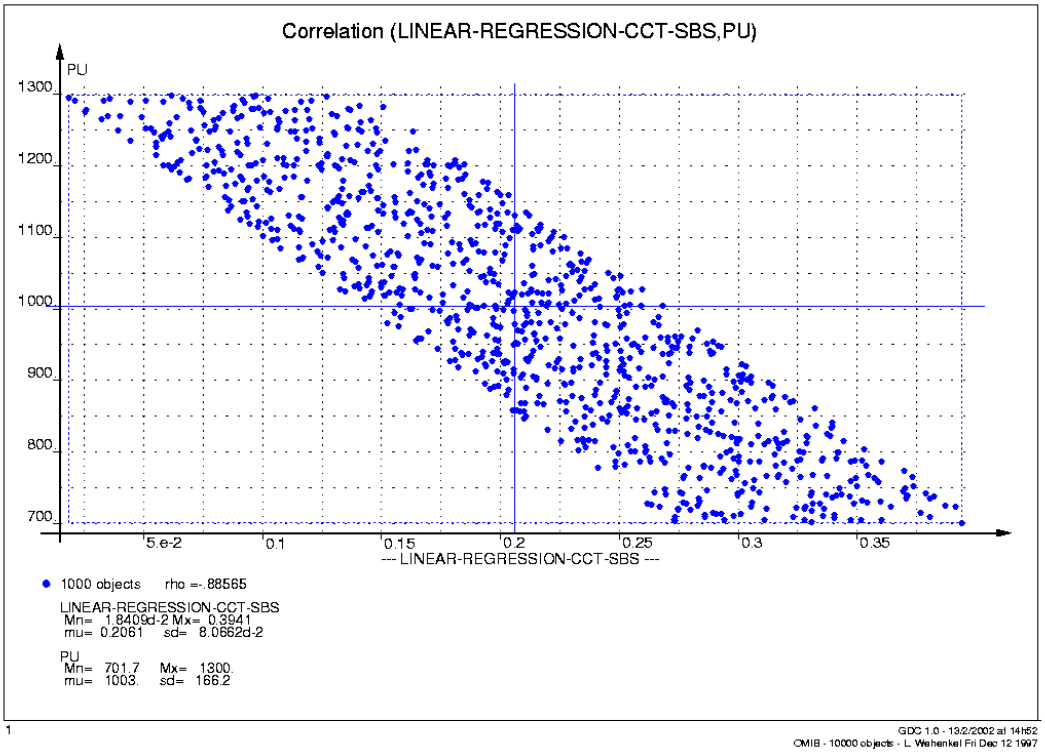


Figure 25

4.5 Linear Hinges Model

4.5.1 What is it?

Definition. Linear Hinges model is a one-dimensional regression problem, i.e. curve fitting from two-dimensional scatter-plot data. The model predicts one attribute (the output) by means of other attribute (the input) by a piecewise linear model. It has a numerical output and non-constant numerical inputs.

Method characteristics. The model is very computationally efficient. The number of linear pieces of the model gives the model complexity.

4.5.2 Selections to make before starting

Define the problem. Choose `*goal-regression*`.
Select data. Choose `*learning-set*` and `*test-set*` (if you also want to test the model).

4.5.3 Apply the method

Command. ATDIDT 2.2 command[3]: menu :AUTOMATIC_LEARNING, menu :LINEAR_REGRESSION, command :HINGES.

The command prompts for the attribute name considered as input for the model. Admissible input attribute type is "ordonee". Note that the model does not handle constant attribute values, qualitative, linear combination or temporal attributes.

Effects of the method employment. If the input attribute is *xx* and `*goal-regression*` is yy:

- The new functional attribute created by default once the model is built has the name *linear-hinges-xx-yy*
- No log file is generated.

Interesting displayed information while building the model: number of knots, learning and pruning set sizes, new created attribute name, CPU time.

4.5.4 Test the model

Command. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :LINEAR_REGRESSION, command :TEST_REGRESSION_APPROXIMATION

[3] Model not available in ATDIDT 3.0 version

Interesting displayed information while describing the results: statistics (mean, max, min, standard deviation, standard error) on: errors and absolute errors; CPU time.

4.5.5 Results visualization / interpretation

Displayed graphic. Before testing, the user is prompted for a numerical attribute name *xx* (for example the new created linear hinges attribute), to be compared with the `*goal-regression*` *yy*. After testing, a scatter-plot is automatically displayed, (*xx*, *yy*) on TS (see Figure 26). The corresponding created postscript file is named *linear_regtst-tem.ps* and is located in the current directory.

Other ideas for graphics. If the input attribute is *xx* and `*goal-regression*` is *yy*:

- Histogram for *linear-hinges-xx-yy* on LS, TS
- Scatter-plot (*linear-hinges-xx-yy*, *yy*) on LS
- Define a functional attribute *zz* as the error / absolute error / squared error of the linear model and visualize a scatter-plot for (*yy*, *zz*) or a histogram for *zz* on LS, TS
- Scatter-plot (*linear-hinges-xx-yy*, *xx*) on LS (see Figure 27) or TS.

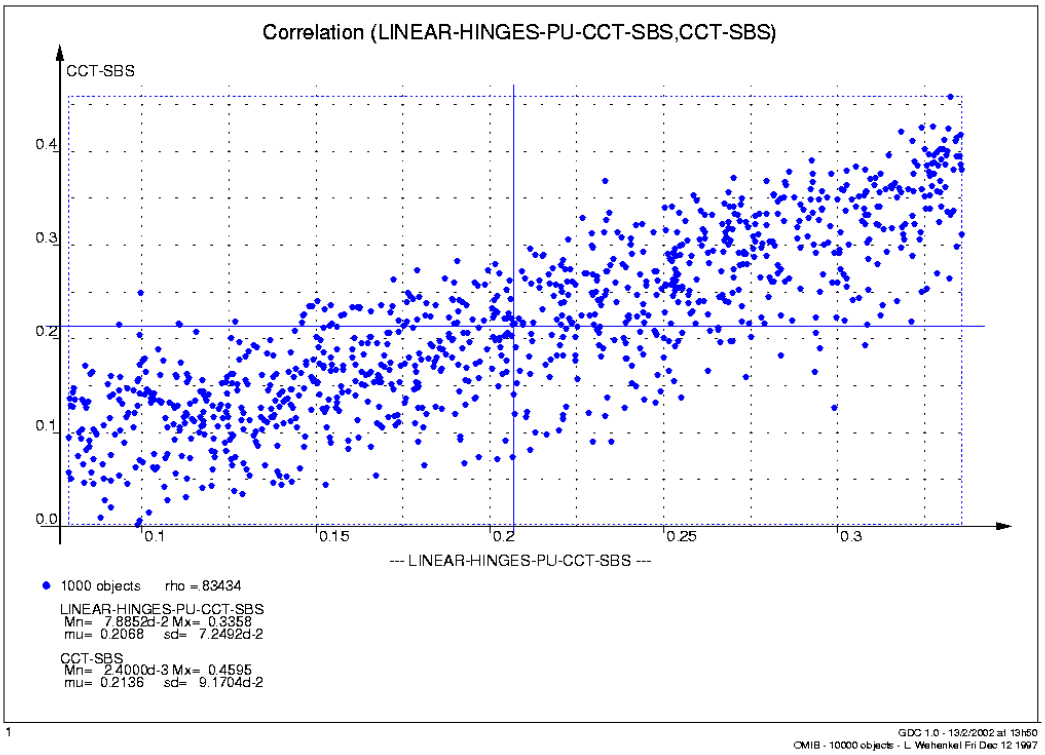


Figure 26

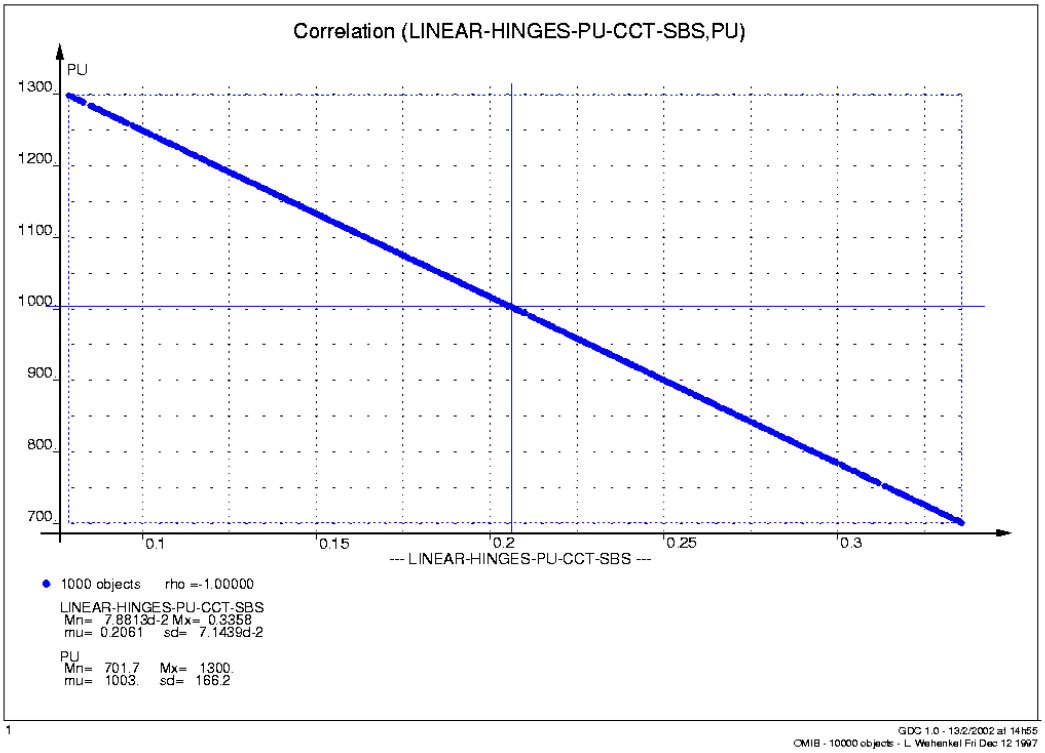

Correlation (LINEAR-HINGES-PU-CCT-SBS,PU)
PU
1300
1200
1100
1000
900
800
700
0.1
0.15
0.2
0.25
0.3
--- LINEAR-HINGES-PU-CCT-SBS ---
1000 objects rho =-1.00000
LINEAR-HINGES-PU-CCT-SBS
Mn= 7.8813d-2 Mx= 0.3358
mu= 0.2061 sd= 7.1439d-2
PU
Mn= 701.7 Mx= 1300.
mu= 1003. sd= 166.2
1
GDC 1.0 - 13/2/2002 at 14h55
CMIB - 10000 objects - L. Wehenkel Fri Dec 12 1997

Figure 27

4.6 Regression Tree Bagging

4.6.1 What is it?

Definition. Regression tree bagging is used in regression problems. The model predicts one attribute (the output) by means of other attributes (the inputs) by averaging multiple regression trees estimations. The model has a numerical output and numerical inputs. A number of regression trees are built in the iterative way: a random subset of the `*learning-set*` is internally selected (size = 50% of the `*learning-set*` size), then a regression tree is built on this subset. At the end, a new model is constructed by aggregating all these trees, and the model's prediction is the average prediction of the trees.

Method characteristics. It provides more accurate output estimators than single regression tree building and less accurate than neural networks or regression tree boosting. With respect to regression trees, the averaged model looses the interpretability character. The CPU time is rather high with respect to other regression methods.

4.6.2 Selections to make before starting

Define the problem. Choose `*goal-regression*` and choose `*candidate attributes*`. Admissible input attribute type is "ordonee". Note that the model does not handle linear-combination or constant attribute values, they being excluded from the `*candidate-attributes*` list prior to regression tree model building. Equally, the model does not handle qualitative attributes. Temporal attribute values are replaced by a list of scalar ones.
Select data. Choose `*learning-set*` and `*test-set*` (if you also want to test the model).
Set method parameters. Method parameters are `*size-of-trees-for-bagging*` and `*number-of-bagging-terms*`. Also, the model is based on the regression trees intrinsic parameters: `*alfa-rt*`, `*v-min*` and `*total-variance-min*`.

size-of-trees-for-bagging
- The upper bound of regression tree complexity during bagging
- Takes integer values between 0 (trivial tree) and 10.000 (large tree)
- Default value 500
- Typically, it is preferable to build large trees so as to reduce bias as much as possible
- ATDIDT 3.0 command: menu :AUTOMATIC_LEARNING, menu :META_LEARN, command :BAGG_COMPLEXITY.

number-of-bagging-terms
- The number of regression trees which are built during bagging
- Takes integer values between 0 (no model) and 50
- Default value 20
- ATDIDT 3.0 command: menu :AUTOMATIC_LEARNING, menu :META_LEARN, command :BAGG_NUMBER.

4.6.3 Apply the method

Command. ATDIDT 3.0 command[4]: menu :AUTOMATIC_LEARNING, menu :META_LEARN, command :TREE_BAGGING

Effects of the method employment.

- The new functional attribute created by default once the model is built has the name RT-BAGG<i>
- No log file is generated.

Interesting displayed information while building the model: status variables, summary description of each intermediary regression tree, CPU times.

4.6.4 Test the model

Command. ATDIDT 3.0 command: menu :AUTOMATIC_LEARNING, menu :META_LEARN, command :TEST_REGRESSION_APPROXIMATION

Interesting displayed information while describing the results: statistics (mean, max, min, standard deviation, standard error) on: errors and absolute errors; CPU time.

4.6.5 Results visualization / interpretation

Displayed graphic. Before testing, the user is prompted for a numerical attribute name *xx* (for example the new created model), to be compared with the *goal-regression* *yy*. After testing, a scatter-plot is automatically displayed, (*xx*, *yy*) on TS (see Figure 28). The corresponding created postscript file is named *linear_regtst-tem.ps* and is located in the current directory.

Other ideas for graphics. If the *goal-regression* is *yy*, *ww* is the model's name and *xx* is one input attribute:

- Scatter-plot (*ww*, *yy*) on LS
- Histogram for *ww* on LS, TS
- Define a functional attribute zz as the error / absolute error / squared error of the linear model and visualize a scatter-plot for (*yy*, *zz*) or a histogram for zz on LS, TS
- Scatter-plot (*ww*, *xx*) on LS (see Figure 29) or TS.

[4] Model not available in ATDIDT 2.2 version

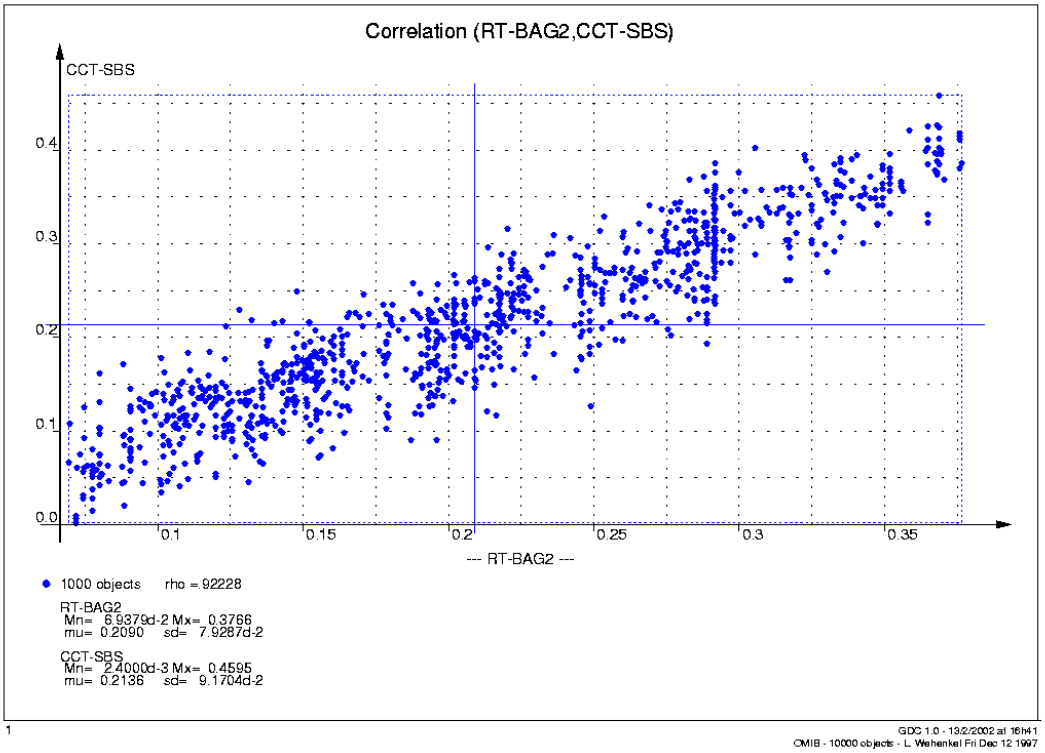


Figure 28

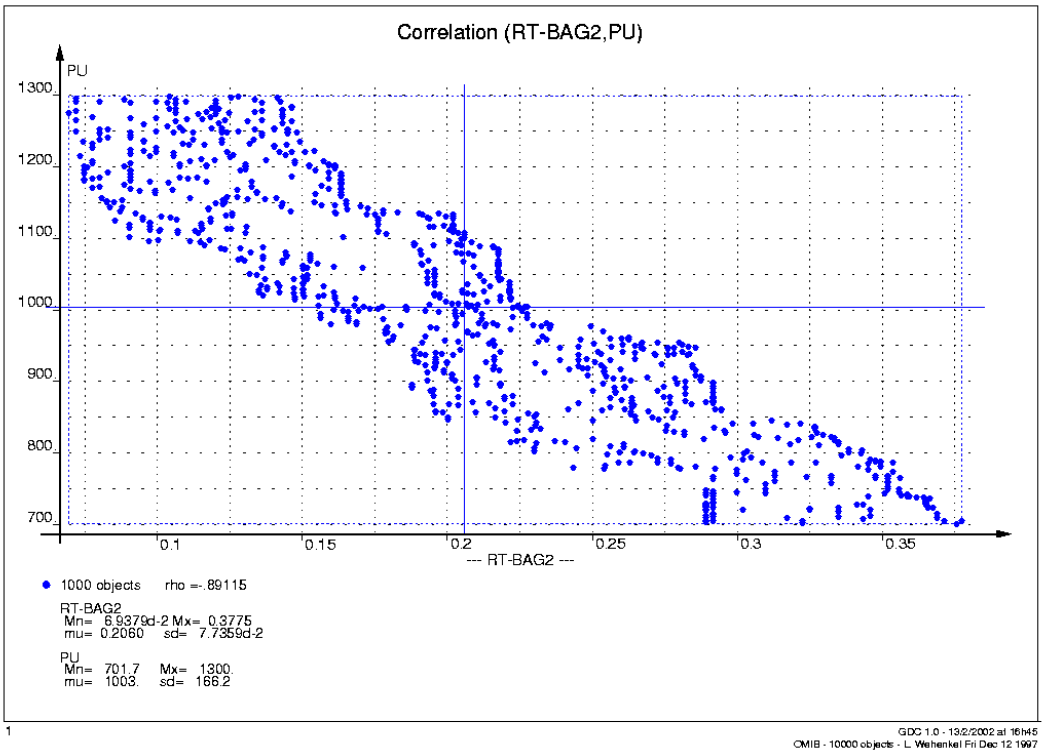


Figure 29

4.7 Regression Tree Boosting

4.7.1 What is it?

Definition. Regression tree boosting is used in regression problems. The model has a numerical output and numerical inputs. A number of regression trees are built in the iterative way: first a linear regression is built to fit the *goal-regression*; then a number of small regression trees is built using an iterative residual fitting method; finally, the tree-models and attributes are combined in a generalized linear model to fit the *goal-regression*.

Method characteristics. The interpretability character is lost with respect to regression trees. It is computationally efficient. It provides more accurate output estimators than single regression tree building or regression tree bagging, and comparative results with neural networks.

4.7.2 Selections to make before starting

Define the problem. Choose *goal-regression* and choose *candidate attributes*. Admissible input attribute type is "ordonee". Note that the model does not handle linear-combination or constant attribute values, they being excluded from the *candidate-attributes* list prior to model building. Equally, the model does not handle qualitative attributes. Temporal attribute values are replaced by a list of scalar ones.
Select data. Choose *learning-set* and *test-set* (if you also want to test the model).
Set method parameters. Method parameters are *size-of-trees-for-boosting* and *number-of-boosting-terms*. Also, the model is based on the regression trees intrinsic parameters. The user cannot control them, they being settled by default as *alfa-rt*=0.1, *v-min*=0.0 and *total-variance-min*=0.0.

size-of-trees-for-boosting

- The upper bound of regression tree complexity during boosting
- Takes integer values between 0 (trivial tree) and 10.000 (large tree)
- Default value 10
- It is preferable to build small trees so as to reduce variance as much as possible
- ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :LINEAR_REGRESSION, command :BOOST_COMPLEXITY
- ATDIDT 3.0 command: menu :AUTOMATIC_LEARNING, menu :META_LEARN, command :BOOST_COMPLEXITY.

number-of-boosting-terms

- The number of regression trees which are built during boosting
- Takes integer values between 0 (no model) and 50
- Default value 10
- ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :LINEAR_REGRESSION, command :BOOST_NUMBER

- ATDIDT 3.0 command: menu :AUTOMATIC_LEARNING, menu :META_LEARN, command :BOOST_NUMBER.

4.7.3 Apply the method

Command. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :LINEAR_REGRESSION, command :TREE_BOOSTING
ATDIDT 3.0 command: menu :AUTOMATIC_LEARNING, menu :META_LEARN, command :TREE_BOOSTING

Effects of the method employment.

- The new functional attribute created by default once the model is built has the name *RT-BOOST<i>*
- No log file is generated.

Interesting displayed information while building the model: status variables, summary description of each intermediary regression tree and linear regression, CPU times.

4.7.4 Test the model

Command. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :LINEAR_REGRESSION, command :TEST_REGRESSION_APPROXIMATION
ATDIDT 3.0 command: menu :AUTOMATIC_LEARNING, menu :META_LEARN, command :TEST_REGRESSION_APPROXIMATION

Interesting displayed information while describing the results: statistics (mean, max, min, standard deviation, standard error) on: errors and absolute errors; CPU time.

4.7.5 Results visualization / interpretation

Displayed graphic. Before testing, the user is prompted for a numerical attribute name *xx* (for example the new created model), to be compared with the *goal-regression* *yy*. After testing, a scatter-plot is automatically displayed, (*xx*, *yy*) on TS (see Figure 30). The corresponding created postscript file is named *linear_regtst-tem.ps* and is located in the current directory.

Other ideas for graphics. If the *goal-regression* is *yy*, *ww* is the model's name and *xx* is one input attribute:

- Scatter-plot (*ww*, *yy*) on LS
- Histogram for *ww* on LS, TS
- Define a functional attribute *zz* as the error / absolute error / squared error of the linear model and visualize a scatter-plot for (*yy*, *zz*) or a histogram for *zz* on LS, TS
- Scatter-plot (*ww*, *xx*) on LS (see Figure 31) or TS.

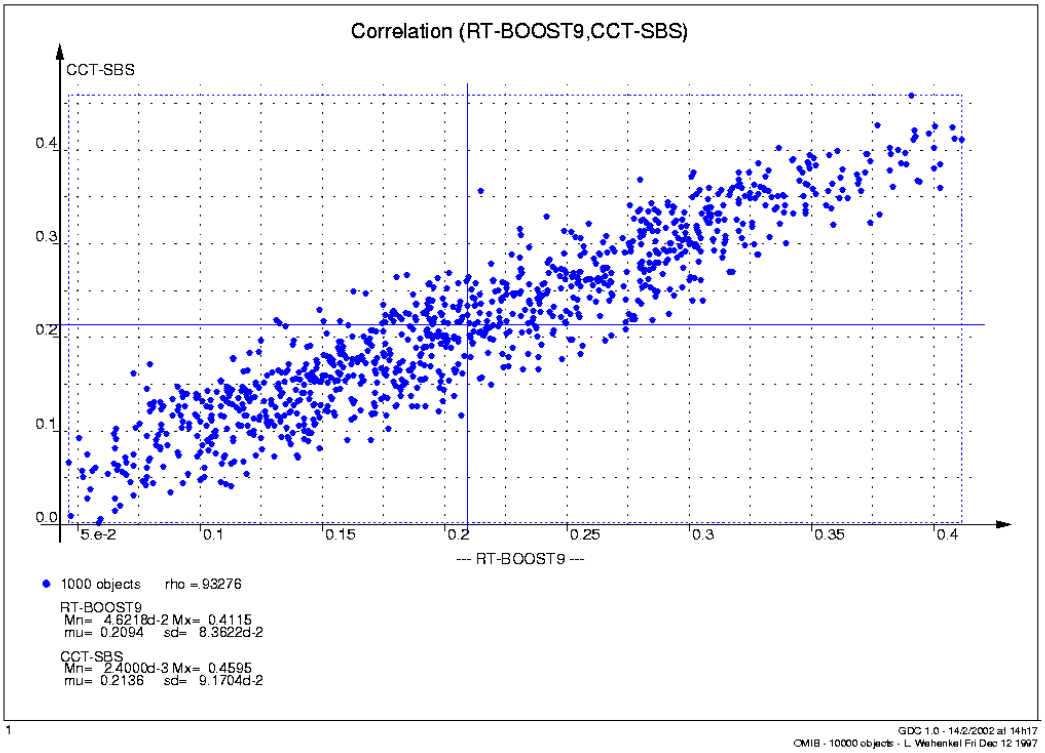


Figure 30

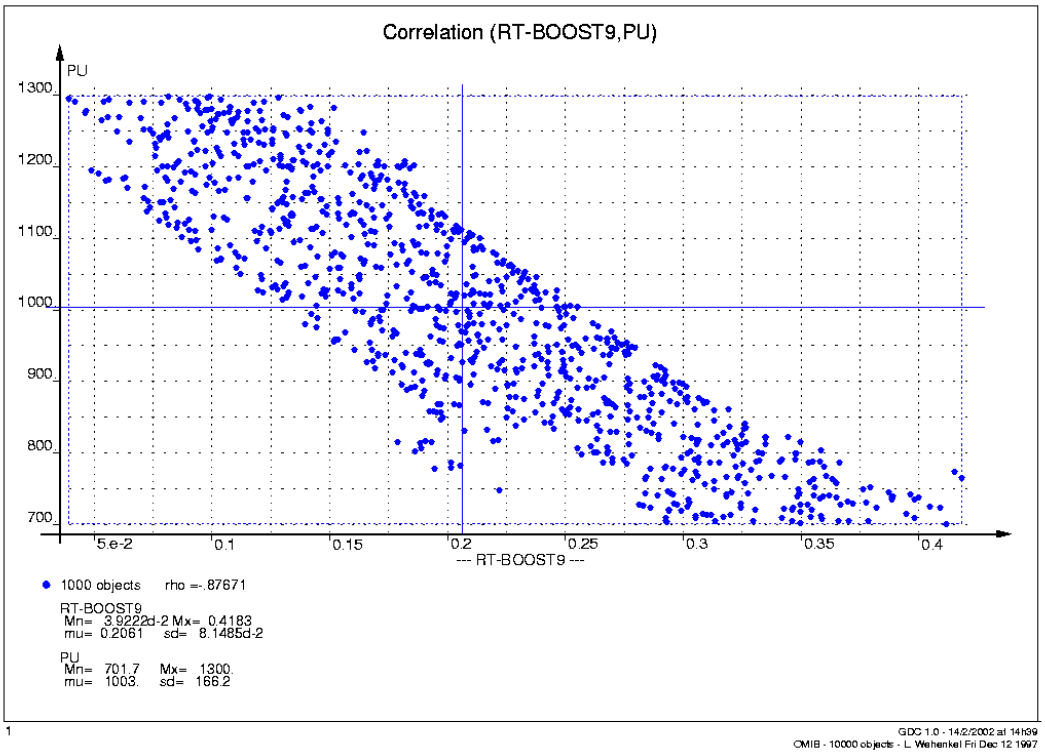


Figure 31

4.8 Multilayer Perceptron

4.8.1 What is it?

Definition. Multilayer perceptrons (MLP) are tools used in nonlinear regression and in nonlinear classification problems. The model predicts one attribute (the output) by means of other attributes (the inputs) by a nonlinear function. It supports numerical inputs and symbolic or numerical output depending on the problem.

Method characteristics. The main strength of MLP is its universal approximation capability. Among the ATDIDT data mining methods it is probably the most accurate one. Unfortunately, from the point of view of interpretability it is perceived as a black box. It is heavy in terms of CPU time concerning the training stage and may become cumbersome for highly dimensioned input spaces. That is why, it is advisable to be used in conjunction with other methods that firstly reduce the input space, like decision/regression trees or dendrograms (hybrid approaches). The criterion used for training is the minimum squared error without weight-decay term.

4.8.2 Selections to make before starting

Define the problem. Choose `*goal-regression*` or `*goal-classification*` and choose `*candidate attributes*`. Admissible input attribute type is "ordonee". Note that the model does not handle linear-combination or qualitative attribute values, they being excluded from the `*candidate-attributes*` list prior to model building. Temporal attribute values are replaced by a list of scalar ones.
Select data. Choose `*learning-set*` and `*test-set*` (if you also want to test the model).
Set method parameters. Method parameters are `*output-activation-function-name*`, `*hidden-layers*`, `*mlp-test-set-monitoring*`, `*mlp-cycle-number*`.

output-activation-function-name

- Indicates the type of the activation function of the output layer (note that at hidden layers the activation function is always tanh)
- Possible choices: "identite" (linear function), "tanh" (hyperbolic tangent), and "echelon" (Heaviside threshold function)
- Default type: "identite"
- ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :NON_LINEAR_REGRESSION, command :OUTPUT_LAYER_ACTIVATION.

hidden-layers

- Determines the structure of the MLP
- Default structure: one hidden layer with 10 neurons
- Multiple hidden layers are supported

- ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :NON_LINEAR_REGRESSION, command :SET_MLP_HIDDEN_STRUCTURE

mlp-test-set-monitoring

- Takes values t (if the *test-set* is not empty, monitoring of the test set error during training) or nil (monitoring of the learning set error during training)
- Default value nil
- If the toggle is on, the program returns the MLP approximation found during training which obtained the least error on the test set, otherwise it returns the last MLP obtained during training
- ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :NON_LINEAR_REGRESSION, command :MONITOR_TEST_SET.

mlp-cycle-number

- Defines the maximum number of iterations for MLP training; the training stops either when it converged (from the point of view of the mean squared error function) or when a given number of cycles have been scrolled out
- Takes integer values
- Default value 500
- ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :NON_LINEAR_REGRESSION, command :MLP_STOPPING_PARS.

Other parameters settled also by the command :MLP_STOPPING_PARS, used to decide when to stop the iterative gradient descent, are: the minimum error (default value 1.0e-10) and the minimum gradient size (default value 1.0e-10).

4.8.3 Apply the method

Command. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :NON_LINEAR_REGRESSION, command :TRAIN_MLP_REGRESSION or :TRAIN_MLP_CLASSIFICATION.

Effects of the method employment. If the MLP name is *xx* and the *goal-regression* is *yy*:

- The new functional attribute created by default once the model is built has the name *xx* (by default the name is *MLP<a>-<b>-yy* if the model has two hidden layers, <*a*> neurons on the first layer and <*b*> neurons on the second layer)
- The created file containing information about the training process has the name xx.log and is located in the current directory
- The new MLP *xx* is pushed in the global variable *mlp-structures*
- The global variable *current-mlp* keeps the last built (classification or regression) MLP model
- A postscript file named *xx.ps* is generated in the local directory and automatically displayed, representing the MLP structure for one object (by default for the first object in the current *learning-set*).

Interesting displayed information while training the MLP: status variables, training stage with mean squared errors on LS and TS (if test set error monitoring is on) for every cycle, CPU time.

4.8.4 Test the model

Regression

Command. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :NON_LINEAR_REGRESSION, command :TEST_REGRESSION_APPROXIMATION

Before testing, the user is prompted for a numerical attribute name *xx* (for example the new created MLP model), to be compared with the *goal-regression* *yy*. After testing, a scatter-plot is automatically displayed, (*xx*, *yy*) on TS (see Figure 36). The corresponding created postscript file is named *linear_regtst-tem.ps* and is located in the current directory.

Interesting displayed information while describing the results: statistics (mean, max, min, standard deviation, standard error) on: errors and absolute errors; CPU time.

Classification

Command. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :NON_LINEAR_REGRESSION, command :TEST_CLASSIFICATION_APPROXIMATION

Before testing, the user is prompted for a symbolic attribute name *xx* (for example the new created MLP model), to be compared with the *goal-classification* *yy*.

Interesting displayed information while describing the results: non-detection costs (values between 0 and 1), confusion matrix on TS (number of objects correctly classified and misclassified), classification error rate on TS, CPU time.

After testing, the global variable *classification-errors* contains all the misclassified objects.

4.8.5 Results visualization / interpretation

Display MLP. ATDIDT 2.2 command :DRAW_MLP prompts for an object name and displays the current MLP for this object. The command generates a postscript file (named *xx.ps* for a MLP called *xx*) located in the current directory that may be visualized at any time using the GhostView tool. Note that this file is generated and displayed automatically just after each new MLP model training / retraining (but only for the first object in *learning-set*).

Figure 32 draws a regression MLP model for a `*goal-regression*` called "cct-sbs" and Figure 33 draws a classification MLP model for a `*goal-classification*` called "security" (see the attribute definitions in database declaration file example of appendix), both MLPs built on a learning set of 1000 objects and tested on a independent test set of 1000 objects. The networks have as many neurons in the input layer as inputs in both cases, one output neuron in regression and as many output neurons as classes in classification. The numbers marked in each neuron in red colors are valid only for the object the network is applied to, all the others are valid for any object. The lowest number in each neuron represents a measure of the neuron's importance in the model.

Display training curves. ATDIDT 2.2 command `:SHOW_TRAINING_CURVES` prompts for a MLP name and displays training curves on learning and test sets. The test set error will always be zero if the test set monitoring is not enabled. Figure 34 and Figure 35 give the curves for training the regression and classification models of Figure 32 and Figure 33 respectively.

HTML format. ATDIDT 2.2 commands `:SAVE_MLP` and `:INSPECT_MLP` give another way of visualizing results, in html format. For a MLP model named *xx*, the first command creates a new directory called */Sav/xx/* in the current directory, and puts 5 files concerning the MLP in this new directory. The second command opens a *Welcome.html* file that displays general information about the model together with hyperlinks for all the created files:

- *xx-train.lst* - displays information that describe the training processes
- *xx.dump* – outputs the internal lisp structure of the MLP model *xx*
- *xx.lsp* – contains the lisp function of the new created functional attribute
- *xx-sp.pdf* – is the MLP structure display.

Explicit function. Example of nonlinear regression and classification functions deducted from MLP model of Figure 32 and Figure 33 are given in appendix.

Other ideas for graphics. If the MLP name is *xx*, the `*goal-regression*` is *yy* and *ww* is one input attribute:

- Conditional scatter-plot (*xx*, *yy*) on LS (see Figure 38) or TS
- Conditional histogram for *xx* on LS (see Figure 39), TS
- Settle LS as the objects misclassified by the tree `(*classification-errors*)` and apply a conditional histogram *for xx*
- Define a functional attribute *zz* as the error / absolute error / squared error of MLP and visualize a scatter-plot for (*yy*, *zz*) or a histogram for *zz* on LS, TS
- Scatter-plot (*xx*, *ww*) on LS (see Figure 37) or TS.

4.8.6 Features extraction

Definition. Feature extraction methods aim at defining a set of feature (attribute) combinations. The objective is to transform the initial attributes in order to concentrate the maximum amount of information in a minimum number of transformed attributes.

Features extraction by MLP. A MLP is build having the input attributes as MLP inputs and equally as MLP outputs. The hidden neuron activations give thus the compressed set of new functional attributes that concentrate the information of all input attributes. The approach becomes really useful when the number of input attributes is more less than the number of neurons in the hidden layers.

Step 1. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :NON_LINEAR_REGRESSION, command :EXTRACT_FEATURES

Effects. If the MLP name is *xx*:

- The new functional attribute created by default once the model is built has the name *xx* (by default the name is *MLP<a>-<b>-compress* if the model has two hidden layers, <*a*> neurons on the first layer and <*b*> neurons on the second layer)
- The created file containing information about the training process has the name *xx.log* and is located in the current directory
- The new MLP *xx* is pushed in the global variable *mlp-structures*
- The global variable *current-mlp* keeps the last built MLP model
- A postscript file named *xx.ps* is generated in the local directory and automatically displayed, representing the MLP structure for one object (by default for the first object in the current *learning-set*).

Interesting displayed information while training the MLP: status variables, training stage with mean squared errors on LS and TS (if test set error monitoring is on) for every cycle, CPU time.

The commands for testing cannot be employed here. All the others commands related to neural networks may be useful.

Step 2. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :NON_LINEAR_REGRESSION, command :GET_HIDDEN_NEURON_ACTIVATIONS

Effects. The command creates <*i*> new functional attributes, where *i* is the number of the hidden neurons of the current MLP (stored in the variable *current-mlp*). If the MLP name is *xx* and the (regression or classification) goal is *yy*, the new attributes have the name *MLP<a>-<b>-yy-tanh<i>* if the model has two hidden layers, <*a*> neurons on the first layer and <*b*> neurons on the second layer.

Example of the function for such functional attribute (of neural network of Figure 33):

```
Command: (print (get 'mlp10-security-tanh8 'fonction))

Effect: (COERCE (MULTI-OR (LET ((I1
                  (+ (* 0.006016299369492383d0 (PU OBJET))
                    -6.036782344305822d0))
                 (I2
                  (+ (* 0.0020852237395437285d0 (QU OBJET))
                    -0.3143199907124104d0)))
            (TANH (+ -3.47635289612401d0
```

```
                (* -0.4666199692369496d0 I1)
                (* -2.830416533641814d0 I2))))
        '0.0)
'FLOAT)
```

4.8.7 Other possible actions

`:RETRAIN_MLP` – retrains `*current-mlp*` with the currently selected `*learning-set*` and `*test-set*.` It produces a new attribute, symbolic or numerical depending on the `*current-mlp*` type, and a log file with the training information (or append this information to an already existent log file). The training process is restarted from where it stopped not from the scratch.

`:CHOOSE_MLP` – chooses a MLP model; the command may be applied at any time, once a (classification or regression) MLP model is stored in the variable `*current-mlp*.`

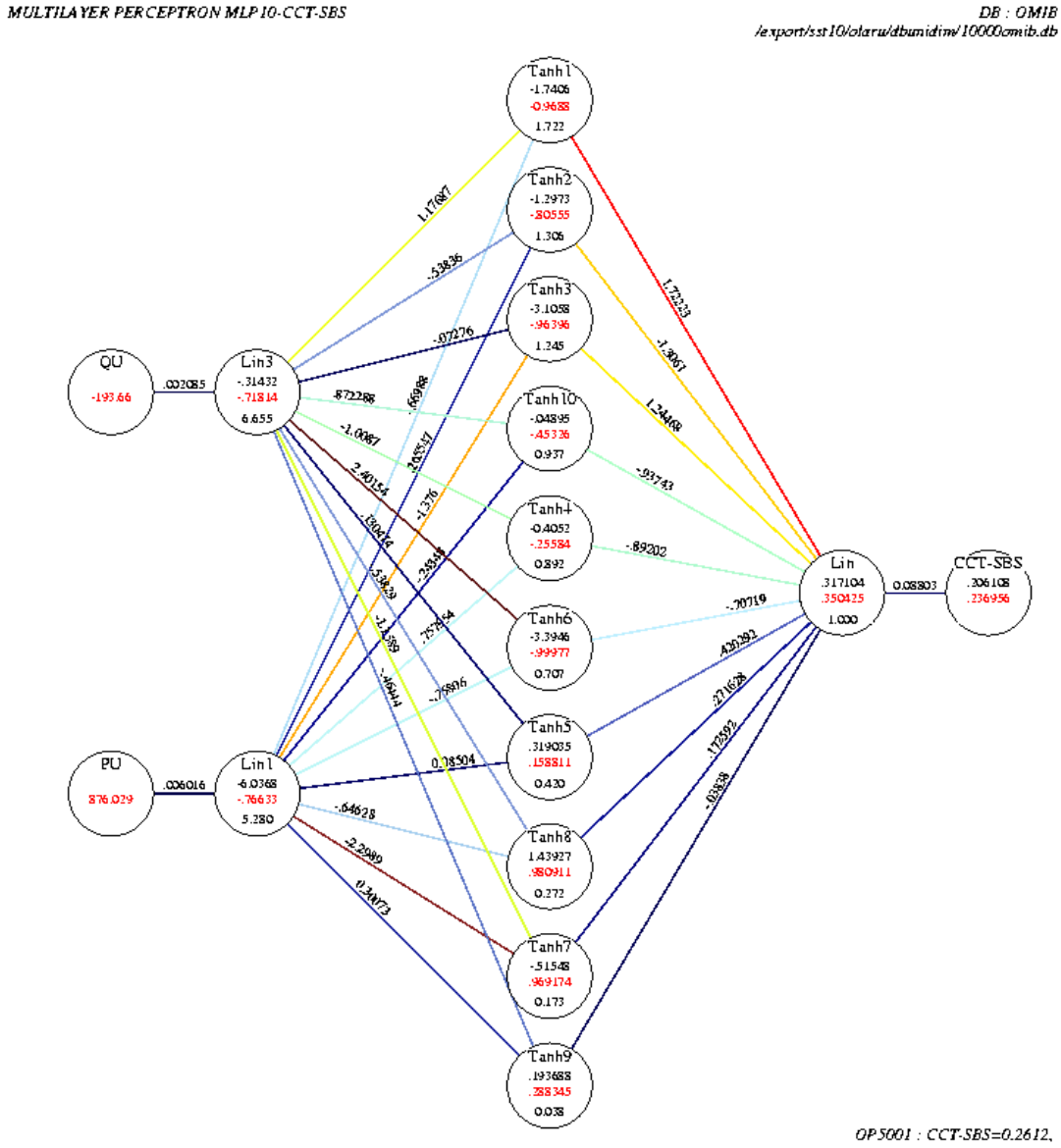

MULTILAYER PERCEPTRON MLP 10-CCT-SBS
DB : OMIB
/export/sst10/olaru/dbunidim/10000omib.db
QU
-193.66
.032085
Lin3
-.31432
-.71814
6.655
PU
876.029
.006016
Lin1
-6.0368
-.76633
5.280
Tanh1
-1.7406
-0.9688
1.722
Tanh2
-1.2973
-.80555
1.306
Tanh3
-3.1058
-.96396
1.245
Tanh10
-.04895
-.45326
0.937
Tanh4
-0.4052
-.25584
0.892
Tanh6
-3.3946
-.99977
0.707
Tanh5
.319035
.158811
0.430
Tanh8
1.43927
.980911
0.272
Tanh7
-.51548
.969174
0.173
Tanh9
.193688
.288345
0.038
Lin
.317104
.350425
1.000
0.08803
CCT-SBS
.206108
.236956
OP 5001 : CCT-SBS=0.2612,

Figure 32

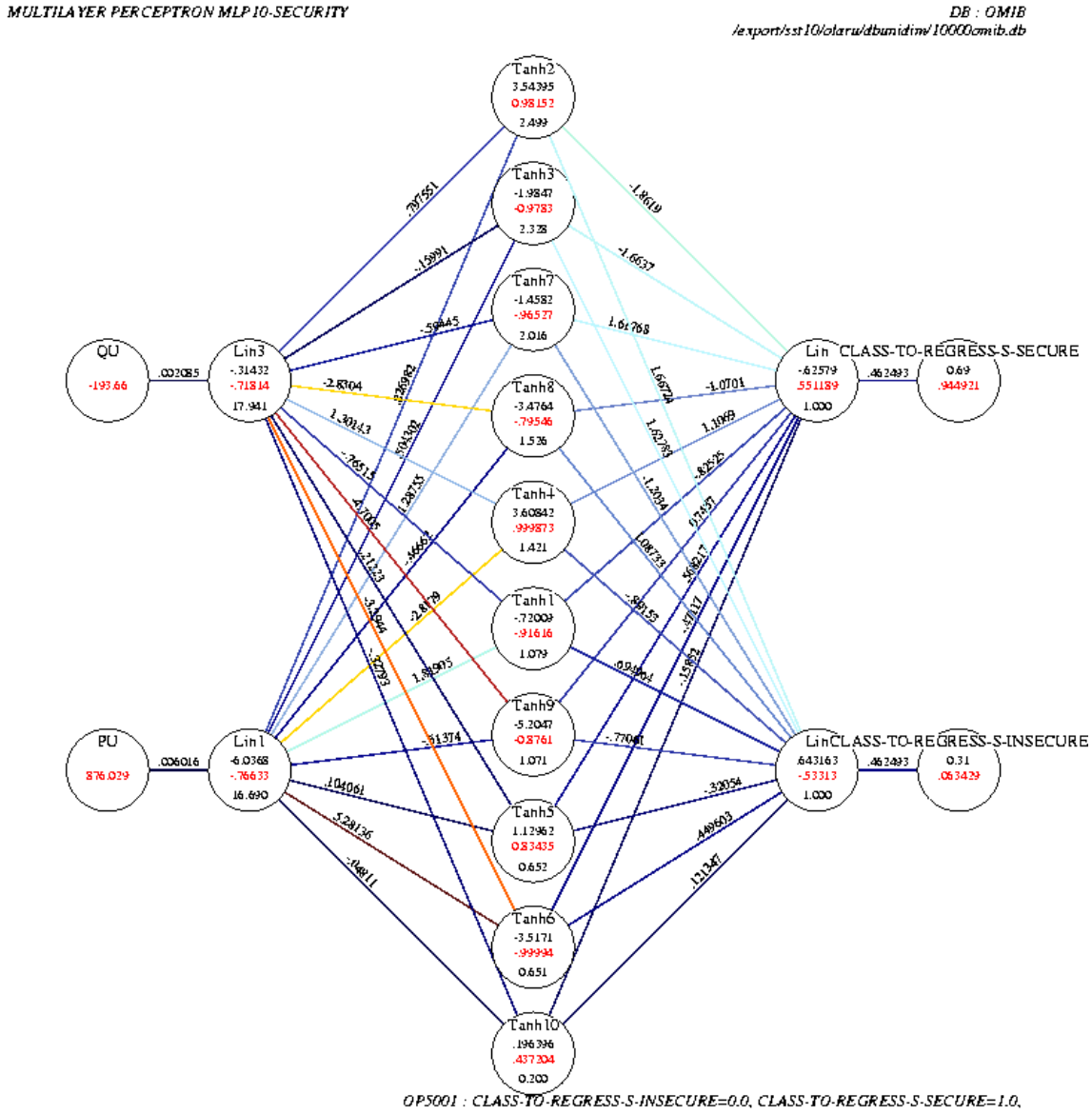

MULTILAYER PERCEPTRON MLP 10-SECURITY
DB : OMIB
/export/sst10/olaru/dbunidim/10000omib.db
QU
-193.66
.032085
Lin3
-.31432
-.71814
17.941
PU
876.029
.006016
Lin1
-6.0368
-.76633
16.690
Tanh2
3.54395
0.98152
2.499
Tanh3
-1.9847
-0.9783
2.328
Tanh7
-1.4582
-.96527
2.016
Tanh8
-3.4764
-.79546
1.536
Tanh4
3.60842
.999873
1.421
Tanh1
-.72009
-.91616
1.079
Tanh9
-5.2047
-0.8761
1.071
Tanh5
1.12962
0.833435
0.652
Tanh6
-3.5171
-.999941
0.651
Tanh10
.196396
.437204
0.200
Lin
-.62579
.551189
1.000
CLASS-TO-REGRESS-S-SECURE
.462493
0.69
.944921
Lin
.643163
-.53313
1.000
CLASS-TO-REGRESS-S-INSECURE
.462493
0.31
.663429
OP5001 : CLASS-TO-REGRESS-S-INSECURE=0.0, CLASS-TO-REGRESS-S-SECURE=1.0,

Figure 33

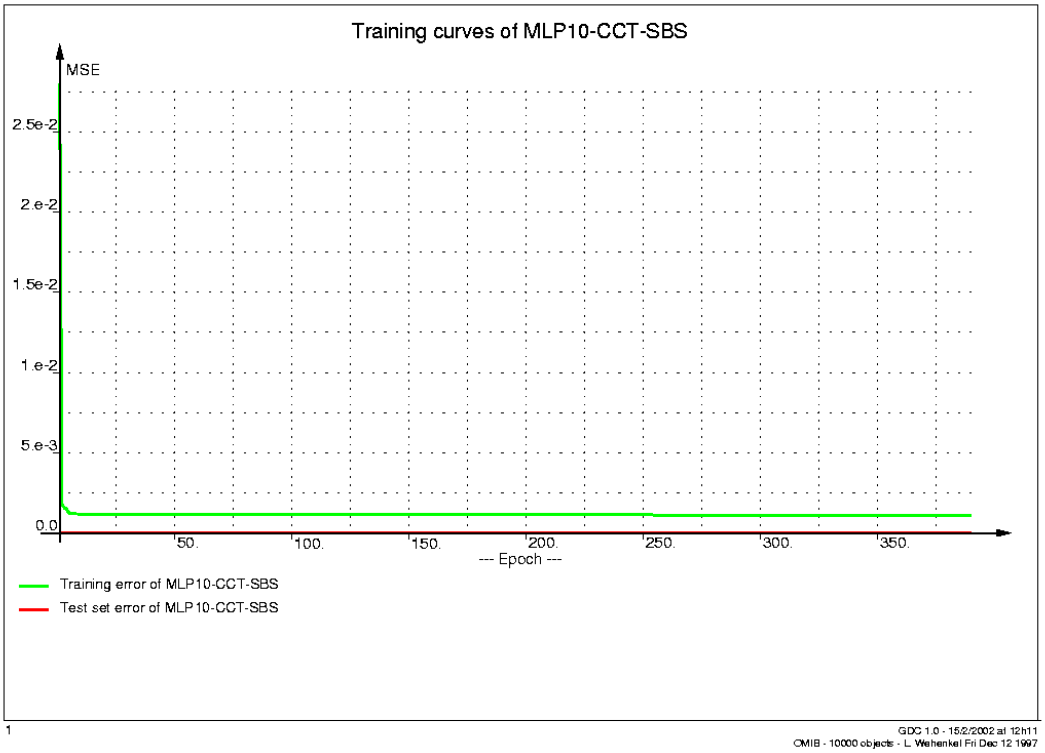


Figure 34

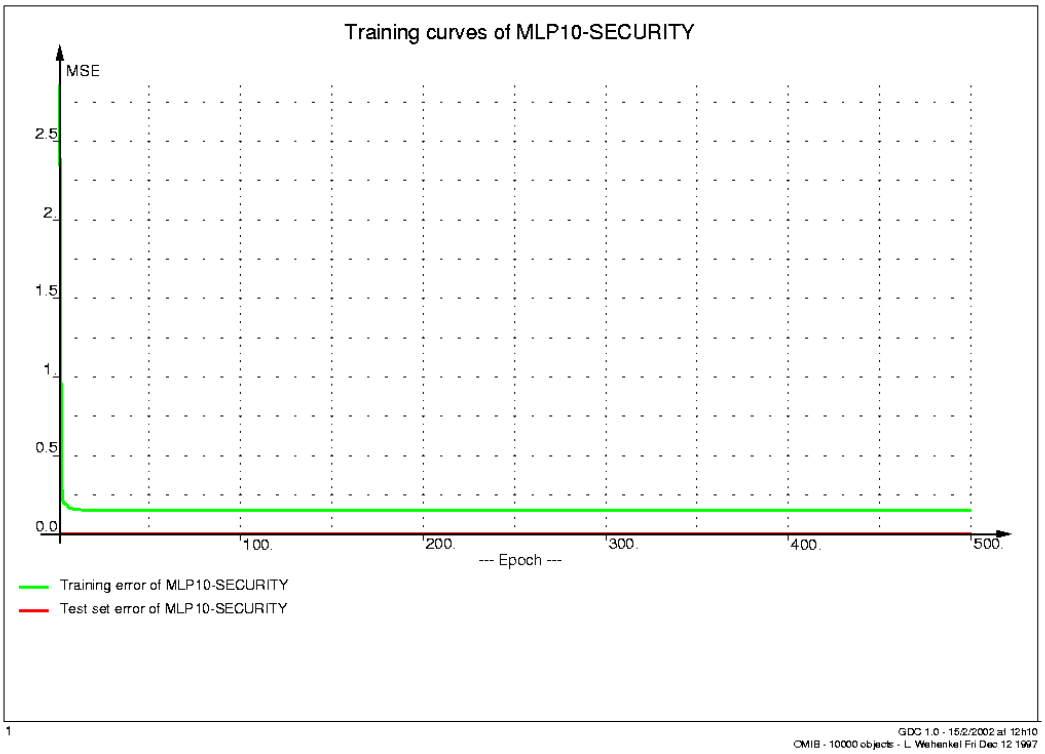


Figure 35

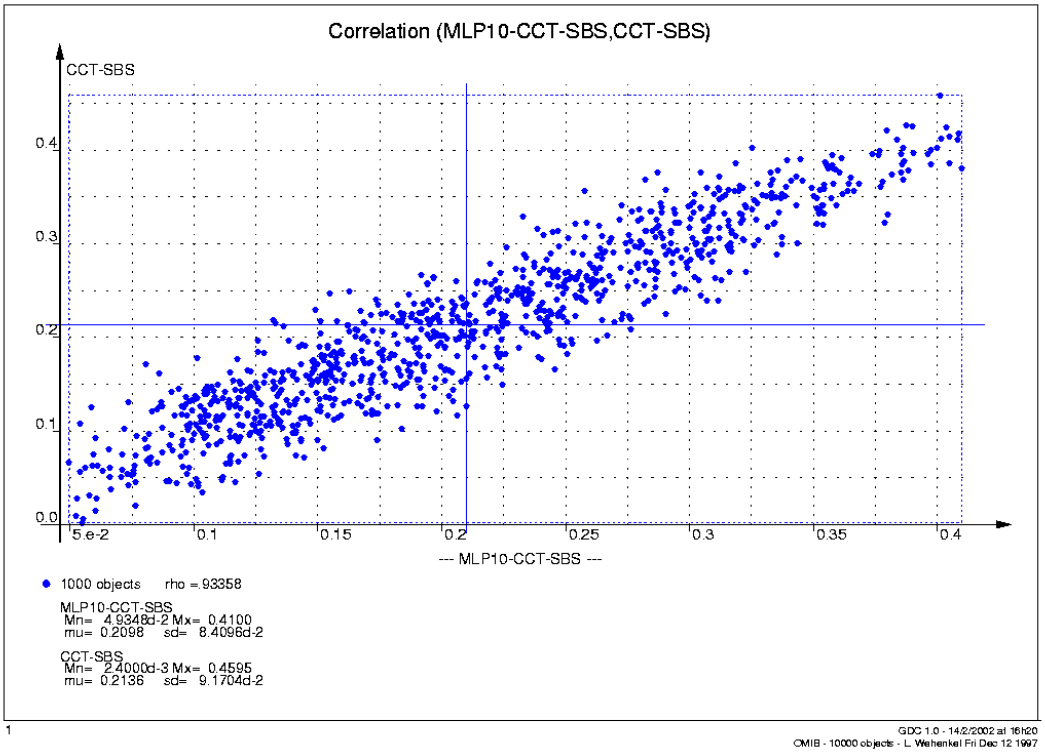


Figure 36

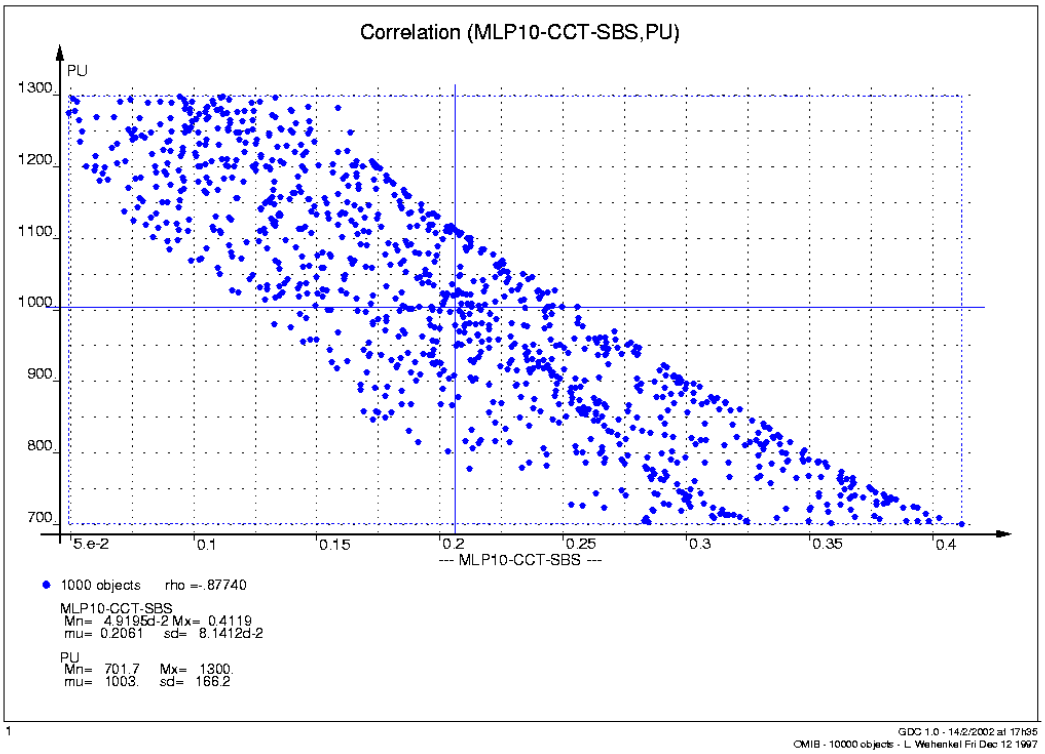


Figure 37

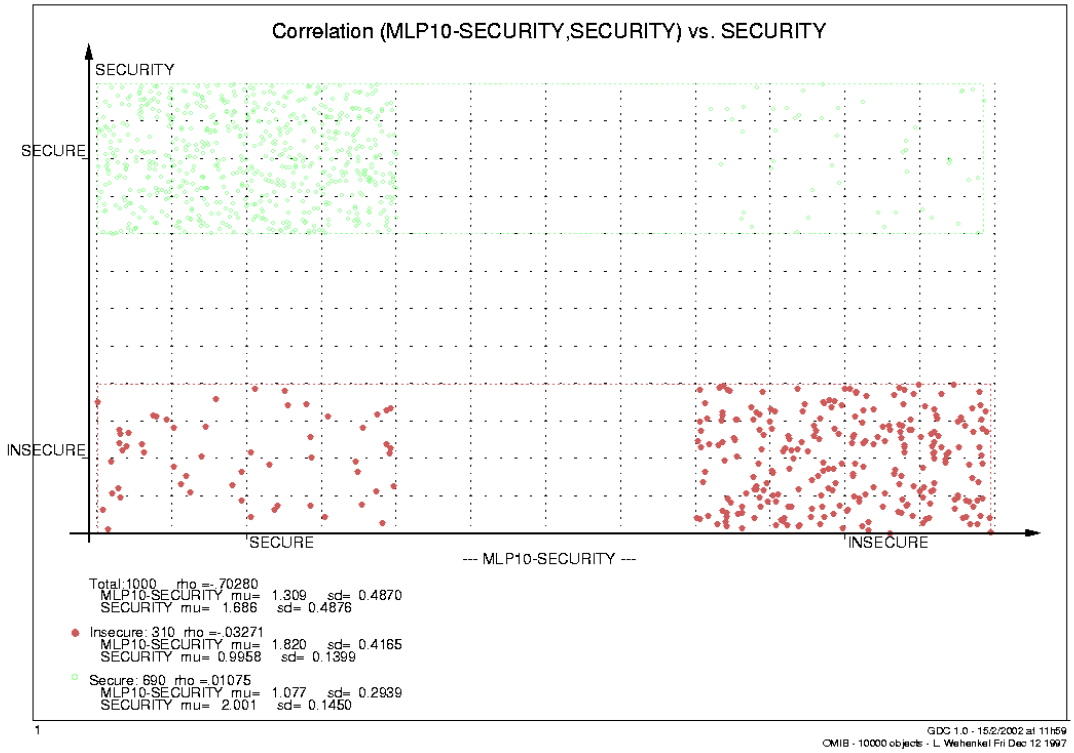


Figure 38

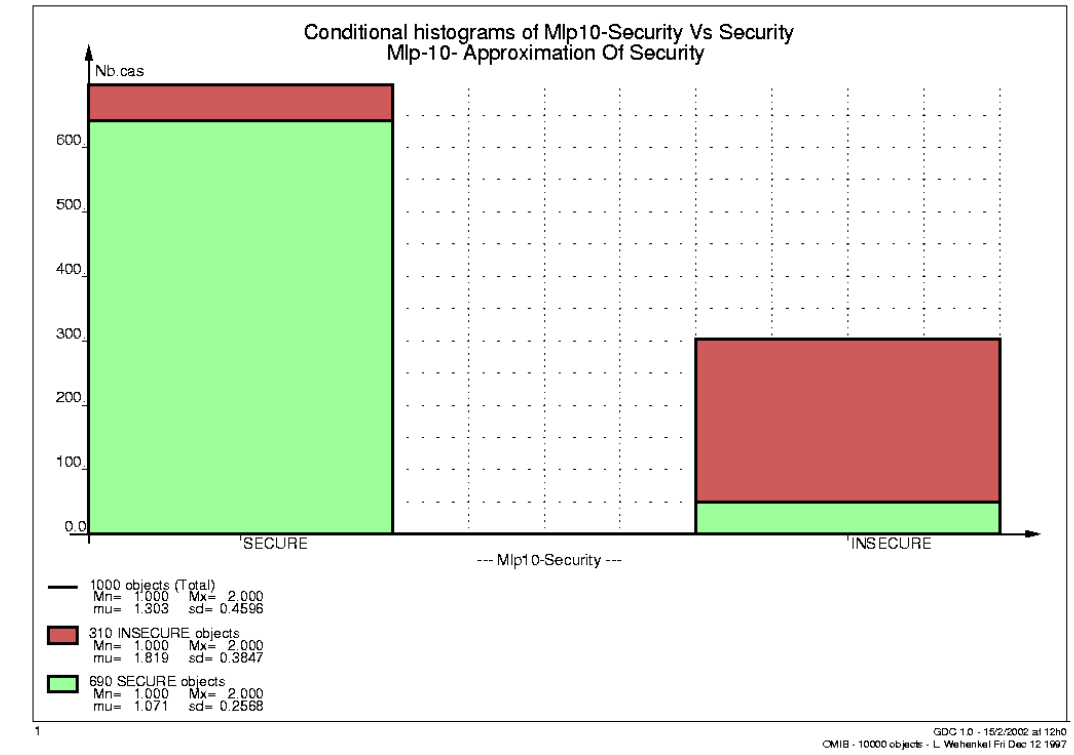


Figure 39

4.9 K-Nearest Neighbors

4.9.1 What is it?

Definition. K-Nearest Neighbors technique is a statistical tool that consists in matching an unseen situation (object) with similar situations (objects) present in the database called nearest neighbors. The unseen object inherits all these nearest neighbors' characteristics, as the value of the numerical output attribute (in regression problems) or the class (in classification problems), also the distance to these nearest neighbors, and generally, any type of information attached to the nearest neighbors. The model supports only numerical attributes and symbolic or numerical output depending on the problem.

Method characteristics. The method is very simple and similar to human reasoning (recalling similar situations seen in the past) thus interpretable. It is less accurate than a MLP and more accurate that regression trees. It is a very slow method. The main disadvantage is that it requires a large number of learning objects. In particular, for high dimensional attribute spaces the method may require prohibitively large samples. Thus, to be effective, a prior feature selection may reduce the input spac (hybrid approaches). For a symbolic output the method uses majority voting among the nearest neighbors, and for numerical output interpolation by the inverse of the squared distance.

4.9.2 Selections to make before starting

Define the problem.

- Choose `*candidate attributes*`. Admissible input attribute type is "ordonee". Note that the model does not handle linear-combination or qualitative attribute values, they being excluded from the `*candidate-attributes*` list prior to attributes' normalization. Temporal attribute values are replaced by a list of scalar ones.
- Choose `*knn-output*`, i.e. the output for the KNN model, a symbolic or numerical attribute. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :SIMILARITY, command :SET_KNN_OUTPUT.

Select data.

- Choose `*knn-reference-set*`, i.e. the set of objects used as learning set. The selection is done exactly as a `*learning-set*` selection. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :SIMILARITY, command :SET_KNN_REFERENCE_SET.
- Choose `*test-set*` (if you also want to test the model).

Transform data. Normalize the attributes from `*candidate-attributes*` list, by computing their standard-deviation in the `*knn-reference-set*`. A list called `*knn-attributes*` is built used to define the Euclidian distance. ATDIDT 2.2 command: menu

:AUTOMATIC_LEARNING, menu :SIMILARITY, command :NORMALIZE_KNN_ATTRIBUTES.

Set method parameters. Method parameter is *knn-k*.

knn-k
- Indicates the number of neighbors effectively used
- Default value 1
- Maximum value 15 (knn-k-max)
- May be settled manually by a command, or automatically by a cross-validation method
- Manually setting by ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :SIMILARITY, command :SET_KNN_K
- Automatically setting by ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :SIMILARITY, command :KNN_CROSS-VALIDATION_TEST. The command applies leave-one-out method to *knn-reference-set*, for *knn-k* increasing from 1 to knn-k-max and automatically sets *knn-k* to the value which yielded the best accuracy. Note that the algorithm is quadratic computationally in the size of the *knn-reference-set*, that is why this command is very slow. The command displays for every value of K: in the case of numerical output, statistics (mean, max, min, standard deviation, standard error) on errors and absolute errors, total CPU time, and in the case of symbolic output, confusion matrix, test set error rates and total CPU time.

4.9.3 Apply the method

Command. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :SIMILARITY, command :FIND_NEAREST_NEIGHBORS

Effects of the method employment.
- The command prompts for an *object-name* and searches the 15 nearest neighbors of object *object-name*, selecting in the variable *learning-set* the most similar *knn-k* objects together with the considered object. These objects may be inspected afterwards by the ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :SIMILARITY, command :VIEW_OBJECTS or by other graphics.
- If the *knn-output* is *yy*, the new functional attribute created by default once the KNN model is built has the name *knn-approx-of-yy* (a numerical value if *yy* is numerical, otherwise a class).
- No log file generated.

4.9.4 Test the model

Command. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :SIMILARITY, command :KNN_TEST_SET_TEST

The command compares *KNN-output* *yy* with the output approximated by the KNN model, *knn-approx-of-yy*. It displays in the case of numerical output, statistics (mean, max, min, standard deviation, standard error) on errors and absolute errors, total CPU time and in the case of symbolic output, confusion matrix, test set error rate andtotal CPU time.

4.9.5 Results visualization / interpretation

Visualizing the *knn-k* nearest neighbors of an object. ATDIDT 2.2 command :VIEW_OBJECTS displays information and graphics for every of the *knn-k* nearest neighbors, objects selected in *learning-set* once built the model.

Other ideas for graphics. If the *knn-output* is *yy* and *xx* is one input attribute:

- Conditional/normal scatter-plot (*knn-approx-of-yy, yy*) on LS (settled as *knn-reference-set*, see Figure 40 and Figure 41), TS, or *knn-k* nearest neighbors
- Conditional/normal histogram for *knn-approx-of-yy* on LS, TS, or *knn-k* nearest neighbors
- Scatter-plot (*knn-approx-of-yy, xx*) on LS (see Figure 42), TS, or *knn-k* nearest neighbors

Statistics. For a *knn-output* *yy*, the ATDIDT 2.2 command :KNN_STATISTICS defines a new functional attribute called *error-of-knn-yy* reflecting the absolute error between KNN model output *knn-approx-of-yy* and the reference output *yy*. The command also displays scatter-plots for every nearest neighbor of this *error-of-knn-yy* attribute, in terms of the distance to the neighbor (see Figure 43). The corresponding generated postscript file is named *knn_stats-tem.ps* and is located in the current directory.

4.9.6 Other possible actions

:HYBRID_DT_KNN – allows to inherit in a single step all the parameters *knn-reference-set*, *knn-attributes* and *knn-output* from a previously built decision or regression tree (the *current-dt*). Thus, the next KNN model built will consider only the attributes selected by the tree. The command is valuable especially for high dimensional input spaces.

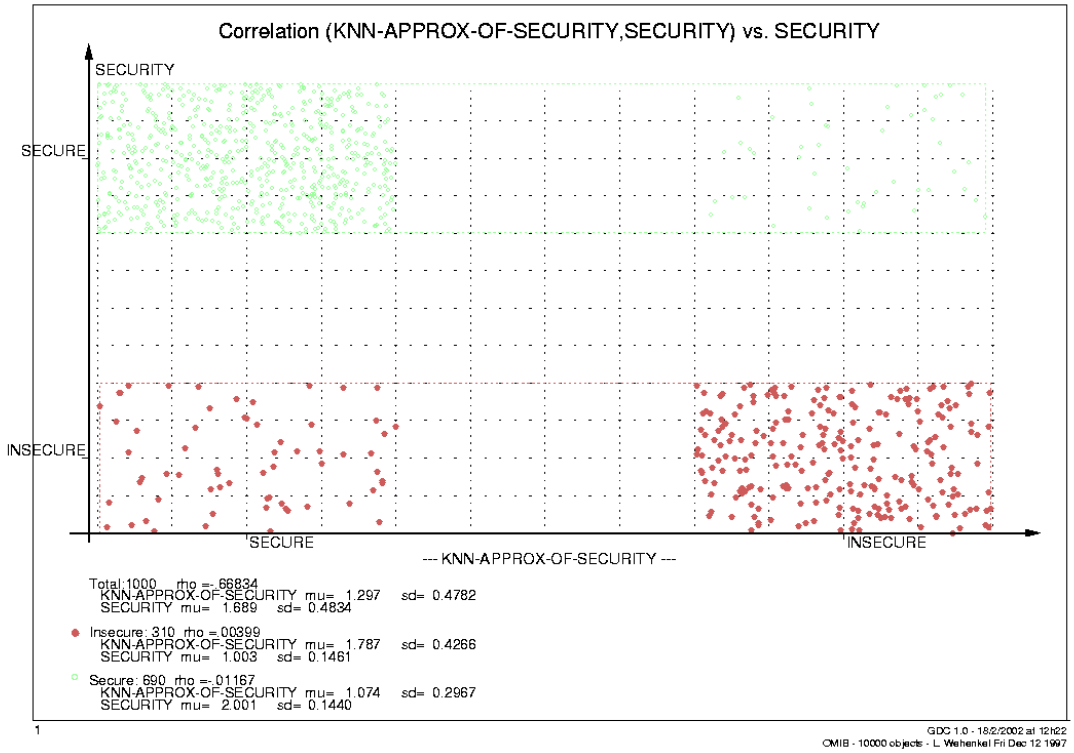


Figure 40

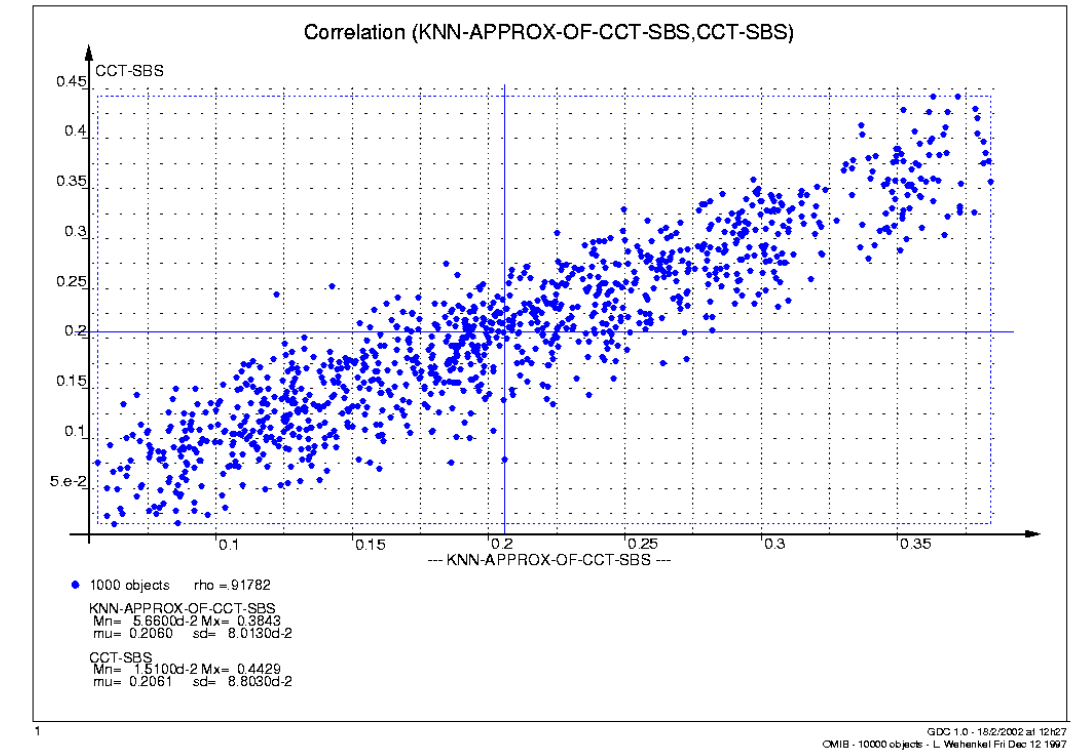


Figure 41

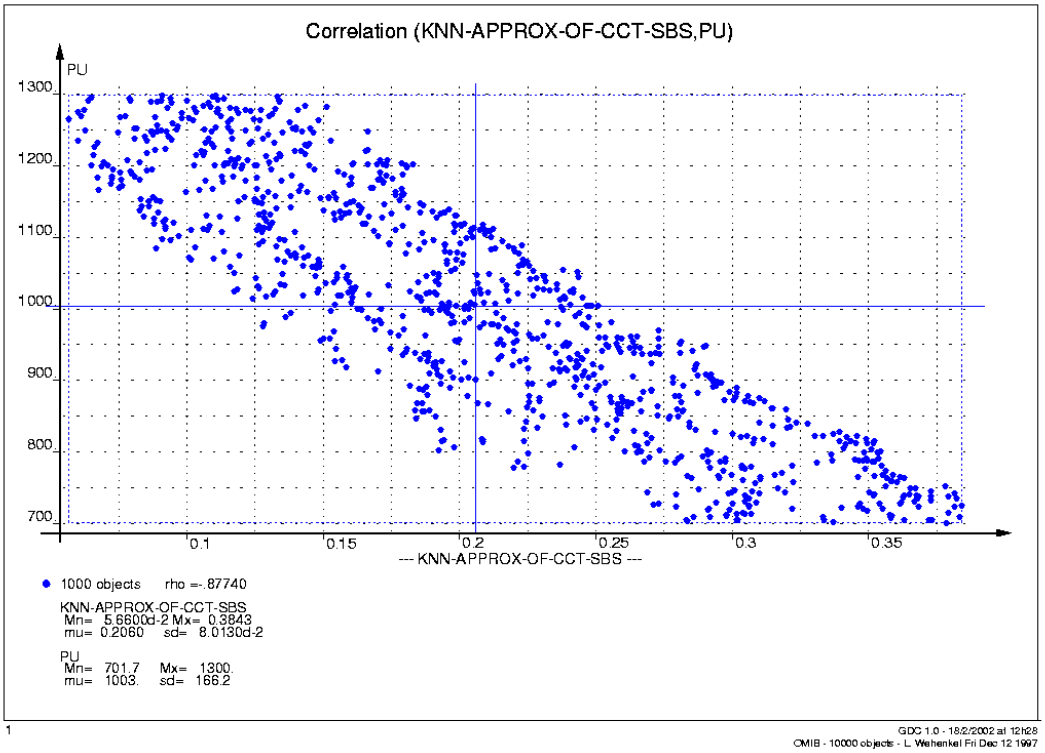
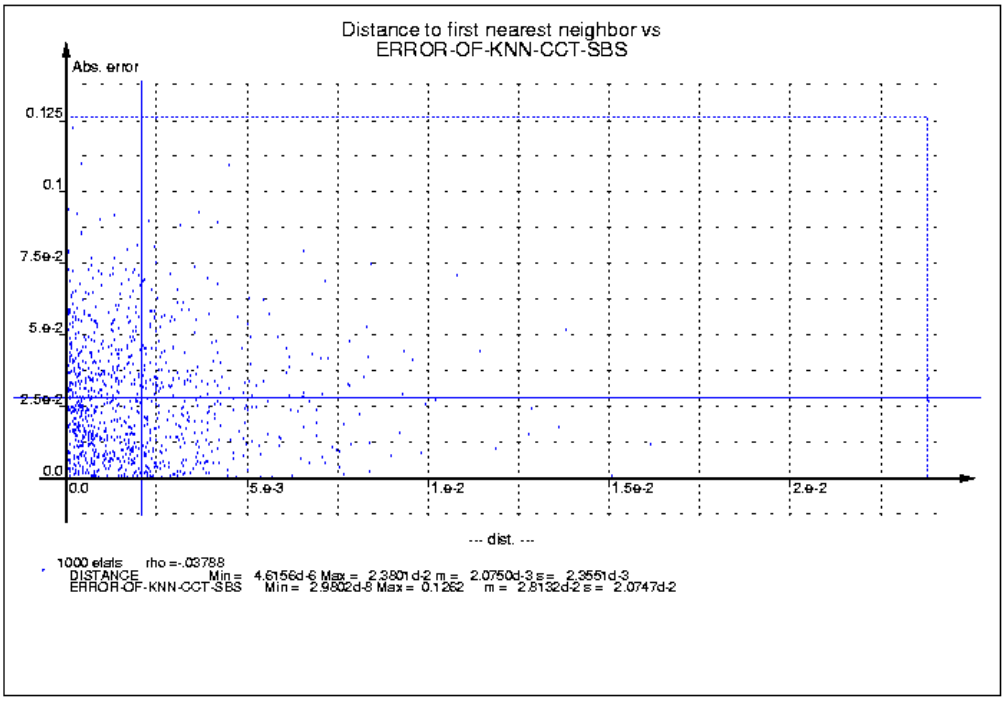
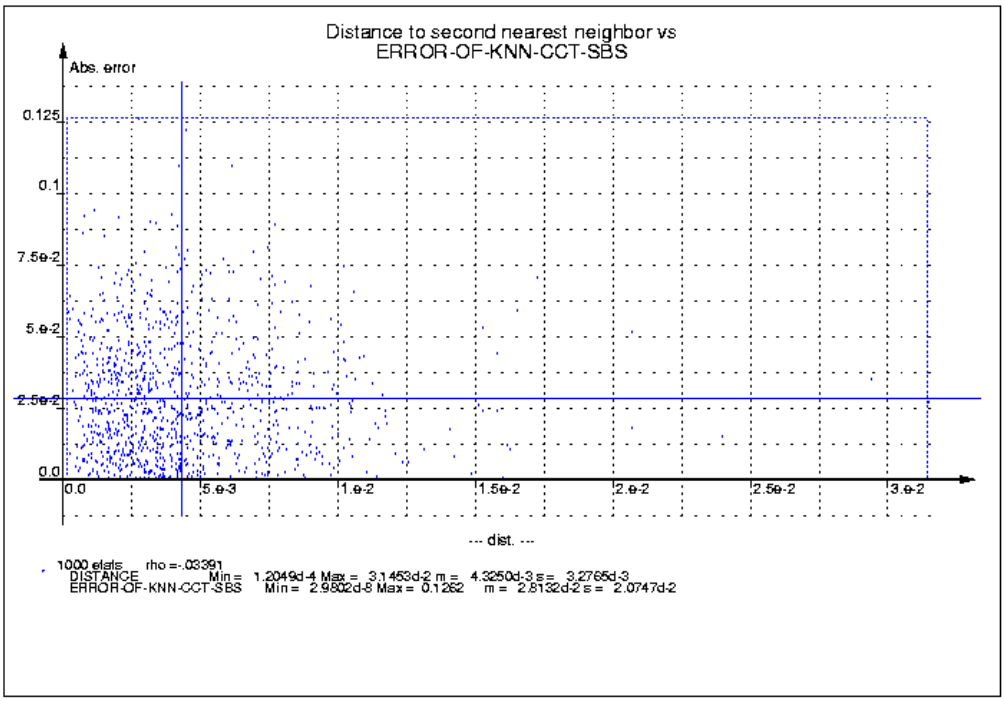

Correlation (KNN-APPROX-OF-CCT-SBS,PU)
PU
1300
1200
1100
1000
900
800
700
0.1
0.15
0.2
0.25
0.3
0.35
--- KNN-APPROX-OF-CCT-SBS ---
1000 objects rho =-.87740
KNN-APPROX-OF-CCT-SBS
Mn= 5.6600d-2 Mx= 0.3843
mu= 0.2060 sd= 8.0130d-2
PU
Mn= 701.7 Mx= 1300.
mu= 1003. sd= 166.2
1
GDC 1.0 - 18/2/2002 at 12h28
CMIB - 10000 objects - L. Wehenkel Fri Dec 12 1997

Figure 42





1

GDC 1.0 - 18/2/2002 at 12h29
OMIB - 10000 objects - L. Wehenkel Fri Dec 12 1997

Figure 43

4.10 K-Means

4.10.1 What is it?

Definition. K-Means technique is a statistical tool useful for clustering a large number of objects into a small number of groups based on numerical input attributes. It is not oriented towards a particular prediction task. It tries to find by itself, the existing relationships among objects characterized by a set of input attributes. The procedure determines a set of K clusters, where K is a priori fixed by the user.

Method characteristics. Like any unsupervised learning method, it becomes really useful in the context of large-scale databases, with many objects and many attributes. It is a slow method. Scatter-plots are useful tools in order to visualize the clusters.

4.10.2 Selections to make before starting

Define the problem. Choose `*candidate attributes*`. Admissible input attribute type is "ordonee". Note that the model does not handle linear-combination or qualitative attribute values, they being excluded from the `*candidate-attributes*` list prior to attributes' normalization. Temporal attribute values are replaced by a list of scalar ones.

Select data. Choose `*knn-reference-set*`, i.e. the set of objects used as learning set. The selection is done exactly as a *learning-set* selection. ATDIDT 2.2 command: menu `:AUTOMATIC_LEARNING`, menu `:SIMILARITY`, command `:SET_KNN_REFERENCE_SET`.

Transform data. Normalize the attributes from `*candidate-attributes*` list, by computing their standard-deviation in the `*knn-reference-set*`. A list called `*knn-attributes*` is built used to define the Euclidian distance. ATDIDT 2.2 command: menu `:AUTOMATIC_LEARNING`, menu `:SIMILARITY`, command `:NORMALIZE_KNN_ATTRIBUTES`.

Set method parameters. Method parameter is `*k-means-k*`.

k-means-k

- Indicates the number of clusters
- Default value 5
- Takes values between 0 and 50
- ATDIDT 2.2 command: menu `:AUTOMATIC_LEARNING`, menu `:SIMILARITY`, command `:SET_KMEANS_K`

4.10.3 Apply the method

Command. ATDIDT 2.2 command: menu :AUTOMATIC_LEARNING, menu :SIMILARITY, command :RUN_KMEANS

Effects of the method employment.

- It creates *k-means-k* functional attributes called *distance-to-cluster-<i>*, that give for every object, the distance of the object to the cluster <*i*>
- It creates the functional attribute *nearest-cluster* that gives for every object the name of the nearest cluster to the object: *cluster-<i>*
- It creates the functional attribute *nearest-cluster-yy* that gives for every object the output *yy* (numerical or symbolic) approximated by the nearest cluster to the object
- Creates a global variable called *cluster-centers* containing statistics on LS for every created cluster of objects on input attributes
- It generates a lisp file *cluster-saves.lsp* that defines each cluster as a list of objects
- No log file generated.

Interesting displayed information while clustering: status variables, statistics for every created cluster of objects on input attributes.

4.10.4 Results visualization / interpretation

Visualizing the clusters. ATDIDT 2.2 command :DRAW_CLUSTERS generates automatically a scatter-plot of the clusters taking the *candidate-attributes* two-by-two, thus resulting $\frac{n(n-1)}{2}$ graphics if *n* is the number of attributes (see Figure 44).

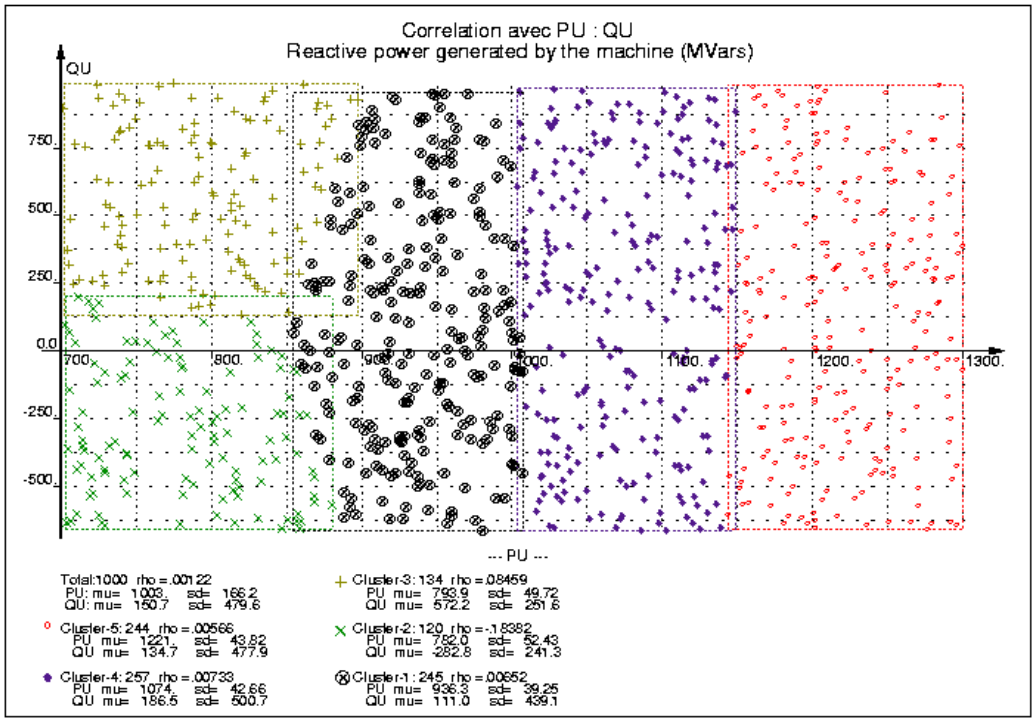

Correlation avec PU : QU
Reactive power generated by the machine (MVars)
QU
750.
500.
250.
0.0
-250.
-500.
700.
800.
900.
1000.
1100.
1200.
1300.
--- PU ---
Total:1000 rho = .00122
PU: mu= 1003. sd= 166.2
QU: mu= 150.7 sd= 479.6
Cluster-5: 244 rho = .00566
PU mu= 1221. sd= 43.82
QU mu= 134.7 sd= 477.9
Cluster-4: 257 rho = .00733
PU mu= 1074. sd= 42.66
QU mu= 186.5 sd= 500.7
Cluster-3: 134 rho = .08459
PU mu= 793.9 sd= 49.72
QU mu= 572.2 sd= 251.6
Cluster-2: 120 rho = -.18382
PU mu= 782.0 sd= 52.43
QU mu= -282.8 sd= 241.3
Cluster-1 : 245 rho = .00652
PU mu= 936.3 sd= 39.25
QU mu= 111.0 sd= 439.1
1
GDC 1.0 - 18/2/2002 at 14h0
OMIB - 10000 objects - L. Wehenkel Fri Dec 12 1997

Figure 44

4.11 Comparative table

All the presented data mining methods have been compared from the point of view of test set errors and CPU times. Table 5 gives an idea of the comparison. The models were built in the following conditions:

- Database – OMIB 10.000 objects
- `*goal-regression*` - cct-sbs
- `*goal-classification*` - security
- `*learning-set*` - (from 5001 6000)
- `*test-set*` - (last 1000)
- `*candidate-attributes*` - (pu, qu)
- All the models parameters leaved as by default.

Table 5

DM Method	DM learning task	Error: MAE or Pe(%)	CPU time (seconds)
MLP	Classification	11.1%	13.2
KNN	Classification	11.3%	1.4
Decision Tree	Classification	12.6%	2.0
MLP	Regression	0.026495	3.2
Regression Tree Boosting	Regression	0.026908	1.1
Linear Regression	Regression	0.027909	0.3
KNN	Regression	0.028132	1.4
Regression Tree Bagging	Regression	0.030672	4.9
Regression Tree	Regression	0.031697	1.5
Linear Hinges Model	Regression	0.041598	0.4

At a graphical perception, you may compare the DM methods estimators for the same task by comparing the following graphics:
- Figure 18, Figure 38, Figure 40;
- Figure 19, Figure 39;
- Figure 22, Figure 24, Figure 26, Figure 28, Figure 30, Figure 36, Figure 41;
- Figure 23, Figure 25, Figure 27, Figure 29, Figure 31, Figure 37, Figure 42.

5 Operational, practical and useful information

5.1 DM tips

- It is advisable the user does not include into the input attributes list `*candidate-attributes*` the output attribute `*goal-regression*` or `*goal-classification*` in the case of DM techniques for regression and classification tasks.
- Make sure that no attribute is repeating in the `*candidate-attributes*` list, case in which any model based on this global variable would do the job needlessly for this attribute more than once.

5.2 Ideas for hybrid methods

In the case of high dimensional input spaces, hybrid methods allow one to significantly reduce the time required to build models, and/or better tailor the model complexity to the problem at hand avoiding structure optimization task, and/or improve the model accuracy with respect to the "pure" methods.

DT+MLP or RT+MLP. First settle as `*candidate-attributes*` the attributes selected by a tree and then train a MLP with this inputs. ATDIDT 2.2 commands:

- build a DT or a RT
- menu `:AUTOMATIC_LEARNING`, menu `:TREE_INDUCTION`, command `:SELECT-DT-TEST-ATTS`
- menu `:NON_LINEAR_REGRESSION`, command `:TRAIN_MLP_REGRESSION` or `:TRAIN_MLP_CLASSIFICATION`

Dendrogram+MLP. First draw a dendrogram and select the most correlated attributes with the output one and then build a MLP on these attributes. ATDIDT 2.2 commands:

- menu `:DATA_BASE`, menu `:GRAPHICS`, command `:DENDROGRAMS`
- select manually the `*candidate-attributes*` list based on dendrogram results
- menu `:NON_LINEAR_REGRESSION`, command `:TRAIN_MLP_REGRESSION` or `:TRAIN_MLP_CLASSIFICATION`

DT+KNN or RT+KNN. ATDIDT 2.2 command:

- build a DT or a RT
- menu `:AUTOMATIC_LEARNING`, menu `:SIMILARITY`, command `:HYBRID_DT_KNN`
- menu `:AUTOMATIC_LEARNING`, menu `:SIMILARITY`, command `:FIND_NEAREST_NEIGHBORS`

Dendrogram+KNN. First draw a dendrogram and select the most correlated attributes with the output one and then build a KNN model on these attributes. ATDIDT 2.2 commands:

- menu :DATA_BASE, menu :GRAPHICS, command :DENDROGRAMS
- select manually the *candidate-attributes* list based on dendrogram results
- menu :AUTOMATIC_LEARNING, menu :SIMILARITY, command :FIND_NEAREST_NEIGHBORS

5.3 Useful functions/commands

ATDIDT 2.2. In order to activate a command you may click with the left mouse button on the command and then enter, or you may click on the middle button of the mouse once positioned on the command. By clicking on the right button of the mouse you get some help concerning the command.

inferior-lisp window is the space provided for interactive use of ACL lisp. Any lisp command should be introduced here.
The lisp commands are not case-sensitive.

(attribute-name object-name) – gives the value of attribute attribute-name for object object-name;
(describe 'attribute-name) – provides information about attribute attribute-name (type, values, file in which is defined, file in which is stored, lisp function if it is a functional attribute);
(describe 'function-name) – provides information about ATDIDT function function-name. The same effect has the command CTRL-C,CTRL-D which prompts for the function name;
(describe 'object) – provides information about object object (all its attribute values);
M-? – being positioned on a symbol, returns information about the symbol; it has identical effect as describe.
(apropos 'string) – prints out all the function / attribute / global variable names that contain the string of characters string;
M-x describe-function – prompts for a lisp function name and returns short documentation about it. The same effect has the command CTRL-H,F;
F1,f – prompts for a lisp function name and displays a short documentation about it;
CTRL-H,CTRL-F – prompts for an emacs command and displays a short documentation about it;
M-/ - finds possible ends for the current string in all the emacs buffers.
CTRL-C,CTRL-K – prompts for an file name and compiles the file;
CTRL-C,CTRL-L – prompts for an file name and loads the file;
? – tape this at any command prompt in *inferior-lisp* buffer or in the *mini-buffer* in order to get some help about the possible choices;

Interaction with *inferior-lisp* buffer. Once an error occurred in the lisp environment, the command line changes from DB-name> (e.g. OMIB>) to [number] DB-name>. If you want more details on the error (where it has been encountered and its kind) you may tape :zoom or :dn. In order to exit this debug mode, you may tape :pop

number or :reset, (:q in older versions of TUTORIAL), and for stepping up one level :pop.
To interrupt a current process / command you may type CTRL-C,CTRL-C.

Create a clone. If the user wishes to interrupt the data mining process, he can save the whole context in a "clone" (an executable image containing the software modules, the initial data and the results produced in the meanwhile), and restart the clone later on.

Status variables. The software environment maintains a list of a certain number of global variables, called status variables, which stores the main information concerning current selections (of attributes and objects) and methods parameters. They may be consulted at any time to get the information about the present state of the system settings, by the ATDIDT 2.2 command: menu :DATA_BASE, menu :ATTRIBUTES_SELECTION, command :STATUS_VARIABLES.

ATDIDT 2.2 command: menu :DATA_BASE, menu :ATTRIBUTES_SELECTION, command:CLEAR_GCL-BUFFER clears the text in *inferior-lisp* buffer.

6 User interface

To be completed.

7 References

Marée R. *Fonctionnalités et architecture*. University of Liege, Stochastic Methods Department, July 2001.

Marée R. *ATDIDT 3. Description et évaluation du noyau de gestion de données*. University of Liege, Stochastic Methods Department, September 2001.

Olaru C. Geurts P. and Wehenkel L. *Data Mining Tools and Applications in Power System Engineering*. In Proceedings of PSCC, Trondheim, Norway, Volume 1, pages 324-330, June 28 - July 2nd, 1999.

Olaru C. and Wehenkel L. *Data Mining*. IEEE Computer Applications in Power, Volume 12, Number 3, pages 19-25, July 1999.

Wehenkel L. *Automatic learning techniques in power systems*. Boston, Kluwer Academic, 1998.

Wehenkel L. and Druet Ch. *ATDIDT User's Guide (version 2.x)*. University of Liege, Stochastic Methods Department, 2000.

Wehenkel L. *GTDIDT (Version 1.0). Organisation du logiciel et documentation des structures de données*. University of Liege, Electrical Circuits Department, January 1997.

8 Appendix

8.1 Example of Database

8.1.1 Data file

Long-way database load – version 1

876.029 -193.66 0.2358
1110.88 -423.19   0.2104
980.132 79.7223 0.2241
974.139 217.073 0.1577
…
1241.88 -442.25 0.0718

Long-way database load – version 2 (it contains also object numbers)

1 876.029 -193.66 0.2358
2 1110.88 -423.19   0.2104
3 980.132 79.7223 0.2241
4 974.139 217.073 0.1577
…
5000 1241.88 -442.25 0.0718

Short-way database load – version 1

;--++ This is javadb type file
;;; Attribute values of db OMIB2
omib2
pu numerical qu numerical cct-sbs numerical
876.029 -193.66 0.2358
1110.88 -423.19   0.2104
980.132 79.7223 0.2241
974.139 217.073 0.1577
…
1241.88 -442.25 0.0718

Short-way database load – version 2 (it contains also object names)

;--++ This is javadb type file
omib3
object name pu numerical qu numerical cct-sbs numerical
OP1 876.029 -193.66 0.2358
OP2 1110.88 -423.19   0.2104
OP3 980.132 79.7223 0.2241
OP4 974.139 217.073 0.1577
…
OP5000 1241.88 -442.25 0.0718

8.1.2 Database declaration file – long-way database load

```
;;; Database definition : 5000 objects, 4 explicit attributes, 4 functional attributes

(DECLARE-BD
  omib "Example of database declaration"
  :OBJETS (integer 1 5000)          ;; or in version 2  :OBJETS (prefixes “OP” 1 5000)
  :RE-INITIALISER nil
  :ATTRIBUTS-EXPLICITES
  ((pu "Generated active power (MW)"
       :valeurs (real 700.0 1300.0) :par-defaut 1000.0 :type ordonne)
   (qu "Generated reactive power (MVar)"
       :valeurs (real -665.0 990.0) :par-defaut 0.0 :type ordonne)
   (cct-sbs "Critical Clearing Time (msec)"
            :valeurs (real 0.0 2.0) :par-defaut 0.0 :type ordonne))
  :ATTRIBUTS-EXPLICITES-TEMPORELS
  ((delta "Rotor angle of machine (fault cleared at t=155ms)"
          :time (0.00 0.06 0.12 0.18 0.24 0.30 0.36 0.42 0.48 0.54 0.60 0.66 0.72 0.78 0.84 0.90 0.96 1.02 1.08
1.14 1.20 1.26 1.32 1.38 1.44 1.50 1.56 1.62 1.68 1.74 1.80 1.86 1.92 1.98 2.04 2.10 2.16 2.22 2.28 2.34 2.40
2.46 2.52 2.58 2.64 2.70 2.76 2.82 2.88 2.94 3.00)
          :valeurs (real) :type (ordonne time)))
  :ATTRIBUTS-FONCTIONS
  ((security "Security class, function of cct-sbs and *tau*"
             :valeurs (member insecure secure) :par-defaut insecure :type qualitatif-quinlan
             :fonction (if (<= (cct-sbs objet) *tau*) 'insecure 'secure))
   (Pu+b*Qu "Linear combination between Pu and Qu"
            :valeurs (+ pu (* alfa qu)) :type linear-combination
            :fonction (+ (pu objet) (* *ponderation* (qu objet))))
   (delta-after-fault "Rotor angle after fault clearing at t=155ms"
                      :valeurs (real 0.0 150.0) :type ordonne
                      :fonction (delta objet 0.155))
   (cct-disk "Discretized CCT, function of cct-sbs and *tau* "
             :valeurs (member <80 80...200 200…320 >320)
             :par-defaut insecure :type ordonne
             :fonction (if (<= (cct-sbs objet) .08) '<80
                           (if (<= (cct-sbs objet) .20) '80...200
                                   (if (<= (cct-sbs objet) .32) '200...320 '>320)))))
   :CHARGEMENT
   (((pu qu cct-sbs)
     :dans (vms-file "omib-data-file.dat")
     :format (objet pu qu cct-sbs))
    ((delta) :suffix delta-file-name))))


;;; Load attributes instruction

(load-attributes
  omib-attribute-values "This set contains all objects for which the attributes have been loaded"
  :bd omib
  :objets t
  :attributs (pu qu cct-sbs delta))
```

8.2 Example of non-linear function detected by a MLP model

8.2.1 Nonlinear regression (MLP of Figure 32)

```
;;; Lisp code for MLP10-CCT-SBS

(defun MLP10-CCT-SBS (objet)
  (LET ((I1
         (+ (* 0.006016299369492383d0 (PU OBJET))
           -6.036782344305822d0))
        (I2
         (+ (* 0.0020852237395437285d0 (QU OBJET))
           -0.3143199907124104d0)))
    (LET ((I1
           (TANH (+ -1.7405822173950702d0 (* -0.6698848359891705d0 I1)
                    (* 1.1768705781157776d0 I2))))
          (I2
           (TANH (+ -1.297346698294738d0 (* 0.26554690500398404d0 I1)
                    (* -0.5383596578361837d0 I2))))
          (I3
           (TANH (+ -3.1058438580764975d0 (* -1.376040755368317d0 I1)
                    (* -0.07276481055133555d0 I2))))
          (I4
           (TANH (+ -0.4052018775059025d0 (* 0.757953980013272d0 I1)
                    (* -1.0086983106073752d0 I2))))
          (I5
           (TANH (+ 0.31903477877975633d0 (* 0.08503960764031138d0 I1)
                    (* 0.13047441296479204d0 I2))))
          (I6
           (TANH (+ -3.394615949290037d0 (* -0.7680560439363036d0 I1)
                    (* 2.4015419701771585d0 I2))))
          (I7
           (TANH (+ -0.5154813092875247d0 (* -2.2989156249329032d0 I1)
                    (* -1.1589011083475231d0 I2))))
          (I8
           (TANH (+ 1.4392749802737381d0 (* -0.64627781470995d0 I1)
                    (* -0.5382878434723491d0 I2))))
          (I9
           (TANH (+ 0.19368775267680255d0 (* 0.30073036955597904d0 I1)
                    (* -0.4644350742563427d0 I2))))
          (I10
           (TANH (+ -0.04894697702940679d0
                    (* -0.24347791732672294d0 I1)
                    (* 0.87228827947917 6d0 I2)))))
      (LIST (+ (* (IDENTITE (+ 0.3171044977566709d0
                               (* 1.7222325338747593d0 I1)
                               (* -1.3060720016456633d0 I2)
                               (* 1.244677273903397d0 I3)
                               (* -0.8920217843256817d0 I4)
                               (* 0.42029165881581876d0 I5)
                               (* -0.7071933855383663d0 I6)
                               (* 0.1725924563090752d0 I7)
                               (* 0.2716281796607567d0 I8)
                               (* -0.03838435975543186d0 I9)
                               (* -0.9374318275089645d0 I10)))
```

```
            0.08803032940484923d0)
           0.2061083002127707d0)))))

;;; value specifications

(setf (get 'MLP10-CCT-SBS 'valeurs) '(REAL
                                0.04966100316211691d0
                                0.3817045034352309d0)
    (get 'MLP10-CCT-SBS 'type) 'ORDONNE)
```

8.2.2 Nonlinear classification (MLP of Figure 33)

```
;;; Lisp code for MLP10-SECURITY

(defun MLP10-SECURITY (objet)
  (LET ((I1
        (+ (* 0.006016299369492383d0 (PU OBJET))
           -6.036782344305822d0))
       (I2
        (+ (* 0.0020852237395437285d0 (QU OBJET))
           -0.3143199907124104d0)))
    (LET ((I1
          (TANH (+ -0.7200878338195826d0 (* 1.8190459531144412d0 I1)
                (* -0.7651537194385958d0 I2))))
         (I2
          (TANH (+ 3.543954299195746d0 (* 0.8269819057655328d0 I1)
                (* 0.7975512202163859d0 I2))))
         (I3
          (TANH (+ -1.9847264842376473d0 (* 0.5043015904752978d0 I1)
                (* -0.15990958684410542d0 I2))))
         (I4
          (TANH (+ 3.608415340512682d0 (* -2.8179427957549885d0 I1)
                (* 1.3014296372121061d0 I2))))
         (I5
          (TANH (+ 1.129618129119666d0 (* 0.10406107510523722d0 I1)
                (* -0.21223032992079402d0 I2))))
         (I6
          (TANH (+ -3.517121724150941d0 (* 5.2813596023373135d0 I1)
                (* -3.2944314432399504d0 I2))))
         (I7
          (TANH (+ -1.4581577308766775d0 (* 1.2875485865854581d0 I1)
                (* -0.5944495661898753d0 I2))))
         (I8
          (TANH (+ -3.47635289612401d0 (* -0.4666199692369496d0 I1)
                (* -2.830416533641814d0 I2))))
         (I9
          (TANH (+ -5.204689741982704d0 (* -0.6137427858633842d0 I1)
                (* -4.700484361704271d0 I2))))
         (I10
          (TANH (+ 0.19639597808687317d0
                (* -0.048111359271076515d0 I1)
                (* -0.32793412065387934d0 I2)))))
      (LIST (+ (* (IDENTITE (+ 0.6431633488498253d0
                           (* 0.6949643753189355d0 I1)
```

```
                    (* 1.667244608169718d0 I2)
                    (* 1.6278343735764025d0 I3)
                    (* -0.8915261525469574d0 I4)
                    (* -0.3205421670441548d0 I5)
                    (* 0.4496034707211712d0 I6)
                    (* -1.2034472814126476d0 I7)
                    (* 1.0873295113607544d0 I8)
                    (* -0.7706073427691704d0 I9)
                    (* 0.12134685684972324d0 I10)))
                0.4624932431938912d0)
              0.31d0)
            (+ (* (IDENTITE (+ -0.6257948278927189d0
                    (* -0.8252517554826746d0 I1)
                    (* -1.861877959473697d0 I2)
                    (* -1.6636560556576392d0 I3)
                    (* 1.1068962397653166d0 I4)
                    (* 0.5682172717786536d0 I5)
                    (* -0.4711678098203339d0 I6)
                    (* 1.6176761632614354d0 I7)
                    (* -1.070087917855041d0 I8)
                    (* 0.7436995993959763d0 I9)
                    (* -0.15852179717406878d0 I10)))
                0.4624932431938912d0)
              0.69d0)))))

;;; value specifications

(setf (get 'MLP10-SECURITY 'valeurs) '(MEMBER SECURE INSECURE)
    (get 'MLP10-SECURITY 'type) 'ORDONNE)
```