adtree and xadtree
user guide

F. De Comité D. Devigne

October 15, 2002

1 Introduction

adtree is an implementation of Freund & Mason [FM99] Alternating Decision Tree Algorithm, extended to multilabel classification [CGT01].
xadtree is a graphic interface designed to simplify the use of the algorithm.

2 Requirements

adtree is written in basic C, and should compile and run on any platform, provided the following softwares are available :

dot : the graph display algorithm dowloadable at : `http://www.research.att.com/sw/tools/graphviz`

display : an image processing and display program provided by quite all Linux release.

gtk : `xadtree` was developped under gtk+ version 1.2.6 release 1 (libgtk+1.2-devel-1.2.10-25mdk).

3 Installation

- Programs sources and documentation are in the file `http://www.grappa.univ-lille3.fr/ftp/Softs/ADTree.tgz`.
- Uncompress the file : `tar -xvzf ADTree.tgz`
- This will create a directory (ADTree) containing two subdirectories :
  - **doc** : The documentation in Postscript and HTML.
  - **src** : programs sources.

  and a Makefile.
- Go into ADTree (cd ADTree) then compile both programs : `make`
- If compilation meets no problems, both programs are now in ADTree directory : `adtree` and `xadtree`. Put them with your other programs, for example in `~/bin` or `/usr/local/bin`.

In case of problem, send us a mail, together wih the error message : `decomite@lifl.fr`.

4 Data Format

The program needs at least two files :

- A first file describing classes and attributes, with suffix `.names`.
- Another file containing the data, with suffix `.data`.
- A test file with other examples can also be provided, its extension is `.test`. Its format is similar to the `.data` file's one.

Two example files are provided in subdirectories:

- `./shortExample` : the example described below.
- `./ReutersMini` : 1100 examples from the Reuters Corpus (Reuters 21450), formatted to be handled by `adtree`.

The file format is very close to Roos Quinlan's C4.5 format [Qui93], and in fact all the files from the UCI [MM98] can be processed by *adtree*. In addition, new datasets allowing examples to have multiple classes are recognized and run by *adtree*.
The first part of file name must be the same for all three files.
For both files, a vertical bar (`|`) stands for the beginning of a comment, continuing till the end of the line. Comments and blank lines can be inserted anywhere.
A line can end with a '.', but it is no compulsory. If the last line of the file ends with a '.', it must be followed by carriage return.

4.0.1 Format of the `.names` file

The first line lists the names of the classes, separated with a comma. A class name can use any character, excepted comma. To put a comma in a class name, one must preceed it with a \.
Each of the following lines describes an attribute, in the same order they will appear in the data files.
An attribute description begins with the name of the attribute (any sequence of characters), followed by ':' and the attribute type.
The attribute type is one of the following :

ignore : this attribute will not be used neither to construct the tree, nor to classify an example.

discrete n : this attribute takes a value among n, those values being unknown until the data file is read. An error message is sent if more than n values are found while reading the data file. Not very used, but interesting for a first approximation of tree building.

continuous : the attribute is a number, comparison tests will be possible on this type of attribute.

the enumeration of all the discrete values the attribute can take, comma separated.

text : A text (with no commas in it). Tests will consider the occurence or the absence of a word in an example.

scoredtext : A list of words, together with their number of occurences in the example. Treatment of such type of attribute is not yet implemented. We plan to introduce it in order to be able to fully compare *adtree* with *boostexter* [SS00].

```
| An example of  .names  file (this line is a comment)
velo,auto,ballon à gaz,fusée.

nombre de roues : continuous.
marque : ignore.
inconvenients : discrete 6.
destination : nulle part,ecole,la mer,ciel,lune,mars
```

4.0.2 Format of the `.data` file

An example is composed with the list of its attribute values, in the same order they were declared in the `.names` file, comma separated.
Attributes values are followed with the list of all classes an example belongs to, each class name being separated from the next one with a comma. (this is a difference beetween `adtree` and `boostexter`).
The following `.data` file corresponds to the above `.names`. Second example belongs to two classes, sixth example belongs to no class.

```
2,Eddy Merckx,crevaison,ecole,velo.
4,Peugeot,crevaison,nulle part,velo,auto
4,Opel,consomme de l'essence,la mer,auto
0,Mongolfier,atterrit n'importe où,ciel,ballon à gaz,fusée
0,Saturn 5,coûte cher,lune,fusée
8,Aldi,pas garanti,nulle part.
```

5 adtree

5.1 Basic commands

Provided `adtree` binary is in your path, the following command :

`adtree -f filename`

will build a ten nodes tree, and display it in text mode.
For the above short example, the first three rules will be :

```
Number of rules : 10
-------------------------
Rule 0
if TRUE
(Number of items verifying the precondition : 6)
 Class :                       velo    auto    ballon  fusée
        If TRUE then           -0.34   -0.34   -0.80   -0.34
Else 0
```

```
-------------------------
Rule 1
if True
(Number of items verifying the precondition : 6)
                               Class :          velo    auto    ballon  fusée
         If nombre de roues >= 1.00 then         0.00    0.00    -2.63   -2.63
                              Class :            velo    auto    ballon  fusée
         If nombre de roues < 1.00 then          -2.29   -2.29   0.00    2.29
Else 0
-------------------------
Rule 2
If nombre de roues >=  1.00000
(Number of items verifying the precondition : 4)
                                Class :          velo    auto    ballon  fusée
         If inconvenients =crevaison then        2.67    0.00    -1.38   -1.38
                                 Class :         velo    auto    ballon  fusée
         If inconvenients !=crevaison then       -2.67   0.00    -1.38   -1.38
Else 0
```

Figure 1 is the tree correponding to the exemple from section 4 (reduced to the first four rules). Option `-d` (see below) was used to tell the program to generated this graphical display.

5.1.1 Reading the results

Decision structure is a tree, each rule corresponds to a branch, or part of a branch.
Branches are made of successive *confidence nodes* and *test nodes*. The root is a confidence node.
Classifying an example means make it go down the tree, according to the tests it encounters on its way, adding accordingly the values it meets.
Rule 0 corresponds to the root of the tree : each example is credited with -0.8 for the class "ballon", and -0.34 for the other classes.
Rule 1 is directly under the root, the value of harvested coefficients depends on the attribute "nombre de roues" (number of wheels). If the number of wheels is less than one, the chance to have an example of class "velo" (bicycle) decreases, which is shown by the -2.29 coefficient value. On the other hand, this increases the chances to have a "fusée"(rocket).
Rule 2 is mode deeply situated in the tree : it is used only for examples having at least one whell.
When an example has a missing value for an attribute used in the test, it is cut into pieces, and those pieces follow all the branches under this test.

5.2 adtree command line options

The option :

`adtree -h`

gives the list of available options.

```
Options:
        usage : adtree -f file_name
        Options :
        -u : Evaluation on test set
        -b #n : number of boosting steps (default 10)
        -v : verbose
        -h : this text
        -n : non normalized weights (default : normalized)
        -d : drawing a postcript visualization of the tree
        -t : enable n_ary tests
        -B : boostexter behavior
        -r : random choice of precondition to develop
        -O : display One-Error
        -H : display Hamming loss
        -C : display confusion matrix and class by class results
        -A : display coverage and avg prec
```

Detail :

-u Evaluate errors and statistics on the provided test set (`.test` extension).

-b #n : Number of boosting steps. Default : 10.

-v : Add verbosity to output : mainly used for debugging, not very useful for normal user.

-h : a reminder of the options.

-n : Each example has weights (one for each class) : choosing this option prevents the algorithm to normalize the weights after each boosting step.

-d : display a graphical representation of the tree. The `dot` binary, part of `graphviz` package must be reachable. You can download it at :

`http://www.research.att.com/sw/tools/graphviz`,

This option create two files :

- `file.dot`, a text description of the tree, used by `dot` to build the image.
- `file.ps` : a PostScript file containing the image.

-t Enable N-ary tests. The default behaviour for discrete tests is to only ask questions like "Does this example have the value x for this attribute or not". N-ary tests add questions like "what is the value of the example for this attribute ?".

-B Forces `adtree` to behave like boostexter (`http://www.research.att.com/~schapire/BoosTexter/`), which means that all tests will be direct sons of the root node (no tree-structure). Nevertheless, it remains two main differences beetween `adtree` in boostexter mode and `boostexter` itself :

- `adtree` computes confidence values for the root node, while for `boostexter`, those values are set to zero.

- `boostexter` is able to handle `SCOREDTEXT`, which is not yet implemented in `adtree`.

-r : Instead of looking for the best attribute to use at the best place in the tree, this option chooses randomly the place where the next node will be created.

-O : Displays One-Error. An exemple is considered as misclassified if `adtree` classifies it in an class it does not belong too.

-H : Displays Hamming loss.

-C : Displays a confusion matrix at each boosting step.

-A : Displays *coverage* and *average precision*.

6 xadtree

`xadtree` is a graphical interface, calling `adtree` to create trees, classifying examples, display statistics.
`xadtree` can load files, choose `adtree`'s options, build and visualize trees, show how an example go through a tree... `xadtree` was written by Damien Devigne (`devigne@lifl.fr`), student in Maîtrise d'Informatique at the University of Sciences (Lille).
Development is still in progress, please contact us for remarks, bugs and wishes.

6.1 Using `xadtree`

The command `xadtree` will launch the program.

6.1.1 Main Window

The first window opens, the only possible action is to enter a file name, without extension in the `Data` field.
(We plan to allow users to enter different file names, but it is still not implemented : button `Use different filenames`)

Once the name entered and validated, files `.data`,`.names`, and eventually `.test`, are loaded.

One can then :

- Inspect examples values (`Data` and `Test` thumb indexes), or classes names and attributes definition (`Names` thumb index).
- Creat a tree (button `New Adtree`) : a new window opens (see below).
- Choose the options to be used while creating a tree (button `Configure`).

Many buttons are reacting when the mouse passes upon them, so they could be later used for new functionnalities (attribute caracteristics, for example ...)

6.2 Configuration window

Allows the user to choose among `adtree options` :

-u : Computing errors on test set. The test set must be present.

-n : No weights normalization.

-t : Enables N-ary tests.

-r : Random choice of precondition.

-v : Verbosity.

-B : Boostexter behavior.

-b #n : Number of boosting steps (limited to 1000).

Once the options are chosen, the user can, from left to right :

- Quit the window and create a tree according to the chosen options.
- Quit the window, save the options, without creating a tree.
- Quit the window, without saving the options.

6.3 Result window

The two ways to reach this screen are from button `New Adtree` in main window, or button `Save and Create new tree` of configuration window.
It consists of two or three parts, whether a test set is available or not :

- The tree.
- A `Data` window listing the examples numbers of teaching set elements.
- A `test` window listing the examples numbers of test set elements (if any).

This page contains also four buttons :

Test item : To see the path followed by an example through the tree.

Clear path : to clear path information and come back to the original tree.

Create PostScript : display, and save a PostScript representation of the tree.

Toggle Tree Display/ Stats Display : to toggle between the tree window and the statistics window.

All those buttons and window are described hereunder.

6.3.1 The tree window

Blue lines are confidence nodes, green lines are test nodes.
A confidence line is splitted into three zones. From left to right :

- Rule number (the order in which they were created).
- The attribute used in this rule.
- The number of examples reaching this node.

A test line contains :

- A row describing the condition.
- For each class, the coefficient to apply at this point.

6.3.2 `Data` and `Test` Windows

Both windows are only lists of increasing numbers. They are the rank of the examples in the data files.
Selecting one example makes button `Test item` active.

6.3.3 `Test Item`

When an example is selected in either `Data` or `Test` window, one can visualize the path followed by this example through the tree.
Green branches are those followed by the example. Red branches are not taken by it.
If an example has a missing value for a test in the tree, all the branches are green : the example follows all of them.
The value inside the brackets (at the top of the tree) is the maximum value of the coefficients vector. If the classification is said to give only one classification per example, this corresponds to the class found by `adtree`.

6.3.4 `Clear path`

Clear all red branches, and return to the original tree.

6.3.5 `Create PostScript`

Call a function which :

- Create a text description of the tree, useable by `dot` (in file *filename.dot*).
- Display a graphical view of the tree, with the same informations as in the `tree` window.
- Save the image in a PostScript file (`file.ps`)

6.3.6 `Toggle Tree Display / Stats Display`

Toggle between the tree window and the statistics window.
The `statistics` window displays One-Error, Hamming loss, coverage and average precision.
Confusion matrices give also class by class informations.

References

[CGT01] F. De Comité, R. Gilleron, and M. Tommasi. Learning multi-label alternating decision trees and applications. In Gilles Bisson, editor, *Proceedings of CAP'01 : Conférence en Apprentissage Automatique*, pages 195–210, 2001.

[FM99] Yoav Freund and Llew Mason. The alternating decision tree learning algorithm. In *Proc. 16th International Conf. on Machine Learning*, pages 124–133, 1999.

[MM98] C.J. Merz and P.M. Murphy. UCI repository of machine learning databases, 1998.

[Qui93] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[SS00] Robert E. Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

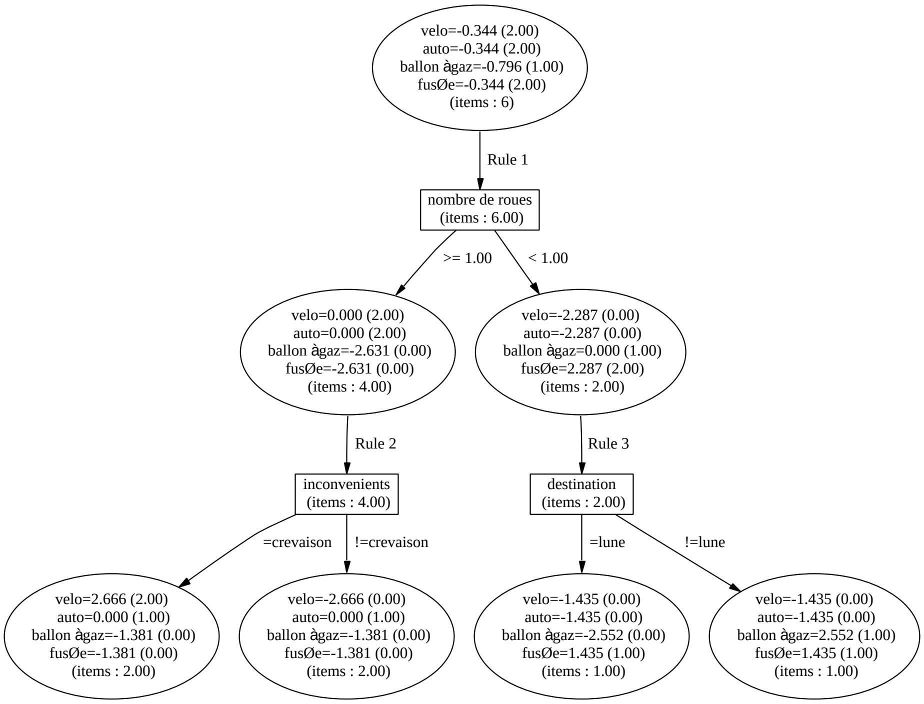

Figure 1: An alternating decision tree

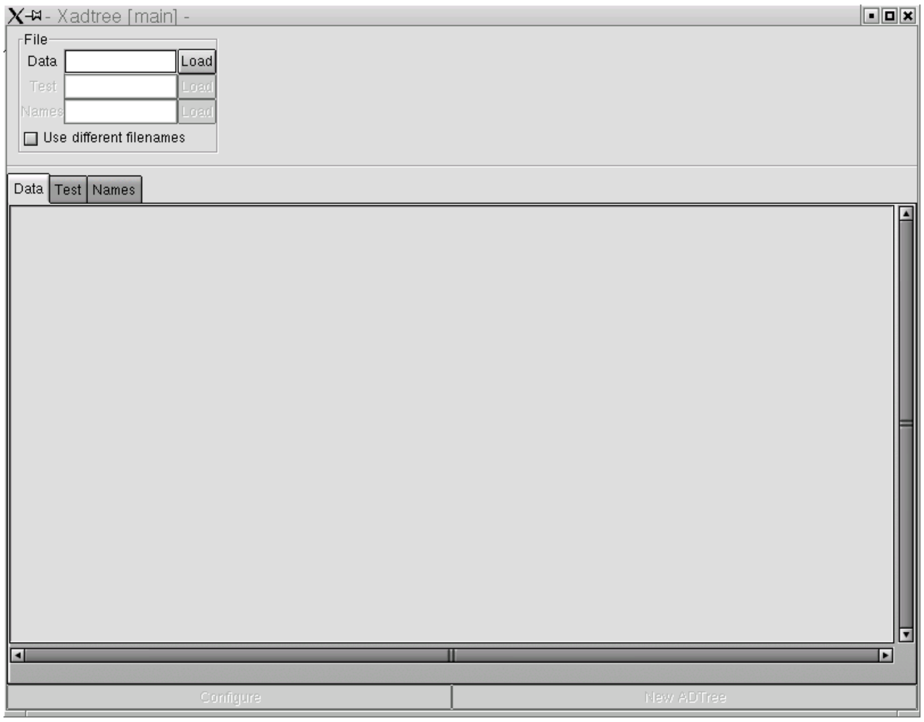

Figure 2: Initial window

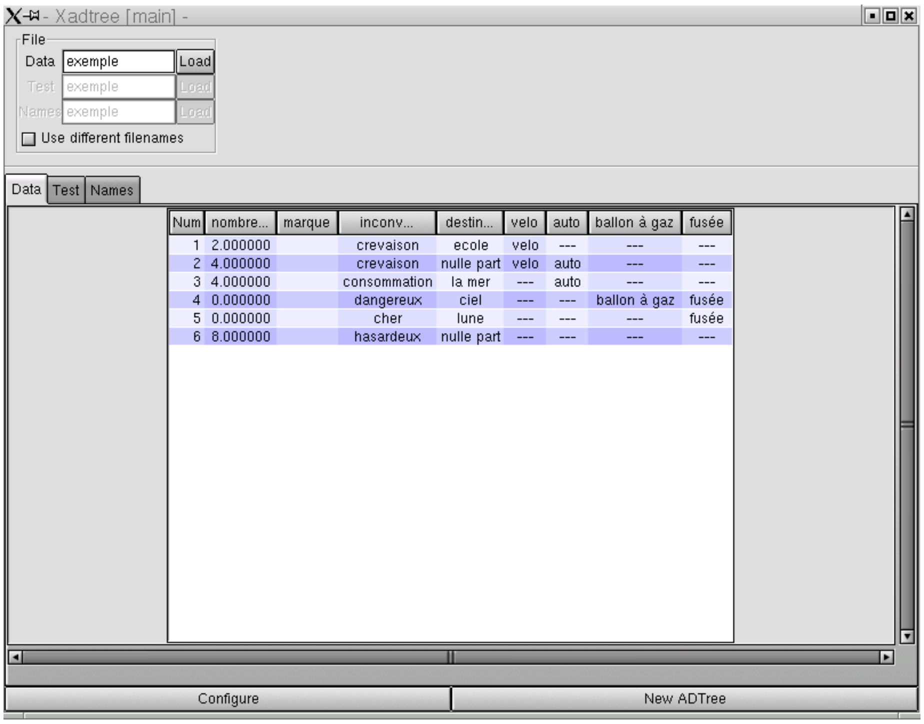

Figure 3: Initial window, files loaded

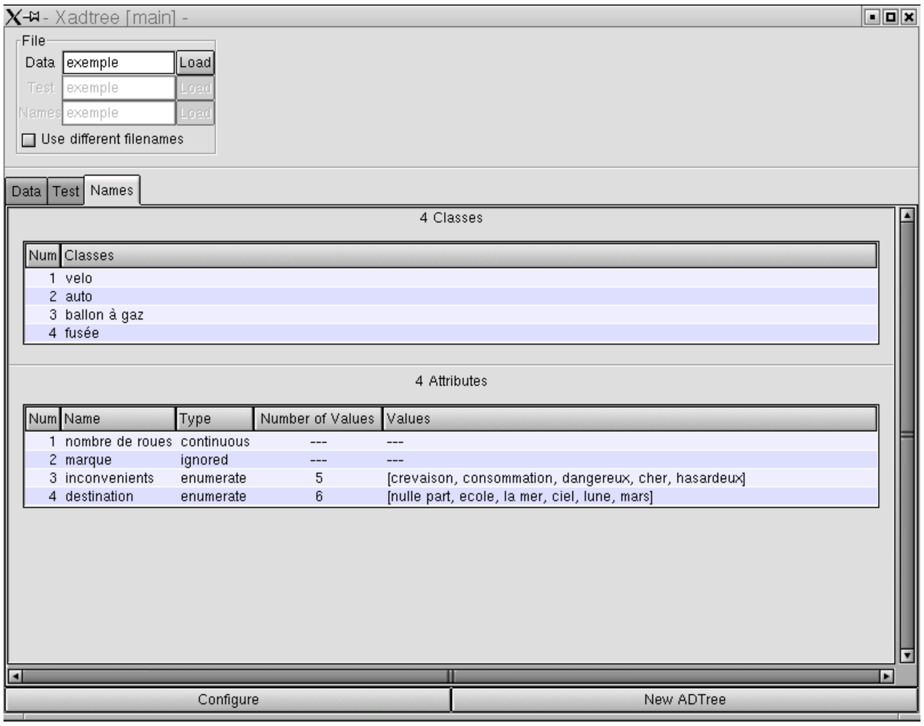

Figure 4: Initial screen, names thumb index

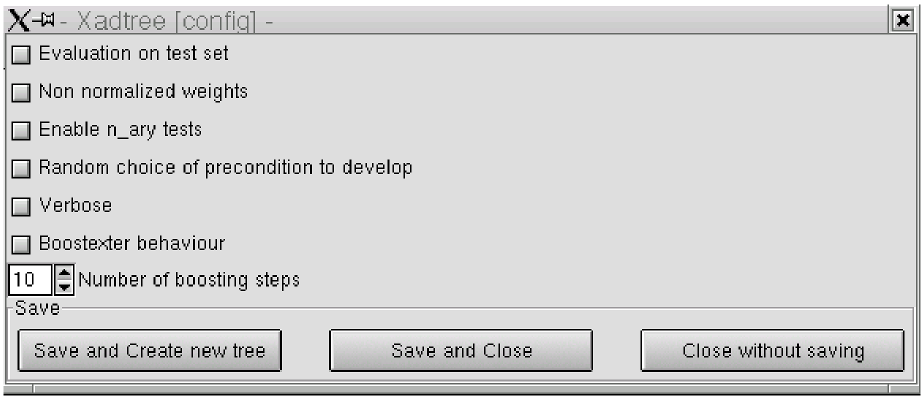

Figure 5: Configuration screen

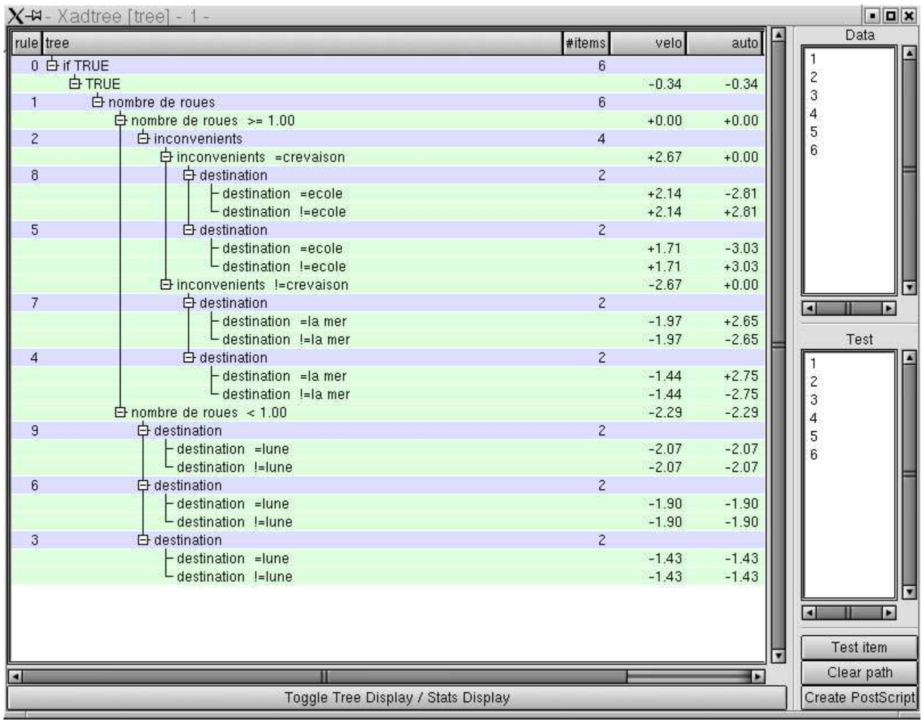

Figure 6: Result Window

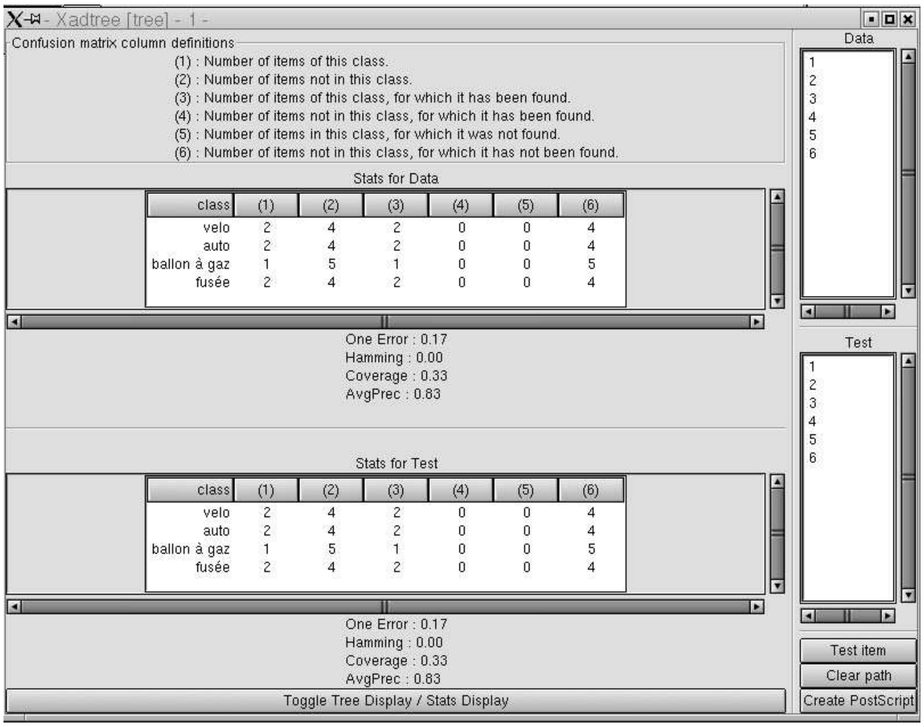

Figure 7: Statistics Window