adtree et xadtree
guide d'utilisation

F. De Comité D. Devigne

15 octobre 2002

1 Introduction

adtree implémente l'algorithme d'arbre de décision de Freund et Mason [FM99], étendu au cas multilabel [CGT01]. *xadtree* est une interface graphique plus conviviale.

2 Configuration

Le programme est écrit en C 'simple', et devrait compiler et fonctionner sur toute plateforme pourvu que soient installés :

dot : le programme de tracé de graphes disponible à : `http://www.research.att.com/sw/tools/graphviz`

display : un programme affichant des images, disponible en standard sous linux.

gtk version 1.2.6 release 1 (libgtk+1.2-devel-1.2.10-25mdk) version pour l'interface graphique.

3 Installation

- Les sources des deux programmes et la doc sont dans le fichier `http://www.grappa.univ-lille3.fr/ftp/Softs/ADTree.tgz`.
- Décompresser le fichier : `tar -xvzf ADTree.tgz`
- Cela créera un répertoire (ADTree) contenant deux répertoires :
  - doc : contenant la doc en version Postscript et en HTML.
  - src : contenant les sources des programmes.

  et un fichier Makefile.
- Se placer dans ADTree (cd ADTree) puis compiler les programmes : make
- Si la compilation se fait sans erreur, on dispose des exécutables `adtree` et `xadtree`. On peut les ranger avec les autres exécutables, dans `~/bin` ou `/usr/local/bin` par exemple.

En cas de problème de compilation, contactez-nous en joignant le message d'erreur : `decomite@lifl.fr`.

4 Format des fichiers

Le programme a besoin de deux fichiers :

- Un fichier décrivant les classes et les attributs, ce fichier a le suffixe `.names`.
- Un fichier contenant les exemples, de suffixe `.data`
- Optionnellement, un fichier de test, contenant d'autres exemples, peut être fourni avec l'extension .test. Son format est le même que celui du fichier `.data`.

Deux séries de fichiers d'exemples se trouvent dans les sous-répertoires :

- `./shortExample`: Le petit exemple décrit ci-dessus.
- `./ReutersMini` : 1100 exemples extraits du corpus Reuters, (Reuters 21450) formattés pour `adtree`.

Le format des fichiers est très proche de celui défini par Ross Quinlan pour C4.5 [Qui93], et tous les fichiers de l'UCI [MM98] peuvent être utilisés par *adtree*. *adtree* est de plus capable de traiter des données où les exemples peuvent avoir plusieurs classifications.

Le nom du fichier (devant les extensions) doit être le même pour les trois fichiers.

Pour les deux types de fichiers, une barre verticale (|) représente le début d'un commentaire qui se continue jusqu'à la fin de la ligne. On peut insérer n'importe ou dans le fichier des lignes de commentaires, ou des lignes vides. Une ligne peut se terminer par un point, mais ce n'est pas obligatoire. Si la dernière ligne se termine par un point, elle doit être suivie d'un passage à la ligne.

4.0.1 Format du fichier `.names`

La première ligne donne la liste des noms de classes séparés par des virgules. Un nom de classe peut comporter n'importe quel caractère, excepté la virgule. Pour mettre une virgule dans un nom de classe, il faut la faire précéder d'un \.

Les lignes suivantes décrivent chacune un attribut, dans l'ordre où ils figureront dans le fichier des exemples.

Une description commence par le nom de l'attribut (toute suite de caractères), suivie de ":", et du type de l'attribut.

Ce type doit être l'un des suivants :

ignore : cet attribut n'est pas utilisé pour la construction de l'arbre.

discrete n : l'attribut peut prendre une valeur parmi n, ces valeurs ne sont pas connues lors de la lecture du fichier names. Le programme s'arrête sur un messsage d'erreur si, lors de la lecture des exemples, on rencontre plus de n valeurs pour cet attribut. Peu utilisé en pratique, mais utile lorsqu'on veut faire un premier arbre rapidement (sans faire de statistiques ou de recodage sur le fichier d'exemples).

continuous : l'attribut est un nombre, les tests "attribut supérieur à un seuil" peuvent s'appliquer.

la liste des valeurs discrètes que peut prendre un attribut, séparés par des virgules.

text : un texte quelconque ne comportant pas de virgules. Les tests porteront sur la présence ou l'absence d'un mot dans le texte.

scoredtext : une liste de mots et de leur nombre d'occurences. Le traitement par adtree de ce type d'attributs n'est pas encore implémenté. Il est introduit pour obtenir un fonctionnement analogue à celui de `boostexter` [SS00].

```
| Un exemple de fichier .names (cette ligne est un commentaire)
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 du fichier `.data`

Un exemple est représenté sur une ligne, par la liste des valeurs d'attributs, dans l'ordre où ils ont été déclarés dans le fichier `.names`, séparées par des virgules.

Les valeurs d'attributs sont suivies des classes auxquelles appartient l'exemple, chaque classe étant séparée de la précédente par une virgule.

L'exemple ci-dessous correspond au fichier `.names` décrit ci-dessus. Le deuxième exemple a deux classes. Le sixième exemple n'en a pas.

```
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 Fonctionnement d'adtree

5.1 Commandes de base

Pourvu qu'`adtree` soit dans les chemins , la commande :

`adtree -f nom_de_fichier`

construit un arbre à 10 nœuds, et affiche cet arbre en mode texte.

Pour le fichier donné en exemple plus haut, les trois premières règles produites sont listées ci-dessous :

```
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
```

La figure 1 est l'arbre correspondant à l'exemple de la section 4 (les quatre premières règles seulement). Il a été obtenu par l'option `-d` (voir ci-dessous).

5.1.1 Lecture du résultat

La structure de décision produite est un arbre, chaque règle en représente une branche.

Les branches de l'arbre sont une succession de *nœuds confiance* et de *nœuds test*, la racine étant un nœud confiance.

Classer un exemple consiste à lui faire parcourir l'arbre, en fonction des valeurs de ses attributs et des tests rencontrés, en tenant à jour un coefficient pour chaque classe.

La règle 0 correspond au nœud confiance en racine : tout exemple que l'on veut classer est crédité de -0.8 comme valeur pour "ballon", et de -0.34 pour les autres classes possibles.

La règle 1 est directement sous la racine, les coefficients glanés dépendent du nombre de roues de l'exemple. Si le nombre de roues est plus petit que un, la quantité associée à la classe vélo décroit de 2.29, tandis qu'on augmente les chances d'être en présence d'une fusée, le coefficient asocié est crédité de 2.29.

La règle 2 est placée plus profondément dans l'arbre : elle se branche sous le nœud confiance correspondant à "nombre de roues au moins égal à 1".

Quand un exemple n'a pas de valeur fixée pour un attribut, et qu'un test concerne cet attribut, il "descend" dans toutes les branches, ne récupérant dans ces branches qu'une portion des coefficients, portion proportionnelle au nombre d'exemples ayant emprunté chaque branche lors de la construction de l'arbre. En conséquence, il devient dès lors difficile de calculer à la main les coefficients correspondants.

L'interêt principal de cette représentation de l'arbre est qu'elle permet de voir quels sont les attributs qui jouent un rôle important dans la décision : ce sont les premiers choisis.

On peut alors, dans un second temps, consulter un expert du domaine, vérifier que ces attributs sont importants ou, s'ils sont trop corrélés avec les classes à découvrir, les ignorer (`ignore` dans le fichier `.names`), et relancer `adtree`.

5.2 Options d'adtree

La liste des options peut être consultée dans adtree par :

`adtree -h`

qui répond :

```
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
```

Détail des options :

-u Indique la présence d'un ensemble test : les différents calculs d'erreurs possibles se feront sur le fichier `.data` et sur le fichier `.test`

-b #n : fixe le nombre de pas de boosting, i.e. le nombre de nœuds de l'arbre (10 par défaut).

-v : Demande à l'algorithme de détailler ses calculs : utilisé principalement dans un but de débuggage.

-h : le résumé des options.

-n : chaque exemple est muni d'un poids : en choisissant cette option, on ne normalise pas les poids pour que leur somme soit toujours égale à un.

-d : affiche une représentation graphique de l'arbre. Le programme `dot`, disponible dans le paquetage `graphviz` sur le site :
`http://www.research.att.com/sw/tools/graphviz`,
doit être installé sur la machine. Cette option produit deux fichiers :
- un fichier `nom_du_fichier.dot` qui est une description de l'arbre en texte, utilisée par `dot` pour générer le dessin.
- un fichier `nom_du_fichier.ps` qui contient le dessin de l'arbre.

Les lettres accentuées ne passent pas très bien dans le dessin : il vaut mieux éviter de les employer dans les noms de classes, d'attributs, ou de valeurs d'attributs.

-t Autorise les tests n-aires. Pour les tests discrets, le test par défaut d'`adtree` est de vérifier qu'un exemple a une valeur donnée pour un attribut ou non.

En autorisant les tests n-aires, on peut lister toutes les valeurs possibles de l'attributs. Les tests binaires existent toujours dans ce cas, et `adtree` peut parfois les préférer.

-B Force `adtree` à se comporter comme boostexter (`http://www.research.att.com/~schapire/BoosTexter/`), ce qui signifie que tous les nœuds tests sont immédiatement sous la racine. Il reste cependant deux différences entre `adtree` en mode boostexter et `boostexter` lui-même :

- `adtree` calcule des valeurs de confiance pour le nœud confiance racine, alors que `boostexter` fixe ces valeurs à 0.
- `boostexter` peut traiter des attributs de type `SCOREDTEXT`, et donc peut travailler sur le nombre d'occurences d'un mot dans un exemple. Actuellement, `adtree` ne traite pas ce cas.

-r : Au lieu de chercher le meilleur endroit et le meilleur test, on fixe aléatoirement l'endroit où placer le prochain test, le programme recherche alors le meilleur test pour cet endroit.

-O : affiche l'erreur 'standard' : rapport du nombre d'exemples mal-classés sur le nombre total d'exemples. Un exemple est mal-classé si on lui attribue une classe qu'il n'a pas.

-H : affiche l'erreur de Hamming. Chaque fois qu'un exemple possédant une certaine classe n'st pas signalé par l'algorithme comme étant de cette classe, ou bien chaque fois qu'un exemple est signalé d'une classe alors qu'il ne l'est pas, une erreur est comptée. On affiche le rapport entre le nombre d'erreurs comptées et le produit du nombre d'exemples par le nombre de classes. Cet indicateur a plus de sens que l'erreur standard dans le cas multilabel.

-C : Détaille les résultats de classification classe par classe, à chaque étape de boosting. La dernière colonne donne l'erreur 'standard' pour chaque classe, même dans le cas multilabel.

-A : Affiche les valeurs de *coverage* et *average precision*, utilisés principalement lorsqu'on travaille sur la classification de textes.

6 xadtree

`xadtree` est une interface graphique qui appelle `adtree` pour la création des arbres, la classification de nouveaux exemples, les calculs d'erreurs et de statistiques.

`xadtree` permet de charger des fichiers, fixer les options d'adtree, construire et visualiser des arbres, classer un exemple, et montrer comment il parcourt l'arbre.

`xadtree` a été réalisée par Damien Devigne (`devigne@lifl.fr`), étudiant en maîtrise d'informatique.

Le développement n'est pas terminé, d'autres fonctionnalités pourraient s'ajouter, vous pouvez nous faire part de vos suggestions.

6.1 Utilisation de `xadtree`

Le programme se lance par la commande `xadtree`.

6.1.1 Fenêtre principale

Une fenêtre s'ouvre, où la seule action possible est d'entrer un nom de fichier, sans extension, dans le champ `Data`. (On a prévu la possibilité d'entrer des noms de fichiers différents, mais ceci n'est pas encore implémenté : bouton `Use different filenames`)

Une fois ce nom entré et validé, les fichiers `.data`,`.names` et éventuellement `.test` sont chargés.

On peut alors :

- Consulter les valeurs des exemples (onglets `Data` et `Test`), inspecter la liste des classes et la définition des attributs (onglet `Names`).
- Lancer la création d'un arbre (bouton `New Adtree`) : une nouvelle fenêtre s'ouvre (voir plus loin).
- Fixer les options à utiliser pour créer un arbre (bouton `Configure`).

De nombreux boutons 'réagissent' aux clics ou au pasage de la souris, on se reserve ainsi la possibilité d'ajouter facilement des fonctionnalités au programme (statistiques sur les valeurs d'attributs, par exemple).

6.2 Fenêtre de configuration

Elle permet de fixer certaines options, à savoir celles qui correspondent dans l'ordre aux options d'`adtree` :

-u : calcul des erreurs sur l'ensemble test. Cette option ne fonctionne évidemment pas lorsqu'aucun fichier test n'est disponible.

-n : ne pas normaliser les poids.

-t : autoriser les tests n-aires.

-r : Choix aléatoire de la précondition.

-v : mode bavard.

-B : fonctionnement à la boostexter.

-b #n : nombre d'étapes de boosting (limité à 1000).

Une fois les options fixées, on peut, de gauche à droite :

- Quitter la fenêtre et créer un arbre à partir de ces options.
- Quitter la fenêtre et sauvegarder le choix des options.
- Quitter sans sauver.

6.3 Fenêtre résultat

On y arrive après avoir cliqué sur le bouton `New Adtree` de la fenêtre principale, ou le bouton `Save and Create new tree` de la fenêtre de configuration.

Elle comprend deux ou trois zones, selon qu'un ensemble test était ou non disponible :

- Une représentation de l'arbre.
- Une fenêtre `Data` listant les numéros des exemples de l'ensemble d'apprentissage.
- Une fenêtre `test` qui fait de même pour l'ensemble test (s'il existe).

On trouve aussi sur cette page quatre boutons :

Test item : pour voir comment un exemple parcourt l'arbre.

Clear path : pour revenir à l'arbre initial.

Create PostScript : afficher à l'écran, et sauvegarder une représentation graphique de l'arbre.

Toggle Tree Display/ Stats Display : pour passer de la représentation de l'arbre à l'affichage des calculs d'erreurs et des statistiques.

Chacun de ces boutons et fenêtre est décrit ci-dessous.

6.3.1 L'arbre

Les lignes bleues correspondent aux nœuds confiance, les lignes vertes aux nœuds test.

Une ligne confiance se divise en trois zones. De gauche à droite, ces zones sont :

- Le numéro de la règle (i.e. l'ordre dans lequel elles ont été créées).
- L'attribut utilisé lors du test.
- Le nombre d'exemples arrivant sur ce nœud.

Une ligne test se compose de :

- Une colonne décrivant la condition utilisée.
- Pour chaque classe, le score correspondant à cette condition.

6.3.2 Les fenêtres `Data` et `Test`

Ces deux fenêtres se présentent sous la forme de listes de numéros, qui sont les indices des exemples présents dans les ensembles d'apprentissage et de test.

Sélectionner un exemple rend opérationnel le bouton `Test item`

6.3.3 `Test Item`

Lorsqu'un exemple est sélectionné dans une des fenêtres `Data` ou `Test`, ce bouton permet de visualiser sur l'arbre le chemin suivi par cet exemple.

Les branches test vertes sont celles suivies par l'exemple. Les branches rouges sont celles qu'il n'emprunte pas.

Lorsque l'arbre contient un test sur un attribut pour lequel l'exemple n'a pas de valeur, toutes les branches correspondant à ce test restent vertes : l'exemple descend dans chacune des branches.

Sur la première ligne, le score de l'exemple pour chacune des classes est affiché.

La valeur entre crochets est le maximum de ces valeurs : il correspond à la classe qui sera donnée à l'exemple (si l'on se contente de désigner une classe pour chaque exemple).

6.3.4 `Clear path`

Réinitialise l'arbre lorsqu'on a auparavant regardé le parcours d'un exemple dans l'arbre.

6.3.5 `Create PostScript`

Appelle une fonction qui :

- Transcrit l'arbre en définition de graphe pour `dot`.
- Affiche une représentation graphique de l'arbre, contenant les mêmes informations que la fenêtre `arbre`.
- Sauvegarde le dessin dans le fichier `nom_du_fichier.ps`

6.3.6 `Toggle Tree Display / Stats Display`

Permet de passer de la fenêtre `arbre` à la fenêtre `statistiques`.

La fenêtre `statistique` affiche l'erreur, l'erreur de Hamming, et les valeurs de *coverage* et *average precision*.

Des matrices permettent d'observer le comportement de la classification selon la classe.

Références

[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.

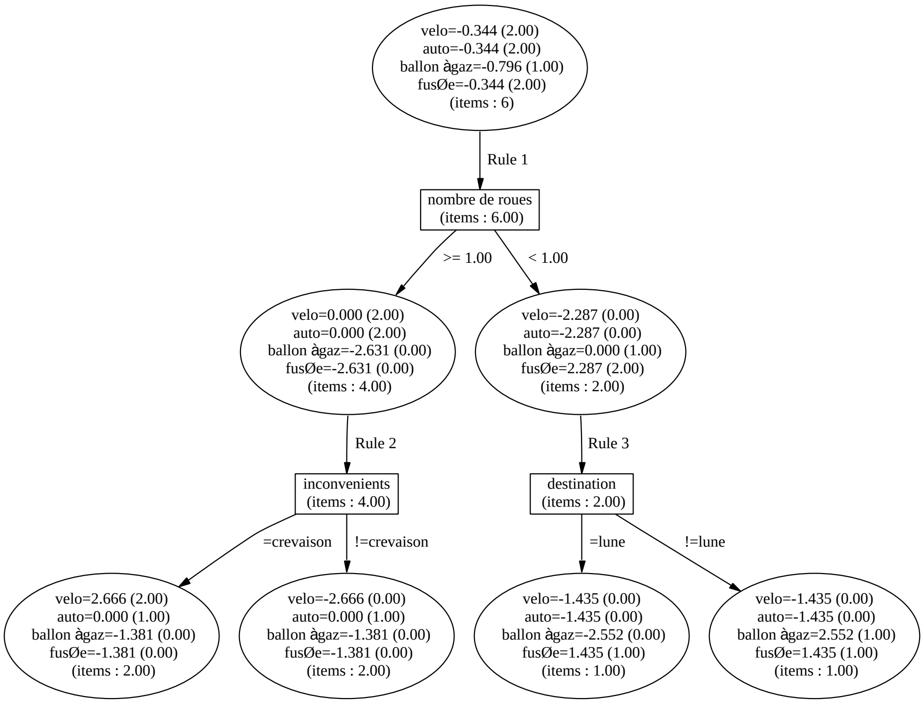

FIG. 1 – *Un arbre alternant*

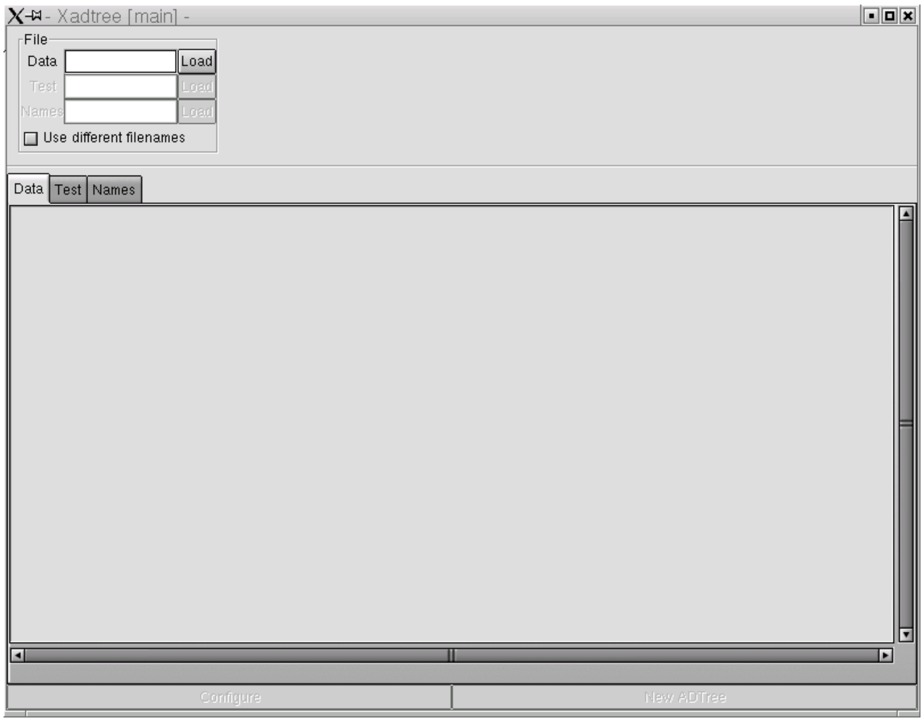

FIG. 2 – *Ecran initial*

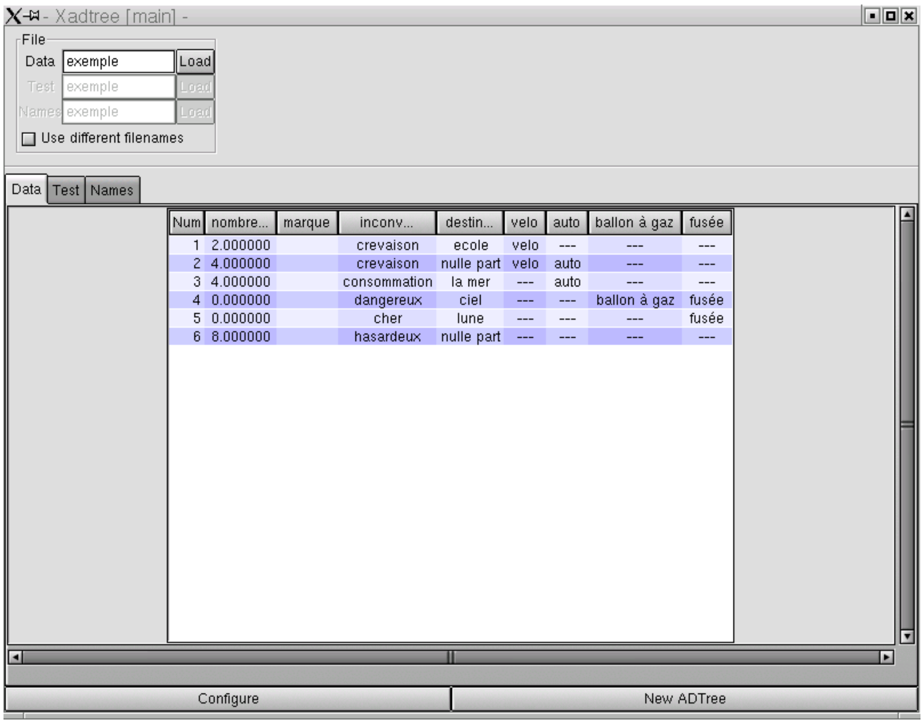

Fig. 3 – *Ecran initial, fichiers chargés*

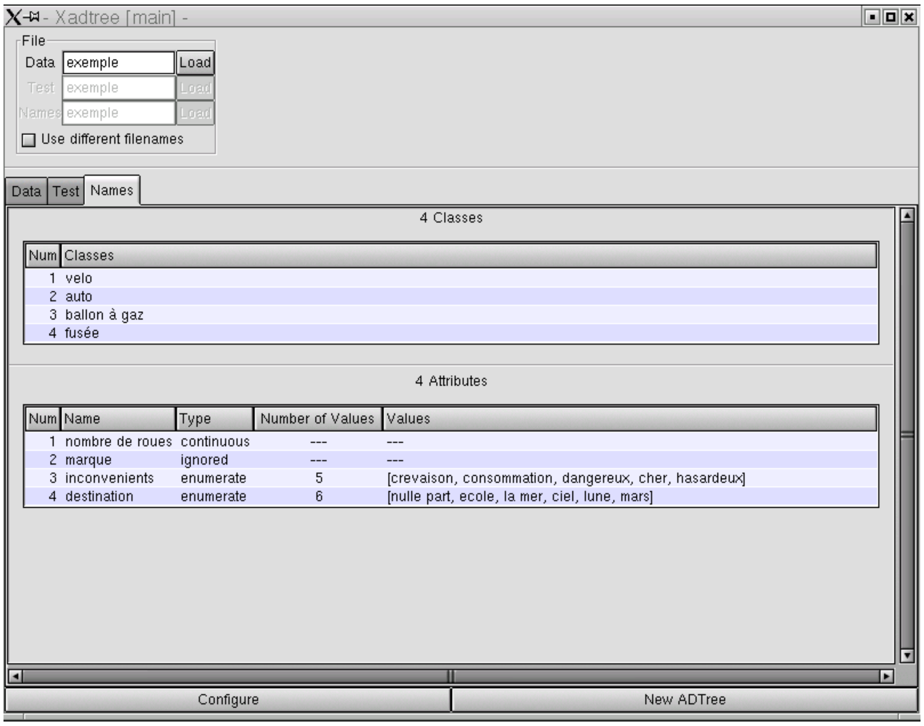

Fig. 4 – *Ecran initial, onglet names*

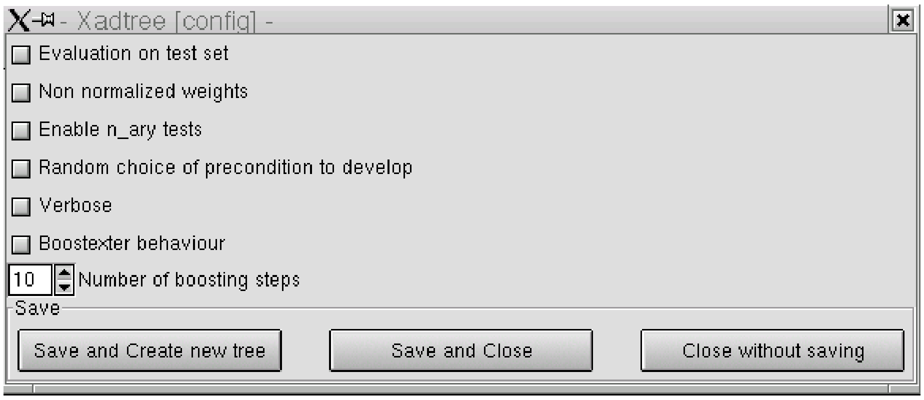

FIG. 5 – *Ecran de configuration*

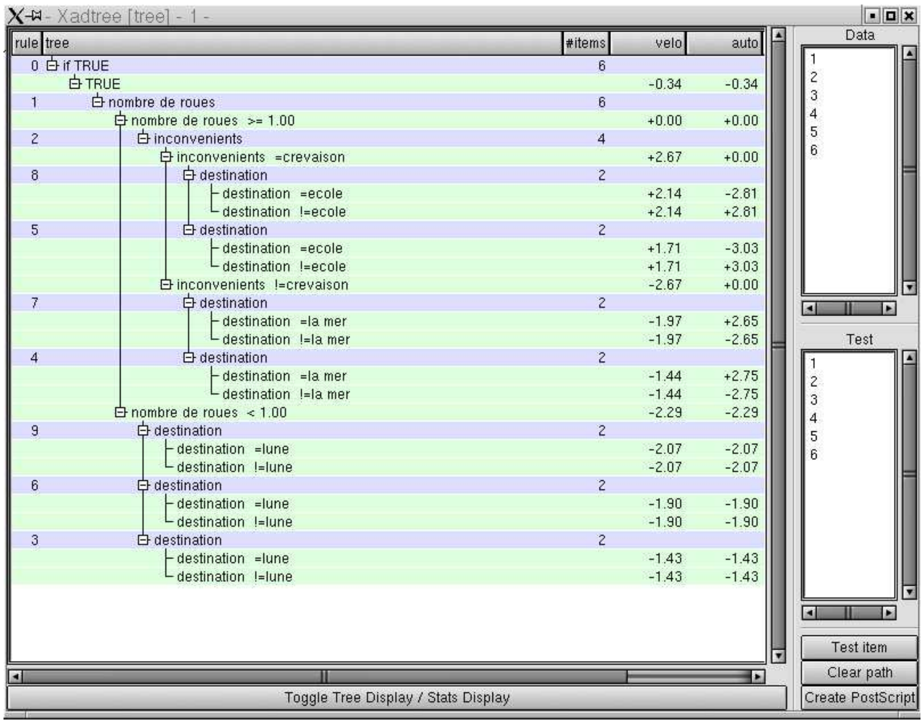

FIG. 6 – *Fenêtre résultat*

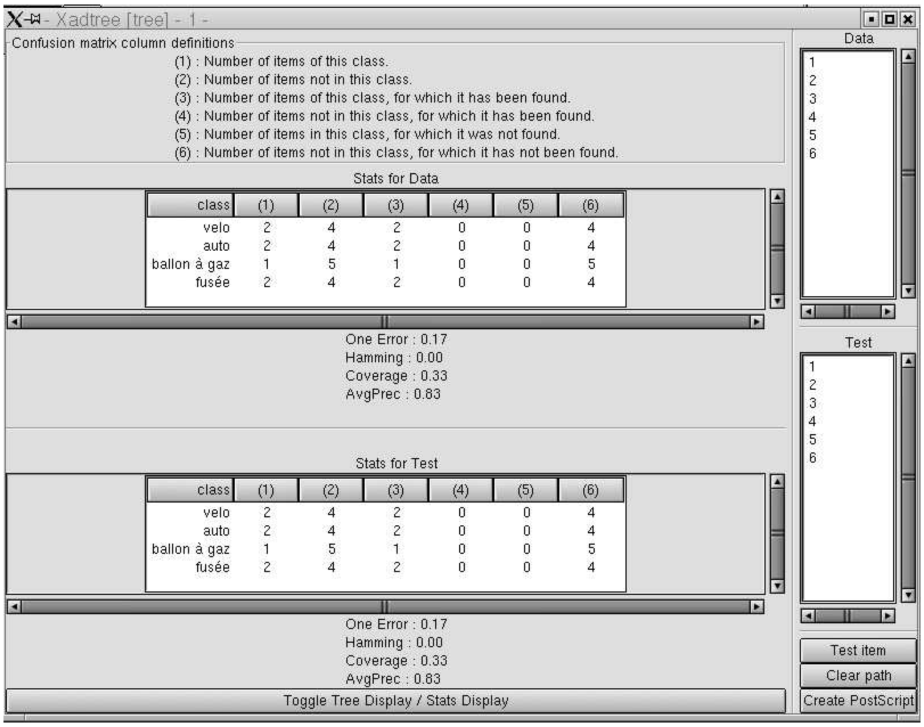

FIG. 7 – *Fenêtre des statistiques*