1 Objectif

Comparer les performances des différentes implémentations libres (gratuites) de l'algorithme C4.5.

La gestion de la volumétrie est une des pierres angulaires du Data Mining. Toute présentation du domaine passe par le sempiternel « depuis quelques années, les entreprises amassent une quantité considérable de données, l'enjeu n'est plus comment les stocker mais plutôt comment les exploiter pour en tirer de l'information », etc., etc. Ok, ok, n'en jetez plus, on est d'accord.

Si le traitement des grandes bases est un enjeu important, on est curieux de savoir comment se comportent les logiciels libres (gratuits) dans ce contexte. Ils sont nombreux dans le Data Mining. J'essaie de suivre un peu leur évolution. La capacité à analyser des grands fichiers est un critère que je regarde souvent pour situer mes propres implémentations. La plupart chargent l'ensemble de données en mémoire centrale. De fait, la différenciation en termes de performances repose essentiellement sur la technologie utilisée (compilé ou pseudo-compilé) et la programmation. Le goulot d'étranglement est la mémoire disponible.

Dans ce didacticiel, nous comparons les performances de plusieurs implémentations de l'algorithme C4.5 (Quinlan, 1993) lors du traitement d'un fichier comportant 500.000 observations et 22 variables. Pour rappel, C4.5 est adepte de la construction « hurdling » c.-à-d. il crée un arbre maximal en acceptant les segmentations avec un gain nul. Dans une seconde partie, il effectue un post élagage basé sur l'erreur pessimiste, sans intervention d'un second échantillon donc, ni de validation croisée.

Les logiciels que nous avons mis en compétition sont les suivantes[1]

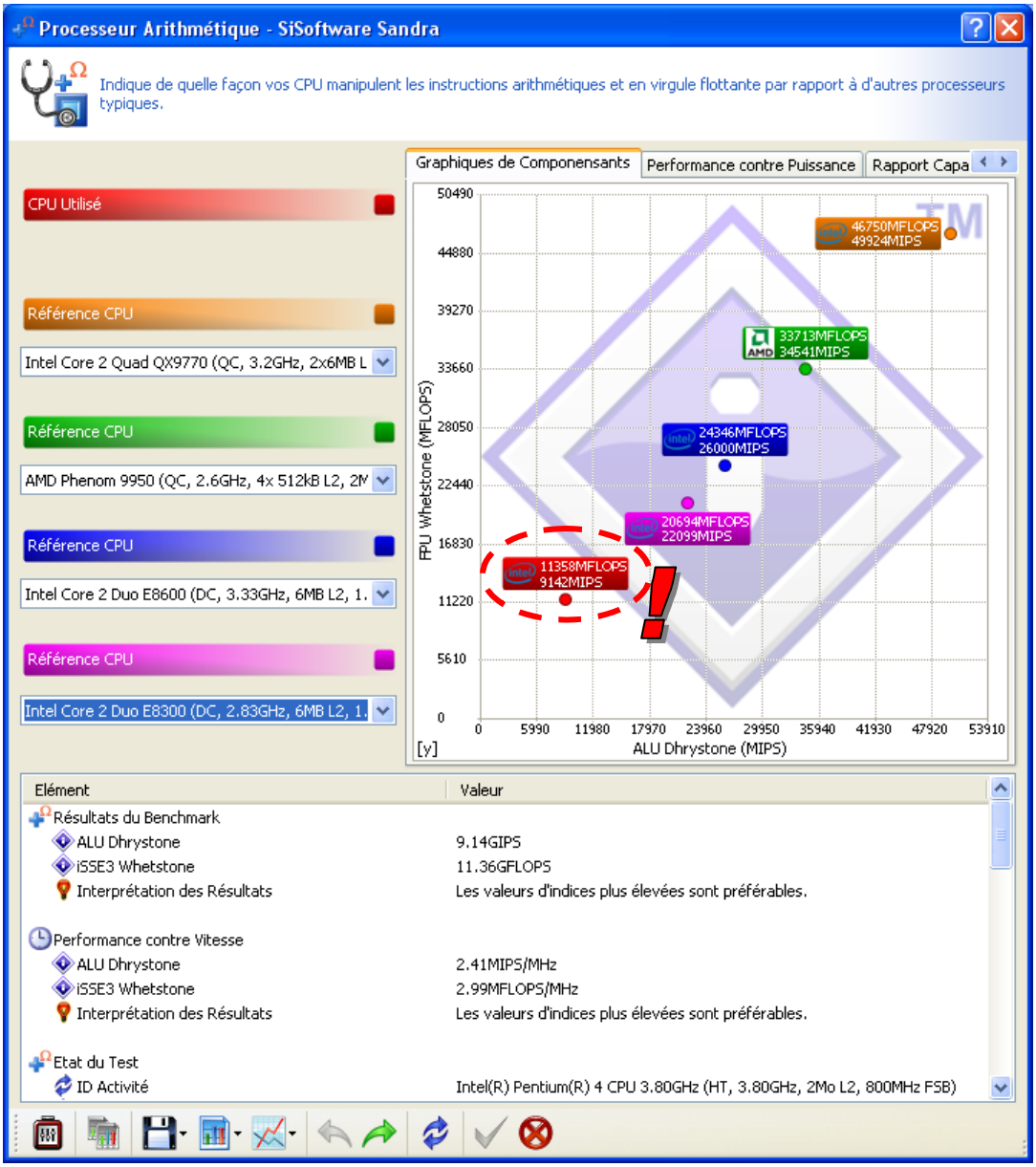
Logiciel	**Version**	**URL**
KNIME	1.3.5	http://www.knime.org/index.html
ORANGE	1.0b2	http://www.ailab.si/orange/
R (package rpart)	2.6.0	http://www.r-project.org/
RAPIDMINER (YALE)	Community Edition	http://rapid-i.com/
SIPINA	Recherche	http://eric.univ-lyon2.fr/~ricco/sipina.html
TANAGRA	1.4.27	http://eric.univ-lyon2.fr/~ricco/tanagra/
WEKA	3.5.6	http://www.cs.waikato.ac.nz/ml/weka/

Ce document vient un complément d'un ancien didacticiel où nous montrions les performances de ID3 de Tanagra sur un fichier encore plus volumineux[2]. Nous retiendrons 2 critères pour comparer les logiciels : l'occupation mémoire et le temps de traitement.

[1] Accessoirement, nous profitons de ce comparatif pour situer un peu les versions actuelles de ces logiciels. Les évolutions sont rapides. La dernière fois que j'ai réalisé une revue de détail, c'était en décembre 2005 (http://tutoriels-data-mining.blogspot.com/2008/04/les-logiciels-gratuits-pour.html).

[2] http://tutoriels-data-mining.blogspot.com/2008/03/traitement-des-grands-fichiers.html ; 581102 observations et 55 variables.

Afin que tout un chacun puisse reproduire l'expérience et comparer les résultats, voici les caractéristiques de notre machine : il s'agit d'un Pentium 4 à 3.8 Ghz avec 2 GB de mémoire vive. Une bête de course il n'y a pas si longtemps, un vieux tromblon aujourd'hui. Il tourne sous Windows XP SP3. Ces chiffres sont à la fois parlants et très imprécis. Pour avoir un élément de comparaison indiscutable, nous avons mesuré les performances à l'aide de la version Lite du logiciel SISOFTWARE SANDRA (http://www.sisoftware.net/). Nous obtenons :



Quand je parlais de vieux tromblon. C'est assez déprimant en fait, on ne devrait jamais acheter un ordinateur pour sa puissance...

2 Les données

Les données sont regroupées dans l'archive WAVE500K .ZIP[3]. Il s'agit des « ondes » de Breiman, très connu au sein de notre communauté (Breiman et al., 1984). A l'aide du générateur, nous avons produit un fichier de 500.000 observations. La variable à prédire comporte 3 classes, les 21

[3] http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/wave500k.zip

descripteurs sont tous continus, la discrétisation jouera un rôle considérable lors de la construction de l'arbre.

Nous traiterons en priorité le format ARFF de WEKA. La grande majorité des logiciels de data mining savent le manipuler. A défaut, le format texte avec séparateur tabulation sera utilisé.

3 Comparaison des performances

Certains logiciels donnent directement une indication sur le temps de traitements, nous la privilégierons. Dans le cas contraire, nous utilisons simplement un chronomètre. Ce n'est pas très précis. Mais nous ne sommes pas non plus en train de juger le 100 mètres de la finale olympique. Le plus important est d'avoir un ordre d'idées pour positionner les logiciels. Nous mesurons principalement le temps dévolu à l'importation des données et à la création de l'arbre.

Notons une caractéristique intéressante de notre machine dans un contexte de comparaison, le processeur étant double cœur et les logiciels étant mono-thread, ces derniers peuvent solliciter au maximum un des cœurs sans être pollués par d'autres applications, notamment les éventuelles opérations systèmes. Le temps obtenu est relativement fiable et représentatif.

Concernant l'occupation mémoire, pour obtenir une mesure globale, indiquant à la fois les allocations interne et l'espace nécessaire à l'environnement global, nous privilégierons les indications du gestionnaire de tâches de WINDOWS (Figure 1). Quatre mesures de l'espace occupé seront effectuées : au lancement du logiciel, après l'importation des données, après la création de l'arbre, et le pic lors des calculs. Cette dernière mesure est importante. En effet il est très possible que des structures intermédiaires soient utilisées pendant la construction de l'arbre. Elles sont détruites à la fin des calculs, pourtant nous devons en tenir compte. Si l'espace alloué dépasse les capacités de la machine, la construction de l'arbre ne pourra pas être menée à son terme.

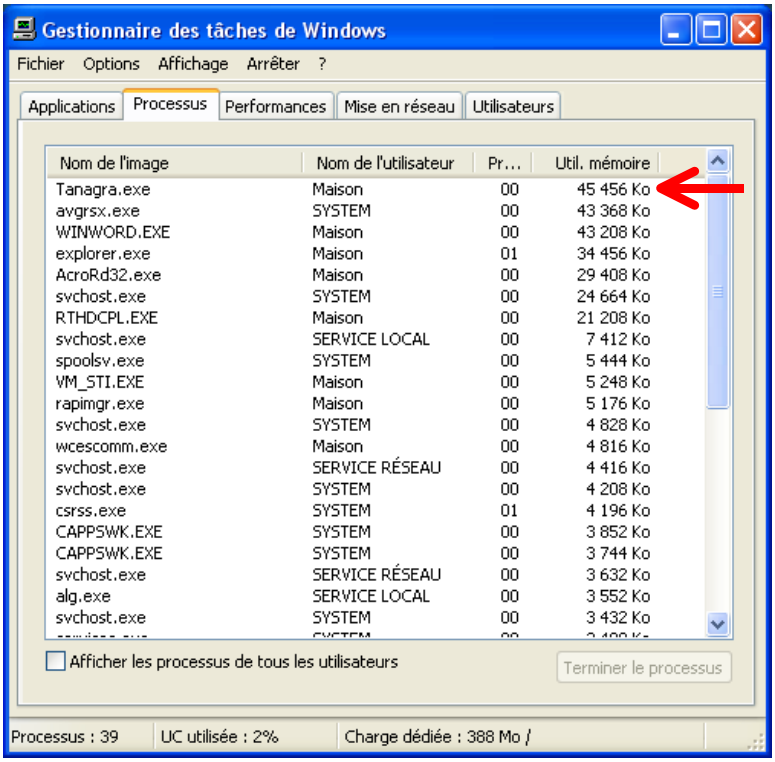


Figure 1 - Mesure de l'occupation mémoire d'un logiciel via le gestionnaire de tâche

3.1 KNIME

KNIME (Konstanz Information Miner -- http://www.knime.org/) est développé par la Chaire de Bioinformatique et Information Mining de l'Université de Constance, en Allemagne. L'interface est très classique s'agissant d'un logiciel de Data Mining, avec un fonctionnement par filières. Les composants sont des opérateurs de traitement de données, les liens entre eux représentent les flux d'informations, essentiellement le flux de données.

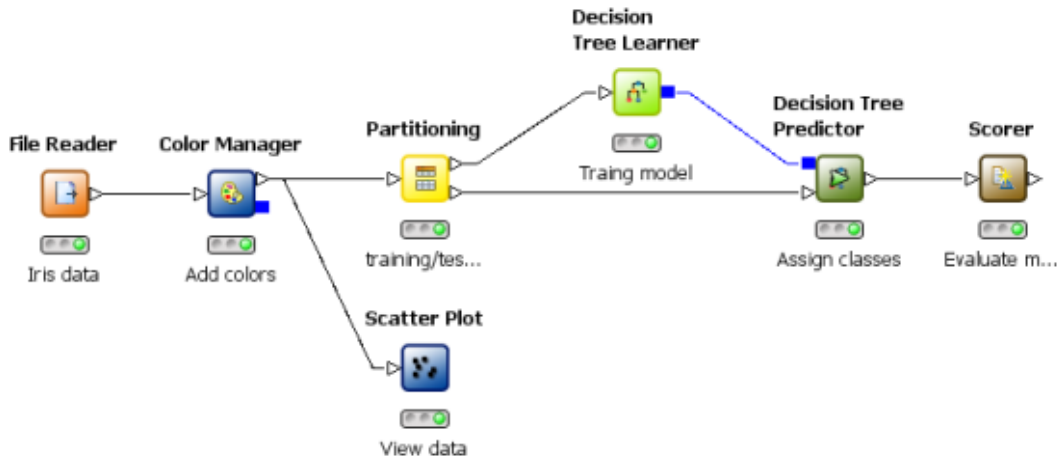
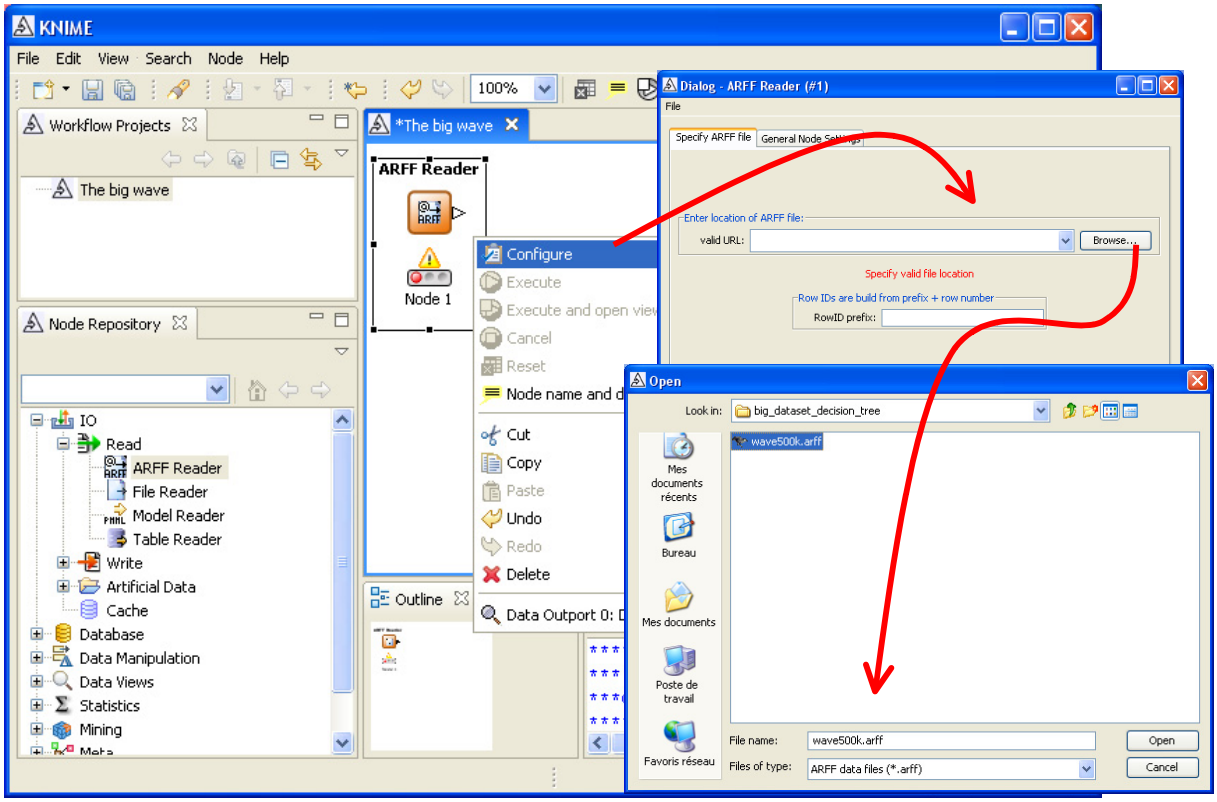


Figure 2 - Exemple de filière KNIME – Apprentissage et évaluation d'un arbre de décision

KNIME semble savoir swaper les fichiers sur disque en certaines circonstances, qui restent à préciser. C'est une manière de lever la contrainte mémoire vive. Il intègre les techniques en provenance de WEKA. Il semble aussi savoir, via des « plugins », exécuter des scripts R. Sa bibliothèque de méthodes est donc potentiellement très large.

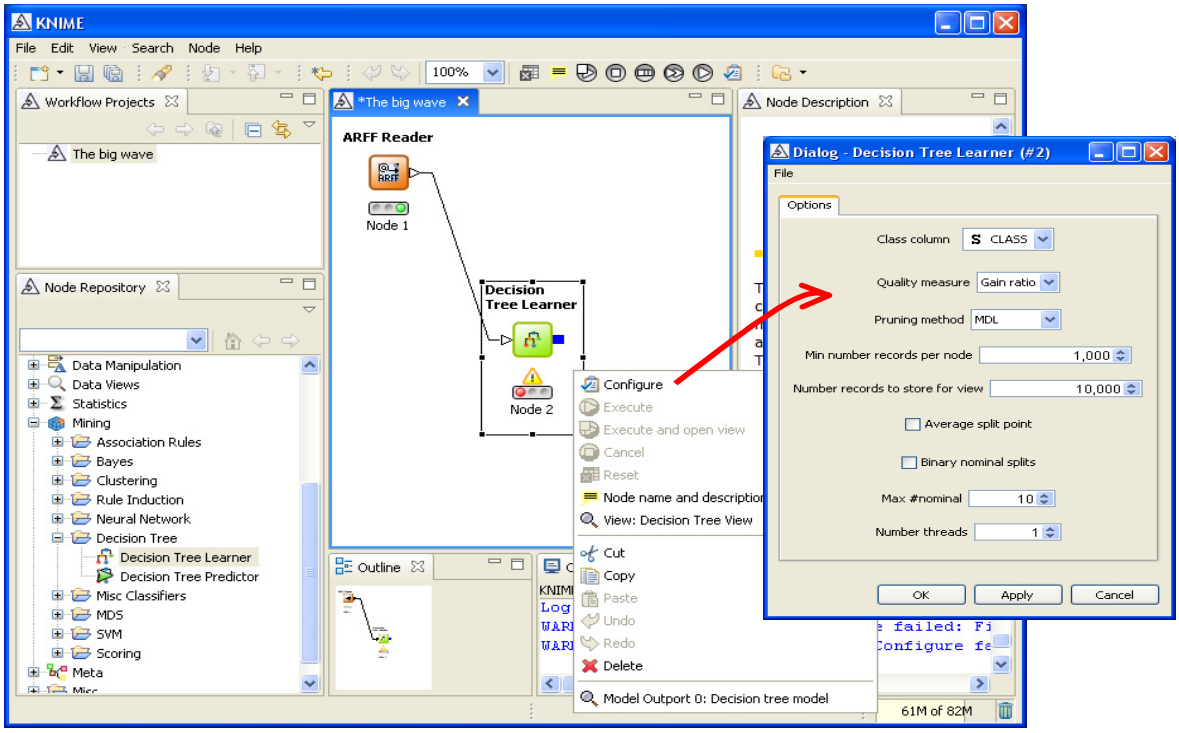
Du point de vue fonctionnel, la présence d'une fenêtre de documentation des méthodes, disponible à tous moments, est un plus appréciable. Nous avons accès directement à la description des paramètres des composants. Cela évite d'avoir à se perdre dans les méandres du fichier d'aide ou dans les « white paper »... quand ils sont disponibles.

KNIME fonctionne avec la machine virtuelle JAVA. Au lancement du logiciel, l'occupation mémoire est de 92.6 Mo. Nous devons créer un nouveau projet. Un wizard nous permet de le faire simplement. Nous plaçons ensuite le composant ARFF READER dans l'espace de travail (le workflow). Nous actionnons le menu CONFIGURE et nous sélectionnons notre fichier de données au format WEKA.



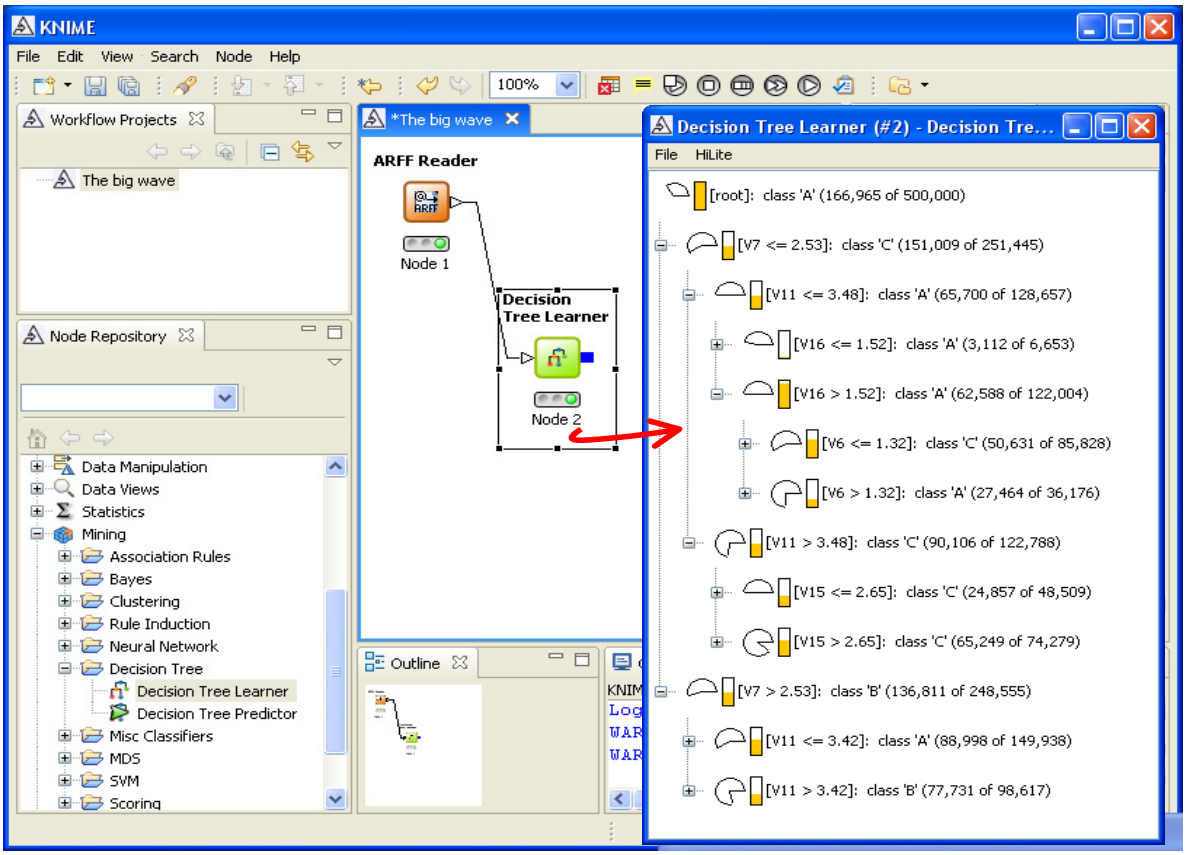
Pour lancer le chargement des données, nous actionnons le menu contextuel EXECUTE. L'opération prend 47 secondes, l'occupation mémoire maintenant est de 160.4 Mo.

Nous insérons le composant DECISION TREE LEARNER dans le workflow, nous remarquons sur la droite une description assez fournie de la méthode et de ses paramètres. Nous lui relions le composant ARFF READER, puis nous actionnons le menu contextuel CONFIGURE pour définir les paramètres.



La méthode n'est pas exactement C4.5. Mais elle y est apparentée. Nous retrouvons à peu près les paramètres adéquats. Nous souhaitons surtout qu'il y ait au moins 1000 individus sur les feuilles de l'arbre. Particularité assez bluffante, le logiciel a détecté un processeur « dual core » sur mon ordinateur. Il propose d'utiliser 2 threads pour les calculs. Pour mettre le logiciel sur un même pied d'égalité que les autres, nous ramenons la valeur à 1.

Il ne reste plus qu'à lancer les calculs en actionnant le menu contextuel EXECUTE AND OPEN VIEW. Après 270 secondes, la fenêtre de visualisation apparaît. Assez curieusement, aucune indication n'est fournie quant aux caractéristiques de l'arbre, je me vois très mal compter les feuilles à la main. Cela restera un mystère. A la fin des opérations, l'occupation mémoire est de 245.8 Mo, idem au plus fort des calculs.



Il est possible de sauvegarder la filière. Il semble cependant que le logiciel cherche à tout stocker dans son format interne, y compris les données. Le temps de traitement peut alors être très important sur des fichiers comme le nôtre.

3.2 ORANGE

ORANGE est développé par le Laboratoire d'Intelligence Artificielle de l'Université de Ljubljana, en Slovénie (http://www.ailab.si/orange/). Il peut être utilisé de deux manières : via la filière, avec le schéma composants – lien entre composant (Figure 3) ; ou via des scripts en Python. Il est donc possible de programmer des analyses complexes, avec des vraies structures algorithmiques.

Pour qu'ORANGE fonctionne correctement, la machine virtuelle (l'interpréteur) Python doit être présente sur la machine. Le programme d'installation du logiciel gère cela très bien.

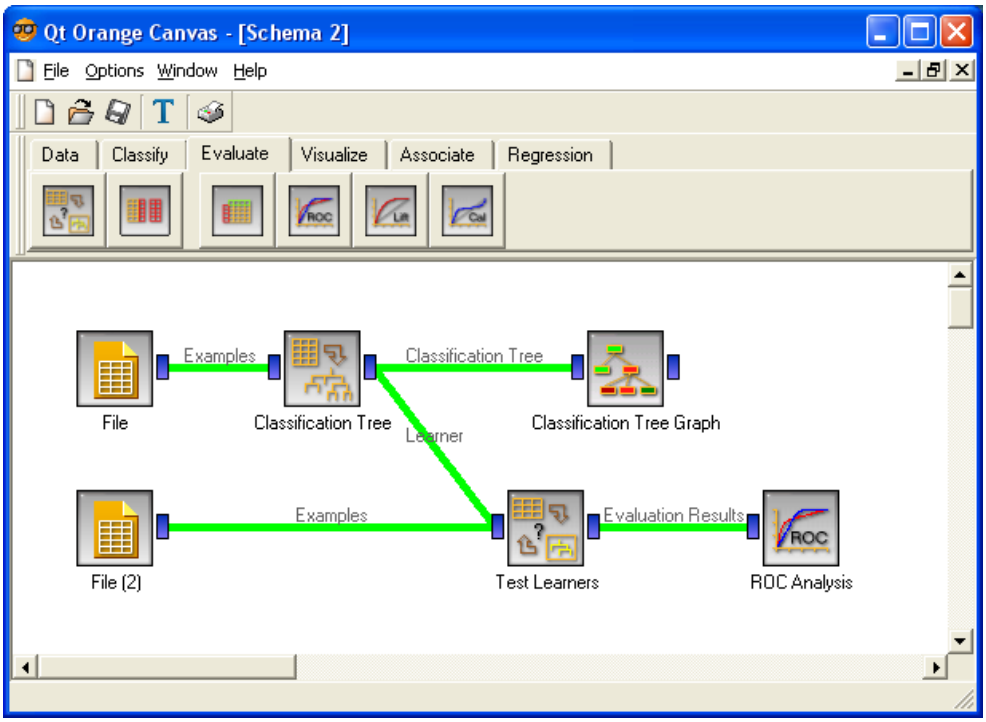
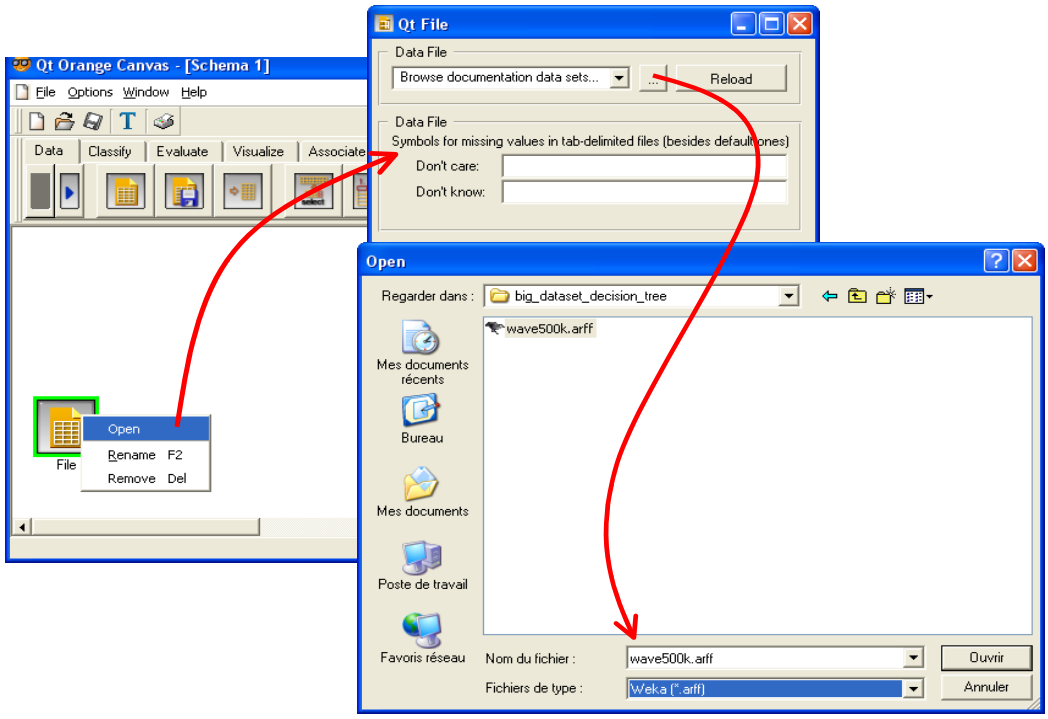


Figure 3 - Apprentissage et évaluation d'un arbre de décision sous ORANGE

ORANGE est clairement d'obédience « machine learning ». On y retrouve essentiellement la trilogie : méthodes supervisées, non supervisées et règles d'association. Particularité intéressante, il intègre des outils graphiques exploratoires interactifs, il est possible par exemple de sélectionner des individus dans un nuage de points et obtenir automatiquement la liste correspondante dans une grille de données.

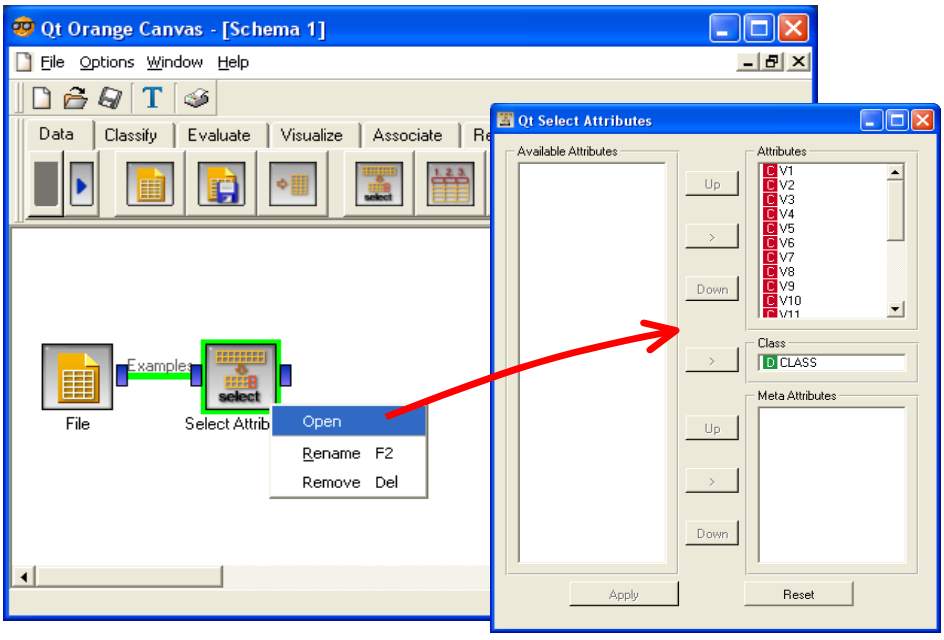
ORANGE a beaucoup évolué depuis le dernier comparatif que j'ai réalisé (http://tutoriels-data-mining.blogspot.com/2008/04/les-logiciels-gratuits-pour.html - décembre 2005). Il y avait de gros problèmes de gestion mémoire à l'époque, essentiellement imputable au moteur Python semble-t-il. Tout cela semble oublié maintenant. Le logiciel a pu traiter sans plantage le fichier que nous lui avons proposé.

Au lancement du logiciel, nous retrouvons une interface familière, avec la palette de composants et un espace de travail qui permet de définir des schémas de traitements. L'occupation mémoire est de 24.9 Mo. Nous introduisons le composant FILE (onglet DATA). Nous actionnons le menu contextuel OPEN pour accéder au fichier. Attention, dès que l'on a sélectionné le fichier dans la boîte de dialogue, le chargement est automatiquement lancé. Il n'y a pas de menu spécifique pour lancer les traitements.

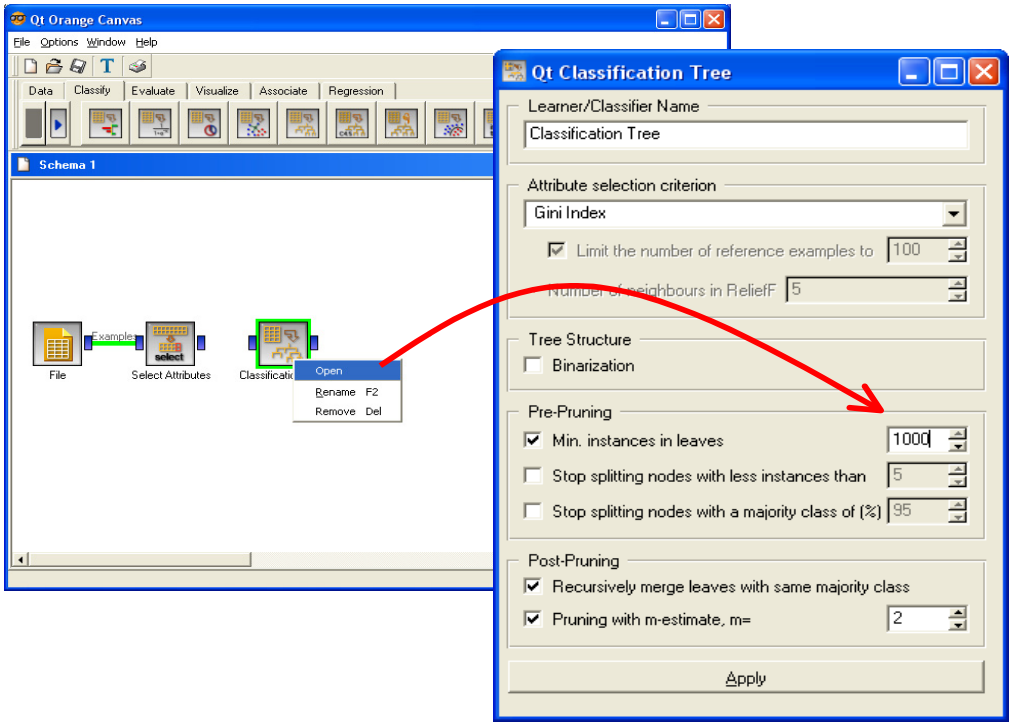


L'opération dure 90 secondes, l'occupation mémoire après coup est de 259.5 Mo mais, assez curieusement, à certains moments durant l'importation, elle est montée à 480 Mo. Elle est très fluctuante d'ailleurs dès que l'on réalise des manipulations dans le logiciel.

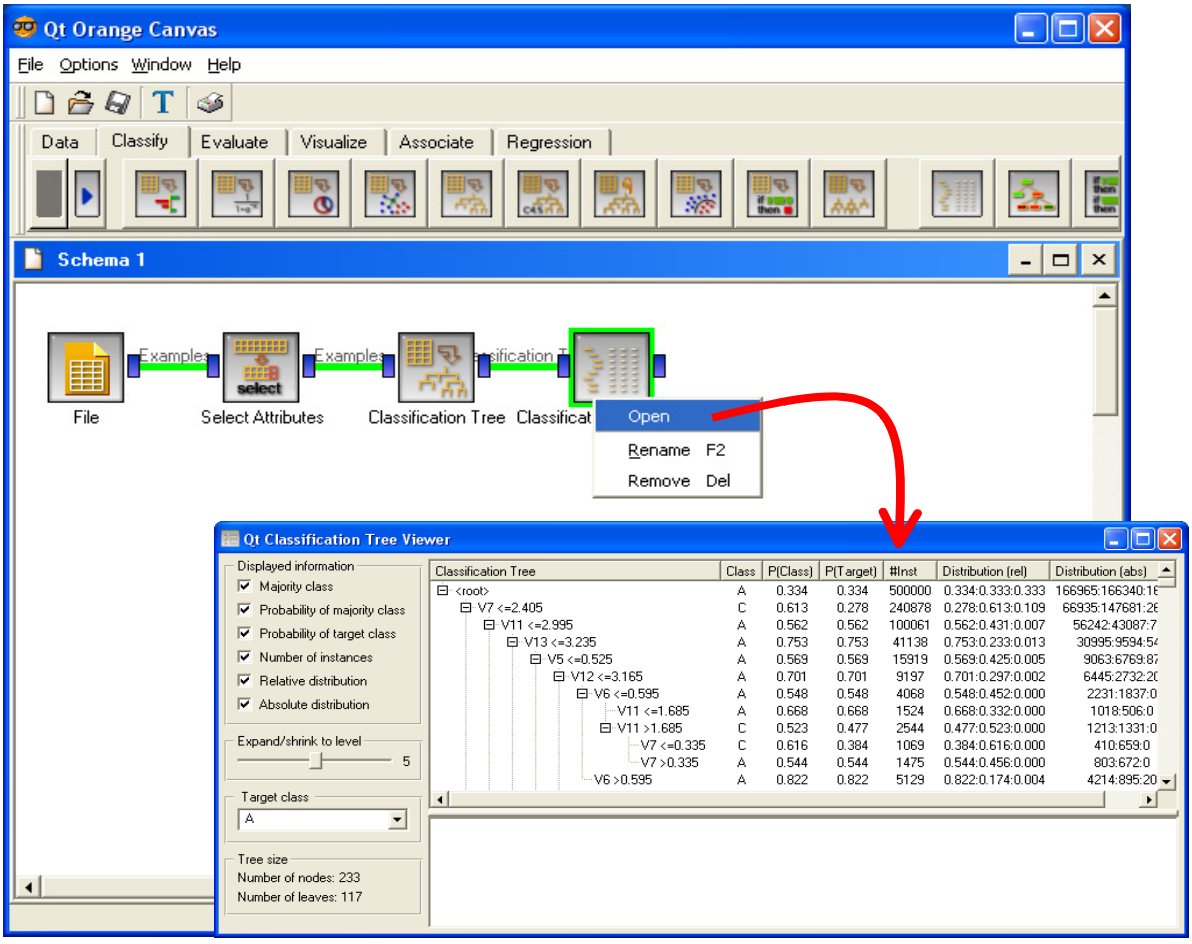
Nous plaçons le composant SELECT ATTRIBUTE dans le schéma. Le composant FILE lui est connecté, nous actionnons alors le menu contextuel OPEN pour préciser le rôle des variables. CLASS est la variable à prédire, les autres sont les prédictives.



Nous plaçons le composant CLASSIFICATION TREE (onglet CLASSIFY) dans l'espace de travail. **Attention, il ne faut surtout pas établir la connexion à ce stade, sinon les calculs seront directement lancés.** Nous devons tout d'abord procéder au paramétrage de la méthode en actionnant le menu contextuel OPEN. Dans la boîte de dialogue, nous définissons les paramètres adéquats afin qu'ORANGE effectue des calculs similaires aux autres logiciels de ce comparatif.



Nous insérons l'outil de visualisation des arbres CLASSIFICATION TREE VIEWER (onglet CLASSIFY). Nous lui connectons le composant précédent. Enfin, pour lancer le traitement, il ne reste plus qu'à connecter le composant SELECT ATTRIBUTES à CLASSIFICATION TREE. La construction de l'arbre est automatiquement démarrée.



L'opération a duré 130 secondes, l'occupation mémoire est de 795.7 Mo à la sortie. L'arbre comporte 117 feuilles.

Notons pour terminer qu'ORANGE propose un composant pour l'exploration interactive des arbres. Un autre outil permet de produire une représentation graphique de l'arbre, fort pimpante au demeurant. A utiliser avec parcimonie cependant. Il a du mal à gérer les arbres complexes, comportant un grand nombre de feuilles. C'est pour cette raison que nous ne l'avions pas mis à contribution dans ce didacticiel.

3.3 Le logiciel R avec le package RPART

Le logiciel R était surtout connu des statisticiens (http://www.r-project.org/), il l'est de plus en plus des data miner. A raison. Voilà un projet très enthousiasmant, avec une organisation suffisamment bien pensée pour que tout un chacun puisse y contribuer sans avoir à faire des contorsions plus ou moins bien maîtrisées.

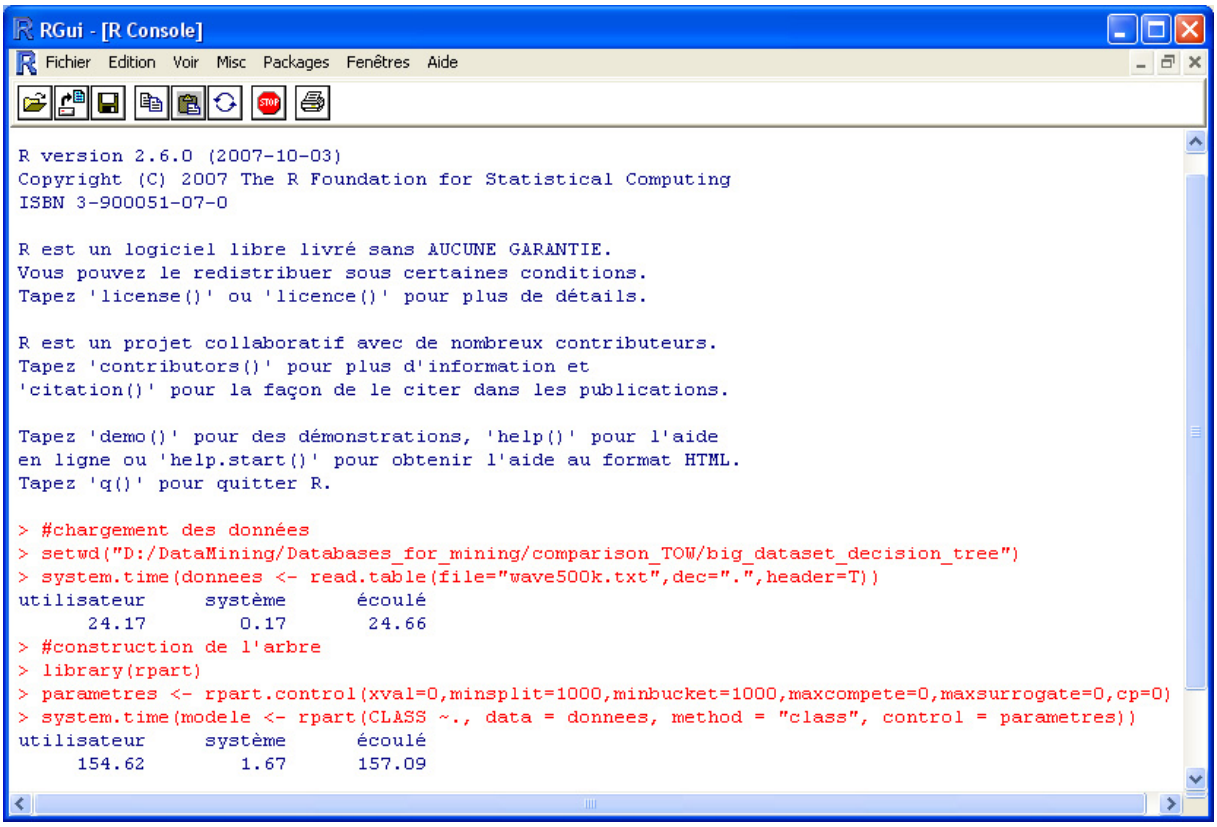
R est un environnement de traitement, avec un système de gestion de données et un vrai langage de programmation. Nous pouvons écrire des scripts pour définir des tâches ou créer de nouvelles techniques. L'autre force de R est le système de package, des plugins externes que l'on peut programmer dans n'importe quel langage compilé et interfacer avec R. Les règles de gestion semblent suffisamment simples et claires si l'on se réfère au nombre des contributeurs qui augmentent de jour en jour.

La multiplicité des packages est un atout maître (http://cran.univ-lyon1.fr/web/packages/). La bibliothèque de techniques est potentiellement infinie, l'expérience montre qu'en cherchant un peu, on est quasiment sûr de trouver quelque chose qui répond aux préoccupations de l'utilisateur.

Mais la médaille a un revers. La multiplicité est certes gage de richesse, mais elle est aussi source de confusion. On passe beaucoup de temps à chercher ce dont on a besoin. Les packages sont de qualité très inégale. On ne sait pas toujours ce qui est réellement programmé, différents modules semblent dévolues aux mêmes tâches, sans pour autant donner exactement les mêmes résultats. Lorsque l'on voit les efforts que font certains logiciels commerciaux pour réaliser une documentation de qualité, qu'ils mettent gratuitement accessible en ligne. On se rend compte qu'il y a des enjeux forts derrière.

Néanmoins, R est véritablement un projet formidable, d'autant plus que le système est fiable et performant. Voyons voir justement comment il se comporte dans notre comparatif.

R fonctionne à l'aide de scripts. Voici les commandes que nous avons utilisées, et les temps de calcul que nous avons pu mesurer à l'aide de la fonction **system.time(.)**



Au démarrage de R, l'occupation mémoire est de 18.8 Mo. R ne sait pas lire directement les fichiers ARFF, en tous les cas, je n'ai pas trouvé de package pour le faire. Le fichier texte WAVE500K.TXT a donc été utilisé. Le temps de chargement a été de 24 secondes, l'espace dévolu à R dans Windows est passé à 184.1 Mo.

La méthode C4.5 n'est pas disponible. Nous avons utilisé le package RPART qui implémente la méthode CART (Breiman et al., 1984). Le paramétrage a été défini de manière à nous rapprocher de notre algorithme de référence dans ce comparatif. Ainsi : MINSPLIT empêche toute segmentation si un nœud comporte moins de 1000 observations ; MINBUCKET empêche la création d'une feuille comportant moins de 1000 observations ; avec CP = 0, nous assurons la construction « hurdling » d'un arbre maximal. Il n'y a pas de post-élagage dans notre dispositif. Ca ne pose pas de problèmes, en effet le temps dévolu à cette partie des calculs est relativement faible avec C4.5, la procédure ne nécessitant pas d'accès aux données.

Le temps écoulé pour la construction de l'arbre est de 157 secondes (environ 2 minutes), R occupe alors 718.9 Mo en mémoire centrale.

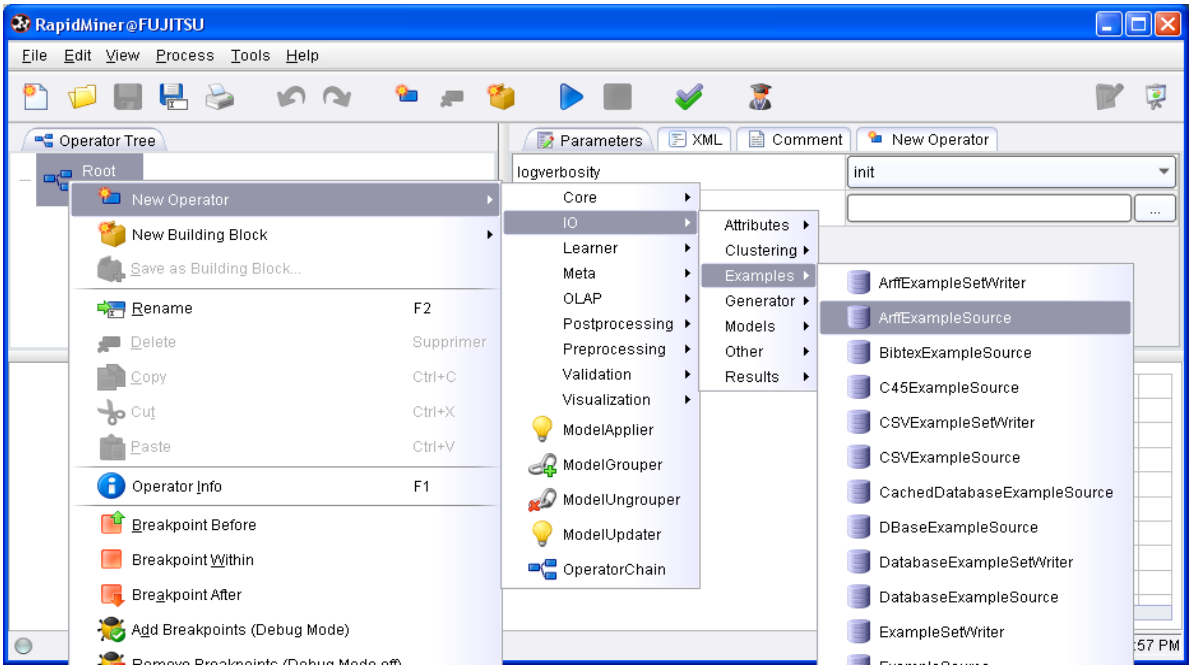
3.4 RAPIDMINER (anciennement YALE)

Le logiciel RAPIDMINER est issu du projet YALE de l'équipe « Intelligence Articificielle » de l'Université de Dortmund (http://www.rapidminer.com/). Le projet a été repris par la Société Rapid-I qui maintient deux versions en parallèle : la « Community Edition » est distribuée gratuitement, la « Enterprise Edition » est commerciale. Nous utilisons bien évidemment la première dans ce comparatif.

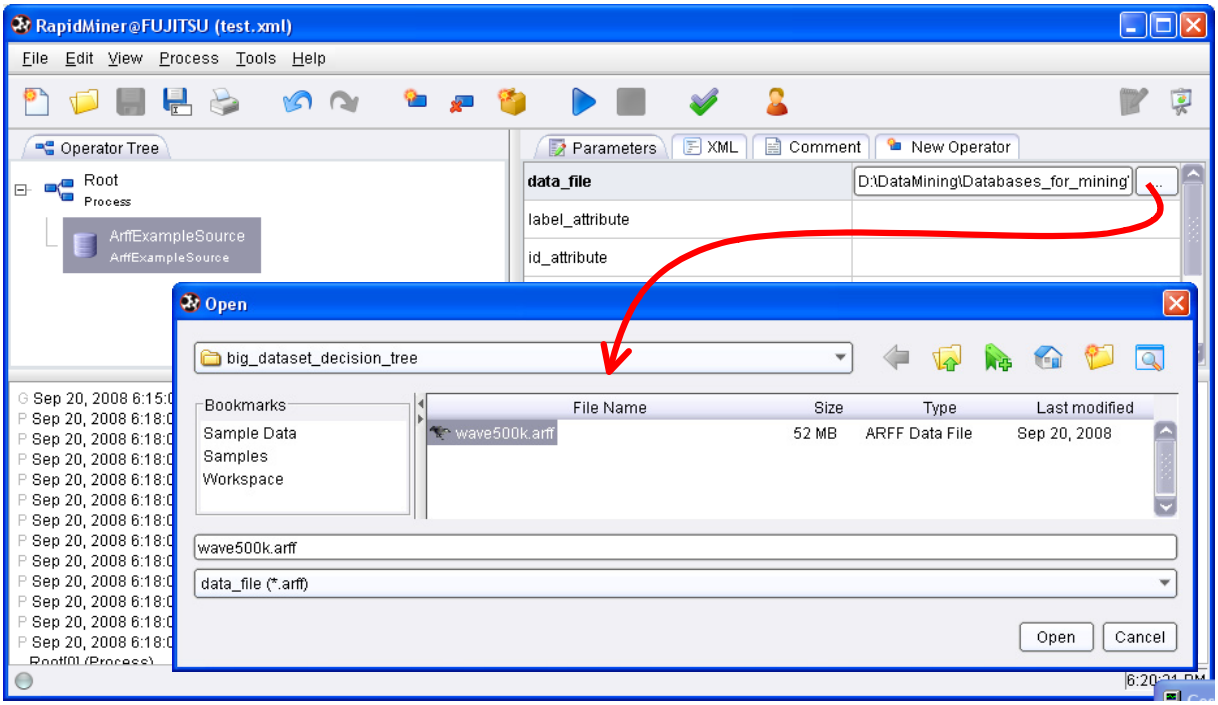
RAPIDMINER reprend le cadre de YALE, avec une bibliothèque de fonctions très riche. Il fonctionne par filières, avec une structure très particulière. La chaîne de traitement est linéaire, mais certains nœuds peuvent représenter un groupe de traitements, avec des ramifications sous forme arborescente. La structure est décrite à l'aide d'un formalisme XML.

Par rapport à YALE toujours, de très gros efforts en termes de fonctionnalités et d'ergonomie ont été réalisés. Néanmoins, au premier abord, le logiciel peut paraître confus, avec des imbrications de menus sur plusieurs niveaux, jamais très propices à une prise en main facile. Après une période d'adaptation, on se rend compte que l'organisation a le mérite de la cohérence.

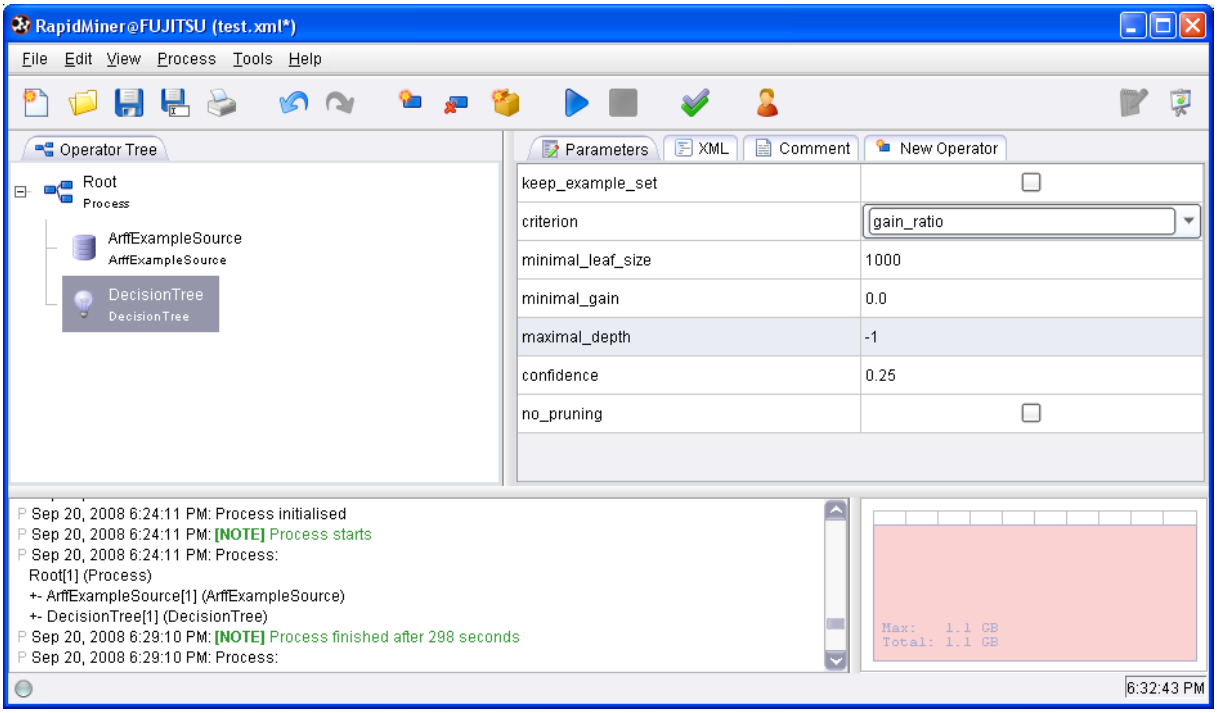
Au démarrage du logiciel, nous disposons d'une structure vide. Nous demandons la création d'un nouveau projet via le menu FILE / NEW. La racine du projet est disponible, nous lui adjoignons la source de données au format ARFF en utilisant le menu contextuel NEW OPERATOR.



Le composant apparaît dans le diagramme, nous le paramétrons en désignant notre fichier de données. Le bouton PLAY dans la barre d'outils permet de lancer l'importation. Elle dure 7 secondes. Concernant la mémoire allouée dans Windows, elle était de 136.3 Mo au démarrage du logiciel, elle passe à 228.1 Mo après le chargement des données.



Il faut maintenant placer le composant « Arbre de décision » dans le diagramme. Nous actionnons de nouveau le menu contextuel NEW OPERATOR sur la racine de l'arborescence. Dans les dédales des menus imbriqués, nous trouvons la procédure DECISIONTREE. Nous la paramétrons pour obtenir des calculs cohérents avec ceux des autres logiciels.



Le bouton PLAY lance les calculs. Au bout de 298 secondes, nous obtenons l'arbre de décision.

Assez curieusement, RAPIDMINER est très gourmand en ressources. 1274.4 Mo lui sont nécessaires. Ca paraît bien excessif par rapport aux autres logiciels. Il faut prendre ce chiffre avec beaucoup de précautions, d'autant que l'arbre produit ne comporte que 4 feuilles. Ce qui est un autre mystère. Nous avions pourtant multiplié les tentatives, à chaque fois nous avons obtenu le même arbre, avec des caractéristiques mémoire et temps de calcul du même ordre.

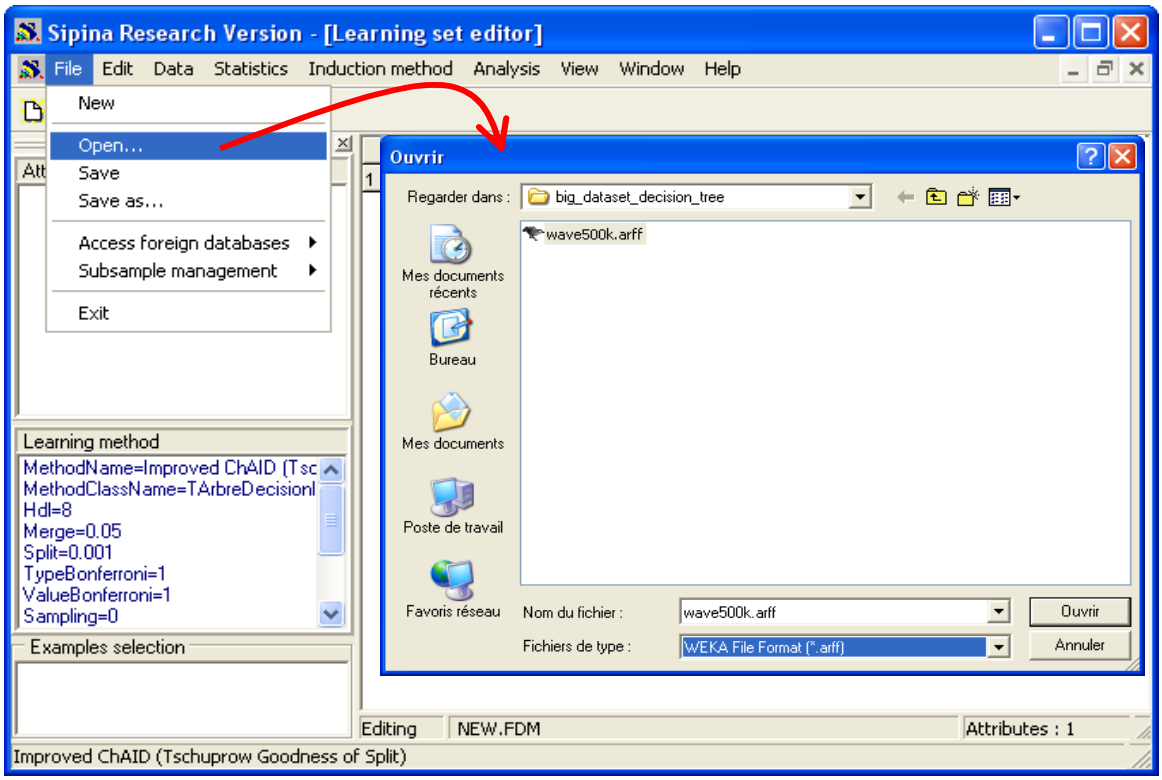
3.5 SIPINA

SIPINA est un de mes anciens projets de logiciel de Data Mining, spécialisé dans les arbres de décision (http://sipina.over-blog.fr/). Il est piloté par menu. Il n'est pas possible de programmer des séquences de traitements. Pour différentes raisons, le développement a été arrêté en 2000. Mais Il est distribué encore parce que c'est un des très rares logiciels gratuits au monde à proposer des fonctionnalités interactives comparables à celles des logiciels commerciaux. Nous disposons à tout moment d'un rapport complet sur la pertinence des variables, des statistiques descriptives comparatives nous permettent d'interpréter finement les spécificités des groupes d'individus associés aux nœuds, nous pouvons élaguer l'arbre manuellement, induire de nouvelles partitions en choisissant la variable de segmentation, etc.

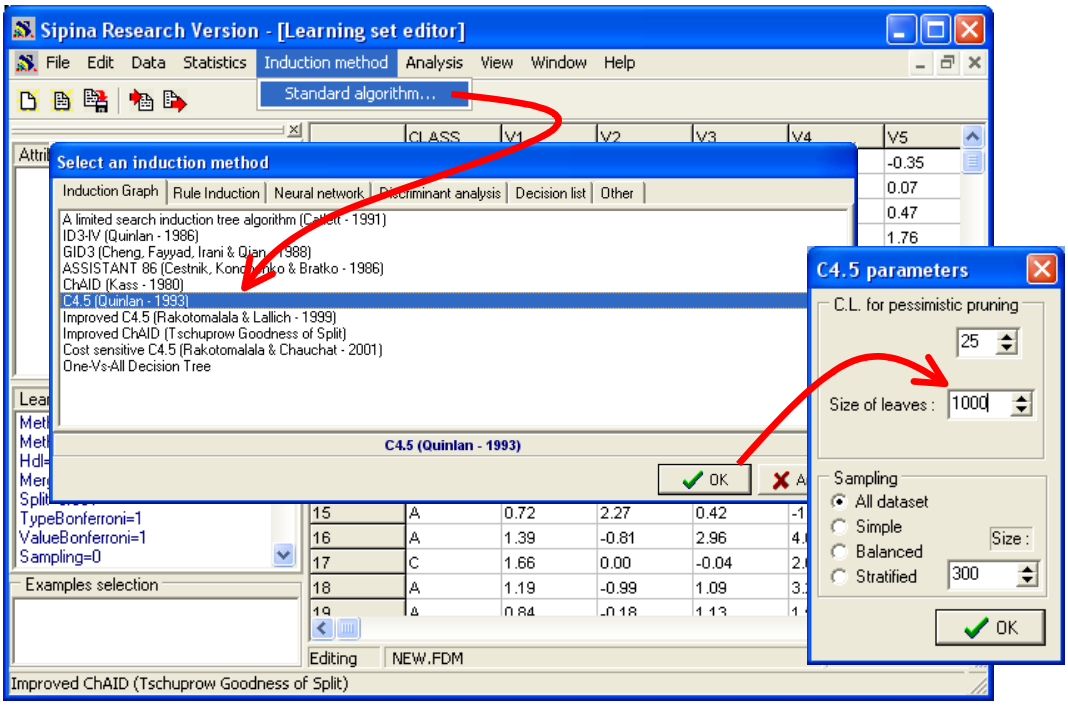
Ces fonctionnalités constituent indéniablement un atout dans l'exploration des données. Mais dans ce comparatif, elles représentent un inconvénient. En effet, nombre d'informations sont systématiquement conservées pour chaque nœud de l'arbre : recensement des variables candidates, leurs performances respectives, les partitions induites et les distributions associées, les statistiques descriptives comparatives, et même dans certaines circonstances, la liste des observations, nécessaire lorsque l'utilisateur veut visualiser la grille des données. L'occupation mémoire devient nécessairement un problème dès que l'arbre atteint une certaine taille. SIPINA est

donc pénalisé par rapport aux autres logiciels de ce comparatif. Il propose des outils qui la désavantagent dans le cadre de notre expérimentation.

Au lancement de SIPINA, son occupation mémoire est de 7.8 Mo. Nous actionnons le menu FILE / OPEN pour charger le fichier WAVE500K.ARFF au format WEKA.

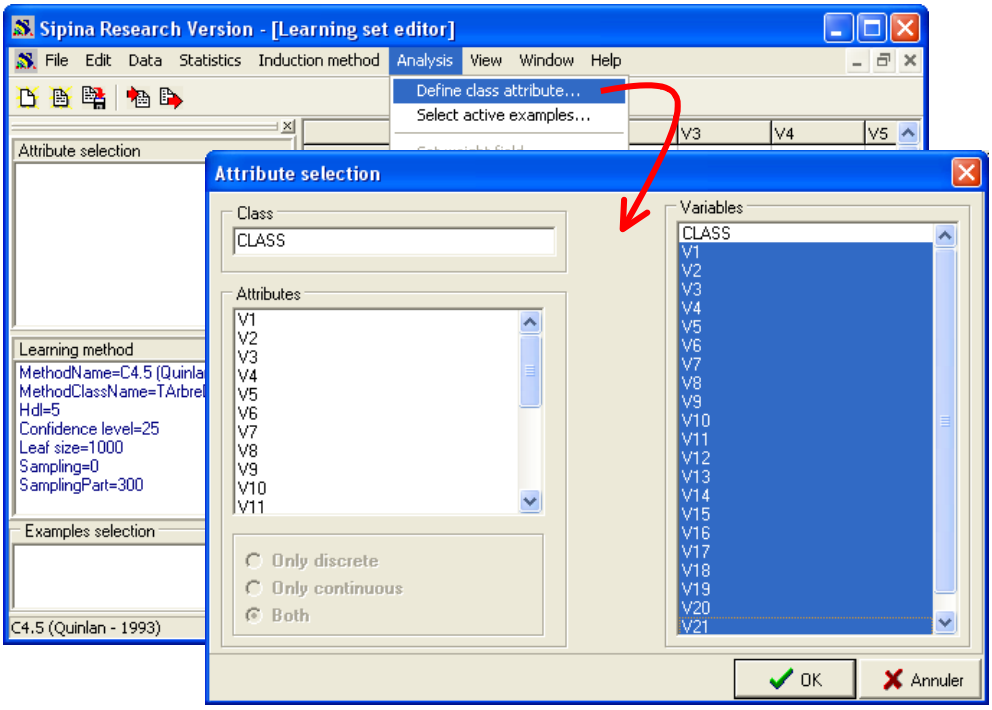


L'importation dure 25 secondes, l'occupation mémoire est passée à 67.1 Mo. Nous sélection le menu INDUCTION METHOD / STANDARD ALGORITHM pour choisir la méthode C4.5. Dans la boîte de paramétrage, nous fixons la taille minimale des sommets à 1000 individus.

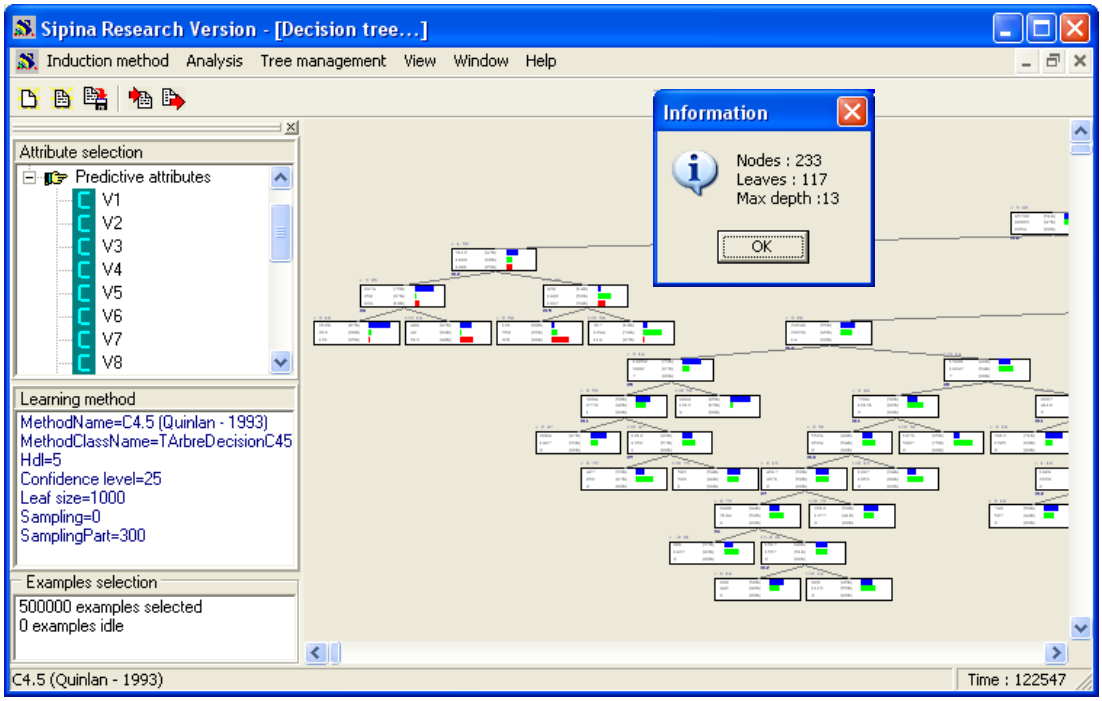


Signalons une option intéressante, que nous n'exploitons cependant pas dans ce comparatif, dans la partie basse de la boîte de paramétrage. Il est possible de réaliser les calculs à partir d'un échantillon d'observations sur chaque nœud. On s'est rendu compte en effet qu'utiliser quelques centaines d'observations pour le choix de la variable de segmentation réduit considérablement le temps de calcul sans dégrader la qualité de la prédiction. Plus particulièrement lorsque la proportion de variables candidates continues est élevée.

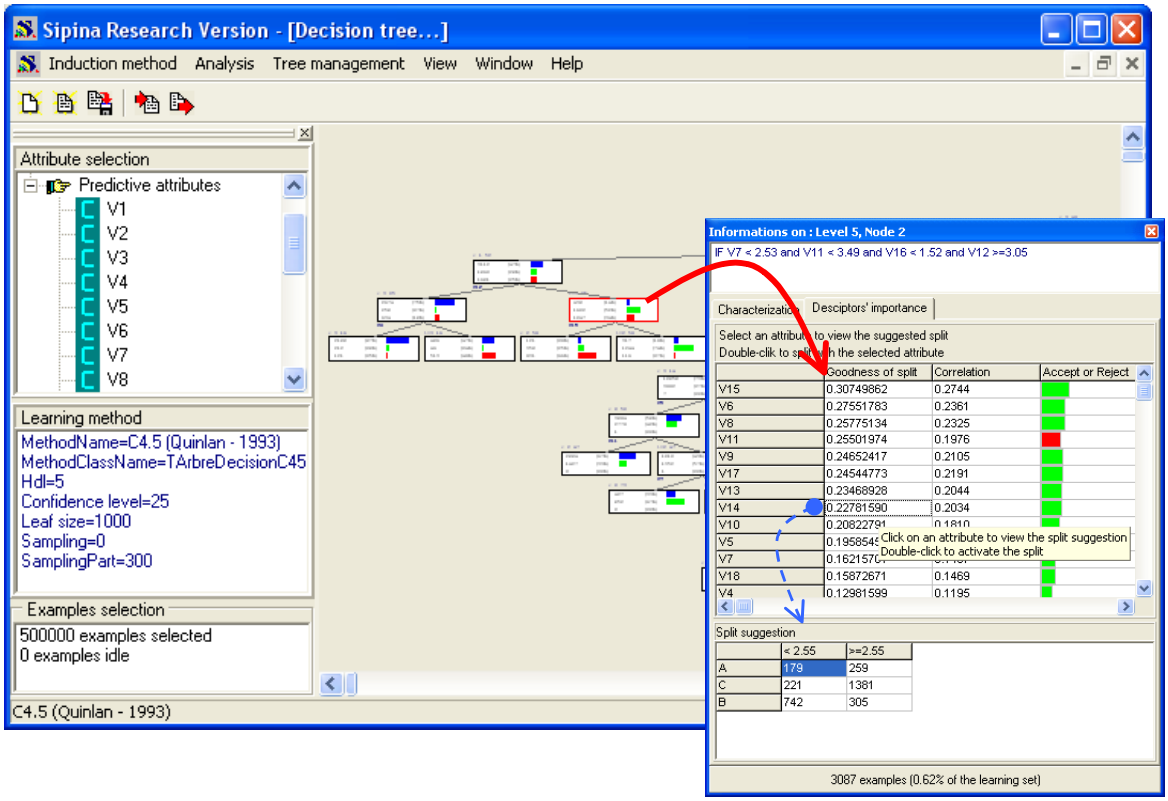
Nous devons définir le rôle des variables dans l'analyse. Nous activons le menu ANALYSIS / DEFINE CLASS ATTRIBUTE. Nous plaçons CLASS en TARGET, les autres variables V1 à V21 en INPUT.



Nous pouvons lancer l'apprentissage. Nous activons le menu ANALYSIS / LEARNING... Le calcul dure 122 secondes. L'occupation mémoire est passée à 539.9 Mo, avec un arbre à 117 feuilles.



Il est clair qu'avec une machine moins puissante, nous aurions eu des problèmes. Mais comme nous l'indiquions plus haut, SIPINA propose en contrepartie la possibilité d'explorer finement les nœuds. Dans la copie d'écran ci-dessous, pour le nœud sélectionné, nous pouvons visualiser les performances de l'ensemble des variables candidates ; et pour chaque variable, la distribution des classes dans les feuilles si l'on introduisait la segmentation.

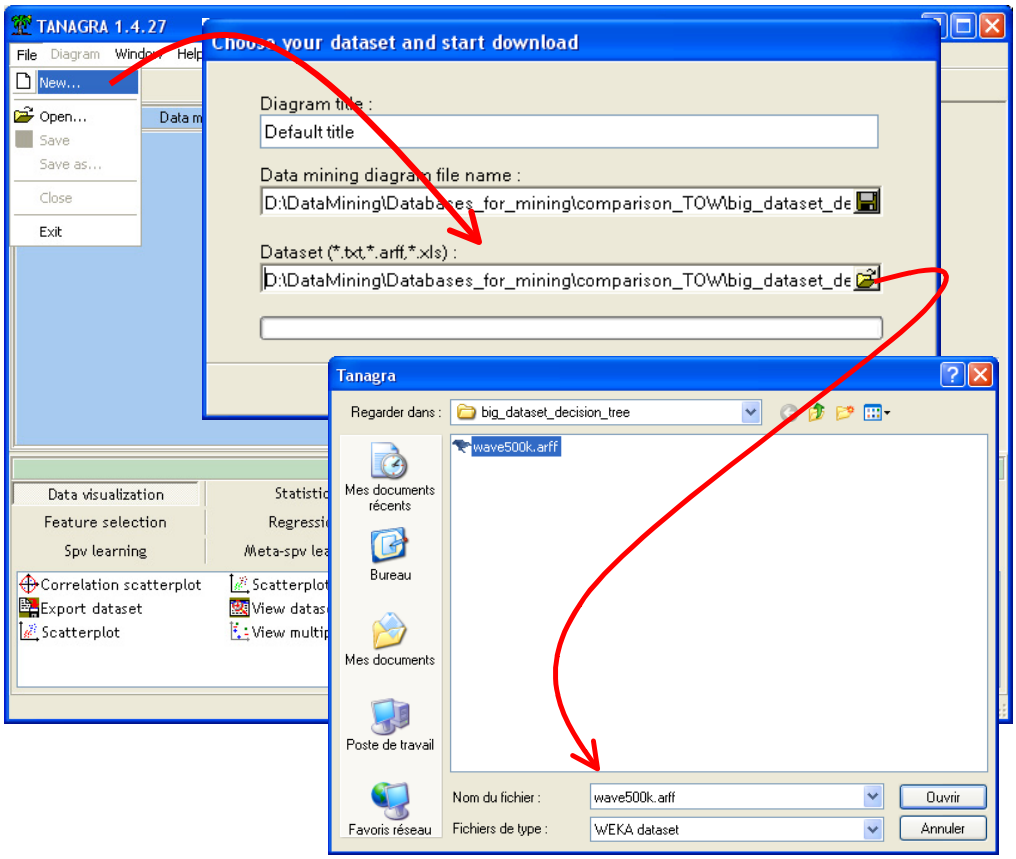


3.6 TANAGRA

TANAGRA est mon projet actuel (http://eric.univ-lyon2.fr/~ricco/tanagra/). Il adopte la présentation filière qui permet de définir des séquences de traitements.

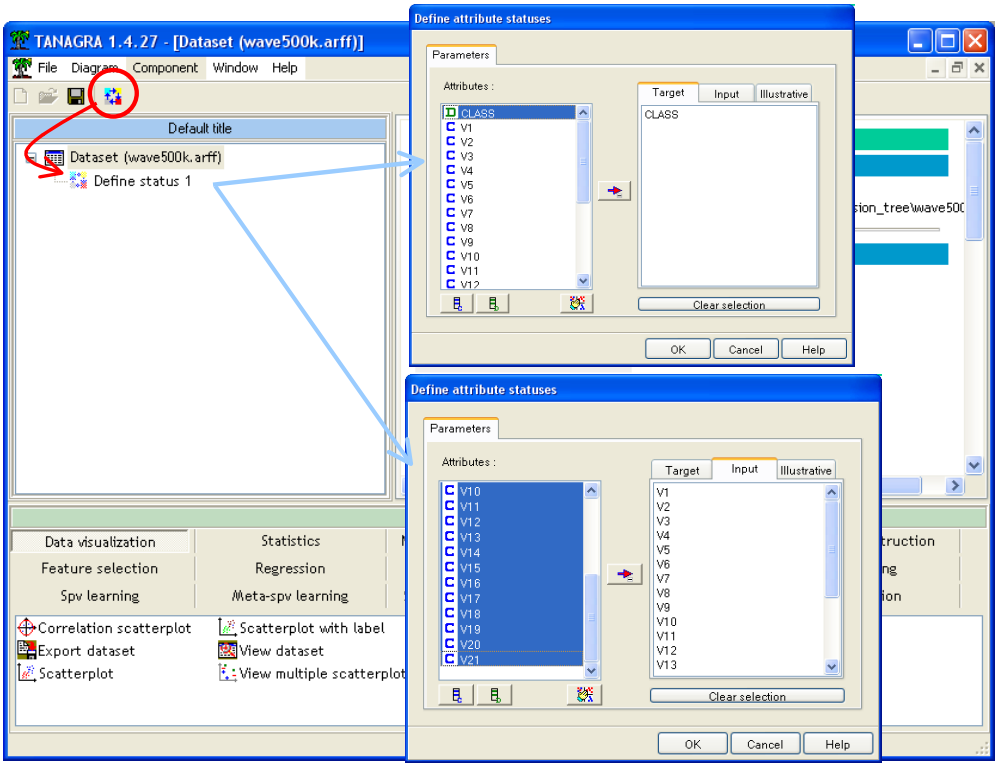
Par rapport à SIPINA, l'implémentation des arbres a été modifiée. Pour chaque descripteur continu, un index est élaboré au préalable. L'avantage est évident, le tri n'est réalisé qu'une seule fois pour chaque variable, avant la construction de l'arbre. L'inconvénient est que cet index doit être conservé quelque part, de fait nous allouons (4 octets) x (nombre d'observations) x (nombre de descripteurs continus) en mémoire centrale avant même le début de la construction de l'arbre. Ainsi, la mémoire réservée durant le traitement est nettement plus élevée qu'à la fin.

Au démarrage, l'occupation mémoire est de 7 Mo. Pour importer un fichier WEKA dans TANAGRA, nous actionnons le menu FILE / NEW. Un nouveau diagramme est crée avec le fichier WAVE500K.ARFF comme source de données.

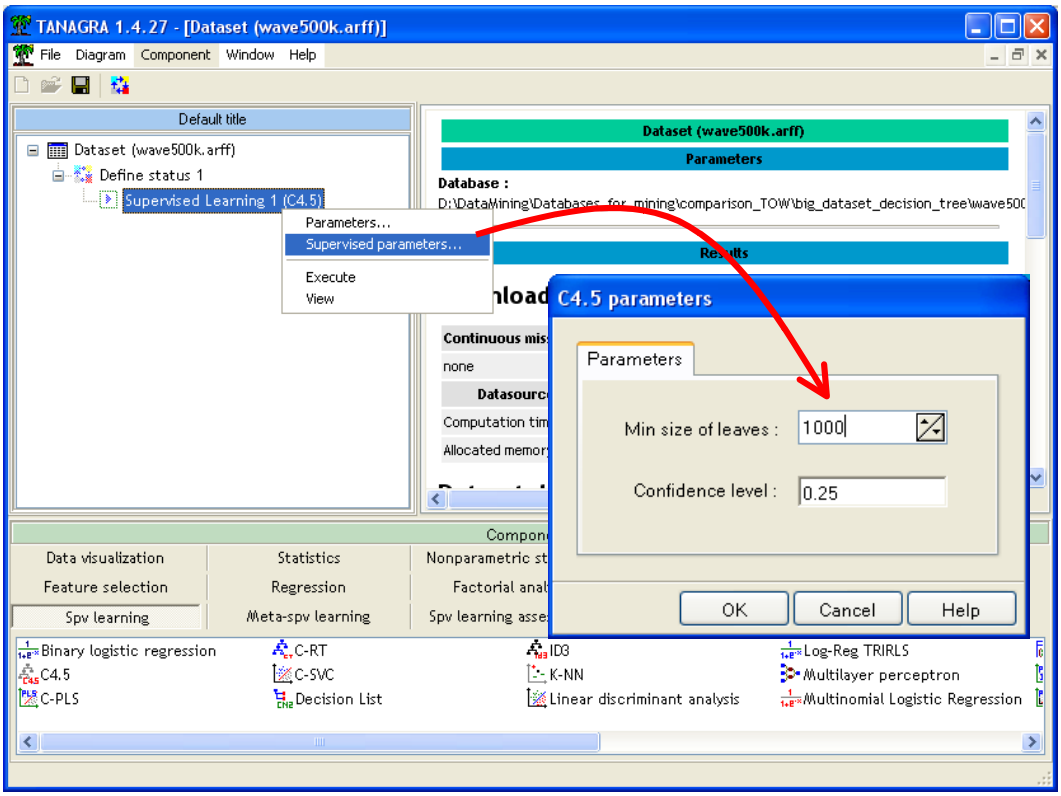


L'importation dure 11 secondes, l'occupation mémoire est passée à 53.1 Mo.

Pour lancer le traitement, nous devons tout d'abord définir le rôle des variables. Nous introduisons pour cela le composant DEFINE STATUS via le raccourci dans la barre d'outils. La variable CLASS est placée en TARGET, les autres en INPUT.



Il ne reste plus qu'à introduire le composant C4.5 et à le paramétrer.



Le menu contextuel VIEW permet de lancer les calculs. La construction de l'arbre a duré 33 secondes, l'occupation mémoire à la fin du processus est de 73.5 Mo, le pic a été de 121.6 Mo durant la construction. L'arbre comporte 117 feuilles.

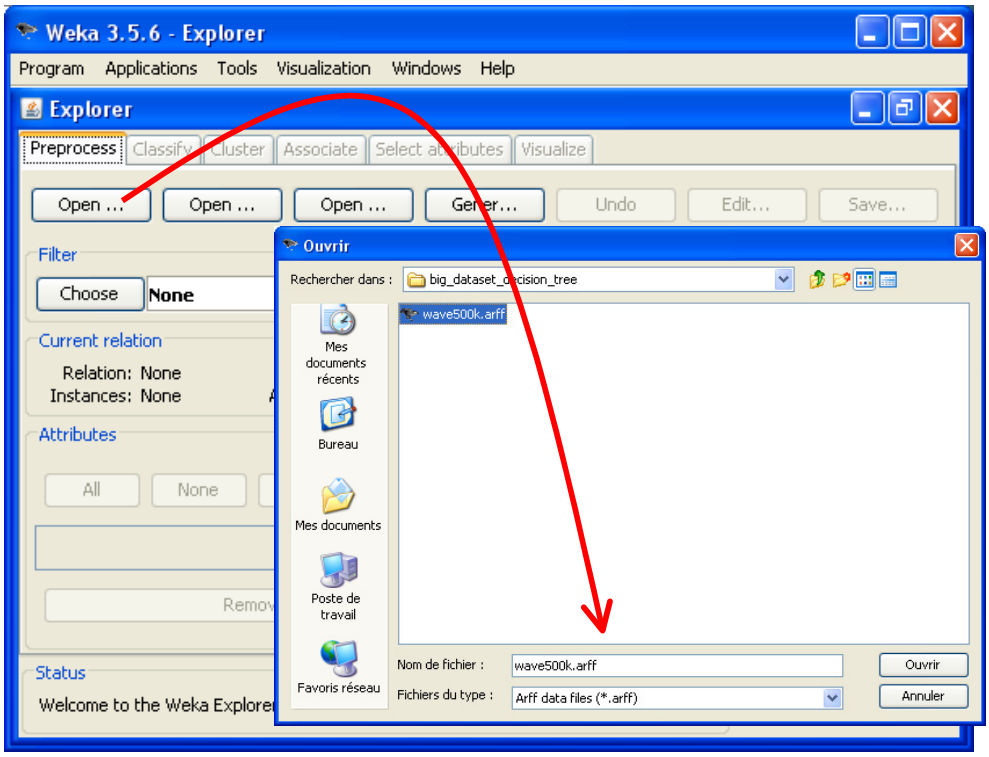
3.7 WEKA

WEKA est un logiciel incontournable (http://www.cs.waikato.ac.nz/ml/weka/). Il intègre un très grand nombre de techniques, essentiellement d'obédience « machine learning ». Il propose peu de techniques issues de la statistique.

WEKA peut fonctionner par filière (Knowledgeflow) ou être piloté par menu (Explorer). Il est également possible de l'utiliser en ligne de commande. Dans notre cas, nous utiliserons le module APPLICATIONS / EXPLORER, très simple à appréhender. WEKA repose sur la technologie JAVA, la machine virtuelle est démarrée automatiquement au lancement du logiciel. L'occupation mémoire s'en ressent, elle est de 53.1 Mo.

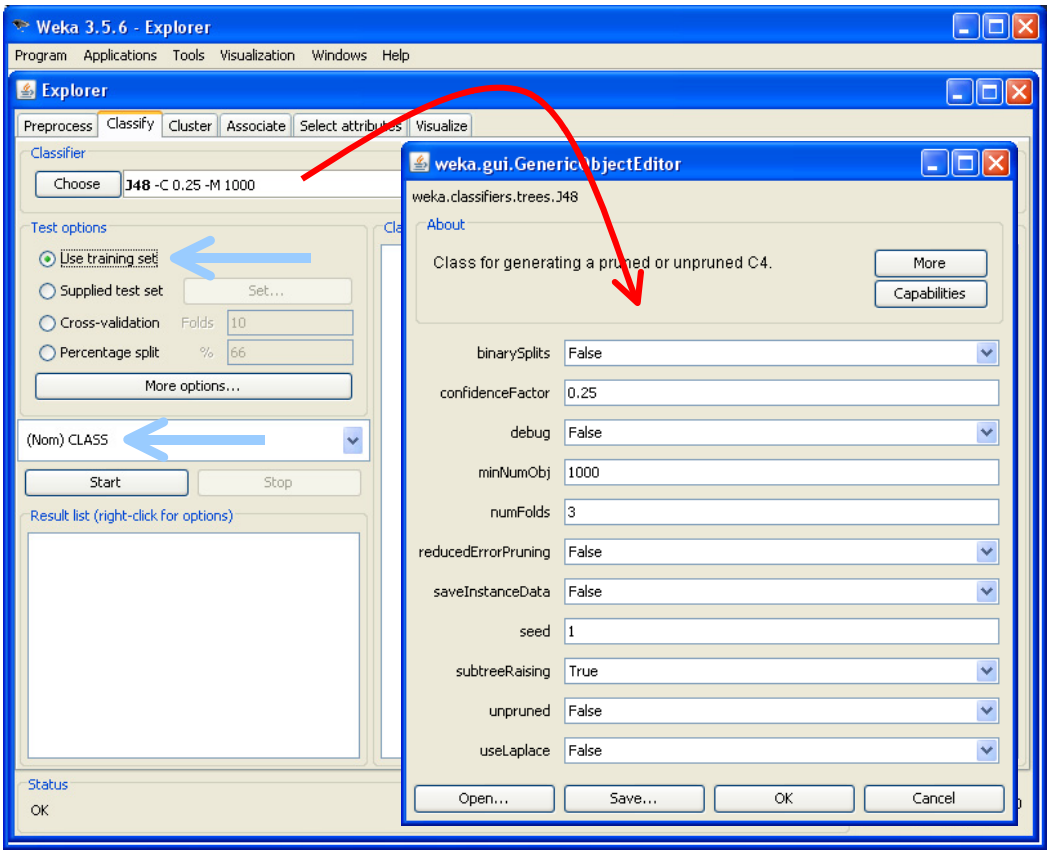
WEKA a énormément progressé. Il n'y a pas si longtemps (http://tutoriels-data-mining.blogspot.com/2008/04/les-logiciels-gratuits-pour.html - décembre 2005), il était impossible de charger un fichier un tant soit peu conséquent. Comme les classes de gestion de données n'ont pas été modifiées entre temps (du moins dans mes comparaisons du code source des deux versions), il faut y voir surtout une amélioration drastique de la machine virtuelle JAVA.

Nous actionnons le bouton OPEN FILE de l'onglet PREPROCESS, nous sélectionnons le fichier WAVE500K.ARFF.



Le temps de chargement est de 10 secondes, la mémoire occupée est maintenant de 253.2 Mo.

Une fois les données chargées, nous pouvons accéder à l'onglet CLASSIFIY. La variable à prédire est CLASS, nous choisissons la méthode J48 dans la famille TREE, tous les individus doivent participer à l'apprentissage. Nous précisons qu'il ne peut y avoir moins de 1000 observations par feuille. Il ne reste plus qu'à cliquer sur le bouton START.



La construction de l'arbre a pris 338 secondes (environ 5 minutes), l'espace mémoire occupée après coup est de 699.6 Mo, elle n'a jamais dépassé cette valeur durant les calculs. L'arbre produit comporte 123 feuilles.

3.8 Récapitulatif

Un petit récapitulatif pour avoir une vue globale des performances :

Logiciel	**Temps de traitement (secondes)**		**Occupation mémoire (Mo)**			
	Importation	Induction arbre	Avant lancement	Après importation	Pic traitement	Après induction
KNIME	**47**	**270**	92.6	160.4	245.8	245.8
ORANGE	**90**	**130**	24.9	259.5	795.7	795.7
R (package rpart)	**24**	**157**	18.8	184.1	718.9	718.9
RAPIDMINER	**7**	**298**	136.3	228.1	1274.4	1274.4
SIPINA	**25**	**122**	7.8	67.1	539.9	539.9
TANAGRA	**11**	**33**	7.0	53.1	121.6	73.5
WEKA	**10**	**338**	52.3	253.2	699.6	699.6

Occupation mémoire. Pouvoir fonctionner avec une machine virtuelle est indéniablement un atout. Sous JAVA ou PYTHON, le logiciel est directement opérationnel quel que soit le système d'exploitation. C'est le cas de KNIME, ORANGE, RAPIDMINER et WEKA. Mais cette portabilité devient un inconvénient lorsque l'on traite de grandes bases de données. L'environnement complet est gourmand en ressources système. L'occupation mémoire atteint très vite un niveau critique. La situation n'est pas figée pour autant. Il y a 3 ans, traiter un fichier de 50.000 observations constituait un problème pour ORANGE. L'évolution est quand même impressionnante. Rien ne permet de dire qu'il n'en sera pas de même pour les années à venir. Affaire à suivre donc.

Le cas de R est finalement assez similaire. Il agit aussi comme un interpréteur de commande multi plate-forme. Le script de notre expérimentation aurait tout aussi bien fonctionné sur une version de R pour Linux. R joue le rôle de machine virtuelle. Il n'est donc pas étonnant de retrouver les mêmes chiffres que pour les programmes fonctionnant via JAVA ou PYTHON.

Deux situations s'opposent cependant : KNIME a une excellente tenue, on note par ailleurs qu'une option permettant de swaper les données sur le disque existe sur le composant d'accès aux fichiers, réduisant d'autant l'occupation mémoire ; à l'inverse, le cas de RAPIDMINER laisse perplexe, je n'ai pas réellement d'explications ici.

Les programmes compilés sont les moins gourmands. SIPINA est bien entendu un cas à part pour les raisons avancées ci-dessus. Les fonctionnalités interactives nous obligent à conserver un maximum d'informations sur les nœuds. Il reste néanmoins que l'espace final occupé est plus faible que pour la majorité des programmes pseudo-compilés. TANAGRA de son côté est nettement moins vorace que les autres. Même au plus fort des calculs lorsque un grand nombre d'informations temporaires (index principalement) sont générés, il utilise moitié moins de mémoire que le programme le plus parcimonieux suivant (KNIME).

Rapidité des calculs. On observe très distinctement 3 groupes. Les programmes sous JAVA sont les plus lents. La machine virtuelle disponible sous Windows a encore des progrès à faire. Peut être aussi que cette unité de comportement est due au fait que tous sont programmés de la même manière. Il faudrait voir en détail dans le code source pour le vérifier.

Viennent ensuite 2 implémentations (ORANGE et RPART dans R) qui fonctionnent dans un environnement venant en surcouche par rapport à Windows. On pourrait de contenter de cette conclusion. Mais SIPINA, qui est compilé, propose des temps d'exécution assez similaires finalement.

L'autre explication possible est donc le traitement des variables continues qui sont répétitivement triées à chaque segmentation des nœuds. TANAGRA s'appuie sur une stratégie différente. Il réduit ainsi considérablement le temps de calcul, au prix d'une allocation mémoire supplémentaire pour les index intermédiaires, peu préjudiciable dans le contexte de notre étude, mais qui peut devenir problématique si le nombre de descripteurs continus est très élevé (de l'ordre de plusieurs milliers).

Enfin, nous noterons que le compilateur DELPHI 6.0 n'est pas si mal que ça finalement[4].

Note : Un didacticiel postérieur à celui-ci (voir http://tutoriels-data-mining.blogspot.com/2008/10/svm-comparaison-de-logiciels.html) nous fait mieux comprendre la nature des écarts de performances. WEKA, entre autres, organise les données en ligne. Il est de ce fait très rapide quand il faut procéder à des opérations nécessitant de passer très rapidement d'une variable à l'autre pour un même individu. C'est le cas des SVM (support vector machine) lors d'un produit scalaire entre deux vecteurs individus. TANAGRA en revanche organise ses données en colonnes. Il est très rapide lorsqu'il s'agit de parcourir très rapidement l'ensemble des observations pour une variable. C'est le cas pour les arbres de décision, lorsque nous procédons à une discrétisation par exemple. La véritable source des écarts de performances semble donc principalement reposer sur les différences entre les structures internes des logiciels.

4 Conclusion

N'allons surtout pas tomber dans le piège des jugements lapidaires du style « qui est bien, qui n'est pas bien, qui est le meilleur… ». D'autres critères sont très importants pour évaluer les logiciels : la portabilité ; l'ergonomie ; la richesse de la bibliothèque des méthodes ; les aspects pratiques qui permettent une prise en main facile ; l'accessibilité et la compréhensibilité des résultats ; etc.

Ce comparatif s'inscrit avant tout dans le cadre du **traitement des grandes bases**. **Les critères de rapidité et surtout d'occupation mémoire deviennent alors critiques**. L'objectif est de cerner le comportement de quelques logiciels libres dans ce contexte, lors de l'induction d'un arbre de décision à l'aide d'une méthode apparentée C4.5. Ils ont pour point commun de réaliser l'ensemble des traitements en mémoire centrale. Nous constatons que **tous, sans exception, ont pu réaliser le traitement demandé**.

Certains sont plus ou moins rapides que les autres, occupent plus ou moins d'espace en mémoire centrale. A souligner également, même si ce n'était pas le sujet principal dans ce didacticiel,

[4] Ah ! Les discussions de fin de repas entre informaticiens sur les mérites respectifs des langages de programmation et des outils de développement….

certains présentent les résultats de manière plus avenante que les autres. Après, c'est aux utilisateurs de faire leur choix selon leurs priorités.

Enfin, ce comparatif donne une vue très parcellaire du traitement des gros volumes. Pour être exhaustif, il faudrait analyser le comportement des logiciels dans d'autres configurations : que se passe-t-il lorsque le nombre de descripteurs candidats commence à augmenter, plusieurs centaines par exemple ; comment évoluent le temps de calcul et la gestion mémoire lorsque des descripteurs discrets sont présents, en proportion plus ou moins importante ; que se passe-t-il lorsque les éventuels descripteurs discrets comportent un nombre élevé de modalités et que nous devons réaliser des regroupements via des algorithmes type CHAID ; etc.

Tout un chacun peut reproduire l'expérimentation décrite dans ce didacticiel. On peut aussi l'orienter à sa guise en utilisant des données présentant des caractéristiques qui sont au centre de nos préoccupations. Finalement, c'est ce qui importe.

5 Update – 11 décembre 2010

La première version de ce didacticiel date de septembre 2008, elle décrit le comportement de Tanagra version 1.4.27 sortie le 22 aout 2008. Depuis, ma machine de développement a changé, je travaille avec un Quad Core Q9400 à 2.66 Ghz tournant sous Windows Vista ; Tanagra lui-même a changé, nous en sommes à ce jour (11 décembre 2010) à la version 1.4.37 ; et Sipina a lui aussi été modifié (version 3.5 à ce jour), avec l'introduction du multithreading pour certaines techniques d'induction d'arbres.

Dans cette section, nous rééditons notre expérimentation. Nous essayons de cerner les améliorations (s'il y a) introduites dans ces nouvelles versions. Nous nous focaliserons uniquement sur les temps d'exécution de **Tanagra 1.4.37** et **Sipina 3.5**.

Nous avons obtenu les résultats suivants pour un arbre dont la taille des feuilles ne doit pas être inférieure à 1000 observations).

Logiciel - Méthode	Importation des données (sec.)	Construction de l'arbre (sec.)
Tanagra – C4.5	**4.8**	**22**
Sipina – C4.5	**15.8**	**50**
Sipina – CHAID (multithreading, 4 threads)	**-**	**24.7**

Il faut distinguer ce qui est du à la machine, et ce qui revient à l'amélioration de l'implémentation. On considérera (peut être à tort ?) que le système d'exploitation pèse peu dans les temps de traitement.

Le temps de construction de l'arbre C4.5 de Tanagra a été amélioré d'un facteur 1.5 (22 secondes vs. 33 secondes). Clairement, ce gain est imputable au processeur puisque l'implémentation de la technique n'a pas été modifiée depuis la précédente expérimentation.

Lorsque l'on passe au C4.5 de Sipina, la durée est réduite d'un facteur de 2.44 (50 secondes vs. 122 secondes). L'amélioration est plus forte que pour Tanagra. Ce n'est pas étonnant après coup. En travaillant récemment sur le code de Sipina pour introduire le multithreading, je me suis rendu compte de plusieurs incongruités. Je me suis attaché à nettoyer tout ça pour réduire les fuites de mémoire et pour simplifier certaines tâches. Cela se traduit par un gain assez conséquent.

Comme nous l'avions souligné précédemment, avec la même méthode C4.5, nous obtenons exactement le même arbre sous Sipina et Tanagra (heureusement !). L'écart entre le temps d'exécution est en grande partie attribuable au traitement des variables continues : elles sont pré-triées une fois pour toutes dans Tanagra, la recherche des points de discrétisation sur les nœuds est très rapide du coup, la contrepartie est que nous devons conserver en mémoire un index des valeurs triées ; pour Sipina, le tri et la recherche sont réitérés à chaque nœud, cela permet de réduire l'occupation mémoire sur les grandes bases sachant que les fonctionnalités interactives du logiciel sont déjà par ailleurs consommatrices d'espace mémoire.

Lorsqu'on passe au multithreading avec Sipina, nous constatons que nous rattrapons Tanagra (24.7 secondes). Certes, la comparaison est un peu biaisée puisque nous n'utilisons pas la même méthode. Mais à la vue de la taille de l'arbre, 337 feuilles contre 117 feuilles pour C4.5, et sachant que le post élagage de ce dernier est de toute manière très rapide puisqu'il n'a pas besoin d'accéder aux données, nous pouvons quand même rapprocher les temps de traitement. Clairement, le multithreading permet de mieux exploiter les ressources machines. Dans un avenir plus ou moins proche, il est à prévoir que cette stratégie sera introduite dans Tanagra.

Enfin, dernier aspect qui saute aux yeux dans cette nouvelle expérimentation : la durée d'importation des données a été divisée (à peu près) par 2 pour les deux logiciels. Les implémentations n'ont pourtant pas été modifiées. La puissance du processeur joue un rôle important ici, c'est indéniable. Il faut aussi y voir l'impact de la bibliothèque FASTMM introduite début 2009 (à partir de la version 1.4.30 pour Tanagra), particulièrement efficace pour les (ré) allocations mémoires répétées lors du chargement des données.