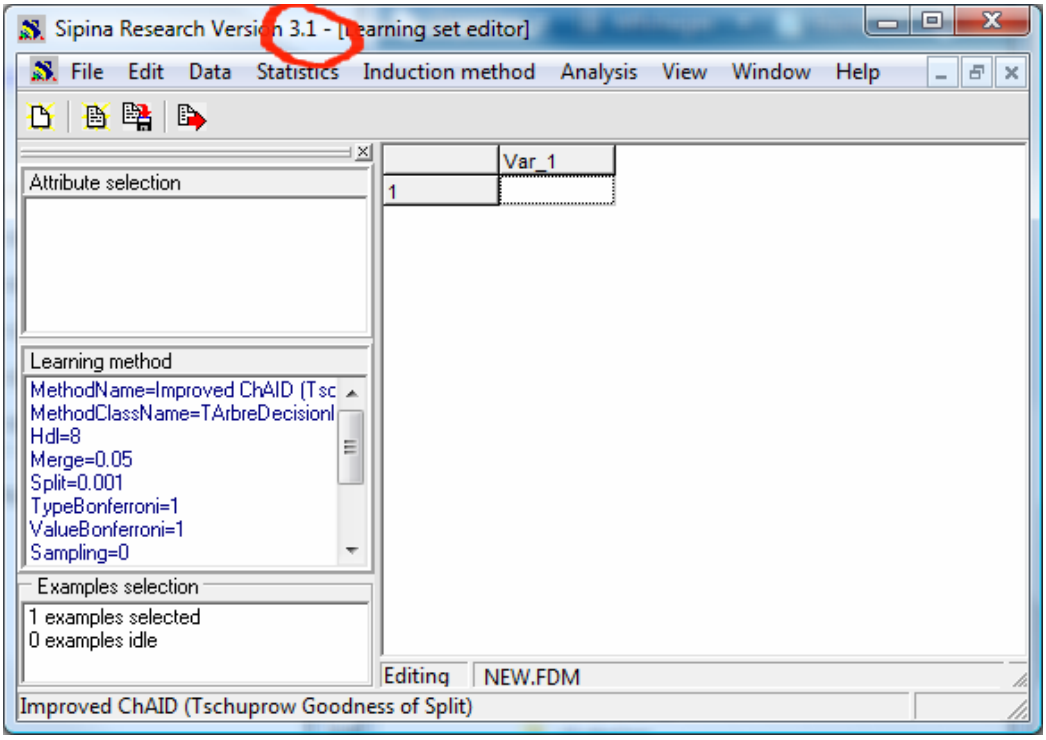
1 Introduction

Stratégies d'échantillonnage pour l'induction des arbres de décision dans SIPINA.

Attention, pour reproduire pleinement les opérations décrites dans ce didacticiel, assurez vous de disposer de la version 3.1 de la version recherche de Sipina[1]. Vous pouvez vérifier cela en observant le numéro de version dans la barre de titre du logiciel lorsqu'il est démarré.



Lors de l'induction d'un arbre de décision, l'algorithme doit détecter la meilleure variable de segmentation pour chaque nœud que l'on souhaite partitionner. L'opération peut prendre du temps si le nombre d'observations est très élevé. Ceci d'autant plus que les variables candidates sont continues, il faut trouver la borne de discrétisation optimale.

Le logiciel SIPINA introduit une **option d'échantillonnage local** dans tous les algorithmes d'induction d'arbres qu'il propose. L'idée est la suivante : sur chaque sommet, plutôt que de travailler sur la totalité des observations présentes pour choisir la variable de segmentation, il réalise les opérations sur un échantillon. Bien entendu, lorsque le nombre d'observations disponibles sur le sommet est plus faible que la taille d'échantillon demandée, il n'y a plus lieu de procéder à un échantillonnage, Sipina utilise toutes les observations. Cela arrive dans les parties bases de l'arbre lorsqu'il est particulièrement profond.

[1] La page de téléchargement du logiciel est http://eric.univ-lyon2.fr/~ricco/sipina_download.html ; chargez et installez la SIPINA RESEARCH VERSION.

Nous avions présenté cette approche dans un ancien article (Chauchat et Rakotomalala, IFCS-2000)[2]. Je l'ai bien entendu implémentée pour réaliser les expérimentations. Puis, comme beaucoup de choses dans SIPINA, je n'ai jamais pensé à la documenter. J'imagine qu'il est temps de le faire.

Nous constaterons dans ce didacticiel que travailler sur un échantillon dans les nœuds permet de réduire considérablement le temps de calculs tout en préservant les performances en classement. Nous n'avons pas la garantie en revanche, à cause des fluctuations d'échantillonnage, d'obtenir exactement le même arbre qu'avec le travail sur la totalité des observations.

Mais est-ce vraiment un problème ? Nous savons que les arbres sont par nature très instables. Changer de base d'apprentissage peut nous conduire à un modèle (visuellement) très différent c.-à-d. les variables sélectionnées ne sont pas les mêmes. Cela ne veut pas dire pour autant que les modèles classent différemment les individus en généralisation. Nous avons le même phénomène lorsque nous travaillons par échantillonnage dans les nœuds.

Nous passerons par les étapes suivantes pour montrer l'intérêt de l'échantillonnage dans SIPINA :

- Dans un premier temps, nous construirons l'arbre avec la totalité des données. Nous mesurerons le temps de calcul et le taux d'erreur en généralisation.
- Puis nous introduisons la nouvelle stratégie, avec une taille d'échantillon égale à n = 5000. Nous comparerons les performances.
- L'effectif n = 5000 ci-dessus est en réalité excessivement prudent. La taille d'échantillon « suffisante » dépend de nombreux facteurs (complexité du concept, densité des points dans l'espace de représentation, etc.), nous verrons que concernant le fichier utilisé nous pouvons le diminuer jusqu'à la centaine d'observations sans dégrader les performances en classement.
- Nous comparons enfin les temps de traitement avec ceux de Tanagra[3] où une autre stratégie a été introduite pour accélérer l'appréhension des variables continues que nous devons discrétiser sur chaque nœud.

2 Données

Nous utilisons les fameux WAVEFORM de Breiman et al. (1984)[4]. La variable à prédire présente 3 modalités, nous disposons de 21 descripteurs continus. Nous avons généré 2.000.000 observations. Nous en utiliserons la moitié pour construire les modèles, le reste servira à mesurer les performances en classement. Le fichier est au format ARFF (Weka)[5].

Nous avions déjà utilisé une version à 500.000 observations pour comparer les performances de

[2] Voir http://sipina.over-blog.fr/10-categorie-10370111.html
ou http://tutoriels-data-mining.blogspot.com/2008/03/echantillonnage-dans-les-arbres-de.html

[3] http://eric.univ-lyon2.fr/~ricco/tanagra/fr/tanagra.html

[4] http://archive.ics.uci.edu/ml/datasets/Waveform+Database+Generator+(Version+1)

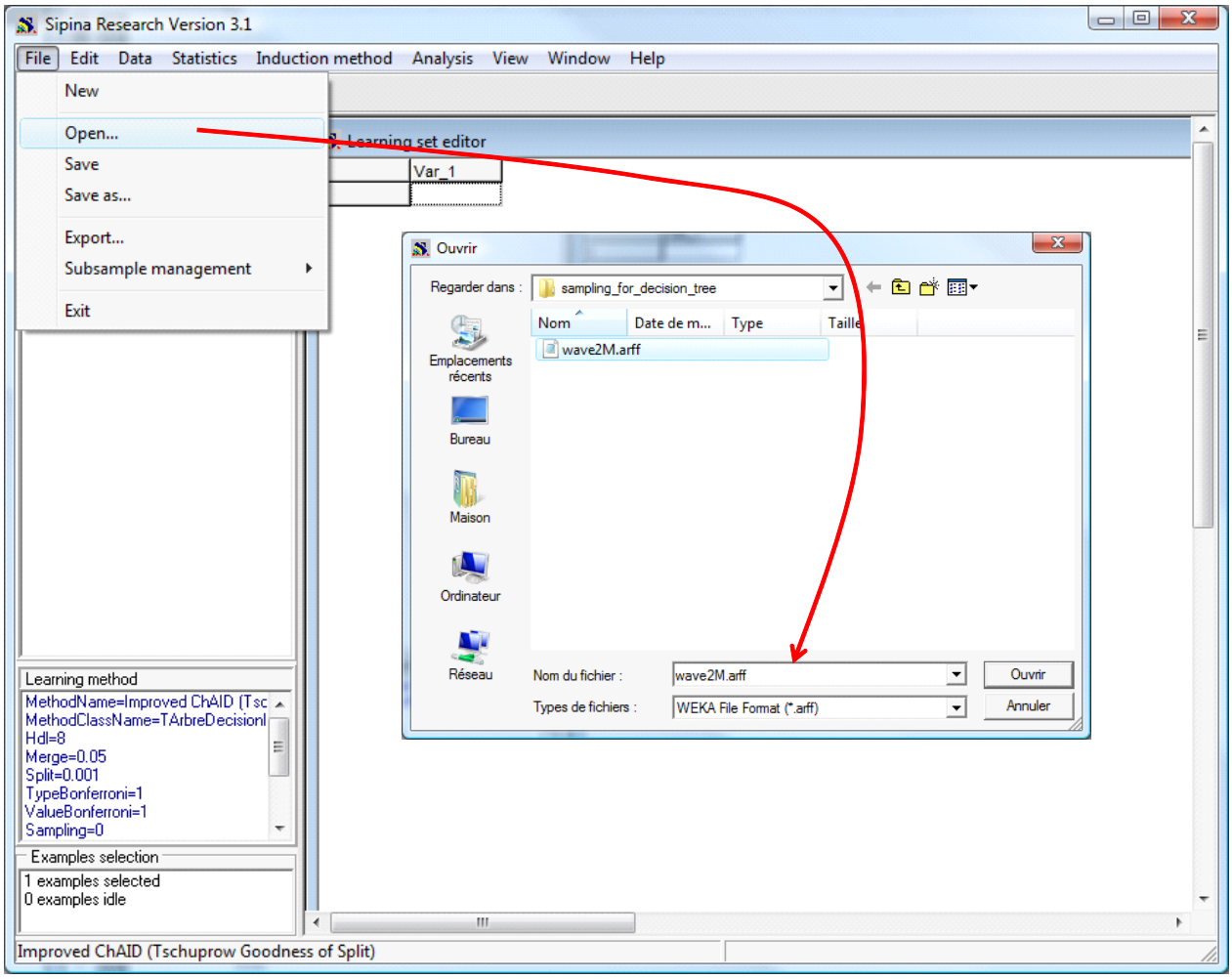
[5] http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/wave2M.zip

plusieurs logiciels libres lors de l'induction des arbres de décision. Il est intéressant de voir le comportement de Sipina et Tanagra lorsque la base comporte, toutes chose égales par ailleurs, 4 fois plus d'observations[6].

3 Traitement avec SIPINA

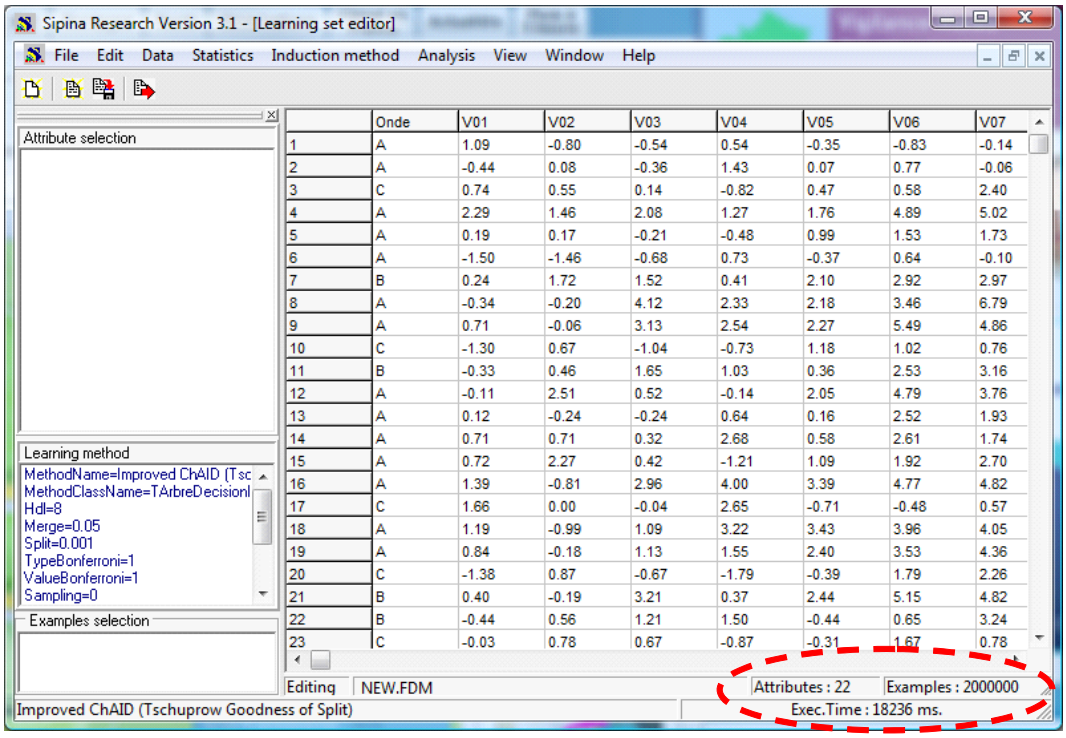
3.1 Préparation pour le traitement

Après avoir démarré SIPINA, nous actionnons le menu FILE / OPEN et choisissons le format WEKA (ARFF). Nous chargeons le fichier WAVE2M.ARFF. Même si tout semble figé. Il ne faut pas s'en inquiéter, le chargement est en cours.



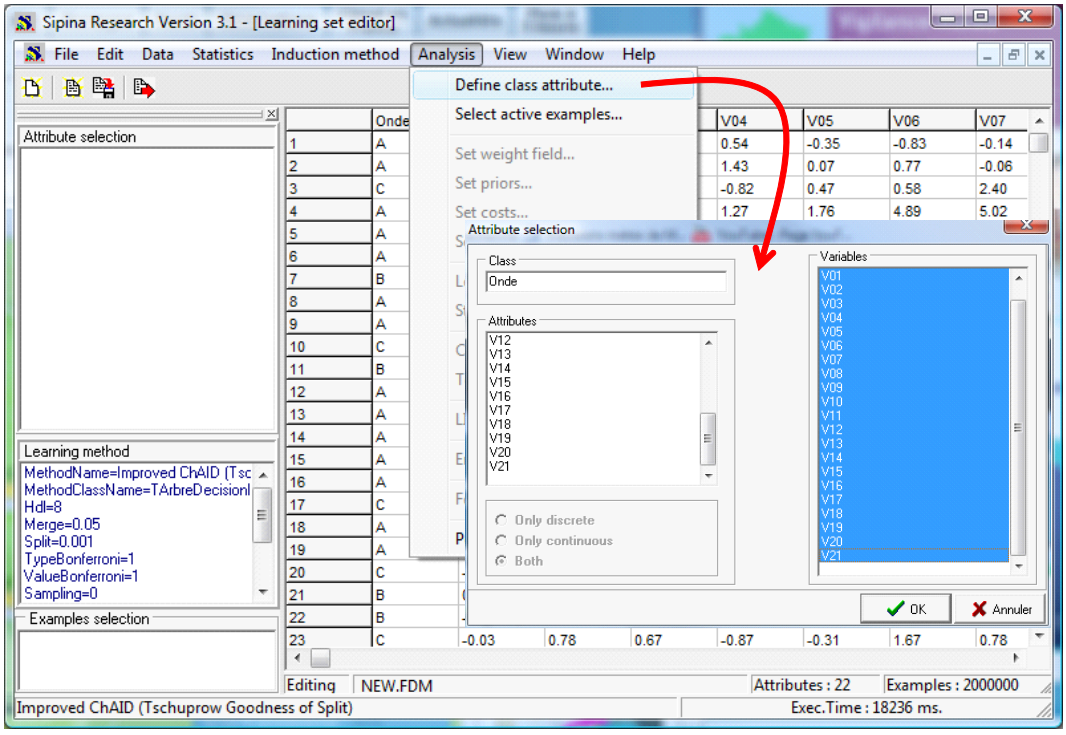
Après 18 secondes, les valeurs apparaissent dans la grille de données. Nous devons avoir 22 colonnes et 2.000.000 observations.

[6] http://tutoriels-data-mining.blogspot.com/2008/09/traitement-de-gros-volumes-comparaison.html ; attention, la méthode d'induction d'arbres utilisée ne sont pas les mêmes, les temps de traitement ne sont pas directement comparables.



Une fois pour toutes, nous allons définir le rôle des variables et subdiviser les observations en ensemble d'apprentissage et ensemble test.

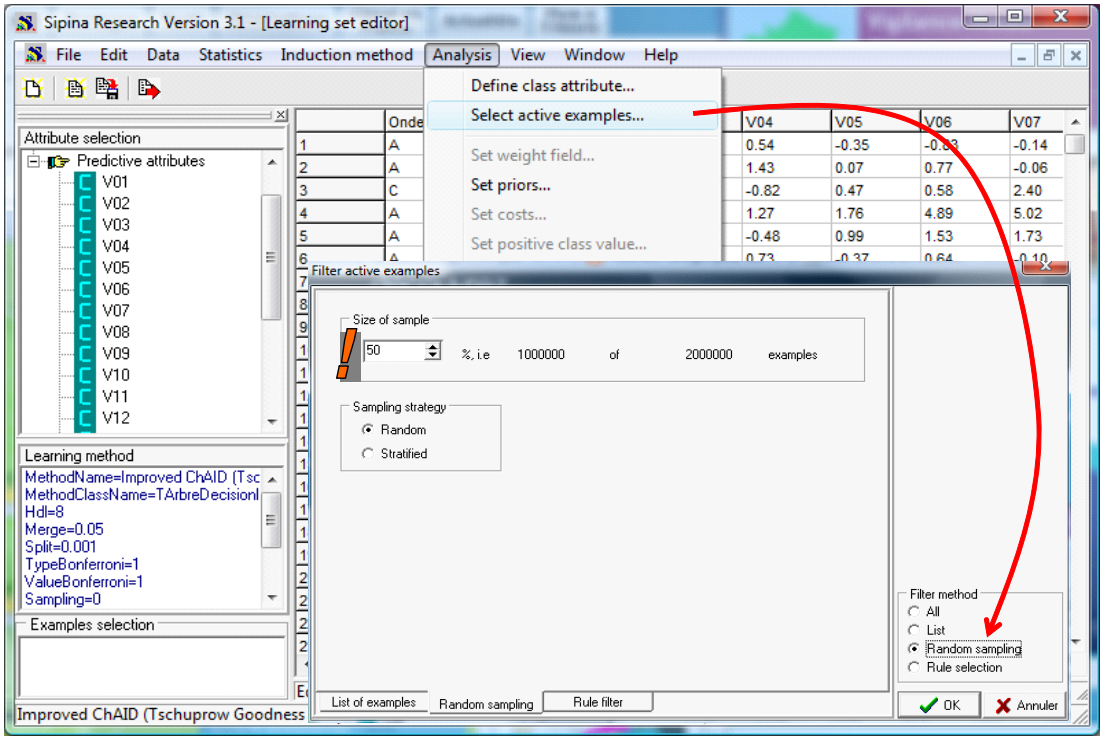
Pour définir le statut des variables, nous actionnons le menu ANALSYS / DEFINE CLASS ATTRIBUTES. Par glisser déposer, nous attribuons ONDE à CLASS, les autres variables (V1 à V21) à ATTRIBUTES.



La sélection apparaît dans la partie gauche de la fenêtre principale.

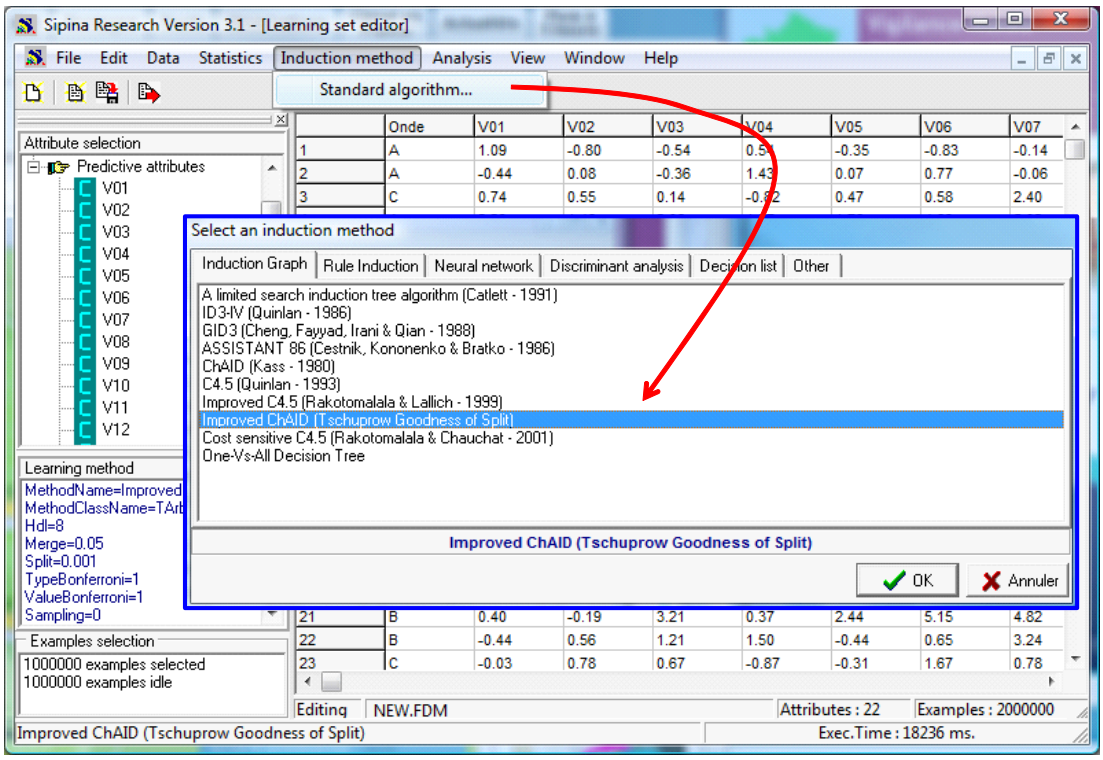
Nous partitionnons les observations en apprentissage et test en actionnant le menu ANALYSIS / SELECT

ACTIVE EXAMPLES. Nous choisissons une subdivision aléatoire RANDOM SAMPLING 50/50.

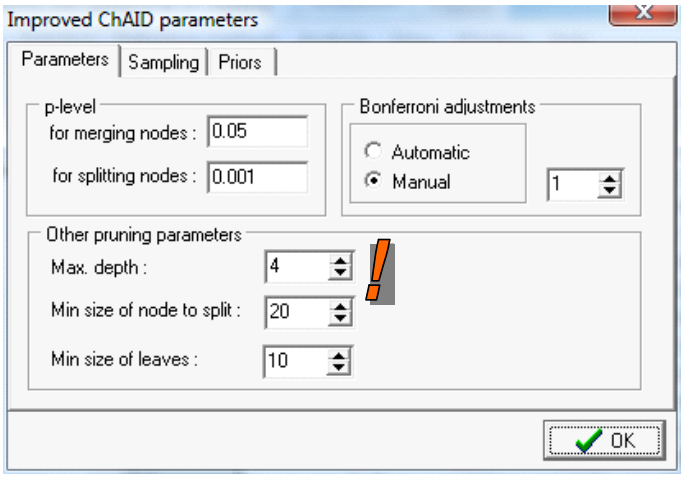


3.2 Apprentissage classique

Choix de la méthode et paramétrage. Pour spécifier la méthode d'apprentissage, nous actionnons le menu INDUCTION METHOD / STANDARD ALGORITHM.

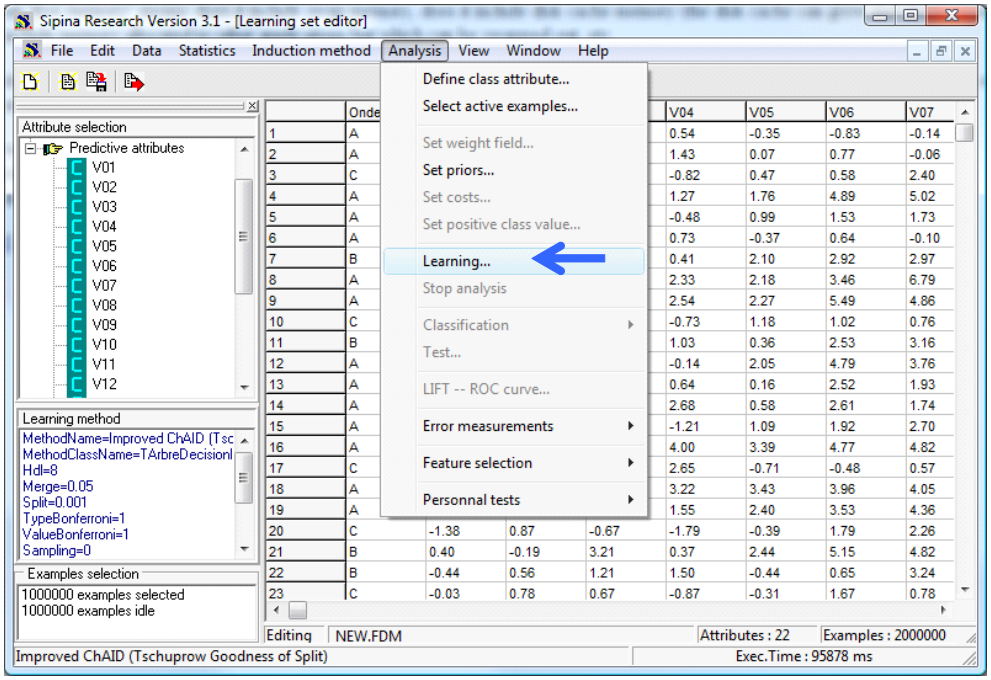


Dans la boîte de dialogue qui apparaît, nous choisissons la méthode IMPROVED CHAID. En cliquant sur OK, nous accédons aux paramètres de la méthode.

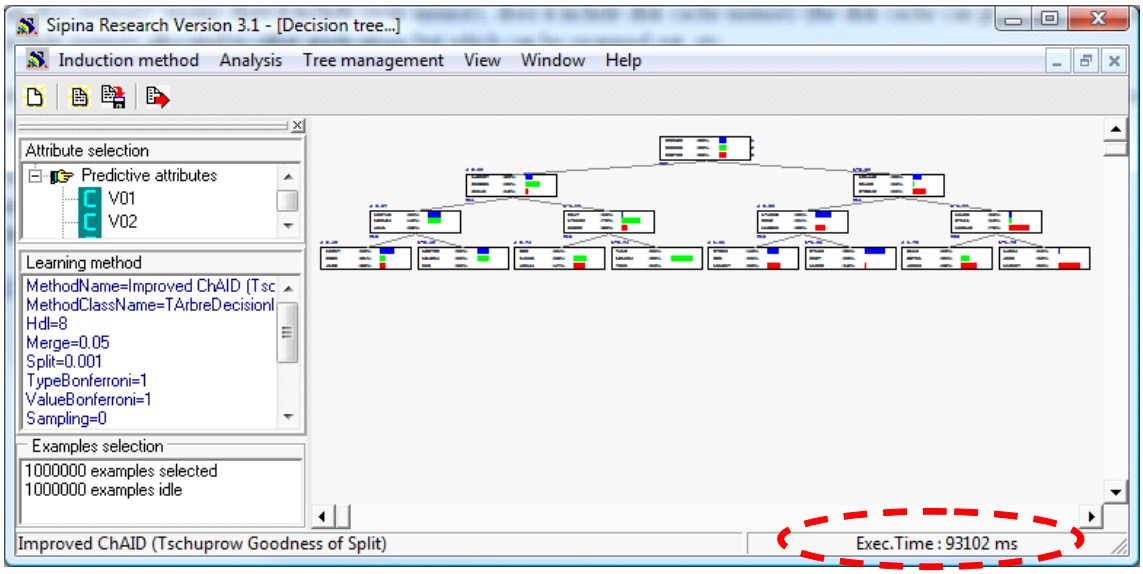


Nous ne modifions rien pour l'instant. Nous nous contentons de noter que l'arbre produit comportera 4 niveaux au maximum (MAX DEPTH = 4). Ce paramètre n'a aucune justification théorique. Il permet simplement de limiter la taille de l'arbre. En effet, sur les grandes bases, obtenir un modèle avec des centaines de feuilles est assez courant. Il est évident qu'une appréciation qualitative des règles et l'interprétation des résultats sont impossibles dans ce cas.

Apprentissage. Il ne reste plus qu'à actionner le menu ANALYSIS / LEARNING pour lancer les traitements.



Après 93 secondes, nous avons un arbre avec 8 feuilles et, effectivement, 4 niveaux (la racine compte pour le niveau n°1).



Détaillons l'arbre de décision (Figure 1) : la variable de segmentation à la racine est V07 ; au second niveau, les deux sommets sont segmentés avec la même variable[7] V11, mais avec des bornes de discrétisation différentes (3.57 à gauche, 3.36 à droite) ; au 3ème niveau, les variables de segmentation sont successivement, de gauche à droite, (V15, V15, V16 et V06).

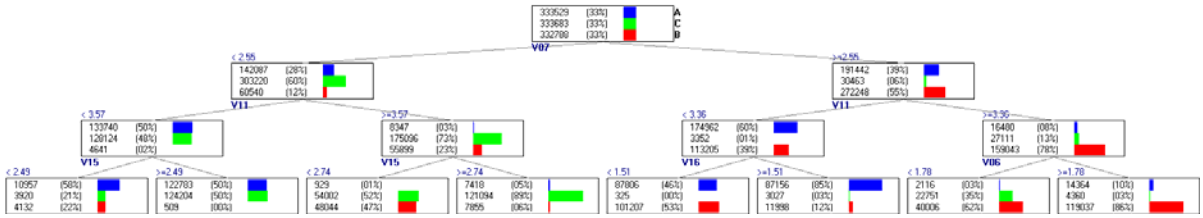


Figure 1 - Arbre de décision, sans échantillonnage sur les noeuds

Test. Pour évaluer l'arbre sur l'échantillon test, nous sélectionnons le menu ANALYSIS / TEST. Nous pouvons élaborer la matrice de confusion sur l'échantillon d'apprentissage (les individus sélectionnés) ou sur l'échantillon test (les autres). Nous choisissons la seconde option (Figure 2).

Nous obtenons la matrice de confusion avec un taux d'erreur de 34% (Figure 3). Nous garderons cette valeur à l'esprit, elle nous servira de référence pour apprécier les stratégies d'échantillonnage dans les arbres.

[7] Ce choix repose sur les calculs, il ne s'agit pas d'une contrainte imposée lors de la construction de l'arbre.

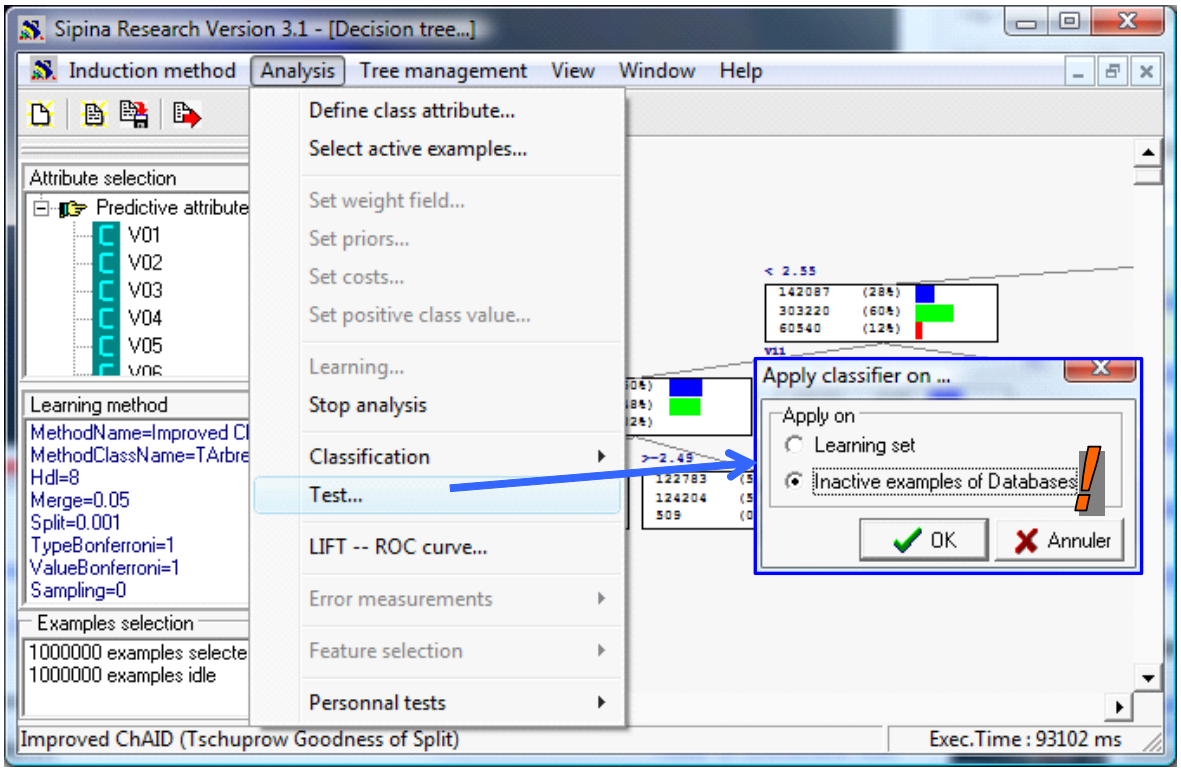


Figure 2 - Evaluation sur la partie "test" des données

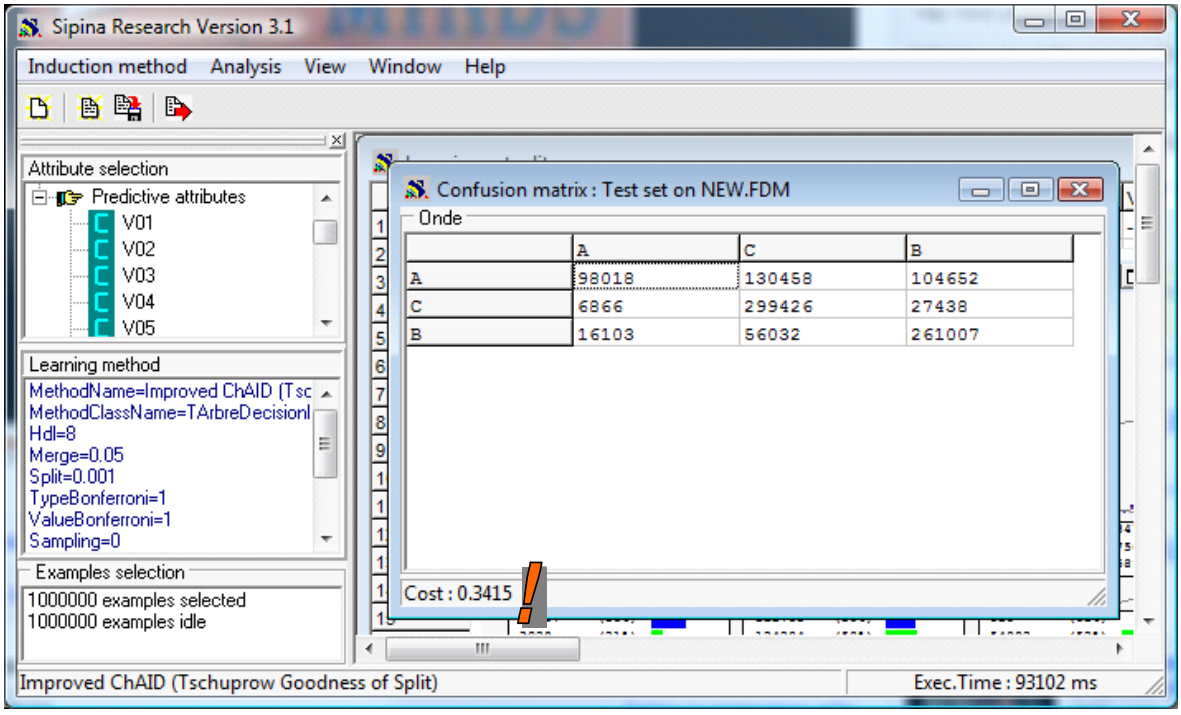


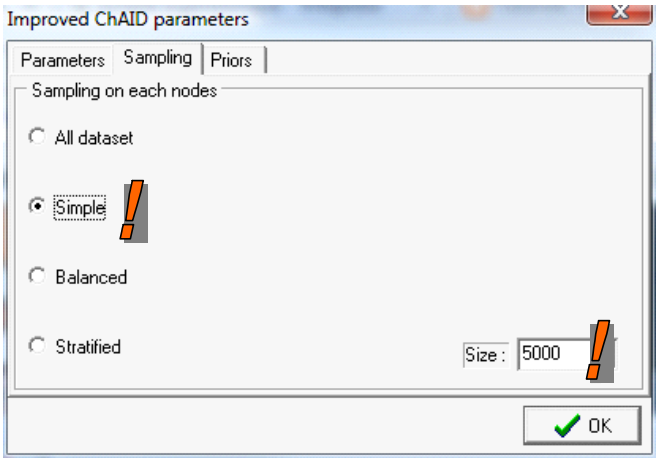
Figure 3 - Matrice de confusion et taux d'erreur en test

3.3 Stratégie d'échantillonnage pour les arbres de décision

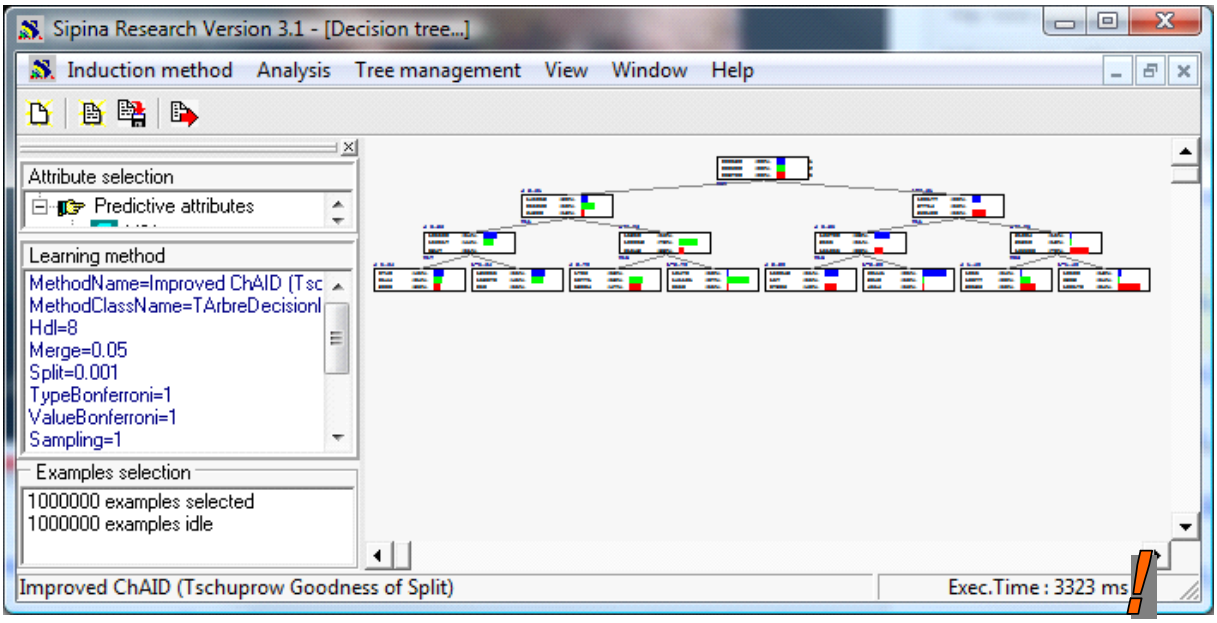
Stoppons tout d'abord l'analyse en cours. Le plus simple est d'actionner le menu WINDOW / CLOSE ALL.

Nous souhaitons maintenant introduire la stratégie d'échantillonnage local lors de l'induction. Pour ce faire, nous cliquons de nouveau sur le menu INDUCTION METHOD / STANDARD ALGORITHM puis, tout comme précédemment, nous sélectionnons la même méthode IMPROVED CHAID.

Tout se joue dans la boîte de paramétrage. Nous activons l'onglet SAMPLING. Au lieu d'exploiter toutes les observations disponibles sur chaque nœud, nous procédons à un échantillonnage SIMPLE, avec n = 5000.



Nous validons. Nous pouvons démarrer une nouvelle analyse avec ANALYSIS / LEARNING. Nous obtenons un nouvel arbre au bout de 3 secondes. Le temps de calcul est sacrément réduit.



SIPINA affiche la distribution des classes sur la totalité des données. Mais il procède bien par

échantillonnage pour les calculs comme peut l'attester la réduction du temps de traitement.

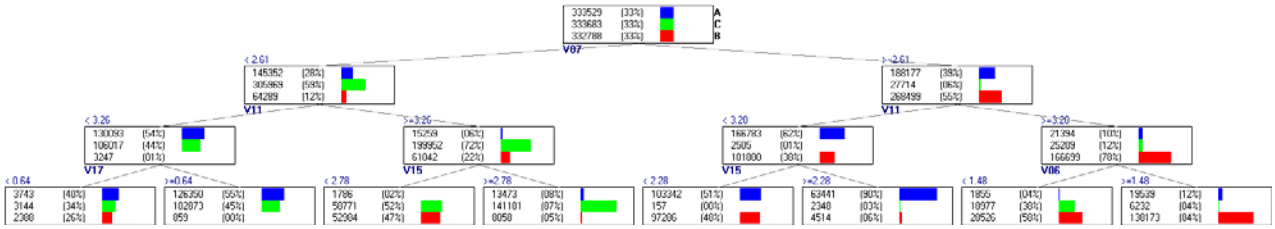


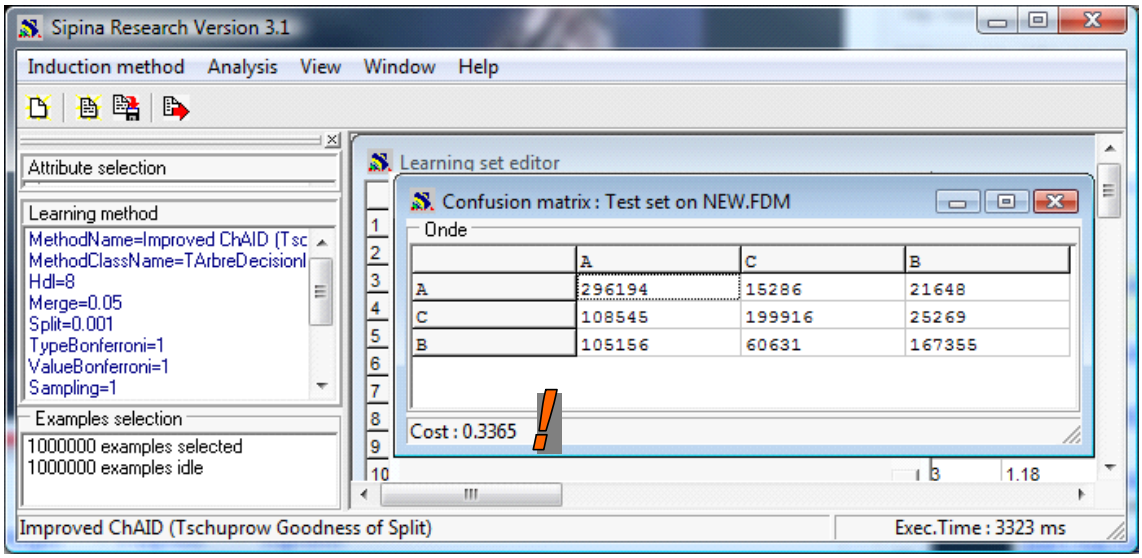
Figure 4 - Arbre avec stratégie d'échantillonnage (n = 5000)

L'arbre obtenu (Figure 4) est un peu différent par rapport au précédent (Figure 1). Sur les deux premiers niveaux, les segmentations sont quasi-identiques. A partir du 3ème niveau, il y a des disparités.

Est-ce que pour autant les modèles sont très différents ?

Il y a plusieurs manières de le savoir. Une technique possible serait de croiser les feuilles des deux arbres pour vérifier si des individus situés dans une même feuille dans le premier arbre le sont également dans le second. Une autre technique, que nous utilisons ici, consiste simplement à mesurer et comparer les performances en classement.

Nous appliquons l'arbre sur les observations tests en actionnant le menu ANALYSIS / TEST, toujours sur les individus non sélectionnés (INACTIVE EXAMPLES OF DATABASE).



Nous obtenons une nouvelle matrice de confusion et un taux d'erreur de 34%[8]. Par rapport au traitement classique, l'approche par échantillonnage sur les nœuds a permis de réduire le temps de calcul (3 secondes vs. 93 secondes) sans modification significative du taux d'erreur en généralisation

[8] Un tout petit peu plus faible même, mais n'allons pas tirer des conclusions définitives pour des écarts à la troisième décimale.

(34% vs. 34%). L'opération s'avère très avantageuse.

3.4 Réduire la taille de l'échantillon

La vraie difficulté réside dans la détermination de la « bonne » taille n de l'échantillon. Plusieurs éléments peuvent peser. Nous optons pour une solution très prudente dans SIPINA en fixant n = 5000 par défaut. Lors de nos expérimentations sur la base WAVE, nous nous sommes rendus compte que quelques centaines d'observations sur chaque nœud suffisent pour déterminer la bonne variable de segmentation.

Concernant notre configuration, nous avons fait varier n sur les valeurs (100, 200, 400, 800, 1600, 3200, 6400), nous avons mesuré à la fois le temps de calcul (Figure 5) et le taux d'erreur en généralisation (Figure 6).

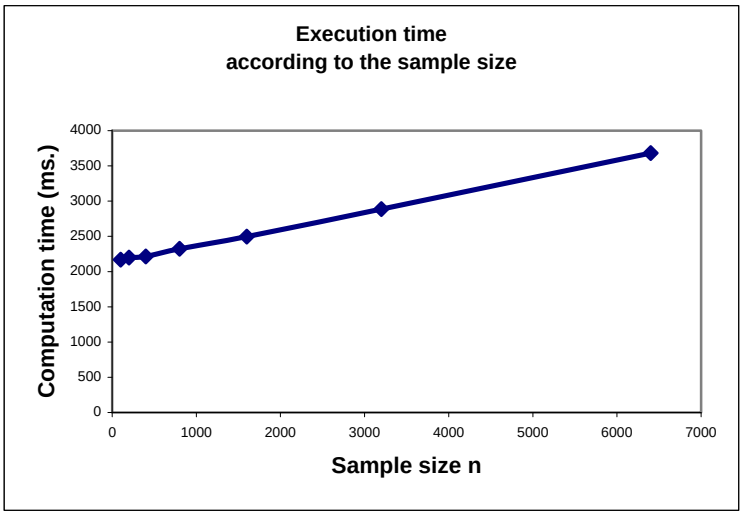


Figure 5 - Temps de calcul selon la taille d'échantillon dans les noeuds

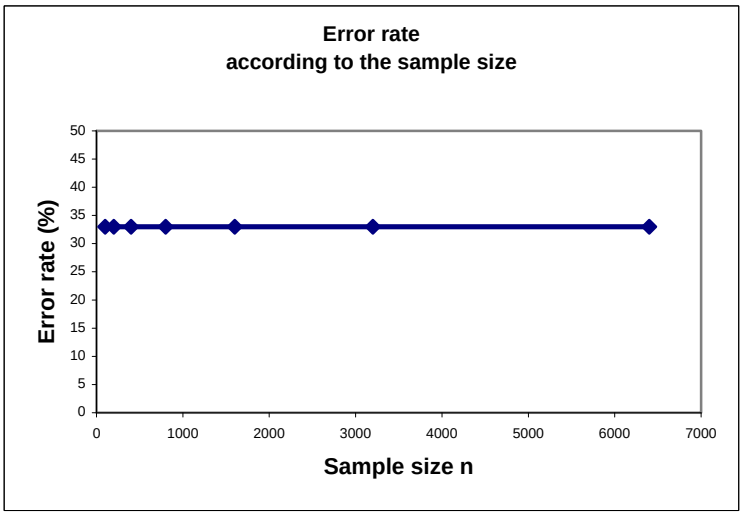


Figure 6 - Taux d'erreur selon la taille d'échantillon dans les noeuds

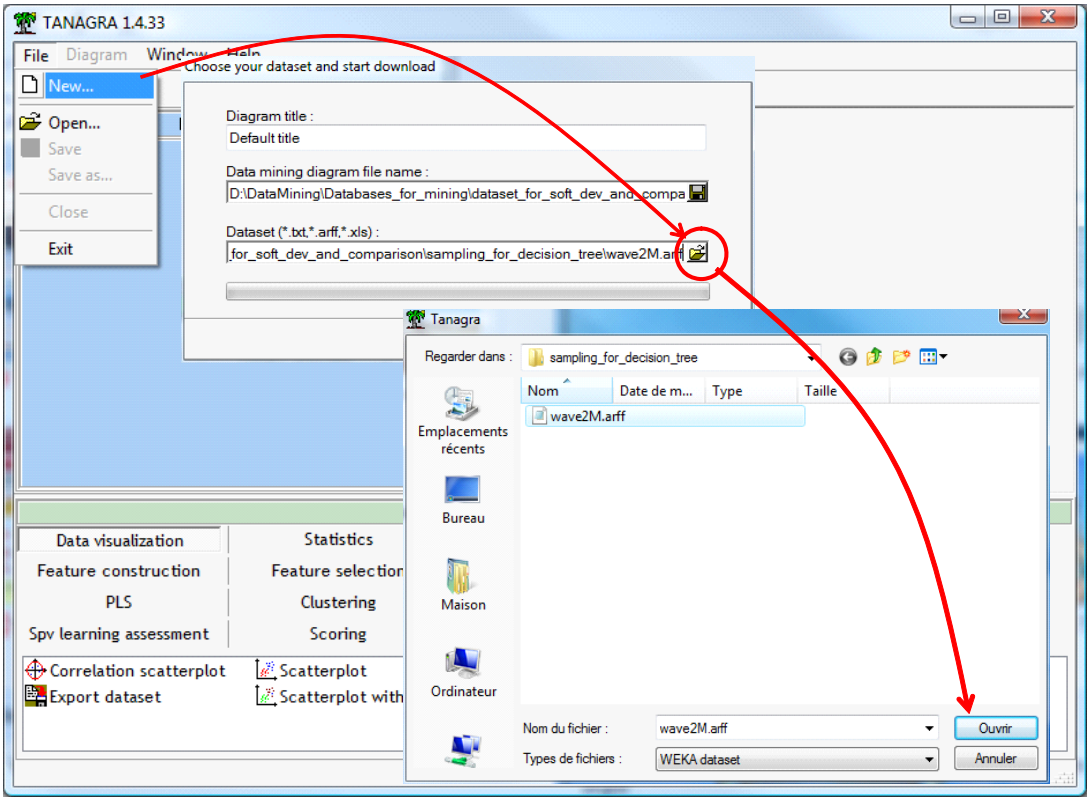
Comme nous pouvons le constater, nous obtenons très rapidement de bons résultats (en termes de taux d'erreur) avec une taille d'échantillon relativement faible (dès n = 100). Le temps de calcul, lui, croit bien évidemment avec n.

4 Traitement avec TANAGRA

Nous avons complètement changé notre fusil d'épaule pour Tanagra. Les descripteurs continus sont triés avant la construction de l'arbre. Un tableau d'index est conservé en mémoire. Pour notre fichier comportant 21 variables, nous avons donc besoin de (21 x 4 x 1.000.000 # 80 Mo) en mémoire pour l'apprentissage. Les raisons de ce revirement sont essentiellement pragmatiques. La RAM est plus abondante sur les ordinateurs récents. Du temps du développement de SIPINA, ma machine disposait de 256 Mo, c'était plutôt élevé pour l'époque ; aujourd'hui, une partie des 4Go dont je dispose n'est même pas utilisée par le système d'exploitation[9]. L'objectif de cette nouvelle stratégie étant d'accélérer les calculs tout en utilisant la totalité des observations, voyons ce qu'il en est sur notre base.

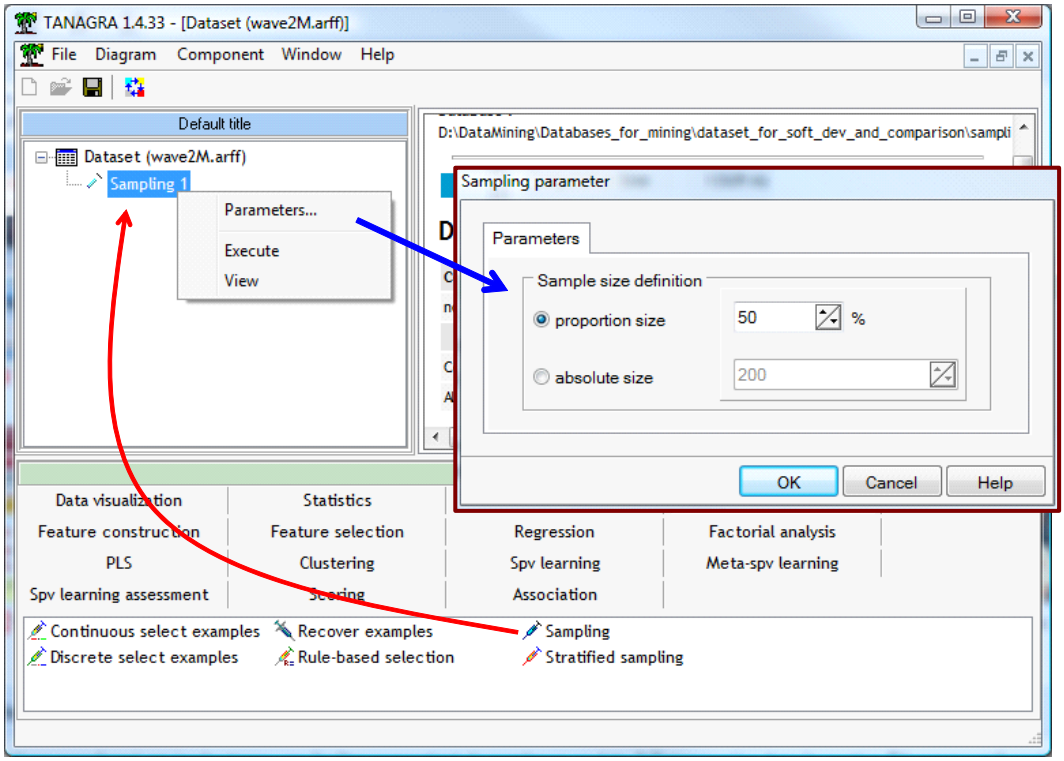
4.1 Importation des données et préparation des calculs

Après avoir démarré Tanagra (**version 1.4.33 ou plus récente**), nous créons un nouveau diagramme avec FILE / NEW. Nous sélectionnons le fichier WAVE2M.ARFF.

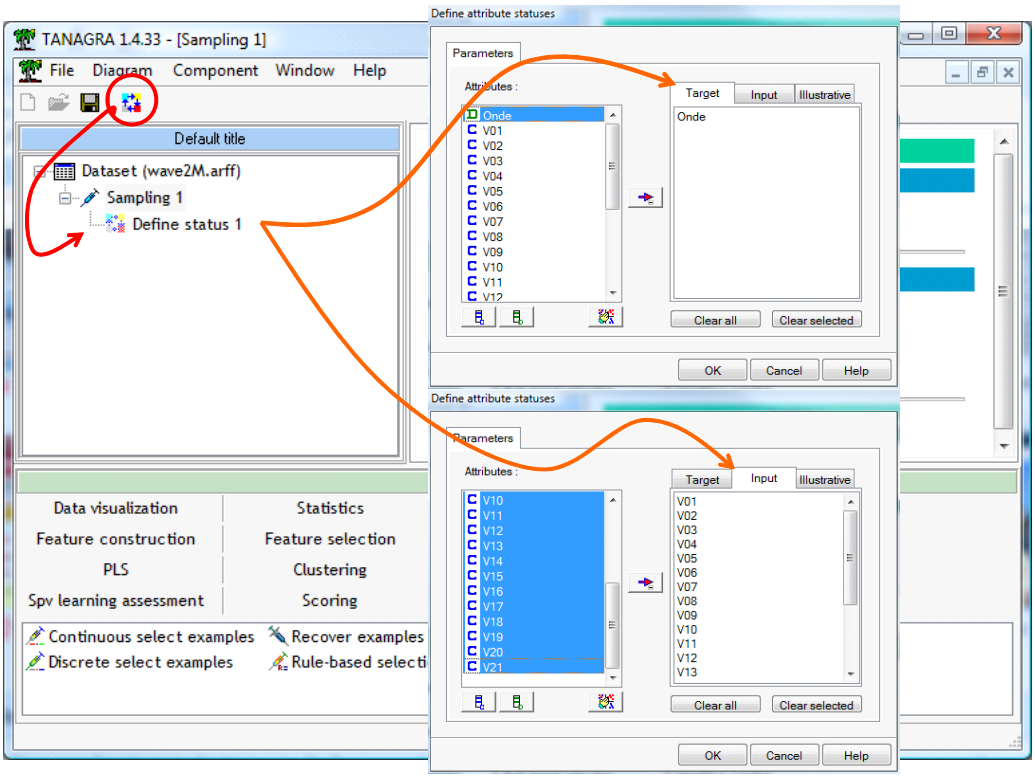


[9] Tanagra intègre une directive de compilation qui permet de revenir sur l'ancien système mis en place dans SIPINA, sans toutefois intégrer le mécanisme d'échantillonnage dans les nœuds.

L'étape suivante consiste à subdiviser les données en ensemble d'apprentissage et ensemble test. Nous utilisons le composant SAMPLING (onglet INSTANCE SELECTION). Nous ouvrons la boîte de paramétrage pour spécifier que 50% des observations seront utilisées pour la construction de l'arbre.

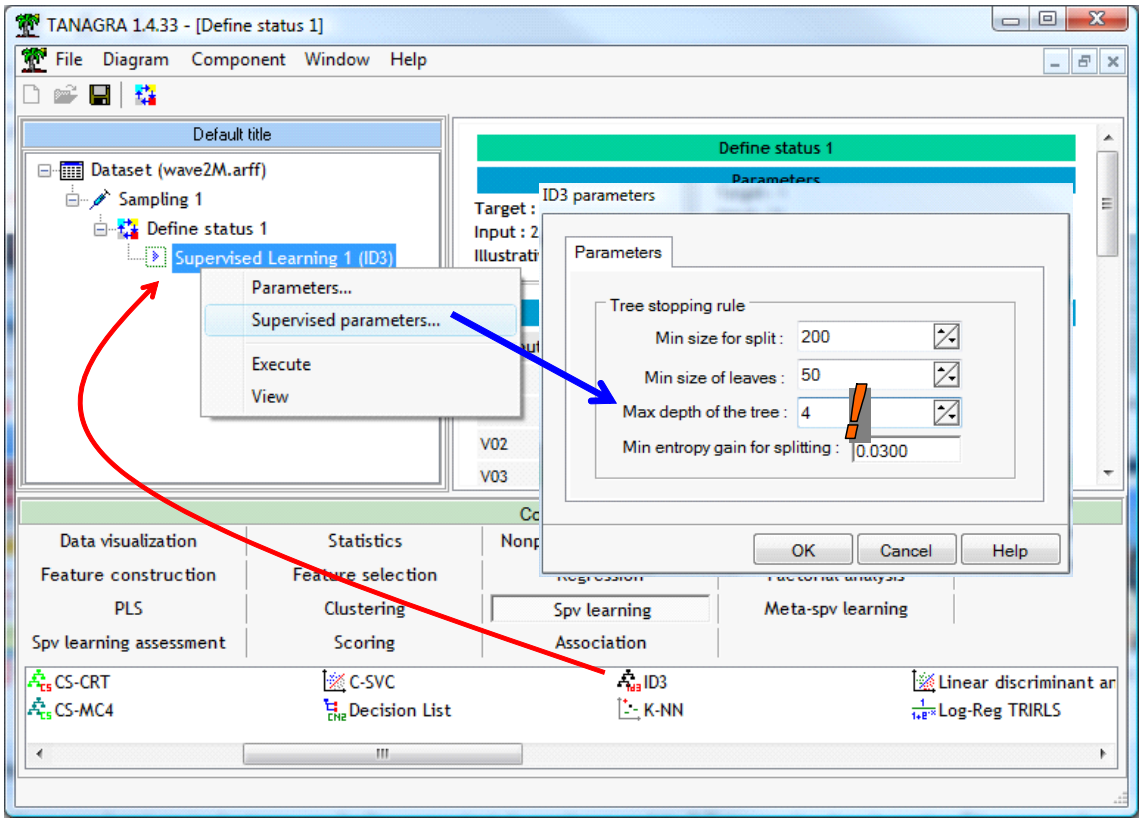


Pour définir le rôle des variables, nous insérons le composant DEFINE STATUS dans le diagramme en utilisant le raccourci dans la barre d'outils, nous plaçons ONDE en TARGET, les autres en INPUT.

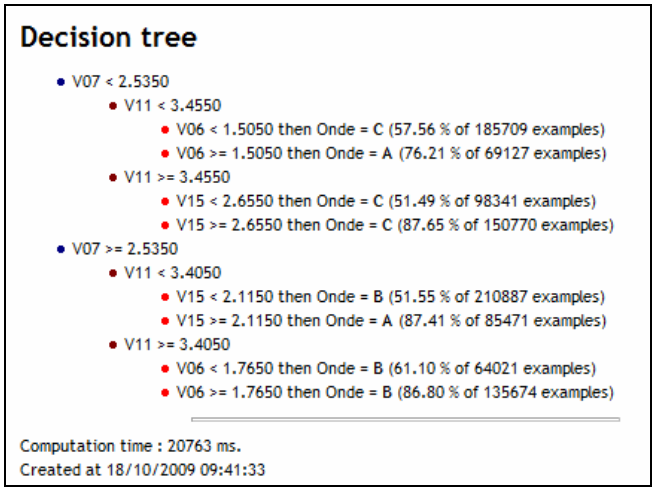


4.2 Apprentissage avec ID3

Nous souhaitons implémenter un arbre de décision. Nous choisissons la méthode ID3 (onglet SPV LEARNING) dans Tanagra car elle intègre un paramétrage similaire à celui du IMPROVED CHAID de Sipina. Nous l'introduisons dans le diagramme et nous actionnons le menu contextuel SUPERVISED PARAMETERS. Comme dans Sipina, nous limitons la profondeur de l'arbre à 4 niveaux (MAX DEPTH OF THE TREE = 4).



Nous validons la modification et nous lançons les calculs en cliquant sur le menu contextuel VIEW.



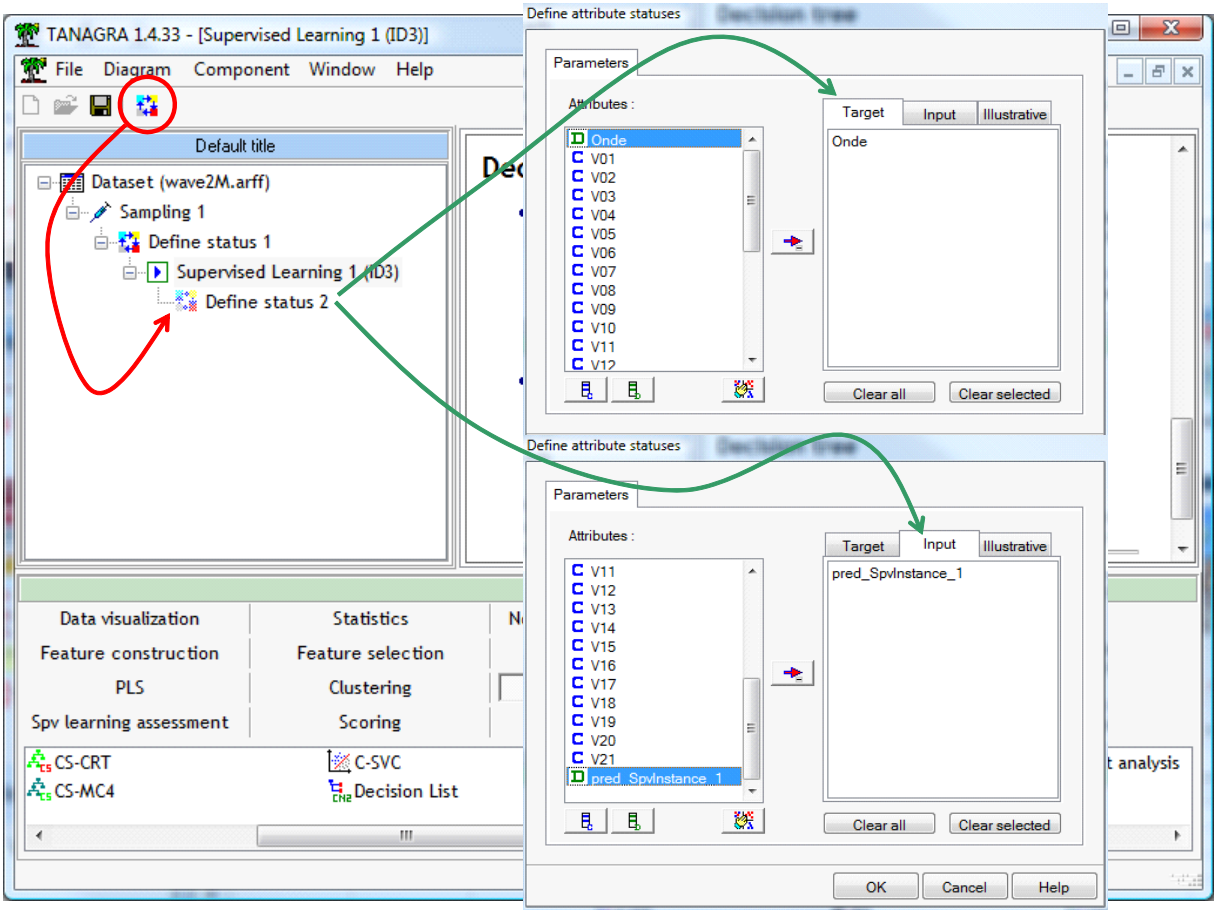
Après 21 secondes, nous obtenons l'arbre. Ainsi, à configuration égale (production d'un arbre à 4

niveaux, avec 7 opérations de segmentation), le calcul préalable des index triés pour chaque variable permet de diviser par 4.42 le temps d'exécution (93 sec. / 21 sec.), au prix d'une occupation mémoire de 80 Mo supplémentaire. Est-ce justifié ? Ne l'est-ce pas ? Souvent, tout est affaire de compromis.

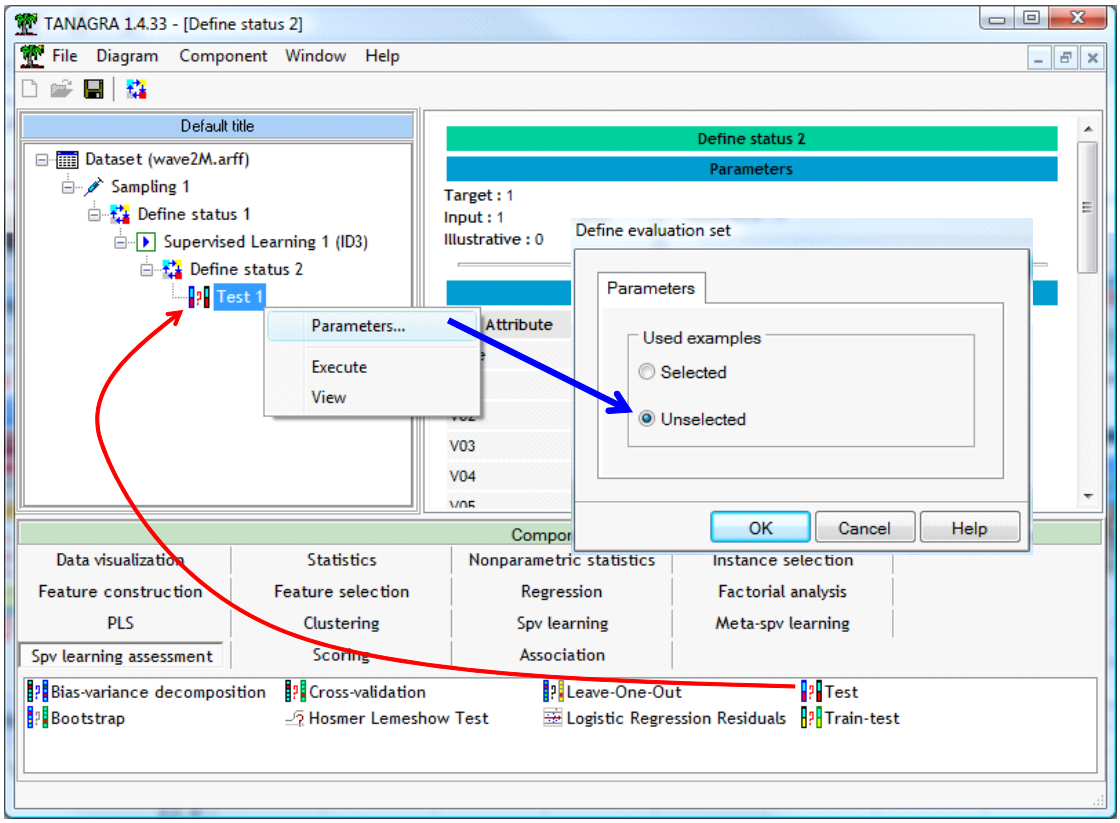
Reste à évaluer les performances en prédiction du modèle.

4.3 Test

Nous introduisons de nouveau DEFINE STATUS, nous plaçons en TARGET la variable à prédire ONDE, en INPUT la nouvelle variable indiquant les prédictions de l'arbre PRED_SPVINSTANCE_1.



Nous utilisons le composant TEST (onglet SPV LEARNING ASSESSMENT) pour calculer la matrice de confusion. Nous actionnons le menu PARAMETERS, nous nous assurons que nous travaillons sur les individus qui ont été préalablement mis de côté (UNSELECTED) c.-à-d. l'échantillon test.



Nous actionnons VIEW. Nous obtenons la matrice de confusion et un taux d'erreur en test de 32%. Nous avons le même ordre de grandeur qu'avec Sipina. Il ne faut pas non plus se focaliser sur cette valeur étant entendu que les deux logiciels n'ont pas utilisé des partitions « apprentissage-test » identiques (les mêmes proportions mais pas les mêmes individus).

Test 1

Parameters

Evaluation set : unselected examples

Results

pred_SpvInstance_1

Error rate	0.3163

Values prediction

Value	Recall	1-Precision
A	0.3846	0.1744
C	0.8682	0.3332
B	0.7987	0.3522

Confusion matrix

	A	C	B	Sum
A	128316	88274	117061	333651
C	16489	289544	27467	333500
B	10609	56396	265844	332849
Sum	155414	434214	410372	1000000

Computation time : 280 ms.
Created at 18/10/2009 10:13:18

5 Conclusion

Comme nous avons pu le constater, travailler sur un échantillon plutôt que sur la totalité des observations dans les nœuds permet de réduire spectaculairement le temps de calcul sans dégrader les performances en classement. C'est une approche qui peut être très profitable lorsque nous avons à analyser de très grandes bases de données.

Nous récapitulons les principaux résultats dans un tableau :

	Temps d'exécution (sec.)	**Taux d'erreur (%)**
Sipina sans échantillonnage	93	34
Tanagra	21	32
Sipina avec échantillonnage (n = 5000)	3	34

Les améliorations introduites dans Tanagra permettent d'accélérer les calculs par rapport à l'implémentation classique de SIPINA (sans échantillonnage). Lorsque nous introduisons l'échantillonnage, nous réduisons encore plus fortement le temps d'exécution (**30 fois !**). Les performances en classement ne sont pas modifiées.

Soyons honnête. Nous nous plaçons dans un cadre particulièrement favorable à la stratégie d'échantillonnage dans ce didacticiel. D'une part, nous produisons un arbre réduit, avec des taux de sondage très faibles dans les sommets ; d'autre part, tous les descripteurs candidats sont continus, nécessitant une discrétisation. Lorsque nous utilisons une méthode d'apprentissage aboutissant à des arbres de grande taille (ex. C4.5) ou lorsque les descripteurs sont majoritairement discrets, les écarts de temps d'exécution sont moins marquants. **SIPINA étant librement accessible en ligne, vous pourrez le charger, l'installer, et voir ce qu'il en est sur vos propres bases**.

Enfin, deux dernières remarques pour conclure. (1) L'idée n'a jamais été explorée, mais l'échantillonnage local, avec remise cette fois-ci, peut être une piste pour équilibrer artificiellement les observations sur les sommets lorsque nous traitons des bases fortement déséquilibrées. On sait que les algorithmes d'induction d'arbres sont en difficulté dans ce cas. (2) Nous nous sommes préoccupés d'échantillonnage local dans ce didacticiel. L'autre alternative est l'échantillonnage global. Nous prélevons une fraction des observations dans la base, que nous utilisons pour construire l'arbre avec les algorithmes usuels, sans jamais revenir sur la totalité des données. La détermination de la taille adéquate de l'échantillon est aussi un problème récurrent dans ce canevas. Des stratégies adaptatives sont souvent mises en avant pour y répondre (« Windowing » avec Quinlan, 1993 ; « Dynamic sampling » avec John, 1996).