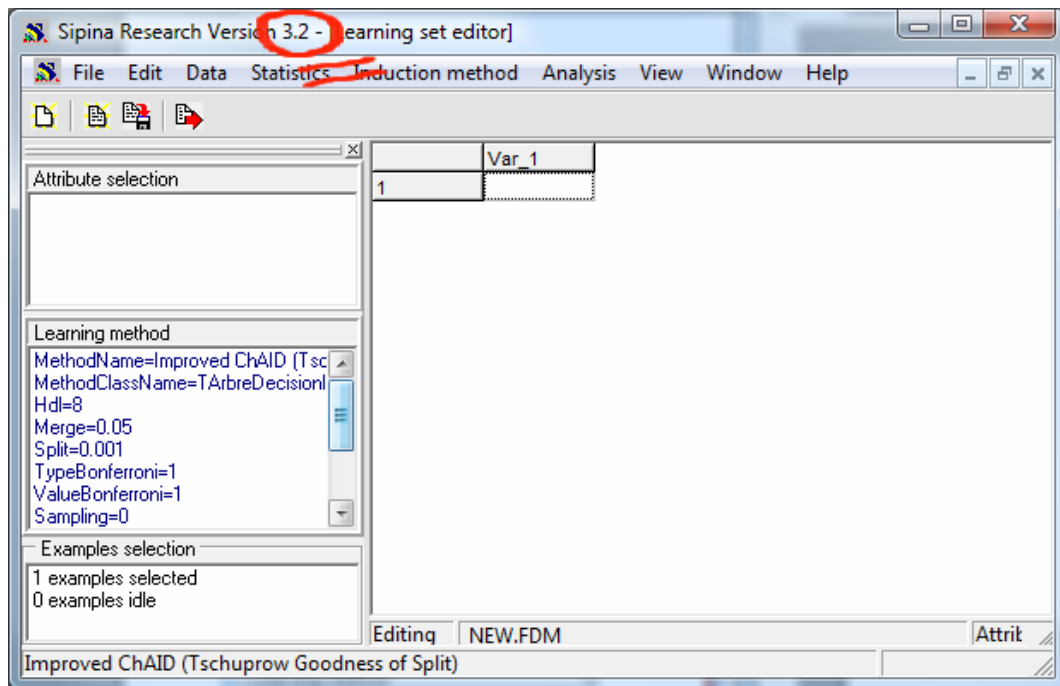


# 1 Introduction

Traitement des grands fichiers (9.634.198 observations et 41 variables) avec SIPINA.

Attention, pour reproduire pleinement les opérations décrites dans ce didacticiel, assurez vous de disposer de la version 3.2 (ou ultérieure) de la version recherche de Sipina<sup>1</sup>. Vous pouvez vérifier cela en observant le numéro de version dans la barre de titre du logiciel lorsqu'il est démarré.



Triturer les très grands fichiers est le fantasme ultime du data miner. On veut pouvoir traiter de très grandes bases dans l'espoir d'y déceler des informations cachées. Malheureusement, rares sont les logiciels libres qui peuvent les appréhender. Tout simplement parce que la quasi-totalité d'entre eux chargent les données en mémoire. Knime semble faire exception (<http://www.knime.org/>). Il sait swapper une partie des données sur le disque<sup>2</sup>. Mais j'avoue ne pas savoir comment exploiter pleinement cet atout (paramétrer ou contrôler l'encombrement mémoire en fonction des données et des algorithmes utilisés par exemple).

Cette rareté n'est guère étonnante. En effet, l'affaire est compliquée. Il ne s'agit pas seulement de copier des informations sur le disque, il faut pouvoir y accéder efficacement compte tenu de la méthode d'apprentissage mise en œuvre. Deux aspects s'entremêlent : (1) comment organiser les données sur le disque ; (2) est-il possible de proposer un système de cache afin d'éviter d'avoir à accéder au disque à chaque fois qu'il faut traiter un individu ou lire la valeur d'une variable.

<sup>1</sup> La page de téléchargement du logiciel est [http://eric.univ-lyon2.fr/~ricco/sipina\\_download.html](http://eric.univ-lyon2.fr/~ricco/sipina_download.html) ; chargez et installez la SIPINA RESEARCH VERSION.

<sup>2</sup> <http://tutoriels-data-mining.blogspot.com/2008/09/traitement-de-gros-volumes-comparaison.html>

Le fonctionnement normal de Sipina est de charger les données en mémoire. Les traitements sont très rapides, le programme a été sécurisé au fil des années. Pourtant, la sobriété est cruciale pour lui. En effet, les fonctionnalités interactives<sup>3</sup> impliquent de calculer et de conserver en mémoire moult informations pour chaque sommet : la liste des variables candidates, les partitions associées, les statistiques descriptives comparatives, etc. De plus, l'utilisateur doit pouvoir demander n'importe quelle opération sur un sommet quelconque : voir le tableau de données, calculer des indicateurs statistiques qui ne sont pas directement proposés, etc. Bref, tout ceci encombre très vite la RAM. Il faut trouver des astuces pour traiter de très grands fichiers confortablement.

Dans ce didacticiel, nous montrons comment exploiter une solution que j'ai naguère implémentée dans Sipina. Elle n'a jamais été valorisée ni documentée. J'avoue l'avoir totalement oubliée jusqu'à ce que je la redécouvre par hasard en préparant le tutoriel sur l'échantillonnage dans les arbres<sup>4</sup>.

Nous montrons qu'il est possible de traiter, en disposant de toutes les fonctionnalités interactives, un fichier comportant **41 variables** et (surtout) **9.634.198 observations** lorsque nous activons cette option.

**Attention** : Toutes les fonctionnalités d'exploration des données sont disponibles lorsque l'option est activée. On s'abstiendra en revanche de faire appel aux fonctionnalités d'édition. On évitera tout particulièrement de supprimer ou d'ajouter des observations dans la grille de données.

Pour apprécier pleinement la solution proposée par Sipina, nous ferons le parallèle avec le comportement des logiciels **Tanagra 1.4.33** et **Knime 2.0.3** face à un tel fichier.

## 2 Données

Nous utilisons une version nettoyée<sup>5</sup> du fichier KDD-CUP 99<sup>6</sup>, nous l'avons dupliquée deux fois (TWICE-KDD-CUP-DISCRETIZED-DESCRIPTORS.TXT). La variable à prédire CLASSE est en dernière position. Tous les descripteurs (40) sont discrets ou ont été discrétisés. Pourvoir charger et afficher un tel fichier dans un logiciel de Data Mining est déjà un défi.

## 3 Traitement en mémoire

Pour montrer qu'il n'est pas possible de manipuler le fichier avec les options usuelles de Sipina, nous allons tenter de le charger comme si de rien n'était.

Nous démarrons Sipina. Nous importons le fichier en actionnant le menu FILE / OPEN. Nous le sélectionnons.

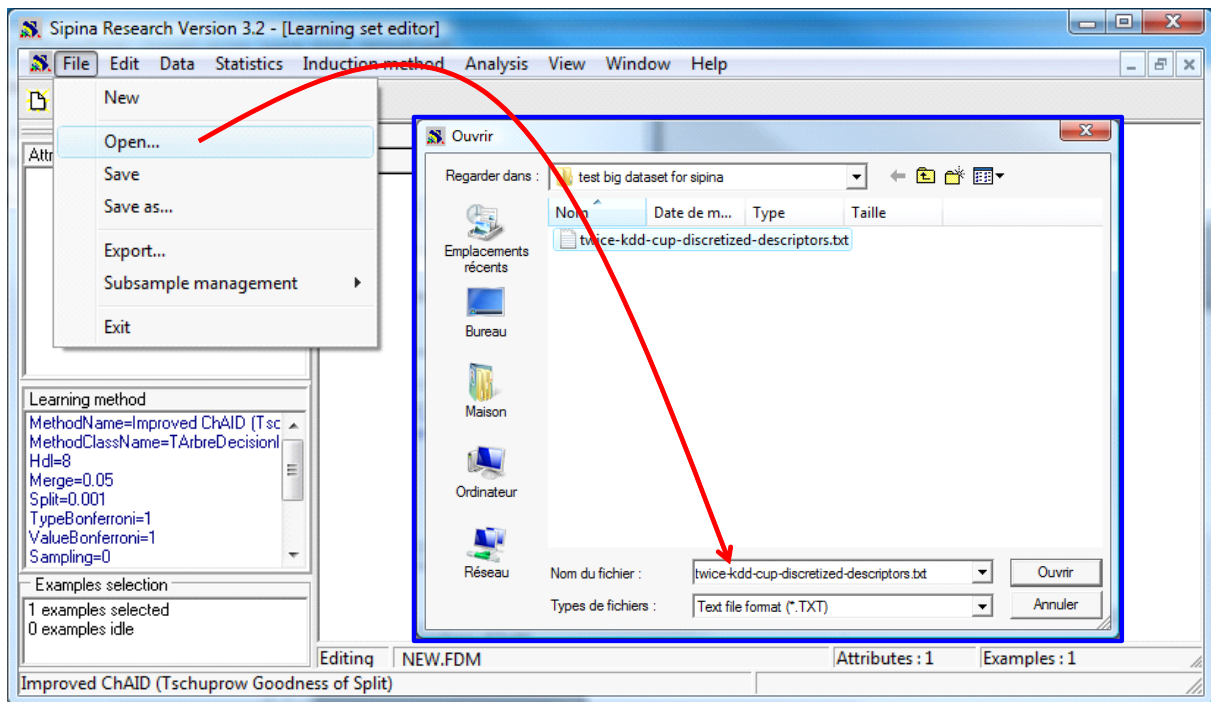
---

<sup>3</sup> Voir <http://tutoriels-data-mining.blogspot.com/2008/03/analyse-interactive-avec-sipina.html> ; <http://tutoriels-data-mining.blogspot.com/2008/03/arbres-interactifs-sipina-et-orange.html>

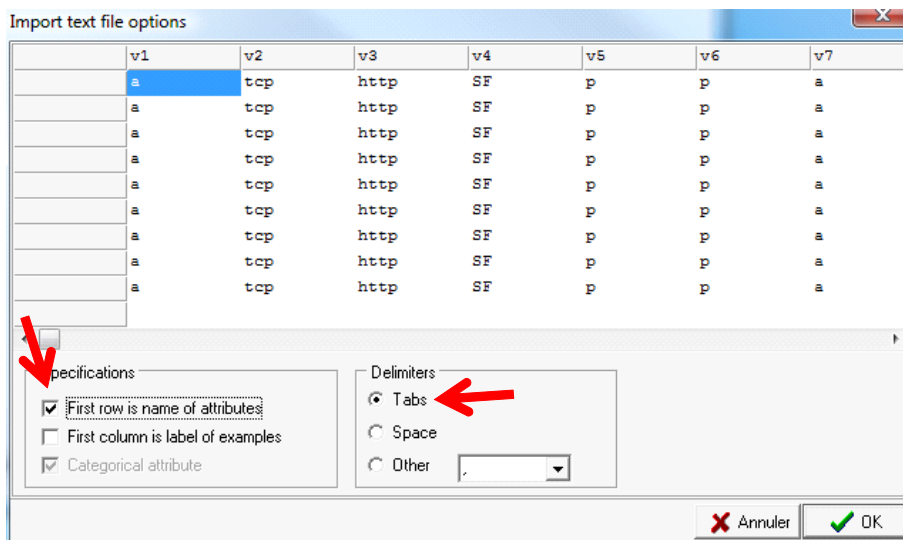
<sup>4</sup> Voir <http://tutoriels-data-mining.blogspot.com/2009/10/sipina-accelerer-par-lechantillonnage.html>

<sup>5</sup> <http://eric.univ-lyon2.fr/~ricco/dataset/twice-kdd-cup-discretized-descriptors.zip>

<sup>6</sup> <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

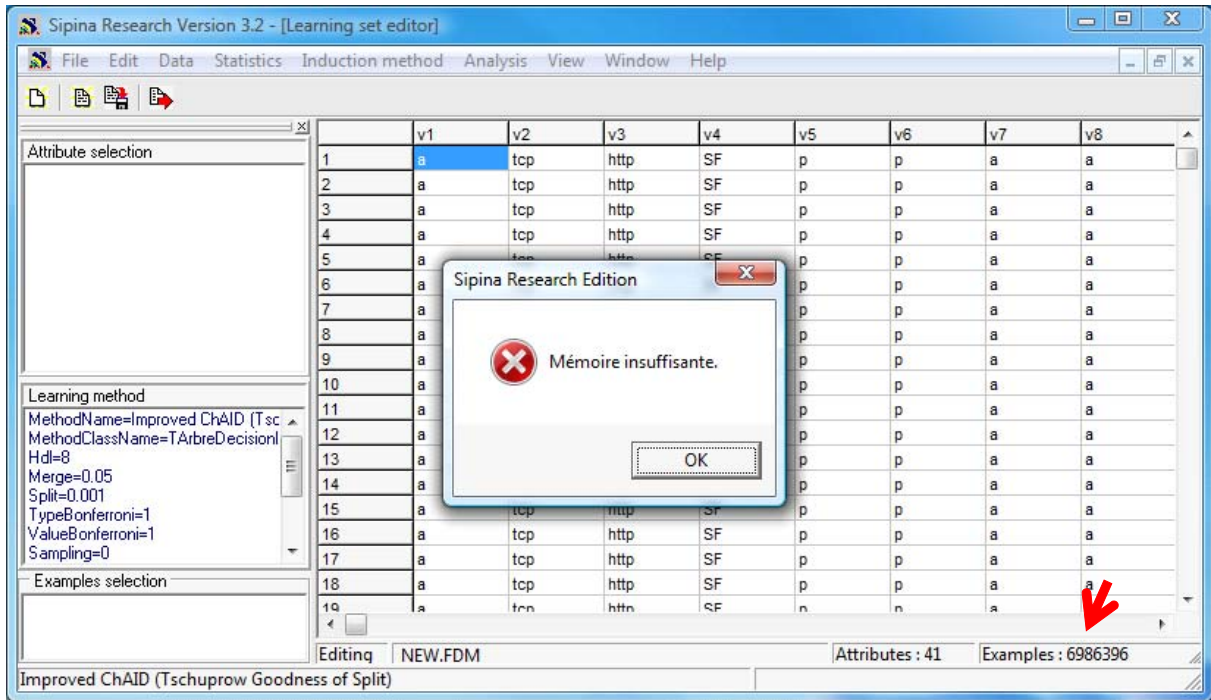


Dans la boîte de dialogue qui apparaît, nous introduisons les options relatives à l'organisation des données dans le fichier (séparateur tabulation, la première ligne correspond au nom des variables).

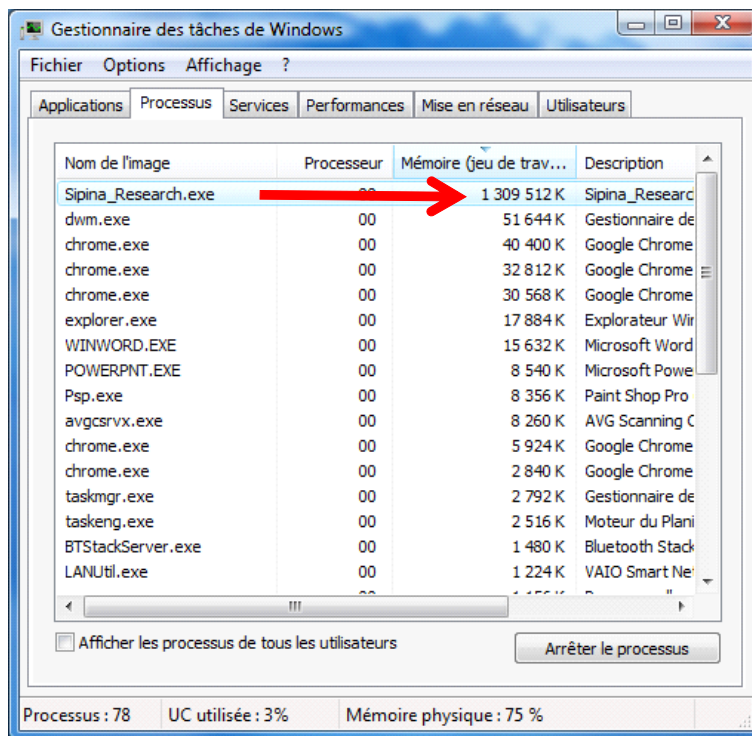


Il ne reste plus qu'à cliquer sur OK. L'importation est démarrée.

Une barre de progression permet de suivre les opérations. L'importation est relativement rapide au regard du volume à traiter. Malheureusement, après un certain temps, le logiciel s'arrête. Le message « Mémoire Insuffisante » apparaît. Il n'est plus possible d'aller plus loin. On se rend compte qu'une fraction du fichier seulement a été traitée (6.986.396 observations).



Le gestionnaire de tâches Windows nous annonce que Sipina occupe 1.309.512 Ko en mémoire centrale.



Pouvoir charger près 7 millions d'observations est déjà intéressant. Mais on se rend compte rapidement que nous ne pouvons réaliser aucune opération, ne serait-ce que des statistiques descriptives. Tout clic intempestif entraîne une erreur.

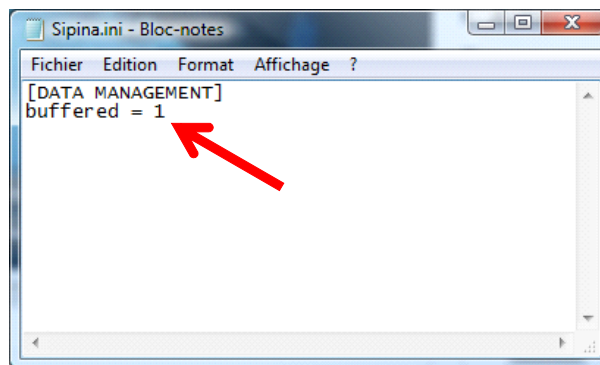
## 4 Swapper les données sur disque

La solution présentée dans ce didacticiel a été mise au point du temps où ma machine de développement ne comptait que 32 Mo de RAM. Elle est tombée en désuétude lorsque le prix de la barrette de mémoire a connu une baisse vertigineuse. En l'étudiant un peu, je me suis rendu compte qu'elle était totalement opérationnelle. Il suffisait de réaliser un petit travail cosmétique pour qu'elle soit facilement accessible aux utilisateurs.

### 4.1 Modifier les options de démarrage

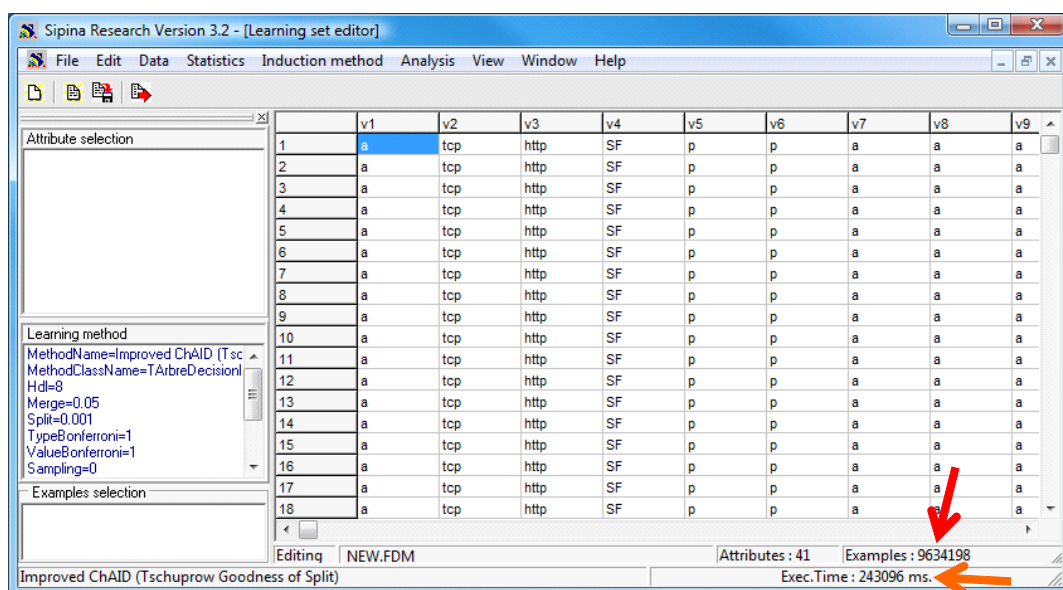
Nous devons refermer SIPINA avant de procéder aux modifications.

Nous allons ensuite dans le répertoire d'installation du logiciel (normalement, sous Windows Vista, C:\Programmes\StatPackages). Nous éditons le fichier **SIPINA.INI**, nous passons l'option BUFFERED à 1.

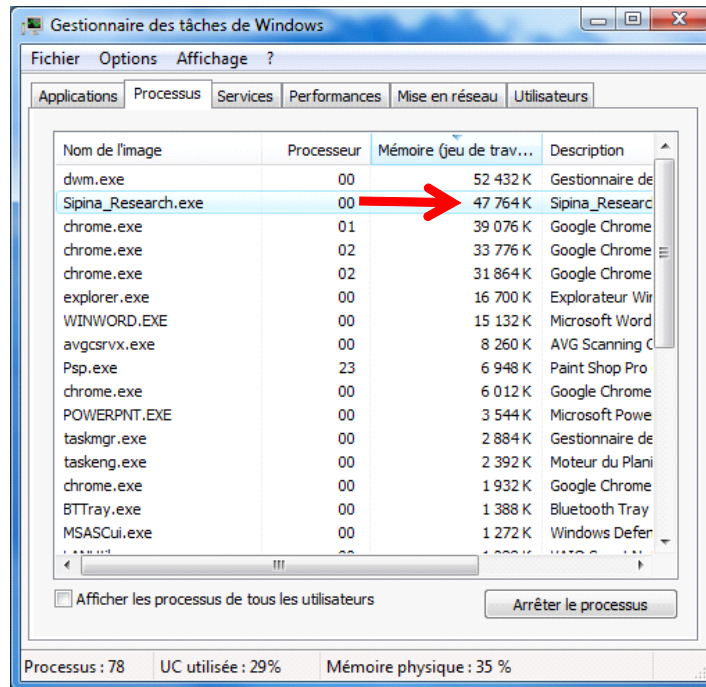


### 4.2 Charger les données

Nous procédons de nouveau à l'importation des données comme précédemment (FILE / OPEN). Elle est menée à bien cette fois-ci, les 9.634.198 observations ont été chargées en 243 secondes (# 4 minutes).



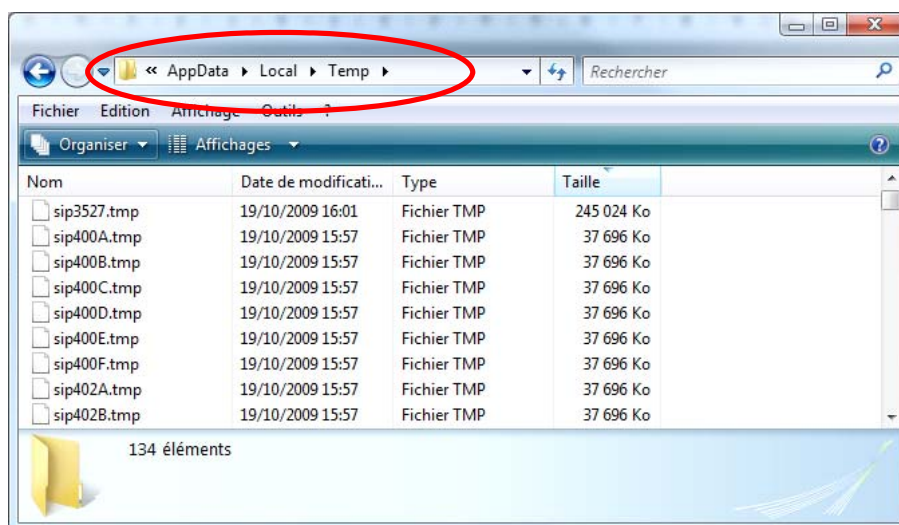
Nous noterons surtout qu'à l'issue de l'importation, l'occupation mémoire de Sipina n'a rien à voir avec la configuration précédente, elle est de 47.764 Ko.



Nous sommes prêts à lancer l'exploration interactive des données.

### 4.3 Stockage des fichiers intermédiaires

Mais auparavant, il est intéressant de préciser l'espace de stockage intermédiaire utilisé par Sipina. Il utilise le répertoire temporaire de l'utilisateur. Nous avons un fichier par variable, et un fichier pour la colonne des labels.



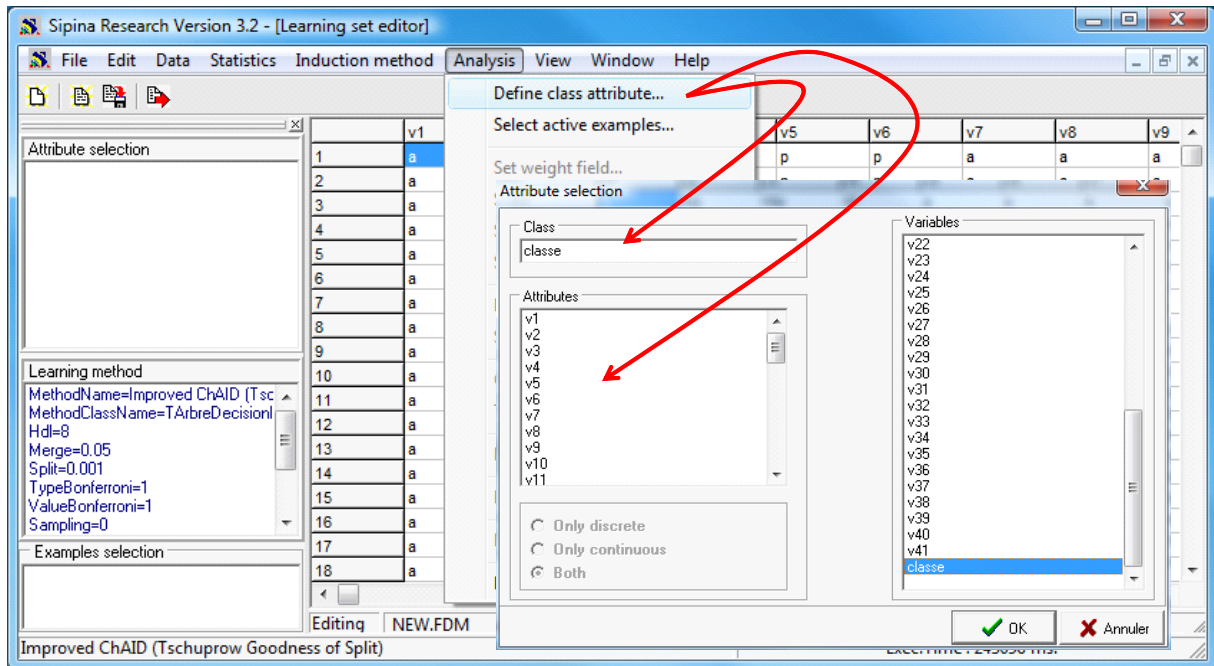
Comme nous pouvons le constater, la taille des fichiers peut être considérable.

**Remarque :** Cette information sur leur emplacement n'est pas anodine. En effet, si Sipina plante pour

une raison quelconque, il n'aura pas le temps de les effacer. Nous devons le faire manuellement.

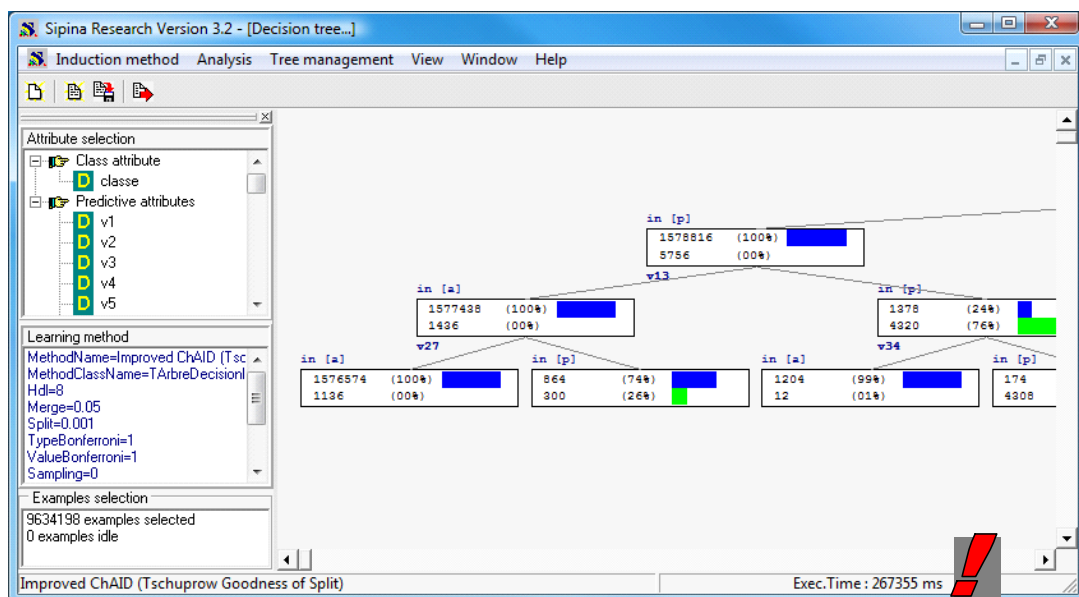
#### 4.4 Définir le problème

Nous devons indiquer à Sipina la variable à prédire et les variables prédictives. Nous actionnons le menu ANALYSIS / DEFINE CLASS ATTRIBUTE. Par glisser déposer, nous effectuons les sélections adéquates.

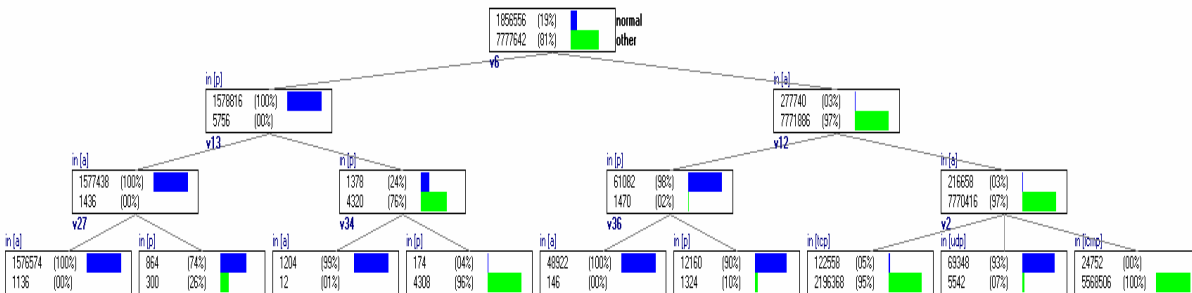


#### 4.5 Lancer l'apprentissage

Par défaut, Sipina utilise la méthode IMPROVED CHAID. Elle limite arbitrairement le nombre de niveaux de l'arbre à 4. Tout cela est paramétrable bien entendu. Nous actionnons le menu ANALYSIS / LEARNING pour lancer les calculs. Nous obtenons un arbre au bout de 267 secondes.

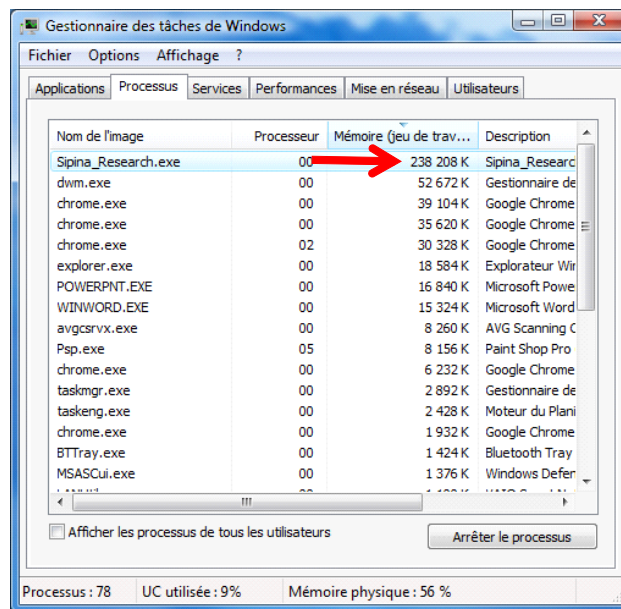


Voici le détail de l'arbre.



## 4.6 Fonctionnalités interactives

L'occupation mémoire de Sipina est de 238.208 Ko si l'on se réfère au gestionnaire de tâches.



Comme nous le disions plus haut, cette inflation est la conséquence du stockage en mémoire de toute une série d'informations utilisées pour l'exploration interactive. Cette fonctionnalité est pleinement opérationnelle ici.

Pour s'en persuader, cliquons sur un des sommets. Nous activons le menu contextuel NODE INFORMATION. Dans la boîte qui apparaît, nous double-cliquons sur la première cellule pour segmenter le sommet courant.

Information window: Level 4, Node 9

Characterization: F v6 in [p] and v13 in [a] and v27 in [p]

Characterization	Goodness of split	Correlation	Accept or Rejection
v40	0.34698424	0.3470	Accept
v11	0.25404169	0.2540	Accept
v10	0.22439654	0.2244	Accept
v3	0.21718857	0.2172	Reject
v35	0.18485279	0.1849	Accept
v29	0.15913448	0.1591	Accept
v12	0.11324158	0.1132	Accept
v5	0.10322406	0.1032	Accept
v4	0.09243306	0.0924	Accept
v1	0.07983717	0.0798	Accept
v37	0.07844353	0.0784	Accept
v34	0.07758900	0.0776	Accept
v36	0.04102273	0.0410	Accept
v26	0.01804028	0.0180	Accept
v30	0.00970836	0.0097	Accept
v28	0.00774291	0.0077	Accept
v25	0.00452920	0.0045	Accept
v17	0.00422705	0.0042	Accept
v33	0.00000000	0.0000	Accept
v38	0.00000000	0.0000	Accept

Split suggestion:

	in [a]	in [p]
normal	748	116
other	76	224

1164 examples (0.01% of the learning set)

Deux nouvelles feuilles sont créées.

De même, nous pouvons inspecter le positionnement des variables dans les sous-groupes associés aux noeuds, calculer les statistiques descriptives localement, etc.

Information window: Level 5, Node 2

Characterization: F v6 in [p] and v13 in [a] and v27 in [p] and v40 in [p]

Continuous attributes: v4 (0.0002)

Values	Strength	Local Dist.	Global Dist.	Recall
SF	-24.22	68 (20%)	7326658 (76%)	0%
S2	-0.10	0 (0%)	308 (0%)	0%
S1	10.38	2 (1%)	1014 (0%)	0%
S3	34.30	2 (1%)	96 (0%)	2%
OTH	-0.06	0 (0%)	114 (0%)	0%
REJ	-4.48	0 (0%)	537748 (6%)	0%
RSTO	303.50	186 (55%)	10612 (0%)	2%
S0	-8.65	0 (0%)	1739170 (18%)	0%
RSTR	107.94	82 (24%)	16154 (0%)	1%
RSTOS0	-0.09	0 (0%)	244 (0%)	0%
SH	-0.27	0 (0%)	2080 (0%)	0%

Discrete attributes: v27 (0.0001)

Values	Strength	Local Dist.	Global Dist.	Recall
a	-73.66	0 (0%)	9066120 (94%)	0%
p	73.66	340 (100%)	568078 (6%)	0%

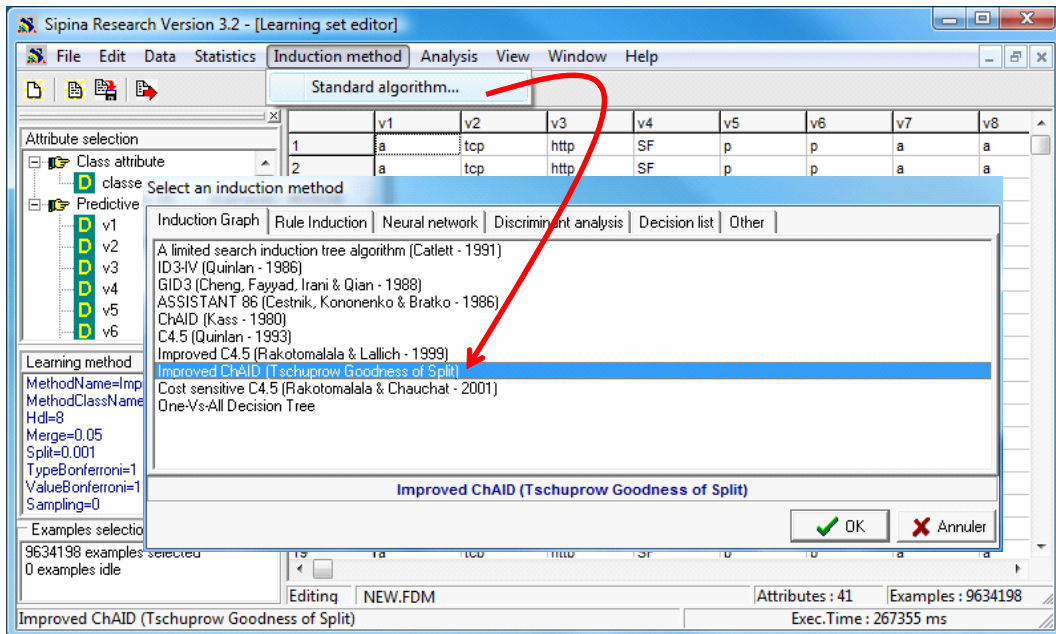
v40 (0.0001)

340 examples (0.00% of the learning set)

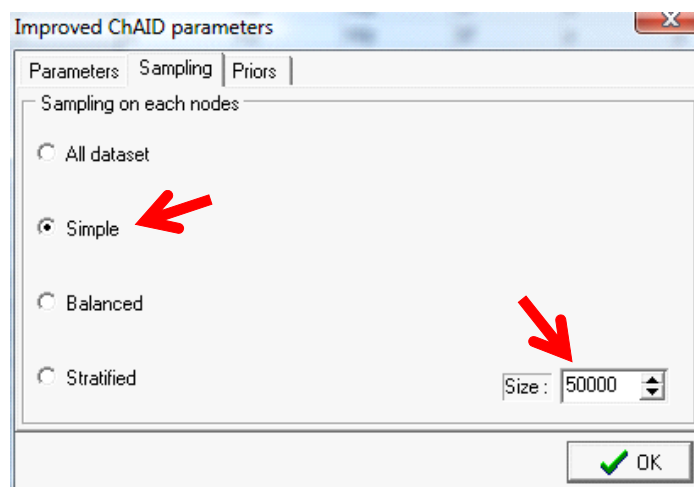
## 5 Utiliser la stratégie d'échantillonnage de Sipina

Il est possible de réduire le temps de calcul en procédant par échantillonnage dans les nœuds. Nous avons développé cette idée dans un de nos didacticiels<sup>7</sup>. Voyons si cette stratégie est toujours opérationnelle lorsque les données sont swappées sur le disque.

Nous arrêtons l'analyse en cours en actionnant le menu WINDOW / CLOSE ALL. Puis, nous définissons la méthode de traitement en cliquant sur INDUCTION METHOD / STANDARD ALGORITHM. Dans la boîte qui apparaît, nous sélectionnons la méthode IMPROVED CHAID.

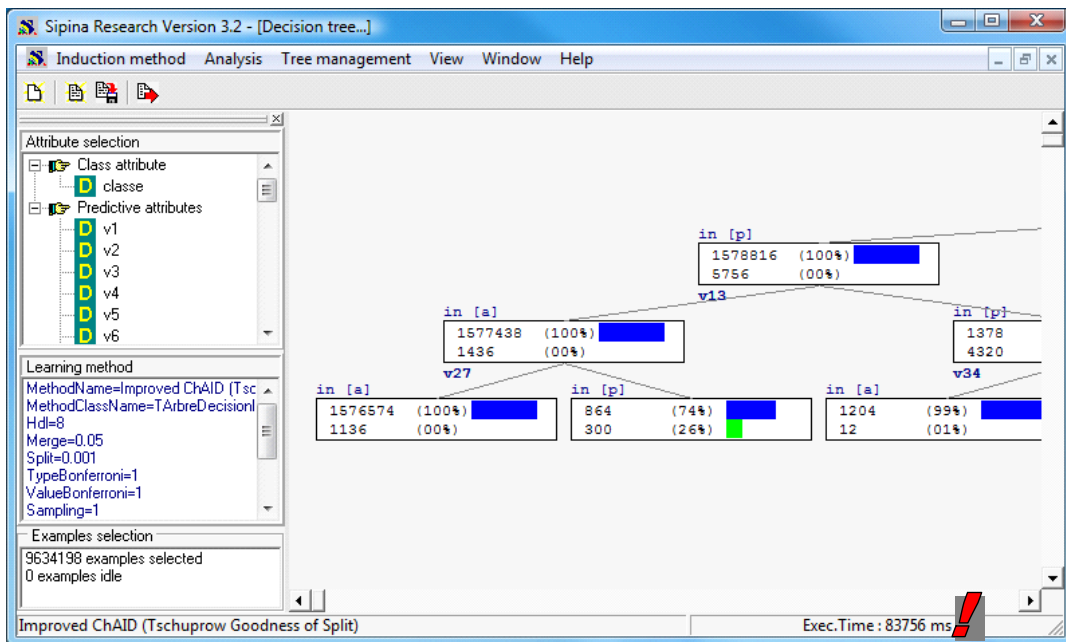


Arrive ensuite la boîte de paramétrage de la méthode, nous allons dans l'onglet SAMPLING : nous activons l'échantillonnage avec une taille d'échantillon à 50.000 pour s'assurer de la stabilité de la solution proposée (nettement moins serait possible).

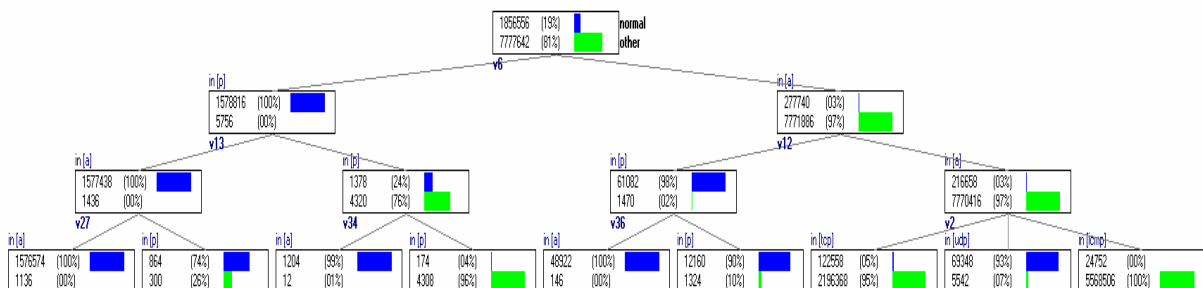


<sup>7</sup> <http://tutoriels-data-mining.blogspot.com/2009/10/sipina-accelerer-par-lechantillonnage.html>

Il ne reste plus qu'à relancer l'analyse en cliquant sur ANALYSIS / LEARNING. Nous obtenons un arbre au bout de 83 secondes (#1 minutes et 20 secondes). La réduction du temps de calcul sera d'autant plus spectaculaire que nous aurons une majorité de descripteurs continus.



Voici le détail de l'arbre, il est identique à celui calcul sur la totalité des données.



Bien entendu, toutes les fonctionnalités interactives sont disponibles (visualisation des individus sur les nœuds, exportation des individus, statistiques descriptives, etc.).

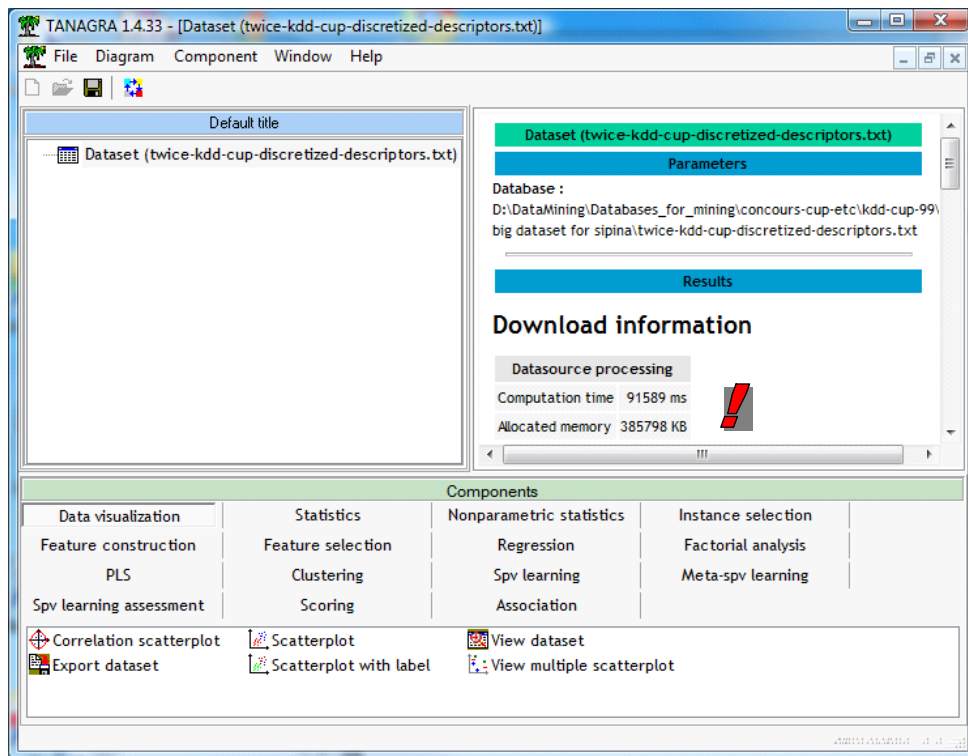
## 6 Les autres logiciels

Face à un tel volume de données, comment se comportent les autres logiciels ?

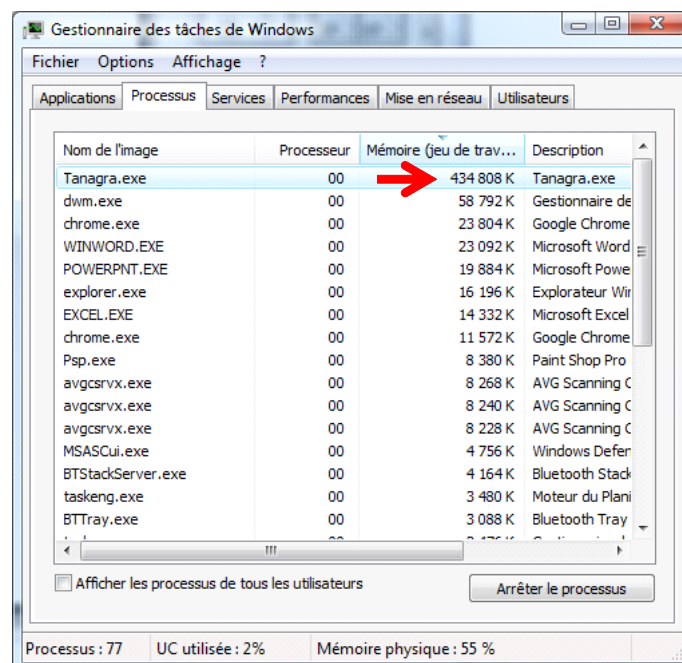
### 6.1 Tanagra

Tanagra charge toutes les données en RAM. Sa chance est qu'il encode les variables qualitatives sur un seul octet (BYTE). C'est pour cela d'ailleurs que le nombre de modalités de ce type de variable est limité à 255. L'occupation mémoire théorique du fichier est connu à l'avance, il est égal à  $(1 \times 41 \times 9.634.198 \# 377 \text{ Mo})$  – il faut compter aussi les autres informations annexes c.-à-d. le dictionnaire des données, les noms des variables, etc.). Voyons ce qu'il en est lorsqu'on procède au traitement de notre fichier.

Nous chargeons le fichier en créant un nouveau diagramme (FILE / NEW)<sup>8</sup>. Après 91 secondes, les données sont chargées.

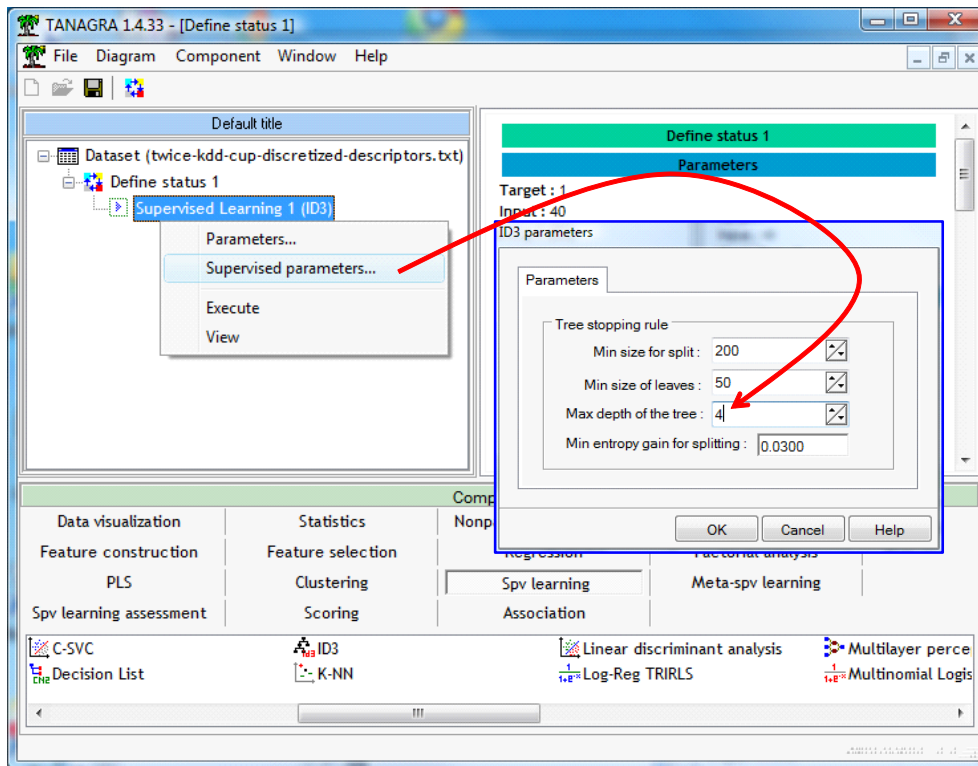


Tanagra occupe 434.808 Ko dans Windows à ce stade.

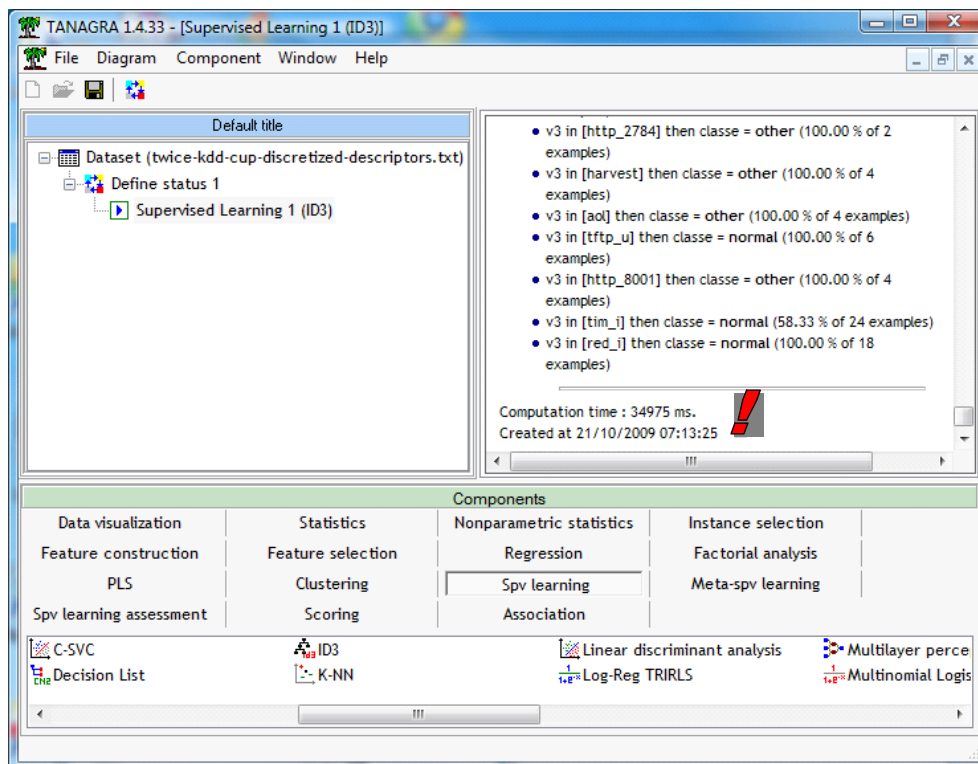


<sup>8</sup> Nous ne donnons que les grandes lignes ici, voir les tutoriels dédiés à la construction des arbres pour le détail de la manœuvre : <http://tutoriels-data-mining.blogspot.com/search/label/Arbres%20de%20d%C3%A9cision>

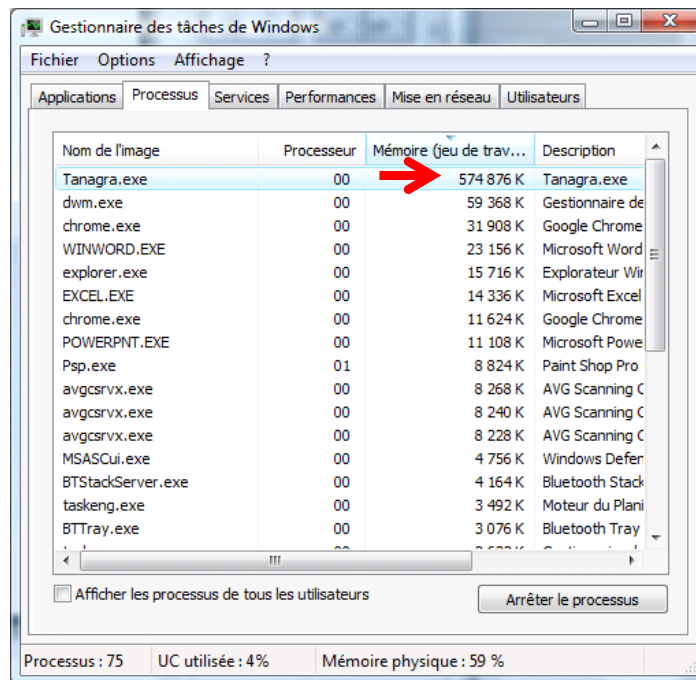
Après avoir défini le rôle des variables à l'aide du composant DEFINE STATUS (CLASSE en TARGET, V1..V41 en INPUT). Nous insérons la méthode ID3. Nous limitons le nombre de niveaux à 4 via la boîte de paramétrage (menu contextuel SUPERVISED PARAMETERS).



Nous lançons les calculs en cliquant sur VIEW.



Nous obtenons l'arbre au bout de 35 secondes, à comparer avec les 267 secondes qui ont été nécessaires à Sipina.



L'occupation mémoire est passée à 574.876 Ko. Mais cette valeur n'est pas comparable avec celle de Sipina. En effet, les arbres ne sont pas interactifs dans Tanagra. De fait, il conserve très peu d'informations sur les nœuds. La « gourmandise » mémoire est donc naturellement moindre.

On notera en tous les cas qu'il a réussi à mener à son terme la construction de l'arbre. Mais la solution n'est pas « scalable ». Si nous avons 4 fois plus d'observations, le chargement de la totalité des données en mémoire n'est pas possible. Sipina, avec la configuration décrite dans ce didacticiel, ne souffrira pas de ce genre de problèmes.

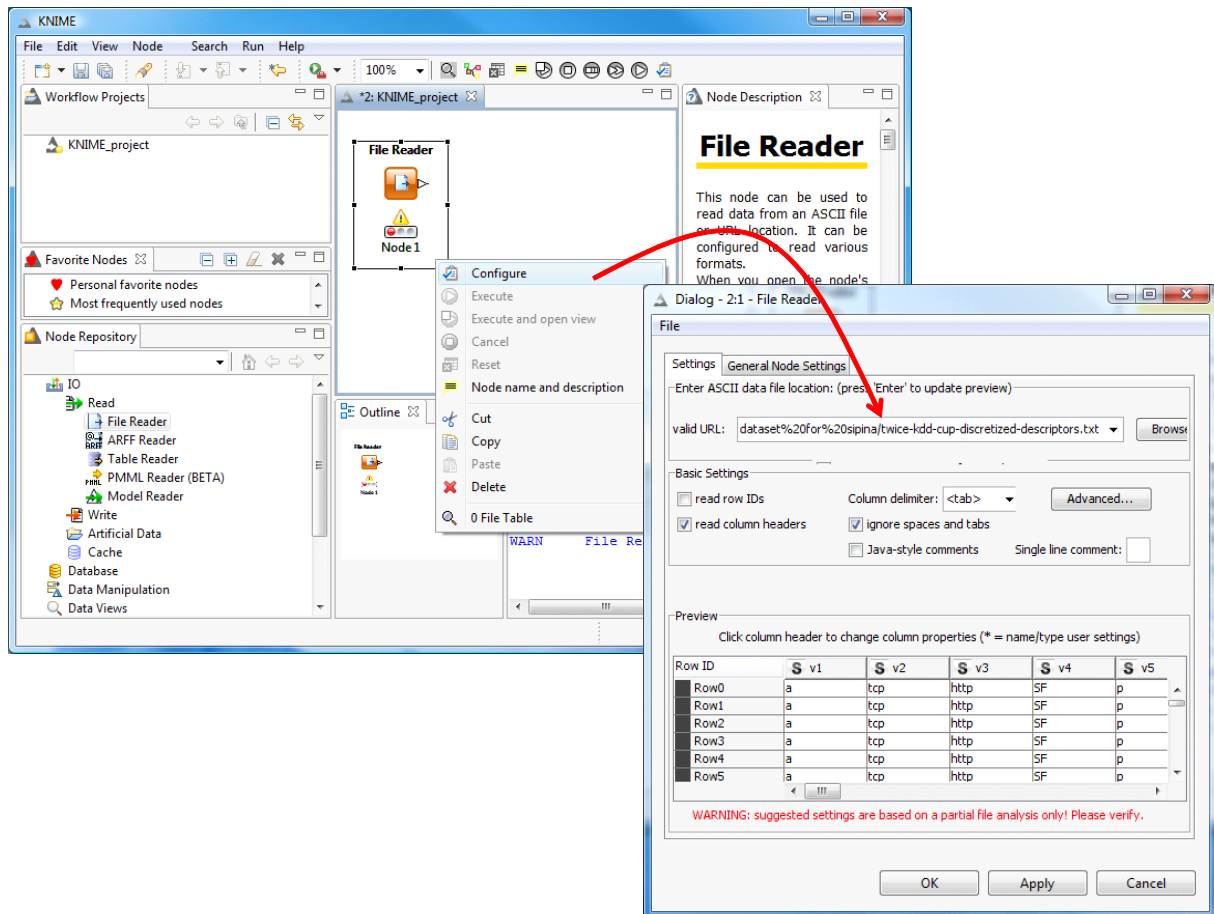
## 6.2 Knime

Parmi les outils libres, Knime (<http://www.knime.org/>) est, semble-t-il, le mieux armé pour attaquer des grandes bases. Il intègre une solution de swap<sup>9</sup>.

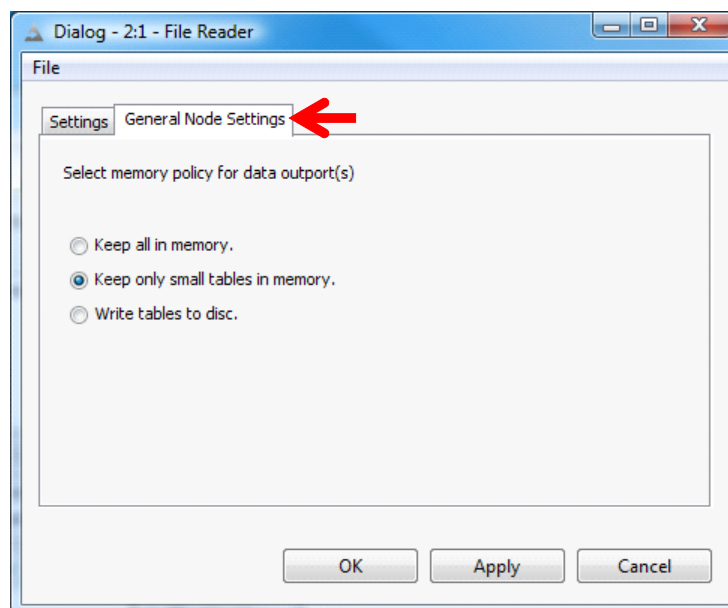
Nous utilisons la version 2.0.3 dans ce didacticiel. Après avoir démarré le logiciel et créé un diagramme (workflow), nous insérons le composant FILE READER.

Que nous branchons sur notre fichier de données, elles sont visibles dans la grille de prévisualisation.

<sup>9</sup> Je n'ai pas su trouver de la documentation explicitant ce sujet sur leur site, si vous avez des informations...

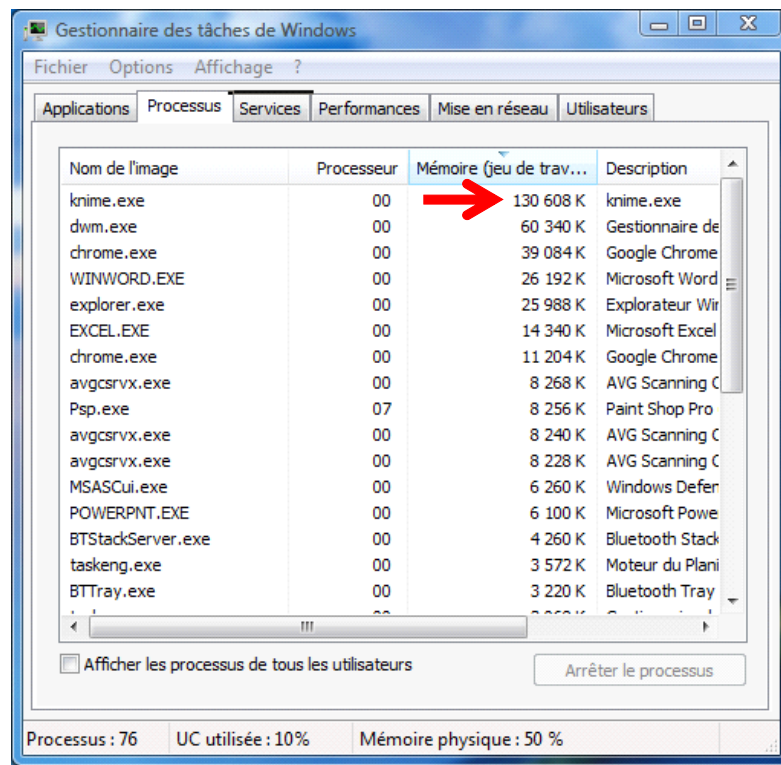


Le secret de l'affaire réside dans le second onglet GENERAL NODE SETTINGS. Knime nous annonce qu'il ne chargera en mémoire que les petits fichiers. Nous pouvons même explicitement écrire les données sur disque.



Il ne reste plus qu'à valider et à cliquer sur le menu contextuel EXECUTE.

Au bout d'un certain temps (aucune indication sur le temps d'exécution n'est disponible), les données sont chargées. Et surtout l'occupation mémoire de Knime est bien contrôlée, elle n'est que de 130.608 Ko, preuve qu'une partie des données au moins est placée sur le disque.



Nous ajoutons ensuite le composant DECISION TREE LEARNER. La documentation contextuelle indique que Knime réalise la construction de l'arbre exclusivement en mémoire. Nous verrons si cela s'avère rédhibitoire en ce qui concerne notre fichier.

Dans la boîte de paramétrage, il n'est pas possible de spécifier le nombre maximum de niveaux. Nous utilisons MIN NUMBER OF RECORDS PER NODE que nous mettons arbitrairement à 50.000 pour limiter la taille de l'arbre. On notera au passage qu'il est possible de spécifier le nombre de « threads ». Ma machine étant un Quad Core, je le fixe à 3 (*le dernier « cœur » est utilisé pour pouvoir surfer tranquillement pendant les calculs, il faut bien passer le temps*) (Figure 1).

EXECUTE and OPEN VIEW lance les calculs. Malheureusement, au bout de quelques secondes, Knime nous annonce qu'il n'est pas possible de traiter les fichiers par manque de mémoire (Figure 2).

Il nous dit aussi qu'en paramétrant différemment la machine virtuelle JAVA, en augmentant la taille du tas (HEAP), il est possible d'améliorer la situation. Il faudrait éditer le fichier KNIME.INI (Figure 3).

Passer par un échantillon est la seconde solution proposée, nous y reviendrons dans la conclusion.

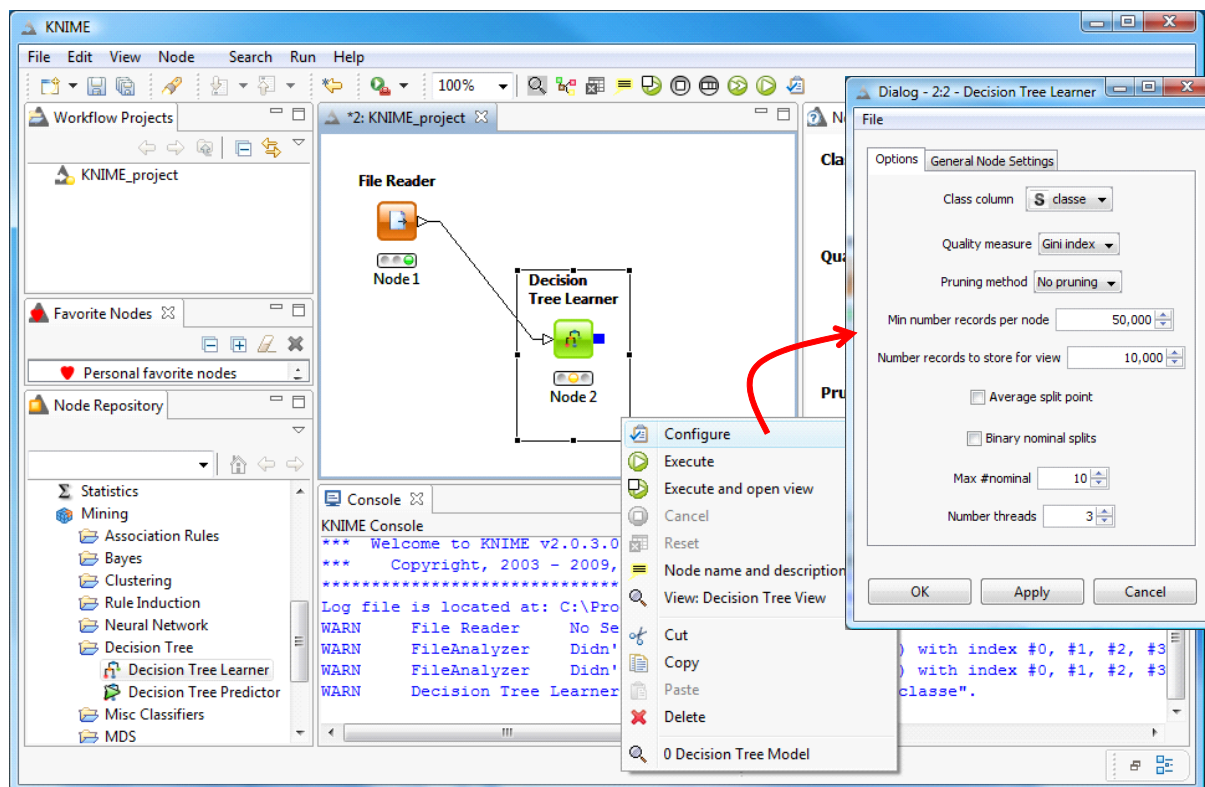


Figure 1 - Configuration de DECISION TREE LEARNER dans Knime

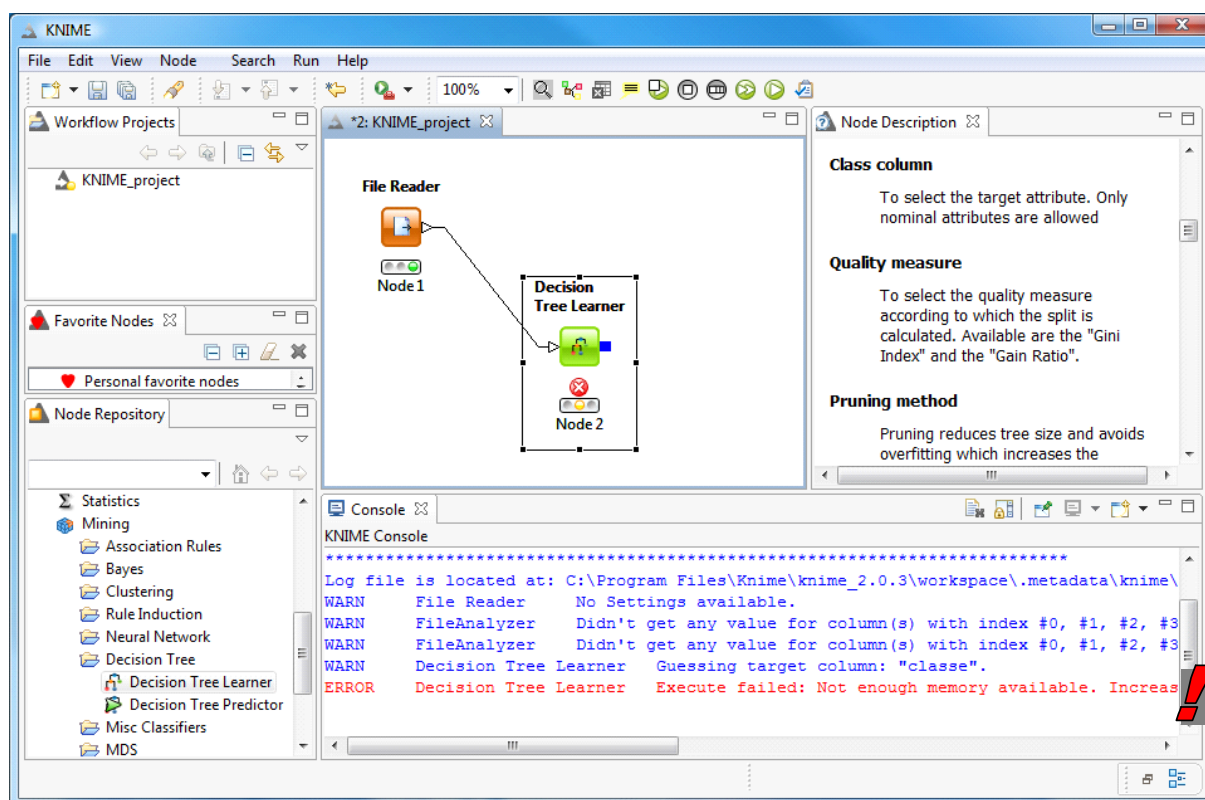


Figure 2 - Knime - Problème d'occupation mémoire avec les arbres

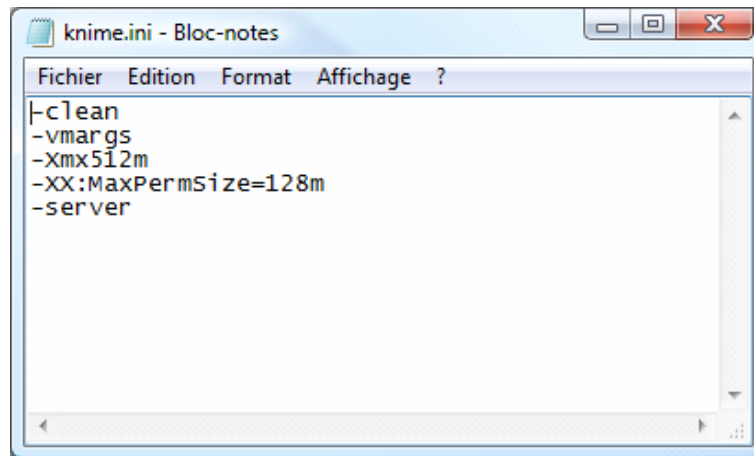


Figure 3 - Knime.ini - Fichier de configuration de Knime

## 7 Conclusion

D'un côté, implémenter des outils capables de traiter un fichier de 9 millions d'observations est assez exaltant pour le développeur que je suis. D'autant que, comme nous avons pu le constater dans ce didacticiel, les temps d'exécution (chargement des données, construction de l'arbre, exploration des sommets) restent très raisonnables au regard du volume de données manipulé.

De l'autre, le data miner reste assez perplexe par rapport à tout ça. L'objectif est d'extraire de l'information « utile » à partir des données. Plutôt que de s'escrimer à manipuler des fichiers titanesques (« tera-octets », quand tu nous tiens), plus ou moins aisément selon les logiciels, se poser la question d'un échantillonnage judicieux avant de procéder aux traitements est bien plus avantageux. C'est ce que nous suggère d'ailleurs Knime face à de très grands fichiers.