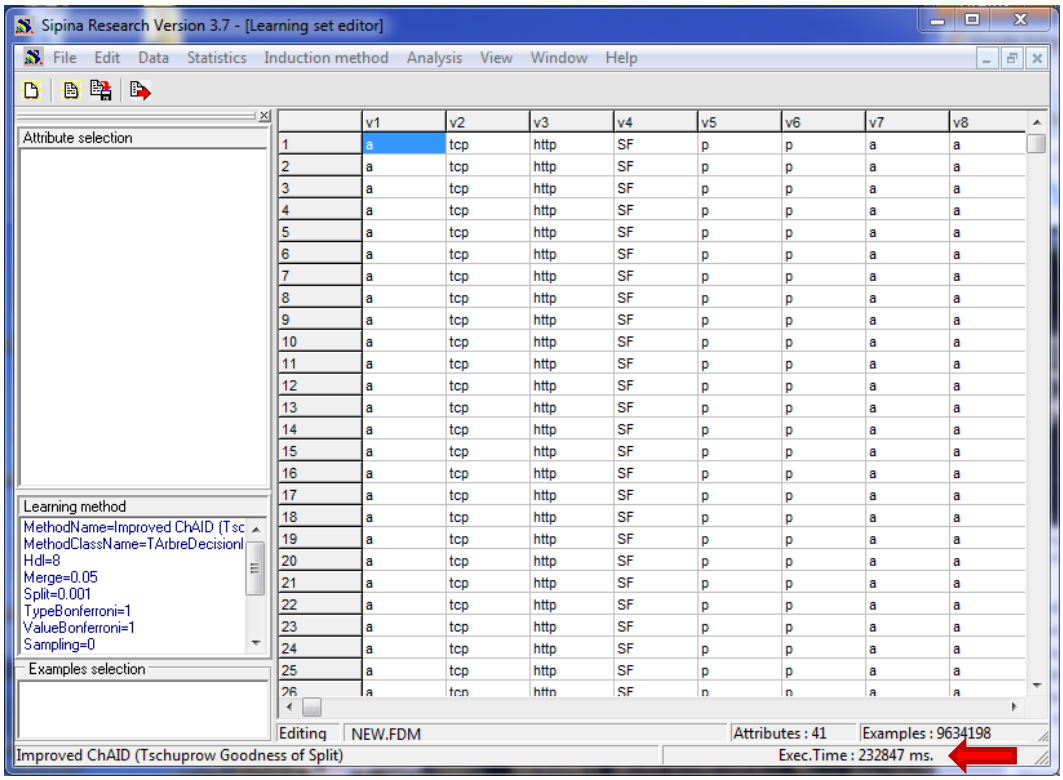
1 Objectif

Etudier le comportement des outils 64 bits lors de la construction d'un arbre de décision sur une très grande base de données.

Triturer des très grands fichiers était de fantasme ultime du data miner a-t-on coutume de dire. Etant passé récemment à un système 64 bits (mieux vaut tard que jamais), je me propose d'étudier le comportement des outils spécifiquement dédiés à ce système, principalement Knime 2.4.2 et RapidMiner 5.1.011. Ce document vient compléter une étude récente où nous traitions une base moyennement volumineuse avec 500.000 observations et 22 variables. Nous poussons le curseur un peu plus loin en reprenant un tutoriel où le fichier à traiter comportait **9.634.198 observations** et **41 variables,** (quasiment) impossible à faire tenir en mémoire sur un système 32 bits. L'idée était alors de montrer qu'un système de swap adapté aux algorithmes d'apprentissage, l'induction d'un arbre de décision en l'occurrence, permettait d'appréhender de très grandes bases avec des temps de traitement raisonnables. La procédure a été implémentée dans Sipina.

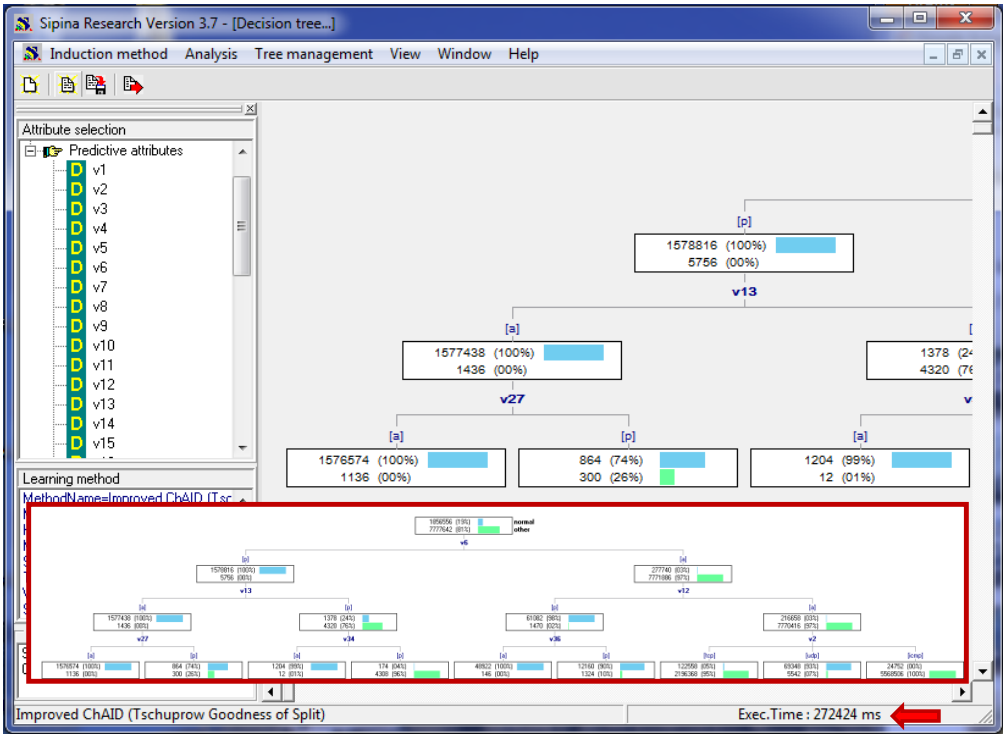
2 Traitement avec SIPINA

Les manipulations adéquates pour que SIPINA « dumpe » la base sur le disque en vue des traitements est décrite dans notre précédent tutoriel[1]. Il faut simplement modifier les options de démarrage dans le fichier **SIPINA.INI**.



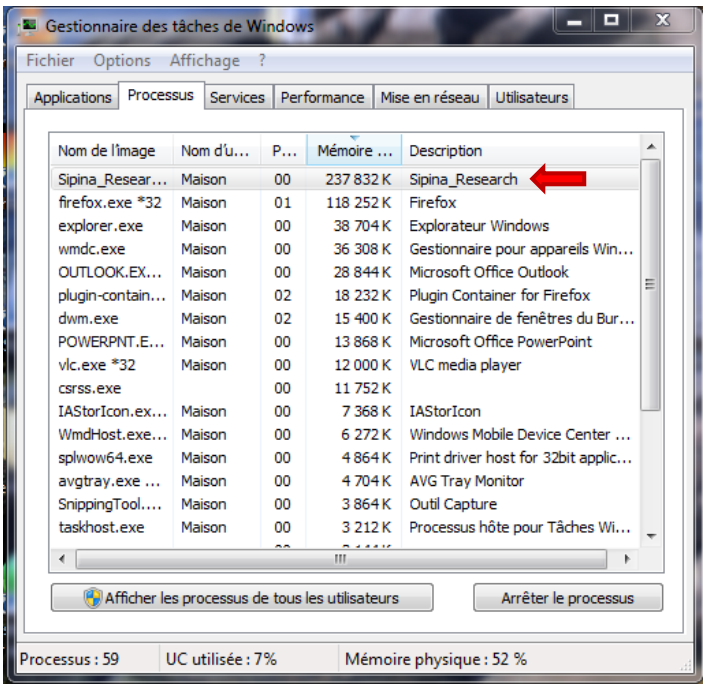
Les données sont importées en 232 secondes, l'arbre est construit en 272 secondes (≈ 5 minutes).

[1] http://tutoriels-data-mining.blogspot.com/2009/10/sipina-traitement-des-tres-grands.html



Toutes les manipulations sont décrites par ailleurs, nous n'y reviendrons pas. Tout juste nous nous bornerons à dire que le passage du système en 64 bits n'a guère d'influence sur le comportement de SIPINA qui est compilé en 32 bits. A machine égale, les temps de traitement sont identiques.

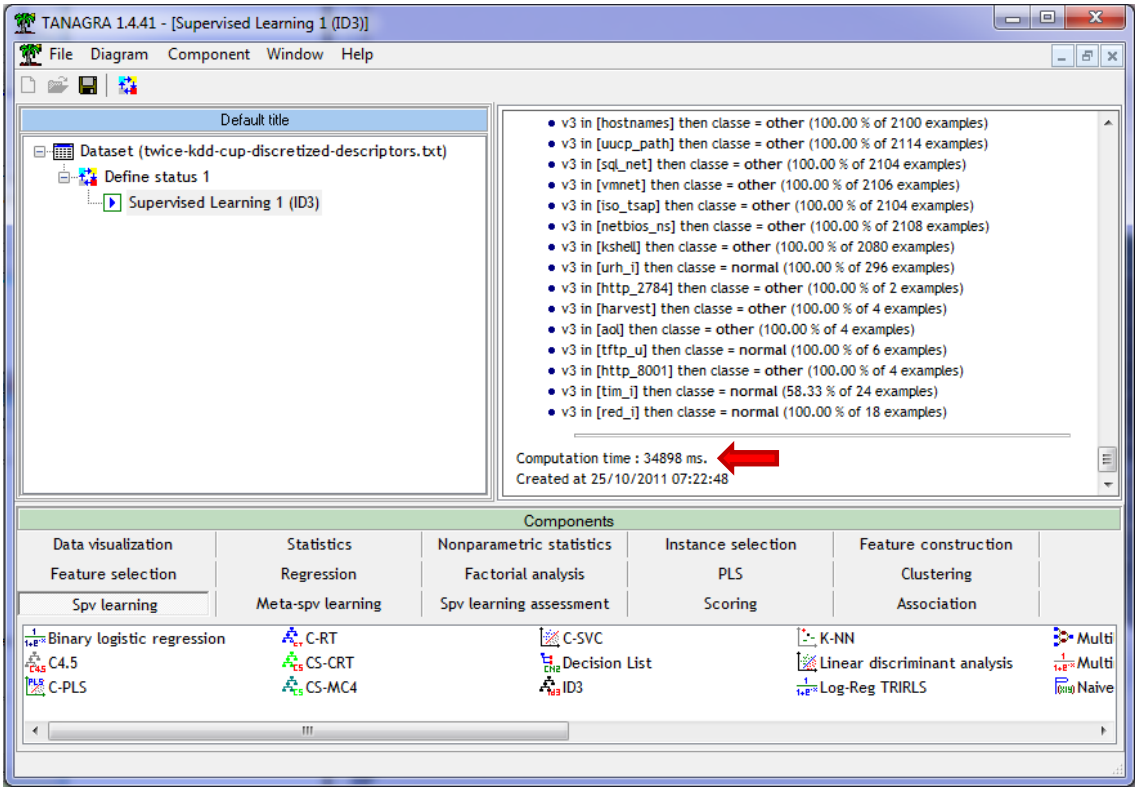
L'occupation mémoire à la sortie des calculs est de ≈ 232 Mo. Nous sommes très loin de saturer les capacités de la machine. En réalité, nous devons pouvoir traiter des bases réellement gigantesques. C'est tout l'intérêt de la solution de swap[2].



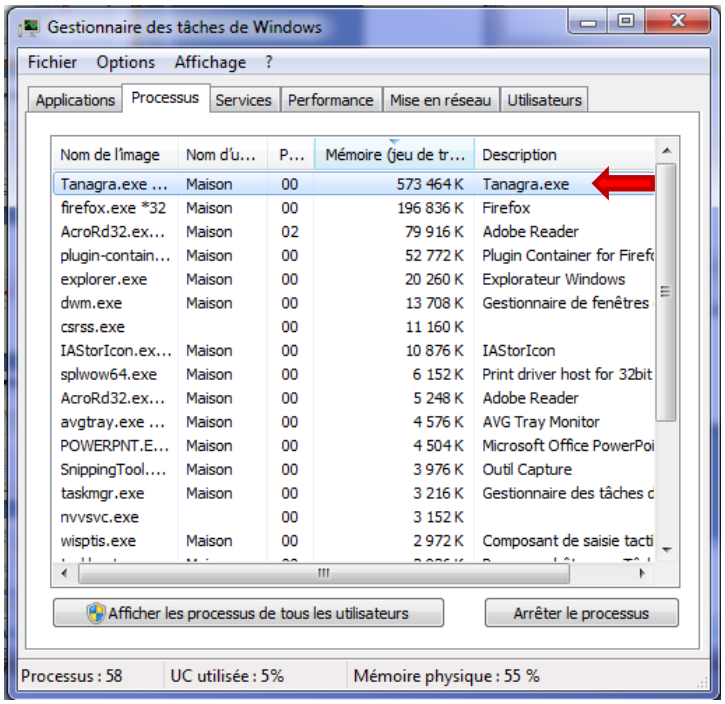
[2] Notons que le système sera d'autant plus efficace que le disque est performant. Avec les nouveaux supports SSD (http://fr.wikipedia.org/wiki/Solid-state_drive), les temps de traitement devraient être encore plus sympathiques.

3 Traitement avec Tanagra

De même, placé dans ce nouveau contexte (Windows 7 – 64 bits), Tanagra ne modifie pas son comportement. La base est importée en 87 secondes, et l'arbre est construit en 34 secondes. Ces valeurs sont très similaires à celles constatées sur un système 32 bits.

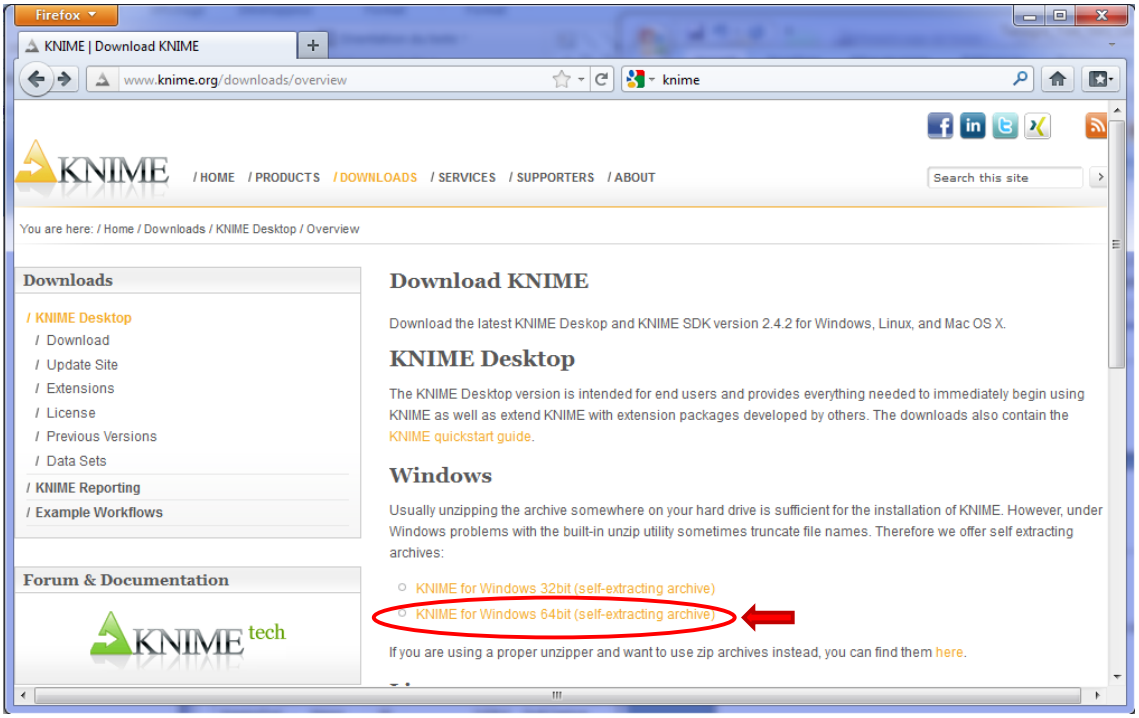


Grâce à un codage spécifique des **variables qualitatives** (1 octet par valeur, ainsi elles sont **limitées à 255 modalités**), l'occupation mémoire reste contenue à l'issue des traitements (≈ 560 Mo).



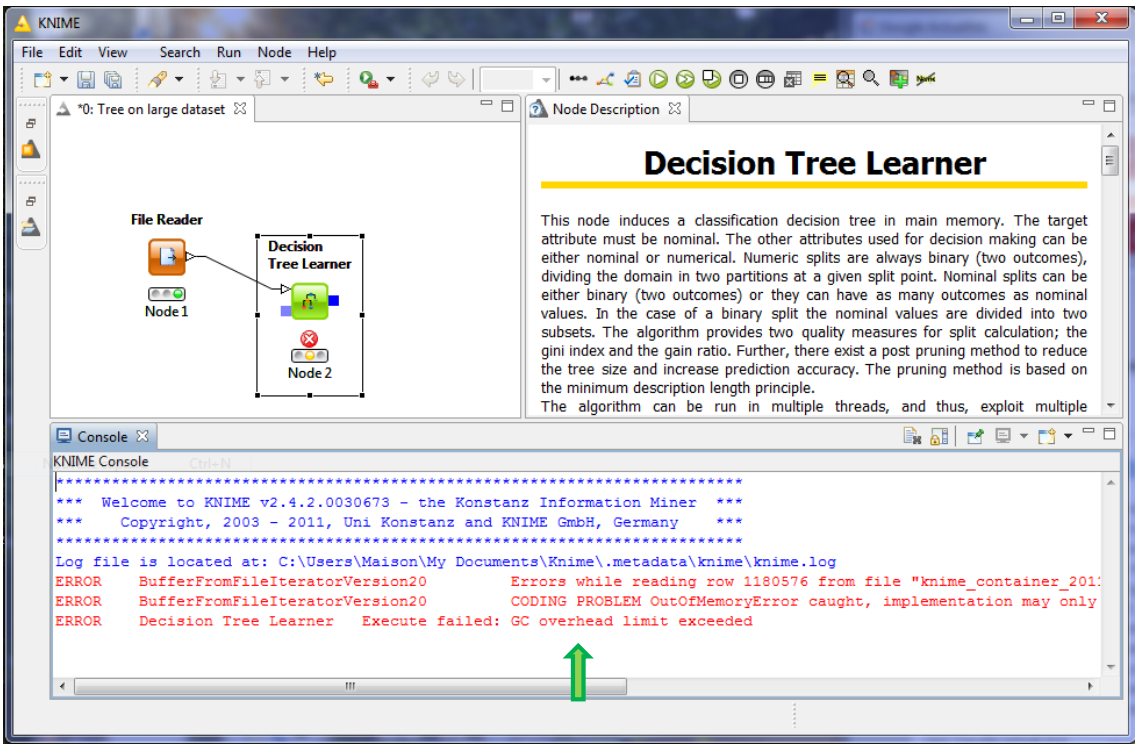
4 Traitement avec Knime

Sur le site de Knime, nous avons la possibilité de charger la version 32 bits ou 64 bits pour Windows.

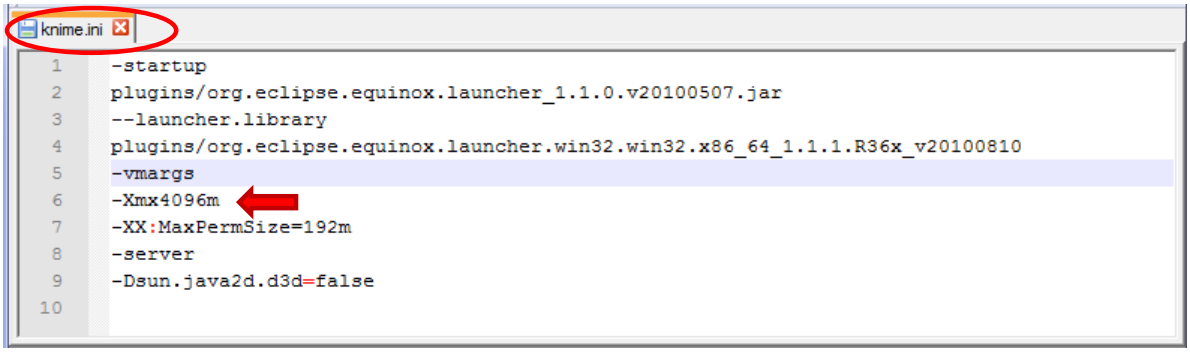


Nous optons pour la seconde bien évidemment pour ce tutoriel. Nous l'installons sur notre machine. Nous passons rapidement sur les manipulations à réaliser pour la définition de l'analyse, ils ont été décrits précédemment (section 6.2).

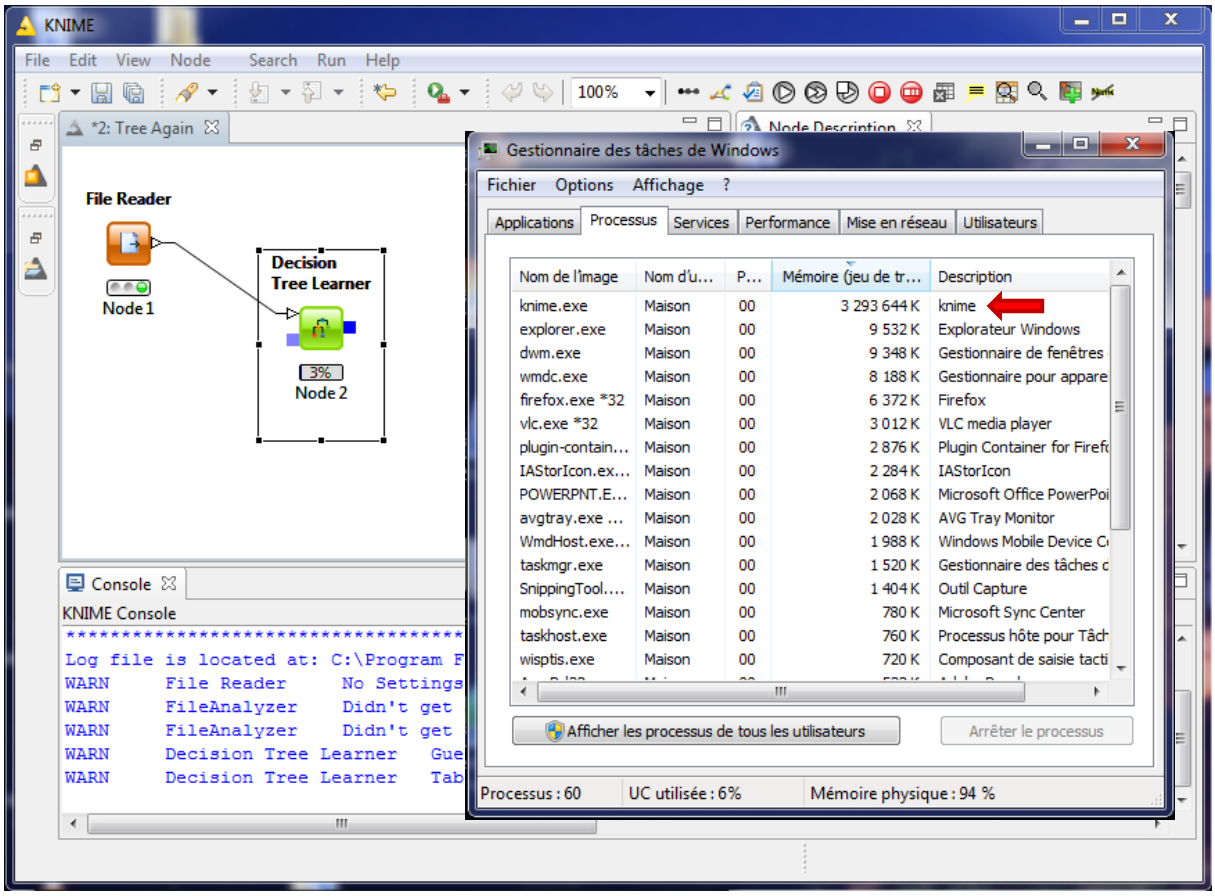
Voyons directement ce qu'il en est à l'issue des traitements. Knime nous envoie un message d'erreur assez sibyllin, tout du moins dans un premier temps.



Une investigation rapide (ex. http://www.petefreitag.com/item/746.cfm) permet de comprendre que nous pouvons dépasser cette limitation en modifiant la mémoire allouée au système dans le fichier **KNIME.INI**. Mais combien devons-nous mettre ? La base comporte 9.634.198 observations et 41 variables. Si les valeurs sont codées sur 4 octets, l'occupation mémoire serait[3] - très approximativement – de 1.47 Go. Si nous sommes sur 8 octets, elle serait de 2.94 Go. Fort de cette information, nous modifions le fichier de configuration en augmentant la taille maximale du tas (4096 Mo = 4 Go).



Maintenant le logiciel peut traiter les données. Ces dernières ont bien été importées et la construction de l'arbre est entamée.

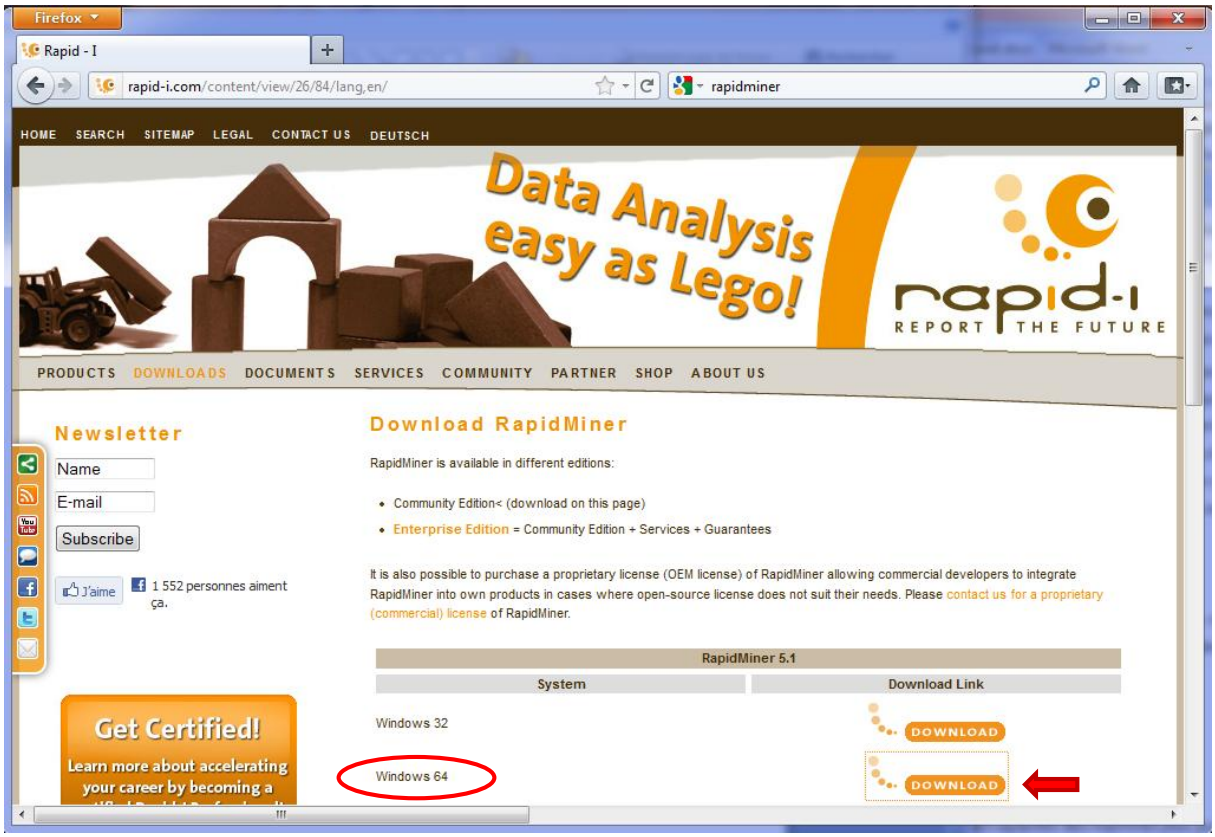


[3] (9.634.198 x 41 x 4) / (1024 x 1024 x 1024) ≈ 1.47 Go. Nous ne tenons pas du tout compte des méta-informations relatives aux données dans ce cas (nom de variables, dictionnaires des valeurs, etc.).

Un coup d'œil sur le gestionnaire de tâches de Windows montre que Knime occupe ≈ 3.14 Go. Survient alors un autre problème, ma machine ne disposant que de 4 Go, Windows s'est mis à utiliser intensivement le fichier d'échange pour libérer de la mémoire. Etant en RAID 1, l'écriture des informations sur le disque prend du temps, et j'avoue qu'après une demi-heure d'attente, l'avancement de l'arbre étant figé à 3% (cf. la copie d'écran), j'ai préféré arrêté les frais.

Que faut-il en penser ? Nul doute que si je disposais plus de mémoire vive, j'aurais pu spécifier une valeur plus élevée dans le fichier de configuration KNIME.INI, et surtout Windows n'ayant pas besoin de copier de contenu de la mémoire dans le fichier d'échange, nous aurions pu mener à terme les calculs plus rapidement. Il n'y a donc plus de limitation théorique au chargement de toutes les données en mémoire, en tous les cas elle est suffisamment élevée pour que nous n'en ayons pas à nous en préoccuper. Les caractéristiques de la machine constituent le seul vrai goulot d'étranglement.

5 Traitement avec RapidMiner

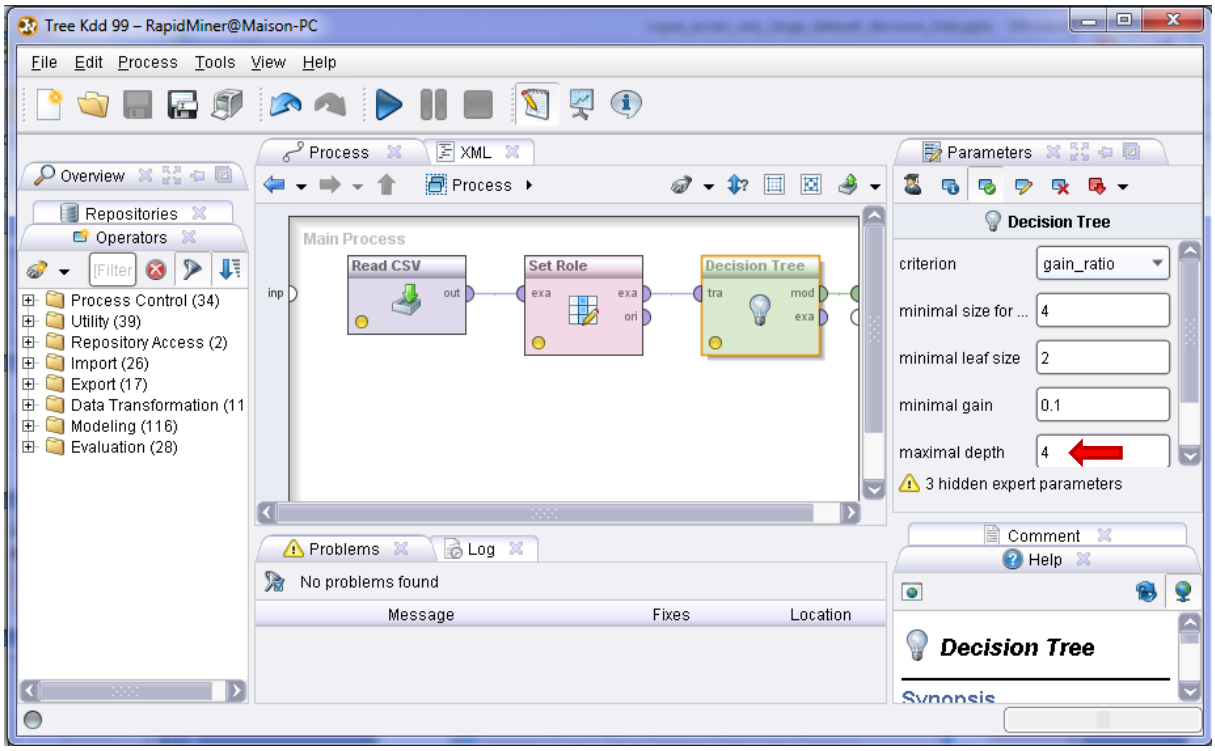


Knime et RapidMiner sont très similaires à différents points de vue. Tous deux mettent à mal l'hégémonie de Weka qui peine à suivre en termes de fonctionnalités et de qualité de l'interface[4]. Nous chargeons la version 64 bits sur le site de Rapid-I.

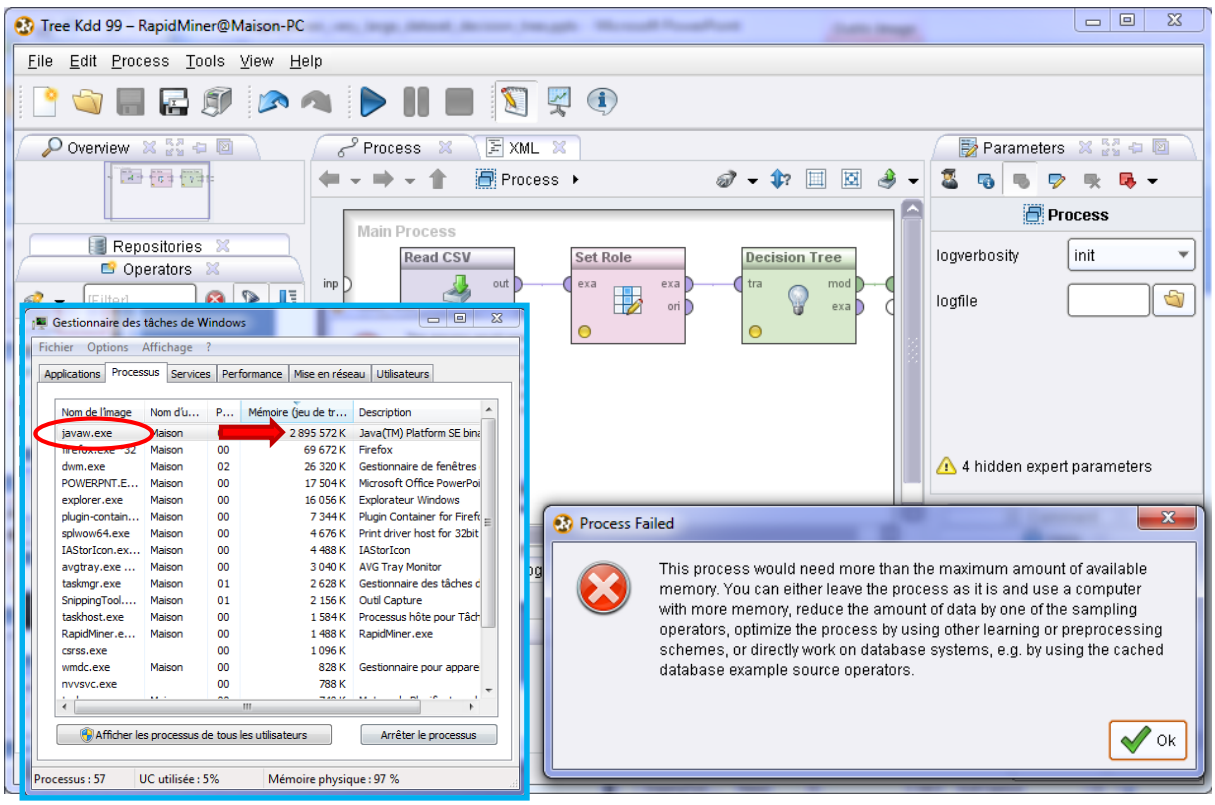
[4] Knime, RapidMiner et Weka, autour desquels s'est développée une activité commerciale ces dernières années, se positionnent sur un créneau différent par rapport à d'autres outils tels que R, SIPINA, TANAGRA ou ORANGE. Ces trois logiciels proposent des variantes payantes d'ailleurs. N'y ayant pas accès à ces dernières, je ne peux pas me prononcer concernant leurs capacités à traiter les très grandes bases.

5.1 Traitement avec les paramètres standards

Nous ne détaillons pas les manipulations dans ce didacticiel. Pour ceux qui ne connaissent peu RapidMiner 5, une description de l'interface est accessible sur notre site des tutoriels.



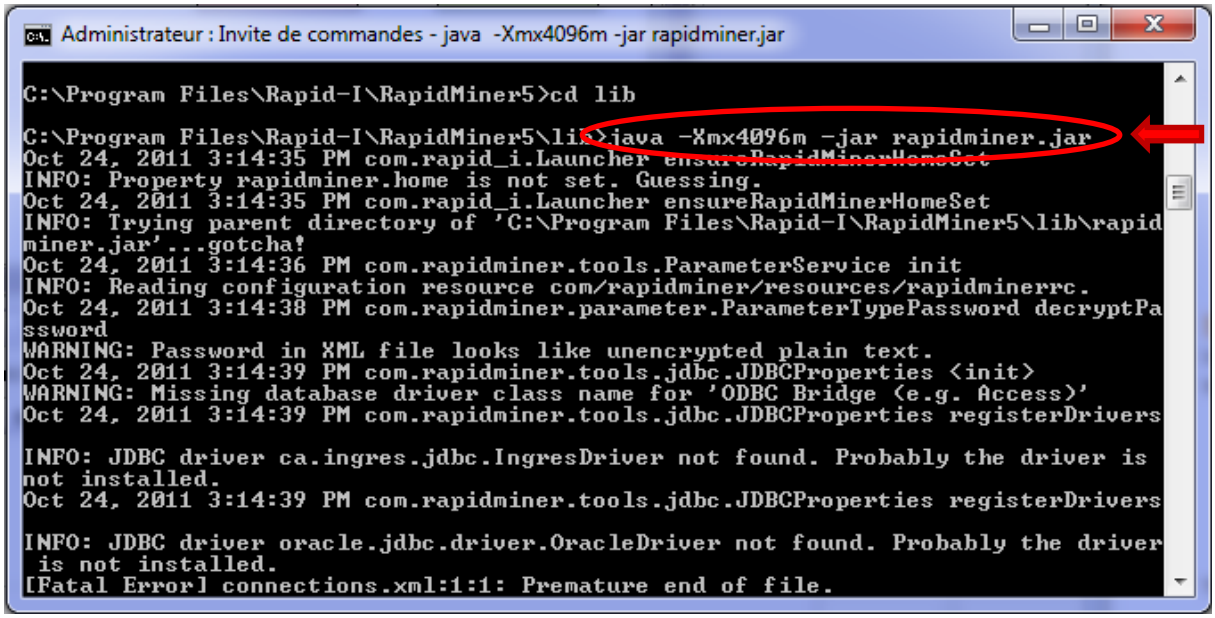
Nous avons démarré RapidMiner via le raccourci sur le bureau. Nous avons défini à 4 niveaux la profondeur maximale de l'arbre dans le processus. Après quelques minutes, un message d'erreur apparaît.



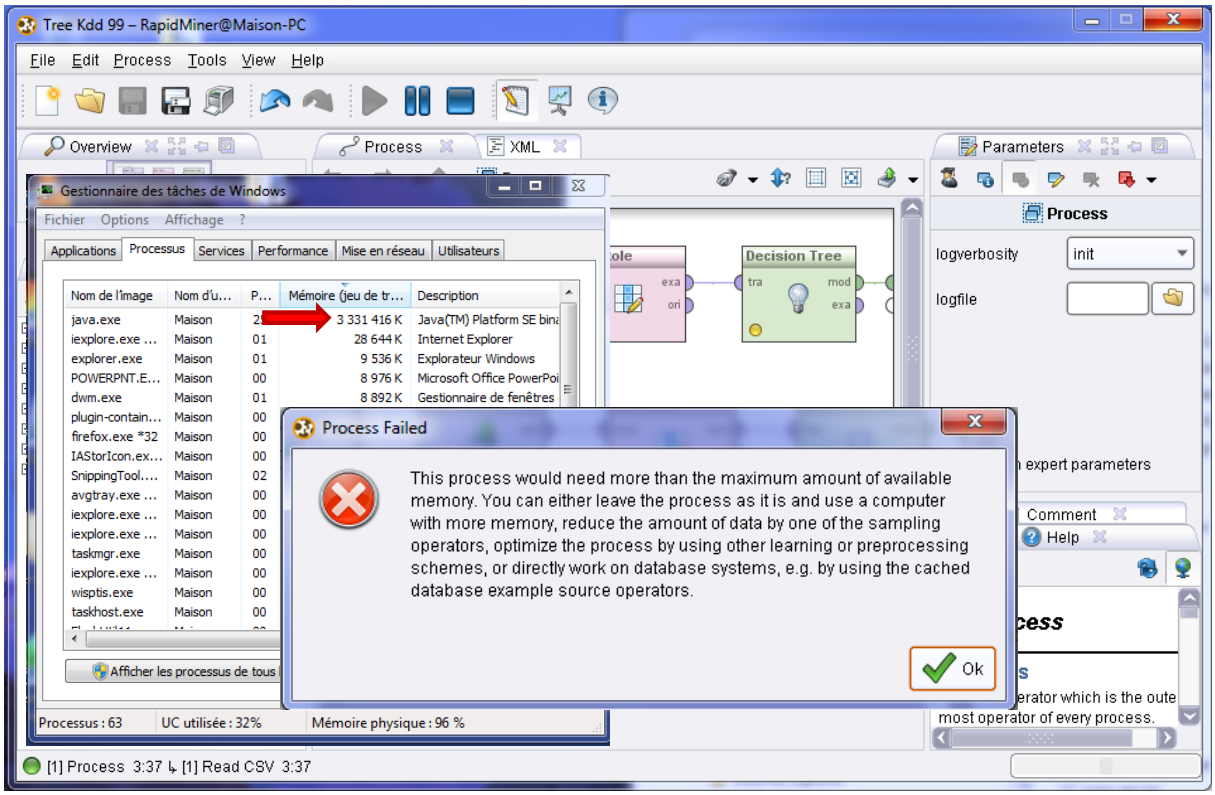
La mémoire disponible n'est pas suffisante pour réaliser les traitements. Nous constatons que RapidMiner occupe 2.76 Go en mémoire via JAVAW.EXE.

5.2 Lancement via la ligne de commande

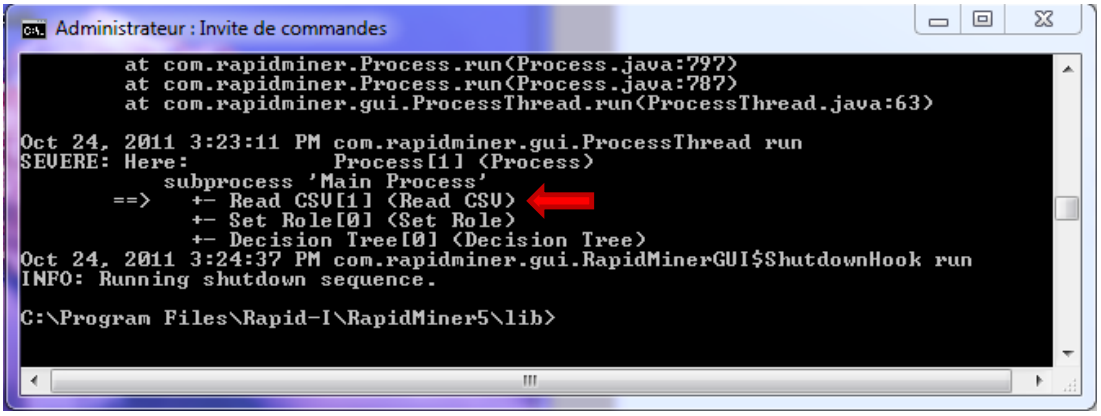
Nous pouvons modifier les fichiers de configurations situés dans le sous-répertoire SCRPITS pour augmenter la mémoire allouée au logiciel. Mes tentatives n'ont pas été très satisfaisantes. J'ai préféré lancer le logiciel via la ligne commande pour contrôler complètement les instructions envoyées au processus.



Après quelques minutes, l'opération échoue de nouveau.



Nous constatons que la mémoire allouée au logiciel est de 3.17 Go quand les calculs ont été interrompus. Un coup d'œil à la fenêtre de commande nous indique que le traitement ne dépasse pas le stade de l'importation des données.



RapidMiner reposant sur la même technologie que Knime, nul doute que si nous disposions de plus de mémoire, il pourrait mener à leur terme les calculs. Notons également qu'au lieu de nous laisser complètement démuni, RapidMiner nous propose judicieusement d'autres pistes pour aller au-delà de l'erreur : l'échantillonnage, l'utilisation d'autres techniques de data mining, ou travailler directement dans les systèmes de gestion de bases de données. Cette dernière piste me paraît particulièrement intéressante. Voilà un bon sujet pour un prochain tutoriel.

5.3 Codage interne des données

L'option **DATAMANAGEMENT** de l'opérateur **READ CSV**, accessible en mode expert, est très intéressante lors de la manipulation des grandes bases de données. Elle nous permet de choisir le mode de codage interne des valeurs. Par défaut, l'option **DOUBLE_ARRAY** est spécifiée. Chaque valeur est donc représentée par un réel en double précision, elle occupe 8 octets en mémoire centrale. Dans notre, cas où toutes les variables sont catégorielles avec un nombre de modalités réduit, ce choix est surdimensionné. Nous pouvons réduire considérablement l'occupation mémoire des données en adoptant un codage adapté. Si nous prenons par exemple l'option **BYTE_ARRAY**, amplement suffisante pour notre fichier, **nous divisons par 8** la taille de la base en mémoire !

6 Conclusion

Le passage aux 64 bits permet surtout de tirer parti des capacités des machines plus puissantes. Avec plus de mémoire vive, les nouveaux systèmes 64 bits peuvent sans aucun problème charger la totalité de la base et construire rapidement l'arbre exclusivement en mémoire[5]. Vu l'évolution des PC – je disposais de 4 Mo de RAM quand j'ai commencé à développer SIPINA, 4 Go aujourd'hui est tout

[5] **Mise à jour – 05/11/2011**. Peu de temps après la publication de cet article (le 25/10/2011), un internaute disposant d'une machine avec 8Go de RAM a réalisé l'expérimentation. Il a augmenté la taille maximale du tas (**-Xmx8192m**). Tant Knime que RapidMiner ont réussi à construire l'arbre de décision en faisant tenir la totalité de la base en mémoire. L'occupation mémoire mesurée était approximativement de 7 Go. C'était juste, mais le traitement est effectivement possible.

à fait banal – il est prévoir que les limitations quant aux bases que l'on peut appréhender avec Knime et RapidMiner seront rapidement levées dans les années à venir.

Pour l'heure néanmoins, les solutions de swap, comme celle implémentée dans SIPINA, restent d'actualité. Elles le sont d'autant plus qu'elles nous permettent de traiter de très grandes bases -- autrement plus importantes que celle décrite dans ce tutoriel – sur des machines aux caractéristiques modestes.