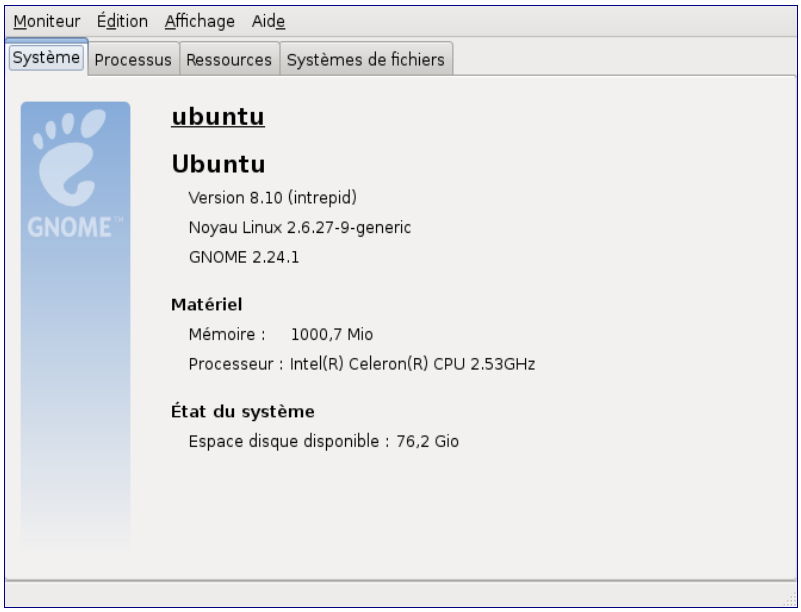
1 Introduction

Construire la courbe de gain avec plusieurs logiciels. Comparer les performances sous Linux sur un grand fichier.

La courbe de gain est un outil important du ciblage marketing. On le retrouve sous des terminologies différentes selon les logiciels (gain chart, courbe lift, lift chart, courbe lift cumulative, etc.). Mais l'idée est toujours la même : nous affectons un score à des individus, nous trions la base selon un score décroissant, nous élaborons alors une graphique nuage de points avec, en abscisse, la proportion des individus dans la cible (les x premiers – en pourcentage - dans la base triée selon le score), et en ordonnée, la fraction des positifs que l'on y retrouve. Le dernier point est de coordonnée (100%, 100%) : lorsque tous les individus sont inclus dans la cible, nous sommes sûrs de retrouver tous les positifs.

L'élaboration de la courbe de gains dans Tanagra est décrite par ailleurs (http://tutoriels-data-mining.blogspot.com/2008/03/ciblage-marketing-scoring-coil.html). Notre idée dans ce didacticiel est d'élargir la description aux autres logiciels libres (Knime, RapidMiner et Weka). La seconde originalité de cette étude est que nous réalisons toutes les opérations sous Linux (distribution Ubuntu 8.10). Nous constaterons que Tanagra, tout comme les logiciels sus-cités, fonctionnent parfaitement. Cela nous amène à la troisième originalité de ce travail, nous traitons un fichier d'une taille importante avec **2.000.000 d'observations et 41 variables**. Nous pourrons évaluer la tenue de ces logiciels lorsqu'on les place dans des situations extrêmes, de surcroît sur une machine très peu performante dont voici les caractéristiques.



Nous adopterons la même démarche pour chaque logiciel. Dans un premier temps, nous traitons un échantillon de 2.000 observations, nous pouvons ainsi paramétrer à notre aise les calculs et obtenir au moins une fois un résultat que l'on peut montrer. Dans un second temps, nous modifions la source de données pour traiter le fichier complet. Nous

mesurons alors le temps d'exécution, nous mesurons également l'occupation mémoire à l'issue de tous les traitements. **Nous constaterons que certains logiciels ne pourront pas mener à leur terme les calculs**.

Enfin, concernant l'algorithme d'apprentissage, nous nous cantonnerons aux méthodes linéaires simples qui, bien souvent, donnent entièrement satisfaction dans le scoring. Pour Tanagra, nous utiliserons l'analyse discriminante linéaire, précédée d'une sélection de variables. Pour les autres logiciels, cette approche n'étant pas disponible, nous utiliserons le modèle d'indépendance conditionnelle (Naive Bayes classifier), sans sélection de variables. Les calculs étant plus simples, ce choix devrait avantager ces logiciels, au moins en termes de rapidité. Nous vérifierons cela.

2 Données

Notre fichier a pour origine la compétition KDD-CUP de 1999. Il s'agit de prédire ou d'expliquer le type d'intrusion sur un serveur à partir des caractéristiques de la connexion (http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html).

Nous avons un peu modifié la nature des données : la variable à prédire est rendue binaire (connexion « normale » ou « autre ») ; tous les descripteurs ont également été codées en variable 0/1. La question que l'on se pose est la suivante : « si on ne laisse entrer que les 20% des connexions, celles ayant un score le plus élevé, quelle serait la proportion des connexions normales qui passeraient ? ».

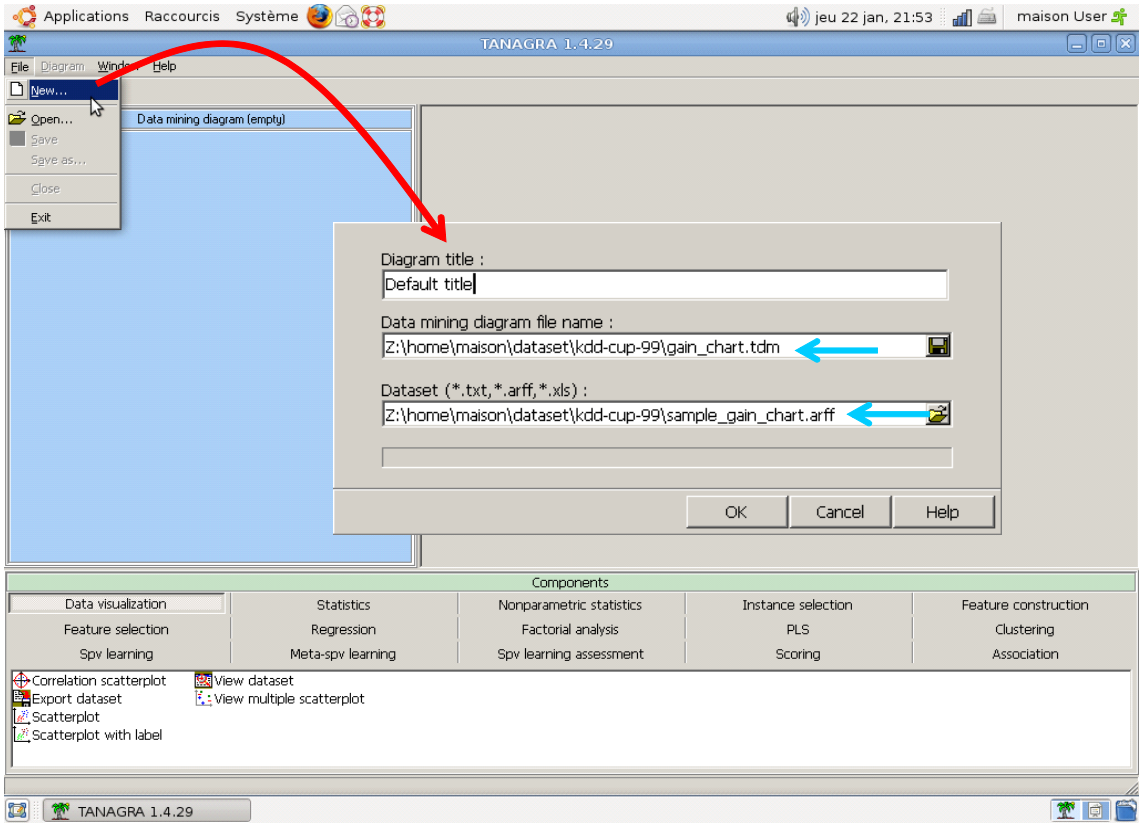
Nous manipulerons 2 fichiers au format WEKA (ARFF). Le premier comporte 2.000.000 d'observations (full_gain_chart.arff), il servira à évaluer les performances des logiciels ; le second est un échantillon de 2.000 lignes (sample_gain_chart.arff), il nous servira à définir et à paramétrer les traitements. Ces deux fichiers sont réunis dans l'archive http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/dataset_gain_chart.zip

3 Tanagra

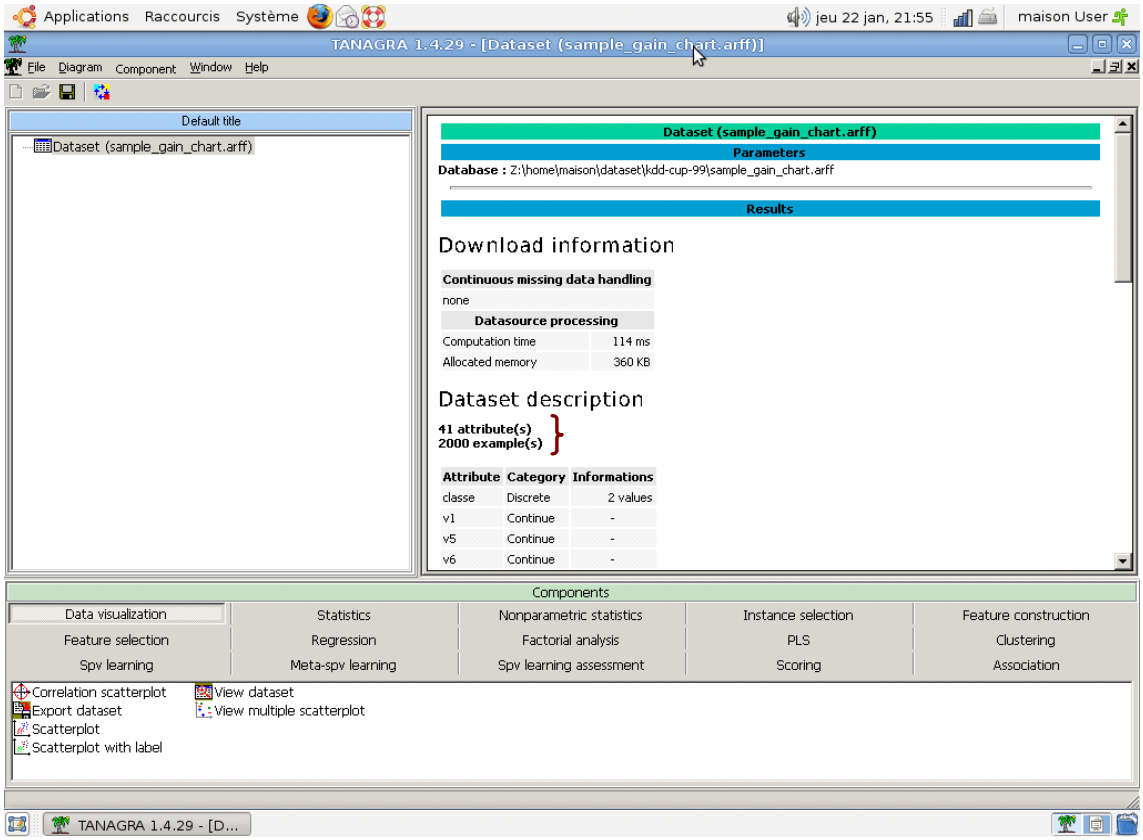
3.1 Paramétrer les traitements

Création d'un diagramme et importation des données. Après avoir démarré Tanagra[1], nous actionnons le menu FILE / NEW pour créer un diagramme et importer les données. Dans la boîte de paramétrage, nous sélectionnons le fichier SAMPLE_GAIN_CHART.ARFF au format Weka, nous nommons le diagramme « GAIN_CHART.TDM ».

1 Voir http://tutoriels-data-mining.blogspot.com/2009/01/tanagra-sous-linux.html pour l'installation et l'utilisation de Tanagra sous Linux.

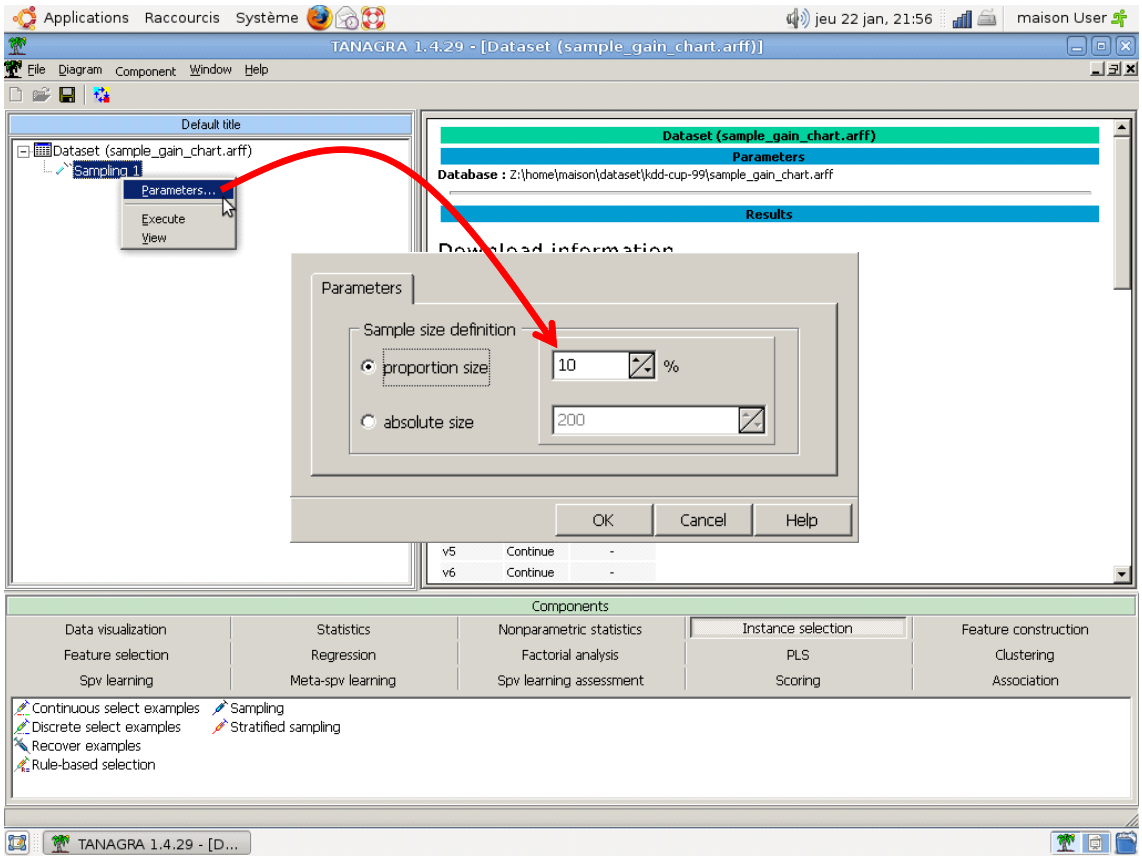


Tanagra nous indique que le fichier comporte 41 variables et 2.000 observations.



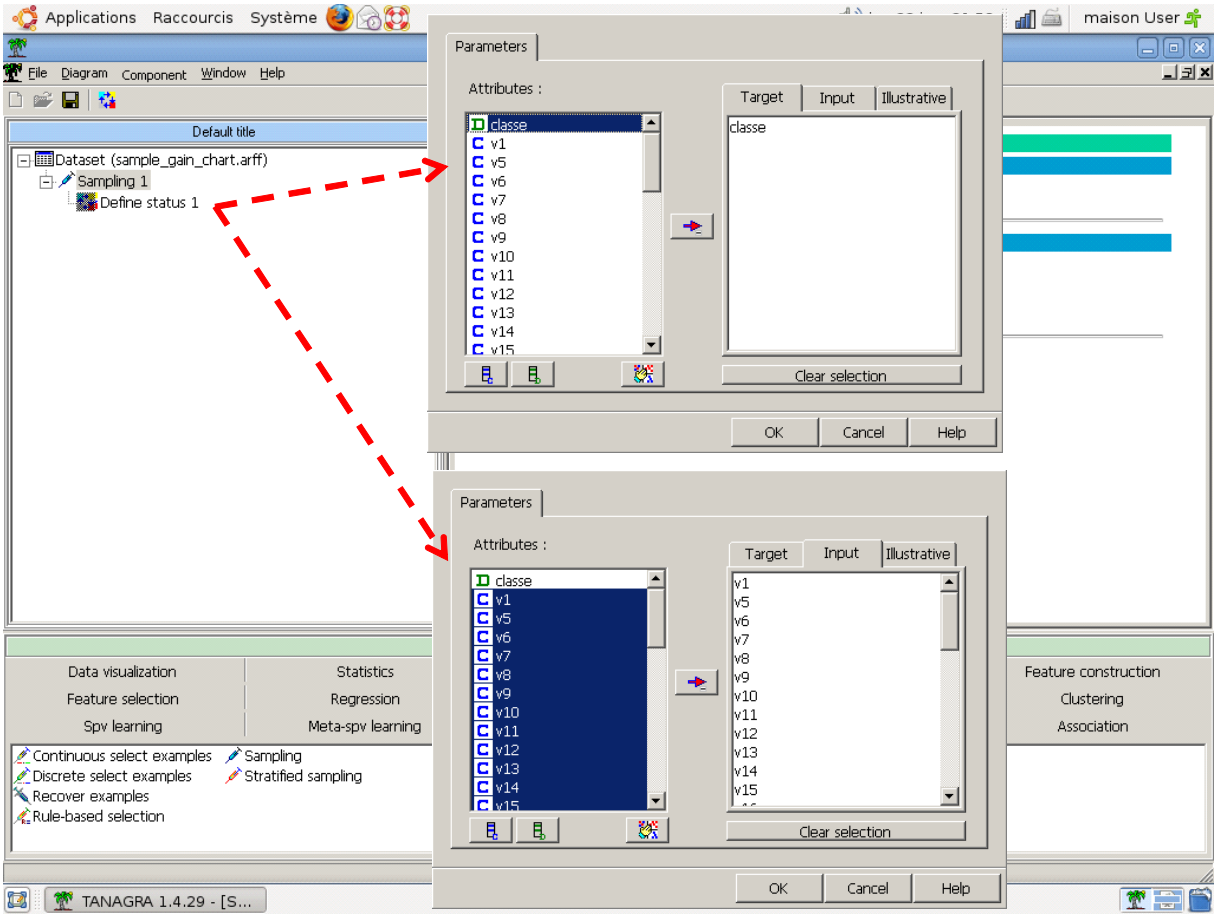
Subdivision des données en apprentissage et test. Nous souhaitons construire le modèle de prédiction sur 10% des individus, puis le valider sur le reste. Le pourcentage paraît faible. Mais n'oublions que l'objectif de tout ceci est de lancer les calculs sur la base comportant 2.000.000 d'observations. Dans ce cas, 10% correspond à 200.000 individus. C'est amplement suffisant pour construire un modèle solide.

Pour réaliser l'échantillonnage, nous insérons le composant SAMPLING (onglet INSTANCE SELECTION). Via le menu contextuel PARAMETERS, nous choisissons une proportion de 10%.

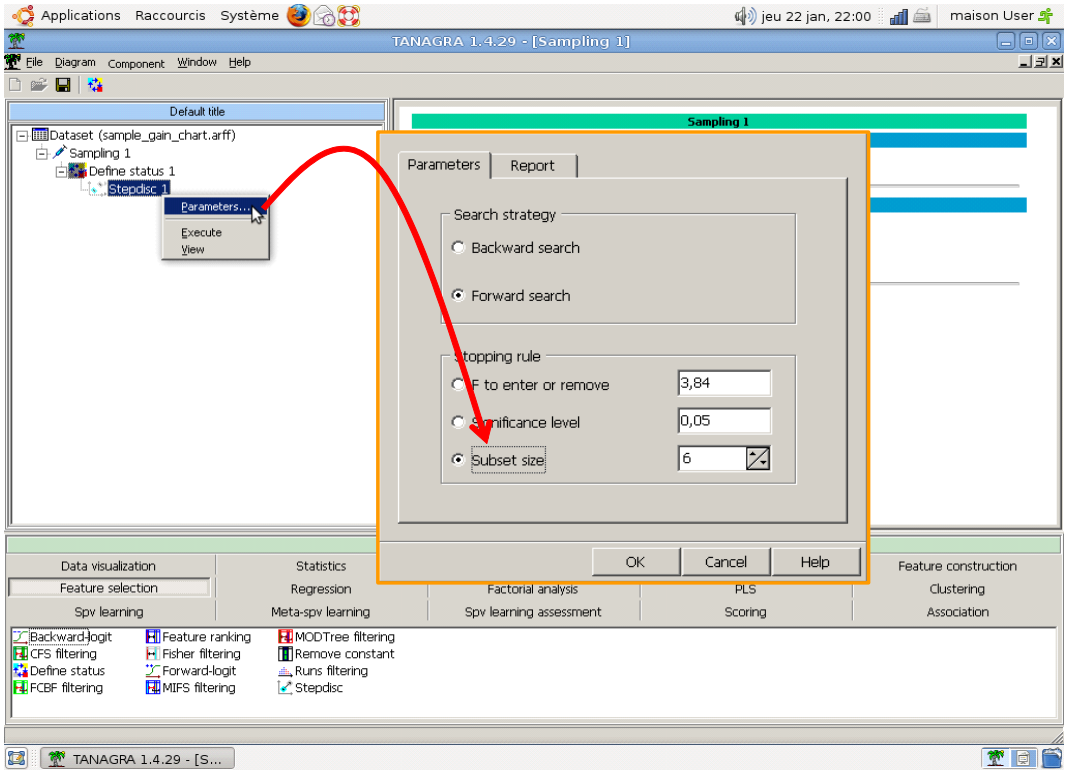


Nous actionnons le menu contextuel VIEW, Tanagra indique que 200 observations sont maintenant réservées à l'apprentissage.

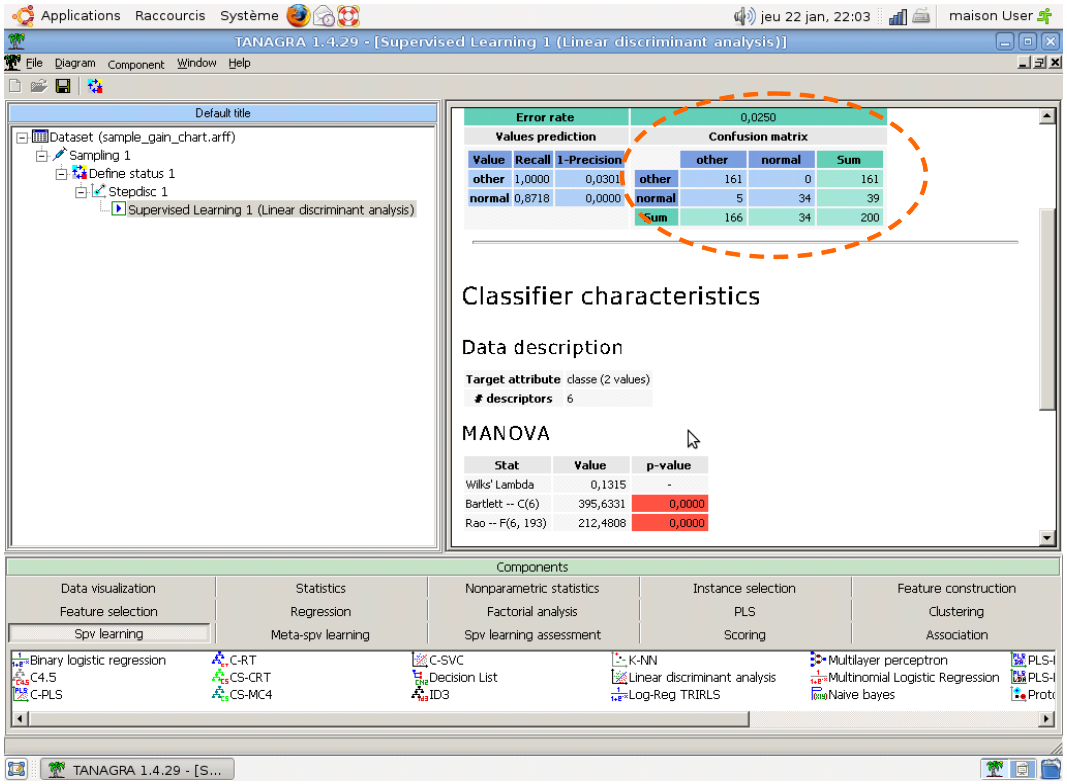
Définition du problème. Nous devons maintenant indiquer à Tanagra la variable à prédire (CLASSE) et les variables prédictives (les autres). Nous insérons le composant DEFINE STATUS en utilisant le raccourci dans la barre d'icônes. Nous plaçons CLASSE en TARGET, toutes les autres variables en INPUT.



Sélection automatique des variables. Certaines des variables prédictives sont redondantes, d'autres ne sont pas pertinentes. Nous introduisons le composant STEPDISC (onglet FEATURE SELECTION). Nous le paramétrons via le menu contextuel PARAMETERS. Il est totalement illusoire de fixer une règle d'arrêt basée sur le risque critique dans notre cas. En effet, lorsque l'apprentissage sera réalisé sur la base complète, la démultiplication du nombre d'observations rendra significatif toutes les variables. Il nous faut fixer une règle d'arrêt ad hoc. Après quelques tâtonnements, et en se basant principalement sur la décroissance du lambda de Wilks, nous choisissons les 6 meilleures variables c.-à-d. SUBSET SIZE = 6.

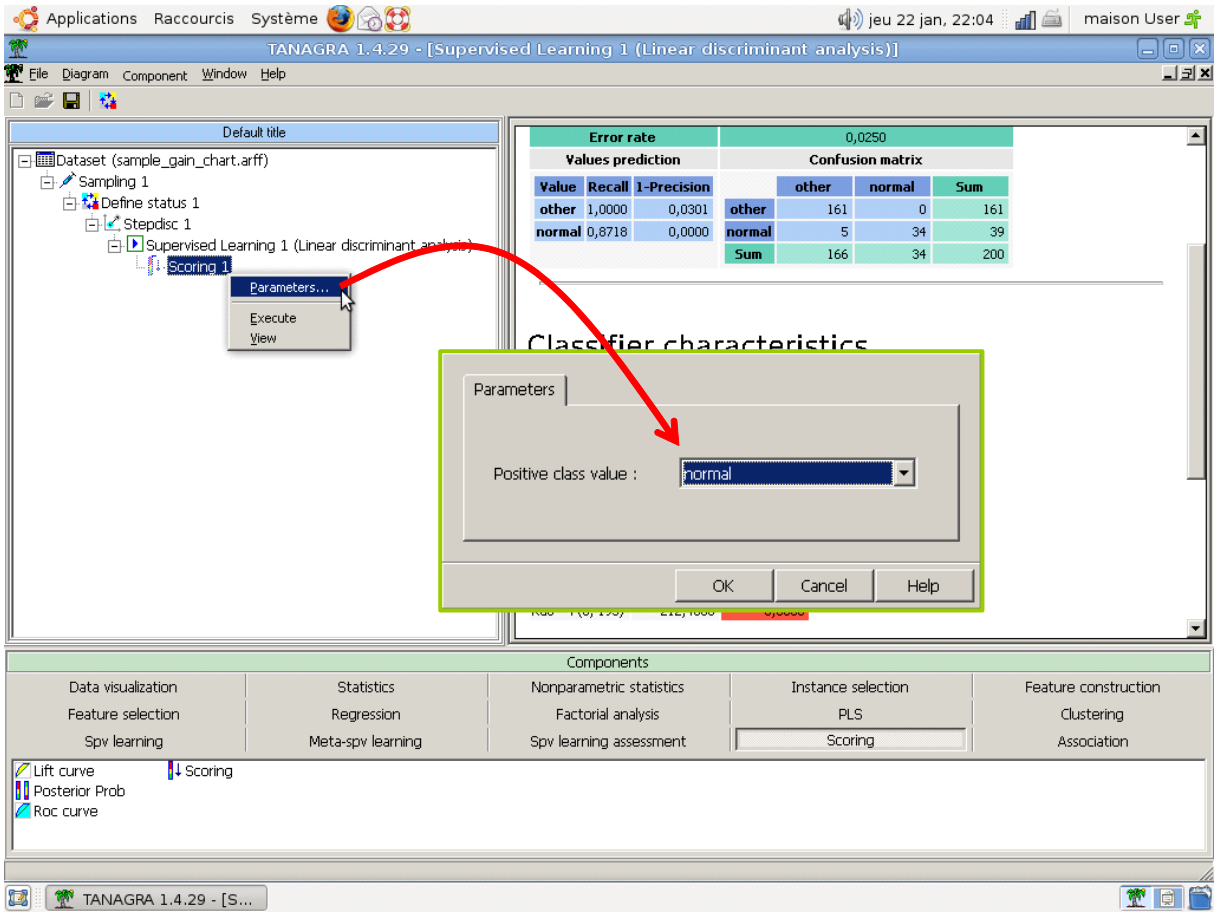


Apprentissage. Il ne nous reste plus qu'à brancher et à lancer le composant LINEAR DISCRIMINANT ANALYSIS (onglet SPV LEARNING). Nous obtenons le résultat suivant. La prédiction semble d'excellente qualité si l'on se réfère à la matrice de confusion.



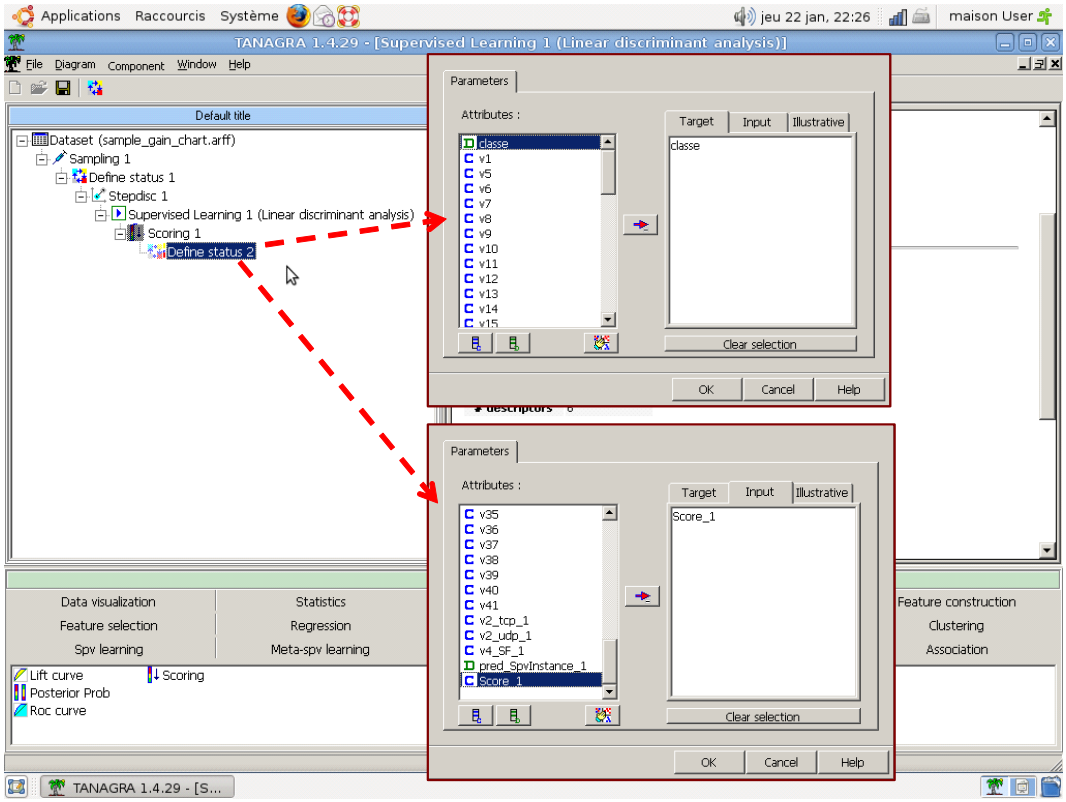
Création de la colonne score. Nous passons à la seconde phase maintenant, celle de l'évaluation sur l'échantillon test. Pour cela, nous devons tout d'abord calculer le score « positif » de toutes les observations c.-à-d. « la propension d'une connexion à être normale ».

Nous introduisons le composant SCORING (onglet SCORING). Nous le paramétrons : nous souhaitons calculer la probabilité (ou tout du moins le score) d'une connexion d'être normale.

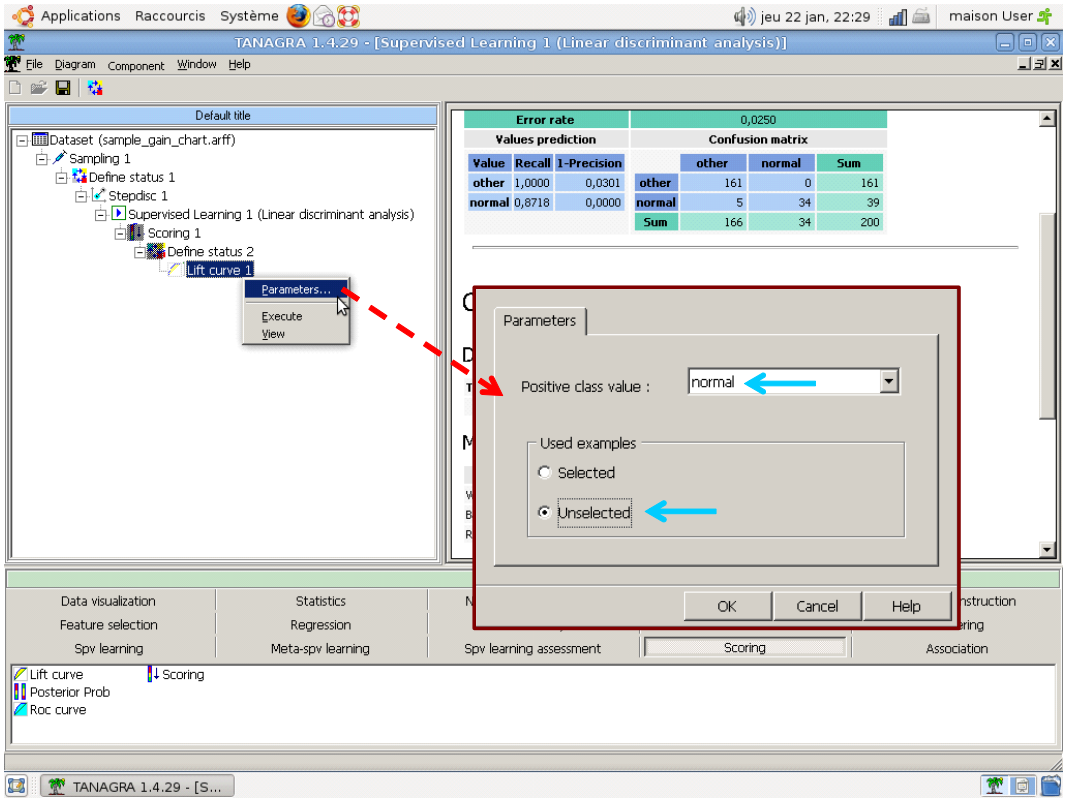


Une nouvelle colonne nommée SCORE_1 est ajoutée à l'ensemble de données.

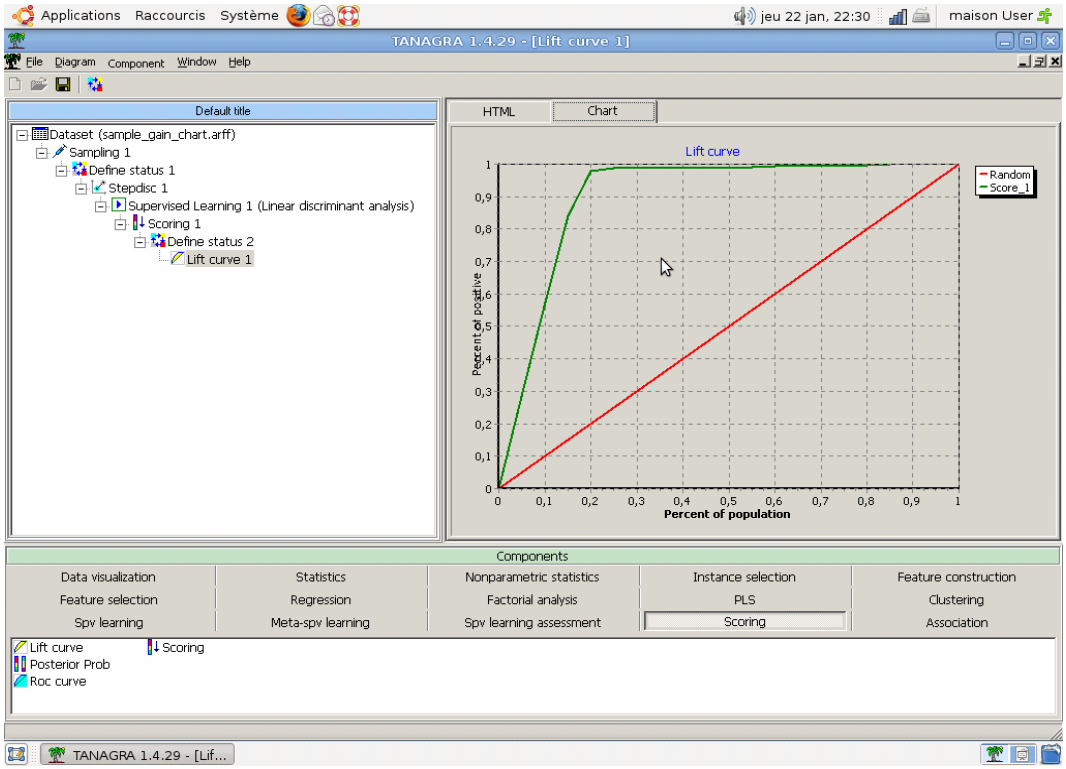
Construction de la courbe de gain. Il ne nous reste plus qu'à construire la courbe de gain. Nous devons tout d'abord spécifier la variable à prédire (TARGET = CLASSE) et la colonne score (INPUT = SCORE_1) pour l'élaboration du graphique. Nous utilisons pour cela le composant DEFINE STATUS.



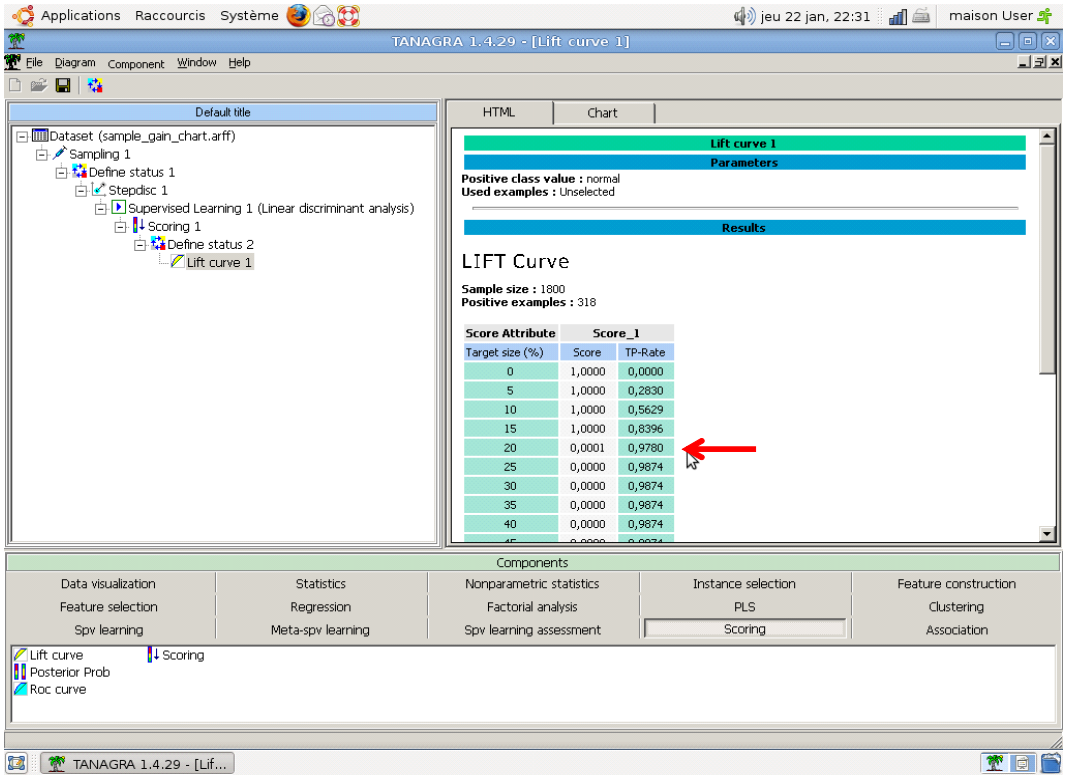
Puis nous insérons le composant LIFT CURVE (onglet SCORING). Nous le paramétrons pour que la modalité « normale » corresponde aux « positifs », la courbe doit être élaborée sur l'échantillon test.



Nous obtenons le graphique suivant.



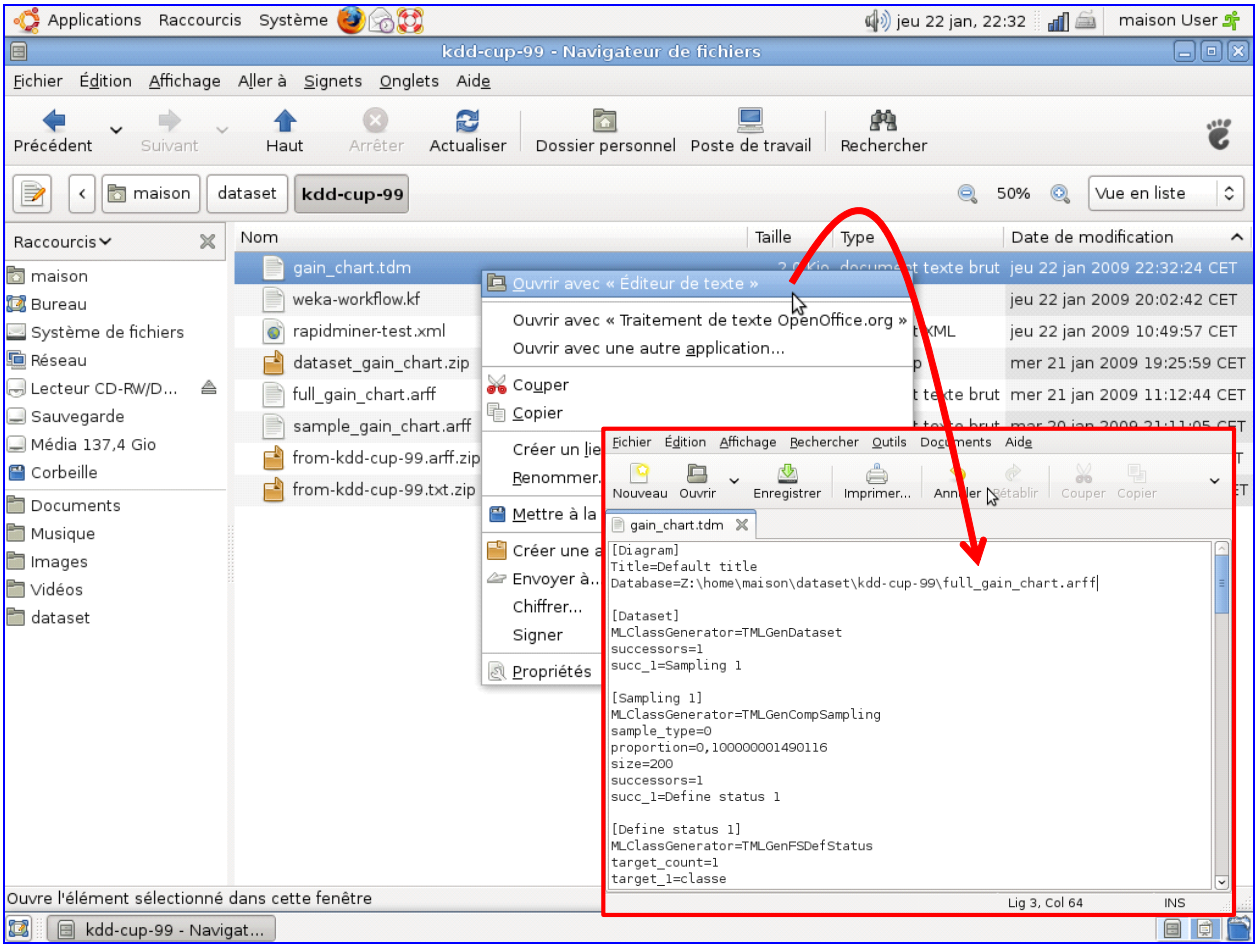
Si on se réfère au tableau de calcul (onglet HTML), dans les 20% première observations classés selon un score décroissant, nous observons 97.80% des positifs (c'est le taux de vrais positifs).



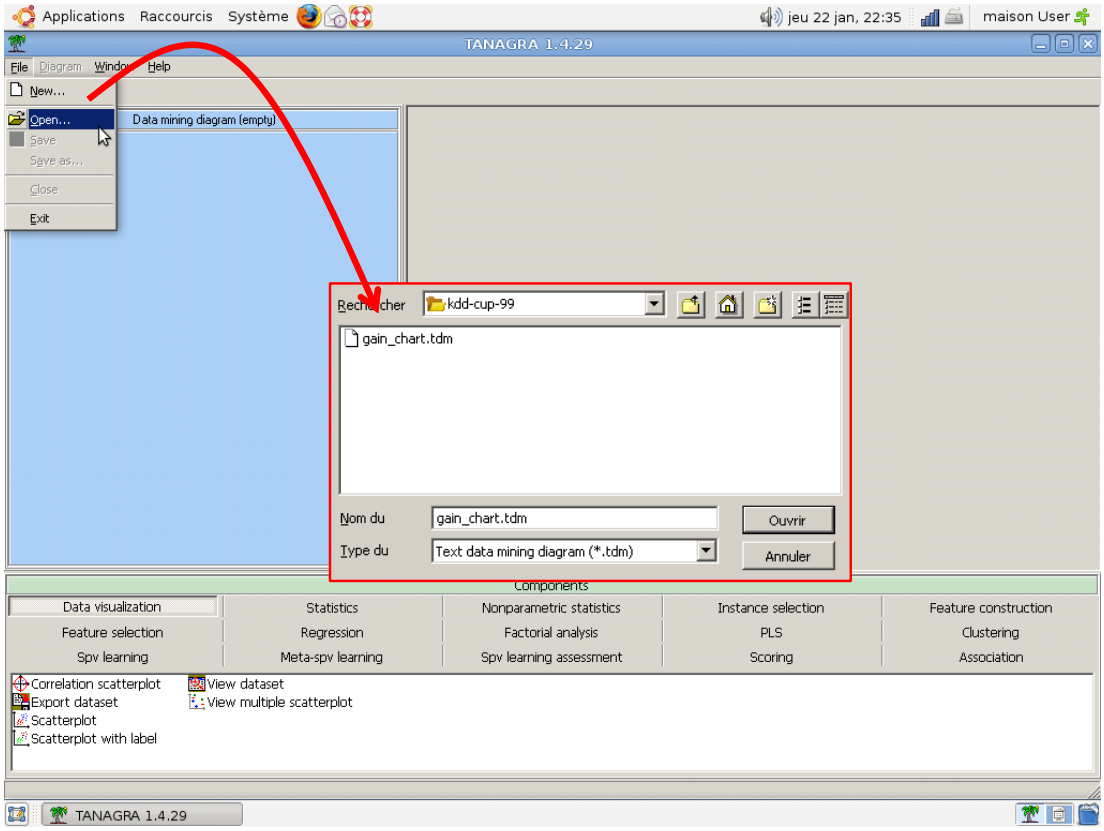
3.2 Calcul sur la base complète

Cette première étape est nécessaire pour définir les calculs, affiner les paramétrages, bref préparer le terrain. Il faut maintenant passer au traitement de la totalité de la base pour obtenir le modèle que l'on déploiera effectivement dans la population. Il sera forcément meilleur car calculé sur un effectif plus grand (200.000), il sera aussi plus stable. De la même manière, nous obtiendrons une évaluation plus précise des performances en calculant la courbe lift sur 1.800.000 observations.

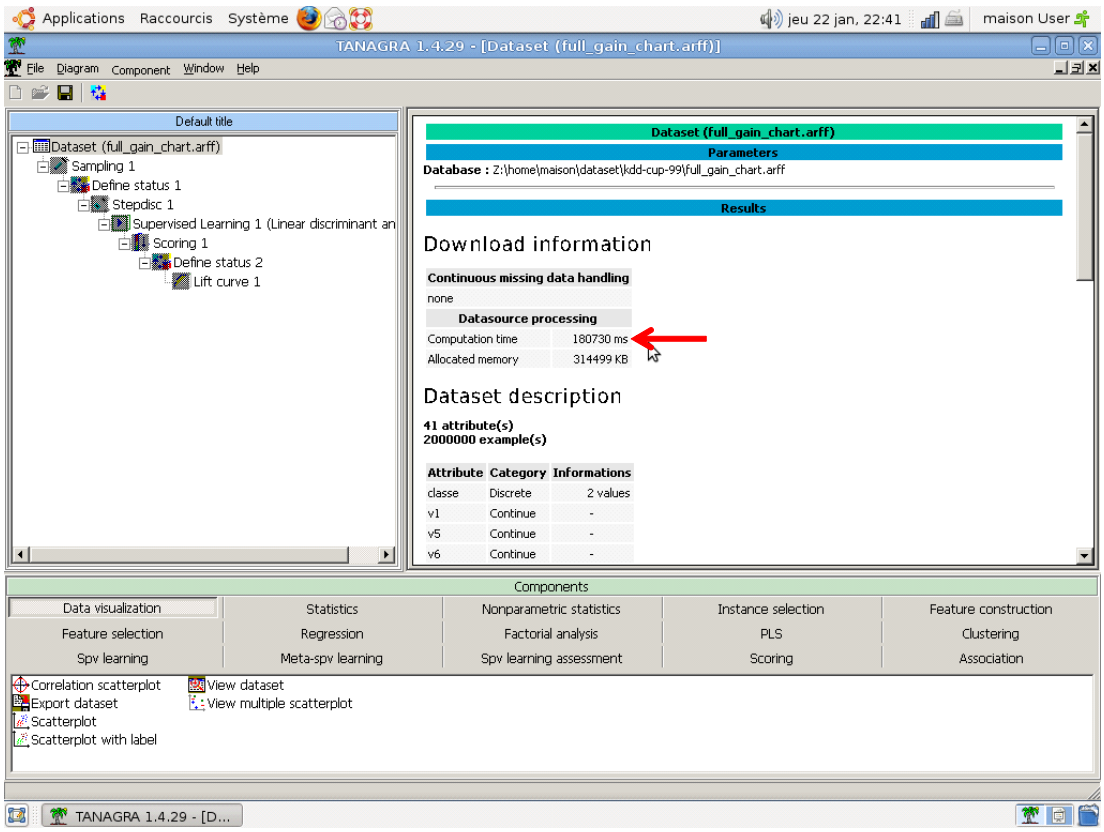
Pour Tanagra, le plus simple est d'enregistrer le diagramme (FILE / SAVE), de fermer l'application et d'aller dans le gestionnaire de fichier NAUTILUS de UBUNTU. Dans le répertoire de travail, nous repérons le fichier du diagramme « gain_chart.tdm » que nous ouvrons à l'aide d'un éditeur de texte. Le passage à une autre base, toutes choses égales par ailleurs, est très simple : nous modifions la source de données en inscrivant le nom de la base complète « full_gain_chart.arff ».



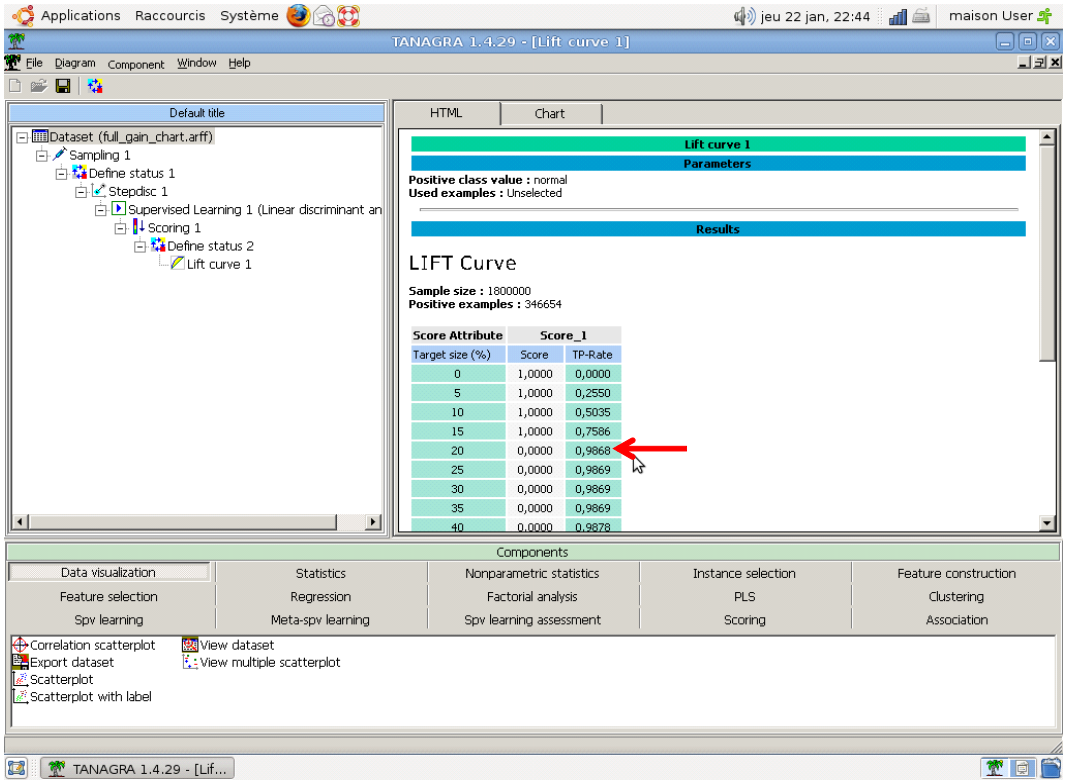
Il ne nous reste plus qu'à relancer Tanagra et à ouvrir le diagramme avec FILE/OPEN. Nous choisissons le fichier « gain_chart.tdm ».



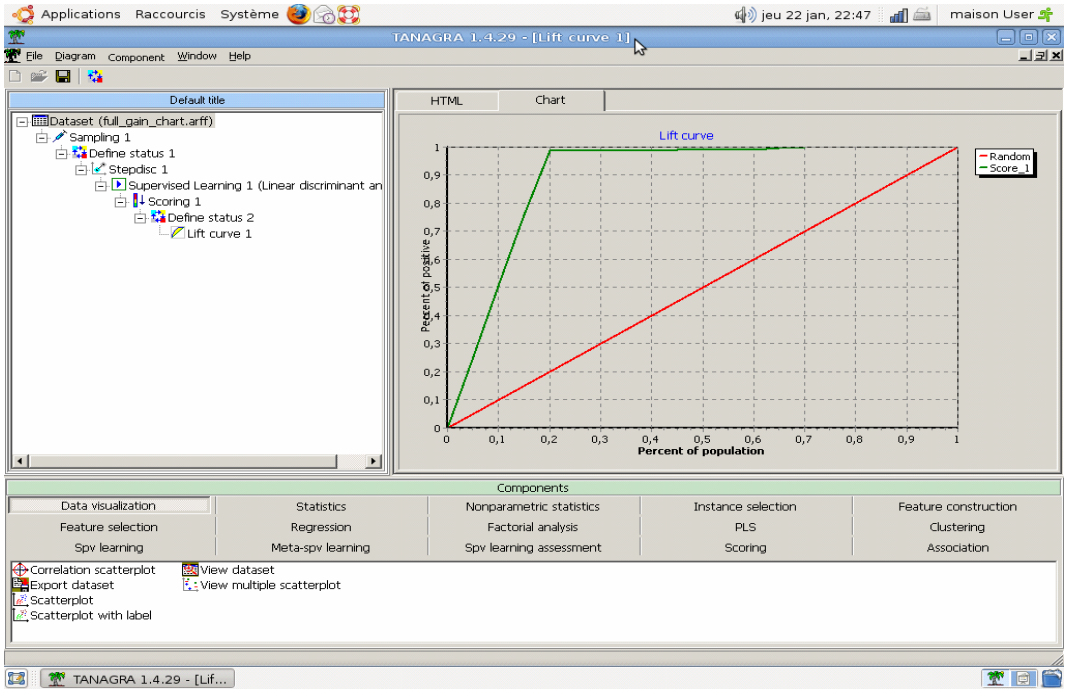
Le chargement de la base dure 180 secondes (3 minutes).



Nous relançons l'ensemble des traitements via le menu DIAGRAM / EXECUTE. Nous obtenons le tableau associé au tracé de la courbe de gain. Parmi les 20% premiers individus scorés, nous retrouvons 98,68% des positifs.

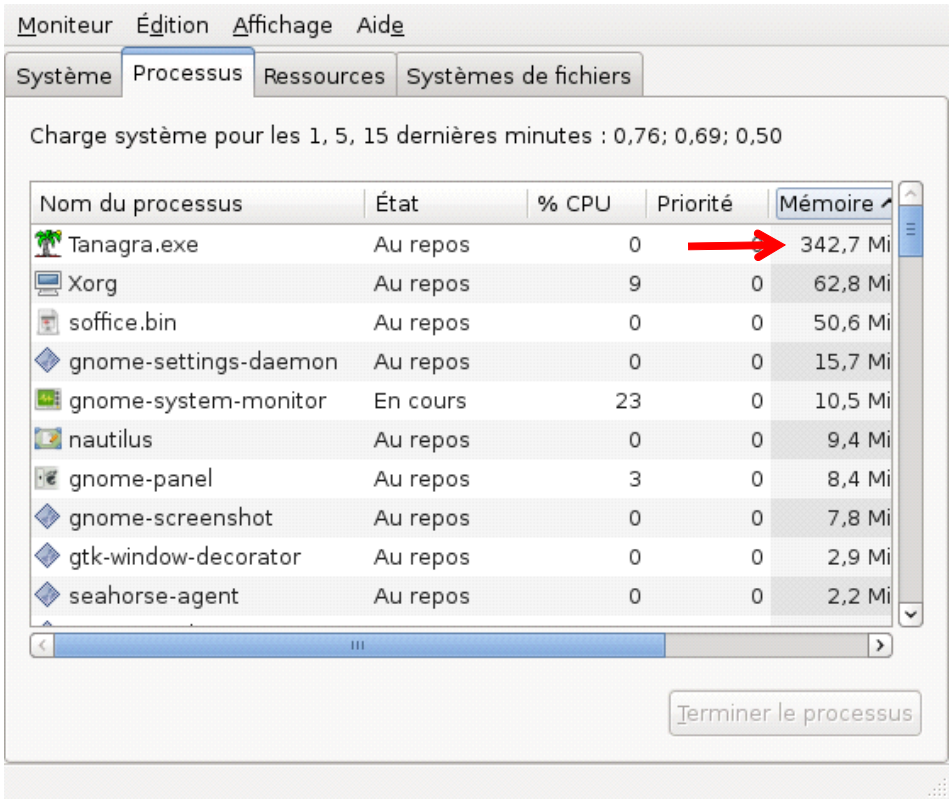


Soit la courbe.



Nous reviendrons plus loin sur la durée des traitements à chaque étape. Nous constatons en tous les cas que

l'occupation mémoire de Tanagra augmente peu avec les calculs. Ce sont les données, toutes chargées en mémoire, qui prennent de la place.

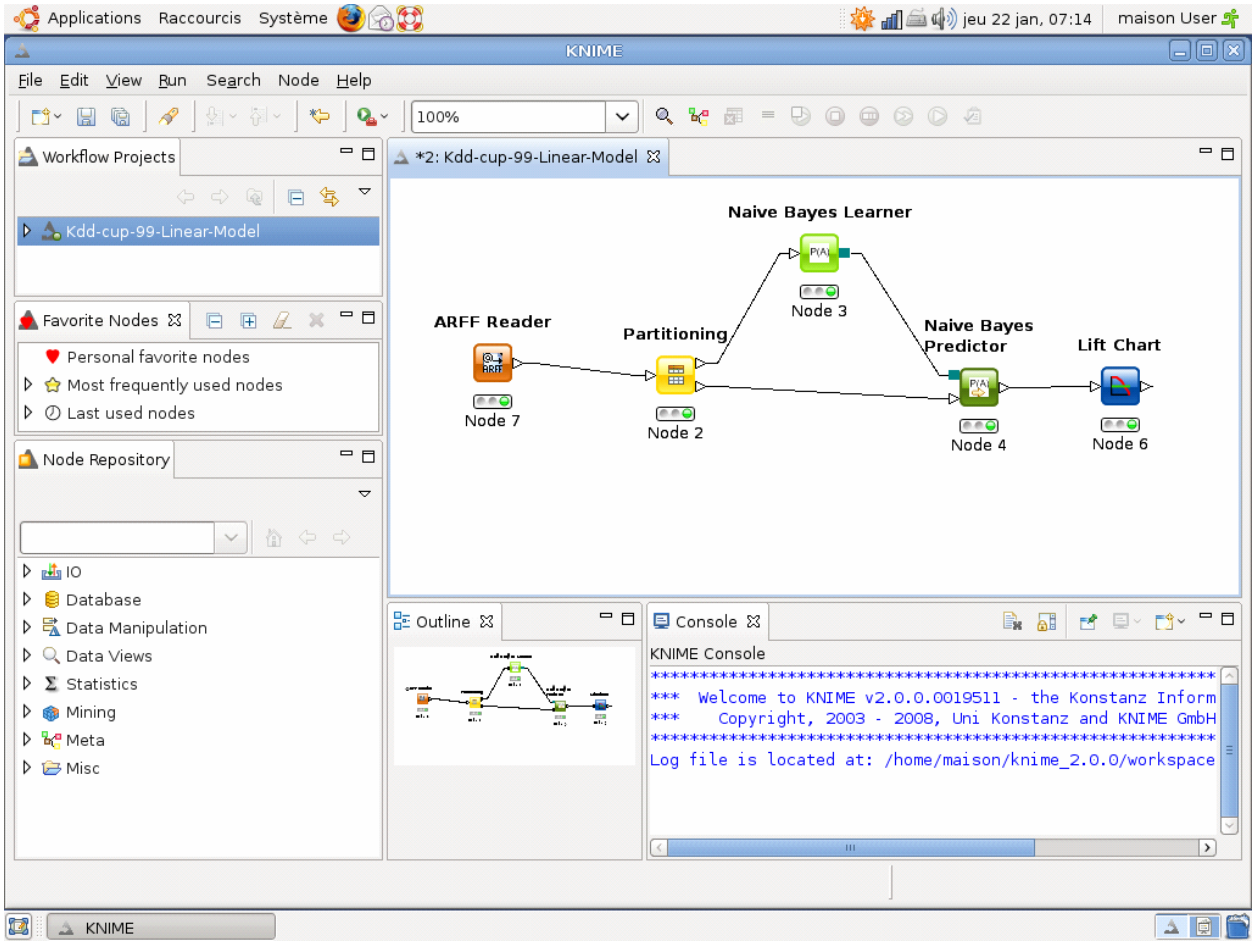


4 Knime

4.1 Paramétrage et exécution sur l'échantillon

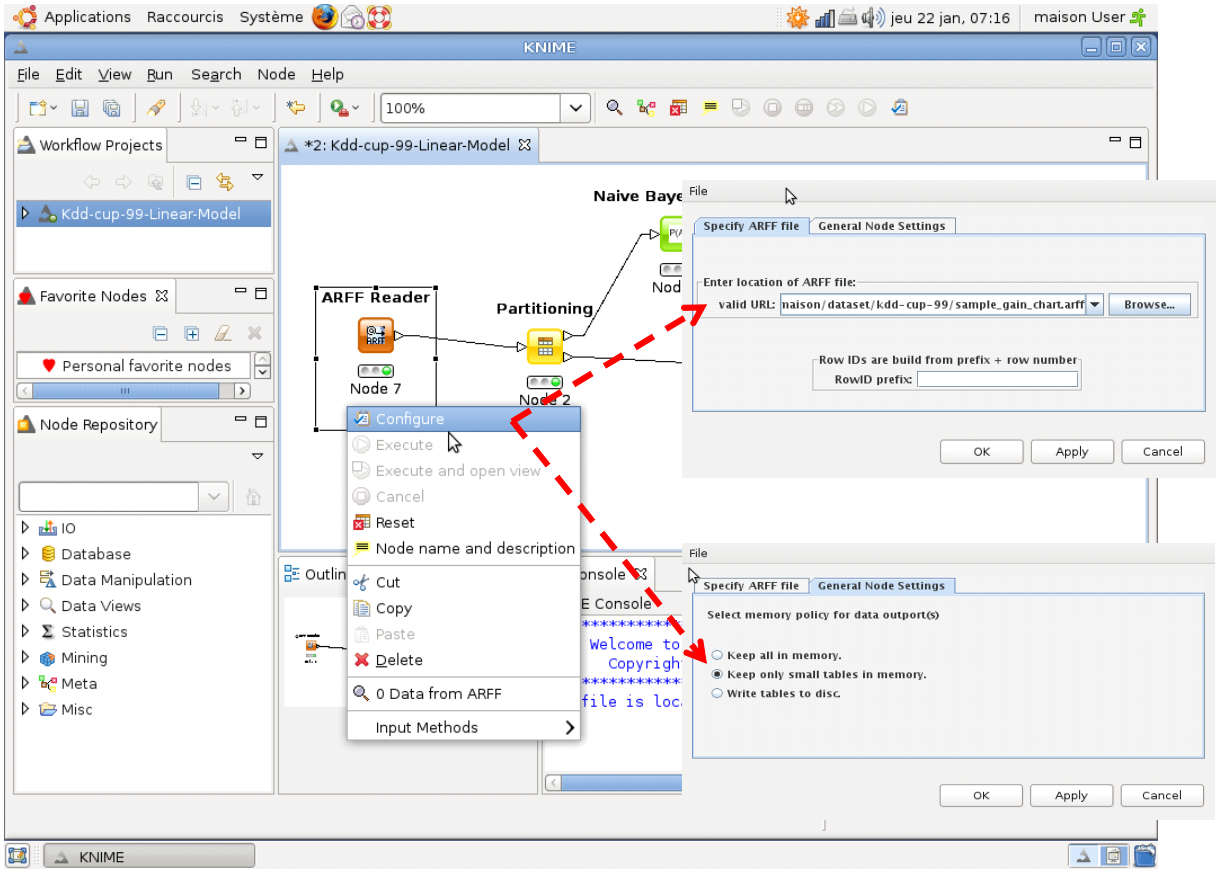
L'installation et la mise en route de Knime (http://www.knime.org/, version 2.0.0) ne posent aucun problème sous Ubuntu. Nous créons un nouveau diagramme (« workflow » dans la terminologie Knime) avec FILE / NEW. Nous demandons un « Projet Knime ».

Comme nous l'indiquions plus haut, il n'y a pas d'Analyse Discriminante sous Knime, nous nous sommes donc rabattus sur le Modèle d'Indépendance Conditionnelle qui produit aussi un séparateur linéaire. C'est assez facile à montrer, surtout lorsque nous avons des descripteurs 0/1 comme chez le cas dans notre fichier (voir http://fr.wikipedia.org/wiki/Classifieur_linéaire). Nous n'avons pas trouvé un outil de sélection automatique de variable adaptée à cette approche dans Knime, nous utilisons donc toutes les variables dans la production du classifieur.

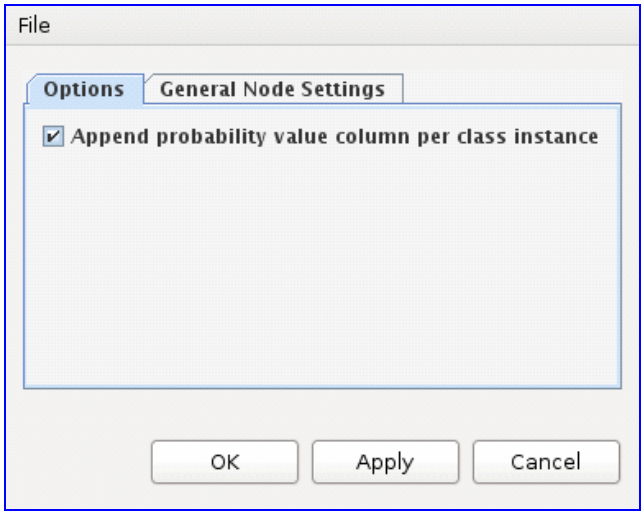


Enumérons les paramètres importants sur certains des composants :

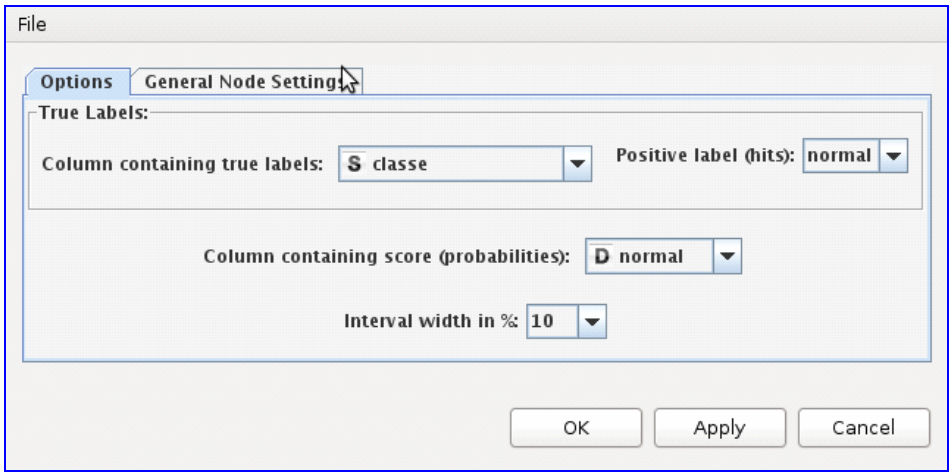
- Avec ARFF READER (menu CONFIGURE), nous spécifions le nom de fichier « sample_gain_chart.arff ». Ce composant intègre surtout une option très importante dans l'onglet « General Node Settings », si la taille de la base est trop importante (supérieure à 100.000 observations selon la documentation), elle est swappée sur le disque. C'est un avantage indéniable, nous verrons les conséquences de ce choix technique lorsque nous aurons à traiter la base complète.



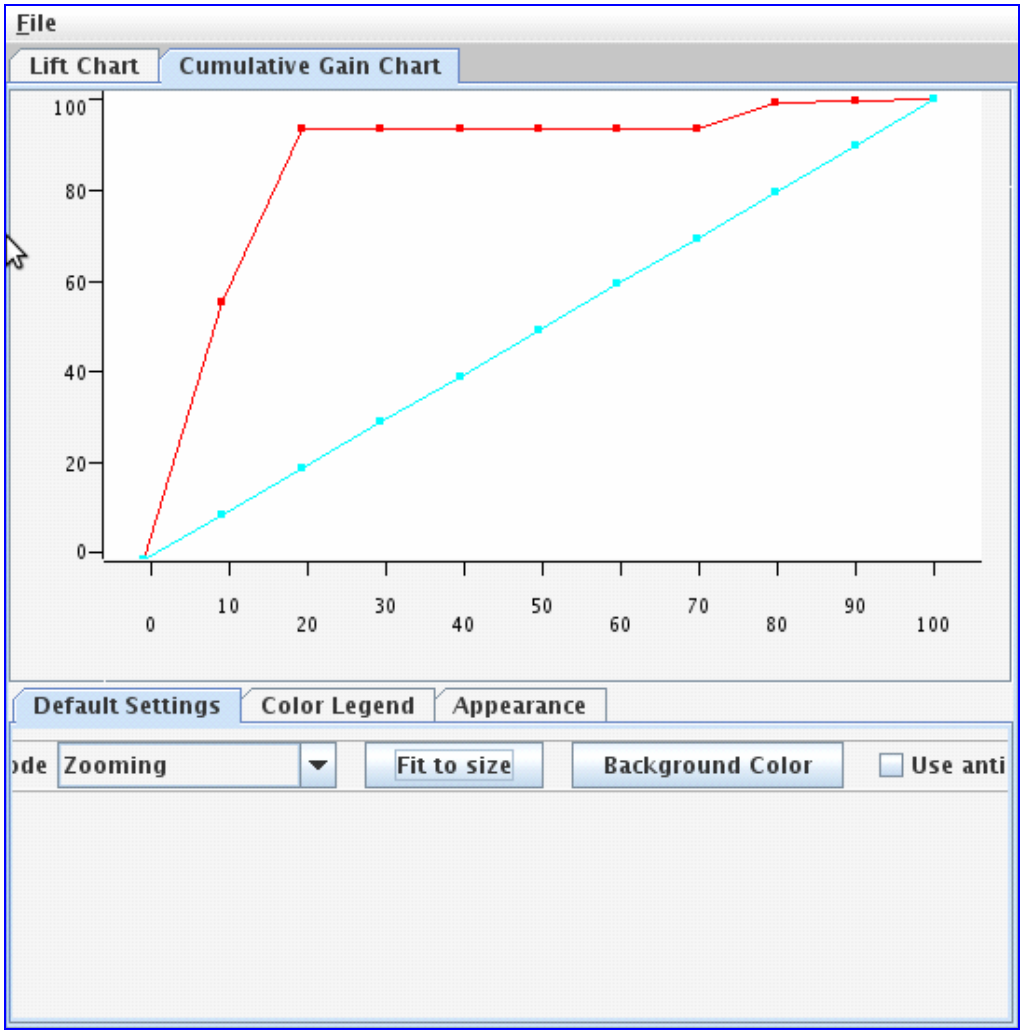
- Dans PARTITIONNING, nous sélectionnons 10% des observations.
- Dans NAIVE BAYES PREDICTOR, nous demandons à ce que le composant produise la colonne score (probabilité conditionnelle) pour chaque modalité de la variable à prédire.



- Enfin, dans le LIFT CHART, nous indiquons que la variable à prédire est CLASSE, que la modalité positive est NORMAL. Nous souhaitons créer un tableau de résultats où les taux de vrais positifs sont calculés pour des intervalles augmentant de 10% (INTERVAL WIDTH).

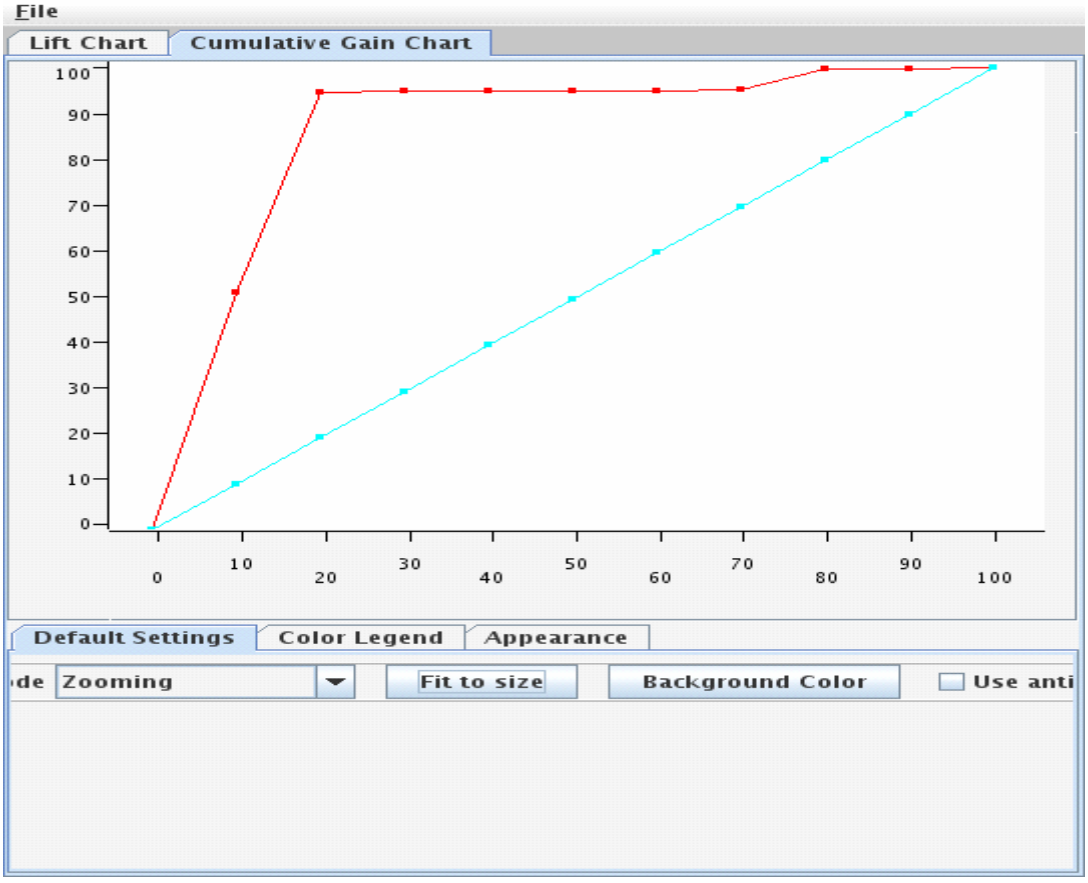


En cliquant sur le menu contextuel EXECUTE AND OPEN VIEW, nous obtenons la courbe de gain suivante dans l'onglet CUMULATIVE GAIN CHART.



4.2 Calcul sur la base complète

Nous sommes prêts à travailler sur la totalité de la base. Nous revenons sur le composant ARFF READER et nous choisissons la base « full_gain_chart.arff » (menu CONFIGURE). Nous relançons la séquence de calculs. Nous obtenons la courbe de gain suivante.



Elle est très proche finalement de celle construite sur l'échantillon de 2000 observations. Les temps de calcul en revanche ne sont absolument pas les mêmes. Ils sont plus importants, largement plus élevés que ceux de Tanagra. Nous les reprenons à la fin de ce document. Pour l'heure, penchons nous un peu plus sur l'occupation mémoire. A l'aide du moniteur système, nous constatons que Knime occupe 193 Mo (Figure 1). C'est nettement moins par rapport à Tanagra.

On voit bien ici les avantages et inconvénients de swapper les données sur le disque : l'occupation mémoire est maîtrisée, mais au prix d'un temps de calcul décuplé. Si nous sommes dans la phase de production du modèle final, cela ne devrait pas trop poser de problèmes. Si nous sommes dans la phase exploratoire, où l'on tâtonne pour définir les meilleures stratégies, un temps de calcul exagéré est dissuasif.

La seule solution possible est alors de travailler sur échantillon comme nous le faisons dans ce didacticiel. Encore faut-il savoir définir une taille d'échantillon suffisante. C'est loin d'être évident. Qui aurait pu croire dans notre cas que travailler sur un échantillon de 2.000 observations, dont on extrait 200 pour l'apprentissage du modèle prédictif,

permet d'obtenir des résultats similaires au travail sur une base de 2.000.000 d'individus ?

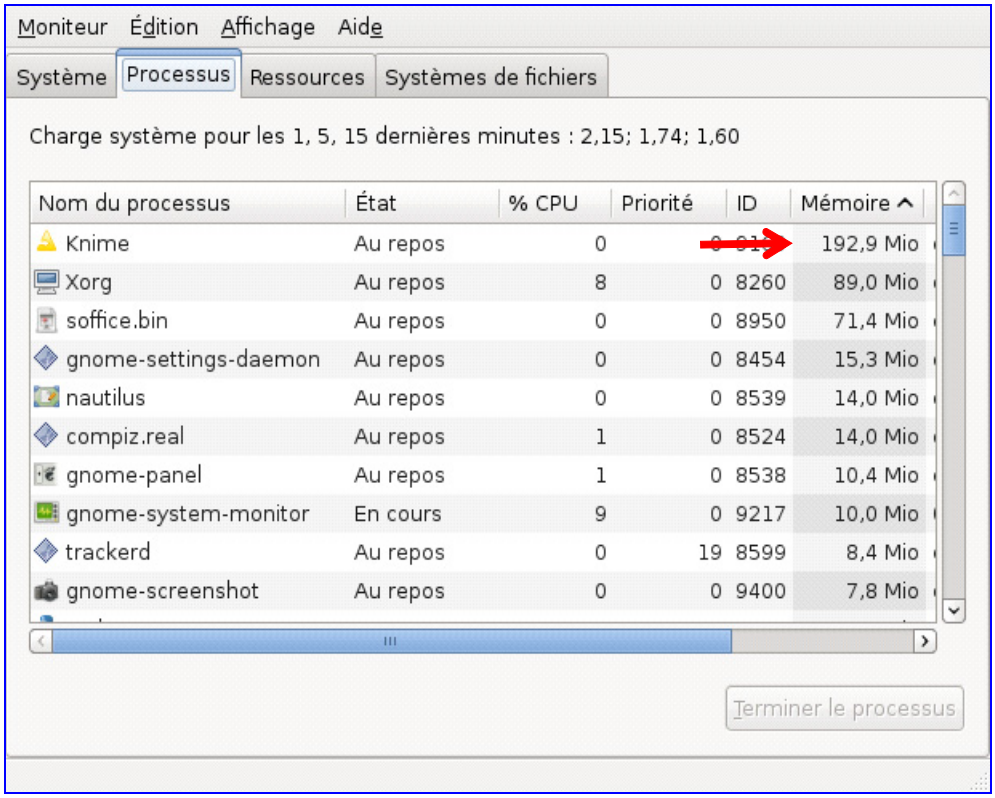


Figure 1 - Occupation mémoire de Knime à l'issue des traitements

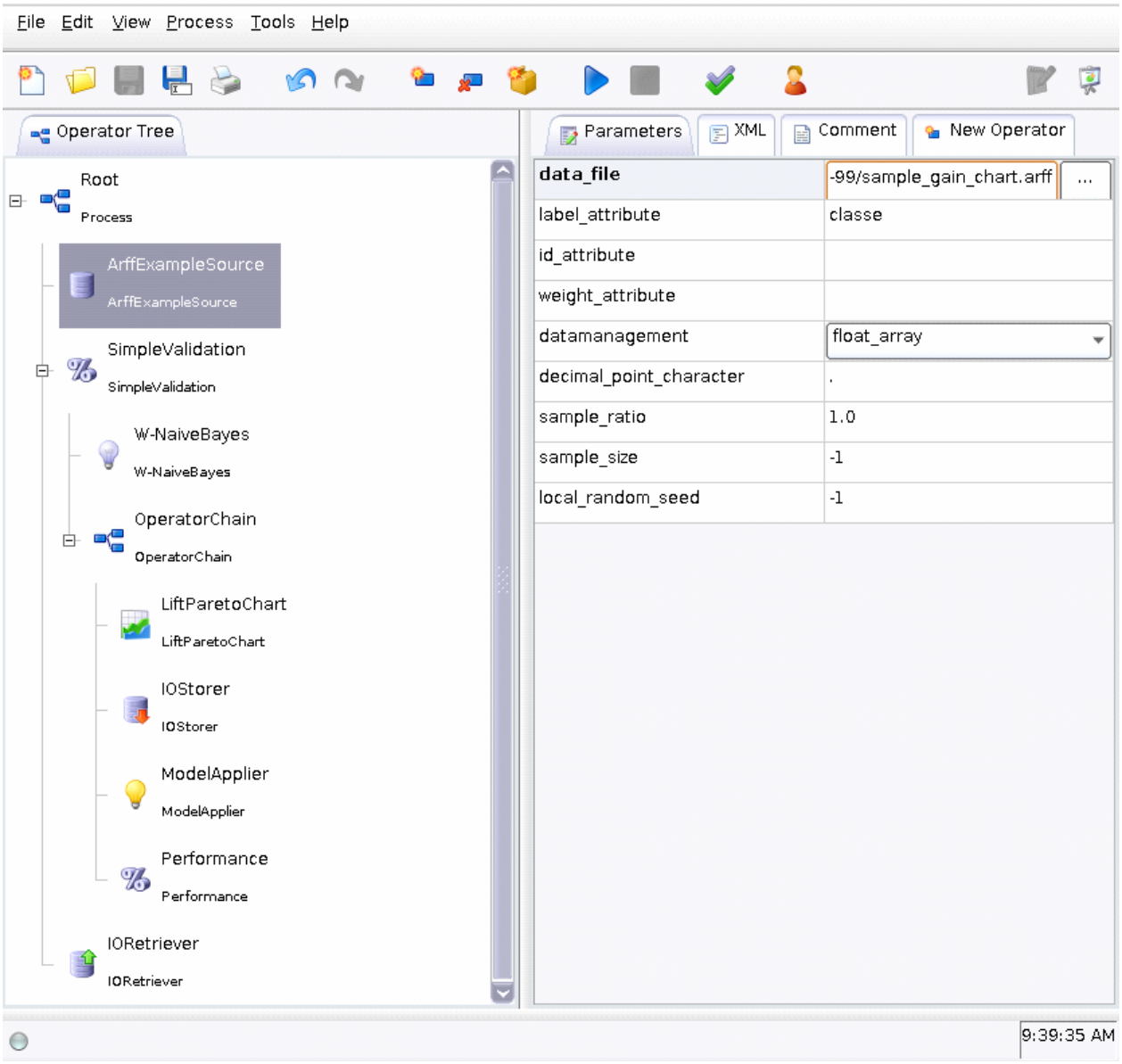
5 RapidMiner

L'installation et la mise en route de RapidMiner (http://rapid-i.com/content/blogcategory/38/69/, Community Edition, version 4.3) ne posent aucun problème. Nous avons créé un raccourci sur le bureau pour démarrer la machine virtuelle Java auquel nous passons en paramètre le fichier « rapidminer.jar ».

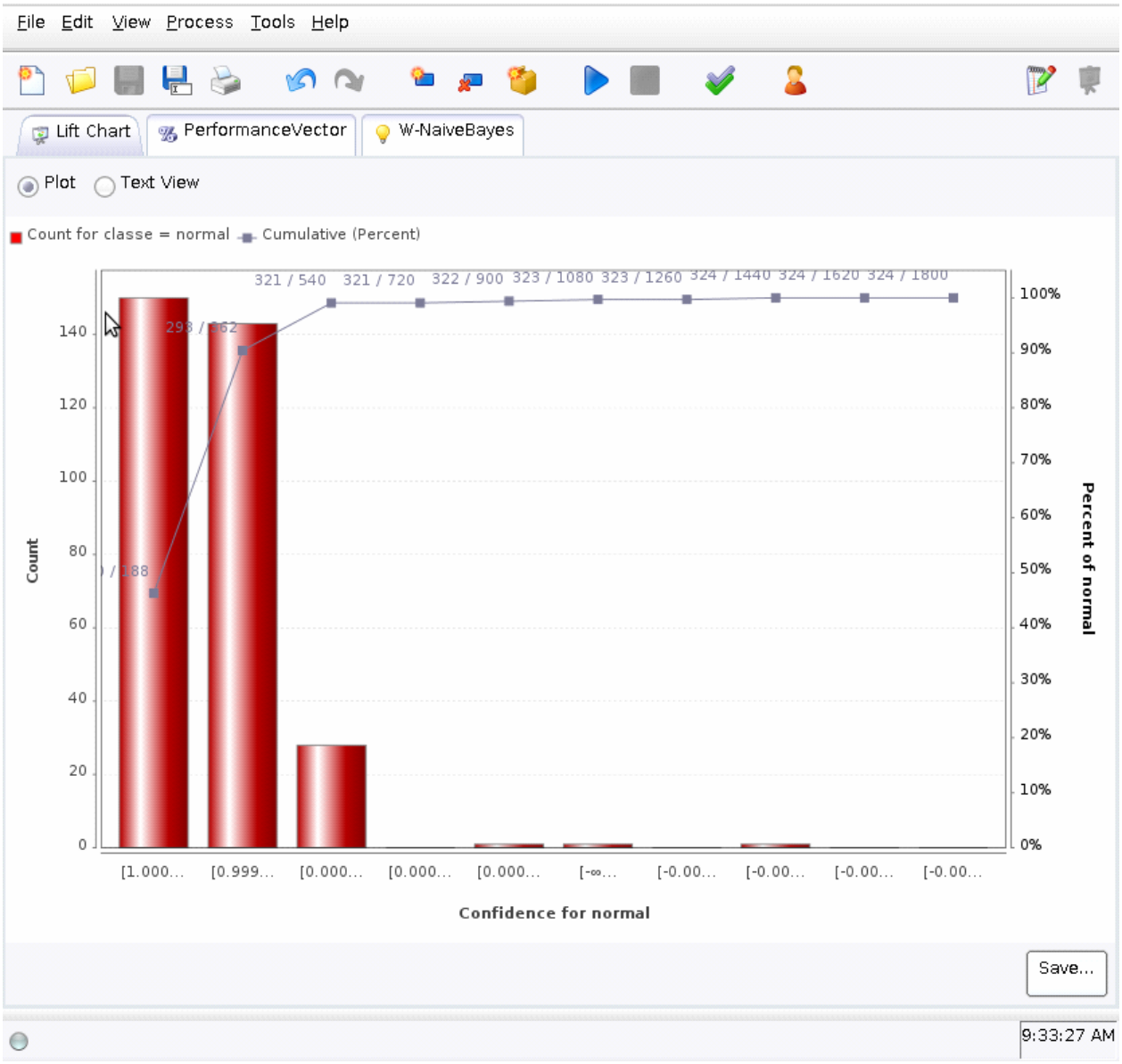
5.1 Définir les traitements sur l'échantillon

Comme pour les autres logiciels, nous paramétrons les traitements sur l'échantillon comportant 2.000 observations. La documentation est très peu disserte sur les problèmes que l'on peut traiter et la manière de procéder avec RapidMiner. Fort heureusement, le logiciel est livré avec un très grand nombre de fichiers exemples.

Pour ce qui est de l'analyse que nous souhaitons mettre en place, nous avons pu trouver dans le sous répertoire « sample/06_Visualization/16_LiftChar.xlm » un exemple dont nous nous sommes inspirés pour produire la bonne séquence de traitements que nous reproduisons ici.



J'avoue ne pas avoir tout compris. Les fonctions IOStorer et IORetriever notamment me paraissent assez mystérieux. Mais bon, l'essentiel est que RapidMiner nous produisent les résultats souhaités. Et c'est bien le cas.



Détaillons un peu ce graphique : il y a 1800 observations dans l'échantillon test, avec 324 positifs (CLASSE = NORMAL). Dans les 20% premiers selon le score (360 observations), nous retrouvons 298 positifs, soit un taux de vrais positifs de 298 / 324 # 92%, ce que nous retrouvons dans l'ordonnée à droite dans le graphique.

Si l'on revient un instant sur les paramètres des traitements, par rapport aux valeurs par défaut, nous avons spécifié :

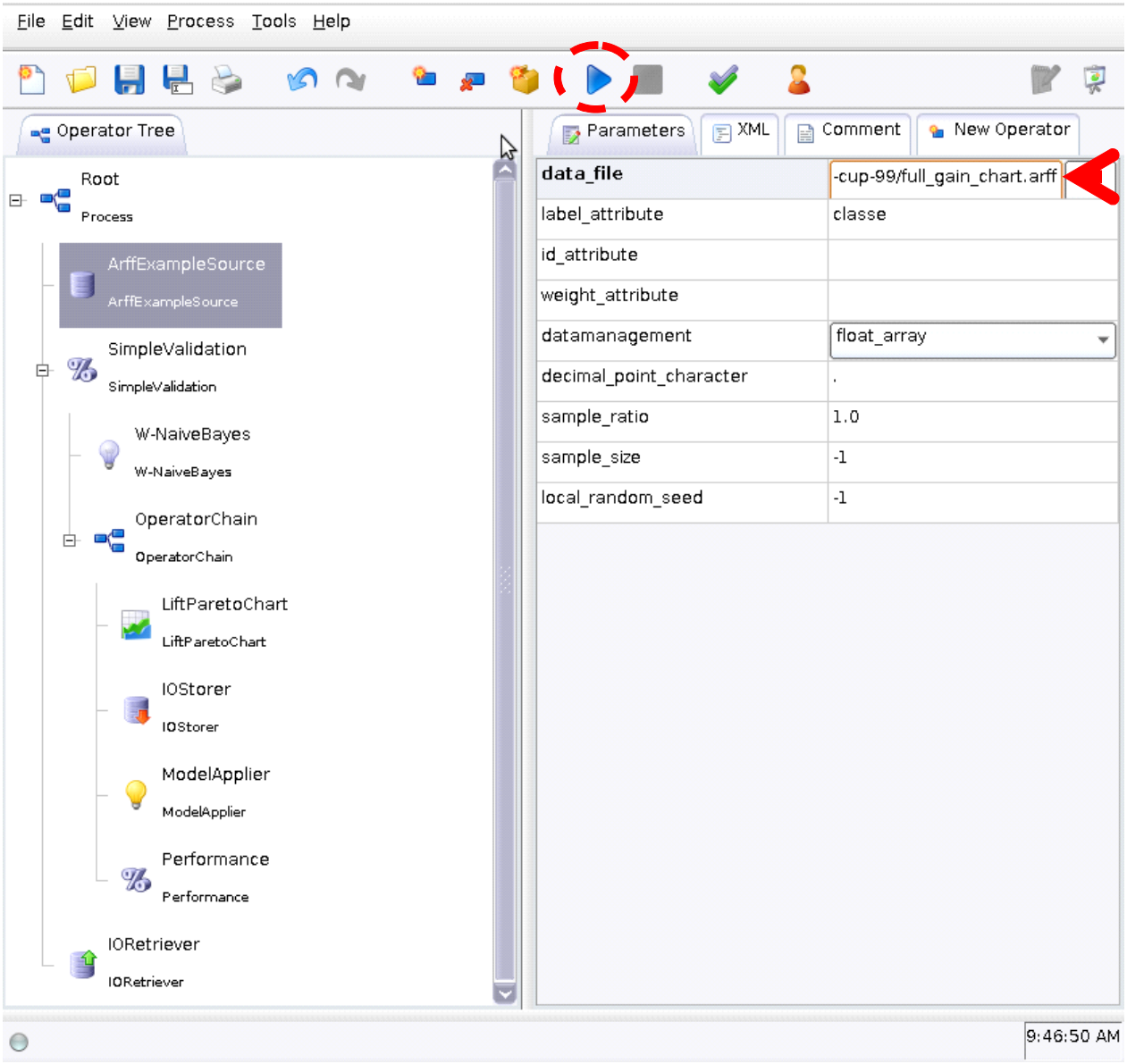
- SIMPLEVALIDATION : SPLIT_RATIO = 0.1 ;
- LIFT PARETO CHART : NUMBER OF BINS = 10

Dans RapidMiner également, en l'absence de l'analyse discriminante, nous avons utilisé le NAIVE BAYES. Une régression logistique est bien présente, en provenance de la bibliothèque Weka, elle est particulièrement performante d'ailleurs. Mais, lorsque nous aurons à traiter la base complète, lancer une optimisation de la log-vraisemblance sur un fichier d'apprentissage de 200.000 observations est irresponsable. D'où le choix du modèle d'indépendance

conditionnelle.

5.2 Traitement sur la totalité de la base

Il ne reste plus qu'à reproduire les mêmes traitements sur la totalité de la base. Nous modifions donc le fichier source dans le composant ARFF EXAMPLE SOURCE. Nous actionnons le gros bouton PLAY dans la barre d'outils.



Le calcul n'est pas pu parvenir à son terme. La base a été chargée en 17 minutes, l'occupation mémoire est passée à 700 Mo lorsque le plantage est survenu. Voici le message renvoyé par le système.

6 Weka

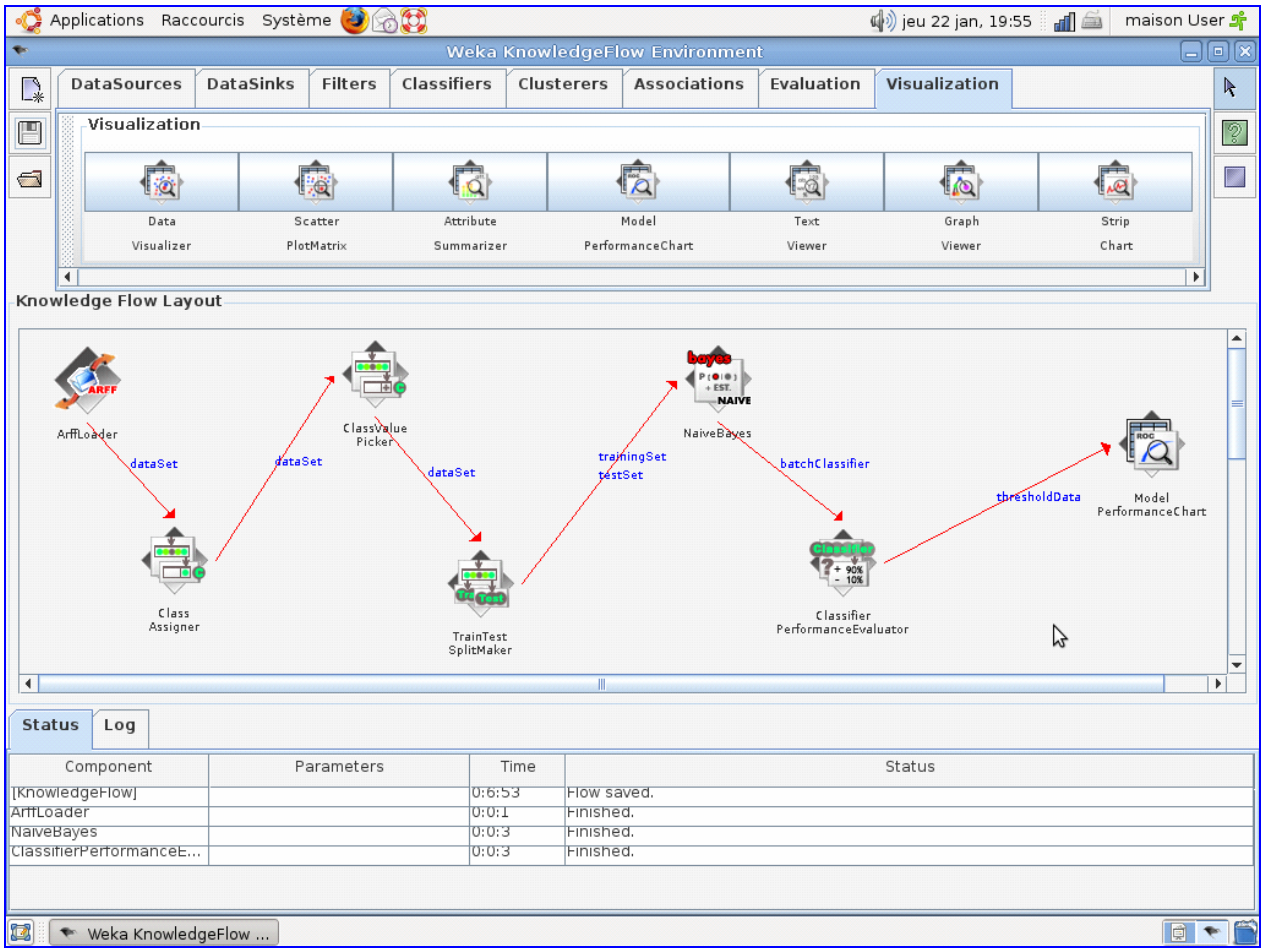
Weka n'est plus à présenter (http://www.cs.waikato.ac.nz/ml/weka/, Version 3-6-0). Fonctionnant sous Java, son installation et son exécution ne posent aucun problème. Nous avons créé un raccourci qui lance automatiquement l'application avec la machine virtuelle Java.

Nous choisissons le mode d'exécution KNOWLEDGEFLOW dans ce didacticiel. Nous travaillons ainsi dans un environnement proche de celui de Knime, présentation adoptée par une très grande majorité de logiciels de Data Mining d'ailleurs.

6.1 Définir les traitements sur l'échantillon

A l'instar des autres logiciels, nous allons définir l'ensemble des traitements sur notre échantillon. Nous créons un nouveau diagramme.

Nous étalons volontairement le positionnement des composants pour que l'on distingue bien, en bleu, le type d'information transmise d'une icône à l'autre. C'est un élément est très important, il conditionne le fonctionnement des composants.

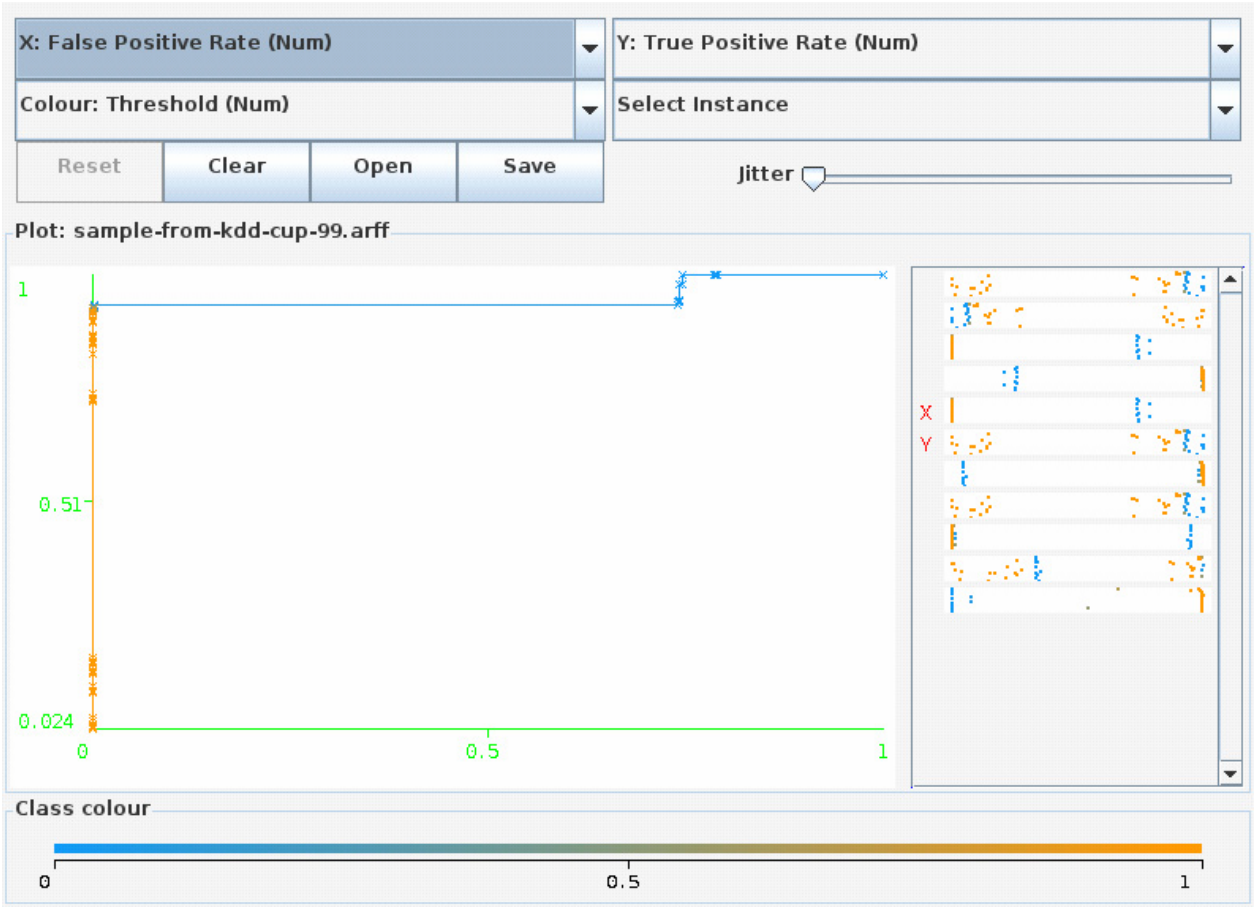


Décrivons un peu les composants et leur paramétrage :

- ARFF LOADER charge le fichier « sample_gain_chart.arff » ;
- CLASS ASSIGNER désigne la variable à prédire « classe », Weka considère que toutes les autres variables sont les prédictives ;
- CLASS VALUE PICKER indique la modalité positive de la variable à prédire, soit « CLASSE = NORMAL » ;
- TRAIN TEST SPLIT MAKER partitionne les données, nous spécifions 10% pour l'apprentissage ;
- NAIVE BAYES est la méthode d'apprentissage, nous lui transmettons à la fois les données apprentissage (TRAINING SET) et les données test (TEST SET), il y a donc 2 connexions entrantes ici ;
- CLASSIFIER PERFORMANCE EVALUATOR évalue les performances du classifieur ;
- MODEL PERFORMANCE CHART produit une série de graphiques destinés à évaluer les performances des modèles, sur l'échantillon test.

Nous lançons les calculs en actionnant le menu contextuel START LOADING du composant ARFF LOADER. Nous consultons le résultat en actionnant le menu SHOW CHART du composant MODEL PERFORMANCE CHART. Weka

propose une fenêtre très astucieuse qui permet de configurer à la volée les séries à placer en abscisse et en ordonnée dans le graphique. Malheureusement, si nous disposons bien du taux de vrais positifs à placer en ordonnée, nous n'avons pas pu trouver la taille de cible que l'on aura du positionner en abscisse pour obtenir la courbe de Gain tel que nous la concevons dans ce document. Nous nous sommes donc rabattus sur la courbe ROC en plaçant en abscisse le taux de faux positifs.



6.2 Calculs sur la base complète

Pour traiter la totalité de la base, nous devons reconfigurer le ARFF LOADER (menu CONFIGURE) et choisir maintenant le fichier « full_gain_chart.arff ». Nous actionnons de nouveau le menu START LOADING.

Au bout d'une vingtaine de minutes, alors le chargement n'est pas achevé, Weka disparaît brutalement. J'ai essayé d'augmenter la taille du tas à la limite de la mémoire disponible sur ma machine, en utilisant l'option de lancement « –Xmx1024m » pour un tas maximum d'une taille de 1024 Mo. Rien n'y a fait.

Weka sous Windows est également confronté au même problème. Le traitement de la base de 2.000.000 d'observations n'est pas faisable.

7 Conclusion

Bien entendu, je ne suis pas à l'abri d'un problème de configuration ou d'un mauvais paramétrage. L'intérêt des logiciels libres est qu'ils sont tous accessibles en ligne. Tout le monde peut reproduire l'expérimentation. Il se peut très bien qu'un chercheur plus habile sache mettre en place un dispositif qui rend les calculs possibles sur sa machine (je pense notamment au paramétrage de la Java Runtime Environment - JRE). Un retour d'expérience sur ce point serait très intéressant.

Nous recensons dans le tableau suivant la durée de chaque étape et l'occupation mémoire pour chaque logiciel.

	Logiciel			
	Tanagra	Knime	Weka	RapidMiner
Durée chargement des données	3 mn	6 mn 40 sec	> 20 mn (erreur)	17 mn
Durée des calculs (app + courbe lift)	25 sec	30 mn	-	erreur
Occupation mémoire à l'issue des calculs (Mo)	342 Mo	193 Mo	> 700 Mo (erreur)	700 Mo (erreur)

Le premier résultat qui attire notre attention est la performance Tanagra via Wine (ou inversement peut être). Malgré que Tanagra tourne via un moteur en sous-main (Wine), les performances ne sont pas altérées, loin de là. Si l'on se réfère au chargement du fichier, où les implémentations sont directement comparables, les écarts entre les temps de traitement laissent dubitatifs.

Knime swappe les données sur disque, la situation est différente. En revanche, Weka et RapidMiner chargent les données en mémoire, exactement comme Tanagra. Nous devrions avoir des valeurs comparables. D'autant plus que les tests que nous avons pu mener sous Windows montraient des temps de chargement similaires sur les grandes bases. Peut être faut-il y voir surtout une efficacité meilleure de la JRE sous Windows ? (cf. http://tutoriels-data-mining.blogspot.com/2008/09/traitement-de-gros-volumes-comparaison.html)

Le second grand résultat de ce comparatif est le comportement impérial de Knime face à la volumétrie. L'occupation mémoire semble totalement maîtrisée. Certes nous utilisons une méthode très simpliste telle que le Bayesien Naïf dans notre étude. On peut se poser la question de la gestion mémoire si nous passons à des techniques plus sophistiquées. Quoi qu'il en soit, en ce qui nous concerne, à aucun moment le système n'a été mis en danger. On doit pouvoir traiter des bases plus importantes encore, à des niveaux où Tanagra serait confronté à des problèmes d'occupation mémoire.

Mais est-il vraiment important de traiter d'aussi gros fichiers ? Un autre aspect sous-jacent de ce comparatif est le bon comportement des méthodes sur un échantillon, pourtant infime par rapport à la base initiale. Pour notre problème de détection des intrusions, travailler sur 2.000 observations nous permet de produire des résultats tout à fait similaires au traitement de la base complète comprenant 2.000.000 d'observations. Voilà un résultat qui donnera à méditer aux ayatollahs du téraoctet.

Enfin, pour être tout à fait exhaustif, un logiciel important manque à ce comparatif. Il s'agit de ORANGE que j'apprécie beaucoup tant pour la qualité des traitements que par son côté très « user-friendly » (http://www.ailab.si/orange/). Malheureusement, je n'ai pas réussi à l'installer (compiler) correctement sous la distribution Ubuntu 8.10. J'ai pourtant suivi à la lettre le descriptif disponible en ligne (http://www.ailab.si/orange/downloads-linux.asp#orange-1.0-ubuntu).

La version Windows, elle, a donné toute satisfaction et a bien produit la courbe LIFT sur notre échantillon. Nous ne reprendrons cependant pas les résultats en termes d'occupation mémoire et de temps de traitement dans ce document, les configurations systèmes ne sont pas comparables.