1 Objectif

Comparer les performances de plusieurs logiciels pour la régression logistique.

Gratter les millisecondes est le péché mignon des informaticiens. Au-delà de la petite satisfaction personnelle, il y a quand même des enjeux forts derrière l'optimisation des programmes. Notre rôle est de produire des logiciels fiables, rapides, avec une occupation mémoire contenue. Dans le cadre du data mining, cela se traduit par la capacité à traiter les grandes bases de données. Certes, dans la phase finale où il s'agit de produire le modèle qui sera déployé dans le système d'information, qu'importe finalement que les calculs durent une ½ heure ou une ½ journée. Mais il y a la phase exploratoire en amont, lorsque nous cherchons les solutions les mieux adaptées à notre problème. Plus rapide sera l'outil, plus de configurations nous pourrons tester. Nous aurons ainsi de meilleures chances de mettre en évidence la solution la plus efficace.

La régression logistique fait certainement partie des algorithmes pour lesquels j'ai tenté de nombreuses optimisations. En définitive, la question a souvent été centrée sur le choix de l'algorithme d'optimisation de la vraisemblance. Des études comparatives ont été menées, ces travaux m'ont d'ailleurs beaucoup aidé[1]. La lecture du code source de Tanagra montre à quel point j'ai longtemps hésité avant d'adopter une solution plus ou moins satisfaisante.

Comme on n'est jamais seul au monde, il m'est apparu intéressant de comparer les temps de traitement et l'occupation mémoire de la régression logistique de Tanagra avec ceux des autres outils gratuits largement répandus au sein de la communauté du Data Mining. J'avais déjà mené un travail similaire par le passé[2]. La nouveauté dans ce tutoriel est que nous nous situons dans un nouveau cadre : j'utilise maintenant un OS 64 bits (Windows 7), et certains de ces logiciels sont justement passés aux 64 bits avec des capacités de traitements accrus comme on a pu le constater pour les algorithmes de construction des arbres de décision[3]. J'ai donc largement augmenté la taille de la base à traiter. Pour corser l'affaire, des attributs générés complètement aléatoirement ou de manière à être corrélés avec les variables initiales ont été rajoutées. L'objectif est d'observer le comportement des logiciels durant la recherche des prédicteurs pertinents.

Dans ce comparatif, outre **Tanagra 1.4.14** (32 bits), nous utiliserons les logiciels **R 2.13.2** (64-bits), **Knime 2.4.2** (64-bits), **Orange 2.0b** (build 15 oct2011, 32 bits) et **Weka 3.7.5** (64 bits).

2 Données

Choisir la base adéquate pour une expérimentation est toujours une opération délicate. Il ne faut pas que ses particularités interfèrent avec ce que nous souhaitons évaluer au risque de biaiser les résultats. C'est une des raisons pour lesquelles j'utilise souvent les mêmes, parce que je connais très bien leurs caractéristiques.

[1] T.P. Minka, « A comparison of numerical optimizers for logistic regression », 2007.

[2] Tutoriel Tanagra, « Régression logistique – Comparaison de logiciels », Octobre 2008.

[3] Tutoriel Tanagra, « Arbres de décision sur les grandes bases (suite) », Janvier 2012.

La base « waveform » décrite dans l'ouvrage de Breiman et al. (1984), fait partie de mes favorites. Notamment parce qu'on peut générer autant d'observations que l'on veut. Nous avons la possibilité d'étudier le comportement des logiciels sur une base potentiellement infinie. Et aussi parce qu'il est relativement facile d'en modifier les caractéristiques à notre guise. En ce qui nous concerne, nous l'avons transformé en un problème à 2 classes, et nous avons rajouté un nombre relativement important de descripteurs susceptibles de perturber l'apprentissage. Nous décrivons dans la section suivante le programme – écrit en langage R – de génération des données.

2.1 Waveform « classique »

Un générateur de la base « waveform » est accessible sur le site de l'ouvrage de Hastie et al. (2009)[4]. Le programme est particulièrement laconique. Mais il permet bien de générer une base de « n » observations, l'attribut classe à 3 modalités, et les 21 variables prédictives.

```
#from http://www-stat.stanford.edu/~tibs/ElemStatLearn/data.html
waveform <- function(n)
{
  class <- as.numeric(cut(runif(n), c(0, 1/3, 2/3, 1)))
  h <- function(xoff)
    pmax(6 - abs(seq(21) - 11 + xoff), 0)
  x <- rbind(h(0), h(-4), h(4))
  x1 <- x[c(1, 1, 2),  ][class,  ]
  x2 <- x[c(2, 3, 3),  ][class,  ]
  u <- runif(n)
  data.frame(x = I(u * x1 + (1 - u) * x2 + rnorm(n * 21)), y = class)
}
```

2.2 Waveform « binaire »

Pour la transformer en problème binaire, nous utilisons un artifice très simple : nous générons plus d'observations qu'il n'en faut, puis nous supprimons la 3ème classe. Si la base résultante comporte plus de n lignes, nous la tronquons. Cela ne porte pas à conséquence parce que l'ordre de génération des individus ne dépend pas de la classe d'appartenance.

```
#modify waveform in a binary problem by removing
#the instances for the third class value
waveform.binary <- function(n){
  tmp <- waveform(trunc(1.75*n))
  output <- subset(tmp, tmp$y != 3)
  if (nrow(output) > n){
    output <- output[1:n,]
  }
  output$y <- factor(output$y)
  levels(output$y) <- c("A","B")
  return(output)
}
```

[4] T. Hastie, R. Tibshirani, J. Friedman, « The elements of statistical learning », Springer, 2009.

2.3 Ajout des descripteurs aléatoires

Nous ajoutons des descripteurs générés aléatoirement pour rendre l'apprentissage plus compliqué. Ils devraient être évacués lors de la sélection de variables. Nous procédons comme suit : nous sélectionnons au hasard une des 21 variables initiales, nous en calculons la moyenne et l'écart-type, nous générons un vecteur gaussien de n observations avec ces caractéristiques. L'idée est d'obtenir des descripteurs crédibles, respectant le domaine de définition des variables initialement présentes.

```
#add K random attributes
add.rnd <- function(wave.data, K = 1){
 n <- nrow(wave.data)
 new.wave <- wave.data
 for (k in 1:K){
  colref <- trunc(runif(1,min=1,max=22))
  newcol <- rnorm(n,mean=mean(wave.data$x[,colref]),sd=sd(wave.data$x[,colref]))
  new.wave <- cbind(new.wave,newcol)
  names(new.wave)[ncol(new.wave)] <- paste("rnd",k,sep="_")
 }
 return(new.wave)
}
```

2.4 Ajout de descripteurs corrélés

Pour ajouter de la difficulté, nous générons des descripteurs corrélés avec les variables initiales. Ils devraient rendre la sélection de variables encore plus périlleuse. Nous procédons de la manière suivante : nous sélectionnons au hasard une des 21 variables initiales, nous lui ajoutons un bruit blanc gaussien de moyenne 0 et d'écart-type « noise » x écart-type de la variable de référence. Lorsque nous diminuons « noise », la corrélation entre la variable de référence et celle générée est plus forte. Par défaut, nous utilisons « noise = 1 ». Ce type de variable est particulièrement vicieux. Elle est corrélée avec une des variables initiales, sans l'être directement avec la classe. Il faut que l'algorithme de sélection puisse détecter cette situation.

```
#add L correlated attributes
add.correlated <- function(wave.data,L = 1, noise=1){
 n <- nrow(wave.data)
 new.wave <- wave.data
 for (l in 1:L){
  colref <- trunc(runif(1,min=1,max=22))
  newcol <- wave.data$x[,colref] + rnorm(n,0,sd=sd(wave.data$x[,colref])*noise)
  new.wave <- cbind(new.wave,newcol)
  names(new.wave)[ncol(new.wave)] <- paste("cor",l,sep="_")
 }
 return(new.wave)
}
```

2.5 Programme principal de génération des données

Le programme suivant génère les données. Nous obtenons à la sortie : n = 300000 observations, 21 variables prédictives originelles (V), 1 variable à prédire (Y), 50 variables générées aléatoirement (rnd), et 50 variables corrélées (cor). Les valeurs numériques sont arrondies à la 3ème décimale.

```
#function for rounding numeric columns
myround <- function(x){
  if (is.factor(x)==T){
  return(x)
  } else
  {
  return(round(x,3))
  }
}

#generate n instances with K rnd attributes and L correlated attributes
generate.binary <- function(n,K=1,L=1,noise=1){
  tmp <- waveform.binary(n)
  tmp <- add.rnd(tmp,K)
  tmp <- add.correlated(tmp,L,noise)
  output                                                              <-
cbind(as.data.frame(matrix(tmp$x,nrow(tmp),21)),subset(tmp,select=-x))
  output <- as.data.frame(lapply(output,myround))
  return(output)
}

#generate and save a dataset
dataset.size <- 300000 #number of instances
nb.rnd <- 50 #number of random variables
nb.cor <- 50 #number of correlated variables
noise.level <- 1 #noise for correlated variables
data.wave <- generate.binary(dataset.size,nb.rnd,nb.cor,noise.level)
summary(data.wave)

#writting the data.frame into a file (tab delimited file format)
write.table(data.wave,file="wavebin.txt",quote=F,sep="\t",row.names=F)
```

3 Régression logistique

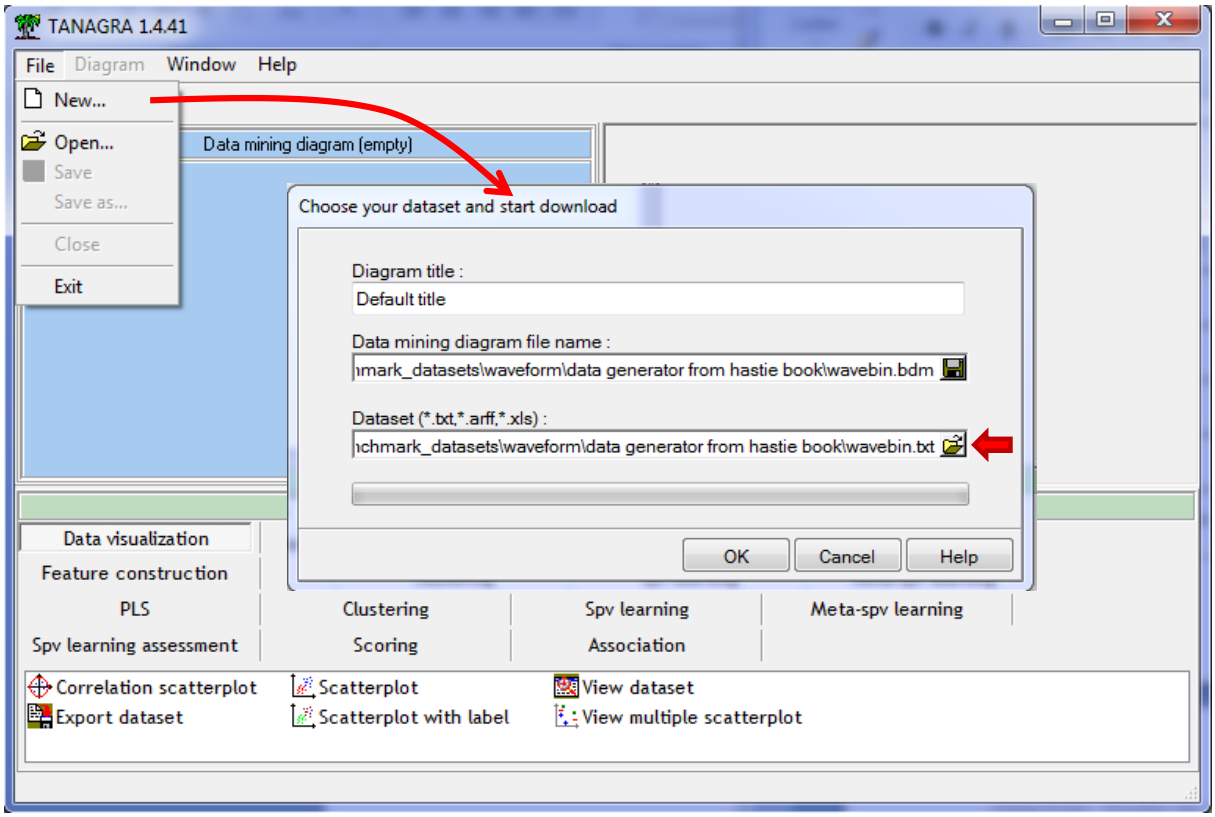
Nous utiliserons le fichier « **wavebin.txt** » dans notre expérimentation. Nous l'avons limité à 300.000 observations mais, bien évidemment, vous avez toute liberté d'utiliser le générateur de données pour créer des bases de taille plus importante, avec plus ou moins de variables perturbatrices.

Attention, nous avons généré un fichier avec le « . » comme point décimal. Si votre machine est configurée pour la virgule « , » (Windows en Français en particulier), vous devez modifier le fichier au préalable en utilisant un éditeur de texte c.-à-d. remplacer les « . » par des « , ».

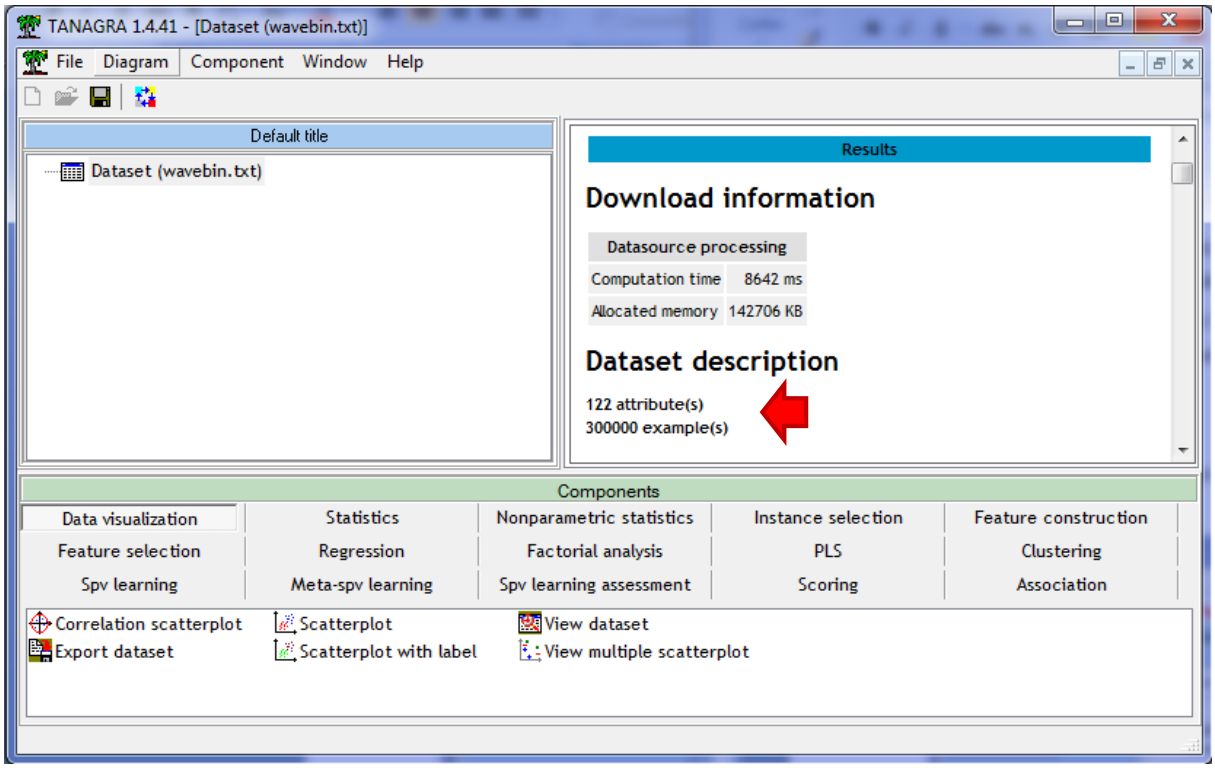
Nous utilisons toutes les variables candidates dans un premier temps c.-à-d. 21 + 50 + 50 = 121 prédicteurs. L'objectif est d'évaluer le comportement des implémentations, toutes choses égales par ailleurs. Nous avons généré un second fichier avec 300 observations « **wavebin_small.txt** ». Nous l'avons utilisé pour paramétrer, pour vérifier le bon fonctionnement des logiciels et, accessoirement, pour réaliser certaines copies d'écran.

3.1 Tanagra

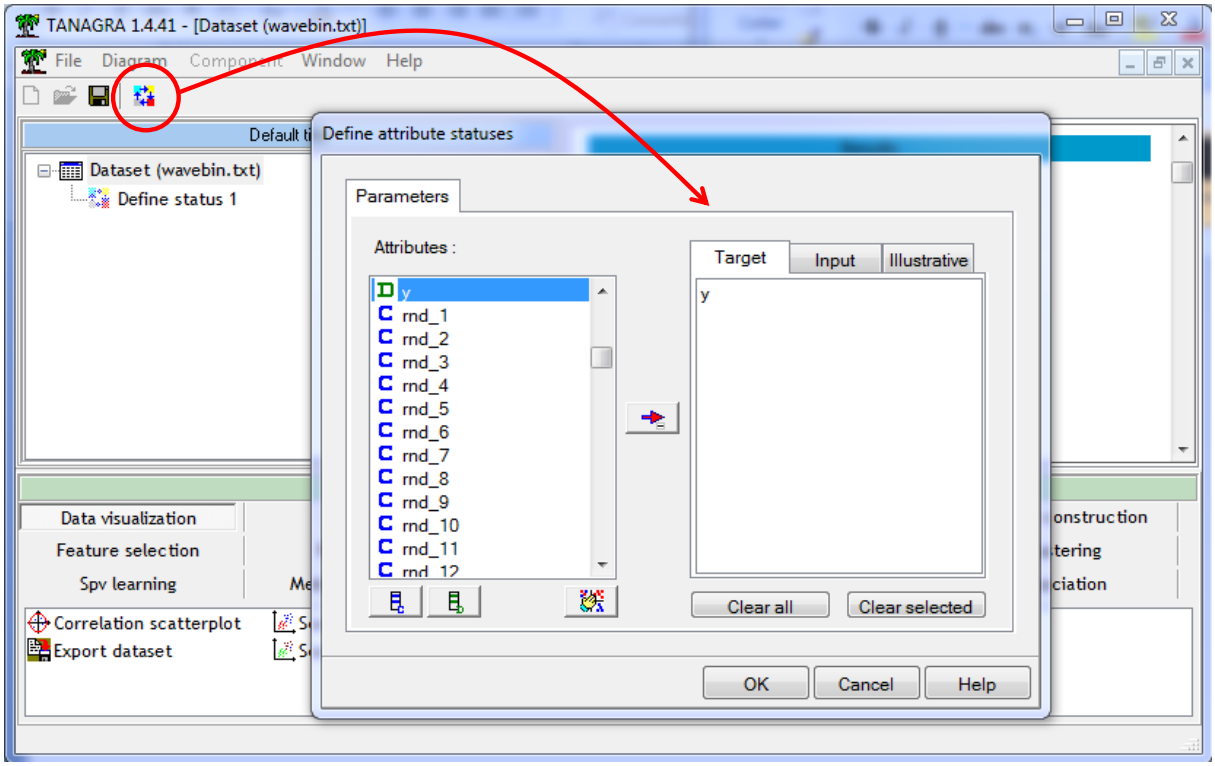
Nous actionnons le menu FILE / NEW pour créer un nouveau diagramme et importer les données.



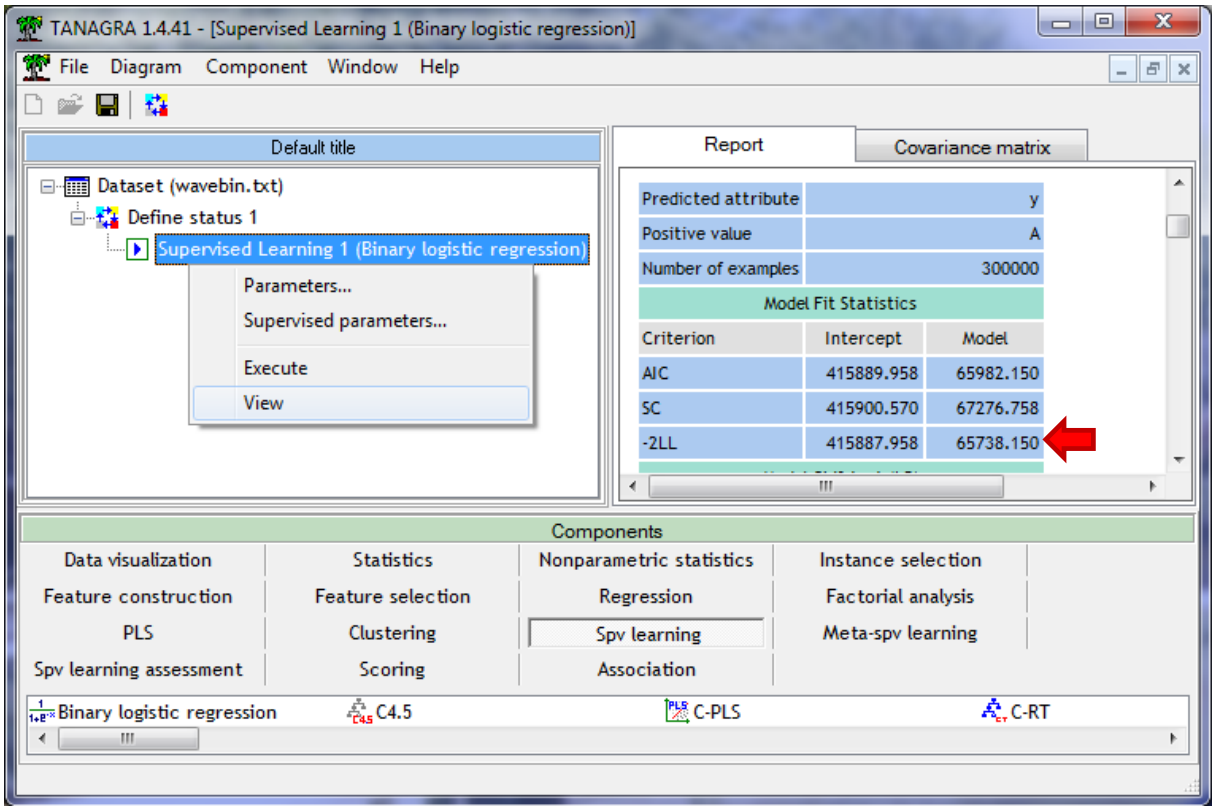
Nous vérifions que les données ont bien été importées.



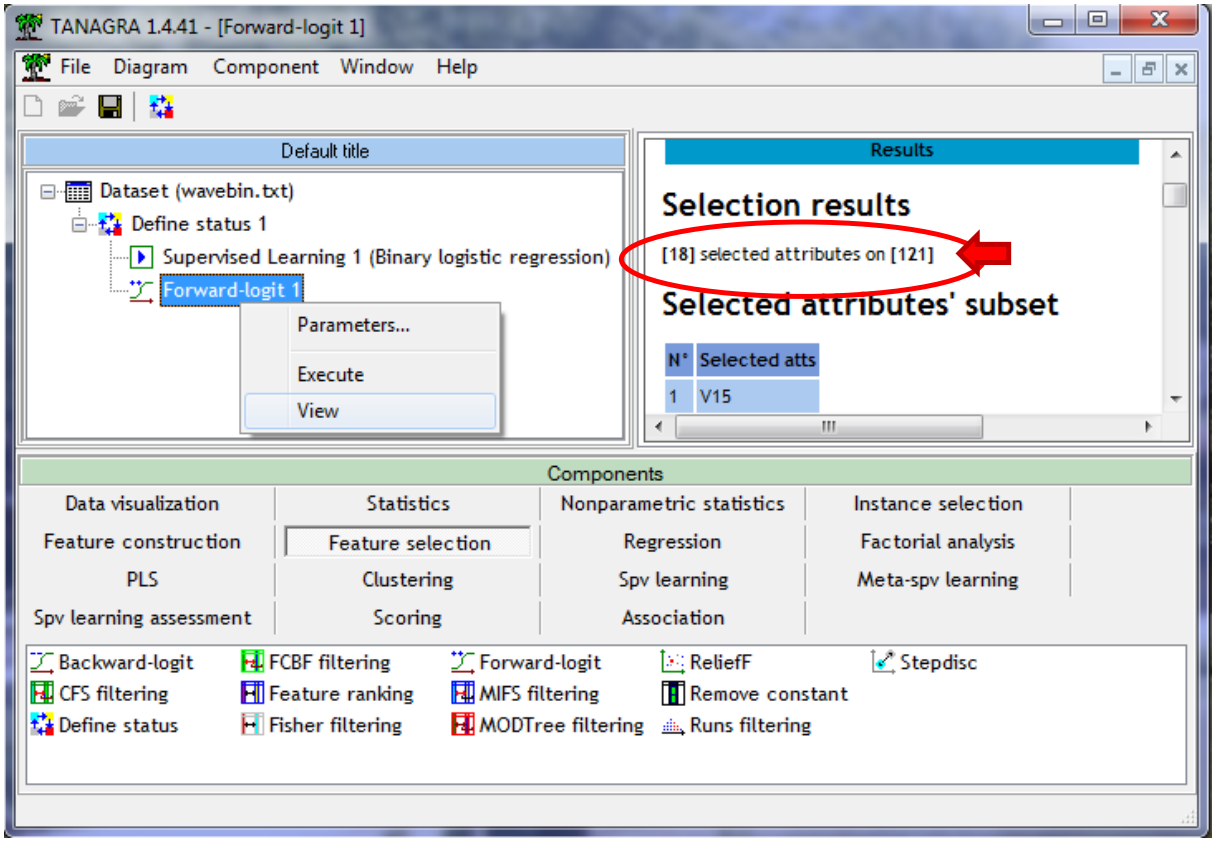
Nous sélectionnons les variables pour l'apprentissage, Y est la variable cible (TARGET), les autres sont les variables prédictives (INPUT).



Il ne nous reste plus qu'à placer le composant BINARY LOGISTIC REGRESSION (onglet SPV LEARNING) et lancer les calculs en actionnant le menu contextuel VIEW.



Pour la sélection de variables, nous utilisons le composant FORWARD LOGIT (onglet FEATURE SELECTION), en laissant les paramètres par défaut.



3.2 Logiciel R

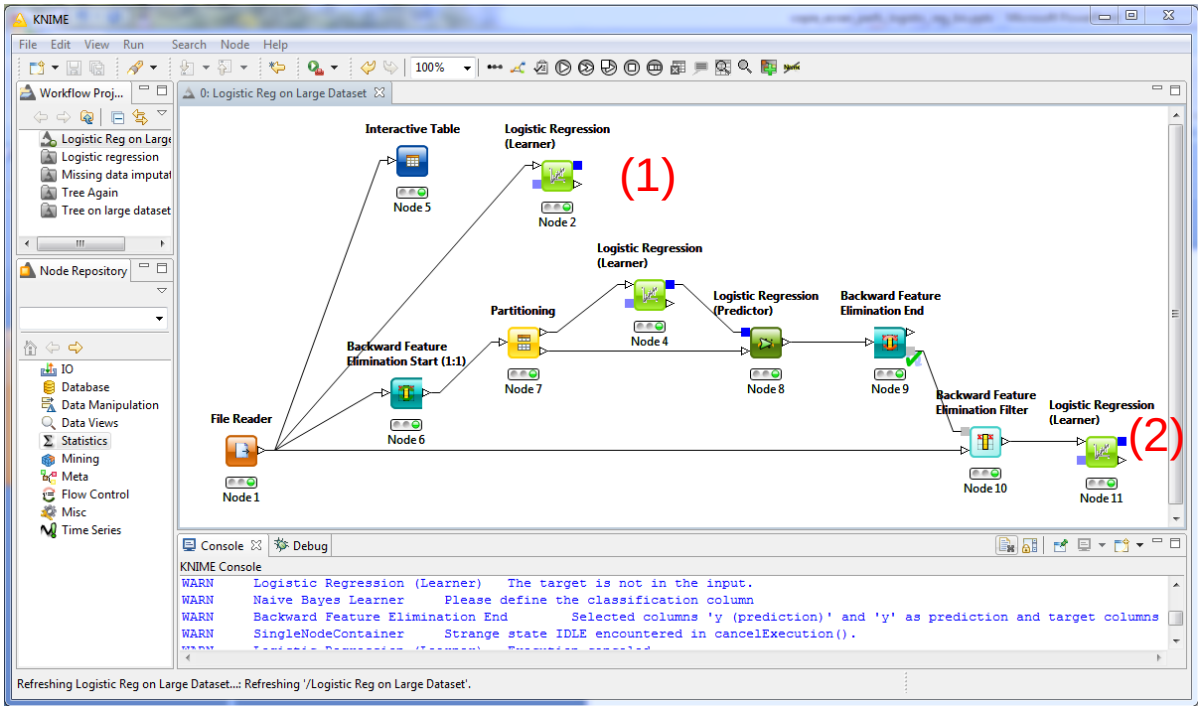
Nous avons utilisé le programme suivant pour R. Le temps de calcul a été mesuré à l'aide de la commande **system.time(…)**.

```
#data importation
system.time(wave <-
read.table(file="wavebin.txt",sep="\t",dec=".",header=T))

#creating the model
system.time(model <- glm(y ~ ., data = wave, family=binomial))
print(model)

#variable selection
library(MASS)
model.default <- glm(y ~ 1, data = wave, family=binomial)
duree <- system.time(model.forward <-
stepAIC(model.default,scope=list(lower=as.formula(model.default),upper=as.f
ormula(model)),direction="forward",k=log(nrow(wave))))
print(model.forward)
```

3.3 Knime

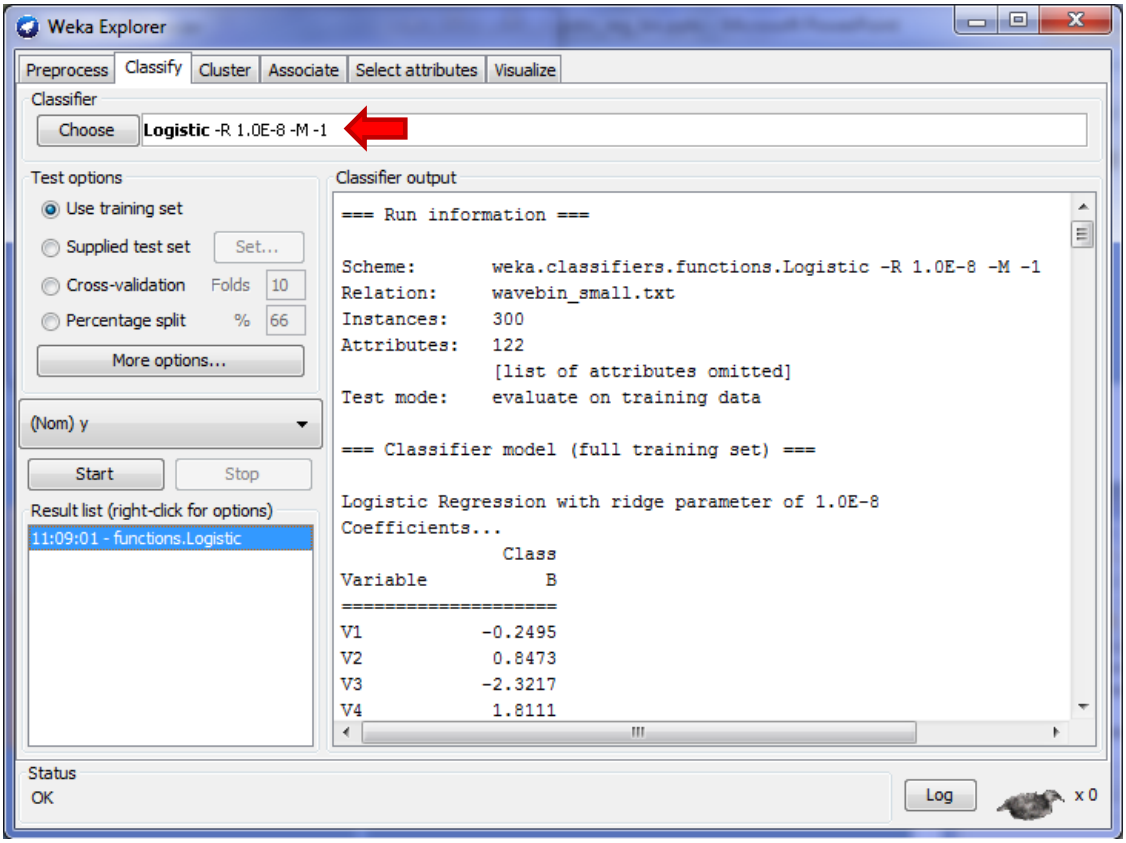
Nous avons défini le projet suivant dans Knime. Il intègre à la fois la construction du modèle sur la totalité des variables (1) et la sélection des variables à l'aide de la méthode « wrapper » (2). La construction de la construction du diagramme est décrite en détail dans un de nos tutoriels[5].

[5] Tutoriel Tanagra, « ''Wrapper'' pour la sélection de variables (suite) », Janvier 2010.



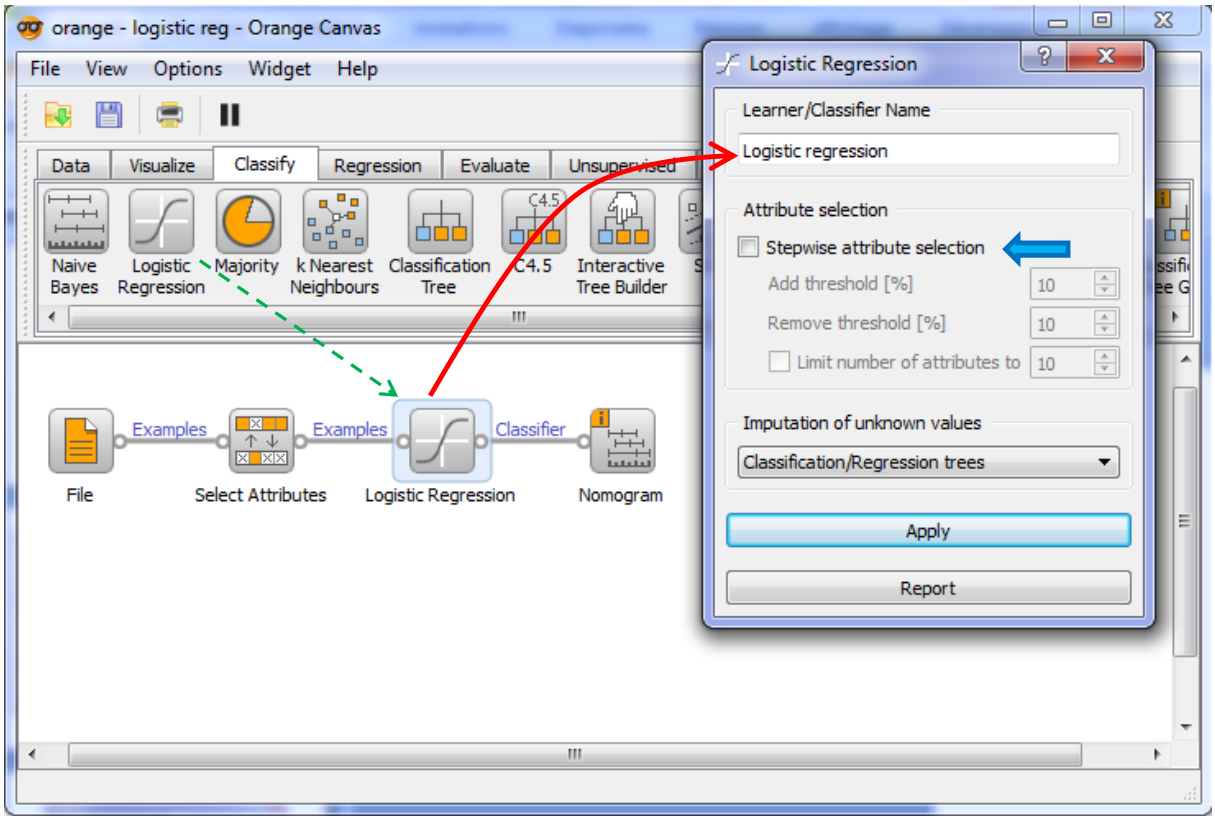
3.4 Weka

Nous utilisons le mode « Explorer » sous Weka. Après avoir importé les données, nous lançons la fonction « Logistic » avec les paramètres suivants.



3.5 Orange

Nous avons défini le schéma suivant sous Orange. Le composant « Select Attributes » permet de sélectionner la variable à prédire Y et les variables prédictives candidates. On notera que la sélection de variables est directement intégrée dans le composant régression logistique.



3.6 Performances comparées

Nous récapitulons les résultats dans un tableau. Notons que tous les logiciels ont trouvé la même solution avec une déviance de **D = 65738.15**. La qualité de la modélisation est la même, quel que soit le logiciel utilisé. C'est un résultat rassurant quant à la bonne tenue de ces outils.

Logiciel (x-bits)	Temps de calcul (**secondes**)		Occupation mémoire (Go)		
	Importation	Modélisation	Après importation	Max. durant modélisation	Ecart
Tanagra 1.4.41 (32)	9	74	0.15	0.37	0.22
R 2.13.2 (64)	51	171	0.57	2.29	1.72
Knime 2.4.2 (64)	34	192	2.95	3.84	0.89
Weka 3.5.7 (64)	63	300	2.1	2.41	0.31
Orange 2.0b (32)	151	-	1.27	-	-

3.6.1 Temps de calcul

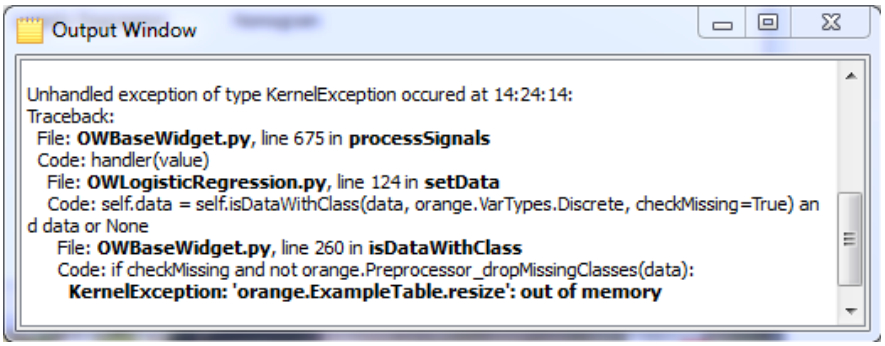
Nous utilisons les chiffres fournis par les logiciels lorsque ces derniers les indiquent directement. Dans le cas contraire, nous utilisons un chronomètre. Les valeurs sont assez approximatives, le plus important pour nous est d'avoir un ordre d'idées.

Tanagra est manifestement plus rapide que les autres implémentations. A ce point là, ça intrigue quand même. Il y a déjà les choix purement algorithmiques. Le travail de ***Minka*** cité plus haut m'a été très précieux. Après, il y a la « bidouille ». L'objectif est d'éviter à tout prix de recalculer deux fois les mêmes valeurs, quitte à prendre plus de mémoire pour le stockage des résultats intermédiaires. A la sortie, les performances sont très encourageantes.

R et **Knime** viennent ensuite. Tous deux sont en 64 bits. Ils sont très performants.

Knime sait stocker les informations sur le disque lorsque l'occupation mémoire atteint un certain seuil (http://tech.knime.org/faq - chercher « *Memory Policy* »). C'est une fonctionnalité particulièrement précieuse dans le cadre du traitement des très grandes bases de données. Toutefois, en ce qui nous concerne, pour mettre tous les logiciels sur un pied d'égalité, nous avons demandé à ce que toutes les informations soient montées en mémoire centrale. De fait, l'occupation mémoire est grevée mais les temps de calculs sont largement améliorés. Avec l'option par défaut qui lui autorise à copier certaines informations sur le disque lors du traitement des grandes bases, le temps de calcul de la régression est doublé (≈ 7 minutes).

Sur l'échantillon de 300 observations, **Orange** a parfaitement fonctionné. Lors du traitement de la base « wavebin », le fichier a bien été importé. En revanche, une erreur est apparue lorsque la régression logistique a démarré.



J'avais pensé initialement à un problème de paramétrage. Mais en cherchant un peu, j'ai trouvé une explication sur le forum du site web (http://orange.biolab.si/forum/viewtopic.php?f=4&t=1369).

3.6.2 Occupation mémoire

Nous mesurons l'occupation mémoire maximum en surveillant le Gestionnaire des tâches de Windows. Le procédé est certes un peu artisanal, mais c'est la seule approche qui m'a semblé la plus fiable. En effet, à la fin du processus, certains logiciels détruisent les structures intermédiaires de calcul. Mesurer la mémoire utilisée à ce stade ne reflète en rien la réalité.

La colonne « écart » indique la consommation en mémoire de l'implémentation de la régression logistique. Une idée intéressante par exemple serait d'analyser cet indicateur à mesure que la taille

de la base augmente, en nombre d'observations d'une part (faibles modifications), et en nombre de variables prédictives d'autre part (fortes modifications, notamment à cause de la matrice hessienne).

L'occupation mémoire globale indique l'aptitude du logiciel à traiter les grandes bases. On peut la mettre directement en rapport avec les caractéristiques de la machine utilisée, en n'oubliant pas que les logiciels/systèmes 32 bits et 64 bits n'ont pas les mêmes capacités d'adressage. Notons que les chiffres ne sont pas toujours directement comparables. Tanagra par exemple semble très parcimonieux parce qu'il stocke les données en simple précision. Seuls les calculs effectués pour la modélisation sont réalisées en double précision, d'où les résultats équivalents aux autres logiciels. L'occupation mémoire est ainsi mécaniquement réduite de moitié (4 octets par valeur en simple précision vs. 8 octets en double précision). Il semble néanmoins que cette particularité n'explique pas complètement les différences.

Concernant **Knime**, avec l'option de gestion mémoire par défaut (une partie des informations sont copiées sur disque), l'occupation est très largement réduite durant le calcul de la régression, passant à ≈ 0.38 Go, décuplant ainsi son potentiel pour le traitement des très grandes bases de données.

4 Sélection de variables

Les logiciels utilisent des techniques complètement différentes pour la sélection de variables. De fait, les temps de calcul ne sont comparables dans l'absolu. Ils dépendent essentiellement du nombre de régressions logistiques effectuées durant le processus de recherche.

Notons « p » le nombre de variables candidates.

Tanagra utilise la méthode de scores dans la recherche « forward ». C'est la plus rapide qui soit. Au pire cas, toutes les variables sont finalement sélectionnées, il effectuera « p » régressions logistiques c.-à-d. « p » optimisations de la log-vraisemblance. L'algorithme est de complexité linéaire [O(p)]. Elle n'en est pas moins efficace puisque, comme nous ne constatons dans le tableau ci-dessous, aucune des variables perturbatrices (« rnd », ça paraissait évident ; « cor », c'était autrement plus difficile) n'est venue s'immiscer dans la sélection.

R avec stepAIC a été paramétré pour réaliser une recherche « forward ». Il vise à optimiser le critère Akaike (AIC). A la première étape, il réalise « p » régressions, à la seconde « p-1 » régressions, etc. Au pire cas, « p x (p+1) / 2 » régressions sont effectuées. L'algorithme est de complexité quadratique [$O(p^2)$]. Le temps de calcul est lourdement pénalisé.

Knime propose essentiellement une recherche « backward », basée sur la stratégie « wrapper ». Il part de la configuration complète, avec l'ensemble des variables. Il cherche à retirer la variable qui améliore le taux d'erreur en test (les données sont partitionnées à la volée). A partir du modèle à « p-1 » variables, il cherche à retirer la seconde variable, etc. A la première étape, il réalise donc « p » régressions, à la seconde « p-1 » régressions. L'algorithme est de complexité quadratique $O(p^2)$ mais, de surcroît, à la différence du « forward », il part des configurations les plus complexes. Il est doublement pénalisé. Rien que sur le jeu d'essai « wavebin_small.txt », le temps de calcul est déjà prohibitif. Nous avons renoncé à lancer la procédure sur la base complète. Manifestement, pour la

stratégie « wrapper », mieux vaut s'en tenir à des techniques particulièrement rapides telles que le classifieur bayésien naïf[6].

Weka, tout comme Knime, n'intègre pas une procédure de sélection de variables dédiée à la régression logistique. Une solution possible est d'insérer une technique de filtrage avant la régression elle-même. Par le passé, nous avions testé la méthode CFS basée sur les corrélations croisées[7]. Mais la solution est bancale. En effet, la sélection est basée sur un critère ad hoc, sans aucun rapport avec la régression. C'est pour cette raison que nous ne l'avons pas intégrée dans cette seconde partie.

Orange intègre la recherche bidirectionnelle (stepwise) dans la régression logistique. Il vérifie que chaque ajout de variable ne remet pas en cause la présence de celles déjà sélectionnées dans le modèle. Il se base sur les tests de significativité. Mais la documentation n'est pas très disserte à ce sujet. Je n'ai pas réussi à déterminer s'il s'agissait d'un test de Wald, d'un test du rapport de vraisemblance, ou d'un test des scores.

Finalement, nous rapportons ici les résultats obtenus avec Tanagra et R.

Logiciel (Méthode)	Variables sélectionnées			Durée des calculs
	# variables	Dont « rnd »	Dont « cor »	
Tanagra (Forward, Méthode des scores)	18	0	0	9' 30''
R (Forward, StepAIC)	18	0	0	4h 54' 00''

Comme je le disais plus haut, il ne faut pas trop s'attacher aux temps de calculs ici, les algorithmes utilisés étant très différents. En revanche, nous observons que les deux stratégies ont su déjouer le rôle perturbateur des variables « rdn » (très bien) et « cor » (excellent). C'est un point très positif.

5 Conclusion

L'énorme intérêt de ce tutoriel est que nous pouvons générer les données à notre guise à l'aide du programme décrit dans la section 2. Ainsi, tout un chacun peut créer des configurations proches de ses préoccupations (plus ou moins d'observations et/ou de variables). Les conclusions seront d'autant plus pertinentes.

Concernant les performances, les arbres de décision et la régression logistique sont les algorithmes que j'ai énormément travaillés dans Tanagra. Les résultats sont plutôt encourageants, tant mieux. Ils sont moins flatteurs en revanche sur d'autres techniques (ex. SVM - http://tutoriels-data-mining.blogspot.com/2008/10/svm-comparaison-de-logiciels.html). Bref, il me reste encore un peu de boulot d'optimisation, tant mieux aussi.

[6] http://tutoriels-data-mining.blogspot.com/2010/01/wrapper-pour-la-selection-de-variables.html

[7] http://tutoriels-data-mining.blogspot.com/2008/10/rgression-logistique-comparaison-de.html