

1 Introduction

Stratégie WRAPPER pour la sélection de variables dans SIPINA et le logiciel R (package RWeKa). Confrontation avec les méthodes FILTRE implémentées dans TANAGRA.

Sélection de variables. La sélection de variables¹ est un aspect essentiel de l'apprentissage supervisé. Nous devons déterminer les variables pertinentes pour la prédiction des valeurs de la variable à prédire, pour différentes raisons : un modèle plus simple sera plus facile à comprendre et à interpréter ; le déploiement sera facilité, nous aurons besoin de moins d'informations à recueillir pour la prédiction ; enfin, un modèle simple se révèle souvent plus robuste en généralisation c.-à-d. lorsqu'il est appliqué sur la population.

Trois familles d'approches sont mises en avant dans la littérature.

Les approches FILTRE consistent à introduire les procédures de sélection préalablement et indépendamment de l'algorithme d'apprentissage mise en oeuvre par la suite. L'avantage est la rapidité et la souplesse. Rien ne nous garantit en revanche que la sélection basée sur des critères ad hoc produise un sous-ensemble de prédicteurs « optimaux » pour tout type de méthode d'apprentissage subséquente. Les méthodes de RANKING sont certainement les plus représentatives de cette famille. On calcule individuellement un indicateur caractérisant le lien entre la classe et chaque prédicteur. On ordonne ces derniers selon la valeur décroissante du critère. On choisit les X premiers ou, si on veut avoir bonne conscience, on utilise un test d'hypothèses statistique pour sélectionner les variables ayant une liaison significative avec la variable à prédire.

Pour les approches INTEGREES, le processus de sélection fait partie de l'apprentissage. Les algorithmes d'induction d'arbres de décision les illustrent parfaitement. Le principal avantage est la cohérence. Par exemple, pour la méthode CART (Breiman et al., 1984), l'indice de Gini est utilisé pour choisir la variable de segmentation sur un nœud, l'objectif étant d'obtenir la partition finale la plus pure possible. Mais cohérence ne signifie pas performances. Or n'oublions pas qu'un des objectifs de la sélection est de produire un classifieur avec les meilleures capacités de généralisation. C'est pour cette raison d'ailleurs que CART introduit un dispositif de post-élagage basé sur un critère directement en rapport avec la performance, différent de celui utilisé lors de l'expansion de l'arbre.

L'idée de l'approche WRAPPER² est justement d'utiliser explicitement le critère de performance pour la recherche du sous-ensemble de prédicteurs pertinents. Le plus souvent, il s'agit du taux d'erreur. Mais en réalité, tout critère peut convenir. Cela peut être le coût si l'on introduit une matrice de coûts de mauvais classements ; cela peut être aussi l'aire sous la courbe

¹ <http://jmlr.csail.mit.edu/papers/special/feature03.html>

² <http://citeseer.ist.psu.edu/cache/papers/cs/1999/http:zSzzSzrobotics.stanford.eduzSz~ronnykzSzwappers-chapter.pdf/kohavi98wrapper.pdf/>

lorsqu'on évalue le classifieur à l'aide d'une courbe ROC ; etc. Dans ce cas, la méthode d'apprentissage agit comme une boîte noire. On lui présente différents groupes de variables prédictives, on choisira au final celui qui optimise le critère.

Stratégie de recherche. La stratégie de recherche des solutions joue un rôle très important dans la stratégie WRAPPER. Elle peut être très simple, avec des approches gloutonnes (ajouter ou retirer au fur et à mesure une variable à la solution courante), ou très élaborée, avec des approches basées sur des métaheuristiques (algorithmes génétiques, colonies de fourmis, etc.). En la matière, on se rend compte que le mieux est l'ennemi du bien. Trop explorer l'espace des solutions nous expose au sur apprentissage. Les approches gloutonnes simplistes conviennent dans la plupart des cas. En effet elles lissent naturellement le parcours de l'espace des solutions. Nous décrivons la méthode FORWARD, ajout séquentiel des variables, dans ce didacticiel.

Mode d'évaluation des performances. L'autre aspect important est l'estimation du critère à partir des données. Utiliser le fichier d'apprentissage pour sélectionner le meilleur sous-ensemble de variables prédictives nous expose au sur apprentissage, surtout si l'on souhaite utiliser le taux d'erreur. Il en serait autrement s'il pouvait mettre en balance la performance brute du modèle (le taux d'erreur par exemple) et la complexité du modèle (en nombre de variables par exemple). C'est ce que traduit, entre autres, le principe de la minimisation du risque structurel. D'autres critères, dans d'autres contextes, s'appuient sur un principe similaire. C'est le cas du critère AIC (Akaike) pour la régression.

Si l'on s'en tient au taux d'erreur, on pourrait subdiviser les données en échantillons d'apprentissage et de test : construire le modèle sur les premières données, évaluer les performances sur les secondes, en vue de sélectionner les variables pertinentes. Séduisante a priori, cette approche n'est pas exempte de reproches. L'échantillon test, censé constituer un juge impartial pour l'évaluation des performances, devient partie prenante dans l'apprentissage. Il est explicitement exploité pour choisir la meilleure solution. Il est de fait inutilisable pour estimer objectivement l'erreur en généralisation.

Il semble qu'une approche viable serait toujours de subdiviser les données en apprentissage et test, mais de se baser sur une méthode de ré échantillonnage (la validation croisée la plupart du temps) sur la première partie des données pour évaluer les différentes solutions et sélectionner celle qui paraît la plus pertinente. L'ensemble test ne sert que pour mesurer les performances du modèle finalement sélectionné. Ainsi, il joue le rôle qui lui est normalement dévolu : évaluer objectivement la performance sans prendre part ni à la construction, ni à la sélection des modèles.

Méthode d'apprentissage. La méthode d'apprentissage est utilisée comme une boîte noire dans le processus de sélection WRAPPER. Nous n'en exploitons pas les caractéristiques. N'importe quelle méthode conviendrait. Nous avons choisi la méthode « NAIVE BAYES » (modèle d'indépendance conditionnelle) pour plusieurs raisons : elle est bien adaptée aux variables prédictives catégorielles, c'est le cas de nos données ; elle n'intègre pas un processus

interne de sélection ; et elle est sensible aux variables non pertinentes. Ainsi, l'influence de la sélection de variables sur la performance sera particulièrement visible.

Objet du didacticiel. Dans ce didacticiel, nous implémentons la méthode WRAPPER avec les logiciels SIPINA et R. Pour ce dernier, le code écrit est le plus générique possible afin que le lecteur puisse comprendre chaque étape du processus de sélection et adapter le programme à d'autres données. Par ailleurs, la lecture attentive du code source pour R donne un meilleur éclairage sur les calculs réalisés en interne par SIPINA.

La stratégie WRAPPER est a priori la meilleure puisqu'elle optimise explicitement le critère de performance. Nous vérifierions cela en comparant les résultats avec ceux fournis par l'approche FILTRE (méthode FCBF) proposée dans TANAGRA. Nous verrons que les conclusions ne sont pas aussi tranchées qu'on pourrait le croire.

2 Données

Nous utilisons le fichier MUSHROOM.TXT dans ce didacticiel³, un fichier bien connu de la communauté de l'apprentissage automatique⁴.

La variable CLASSE est celle que l'on veut prédire, on cherche à savoir si un champignon est comestible ou non à partir de sa description (taille, couleur, etc. ; 22 variables prédictives candidates). Nous avons aléatoirement subdivisé les données en ensemble d'apprentissage (2000 observations) et de test (6124 observations). Ce dernier sert uniquement à évaluer les performances des modèles finaux proposés, il n'intervient en aucune manière lors du processus de sélection.

La colonne STATUT sert à indiquer le rôle de chaque observation.

3 Stratégie « wrapper » avec SIPINA

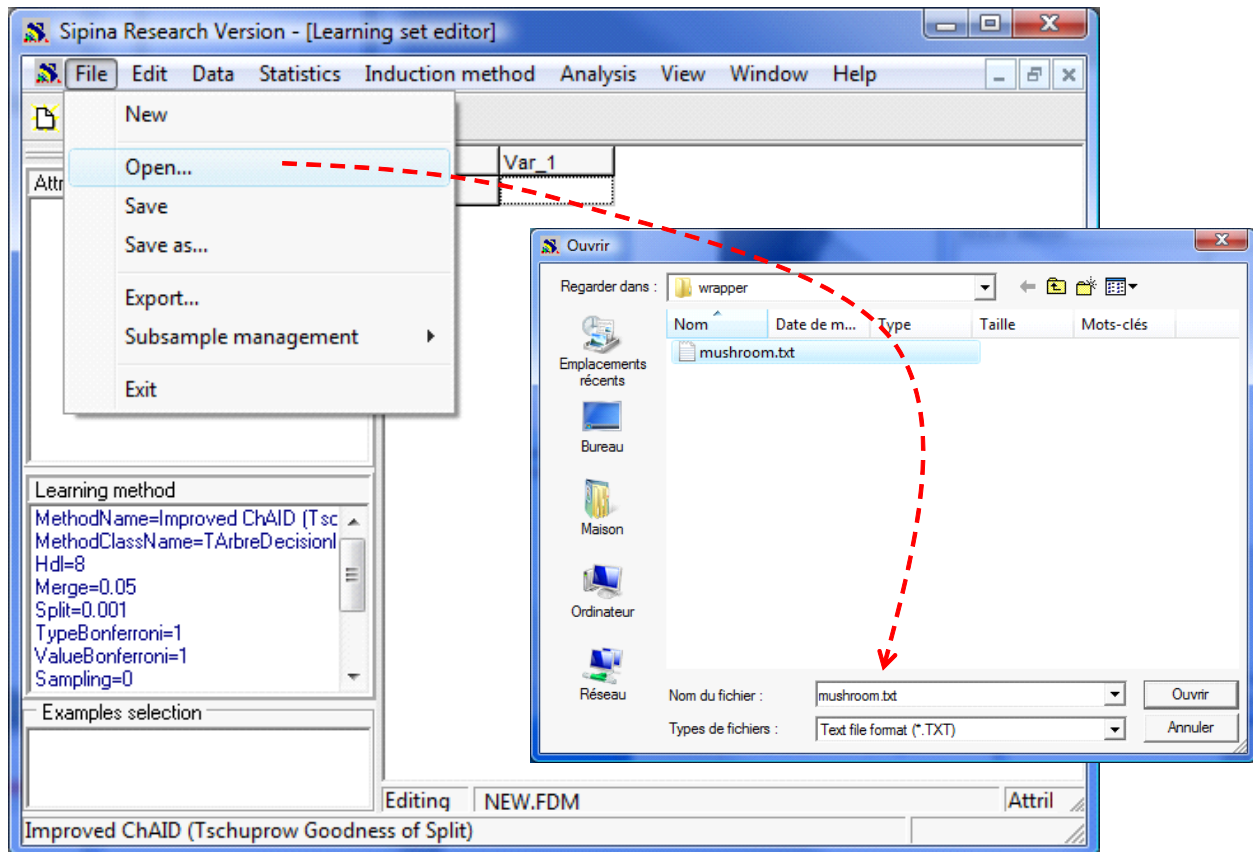
La stratégie « wrapper » est disponible dans SIPINA. Il faut la retrouver dans les dédales des menus. Mais une fois que l'on a trouvé la commande et défini les paramètres adéquats, le logiciel s'occupe de tout.

3.1 Importation des données

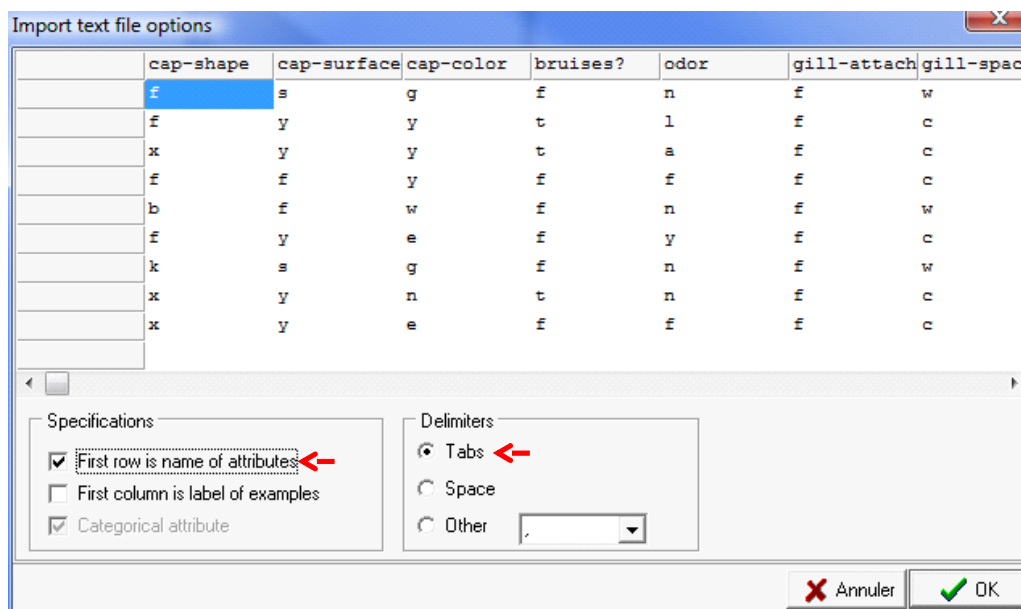
Après avoir lancé SIPINA, nous actionnons le menu FILE / OPEN. Nous sélectionnons le fichier MUSHROOM.TXT.

³ http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/mushroom_wrapper.zip

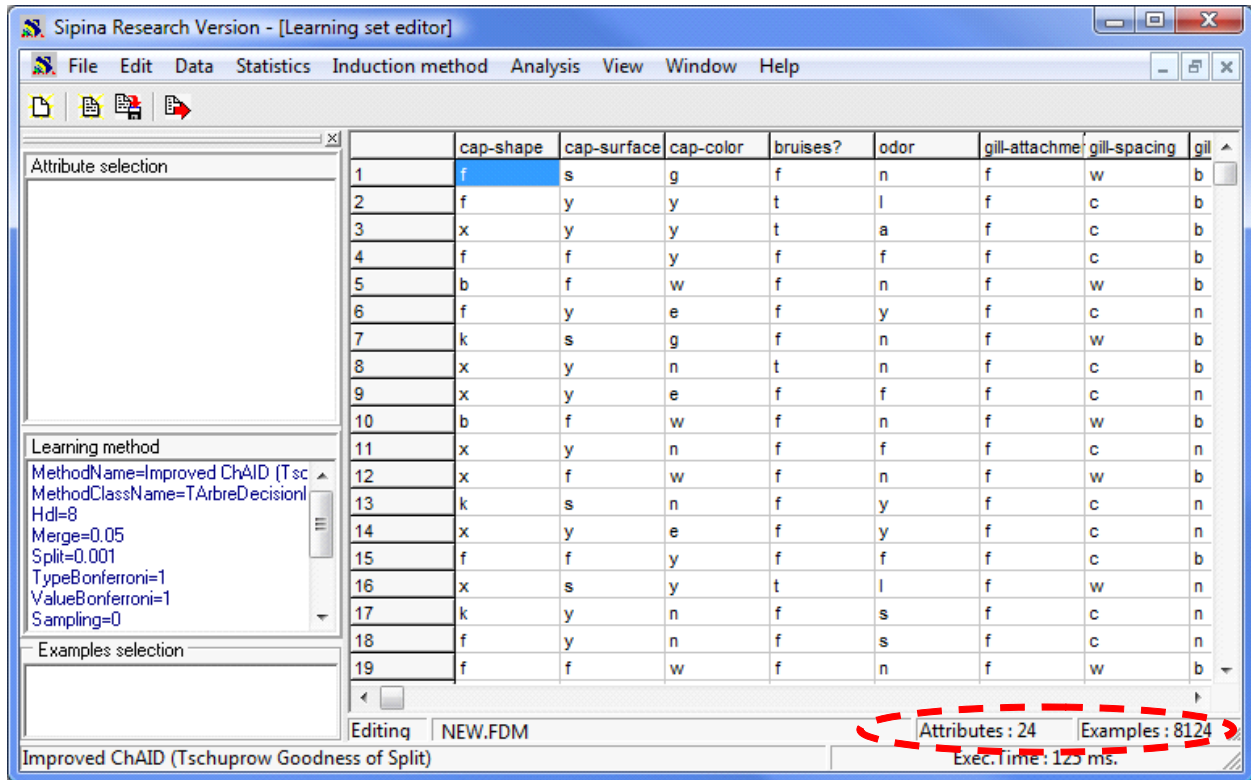
⁴ <http://archive.ics.uci.edu/ml/datasets/Mushroom>



Une boîte de dialogue apparaît. Nous spécifions la configuration du fichier. Dans notre cas, le séparateur est tabulation, la première ligne correspond aux noms de variables. Nous validons.



Le fichier est alors chargé. Les données sont visibles dans la grille d'affichage. Il y a bien 8124 observations et 24 colonnes : la CLASSE, la colonne STATUT et les 22 variables prédictives.

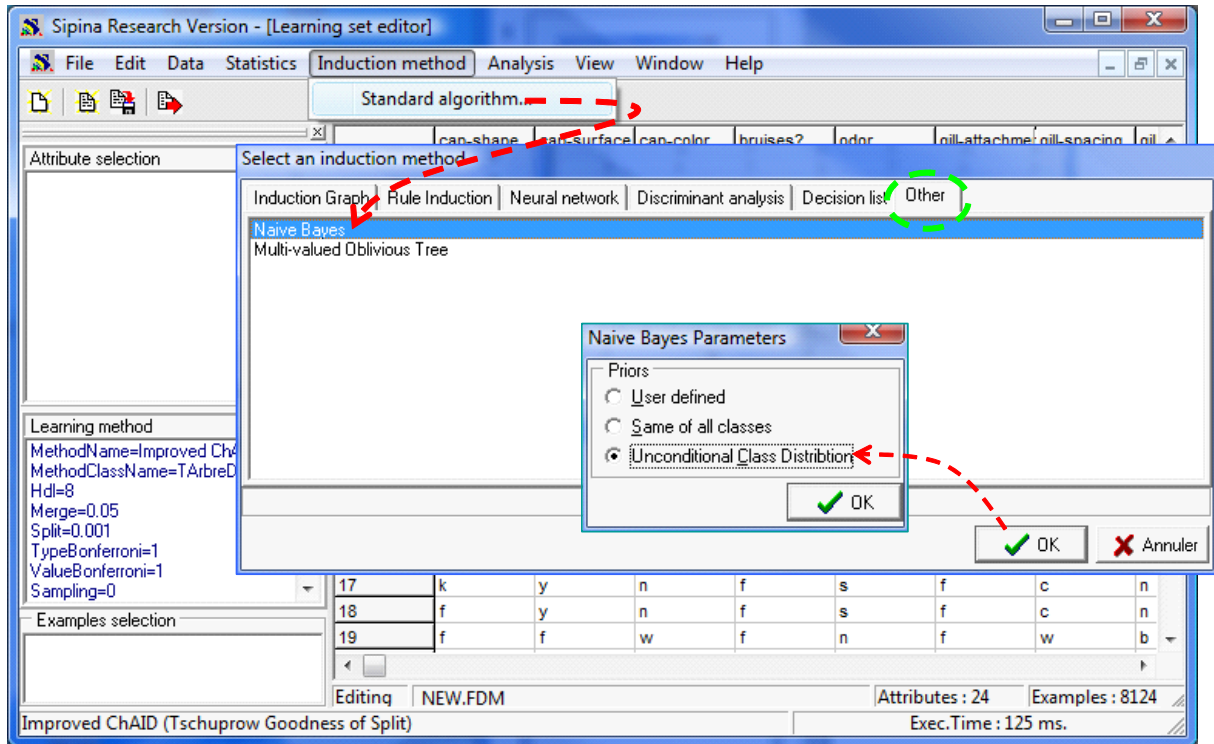


3.2 Choix de l'algorithme d'apprentissage

Nous souhaitons utiliser le modèle bayésien naïf (modèle d'indépendance conditionnelle). Il repose sur une hypothèse apparemment très restrictive : l'indépendance des descripteurs relativement aux valeurs de la classe. Pourtant il est assez performant par rapport aux autres approches. Plusieurs points de vue peuvent expliquer cela : s'il estime mal la probabilité conditionnelle d'affectation, il constitue en revanche un bon estimateur de son mode, et c'est ce qui importe lors du classement ; on se rend compte qu'il a un biais de représentation que l'on connaît bien, il s'agit d'un classifieur linéaire. Finalement, sa bonne tenue n'est guère une surprise si l'on prend en compte ces deux éléments.

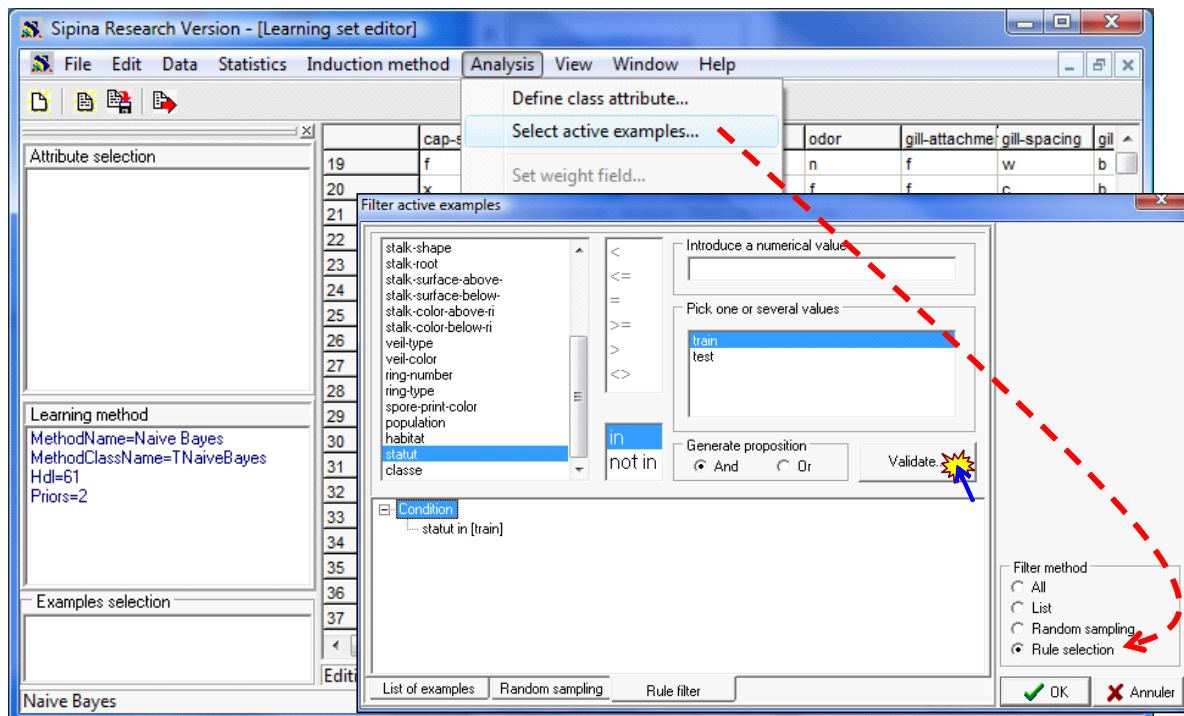
Pour définir la méthode d'apprentissage dans SIPINA, nous actionnons le menu INDUCTION METHOD / STANDARD ALGORITHM. Dans la boîte de dialogue qui apparaît, nous sélectionnons l'onglet OTHER. Nous choisissons la méthode NAIVE BAYES.

Après avoir cliqué sur OK, une boîte de paramétrage est affichée. Nous validons directement : la distribution de la variable à prédire est estimée sur les données d'apprentissage.

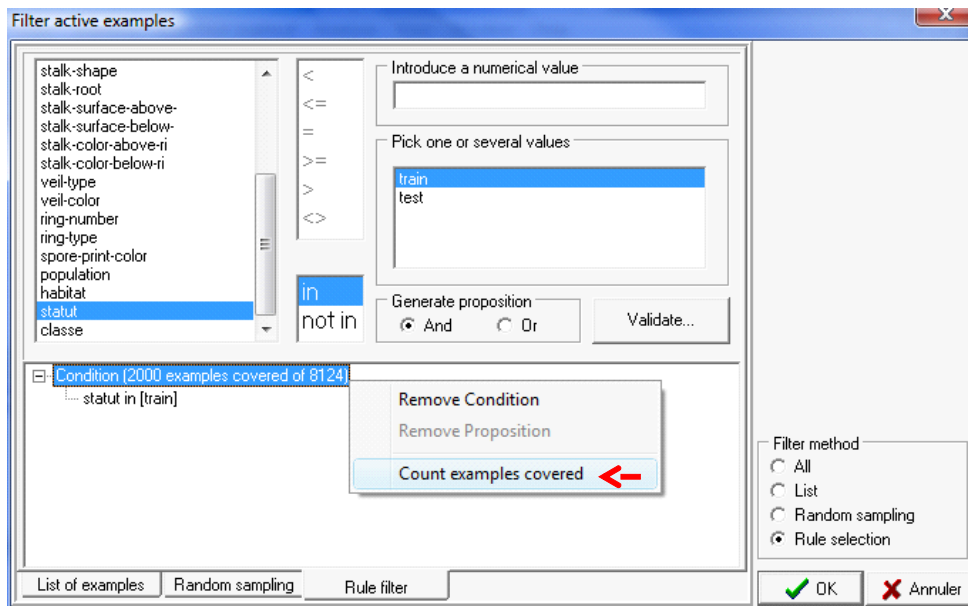


3.3 Partition des observations

Nous devons indiquer au logiciel, via la colonne STATUT, les individus de la partie apprentissage et ceux de la partie test. Nous actionnons le menu ANALYSIS / SELECT ACTIVE EXAMPLES. Nous sélectionnons l'option RULE SELECTION c.-à-d. le partitionnement sera basé sur une règle logique. Nous choisissons la règle [STATUT] IN [TRAIN] puis nous cliquons sur le bouton VALIDATE.

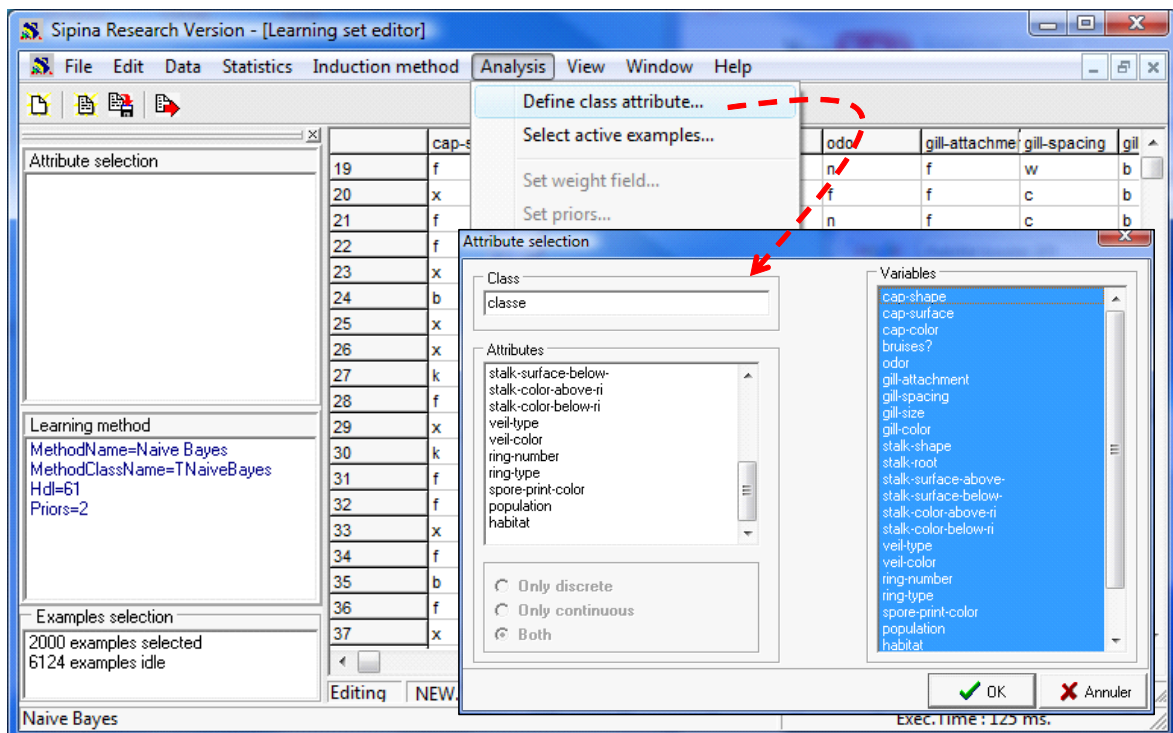


La règle est inscrite dans la partie basse de la fenêtre. Nous pouvons compter les observations couvertes par la règle en cliquant sur le menu contextuel COUNT EXAMPLES COVERED. SIPINA indique que 2000 observations sur les 8124 disponibles seront réservées à l'apprentissage.

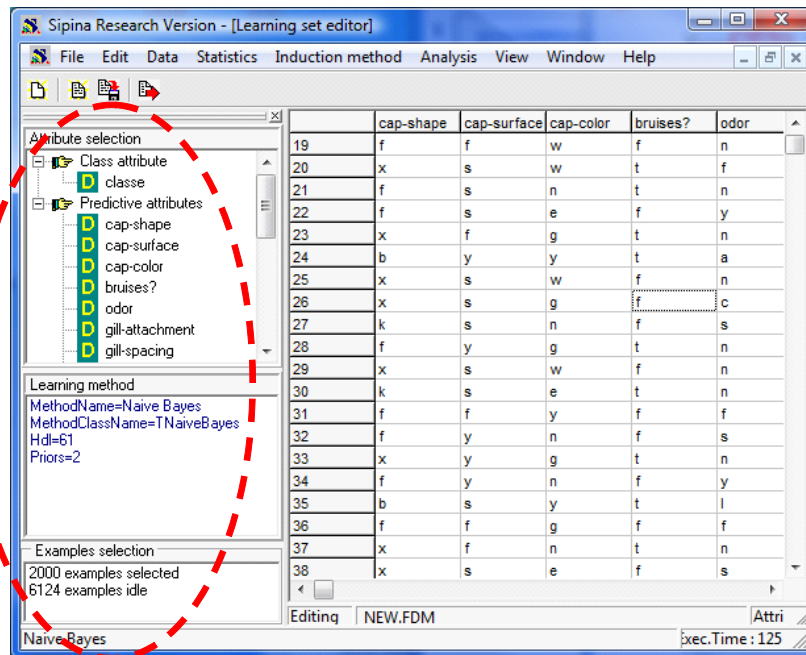


3.4 Définir le rôle des variables

L'étape suivante consiste à spécifier la variable à prédire et les variables prédictives. Nous cliquons sur le menu ANALYSIS / DEFINE CLASS ATTRIBUTE. Par glisser - déposer, nous indiquons le rôle de chaque variable. Bien évidemment, la variable STATUT n'intervient plus à ce stade.

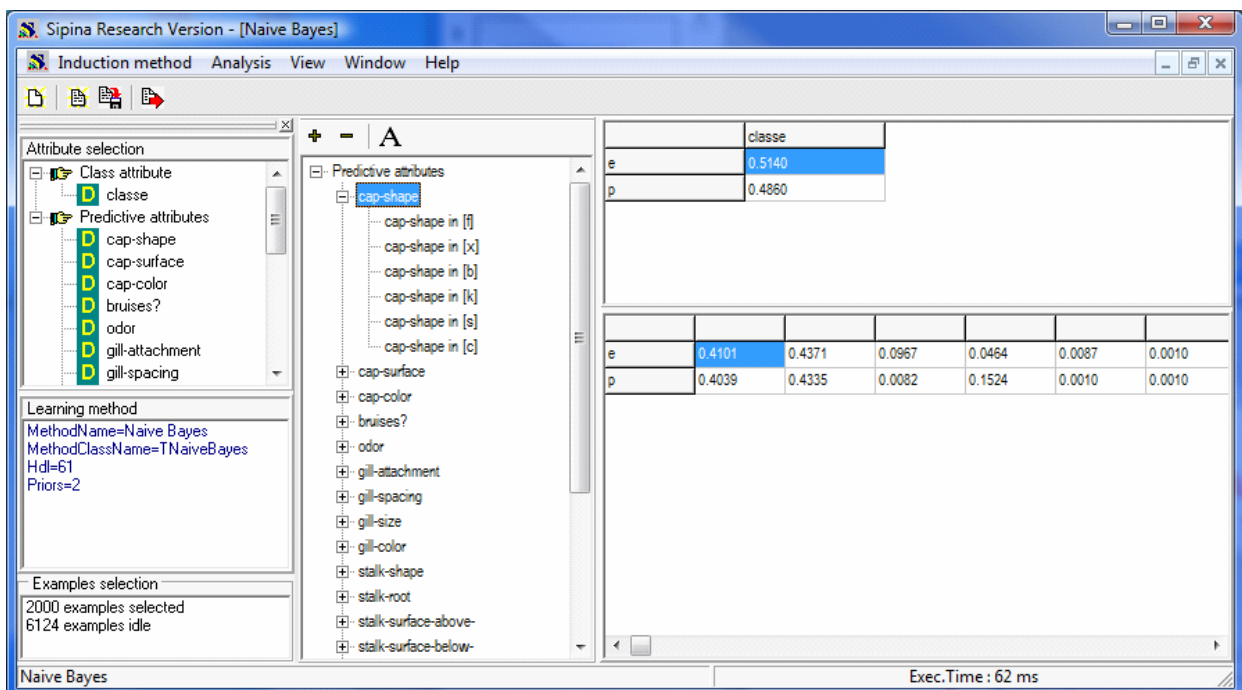


Un résumé de notre configuration apparaît dans la partie gauche de la fenêtre principale.

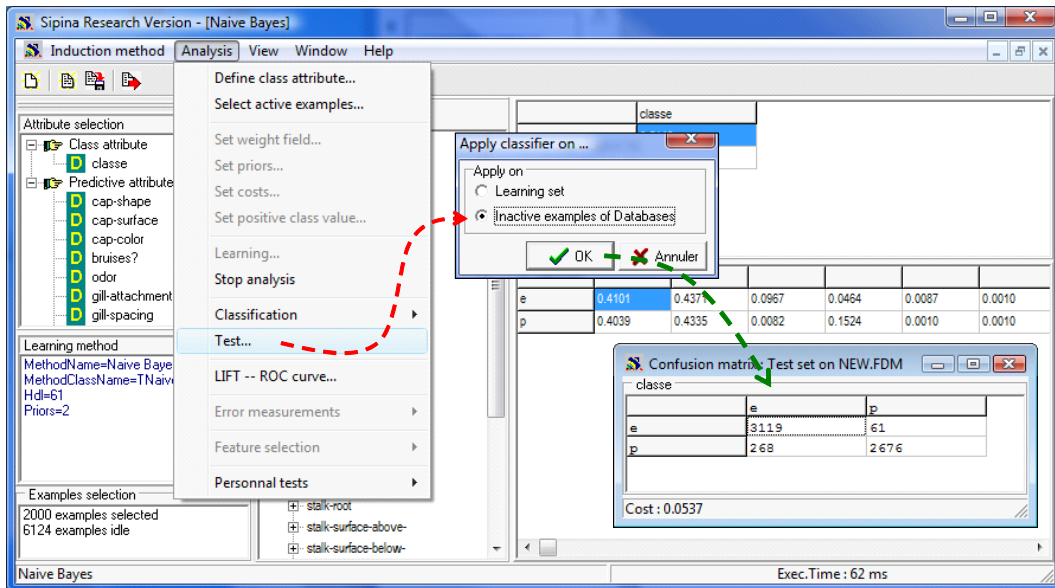


3.5 Performances du modèle complet – Apprentissage et test

Le modèle complet intégrant la totalité des variables prédictives servira de référence. Nous le construisons sur les données d'apprentissage, puis nous l'appliquons sur l'échantillon test. Pour lancer l'apprentissage, nous actionnons le menu ANALYSIS / LEARNING. Une fenêtre apparaît. Nous pouvons y lire les probabilités conditionnelles. Ils sont utilisés en interne pour les calculs. Ils sont affichés pour cette raison. Mais à vrai dire, ces tableaux sont très peu interprétables.



Nous passons directement à l'évaluation. Nous cliquons sur ANALYSIS / TEST. Dans la boîte de dialogue, nous veillons à sélectionner l'option INACTIVE EXAMPLES OF DATABASE. Ainsi, la matrice de confusion sera construite sur l'échantillon test.

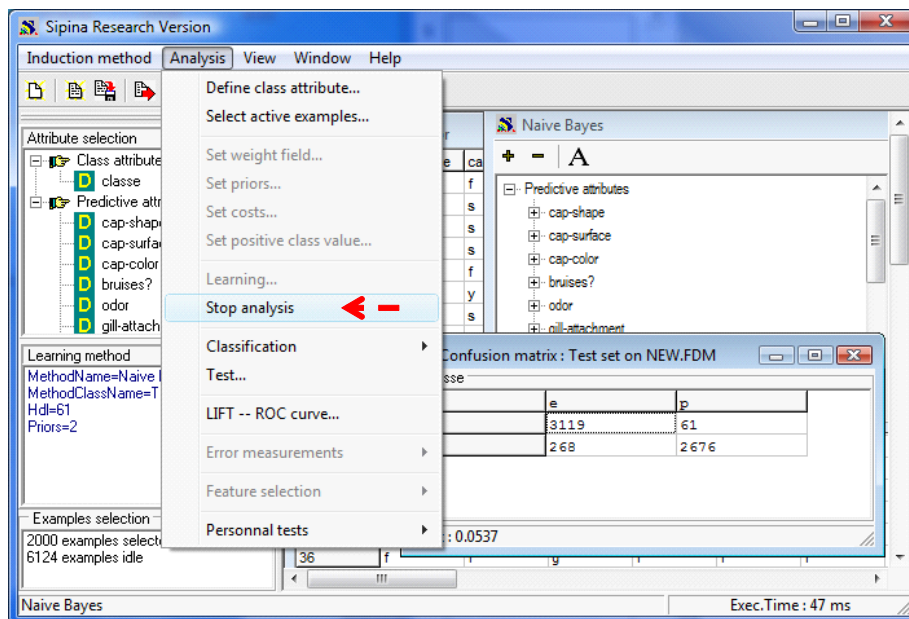


Une fenêtre affiche la matrice de confusion. Le taux d'erreur est indiquée dans la partie basse, il est de 5.37% dans notre exemple.

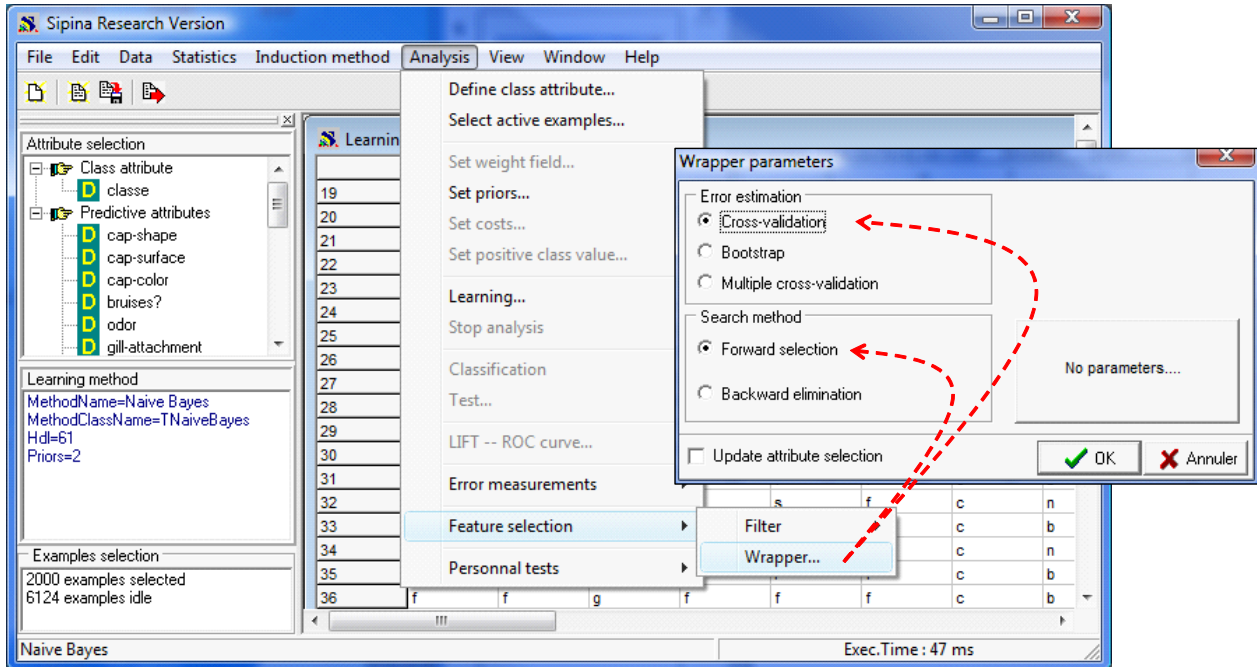
Voyons maintenant s'il est possible de faire mieux lorsque l'on introduit la stratégie WRAPPER de sélection de variables.

3.6 Stratégie « wrapper » pour la sélection de variables

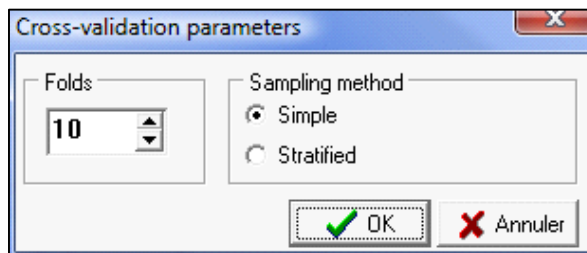
Nous devons au préalable stopper l'analyse en cours, nous cliquons sur ANALYSIS / STOP ANALYSIS. Les résultats précédents sont supprimés.



Nous actionnons le menu ANALYSIS / FEATURE SELECTION / WRAPPER pour lancer la procédure. Dans la boîte de paramétrage, parmi les méthodes de ré échantillonnage disponibles, nous choisissons la validation croisée pour mesurer le taux d'erreur. Nous réalisons une sélection par avant (FORWARD) c.-à-d. la procédure cherche la première variable la plus intéressante, celle-ci étant entérinée, elle cherche la seconde meilleure variable, etc.



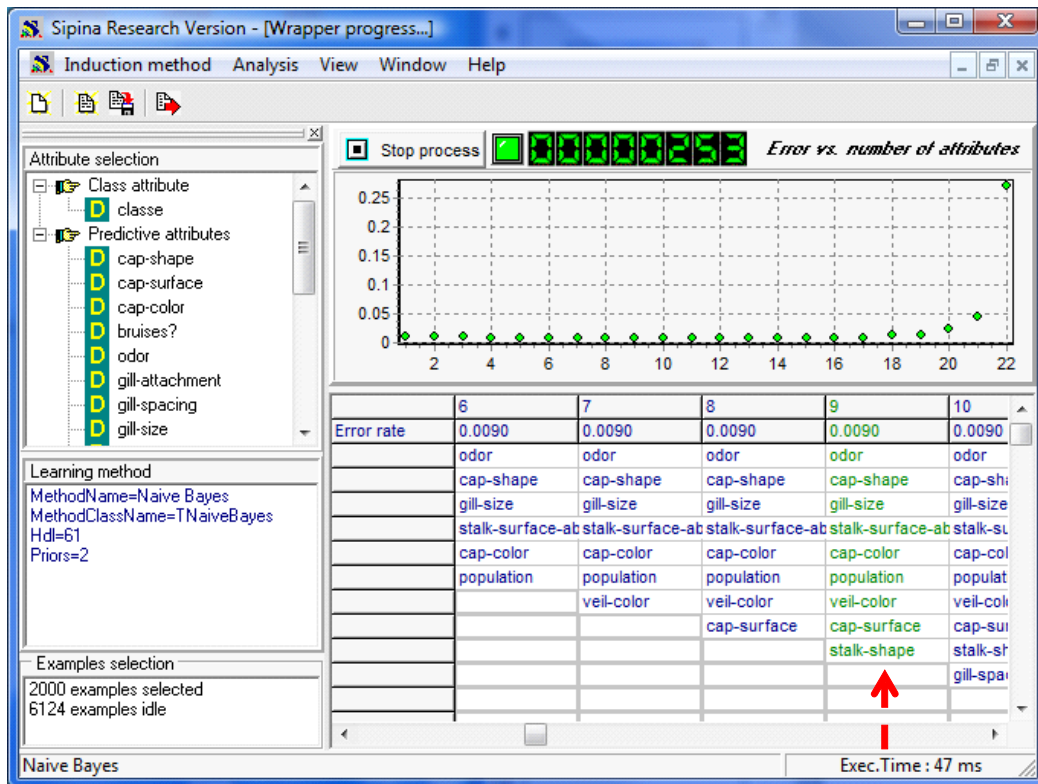
En cliquant sur OK, une seconde boîte arrive, le nombre de portions dans la validation croisée est 10. Nous validons.



Les calculs démarrent. Ils sont relativement rapides compte tenu de la quantité d'opérations à réaliser. SIPINA indique le taux d'erreur à chaque étape c.-à-d. lorsqu'une variable est ajoutée à la solution courante.

L'utilisateur peut stopper le processus s'il juge les calculs trop longs.

Attention, d'un coup sur l'autre, d'une machine à l'autre, les résultats peuvent être différents puisque la méthode de ré échantillonnage s'appuie sur un générateur de nombres aléatoires indexé sur l'horloge de la machine. Il est initialisé de manière différente à chaque fois.



Pour ce coup-ci, SIPINA a trouvé une solution à 9 variables. Elle est mise en évidence par la couleur verte de la colonne associée dans la grille.

Remarque : En étudiant attentivement la courbe d'erreur, on constate que la zone d'optimalité est un plateau assez large. La solution s'est démarquée à un epsilon près. Choisir la solution à 6 variables n'aurait pas été choquant. On pourrait imaginer introduire une préférence à la simplicité en mettant en place un dispositif qui ressemblerait à la règle de l'écart type de l'algorithme CART, par analogie avec ce que réalise celui-ci lors du post élagage d'un arbre de décision. La question reste ouverte.

Pour l'heure, nous sélectionnons la solution à 9 variables.

Nous cliquons sur la grille avec le bouton droit de la souris en veillant à être sur la bonne colonne, nous actionnons le menu contextuel CHOOSE THIS SUBSET FEATURE. Dans le bandeau à gauche de la fenêtre principale, la liste des variables prédictives a été modifiée.

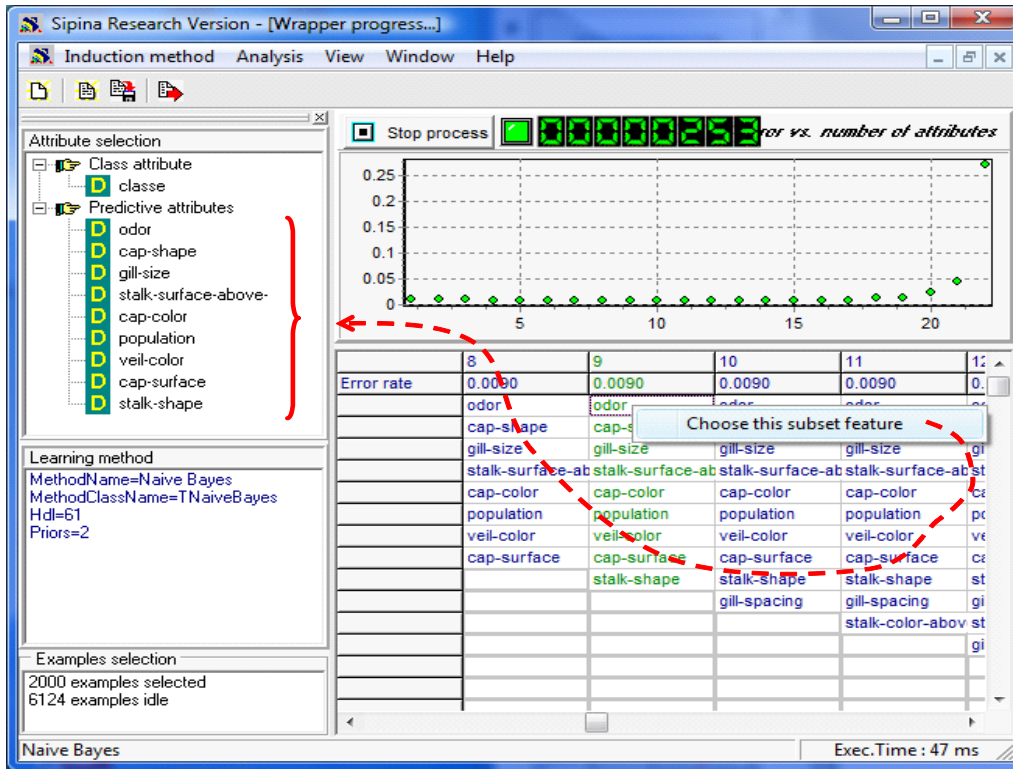
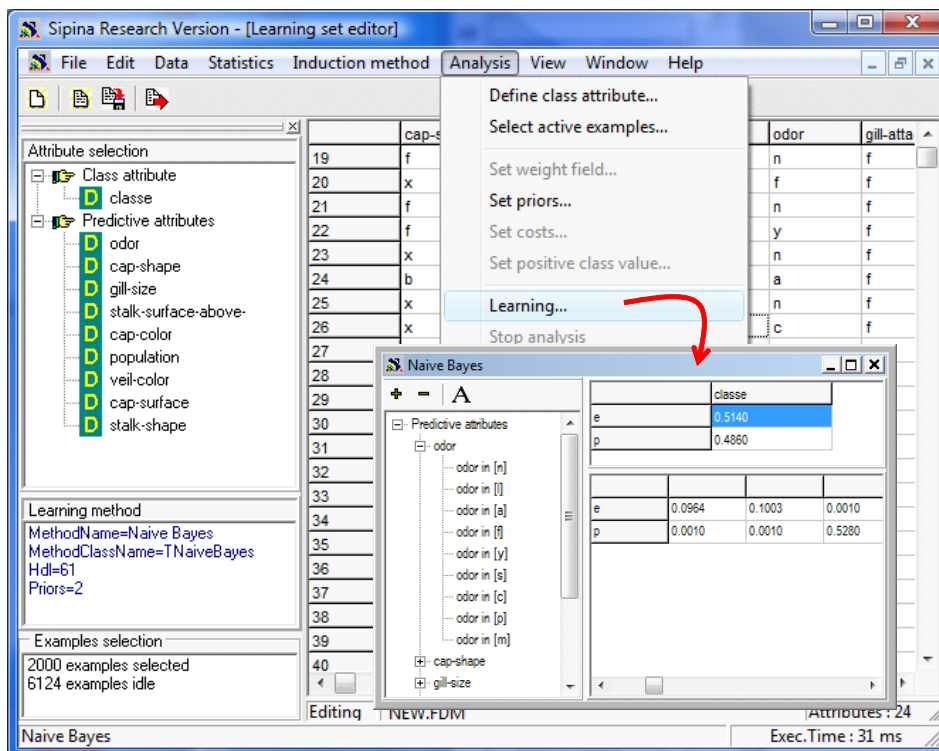


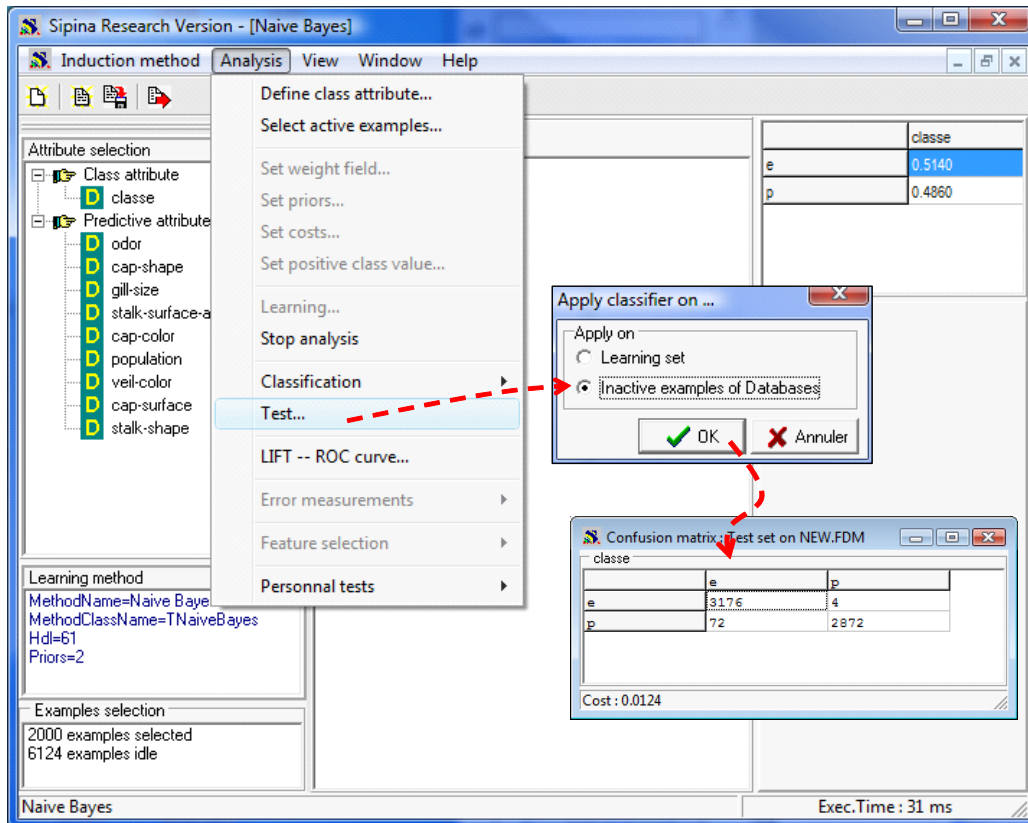
Figure 1 - Détail des étapes - Stratégie « WRAPPER »

3.7 Performances du modèle simplifié

Il ne nous reste plus qu'à évaluer cette solution en réitérant la séquence apprentissage - test : ANALYSIS / LEARNING construit le modèle prédictif.



ANALYSIS / TEST avec l'option INACTIVE EXAMPLES OF DATABASE construit la matrice de confusion sur l'échantillon test.



Plus spectaculaire, c'est difficile. Le taux d'erreur est passé à 1.24%, largement en deçà de la solution incluant toutes les variables prédictives. Le modèle simplifié est réellement plus performant.

3.8 Analyse des solutions

Revenons un instant sur la fenêtre du WRAPPER (Figure 1).

Une grande partie des solutions sont assez équivalentes finalement. Même la première, n'incluant qu'une seule variable prédictive, propose déjà des performances honorables. C'est à partir de l'étape n°20 que les performances se dégradent singulièrement. Les trois dernières variables sélectionnées sont perturbatrices. L'introduction d'une procédure de sélection de variables est totalement justifiée dans ce contexte.

Ici commence le travail de l'expert métier. Si on connaît un peu le domaine, on saura expliquer pourquoi ces variables posent autant problème. Il pourra également nous donner des indications précieuses sur le choix définitif des variables prédictives.

4 Stratégie « wrapper » avec R

L'intérêt de SIPINA est que tout est préparé à l'avance. Il suffit de savoir cliquer au bon endroit pour lancer la procédure. Mais ce n'est pas sans dangers. Il arrive parfois que l'on clique sans trop savoir et ne pas trop saisir la teneur des résultats.

Dans ce didacticiel, nous avons programmé la stratégie « wrapper » dans le logiciel R. En traçant le code source, le lecteur pourra comprendre les différentes étapes et démonter le mécanisme qui est programmé dans les logiciels spécialisés.

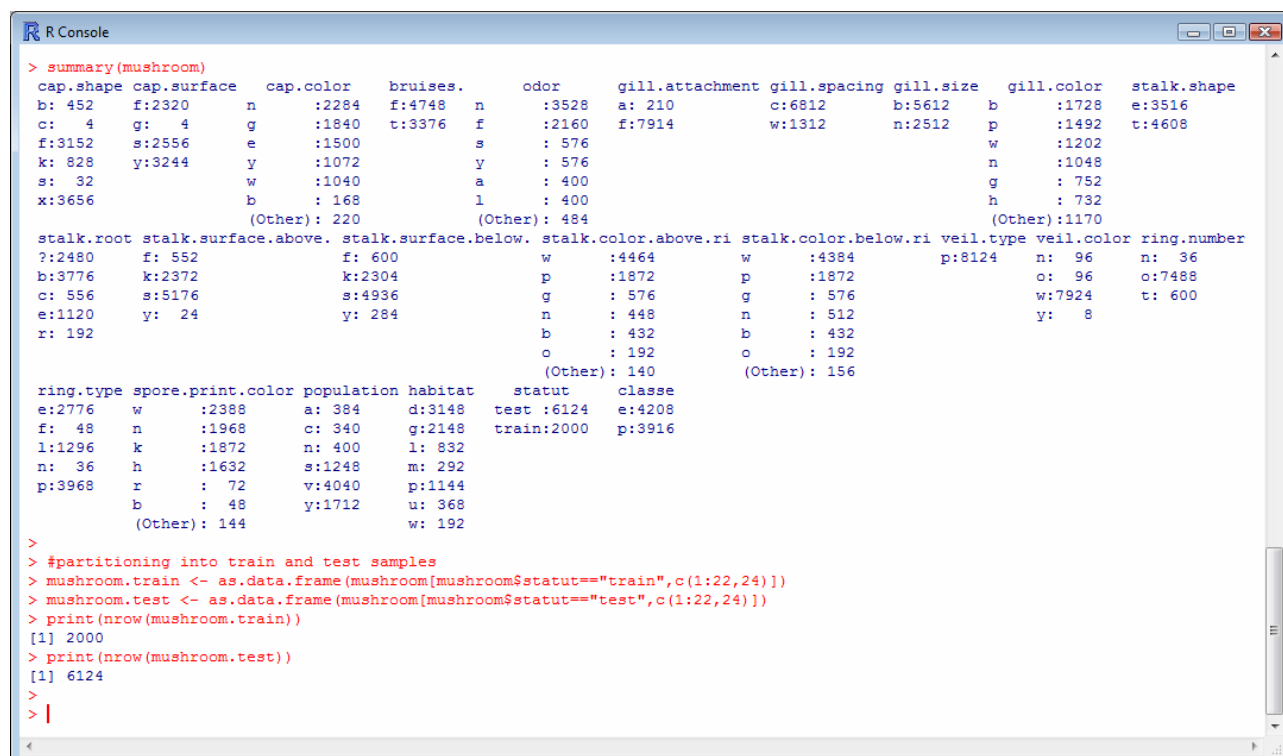
4.1 Lecture et partition des données

Dans un premier temps, nous chargeons les données et nous le partitionnons.

```
#load the data file
mushroom <- read.table(file="mushroom.txt",header=TRUE,sep="\t")
summary(mushroom)

#partitioning into train and test samples
mushroom.train <- as.data.frame(mushroom[mushroom$statut=="train",c(1:22,24)])
mushroom.test <- as.data.frame(mushroom[mushroom$statut=="test",c(1:22,24)])
print(nrow(mushroom.train))
print(nrow(mushroom.test))
```

R nous affiche les informations suivantes.



```
R Console
> summary(mushroom)
  cap.shape cap.surface  cap.color  bruises.  odor  gill.attachment gill.spacing gill.size  gill.color  stalk.shape
b: 452    f:2320      n    :2284  f:4748  n    :3528  a: 210      c:6812      b:5612  b    :1728  e:3516
c:   4     g:   4      g    :1840  t:3376  f    :2160  f:7914      w:1312      n:2512  p    :1492  t:4608
f:3152  s:2556      e    :1500  s    : 576
k: 828    y:3244      y    :1072  y    : 576
s:   32     w    :1040  a    : 400
x:3656      b    : 168  l    : 400
              (Other): 220      (Other): 484
              (Other):1170
stalk.root stalk.surface.above. stalk.surface.below. stalk.color.above.ri stalk.color.below.ri veil.type veil.color ring.number
?:2480     f: 552      f: 600      w    :4464  w    :4384  p:8124  n: 96  n: 36
b:3776     k:2372      k:2304      p    :1872  p    :1872  o: 96  o:7488
c: 556     s:5176      s:4936      g    : 576  g    : 576  w:7924  t: 600
e:1120     y: 24      y: 284      n    : 448  n    : 512  y: 8
r: 192
              (Other): 140      (Other): 156
ring.type spore.print.color population habitat  statut  classe
e:2776    w    :2388  a: 384  d:3148  test :6124  e:4208
f: 48     n    :1968  c: 340  g:2148  train:2000  p:3916
l:1296    k    :1872  n: 400  l: 832
n: 36     h    :1632  s:1248  m: 292
p:3968    r    : 72   v:4040  p:1144
              b    : 48   y:1712  u: 368
              (Other): 144      w: 192

>
> #partitioning into train and test samples
> mushroom.train <- as.data.frame(mushroom[mushroom$statut=="train",c(1:22,24)])
> mushroom.test <- as.data.frame(mushroom[mushroom$statut=="test",c(1:22,24)])
> print(nrow(mushroom.train))
[1] 2000
> print(nrow(mushroom.test))
[1] 6124
>
> |
```

2000 observations sont réservées pour l'apprentissage ; les autres, 6124, sont dédiées au test.

4.2 Apprentissage et test – La procédure Naive Bayes du package RWeka

La méthode « bayésien naïf » n'est pas directement implémentée dans le logiciel R. Nous allons mettre à contribution le package⁵ RWeka (<http://cran.r-project.org/web/packages/RWeka/index.html>) basé sur le logiciel libre Weka (<http://www.cs.waikato.ac.nz/ml/weka/>) qui fait référence au sein de la communauté de l'apprentissage automatique.

La procédure **library(.)** charge le package. Mais la méthode n'est pas disponible pour autant. En effet, seules quelques techniques sont directement accessibles (régression logistique, SVM, etc.). Nous devons charger explicitement le « bayésien naïf » à l'aide de la procédure **make_Weka_classifier(.)** pour pouvoir en disposer. La signature des fonctions dédiées à l'apprentissage supervisé est heureusement standardisée. Nous appelons **NB(.)** cette nouvelle fonction.

Il ne nous reste plus qu'à lancer l'apprentissage ; puis évaluer le modèle sur l'échantillon test à l'aide de la fonction **evaluate_Weka_classifier(.)**.

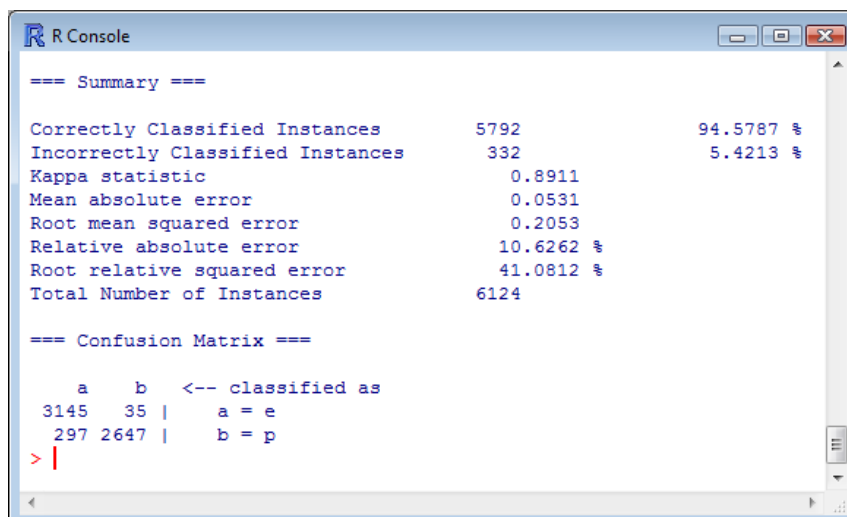
```
#load the RWeka package
library(RWeka)

#make naive bayes classifier function
NB <- make_Weka_classifier("weka/classifiers/bayes/NaiveBayes")

#training phase
full.model <- NB(classe ~ ., data = mushroom.train)

#evaluation on the test set
test.evaluation <- evaluate_Weka_classifier(full.model,newdata=mushroom.test)
print(test.evaluation)
```

Nous obtenons.



```

R Console

=== Summary ===

Correctly Classified Instances      5792      94.5787 %
Incorrectly Classified Instances    332       5.4213 %
Kappa statistic                    0.8911
Mean absolute error                 0.0531
Root mean squared error            0.2053
Relative absolute error            10.6262 %
Root relative squared error        41.0812 %
Total Number of Instances          6124

=== Confusion Matrix ===

   a   b  <-- classified as
3145  35 |   a = e
 297 2647 |   b = p
> |

```

⁵ Voir <http://tutoriels-data-mining.blogspot.com/2009/05/installation-des-packages-sous-r.html> pour l'installation d'un nouveau package dans R.

La fonction affiche une série d'informations, dont la matrice de confusion. Nous noterons pour notre part que le taux d'erreur en test est 5.24% lorsque l'on utilise la totalité des variables prédictives.

4.3 Stratégie « wrapper » sous R

Nous programmons sous R la stratégie « wrapper » basée sur une recherche « forward ». Il y a certainement plusieurs manières de s'y prendre. Le code est relativement simple lorsqu'on arrive à le décomposer judicieusement.

Nous nous sommes appuyés pour notre part sur la procédure **sapply(.)** de R pour explorer les solutions alternatives. Nous lui passons une fonction « callback » qui consiste à évaluer l'adjonction d'une variable prédictive supplémentaire à la solution courante. Elle renvoie le taux d'erreur⁶.

```
#callback function
#searching the best predictive attribute
#for one step of the wrapper process
search_wrapper <- function(x,cur.dataset,K){
  #current dataset
  working <- cbind(cur.dataset,x)
  #learning the model
  cur.model <- NB(as.formula(paste(colnames(working)[1],"~ .")),data = working)
  #evaluating the model
  cur.eval <- evaluate_Weka_classifier(cur.model,numFolds = K,seed=100)
  #getting the error rate
  return (cur.eval$details["pctIncorrect"])
}
```

Remarquons l'appel de la fonction **evaluate_Weka_classifier(.)**. A la place d'un échantillon spécifique, nous lui passons en paramètre le nombre de portions **K** de la validation croisée. En aucune manière, nous n'exploitons l'échantillon test à ce stade. D'un autre côté, même si nous exploitons exclusivement l'échantillon d'apprentissage, nous évitons l'écueil du sur ajustement⁷ en nous basant sur un indicateur, le taux d'erreur en validation croisée, autrement moins biaisé que le taux d'erreur en resubstitution.

Nous pouvons passer à l'écriture de la fonction qui se charge d'explorer les solutions successives, partant du modèle à une variable jusqu'au modèle incluant la totalité des variables.

```
#programming the whole wrapper framework for feature subset selection
#forward selection process
#dataset is the available data (train set only, we do not use the test set here)
```

⁶ Une stratégie alternative fondée sur des boucles imbriquées a également été implémentée. Elle est distribuée avec ce didacticiel.

⁷ <http://en.wikipedia.org/wiki/Overfitting>

```

#we assume that the last column is the class attribute
#K is the number of fold in the cross-validation
wrapper <- function(dataset,K=10){
  #number of predictive attribute
  P <- ncol(dataset)-1
  #predictives dataset
  predictives <- dataset[1:(ncol(dataset)-1)]
  #classe attribute
  cur.dataset <- dataset[ncol(dataset)]
  #current formula
  cur.formula <- paste(colnames(dataset)[P+1],"~")
  #ordered index of the selected variable
  output <- c()
  #error rate at each step
  error <- c()
  #the whole process
  for (p in 1:P){
    #error rate for this step
    cur.error <- sapply(predictives,search_wrapper,cur.dataset,K)
    #getting the best error rate
    id.min <- which.min(cur.error)
    selected.name <- colnames(predictives)[id.min]
    #adding the id. of the column into the selection
    output[p] <- which.min(match(colnames(dataset),selected.name))
    #add the attribute to the current dataset
    cur.dataset <- cbind(cur.dataset,predictives[id.min])
    #removing the attribute to the predictive attributes
    predictives <- predictives[-id.min]
    #filling out the vector of errors
    error[p] <- min(cur.error)
  }
  #returning the error rate and the index of variables
  return (list(error=error,output=output))
}

```

Détailler le code serait fastidieux, voyons-en les grandes lignes et les extensions possibles :

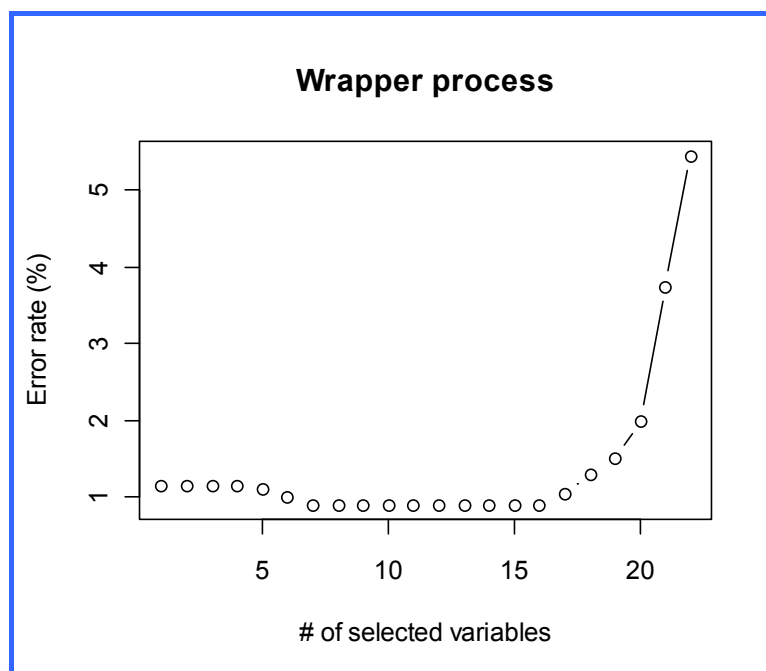
- La fonction prend en entrée les données d'apprentissage, on considère que la variable à prédire est en dernière position.
- La seconde entrée est le nombre de portions K de la validation croisée. Par défaut, sa valeur est 10.
- On peut facilement envisager d'introduire en paramètre la méthode d'apprentissage. La fonction n'en serait que plus générique.
- La stratégie part du modèle trivial, sans variable prédictive, jusqu'au modèle complet, incluant toutes les variables. C'est le sens de la boucle **for (p in 1 :P)**.
- L'appel de **sapply(.)** est destiné à collecter dans un vecteur les performances des variables prédictives que l'on peut ajouter à l'étape courante.
- **which.min(.)** récupère l'index de la variable qui minimise le taux d'erreur. Il est ajouté à la sélection courante (**output**), nous récupérons également le taux d'erreur associé (**error**).

Il ne nous reste plus qu'à lancer la fonction.

```
#launching the wrapper process
mushroom.wrapper <- wrapper(mushroom.train)

#plotting the error rate according the number of selected variables
plot(mushroom.wrapper$error,type="b",main="Wrapper process",ylab="Error rate (%)",xlab="# of selected variables")
```

Nous affichons tout d'abord la courbe des taux d'erreur. Elle n'est pas sans rappeler celle produite par SIPINA.



Les solutions sont pour la plupart quasiment équivalentes ; à partir de 7 variables, nous sommes dans une zone assez large d'optimalité ; l'adjonction des 3 dernières variables dégrade fortement les performances.

Nous souhaitons maintenant sélectionner la solution optimale et lister les variables associées.

```
#getting the number of selected variable
nb.sel <- which.min(mushroom.wrapper$error)

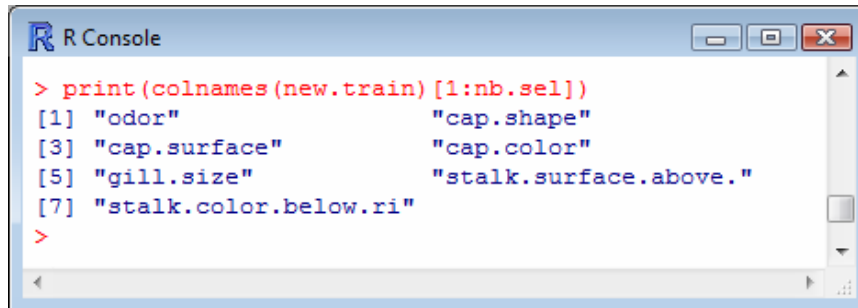
#new dataset -- train and test
new.train <-
mushroom.train[,c(mushroom.wrapper$output[1:nb.sel],ncol(mushroom.train))]
new.test <-
mushroom.test[,c(mushroom.wrapper$output[1:nb.sel],ncol(mushroom.test))]

#printing the name of the selected variables
print("Selected variables : ")
```

Tanagra

```
print(colnames(new.train)[1:nb.sel])
```

La solution à 7 variables s'avère être la meilleure, avec :



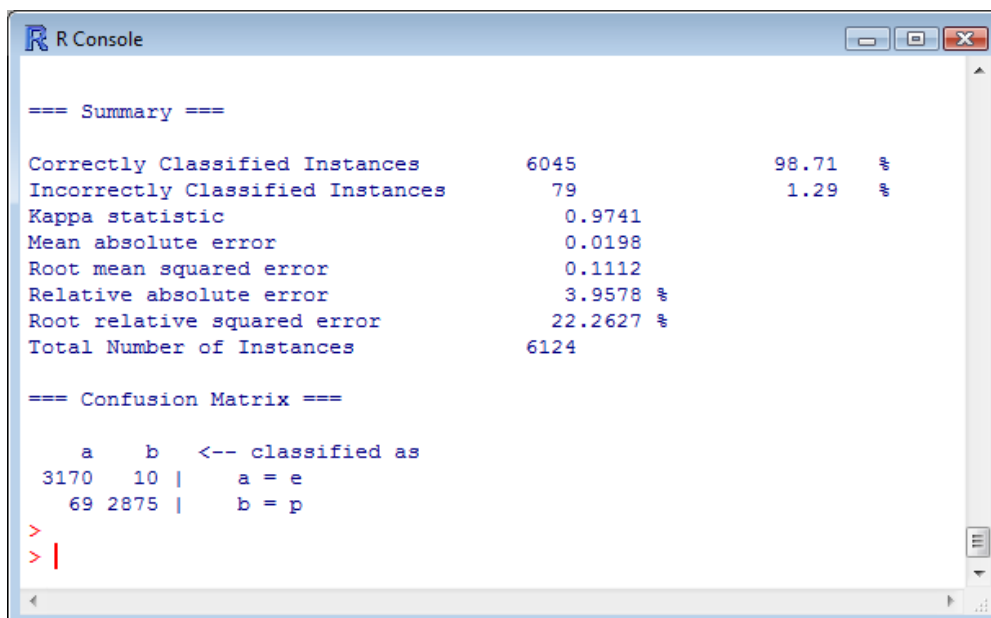
```
R Console
> print(colnames(new.train)[1:nb.sel])
[1] "odor" "cap.shape"
[3] "cap.surface" "cap.color"
[5] "gill.size" "stalk.surface.above."
[7] "stalk.color.below.ri"
>
```

4.4 Apprentissage et test du modèle « optimal »

Dernière étape du processus, nous évaluons sur l'échantillon test les performances du modèle bayésien naïf intégrant ces 7 variables prédictives.

```
#training and test on the selected variables
new.model <- NB(classe ~., data = new.train)
new.eval <- evaluate_Weka_classifieur(new.model, newdata=new.test)
print(new.eval)
```

Le taux d'erreur en test est de 1.29%.



```
R Console
=== Summary ===
Correctly Classified Instances      6045      98.71 %
Incorrectly Classified Instances     79       1.29 %
Kappa statistic                     0.9741
Mean absolute error                  0.0198
Root mean squared error              0.1112
Relative absolute error              3.9578 %
Root relative squared error          22.2627 %
Total Number of Instances           6124

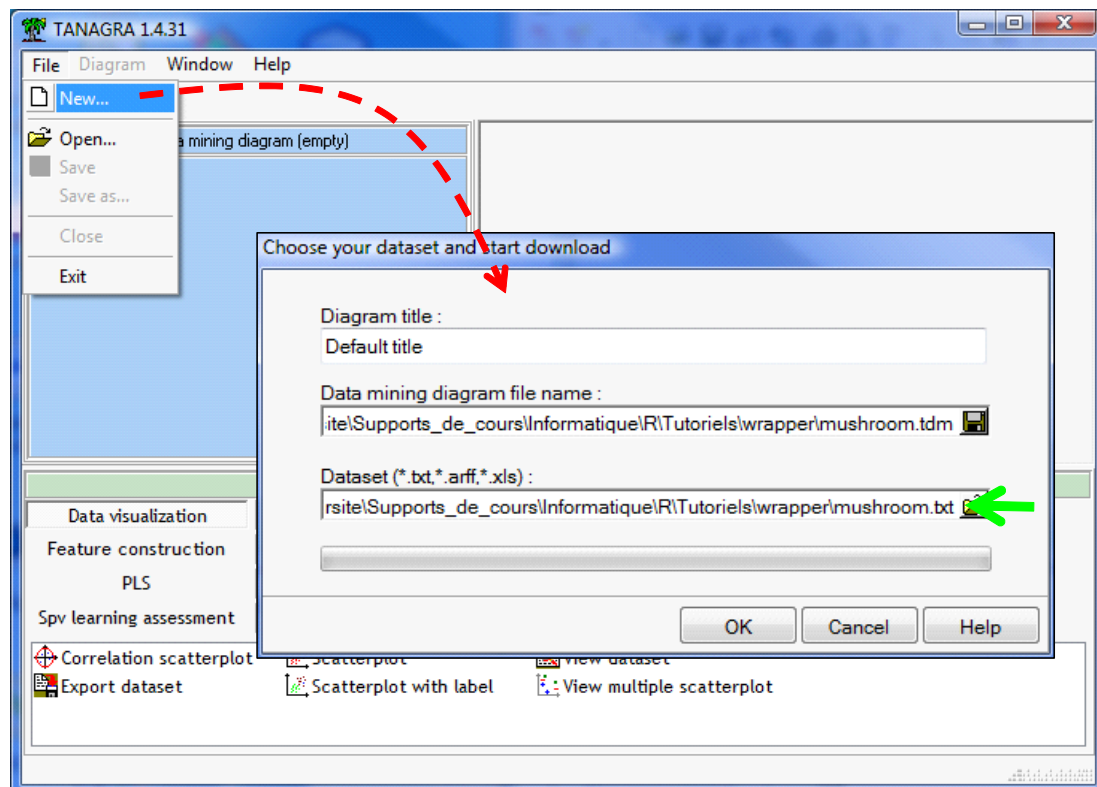
=== Confusion Matrix ===
   a  b  <-- classified as
3170 10 |   a = e
 69 2875 |   b = p
>
> |
```

5 La stratégie FILTRE avec TANAGRA

Pour des raisons trop longues à détailler ici, la méthode WRAPPER n'est pas (*pas encore, version 1.4.31*) disponible dans TANAGRA. En revanche, plusieurs approches FILTRE sont proposées. L'atout est la rapidité, surtout concernant l'algorithme FCBF (Yu et Liu, 2003 ; <http://www.public.asu.edu/~huanliu/FCBF/FCBFsoftware.html>) que nous souhaitons évaluer. FCBF agit en amont. Mais, comme nous l'évoquions plus haut, rien ne nous garantit que la solution proposée soit adaptée pour la méthode d'apprentissage que nous lui collerons en aval, en particulier le classifieur bayésien naïf. La seule manière de vérifier est de comparer les performances sur le même ensemble de données. C'est ce que nous allons faire.

5.1 Importer les données

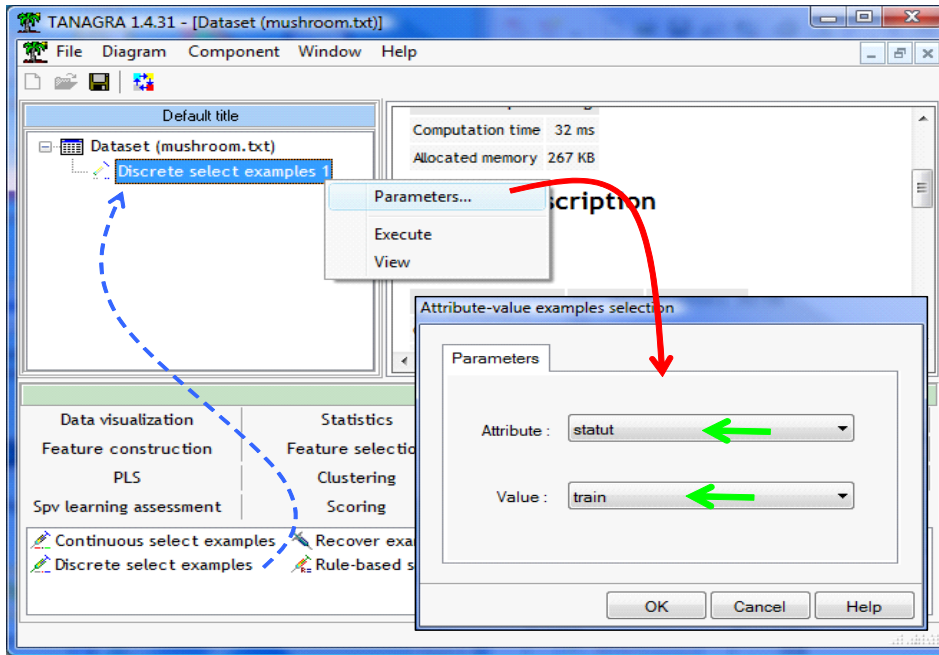
Après avoir lancé TANAGRA, nous créons un nouveau diagramme en actionnant le menu FILE / NEW. Nous sélectionnons le fichier MUSHROOM.TXT.



TANAGRA nous indique que 24 colonnes et 8124 observations ont bien été importées.

5.2 Partitionnement apprentissage – test

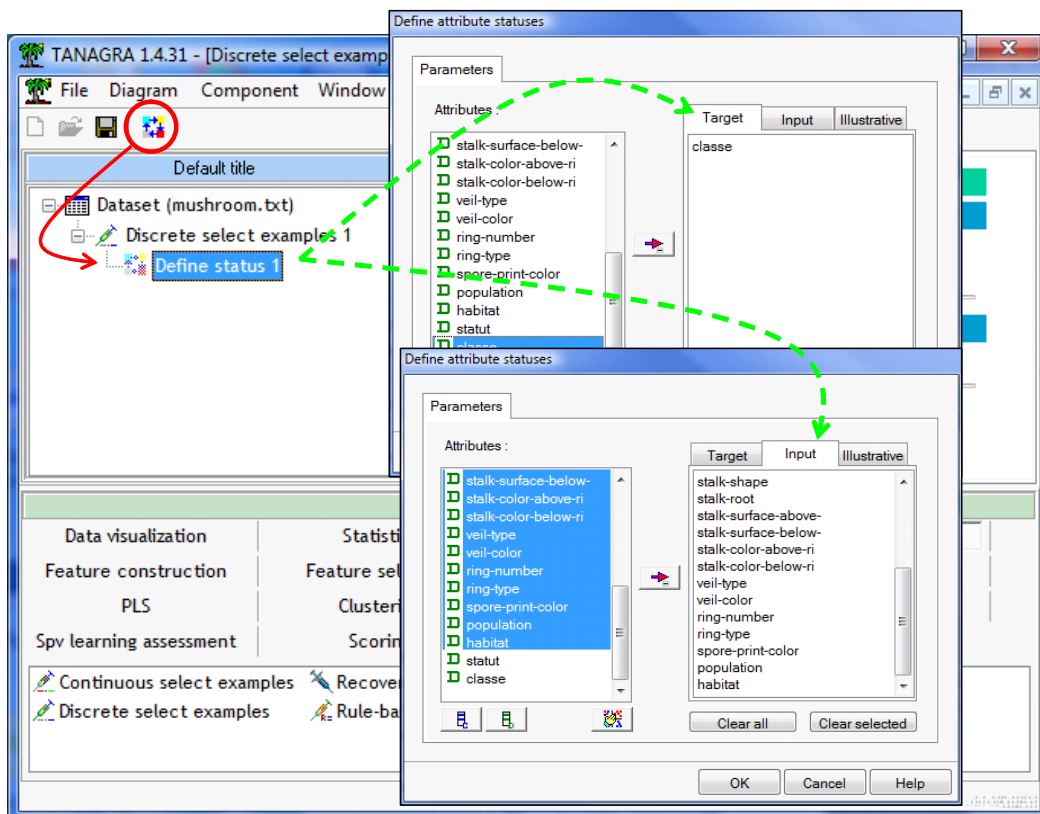
Nous exploitons la colonne STATUT pour partitionner les données. Pour ce faire, nous insérons le composant DISCRETE SELECT EXAMPLES (onglet INSTANCE SELECTION) dans le diagramme. Nous actionnons le menu contextuel PARAMETERS et nous spécifions les paramètres suivants.



Après avoir validé, nous cliquons sur VIEW. TANAGRA nous indique que 2000 observations sont réservées à l'apprentissage.

5.3 Sélection de variables avec FCBF

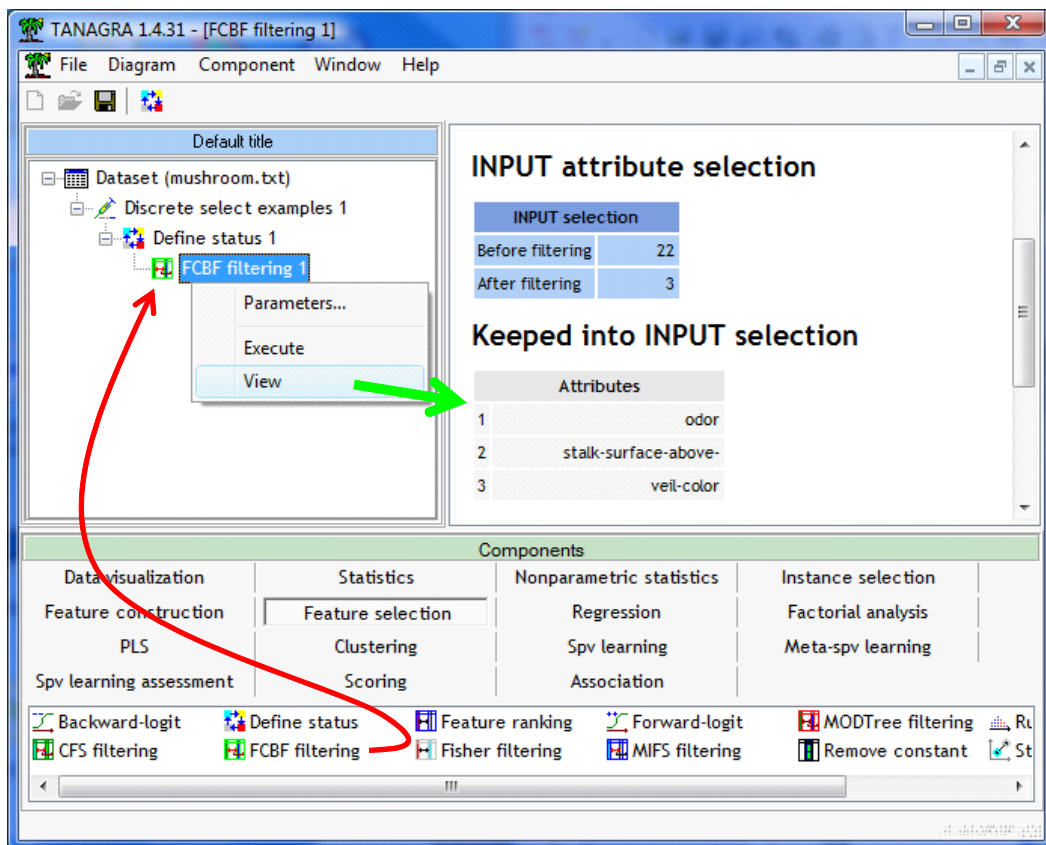
Pour mettre en place le processus de sélection automatique, nous devons au préalable indiquer les variables prédictives candidates à l'aide du composant DEFINE STATUS (barre d'outils).



En TARGET, nous mettons la variable à prédire CLASSE ; en INPUT, toutes les autres variables, à l'exception de STATUT bien sûr.

Nous pouvons insérer à la suite l'outil de sélection de variables FCBF (onglet FEATURE SELECTION). Notons au passage que cette méthode est très rapide, elle peut traiter à la volée plusieurs milliers de prédicteurs candidats. Autre particularité, elle sait gérer la redondance entre les attributs. Si deux variables sont strictement identiques parmi les candidats, seule l'une d'entre elles apparaîtra dans la solution proposée.

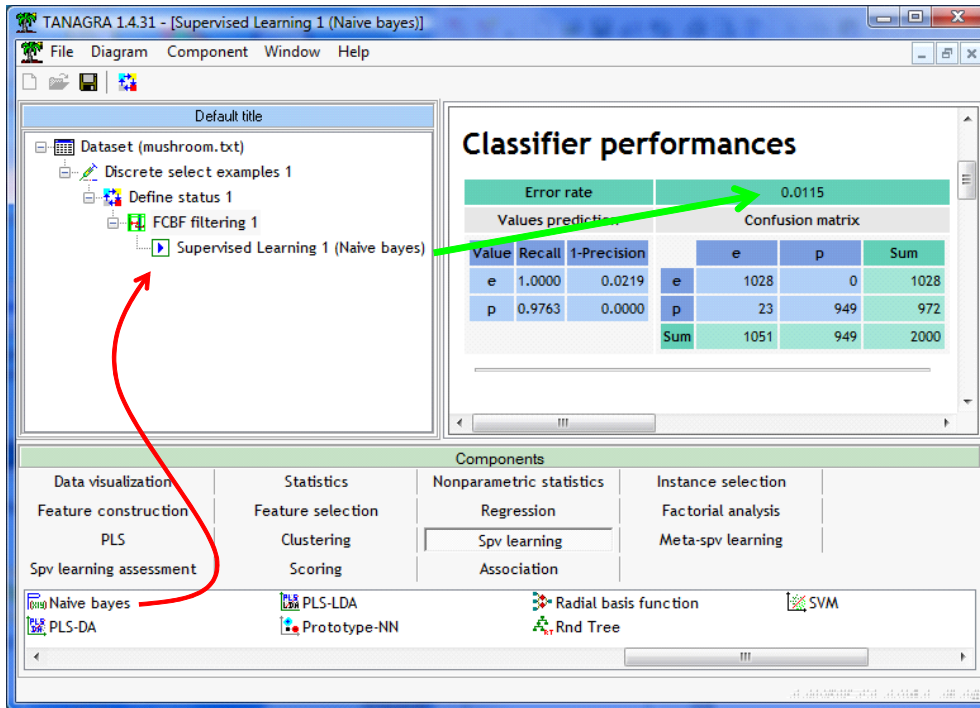
Nous cliquons sur VIEW.



3 variables seulement ont été retenues. Il s'agit de ODOR, STALK SURFACE ABOVE et VEIL COLOR. Dorénavant, en sortie du composant FCBF sont automatiquement sélectionnées : en TARGET, la variable à prédire CLASSE ; en INPUT, les 3 variables ci-dessus. Nous pouvons donc directement introduire une méthode d'apprentissage quelconque en aval.

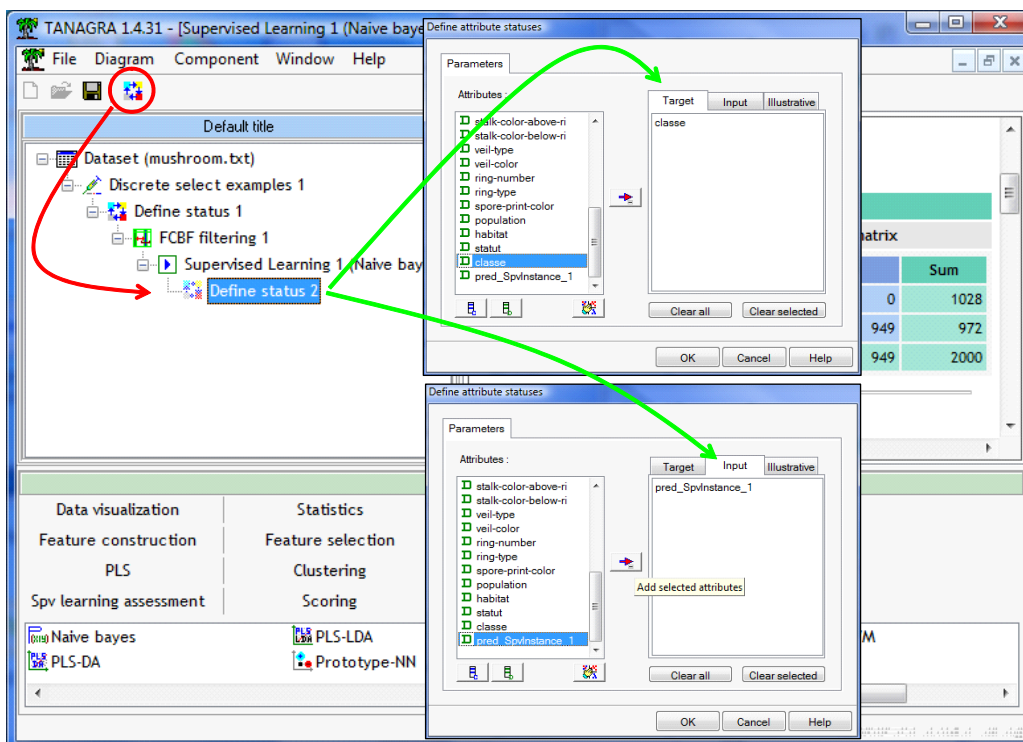
5.4 Apprentissage avec NAIVE BAYES

Nous plaçons le composant NAIVE BAYES (onglet SPV LEARNING). Nous cliquons sur VIEW. Nous obtenons, entre autres, la matrice de confusion en resubstitution.



5.5 Evaluation sur l'échantillon test

Passons directement à l'étape suivante, l'évaluation sur l'échantillon test. Nous introduisons le composant DEFINE STATUS, nous plaçons en TARGET la variable CLASSE, en INPUT la variable PRED_SPV_INSTANCE_1 automatiquement produite par le composant d'apprentissage supervisé. Il contient la prédiction du modèle, à la fois sur les données d'apprentissage et sur les données de test.



Nous insérons enfin le composant TEST (onglet SPV LEARNING ASSESSMENT). Il confronte les valeurs observées de la variable à prédire et celles prédites par le modèle. Par défaut, il est paramétré pour réaliser les opérations sur les observations non sélectionnées c.-à-d. l'échantillon test. C'est ce que nous voulons. Nous cliquons sur VIEW.

The screenshot shows the TANAGRA 1.4.31 interface. On the left, a workflow diagram includes components like 'Dataset (mushroom.txt)', 'Discrete select examples 1', 'Define status 1', 'FCBF filtering 1', 'Supervised Learning 1 (Naive bayes)', 'Define status 2', and 'Test 1'. A red arrow points from 'Test 1' to the 'Test' component in the 'Components' panel at the bottom. The 'Results' panel on the right shows the 'Error rate' as 0.0158, which is circled in red. Below it is a confusion matrix table.

Value	Recall	1-Precision	e	p	Sum
e	1.0000	0.0296	3180	0	3180
p	0.9671	0.0000	97	2847	2944
Sum			3277	2847	6124

Le taux d'erreur en test est de 1.58%, à comparer avec les 1.29% obtenus avec la méthode WRAPPER dans R, ou les 1.24% de SIPINA.

Les écarts, calculés sur 6124 observations, paraissent faibles. Mais gardons nous bien de trancher définitivement. Plusieurs points de vue peuvent prévaloir. En tous les cas, ces taux sont directement comparables : la démarche d'expérimentation et les échantillons (apprentissage / test) sont exactement les mêmes pour les différents logiciels.

6 Conclusion

L'objectif de ce didacticiel est de montrer et confronter deux stratégies de sélection de variables dans le cadre de l'apprentissage supervisé.

Nous avons présenté dans un premier temps la stratégie WRAPPER. Elle est directement disponible dans SIPINA. Puis nous avons détaillé les calculs en la programmant dans R. WRAPPER est censée être la meilleure stratégie de sélection puisqu'elle utilise explicitement le critère de performance pour déterminer le sous-ensemble de variables « optimal » pour la prédiction.

Dans un deuxième temps, nous avons souhaité confronter WRAPPER avec l'approche FILTRE. Cette dernière agit en amont, avec un critère ad hoc, indépendamment de toute méthode d'apprentissage supervisée. Nous avons utilisé la méthode FCBF implémentée dans TANAGRA.

La conclusion est assez contrastée. Certes, la stratégie WRAPPER, tant dans SIPINA que dans R, est meilleure que FILTRE. Mais les écarts de performances relativement faibles justifient-ils le surcroît de calcul ? Sur une base « benchmark » comme MUSHROOM, la procédure est quasi instantanée, ce n'est pas un problème. Mais sur une base réelle comportant plusieurs centaines de variables candidates, la question mérite d'être posée.