1 Objectif

Utiliser la validation croisée pour l'évaluation des arbres de décision avec R, Knime et RapidMiner.

Ce didacticiel reprend un de nos anciens articles consacrés à la mise en œuvre de la validation croisée pour l'évaluation des performances des arbres de décision[1]. Nous comparions la démarche à suivre et la lecture des résultats pour Tanagra, Orange et Weka.

Dans ce document, nous étendons le descriptif aux logiciels R 2.7.2 (http://www.r-project.org/), Knime 1.3.51 (http://www.knime.org/) et RapidMiner Community Edition (http://rapid-i.com/content/blogcategory/38/69/).

Les objectifs et le cheminement sont les mêmes. Le lecteur peut se reporter à notre précédent didacticiel s'il souhaite avoir des précisions sur ces éléments. Nous utilisons le fichier HEART.TXT (UCI IRVINE, http://archive.ics.uci.edu/ml/datasets/Heart+Disease). L'objectif est de prédire l'occurrence des maladies cardio-vasculaires (COEUR). Le fichier a été nettoyé, le nombre de descripteurs a été réduit (12 variables prédictives), il en est de même pour les observations (270 individus).

2 Validation croisée avec R (package rpart)

Ces dernières années, le logiciel R a pris un essor considérable. Ses qualités sont connues, il offre des possibilités d'analyses incommensurables ; ses défauts aussi, il faut avoir de solides notions en programmation pour en tirer parti convenablement. Cela se vérifiera de nouveau dans cette étude.

Chargement de la bibliothèque rpart. Nous souhaitons utiliser la méthode rpart du package éponyme dans ce didacticiel. Il nous faut donc l'installer, puis le charger avec la commande **library(.)**.

```
#charger la bibliothèque rpart
library(rpart)
```

Chargement des données. Nous chargeons maintenant le fichier HEART.TXT[2]. Les données sont intégrées dans un data.frame que nous appelons **donnees**. Nous calculons, essentiellement pour vérifier le bon déroulement de l'importation, les statistiques descriptives avec **summary(.)**.

```
#charger les données
setwd("D:/DataMining/Databases_for_mining/comparison_TOW/validation_croisee")
donnees <- read.table(file="heart.txt",dec=".",header=TRUE)
summary(donnees)
```

Nous obtenons.

[1] Voir http://tutoriels-data-mining.blogspot.com/2008/04/arbres-de-dcision-avec-orange-tanagra.html, ou: http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/fr_Tanagra_TOW_Decision_Tree.pdf pour un accès direct au PDF.

[2] http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/heart.txt ; en y intégrant le code source pour R, http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/heart.zip

```
      age              sexe      type_douleur    pression       cholester
 Min.   :29.00   feminin : 87   A: 20        Min.   : 94.0   Min.   :126.0
 1st Qu.:48.00   masculin:183   B: 42        1st Qu.:120.0   1st Qu.:213.0
 Median :55.00                  C: 79        Median :130.0   Median :245.0
 Mean   :54.43                  D:129        Mean   :131.3   Mean   :249.7
 3rd Qu.:61.00                               3rd Qu.:140.0   3rd Qu.:280.0
 Max.   :77.00                               Max.   :200.0   Max.   :564.0
 sucre    electro    taux_max      angine      depression        pic
 A:230    A:131    Min.   : 71.0   non:181   Min.   : 0.0   Min.   :1.000
 B: 40    B:  2    1st Qu.:133.0   oui: 89   1st Qu.: 0.0   1st Qu.:1.000
          C:137    Median :153.5             Median : 8.0   Median :2.000
                   Mean   :149.7             Mean   :10.5   Mean   :1.585
                   3rd Qu.:166.0             3rd Qu.:16.0   3rd Qu.:2.000
                   Max.   :202.0             Max.   :62.0   Max.   :3.000
 vaisseau      coeur
 A:160    absence :150
 B: 58    presence:120
 C: 33
 D: 19
```

Nous constatons, entres autres, que la variable à prédire CŒUR possède deux modalités (présence et absence), avec respectivement 150 et 120 observations. Il y a 270 individus en tout. Sur un aussi petit fichier, partitionner les données en « échantillon apprentissage » et « échantillon test » n'est pas une bonne idée. Il faut passer par les méthodes de ré échantillonnage.

Création de l'arbre de décision sur la totalité des données. Dans un premier temps, nous construisons l'arbre sur la totalité des données avec la méthode rpart(.). Nous conservons les paramètres par défaut.

```
#construire et imprimer l'arbre calculé sur la totalité des individus
arbre.full <- rpart(coeur ~ ., data = donnees, method = "class")
print(arbre.full)
```

Nous obtenons l'arbre de décision de décision suivant.

```
1) root 270 120 absence (0.55555556 0.44444444)
  2) type_douleur=A,B,C 141  29 absence (0.79432624 0.20567376)
    4) depression< 19.5 125  19 absence (0.84800000 0.15200000)
      8) age< 55.5 76   5 absence (0.93421053 0.06578947) *
      9) age>=55.5 49  14 absence (0.71428571 0.28571429)
       18) cholester< 245.5 21   1 absence (0.95238095 0.04761905) *
       19) cholester>=245.5 28  13 absence (0.53571429 0.46428571)
         38) sexe=feminin 16   2 absence (0.87500000 0.12500000) *
         39) sexe=masculin 12   1 presence (0.08333333 0.91666667) *
    5) depression>=19.5 16   6 presence (0.37500000 0.62500000) *
  3) type_douleur=D 129  38 presence (0.29457364 0.70542636)
    6) vaisseau=A 61  28 absence (0.54098361 0.45901639)
     12) depression< 7 30   7 absence (0.76666667 0.23333333) *
     13) depression>=7 31  10 presence (0.32258065 0.67741935)
       26) angine=non 12   4 absence (0.66666667 0.33333333) *
       27) angine=oui 19   2 presence (0.10526316 0.89473684) *
    7) vaisseau=B,C,D 68   5 presence (0.07352941 0.92647059) *
```

Erreur en resubstitution. Nous souhaitons connaître les performances de cet arbre appliqué sur les données d'apprentissage c.-à-d. les données ayant servi à la construction du modèle, la totalité du fichier dans notre cas. Nous formons la matrice de confusion et nous en déduisons le taux d'erreur.

```
#matrice de confusion et erreur en resubstitution
pred <- predict(arbre.full, newdata = donnees, type = "class")
mc <- table(donnees$coeur,pred) #matrice de confusion
print(mc)
err.resub <- 1.0 - (mc[1,1]+mc[2,2])/sum(mc)
print(err.resub)
```

Nous calculons les modalités prédites par le modèle pour chaque individu du fichier de données. Nous croisons ce vecteur (**pred**) avec le vecteur des classes observées (**donnees$coeur**). Nous obtenons ainsi :

```
> print(mc)
          pred
           absence presence
  absence      136       14
  presence      19      101
> err.resub <- 1.0 - (mc[1,1]+mc[2,2])/sum(mc)
> print(err.resub)
[1] 0.1222222
```

Le taux d'erreur en resubstitution est 12.22%. Il est censé indiquer la performance du modèle lorsqu'il sera déployé dans la population. On sait néanmoins qu'étant estimé sur les données d'apprentissage, il est biaisé, sous estimant le véritable taux d'erreur.

Validation croisée. Nous devons nous tourner vers les techniques de ré-échantillonnage lorsqu'il n'est pas possible de réserver une partie des données pour l'évaluation des modèles. La validation croisée est systématiquement proposée dans la très grande majorité des logiciels de Data Mining. Dans R, nous devons la programmer. Pour certains, ce préalable représente une barrière à l'entrée rédhibitoire. Pourtant, le code source associé est relativement simple[3].

Schématiquement, la validation consiste à subdiviser aléatoirement les données en K blocs. Nous réitérons le processus suivant, en faisant tourner les sous-échantillons : apprentissage du modèle sur les (K-1) blocs, évaluation du taux d'erreur en prédiction sur le K^{ème} bloc. Le taux d'erreur en validation croisée est la moyenne des taux d'erreurs ainsi collectés. C'est un estimateur de meilleure qualité que le taux d'erreur en resubstitution.

A partir de ce descriptif, nous retranscrivons les opérations dans R. Tout d'abord, nous allons **créer « aléatoirement » une colonne indiquant l'appartenance des individus aux blocs**.

```
#déterminer le numéro de bloc de chaque individu
n <- nrow(donnees) #nombre d'observations
K <- 10 # pour 10-validation croisée
taille <- n%/%K #déterminer la taille de chaque bloc
set.seed(5) #pour obtenir la même séquence tout le temps
alea <- runif(n) #générer une colonne de valeurs aléatoires
rang <- rank(alea) #associer à chaque individu un rang
bloc <- (rang-1)%/%taille + 1 # associer à chaque individu un numéro de bloc
bloc <- as.factor(bloc) #transformer en factor
print(summary(bloc)) #impression de contrôle
```

L'idée est de créer une colonne de valeurs aléatoires, de la transformer en rang associé à chaque individu. Nous en déduisons le numéro de bloc. La dernière instruction est destinée à vérifier que les effectifs sont identiques dans les blocs. Ce qui est le cas.

[3] Voir http://eric.univ-lyon2.fr/~ricco/cours/slides/resampling_evaluation.pdf pour l'explicitation de la technique.

```
> print(summary(bloc)) #impression de contrôle
 1  2  3  4  5  6  7  8  9 10
27 27 27 27 27 27 27 27 27 27
```

Nous pouvons maintenant **réitérer la séquence « apprentissage – test »**. Nous le réalisons à l'aide d'une boucle. A chaque modèle construit, nous évaluons le taux d'erreur sur le $k^{ème}$ bloc. Nous collectons les taux d'erreur dans le vecteur **all.err**.

```
#lancer la validation croisée
all.err <- numeric(0)
for (k in 1:K){
  #apprendre le modèle sur tous les individus sauf le bloc k
  arbre <- rpart(coeur ~., data = donnees[bloc!=k,], method = "class")
  #appliquer le modèle sur le bloc numéro k
  pred <- predict(arbre,newdata=donnees[bloc==k,], type = "class")
  #matrice de confusion
  mc <- table(donnees$coeur[bloc==k],pred)
  #taux d'erreur
  err <- 1.0 - (mc[1,1]+mc[2,2])/sum(mc)
  #conserver
  all.err <- rbind(all.err,err)
}

#vecteur des erreurs recueillies
print(all.err)
```

Nous obtenons la liste

```
> #vecteur des erreurs recueillies
> print(all.err)
          [,1]
err 0.2962963
err 0.2222222
err 0.1481481
err 0.1481481
err 0.1851852
err 0.1851852
err 0.2962963
err 0.3333333
err 0.1111111
err 0.1851852
```

Figure 1 - Détail des taux d'erreur dans les blocs

Puisque nous avons exactement le même nombre d'observations dans chaque bloc, nous pouvons calculer directement la moyenne non pondérée pour obtenir l'erreur en validation croisée.

```
#erreur en validation croisée
#on peut se contenter d'une moyenne non pondérée puisque les blocs sont
#de taille identique
err.cv <- mean(all.err)
print(err.cv)
```

Soit

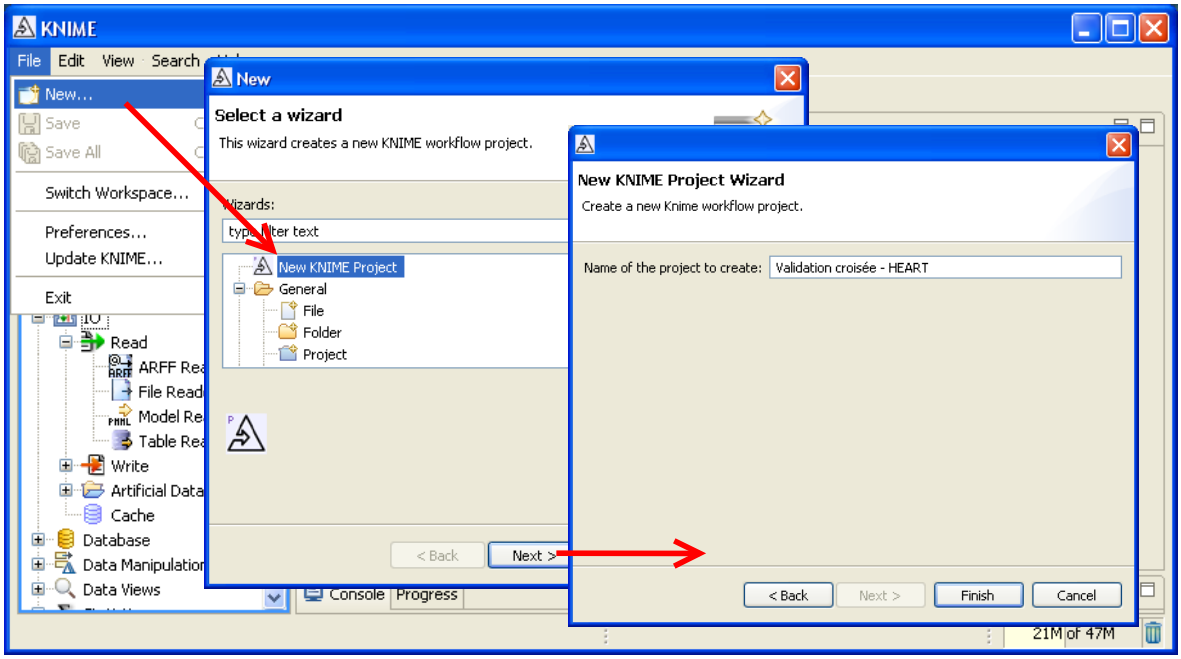
```
> print(err.cv)
[1] 0.2111111
```

Le taux d'erreur en validation croisée de la méthode rpart sur le fichier HEART est de 21.11%.

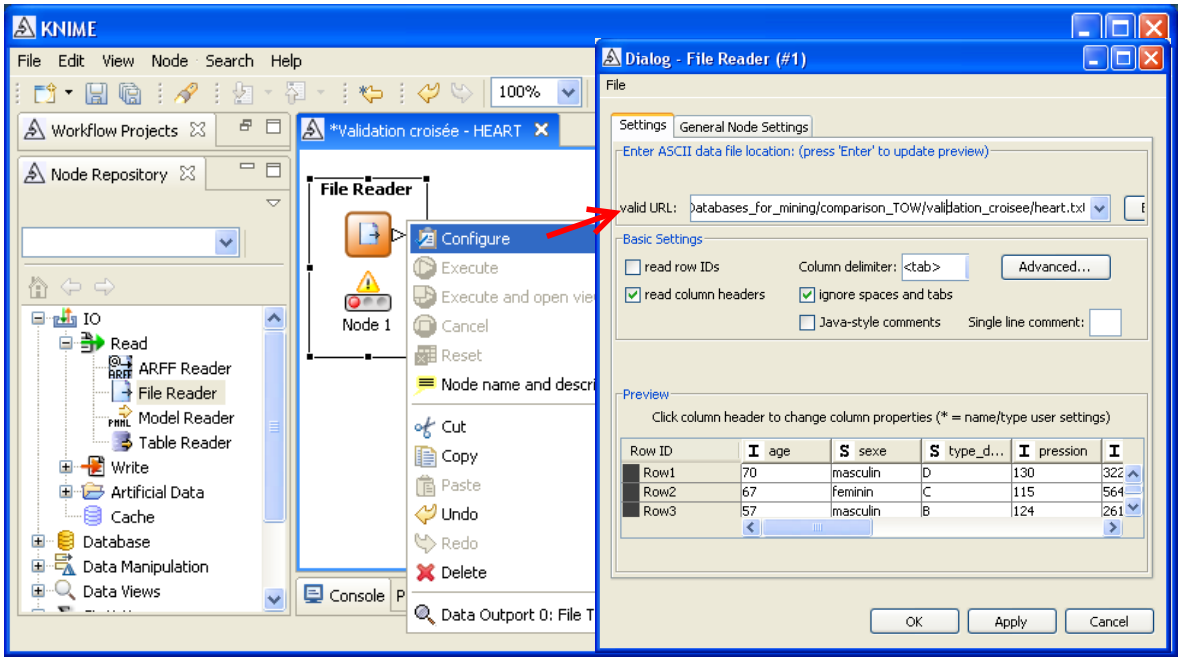
3 Validation croisée avec KNIME

KNIME est un logiciel que nous avons présenté à plusieurs reprises déjà (http://tutoriels-data-mining.blogspot.com/search?q=knime). La mise en place de la validation croisée est relativement complexe. Pour simplifier l'opération, Knime propose la notion de « méta-opérateur ». Elle correspond à une séquence prédéfinie de composants. Notre rôle consiste à la compléter avec la technique d'apprentissage supervisée que nous souhaitons évaluer.

Création d'un diagramme et importation des données. Pour créer un diagramme dans KNIME, nous actionnons le menu FILE / NEW. Dans le « wizard » qui apparaît, nous demandons un nouveau projet KNIME. Nous le nommons « Validation croisée - HEART ».

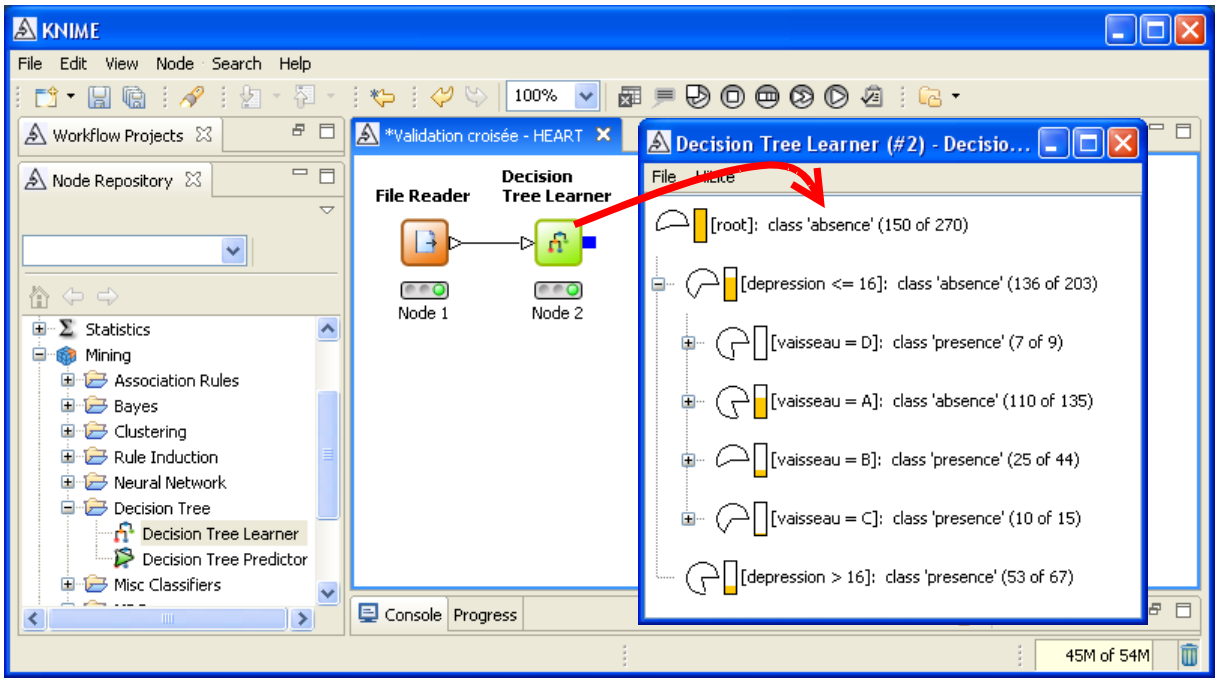


Un nouveau projet est créé. Nous importons le fichier HEART.TXT avec le composant FILE READER dans le diagramme. Avec le menu CONFIGURE, nous sélectionnons le fichier adéquat.



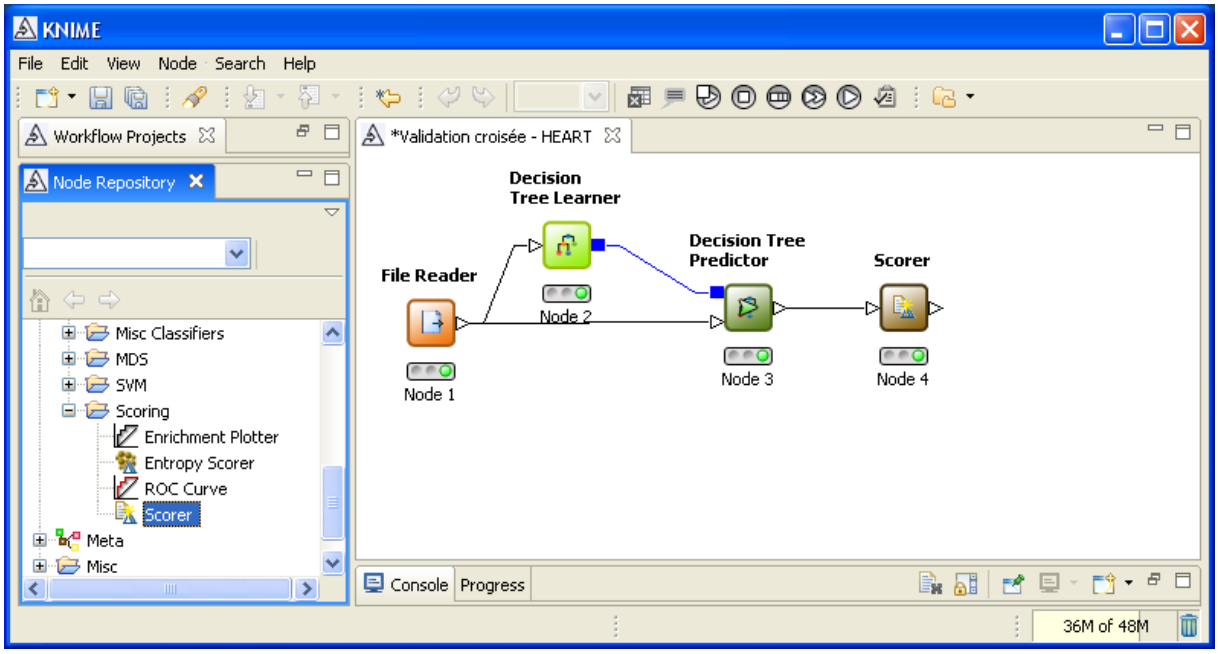
Une fraction des données apparaît dans le cadre de pré visualisation. Nous validons lorsque la configuration nous satisfait. Nous cliquons sur le menu contextuel EXECUTE pour finaliser le chargement.

Création de l'arbre sur la totalité des données. Pour créer l'arbre de décision à partir de la totalité des données, nous introduisons le composant DECISION TREE LEARNER. Nous lui connectons FILE READER et nous actionnons le menu contextuel EXECUTE AND OPEN VIEW.

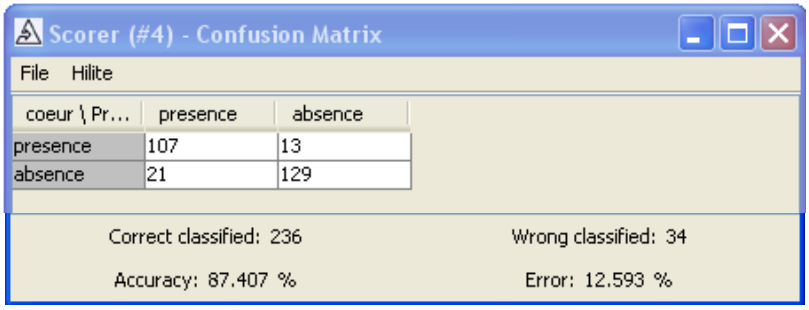


Par défaut, Knime utilise la dernière colonne comme variable à prédire. Cela convient dans notre situation. Dans le cas contraire, c.-à-d. la variable à prédire n'est pas située en dernière position, nous pouvons la définir dans la boîte de paramétrage (menu CONFIGURE).

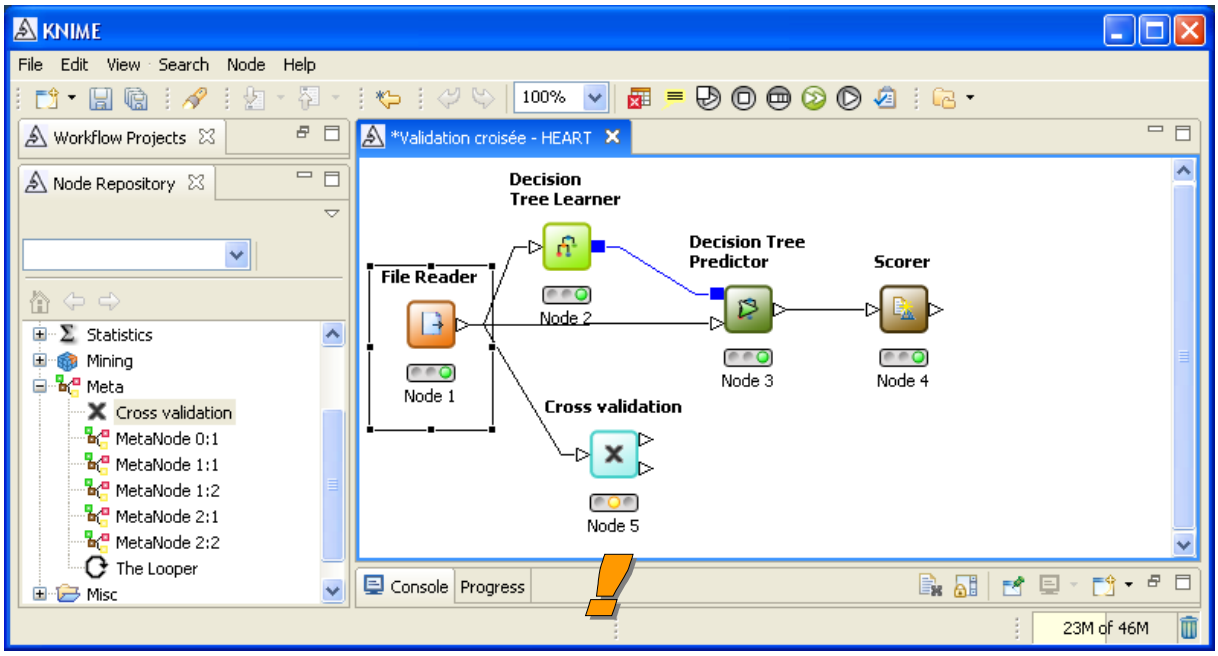
Evaluation en resubstitution. Nous appliquons le modèle sur les données d'apprentissage pour en évaluer les performances en resubstitution.



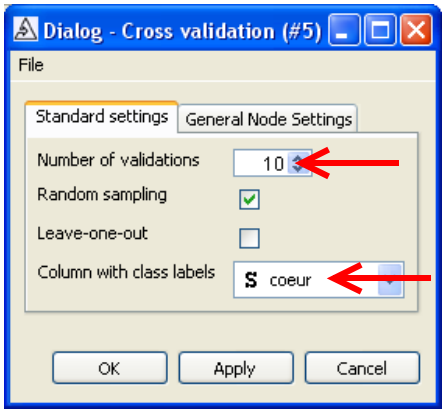
La démarche consiste à récupérer le modèle avec l'outil de prédiction DECISION TREE PREDICTOR. Nous lui connectons la source de données initiale. Une colonne « prédiction » est ajoutée à l'ensemble de données. Le composant SCORER sert à confronter les valeurs observées de CŒUR avec les valeurs prédites par l'arbre. En cliquant sur EXECUTE AND OPEN VIEW, nous obtenons la matrice de confusion, avec le taux d'erreur dans la partie basse de la fenêtre. Il est de 12.593%



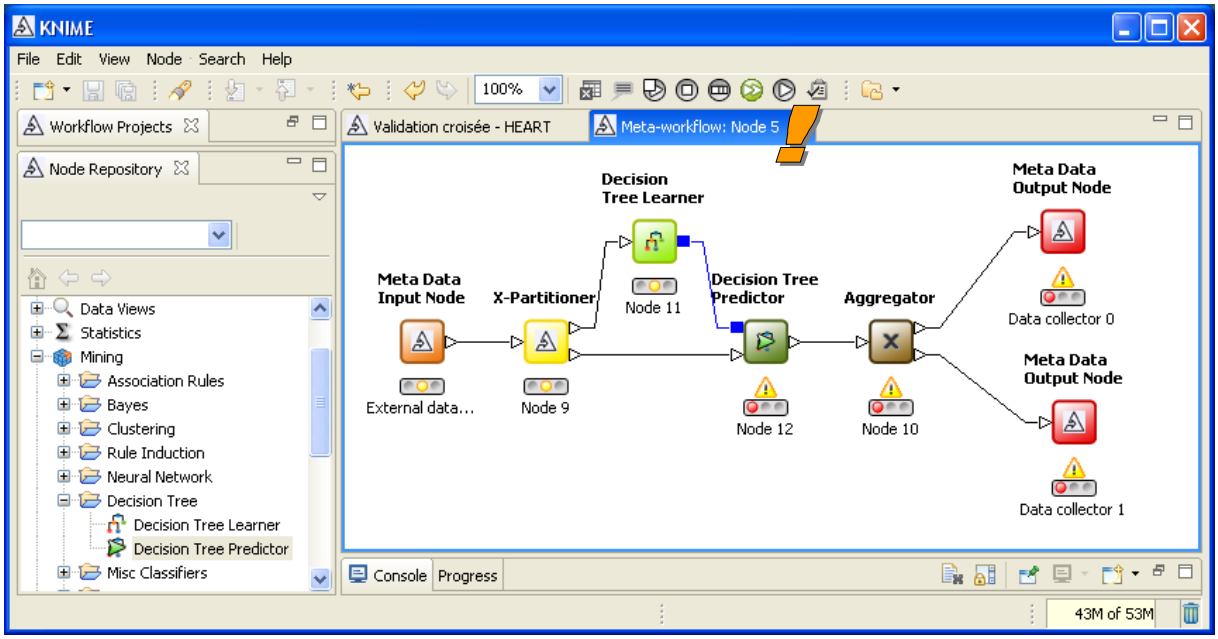
Erreur en validation croisée. Pour réaliser la validation croisée, Knime introduit la notion de « méta nœud ». Il s'agit de composants constitués d'une série d'opérateurs. Nous l'insérons dans le diagramme, nous lui relions le FILE READER.



Avec le menu CONFIGURE, nous spécifions : la variable à prédire (CŒUR) et le nombre de subdivision (10).

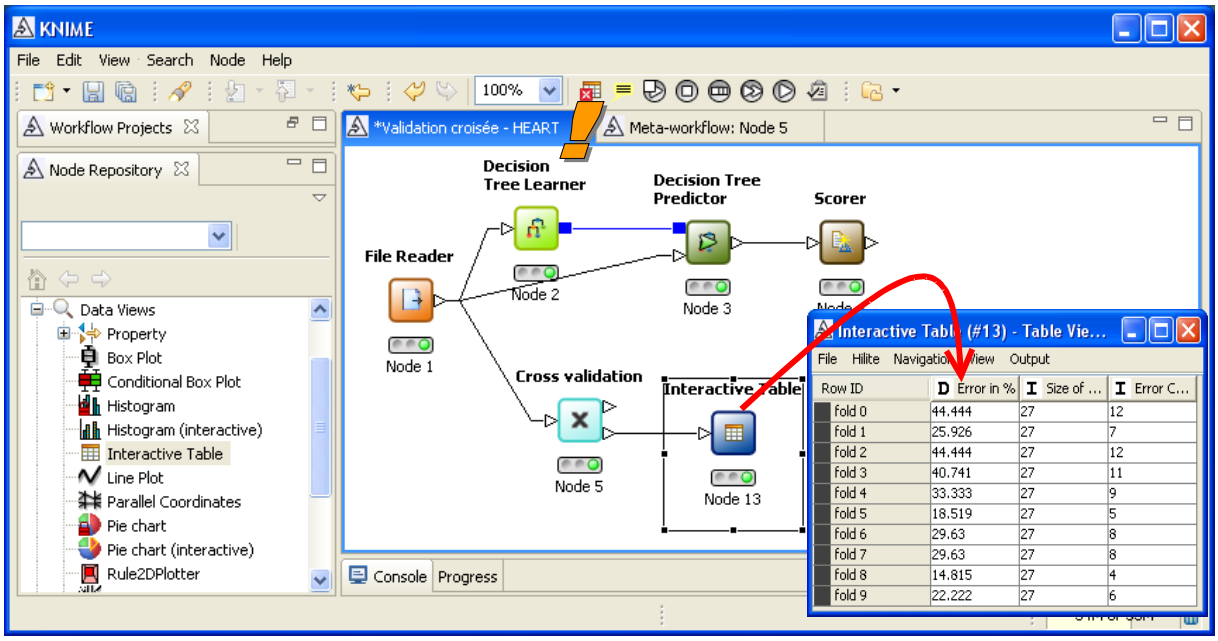


Pour compléter les composants internes du méta nœud, nous actionnons le menu OPEN META – WORKFLOW EDITOR. Un nouvel espace de travail apparaît. Par analogie avec la démarche d'évaluation du classifieur en resubstitution, nous complétons le méta nœud avec la méthode d'apprentissage (DECISION TREE LEARNER) et la prédiction associée (DECISION TREE PREDICTOR).



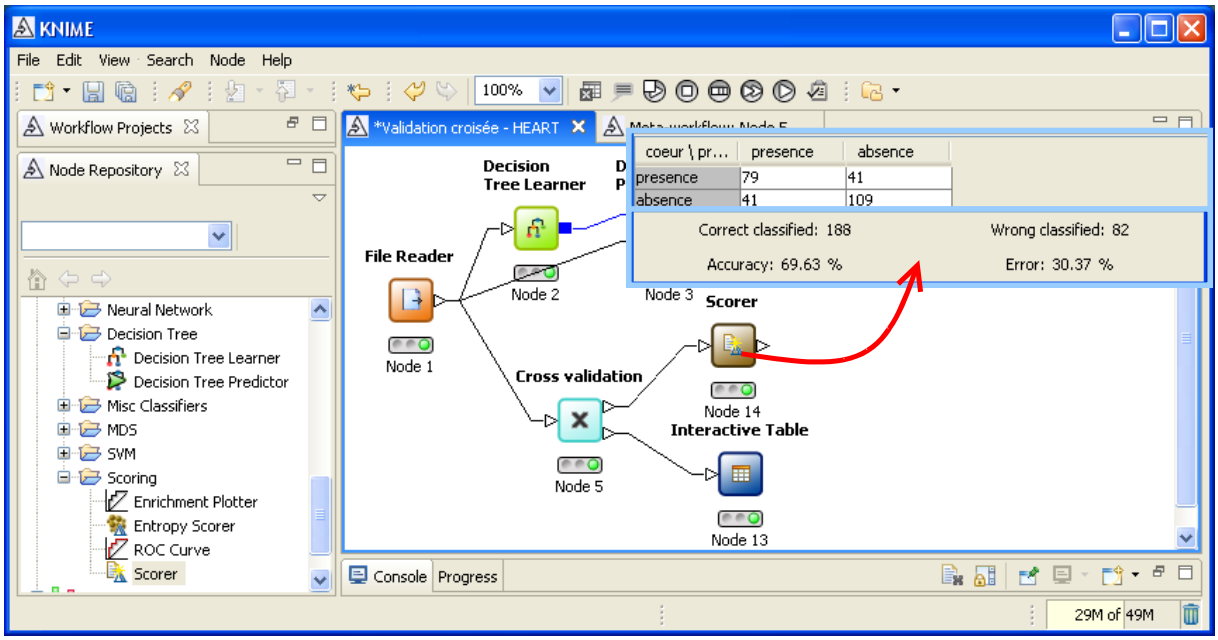
Nous revenons à l'espace de travail initial. Nous pouvons visualiser les résultats de 2 manières : une table contenant les taux d'erreurs pour chaque subdivision du fichier en validation croisée ; une colonne prédiction que nous pouvons croiser avec les valeurs de la variable à prédire.

Pour obtenir le détail des taux d'erreurs dans chaque bloc, nous utilisons le composant interactive table. Nous le branchons et nous actionnons le menu EXECUTE AND OPEN VIEW.



Ce tableau est à rapprocher avec celui que nous avions produits à l'aide de R (Figure 1).

Pour obtenir le taux d'erreur global en validation croisée, nous utilisons de nouveau l'outil SCORER. La matrice de confusion est également affichée.



Le taux d'erreur en validation croisée est 30.37%, fortement majoré par rapport au taux en resubstitution. Ce qui montre encore une fois, si besoin était, combien ce dernier est biaisé.

Par rapport au résultat de rpart de R (21.11%), le taux d'erreur en validation croisée semble un peu exagéré ici. Il ne faut pas trop s'affoler à ce sujet. On sait que les arbres sont très sensibles au paramétrage. Il faudrait se pencher sur les arcanes des implémentations dans chaque logiciel pour situer précisément les différences.

4 Validation croisée avec RAPIDMINER

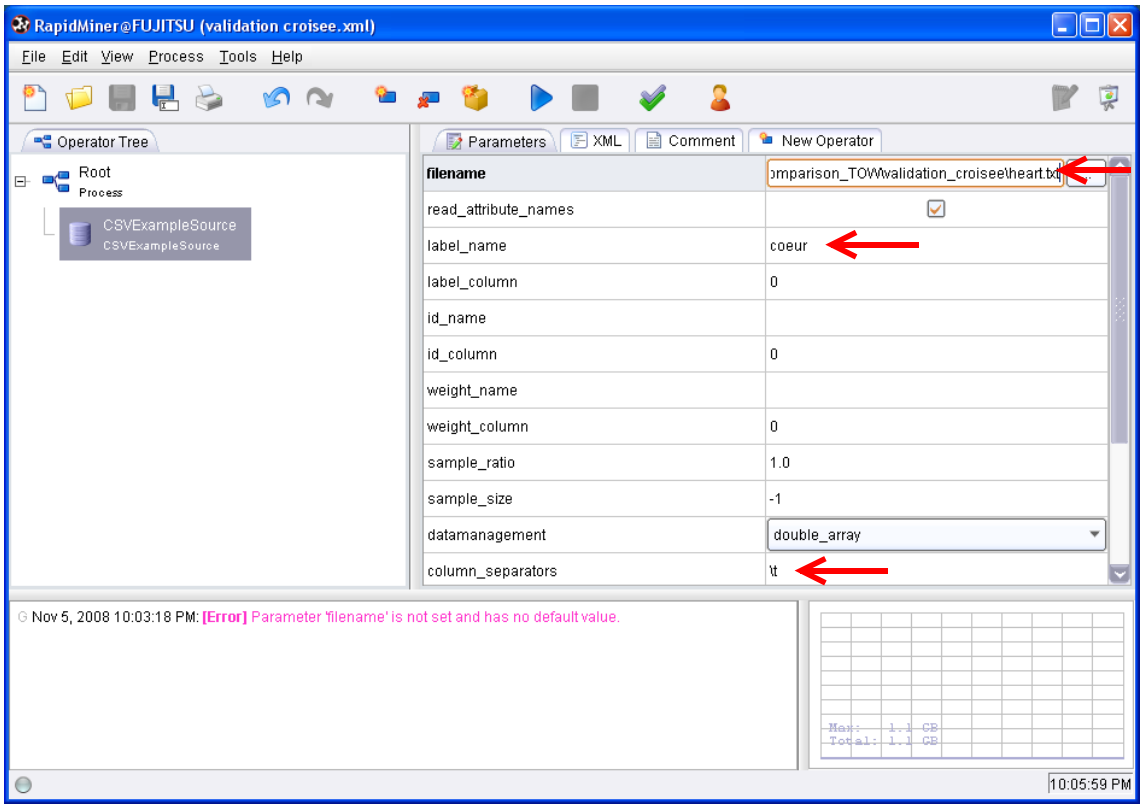
RAPIDMINER semble très connu dans le monde anglo-saxon[4], c'est moins le cas dans la communauté francophone. Pourtant, il présente d'excellentes qualités. Il propose entres autres un nombre impressionnant de méthodes (on s'y perd un peu d'ailleurs). Ici aussi, nous pouvons mettre en œuvre relativement facilement la validation croisée... pour peu que l'on se plie à la philosophie de l'outil.

A la différence des autres logiciels, nous avons tout intérêt à définir toute la filière d'une traite avant de lancer les calculs dans RapidMiner. En effet, à chaque exécution, il relance les calculs sur l'ensemble des composants. Il n'est possible de demander une exécution sélective des branches du diagramme.

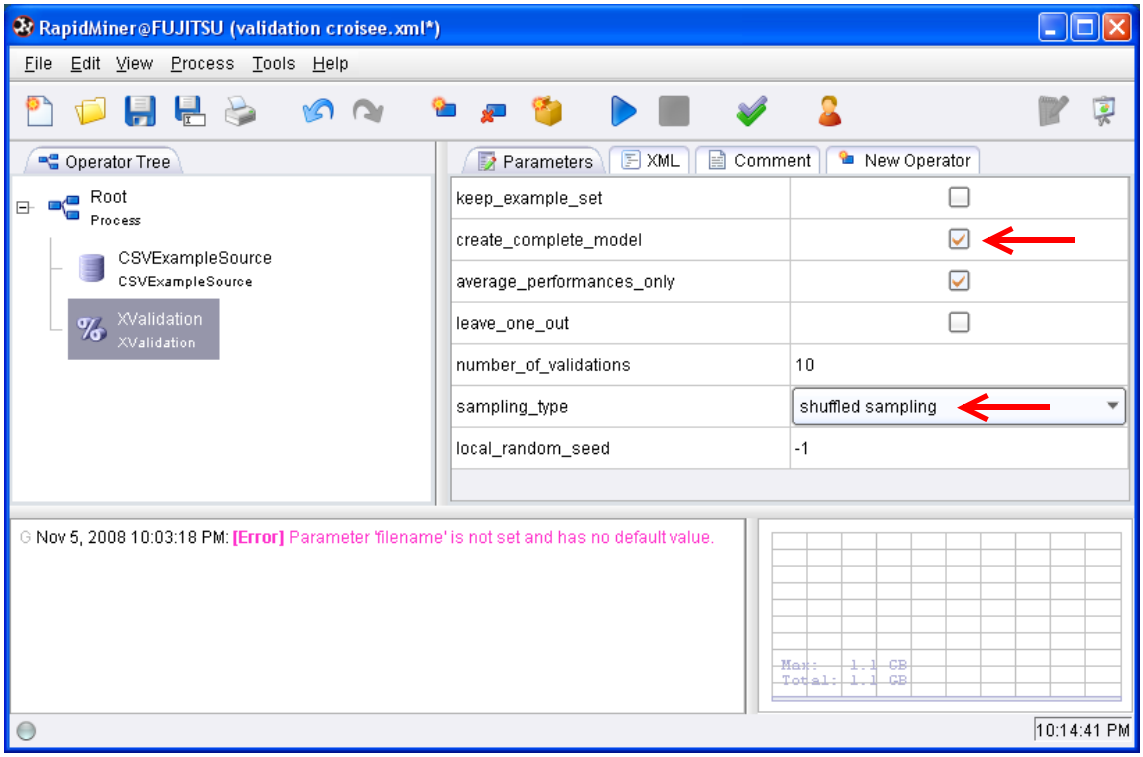
Autre différence notable, la validation croisée dans RapidMiner produit directement l'arbre construit sur la totalité des données. Il n'est donc pas nécessaire de réaliser ce calcul à part.

Création d'un diagramme et importation des données. Au lancement de RapidMiner, on demande la création d'un nouveau diagramme (arbre des opérations) via le menu FILE / NEW. Nous insérons le composant CSV EXAMPLE SOURCE à la racine de l'arbre. Nous lui indiquons le fichier (FILENAME), la variable à prédire (LABEL NAME), le séparateur de colonne (tabulation, COLUMN SEPARATORS).

[4] http://www.kdnuggets.com/polls/2008/data-mining-software-tools-used.htm

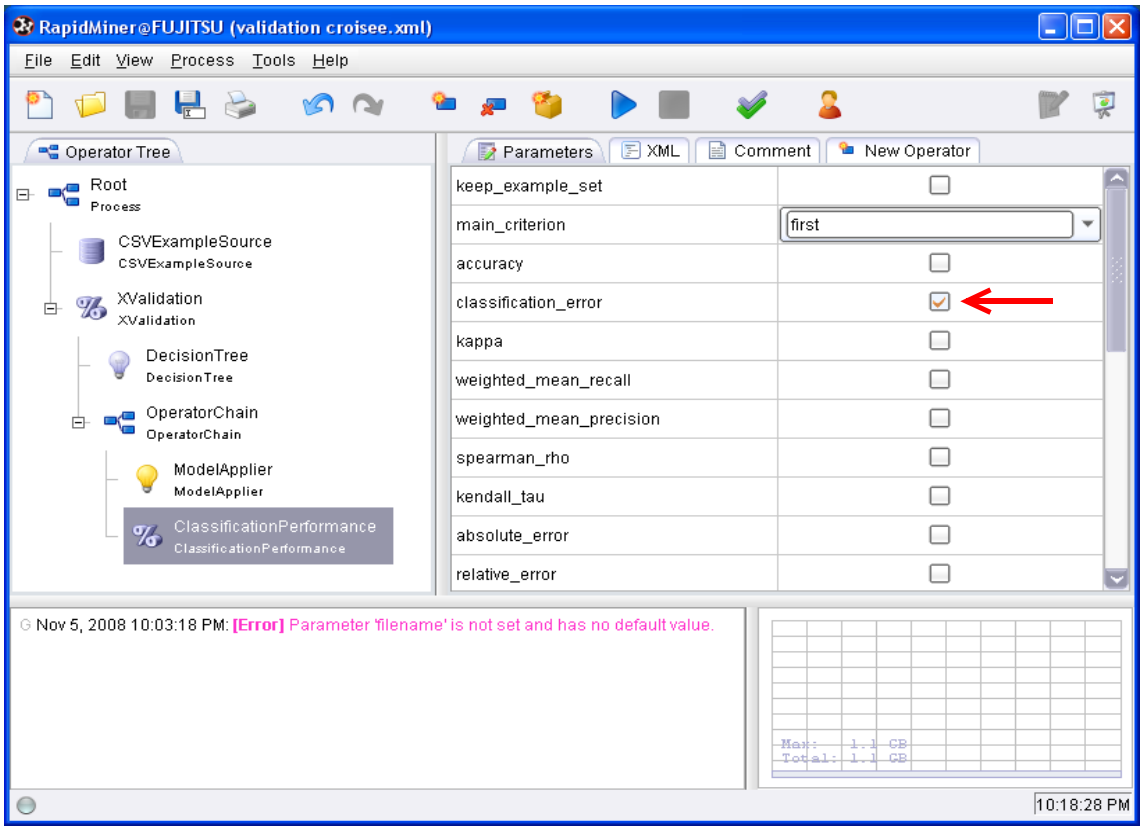


Validation croisée. D'emblée nous devons introduire la validation croisée avec le composant XVALIDATION. Le paramétrage est important : nous demandons l'affichage du modèle élaboré sur la totalité des données (CREATE COMPLETE MODEL), l'échantillonnage doit être simple (SAMPLING TYPE = SHUFFLED SAMPLING).

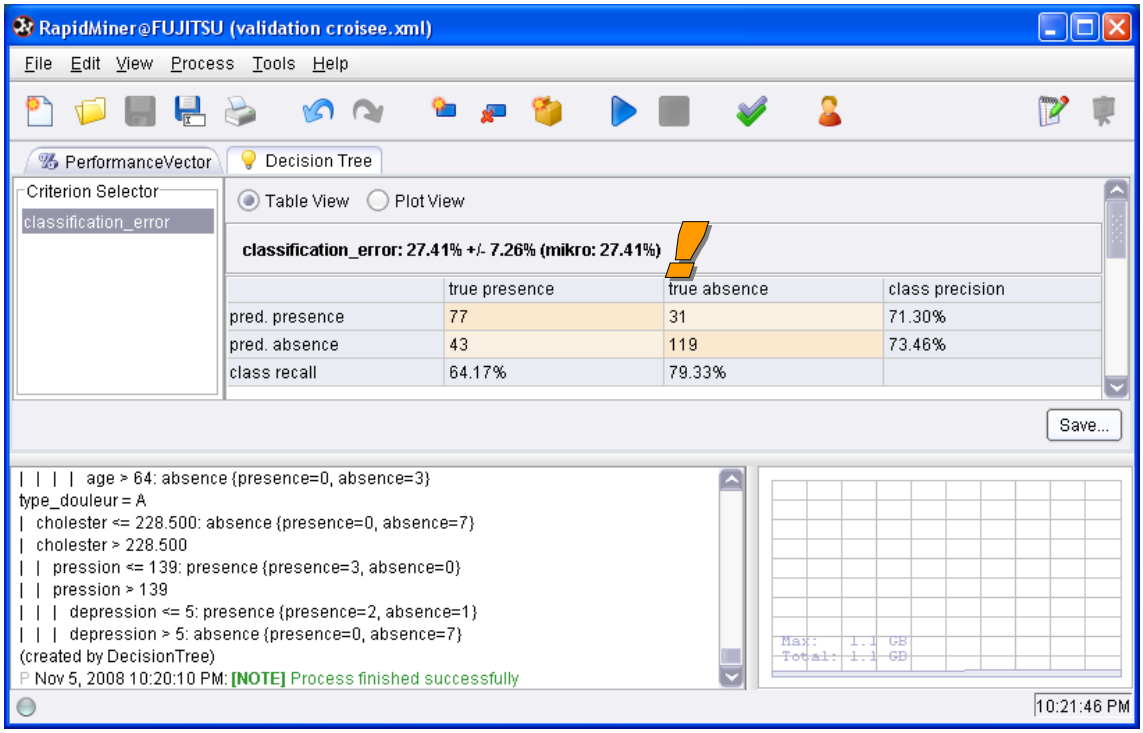


Dans cette sous branche « validation croisée », nous devons introduire : l'outil d'apprentissage (DECISION TREE) et la séquence « application du classifieur sur le bloc de données et calcul du taux

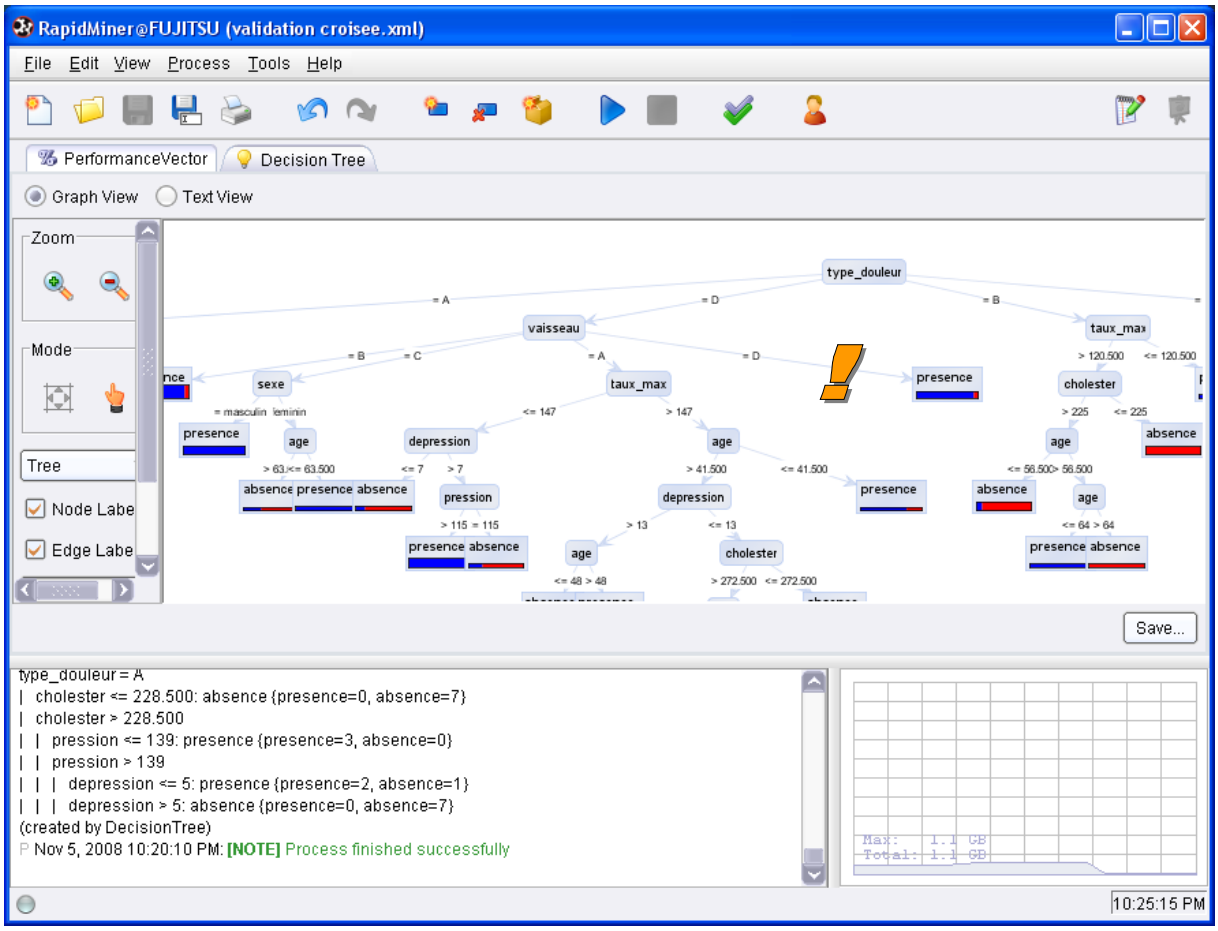
d'erreur ». Le diagramme se présente comme suit. Nous demandons le taux d'erreur pour le composant CLASSIFICATION PERFORMANCE.



Exécution du diagramme. Nous pouvons à ce stade exécuter le diagramme en cliquant sur le gros bouton PLAY dans la barre d'outils. RapidMiner réunit les principaux résultats dans une fenêtre à onglets. Le premier résultat disponible est la matrice de confusion et le taux d'erreur en validation croisée. Il est de 27.41%.



Dans l'onglet DECISION TREE, RapidMiner affiche l'arbre de décision calculé sur la totalité des données.



Deux modes de visualisation sont disponibles, nous montrons ici l'affichage graphique.

5 Conclusion

Dans ce didacticiel, nous avons montré comment mettre en œuvre la validation croisée dans trois logiciels : R, KNIME et RAPIDMINER. Ce document complète une précédente étude où nous mettions en place les mêmes schémas de traitements dans trois autres logiciels : TANAGRA, ORANGE et WEKA.

Au final, nous retombons sur une conclusion qui constitue un de nos leitmotiv : lorsque l'on sait exprimer clairement ce que l'on souhaite obtenir, on trouve assez facilement les bons outils dans les logiciels. La plupart proposent des fonctionnalités et des modalités opératoires assez proches, surtout lorsqu'il s'agit de procédures très courantes telles que la validation croisée.