1 Objectif

Choix de représentation et préférences d'apprentissage. Comparaison de classifieurs linéaires sur des données artificielles.

L'apprentissage supervisé a pour objectif de mettre au jour une liaison fonctionnelle *f(.)* entre une variable cible Y (variable à prédire, variable expliquée, etc.) que l'on cherche à prédire et une ou plusieurs variables prédictives (les descripteurs, les variables explicatives, etc.) (X_1, X_2, …, X_p). La fonction est paramétrée c.-à-d. $Y = f(X_1, X_2, \ldots, X_p ; \alpha)$, où α est un vecteur de paramètres associée à *f(.)*. Nous sommes confrontés à deux écueils dans ce processus.

Le premier problème est le choix de la fonction *f(.)*. Il existe une multitude de formes de liaisons possibles. On distingue généralement les modèles linéaires des modèles non-linéaires. A priori, nous avons toujours intérêt à choisir la formulation la plus complexe, c.-à-d. un modèle non-linéaire : "qui peut le plus, peut le moins". En réalité, la situation est un peu plus subtile. La relation entre les descripteurs et la variable à prédire est une vue de l'esprit. Essayer de retraduire une hypothétique causalité avec une fonction mathématique est toujours hasardeux. Ainsi, certains auteurs préconisent, ne serait-ce que dans un premier temps lorsque nous n'avons aucune idée sur la nature de la relation entre la cible et les descripteurs, de vérifier le comportement des modèles linéaires sur les données que nous traitons (Duda et al., 2001 ; page 215).

Le second problème réside dans le calcul des paramètres de la fonction de prédiction. Nous souhaitons construire la fonction la plus efficace possible sur la population. Mais nous ne disposons que d'un échantillon, dit d'apprentissage, pour les calculs. La préférence d'apprentissage décrit le mode d'exploration des solutions. Elle permet de choisir entre deux solutions concurrentes de la même représentation. Elle permet aussi de restreindre la recherche. Bien souvent, mais ce n'est pas toujours le cas, les caractéristiques du mode d'exploration est retraduite par le critère à optimiser durant l'apprentissage (maximum de vraisemblance, moindres carrés, maximisation de la marge, etc.). A priori, nous avons tout intérêt à choisir une méthode qui explore toutes les hypothèses possibles de manière à choisir la meilleure. Mais là non plus, ce n'est pas aussi simple. Nous courrons le risque d'ingérer les particularités propres au fichier d'apprentissage au détriment de la « vraie » relation que nous voulons mettre en évidence. On parle de « surapprentissage » c.-à-d. l'algorithme intègre dans le modèle prédictif des informations spécifiques à l'échantillon d'apprentissage qui ne sont pas transposables dans la population. La situation est d'autant plus difficile qu'il est vraisemblable que certains descripteurs ne soient pas pertinents dans la prédiction. Ils peuvent perturber les calculs. S'en suit donc une caractéristique très importante de l'apprentissage supervisé : en admettant que la relation existe, et que nous avons adopté la représentation *f(.)* correcte, nous n'avons pas pour autant la garantie que l'algorithme d'apprentissage trouve les valeurs adéquates des paramètres. Certains ont besoin de plus d'observations que d'autres pour converger vers la bonne solution ; d'autres, de par leurs hypothèses intrinsèques, peuvent s'avérer incapables de la calculer correctement sur certains jeux de données.

Dans ce tutoriel, nous étudions le comportement de 5 classifieurs linéaires sur des données artificielles. Les modèles linéaires sont des outils privilégiés de l'apprentissage supervisé. En effet,

s'appuyant sur une simple combinaison linéaire des variables prédictives, ils présentent l'avantage de la simplicité : la lecture de l'influence de chaque descripteur est relativement facile (signes et valeurs des coefficients) ; les techniques d'apprentissage sont souvent rapides, même sur de très grandes bases de données. Nous nous intéresserons plus particulièrement à[1] : (1) le modèle bayésien naïf (modèle d'indépendance conditionnelle), (2) l'analyse discriminante, (3) la régression logistique, (4) le perceptron simple, et (5) les machines à vaste marge (SVM, support vector machine)

Nous nous plaçons dans un cadre particulier concernant les données. Nous les avons générées artificiellement pour un problème à 2 classes Y $\in$ {positif, négatif}. Le nombre de variables prédictives p peut être quelconque (p $\geq$ 2), mais seules les deux premières (X1 et X2) sont pertinentes. De fait, la frontière permettant de distinguer les positifs des négatifs est représentée par une droite dans le plan (X1, X2). Pour corser l'affaire, nous pouvons bruiter aléatoirement les étiquettes. Nous connaissons ainsi d'emblée les performances optimales que l'on est en mesure d'obtenir. Nous verrons alors quelles seront les méthodes qui s'en approchent le mieux, en fonction de la taille de l'échantillon d'apprentissage et du nombre de descripteurs.

L'expérimentation a été menée entièrement sous R. Le code source accompagne ce document. Mon idée, outre le thème des classifieurs linéaires qui nous préoccupe, est aussi de décrire les différentes étapes de l'élaboration d'une expérimentation pour la comparaison de techniques d'apprentissage. Bien souvent, on nous demande de produire des grands tableaux d'indicateurs de performances sur des bases UCI[2] dans les revues ou conférences pour montrer « statistiquement » la prétendue supériorité d'une méthode sur les autres. On se rend compte dans ce tutoriel que quelques lignes de codes sous R suffisent pour les construire. Mais les tableaux de chiffres sont souvent parcellaires. Ils masquent parfois des choix implicites ou explicites. La seule manière d'accéder à une vision exhaustive de l'expérimentation est de disposer du code source et des données réellement utilisées par le chercheur (ex. comment ont été traitées les données manquantes sur les bases UCI qui en comportent ? quand la technique d'apprentissage a échoué, comment a été comptabilisé son erreur sur l'échantillon test ?).

2 Données

Nous utilisons le code R suivant pour générer un ensemble de données de « n » observations, « p » descripteurs (+ la variable cible) et un niveau de bruit « noise ».

```
generate.data <- function(n=100, p=2, noise=0.05){
  #génération des descripteurs
  X <- data.frame(lapply(1:p,function(x){runif(n)}))
  colnames(X) <- paste("x",1:p,sep="")
  #étiquetage des observations et bruit sur les étiquettes
  y.clean <- ifelse(X$x2 > 4*X$x1,1,2)
  y <- factor(ifelse(runif(n)>(1.0-noise),3-y.clean,y.clean))
  levels(y) <- c("neg","pos")
  all.data <- cbind(y,X)
```

[1] http://en.wikipedia.org/wiki/Linear_classifier

[2] UCI Machine Learning Repository, http://archive.ics.uci.edu/ml/

```
  return(all.data)
}
```

Les descripteurs sont générés selon une loi uniforme (0, 1). L'étiquette est attribuée selon la règle :

Si (X2 > 4 * X1) **Alors** Y = Négatif **Sinon** Y = Positif

Les classes sont relativement déséquilibrées avec (à peu près) 12.5% de négatifs et 87. 5% de positifs.

Le bruitage est réalisée sur la classe en retournant l'étiquette dans (100*noise)% des cas c.-à-d. chaque individu, quelle que soit sa classe, a (100*noise)% de chances de voir son étiquette retournée[3]. Par construction, il est impossible avec un classifieur linéaire d'obtenir un taux d'erreur inférieur à « noise » (ponctuellement sur certains échantillons tests, mais pas en espérance).

Sur un échantillon de 20.000 observations bruité à 5%, généré à l'aide de la commande suivante…

```
#number of descriptors
p <- 2
#noise on the class attribute - theoretical error rate
noise <- 0.05
#generating test set
n.test <- 20000
test.data <- generate.data(n.test,p,noise)
#plotting test set
plot(test.data$x1,test.data$x2,pch=21,bg=c("red","blue")[unclass(test.data$y)])
```

… nous distinguons nettement la frontière linéaire séparant les classes. Nous observons également les quelques points (5% des observations puisque **noise = 0.05**) disposés du mauvais côté (Figure 1).

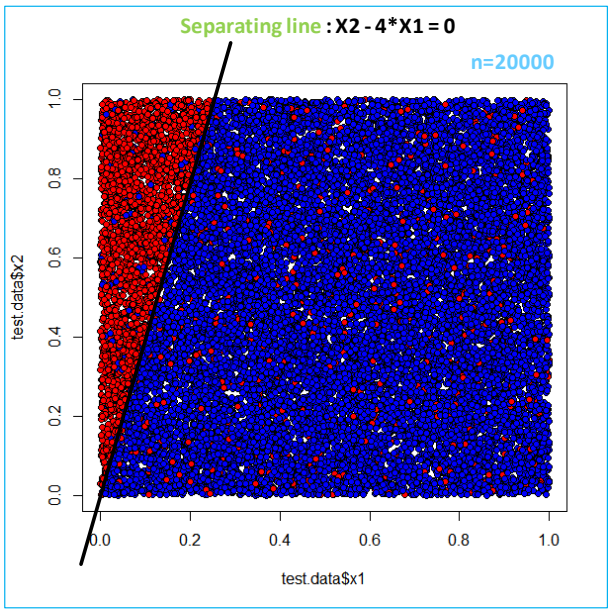


Figure 1 - "Vraie" frontière visible sur un échantillon de 20.000 observations

[3] Attention, **ce mode de bruitage modifie les proportions des positifs et négatifs dans l'échantillon**. Les positifs étant plus nombreux, ils seront plus souvent retournés. Ainsi, le pourcentage apparent des négatifs dans l'échantillon d'apprentissage est plus élevé après bruitage. Une autre piste de réflexion aurait été d'utiliser une stratégie de perturbation préservant les proportions initiales.

Malheureusement, et c'est tout le problème de l'apprentissage supervisé, les données étiquetées sont rares, difficiles à obtenir dans certains contextes.

```
#training set size
n.train <- 300
#training set
train.data <- generate.data(n.train,p,noise)
#plotting training set
plot(train.data$x1,train.data$x2,pch=21,bg=c("red","blue")[unclass(train.data$y)])
```

La frontière est moins marquée sur un échantillon de 300 observations (Figure 2). Si nous essayons quand même de la tracer à main levée, nous ne coïncidons pas vraiment avec la bonne solution. Et surtout, plus ennuyeux encore serais-je tenté de dire, si nous disposons d'un autre échantillon de même taille, la droite séparatrice induite sera (un peu) différente. Et pourtant, les algorithmes d'apprentissage dispose uniquement de ces informations (n.train = 300) pour essayer de reproduire la « vraie » droite de séparation.

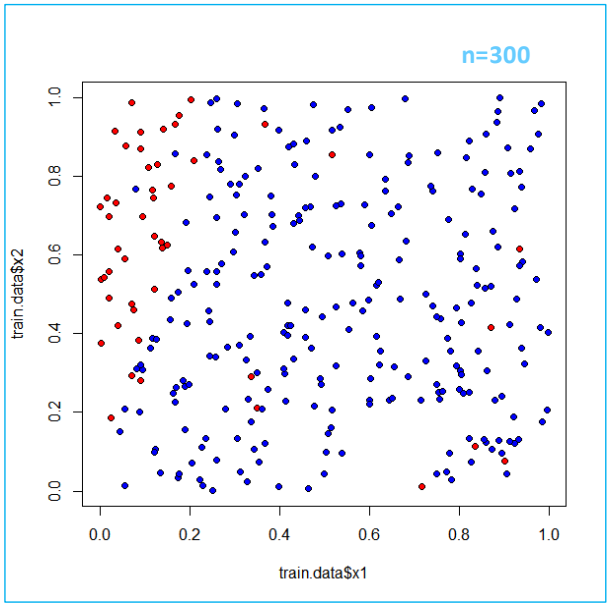


Figure 2 - Echantillon de 300 observations soumis aux algorithmes d'apprentissage

Remarque : L'intérêt d'utiliser des données artificielles est que nous contrôlons parfaitement les conditions de l'expérimentation. Ici, nous savons très bien que nos données vont à l'encontre des hypothèses sous-jacentes à l'analyse discriminante ou au bayésien naïf (normalité conditionnelle). La nature des résultats ne devrait pas nous surprendre, ces méthodes seront pénalisées. En revanche, l'ampleur des écarts sera intéressante à étudier, surtout en fonction des effectifs et du nombre de descripteurs. De plus, nous maîtrisons complètement le niveau de bruit, que nous pouvons faire évoluer à notre guise. Nous connaissons d'avance le taux d'erreur optimal que nous sommes en mesure obtenir. J'insiste sur ces points car on voit souvent des expérimentations à grande échelle sur des bases UCI où on ne connaît ni la nature du concept associant la cible aux variables prédictives, ni le niveau de bruit initial sur les classes. Si ça se trouve, en retournant les étiquettes, nous sommes en train d'attribuer les bonnes valeurs à la variable à prédire. Analyser les résultats dans ces conditions tient plus de la plaisanterie que de la démarche scientifique.

3 Comparaison des classifieurs linéaires

Nous évaluons plusieurs techniques d'apprentissage dans cette section. Le schéma est toujours le même : nous construisons le modèle sur l'échantillon de « n.train = 300 » observations (Figure 2) ; nous évaluons les performances sur le second échantillon de « n.test = 20000 » instances (Figure 1) ; nous comparons la frontière induite avec la droite de séparation théorique.

Pour mesurer le taux d'erreur et visualiser la frontière induite, nous utilisons le code suivant sous R :

```
#fonction pour calculer le taux d'erreur
#et tracer la séparation induite
#data.test représente l'échantillon test, data.test$y la variable cible
#pred est la prédiction du modèle à évaluer
#la fonction affiche la matrice de confusion et renvoie le taux d'erreur
error.rate.plot <- function(data.test,pred){
  #affichage des points selon la classe prédite
  plot(data.test$x1,data.test$x2,pch=21,bg=c("red","blue")[unclass(pred)])
  #la frontière forme la droite de séparation induite par la méthode
  abline(0,4,col="green",lwd=5)
  #matrice de confusion et taux d'erreur
  mc <- table(data.test$y,pred)
  print(mc)
  err.rate <- 1-sum(diag(mc))/sum(mc)
  return(err.rate)
}
```

3.1 Modèle théorique

Dans un premier temps, afin de calibrer notre échantillon test, nous avons calculé le taux d'erreur du modèle théorique à l'aide des instructions suivantes :

```
#prédiction théorique
pred.thq <- factor(ifelse(test.data$x2-4*test.data$x1>0,1,2))
print(error.rate.plot(test.data,pred.thq))
```

Nous obtenons un taux d'erreur de 5.19%, proche de l'erreur théorique (5%). Elle sera d'autant plus précise que nous augmentons la taille de l'échantillon test.

```
> print(error.rate.plot(test.data,pred.thq))
     pred
        1     2
  neg  2320   907
  pos   131 16642
[1] 0.0519
```

La frontière théorique (en vert) est exactement reproduite avec : en rouge les prédictions « négatives », en bleu les « positives ». Bien évidemment les prédictions sont totalement homogènes de part et d'autre de la droite de séparation. Les points précédemment en bleu dans la partie rouge (et inversement en rouge dans la partie bleu) dans les données initiales (Figure 1) constituent les observations mal classées par le modèle.

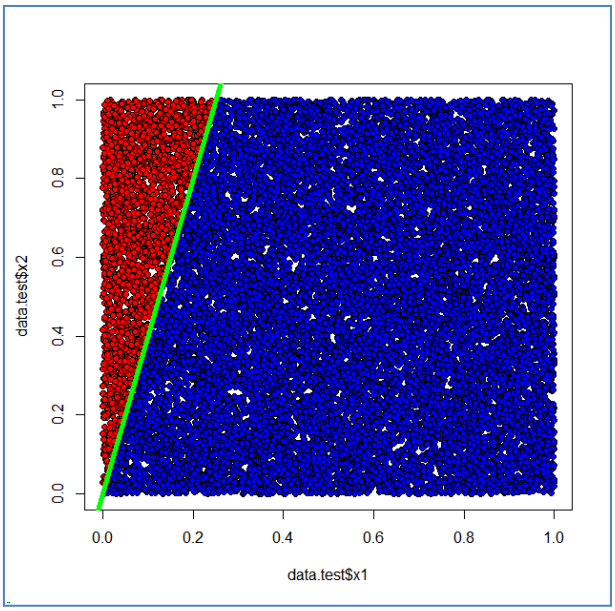


Figure 3 - Prédiction du modèle théorique (avec la « varie » frontière) sur l'échantillon test

3.2 Bayésien naïf (modèle d'indépendance conditionnelle)

Le bayésien naïf (traduction de « Naive Bayes ») repose sur deux hypothèses relatives aux distributions des variables prédictives conditionnellement à Y : elles suivent une loi normale, elles sont deux à deux indépendantes. Ces deux postulats sont mis à mal sur nos données. Les densités conditionnelles notamment montrent que le caractère gaussien des distributions n'est absolument pas respecté (Figure 4)[4].

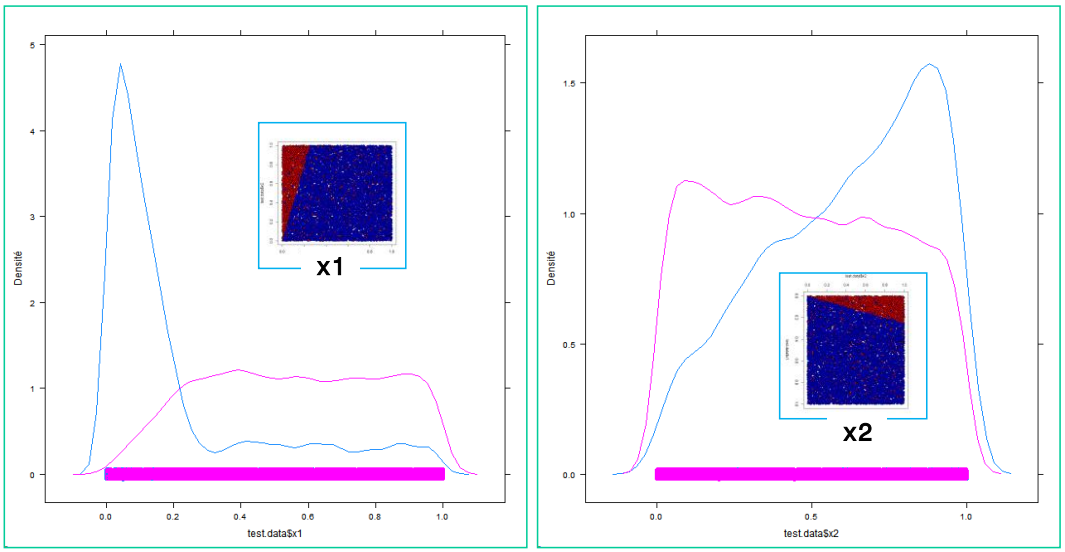


Figure 4 - Distributions de X1 et X2 conditionnellement aux valeurs de Y

Voyons ce qu'il en est lorsque nous construisons le modèle sur l'échantillon « train » et que nous procédons à l'évaluation sur « test ».

[4] Nous avons utilisé le code suivant sous R :

```
library(lattice)
densityplot(test.data$x1,groups=test.data$y)
densityplot(test.data$x2,groups=test.data$y)
```

```
#chargement du package e1071 qu'il faut installer au préalable
library(e1071)
model.nb <- naiveBayes(y ~ ., data = train.data)
print(model.nb)
#fonction de prédiction pour le classifieur bayésien naïf
prediction.nb <- function(model,test.data){
  return(predict(model,newdata=test.data))
}
#projection de la frontière dans le plan, affichage du taux d'erreur
print(error.rate.plot(test.data,prediction.nb(model.nb,test.data)))
```

R affiche les moyennes et écarts-type conditionnels pour les variables X1 et X2. Nous obtenons un taux d'erreur en test de **10.23%**. Nous sommes loin du taux théorique (5%).

```
Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
      neg       pos
0.1633333 0.8366667

Conditional probabilities:
     x1
Y           [,1]      [,2]
  neg 0.1884570 0.2487047
  pos 0.5369284 0.2657678

     x2
Y           [,1]      [,2]
  neg 0.6188492 0.2586698
  pos 0.4702892 0.2810891

>
> #function for naive bayes prediction
> prediction.nb <- function(model,test.data){
+   return(predict(model,newdata=test.data))
+ }
>
> print(error.rate.plot(test.data,prediction.nb(model.nb,test.data)))
     pred
        neg   pos
  neg  1246  1981
  pos    65 16708
[1] 0.1023
```

La frontière est décalée par rapport à la frontière optimale (Figure 5).

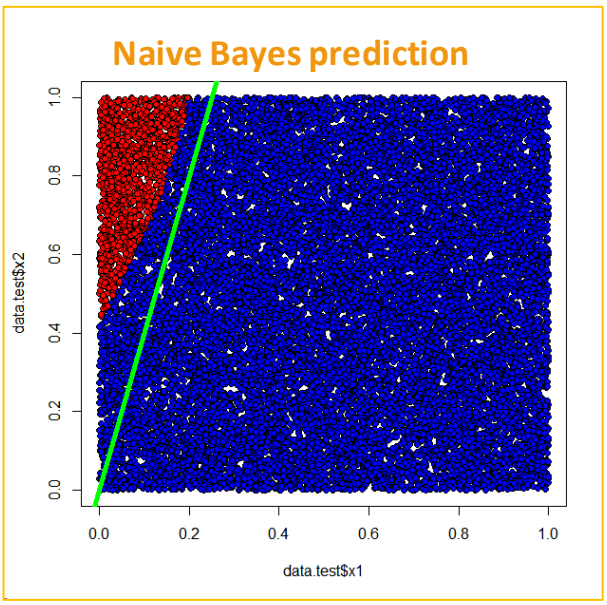


Figure 5 - Frontière induite par le classifieur bayésien naïf (en vert la frontière optimale)

Nous pouvons déduire des informations fournies par R les coefficients de l'équation définissant la droite de séparation. Nous le verrons plus loin (section 4), Tanagra peut les fournir directement (il en sera de même pour l'analyse discriminante et les SVM).

3.3 Analyse discriminante linéaire

L'analyse discriminante linéaire est également paramétrique. Elle repose sur deux postulats : les distributions de X (prises conjointement) conditionnellement à Y suivent une loi normale multidimensionnelle ; les matrices de variance covariance sont identiques c.-à-d. les nuages de points ont la même forme. Un coup d'œil sur nos différents graphiques suffit pour constater qu'à l'évidence, ces hypothèses ne sont pas respectées. Nous savons néanmoins que la méthode est robuste. Nous vérifions jusqu'à quel point sur nos données.

```
#charger le package MASS au préalable
library(MASS)
model.lda <- lda(y ~ ., data = train.data)
print(model.lda)
#fonction de prédiction
prediction.lda <- function(model,test.data){
  return(predict(model,newdata=test.data)$class)
}
#graphique et taux d'erreur
print(error.rate.plot(test.data,prediction.lda(model.lda,test.data)))
```

Ce n'est guère mieux par rapport au bayésien naïf, le taux d'erreur est de **10.35%**,

```
Call:
lda(y ~ ., data = train.data)

Prior probabilities of groups:
      neg       pos
0.1633333 0.8366667

Group means:
           x1        x2
neg 0.1884570 0.6188492
pos 0.5369284 0.4702892

Coefficients of linear discriminants:
          LD1
x1  3.535681
x2 -1.468128
>
> #function for linear discriminant analysis prediction
> prediction.lda <- function(model,test.data){
+   return(predict(model,newdata=test.data)$class)
+ }
>
> print(error.rate.plot(test.data,prediction.lda(model.lda,test.data)))
     pred
      neg   pos
  neg  1221  2006
  pos    65 16708
[1] 0.10355
```

La frontière induite est tout aussi à l'ouest (Figure 6). Cela tient en grande partie au fait que les nuages de points associés aux classes sont de formes différentes comme en atteste les matrices de variance covariance conditionnelles. Les valeurs hors diagonale principale sont de signes opposés c.-à-d. les nuages ne sont pas orientés de manière identique.

```
> cov(train.data[unclass(train.data$y)==1,2:3])
            x1          x2
x1  0.06185403 -0.02401864
x2 -0.02401864  0.06691007
> cov(train.data[unclass(train.data$y)==2,2:3])
            x1          x2
x1 0.070632534 0.008217225
x2 0.008217225 0.079011072
```

Y = neg

Y = pos

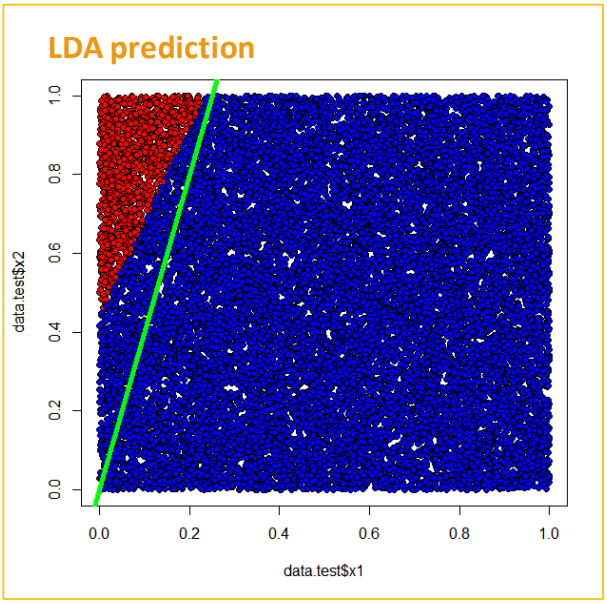


Figure 6 - Frontière induite par l'analyse discriminante linéaire (en vert la frontière optimale)

3.4 Régression logistique

Contrairement à ce qui est annoncé sur Wikipédia (version 04/05/2013)[5], une hypothèse sur les distributions est sous-jacente à la régression logistique : le rapport des distributions conditionnelles est issue d'une famille de lois incluant la normalité multidimensionnelle (Bardos, 2001 ; page 64). La méthode est donc semi-paramétrique. Le postulat est cependant moins fort par rapport au bayésien naïf ou à l'analyse discriminante.

```
#logistic regression
model.glm <- glm(y ~ ., data = train.data, family = binomial)
print(summary(model.glm))

#function for logistic regression prediction
prediction.glm <- function(model,test.data){
  return(factor(ifelse(predict(model,newdata=test.data)>0.5,2,1)))
}

#error rate
print(error.rate.plot(test.data,prediction.glm(model.glm,test.data)))
```

Nous obtenons un taux d'erreur de **7.45%**, bien meilleur que ceux du bayésien naïf et de l'analyse discriminante.

[5] Version originale en anglais, http://en.wikipedia.org/wiki/Linear_classifier ; sa traduction en Français, http://fr.wikipedia.org/wiki/Classifieur_linéaire

```
Call:
glm(formula = y ~ ., family = binomial, data = train.data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.6337   0.0803   0.2125   0.5173   1.4270

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   0.6660     0.4759   1.399 0.161699
x1            6.7790     1.0986   6.171 6.79e-10 ***
x2           -2.2961     0.6943  -3.307 0.000944 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 267.09  on 299  degrees of freedom
Residual deviance: 182.55  on 297  degrees of freedom
AIC: 188.55

Number of Fisher Scoring iterations: 6

>
> #function for logistic regression prediction
> prediction.glm <- function(model,test.data){
+   return(factor(ifelse(predict(model,newdata=test.data)>0.5,2,1)))
+ }
>
> print(error.rate.plot(test.data,prediction.glm(model.glm,test.data)))
     pred
        1     2
  neg  2290   937
  pos   554 16219
[1] 0.07455
```

Nous nous rapprochons de la frontière théorique (Figure 7).

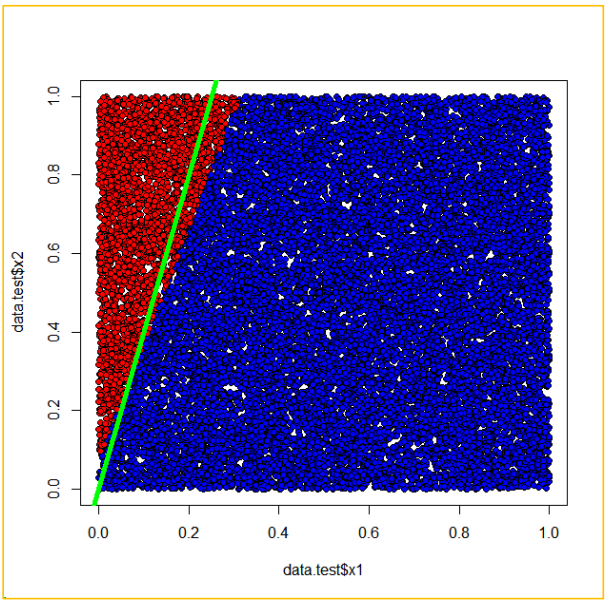


Figure 7 - Frontière induite par la régression logistique (en vert la frontière optimale)

3.5 Perceptron simple

Le perceptron est non-paramétrique. Il cherche à minimiser l'erreur quadratique avec pour seule contrainte la représentation adoptée. Dans le cas du perceptron simple, un perceptron sans couche cachée, nous obtenons une estimation des coefficients du classifieur linéaire[6].

Nous devons installer et charger le package **nnet** avant de procéder à la modélisation.

```
#single layer perceptron (neural network)
library(nnet)
```

[6] Voir « Paramétrer le perceptron multicouche », http://tutoriels-data-mining.blogspot.fr/2013/04/parametrer-le-perceptron-multicouche.html

```
model.nn <- nnet(y ~ ., data = train.data,skip=TRUE,size=0)
print(summary(model.nn))
#fonction de prédiction
prediction.nn <- function(model,test.data){
  return(factor(predict(model,newdata=test.data,type="class")))
}
#taux d'erreur et graphique
print(error.rate.plot(test.data,prediction.nn(model.nn,test.data)))
```

Nous obtenons un taux d'erreur de **8.715%**.

```
> #single layer perceptron (neural network)
> library(nnet)
> model.nn <- nnet(y ~ ., data = train.data,skip=TRUE,size=0)
# weights:  3
initial  value 152.044979
iter  10 value 91.275586
final  value 91.275522
converged
> print(summary(model.nn))
a 2-0-1 network with 3 weights
options were - skip-layer connections  entropy fitting
 b->o i1->o i2->o
 0.67  6.78 -2.30
>
> prediction.nn <- function(model,test.data){
+   return(factor(predict(model,newdata=test.data,type="class")))
+ }
>
> print(error.rate.plot(test.data,prediction.nn(model.nn,test.data)))
     pred
        neg   pos
  neg  1571  1656
  pos    87 16686
[1] 0.08715
```

Nous avons le même type de décalage que les autres par rapport à la frontière optimale (Figure 8).

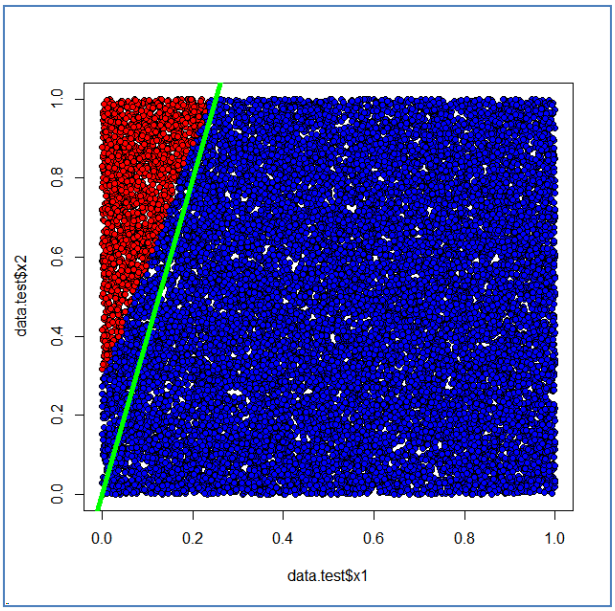


Figure 8 - Frontière induite par le perceptron simple (en vert la frontière optimale)

3.6 Support vector machine

Avec un noyau linéaire, le modèle issu des machines à support de vecteur peut s'exprimer sous la forme d'une combinaison linéaire des descripteurs. Peu de logiciels fournissent l'équation explicite de la frontière cependant. La procédure **svm()** du package **e1071** ne déroge pas à la règle et se

contente de produire la liste des points supports et des pondérations associées. C'est déjà pas mal. Il est possible avec ces informations de déployer le modèle sur des individus supplémentaires.

```
#linear support vector machine
library(e1071)
model.svm <- svm(y ~ ., data = train.data,kernel="linear")
print(model.svm)
#function for svm prediction
prediction.svm <- function(model,test.data){
  return(predict(model,newdata=test.data))
}
#taux d'erreur et représentation de la frontière
print(error.rate.plot(test.data,prediction.svm(model.svm,test.data)))
```

Nous obtenons un taux d'erreur de **7.465%**, le meilleur sur l'ensemble des méthodes linéaires présentées dans cette section.

```
Call:
svm(formula = y ~ ., data = train.data, kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.5

Number of Support Vectors:  92

>
> #function for svm prediction
> prediction.svm <- function(model,test.data){
+   return(predict(model,newdata=test.data))
+ }
>
> print(error.rate.plot(test.data,prediction.svm(model.svm,test.data)))
     pred
        neg   pos
  neg  1856  1371
  pos   122 16651
[1] 0.07465
```

On se rapproche de la frontière théorique (Figure 9).

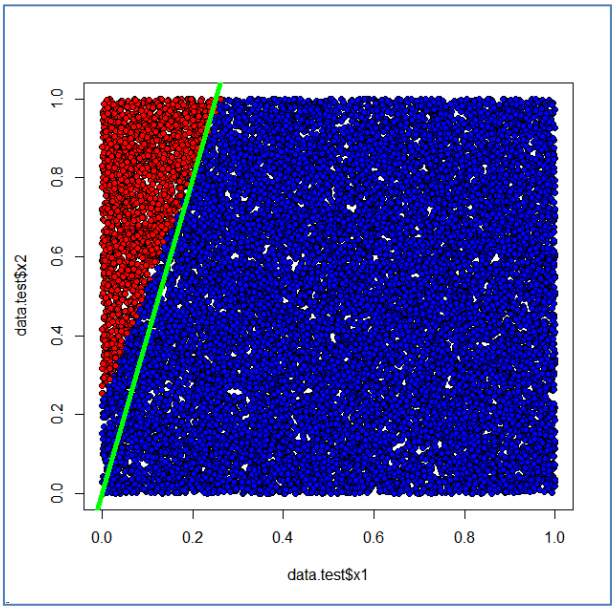


Figure 9 - Frontière induite par le SVM linéaire (en vert la frontière optimale)

Le package e1071 propose un excellent outil qui permet de visualiser dans le plan les régions associées aux classes, et par conséquent les frontières. Il identifie également les points supports qui ont permis d'établir la solution.

```
#dessin des points « o » et des points supports « x »
plot(model.svm,data=train.data,svSymbol="x",dataSymbol="o")
```

Le graphique est transposé par rapport aux nôtres (X2 en abscisse, X1 en ordonnée). Mais la nature des résultats est bien la même : les régions associées aux classes sont délimitées linéairement. A cause du bruit (noise = 5%), les points supports sont relativement nombreux malgré la simplicité du concept (Figure 10). Ils seraient moins nombreux et situés le long de la frontière si les données n'étaient pas bruitées.

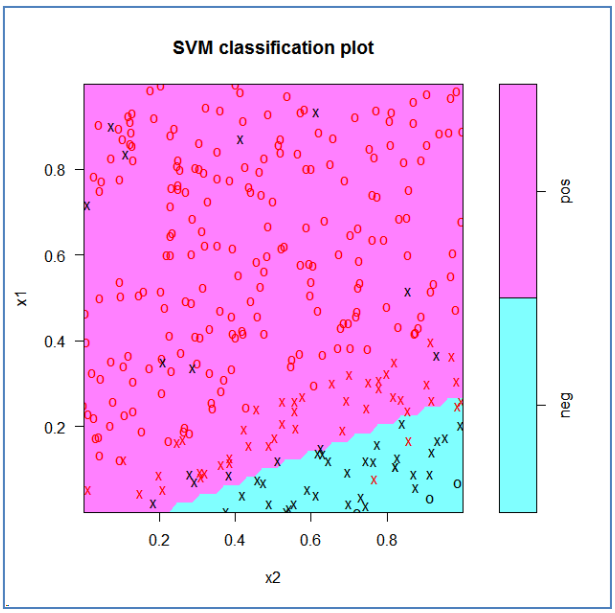


Figure 10 - SVM : régions associées aux classes, points ("o") et points supports ("x")

3.7 D'autres méthodes non linéaires

Nous savons que la frontière est linéaire parce qu'elle a été sciemment générée. Dans les études réelles, nous ne disposons pas de cette information. Nous serions donc obligés de tester différentes solutions avant de choisir le modèle adapté au problème à traiter. A priori disions nous en introduction, les modèles non linéaires disposant d'un système de représentation plus performant seraient tentants. Mais, outre les difficultés d'interprétation des résultats, nous nous heurtons à la plus grande variabilité de ces techniques. Il nous faut plus d'observations pour conjurer le surapprentissage. Or, ce n'est pas toujours possible.

Dans cette section, nous étudions le comportement de quelques approches non linéaires. Nous analysons leurs performances et la forme des frontières induites.

3.7.1 Arbres de décision – CART

Un arbre de décision est un modèle linéaire par morceaux, à la manière d'un perceptron multicouches. Sauf que chaque bout de droite doit être forcément perpendiculaire à un des axes puisque la segmentation de chaque sommet est effectuée à l'aide d'une et une seule des variables

prédictives. Nous utilisons la procédure **rpart()** – qui s'apparente à la méthode CART (Breiman et al., 1984) – dans notre expérimentation.

```
#decision tree learning
library(rpart)
model.tree <- rpart(y ~ ., data = train.data)
print(model.tree)
pred.tree <- predict(model.tree,newdata=test.data,type="class")
print(error.rate.plot(test.data,pred.tree))
```

L'arbre enchaîne les deux variables prédictives X1 et X2 dans les segmentations.

```
> model.tree <- rpart(y ~ ., data = train.data)
> print(model.tree)
n= 300

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 300 49 pos (0.16333333 0.83666667)
   2) x1< 0.1527279 49 14 neg (0.71428571 0.28571429)
     4) x2>=0.4041346 31  1 neg (0.96774194 0.03225806) *
     5) x2< 0.4041346 18  5 pos (0.27777778 0.72222222) *
   3) x1>=0.1527279 251 14 pos (0.05577689 0.94422311)
     6) x1< 0.2079262 22  5 pos (0.22727273 0.77272727)
      12) x2>=0.6219094 7  2 neg (0.71428571 0.28571429) *
      13) x2< 0.6219094 15  0 pos (0.00000000 1.00000000) *
     7) x1>=0.2079262 229  9 pos (0.03930131 0.96069869) *
> pred.tree <- predict(model.tree,newdata=test.data,type="class")
> print(error.rate.plot(test.data,pred.tree))
     pred
        neg   pos
  neg  1859  1368
  pos   327 16446
[1] 0.08475
```

Le taux d'erreur est de **8.475%**. Même si l'estimation de la frontière paraît visuellement assez grossière (Figure 11), elle est parfaitement alignée sur la droite optimale. A la sortie, les performances sont tout à fait comparables à celles des méthodes linéaires.

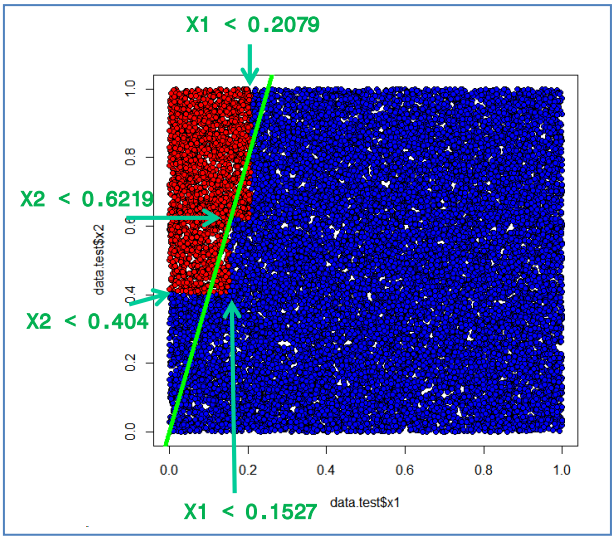


Figure 11 - Frontière induite par un arbre de décision (en vert la frontière optimale)

L'approximation dépend du pas de l'escalier qui, lui-même, est tributaire de la taille de l'échantillon d'apprentissage. Si elle est infinie – on peut toujours rêver – le pas sera suffisamment fin pour que la droite de séparation soit parfaitement reproduite. De fait, les performances des arbres, plus que tout autre méthode, dépend fortement de la disponibilité des observations.

3.7.2 Forêts aléatoires - Random Forest

Les « forêts aléatoires » (*Random Forest* - RF, Breiman & Cutler) constituent la quintessence de l'approche bagging. Il s'agit d'agréger des modèles individuels construits sur des échantillons bootstrap. Concernant les random forest, nous agrégeons des arbres construits de manière particulière. Le processus agit sur deux aspects : il réduit la dépendance à l'échantillon, il améliore le pouvoir de représentation du modèle. Dans le cas des arbres, il transcende la contrainte de représentation ci-dessus (Figure 11, les droites de séparation sont perpendiculaires aux axes) au point de pouvoir approcher directement la frontière linéaire, **avec le même échantillon d'apprentissage de 300 observations**.

Nous installons et chargeons le package **rf** avant de pouvoir utiliser la procédure **randomForest()**.

```
#random forest
library(randomForest)
model.rf <- randomForest(y ~ ., data = train.data)
print(model.rf)
pred.rf <- factor(predict(model.rf,newdata=test.data,type="response"))
print(error.rate.plot(test.data,pred.rf))
```

Le taux d'erreur est de **6.96%**. C'est le meilleur modèle de notre comparatif.

```
Call:
 randomForest(formula = y ~ ., data = train.data)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 1

        OOB estimate of  error rate: 7%
Confusion matrix:
    neg pos class.error
neg  32  17  0.34693878
pos   4 247  0.01593625
> pred.rf <- factor(predict(model.rf,newdata=test.data,type="response"))
> print(error.rate.plot(test.data,pred.rf))
     pred
        neg   pos
  neg  2088  1139
  pos   253 16520
[1] 0.0696
```

Le classifieur n'est pas linéaire par nature. Il réussit néanmoins à produire une approximation de qualité (Figure 12).

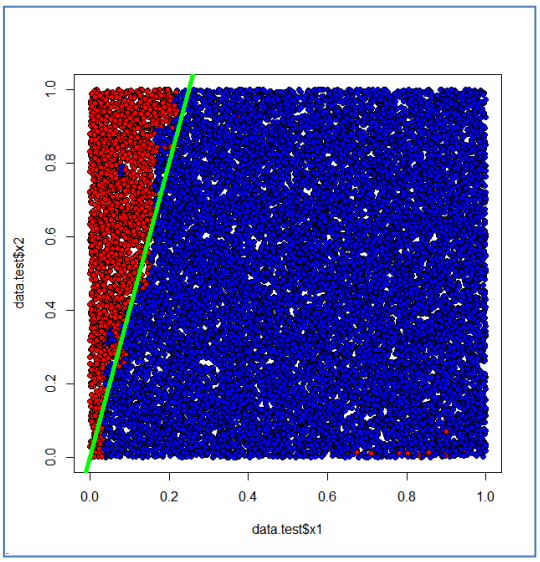


Figure 12 - Frontière induite par les random forest (en vert la frontière optimale)

C'est fort, vraiment très fort. Surtout que, par rapport aux caractéristiques du random forest, nous nous plaçons dans des conditions extrêmes : le nombre d'instances est faible, produire des échantillons bootstrap suffisamment dissemblables est difficile ; nous n'avons que les deux variables pertinentes dans la base ; le type de représentation du modèle de base (les arbres construisent une succession de droites perpendiculaires aux axes) est éloigné du type de frontière séparant les positifs des négatifs (une droite oblique).

3.7.3 k-Plus proches voisins (Nearest Neighbors)

Dans cette section, nous utilisons une méthode non contrainte par un système de représentation : la méthode des plus proches voisins[7] (k-PPV). La forme de la frontière n'est pas explicite, chaque observation définit une « zone d'influence » pour le classement de nouvelles observations.

La méthode est paramétrée par « k », le nombre de voisins à prendre en considération lors du classement d'un nouvel individu. Le choix de « k » est toujours difficile. Lorsque nous la diminuons (vers k = 1), nous améliorons la capacité à reproduire des concepts complexes, mais nous exacerbons la dépendance à l'échantillon d'apprentissage. Ce qui peut être particulièrement néfaste lorsque les données sont bruitées. Lorsque nous l'augmentons, nous agissons dans le sens inverse, le modèle est moins puissant mais de variance plus faible. Dans notre exemple, nous testons « k = 1 » et « k = 5 ».

La procédure **knn()** est incluse dans le package **class**. Il n'y a pas de phase d'apprentissage à proprement parler, nous déployons directement le système sur l'échantillon test. Pour chaque individu à classer, il recherche les « k » voisins qui lui sont les plus proches dans l'échantillon d'apprentissage et lui assigne l'étiquette la plus représentée.

```
#nearest neighbor
library(class)
#k = 1
print(error.rate.plot(test.data,knn(train.data[,2:3],test.data[,2:3],train.data$y,k=1)))
```

Avec un taux d'erreur de **10.84%**, les performances semblent honorables par rapport à certaines techniques linéaires.

```
> print(error.rate.plot(test.data,knn(train.data[,2:3],test.data[,2:3],train.data$y,k=1)))
     pred
        neg   pos
  neg  2006  1221
  pos   947 15826
[1] 0.1084
```

En effet, si la frontière semble reproduite plus ou moins fidèlement (Figure 13), nous constatons également que certaines observations mal étiquetées ont défini des zones d'influence erronées dans les deux régions : les pâtés de points rouges (modalité négative de Y) parmi les bleus (modalité positive), et inversement. Bien sûr, cette situation n'aurait pas existé si les classes n'avaient pas été bruitées.

[7] http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

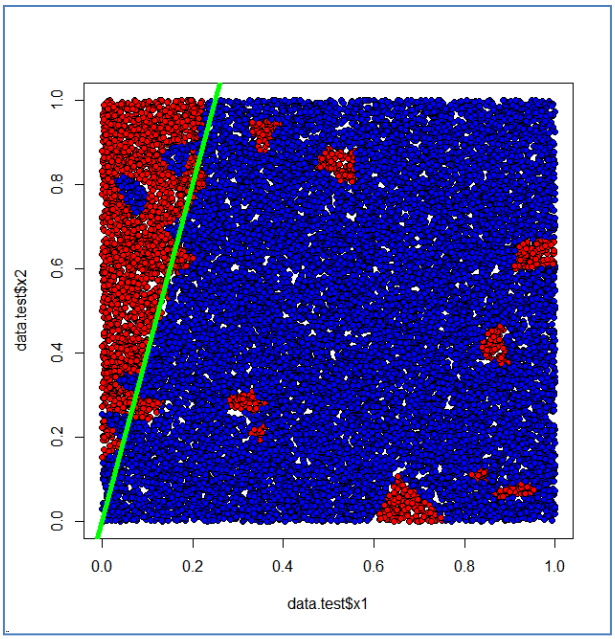


Figure 13 - Zones d'affectation induites par le 1-ppv (en vert la frontière optimale)

Ces zones disparaissent lorsque nous passons à k = 5, améliorant le taux d'erreur (**7.55%**).

```
> #k=5
> print(error.rate.plot(test.data,knn(train.data[,2:3],test.data[,2:3],train.data$y,k=5)))
     pred
        neg   pos
  neg  1976  1251
  pos   260 16513
[1] 0.07555
```

Il reste néanmoins les zones de mauvaise décision tout au long de la frontière (Figure 14).

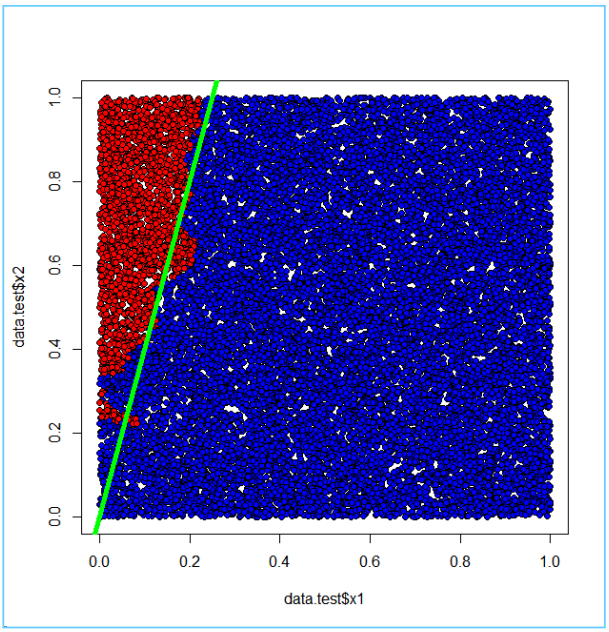


Figure 14 - Zones d'affectation induites par le 5-ppv (en vert la frontière optimale)

3.8 Récapitulatif des résultats

Comme tout le monde, nous allons produire un tableau de résultats. A la différence que, d'emblée, nous conseillons de prendre beaucoup de recul par rapport aux chiffres annoncés : tout ceci ne

repose que sur un seul essai c.-à-d. avec un seul couple d'échantillons d'apprentissage et de test. Il est très vraisemblable qu'ils soient différents (un peu ? beaucoup ? nous préciserons cela dans les sections 6 et 7) sur un autre couple d'échantillons.

Méthode	Taux d'erreur (%)
Modèle optimal	**5.19**
Méthodes linéaires	
Bayésien naïf	10.23
Analyse discriminante linéaire	10.35
Régression logistique	7.45
Perceptron simple	8.71
SVM Linéaire	7.46
Méthodes non linéaires	
Arbre de décision	8.48
Random Forest	6.96
1-ppv	10.84
5-ppv	7.55

Finalement, à part les 3 cas pathologiques, parce que reposant sur des hypothèses inadaptées à nos données (bayésien naïf, analyse discriminante linéaire) ou mal paramétré (1-ppv), les méthodes se tiennent à peu près. Sauf, et c'est un élément très important, que les classifieurs linéaires proposent des modèles explicites que l'on peut aisément interpréter (les coefficients de la combinaison linéaire) et déployer. Si l'on met à part les arbres, déployer les forêts aléatoires (il faut stocker tous les arbres intermédiaires) ou les plus proches voisins (il faut stocker tous les individus)[8] est fastidieux.

4 Traitements sous Tanagra

Alors que l'on produit des classifieurs linéaires, R (le package utilisé tout du moins) ne fournit pas les coefficients des droites de séparation pour certaines méthodes, en particulier pour le bayésien naïf (section 3.2) et pour le SVM linéaire (section 3.6). Dans cette section, nous reproduisons les calculs sur le même échantillon d'apprentissage à l'aide du logiciel Tanagra. L'intérêt est que ce dernier fournit l'équation explicite dès qu'il s'agit de produire un modèle linéaire. Nous pourrons ainsi confronter les coefficients obtenus.

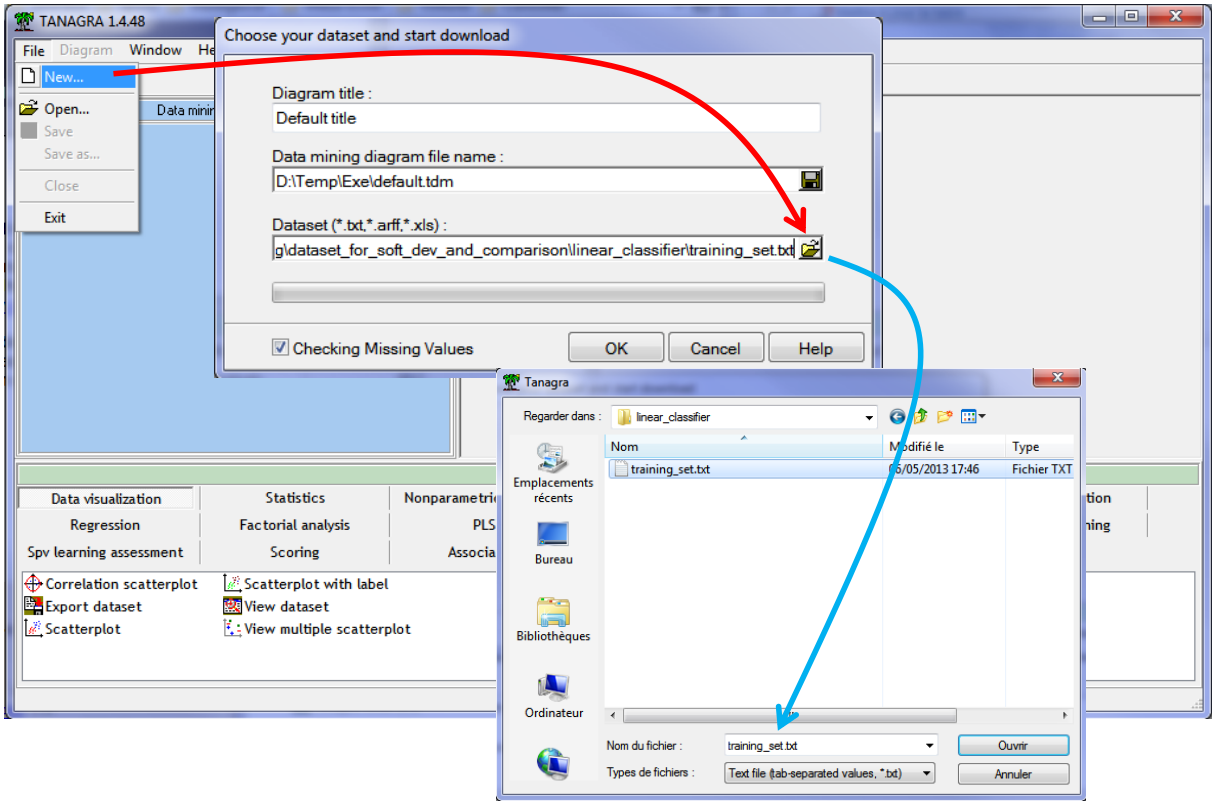
4.1 Importation des données

De R, nous exportons l'échantillon d'apprentissage avec la commande **write.table()** (fichier texte avec séparateur tabulation).

```
write.table(train.data,file="training_set.txt",sep="\t",dec=".",quote=F,row.names=F)
```

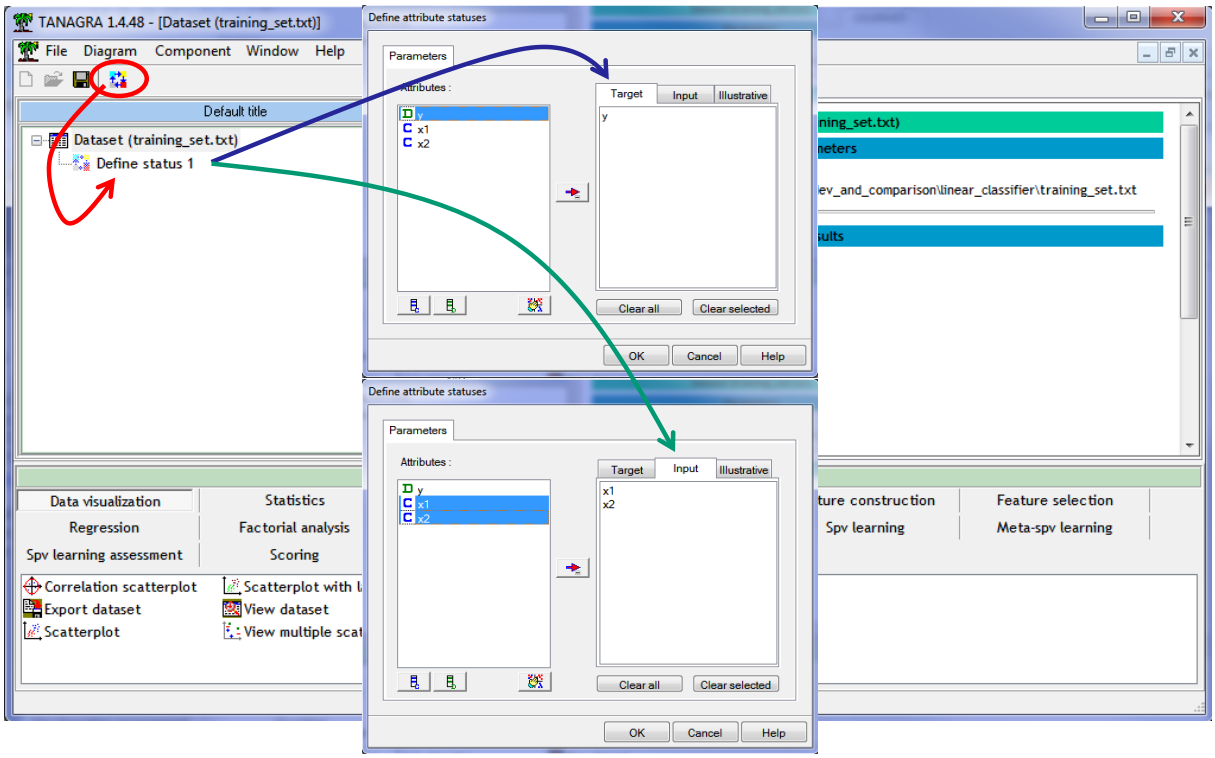
Après avoir démarré Tanagra, nous créons un nouveau diagramme (menu FILE / NEW) et nous importons le fichier de données.

[8] Voir les solutions proposées par le standard PMML pour les modèles multiples (http://www.dmg.org/v4-1/MultipleModels.html) et les plus proches voisins (http://www.dmg.org/v4-1/KNN.html).

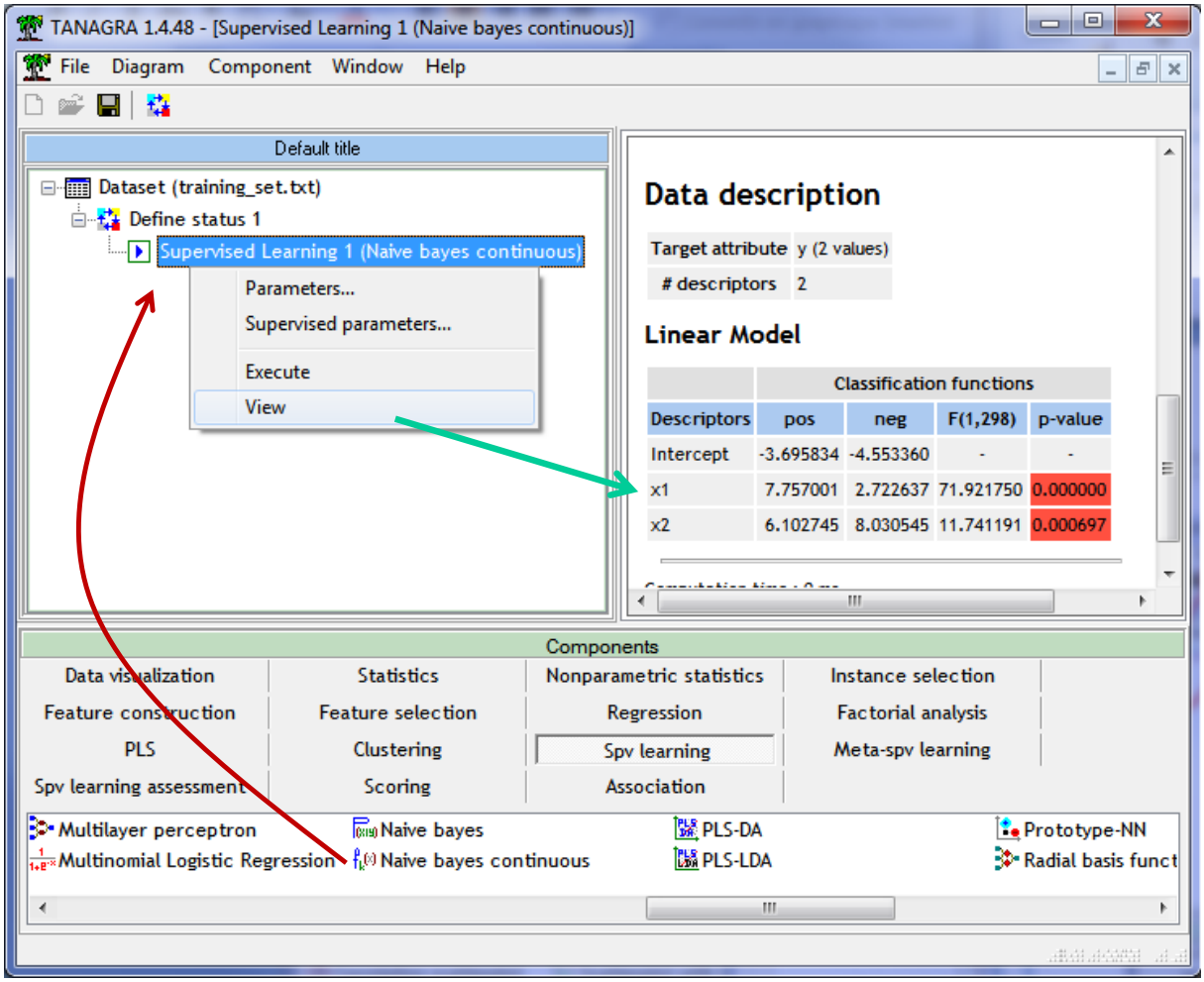


4.2 Bayésien naïf

Nous devons définir le rôle des variables avant de lancer une technique d'apprentissage supervisé. Via le raccourci dans la barre d'outils, nous insérons le composant DEFINE STATUS dans le diagramme. Nous plaçons Y en TARGET, X1 et X2 en INPUT.



Nous pouvons dès lors insérer le composant NAIVE BAYES CONTINUOUS (onglet SPV LEARNING) dans le diagramme, puis actionner le menu VIEW pour obtenir les résultats.



Des deux fonctions de classement, nous pouvons déduire l'équation de la droite de séparation induite par la méthode.

	Classification functions			Separating
Descriptors	**pos**	**neg**	**pos-neg**	**Line**
Intercept	-3.6958	-4.5534	0.8575	**-0.4448**
x1	7.7570	2.7226	5.0344	**-2.6115**
x2	6.1027	8.0305	-1.9278	**1.0000**

L'équation de la droite fournie par le bayésien naïf s'écrit :

Bayésien naïf : X2 – 2.6115 * X1 = 0.4448

Sachant que la « vraie » frontière avait été définie par :

Frontière optimale : X2 – 4.0 * X1 = 0

4.3 Les autres approches

Nous avons fait de même pour les autres techniques linéaires. Voici le diagramme de traitements sous Tanagra (Figure 15) :

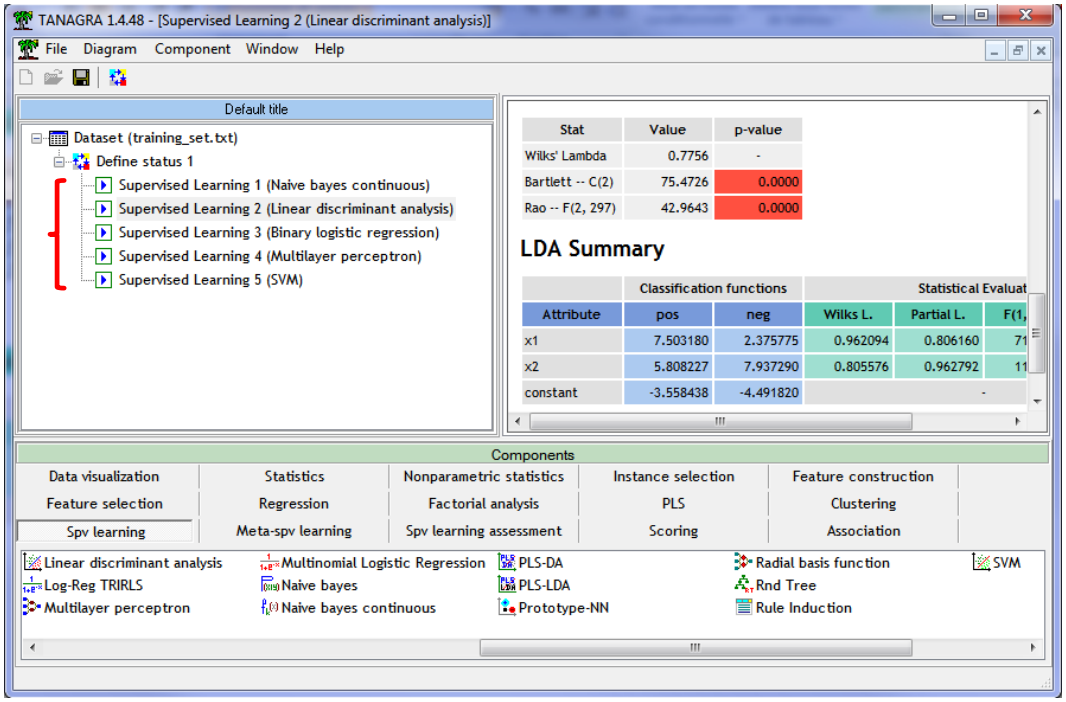


Figure 15 - Diagramme de traitements sous Tanagra - Résultats de l'analyse discriminante

Toutes les méthodes en mesure de le faire ont détecté la pertinence des deux variables (bayésien naïf, analyse discriminante linéaire, régression logistique).

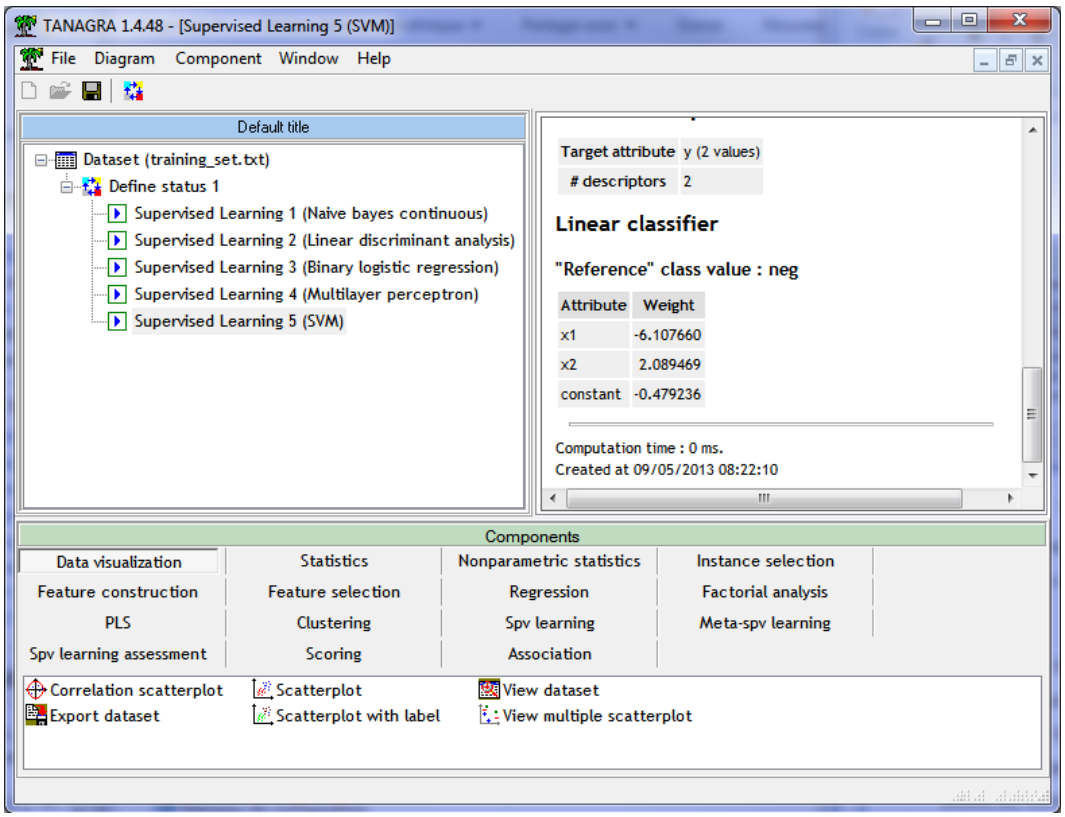


Figure 16 – Tanagra - Résultats du composant SVM (Linéaire)

Concernant le SVM (Figure 16), nous avons utilisé notre implémentation car elle fournit les coefficients du modèle linéaire. Le composant C-SVC de Tanagra basée sur la librairie LIBSVM ne le

fait pas. SVM a échoué avec les paramètres par défaut (idem sous R lorsque nous traitons les données comportant peu d'observations). Nous avons standardisé les variables (centrer et réduire).

4.4 Récapitulatif – Comparaison des coefficients des droites de séparation

Récapitulons les équations obtenues en les positionnant par rapport à la frontière théorique.

Méthode	Equation frontière
Frontière optimale	X2 - 4.0000 * X1 = 0.0000
1 - Bayésien naïf	X2 - 2.6115 * X1 = 0.4448
2 - Analyse discriminante linéaire	X2 - 2.4083 * X1 = 0.4384
3 - Régression logistique	X2 - 2.9525 * X1 = 0.2901
4 - Perceptron simple	X2 - 2.9468 * X1 = 0.2024
5 - SVM Linéaire (Composant SVM – Tanagra)	X2 - 2.9231 * X1 = 0.2294

Toutes les frontières sont décalées à gauche de la frontière optimale. Ce positionnement est la conséquence même du mode de bruitage que nous avons adopté. Mais pas de la même manière : les 3 dernières méthodes sont à peu près équivalentes sur notre échantillon d'apprentissage, elles se différencient légèrement par l'origine ; le 2 premières se démarquent des autres à la fois par la pente et par l'origine. Nous pouvons tracer ces droites dans le plan (Figure 17). C'est une autre manière de visualiser les divergences et similitudes entres les classifieurs.

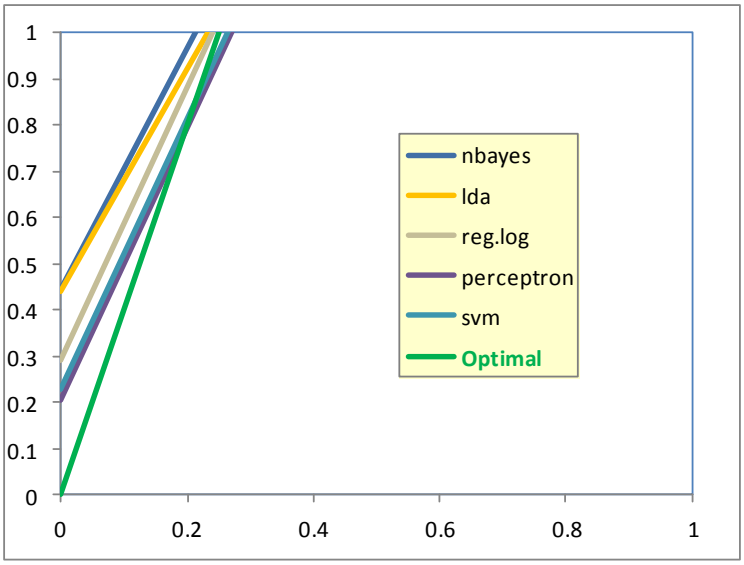


Figure 17 - Frontières induites par les différents classifieurs linéaires (en vert la frontière optimale)

5 Traitements avec les autres logiciels libres

Des logiciels libres proposent tout ou partie des approches linéaires présentées dans ce document. Certains introduisent des variantes (ex. estimation par noyau des probabilités pour le bayésien naïf, une présentation originale des résultats avec les « nomogram » sous Orange). Dans cette section, nous avons construit un diagramme de traitements similaire à celui élaboré sous Tanagra pour les logiciels Knime (http://www.knime.org/), Orange (http://orange.biolab.si/), RapidMiner (http://rapid-i.com/content/view/181/190/) et Weka (http://www.cs.waikato.ac.nz/ml/weka/). Nous récapitulons les caractéristiques des méthodes implémentées dans ces logiciels.

Méthode	Knime 2.6.4 (Figure 18)	Orange 2.6.1 (Figure 19)[9]	RapidMiner 5.2.008 (Figure 20)	Weka 3.7.4 (Figure 21)
Bayésien naïf	Hypothèse gaussienne. Affichage des moyennes et écarts-type conditionnels.	Estimation par noyau des probabilités conditionnelles. Description de l'influence des variables à l'aide un « nomogram ».	Hypothèse gaussienne. Affichage des moyennes et écarts-type conditionnels.	Possibilité d'estimation par noyau ou de discrétisation à la volée. Affichage des moyennes et écarts-type conditionnels.
Analyse discriminante linéaire	-	-	Affichage limité aux distributions a priori des classes (???).	-
Régression logistique	Affichage des coefficients de l'équation, incluant les écarts-type estimés et les tests de significativité. **X2 – 2.9524 * X1 = 0.2901**	Possibilité de sélection de variables (stratégie pas à pas). Description de l'influence des variables via un « nomogram ».	S'appuie sur l'implémentation myKLR[10] et non sur le « Fisher scoring » usuel (d'autres types de noyaux sont possibles). Affichage des coefficients de la combinaison linéaire. **X2 – 2.9606 * X1 = 0.2319**	S'appuie sur un BFGS pour l'optimisation. Fournit les coefficients sans les tests de significativité. **X2 – 2.9524 * X1 = 0.2901**
Perceptron	Pour un perceptron simple, il faut mettre une seule couche cachée avec un neurone. L'affichage standard ne montre que la décroissance de l'erreur. Les poids synaptiques sont visibles dans la description PMML. **X2 – 4.0106 * X1 = 0.0617**	-	Le perceptron simple est proposé dans un composant dédié. Affichage des coefficients de la combinaison linéaire. **X2 – 13 * X1 = 0 (???)**	0 neurone dans la couche cachée pour un perceptron simple. **X2 – 3.9886 * X1 = 0.0219**
SVM Linéaire	Mettre polynôme de degré 1 pour un SVM linéaire. Affichage des vecteurs supports par classe.	Basée sur la librairie LIBSVM. Les points supports sont visibles dans une table ou dans un graphique nuage de points en 2D (limité à 2 variables).	Basé sur LIBSVM. Propose à la fois les points supports et les coefficients de l'hyperplan pour le noyau linéaire. **X2 – 2.5144 * X1 = 0.1315**	Implémentation ad hoc. Propose l'équation de l'hyperplan séparateur lorsque le noyau est linéaire. **X2 – 2.8646 * X1 = 1.3680**

La majorité des résultats sont cohérents. Il existe quand même certaines disparités pour certains logiciels/méthodes. L'échec des calculs peut être la conséquence d'un paramétrage mal maîtrisé. J'ai essayé de mettre les mêmes paramètres d'un logiciel à l'autre, quand la transposition était possible. J'ai notamment systématiquement désactivé la normalisation/standardisation des variables puisque X1 et X2 sont définies sur la même échelle (0, 1).

[9] Documentation en ligne : http://orange.biolab.si/docs/latest/widgets/rst/

[10] http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYKLR/index.html.en

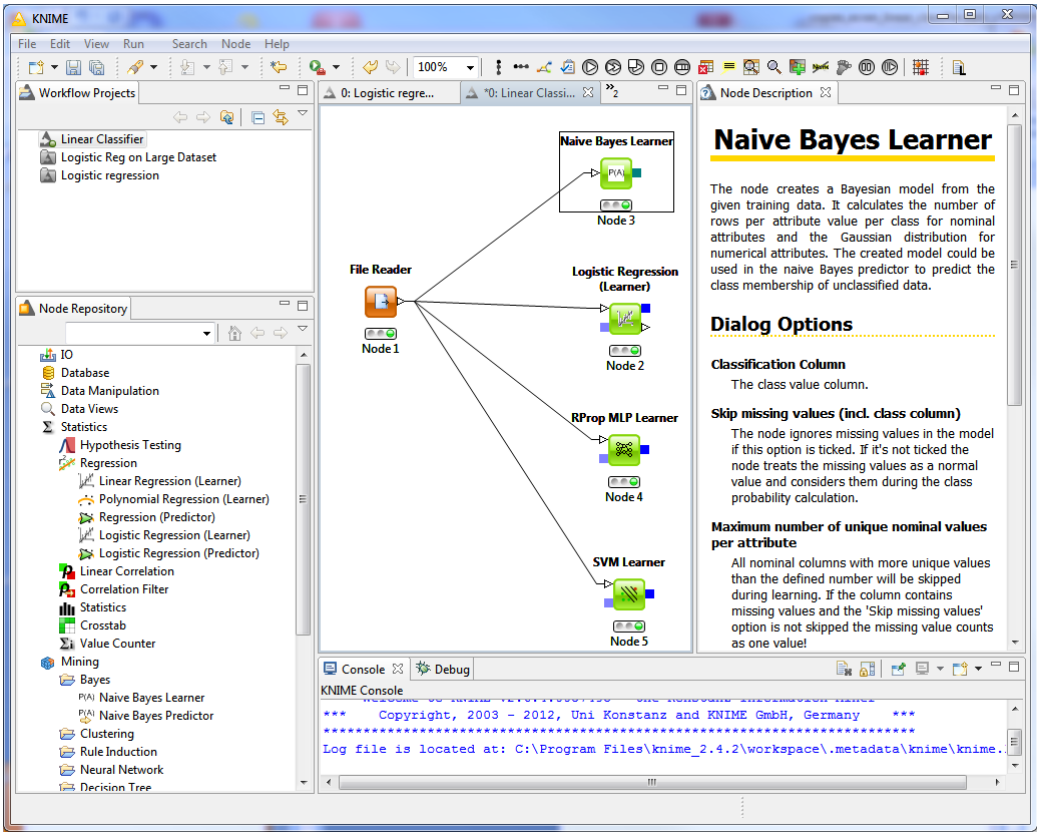


Figure 18 - Classifieurs linéaires sous Knime

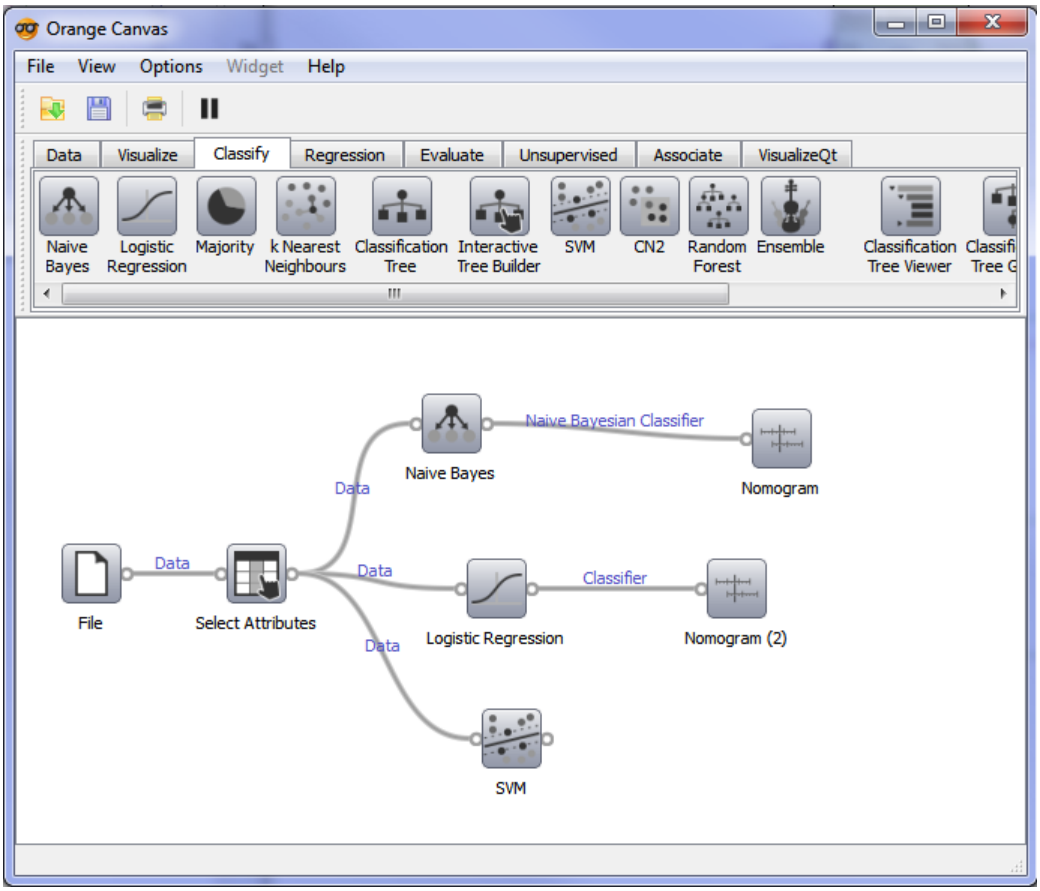


Figure 19 - Classifieurs linéaires sous Orange

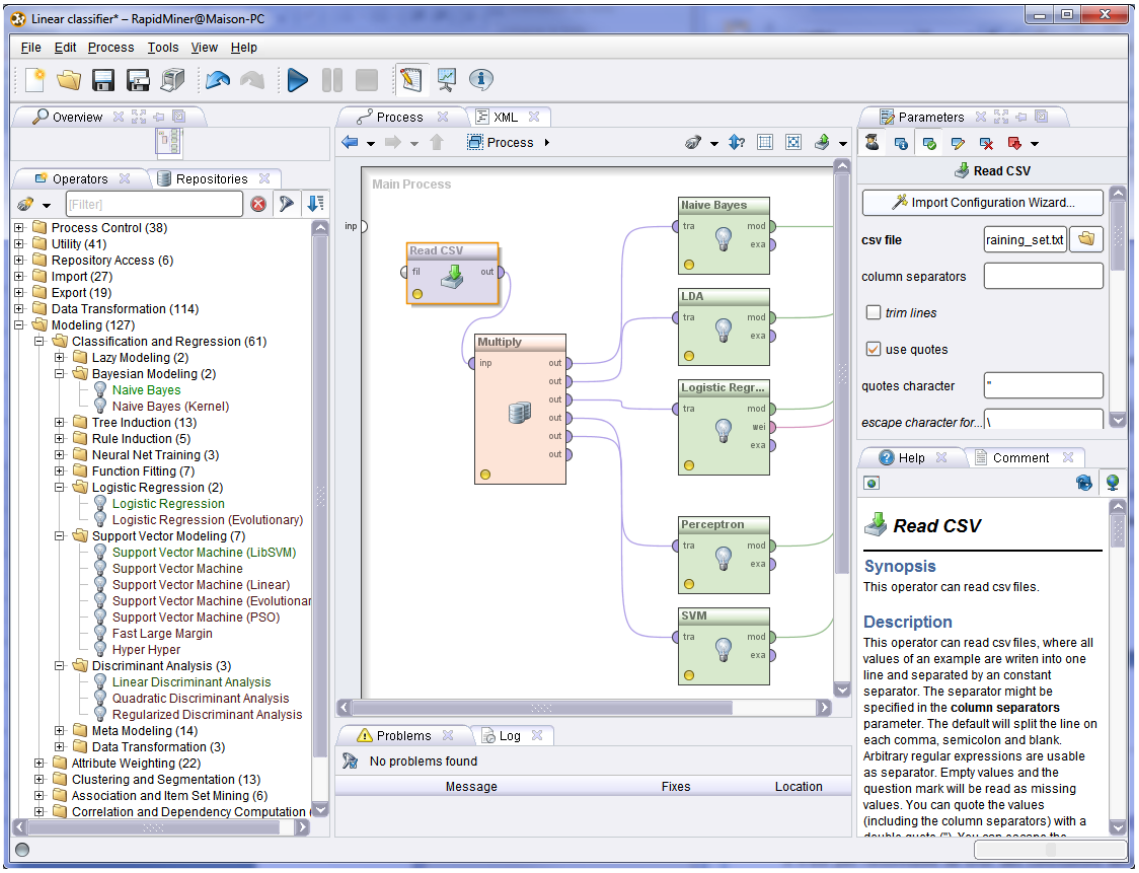


Figure 20 - Classifieurs linéaires sous RapidMiner

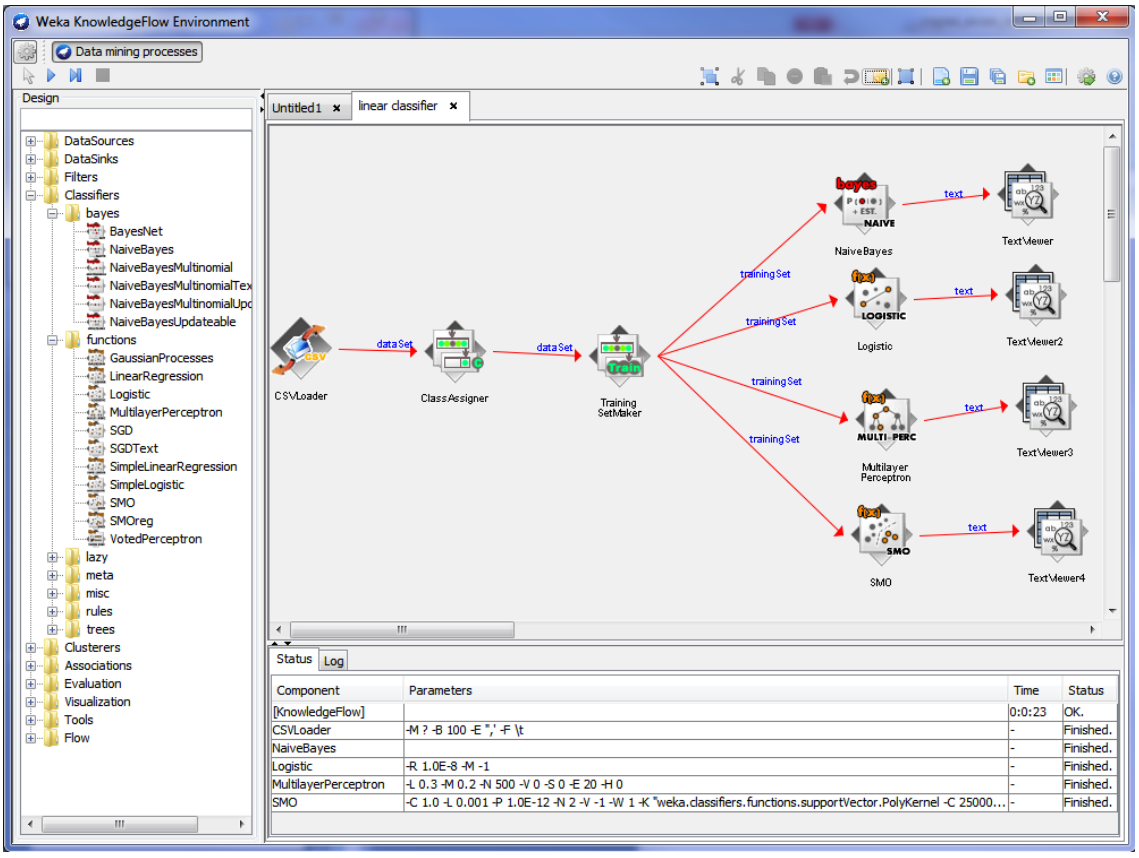


Figure 21 - Classifieurs linéaires sous Weka

6 Expérimentation 1 – Taille d'échantillon « n »

Il n'est pas raisonnable de tirer des conclusions définitives à partir de l'expérimentation menée dans la section 3. En effet, les résultats sont empreints d'une double variabilité : celle résultant de l'échantillon d'apprentissage utilisé, celle associée à l'échantillon test utilisé. Il faudrait répéter les expériences en utilisant plusieurs exemplaires à la fois d'échantillons d'apprentissage et de test. C'est faisable. Mais extraire des indicateurs synthétiques afin de démontrer des tendances de fond se révèlera rapidement difficile. Surtout si l'on souhaite étudier, comme nous désirons le faire dans cette section, l'impact de la taille de l'échantillon d'apprentissage sur les performances des classifieurs. En effet, il faudrait croiser dans ce cas : les versions de l'échantillon d'apprentissage, les versions de l'échantillon test, les différentes tailles d'échantillon et les différentes méthodes.

Dans cette section, nous ne pouvons pas déroger à la création de plusieurs fichiers d'apprentissage pour chaque taille d'échantillon « n.train » à analyser. Nous procédons à K = 100 essais. En revanche, nous pouvons contourner la génération de multiples versions d'échantillon test en créant un fichier de très grande taille. Nous réduisons ainsi la variabilité des résultats. Nous avons choisi « n.test = 100.000 » observations. De plus, en utilisant le même échantillon test tout au long de l'expérimentation, les performances des modèles sont directement comparables, c'est une forme d'appariement.

6.1 Programmation de l'expérimentation

Ainsi, nous n'avons plus qu'à croiser les différentes valeurs de **n.train = (250, 500, 1000, 2000, 3000, 5000)** et les 5 techniques d'apprentissage (bayésien naïf, analyse discriminante linéaire, régression logistique, perceptron simple et SVM linéaire), et répéter chaque itération K = 100 fois. Nous détaillons ci-après le programme pour R.

```
#fonction pour calculer le taux d'erreur sur l'échantillon test data.test
#pred représente la prédiction d'un modèle quelconque
error.rate <- function(data.test,pred){
  mc <- table(data.test$y,pred)
  err.rate <- 1-sum(diag(mc))/sum(mc)
  return(err.rate)
}

#une expérimentation pour une taille « n » de l'échantillon d'apprentissage
#la fonction renvoie un vecteur le taux d'erreur en test de chaque méthode
experiments <- function(n,test.set){

  #génération de l'échantillon d'apprentissage de taille « n »
  learning <- generate.data(n,p,noise)
  print(nrow(learning))

  #préparation du vecteur collectant les résultats
  #5 méthodes à tester donc 5 cases à préparer
  result <- numeric(5)
```

```
  #naive bayes classifier
  model.nb <- naiveBayes(y ~ ., data = learning)
  result[1] <- error.rate(test.set,prediction.nb(model.nb,test.set))

  #linear discriminant analysis
  model.lda <- lda(y ~ ., data = learning)
  result[2] <- error.rate(test.set,prediction.lda(model.lda,test.set))

  #logistic regression
  model.glm <- glm("y ~ .", data = learning, family = binomial)
  result[3] <- error.rate(test.set,prediction.glm(model.glm,test.set))

  #single layer perceptron
  model.nn <- nnet(y ~ ., data = learning,skip=TRUE,size=0)
  result[4] <- error.rate(test.set,prediction.nn(model.nn,test.set))

  #linear support vector machine
  model.svm <- svm(y ~ ., data = learning,kernel="linear")
  result[5] <- error.rate(test.set,prediction.svm(model.svm,test.set))

  print(result)
  return(result)
}

#différentes tailles d'échantillon d'apprentissage à évaluer
size.training <- c(250,500,1000,2000,3000,5000)

#génération de l'échantillon test unique utilisée
#durant toute l'expérimentation
set.seed(25032003)
other.data.test <- generate.data(100000,p,noise)

#charger les packages nécessaires aux méthodes (au cas où…)
library(MASS)
library(e1071)
library(nnet)

#une expérimentation pour une taille d'échantillon « size.learning » donnée
one.expe.session <- function(size.learning){
  results <- mapply(experiments,size.learning,MoreArgs=list(test.set=other.data.test))
  return(results)
}

#K : nombre de répétition de chaque expérimentation
K <- 100
set.seed(05092008)
all.results <- replicate(K,one.expe.session(size.learning=size.training),simplify="matrix")

#all.results est un tableau avec K = 100 colonnes (K = 100 essais)
```

```
#et 30 lignes (croisements entre 5 méthodes et  6 tailles d'échantillons)

#mise en forme des résultats dans un nouveau tableau croisant :
#en ligne, les 5 méthodes ; en colonne, les 6 tailles d'échantillon
#l'indicateur synthétique est la moyenne des K = 100 essais
mean.results <- matrix(0,nrow=5,ncol=length(size.training))
colnames(mean.results) <- size.training
rownames(mean.results) <- c("naive.bayes","lda","log.reg","perceptron","svm.linear")
for (i in 1:5){
  for (j in 1:length(size.training)){
    mean.results[i,j] <- mean(all.results[i+(j-1)*5,])
  }
}
print(mean.results)

#même calcul mais avec la médiane comme indicateur synthétique
med.results <- mean.results
for (i in 1:5){
  for (j in 1:length(size.training)){
    med.results[i,j] <- median(all.results[i+(j-1)*5,])
  }
}
print(med.results)
```

Les calculs sont assez longs. Nous avons vraiment intérêt à effectuer plusieurs essais pour calibrer l'expérimentation, s'assurer que les résultats sont générés correctement, et qu'ils sont collectés de manière adéquate.

6.2 Analyse des résultats

Voici la courbe des erreurs moyennes selon la taille d'échantillon pour chaque méthode (Figure 22).

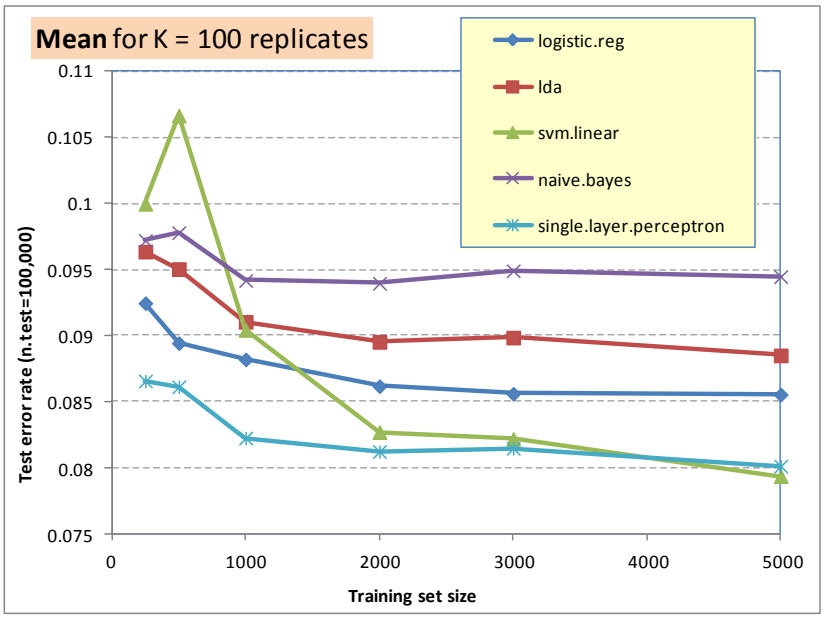


Figure 22 – Moyenne des performances des méthodes selon la taille de l'échantillon d'apprentissage

Concernant les performances comparées :

1. Manifestement, les méthodes reposant sur des hypothèses qui ne concordent pas avec nos données sont pénalisées : le bayésien naïf, l'analyse discriminante et, dans une moindre mesure, la régression logistique. Plus le postulat sous-jacent à la méthode est restrictive, moins performante est la méthode sur le concept que nous avons créé.
2. SVM et le perceptron sont donc les meilleurs sur nos données, à égalité.

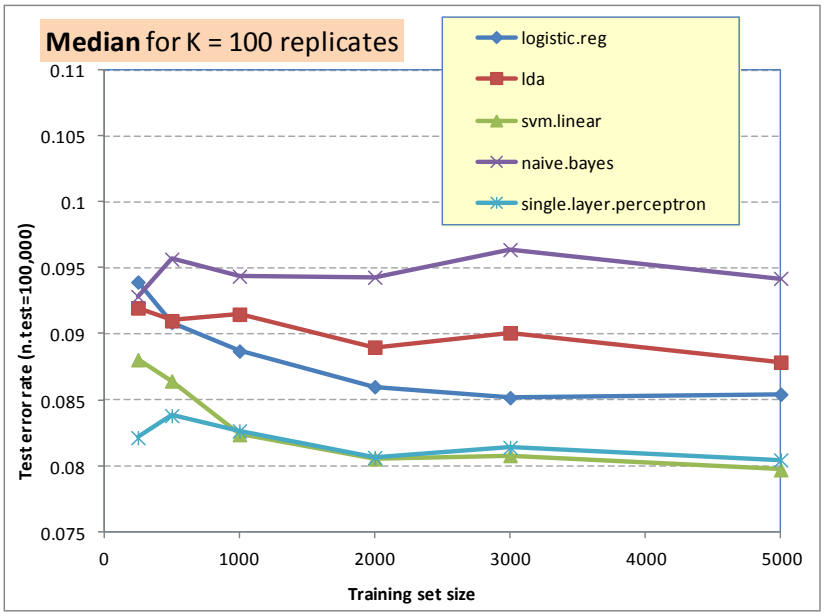


Figure 23 - Médiane des performances des méthodes selon la taille de l'échantillon d'apprentissage

Mais nous ne pouvons pas nous arrêter à ces constats :

3. Les SVM semblent avoir un comportement désastreux sur les petits échantillons (n.train ≤ 500). En réalité, la bibliothèque de calcul a planté sur une forte proportion des essais. La courbe est tout autre - pour les SVM - lorsque nous prenons la médiane comme indicateur synthétique (Figure 23). Ce genre de plaisanterie douteuse arrive parfois dans les expérimentations. Il faut absolument vérifier, re-vérifier, et encore vérifier les résultats avant de penser à les publier.
4. Les performances s'améliorent lorsque la taille de l'échantillon d'apprentissage augmente. Heureusement. Le contraire aurait été très inquiétant.
5. A partir de « n.train = 2000 », les gains sont marginaux cependant. Cela est du en grande partie à la simplicité du concept reliant la variable cible au descripteur (une frontière linéaire).
6. Pour autant, aucune des méthodes ne converge vers la performance optimale (5%), même avec un échantillon d'apprentissage de taille conséquente. On le comprend pour les méthodes reposant sur des hypothèses non adaptées à nos données. Un peu moins lorsqu'il s'agit des techniques telles que le perceptron ou les SVM.
7. **Il faut y voir la conséquence du bruitage des classes**. En effet, **lorsque nous générons un échantillon d'apprentissage exempt de bruit, le perceptron et les SVM trouvent la frontière théorique, la régression logistique aussi d'ailleurs**. Pas le bayésien naïf et l'analyse discriminante qui reposent sur des biais d'apprentissage trop contraignants. Ils restent en retrait.

7 Expérimentation 2 – Nombre de variables « p »

Nous souhaitons évaluer l'influence de la dimensionnalité à taille d'échantillon fixée « n.train = 500 ». Nous savons que 2 variables (X1, X2) seulement sont pertinentes pour la discrimination. Que se passe-t-il si nous ajoutons des variables qui n'ont aucun lien avec la cible ? Il s'agit dans cette section d'apprécier la robustesse des méthodes par rapport à la présence de descripteurs non pertinents. C'est une autre forme de bruitage.

7.1 Programmation de l'expérimentation

Nous croisons les différentes valeurs de **p = (2, 5, 10, 25, 50, 70)** - *sachant que [p-2] d'entre elles ne sont pas pertinentes pour la prédiction* - et les 5 techniques d'apprentissage (bayésien naïf, analyse discriminante linéaire, régression logistique, perceptron simple et SVM linéaire). Nous répétons chaque itération K = 100 fois. Nous détaillons ci-après le programme pour R. Il présente de très fortes similitudes avec le code de la section précédente.

```
#Expérimentation : influence du nombre de descripteurs p
#Taille d'échantillon d'apprentissage = 500
experiments.dimension <- function(p,test.set){

  #learning set
  learning <- generate.data(500,p,noise)

  #vector containing the results
  result <- numeric(5)

  #naive bayes classifier
  model.nb <- naiveBayes(y ~ ., data = learning)
  result[1] <- error.rate(test.set,prediction.nb(model.nb,test.set))

  #linear discriminant analysis
  model.lda <- lda(y ~ ., data = learning)
  result[2] <- error.rate(test.set,prediction.lda(model.lda,test.set))

  #logistic regression
  model.glm <- glm("y ~ .", data = learning, family = binomial)
  result[3] <- error.rate(test.set,prediction.glm(model.glm,test.set))

  #single layer perceptron
  model.nn <- nnet(y ~ ., data = learning,skip=TRUE,size=0)
  result[4] <- error.rate(test.set,prediction.nn(model.nn,test.set))

  #linear support vector machine
  model.svm <- svm(y ~ ., data = learning,kernel="linear")
  result[5] <- error.rate(test.set,prediction.svm(model.svm,test.set))

  print(result)
  return(result)
```

```
}

#génération de l'échantillon test unique
#n.test = 100000 individus et p = 100 variables prédictives
set.seed(25032003)
second.data.test <- generate.data(100000,100,noise)

#print
print(colnames(second.data.test))
print(table(second.data.test$y))

#various dimension size
size.p <- c(2,5,10,25,50,70)

#one experiment for various dimensionality
one.expe.dimension <- function(size.dimension){
  results <- mapply(experiments.dimension,size.dimension,MoreArgs=list(test.set=second.data.test))
  return(results)
}

#repeat K times the experiments
K <- 100
set.seed(21102011)
all.results <- replicate(K,one.expe.dimension(size.dimension=size.p),simplify="matrix")

#retrieving the results
mean.results <- matrix(0,nrow=5,ncol=length(size.p))
colnames(mean.results) <- size.p
rownames(mean.results) <- c("naive.bayes","lda","log.reg","perceptron","svm.linear")

for (i in 1:5){
  for (j in 1:length(size.p)){
    mean.results[i,j] <- mean(all.results[i+(j-1)*5,])
  }
}
print(mean.results)

#median.results
med.results <- mean.results
for (i in 1:5){
  for (j in 1:length(size.p)){
    med.results[i,j] <- median(all.results[i+(j-1)*5,])
  }
}
print(med.results)
```

L'échantillon test « second.data.test » est généré une seule fois avec 100000 observations et 100 variables. Il est utilisable pour les différentes configurations que nous souhaitons évaluer p = (2, 5, 10, 25, 50, 70).

```
> print(table(second.data.test$y))

  neg   pos
16247 83753
```

Il est composé de 16.25% de négatifs, et 83.75% de positifs. C'est une information importante. **Le taux d'erreur du classifieur par défaut** (on effectue une prédiction endogène sans s'appuyer sur les descripteurs, ici on prédirait systématiquement « positif ») est de **16.25%.** Nous verrons que des méthodes s'en rapprochent dangereusement dans certaines situations.

7.2 Analyse des résultats

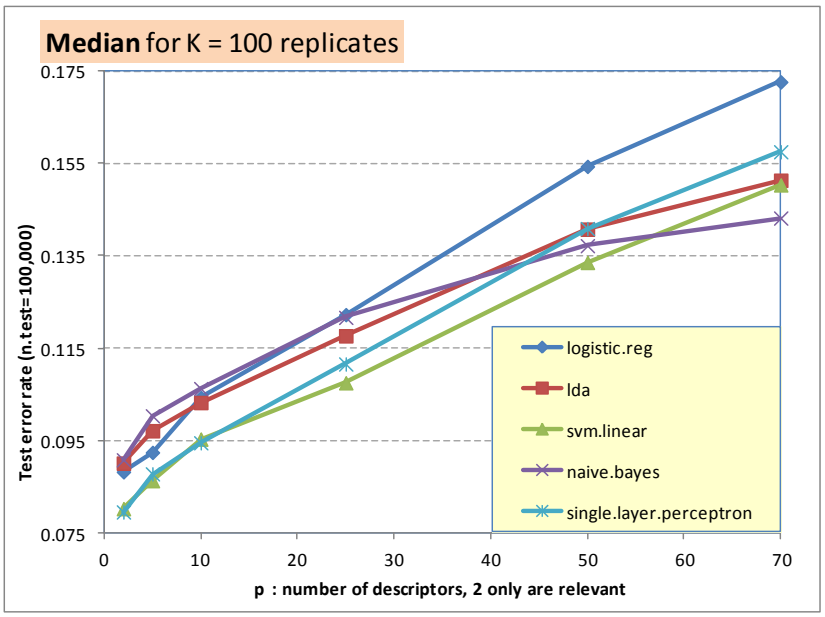


Figure 24 - Taux d'erreur des méthodes selon le nombre de descripteurs

Nous présentons les taux d'erreurs médians des méthodes pour K = 100 essais avec les différentes valeurs de p (Figure 24). Ils amènent plusieurs commentaires :

1. La malédiction de la dimensionnalité n'est pas un mythe. Toutes les méthodes s'effondrent lorsque l'on augmente la proportion de variables non pertinentes n'ayant aucun lien avec la cible.
2. Mais pas de la même manière cependant. Le perceptron et les SVM, les meilleures dans un environnement parfait, se dégradent différemment. Manifestement, les SVM résistent bien à l'adjonction de variables bruitées. Le taux d'erreur du perceptron se rapproche de celui du classifieur par défaut.
3. La bonne surprise est que l'analyse discriminante et le bayésien naïf, auparavant dominés, se révèlent plutôt robustes face à la dimensionnalité (sachant que nous sommes loin de l'erreur optimale quoiqu'il en soit). Mais est-ce vraiment une surprise ? Le biais d'apprentissage qui

paraissait si contraignant devient salutaire lorsqu'on présente des informations totalement erratiques à l'algorithme d'apprentissage.

4. Le bayésien naïf se démarque notamment parce que le nombre de paramètres à estimer est très faible (moyennes et écarts-type conditionnels des variables prédictives simplement). Au point qu'il rattrape et surclasse même les SVM lorsque l'environnement est extrêmement bruité (sur nos données, entendons-nous bien). A ce point là, c'est quand même assez extraordinaire je trouve. Surtout qu'on nous annonce dans de nombreuses publications que cette méthode est très sensible à la dimensionnalité. Pas plus que les autres apparemment.
5. La régression logistique en revanche est totalement en perdition à mesure que la dimension augmente. Au point de ne pas faire mieux que le classifieur par défaut lorsque p = 70 (68 variables non pertinentes).

Tout ceci montre surtout que la sélection de variables est un aspect essentiel de l'apprentissage supervisé, tant pour l'interprétation des modèles que pour leurs qualités prédictives, y compris sur le concept facile à apprendre que nous avons utilisé pour générer les données de ce tutoriel.

8 Conclusion

Notre objectif initial était de montrer et comparer le comportement des classifieurs linéaires les plus populaires. Nous avons détaillé dans un premier temps le fonctionnement des méthodes en décrivant les frontières induites sur un jeu de données artificielles (sections 3 et 4). Il n'y aucun doute, les méthodes linéaires subdivisent l'espace de représentation en régions à l'aide de droites (ou d'hyperplan si nous sommes dans des espaces de plus grandes dimensions). Nous avons vu également que certaines techniques, de par leurs hypothèses sous-jacentes sur les distributions de données, ont du mal à produire les solutions adéquates lorsqu'elles sont placées dans des situations qui les désavantagent. De ce point de vue, ce tutoriel joue pleinement son rôle.

Dans un deuxième temps, pour donner une meilleure assise aux résultats, nous avons monté une expérimentation à grande échelle, en essayant d'analyser les impacts de la taille d'échantillon d'apprentissage et de la dimensionnalité sur la qualité des résultats. Nous nous sommes entourés d'un luxe de précautions pour ne pas biaiser les résultats. (A) Nous avons utilisé des données artificielles pour maîtriser complètement le concept que l'on cherche à apprendre. (B) Nous avons mis en place un dispositif de bruitage des données pour ajouter de la difficulté à la construction des modèles et donner un tour réaliste à l'expérimentation. L'intérêt d'utiliser des données artificielles est que nous contrôlons entièrement le processus d'évaluation. Les caractéristiques des données générées expliquent la nature des résultats obtenus. Nous pouvons ainsi analyser finement les méthodes en disséquant les aspects qui guident leur comportement. Sur données réelles, nous prenons pour argent comptant les taux d'erreur mesurés sans vraiment comprendre les composantes qui les ont influencées.

Enfin, à titre de prospective, une piste d'évaluation non explorée des algorithmes d'apprentissage de notre étude serait de moduler le niveau et la nature du bruit sur la classe pour jauger la robustesse des approches par rapport à cet aspect. Peu de modifications sont à apporter au programme accompagnant ce tutoriel.

9 Références

Bardos M., « Analyse Discriminante – Application au risque et scoring financier », Dunod, 2001 ; chapitre 2, « Analyse Discriminante de Fisher », pp. 29 à 59 ; chapitre 3, « Discrimination Logistique », pp. 61 à 79.

Bishop C., « Pattern Recognition and Machine Learning », Springer, 2006 ; Chapitre 4, « Liner Models for Classification », pp. 179 à 224.

Duda R., Hart P., Stork D., « Pattern Classification », John Wiley and Sons, 2001 ; chapitre 5, « Linear Discriminant Functions », pp. 215 à 281.

Hastie T., Tibshirani R., Friedman J., « Elements of Statistical Learning », 10th printing, Janvier 2013, http://www-stat.stanford.edu/~tibs/ElemStatLearn/ ; chapitre 4, « Linear Methods for Classification », pp. 101 à 137.

Theodoridis S., Koutroumbas K., « Pattern Recognition », Elsevier Inc., 2009 ; chapitre 3, « Linear Classifiers », pp. 91 à 150.