Module ADC : Classification supervisée
Devoir maison – Seconde partie

Simon Besson-Girard
Nicolas Wattiez

vendredi 19 décembre 2014

Avant toutes choses, certaines commandes requièrent l'import de librairies. Pour ce faire, faites les commandes suivantes dans l'invité de commande `R` :

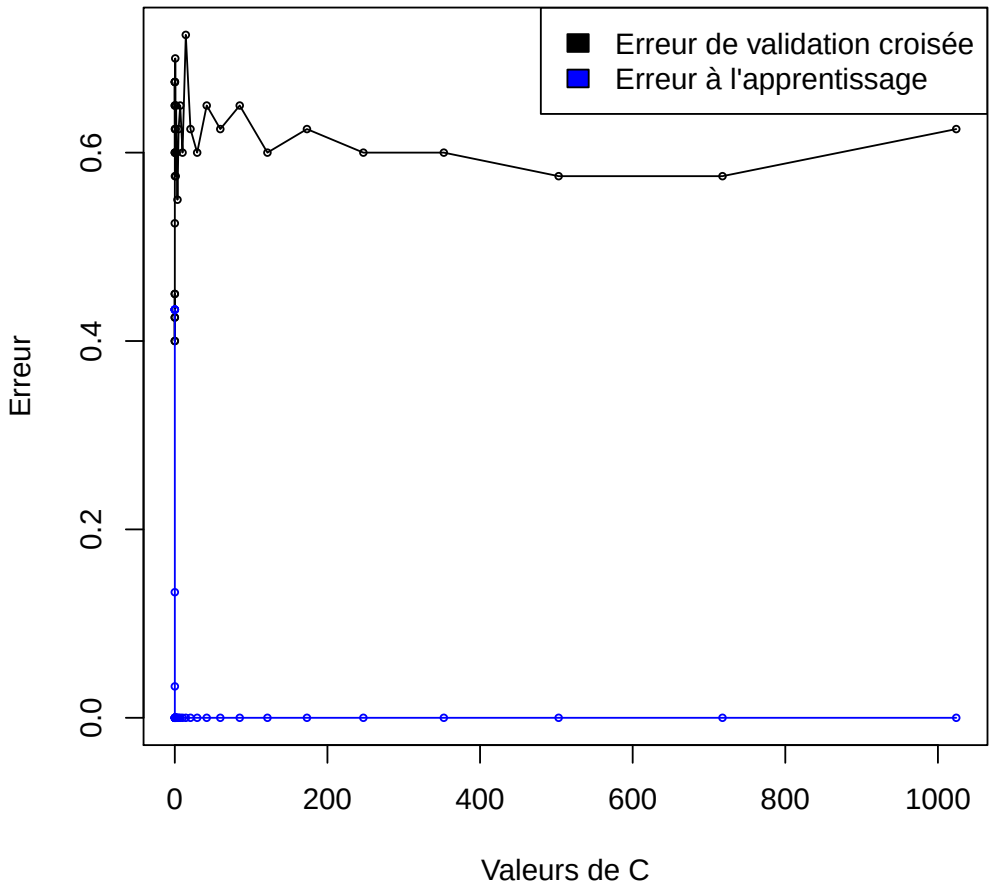
```
> install.packages("kernlab")
> install.packages("ROCR")
> install.packages("plotrix")
> library(MASS)
> library(kernlab)
> library(ROCR)
> library(plotrix)
```

Pour une raison qui nous est inconnue, il est déconseillé de copier-coller tout le script (fichier .R attaché) en une seule fois. Des erreurs de compilation pourraient apparaitre. Il est recommandé d'utiliser la fonction `source("script.R")`.

1 Utilisation des machines à vecteurs supports

5.1. Fonction `SVM.accuracy.wrt.C(d)` qui trace, pour un ensemble d'apprentisage `d`, le taux d'erreur en validation croisée des SVMs sur un jeu de données en fonction du paramètre `C` :

```
> SVM.accuracy.wrt.C <- function(d) {
+         C.values <- sapply(c(seq(-10,10,le=40)), function (x) 2^x)
+         svm.cross <- sapply(C.values,function(x) cross(ksvm(Y~.,data=d,type='C-svc',
+                         kernel='vanilladot',C=x,cross=20)) )
+         svm.error <- sapply(C.values,function(x) error(ksvm(Y~.,data=d,type='C-svc',
+                         kernel='vanilladot',C=x,cross=20)) )
+         plot(C.values,svm.cross,type='o',xlab="Valeurs de C",ylab="Erreur",cex=.5,
+                         ylim=c(min(c(svm.error,svm.cross)),max(c(svm.cross,svm.error))))
+         points(C.values,svm.error,type='o',col='blue',cex=.5)
+         legend("topright",c("Erreur de validation croisée",
+                         "Erreur à l'apprentissage"),fill=c("black","blue"))
+ }
> SVM.accuracy.wrt.C(generateDifficultDatasetAlt(100,30))#
```



Commentaire des résultats :

5.2. Fonction `selectC(d)` qui choisit une valeur de `C` pour un ensemble d'apprentissage `d` :

```
> selectC <- function(d) {
+         C.values <- sapply(c(seq(-10,10,by=1)), function (x) 2^x)
+         svm.cross <- sapply(C.values,function(x) cross(ksvm(Y~.,data=d,type='C-svc',
+                         kernel='vanilladot',C=x,cross=20)) )
+         tab <- as.data.frame(cbind(C.values,svm.cross))
+         return(tab$C.values[tab$svm.cross==min(svm.cross)])
+ }
> resultat.selectC <- selectC(generateDifficultDatasetAlt(100,30))

> resultat.selectC

[1] 0.001953125
```

5.3. Détermination du taux d'erreur moyen de SVMs sur les données générées :

```
> compare.SVM.mH <- function(nbjeux,fonctionquigenere,taillejeu) {
+         f <- function() {
+                 d <- fonctionquigenere(taillejeu)
+                 C.value <- selectC(d)
+                 error.SVM <- mean(sapply(C.values,function(x) cross(ksvm(Y~.,data=d,type='C-svc',
```

```
+                         kernel='vanilladot',C=C.value,cross=5))) )
+                 error.mediatorHyperplane <- error.wrt.n(taillejeu,fonctionquigenere)[1]
+                 return(c(error.SVM,error.mediatorHyperplane))
+         }
+         moyennes <- replicate(nbjeux,f())
+         return(c(mean(moyennes[1,]),(mean(moyennes[2,]))))
+ }
```

Tracé de la courbe `ROC` :

```
>
```

Comparaison des résultats à l'algorithme de l'hyperplan médiateur :

```
>
```

5.4. Représentation graphique du comportement des deux algorithmes sur un ensemble d'apprentissage généré par `generateDifficultDataset` :

```
>
```

Explication intuitive de pourquoi l'un s'en sort mieux que l'autre :

2 Classification de tissus tumoraux basée sur l'expression génique

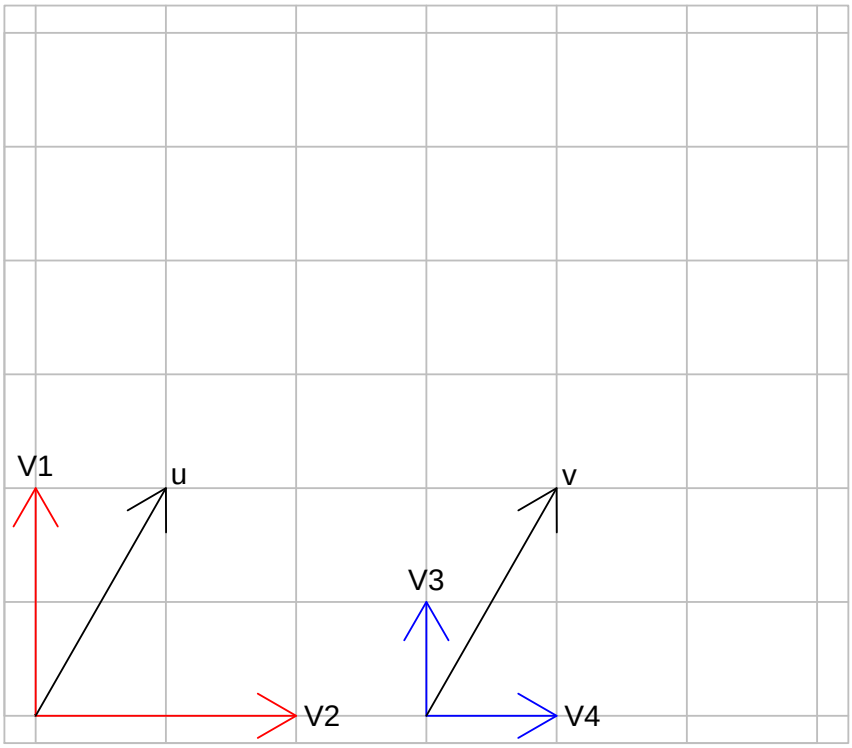
6.1. Fonction `read.prostate.dataset` pour charger les données de `prostate.txt` dans `R` :

```
> read.prostate.dataset <- function(nomfichier) {
+         prostate <- read.table(nomfichier)
+         colnames(prostate)[1] <- "Y"
+         for (i in 1:nrow(prostate)) {if (prostate[i,1]==0) {prostate[i,1] <- -1} }
+         return(prostate)
+ }
```

6.2. Modification de `read.prostate.dataset` pour qu'elle normalise les données en entrée :

```
> read.prostate.dataset <- function(nomfichier) {
+         prostate <- read.table(nomfichier)
+         colnames(prostate)[1] <- "Y"
+         for (i in 1:nrow(prostate)) {if (prostate[i,1]==0) {prostate[i,1] <- -1} }
+         prostate.moyenne <- sapply((1:ncol(prostate)),function(x) mean(prostate[,x]) )
+         prostate.ecart.type <- sapply((1:ncol(prostate)),function(x) sd(prostate[,x]) )
+         for (c in 2:ncol(prostate)) {for (l in 1:nrow(prostate)) {
+                 prostate[l,c] <- (prostate[l,c]-prostate.moyenne[c])/prostate.ecart.type[c]}
+         }
+         return(prostate)
+ }
```

6.3. Montrons ce qui se passe lorsque l'on calcule le produit scalaire de deux vecteurs lorsque les 2 coordonnées ne sont pas à la même échelle :



Déduction de pourquoi il est impératif de normaliser les données avant de travailler avec :
Soit R_1 le référentiel composé par les vecteurs V_1 et V_2, R_2 le référentiel composé par les vecteurs V_3 et V_4, $\vec{u}$ et $\vec{v}$ deux vecteurs.
Si l'on regarde les coordonnées de $\vec{u}$ et $\vec{v}$ exprimée dans R_1, on obtient :

- $\vec{u} = (1, 2)$
- $\vec{v} = (1, 2)$

Le produit scalaire de ces deux vecteurs est alors égale à 5. Si par contre on exprime $\vec{v}$ dans R_2, ce qui donne $\vec{v} = (2, 4)$, et qu'on calcule leur produit scalaire, on obtient 10. On constate donc que le produit de deux vecteurs ne possédant pas la même norme n'est pas égale au moins dans certains cas.

6.4. Application des deux méthodes de classification (hyperplan médiateur et SVMs) sur le jeu de données "prostate" :

```
> as.vector(t(pred.prostate.mediatorHyperplane <- mediatorHyperplane(prostate)$pred(prostate)))
```

```
>
```

Discussion des résultats obtenus :

3 Bonus : algorithme k-NN

7.1. Algorithme des k plus proches voisins :

```
>
```

7.2. Comparaison de ses performances aux SVMS :

>