

SY09 Printemps 2017  
TP 3  
Discrimination, théorie bayésienne de la décision

## 1 Classifieur euclidien, $K$ plus proches voisins

On souhaite étudier les performances du classifieur euclidien et des  $K$  plus proches voisins sur différents jeux de données *binaires* (constitués d'individus de  $\mathbb{R}^p$  répartis dans  $g = 2$  classes).

Vous commencerez par programmer les fonctions puis vous les testerez selon un protocole décrit au paragraphe 1.2, tout d'abord sur des données synthétiques (générées selon une distribution prédéfinie), puis sur des données réelles.

### 1.1 Programmation

#### 1.1.1 Classifieur euclidien

Compléter les fonctions `ceuc.app` et `ceuc.val`.

```
ceuc.app <- function(Xapp, zapp)
{
  ...
}

ceuc.val <- function(mu, Xtst)
{
  ...
}
```

La première fait l'apprentissage des paramètres du classifieur euclidien ; elle a pour arguments d'entrée :

- un tableau individus-variables `Xapp` (de dimensions `napp`  $\times$   $p$ ) correspondant aux individus d'apprentissage,
- le vecteur `zapp` (de longueur `napp`) des étiquettes associées à chacun des individus.

Les arguments de sortie sont les paramètres estimés du classifieur euclidien, sous la forme d'une matrice `mu` de dimensions  $g \times p$ .

La seconde fait le classement d'un tableau individus-variables `Xtst` (de dimensions `ntst`  $\times$   $p$ ) : elle doit donc prendre en argument d'entrée

- la matrice `mu` des paramètres estimés,
- l'ensemble à évaluer `Xtst`,

et retourner un vecteur (de longueur `ntst`) d'étiquettes prédites.

On pourra s'aider de la fonction `distXY` fournie qui calcule les distances (euclidiennes au carré) entre les individus de deux ensembles `X` et `Y`, et de la fonction `which.min` qui détermine l'indice de l'élément minimal d'un vecteur.

#### 1.1.2 $K$ plus proches voisins

Compléter les fonctions `kppv.val` et `kppv.tune`.

```

kppv.val <- function(Xapp, zapp, K, Xtst)
{
  ...
}

kppv.tune <- function(Xapp, zapp, Xval, zval, nppv)
{
  ...
}

```

La première fait le classement d'un tableau individus-variables  $X_{tst}$  : elle prend donc en argument

- les données étiquetées ( $X_{app}$  et  $z_{app}$ ),
- le nombre de voisins  $K$  utilisé pour le classement,
- l'ensemble à évaluer  $X_{tst}$  ;

elle retourne donc un vecteur (de longueur  $ntst$ ) d'étiquettes prédites.

La seconde détermine le nombre « optimal » de voisins  $K_{opt}$  (choisi parmi un vecteur  $nppv$  de valeurs possibles), c'est-à-dire donnant les meilleures performances sur un ensemble de validation étiqueté (matrice  $X_{val}$  de dimensions  $nval \times p$  et vecteur  $z_{val}$  de longueur  $nval$ ). Elle doit donc prendre en argument d'entrée

- le tableau individus-variables  $X_{app}$  et le vecteur  $z_{app}$  des étiquettes associées, à partir desquels le classement est réalisé ;
- le tableau individus-variables  $X_{val}$  et le vecteur  $z_{val}$  à utiliser pour la validation ;
- un ensemble de valeurs  $nppv$  correspondant aux différents nombres de voisins que l'on cherche à tester.

Elle retourne la valeur  $K_{opt}$  choisie dans l'ensemble  $nppv$  et donnant les meilleurs résultats sur  $X_{val}$ .

Il sera évidemment judicieux de faire appel à la fonction `kppv.val` dans la fonction `kppv.tune`. On pourra également utiliser les fonctions `distXY` et `which.max` dans la fonction `kppv.val`.

### 1.1.3 Test des fonctions

Vous pouvez utiliser le jeu de données `Synth1-40` pour vérifier que vos fonctions sont justes. Pour visualiser les frontières de décision obtenues à l'aide des fonctions `ceuc.val` et `kppv.val`, vous pouvez utiliser les fonctions `front.ceuc` et `front.kppv` fournies (voir paragraphe 1.3.1).

Les frontières de décision qu'il faut obtenir en utilisant la totalité des données pour l'apprentissage du modèle sont représentées dans la Figure 1.

```

donn <- read.csv("Synth1-40.csv")
X <- donn[,1:2]
z <- donn[,3]

```

## 1.2 Évaluation des performances

Pour un jeu de données, lorsqu'on souhaite **estimer** le taux d'erreur  $\varepsilon$  d'un classifieur, on utilise généralement la technique suivante. On répète  $N$  fois la procédure qui consiste à :

- séparer aléatoirement l'ensemble des données disponibles, de manière à former un ensemble d'apprentissage et un ensemble de test (et, de manière optionnelle, un ensemble de validation, s'il est nécessaire d'estimer un ou plusieurs hyper-paramètres) ;
- apprendre les paramètres du modèle sur l'ensemble d'apprentissage ainsi formé (après avoir éventuellement optimisé les hyper-paramètres sur l'ensemble de validation, s'il y a lieu), et calculer le taux d'erreur obtenu sur l'ensemble de test.

En déterminant de la sorte  $N$  séparations aléatoires de l'ensemble de données en un ensemble d'apprentissage et un ensemble de test, on peut ainsi **recueillir un échantillon de  $N$  estimations  $E_1, \dots, E_N$  du taux d'erreur** (généralement effectuées sur l'ensemble de test). **On peut alors déterminer une estimation ponctuelle  $\hat{\varepsilon}$  de  $\varepsilon$  (moyenne) et un intervalle de confiance sur  $\varepsilon$ .**

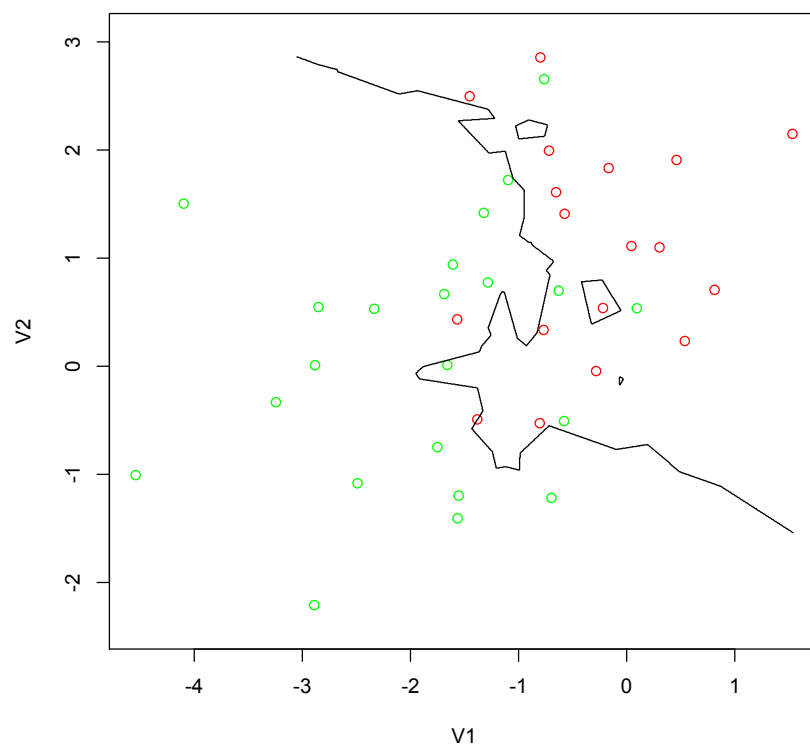
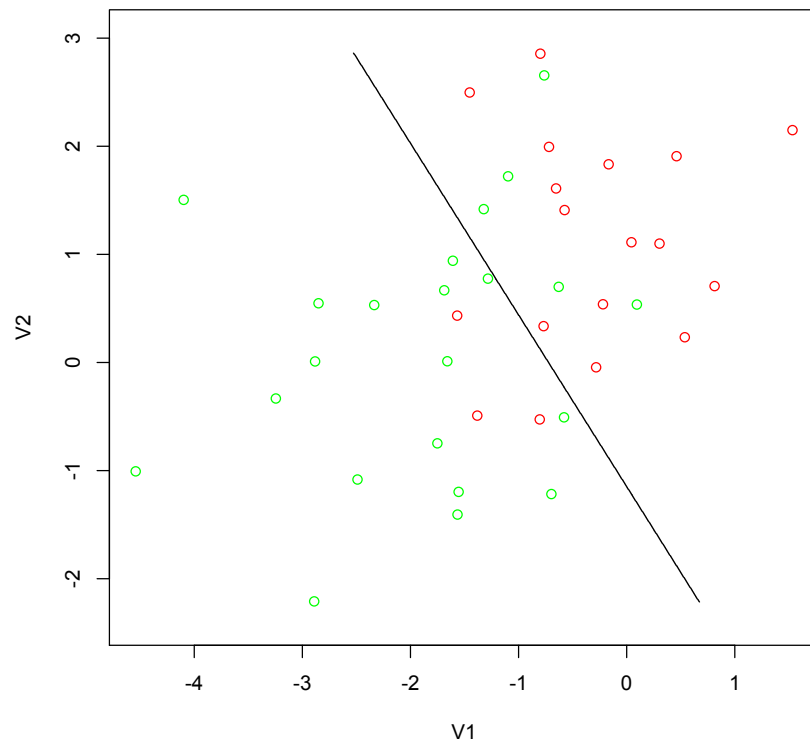


FIGURE 1 – Frontières de décision obtenues en utilisant toutes les données pour l'apprentissage ; classifieur euclidien (haut) et 3-plus proches voisins (bas).

### 1.2.1 Jeux de données Synth1-40, Synth1-100, Synth1-500 et Synth1-1000

On dispose de cinq jeux de données (téléchargeables sur le site de l'UV) : Synth1-40, Synth1-100, Synth1-500, Synth1-1000, et Synth2-1000. Pour chacun de ces jeux de données, chaque classe a été générée suivant une loi normale bivariée. Les distributions sont les mêmes pour les jeux de données Synth1-40, Synth1-100, Synth1-500 et Synth1-1000, qui ne diffèrent que par le nombre d'exemples disponibles. En revanche, la distribution des données dans Synth2 est différente.

1. Pour chacun des jeux de données, estimer les paramètres  $\mu_k$  et  $\Sigma_k$  des distributions conditionnelles, ainsi que les proportions  $\pi_k$  des classes.
2. Écrire un script qui effectue  $N = 20$  séparations aléatoires de chaque jeu de données en ensembles d'apprentissage et de test, et qui calcule (et stocke) pour chacune le taux d'erreur d'apprentissage et le taux d'erreur de test. On pourra utiliser la fonction `separ1` pour séparer les données (voir paragraphe 1.3.2).

ADL ?

En considérant ensuite l'ensemble des résultats obtenus lors des  $N = 20$  expériences, donner l'estimation ponctuelle du taux d'erreur  $\varepsilon$  ainsi qu'un intervalle de confiance, d'abord à partir des estimations faites sur l'ensemble d'apprentissage, puis celles faites sur l'ensemble de test. Qu'observez-vous ?

3. Effectuer une séparation aléatoire de l'ensemble de données en un ensemble d'apprentissage et un ensemble de test. Déterminer le nombre optimal de voisins à l'aide de la fonction `kppv.tune`, en utilisant l'ensemble d'apprentissage comme ensemble de validation. Quel est le nombre optimal de voisins déterminé ? Pourquoi ?
4. Comme pour le classifieur euclidien, écrire un script qui effectue  $N = 20$  séparations aléatoires de chaque jeu de données en ensembles d'apprentissage, de validation, et de test ; et qui pour chacune détermine le nombre optimal de voisins à partir d'un ensemble de validation spécifique, puis calcule (et stocke) le taux d'erreur sur l'ensemble d'apprentissage et sur l'ensemble de test. On pourra utiliser la fonction `separ2` pour séparer les données (voir paragraphe 1.3.2).

Comme précédemment, déterminer les estimations ponctuelles du taux d'erreur et les intervalles de confiance obtenus à partir des erreurs d'apprentissage, puis des erreurs de test. Comparer avec les résultats obtenus pour le classifieur euclidien, et commenter.

### 1.2.2 Jeu de données Synth2-1000

On considère maintenant le jeu de données Synth2-1000.

1. Estimer les paramètres  $\mu_k$  et  $\Sigma_k$  ainsi que les proportions  $\pi_k$  des classes.
2. Comme précédemment, calculer les estimations (ponctuelles et intervalles de confiance) de  $\varepsilon$  sur l'ensemble d'apprentissage et sur l'ensemble de test. Commenter et interpréter.

### 1.2.3 Jeux de données réelles

On considère maintenant les jeux de données Pima et Breastcancer. Calculer les estimations (ponctuelles et intervalles de confiance) de  $\varepsilon$  sur l'ensemble d'apprentissage et sur l'ensemble de test. Commenter et interpréter.

## 1.3 Note sur les fonctions mises à disposition

### 1.3.1 Frontières de décision

Les fonctions `front.ceuc` et `front.kppv` échantillonnent l'espace des caractéristiques en déterminant une grille de points : on détermine les décisions prises pour les points de cette grille, puis on trace les frontières de décision en utilisant la fonction `contour`. On peut les appeler comme suit.

```

mu <- ceuc.app(Xapp, zapp)
front.ceuc(mu, Xapp, zapp, 1000)

Kopt <- kppv.tune(Xapp, zapp, Xval, zval,
  2*(1:6)-1)
front.kppv(Xapp, zapp, Kopt, 1000)

```

### 1.3.2 Séparation des données

La fonction `separ1` détermine un ensemble d'apprentissage (de taille  $n_{app} = 2n/3$ ) et un ensemble de test (de taille  $n_{tst} = n/3$ ). La fonction `separ2` détermine des ensembles d'apprentissage (de taille  $n_{app} = n/2$ ), de validation (de taille  $n_{val} = n/4$ ) et de test (de taille  $n_{tst} = n/4$ ).

```

donn.sep <- separ1(X, z)
Xapp <- donn.sep$Xapp
zapp <- donn.sep$zapp
Xtst <- donn.sep$Xtst
ztst <- donn.sep$ztst

donn.sep <- separ2(X, z)
Xapp <- donn.sep$Xapp
zapp <- donn.sep$zapp
Xval <- donn.sep$Xval
zval <- donn.sep$zval
Xtst <- donn.sep$Xtst
ztst <- donn.sep$ztst

```

## 2 Règle de Bayes

En réalité, les jeux de données étudiés précédemment ont été obtenus de la manière suivante :

1. tout d'abord, l'effectif  $n_1$  de la classe  $\omega_1$  a été déterminé par tirage aléatoire suivant une loi binomiale de paramètres  $n$  et  $\pi_1 = 0.5$  ;
2.  $n_1$  individus ont ensuite été générés dans la classe  $\omega_1$  suivant une loi normale bivariée de paramètres  $\mu_1$  et  $\Sigma_1$ , et  $n_2 = n - n_1$  individus ont été générés dans la classe  $\omega_2$  suivant une loi normale bivariée de paramètres  $\mu_2$  et  $\Sigma_2$ .

Pour les jeux de données `Synth1-40`, `Synth1-100`, `Synth1-500` et `Synth1-1000`, on a utilisé comme paramètres  $n = 40$ ,  $n = 100$ ,  $n = 500$ , et  $n = 1000$ , respectivement ; et

$$\mu_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad \mu_2 = \begin{pmatrix} -2 \\ 0 \end{pmatrix}, \quad \Sigma_1 = \Sigma_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix};$$

Pour le jeu de données `Synth2-1000`, on a utilisé  $n = 1000$ , et

$$\mu_1 = \begin{pmatrix} 3 \\ 0 \end{pmatrix}, \quad \mu_2 = \begin{pmatrix} -2 \\ 0 \end{pmatrix}, \quad \Sigma_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \Sigma_2 = \begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix}.$$

On notera  $\mathbf{x} = (x_1, x_2)^T$ ,  $\mu_k = (\mu_{k1}, \mu_{k2})^T$  et  $\Sigma_k = \text{diag}(\sigma_{k1}^2, \sigma_{k2}^2)$  pour  $k = 1, 2$ .

Pour chacun des jeux de données :

1. quelles sont les distributions marginales des variables  $X^1$  et  $X^2$  dans chaque classe ?
2. Calculer l'expression des courbes d'iso-densité dans chacune des classes, en fonction de  $\mu_1$ ,  $\mu_2$ ,  $\sigma_1$  et  $\sigma_2$ . À quoi correspondent ces courbes ?
3. Montrer que pour les jeux de données `Synth1-n`, la règle de Bayes est une fonction linéaire de  $x_1$  et  $x_2$  et d'une constante  $k$ , que l'on précisera. Représenter la frontière correspondante dans le plan formé par ces variables (on représentera également les données `Synth1-1000`).
4. Calculer l'expression de la règle de Bayes pour le jeu de données `Synth2-1000` en fonction de  $x_1$  et  $x_2$ , et montrer qu'elle s'écrit en fonction de deux constantes  $k_1$  et  $k_2$  que l'on précisera. Représenter la frontière correspondante dans le plan formé par les variables  $x_1$  et  $x_2$  (on représentera également les données `Synth2-1000`).
5. Pour les jeux de données `Synth1-n` et `Synth2-1000`, calculer l'expression formelle de l'erreur de Bayes en fonction de  $k$ , ou  $k_1$  et  $k_2$  ; calculer sa valeur effective, comparer aux résultats de l'exercice précédent et expliquer.