SY09 - TP03
Discrimination, théorie bayésienne de la décision

Nicolas Szewe - Marie Chatelin

May 28, 2015

1 Classifieur euclidien, K plus proches voisins

1.1 Programmation

Vous trouverez en annexe de ce rapport, les fonctions ceuc.app,ceuc.val, kppv.app et kppv.val. Ces fonctions pourraient être amélioré en ... Cependant nous sommes satisfait de leurs comportements car ...

Veuillez trouver ci-dessous, les graphiques correspondant aux résultats de ceuc.val et kppv.val (pour K=1, K=3 et K=15).

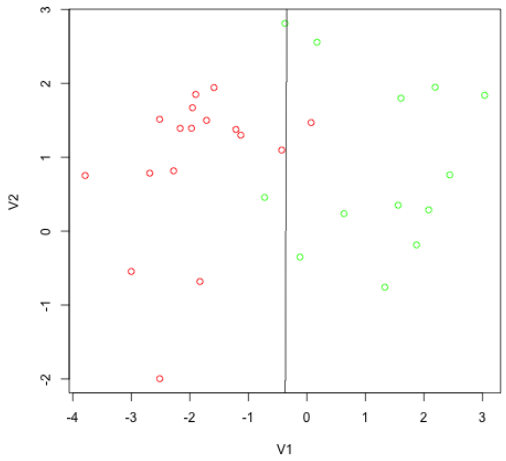
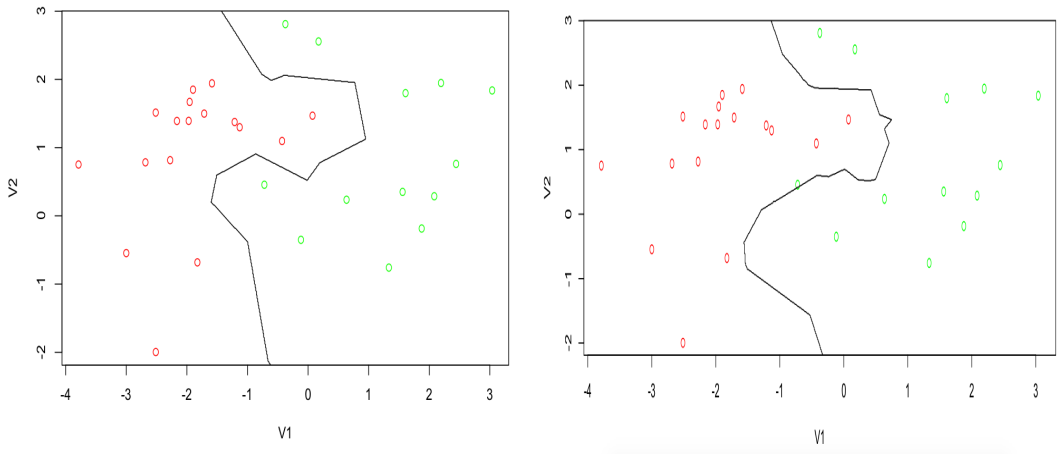


Table 1 : Repartition de la population selon le classificateur Euclidien



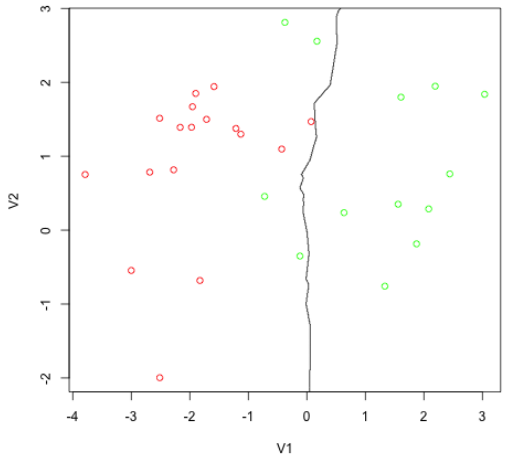


Table 2 : Repartition de la population en fonction des 1,3 et 15 plus proches voisins

1.2 Evaluation des performances

1.2.1 Calcul de paramètres pour les jeux de données Synth1

Pour chacun des jeux de données, on calcule les paramètres des distributions conditionnelles ainsi que les proportions des classes.

$$\begin{pmatrix} & Synth1-40 & Synth1-100 \\ mu & \begin{pmatrix} k1 & -1.9046815 & 0.6988410 \\ k2 & 0.8808594 & 0.8587835 \end{pmatrix} & \begin{pmatrix} k1 & -1.8082330 & 1.057774 \\ k2 & 0.9831412 & 1.161077 \end{pmatrix} \\ somme & \begin{pmatrix} k1 & 1.765911 & 9.550731 \\ k2 & 10.275681 & 2.490861 \end{pmatrix} & \begin{pmatrix} k1 & 2.420586 & 10.223027 \\ k2 & 9.332274 & 1.529833 \end{pmatrix} \\ proportion d'element de famille 1(\%) & 55 & 53 \end{pmatrix}$$

$$\begin{pmatrix} & Synth1-500 & Synth1-1000 \\ mu & \begin{pmatrix} k1 & -1.9101695 & 0.9359349 \\ k2 & 0.9979001 & 0.9843533 \end{pmatrix} & \begin{pmatrix} k1 & -1.998992 & 1.0081863 \\ k2 & 1.090890 & 0.9837324 \end{pmatrix} \\ somme & \begin{pmatrix} k1 & 2.029289 & 10.488501 \\ k2 & 10.325960 & 1.866747 \end{pmatrix} & \begin{pmatrix} k1 & 2.006651 & 11.554622 \\ k2 & 11.568773 & 2.020802 \end{pmatrix} \\ proportion d'element de famille 1(\%) & 48 & 48,8 \end{pmatrix}$$

1.2.2 Calcul de l'intervalle de confiance pour Synth1

On a :

$$E = \frac{1}{m}\sum 1_{z_i \neq z_i},\ T_i = 1_{z_i \neq z_i}$$

Cette fonction suit une loi de Bernouilli de paramètre ε car

$$P(T_i = 1) = E(E) = E(\frac{1}{m}\sum_i^m T_i) = \frac{1}{m}E(\sum_i^m T_i) = m\sum_i^m E(T_1) = \frac{m\varepsilon}{m} = \varepsilon.$$

mE suit donc une loi binomiale. On peut donc approchée cette loi par une loi normale pour des valeurs de m grandes.

$$mE \sim \mathcal{N}(m\varepsilon, m\varepsilon(1-\varepsilon)).$$
$$E \sim \mathcal{N}(\varepsilon, m^{-1}\varepsilon(1-\varepsilon))$$

Comme la variance est inconnue, on a :

$$\frac{\bar{E}-\mu}{S^*/\sqrt{N}} \sim \tau_{N-1}$$

Les taux d'erreur E_j étant indépendants, nous allons déduire un intervalle de confiance pour l'espérance de E a partir de cette fonction.

Soit μ tel que :

$$P(\frac{\bar{E}-\mu}{S^*/\sqrt{N}}) < 1-\alpha$$

Après simplification :

$$IC = [\bar{E} - t_{N-1;1-\alpha/2}\frac{S^*}{\sqrt{N}}, \bar{E} + t_{N-1;1-\alpha/2}\frac{S^*}{\sqrt{N}}]$$

1.2.3 Taux d'erreur du classifieur Euclidien pour N = 20

Nous réalisons le protocole décrit dans le sujet afin d'estimer les taux d'erreurs de nos scripts. Après avoir obtenu les erreurs pour les 20 expériences, nous réalisons une moyenne sur ces erreurs. Les moyennes ainsi obtenues sont :

$$\begin{pmatrix} & Synth1-40 & Synth1-100 & Synth1-500 & Synth1-1000 \\ epsilon & 0.1269231 & 0.1029412 & 0.07035928 & 0.06691617 \end{pmatrix}$$

On remarque que plus l'échantillon est grand plus le taux d'erreur est faible. Vous trouverez en annexe une fonction calcul_E, responsable de ce calcul d'erreur.

Nous obtenons de plus les intervalles de confiance avec $\alpha = 0.95$:

Synth1-40

$$IR_test = [(1-0.1269231) - 2,093 * \frac{0,0022}{\sqrt{20}}, (1-0.1269231) + 2,093 * \frac{0,0022}{\sqrt{20}}] = [0.8720473, 0.8741065]$$

Synth1-100

$$IR_test = [(1-0.1029412) - 2,093 * \frac{0,0022}{\sqrt{20}}, (1-0.1029412) + 2,093 * \frac{0,0022}{\sqrt{20}}] = [0.8960292, 0.8980884]$$

Synth1-500

$$IR_test = [(1-0.07035928) - 2,093 * \frac{0,0022}{\sqrt{20}}, (1-0.07035928) + 2,093 * \frac{0,0022}{\sqrt{20}}] = [0.9286111, 0.9306703]$$

Synth1-1000

$$IR_test = [(1-0.06691617) - 2,093 * \frac{0,0022}{\sqrt{20}}, (1-0.06691617) + 2,093 * \frac{0,0022}{\sqrt{20}}] = [0.9320542, 0.9341134]$$

1.2.4 Calcul du K optimal

Nous recherchons le K optimal parmis des intervalles de K variables en fonction de la taille de l'échantillon. Nous testons avec des K de la manière suivante :

Synth1-40 : nppv = c(1,3,7,13,17)
Synth1-100 nppv = c(1,3,13,25,37)
Synth1-500 nppv = c(1,3,75,125,225)
Synth1-1000 nppv = c(1,3,75,225,401)

Le nombre de voisins optimal est dans tous les cas un voisins. En effet, lorsque l'on choisit un seul voisin, le résultat est meilleur puisque l'on utilise les résultats du point lui-même.

1.2.5 Taux d'erreur du classifieur des K-voisins pour N = 20

Nous réalisons le protocole décrit dans le sujet afin d'estimer les taux d'erreurs de nos scripts. Après avoir obtenu les erreurs pour les 20 expériences, nous réalisons une moyenne sur ces erreurs. Les moyennes ainsi obtenues sont :

$$\begin{pmatrix} & Synth1-40 & Synth1-100 & Synth1-500 & Synth1-1000 \\ epsilon & 0.125 & 0.11 & 0.0824 & 0.0706 \end{pmatrix}$$

On remarque que plus l'échantillon est grand plus le taux d'erreur est faible. On remarque que le taux d'erreur de la fonction des K-voisins est supérieur à celui du classificateur Euclidien. Ce résultat est surprenant de part la complexité plus importante de l'algorithme des K-voisins.

Nous obtenons de plus les intervalles de confiance du taux d'erreur avec $\alpha = 0.95$:

Synth1-40

$$IR_test = [(1 - 0.125) - 2,093 * \frac{0,0022}{\sqrt{20}}, (1 - 0.125) + 2,093 * \frac{0,0022}{\sqrt{20}}] = [0.8739704, 0.8760296]$$

Synth1-100

$$IR_test = [(1 - 0.11) - 2,093 * \frac{0,0022}{\sqrt{20}}, (1 - 0.11) + 2,093 * \frac{0,0022}{\sqrt{20}}] = [0.8889704, 0.8910296]$$

Synth1-500

$$IR_test = [(1 - 0.0824) - 2,093 * \frac{0,0022}{\sqrt{20}}, (1 - 0.0824) + 2,093 * \frac{0,0022}{\sqrt{20}}] = [0.9165704, 0.9186296]$$

Synth1-1000

$$IR_test = [(1 - 0.0706) - 2,093 * \frac{0,0022}{\sqrt{20}}, (1 - 0.0706) + 2,093 * \frac{0,0022}{\sqrt{20}}] = [0.9283704, 0.9304296]$$

1.2.6 Calcul de paramètres pour les jeux de données Synth2

On calcule les paramètres des distributions conditionnelles ainsi que les proportions des classes pour le jeu de données Synth2.

$$\begin{pmatrix} & Synth2-1000 \\ mu & \begin{pmatrix} k1 & -4.055942 & 1.011496 \\ k2 & 4.028957 & 1.067821 \end{pmatrix} \\ somme & \begin{pmatrix} k1 & 1.014560 & 0.018451 \\ k2 & 0.018451 & 0.939754 \end{pmatrix} \\ proportiond'elementdefamille1(\%) & 48.8 \end{pmatrix}$$

1.2.7 Calcul de l'intervalle de confiance pour Synth2

Nous obtenons pour ce jeu de données des erreurs et intervalle de confiance sur la valeur d'espsilon répartit comme ci-dessous :

Classifieur Euclidien :

$$\text{Epsilon} = 0.006587 \; IR_test = [0.006585, 0.006588]$$

Classifieur des K voisins :

$$\text{Epsilon} = 0.0066 \; IR_test = [0.006597, 0.006603]$$

2 Règle de Bayes

Pour déterminer la distribution marginale de X1 et X2 dans chaque classe, on se base sur les vecteurs définissant les paramètres de nos distributions. Pour nos 2 classes les matrices de covariance conditionnelles Σ_1 et Σ_2 sont diagonales, donc les variables X_1 et X_2 sont indépendantes. Ainsi, la densité de probabilité conditionnelle s'exprime $f(x|w_k) = f(X_k1|w_k)f(X_k2|w_k), k \in 1, 2$ pour chaque classe.
Les densités conditionnelles du vecteur $\mathbf{X} = (X_1, X_2)^T$ sont donc
f_1(x) = $f_{11}(X_{11})f_{12}(X_{12})$ dans la classe ω_1 et f_2(x) = $f_{21}(X_{21})f_{22}(X_{22})$ dans la classe ω_2.
De plus, les individus des deux classes, ω_1 et ω_2 suivent une loi normale bivariée. Or tout sous-vecteur d'un vecteur aléatoire gaussien suit aussi une loi gaussienne. Les composantes de X, X_1 et X_2 suivent donc aussi une loi gaussienne.

Finalement, pour les 4 premiers jeux de données :
$X_{11} \sim \mathcal{N}(\mu_{11}, 1)$ et $X_{12} \sim \mathcal{N}(\mu_{12}, 1)$
$X_{21} \sim \mathcal{N}(\mu_{21}, 1)$ et $X_{22} \sim \mathcal{N}(\mu_{22}, 1)$

Pour le dernier jeu de données Synth2-1000 :
$X_{11} \sim \mathcal{N}(\mu_{11}, 1)$ et $X_{12} \sim \mathcal{N}(\mu_{12}, 1)$
$X_{21} \sim \mathcal{N}(\mu_{21}, 5)$ et $X_{22} \sim \mathcal{N}(\mu_{22}, 5)$

2.1 Montrer que dans chaque classe, les courbes d'iso-densité sont des cercles dont on précisera les centres et les rayons.

Pour les 4 premiers jeux de données ainsi que la classe ω_1 du jeu de données Synth2-1000 : Σ_1 et Σ_2 sont des matrices identité. Les courbes iso-densité ont donc pour équation $(x - \mu_1)^t(x - \mu_1) = c_1$ et $(x - \mu_2)^t(x - \mu_2) = c_2$, où c_1, c_2 sont des constantes.

Il s'agit d'équations de cercles dont les centres sont μ_1 et μ_2 respectivement, et de rayons $\sqrt{c_1}$ et $\sqrt{c_2}$ respectivement. Dans le cas où on a une matrice de variance avec des valeurs sur la diagonale égales à 5 on obtient aussi un cercle à un facteur près.

2.2 Déterminer l'expression de la règle de Bayes pour le problème de discrimination des classes 1 et 2.

On définit alors la règle de Bayes permettant de choisir à quelle classe affecter une observation quelconque. On appelle action $a1$ le fait d'affecter l'observation à la classe ω_1 et c_{kl} le coût associé au choix de l'action a_l lorsque la vraie classe est ω_k.

On définit la matrice des coûts ainsi : Les coûts sont définis ainsi :

	ω_1	ω_2
a_1	c_{11}	c_{12}
a_2	c_{21}	c_{22}

La règle s'écrit alors ainsi :

Avec π_1 et π_2 les probabilité à priori des deux classes.

2.3 Pour les quatre premiers jeux de données, tracer avec R les frontières de décision dans le plan formé par les variables X1 et X2.

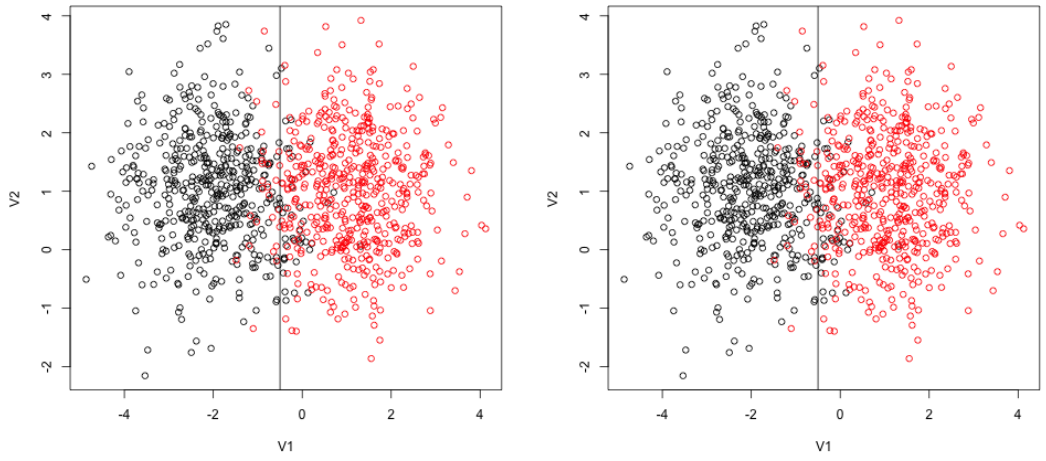
Pour obtenir la classification optimale, on choisit ces valeurs dans la matrice de coût :

	ω_1	ω_2
a_1	0	1
a_2	1	0

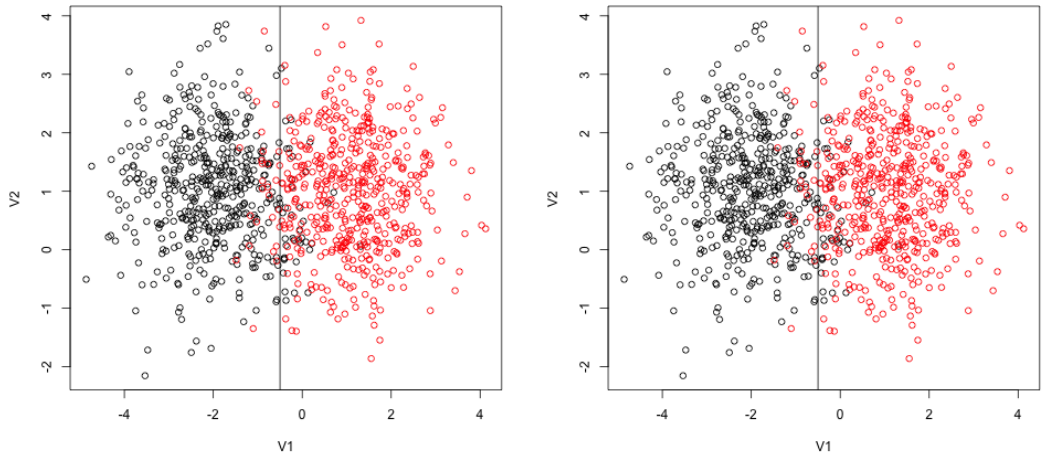
On peut choisir des valeurs de coûts différents pour diminuer le risque d'erreur de prédiction pour la classification vers ω_1 ou ω_2. Comme on a $\pi_1 = \pi_2 = \frac{1}{2}$, nous pouvons alors simplifier l'écriture de la règle de Bayes et obtenir une règle de décision linéaire. On a une frontière qui correspond à un hyperplan passant par le milieu du segment entre μ_1 et $\mu 2$. On connaît sa direction en inversant la matrice de variance, le vecteur normal est donc :

L'hyperplan passe par le milieu du segment d'extrémités μ_1 et μ_2.

Avec R, nous obtenons alors les frontières de décision suivantes :



Frontière de décision pour Synth1_40 et Synth1_100



Frontière de décision pour Synth1_500 et Synth1_1000

Conclusion

Durant ce TP, nous avons eu la possibilité d'utiliser deux méthodes de classification supervisée :
- le classifieur euclidien, l'un des plus simples pour se baser un ensemble d'apprentissage pour déterminer le centre de classes. Cette simplicité vient de paire avec une limitation du coté de la gestion des distributions très distribuées dans le plan.
- la règle de Bayes qui permet de contrôler les risques ce qui ajoute une gestion des erreurs plus poussée, que nous n'avons cependant pas réussi à mettre en place jusqu'au bout.

3 Annexe

```
ceuc.app <- function(Xapp,zapp){
napp <- dim(Xapp)[1]
nbDimension <- dim(Xapp)[2]
x<- NULL
n <- NULL
x <- rbind(rep(0, nbFamille),rep(0,nbDimension))
for(z in 1:nbFamille){
        for(i in 1:napp){
                if(zapp[i]==z){
                        n[z] <- length(zapp[zapp==z])
                        for(j in 1:nbDimension){
                                x[z,j] <- x[z,j] + Xapp[i,j]
                        }
                }
                }
                for(j in 1:nbDimension){
                                x[z,j] <- x[z,j] * (1/n[z])
                }
        }
        return(x)
}
}

ceuc.val <- function(mu,Xtst){
napp <- dim(Xtst)[1]
min <- rbind(rep(99999, napp))
nbDimension <- dim(Xtst)[2]
etiquette <- rbind(rep(0,napp))
distance <- rbind(rep(0, nbFamille),rep(0,napp))
for(i in 1:napp){
        for(z in 1:nbFamille){
                distance[z,i] <- distanceEuclidienne(Xtst[i,1],mu[z,1],Xtst[i,2],mu[z,2])
                if(distance[z,i]< min[i]){
```

```
                        min[i] <- distance[z,i]
                        etiquette[i]<-z
                }
        }
}
        return(etiquette)
}
}


distanceEuclidienne <- function(x1,xt,y1,yt){
        return(sqrt((x1-xt)^2 + (y1-yt)^2))
}

getVoisins <- function(Xapp,zapp,K,Xtst){
        napp <- dim(Xapp)[1]
        ntst <- dim(Xtst)[1]
        distance <- matrix(0, ntst, napp)
        voisins <- matrix(0,ntst,K)
        for(i in 1:ntst){
                min <- matrix(99999,K,2)
                for(j in 1:napp){
                distance[i,j] <- distanceEuclidienne(Xtst[i,1],Xapp[j,1],Xtst[i,2],Xapp[j,2])
                if(j <= K ){
                        min[j,1] <- distance[i,j]
                        min[j,2] <- j
                        if(K!=1){
                                min <- min[order(min[,1]),]
                        }
                }
                else if(distance[i,j]< min[K,1]){
                        min[K,1] <- distance[i,j]
                        min[K,2] <- j
                        if(K!=1){
                                min <- min[order(min[,1]),]
                        }
                }
                }
                voisins[i,] <- min[,2]
        }
        return(voisins)
}
}

kppv.app <- function(Xapp,zapp,Xval,zval,nppv){
erreur <- 1
for(i in nppv){
    famille_K <- kppv.val(Xapp, zapp, i, Xval)
    erreur_K <- sum((famille_K == zval)==TRUE)/length(zval)
    if(erreur > erreur_K){
      erreur <- erreur_K
      K <- i
    }
  }
  return(K)
}
}

kppv.val <- function(Xapp,zapp,K,Xtst){
        ntst <- dim(Xtst)[1]
        etiquette <- rbind(rep(0,ntst))
        voisins <- getVoisins(Xapp,zapp,K,Xtst)
        for(i in 1:ntst){
                count1 <- 0
                count2 <- 0
```

```
                for(k in 1:K){
                        if(zapp[voisins[i,k]]==1){
                                count1 <- count1 + 1
                        }
                        else{
                                count2 <- count2 + 1
                        }
                }
                if(count1 > count2){
                        etiquette[i]<-1
                }
                else{
                        etiquette[i]<-2
                }
        }
        return(etiquette)
}
}

calcul_E <- function(zappDonnees,zappCalcule) {
        m = length(zappDonnees)
        count1 <- 0
        for(i in 1:m){
                        if(zappDonnees[i]!= zappCalcule[i]){
                                count1 <- count1 + 1
                                }
                        }
                        return(count1/m)
}
```