utc
SORBONNE UNIVERSITES

28 mai 2017

TP 3 - Discrimination, théorie bayésienne de la décision

UV : **SY09**
Branche : **Génie Informatique**
Filière : **Fouille de Données et Décisionnel**
Auteurs : **LU Han - HAMONNAIS Raphaël**

Table des matières

1. Classifieur euclidien, K plus proches voisins

1.1 Programmation

1.1.1 Classifieur euclidien

Nous sommes ici dans le cas de la règle de Bayes de coûts $\{0, 1\}$: classer un individu x dans une classe reviendra à affecter l'individu à la classe dont le centre est le plus proche de x, au sens de la distance euclidienne.

Fonction `ceuc.app()` : apprentissage des paramètres

- Objectif :
  - Apprendre les paramètres du classifieur Euclidien.
- Paramètres en entrée :
  - `Xapp` : les données d'apprentissage, de taille $n \times p$;
  - `zapp` : les étiquettes des données d'apprentissage, de taille n ;
- Algorithme :
  - Séparer les ensembles d'individus en fonction de leur classe ;
  - Pour chaque sous-ensemble, calculer la moyenne des observations pour chaque variable aléatoire :
    - Le vecteur ainsi trouvé, à 1 ligne et p colonnes, représente le centre de gravité pour une classe donnée.
- Sortie :
  - Une matrice de taille $g \times p$ des centres de gravité des g classes présentes dans les données.

```r
ceuc.app <- function(Xapp, zapp) {
  zapp = factor(zapp)
  g = length(levels(zapp)) # nombre de classes
  p = dim(Xapp)[2] # p = nombre de variables
  mu = matrix(nrow = g, ncol = p)
  rownames(mu) = levels(zapp)
  colnames(mu) = colnames(Xapp)
  for (k in 1:g) {
    dataClassK = Xapp[zapp == levels(zapp)[k],]
    meanForClassK = apply(dataClassK, MARGIN = 2, mean)
    mu[k,] = meanForClassK
  }
  return(mu)
}
```

FIGURE 1.1 – Code de la fonction `ceuc.app()`

Fonction `ceuc.val()` : discrimination d'un ensemble de données

- Objectif :
  - Classer un ensemble de données grâce au classifieur Euclidien.
- Paramètres en entrée :
  - `mu` : les paramètres du classifieur Euclidien, de taille $g \times p$;
  - `Xtst` : les données de test à étiqueter ;

» Algorithme :

— Pour chaque individu x des données de test `Xtst` :

— Calculer les distances de x aux centres de gravité de chaque classe ;

— Affecter x à la classe dont le centre de gravité est le plus proche.

» Sortie :

— Un vecteur d'étiquettes prédites pour les données de test `Xtst`.

```r
ceuc.val <- function(mu, Xtst) {
  g = dim(mu)[1] # nb classes = nb lignes paramètres mu
  p = dim(Xtst)[2] # nb variables = nb colonnes
  n = dim(Xtst)[1] # nb individus de test
  predictedClasses = vector(length = n)
  classes = factor(rownames(mu))
  distances = distXY(Xtst, mu)
  # récupérer les classes dont on est le plus proche du centre de gravité
  intClasses = apply(distances, MARGIN = 1, which.min)
  # obtenir le nom réel des classes
  predictedClasses = levels(classes)[intClasses]
  return(factor(predictedClasses))
}
```

FIGURE 1.2 – Code de la fonction `ceuc.val()`

1.1.2 K plus proches voisins

Fonction `kppv.val()` : discrimination d'un ensemble de données

» Objectif :

— Classer un ensemble de données grâce au classifieur des K plus proches voisins.

» Paramètres en entrée :

— `Xapp` : les données d'apprentissage ;

— `zapp` : les étiquettes des données d'apprentissage ;

— `K` : le nombre de voisins ;

— `Xtst` : les données de test à étiqueter.

» Algorithme :

— Calculer les distances euclidiennes entre les individus `Xapp` et `Xtst` ;

— Pour chaque individu x de `Xtst` :

— Sélectionner les K plus proches voisins de x ;

— Déterminer la classe g la plus représentée parmi les K plus proches voisins ;

— Affecter cette classe à x.

» Sortie :

— Un vecteur d'étiquettes prédites pour les données de test `Xtst`.

```r
kppv.val = function(Xapp, zapp, K, Xtst) {
    zapp = factor(zapp)
    testSize = nrow(Xtst)
    ztst = vector(length = testSize)

    # calculer les distances
    distances = distXY(Xapp, Xtst) # les distances de Xtst à Xapp sont en colonnes
    orderedDistancesIndexes = apply(distances, 2, order) # contient les indexs des distances triées dans l'ordre croissants

    # trouver les K plus proches voisins pour chaque individu de la population de test
    for (i in 1:testSize) {
        neighborsIndexes = orderedDistancesIndexes[1:K,i] # récupérer les K premiers index, c'est à dire les K plus proches voisins
        #cat(neighborsIndexes, "   ")
        neighborsClasses = zapp[neighborsIndexes] # récupérer les classes des K plus proches voisins
        #cat(neighborsClasses, "   ")
        neighborsClassesContingency = table(neighborsClasses) # compter le nombre de représentants par classe
        #cat(neighborsClassesContingency, "   ")
        mostRepresentedClass = names(which.max(neighborsClassesContingency)) # récupérer le nom de la classe la plus représenté
        #cat(mostRepresentedClass, "   ")
        ztst[i] = mostRepresentedClass
    }
    return(factor(ztst))
}
```

FIGURE 1.3 – Code de la fonction `kppv.val()`

Fonction `kppv.tune()` : optimisation du nombre de voisins K

» Objectif :

— Trouver le nombre K optimal de voisins pour le classifieur des K plus proches voisins.

» Paramètres en entrée :

— `Xapp` : les données d'apprentissage ;

— `zapp` : les étiquettes des données d'apprentissage ;

— `Xval` : les données de validation ;

— `zval` : les étiquettes des données de validation ;

— `nppv` : la liste des valeurs de K à tester ;

— `skipOneNeighbor` : si `VRAI`, ne prend pas en compte $K = 1$ pour éviter le sur-apprentissage des données d'apprentissage.

— `useRandIndexes` : si `VRAI`, utilise les indice de Rand afin de déterminer l'efficacité de la classification ; si `FAUX`, la proportion réelle d'étiquettes correctement prédites est utilisée pour calculer le taux de succès.

» Algorithme :

— Pour chaque valeur k de la liste `nppv` ;

— Obtenir les étiquettes prédites grâce à la fonction `kppv.val()` ;

— Calculer le taux de succès de la discrimination grâce à la fonction `compute.sucess.rate()` ou bien `adjustedRandIndex()` ;

— Si le taux de succès est égal au précédent, ajouter k à la liste des K optimum ;

— Sinon le taux de succès est supérieur au précédent, remplace la liste des K optimum par k ;

— Sinon ne rien faire, le k actuellement testé est moins efficace que le précédent ;

» Sortie :

— Un vecteur contenants les valeurs des K optimum (il peut y avoir plusieurs K optimum).

```r
kppv.tune <- function(Xapp, zapp, Xval, zval, nppv, skipOneNeighbor = TRUE, useRandIndexes = FALSE) {
    nbKToTest = length(nppv)
    maxSuccessRate = 0
    optimumForK = c()

    for (i in 1:nbKToTest) {
        K = nppv[i]
        if (skipOneNeighbor) if (K == 1) next
        zvalPredicted = kppv.val(Xapp, zapp, K, Xval)

        if (useRandIndexes)
            successRate = adjustedRandIndex(zval, zvalPredicted)
        else
            successRate = compute.sucess.rate(predictedClasses = zvalPredicted, actualClasses = zval)

        if (successRate >= maxSuccessRate) {
            if (successRate == maxSuccessRate) {
                optimumForK = c(optimumForK, K) # ajouter une autre valeur optimale possible de K
            }
            else { # "successRate" supérieur à "maxSuccessRate"
                optimumForK = c(K) # remplacer la valeur optimale de K
                maxSuccessRate = successRate # mettre à jour le "maxSuccessRate"
            }
        }
    }
    return(optimumForK)
}
```

FIGURE 1.4 – Code de la fonction `kppv.tune()`

1.1.3 Test des fonctions

Classifieur Euclidien

- Données d'apprentissage : fichier *Synth1-1000.csv* ;
- Données d'affichage : fichier *Synth1-40.csv*.

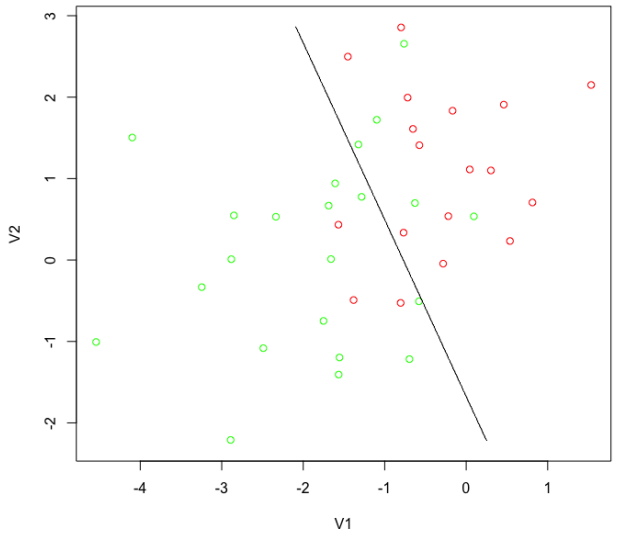


FIGURE 1.5 – Frontière de décision obtenue avec le classifieur Euclidien

	V1	V2
Classe 1	-0.01	0.92
Classe 2	-1.96	0.02

TABLE 1.1 – Coordonnées des centres de gravité des classes

K plus proches voisins

- Données d'apprentissage : fichier *Synth1-1000.csv* ;
- Données de validation : fichier *Synth1-500.csv* ;
- Données d'affichage : fichier *Synth1-40.csv* ;
- K optimum trouvé : 11.

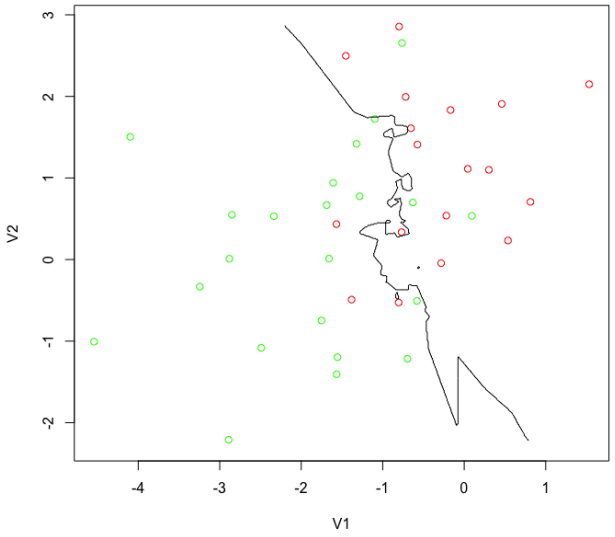


FIGURE 1.6 – Frontière de décision obtenue avec le classifieur des K plus proches voisins sur le jeu de données du fichier *Synth1-40.csv*, avec $Koptimum = 11$

1.2 Évaluation des performances

On va ici évaluer et analyser le taux d'erreur ϵ des classifieurs.

Voici la procédure suivie pour chaque jeu de données, à répéter N fois les actions suivantes :

- 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, tel le nombre K de voisins optimum pour le classifieur des K plus proches voisins) ;
- 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 $E_1, ..., E_N$ du taux d'erreur

(généralement effectuées sur l'ensemble de test). On peut alors déterminer une estimation ponctuelle $\hat{\epsilon}$ (moyenne) et un intervalle de confiance sur ϵ.

Notes sur les intervalles de confiances

On suppose que l'échantillon E constitué des N estimations $E_1, ..., E_N$ du taux d'erreur est ***iid*** :

- Les N observations sont indépendantes car les ensembles de test, d'apprentissage et de validation sont tirés de façon aléatoire ;
- Les N observations sont identiquement distribuées car, quand bien même on ne connaît pas la loi des observations originelles ayant conduit à l'obtention des taux d'erreur, l'échantillon est constituée de moyennes, qui suivent par définition une loi gaussienne.

La variance étant inconnue, on peut alors approcher ϵ (l'espérance de l'échantillon) avec la loi de Student à $N-1$ degrés de liberté :

$$\sqrt{N} \times \frac{\overline{E_N} - \epsilon}{\sigma^*} \sim S(N-1)$$

Et donc l'intervalle de confiance est donné par :

$$IC(\epsilon) = [\overline{E_N} - t_{1-\frac{\alpha}{2}} \frac{\sigma^*}{\sqrt{N}}, \overline{E_N} + t_{1-\frac{\alpha}{2}} \frac{\sigma^*}{\sqrt{N}}]$$

où $t_{1-\frac{\alpha}{2}}$ est le fractile d'ordre $1-\frac{\alpha}{2}$ de la loi de Student à $N-1$ degrés de liberté.

Notes sur l'analyse des résultats obtenus

La démarche ici est d'analyser la bonne performance ou non d'un classifieur.

Le classifieur Euclidien, par exemple, est basé sur un modèle supposant que :

- Les classes possèdent la même dispersion : les matrices Σ sont communes à toutes les classes
- La dispersion des classes est spérique : les matrices Σ sont scalaires et diagonales
- Les classes ont toutes la même proportion égale à $\frac{1}{g}$

De plus, le classifieur étant Euclidien, il aura tendance à mieux marcher si les centres de gravité μ_k des classes sont éloignés entre eux que s'ils sont proches.

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

Estimation des paramètres

Fichier source	Centres de gravité μ	Variances Σ	Proportions π
Synth1-40	$\mu_1 = \begin{pmatrix} -0.32 \\ 1.09 \end{pmatrix}$, $\mu_2 = \begin{pmatrix} -1.88 \\ 0.11 \end{pmatrix}$	$\Sigma_1 = \begin{pmatrix} 0.68 & 0.12 \\ 0.12 & 1.01 \end{pmatrix}$, $\Sigma_2 = \begin{pmatrix} 1.38 & 0.32 \\ 0.32 & 1.44 \end{pmatrix}$	$\begin{pmatrix} \pi_1 = 0.45 \\ \pi_2 = 0.55 \end{pmatrix}$
Synth1-100	$\mu_1 = \begin{pmatrix} 0.03 \\ 0.82 \end{pmatrix}$, $\mu_2 = \begin{pmatrix} -1.97 \\ -0.13 \end{pmatrix}$	$\Sigma_1 = \begin{pmatrix} 0.88 & -0.13 \\ -0.13 & 1.11 \end{pmatrix}$, $\Sigma_2 = \begin{pmatrix} 0.76 & -0.04 \\ -0.04 & 0.76 \end{pmatrix}$	$\begin{pmatrix} \pi_1 = 0.54 \\ \pi_2 = 0.46 \end{pmatrix}$
Synth1-500	$\mu_1 = \begin{pmatrix} 0.13 \\ 0.88 \end{pmatrix}$, $\mu_2 = \begin{pmatrix} -1.88 \\ -0.08 \end{pmatrix}$	$\Sigma_1 = \begin{pmatrix} 1.05 & 0.05 \\ 0.05 & 0.98 \end{pmatrix}$, $\Sigma_2 = \begin{pmatrix} 0.97 & -0.11 \\ -0.11 & 0.98 \end{pmatrix}$	$\begin{pmatrix} \pi_1 = 0.53 \\ \pi_2 = 0.47 \end{pmatrix}$
Synth1-1000	$\mu_1 = \begin{pmatrix} -0.01 \\ 0.92 \end{pmatrix}$, $\mu_2 = \begin{pmatrix} -1.96 \\ 0.02 \end{pmatrix}$	$\Sigma_1 = \begin{pmatrix} 0.97 & -0.07 \\ -0.07 & 1.08 \end{pmatrix}$, $\Sigma_2 = \begin{pmatrix} 0.99 & 0.02 \\ 0.02 & 0.94 \end{pmatrix}$	$\begin{pmatrix} \pi_1 = 0.5 \\ \pi_2 = 0.5 \end{pmatrix}$

TABLE 1.2 – Paramètres estimés des fichiers

Plus on estime les paramètres sur un ensemble de données conséquent, plus ils se rapprochent des hypothèses du modèle Euclidien avec $\Sigma_1 = \Sigma_2$ matrices diagonales et scalaires et $\pi_1 = \pi_2 = \frac{1}{g}$.

Évaluation des performances du classifieur Euclidien

Le tableau ci-dessous représente les estimations ponctuelle du taux d'erreur $\hat{\epsilon}$ et de l'intervalle de confiance sur ϵ (pour $\alpha = 0.05$) lors des discriminations effectuées avec le classifieur Euclidien. Ces mesures sont obtenues à partir d'un échantillon de $N = 20$ séparations aléatoires des jeux de données en ensembles d'apprentissage et de test.

Fichier source	Données d'apprentissage	Données de test
Synth1-40	$\hat{\epsilon} = 0.206$ $IC(\epsilon) = [0.18, 0.231]$	$\hat{\epsilon} = 0.242$ $IC(\epsilon) = [0.19, 0.295]$
Synth1-100	$\hat{\epsilon} = 0.093$ $IC(\epsilon) = [0.083, 0.103]$	$\hat{\epsilon} = 0.083$ $IC(\epsilon) = [0.062, 0.104]$
Synth1-500	$\hat{\epsilon} = 0.127$ $IC(\epsilon) = [0.121, 0.133]$	$\hat{\epsilon} = 0.146$ $IC(\epsilon) = [0.134, 0.158]$
Synth1-1000	$\hat{\epsilon} = 0.143$ $IC(\epsilon) = [0.139, 0.148]$	$\hat{\epsilon} = 0.142$ $IC(\epsilon) = [0.135, 0.15]$

TABLE 1.3 – Estimation du taux d'erreur $\hat{\epsilon}$ et de l'intervalle de confiance sur ϵ pour $N = 20$ avec le classifieur Euclidien

On observe plusieurs choses :

- Variation du taux d'erreur $\hat{\epsilon}$ entre les données d'apprentissage et de test
  - Parfois $\hat{\epsilon}_{app} < \hat{\epsilon}_{test}$ et parfois c'est l'inverse, $\hat{\epsilon}_{test} < \hat{\epsilon}_{app}$;
  - Cela dépend fortement des tirages des données d'apprentissage et de test qui sont aléatoires.
  - Sans tirer de conclusions, on peut supposer qu'avoir $\hat{\epsilon}_{app} < \hat{\epsilon}_{test}$ signale un sur-apprentissage, les centres de gravité des données d'apprentissage n'étant pas particulièrement représentatifs des centres de gravité des données de test ;
- Étendue des intervalles de confiance
  - L'étendue des intervalles de confiance sur ϵ_{app} est systématiquement plus réduite que celle des intervalles de confiance sur sur ϵ_{test} ;
  - L'apprentissage des paramètres du modèle se fait sur les données d'apprentissage, il est donc logique que l'écart-type de $\hat{\epsilon}_{app}$ (racine carré de la dispersion du taux d'erreur autour de son espérance) soit plus petit que celui de $\hat{\epsilon}_{test}$;
  - Ainsi, la valeur $\sigma^*/\sqrt{N}$ sera inférieure pour les données d'apprentissage, et donc l'étendue de l'intervalle de confiance $(t_{1-\alpha/2} \times \sigma^*/\sqrt{N})$ sera plus petite.
- Non corrélation apparente de la taille des jeux de données et des taux d'erreurs
  - Selon toute logique, plus le jeu de données est conséquent, plus le taux d'erreur devrait diminuer et converger vers le taux d'erreur de Bayes ;
  - Néanmoins, on peut toujours observer un taux d'erreur inférieur à celui de Bayes, tout dépend du jeu de données en question : s'il ne contient que deux points très éloignés, on aura par exemple un taux d'erreur nul ;
  - Cas du fichier « *Synth1-100* »
    - Ce fichier présente une estimation du taux d'erreur très faible par rapport aux autres, et il en va de même pour son intervalle de confiance ;

— Cela s'explique par les dispersions des deux classes, et plus particulièrement par celle de la classe numéro deux avec deux variances égales à 0.76 ;

— La zone de mélange entre les deux classes et donc très réduite, d'où le faible taux d'erreur.

Classifieur des K plus proches voisins et sur-apprentissage

On cherche ici déterminer le nombre optimal de voisins à l'aide de la fonction `kppv.tune()`, en utilisant l'ensemble d'apprentissage comme ensemble de validation.

On obtient $K = 1$, car on fait du sur-apprentissage. Le plus proche voisin d'un point étant lui-même, le taux de succès de la classification est de 100% pour $K = 1$.

Évaluation des performances du classifieur des K plus proches voisins

Le tableau ci-dessous représente les estimations ponctuelle du taux d'erreur $\hat{\epsilon}$ et de l'intervalle de confiance sur ϵ (pour $\alpha = 0.05$) lors des discriminations effectuées avec le classifieur des K plus proches voisins. Ces mesures sont obtenues à partir d'un échantillon de $N = 20$ séparations aléatoires des jeux de données en ensembles d'apprentissage et de test.

Fichier source	Données d'apprentissage	Données de test
Synth1-40	$\hat{\epsilon} = 0.19$ $IC(\epsilon) = [0.157, 0.223]$	$\hat{\epsilon} = 0.27$ $IC(\epsilon) = [0.219, 0.321]$
Synth1-100	$\hat{\epsilon} = 0.090$ $IC(\epsilon) = [0.072, 0.108]$	$\hat{\epsilon} = 0.112$ $IC(\epsilon) = [0.081, 0.144]$
Synth1-500	$\hat{\epsilon} = 0.122$ $IC(\epsilon) = [0.115, 0.129]$	$\hat{\epsilon} = 0.146$ $IC(\epsilon) = [0.133, 0.158]$
Synth1-1000	$\hat{\epsilon} = 0.126$ $IC(\epsilon) = [0.119, 0.133]$	$\hat{\epsilon} = 0.160$ $IC(\epsilon) = [0.15, 0.169]$

TABLE 1.4 – Estimation du taux d'erreur $\hat{\epsilon}$ et de l'intervalle de confiance sur ϵ pour $N = 20$ avec le classifieur des K plus proches voisins

Comparaison avec le classifieur Euclidien :

» Le classifieur des K plus proches voisins obtient de meilleurs résultats que le classifieur Euclidien sur les données d'apprentissage ;

» Il est par contre moins efficace que le classifieur Euclidien sur les données de test.

D'après l'estimation des paramètres, les données ont été générées de manière à respecter les hypothèses du classifieur Euclidien. Il est donc logique que ce dernier soit plus efficace sur les données de test. De plus, le fait d'avoir la même dispersion dans les deux classes peut pénaliser le travail du classifieur des K plus proches voisins : plus la dispersion d'une classe est faible, plus ses voisins sont proches et donc plus discrimination pourra être efficace pour cette classe.

1.2.2 Jeux de données *Synth2-1000*

Estimation des paramètres

Fichier source	Centres de gravité μ	Variances Σ	Proportions π
Synth2-1000	$\mu_1 = \begin{pmatrix} 3.02 \\ -0.01 \end{pmatrix}$, $\mu_2 = \begin{pmatrix} -2.14 \\ -0.03 \end{pmatrix}$	$\Sigma_1 = \begin{pmatrix} 0.99 & 0.11 \\ 0.11 & 1.09 \end{pmatrix}$, $\Sigma_2 = \begin{pmatrix} 4.43 & -0.15 \\ -0.15 & 1.03 \end{pmatrix}$	$\begin{pmatrix} \pi_1 = 0.52 \\ \pi_2 = 0.48 \end{pmatrix}$

TABLE 1.5 – Paramètres estimés du fichier « *Synth2-1000* »

Paramètres estimés versus modèle du classifieur Euclidien :

- Ces données semblent ne respecter que la condition d'égalité des proportions du modèle du classifieur Euclidien ;
- Seule la classe 1 possède une matrice de variance scalaire et diagonale (les termes non diagonaux ont des valeurs négligeables) ;
- Les Σ_k ne sont pas égales entre-elles : les classes n'ont pas la même dispersion.

Le classifieur Euclidien semblerait donc à première vue déconseillé. Par contre, on remarque que les centres de gravité sont à la fois alignés particulièrement éloignés et que la dispersion des classes définit une surface de chevauchement faible. Dans les faits, il est donc fort probable que le classifieur Euclidien soit efficace.

De même pour le classifieur des K plus proches voisins qui sera lui aussi vraisemblablement efficace. En raison de la faible surface de chevauchement et de la différence de dispersion, il y aura beaucoup plus de voisins de la classe 1 au sein de la zone de chevauchement que de voisin de classe 2.

Voici une représentation graphique des deux classes :

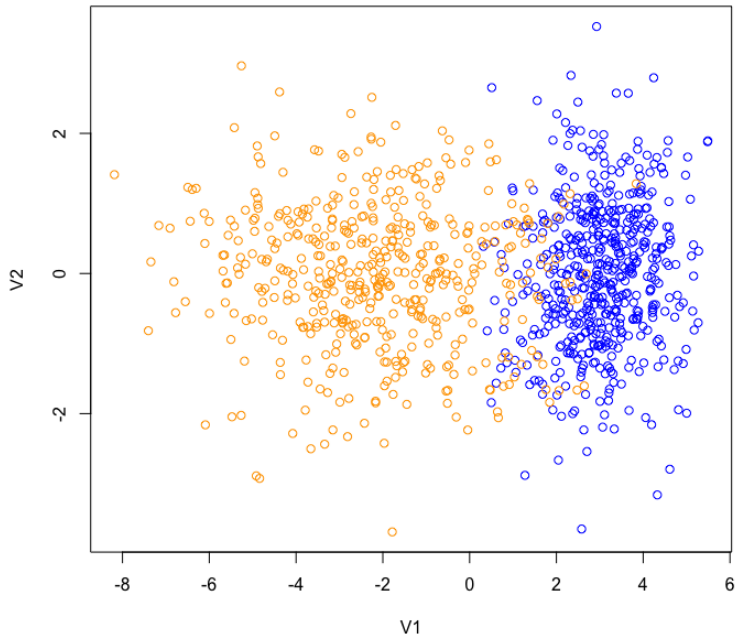


FIGURE 1.7 – Représentation graphique des données du fichier « *Synth2-1000* » en fonction des classes 1 et 2

Performances des classifieurs Euclidien et des K plus proches voisins

Fichier source	Données d'apprentissage	Données de test
Synth2-1000	$\hat{\epsilon} = 0.062$	$\hat{\epsilon} = 0.063$
	$IC(\epsilon) = [0.061, 0.064]$	$IC(\epsilon) = [0.059, 0.067]$

TABLE 1.6 – Estimation du taux d'erreur $\hat{\epsilon}$ et de l'intervalle de confiance sur ϵ pour $N = 20$ avec le classifieur Euclidien

Fichier source	Données d'apprentissage	Données de test
Synth2-1000	$\hat{\epsilon} = 0.052$	$\hat{\epsilon} = 0.061$
	$IC(\epsilon) = [0.049, 0.055]$	$IC(\epsilon) = [0.056, 0.066]$

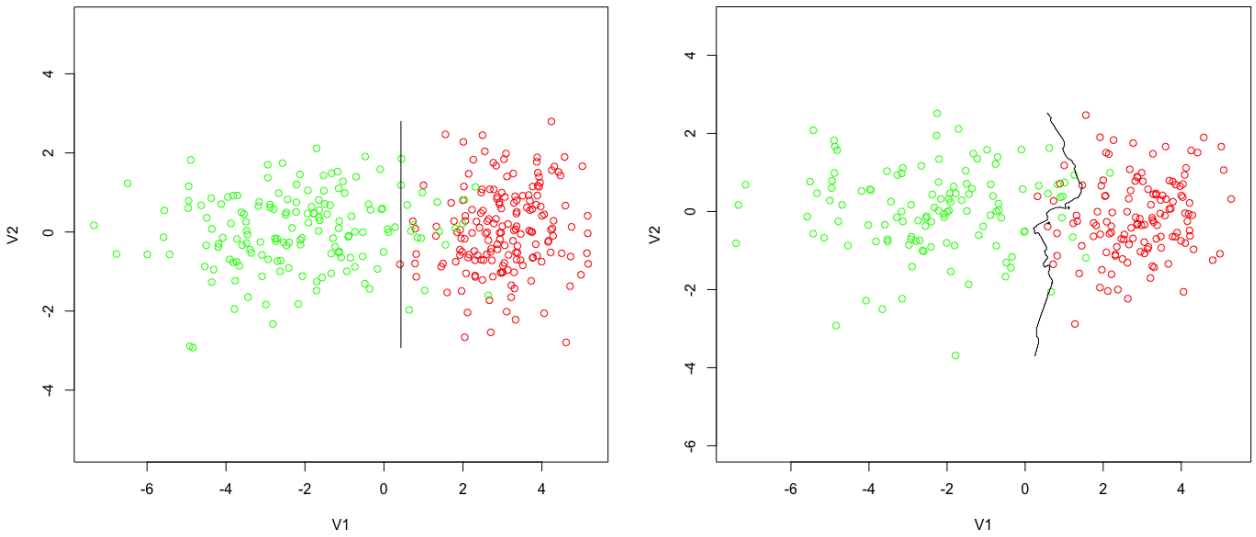
TABLE 1.7 – Estimation du taux d'erreur $\hat{\epsilon}$ et de l'intervalle de confiance sur ϵ pour $N = 20$ avec le classifieur des K plus proches voisins

Comme supposé à la vue des estimations des paramètres, les deux classifieurs sont très efficace, avec moins de 7% d'erreur.

Le classifieur Euclidien est assez stable, avec des taux d'erreur quasiment identiques entre les données d'apprentissage et de test.

Le classifieur des K plus proches voisins est encore une fois meilleur sur les données d'apprentissage que sur les données de test. Il est aussi légèrement plus efficace que le classifieur Euclidien, mais à peine. C'est surement du à sa capacité à mieux discriminer les individus au sein de la zone de chevauchement, en raison des différences de dispersion entre les classes.

Voici les frontières de décision obtenues sur les données de test une séparation aléatoire :



(a) Classifieur Euclidien

(b) Classifieur des K plus proches voisins

FIGURE 1.8 – Frontières de décision pour le fichier « *Synth2-1000* »

1.2.3 Jeux de données réelles

Taux d'erreur et intervalles de confiances

Classifieur Euclidien

Fichier source	Données d'apprentissage	Données de test
Breastcancer	$\hat{\epsilon} = 0.037$	$\hat{\epsilon} = 0.043$
	$IC(\epsilon) = [0.035, 0.039]$	$IC(\epsilon) = [0.038, 0.048]$
Pima	$\hat{\epsilon} = 0.243$	$\hat{\epsilon} = 0.254$
	$IC(\epsilon) = [0.235, 0.25]$	$IC(\epsilon) = [0.242, 0.267]$

TABLE 1.8 – Estimation du taux d'erreur $\hat{\epsilon}$ et de l'intervalle de confiance sur ϵ pour $N = 20$ avec le classifieur Euclidien

Classifieur des K plus proches voisins

Fichier source	Données d'apprentissage	Données de test
Breastcancer	$\hat{\epsilon} = 0.020$	$\hat{\epsilon} = 0.042$
	$IC(\epsilon) = [0.017, 0.024]$	$IC(\epsilon) = [0.036, 0.048]$
Pima	$\hat{\epsilon} = 0.188$	$\hat{\epsilon} = 0.265$
	$IC(\epsilon) = [0.174, 0.201]$	$IC(\epsilon) = [0.253, 0.277]$

TABLE 1.9 – Estimation du taux d'erreur $\hat{\epsilon}$ et de l'intervalle de confiance sur ϵ pour $N = 20$ avec le classifieur des K plus proches voisins

De toute évidence, la discrimination sur les données *Breastcancer* est très efficace, que ce soit avec le classifieur Euclidien ou celui des K plus proches voisins. Par contre pour ce qui est des données *Pima* le taux d'erreur avoisine les 25% en moyenne. 75% de chance de trouver juste est déjà bien, mais ce n'est pas fiable.

Possible explication grâce aux estimations des paramètres

Données *Breastcancer* :

	V2	V3	V4	V5	V6	V7	V8	V9	V10
μ_1	2.96	1.31	1.41	1.35	2.11	2.41	2.08	1.26	1.07
μ_2	7.19	6.58	6.56	5.59	5.33	4.71	5.97	5.86	2.60

TABLE 1.10 – Breastcancer : Centre de gravité des classes

	Proportions
π_1	0.65
π_2	0.35

TABLE 1.11 – Breastcancer : Proportions des classes

	V2	V3	V4	V5	V6	V7	V8	V9	V10
V2	**2.80**	0.39	0.48	0.39	0.23	0.21	0.18	0.33	-0.03
V3	0.39	**0.73**	0.57	0.22	0.31	0.48	0.24	0.40	0.02
V4	0.48	0.57	**0.92**	0.21	0.29	0.41	0.20	0.36	-0.00
V5	0.39	0.22	0.21	**0.84**	0.24	0.35	0.11	0.22	0.03
V6	0.23	0.31	0.29	0.24	**0.77**	0.30	0.14	0.37	-0.01
V7	0.21	0.48	0.41	0.35	0.30	**1.49**	0.28	0.34	0.05
V8	0.18	0.24	0.20	0.11	0.14	0.28	**1.13**	0.35	-0.02
V9	0.33	0.40	0.36	0.22	0.37	0.34	0.35	**0.91**	0.03
V10	-0.03	0.02	-0.00	0.03	-0.01	0.05	-0.02	0.03	**0.26**

TABLE 1.12 – Breastcancer : Matrice de covariance Σ_1

	V2	V3	V4	V5	V6	V7	V8	V9	V10
V2	**5.94**	0.65	0.70	-1.12	0.10	-0.42	-0.10	-0.11	0.74
V3	0.65	**7.42**	5.04	2.79	3.07	-0.41	2.42	2.73	1.68
V4	0.70	5.04	**6.60**	2.20	2.40	-0.25	1.98	2.66	1.38
V5	-1.12	2.79	2.20	**10.22**	1.51	-1.74	2.46	1.98	1.65
V6	0.10	3.07	2.40	1.51	**5.97**	-0.25	1.21	1.89	2.09
V7	-0.42	-0.41	-0.25	-1.74	-0.25	**7.06**	-0.61	0.34	0.03
V8	-0.10	2.42	1.98	2.46	1.21	-0.61	**5.21**	1.94	0.35
V9	-0.11	2.73	2.66	1.98	1.89	0.34	1.94	**11.21**	1.91
V10	0.74	1.68	1.38	1.65	2.09	0.03	0.35	1.91	**6.58**

TABLE 1.13 – Breastcancer : Matrice de covariance Σ_2

On remarque deux choses qui peuvent expliquer la bonne discrimination euclidienne et des K plus proches voisins :

- Les différences de valeurs entre les coordonnées des centres de gravité des classes
  - Toutes les coordonnées sont positives mais les coordonnées de la classe 2 sont deux à trois fois plus élevées ;
  - On peut donc en conclure que les centres de gravité des classes sont éloignés.
- Les différences de valeurs des dispersions des classes sur chaque axe (variables V1,...,V10)
  - La classe n°1 semble très peu dispersée et d'une forme assez sphérique. Seules les variances des variables V1 et V10.
  - Comparativement, la dispersion de la seconde classe est beaucoup plus importante avec un minimum de 5.21 pour V8 et un maximum de 11.21 pour V9.

En conclusion, on peut s'attendre à une très faible zone de chevauchement ainsi qu'à une classe légèrement sphérique et concentrée et une autre moins sphérique et plus dispersée.

Données *Pima* :

	npreg	glu	bp	skin	bmi	ped	age
μ_1	2.93	110.02	69.91	27.29	31.43	0.45	29.22
μ_2	4.70	143.12	74.70	32.98	35.82	0.62	36.41

TABLE 1.14 – Pima : Centre de gravité des classes

	Proportions
π_1	0.67
π_2	0.33

TABLE 1.15 – Pima : Proportions des classes

	npreg	glu	bp	skin	bmi	ped	age
npreg	**7.77**	4.62	6.63	3.67	0.01	-0.04	18.30
glu	4.62	**589.85**	56.25	32.57	25.48	0.66	42.91
bp	6.63	56.25	**141.68**	28.70	23.08	-0.13	39.42
skin	3.67	32.57	28.70	**101.61**	44.32	0.06	17.46
bmi	0.01	25.48	23.08	44.32	**42.86**	0.09	4.47
ped	-0.04	0.66	-0.13	0.06	0.09	**0.09**	0.06
age	18.30	42.91	39.42	17.46	4.47	0.06	**98.08**

TABLE 1.16 – Pima : Matrice de covariance Σ_1

	npreg	glu	bp	skin	bmi	ped	age
npreg	**15.36**	-9.87	6.14	-4.14	-4.65	-0.09	23.53
glu	-9.87	**977.50**	32.84	31.18	10.24	0.23	34.69
bp	6.14	32.84	**156.85**	12.36	18.01	-0.18	36.27
skin	-4.14	31.18	12.36	**108.06**	35.55	0.54	-7.44
bmi	-4.65	10.24	18.01	35.55	**43.71**	0.39	-13.77
ped	-0.09	0.23	-0.18	0.54	0.39	**0.16**	-0.15
age	23.53	34.69	36.27	-7.44	-13.77	-0.15	**117.45**

TABLE 1.17 – Pima : Matrice de covariance Σ_2

On remarque deux choses qui peuvent expliquer la mauvaise discrimination euclidienne et des K plus proches voisins :

- On n'observe pas de différences flagrantes entre les coordonnées des centres de gravité des classes, il se peut donc qu'ils soient relativement proches ;
- Valeurs des dispersions des classes
  - Les classes ne sont absolument pas sphériques ;
  - Elles ont par contre des dispersions qui sembles proches l'une de l'autres.

En conclusion, on peut s'attendre à une zone de chevauchement conséquentes, les centres de gravité étant relativement proche. De plus, la dispersion étant elle aussi proche entre les classes, le classifieur des K plus proches voisins ne sera pas beaucoup plus efficace que le classifieur Euclidien.

A. Fonction `compute.sucess.rate()`

```
compute.sucess.rate = function(predictedClasses, actualClasses) {
    zvalPredictedKnowingZvalContingency = table(predictedClasses, actualClasses)
    correctPredictions = sum(diag(zvalPredictedKnowingZvalContingency))
    totalPredictions = sum(zvalPredictedKnowingZvalContingency)
    successRate = correctPredictions / totalPredictions
}
```

FIGURE A.1 – Code de la fonction `compute.sucess.rate()`