Rapport TP3, Discrimination et théorie bayésienne de la décision

Sarah Richard et Solène Weill

26 mai 2016

1 Introduction

Lors du TP précédent, nous avons utilisé différentes techniques de classification automatique ou non supervisée. Nous allons maintenant nous intéresser à différentes méthodes de classification ou apprentissage supervisé. Le but de cette méthode est de construire des systèmes de prédiction, permettant de classer de nouveaux individus, à partir d'une règle de décision mise au point grâce à des observations sur un ensemble d'apprentissage. Dans le premier exercice, nous comparerons ainsi deux classifieurs, le *classifieur euclidien* et le *classifieur des K plus proches voisins*. Dans un second temps, nous utiliserons la théorie bayésienne de la décision qui permet de trouver une règle de décision optimale selon un certain critère.

2 Exercice 1 : Classifieur euclidien, *K* plus proches voisins

Dans cette première partie, nous souhaitons étudier et comparer les performances de deux classifieurs : le *classifieur euclidien* et le *classifieur des K plus proches voisins*.

2.1 Programmation des fonctions pour le *classifieur euclidien*

Nous avons codé 2 fonctions permettant l'utilisation de la méthode d'apprentissage avec le classifieur euclidien.

2.1.1 ceuc.app

La fonction *ceuc.app* permet de trouver les coordonnées du centre de chaque classe. Pour cela, on calcul la moyenne des coordonnées (abscisse et ordonnée) pour chaque classe, que l'on stocke dans une matrice.

2.1.2 ceuc.val

La fonction *ceuc.val* permet de calculer le vecteur des étiquettes d'un ensemble de test. Cela consiste à calculer la distance euclidienne entre chaque individu de l'ensemble et chaque centre de classe. Pour chaque individu, on lui affecte la classe du centre dont il est le plus proche.

Les codes des fonctions sont disponibles en *Annexe*.

2.2 Programmation des fonctions pour le *classifieur des K plus proches voisins*

2.2.1 kppv.val

La fonction *kppv.val* retourne un vecteur contenant les étiquettes des classes de l'ensemble de test pour une application de l'algorithme avec un nombre K de voisins déterminé. Pour ce faire, on utilise la fonction distXY, mise à disposition, qui permet de calculer la distance euclidienne entre l'ensemble d'apprentissage et l'ensemble de test. On sélectionne, ensuite, pour chaque individu de l'ensemble de test, la classe des k éléments de l'ensemble d'apprentissage pour lesquels la distance euclidienne est minimale. On concatène les résultats dans un vecteur. On calcule ensuite la fréquence de la classe 1 et de la classe 2 dans ce vecteur. Si la fréquence de la classe 1 est supérieure ou égale à celle de la classe 2, alors l'élément de l'ensemble de test sera étiqueté comme appartenant à la classe 1. Sinon, il sera étiqueté comme appartenant à la classe 2. On répète cette opération pour chaque individu.

2.2.2 kppv.tune

La fonction *kppv.tune* retournait, tout d'abord, un vecteur qui associait à chaque élément de nppv le nombre de bons classements effectués. Nous avons ensuite décidé de seulement retourner le K permettant de bien classer le plus d'individus. Dans le cas d'égalité entre plusieurs K, nous retournons le plus grand.
Pour chaque valeur de k voisin possible, on utilise la fonction décrite précédemment grâce à laquelle on obtient le vecteur d'étiquettes prédites pour l'ensemble de test. On parcourt ensuite ce vecteur et, pour chaque individu, on compare la classe trouvée avec la classe réelle qui est stockée dans un vecteur. Si la classe est la même, on incrémente le nombre de bonnes valeurs trouvées. On compare

ensuite ce nombre à un nombre de bonnes valeurs maximales que l'on a pris soin d'initialiser à 0. Si le nombre de bonnes valeurs est supérieur ou égal au nombre de bonnes valeurs maximales, K optimal prend la valeur de k.

Les codes des fonctions sont disponibles en *Annexe*.

2.3 Test des fonctions et visualisation des régions de décision

Nous avons exécuté les lignes de commande fournies dans le sujet de TP.

2.3.1 Pour le classifieur euclidien :

On obtient $\mu = \begin{pmatrix} 0.11145950 & 1.8673620 \\ -0.06781616 & -0.8345254 \end{pmatrix}$

Le vecteur contenant les étiquettes pour chaque individu est :

$$\begin{pmatrix} individus: & 16 & 17 & 18 & 19 & 20 & 36 & 37 & 38 & 39 & 40 \\ classe: & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 2 & 2 & 1 \end{pmatrix}$$

2.3.2 Pour le classifieur des K plus proche voisins :

Nous avons observé qu'avec $K = 1, 3, 5$ on observe une seule erreur d'étiquetage. Le K optimal se trouve parmi ces trois valeurs. Lorsque l'on choisi $K = 1$, cela revient à identifier la classe d'un individu à son voisin le plus proche, ce qui n'est pas très pertinent. On peut alors choisir un K optimal entre 3 et 5. Pour se faire, nous avons tracé les deux frontières de classes pour les classifieurs 3 et 5. Au vu des résultats, nous avons décidé de garder k=3. En effet, la décision dépend de l'ensemble d'apprentissage et moins la frontière de décision est précise plus nous aurons d'erreurs sur les valeurs de test.

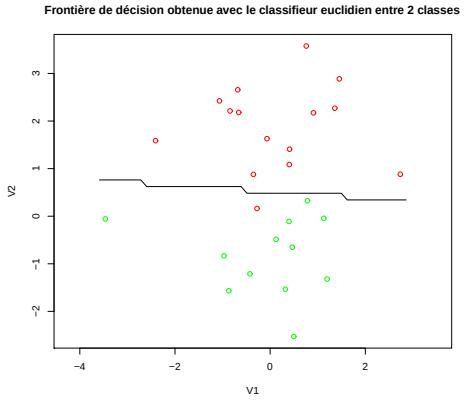

FIGURE 1 – Régions de décision obtenues avec le classifieur euclidien pour le jeu de données Synth1-40

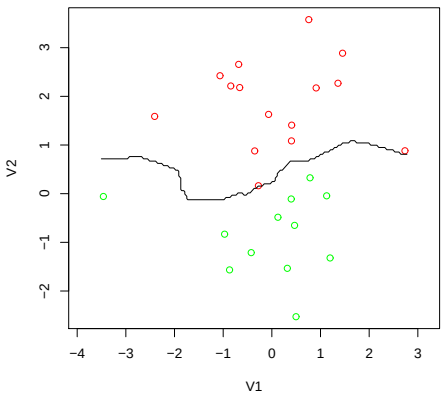

FIGURE 2 – Régions de décision obtenues avec le classifieur des K plus proche voisin $k = 3$ pour le jeu de données Synth1-40

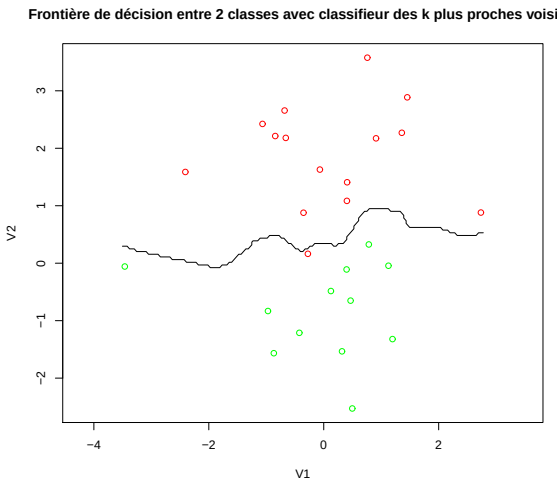

FIGURE 3 – Régions de décision obtenues avec le classifieur des K plus proche voisin $k = 5$ pour le jeu de données Synth1-40

2.4 Évaluation des performances, Jeux de données Synth1-40, Synth1-100, Synth1-500, Synth1-1000

On commence par calculer μ, Σ et π_k pour chacun des quatre jeux de données.

Synth1-40 :
$$\mu = \begin{pmatrix} -0.04917257 & 1.9760054 \\ 0.10989991 & -0.6960198 \end{pmatrix}, \Sigma = \begin{pmatrix} 1.5028251 & -0.2925628 \\ -0.2925628 & 2.5693724 \end{pmatrix} \text{ et } \pi_k = \begin{pmatrix} classe1 & classe2 \\ 0.575 & 0.425 \end{pmatrix}$$

Synth1-100 :
$$\mu = \begin{pmatrix} 0.1684090 & 2.059974 \\ -0.1872786 & -1.063586 \end{pmatrix}, \Sigma = \begin{pmatrix} 0.9102199 & 0.3446472 \\ 0.3446472 & 3.1470043 \end{pmatrix} \text{ et } \pi_k = \begin{pmatrix} classe1 & classe2 \\ 0.46 & 0.54 \end{pmatrix}$$

Synth1-500 :
$$\mu = \begin{pmatrix} 0.02752285 & 2.0036738 \\ -0.03351599 & -0.9013625 \end{pmatrix}, \Sigma = \begin{pmatrix} 0.94498940 & 0.03510779 \\ 0.03510779 & 2.98624360 \end{pmatrix} \text{ et } \pi_k = \begin{pmatrix} classe1 & classe2 \\ 0.516 & 0.484 \end{pmatrix}$$

Synth1-1000 :
$$\mu = \begin{pmatrix} -0.08398649 & 1.984757 \\ -0.03848676 & -1.013359 \end{pmatrix}, \Sigma = \begin{pmatrix} 1.00089859 & -0.01799136 \\ -0.01799136 & 3.22599880 \end{pmatrix} \text{ et } \pi_k = \begin{pmatrix} classe1 & classe2 \\ 0.481 & 0.519 \end{pmatrix}$$

2.4.1 Evaluation des performances pour le *classifieur euclidien*

Nous avons tout d'abord codé une fonction nous permettant de générer 20 séparations aléatoires de chaque jeu de données en un ensemble d'apprentissage et de test. Il calcule ensuite le taux d'erreur d'apprentissage et de test en utlisant le classifieur euclidien pour prédire des étiquettes. Elles sont par la suite comparées aux véritables classes des individus. Un vecteur des taux d'erreur pour chaque ensemble est retourné. La fonction est disponible en *Annexe*.
Résultats avec le jeu de 40 données :

	Taux d'erreur d'apprentissage	Taux d'erreur de test
1	0.07692308	0.07142857
2	0.03846154	0.14285714
3	0.07692308	0.07142857
4	0.03846154	0.14285714
5	0.07692308	0.00000000
6	0.03846154	0.14285714
7	0.07692308	0.14285714
8	0.03846154	0.07142857
9	0.03846154	0.21428571
10	0.07692308	0.14285714
11	0.07692308	0.07142857
12	0.07692308	0.21428571
13	0.03846154	0.07142857
14	0.07692308	0.1428571
15	0.1153846	0.07142857
16	0.03846154	0.21428571
17	0.07692308	0.07142857
18	0.03846154	0.14285714
19	0.0000000	0.2857143
20	0.03846154	0.14285714

Pour chaque jeu de données, on calcule le taux d'erreur de test et d'apprentissage. Pour cela, on effectue pour chaque jeu de données la moyenne des taux d'erreur de test et celle des taux d'erreur d'apprentissage.

	Synth1-40	Synth1-100	Synth1-500	Synth1-1000
Taux d'erreur d'apprentissage	0.05769231	0.05223881	0.0493994	0.06169415
Taux d'erreur de test	0.1285714	0.05757576	0.05628743	0.07372372

On peut ainsi remarquer que plus le nombre d'individus augmente, plus le taux d'erreur de test est faible. L'apprentissage s'effectue sur un plus grand nombre de données, la règle de décision est donc plus précise. Cela est cohérent avec le fait que la fiabilité du classifieur euclidien est en partie basée sur la géométrie des classes. Plus le nombre d'individus est grand plus la forme géométrique est repérable.
Cependant, on remarque une augmentation du taux pour le jeu de données de 1000 valeurs. Cela s'explique par le fait que les données sont plus dispersées. De plus, les points des différentes classes sont mélangés au niveau de la frontière de décision, ce qui rend la classification plus difficile car la frontière est linéaire. On remarque aussi qu'à l'exception du jeu de données *Synth1-40*, le taux d'erreur sur l'ensemble de test est proche de celui sur l'ensemble d'apprentissage.

On calcule ensuite à partir de ces taux d'erreur l'intervalle de confiance. On cherche un intervalle de confiance pour la moyenne et l'espérance est inconnue, on a donc pour fonction pivotale :

$$\frac{\bar{X} - \mu}{\frac{S^*}{\sqrt{n}}} \sim \tau_{n-1} \tag{1}$$

Ce qui donne comme intervalle de confiance :

$$\bar{X} - t_{n-1,1-\frac{\alpha}{2}} \frac{S^*}{\sqrt{n}} \leqslant \mu \leqslant \bar{X} + t_{n-1,1-\frac{\alpha}{2}} \frac{S^*}{\sqrt{n}} (2)$$

On obtient ainsi pour chaque jeu de données :

	Intervalle de confiance d'apprentissage	Intervalle de confiance de test
Synth1-40	[-0.02284413 ; 0.1382287]	[-0.07819531 ; 0.3353382]
Synth1-100	[0.009285741 ; 0.09519187]	[-0.057886140 ; 0.17303765]
Synth1-500	[0.03330621 ;0.06549259]	[0.01871807 ;0.09385678]
Synth1-1000	[0.04160476 ; 0.08178355]	[0.03479427 ; 0.11265318]

Il est ainsi possible de remarquer que les intervalles de confiance sont liés aux taux d'erreur. En effet, plus les taux d'erreur sont élevés, plus l'intervalle de confiance est large et inversement.

2.4.2 Exécution du *classifieur des K plus proches voisins* sur l'ensemble d'apprentissage

On effectue, tout d'abord, une séparation aléatoire de l'ensemble des données pour le jeu de 40 individus en un ensemble d'apprentissage et un ensemble de test.
Avec la fonction kppv.tune, on détermine le nombre de voisins optimal en utilisant l'ensemble d'apprentissage comme ensemble de validation. Le nombre de voisins optimal est alors de 1. Ce résultat est logique car on compare les mêmes ensemble. En effet, on affecte au point l'étiquette qui correspond au plus proche voisin. Or, le plus proche voisin du point est le point lui-même.

2.4.3 Evaluation des performances pour le *classifieur des K plus proches voisins*

Nous avons, tout d'abord, codé une fonction nous permettant de générer 20 séparations aléatoires de chaque jeu de données en ensemble d'apprentissage, de validation et de test. Celle-ci calcule ensuite le taux d'erreur d'apprentissage et de test en utlisant la méthode des k plus proches voisins pour prédire des étiquettes qu'elle compare ensuite aux véritables classes des individus. Un vecteur des taux d'erreur pour chaque ensemble est retourné. La fonction est disponible en *Annexe*.
Résultats avec le jeu de 40 données :

	Taux d'erreur d'apprentissage	Taux d'erreur de test
1	0.2	0.0
2	0.15	0.10
3	0.1	0.3
4	0.1	0.1
5	0.25	0.00
6	0.1	0.1
7	0.05	0.30
8	0.05	0.10
9	0.1	0.3
10	0.25	0.10
11	0.2	0.2
12	0.05	0.00
13	0	0
14	0.05	0.30
15	0.1	0.1
16	0.15	0.00
17	0.1	0.3
18	0.15	0.20
19	0.15	0.10
20	0.1	0.20

Pour chaque jeu de données, on calcule le taux d'erreur de test et d'apprentissage. Pour cela, on effectue pour chaque jeu de données la moyenne des taux d'erreur de test et celle des taux d'erreur d'apprentissage.

	Synth1-40	Synth1-100	Synth1-500	Synth1-1000
Taux d'erreur d'apprentissage	0.12	0.044	0.0504	0.0607
Taux d'erreur de test	0.14	0.06458333	0.06626984	0.0748

En utilisant la méthode des k plus proche voisins, on remarque que les taux d'erreur sont forts lorsqu'il y a très peu de valeurs. Pour les jeux de données de 100, 500 et 1000 valeurs, on observe une augmentation continue du taux d'erreur sur l'ensemble d'apprentissage comme sur l'ensemble de test. Cela peut être dû au fait que plus il y a de points dans la classe plus les distances entre le point et ses voisins sont faibles, ce qui, par un maque de précision, peut entraîner une erreur de classification.
On utilise la même fonction pivotale que précédemment pour l'intervalle de confiance. On obtient ainsi pour chaque jeu de données :

	Intervalle de confiance d'apprentissage	Intervalle de confiance de test
Synth1-40	[-0.08587524 ; 0.3258752]	[-0.20759245 ; 0.4875925]
Synth1-100	[-0.02323857 ; 0.1112386]	[-0.06853114 ; 0.1976978]
Synth1-500	[-0.001592899 ; 0.1023929]	[0.015939197 ; 0.1166005]
Synth1-1000	[0.03133326 ; 0.09006674]	[0.03225627 ; 0.11734373]

De même que précédemment, les intervalles de confiance sont liés à la grandeur des taux d'erreur.

En comparant les deux classifieurs, on réalise que le classifieur euclidien est le plus adapté et performant sur ces jeux de données. En effet, les taux d'erreurs sont plus faibles et les intervalles de confiance plus précis. Cela est dû à la répartition des points dans l'espace. En visualisant les jeux de données (cf Annexes), on remarque que les points des jeux de données sont répartis dans des classes sphériques de taille homogène. Plus le jeu de données contient de points plus la géométrie de la classe est visible.

2.4.4 Jeu de données Synth2-1000

On commence par calculer μ, Σ et π_k

Synth2-1000 :

$$\mu = \begin{pmatrix} -0.08117442 & 2.972041 \\ -0.02142483 & -5.093154 \end{pmatrix}, \Sigma = \begin{pmatrix} 3.19503880 & 0.08422374 \\ 0.08422374 & 19.34216966 \end{pmatrix} \text{ et } \pi_k = \begin{pmatrix} classe1 & classe2 \\ 0.486 & 0.514 \end{pmatrix}$$

2.4.5 Evaluation des performances pour le *classifieur euclidien*

Résultats avec Synth2-1000 :

	Taux d'erreur d'apprentissage	Taux d'erreur de test
1	0.02398801	0.01801802
2	0.02248876	0.02402402
3	0.02398801	0.01801802
4	0.02098951	0.01801802
5	0.02398801	0.01801802
6	0.01799100	0.03003003
7	0.01949025	0.02702703
8	0.02398801	0.01201201
9	0.01799100	0.03003003
10	0.01649175	0.03303303
11	0.02248876	0.02102102
12	0.02098951	0.01801802
13	0.02398801	0.01801802
14	0.02098951	0.02402402
15	0.01649175	0.03003003
16	0.01649175	0.03303303
17	0.02398801	0.01801802
18	0.02248876	0.02102102
19	0.01799100	0.03003003
20	0.02698651	0.02102102

On calcule ensuite le taux d'erreur de test et d'apprentissage. Pour cela, on effectue la moyenne des taux d'erreur de test et celle des taux d'erreur d'apprentissage.

	Synth2-1000
Taux d'erreur d'apprentissage	0.02121439
Taux d'erreur de test	0.02312312

On peut remarquer que les erreurs du taux d'apprentissage et de test sont faibles.
On utilise les mêmes formules que précédemment pour calculer l'intervalle de confiance. On obtient ainsi :

	Synth2-1000
Intervalle de confiance d'apprentissage	[0.011706655 ; 0.03072213]
Intervalle de confiance de test	[0.004352121 ; 0.04189412]

2.4.6 Evaluation des performances pour le *classifieur des K plus proches voisins*

Nous avons tout d'abord codé une fonction nous permettant de générer 20 séparations aléatoires de chaque jeu de données en ensemble d'apprentissage, de validation et de test. Celle-ci calcule ensuite le taux d'erreur d'apprentissage et de test en utlisant la méthode des k plus proche voisins pour prédire des étiquettes qu'elle compare ensuite aux véritables classes des individus. Un vecteur des taux d'erreur pour chaque ensemble est retourné. La fonction est disponible en *Annexe*.

Résultats avec Synth2-21000 :

	Taux d'erreur d'apprentissage	Taux d'erreur de test
1	0.000	0.004
2	0.004	0.004
3	0.004	0.008
4	0.010	0.004
5	0.006	0.008
6	0.006	0.000
7	0.006	0.012
8	0.008	0.004
9	0.008	0.004
10	0.006	0.004
11	0.012	0.004
12	0.000	0.012
13	0.000	0.008
14	0.000	0.000
15	0.004	0.008
16	0.004	0.004
17	0.000	0.004
18	0.008	0.016
19	0.000	0.004
20	0.004	0.020

On calcule le taux d'erreur de test et d'apprentissage. Pour cela, on effectue la moyenne des taux d'erreur de test et celle des taux d'erreur d'apprentissage.

	Synth2-1000
Taux d'erreur d'apprentissage	0.0045
Taux d'erreur de test	0.0066

Les taux d'erreur sur ce jeu de données sont encore plus faible que précédemment. On peut donc supposer qu'en plus de la taille du jeu de données, la répartition des individus dans les classes a une influence sur la précision et la fiabilité des classifieurs.

On utilise la même fonction pivotale que précédemment pour l'intervalle de confiance. On obtient ainsi pour chaque jeu de données :

	Synth2-1000
Intervalle de confiance d'apprentissage	[-0.006645886 ; 0.01564589]
Intervalle de confiance de test	[-0.008831569 ; 0.02203157]

On observe que, pour ce jeu de données, le classifieur des k plus proches voisins est le plus performant. Lorsque l'on visualise la répartition des points dans l'espace, on remarque que les classes ne sont pas homogènes. Elles sont sphériques mais les points de la classe ω_1 sont moins dispersés dans l'espace. La frontière de décision va donc avoir tendance à se déplacer vers la classe ω_2.

3 Exercice 2 : Règle de Bayes

3.1 Calcul des distributions marginales des variables dans chaque classe

Nous avons réalisé le calcul des distributions marginales des variables dans chaque classe à la main.
La fonction de densité de la loi normale bivariée est : $f = \frac{1}{(2\pi)^{r/2}|\Sigma|^{1/2}} \cdot \exp\left(-1/2(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$

3.1.1 Jeux de données *Synth-1.40, Synth-1.100, Synth-1.500, Synth-1.1000*

Pour la classe ω_1 : $f_{X^2}(x_2) = \frac{1}{\sqrt{2\pi}} \cdot \exp\left(-1/2(x_1)^2\right) \cdot \frac{1}{\sqrt{2\pi}} \cdot \exp\left(-1/2(x_2-2)^2\right)$
et par identification, $\sigma = 1$ $\mu = (0, 2)$
Ainsi, la répartition des variables dans la classe est :
$X^1 \sim N(0, 1)$
$X^2 \sim N(2, 1)$
Pour la classe ω_2 : $f_{X^1}(x_1) = \frac{1}{\sqrt{2\pi}} \cdot \exp\left(-1/2(x_1)^2\right) \cdot \frac{1}{\sqrt{2\pi}} \cdot \exp\left(-1/2(x_2+1)^2\right)$
et par identification, $\sigma = 1$ $\mu = (0, -1)$
Ainsi, la répartition des variables dans la classe est :
$X^1 \sim N(0, 1)$
$X^2 \sim N(-1, 1)$

3.1.2 Jeux de données *Synth-2.1000*

Pour la classe ω_1 : $f_{X^2}(x_2) = \frac{1}{\sqrt{2\pi}} \cdot \exp\left(-1/2(x_1)^2\right) \cdot \frac{1}{\sqrt{2\pi}} \cdot \exp\left(-1/2(x_2-3)^2\right)$
et par identification, $\sigma = 1$ $\mu = (0, 3)$
Ainsi, la répartition des variables dans la classe est :
$X^1 \sim N(0, 1)$
$X^2 \sim N(3, 1)$
Pour la classe ω_2 : $f_{X^1}(x_1) = \frac{1}{\sqrt{2\pi}\sqrt{5}} \cdot \exp\left(-1/10(x_1)^2\right) \cdot \frac{1}{\sqrt{2\pi}\sqrt{5}} \cdot \exp\left(-1/10(x_2+5)^2\right)$
et par identification, $\sigma = \sqrt{5}$ $\mu = (0, -5)$
Ainsi, la répartition des variables dans la classe est :
$X^1 \sim N(0, \sqrt{5})$
$X^2 \sim N(-5, \sqrt{5})$

3.2 Calcul des courbes d'iso-densité

Pour montrer que les courbes d'iso-densité sont des cercles, on utilise la loi jointe des variables pour chaque classe et on résout $f_i(x) = k$ et on montre que ces équations sont de la forme, $(x-a)^2 + (y-b)^2 = r^2$ qui est l'équation d'un cercle.

3.2.1 Jeux de données *Synth-1.40, Synth-1.100, Synth-1.500, Synth-1.1000*

Pour la classe ω_1 :
$\frac{1}{2\pi} \cdot \exp\left(-1/2\left((x_1)^2 + (x_2-2)^2\right)\right) = k$
Après passage au logarithme, on obtient $(x_1)^2 + (x_2-2)^2 = -2\ln(2\pi k)$
On peut ainsi tracer un cercle de centre $c_1 = (0, 2)$ et de rayon $R_2 = \sqrt{-2\ln(2\pi k)}$
Pour la classe ω_2 :
$\frac{1}{2\pi} \cdot \exp\left(-1/2\left((x_1)^2 + (x_2+1)^2\right)\right) = k$
Après passage au logarithme, on obtient $(x_1)^2 + (x_2+1)^2 = -2\ln(2\pi k)$

On peut ainsi tracer un cercle un cercle de centre $c_2 = (0, -1)$ et de rayon $R_2 = \sqrt{-2\ln(2\pi k)}$

3.2.2 Jeux de données *Synth-2.1000*

Pour la classe ω_1 :
On pose $\frac{1}{2\pi} \cdot \exp\left(-1/2\left((x_1)^2 + (x_2 - 3)^2\right)\right) = k$
Après résolution, on obtient un cercle de centre $c_1 = (0, 3)$ et de rayon $R_1 = \sqrt{-2\ln(2\pi k)}$.
Pour la classe ω_2 :
$\frac{1}{5*2\pi} \cdot \exp\left(-1/10\left((x_1)^2 + (x_2 + 5)^2\right)\right) = k$
Après résolution, on obtient un cercle de centre $c_2 = (0, -5)$ et de rayon $R_2 = \sqrt{-10\ln(10\pi k)}$.

3.3 Expression de la règle de Bayes

La règle de Bayes consiste à attribuer à chaque individu x la classe de la plus grande probabilité a posteriori. Nous avons deux classes, c'est pourquoi nous pouvons exprimer cette règle en fonction du rapport de vraisemblance $\delta^*(x) = \frac{f_1(x)}{f_2(x)} > \frac{\pi_2}{\pi_1}$

Ainsi, la règle de décision pour les 4 premiers jeux de données est : $\delta^*(x) = \begin{cases} \omega_1 & \text{si } x_2 > \frac{1}{2} \\ \omega_2 & \text{sinon.} \end{cases}$

Et celle du jeux de données Synth.2-1000 est : $\delta^*(x) = \begin{cases} \omega_1 & \text{si } x_1^2 + (x_2 - 5)^2 < 5(\frac{\ln 5}{2} + 4) \\ \omega_2 & \text{sinon.} \end{cases}$

3.4 Frontière de décision des quatre premiers jeux de données

Grâce à la règle de décision calculée précédemment, nous obtenons les quatre graphiques suivants :

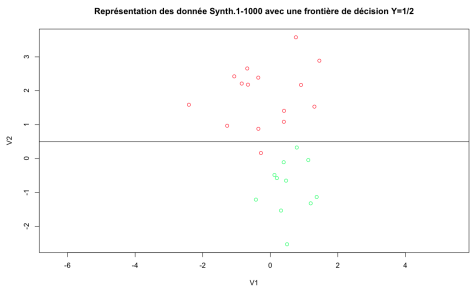

FIGURE 4 – Frontière de décision pour le jeu de données Synth1-40.

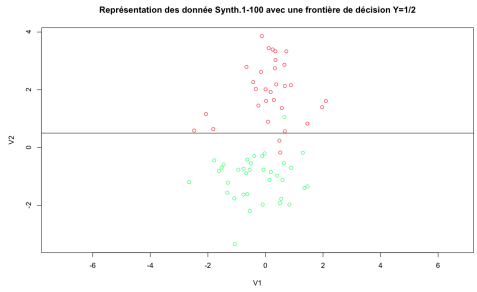

FIGURE 5 – Frontière de décision pour le jeu de données Synth1-100.

Cette frontière de décision linéaire concorde avec celle que nous aurions pu trouver avec le classifieur euclidien. On remarque, en effet, que la matrice de covariance est la même pour les deux classes et est égale à $\Sigma_k = \sigma^2 I_p$ et les probabilités a priori des classes sont $\pi_k = 1/g$ avec $g = 2$.
Le frontière trouvée ressemble bien à celle trouvée dans le premier exercice et sépare les classes efficacement. En effet, la majorité des points se situe du bon côté de la frontière de décision.

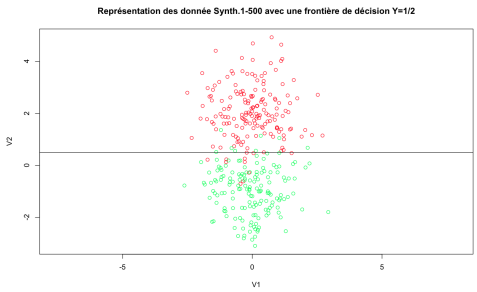

FIGURE 6 – Frontière de décision pour le jeu de données Synth1-500.

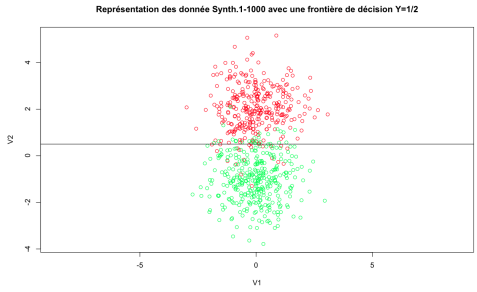

FIGURE 7 – Frontière de décision pour le jeu de données Synth1-1000.

3.5 Calcul de l'erreur de Bayes pour les quatre premiers jeux de données

Le cas des 4 premiers jeux de données est représentatif d'un cas simple où $g = 2$ et $\Sigma_k = \Sigma$ et où $\pi_1 = \pi_2$. On peut ainsi calculer la probabilité d'erreur exacte de Bayes grâce à la formule $\epsilon^* = \Phi\left(\frac{-\Delta}{2}\right)$. Avec Δ la racine carré de la distance de Mahalanobis entre les deux classes ω_1 et ω_2. $\Delta^2 = (\mu_2 - \mu_1)^T \Sigma^{-}1(\mu_2 - \mu_1)$.
Après application numérique, on trouve $\Delta^2 = 9$ d'où $\Delta = 3$ $\epsilon^* = \Phi\left(\frac{-\Delta}{2}\right) = \Phi\left(\frac{-3}{2}\right) = 1 - \Phi\left(\frac{3}{2}\right) = 1 - 0,9332 = 0,0668$

L'erreur de Bayes est la même pour les quatre jeux de données. Cette erreur est similaire à celle trouvée dans l'exercice précédent ce qui conforte nos résultats.

4 Conclusion

En conclusion, ce TP nous a permis de découvrir et de mettre en pratique les algorithmes pour trois méthodes d'apprentissage supervisé.
Tout d'abord, le classifieur euclidien est un classifieur simple. Il fournit de bons résultats lorsque les nuages de points sont sphériques et de même volume afin d'avoir une frontière de décision équidistante des centres de classes. La frontière représente alors l'hyperplan médian entre les deux classes.
Le classifieur des k plus proche voisins, quant-à-lui, est plus efficace sur les jeux de données où les classes ne sont pas homogènes.
Finalement, nous avons utilisé la règle de Bayes afin de contrôles les risques. Ainsi, il devient possible de réduire le risque d'erreur et de privilégier un certain type d'erreur à un autre. Ce TP nous aura également fait découvrir de nouveaux outils de R.

5 Annexe

```
> ceuc.app <- function(Xapp, zapp)
{
mu<-matrix(0, 2, length(Xapp));
nbInd <- table(zapp);
for(i in 1:length(Xapp)){
sumXapp<-0;
sumYapp<-0;
    for(j in 1:length(Xapp[,1])) {
        if(zapp[j]==1){
            sumXapp<-sumXapp+Xapp[j,i];
        }
        else if(zapp[j]==2){
            sumYapp<-sumYapp+Xapp[j,i];
        }
    }
mu[1,i]<-sumXapp/nbInd[1];
mu[2,i]<-sumYapp/nbInd[2];
}
mu
}
```

FIGURE 8 – Code de la fonction ceuc.app calculant les centres de classe.

```
> ceuc.val <- function(mu, Xtst)
{
V<-c();
d<-distXY(mu, Xtst);
V<-apply(d,2 ,which.min)
}
```

FIGURE 9 – Code de la fonction ceuc.val calculant le vecteur d'étiquettes prédites de l'ensemble de test.

```
> kppv.val <- function(Xapp, zapp, k, Xtst){
class<-c();
d<-distXY(as.matrix(Xapp), as.matrix(Xtst));
for(i in 1:length(Xtst[,1])){
    cpos<-c();
    for(j in 1:k){
            minpos<-which.min(d[,i]);
                d[minpos,i]<-NA;
                cpos<-cbind(cpos, zapp[minpos]);
    }
    freq<-table(cpos);
            if(length(freq)>1) {
                if(freq[1]>=freq[2]) {
                    class<-cbind(class,1);
                }
                else {
                    class<-cbind(class,2);
                }
            }
            else {
                class<-cbind(class,cpos[1]);
            }
}
class
}
```

FIGURE 10 – Code de la fonction kppv.val calculant les étiquettes des individus pour un k donné.

```
> kppv.tune <- function(Xapp, zapp, Xval, zval, nppv)
{
    d<-distXY(as.matrix(Xapp), as.matrix(Xval));
    class<-matrix(0, dim(d)[2], length(nppv));
    comptMax<-0;
    Kopt<-matrix(0, 2, length(nppv))
    #dsort<-c();
    for(k in 1:length(nppv)) { #Pour le nombre de voisins optimaux possibles
        #nppv de K voisins
        compt<-0;
        class<-kppv.val(Xapp, zapp, nppv[k], Xval);
        print(zval);
        for(j in 1:length(class)) {
            if(class[j] == zval[j]) {
                compt<-compt+1;
            }
        }
        if(compt>=comptMax) {
            comptMax<-compt;
            Kopt[1,k]<-nppv[k];
            Kopt[2,k]<-comptMax;
        }
    }
    Kopt;
}
```

FIGURE 11 – Code de la fonction kppv.tune calculant le ou les K optimaux.

5.1 Fonction de calcul des taux d'erreur et d'apprentissage pour les 20 séparations générées aléatoirement par la méthode du classifieur euclidien.

```
aleaEuclidien<-function(donnees) {
terrorapp<-c();
terrortst<-c();
```

```
X <- donnees[,1:2]
z <- donnees[,3]
res<-matrix(0,2,20);
for(i in 1:20) {
donnees.sep <- separ1(X, z)
comptapp<-0;
compttst<-0;
Xapp <- donnees.sep$Xapp
zapp <- donnees.sep$zapp
Xtst <- donnees.sep$Xtst
ztst <- donnees.sep$ztst
mu<-ceuc.app(Xapp, zapp);
ClassApp<-c();
ClassTst<-c();
ClassApp<-ceuc.val(mu, Xapp);
ClassTst<-ceuc.val(mu, Xtst);
for(j in 1:length(ClassApp)) {
if(ClassApp[j] != zapp[j]) {
comptapp<-comptapp+1;
}
}
terrorapp[i]<-comptapp/(length(ClassApp));
res[1,i]<-terrorapp[i];
for(j in 1:length(ClassTst)) {
if(ClassTst[j] != ztst[j]) {
compttst<-compttst+1;
}
}
terrortst[i]<-compttst/(length(ClassTst));
res[2,i]<-terrortst[i];
}
res
}
```

5.2 Fonction de calcul des taux d'erreur et d'apprentissage pour les 20 séparations générées aléatoirement par la méthode des K plus proches voisins.

```
aleaVoisin<-function(donnees) {
terrorapp<-c();
terrortst<-c();
X <- donnees[,1:2]
z <- donnees[,3]
res<-matrix(0,2,20);
for(i in 1:20) {
donnees.sep <- separ2(X, z)
comptapp<-0;
compttst<-0;
Xapp <- donnees.sep$Xapp
zapp <- donnees.sep$zapp
```

```
Xval <- donnees.sep$Xval
zval <- donnees.sep$zval
Xtst <- donnees.sep$Xtst
ztst <- donnees.sep$ztst
ClassApp<-c();
ClassTst<-c();
Kopt <- kppv.tune(Xapp, zapp, Xval, zval, 1:10)
ClassApp <- kppv.val(Xapp, zapp, Kopt, Xapp)
ClassTst <- kppv.val(Xapp, zapp, Kopt, Xtst)

for(j in 1:length(ClassApp)) {
if(ClassApp[j] != zapp[j]) {
comptapp<-comptapp+1;
}
}
terrorapp[i]<-comptapp/(length(ClassApp));
res[1,i]<-terrorapp[i];

for(y in 1:length(ClassTst)) {
if(ClassTst[y] != ztst[y]) {
compttst<-compttst+1;
}
}
terrortst[i]<-compttst/(length(ClassTst));
res[2,i]<-terrortst[i];

}
res
}
```

5.3 Représentation des données Synth1-40.

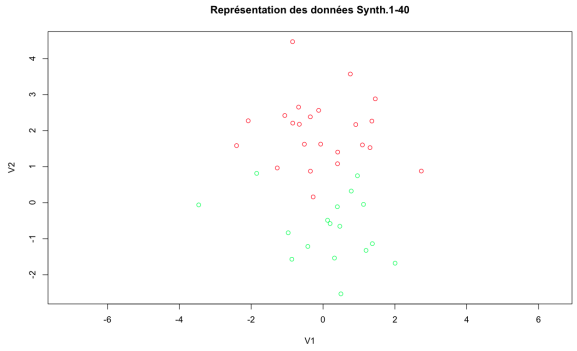

FIGURE 12 – Représentation des données Synth1-40.

5.4 Représentation des données Synth1-100.

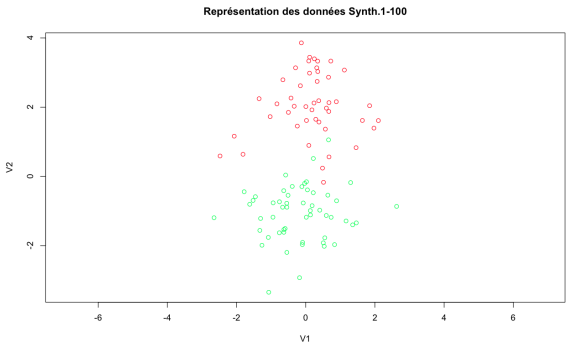

FIGURE 13 – Représentation des données Synth1-100.

5.5 Représentation des données Synth1-500.

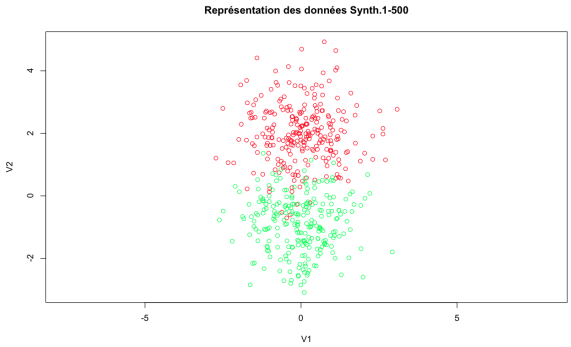

FIGURE 14 – Représentation des données Synth1-500.

5.6 Représentation des données Synth1-1000.

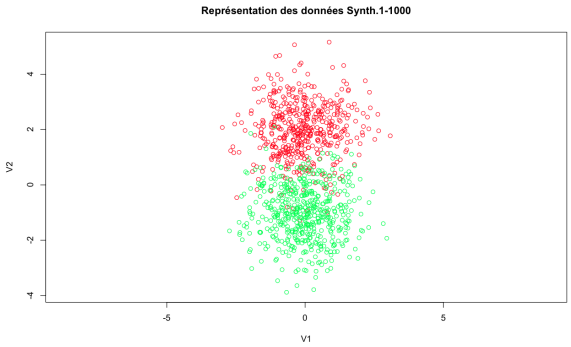

FIGURE 15 – Représentation des données Synth1-1000.

5.7 Représentation des données Synth2-1000.

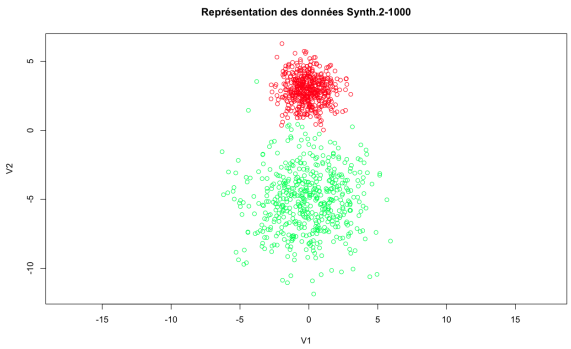

FIGURE 16 – Représentation des données Synth2-1000.