TP03 SY09 - Discrimination, théorie bayésienne de la décision

Laura BROS Alexis DUROCHER

Mai 2017

Table des matières

Chapitre 1

Introduction

Au cours des précédents TPs, nous avons pu découvrir différentes méthodes d'analyses de données de manière non supervisée. Nous avons commencé par nous familiariser avec les outils de statistique exploratoire qui permettent de prendre une première perspective sur nos données puis nous nous sommes penchés sur des méthodes plus complexes de représentation des données (ACP ou AFTD). Nous avons aussi utilisé des méthodes de classification automatique qui permettent d'organiser en classes homogènes des élements d'un ensemble Ω.
Désormais, nous allons nous pencher sur des méthodes d'apprentissage supervisé. Ces méthodes sont dites supervisées car la valeur de la variable Z, à "predire", est connue pour un ensemble d'individus d'apprentissage. L'objectif de ces méthodes est alors d'établir un modèle qui permettra d'identifier la valeur de la variable Z en fonction des valeurs prises par les variables explicatives. L'objectif de ce TP est donc d'appliquer des méthodes d'apprentissage supervisé. Nous utiliserons deux classifieurs différents : le classifieur euclidien et la méthode des K plus proches voisins. Nous aborderons ensuite la théorie bayésienne de la décision, qui nous aidera à trouver une règle de décision optimale.

Chapitre 2

Classifieur euclidien, K plus proches voisins

Dans ce premier chapître nous allons chercher à comparer les performances de deux méthodes d'apprentissage supervisé sur notre jeu de données *binaire*. On sait d'après l'énoncé, que l'ensemble des individus de ce jeu de données est défini sur $\mathbf{R}^2$ et répartit sur $g=2$ classes distinctes (ω_1 et ω_2). C'est à dire que, pour chaque individus x_i de cet ensemble, nous connaissons z_i, le label indiquant à quelle classe ce dernier appartient. Dans une première section, nous allons programmer des fonctions qui appliquent le classifieur euclidien et la méthode des K plus proches voisins sur ces données. Ensuite nous évaluerons les performances de leurs résultats. *Remarque : nous travaillerons sur des données décrites par un vecteur aléatoire $V = (X^1, X^2)$.*

2.1 Programmation

2.1.1 Classifieur euclidien

Le classifieur euclidien utilise le centre de gravité des nuages de points (différentes classes), pour calculer et comparer la distance (euclidienne ici) d'un nouvel individu avec les différents centres de gravité. La règle de décision du classifieur euclidien choisira, pour ce nouvel individu, la classe dont la distance calculée précedement sera la plus petite.

Nous allons donc programmer deux fonctions. La première, *ceuc.app*, prend en paramètre le tableau d'individus-variable de dimension n x p, et le vecteur *Zapp* des labels associés aux individus. Cette fonction déterminera en premier lieu, les centres de gravité (vecteurs de dimension p) en appliquant la fonction *mean* sur le sous-ensemble de *Xapp* des individus de chaque classe.
La seconde prendra en paramètre une matrice *Xtst*, d'individus de test (non labelisés) ainsi que la matrice *mu* (retournée par ceuc.app). Finalement cette seconde fonction ceuc.val calculera les distances entre individus de *Xtst* et centre de gravité de *mu* afin de déterminer, pour chaqun d'entre eux, la classe qui lui sera attribuée (distance minimum) . (cf Annexe Figure A.1)

2.1.2 K plus proches voisins

La méthode des K plus proches voisins base sa règle de décision sur l'étude des classes des K plus proches individus voisins (distance euclidienne ici) d'un nouvel individus x_i. La règle de décision choisira donc la classe majoritaire parmi ses voisins et l'attribuera à l'individus x_i. Ce sera le rôle de notre première fonction *kppv.val*. Remarquons qu'il n'y a pas d'apprentissage a proprement parlé dans cette méthode, l'enjeu est alors de déterminer le nombre K qui minimise au plus l'erreur de décision. Ce sera le rôle de notre seconde fonction *kppv.tune*.

kppv.val Elle prend en paramètre les données étiquetées (*Xapp* et *zapp*) ainsi que le K (que l'on choisira arbitrairement pour l'instant) puis les données non-labélisées *Xtst*. Le fonctionnement de *kppv.val* peut se résumer à trois étapes.
Premierement, elle calcule la matrice de distances entre les individus de *Xapp* et ceux de *Xtst* (distXY), ordonne ces dernieres par ordre croissant et utilise leur index matriciel pour les convertir en classe (ex : si dist[i,j] = 15, alors le i eme plus proche voisins de $Xtst_j$ est le 15ième indiv de Xapp, donc z = dist[Mat[i,j]]). Finalement la fonction récupère la valeur majoritaire de z en calculant la moyenne sur les valeurs des

K voisins.(cf Annexe Figure A.2)

kppv.tune Elle prend en paramètres les données étiquetées (*Xapp* et *zapp*) ainsi que les données de validation (*Xval* et *zval*). Aussi, un autre paramètre sera *nkppv* contenant les différentes valeurs de K à tester. Son fonctionnement se résume en deux étapes. Pour chaque valeur de k, la fonction utilise *kppv.val* sur les données d'apprentissage et de validation pour obtenir des labels prédits. Ensuite elle compare les valeurs obtenues avec les valeurs réelles de validation (*zval*). Ce taux d'erreur trouvé est alors comparé avec la valeur minimale actuelle pour conserver et finalement retourner le meilleur *k*.(cf Annexe Figure A.3)

2.1.3 Test des fonctions

Rappelons qu'un modèle d'apprentissage supervisé se base sur des données d'entrainement (ou apprentissage) et se teste sur des données de test. En effet, notre modèle ne peut pas être testé sur les données d'apprentissage car elles n'indiquent en rien que le modèle fonctionne pour d'autres données que celles sur lequel il s'est construit. Il est donc important de diviser notre jeu de données en différents sous ensemble pour entraîner notre modèle sur des données différentes de celles testées.
Nous utiliserons ici la fonction *separ1* pour déterminer nos différents *dataframes* pour le classifieur euclidien. Et nous utiliserons la fonction *separ2* pour la méthode des kppv. En effet cette dernière partitionne, en plus des ensembles d'apprentissage et de test, un jeu de données dis "de validation" depuis le jeu initial. Ce jeu de validation permettra à notre algorithme *kppv.tune* de trouver l'hyperparamètre K-optimale sans considérer les données de test. Ceci permet à notre estimation de la probabilité d'erreur de ne pas être optimiste (cas où la règle est appliquée sur la pop totale).

classifieur euclidien Afin de se faire une première idée du bon fonctionnement ou non du classifieur, nous afficherons la frontière de décision de ce classifieur euclidien qui utilise nos fonctions (matrice mu et fonction ceuc.val) et calcule l'hyperplan médiateur entre les centres de gravité de ω_1 et ω_2. Nous appliquerons ici la fonction *front.ceuc* sur nos données d'apprentissage *Xapp* et *zapp*.

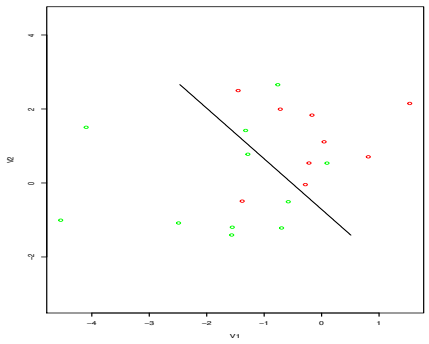


FIGURE 2.1 – Frontière de décision obtenus avec ceuc sur Synth1-40

Interprétation On remarque que la frontière de décision n'a pas discriminé tous les individus appartenant à ω_1 de ceux appartenant à ω_2. Ceci est du au fait que la frontière de décision du classifieur euclidien est linéaire et correspond à un hyperplan médiateur joignant les centres des deux classes. La précision du classifieur euclidien est limitée par sa frontière de décision. Ici nous voyons bien que l'hyperplan ne peut diviser parfaitement l'espace en deux classes homogènes. Le classifieur euclidien est efficace sur des ensembles dont les classes sont linéairement distinctes, de même volume et approximativement sphérique. Ici aux vus de la faible quantité du jeu de données, il est difficile de juger de la repartition et de la dispersion des données dans l'espace. Ces résultats sont donc à relativiser.

K plus proches voisins Pour tester le classfieur, nous allons utiliser la fonction kppv.tune sur nos jeux de données d'apprentissage et de validation et pour un vecteur de valeur de K pouvant variée de 1 à 10 (inutile d'en prendre plus car il n'y a que 20 individus dans Xapp). Nous pourrons ainsi récupérer la valeur de k

optimal (parmis celles proposées). A nouveau nous allons dessiner la frontière de décision obtenue via cette méthode supervisée. Nous appliquerons ici la fonction *front.kppv* sur nos données d'apprentissage *Xapp* et *zapp*.

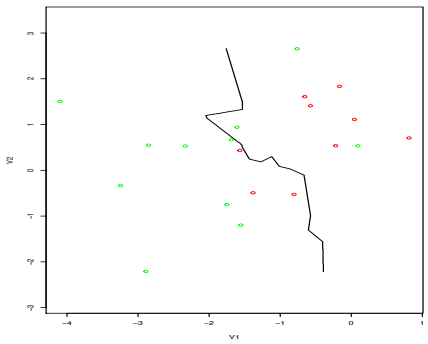


FIGURE 2.2 – Frontière de décision obtenus avec kppv sur sur Synth1-40 (discretisation = 1000)

Interprétation La figure 2.2 montre bien que la force de la méthode des kppv par rapport au classifieur euclidien est bien le fait que sa frontière de décision n'est pas linéaire (s'adapte mieux à la dispersion des données). Au contraire, la méthode des k-ppv base sa décision sur ses voisins locaux, ce qui entraîne une frontière de décision quasi locale à chaque individus et donc non linéaire.
Toutefois, la faible quantité du jeu de données (encore plus faible que celui utilisé pour le ceuc dû au données utilisées pour la validation) nous pousse à relativiser les résultats obtenus qui montrent ici une marge d'erreur significative.

2.2 Évaluation des performances

2.2.1 Jeux de données Synth1-*

Estimation des paramètres

Dans un premier lieu, nous allons simplement calculer les paramètres de chaque classe ω_k avec k=1 et k=2, μ_k (moyenne de classe), Σ_k (matrice de covariances) ainsi que π_k (proportion de classe) pour nos 4 jeux de données. Dans tous les cas, les classes ont été générées suivant une loi normale bivariée. Tous les jeux de données sont répartis de la même manière que celui sur lequel nous avons travaillé dans la section précédente (*binaire* : Synth1-40). Les données sont réparties sur deux classes homogènes ω_1 et ω_2. Nous allons donc calculer tous les paramètres pour ces deux classes.
Remarque : nous arrondirons au 3e décimale près

	Synth1-40	Synth1-100	Synth1-500	Synth1-1000
μ_1	$\begin{pmatrix}-0.316 & 1.092\end{pmatrix}$	$\begin{pmatrix}0.026 & 0.815\end{pmatrix}$	$\begin{pmatrix}0.132 & 0.880\end{pmatrix}$	$\begin{pmatrix}-0.013 & 0.916\end{pmatrix}$
μ_2	$\begin{pmatrix}-1.883 & 0.105\end{pmatrix}$	$\begin{pmatrix}-1.965 & -0.126\end{pmatrix}$	$\begin{pmatrix}-1.883 & -0.088\end{pmatrix}$	$\begin{pmatrix}-1.961 & 0.018\end{pmatrix}$
π_1	0.450	0.540	0.528	0.496
π_2	0.550	0.460	0.472	0.504
π_1 - π_2	**0.100**	**0.080**	**0.056**	**0.008**
Σ_1	$\begin{pmatrix}0.682 & 0.120\\0.120 & 1.013\end{pmatrix}$	$\begin{pmatrix}0.882 & -0.131\\-0.131 & 1.109\end{pmatrix}$	$\begin{pmatrix}1.050 & 0.052\\0.052 & 0.984\end{pmatrix}$	$\begin{pmatrix}0.969 & -0.065\\-0.065 & 1.079\end{pmatrix}$
Σ_2	$\begin{pmatrix}1.375 & 0.323\\0.323 & 1.440\end{pmatrix}$	$\begin{pmatrix}0.757 & -0.038\\-0.038 & 0.761\end{pmatrix}$	$\begin{pmatrix}0.967 & -0.111\\-0.111 & 0.979\end{pmatrix}$	$\begin{pmatrix}0.990 & 0.021\\0.021 & 0.941\end{pmatrix}$

Nous pouvons remarquer que plus l'ensemble de données étudié est grand, plus les paramètres de chaque classe tendent vers ces valeurs :

$$\pi_1 = \pi_2 = 0.5,\ \mu_1 = \begin{pmatrix}0 & 1\end{pmatrix},\ \mu_2 = \begin{pmatrix}-2 & 0\end{pmatrix},\ \Sigma_1 = \Sigma_2 = \begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$$

Distance euclidienne entre μ_1 et μ_2 est $\sqrt[2]{5}$.

Estimation des taux d'erreur de Classifieur euclidien

Grâce aux scripts réalisés, nous avons pu effectuer 20 séparations aléatoires. Pour chaque jeu de données, nous avons séparés un ensemble d'apprentissage et un ensemble de test afin de comparer les taux d'erreur entre les deux ensembles obtenus. En utilisant les résultats obtenus lors des 20 expériences, nous en avons ensuite déduit une estimation ponctuelle du taux d'erreur et un intervalle de confinace pour chacun des deux ensembles créés ci-dessus.
Ici, la taille de l'échantillon de nos taux d'erreur calculé est de n = 20, ce qui est un nombre assez grand pour appliquer le Théorème Central Limite, et donc de considérer que notre échantillon de ϵ suit une loi normale. Nous sommes donc dans le cas gaussien avec une variance inconnue, nous allons donc utiliser l'intervalle bilatéral de Student pour estimer l'intervalle de confiance. *Remarque, nous utilerons le fractile de student pour n = 19 et (1 - $\alpha/2$) = 0.975*

	Synth1-40	Synth1-100	Synth1-500	Synth1-1000
ϵ app	0.192	0.086	0.138	0.142
Intervalle Confiance App	[0.167 0.218]	[0.074 0.080]	[0.133 0.143]	[0.138 0.146]
ϵ test	0.269	0.097	0.122	0.144
Intervalle Confiance Test	[0.210 0.329]	[0.075 0.119]	[0.113 0.131]	[0.137 0.152]

Interprétation

Interprétons d'abord l'évolution en fonction de n. Ici, nous constatons qu'il est difficile de déterminer une tendance globale. En effet que ce soit pour l'ensemble de test ou d'apprentissage, les taux d'erreurs diminuent entre n = 40 et n = 100, mais augmentent pour n = 500 et n =1000. Toutefois, dans les deux cas, les IC du taux d'erreur de n = 40 est disjoint et bien supérieur à celui de n = 1000. On peut donc affirmer, à 0.95 près, que le taux d'erreur est plus faible pour n = 1000 que pour n = 40. Et ce pour les deux ensembles.
Comparons maintenant les performances des deux ensembles deux à deux. Il est impossible de dégager un gagnant (l'un dont le taux d'erreur serait le plus faible) car les IC se chevauchent pour toutes les valeurs de n. Le classifieur euclidien serait donc aussi performant quand il a été testé sur l'ensemble d'apprentissage que sur l'ensemble de test. Cela s'explique par le fait que notre échantillon d'individus reste fixe et le même pour les 20 séparations aléatoire. Cela implique que malgré le caractère aléatoire de notre séparation, les individus restent les même et se retrouvent parfoisdans l'ensemble de test et parfois dans l'ensemble d'apprentissage. Il n'y a donc pas de réel distinction entre ces deux ensembles. Par conséquent, les taux d'erreur sont sensiblement les mêmes. *Il aurait été éventuellement intéressant de régénérer de nouveaux individus d'app et de test plutôt que de piocher aléatoirement dans un ensemble fixe pour comparer leur performance respective.*

Détermination du nombre optimal de voisins (Kopt)

La fonction kppv.tune permet de déterminer le nombre K de voisins optimal, c'est à dire celui qui minimisera le taux d'erreur. Ici K 1 . . . 10. On utilise l'ensemble d'apprentissage comme ensemble de validation pour trouver le nombre de voisins optimal. Il en ressort les Kopt d'erreur suivants pour les différents jeux de données :

Synth1-40

K	1	2	3	4	5	6	7	8	9	10
ϵ	0	6	8.22	6.30	5.23	5.19	5.19	7.19	6.26	6.23

Synth1-100

K	1	2	3	4	5	6	7	8	9	10
ϵ	0	4	3.06	3.05	3.05	4.05	4.06	4.06	5.06	5.07

Synth1-500

K	1	2	3	4	5	6	7	8	9	10
ϵ	0	38	36.11	53.11	48.16	46.14	45.14	44.14	47.13	45.14

Synth1-1000

K	1	2	3	4	5	6	7	8	9	10
ϵ	0	66	62.10	82.10	81.12	81.12	86.12	89.13	86.13	86.13

Interprétation

Nous avons utilisé l'ensemble d'apprentissage comme ensemble de validation, le même que celui sur lequel le modèle a été formé, il est donc normal d'observer Kopt = 1. En effet, chaque individus de l'ensemble de validation possède lui-même comme plus proche voisin dans l'ensemble d'apprentissage car les mêmes ensembles ont été utilisés. Cet exemple illustre bien l'importance de prendre deux ensembles distincts pour l'apprentissage et la validation. Toutefois, nous allons regarder les résultats obtenus si nous ne considérons pas le Kopt = 1 comme un résultat valide, car trop dépendant de la répartition des données et des erreurs locales à chaques individus (bruit) et surtout ici de l'effet mirroir que suggère l'utilisation de mêmes ensembles.

Si on décide de raisonner sans le K = 1, on choisit alors la plus petite valeur de K qui minimise le taux d'erreur. Remarquons que nous étudions une classification supervisée sur 2 classes, donc pour éviter les cas d'ex-aequo, nous ne choisirons que les valeurs de K impaire. On trouve alors Kopt = 7 pour Synth1-40, Kopt = 5 pour Synth1-100, Kopt = 3 pour Synth1-500 et Kopt = 3 pour Synth1-1000.

Il en résulte que plus les données sont importantes, plus le nombres de voisins nécessaires pour minimiser le taux d'erreur semble diminuer. La valeur de Kopt semble converger vers 3 quand n grandit. Toutefois, ici, la valeur de Kopt est raisonnable considérant la taille du jeux de données (ici max n = 1000). En effet, 3 n'est ni trop grand (pour considérer trop de voisins et donc réduire la précision locale), ni trop petit (pour porter trop d'attention au bruit local et donc s'aveugler avec de faux voisins). Kopt = 3 est donc compréhensible pour un n = 1000. En revanche, on pourrait penser que pour un jeu de données 100 fois plus important (n = 100 000), Kopt = 3 soit trop petit au regard du bruit local qu'un individu pourrait avoir.

Estimation du taux d'erreur de K plus proches voisins

On considerera maintenant des valeurs du nombre de voisins différents de 1 pour la suite. Nous avons utilisé le même raisonnement que celui appliqué précédemment. Cependant, nous avons cette fois-ci effectué 20 séparations aléatoires de chaque jeu de données en ensembles d'apprentissage, de validation et de test. Nous utilisons ces résultats afin d'obtenir les estimations ponctuelles du taux d'erreur et les intervalles de confiance. Ces derniers seront calculés à partir des erreurs sur des estimations basées sur les ensembles d'apprentissage puis de test. (cf Table B.1, Annexe B)

Interprétation : (cf Table B.1, Annexe B)

Similairement au classifieur, il est difficile de tirer une interprétation de la tendance du taux d'erreur en fonction de n. Il suit la même evolution que celui du ceuc (diminue pour 100 puis augmente).

En comparant les performances des classifieurs deux à deux, on peut toutefois observer des différences de performance sur l'ensemble d'apprentissage mais pas réellement pour les estimations sur l'ensemble de test. En effet le taux d'erreur sur l'ensemble de test de kppv semble tendre vers la même valeur que le class. euclident (vers 15% de taux d'erreur). Sur l'ensemble d'apprentissage kppv semble meilleur tant que n < 1000 (les IC ne chevauchent pas et sont plus faible que ceux du class. euclidien pour les même valeur de n) et sa tendance semble tendre vers 13.

2.2.2 Jeu de données Synth2-1000

Afin de comparer sur un jeu de données dont la répartition des individus est différente, nous allons réitérer l'opération d'estimation de paramètres et de taux d'erreur sur nos deux classi-

fieurs sur le jeux de données Synth2-1000.

Estimation des paramètres

	Synth2-1000
μ_1	$\begin{pmatrix}3.018 & -0.006\end{pmatrix}$
μ_2	$\begin{pmatrix}-2.142 & -0.027\end{pmatrix}$
π_1	0.523
π_2	0.477
π_1 - π_2	**-0.028**
Σ_1	$\begin{pmatrix}0.990 & 0.113\\ 0.113 & 0.092\end{pmatrix}$
Σ_2	$\begin{pmatrix}4.435 & -0.154\\ -0.154 & 1.034\end{pmatrix}$

Interprétation
En observant les résultats, on peut alors supposer que le jeu de données Synth2-1000 à été repartis selon une loi bivariée avec les paramètres suivants :

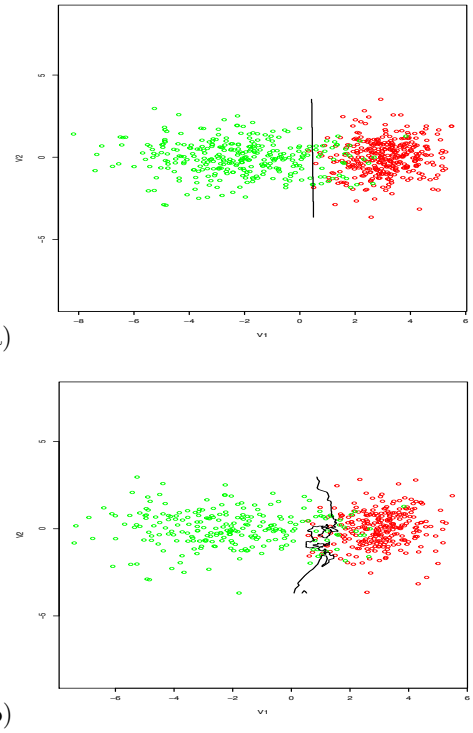
$$\begin{cases}\pi_1 = \pi_2 = 0.5,\ \mu_1 = \begin{pmatrix}3 & 0\end{pmatrix},\ \mu_2 = \begin{pmatrix}-2 & 0\end{pmatrix},\\ \Sigma_1 = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix},\ \Sigma_2 = \begin{pmatrix}5 & 0\\ 0 & 1\end{pmatrix}\end{cases}$$

Nous constatons donc que, à part pour la probabilité des classes, les paramètres sont différents de ceux de la loi utilisée dans Synth1-n. En effet, dans le cas de Synth2, les matrices de variances sont différentes (Σ_1 != Σ_2) et les vecteurs μ_1 et μ_2 ont une distance euclidienne de 5 qui est nettement plus importante que celle estimée pour Synth1-n ($\sqrt[2]{5}$). Intuitivement, nous pouvons supposer que les deux classes d'individus n1 et n2, sont plus distinctes (en terme de postion dans l'espace et de répartition) que dans Synth1-n. Observons les conséquences sur la performance des classifieurs.

Estimation taux d'erreur

Estimation du taux d'erreur et son intervalle de confiance en considérant les estimations faîtes sur l'ensemble d'Apprentissage d'abord, puis sur l'ensemble de Test.

	Synth2-1000 ceuc	Synth2-1000 kppv
ϵ app	0.062	0.050
IC App	[0.061 0.064]	[0.048 0.053]
ϵ test	0.060	0.074
IC Test	[0.054 0.065]	[0.022 0.125]



(a) (b)

FIGURE 2.3 – Frontières de décisions obtenues avec ceuc (a) et kppv (b)

Interprétation
On remarque que les taux d'erreur sur les ensembles d'apprentissage sont approximativement égaux pour les deux classifieurs. On peut également observer que ce taux d'erreur est bien inférieur à ceux calculés pour les jeux de données précédents. On peut conclure, grâce à l'apprentissage tiré de l'étude des différents jeux de données Synth*, que les paramètres jouent un rôle important dans l'estimation des taux d'erreur. Ici, on peut supposer que le fait que les centres de gravités des nuages soient plus éloignés dans Synth2 que dans Synth1, implique-

rait une meilleur distinction des classes homogènes et donc un taux d'erreur pour les deux classifieurs, plus faible. D'après la figure 2.3, on remarque bien la très forte dispersion des individus vert (n2). Ainsi, malgré des centres de gravités ecartés, la forte variance du nuage n2 entraîne un taux significatif d'erreur de seconde espèce (C'est encore plus visible pour le classifieur euclidien où l'on a trouvé une erreur de seconde espèce = 13%).

2.2.3 Jeux de données réelles

Après avoir utilisé nos classifieurs et évaluer leurs performances sur des jeux de données dont le "sens intrinsèque" des individus et leurs valeurs explicatives nous étaient inconnus (z = 1, n'avait pas plus de sens que z = 2). Nous allons maintenant observer l'efficacité de ces deux méthodes supervisées sur des jeux de données dîtes réelles (dont le sens des variables explicatives et du label des classes est connu).
Les jeux de données mis alors à notre disposition seront Pima, dont le 'z' correspond à "l'individu est diabétique (2) ou sain (1)", et Breast-cancer, dont le 'z' correspondra alors à "l'individu est atteint d'un cancer de la poitrine ou non"

Estimation taux d'erreur

Dans cette section nous allons étudier le taux d'erreur global comme dans les sections précédentes mais nous allons porter une attention particulière à l'erreur de seconde espèce β. En effet, nous sommes ici en présence de données médicales donc l'erreur de seconde espèce (classé un individus malade comme sain) est à minimiser en priorité. Cette erreur sera calculé par la somme des individus de test classé comme appartenant à ω_1 alors qu'ils appartiennent à ω_2 divisé par le nombre n2 d'individus total de ω_2 .

Interprétation (cf Table B.2, Annexe B)
Premier constat, les taux d'erreur globaux ϵ sur ensemble d'apprentissage et de test et pour les deux classifieurs sont nettement plus elevés sur le jeux de données Pima que sur celui de Breast-cancer. On sait d'après le TP01, que certaines variables du jeux de données Pima sont inutiles / trop correlées pour expliquer correctement le label Z. Afin d'avoir une idée plus juste des performances, il faudrait ici, chercher les variables réellement explicatives et réitérer les calculs.

Concernant Pima, le classifieur euclidien semble bien moins performant que le kppv à 0.95 près (les IC ne se chevauchent pas et celui de ceuc est nettement supérieur à celui de kppv). Un autre indicateur est que l'erreur de seconde espèce β est amplifier avec le class. euclidien alors qu'elle est diminuée avec les kppv. Dans un contexte médical et sans nettoyage de variables explicative, nous retiendrons donc plus le kppv comme classifieur pour les données Pima.

Concernant Breast-cancer, il est difficile de dégager un meilleur classifieur, car les IC des taux d'erreurs sur tests se chevauchent pour ceuc et kppv. Nous trancherons donc avec l'erreur de seconde espèce β. En effet, comme sur Pima, β est nettement amplifiée avec le classif. euclid et amplifiée que très légerement pour le kppv. Dans un contexte médical, nous retiendrons donc à nouveau le kppv comme classifieur pour les données Breast-cancer.

Chapitre 3

Règle de Bayes

Introduction Nous savons que les individus n1 de la classe ω_1 ont été générés selon une loi normale bivariée de paramètres μ_1 et Σ_1. Parallèlement, les individus n2 de la classe μ_2 ont été générés selon la même loi mais de paramètres μ_2 et Σ_2.

Commencons cette section par observer la répartition des individus en considérant les variables descriptives séparemment (selon X^1 puis selon X^2, cf figure 3.1) puis en considérant l'espace bi-dimensionnelle au complet (cf figure 3.2).
Les matrices de covariances étant diagonales, nous savons que X^1 et X^2 sont indépendantes. Nous ne perdons donc pas de sens en considérant les variables marginalement.

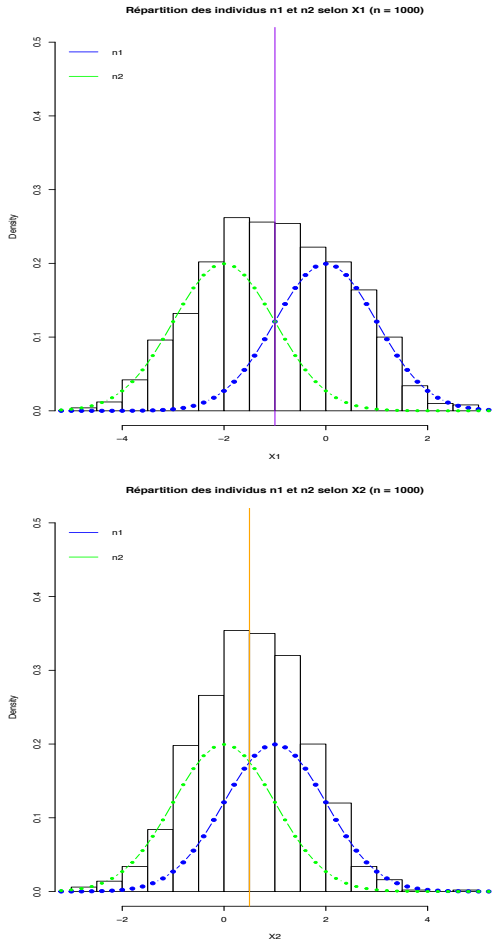


FIGURE 3.1 – Répartition des n1 et n2 selon X^1 puis selon X^2 pour n = 1000

Interprétation
Figure 3.1 : Pour chaque variable explicative, nous constatons que les zones partagées entre les deux courbes de densité sont très large. En particulier, selon l'axe X^2, la zone partagée est si large (frontière ligne jaune très proche des maximum de densité de chaque échantillons) que leur distribution selon cet axe X^2 pourrait presque être confondue avec une et une seule distribution d'échantillon appartenant à la même classe. Cette très faible distinction des classes expliquerait le fort taux d'erreur obtenu par nos classifieurs sur les jeux de données Synth1-n dans les sections précédentes.

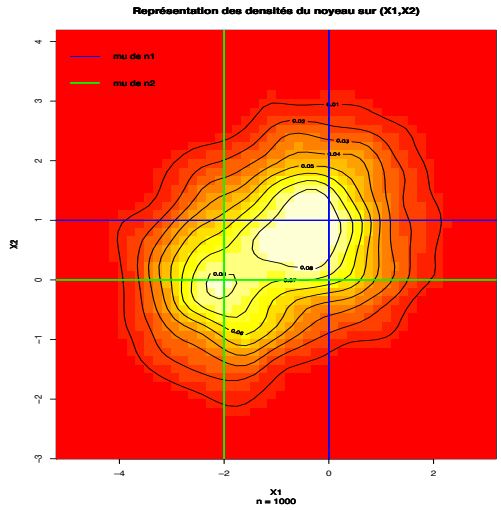


FIGURE 3.2 – Repartitions des n1 et n2 dans (X^1 ,X^2)

Interprétation
Figure 3.2 : La méthode de représentation de notre loi multivariée utilisée ici est l'estimation par noyau (ou encore méthode de Parzen-Rosenblatt). C'est une méthode non-paramétrique d'estimation de la densité de probabilité. Nous la baserons sur nos echantillons d'individus n1 et n2 afin d'estimer la variation des densité de nos échantillons n1 et n2 en tout point du support dans l'espace (X^1, X^2) (couleur rouge : faible densité sous le noyaux, blanche : densité elevée). Numériquement, nous observons que l'espace elliptique dont le contour est définie par la courbe d'iso-densité égale à 0.07, regroupe déjà les deux "ellipses" centrée sur les vecteurs moyennes (μ_1 en bleu et μ_2 en vert). L'idée générale qui se dégage de cette représentation est cohérente avec l'interprétation de la figure 3.1 : Les deux echantillons n1 et n2 sont très confondus et ne peuvent que difficilement être distingués en classes homogènes distinctes dans notre espace caractéristique (X^1 ,X^2)

Dans cette section nous allons appliquer la théorie Bayesienne de la décision, qui est enfait une stratégie pour trouver la règle de décision optimale : celle qui minimise le taux d'erreur. En effet, comme observée ci-dessus et empiriquement dans les sections précédente, nos individus sont répartis de manière difficilement distinguables (en classes homogènes) et sont donc prône aux erreurs de décision. Nous allons voir comment minimiser ces erreurs.

distribution marginale Considérer une distribution marginale de X^i pour les individus d'une classe revient à ne prendre en compte que cette variable aléatoire et d'estimer sa fonction de densité. Ici, une solution serait d'intégrer la fonction de densité (cf. equation) de notre vecteur aléatoire V = (X^1, X^2) en marginalisant les variables l'une après l'autre.

$$f(x) = \frac{1}{(2\pi)^{N/2}\,|\Sigma|^{1/2}}\;\; e^{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)} \tag{3.1}$$

Cependant, ici nous sommes dans le cas gaussien multivariée donc ces calculs ne seront pas nécessaires. En effet, une propriété importante d'une loi normale multivariée est que toute sous-partie ou composante d'un vecteur aléatoire gaussien, suit aussi une loi normale. Nous pouvons donc inférer des paramètres de notre loi normale multivariée, les paramètres des lois normales suivis par nos variables aléatoires marginales (X^1 et X^2). Nous obtenons ainsi :

$$\begin{cases} X_1^1 \sim \mathcal{N}(\mu_1^1,\, \Sigma_{1(1,1)}) \sim \mathcal{N}(0,\, 1)\,. \\ X_2^1 \sim \mathcal{N}(\mu_2^1,\, \Sigma_{2(1,1)}) \sim \mathcal{N}(-2,\, 1)\,. \\ X_1^2 \sim \mathcal{N}(\mu_1^2,\, \Sigma_{1(2,2)}) \sim \mathcal{N}(1,\, 1)\,. \\ X_2^2 \sim \mathcal{N}(\mu_2^2,\, \Sigma_{2(2,2)}) \sim \mathcal{N}(0,\, 1)\,. \end{cases}$$

Les courbes gaussiennes sont disponibles sur la figure figure 3.1.

Courbe d'iso-densité Une courbe d'iso-densité est une courbe pour laquelle la densité de probabilité des individus est égale à une constance c dans l'espace considéré. L'équation de f(x), fonction de densité de probabilité est donnée dans l'équation 3.1. Nous cherchons à trouver l'équation f(x) = c, qui revient à : $(x-\mu)^T\Sigma^{-1}(x-\mu) = c$ avec c une constante réelle.
On peut donc poser :

$$\begin{cases} \omega_1 : (\text{x - } \mu_1)^T \Sigma_1^{-1} (x - \mu_1) = c \\ \omega_2 : (\text{x - } \mu_2)^T \Sigma_2^{-1} (x - \mu_2) = c \end{cases}$$

En utilisant les paramètres de *Synth1*, on retrouve :

Classe 1 :
- $x_1^2 + (x_2 - 1)^2 = c$ –> on obtient donc l'équation d'une ellipsoïde cercle de centre μ_1 et de rayon $\sqrt{c}$.

Classe 2 :
- $(x_1 + 2)^2 + x_2^2 = c$ –> on obtient donc l'équation d'un cercle de centre μ_2 et de rayon $\sqrt{c}$.

Parallelement, en utilisant les paramètres de *Synth2*, on retrouve :

Classe 1 :
- $(x_1 - 3)^2 + x_2^2 = c$ cercle de centre μ_1 et de rayon $\sqrt{c}$.

Classe 2 :
- $(x_1+2)^2+5x_2^2 = 10c \Leftrightarrow \frac{(x_1+2)^2}{10c} + \frac{x_2^2}{2c} = 1$ ellispe de grand rayon $\sqrt{10c}$, de petit rayon $\sqrt{2c}$ et de centre $(2\,;\,0)$.

On remarque que lorsque la matrice Σ_k est diagonale, les courbes d'isodensité calculées correspondent à des ellipsoïdes de centre μ_k.

Règle de Bayes - Synth1 La règle de Bayes correspond à une règle de décision qui choisit la classe d'un individus x en fonction de la valeur des probabilités conditionnelles de w1 sachant x ou de w2 sachant x. C'est une règle de décision minimisant la probabilité d'erreur. Ici, la matrice de variance est commune à toutes les classes. Nous sommes donc dans l'hypothèse d'homoscédasticité et on choisira donc (Analyse Discriminante Linéaire ?)

$$f_k(x) = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}} \; e^{-\frac{1}{2}((x-\mu_k)^\top \Sigma^{-1}(x-\mu_k))}$$

avec k = 1 ou 2 et p = 2. Rappelons que f(x) la densité de mélange est donnée par

$$f(x) = \pi_1 f_1(x) + \pi_2 f_2(x)$$

En utilisant les paramètres de *Synth1*, on retrouve :

$$\begin{cases} f_1(x) = \dfrac{1}{(2\pi)} \; e^{-\frac{1}{2}(x_1^2 + (x_2-1)^2)} \\ f_2(x) = \dfrac{1}{(2\pi)} \; e^{-\frac{1}{2}((x_1+2)^2 + x_2^2)} \end{cases}$$

La règle de Bayes peut donc s'écrire :

$$\begin{cases} \delta^*(x) = \omega_1 \Leftrightarrow P(\omega_1|x) > P(\omega_2|x) \\ \Leftrightarrow \frac{\pi_1 f_1(x)}{f(x)} > \frac{\pi_2 f_2(x)}{f(x)} \Leftrightarrow \frac{f_1(x)}{f_2(x)} > \frac{\pi_2}{\pi_1} \\ \Leftrightarrow ln(e^{-\frac{1}{2}(-4x_1 - 2x_2 - 3)}) > ln(\frac{\pi_2}{\pi_1}) \\ \Leftrightarrow 2x_1 + x_2 - \frac{3}{2} > 0 \end{cases}$$

On peut donc alors en déduire la règle de Bayes de décision suivante :

$$\delta^*(x) = \begin{cases} \omega_1 & \text{si } 2\text{x}_1 + x_2 > k, \text{ avec } k = -\frac{3}{2} \\ \omega_2 & \text{sinon,} \end{cases} \tag{3.2}$$

Voici donc, dans le plan formé par les variables X^1 et X^2, la frontière correspondante d'équation : $x_2 = -2x_1 - \frac{3}{2}$:

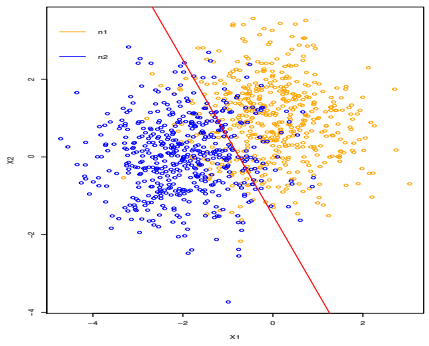


FIGURE 3.3 – Frontière de décision | règle de Bayes | Synth1-1000

Interprétation

Nous constatons que la frontière est une fonction linéaire de x_1 et x_2. Elle distingue les deux nuages n_1 et n_2, mais leur forte intersection implique toutefois un taux d'erreur significatif. En effet de nombreux individus seront mal labélisés par cette règle de décision.

Règle de Bayes - Synth2 Nous reprenons la formule de la règle de Bayes :

$$\delta(x) = \omega_1 \Leftrightarrow \frac{f_1(x)}{f_2(x)} > \frac{\pi_2}{\pi_1}$$

Toutefois, les Σ_k étant différent dans Synth2, nous ne sommes plus dans l'hypotèse d'homoscédasticité :

$$f_k(x) = \frac{1}{(2\pi)^{p/2}\left|\Sigma_k\right|^{1/2}} \quad e^{-\frac{1}{2}((x-\mu_k)^\top \Sigma_k^{-1}(x-\mu_k))}$$

En utilisant les paramètres de *Synth2*, on retrouve :

$$\begin{cases} f_1(x) = \dfrac{1}{(2\pi)} \quad e^{-\frac{1}{2}((x_1-3)^2+x_2^2)} \\ \\ f_2(x) = \dfrac{1}{(2\pi)\sqrt{5}} \quad e^{-\frac{1}{2}(5(x_1+2)^2+x_2^2)} \end{cases}$$

La règle de Bayes peut donc s'écrire :

$$\begin{cases} \delta^*(x) = \omega_1 \Leftrightarrow P(\omega_1|x) > P(\omega_2|x) \\ \\ \Leftrightarrow \frac{\pi_1 f_1(x)}{f(x)} > \frac{\pi_2 f_2(x)}{f(x)} \Leftrightarrow \frac{f_1(x)}{f_2(x)} > \frac{\pi_2}{\pi_1} \\ \\ \Leftrightarrow ln(\sqrt{5}e^{-\frac{1}{2}(\frac{4}{5}x_1^2 - \frac{34}{5}x_1 + \frac{41}{5})}) > ln(\frac{\pi_2}{\pi_1}) \\ \\ \Leftrightarrow -4x_1^2 + 34x_1 > 41 - 10ln(\sqrt{5}) \end{cases}$$

On peut donc alors en déduire la règle de Bayes de décision suivante, avec $k_1 = 41$ et $k_2 = -10ln(\sqrt{5})$:

$$\delta^*(x) = \begin{cases} \omega_1 & \text{si -4x}_1^2 + 34x_1 > \text{k1} + \text{k2} \\ \omega_2 & \text{sinon,} \end{cases} \tag{3.3}$$

Voici donc, dans le plan formé par les variables X^1 et X^2, la frontière correspondante d'équation $-4x_1^2 + 34x_1 = 41 - 10ln(\sqrt{5})$:

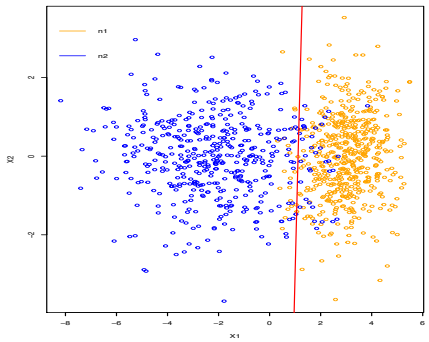


FIGURE 3.4 – Frontière de décision | règle de Bayes | Synth2-1000

Interprétation Nous constatons que la variable X^2 n'a pas d'influence sur la frontière de la règle de Bayes sur Synth2. Cela s'explique par le fait que les coordonnées sur X^2, des vecteurs μ_k sont nulles (placées sur l'origine). En revanche, nous avons bien une frontière dont l'équation est un polynôme du second degré, cohérente avec le fait que les matrice de covariances des échantillons n1 et n2 soient différentes.

Probabilité d'erreur de Bayes La probabilité d'erreur $\epsilon(\delta^*)$ de la règle de Bayes est appelée probabilité d'erreur de Bayès. On la note ϵ^*.
Elle représente la plus petite erreur possible que peut atteindre une règle de décision utilisant uniquement la variable explicative. Dans les situations où l'étude porte sur deux classes, la probabilité d'erreur de Bayès se note :

$\epsilon^* =$

$$\int min(P(\omega_1|x), P(\omega_2|x))f(x)dx$$

Synth1-n

Expression Formelle de l'erreur de Bayès
Pour les échantillons de données Synth1-n, nous avons g= 2 ainsi que

$$\Sigma_k = \Sigma$$

Ainsi, il est possible de calculer exactement la probabilité d'erreur de Bayes en se plaçant avec l'hypothèse d'homoscédasticité. De plus, nous avons $\pi_1 = \pi_2$, on peut donc exprimer ϵ^*. de la manière suivante : $\epsilon^*. = \phi(-\Delta/2)$
avec ϕ la fonction de répartition de la loi normale univariée centrée réduite.

La quantite Δ^2 est appelée carré de la distance de Mahalanobis entre les deux classes. Elle est définie de la manière suivante :
$\Delta^2 = (\mu_2 - \mu_1)^T \Sigma^{-1} (\mu_2 - \mu_1)$

Après calcul on obtient donc $\Delta^2 = 5$. On cherche donc la valeur de ϕ pour $-\Delta/2 = -1.58$ On trouve alors $\epsilon^* = 0.13$ comme erreur de Bayès.

Synth2-1000
Dans le cas plus général où les variances sont différentes, même en se limitant à deux classes, il n'est pas possible d'exprimer analytiquement l'erreur de Bayes. On utilise alors des approximations grâce à la borne de Bhattacharyya. L'objectif est d'obtenir une borne supérieur de l'erreur de Bayes définie comme suit :

$$\epsilon^* \leq \sqrt{\pi_1 \pi_2} \int \sqrt{f_1(x) f_2(x)} dx = \sqrt{\pi_1 \pi_2} \exp^{-\Delta_B^2}$$

La quantité Δ_B^2 est appelée carré de la distance de Bhattacharyya entre les deux classes. Dans le cas gaussien elle est définie par : $\Delta_B^2 = \frac{1}{8}(\mu_2 - \mu_1)^T (\frac{\Sigma_1 + \Sigma_2}{2})^{-1} (\mu_2 - \mu_1) + \frac{1}{2} ln \frac{det \frac{\Sigma_1 + \Sigma_2}{2}}{\sqrt{det \Sigma_1 det \Sigma_2}}$

On trouve, en remplaçant par les paramètres trouvés à la question précédente : $\Delta_B^2 = 0.84$

Enfin en remplaçant dans la formule, on obtient une probabilité d'erreur de Bayès égale à $\epsilon^* < 0.22$

Interprétation En comparant avec les résultats obtenus précédemment, nous remarquons que la probabilité d'erreur de Bayès est significativement meilleure sur Synth1-n qu'avec les deux autres classifieurs. En effet alors que le taux d'erreur tendait vers 14/15 ‰ pour le classifieur euclidien et vers 15 pour le kppv (en considérant les estimations faîtes sur l'ensemble de test pour ne pas suggérer de cas optimiste), nous obtenons, avec Bayes, une erreur de 13. Toutefois pour Synth2, Bayes de nous propose qu'une borne supérieur qui, bien que cohérente avec les résultats précédents (la borne est bien supérieure aux taux calculées avec ceuc et kppv), ne nous propose pas de suggestion optimale.

Chapitre 4

Conclusion

Au cours de ce troisième TP, nous avons eu l'occasion de nous familiariser avec des méthodes supervisées de classification. Nous avons, en effet, étudié les méthodes du classifieurs euclidien ainsi que celle des K plus proches voisins. La partie programmation sur R, nous a permi d'en comprendre les rouages algorithmiques. A travers la section suivante nous avons évalué leur performances respectives en se basant sur leurs taux d'erreur lorsqu'appliqués sur des jeux de données dont la quantité d'individus ou les paramètres de générations d'échantillons différaient.
Nous nous sommes alors rendus compte à quel point les performances d'un classifieur étaient dépendantes du jeux de données sur lequel il est appliqué (fonction des paramètres, du volume de données et même de leur sens intrinsèque selon les cas). Finalement, nous avons étudié les méthodes supervisées sous un angle encore plus théorique en appliquant la règle de Bayes et en calculant les probabilités d'erreur associées sur nos mêmes jeux de données. Nous avons pu constater que les performances de la règle de décision de Bayes sont, elles aussi, dépendantes du jeux de données sur lequel elle est appliquée et que ses conclusions ne sont pas très précises lorsque les nuages de points ont une dispersion différente dans l'espace (matrice de covariances différentes).

Annexe A

```r
###CEUC.APP###
ceuc.app <- function(Xapp, zapp)
{
  #On part du tableau individus-variables Xapp.
  #On applique pour chaque colonne (chaque attribut) la fonction moyenne en distinguant les
  #deux classes differentes renseignees dans le vecteur zapp.
  apply(Xapp, 2, by, zapp, mean)
}


###CEUC.VAL###
ceuc.val <- function(mu, Xtst)
{
  # Détermine les distances centre de grav / individus test
  a = distXY(mu,Xtst)
  # garde le centre de grav le plus proche
  b = apply(a,2,which.min)
  b
}
```

FIGURE A.1 – Fonctions ceuc.app et ceuc.val

```r
###KPPV.VAL###
kppv.val <- function(Xapp,zapp,K,Xtst)
{
  # matrice de toutes distances entre indiv de App et Test
  a = distXY(as.matrix(Xapp),as.matrix(Xtst))
  tmp = apply(a,2,order)
  dist = tmp
  for (i in 1:ncol(tmp)){
    for (j in 1:nrow(tmp)){
      # remplace distances par classes correspondantes
      dist[j,i]=zapp[tmp[j,i]]
    }
  }
  # on ne cherche que les K meilleures distances (devenues classes)
  dist <- dist[1:K,]
  if (K>1){
    # round renverra la classe majoritaire parmis les K
    # (en arrondissant la moyenne des K classes voisines de Xtst_j)
    dist = round(apply(dist,2,mean))
  }
  return (dist)
}
```

FIGURE A.2 – Fonction kppv.val

```r
###KPPV.TUNE###
kppv.tune <- function(Xapp,zapp,Xval,zval,nppv)
{
  min = 1
  tmp = kppv.val(Xapp,zapp,1,Xval)
  tmp = as.vector(tmp)
  erreur = 0
  # recupere l'erreur de reference (Kopt = 1)
  for (j in 1:length(tmp)){
    if (tmp[j]!=zval[j]){
      erreur = erreur +1
    }
  }
  erreur = erreur / length(zval)
  # calcul erreur avec autre valeur de k
  erreur_min <- erreur
  for (i in nppv){
    tmp = c()
    tmp = kppv.val(Xapp,zapp,i,Xval)
    tmp = as.vector(tmp)
    for (j in 1:length(tmp)){
      if (tmp[j]!=zval[j]){
        erreur = erreur +1
      }
    }
    erreur = erreur / length(zval)
    print(erreur)
    # conserve erreur min et valeur de k opt
    if(erreur <= erreur_min){
      erreur_min <- erreur
      min <- i
    }
  }
  return (min)
}
```

FIGURE A.3 – Fonction kppv.tune

Annexe B

	Synth1-40	Synth1-100	Synth1-500	Synth1-1000
ϵ app	0.158	0.059	0.122	0.132
IC App	[0.156 0.159]	[0.058 0.060]	[0.122 0.122]	[0.132 0.132]
ϵ test	0.179	0.087	0.145	0.153
IC Test	[0.176 0.181]	[0.084 0.089]	[0.143 0.147]	[0.151 0.155]

TABLE B.1 – Estimation du taux d'erreur de kppv sur Synth1-n

	Pima ceuc	Breast-Cancer ceuc	Pima kppv	Breast-Cancer kppv
ϵ app	0.25	0.039	0.190	0.025
Intervalle Confiance App	[0.245 0.255]	[0.037 0.041]	[0.173 0.207]	[0.024 0.025]
ϵ test	0.242	0.039	0.211	0.048
Intervalle Confiance Test	[0.232 0.251]	[0.034 0.043]	[0.162 0.261]	[-0.004 0.099]
β Tst	0.313	0.075	0.202	0.049

TABLE B.2 – Estimation du taux d'erreur de kppv et ceuc sur Pima et Breast-cancer