SY09 - TP3 Discrimination & Théorie Bayésienne de la décision

Stéphane LOUIS et Paul GOUJON

Mai 2016

Table des matières

Chapitre 1

Introduction

1.1 Avant propos

Contenu Ce document présente les résultats obtenus par notre binôme au cours de ce troisième TP de SY09.

1.2 Code & Résultats : Annexes

Annexes Vous trouverez en annexes de ce rapport les fichiers suivants.

1. **ceuc.R** : Les fonctions concernant le classifieur euclidien.
2. **kppv.R** : Les fonctions concernant les k plus proches voisins.
3. **TP3.R** : Le script que nous avons utilisé au long de la partie évaluation des performances.
4. **Dossier "results"** : Les exports en `.csv` de nos résultats.

1.3 Objectifs du TP

Objectifs Les objectifs de ce TP sont multiple. Le principal d'entre eux est de commencer à nous familiariser avec les concepts d'apprentissage supervisé abordés en cette fin de semestre en SY09. Nous aborderons différents types de classifieur, nous permettant d'effectuer des tâches de discrimination entre individus, comme la classifieur euclidien, la méthode des k plus proches voisins, ou encore la règle de Bayes et appliquerons ces différents concepts notamment en codant les principales fonctions d'apprentissage et de test de ces derniers en `R`.

Chapitre 2

Programmation

2.1 Introduction

Objectif Notre objectif est dans un premier temps de coder sous `R` deux algorithmes de discrimination : le classifieur euclidien (euclidian classifier) et l'algorithme des k plus proches voisins ou "kppv" (k nearest neighbours, "knn").

2.2 Classifieur euclidien

Introduction Détaillons au sein de cette section les différents algorithmes que nous avons décidé d'appliquer.

NB : Le code est également commenté.

Algorithme ceuc.app Ci-dessous l'algorithme utilisé pour l'apprentissage du classifieur euclidien.

input : $Xapp$: matrice $n \times p$ d'individus, $zapp$: vecteur d'étiquettes des n individus
output: μ : matrice $k \times p$ des paramètres du classifieur euclidien, correspondant aux centres de gravités des nuages de points des différentes classes

k = nombre de classes différentes;
μ = matrice $k \times p$ (p étant la dimension de chaque individu);
for $i \leftarrow$ *1* **to** k **do**
  $\omega_i \leftarrow i$ème classe (parmi le vecteur d'uniques classes);
  $X_i \leftarrow$individus de $Xapp$ tels que $\omega(x_i) = \omega_i$;
  $\mu_i \leftarrow mean(Xi)$ par colone;
end
return μ

Algorithm 1: Algorithme : ceuc.app

Algorithme ceuc.val Ci-dessous l'algorithme utilisé pour la classification de n individus de test à partir des paramètres μ_i déterminés au cours de l'apprentissage.

input : μ : la matrice $k \times p$ de paramètre renvoyée par la fonction d'apprentissage, $Xtst$: la matrice $n \times p$ d'individus de test
output: $Ntst$: vecteur de taille n des étiquettes des individus de test

$n \leftarrow$ nombre d'individus dans l'échantillon de test;
$Ntst \leftarrow$ vecteur de taille n;
for $i \leftarrow$ *1* **to** n **do**
  $decision \leftarrow (\bar{x_2} - \bar{x_1})^T (x - \frac{\bar{x_1} + \bar{x_2}}{2})$;
  if $decision \leq 0$ **then**
    $Ntst[i] \leftarrow 1$;
  else
    $Ntst[i] \leftarrow 2$;
  end
end
return $Ntst$

Algorithm 2: Algorithme : ceuc.val

2.3 k Plus Proches Voisins

Introduction Idem, détaillons au sein de cette section les différents algorithmes que nous avons décidé d'appliquer.

NB : Le code est également commenté.

Algorithme kppv.app Ci-dessous l'algorithme utilisé pour la classification d'un ensemble de test, à l'aide d'un ensemble d'apprentissage, en choisissant de prendre en considération k voisins.

input : $Xapp$: matrice $m \times p$ d'individus, $zapp$: vecteur d'étiquettes des m individus, K : le nombre de voisins à prendre en considération, $Xtst$ un ensemble de n individus à classifier
output: $ztst$: vecteur de taille n contenant les étiquettes attribuées aux individus de test

$n \leftarrow$ nombre d'individus à classer;
$p \leftarrow$ dimension de chaque individu;
$nbclass \leftarrow$ nombre de classes différentes;
$ztst \leftarrow$ vecteur de taille n;

for $i \leftarrow 1$ **to** n **do**
  $zknn \leftarrow$ vecteur de classes des K individus de l'ensemble $Xapp$ les plus proches de l'individu $Xtst[i,]$;
  $scores\_classes \leftarrow$ table de contingence (castée en data frame) tirée du vecteur zknn, afin de connaitre le nombre d'apparition de chaque classe dans les K voisins les plus proches de l'individu de test examiné;
  $max\_score \leftarrow$ nombre maximum d'apparition d'une même classe parmi les voisins de l'individu de test examiné;
  $new\_k \leftarrow nbclass$;
  while *Nombre de classes ayant un score égal à* $max\_score > 1$ **do**
    $new\_k \leftarrow new\_k + 1$;
    $zknn \leftarrow$ vecteur de classes des new_k individus de l'ensemble $Xapp$ les plus proches de l'individu $Xtst[i,]$ (new_k nous sert à prendre en compte un individu en plus tant qu'il y a égalité) ;
    $scores\_classes \leftarrow$ table de contingence (castée en data frame) tirée du vecteur zknn, afin de connaitre le nombre d'apparition de chaque classe dans les new_k voisins les plus proches de l'individu de test examiné;
    $max\_score \leftarrow$ nombre maximum d'apparition d'une même classe parmi les voisins de l'individu de test examiné;
  end
  $ztst[i] \leftarrow$ classe dont le score dans $scores\_classes$ est maximum
end
return $ztst$

Algorithm 3: Algorithme : kppv.app

Algorithme kppv.tune Ci-dessous l'algorithme utilisé pour la détermination du nombre K optimal de voisins à prendre en compte pour la classification en utilisant la méthode des k plus proches voisins.

input : $Xapp$: matrice $n \times p$ d'individus d'apprentissage, $zapp$: vecteur d'étiquettes des n individus d'apprentissage, $Xval$: matrice $m \times p$ d'individus de validation, $zval$ vecteur d'étiquettes des m individus de validation, $nppv$ vecteur contenant les valeurs de K à tester

output: $Kopt$: valeur de K pour laquelle nous obtenons le taux d'erreur le plus bas

$Kopt \leftarrow$ vecteur de même taille que le vecteur $nppv$ (nous renverrons seulement la valeur du maximum de ce vecteur en fin de fonction);

for $i \leftarrow 1$ **to** $taille(nppv)$ **do**

$ztst \leftarrow kppv.app(Xapp, zapp, nppv[i], Xval)$;

$Kopt[i] \leftarrow$ rapport de la taille du vecteur contenant les individus mal étiquetés sur la taille totale de l'échantillon de validation, multiplié par 100 ($\Leftrightarrow$ pourcentage d'erreur)

end

return $min(Kopt)$

Algorithm 4: Algorithme : kppv.tune

Chapitre 3

Evaluation des performances

3.1 Introduction

Evaluation des performances Dans cette seconde partie, nous souhaitons évaluer les performances de nos classifieurs, c'est à dire obtenir des estimations ε de nos taux d'erreurs et ainsi que leurs intervalles de confiance.

Estimations préalable des paramètres des distributions Nous estimerons au préalable les paramètres des distributions des individus.

3.2 Application

3.2.1 Estimation des paramètres des distributions conditionnelles

Génération des jeux de données Pour chacun des jeux de données, chaque classe a été générée suivant une loi normale bivariée.

Estimation des paramètres d'une loi normale multidimensionnelle Si $X_1, ..., X_n$ est un échantillon *iid* de vecteur aléatoire parent $X \sim \mathcal{N}(\mu, \Sigma)$, alors les estimateurs du maximum de vraisemblance $\hat{\mu}$ et $\hat{\Sigma}$) de μ et Σ) sont le vecteur moyenne empirique $\bar{X}$ et la matrice de variance empirique V et on a $\hat{\mu} \sim \mathcal{N}(\mu, \frac{1}{n}\Sigma)$ (cf poly p.137).

Q1.2.1.1. Résultats Nous appliquons donc la propriété précédente et obtenons les résultat suivants.

TABLE 3.1 – Estimations des paramètres des distributions conditionnelles

	Synth1-40	Synth1-100	Synth1-500	Synth1-1000
π_1	0.575	0.460	0.516	0.481
π_2	0.425	0.540	0.484	0.519
μ_1	$\begin{pmatrix} -0.049 \\ 1.976 \end{pmatrix}$	$\begin{pmatrix} 0.168 \\ 2.060 \end{pmatrix}$	$\begin{pmatrix} 0.028 \\ 2.004 \end{pmatrix}$	$\begin{pmatrix} -0.084 \\ 1.985 \end{pmatrix}$
μ_2	$\begin{pmatrix} 0.110 \\ -0.696 \end{pmatrix}$	$\begin{pmatrix} -0.187 \\ -1.064 \end{pmatrix}$	$\begin{pmatrix} -0.034 \\ -0.901 \end{pmatrix}$	$\begin{pmatrix} -0.038 \\ -1.013 \end{pmatrix}$
Σ_1	$\begin{bmatrix} 1.449 & -0.108 \\ -0.108 & 0.894 \end{bmatrix}$	$\begin{bmatrix} 0.892 & 0.071 \\ 0.071 & 0.887 \end{bmatrix}$	$\begin{bmatrix} 0.994 & 0.004 \\ 0.004 & 0.943 \end{bmatrix}$	$\begin{bmatrix} 1.030 & 0.054 \\ 0.054 & 1.016 \end{bmatrix}$
Σ_2	$\begin{bmatrix} 1.749 & -0.323 \\ -0.323 & 0.832 \end{bmatrix}$	$\begin{bmatrix} 0.900 & 0.069 \\ 0.069 & 0.612 \end{bmatrix}$	$\begin{bmatrix} 0.899 & -0.023 \\ -0.023 & 0.817 \end{bmatrix}$	$\begin{bmatrix} 0.977 & -0.019 \\ -0.019 & 0.955 \end{bmatrix}$

Interprétation Sachant que les jeux de données ont été "générés", nous pourrions pré-supposer que les paramètres choisis pour la génération sont les suivants.

TABLE 3.2 – Hypothèse concernant le choix des paramètres des distributions conditionnelles

π_1	π_2	μ_1	μ_2	Σ_1	Σ_2
0.5	0.5	$\begin{pmatrix} 0 \\ 2 \end{pmatrix}$	$\begin{pmatrix} 0 \\ -1 \end{pmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

3.2.2 Classifieur euclidien - Estimation du taux d'erreur

Estimation ponctuelle du taux d'erreur En déterminant N séparations aléatoires de l'ensemble de données en un ensemble d'apprentissage et un ensemble de test, on peut recueillir un échantillon de N estimations $E_1, ..., E_n$ du taux d'erreur. La moyenne de ces estimations est un estimateur ponctuel de ε, le taux d'erreur.

Intervalle de confiance Le taux d'erreur ponctuel d'un échantillon peut s'écrire sous la forme suivante

$$E_j = \sum_{i=1}^{n} \mathbb{1}_{\omega(x_i) \neq \omega_i}$$

avec j indice de l'expérience, n le nombre d'individus dans l'ensemble testé, ω_i la classe réelle de l'individu i, et $\omega(x_i)$ la classe attribuée à l'individu i par notre classifieur.

Loi binomiale (cf poly SY02 p.20) On peut donc considérer que le taux d'erreur ponctuel d'un échantillon est l'équivalent de la somme de n expériences aléatoires suivant une loi de Bernouilli. Il est donc possible d'affirmer que notre taux d'erreur suit une loi binomiale comme suit

$$E_j \sim \mathcal{B}(n, \varepsilon)$$

TLC & Théorême de Slutsky (cf poly SY02 p.55) Par suite du TLC, nous avons

$$\frac{E_j - n\varepsilon}{\sqrt{n\varepsilon(1-\varepsilon)}} \xrightarrow{\text{L}} \mathcal{N}(0,1)$$

C'est à dire

$$\frac{\frac{E_j}{n} - \varepsilon}{\sqrt{\frac{\varepsilon(1-\varepsilon)}{n}}} \xrightarrow{\text{L}} \mathcal{N}(0,1)$$

Qui est une fonction asymptotiquement pivotale pour ε. De plus, $E_j \xrightarrow{\text{L}} \varepsilon$. Ainsi, par suite du théorême de Slutsky, nous obtenons

$$\frac{E_j - n\varepsilon}{\sqrt{n\varepsilon(1-\varepsilon)}} \sqrt{\frac{\varepsilon(1-\varepsilon)}{\frac{E_j}{n}(1-\frac{E_j}{n})}} \xrightarrow{\text{L}} \mathcal{N}(0,1)$$

Nous en déduisons l'intervalle de confiance suivant pour notre taux d'erreur ε

$$IC = [\frac{E_j}{n} - u_{1-\frac{\alpha}{2}} \sqrt{\frac{\frac{E_j}{n}(1-\frac{E_j}{n})}{n}}; \frac{E_j}{n} + u_{1-\frac{\alpha}{2}} \sqrt{\frac{\frac{E_j}{n}(1-\frac{E_j}{n})}{n}}]$$

Nous utiliserons cet Interval de Confiance pour toutes nos estimations de taux d'erreur.

Q1.2.1.2 - Code Le code effectuant la série de 20 expériences à l'aide du classifieur euclidien est disponible au sein du fichier `TP3.R` fourni en annexe.

Q1.2.1.2 - Résultats Le tableau ci-dessous récapitule nos résultats pour cette question.

TABLE 3.3 – Estimations des taux d'erreurs du classifieur euclidien pour les différents jeux de données

	Synth1-40	Synth1-100	Synth1-500	Synth1-1000
ε *train*	0.063	0.051	0.052	0.066
IC train	$[-0.012; 0.139]$	$[0.008; 0.094]$	$[0.033; 0.072]$	$[0.050; 0.081]$
ε *test*	0.107	0.058	0.050	0.066
IC test	$[0.011; 0.203]$	$[0.012; 0.103]$	$[0.031; 0.069]$	$[0.051; 0.081]$

Observations Les résultats présentent un taux d'erreur très stable sur la classification de l'ensemble d'apprentissage (environ 5%), et un taux d'erreur diminuant au fur et à mesure que la taille du jeu de données s'agrandit pour la classification de l'ensemble de test (d'environ 10% à environ 5%). Nous remarquons aussi que la largeur de notre intervalle de confiance diminue lorsque la taille du jeu de données augmente dans les deux cas.

Interprétation Il est tout à fait normal (et rassurant) que notre taux d'erreur s'améliore (c'est à dire diminue) lorsque l'on augmente la taille du jeu de données d'apprentissage. De même, il est normal que l'Intervalle de Confiance de nos estimations restrecisse, étant donné qu'en augmentant la taille des jeux de données sur lesquels nous appliquons nos classifieurs, nos estimations des taux d'erreurs deviennent de plus en plus sûres.

3.2.3 Détermination du nombre optimal de voisins sur un ensemble d'apprentissage

Q1.2.1.3 - Code Notre code est disponible en annexe de ce rapport au sein du fichier `TP3.R`.

Résultat Le nombre optimal de voisins lorsque l'on utilise l'ensemble d'apprentissage comme ensemble de validation est $Kopt = 1$.

Interprétation Le résultat précédent est tout à fait logique. En effet, lorsque l'on utilise l'ensemble d'apprentissage comme ensemble de validation, avec $K = 1$, l'algorithme ne prend en compte qu'un seul voisin le plus proche. Or ce voisin le plus proche est l'individu lui même (faisant partie de l'ensemble d'apprentissage ET de l'ensemble de validation). Ainsi, l'étiquette attribuée à cet individu est alors sa propre étiquette, par conséquent tous les individus seront bien classés, et notre taux d'erreur sera égal à $\varepsilon = 0$.

3.2.4 KPPV - Estimation du taux d'erreurs

Introduction De la même manière que nous l'avons fait au sein de la question 1.2.1.1, nous allons estimer les taux d'erreurs de notre classifieur utilisant la méthode des k plus proches voisins, ainsi que les Intervalles de Confiance de ces taux sur les différents jeux de données.

Q1.2.1.4 - Code Notre code pour cette question est également disponible en annexe au sein du fichier `TP3.R`.

Résultats Le tableau ci-dessous récapitule nos résultats expérimentaux.

TABLE 3.4 – Estimations des taux d'erreurs de la méthode des KPPV pour les différents jeux de données

	Synth1-40	Synth1-100	Synth1-500	Synth1-1000
ε *train*	0.030	0.031	0.048	0.058
IC train	$[-0.023; 0.058]$	$[-0.003; 0.065]$	$[0.029; 0.067]$	$[0.043; 0.072]$
ε *test*	0.125	0.044	0.060	0.080
IC test	$[0.032; 0.248]$	$[0.004; 0.084]$	$[0.039; 0.080]$	$[0.063; 0.096]$

Observations Les résultats que nous obtenons sont relativement les mêmes qu'avec le classifieur euclidien. De nouveau notre taux d'erreur est relativement stable pour la classification de notre ensemble d'apprentissage, et diminue au fur et à mesure de l'augmentation de la taille du jeu de données d'apprentissage dans le cas de la classification de notre ensemble de test. De la même manière, les intervalles de Confiance se "reserrent" lorsque l'on augmente la taille du jeu de données examiné.

Interprétations Nos interprétations restent globalement les mêmes que pour le classifieur euclidien : l'augmentation de la taille du jeu de données d'apprentissage nous permet d'obtenir de meilleures résultats de classification, et une estimation plus fiable du taux d'erreur.

3.2.5 Changement de jeu de données - Synth2

Introduction La distribution des données dans le jeu `Synth2-1000` n'est pas la même que précédemment. Des paramètres différents ont été utilisés pour la génération de ce jeu de données. De la même manière que précédemment, nous commencerons par estimer les paramètres de distribution conditionnelles, puis observerons les résultats que nous obtenons lors de l'estimation des taux d'erreurs et Intervalles de Confiance.

Q1.2.2.1 & 1.2.2.2 - Code Une nouvelle fois, le code que nous avons utilisé afin de répondre à ces questions est disponible dans le fichier `TP3.R` fourni en annexe.

Estimation des paramètres de distribution - Résultat Le tableau ci-dessous récapitule nos résultats d'estimation des paramètres de distribution.

TABLE 3.5 – Estimation des paramètres des distributions conditionnelles du jeu Synth2-1000

π_1	π_2	μ_1	μ_2	Σ_1	Σ_2
0.486	0.514	$\begin{pmatrix} -0.072 \\ 2.945 \end{pmatrix}$	$\begin{pmatrix} -0.098 \\ -5.101 \end{pmatrix}$	$\begin{bmatrix} 1.069 & -0.053 \\ -0.053 & 1.069 \end{bmatrix}$	$\begin{bmatrix} 5.217 & 0.114 \\ 0.114 & 5.169 \end{bmatrix}$

Interprétation De nouveau, le jeu de données est généré, nous pourrions donc émettre l'hypothèse que les paramètres renseignés par l'enseignant pour la génération du jeu de données sont les suivants.

TABLE 3.6 – Hypothèse concernant le choix des paramètres des distributions conditionnelles du jeu Synth2-1000

π_1	π_2	μ_1	μ_2	Σ_1	Σ_2
0.5	0.5	$\begin{pmatrix} 0 \\ 3 \end{pmatrix}$	$\begin{pmatrix} 0 \\ -5 \end{pmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}$

Classifieur euclidien - Estimations des taux d'erreurs Le tableau ci-dessous récapitule les résultats d'estimation des taux d'erreurs obtenus en effectuant la classification du jeu de données `Synth2-1000` à l'aide du classifieur euclidien.

TABLE 3.7 – Estimations des taux d'erreurs de la discrimination par classifieur euclidien pour le jeu Synth2-1000

ε *train*	0.021
IC train	$[0.012; 0.029]$
ε *test*	0.024
IC test	$[0.015; 0.034]$

Observations Nous observons que les taux d'erreurs sont ici beaucoup moins élevés que lors de l'application du classifieur euclidien aux jeux de données précédents (notamment celui contenant le même nombre d'individus). Notons également que les Intervalles de Confiance sont légèrement moins larges que précédemment.

Interprétation Nous pouvons émettre l'hypothèse que la distance euclidienne entre les deux centres de gravités, des deux nuages d'individus correspondant aux deux classes est plus élevée, ce qui expliquerait ainsi que le classifieur euclidien ait plus de facilités à différencier les individus des deux classes. Cette hypothèse semble confirmée par notre estimation préalable des paramètres de distribution conditionnelle.

KPPV - Estimations des taux d'erreurs Répétons l'opération avec la méthode des KPPV.

TABLE 3.8 – Estimations des taux d'erreurs de la discrimination par la méthode des KPPV pour le jeu Synth2-1000

ε *train*	0.001
IC train	$[-0.001; 0.003]$
ε *test*	0.007
IC test	$[0.002; 0.012]$

Observations De nouveau, nous obtenons des taux d'erreurs inférieurs à ceux obtenus en effectuant la discriminations des individus des jeux précédents, et également inférieurs à ceux obtenus pour le même jeu de données en utilisant le classifieur euclidien. Nos Intervalles de Confiance sont également très restreints, ce qui traduit une bonne fiabilité de taux d'erreurs.

Interprétation Ces résultats semblent confirmer ceux obtenus via le classifieur euclidien, ainsi que notre hypothèse précédente. Les deux nuages d'individus semblent mieux séparés, plus éloignés, ce qui expliquerait de meilleures performances de nos classifieurs.

Chapitre 4

Règle de Bayes

4.1 Introduction

Objectif Dans cette section, les méthodes de génération des jeux de données nous sont explicitées. L'objectif est maintenant de mettre en application les notions concernant la règle de Bayes apprises en cours.

Densité d'une loi normale multidimensionnelle - Rappel (cf poly SY09 p.137) Lorsque X suit une loi normale multidimensionnelle ($X \sim \mathcal{N}(\mu, \Sigma)$), la densité de X s'exprime comme suit.

$$f(x) = \frac{1}{(2\pi)^{\frac{p}{2}}(det\Sigma)^{\frac{1}{2}}} exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$$

Lois marginales - Rappel (cf poly SY09 p.134) Dans un premier temps, rappelons que tout sous-vecteur d'un vecteur aléatoire X, c'est à dire tout sous-ensemble de l'ensemble des variables aléatoires $X_1, ..., X_p$ est lui-même un vecteur aléatoire. La loi d'un tel vecteur aléatoire est appelée loi marginale.

4.2 Application

4.2.1 Distributions marginales des X dans chaque classe

Introduction Les différentes classes des différents jeux de données sont générées suivant des lois normales tel que suit.

TABLE 4.1 – Lois normales selon lesquels sont répartis les X

Lois marginales	ω_1	ω_2
X_{ω_i}	$X_{\omega_1} \sim \mathcal{N}(\mu_1, \Sigma 1)$	$X_{\omega_2} \sim \mathcal{N}(\mu_2, \Sigma_2)$

Avec pour paramètres :

TABLE 4.2 – Paramètres fournis pour la génération des différents jeux de données

	μ_1	μ_2	Σ_1	Σ_2
Synth1	$\binom{0}{2}$	$\binom{0}{-1}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Synth2	$\binom{0}{3}$	$\binom{0}{-5}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}$

Note Nous notons par la suite X_i^k la $i$ème composante du vecteur d'individus X^k appartenant à la classe k.

Q2.2.1 - Distributions marginales Les propriétés fondamentales de distribution marginale d'une loi normale multivariée nous permettent d'obtenir la répartition des distributions marginales suivante :

TABLE 4.3 – Distributions marginales des X dans chaque classe

	X_1^1	X_2^1	X_1^2	X_2^2
Synth1	$X_1^1 \sim \mathcal{N}(0,1)$	$X_2^1 \sim \mathcal{N}(2,1)$	$X_1^2 \sim \mathcal{N}(0,1)$	$X_2^2 \sim \mathcal{N}(-1,1)$
Synth2	$X_1^1 \sim \mathcal{N}(0,1)$	$X_2^1 \sim \mathcal{N}(3,1)$	$X_1^2 \sim \mathcal{N}(0,5)$	$X_2^2 \sim \mathcal{N}(-5,5)$

4.2.2 Courbes d'iso-densité

Q2.2.2 - Courbes d'iso-densité (cf poly SY09 p.137) L'expression des courbes d'isodensité d'une loi normale multidimensionnelle est de la forme

$$(x-\mu)^T\Sigma^{-1}(x-\mu) = c$$

avec c constante.

Remarque Lorsque Σ est diagonale, ces courbes d'isodensité sont des ellipsoïdes de centre μ.

Preuve Prouvons que dans le cas de la génération des différentes classes de nos différents jeux de données il s'agit de cercle. Nous avons ici des paramètres suivant les formes suivantes.

$$X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$\mu = \begin{pmatrix} \mu_1 \\ \mu 2 \end{pmatrix}$$

$$\Sigma = \begin{bmatrix} k & 0 \\ 0 & k \end{bmatrix}$$

or

$$\Sigma^{-1} = \begin{bmatrix} K & 0 \\ 0 & K \end{bmatrix}$$

avec $K \neq k$ lorsque $k \neq 1$ et

$$(x-\mu) = \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{pmatrix}$$

ainsi

$$(x-\mu)^T\Sigma^{-1}(x-\mu) = c$$
$$\Leftrightarrow K(x_1-\mu_1)^2 + K(x_2-\mu_2)^2 = c$$

Nous retrouvons l'équation basique d'un cercle de centre $\mu = \binom{\mu_1}{\mu 2}$ et de rayon $\sqrt{c}$.

4.2.3 Expression de la règle de Bayes

Introduction La règle de Bayes est la règle de décision minimisant la probabilité d'erreur. Elle peut s'écrire dans le cas ou le nombre de classe est $g = 2$ classes comme suit.

$$\delta^*(x) = a_1 \Leftrightarrow \mathbb{P}(\omega_1|x) > \mathbb{P}(\omega_2|x)$$

$$\Leftrightarrow \frac{f_1(x)\pi_1}{f(x)} > \frac{f_2(x)\pi_2}{f(x)}$$

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

Ainsi nous avons

$$\delta^*(x) = \begin{cases} a_1, & \text{si } \frac{f_1(x)}{f_2(x)} > \frac{\pi_1}{\pi_2} \\ a_2, & \text{sinon} \end{cases}$$

Or, au sein de nos deux échantillons

$$\pi_1 = \pi_2 = 0.5$$

et

$$f_k(x) = f(x|\omega_k) = \frac{1}{(2\pi)^{\frac{p}{2}}(det\Sigma_k)^{\frac{1}{2}}} exp(-\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k))$$

avec $p = 2, \forall k$

Chapitre 5

Conclusion

Facteur temps Malheureusement, ce rapport de TP n'ira pas plus loin, nous avons mal géré notre temps et n'avons pas réussi à conclure la partie sur la Théorie Bayésienne de la décision avant la date de rendu de notre TP.

Conclusion Ce troisième TP de SY09 a été pour nous l'occasion de mettre en pratique les premiers concepts de l'apprentissage supervisé abordés en cours. Nous avons notamment eu l'opportunité de coder les fonctions d'apprentissage et de test d'un classifieur euclidien et de la méthode des k plus proches voisins sous. Nous avons également appliqué les méthodes d'estimation de taux d'erreurs et de détermination d'Intervalles de Confiance. Finalement, bien que notre application de ces concepts soit incomplète, la dernière partie de ce TP sur la théorie Bayésienne de la décision a été pour nous une très bonne occasion de réviser ces concepts, notamment en travaillant sur la règle de Bayes et l'erreur de Bayes.