SY09 Printemps 2017
TP 4 – Discrimination

1 Programmation

1.1 Analyse discriminante

On se propose de programmer trois modèles d'analyse discriminante *dans le cas binaire* (jeux de données comptant $g = 2$ classes) : l'analyse discriminante quadratique, l'analyse discriminante linéaire, et le classifieur bayésien naïf. On complétera pour cela quatre fonctions : adq.app, adl.app, nba.app et ad.val.

Les trois premières font l'apprentissage des trois modèles d'analyse discriminante mentionnés ci-dessus : elles doivent donc prendre en argument d'entrée le tableau de données Xapp et le vecteur zapp des étiquettes associées, et retourner les paramètres du modèle (proportions, vecteurs de moyennes et matrices de covariance des deux classes) que l'on pourra stocker dans une structure appropriée.

La fonction ad.val calcule les probabilités a posteriori pour un ensemble de données, puis effectue le classement en fonction de ces probabilités : elle prend donc en compte les paramètres du modèle et l'ensemble de données Xtst à classer, et retourne une structure contenant les probabilités a posteriori prob estimées et le classement associé.

Cette fonction pourra s'appuyer sur la fonction mvdnorm, disponible sur le site de l'UV, qui permet de calculer la densité d'une loi normale multivariée pour un tableau de données. On suggère de stocker les matrices de covariance de chaque modèle dans un tableau à trois dimensions (structure array).

On pourra également utiliser la fonction prob.ad, disponible sur le site de l'UV, pour afficher les courbes de niveau des probabilités a posteriori $\widehat{\mathbb{P}}(\omega_1|\boldsymbol{x})$ estimées. La frontière de décision correspond à la courbe de niveau $\widehat{\mathbb{P}}(\omega_1|\boldsymbol{x}) = 0.5$.

1.2 Régression logistique

On souhaite implémenter le modèle logistique *binaire*. On complétera tout d'abord deux fonctions, l'une permettant de faire l'apprentissage du modèle (on utilisera l'algorithme de Newton-Raphson présenté en cours), l'autre permettant d'appliquer le modèle obtenu sur un ensemble de données.

Apprentissage. La fonction log.app, permettant d'apprendre les paramètres du modèles, prendra comme arguments d'entrée le tableau de données Xapp, le vecteur zapp des étiquettes associées, ainsi qu'une variable binaire intr indiquant s'il faut ou non ajouter une ordonnée à l'origine (*intercept*) à la matrice d'exemples et un scalaire epsi correspondant au seuil ε en-deçà duquel on considère que l'algorithme d'apprentissage a convergé.

Elle devra retourner la matrice beta correspondant à l'estimateur du maximum de vraisemblance $\widehat{\beta}$ des paramètres, de dimensions $p \times 1$ (ou $(p+1) \times 1$ si une ordonnée à l'origine a été ajoutée), le nombre niter d'itérations effectuées par l'algorithme de Newton-Raphson, et la valeur logL de la vraisemblance à l'optimum.

On pourra utiliser comme matrice de paramètres initiale $\beta^{(0)} = (0, \ldots, 0)^T$. La convergence sera testée en comparant la norme de la différence entre deux estimations successives $\beta^{(q)}$ et $\beta^{(q+1)}$ au seuil ε (on pourra choisir $\varepsilon = 1e-5$).

Classement. La fonction log.val, permettant de classer un ensemble d'individus, prendra comme arguments d'entrée le tableau de données Xtst à classer et la matrice beta des paramètres déterminés par la fonction log.app. Cette fonction fera un test sur les dimensions de la matrice beta, pour déterminer si une ordonnée à l'origine doit être ou non ajoutée à la matrice d'exemples Xtst à évaluer.

Elle devra retourner une structure contenant la matrice prob des probabilités a posteriori estimées et le vecteur des classements associés.

Ces fonctions pourront s'appuyer sur une fonction post.pr calculant les probabilités a posteriori à partir d'une matrice de paramètres beta et d'un tableau de données X.

Régression logistique quadratique. Il est possible de généraliser le modèle de régression logistique de manière très simple. La stratégie consiste à transformer les données dans un espace plus complexe, dans lequel les classes peuvent être séparées par un hyperplan. La régression logistique est alors effectuée dans cet espace.

Le modèle ainsi défini est donc plus flexible (il permet de construire une frontière de décision plus complexe) ; mais le nombre de paramètres à estimer étant plus important, il peut également s'avérer moins robuste que le modèle classique déterminé dans l'espace des caractéristiques initiales.

Par exemple, dans le cas où les individus sont décrits par les variables naturelles X^1, X^2 et X^3, la régression logistique quadratique consiste à apprendre un modèle de régression logistique classique dans l'espace $\mathcal{X}^2 = \{X^1, X^2, X^3, X^1X^2, X^1X^3, X^2X^3, (X^1)^2, (X^2)^2, (X^3)^2\}$, plutôt que dans l'espace $\mathcal{X} = \{X^1, X^2, X^3\}$.

Pour ce faire, on calcule les produits des variables décrivant les individus d'apprentissage. On apprend ensuite le modèle logistique sur les $p(p+3)/2$ nouvelles variables ainsi obtenues. Notons qu'il sera nécessaire de calculer de même les produits des variables décrivant les individus à classer, *dans le même ordre que pour les individus d'apprentissage.*

$$\mathsf{X} = \begin{pmatrix} 1 & 3 & 2 \\ 2 & 4 & 3 \\ 3 & 1 & 4 \\ \vdots & \vdots & \vdots \end{pmatrix} \quad \longrightarrow \quad \mathsf{X2} = \begin{pmatrix} 1 & 3 & 2 & 3 & 2 & 6 & 1 & 9 & 4 \\ 2 & 4 & 3 & 8 & 6 & 12 & 4 & 16 & 9 \\ 3 & 1 & 4 & 3 & 12 & 4 & 9 & 1 & 16 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

On pourra utiliser les fonctions prob.log et prob.log2 (on veillera à ce que l'ordre des termes quadratiques soit bien le même dans le code développé et dans la fonction prob.log2), disponible sur le site de l'UV, pour visualiser les courbes de niveau ou les frontières de décision obtenues respectivement par régression logistique et par régression logistique quadratique.

1.3 Vérification des fonctions

La Figure 1 montre les frontières de décision obtenues lorsque la totalité des données Synth1-40 (voir TP3) sont utilisées pour l'apprentissage du modèle.

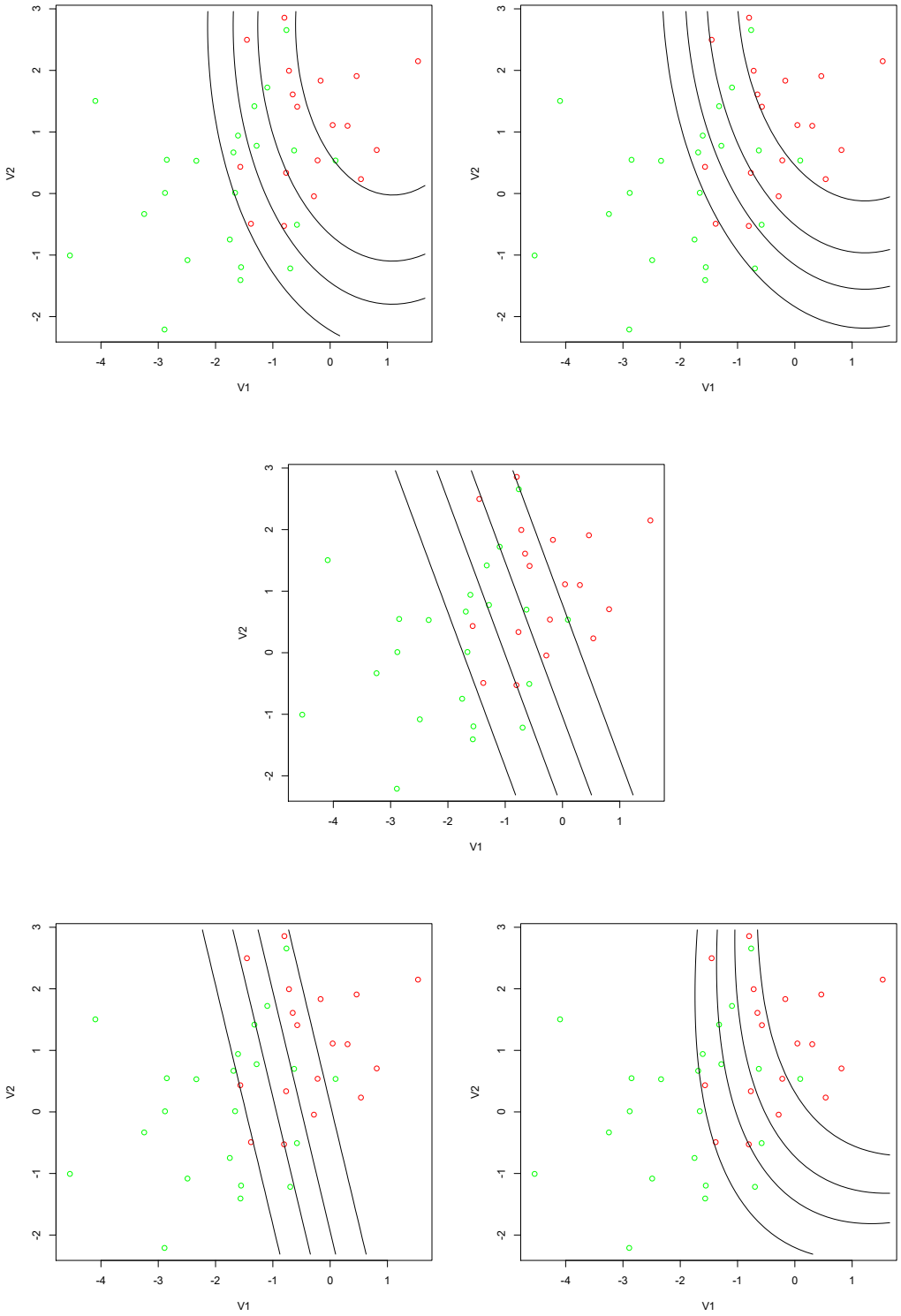

FIGURE 1 – Frontières de décision obtenues en utilisant toutes les données pour l'apprentissage ; analyse discriminante quadratique (haut, gauche), classifieur bayésien naïf (haut, droite), analyse discriminante linéaire (centre), régression logistique (bas, gauche) et régression logistique quadratique (bas, droite).

2 Application

2.1 Test sur données simulées

On souhaite comparer les performances de l'analyse discriminante, de la régression logistique (linéaire et quadratique), et des arbres de décision sur les jeux de données simulées Synth1-1000, Synth2-1000 et Synth3-1000 disponibles sur le site de l'UV. Pour ce faire, on utilisera le même protocole expérimental que dans le TP précédent, que l'on répétera $N = 20$ fois pour chaque jeu de données :

1. séparer le jeu de données en un ensemble d'apprentissage et un ensemble de test ;
2. apprendre le modèle sur l'ensemble d'apprentissage,
3. effectuer le classement des données de test et calculer le taux d'erreur associé.

Pour chaque jeu de données, calculer le taux d'erreur (de test) moyen sur les $N = 20$ séparations effectuées. On pourra s'appuyer sur les frontières de décision obtenues pour analyser les résultats. Sachant que les données suivent dans chaque classe une loi normale multivariée, comment peut-on interpréter ces résultats ?

Arbres L'implémentation des arbres de décision est complexe ; on pourra utiliser la bibliothèque tree. La fonction tree permet de faire l'apprentissage du modèle — on obtiendra l'arbre complet en utilisant le paramètre `control=tree.control(nobs=dim(Dapp)[1],mindev = 0.0001)`. La fonction cv.tree permet de construire la séquence d'arbres emboîtés dans celui obtenu via tree, et d'en estimer l'erreur par validation croisée. La fonction prune.misclass permet d'élaguer l'arbre (en utilisant les erreurs estimées par validation croisée). Enfin, la fonction predict permet de classer un jeu de données au moyen d'un arbre.

Attention : la construction d'un arbre de décision via la fonction tree à des fins de discrimination nécessite d'utiliser une variable à expliquer déclarée sous forme de factor.

2.2 Test sur données réelles

2.2.1 Données « Pima »

On souhaite appliquer les trois modèles d'analyse discriminante, les deux modèles de régression logistique, et les arbres de décision à la prédiction du diabète chez les individus d'une population d'amérindiens. On pourra charger les données au moyen du code suivant :

```
Donn <- read.csv("Pima.csv", header=T)
X <- Donn[,1:7]
z <- Donn[,8]
```

Au cours d'une expérience, on pourra utiliser le code suivant pour créer les données sur lesquelles apprendre le modèle de régression logistique quadratique :

```
Xapp2 <- Xapp
Xtst2 <- Xtst

for (p in 1:(dim(Xapp)[2]-1))
{
    for (q in (p+1):dim(Xapp)[2])
    {
        Xapp2 <- cbind(Xapp2, Xapp[,p]*Xapp[,q])
        Xtst2 <- cbind(Xtst2, Xtst[,p]*Xtst[,q])
    }
}
```

```
for (p in 1:dim(Xapp)[2])
{
    Xapp2 <- cbind(Xapp2, Xapp[,p]^2)
    Xtst2 <- cbind(Xtst2, Xtst[,p]^2)
}
```

On utilisera ensuite le même protocole expérimental que pour les tests sur données simulées, en répétant l'expérience $N = 100$ fois. Calculer les taux moyens d'erreur de test pour chacun des cinq modèles étudiés. Que constate-t-on ? Comment expliquez-vous ces résultats ?

2.2.2 Données « breast cancer Wisconsin »

On considère à présent un problème de prédiction du niveau de gravité d'une tumeur à partir de descripteurs physiologiques. On récupérera les données sur le site de l'UV et on les chargera en utilisant le code suivant :

```
Donn <- read.csv("bcw.csv", header=T)
X <- Donn[,1:9]
z <- Donn[,10]
```

On répétera l'expérience (séparation, apprentissage et évaluation des performances) $N = 100$ fois. Calculer les taux moyens d'erreur de test pour les trois modèles d'analyse discriminante, la régression logistique classique (on n'utilisera pas la régression logistique quadratique ici), et les arbres de décision. Interpréter et commenter les résultats obtenus.

3 Challenge : données « Spam »

On considère enfin un problème de détection de spams à partir d'indicateurs calculés sur des messages électroniques. On récupérera les données sur le site de l'UV et on les chargera en utilisant le code suivant :

```
Donn <- read.csv("spam.csv", header=T)
X <- Donn[,1:57]
z <- Donn[,58]
```

Proposer une stratégie pour utiliser les modèles étudiés dans ce TP sur ces données. Commenter les résultats obtenus.