SY09 TP4

Thibault Subreville & Yuting Chen

P17

L'objectif de ce TP est d'apprendre à manipuler plusieurs types de classifieurs : analyse discriminante, régression logistique, arbres de décision. Dans la première partie de ce document, nous allons construire ces classifieurs tout en les testant sur des jeux de données simulées. Ensuite, nous comparerons le taux d'erreur de chaque classifieurs sur les jeux de données réels, tout en essayant des nouvelles techniques de classification pour le challenge.
L'outil d'analyse statistique utilisé est le logiciel R.
N.B.: Les résultats numériques dans ce rapport seront arrondis à 10^{-3} près sauf mention contraire.

1 Programmation

1.1 Analyse discriminante

Nous devons contruire les classifieurs d'analyse discriminante quadratique (**adq.app**), linéaire (**adl.app**), et le classifieur bayésien naïf (**nba.app**) qui réalise l'apprentissage d'un tableau d'individus-variables.
Pour l'analyse de trois formes de modèle, les paramètres du modèle sont les suivants :

	Proportions $\hat{\pi_k}$	Espérance $\hat{\mu_k}$	Covariances $\hat{\Sigma_k}$
adq.app	$\frac{n_k}{n}$	$\frac{1}{n_k}\sum_{i=1}^{n} z_{ik}x_i$	$\frac{1}{n_k}\sum_{i=1}^{n} z_{ik}(x_i-\hat{\mu_k})(x_i-\hat{\mu_k})^T = V_k$
adl.app	$\frac{n_k}{n}$	$\frac{1}{n_k}\sum_{i=1}^{n} z_{ik}x_i$	$\frac{1}{n_k}\sum_{i=1}^{g}\sum_{i=1}^{n} z_{ik}(x_i-\hat{\mu_k})(x_i-\hat{\mu_k})^T = \frac{1}{n_k}\sum_{i=1}^{g} n_k V_k$
nba.app	$\frac{n_k}{n}$	$\frac{1}{n_k}\sum_{i=1}^{n} z_{ik}x_i$	$diag(diag(V_k))$

N.B.: Ces fonctions peut être utilisées pour un nombre quelconques de classes ($g \geq 2$).
Nous retrouvons les figures de l'énoncé pour les niveaux égaux à 0.2, 0.4, 0.6 et 0.8

```
#Analyse discriminante quadratique
adq.app <- function(Xapp, zapp){
  n <- dim(Xapp)[1]
  p <- dim(Xapp)[2]
  g <- max(unique(zapp))

  param <- NULL
  param$MCov <- array(0, c(p,p,g))
  param$mean <- array(0, c(g,p))
  param$prop <- rep(0, g)

  for (k in 1:g){
    Xappk <- Xapp[which(zapp==k),]

    param$MCov[,,k] <- cov(Xappk);
    param$mean[k,] <- apply(Xappk,2,mean);
    param$prop[k] <- dim(Xappk)[1]/n;
  }

  param
}
```

```
#Analyse discriminante linéaire
adl.app <- function(Xapp, zapp){
  n <- dim(Xapp)[1]
  p <- dim(Xapp)[2]
  g <- max(unique(zapp))

  param <- NULL
  MCov <- array(0, c(p,p))
  param$MCov <- array(0, c(p,p,g))
  param$mean <- array(0, c(g,p))
  param$prop <- rep(0, g)

  for (k in 1:g){
    Xappk <- Xapp[which(zapp==k),]

    param$mean[k,] <- apply(Xappk,2,mean);
    param$prop[k] <-  dim(Xappk)[1]/n;
    MCov <- MCov+ (param$prop[k])*cov(Xappk);
  }

  MCov <- MCov;

  for (k in 1:g){
    param$MCov[,,k] <- MCov ;
  }

  param
}
```

Figure 1: La fonction **adq.app** et **adl.app**

```r
#Classifieur bayésien naïf.
nba.app <- function(Xapp, zapp){
  n <- dim(Xapp)[1]
  p <- dim(Xapp)[2]
  g <- max(unique(zapp))

  param <- NULL
  param$MCov <- array(0, c(p,p,g))
  param$mean <- array(0, c(g,p))
  param$prop <- rep(0, g)

  for (k in 1:g){
    Xappk <- Xapp[which(zapp==k),]

    param$MCov[,,k] <- diag(diag(cov(Xappk)));
    param$mean[k,] <- apply(Xappk,2,mean);
    param$prop[k] <- dim(Xappk)[1]/n;
  }
  param
}
```

```r
#Cacul des probabilités à posteriori
ad.val <- function(param, Xtst){
  n <- dim(Xtst)[1]
  p <- dim(Xtst)[2]
  g <- length(param$prop)

  out <- NULL
  prob <- matrix(0, nrow=n, ncol=g)

  for (k in 1:g)  {
   prob[,k] <- mvdnorm(Xtst, param$mean[k,], param$MCov[,,k]) * param$prop[k]
  }
  prob <- prob/apply(prob,1,sum)
  pred <- max.col(prob)

  out$prob <- prob
  out$pred <- pred

  out
}
```

Figure 2: La fonction **nba.app** et **ad.val**

1.2 Régression logistique

Nous nous proposons maintenant d'étudier la régression logistique linéaire et quadratique. Pour ce faire, nous développons comme précédemment une fonction permettant l'apprentissage du modèle basée sur l'algorithme de Newton-Raphson.

```r
log.app <- function(Xapp, zapp, intr, epsi){
  n <- dim(Xapp)[1]
  p <- dim(Xapp)[2]
  Xapp <- as.matrix(Xapp)
  MatW <- matrix(0,nrow=n,ncol=n)
  if (intr == T){
    Xapp <- cbind(rep(1,n),Xapp)
    p <- p + 1
  }
  targ <- matrix(as.numeric(zapp),nrow=n)
  targ[which(targ==2),] <- 0
  tXap <- t(Xapp)
  beta <- matrix(0,nrow=p,ncol=1)
  conv <- FALSE
  iter <- 0
  while (conv == FALSE){
    iter <- iter + 1
    bold <- beta
    prob <- postprob(beta, Xapp)
    diag(MatW) <- prob*(1-prob)
    beta <- bold + solve(tXap%*%MatW%*%Xapp) %*% tXap %*% (targ-prob)
    if (((norm(beta-bold)) < epsi)){
      conv <- TRUE
    }
  }
  prob <- postprob(beta, Xapp)
  out <- NULL
  out$beta <- beta
  out$iter <- iter
  out$logL <- sum(targ*log(prob)+(1-targ)*log(1-prob))
  out
}
```

Figure 3: La fonction **log.app** : appliquant l'algorithme de Newton-Raphson

N.B.: Nous retrouvons les figures de l'énoncé pour les niveaux égaux à 0.2, 0.4, 0.6 et 0.8

Nous mettons en annexe le programme qui permet de passer d'un modèle de régression logistique classique à un modèle de régression quadratique et la fonction **postprob**.

```r
log.val <- function(beta, Xtst){
  m <- dim(Xtst)[1]
  p <- dim(beta)[1]
  pX <- dim(Xtst)[2]

  Xtst <- as.matrix(Xtst)

  if (pX == (p-1)){
    Xtst  <- cbind(rep(1,m),Xtst)
  }

  prob <- postprob(beta, Xtst)
  #pred <- max.col(prob)

  out <- NULL
  out$prob <- prob
  out$pred <- cbind(prob>=0.5,prob<0.5)%*%c(1,2)
  return(out)
}
```

Figure 4: La fonction **log.val** : renvoit un vecteur d'étiquette prédite

2 Application

2.1 Test sur données simulées

Nous travaillons dans cette sous-partie sur des données simulées construites suivant une loi normale multivariée (gausienne).
Les frontières de décision sont disponibles en annexe de ce document.

- Jeu de données **Synth1-1000**
  Estimation des matrices de variances (estimateur du maximun de vraisemblance) à 10^{-3} près:

$$\Sigma_1 = \begin{pmatrix} 2.891 & -1.529 \\ -1.529 & 1.960 \end{pmatrix} \quad \Sigma_2 = \begin{pmatrix} 2.087 & -0.000 \\ -0.000 & 2.812 \end{pmatrix}$$

  En appliquant les différents algorithmes de discrimination que nous avons programmé précédemment nous obtenons pour le jeu de données **Synth1-1000** les valeurs suivantes:

Méthode de discrimination	Taux d'erreur de test sur 20 séparations
Analyse discriminante quadratique (ADQ)	3.023%
Analyse discriminante linéaire (ADL)	4.116%
Classifieur bayésien naïf	3.967%
Régression logistique linéaire	3.293%
Régression logistique quadratique	3.158%
Arbre de décision	4.640%

  Les résutats des erreurs sont assez faibles et cela ne nous surprend guère sur un jeu de données simulées. Néanmoins, ce qui nous étonne le plus c'est la différence peu flagrante entre les 3 premiers classifieurs qui pour l'analyse présupppose des propriétés des matrices de variance des classes. Cela nous montre que ces classifieurs sont relativement robustes pour ce jeu de données, mais surtout que le jeu de données est assez "facile" à départager (peu de chevauchement de classes).
  Concernant l'hypothèse de diagonalité du classifieur bayésien naïf, on peut vraisemblablement dire qu'elle n'est pas vérifiée. Il est donc normal pour un jeu de données de cette taille de trouver que l'analyse discriminante donne de meilleure performance dans la mesure où les données sont gausiennes et les hypothèses de cette méthode sont vérifiées (contrairement aux autres). Notons tout de même que ce classifieurs a besoin de plus de données pour pouvoir estimer les paramètres du problèmes de discriminations.
  Par ailleurs, les frontières de décisions sont très similaires graphiquement.

- Jeu de données **Synth2-1000**
  Estimation des matrices de variances (estimateur du maximun de vraisemblance) à 10^{-3} près:

$$\Sigma_1 = \begin{pmatrix} 2.811 & -0.187 \\ -0.187 & 0.901 \end{pmatrix} \quad \Sigma_2 = \begin{pmatrix} 0.899 & -0.043 \\ -0.043 & 4.595 \end{pmatrix}$$

Méthode de discrimination	Taux d'erreur de test sur 20 séparations
Analyse discriminante quadratique	6.366%
Analyse discriminante linéaire	8.078%
Classifieur bayésien naïf	6.351%
Régression logistique linéaire	6.996%
Régression logistique quadratique	6.456%
Arbre de décision	7.252%

  Pour ce jeu de données, les résultats sont aussi assez bon de l'ordre de 7-8%. On remarque une meilleur performance de la part du classifieur de Bayse. Cette performance s'explique principalement par le fait que que les matrices de variances sont assez proche d'être des matrices diagonales.

- Jeu de données **Synth3-1000**
  Estimation des matrices de variances (estimateur du maximun de vraisemblance) à 10^{-3} près:

$$\Sigma_1 = \begin{pmatrix} 2.875 & -1.553 \\ -1.553 & 3.497 \end{pmatrix} \quad \Sigma_2 = \begin{pmatrix} 2.922 & -2.019 \\ -2.019 & 4.165 \end{pmatrix}$$

Méthode de discrimination	Taux d'erreur de test sur 20 séparations
Analyse discriminante quadratique	4.339%
Analyse discriminante linéaire	4.534%
Classifieur bayésien naïf	5.045%
Régression logistique linéaire	4.294%
Régression logistique quadratique	4.519%
Arbre de décision	6.861%

  Encore une fois les résultats des classifieurs sont sensiblement identiques. On remarque néanmoins, une performance de meilleure qualité pour l'analyse discriminante linéaire dans la mesure où les matrices de variances des 2 classes sont assez proches (hypothèse discriminatoire linéaire vérifiée).

Remarque générale: Nous remarquons que les arbres de décision donnent de moins bonnes performances sur ces jeux de données en règle général. La précision de ces estimations dépend de la taille du sous-ensemble d'apprentissage et varie fortement en fonction de ce dernier. En effet, l'étape d'élagage est très sensible aux données d'apprentissages et donc toute la structure de l'arbre obtenue.
Nous avons utilisé le critère d'impureté d'entropie, mais les résultats en pratique change peu en utilisant le critère de Gini.

2.2 Test sur données réelles

2.2.1 Données "Pima"

Le jeu de données Pima porte sur des individus d'une population d'Amérindiens et nous souhaitons l'utiliser afin de prédire le diabète chez ces individus. Pour cela, nous allons appliquer les 6 modèles dont nous disposons et nous avons calculer pour chacun le taux d'erreur moyen après 100 répétitions. Les résultats sont présentés dans le tableau ci-dessous:

Méthode de discrimination	Taux d'erreur de test sur 100 séparations
Analyse discriminante quadratique	24.384%
Analyse discriminante linéaire	22.094%
Classifieur bayésien naïf	23.808%
Régression logistique linéaire	21.932%
Régression logistique quadratique	24.876%
Arbre de décision	24.870%

Comme les taux d'erreurs sont élevés, on peut faire l'hypothèse que les distributions du jeu de données Pima ne suivent pas forcément une loi normale multidimensionnelle dans chaque classe. En effet, on le prouve avec un test Kolmogorov-Smirnov (après réduction et centrage) sur une loi normale centrée réduite, nous nous trouvons dans la région critique du test même au seuil de 1% pour les 2 classes. Les classifieurs sont donc moins performants car les hypothèses des méthodes paramétriques ne sont clairement pas respectées.

2.2.2 Données "breast cancer Wisconsin"

Nous voulons utiliser le jeu de données breast cancer Wisconsin (bcw) afin de prédire la présence/abscence de cancer. Les données portent sur des descripteurs physiologiques. Nous allons appliquer le même protocole que pour les données précédentes. Les résultats sont présentés dans le tableau ci-dessous :

Méthode de discrimination	Taux d'erreur de test sur 100 séparations
Analyse discriminante quadratique	5.110%
Analyse discriminante linéaire	4.803%
Classifieur bayésien naïf	4.035%
Régression logistique linéaire	4.540%
Arbre de décision	5.899%

Les classifieurs ont de meilleurs résultats avec ce jeu de données qu'avec les données Pima. La méthode qui donne les meilleures performances est le classifieur bayésien naïf.
Dans le cas de la régression logistique quadratique, elle ne fonctionne pas à cause d'une *instabilité numérique*. La matrice à inversé ω possède des valeurs trop petites (de l'ordre de 10^{-16}) la rendant impossible le calcul pour l'ordinateur.

3 Challenge : données "Spam"

Pour ce jeu de données, du fait de la *instabilité numérique* il nous est impossible d'utiliser les méthodes régressions logistiques qui demande d'inverser des matrices sans traitement préalable des données.
Nous réalisons une ACP sur les données Spam en utilisant la fonction **princomp**, puis nous prenons les 2 premiers axes factoriels (avec une inertie expliqué dans le premier plan factoriel proche de 99%), nous obtenons:

Méthode de discrimination	Taux d'erreur de test sur 20 séparations
Analyse discriminante quadratique	24.846%
Analyse discriminante linéaire	30.061%
Classifieur bayésien naïf	23.894%
Régression logistique linéaire	26.373%
Régression logistique quadratique	25.551%

Les résultats ne sont pas bons mais cela peut s'expliquer par le fait que les données présentes de nombreux 0. Ces 0 ne sont pas des valeurs manquantes mais l'absence d'un mot dans le mail. Cela a pour effet de "fausser" notre ACP.
En utilisant les données brutes (sans traitement préalable) du **spam.csv**, on obtient:

Méthode de discrimination	Taux d'erreur de test sur 20 séparations
Arbre de décision	7.941%

En remarquant que les arbres de décision nous donnent de bons résultats; en tout cas meilleur que les classifieurs précédents. Nous utilisons les forêts aléatoires (qui permettra de réduire la variance des arbres qui sont très sensible aux données d'apprentissage).
La fonction utilisée est **randomForest**. Nous obtenons alors les résultats suivant:

Méthode	Taux d'erreur de test à 10^{-2} près
Forêt aléatoire (20 arbres)	1.91%
Forêt aléatoire (500 arbres)	1.43%
Forêt aléatoire (1000 arbres)	1.26%

Pour la forêt aléatoire constituée de 1000 arbres, nous obtenons la matrice de confusion suivante:

$$\begin{array}{c} \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cc} 1 & 2 \end{array} \\ \begin{pmatrix} 1781 & 32 \\ 26 & 2762 \end{pmatrix} \end{array}$$

Conclusion du jeu données Spam: Les méthodes de réduction de variables qui au départ nous semblaient prometteuses nous donne des résultats assez faibles. Il aurait peut-être fallu changer le critère de choix des variables en amont. Néanmoins, les arbres et surtout les forêts aléatoires nous donnent de bons résultats. En réalisant des forêts aléatoires de plus de 1000 arbres les résultats tendent vers 1.30% qui semble être l'optimal pour cette méthode sur ce jeu de données.

4 Conclusion du TP

Ce TP nous a permis d'appliquer les différentes méthodes d'analyse discriminantes, de régression logistique et d'arbres de décision binaires sur des données binaires tout en comparant leurs performances.
Il est intéressant de constater la différence entre l'application sur des données simulées (i.e. dont on connait la distribution conditionnelle des classes) et des données réelles pour lesquelles les performances des classifieurs varient beaucoup.
Les forêts aléatoires nous on beaucoup étonnées par leur performance sur le jeu de données spam. Néanmoins, nous avons testé cette méthode sur les jeux de données simulées et nous constatons de moins bonnes performances que les autres classifieurs (où les hypothèses d'utilisation sont vérifiées).

5 Annexes

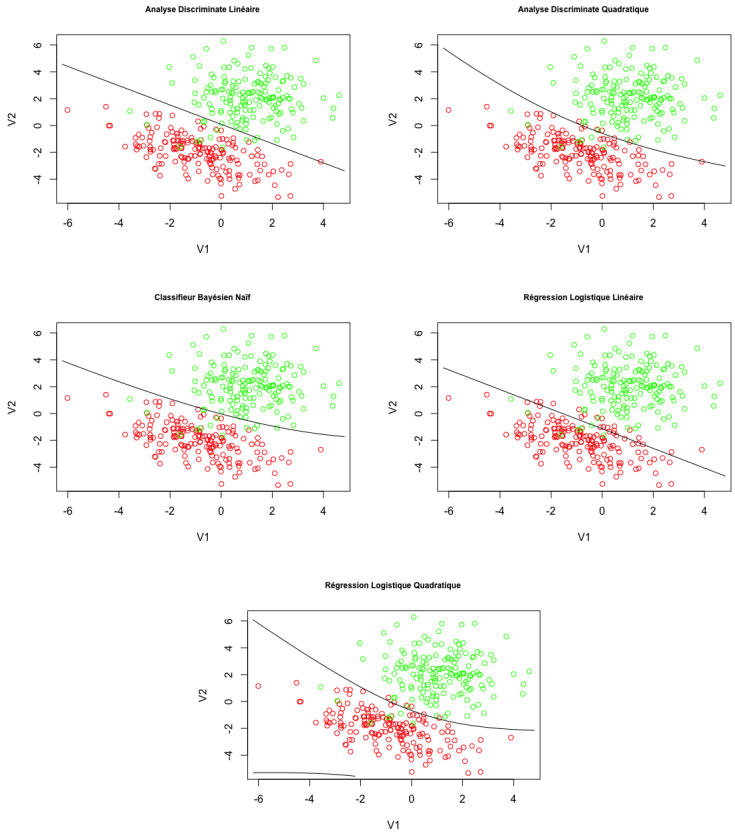

Figure 5: Frontière de décision pour synth1 (données de test)

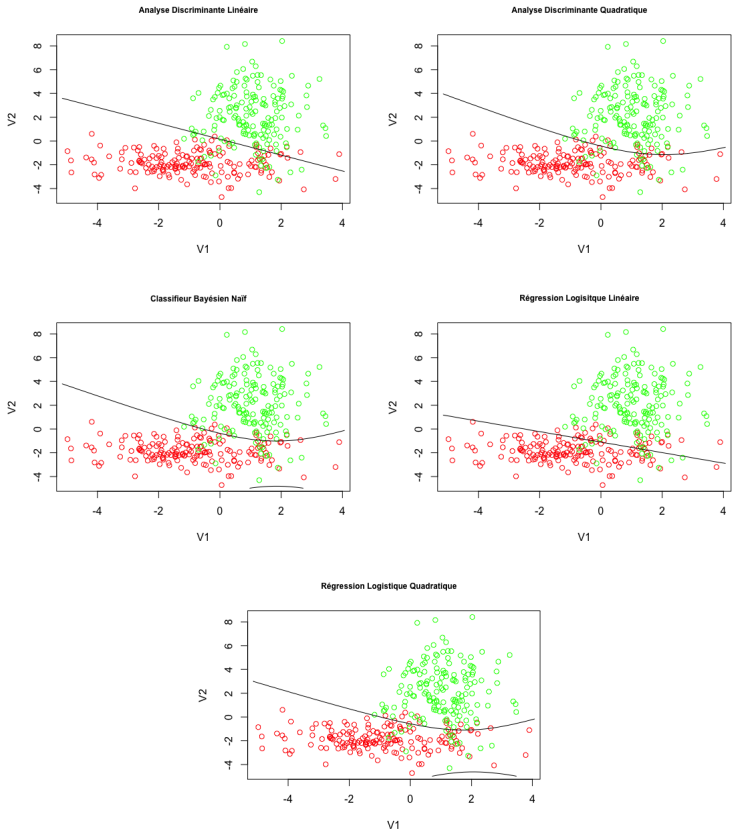

Figure 6: Frontière de décision pour synth2 (données de test)

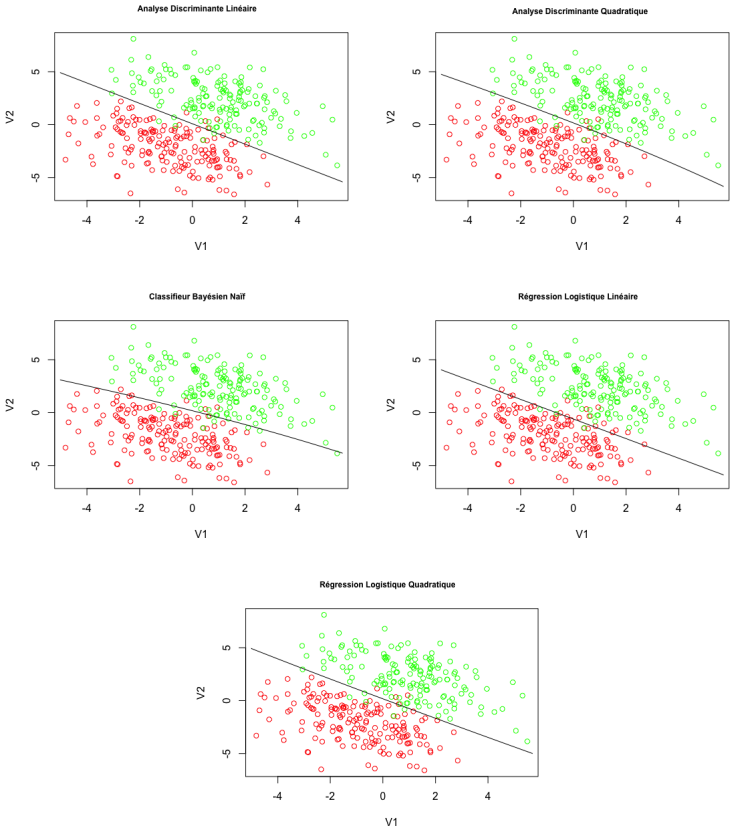

Figure 7: Frontière de décision pour synth3 (données de test)

```r
#Pour transformer dans un espace de régression logistique quadratique
log.transquadra <- function(X){
  X2 <- X
  for (p in 1:(dim(X)[2]-1)){
    for (q in (p+1):dim(X)[2]){
      X2 <- cbind(X2, X[,p]*X[,q])
    }
  }
  for (p in 1:dim(X)[2]){
    X2 <- cbind(X2, X[,p]^2)
  }
  X2
}
```

Figure 8: La fonction **log.transquadra** : renvoit la matrice d'espace χ^2

```r
postprob <- function(beta,X){
  X <- as.matrix(X)
  beta<-as.matrix(beta)
  exp(X%*%beta) / (1 + exp(X%*%beta))
}
```

Figure 9: La fonction **postprob** : cacule la probabilité à postériori