SY09 - Projet 2

Analyse de données et apprentissage automatique

29 juin 2018

Étudiants : Arnaud Dalie - Nour Rammal

Responsables UV : QUOST Benjamin - ROUSSEAU Sylvain
Responsable TD : QUOST Benjamin

Table des matières

1 Discrimination

Dans cette partie, on effectue une comparaison entre les performances des différents classifieurs sur 5 jeux de données. Nous avons choisi de séparer nos données aléatoirement en un ensemble d'apprentissage et un ensemble de test. On répète 200 fois le processus de séparation, et pour chaque modèle nous calculons le taux d'erreur obtenus sur le classement effectué.

Nous expliquerons au fur et à mesure les raisons pour lesquelles nous avons choisis d'utiliser les fonctions natives de R.

1.1 Breast cancer

Ce jeu de données comporte 31 variables continues et 569 individus, et ne comporte pas de valeurs manquantes. On remarque qu'il s'agit uniquement de valeurs positives.

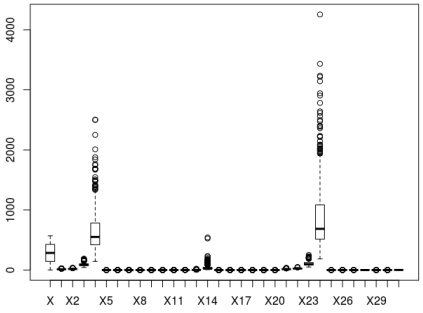


FIGURE 1 – Boites à moustaches des données

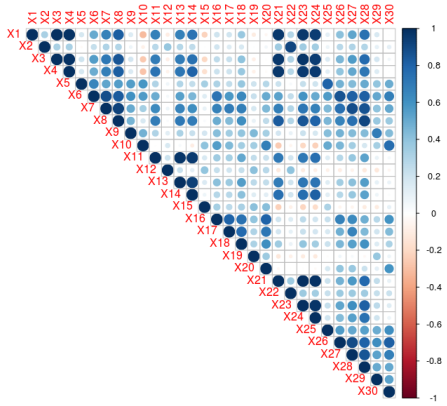


FIGURE 2 – Matrice de corrélation des variables

Le diagramme à moustaches nous montre que les données sont atypiques. Beaucoup d'individus pour quelques variables sont loin du 1er ou 3ème quartile, et donc ne suivent pas la distribution caractéristique du reste des données.
Cependant, ceci peut poser problème lors des calculs des distributions de chaque variable. La plupart des individus peuvent être concentrés alors qu'une minorité est étalonné avec une grande variance. Ainsi, calculer la fonction de densité peut être presque nulle pour les individus étalonnés. Ce calcul parait théoriquement possible mais nous provoque une erreur dans nos fonctions. En effet, la fonction de densité sur certains intervalles donne une densité nulle, à cause de la distribution asymétrique des données. Ce calcul est donc numériquement impossible. Une solution est de centrer réduire les données, ce qui diminue les variances entre les individus.

Il est de même intéressant d'analyser les corrélations des variables. La matrice de corrélation des variables montre que plusieurs variables sont corrélées. Certaines variables sont positivement très corrélés comme par exemple X1 et X3 avec corr = 0.997855281493811. Nous avons codées une fonction qui permet de renvoyer les variables dont la corrélation est supérieure à 0.9 (ou inférieure à -0.9). Il existe 21 variables fortement corrélées.

Avoir une corrélation très forte entre deux variables différentes, c'est avoir une matrice dont la diagonale n'est pas dominante, ce qui entraîne à avoir une valeur propre nulle et par suite un déterminant nul. On peut conclure que notre matrice de corrélation n'est pas inversible. Ce cas n'a pas été pris en compte dans nos fonctions ce qui provoquait des erreurs.

ADQ	ADL	NBA	RL	AB
0.04552632	0.04310526	0.06889474	0.05847368	0.06744737

TABLE 1 – Taux d'erreur des méthodes d'analyse étudiées

Nous remarquons que l'analyse discriminante linéaire apporte le meilleur résultat de prédiction. Toutefois, les résultats des analyses quadratique et régression linéaire se différencie du meilleur résultat à un centième près. Le résultat de l'analyse bayésien naïf est le moins bon car ce dernier suppose l'indépendance des variables. Or l'analyse exploratoire de nos jeu de données montre qu'ils existent beaucoup de variables fortement corrélées et donc comme l'hypothèse d'indépendance est violée, ceci justifie pourquoi on obtient un taux d'erreur élevé par rapport aux autres classifieurs.

Les méthodes discriminantes suppose la normalité des hypothèses. Trouver une variable qui ne suit pas la loi normale permet de déduire que l'hypothèse de normalité est rejetée. Par contre, si chacune des variables suit une loi normale ceci ne permet pas de déduire la normalité de l'ensemble du jeu de données. Dans notre jeu de données, toutes les variables ne suivent pas la loi normale. Donc cette hypothèse est violée pour les analyses discriminantes. Pourtant on obtient de très bon résultats. En effet, avoir un bon résultat de prédiction ne veut pas dire que nous avons de bons résultats de modélisation. On ne peut donc pas faire le lien entre le résultat et l'hypothèse de normalité sur les données.

Il est aussi intéressant de comparer l'analyse discriminante linéaire et la régression, qui possèdent un hyperplan linéaire. On peut supposer que l'analyse linéaire a de meilleures résultats car elle c'est un analyseur simple, donc a besoin de moins de données.

1.2 Ionosphere

Ce jeu de données dispose de 350 individus et 35 variables. Ce jeu ne comporte pas de valeurs manquantes. La première variable présente des variables binaires et la deuxième variable présente une valeur constante.

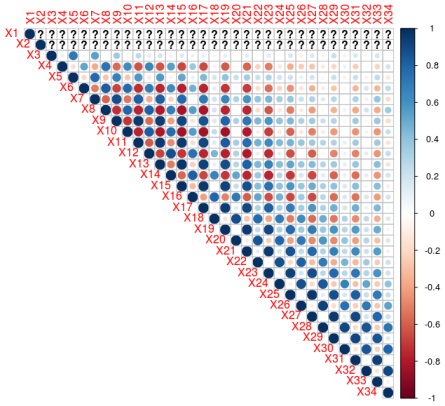


FIGURE 3 – Matrice de corrélation des variables de la classe 1

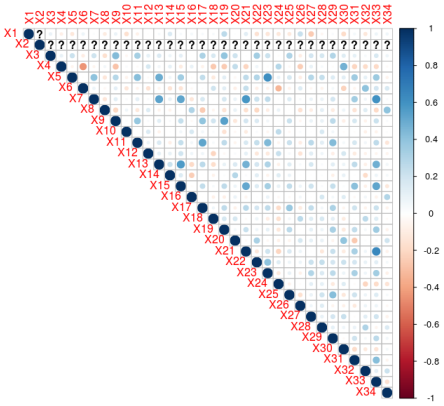


FIGURE 4 – Matrice de corrélation des variables de la classe 2

Nous avons analysé les matrices de corrélation des chacune des deux classes. Nous remarquons la présence de deux colonnes vide dans la première matrice, et une colonne vide dans la seconde. Ceci est due à la présence des constantes, l'écart-type de chaque colonne est donc nul. Or on sait que si une matrice possède une colonne nulle ou si elle possède deux colonnes identiques alors son déterminant est nul.

Les deux matrices ne sont donc pas donc inversible. Ce qui bloque notre analyse dans nos fonctions développés et les fonction natives de R au moment du calcul de la distribution des classes. On se permet alors d'éliminer les deux premières variables lors de l'analyse. Ceci ne changera pas nos résultats car on ne fait

que réduire le nombre de variables.

ADQ	ADL	NBA	RL	AB
0.1311966	0.1520513	0.1826496	0.1609402	0.1238889

TABLE 2 – Taux d'erreur des méthodes d'analyse étudiées

Les données ne suivent pas la loi normale, ceci affecte légèrement le résultats des analyses discriminantes. En effet, la vérification de cette hypothèse garantit uniquement l'optimalité des résultats, ce qui n'est pas le cas ici. De plus, nous remarquons la présence de corrélations très positives et négatives dans la première matrice, ceci explique le taux d'erreur le plus élevé pour l'analyse bayésien naïf qui suppose l'indépendance des variables.

L'analyse d'arbres binaires semble donner le meilleur taux d'erreur pour ce jeu de données car elle ne requière pas d'hypothèses sur les distributions des variables et semblent particulièrement adaptés au cas où les variables explicatives sont nombreuses (35 variables).

1.3 Sonar

Ce jeu de données dispose de 208 individus et 61 variables. Il n'existe pas de valeurs manquantes. On remarque que le nombre de données n'est pas grand par rapport aux nombre de variables.

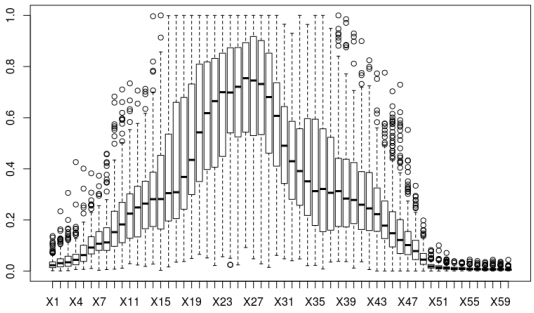


FIGURE 5 – Boites à moustaches des données

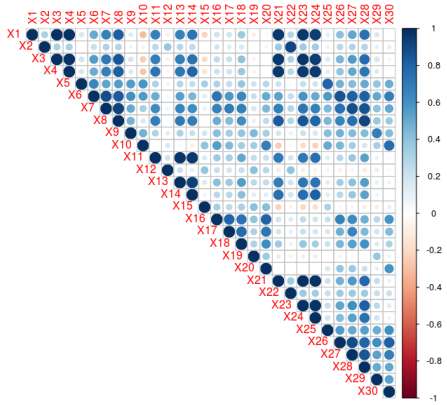


FIGURE 6 – Matrice de corrélation des variables

L'analyse sur ce jeu de données donnent de mauvais résultats par rapports aux tests des jeux précédents. Cette différence importante s'explique par le manque de données qu'on dispose par rapport aux variables.

ADQ	ADL	NBA	RL	AB
0.3802899	0.2742029	0.3169565	0.2989855	0.2721014

TABLE 3 – Taux d'erreur des méthodes d'analyse étudiées

Les meilleurs taux d'erreur sont donnés par l'analyse discriminante linéaire et l'arbre binaire.
Concernant le résultat de l'ADL, ceci est justifié par le fait qu'il s'agit d'un classifieur simple et n'a pas besoin d'un grand nombres de données par rapport aux variables. On peut également supposer que dans ce cas les individus sont dispersés de la même façon dans les deux classes, les matrices de covariance doivent donc être égales. Pour comparer ces matrices, il ne suffit pas uniquement de comparer l'égalité des matrices de covariance calculée à l'aide de la fonction cov(). L'idéal serait de comparer les matrices de diagonalisation,

comparer les valeurs propres mais aussi les matrices de passages. La vérification de ces égalités vérifient alors l'hypothèse de dispersion.

Par manque de temps cette méthode n'a pas été mise en place. Cependant, nous avons fait une ACP pour mieux visualiser les individus :

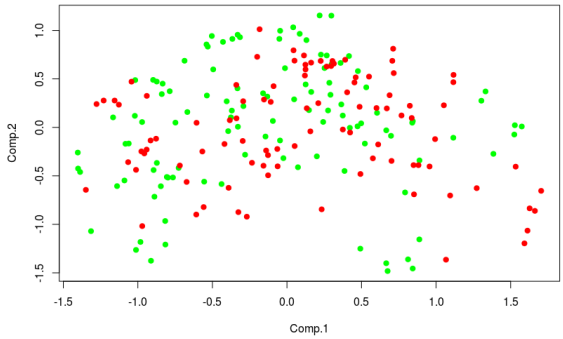


FIGURE 7 – Analyse des données en composantes principales

Nous remarquons que les données sont difficilement séparables, et donc rendent notre modèle plus complexe. Les classifieur trouve une difficulté à séparer les données, ce qui explique nos taux d'erreurs élevés. Le seul inconvénient de cette méthode est qu'elle dépend du taux d'inertie expliquée par chacune des composantes. Dans notre cas, l'inertie expliqué par l'axe 1 et 2 correspond à 52% environ. Ce taux reste faible et ne nous permet pas de conclure sur l'ensemble des données.

Le résultat de l'arbre binaire semble être meilleure que l'analyse linéaire à environ 0.002%. La raison est qu'il ne dispose pas d'hypothèse sur la distribution des variables et ne possède pas de paramètres à estimer.

1.4 Spambase

Ce jeu de données dispose de 4601 individus et 58 variables, et ne présente pas de valeurs manquantes. De plus, la plupart des variables ne possède pas de données atypiques.

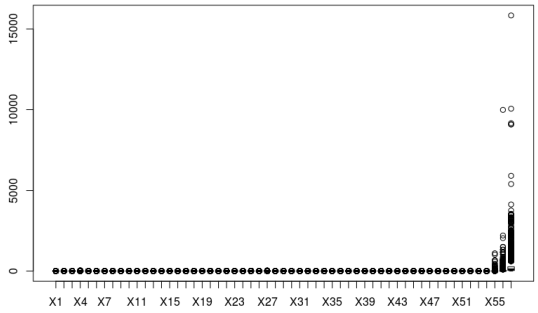


FIGURE 8 – Boites à moustaches des données

ADQ	ADL	NBA	RL	AB
Erreur	0.1118167	0.2891618	0.07667645	0.1003066

TABLE 4 – Taux d'erreur des méthodes d'analyse étudiées

De même que les données précédentes, Spambase ne suit pas une loi normale donc on sait que nos résultats ne sont pas optimales. De plus, l'analyse discriminante quadratique ne marche pas sur ces données. En effet, il s'agit d'un classifieur complexe et donc a besoin de plus de données pour estimer les paramètres. Il s'agit d'un problème de fléau de dimension.

L'arbre binaire donne le taux d'erreur le moins élevée. Nous pouvons lié ceci à l'absence presque totale de données atypiques. L'arbre binaire est donc sensible aux données atypiques. Une analyse plus profonde permet de visualiser la complexité de l'arbre obtenu. Cette complexité est lié à la profondeur de l'arbre qui ici, est de 7 niveaux.

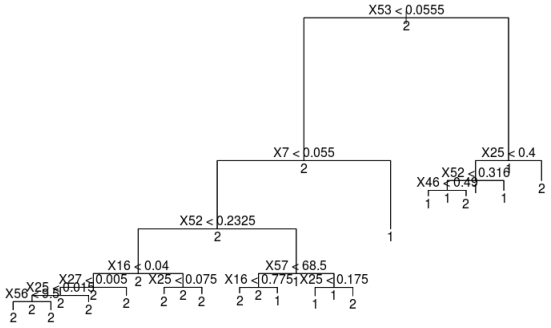


FIGURE 9 – Arbre de décision obtenu après élagage

1.5 Spambase 2

Il s'agit des valeurs binaires du jeu de donnée Spambase. Comme il s'agit de valeurs binaires, on peut confirmer l'absence de valeurs atypiques.

ADQ	ADL	NBA	RL	AB
Erreur	0.07212655	0.1274299	0.06265492	0.1121168

TABLE 5 – Taux d'erreur des méthodes d'analyse étudiées

Nous remarquons tout d'abord que l'analyse des données binaires donne de meilleurs résultats que les données précédentes. De plus, la régression linéaire donne le meilleur taux d'erreur. Ce résultat peut être expliqué par plusieurs raisons :

1. L'hypothèse de normalité est violée, et la régression linéaire n'a pas d'hypothèse sur la distribution de la loi.
2. La frontière de décision qui sépare mieux les deux classes semble être linéaire. C'est pourquoi nous observons de meilleurs résultat pour l'analyse linéaire et la régression (hyperplan linéaire) que les autres classifieurs.

2 Analyse discriminante de données binaires

2.1 Modèle

1. Chaque variable peut prendre la valeur 0 ou 1, on à donc à faire à une loi de Bernouilli. Le paramètre associé est donné, c'est pkj.

$$X^j \text{ B}(p_{kj}) \tag{1}$$

$$Pr(X^j = x_j | Z = w_k) = p_{kj}^{xj}(1 - p_{kj})^{1-xj} \tag{2}$$

2. La probabilité du vecteur x sachant la classe k est le produit de ses probabiulités marginales. Cela est rendu possible grâce à l'hypothèse d'indépendance des variables entre elles. On a alors :

$$Pr(X = x | Z = w_k) = \prod_{j=1}^{p} p_{kj}^{xj}(1 - p_{kj})^{1-xj} \tag{3}$$

3. La formule des probabilités jointe nous donnes

$$Pr(X = x_i; Z = z_i) = Pr(Z = z_i) \cdot Pr(X = x_i | Z = z_i) \tag{4}$$

On peut donc exprimer :

$$Pr(X = x_i; Z = z_i) = \prod_{k=1}^{g} Pr(Z = w_k)^{z_i k} \cdot \prod_{k=1}^{g} Pr(X = x_i | Z = w_k)^{z_i k} \tag{5}$$

4.

La vraissemblace jointe L s'exprime comme le produit des probabbilités jointes sur X et Z sachant les paramètre π_k et p_{kj} recherchés :

$$L(P(X = x_i; Z = z_i | (\pi_k, p_{kj}))) = \prod_{i=1}^{n} \prod_{k=1}^{g} \pi_k^{z_i k} \cdot \prod_{k=1}^{g} \left(\prod_{j=1}^{p} p_{kj}^{x_{ij}(1-p_{kj})^{1-x_{ij}}} \right)^{z_i k} \tag{6}$$

5. On commence par linéariser L :

$$\log L(P(X = x_i | Z = z_i | (\pi_k, p_{kj}))) = \sum_{i=1}^{n} \sum_{k=1}^{g} z_{ik}(ln(\pi_k) + \sum_{j=1}^{p} x_{ij} ln(p_{kj}) + (1 - x_{ij}) ln(1 - p_{kj})) \tag{7}$$

NB : pour le calcul des EMV on va se permettre de "supprimer" les sommes sur k et j car on ne cherche des EMV des estimateurs pour une seule classe ou pour une seule variable d'une seule classe.

On trouve la dérivée partielle sur π_k

$$\frac{\delta \log L(P(X = x_i; Z = z_i | (\pi_k, p_k)))}{\delta \pi_k} = \frac{\delta \sum_{i=1}^{n} z_{ik} ln(\pi_k)}{\delta \pi_k} = \frac{\delta ln(\pi_k) \cdot \sum_{i=1}^{n} z_{ik}}{\delta \pi_k} = \frac{1}{\pi_k} \cdot \sum_{i=1}^{n} z_{ik} \tag{8}$$

On reprend ensuite la démonstration page 115, chapitre 10 section 6.1 et on retrouve que sous la contrainte $\sum_{k=1}^{g} \pi_k = 1$

$$\frac{1}{n} \cdot \sum_{i=1}^{n} z_{ik} = \frac{n_k}{n} \tag{9}$$

Pour p_{kj}, on calculera :

$$\frac{\delta \log L(P(X = x_i; Z = z_i | (\pi_k, p_{kj})))}{\delta p_{kj}} = \frac{\delta \sum_{i=1}^{n} z_{ik}(x_{ij} ln(p_{kj}) + (1 - x_{ij}) ln(1 - p_{kj}))}{\delta p_{kj}} \tag{10}$$

$$= \sum_{i=1}^{n} z_{ik}(\frac{x_{ij}}{p_{kj}} + \frac{-(1 - x_{ij})}{1 - p_{kj}}) \tag{11}$$

$$= \sum_{i=1}^{n} z_{ik}\left(\frac{x_{ij}(1-p_{kj}) - p_{kj}(1-x_{ij})}{p_{kj}(1-p_{kj})}\right) \tag{12}$$

Alors :

$$\sum_{i=1}^{n} z_{ik}\left(\frac{x_{ij}(1-p_{kj}) - p_{kj}(1-x_{ij})}{p_{kj}(1-p_{kj})}\right) = 0 \tag{13}$$

$$\Leftrightarrow \sum_{i=1}^{n} z_{ik}(x_{ij} - p_{kj}) = 0 \tag{14}$$

$$\Leftrightarrow \sum_{i=1}^{n} z_{ik}x_{ij} = \sum_{i=1}^{n} z_{ik}p_{kj} \tag{15}$$

$$\Leftrightarrow \sum_{i=1}^{n} z_{ik}x_{ij} = p_{kj}\sum_{i=1}^{n} z_{ik} \tag{16}$$

$$\Leftrightarrow p_{kj} = \frac{\sum_{i=1}^{n} z_{ik}x_{ij}}{\sum_{i=1}^{n} z_{ik}} = \frac{1}{n_k}\sum_{i=1}^{n} z_{ik}x_{ij} \tag{17}$$

2.2 Programmation

Le programme est construit comme demandé en deux fonctions principales : *binaryNBCfit* qui gère l'apprentissage et *binaryNBCval* qui teste les paramètres du modèle sur un autre ensemble. Enfin, la fonction *classifier* permet le prétraitement des données.

binaryNBCfit reçoit l'ensemble d'apprentissage et pour chaque classe calcule les EMV comme décrit dans la partie précédente. La fonction renvoie l'objet params qui contient les π_k et les pk_j.

binaryNBCval prend en argument l'ensemble de test et les *params* calculés précédemment. On commence par calculer pour chaque x $P(X|Z)$. On multiplie ensuite par les π_k pour obtenir une "pré-probabilité". A ce moment là on peut déjà créer le vecteur des prédictions. On finit par diviser les "pré-probabilités" par la somme des $\pi_k * P(X|Z)$ afin d'obtenir de vrais probabilités qui seront retournées.

Enfin nous allons parler des traitements préalables à l'application des deux fonctions précédentes. On commence par mélanger les *rows* dataframe et on le sépare ensuite en deux, l'ensemble d'apprentissage et l'ensemble de test. Cela nous assure (presque toujours) d'avoir des exemples de toutes les classes dans les deux ensembles. Enfin, on extrait les label Z et les numéros de ligne de la première colonne. On peut ensuite appeler les deux fonctions.

Le code décrit ci-dessus est consultable en annexe. (cf. annexe A).

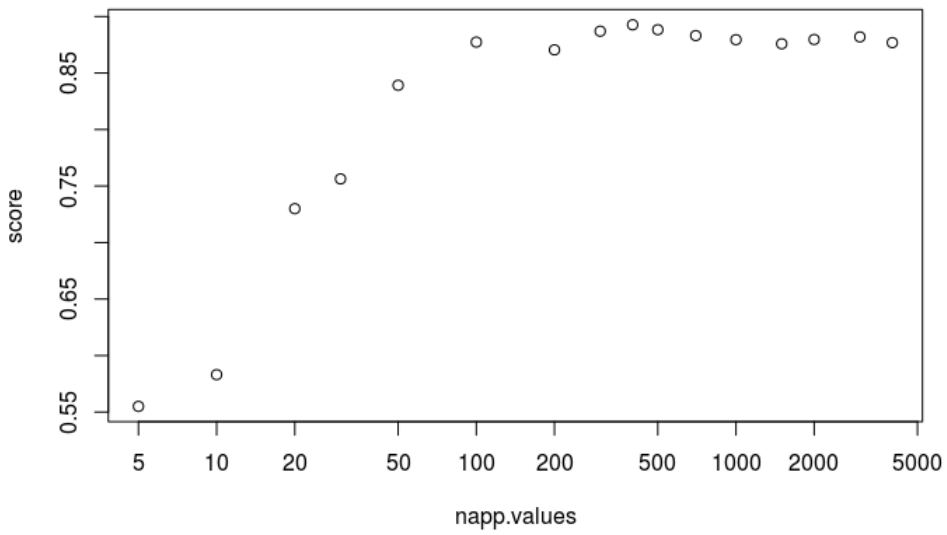
2.3 Test

Pour tester notre classifieur, nous avons utilisé plusieurs *napp* différents.

```
score <- c()
napp.values <- c(5, 10, 20, 30, 50, 100, 200, 300 ,400, 500, 700, 1000, 1500, 2000, 3000,
    4000)
for (i in napp.values) {
  score <- c(score, classifier(napp = i))
}
plot(score, x = napp.values, log = 'x')
```

Listing 1 – Code utilisé pour trouver le napp suffisant à un apprentissage correct

Cela nous renvoie le graphique suivant (échelle logarithmique) :



On remarque que l'augmentation du *napp* fait évoluer la précision du classifieur de manière logarithmique jusqu'à atteindre un score "plafond" autour de 0.89. On arrive à ce score avec 300 exemples d'apprentissage. Le taux de réussite est moins bon que le taux donné par la régression logistique sur ce même jeu de données (0.93734508%). Pourtant, notre algorithme a été conçue pour répondre à un jeu de données binaires. Un phénomène de sur-apprentissage peut expliquer ce résultat.

A Fonctions de l'analyse discriminante des données binaires

```r
#Apprentissage
binaryNBCfit <- function(Xapp, zapp) {
  K <- length(unique(zapp))
  #Efefectif par classe
  pi_ks <- c()
  #prob a posteriori
  p_kjs <- c()
  #Pour chaque classe : on extrait les individus appartenant a la classe puis on calcule les EMV
  for (k in 1:K){
    Xk <- Xapp[which(zapp == k),]
    pi_k <- dim(Xk)[1] / dim(Xapp)[1]
    p_k <- colSums(Xk) / dim(Xk)[1]
    pi_ks <- c(pi_ks, pi_k)
    p_kjs <- c(p_kjs, p_k)
  }
  p_kjs <- matrix(p_kjs, nrow = dim(Xapp)[2], ncol = length(pi_ks), byrow = FALSE)
  return( list(pi_ks = pi_ks, p_kjs = p_kjs))

}
```

Listing 2 – Apprentissage du modèle

```r
#Test
binaryNBCval <- function(Xtst, params) {
  #Nb indicvidus
  n <- dim(Xtst)[1]
  #Nb Variables
  p <- dim(Xtst)[2]
  #Nb classes
  g <- length(params$pi_ks)

  prob <- matrix(0, ncol=g, nrow = n)
  p_kjs <- params$p_kjs

  #Pour chaque classe k on calcule P(Z = w_k) * P(X = x | Z = w_k)
  for(k in 1:g) {
    temp <- apply(Xtst, 1, function(x) {
      prod((p_kjs[,k] ** x) * ((1 - p_kjs[,k]) ** (1-x)))
    })
    prob[,k] <- params$pi_ks[k] * temp
  }
  pred <- max.col(prob)
  #Comme vu en TD, on divise par la somme de toutes les classes afin d'obtenir une probabilite d'appartenance pour chaque classe

  prob <- prob / rowSums(prob)

  #prob[which(is.na(test$prob))] <- c(1, 0)

  return(list(prob = prob, pred = pred))
}
```

Listing 3 – Test du modèle sur des données de test

```r
classifier <- function(file = "donnees/donnees/spambase2.csv", napp = 100) {

  #Chargement du CSV
  spambase2 <- read.csv(file)

  #On randomise le CSV afin d'avoir des individus de chaque classe
  spambase2 <- spambase2[sample(nrow(spambase2), nrow(spambase2), replace = FALSE),]
  row.names(spambase2) <- NULL

```

```
  #On separe la base obtenue en deux echantillons : Xapp Zapp et Xtest Ztest
  Xapp <- spambase2[1:napp,]
  row.names(Xapp) <- NULL

  Zapp <- Xapp[,dim(Xapp)[2]]
  #On retire les colones des numeros de lignes et d'etiquetage
  Xapp <- Xapp[, -c(1, dim(Xapp)[2])]

  #Entrainement du classificateur
  params <- binaryNBCfit(Xapp[1:napp,], Zapp[1:napp])

  #On fait de meme pour Xtest
  Xtst <- spambase2[(napp + 1):dim(spambase2)[1],]
  row.names(Xtst) <- NULL

  Ztst <- Xtst[, dim(Xtst)[2]]
  Xtst <- Xtst[, -c(1, dim(Xtst)[2])]

  #On va tester le classificateur avec le parametre obtenus precedemment
  test <- binaryNBCval(Xtst, params)
  #Pour le taux d'erreur, on s'appuie sur la taille du tableau des donnees mal etiquetees
  err <- Ztst != test$pred

  return (1 - (length(which(err))/(dim(Xtst)[1])))
}
```

Listing 4 – Pré-traitement des données, utilisation des fonctions et calcul des performances du modèle