SORBONNE UNIVERSITES

21 juin 2017

TP 4 - Discrimination

UV : **SY09**
Branche : **Génie Informatique**
Filière : **Fouille de Données et Décisionnel**
Auteurs : **LU Han - HAMONNAIS Raphaël**

Table des matières

1. Programmation

1.1 Analyse discriminante

On se situe ici dans le cadre binaire : on considère $g = 2$ classes.

1.1.1 Apprentissage

A chaque modèle d'analyse discriminante correspond une fonction d'apprentissage :

- `adq.app` pour l'analyse discriminante quadratique ;
- `adl.app` pour l'analyse discriminante linéaire ;
- `nba.app` pour le classifieur bayésien naïf.
- **Remarque** : n'ayant pas vu que des fonctions « à compléter » (même si c'était clairement dit dans le sujet) étaient disponibles, nous avons implémenté nos propres fonctions dont vous trouverez le code en annexe A.

Prototype des fonctions `nba.app`, `adq.app` et `adl.app`

- Objectif :
  - Apprendre les paramètres (proportions, vecteurs de moyennes et matrices de covariance des deux classes) du modèle considéré.
- Paramètres en entrée :
  - `Xapp` : Les données d'apprentissage, de taille $n \times p$;
  - `zapp` : Les étiquettes des données d'apprentissage, de taille n ;
- Sortie :
  - Une structure `params` contenant les paramètres du modèle :
    - `params$pi` : liste des proportions de chaque classe ;
    - `params$mu` : liste des vecteurs de moyennes de chaque classe ;
    - `params$sigma` : liste des matrices de covariance de chaque classe ;
- Calcul des paramètres :
  - Proportions `pi` identiques pour les trois modèle, égales à $\frac{n_k}{n}$;
  - Vecteurs de moyennes `mu` identiques pour les trois modèle, égaux à $\overline{x_k} = \frac{1}{n_k} \sum_{i=1}^{n} z_{ik} x_i$ avec $z_{ik} = 1$ si $x_i \in Classe\ k$, 0 sinon ;
  - Matrices de covariance `sigma` diffèrent selon les modèles :
    - AD Quadratique : chaque classe k a sa propre matrice de covariance V_k ;
    - AD Linéaire : la matrice de covariance est commune à toutes les classes et vaut $\Sigma = \frac{1}{n} \sum_{k=1}^{g} n_k V_k$;
    - Classifieur bayésien naïf : hypothèse d'indépendance conditionnelle des variables, on a alors Σ_k égale à la matrice diagonale obtenue à partir des valeurs diagonales de la matrice de covariance V_k, soit `sigma_k = diag(diag(V_k))` en code `R` ;

Le code des fonctions d'apprentissage est disponible en annexe A.1.

1.1.2 Discrimination

Prototype de la fonction de discrimination `ad.val`

» Objectif :

— Calculer les probabilités à postériori des individus de test pour chaque classe et effectuer le classement en fonction de ces probabilités.

» Paramètres en entrée :

— `params` : La structure contenant les paramètres du modèle utilisé pour effectué la discrimination ;

— `Xtst` : Les données de test à classer.

» Sortie :

— Une structure `discrimination` contenant les probabilités à postériori et le classement associé :

— `discrimination$pw1` : les probabilités à postériori pour la classe 1 ;

— `discrimination$pw2` : les probabilités à postériori pour la classe 2 ;

— `discrimination$pred` : le classement des données `Xtst` dans la classe 1 ou 2 ;

» Calcul des probabilités à postériori :

— Utilisation de la formule $\mathbb{P}(\omega_k|x) = \frac{f_k(x)\pi_k}{f(x)} = \frac{f_k(x)\pi_k}{f_1(x)\pi_1+f_2(x)\pi_2}$;

— Utilisation de la fonction `mvdnorm` afin de calculer $f_1(x)$ et $f_2(x)$:

— $f_1(x) = mvdnorm(Xtst, \mu_1, \Sigma_1)$

— $f_2(x) = mvdnorm(Xtst, \mu_2, \Sigma_2)$

— Remarquons que dans notre cas, avec 2 classes, $\mathbb{P}(\omega_2|x) = 1 - \mathbb{P}(\omega_1|x)$.

» Détermination de la classe :

— Si $P(\omega_1|x) \geq P(\omega_2|x)$, alors x appartient à la classe 1, sinon à la classe 2.

Le code de la fonction de discrimination est disponible en annexe A.2.

1.2 Régression logistique

1.2.1 Apprentissage

On rappelle qu'on est dans un cas binaire, c'est à dire avec $g = 2$ classes.

Alors

$$\mathbf{t} = (t_1, ..., t_n)^t \quad tel\ que \quad t_i = \begin{cases} 1 & si\ Z_i = \omega_1 \\ 0 & si\ Z_i = \omega_2 \end{cases}$$

$$\mathbf{p} = p(x;\beta) = \mathbb{P}(\omega_1|x)$$
$$\mathbf{1\text{-}p} = 1 - p(x;\beta) = \mathbb{P}(\omega_2|x)$$

Notons pour plus de simplicité

$$\mathbf{p_i} = p(x_i;\beta) = \mathbb{P}(\omega_1|x_i)$$
$$et \quad \mathbf{p} = (p_1, ..., p_n)^t$$

Calcul des probabilités à postériori avec la fonction `postprob`

» Objectif :

— Calculer les probabilités à postériori de la classe 1 $\mathbb{P}(\omega_1|x)$.

» Paramètres en entrée :

— `beta` : estimateur du maximum de vraisemblance $\hat{\beta}$ des paramètres du modèle ;

— `X` : données dont on veut calculer $\mathbb{P}(\omega_1|x)$.

» Sortie :

— Le vecteur des probabilités $\mathbb{P}(\omega_1|x)$.

On utilise la formule suivante :

$$\mathbb{P}(\omega_1|x) = \frac{exp(\beta^t x)}{1 + exp(\beta^t x)}$$

Le code de la fonction `postprob` est disponible en annexe B.1.

Prototype de la fonction `log.app`

» Objectif :

— Apprendre les paramètres (matrice beta) du modèle de régression logistique à l'aide de l'algorithme de Newton-Raphson.

» Paramètres en entrée :

— `Xapp` : les données d'apprentissage, de taille $n \times p$;

— `zapp` : les étiquettes des données d'apprentissage, de taille n ;

— `intr` : variable binaire indiquant s'il faut ou non ajouter une ordonnée à l'origine (*intercept*) à la matrice `Xapp` ;

— `epsi` : scalaire correspondant au seuil ϵ en-deça duquel on considère que l'algorithme d'apprentissage a convergé.

» Sortie :

— Une structure `params` contenant les paramètres du modèle :

— `params$beta` : estimateur du maximum de vraisemblance $\hat{\beta}$ des paramètres, de dimensions $p \times 1$ (ou $(p+1) \times 1$ si une ordonnée à l'origine a été ajoutée) ;

— `params$niter` : le nombre d'itération effectuées par l'algorithme de Newton-Raphson avant convergence ;

— `params$logL` : valeur de la vraisemblance à l'optimum.

Calcul des paramètres β

Estimateur du maximum de vraisemblance $\hat{\beta}$ des paramètres :

» La méthode de Newton-Raphson consiste à approximer β par développement limité :

— On choisit un point de départ $\beta_{(q)}$, ($\beta_{(0)}$ par exemple) ;

— On calcule $\nabla \ln L_{(q)} = \frac{\partial \ln L(\beta_{(k)})}{\partial \ln L}$;

— On calcule $H_{(q)} = \frac{\partial^2 \ln L(\beta_{(k)})}{\partial \beta \partial \beta^t}$;

— On calcule $\beta_{(q+1)}$ en fonction de $\beta_{(q)}$, $\nabla \ln L_{(q)}$ et $H_{(q)}$;

  - On itère jusqu'à la convergence définie par la valeur ϵ (c'est à dire $\nabla \ln L$ proche de zéro, donc maximum de vraisemblance atteint).

» Équation de mise à jour de β

  - $\beta_{(q+1)} = \beta_{(q)} - H_{(q)} \nabla \ln L_{(q)}$;
  - Avec $\nabla \ln L_{(q)} = X^t(t - p_{(q)})$ où $p_{(q)} = p(x; \beta_{(q)})$
  - Avec $H_{(q)} = -X^t W_{(q)} X$ où $W_{(q)} = diag\left(p_{(q)} \ (1 - p_{(q)}) \right)$

Valeur de la vraisemblance à l'optimum :

»

Le code de la fonction d'apprentissage `log.app` est disponible en annexe B.2.

1.2.2 Discrimination

Prototype de la fonction `log.app`

» Objectif :

  - Classer les individus dans leur classes respectives en fonction des probabilités à postériori de chaque classe.

» Paramètres en entrée :

  - `beta` : estimateur du maximum de vraisemblance $\hat{\beta}$ des paramètres du modèle ;
  - `Xtst` : données à discriminer de taille $n \times p$.

» Sortie :

  - Une structure `discrimination` contenant les probabilités à postériori et le classement associé :
    - `discrimination$prob` : matrice de taille $n \times 2$ les probabilités à postériori des classes 1 et 2 ;
    - `discrimination$pred` : vecteur de taille n contenant le classement des données `Xtst` dans la classe 1 ou 2 ;

» Détermination de la classe :

  - Si $P(\omega_1|x) \geq P(\omega_2|x)$, alors x appartient à la classe 1, sinon à la classe 2.

Le code de la fonction de discrimination `log.val` est disponible en annexe B.3.

1.2.3 Régression Logistique Quadratique

Cette méthode consiste à transformer l'espace des données en un espace plus complexe (contenant plus de variables) et à appliquer la régression logistique sur ce nouvel 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.

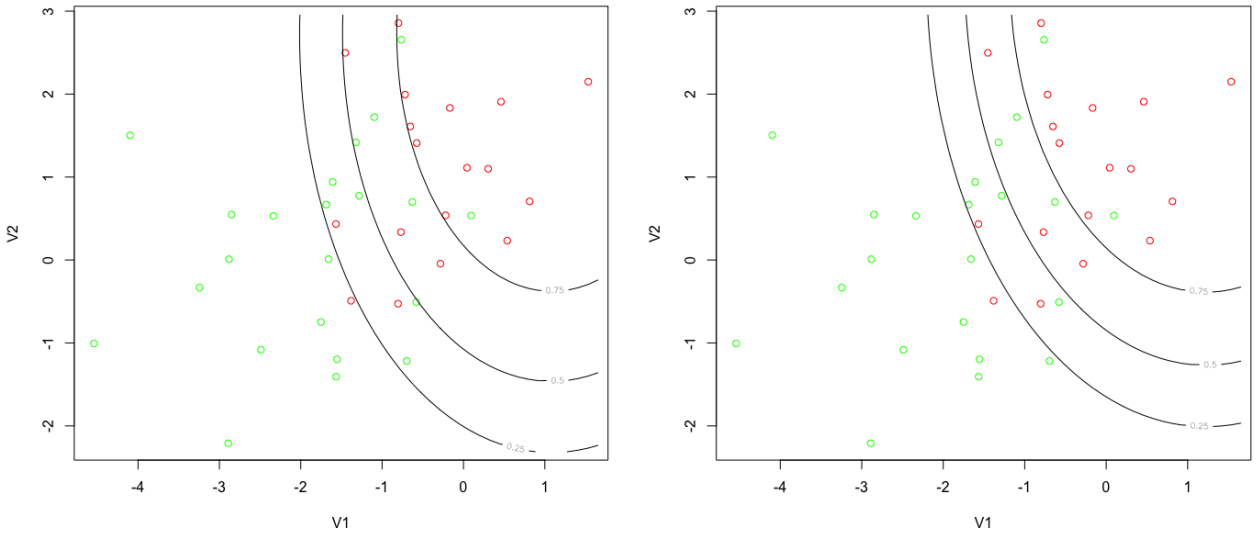
Par exemple, dans le cas où les individus sont décrits par les variables naturelles X1, X2 et X3, la régression logistique quadratique consiste à apprendre un modèle de régression logistique classique dans l'espace $\mathbf{X}^2 = \{X^1, X^2, X^3, X^1X^2, X^1X^3, X^2X^3, (X^1)^2, (X^2)^2, (X^3)^2\}$, plutôt que dans l'espace $\mathbf{X} = \{X^1, X^2, X^3\}$.

La dimension du jeu de données p devient alors $p(p+3)/2$.

Le code de la fonction `log_quad` transformant l'espace originel en un espace quadratique est disponible en annexe B.4.

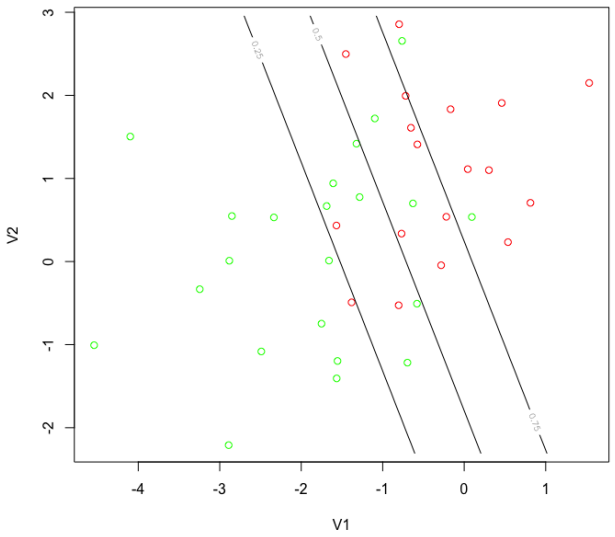
1.3 Vérification des fonctions

1.3.1 Analyse discriminante



(a) A. D. Quadratique

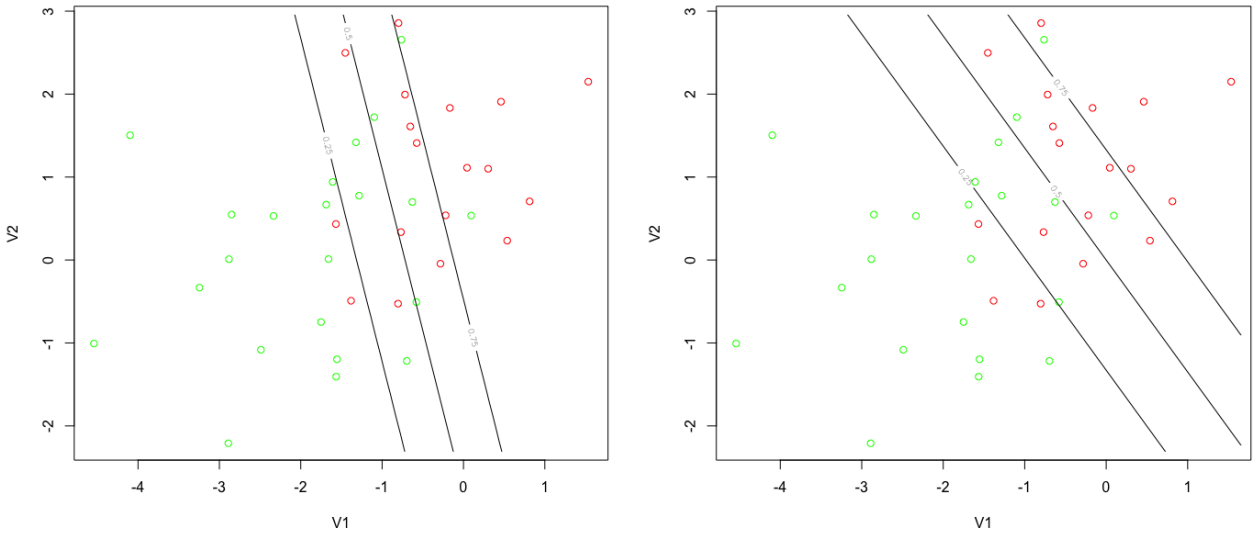
(b) Classifieur bayésien naïf



(c) A. D. Linéaire

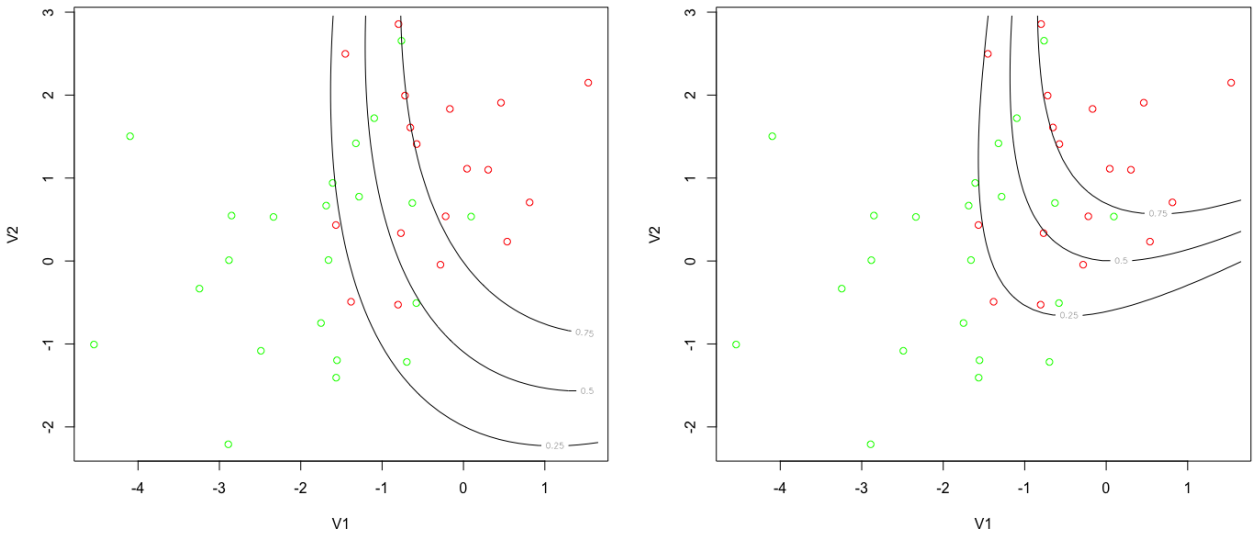
FIGURE 1.1 – Frontières de décision pour l'analyse discriminante

1.3.2 Régression logistique



(a) Avec *intercept* (b) Sans *intercept*

FIGURE 1.2 – Frontières de décision pour la régression logistique



(a) Avec *intercept* (b) Sans *intercept*

FIGURE 1.3 – Frontières de décision pour la régression logistique quadratique

2. Application

On va s'attacher ici à comparer les performances de nos cinq classifieurs :

- Analyse discriminante quadratique, linéaire et bayésien naïf ;
- Régression logistique simple et quadratique ;
- Arbre de décision.

Il est important de bien noter les différence entre ces familles de classifieur, et plus particulièrement entre l'analyse discriminante et les deux autres.

L'analyse discriminante ***suppose*** que les données, conditionnellement à chaque classe ω_k, suivent une loi normale multidimensionnelle d'espérance μ_k et de variance Σ_k. En faisant ensuite différentes ***hypothèses sur ces paramètres***, on obtient différentes expressions de la règle de Bayes, d'où l'on déduit différentes probabilités à postériori et donc différentes règles de décision en remplaçant les paramètres théoriques par leurs estimations. Dans le cas qui nous intéresse, seules les hypothèses sur les matrices de variances changent en fonction des classifieurs, et donc leurs estimations changent :

- AD Quadratique : chaque classe k a sa propre matrice de covariance V_k ;
  - Ce classifieur sera plus fidèle aux les données d'apprentissage et donnera une frontière de décision quadratique ;
  - Il pourra donc être très efficace dans le cas de classes à distribution quadratique mais risque de faire du sur-apprentissage dans le cas où les données d'apprentissage ne sont pas représentatives des données réelles.
- AD Linéaire : la matrice de covariance est commune à toutes les classes et vaut $\Sigma = \frac{1}{n}\sum_{k=1}^{g} n_k V_k$;
  - Ce classifieur fait une supposition forte qui peut l'amener a avoir de moins bonne performances que l'AD quadratique ;
  - Par contre, la frontière de décision obtenue étant linéaire, elle sera plus robuste et donc moins sensible au bruit des données (i.e. données d'apprentissage non représentatives).
- Classifieur bayésien naïf : hypothèse d'indépendance conditionnelle des variables, on a alors Σ_k égale à la matrice diagonale obtenue à partir des valeurs diagonales de la matrice de covariance V_k.
  - Ce classifieur propose des frontières de décision quadratique, tout comme l'A.D. quadratique ;
  - Par contre, l'hypothèse d'indépendance conditionnelle des variables suppose que les classe sont des ellipsoïde dont les axes sont parallèles aux axes du repère.

Dans le cas où les données ***respectent effectivement les hypothèses avancées*** par les classifieurs d'analyse discriminante, alors il est fort probable que ces derniers présentent de bonnes performances. En effet, les estimations des probabilités à postériori seront d'autant plus précises que les hypothèses portant sur la distribution des données sont vérifiées.
Mais cela n'est pas obligatoire : des classes dont les espérances μ_k sont très proches seront dans la plupart des cas très difficiles à discriminer.

La régression logistique ***ne fait pas d'hypothèses sur les distributions conditionnelles*** $\mathbf{f_k}$: elle estime directement les probabilités à postériori $\mathbb{P}(\omega_k|x)$ d'appartenance aux classes.
La régression logistique simple donnera une frontière de décision linéaire. Lorsqu'on change la dimension des données en rajoutant des variables qui sont des combinaisons des variables originales, la régression logistique devient quadratique et on pourra alors avoir un hyperplan comme frontière de décision.

2.1 Test sur les données simulées

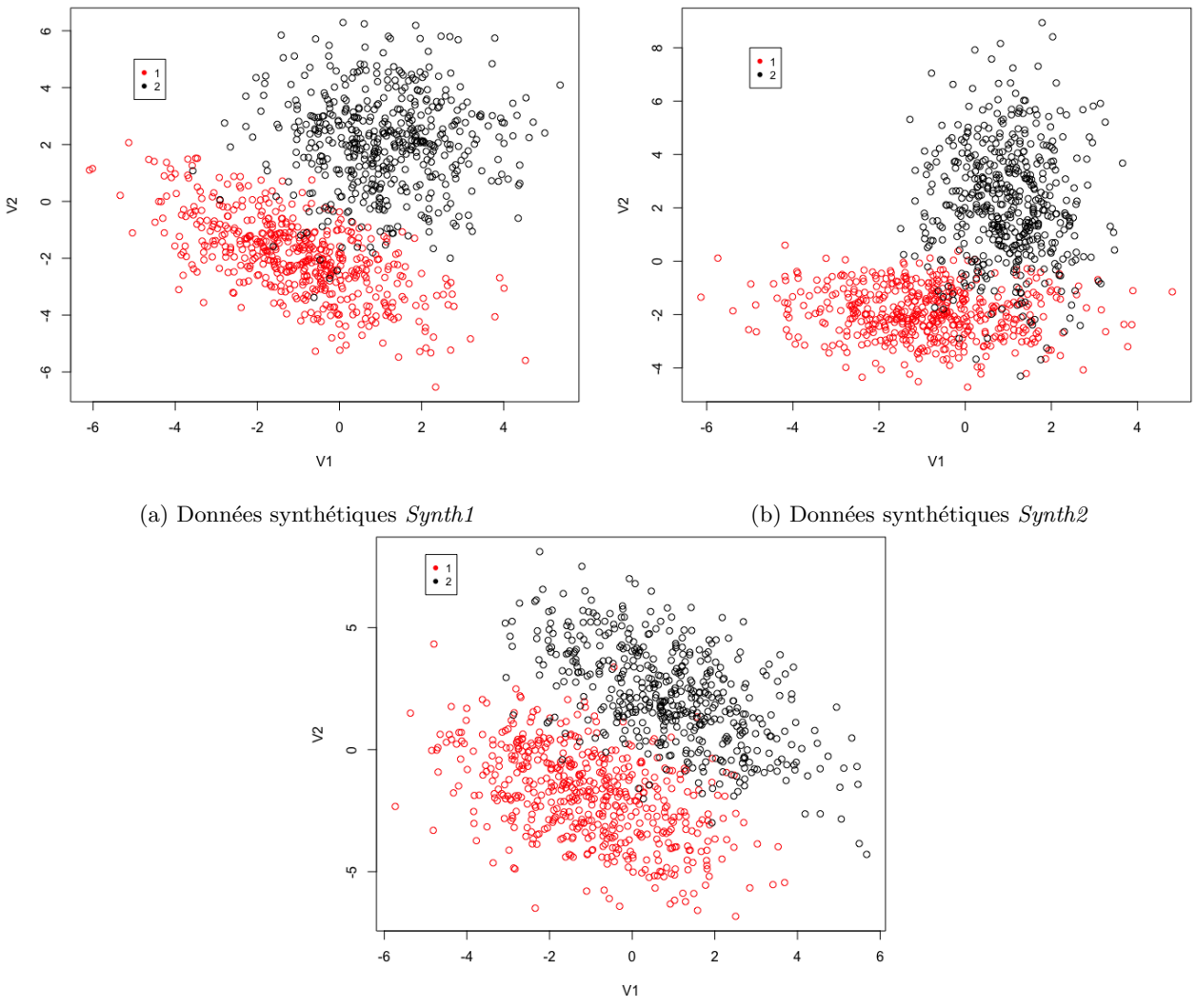
On souhaite comparer les performances de l'analyse discriminante, de la régression logistique (linéaire et quadratique), et des arbres de décision sur les jeux de données simulées *Synth1*, *Synth2* et *Synth3*.
Pour ce faire, on répétera $N = 20$ fois pour chaque jeu de données :

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

Pour chaque jeu de données, on calculera ensuite le taux d'erreur (de test) moyen $\hat{\epsilon}$ sur les $N = 20$ séparations effectuées.

2.1.1 Visualisation graphique des données

Les données synthétiques sont de dimension $p = 2$, ce qui nous permet d'en tracer une visualisation graphique simple. Cette visualisation nous donnera des intuitions quand aux modèles de discrimination qui pourront être efficaces ou non sur ces données.



(a) Données synthétiques *Synth1*

(b) Données synthétiques *Synth2*

(c) Données synthétiques *Synth3*

FIGURE 2.1 – Visualisation des données synthétiques *Synth1*, *Synth2* et *Synth3*

Remarques possibles suite à la visualisation :

» Données *Synht1* :

— Classes dont les variances Σ semblent différentes, Σ_1 donnant une forme d'ellipsoïde non parallèle aux axes quand Σ_2 semble définir une classe plutôt circulaire ;

— Il semble y avoir peu de chevauchement entre les classes ;

— Compte tenu de leurs formes, un classifieur à frontière de décision quadratique tendra à être plus efficace qu'un classifieur à frontière de décision linéaire.

» Données *Synht2* :

— Classes dont les variances Σ semblent inversées : la classe ω_1 est très étendues sur l'axe V1 quand la classe ω_1 est très étendue sur l'axe V2 ;

— Le chevauchement entre les classes est plus important que dans les deux autres jeux de données ;

— De façon plus prononcée que pour *Synth1* ou *Synth3*, un classifieur à frontière de décision quadratique sera plus performant qu'un classifieur à frontière de décision linéaire.

» Données *Synht3* :

— Classes dont les variances Σ semblent identiques ;

— Peu de chevauchement entre les classes ;

— L'analyse discriminante linéaire semble particulièrement indiquée ici. On peut même s'attendre à ce que les frontières de décision quadratiques aient une forme quasiment linéaire.

» Classifieur des arbres de décision :

— Les classes se présentant visuellement sous forme d'ellipsoïdes (pas toujours parallèles aux axes), ce classifieur – proposant un frontière de décision linéaire par morceaux **et** parallèle aux axes des dimensions (= variables) – pourra logiquement donner de plus mauvais résultats que les autres sur ces jeux de données.

2.1.2 Visualisation « mathématique » des données via l'estimation des paramètres

Fichier source	Centres de gravité μ	Variances Σ	Proportions π
Synth1	$\mu_1 = \begin{pmatrix} -1.03 \\ -1.95 \end{pmatrix}$, $\mu_2 = \begin{pmatrix} 1.10 \\ 2.07 \end{pmatrix}$	$\Sigma_1 = \begin{pmatrix} 2.89 & -1.53 \\ -1.53 & 1.96 \end{pmatrix}$, $\Sigma_2 = \begin{pmatrix} 2.09 & 0.00 \\ 0.00 & 2.81 \end{pmatrix}$	$\begin{pmatrix} \pi_1 = 0.51 \\ \pi_2 = 0.49 \end{pmatrix}$
Synth2	$\mu_1 = \begin{pmatrix} -0.90 \\ -1.92 \end{pmatrix}$, $\mu_2 = \begin{pmatrix} 0.97 \\ 2.17 \end{pmatrix}$	$\Sigma_1 = \begin{pmatrix} 2.81 & -0.19 \\ -0.19 & 0.90 \end{pmatrix}$, $\Sigma_2 = \begin{pmatrix} 0.90 & -0.04 \\ -0.04 & 4.60 \end{pmatrix}$	$\begin{pmatrix} \pi_1 = 0.5 \\ \pi_2 = 0.5 \end{pmatrix}$
Synth3	$\mu_1 = \begin{pmatrix} -1.06 \\ -1.91 \end{pmatrix}$, $\mu_2 = \begin{pmatrix} 0.92 \\ 2.19 \end{pmatrix}$	$\Sigma_1 = \begin{pmatrix} 2.88 & -1.55 \\ -1.55 & 3.50 \end{pmatrix}$, $\Sigma_2 = \begin{pmatrix} 2.92 & -2.02 \\ 2.02 & 4.17 \end{pmatrix}$	$\begin{pmatrix} \pi_1 = 0.52 \\ \pi_2 = 0.48 \end{pmatrix}$

TABLE 2.1 – Paramètres estimés des fichiers *Synth1*, *Synth2* et *Synth3*

Remarques :

» Données *Synht1* :

— Comme nous en avons eu l'intuition précédemment lors de la visualisation, on a bien $\Sigma_1 \neq \Sigma_2$ avec ω_1 légèrement plus dispersée sur l'axe V1 après rotation et ω_2 légèrement plus dispersée sur l'axe V2 (et non circulaire comme on l'avait supposé) ;

— Compte tenu des paramètres estimés, la classification discriminante quadratique sera plus indiquée que la linéaire ou le classifieur bayésien naïf, car leurs hypothèses ne sont pas vérifiées ;

— Par équivalence, la régression logistique quadratique sera plus indiquée que la régression logistique simple.

» Données *Synht2* :

— Compte tenu des paramètres estimés, le classifieur bayésien naïf sera plus indiqué que la discrimination linéaire ou quadratique, car il semble bien qu'on ait l'hypothèse d'indépendance conditionnelle des variables vérifiée (covariances nulles ou insignifiantes dans les matrices Σ_k) ;

— Par équivalence, la régression logistique quadratique sera plus indiquée que la régression logistique simple (le classifieur bayésien naïf donne une frontière de décision quadratique).

» Données *Synht3* :

— Compte tenu des paramètres estimés, l'analyse discriminante linéaire semble la plus indiquée. Même si les matrices Σ_1 et Σ_2 ne sont pas exactement égales entre-elles, elles sont tout de même relativement proches ;

— Par équivalence, la régression logistique simple sera plus indiquée que la régression logistique quadratique.

2.1.3 Frontières de décisions

Des exemples de frontières de décisions sont disponibles en annexe :

» Données *Synht1* : voir section D.1 ;

» Données *Synht2* : voir section D.2 ;

» Données *Synht3* : voir section D.3.

2.1.4 Calculs des pourcentages d'erreur des classifieurs

Remarque sur l'espérance et la variance du pourcentage d'erreur $\hat{\epsilon}$ *: on va par la suite calculer ces statistiques à des fins analytiques. Il faut prendre note que nous allons les calculer sur un échantillon de taille* $n = 6$*, voire moins lorsqu'on comparera les différents types de classifieur. Bien que donnant certaines indications et servant notre propos lors des comparaisons, ces statistiques doivent être considérées avec prudence.*

Calcul du pourcentage d'erreur $\hat{\epsilon}$ sur les données synthétiques :

Fichier source	ADQ	ADL	NBA	Log	Log Quad	Tree
Synth1	**3.43**	4.52	4.22	3.67	**3.25**	4.73
Synth2	6.23	8	**6.17**	6.85	**6.44**	8.09
Synth3	4.23	**4.16**	5.45	**4.2**	4.37	5.93

TABLE 2.2 – Estimation du pourcentage d'erreur $\hat{\epsilon}$ sur $N = 20$ itérations des différents classifieurs sur les données synthétiques

Comme on avait pu le supposer précédemment lors de l'analyse des visualisations graphiques et des paramètres des classes, on a bien pour :

» Données *Synht1* :

— De meilleures performances avec l'analyse discriminante quadratique et la régression logistique quadratique.

» Données *Synht2* :

— De meilleures performances avec le classifieur bayésien naïf et la régression logistique quadratique.

» Données *Synht3* :

— De meilleures performances avec l'analyse discriminante linéaire et la régression logistique simple.

» Classifieur des arbres de décision :

— Les plus mauvaises performance pour chaque jeu de données.

Il faut néanmoins nuancer nos propos car les performances des classifieurs par fichier sont à première vue sensiblement les mêmes (faible variance) :

Fichier source	Espérance de $\hat{\epsilon}$	Variance de $\hat{\epsilon}$
Synth1	3.97	0.37
Synth2	6.96	0.76
Synth3	4.72	0.59

TABLE 2.3 – Espérances et variances des pourcentages d'erreurs de tous les classifieurs pour un fichier synthétique donné

Remarque *: dans la suite de l'analyse des performances des classifieurs, on laissera de côté l'arbre de décision. Ce dernier propose une frontière linéaire parallèle aux axes « par morceaux » (voir frontières de décision en annexe Figure D.1f, Figure D.2f et Figure D.3f) et donne les résultats les plus mauvais, comparativement aux autres classifieurs.*
Les données étant distribuées de façon ellipsoïdale, il n'est pas étonnant que ce soit ce classifieur qui présente la plus mauvaise performance.

Nous nous proposons maintenant de faire une analyse comparative des performances en deux parties :

» les modèles d'analyse discriminante comparés à la régression logistique ;

» les modèles à frontière de décision linéaire comparés aux modèles à frontière de décision quadratique.

Comparaison du pourcentage d'erreur $\hat{\epsilon}$ entre analyse discriminante et régression logistique

Fichier source	AD : $\mathbb{E}[\hat{\epsilon}]$	AD : $Var[\hat{\epsilon}]$	RL : $\mathbb{E}[\hat{\epsilon}]$	RL. : $Var[\hat{\epsilon}]$
Synth1	4.06	0.32	**3.46**	**0.09**
Synth2	6.8	1.08	**6.64**	**0.08**
Synth3	4.61	0.53	**4.29**	**0.01**

TABLE 2.4 – Espérances et variances des pourcentages d'erreurs des classifieurs de type analyse discriminante versus les classifieurs de type régression logistique

Étonnamment, et quand bien même les données suivent dans chaque classe une loi normale multivariée et respectent à tour de rôle les hypothèses d'un classifieur d'analyse discriminante donné, on a quand même des performances *légèrement* supérieures pour la régression logistique (simple ou quadratique). Mais cette différence est tout de même trop faible pour être significative (0.35 d'écart en moyenne).
La variance du pourcentage d'erreur est par contre significativement plus faible pour la régression logistique. Cela peut s'expliquer par le fait que cette dernière ***ne fait pas de supposition*** sur la distribution des données, contrairement à l'analyse discriminante.

Comparaison du pourcentage d'erreur $\hat{\epsilon}$ entre classifieurs à frontière de décision linéaire ou quadratique

Fichier source	Front. Lin : $\mathbb{E}$ [$\hat{\epsilon}$]	Front. Lin : Var [$\hat{\epsilon}$]	Front. Quad : $\mathbb{E}$ [$\hat{\epsilon}$]	Front. Quad. : Var [$\hat{\epsilon}$]
Synth1	4.09	0.36	**3.63**	**0.27**
Synth2	7.42	0.66	**6.28**	**0.02**
Synth3	**4.18**	**0**	4.68	0.45

TABLE 2.5 – Espérances et variances des pourcentage d'erreurs des classifieurs donnant une frontière de décision linéaire versus les classifieurs donnant une frontière de décision quadratique

Cette comparaison renforce les hypothèses émises précédemment lors de l'analyse des paramètres estimés des classes (voir sous-section 2.1.2) :

- Les modèles quadratiques proposant des frontières de décision quadratiques sont plus performants sur les données *Synth1* et *Synth2* ;
- Les modèles linéaires proposant des frontières de décision linéaires sont plus performants sur les données *Synth3*.

On remarquera aussi en observant les variances que les modèles les plus adéquats sont aussi les plus fiables, c'est à dire les plus stables dans leurs performances : leurs variances sont systématiquement plus faibles que celles des modèles non adéquats.

2.2 Test sur données réelles : « Pima »

2.2.1 Visualisation graphique des données avec ACP

2.2.2 Visualisation « mathématique » des données via l'estimation des paramètres

2.2.3 Calculs des pourcentages d'erreur des classifieurs

Fichier source	ADQ	ADL	NBA	Log	Log Quad	Tree
Pima	23.97	22.62	23.8	21.77	24.72	26.23

TABLE 2.6 – Estimation du pourcentage d'erreur $\hat{\epsilon}$ sur $N = 100$ itérations des différents classifieurs sur les données « Pima »

2.3 Test sur données réelles : « Breast Cancer Wisconsin »

2.3.1 Visualisation graphique des données avec ACP

2.3.2 Visualisation « mathématique » des données via l'estimation des paramètres

2.3.3 Calculs des pourcentages d'erreur des classifieurs

Fichier source	ADQ	ADL	NBA	Log	Tree
Brestcancer	4.93	4.46	3.99	3.94	5.39

TABLE 2.7 – Estimation du pourcentage d'erreur $\hat{\epsilon}$ sur $N = 100$ itérations des différents classifieurs sur les données « breast cancer Wisconsin »

3. Challenge : données « Spam »

3.1 Analyse rapide des données

- Données regroupant un grand nombre d'indicateurs calculés sur des messages électroniques classés en deux classes : spams et non spams ;
- 57 dimensions numériques (variables `V1` à `V57`) pour 4600 observations (il faudra penser à ne pas prendre en compte la première colonne qui représente les numéro des lignes) ;
- Beaucoup d'observations égales à 0 pour chaque variable ;
- Disparité des distributions des valeurs :
  - Variances comprises entre 0 et 4 pour les variables `V1` à `V54` (sauf `V27` dont la variance vaut 11.33)
  - Variances élevées voire extrêmement élevées pour les 3 dernières variables avec 1 006.76 pour `V55`, 37 982.62 pour `V56` et 367 657.71 pour `V57` ;
  - Réduire (et centrer) les données sera donc fortement conseillé en cas d'ACP.

3.2 Essai des classifieurs sans traitement des données

Nous allons commencer par essayer de discriminer les données *Spam* directement avec nos classifieurs afin de voir ce qu'il se passe.

Analyse discriminante quadratique

- Erreurs lorsqu'on calcule les densités des données de chaque classe grâce à la fonction `mvdnorm(X, mu, Sigma)`, plus particulièrement lors de l'appel à `chol(Sigma)` qui calcule la factorisation de Cholesky : certaines composantes (i.e. variables) de la matrice Σ ne sont pas définis positives ;
  - Log d'erreur de R :
    `Error in chol.default(Sigma) : le mineur dominant d'ordre 41 n'est pas défini positif` ;
  - Cette erreur e répète pour les « mineurs dominants » 31, 32 et 41, soit pour les variables `V31`, `V32` et `V41` ;
  - Une analyse de ces variables ne nous a pas permis de comprendre pourquoi elles sont à l'origine de cette erreur.
- La solution mise en place pour appliquer l'analyse discriminante quadratique sur les données brutes fut de ne pas considérer ces trois variables ;
- On obtient alors un pourcentage d'erreur $\hat{\epsilon}$ égal à 16.93 pour $N = 20$ itérations.

Analyse discriminante linéaire

- Erreurs lorsqu'on calcule les densités des données de chaque classe grâce à la fonction `mvdnorm(X, mu, Sigma)` :
  - Les densités des deux classes sont toutes deux égales à 0 ;
  - La formule alors utilisée pour calculer $\mathbb{P}(\omega_k|x) = \frac{f_k(x)\pi_k}{f(x)} = \frac{f_k(x)\pi_k}{f_1(x)\pi_1+f_2(x)\pi_2}$ revient donc à diviser par 0, ce qui donne une valeur `NaN` ;
  - Ces observations sont en fait situées tellement loin du centre de la courbe gaussienne que leur densité est infinitésimale ;

  - Des tests effectués en supprimant ces observations extrêmes ne sont pas concluants et mènent à d'autres erreurs.
- La solution mise en place pour appliquer l'analyse discriminante linéaire sur les données brutes fut d'attribuer au hasard une classe aux observations posant ce problème :
  - Elles sont en effet très peu nombreuse, de l'ordre de moins de 0.5%.
- On obtient alors un pourcentage d'erreur $\hat{\epsilon}$ égal à 11.19 pour $N = 20$ itérations (avec 0.38% des données ayant $f_1(x) = f_2(x) = 0$).

Classifieur bayésien naïf

- Erreurs identique au modèle d'analyse discriminante quadratique ;
- Même chose que pour l'analyse discriminante quadratique, on ne considère pas les variables `V31`, `V32` et `V41` ;
- On obtient alors un pourcentage d'erreur $\hat{\epsilon}$ égal à 17.97 pour $N = 20$ itérations.
- Remarque :
  - On notera la corrélation de l'erreur entre le classifieur bayésien naïf et l'analyse discriminante quadratique : tous deux supposent que les classes ont leurs propres matrices de covariances (même si le classifieur bayésien naïf suppose en plus que les covariances sont nulles) ;
  - Par contre, l'analyse discriminante linéaire, qui ne présente pas cette erreur, suppose que les matrices de covariances sont identiques et l'estime en en faisant la moyenne ;
  - Bien qu'incapables de l'expliquer, nous pensons que ce pourrait être une piste à explorer afin d'expliquer ce type d'erreur.

Régression logistique

- Erreur aléatoire lors de l'inversion de la matrice hessienne :
  - Cette erreur n'est pas systématique mais apparait de temps en temps ;
  - La matrice hessienne n'est parfois pas inversible, on en déduit que son déterminant vaut alors 0 ;
  - On a donc la dérivée seconde de $\ln L(\beta_{(q)})$ à l'itération q qui vaut 0, ce qui veut dire que la courbe est localement rectiligne ;
  - On est situé tellement loin de l'optimum de vraisemblance que l'algorithme de Newton utilisé n'est alors pas capable de déterminer une direction à prendre (absence de courbure) afin de se rapprocher de l'optimum.
- Une suite de solutions a été trouvée :
  - Centrer et réduire les données afin d'éviter les valeurs propres trop grandes ou trop petites dans la matrice hessienne, cause supposée de sa non inversibilité ;
  - Cela marche effectivement, par contre, il y a de nouvelles erreurs du fait que la matrice hessienne est alors parfois singulière ;
  - Nous avons pris le parti de ne plus utiliser la méthode `solve` mais `ginv` (qui permet l'inversion de matrices singulières) afin d'obtenir l'inverse de la matrice hessienne.
- On obtient alors un pourcentage d'erreur $\hat{\epsilon}$ égal à 7.25 pour $N = 20$ itérations.

Régression logistique quadratique

Utiliser la régression logistique quadratique impliquerait d'estimer $p(p+3)/2$ paramètres, avec $p = 57$, soit 1 710 paramètres, ce qui est beaucoup trop important. Non seulement pour le temps de calcul qui serait nécessaire, mais aussi et surtout compte tenu du faible nombre d'observations au regard des 1 710 paramètres.
On peut aussi supposer, en remarquant la différence de performance entre l'analyse discriminante linéaire et quadratique (même si on a enlevé 3 dimensions pour cette dernière), que la régression logistique simple sera de toute façon plus efficace que la régression logistique quadratique.
Nous n'utiliserons donc pas la régression logistique quadratique sur les données brutes *Spam*.

Arbre de décision

- Seul modèle qui ne provoque pas d'erreurs sur le jeu de données *Spam* :
  - Ce modèle va effectuer des choix successif en ne considérant qu'une seule variable à la fois.
- On obtient un pourcentage d'erreur $\hat{\epsilon}$ égal à 8.61 pour $N = 20$ itérations.

Voici des exemples d'arbres de décision obtenus : ils peuvent être « relativement » simples comme très profonds et complexes.

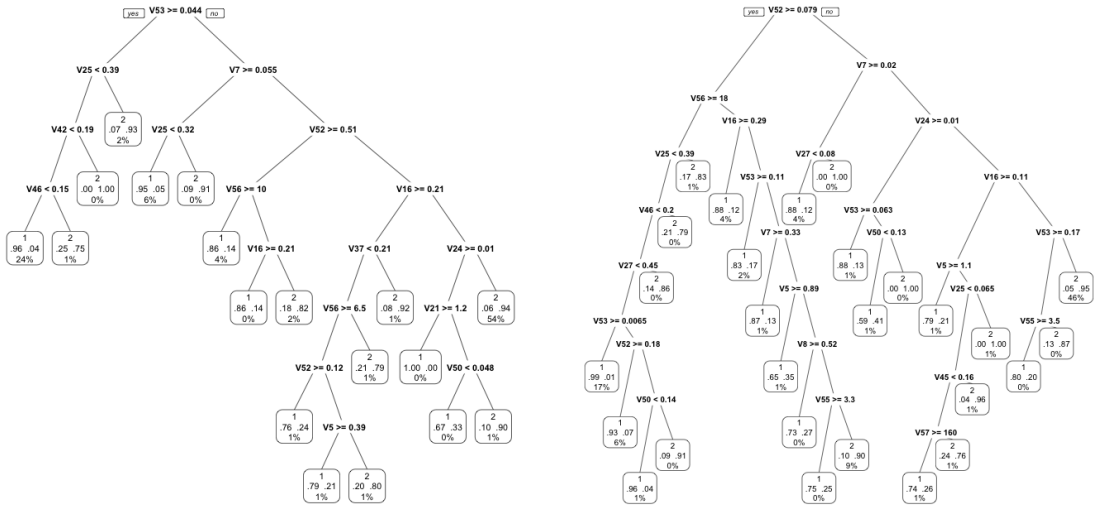


FIGURE 3.1 – Exemples d'arbres de décisions obtenus sur les données *Spam*

Dans chaque feuille, les informations données correspondent à (de haut en bas) :

- Le nom de la classe (1 ou 2) dans laquelle la feuille va placer les données ;
- Les proportions d'individus pour chaque classe :
  - Classe 1 à gauche, et classe 2 à droite ;
  - Pour une feuille donnée, le taux d'erreur va donc être la proportion d'individus n'appartenant pas à la classe de la feuille.
- Le pourcentage d'individus classés par la feuille :
  - Sur le premier arbre, on remarque par exemple que la feuille la plus à gauche classe 24% des individus dans la classe 1 (avec un taux d'erreur de 0.04) et que la feuille la plus à droite classe 54% des individus dans la classe 2 (avec un taux d'erreur de 0.06) ;
  - A noter que lorsqu'une feuille affiche 0%, c'est que la proportion d'individus classés est comprise entre 0 et 0.49 %, l'arrondi à l'entier inférieur donnant alors 0%.

Remarque à propos du classifieur des K plus proches voisins

Étant en dimension 57, la distance entre les individus est immensément plus grande qu'en dimension 2 ou 3. Pour un individu, la probabilité qu'un de ses voisins soit de la même classe que lui est plus faible. Le pourcentage d'erreur obtenu sur $N = 20$ itérations parle d'ailleurs de lui-même : 21.62%.

Récapitulatif des pourcentages d'erreur

Fichier source	ADQ	ADL	NBA	Rég. Log.	Tree
Spam	16.93	**11.19**	17.97	**7.25**	**8.61**

TABLE 3.1 – Estimation du pourcentage d'erreur $\hat{\epsilon}$ sur $N = 20$ itérations des différents classifieurs sur les données « Spam »

On remarque, de façon encore plus flagrante que lors de l'analyse des données synthétiques, que les classifieurs ne faisant pas de supposition sur la distribution des données sont meilleurs que les autres. La régression logistique et les arbres de décisions présentent effectivement de bien meilleures performances que les classifieurs d'analyse discriminante.
On note aussi qu'une frontière de décision linéaire est plus indiquée qu'une frontière de décision quadratique.

3.3 Classification sur les données résultantes d'une ACP

On effectue une ACP (centrée et réduite) sur les données *Spam* afin de diminuer le nombre de dimensions. Par contre, rien ne nous garantit que la dispersion des valeurs autour des axes des variables soit le facteur discriminant des classes *spam* ou *non spam*. On va donc vérifier que les informations de classe ne se soient pas perdus durant l'ACP en représentant graphiquement ces informations sur les deux premières composantes principales.

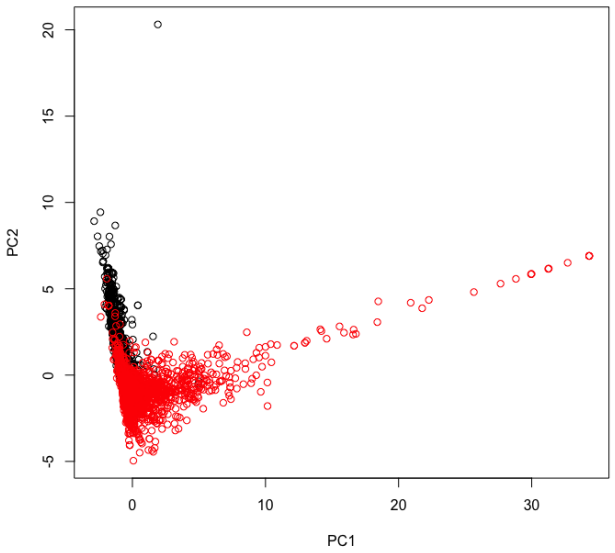


FIGURE 3.2 – Représentation des résultats de l'ACP (effectué sur les données *Spam*) sur les deux premières composantes principales en fonction des classes

On voit que même s'il y a une zone importante de mélange, l'ACP ne supprime pas les informations de classe.

On va donc pouvoir l'utiliser pour effectuer nos discriminations et attester ou non de son utilité :

- L'ACP, en normalisant les données, permet-elle de supprimer les erreurs obtenues lors des discriminations effectuées sur les données brutes ?
- L'ACP est-elle concluante, c'est à dire arrive-t-on à obtenir les mêmes performances que précédemment (voir section 3.2) avec moins de dimensions ?

L'ACP, en normalisant les données, permet-elle de supprimer les erreurs obtenues lors des discriminations effectuées sur les données brutes ? La réponse cette question est oui. Effectuer les discriminations sur les données résultantes de l'ACP ne génère plus d'erreur sauf pour l'analyse discriminante linéaire où le problème des densités égales à 0 pour les deux classes persiste.
Nous pensons que cette absence d'erreur est due à la normalisation sur les données (qui ont été centrée et réduites) *ainsi* qu'au changement de base. En effet, nous avons testé les classifieurs sur les données brutes seulement normalisées via la fonction `scale` (centrées et réduites) sans succès : les erreurs étaient toujours présentes.

L'ACP est-elle concluante, c'est à dire arrive-t-on à obtenir les mêmes performances que précédemment avec moins de dimensions ? Nous avons commencé par analyser l'inertie expliquée cumulée afin de voir si, en termes d'inertie, quelques dimensions étaient suffisantes ou bien s'il fallait en considérer beaucoup pour ne pas perdre d'information :

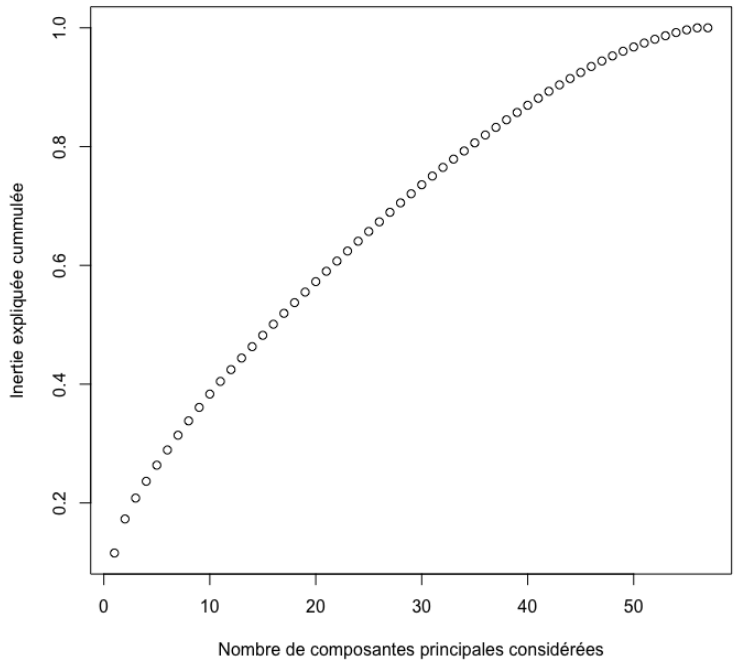


FIGURE 3.3 – Inertie cumulée de l'ACP (effectué sur les données *Spam*)

On ne voit pas vraiment de démarcation apparaître nous indiquant quel pourrait être le nombre optimal de composantes principales à considérer.
Nous allons donc estimer cet hyper-paramètre en appliquant successivement les classifieurs sur les deux premières composantes principales, puis sur les trois premières, puis sur les quatre premières, etc. jusqu'à prendre en compte l'ensemble des composantes principales.

Voici le graphique obtenus sur les deux classifieurs qui donnent les meilleurs résultats, à savoir la régression logistique et les arbres de décision :

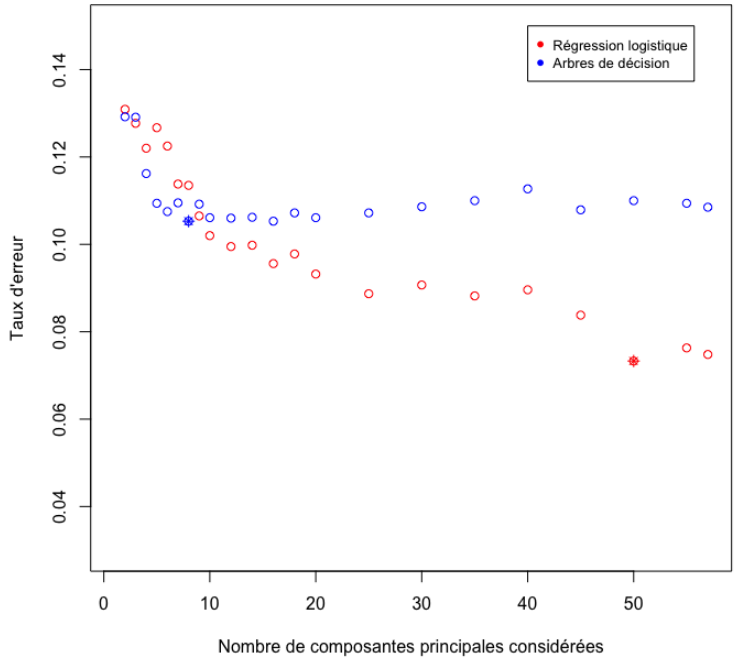


FIGURE 3.4 – Taux d'erreurs en fonction du nombre de composantes principales prises en considération

Ainsi, il faudrait seulement les 9 premières composantes principales pour le modèle des arbres de décision et les 50 premières pour la régression logistique afin d'avoir un pourcentage d'erreur optimal :

- 10.53% d'erreur pour les arbres de décision ;
- 7.33% d'erreur pour la régression logistique.

Comparaison des résultats des classifications sur les données originales et sur les données après ACP

Classifieur	Données originales	Données après ACP
Régression logistique	**7.25**	7.33
Arbres de décision	**8.61**	10.53

TABLE 3.2 – Comparaison des pourcentages d'erreur entre les classifications effectuées sur les données originales et celles effectuées sur les données obtenues après ACP

Conclusion, l'ACP n'est pas réellement efficace ici comme outil de réduction de dimensions :

- On réduit drastiquement le nombre de dimensions pour le classifieur des arbres de décision (de 57 à 9), par contre, on passe de 8.61% à 10.53% d'erreur ;
- On réduit très peu le nombre de dimensions pour le modèle de régression logistique (de 57 à 50) et on passe de 7.25% à 7.33% d'erreur (très légère augmentation).

Remarque sur la réduction du nombre de dimensions

L'ACP est une méthode naïve de réduction de variables à visée de discrimination de classe. C'est à dire qu'elle ne garantie pas de sélectionner les axes discriminants. Si les axes discriminants sont aussi ceux présentant la plus grande dispersion, alors effectivement la réduction du nombre de dimensions sera pertinente, dans le sens ou le maximum d'inertie expliquée signifiera aussi la meilleure discrimination de classe possible. Dans le cas

contraire, on perdra l'information de classe au profit de l'inertie expliquée, ce qui ne nous servira à rien dans le cadre d'une classification supervisée.

Des méthodes de sélection de variables, afin de déterminer les variables les plus discriminantes, seraient plus indiquée afin de réduire le nombre de dimensions.

3.4 Classification à l'aide d'une forêt d'arbre

Le fait que la classification grâce aux arbres de décision soit la seconde meilleure après la régression logistique, et qu'elle marche sur les données brutes sans traitement préalable nous a poussé à considérer le classifieur de forêt d'arbres aléatoires.

Cela a été très simple à mettre en place grâce au package R `randomForest`. Le code de la fonction d'apprentissage et d'évaluation peut être trouvée en annexe section C.2.

Performances obtenues

Nombre d'arbres aléatoires	Pourcentage d'erreur
2 000	4.04
4 000	1.44
10 000	1.44

TABLE 3.3 – Pourcentage d'erreurs obtenus lors de classifications des données *Spam* avec le classifieur des forêts aléatoires pour $N = 20$ itérations

Avec 1.44% d'erreur, on a bien là un classifieur extrêmement efficace.

Importances des variables dans la construction de la forêt

Le package `randomForest` permet d'avoir un aperçu de l'importance des variables :

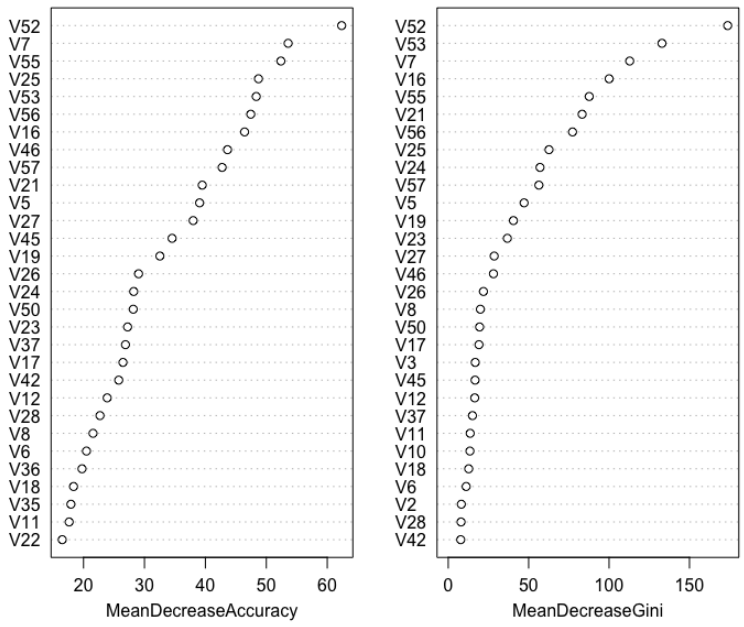


FIGURE 3.5 – Importance des variables dans la construction de la forêt aléatoire

A. Fonctions pour l'analyse discriminante

A.1 Fonctions d'apprentissage `adq.app`, `adl.app` et `nba.app`

Fonctions utilitaires de calcul des proportions, moyennes et matrices de co-variance

```
# calcul des proportions
prop.app = function(Xapp, zapp) {
    zapp = factor(zapp)
    g = length(levels(zapp)) # nombre de classes
    n = dim(Xapp)[1] # n = nombre d'individus total
    p = dim(Xapp)[2] # p = nombre de variables
    prop = list()
    for (k in 1:g) {
        prop[[k]] = table(zapp)[k] / length(zapp)
    }
    return(prop)
}


# calcul des centres de gravité
mu.app = function(Xapp, zapp) {
    zapp = factor(zapp)
    g = length(levels(zapp)) # nombre de classes
    n = dim(Xapp)[1] # n = nombre d'individus total
    p = dim(Xapp)[2] # p = nombre de variables
    mu = list()
    for (k in 1:g) {
        dataClassK = Xapp[zapp == levels(zapp)[k],]
        mu[[k]] = apply(dataClassK, MARGIN = 2, mean)
    }
    return(mu)
}


# calcul des matrices de covariance
sigma.app = function(Xapp, zapp) {
    zapp = factor(zapp)
    g = length(levels(zapp))
    n = dim(Xapp)[1]
    p = dim(Xapp)[2]
    sigma = list()
    for (k in 1:g) {
        dataClassK = Xapp[zapp == levels(zapp)[k],]
        sigma[[k]] = var(dataClassK)
    }
    return(sigma)
}
```

Fonctions d'apprentissage des paramètres

```r
adq.app <- function(Xapp, zapp) {
    zapp = factor(zapp)
    params = list()
    params[["pi"]] = prop.app(Xapp, zapp)
    params[["mu"]] = mu.app(Xapp, zapp)
    params[["sigma"]] = sigma.app(Xapp, zapp)
    return(params)
}
```

```r
adl.app <- function(Xapp, zapp) {
    zapp = factor(zapp)
    g = length(levels(zapp))
    p = dim(Xapp)[2]
    params = list()
    prop = prop.app(Xapp, zapp)
    mu = mu.app(Xapp, zapp)
    classes_sigma = sigma.app(Xapp, zapp)
    sigma = matrix(0, p, p)
    for (k in 1:g) {
        sigma = sigma + (prop[[k]] * classes_sigma[[k]])
    }
    for (k in 1:g)
        params[["sigma"]][[k]] = sigma
    params[["pi"]] = prop
    params[["mu"]] = mu
    return(params)
}
```

```r
nba.app <- function(Xapp, zapp) {
    zapp = factor(zapp)
    g = length(levels(zapp))
    params = list()
    prop = prop.app(Xapp, zapp)
    mu = mu.app(Xapp, zapp)
    classes_sigma = sigma.app(Xapp, zapp)
    for (k in 1:g)
        classes_sigma[[k]] = diag(diag(classes_sigma[[k]]))
    params[["pi"]] = prop
    params[["mu"]] = mu
    params[["sigma"]] = classes_sigma
    return(params)
}
```

A.2 Fonction de discrimination `ad.val`

```
ad.val <- function(params, Xtst) {
    n = nrow(Xtst)

    f1 = mvdnorm(Xtst, params$mu[[1]], params$sigma[[1]])
    f2 = mvdnorm(Xtst, params$mu[[2]], params$sigma[[2]])

    discrimination = list()
    discrimination[["pw1"]] = vector(length = n)
    discrimination[["pw2"]] = vector(length = n)
    discrimination[["pred"]] = vector(length = n)

    for (i in 1 : n) {
        discrimination[["pw1"]][i] = f1[i]*params$pi[[1]]/(f1[i]*params$pi[[1]]+
f2[i]*params$pi[[2]])
        discrimination[["pw2"]][i] = f2[i]*params$pi[[2]]/(f1[i]*params$pi[[1]]+
f2[i]*params$pi[[2]])

        if(discrimination[["pw1"]][i] > discrimination[["pw2"]][i])
            discrimination[["pred"]][i] = 1
        else
            discrimination[["pred"]][i] = 2
    }
    discrimination[["pred"]] = factor(discrimination[["pred"]])
    return(discrimination)
}
```

B. Fonctions pour la régression logistique

B.1 Fonction de calcul des probabilités à postériori `postprob`

```
postprob <- function(beta, X) {
    X <- as.matrix(X)

    prob <- t(exp(t(beta)%*%t(X))/(1+exp(t(beta)%*%t(X))))
}
```

B.2 Fonction d'apprentissage `log.app`

```
log.app <- function(Xapp, zapp, intr, epsi) {
    n <- dim(Xapp)[1]
    p <- dim(Xapp)[2]

    Xapp <- as.matrix(Xapp)

    if (intr == T) { # on ajoute une ordonnée à l'origine
        Xapp <- cbind(rep(1,n),Xapp)
        p <- p + 1
    }

    targ <- matrix(as.numeric(zapp),nrow=n) # ti: la réalisation d'une variable Ti~B(pi)
    targ[which(targ==2),] <- 0              # remplacer la classe 2 par 0
    Xapp_transposed <- t(Xapp)

    beta <- matrix(0,nrow=p,ncol=1)

    conv <- F
    iter <- 0
    while (conv == F) {
        iter <- iter + 1
        beta_old <- beta

        prob_w1 <- postprob(beta, Xapp) # P(w1|x)
        prob_w2 <- 1 - prob_w1
        MatW <- diag(as.numeric(prob_w1 * prob_w2)) # W: Wii = pi(1-pi)

        mat_hessienne <- -Xapp_transposed %*% MatW %*% Xapp
        mat_hessienne_inverse <- solve(mat_hessienne)
        gradient_w1 <- Xapp_transposed %*% (targ - prob_w1)
        beta <- beta_old - (mat_hessienne_inverse %*% gradient_w1)

        if (norm(beta - beta_old) < epsi) {
            conv <- T
        }
    }

    prob_w1 <- postprob(beta, Xapp) # P(w1|x)
    prob_w2 <- 1 - prob_w1
    out <- NULL
    out$beta <- beta
    out$iter <- iter
    out$logL <- sum(targ*prob_w1+(1-targ)*(prob_w2))
    out
}
```

B.3 Fonction de discrimination `log.val`

```
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_w1 <- postprob(beta, Xtst) # P(w1|x)
    prob_w2 <- 1 - prob_w1
    prob <- cbind(prob_w1, prob_w2)
    pred <- max.col(prob)

    out <- NULL
    out$prob <- prob
    out$pred <- pred
    return(out)
}
```

B.4 Fonction pour la régression logistique quadratique

Fonction qui transforme l'espace de X en un espace quadratique :

```
log_quad <- 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)

    return(X2)
}
```

C. Fonctions pour les arbres de décision et les forêt aléatoires

C.1 Apprentissage et prédiction pour les arbres de décision

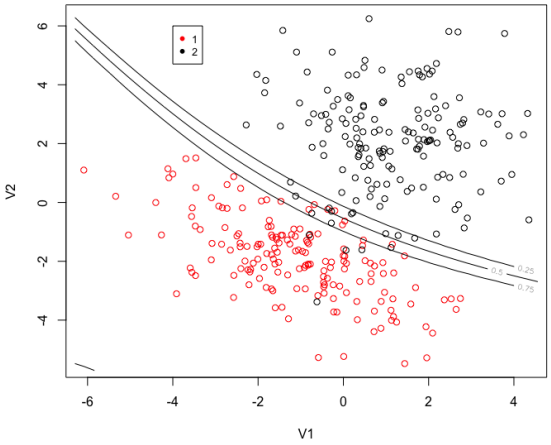
```
tree.app <- function(Xapp, zapp, Xtst) {
    zapp = factor(zapp)
    ctrl = rpart.control(cp=0.0001)
    tree = rpart(zapp~., data = Xapp, control = ctrl)
    treeOptimal <- prune(tree,cp=tree$cptable[which.min(tree$cptable[,4]),1])
    pred = predict(treeOptimal, Xtst, type = "class")

    # for tree.partition afin de voir les frontières de décision
    tree_package_tree = tree(zapp~., Xapp, control = tree.control(nobs=dim(Xapp)[1],mindev=0.0001))

    out <- NULL
    out$pred <- factor(pred)
    out$fullTree = tree
    out$optimalTree = treeOptimal
    out$treeForDecisionBorder = tree_package_tree
    return(out)
}
```

C.2 Apprentissage et prédiction pour les forêts aléatoires

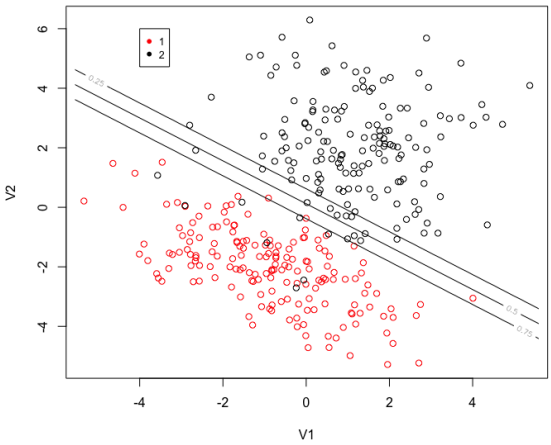
```
random.forest.app <- function(Xapp, zapp, Xtst, nbTrees = 1000) {
    out <- NULL
    random_forest <- randomForest(as.factor(sample$zapp) ~ .,
                        data=sample$Xapp,
                        importance=TRUE,
                        ntree=nbTrees)
    out$random_forest = random_forest
    out$pred <- predict(random_forest, sample$Xtst)
    return(out)
}
```

D. Frontières de décision

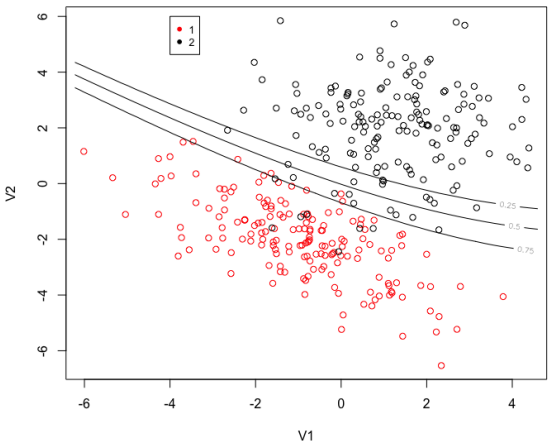
D.1 Données *Synth1*



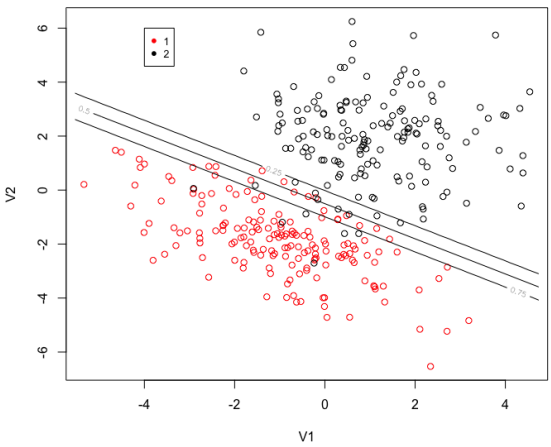
(a) A.D. Quadratique



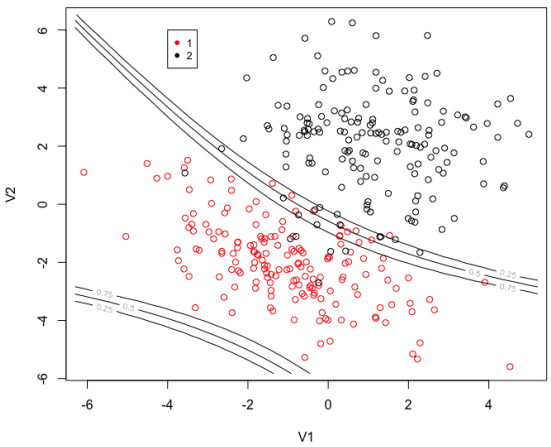
(b) A.D. Linaire



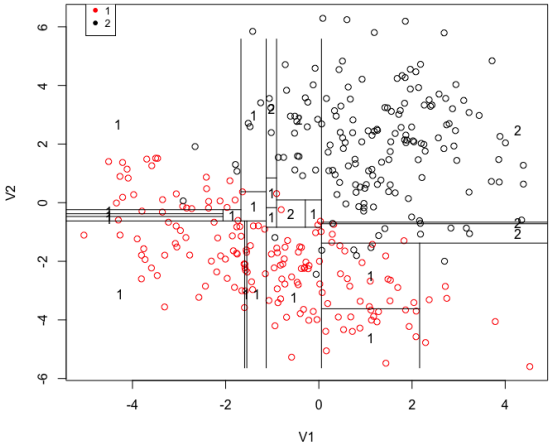
(c) Bayésien naïf



(d) Régression logistique



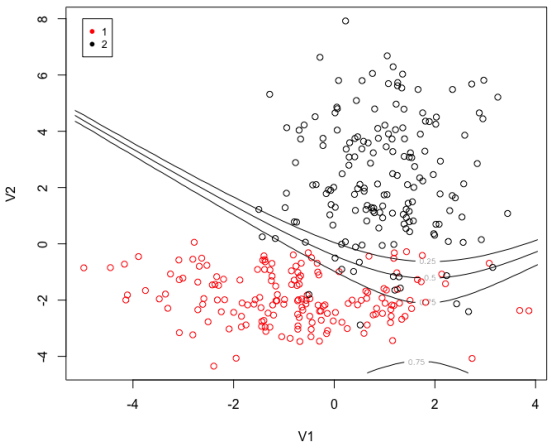
(e) Régression logistique quadratique



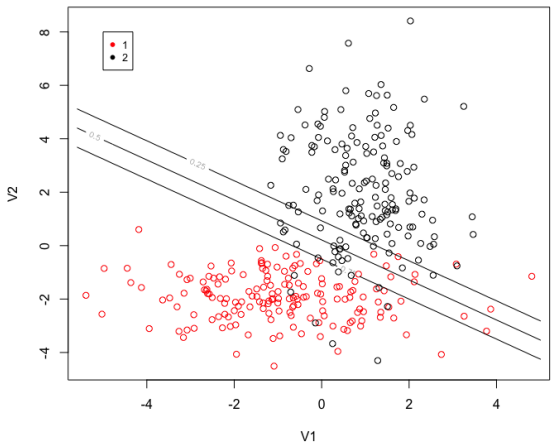
(f) Arbre de décision

FIGURE D.1 – Frontières de décision des classifieurs pour les données *Synth1* au niveaux $\mathbb{P}(\omega_1|x) = \{0.25, 0.5, 0.75\}$

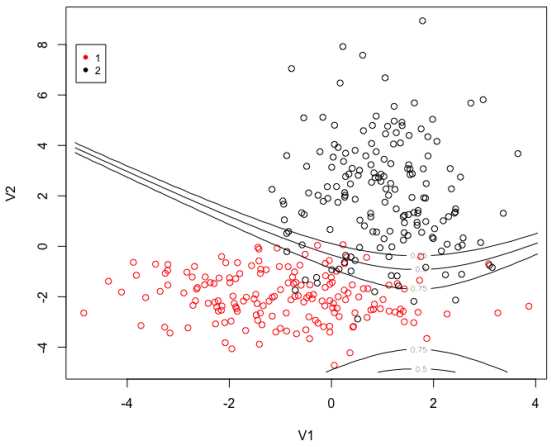
D.2 Données *Synth2*



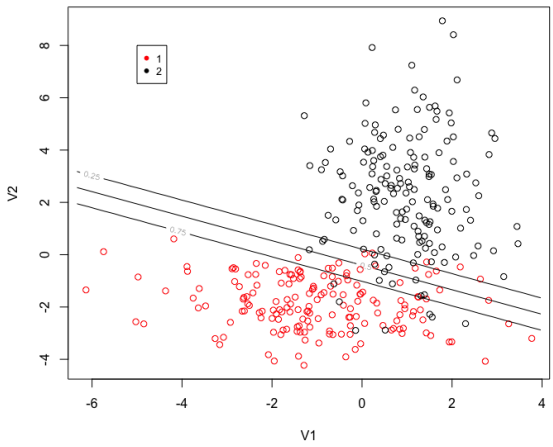
(a) A.D. Quadratique



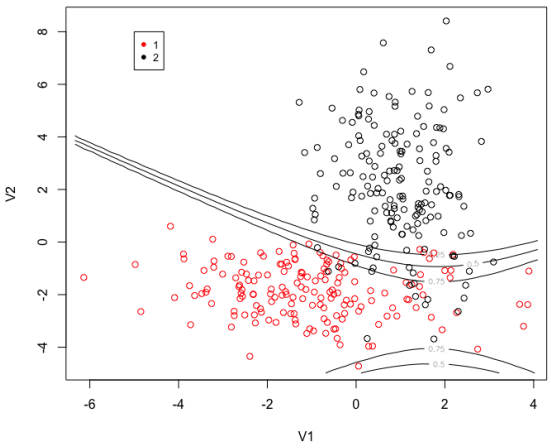
(b) A.D. Linaire



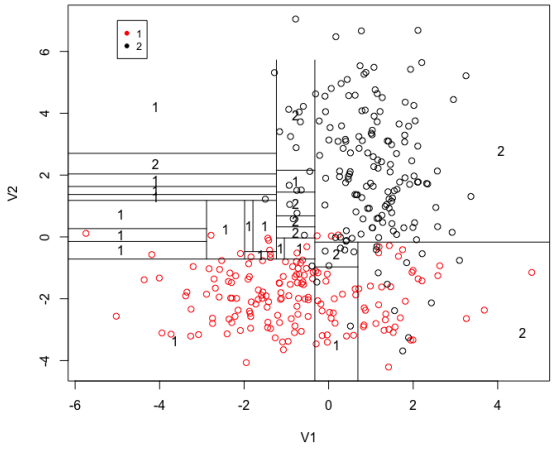
(c) Bayésien naïf



(d) Régression logistique



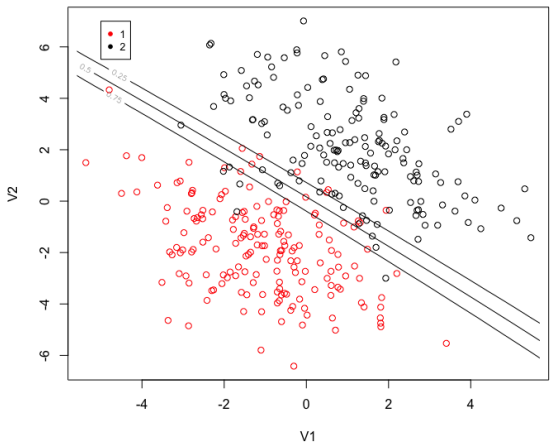
(e) Régression logistique quadratique



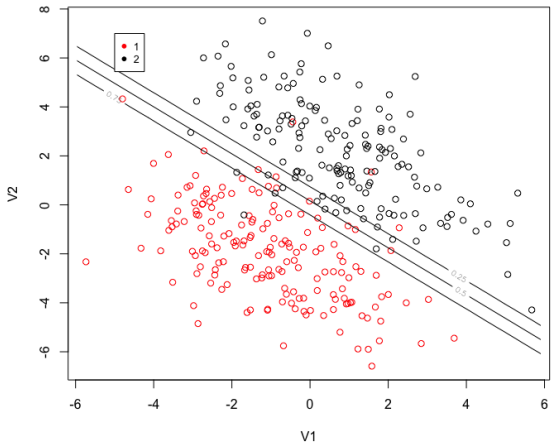
(f) Arbre de décision

FIGURE D.2 – Frontières de décision des classifieurs pour les données *Synth2* au niveaux $\mathbb{P}(\omega_1|x) = \{0.25, 0.5, 0.75\}$

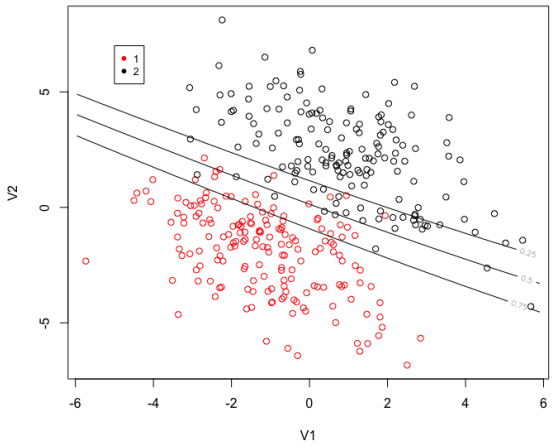
D.3 Données *Synth3*



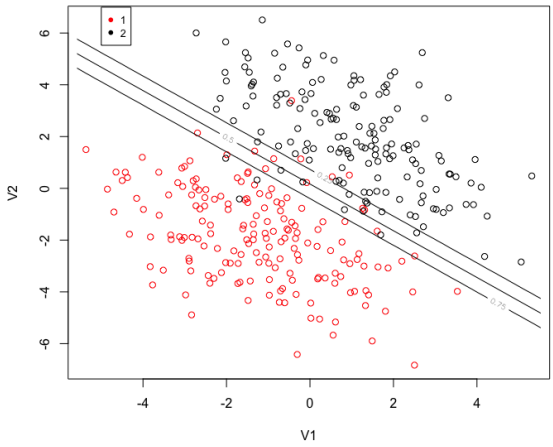
(a) A.D. Quadratique



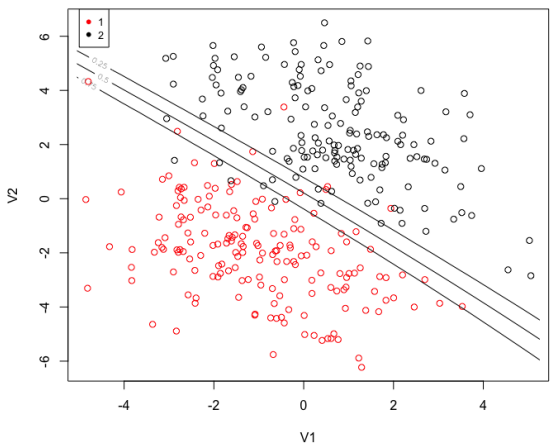
(b) A.D. Linaire



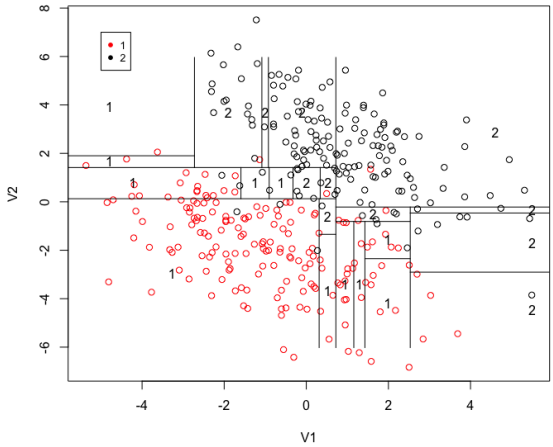
(c) Bayésien naïf



(d) Régression logistique



(e) Régression logistique quadratique



(f) Arbre de décision

FIGURE D.3 – Frontières de décision des classifieurs pour les données *Synth3* au niveaux $\mathbb{P}(\omega_1|x) = \{0.25, 0.5, 0.75\}$