TP04 SY09 - Discrimination

Laura BROS Alexis DUROCHER

Juin 2017

Table des matières

0.1 Introduction

L'objectif de ce TP est de découvrir de nouvelles méthodes supervisées afin de pouvoir juger de leur efficacité selon le contexte. Pour ce faire, nous allons dans un premier temps apprendre à utiliser différentes méthodes d'Analyse discriminante, puis dans un second temps nous aborderons les méthodes de régression logistique. Pour finir, nous allons apprendre à utiliser les arbres binaires de décision. L'étude de ces différents modèles de classifieurs portera sur différents jeux de données, l'interprétation des résultats dépendra donc des données de base, des hypothèses assumées par chacun des classifieurs, ainsi que du nombre de paramètres estimés lors de leur application. Afin de pouvoir comparer les différentes méthodes, nous allons commencer par les programmer en utilisant le langage R. Nous travaillerons dans un premier temps sur des données simulées, puis nous appliquerons les méthodes sur des jeux de données réelles.

0.2 Programmation

0.2.1 Analyse Discriminante

L'analyse discriminante se base sur l'hypothèse que le vecteur aléatoire de caractéristique X étudié suit, conditionnellement à chaque classe, une loi normale multidimensionnelle d'espérance μ_k et de variance Σ_k.

Nous avons dans un premier temps programmé sous R trois modèles d'analyse discriminante dans le cas binaire (jeux de données comptant g=2 classes) :
— L'Analyse Discriminante Quadratique (ADQ)
— L'Analyse Discriminante Linéaire (ADL)
— Le Classifieur Bayesien Naïf (NBA)
Les fonctions seront détailles en annexe A.

Analyse Discriminante Quadratique C'est le cas général où μ_k et de variance Σ_k sont différents pour chaque classe. Les paramètres sont inconnus mais on utilise l'ensemble d'apprentissage ainsi que les estimateurs du maximum de vraisemblance qui nous donnent :

$$\begin{cases} \pi_k = \frac{n_k}{n}, \\ \mu_k = \frac{1}{n_k} \sum_{i=1}^{n} z_{ik} x_i, \\ \Sigma_k = V_k \end{cases}$$

Nous avons utilisé les variables suivantes :
— $z_i k$ indique l'appartenance à la classe ω_k
— $n_k = \sum_{i=1}^{n} z_{ik}$
La fonction correponsante (adq.R) prend en entrées la matrice d'apprentissage ainsi que ses étiquettes et renvoie les paramètres μ,π et Σ. Ils sont respectivement stockés dans un vecteur de longueur k, une matrice de dimension (1 × p), et un tableau à 3 dimensions (p×p×k).

Analyse Discriminante Linéaire Dans ce cas, on suppose que la matrice de variance Σ_k est commune à toutes les classes. Les estimateurs du maximum de vraisemblance deviennent :

$$\begin{cases} \pi_k = \frac{n_k}{n}, \\ \mu_k = \frac{1}{n_k} \sum_{i=1}^{n} z_{ik} x_i, \\ \Sigma_k = \frac{1}{n} \sum_{k=1}^{g} n_k V_k, \end{cases}$$

La fonction correponsante (adl.R) prend également en entrées la matrice d'apprentissage ainsi que ses étiquettes et renvoie les paramètres μ,π et Σ stockés de la même manière que pour l'ADQ.

Classifieur naïf bayésien Ce modèle suppose l'indépendance des variables X_j conditionnellement à la classe Z ce qui revient à supposer les matrices de variances Σ_k diagonales dans le modèle gaussien. Les paramètres μ et π restent les mêmes mais les matrices de variances Σ_k sont estimées par la matrice diagonale :

$$\begin{cases} \pi_k = \frac{n_k}{n}, \\ \mu_k = \frac{1}{n_k} \sum_{i=1}^{n} z_{ik} x_i, \\ \Sigma_k = diag(s_k^2 1, ..., s_k^2 j, ... s_k^2 p), \end{cases}$$

La fonction correponsante (nba.R) prend les mêmes données d'entrée que les fonctions précédentes et renvoie de la même manière l'estimation des paramètres.

0.2.2 Régression Logistique

Pour ce modèle, le principe revient à estimer directement les probabilités d'appartenance aux classes. Cette méthode ne fait donc pas, contrairement à l'analyse discriminante, d'hypothèses sur la distribution des données. L'apprentissage se fait grâce à l'algorithme d'optimisation itératif de Newton-Raphson. Cette méthode consiste à sélectionner un vecteur de poids initial puis à calculer une séquence de vecteurs en appliquant itérativement la formule. En pratique, on arrête l'algorithme lorsqu'un critère d'arrêt est vérifié. La suite de vecteurs converge vers un maximum local de la log-vraisemblance. L'application de cette méthode nécessite le calcul des coefficients de la matrice hessienne.

Notre fonction log.app est visible en annexe. Lors de nos tests, nous avons pris soin de vérifier que la matrice Hessienne était bien définie négativement et que la log-vraisemblance augmente au fil des itérations (en effet, la fonction est concave). A partir de Newton-Raphson, nous avons développé une deuxième fonction dont les arguments d'entrée sont les données de test et l'estimateur du maximum de vraisemblance, beta, retourné par log.app. En prenant soin d'ajouter un intercept aux données de test en fonction de beta, la fonction retourne les probabilités à posteriori grâce à la formule exp(tx) et le classement associé. 1+exp(tx)

Régression logistique linéaire Nous avons effectué la régression logistique linéaire en implémentant deux fonctions, l'une permettant de faire l'apprentissage du modèle (log.app), l'autre permettant d'appliquer le modèle obtenu sur un ensemble de données (log.val). La fonction log.app, permettant d'apprendre les paramètres du modèles, prendra comme arguments d'entrée le tableau de données Xapp, le vecteur zapp des étiquettes associées, ainsi qu'une variable binaire intr indiquant s'il faut ou non ajouter une ordonnée à l'origine (intercept*) à la matrice d'exemples et un scalaire epsi correspondant au seuil en-deçà duquel on considère que l'algorithme d'apprentissage a convergé. *L'intercept est utile car il permet à la frontière de ne pas nécessairement passer par l'origine. La méthode de régression linéaire présuppose que la frontière de décision est linéaire. En effet, P(1|x) = P(2|x) ssi ${}^t x = 0$. Les frontières de décision sont donc représentées par des hyperplans passant par l'origine, ce que l'intercept permet de changer.
La fonction log.app retourne la matrice beta correspondant à l'estimateur du maximum de vraisemblance des paramètres, de dimensions p 1 (ou (p + 1) 1 si une ordonnée à l'origine a été ajoutée), le nombre niter d'itérations effectuées par l'algorithme de Newton-Raphson, et la valeur logL de la vraisemblance à l'optimum.

La fonction log.val, permettant de classer un ensemble d'individus, prend comme arguments d'entrée le tableau de données Xtst à classer et la matrice beta des paramètres déterminée par la fonction log.app. Elle réalise un test sur les dimensions de la matrice beta, pour déterminer si une ordonnée à l'origine doit être ou non ajoutée à la matrice d'exemples Xtst à évaluer. Elle retourne une structure contenant la matrice prob des probabilités a posteriori estimées et le vecteur des classements associés.

Nous avons utilisé deux fonctions fournies :

- post.pr qui calcule les probabilités a posteriori à partir d'une matrice de paramètres beta et d'un tableau de données X,
- prob.log pour visualiser les courbes de niveau

Régression logistique quadratique Ce modèle nous a permis de généraliser le modèle de régression logistique. Nous avons transformé les données dans un espace plus complexe dans lequel les classes peuvent être séparées par un hyperplan.

Pour ce faire, nous avons commencé par calculer les produits des variables décrivant les individus d'apprentissage. On effectue ensuite l'apprentissage sur les p(p + 3)/2 nouvelles variables obtenues.

De même nous avons utilisé une fonction fournie, prob2.log qui permet de visualiser les frontières de décision obtenues.

0.2.3 Vérification des données

Les courbes présentées ci-dessous ont été obtenues en utilisant la totalité des données Synth1-40 pour l'apprentissage des données.

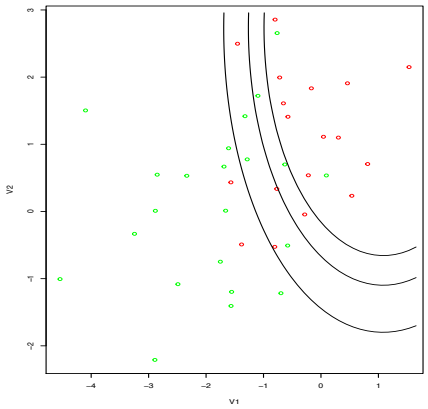


FIGURE 1 – Analyse discriminante quadratique

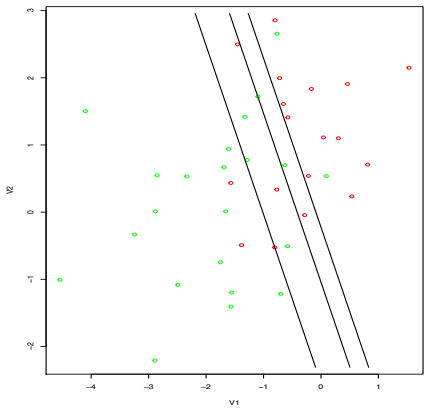


FIGURE 2 – Analyse discriminante linéaire

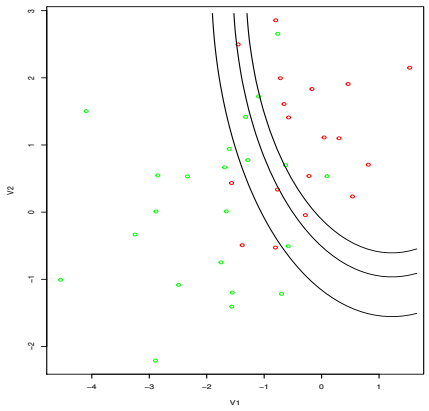


FIGURE 3 – Classifieur bayésien naïf

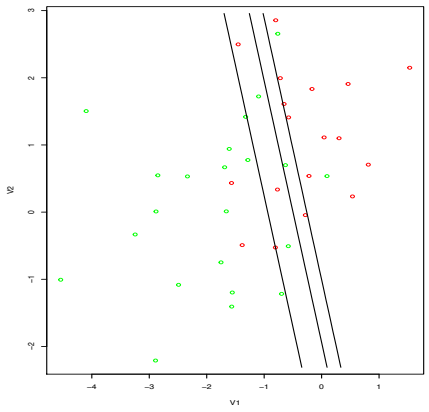


FIGURE 4 – Régression Logistique Linéaire

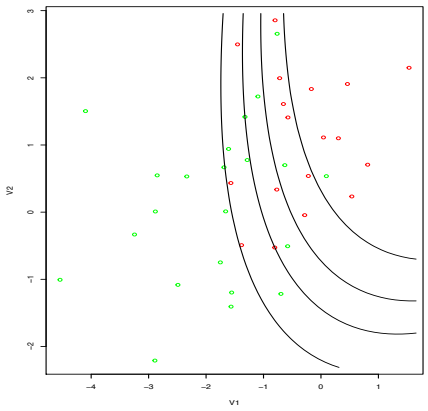


FIGURE 5 – Régression Logistique Quadratique

0.3 Application

Nous allons tester nos différents classifieurs sur les jeux de données mis à notre disposition. Dans les 2 contextes (données simulées ou données réelles), nous commencerons toujours par se faire une idée de la dispersion des données dans l'espace avant de commencer à calculer les performances de nos classifieurs. En effet, nous avons déjà remarqué dans les TPs précédents que les performances d'un classifieur étaient très largement dépendantes du jeux de données sur lequel ces derniers sont appliqués. Ici, nous allons essayer de voir l'efficacité de nos classifieurs selon les paramètres des jeux de données considérés. Nous ne ferons ici, qu'une analyse superficielle de la dispersion des deux classes (comparaison de leur matrice de covariance respective) afin d'estimer quelles hypothèses peuvent ou non être vérifiées pour chaque jeux de données.
Rappelons toutefois que la régression ne dépend pas des paramètres : les probabilités conditionnelles sont directement estimées par optimisation itérative. Nous pouvons d'ailleurs utiliser la régression ici car la variable z à expliquer est quantitative. Cela nous permettra de comparer l'efficacité des classifieurs paramétriques (ad) avec d'autres indépendants de toutes hypothèse à priori sur les données (reg).
Concernant les arbres de décisions, nous allons chercher à estimer le nombre optimal de feuilles pour obtenir les meilleurs résultats sur nos données. En effet, un arbre trop 'feuillus' perdra en robustesse mais gagnera en flexibilité (il s'attachera beaucoup au particularités locales), à l'inverse, un arbre trop élagué gagnera en robustesse mais perdra sa précision qui fait sa force. Nous utiliserons une fonction prédéfinie dans R qui permet d'estimer ce nombre optimal de feuilles.

0.3.1 Analyse des données simulées

Nous travaillons ici sur des données simulées avec des paramètres différents. Commençons par estimer ces paramètres.

Synth1-1000

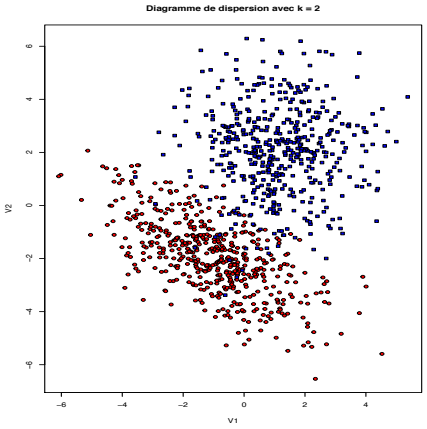


FIGURE 6 – Synth1-1000

$$\begin{cases} \Sigma_1 = \begin{pmatrix} 2.89 & -1.53 \\ -1.53 & 1.96 \end{pmatrix} \\ \Sigma_2 = \begin{pmatrix} 2.09 & -0.00 \\ -0.000 & 2.81 \end{pmatrix} \end{cases}$$

Sur la figure, on constate que la dispersion des données est relativement proche pour les deux classes. Toutefois, leur matrice de covariance diffère, la dispersion est donc différente. On constate aussi que les classes ne sont pas homogènes et distinctes. En effet les échantillons sont assez confondus.

Synth2-1000

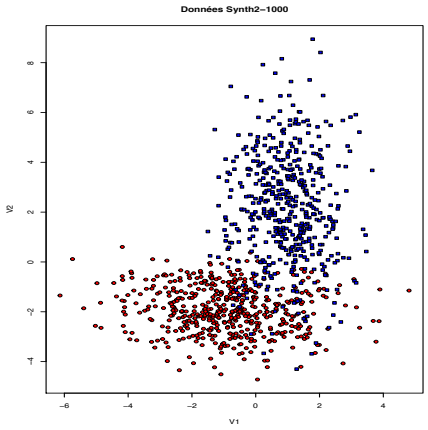


FIGURE 7 – Synth2-1000

$$\begin{cases} \Sigma_1 = \begin{pmatrix} 2.81 & -0.19 \\ -0.19 & 0.90 \end{pmatrix} \\ \Sigma_2 = \begin{pmatrix} 0.90 & -0.04 \\ -0.04 & 4.59 \end{pmatrix} \end{cases}$$

Sur la figure, on constate que la dispersion des données semble très différente pour les deux classes. Cela se confirme en comparant leur matrice de covariance qui sont nettement différentes. Aussi, graphiquement, nous constatons que les deux échantillons sont très confondus. Les valeurs sont "inversées" dans leur diagonale. Cela se traduit par des nuages presque "perpendiculaires" dans l'espace.

Synth3-1000

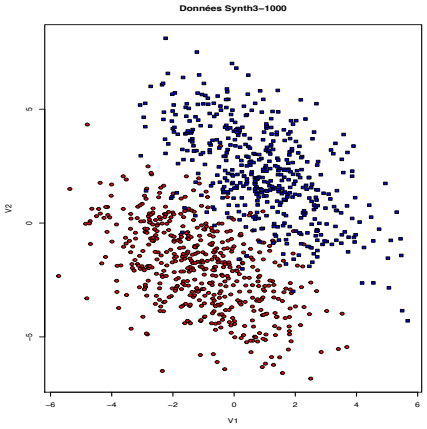


FIGURE 8 – Synth3-1000

$$\begin{cases} \Sigma_1 = \begin{pmatrix} 2.88 & -1.55 \\ -1.55 & 3.50 \end{pmatrix} \\ \Sigma_2 = \begin{pmatrix} 2.92 & -2.02 \\ -2.02 & 4.17 \end{pmatrix} \end{cases}$$

Sur la figure, on constate que la dispersion des données semble très similaire pour les deux classes. Cela se confirme en comparant leurs matrices de covariance qui sont très proches. On pourra même les considérer égales en arrondissant grossièrement nos résultats. Les deux nuages de points semblent donc séparés par une frontière linéaire.

0.3.2 Etude des performances sur données simulées

Pour chaque classifieur nous allons itérer 20 fois l'estimation ponctuelle de son taux d'erreur. Ponctuelle car nos échantillons d'apprentissage et de test ne sont pas regénérés (les individus se retrouvent une fois dans l'ensemble d'apprentissage ou une fois dans l'ensemble de test).
20 itérations seront suffisantes pour considérer un échantillons assez grand de taux d'erreur, ce qui nous permet d'appliquer le Théorème Centrale Limite (TCL) : notre échantillon d'estimation suit donc une loi normale.
Cas Gaussien et variance inconnue, nous calculerons ainsi un intervalle de confiance de

Student à 95%, pour obtenir plus de précision sur les résultats obtenus (estimations d'erreur très variables).

Remarque : Pour la régression logistique et quadratique, nous travaillerons avec une ordonnée à l'origine (intr = T) et un seuil de convergence égale (1e5).

	Synth1-1000	Synth2-1000	Synth3-1000
discrimination			
adq	0.064	0.081	0.076
IC	[0.056,0.071]	[0.075,0.088]	[0.071,0.082]
adl	0.075	0.091	0.103
IC	[0.068,0.081]	[0.086,0.096]	[0.096,0.111]
nba	0.086	0.087	0.089
IC	[0.079,0.092]	[0.078,0.096]	[0.082,0.096]
top	**adq**	**adq≃nba**	**adq**
regression			
log	0.035	0.075	0.044
IC	[0.032,0.039]	[0.068,0.082]	[0.040,0.048]
quad	0.034	0.064	0.043
IC	[0.029,0.038]	[0.057,0.069]	[0.037,0.048]
top	≃	**quad**	≃
arbre			
arbre	0.044	0.069	0.062
IC	[0.039,0.049]	[0.066,0.073]	[0.054,0.071]

Frontières de décision obtenues Nous avons sélectionné pour chaque jeu de données le meilleur classifieur ainsi que le moins bon puis nous avons choisi de ne représenter les frontières de décision obtenues seulement pour ces classifieurs, afin de montrer comment les performances se traduisent graphiquement aussi. cf. Annexe B.

Interprétation

Premier constat, les taux d'erreur de tous nos classifieurs et ce pour toutes nos données sont relativement faibles (entre 3 et 10%). Ces résultats sont bien meilleurs que ceux obtenus avec les classifieurs euclidien et kppv du TP précédent.

Sur Synht1-1000, le meilleur classifieur est la régression, logistique ou quadratique.
En observant les a.d une à une, l'a.d quadratique reste la meilleure des trois méthodes de discrimination, même si l'intervalle de confiance (IC) de son taux d'erreur chevauche légerement celui de l'a.d linéaire. L'a.d quadratique est plus efficace que l'a.d linéaire car l'hypothèse d'homoscedasticité n'est pas vérifiée dans Synth1-1000 (cf analyse des données plus haut). De même, l'hypothèse d'indépendance conditionelle n'est pas vérifiée, il est donc normal que le classifieur baysien naïf soit moins efficace.
La regression est ici plus performante que les a.d car elle ne se base pas sur des hypothèses (dont aucune n'est vérifiée ici). L'arbre reste meilleur que les a.d, mais n'arrive pas à surpasser les performances de la regression sur ce jeux de données.

Sur Synth2-1000, les meilleur classifieurs sont l'arbre de décision et la regression (la quadratique est légèrement plus efficace ici).
En observant les a.d une à une, nous constatons étrangement que le classifieur baysien naïf n'est pas tellement plus efficace que les autres a.d. En effet, l'hypothèse d'indépendance conditionnelle était presque vérifiée ici (matrice de covariance quasi-diagonale) pour chaque classe de Synth2-1000. Notons que l'IC à 95‰ de nba est très large par rapport aux autres a.d, ses résultats sont beaucoup plus variables. En revanche, l'a.d quadratique est plus performante que l'a.d linéaire, cela s'explique probablement par la dispersion des données de Synth2-1000 : Ici, une frontière quadratique fait plus de sens car les deux nuages ne sont pas 'parallèles' dans l'espace.
En se penchant plus sur l'arbre obtenu (cf. figure 9), nous observons qu'il ne possède que deux feuilles (après élagage), il est donc extrêmement robuste car il ne s'est pas trop attaché aux particularité locales des données et reste pour autant l'un de des deux meilleurs classifieurs sur Synth2-1000. Cette raison pourrait nous faire pencher pour cet arbre si un choix devait être fait entre régression quadratique et arbre de décision ici.

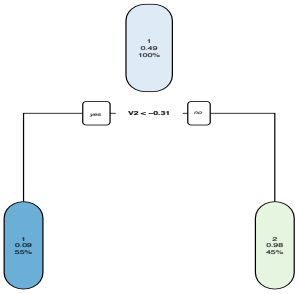


FIGURE 9 – Arbre de décision obtenu sur Synth2-1000

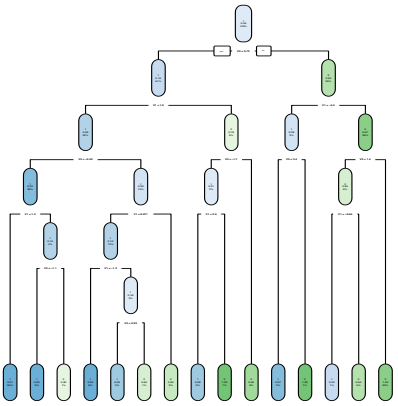


FIGURE 10 – Arbre de décision obtenu sur Synth3-1000

Sur Synth3-1000, la regression (logistique et quadratique) est largement plus performante que les autres classifieurs.
En observant une à une les a.d nous remarquons que l'a.d quadratique est étrangement la meilleur des a.d. En effet, l'hypothèse d'homoscédatiscité est presque vérifiée sur ce jeu de données, on aurait pu penser que l'a.d linéaire soit plus performante. Une raison pour ce résultat serait la forte intersection des deux classes (cf figure 8). L'arbre de décision est ici moins performant que la régression car les arbres obtenus pour Synth3-1000 sont très flexibles mais pas du tout robustes (malgré l'elagage, il possède beaucoup de feuilles et de noeuds (cf figure 10)). Il a effectué un sur-apprentissage sur le modèle et perd donc des performances sur les nouvelles données (ici sur l'ensemble de Test).

0.3.3 Analyse des données réelles

A nouveau, nous commencerons par étudier le paramètre de dispersion et nous afficherons nos nuages de points avant de commencer notre étude sur les performances de nos classifieurs. Nous étudions ici des jeux de données réelles, les dimensions ne sont plus p = 2 mais bien plus complexes (7 pour Pima et 9 pour Breast-cancer-wisconsin). Rappelons que plus la dimension d'un ensemble est élevée plus le nombre d'individus doit être élevé si l'on veut tirer du sens avec des classifieurs (fléau de la dimension). Ici nApp = 355 pour Pima et nApp = 455 ce qui est très faible au vue de la dimension des jeux de données. Gardons donc en tête que nos jeux de données, ici, ne sont pas beaucoup étoffés et que cela risque d'impacter les performances de nos classifieurs.
Afin de représenter nos ensembles mutlidimensionelles en deux dimensions (plan), nous ferons une ACP et nous les représenterons dans le premier plan factoriel. Parallèlement, nous estimerons la matrice de covariance en utilisant les deux premières composantes principales comme jeux de données.
Attention : le problème de la dimension peut entraîner des matrices de variances mal conditionnées, c'est à dire que le conditionnement associé au problème est élevé (difficulté d'obtenir

des résultats de calculs numériques sans être entachés d'erreur). Nous garderons donc sous réserves les résultats obtenus dans cette section. Attention 2 : nous utiliserons ici l'ACP qu'à des fins de représentation. Le sens que nous tirons des figures (classes plus ou moins bien discriminées par exemple) devra être relativiser car l'ACP ne cherche pas forcément les axes factoriels les plus discriminants.

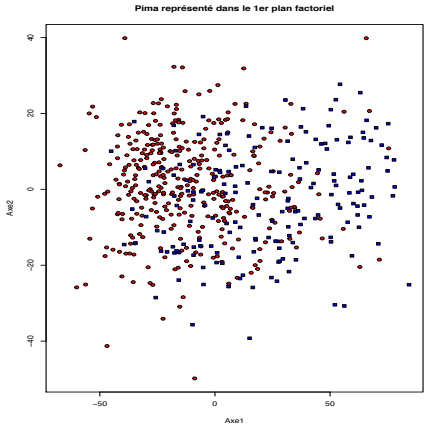


FIGURE 11 – ACP sur pima | 1er plan factoriel

$$\begin{cases} \Sigma_1 = & \begin{pmatrix} 605.40 & -2.80 \\ -2.79 & 174.68 \end{pmatrix} \\ \Sigma_2 = & \begin{pmatrix} 968.24 & 96.68 \\ 96.68 & 184.04 \end{pmatrix} \end{cases}$$

Sur la figure, on constate que les deux classes sont presque complètement confondues, il n'y a pas de réelle frontière entre deux classes homogènes. La dispersion des deux classes n'est pas la même (visible aussi en comparant les deux matrices de covariances du premier plan factoriel), mais on pourrait eventuellement considérer qu'elles sont grossierement semblables.

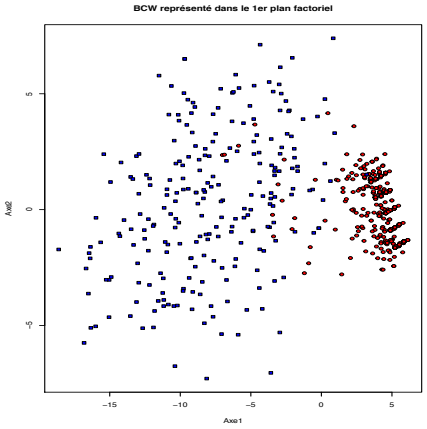


FIGURE 12 – ACP sur bcw | 1er plan factoriel

$$\begin{cases} \Sigma_1 = & \begin{pmatrix} 3.09 & -0.91 \\ -0.91 & 1.58 \end{pmatrix} \\ \Sigma_2 = & \begin{pmatrix} 18.81 & 4.77 \\ 4.77 & 9.63 \end{pmatrix} \end{cases}$$

Sur la figure, on constate que l'on peut relativement bien distinguer deux classes homogènes (ou presque). Il semble en effet exister une frontière entre les deux classes. Les deux matrices de covariances du premier plan factoriel conditionnellement à chaque classes ne sont pas du tout égales, en revanche (ni même semblables ou proportionelles). Ce qui est d'ailleurs visible sur la figure 12, un nuage est dense tandis que l'autre est très éparse et montre une répartition où les axes factoriels sont quasi-indépendants conditionnellement à la classe 1(Σ_1 est presque diagonale).

0.3.4 Etude des performances sur données réelles

Cette fois nous répèterons N = 100 fois le calcul de chaque taux d'erreur afin d'augmenter la précision de nos résultats (qui sont déjà biaisés par le faible volume de données à notre disposition). Remarque : nous ne calculerons pas les erreurs de la regression quadratique sur B-C-W.

	Pima	B-C-W
discrimination		
adq	0.349	0.227
IC	[0.343,0.355]	[0.197,0.258]
adl	0.341	0.256
IC	[0.338,0.345]	[0.244,0.268]
nba	0.334	0.089
IC	[0.329,0.340]	[0.066,0.111]
top	**≃nba**	**nba**
regression		
log	0.220	0.040
IC	[0.215,0.224]	[0.038,0.042]
quad	0.249	X
IC	[0.244,0.254]	[X,X]
top	**log**	**log**
arbre		
	0.245	0.054
IC	[0.239,0.251]	[0.052,0.057]

Interprétation
Un premier constat sur Pima est que les taux d'erreurs sont bien plus elevés que ceux estimés sur les données simulées et sur Breast-cancer-wisconsin (près de 20% de plus). Cela se comprend au vue de la répartition des données dans l'espace (cf figure 11). En effet, comme noté précédemment, les nuages des deux classes sont très confondus.

Sur Pima, le meilleur classifieur est la regression logistique non quadratique. En comparant les a.d une par une, nous constatons qu'il n'y a pas de réel 'meilleur' classifieur (leurs IC se chevauchent). La regression logistique est meilleure que la quadratique. Ici, une raison pourrait être le fléau de la dimension. En effet, la regression quadratique augmente signifativement la dimension p de l'ensemble d'apprentissage et de test afin d'amplifier les distances entre les points, ce qui lui permet donc de classer plus facilement. La dimension de départ était déjà elevée (p = 7 sur Pima) pour un volume d'individus faible. En amplifiant la dimension de l'ensemble, l'effet du fléau de la dimension est aussi amplifié : ce qui impacte négativement et significativement les performances de la régression quadratique (près de 3‰ d'écart avec le taux d'erreur de la logistique). L'arbre est assez performant aussi car un arbre de décision n'est pas tellement impacté par le fléau de la dimension.

Sur Bcw, un constat frappant est la performance du classifieur naïf par rapport aux autres a.d (écart de près de 12%). Une raison pourrait être l'indépendance des variables conditionnellement à la classe 1 (matrice de covariance du premier axe factoriel 1 est diagonale et pourrait être représentatif du comportement des variables originelles).
Le meilleur classifieur pour Bcw reste la regression logistique. Une raison, à nouveau, pourrait être le manque de robustesse des classifieurs paramétriques (ici a.d) qui se ressent encore plus dans un contexte où le volume de données est faible et la dimension élevée. Sur Bcw, l'arbre de décision propose des performances intéressantes pour les mêmes raisons que celles concernant l'abre appliqué sur Pima.

0.4 Challenge : données Spam

Dans cette partie, nous travaillons sur un problème de détection de spams à partir d'indicateurs calculés sur des messages électroniques. Le jeu de données contient n = 4601 individus décrits chacun par p = 57 variables explicatives. Nous voulions dans un premier temps effectuer la même procédure que celle appliquée aux jeux précédents (séparation, apprentissage et évaluation des performances) t fois avant d'évaluer les taux moyens d'erreur de test pour les trois modèles d'analyse discriminante, la régression logistique classique, la régression logistique quadratique et les arbres de décision. Cependant, ce jeux de données est particulier. Sa dimension très élevée en fait un dataset complexe et donc nécessitant un pré-traitement. Il ne fait en effet pas tellement de sens d'appliquer nos différents classifieurs sans considérer un *feature-selection* (choix de variables significativement discriminantes) au préalable. Rappelons avant tout, que dans un problème discriminatoire comme celui-ci, le juge de paix (l'indicateur de bonne ou mauvaise méthode) reste le taux d'erreur final obtenu. Notre objectif ici est donc de trouver une méthode qui propose les taux d'erreurs les plus faibles pour un ou plusieurs classifieurs.

Une première technique "violente" serait de tester toutes les combinaisons possibles de variables et évaluer nos performances pour chaque sous espace considéré. Ici nous avons 57 variables descriptives ce qui ferait $2^{58} = 2.88230376e17$ combinaisons possibles, ce qui est bien évidemment impossible.

Nous allons aborder le problème en trois points. Premièrement, nous allons chercher à estimer les taux d'erreur de nos classifieurs sur nos données brutes afin de possèder une sorte de *taux de référence*. Notons que les données seront toutefois centrées et réduites afin de considérer les dispersions (très élevées ici) de notre jeu de données. Une seconde méthode sera d'utiliser, avec précaution, l'Analyse en composante Principale (ACP). Avec précaution pourquoi ? Car l'ACP est une méthode de réprésentation des variables sur des axes plus réprésentatifs mais pas forcément plus discriminants. Or, étant ici dans un problème de discrimination, l'ACP ne semble au premier abord pas faire beaucoup de sens, mais nous verrons en quoi elle reste intéressante à considérer.
Finalement, nous utiliserons une dernière méthode qui se rapproche le plus des méthodes actuelles de *feature-selection*. Nous allons utiliser les arbres de décisions pour estimer le taux *d'importance* des variables afin de ne récupérer que les plus significativement intéressantes pour discriminer correctement nos individus. Nous détaillerons ici en particulier la justification de la méthode des randoms forest.

Nous testerons ensuite les différentes méthodes afin de voir quelle est la meilleure au sens du plus faible taux d'erreur obtenu. Ici, nous n'estimerons pas les analyses discriminantes car le test de shapiro nous a indiqué que le vecteur de variable descriptif ne suit pas de loi normale. L'Hypothèse du cas gaussien, n'est donc pas vérifiée.

Méthode centrage - réduction sur X

Nous avons commencé par centrer et réduire l'ensemble de données. Après avoir séparé l'ensemble en deux ensembles, un d'apprentissage Xapp et un de test Xtst, nous avons centré et réduit Xapp. Nous avons pris garde de conserver les moyennes et les ecarts types de chaque colonne afin de pouvoir soustraire a chaque colonne de Xtst la moyenne de la colonne correspondante dans Xapp et de même nous avons pu diviser chaque colonne de Xtst par l'écart-type de chaque colonne de Xapp. Ainsi on obtient des données indépendantes de l'unité ou de l'échelle choisie ainsi que des variables ayant la même moyenne et la même dispersion. On peut alors comparer plus aisément les variations. Centrer-réduire les variables est donc très utile en analyse de données car cela équivaut à un changement d'unité, et n'a pas d'incidence sur les profils de variation. Remarquons aussi que dans le cas de la regression, le fait de centrer / réduire permettra d'inverser la matrice hessienne (algo de Newton-Raphson) utilisée dans la regression car nous limitons ses valeurs propres trop grandes ou trop petites.

Les valeurs des coefficients de corrélation entre variables centrées réduites demeurent identiques à ce qu'elles étaient avant l'opération de centrage et réduction.
Nous obtenons les taux d'erreur suivants :

	scaled
regression	
log IC	0.121 [0.108,0.134]
quad IC	0.117 [0.110,0.115]
tree	
	0.191 [0.165,0.217]

Nous pouvons remarquer que les taux d'erreur trouvés sont relativement proches. Cette méthode ne permet donc pas d'élire un classifieur significativement meilleur que l'autre. Nous trouvons des taux d'erreurs 'de référence' que nous allons essayer de peaufiner en appliquant les méthodes suivantes.

Méthode ACP : sélection naïve

Comme mentionné en Introduction de cette section sur SPAM, l'ACP ne permet pas de révéler avec certitude les axes discriminants. Or c'est ce que nous cherchons. Nous allons ici utiliser l'ACP pour tester presque aveuglement, si les axes factoriels obtenus sont bel et bien discriminants. Nous testerons d'abord avec une

matrice d'individus non réduite, puis sur une matrice d'individus réduite. Réduire permettra en effet de 'tasser' les nuages de points et donc de diminuer l'inertie expliquée par chaque axe factoriel trouvé. Nous procéderons de la sorte car nous pensons que la première ACP serait trop 'violente' dans le sens où elle réduira à un espace de trop faible dimensions les individus, ce qui risque d'entraîner une trop grosse perte d'informations discriminantes que proposaient les variables dans la base d'origine. L'ACP sur données réduites sera plus 'douce' en terme d'écart des inerties expliquées par les axes factoriels obtenus par cette dernière.
Il est très important de transposer nos individus de test dans le même espace factoriel que les individus d'apprentissage. En effet le classifieurs doit apprendre sur un espace Omega, qui doit être le même que l'espace sur lequel il va être testé. Nous calculerons donc, pour les deux ACPs effectuées ici, la matrice Uapp des axes factoriels sur Xapp. Nous utiliserons cette même matrice Uapp pour le calcul des composantes principales des individus de test Ctest et d'apprentissage Capp : cela nous assure que les individus seront représentés dans le même espace Uapp avant d'être utilisés par nos classifieurs. Remarque : que ce soit pour le centrage ou pour la réduction, nous utiliserons à chaque fois l'ensemble d'apprentissage pour récupérer les paramètres (moyenne pour centrage ou ecart type pour réduction). Nous utiliserons ces paramètres pour centrer - réduire notre ensemble de Test.

ACP sans réduction Nous obtenons plus de 99% d'inertie expliquée avec les deux premiers axes factoriels, et près de 99,99‰ en considérant les 3 premiers axes. Nous choisirons donc ici d'utiliser le premier cube factoriel obtenu pour représenter nos individus et utiliser nos classifieurs. (cf. figure 13)

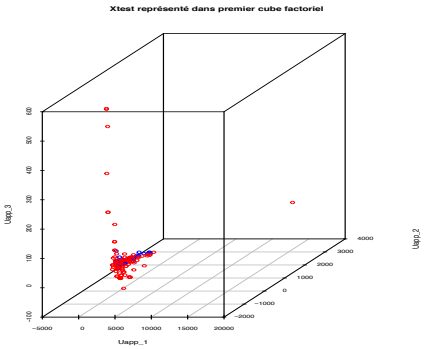


FIGURE 13 – Un exemple d'ensemble Xtst représenté dans le premier 'cube factoriel'

La figure 13 montre clairement que le premier cube factoriel obtenu ne met pas en évidence, de manière significative, la distinction entre les classes.

ACP avec réduction : sélection naïve Nous choisissons cette fois de réduire la matrice Xapp avant de calculer la matrice des axes factoriels. Nous avons choisi de représenter, dans le graphique ci-dessous, les pourcentages cumulés des inerties expliquées par les axes factoriels obtenu. (cf figure. 14)

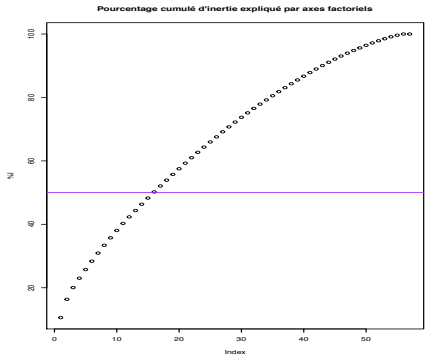


FIGURE 14 – Pourcentages cumulés des inerties expliquées par axes factoriels

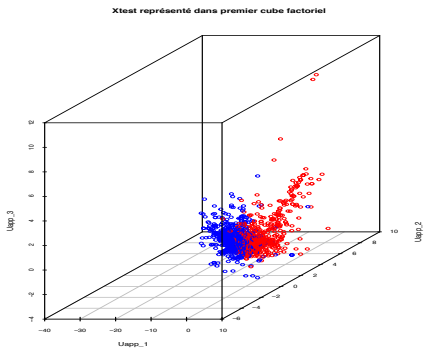


FIGURE 15 – Un exemple d'ensemble Xtst représenté dans le premier 'cube factoriel'

La figure. 14 montre bien que l'inertie est beaucoup moins bien expliquée par les premiers axes factoriels (le premier cube n'expliquerait que légèrement plus de 20‰ de l'inertie totale du nuage. En revanche la figure. 15, nous montre que les données sont bien mieux discriminées (des classes distinctes et un peu plus homogènes apparaissent), et ce, en ne considérant dans la figure que le premier cube factoriel. Nous allons donc choisir d'élargir les axes factoriels considérés en conservant les 16 premiers pour cette seconde ACP, car ils expliquent ensemble plus de 50% de l'inertie totale du nuage (cf figure. 14, courbe violette) et nous avons vu que déjà les 3 premiers proposaient des résultats intéressants. Nous jugerons de la validité de notre bon sens en regardant les taux d'erreurs obtenus.

Evaluation des performances

	ACP	ACP post-réduction
discrimination		
adq	0.378	0.237
IC	[0.376, 0.380]	[0.235, 0.239]
adl	0.380	0.187
IC	[0.378, 0.381]	[0.184, 0.189]
nba	0.373	0.285
IC	[0.370, 0.375]	[0.280, 0.291]
top	**≃nba**	
regression		
log	0.261	0.093
IC	[0.257,0.266]	[0.091,0.096]
quad	0.326	X
IC	[0.324,0.328]	X X
top	**log**	
arbre		
tree	0.228	0.107
IC	[0.224, 0.233]	[0.104,0.111]

0.4.1 Méthode arbres de décisions (random forest)

Pour élaborer nos arbres de décision au cours de ce TP, nous avons utilisé des packages R enrichis qui proposent des fonctionnalités intéressantes. Un arbre de décision estime, entre autres, l'importance de chaque variable d'un espace en se basant sur le critère de GINI. Nous allons donc utiliser ces estimations d'importances de variables comme outils de sélection de variable.
Cependant, un arbre de décision se base sur une initialisation qui lui est propre. Chaque arbre de décision estime sa propre importance de variables en fonction de comment l'espace est découpé pour établir leur décision. Cette estimation d'importance des variables d'un seul arbre est donc peu fiable. En revanche, effectuer n arbres aléatoires et récupérer leur estimation globale de l'importance de chaque variable peut être très fiable. Nous allons donc appliquer cette méthode de random forest avec un package spécial *randomForest* qui nous retournera cette information d'importance des variables.

choix des variables Nous utilisons ici l'indice de GINI qui estime l'influence qu'a une variable sur l'impureté d'un noeud et donc de sa force discriminante : plus l'indice de Gini sera

faible, meilleur sera la variable en terme de discrimination. Le random forest nous permet de faire, ici la moyenne de ces estimations d'influence sur 500 arbres générés aléatoirement. Sur la figure ci-dessous nous présentons les résultats obtenus par la random forest appliquée sur l'ensemble totale.

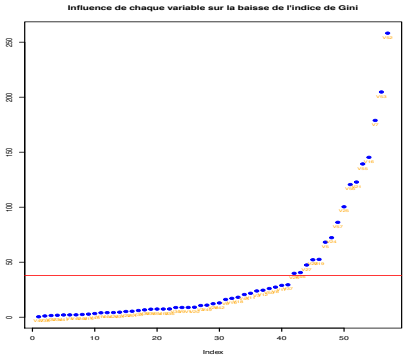


FIGURE 16 – Moyenne de la baisse de l'indice de Gini pour chaque variable sur l'ensemble totale

Nous choisirons de ne garder que les variables ayant une influence significative sur la baisse de l'indice de Gini. Sur la figure 16 la ligne rouge montre la frontière (baisse sur i_{gini} = 38) que nous avons choisie. Nous l'avons choisie car elle représente là où les écarts d'influence commencent à réellement devenir significatif. Ce choix est relativement subjectif, le calcul du taux d'erreur obtenu en ne gardant que les variables choisies, nous permettra, ou non, de valider notre idée.

Nous considérons donc l'espace Ω_{16} définies par les variables suivantes pour évaluer nos performances : V26, V46, V27, V23, V19, V5, V24, V57, V25, V56, V21, V55, V16, V7, V53, V52.

Remarque : Nous n'étudierons pas la regression quadratique ici.

Evaluation des performances

	Ω_{16}
discrimination	
adq IC	0.243 [0.242, 0.245]
adl IC	0.196 [0.195,0.198]
nba IC	0.242 [0.240,0.243]
top	**≃adl**
regression	
log IC	0.093 [0.090,0.097]
arbre	
tree IC	0.082 [0.079,0.086]

0.4.2 Conclusion sur Spam

Nous constastons que les résultats des différentes ADs sont bien moins bons par rapport aux autres classifieurs non paramétriques. Cela est probablement dû au fait que l'hypothèse Gaussienne n'est pas vérifiée ici. La réduction de variables proposée par l'ACP propose des performances sur les classifieurs non paramétriques meilleures que celles obtenues sur le jeux de données en dimension d'origine. Parmi tous nos résultat obtenus, le classifieur proposant le meilleur taux d'erreur reste l'arbre de décision après avoir sélectionné les variables significativement discriminantes estimées par la méthode de random-forest. Nous obtenons en effet un taux d'erreur d'environ 8‰.
Une autre méthode de sélection de variables aurait pu être l'utilisation de la régression qui permet d'interpréter les coefficients et leur significativité avec le test de Wald.

0.5 Conclusion

Ce dernier TP nous aura permi de tester différents classifieurs sur des jeux de données diverses et de volume, dispersion et répartition différentes. Les performances des différents classifieurs paramétriques dépendent très largement de la vérification des différentes hypothèses (en particulier, ici, pour les analyses discriminantes). En revanche, les classifieurs non paramétriques semblent moins sensibles à la ré-

partition des données. En effet, ici la régression et les arbres de décision proposaient relativement fréquemment et pour tous les types de jeux de données des performances intéressantes. Finalement, avec le jeux de données Spam, nous avons eu l'occasion de pratiquer un protocole typique d'analyse de données et en particulier ici de pré-traitement. En effet, nous avons cherché, parmi les différentes méthodes que nous connaissions, celles qui pouvaient sélectionner les variables significativement plus discriminantes. Nous avons appliqué, relativement naïvement, nos méthodes de sélection de variables (en particulier l'ACP qui était relativement aveugle) pour tâtonner et chercher des techniques pour peaufiner l'espace considéré en cherchant les variables intéressantes. Au final, nous avons réussi à trouver de meilleurs résultats après sélection de variables par méthode de forêt aléatoire.

Annexe A

Analyse Discriminante

```r
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)
    {
        indk <- which(zapp==k)
        Xappk <- Xapp[as.numeric(rownames(Xapp)) %in% indk,]

        param$MCov[,,k] <- cov(Xappk)
        param$mean[k,] <- colMeans(Xappk)
        param$prop[k] <- length(indk)/n
    }

    param
}
```

FIGURE A.1 – Analyse Discriminante Quadratique

```r
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)
    {
        indk <- which(zapp==k)
        Xappk <- Xapp[as.numeric(rownames(Xapp)) %in% indk,]

        MCov <- MCov + length(indk)*cov(Xappk)
        param$mean[k,] <- colMeans(Xappk)
        param$prop[k] <- length(indk)/n
    }
    MCov <- MCov/n


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

    param
}
```

FIGURE A.2 – Analyse Discriminante Linéaire

```r
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)
    {
        indk <- which(zapp==k)
        Xappk <- Xapp[as.numeric(rownames(Xapp)) %in% indk,]

        param$MCov[,,k] <- diag(diag(cov(Xappk)))
        param$mean[k,] <- colMeans(Xappk)
        param$prop[k] <- length(indk)/n
    }

    param
}
```

FIGURE A.3 – Classifieur Baysien Naïf

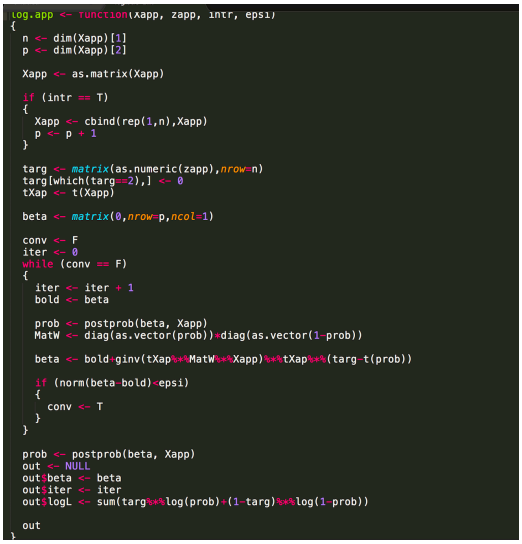
```r
nba.app <- function(Xapp, zapp)
{
    n <- dim(Xapp)[1]
    p <- dim(Xapp)[2]
    g <- max(unique(zapp))

    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)
    {
        indk <- which(zapp==k)
        Xappk <- Xapp[as.numeric(rownames(Xapp)) %in% indk,]

        param$MCov[,,k] <- diag(diag(cov(Xappk)))
        param$mean[k,] <- colMeans(Xappk)
        param$prop[k] <- length(indk)/n
    }

    param
}
```

FIGURE A.4 – Classifieur Baysien Naïf

```r
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] <- (param$prop[k]*mvdnorm(Xtst, param$mean[k,], param$MCov[,,k]))
    }
    prob <- prob / rowSums(prob, na.rm = FALSE, dims = 1)
    pred <- max.col(prob)

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

    out
}
```

FIGURE A.5 – Evaluation des ad

Regression Logistique



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

  Xapp <- as.matrix(Xapp)

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

    prob <- postprob(beta, Xapp)
    MatW <- diag(as.vector(prob))*diag(as.vector(1-prob))

    beta <- bold+ginv(tXap%*%MatW%*%Xapp)%*%tXap%*%(targ-t(prob))

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

  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 A.6 – Regression Logistique apprentissage

```
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 <-  cbind(t(exp(t(beta) %*% t(Xtst))/(1+exp(t(beta) %*% t(Xtst)))),
                 t(1- exp(t(beta) %*% t(Xtst))/(1+exp(t(beta) %*% t(Xtst)))))
  pred <- max.col(prob)

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

  return(out)
}

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

FIGURE A.7 – Regression Logistique évaluation

Annexe B

Synth1-1000

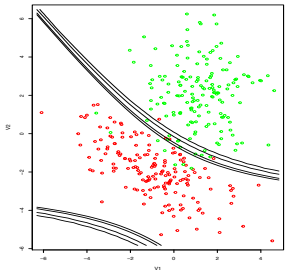


FIGURE B.1 – Synth1 - Meilleur | Log Quadra

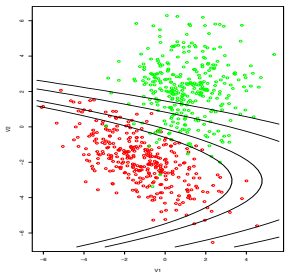


FIGURE B.2 – Synth1 - Moins bon | NBA

Synth2-1000

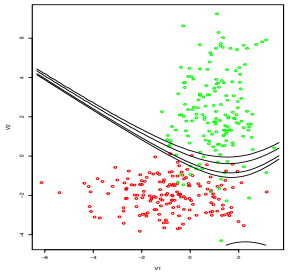


FIGURE B.3 – Synth2 - Meilleur | Log Quadra

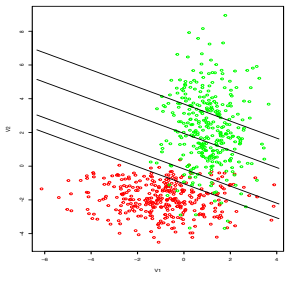


FIGURE B.4 – Synth2 - Moins bon | ADL

Synth3-1000

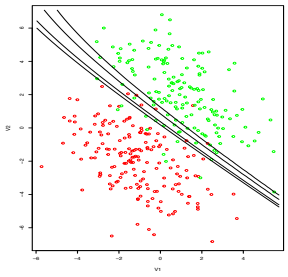


FIGURE B.5 – Synth3 - Meilleur | Log Quadra

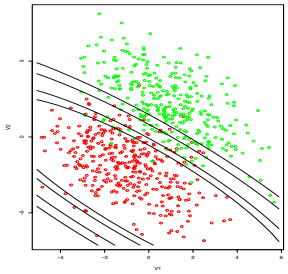


FIGURE B.6 – Synth3 - Moins bon | NBA