UNIVERSITÉ DE TECHNOLOGIE DE COMPIÈGNE

RAPPORT TP4 SY09

Discrimination

Minh Tri LÊ GI02

Yanqing ZENG GI05

24 juin 2017

Table des matières

Table des figures

Liste des tableaux

Introduction

Les objectifs de ce TP sont d'une part d'étudier diverses méthodes de classifieurs sur différents jeux de données afin d'en comparer les performances. D'autre part, de pouvoir interpréter les résultats obtenus afin d'être capable de choisir le classifieur le plus adapté aux problème par la suite.

1 Programmation

Le code des 3 modèles d'analyse discriminante, ainsi que la fonction effectuant les prédictions en fonction des paramètres [D] sont fournis en annexe.

1.1 Analyse discriminante

Nous ferons quelques rappels sur les hypothèses et les estimateurs spécifiques à chaque méthode (Analyse discriminaten quadratique, analyse discriminante linéaire, classifieur bayésien naïf).

Pour les 3 classifieurs, on émet l'hypothèse suivante : La matrice X suit conditionnellement à chaque classe w_k une loi normale multidimensionnelle d'espérance μ_k et de variance $\sum_k$.

Chaque classifieur se différencie d'un autre en fonction des hypothèses faites sur les paramètres de la loi normale.

La loi normale multidimensionnelle pour chaque classe k s'énonce comme-ci :

$$f_k(x) = \frac{1}{(2\pi)^{p/2}(det\sum_k)^{\frac{1}{2}}} exp\left(-\frac{1}{2}((x-\mu_k)^T)\sum_k^{-}1(x-\mu_k)\right)$$

La règle de Bayes s'écrit alors :

$$\delta^*(x) = arg\ \max_k\ P(w_k|x)$$

Avec

$$P(w_k|k) = \frac{\pi_k f_k(x)}{f(x)}$$

(pour $g = 2$ classes)

$$= P(w_k|k) = \frac{\pi_k f_k(x)}{\pi_1 f_1(x)) + \pi_2 f_2(x)}$$

1.2 Analyse discrimante quadratique

L'analyse discriminante quadratique considère le cas général : La distribution des variables dans chaque classe est caractérisée par des μ_k et $\sum_k$ différents.

Pour l'estimation des paramètres, nous avons choisi les estimateurs sans biais pour la proportion $\widehat{\pi_k}$, $\widehat{\mu_k}\widehat{\sum_k}$

$$\widehat{\pi_k} = \frac{n_k}{n},$$

$$\widehat{\mu_k} = \overline{x_k} = \frac{1}{n_k}\sum_{i=1}^{n} z_{ik}x_i,$$

$$\widehat{\Sigma_k^*} = V_k^* = \frac{n_k}{n_k - 1}V_k$$

avec

$$V_k^* = \frac{1}{n_k}\sum_{i=1}^{n} z_{ik}(x_i - \widehat{\mu_k})(x_i - \widehat{\mu_k})^T$$

Le code est en annexe A.

1.3 Analyse discrimante linéaire

L'analyse discriminante linéaire considère que la variance est commune à toutes les classes (hypothèse d'homoscédasticité) : $\Sigma_k = \Sigma,\ k \in \{1, ..., g\}$.

De plus, l'estimation de $\widehat{\pi_k}$ et $\widehat{\mu_k}$ est identique aux paramètres de l'analyse discrimnante linéaire.

Nous avons choisi les estimateurs sans biais pour la variance $\widehat{\sum_k}$:

$$V_W^* = \frac{1}{n-g}\sum_{k=1}^{g}(n_k - 1)V_k^*$$

Le code est en annexe B.

1.4 Classifieur bayésien naïf

Classifieur bayésien naïf considère que les variables X_j sont indépendantes entre-elles conditionnellement à Z, ce qui revient à supposer que les matrices de variance $\sum_k$ sont diagonales.

De plus, les estimations de π_k et μ_k sont les mêmes que précedemment.
Pour la variance, on a :

$$\widehat{\Sigma_k} = diag(s_{k1}^2, ..., s_{kj}^2, ..., s_{kp}^2)$$

$$\widehat{\Sigma_k^*} = diag(V_k)$$

Le code est en annexe C.

1.5 Régression logistique

La méthode de régression logistique consiste à modéliser les probabilités à posteriori $P(w_k|x)$ par des fonctions de x. Dans ce TP, nous appliquons la fonction logit, qui est le logarithme du rapport des probabilité a posteriori $P(k_k|x)$ et $P(w_g|x)$.

$$ln\frac{P(w_k|x)}{P(w_g|x)} = w_k^T x$$

Pour estimer les valeurs de composantes de w, on va maximiser la vraisemblance :

$$logL(w; t_1, t_2, ..., t_n) = \sum_{i=1}^{n}(t_i W^T x_i - log(1 + exp(w^T x_i)))$$

Le gradient est donc :

$$\frac{\delta logL(w)}{\delta w} = X^T(t - p)$$

On cherche donc les paramètres tels que :

$$X^T(t - p) = 0$$

On ne peut pas résoudre ce système directement, donc on utilise : l'algorithme de Newton Raphson pour trouver des valeurs qui approchent la solution optimale locale par itération successive.

$$w^{(k+1)} = w^{(q)} + (X^T W_{(q)} X)^{-1} X^T (t - p^{(q)})$$

Enfin, notre fonction de régression logistique admet une variable booléenne pour l'ajout ou non d'une ordonnée à l'origine aux vecteurs x_i d'individus. Cela permet d'ajout une dimension à notre frontière de décision et ajoute de la flexibilité au classifieur.
Dans ce TP nous tenterons de si cet ajout est pertinent pour la prédiction de nos jeux de données.

Le code est en annexe E.

Pour la méthode de régression logistique quadratique, on a construit la matrice quadratique par la fonction quadratique(X), le code est en annexe G. La régression logistique quadratique est plus flexible mais nécessite d'estimer un nombre de paramètres plus important ($\frac{p(p+3)}{2}$).

1.6 Vérification des fonctions

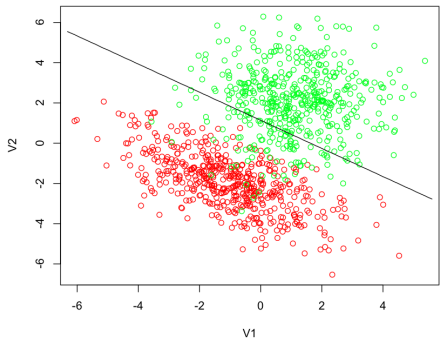

FIGURE 1 – Vérification : adl.app

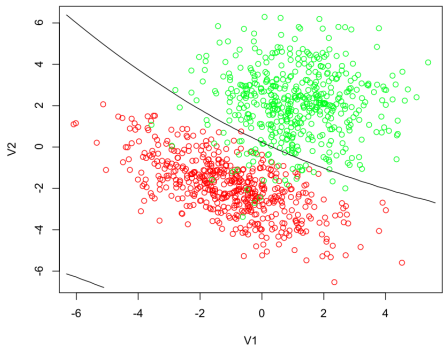

FIGURE 2 – Vérification : adq.app

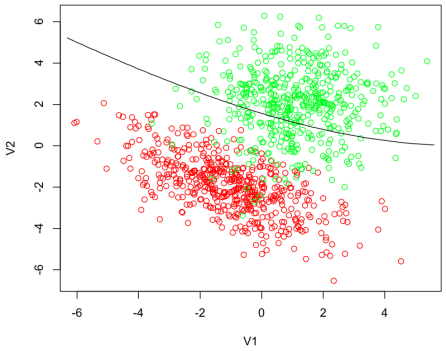

FIGURE 3 – Vérification : nba.app

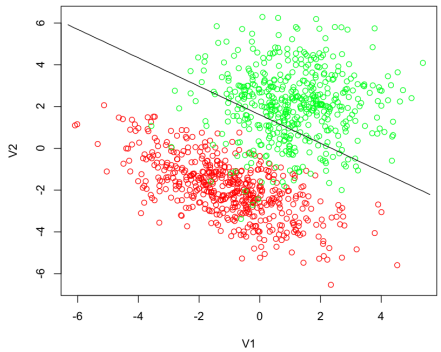

FIGURE 4 – Vérification : logistique

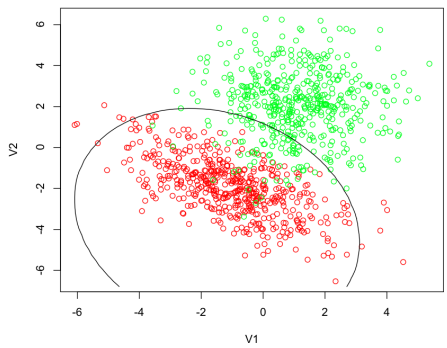

FIGURE 5 – Vérification : logistique quadratique

2 Application

2.1 Test sur des données simulées

On a appliqué la méthode de analyse discriminante quadratique, linéaire, classifieur bayésien naif et la méthode de régression logistique (et quadratique) sur les jeux de données simulées, qui suivent une distribution gaussienne.

Au point de vue de la complexité temporelle, nous avons constaté que la méthode de l'arbre de décision est la moins coûteuse, car c'est un modèle non paramétrisé. Il a la variance la plus élevé, car il se concentre plus sur les individus, et le modèle varie beaucoup selon la partition des données faite.

2.1.1 Synth1-1000

TABLE 1 – Calcul du taux d'erreur d'apprentissage ε et de test pour le jeu de données Synth1-1000 avec plusieurs méthodes (arrondi à 10^{-4})

Méthode	**Données**	$\overline{\varepsilon}$	IC
Analyse discriminante quadratique	Apprentissage	0.0312	[0.0228; 0.0395]
	Test	0.0310	[0.0192; 0.0428]
Analyse discriminante linéaire	Apprentissage	0.0420	[0.0346; 0.0494]
	Test	0.0455	[0.0309; 0.0601]
Classifieur bayésien naïf	Apprentissage	0.0419	[0.0338; 0.0500]
	Test	0.0436	[0.0236; 0.0636]
Régression logistique (intercept=1)	Apprentissage	0.0327	[0.0235; 0.0419]
	Test	0.0325	[0.0183; 0.0466]
Régression logistique (intercept=0)	Apprentissage	0.0417	[0.0334; 0.0501]
	Test	0.0412	[0.0277; 0.0546]
Régression logistique quadratique (intercept=1)	Apprentissage	0.0312	[0.0203; 0.0420]
	Test	0.0356	[0.0215; 0.0497]
Régression logistique quadratique (intercept=0)	Apprentissage	0.0420	[0.0342; 0.0498]
	Test	0.0425	[0.0252; 0.0598]
Arbre de décision	Apprentissage	0.0261	[0.0172; 0.0349]
	Test	0.0485	[0.0259; 0.0711]

2.1.2 Synth2-1000

TABLE 2 – Calcul du taux d'erreur d'apprentissage ε et de test pour le jeu de données Synth2-1000 avec plusieurs méthodes (arrondi à 10^{-4})

Méthode	**Données**	$\overline{\varepsilon}$	IC
Analyse discriminante quadratique	Apprentissage	0.0603	[0.0519; 0.0688]
	Test	0.0602	[0.0448; 0.0757]
Analyse discriminante linéaire	Apprentissage	0.0805	[0.0662; 0.0948]
	Test	0.0824	[0.0633; 0.1016]
Classifieur bayésien naïf	Apprentissage	0.0615	[0.0504; 0.0725]
	Test	0.0650	[0.0503; 0.0797]
Régression logistique (intercept=1)	Apprentissage	0.0690	[0.0551; 0.0829]
	Test	0.0731	[0.0548; 0.0915]
Régression logistique (intercept=0)	Apprentissage	0.0720	[0.0555; 0.0885]
	Test	0.0763	[0.0473; 0.1053]
Régression logistique quadratique (intercept=1)	Apprentissage	0.0619	[0.0492; 0.0747]
	Test	0.0643	[0.0399; 0.0887]
Régression logistique quadratique (intercept=0)	Apprentissage	0.0614	[0.0515; 0.0765]
	Test	0.0669	[0.0565; 0.0774]
Arbre de décision	Apprentissage	0.0445	[0.0304; 0.0587]
	Test	0.07808	[0.0552; 0.1009]

2.1.3 Synth3-1000

TABLE 3 – Calcul du taux d'erreur d'apprentissage ε et de test pour le jeu de données Synth3-1000 avec plusieurs méthodes (arrondi à 10^{-4})

Méthode	**Données**	$\overline{\varepsilon}$	IC
Analyse discriminante quadratique	Apprentissage	0.0628	[0.0528; 0.0728]
	Test	0.0656	[0.0450; 0.0862]
Analyse discriminante linéaire	Apprentissage	0.0795	[0.0657; 0.0933]
	Test	0.0827	[0.0567; 0.1087]
Classifieur bayésien naïf	Apprentissage	0.0646	[0.0526; 0.0766]
	Test	0.0662	[0.0521; 0.0803]
Régression logistique (intercept=1)	Apprentissage	0.0410	[0.0355; 0.0465]
	Test	0.0390	[0.0212; 0.0569]
Régression logistique (intercept=0)	Apprentissage	0.0403	[0.0296; 0.0511]
	Test	0.0393	[0.0257; 0.0530]
Régression logistique quadratique (intercept=1)	Apprentissage	0.0396	[0.0297; 0.0494]
	Test	0.0410	[0.0223; 0.0597]
Régression logistique quadratique (intercept=0)	Apprentissage	0.0395	[0.0303; 0.0487]
	Test	0.0362	[0.0242; 0.0482]
Arbre de décision	Apprentissage	0.0362	[0.0234; 0.0490]
	Test	0.0611	[0.0308; 0.0914]

Interprétation : Pour tous les jeux de données simulées (Synth-n), tous les classifieurs ont un taux d'erreur très bas.

La méthode discrimiante quadratique et la régression logistique quadratique avec intercept=1 ont la meilleure performance. Les paramètres à estimer pour ces deux méthodes sont les plus importantes, donc ces modèles ont plus de flexibilité que les autres.

En effet, les modèles analyse discriminante linéaire et classifieur bayésien naïf sont des modèles plus simplifiés. On peut donc appliquer ces deux méthodes si on ne veut pas avoir

un fort coût de la complexité temporelle, comme il y a moins de paramètres à estimer.

Plus généralement, les classifieurs se basant sur la distribution gaussienne ont de très bons résultats sur ces données car cela correspond effectivement à la vraie distribution des données.

2.2 Test sur des données réelles

Pour calculer le taux d'erreur ponctuelle $\widehat{\varepsilon}$ d'une prédiction, il faut simplement compter le nombre de prédictions erronées ε_i entre la prédiction faite et les données réelles (d'apprentissage ou de test), puis diviser par le nombre total d'individus n.

On répète cette procédure N fois pour obtenir N réalisations du taux d'erreur. Pour obtenir l'estimation du taux d'erreur $\bar{\varepsilon}$, il ne reste plus qu'à faire la moyenne des N réalisations.
L'intervalle de confiance à 95% est obtenu comme-ci : $IC = \left[\mu - 1.96 * \frac{s^*}{\sqrt{N}};\ \mu + 1.96 * \frac{s^*}{\sqrt{N}}\right]$.

Avec :

— μ : La moyenne des N réalisations de ε

— s^* : L'écart-type corrigé des N réalisations de ε

— N : Le nombre de réalisations de ε

2.2.1 Pima

Nous avons appliqué les différents modèles d'analyses discriminantes, de régression logistique ainsi que les arbres de décision.

Pour calculer le taux d'erreur sur les modèles d'analyse discriminantes, nous avons choisi d'effectuer $N = 100$ répétitions pour calculer le taux d'erreur et son intervalle de confiance (à 96%). Le résumé des résultats sont en table 4.

TABLE 4 – Calcul du taux d'erreur d'apprentissage ε et de test pour le jeu de données Pima avec plusieurs méthodes (arrondi à 10^{-4})

Méthode	**Données**	$\bar{\varepsilon}$	IC
Analyse discriminante quadratique	Apprentissage	0.2135	[0.2105; 0.2165]
	Test	0.2385	[0.2332; 0.2438]
Analyse discriminante linéaire	Apprentissage	0.2096	[0.2070; 0.2122]
	Test	0.2233	[0.2172; 0.2273]
Classifieur bayésien naïf	Apprentissage	0.2291	[0.2268; 0.2314]
	Test	0.2363	[0.2307; 0.2420]
Régression logistique (intercept=1)	Apprentissage	0.2090	[0.2069; 0.2111]
	Test	0.2196	[0.2149; 0.2244]
Régression logistique (intercept=0)	Apprentissage	0.2759	[0.2735; 0.2783]
	Test	0.2853	[0.2807; 0.2899]
Régression logistique quadratique (intercept=1)	Apprentissage	0.1861	[0.1832; 0.1890]
	Test	0.2459	[0.2411; 0.2508]
Régression logistique quadratique (intercept=0)	Apprentissage	0.1871	[0.1843; 0.1899]
	Test	0.2407	[0.2360; 0.2454]
Arbre de décision	Apprentissage	0.1958	[0.1856; 0.2019]
	Test	0.2429	[0.2374; 0.2485]

D'après 4, parmi les trois méthodes discriminantes, l'analyse discriminante linéaire permet d'obtenir les meilleurs résultats, suivi de près par le classifieur bayésien et l'analyse discriminante quadratique.
Nous pouvons expliquer cela par le fait que le modèle linéaire comporte moins de paramètres que le modèle quadratique donc il est plus général et plus robuste sur de nouvelles données. En revanche, le classifieur bayésien possède le moins de paramètres et n'est donc pas assez flexible pour ce jeu de données.

La régression logistique (avec l'ajout d'une ordonnée à l'origine) permet d'obtenir le plus faible taux d'erreur. Tandis que la régression logistique (sans l'ajout d'une ordonnée à l'origine) obtient le taux d'erreur le plus fort.
On constate donc que l'ajout de l'ordonnée à l'origine est un paramètre à considérer pour la régression logistique car l'écart de résultat est net.

De manière plus générale, la performance des prédictions obtenues avec toutes nos méthodes est nettement moins effective que pour les données simulées. Nous pouvons donc supposer que les variables décrivant les individus de *pima* ne suivent pas une loi normale multidimensionnelle. Ce qui pourrait expliquer la performance moyenne des modèles faisant cette hypothèse.

2.2.2 Breast cancer Wisconsin

Comme pour le précédent jeu de données, nous appliquons les mêmes méthodes de prédiction (sauf la régression logistique quadratique) avec $N = 100$ répétitions.

TABLE 5 – Calcul du taux d'erreur d'apprentissage ε et de test pour le jeu de données BCW avec plusieurs méthodes (arrondi à 10^{-4})

Méthode	**Données**	$\bar{\varepsilon}$	IC
Analyse discriminante quadratique	Apprentissage	0.0437	[0.0426; 0.0448]
	Test	0.0522	[0.0499; 0.0546]
Analyse discriminante linéaire	Apprentissage	0.0389	[0.0378; 0.0401]
	Test	0.0450	[0.0427; 0.0473]
Classifieur bayésien naïf	Apprentissage	0.0368	[0.0356; 0.0380]
	Test	0.0384	[0.0366; 0.0407]
Régression logistique (intercept=1)	Apprentissage	0.0383	[0.0371; 0.0395]
	Test	0.0465	[0.0443; 0.0487]
Régression logistique (intercept=0)	Apprentissage	0.1444	[0.1424; 0.1464]
	Test	0.1574	[0.1534; 0.1613]
Arbre de décision	Apprentissage	0.0312	[0.0292; 0.0332]
	Test	0.0579	[0.0548; 0.0609]

D'après la table 5, le classifieur bayésien naïf permet d'obtenir les meilleurs résultats, alors que la régression logistique (sans l'ajout de l'ordonnée à l'origine) obtient une fois encore le taux d'erreur le plus fort.

L'écart de performance entre l'ajout ou non de l'ordonnée à l'origine est très marqué pour la régression logistique. Cela confirme donc que c'est un paramètre important à prendre en compte.

Parmi les trois méthodes discriminantes, le taux d'erreur augmente quand le nombre de paramètre est plus important (le modèle linéaire obtient de meilleurs résultats que le modèle quadratique qui a plus de paramètres.)

De manière plus générale, la performance des prédictions obtenues avec toutes nos méthodes est très effective. Nous pouvons donc supposer que les variables décrivant les individus de *pima* suivent une loi normale multidimensionnelle. Ce qui explique la bonne performance des modèles faisant cette hypothèse.

Enfin, il s'avère qu'un modèle générale et donc avec peu de paramètres est le plus adapté à ce jeu de données.

3 Challenge : Spam

Ce jeu de données comporte 58 variables pour 4601 individus. On a exclu la première colonne, car cela correspond à l'index qui est inutile pour notre analyse.

3.1 L'arbre de décision

Nous avons tout d'abord tenté la méthode de l'arbre de décision, car cela nous permet d'effectuer une classification générale avec un coût temporel peu important. En plus, cette méthode minimise l'impureté à chaque itération, donc on peut obtenir un modèle convenable sans prétraitement de données.

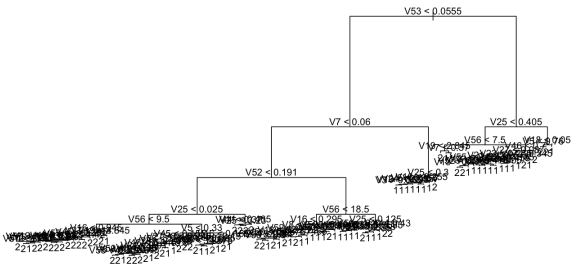

FIGURE 6 – La classification de spam avec l'arbre de décision

On a effectué N=100 répétitions, et on obtient $\bar{\varepsilon} = 0.0867$, avec $sd = 0.0086$ et $IC = [0.0698, 0.1035]$

3.2 L'analyse discriminante

Nous mettons ensuite à l'essai les modèles d'analyse discriminante.

Pour l'analyse discriminante linéaire, nous n'avons effectué aucun pré-traitement de données avec $N = 100$ répétitions.

TABLE 6 – Calcul du taux d'erreur d'apprentissage ε et de test pour le jeu de données *Spam* avec l'analyse discriminante linéaire (arrondi à 10^{-4})

Méthode	**Données**	$\bar{\varepsilon}$	IC
Analyse discriminante linéaire	Apprentissage	0.1132	[0.1123; 0.1141]
	Test	0.1172	[0.1156; 0.1188]

En revanche, pour l'analyse discriminante quadratique et le classifieur bayésien, nous avons rencontré des difficultés. La matrice de variance contient des valeurs propres nulles ou très proches de 0. De ce fait, la factorisation de Cholesky sur la matrice de variance échoue.

```
Error in chol.default(Sigma): the leading minor of order 40 is not positive definite
```

Notons que ce problème apparaît à intermittence : il est parfois possible (selon la séparation aléatoire des données d'apprentissage), d'obtenir une prédiction. Dans ce cas, le taux d'erreur avoisine les 0.160 pour l'analyse discriminante quadratique et 0.180 pour le classifieur bayésien. Par expérience, nous avons observé plus d'échec dans la factorisation de Cholesky que de succès.

Pour résoudre ce problème, nous avons donc 2 solutions :

— Faire un centrage-réduction pour le prétraitement

— Utiliser une méthode de sélection de variable

3.2.1 Solution 1 : Par centrage-réduction

Nous avons encore rencontré une erreur car la capacité mémoire a été atteinte.

```
Error in array(0, c(p, p, g)): negative length vectors are not allowed
Traceback:

1. run_test(spam_sc[, 2:ncol(spam_sc)], nba.app, 1)
2. func(Xapp, zapp)   # at line 32 of file <text>
3. array(0, c(p, p, g))   # at line 110 of file <text>
```

Cela correspond à cette ligne de la fonction nba.app :

```
param$MCov <- array(0, c(p,p,g))
```

lors de la création de l'array multi-dimensionnel de la matrice de covariance. Le problème est identique pour adq.app, le besoin mémoire est trop important pour ce jeu de données.

Nous ne pouvons donc pas utiliser le centrage-réduction.

3.2.2 Solution 2 : Par sélection de variable

Les méthodes de sélection de variables n'est pas au programme de SY09.

En revanche, nous pouvons utiliser l'ACP (analyse en composante principale), comme vu au premier et second TP. C'est une méthode de sélection de variable assez naïve et moins efficace que les méthodes traditionnelles.

En effectuant un centrage-réduction puis l'ACP, nous obtenons les pourcentage d'inertie expliquée suivants : 7

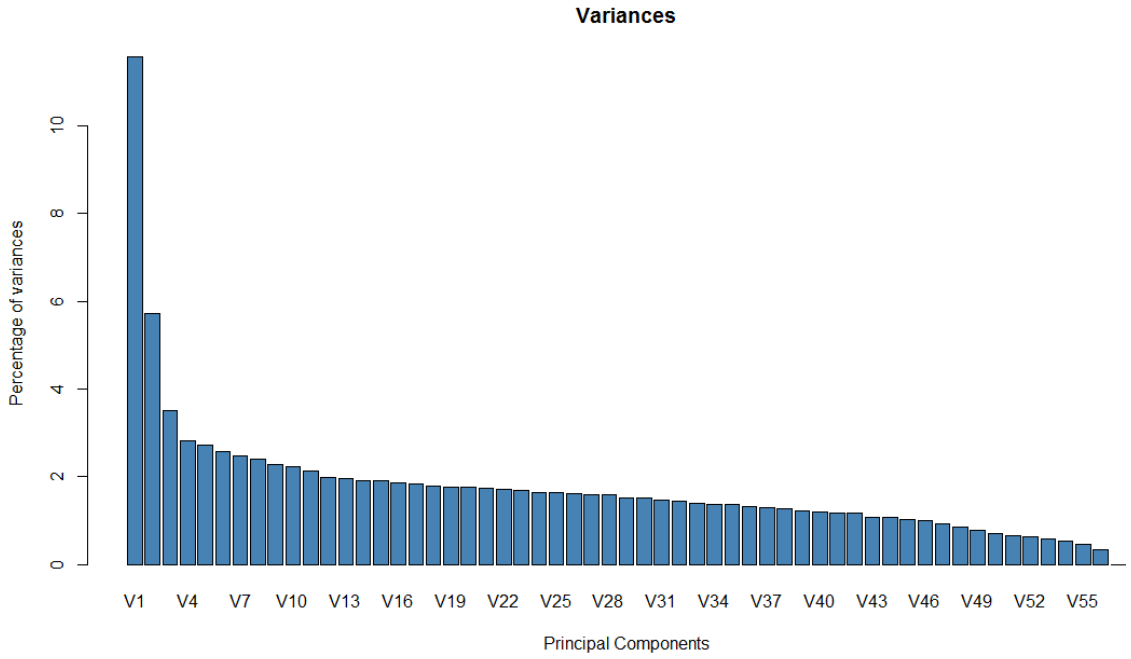

FIGURE 7 – Pourcentage d'inertie expliquée selon les axes factoriels

Les axes factoriels expliquent peu d'information, il faut prendre en compte les 45^{me} premiers axes pour avoir au moins 90% de l'inertie expliquée. (8)

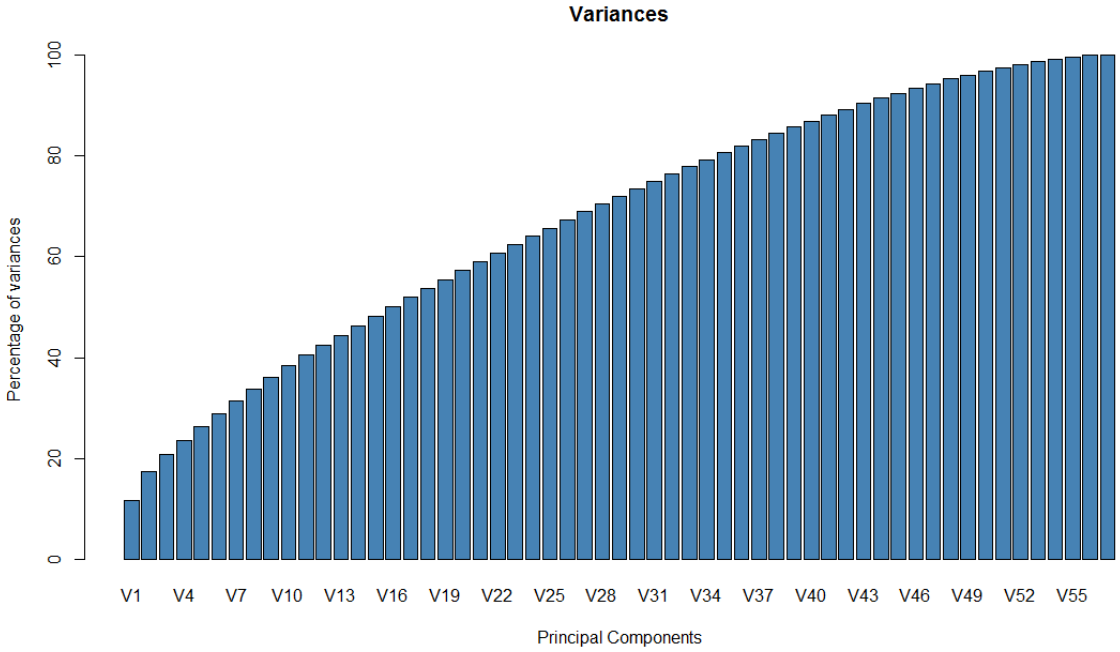

FIGURE 8 – Pourcentage d'inertie expliquée cumulé selon les axes factoriels

Ensuite, en effectuant la représentation des variables (9).

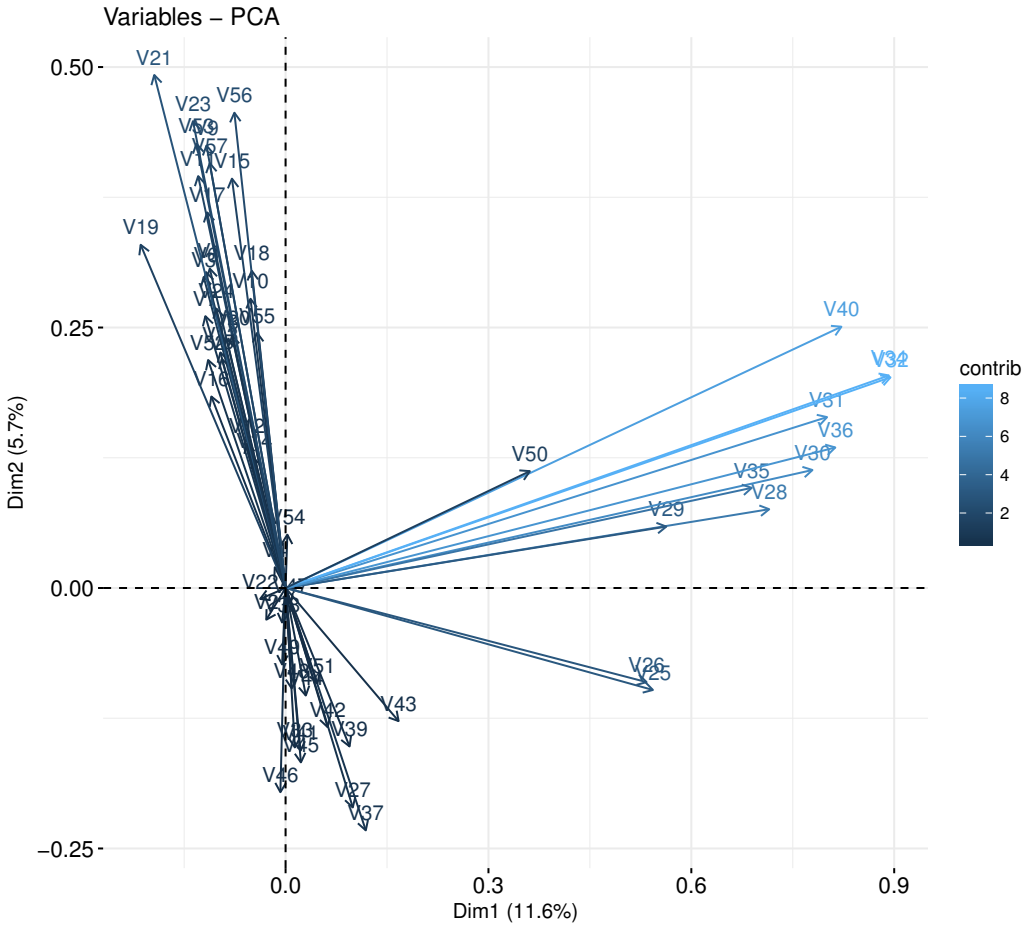

FIGURE 9 – Représentation des variables dans le premier plan factoriel

Nous en déduisons que la qualité de la représentation obtenues par l'ACP n'est pas très bonne car la réduction des variables n'est pas effective.

Nous décidons de tout de même appliquer l'analyse discriminante quadratique et le classifieur bayésien avec l'ACP, en tenant compte de toutes les variables.

Nous obtenons les taux d'erreurs suivants :

TABLE 7 – Calcul du taux d'erreur d'apprentissage ε et de test pour le jeu de données *Spam* avec l'analyse discriminante quadratique et le classifieur bayésien naïf, avec centrage-réduction et ACP (arrondi à 10^{-4})

Méthode	**Données**	$\overline{\varepsilon}$	IC
Analyse discriminante quadratique	Apprentissage	0.1659	[0.1649; 0.1669]
	Test	0.1703	[0.1686; 0.1720]
Classifieur bayésien naïf	Apprentissage	0.2284	[0.2257; 0.2311]
	Test	0.2314	[0.2279; 0.2349]

Nous remarquons que l'estimation de l'erreur pour $N = 100$ répétitions de l'analyse discriminante est proche de la valeur ponctuelle trouvée précédemment sans pré-traitement de données. Ce n'est pas le cas pour le classifieur bayésien où le taux d'erreur avec l'ACP est plus important que la valeur trouvée précédemment.

3.3 Régression logistique

Nous testons ensuite la régression logistique pour ce jeu de données, toujours avec $N = 100$ répétitions.

TABLE 8 – Calcul du taux d'erreur d'apprentissage ε et de test pour le jeu de données *Spam* avec la régression logistique (arrondi à 10^{-4})

Méthode	**Données**	$\overline{\varepsilon}$	IC
Régression logistique (intercept=1)	Apprentissage	0.1097	[0.1088; 0.1106]
	Test	0.1125	[0.1113; 0.1138]
Régression logistique (intercept=0)	Apprentissage	0.1064	[0.1052; 0.1077]
	Test	0.1111	[0.1094; 0.1129]

Notons qu'il n'a pas été possible d'effectuer la régression logistique quadratique car nous sommes tombés sur ce problème :

```
Error in solve.default(tXap% * %MatW% * %Xapp): Lapack routine dgesv: system is
    exactly singular: U[254,254] = 0
Traceback:

1. run_test_logit_quadra(spam[, 2:ncol(spam)], log.app, 100, 1)
2. func(Xapp, zapp, orig, 1e-06)   # at line 90 of file <text>
3. solve(tXap% * %MatW% * %Xapp)   # at line 32 of file <text>
4. solve.default(tXap% * %MatW% * %Xapp)
```

En d'autre terme, à un moment donné, nous obtenons ce résultat :

$$det(X^T W_{(q)} X) = 0$$

Par définition, il n'est donc pas possible d'inverser cette matrice. Or, on rappelle qu'il est nécessaire de calculer $(X^T W_{(q)} X)^{-1}$ pour calculer la mise à jour du vecteur de poids $w^{(q)}$ à l'itération $q + 1$.

Conclusion sur les classifieurs pour *Spam*

Parmi nos différents classifieur, le meilleur est nettement l'arbre de décision. Derrière se place la régression logistique (avec ou sans ajout de l'origine), l'analyse discriminante linéaire puis encore après, l'analyse discriminante quadratique et enfin le classifieur bayésien naïf.

Nous pouvons donc supposer que les données *spam* ne sont pas compatibles avec des modèles ayant beaucoup de paramètres (analyse discriminante quadratique) ou trop peu (classifieur bayésien naïf) et que les variables ne semblent pas suivre pas une loi normale multivariée.

Conclusion

Cette deuxième partie sur l'apprentissage supervisé nous a permis de mettre en pratique de nombreuses méthodes de discriminations sur des jeux de données simulés et réels.
Nous avons donc pu être confronté aux difficultés de trouver le modèle le plus adapté au jeu de données. En particulier, nos différents résultats souligne l'importance de bien comprendre les hypothèses faites par chaque méthode ainsi que d'avoir un bon compromis entre robustesse et flexibilité. Il faut mettre ce dernier en rapport avec la quantité de données disponible.
Enfin, il est essentiel d'évaluer les performances des méthodes utilisées car nous pouvons alors écarter les modèles prédictifs qui ne sont pas adéquats.

Références

[1] Principal component analysis in R : prcomp() vs. princomp() - R software and data mining

[2] Memory limits in data table: negative length vectors are not allowed

A Analyse discriminante quadratique : adq.app

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


  param <- NULL
  param$MCov <- array(0, c(p,p,g))# g tableaux de matrices de covariance de dim : p x p :
                                  # on a donc 1 tableau de matrice de covariance par classe
  param$mean <- array(0, c(g,p))
  param$prop <- rep(0, g)

  mean_vector<- rep(0,p)


  for (k in 1:g)# calculer parametre pour chaque classe
  {

    indk <- which(zapp==k)# index des individus de la classe k

    class_k<- as.matrix(0,nrow=length(indk), ncol=p)

    Xindiv_k <- Xapp[indk,]

    for(i in 1:p)# calculer moyenne de chaque var pour les individus de la classe k
    {
```

```
      class_k <- Xapp[indk,];
      mean_vector[i] <-mean(class_k[,i])

    }

    mean_matrix_k< matrix(mean_vector,ncol=p,nrow=length(indk),byrow=TRUE) # matrice
        p*nombre d'individu de la classe k
    # replication en ligne de mean_vector

    covar <- t(as.matrix(Xindiv_k-mean_matrix_k))*(as.matrix(Xindiv_k-mean_matrix_k)
        )/length(indk)
    covar_corr <- covar*(length(indk)/(length(indk)-1))

    param$MCov[,,k] <- covar_corr
    param$mean[k,] <- mean_vector;
    param$prop[k] <- length(indk)/n;
  }

  param #renvoie les parametres du modele
}
```

B Analyse discriminante linéaire : adl.app

```
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)

  mean_vector<- rep(0,p)

  for (k in 1:g)
  {
    indk <- which(zapp==k)

    class_k<- as.matrix(0,nrow=length(indk), ncol=p)

    Xindiv_k <- Xapp[indk,]

    for(i in 1:p)# calculer moyenne de chaque var pour les individus de la classe k
    {
      class_k <- Xapp[indk,];
      mean_vector[i] <-mean(class_k[,i])

    }

    mean_matrix_k<-matrix(mean_vector,ncol=p,nrow=length(indk),byrow=TRUE) # matrice
        p*nombre d'individu de la classe k

    V_k <- t(as.matrix(Xindiv_k-mean_matrix_k))*(as.matrix(Xindiv_k-mean_matrix_k))/
        length(indk)

    n_k <- length(indk)

    V_k_corr <- (n_k*V_k)/(n_k-1) # Variance de classe corrigee

    MCov <- MCov+ V_k_corr*(n_k-1)
    param$mean[k,] <- mean_vector
```

```
    param$prop[k] <- length(indk)/n
  }# Fin pour chaque classe

  MCov <- MCov/(n-g)


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

  param
} #adl.app
```

C Classifieur bayésien naïf : nba.app

```
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)

  mean_vector<- rep(0,p)

  for (k in 1:g)
  {
    indk <- which(zapp==k)

    class_k<- as.matrix(0,nrow=length(indk), ncol=p)

    Xindiv_k <- Xapp[indk,]

    for(i in 1:p)# calculer moyenne de chaque var pour les individus de la classe k
    {
      class_k <- Xapp[indk,];
      mean_vector[i] <-mean(class_k[,i])

    }

    mean_matrix_k<-matrix(mean_vector,ncol=p,nrow=length(indk),byrow=TRUE) # matrice
        p*nombre d'individu de la classe k
    # replication en ligne de mean_vector

    covar <- t(as.matrix(Xindiv_k-mean_matrix_k))*(as.matrix(Xindiv_k-mean_matrix_k)
        )/length(indk)



    param$MCov[,,k] <- diag(covar)* diag(1,p)
    param$mean[k,] <- mean_vector;
    param$prop[k] <- length(indk)/n;
  }

  param
}
```

D Prédiction du modèle d'analyse discriminante : ad.val

```
ad.val <- function(param, Xtst) # P(w_k|x)=f_k(x)/f(x) * pi_k
{
  n <- dim(Xtst)[1]
  p <- dim(Xtst)[2]
  g <- length(param$prop)

  out <- NULL

  prob <- matrix(0, nrow=n, ncol=g)
  f_x <- matrix(0, nrow=n, ncol=1)
  f_k <- matrix(0, nrow=n, ncol=1)

  for (k in 1:g) # pour chaque classe, calculer f_k(x) puis construire f(x)=sum(pi_k
       *f_k)
  {
    f_k <- mvdnorm(Xtst, param$mean[k,], param$MCov[,,k]) # f_k(x)

    f_x= f_x+param$prop[k]*f_k # f(x)= somme (pi_k* f_k(x))

    prob[,k] <- f_k


  };

  for (k in 1:g) # calcule la probabilite a posteriori : f_k/f_x
  {

    prob[,k] <- (prob[,k]*param$prop[k]) /f_x

  };
  pred <- max.col(prob)

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

  out
}
```

E Classifieur logistique : log.app

```
postprob <- function(beta, X)
{
  X <- as.matrix(X)
  n<-nrow(X)
  beta<-as.matrix(beta)
  Mbeta<-matrix(rep(t(beta),n),nrow=n,byrow=T)
  tmp<-exp(rowSums(Mbeta*X))
  prob <- tmp/(1+tmp)# calcule la fonction logit
  prob
}

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 # stocker l'ancien valeur dans bold

    prob <- postprob(beta, Xapp)
    MatW <- diag(prob,n) %*% diag((1-prob),n)

    beta <- bold+solve(tXap%*%MatW%*%Xapp)%*%tXap%*%(targ-prob) #mise a jour de
        la valeur de beta a chaque iteration, l'algorithme de Newton Raphson

    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
}
```

F Classifieur logistique : 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 <- postprob(beta,Xtst)
  pred <- max.col(cbind(prob,1-prob)) #comparer la probabilite de chaque classe et
      choisir le maximum

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

  return(out)
}
```

G Construire la matrice quadratique

```
quadratique<-function(X){
  X<-as.matrix(X)
  i=1;
  j=1;
  n<-dim(X)[1]
  p<-dim(X)[2]
  c<-p+1
  Xnew<-matrix(0,ncol=p*p/2+3*p/2,nrow=n)
  Xnew[,1:p]<-X
  for(i in 1:p){
    for(j in i:p){
      Xnew[,c]<-X[,i]*X[,j] #construire le terme quadratique
      c<-c+1
    }
  }
  Xnew
}
```

H Arbre de décision

```
d<-read.csv("~/sy09/tp4/donnees/Synth3-1000.csv")############
X<-d[,1:2]
z<-d[,3]
for(i in 1:20){
  donn.sep <- separ1(X, z);
  Xapp <- donn.sep$Xapp;
  zapp <- donn.sep$zapp;
  Xtst <- donn.sep$Xtst;
  ztst <- donn.sep$ztst;
  zpred<-rep(0,length(ztst));
  zapp <- as.factor(zapp)
  #joindre tous les colnames pour creer le formule string
  formule<-paste(names(d)[1:2],collapse='+')
  formule<-paste(c("zapp",formula),collapse='~')
  #construire l'arbre
  origin_tree = tree(formule, data=Xapp, control=tree.control(nobs=nrow(Xapp),
      mindev = 0.0001))
  best_k = cv.tree(origin_tree)$size[which.min(cv.tree(origin_tree)$dev)]
  #prune
  pruned_tree = prune.misclass(origin_tree, best=best_k)

  prediction = predict(pruned_tree, Xtst)
  zpred = max.col(prediction)erreur[i]<-validation(zpred,ztst);
}
m<-mean(erreur);m;
s<-sd(erreur);s;
q<-qnorm(0.025)
m+s*q
m-s*q
```