Compte rendu TP 4 SY09

ARTCHOUNIN Daniel / VALLOIS Célestin

23 juin 2016

Résumé

Dans le cadre du quatrième sujet des séances de Travaux Pratiques (TP) de l'Unité de Valeur (UV) SY09 enseignée à l'Université de Technologie de Compiègne (UTC), nous avons principalement effectué des classifications sur plusieurs jeux de données binaires.

Tout d'abord, nous avons implémenté trois modèles d'analyse discriminante dans le cas binaire. Nous avons également programmé le modèle de régression logistique classique ainsi que le modèle de régression logistique quadratique.

Ensuite, nous avons testé nos fonctions sur des données simulées (`Synth1-1000`, `Synth2-1000` et `Synth3-1000`), puis, sur des données réelles (`Pima.csv` et `bcw.csv`).

Enfin, nous nous sommes intéressés à un problème de détection de spams à partir d'indicateurs calculés sur des messages électroniques.

Le dossier `code_source` associé au présent rapport et contenant le code source `R` écrit afin de répondre aux différentes questions présentes dans le sujet s'organise ainsi :

— `ex1_1.R` : le script `R` associé à la sous-section 1
— `ex1_2.R` : le script `R` associé à la sous-section 1
— `ex2_1.R` : le script `R` associé à la sous-section 2.1
— `ex2_2.R` : le script `R` associé à la sous-sous-section 2.2.1
— `ex2_3.R` : le script `R` associé à la sous-sous-section 2.2.2
— `ex31.R` : le script `R` associé au début de la section 3
— `ex32.R` : le script `R` associé à la fin de la section 3

1 Programmation

Cf. annexes A pour retrouver les fonctions importantes demandées durant ce TP.

Sachant que `R` est plus performant sur des opérations matricielles, nous avons tenté d'éviter d'utiliser des boucles et de privilégier ce type d'opérations dans notre implémentation.

On notera la nécessité de calculer l'inverse d'une matrice à chaque itération de l'algorithme de Newton-Raphson. Or, il arrive que ce ne soit pas le cas. Afin de faire face à ce problème, nous avons décider d'utiliser le pseudo-inverse de **Moore-Penrose** à l'aide la fonction **ginv** du package `MASS` de `R`.

Nous ne détaillerons pas plus que ça les différentes méthodes dans cette partie afin d'éviter un rapport trop lourd. Les différentes explications données en cours permettent de bien comprendre leur principe et le code disponible en annexe, leur implémentation.

2 Application

Notre but est de trouver le modèle ou les modèles le(s) "plus" adapté(s) parmi l'Analyse Discriminante Quadratique, l'Analyse Discriminante Linéaire, le classifieur bayésien naïf, les modèles de régression logistique et régression logistique quadratique ainsi que finalement les arbre binaires pour chaque jeu de données à étudier.

Dans la suite du rapport, nous adopterons la convention suivante : `ADQ` désigne l'analyse discriminante quadratique, `ADL` désigne l'analyse discriminante linéaire, `NBA` le classifieur bayésien naïf, `RLCSI` la régression logistique classique sans intercept, `RLCAI` celle avec intercept, `RLQ` la régression logistique quadratique, et, `BT`,

les arbres de décision.

Enfin, comme dans tous les jeux de données suivants, il est particulièrement logique que l'on obtienne de moins bons résultats avec la Régression Logistique Classique sans intercept qu'avec la Régression Logistique Classique sans intercept. Effectivement, avec intercept, la frontière est un hyperplan ne passant pas nécessairement par l'origine séparant les deux classes. Cependant, sans intercept, cette dernière passe nécessairement par l'origine et donc cela est plus contraignant. Ceci explique donc les moins bons résultats.

Enfin, concernant la régression Logistique Quadratique, si la taille du jeux de données est "suffisamment" grand, les résultats de la classification seront meilleures. Toutefois, si ce n'est pas le cas, il peut arriver que les résultats obtenus à l'issue d'une régression Logistique Classique soit meilleur vu qu'il y aura moins de paramètres à apprendre.

2.1 Test sur données simulées

Dans cette partie, nous allons estimé le meilleur modèle pour les jeux de données `synth1_1000`, `synth2_1000` et `synth3_1000` qui sont issus d'une loi bivariée normale et crée aléatoirement.

Il faut savoir que ces jeux de données sont représentables dans le plan, nous allons pouvoir émettre des hypothèses concernant les modèles les plus adaptés tout en les additionnant à une représentation visuel.

Pour ce faire, nous utiliserons le même protocole expérimental que dans le TP précédent. On répétera ce dernier $N = 20$ fois pour chaque jeu de données :

1. séparation du jeu de données en un ensemble d'apprentissage et un ensemble de test ;
2. apprentissage du modèle sur l'ensemble d'apprentissage,
3. classification des données de test et calcul du taux d'erreur associé.

Enfin, en fonction des résultats obtenues, nous confirmerons ou réfuterons nos hypothèses émises sur l'étude des jeux de données. Pour le premier jeu de données, nous représenterons les frontières de décisions dans le plan des différents modèle ainsi que l'arbre binaire afin de comprendre leurs fonctionnements de manière visuel. Ensuite, nous ne représenterons que les modèles que nous jugeons comme les "plus" adaptés.

On rappellera que le choix du modèle dépend des données comme nous allons le voir et que nous savons que ces données sont issus de lois gaussiennes. Le premier effectif a été généré à partir d'une loi binomiale de paramètre n avec $\pi_1 = 0.5$ (donc aucune classe n'est logiquement sous représentée). On peut à partir de cette information prévoir que l'analyse discriminante (ADX), du moins un des modèles de l'analyse discriminante sera certainement le meilleur modèle.

2.1.1 Synth1-1000

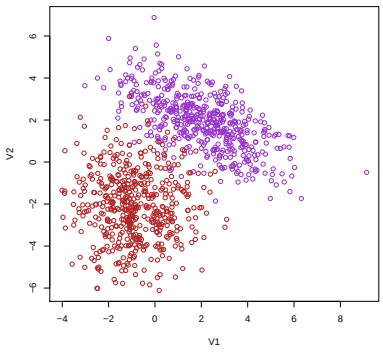

FIGURE 1 – Jeu de données `Synth1-1000` afin d'émettre des hypothèses

Pour le premier jeu de données observable ci-dessus, l'**ADQ** semble bien s'y adapter pour les raisons évoquer précédemment, les différents conditions sont respectés. Cependant, l'**ADL** ou bien le Classifieur Bayésien Naïf semblent moins bien s'adapter à la répartition des données. Effectivement, nous savons que a frontière de décision à associée à l'`ADL` est linéaire. Dans la figure 1, nous remarquons que les données des deux classes ne peuvent pas bien se scinder selon une droite dans le plan. De plus, nous savons que la frontière de décision associée au `NBA` est plus adaptée si les dispersions associées aux données des différentes classes se font suivant les axes. Concernant les arbres binaires, il est plus

complexe de conclure concernant l'adéquation du modèle avec le jeu de données. Toutefois, il est très probable que les résultats soient moins bons puisque cela revient à tenter de partition le plan sous forme de rectangles. Or, il semble que cela ne soit pas adapté à notre jeu de données. Si l'on souhaite une bonne précision pour ce jeu de données, il faudra une complexité très grande afin de bien représenter la limite des classes. Or, nous ne souhaitons pas avoir une telle complexité. Finalement, la régression logistique, qu'elle soit classique ou quadratique devrait être mois bien adaptée que l'analyse discriminante car le jeu de donnée possède de très bonne caractéristique pour le modèle de l'analyse discriminante et il respecte ses contraintes.

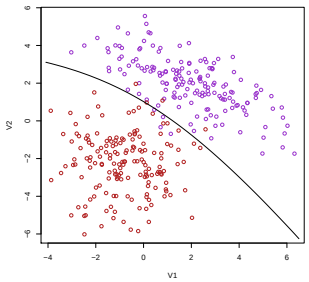

FIGURE 2 – Analyse discriminante quadratique pour le jeu de données `Synth1-1000`

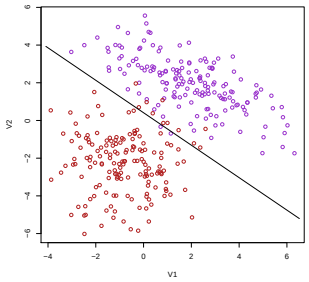

FIGURE 3 – Analyse discriminante linéaire pour le jeu de données `Synth1-1000`

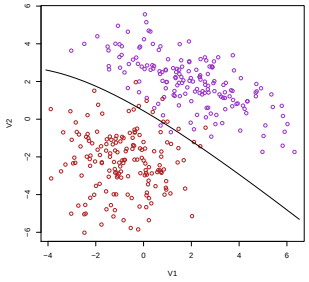

FIGURE 4 – classifieur bayésien naïf pour le jeu de données `Synth1-1000`

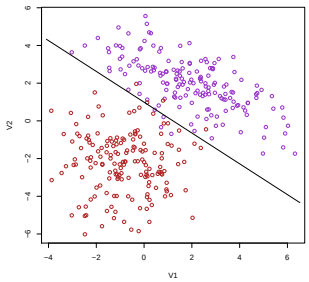

FIGURE 5 – Régression Logistique classique avec intercept pour le jeu de données `Synth1-1000`

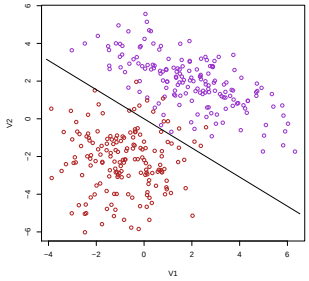

FIGURE 6 – Régression Logistique classique sans intercept pour le jeu de données `Synth1-1000`

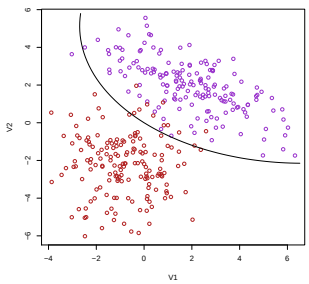

FIGURE 7 – Régression Logistique quadratique pour le jeu de données `Synth1-1000`

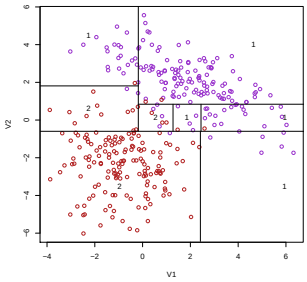

FIGURE 8 – Arbre binaire pour le jeu de données `Synth1-1000`

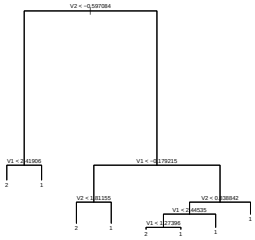

FIGURE 9 – Arbre binaire obtenu pour le jeu de données `Synth1-1000`

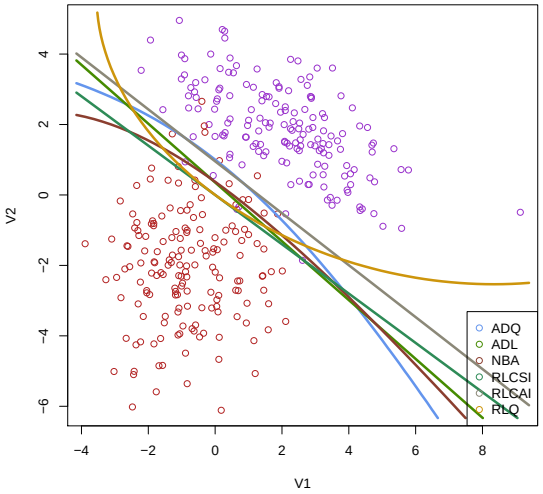

FIGURE 10 – Représentation des frontières de décisions des modèles pour `Synth1-1000`

Cette dernière représentation permet d'observer les frontières de décisions pour tous les modèles sauf l'arbre binaire. Nous allons maintenant montrer les différents résultats obtenus ainsi que leurs erreurs résumé dans un tableau :

Synth1-1000	$\widehat{\varepsilon_i}$	IC_{ε_i}
ADQ	2.52 %	[2.16, 2.88]
ADL	3.54 %	[3.14, 3.94]
NBA	3.56 %	[3.17, 3.95]
RLCSI	4.13 %	[3.72, 4.53]
RLCAI	2.56 %	[2.18, 2.95]
RLQ	3.74 %	[3.37, 4.11]
BT	3.90 %	[3.51, 4.30]

Les erreurs semblent nous indiquer que l'**ADQ** est le meilleur modèle pour ce jeu de données *synth1_1000* ! Son point négatif étant sa complexité et le besoin d'un grand nombre de données. Cela n'est pas un problème dans ce cas de figure mais il faut garder à l'esprit ses inconvénients. La **régression logistique avec intercept** fait aussi office de sérieux candidat, les autres modèles présentent des performances acceptables mais très en dessous de celle de l'**ADQ**.

2.1.2 Synth2-1000

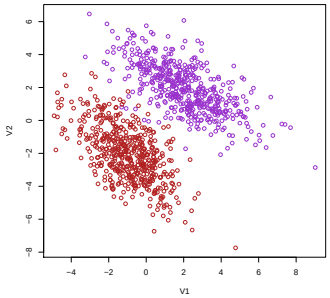

FIGURE 11 – Jeu de données `Synth2-1000` afin d'émettre des hypothèses

Nous émettons de nouvelles hypothèses à partir de ce second jeu de données *synth2_1000*. Dans ce cas, il nous semble que l'ADQ sera toujours aussi efficient mais l'ADL semble être un très bon candidat dû à la distribution des valeurs comme cela peut s'observer sur le graphique, la création d'une frontière linéaire entre les 2 classes semblent aisées car celle-ci ne se mélangent peu et suivent une même orientations. Les hypothèses concernant les autres jeu sont assez proches de celle émise pour le jeu précédent, nous ne les détaillerons pas.

Voici le tableau récapitulatif des erreurs :

Synth2-1000	$\widehat{\varepsilon_i}$	IC_{ε_i}
ADQ	0.97 %	[0.78, 1.16]
ADL	1.06 %	[0.92, 1.21]
NBA	1.60 %	[1.40, 1.80]
RLCSI	1.16 %	[0.91, 1.42]
RLCAI	1.15 %	[0.99, 1.31]
RLQ	1.14 %	[0.98, 1.29]
BT	1.68 %	[1.38, 1.97]

Comme émit en hypothèse, les modèles **ADQ** et **ADL** semble être les plus adapté avec aussi **la régression logistique sans intercept**. L'ADQ possède de légère meilleur performance mais sa complexité nécessite beaucoup plus de données. De plus, l'ADL est plus robuste. Ici, nous avons donc le choix de l'ADQ qui est un peu mieux mais qui dépend vraiment du jeu de données et notamment de sa taille sinon l'ADL, plus robuste, est bon modèle aussi. On peut voir graphiquement que les courbes sont d'ailleurs très similaires.

Voici le graphique avec les frontières de décisions des modèles (hormis l'arbre binaire) pour ce jeu de données :

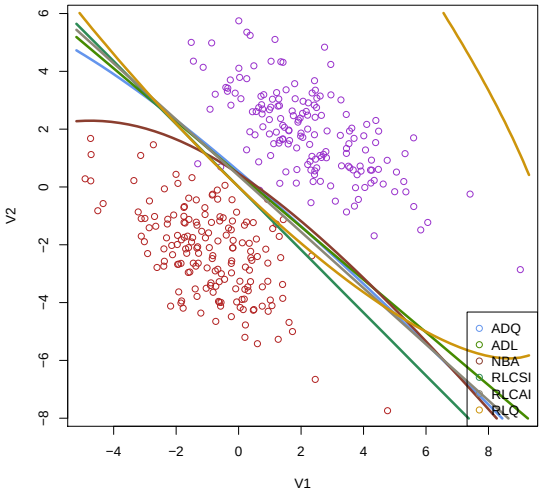

FIGURE 12 – Représentation des frontières de décisions des modèles pour `Synth2-1000`

2.1.3 Synth3-1000

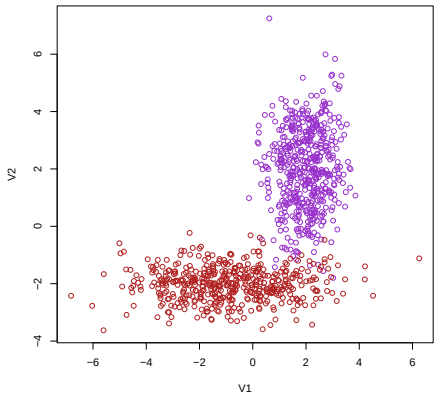

FIGURE 13 – Jeu de données `Synth3-1000` afin d'émettre des hypothèses

Voici notre troisième et dernier jeu de données synthétisée, *synth3_1000*. Au vu de la distribution des 2 classes qui semble chacune suivre un axe (la classe violette suit les ordonnées alors que la rouge suit les abscisses), le classifieur bayésien naïf en plus de l'ADQ nous semble être un choix correct cette fois-ci.

Voici le tableau récapitulatif des erreurs :

`Synth3-1000`	$\widehat{\varepsilon_i}$	IC_{ε_i}
`ADQ`	1.25 %	[1.04, 1.45]
`ADL`	2.43 %	[2.14, 2.72]
`NBA`	1.35 %	[1.14, 1.56]
`RLCSI`	2.55 %	[2.30, 2.81]
`RLCAI`	1.80 %	[1.52, 2.09]
`RLQ`	1.41 %	[1.19, 1.63]
`BT`	1.80 %	[1.58, 2.03]

Cette fois il semblerait que l'**ADQ**, le **NBA** et la **régression logistique quadratique** soit les modèles avec les meilleurs performances. Cependant au vu de la complexité de mettre en place le NBA, son choix final n'est sûrement pas le plus pertinent malgré ses performances. L'ADQ semble une nouvelle fois le plus appropriée. On pourra remarquer éventuellement la taille de l'intervalle de confiance pour l'arbre binaire qui est logiquement plus élevée.

Voici les graphiques illustrant ces résultats (hormis l'arbre binaire) :

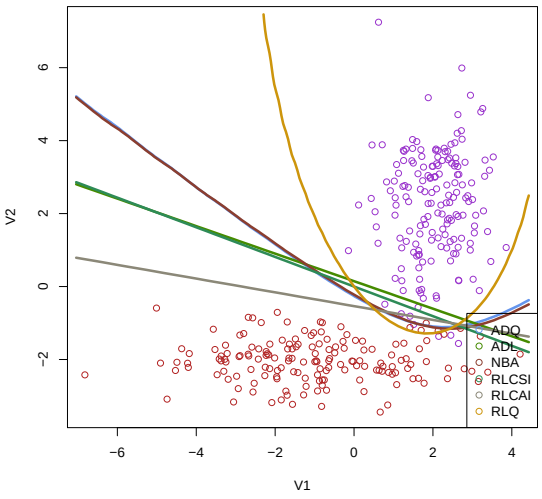

FIGURE 14 – Représentation des frontières de décisions des modèles pour `Synth3-1000`

2.2 Test sur données réelles

2.2.1 Données « Pima »

Dans cette sous-sous-section, nous avons appliqué les trois modèles d'analyse discriminante, les deux modèles de régression logistique et les arbres de décision à la prédiction du diabète chez les individus d'une population d'amérindiens. Les données étaient présentes dans le fichier `Pima.csv` sur le site de l'UV. Ce dernier contient $n = 532$ individus décrits chacun par $p = 7$ variables explicatives.

Afin d'apprendre et pouvoir faire des prédictions à partir de ce dernier, nous avons utiliser le même protocole expérimental que pour les tests sur les données simulées (sous-section 2.1) en répétant cette fois-ci l'expérience $N = 100$ fois. Ainsi, pour chacun des modèles, nous avons calculé les estimations ponctuelles $\widehat{\varepsilon}_i = \bar{\varepsilon}_i$ des taux d'erreurs ε_i de test et les intervalles de confiance bilatéraux IC_{ε_i} pour les mêmes paramètres au niveau de confiance $1 - \alpha$, où $\alpha = 0.05$. Nous noterons que l'approximation sera probablement de bonne qualité puisque $N \geq 30$.

Ci-dessous, figurent les estimations pour les différents modèles.

`Pima.csv`	$\widehat{\varepsilon}_i$ %	IC_{ε_i} %
`ADQ`	23.84	[23.30, 24.37]
`ADL`	21.69	[21.16, 22.23]
`NBA`	23.30	[22.73, 23.87]
`RLCSI`	28.46	[27.96, 28.97]
`RLCAI`	21.44	[20.90, 21.98]
`RLQ`	24.54	[24.04, 25.03]
`BT`	23.88	[23.24, 24.52]

Tout d'abord, nous constatons que les estimations $\widehat{\varepsilon}_i$ des taux d'erreurs de test ε_i sont toutes supérieures à 20%. Cela semble cohérent puisque les taux d'erreur de prédiction du diabète ont généralement ces ordres de grandeur.

Nous constatons que l'estimation du taux d'erreur associée à la `RLCAI` est plus faible que celle associée à la `RLCSI`. Cela est tout à fait cohérent puisque le jeu de données est relativement grand $p = 7$ variables descriptives $n = 532$ individus. Or, nous savons que la `RLCAI` apprend un paramètre de plus que la `RLCSI`. Ainsi, la taille de la population couplée à ce paramètre d'apprentissage supplémentaire permettent d'obtenir de meilleurs résultats.

Cependant, on constate également que l'estimation du taux d'erreur associée à la `RLQ` est supérieure à celle associée à la `RLCAI`. Certes, on apprend plus de paramètres avec la `RLQ`. Toutefois, ce nombre plus élevé de paramètres à apprendre requiert nécessairement un jeu de données plus imposant pour que les paramètres correspondent à l'existant. Ainsi, il semblerait que le jeu de données ne soit pas assez conséquent pour que les résultats obtenus via la `RLQ` soient meilleures que ceux obtenus via la `RLCAI`.

Par ailleurs, on constate également que l'hypothèse que le vecteur caractéristique X suit, conditionnellement à chaque classe ω_k, une loi normale multidimensionnelle d'espérance μ_k et de variance Σ_k n'est pas mauvaise puisque les estimations des taux d'erreur obtenues via les analyses discriminantes sont du même ordre de grandeur que celles obtenues via les régressions logistiques.

De plus, on constate que l'estimation du taux d'erreur associée à l'`ADQ` est supérieure à celle de l'`ADL`. A l'instar de la `RLQ`, cela peut s'expliquer par le fait que le nombre plus élevé de paramètres à apprendre requiert un jeu de données de taille plus imposante pour que les paramètres à apprendre correspondent à l'existant. Ainsi, il semblerait à nouveau que le jeu de données ne soit pas assez conséquent pour que les résultats obtenus via l'`ADQ` soient meilleurs que ceux obtenus via l'`ADL`.

De plus, on constate que l'estimation du taux d'erreur associée au `NBA` est supérieure à celle de l'`ADL`. Il semblerait que l'hypothèse d'indépendance des variables X_j conditionnellement à Z ne soit pas adaptée. Autrement dit, il semblerait qu'il y a corrélation entre certaines des variables. Cela suffit à expliquer les résultats de moins bonne qualité obtenus.

Enfin, l'estimation du taux d'erreur associée au `BT` est plus élevée que celle associée à l'`ADL` et que celle associée à la `RLCAI`. En fonction du contexte et des besoins du client, l'usage de la `BT` peut se révéler être particulièrement utile. Effectivement, cela permet d'attribuer des probabilités a posteriori à chacune des feuilles de l'arbre. A titre d'exemple, cela peut-être utilisée à des fins de prise de décision. On peut dé-

cider de faire effectuer des tests plus onéreux à un patient dont la probabilité d'être diabétique dépasse un certain seuil. De même, l'interprétabilité de cette méthode est un de ses autres avantages. Cette méthode peut également permettre de justifier une prise de décision. La justification est particulièrement simple et compréhensible grâce aux variables explicatives du jeu de données. A titre d'exemple (figure 15), si l'individu a un taux de glucose inférieur à 127.5, il a de fortes chances de ne pas être diabétique. S'il présente un taux de glucose supérieur à 157.5, il a de fortes de chances de l'être.

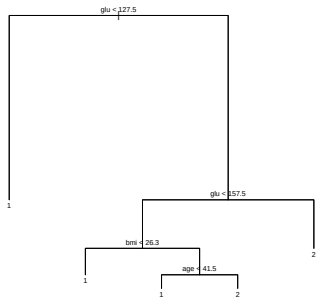

FIGURE 15 – Arbre de décision obtenue à l'issue d'une séparation aléatoire de $\frac{2}{3}$ du jeu de données `Pima.csv`

2.2.2 Données "breast cancer Wisconsin"

Dans cette sous-sous-section, nous nous sommes intéressés à un problème de prédiction du niveau de gravité d'une tumeur à partir de descripteurs physiologiques. Les données figuraient dans le fichier `bcw.csv` sur le site de l'UV. Ce dernier contient $n = 683$ individus décrits chacun par $p = 9$ variables explicatives.

A l'instar de la sous-sous-section 2.2.1, nous avons répété l'expérience (séparation, apprentissage et évaluation des performances) $N = 100$ fois. De plus, nous avons évalué les taux moyens d'erreur de test pour les trois modèles d'analyse discriminante, la régression logistique classique (il était demandé dans l'énoncé de ne pas utiliser la régression logistique quadratique) et les arbres de décision.

Ci-dessous, figurent les estimations pour les différents modèles.

`Pima.csv`	$\widehat{\varepsilon_i}$ %	IC_{ε_i} %
`ADQ`	5.07	[4.83, 5.31]
`ADL`	4.55	[4.34, 4.76]
`NBA`	3.87	[3.67, 4.07]
`RLCSI`	15.77	[15.39, 16.14]
`RLCAI`	4.12	[3.91, 4.34]
`BT`	5.64	[5.35, 5.93]

Tout d'abord, nous constatons que les estimations $\widehat{\varepsilon_i}$ des taux d'erreurs de test ε_i sont toutes supérieurs à 6% sauf celle de `RLCSI`. Le fait que la quasi-totalité des modèles présente des estimations de taux d'erreur du même ordre semble confirmer la cohérence de nos résultats.

Tout d'abord, nous constatons que l'estimation du taux d'erreur associé à la `RLCAI` est beaucoup plus faible que celui associé à la `RLCSI`. A nouveau cela est parfaitement cohérent puisque le jeu de données est relativement grand $p = 9$ variables descriptives $n = 683$ individus. Or, nous savons que la `RLCAI` apprend un paramètre de plus que la `RLCSI`. Le taille de la population couplée à ce paramètre supplémentaire d'apprentissage justifie l'obtention de meilleurs résultats.

Par ailleurs, on constate que l'hypothèse que le vecteur caractéristique X suit, conditionnellement à chaque classe ω_k, une loi normale multidimensionnelle d'espérance μ_k et de variance Σ_k n'est pas mauvaise puisque les estimations des taux d'erreur obtenues via les analyses discriminantes sont du même ordre de grandeur que celle obtenue via la `RLCAI`.

De plus, on constate que l'estimation du taux d'erreur associée à l'`ADQ` est supérieure à celle de l'`ADL`. Cela s'explique à nouveau par le fait que le nombre plus élevé de paramètres à apprendre requiert un jeu de données plus imposant pour que les paramètres appris soient pertinents. Ainsi, il semblerait que le jeu de données ne soit pas assez conséquent pour que les résultats obtenus via l'`ADQ` soient meilleurs que ceux obtenus via l'`ADL`.

De plus, on constate que l'estimation du taux d'erreur associé au `NBA` est inférieure à celle de l'`ADL`. Il semblerait que l'hypothèse d'indépendance des variables X_j conditionnellement à Z soit adaptée. Autrement dit, il semblerait qu'il n'y a pas ou peu de corrélation entre les différentes variables. Cela justifie les meilleurs résultats obtenus.

Enfin, l'estimation du taux d'erreur associée au `BT` est plus élevée que celle associée à la `NBA` et que celle associée à la `RLCAI`. A nouveau, le choix de ce modèle dépendra du contexte d'utilisation de ce dernier. Effectivement, si les clients désirent obtenir des probabilités a posteriori associées à la situation de l'individu, l'usage de ce modèle peut s'avérer pertinent. De même, si chaque proposition d'action doit faire l'objet d'une justification particulièrement précise (cela semble être le cas pour ce jeu de données), ce modèle est à nouveau particulièrement adapté. Cela se démontre assez trivialement à l'aide de la figure 16

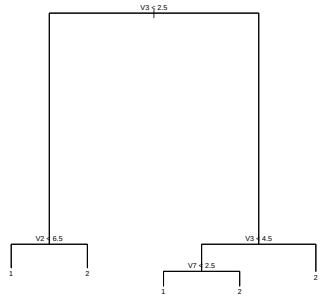

FIGURE 16 – Arbre de décision obtenue à l'issue d'une séparation aléatoire de $\frac{2}{3}$ du jeu de données `bcw.csv`

3 Challenge : données "Spam"

Dans cette section, nous considérons un problème de détection de spams à partir d'indicateurs calculés sur des messages électroniques. Les données figuraient dans le fichier `spam.csv` sur le site de l'UV. Le jeu de données contient $n = 4601$ individus décrits chacun par $p = 57$ variables explicatives.

A l'instar des sous-sous-sections 2.2.1 et 2.2.2, nous avions pour ambition de répéter l'expérience (séparation, apprentissage et évaluation des performances) N 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.

Toutefois, étant donné la taille du jeu de données, il était difficile de faire ces calculs pour $N \geq 30$ et donc d'obtenir de bons intervalles de confiance pour ces estimations. Par ailleurs, parmi les 7 modèles utilisés dans ce présent rapport, il était presque sûr que certains d'entre eux ne seraient pas adaptés à notre jeu de données. En nous basant sur ce constat, nous avons choisi de suivre une stratégie particulière : nous avons décidé de commencer par discriminer les modèles inadaptés. Évidemment, étant donné les coûts en calcul particulièrement importants, il était dommage de travailler directement sur le jeu de données pour procéder à cette sélection.

Ainsi, nous avons décidé d'effectuer une ACP : les pourcentages d'inertie expliquée par les sous-espaces principaux sont consultables dans la figure 17.

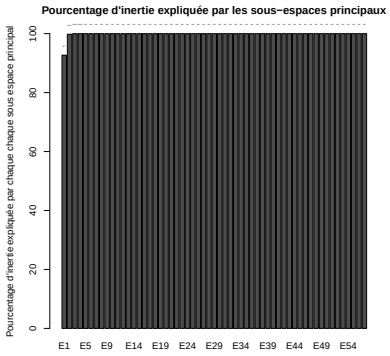

FIGURE 17 – Pourcentages d'inertie expliquée par les sous-espaces principaux (`spam.csv`)

Dans la figure 17, nous constatons que le pourcentage d'inertie expliquée par le sous espace vectoriel E_1 est $92.7 \geq 80$. Toutefois, afin que notre sélection soit pertinente, nous avons décidé de sélectionner les 10 premières composantes principales.

Ensuite, nous avons répété l'expérience (séparation, apprentissage et évaluation des performances) $N = 10$ fois sur ces 10 dernières avant d'évaluer les taux moyens d'erreur de test pour les différents modèles. Ci-dessous, figurent les estimations obtenues.

spam.csv	$\widehat{\varepsilon}_i$ %	IC_{ε_i} %
ADQ	22.45	[20.47, 24.44]
ADL	18.76	[18.43, 19.09]
NBA	32.48	[31.85, 33.11]
RLCSI	15.29	[14.94, 15.64]
RLCAI	14.89	[14.51, 15.26]
RLQ	48.29	[43.18, 53.40]
BT	15.48	[14.96, 16.00]

Tout d'abord, nous constatons remarquablement que la RLQ ne semble pas adaptée pour ce jeu de données. De même, l'ADQ et le NBA sont moins adaptés que l'ADL donc on ne les utilisera pas non plus. De plus, nous constatons que la RLCAI est plus performante que la RLCSI. Certes, cette différence n'est pas imposante. Toutefois, si c'est le cas pour les composantes principales, il y a de fortes chances que l'apprentissage de l'intercept aie des conséquences positives sur le modèle. Par conséquent, on ne retiendra pas la RLCSI. On retiendra également le BT en raison de ses nombreux avantages et de sa faible estimation du taux d'erreur.

Ensuite, nous avons répété la même expérience (séparation, apprentissage et évaluation des performances) $N = 50$ fois sur les $p = 57$ variables explicatives avant d'évaluer les taux moyens d'erreur de test pour les trois modèles : ADL, RLQ et BT. Ci-dessous, figurent les estimations obtenues.

Spam.csv	$\widehat{\varepsilon}_i$ %	IC_{ε_i} %
ADL	11.32	[11.10, 11.54]
RLQ	7.42	[7.23, 7.60]
BT	9.22	[8.97, 9.46]

Il semblerait que l'ADL soit moins adaptée que la RLQ ou que le BT qui présentent tout deux moins de 10 % de taux d'erreur. Concernant le choix du modèle, cela dépendra une fois de plus du contexte. S'il peut arriver qu'il soit nécessaire de justifier les raisons pour lesquelles un mail non spam a été classifié comme spam, il pourrait être intéressant d'avoir recours au BT (figure 18). Cependant, si on souhaite privilégier les performances du classifieur à sélectionner, on aura tendance à ce tourner vers la RLQ.

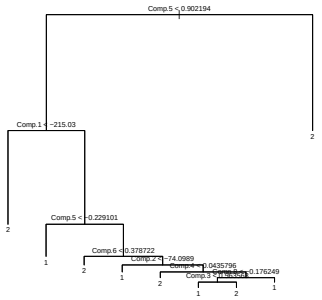

FIGURE 18 – Arbre de décision obtenue à l'issue d'une séparation aléatoire de $\frac{2}{3}$ du jeu de données spam.csv

4 Conclusion

En conclusion, au cours des trois séances de Travaux Pratiques (TP) et de la rédaction du présent rapport, nous avons pu à nouveau prendre conscience de la puissance et des multiples possibilités qui s'offrent à nous en terme de traitement statistique de données avec R. Nous avons également eu l'occasion d'implémenter trois modèles d'analyse discriminante et deux modèles de régression logistique. De plus, nous avons pu tester les performances de ces derniers et des arbres de décision sur différents jeux de données simulées et réelles. Enfin, ce TP nous a permis de mieux comprendre ce qu'était l'apprentissage supervisé.

A Code source Analyse discriminante

A.1 Apprentissage des trois modèles d'analyse discriminante : adq.app, adl.app et nba.app

```
adq.app <- function(Xapp, zapp){
  g <- max(zapp); # The number of classes
  parameters <- list();
  n <- dim(Xapp)[1];
  for (k in 1:g){
    # K class management
    Xk <- Xapp[which( zapp==k ), ]; # We get the indivuals of this class
    nk <- dim(Xk)[1]; # Number of individuals of this class
    pik <- nk/n; # Proportion of individuals in this class
    muk <- apply(Xk, 2, mean); # Estimation of mu for this class
    sigmak <- cov.wt(Xk, method='ML')$cov; # Estimation of Sigma for this class
    # Here, we store the parameters of the k class
    classK <- list()
    classK$class <- k;
    classK$nk <- nk;
    classK$pik <- pik;
    classK$muk <- muk;
    classK$sigmak <- sigmak;
    parameters[[k]] <- classK;
  }
  parameters;
}
```

```
adl.app <- function(Xapp, zapp){
  g <- max(zapp); # The number of classes
  parameters <- list();
  n <- dim(Xapp)[1];
  p <- dim(Xapp)[2];
  sumVk <- matrix(0, nrow=p, ncol=p);
  for (k in 1:g){
    # K class management
    Xk <- Xapp[which( zapp==k ), ]; # We get the indivuals of this class
    nk <- dim(Xk)[1]; # Number of individuals of this classn
    pik <- nk/n; # Proportion of individuals in this class
    muk <- apply(Xk, 2, mean); # Estimation of mu for this class
    sumVk <- sumVk + (nk - 1) * cov.wt(Xk, method='unbiased')$cov;
    # Here, we store the parameters of the k class
    classK <- list()
    classK$class <- k;
    classK$nk <- nk;
    classK$pik <- pik;
    classK$muk <- muk;
    parameters[[k]] <- classK;
  }
  sigmaEstimator <- sumVk/(n-g);
  for (k in 1:g){
    parameters[[k]]$sigmak <- sigmaEstimator;
  }
  parameters;
}
```

```
nba.app <- function(Xapp, zapp){
  g <- max(zapp); # The number of classes
  parameters <- list();
  n <- dim(Xapp)[1];
  for (k in 1:g){
    # K class management
    Xk <- Xapp[which( zapp==k ), ]; # We get the indivuals of this class
    nk <- dim(Xk)[1]; # Number of individuals of this class
    pik <- nk/n; # Proportion of individuals in this class
    muk <- apply(Xk, 2, mean); # Estimation of mu for this class
    sigmak <- diag(diag(cov.wt(Xk, method='ML')$cov)); # Estimation of Sigma for this class
    # Here, we store the parameters of the k class
    classK <- list()
    classK$class <- k;
    classK$nk <- nk;
    classK$pik <- pik;
    classK$muk <- muk;
    classK$sigmak <- sigmak;
    parameters[[k]] <- classK;
  }
  parameters;
}
```

A.2 Calcul des probabilités a posteriori : ad.val

```
ad.val <- function(parameters, Xtst){
  n <- dim(Xtst)[1];
  g <- length(parameters);
  densities <- list();
  sumDensities <- rep(0, n);
  aPosterioriProbabilities <- NULL;
  for(k in 1:g){
    densities[[k]] <- mvdnorm(Xtst, parameters[[k]]$muk, parameters[[k]]$sigmak);
    sumDensities <- sumDensities + parameters[[k]]$pik * densities[[k]];
  }
  for(k in 1:g){
    if(k == 1){
      aPosterioriProbabilities <- as.matrix((parameters[[k]]$pik * densities[[k]])/sumDensities);
    }else{
      aPosterioriProbabilities <- cbind(aPosterioriProbabilities, as.matrix((parameters[[k]]$pik * densities[[k]])/sumDensities));
    }
  }
  predictions <- apply(aPosterioriProbabilities, 1, which.max);
  predictionsList <- list();
```

```
  predictionsList$prob <-
      aPosterioriProbabilities;
  predictionsList$predictions <- predictions;
  predictionsList;
}
```

B Code source Régression logistique

B.1 Apprentissage : log.app

```
log.app  <- function(Xapp, zapp, intr, epsi,
    pseudoInv=FALSE){
  beta <- NULL; # Estimation of parameters
  niter <- 0; # Number of iterations
  logL <- NULL; # Log vraisemblance
  Xapp <- as.matrix(Xapp);
  n <- dim(Xapp)[1]; # Number of individuals
  p <- dim(Xapp)[2]; # Number of
      caracteristics
  t <- cleanZ(zapp);
  minusH <- NULL;
  betaNew <- matrix(0, p, 1);
  betaOld <- matrix(1, p, 1);
  if(intr){ # We should add an intercept
    Xapp <-cbind(matrix(1, n, 1), Xapp);
    betaOld <- matrix(1, p+1, 1);
    betaNew <- matrix(0, p+1, 1);
  }
  while(norm(betaNew - betaOld, type="2") >
      epsi){
    niter <- niter + 1;
    betaOld <- betaNew;
    minusH <- t(Xapp) %*% diag( diag(
        pOmega1X(Xapp, betaOld) %*% t(
        pOmega2X(Xapp, betaOld)) ) )  %*%
        Xapp;
    if(pseudoInv){
      betaNew <- betaOld + ginv(minusH) %*% t
          (Xapp) %*% (t - pOmega1X(Xapp,
          betaOld));
    }else{
      betaNew <- betaOld + solve(minusH) %*%
          t(Xapp) %*% (t - pOmega1X(Xapp,
          betaOld));
    }
  }
  results <- list()
  results$beta <- betaNew;
  results$niter <- niter;
  results$logL <- t(Xapp) %*% (t - pOmega1X(
      Xapp, betaNew));
  results;
}
```

B.2 Classement : log.val

```
log.val <- function(beta, Xtst){
  Xtst <- as.matrix(Xtst);
  n <- dim(Xtst)[1];
  p <- dim(Xtst)[2];
  pBeta <- dim(beta)[1];
  if(p != pBeta){
    Xtst <-cbind(matrix(1, n, 1), Xtst);
  }
  results <- list();
  prob <- post.pr(Xtst, beta);
  predictions <- apply(prob, 1, which.max);
  results$prob <- prob;
  results$predictions <- predictions;
  results;
}
```

B.3 Régression logistique quadratique : log.quad.app et log.quad.val

```
log.quad.app  <- function(Xapp, zapp, epsi,
    pseudoInv=FALSE){
  Xapp <- as.matrix(Xapp);
  Xapp <- computeQuadraticModel(Xapp);
  log.app(Xapp, zapp, FALSE, epsi, pseudoInv)
      ;
}

computeQuadraticModel <- function(X){
  n <- dim(X)[1];
  p <- dim(X)[2];
  XNew <- matrix(0, n, (p*(p+1))/2);
  iterator <- 0;
  for(i in 1:(p-1)){
    for(j in (i+1):p){
      iterator <- iterator + 1;
      XNew[, iterator] <- X[, i] * X[, j];
    }
  }
  for(i in 1:p){
    iterator <- iterator + 1;
    XNew[, iterator] <- X[, i] * X[, i];
  }
  cbind(X, XNew);
}
```