Université de Technologie de Compiègne
SY09:Analyse des données et Data-Mining , P17

Rendu TP4

Zineb Slam, Oumaima Talouka

26 juin 2017

Résumé

Dans de TP4 nous allons étudier 5 techniques de discriminante : l'analyse discriminante linéaire, l'analyse discriminante quadratique, le bayésien naive, la régression logistique et la régression logistique quadratique.

1 Programmation

1.1 Analyse discriminante

Pour la programmation des 3 modèles de l'analyse discriminante, il nous est demandé de compléter 4 fonctions pré-implémentées. Les fonctions `adq.app`, `adl.app` et `nba.app` permettent l'apprentissage des 3 modèles, respectivement l'analyse discriminante quadratique, linéaire et enfin le classifieur bayésien naif. Elles doivent retourner les paramètres des modèles : proportions, moyennes et matrices de covariance des deux classes. Pour cela nous nous sommes appuyés sur les formules suivantes pour chacun des paramètres dans le cas de l'ADQ :

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

$$\widehat{\mu_k} = \frac{1}{n_k} \sum_{i=1}^{n} z_{ik} xi$$

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

En ce qui concerne l'ADL, les proportions et le vecteur des moyennes suivent les mêmes formules. Ce modèle suppose en plus l'hypothèse d'homoscédasticité,

donc d'égalité des matrice de variances pour les classes. Celle-ci s'exprime comme suit :

$$\widehat{\Sigma} = \frac{1}{n}\sum_{k=1}^{g} n_k \widehat{\Sigma}$$

Il suffit de sommer les matrices de covariances obtenues par l'ADQ multipliées par les proportions de leurs classes respectives.

Ensuite, pour le modèle du classifieur bayésien naif, nous supposons l'indépendance des variables conditionnellement a la variable a expliquer Zapp. Donc les formules des proportions et le vecteur des moyennes restent les mêmes, les matrices des covariances se résument aux termes diagonaux, d'ou la multiplication des matrices de covariance obtenues par l'ADQ par la matrice identité Ce qui donne la formule :

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

Enfin, la 4ème fonction a compléter concerne le calcul des propriétés a posteriori et effectue un classement en fonction de ces probabilités `ad.val`. Pour le calcul des probabilités a posteriori, nous récupérons les proportions de chaque classe que nous multiplions par la densité a priori de la classe en question calculée grâce a la fonction donnée mvdnorm (en lui passant la moyenne et la matrice de la covariance de la classe). Puis nous retournons la probabilité a posteriori finale en divisant le ce produit par la somme de tous les produits effectues représentant la densité de mélange grâce a `rowSums` sur R.

1.2 Regression logistique

Pour la programmation de la régression logistique, il nous est propose de compléter deux fonctions `log.app` pour l'apprentissage selon la méthode de Newton-Raphson et `log.val` pour la validation sur un ensemble de données La fonction d'apprentissage doit retourner la matrice beta correspondant à l'estimateur du maximum de vraisemblance $\widehat{\beta}$ des paramètres, le nombre `niter` d'itérations effectuées par l'algorithme de Newton-Raphson, et la valeur $\log L$ de la vraisemblance à l'optimum. Pour cela, nous utilisons les formules suivantes respectivement aux paramètres retournes par la fonction a compléter et de manière itérative la formule (1) a l'aide du calcul des formules détaillées de la matrice hessienne et le gradient de la log-vraissemblance calculés en $w^{(q)}$ jusqu'à convergence (nous rappelons q'un vecteur de poids initial $w^{(0)}$ est choisi avant le boucle) :

$$w^{(q+1)} = w^{(q)} - H_{(q)}^{-1} * \frac{\partial \log L}{\partial w}(w^{(q)}) \tag{1}$$

$$H_{(q)} = -X^T W_{(q)} X$$

$$\frac{\partial \log L}{\partial w}(w^{(q)}) = X^T (t - p^{(q)})$$

Nous arrêtons la séquence d'itérations lorsque la nouvelle estimation $w^{(q+1)}$ et $w^{(q)}$ est inférieur au seuil fixe ϵ comme variable d'entrée de la fonction `epsi`.

Nous avons aussi complété la fonction `post.prob` qui calcul les probabilités a posteriori en fonction du paramètre beta retourne par la fonction d'apprentissage. Nous utilisons pour cela les formules suivantes :

$$P(w_1|x) = p(x, w) = \frac{\exp(w^t x)}{1 + \exp(w^t x)}$$

$$P(w_2|x) = 1 - p(x, w)$$

Cette dernière nous aida dans l'implémentation de la fonction de validation `log.val`. Nous l'utilisons pour récupérer les fonctions de probabilités et nous utilisons `cbind` de R afin de retourner la matrice des probabilités a posteriori des deux classes dans le cas binaire de la régression logistique correspondant a `prob` et `1-prob` et le vecteur des classements associes.

Régression logistique quadratique Pour pouvoir appliquer ce modèle, il faudra adapter l'ensemble des données d'entrée `Xapp` a la méthode décrite dans l'énoncé Pour cela nous procédons comme suit :

```
1    Xapp = as.matrix(Xapp)
2    Xapp = cbind(Xapp, Xapp[,1]*Xapp[,2], Xapp[,1]*Xapp[,1],
         Xapp[,2]*Xapp[,2])
```

2 Application

2.1 Test sur données simulées

Pour séparer nos données en ensemble d'apprentissage et de test nous faisons de la sorte :

```
1 train   = sample(1:n, round(2*n/3))
2 Xapp    = X[train, ]
3 zapp    = z[train]
4 Xtst  = X[-train, ]
5 ztst  = z[-train]
```

2.1.1 Synth1-1000

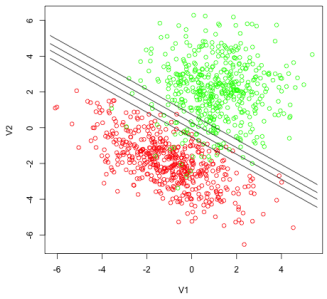
Nous avons représenté les résultats de calcul de l'erreur de test dans les données Synth1-1000.

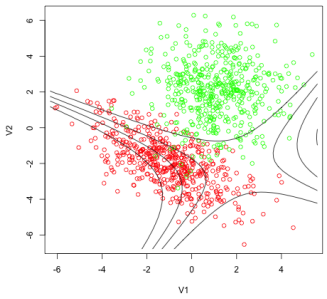
LDA	QDA	Naive Bayes	Log Reg	Quad Log Reg
4.62%	3.60%	4.17%	3.75%	3.63%

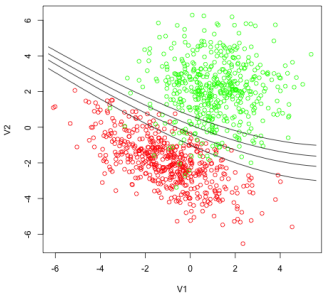
Nous retrouvons les taux d'erreur les plus bas pour l'ADQ et les régressions logistiques. Nous en déduisons que les données suivent bien des distributions

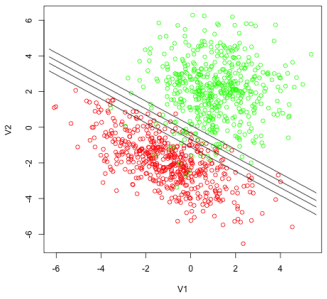
normales et qu'il est fort probable que les classes ont des matrices de covariances différentes (l'hypothèse d'homoscedasticite n'est pas vérifiée).

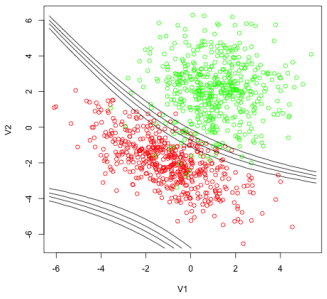
Les graphes sont représentés ci-dessous avec les frontières de décisions.





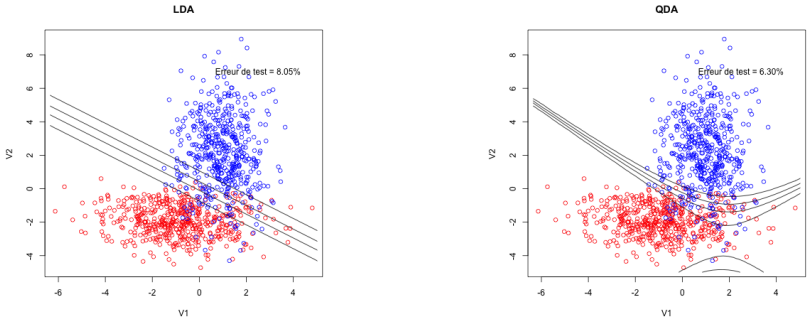


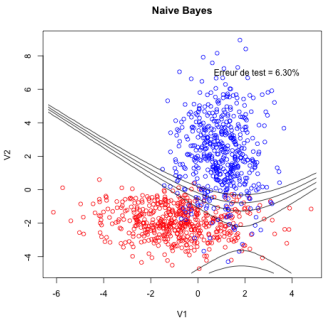


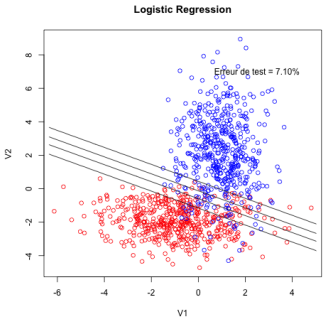


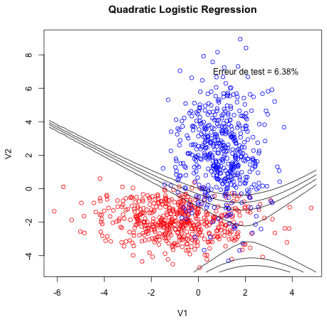
2.1.2 Synth2-1000

Dans ce qui suit nous avons représenté les résultats de calcul de l'erreur de test dans les données Synth2-1000. Les graphes sont représentés ci-dessous avec les frontières de décisions et les erreurs pour chaque classifieur.





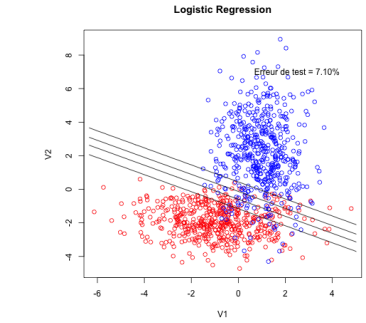


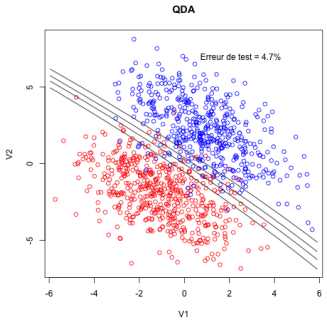


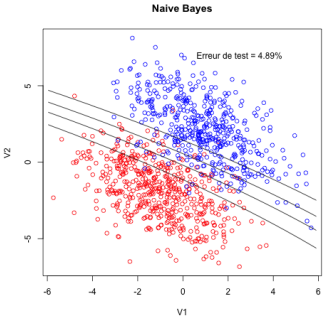
On remarque que l'erreur de test est inférieur a 9% pour tous les classifieurs. Selon la représentation des points on remarque que les classes ont a peu pres la même distribution et on n'a qu'une dizaine de points qui sont confondus entre les 2 classes. Par conséquent les classes peuvent être linéairement séparés, ceci est illustre par l'erreur de test puisqu'on voit que la regression quadratique n'apporte qu'une légère amélioration de l'ordre de 1%. Neanmois il faut remarquer que le classifieur Bayésien Naïve performe aussi bien que les régressions quadratiques. Ceci était prévisible vu que nos classes sont orientes vers les axes du plan et ... Ainsi il est plus intéressant dans ce cas d'utiliser le classifieur Bayésien Naïve vu qu'on n'estime que paramétrés au lieu de avec la régression quadratique ou même la logistique quadratique.

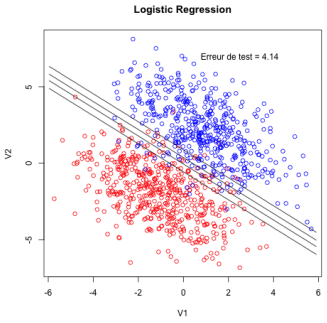
2.1.3 Synth3-1000

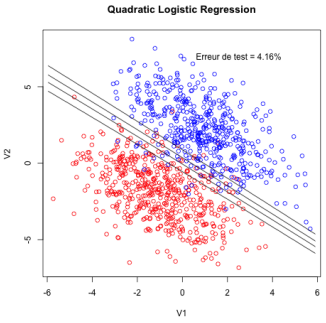
Nous nous intéressons a présent aux données de Synth3-1000 avec 1000 individus.











On remarque en représentant les données que les classes ont les mêmes orientations et distributions, et peuvent êtres linéairement séparés. Il est donc normal de voir que les erreurs de test du LDA sont faibles et que celles ci sont approximativement égales a celles du QDA.

2.2 Test sur données réelles

2.3 Pima

Ce jeu de données concerne des individus d'une population d'amérindiens et la prédiction du diabète selon les caractéristiques présentées.

LDA	QDA	Naive Bayes	Log Reg	Quad Log Reg	Tree
22.31%	24.20%	24.02%	22.18%	23.65%	25.72%

Nous remarquons que les taux d'erreurs sont considérablement élevés pour ce dataset. Ils tournent autour du 24% ce qui fait 20% plus élevés que les données simulées analysées précédemment Nous nous rappelons que les plots des donnees quantitatives de Pima lors de leur analyse au TP1 ne nous permettaient pas de distinguer les classes présentant le critère de présence du diabète ou pas. Il fut difficile de déterminer une si une des variables explicatives permettait de décider de leur différenciation.

2.4 Breast Cancer

2.4.1 Description des données

Avant de se lancer dans l'analyse de données, nous allons essayer de décrire ces données pour mieux analyser les résultats obtenus. Breast Cancer Winsconsin compte 683 individus et 9 variables quantitatives explicatives.

2.4.2 Analyse discriminantes et régression logistique

Le tableau ci-dessous montrent les résultats obtenus avec les différentes méthodes d'analyses discriminantes.

LDA	QDA	Naive Bayes	Log Reg
4.57%	5.03%	3.98%	4.02%

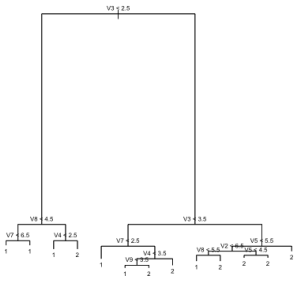
Contrairement aux résultats obtenus dans les données de *Synth* on remarque ici que les performances des classifieurs diffèrent. En effet le classifieur Bayésien Naïf et la régression logistique sont ceux qui classent au mieux nos données suivi par la methode d'analyse discriminante linéaire et quadratique. La methode la plus performante ici est le classifieur Bayésien Naïf. En effet si on s'intéresse a la matrice de variance on remarque que les valeurs dans la diagonale sont les plus grandes. La matrice est donc quasi-diagonale. Ce qui vérifie l'hypothèse d'indépendance des données du classifieur Bayésien Naïf.
La methode d'analyse linéaire n'est pas aussi performante bien car si on observe les matrice de variances de chaque classe on remarque que l'hypothèse d'homoscédasticité n'est pas vérifiée.

2.4.3 Conclusion

On remarque que dans les 3 cas la régression logistique fournir le pourcentage d'erreur le plus faible. Néanmoins cette méthode est trop couteuse quant aux nombre de paramètre a estimer. Remarquons qu'appliquer la la méthode de régression logistique quadratique aurait été trop couteux dans ce cas ou on a 8 variables d'une part.

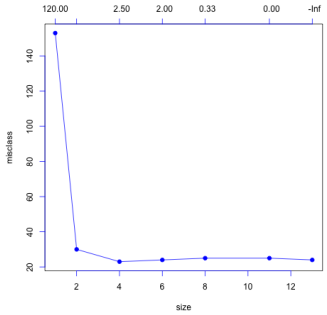
2.4.4 Arbres de décisions

Pour obtenir l'arbre de décision de l'ensemble d'apprentissage nous utilisons la fonction *tree* du *package tree*. Nous fixons les paramètres *control=tree.control(nobs=dim(Dapp)[1],mindev = 0.0001)* comme demandé dans ce TP. *nobs* est le nombre d'observations des données d'apprentissage , *mindev* est la deviance entre les noeuds. Si mindev=0 il essaye de fit parfaitement les données. La fixation de ces paramètres influence directement la taille (le nombre de nœuds) de l'arbre.



```
Variables actually used in tree construction:
[1] "V3" "V8" "V7" "V4" "V9" "V5" "V2"
Number of terminal nodes:  13
Residual mean deviance:  0.07417 = 32.78 / 442
Misclassification error rate: 0.01758 = 8 / 455
```

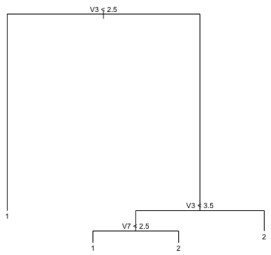
Ainsi sur l'ensemble d'apprentissage on a une erreur de mal classement de 1.76%. On remarque ici V3, V8, V7, V8, V9, V5 et V2. La complexité d'un arbre est évaluée avec le nombre de feuilles qu'il a puisque se sont les différentes classes. Ainsi la fonction **tree** nous retourne un arbre avec 13 nœuds et donc 13 partitions.
Nous allons a présent utiliser la méthode de cross validation pour trouver l'hauteur optimale de l'arbre. On utilise la fonction R **tree.cv** avec k=10 pour faire une *10-fold cross validation*, qui consiste a diviser l'ensemble d'apprentissage en 10 ensembles et utiliser un a chaque fois comme ensemble de validation. Les résultats sont représentés ci-dessous sous forme du graphe de l'évolution du nombre d'individus mal classes en fonction de la




```
min= which(bcw.cv$dev==min(bcw.cv$dev))
best.size= bcw.cv$size[min]
```

FIGURE 1 – Graphe de l'evolution des individus mal-classes en fonction du nombre de feuilles

On récupère donc la taille de l'arbre qui minimise l'erreur (le nombre d'individus mal classes) qui est 4. Ensuite on utilise la fonction**prune.misclass**permet d'élaguer l'arbre en mettant 4 comme taille de l'arbre. Grâce a la fonction **predict** on obtient en sortie le vecteur de prédiction de classes. La matrice ci-dessous est la matrice de confusion des vrais classements et des prédictions.



	1	2
1	135	7
2	6	80

Nous utilisons ensuite les instructions pour calculer l'erreur de test :

```
tree.pred = predict(bcw.cv.pruned, Xtst, type = "class")
res = with(Xtst, table(tree.pred, ztst))
error.test = res[1,2] + res[2,1] / nrow(Xtst)
print(paste("Test Error  ",error.test))
```

Nous obtenons 6.03% comme erreur de test.

3 Spams challenge