Rapport TP4, Discrimination

Sarah Richard et Solène Weill

24 juin 2016

1 Introduction

La discrimination regroupe l'ensemble des techniques visant à expliquer une variable Z à partir de variables explicatives X décrivant les individus d'une population P. L'objectif de ce TP est d'étudier à travers différents jeux de données plusieurs de ces méthodes afin d'en comprendre les différences et d'être capable d'en choisir une adéquate en fonction de nos connaissances sur les jeux de données et sur le domaine d'étude.
Dans un premier temps, nous expliquerons comment nous avons implémenté les fonctions permettant de construire les classifieurs. Nous expliquerons ensuite les résultats que nous avons obtenus avec les différents classifieurs sur des jeux de données de synthèse et sur des jeux de données réels. Pour finir, comme nous avons travaillé sur des données challenge, nous décrirons la démarche suivie afin de déterminer une méthode permettant une bonne classification des données.

2 Programmation

2.1 Analyse discriminante

L'analyse discriminante regroupe un ensemble de méthodes permettant de construire automatiquement des systèmes de prédiction. Pour ce faire, il faut donner une estimation des probabilités a posteriori d'appartenance aux classes. Dans cette première partie, nous avons implémenté des fonctions permettant de programmer trois modèles d'analyse discriminante dans le cas de jeux de données comptant 2 classes. Les trois premières fonctions consistent, pour chaque modèle, à calculer les estimateurs du maximum de vraisemblance des paramètres des lois.

En analyse discriminate, les hypothèses suivantes sont émises :

— X suit conditionnellement à chaque classe w_k, une loi normale multidimensionnelle d'espérance μ_k et de variance σ_k.
— Des hypothèses supplémentaires sur les valeurs des paramètres du modèle sont faites pour chaque classifieur.
— La variable Z à expliquer est qualitative.

Le code de chaque fonction réalisé est disponible en annexe. Nous tenons par ailleurs à signaler que nous avons décidé d'utiliser l'estimateur sans biais des matrices de covariance σ_k.

2.1.1 Analyse discriminante quadratique

L'hypothèse supplémentaire émise avec ce classifieur est que μ_k et σ_k sont différents dans chaque classe. La fonction prend en entrée les données d'apprentissage et le vecteur d'étiquettes associé. On calcule les trois paramètres suivant que l'on retourne dans une liste.

— π_k les proportions d'individus dans chaque classe.
— μ_k les centres des classes.
— σ_k les matrices de covariance de chaque classe.

2.1.2 Analyse discriminante linéaire

L'hypothèse supplémentaire émise avec ce classifieur est que μ_k est différent dans chaque classe mais que σ_k est la même pour toutes les classes (homoscédasticité). La fonction prend en entrée les données d'apprentissage et le vecteur d'étiquettes associé. On calcule les trois paramètres suivant que l'on retourne dans un objet.

— π_k les proportions d'individus dans chaque classe.
— μ_k les centres des classes.
— σ_k la matrice de covariance des deux classes.

2.1.3 Classifieur naïf bayésien

L'hypothèse supplémentaire émise avec ce classifieur est que μ_k et σ_k sont différents dans chaque classe et que les σ_k sont des matrices diagonales. Cela implique l'indépendance des variables X_j conditionnellement à Z. Ce modèle permet de réduire le nombre de paramètres à estimer et de gagner en robustesse. Cependant, on perd en flexibilité du modèle. La fonction prend en entrée les données d'apprentissage et le vecteur d'étiquettes associé. On calcule les trois paramètres suivant que l'on retourne dans une liste.

— π_k les proportions d'individus dans chaque classe.
— μ_k les centres des classes.
— σ_k les matrices diagonales de covariance de chaque classe.

2.1.4 Fonction permettant de prédire les étiquettes des individus du jeu de test

Cette fonction est commune aux trois classifieurs. Elle prend en paramètres d'entrée le jeu de données test ainsi qu'une liste contenant l'estimation des paramètres du modèle. Pour chaque classe, on calcule la fonction de densité conditionnelle de la classe. On calcule ensuite, pour chaque individu, la probabilité a posteriori qu'il appartienne à chaque classe. On associe enfin chaque individu à la classe dont la probabilité a posteriori est la plus élevée. L'objet retourné est une liste contenant le vecteur d'étiquettes et la matrice des probabilités a posteriori qui contient la probabilité d'appartenir à chacune des deux classes pour tous les individus.

2.2 Régression logistique

La régression logistique est une technique qui consiste à estimer directement les probabilités d'appartenance aux classes. On ne fait pas d'hypothèses sur la distribution conditionnelle des classes et la variable Z à expliquer est ici quantitative. Dans cette partie, le but était d'implémenter l'algorithme de Newton-Raphson. Cet algorithme itératif permet de trouver un optimum pour maximiser la log vraisemblance du vecteur de poids w. w_k est un indicateur de la pertinence de la variable explicative correspondante dans le processus de prédiction.

2.2.1 Régression logistique

La fonction prend en paramètres d'entrée les données d'apprentissage, le vecteur d'étiquettes associé, un booléen permettant de savoir s'il faut ajouter une ordonnée à l'origine et le seuil de précision, condition d'arrêt de la boucle. Ajouter une ordonnée à l'origine est intéressant lorsque les données ne sont pas répartie symétriquement par rapport à l'origine du repère. On commence par calculer le vecteur permettant de savoir à quelle classe l'individu x appartient. On crée ensuite la matrice colonne beta et on l'initialise comme le vecteur nul. Sa taille dépend du booléen *intr* indiquant la présence ou non d'une ordonnée à l'origine. Ensuite, tant que l'algorithme n'a pas convergé, on calcule les probabilité a posteriori d'appartenance aux classes de chaque individu, puis la matrice Hessienne et la nouvelle matrice beta. On incrémente le nombre d'itération. Une fois

que l'algorithme a convergé, on calcul la log vraisemblance de l'optimum. Pour finir, on renvoie un objet sous forme de liste contenant la matrice beta, le nombre d'itérations et la valeur de la log vraisemblance de l'optimum

La seconde fonction permet, à partir de la matrice beta renvoyée par la fonction précédente, de calculer les étiquettes du jeu de données test. Pour cela, on calcule les probabilités a posteriori pour chaque classe et on attribue à l'individu x la classe pour laquelle la probabilité a posteriori est la plus élevée.

2.2.2 Régression logistique quadratique

La régression logistique quadratique suit le même principe sauf que l'on augmente le nombre de variables explicatives pour résoudre le problème dans un espace plus compliqué que l'on espère linéaire. Ainsi, nous avons pu réutiliser les fonctions écrites précédemment en passant en paramètre le jeu de données d'apprentissage auquel on a ajouté des colonnes.

2.3 Arbres binaires

Les arbres binaires sont des méthodes d'apprentissage génériques. Ils permettent de résoudre, aussi bien, des problèmes de discrimination que de régression consistant à partitionner de manière récursive l'espace des caractéristiques en régions homogènes. La fonction prend un des jeux de données en entrée. On réalise une séparation des données en ensemble d'apprentissage et ensemble de test.
On commence par développer l'arbre complètement pour l'ensemble d'apprentissage puis on cherche le noeud pour lequel la valeur de déviance est minimum. Cela nous permet, par la suite, d'élaguer l'arbre à partir de ce noeud pour obtenir la séparation optimale des données en régions homogènes. A partir de ce nouvel arbre, on prédit la valeur des étiquettes des individus du jeu de test. Nous répétons ces opérations 100 fois afin de calculer une moyenne du taux d'erreur de prédiction sur le jeu de données test et sur le jeu de données d'apprentissage. Si on ne garde pas l'arbre complet, c'est parce que celui-ci est trop spécifique à l'échantillon d'apprentissage et se généralisera mal, ce qui fera augmenter le taux d'erreur de prédiction des classes des individus de test.

3 Application

3.1 Analyses sur données simulées

3.1.1 Données Synth1.1000

Le jeu de données Synth1.1000 est un jeu de données simulées, généré suivant une loi normale bivariée. L'hypothèse d'analyse discriminante est donc vérifiée.
Afin de vérifier les hypothèses sur les matrices de covariance et sur les centres de classe, nous avons séparé les données fournies par le fichier selon l'appartenance des individus à la classe 1 ou à la classe 2. Nous avons ensuite calculé les matrices de covariance. On se rend compte que celles-ci sont distinctes et ne sont pas diagonales. Les hypothèses supplémentaires satisfaites sont donc celles du classifieur d'analyse discriminante quadratique, ce qui corrobore les résultats d'erreur trouvés. Des trois classifieurs d'analyse discriminante, c'est celui qui a les meilleurs résultats. Comme nous pouvons nous y attendre, le taux d'erreur sur le jeu de données d'apprentissage est plus faible que celui du jeu de données test.
Nous avons effectué les régressions logistiques avec une ordonnée à l'origine. Nous avons fait ce choix car après avoir visualisé les données dans un repère, nous avons vu que celles-ci ne sont pas réparties de façon symétrique par rapport à l'origine du repère. Nous obtenons de meilleurs résultats avec la régression logistique simple et l'on peut voir que les résultats sont comparables à ceux de l'analyse

discriminante quadratique.
Avec les arbres binaires, on obtient un taux d'erreur élevé pour les données d'apprentissage et moins bon pour les données de test. Cela peut être dû au fait que le jeu de données est petit et que l'arbre final est encore trop proche des données d'apprentissage.

Nom du classifieur	Taux d'erreur sur les données d'apprentissage	Taux d'erreur sur les données de test
Analyse Discriminante Quadratique	0.02526237	0.02612613
Analyse Discriminante Linéaire	0.08005997	0.07642643
Classifieur Naïf Bayésien	0.03397301	0.03372372
Arbre binaire	0.02173913	0.03408408
Régression logistique	0.02466267	0.02612613
Regression logistique quadratique	0.02233883	0.03138138

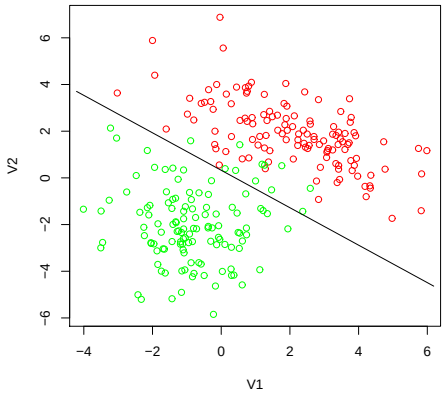

FIGURE 1 – Frontière de décision sur Synth1-1000 avec l'analyse discriminante linéaire

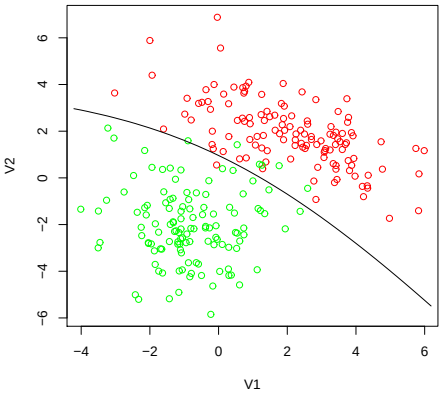

FIGURE 2 – Frontière de décision sur Synth1-1000 avec l'analyse discriminante quadratique

L'observation des frontières de décision et de l'arbre binaire nous permet de confirmer notre choix.

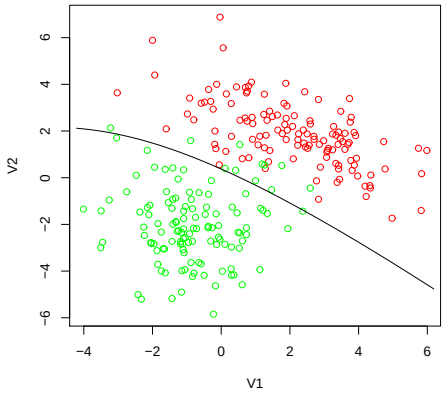

FIGURE 3 – Frontière de décision sur Synth1-1000 avec le classifieur naïf bayésien

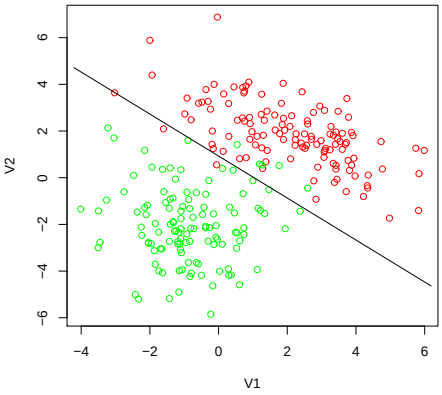

FIGURE 4 – Frontière de décision sur Synth1-1000 avec régression logistique

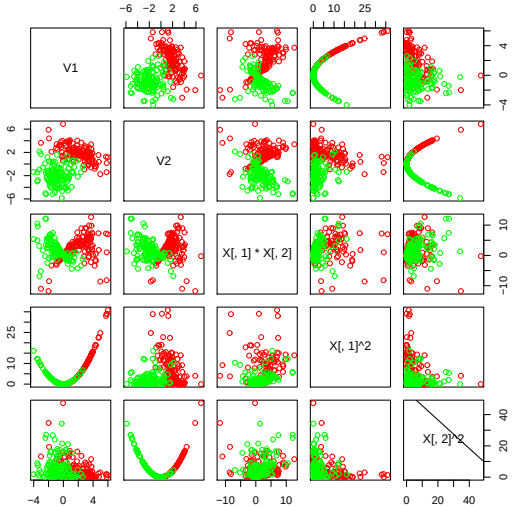

FIGURE 5 – Frontières de décision sur Synth1-1000 avec régression logistique quadratique

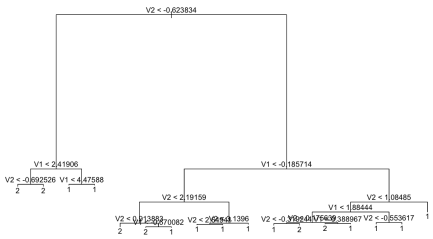

FIGURE 6 – Arbre Complet

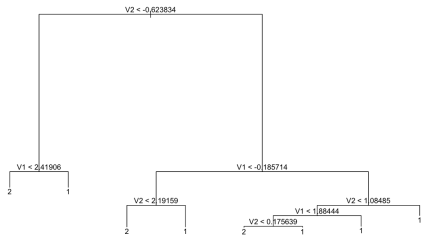

FIGURE 7 – Arbre élagué

3.1.2 Données Synth2.1000

Le jeu de donnée Synth2.1000 est un jeu de données simulées, généré suivant une loi normale bivariée. L'hypothèse d'analyse discriminante est donc vérifiée. Afin de vérifier les hypothèses sur les matrices de covariance et sur les centres de classe, nous avons séparé les données fournies par le fichier selon l'appartenance des individus à la classe 1 ou à la classe 2. Nous avons ensuite calculé les matrices de covariance. On se rend compte que celles-ci sont distinctes et ne sont pas diagonales. Cependant, les deux matrices sont plus similaires que celles du jeu de données synth1.1000. Les hypothèses supplémentaires satisfaites sont donc celles du classifieur d'analyse discriminante quadratique ce qui corrobore les résultats d'erreur trouvés. Des trois classifieurs d'analyse discriminante, c'est celui qui a les meilleurs résultats. Le fait que les matrices de covariance soient plus similaires permet de baisser le taux d'erreur des deux autres classifieurs d'analyse discriminante. Comme nous pouvons nous y attendre, le taux d'erreur sur le jeu de données d'apprentissage est plus faible que celui du jeu de données test.
Nous avons effectué les régressions logistiques avec une ordonnée à l'origine. Nous avons fait ce choix car après avoir visualisé les données dans un repère, nous avons vu que celles-ci ne sont pas réparties de façon symétrique par rapport à l'origine du repère. Nous obtenons de meilleurs résultats avec la régression logistique quadratique, ce qui signifie que nos données se trouve dans un espace plus linéaire que précédemment, et l'on peut voir que les résultats sont comparables à ceux de l'analyse discriminante quadratique. En effet, on peut observer que les données ne sont pas réparties de façon linéaire, ce qui explique pourquoi l'analyse discriminante quadratique donne de bons résultats.
Avec les arbres binaires, on obtient un taux d'erreur élevé pour les données d'apprentissage et moins bon pour les données de test. Cela peut être dû au fait que le jeu de données est petit et que l'arbre final est encore trop proche des données d'apprentissage.

Nom du classifieur	Taux d'erreur sur les données d'apprentissage	Taux d'erreur sur les données de test
Analyse Discriminante Quadratique	0.009009009	0.01017964
Analyse Discriminante Linéaire	0.01876877	0.0247006
Classifieur Naïf Bayésien	0.01478979	0.0160479
Arbre binaire	0.009159159	0.02275449
Régression logistique	0.01058559	0.01227545
Regression logistique quadratique	0.01051051	0.01047904

L'observation des frontières de décision et de l'arbre binaire nous permet de confirmer notre choix.

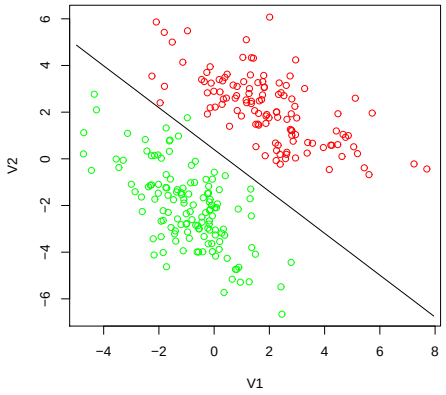

FIGURE 8 – Frontière de décision sur Synth2-1000 avec l'analyse discriminante linéaire

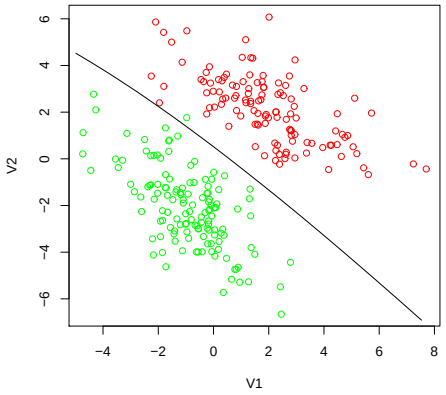

FIGURE 9 – Frontière de décision sur Synth2-1000 avec l'analyse discriminante quadratique

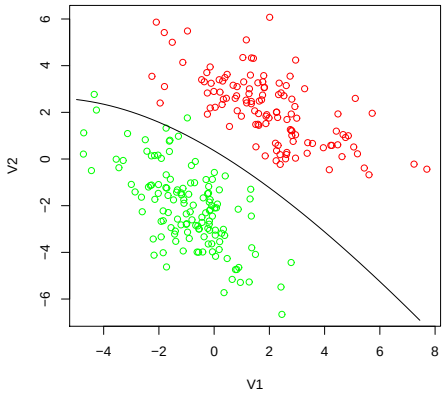

FIGURE 10 – Frontière de décision sur Synth2-1000 avec le classifieur naïf bayésien

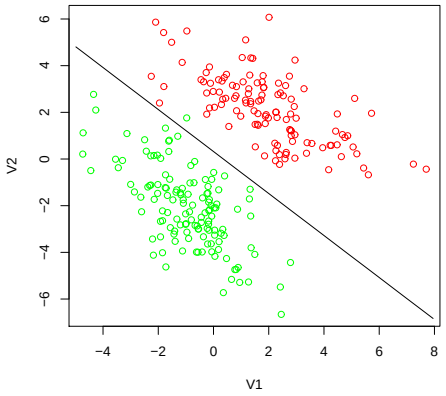

FIGURE 11 – Frontière de décision sur Synth2-1000 avec régression logistique

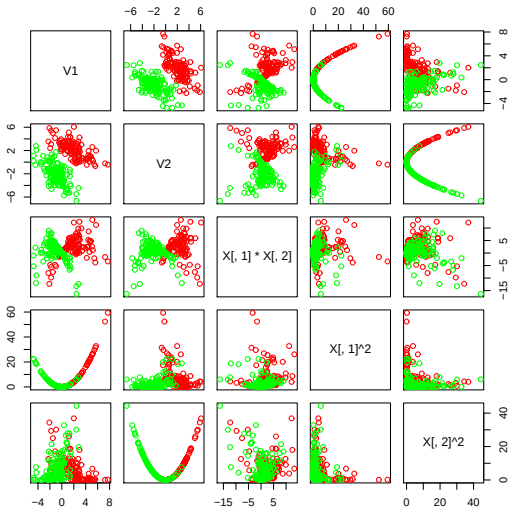

FIGURE 12 – Frontières de décision sur Synth2-1000 avec régression logistique quadratique

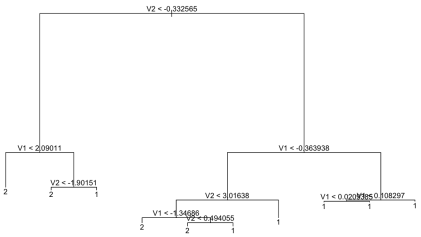

FIGURE 13 – Arbre Complet

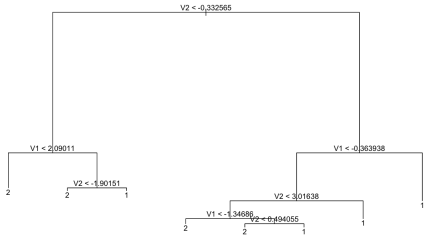

FIGURE 14 – Arbre élagué

3.1.3 Données Synth3.1000

Le jeu de données Synth3.1000 est un jeu de données simulées, généré suivant une loi normale bivariée. L'hypothèse d'analyse discriminante est donc vérifiée. Afin de vérifier les hypothèses sur les matrices de covariance et sur les centres de classe, nous avons séparé les données fournies par le fichier selon l'appartenance des individus à la classe 1 ou à la classe 2. Nous avons ensuite calculé les matrices de covariance. On se rend compte que celles-ci sont distinctes et ne sont pas diagonales. Les deux matrices ne contiennent que des valeurs positives. Les hypothèses supplémentaires satisfaites sont donc celles du classifieur d'analyse discriminante quadratique ce qui corrobore les résultats d'erreur trouvés. Des trois classifieurs d'analyse discriminante, c'est celui qui a les meilleurs résultats. Comme nous pouvons nous y attendre, le taux d'erreur sur le jeu de données d'apprentissage est plus faible que celui du jeu de données test.
Nous avons effectué les régressions logistiques avec une ordonnée à l'origine. Nous avons fait ce choix car après avoir visualisé les données dans un repère, nous avons vu que celles-ci ne sont pas réparties de façon symétrique par rapport à l'origine du repère. Nous obtenons de meilleurs résultats avec la régression logistique quadratique, ce qui signifie que nos données se trouve dans un espace plus linéaire que précédemment, et l'on peut voir que les résultats sont comparables à ceux de l'analyse discriminante quadratique. En effet, on peut observer que les données ne sont pas réparties de façon

linéaire, ce qui explique pourquoi l'analyse discriminante quadratique donne de bons résultats.
Avec les arbres binaires, on obtient un taux d'erreur plus élevé que pour la plupart des autres classifieurs, cela peut s'expliquer car le jeu de données est petit et que l'arbre final est encore trop proche des données d'apprentissage.

Nom du classifieur	Taux d'erreur sur les données d'apprentissage	Taux d'erreur sur les données de test
Analyse Discriminante Quadratique	0.01131934	0.01141141
Analyse Discriminante Linéaire	0.02397301	0.02336336
Classifieur Naïf Bayésien	0.01277361	0.01258258
Arbre binaire	0.01161919	0.01681682
Régression logistique	0.01656672	0.01591592
Regression logistique quadratique	0.01124438	0.01306306

L'observation des frontières de décision et de l'arbre binaire nous permet de confirmer notre

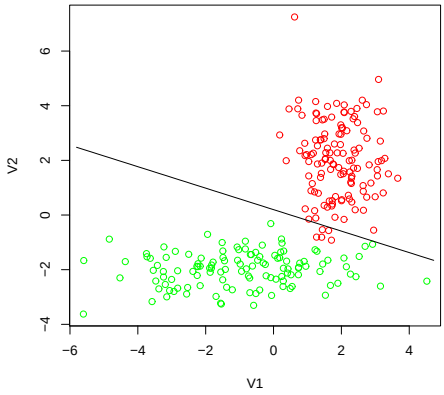

FIGURE 15 – Frontière de décision sur Synth3-1000 avec l'analyse discriminante linéaire

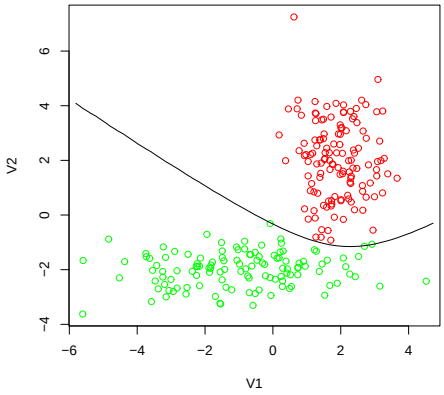

FIGURE 16 – Frontière de décision sur Synth3-1000 avec l'analyse discriminante quadratique

choix.

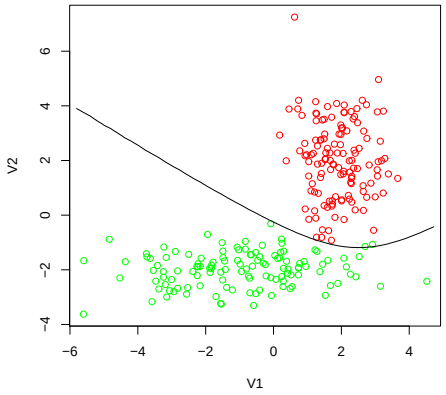

FIGURE 17 – Frontière de décision sur Synth3-1000 avec le classifieur naïf bayésien

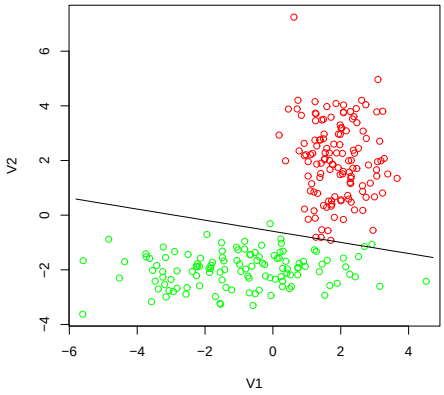

FIGURE 18 – Frontière de décision sur Synth3-1000 avec régression logistique

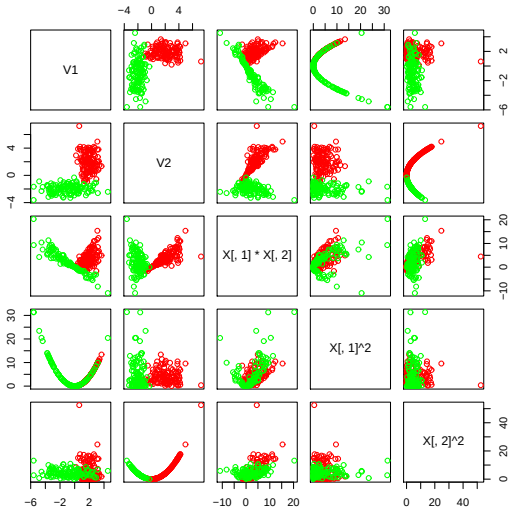

FIGURE 19 – Frontières de décision sur Synth3-1000 avec régression logistique quadratique

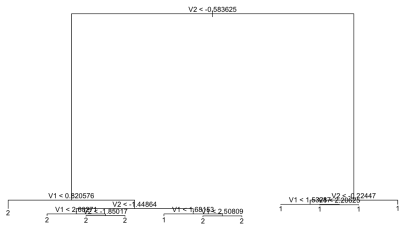

FIGURE 20 – Arbre Complet

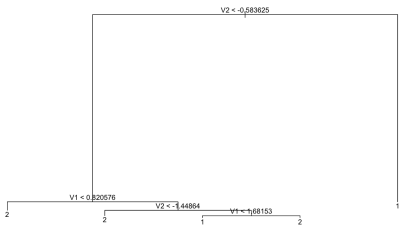

FIGURE 21 – Arbre élagué

3.2 Analyses sur données réelles

3.2.1 Données "Pima"

Le jeu de données "Pima" a été réalisé grâce à une étude menée par l'institut national du diabète et des maladies digestives et du rein sur des femmes indiennes pima vivant près de Phoenix.

Nom du classifieur	Taux d'erreur sur les données d'apprentissage	Taux d'erreur sur les données de test
Analyse Discriminante Quadratique	0.212507	0.240678
Analyse Discriminante Lineaire	0.209662	0.2198305
Classifieur Naif Bayesien	0.2297183	0.2359322
Régression logistique	0.2060845	0.2217514
Regression logistique quadratique	0.1954085	0.2339548
Arbre Binaire	0.1184225	0.2623164

On observe ainsi que l'analyse discriminante linéaire nous renvoie le taux d'erreur le plus faible. On classe les individus en fonction de leur distance avec le centre de la classe dont ils sont la plus proche. L'utilisation de ce modèle est supposé signifier que la matrice de variance est commune à toutes les classes (hypothèse d'homoscédasticité). Cependant, après observation de nos matrices de variance, nous avons découvert que ce n'était pas le cas. Ainsi, le classifieur était trop adapté au modèle d'apprentissage. Nous avons donc décidé de garder le modèle de la régression logistique. Cependant, le taux d'erreur reste élevé, plus de 20% pour chacun des classifieurs. Il faudrait utiliser un autre classifieur pour ces données.

3.2.2 Données "breast cancer Wisconsin"

Dans ce jeu de données, il s'agit de prévoir la nature maligne ou bénigne de la tumeur à partir des variables biologiques. L'objectif final, à ne pas perdre de vue, est la comparaison de ces méthodes afin de déterminer la plus efficace pour répondre au problème de prévision.

Nom du classifieur	Taux d'erreur sur les données d'apprentissage	Taux d'erreur sur les données de test
Analyse Discriminante Quadratique	0.0438022	0.05144737
Analyse Discriminante Linéaire	0.03903297	0.04407895
Classifieur Naïf Bayésien	0.0363956	0.03877193
Arbre binaire	0.03182018	0.03605727
Régression logistique	0.03197802	0.04026316

Nous avons réalisé la régression logistique avec une ordonnée à l'origine car après visualisation

des données de départ, nous avons vu qu'elles ne sont pas réparties autour de l'origine du repère. Nous étions donc sûres d'avoir de meilleurs résultats avec une ordonnée à l'origine. La méthode qui a le taux d'erreur le plus faible pour le jeu de données de test est l'arbre binaire. Il était très probable que les méthodes d'analyse discriminante est le taux d'erreur le plus élevé car les hypothèses de base ne sont pas vérifiées. Les taux d'erreur obtenus avec la régression logistique sont proches de ceux obtenus pour les arbres. Pour ces deux méthodes, on ne fait pas d'hypothèses de départ il est donc normal que ces deux classifieurs soient plus robustes au changement des distributions des individus dans les classes des jeux de données réels.

4 Challenge : données "Spam"

Il est possible d'utiliser l'ACP afin de réduire le nombre de variables tout en assurant la cohérence des données. Pour cela, nous calculons les composantes principales et obtenons 842.8351 en somme des valeurs propres. Or, avec les trois premières composantes principales, nous obtenons un pourcentage d'inertie expliqué de 96%. On peut en conclure que nous nous situons dans un espace de dimension 3. En utilisant la forêt aléatoire avec cette espace, nous obtenons :

Nom du classifieur	Taux d'erreur sur les données d'apprentissage	Taux d'erreur sur les données de test
Forêts aléatoires	0.0004563233	0.06030659

Ces résultats sont très faibles pour l'ensemble d'apprentissage mais restent élevés en ce qui concerne l'ensemble de test. Le taux d'erreur est plus faible en conservant l'ensemble des données Spam.

Nous testons ainsi sur l'ensemble des données Spams différents classifieurs afin de trouver le meilleur modèle à utiliser.

Nom du classifieur	Taux d'erreur sur les données d'apprentissage	Taux d'erreur sur les données de test
Régression Logistique	0.06835072	0.07296151
Arbre Binaire	0.03578879	0.0816047
Forêts aléatoires	0.02710235	0.05720809

Ainsi, le classifieur le plus efficace est celui des forêts aléatoires avec seulement 5.7% d'erreurs. L'algorithme des forêts aléatoires est une méthode de bagging. Il donne de meilleurs résultats sur le jeu de données spam car celui-ci étant relativement petit, créer des arbres pour chaque échantillon de l'ensemble d'apprentissage et réaliser une procédure de vote est efficace. Le résultat gardé est celui de la majorité des arbres cela permet donc de réduire les erreurs de classification commises.

5 Conclusion

Ce TP nous a donné l'occasion d'appliquer l'analyse discriminante au travers trois modèles différents respectant différentes hypothèses, ainsi que la regression linéaire au travers deux modèles, les arbres binaires et la forêt aléatoire. Grâce aux jeux de données fournis, nous avons pu tester ces modèles afin de savoir dans quelles conditions ils étaient les plus efficaces. De plus, nous avons ensuite pu les appliquer sur de véritables jeux de données et confirmer que les modèles de régression linéaire étaient plus facilement applicables car ils ne comportent pas d'hypothèses à vérifier pour être utilisés.

6 Annexe

6.1 Fonction de calcul de la matrice de covariance.

```
sigmaf<-function(Xapp, zapp, mu){
nbInd <- table(zapp);
sigmaFinal1<-matrix(0, 2, length(Xapp))
sigmaFinal2<-matrix(0, 2, length(Xapp))
 for(i in 1:length(Xapp)){
sigm1<-0;
sigm2<-0;
 for(j in 1:length(Xapp[,1])) {
if(zapp[j]==1){
sigm1<-sigm1 + t(as.matrix(Xapp[j,])-mu[1,])%*%(as.matrix(Xapp[j,])-mu[1,]);
}
else{
sigm2<-sigm2 +t(as.matrix(Xapp[j,])-mu[2,])%*%(as.matrix(Xapp[j,])-mu[2,]);
}
 }

 }
sigmaFinal1<-sigm1/nbInd[1];
sigmaFinal2<-sigm2/nbInd[2];
sig<-list(sigmaFinal1 = sigmaFinal1, sigmaFinal2 = sigmaFinal2)
sig
}
```

6.2 Fonction pour l'analyse discriminante quadratique.

```
adq.app<-function(Xapp, zapp){
pi<-table(zapp)/length(zapp);
mu<-matrix(0, 2, length(Xapp));
param<-c();
nbInd <- table(zapp);
for(i in 1:length(Xapp)){
sumXapp<-0;
sumYapp<-0;
for(j in 1:length(Xapp[,1])) {
if(zapp[j]==1){
sumXapp<-sumXapp+Xapp[j,i];
}
else if(zapp[j]==2){
sumYapp<-sumYapp+Xapp[j,i];
}
}
mu[1,i]<-sumXapp/nbInd[1];
mu[2,i]<-sumYapp/nbInd[2];
}
sigmaFinal1<-matrix(0, 2, length(Xapp))
sigmaFinal2<-matrix(0, 2, length(Xapp))
 for(i in 1:length(Xapp)){
```

```
sigm1<-0;
sigm2<-0;
 for(j in 1:length(Xapp[,1])) {
if(zapp[j]==1){
sigm1<-sigm1 + t(as.matrix(Xapp[j,])-mu[1,])%*%(as.matrix(Xapp[j,])-mu[1,]);
}
else{
sigm2<-sigm2 +t(as.matrix(Xapp[j,])-mu[2,])%*%(as.matrix(Xapp[j,])-mu[2,]);
}
 }

 }
sigmaFinal1<-as.array((sigm1/nbInd[1])*(nbInd[1]/(nbInd[1]-1)));
sigmaFinal2<-as.array((sigm2/nbInd[2])*(nbInd[2]/(nbInd[2]-1)));
sig<-list(sigmaFinal1 = sigmaFinal1, sigmaFinal2 = sigmaFinal2);
param$proportion<-pi;
param$mu<-mu;
param$sigmak<-sig
param;
}
```

6.3 Fonction pour l'analyse discriminante linéaire.

```
adl.app<-function(Xapp, zapp) {
pi<-table(zapp)/length(zapp);
mu<-matrix(0, 2, length(Xapp));
nbInd <- table(zapp);
param<-c();
for(i in 1:length(Xapp)){
sumXapp<-0;
sumYapp<-0;
for(j in 1:length(Xapp[,1])) {
if(zapp[j]==1){
sumXapp<-sumXapp+Xapp[j,i];
}
else if(zapp[j]==2){
sumYapp<-sumYapp+Xapp[j,i];
}
}
mu[1,i]<-sumXapp/nbInd[1];
mu[2,i]<-sumYapp/nbInd[2];
}
sigmaFinal1<-matrix(0, 2, length(Xapp))
sigmaFinal2<-matrix(0, 2, length(Xapp))
 for(i in 1:length(Xapp)){
sigm1<-0;
sigm2<-0;
 for(j in 1:length(Xapp[,1])) {
if(zapp[j]==1){
sigm1<-sigm1 + t(as.matrix(Xapp[j,])-mu[1,])%*%(as.matrix(Xapp[j,])-mu[1,]);
}
```

```
else{
sigm2<-sigm2 +t(as.matrix(Xapp[j,])-mu[2,])%*%(as.matrix(Xapp[j,])-mu[2,]);
}
 }

 }
sigmaFinal1<-(sigm1/nbInd[1])*(nbInd[1]/(nbInd[1]-1));
sigmaFinal2<-(sigm2/nbInd[2])*(nbInd[2]/(nbInd[2]-1));
sig<-(1/(length(Xapp[,1])-2))*(((nbInd[1]-1)*sigmaFinal1)+((nbInd[2]-1)*sigmaFinal2))
sig<-list(sigmaFinal1 = sig, sigmaFinal2 = sig);
param$proportion<-pi;
param$mu<-mu;
param$sigmak<-sig
param;
}
```

6.4 Fonction pour le classifieur naïf bayésien

```
nba.app<-function(Xapp, zapp) {
pi<-table(zapp)/length(zapp);
mu<-matrix(0, 2, length(Xapp));
nbInd <- table(zapp);
param<-c();
for(i in 1:length(Xapp)){
sumXapp<-0;
sumYapp<-0;
for(j in 1:length(Xapp[,1])) {
if(zapp[j]==1){
sumXapp<-sumXapp+Xapp[j,i];
}
else if(zapp[j]==2){
sumYapp<-sumYapp+Xapp[j,i];
}
}
mu[1,i]<-sumXapp/nbInd[1];
mu[2,i]<-sumYapp/nbInd[2];
}

sigmaFinal1<-matrix(0, 2, length(Xapp))
sigmaFinal2<-matrix(0, 2, length(Xapp))
 for(i in 1:length(Xapp)){
sigm1<-0;
sigm2<-0;
 for(j in 1:length(Xapp[,1])) {
if(zapp[j]==1){
sigm1<-sigm1 + t(as.matrix(Xapp[j,])-mu[1,])%*%(as.matrix(Xapp[j,])-mu[1,]);
}
else{
sigm2<-sigm2 +t(as.matrix(Xapp[j,])-mu[2,])%*%(as.matrix(Xapp[j,])-mu[2,]);
}
 }
```

```
 }
sigmaFinal1<-diag((sigm1/nbInd[1])*(nbInd[1]/(nbInd[1]-1)));
sigmaFinal2<- diag((sigm2/nbInd[2])*(nbInd[2]/(nbInd[2]-1)));
sig<-list(sigmaFinal1 = diag(sigmaFinal1), sigmaFinal2 = diag(sigmaFinal2))
param$proportion<-pi;
param$mu<-mu;
param$sigmak<-sig
param;
}
```

6.5 Fonction de calcul des probabilités a posteriori

```
ad.val<-function(param, Xtst){
densite1<-mvdnorm(Xtst,param$mu[1,],as.matrix(param$sigmak$sigmaFinal1));
densite2<-mvdnorm(Xtst,param$mu[2,],as.matrix(param$sigmak$sigmaFinal2));
etiquette<-c();
prob1<-(densite1*param$proportion[1])/(densite1*param$proportion[1] + densite2*param$proportion[2] )
prob2<-(densite2*param$proportion[2])/(densite1*param$proportion[1] + densite2*param$proportion[2] )
densite<-matrix(0,length(Xtst[,1]),2)
for(i in 1:length(prob1)){

if (prob1[i]>prob2[i]){
etiquette[i]<-1

}else{
etiquette[i]<-2
}
}
densite[,1]<-prob1
densite[,2]<-prob2
res<-list(etiquette=etiquette, prob=densite);
res;
}
```

6.6 Fonction d'apprentissage pour la régression logistique.

```
log.app<-function(Xapp, zapp, intr, epsi){

logarithme<-0;
tab<-c();
pTot<-c();
nbIteration<-0;
diff<-Inf;

for(j in 1:length(Xapp[,1])){
if(zapp[j]==1){
tab[j]<-1;
}else{
tab[j]<-0;
}
```

```
}

#si on ajoute une ordonnée à l'origine alors beta est de dimension n+1*1 sinon beta est de dimension
if(intr==1){
beta<-matrix(0,dim(Xapp)[2]+1,1);
Xapp<-cbind(rep(1,dim(Xapp)[1]), Xapp);
}else{
beta<-matrix(0,dim(Xapp)[2],1);
}
while(epsi < diff){
print("pascoucou")
MXapp<-as.matrix(Xapp);
pTot<-post.pr(beta,MXapp);
gradient<-t(MXapp)%*%(tab-pTot);
W<-diag(as.vector(pTot*(1-pTot)));
matHessienne<- -t(MXapp)%*%W%*%MXapp;
wpred<-beta;
beta<-beta - ginv(matHessienne)%*%gradient;
diff<-sqrt(sum((wpred-beta)^2));
nbIteration<-nbIteration+1;
}
logarithme<-sum(tab*(MXapp%*%beta)-log(1+exp(MXapp%*%beta)));
resultat<-list(beta=beta, nbIteration=nbIteration, logarithme=logarithme);
resultat
}
```

6.7 Fonction de classement pour la régression logistique.

```
log.val<-function(Xtst, beta){
class<-c();
#test pour savoir si on doit ajouter une ordonnée à Xtst
if(length(beta)!=length(Xtst[1,])){
Xtst<-cbind(rep(1,dim(Xtst)[1]), Xtst);
}
 Xtst<-as.matrix(Xtst);
  beta<-as.matrix(beta);
  probaPosteriori1<-post.pr(beta, Xtst);
 probaPosteriori2<-1-probaPosteriori1;

 for(i in 1:length(probaPosteriori1[,1])){
  if(probaPosteriori1[i]>probaPosteriori2[i]){
  class[i]<-1;
  }else{
  class[i]<-2;
  }
 }
 matriceProb<-cbind(probaPosteriori1,probaPosteriori2)
 result<-list(prob=matriceProb, etiquette=class)
 result;
}
```

6.8 Fonction de calcul des probabilités a posteriori pour la régression logistique.

```
post.pr<-function(beta, X){
somme<-X%*%beta;
probaPosteriorie<-exp(somme)/(1+exp(somme));
probaPosteriorie;
}
```

6.9 Fonction pour le calcul de taux d'erreur pour les arbres binaires

```
aleaArbre<-function(donnees) {
comptapp<-0;
compttst<-0;
X <- donnees[,1:7]
z <- donnees[,8]
for(i in 1:100) {
donnees.sep <- separ1(X, z)
Xapp <- donnees.sep$Xapp;
zapp <- donnees.sep$zapp;
Xtst <- donnees.sep$Xtst;
#Xtst2 <- data.frame(V1 = donnees.sep$Xtst[,1], V2 = donnnees.sep$Xtst[,2]);
ztst <- donnees.sep$ztst;
#Dapp<-cbind(Xapp, classa=as.factor(zapp))
Dapp<-cbind(Xapp, class = as.factor(zapp))
#Dtst<-cbind(Xtst, classt=as.factor(ztst))
tapp<-tree(Dapp$class~., data=Dapp,control=tree.control(nobs=dim(Dapp)[1],mindev = 0.0001));

  #cvapp<-cv.tree(tapp, FUN=prune.misclass)
  cvapp<-cv.tree(tapp);
   min<-min(cvapp$dev);
   for(i in 1:length(cvapp$size)) {
    if(cvapp$dev[i]==min && cvapp$size[i]!=1) {
    node<-cvapp$size[i]
    }
   }
   papp<- prune.misclass(tapp, best = node);
  #ptst<- prune.misclass(ttst, best = node);
  #papp<-prune.misclass(tapp, k=cvapp$k)
   ClassAppEt<-predict(papp, Dapp, type="class"); #Etiquettes classes app
    ClassTstEt<-predict(papp, newdata=Xtst, type="class"); #Etiquettes classes tst

     for(j in 1:length(ClassAppEt)) {
     #Si classe 1 > class 2 ... Sinon
     if(ClassAppEt[j] != zapp[j]) {
     comptapp<-comptapp+1;
     }
     }
     for(j in 1:length(ClassTstEt)) {
     #Si classe 1 > class 2 ... Sinon
     if(ClassTstEt[j] != ztst[j]) {
```

```
     compttst<-compttst+1;
     }
     }
   }
    terrapp<-comptapp/(length(ClassAppEt)*100);
    terrtst<-compttst/(length(ClassTstEt)*100);
    res<-cbind(terrapp, terrtst);
    res
}
```

6.10 Fonction pour le calcul des taux d'erreur avec la forêt.

```
aleaForet<-function(donnees) {
comptapp<-0;
compttst<-0;
X <- donnees[,1:3]
z <- donnees[,4]
#X <- donnees[,2:58]
#z <- donnees[,59]
for(i in 1:20) {
donnees.sep <- separ1(X, z)
Xapp <- donnees.sep$Xapp;
zapp <- donnees.sep$zapp;
Xtst <- donnees.sep$Xtst;
ztst <- donnees.sep$ztst;
Dapp<-cbind(Xapp, class = as.factor(zapp))
  rf <- randomForest(Dapp$class ~ ., Xapp, ntree = 500, mtry = 2, importance = TRUE)
  ClassAppEt<-predict(rf, newdata = Xapp, type="class"); #Etiquettes classes app
  ClassTstEt<-predict(rf, newdata = Xtst, type = "class")

     for(j in 1:length(ClassAppEt)) {
     #Si classe 1 > class 2 ... Sinon
     if(ClassAppEt[j] != zapp[j]) {
     comptapp<-comptapp+1;
     }
     }
     for(j in 1:length(ClassTstEt)) {
     #Si classe 1 > class 2 ... Sinon
     if(ClassTstEt[j] != ztst[j]) {
     compttst<-compttst+1;
     }
     }
   }
    terrapp<-comptapp/(length(ClassAppEt)*20);
    terrtst<-compttst/(length(ClassTstEt)*20);
    res<-cbind(terrapp, terrtst);
    res
}
```