SORBONNE UNIVERSITES

SY09 : Rapport Second Projet

Emmanuelle Lejeail, Xie Yihan

29 juin 2018

1 Discrimination

1.1 Description des données

Nous sommes en présence de 5 jeux de données :

- Breastcancer : 30 variables descriptives et 569 observations
- Ionosphere : 34 variables descriptives et 360 observations
- Sonar : 60 variables descriptives et 208 observations
- Spambase : 57 variables descriptives et 4601 observations
- Spambase2 : 57 variables descriptives et 4601 observations

Nous sommes donc en présence de jeux de données possédant pour certains beaucoup de variables (Sonar) et un faible nombre d'observations ce qui risque d'entraver l'apprentissage d'un modèle avec de bonnes performances. De l'autre côté Spambase et Spambase2 possèdent beaucoup d'observations, la nombre d'observations n'est donc pas limitant pour l'apprentissage et cela peut même s'avérer être un avantage pour l'apprentissage de modèles complexes comme la régression Logistique ou les Arbres de décisions.

Dans l'annexe A, nous pouvons visualiser la projection de nos jeux de données sur le premier plan factoriel, grâce à la PCA. Ceci nous permet d'avoir une meilleure idée de la dispersion de nos classes afin de mieux comprendre leur géométrie et la manière dont cela pourrait influencer les performances des différents modèles que nous allons utiliser.

Pour le jeu de données Breastcancer par exemple, nous pouvons identifier visuellement une frontière relativement linéaire ou quadratique. Ceci peut nous laisser penser que des modèles comme l'ADL ou l'ADQ devraient avoir d'assez bonnes performances bien que les classes ne semblent pas suivre une loi normale multivariée (testé avec la fonction `mvShapiro.Test()` qui permet de tester si une variable suit une loi normale multivariée). Les matrices de covariances des classes sont bien définies positives mais elles sont différentes selon la classe considérée. Des tests de Pearson pour tester la corrélation de 2 variables 2 à 2 semblent indiquer que la plupart des variables sont bien corrélées les unes aux autres conditionnellement à la classe.

Pour le jeu de données Ionosphère, cette fois-ci la frontière entre les deux classes n'est pas visible à l'oeil nu sur le premier plan factoriel. Les tests de Pearson indiquent que les variables semblent corrélées sachant la classe. Les classes semblent entre-mêlées ce qui peut laisser penser que les performances des modèles d'apprentissages seront moindres surtout en ce qui concerne l'ADL, l'ADQ puisque les hypothèses d'homos-

cédasticité et de normalité des classes semblent violées.

En ce qui concerne le jeu de données Sonar, le constat est assez similaire à celui du jeu de données précédent mais nous possèdons un échantillon de données encore plus mince. Lorsque l'on étudie les matrices de covariances des classes, nous pouvons nous apercevoir qu'elles sont différentes (visibles en annexe G). Ceci laisse penser que l'hypothèse d'homoscédasticité n'est pas respectée et que les classes ne sont pas gaussiennes. Il est a noté que les variables ne semblent pas indépendantes conditionnellement à la classe (tests de Pearson), les performances du classifieur baysien naïf seront donc certainement faibles.

Le jeu de données Spambase quant à lui, semble possèder une frontière discernable à l'oeil nu mais dont la nature complexe n'est vraisemblablement pas linéaire ou quadratique. De ce fait, l'on peut penser que le modèle de la régression logistique pourrait être adapté. De plus, les matrices de covariances des deux classes sont très différentes, les classes ne semblent pas suivre une distribution gaussienne et les tests de Pearson nous indiquent que beaucoup de variables ne sont pas indépendante conditionnellement à la classe, ce qui laisse penser que le classifieur Baysien naïf aura de piètre performances.

Le jeu de données Spambase2 est tiré du jeu Spambase à la différence que les données ont été transformées en données binaires. Encore une fois, il semble exister une frontière mais difficilement perceptible à l'oeil nu de nature incertaine. Les variables sont souvent corrélées les unes aux autres et ne semblent pas suivre de distribution gaussienne. Ceci laisse à penser que les modèles complexes comme la régression logistique ou encore l'arbre de décision pourraient avoir de bonnes performances. Mais il est également possible que l'ADL ou l'ADQ puissent établir des prédictions pertinentes.

1.2 Traitements préalables sur les données

Afin de pouvoir traiter tous nos jeux de données et de leur appliquer chacun des différents modèles, il est nécessaire d'effectuer certains traitement qui entravent le bon apprentissage à l'aide du modèle de l'ADQ par exemple. En effet, certaines variables descriptives sont parfois trop corrélées les unes aux autres. Les jeux de données concernés sont : Ionosphère, Spambase et Spambase2.

Pour résoudre ce problème, une première solution possible est d'utiliser l'ACP sur notre jeu de données. Les nouvelles coordonnées récupérées sont exprimés par rapport à un nouveau repère dans lequel nos variables sont moins corrélées qu'auparavant. Ceci nous permet d'appliquer nos modèles sans soucis de déficience de rang pour l'ADQ.

Nous avons décider de réaliser les apprentissages avec suffisamment de composantes principales de manière à atteindre une 90% d'inertie cumulée. Pour Breastcancer nous avons sélectionné les 17 premières composantes ; pour Ionosphère, les 26 premières composantes ; pour Sonar, les 39 premières composantes ; pour Spambase les 47 premières composantes et pour Spambase2 les 48 premières composantes.

Comme seconde solution de traitement, nous avons décidé de sélectionner à partir de notre table des `p.value` des tests de Pearson, les variables possédant une forte probabilité de corrélation aux autres, afin de les retirer de notre jeu de données. De ce fait, il est certains que nous perdons de l'information qui aurait pu nous être utile pour mieux comprendre les frontières de nos classes, cependant nous tentons de minimiser cet effet en retirant le moins de variables possible.

Pour les données Spambase les variables qui semblent avoir une forte corrélation avec les autres sont : X2, X4, X14, X22, X32-36, X38, X41-42, X44, X47-49, X54-55. Pour établir cet ensemble de variables, nous procédons de manière itérative en supprimant les probabilité de corrélation en décroissant (en commençant par la variable de probabilité la plus forte). Nous nous arrêtons lorsque nous avons atteint un ensemble de variables nous nous permettant d'appliquer tous nos modèles. Nous procédons de la même manière pour Spambase2. Les variables qui semblent avoir une forte corrélation aux autres sont : X4, X10, X19, X31-39, X41-42, X44, X46-48, et X55.

1.3 Résultats des différents modèles

Pour tester les différents modèles sur chaque jeu de données, nous avons décidés de créer une fonction nous permettant, en donnant X (la table des individus sans leur classe) et z (le vecteur contenant les classes) en entrée et un nombre N, de tester l'apprentissage de chaque modèle N fois sur le jeu de données. Cette fonction nous retourne une matrice contenant les taux d'erreurs de chaque modèle, en colonnes pour chaque apprentissage effectué.

Cette fonction s'inspire de la méthode du bootstrap en séparant au hasard notre jeu de données en des données d'apprentissages et de tests différentes à chaque nouvelle itération. L'objectif est de pouvoir se faire une bonne idée des performances de nos modèles sur nos jeux de données en répétant un grand nombre de fois l'apprentissage afin de faire ressortir un taux d'erreur moyen pour un modèle donné pour une jeu de données.

Ainsi comme nous nous en étions douté, l'ADL et l'ADQ fournissent d'excellents résultats sur le jeu de données Breastcancer de part la nature linéaire de la frontière entre les deux classes et la répartition relativement homogène de celles-ci malgré le fait que les matrices de covariances soient différentes selon la classe et que les classes ne soient pas de distribution gaussienne.

Ionosphere qui possédait deux classes difficilement discernables l'une de l'autre n'obtient pas de très bonnes performances avec les modèles ADL, ADQ et Baysién naïf, tandis que les performances de l'arbre de décision se révèlent assez bonne avoisinant les 10% d'erreurs seulement. Ceci est du au fait que la frontière de décision est complexe mais que comme nous disposons de suffisamment de données disponibles pour l'apprentissage les performances restent bonnes. En appliquant la PCA, nous remarquons que le classifieur Baysien Naïf obtiens d'assez bons résultats ce qui pourrait être du au fait que l'application de la PCA tend a dé-corréler certaines variables.

Les performances des modèles sur les données Sonar sont toutes relativement mauvaises particulièrement le classifieur baysien mais le taux d'erreur, avec ou sans PCA varient entre 24% et 33%. Comme nous l'avons dis précédemment, les données sont complexes et nous n'avons que peu d'observation pour faire l'apprentissage ce qui explique pourquoi nos performances restent faibles comparées à celle que l'on peut obtenir sur Breastcancer et cela même pour l'arbre de décision et la régression logistique.

En ce qui concerne les données Spambase et Spambase2 les meilleures performances sont obtenues avec la régression logistique comme nous l'avions plus ou moins prévu. En effet, la regression logistique est un modèle qui convient bien aux données binaires avec des frontières complexes ce qui est le cas ici et les jeux de données Spambase et Spambase2 possédant un grand nombre d'observations cela permet un apprentissage plus performant. L'arbre de décision obtient même d'assez bons résultats sur Spambase tandis que ces performances sont moins bonnes comparées à l'ADQ ou l'ADL sur Spambase2. Cela peut être du au fait que les arbres sont moins adaptés aux données binaires.

Données	ADL	ADQ	Logistic	Baysien Naïf	Arbre
Breastcancer	0.04629371	0.04923077	0.05790210	0.06643357	0.07104895
Breastcancer PCA	0.06048951	0.05314685	0.04825175	0.08741259	0.07797203
Ionosphere	0.15113636	0.12329545	0.17329545	0.17045455	0.09545455
Ionosphere PCA	0.1210227	0.1147727	0.1170455	0.0875000	0.1107955
Sonar	0.2740385	0.3298077	0.2923077	0.3182692	0.3028846
Sonar PCA	0.2951923	0.2480769	0.2759615	0.3298077	0.2875000
Spambase	0.12215465	0.20734144	0.08284101	0.28692441	0.10573414
Spambase PCA	0.11476977	0.17493484	0.07384883	0.15091225	0.11985230
Spambase2	0.09248480	0.09487402	0.07710686	0.12080799	0.12423979
Spambase2 PCA	0.07450043	0.09157255	0.06307559	0.11711555	0.10052129

2 Analyse de données binaires

2.1 Le modèle

2.1.1 Distribution conditionnelle de X^j étant donné Z

X^j peut prendre deux valeurs : 0 ou 1. La probabilité de $X^j = 1$ sachant que $Z = w_k$ vaut p_{kj}. Nous pouvons donc assimiler cette distribution à celle d'un test de Bernouilli de probabilité p_{kj}. Ceci nous donne la distribution suivante :

$$Pr(X^j = x_j | Z = w_k) = p_{kj}^{x_j}(1 - p_{kj})^{1-x_j} \tag{1}$$

2.1.2 Probabilité jointe du vecteur X conditionnellement à la classe w_k

Nous savons que les variables aléatoires composant le vecteur X sont indépendantes conditionnellement à la classe Z. De ce fait la probabilité jointe est le produit des probabilités de chaque variable X^j. Ainsi nous obtenons :

$$Pr(X = x | Z = w_k) = \prod_{j=1}^{p} p_{kj}^{xj}(1 - p_{kj})^{1-x_j} \tag{2}$$

2.1.3 Probabilité jointe de $Pr(X = x_i, Z = z_i)$

$$Pr(X = x_i, Z = w_k) = Pr(X = x_i | Z = w_k) Pr(Z = w_k) \tag{3}$$

Sachant que $Pr(Z = w_k) = \pi_k$ correspond à la probabilité d'appartenir à la classe k et que z_i est le vecteur de classe de x_i ; on peut donc assimiler que $Pr(Z = w_k) = Pr(Z = z_i)$.

$$Pr(X = x_i, Z = z_i) = \pi_k \prod_{j=1}^{p} p_{kj}^{x_{ij}}(1 - p_{kj})^{1-x_{ij}} \tag{4}$$

2.1.4 Vraisemblance jointe des paramètres du modèle

Les paramètres du modèles sont π_k et p_{kj}. La fonction de vraisemblance jointe à donc pour expression :

$$L(\pi_k, p_{kj}; (x_1, z_1)...(x_n, z_n)) = \prod_{i=1}^{n} \prod_{k=1}^{g} Pr(X = x_i, Z = z_i)^{z_{ik}} \tag{5}$$

$$= \prod_{i=1}^{n} \prod_{k=1}^{g} \pi_k^{z_{ik}} \prod_{j=1}^{p} p_{kj}^{z_{ik}x_{ij}}(1 - p_{kj})^{z_{ik}(1-x_{ij})} \tag{6}$$

En passant à la log-vraisemblance on obtient :

$$ln(L(\pi_k, p_{kj}; (x_1, z_1)...(x_n, z_n))) = \sum_{i=1}^{n} \sum_{k=1}^{g} z_{ik} ln(Pr(X = x_i, Z = z_i)) \tag{7}$$

$$= \sum_{i=1}^{n} \sum_{k=1}^{g} z_{ik}(ln(\pi_k) + \sum_{j=1}^{p}(x_{ij} ln(p_{kj}) + (1 - x_{ij}) ln(1 - p_{kj}))) \tag{8}$$

2.1.5 EMVs de p_{kj} et de π_k

Pour obtenir les EMVs des paramètres du modèle, nous allons devoir calculer les dérivées partielles respectives et utiliser la formulation de Lagrange pour pouvoir appliquer les conditions d'optimalités.

$$\frac{\partial ln(L(\pi_k, p_{kj}))}{\partial \pi_k} = \frac{\partial(\sum_{i=1}^{n}\sum_{k=1}^{g} z_{ik} ln(\pi_k))}{\partial \pi_k} = \frac{1}{\pi_k}\sum_{i=1}^{n} z_{ik} \tag{9}$$

En effet, ici nous dérivons par rapport à π_k dont nous fixons la classe ce qui fait disparaître la somme lors de la dérivation. Nous devons tenir compte de la contrainte $\sum_{k=1}^{g}\pi_k = 1$. Comme nous l'avons déjà vu en cours et disponible en annexe E, nous utilisons la formulation lagrangienne de ce problème d'optimisation sous contrainte qui nous permet de déduire que le multiplicateur de Lagrange, λ vaut n.

$$\frac{\partial \mathscr{L}(L(\pi_k, p_{kj}))}{\partial \pi_k} = 0 \iff \frac{1}{\pi_k}\sum_{i=1}^{n} z_{ik} = \lambda \iff \pi_k = \frac{1}{n}\sum_{i=1}^{n} z_{ik} \tag{10}$$

Ainsi nous retrouvons bien la valeur de l'estimateur du maximum de vraisemblance $\hat{\pi}_k$.

Pour obtenir l'EMV de p_{kj} nous dérivons de nouveau la fonction de vraisemblance par rapport à ce paramètre :

$$\frac{\partial ln(L(\pi_k, p_{kj}))}{\partial p_{kj}} = \sum_{i=1}^{n} z_{ik}\left(\frac{x_{ij}}{p_{kj}} + \frac{-(1-x_{ij})}{1-p_{kj}}\right) \tag{11}$$

En effet, nous dérivons sur p_{kj} donc nous fixons un k et un j pour dériver ce qui implique que nous ne conservons qu'un seul terme des sommes sur k et j après dérivation, d'où leur disparition de l'équation.

$$= \sum_{i=1}^{n} z_{ik}\left(\frac{x_{ij}(1-p_{kj}) - p_{kj} + p_{kj}x_{ij}}{p_{kj}(1-p_{kj})}\right) \tag{12}$$

Avec les conditions d'optimatilités, nous pouvons poser :

$$\frac{\partial ln(L(\pi_k, p_{kj}))}{\partial p_{kj}} = 0 \iff \sum_{i=1}^{n} z_{ik}x_{ij} - p_{kj}z_{ik}x_{ij} - p_{kj}z_{ik} + p_{kj}z_{ik}x_{ij} = 0 \tag{13}$$

$$\iff p_{kj}\sum_{i=1}^{n} z_{ik} = \sum_{i=1}^{n} z_{ik}x_{ij} \tag{14}$$

$$\iff p_{kj} = \frac{1}{\sum_{i=1}^{n} z_{ik}}\sum_{i=1}^{n} z_{ik}x_{ij} \tag{15}$$

Sachant que $\sum_{i=1}^{n} z_{ik} = n_k$ nous pouvons remplacer comme suit :

$$\iff p_{kj} = \frac{1}{n_k}\sum_{i=1}^{n} z_{ik}x_{ij} \tag{16}$$

Ce qui correspond bien à l'EMV $\hat{p}_{kj}$.

(17)

2.2 Programmation

Le code source des fonctions `binaryNBCfit` et `binaryNBCval` est disponible en annexe F.

`binaryNBCfit` : Cette fonction calcule le vecteur des probabilités a priori `pik` et le vecteur des paramètres `pkj` en utilisant un tableau individus-variables X et un vecteur d'indicateurs de classe z.

`pik` contient la proportion des individus qui appartiennent à la classe K par rapport au nombre total d'individus n. D'abord nous utilisons le vecteur z pour obtenir le nombre de classe g de notre jeu de données, après nous notons le nombre d'individus `nk` qui appartiennent à chaque classe K et nous obtenons la valeur `pik` par le calcul : $\frac{nk}{n}$.

`pkj` est le vecteur qui contient pour chaque classe K et chaque variable X^j la moyenne des valeurs. Ici nous avons utilisé la fonction `colMeans()` pour calculer les moyennes en colonnes.

`binaryNBCval` : Cette fonction évalue un ensemble de données de tests en utilisant `pik` et `pkj` que nous avons calculés dans la fonction `binaryNBCfit`. Cette fonction doit retourner le vecteur `prob` des probabilités a posteriori calculées et les prédictions de classe associée à chaque individu `pred` à chaque individu d'un jeu de données `Xtest`.

Premièrement nous avons calculé la densité de probabilité de la classe K : `fkv`, selon la formule (2) que nous avons montrée dans la partie 2.1.2, ensuite nous avons calculé la probabilité à posteriori en utilisant la formule :

$$P(Z = w_k|X = x) = \frac{(f_k(x)\pi_k)}{\pi_1 f_1(x) + \pi_2 f_2(x)} \quad (18)$$

.

Nous avons ensuite sommé les produits des `pik` de chaque classe avec la fonction de densité `fkv` qui lui correspond pour chaque individu. Le résultat du produit est stocké dans `fpi`. Il nous suffit ensuite de diviser pour chaque individu et pour chaque classe potentielle le produit `pik * fkv` par la somme de `fpi` que nous avons calculé un peu plus tot. Ceci nous permet d'obtenir pour chaque individu, sa probabilité d'appartenance à chacune des classes potentielles.

A la fin nous obtenons les prédictions `pred` de chaque individu en comparant les valeurs de probabilité à posteriori de chaque classe. Nous sélectionnons la classe de probabilité la plus grande et l'assignons comme prédiction à l'individu étudié.

Notre fonction est bien adaptée à un jeu de données ne possédant que deux classes. Si nous avions besoin de considérer un nombre de classes plus important, nous devrions apporter quelques modifications à la partie sélection de la classe lors de la prédiction. Ces modifications restent relativement simples à faire : sélectionner la classe de plus grande probabilité avec un `which.max` sur chaque ligne par exemple.

2.3 Tests

En testant nos fonctions sur le jeu de données Spambase2, nous obtenons une erreur moyenne de : 0.117505. Le boxplot des erreurs obtenues sur 20 essais est disponible en annexe H. Ce taux d'erreur n'est pas aussi faible que ceux que l'on a pu obtenir après traitement des données sur Spambase2, cependant il relativement meilleur que la plupart des modèles que nous avons pu tester sur Spambase (à l'exception de la régression logistique). Cela indique que notre modèle est assez performant surtout si on le compare aux performances du classifieur Baysien Naïf dont il s'inspire. Notre modèle est surtout capable de fonctionner sans pré-traitement de nos données. Le fait de ne pas effectuer de traitements (et particulièrement le fait de ne pas retirer de variables qui seraient trop corrélées) permet de ne pas perdre d'informations ce qui peut nous laisser penser que le modèle appris pourrait être plus efficace car il tiens compte de toutes les variables de notre modèle pour l'apprentissage et donc pour la prédiction. La construction d'un classifieur spécifique peut donc être utile dans le cas où les données sont atypiques ou bien si aucun classifieur existant n'est capable de produire de bonnes performances sur celui-ci.

Appendices

A Visualisation des données

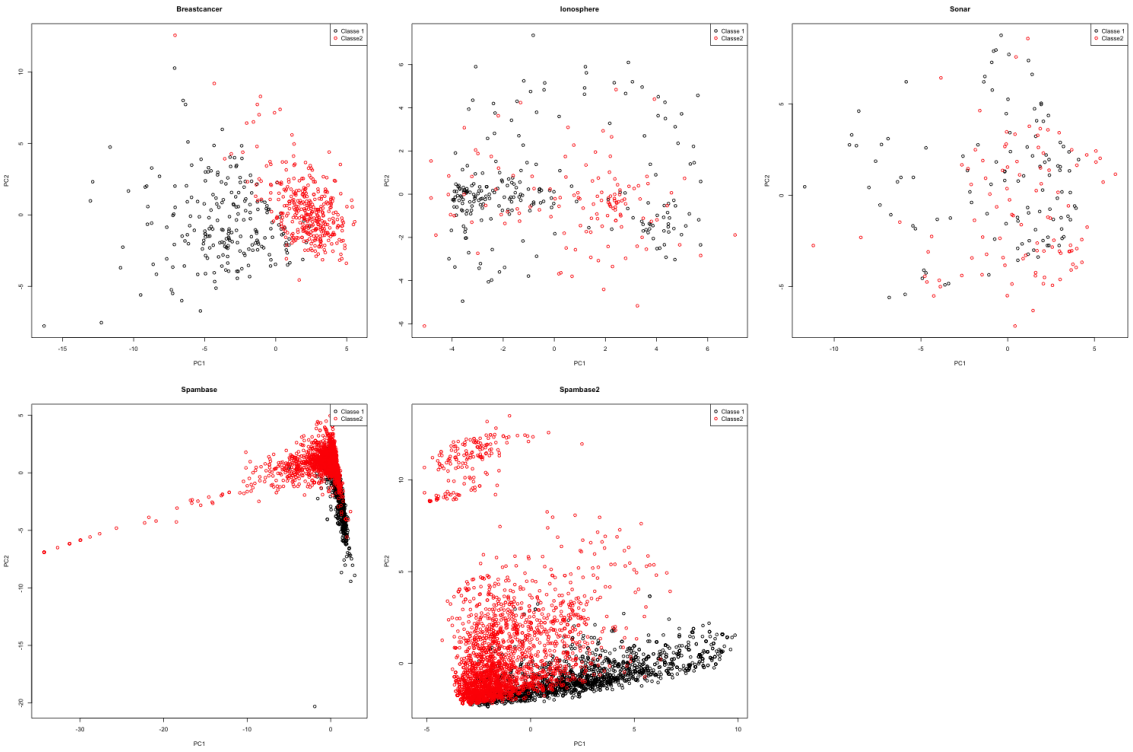

FIGURE 1 – Représentation des différents jeu de données dans le premier plan factoriel, colorés selon les classes.

B Boxplots des résultats des modèles sur les différents jeu de données

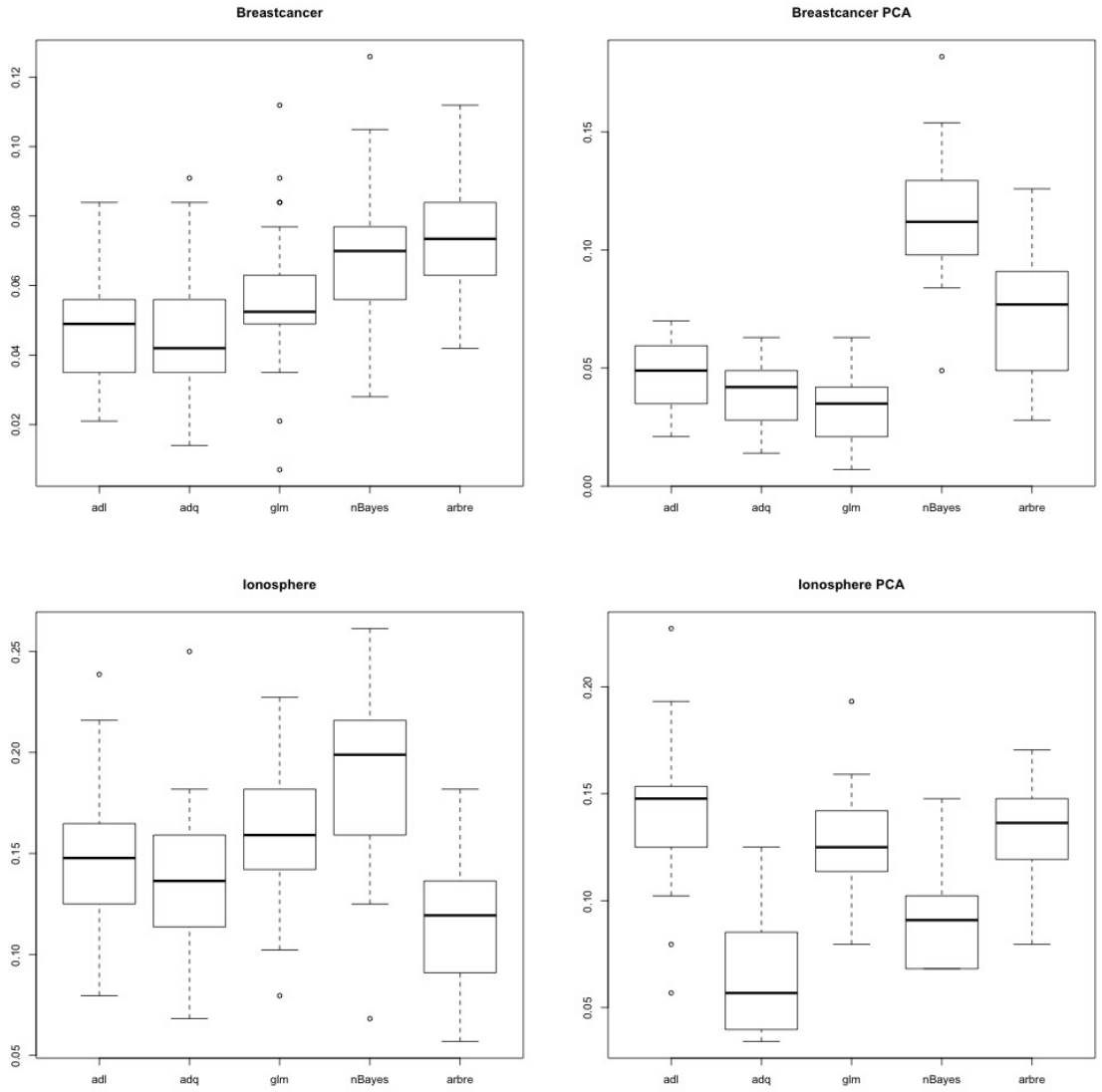

FIGURE 2 – Boxplot des taux d'erreurs sur les jeux de données Breastcancer et Ionosphere (avec et sans PCA) avec l'application des différents modèles.

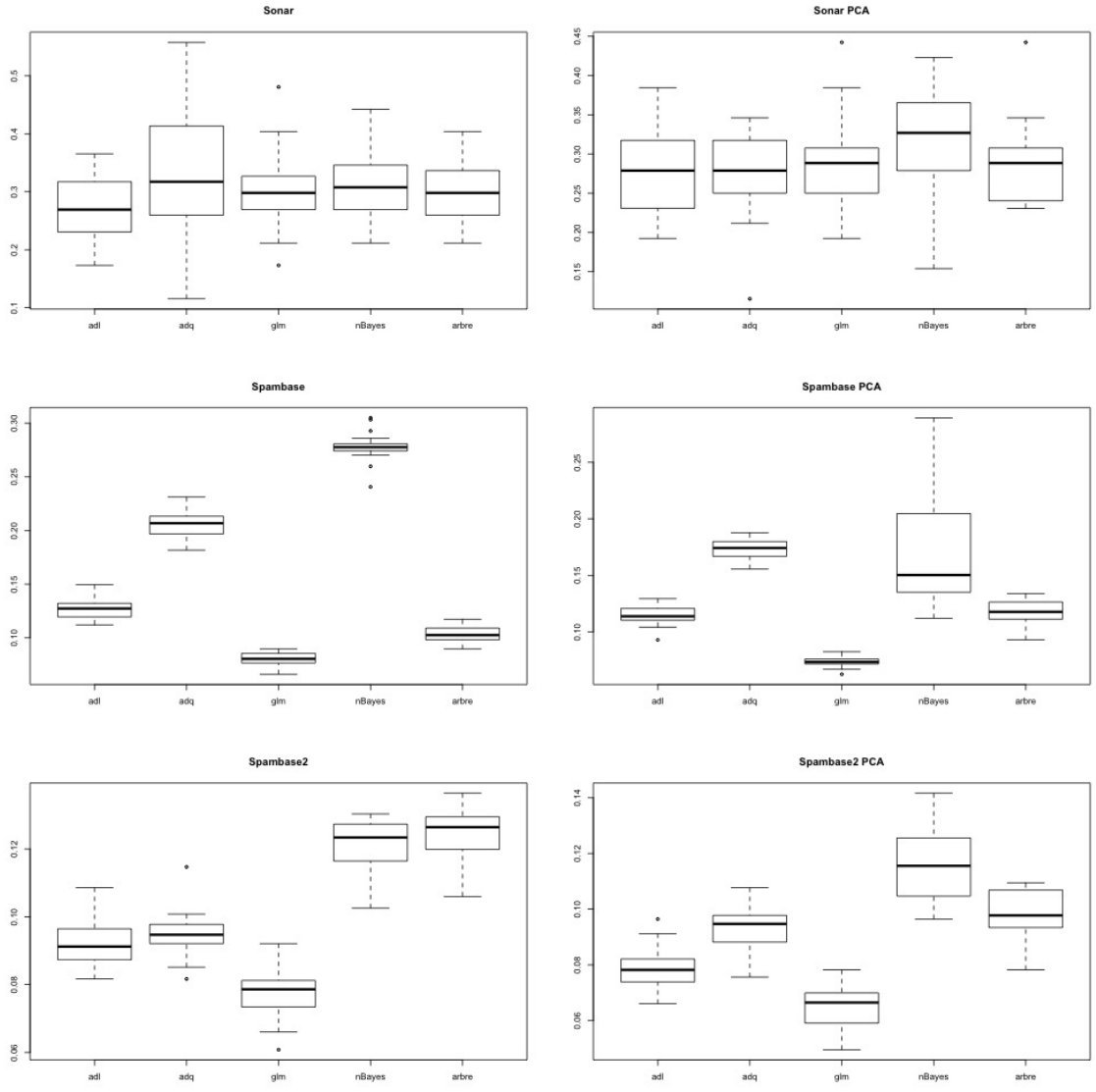

FIGURE 3 – Boxplot des taux d'erreurs sur les jeux de données Sonar, Spambase et Spambase2 (avec et sans PCA) avec l'application des différents modèles.

C Plot Breatscancer

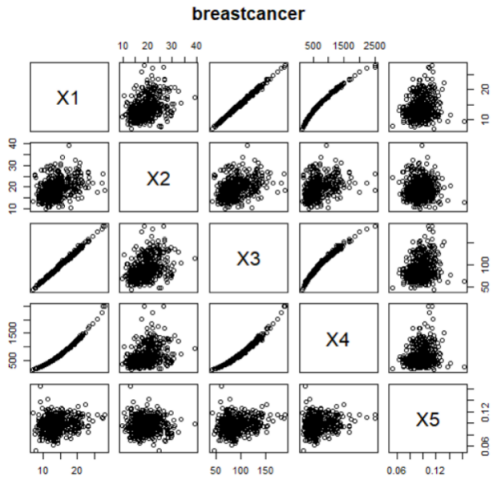

FIGURE 4 – Plot des 5 premières variables de Breastcancer les unes en fonction des autres

D Plot Spambase

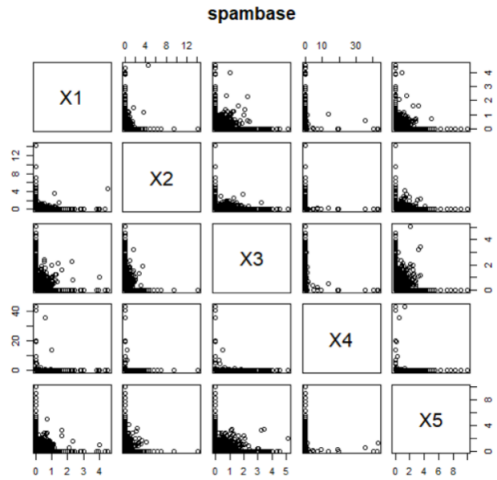

FIGURE 5 – Plot des 5 premières variables de Spambase les unes par rapport aux autres

E Démonstration $\lambda = n$

Nous devons prendre en compte la contrainte : $\sum_{k=1}^{g} \pi_k = 1$. La formulation lagrangienne de ce problème d'optimisation de $L(\pi_k, p_{kj})$ sous contrainte est la suivante :

$$\mathscr{L}(L(\pi_k, p_{kj}), \lambda) = ln(L(\pi_k, p_{kj})) - \lambda(\sum_{k=1}^{g} \pi_k - 1) \tag{19}$$

avec λ multiplicateur de Lagrange associé à notre contrainte. En appliquant les conditions d'optimalités, nous obtenons :

$$\frac{\partial \mathscr{L}(L(\pi_k, p_{kj}))}{\partial \pi_k} = 0 \iff \frac{1}{\pi_k} \sum_{i=1}^{n} z_{ik} = \lambda \iff \frac{1}{\lambda} \sum_{i=1}^{n} z_{ik} = \pi_k \tag{20}$$

$$\frac{\partial \mathscr{L}(L(\pi_k, p_{kj}))}{\partial \lambda} = 0 \iff \sum_{k=1}^{g} \pi_k = 1 \tag{21}$$

ceci permet de déduire la valeur de λ de cette manière :

$$\sum_{k=1}^{g} = 1 \iff \sum_{k=1}^{g} \frac{1}{\lambda} \sum_{i=1}^{n} z_{ik} = 1 \iff \lambda = \sum_{k=1}^{g} \sum_{i=1}^{n} z_{ik} \iff \lambda = n \tag{22}$$

F Code source

Listing 1 – Fonction binaryNBCfit

```
binaryNBCfit <- function(X, z)
{
  n <- dim(X)[1]
  p <- dim(X)[2]
  g <- max(unique(z))
  result <- NULL
  result$pik <- rep(0,g)
  result$pkj <- array(0,c(g,p))

  for( k in 1:g){
    indk <- which(z==k)
    result$pik[k] = length(indk)/n
    result$pkj[k,] <- colMeans(X[indk,])
  }
  result
}
```

Listing 2 – Fonction binaryNBCval

```
binaryNBCval <- function(pik, pkj, Xtst){
  g <- length(pik)
  p <- dim(pkj)[2]
  n <- dim(Xtst)[1]
```

```
  sumprob <- {}
  result <- NULL
  result$prob <- array(0,c(n,g))
  result$pred <- rep(0, n)
  fkv <- array(0,c(n, g))

  for (i in 1:n){
    for (k in 1:g) {
      fkv[i,k] <- prod(pkj[k,] ** Xtst[i,] * (1-pkj[k,]) ** (1 - Xtst[i,]))
    }
  }

  for ( i in 1:n){
    fpi <- {}
    for( k in 1:g){
      fpi[k] <- pik[k] * fkv[i,k]
    }
    sumprob[i] = sum(fpi)
  }

  for (i in 1:n){
    for (k in 1:g) {
      result$prob[i, k] <- (fkv[i,k] * pik[k]) / sumprob[i]
    }
  }

  for( i in 1:n) {
    if (result$prob[i,1] > result$prob[i,2]){
      result$pred[i] = 1;
    }else {
      result$pred[i] = 2;
    }
  }
  result
}
```

G Extraits des matrices de covariances

G.1 Sonar classe 1

```
sonar.class1 <- sonarX[which(sonar$Z==1),]
cov.wt(sonar.class1)
```

variables	X1	X2	X3	X4	X5
X1	7.330144e-04	8.031432e-04	7.235821e-04	7.940887e-04	6.067071e-04
X2	8.031432e-04	1.431858e-03	1.329448e-03	1.281167e-03	9.153484e-04
X3	7.235821e-04	1.329448e-03	1.937206e-03	1.938604e-03	1.510055e-03
X4	7.940887e-04	1.281167e-03	1.938604e-03	2.969894e-03	2.463621e-03
X5	6.067071e-04	9.153484e-04	1.510055e-03	2.463621e-03	3.574867e-03

G.2 Sonar classe 2

```
sonar.class2 <- sonarX[which(sonar$Z==1),]
cov.wt(sonar.class2)
```

variables	X1	X2	X3	X4	X5
X1	2.157087e-04	1.795010e-04	1.602185e-04	6.657739e-05	8.808714e-05
X2	1.795010e-04	5.765334e-04	4.853103e-04	3.464918e-04	4.051657e-04
X3	1.602185e-04	4.853103e-04	8.468425e-04	6.070136e-04	5.870584e-04
X4	6.657739e-05	3.464918e-04	6.070136e-04	9.717181e-04	9.181586e-04
X5	8.808714e-05	4.051657e-04	5.870584e-04	9.181586e-04	2.229444e-03

H Spambase2 avec binaryNBCfit

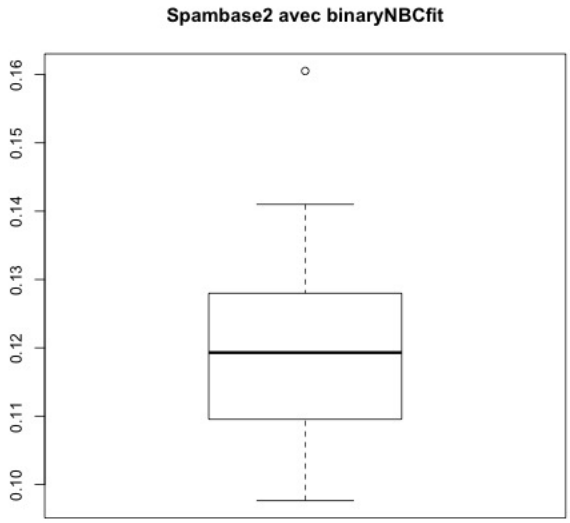

FIGURE 6 – Boxplot des erreurs pour le jeu de données Spambase2 avec le modèle implémenté binaryNBCfit.