# Rapport TP4 SY19 GR D2P1G

**Rapport de projet 1 SY19 A22 - Sun Jingwen, Ho Nhu Hoang, Asselin Cecile**

**Sommaire**



## I. **REGRESSION**

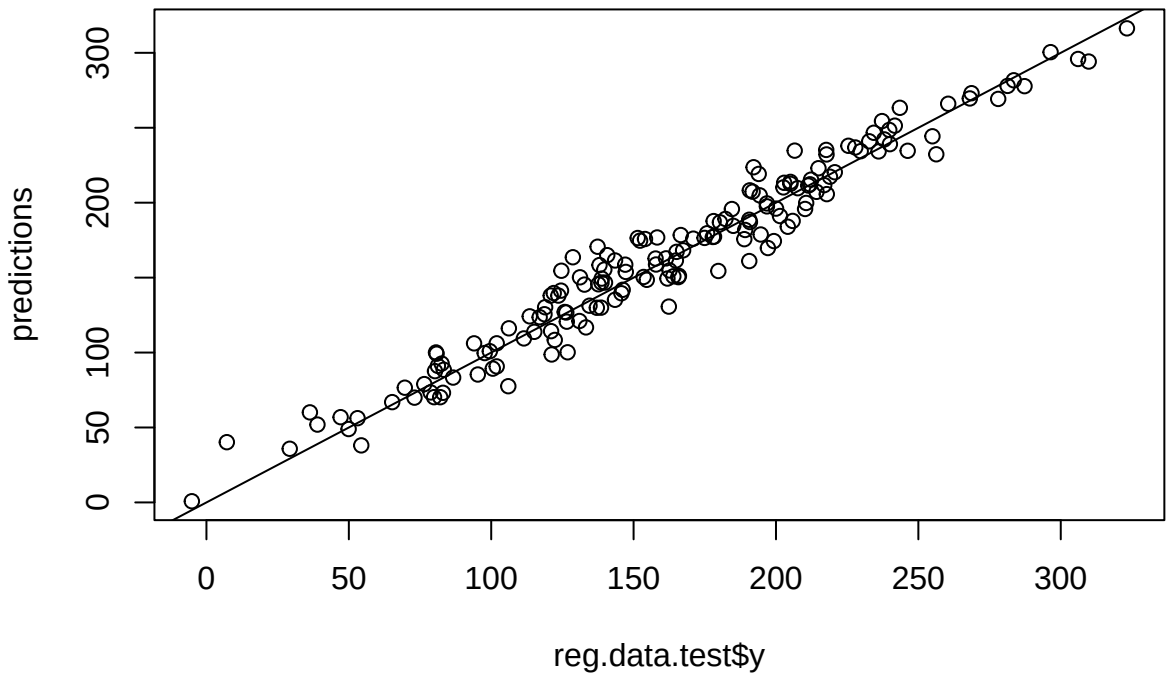
### A. Exploration et préparation des données

Nous commençons par explorer les données.

- La taille de l'échantillon est n = 500 et le nombre de prédicteurs p = 100. Notre jeu de données est donc de grande dimension (n>p), ce qui sera fondamental dans la partie modèle.
- En regardant les statistiques récapitulatives des différents prédicteurs, nous comprenons que tous ont une valeur médiane comprise entre 4,7 et 5,29 avec une valeur échelonnée de 0 à 10. Ils ont donc distributions identiques, il n'est donc pas nécessaire de mettre à l'échelle le prédicteur de données.

Dans un premier temps, nous évaluerons rapidement l'ensemble du modèle dans sa complexité, sans chercher à l'améliorer ou à le simplifier, en appliquant une simple régression linéaire après la séparation des données en train et en test.

Nous créons une fonction pour obtenir l'erreur quadratique moyenne des différents modèles que nous allons évaluer et comparer pour trouver le meilleur : MSE=mean((y_test-y_predict)^2).




```
## [1] "full.model.reg.mse = 182.384369"
```

Cette première étude montre que le modèle tel qu'il inclut tous les prédicteurs : Xi i = 1,..., 100 génère déjà une bonne régression linéaire multidimensionnelle pour prédire y. L'idée est maintenant de déterminer un meilleur modèle pour obtenir la meilleure régression possible.

**B. Comparaison des différentes méthodes utlisées**

On utilise les méthodes suivantes avec la k-fold cross-validation où k=10 :

- Stepwise selection: 1 Forward selection 2 Backward selection
- Penalized regression: 1 Ridge regression 2 Lasso regression 3 Elasticnet regression

Nous avons pensé de prime abord que la méthode de sélection du meilleur sous-ensemble de prédicteurs serait très efficace mais, pour des raisons de calcul, elle ne peut pas être appliquée ici car p est trop grand. De plus, même si la sélection du meilleur sous-ensemble était possible, peut également souffrir de problèmes statistiques lorsque p est grand : plus l'espace de recherche est grand, plus il est probable de trouver des modèles qui semblent bons sur les données d'apprentissage, même s'ils n'ont aucun pouvoir prédictif sur les données futures. Ainsi, un espace de recherche énorme peut conduire à un surapprentissage et à une variance élevée des estimations des coefficients.

Pour ces deux raisons, les méthodes pas à pas explorant un ensemble beaucoup plus restreint de modèles représentent ici de très bonnes alternatives à la meilleure sélection de sous-ensemble que nous allons donc mettre en œuvre.

De plus, après avoir vu en classe que les méthodes des plus proches voisins peuvent mal fonctionner lorsque p est grand car les plus proches voisins ont tendance à être éloignés en grandes dimensions, nous avons décidé

de ne pas les utiliser dans la cross-validation suivante en raison du grand nombre de prédicteurs disponibles ici.

En outre, la cross-validation peut également être utilisée dans un plus large éventail de tâches de sélection de modèles. Comme chaque ensemble d'apprentissage est seulement (K - 1)/K aussi grand que l'ensemble d'apprentissage d'origine, les estimations d'erreur de prédiction seront généralement biaisées vers le haut. Ce biais est minimisé lorsque K = n (LOOCV), mais cette estimation a une variance élevée, car les estimations pour chaque pli sont fortement corrélées. K = 5, compte tenu de notre grande taille d'échantillon n, fournit un bon compromis pour ce compromis biais-variance.

Passons maintenant à la sélection du modèle : nous allons passer en revue toutes les méthodes vues en cours pour trouver les meilleurs modèles et les tester pour voir s'ils « correspondent » à notre jeu de données. Pour chaque méthode de déduction du modèle optimisé :
- Nous allons d'abord estimer les paramètres de réglage du modèle associé à la méthode actuelle en utilisant tous les plis sauf le pli k.
- A partir de là, nous obtiendrons les meilleurs paramètres de réglage associés à la méthode actuelle.
- Enfin, nous allons réestimer le modèle en utilisant tous les plis sauf le pli k, et nous l'évaluons avec le pli k en calculant son MSE, que nous stockons dans la table des cv_errors correspondant à la méthode courante.

Nous sommes maintenant en mesure de comparer les résultats des performances de chaque méthode.

```
## [1] forward step regression model MSE: 186.442
```

```
## [1] backward step regression MSE: 186.415
```

```
## [1] elastic net model MSE: 176.645
```

```
## [1] lasso net model MSE: 178.607
```

```
## [1] ridge model MSE: 189.728
```

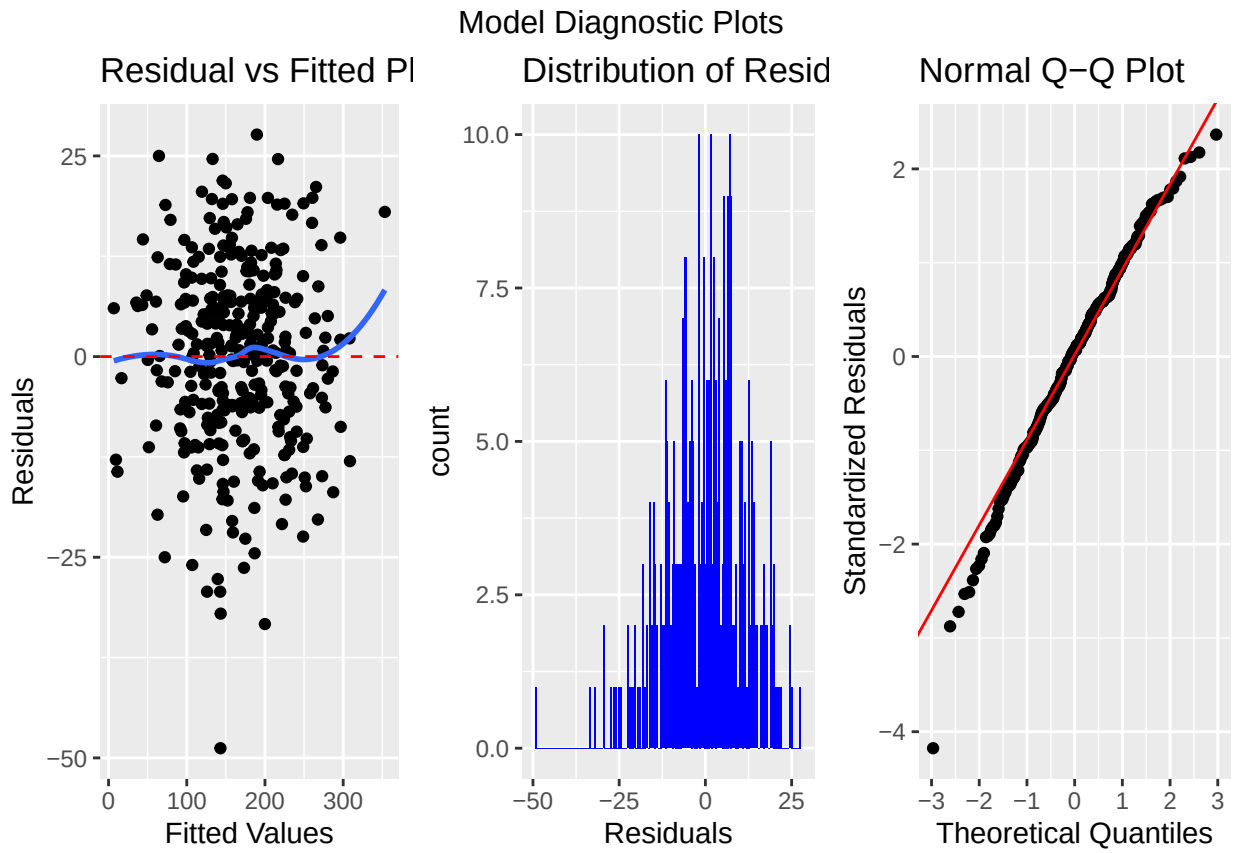
```
## [1] full model regression MSE: 184.809
```

### C. Analyse des résultats

En ce qui concerne la sélection du meilleur modèle, nous avons vu que le meilleur modèle est défini comme le modèle avec l'erreur de prédiction la plus faible (RMSE) et, par conséquent, la MSE la plus faible. Ainsi, le k-CV, une méthode "d'estimation d'erreur directe", a un avantage sur l'AIC, le BIC et le R2 ajusté vus en classe, en ce qu'il fournit une estimation directe de l'erreur de prédiction.

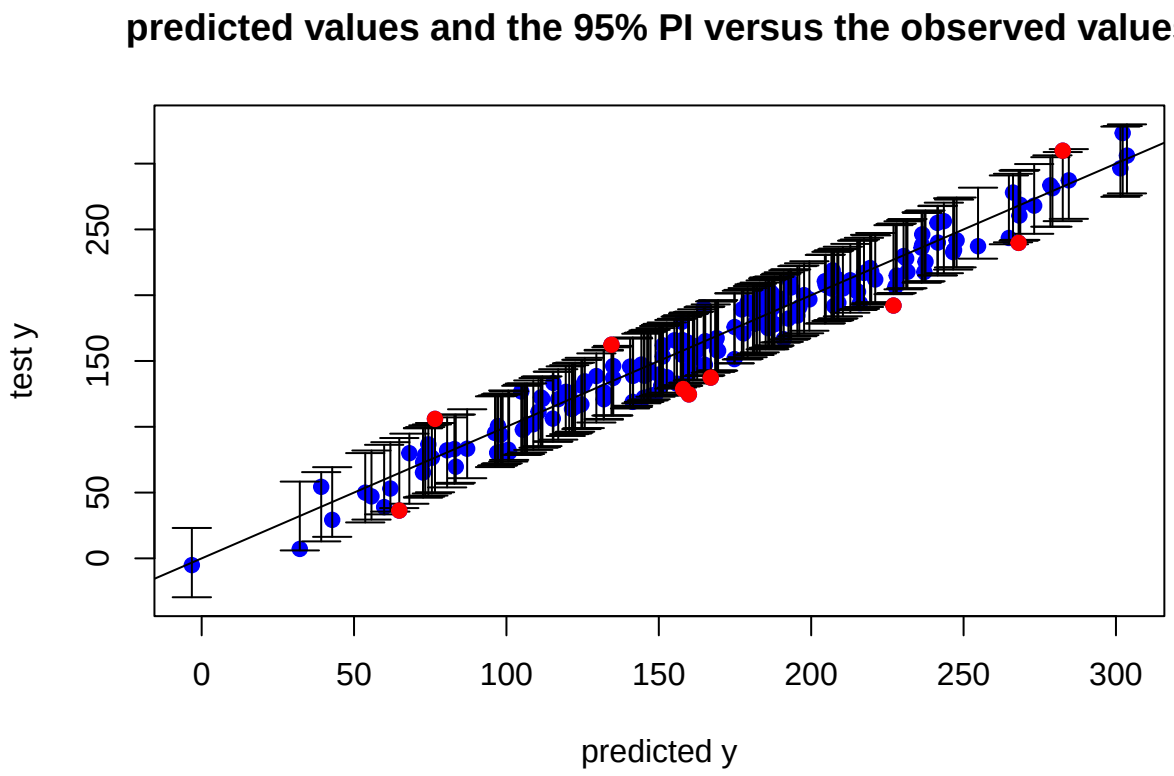
Nous en déduisons que parmi tous les travaux que nous avons effectués, la sélection pas à pas précédente donne le modèle avec le meilleur RMSE en toutes circonstances. Ce modèle pas à pas devient alors notre meilleur modèle.

```
# Best model
forward.step.model <- train(y ~., data = reg.data.train, method = "leapForward",
                tuneGrid = data.frame(nvmax = 1:100), trControl = train.control)

step.predictor.names <- names(coef(forward.step.model$finalModel,
  forward.step.model$bestTune$nvmax))[1:forward.step.model$bestTune$nvmax+1]

step.model.linreg = lm(paste("y","~", paste(step.predictor.names, collapse=" + ") ),
                       data = reg.data.train)

ypred <- predict(step.model.linreg,newdata=reg.data.test)
```




```
## [1] Best model MSE: 177.581
```

Le premier graphique représente les résidus par rapport aux valeurs prédites pour l'ensemble de données. La ligne bleue est un ajustement en douceur aux résidus, destiné à créer une tendance. Le deuxième graphique détermine si les résidus sont autour de zéro pour la plage de valeurs ajustées. Le dernier graphique (Q-Q des résidus) montre si les résidus suivent une distribution normale.




Pour résumer,

- summary(step.model.linreg) nous montre que la plupart de nos prédicteurs sélectionnés sont significatifs et que, selon la p-value (=2.2e-16 <0.05 => nous rejetons le H0 hypothèse que tous les coefficients bêta j sont égaux à 0), notre modèle est globalement significatif pour expliquer la réponse variable y.
- Les diagrammes de diagnostic du modèle ci-dessus montrent que le modèle est passable. Il y a une bonne dispersion des résidus autour de zéro pour la plage des valeurs ajustées (la valeur moyenne des résidus est, en fait, nulle). De plus, le graphique Q-Q des résidus et l'histogramme montrent une distribution plutôt normale.
- Nous observons que les intervalles de prédiction sont assez larges car l'intervalle de prédiction 0 prend en compte l'incertitude autour de lui (la variable aléatoire) et la prédiction moyenne. De plus, nous pouvons voir très peu de valeurs observées en dehors de l'intervalle de prédiction (en point rouge).
- Le pourcentage que la nouvelle observation se trouve dans l'intervalle de prédiction est : P * 100.

Nous pouvons donc conclure que notre meilleur modèle est passable. Cependant en le testant sur le jeu de données de test du site de l'UV, nous nous sommes rendu compte qu'il overfittait les dpnnées. Nous sommes donc repassés à un modèle de regression linéaire plus simple (prenant en compte tous les prédicteurs). Cela nous a effectivement permis de faire baisser la MSE de 177 à 157.

## II. CLASSIFICATION

### A. Exploration et préparation des données

En explorant les données, nous observons que les colonnes de X1 à X45 sont des prédicteurs quantitatifs et les colonnes de X46 à X50 sont des prédicteurs qualitatifs. La dernière colonne y représente les classes à

prédire (3 classes différentes). Nous déclarons cette colonne comme facteur contrairement aux autres colonnes qualitatives que nous comptons prendre en compte dans nos fonctions de dicrimination.

Nous avons ensuite cherché à analyser les éventuels liens et dépendances entre prédicteurs. En raison de la grande taille des données, il n'etait pas possible de visualiser les dépendances entre toutes les combinaisons possibles en même temps, parmi toutes les variables existantes.

Nous utilisons donc la correlation de Pearson pour identifier s'il existe une liaison linéaire ou de rangement afin de diminuer le nombre de prédicteurs, d'améliorer la précision du modèle et de réduire le temps d'exécution. N'ayant pas identifié de corrélations saillantes avec Pearson, nous avons ensuite tenté de réduire le nombre de variables utiles en les transformant avec une PCA. Cette PCA ne nous a pas permis non plus de réduire la dimension de notre problème de regression.

Nous avons donc finalement décidé de commencer à comparer nos modèles en prenant en compte tous les prédicteurs, et reporter à plus tard une eventuelle selection des variables. Avant de nous lancer dans les différentes méthodes de classification nous séparons enfin les données en test set (1/3 des données) et en train set (2/3 des données). Pour que les résultats soient reproductibles, une seed est fixée (à 1729).

**B. Choix d'un modèle de classification et optimisation de ce modèle**

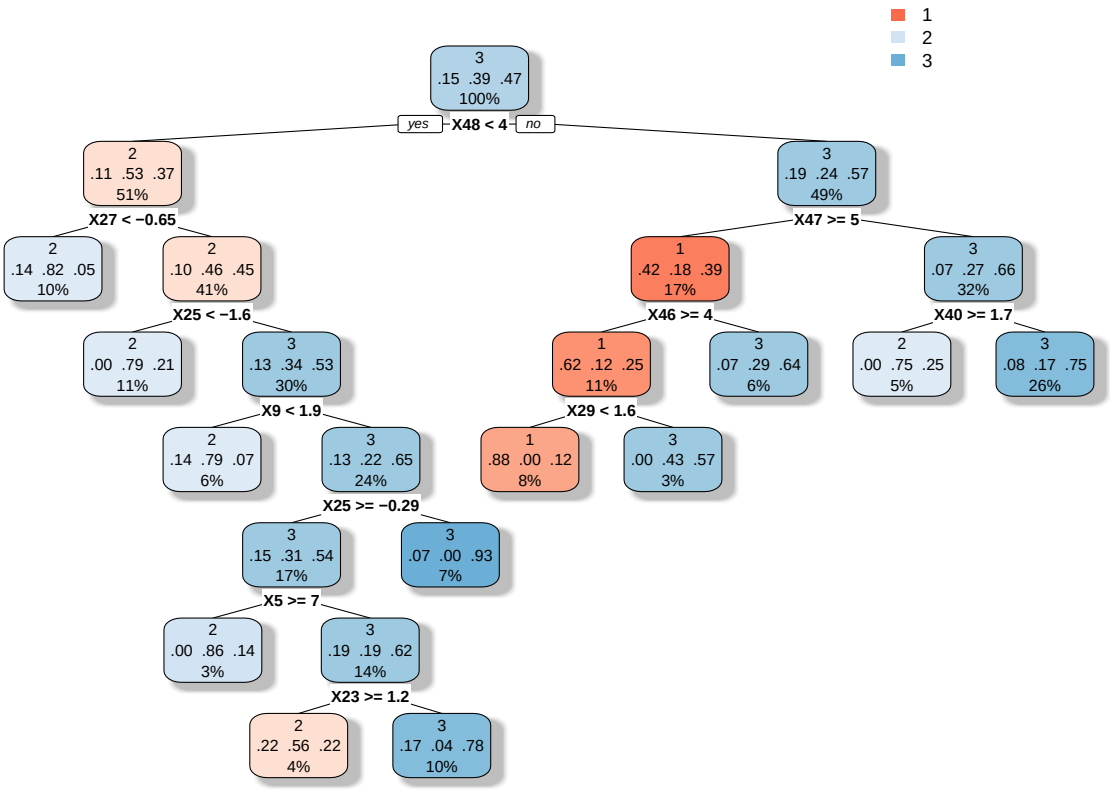
Nous avons testé 7 classifieurs différents : Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Regularized Discriminant Analysis (RDA), Naive Bayes (NB), Logistic Regression, Classification / Decision Tree et K-Nearest Neighbor (KNN). Les fonctions utilisées sont respectivement : lda, qda, rda, naive_bayes, multinom, rpart et knn.

Le but était de comparer les performances obtenues avec ces différentes méthodes, d'optimiser les différents modèles et de les stabiliser avec de la cross-validation notamment. Nous avons ensuite comparé leurs performances avec plusieurs outils : comparaison du taux d'erreur moyenné sur les K-fold de la cross validation, et courbes ROC multinomiales.

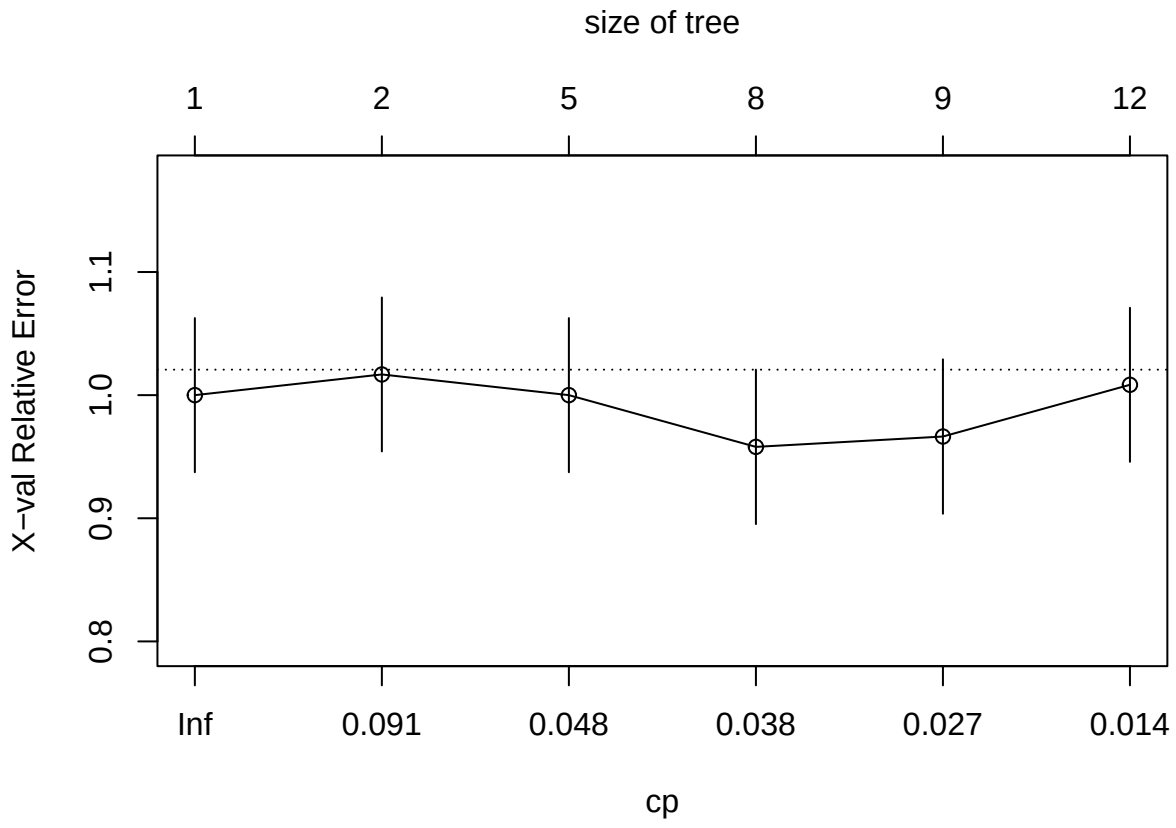
1. **Tests de différents classifieurs : exemple des arbres et de la Regularized Discriminant Analysis**

Voici des extraits du code utilisé pour la méthode des arbres de décision :

```
tree <- rpart(y~., data = classif.train, method="class",subset=train, parms = list(split = 'gini'))
rpart.plot(tree, box.palette="RdBu", shadow.col="gray",fallen.leaves=FALSE)
```


1
2
3
3
.15 .39 .47
100%
yes
X48 < 4
no
2
.11 .53 .37
51%
3
.19 .24 .57
49%
X27 < −0.65
X47 >= 5
2
.14 .82 .05
10%
2
.10 .46 .45
41%
1
.42 .18 .39
17%
3
.07 .27 .66
32%
X25 < −1.6
X46 >= 4
X40 >= 1.7
2
.00 .79 .21
11%
3
.13 .34 .53
30%
1
.62 .12 .25
11%
3
.07 .29 .64
6%
2
.00 .75 .25
5%
3
.08 .17 .75
26%
X9 < 1.9
X29 < 1.6
2
.14 .79 .07
6%
3
.13 .22 .65
24%
1
.88 .00 .12
8%
3
.00 .43 .57
3%
X25 >= −0.29
3
.15 .31 .54
17%
3
.07 .00 .93
7%
X5 >= 7
2
.00 .86 .14
3%
3
.19 .19 .62
14%
X23 >= 1.2
2
.22 .56 .22
4%
3
.17 .04 .78
10%

```
plotcp(tree)
```




```
idx<-which.min(tree$cptable[,"xerror"])
cp<-tree$cptable[idx,"CP"]
pruned_tree<-prune(tree,cp=cp)
# rpart.plot(pruned_tree, box.palette="RdBu", shadow.col="gray", fallen.leaves=FALSE)
pred.tree<-predict(pruned_tree,newdata=classif.test,type='class')
matrix.pred.tree<-table(classif.test$y,pred.tree)
err.tree<-1-mean(classif.test$y==pred.tree)
```

La fonction plotcp() peux tracer de l'erreur de validation croisée en fonction du paramètre de complexité. Nous retirons la meilleure complexité grâce à tree$cptable.

Les arbres de décision étaient utiles notamment au début de notre analyse, car ils nous permettaient d'observer l'importance de chaque variable grâce à leurs forte interpretabilité. Nous avons ensuite tenté d'optimiser leurs performances avec du bagging et des Random Forest, mais les performances restaient inférieures à celles obtenues avec d'autres classifieurs (Naive Bayes, QDA et RDA notamment), nous n'avons donc pas retenu cette méthode.

Voici des extraits du code utilisé pour la Regularized Discriminant Analysis :

```
fit.rda <- rda(y~.,data=classif.train,scale=FALSE)
pred.rda <- predict(fit.rda,newdata=classif.test)
err.rda <- mean(classif.test$y != pred.rda$class)
```

2. **Comparaison des performances des modèles : courbes ROC, et tableau récapitulatif des performances sur la Cross-Validation**

Nous pouvons calculer le taux d'erreur empirique en deux manière, prenons LDA comme exemple:

```
# moyen1
err.lda <- mean(pred.classif.lda$class!=classif.test$y)
# moyen2 en utilisant la matrice de confusion
matrix.conf.lda <- table(classif.test$y, pred.classif.lda$class)
err.lda <- 1-sum(diag(matrix.conf.lda))/nb.test
```

Nous obtenons ainsi le taux d'erreur de chaque méthode.

```
## [1] "Taux d'erreur de LDA : 0.419162"

## [1] "Taux d'erreur de QDA : 0.371257"

## [1] "Taux d'erreur de RDA : 0.353293"

## [1] "Taux d'erreur de Naive Bayes : 0.305389"

## [1] "Taux d'erreur de classification/decision trees : 0.604790"

## [1] "Taux d'erreur de multinomial logistic regression : 0.395210"
```

Nous constatons que QDA, RDA et Naive Bayes ont les meilleures performances.

Ensuite, nous affichons les courbes ROC adaptées à un problème de classification multinomiale en important la library "pROC".

Plus la surface sous la courbe est grande, plus la méthode est considérée "performante". On affiche toutes les courbes dans la même figure pour mieux comparer.

Nous pouvons donc résumer que la performance des modèles dans ce cas-là est NB > LDA > RDA > Logistic Regression > Decision Tree > QDA.

Ensuite, nous avons voulu régulariser nos performances avec de la Cross-Validation. Nous avons choisi d'opérer avec 10 folds.

En comparant le taux d'erreur moyen obtenu sur les 10 folds pour chaque modèle, nous en venons à la conclusion que RDA semble le meilleur modèle (compromis entre un taux d'erreur faible, et un modèle qui reste relativement peu complexe). Nous n'avons pas retenu RDA n'ayant pu vérifier de manière fiable l'hypothèse naive d'indépendance entre predicteurs, et car avec la validation croisée il nous semble qu'en moyenne cette methode reste moins performante que la RDA.

**C. Analyse des résultats**

Nous avons enfin effectué le test de Mc Nemar pour évaluer la significativité des ecarts de performance. L'écart de performance entre la LDA et la QDA est jugé non significatif, alors qu'entre la LDA et la RDA cet écart est significatif. Cela renforce notre choix pour la méthode RDA qui est celle que nous avons finalement choisi.

Pour l'apprentissage final du modèle de RDA nous avons procédé par validation croisée sur l'ensemble des données.

```
# apprentissage du modèle : on cherche les meilleurs paramètres lambda et gamma pour la RDA
cv_5_grid <- trainControl(method = "cv", number = 5)
fit_rda_grid <- train(y ~ ., data = classif, method = "rda", trControl = cv_5_grid)
# The final values used for the model were gamma = 0 and lambda = 0.5

# création du modèle avec les paramètres appris
model.cls <- rda(y ~ ., data = classif, gamma = 0, lambda = 0.5)
```

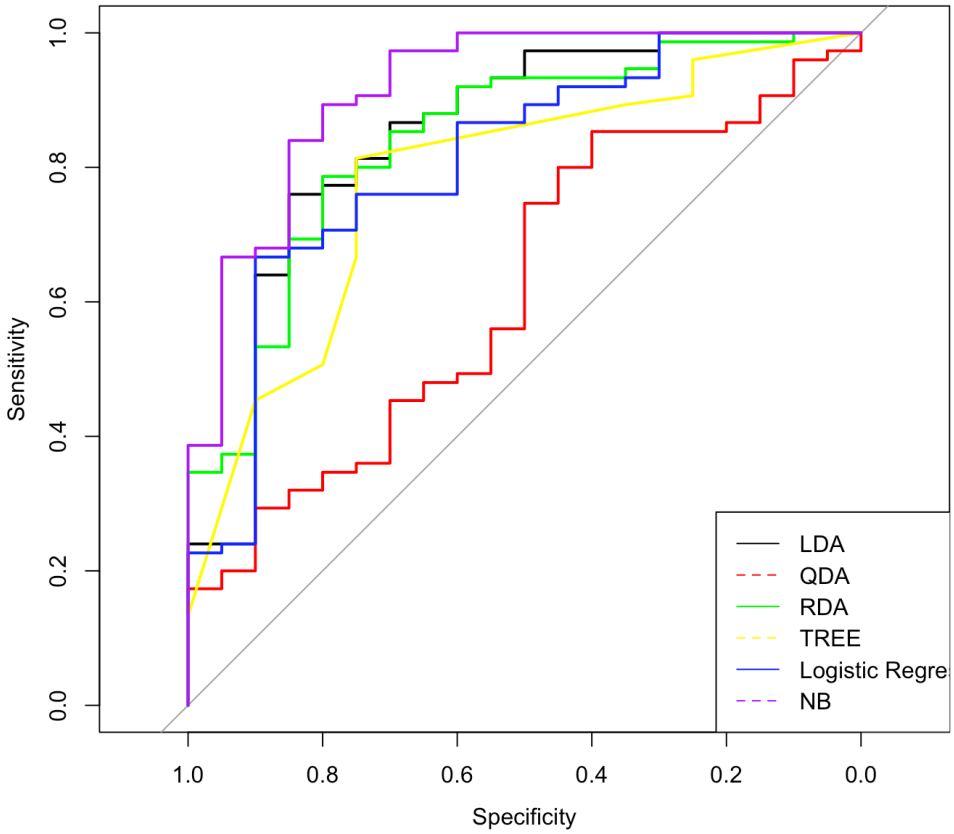



Figure 1: Comparaison des courbes ROC

| | method_name | mean_cv_errors | IS_BEST |
|---|---|---|---|
| 1 | LDA | 0.4423451 | FALSE |
| 2 | QDA | 0.3283477 | FALSE |
| 3 | RDA | 0.3134081 | TRUE |
| 4 | NB | 0.3660705 | FALSE |
| 5 | LR | 0.4300134 | FALSE |
| 6 | Trees | 0.5109985 | FALSE |

Figure 2: Comparaison des moyennes du taux d'erreur sur la validation croisée

En testant nos performances sur le site de test de l'UV nous observons une accuracy de 0.71 sur les données de test. Ce resultat au dessus de la baseline est satisfaisant car on observe une amelioration des performances entre la classification opérée sur le set de validation (classifieur entrainé avec le set d'apprentissage) et celle opérée sur le dataset de test auquel nous n'avons pas accès (classifieur entrainé sur toutes les données disponible). Nous avons donc évité l'ecueil principal de l'overfitting.