Projet d'apprentissage

Ugo Devoille & Zoé Joubert

Octobre 2020

Contents

1 Introduction

Le but de ce projet va être de mettre en oeuvre les différentes méthodes vues dans le cours d'apprentissage en pratique afin de faire des prévisions, et aussi de voir quelles sont les meilleures méthodes. Le projet va être divisé en deux parties: Une partie étudiera le prix d'une voiture Audi en fonction de différents critères, et l'autre partie étudiera le risque de développer une maladie chez un patient en fonction de ses données.

2 Etude et prévision du prix des voitures Audi

La première partie de ce projet aura pour but d'étudier la base de données qui donne des informations sur un certain nombre de voitures Audi. Cette étude sera un peu moins poussée que la classification qui contient plus de ressources méthodologiques. L'objectif principal va être de pouvoir expliquer le prix que coûte une voiture en fonction de toutes les autres variables.

Nous pouvons ouvrir notre base de données et effectuer nos premières manipulations dessus. On a constaté en faisant quelques modèles que les 10668 individus posaient problème dans le sens où chaque compilation prenait beaucoup trop de temps. On choisit pour ne pas trop se poser de problème de restreindre cette base de données à 1000 individus.

```
data <- read.csv(file="audi.csv", header=T, sep=",")
set.seed(7777777)
s <- sample(10668, 1000)
data <- data[s, ]
p <- ncol(data)-1
summary(data)
```

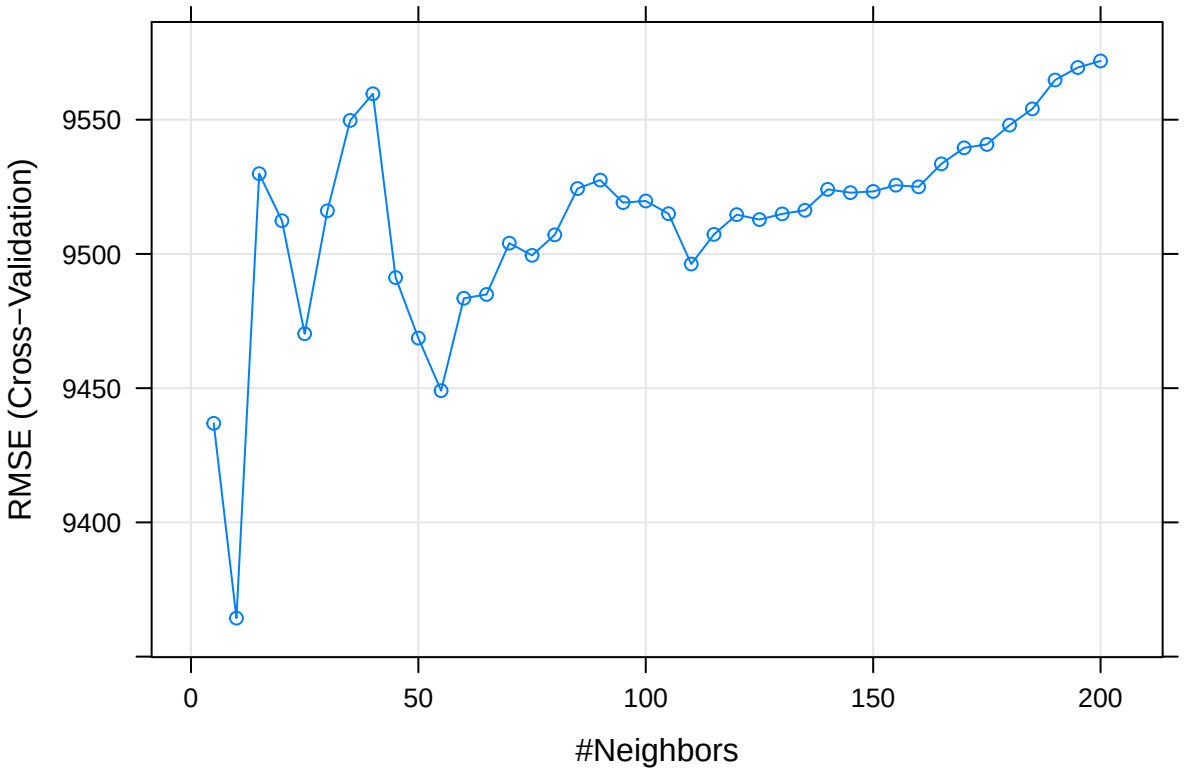
```
##     model                year          price        transmission
##  Length:1000        Min.   :1998   Min.   :  2895   Length:1000
##  Class :character   1st Qu.:2016   1st Qu.: 14998   Class :character
##  Mode  :character   Median :2017   Median : 19999   Mode  :character
##                     Mean   :2017   Mean   : 22635
##                     3rd Qu.:2019   3rd Qu.: 27743
##                     Max.   :2020   Max.   :137500
##     mileage         fuelType              tax             mpg
##  Min.   :    10   Length:1000        Min.   :  0.0   Min.   : 21.40
##  1st Qu.:  6000   Class :character   1st Qu.:145.0   1st Qu.: 40.90
##  Median : 20849   Mode  :character   Median :145.0   Median : 49.60
##  Mean   : 26289                      Mean   :129.6   Mean   : 50.52
##  3rd Qu.: 37969                      3rd Qu.:150.0   3rd Qu.: 57.60
##  Max.   :136000                      Max.   :570.0   Max.   :176.60
##    engineSize
##  Min.   :0.000
##  1st Qu.:1.500
##  Median :2.000
##  Mean   :1.946
##  3rd Qu.:2.000
##  Max.   :5.200
```

On peut constater sur la table qu'on voit une grande variété de modèles de voitures, avec une dominance de modèles A3. On observe aussi les variables d'année, le type du boitier de vitesse, le nombre de km déjà parcourus, le type d'essence, la taxe routière, le nombre de miles par gallon (mpg), et la taille (en litres) du moteur. Il ne reste que la variable prix qui va donc être celle qu'on va tenter d'expliquer.

2.1 Algorithme des K plus proches voisins

Afin de prédire notre variable, nous allons dans un premier temps utiliser l'algorithme des K plus proches voisins en utilisant la validation croisée à 5 couches. Le nombre de voisins est à optimiser afin de minimiser le RMSE (erreur quadratique moyenne). Ainsi on le fait varier de 5 à 200 avec un pas de 5, et on va regarder grâce à la courbe vers quel K on s'approche du minimum.

```
set.seed(7777777)
ctrl <- trainControl(method="cv", number=5)
param <- expand.grid(k=seq(5,200,5))
fit_knn <- train(price~., data, method="knn", trControl=ctrl, tuneLength=100, tuneGrid=param)
plot(fit_knn)
```



On remarque que le RMSE est minimal quand k est vers 10. Ainsi on va refaire le même procédé avec K qui varie de 50 à 70, et en trouvant le K optimal on pourra obtenir le meilleur RMSE, ainsi que les meilleurs critères R^2 et MAE.

```
set.seed(7777777)
param <- expand.grid(k=seq(5,15))
fit_knn <- train(price~., data, method="knn", trControl=ctrl, tuneLength=100, tuneGrid=param)
print(fit_knn)
```

```
## k-Nearest Neighbors
##
## 1000 samples
##    8 predictor
```

```
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 800, 800, 800, 800, 800
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    5  9436.909  0.3628322  6242.673
##    6  9346.761  0.3716671  6188.099
##    7  9318.184  0.3725008  6162.671
##    8  9339.422  0.3695317  6110.179
##    9  9332.016  0.3688075  6106.973
##   10  9364.333  0.3649725  6145.597
##   11  9394.304  0.3599980  6151.924
##   12  9423.032  0.3558120  6163.825
##   13  9454.090  0.3518593  6159.622
##   14  9463.872  0.3493967  6168.271
##   15  9529.893  0.3418223  6196.752
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 7.
```

On constate que le RMSE est minimal quand K=7. On obtient donc une erreur quadratique moyenne de 9318.184, un R^2 de 37.25%, et un critère MAE de 6162.671.

2.2 Méthode du bagging et Random Forest

Dans cette partie, nous n'utiliserons pas le package caret pour coder avec des méthodes plus classiques. Plus en général, la représentation des arbres avec les packages party ne seront pas affichés dans cette partie car trop peu intéressant sur la lisibilité dans le cas d'une régression.

Afin de faire ces méthodes, nous allons dans un premier temps diviser notre base de données en 2 parties; un échantillon d'apprentissage, constituant les 3/4 des individus, et un échantillon test qui contient le quart restant.

```
set.seed(7777777)
train <- sample(1:nrow(data), 0.75*nrow(data))
test <- (1:nrow(data))[-train]
data_test <- data[test, "price"]
```

On peut ensuite passer à une seconde méthode en utilisant cette fois ci la grand famille des arbres de régression. Le concept est le même que les arbres de classification, mais les sorties à la fin de chaque branche correspond cette fois à un boxplot, nous permettant de donner une estimation de la variable à expliquer.

Dans un premier temps, on utilise la méthode du bagging, où le nombre de variables qu'on utilise dans chaque arbre est de m=p="Toutes les variables explicatives".

```
set.seed(7777777)
fit_bag <- randomForest(price~., data, subset=train, mtry=p, importance=T)
fit_bag
```

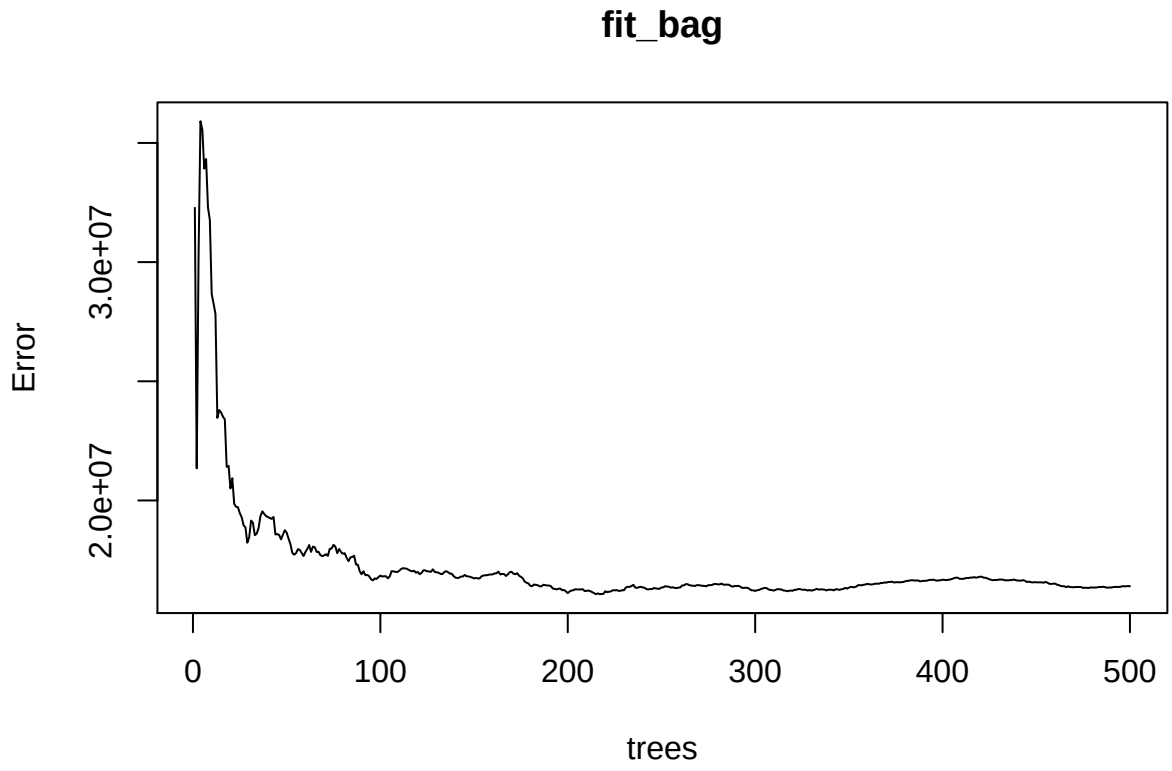
```
##
## Call:
```

```
##  randomForest(formula = price ~ ., data = data, mtry = p, importance = T,      subset = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 8
##
##           Mean of squared residuals: 16398244
##                     % Var explained: 88.57
```

On est sûrs de ne pas avoir de sur-apprentissage concernant le nombre d'arbres utilisés par définition de la méthode. On constate ici une erreur quadratique moyenne de 19589551, et on remarque que les variables expliquent 86.34% de la variance. Cette erreur est appliquée sur l'échantillon d'apprentissage, elle n'a donc pas énormément de sens ici.

De plus, le graphe ci-dessous confirme qu'on a bien convergence de l'erreur lorsque le nombre d'arbres augmente; le sur-apprentissage sur cette variable est impossible.

```
plot(fit_bag)
```



Nous pouvons enfin tester ce modèle sur l'échantillon test.

```
yhat_bag <- predict(fit_bag, newdata=data[test,])
test_err_bag <- mean((yhat_bag-data_test)^2)
test_err_bag
```

```
## [1] 11733954
```

L'erreur moyenne quadratique qui ressort est de 10463341. Elle est bien évidemment plus faible et c'est le résultat qu'on va chercher à comparer avec d'autres modèles.

On peut généraliser la méthode du Bagging à celle des Random Forest, ou cette fois ci le nombre de variables utilisées pour la création de chaque arbre est de m inférieur ou égal é p. On fait le choix de laisser le mtry par défaut cette fois ci, càd partEnt(8/3)=2. Il est aussi possible de faire varier de 1 à 8 ce nombre et trouver le meilleur par validation croisée, mais on préfère ici présenter un éventail des différentes méthodes utilisables.

```
set.seed(7777777)
fit_rf <- randomForest(price~., data, subset=train, importance=T)
fit_rf
```

```
##
## Call:
##  randomForest(formula = price ~ ., data = data, importance = T,      subset = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##           Mean of squared residuals: 17075900
##                     % Var explained: 88.1
```

Pour m=2, on obtient une erreur myoyenne quadratique de 19433334 et les variables expliquent 86.45% de la variance. L'erreur moyenne quadratique a légerement augmenté, mais il est plus important de voir comment ce résultat a varié sur l'échantillon test...

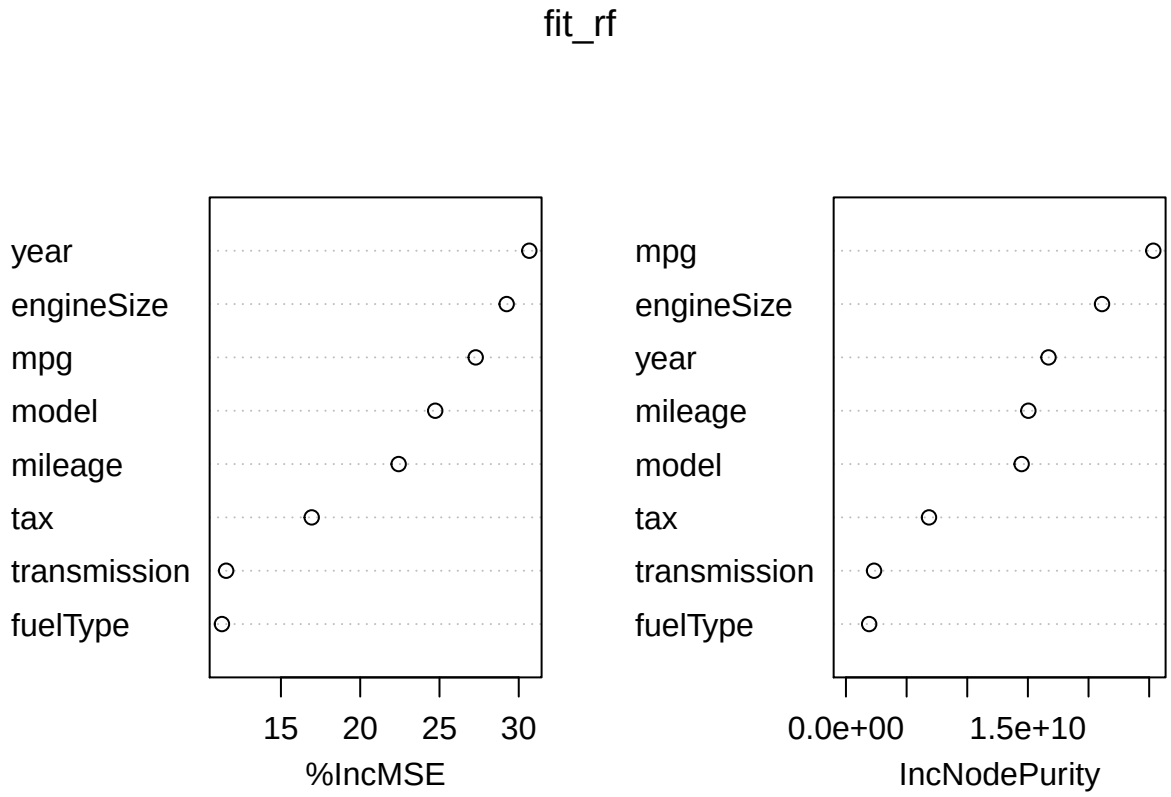
```
yhat_rf <- predict(fit_rf, newdata=data[test,])
test_err_rf <- mean((yhat_rf-data_test)^2)
test_err_rf
```

```
## [1] 9852741
```

On constate (sans surprise) que l'erreur moyenne quadratique est tombée à 8764918 sur l'échantillon test. On conclut ainsi que ce modèle est meilleur que le précédent, et que la méthode de bagging faisait du sur-apprentissage par rapport à la variable mtry.

On peut ensuite regarder quelles sont les variables les plus influentes dans notre modèle.

```
varImpPlot(fit_rf)
```

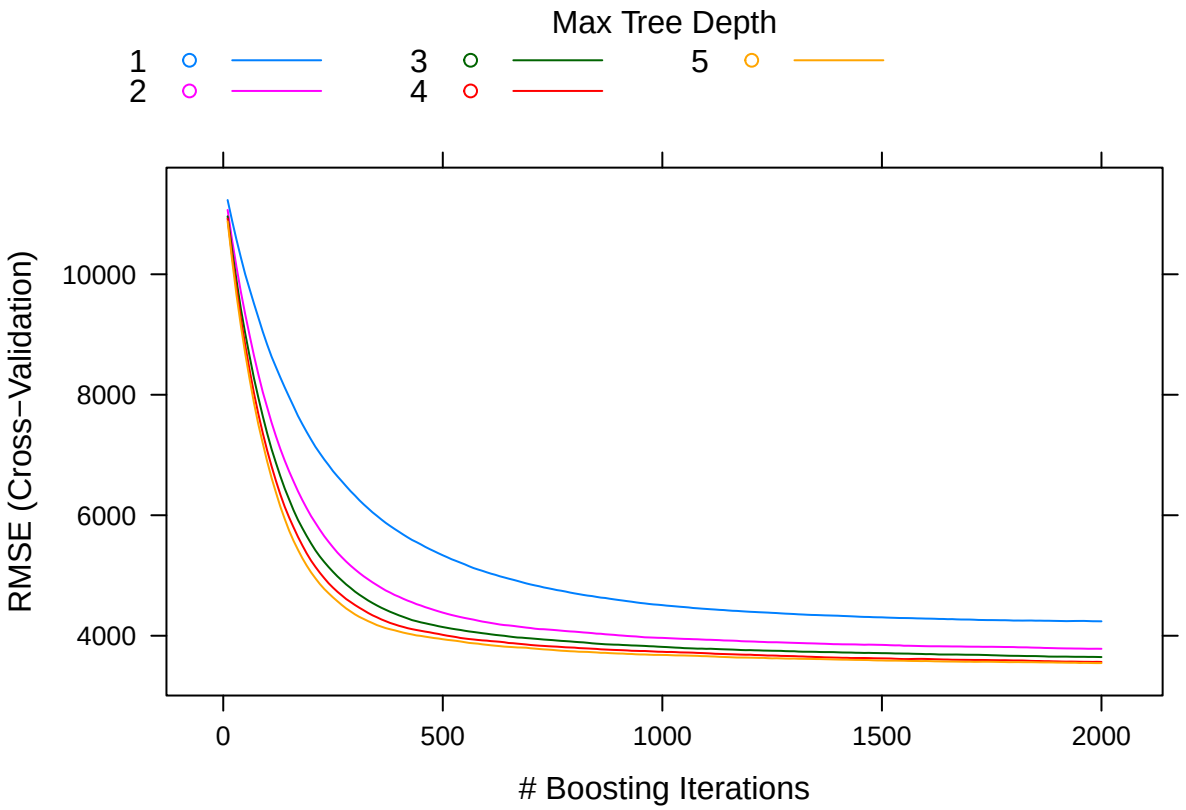


On obtient deux graphes basés sur deux différents critères. On observe que c'est globalement la variable "model" qui est la plus importante. Les variables "year", "mpg", et "engineSize" ont aussi un impact non négligeable. On se doute évidemment que le modèle d'une voiture, sa datation, la taille de son moteur et le nombre de miles par gallon ont un effet sur le prix de celle ci.

2.3 Méthode du Boosting

Nous pouvons ensuite faire une nouvelle méthode nommée le boosting, où chaque arbre tient compte des erreurs qu'a pu faire les précédents. On réutilise cette fois le package "caret". On va faire varier le nombre d'arbres utilisés, car cette fois ci nous n'avons pas de convergence de l'erreur quand celui ci augmente, ainsi que le nombre de split dans chaque arbre.

```
set.seed(7777777)
ctrl <- trainControl(method="cv", number=5)
param <- expand.grid(n.trees=seq(10,2000,10), interaction.depth=1:5, shrinkage=0.01, n.minobsinnode=10)
fit_gbm <- train(price~., data, method="gbm", trControl=ctrl, tuneLength=100, tuneGrid=param, verbose=F
plot(fit_gbm, type="l")
```



On constate sur le graphe qu'avoir une profondeur d'interaction de 25 est meilleur que toutes les autres peu importe le nombres d'arbres. Cependant augmenter ce chiffre rend la compilation du code de plus en plus complexe, pour une erreur qui ne descend que très peu.

On peut ensuite rechercher dans le modèle pour quels paramètres l'erreur quadratique moyenne est minimisée. C'est le cas quand le nombre d'arbres est de 1960, la profondeur d'interaction est de 5, et le shrinkage (influence des précédents arbres) est de 1%.

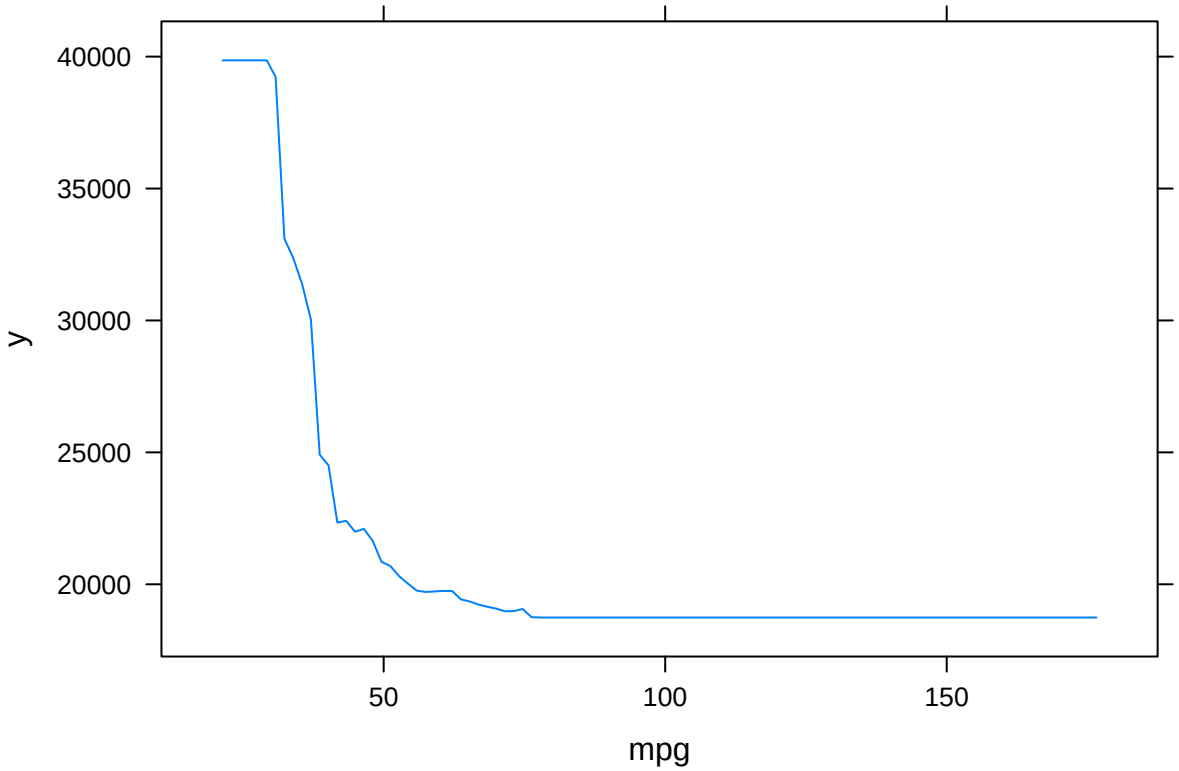
```
print(fit_gbm$bestTune)
```

```
##     n.trees interaction.depth shrinkage n.minobsinnode
## 999    1990                 5      0.01             10
```

Le RMSE de ce modèle est de moins de 4000, ce qui est excellent.

On peut enfin regarder comment les variables mpg et engineSize agissent sur le prix de la voiture.

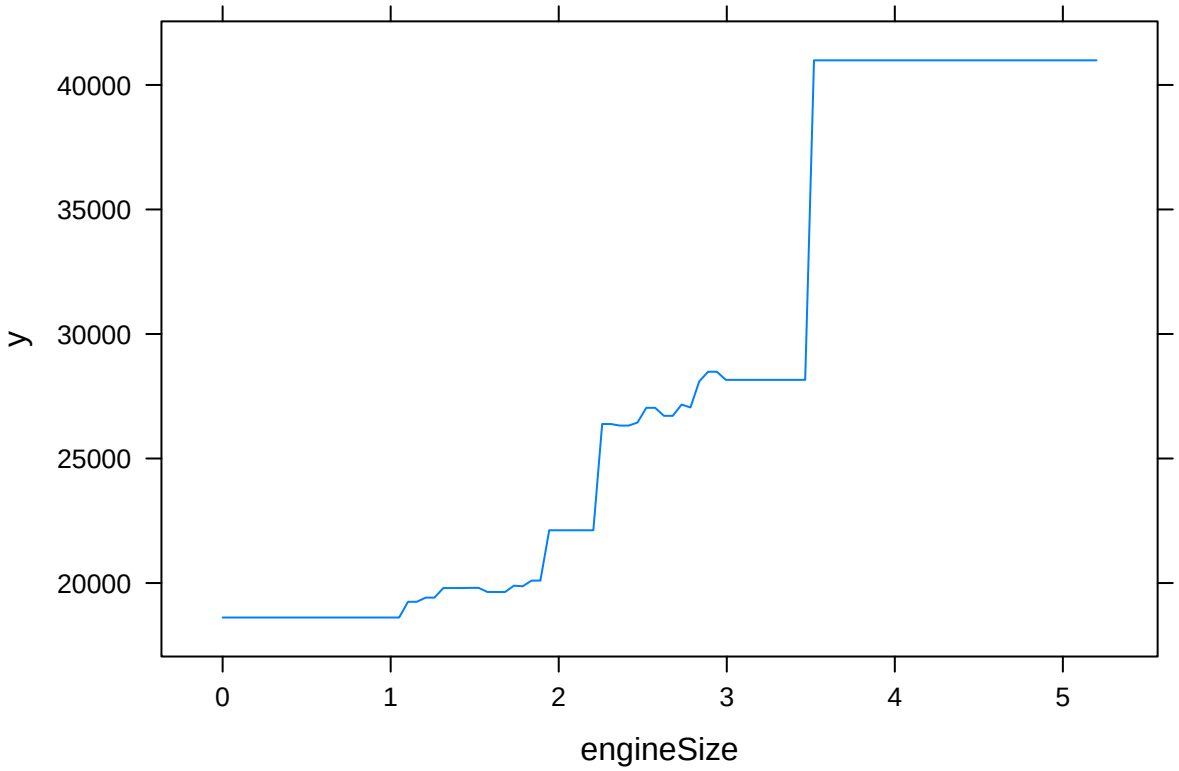
```
plot(fit_gbm$finalModel, i="mpg")
```



Pour mpg, on remarque qu'avoir environ 30 miles par gallon permet de faire grandement monter le prix de la voiture. Mais dès que cette valeur augmente, le prix peut très vite chuter à 20000$ (pour 50mpg).

Le même graphe est fait pour la variable engineSize.

```
plot(fit_gbm$finalModel, i="engineSize")
```



Lorsque la taille du moteur augmente, le prix de la voiture augmente aussi. On observe un bon énorme entre une taille de 3 et 4, avec un prix qui passe de moins de 30000$, à plus de 40000$. Cela s'explique par le fait qu'un gros moteur est souvent plus puissant, ce qui fait monter le prix.

2.4 Conclusion partie régression

On constate finalement que c'est la méthode du boosting qui surclasse les autres méthodes en minimisant le RMSE. Ce modèle effectue les meilleures prédictions du prix en fonction des autres variables. On constate que l'éventail de méthodes est assez faible ici, et on mettra en avant un plus large éventail pour la partie classification.

3 Etude et prévision des décès suite à insuffisance cardique

Les maladies cardiovasculaires sont les premières causes de décès dans le monde, faisant environ 17,9 millions de victimes chaque année, soit 31 % des décès. L'insuffisance cardiaque est un problème couramment causé par les maladies cardiovasculaires et cet ensemble de données contient 12 variables qui peuvent être utilisées pour prédire la mortalité par insuffisance cardiaque.

En effet, la plupart des maladies cardiovasculaires peuvent être évitées en s'attaquant aux facteurs de risque comportementaux tels que le tabagisme, la mauvaise alimentation, l'obésité, l'inactivité physique et la consommation d'alcool.

Les personnes atteintes de maladies cardiovasculaires ou présentant un risque cardiovasculaire élevé (en raison de la présence d'un ou de plusieurs facteurs de risque tels que l'hypertension, le diabète, l'hyperlipidémie ou une maladie déjà établie) ont besoin d'un dépistage et d'une prise en charge précoces, pour lesquels un modèle d'apprentissage automatique peut être d'une grande utilité.

Voici notre table de données :

```
heart <- read.csv(file="heart_failure.csv", header=T, sep=",")
summary(heart)
```

```
##       age           anaemia       creatinine_phosphokinase    diabetes
##  Min.   :40.00   Min.   :0.0000   Min.   :  23.0           Min.   :0.0000
##  1st Qu.:51.00   1st Qu.:0.0000   1st Qu.: 116.5           1st Qu.:0.0000
##  Median :60.00   Median :0.0000   Median : 250.0           Median :0.0000
##  Mean   :60.83   Mean   :0.4314   Mean   : 581.8           Mean   :0.4181
##  3rd Qu.:70.00   3rd Qu.:1.0000   3rd Qu.: 582.0           3rd Qu.:1.0000
##  Max.   :95.00   Max.   :1.0000   Max.   :7861.0           Max.   :1.0000
##  ejection_fraction high_blood_pressure   platelets      serum_creatinine
##  Min.   :14.00     Min.   :0.0000      Min.   : 25100   Min.   :0.500
##  1st Qu.:30.00     1st Qu.:0.0000      1st Qu.:212500   1st Qu.:0.900
##  Median :38.00     Median :0.0000      Median :262000   Median :1.100
##  Mean   :38.08     Mean   :0.3512      Mean   :263358   Mean   :1.394
##  3rd Qu.:45.00     3rd Qu.:1.0000      3rd Qu.:303500   3rd Qu.:1.400
##  Max.   :80.00     Max.   :1.0000      Max.   :850000   Max.   :9.400
##   serum_sodium        sex            smoking            time
##  Min.   :113.0   Min.   :0.0000   Min.   :0.0000   Min.   :  4.0
##  1st Qu.:134.0   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 73.0
##  Median :137.0   Median :1.0000   Median :0.0000   Median :115.0
##  Mean   :136.6   Mean   :0.6488   Mean   :0.3211   Mean   :130.3
##  3rd Qu.:140.0   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:203.0
##  Max.   :148.0   Max.   :1.0000   Max.   :1.0000   Max.   :285.0
##   DEATH_EVENT
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.3211
##  3rd Qu.:1.0000
##  Max.   :1.0000
```

Ici, le but va être de prédire la variable DEATH_EVENT, c'est-à-dire de prédire si un patient va mourir d'insuffisance cardiaque ou non. On va utiliser entre autre le package "caret" afin de réaliser toutes les méthodes étudiées en cours.

Cette table de données est constituée de 12 variables explicatives : l'âge du patient, le fait qu'il fasse de l'anémie ou non (booléen), son taux de créatinine phosphokinase, le fait qu'il soit diabétique ou non (booléen),

sa fraction d'éjection, le fait qu'il fasse de l'hypertension ou non (booléen), son taux de plaquettes, son taux de créatinine, son tauc de sodium, son sexe, le fait qu'il fume ou non (booléen) et le temps de suivi du patient.

On décide de supprimer la variable "temps de suivi du patient" car son usage n'est pas pertinent. En effet, cette variable n'a pas de lien avec la prédiction du décès par insuffisance cardiaque du patient.

```
heart <- heart[-c(12)]
```

On recode aussi la variable DEATH_EVENT en variable qualitative de modalités "vivant" et "mort".

```
heart$DEATH_EVENT <- as.factor(heart$DEATH_EVENT)
levels(heart$DEATH_EVENT)<-c("vivant","mort")
```

3.1 Régression logistique

Afin de prédire notre variable, nous allons dans un premier temps utiliser une regréssion logistique en utilisant la validation croisée à 5 couches.

```
set.seed(1)
ctrl <- trainControl(method="cv", number=5)
model_glm <- train(DEATH_EVENT~., data=heart, method='glm', trControl=ctrl)
print(model_glm)
```

```
## Generalized Linear Model
##
## 299 samples
##  11 predictor
##   2 classes: 'vivant', 'mort'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 240, 239, 239, 238, 240
## Resampling results:
##
##   Accuracy   Kappa
##   0.7393489  0.353023
```

Premièrement, on voit que le taux de succès ("Accuracy" en anglais) est de 74.26%. Cela signifie donc qu'environ trois quarts des prédictions données par la régression logistique sont bonnes. On rappelle tout de même qu'un algorithme qui répondrait au hasard obtiendrait théoriquement un taux de succès de 50%.

Ensuite, le Kappa est de 36.43%. On rappelle que ce coefficient est une "amélioration" du coefficient précédent. En effet, celui ci prend en compte la base du hasard qu'on a vu juste au dessus. Cette valeur n'est pas très haute.

On peut ensuite utiliser la commande "summaryFunction=twoClassSummary" pour obtenir d'autres valeurs intéressantes.

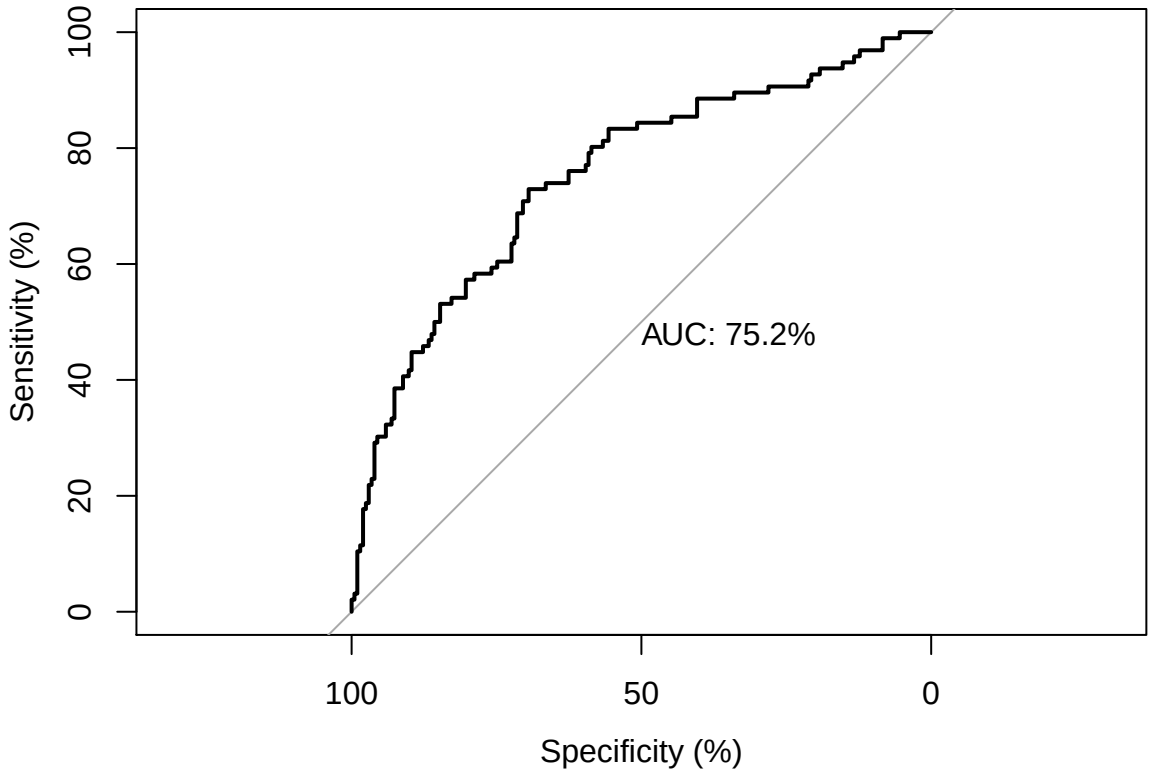
```
set.seed(1)
ctrl <- trainControl(method="cv", number=5, summaryFunction=twoClassSummary, classProbs=T, savePredicti
model_glm2 <- train(DEATH_EVENT~., data=heart, method='glm', trControl=ctrl)
print(model_glm2)
```

```
## Generalized Linear Model
##
## 299 samples
##  11 predictor
##   2 classes: 'vivant', 'mort'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 240, 239, 239, 238, 240
## Resampling results:
##
##   ROC        Sens       Spec
##   0.7576893  0.8773171  0.4484211
```

La sensibilité est le taux de vrais positifs, c'est à dire le taux de patients qui ont été prédits comme étant décédés, en l'étant réellement. Il se calcule en prenant le nombre de vrais positifs, divisé par tous les patients qui sont décédés. Ce coefficient est de 87,73%, ce qui est très bon.

La spécificité est le taux de vrais négatifs, c'est à dire le taux de patients qui ont été prédits comme étant vivant, en l'étant réellement. Il se calcule en prenant le nombre de vrais négatifs, divisé par tous les patients vivants. Ce coefficient est de 44,84%, ce qui est assez peu.

```
plot.roc(model_glm2$pred$obs, model_glm2$pred$mort, percent=T, print.auc=T)
```



On peut tracer la courbe ROC qui a en fait la spécificité en abcisse et la sensibilité en ordonnée avec un paramètre p qui varie le long de la courbe. Ce paramètre p représente le seuil utilisé pour la régression logistique.

L'AUC représente l'aire sous la courbe et détermine l'efficacité d'une méthode de classification. Ici il vaut 76,1%, ce qui semble bon mais il est à comparer avec les autres méthodes.

3.2 Analyse discriminante linéaire

On procède ensuite à une analyse discriminante linéaire. Une sorte de "variante" de l'analyse en composantes principales (ACP) ou cette fois ci c'est la distance entre les groupe de points que l'algorithme cherche à maximiser; ce qui rend la manipulation plus simple par la suite.

```
set.seed(1)
ctrl <- trainControl(method="cv", number=5)
model_lda <- train(DEATH_EVENT~., data=heart, method='lda', trControl=ctrl)
print(model_lda)
```

```
## Linear Discriminant Analysis
##
## 299 samples
##  11 predictor
##   2 classes: 'vivant', 'mort'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 240, 239, 239, 238, 240
## Resampling results:
##
##   Accuracy   Kappa
##   0.7459591  0.3566581
```

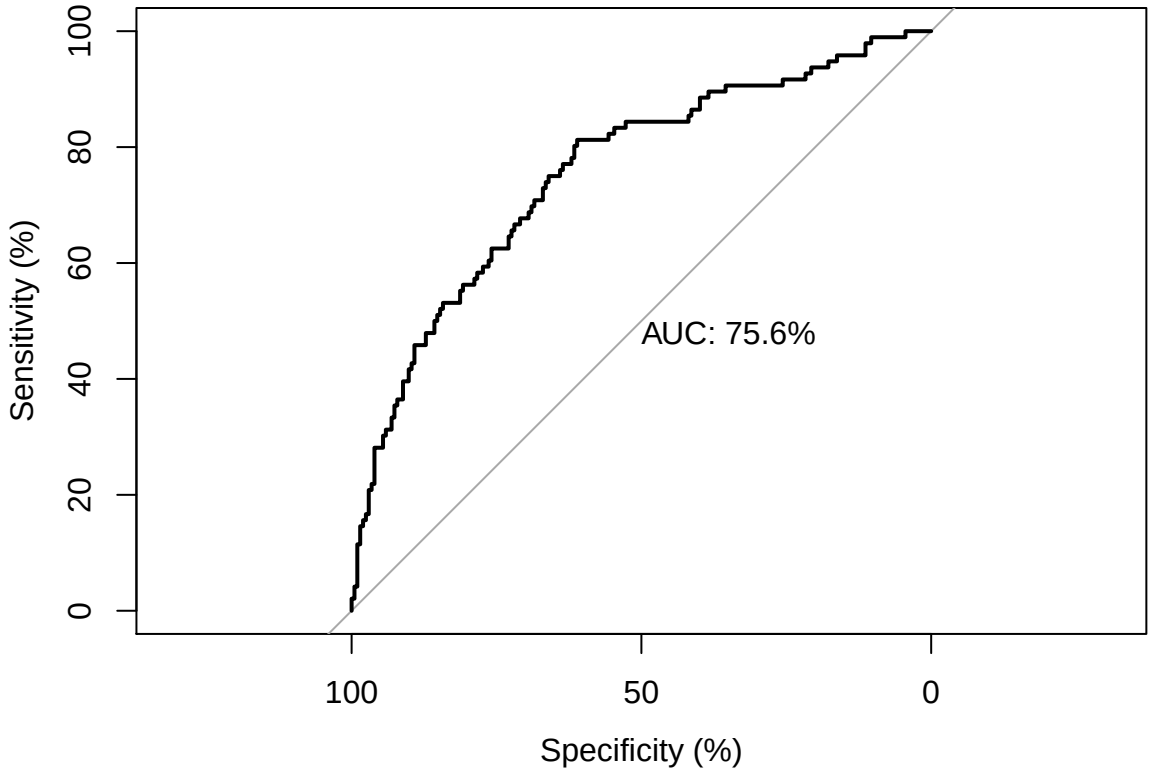
On obtient un taux de succès de 74,59% et un Kappa de 35,66%. Ce n'est pas mauvais mais loin d'être excellent.

```
set.seed(1)
ctrl <- trainControl(method="cv", number=5, classProbs=T, savePredictions=T, summaryFunction=twoClassSu
model_lda2 <- train(DEATH_EVENT~., data=heart, method='lda', trControl=ctrl)
print(model_lda2)
```

```
## Linear Discriminant Analysis
##
## 299 samples
##  11 predictor
##   2 classes: 'vivant', 'mort'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 240, 239, 239, 238, 240
## Resampling results:
##
##   ROC        Sens       Spec
##   0.7602439  0.8969512  0.4273684
```

Encore une fois la sensibilité est très bonne (89,69%) alors que la spécificité n'est pas très satisfaisante (42,73%).

```r
plot.roc(model_lda2$pred$obs, model_lda2$pred$mort, percent=T, print.auc=T)
```



L'AUC vaut 76%, c'est équivalent à ce que la régression logistique à pu produire.

3.3 Analyse discriminante quadratique

On passe à l'analyse discriminante quadratique. Cette méthode correspond en quelque sorte à une "amélioration" de la LDA, puisqu'on passe à un aspect quadratique plutôt que linéaire cette fois ci.

```r
set.seed(1)
ctrl <- trainControl(method="cv", number=5)
model_qda <- train(DEATH_EVENT~., data=heart, method='qda', trControl=ctrl)
print(model_qda)
```

```
## Quadratic Discriminant Analysis
##
## 299 samples
##  11 predictor
##   2 classes: 'vivant', 'mort'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 240, 239, 239, 238, 240
## Resampling results:
##
```

```
##   Accuracy  Kappa
##   0.685609  0.166981
```

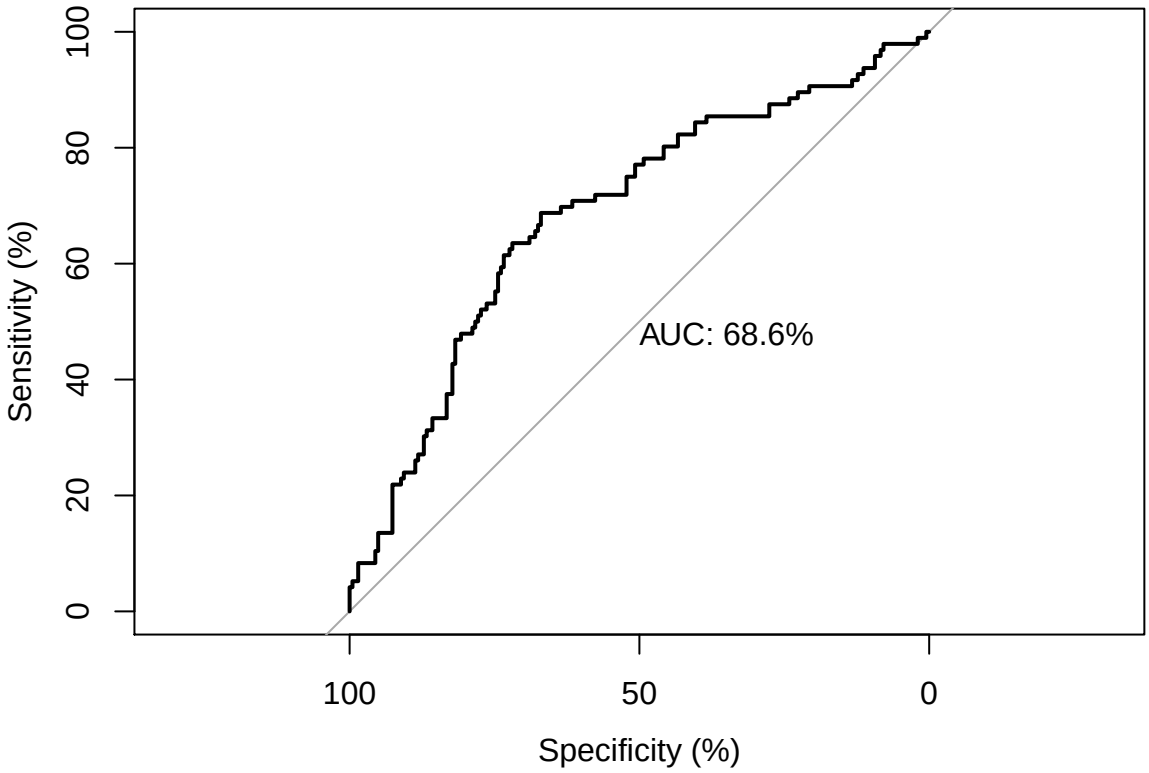
Le taux de succès vaut 68,56% et le Kappa vaut 16,69%. Cela n'est pas très satisfaisant et plus faible que pour les méthodes précédentes.

```
set.seed(1)
ctrl <- trainControl(method="cv", number=5, classProbs=T, savePredictions=T, summaryFunction=twoClassSu
model_qda2 <- train(DEATH_EVENT~., data=heart, method='qda', trControl=ctrl)
print(model_qda2)
```

```
## Quadratic Discriminant Analysis
##
## 299 samples
##  11 predictor
##   2 classes: 'vivant', 'mort'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 240, 239, 239, 238, 240
## Resampling results:
##
##   ROC        Sens       Spec
##   0.6862452  0.8865854  0.2621053
```

La sensibilité est encore bonne (88,65%) mais la spécificté est très mauvaise (26,21%), ce qui peut être problématique... En effet, cela signifie que lorsque le modèle prédit que le patient ne va pas mourir il se trompe très souvent et ce genre d'erreur est très grave.

```
plot.roc(model_qda2$pred$obs, model_qda2$pred$mort, percent=T, print.auc=T)
```



Le critère AUC vaut 68,9%. Il est moins bon que celui de la régression logistique et de l'analyse discriminante linéaire. Finalement, cette méthode montre parfois son inferiorité à sa comparse, la LDA.

3.4 Algorithme des proches voisins

On va ensuite utiliser l'algorithme des K plus proches voisins avec une validation croisée à 5 couches. On fait varier le nombre k de proches voisins de 5 à 200 avec un pas de 5.

```
set.seed(1)
param <- expand.grid(k=seq(5,100,5))
ctrl <- trainControl(method="cv", number=5)
model_knn <- train(DEATH_EVENT~., data=heart, method='knn', trControl=ctrl, preProcess=c("center","scal
print(model_knn)
```

```
## k-Nearest Neighbors
##
## 299 samples
##  11 predictor
##   2 classes: 'vivant', 'mort'
##
## Pre-processing: centered (11), scaled (11)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 240, 239, 239, 238, 240
## Resampling results across tuning parameters:
##
```

```
##   k    Accuracy   Kappa
##     5  0.7056710  0.21628144
##    10  0.6956692  0.14893469
##    15  0.7125053  0.16498098
##    20  0.6990044  0.10254654
##    25  0.6922812  0.06883238
##    30  0.6856108  0.04903091
##    35  0.6887784  0.05481478
##    40  0.6854450  0.04172147
##    45  0.6922247  0.05503145
##    50  0.6856127  0.02822719
##    55  0.6856127  0.02822719
##    60  0.6856127  0.02822719
##    65  0.6856127  0.02822719
##    70  0.6856127  0.02822719
##    75  0.6822793  0.01411360
##    80  0.6822793  0.01411360
##    85  0.6822793  0.01411360
##    90  0.6822793  0.01411360
##    95  0.6822793  0.01411360
##   100  0.6822793  0.01411360
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 15.
```

On trouve k=15 et le taux de succès associé vaut 71,25% et le Kappa vaut 16,50%. Ce n'est pas énorme...

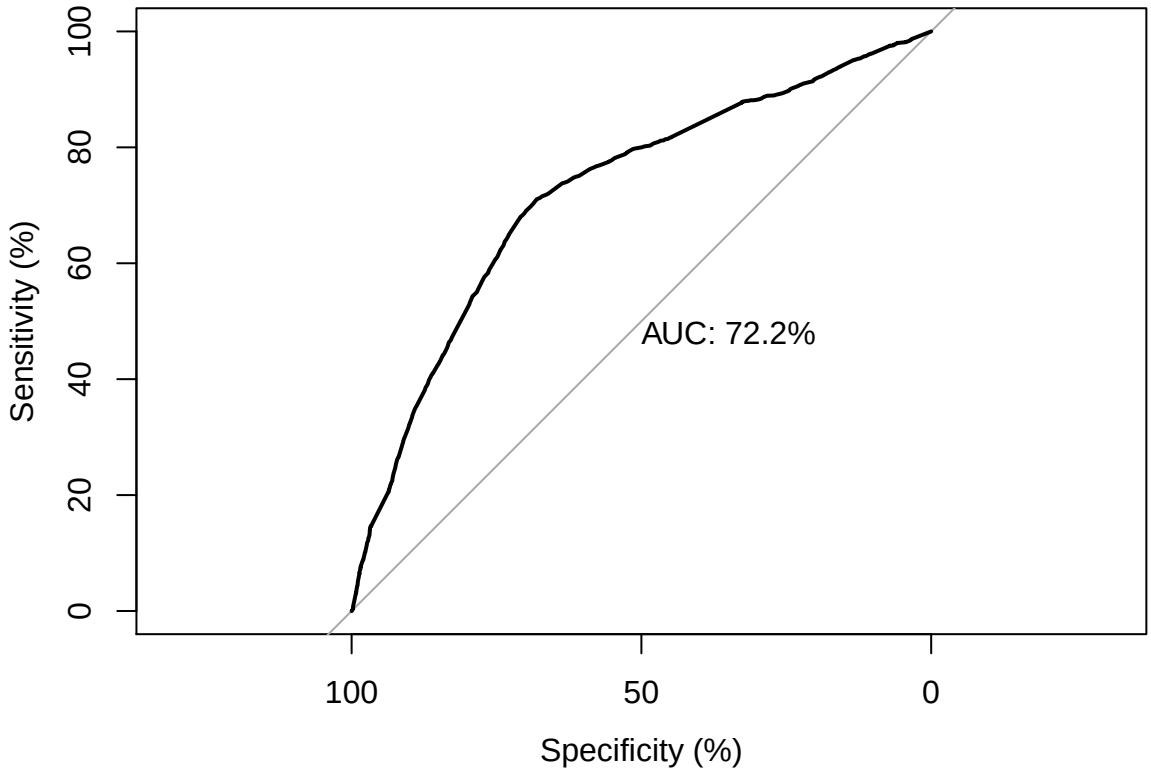
```
set.seed(1)
ctrl <- trainControl(method="cv", number=5, classProbs=T, savePredictions=T, summaryFunction=twoClassSu
model_knn2 <- train(DEATH_EVENT~., data=heart, method='knn', trControl=ctrl, preProcess=c("center","sca
print(model_knn2)
```

```
## k-Nearest Neighbors
##
## 299 samples
##  11 predictor
##   2 classes: 'vivant', 'mort'
##
## Pre-processing: centered (11), scaled (11)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 240, 239, 239, 238, 240
## Resampling results across tuning parameters:
##
##   k    ROC        Sens       Spec
##     5  0.6780873  0.9063415  0.28263158
##    10  0.6961585  0.9358537  0.18894737
##    15  0.7126027  0.9753659  0.15736842
##    20  0.7302118  0.9852439  0.09368421
##    25  0.7239891  0.9901220  0.06263158
##    30  0.7353177  0.9851220  0.05263158
##    35  0.7257221  0.9900000  0.05210526
##    40  0.7280841  0.9900000  0.04157895
##    45  0.7212484  1.0000000  0.04157895
```

```
##    50  0.7289666  1.0000000  0.02105263
##    55  0.7263703  1.0000000  0.02105263
##    60  0.7323042  1.0000000  0.02105263
##    65  0.7404108  1.0000000  0.02105263
##    70  0.7343485  1.0000000  0.02105263
##    75  0.7390693  1.0000000  0.01052632
##    80  0.7426990  1.0000000  0.01052632
##    85  0.7402503  1.0000000  0.01052632
##    90  0.7430520  1.0000000  0.01052632
##    95  0.7412869  1.0000000  0.01052632
##   100  0.7318261  1.0000000  0.01052632
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 90.
```

Pour k=90, la sensibilité vaut 100% et la spécificité vaut 10,53%. La sensibilité est parfaite (ce qui peut sembler étonnant) tandis que la spécificité est très mauvaise, ce qui est encore une fois très problématique.

```
plot.roc(model_knn2$pred$obs, model_knn2$pred$mort, percent=T, print.auc=T)
```



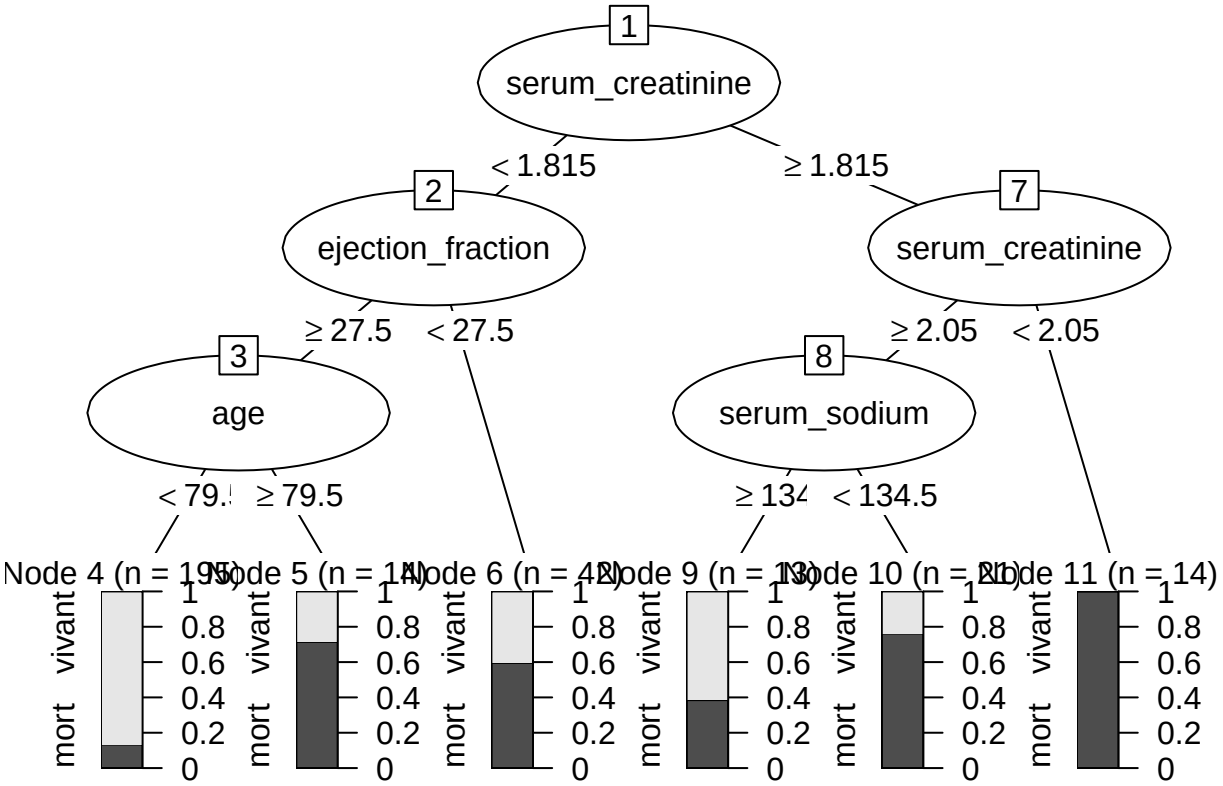
Le critère AUC vaut 71,8%, un bon score qui surpasse la QDA, mais qui ne dépasse pas les deux autres méthodes, mais reste à voir si celles ci font le poids face aux célèbres arbres de classification...

3.5 Arbres de classification

Les arbres de classification correspondent surement à une partie les plus influente dans l'apprentissage. Nous allons faire connaissance de quelques une d'entre elles dans le cadre de notre étude.

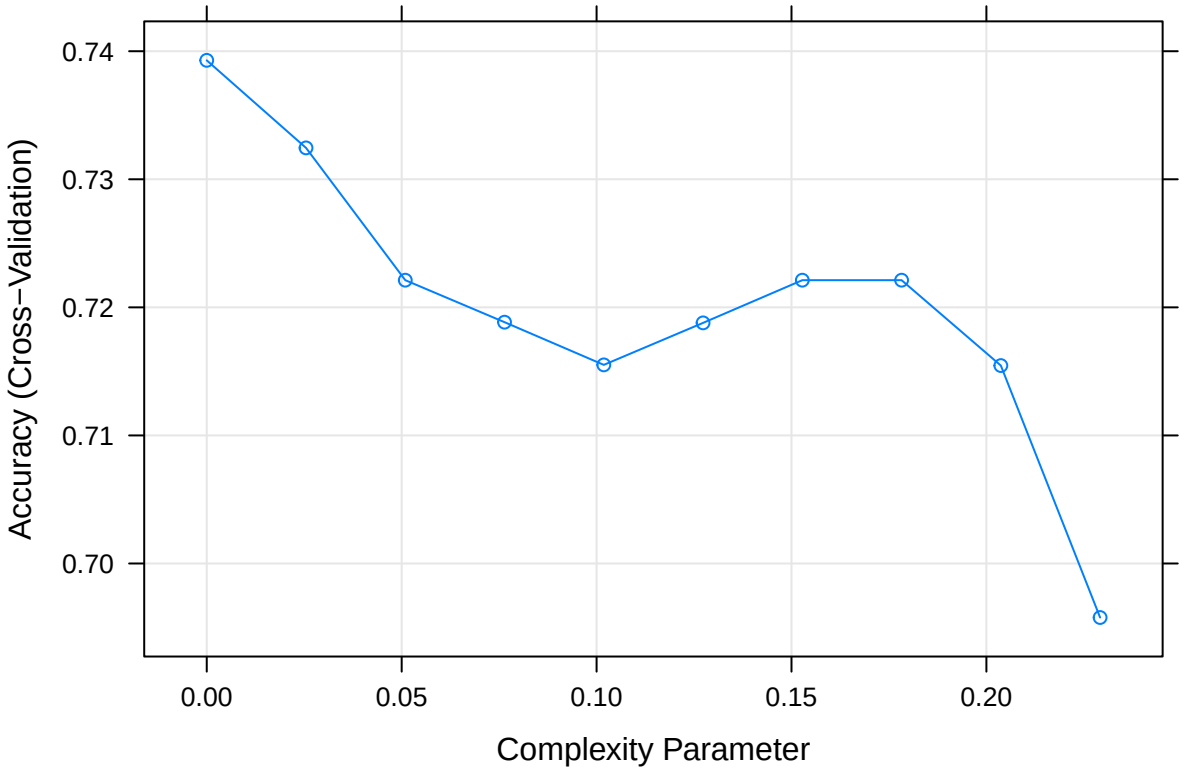
On crée d'abord un arbre de régression avec maxdepth=3 et on le visualise avec les packages "partykit" et "party".

```
ctrl <- rpart.control(maxdepth = 3)
tree <- rpart(DEATH_EVENT~.,data=heart,method='class',control=ctrl)
plot(as.party(tree))
```



On sélectionne ensuite le meilleur arbre en utilisant une validation croisée à K=5 couches puis on le visualise.

```
ctrl <- trainControl(method = "cv",number = 5)
set.seed(1)
fit_tree <- train(DEATH_EVENT~.,data=heart,method="rpart",trControl=ctrl,tuneLength=10)
plot(fit_tree)
```

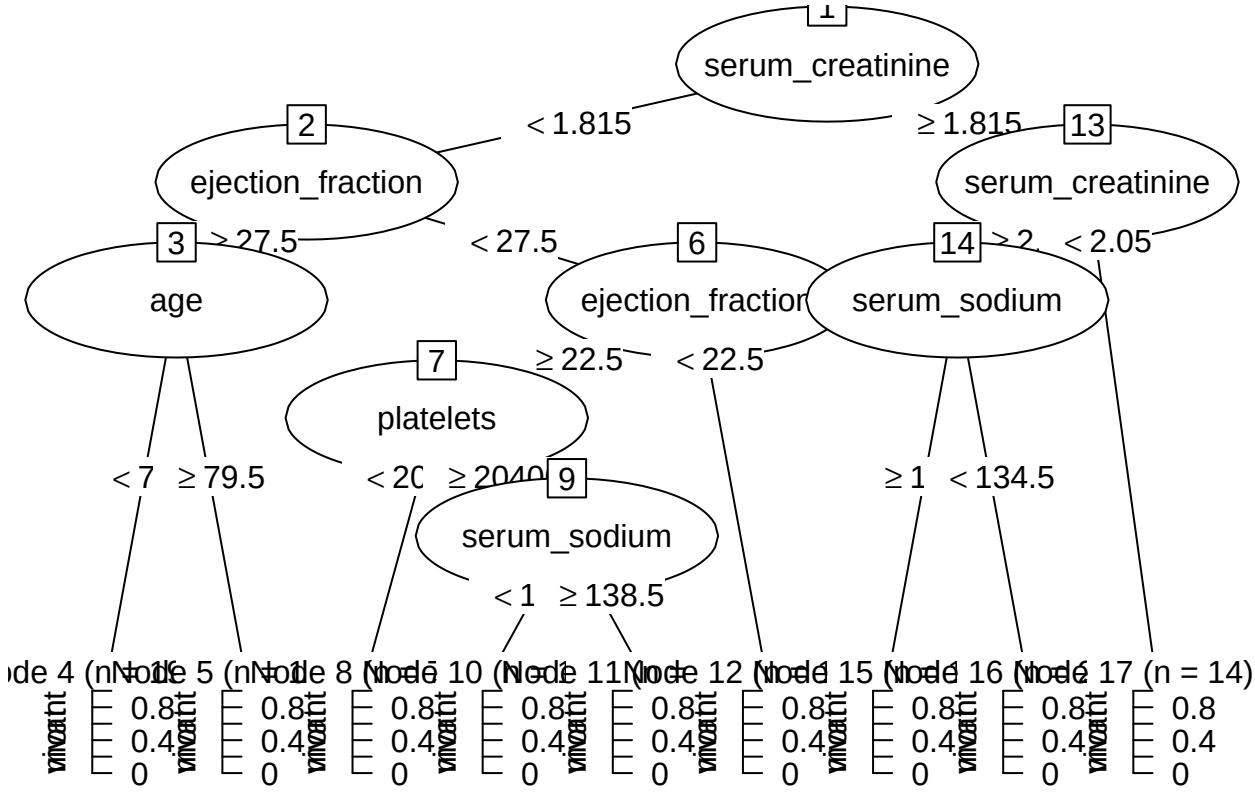



```
print(fit_tree)
```

```
## CART
##
## 299 samples
##  11 predictor
##   2 classes: 'vivant', 'mort'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 240, 239, 239, 238, 240
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.00000000  0.7392868  0.3934519
##   0.02546296  0.7324525  0.3866660
##   0.05092593  0.7221210  0.3004441
##   0.07638889  0.7188423  0.2885473
##   0.10185185  0.7155089  0.2628123
##   0.12731481  0.7187876  0.2594149
##   0.15277778  0.7221210  0.2547261
##   0.17824074  0.7221210  0.2547261
##   0.20370370  0.7154543  0.2208499
##   0.22916667  0.6957822  0.1402904
##
```

```
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.
```

```
fit_tree_best <- fit_tree$finalModel
plot(as.party(fit_tree_best))
```



On voit que le taux de succès est le plus haut lorsque le paramètre de complexité est nulle. Il vaut alors 73,93% et le Kappa 39,34%, ce qui est relativement bon.

3.5.1 Bagging

Nous allons ensuite passer à la méthode du Bagging, ou Bootstrap aggregation, qui est une des méthodes a base d'arbre les moins développées.

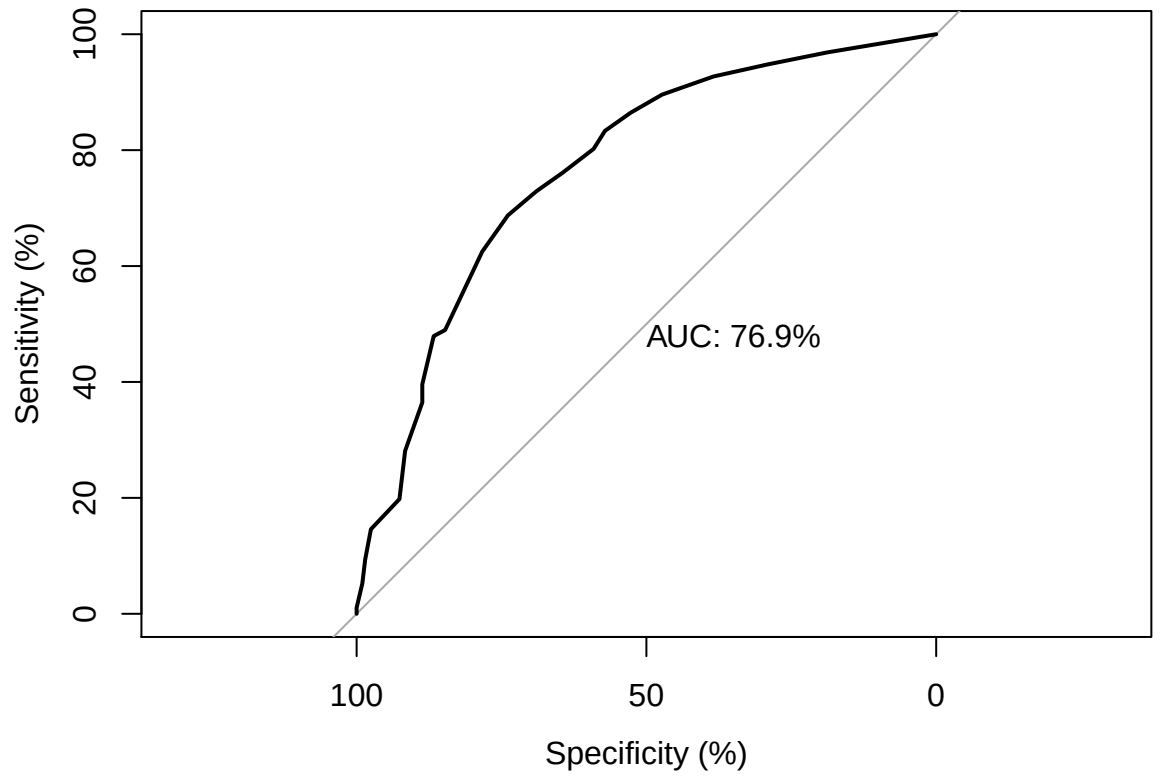
```
set.seed(1)
ctrl <- trainControl(method="cv",number=5,classProbs = T,savePredictions = T)
bagg <- train(DEATH_EVENT~.,data=heart,method="treebag",trControl=ctrl,importance=T)
print(bagg)
```

```
## Bagged CART
##
## 299 samples
##  11 predictor
##   2 classes: 'vivant', 'mort'
##
```

```
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 240, 239, 239, 238, 240
## Resampling results:
##
##   Accuracy  Kappa
##   0.732622  0.3839822
```

Le taux de succès vaut 73,26% et le Kappa vaut 38,40%. Ces résultats sont bons.

```
plot.roc(bagg$pred$obs, bagg$pred$mort, percent=T, print.auc=T)
```



Le critère AUC vaut 73,9%, ce qui est un peu en dessous de la LDA, et de la régression logistique. Cependant, d'autres méthodes plus preformantes vont faire leurs apparitions.

3.5.2 Random forest

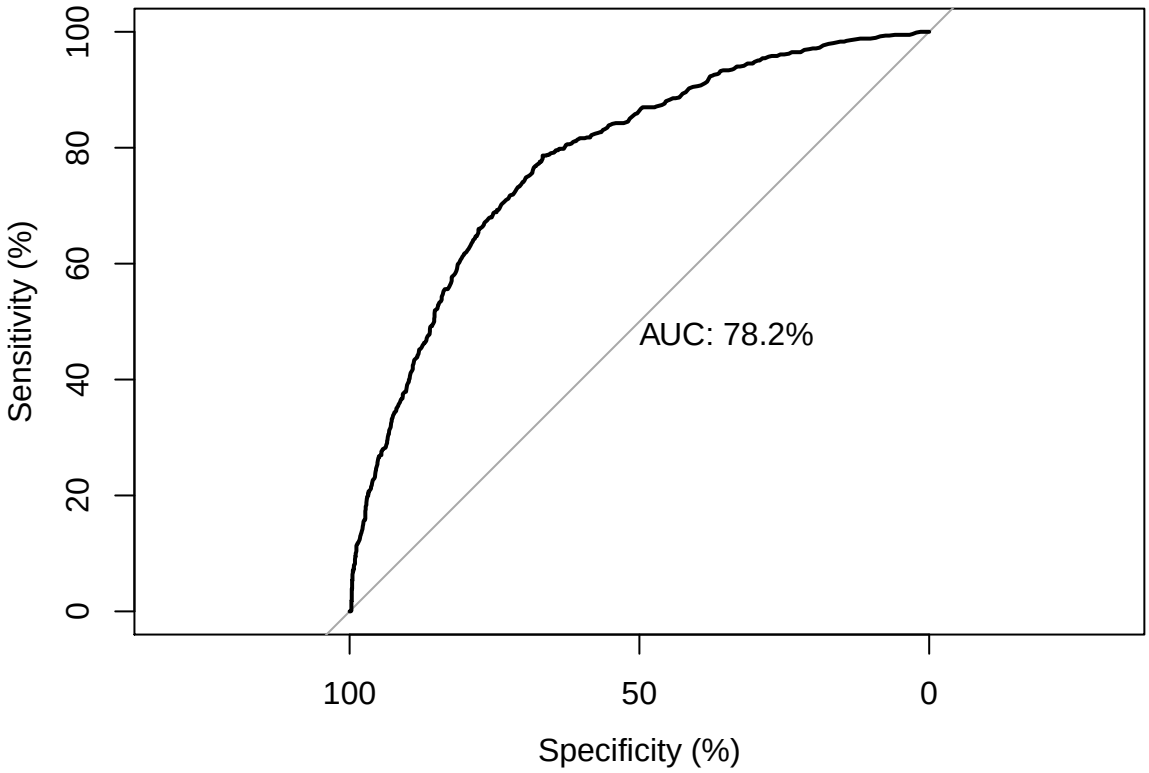
On passe ensuite à la méthode des forêts aléatoires. Cette méthode se différencie de la précédente par une utilisation des différentes variables moins diverse; en effet, le paramètre mtry contrôle le nb de variable utilisable pour la création de chaque arbre, ce qui permet d'éviter le sur-apprentissage.

```
set.seed(1)
param <- data.frame(mtry=1:p)
ctrl <- trainControl(method="cv",number=5,classProbs = T,savePredictions = T)
rf <- train(DEATH_EVENT~.,data=heart,method="rf",trControl=ctrl,tuneGrid=param,importance=T)
print(rf)
```

```
## Random Forest
##
## 299 samples
##  11 predictor
##   2 classes: 'vivant', 'mort'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 240, 239, 239, 238, 240
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   1     0.7125637  0.1479466
##   2     0.7360665  0.3302991
##   3     0.7461248  0.3720801
##   4     0.7529592  0.3999410
##   5     0.7493470  0.3937502
##   6     0.7494017  0.3984743
##   7     0.7258424  0.3451736
##   8     0.7494582  0.4068889
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 4.
```

On trouve que le mtry optimal vaut 3, ce n'est pas étonnant puisque l'on a appris qu'en classification il est environ égal à la racine du nombre de prédicteurs (ici on en a 11). Le taux de succès est alors de 75,96% et le Kappa de 40,32%. Ce sont les meilleures valeurs obtenues pour le moment.

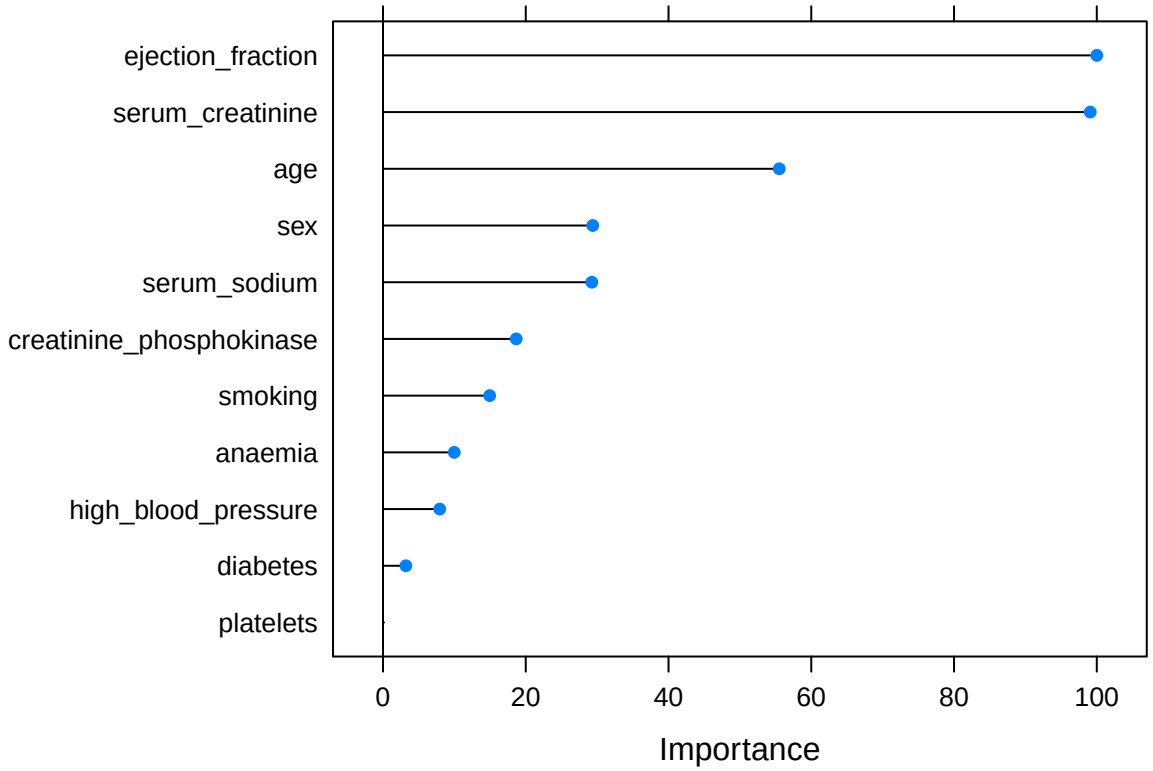
```
plot.roc(rf$pred$obs, rf$pred$mort, percent=T, print.auc=T)
```



Le critère AUC vaut 76,3%, c'est le meilleur obtenu pour l'instant !

On peut aussi tracer l'importance des variables.

```
plot(varImp(rf))
```

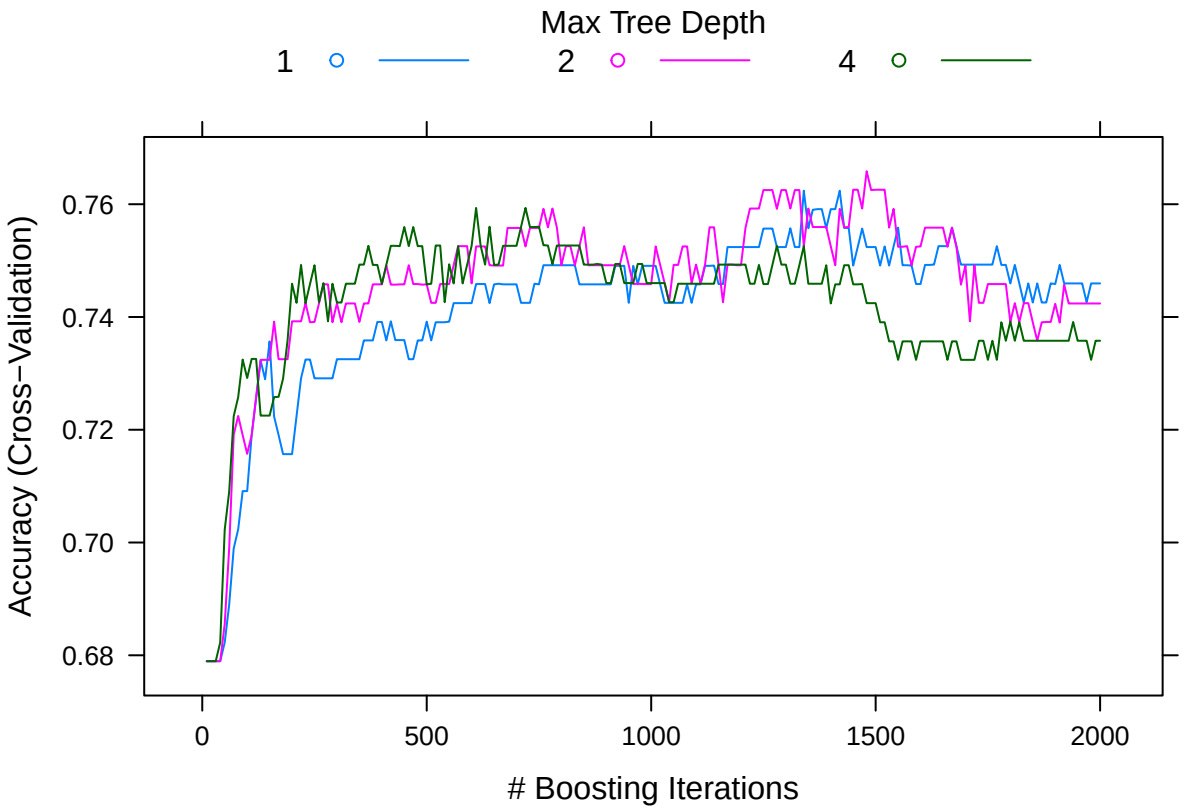


On voit que le taux de créatinine et la fraction d'éjection sont les 2 variables les plus importantes.

3.5.3 Boosting

On passe ensuite à la méthode du boosting. Celle ci est assez proche des précédentes méthodes, mais elle varie dans le sens où celle ci prend en compte les erreurs commises par les précédents arbres. Cela permet en général une amélioration de l'efficacité.

```
set.seed(1)
ctrl <- trainControl(method="cv",number=5,classProbs = T,savePredictions = T)
param <- expand.grid(shrinkage=0.01,interaction.depth=c(1,2,4),n.trees=seq(10,2000,10),n.minobsinnode=1
boost <- train(DEATH_EVENT~.,data=heart,method="gbm",trControl=ctrl,tuneGrid=param,verbose=F)
plot(boost,type="l")
```



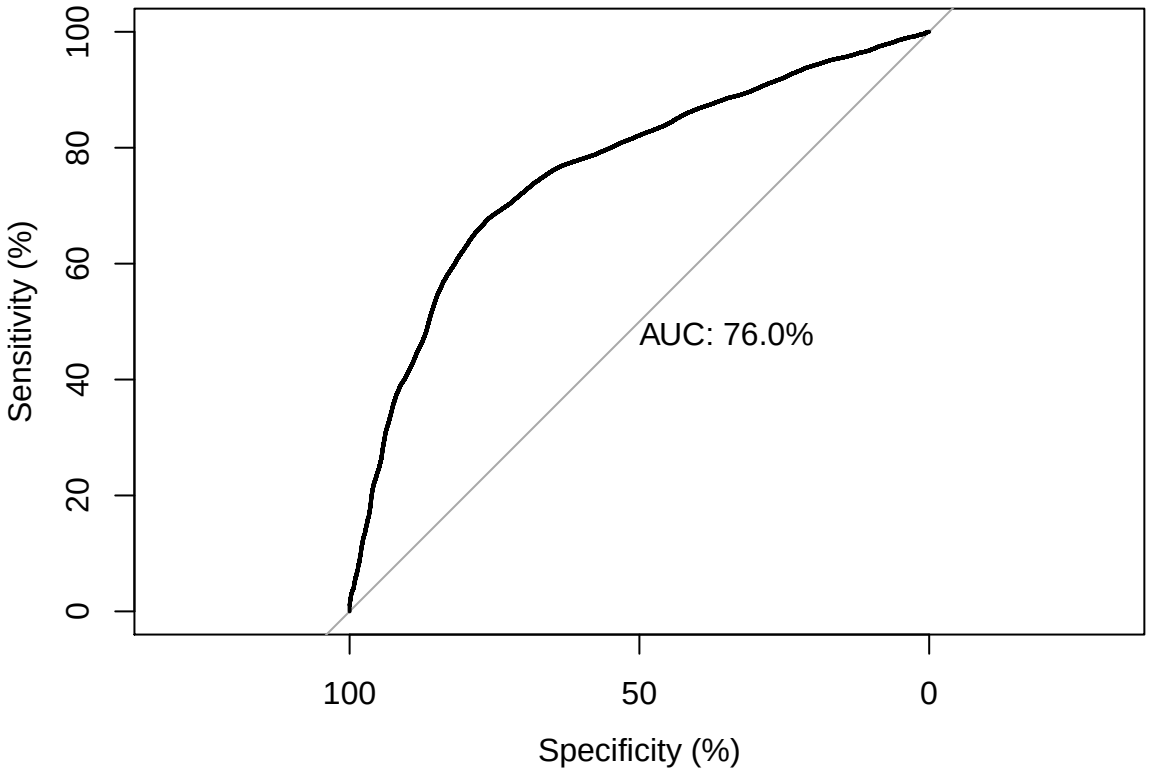
On voit ici le taux de succès en fonction du nombre d'itérations pour une profondeur de 1, 2 et 4. Logiquement, le résultat s'améliore lorsqu'on augmente cette variable.

```
print(boost$bestTune)
```

```
##     n.trees interaction.depth shrinkage n.minobsinnode
## 348    1480                 2      0.01             10
```

Voici les meilleurs paramètres. Il suffit finalement de peu d'abres pour obtenir un résultat optimal.

```
plot.roc(boost$pred$obs, boost$pred$mort, percent=T, print.auc=T)
```



Le critère AUC vaut 74,5%, il est relativement bon mais pas suffisant pour concurrencer les quelques autres meilleurs méthodes.

3.6 Support Vector Machine

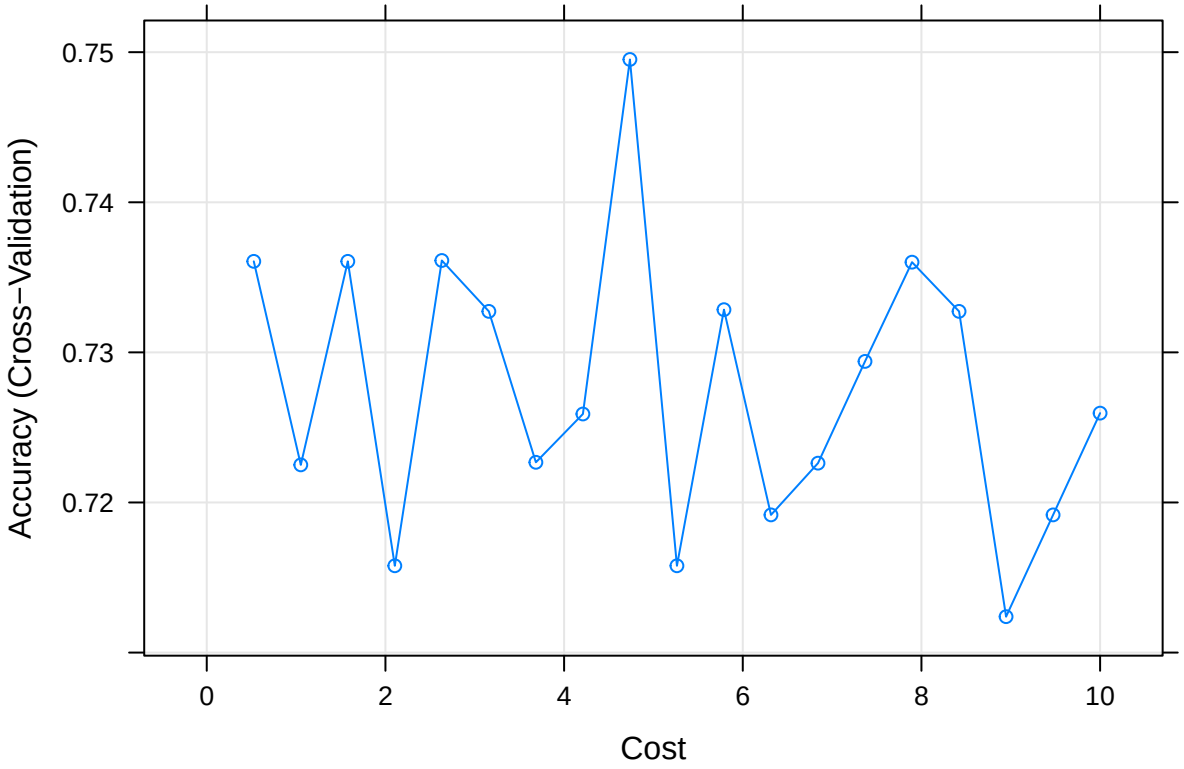
La SVM (Support Vector Machine) est une technique de classification binaire très populaire dans le machine learning. Son approche est très géométrique, ce qui difffère des autres méthodes. Le but de cette méthode est de séparer les deux groupe de points sur un plan à l'aide d'une droite (ou plan). Cependant, étant donné qu'il est parfois difficile de faire une bonne séparation avec une simple droite (ou plan), il est possible de parfois augmenter le nombre de dimensions en prenant par exemple le carré d'une des variables, ce qui aura pour conséquencer de séparer un peu plus les points, et donc de rendre la manipulation plus simple.

3.6.1 Méthode avec noyau Linéaire

Il y a la possibilité d'utiliser un noyau dans le support vector machine. On appelle cela le "kernel trick", une méthode possèdant des propriétés intéressantes dans le cadre de la classification. La méthode que nous allons utiliser ici prend en compte un noyau linéaire. Nous allons donc utiliser ce noyau dans un premier temps, puis un autre sera utilisé ensuite, ce qui servira de comparaison aux 2 methodes SVM. On notera que le noyau linéaire correspond simplement au produit scalaire.

Cette méthode implique un problème d'optimisation complexe à résoudre, qui implique un paramètre C qu'il faut bien régler puisque celui ci contrôle la marge, ainsi que le nombre de supports vectoriels. Ce paramètre est donc choisi par validation croisée comme démontré ci-dessous.

```
set.seed(1)
ctrl <- trainControl(method="cv",number=5,classProbs = T,savePredictions = T)
param <- expand.grid(C = seq(0, 10, length = 20))
fit_svm <- train(DEATH_EVENT~.,data=heart,method="svmLinear",trControl=ctrl,preProcess = c("center","sca
plot(fit_svm)
```



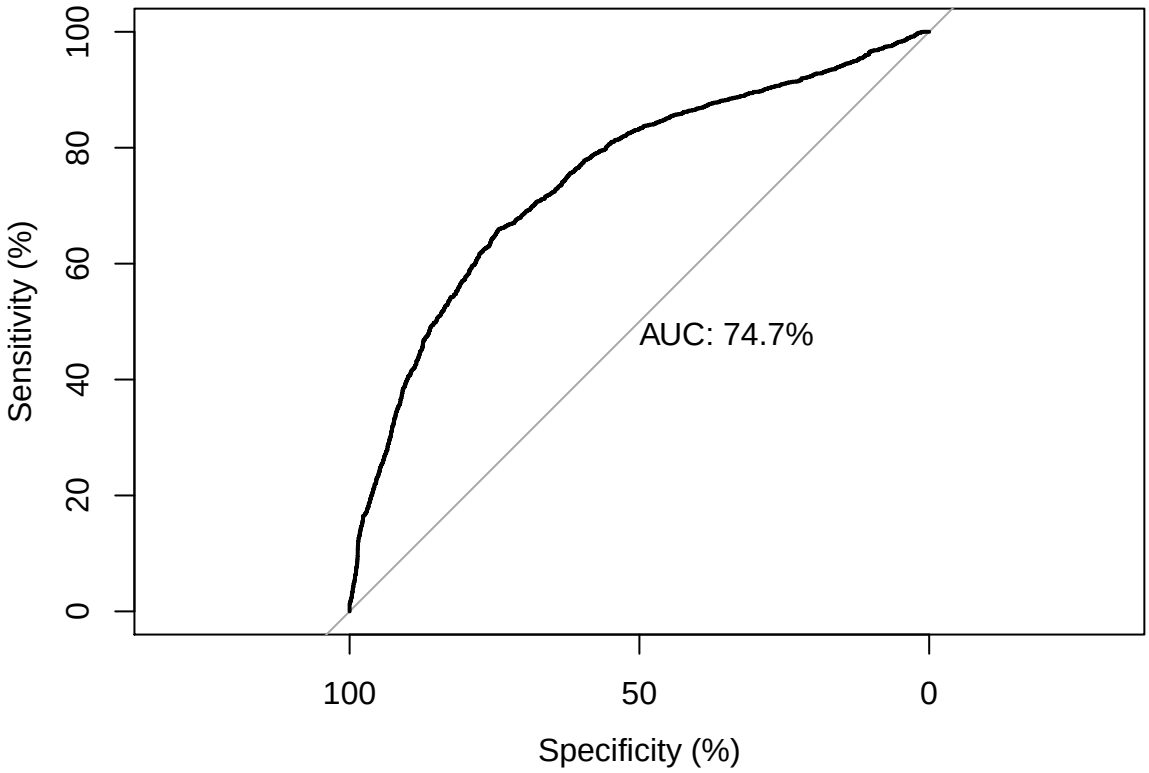
Ce graphique permet de voir qu'il faut prendre un paramètre C "petit" pour obtenir les meilleurs résultats possibles. Comme le résultat suivant le mettre, l'étude est optimale pour C=3.68 environ.

```
fit_svm$bestTune
```

```
##          C
## 10 4.736842
```

On trace une nouvelle fois la courbe ROC:

```
plot.roc(fit_svm$pred$obs, fit_svm$pred$mort, percent=T, print.auc=T)
```

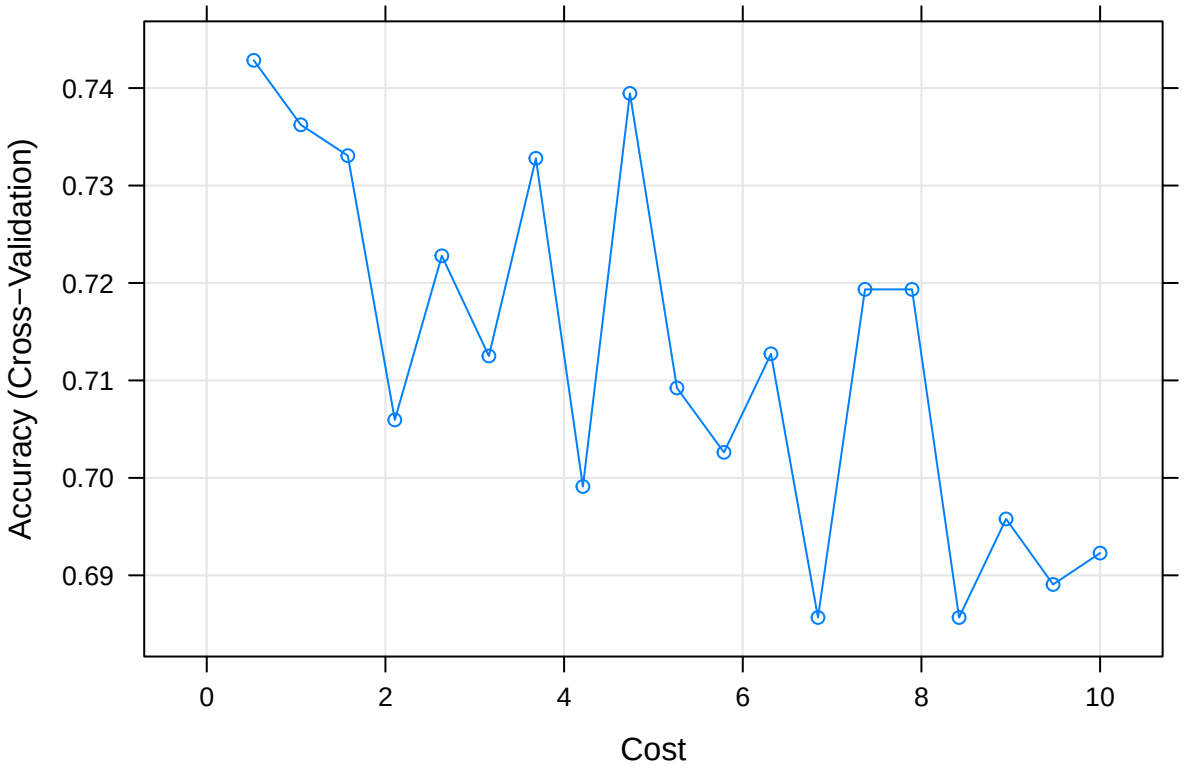


Finalement, la courbe ROC est décente avec un coefficient AUC de 75.9%. Cette méthode montre donc des résultats intéressants, que nous allons pouvoir comparer aux autres dans la partie qui suit.

3.6.2 Méthode avec noyau Radial

Pour cette seconde méthode, le nouveau noyau utilisé est le noyau Gaussien Radial, définit par $K(x, x') = exp(-\sigma^2||x - x'||^2)$.

```
set.seed(1)
ctrl <- trainControl(method="cv",number=5,classProbs = T,savePredictions = T)
param <- expand.grid(C = seq(0, 10, length = 20), sigma=0.05)
fit_svm_rad <- train(DEATH_EVENT~.,data=heart,method="svmRadial",trControl=ctrl,preProcess = c("center"
plot(fit_svm_rad)
```



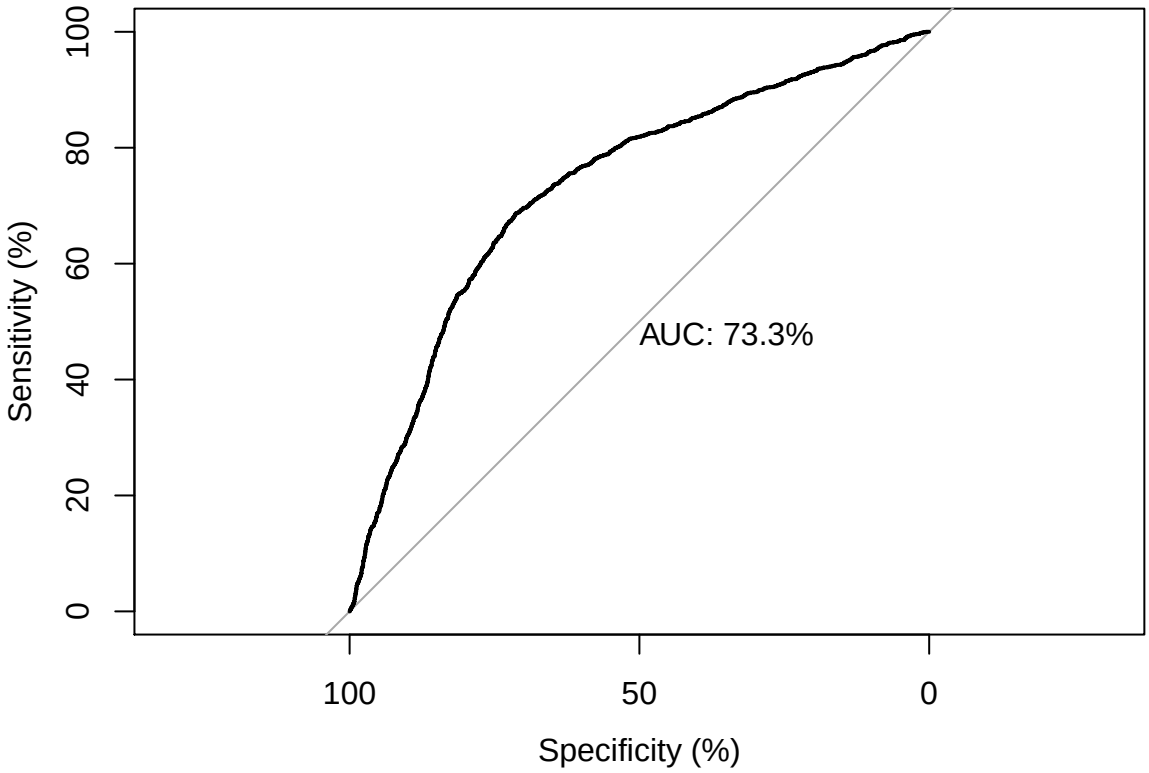
Cette fois ci, c'est pour C=0.53 environ que la méthode est optimale. Nous avons par ailleurs fait le choix de fixer le sigma présent dans le noyau à 0.05.

```
fit_svm_rad$bestTune
```

```
##   sigma         C
## 2  0.05 0.5263158
```

Nous traçons donc la courbe ROC, et on constate finalement que le coefficient AUC est de 72.4%. La méthode est finalement moins performante avec le noyau gaussien qu'avec le noyau linéaire.

```
plot.roc(fit_svm_rad$pred$obs, fit_svm_rad$pred$mort, percent=T, print.auc=T)
```



Conclusion de la partie

```
df <- data.frame(Plan = c("RegLog", "LDA", "QDA","k-nn", "Bagging", "RF", "Boosting", "SVMlin", "SVMrad
df
```

```
##       Plan  AUC
## 1   RegLog 76.1
## 2      LDA 76.0
## 3      QDA 68.9
## 4     k-nn 71.8
## 5  Bagging 73.9
## 6       RF 76.3
## 7 Boosting 74.5
## 8   SVMlin 75.9
## 9   SVMrad 72.4
```

Après avoir effectué toutes nos études, il est possible de faire un tableau comparatif de toutes nos méthodes selon le critère AUC. On notera que ce critère n'est pas LE meilleur critère utilisable, l'étude d'une base de donnée et des meilleures méthodes pour effectuer des prédictions ne s'arrêtent pas là. Pour revenir au sujet principal, c'est finalement les Random Forest qui comptabilisent le plus gros pourcentage, un peu devant la régression logistique, la LDA, et le SVM.

4 Conclusion du projet

Ce projet d'apprentissage nous as permis d'utiliser sur des données trouvées par nous même, différentes méthodes permettant la prédiction de caractère qualitatif, ou quantitatif. Cela nous a permit de nous faire une idée des méthodes qui sont les plus souvent utilisées, mais aussi les plus efficaces, comme les méthodes d'arbres en général. Cet éventail de connaissance pourra être réutilisable dans le futur.