TP1 : Arbre de décision et Forêt aléatoire

Youness Bouallou
Taha Ez-zoury
Marouane Elbissouri

29 octobre 2023

1 Problème I : Arbres de décision :

1. Pour déterminer le meilleur des deux splits, on calcule la variation de l'impureté :

$$\Delta\text{Imp(t)} = \text{Imp(t)} - \pi_L\text{Imp}(t_L) - \pi_R\text{Imp}(t_R)$$

On utilise comme fonction d'impureté (Imp) la focntion d'entropie (H) définie par :

$$H(t) = -\sum_{k=1}^{p} \text{p}(k|t)\log_2 \text{p}(k|t)$$

Donc :

. Pour le split 1: $\Delta\text{Imp}_1(t) \approx 1 - \frac{1}{2} \times 0.81 - \frac{1}{2} \times 0.81 \approx 0.19$

. Pour le split 2: $\Delta\text{Imp}_2(t) \approx 1 - \frac{3}{4} \times 0.92 - \frac{1}{4} \times 0 \approx 0.31$

En conclusion, le meilleur split est celui qui présente la plus grande variation d'impureté, donc le split 2 est préférable.

2. R(T) est défini par : $\text{R(T)} = \sum_{i=1}^{m} \frac{N(t_i)}{n} \cdot \text{R}(\text{t}_i)$ où les t_i sont les noeuds terminaux.

$$et \quad R(t) = \sum_{k=1, k\neq Y(t)}^{K} p(k|t)$$

$$avec : \quad Y(t) = \arg\max_{k=1,\ldots,K} p(k|t)$$

R_α(T) est défini par : $R_\alpha(\text{T}) = \text{R(T)} + \alpha \times taille(T)$

2.(a). Dans notre cas m = 4 et K = 2. On obtient alors :

$$\mathrm{R(T)} = \frac{350}{1000}\,\frac{50}{350} + \frac{250}{1000}\,\frac{100}{250} + \frac{250}{1000}\,\frac{50}{250} + \frac{150}{1000}\,\frac{50}{150} = 0.25$$

$$R_\alpha(\mathrm{T}) = 0.25 + 4\alpha$$

2.(b). Pour que T_0 soit préférable à T_1, il faut que $R_\alpha(\mathrm{T}_0) \leq \mathrm{R}_\alpha(\mathrm{T}_1)$

$$R_\alpha(\mathrm{T}_0) = 1 \times \frac{400}{1000} + \alpha \quad et \quad \mathrm{R}_\alpha(\mathrm{T}_1) = \frac{600}{1000}\frac{200}{600} + \frac{400}{1000}\frac{200}{400} + 2\alpha$$

$$R_\alpha(\mathrm{T}_0) = 0.4 + \alpha \quad et \quad \mathrm{R}_\alpha(\mathrm{T}_1) = 0.4 + 2\alpha$$

donc: $R_\alpha(\mathrm{T}_1) - \mathrm{R}_\alpha(\mathrm{T}_0) = \alpha$

En conclusion : T_0 est préférable à $T_1 \iff \alpha \geq 0$

3.

3.(a). Cas X_j continue : Puisque on a N_t individus
alors le nombre de split binaire est N_t - 1.

3.(b). Cas X_j discrète : Puisque X_j peut prendre L valeurs différentes alors le nombre de split binaire est L.

4.

4.(a). Si on a des données manquantes, les splits suppléants permettent de contourner ce problème en choisissant un autre split (càd une autre variable) qui imite le plus notre split optimal. Plus concrétement, càd les splits qui permettent de conserver les individus à droite dans la partie droite du split et les individus à gauche dans la partie gauche du split, en maximisant la formule de probabilité mentionnée dans le cours.

4.(b). Le fait de trouver tous les splits suppléants par ordre, qui imite le split optimal nous permettera de classer nos variables par ordre d'impotance, la variable la plus importante lors d'un split est la variable associé au premier split suppléant.

FIN du Problème I

2 Problème II : Arbres de décision, une application sur des données réelles :

2.1 Analyse exploratoire et traitement de la base de données:

Durant tout le TP, on travaillera avec la base de données de detection du diabète. On essaie permièrement de visualiser notre base de données:

```
# Clear the R environment
rm(list=ls())

# Visualize the data
data = read.csv("diabetes detection.csv", sep=";", header= TRUE)
head(data)

# Boxplots
for (i in 1:length(colnames(data))) {
  boxplot(data[, colnames(data)[i]], main = colnames(data)[i])
}
```

R Output:

	preg	plas	pres	skin	insu	mass	pedi	age	class
1	6	148	72	35	0	33.6	0.627	50	*tested_positive*
2	1	85	66	29	0	26.6	0.351	31	*tested_negative*
3	8	183	64	0	0	23.3	0.672	32	*tested_positive*
4	1	89	66	23	94	28.1	0.167	21	*tested_negative*
5	0	137	40	35	168	43.1	2.288	33	*tested_positive*
6	5	116	74	0	0	25.6	0.201	30	*tested_negative*

Commençons par donner une définition à chacun des paramètres :

preg = Le nombre de grossesses antérieures de la personne.
plas = la concentration de glucose dans le plasma sanguin.
pres = pression sanguine.
skin = épaisseur de la peau.
insu = le niveau du serum d'insuline, indicateur de la résistance à l'insuline.
mass = mesure de la masse grasse corporelle en se basant sur la longeur et la

masse de l'individu.
pedi = "DiabetesPedigreeFunction" Une fonction qui représente la probabilité de diabète en fonction des antécédents familiaux. Plus le pedi est élevé, plus il est probable qu'un individu ait des antécédents familiaux de diabète.

Toutes les variables de la base de données, à l'exception de "class", sont quantitatives, comme le montrent les boxplots dans le code.

À présent, effectuons une transformation de notre base de données, où "class" prendra la valeur 0 pour les tests négatifs et 1 pour les tests positifs.

Après on vérifie et on trouve qu'il n'y a pas de données manquantes :

```
# transformation de la colonne "class"
data$class <- ifelse(data$class == "tested_positive", 1, 0)
data$class <- factor(data$class)

# Verification de la presence des valeurs manquantes
print(paste("somme des na values:", sum(is.na(data))))

# Au cas ou il y a des valeurs manquantes on essaie de les predire en construisant des forets sur les valeurs non manquantes
clean_data= rfImpute(class ~ ., data= data, iter=6) #iter= nb de forets
```

R Output:

somme des na values: 0

Error in rfImpute.default(m, y, ...) : No NAs found in m

Par la suite, nous divisons notre base de données en deux parties distinctes : une pour l'entraînement et une autre pour les tests.

```r
# On choisit aleatoirement 30% de data pour le test
index <- sample(1:nrow(data),round(0.70*nrow(data)))
train <- data[index,]
test <- data[-index,]
```

2.2 Construction et analyse de l'arbre de décision

Commençons par construire l'arbre de décision maximal brut (On peut pas la visualiser ici vu qu'elle est plus grande). Ensuite, on procéde à l'élagage en sélectionnant l'arbre qui présente la plus faible valeur de l'erreur R_α:

Remarque importante: il faut bien fixer la graine aléatoire (seed) au début du programme, car sinon on trouve qu'à chaque fois on exécute on a une table cp différente (à cause du choix aléatoire de 30% de la population).

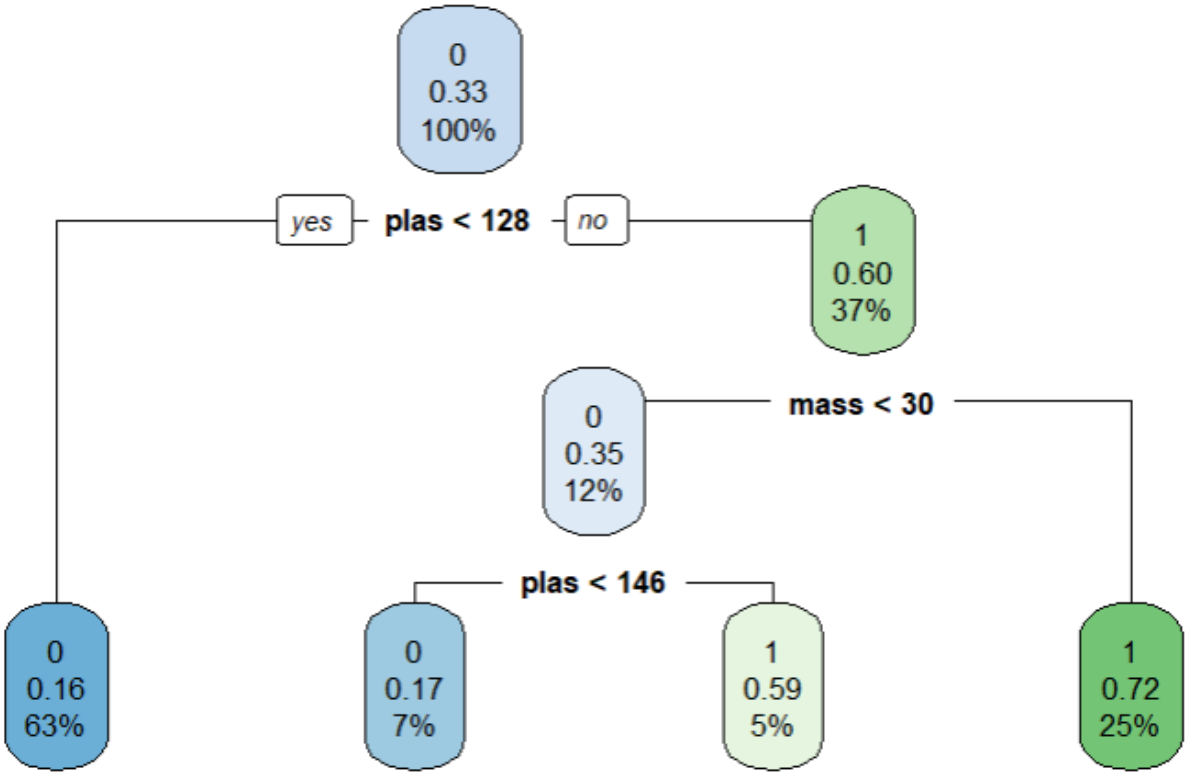
```r
# Construction de l'arbre brute maximale
tree = rpart(class~preg+plas+pres+skin+insu+mass+pedi+age, data=train, method="class")

# Visualisation de la table cp
tree$cptable
```

R Output:

	CP	nsplit	rel error	xerror	xstd
1	0.232955	0	1.00000	1.0000000	0.06183108
2	0.107955	1	0.76705	0.7727273	0.05727679
3	0.028409	2	0.65909	0.7443182	0.05656256
4	0.022727	3	0.63068	0.7045455	0.05550182
5	0.015152	4	0.60795	0.7215909	0.05596532
6	0.013258	8	0.54545	0.7500000	0.05670824
7	0.010000	12	0.48864	0.7670455	0.05713676

Donc on peut voir que $R_\alpha = 0.7045455$ qui correspond à une arbre élagué contenant 3 splits.

```r
# Elagage
min_ind <- which.min(tree$cptable[, "xerror"])
min_cp <- tree$cptable[min_ind, "CP"]
pruned_tree <- rpart::prune(tree, cp = min_cp)

# On trace l'arbre elague
rpart.plot::rpart.plot(pruned_tree)
```



Arbre optimisé (Arbre après élagage)

2.3 Application sur les données test et analyse

Effectuons des prédictions en utilisant notre modèle, puis comparons ces prédictions avec les valeurs réelles de la classe test. On évaluera la précision, la sensibilité et la spécificité de notre modèle.

```
# Prediction des donnees test
predicted <- predict(pruned_tree, test, type="class")

# Calculer l'erreur de prediction et la precision
error1=sum(test$class != predicted)/length(predicted)
print(paste("precision :",(1-error1)*100,"%"))

# Matrice de confusion
predicted <- ordered(predicted, levels = c(1, 0))
actual<- ordered(test$class, levels = c(1, 0))
mc=table(predicted,actual, dnn=c( "Predicted","reelle"))
#matrice de confusion
mc

#sensibilite
se= mc[1,1]/(mc[1,1]+mc[2,1])
#ou
se= sensitivity(mc)
print(paste("sensibilite :",(se)*100,"%"))

#specificite
sp= mc[2,2]/(mc[2,2]+mc[1,2])
#ou
sp= specificity(mc)
print(paste("specificite :",(sp)*100,"%"))

#f-score
print(paste("f-score :", 100*2*(sp*se)/(se+sp), "%"))
```

R Output :

		Reelle 1	0
Predicted	1	52	19
	0	40	119

Précision : 74.3478260869565%
Sensibilité : 56.5217391304348%
Spécificité : 86.231884057971%
f-score : 68.2851467667182%

La précision globale de ce modèle est bonne (à peu près 3 prédictions exactes parmi 4), mais dans quelle classe il est plus précis ? Passant alors au Se et Sp.

On observe que la spécificité est nettement plus élevée que la sensibilitée, ce qui signifie que le modèle a une capacité plus forte à identifier les personnes non diabétiques (classe 0) qu'à identifier les personnes diabétiques (classe 1).

On peut aller jusqu'à affirmer que le modèle n'est pas fiable pour la détection des diabétiques (car c'est proche à choisir au hasard si une personne est malade, càd avec une probabilité de 50%).

Le f-score est de 64 % montre que la performance globale du modèle est assez bonne.

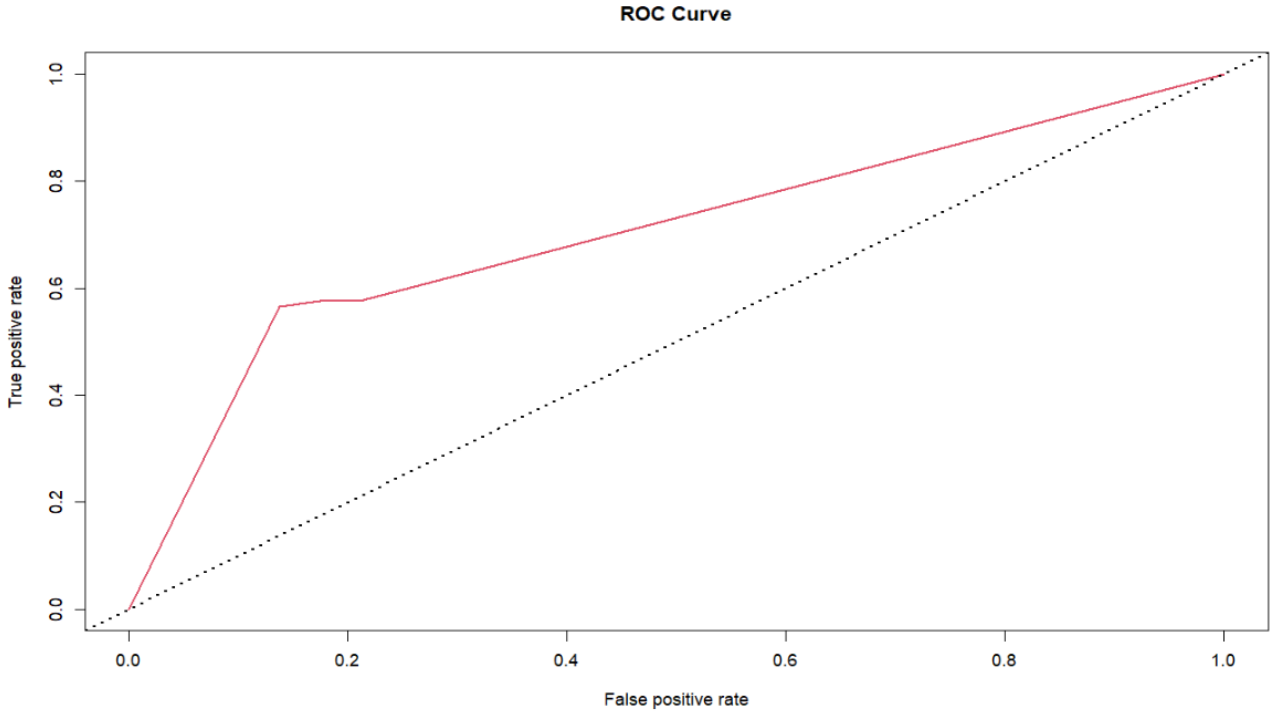
Traçons la courbe ROC:

```
#ROC curve
Predprob <- predict(pruned_tree, newdata = test
  ,type = "prob")
Predprob = as.data.frame(Predprob)
Prediction <- prediction(Predprob[2],test$class
  )
performance <- performance(Prediction, "tpr","
  fpr")
plot(performance,main = "ROC Curve",col = 2,lwd
```

```
    = 2)
abline(a = 0,b = 1,lwd = 2,lty = 3,col = "black
  ")

#final metric: area under curve
aucDT <- performance(Prediction, measure = "auc
  ")
aucDT <- aucDT@y.values[[1]]
print(paste("area under curve :", aucDT))
```

.

R Output :



area under curve : 0.7028986

On a l'AUC du modèle est entre 0.7 et 0.8, donc on peut dire que la performance du modèle est assez bonne.

Identifions maintenant le seuil le plus proche du point (0,1), ce seuil correspond au point où le modèle atteint sa meilleure performance:

```
closest_to_01 = function(fpr, tpr){
  n=length(fpr)
  xopt=0
  yopt=0
  distance=1
  for (i in 1:n){
    if (fpr[i]*fpr[i]+(1-tpr[i])*(1-tpr[i])<
      distance*distance){
      distance= sqrt(fpr[i]*fpr[i]+(1-tpr[i])*
        (1-tpr[i]))
      xopt=fpr[i]
      yopt=tpr[i]
    }
  }
  return(c(xopt, yopt))
}
closest_to_01(fpr, tpr)
```

.

R Output : (0.1376812, 0.5652174)

Donc avec une sensibilié de 56% et une spécificité de 86% le modèle est plus performant, c'est le cas de l'arbre élaguée trouvée avant.

Conclusion :

- Ce modèle a montré une bonne capacité à identifier les cas négatifs par rapport aux cas positifs.
- En général, on peut dire que le modèle a bien performé, bien que l'on ne puisse pas le qualifier d'excellent.

FIN du Problème II

3 Problème III : Courbe LIFT :

1. Soit l'expression de LIFT :

$$\text{LIFT} = \frac{\frac{A}{M}}{\frac{a}{n}} \tag{1}$$

Avec : - M est le nombre des individus choisis parmi n individus, il contient les vrais prédits et les faux prédits, avec les notations du TP : M=(#VP + #FP)

-A est le nombre des vrais prédits dans M : A=(#VP)

-a est le nombre des "Bons" clients parmi n clients, a=($\#G_2$)

$$\text{LIFT} = \frac{\frac{A}{M}}{\frac{a}{n}} = \frac{A}{a}.\frac{n}{M} \tag{2}$$

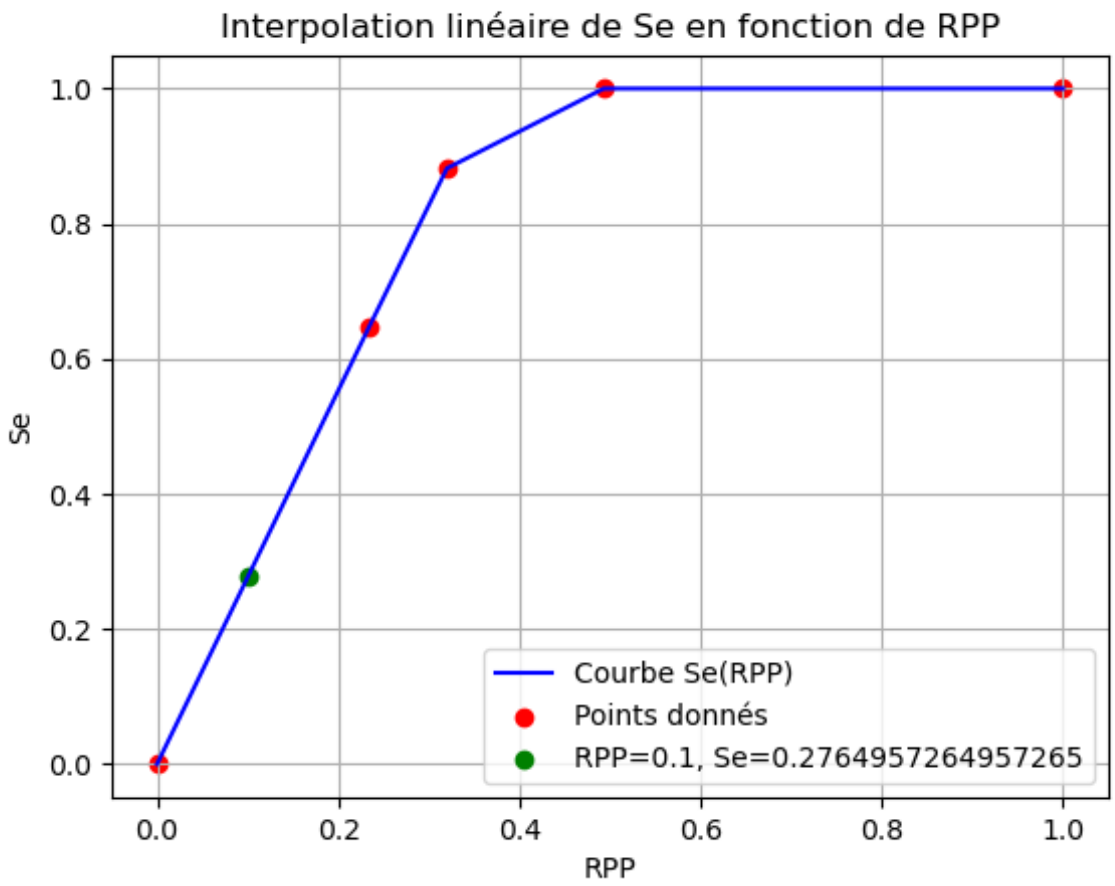
$$= \frac{\#VP}{\#G_2}.\frac{n}{\#VP + \#FP} \tag{3}$$

$$\boxed{\text{LIFT} = \frac{Se}{RPP}} \tag{4}$$

2. Le tableau LIFT :

$\hat{T}^*_{opt}$	Score			
	≥ 0.5789	≥ 0.5714	≥ 0.1428	≥ 0.0
# individus	19	26	40	81
RPP	0.234	0.32	0.493	1
Se	0.647	0.882	1	1
LIFT	2.764	2.756	2.02	1

3. La courbe LIFT (Se en fonction de RPP) :



4. Analysons les points de la bissectrice:

À un point de la bissectrice, on a : $Se = RPP \iff \frac{A}{a} = \frac{M}{n}$

C'est à dire choisir M parmi n est la même chose que choisir A parmi a. Autrement dit si on choisit un pourcentage p de la population à prélever, le poucentage des vrais prédits (A/a) est aussi p. Si la courbe LIFT est la bissectrice, il s'agit du modèle le plus couteux car pour arriver à 100% des "Bons" clients il faut prélever 100% de la population.

On a $M.\frac{a}{n} = A$, alors les points de la bissectrice correspondent exactement au cas ou on choisit les clients au hasard.

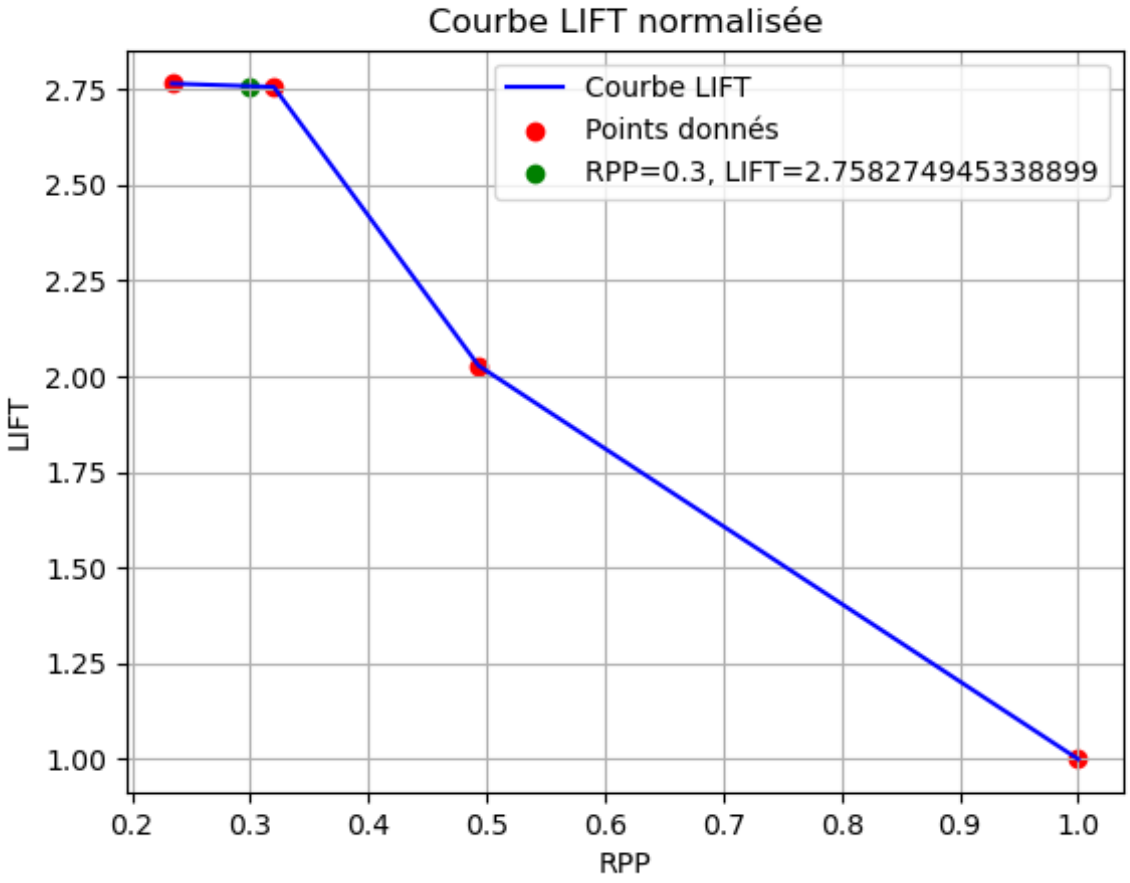
5. En utlisant une interpolation linéaire sur les points du tableau (en ajoutant le point (0,0)), on a à $RPP = 10\%$: $Se = 0.276$ (Voir la courbe), alors le LIFT à 10% est :

$$LIFT = \frac{0.276}{0.1} = 2.76$$

6. La construction des deux courbes est différente, la courbe ROC représente la relation entre le taux de vrais positifs (sensibilité) et le taux de faux positifs (1 - spécificité) à différents seuils de classification, alors que la courbe LIFT se concentre uniquement sur le taux de vrais positifs en le représentant en fonction du pourcentage des individus à prédire (RPP) sans prendre en considération la spécificité.

Leurs objectifs sont differents, La courbe ROC met l'accent sur la discrimination entre les classes, tandis que la courbe LIFT se concentre sur l'amélioration de la détection des vrais positifs par rapport à une prédiction aléatoire.

7. Courbe LIFT normalisée:



8. D'après la dernière courbe, à un RPP à 30% (qui correspont à 24 clients de 81) on a un LIFT égale à 2.75, qui correspond à une sensibilité égale à $LIFT.RPP = 0.825$, le nombre de "Bons" clients parmis ces 24 est $Se.A = 14$ clients. Alors on a bien augmenté le nombre de bons clients à 3 de plus.

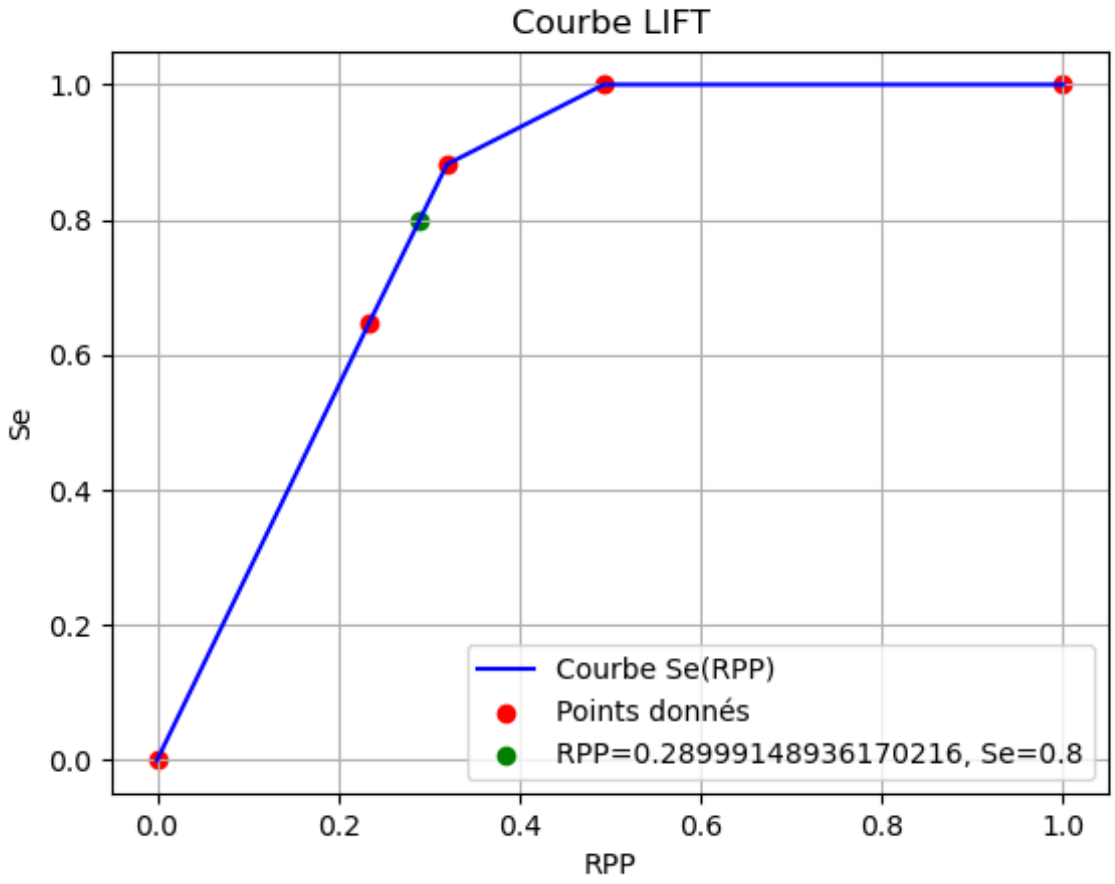
Donc pour répondre à la demande du vendeur on peut choisir un LIFT à 30%, qui correspond à ajouter 5 euros de plus (de 19 clients à 24 clients).

Si on veut arriver aux même résultats avec un choix au hasard, il faut cibler $\frac{14}{17}.81 = 66$ clients, qui correspond à un cout de 4.71 euros pour rejoindre un bon client au lieu de 1.71 euros trouvé par notre modèle, qui correspond à une économie totale de 42 euros.

Remarque: notre modèle n'est pas à 100% fiable prace qu'on un estimé ce LIFT à 30% par une interpolation linéaire, ce qui pourra nous emmener à fausse estimation, et pour mieux diminuer cette erreur, il nous faut plusieurs seuils pour construire la courbe.

9.a En s'appuyant sur ce qui a été mentionné dans la question 4, le pourcentage des clients à contacter au hasard pour espérer toucher 80 % des bons clients est 80% de la population. (Les points de la bissectrice)

9.b Espérer 80% des bons clients, c'est à dire arriver à une sensibilité de 80%, d'après la courbe de la question 3, on peut arriver à cette sensiblité avec un LIFT de 29% (avec une interpolation linéaire), c'est à dire contacter seulement 29% de la population au lieu de 80% (= contacter au hasard), donc on a pu réduire plus que la moitié de la population ciblée en gardant les mêmes performances.



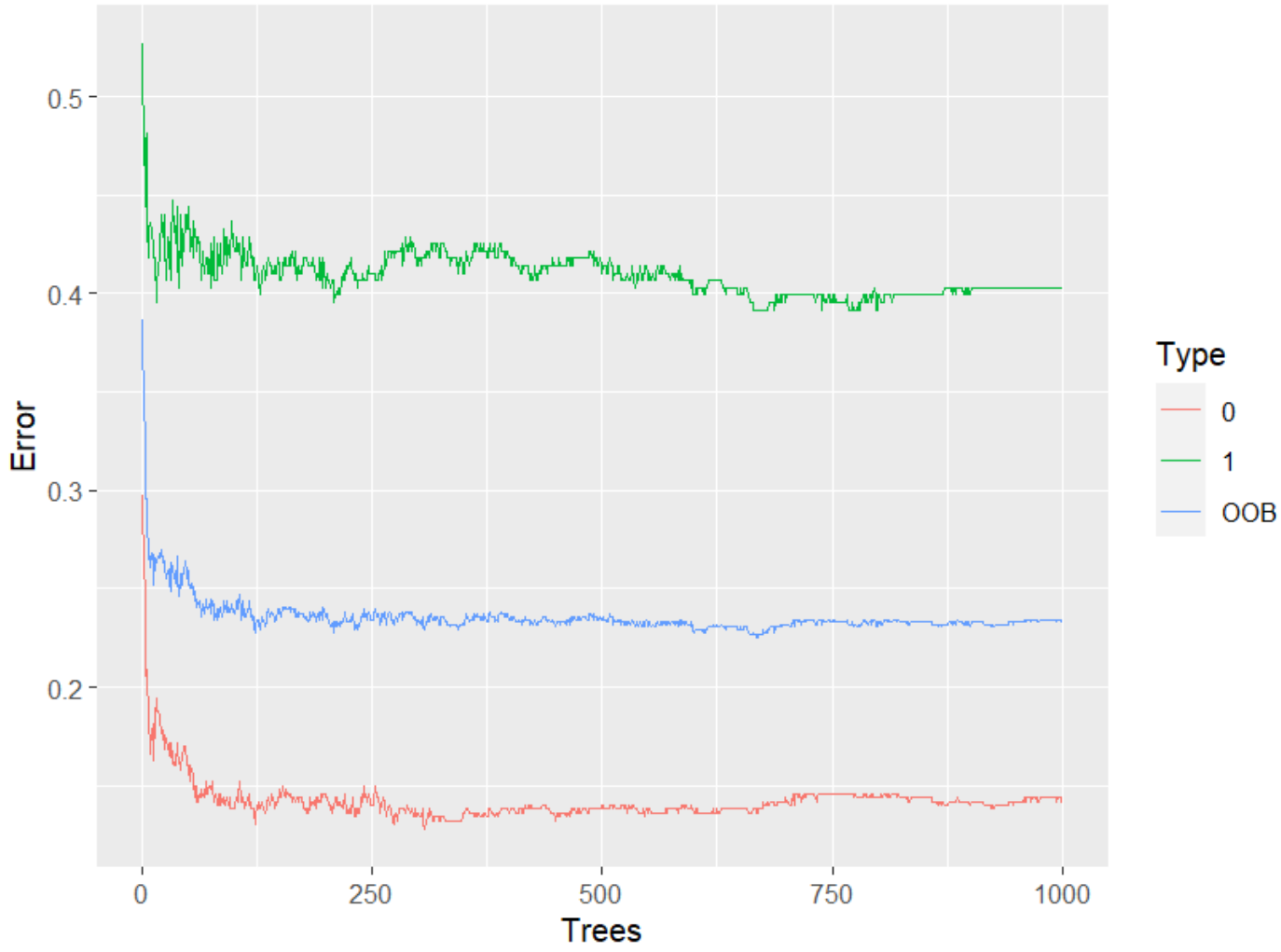
FIN du Problème III

4 Problème IV : Foret aléatoire, une application sur des données réelles :

Avant de choisir nos modèles de forêts aléatoires, commençons par choisir le nombre d'arbres et le nombre de variables de selection aléatoire dans la construction de nos forêts :

```
library(ggplot2)
library(cowplot)
set.seed(42)
md <- randomForest(class ~ ., data=data,
  proximity=TRUE, ntree=1000)
head(md$err.rate)
oob.error.data <- data.frame(
  Trees=rep(1:nrow(md$err.rate), times=3),
  Type=rep(c("OOB", "0", "1"), each=nrow(md$
    err.rate)),
  Error=c(md$err.rate[,"OOB"],
          md$err.rate[,"0"],
          md$err.rate[,"1"]))
ggplot(data=oob.error.data, aes(x=Trees, y=
  Error)) +
  geom_line(aes(color=Type))
```

R Output :



On remarque que l'out-of-bag error et l'erreur de classification des 1 et des 0 se stabilisent à partir d'un nombre d'arbres égale à 700, donc on choisira 700 arbres pour construire nos modèles de forêts.

Désormais, déterminons le nombre optimal de variables aléatoires à sélectionner afin de construire la forêt qui minimise l'erreur "Out-of-Bag" (OOB).

```
oob.values <- vector(length=10)
for(i in 1:10) {
  temp.model <- randomForest(class ~ ., data=
    data, mtry=i, ntree=1000)
  oob.values[i] <- temp.model$err.rate[nrow(
    temp.model$err.rate),1]
}
oob.values;which(oob.values == min(oob.values))
```

R Output : 2 4

Donc, on choisira mtree=2 pour la construction des forêts.

On va créer différents modèles à partir de ces résultats, et nous allons évaluer les performances de chaque modèle en les examinant en détail et les comparons entre eux. Et à la fin on donnera une comparaison globale des modèles.

4.1 Modèle 1: modèle simple avec out-of-sample-estimation

```
library(randomForest)
ranf2 <- randomForest(class ~ ., data = train,
  ntree = 700, mtry = 2)
print(ranf2)

# Variable importance
imp= ranf2$importance[order(ranf2$importance[,
  1], decreasing = TRUE), ]; imp
```

Random Forest:
Formula: class ∼ .
Data: train
Number of trees: 700
No. of variables tried at each split: 2
Type of random forest: classification
OOB estimate of error rate: 23.42%

Matrice de confusion :

	0	1	Erreur de classe
0	307	49	0.1376404
1	77	105	0.4230769

Variable Importance:

plas:	61.25658,	mass:	41.06847,	age:	32.71127
pedi:	31.24310,	preg:	20.75897,	pres:	20.31557
skin:	15.92046,	insu:	15.53154		

Remarque : Il ne faut pas confondre la "Confusion Matrix" présentée ici avec celle appliquée au données test, ici elle mesure la précision de la réponse moyenne des arbres créés **à partir des données d'apprentissage** (avec le même principe du calcul de l'OBB présenté dans le cours.)

On constate que le modèle classifie les non positives mieux que les cas positives, on valide ça par les données test juste après.

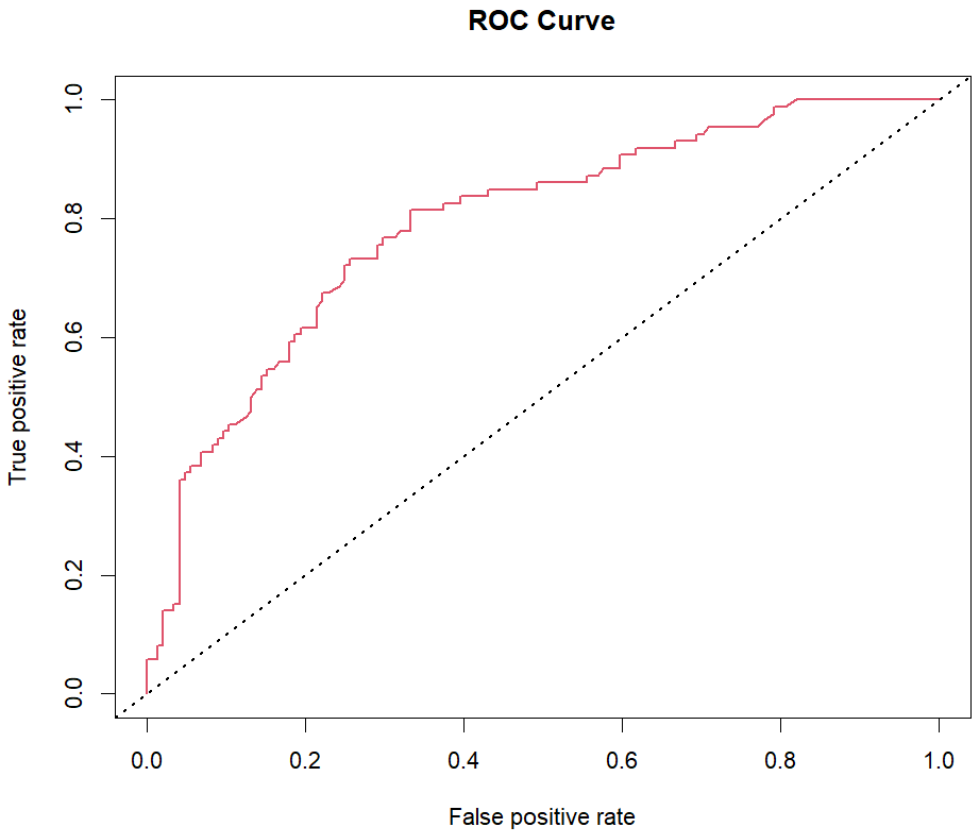
"plas", "mass", "pedi", et "age" sont les variables les plus déterminantes de l'etat du patient avec une certitude de:

$$\frac{61.25658+41.06847+32.71127+31.24310}{61.25658+41.06847+32.71127+31.24310+20.75897+20.31557+15.92046+15.53154} = 69.63\%$$

Maintenant évaluons la performance du modèle sur les données test:

```
#error
pr1 <- predict(ranf1, newdata = test)
mc1 <- table(pr1, test$class)
err1= 1-((mc1[1,1]+mc1[2,2])/sum(mc1))
print(paste("Pr cision :",(1-err1)*100,"%"))

# sensibilite et specificite
se1 = mc1[2,2]/(mc1[1,2]+mc1[2,2])
sp1 = mc1[1,1]/(mc1[1,1]+mc1[2,1])
print(paste("sensibilit  :",se1*100,"%"))
print(paste("specificit  :",sp1*100,"%"))

# ROC & auc
library(pROC)
Predprob1 <- predict(ranf1, newdata = test,type
    = "prob")
Predprob1 = as.data.frame(Predprob1)
```

```r
Prediction1 <- prediction(Predprob1[2],test$
  class)
performance1 <- performance(Prediction1, "tpr",
  "fpr")
plot(performance1,main = "ROC Curve",col = 2,
  lwd = 2)
abline(a = 0,b = 1,lwd = 2,lty = 3,col = "black
  ")
aucDT1 <- performance(Prediction1, measure = "
  auc")
aucDT1 <- aucDT1@y.values[[1]]
print(paste("area under curve :", aucDT1))
# point optimale
fpr1 = attr(performance1, "x.values")[[1]]
tpr1 = attr(performance1, "y.values")[[1]]
print(paste("point optimale :", closest_to_01(
  fpr1, tpr1)[1], closest_to_01(fpr1, tpr1)[2])
  )
```



R Output:
Précision : 72.1739130434783%
sensibilité : 47.6744186046512%
specificité : 86.8055555555556%
area under curve : 0.789526808785529
point optimal : (0.256944444444444, 0.732558139534884)

Donc on a bien trouvé que le modèle a une specificité très élevée par rapport à sa sensibilité, il permet bien de classifier les personnes non malades que les personnes malades.

Passant maintenant à un modèle qui prend en compte que les variables les plus importantes.

4.2 Modèle 2: Modèle ne prenant en compte que les variables les plus importants

On a vu dans le modèle précédent que "plas", "mass", "pedi", et "age" représentent presque 70 % de l'importance des variables. Essayons de construire un modèle avec une base de données des colonnes sélectionnées et evaluons sa performance sur les données test.
On crée deux bases de données select_train et select_test juste avec les colonnes selectionnées.

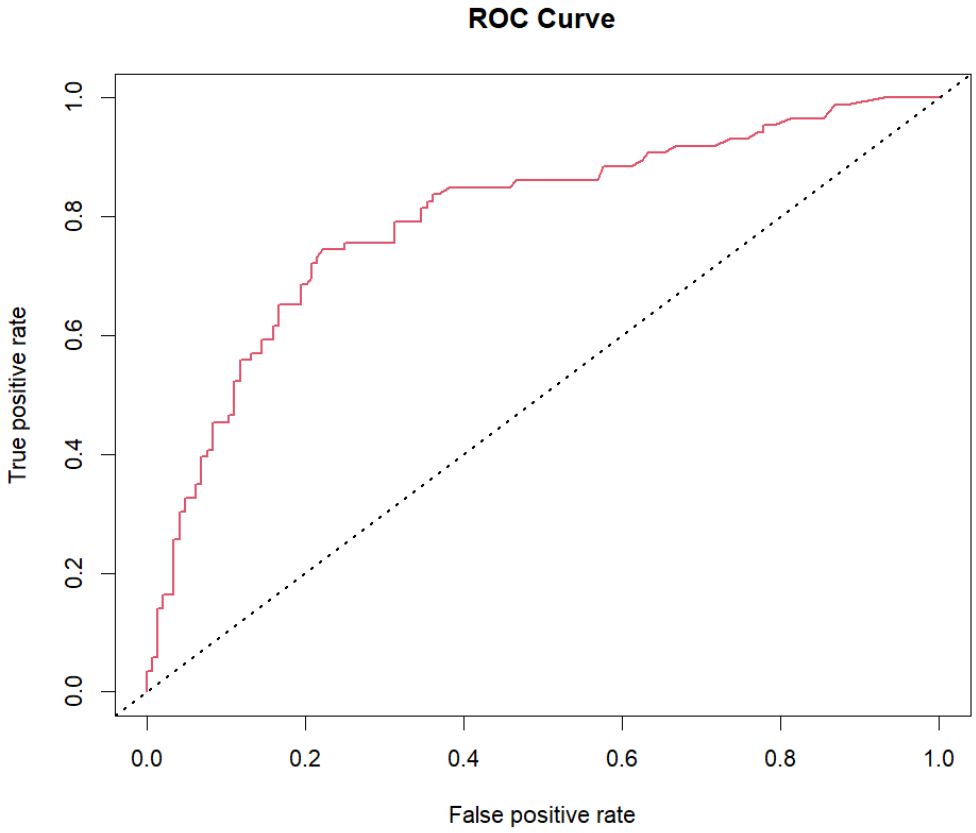
```
select_train= train[, c("plas","age","mass","
  pedi","class")]
formula <- class ~ plas+age+mass+pedi
ranf2<- randomForest(formula, data = select_
  train, ntree = 700, mtry = 2)
select_test= test[, c("plas","age","mass","pedi
  ","class")]
```

```
#error
pr2 <- predict(ranf2, newdata = select_test)
mc2 <- table(pr2, select_test$class)
err2= 1-((mc2[1,1]+mc2[2,2])/sum(mc2))
```

```r
print(paste("Pr cision :",(1-err2)*100,"%"))


# sensibilite et specificite
se2 = mc2[2,2]/(mc2[1,2]+mc2[2,2])
sp2 = mc2[1,1]/(mc2[1,1]+mc2[2,1])
print(paste("sensibilit  :",se2*100,"%"))
print(paste("specificit  :",sp2*100,"%"))

# ROC & auc
library(pROC)
Predprob2 <- predict(ranf2, newdata = test,type
    = "prob")
Predprob2 = as.data.frame(Predprob2)
Prediction2 <- prediction(Predprob2[2],select_
   test$class)
performance2 <- performance(Prediction2, "tpr",
   "fpr")
plot(performance2,main = "ROC Curve",col = 2,
   lwd = 2)
abline(a = 0,b = 1,lwd = 2,lty = 3,col = "black
   ")
```

```r
aucDT2 <- performance(Prediction2, measure = "
   auc")
aucDT2 <- aucDT2@y.values[[1]]
print(paste("area under curve :", aucDT2))
# point optimale
fpr2 = attr(performance2, "x.values")[[1]]
tpr2 = attr(performance2, "y.values")[[1]]
print(paste("point optimale :", closest_to_01(
   fpr2, tpr2)[1], closest_to_01(fpr2, tpr2)[2])
   )
```

Output:
Précision : 75.2173913043478%
sensibilité : 52.3255813953488%
specificité : 88.8888888888889%
area under curve : 0.795502260981912
point optimal : (0.222222222222222, 0.744186046511628)



Avant d'aborder le dernier modèle, un bref commentaire s'impose : ce modèle a affiché de meilleures performances sur l'ensemble des métriques par rapport au modèle précédent.

Cela pourra être du par le fait qu'on choisit aléatoirement 2 variables parmi les variables les plus performants au lieu de choisir 2 variables parmi tous les variables.

4.3 Modèle 3: Modèle avec 5-fold cross validation

Avant de procéder essayons d'éxpliquer le processus de ce modèle:

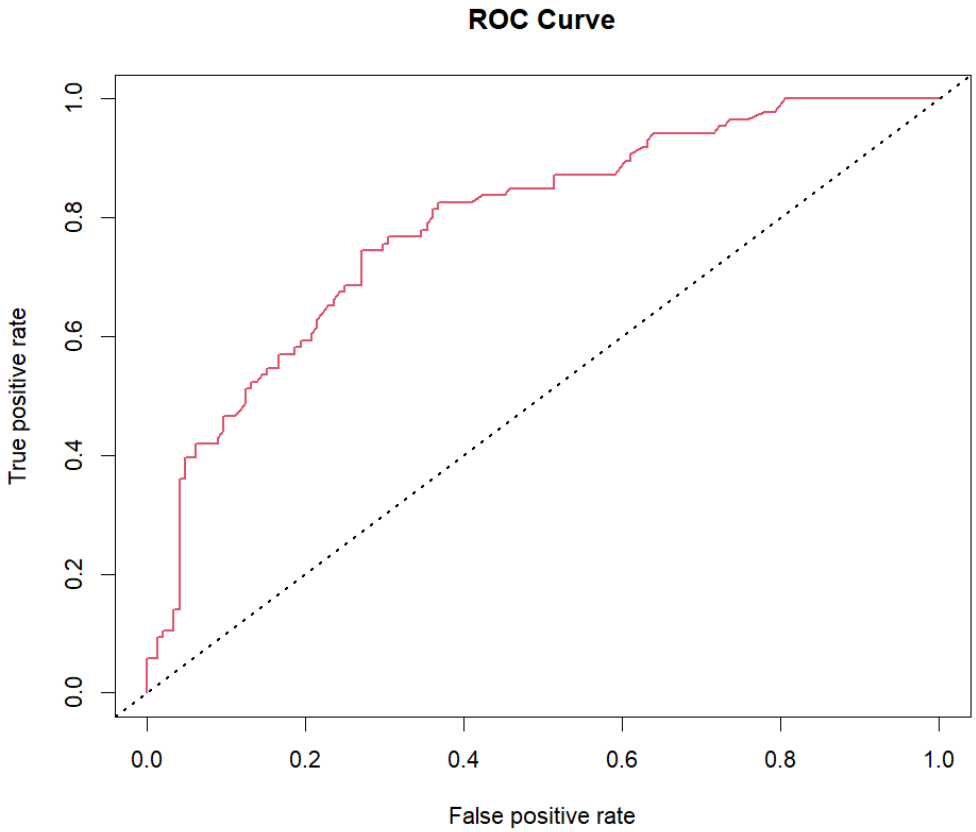
On divise notre base de données en un échantillion d'apprentissage et en un échantillion de test.

On prend notre base d'apprentissage et on effectue une cross validation sur elle: on divise notre base sur 5 parties; 4 pour construire le modèle et une pour la validation, on repète ce processus 5 fois avec 5 bases de données de validation différents.

Les metriques mésurées à chaque fois sont les moyenne des métriques obtenus sur chaque itération, et la decision de classification est celle qui est votée par la majorité d'itérations.

```
#cross validation k=5
trainControl <- trainControl(method = "cv",
  number = 5)
ranf3 <- train(class ~ ., data = train, method =
  "rf", trControl = trainControl)
pr3= predict(ranf3,newdata = test)

#error
mc3= confusionMatrix(pr3, test$class)
print(mc3)
acc3 <- mc3$overall["Accuracy"]
err3 = 1-acc3
print(paste("Pr cision :",(1-err3)*100,"%"))

#sensibilite et specificite
sp3 = mc3$table[1, 1] / sum(mc3$table[1, ])
se3 <- mc3$table[2, 2] / sum(mc3$table[2, ])
print(paste("sensibilit  :",se3*100,"%"))
print(paste("specificit  :",sp3*100,"%"))

# ROC & auc
library(pROC)
Predprob3 <- predict(ranf3, newdata = test,type =
   "prob")
```

```
Predprob3 = as.data.frame(Predprob3)
Prediction3 <- prediction(Predprob3[2],select_
  test$class)
performance3 <- performance(Prediction3, "tpr","
  fpr")
plot(performance3,main = "ROC Curve",col = 2,lwd
  = 2)
abline(a = 0,b = 1,lwd = 2,lty = 3,col = "black")
aucDT3 <- performance(Prediction3, measure = "auc
  ")
aucDT3 <- aucDT3@y.values[[1]]
print(paste("area under curve :", aucDT2))
# point optimale
fpr3 = attr(performance3, "x.values")[[1]]
tpr3 = attr(performance3, "y.values")[[1]]
print(paste("point optimale :", closest_to_01(
  fpr3, tpr3)[1], closest_to_01(fpr3, tpr3)[2]))
```



Output:

Précision : 73.4782608695652%

sensibilité : 70.4918032786885%

specificité : 74.5562130177515%

area under curve : 0.795502260981912

point optimal : (0.222222222222222, 0.744186046511628)

On peut voir dans ce modèle que la sensibilité a pu augmenter par rapport aux deux derniers modèles, alors que la spécificité a diminué, une chose assez normale puisque un gain de sensibilité est obtenu contre une perte de spéecificité.

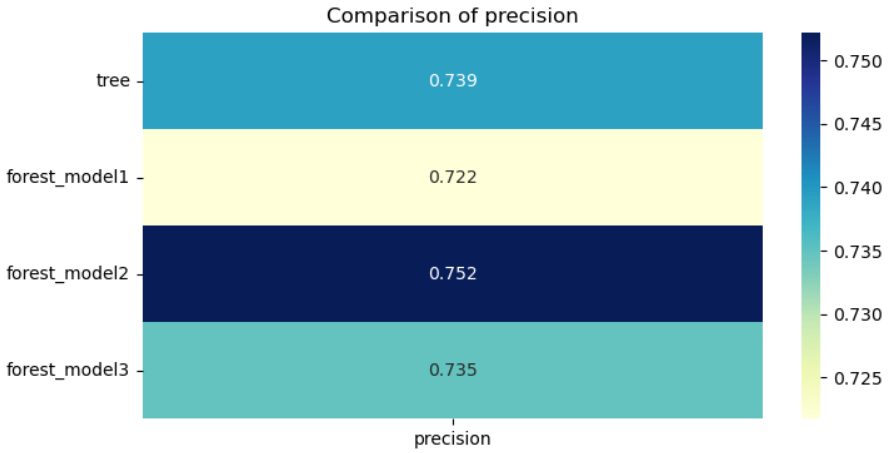
4.4 Comparaison globale entre les modèles

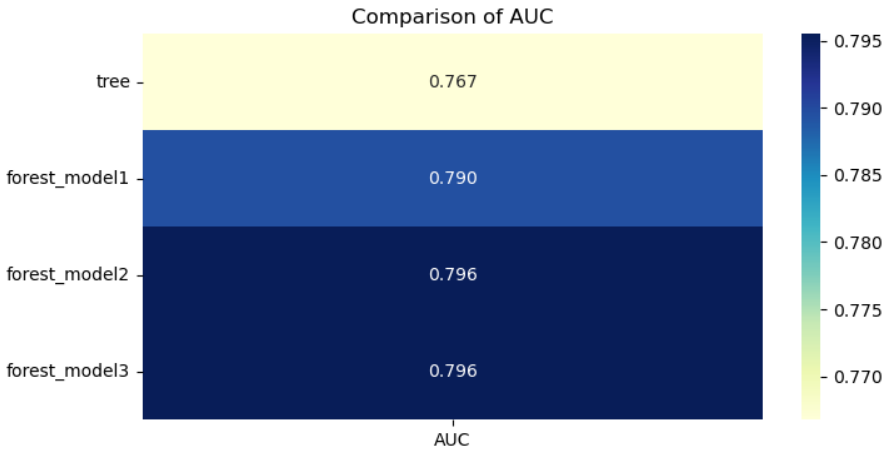
Pour une comparaison plus simple, on essaie de grouper nos métriques de performances dans des bases de données dans python et les visualiser à l'aide des Heatmaps.

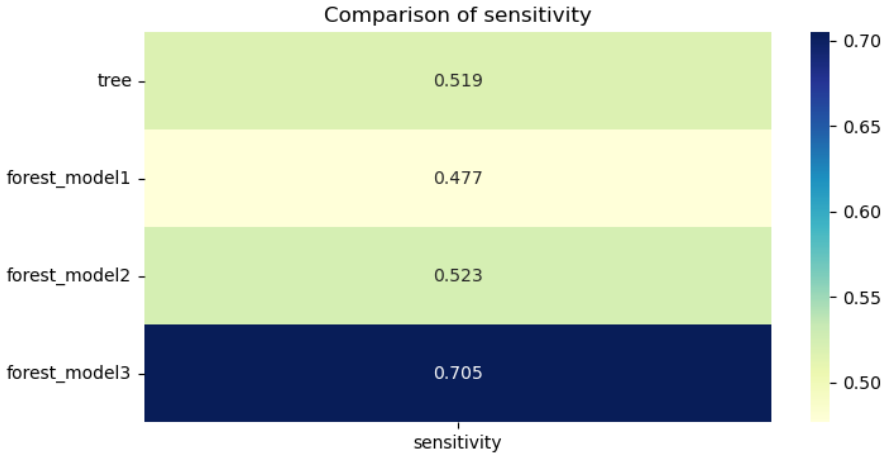
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

data = {
'tree': [0.739130434782609, 0.518518518518518,
0.838926174496644, 0.7667578],
'forest_model1': [0.721739130434783, 0.476744186046512,
0.868055555555556, 0.789526808785529],
```

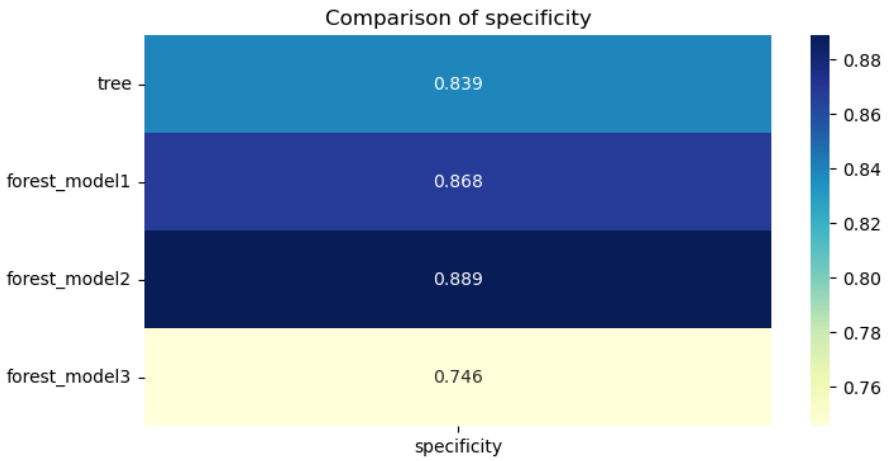
```
'forest_model2': [0.752173913043478, 0.523255813953488,
0.888888888888889, 0.795502260981912],
'forest_model3': [0.734782608695652, 0.704918032786885,
0.745562130177515, 0.795502260981912]
}
```

```
df = pd.DataFrame(data, index=['precision', 'sensitivity',
'specificity', 'AUC'])

for column in df.index:
    plt.figure(figsize=(8, 4))
    sns.heatmap(df.loc[[column], :].T,annot=True,cmap='YlGnBu',
    fmt=".3f", cbar=True)
    plt.title(f'Comparison of {column}')
    plt.show()
```


Comparison of precision
tree
forest_model1
forest_model2
forest_model3
0.739
0.722
0.752
0.735
precision
0.750
0.745
0.740
0.735
0.730
0.725


Comparison of AUC
tree
forest_model1
forest_model2
forest_model3
0.767
0.790
0.796
0.796
AUC
0.795
0.790
0.785
0.780
0.775
0.770


Comparison of sensitivity
tree
forest_model1
forest_model2
forest_model3
0.519
0.477
0.523
0.705
sensitivity
0.70
0.65
0.60
0.55
0.50


Comparison of specificity
tree
forest_model1
forest_model2
forest_model3
0.839
0.868
0.889
0.746
specificity
0.88
0.86
0.84
0.82
0.80
0.78
0.76

Remarques :

Pour la précision le deuxième modèle des forêts avec les prédicteurs sélectionnés était le plus performant. Pendant que le modèle brute pour les forêts était le moins précis.

Pour l'AUC les deux derniers modèles des forêts étaient les plus perforamnts, cad qu'ils ont une capacité plus forte d'assigner des pronostics probabilistes à la classe correcte.

Avec une large marge des autres, le 3eme modèle des forêts avec cross validation a une capacité plus forte que les autres à identifier les individus malades.

Le modèle d'arbre du problème II et les deux premiers modèles des forêts ont une capacité très forte à identifier les personnes non malade (de class 0) par rapport au 3ème modèle des forêts, qui quand même a une spécifité élevée.

En genéral le modèle de cross validation est le modèle qui a pu le mieux modérer entre sa capacité de distinguer les individus positives et les individus négatives.

N.B : Enfin il faut noter, que dans ce TP, on a expérimenté que sur une seule division de base de données d'entrainement et de test. Pour une mise au point des prédictions, il vaut mieux performer les différents modèles présentés ci-dessus pour plusieurs divisions aléatoires de la base de données en données entrainement et données test.

FIN du Problème IV

FIN