ACT6100 H2019	Analyse de données Solutions 7

1. (a) La Table 1 explicite comment est déterminée la première règle. La première règle est alors

$$r_1(\mathbf{X}_i) = \begin{cases} -1, & x_{i1} < -2 \\ 1, & x_{i1} > -2 \end{cases}$$

ou

$$r_1(\mathbf{X}_i) = \begin{cases} 1, & x_{i1} < 2 \\ -1, & x_{i1} > 2. \end{cases}$$

En utilisant la première, on obtient une erreur apparente

$$\begin{aligned} \epsilon_1 &= \sum_{i:r_1(\mathbf{X}_i)\neq Y_i} p_{1i}(\mathbf{X}_i) \\ &= 1/8 + 1/8 = 0.25 \end{aligned}$$

et $\alpha_1 = 0.5\ln((1-0.25)/0.25) \approx 0.5493$. Les valeurs prédites par la règle r_1 et les nouveaux poids sont présentés dans la Table 2.

j	Intervalle pour c	Meilleur taux de classement
1	$(-\infty, -3)$	0.50
	$(-3, -1)$	0.75
	$(-1, 1)$	0.50
	$(1, 3)$	0.75
	$(3, \infty)$	0.50
2	$(-\infty, -1)$	0.50
	$(-1, 1)$	0.50
	$(1, \infty)$	0.50

TABLE 1 – Base de données.

i	X_{i1}	X_{i2}	Y_i	$r_1(\mathbf{X}_i)$	p_{i2}
1	-3	-1	-1	-1	1/12
2	-3	1	-1	-1	1/12
3	3	-1	-1	1	3/12
4	3	1	-1	1	3/12
5	-1	-1	1	1	1/12
6	-1	1	1	1	1/12
7	1	-1	1	1	1/12
8	1	1	1	1	1/12

TABLE 2 – Base de données.

(b) La Table 3 explicite comment est déterminée la deuxième règle. La deuxième règle est alors

$$r_2(\mathbf{X}_i) = \begin{cases} 1, & x_{i1} < 2 \\ -1, & x_{i1} > 2. \end{cases}$$

En utilisant la première, on obtient une erreur apparente

$$\begin{aligned} \epsilon_2 &= \sum_{i:r_2(\mathbf{X}_i)\neq Y_i} p_{i2}(\mathbf{X}_i) \\ &= 1/12 + 1/12 = 2/12 \end{aligned}$$

j	Intervalle pour c	Meilleur taux de classement
1	$(-\infty, -3)$	8/12
	$(-3, -1)$	6/12
	$(-1, 1)$	8/12
	$(1, 3)$	10/12
	$(3, \infty)$	8/12
2	$(-\infty, -1)$	8/12
	$(-1, 1)$	6/12
	$(1, \infty)$	8/12

TABLE 3 – Base de données.

et $\alpha_2 = 0.5 \ln((1 - 2/12)/(2/12)) \approx 0.804719$. Ainsi, on obtient comme règle agrégée

$$R(\mathbf{X}_i) = \begin{cases} 1, & \alpha_1 r_1(\mathbf{X}_i) + \alpha_2 r_2(\mathbf{X}_i) > 0 \\ -1, & \alpha_1 r_1(\mathbf{X}_i) + \alpha_2 r_2(\mathbf{X}_i) < 0. \end{cases}$$

2. (a) Le code est présenté à la Figure 1.

```
dataSPAM <- read.table("...", quote="\"", comment.char="")
colnames(dataSPAM)[58] <- "Y"

set.seed(1001)
indice <- sample(1:length(dataSPAM$V1), 2300, replace = FALSE)
dataTRAIN <- dataSPAM[indice,]
dataTEST <- dataSPAM[-indice,]
```

FIGURE 1 – Code informatique.

(b) Le code est présenté à la Figure 2.

```
K <- seq(1, 50, by = 1)
err <- K
ind <- 0

for (i in K)
  {
  ind <- ind + 1
  mod_ppv <- knn.reg(train = as.matrix(dataTRAIN[,-58]),
             test = as.matrix(dataTEST[, -58]),
             y = dataTRAIN$Y, k = K[ind])$pred
  err[ind] <- sum(mod_ppv != dataTEST$Y)/nrow(dataTEST)
  }

min(err)

0.1868753
```

FIGURE 2 – Code informatique.

(c) Le code est présenté à la Figure 3.

(d) Le code est présenté à la Figure 4.

(e) La Figure 5 présente les différents résultats. Sans surprise, la méthode des K plus proches

```
arbre <- rpart(as.factor(Y)~. , control = rpart.control(cp = 0),
               data = dataTRAIN)
CP <- arbre$cptable[which.min(arbre$cptable[,4]), 1]

arbre1 <- rpart(as.factor(Y)~., control = rpart.control(cp = CP),
                data = dataTRAIN)
prev_arbre <- predict(arbre1, newdata = dataTEST, type="class")
err_arbre <- sum(prev_arbre != dataTEST$Y)/nrow(dataTEST)
err_arbre

0.09995654
```

FIGURE 3 – Code informatique.

```
mod_RF1 <- randomForest(as.factor(Y)~., data = dataTRAIN,
                        ntree = 2500,
                        xtest = dataTEST[,-58],
                        ytest = as.factor(dataTEST[,58]))
```

FIGURE 4 – Code informatique.

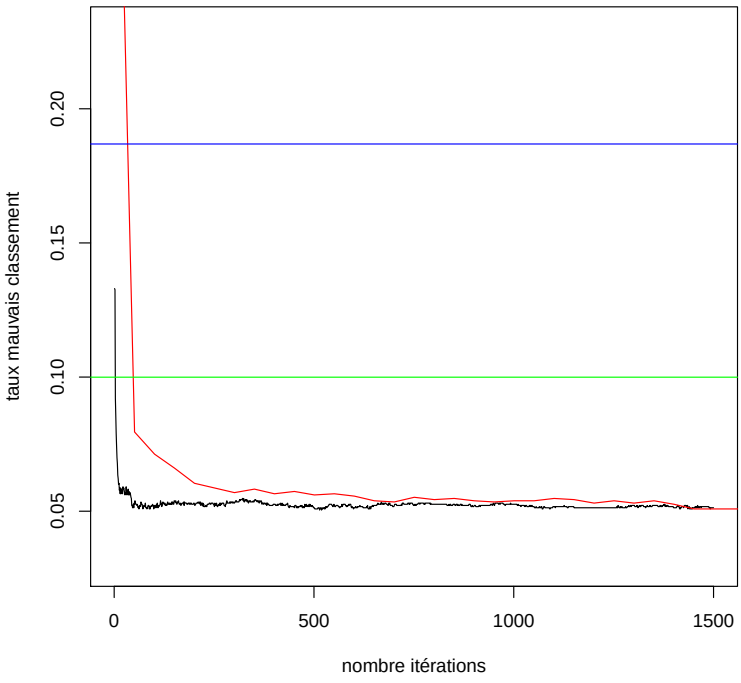

FIGURE 5 – Taux de mauvais classement pour la méthode des K plus proches voisins (bleu), la méthode des arbres de décision (vert), le *Random Forests* (noir) et l'algorithme *AdaBoost* (rouge).

voisins est la plus mauvaise (on a vu qu'elle n'était pas très intéressante lorsque l'espace des variables explicatives est en dimension élevée). On voit que les deux méthodes d'agrégation de modèles (*Random Forests* et *AdaBoost*) permettent d'améliorer nettement la performance de l'arbre seul. L'algorithme du *Random Forests*, dans cet exemple en particulier, se stabilise très rapidement : on pourrait arrêter après 100 ou 200 itérations. Enfin, la performance des

deux méthodes d'agrégation de modèles (*Random Forests* et *AdaBoost*) est similaire dans cet exemple.

3. (a) On a

$$\begin{aligned}(\alpha_t, R_t) &= \arg\min_{(\alpha,r)\in\mathbb{R}\times\mathcal{R}} \sum_{i=1}^{n} e^{-Y_i(r_{t-1}(\mathbf{X}_i)+\alpha r(\mathbf{X}_i))} \\ &= \arg\min_{(\alpha,r)\in\mathbb{R}\times\mathcal{R}} \sum_{i=1}^{n} e^{-Y_i r_{t-1}(\mathbf{X}_i)} e^{-Y_i \alpha r(\mathbf{X}_i)} \\ &= \arg\min_{(\alpha,r)\in\mathbb{R}\times\mathcal{R}} \sum_{i=1}^{n} w_i^t e^{-Y_i \alpha r(\mathbf{X}_i)} \\ &= \arg\min_{(\alpha,r)\in\mathbb{R}\times\mathcal{R}} \left(e^{\alpha} - e^{-\alpha}\right) \sum_{i:r(\mathbf{X}_i)\neq Y_i} w_i^t + e^{-\beta} \sum_{i=1}^{n} w_i^t.\end{aligned}$$

Je vais noter cette dernière équation $\mathcal{O}$ pour la suite. Ainsi, les poids sont donnés par

$$w_i^t = e^{-Y_i r_{t-1}(\mathbf{X}_i)}.$$

(b) i. Si on suppose $\alpha > 0$ connu, le problème d'optimisation devient

$$\begin{aligned}R_t(\mathbf{X}_i) &= \arg\min_{r\in\mathcal{R}} \sum_{i=1}^{n} e^{-Y_i(r_{t-1}(\mathbf{X}_i)+\alpha r(\mathbf{X}_i))} \\ &= \arg\min_{r\in\mathcal{R}} \left(e^{\alpha} - e^{-\alpha}\right) \sum_{i:r(\mathbf{X}_i)\neq Y_i} w_i^t + e^{-\beta} \sum_{i=1}^{n} w_i^t \\ &= \arg\min_{r\in\mathcal{R}} \sum_{i=1}^{n} w_i^t \mathbb{I}_{r(\mathbf{X}_i)\neq Y_i}.\end{aligned}$$

Ainsi, R_t est obtenue en minimisant l'erreur de prédiction pondérée par les poids w_i^t.

ii. Si on considère r connue, on a

$$\begin{aligned}\frac{\partial \mathcal{O}}{\partial \alpha} &= \left(e^{\alpha} + e^{-\alpha}\right) \sum_{i:r(\mathbf{X}_i)\neq Y_i} w_i^t - e^{-\alpha} \sum_{i=1}^{n} w_i^t = 0 \\ \rightarrow \alpha + \ln\left(\sum_{i:r(\mathbf{X}_i)\neq Y_i} w_i^t\right) &= -\alpha + \ln\left(\sum_{i=1}^{n} w_i^t\right) \\ \rightarrow \alpha_t &= 0.5 \ln\left(\frac{\sum_{i:r(\mathbf{X}_i)=Y_i} w_i^t}{\sum_{i:r(\mathbf{X}_i)\neq Y_i} w_i^t}\right).\end{aligned}$$

(c) Si on regroupe les deux étapes d'optimisation précédentes, on obtient

$$\alpha_t = 0.5 \ln\left(\frac{\sum_{i:R_t(\mathbf{X}_i)=Y_i} w_i^t}{\sum_{i:R_t(\mathbf{X}_i)\neq Y_i} w_i^t}\right),$$

que l'on peut réécrire comme étant

$$\alpha_t = 0.5 \ln\left(\frac{1 - E_t}{E_t}\right),$$

où

$$E_t = \frac{\sum_{i:R_t(\mathbf{X}_i)\neq Y_i} w_i^t}{\sum_{i=1}^{n} w_i^t} = \sum_{i:R_t(\mathbf{X}_i)\neq Y_i} w_i^t$$

si on suppose que les w_i^t sont des poids qui somment à 1. Ainsi, les E_t peuvent être considérées comme les erreurs apparentes calculées à partir des poids w_i^t et on retrouve une expression similaire à celle de l'étape 2c de l'algorithme.

Il faut maintenant s'assurer que la mise à jour des poids correspond à celle indiquée à l'étape 2d de l'algorithme :

$$\begin{aligned} w_i^{t+1} &= e^{-Y_i r_t(\mathbf{X}_i)} \\ &= e^{-Y_i(r_{t-1}(\mathbf{X}_i)+\alpha_t R_t(\mathbf{X}_i))} \\ &= w_i^t e^{-Y_i \alpha_t R_t(\mathbf{X}_i)} \\ &= \begin{cases} w_i^t e^{-\alpha_t}, & R_t(\mathbf{X}_i) = Y_i \\ w_i^t e^{\alpha_t}, & R_t(\mathbf{X}_i) \neq Y_i. \end{cases} \end{aligned}$$

4. Le code permettant d'ajuster les modèles et de réaliser les graphiques est présenté à la Figure 6. Les graphiques obtenus sont présentés aux Figures 7 et 8. Avec un taux d'apprentissage de 1,

```
### Je décide de faire un maximum de 7000 itérations
T <- 8000
modele1 <- gbm(Y~X, data = data, distribution = "adaboost", interaction.depth = 1,
               shrinkage = 1, n.trees = T)
modele2 <- gbm(Y~X, data = data, distribution = "adaboost", interaction.depth = 1,
               shrinkage = 0.01, n.trees = T)

### Code pour le graphique du modèle 1
indice <- seq(1, T, by = 100)
errapp <- rep(0, length(indice))
errtest <- rep(0, length(indice))
k <- 0
for (i in indice)
{
  k <- k+1
  pred_app <- predict(modele1, newdata = data, n.trees = i)
  errapp[k] <- sum(as.numeric(pred_app > 0) != data$Y)/nrow(data)
  pred_test <- predict(modele1, newdata = datavalid, n.trees = i)
  errtest[k] <- sum(as.numeric(pred_test > 0) != datavalid$Y)/nrow(datavalid)
}
plot(indice, errapp, type = "l", col = "black", ylim = c(0, 0.4),
     xlab = "itérations", ylab = "erreur", main = "alpha = 1")
lines(indice, errtest, col = "red")

### Le code pour le graphique du modèle 2 est similaire.
```

FIGURE 6 – Code informatique.

on remarque que le modèle apprend trop rapidement : la courbe rouge saute dès le départ à un niveau d'environ 0.4 (situation de surajustement) et reste plus ou moins stable à ce niveau par la suite. Avec un taux d'apprentissage de 0.01, on remarque que cette montée est beaucoup plus progressive et permet de sélectionner un nombre optimal d'itérations.

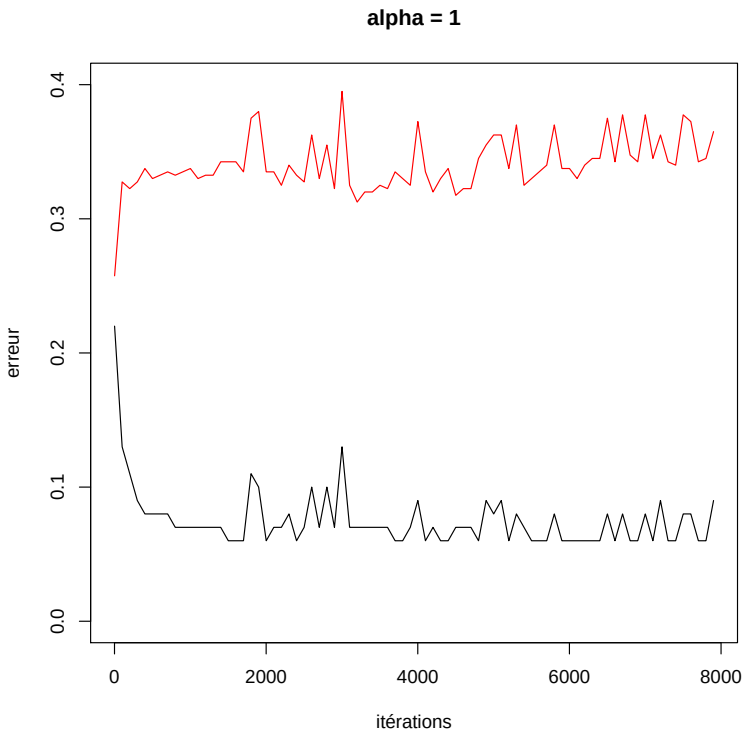

FIGURE 7 – Proportion de mauvaise classification en fonction du nombre d'itérations calculée sur la base d'entrainement (noir) et sur la base de validation (rouge) pour un taux d'apprentissage de 1.

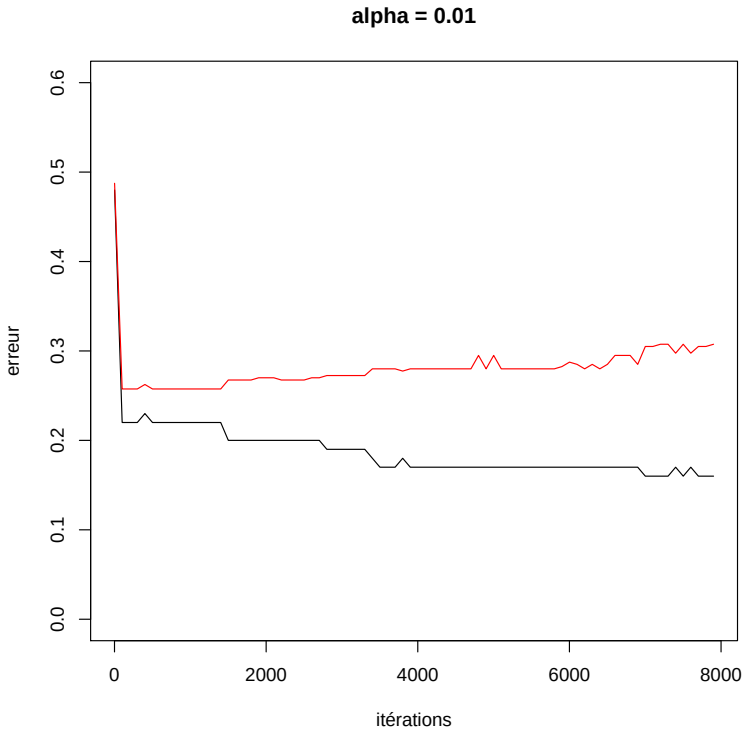

FIGURE 8 – Proportion de mauvaise classification en fonction du nombre d'itérations calculée sur la base d'entrainement (noir) et sur la base de validation (rouge) pour un taux d'apprentissage de 0.01.