GLM - Data challenge Epismoke 2.1

Garance Malnoë, Matthias Mazet, Léonie Breuzat

2024-12-17

Contents

Code de mise en forme des titres :

- "#" => ***titre***
- "##" => ***sous-titre***
- "###" => **sous-sous-titre**

!! Attention dans les blocs de code aux lignes trop longues !!

Ce data challenge consiste à prédire le statut tabagique d'individu : actuellement fumeur (`current`), ancien fumer (`former`) ou n'ayant jamais fumé (`never`). Pour cela, nous allons étudier trois méthodes différentes que nous entraînerons sur le jeu de données `data_train` et que nous testerons sur le jeu de données `data_test`.

```
data_train <- readRDS(file = "data_train.rds") # Données de train
data_test <- readRDS(file = "data_test.rds") # Données de test
dim(data_train)
dim(data_test)
kable(head(data_train[,1:6]), align = "r")
```

Les deux jeux de données `data_train` et `data_test` sont composés de respectivement 421 et 200 individus et de 10 004 variables :

- `smoking_status` : variable qualitative à prédire, prenant les modalités `current`, `former` et `never`.
- `never01`, `former01`, `current01` : les transformations en variables binaires de la variable `smoking_status` qui vont nous permettre de construire les modèles.
- `cg########` : variables quantitatives (sondes) représentant le taux de méthylation de différents gènes. Elles joueront ici le rôle des variables explicatives.

1 *Prétraitement*

Commençons par un pré-traitement qui sera commun aux trois modèles.

Récupérons d'abord le noms des colonnes correspondants aux sondes (variables explicatives).

```
# Nom des sondes
probes <- colnames(data_train)[5:length(data_train)]
probes[1:6]
```

Nous pouvons tracer leur densité générale :

Sur le graphique, nous pouvons voir apparaître deux pics : un premier dans l'intervalle $[0; 0.2]$ (absence de méthylation) et un second dans l'intervalle $[0; 0.7]$ (présence de méthylation).

Regardons la distribution des statuts tabagiques :

Les 3 statuts sont présents de manière presque équidistribuée. Si on choisissait le modèle nul renvoyant dans tous les cas `former` (le statut le plus courant), on pourrait donc s'attendre à un taux d'erreur autour de 65%.

Pour entraîner nos modèles, faire de l'hyperparamétrage et pouvoir les tester, nous allons séparer le jeu de données `data_train` en deux parties : une partie pour l'entraînement `data_train1` (75%) et une partie pour le test `data_train2` (25%).

```
set.seed(1) # Pour avoir des résultats aléatoire reproductibles.

data_train1 <- data_train[sample(round(nrow(data_train) * 0.75)), ]
data_train2 <- data_train[setdiff(rownames(data_train), rownames(data_train1)), ]
```

Nous pouvons remarquer que les proportions de chaque groupe (chaque statut tabagique) ne sont pas exactement les mêmes entre `data_train1` et `data_train2` :

Tri des sondes à revoir

Tri sondes à revoir Pour les trois modèles, nous n'allons pas utiliser toutes les sondes pour construire nos modèles pour éviter le sur-ajustement et parce que tout simplement certaines sondes n'apportent pas ou peu d'informations. Pour sélectionner les sondes, nous allons construire les modèles linéaires de la forme `lm(sonde~smoking_status)` pour chaque sonde. Cela nous permettra d'étudier la significativité du lien entre le statut tabagique et chaque sonde à partir de la p-valeur associée au coefficient directeur et ainsi de classer les sondes.

```
siscreening <- function(data_train) {
  probes <- colnames(data_train)[5:length(data_train)]  # Noms des sondes
  pval_fisher <- c()
  r2 <- c()
  for (p in probes) {
    # On crée un modèle linéaire sonde ~ smoking_status
    m <- lm(data_train[,p] ~ data_train[, "smoking_status"])
    # On récupère la p-valeur du modèle
    pval_fisher <- c(pval_fisher, anova(m)[1,5])
    # On récupère le r2 associé à ce modèle
    r2 <- c(r2, summary(m)$r.squared)
  }
  names(pval_fisher) <- probes
  names(r2)          <- probes
  return(data.frame(pval_fisher = pval_fisher, r2 = r2))
}

# On mémoise la fonction pour mettre le résultat en cache.
if (!exists("msiscreening")) {msiscreening = memoise::memoise(siscreening)}
sis_res <- msiscreening(data_train1) # Dataframe résultat
```

```r
# Graphe R2 et p-value
plot(sis_res$r2, -log10(sis_res$pval),
     main = "-log10(p-value) des modèles linéaire à une sonde en fonction du R2",
     xlab = "R²", ylab = "-log10(p-value)")
```

On observe une relation logarithmique entre la p-valeur et le R^2 : quand la p-valeur devient très petite (-log10(p_val) grand), le R^2 augmente.

On tri les sondes par ordre croissant de p-valeur.

```r
# Tri des sondes en fonction de la p-value
sis_probes <- rownames(sis_res)[order(sis_res$pval_fisher)]
head(sis_res[sis_probes, ])
# Commentaire : ils ont tous un R2 proche de 0.1, c'est mieux que rien mais c'est pas top.
```

2 *Méthodes étudiées*

Nous allons étudier trois méthodes différentes pour ce data challenge : un arbre de décision construit à partir de modèles linéaires, un algorithme Elastic Net et un algorithme XGBoost. Nous détaillons chacune de ces méthodes dans les parties suivantes.

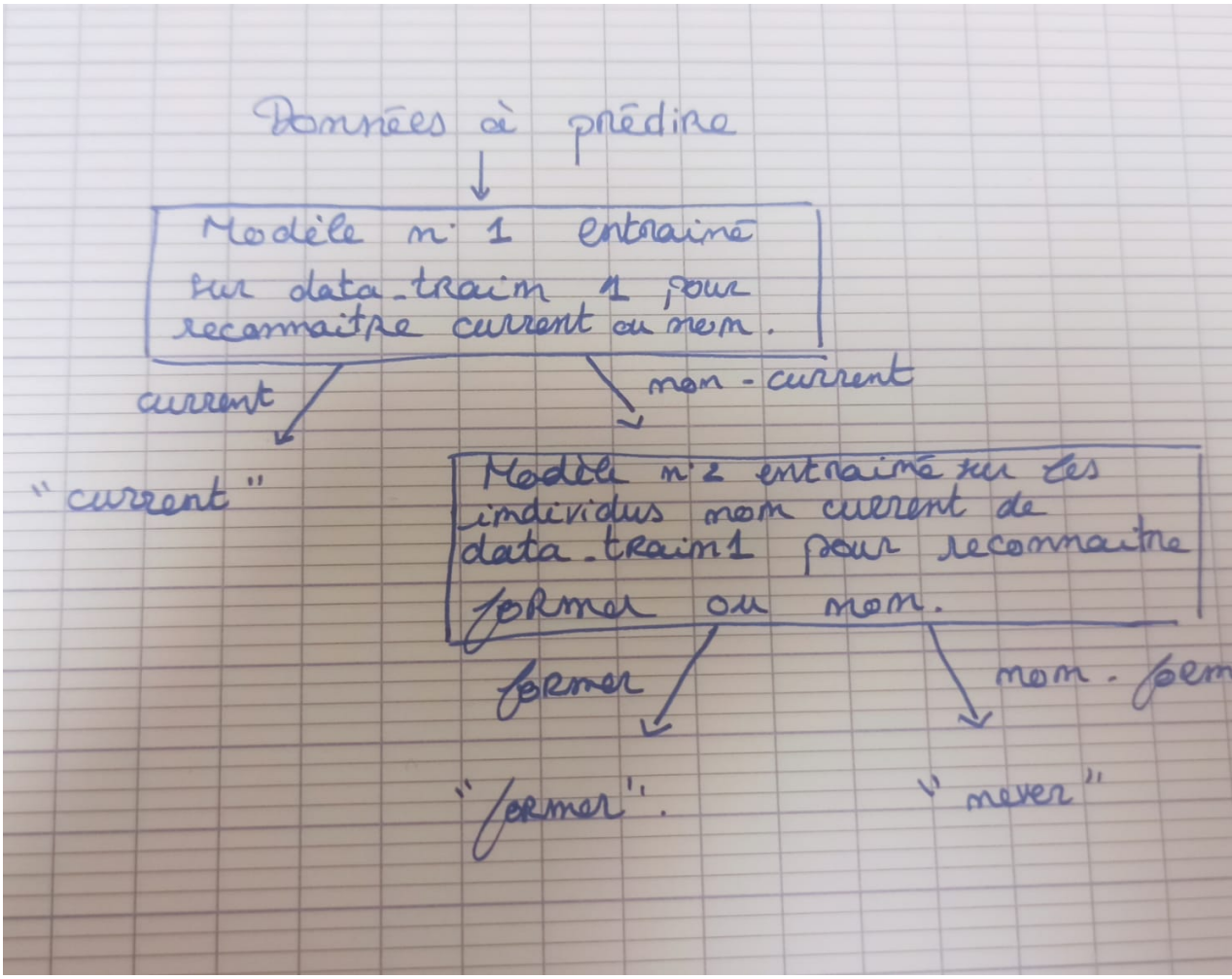
2.1 *Arbre de décision à partir de modèles linéaires*

2.1.1 Idée du modèle

L'idée de ce modèle se construit en trois étapes : - d'abord, un premier modèle permettant de séparer les individus en deux groupes selon un des trois statuts tabagiques ;

- ensuite, un second modèle entraîné uniquement sur les données des individus des deux autres statuts pour à nouveau séparer les individus restants dans deux groupes selon un des deux statuts tabagiques restants ;
- finalement, combiner ces deux modèles pour n'en former qu'un seul : un arbre de décision.

Voici un exemple :



2.1.2 Premier modèle

Nous allons construire pour les trois statuts tabagiques (`never`, `current` et `former`) le premier modèle ayant pour but de prédire si un individu est du statut considéré ou non. Pour chacun de ces modèles, nous allons également réaliser

une études des meilleurs hyperparamètres (nombres de sondes et seuil de décision) pour obtenir le meilleur modèle possible. Finalement, nous comparerons les trois modèles sur leur précision (proportion de prédiction correctes).

Commençons avec le statut `current`. Nous utilisons la variable binaire `current01` pour construire le modèle.

```
# Fonction pour créer un modèle linéaire avec les i meilleures sondes
model_current_sis_i <- function(data_train, i, screening_func=msiscreening){
  formula <- as.formula(paste0(c("current01 ~ 1", sis_probes[0:i]), collapse = "+"))
  m <- glm(formula, data_train, family = binomial(link = "logit"))
  return(m)
}

# On regarde le sur apprentissage sur les 50 sondes
iap1 <- c()
iap2 <- c()
for (i in 0:50) {
  m <- model_current_sis_i(data_train1, i)
  pred_train1 <- predict.glm(m, data_train1, type = "response")
  pred_train2 <- predict.glm(m, data_train2, type = "response")
  iap1 <- c(iap1, sum(ifelse(pred_train1 > 0.5, 1, 0) != data_train1$current01) / nrow(data_train1))
  iap2 <- c(iap2, sum(ifelse(pred_train2 > 0.5, 1, 0) != data_train2$current01) / nrow(data_train2))
}
plot(iap1, col = 4, pch = 16, cex = 1,
     main = "Incorrect Answers Proportion en fonction du nombre de sondes sur les deux jeux de données",
     xlab = "Nombre de sondes", ylab = "Incorrect Answers Proportion", cex.main = 0.8)
points(iap2, col = 2, pch = 16, cex = 1)
legend("bottomright", c("iap1", "iap2"), col = c(4, 2), pch = 16, cex = 0.8)
i <- 8
abline(v = i, col = 2)
```

On voit sur la figure précédente qu'en prenant plus de 8 sondes l'erreur diminue sur le jeu de données `data_train1` mais qu'il augmente sur le jeu de données `data_train2` : il y a du sur-apprentissage. Certaines sondes sont corrélées et n'apporte rien. Pour sélectionner les meilleures sondes, nous allons utiliser la fonction `step` qui sélectionne les sondes à partir du critère d'Akaike (AIC).

```
# Fonction de création de modèles step pour CURRENT
stepforward_current <-function(data_train, sis_probes, nb_sis_probes=200, trace=0, k=2) {
  m_lo <- glm(current01 ~ 1, data = data_train[, c("current01", sis_probes[1:nb_sis_probes])])
  m_sup <- glm(current01 ~ ., data = data_train[, c("current01", sis_probes[1:nb_sis_probes])])
  m_fwd <- step(m_lo, method = "forward", scope = list(upper = m_sup, lower = m_lo), trace = trace, k = k)
  return(m_fwd)
}
```

Testons les résultats sur `data_train2` avec le modèle `step` obtenu à partir des 90 premières sondes.

```
# Exemple Modèle step obtenu à partir des 90 meilleurs sondes.
step_model_current_90 <- stepforward_current(data_train1, sis_probes, nb_sis_probes = 90,trace=0)

# Test de l'accuracy sur le step_model_current_90
predicted_probs <- predict(step_model_current_90, newdata = data_train2, type = "response")
predicted_classes <- ifelse(predicted_probs > 0.5, 1, 0) # On pose le seuil à 0.5
confusion_matrix <- table(Predicted = predicted_classes, Actual = data_train2$current01)
print(confusion_matrix)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(paste("Précision:", accuracy))
```

Les résultats sont plutôt bons, on a une précision de 80,9% mais ce résultat est sans doute améliorable. En effet, on peut jouer sur deux hyperparamètres : le nombre de sondes utilisées pour former le modèle avec `step` et le seuil de

décision (le modèle prédit une valeur numérique, on avait choisit arbitrairement que si la valeur était supérieur à 0.5 alors l'individu appartenait à la classe `current`).

Pour faire cela, on va tout d'abord définir une fonction générale de prédiction qui prend en argument des données dont on veut prédire la classe, un modèle (pour faire varier le nombre de sonde) et un seuil.

```
# Fonction générale de prédiction à partir d'un modèle step + seuil de décision
predict_current <- function(data,step_model, seuil){
  predicted_probs <- predict(step_model, newdata = data, type = "response")
  predicted_classes <- ifelse(predicted_probs > seuil, 1, 0)
  return(predicted_classes)
}
```

En suite, on crée les modèles pour les différents nombres de sondes. On choisir de prendre 50 à 200 sondes par pas de 10. Cela fait 16 possibilités.

```
# Modèles pour différents nombres de sondes 50 à 200 avec un pas de 10.
create_modeles_current <- function(from = 50, to = 200, by = 10) {
  res <- list()
  for (i in seq(from = from, to = to, by = by)) {
    res[[length(res) + 1]] <- stepforward_current(data_train1, sis_probes, nb_sis_probes = i, trace = 0)
  }
  return(res)
}
modeles_current <- create_modeles_current()
```

On crée une fonction pour automatiser l'hyperparamétrage, elle va tester toutes les combinaison de modèles et de seuil et renvoyer la meilleur précision obtenue sur `data_train2` et les modalités associées.

```
# Fonction pour l'hyper-paramétrage de CURRENT
hyperparametrage_current<- function(data, list_modeles_current, list_seuil_current) {
  best_accuracy <- 0
  best_seuil_current <- 0
  best_num_model_current <- 1


  nb <- length(list_modeles_current) * length(list_seuil_current)
  i <- 0

  num_model_current <- 0
  for (step_model_current in list_modeles_current) {
    num_model_current <- num_model_current +1
    for (seuil_current in list_seuil_current) {
      i <- i+1
      if (i%%100 == 0) {
        print(paste0(i,"/",nb))
      }
      confusion_matrix <- table(Predicted = predict_current(data, step_model_current, seuil_current),
                                Actual = data$current01)
      accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
      if (accuracy > best_accuracy) {
        best_accuracy <- accuracy
        best_num_model_current <- num_model_current
        best_seuil_current <- seuil_current
      }
    }
  }
```

```
  return(c(best_accuracy, best_num_model_current, best_seuil_current))
}

liste_seuils <- seq(from = 0, to = 1, by = 0.025)
hyperparametrage_current(data_train2, modeles_current, liste_seuils)
```

Les meilleurs hyperparamètres sont donc 190 sondes avec un seuil de 0.575 qui donne une précision de 0.829.

On réitère le même processus avec le statut `never`.

Modèles sis avec les i meilleures sondes :

```
# NEVER
model_never_sis_i <- function(data_train, i, screening_func=msiscreening) {
  formula <- as.formula(paste0(c("never01 ~ 1", sis_probes[0:i]), collapse = "+"))
  m <- glm(formula, data_train, family = binomial(link = "logit"))
  return(m)
}

# On regarde le sur apprentissage sur les 50 sondes
iap1 = c()
iap2 = c()
for (i in 0:50) {
  m = model_never_sis_i(data_train1, i)
  pred_train1 = predict.glm(m, data_train1, type = "response")
  pred_train2 = predict.glm(m, data_train2, type = "response")
  iap1 = c(iap1, sum(ifelse(pred_train1 > 0.5, 1, 0) != data_train1$never01) / nrow(data_train1))
  iap2 = c(iap2, sum(ifelse(pred_train2 > 0.5, 1, 0) != data_train2$never01) / nrow(data_train2))
}
plot(iap2, col = 2, pch = 16, cex = 1, ylim = c(0.10, 0.40),
     main = "Incorrect Answers Proportion en fonction du nombre de sondes sur les deux jeux de données",
     xlab = "Nombre de sondes", ylab = "Incorrect Answers Proportion", cex.main = 0.8)
points(iap1, col = 4, pch = 16, cex = 1)
legend("bottomright", c("iap1", "iap2"), col = c(4, 2), pch = 16, cex = 0.8)
i <- 10
abline(v = i, col = 2)
```

Ici aussi, on voit qu'en prenant trop de sondes la proportion d'erreur augment sur les données de `data_train2`. À nouveau, nous allons utiliser la fonction `step` pour sélectionner les sondes.

Fonction `step` :

```
stepforward_never <- function(data_train, sis_probes, nb_sis_probes = 200, trace = 0, k = 2) {
  m_lo <- glm(never01 ~ 1, data = data_train[, c("never01", sis_probes[1:nb_sis_probes])])
  m_sup <- glm(never01 ~ ., data = data_train[, c("never01", sis_probes[1:nb_sis_probes])])
  m_fwd <- step(m_lo, method = "forward", scope = list(upper = m_sup, lower = m_lo), trace = trace, k = k)
  # print(m_fwd$call)
  return(m_fwd)
}
```

Exemple avec 90 sondes et un seuil à 0.55 :

```
# Exemple avec 90 sondes
step_model_never_90 <- stepforward_never(data_train1, sis_probes, nb_sis_probes = 90, trace = 0)

# On regarde les résultats sur data_train2
predicted_probs <- predict(step_model_never_90, newdata = data_train2, type = "response")
```

```r
predicted_classes <- ifelse(predicted_probs > 0.55, 1, 0)
confusion_matrix <- table(Predicted = predicted_classes, Actual = data_train2$never01)
print(confusion_matrix)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(paste("Précision:", accuracy))
```

En prenant 90 sondes, on obtient une précision nettement plus faible pour `never` que pour `current`. Essayons de nouveau de trouver de meilleurs hyperparamètres (seuil et nombre de sondes).

On commence par coder la fonction générale de prédiction qui prend en entrée les données dont il faut prédire la classe, le modèle à utiliser et le seuil.

```r
# Fonction générale
predict_never <- function(data,step_model, seuil){
  predicted_probs <- predict(step_model, newdata = data, type = "response")
  predicted_classes <- ifelse(predicted_probs > seuil, 1, 0)
  return(predicted_classes)
}
```

Modèles pour différents nombres de sondes, on va pendre 50 à 200 sondes avec un pas de 10.

```r
# On crée des modèles pour l'hyperparamétrage.
create_modeles_never <- function(from = 50, to = 200, by = 10) {
  res <- list()
  for (i in seq(from = from, to = to, by = by)) {
    res[[length(res) + 1]] <- stepforward_never(data_train1, sis_probes, nb_sis_probes = i, trace = 0)
  }
  return(res)
}
modeles_never <- create_modeles_never()
```

```r
# Fonction pour l'hyper-paramétrage de never
hyperparametrage_never <- function(data, list_modeles_never, list_seuil_never) {
  best_accuracy <- 0
  best_seuil_never <- 0
  best_num_model_never <- 1

  nb <- length(list_modeles_never) * length(list_seuil_never)

  num_model_never <- 0
  for (step_model_never in list_modeles_never) {
    num_model_never <- num_model_never +1
    for (seuil_never in list_seuil_never) {
      confusion_matrix <- table(Predicted = predict_never(data, step_model_never, seuil_never),
                                Actual = data$never01)
      accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
      if (accuracy > best_accuracy) {
        best_accuracy <- accuracy
        best_num_model_never <- num_model_never
        best_seuil_never <- seuil_never
      }
    }
  }
  return(c(best_accuracy, best_num_model_never, best_seuil_never))
}
```

```
liste_seuils <- seq(from = 0, to = 1, by = 0.025)
hyperparametrage_never(data_train2, modeles_never, liste_seuils)
```

Pour `never`, les meilleurs hyperparamètres sont donc 70 sondes avec un seuil de 0.425 qui donne une précision de 0.724 : c'est moins bon que `current`.

Enfin, on essaye de construire un modèle à partir du statut `former`.

On crée une fonction pour construire des modèles avec les i meilleures sondes.

```
# FORMER
model_former_sis_i <- function(data_train, i, screening_func = msiscreening) {
  formula <- as.formula(paste0(c("former01 ~ 1", sis_probes[0:i]), collapse = "+"))
  m <- glm(formula, data_train, family = binomial(link = "logit"))
  return(m)
}

# On regarde le sur apprentissage sur les 50 sondes
iap1 = c()
iap2 = c()
for (i in 0:50) {
  m <- model_former_sis_i(data_train1, i)
  pred_train1 <- predict.glm(m, data_train1, type = "response")
  pred_train2 <- predict.glm(m, data_train2, type = "response")
  iap1 = c(iap1, sum(ifelse(pred_train1 > 0.5, 1, 0) != data_train1$former01) / nrow(data_train1))
  iap2 = c(iap2, sum(ifelse(pred_train2 > 0.5, 1, 0) != data_train2$former01) / nrow(data_train2))
}
plot(iap1, col = 4, pch = 16, cex = 1, ylim = c(0.15, 0.5), cex.axis = 0.8,
     main = "Incorrect Answers Proportion en fonction du nombre de sondes sur les deux jeux de données",
     xlab = "Nombre de sondes", ylab = "Incorrect Answers Proportion", cex.main = 0.8)
points(iap2, col = 2, pch = 16, cex = 1)
legend("bottomright", c("iap1", "iap2"), col = c(4, 2), pch = 16, cex = 0.8)
i <- 9
abline(v=i, col=2)
```

A nouveau, on voit qu'en prenant un trop grand nombre de sondes la proportion d'erreur pour la proportion sur les données de `data_train2` augmente. Il y a sur-apprentissage.

A nouveau, on va utiliser la fonction `step` pour essayer de sélectionner les meilleures sondes et éviter de prendre des sondes fortement corrélées entre elles.

```
stepforward_former <- function(data_train, sis_probes, nb_sis_probes = 200, trace = 0, k = 2) {
  m_lo <- glm(former01 ~ 1, data = data_train[, c("former01", sis_probes[1:nb_sis_probes])])
  m_sup <- glm(former01 ~ ., data = data_train[, c("former01", sis_probes[1:nb_sis_probes])])
  m_fwd <- step(m_lo, method = "forward", scope = list(upper = m_sup, lower = m_lo), trace = trace, k = k)
  return(m_fwd)
}
```

On teste la précision sur le modèle `step` obtenu à partir des 90 meilleures sondes.

```
step_model_former_90 <- stepforward_former(data_train1, sis_probes, nb_sis_probes = 90, trace = 0)

# On regarde les résultats sur data_train2
predicted_probs <- predict(step_model_former_90, newdata = data_train2, type = "response")
predicted_classes <- ifelse(predicted_probs > 0.55, 1, 0)
confusion_matrix <- table(Predicted = predicted_classes, Actual = data_train2$former01)
print(confusion_matrix)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(paste("Précision :", accuracy))
```

On obtient une précision assez faible. Pour essayer de remédier à ça, on va chercher à trouver les meilleurs hyperparamètres (nombre de sondes et seuil).

On crée une fonction générale de prédiction qui prend en paramètres les données à prédire, le modèle à utiliser et le seuil de décision.

```
# Fonction générale
predict_former <- function(data,step_model, seuil) {
  predicted_probs <- predict(step_model, newdata = data, type = "response")
  predicted_classes <- ifelse(predicted_probs > seuil, 1, 0)
  return(predicted_classes)
}
```

On crée les modèles `step` obtenus à partir de différents nombres de sondes de 50 à 200 par pas de 10.

```
# On crée des modèles pour l'hyperparamétrage.
create_modeles_former <- function(from = 50, to = 200, by = 10) {
  res <- list()
  for (i in seq(from = from, to = to, by = by)) {
    res[[length(res) + 1]] <- stepforward_former(data_train1, sis_probes, nb_sis_probes = i, trace = 0)
  }
  return(res)
}
modeles_former <- create_modeles_former()
```

On crée une fonction pour automatiser l'hyperparamétrage, qui nous renvoie le meilleur nombre de sondes pour construire le modèle `step`, le seuil et la précision associée.

```
# Fonction pour l'hyperparamétrage de former
hyperparametrage_former<- function(data, list_modeles_former, list_seuil_former) {
  best_accuracy <- 0
  best_seuil_former <- 0
  best_num_model_former <- 1

  nb <- length(list_modeles_former) * length(list_seuil_former)

  num_model_former <- 0
  for (step_model_former in list_modeles_former) {
    num_model_former <- num_model_former +1
    for (seuil_former in list_seuil_former) {
      confusion_matrix <- table(Predicted = predict_former(data, step_model_former, seuil_former),
                                Actual = data$former01)
      accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
      if (accuracy > best_accuracy) {
        best_accuracy <- accuracy
        best_num_model_former <- num_model_former
        best_seuil_former <- seuil_former
      }
    }
  }
  return(c(best_accuracy, best_num_model_former, best_seuil_former))
}

liste_seuils <- seq(from = 0, to = 1, by = 0.025)
hyperparametrage_former(data_train2, modeles_former, liste_seuils)
```

Pour former, les meilleurs hyperparamètres sont donc 140 sondes avec un seuil de 0.575 qui donne une précision de 0.714 : c'est moins bon que `current` et `never`.

Finalement, c'est le modèle basé sur `current` qui permet d'obtenir la meilleure précision sur le jeu de données `data_train2`. On va le choisir comme premier modèle pour notre arbre de décision. Nous allons maintenant construire deux autres modèles pour `former` et `never` mais cette fois, nous les entraîneront seulement sur les données des individus n'étant pas labellisés comme `current` (puisqu'ils sont censés avoir été labellisés `current` par le premier modèle de l'arbre).

2.1.3 Deuxième modèle

On commence tout d'abord par enlever de `data_train1` et `data_train2` les individus labellisés comme `current`.

```
data_train1_notcurrent <- data_train1[data_train1$current01! = 1,]
data_train2_notcurrent <- data_train2[data_train2$current01! = 1,]
```

```
dim(data_train1)
dim(data_train1_notcurrent)
```

Pour `data_train1`, on passe de 316 individus à 217.

```
dim(data_train2)
dim(data_train2_notcurrent)
```

Pour `data_train2`, on passe de 105 individus à 69.

Commençons par le modèle `never`. Le processus est le même que précédemment : on a une fonction pour crée les modèles basés sur les i meilleures sondes, on a une fonction `step` qui permet de trouver le meilleur modèle (AIC) pour les i meilleurs sondes, on effectué un hyperparamétrage pour trouver le meilleur nombre de sondes et le meilleur seuil pour améliorer la précision mais cette fois en testant sur `data_train2_notcurrent` et en s'entraînant sur `data_train1_notcurrent`.

```
# NOT CURRENT - NEVER
model_never_sis_i <- function(data_train, i, screening_func=msiscreening) {
  formula <- as.formula(paste0(c("never01 ~ 1", sis_probes[0:i]), collapse = "+"))
  m <- glm(formula, data_train, family = binomial(link = "logit"))
  return(m)
}

# On effectue un step
stepforward_never <- function(data_train, sis_probes, nb_sis_probes = 200, trace = 0, k = 2) {
  m_lo <- glm(never01 ~ 1, data = data_train[, c("never01", sis_probes[1:nb_sis_probes])])
  m_sup <- glm(never01 ~ ., data = data_train[, c("never01", sis_probes[1:nb_sis_probes])])
  m_fwd <- step(m_lo, method = "forward", scope = list(upper = m_sup, lower = m_lo), trace = trace, k = k)
  # print(m_fwd$call)
  return(m_fwd)
}

# Fonction générale
predict_notcurrent_never <- function(data,step_model, seuil) {
  predicted_probs <- predict(step_model, newdata = data, type = "response")
  predicted_classes <- ifelse(predicted_probs > seuil, 1, 0)
  return(predicted_classes)
}

# On crée des modèles pour l'hyperparamétrage.
create_modeles_notcurrent_never <- function(from = 50, to = 200, by = 10) {
  res <- list()
  for (i in seq(from = from, to = to, by = by)) {
```

```
    res[[length(res) + 1]] <- stepforward_never(data_train1_notcurrent, sis_probes, nb_sis_probes = i, trac
  }
  return(res)
}
modeles_notcurrent_never <- create_modeles_notcurrent_never()

# Fonction pour l'hyperparamétrage de never
hyperparametrage_notcurrent_never<- function(data, list_modeles_never, list_seuil_never) {
  best_accuracy <- 0
  best_seuil_never <- 0
  best_num_model_never <- 1

  nb <- length(list_modeles_never) * length(list_seuil_never)

  num_model_never <- 0
  for (step_model_never in list_modeles_never) {
    num_model_never <- num_model_never +1
    for (seuil_never in list_seuil_never) {
      confusion_matrix <- table(Predicted = predict_never(data, step_model_never, seuil_never),
                                Actual = data$never01)
      accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
      if (accuracy > best_accuracy) {
        best_accuracy <- accuracy
        best_num_model_never <- num_model_never
        best_seuil_never <- seuil_never
      }
    }
  }
  return(c(best_accuracy, best_num_model_never, best_seuil_never))
}

liste_seuils <- seq(from = 0, to = 1, by = 0.025)
hyperparametrage_notcurrent_never(data_train2_notcurrent, modeles_notcurrent_never, liste_seuils)
```

Pour `never`, les meilleurs hyperparamètres sont donc 100 sondes avec un seuil de 0.725 qui donne une précision de 0.5797.

On fait la même chose pour `former`.

```
# NOT CURRENT - former
# former
model_former_sis_i <- function(data_train, i, screening_func = msiscreening) {
  formula <- as.formula(paste0(c("former01 ~ 1", sis_probes[0:i]), collapse = "+"))
  m <- glm(formula, data_train, family = binomial(link = "logit"))
  return(m)
}

# On effectue un step
stepforward_former <- function(data_train, sis_probes, nb_sis_probes = 200, trace = 0, k = 2) {
  m_lo <- glm(former01 ~ 1, data = data_train[, c("former01", sis_probes[1:nb_sis_probes])])
  m_sup <- glm(former01 ~ ., data = data_train[, c("former01", sis_probes[1:nb_sis_probes])])
  m_fwd <- step(m_lo, method = "forward", scope = list(upper = m_sup, lower = m_lo), trace = trace, k = k)
  # print(m_fwd$call)
  return(m_fwd)
}

# Fonction générale
```

```r
predict_notcurrent_former <- function(data,step_model, seuil) {
  predicted_probs <- predict(step_model, newdata = data, type = "response")
  predicted_classes <- ifelse(predicted_probs > seuil, 1, 0)
  return(predicted_classes)
}

# On crée des modèles pour l'hyperparamétrage
create_modeles_notcurrent_former <- function(from = 50, to = 200, by = 10) {
  res <- list()
  for (i in seq(from = from, to = to, by = by)) {
    res[[length(res) + 1]] <- stepforward_former(data_train1_notcurrent, sis_probes, nb_sis_probes = i, tra
  }
  return(res)
}
modeles_notcurrent_former <- create_modeles_notcurrent_former()

# Fonction pour l'hyperparamétrage de former
hyperparametrage_notcurrent_former<- function(data, list_modeles_former, list_seuil_former) {
  best_accuracy <- 0
  best_seuil_former <- 0
  best_num_model_former <- 1


  nb <- length(list_modeles_former) * length(list_seuil_former)

  num_model_former <- 0
  for (step_model_former in list_modeles_former) {
    num_model_former <- num_model_former +1
    for (seuil_former in list_seuil_former) {
      confusion_matrix <- table(Predicted = predict_former(data, step_model_former, seuil_former),
                                Actual = data$former01)
      accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
      if (accuracy > best_accuracy) {
        best_accuracy <- accuracy
        best_num_model_former <- num_model_former
        best_seuil_former <- seuil_former
      }
    }
  }
  return(c(best_accuracy, best_num_model_former, best_seuil_former))
}

liste_seuils <- seq(from = 0, to = 1, by = 0.025)
hyperparametrage_notcurrent_former(data_train2_notcurrent, modeles_notcurrent_former, liste_seuils)
```

Pour `former`, les meilleurs hyperparamètres sont donc 100 sondes avec un seuil de 0.275 qui donne une précision de 0.5797.

La précision est la même pour les deux, nous sélectionnons arbitrairement `never` comme second modèle de notre arbre de décision.

2.1.4 Combinaison des deux modèles

Enfin, on combine les deux modèles trouvés précédemment pour construire notre arbre de décision.

```r
predict_current_never<- function(data, step_model_current, seuil_current, step_model_never, seuil_never) {

  # Initialisation du vecteur résultat
  n <- nrow(data)
  predictions <- rep(NA, times = n)

  # Prédictions pour "current"
  is_current <- predict_current(data, step_model_current, seuil_current) == 1
  predictions[is_current] <- "current"

  # Prédictions pour "never" parmi les non-classés
  remaining <- is.na(predictions)
  is_never <- predict_notcurrent_never(data[remaining, ], step_model_never, seuil_never) == 1
  predictions[which(remaining)[is_never]] <- "never"

  # Tout le reste est "former"
  predictions[is.na(predictions)] <- "former"

  return(predictions)
}

model_current_final <- stepforward_current(data_train1, sis_probes, nb_sis_probes = 190,trace = 0)

model_never_final <- stepforward_never(data_train1_notcurrent, sis_probes, nb_sis_probes = 100, trace =0)
```

```r
# Essai sur data_train2
predicted <- predict_current_never(data_train2, model_current_final, 0.575, model_never_final, 0.725)
head(predicted, 10)
```

2.2 *Elastic Net*

Nous voulions initialement faire un algorithme de Random Forest en deuxième modèle mais, après quelques tests effectués, nous nous sommes rendu compte qu'il n'était pas performant. Nous avons donc décidé d'explorer le modèle Elastic Net.

2.2.1 Idée du modèle

Elastic Net est un modèle réalisant un ajustement entre deux pénalités tirés d'autres modèles :

- la pénalité L1 du modèle LASSO. Ce modèle vise à minimiser la somme de l'erreur quadratique moyenne et d'une pénalité sur la valeur absolue des coefficients β du modèle. Cette pénalité force certains coefficients à 0, faisant donc une sélection dans nos variables explicatives.
- la pénalité L2 du modèle Ridge. Ce modèle vise à minimiser la somme de l'erreur quadratique moyenne et d'une pénalité sur la somme des carrés des coefficients β du modèle. Ceci permet de réduire les coefficients sans pour autant les annuler, et donc de mieux gérer des variables fortement corrélées.

Elastic Net réalise un ajustement de ces pénalités via un hyperparamètre α : $P_{EN} = \alpha L1 + \frac{1-\alpha}{2} L2$.

Ainsi, lorsque $\alpha = 0$, nous avons simplement un modèle Ridge, et lorsque $\alpha = 1$, nous avons un modèle LASSO. Comme pour les deux autres modèles, Elastic Net possède aussi un hyperparamètre de régularisation λ qui contrôle l'intensité de la pénalité : plus il est grand, plus la pénalité est forte, et donc plus les coefficients sont réduis. Finalement, ce modèle vise à minimiser la somme de l'erreur quadratique moyenne (MSE) et de la pénalité ajustée et régularisée, c'est-à-dire réduire $MSE + \lambda \cdot P_{EN}$.

Elastic Net semble donc assez bien adapté aux données du data challenge puisque nous voulons à la fois réduire le nombre de sondes *et* gérer la corrélation de celles sélectionnées. Ce modèle est aussi bien construit afin d'éviter le risque sur-apprentissage, risque assez élevé avec des jeu de données comme ici où le nombres d'individus est très inférieur au nombre de variables.

2.2.2 Premier modèle

Avant de partir dans de l'hyperparamétrage, commençons d'abord par nous familiariser avec le modèle sur le jeu de données d'entraînement.

Il faut d'abord séparer nos variables prédictives de notre variable à expliquée. De plus, Elastic Net ne fonctionne qu'avec des matrices, donc il faut au passage convertir les variables explicatives en matrices :

```
    # Matrices des variables prédictives
X_train <- as.matrix(data_train1[, 5:ncol(data_train1)])
X_test <- as.matrix(data_train2[, 5:ncol(data_train2)])

    # Variable cible
Y <- data_train1$smoking_status
```

Notre variable cible possède plusieurs labels ; il faudra le spécifier dans les arguments du modèle : `family = "multinomial"`.

Afin d'obtenir automatiquement le meilleur λ pour nos données, nous utilisons la fonction `cv.glmnet` du package `glmnet`. Celle-ci permet de faire une cross-validation de modèles pour différentes valeurs de λ et possède dans ses arguments de sorties deux valeurs intéressantes pour cet hyperparamètre : `lambda.min` et `lambda.1se`. La première indique la valeur de λ pour laquelle l'erreur commise est la plus faible et la deuxième correspond au λ du modèle commettant la plus grande erreur "acceptable" (dans une plage de 1 écart-type autour de l'erreur minimale). Le premier modèle permet donc de mieux réduire l'erreur commise mais le deuxième est plus robuste lors de généralisation.

L'argument `nfolds` permet de renseigner le nombre de sous-échantillons créé pour effectuer la cross-validation.

Fixons enfin $\alpha = 0,5$ pour avoir le modèle médian entre Ridge et LASSO. Ainsi, notre modèle s'écrit comme suit :

```
elast_model <- cv.glmnet(x = X_train, y = Y, family = "multinomial",
                         alpha = 0.5, nfolds = 20)
```

Nous pouvons maintenant tenter de prédire les valeur de `smoking_status` dans `data_train2` :

```
predictions_elast <- predict(elast_model, newx = X_test,
                             s = elast_model$lambda.min, type = "class")
accuracy_elast <- sum(predictions_elast == data_train2$smoking_status) / nrow(data_train2)
```

Le but étant de minimiser l'erreur commise, nous utiliserons ici la valeur de `lambda.min` pour l'hyperparamètre λ. L'argument `type = "class"` dans la fonction `predict()` permet d'obtenir directement les classes prédites :

```
kable(head(predictions_elast), align = "r")
```

Ainsi, ce premier modèle obtient un taux d'erreur de `round(1 - accuracy_elast, 3)`, ce qui est déjà plutôt bon.

2.2.3 Paramétrage du modèle

Maintenant que nous connaissons un peu mieux le modèle, nous pouvons essayer de le paramétrer afin de réduire son taux d'erreur. Étant donné que la sélection du λ est déjà réalisé automatiquement, nous pouvons nous concentrer sur d'autre paramètres :

- la fonction `cv.glmnet()` possède un argument `weights` permettant d'affecter un poids aux classes à prédire afin d'éviter qu'il y en ait une qui tire les autres si les données sont déséquilibrées.
- le nombre `nfolds` peut aussi être intéressant à calibrer.
- la valeur d'ajustement α.
- le nombre de sondes à utiliser dans la matrice de prédiction.

Commençons donc d'abord par trier les sondes. La méthode de tri utiliser dans le modèles SIS ne tenant pas compte de la corrélation entre les sondes, nous allons ici les classer différemment grâce à une Random Forest. En effet, ce modèle possède un argument `importance` permettant de trier les variables prédictives selon deux valeurs : `MeanDecreaseAccuracy` ou `MeanDecreaseGini`. Ici, nous utiliserons le premier pour trier les sondes.

Construisons donc un modèle de Random Forest basique à 100 arbres et avec toutes les sondes en variables explicatives :

```
rf_model <- randomForest(smoking_status ~ .,
                         data = data_train1[, c("smoking_status", probes)],
                         ntree = 100, importance = TRUE)
```

Nous récupérons ensuite les sondes trier selon leur `MeanDecreaseAccuracy` afin d'avoir les plus importantes en premières :

```
    # Importance des variables
imp_sondes <- importance(rf_model)
sondes_tri <- names(sort(imp_sondes[, "MeanDecreaseAccuracy"], decreasing = TRUE))
```

Ainsi, le vecteur `sondes_tri` possède le noms des sondes trier par leur importance de prédiction dans un modèle de Random Forest. Les premières sondes sont donc les plus utiles à la prédiction.

L'avantage de cette méthode sur celle du modèle SIS est la manière dont elle traite la corrélation des sondes. Ici, la corrélation entre en jeu lors du calcul de l'importance de la sonde, permettant aux premières sondes d'être le moins corrélées possible, ce qui n'est pas le cas avec l'autre méthode.

Reste maintenant à savoir quel nombre de sondes sera optimale pour le modèle, mais nous y reviendront un peu plus tard. Avant cela, intéressons nous au poids des classes de `smoking_status`. Bien que le jeu de données présente ici des classes assez équilibrées, il vaut mieux les pondérées au cas où, surtout que le coût computationnel de cette opération n'est pas très important :

```
    # Pondération des données (pour éviter qu'une classe ne tire les autres)
poids <- table(Y)
poids <- 1 / poids[as.character(Y)]
# poids inverse à la fréquence pour l'équilibrage

poids <- poids / sum(poids)
# pour avoir une somme égale à 1
```

Comme énoncé plus haut, nous pourrions aussi chercher à calibrer l'argument `nfolds`, mais nos données font qu'il n'est pas spécialement utile de le faire. Nous le fixons donc à une valeur arbitraire de 10 (valeur par défaut de la fonction) ; il n'est ni trop petit, pour ne pas perdre de fiabilité sur l'erreur, ni trop grand, pour éviter le sur-apprentissage et l'augmentation du temps de calcul.

Nous pouvons donc maintenant calibrer nos deux paramètres restants : α et le nombre de sondes. Pour cela, nous construisons deux vecteurs, un pour les valeurs de α à tester et un pour les valeurs du nombre de sondes, et nous construisons deux boucles imbriquées qui vont parcourir ces vecteurs et retourner le taux de réussite de chaque modèle :

```
    # Vecteurs de valeurs pour alpha et le nombre de sondes
alpha <- seq(from = 0, to = 1, by = 0.1)
# alpha est compris entre 0 et 1 par définition

nb_sondes <- seq(from = 500, to = 1000, by = 100)
# Première recherches avec une réduction importante de sondes

    # Vecteurs vide pour venir y stocker nos résultats
alpha_top <- c()
nb_sondes_opt <- c()
accuracy_int <- c()
accuracy_ext <- c()


for (i in 1:length(nb_sondes)) {
  X_train_n <- as.matrix(data_train1[, sondes_tri[1:nb_sondes[i]]])
  X_test_n <- as.matrix(data_train2[, sondes_tri[1:nb_sondes[i]]])

  for (j in 1:length(alpha)) {
    elast_model <- cv.glmnet(x = X_train_n, y = Y, family = "multinomial",
                             weights = poids, alpha = alpha[j], nfolds = 10)
    predictions_elast <- predict(elast_model, newx = X_test_n,
                                 s = elast_model$lambda.min, type = "class")
    accuracy_int[j] <-
      sum(predictions_elast == data_train2$smoking_status) / nrow(data_train2)
  }

  rapport_alpha <- data.frame(Alpha = alpha, Precision = accuracy_int)
  alpha_top[i] <- rapport_alpha[which.max(rapport_alpha$Precision), "Alpha"]
  accuracy_ext[i] <- rapport_alpha[which.max(rapport_alpha$Precision), "Precision"]
  # print(paste(i, "croisements trouvés sur", length(nb_sondes)))
}
```

La boucle agit ainsi :

- On sélectionne un nombre de sondes et on construit nos matrices de prédictions avec.
- On parcours les valeurs possible de α, on construit le modèle Elastic Net associé à chacune et on stock la précision des `length(alpha)` modèles dans `accuracy_int`.
- On stock ensuite dans `alpha_top` la valeur de α pour laquelle la précision est la meilleure. On stock aussi cette dite prédiction dans `accuracy_ext`.
- On recommence jusqu'à avoir parcouru toutes les valeurs voulues pour le nombre de sondes. À chaque fois que la valeur `nb_sondes` change, le vecteur `accuracy_int` est entièrement modifié. Au contraire, les vecteurs `alpha_top` et `accuracy_ext` prennent juste une valeur supplémentaire à chaque itération.

On peut ensuite construire un tableau avec nos résultats :

```
rapport <- data.frame(nb_sondes, alpha_top, round(accuracy_ext, 3))
colnames(rapport) <- c("Nombre de sondes", "Meilleur alpha", "Précision")
kable(rapport, align = "r")
```

Ses résultats ne sont pas très convaincant, surtout si nous les comparons au résultat obtenu sur le premier modèle. Nous pouvons tout de même noté une tendance dans le tableau : plus le nombres de sondes augmente, plus la précision est bonne. Essayons donc différents modèles jusqu'au nombre maximal de sondes (10 000) :

```
    # Vecteurs de valeurs pour alpha et le nombre de sondes
alpha <- seq(from = 0, to = 1, by = 0.1)
nb_sondes <- seq(from = 500, to = 10000, by = 500)

    # Vecteurs vide pour venir y stocker nos résultats
alpha_top <- c()
nb_sondes_opt <- c()
accuracy_int <- c()
accuracy_ext <- c()


for (i in 1:length(nb_sondes)) {
  X_train_n <- as.matrix(data_train1[, sondes_tri[1:nb_sondes[i]]])
  X_test_n <- as.matrix(data_train2[, sondes_tri[1:nb_sondes[i]]])

  for (j in 1:length(alpha)) {
    elast_model <- cv.glmnet(x = X_train_n, y = Y, family = "multinomial",
                             weights = poids, alpha = alpha[j], nfolds = 10)
    predictions_elast <- predict(elast_model, newx = X_test_n,
                                 s = elast_model$lambda.min, type = "class")
    accuracy_int[j] <-
      sum(predictions_elast == data_train2$smoking_status) / nrow(data_train2)
  }

  rapport_alpha <- data.frame(Alpha = alpha, Precision = accuracy_int)
  alpha_top[i] <- rapport_alpha[which.max(rapport_alpha$Precision), "Alpha"]
  accuracy_ext[i] <- rapport_alpha[which.max(rapport_alpha$Precision), "Precision"]
  # print(paste(i, "croisements trouvés sur", length(nb_sondes)))
}

rapport <- data.frame(nb_sondes, alpha_top, round(accuracy_ext, 3))
colnames(rapport) <- c("Nombre de sondes", "Meilleur alpha", "Précision")
kable(rapport, align = "r")
```

Au vu de ces résultats, il semble que le nombre optimal de sondes soit autour des 4 000. En effet, la précision y est aussi bonne, voir meilleure, qu'avec toutes les sondes. Il serait donc judicieux de choisir 4 000 pour une précision optimal avec un temps de calcul réduit et un sur-apprentissage limité au maximum.

Le choix de *alpha* semble quand à lui évident, une valeur de 0,1 semble la meilleure.

Lançons donc une dernière fois un calibrage afin d'obtenir un peu plus de précision autour de ces valeurs clés :

```
    # Vecteurs de valeurs pour alpha et le nombre de sondes
alpha <- seq(from = 0, to = 0.3, by = 0.02)
nb_sondes <- seq(from = 3500, to = 4500, by = 100)

    # Vecteurs vide pour venir y stocker nos résultats
alpha_top <- c()
nb_sondes_opt <- c()
accuracy_int <- c()
accuracy_ext <- c()


for (i in 1:length(nb_sondes)) {
  X_train_n <- as.matrix(data_train1[, sondes_tri[1:nb_sondes[i]]])
  X_test_n <- as.matrix(data_train2[, sondes_tri[1:nb_sondes[i]]])

  for (j in 1:length(alpha)) {
```

```
    elast_model <- cv.glmnet(x = X_train_n, y = Y, family = "multinomial",
                             weights = poids, alpha = alpha[j], nfolds = 10)
    predictions_elast <- predict(elast_model, newx = X_test_n,
                                 s = elast_model$lambda.min, type = "class")
    accuracy_int[j] <-
      sum(predictions_elast == data_train2$smoking_status) / nrow(data_train2)
  }

  rapport_alpha <- data.frame(Alpha = alpha, Precision = accuracy_int)
  alpha_top[i] <- rapport_alpha[which.max(rapport_alpha$Precision), "Alpha"]
  accuracy_ext[i] <- rapport_alpha[which.max(rapport_alpha$Precision), "Precision"]
  # print(paste(i, "croisements trouvés sur", length(nb_sondes)))
}

rapport <- data.frame(nb_sondes, alpha_top, round(accuracy_ext, 3))
colnames(rapport) <- c("Nombre de sondes", "Meilleur alpha", "Précision")
kable(rapport, align = "r")
```

Ces derniers résultats permettent de conclure que les meilleures valeurs pour le nombre de sondes et α sont :
`rapport[which.max(rapport$Précision), c("Sondes", "Alpha")]`

2.2.4 Modèle final

Finalement, le meilleur modèle adapté aux données de ce data challenge avec Elastic Net est donc le suivant :

```
# Sondes (variables explicatives)
sondes <- colnames(data_train)[5:length(data_train)]

# Modèle de Random Forest pour estimer l'importance des sondes dans la prédiction
rf_model <- randomForest(smoking_status ~ .,
                         data = data_train[, c("smoking_status", sondes)],
                         ntree = 100, importance = TRUE)

# Importance des sondes
imp_sondes <- importance(rf_model)
sondes_4300 <- names(sort(imp_sondes[, "MeanDecreaseAccuracy"],
                          decreasing = TRUE)[1:4300]) # 4300 meilleures sondes

# Matrice de prédiction
X_train_rf <- as.matrix(data_train[, sondes_4300])

# Variable cible
Y <- data_train$smoking_status

# Pondération des données (pour éviter qu'une classe ne tire les autres)
poids <- table(Y)
poids <- 1 / poids[as.character(Y)]      # poids inverse à la fréquence
poids <- poids / sum(poids)              # pour avoir une somme égale à 1

# Modèle Elastic Net
elast_model <- cv.glmnet(x = X_train_rf, y = Y, family = "multinomial",
                         weights = poids, alpha = 0.02, nfolds = 10)
```

2.3 *XGBoost*

3 *Résultats*

3.1 *Arbre de décision à partir de modèles linéaires*

3.1.1 Résultats sur data_train2

```
predicted <- predict_current_never(data_train2, model_current_final, 0.575, model_never_final, 0.725)
confusion_matrix <- table(Predicted = predicted, Actual = data_train2$smoking_status)
print(confusion_matrix)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(paste("Précision :", accuracy))
```

3.1.2 Résultats sur data_test

La soumission correspondante sur Codabench est la numéro 493 du 16 décembre 2024 publiée par l'utilisateur malnoe. Le résultat obtenu est de 0.475.

3.2 *Elastic Net*

3.2.1 Résultats sur data_train2

```
    # Modèle de Random Forest pour estimer l'importance des sondes
rf_model <- randomForest(smoking_status ~ .,
                         data = data_train1[, c("smoking_status", probes)],
                         ntree = 100, importance = TRUE)

    # Importance des sondes
imp_sondes <- importance(rf_model)
sondes_4300 <- names(sort(imp_sondes[, "MeanDecreaseAccuracy"],
                          decreasing = TRUE)[1:4300])
# 4300 meilleures sondes

    # Variables prédictives
X_train_rf <- as.matrix(data_train1[, sondes_4300])
X_test_rf <- as.matrix(data_train2[, sondes_4300])

    # Variable cible
Y <- data_train1$smoking_status

    # Pondération des données (pour éviter qu'une classe ne tire les autres)
poids <- table(Y)
poids <- 1 / poids[as.character(Y)]
poids <- poids / sum(poids)

    # Modèle Elastic Net
elast_model <- cv.glmnet(x = X_train_rf, y = Y, family = "multinomial",
                         weights = poids, alpha = 0.02, nfolds = 10)

    # Prédiction
data_pred <- predict(elast_model, newx = X_test_rf,
                     s = elast_model$lambda.min, type = "class")

    # Taux de précision
accuracy <- sum(data_pred == data_train2$smoking_status) / nrow(data_train2)
```

Le taux d'erreur avec le modèle final d'Elastic Net est `1 - accuracy`.

3.2.2 Résultats sur data_test

La soumission correspondante sur Codabench est la numéro 553 du 17 décembre 2024 publiée par l'utilisateur mazetma. Le résultat obtenu est de 0,295. Il a été obtenu avec le programme suivant :

```r
program <- function(data_train, data_test) {
  library(glmnet)
  library(randomForest)

  # Sondes (variables explicatives)
  sondes <- colnames(data_train)[5:length(data_train)]

  # Modèle de Random Forest pour estimer l'importance des sondes dans la prédiction
  rf_model <- randomForest(smoking_status ~ .,
                           data = data_train[, c("smoking_status", sondes)],
                           ntree = 100, importance = TRUE)

  # Importance des sondes
  imp_sondes <- importance(rf_model)
  sondes_4300 <- names(sort(imp_sondes[, "MeanDecreaseAccuracy"],
                            decreasing = TRUE)[1:4300]) # 4300 meilleures sondes

  # Variables prédictives
  X_train_rf <- as.matrix(data_train[, sondes_4300])
  X_test_rf <- as.matrix(data_test[, sondes_4300])

  # Variable cible
  Y <- data_train$smoking_status

  # Pondération des données (pour éviter qu'une classe ne tire les autres)
  poids <- table(Y)
  poids <- 1 / poids[as.character(Y)]      # poids inverse à la fréquence
  poids <- poids / sum(poids)              # pour avoir une somme égale à 1

  # Modèle Elastic Net
  elast_model <- cv.glmnet(x = X_train_rf, y = Y, family = "multinomial",
                           weights = poids, alpha = 0.02, nfolds = 10)
  # Valeur de alpha déterminée par essais sur data_train.

  # Prédiction
  data_pred <- predict(elast_model, newx = X_test_rf,
                       s = elast_model$lambda.min, type = "class")

  return(data_pred)
}
```

3.3 *XGBoost*

3.3.1 Résultats sur data_train2

3.3.2 Résultats sur data_test

Mettre numéro de la soumission sur Codabench.

4 *Discussion*

4.1 *Commentaires des résultats*

4.2 *Proposer des faits pour détailler les résultats (cohérence ou différences)*