GLM - Data challenge Epismoke 2.1

Garance Malnoë, Matthias Mazet, Léonie Breuzat

2024-12-16

Contents

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.

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)[5:10004]
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
```

```r
  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.
```

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.

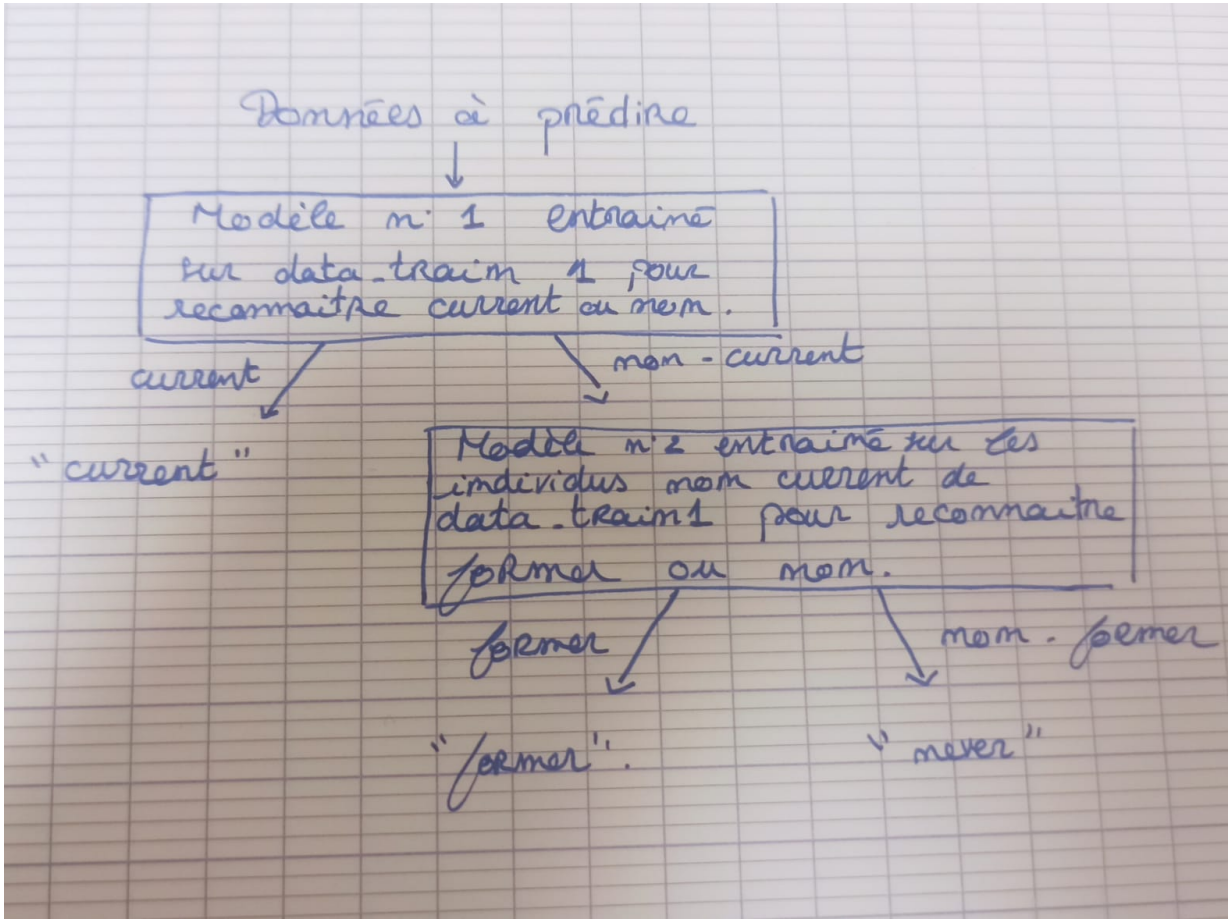
Arbre de décision à partir de modèles linéaires

Idée

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 :



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.

```r
# 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).

```r
# 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 =
  return(m_fwd)
}
```

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

```r
# 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)
```

```r
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.

```r
# 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.

```r
# 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.

```r
# 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 =
  # 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")
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.

```
# 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.

```
# 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()
```

```
# 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.

```r
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 =
  return(m_fwd)
}
```

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

```r
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.

```r
# 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.

```r
# 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.

```r
# 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).

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 =
  # 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,
  }
  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 =
  # print(m_fwd$call)
  return(m_fwd)
}

# Fonction générale
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,
  }
  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.

Combinaison des deux modèles

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

```
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
```

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

Elastic Net

XGBoost

Résultats

Arbre de décision à partir de modèles linéaires

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))
```

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.

Random forest

Résultats sur data_train2

Résultats sur data_test

Mettre numéro de la soumission sur Codabench.

XGBoost

Résultats sur data_train2

Résultats sur data_test

Mettre numéro de la soumission sur Codabench.

Discussion

Commentaires des résultats

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