

# Stability selection pour la prédiction de la résistance d’une souche bactérienne à un antibiotique

M1 parcours SSD – UE Apprentissage Statistique I

Vadim BERTRAND   Lola COTTIN   Marie GAFFET

29/04/2022

## 1 Introduction

Certaines souches bactériennes présentent une résistance particulière aux antibiotiques développés pour lutter contre elles.

Cette résistance peut être modélisée par une régression logistique ayant pour variables explicatives l’activation ou non des gènes du génome des souches. Seulement, le grand nombre de gènes (plusieurs dizaines de milliers) impose de sélectionner les gènes explicatifs avant d’entraîner le modèle de régression logistique. Pour cela des méthodes de régression pénalisant le nombre de variables sélectionnées existent. Lasso notamment peut être utilisé, mais quand les variables mises en jeu sont corrélées il présente l’inconvénient d’être instable et de sélectionner plus de variables que nécessaire.

Pour palier ce problème, la technique de stability selection [1] peut être intéressante. Il s’agit de répéter plusieurs fois l’ajustement d’une régression logistique Lasso basée sur un sous-échantillonnage du jeu de données et d’en déduire la fréquence de sélection des variables par l’ensemble des régressions. Les variables dont la fréquence de sélection dépasse un seuil donné sont enfin utilisées pour l’ajustement d’une régression logistique non pénalisée.

Nous verrons ici comment implémenter la procédure de stability selection sur R, puis nous commenterons les résultats obtenus selon le seuil de sélection en les comparant à ceux d’une régression logistique Lasso “classique”.

## 2 Implémentation de la stability selection

L’ensemble du code mise en oeuvre pour l’implémentation et l’application de la stability selection étant présent en annexe, nous ne reviendrons donc que brièvement sur les éléments clés de celles-ci.

### 3 Application à la prédiction de la résistance d'une souche bactérienne à un antibiotique

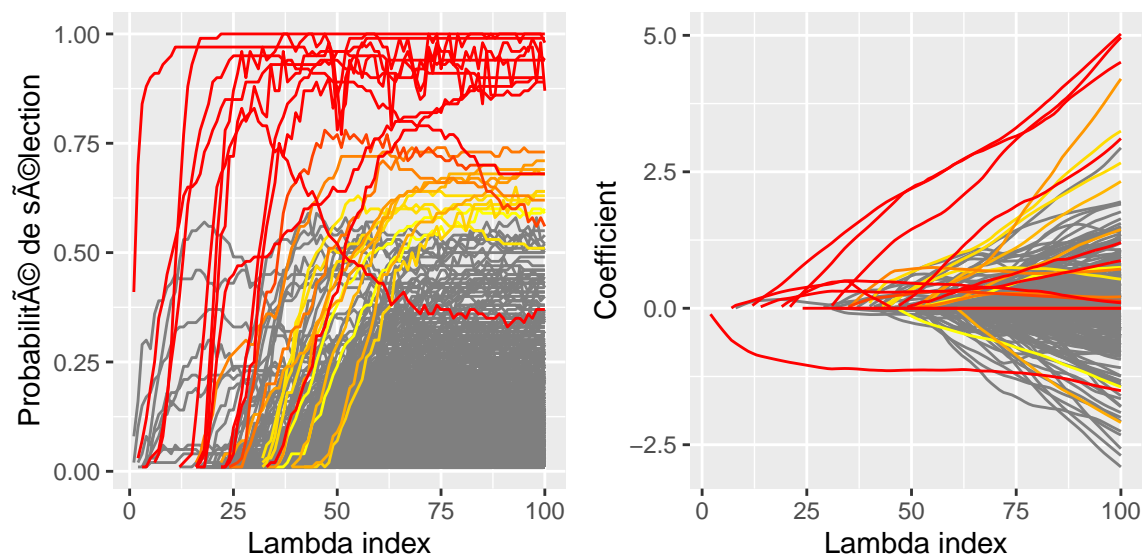


Figure 1: Chemins de stabilité (à gauche) et de régularisation (à droite)

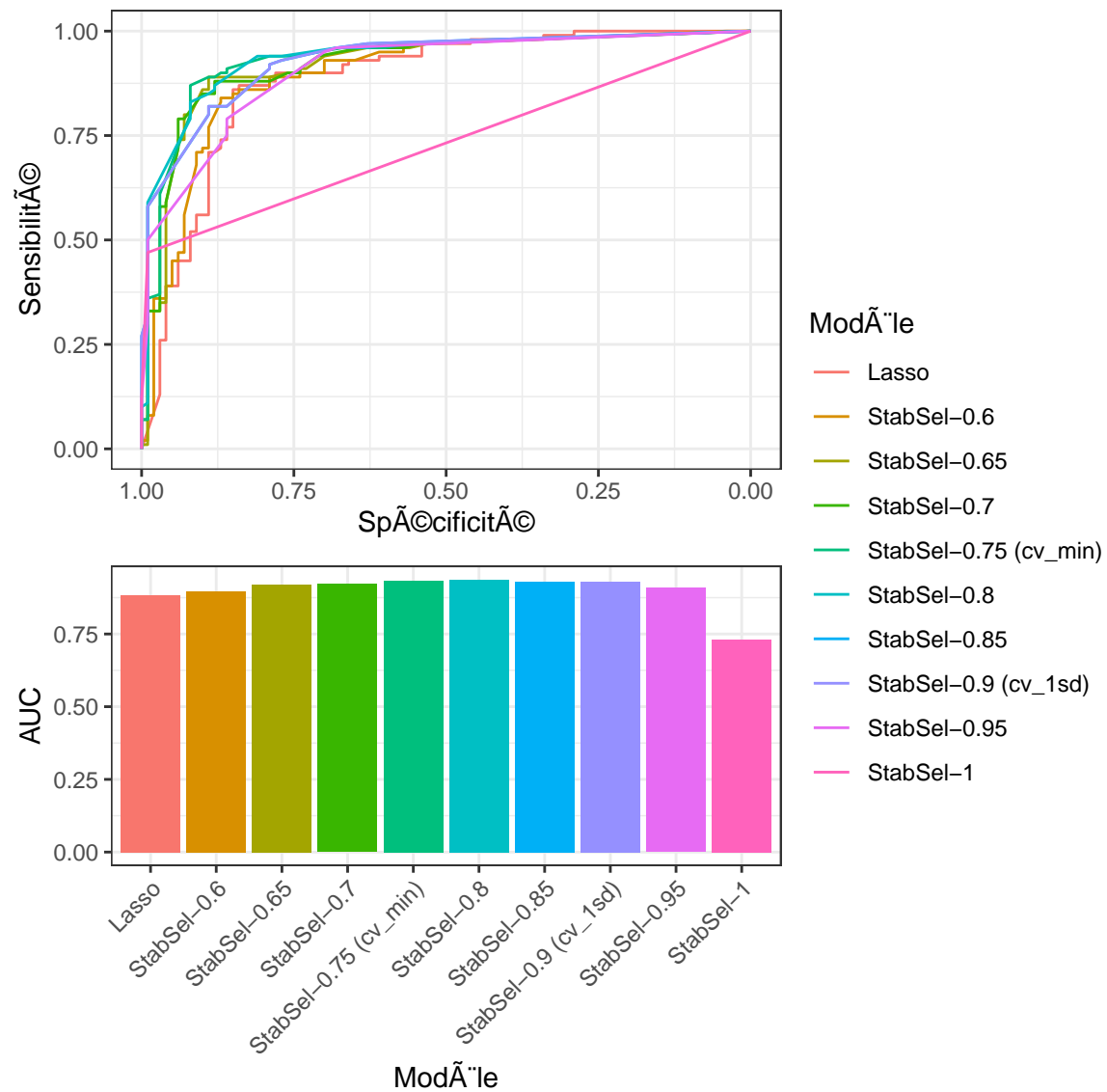


Figure 2: Courbes ROC (en haut) et diagramme à bâtons représentant les AUC (en bas) des différents modèles

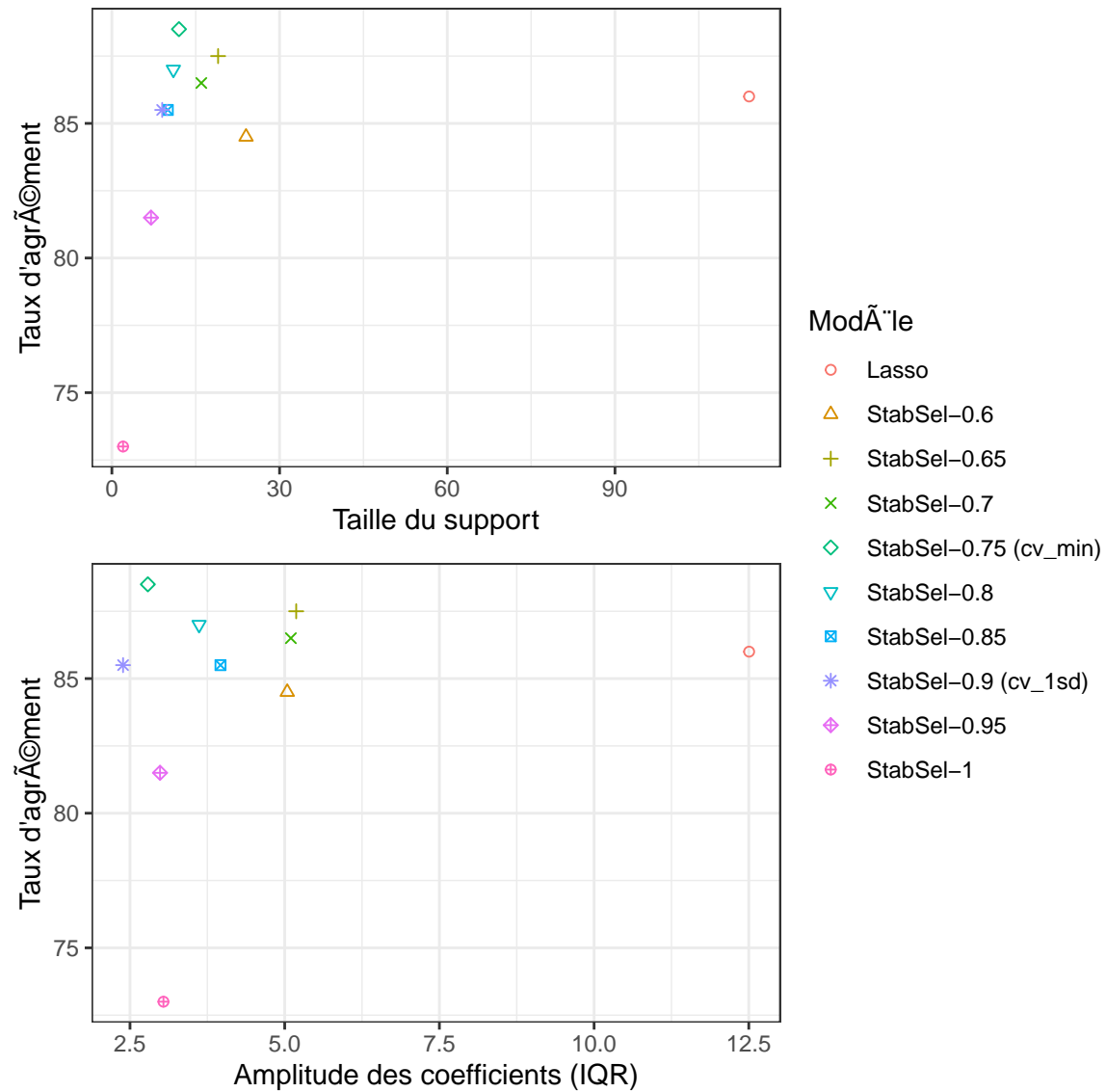


Figure 3: Taux d'agrément en fonction de la taille du support (en haut) et taux d'agrément en fonction de l'amplitude des coefficients (en bas) des différents modèles

## 4 Conclusion

Blabla

## 5 Annexes

### 5.1 stabsel\_func.R

```
library(glmnet)
library(pROC)
```

```

library(purrr)

## ---- eval_func -----
# retourne un modèle de régression logistique entraîné
get_glm <- function(X_train, y_train, vars_idx) {
  return(glm("y ~ .",
            data = cbind(as.data.frame(as.matrix(X_train[, vars_idx])),
                          data.frame(y = y_train)),
            family = "binomial"))
}

# retourne les probabilités des classes
get_predictions <- function(X_train, y_train, X_test, vars_idx) {
  mod <- get_glm(X_train, y_train, vars_idx)
  return(predict(mod,
                newdata = as.data.frame(as.matrix(X_test[, vars_idx])),
                type = "response"))
}

# retourne le taux d'agrément
get_score <- function(y_test, y_pred) {
  return(sum(y_test == y_pred) / length(y_test) * 100)
}

# entraîne un modèle de régression logistique non pénalisé (avec un jeu
# restreint aux variables stables)
# estime les classes
# retourne le taux d'agrément
get_performance <- function(X_train, y_train, X_test, y_test, vars_idx) {
  y_pred <- as.integer(get_predictions(X_train, y_train, X_test, vars_idx) > .5)
  return(get_score(y_test, y_pred))
}

# entraîne un modèle de régression logistique non pénalisé (avec un jeu
# restreint aux variables stables)
# retourne l'amplitude des coefficients
get_coef_range <- function(X_train, y_train, vars_idx) {
  mod <- get_glm(X_train, y_train, vars_idx)
  return(max(mod$coefficients, na.rm = T) - min(mod$coefficients, na.rm = T))
}

get_coef_IQR <- function(X_train, y_train, vars_idx) {
  mod <- get_glm(X_train, y_train, vars_idx)
  return(quantile(mod$coefficients, .75, na.rm = T) -
        quantile(mod$coefficients, .25, na.rm = T))
}

get_coef_IDR <- function(X_train, y_train, vars_idx) {
  mod <- get_glm(X_train, y_train, vars_idx)
  return(quantile(mod$coefficients, .9, na.rm = T) -
        quantile(mod$coefficients, .1, na.rm = T))
}

# entraîne un modèle de régression logistique non pénalisé (avec un jeu

```

```

# restreint aux variables stables)
# estime les probabilités des classes
# retourne la courbe ROC
get_roc <- function(X_train, y_train, X_test, y_test, vars_idx) {
  y_pred <- get_predictions(X_train, y_train, X_test, vars_idx)
  return(roc(y_test, y_pred))
}

get_rocs <- function(X_train, y_train, X_test, y_test, mods_summary) {
  mods_roc <- lapply(mods_summary$vars_idx,
                     function(vars) get_roc(X_train, y_train, X_test, y_test,
                                              vars))

  names(mods_roc) <- rownames(mods_summary)
  return(mods_roc)
}

# à partir d'une matrice de coefficients (chemins) retourne un data.frame
get_path_as_df <- function(path, colnames) {
  vars <- which(path != 0, arr.ind = TRUE)
  path_df <- data.frame(cbind(path[vars], vars))
  colnames(path_df) <- colnames
  path_df$lambda <- as.integer(path_df$lambda)
  return(path_df)
}

## ---- stab_sel_func -----
# retourne une matrice de 0 / 1 indiquant si le coefficient d'une variable
# est non nul pour un lambda donné
get_selected_vars <- function(X_train, y_train, lambda, sample_size_coef) {
  full_size <- nrow(X_train)
  fold_idx <- sample(full_size, full_size * sample_size_coef)
  X_fold <- X_train[fold_idx, ]
  y_fold <- y_train[fold_idx]
  return(glmnet(X_fold, y_fold, family = "binomial", lambda = lambda)$beta != 0)
}

# retourne un chemin de stabilité
get_stability_path <- function(X_train, y_train, lambda, n_models = 100,
                              sample_size_coef = .5) {
  # liste de n_models matrices de 0 / 1 indiquant la sélection ou non d'une
  # variable pour un lambda donné
  vars_select <- sapply(1:n_models,
                       function(i) get_selected_vars(X_train, y_train, lambda,
                                                       sample_size_coef))

  # passage à une matrice de fréquences des variables
  # (pour des lambda différents)
  return(reduce(vars_select, `+`) / n_models)
}

# retourne un data.frame donnant le score et la taille du support selon le
# seuil de stabilité
run_stability_selection_model <- function(X_train, y_train, X_test, y_test,
                                          stability_path) {

```

```

# on garde la fréquence max sur les différents lambdas de chaque variable
vars_max_freq <- apply(stability_path, 1, max)
stability_indices <- seq(.6, 1, by = .05)
names(stability_indices) <- stability_indices
# index des variables stables pour différents seuils
vars_idx <- sapply(stability_indices,
  function(s_idx) which(vars_max_freq >= s_idx))
# scores des régressions logistiques sans pénalisation associés aux
# différents seuils
scores <- sapply(vars_idx,
  function(v_idx) get_performance(X_train, y_train, X_test,
    y_test, v_idx))
# nombre de variables sélectionnées pour les différentes seuils
nzero <- sapply(vars_idx, length)
# amplitude des coefficients
ranges <- sapply(vars_idx, function(v_idx) get_coef_IQR(X_train, y_train,
  v_idx))
return(data.frame(indice = stability_indices, score = scores, nzero = nzero,
  range = ranges, vars_idx = I(vars_idx)))
}

```

```

## ---- cv_func -----
# retourne un data.frame donnant le score et la taille du support selon le
# seuil de stabilité pour n_folds modèles
cv_run_stability_selection_model <- function(X_train, y_train, lambda,
  n_folds = 10) {
  # assignation de chaque observation à un fold
  folds_id <- sample(rep(seq(n_folds), length = nrow(X_train)))
  cv_stability_indices_summary <- lapply(1:n_folds, function(fid) {
    whichs <- folds_id == fid
    # restriction du jeu d'entraînement aux observations n'appartenant pas au
    # fold fid
    fold_X_train <- X_train[! whichs, ]
    fold_y_train <- y_train[! whichs]
    # et du jeu de test à celles appartenant à fid
    fold_X_test <- X_train[whichs, ]
    fold_y_test <- y_train[whichs]
    # appel à la fonction générique
    stability_path <- get_stability_path(fold_X_train, fold_y_train, lambda)
    return(run_stability_selection_model(fold_X_train, fold_y_train,
      fold_X_test, fold_y_test,
      stability_path))
  })
  return(do.call(rbind, cv_stability_indices_summary))
}

```

```

# à partir d'un data.frame contenant les résultats de la cross-validation
# retourne les indices optimaux
cv_get_optimal_stability_indices <- function(stability_indices_summary) {
  stability_indices <- unique(stability_indices_summary$indice)
  mean_sd <- lapply(stability_indices, function(sid) {
    scores <-

```

```

    stability_indices_summary$score[stability_indices_summary$indice == sid]
    return(data.frame(mean = mean(100 - scores), sd = sd(100 - scores)))
  })
  mean_sd <- do.call(rbind, mean_sd)
  mean_sd$indice <- stability_indices
  # indice de stabilité pour la meilleure classification
  indice_min <- max(mean_sd$indice[which(mean_sd$mean == min(mean_sd$mean))])
  # indice de stabilité pour le meilleur
  # compromis classification / taille du support
  indice_1sd <- max(mean_sd$indice[which(mean_sd$mean <=
                                          mean_sd$mean[[indice_min]] +
                                          mean_sd$sd[[indice_min]])])
  return(list(indice.min = indice_min, indice.1sd = indice_1sd))
}

```

## 5.2 stabsel\_data.R

```

load("data/TB.Rdata")
# transformation de la réponse en 0 / 1
y.train <- (y.train + 1) / 2
y.test <- (y.test + 1) / 2

```

## 5.3 stabsel\_run.R

```

## ---- stabsel_func -----
source("stabsel_func.R")

## ---- stabsel_data -----
source("stabsel_data.R")

## ---- lasso -----
lasso_mod <- cv.glmnet(X.train, y.train, type.measure = "class",
                      family = "binomial")
lasso_lambda <- lasso_mod$lambda
lasso_pred <- predict(lasso_mod, newx = X.test, type = "class",
                     s = lasso_mod$lambda.1se)[, 1]
lasso_vars_idx <- which(lasso_mod$glmnet.fit$beta[,
                     which(lasso_mod$lambda == lasso_mod$lambda.1se)] != 0)
lasso_summary <- data.frame(indice = NA,
                          score = get_score(y.test, lasso_pred),
                          nzero = lasso_mod$nzero[
                            which(lasso_mod$lambda == lasso_mod$lambda.1se)],
                          range = get_coef_IQR(X.train, y.train,
                                                lasso_vars_idx),
                          vars.idx = I(list(lasso_vars_idx)))
rownames(lasso_summary) <- "Lasso"

```



```

## ---- stab_sel -----
stability_path <- get_stability_path(X.train, y.train, lasso_lambda)
stab_sel_summary <- run_stability_selection_model(X.train, y.train, X.test,
                                                  y.test, stability_path)
rownames(stab_sel_summary) <- paste0("StabSel-", round(stab_sel_summary$indice,
                                                       digits = 2))

## ---- cv_stab_sel -----
cv_stab_sel_summary <- cv_run_stability_selection_model(X.train, y.train,
                                                       lasso_lambda)

best_indices <- cv_get_optimal_stability_indices(cv_stab_sel_summary)
indice_min_idx <- which(stab_sel_summary$indice == best_indices$indice.min)
indice_1sd_idx <- which(stab_sel_summary$indice == best_indices$indice.1sd)
rownames(stab_sel_summary)[[indice_min_idx]] <-
  paste0(rownames(stab_sel_summary)[[indice_min_idx]], " (cv_min)")
rownames(stab_sel_summary)[[indice_1sd_idx]] <-
  paste0(rownames(stab_sel_summary)[[indice_1sd_idx]], " (cv_1sd)")

## ---- paths -----
# Construction des data.frames des chemins
# De régularisation
reg_path <- get_path_as_df(lasso_mod$glmnet.fit$beta,
                          c("beta", "var", "lambda"))

## De sélection
stab_path <- get_path_as_df(stability_path, c("freq", "var", "lambda"))

# Jointure avec les fréquences/probabilités de sélection maximales pour la
# coloration
max_freq <- by(stab_path, list(stab_path$var),
               function(df) data.frame(var = unique(df$var),
                                         max_freq = max(df$freq)))
max_freq <- as.data.frame(do.call(rbind, max_freq))
max_freq$var <- as.factor(max_freq$var)
max_freq$max_freq <- as.numeric(max_freq$max_freq)
reg_path <- merge(reg_path, max_freq, by = "var", all.x = TRUE)
reg_path[is.na(reg_path$max_freq), "max_freq"] <- 0
stab_path <- merge(stab_path, max_freq, by = "var")

# cast en factor avec un ordre de levels précis pour que les chemins des
# variables les plus stables apparaissent biens
reg_path$var <- factor(reg_path$var,
                      levels = unique(reg_path[order(reg_path$max_freq),
                                                       "var"]))
stab_path$var <- factor(stab_path$var,
                      levels = unique(stab_path[order(stab_path$max_freq),
                                                       "var"]))

## ---- save -----
save(lasso_summary, stab_sel_summary, file = "data/summaries.Rdata")

```

```
save(reg_path, stab_path, file = "data/paths.Rdata")
```

## 5.4 stabsel\_fig.R

```
## ---- stabsel_func -----
source("stabsel_func.R")

library(ggplot2)

## ---- fig_paths -----
get_fig_stab <- function(stab_path) {
  # meme couleur de .8 a 1
  stab_path$color <- sapply(stab_path$max_freq, function(f) min(f, .8))
  ggplot(data = stab_path) +
    geom_line(aes(x = lambda, y = freq, group = var, color = color)) +
    scale_color_gradient(low = "yellow", high = "red", limits = c(.6, .8),
                        guide = "none") +
    labs(x = "Lambda index", y = "Probabilité de sélection")
}

get_fig_reg <- function(reg_path) {
  # meme couleur de .8 a 1
  reg_path$color <- sapply(reg_path$max_freq, function(f) min(f, .8))
  ggplot(data = reg_path) +
    geom_line(aes(x = lambda, y = beta, group = var, color = color)) +
    scale_color_gradient(low = "yellow", high = "red", limits = c(0.6, .8),
                        guide = "none") +
    labs(x = "Lambda index", y = "Coefficient")
}

## ---- fig_roc -----
get_fig_roc <- function(mods_roc) {
  ggroc(mods_roc) +
    labs(x = "Spécificité", y = "Sensibilité") +
    scale_color_discrete(name = "Modèle") +
    theme_bw()
}

get_fig_auc <- function(mods_roc) {
  mods_auc <- data.frame(auc = sapply(mods_roc, function(mod_roc) mod_roc$auc),
                        model = names(mods_roc))
  ggplot(data = mods_auc) +
    geom_bar(aes(x = model, y = auc, fill = model), stat = "identity") +
    scale_fill_discrete(name = "Modèle") +
    labs(x = "Modèle", y = "AUC") +
    theme_bw() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))
}
```

```
## ---- fig_score -----
get_fig_nz_score <- function(mods_summary) {
  ggplot(data = mods_summary) +
    geom_point(aes(x = nzzero, y = score, colour = rownames(mods_summary),
                  shape = rownames(mods_summary))) +
    scale_shape_manual(name = "Modèle", values = 1:nrow(mods_summary)) +
    scale_color_discrete(name = "Modèle") +
    labs(x = "Taille du support", y = "Taux d'agrément") +
    theme_bw()
}

get_fig_amp_score <- function(mods_summary) {
  ggplot(data = mods_summary) +
    geom_point(aes(x = range, y = score, colour = rownames(mods_summary),
                  shape = rownames(mods_summary))) +
    scale_shape_manual(name = "Modèle", values = 1:nrow(mods_summary)) +
    scale_color_discrete(name = "Modèle") +
    labs(x = "Amplitude des coefficients (IQR)", y = "Taux d'agrément") +
    theme_bw()
}
```

## References

- [1] Peter Bühlmann Nicolai Meinshausen. “Stability selection”. In: *Journal of the Royal Statistical Society, Series B* 72.4 (2010), pp. 417–473. DOI: <https://doi.org/10.48550/arXiv.0809.2932>.