Model

Eduardo Yuki Yada

Imports

```
library(tidyverse)
library(yaml)
library(tidymodels)
library(usemodels)
library(vip)
```

Loading data

```
load('../dataset/processed_data.RData')
load('../dataset/processed_dictionary.RData')

columns_list <- yaml.load_file("./auxiliar/columns_list.yaml")

outcome_column <- params$outcome_column
features_list <- params$features_list
```

Filtering eligible pacients

```
df = df %>%
  filter(disch_outcomes_t0 == 0)

df %>% dim
```

```
## [1] 15766   239
```

Eligible features

```
eligible_columns = df_names %>%
  filter(momento.aquisicao == 'Admissão t0') %>%
  .$variable.name

exception_columns = c('death_intraop', 'death_intraop_1')

correlated_columns = c('year_procedure_1', # com year_adm_t0
                       'age_surgery_1', # com age
                       'admission_pre_t0_count', # com admission_t0
                       'atb', # com meds_antimicrobianos
                       'classe_meds_cardio_qtde', # com classe_meds_qtde
                       'suporte_hemod' # com proced_invasivos_qtde
                       )

eligible_features = eligible_columns %>%
  base::intersect(c(columns_list$categorical_columns, columns_list$numerical_columns)) %>%
  setdiff(c(exception_columns, correlated_columns))
```

```r
if (is.null(features_list)) {
  features = eligible_features
} else {
  features = base::intersect(eligible_features, features_list)
}

length(features)
```

```
## [1] 78
```

```r
print(features)
```

```
##  [1] "education_level"          "underlying_heart_disease" "heart_disease"
##  [4] "nyha_basal"               "prior_mi"                 "heart_failure"
##  [7] "transplant"               "endocardites"             "hemodialysis"
## [10] "comorbidities_count"      "procedure_type_1"         "reop_type_1"
## [13] "procedure_type_new"       "cied_final_1"             "cied_final_group_1"
## [16] "admission_t0"             "admission_pre_t0_180d"    "icu_t0"
## [19] "dialysis_t0"              "disch_outcomes_t0"        "admission_t0_emergency"
## [22] "aco"                      "antiarritmico"            "betabloqueador"
## [25] "ieca_bra"                 "dva"                      "digoxina"
## [28] "estatina"                 "diuretico"                "vasodilatador"
## [31] "insuf_cardiaca"           "espironolactona"          "bloq_calcio"
## [34] "antiplaquetario_ev"       "insulina"                 "anticonvulsivante"
## [37] "psicofarmacos"            "antifungico"              "antiviral"
## [40] "classe_meds_qtde"         "meds_cardiovasc_qtde"     "meds_antimicrobianos"
## [43] "cec"                      "transplante_cardiaco"     "outros_proced_cirurgicos"
## [46] "icp"                      "intervencao_cv"           "cateterismo"
## [49] "eletrofisiologia"         "cateter_venoso_central"   "proced_invasivos_qtde"
## [52] "cve_desf"                 "transfusao"               "equipe_multiprof"
## [55] "ecg"                      "holter"                   "tilt_teste"
## [58] "metodos_graficos_qtde"    "laboratorio"              "cultura"
## [61] "analises_clinicas_qtde"   "citologia"                "biopsia"
## [64] "histopatologia_qtde"      "angio_rm"                 "angio_tc"
## [67] "cintilografia"            "ecocardiograma"           "endoscopia"
## [70] "flebografia"              "pet_ct"                   "ultrassom"
## [73] "tomografia"               "radiografia"              "ressonancia"
## [76] "exames_imagem_qtde"       "bic"                      "mpp"
```

Train test split (70%/30%)

```r
set.seed(42)

df[columns_list$outcome_columns] <- lapply(df[columns_list$outcome_columns], factor)

df_split <- initial_split(df %>% dplyr::select(all_of(c(features, outcome_column))),
                          prop = .7, strata = all_of(outcome_column))
df_train <- training(df_split)
df_test <- testing(df_split)

dim(df_train)[1] / dim(df)[1]
```

```
## [1] 0.6999873
```

```r
dim(df_test)[1] / dim(df)[1]
```

```
## [1] 0.3000127
```

Global parameters

```r
k = 4 # Number of folds for cross validation

set.seed(234)
df_folds <- vfold_cv(df_train, v = k,
                     strata = all_of(outcome_column))
```

Functions

```r
validation = function(model_fit, new_data) {
  library(pROC)
  library(caret)

  test_predictions_prob <-
    predict(model_fit, new_data = new_data, type = "prob") %>%
    rename_at(vars(starts_with(".pred_")), ~ str_remove(., ".pred_")) %>%
    .$`1`

  pROC_obj <- roc(
    new_data[[outcome_column]],
    test_predictions_prob,
    smoothed = TRUE,
    # arguments for ci
    ci = TRUE,
    ci.alpha = 0.9,
    stratified = FALSE,
    # arguments for plot
    plot = TRUE,
    auc.polygon = TRUE,
    max.auc.polygon = TRUE,
    grid = TRUE,
    print.auc = TRUE,
    show.thres = TRUE
  )

  sens.ci <- ci.se(pROC_obj)
  plot(sens.ci, type = "shape", col = "lightblue")
  plot(sens.ci, type = "bars")

  test_predictions_class <-
    predict(model_fit, new_data = new_data, type = "class") %>%
    rename_at(vars(starts_with(".pred_")), ~ str_remove(., ".pred_")) %>%
    .$class

  conf_matrix = table(test_predictions_class, new_data[[outcome_column]])

  confusionMatrix(conf_matrix) %>% print

  return(pROC_obj)
}
```

Boosted Tree (XGBoost)

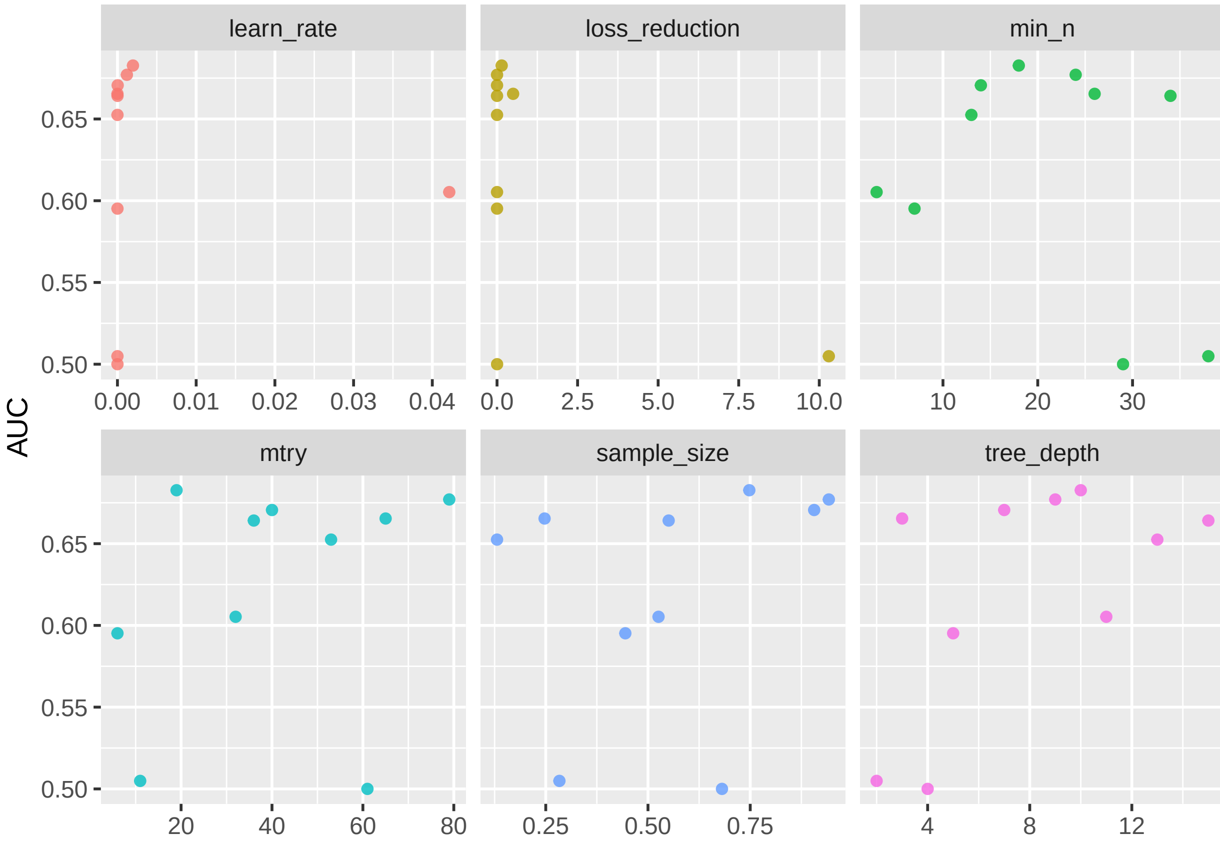
```r
xgboost_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
  step_string2factor(all_nominal_predictors()) %>%
  step_novel(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors(), one_hot = TRUE) %>%
```

```r
  step_zv(all_predictors())

xgboost_spec <- boost_tree(
  trees = 500,
  tree_depth = tune(),
  min_n = tune(),
  loss_reduction = tune(),
  sample_size = tune(),
  mtry = tune(),
  learn_rate = tune()
) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

xgboost_grid <- grid_latin_hypercube(
  tree_depth(),
  min_n(),
  loss_reduction(),
  sample_size = sample_prop(),
  finalize(mtry(), df_train),
  learn_rate(),
  size = 10
)

xgboost_workflow <-
  workflow() %>%
  add_recipe(xgboost_recipe) %>%
  add_model(xgboost_spec)

xgboost_tune <-
  xgboost_workflow %>%
  tune_grid(resamples = df_folds,
            grid = xgboost_grid)

xgboost_tune %>%
  show_best("roc_auc")
```

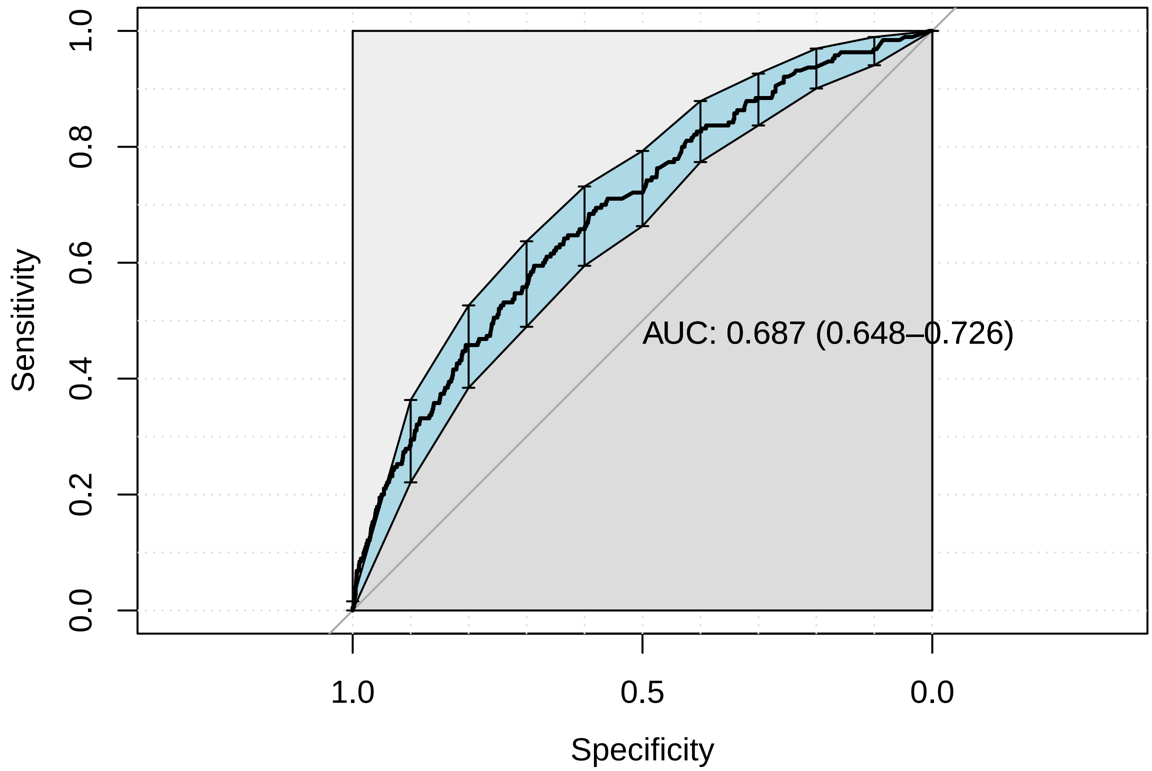
```
## # A tibble: 5 x 12
##    mtry min_n tree_depth  learn_rate loss_reduction sample_size .metric .estimator  mean
##   <int> <int>      <int>       <dbl>          <dbl>       <dbl> <chr>   <chr>      <dbl>
## 1    19    18         10 0.00197           1.45e- 1       0.748 roc_auc binary     0.683
## 2    79    24          9 0.00120           1.38e- 7       0.942 roc_auc binary     0.677
## 3    40    14          7 0.0000268         1.69e- 9       0.906 roc_auc binary     0.671
## 4    65    26          3 0.000000996       4.99e- 1       0.247 roc_auc binary     0.665
## 5    36    34         15 0.00000849        1.43e-10       0.551 roc_auc binary     0.664
## # ... with 3 more variables: n <int>, std_err <dbl>, .config <chr>
```

```r
best_xgboost <- xgboost_tune %>%
  select_best("roc_auc")

xgboost_tune %>%
  collect_metrics() %>%
  filter(.metric == "roc_auc") %>%
  select(mean, mtry:sample_size) %>%
  pivot_longer(mtry:sample_size,
               values_to = "value",
               names_to = "parameter"
  ) %>%
  ggplot(aes(value, mean, color = parameter)) +
  geom_point(alpha = 0.8, show.legend = FALSE) +
  facet_wrap(~parameter, scales = "free_x") +
```

```
  labs(x = NULL, y = "AUC")
```

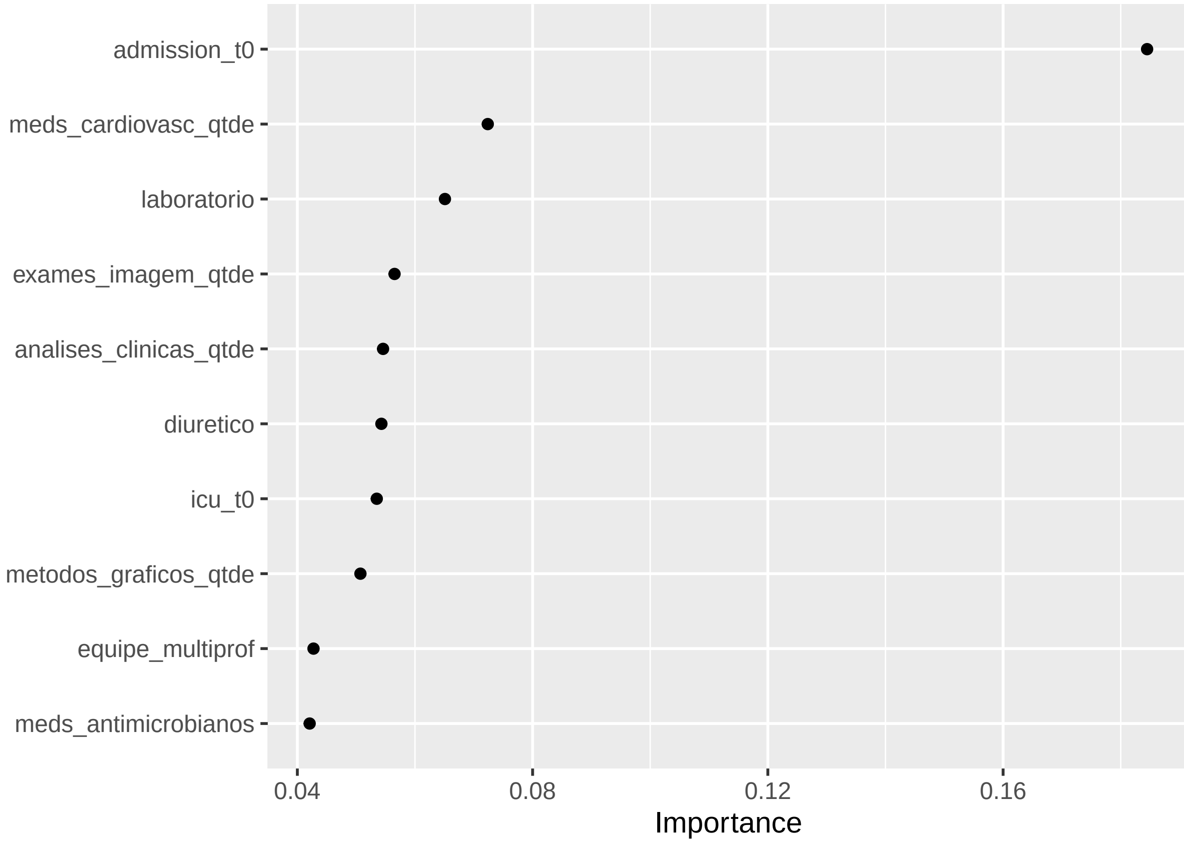



```
final_xgboost_workflow <-
  xgboost_workflow %>%
  finalize_workflow(best_xgboost)

last_xgboost_fit <-
  final_xgboost_workflow %>%
  last_fit(df_split)

final_xgboost_fit <- extract_workflow(last_xgboost_fit)

xgboost_auc = validation(final_xgboost_fit, df_test)
```



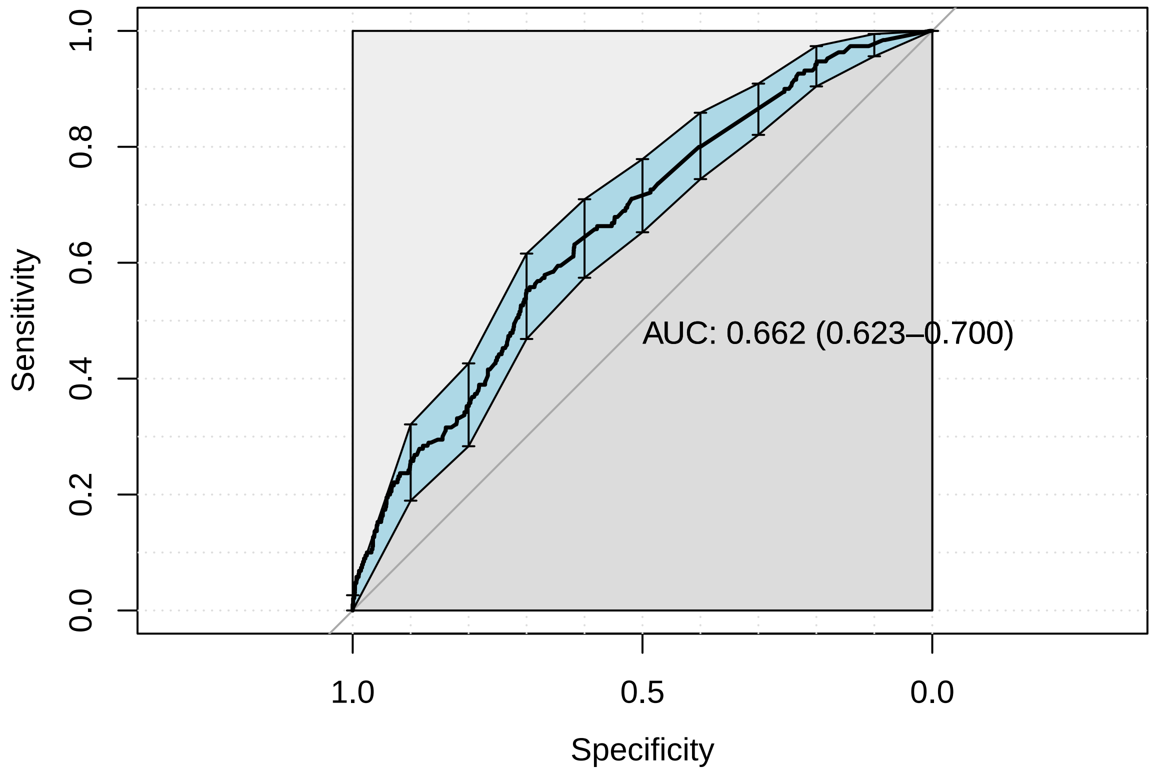

```
## Confusion Matrix and Statistics
##
##
## test_predictions_class    0    1
##                      0 4540  190
##                      1    0    0
##
##                Accuracy : 0.9598
##                  95% CI : (0.9538, 0.9652)
##     No Information Rate : 0.9598
##     P-Value [Acc > NIR] : 0.5193
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9598
##          Neg Pred Value :    NaN
##              Prevalence : 0.9598
##          Detection Rate : 0.9598
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

```
final_xgboost_fit %>%
  fit(data = df_train) %>%
  extract_fit_parsnip() %>%
  vip(geom = "point")
```



GLM

```
glmnet_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
  step_string2factor(all_nominal_predictors()) %>%
  step_novel(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_numeric_predictors())

glmnet_spec <-
  logistic_reg(penalty = 0) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

glmnet_workflow <-
  workflow() %>%
  add_recipe(glmnet_recipe) %>%
  add_model(glmnet_spec)

glm_fit <- glmnet_workflow %>%
  fit(df_train)

# glm_fit %>%
#   pull_workflow_fit() %>%
#   tidy()

glm_auc = validation(glm_fit, df_test)
```

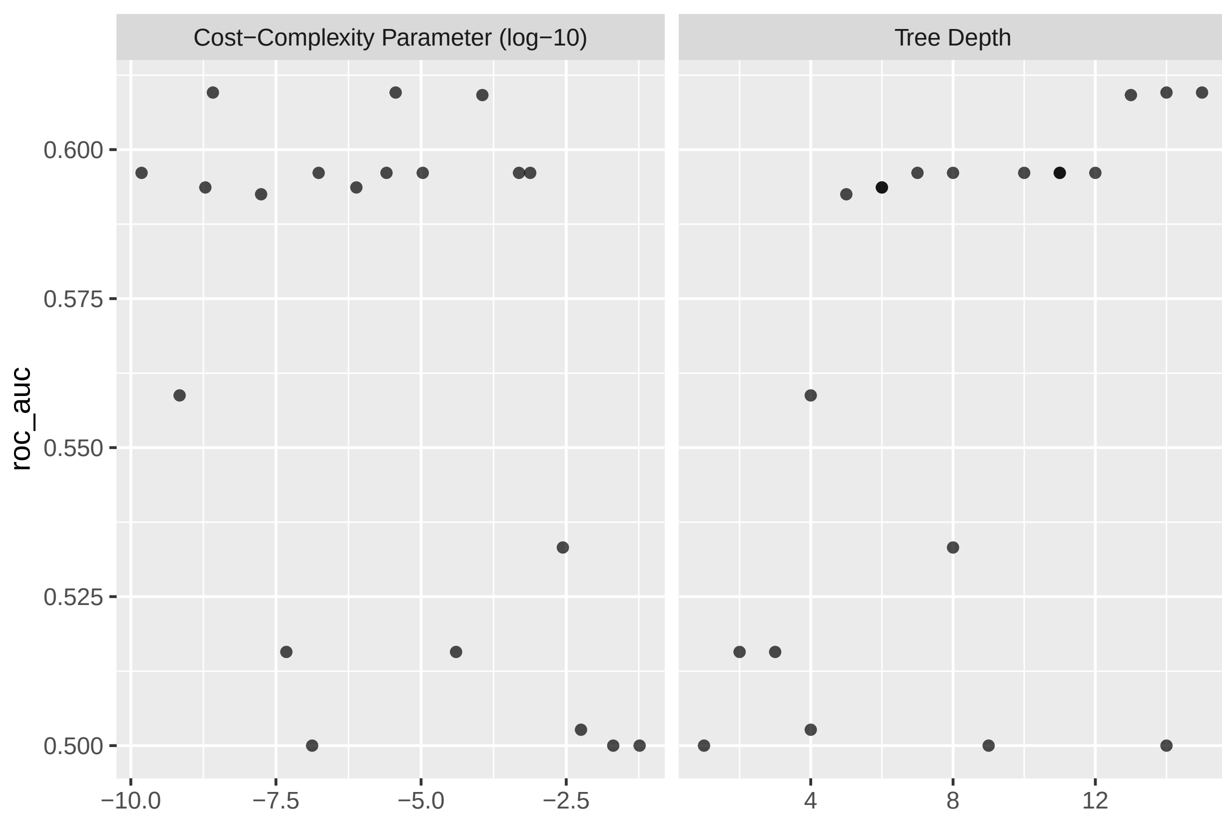



```
## Confusion Matrix and Statistics
##
##
## test_predictions_class    0    1
##                      0 4540  189
##                      1    0    1
##
##                Accuracy : 0.96
##                  95% CI : (0.9541, 0.9654)
##     No Information Rate : 0.9598
##     P-Value [Acc > NIR] : 0.4898
##
##                   Kappa : 0.0101
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.000000
##             Specificity : 0.005263
##          Pos Pred Value : 0.960034
##          Neg Pred Value : 1.000000
##              Prevalence : 0.959831
##          Detection Rate : 0.959831
##    Detection Prevalence : 0.999789
##       Balanced Accuracy : 0.502632
##
##        'Positive' Class : 0
##
```

Decision Tree

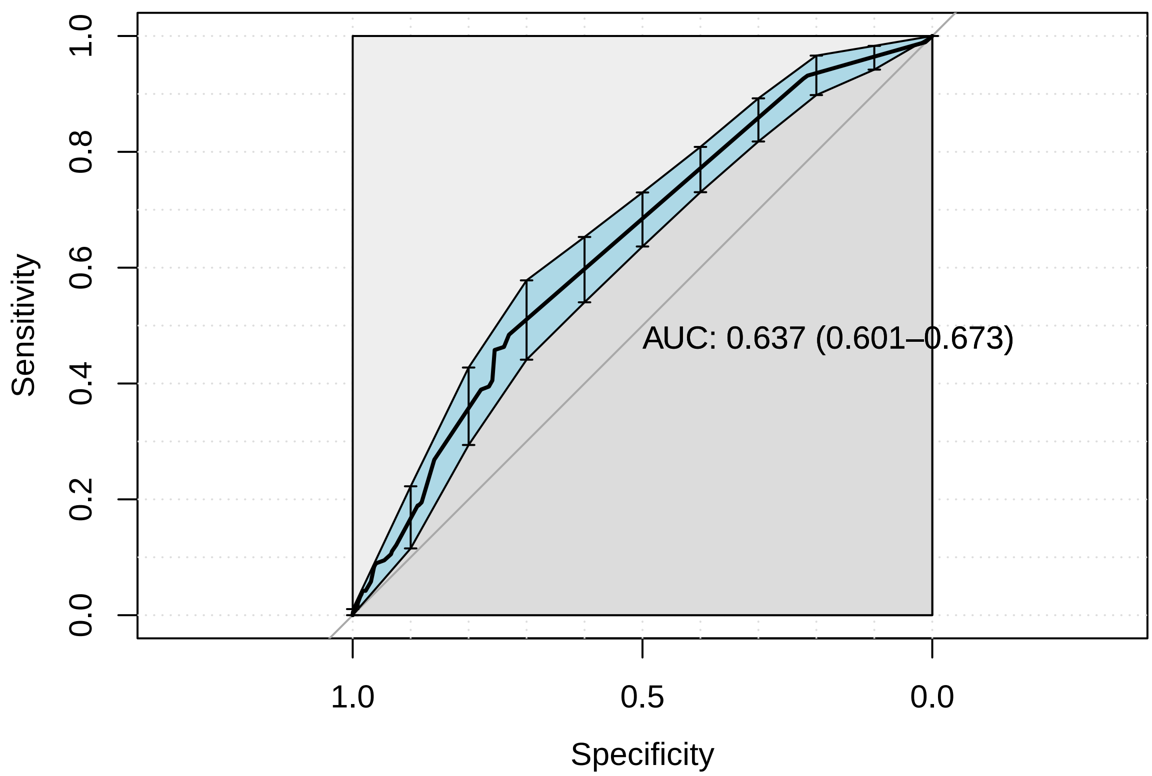
```
tree_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
```

```
  step_string2factor(all_nominal_predictors()) %>%
  step_novel(all_nominal_predictors()) %>%
  step_zv(all_predictors())

tree_spec <-
  decision_tree(cost_complexity = tune(),
                tree_depth = tune()) %>%
  set_mode("classification") %>%
  set_engine("rpart")

tree_grid <- grid_latin_hypercube(cost_complexity(),
                                  tree_depth(),
                                  size = 20)

tree_workflow <-
  workflow() %>%
  add_recipe(tree_recipe) %>%
  add_model(tree_spec)

tree_tune <-
  tree_workflow %>%
  tune_grid(resamples = df_folds,
            grid = tree_grid)

tree_tune %>%
  collect_metrics()

autoplot(tree_tune, metric = "roc_auc")
```

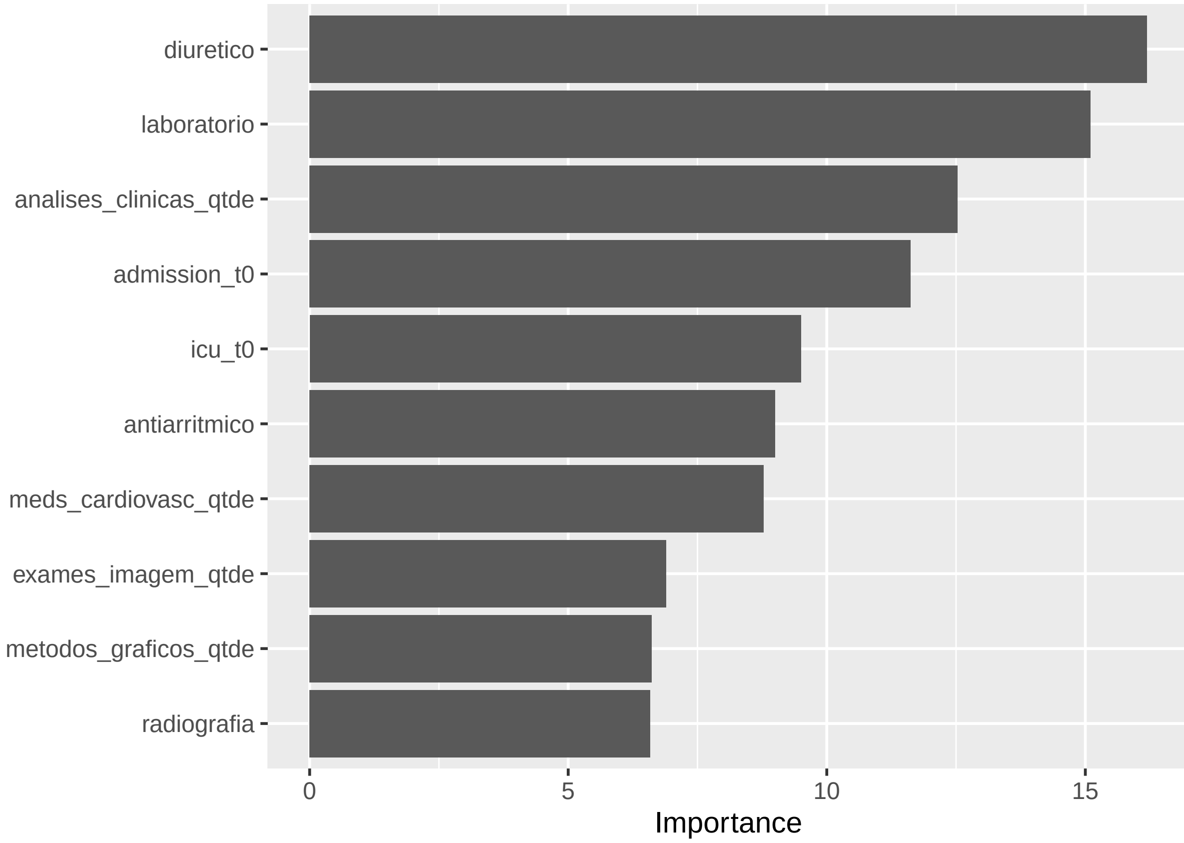



```
tree_tune %>%
  show_best("roc_auc")

best_tree <- tree_tune %>%
```

```
  select_best("roc_auc")

final_tree_workflow <-
  tree_workflow %>%
  finalize_workflow(best_tree)

last_tree_fit <-
  final_tree_workflow %>%
  last_fit(df_split)

final_tree_fit <- extract_workflow(last_tree_fit)

tree_auc = validation(final_tree_fit, df_test)
```



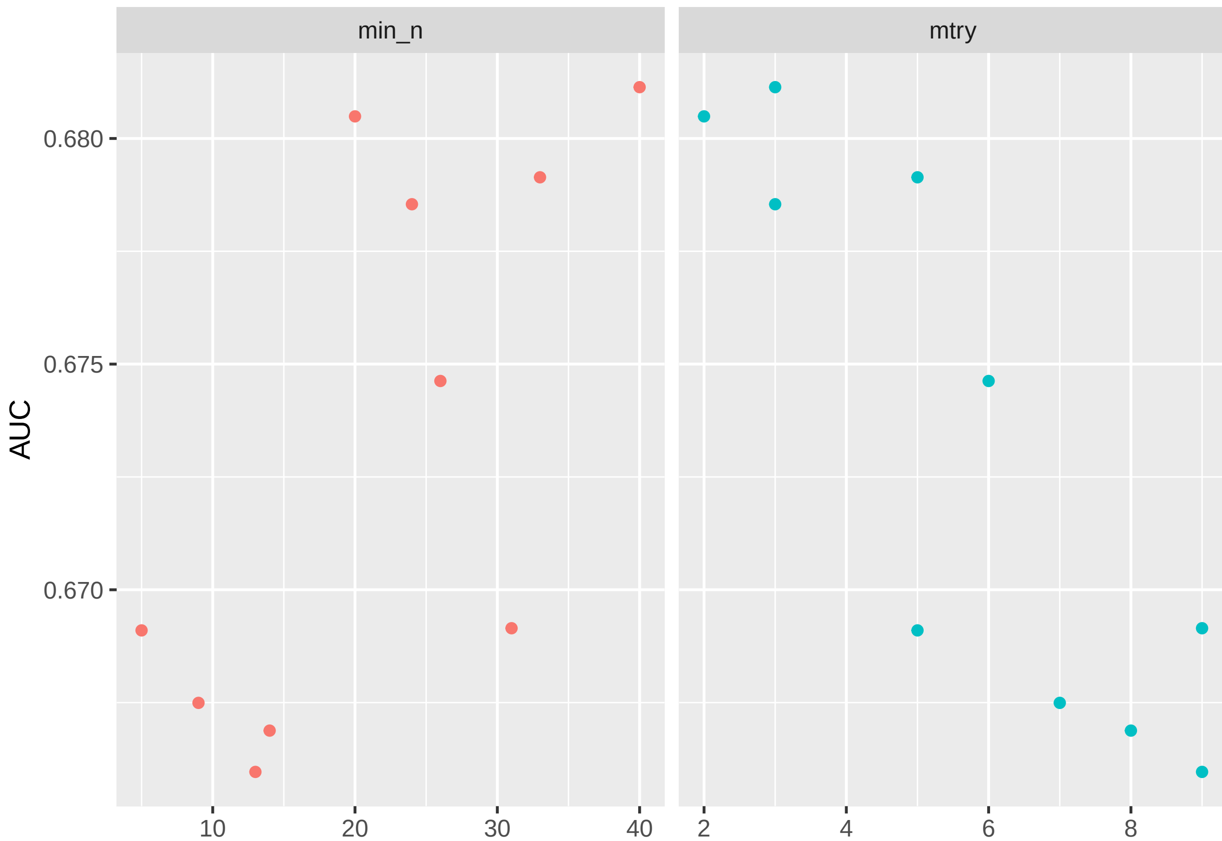

```
if (tree_auc$auc > 0.55){
  final_tree_fit %>%
    extract_fit_parsnip() %>%
    vip()
}
```



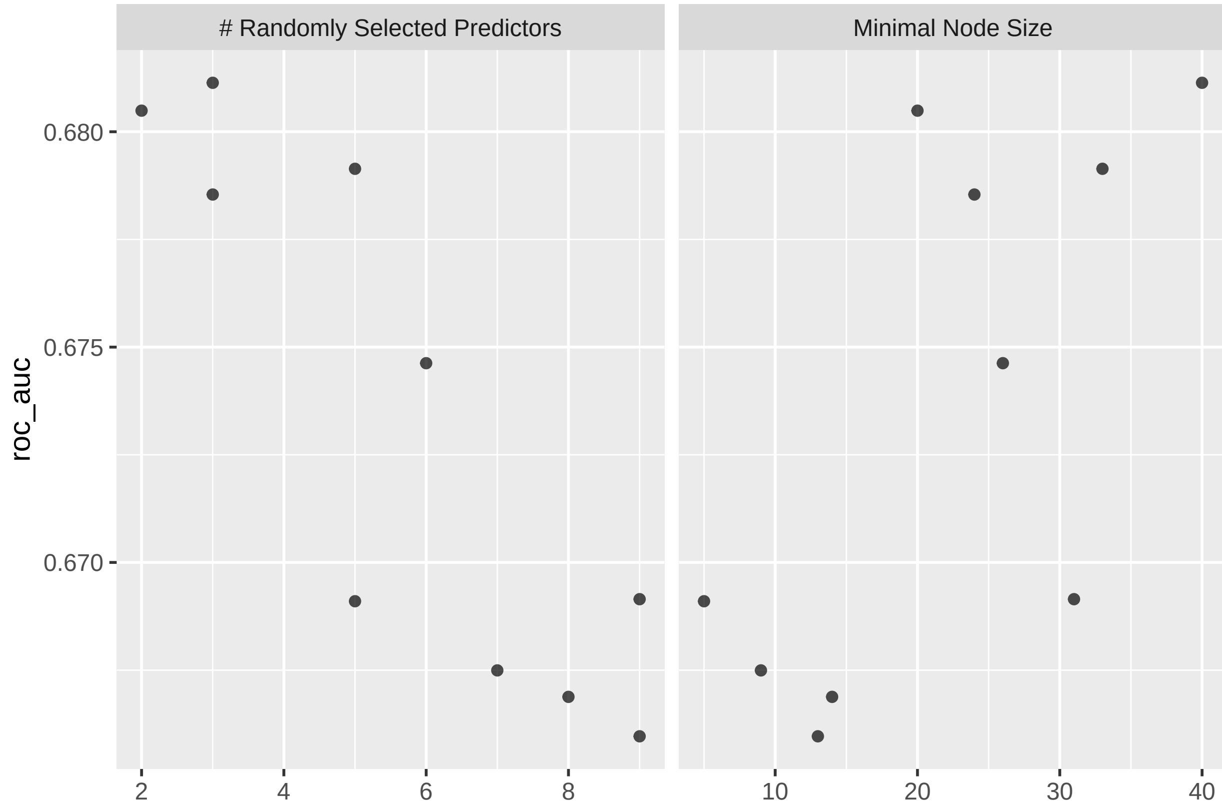
Random Forest

```
rf_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
  step_string2factor(all_nominal_predictors()) %>%
  step_novel(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_impute_mode(all_nominal_predictors()) %>%
  step_impute_mean(all_numeric_predictors())

rf_spec <-
  rand_forest(mtry = tune(),
              trees = 1000,
              min_n = tune()) %>%
  set_mode("classification") %>%
  set_engine("ranger")

rf_grid <- grid_latin_hypercube(mtry(range = c(1, 10)),
                                min_n(),
                                size = 10)

rf_workflow <-
  workflow() %>%
  add_recipe(rf_recipe) %>%
  add_model(rf_spec)

rf_tune <-
  rf_workflow %>%
  tune_grid(resamples = df_folds,
            grid = rf_grid)
```

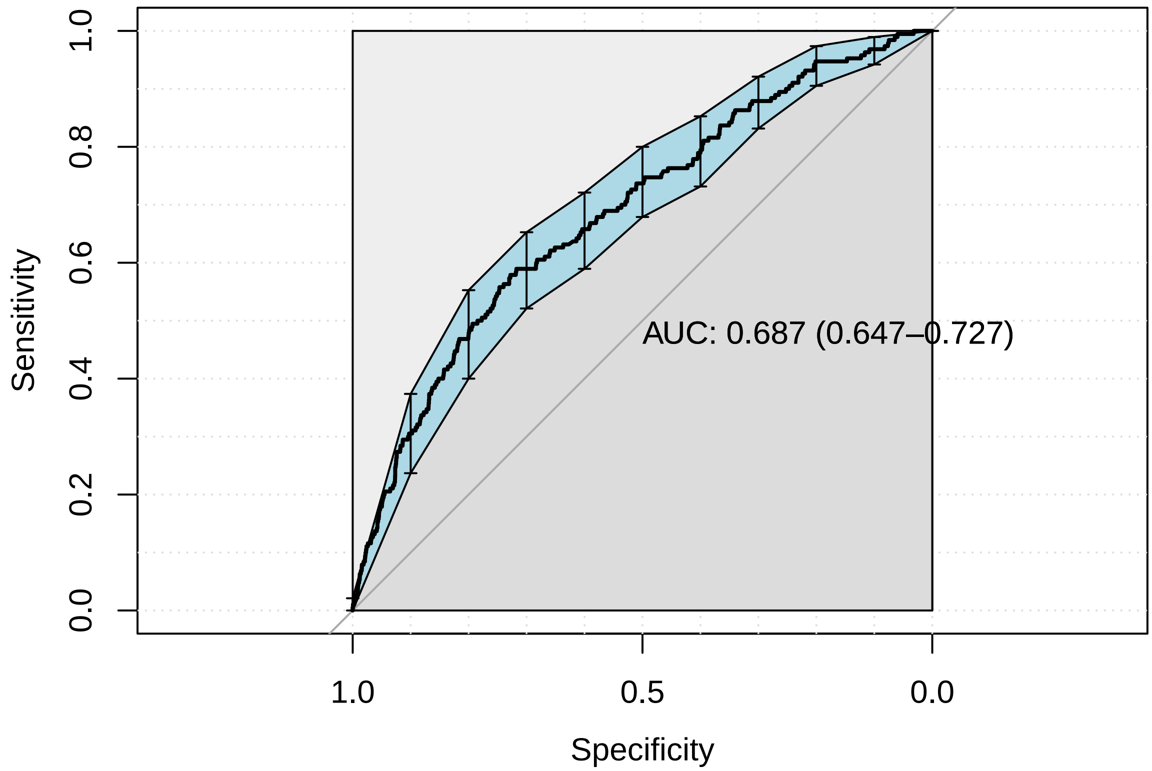
```
rf_tune %>%
  collect_metrics()

rf_tune %>%
  collect_metrics() %>%
  filter(.metric == "roc_auc") %>%
  select(mean, min_n, mtry) %>%
  pivot_longer(min_n:mtry,
    values_to = "value",
    names_to = "parameter"
  ) %>%
  ggplot(aes(value, mean, color = parameter)) +
  geom_point(show.legend = FALSE) +
  facet_wrap(~parameter, scales = "free_x") +
  labs(x = NULL, y = "AUC")
```




```
autoplot(rf_tune, metric = "roc_auc")
```




```
rf_tune %>%
  show_best("roc_auc")

best_rf <- rf_tune %>%
  select_best("roc_auc")

final_rf_workflow <-
  rf_workflow %>%
  finalize_workflow(best_rf)

last_rf_fit <-
  final_rf_workflow %>%
  last_fit(df_split)

final_rf_fit <- extract_workflow(last_rf_fit)

rf_auc = validation(final_rf_fit, df_test)
```




```
# final_rf_fit %>%
#   extract_fit_parsnip() %>%
#   vip()
```

KNN

```
# knn_recipe <-
#   recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
#   step_string2factor(all_nominal_predictors()) %>%
#   step_novel(all_nominal_predictors()) %>%
#   step_zv(all_predictors()) %>%
#   step_dummy(all_nominal_predictors()) %>%
#   step_impute_mode(all_nominal_predictors()) %>%
#   step_impute_mean(all_numeric_predictors())
#
# knn_spec <-
#   nearest_neighbor(neighbors = tune(),
#                    weight_func = tune(),
#                    dist_power = tune()) %>%
#   set_mode("classification") %>%
#   set_engine("kknn")
#
# knn_grid <- grid_latin_hypercube(neighbors(),
#                                  weight_func(),
#                                  dist_power(),
#                                  size = 10)
#
# knn_workflow <-
#   workflow() %>%
#   add_recipe(knn_recipe) %>%
#   add_model(knn_spec)
#
```

```r
# knn_tune <-
#   knn_workflow %>%
#   tune_grid(resamples = df_folds,
#             grid = knn_grid)
#
# knn_tune %>%
#   collect_metrics()
#
# autoplot(knn_tune, metric = "roc_auc")
#
# knn_tune %>%
#   show_best("roc_auc")
#
# best_knn <- knn_tune %>%
#   select_best("roc_auc")
#
# final_knn_workflow <-
#   knn_workflow %>%
#   finalize_workflow(best_knn)
#
# last_knn_fit <-
#   final_knn_workflow %>%
#   last_fit(df_split)
#
# final_knn_fit <- extract_workflow(last_knn_fit)
#
# knn_auc = validation(final_knn_fit, df_test)
```

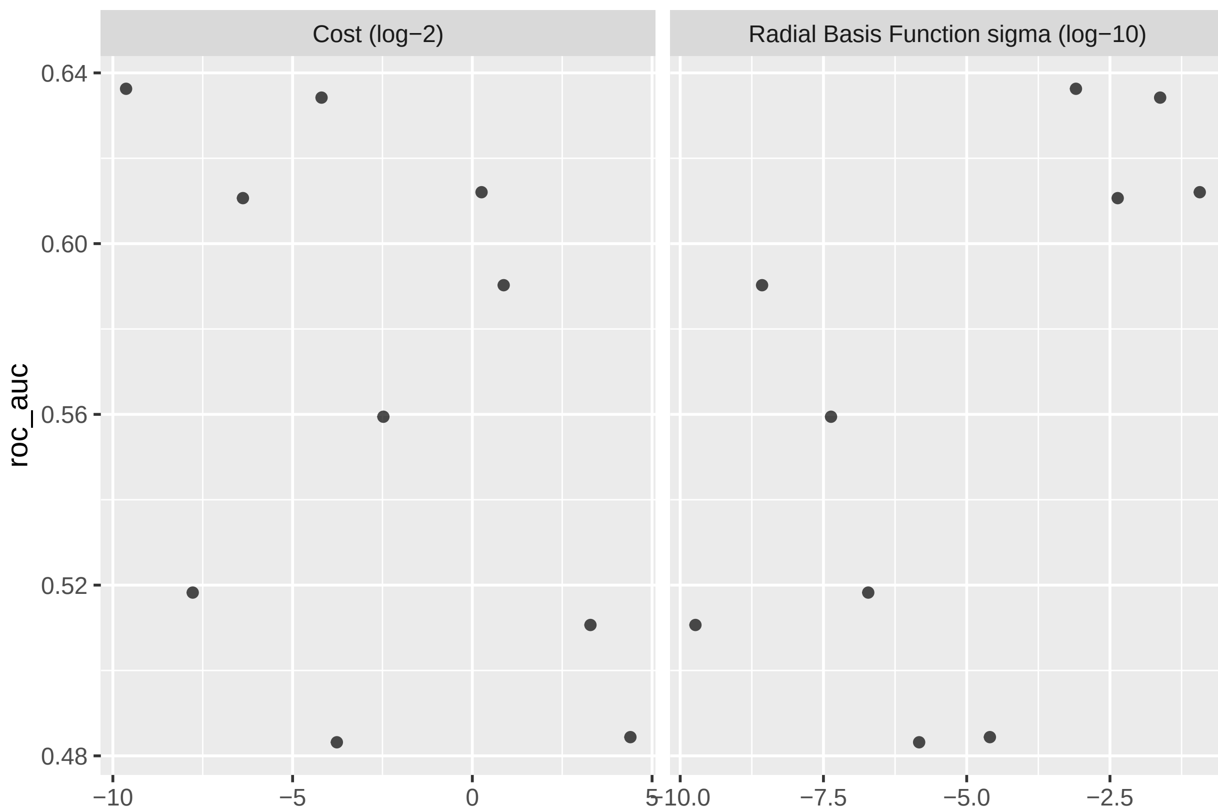
SVM

```r
svm_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
  step_string2factor(all_nominal_predictors()) %>%
  step_novel(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_impute_mode(all_nominal_predictors()) %>%
  step_impute_mean(all_numeric_predictors())

svm_spec <-
  svm_rbf(cost = tune(), rbf_sigma = tune()) %>%
  set_mode("classification") %>%
  set_engine("kernlab")

svm_grid <- grid_latin_hypercube(cost(),
                                 rbf_sigma(),
                                 size = 10)

svm_workflow <-
  workflow() %>%
  add_recipe(svm_recipe) %>%
  add_model(svm_spec)

svm_tune <-
  svm_workflow %>%
  tune_grid(resamples = df_folds,
            grid = svm_grid)

svm_tune %>%
```

```r
  collect_metrics()
```

```
## # A tibble: 20 x 8
##        cost rbf_sigma .metric  .estimator  mean     n std_err .config
##       <dbl>     <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##   1  0.00126  8.06e- 4 accuracy binary     0.963     4 0.00309 Preprocessor1_Model01
##   2  0.00126  8.06e- 4 roc_auc  binary     0.636     4 0.0200  Preprocessor1_Model01
##   3  0.00455  1.91e- 7 accuracy binary     0.963     4 0.00309 Preprocessor1_Model02
##   4  0.00455  1.91e- 7 roc_auc  binary     0.518     4 0.0271  Preprocessor1_Model02
##   5  0.0546   2.38e- 2 accuracy binary     0.963     4 0.00309 Preprocessor1_Model03
##   6  0.0546   2.38e- 2 roc_auc  binary     0.634     4 0.00775 Preprocessor1_Model03
##   7  0.0733   1.48e- 6 accuracy binary     0.963     4 0.00309 Preprocessor1_Model04
##   8  0.0733   1.48e- 6 roc_auc  binary     0.483     4 0.0282  Preprocessor1_Model04
##   9  9.75     1.85e-10 accuracy binary     0.963     4 0.00309 Preprocessor1_Model05
## 10  9.75     1.85e-10 roc_auc  binary     0.511     4 0.0231  Preprocessor1_Model05
## 11  1.83     2.68e- 9 accuracy binary     0.963     4 0.00309 Preprocessor1_Model06
## 12  1.83     2.68e- 9 roc_auc  binary     0.590     4 0.0182  Preprocessor1_Model06
## 13 21.1      2.54e- 5 accuracy binary     0.961     4 0.00298 Preprocessor1_Model07
## 14 21.1      2.54e- 5 roc_auc  binary     0.484     4 0.0139  Preprocessor1_Model07
## 15  1.19     1.17e- 1 accuracy binary     0.963     4 0.00309 Preprocessor1_Model08
## 16  1.19     1.17e- 1 roc_auc  binary     0.612     4 0.0111  Preprocessor1_Model08
## 17  0.0120   4.32e- 3 accuracy binary     0.963     4 0.00309 Preprocessor1_Model09
## 18  0.0120   4.32e- 3 roc_auc  binary     0.611     4 0.0107  Preprocessor1_Model09
## 19  0.180    4.29e- 8 accuracy binary     0.963     4 0.00309 Preprocessor1_Model10
## 20  0.180    4.29e- 8 roc_auc  binary     0.559     4 0.0116  Preprocessor1_Model10
```

```r
autoplot(svm_tune, metric = "roc_auc")
```

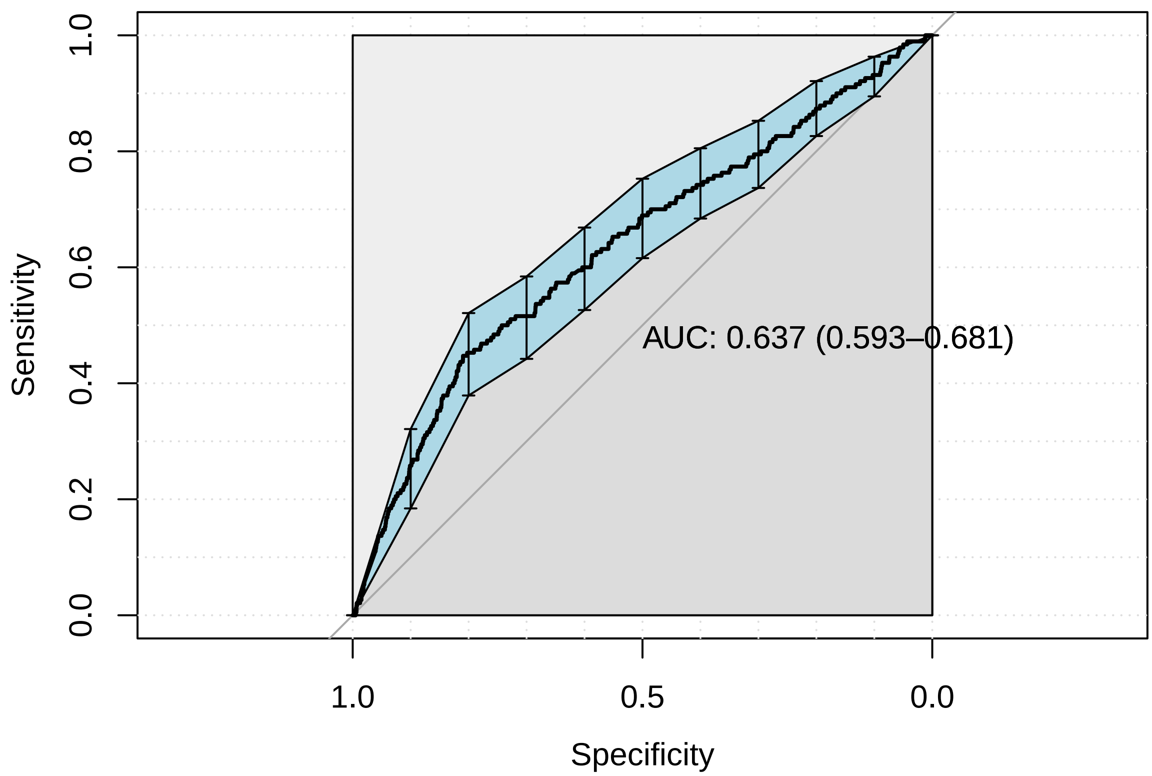



```r
svm_tune %>%
  show_best("roc_auc")
```

```
## # A tibble: 5 x 8
##      cost     rbf_sigma .metric .estimator  mean     n std_err .config
##     <dbl>         <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
```

```
## 1 0.00126 0.000806      roc_auc binary     0.636     4 0.0200  Preprocessor1_Model01
## 2 0.0546  0.0238        roc_auc binary     0.634     4 0.00775 Preprocessor1_Model03
## 3 1.19    0.117         roc_auc binary     0.612     4 0.0111  Preprocessor1_Model08
## 4 0.0120  0.00432       roc_auc binary     0.611     4 0.0107  Preprocessor1_Model09
## 5 1.83    0.00000000268 roc_auc binary     0.590     4 0.0182  Preprocessor1_Model06
```

```
best_svm <- svm_tune %>%
  select_best("roc_auc")

final_svm_workflow <-
  svm_workflow %>%
  finalize_workflow(best_svm)

last_svm_fit <-
  final_svm_workflow %>%
  last_fit(df_split)

final_svm_fit <- extract_workflow(last_svm_fit)

svm_auc = validation(final_svm_fit, df_test)
```

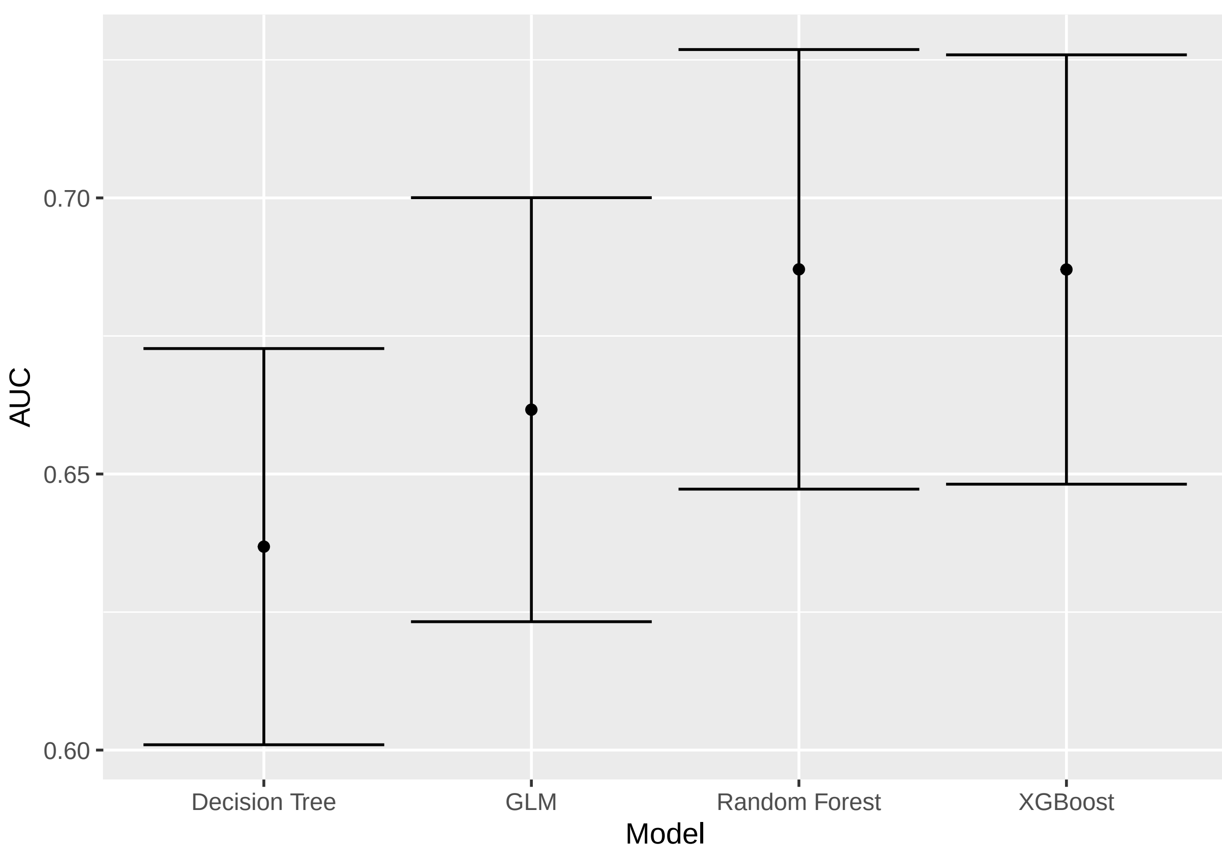



```
## Confusion Matrix and Statistics
##
##
## test_predictions_class    0    1
##                      0 4540  190
##                      1    0    0
##
##                Accuracy : 0.9598
##                  95% CI : (0.9538, 0.9652)
##     No Information Rate : 0.9598
##     P-Value [Acc > NIR] : 0.5193
##
##                   Kappa : 0
```

```
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9598
##          Neg Pred Value :    NaN
##              Prevalence : 0.9598
##          Detection Rate : 0.9598
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

Models Comparison

```
df_auc <- tibble::tribble(
  ~Model, ~`AUC`, ~`Lower Limit`, ~`Upper Limit`,
  'XGBoost', as.numeric(xgboost_auc$auc), xgboost_auc$ci[1], xgboost_auc$ci[3],
  'GLM', as.numeric(glm_auc$auc), glm_auc$ci[1], glm_auc$ci[3],
  'Decision Tree', as.numeric(tree_auc$auc), tree_auc$ci[1], tree_auc$ci[3],
  'Random Forest', as.numeric(rf_auc$auc), rf_auc$ci[1], rf_auc$ci[3]
) %>%
  mutate(Target = outcome_column)

df_auc %>%
  ggplot(aes(x = Model, y = AUC, ymin = `Lower Limit`, ymax = `Upper Limit`)) +
    geom_point() +
    geom_errorbar()
```




```
saveRDS(df_auc, sprintf("../EDA/auxiliar/performance/%s_auc_result.RData", outcome_column))
```