Model Selection

Eduardo Yuki Yada

Imports

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

Minutes to run: 0

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
```

Minutes to run: 0.001

Filtering eligible pacients

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

df %>% dim
```

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

Minutes to run: 0.007

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))

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

gluedown::md_order(features, seq = TRUE, pad = TRUE)
```

```
## 01. sex
## 02. age
## 03. race
## 04. education_level
## 05. patient_state
## 06. underlying_heart_disease
## 07. heart_disease
## 08. nyha_basal
## 09. prior_mi
## 10. heart_failure
## 11. af
## 12. cardiac_arrest
## 13. transplant
## 14. valvopathy
## 15. endocardites
## 16. diabetes
## 17. renal_failure
## 18. hemodialysis
## 19. copd
## 20. comorbidities_count
## 21. procedure_type_1
## 22. reop_type_1
## 23. procedure_type_new
## 24. cied_final_1
## 25. cied_final_group_1
## 26. admission_t0
## 27. admission_pre_t0_180d
## 28. year_adm_t0
## 29. icu_t0
## 30. dialysis_t0
## 31. disch_outcomes_t0
## 32. admission_t0_emergency
## 33. aco
## 34. antiarritmico
## 35. betabloqueador
## 36. ieca_bra
## 37. dva
## 38. digoxina
## 39. estatina
## 40. diuretico
## 41. vasodilatador
## 42. insuf_cardiaca
## 43. espironolactona
## 44. bloq_calcio
## 45. antiplaquetario_ev
## 46. insulina
```

```
## 47. anticonvulsivante
## 48. psicofarmacos
## 49. antifungico
## 50. antiviral
## 51. antiretroviral
## 52. classe_meds_qtde
## 53. meds_cardiovasc_qtde
## 54. meds_antimicrobianos
## 55. vni
## 56. cec
## 57. transplante_cardiaco
## 58. outros_proced_cirurgicos
## 59. icp
## 60. intervencao_cv
## 61. angioplastia
## 62. cateterismo
## 63. eletrofisiologia
## 64. cateter_venoso_central
## 65. proced_invasivos_qtde
## 66. cve_desf
## 67. transfusao
## 68. interconsulta
## 69. equipe_multiprof
## 70. ecg
## 71. holter
## 72. teste_esforco
## 73. espiro_ergoespiro
## 74. tilt_teste
## 75. metodos_graficos_qtde
## 76. laboratorio
## 77. cultura
## 78. analises_clinicas_qtde
## 79. citologia
## 80. biopsia
## 81. histopatologia_qtde
## 82. angio_rm
## 83. angio_tc
## 84. arteriografia
## 85. cintilografia
## 86. ecocardiograma
## 87. endoscopia
## 88. flebografia
## 89. pet_ct
## 90. ultrassom
## 91. tomografia
## 92. radiografia
## 93. ressonancia
## 94. exames_imagem_qtde
## 95. bic
## 96. mpp
```

Minutes to run: 0

Train test split (70%/30%)

```
set.seed(42)

df[columns_list$outcome_columns] <- lapply(df[columns_list$outcome_columns], factor)
df <- mutate(df, across(where(is.character), as.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)
```

Minutes to run: 0.005

Minutes to run: 0

Global parameters

```
k = 4 # Number of folds for cross validation
grid_size = 10 # Number of parameter combination to tune on each model

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

Minutes to run: 0

Functions

```
validation = function(model_fit, new_data, plot=TRUE) {
  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,
    direction = "<",
    levels = c(0, 1),
    smoothed = TRUE,
    ci = TRUE,
    ci.alpha = 0.9,
    stratified = FALSE,
    plot = plot,
    auc.polygon = TRUE,
    max.auc.polygon = TRUE,
    grid = TRUE,
    print.auc = TRUE,
    show.thres = TRUE
  )

  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]])

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

    confusionMatrix(conf_matrix) %>% print
```

```
  }

  return(pROC_obj)
}
```

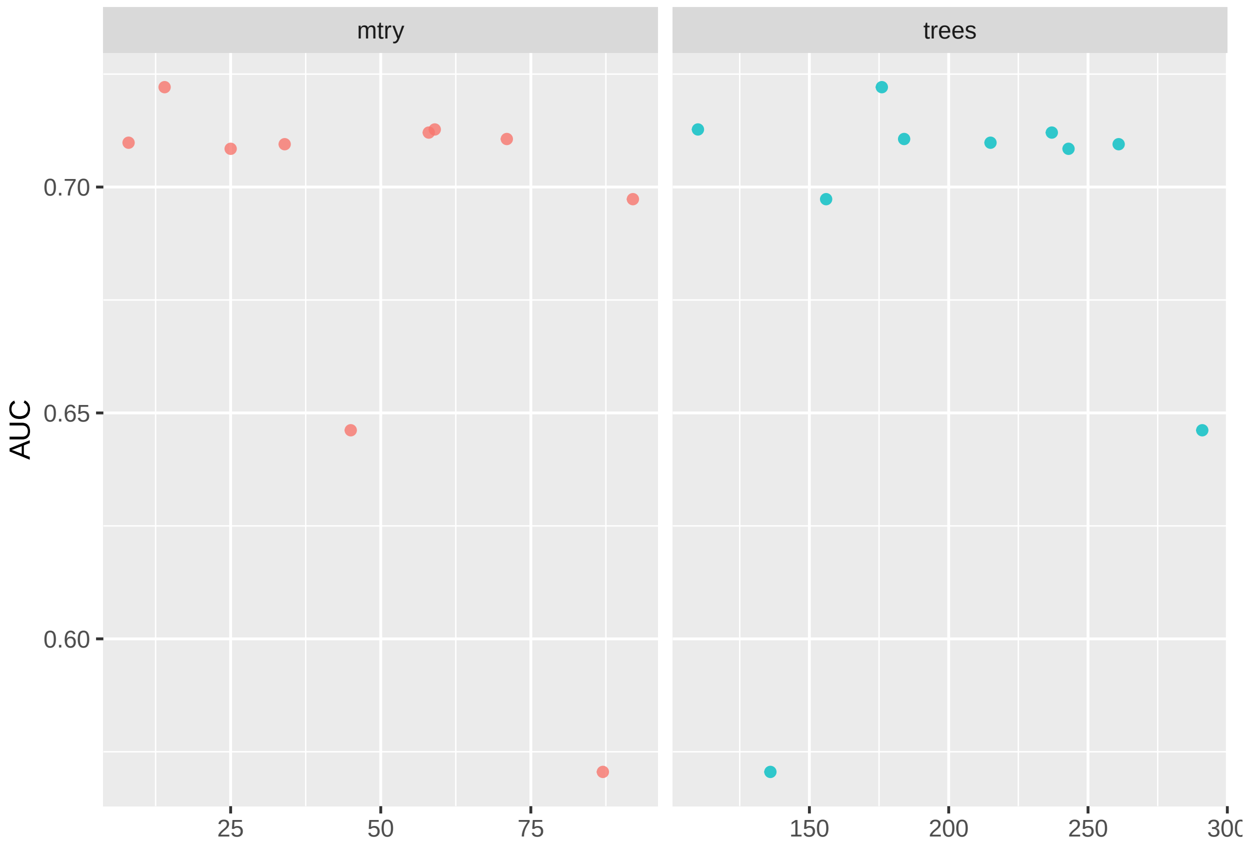
Minutes to run: 0

Boosted Tree (XGBoost)

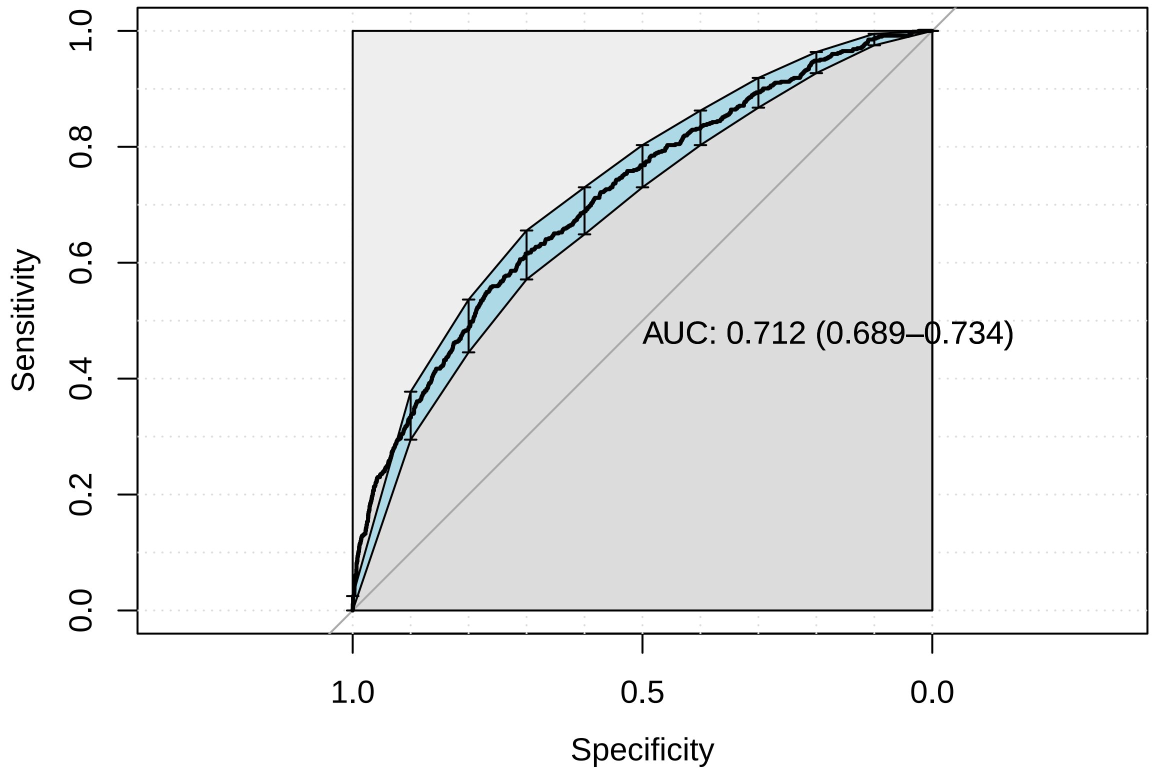
```
xgboost_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_other(all_nominal_predictors(), threshold = 0.05, other=".merged") %>%
  step_dummy(all_nominal_predictors(), one_hot = TRUE) %>%
  step_zv(all_predictors())

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

xgboost_grid <- grid_latin_hypercube(
  finalize(mtry(), df_train),
  dials::trees(range = c(100L, 300L)),
  min_n(),
  tree_depth(),
  learn_rate(),
  loss_reduction(),
  size = grid_size
)

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 trees min_n tree_depth learn_rate loss_re~1 .metric .esti~2  mean     n std_err .config
##   <int> <int> <int>      <int>      <dbl>     <dbl> <chr>   <chr>   <dbl> <int>   <dbl> <chr>
## 1    14   176    27          9 0.0386       2.01e-6 roc_auc binary  0.722     4 0.00270 Prepro~
## 2    59   110    12          6 0.000376     3.01e-2 roc_auc binary  0.713     4 0.00656 Prepro~
## 3    58   237    32         12 0.00000516   6.37e-1 roc_auc binary  0.712     4 0.00689 Prepro~
## 4    71   184    18          5 0.00000269   4.36e-3 roc_auc binary  0.711     4 0.00634 Prepro~
## 5     8   215    22         11 0.0000451    1.00e-4 roc_auc binary  0.710     4 0.00692 Prepro~
## # ... with abbreviated variable names 1: loss_reduction, 2: .estimator
```

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

xgboost_tune %>%
  collect_metrics() %>%
  filter(.metric == "roc_auc") %>%
  select(mean, mtry:trees) %>%
  pivot_longer(mtry:trees,
               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")
```

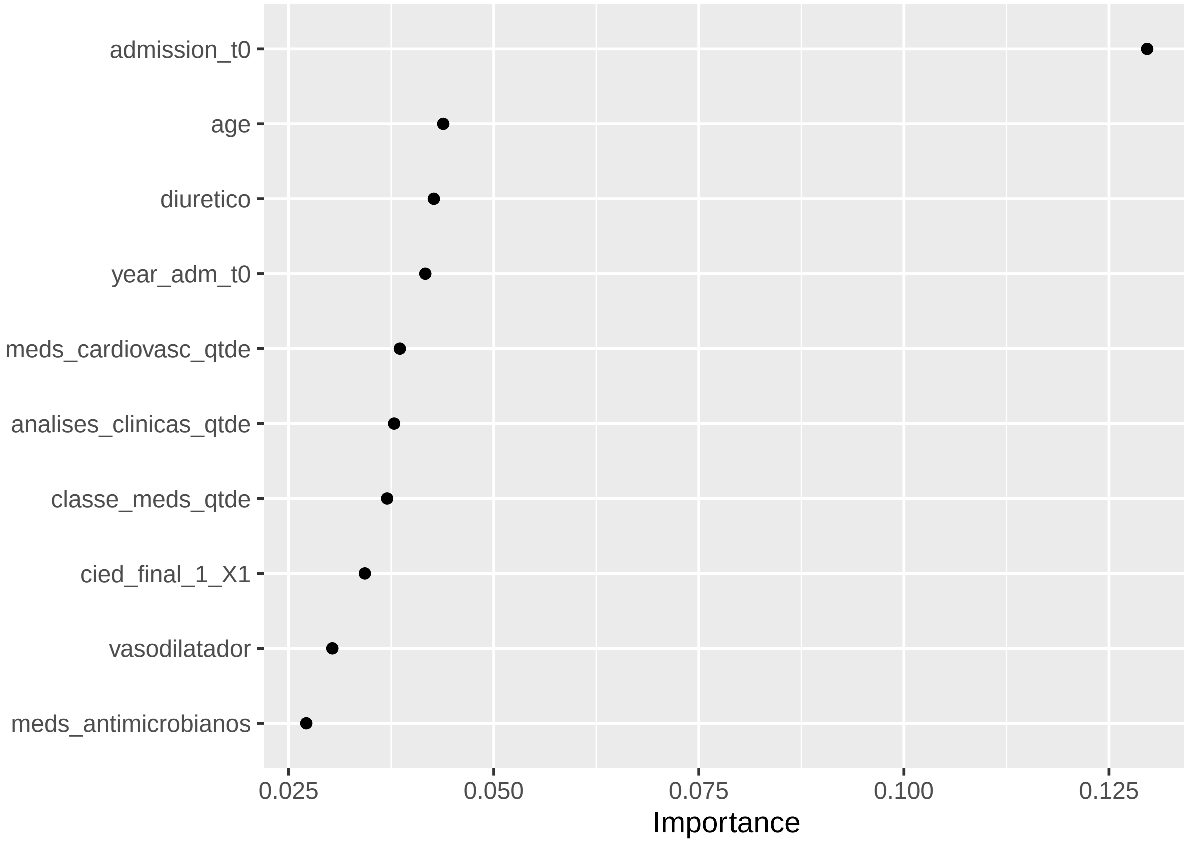



```r
final_xgboost_workflow <-
  xgboost_workflow %>%
  finalize_workflow(best_xgboost)

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 4114  580
##                      1   13   24
##
##                Accuracy : 0.8747
##                  95% CI : (0.8649, 0.884)
##     No Information Rate : 0.8723
##     P-Value [Acc > NIR] : 0.3252
##
##                   Kappa : 0.061
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99685
##             Specificity : 0.03974
##          Pos Pred Value : 0.87644
##          Neg Pred Value : 0.64865
##              Prevalence : 0.87233
##          Detection Rate : 0.86958
##    Detection Prevalence : 0.99218
##       Balanced Accuracy : 0.51829
##
##        'Positive' Class : 0
##
```

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




```
xgboost_parameters <- xgboost_tune %>%
  show_best("roc_auc", n=1) %>%
  select(trees, mtry, min_n, tree_depth, learn_rate, loss_reduction) %>%
  as.list

saveRDS(
  xgboost_parameters,
  file = sprintf(
    "../EDA/auxiliar/hyperparameters/model_selection/xgboost_parameters_%s.rds",
    outcome_column
  )
)
```

Minutes to run: 3.835

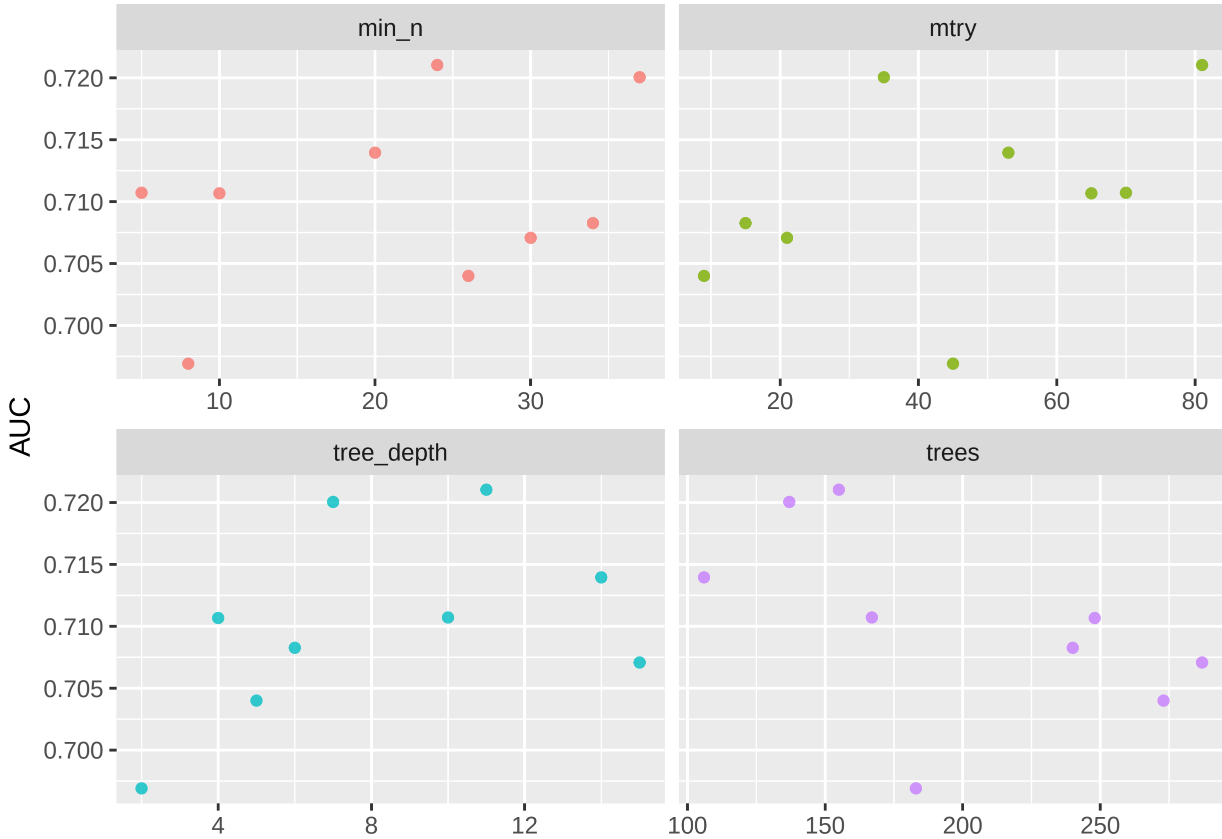
Boosted Tree (LightGBM)

```
lightgbm_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_other(all_nominal_predictors(), threshold = 0.05, other=".merged") %>%
  step_impute_mean(all_numeric_predictors()) %>%
  step_zv(all_predictors())

lightgbm_spec <- boost_tree(
  mtry = tune(),
  trees = tune(),
  min_n = tune(),
  tree_depth = tune(),
  learn_rate = tune(),
  loss_reduction = tune(),
  sample_size = 1
```

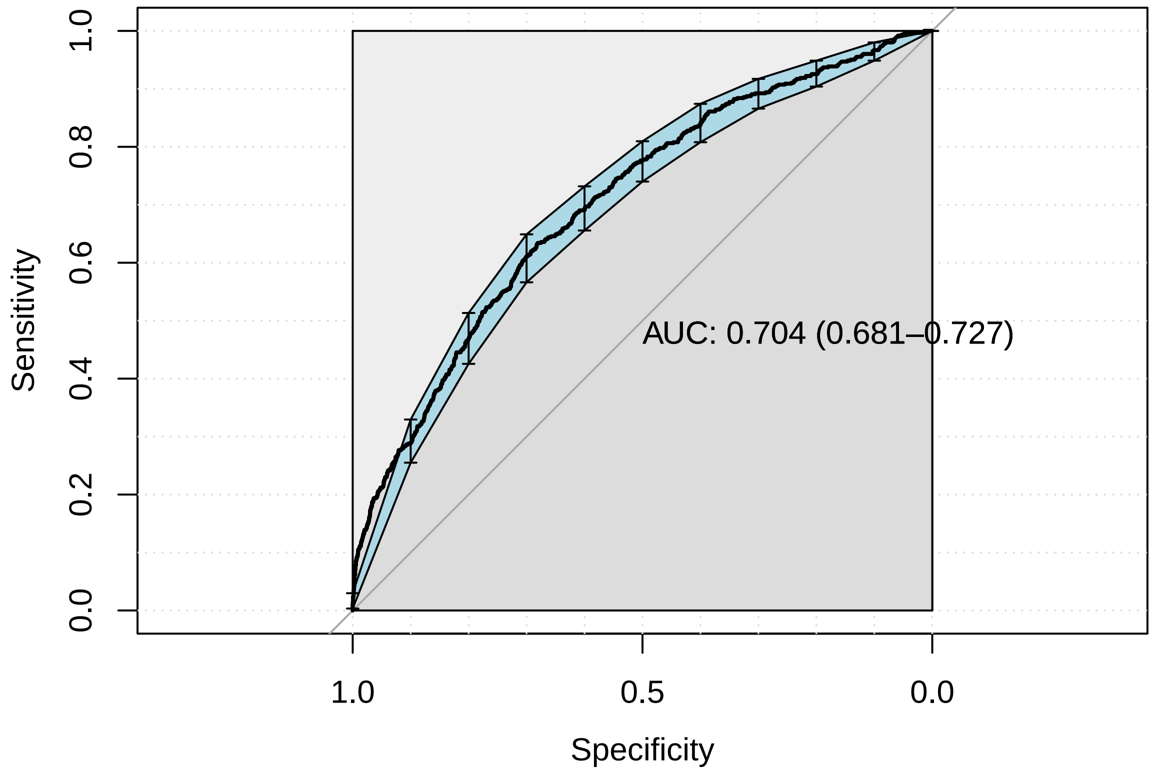
```r
) %>%
  set_engine("lightgbm") %>%
  set_mode("classification")

lightgbm_grid <- grid_latin_hypercube(
  finalize(mtry(), df_train),
  dials::trees(range = c(100L, 300L)),
  min_n(),
  tree_depth(),
  learn_rate(),
  loss_reduction(),
  size = grid_size
)

lightgbm_workflow <-
  workflow() %>%
  add_recipe(lightgbm_recipe) %>%
  add_model(lightgbm_spec)

lightgbm_tune <-
  lightgbm_workflow %>%
  tune_grid(resamples = df_folds,
            grid = lightgbm_grid)

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

```
## # A tibble: 5 x 12
##    mtry trees min_n tree_depth learn_rate loss_re~1 .metric .esti~2  mean     n std_err .config
##   <int> <int> <int>      <int>      <dbl>     <dbl> <chr>   <chr>   <dbl> <int>   <dbl> <chr>
## 1    81   155    24         11   1.22e- 2   2.72e-9 roc_auc binary  0.721     4 0.00388 Prepro~
## 2    35   137    37          7   1.97e- 2   1.77e-4 roc_auc binary  0.720     4 0.00287 Prepro~
## 3    53   106    20         14   1.01e- 6   2.05e-1 roc_auc binary  0.714     4 0.00674 Prepro~
## 4    70   167     5         10   3.98e- 9   2.36e-5 roc_auc binary  0.711     4 0.00733 Prepro~
## 5    65   248    10          4   5.73e-10   2.66e-7 roc_auc binary  0.711     4 0.00650 Prepro~
## # ... with abbreviated variable names 1: loss_reduction, 2: .estimator
```

```r
best_lightgbm <- lightgbm_tune %>%
  select_best("roc_auc")

lightgbm_tune %>%
  collect_metrics() %>%
  filter(.metric == "roc_auc") %>%
  select(mean, mtry:tree_depth) %>%
  pivot_longer(mtry:tree_depth,
               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_lightgbm_workflow <-
  lightgbm_workflow %>%
  finalize_workflow(best_lightgbm)

last_lightgbm_fit <-
  final_lightgbm_workflow %>%
  last_fit(df_split)

final_lightgbm_fit <- extract_workflow(last_lightgbm_fit)

lightgbm_auc <- validation(final_lightgbm_fit, df_test)
```




```
## Confusion Matrix and Statistics
##
##
## test_predictions_class    0    1
##                      0 4125  592
##                      1    2   12
##
##                Accuracy : 0.8744
##                  95% CI : (0.8647, 0.8838)
##     No Information Rate : 0.8723
##     P-Value [Acc > NIR] : 0.3411
##
##                   Kappa : 0.0332
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99952
##             Specificity : 0.01987
##          Pos Pred Value : 0.87450
##          Neg Pred Value : 0.85714
##              Prevalence : 0.87233
##          Detection Rate : 0.87191
##    Detection Prevalence : 0.99704
##       Balanced Accuracy : 0.50969
##
##        'Positive' Class : 0
##
```

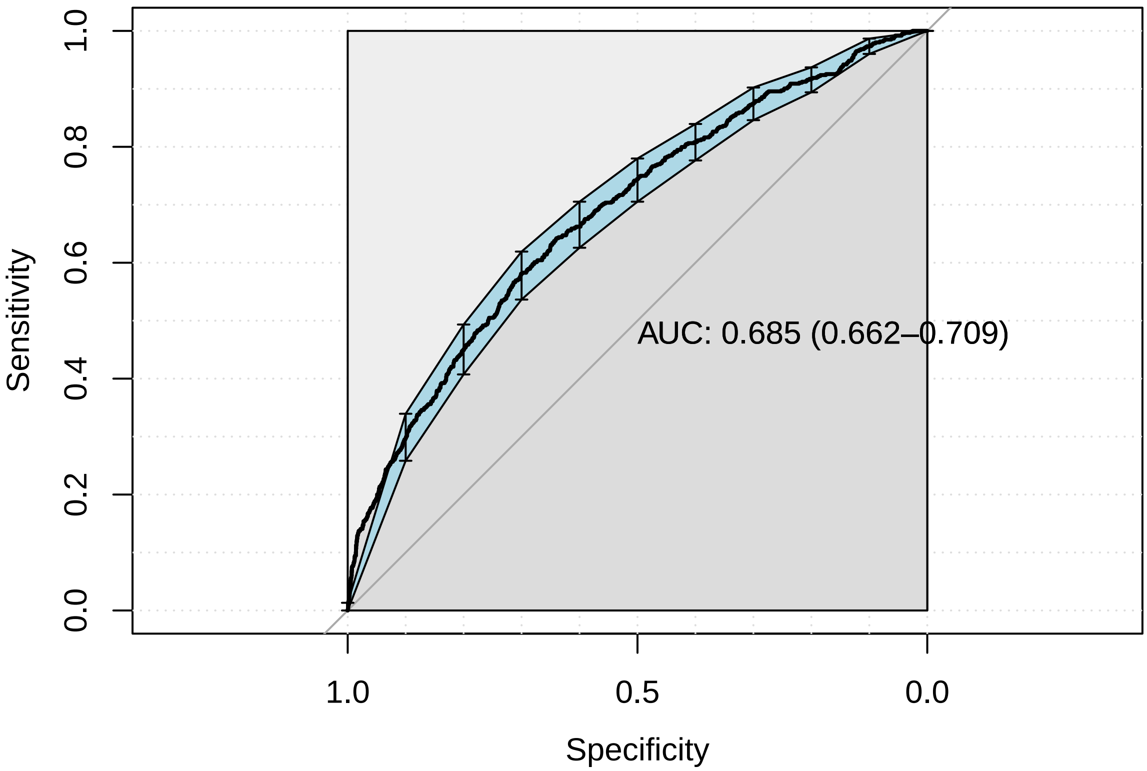
```
lightgbm_parameters <- lightgbm_tune %>%
  show_best("roc_auc", n=1) %>%
  select(trees, mtry, min_n, tree_depth, learn_rate, loss_reduction) %>%
  as.list

saveRDS(
```

```
  lightgbm_parameters,
  file = sprintf(
    "../EDA/auxiliar/hyperparameters/model_selection/lightgbm_parameters_%s.rds",
    outcome_column
  )
)
```

Minutes to run: 1.218

GLM

```
glmnet_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_other(all_nominal_predictors(), threshold = 0.05, other=".merged") %>%
  step_dummy(all_nominal_predictors(), one_hot = TRUE) %>%
  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_auc = validation(glm_fit, df_test)
```

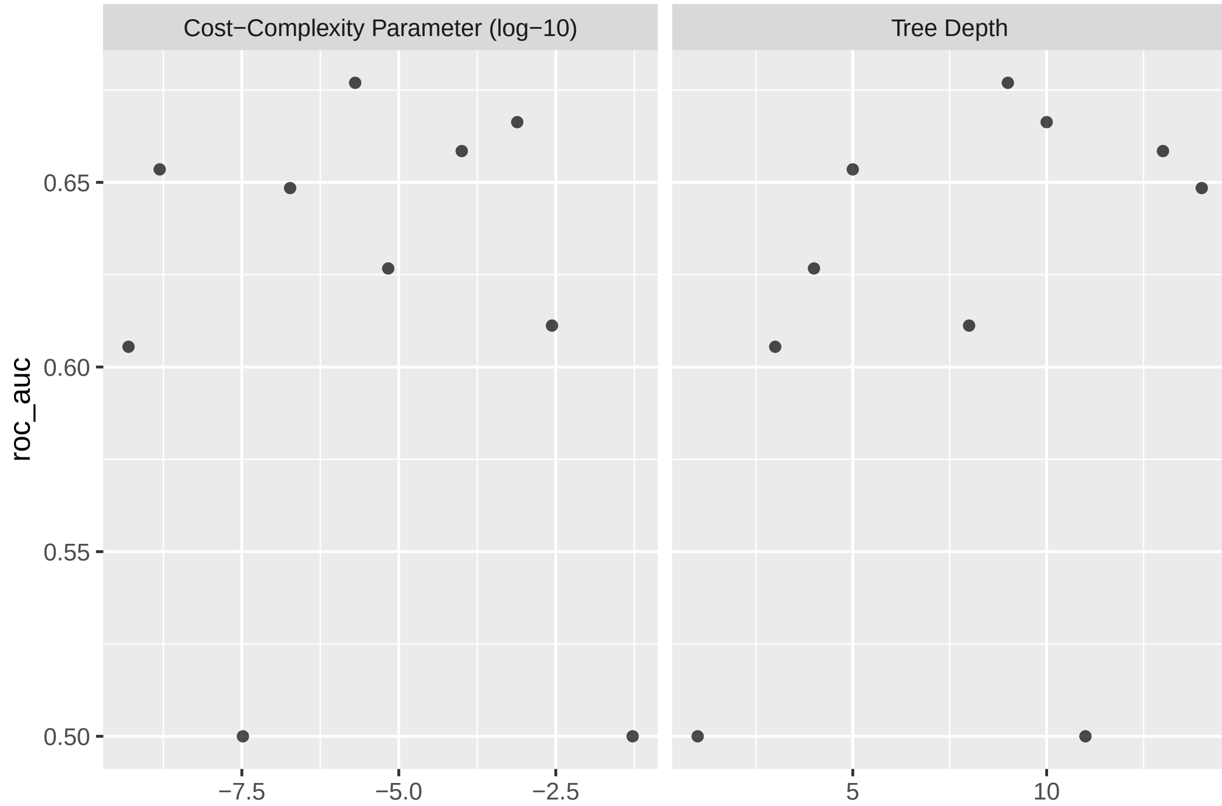



```
## Confusion Matrix and Statistics
##
##
## test_predictions_class    0    1
##                      0 4105  571
##                      1   22   33
##
##                Accuracy : 0.8747
##                  95% CI : (0.8649, 0.884)
##     No Information Rate : 0.8723
##     P-Value [Acc > NIR] : 0.3252
##
##                   Kappa : 0.0806
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99467
##             Specificity : 0.05464
##          Pos Pred Value : 0.87789
##          Neg Pred Value : 0.60000
##              Prevalence : 0.87233
##          Detection Rate : 0.86768
##    Detection Prevalence : 0.98837
##       Balanced Accuracy : 0.52465
##
##        'Positive' Class : 0
##
```

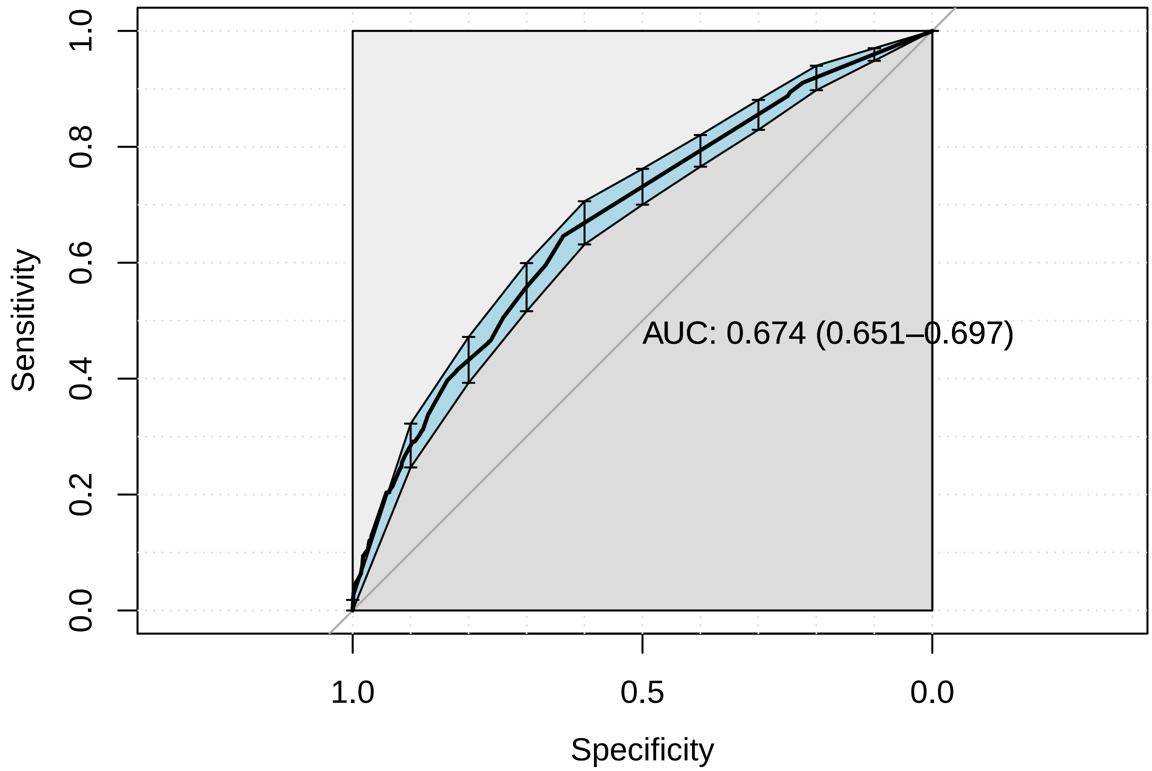
Minutes to run: 0.168

Decision Tree

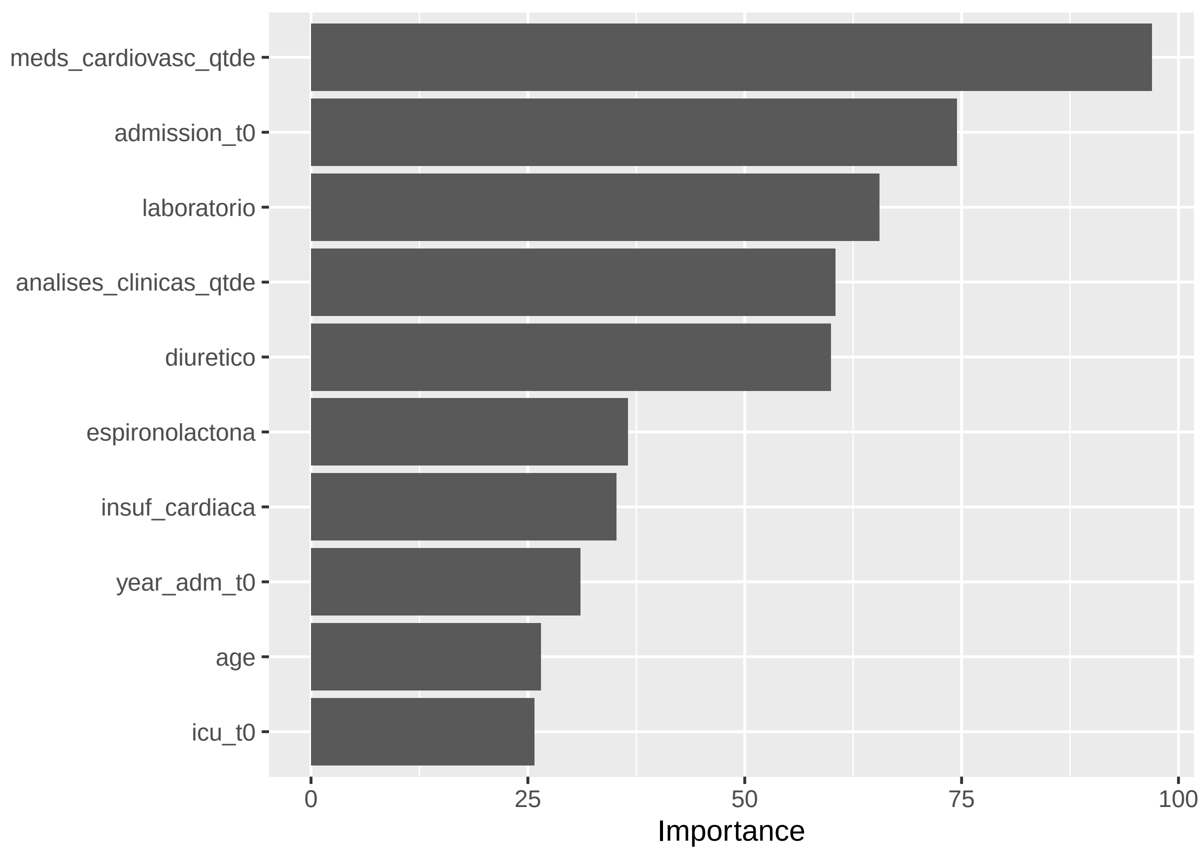
```
tree_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_other(all_nominal_predictors(), threshold = 0.05, other=".merged") %>%
  step_dummy(all_nominal_predictors(), one_hot = TRUE) %>%
  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 = grid_size)

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")
```




```r
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)
```



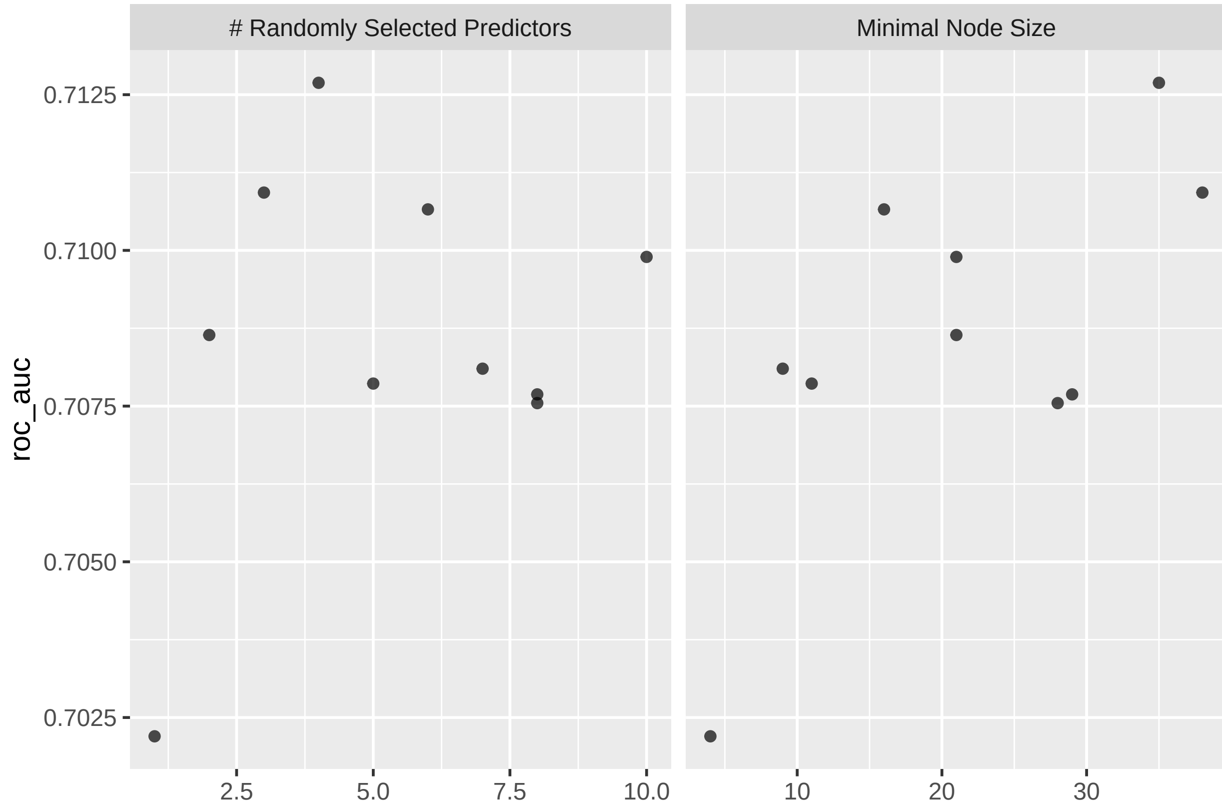

```r
if (tree_auc$auc > 0.55){
  final_tree_fit %>%
    extract_fit_parsnip() %>%
    vip()
}
```



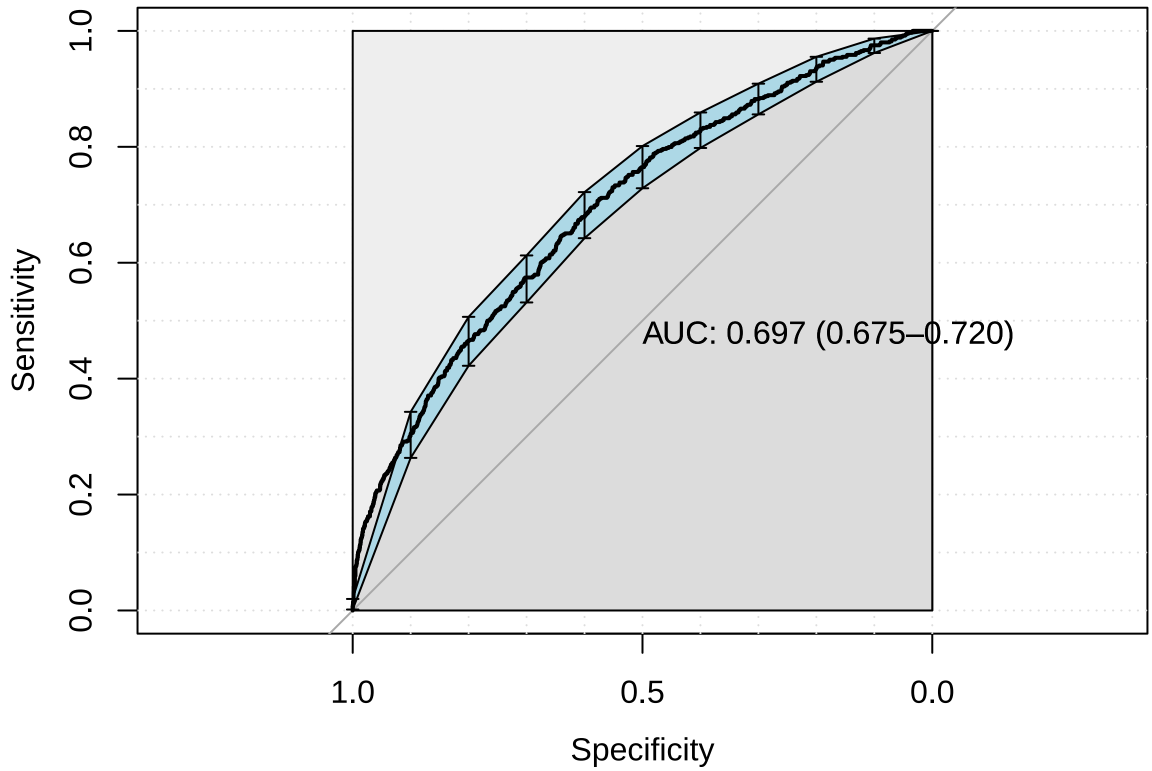
Minutes to run: 1.221

Random Forest

```
rf_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_other(all_nominal_predictors(), threshold = 0.05, other=".merged") %>%
  step_dummy(all_nominal_predictors(), one_hot = TRUE) %>%
  step_zv(all_predictors()) %>%
  step_impute_mean(all_numeric_predictors())

rf_spec <-
  rand_forest(mtry = tune(),
              trees = 100,
              min_n = tune()) %>%
  set_mode("classification") %>%
  set_engine("ranger")

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

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()

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




```r
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)
```



Minutes to run: 1.179

KNN

```
# knn_recipe <-
#   recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
#   step_novel(all_nominal_predictors()) %>%
#   step_unknown(all_nominal_predictors()) %>%
#   step_other(all_nominal_predictors(), threshold = 0.05, other=".merged") %>%
#   step_dummy(all_nominal_predictors(), one_hot = TRUE) %>%
#   step_zv(all_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 = 5)
#
# knn_workflow <-
#   workflow() %>%
#   add_recipe(knn_recipe) %>%
#   add_model(knn_spec)
#
# 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)
```

Minutes to run: 0

SVM

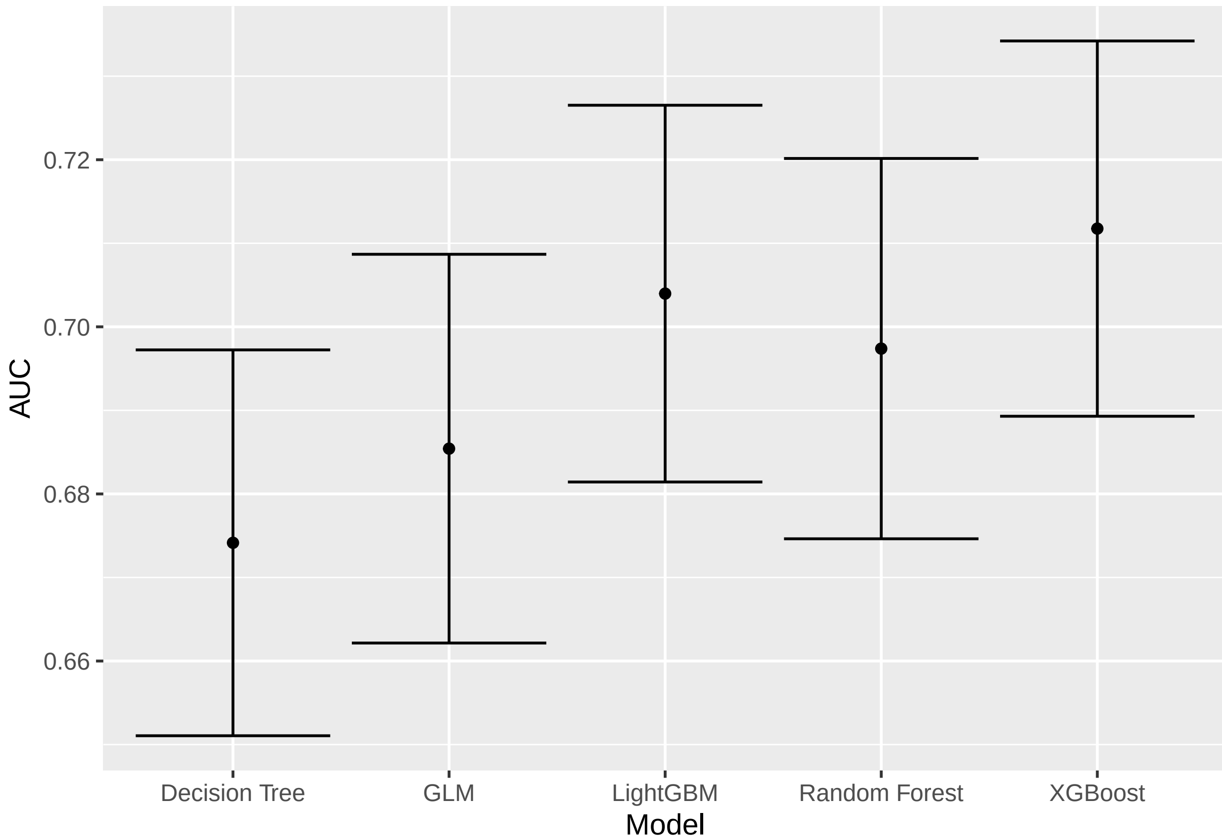
```
# svm_recipe <-
#   recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
#   step_novel(all_nominal_predictors()) %>%
#   step_unknown(all_nominal_predictors()) %>%
#   step_other(all_nominal_predictors(), threshold = 0.05, other=".merged") %>%
#   step_dummy(all_nominal_predictors(), one_hot = TRUE) %>%
#   step_zv(all_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 = grid_size)
#
# svm_workflow <-
#   workflow() %>%
#   add_recipe(svm_recipe) %>%
#   add_model(svm_spec)
#
# svm_tune <-
#   svm_workflow %>%
#   tune_grid(resamples = df_folds,
#             grid = 5)
#
# svm_tune %>%
#   collect_metrics()
#
```

```r
# autoplot(svm_tune, metric = "roc_auc")
#
# svm_tune %>%
#   show_best("roc_auc")
#
# 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)
```

Minutes to run: 0

Models Comparison

```r
df_auc <- tibble::tribble(
  ~Model, ~`AUC`, ~`Lower Limit`, ~`Upper Limit`,
  'XGBoost', as.numeric(xgboost_auc$auc), xgboost_auc$ci[1], xgboost_auc$ci[3],
  'LightGBM', as.numeric(lightgbm_auc$auc), lightgbm_auc$ci[1], lightgbm_auc$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))
```

Minutes to run: 0.003