Model Selection - readmission_1year

Eduardo Yuki Yada

Imports

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

Minutes to run: 0

Loading data

```r
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

df <- mutate(df, across(where(is.character), as.factor))
```

Minutes to run: 0.006

Eligible features

```r
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_t0', # com admission_pre_t0_count
                       '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_pre_t0_count
## 27. admission_pre_t0_180d
## 28. year_adm_t0
## 29. icu_t0
## 30. dialysis_t0
## 31. admission_t0_emergency
## 32. aco
## 33. antiarritmico
## 34. betabloqueador
## 35. ieca_bra
## 36. dva
## 37. digoxina
## 38. estatina
## 39. diuretico
## 40. vasodilatador
## 41. insuf_cardiaca
## 42. espironolactona
## 43. bloq_calcio
## 44. antiplaquetario_ev
## 45. insulina
## 46. anticonvulsivante
## 47. psicofarmacos
## 48. antifungico
## 49. antiviral
## 50. antiretroviral
## 51. classe_meds_qtde
## 52. meds_cardiovasc_qtde
## 53. meds_antimicrobianos
## 54. cec
## 55. transplante_cardiaco
```

```
## 56. cir_toracica
## 57. outros_proced_cirurgicos
## 58. icp
## 59. intervencao_cv
## 60. angioplastia
## 61. cateterismo
## 62. eletrofisiologia
## 63. cateter_venoso_central
## 64. proced_invasivos_qtde
## 65. cve_desf
## 66. transfusao
## 67. interconsulta
## 68. equipe_multiprof
## 69. ecg
## 70. holter
## 71. teste_esforco
## 72. espiro_ergoespiro
## 73. tilt_teste
## 74. metodos_graficos_qtde
## 75. laboratorio
## 76. cultura
## 77. analises_clinicas_qtde
## 78. citologia
## 79. biopsia
## 80. histopatologia_qtde
## 81. angio_rm
## 82. angio_tc
## 83. aortografia
## 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)

if (outcome_column == 'readmission_30d') {
  df_split <- readRDS("./dataset/split_object.rds")
} else {
  df_split <- initial_split(df, prop = .7, strata = all_of(outcome_column))
}


df_train <- training(df_split) %>% dplyr::select(all_of(c(features, outcome_column)))
df_test <- testing(df_split) %>% dplyr::select(all_of(c(features, outcome_column)))
```

Minutes to run: 0.001

Global parameters

```r
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

```r
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)

```r
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_reduction .metric .estimator  mean     n std_err .config
##    <int> <int> <int>      <int>      <dbl>          <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     14   176    27          9     0.0386     0.00000201 roc_auc binary     0.722     4 0.00535 Preprocessor1
## 2     59   110    12          6   0.000376     0.0301     roc_auc binary     0.711     4 0.00874 Preprocessor1
## 3     71   184    18          5  0.00000269    0.00436    roc_auc binary     0.709     4 0.00809 Preprocessor1
## 4     58   237    32         12  0.00000516    0.637      roc_auc binary     0.709     4 0.00789 Preprocessor1
## 5      8   215    22         11   0.0000451    0.000100   roc_auc binary     0.708     4 0.00675 Preprocessor1
```
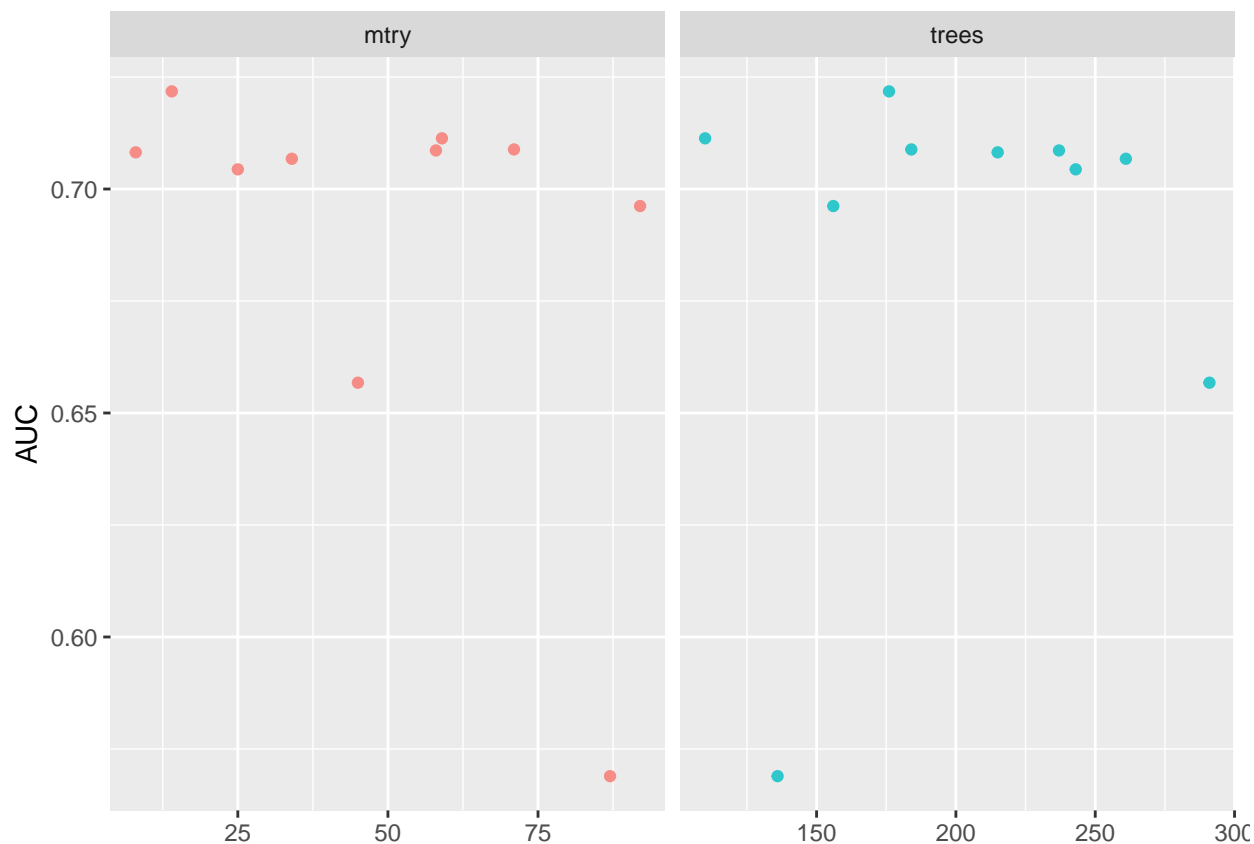
```r
best_xgboost <- xgboost_tune %>%
  select_best("roc_auc")

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")
```
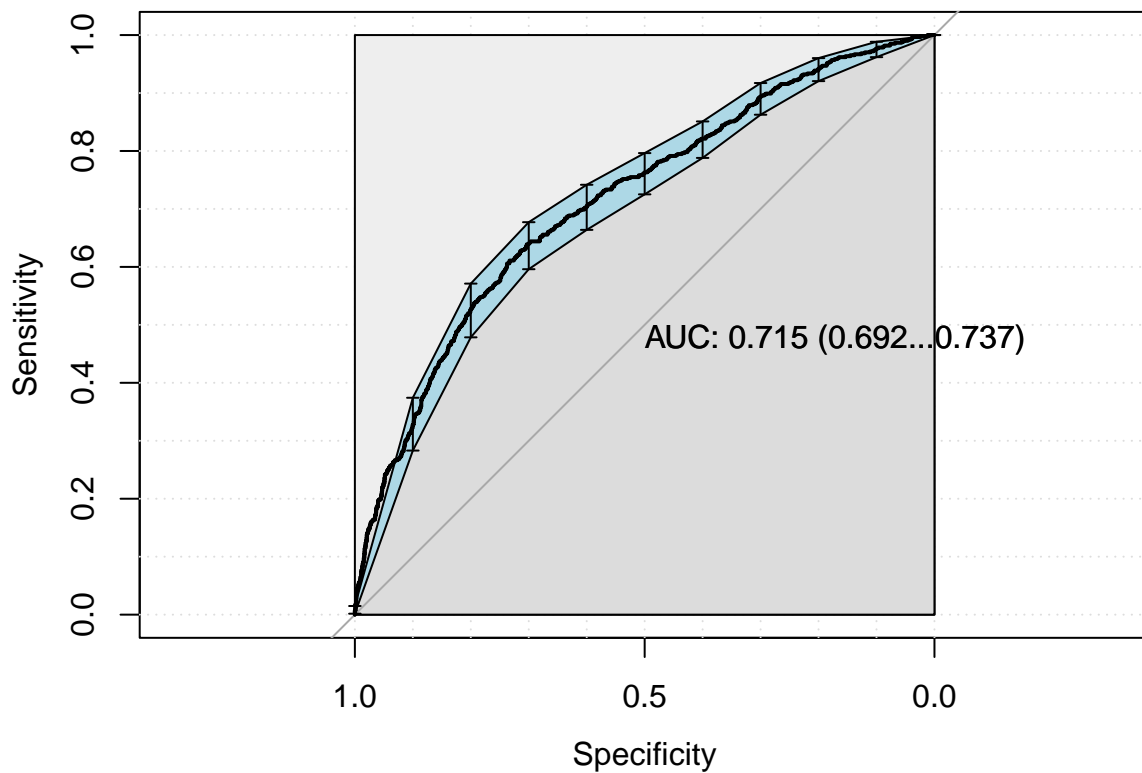


```
final_xgboost_workflow <-
  xgboost_workflow %>%
  finalize_workflow(best_xgboost)

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 4101  573
##                     1   26   31
##
##              Accuracy : 0.8734
##                95% CI : (0.8636, 0.8827)
##    No Information Rate : 0.8723
##    P-Value [Acc > NIR] : 0.4243
##
##                 Kappa : 0.0734
##
##  Mcnemar's Test P-Value : <2e-16
##
##            Sensitivity : 0.99370
##            Specificity : 0.05132
##         Pos Pred Value : 0.87741
##         Neg Pred Value : 0.54386
##             Prevalence : 0.87233
##         Detection Rate : 0.86684
##   Detection Prevalence : 0.98795
##      Balanced Accuracy : 0.52251
##
##       'Positive' Class : 0
##
```

```
final_xgboost_fit %>%
  fit(data = df_train) %>%
  extract_fit_parsnip() %>%
  vip(geom = "point")
```
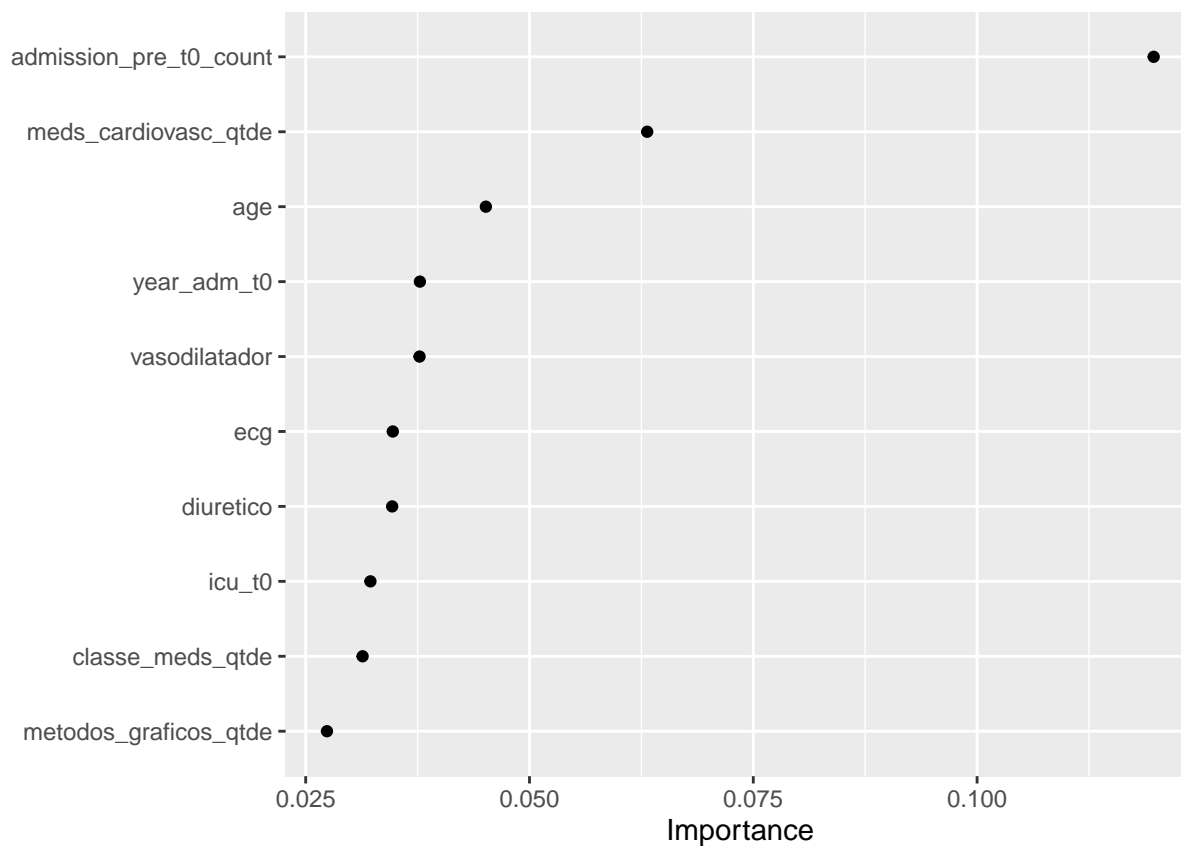
```r
xgboost_parameters <- xgboost_tune %>%
  show_best("roc_auc", n=1) %>%
  select(trees, mtry, min_n, tree_depth, learn_rate, loss_reduction) %>%
  as.list

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

Minutes to run: 4.276

Boosted Tree (LightGBM)

```r
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_reduction .metric .estimator  mean     n std_err .config
##   <int> <int> <int>      <int>      <dbl>          <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    30   143    23         15   1.01e- 2    0.000000891 roc_auc binary     0.720     4 0.00556 Preprocessor1
## 2    13   237    36          2   7.16e- 2    0.00239     roc_auc binary     0.718     4 0.00459 Preprocessor1
## 3    49   241    14          7   4.02e- 8    0.000195    roc_auc binary     0.713     4 0.00748 Preprocessor1
## 4    46   265    36          4   1.73e- 4    0.00000000155 roc_auc binary   0.710     4 0.00728 Preprocessor1
## 5    69   165    12         12   1.67e-10    0.0000249   roc_auc binary     0.708     4 0.00695 Preprocessor1
```

```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")
```
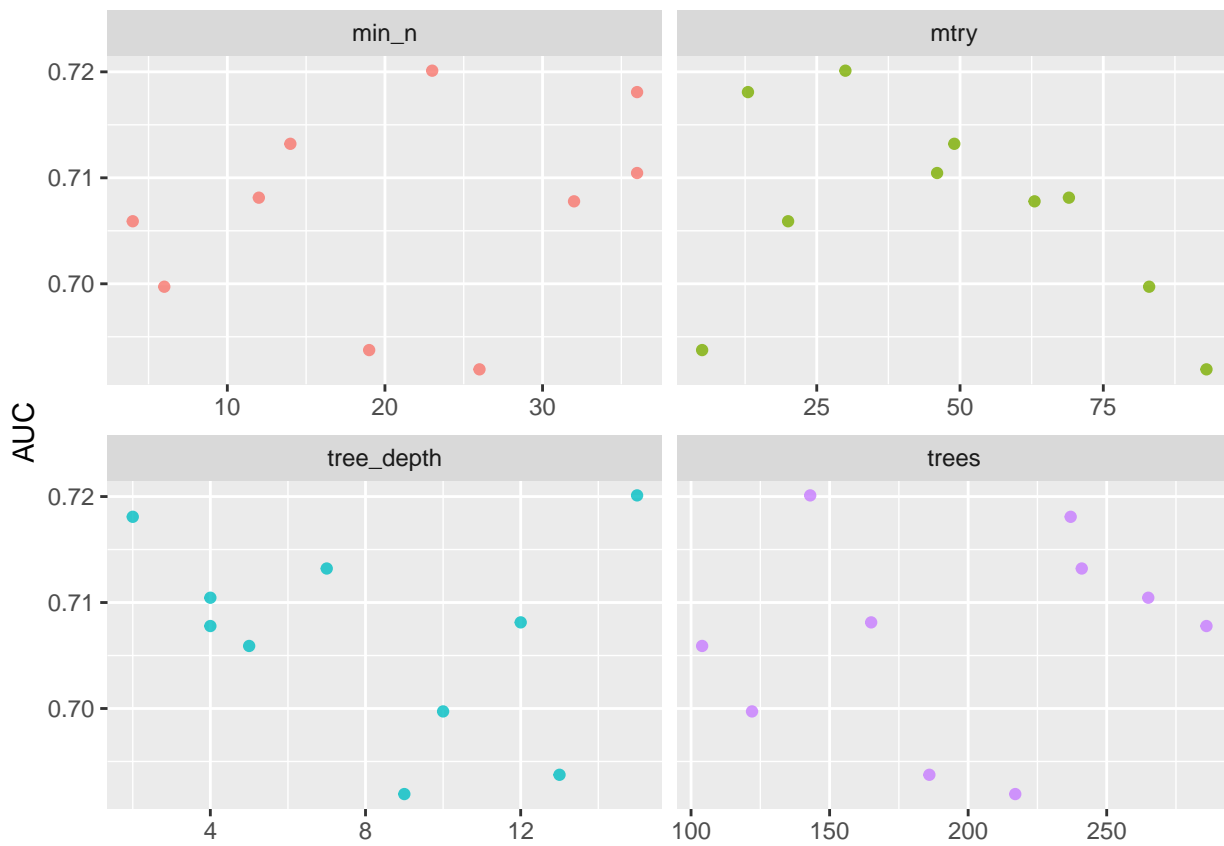
```
final_lightgbm_workflow <-
  lightgbm_workflow %>%
  finalize_workflow(best_lightgbm)

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)
```
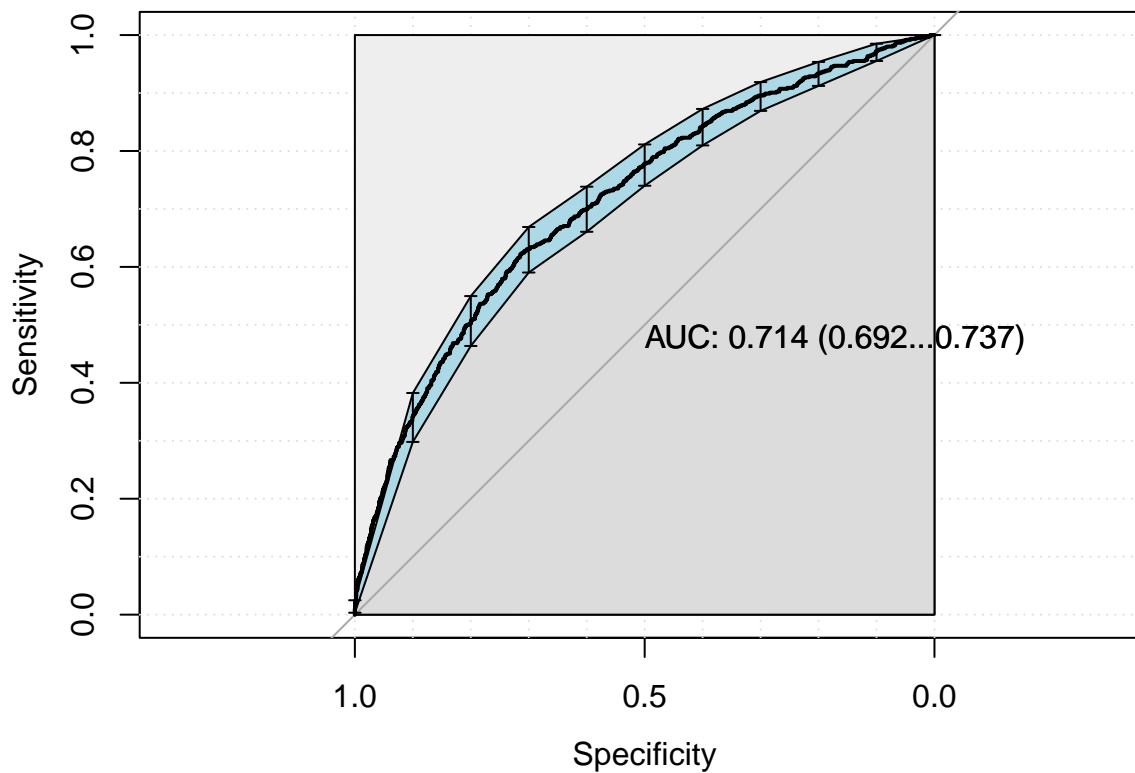
```
##   |
## Confusion Matrix and Statistics
##
##
## test_predictions_class    0    1
##                     0 4127  598
##                     1    0    6
##
##               Accuracy : 0.8736
##                 95% CI : (0.8638, 0.8829)
##    No Information Rate : 0.8723
##    P-Value [Acc > NIR] : 0.4073
##
##                  Kappa : 0.0172
##
##  Mcnemar's Test P-Value : <2e-16
##
##            Sensitivity : 1.000000
##            Specificity : 0.009934
##         Pos Pred Value : 0.873439
##         Neg Pred Value : 1.000000
##             Prevalence : 0.872331
##         Detection Rate : 0.872331
##   Detection Prevalence : 0.998732
##       Balanced Accuracy : 0.504967
##
##        'Positive' Class : 0
##
```

```r
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(
```

```
    "./auxiliar/model_selection/hyperparameters/lightgbm_%s.rds",
    outcome_column
  )
)
```

Minutes to run: 0.605

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)
```
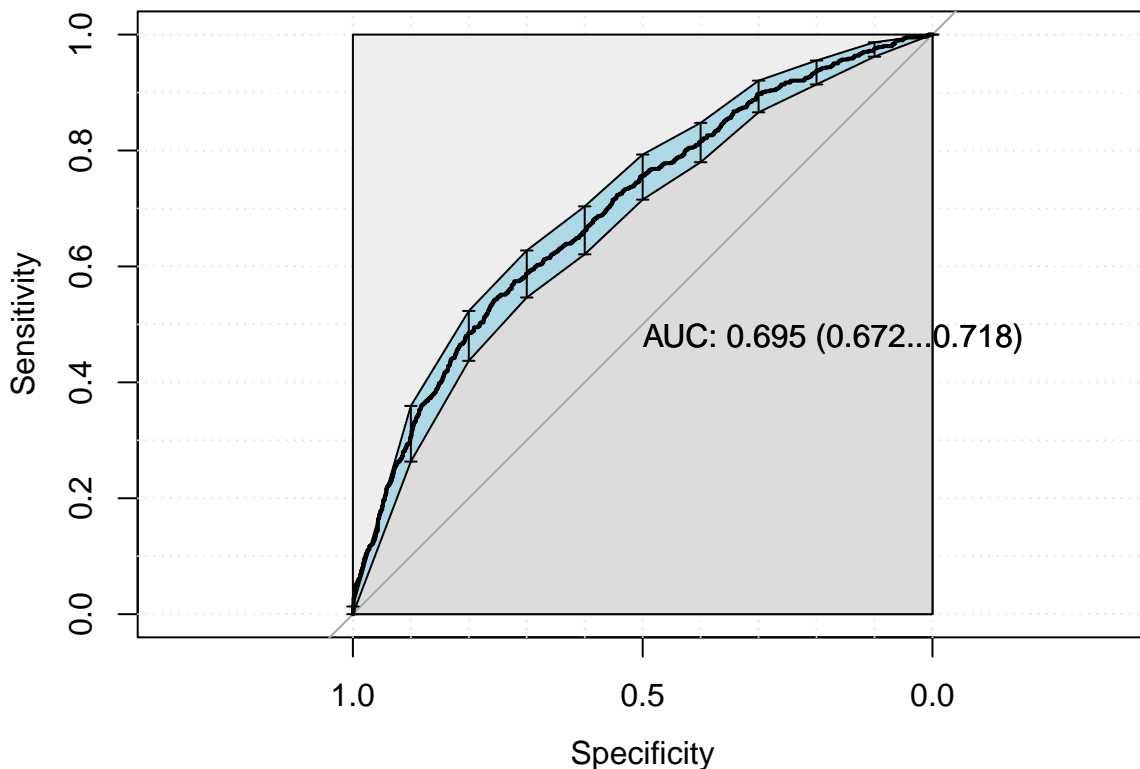


```
##    |
## Confusion Matrix and Statistics
##
```

```
## 
## test_predictions_class    0     1
##                     0 4103   574
##                     1   24    30
## 
##                Accuracy : 0.8736
##                  95% CI : (0.8638, 0.8829)
##     No Information Rate : 0.8723
##     P-Value [Acc > NIR] : 0.4073
## 
##                   Kappa : 0.0717
## 
##  Mcnemar's Test P-Value : <2e-16
## 
##             Sensitivity : 0.99418
##             Specificity : 0.04967
##          Pos Pred Value : 0.87727
##          Neg Pred Value : 0.55556
##              Prevalence : 0.87233
##          Detection Rate : 0.86726
##    Detection Prevalence : 0.98859
##       Balanced Accuracy : 0.52193
## 
##        'Positive' Class : 0
## 
```

Minutes to run: 0.096

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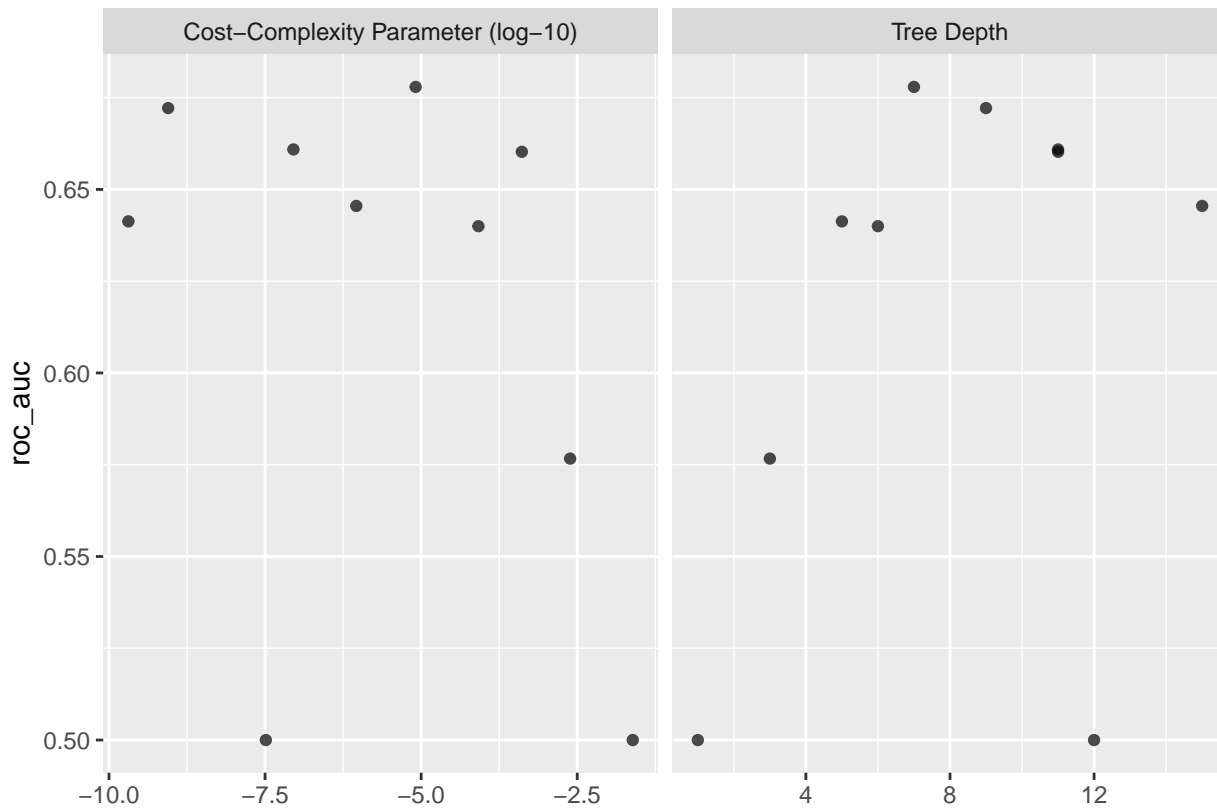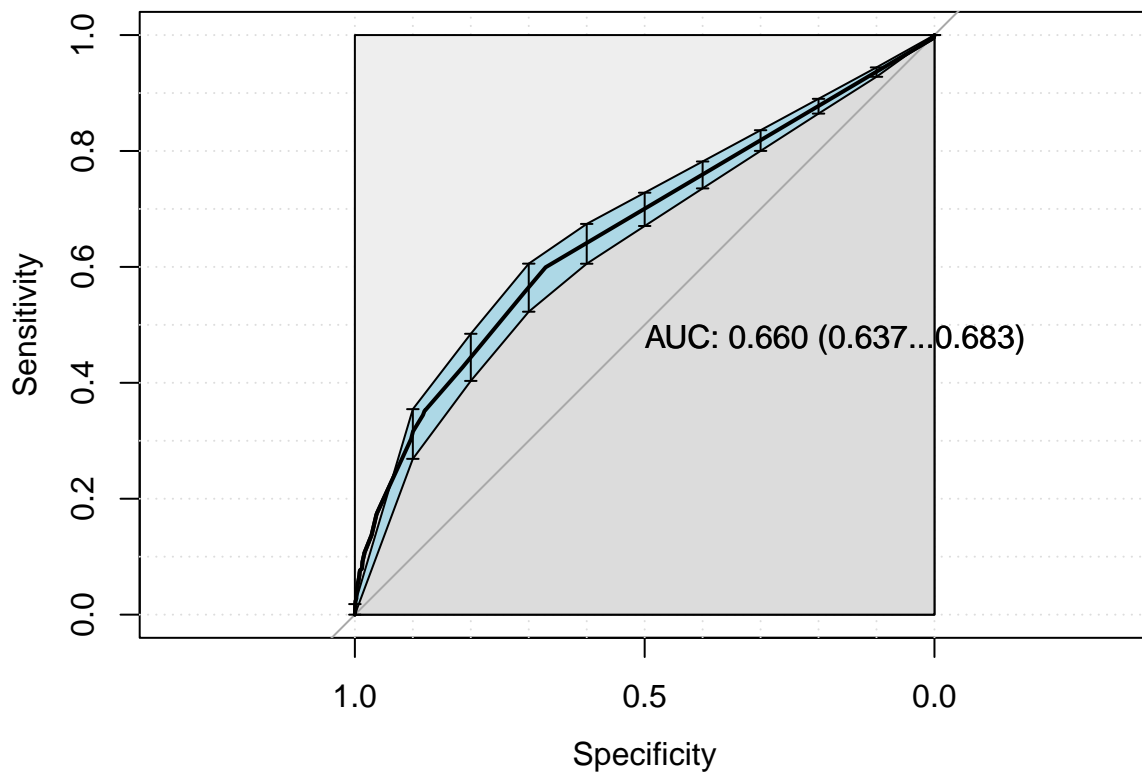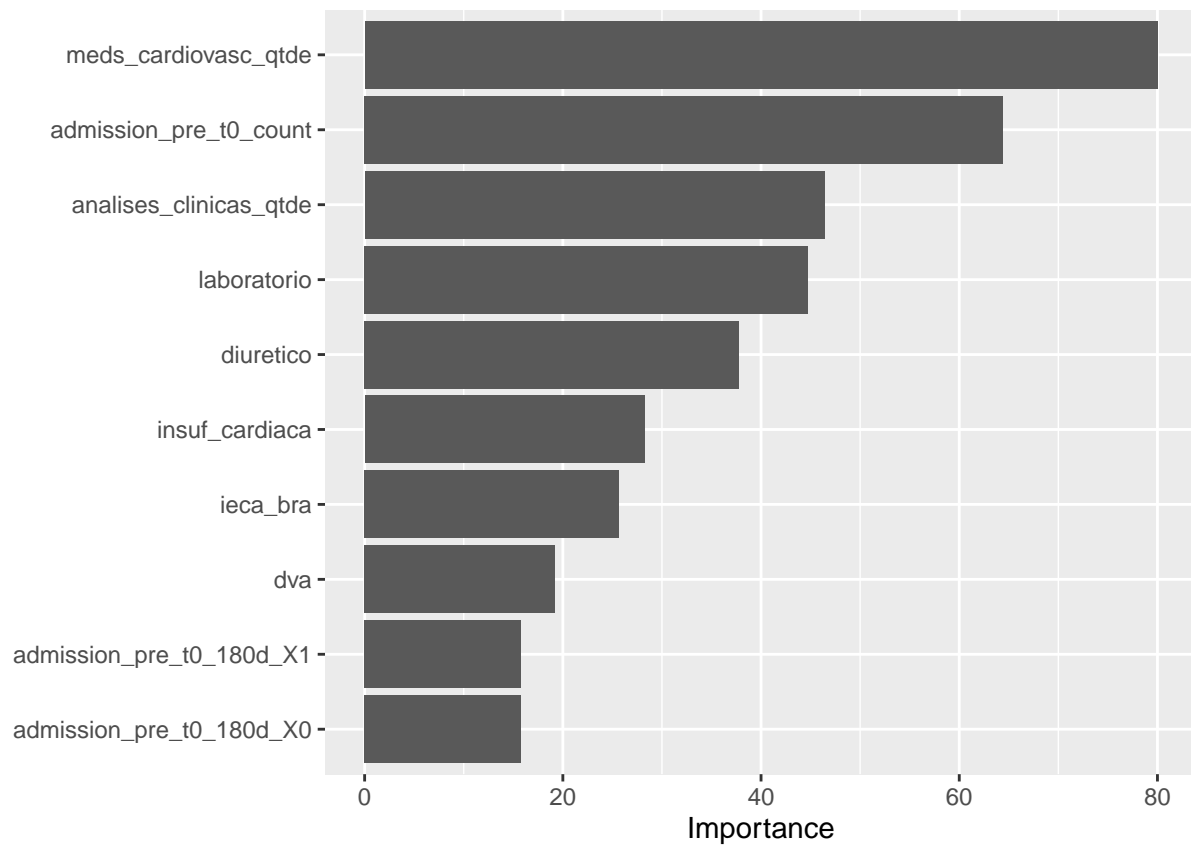
```
autoplot(tree_tune, metric = "roc_auc")
```



```
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()
}
```


Minutes to run: 1.223

Random Forest

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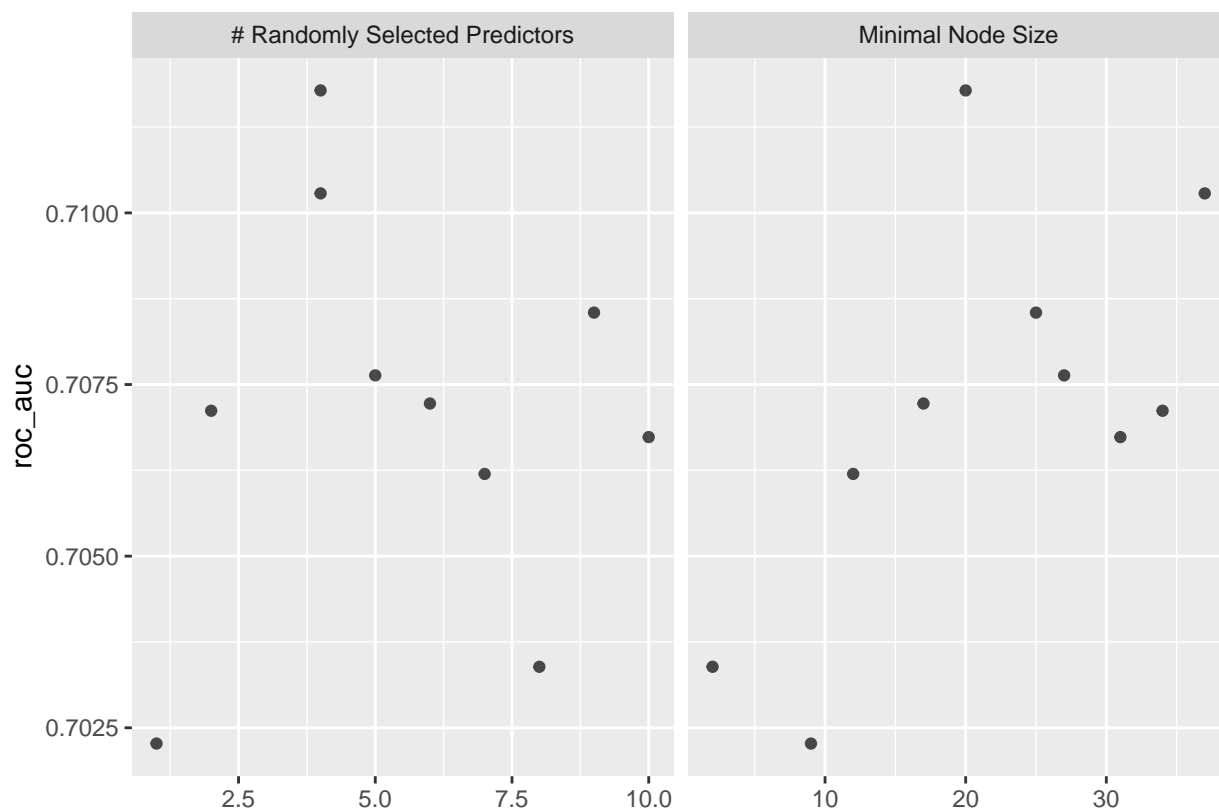
```r
rf_tune %>%
  show_best("roc_auc")

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)
```
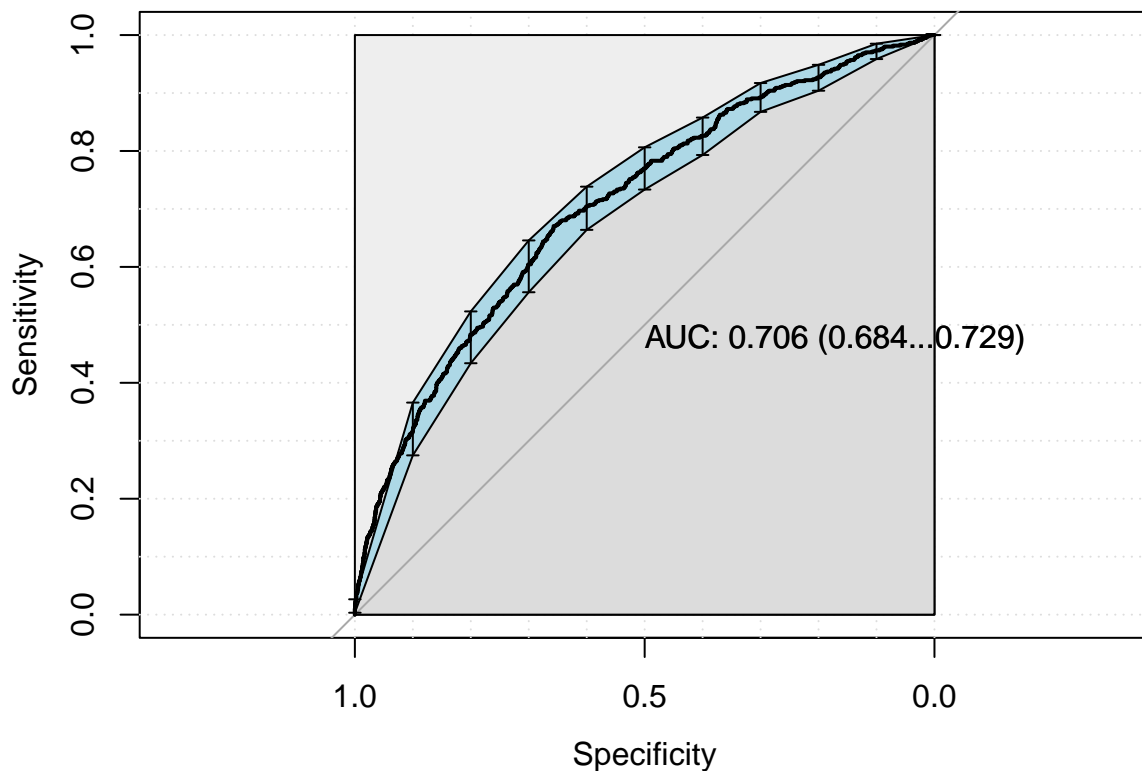
AUC: 0.706 (0.684...0.729)

Minutes to run: 1.313

KNN

```r
# 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()
#
# 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)
```
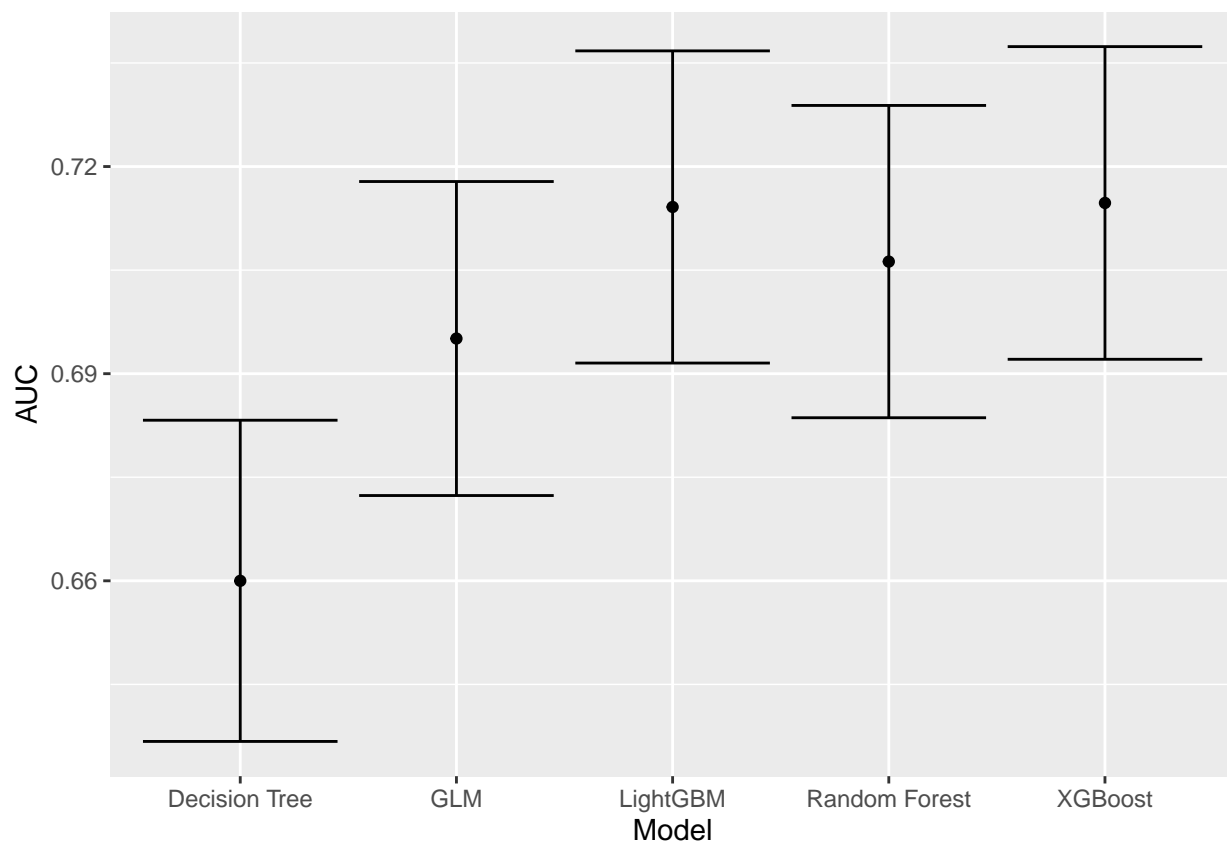
Minutes to run: 0

Models Comparison

```
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("./auxiliar/model_selection/performance/%s.RData", outcome_column))
```

Minutes to run: 0.002