

# 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: 0.002

## Filtering eligible patients

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

df %>% dim
```

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

Minutes to run: 0.008

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

```

```
## 47. cec
## 48. transplante_cardiaco
## 49. cir_toracica
## 50. outros_proced_cirurgicos
## 51. icp
## 52. angioplastia
## 53. cateterismo
## 54. eletrofisiologia
## 55. cateter_venoso_central
## 56. proced_invasivos_qtde
## 57. cve_desf
## 58. transfusao
## 59. equipe_multiprof
## 60. ecg
## 61. holter
## 62. teste_esforco
## 63. espiro_ergoespiro
## 64. tilt_teste
## 65. metodos_graficos_qtde
## 66. laboratorio
## 67. cultura
## 68. analises_clinicas_qtde
## 69. citologia
## 70. biopsia
## 71. histopatologia_qtde
## 72. angio_rm
## 73. angio_tc
## 74. arteriografia
## 75. cintilografia
## 76. ecocardiograma
## 77. endoscopia
## 78. pet_ct
## 79. ultrassom
## 80. tomografia
## 81. radiografia
## 82. ressonancia
## 83. exames_imagem_qtde
## 84. bic
```

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: 0.001

## 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

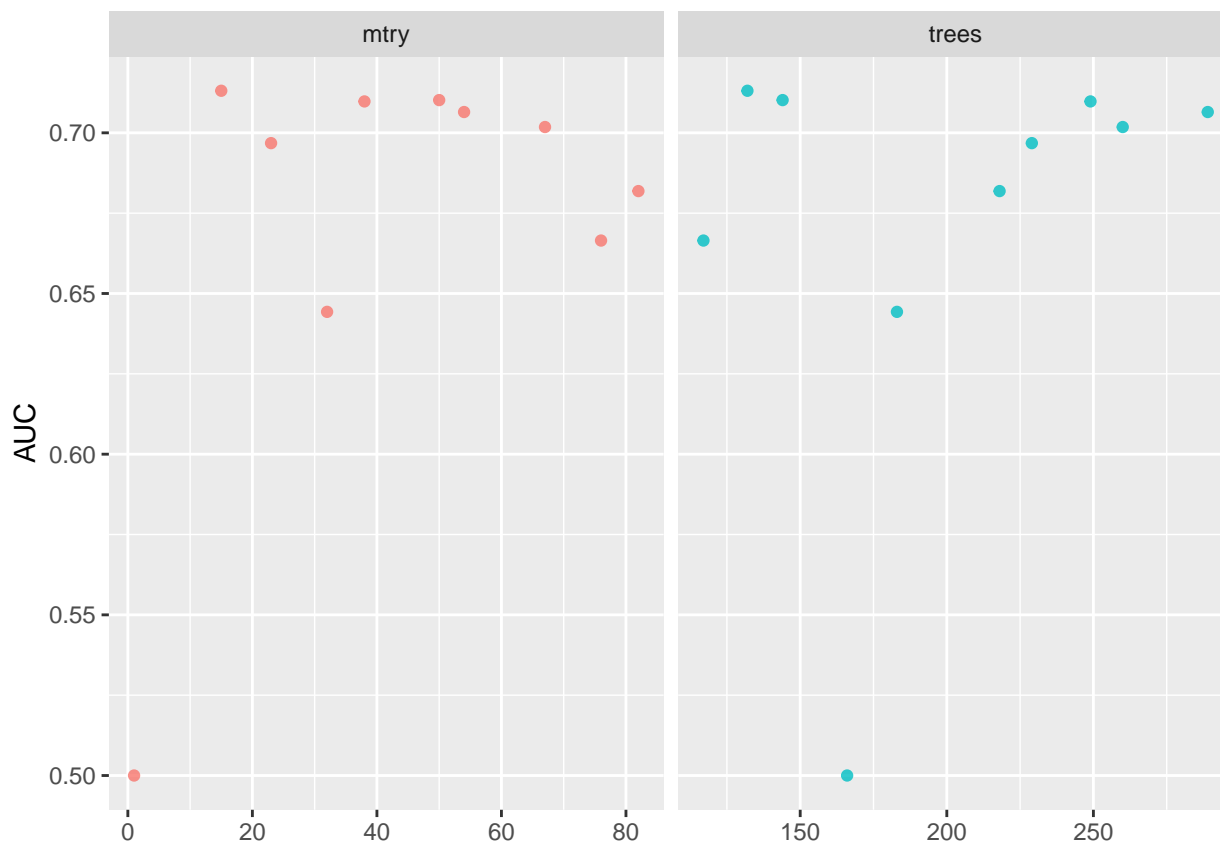
## Boosted Tree (XGBoost)

```
xgboost_recipe <-  
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%  
  step_nominal(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_r~1 loss_r~2 .metric .esti~3 mean  
##   <int> <int> <int>      <int>      <dbl>      <dbl> <chr>      <chr>      <dbl>  
## 1    15   132     6         7  2.08e-2  5.12e- 8 roc_auc binary  0.713  
## 2    50   144    26        15  1.73e-6  5.15e-10 roc_auc binary  0.710  
## 3    38   249    30         5  2.69e-3  1.41e+ 0 roc_auc binary  0.710  
## 4    54   289     6        13  2.22e-5  2.81e- 4 roc_auc binary  0.706  
## 5    67   260    40        11  9.74e-5  3.79e- 7 roc_auc binary  0.702  
## # ... with 3 more variables: n <int>, std_err <dbl>, .config <chr>,  
## #   and abbreviated variable names 1: learn_rate, 2: loss_reduction,  
## #   3: .estimator  
## # i Use 'colnames()' to see all variable names  
  
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")

```



```

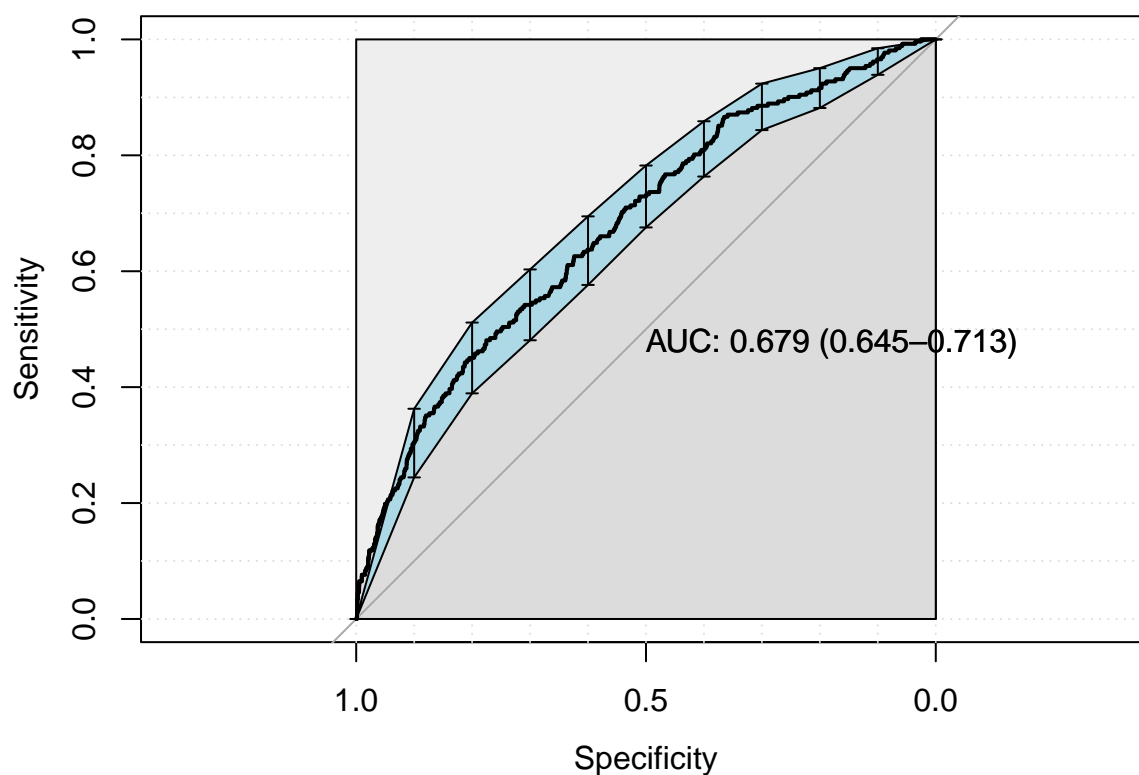
final_xgboost_workflow <-
  xgboost_workflow %>%
  finalize_workflow(best_xgboost)

last_xgboost_fit <-
  final_xgboost_workflow %>%
  last_fit(df_split)

final_xgboost_fit <- extract_workflow(last_xgboost_fit)

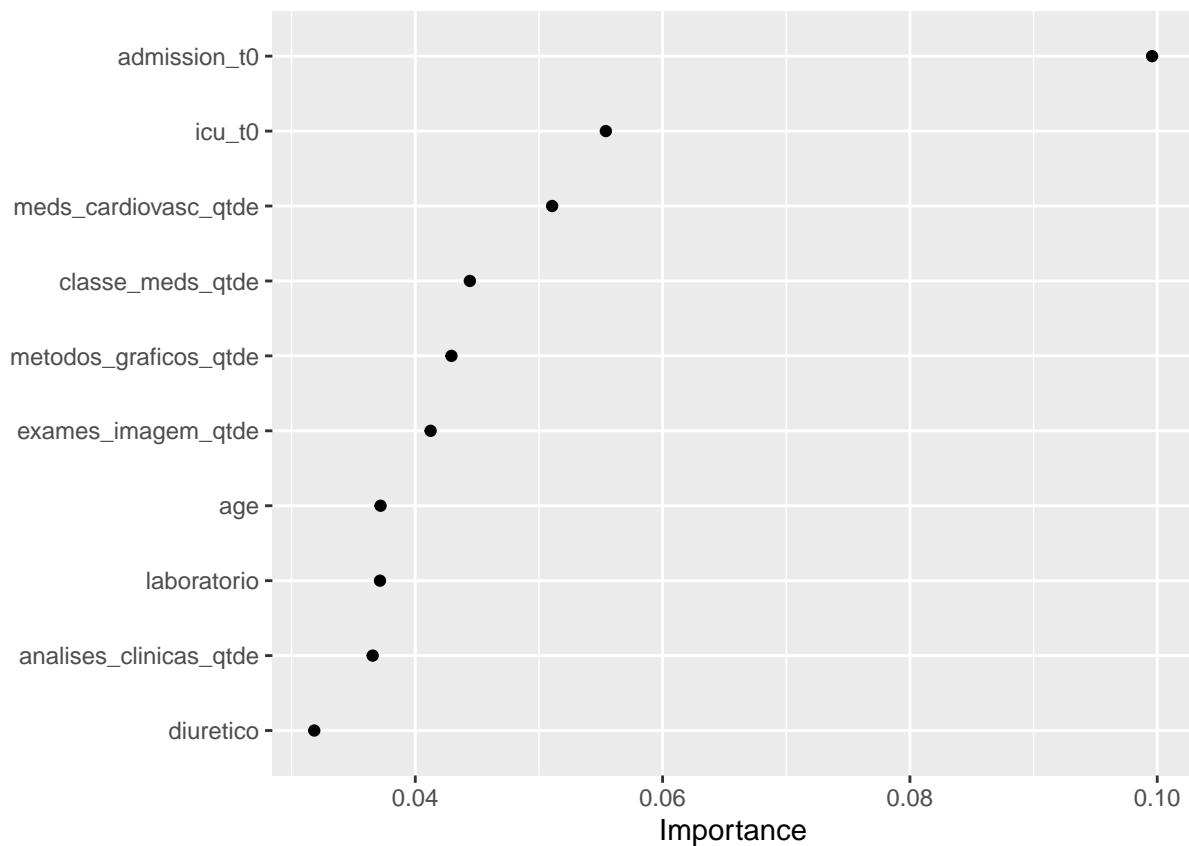
xgboost_auc <- validation(final_xgboost_fit, df_test)

```



```
## Confusion Matrix and Statistics
##
##
## test_predictions_class    0    1
##                0 4468  262
##                1    0    0
##
##          Accuracy : 0.9446
##          95% CI   : (0.9377, 0.951)
##    No Information Rate : 0.9446
##    P-Value [Acc > NIR] : 0.5164
##
##          Kappa : 0
##
##  McNemar's Test P-Value : <2e-16
##
##          Sensitivity : 1.0000
##          Specificity : 0.0000
##    Pos Pred Value : 0.9446
##    Neg Pred Value :    NaN
##    Prevalence : 0.9446
##    Detection Rate : 0.9446
##    Detection Prevalence : 1.0000
##    Balanced Accuracy : 0.5000
##
##    '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: 3.71

## 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
)
```



```

) %>%
  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)

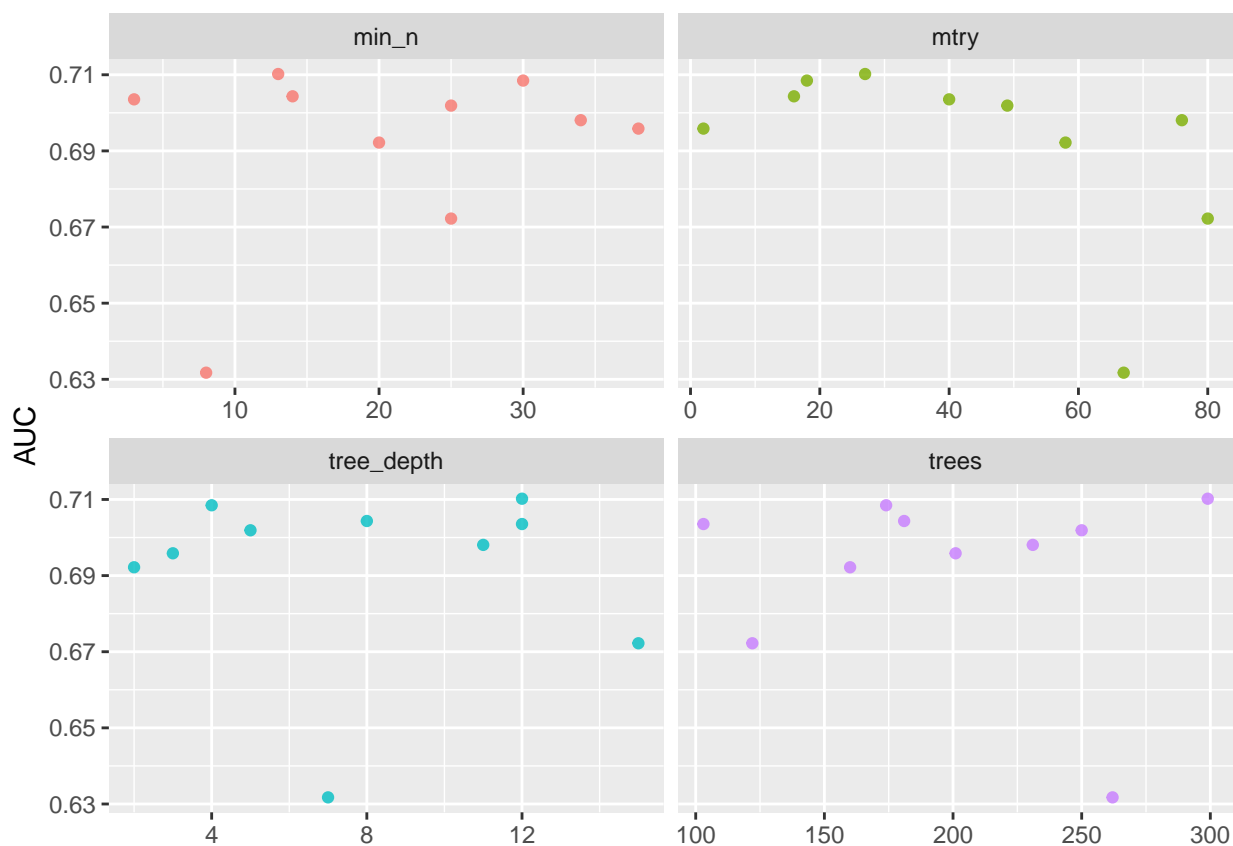
lightgbm_tune %>%
  show_best("roc_auc")

## # A tibble: 5 x 12
##   mtry trees min_n tree_depth learn_r~1 loss_~2 .metric .esti~3 mean
##   <int> <int> <int>      <int>      <dbl>   <dbl> <chr>   <chr>   <dbl>
## 1     27   299    13         12  7.21e-7 2.90e-3 roc_auc binary  0.710
## 2     18   174    30          4  6.62e-9 9.01e-6 roc_auc binary  0.708
## 3     16   181    14          8  2.68e-3 2.23e-7 roc_auc binary  0.704
## 4     40   103     3         12  1.22e-4 9.31e-1 roc_auc binary  0.704
## 5     49   250    25          5  2.02e-4 3.75e-6 roc_auc binary  0.702
## # ... with 3 more variables: n <int>, std_err <dbl>, .config <chr>,
## #   and abbreviated variable names 1: learn_rate, 2: loss_reduction,
## #   3: .estimator
## # i Use 'colnames()' to see all variable names

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")

```

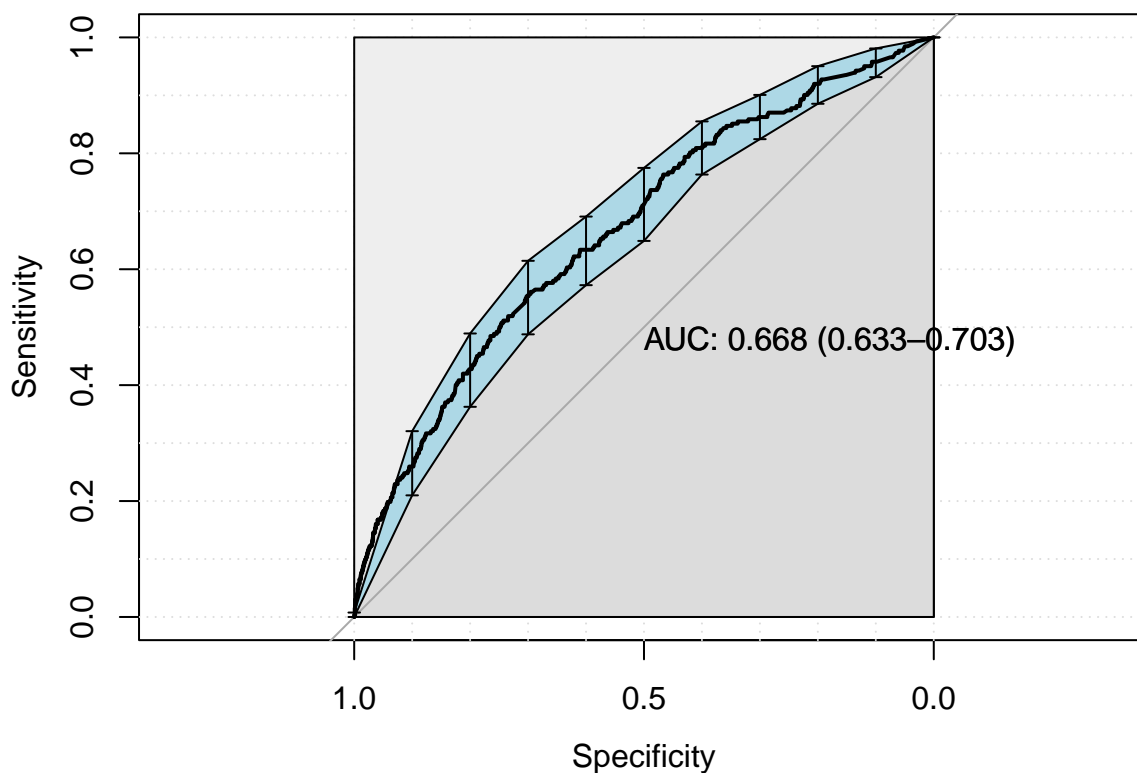


```
final_lightgbm_workflow <-
  lightgbm_workflow %>%
  finalize_workflow(best_lightgbm)

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 4468  262
##                1    0    0
##
##          Accuracy : 0.9446
##          95% CI   : (0.9377, 0.951)
##    No Information Rate : 0.9446
##    P-Value [Acc > NIR] : 0.5164
##
##          Kappa : 0
##
##  McNemar's Test P-Value : <2e-16
##
##          Sensitivity : 1.0000
##          Specificity : 0.0000
##    Pos Pred Value : 0.9446
##    Neg Pred Value :   NaN
##    Prevalence : 0.9446
##    Detection Rate : 0.9446
##    Detection Prevalence : 1.0000
##    Balanced Accuracy : 0.5000
##
##    '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: 1.418

## 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())

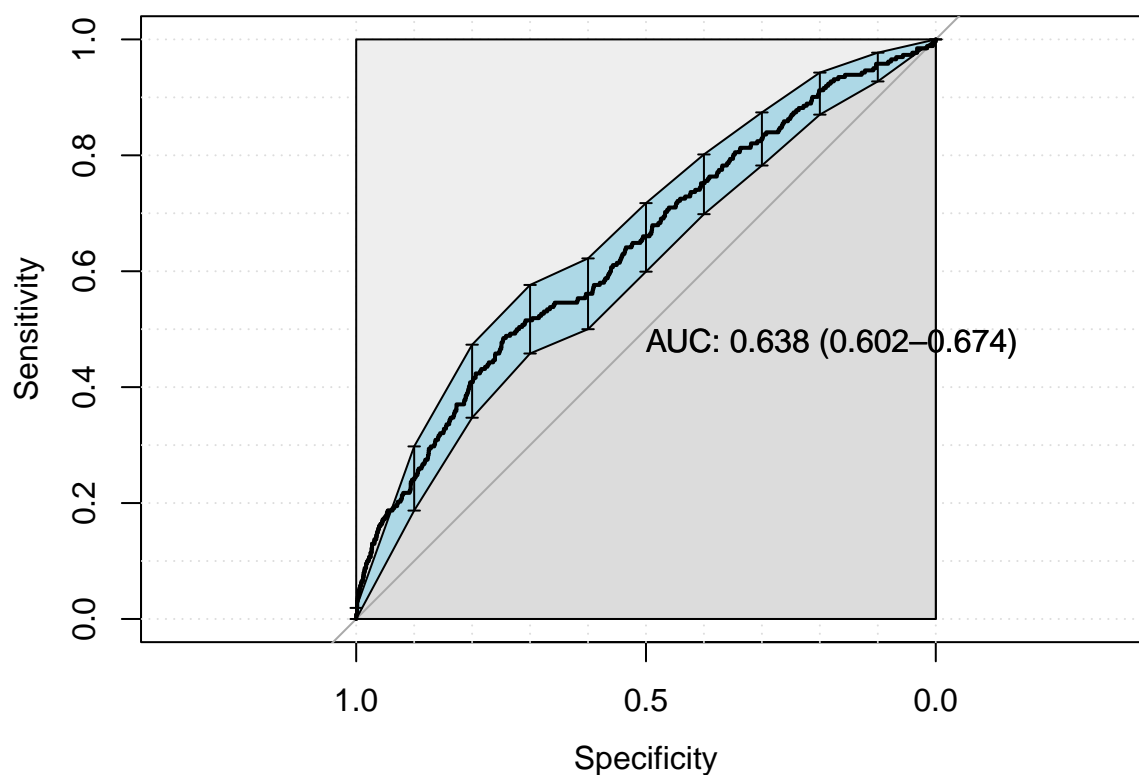
glmnet_spec <-
  logistic_reg(penalty = 0) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

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 4466 260
##                        1    2    2
```

##

```
##           Accuracy : 0.9446
##           95% CI   : (0.9377, 0.951)
##    No Information Rate : 0.9446
##    P-Value [Acc > NIR] : 0.5164
```

##

```
##           Kappa : 0.0134
```

##

```
## McNemar's Test P-Value : <2e-16
```

##

```
##           Sensitivity : 0.999552
```

```
##           Specificity : 0.007634
```

```
##    Pos Pred Value : 0.944985
```

```
##    Neg Pred Value : 0.500000
```

```
##           Prevalence : 0.944609
```

```
##    Detection Rate : 0.944186
```

```
##    Detection Prevalence : 0.999154
```

```
##    Balanced Accuracy : 0.503593
```

##

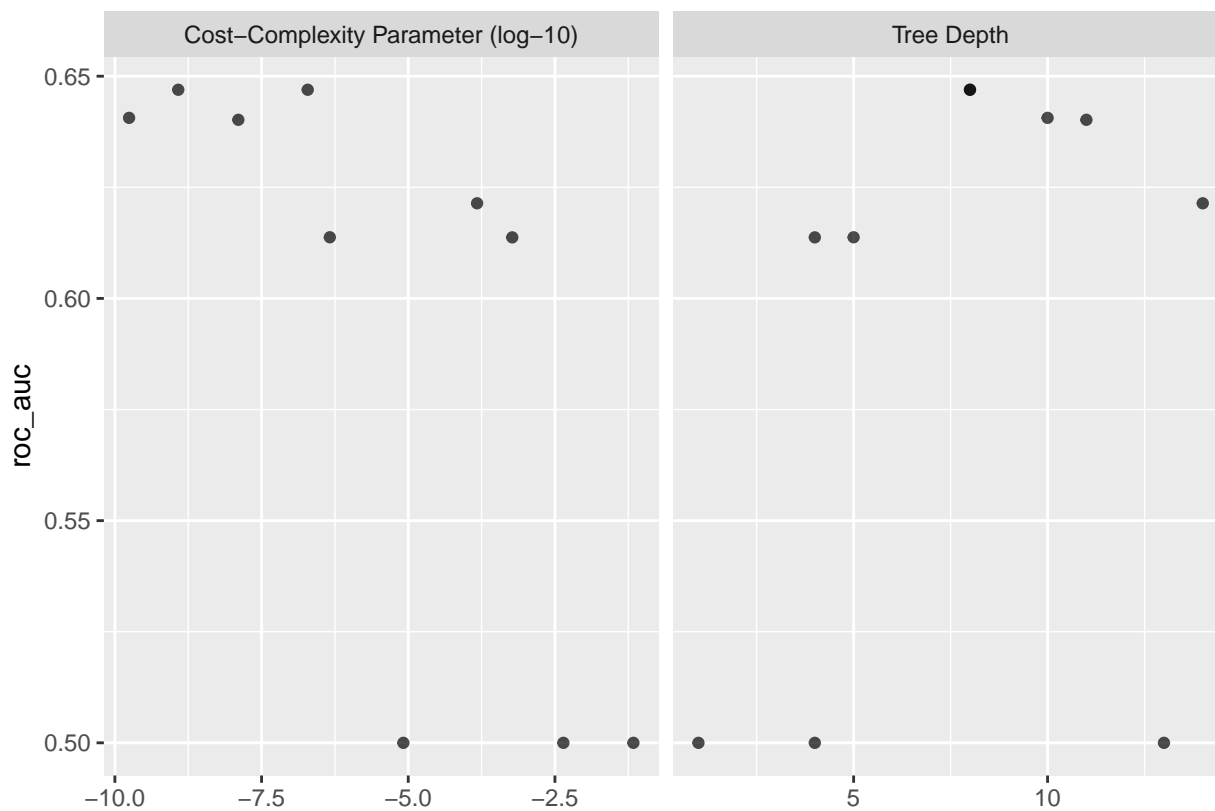
```
##           'Positive' Class : 0
```

##

Minutes to run: 0.153

## Decision Tree

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

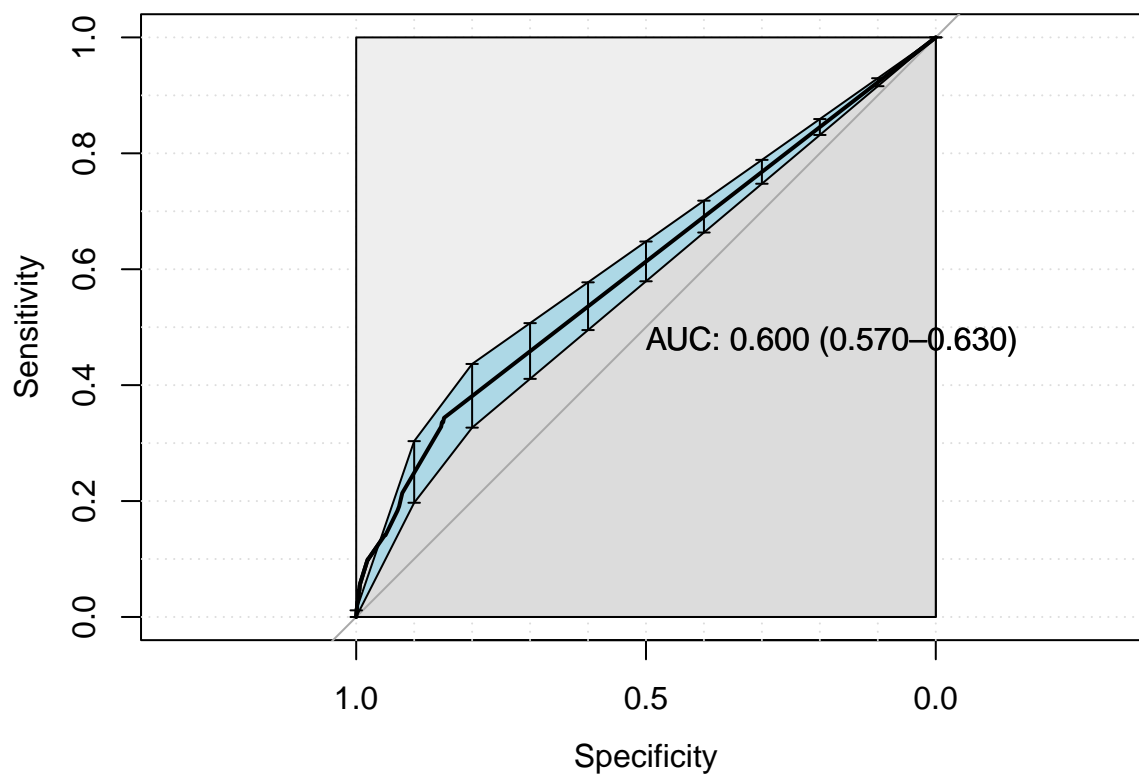
best_tree <- tree_tune %>%
  select_best("roc_auc")

final_tree_workflow <-
  tree_workflow %>%
  finalize_workflow(best_tree)

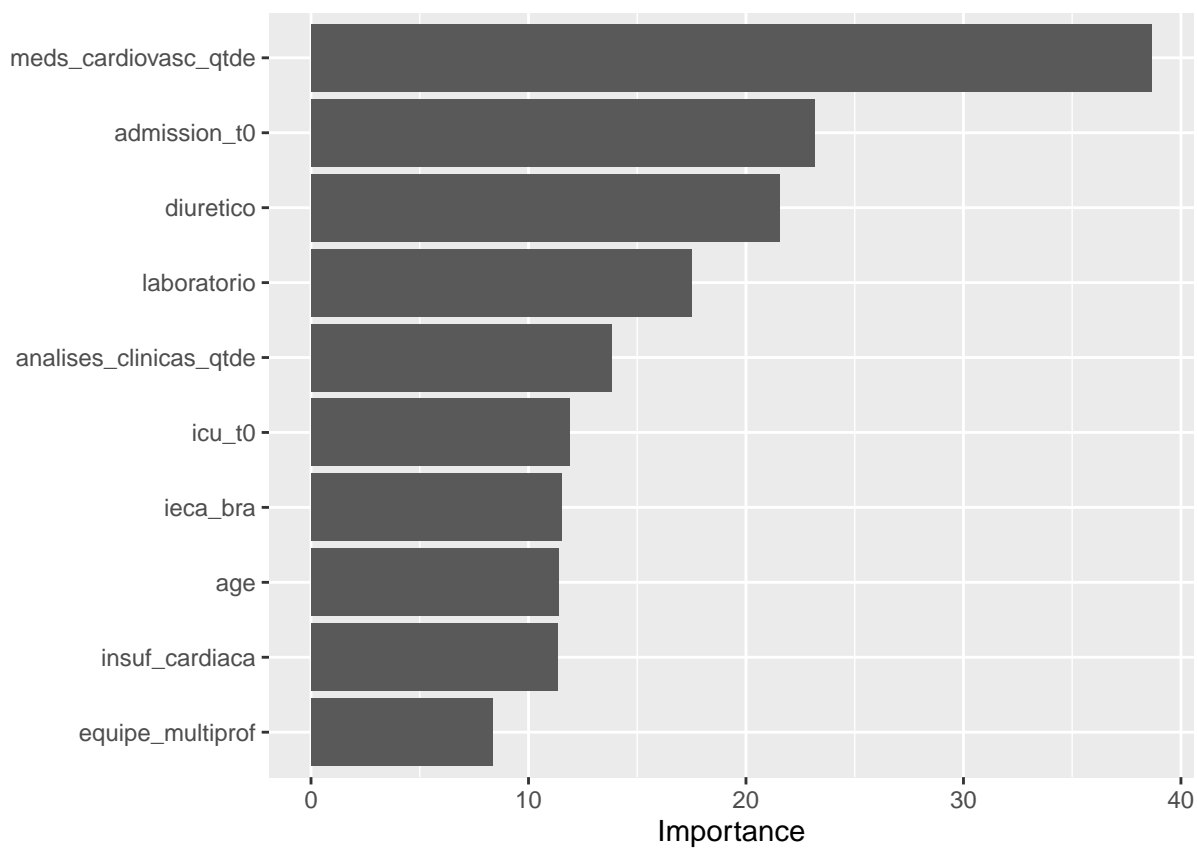
last_tree_fit <-
  final_tree_workflow %>%
  last_fit(df_split)

final_tree_fit <- extract_workflow(last_tree_fit)

tree_auc = validation(final_tree_fit, df_test)
```



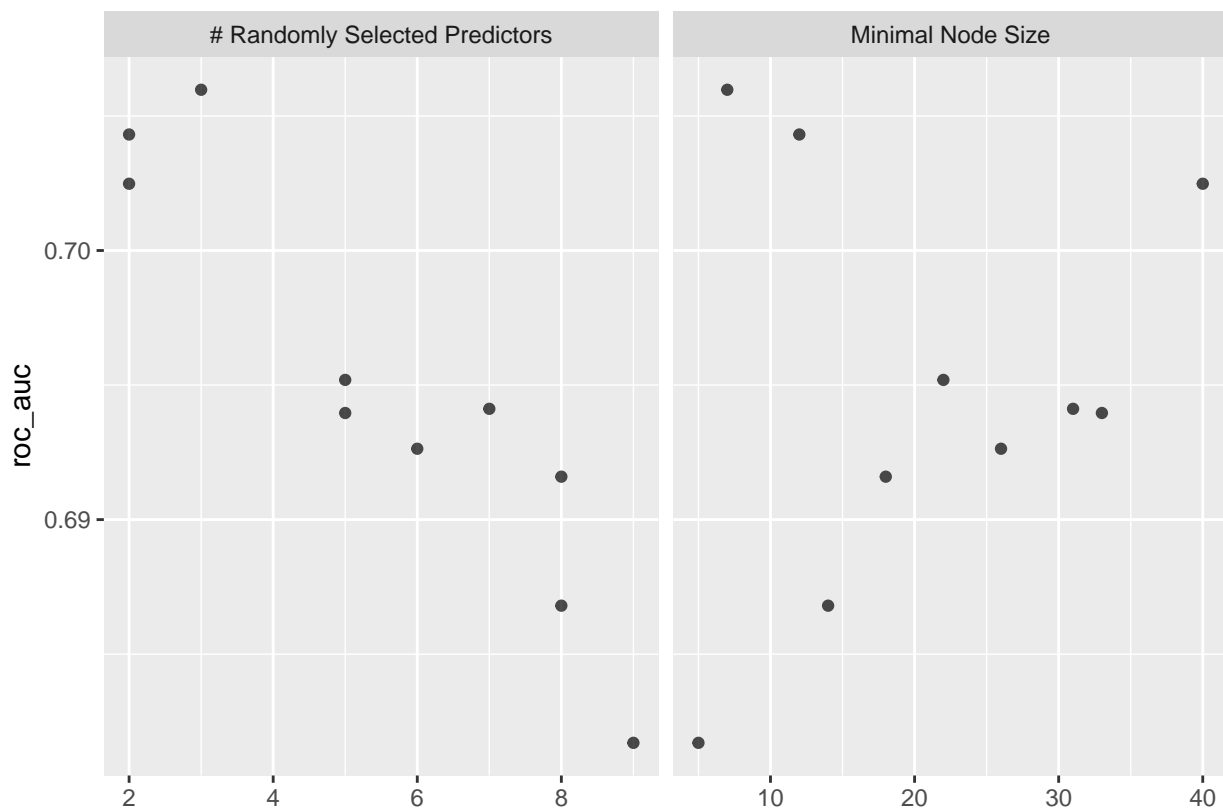
```
if (tree_auc$auc > 0.55){
  final_tree_fit %>%
    extract_fit_parsnip() %>%
    vip()
}
```





# 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")
```



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

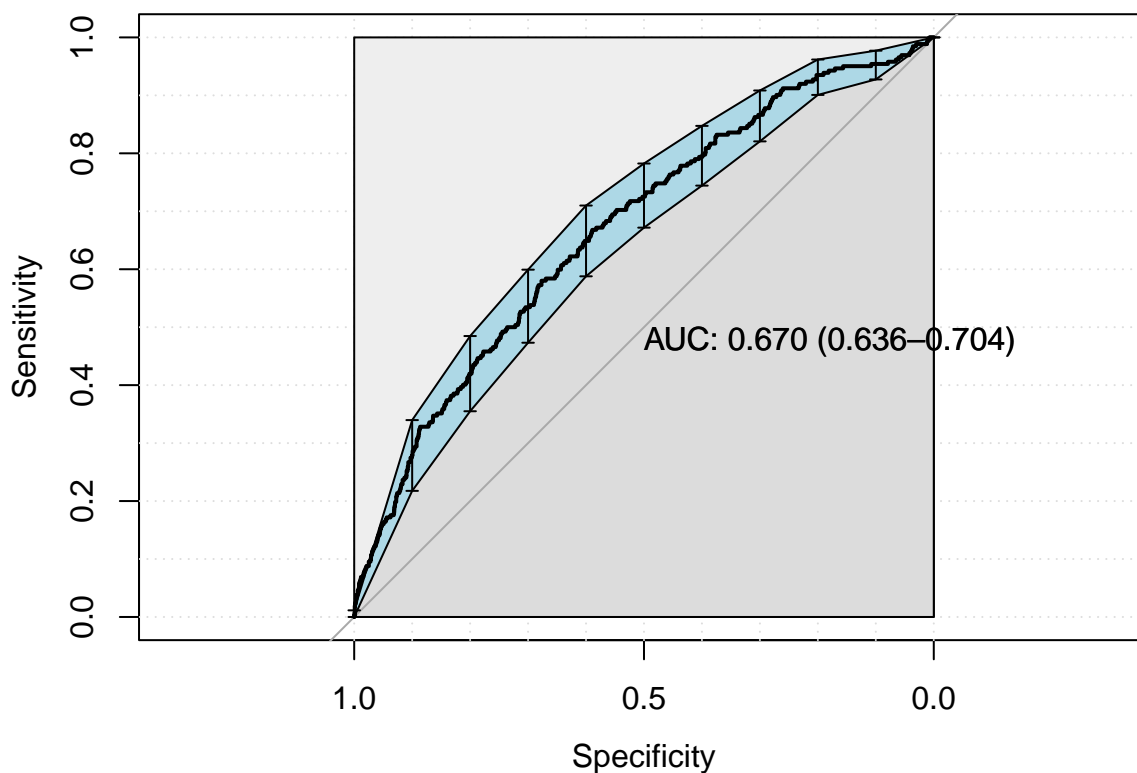
best_rf <- rf_tune %>%
  select_best("roc_auc")

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:

1.103

## 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_nominal(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(sum_tune, metric = "roc_auc")
#
# sum_tune %>%
#   show_best("roc_auc")
#
# best_sum <- sum_tune %>%
#   select_best("roc_auc")
#
# final_sum_workflow <-
#   sum_workflow %>%
#   finalize_workflow(best_sum)
#
# last_sum_fit <-
#   final_sum_workflow %>%
#   last_fit(df_split)
#
# final_sum_fit <- extract_workflow(last_sum_fit)
#
# sum_auc = validation(final_sum_fit, df_test)

```

Minutes to run: 0

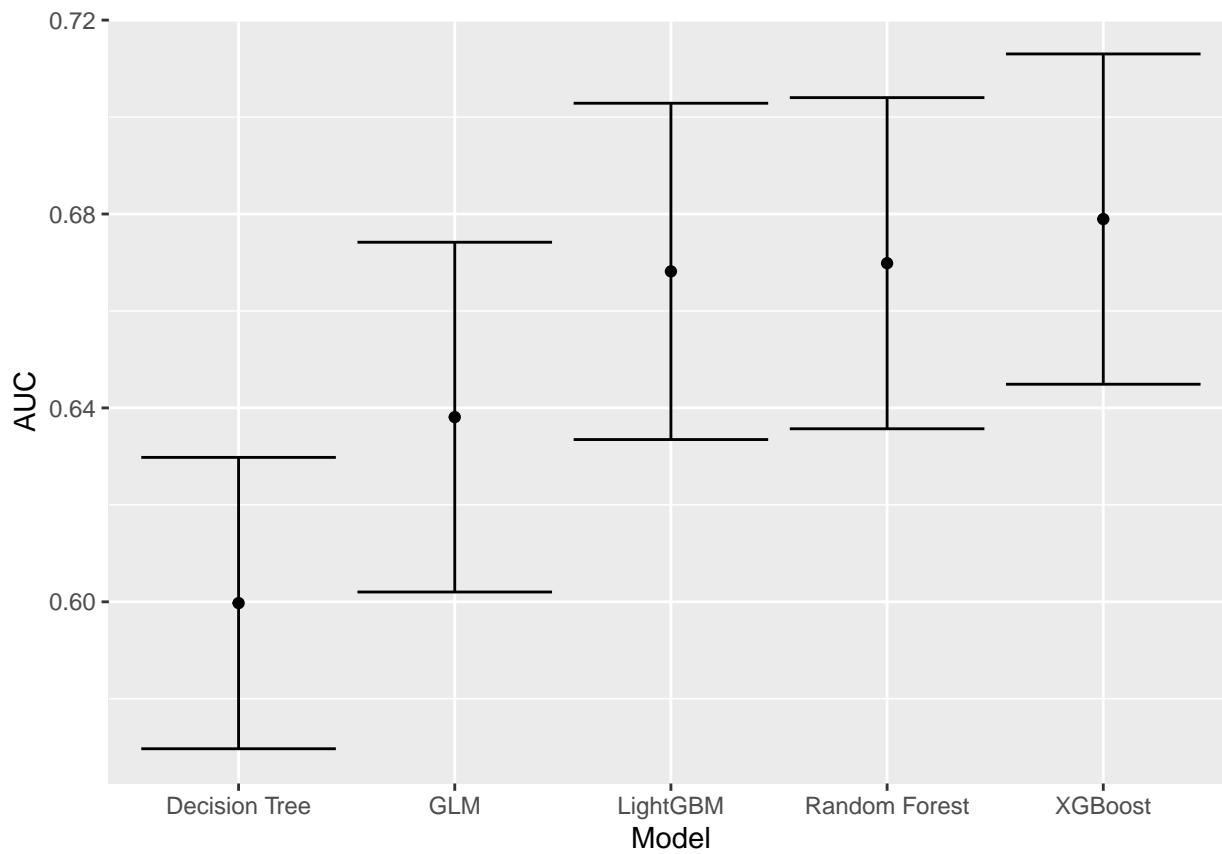
## Models Comparison

```

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

Minutes to run: 0.009