

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: 0.006

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_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. education_level
## 04. underlying_heart_disease
## 05. heart_disease
## 06. nyha_basal
## 07. hypertension
## 08. prior_mi
## 09. heart_failure
## 10. af
## 11. cardiac_arrest
## 12. valvopathy
## 13. diabetes
## 14. renal_failure
## 15. hemodialysis
## 16. stroke
## 17. copd
## 18. comorbidities_count
## 19. procedure_type_1
## 20. reop_type_1
## 21. procedure_type_new
## 22. cied_final_1
## 23. cied_final_group_1
## 24. admission_pre_t0_count
## 25. admission_pre_t0_180d
## 26. year_adm_t0
## 27. icu_t0
## 28. dialysis_t0
## 29. disch_outcomes_t0
## 30. admission_t0_emergency
## 31. aco
## 32. antiarritmico
## 33. ieca_bra
## 34. dva
## 35. digoxina
## 36. estatina
## 37. diuretico
## 38. vasodilatador
## 39. insuf_cardiaca
## 40. espironolactona
## 41. antiplaquetario_ev
## 42. insulina
## 43. psicofarmacos
## 44. classe_meds_qtde
## 45. meds_cardiovasc_qtde
## 46. meds_antimicrobianos

```

```
## 47. outros_proced_cirurgicos
## 48. icp
## 49. angioplastia
## 50. cateterismo
## 51. eletrofisiologia
## 52. cateter_venoso_central
## 53. proced_invasivos_qtde
## 54. transfusao
## 55. equipe_multiprof
## 56. ecg
## 57. holter
## 58. teste_esforco
## 59. tilt_teste
## 60. metodos_graficos_qtde
## 61. laboratorio
## 62. cultura
## 63. analises_clinicas_qtde
## 64. citologia
## 65. histopatologia_qtde
## 66. angio_tc
## 67. angiografia
## 68. cintilografia
## 69. ecocardiograma
## 70. endoscopia
## 71. flebografia
## 72. pet_ct
## 73. ultrassom
## 74. tomografia
## 75. radiografia
## 76. ressonancia
## 77. exames_imagem_qtde
## 78. 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: 0.004

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.polygons = TRUE,
    max.auc.polygons = 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)

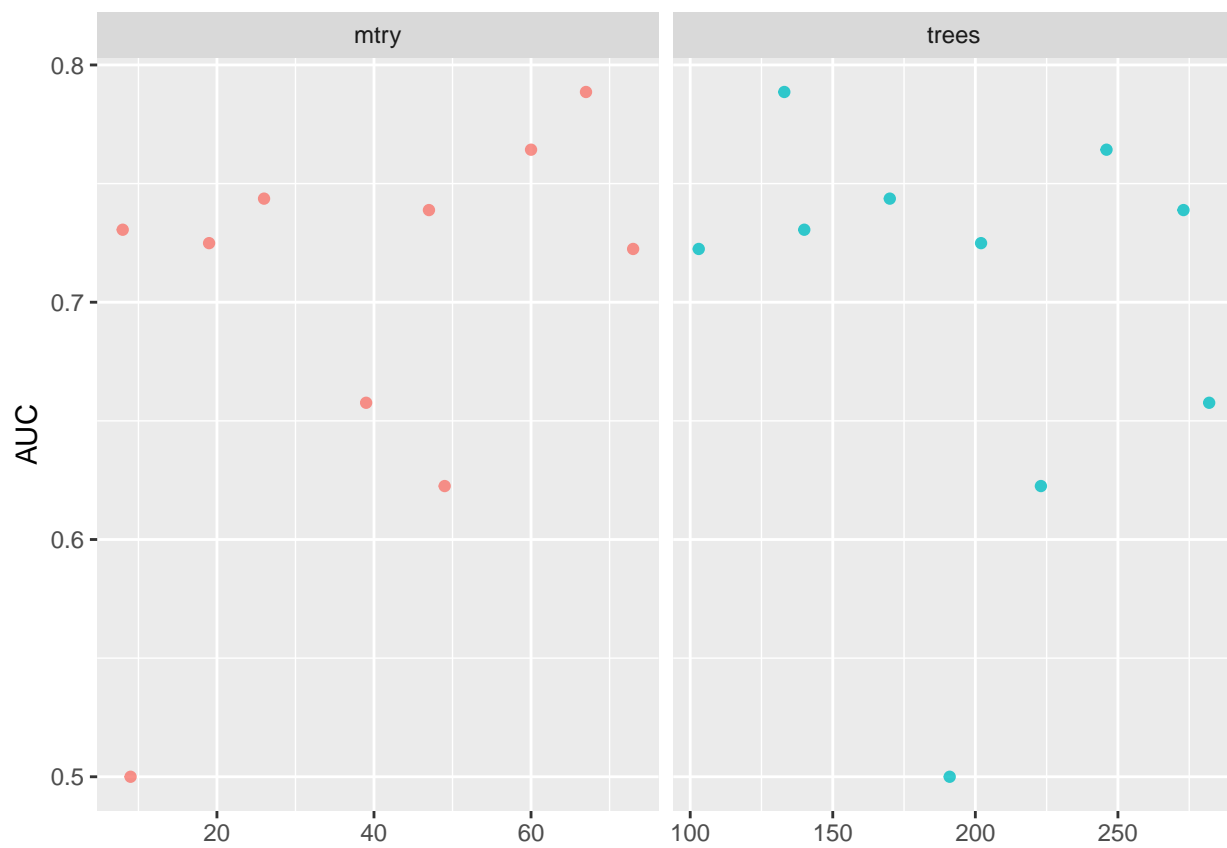
xgboost_tune %>%
  show_best("roc_auc")

## # A tibble: 5 x 12
##   mtry trees min_n tree_depth learn_rate loss_reduction .metric .estimator mean n std_err .config
##   <int> <int> <int>    <int>    <dbl>    <dbl> <chr>    <chr>    <dbl> <int>    <dbl> <chr>
## 1     67   133     9        13 0.0779      8.83e- 6 roc_auc binary    0.789     4 0.00828 Preprocessor
## 2     60   246    23        14 0.00557      9.89e- 4 roc_auc binary    0.764     4 0.0192  Preprocessor
## 3     26   170    16         9 0.000000189 7.14e-10 roc_auc binary    0.744     4 0.0210  Preprocessor
## 4     47   273    33         6 0.000158     1.57e- 1 roc_auc binary    0.739     4 0.0175  Preprocessor
## 5      8   140    39         4 0.00000215 7.36e- 4 roc_auc binary    0.731     4 0.0194  Preprocessor

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")

```

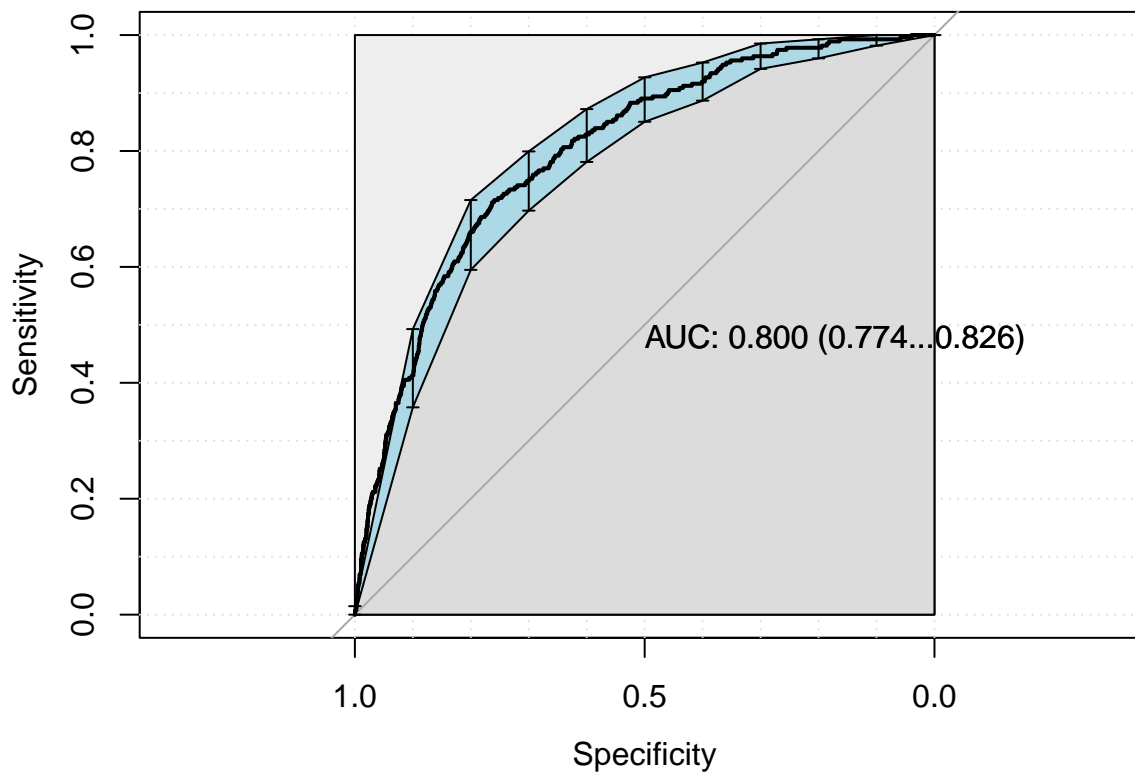


```
final_xgboost_workflow <-
  xgboost_workflow %>%
  finalize_workflow(best_xgboost)

last_xgboost_fit <-
  final_xgboost_workflow %>%
  last_fit(df_split)

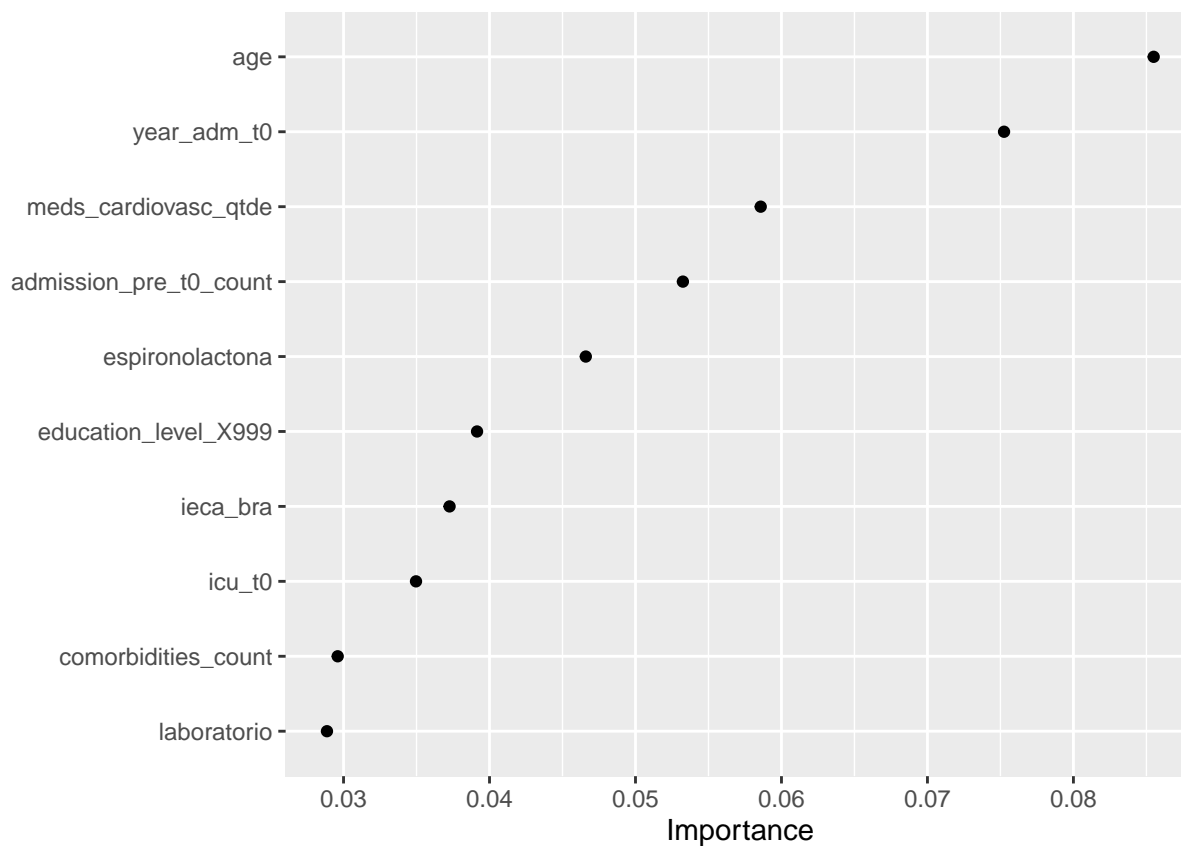
final_xgboost_fit <- extract_workflow(last_xgboost_fit)

xgboost_auc <- validation(final_xgboost_fit, df_test)
```



```
## |
## Confusion Matrix and Statistics
##
##
## test_predictions_class    0    1
##                0 4445  268
##                1   11    6
##
##                Accuracy : 0.941
##                95% CI : (0.9339, 0.9476)
##      No Information Rate : 0.9421
##      P-Value [Acc > NIR] : 0.637
##
##                Kappa : 0.0347
##
## Mcnemar's Test P-Value : <2e-16
##
##                Sensitivity : 0.9975
##                Specificity : 0.0219
##      Pos Pred Value : 0.9431
##      Neg Pred Value : 0.3529
##      Prevalence : 0.9421
##      Detection Rate : 0.9397
##      Detection Prevalence : 0.9964
##      Balanced Accuracy : 0.5097
##
##      '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: 3.791

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)

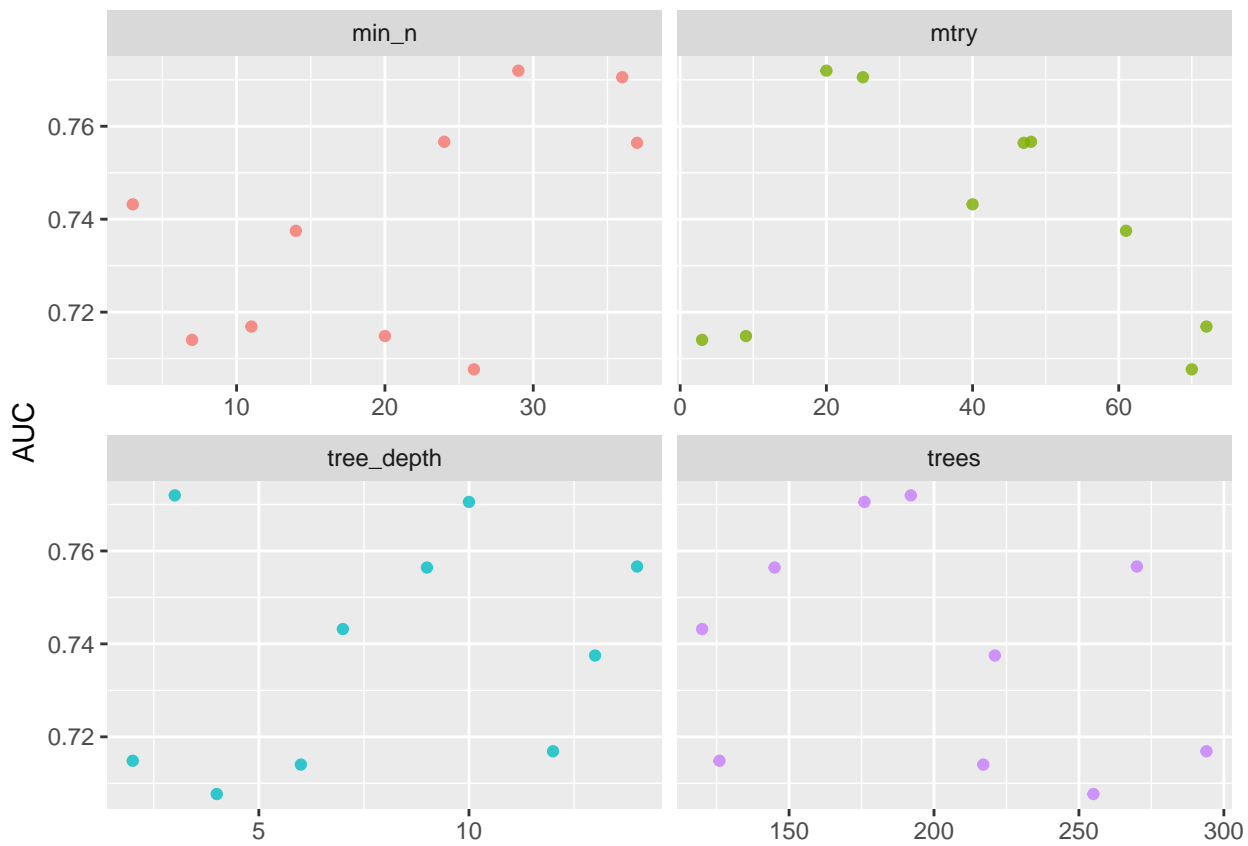
lightgbm_tune %>%
  show_best("roc_auc")

## # A tibble: 5 x 12
##   mtry trees min_n tree_depth   learn_rate loss_reduction .metric .estimator   mean     n std_err .config
##   <int> <int> <int>     <int>         <dbl>         <dbl> <chr>   <chr>     <dbl> <int>   <dbl> <chr>
## 1     20   192    29         3 0.0124             0.000000404 roc_auc binary    0.772     4 0.0151 Preprocess
## 2     25   176    36        10 0.0795             9.19          roc_auc binary    0.771     4 0.0132 Preprocess
## 3     48   270    24        14 0.000543           0.155          roc_auc binary    0.757     4 0.0249 Preprocess
## 4     47   145    37         9 0.0000000353       0.000000144 roc_auc binary    0.756     4 0.0224 Preprocess
## 5     40   120     3         7 0.0000000606       0.00799        roc_auc binary    0.743     4 0.0248 Preprocess

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")

```

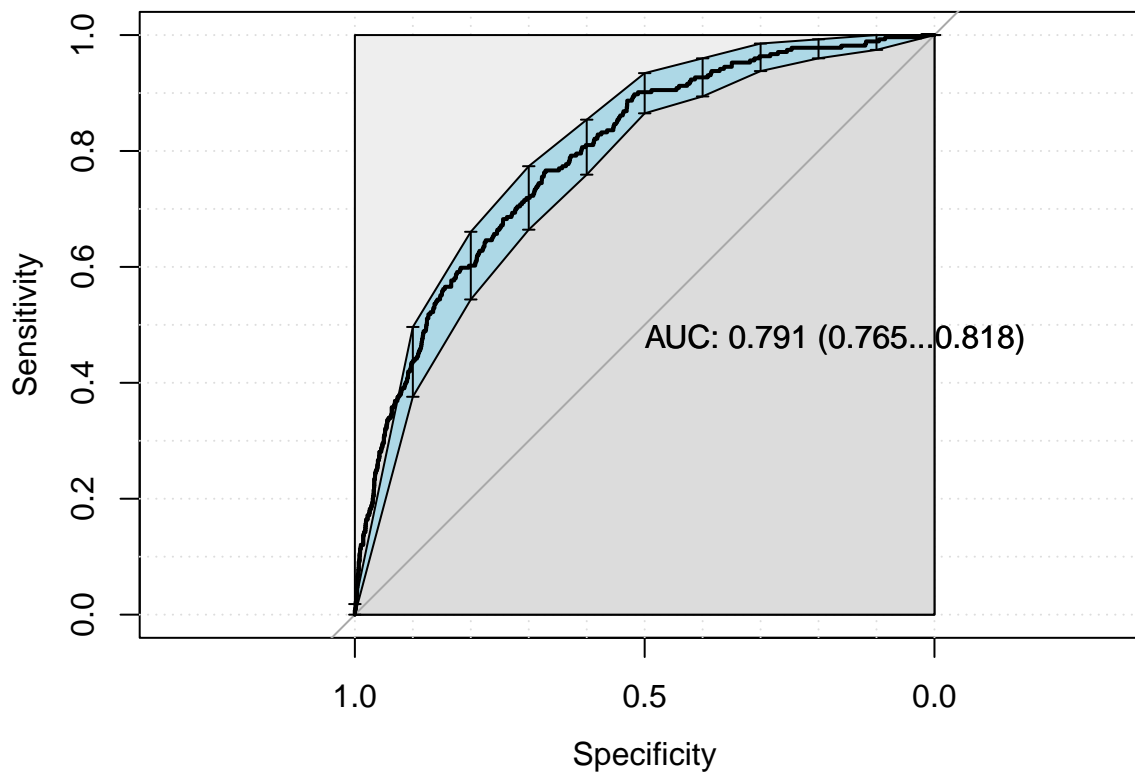


```
final_lightgbm_workflow <-
  lightgbm_workflow %>%
  finalize_workflow(best_lightgbm)

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 4456  274
##                1    0    0
##
##          Accuracy : 0.9421
##          95% CI   : (0.935, 0.9486)
##    No Information Rate : 0.9421
##    P-Value [Acc > NIR] : 0.5161
##
##          Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##          Sensitivity : 1.0000
##          Specificity : 0.0000
##    Pos Pred Value : 0.9421
##    Neg Pred Value :   NaN
##    Prevalence : 0.9421
##    Detection Rate : 0.9421
##    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: 0.55

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())

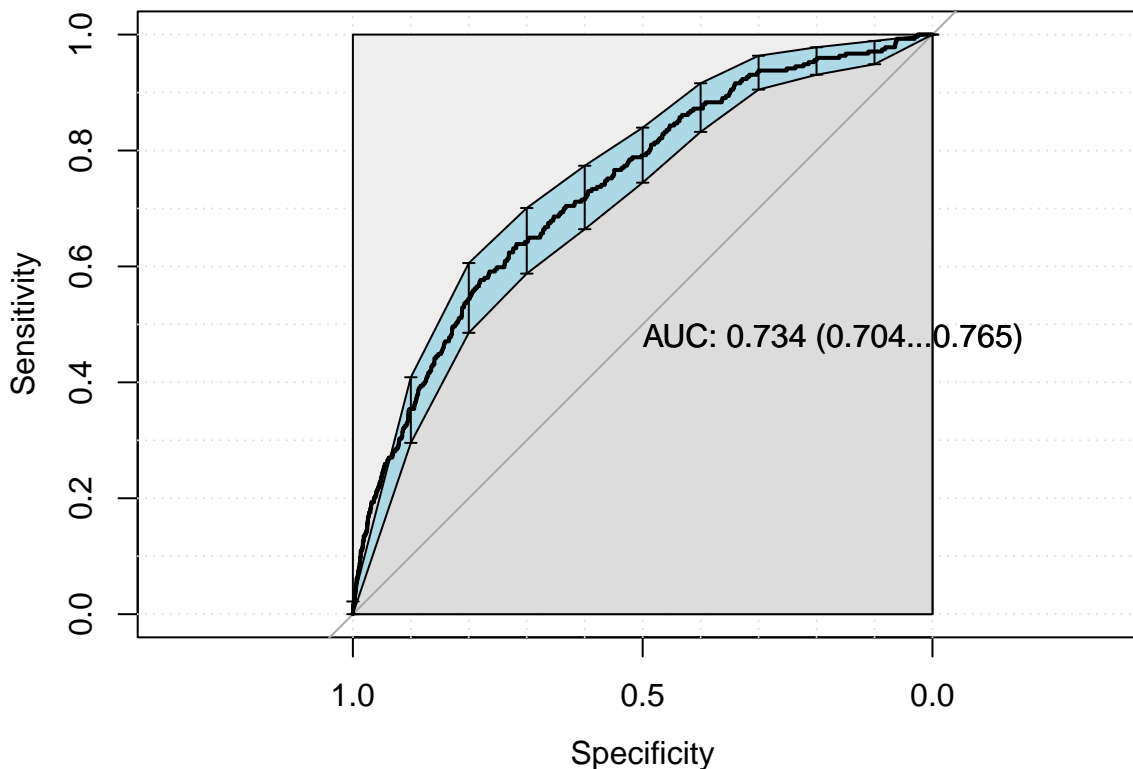
glmnet_spec <-
  logistic_reg(penalty = 0) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

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 4450  270
##                1      6    4
##
##                Accuracy : 0.9416
##                95% CI : (0.9346, 0.9482)
##      No Information Rate : 0.9421
##      P-Value [Acc > NIR] : 0.5653
##
##                Kappa : 0.0242
##
## Mcnemar's Test P-Value : <2e-16
##
##      Sensitivity : 0.9987
##      Specificity : 0.0146
##      Pos Pred Value : 0.9428
##      Neg Pred Value : 0.4000
##      Prevalence : 0.9421
##      Detection Rate : 0.9408
##      Detection Prevalence : 0.9979
##      Balanced Accuracy : 0.5066
##
##      'Positive' Class : 0
##
```

Minutes to run: 0.088

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")

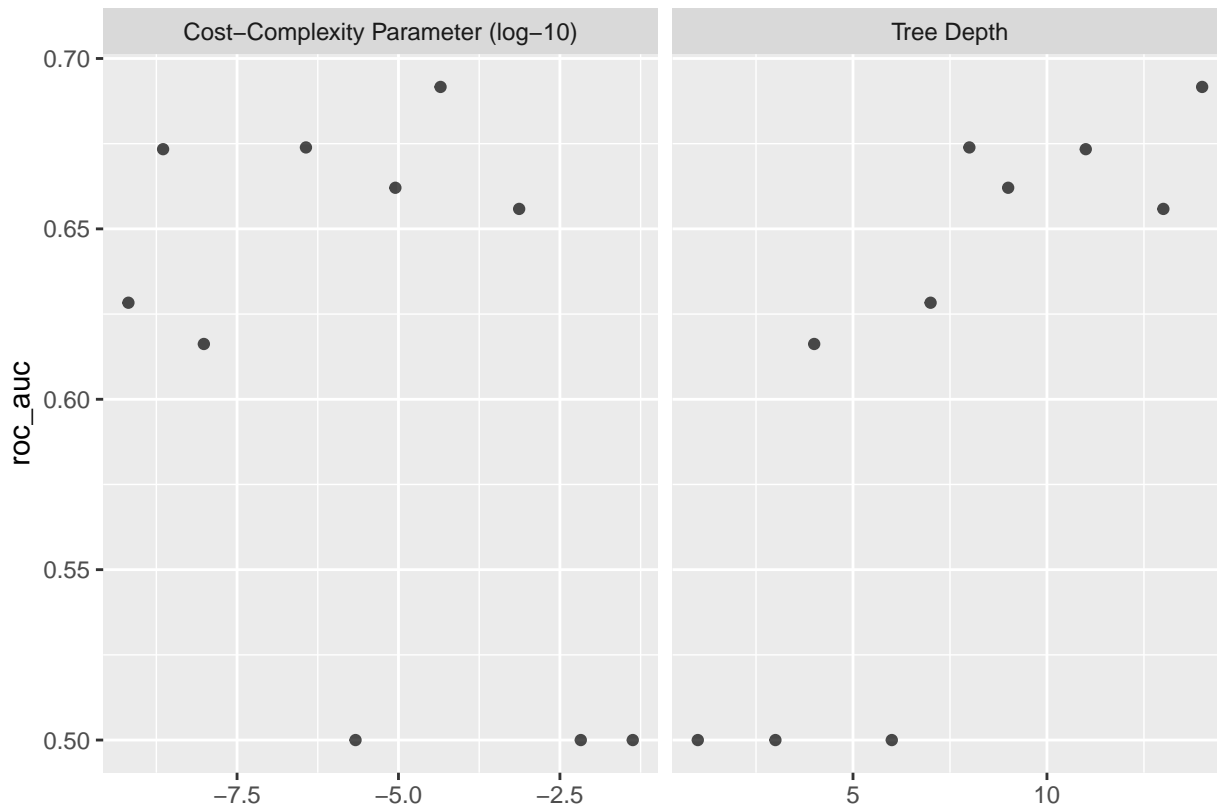
tree_grid <- grid_latin_hypercube(cost_complexity(),
                                  tree_depth(),
                                  size = grid_size)

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

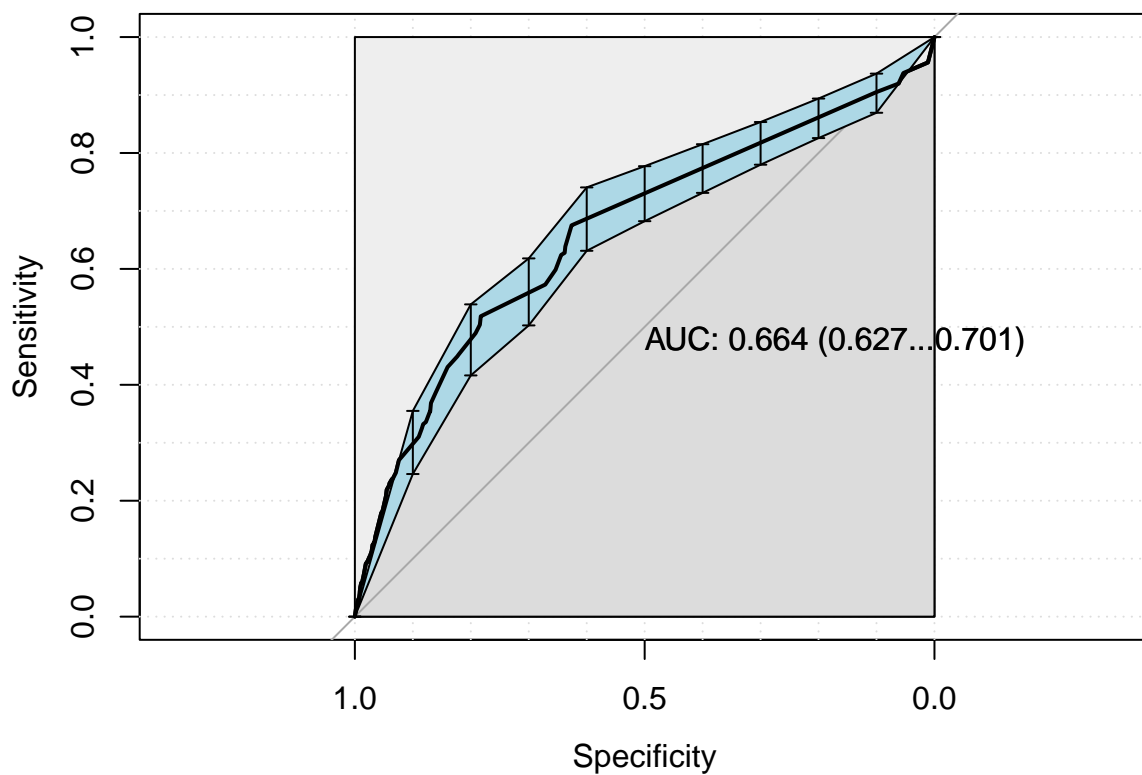
tree_tune <-
  tree_workflow %>%
  tune_grid(resamples = df_folds,
            grid = tree_grid)

tree_tune %>%
  collect_metrics()
```

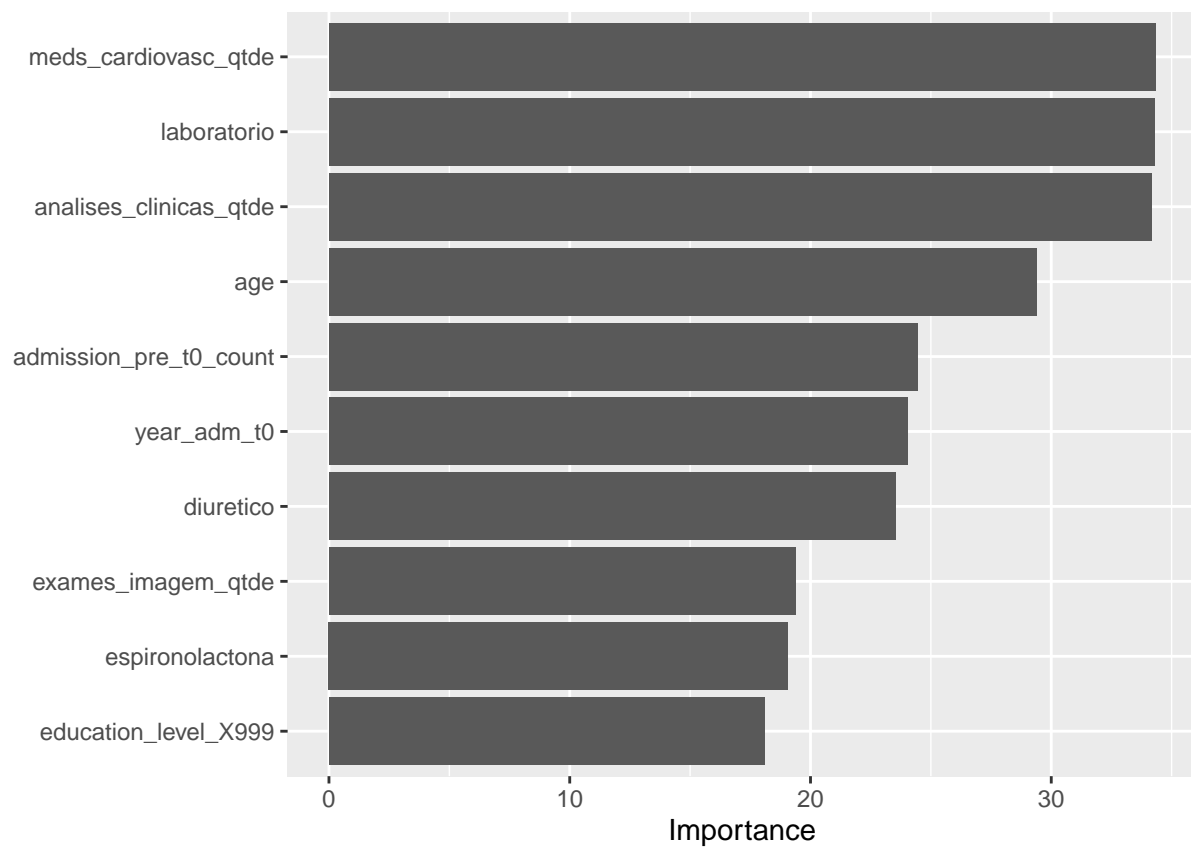
```
autoplot(tree_tune, metric = "roc_auc")
```



```
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)
```



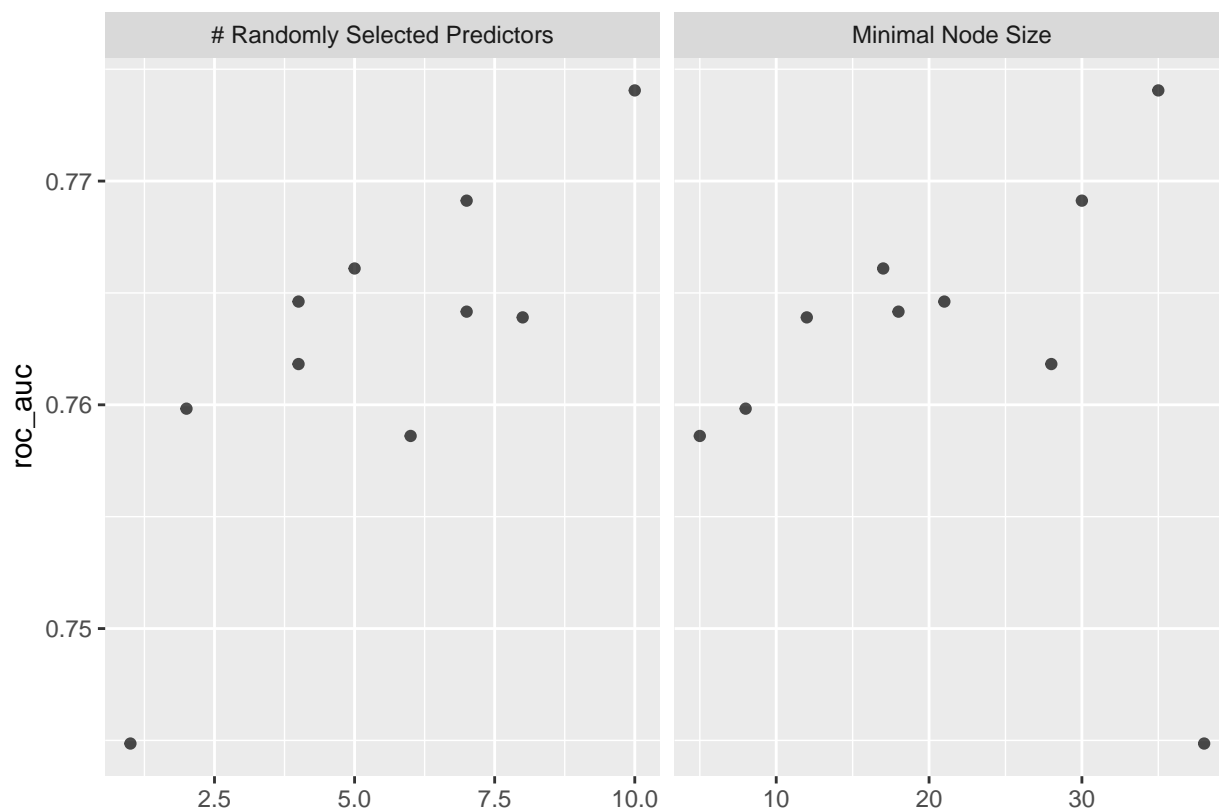
```
if (tree_auc$auc > 0.55){
  final_tree_fit %>%
    extract_fit_parsnip() %>%
    vip()
}
```



Minutes to run: 1.007

Random Forest

```
rf_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())  
  
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")

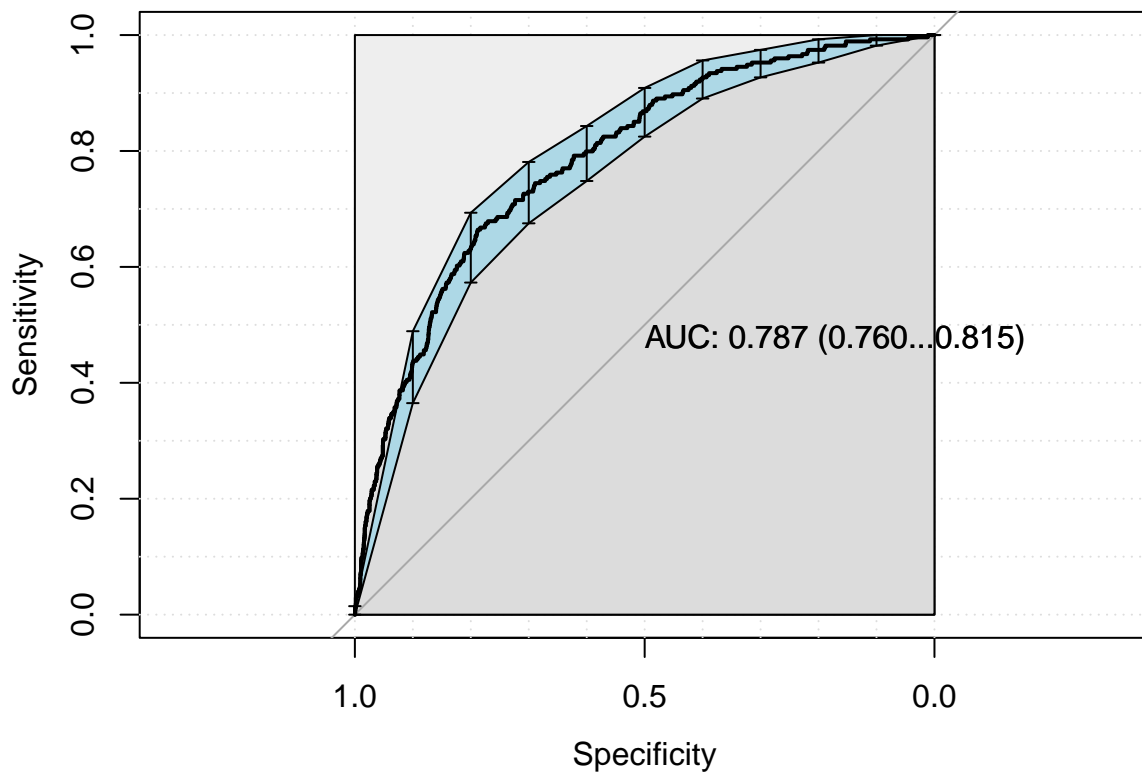
best_rf <- rf_tune %>%
  select_best("roc_auc")

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: 0.943

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

```
# sum_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())
#
# sum_spec <-
#   svm_rbf(cost = tune(), rbf_sigma = tune()) %>%
#   set_mode("classification") %>%
#   set_engine("kernlab")
#
# sum_grid <- grid_latin_hypercube(cost(),
#                                   rbf_sigma(),
#                                   size = grid_size)
#
# sum_workflow <-
#   workflow() %>%
#   add_recipe(sum_recipe) %>%
#   add_model(sum_spec)
#
# sum_tune <-
#   sum_workflow %>%
#   tune_grid(resamples = df_folds,
#             grid = 5)
#
# sum_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

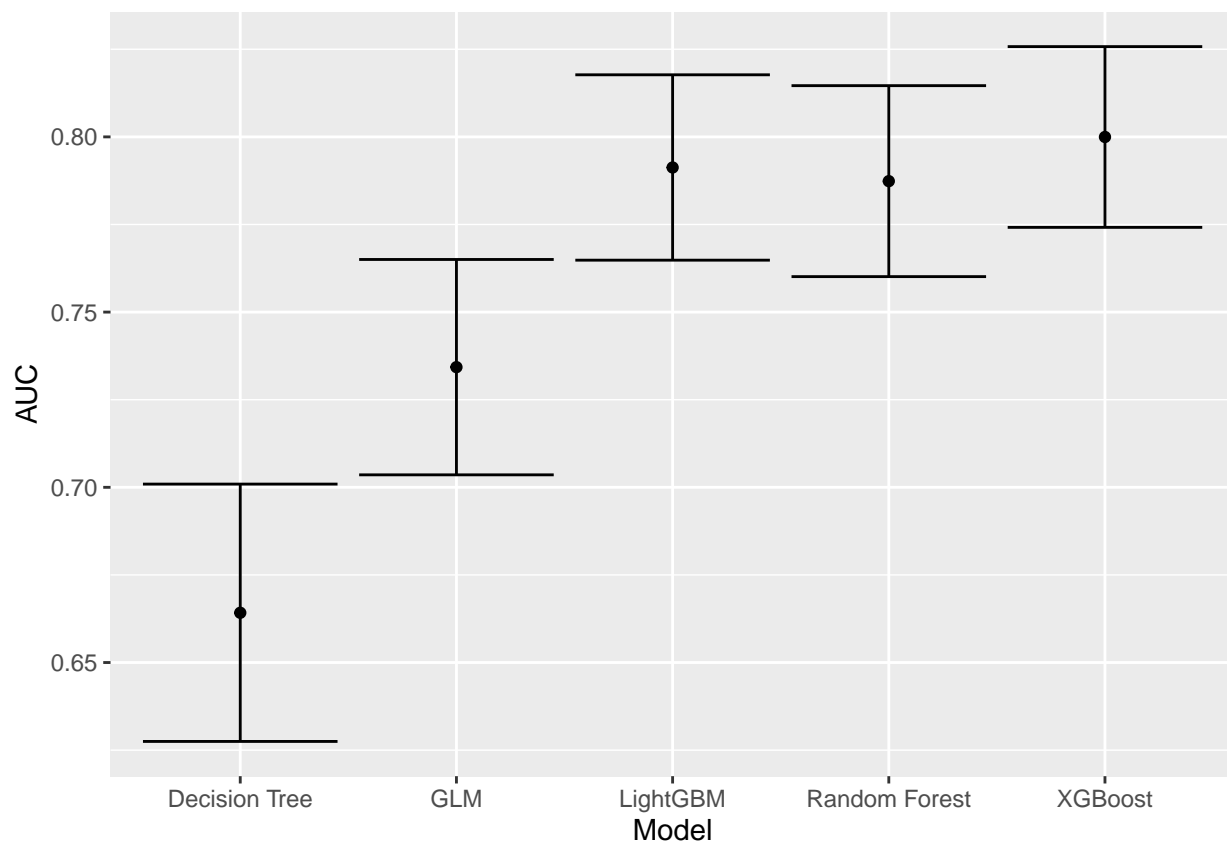
Models Comparison

```

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: 0.002