

Model Selection - readmission_30d

Eduardo Yuki Yada

Imports

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

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

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

Minutes to run: 0.005

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. education_level
## 02. underlying_heart_disease
## 03. heart_disease
## 04. nyha_basal
## 05. prior_mi
## 06. heart_failure
## 07. transplant
## 08. endocardites
## 09. hemodialysis
## 10. comorbidities_count
## 11. procedure_type_1
## 12. reop_type_1
## 13. procedure_type_new
## 14. cied_final_1
## 15. cied_final_group_1
## 16. admission_pre_t0_count
## 17. admission_pre_t0_180d
## 18. icu_t0
## 19. dialysis_t0
## 20. admission_t0_emergency
## 21. aco
## 22. antiarritmico
## 23. betabloqueador
## 24. ieca_bra
## 25. dva
## 26. digoxina
## 27. estatina
## 28. diuretico
## 29. vasodilatador
## 30. insuf_cardiaca
## 31. espironolactona
## 32. bloq_calcio
## 33. antiplaquetario_ev
## 34. insulina
## 35. anticonvulsivante
## 36. psicofarmacos
## 37. antifungico
## 38. antiviral
## 39. classe_meds_qtde
## 40. meds_cardiovasc_qtde
## 41. meds_antimicrobianos
## 42. cec
## 43. transplante_cardiaco
## 44. outros_proced_cirurgicos
## 45. icp
## 46. intervencao_cv
## 47. cateterismo
## 48. eletrofisiologia
## 49. cateter_venoso_central
## 50. proced_invasivos_qtde
## 51. cve_desf
## 52. transfusao
## 53. equipe_multiprof
## 54. ecg
## 55. holter

```

```
## 56. metodos_graficos_qtde
## 57. laboratorio
## 58. cultura
## 59. analises_clinicas_qtde
## 60. citologia
## 61. biopsia
## 62. histopatologia_qtde
## 63. angio_rm
## 64. angio_tc
## 65. cintilografia
## 66. ecocardiograma
## 67. endoscopia
## 68. flebografia
## 69. pet_ct
## 70. ultrassom
## 71. tomografia
## 72. radiografia
## 73. ressonancia
## 74. exames_imagem_qtde
## 75. bic
## 76. 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: 0.004

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)

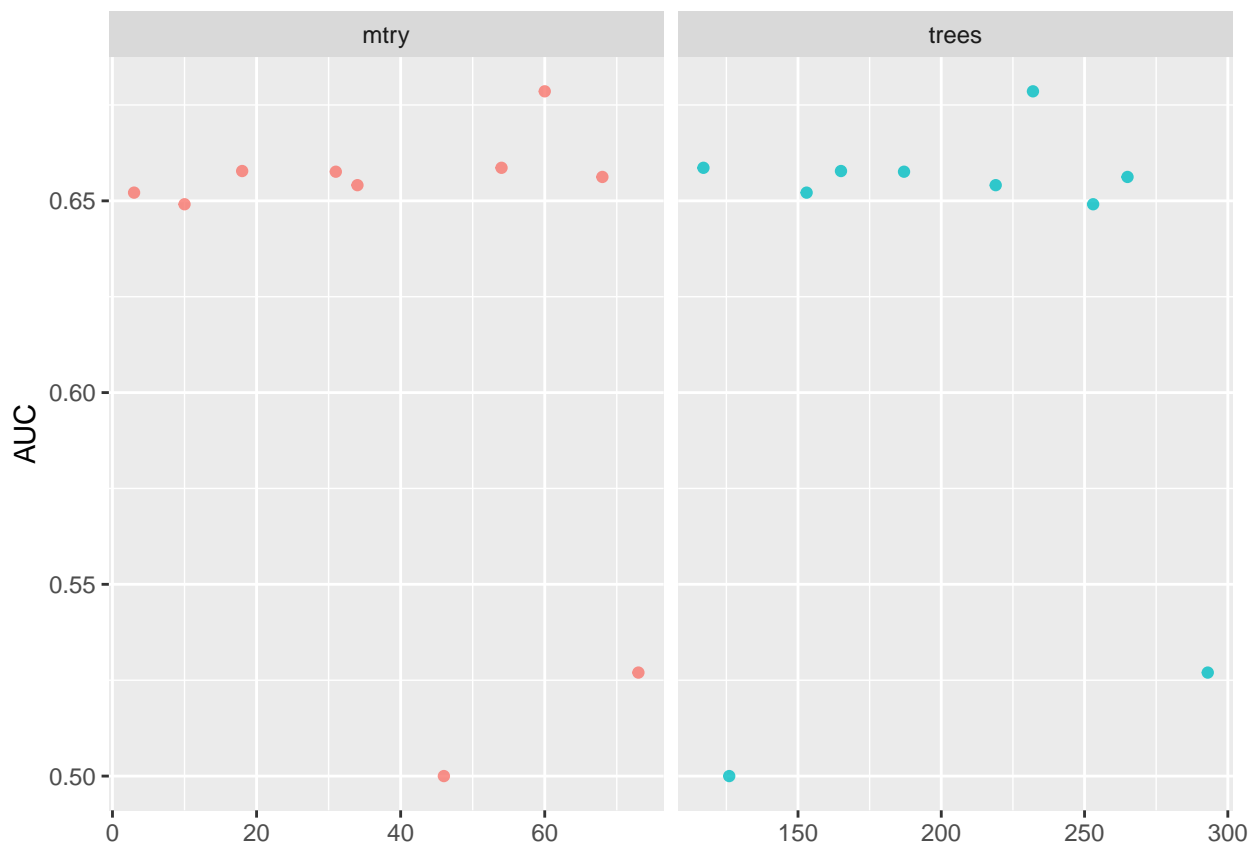
xgboost_tune %>%
  show_best("roc_auc")

## # A tibble: 5 x 12
##   mtry trees min_n tree_depth learn_r~1 loss_~2 .metric .esti~3 mean
##   <int> <int> <int>      <int>      <dbl>   <dbl> <chr>   <chr>   <dbl>
## 1     60   232    14         3 0.0108   7.46e-4 roc_auc binary  0.679
## 2     54   117    13         8 0.0789   2.62e+0 roc_auc binary  0.659
## 3     18   165    31         2 0.0000134 1.41e-1 roc_auc binary  0.658
## 4     31   187     6        12 0.0000653 5.41e-9 roc_auc binary  0.658
## 5     68   265    20        15 0.00108   3.14e-5 roc_auc binary  0.656
## # ... 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")

```

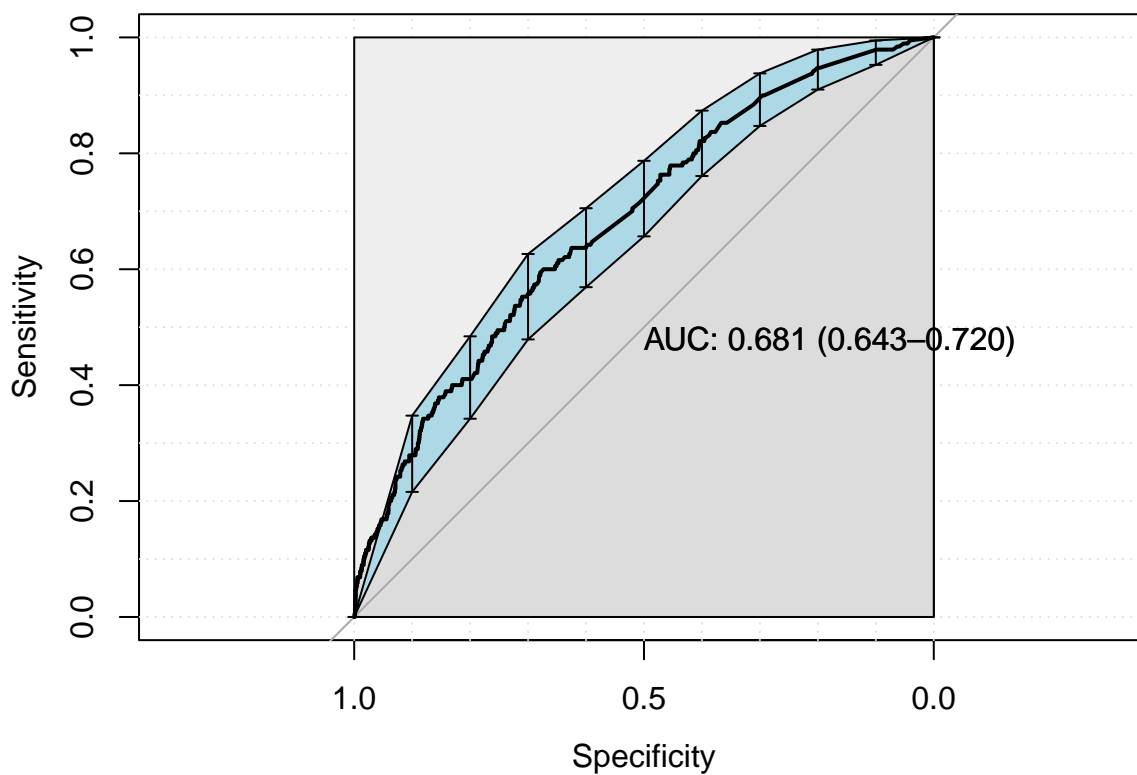


```
final_xgboost_workflow <-
  xgboost_workflow %>%
  finalize_workflow(best_xgboost)

last_xgboost_fit <-
  final_xgboost_workflow %>%
  last_fit(df_split)

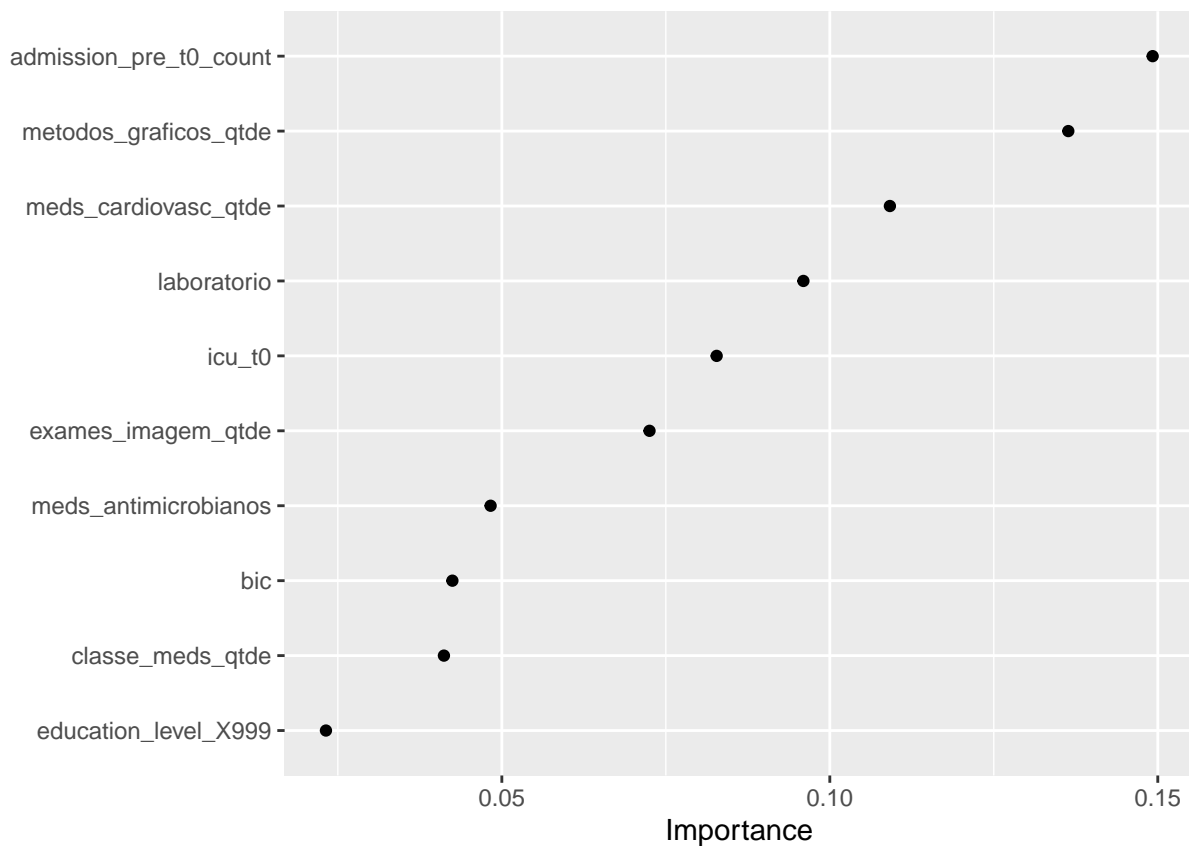
final_xgboost_fit <- extract_workflow(last_xgboost_fit)

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



```
## Confusion Matrix and Statistics
##
##
## test_predictions_class    0    1
##                0 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")
```



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

Minutes to run: 2.349

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)

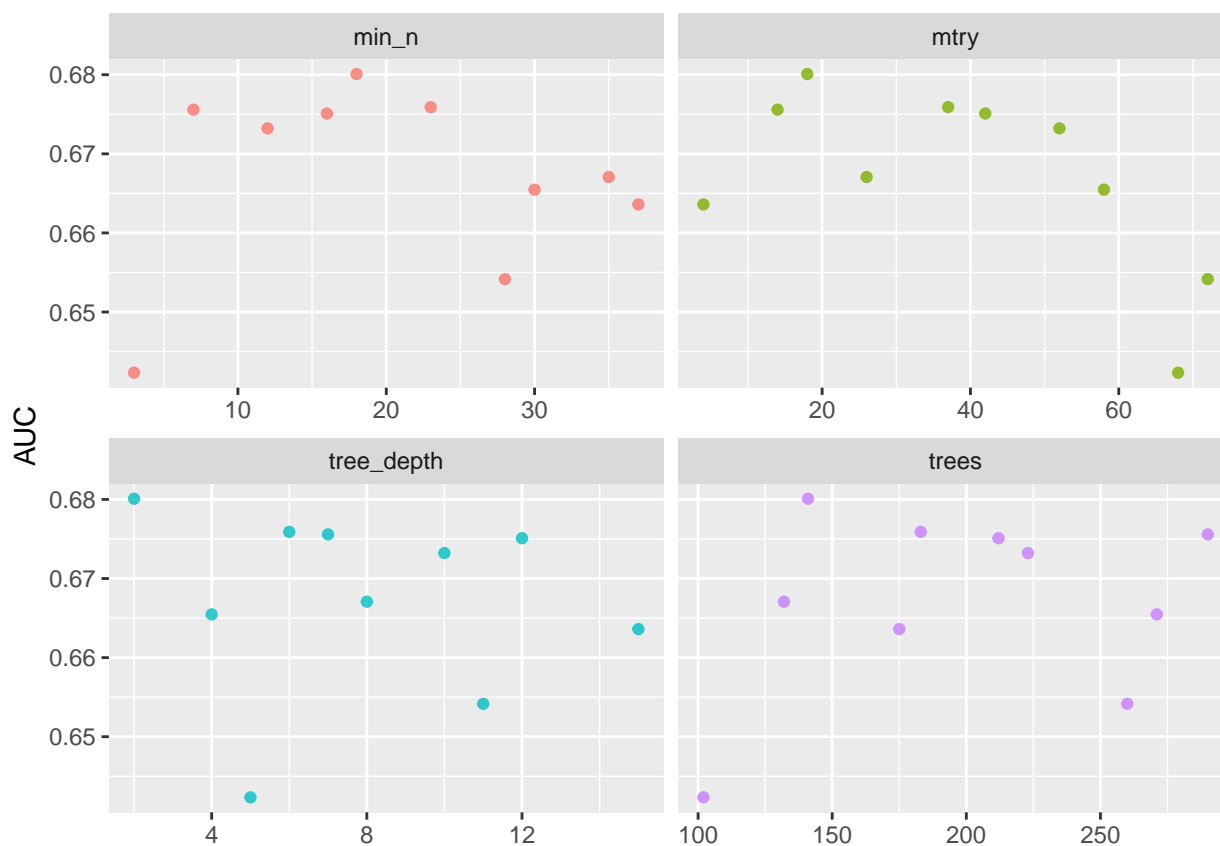
lightgbm_tune %>%
  show_best("roc_auc")

## # 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     18    141    18         2  3.41e- 7  1.19e-9 roc_auc binary  0.680
## 2     37    183    23         6  2.29e-10 6.65e-5 roc_auc binary  0.676
## 3     14    290     7         7  6.20e- 5  2.97e-5 roc_auc binary  0.676
## 4     42    212    16        12  2.28e- 9  1.60e-9 roc_auc binary  0.675
## 5     52    223    12        10  5.42e- 2  6.17e+0 roc_auc binary  0.673
## # ... 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")

```

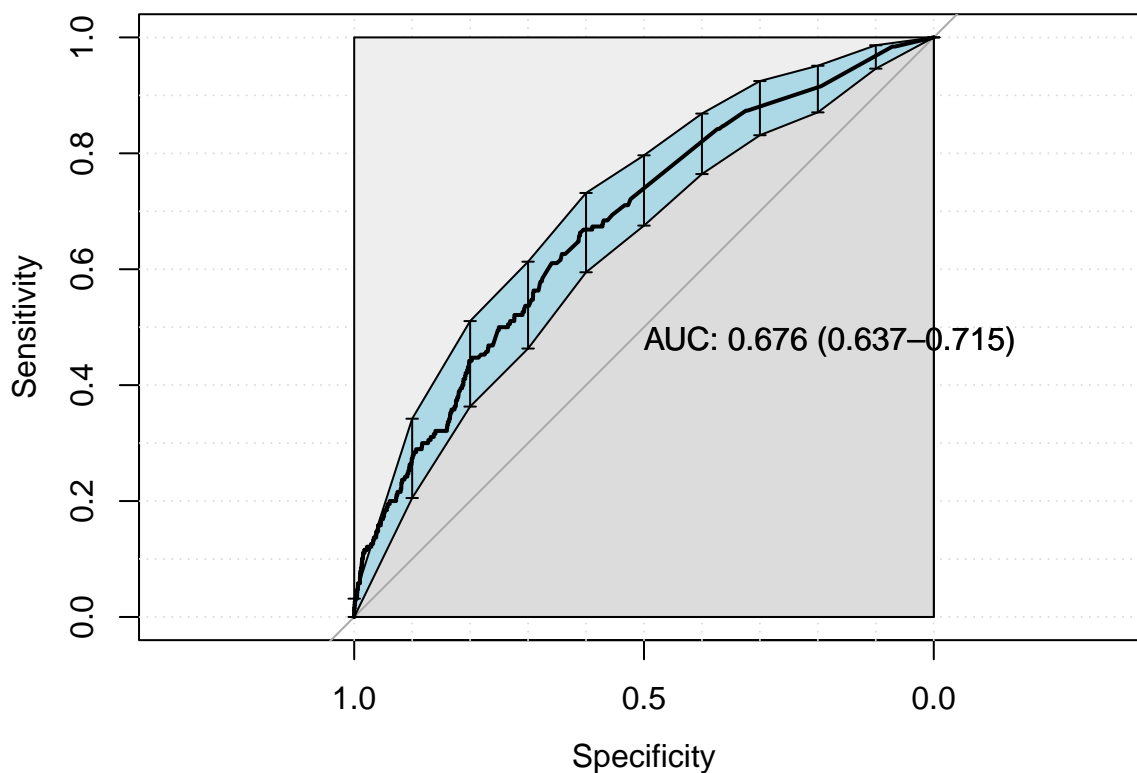


```
final_lightgbm_workflow <-
  lightgbm_workflow %>%
  finalize_workflow(best_lightgbm)

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

```

Minutes to run: 1.155

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())

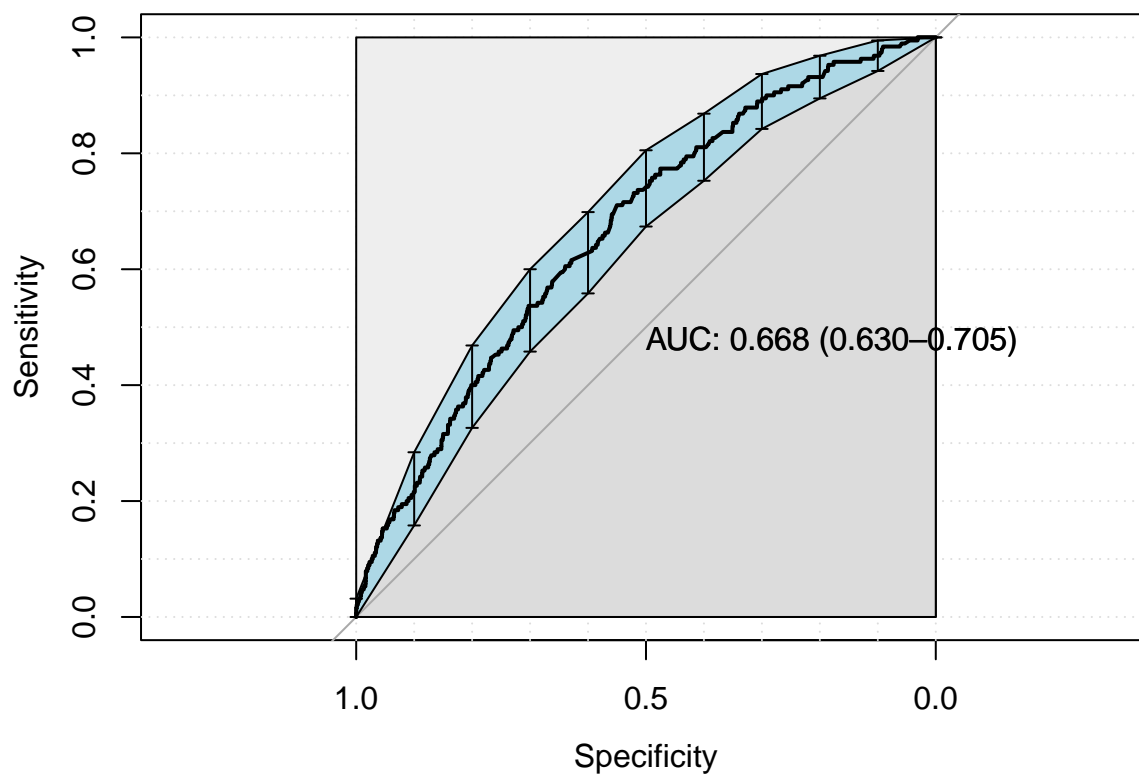
glmnet_spec <-
  logistic_reg(penalty = 0) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

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 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
```

##

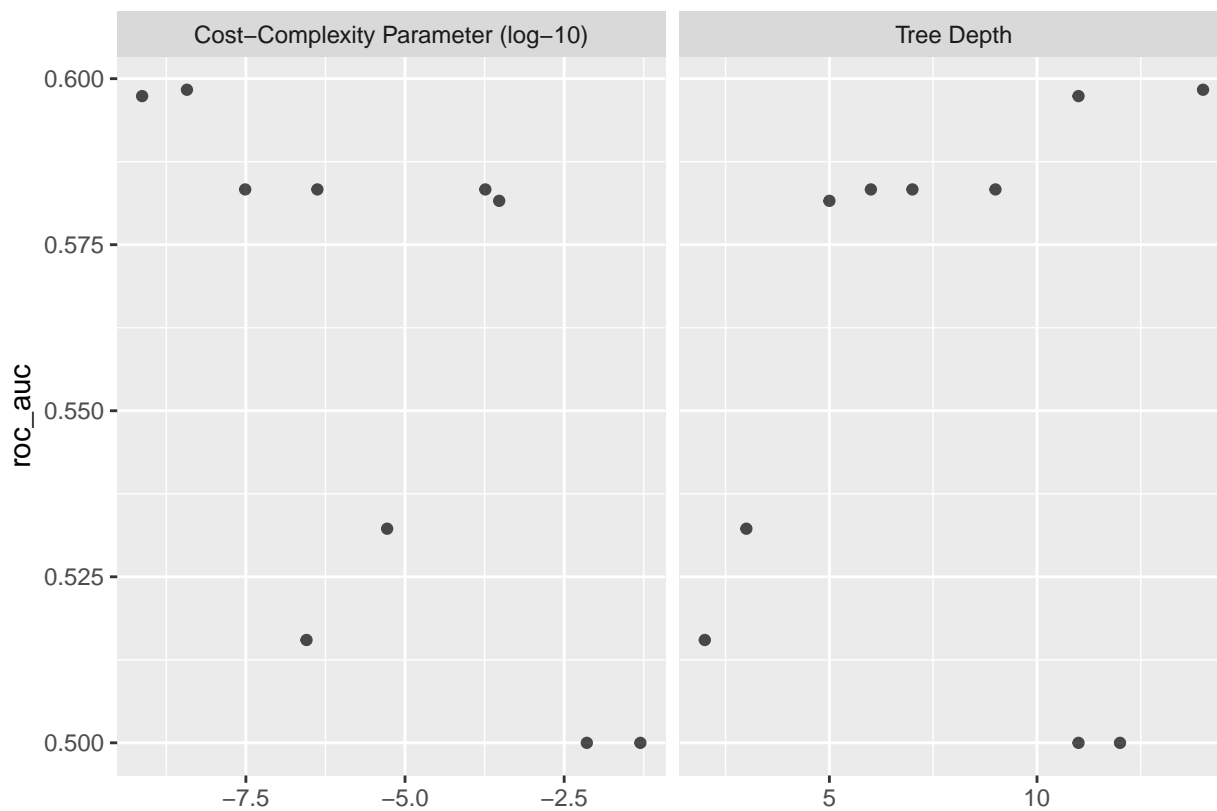
```
##           'Positive' Class : 0
```

##

Minutes to run: 0.1

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")

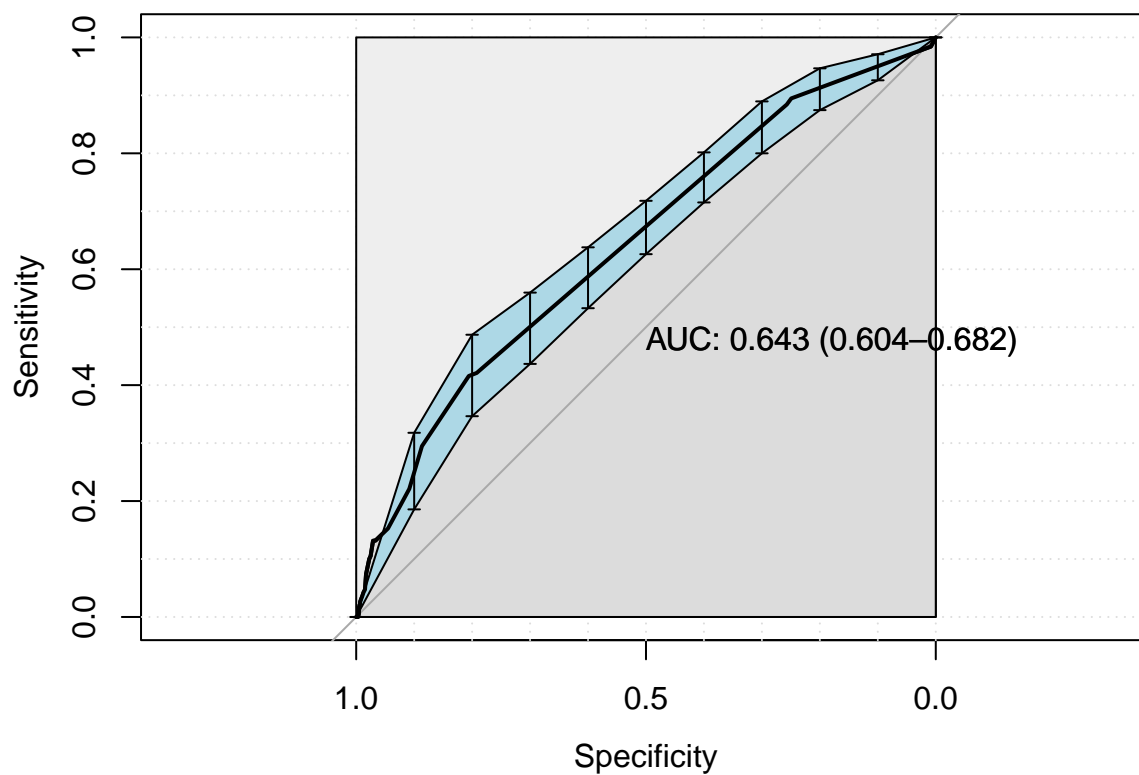
best_tree <- tree_tune %>%
  select_best("roc_auc")

final_tree_workflow <-
  tree_workflow %>%
  finalize_workflow(best_tree)

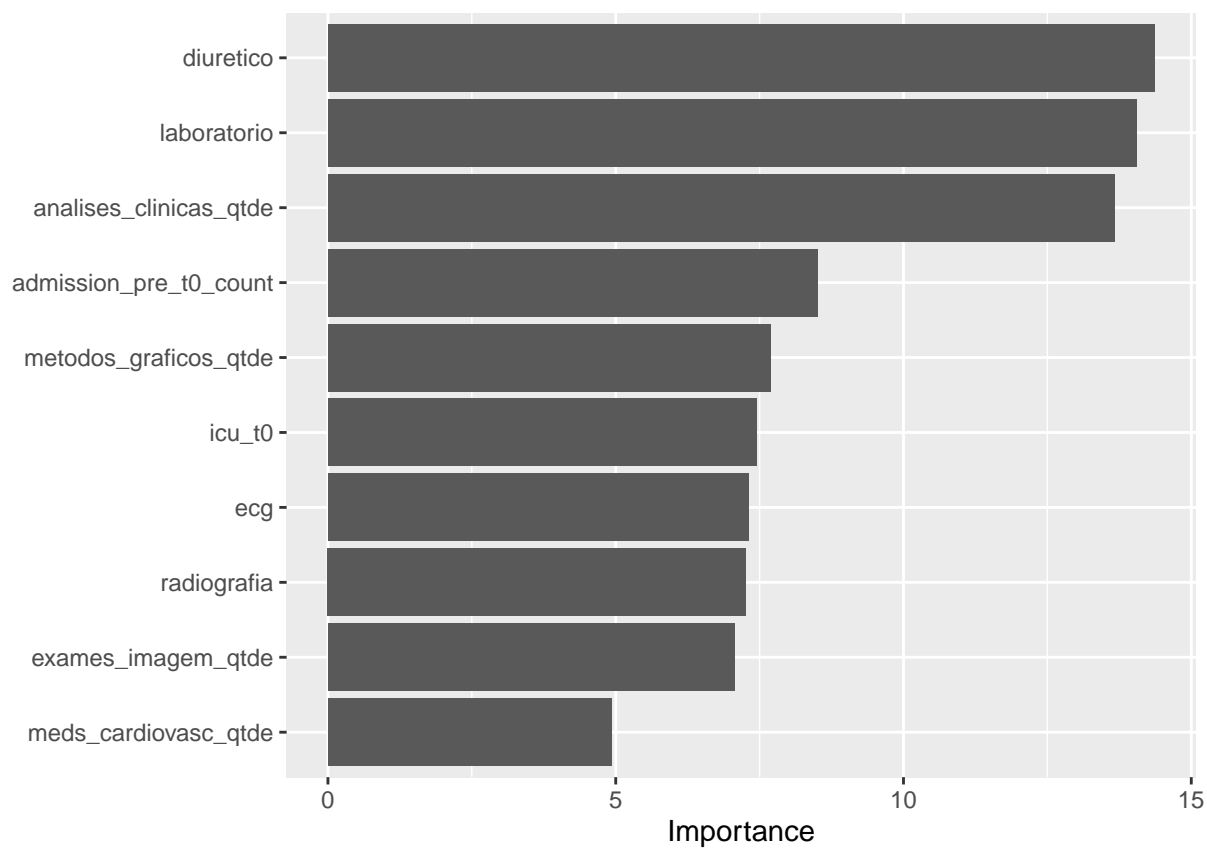
last_tree_fit <-
  final_tree_workflow %>%
  last_fit(df_split)

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()
}
```

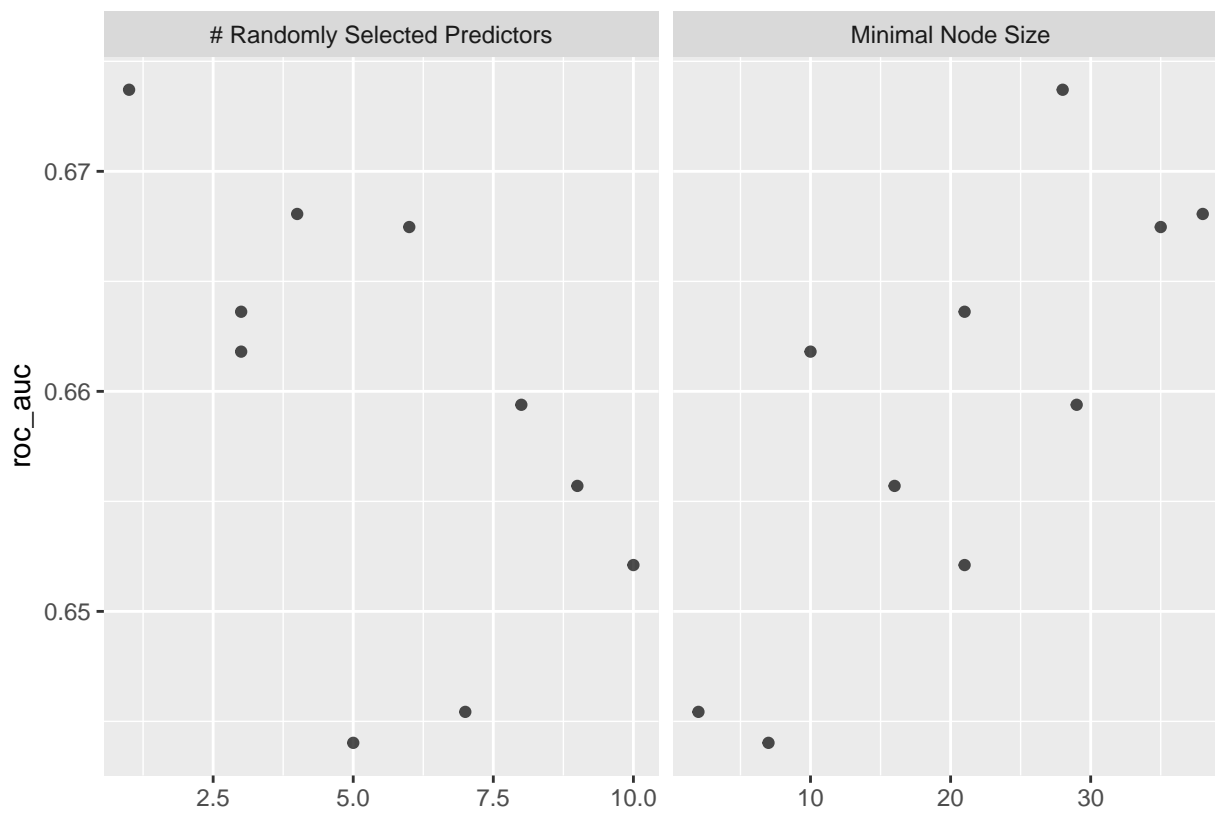


0.855

Minutes to run:

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")

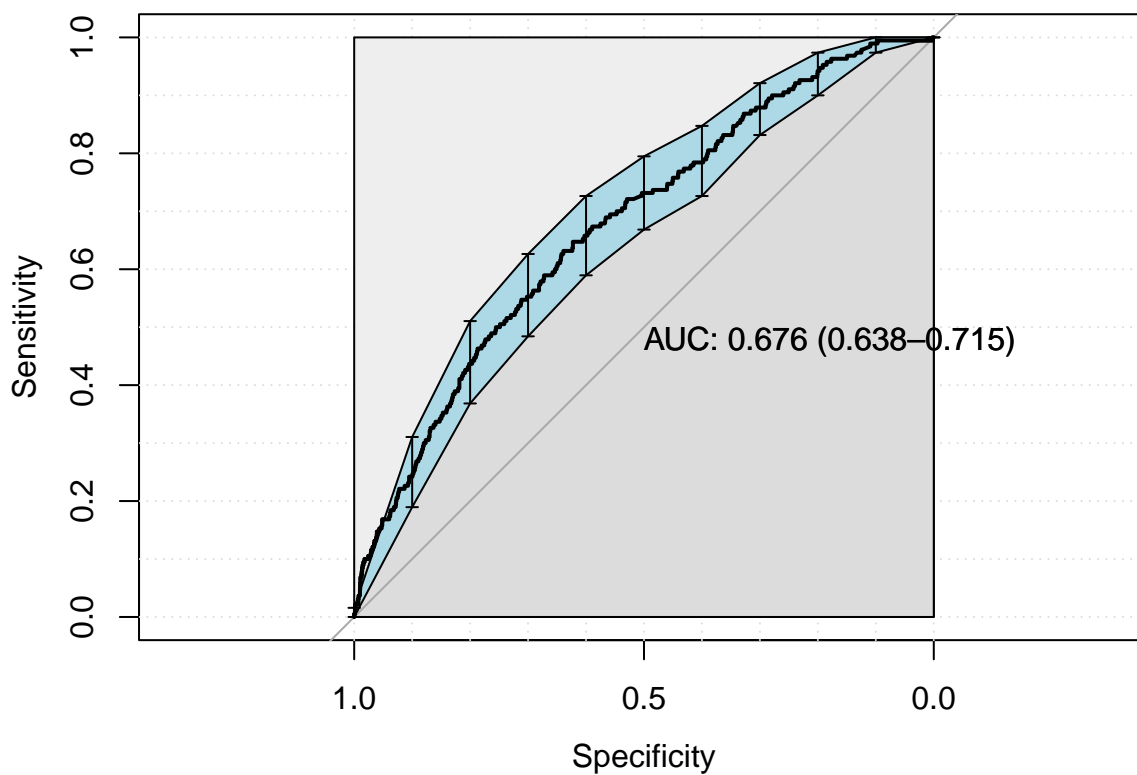
best_rf <- rf_tune %>%
  select_best("roc_auc")

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)
```



0.798

Minutes to run:

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()
#

```

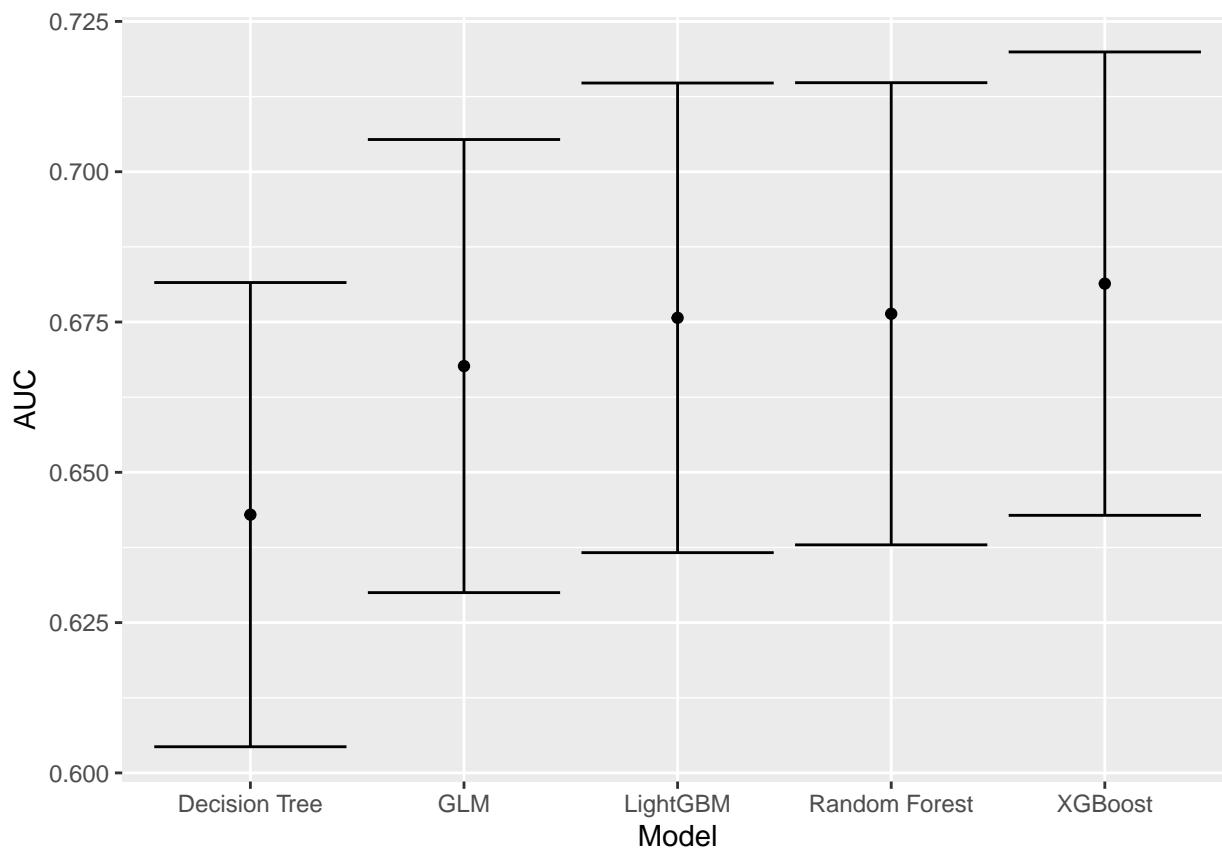
```
# autoplot(sum_tune, metric = "roc_auc")
#
# sum_tune %>%
#   show_best("roc_auc")
#
# best_sum <- sum_tune %>%
#   select_best("roc_auc")
#
# final_sum_workflow <-
#   sum_workflow %>%
#   finalize_workflow(best_sum)
#
# last_sum_fit <-
#   final_sum_workflow %>%
#   last_fit(df_split)
#
# final_sum_fit <- 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/model_selection/performance/%s.RData", outcome_column))
```

Minutes to run: 0.002