Model Selection - death_30days

Eduardo Yuki Yada

Global parameters

```
k = 5 # Number of folds for cross validation
grid_size = 15 # Number of parameter combination to tune on each model
```

Minutes to run: 0

Imports

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

source("aux_functions.R")
```

Minutes to run: 0.02

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

```
dir.create(file.path("./auxiliar/model_selection/hyperparameters/"),
           showWarnings = FALSE,
           recursive = TRUE)

dir.create(file.path("./auxiliar/model_selection/performance/"),
           showWarnings = FALSE,
           recursive = TRUE)
```

Minutes to run: 0

Eligible features

```
eligible_columns = df_names %>%
  filter(momento.aquisicao == 'Admissão t0') %>%
  .$variable.name

exception_columns = c('death_intraop', 'death_intraop_1', 'disch_outcomes_t0')

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,
                       'radiografia', # com exames_imagem_qtde
                       'ecg' # com metodos_graficos_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. valvopathy
## 12. diabetes
## 13. renal_failure
## 14. hemodialysis
## 15. cancer
## 16. comorbidities_count
## 17. procedure_type_1
## 18. reop_type_1
## 19. procedure_type_new
## 20. cied_final_1
## 21. cied_final_group_1
## 22. admission_pre_t0_count
## 23. admission_pre_t0_180d
## 24. year_adm_t0
## 25. icu_t0
## 26. antiarritmico
## 27. antihipertensivo
## 28. betabloqueador
## 29. dva
## 30. diuretico
## 31. vasodilatador
```

```
## 32. espironolactona
## 33. antiplaquetario_ev
## 34. insulina
## 35. psicofarmacos
## 36. antifungico
## 37. classe_meds_qtde
## 38. meds_cardiovasc_qtde
## 39. meds_antimicrobianos
## 40. vni
## 41. ventilacao_mecanica
## 42. intervencao_cv
## 43. cateter_venoso_central
## 44. proced_invasivos_qtde
## 45. transfusao
## 46. interconsulta
## 47. equipe_multiprof
## 48. holter
## 49. metodos_graficos_qtde
## 50. laboratorio
## 51. cultura
## 52. analises_clinicas_qtde
## 53. citologia
## 54. histopatologia_qtde
## 55. angio_tc
## 56. angiografia
## 57. cintilografia
## 58. ecocardiograma
## 59. flebografia
## 60. ultrassom
## 61. tomografia
## 62. ressonancia
## 63. exames_imagem_qtde
## 64. bic
## 65. hospital_stay
```

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)))

df_folds <- vfold_cv(df_train, v = k,
                     strata = all_of(outcome_column))
```

Minutes to run: 0.001

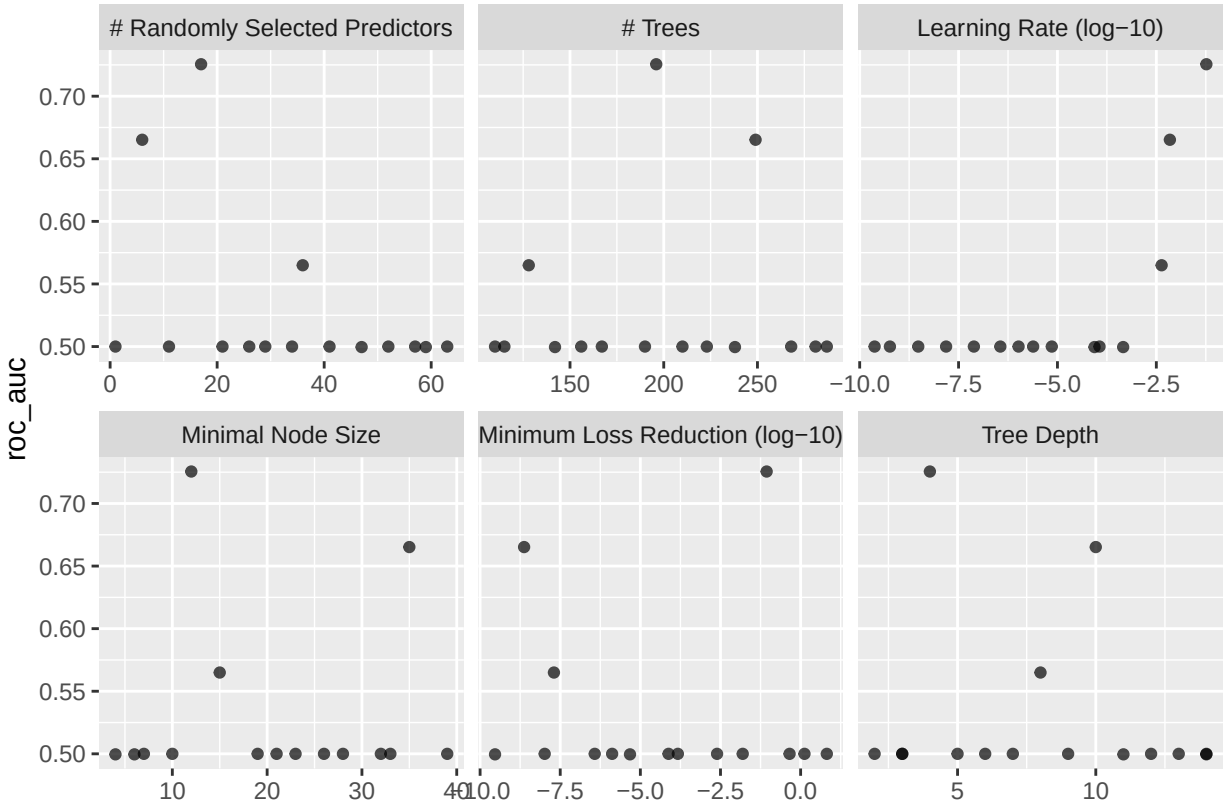
Boosted Tree (XGBoost)

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

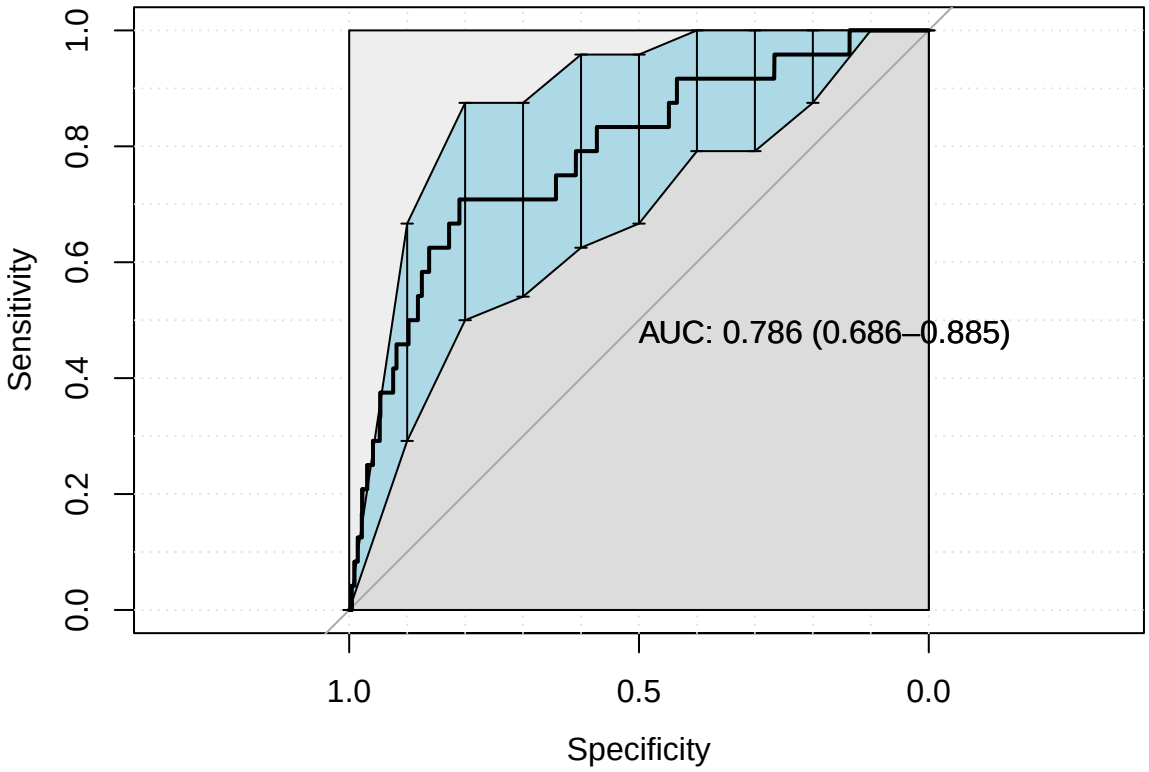
```
  step_unknown(all_nominal_predictors()) %>%
  step_other(all_nominal_predictors(), threshold = 0.05, other = ".merged") %>%
  step_dummy(all_nominal_predictors())

xgboost_spec <- boost_tree(
  mtry = tune(),
  trees = tune(),
  min_n = tune(),
  tree_depth = tune(),
  learn_rate = tune(),
  loss_reduction = tune()
) %>%
  set_engine("xgboost",
             nthread = 8) %>%
  set_mode("classification")

xgboost_grid <- grid_latin_hypercube(
  finalize(mtry(), df_train),
  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 .estima~1  mean     n std_err .config
##   <int> <int> <int>      <int>        <dbl>          <dbl> <chr>   <chr>     <dbl> <int>   <dbl> <chr>
## 1    17   196    12          4 0.0582        0.0872        roc_auc binary    0.726     5  0.0440 Prepro~
## 2     6   249    35         10 0.00696       0.00000000236 roc_auc binary    0.665     5  0.0294 Prepro~
## 3    36   128    15          8 0.00430       0.0000000201  roc_auc binary    0.565     5  0.0241 Prepro~
## 4     1   167     7          9 0.0000000153  0.0156        roc_auc binary    0.5       5  0      Prepro~
## 5    11   190    26          3 0.000000360   0.452         roc_auc binary    0.5       5  0      Prepro~
## # ... with abbreviated variable name 1: .estimator
```

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

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

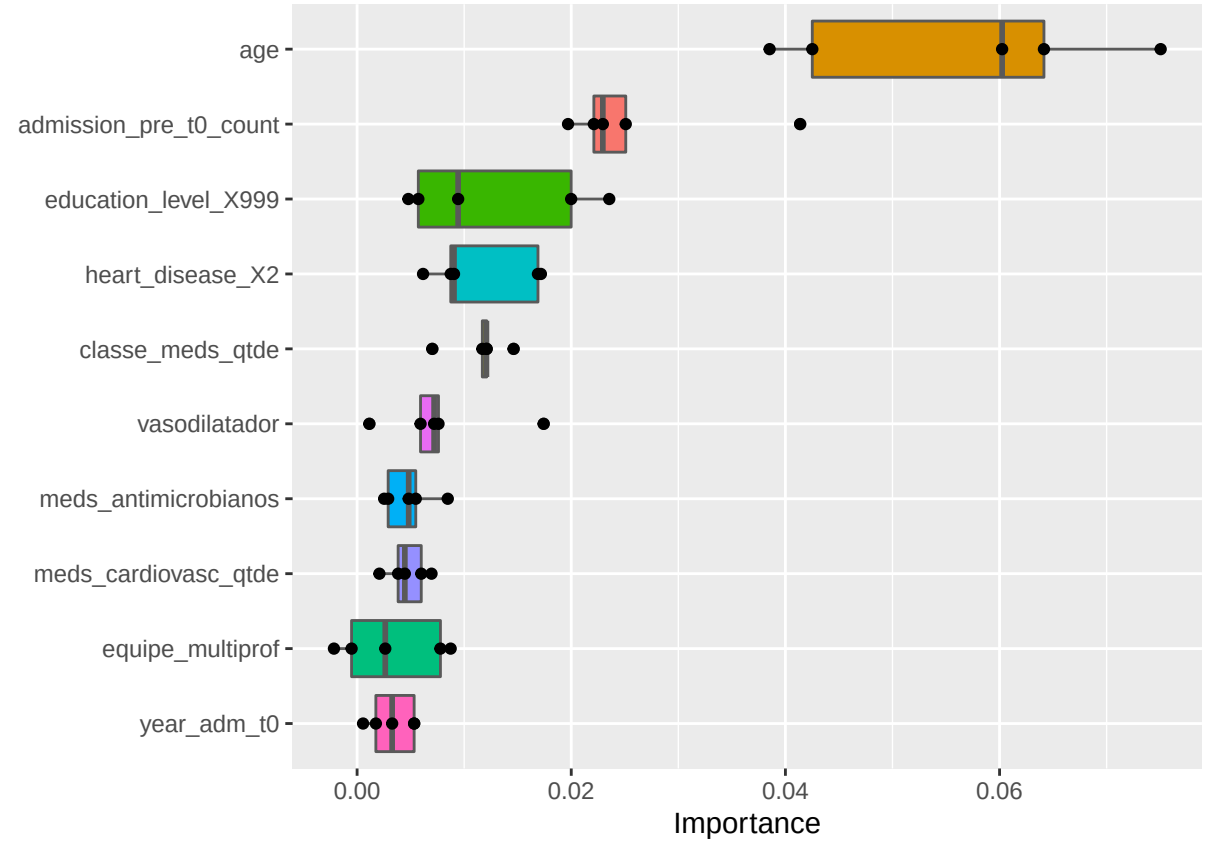



```
final_xgboost_workflow <-
  xgboost_workflow %>%
  finalize_workflow(best_xgboost)

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)
```




```
## [1] "Optimal Threshold: 0.01"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 3811    7
##    1  895   17
##
##                Accuracy : 0.8093
##                  95% CI : (0.7978, 0.8204)
##     No Information Rate : 0.9949
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0267
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.80982
##             Specificity : 0.70833
##          Pos Pred Value : 0.99817
##          Neg Pred Value : 0.01864
##              Prevalence : 0.99493
##          Detection Rate : 0.80571
##    Detection Prevalence : 0.80719
##       Balanced Accuracy : 0.75908
##
##        'Positive' Class : 0
##
```

```
extract_vip(final_xgboost_fit, pred_wrapper = predict,
            reference_class = "0")
```




```
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: 1.22

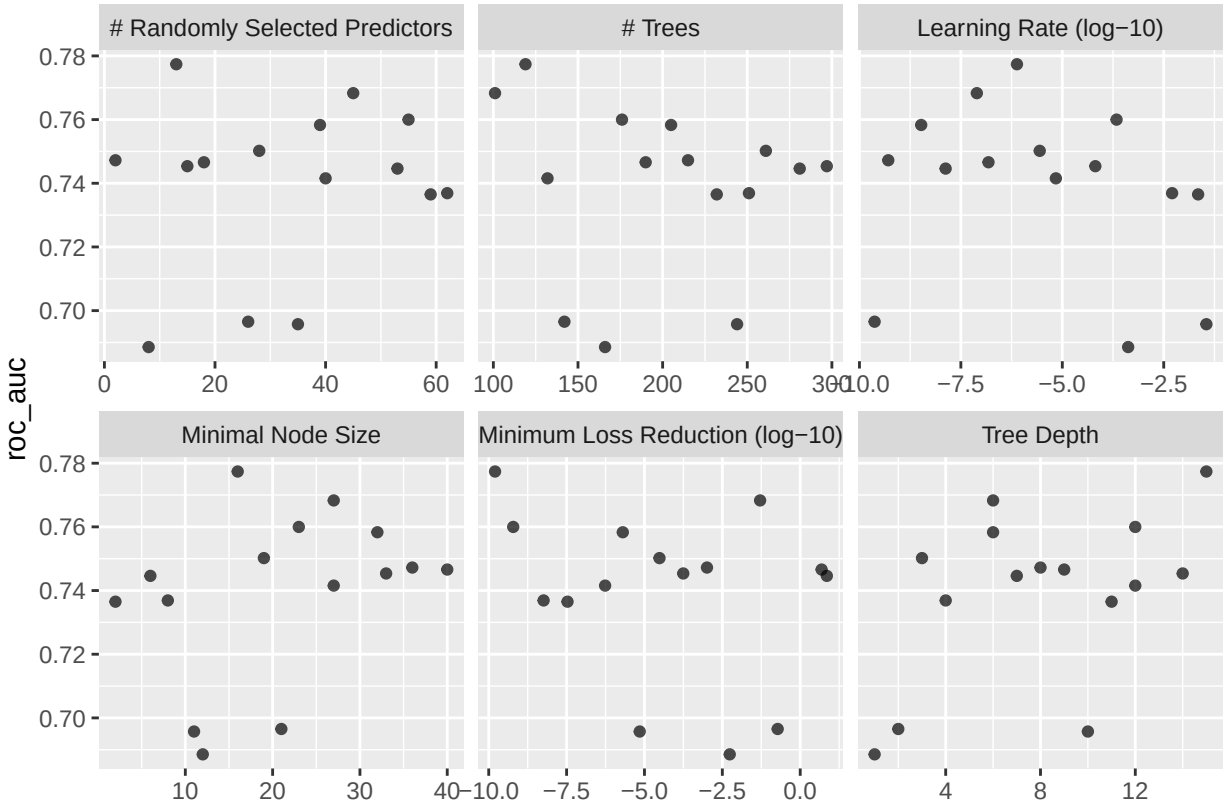
Boosted Tree (LightGBM)

```
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_dummy(all_nominal_predictors())

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

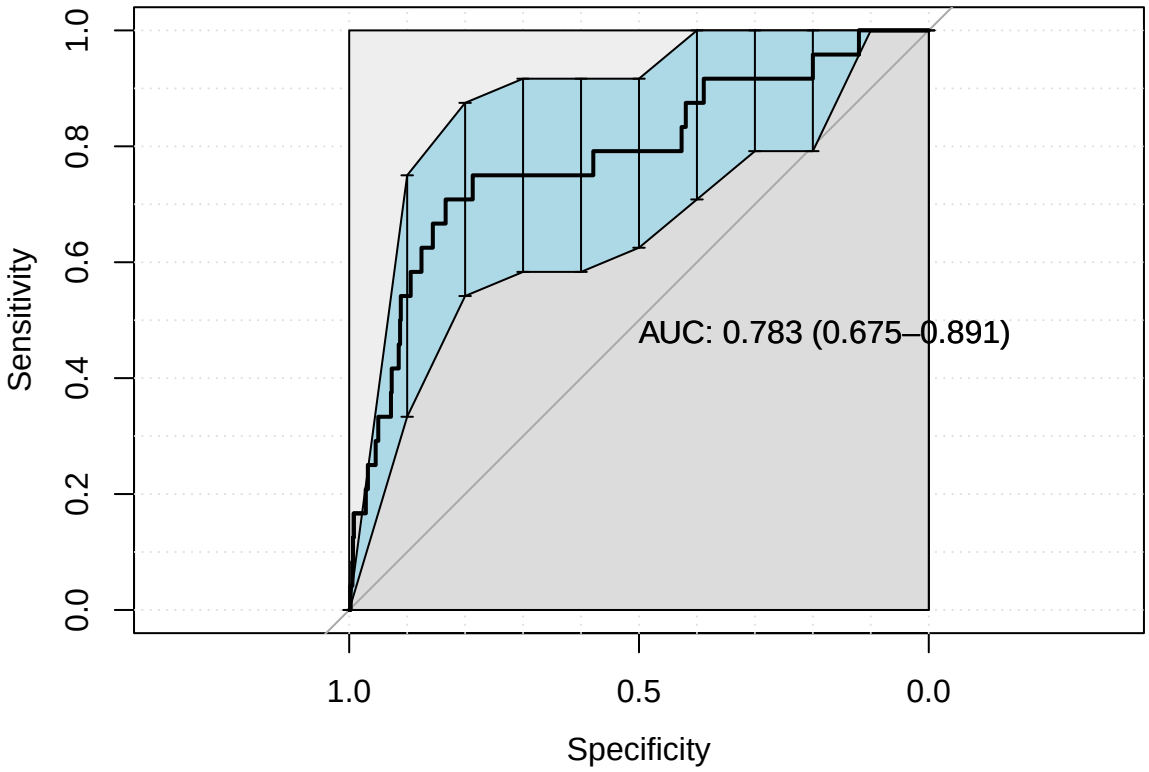
```
  set_engine("lightgbm",
             nthread = 8) %>%
  set_mode("classification")

lightgbm_grid <- grid_latin_hypercube(
  finalize(mtry(), df_train),
  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 .estim~1  mean     n std_err .config
##   <int> <int> <int>      <int>         <dbl>          <dbl> <chr>   <chr>    <dbl> <int>   <dbl> <chr>
## 1    13   119    16         15 0.000000763         1.59e-10 roc_auc binary   0.777     5  0.0270 Prepro~
## 2    45   101    27          6 0.0000000783        5.16e- 2 roc_auc binary   0.768     5  0.0483 Prepro~
## 3    55   176    23         12 0.000217            6.12e-10 roc_auc binary   0.760     5  0.0403 Prepro~
## 4    39   205    32          6 0.00000000333       2.02e- 6 roc_auc binary   0.758     5  0.0419 Prepro~
## 5    28   261    19          3 0.00000278          3.03e- 5 roc_auc binary   0.750     5  0.0394 Prepro~
## # ... with abbreviated variable name 1: .estimator
```

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

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

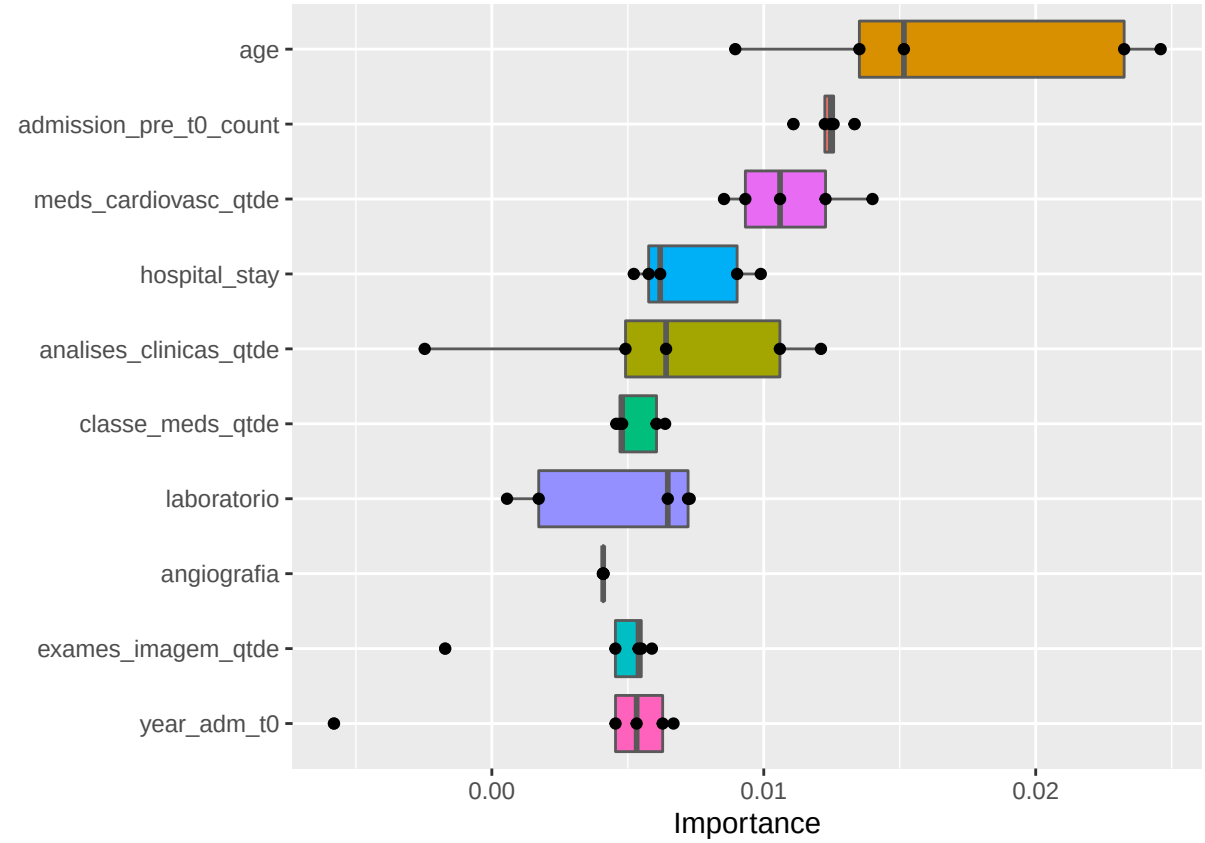



```
final_lightgbm_workflow <-
  lightgbm_workflow %>%
  finalize_workflow(best_lightgbm)

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)
```




```
## [1] "Optimal Threshold: 0.00"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 3923    7
##    1  783   17
##
##                Accuracy : 0.833
##                  95% CI : (0.822, 0.8435)
##     No Information Rate : 0.9949
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0317
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.83362
##             Specificity : 0.70833
##          Pos Pred Value : 0.99822
##          Neg Pred Value : 0.02125
##              Prevalence : 0.99493
##          Detection Rate : 0.82939
##    Detection Prevalence : 0.83087
##       Balanced Accuracy : 0.77097
##
##        'Positive' Class : 0
##
```

```
pfun_lightgbm <- function(object, newdata) predict(object, data = newdata)
extract_vip(final_lightgbm_fit, pred_wrapper = pfun_lightgbm,
            reference_class = "1")
```



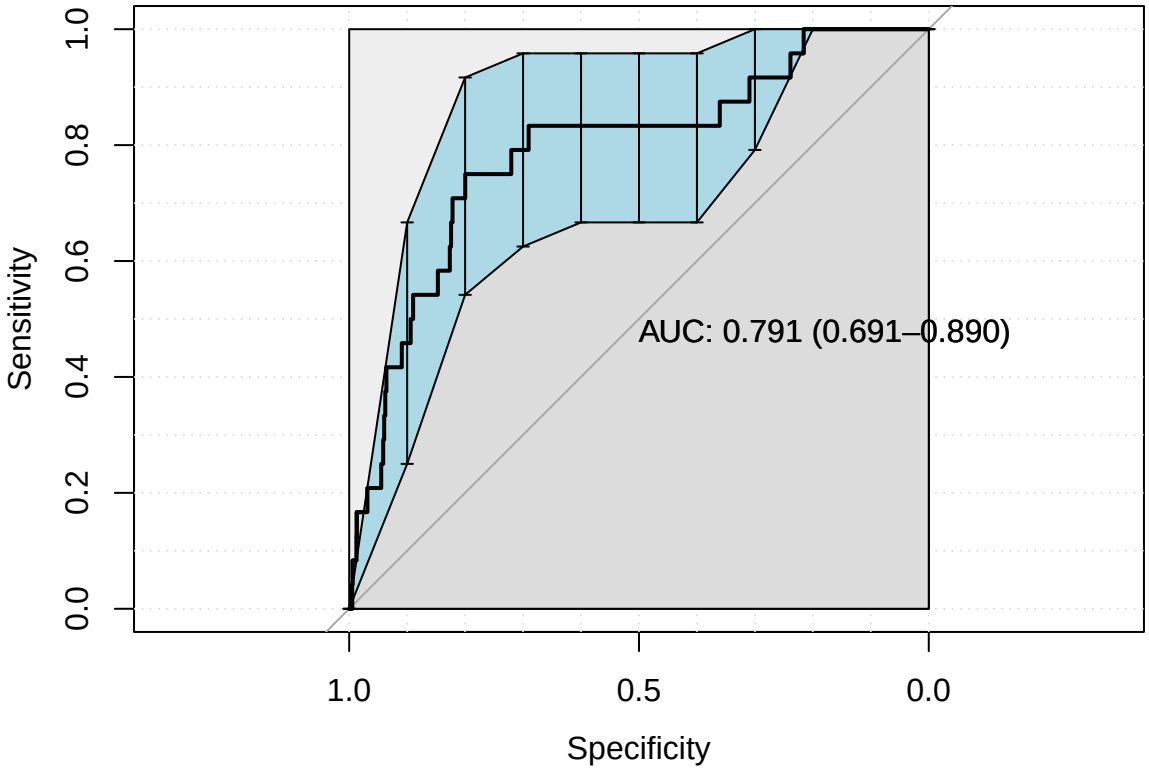

```
lightgbm_parameters <- lightgbm_tune %>%
  show_best("roc_auc", n = 1) %>%
  select(trees, mtry, min_n, tree_depth, learn_rate, loss_reduction) %>%
  as.list

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

Minutes to run: 2.209

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()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_numeric_predictors())

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

glmnet_workflow <-
  workflow() %>%
```

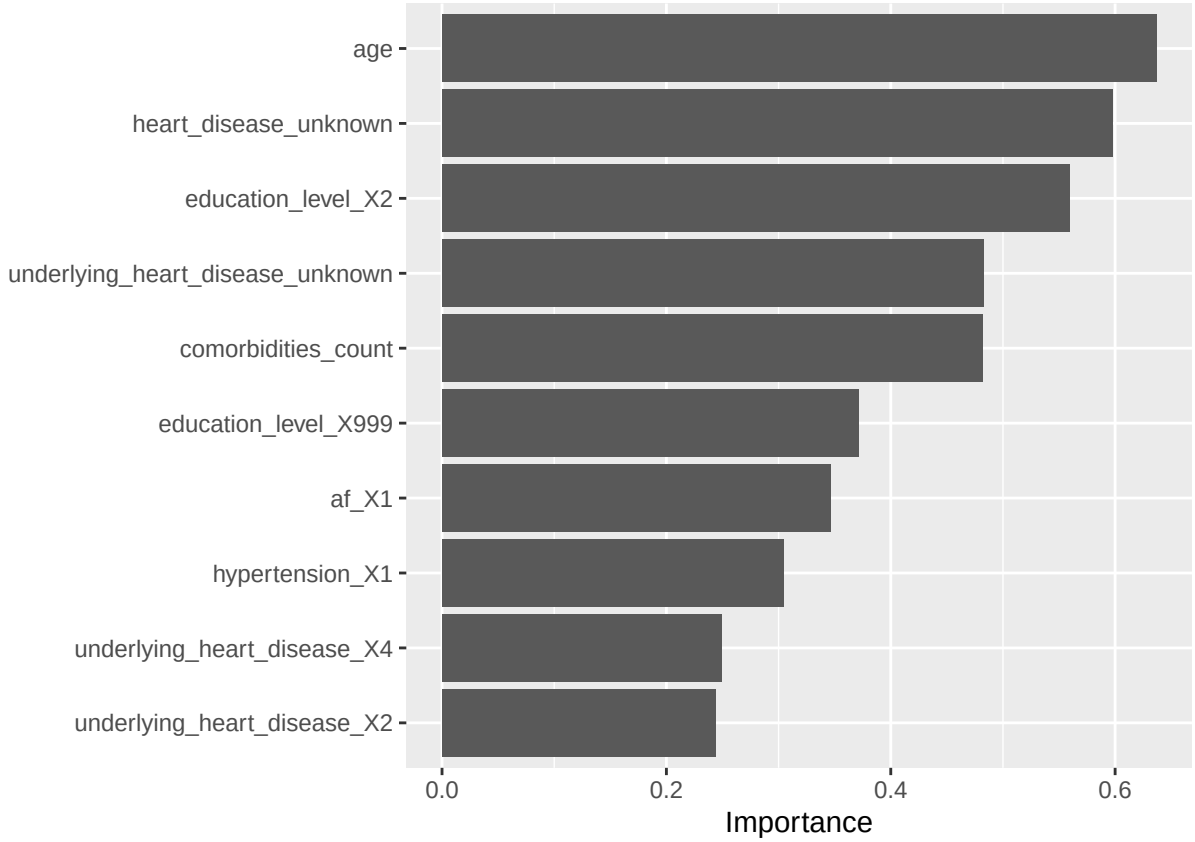
```
  add_recipe(glmnet_recipe) %>%
  add_model(glmnet_spec)

glm_fit <- glmnet_workflow %>%
  fit(df_train)

glmnet_auc <- validation(glm_fit, df_test)
```



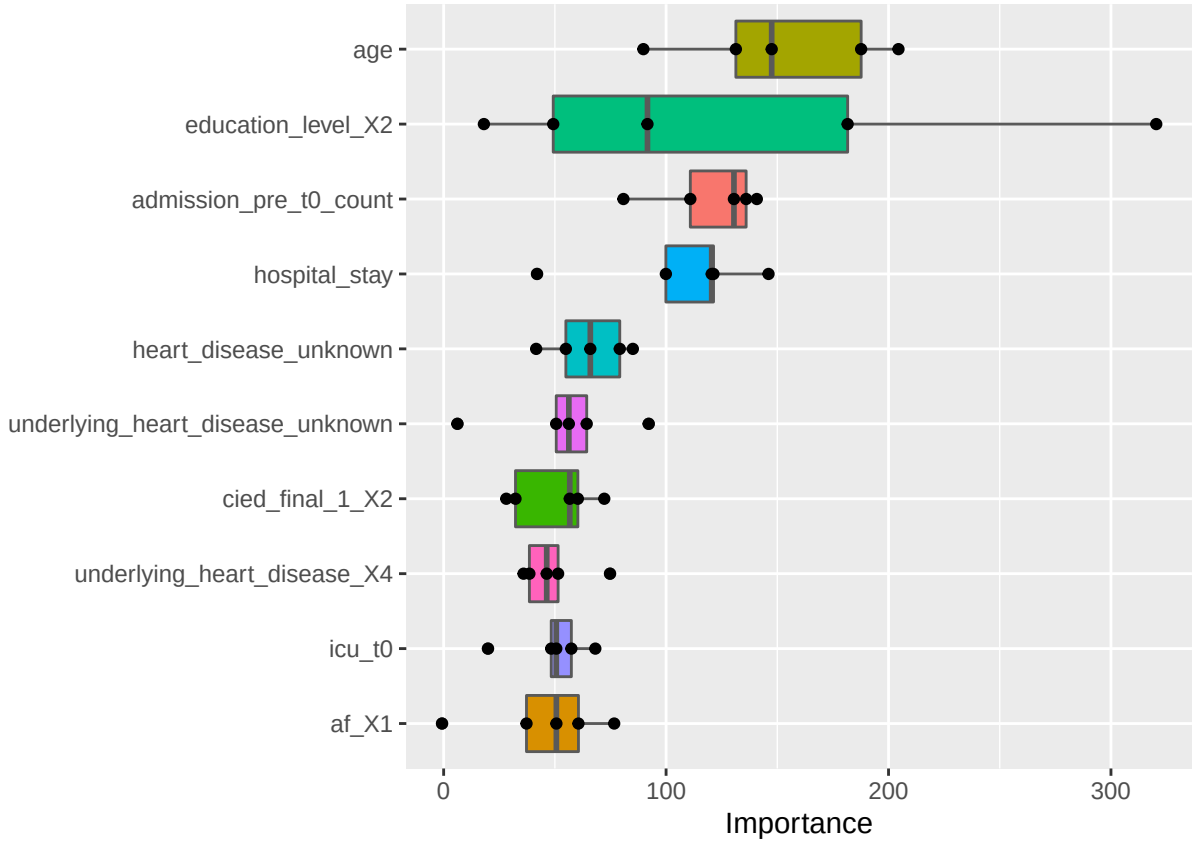

```
## [1] "Optimal Threshold: 0.01"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 3764    6
##    1  942   18
##
##                Accuracy : 0.7996
##                  95% CI : (0.7879, 0.8109)
##     No Information Rate : 0.9949
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.027
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.79983
##             Specificity : 0.75000
##          Pos Pred Value : 0.99841
##          Neg Pred Value : 0.01875
##              Prevalence : 0.99493
##          Detection Rate : 0.79577
##    Detection Prevalence : 0.79704
##       Balanced Accuracy : 0.77492
```

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

```r
pfun_glmnet <- function(object, newdata) predict(object, newx = newdata)

extract_vip(glm_fit, pred_wrapper = pfun_glmnet,
            reference_class = "1", method = 'model')
```




```r
extract_vip(glm_fit, pred_wrapper = pfun_glmnet,
            reference_class = "1", method = 'permute')
```

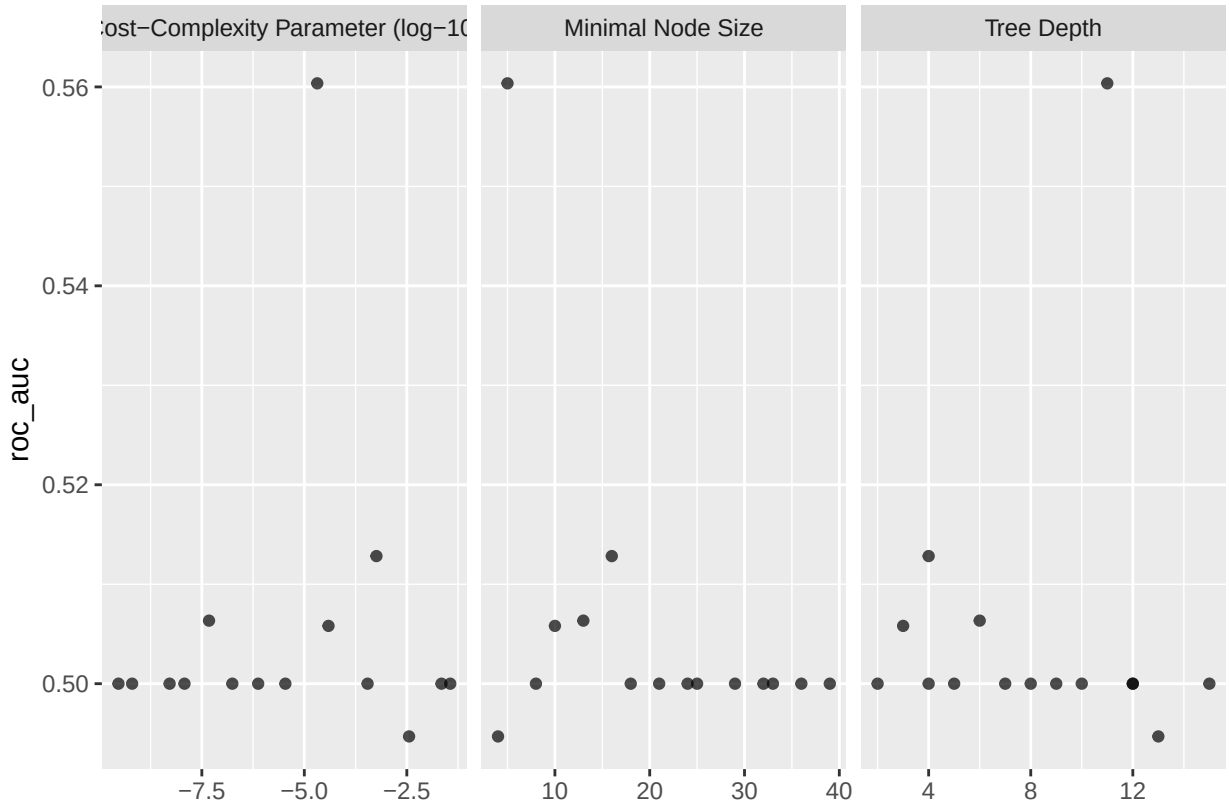


Minutes to run: 1.581

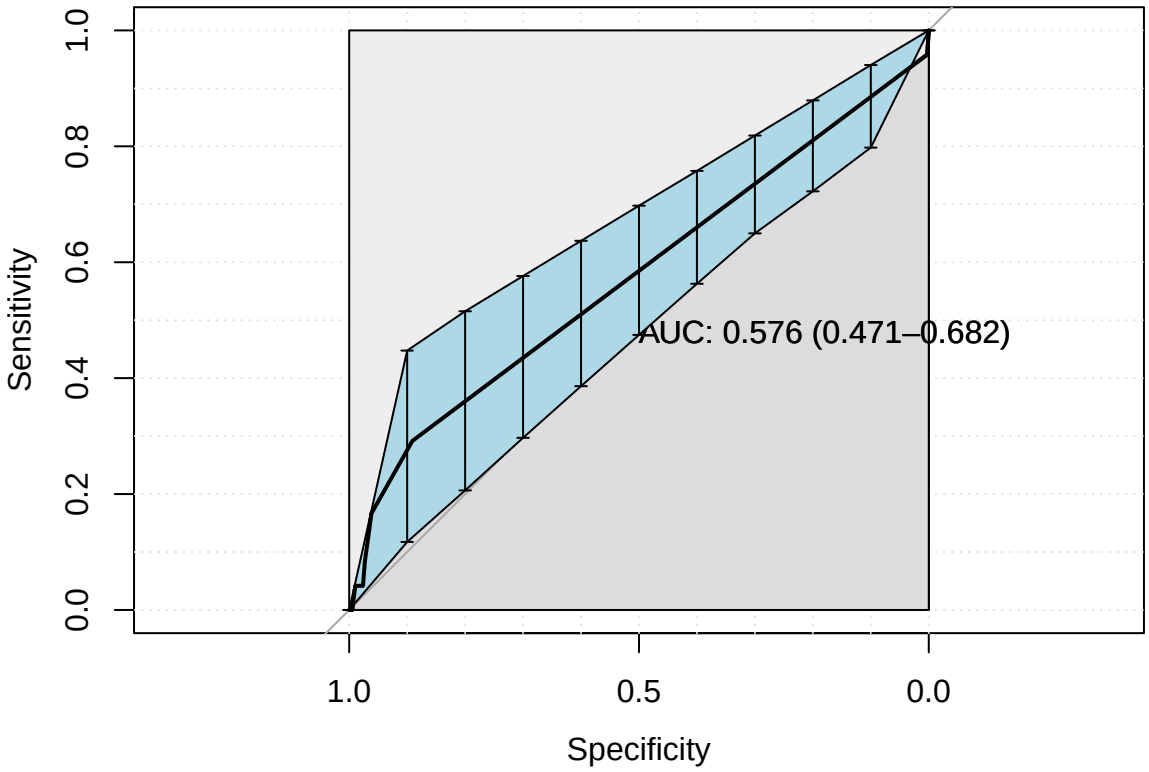
Decision Tree

```
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()) %>%
  step_zv(all_predictors())

tree_spec <-
  decision_tree(cost_complexity = tune(),
                tree_depth = tune(),
                min_n = tune()) %>%
  set_mode("classification") %>%
  set_engine("rpart")

tree_grid <- grid_latin_hypercube(cost_complexity(),
                                  tree_depth(),
                                  min_n(),
                                  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)
```

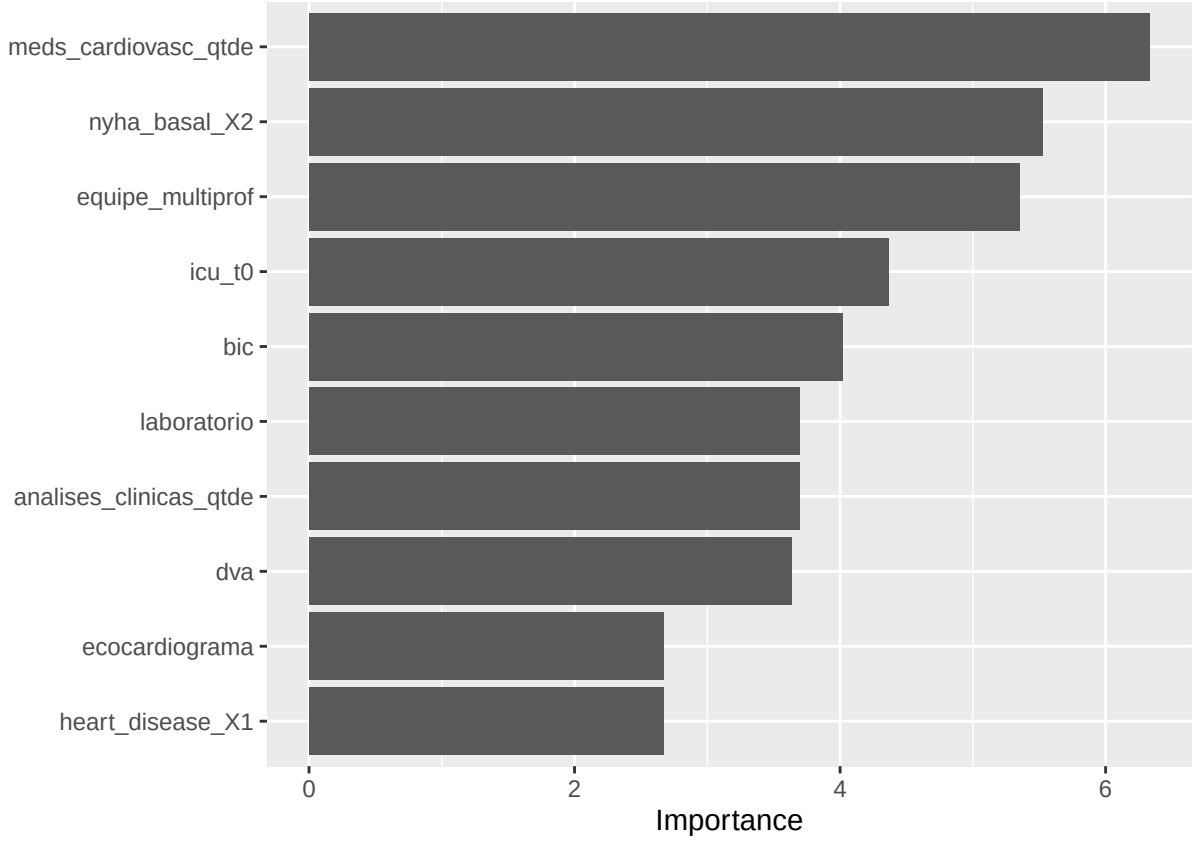
```
tree_tune %>%
  collect_metrics()

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



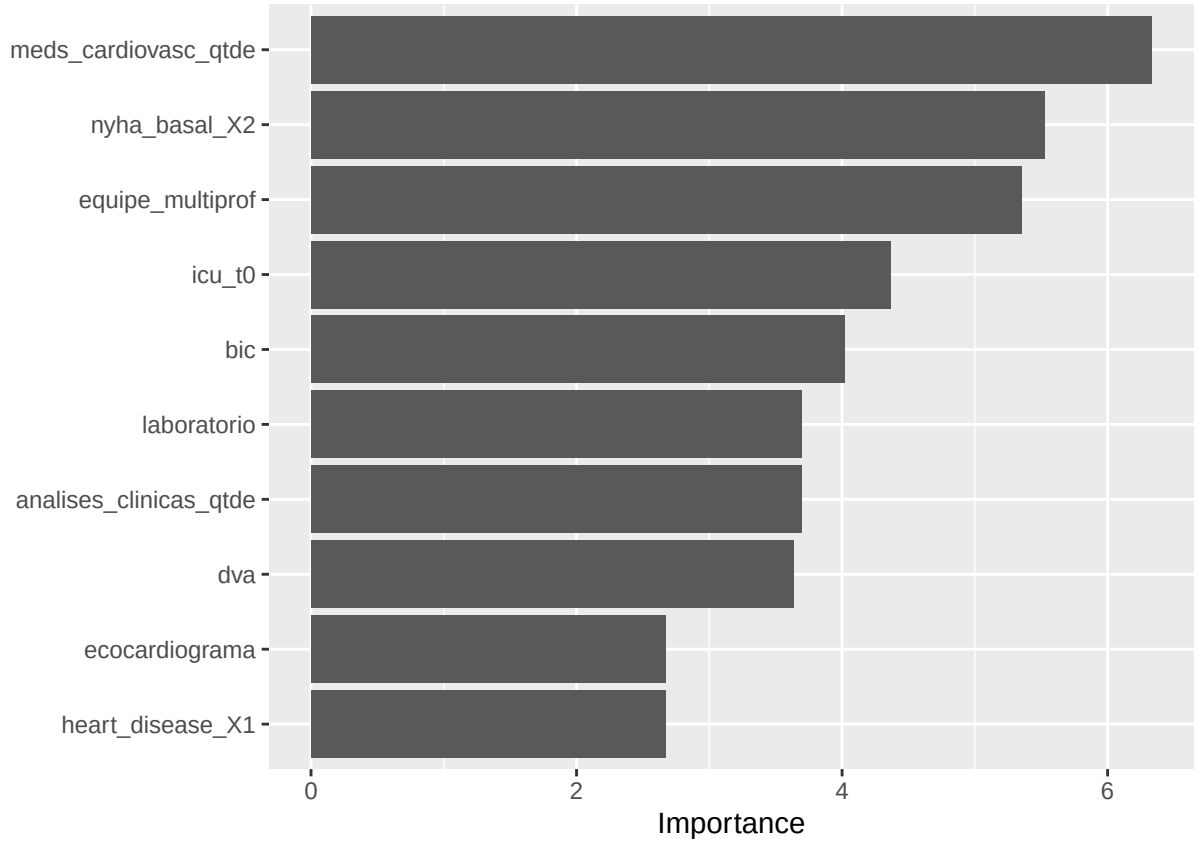

```
tree_tune %>%
  show_best("roc_auc")

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)
```



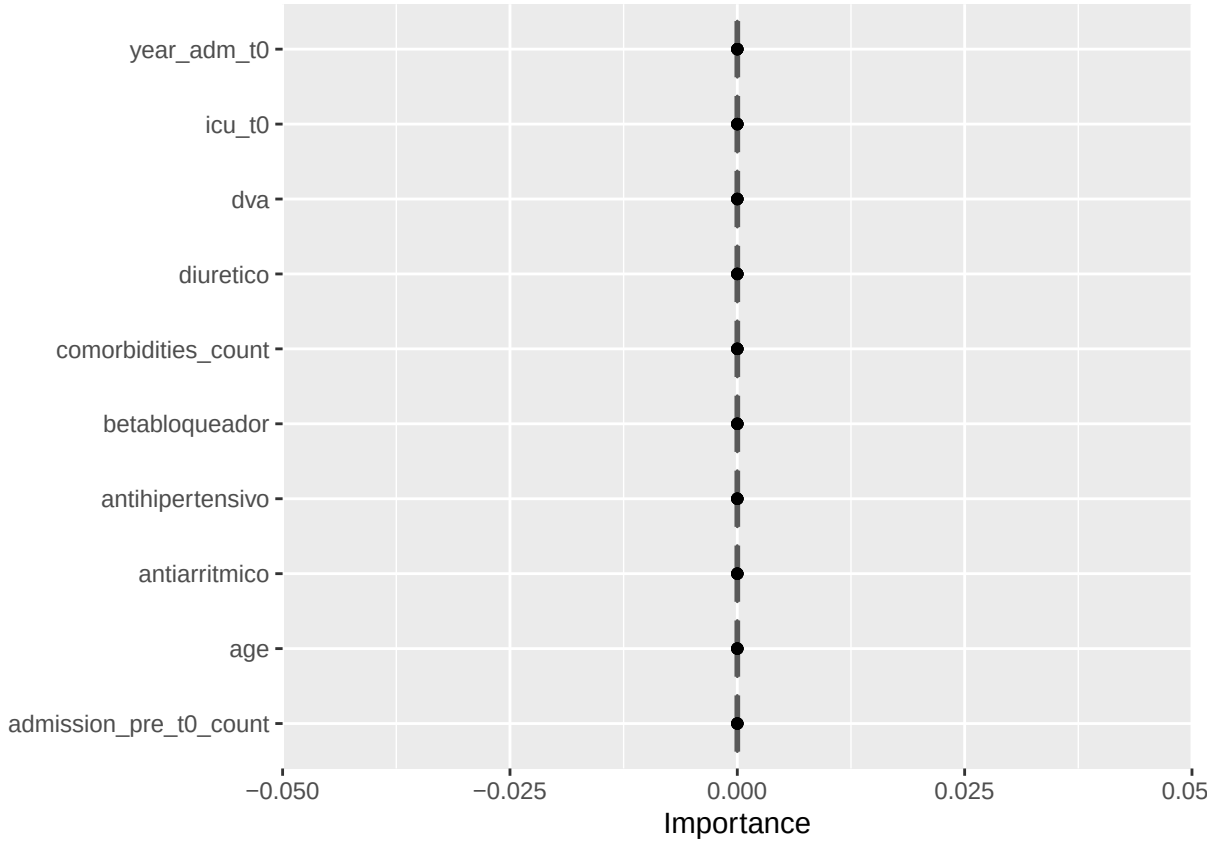

```
if (tree_auc$auc > 0.55) {
  final_tree_fit %>%
    extract_fit_parsnip() %>%
    vip()
}
```



```
extract_vip(final_tree_fit, pred_wrapper = predict,
            reference_class = "0", use_matrix = FALSE,
            method = 'model')
```




```
extract_vip(final_tree_fit, pred_wrapper = predict,
            reference_class = "1", use_matrix = FALSE,
            method = 'permute')
```

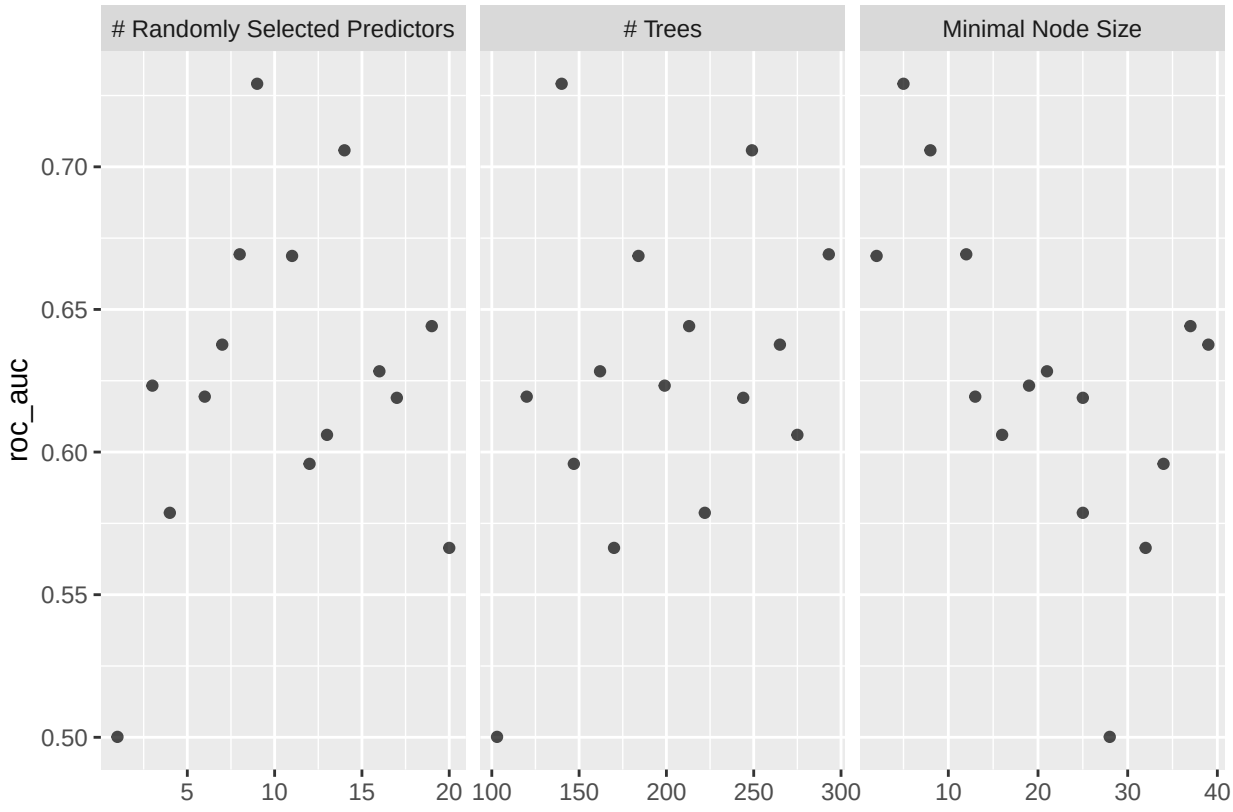


Minutes to run: 1.044

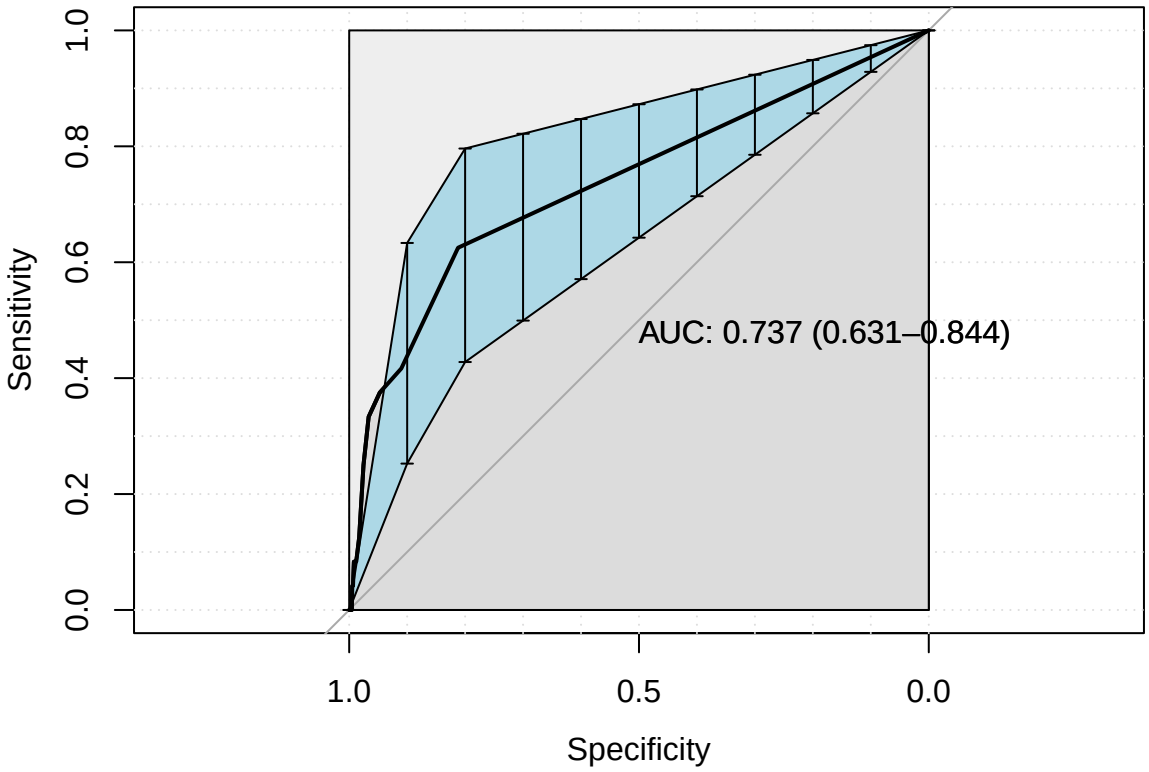
Random Forest

```
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()) %>%
  step_zv(all_predictors()) %>%
  step_impute_mean(all_numeric_predictors())

rf_spec <-
  rand_forest(mtry = tune(),
              trees = tune(),
              min_n = tune()) %>%
  set_mode("classification") %>%
  set_engine("randomForest",
             probability = TRUE,
             nthread = 8)

rf_grid <- grid_latin_hypercube(mtry(range = c(1L, 20L)),
                                trees(range = c(100L, 300L)),
                                min_n(),
                                size = grid_size)

rf_workflow <-
  workflow() %>%
  add_recipe(rf_recipe) %>%
  add_model(rf_spec)
```

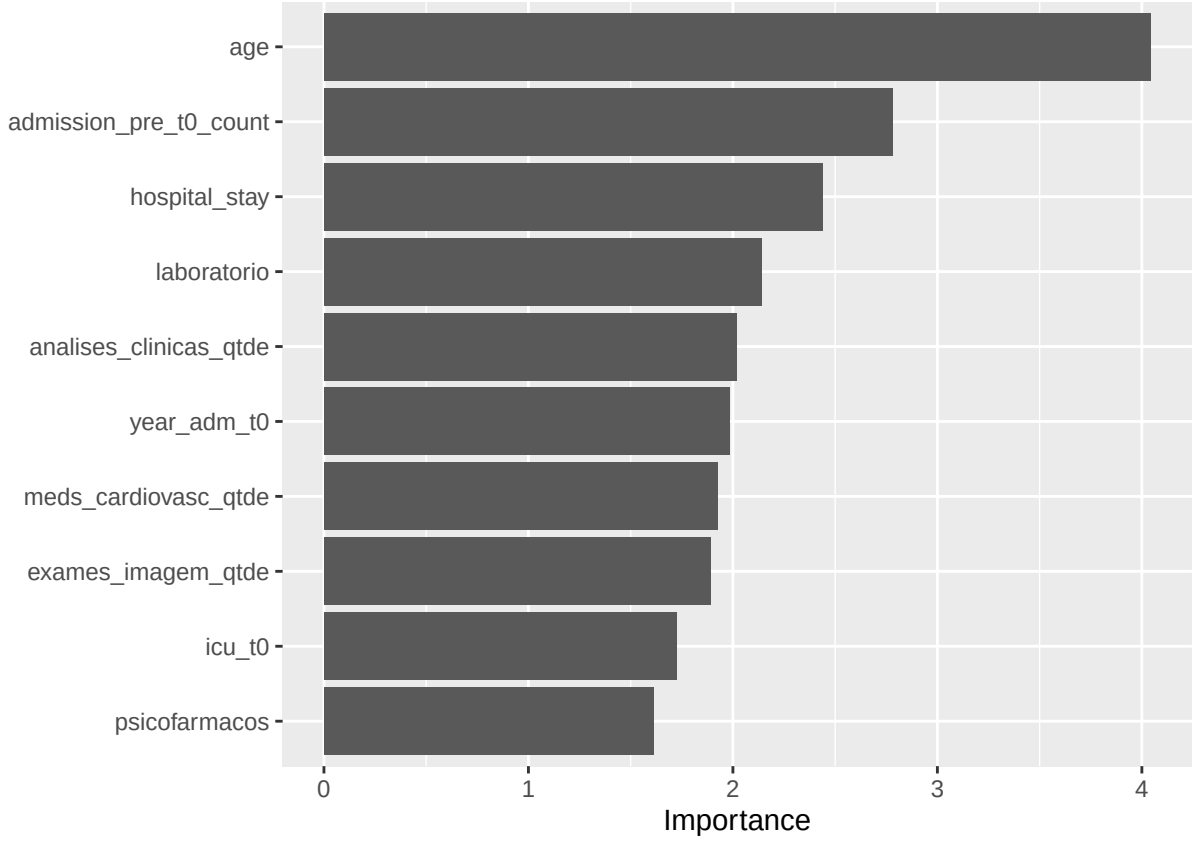
```
rf_tune <-
  rf_workflow %>%
  tune_grid(resamples = df_folds,
            grid = rf_grid)

rf_tune %>%
  collect_metrics()

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



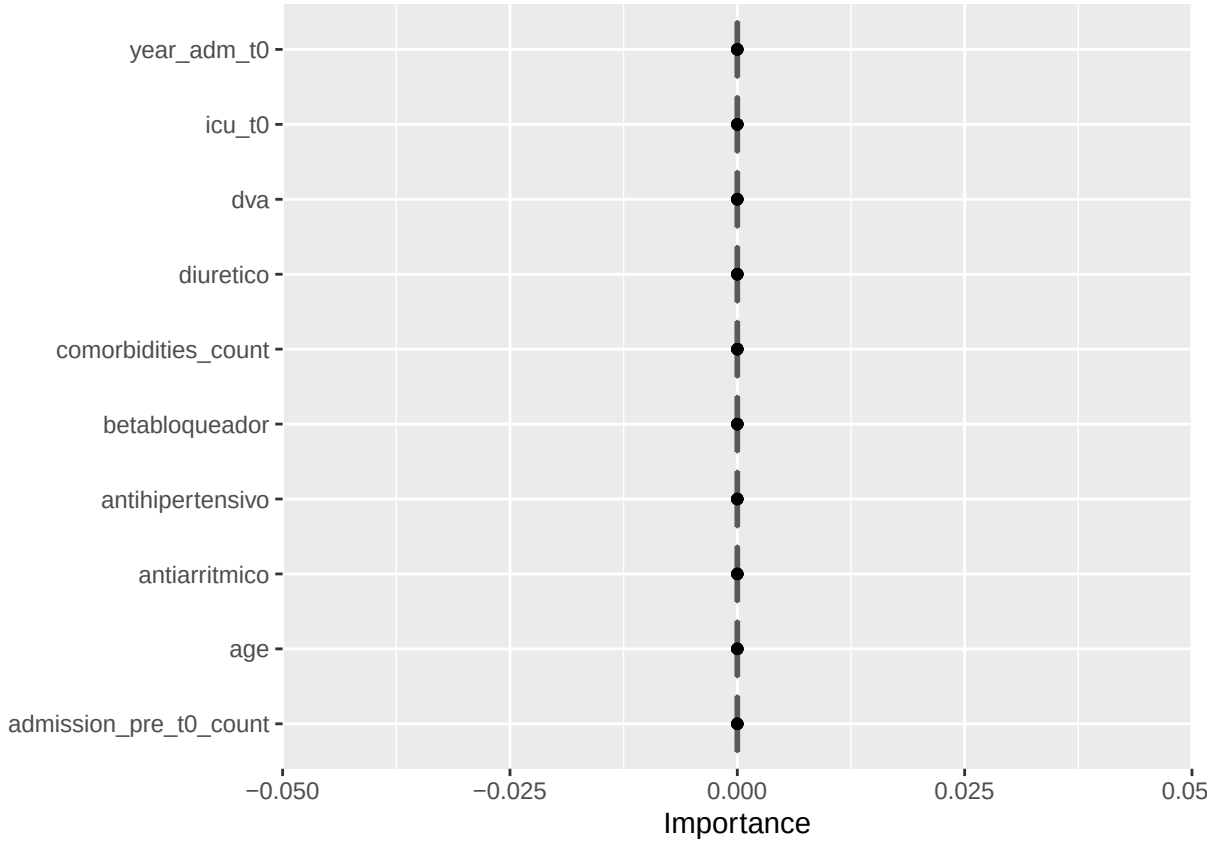

```
rf_tune %>%
  show_best("roc_auc")

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)
```




```
pfun_rf <- function(object, newdata) predict(object, data = newdata)

extract_vip(final_rf_fit, pred_wrapper = predict,
            reference_class = "1", use_matrix = FALSE,
            method = 'model')
```



```
extract_vip(final_rf_fit, pred_wrapper = predict,
            reference_class = "1", use_matrix = FALSE,
            method = 'permute')
```



Minutes to run: 10.679

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()) %>%
#   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 = grid_size)
#
# 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

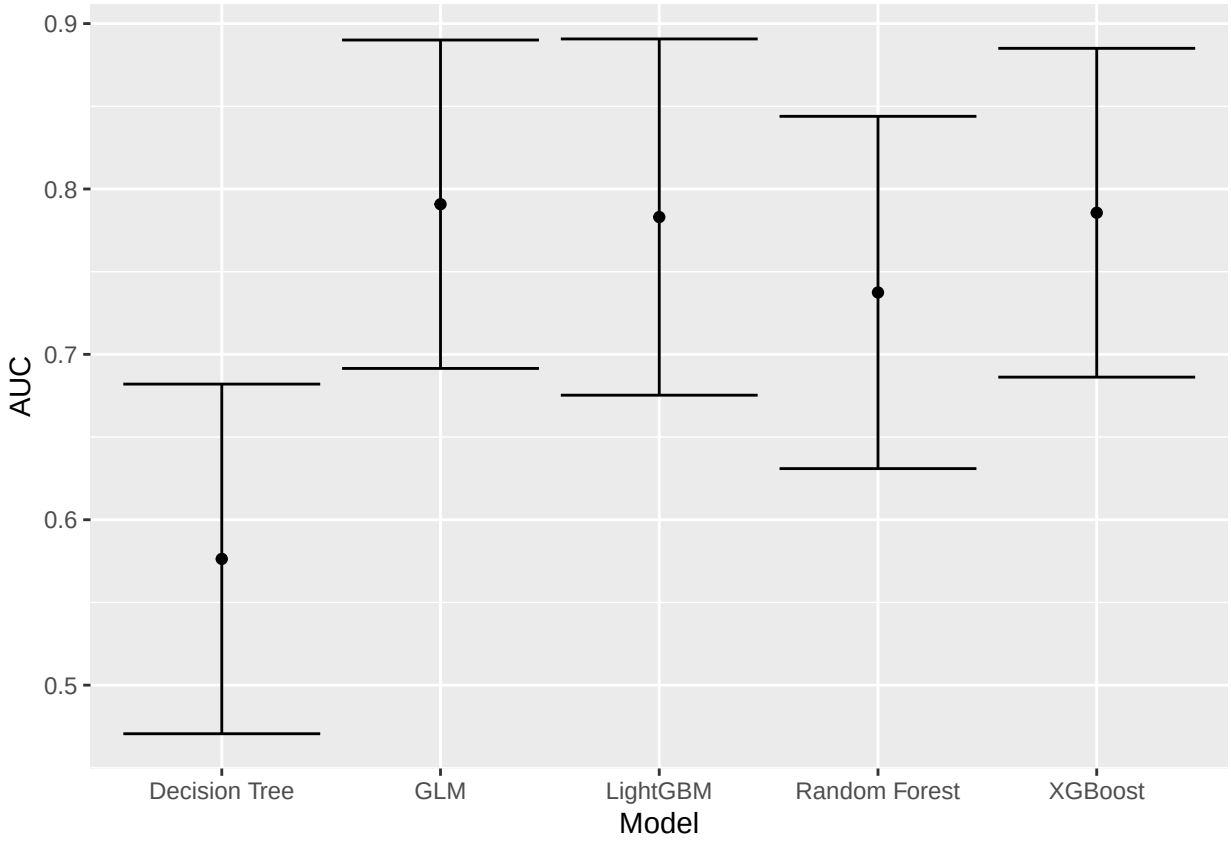
```
# svm_recipe <-
#   recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
#   step_novel(all_nominal_predictors()) %>%
#   step_unknown(all_nominal_predictors()) %>%
#   step_other(all_nominal_predictors(), threshold = 0.05, other = ".merged") %>%
#   step_dummy(all_nominal_predictors())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

```
#
# 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(glmnet_auc$auc), glmnet_auc$ci[1], glmnet_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