Model Selection - readmission_60d

Eduardo Yuki Yada

Global parameters

```
k <- params$k # Number of folds for cross validation
grid_size <- params$grid_size # Number of parameter combination to tune on each model
repeats <- params$repeats
RUN_ALL_MODELS <- params$RUN_ALL_MODELS
Hmisc::list.tree(params)
```

```
##  params = list 5 (952 bytes)
## .  outcome_column = character 1= readmission_60d
## .  k = double 1= 10
## .  grid_size = double 1= 20
## .  repeats = double 1= 2
## .  RUN_ALL_MODELS = logical 1= TRUE
```

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")
predict <- stats::predict
```

Minutes to run: 0.046

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,
```

```r
          recursive = TRUE)

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

Minutes to run: 0

Eligible features

```r
cat_features_list = read_yaml(sprintf(
  "./auxiliar/significant_columns/categorical_%s.yaml",
  outcome_column
))

num_features_list = read_yaml(sprintf(
  "./auxiliar/significant_columns/numerical_%s.yaml",
  outcome_column
))

features_list = c(cat_features_list, num_features_list)
```

Minutes to run: 0

```r
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))

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_pre_t0_count
## 21. admission_pre_t0_180d
## 22. icu_t0
## 23. dialysis_t0
## 24. admission_t0_emergency
## 25. aco
## 26. antiarritmico
## 27. betabloqueador
## 28. ieca_bra
## 29. dva
## 30. digoxina
## 31. estatina
## 32. diuretico
## 33. vasodilatador
## 34. insuf_cardiaca
## 35. espironolactona
## 36. bloq_calcio
## 37. antiplaquetario_ev
## 38. insulina
## 39. anticonvulsivante
## 40. psicofarmacos
## 41. antifungico
## 42. antiviral
## 43. classe_meds_qtde
## 44. meds_cardiovasc_qtde
## 45. meds_antimicrobianos
## 46. ventilacao_mecanica
## 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. interconsulta
## 60. equipe_multiprof
## 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. ressonancia
## 82. exames_imagem_qtde
## 83. bic
## 84. hospital_stay
```

Minutes to run: 0

Train test split (70%/30%)

```r
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

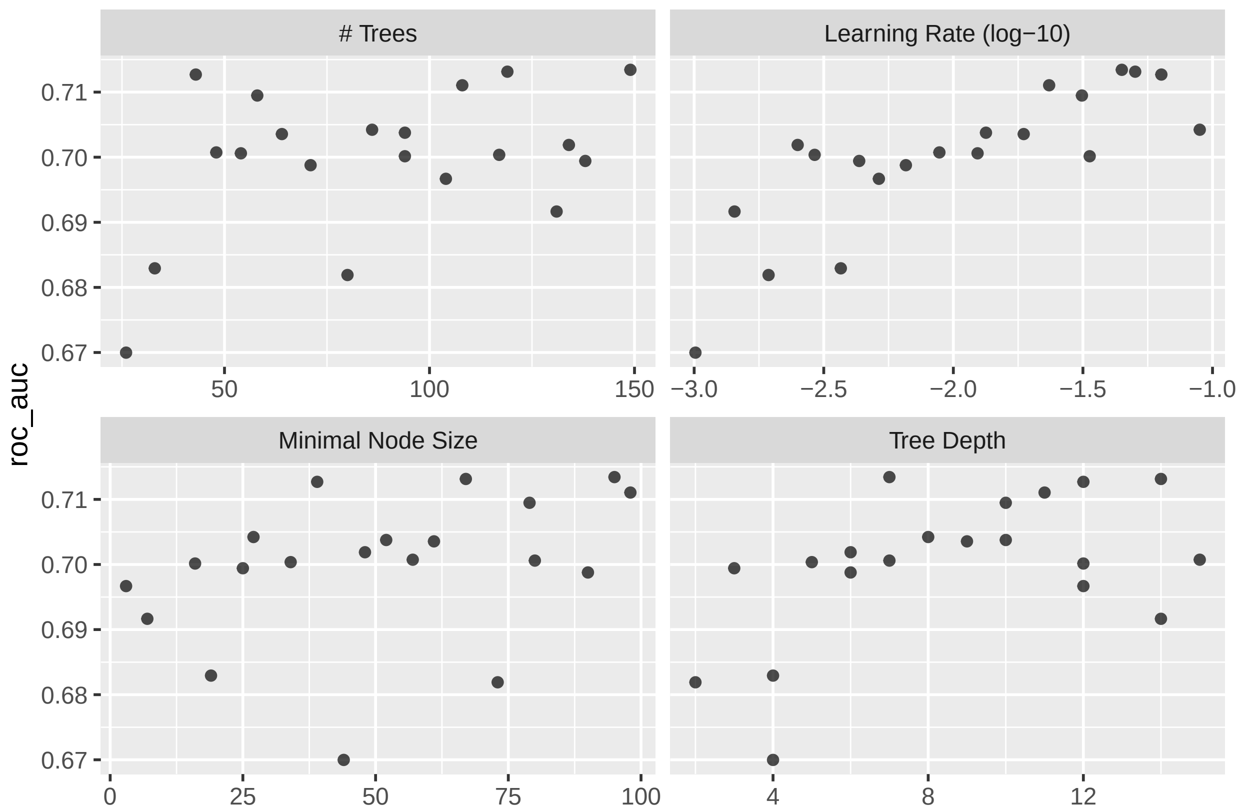
Boosted Tree (XGBoost)

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

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

xgboost_grid <- grid_latin_hypercube(
  trees(range = c(25L, 150L)),
  min_n(range = c(2L, 100L)),
  tree_depth(range = c(2L, 15L)),
  learn_rate(range = c(-3, -1), trans = log10_trans()),
  size = grid_size
)

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

```r
  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")
```

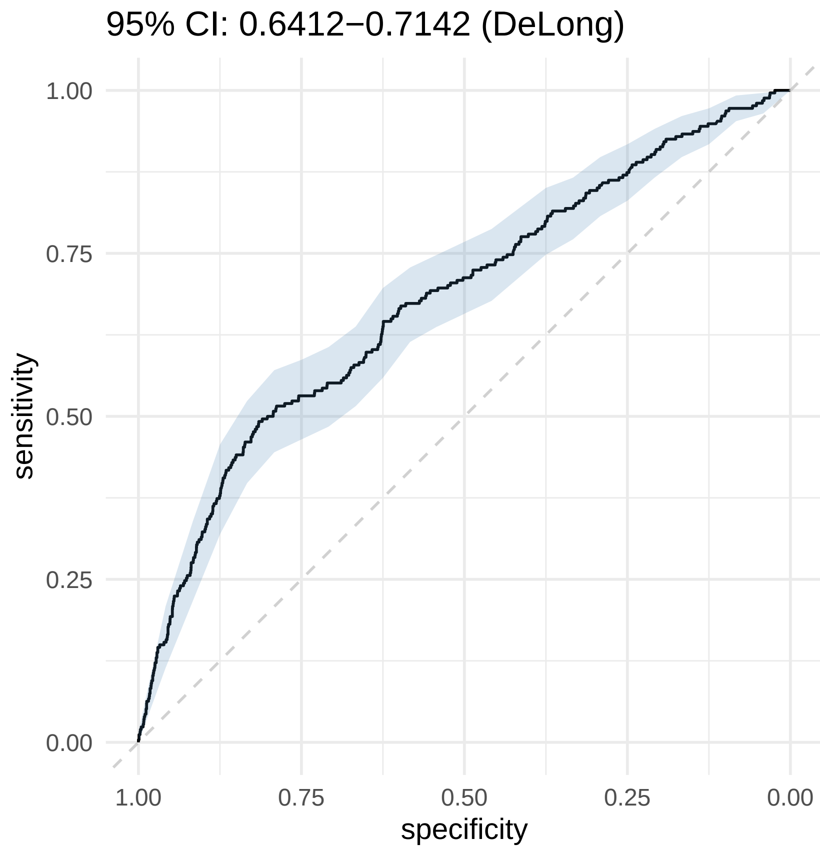
```
## # A tibble: 5 x 10
##   trees min_n tree_depth learn_rate .metric .estimator  mean     n std_err .config
##   <int> <int>      <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1   149    95          7     0.0447 roc_auc binary     0.713    10  0.0122 Prepro~
## 2   119    67         14     0.0503 roc_auc binary     0.713    10  0.0118 Prepro~
## 3    43    39         12     0.0635 roc_auc binary     0.713    10  0.0115 Prepro~
## 4   108    98         11     0.0234 roc_auc binary     0.711    10  0.0119 Prepro~
## 5    58    79         10     0.0313 roc_auc binary     0.709    10  0.0123 Prepro~
```

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

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



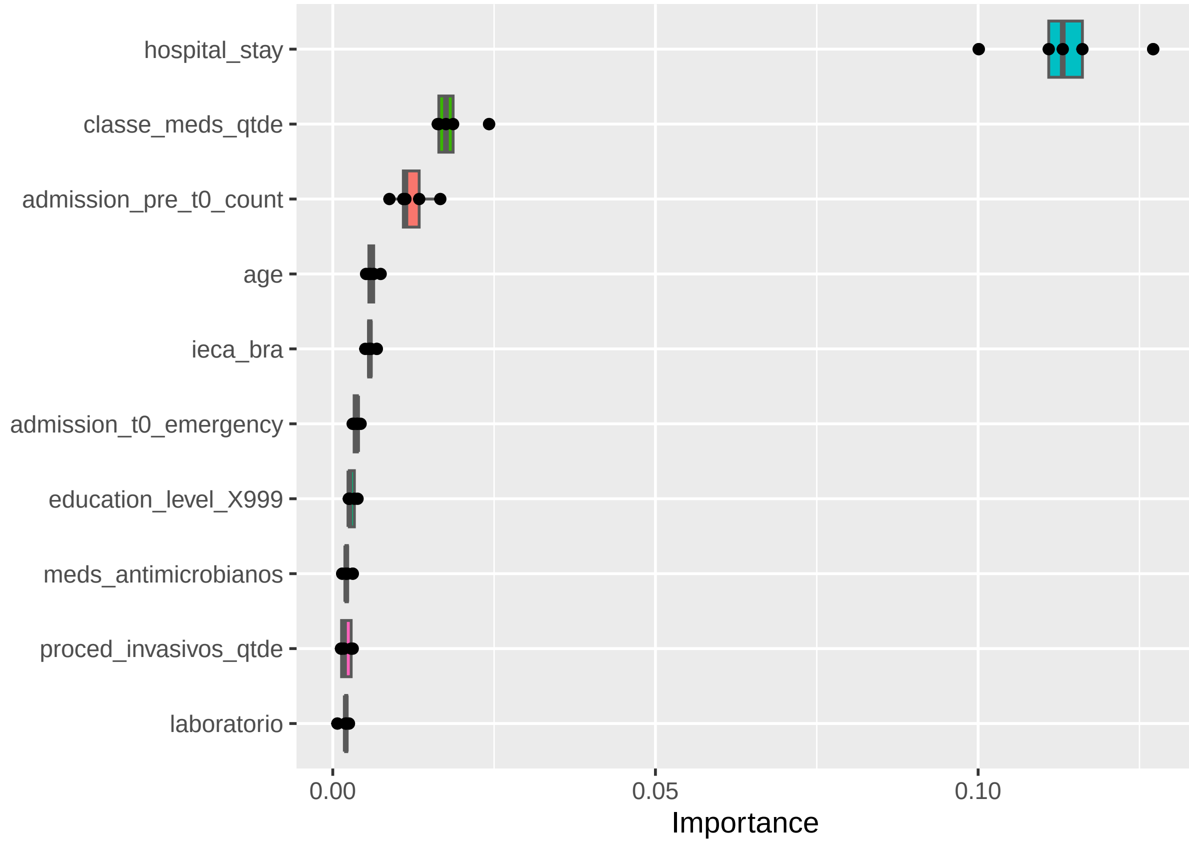

```r
final_xgboost_workflow <-
  xgboost_workflow %>%
  finalize_workflow(best_xgboost)

last_xgboost_fit <-
  final_xgboost_workflow %>%
  last_fit(df_split)

final_xgboost_fit <- extract_workflow(last_xgboost_fit)
```

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




```
## [1] "Optimal Threshold: 0.09"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 3650  129
##    1  826  125
##
##                Accuracy : 0.7981
##                  95% CI : (0.7864, 0.8095)
##     No Information Rate : 0.9463
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1341
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.8155
##             Specificity : 0.4921
##          Pos Pred Value : 0.9659
##          Neg Pred Value : 0.1314
##              Prevalence : 0.9463
##          Detection Rate : 0.7717
##    Detection Prevalence : 0.7989
##       Balanced Accuracy : 0.6538
##
##        'Positive' Class : 0
##
```

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




```
xgboost_parameters <- xgboost_tune %>%
  show_best("roc_auc", n = 1) %>%
  select(-.metric, -.estimator, -.config, -mean, -n, -std_err) %>%
  as.list
```

Minutes to run: 4.364

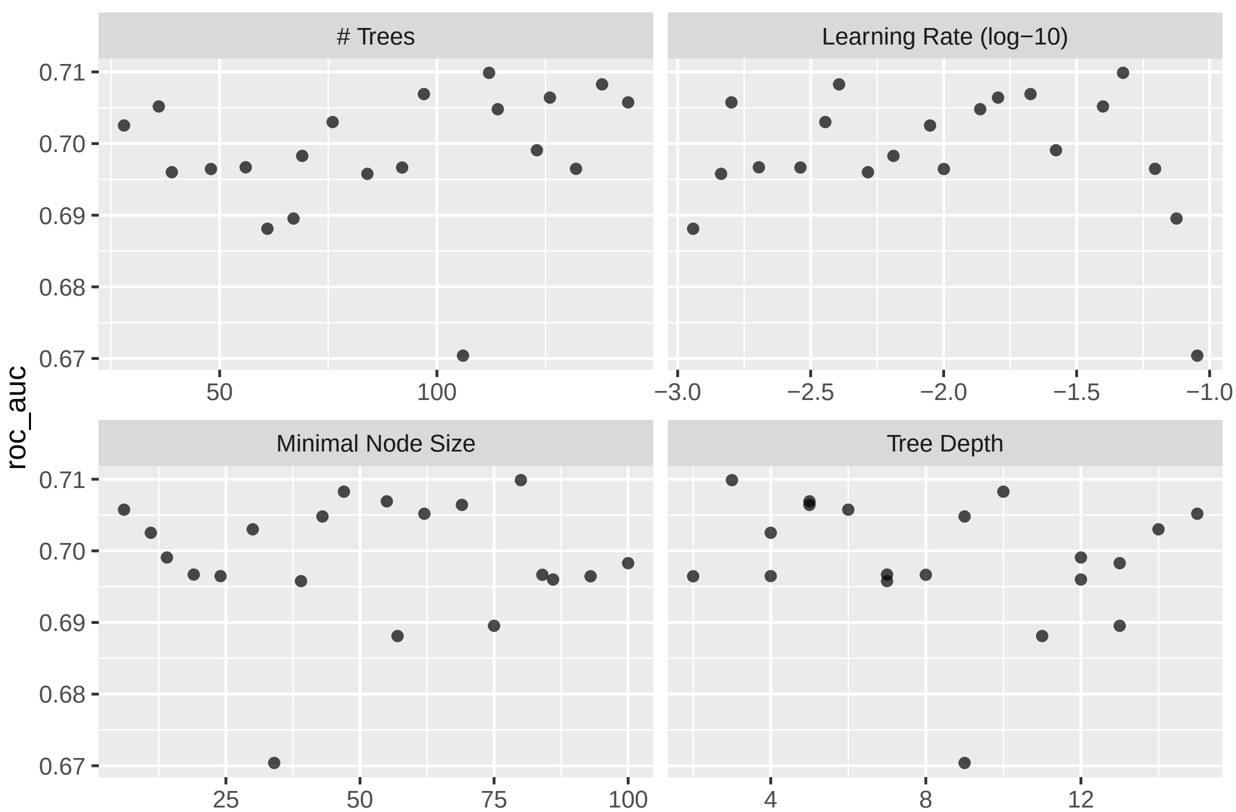
Boosted Tree (LightGBM)

```
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(
  trees = tune(),
  min_n = tune(),
  tree_depth = tune(),
  learn_rate = tune(),
  sample_size = 1
) %>%
  set_engine("lightgbm",
             nthread = 8) %>%
  set_mode("classification")

lightgbm_grid <- grid_latin_hypercube(
  trees(range = c(25L, 150L)),
  min_n(range = c(2L, 100L)),
  tree_depth(range = c(2L, 15L)),
  learn_rate(range = c(-3, -1), trans = log10_trans()),
  size = grid_size
```

```r
)

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")
```

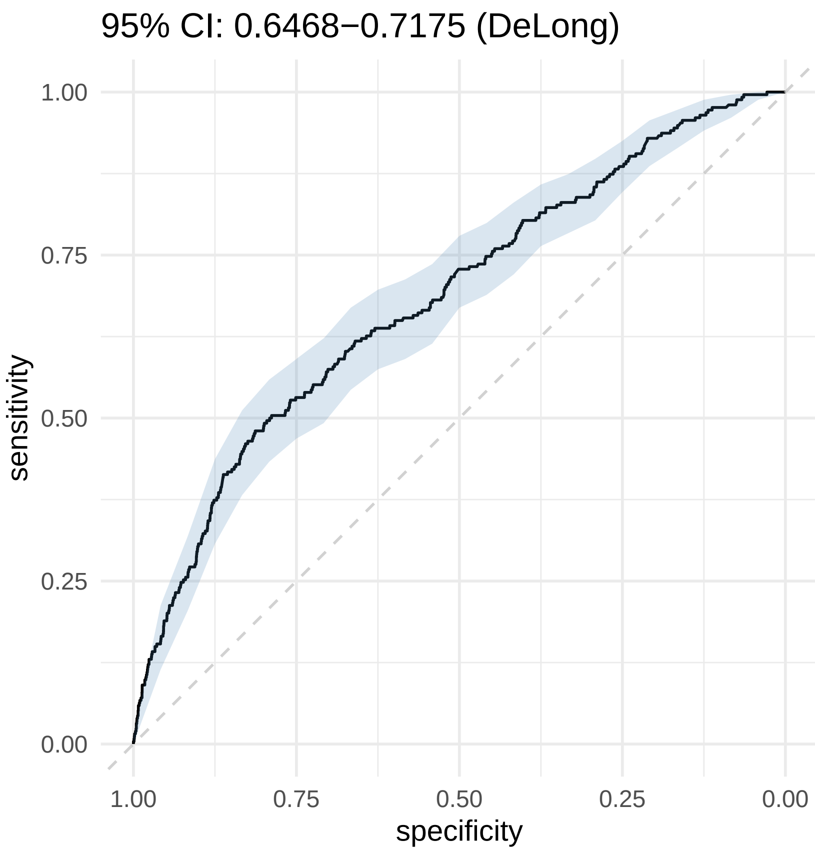
```
## # A tibble: 5 x 10
##   trees min_n tree_depth learn_rate .metric .estimator  mean     n std_err .config
##   <int> <int>      <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1   112    80          3    0.0473  roc_auc binary     0.710    10  0.0134 Prepro~
## 2   138    47         10    0.00404 roc_auc binary     0.708    10  0.0112 Prepro~
## 3    97    55          5    0.0212  roc_auc binary     0.707    10  0.0124 Prepro~
## 4   126    69          5    0.0160  roc_auc binary     0.706    10  0.0117 Prepro~
## 5   144     6          6    0.00159 roc_auc binary     0.706    10  0.0130 Prepro~
```

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

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




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

last_lightgbm_fit <-
```

```r
  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: 0.08"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 3640  132
##    1  836  122
##
##                Accuracy : 0.7953
##                  95% CI : (0.7836, 0.8068)
##     No Information Rate : 0.9463
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1272
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.8132
##             Specificity : 0.4803
##          Pos Pred Value : 0.9650
##          Neg Pred Value : 0.1273
##              Prevalence : 0.9463
##          Detection Rate : 0.7696
##    Detection Prevalence : 0.7975
##       Balanced Accuracy : 0.6468
##
```

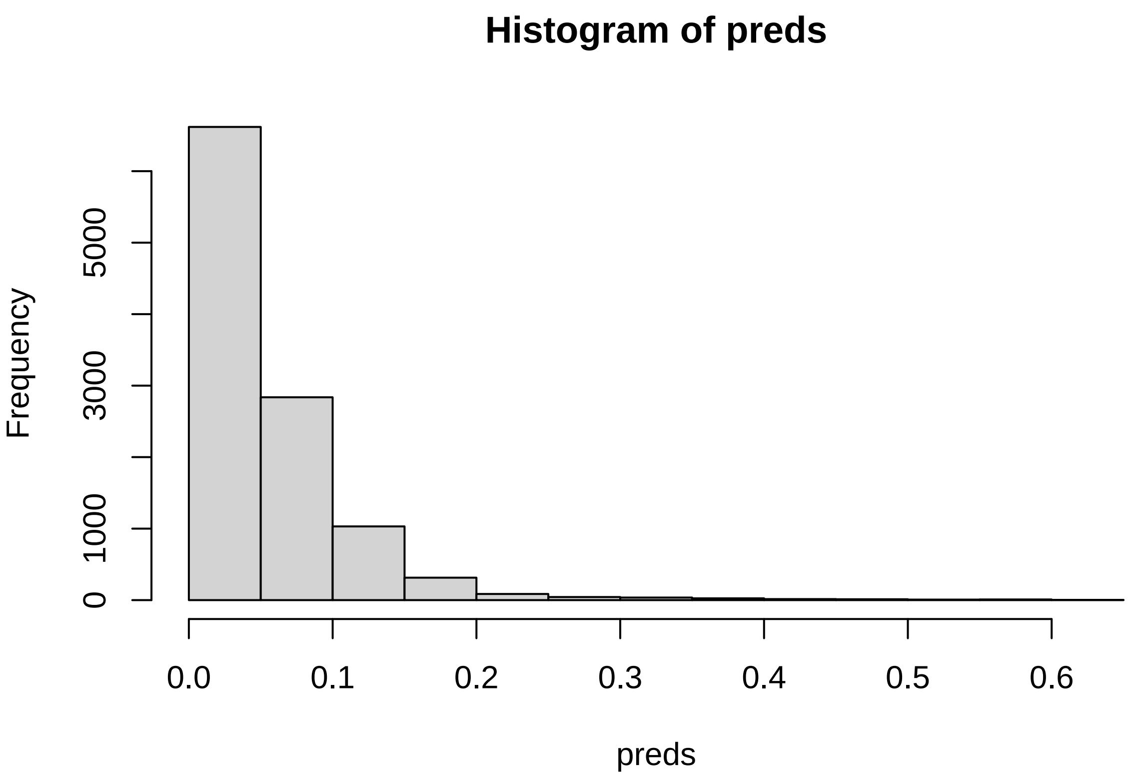
```
##        'Positive' Class : 0
##
```

```
lightgbm_parameters <- lightgbm_tune %>%
  show_best("roc_auc", n = 1) %>%
  select(-.metric, -.estimator, -.config, -mean, -n, -std_err) %>%
  as.list

Hmisc::list.tree(lightgbm_parameters)
```

```
##  lightgbm_parameters = list 4 (736 bytes)
## .   trees = integer 1= 112
## .   min_n = integer 1= 80
## .   tree_depth = integer 1= 3
## .   learn_rate = double 1= 0.047266
```

```
con <- file(sprintf('./auxiliar/model_selection/hyperparameters/%s.yaml', outcome_column), "w")
write_yaml(lightgbm_parameters, con)
close(con)
```

Minutes to run: 3.226

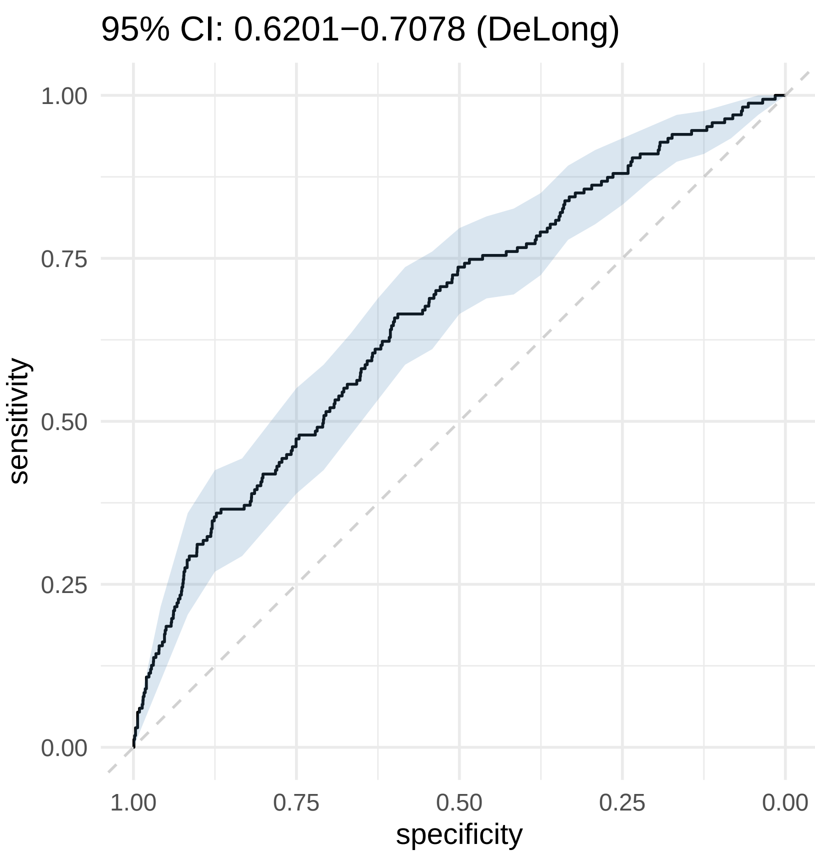


Minutes to run: 0.011

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())
```

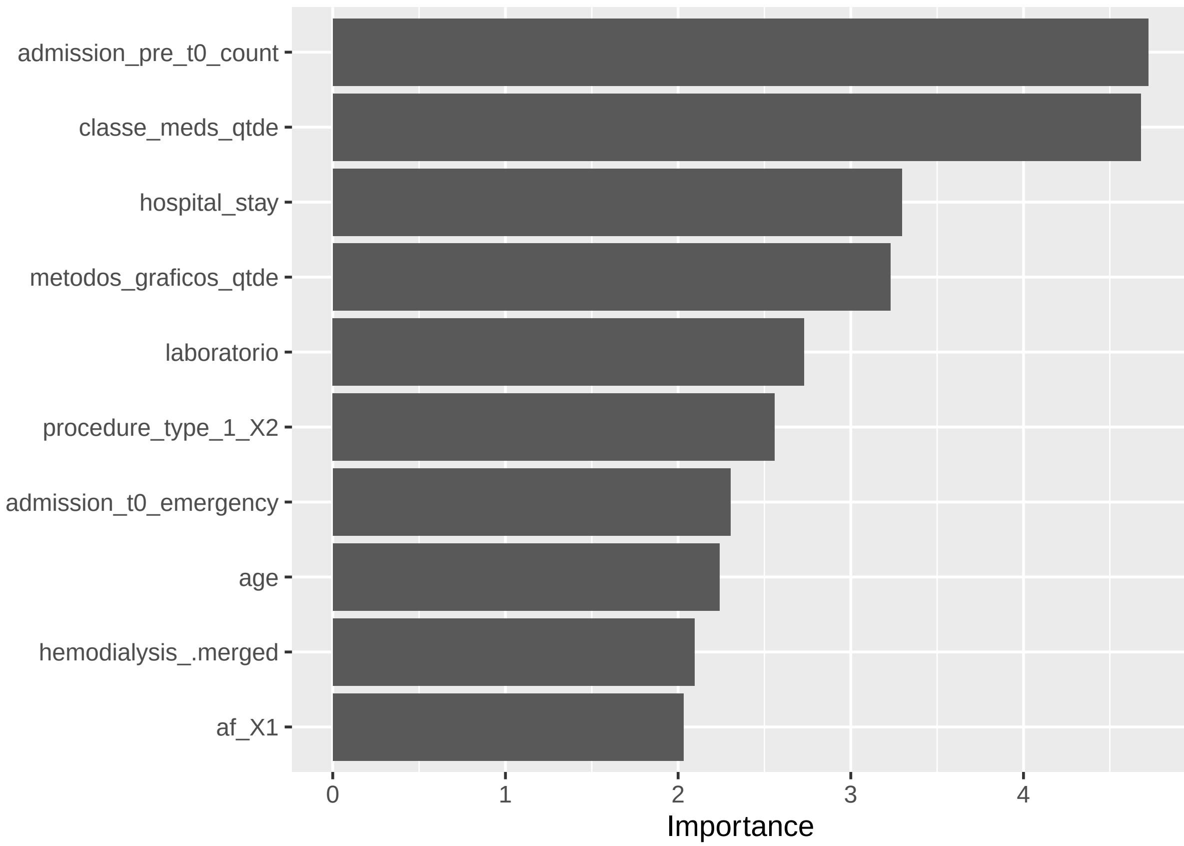
```r
glmnet_spec <-
  logistic_reg(penalty = 0) %>%
  set_mode("classification") %>%
  set_engine("glm")

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)
```



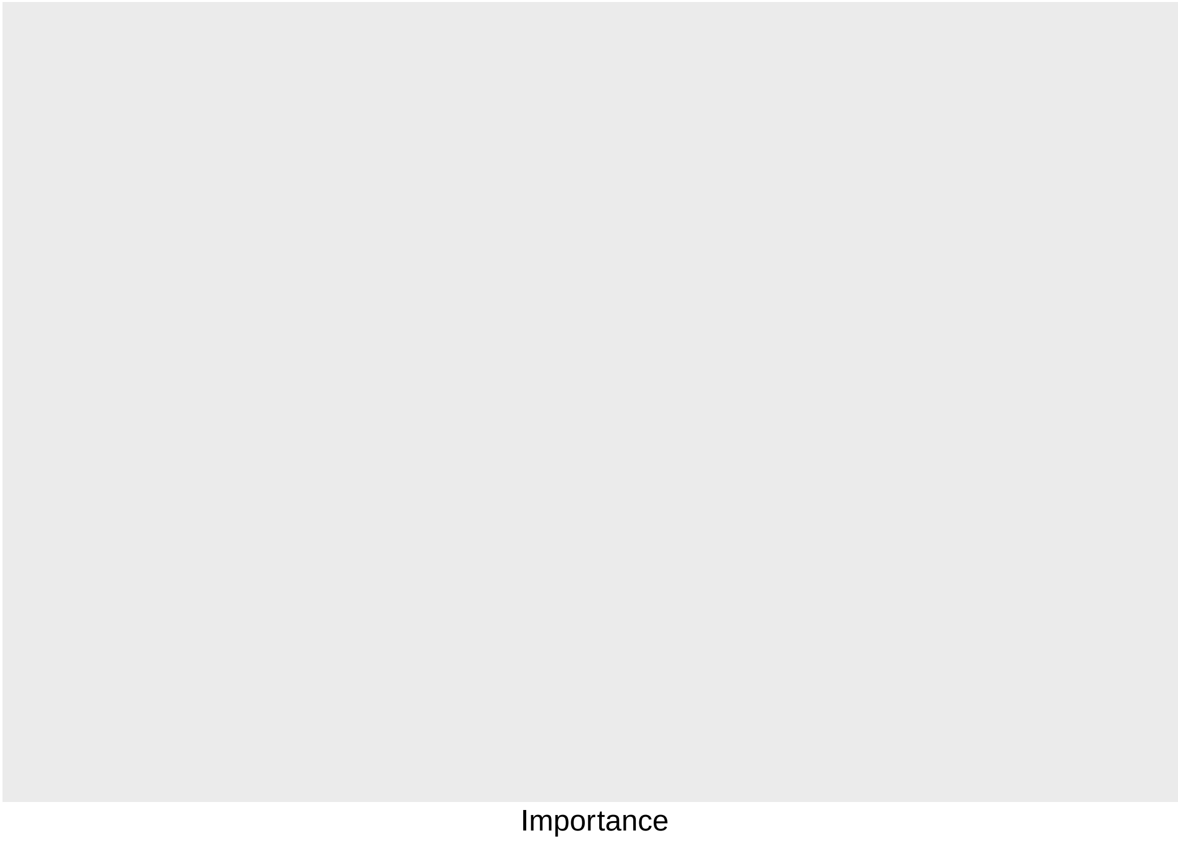

```
## [1] "Optimal Threshold: 0.04"
## Confusion Matrix and Statistics
##
##      reference
## data    0    1
##    0 1673   56
##    1 1141  111
##
##                  Accuracy : 0.5985
##                    95% CI : (0.5806, 0.6161)
##       No Information Rate : 0.944
##       P-Value [Acc > NIR] : 1
##
##                     Kappa : 0.0639
##
##    Mcnemar's Test P-Value : <2e-16
##
##               Sensitivity : 0.59453
```

```
##              Specificity : 0.66467
##           Pos Pred Value : 0.96761
##           Neg Pred Value : 0.08866
##               Prevalence : 0.94398
##           Detection Rate : 0.56122
##     Detection Prevalence : 0.58001
##        Balanced Accuracy : 0.62960
##
##         'Positive' Class : 0
##
```

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

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




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



Minutes to run: 0.212

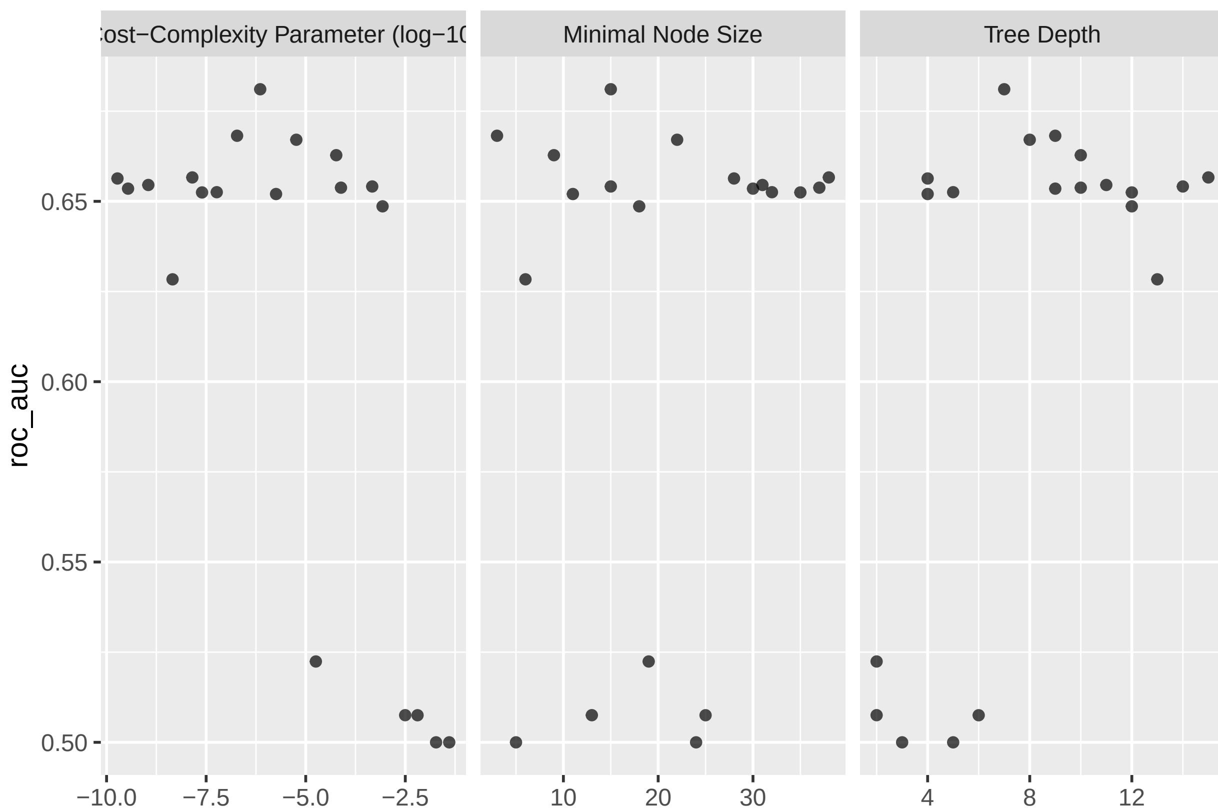
Decision Tree

```
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)
```

```r
tree_tune %>%
  collect_metrics()
```

```
## # A tibble: 40 x 9
##    cost_complexity tree_depth min_n .metric .estimator  mean     n std_err .config
##              <dbl>      <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1    0.0000000250         12    35 accura~ binary     0.937    10 0.00163 Prepro~
##  2    0.0000000250         12    35 roc_auc binary     0.652    10 0.0114  Prepro~
##  3    0.00314               2    13 accura~ binary     0.940    10 0.00186 Prepro~
##  4    0.00314               2    13 roc_auc binary     0.508    10 0.00753 Prepro~
##  5    0.00000180            4    11 accura~ binary     0.940    10 0.00186 Prepro~
##  6    0.00000180            4    11 roc_auc binary     0.652    10 0.0125  Prepro~
##  7    0.0000585            10     9 accura~ binary     0.933    10 0.00243 Prepro~
##  8    0.0000585            10     9 roc_auc binary     0.663    10 0.0189  Prepro~
##  9    0.00643               6    25 accura~ binary     0.940    10 0.00186 Prepro~
## 10    0.00643               6    25 roc_auc binary     0.508    10 0.00752 Prepro~
## # i 30 more rows
```

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

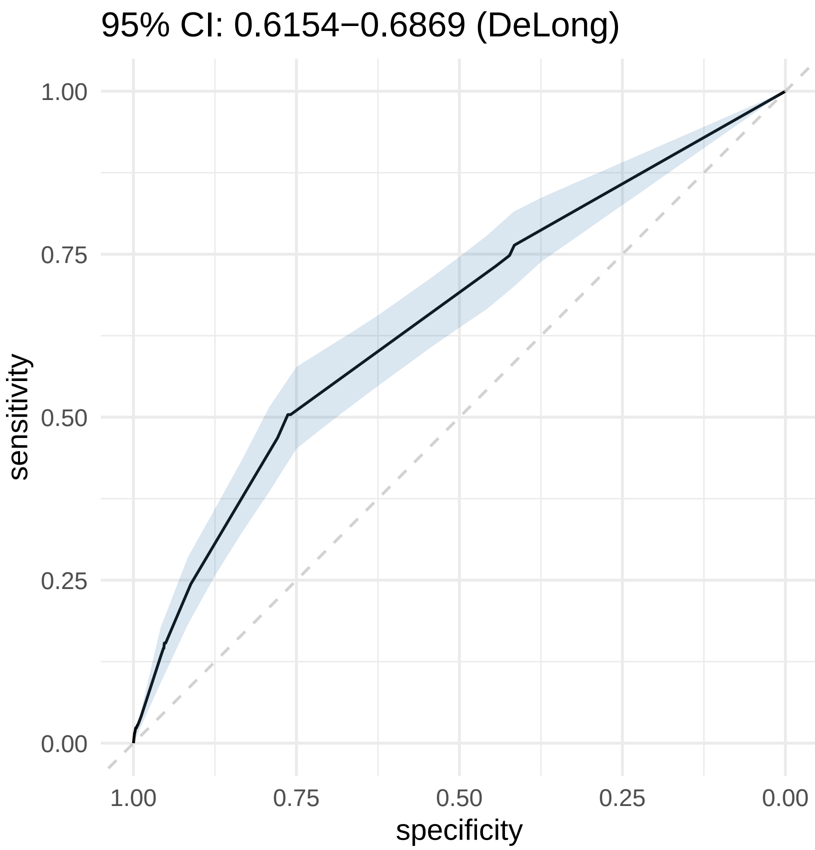



```r
tree_tune %>%
  show_best("roc_auc")
```

```
## # A tibble: 5 x 9
##   cost_complexity tree_depth min_n .metric .estimator  mean     n std_err .config
##             <dbl>      <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    0.000000721           7    15 roc_auc binary     0.681    10  0.0140 Preproc~
## 2    0.000000189           9     3 roc_auc binary     0.668    10  0.0149 Preproc~
## 3    0.00000581            8    22 roc_auc binary     0.667    10  0.0113 Preproc~
## 4    0.0000585            10     9 roc_auc binary     0.663    10  0.0189 Preproc~
## 5    0.0000000142         15    38 roc_auc binary     0.657    10  0.0119 Preproc~
```

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

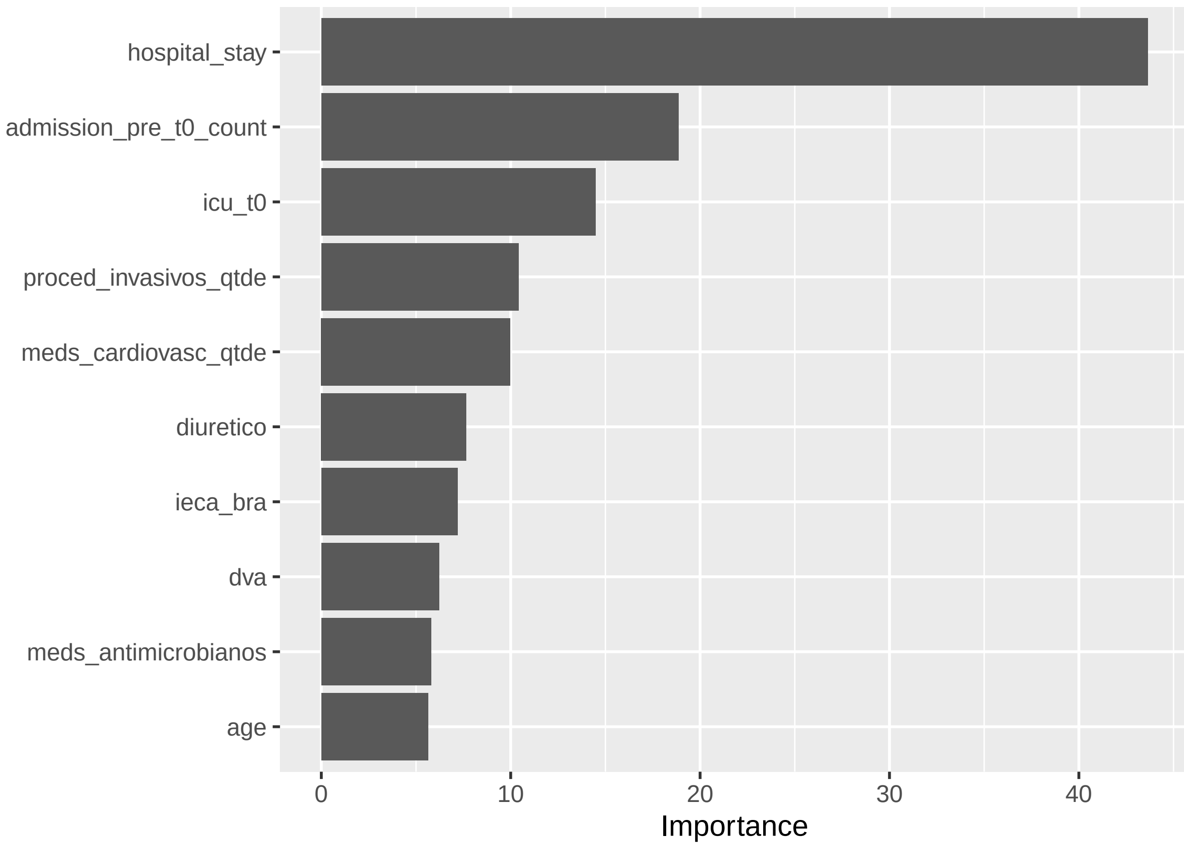
```r
final_tree_workflow <-
  tree_workflow %>%
  finalize_workflow(best_tree)

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)
```




```
## [1] "Optimal Threshold: 0.07"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 3416  126
##    1 1060  128
##
##                Accuracy : 0.7493
##                  95% CI : (0.7367, 0.7616)
##     No Information Rate : 0.9463
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0977
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7632
##             Specificity : 0.5039
##          Pos Pred Value : 0.9644
##          Neg Pred Value : 0.1077
```

```
##               Prevalence : 0.9463
##           Detection Rate : 0.7222
##     Detection Prevalence : 0.7488
##        Balanced Accuracy : 0.6336
##
##         'Positive' Class : 0
##
```

```
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: 4.37

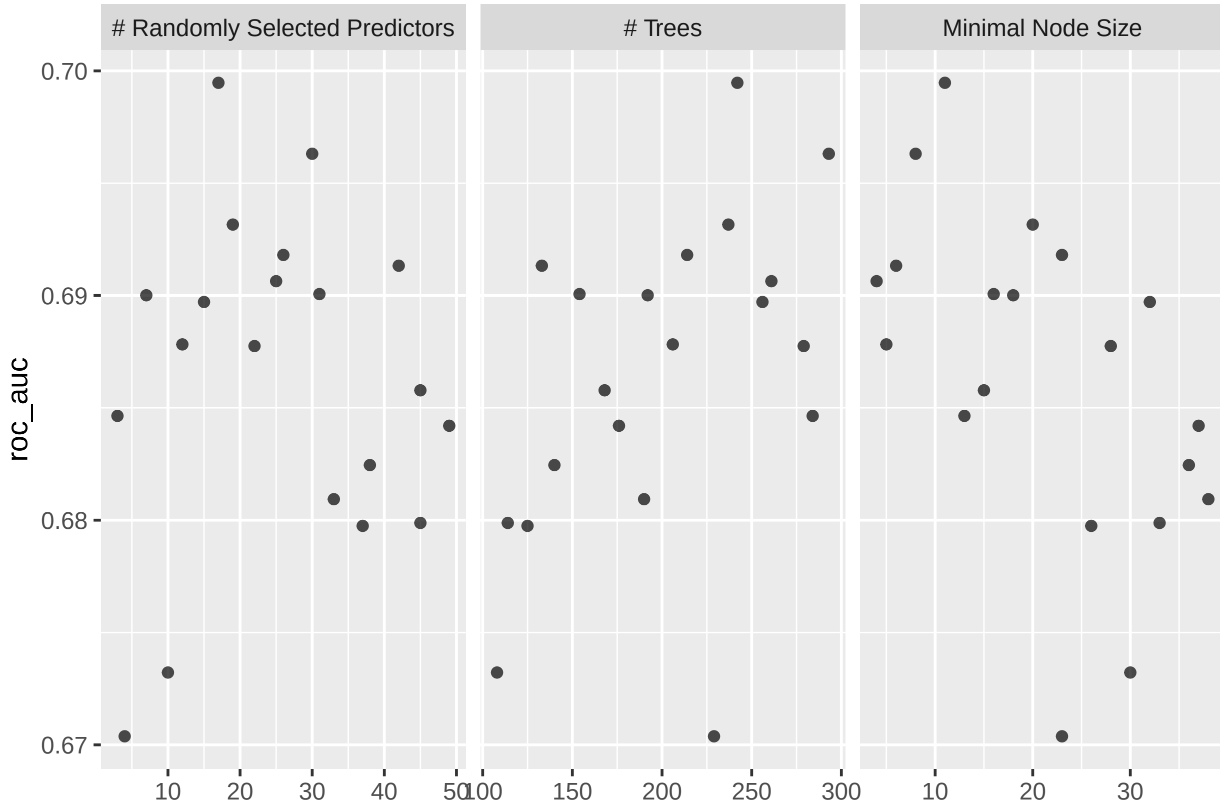
Random Forest

```
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()) %>%
```

```r
  set_mode("classification") %>%
  set_engine("randomForest",
             probability = TRUE,
             nthread = 8)

rf_grid <- grid_latin_hypercube(mtry(range = c(1L, 50L)),
                                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()
```

```
## # A tibble: 40 x 9
##     mtry trees min_n .metric  .estimator  mean     n std_err .config
##    <int> <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1    31   154    16 accuracy binary     0.940    10 0.00207 Preprocessor1_Model01
##  2    31   154    16 roc_auc  binary     0.690    10 0.0148  Preprocessor1_Model01
##  3    49   176    37 accuracy binary     0.940    10 0.00190 Preprocessor1_Model02
##  4    49   176    37 roc_auc  binary     0.684    10 0.0131  Preprocessor1_Model02
##  5    38   140    36 accuracy binary     0.940    10 0.00190 Preprocessor1_Model03
##  6    38   140    36 roc_auc  binary     0.682    10 0.0112  Preprocessor1_Model03
##  7    15   256    32 accuracy binary     0.941    10 0.00201 Preprocessor1_Model04
##  8    15   256    32 roc_auc  binary     0.690    10 0.0138  Preprocessor1_Model04
##  9    17   242    11 accuracy binary     0.941    10 0.00215 Preprocessor1_Model05
## 10    17   242    11 roc_auc  binary     0.699    10 0.0150  Preprocessor1_Model05
## # i 30 more rows
```

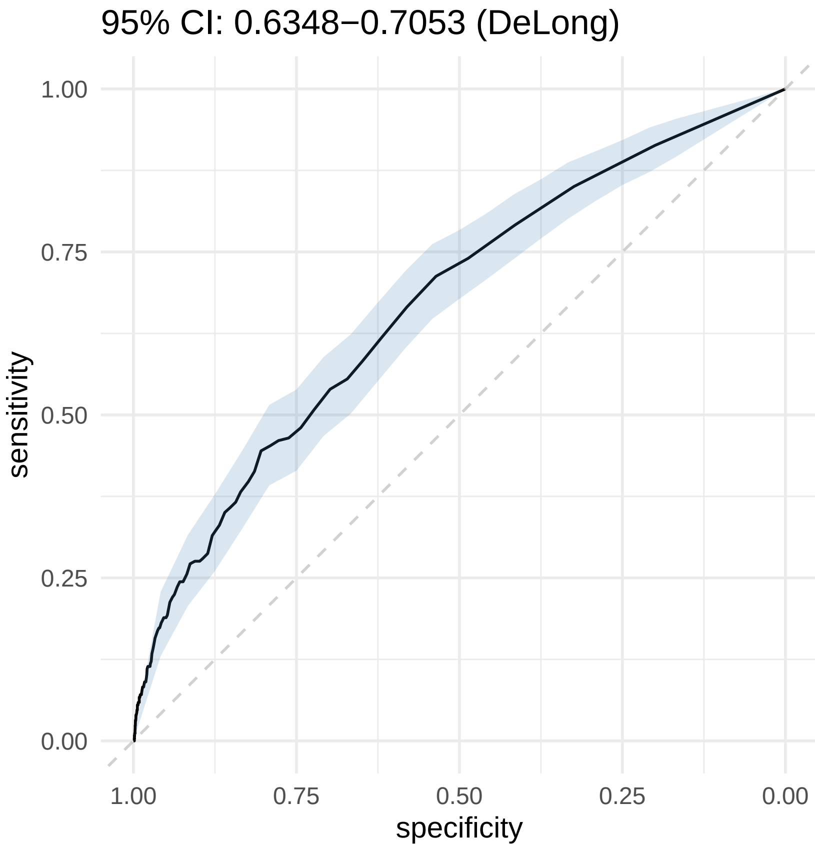
```r
autoplot(rf_tune, metric = "roc_auc")
```




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

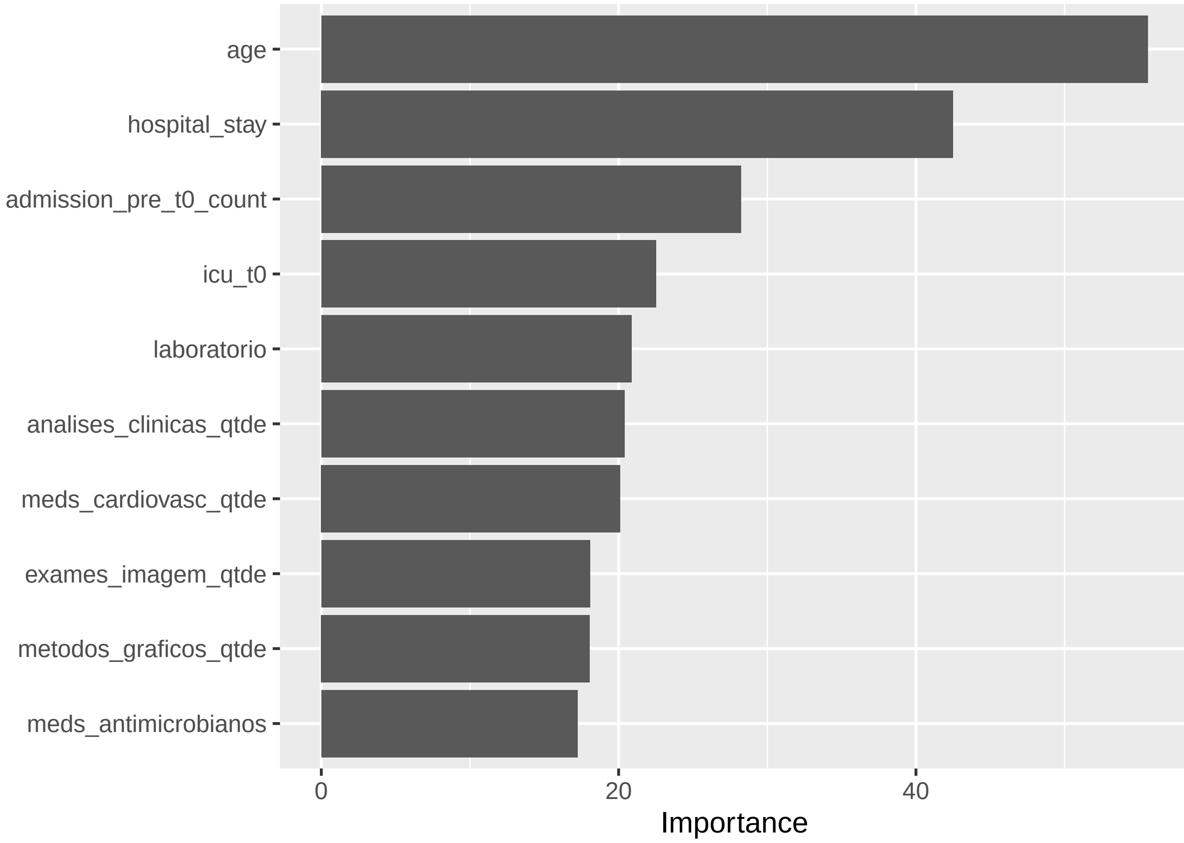
```
## # A tibble: 5 x 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    17   242    11 roc_auc binary     0.699    10  0.0150 Preprocessor1_Model05
## 2    30   293     8 roc_auc binary     0.696    10  0.0150 Preprocessor1_Model20
## 3    19   237    20 roc_auc binary     0.693    10  0.0152 Preprocessor1_Model06
## 4    26   214    23 roc_auc binary     0.692    10  0.0159 Preprocessor1_Model13
## 5    42   133     6 roc_auc binary     0.691    10  0.0136 Preprocessor1_Model18
```

```
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)
```




```
## [1] "Optimal Threshold: 0.06"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 3600  141
##    1  876  113
##
##                Accuracy : 0.785
##                  95% CI : (0.773, 0.7966)
##     No Information Rate : 0.9463
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1054
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.8043
##             Specificity : 0.4449
##          Pos Pred Value : 0.9623
##          Neg Pred Value : 0.1143
##              Prevalence : 0.9463
##          Detection Rate : 0.7611
##    Detection Prevalence : 0.7909
##       Balanced Accuracy : 0.6246
##
##        'Positive' Class : 0
##
```

```
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')
```

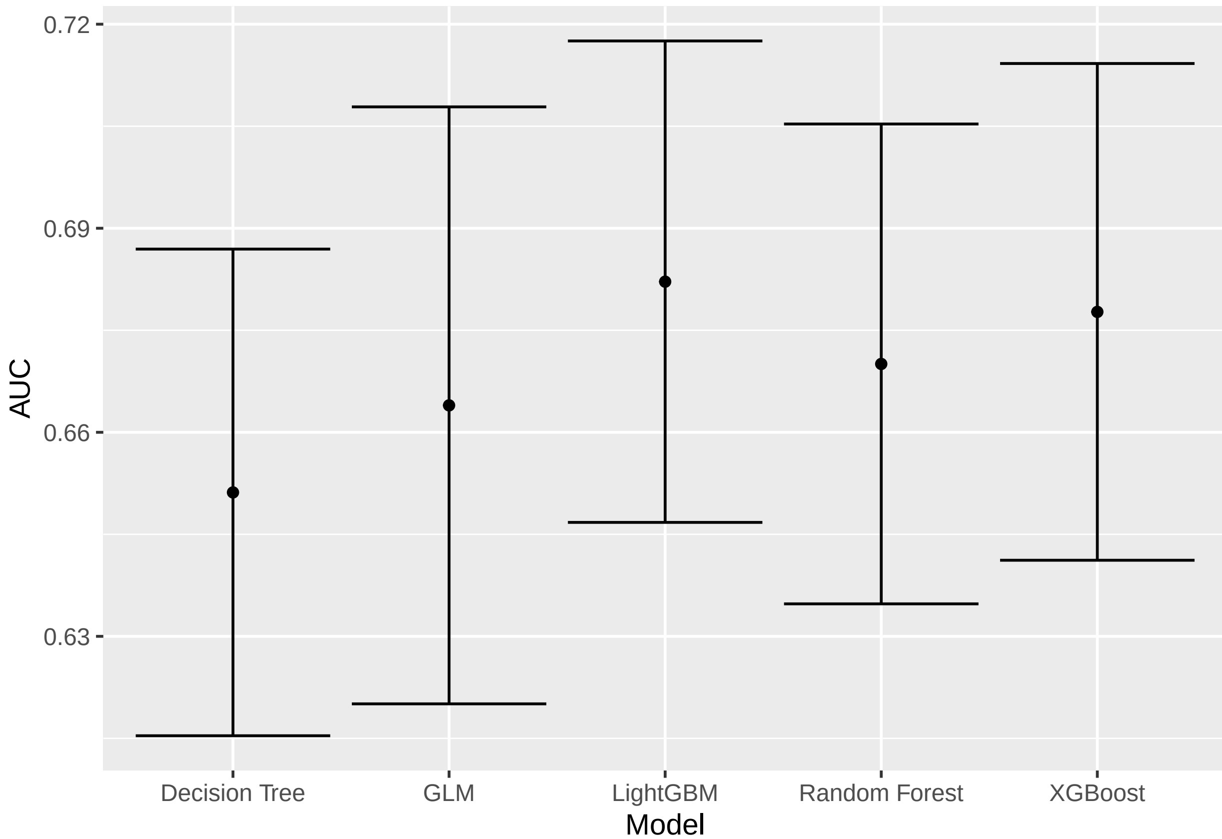



```r
# extract_vip(final_rf_fit, pred_wrapper = predict,
#             reference_class = "1", use_matrix = FALSE,
#             method = 'permute')
```

Minutes to run: 59.402

Models Comparison

```r
if (RUN_ALL_MODELS) {
  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$ci[3]
  ) %>%
    mutate(Target = outcome_column)
} else {
  df_auc <- tibble::tribble(
    ~Model, ~`AUC`, ~`Lower Limit`, ~`Upper Limit`,
    'LightGBM', as.numeric(lightgbm_auc$auc), lightgbm_auc$ci[1], lightgbm_auc$ci[3]
  ) %>%
    mutate(Target = outcome_column)
}

df_auc %>%
  ggplot(aes(x = Model, y = AUC, ymin = `Lower Limit`, ymax = `Upper Limit`)) +
    geom_point() +
    geom_errorbar()
```




```
write_csv(df_auc, sprintf("./auxiliar/model_selection/performance/%s.csv", outcome_column))
```

Minutes to run: 0.01