Model Selection - death_30days

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= death_30days
## .  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

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: 0.006

```
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

```
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

```
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. 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. comorbidities_count
## 16. procedure_type_1
## 17. reop_type_1
## 18. procedure_type_new
## 19. cied_final_1
## 20. cied_final_group_1
## 21. admission_pre_t0_count
## 22. admission_pre_t0_180d
## 23. year_adm_t0
## 24. icu_t0
## 25. antiarritmico
## 26. antihipertensivo
## 27. betabloqueador
## 28. dva
## 29. diuretico
## 30. vasodilatador
## 31. espironolactona
## 32. antiplaquetario_ev
## 33. insulina
## 34. psicofarmacos
## 35. antifungico
## 36. classe_meds_qtde
## 37. meds_cardiovasc_qtde
## 38. meds_antimicrobianos
## 39. vni
## 40. ventilacao_mecanica
## 41. intervencao_cv
## 42. cateter_venoso_central
## 43. proced_invasivos_qtde
## 44. transfusao
## 45. interconsulta
## 46. equipe_multiprof
## 47. holter
## 48. metodos_graficos_qtde
## 49. laboratorio
## 50. cultura
## 51. analises_clinicas_qtde
## 52. citologia
## 53. histopatologia_qtde
## 54. angio_tc
## 55. angiografia
## 56. cintilografia
## 57. ecocardiograma
## 58. flebografia
## 59. ultrassom
## 60. tomografia
## 61. ressonancia
## 62. exames_imagem_qtde
## 63. bic
## 64. 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))
```

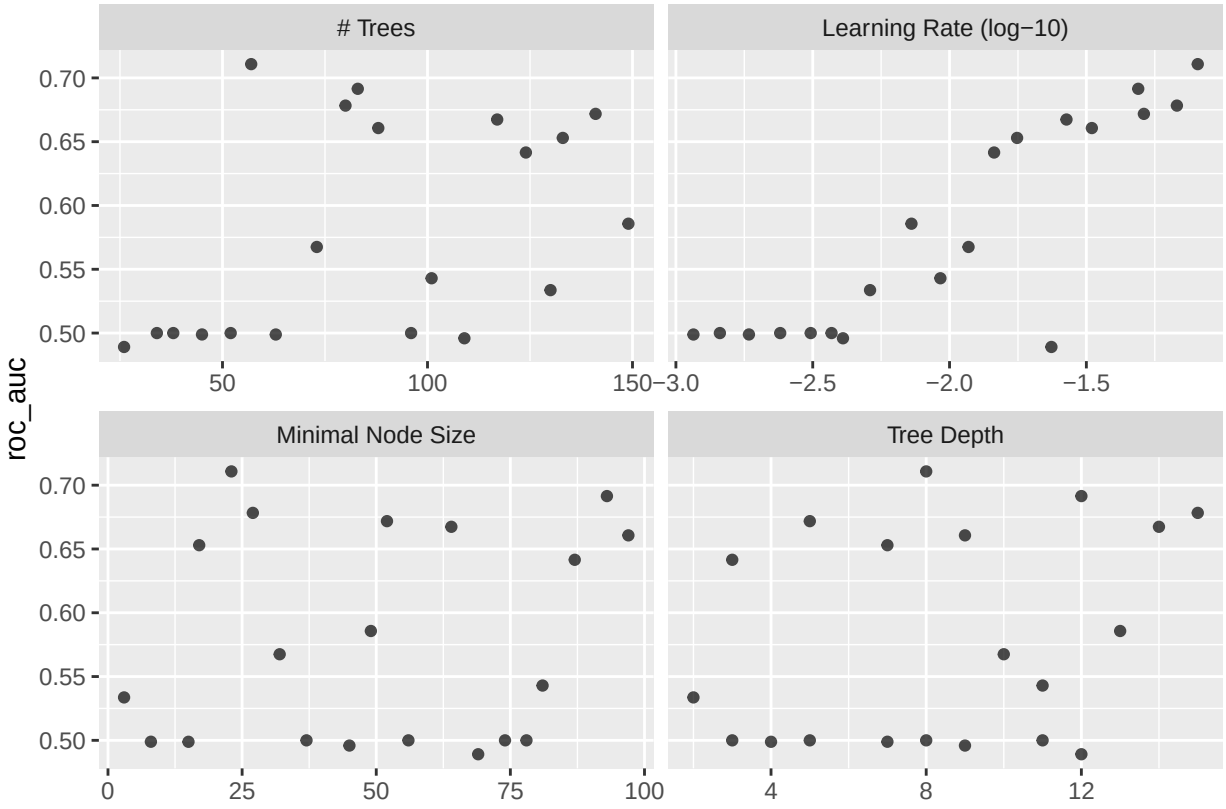
Minutes to run: 0.001

Boosted Tree (XGBoost)

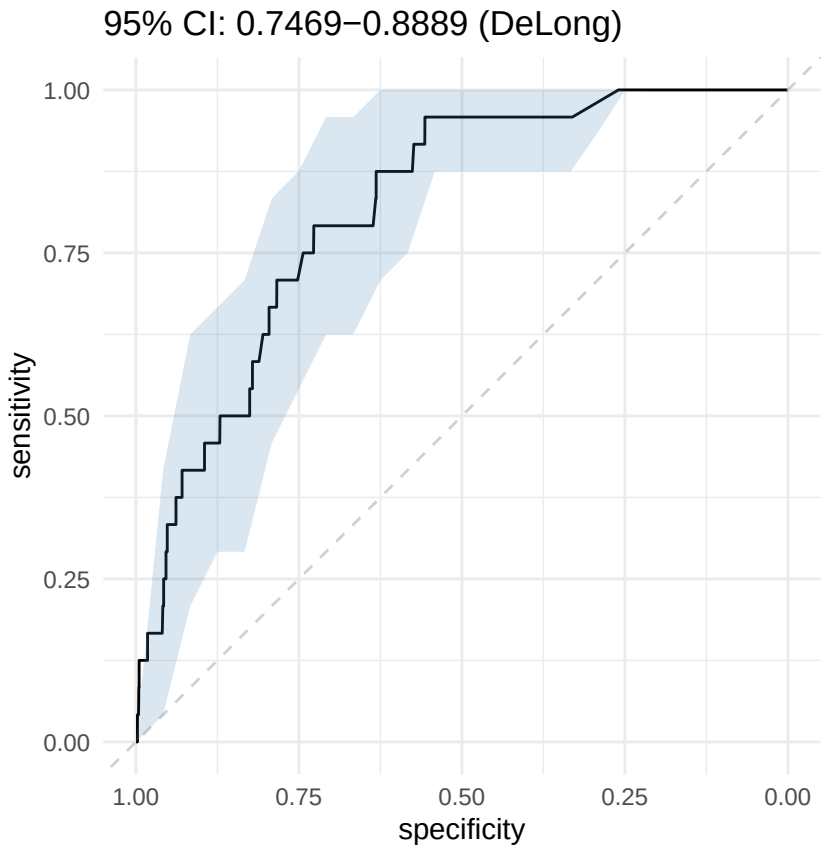
```
xgboost_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_other(all_nominal_predictors(), threshold = 0.05, other = ".merged") %>%
  step_dummy(all_nominal_predictors())

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() %>%
  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    57    23          8     0.0807 roc_auc binary     0.711    10  0.0561 Prepro~
## 2    83    93         12     0.0489 roc_auc binary     0.691    10  0.0277 Prepro~
## 3    80    27         15     0.0677 roc_auc binary     0.678    10  0.0662 Prepro~
## 4   141    52          5     0.0513 roc_auc binary     0.672    10  0.0479 Prepro~
## 5   117    64         14     0.0268 roc_auc binary     0.667    10  0.0450 Prepro~
```

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

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

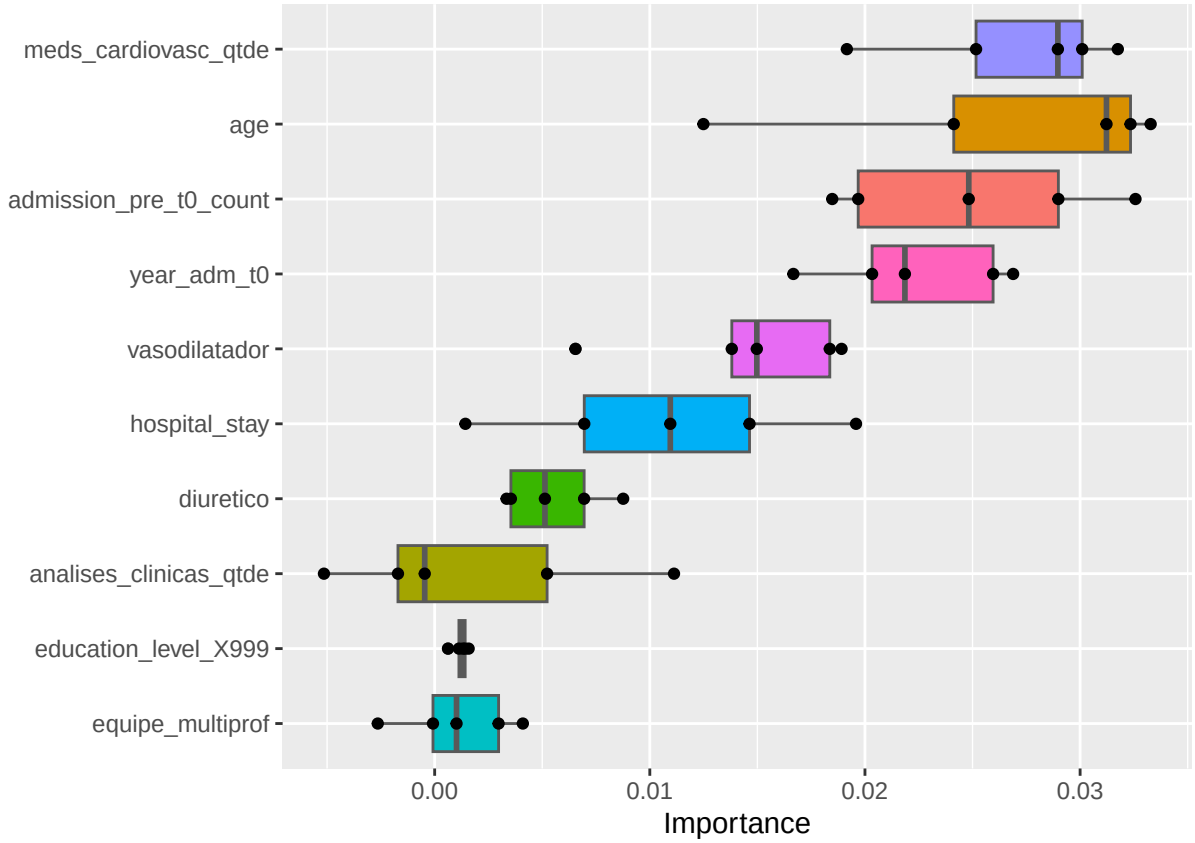



```r
final_xgboost_workflow <-
  xgboost_workflow %>%
  finalize_workflow(best_xgboost)

last_xgboost_fit <-
  final_xgboost_workflow %>%
  last_fit(df_split)

final_xgboost_fit <- extract_workflow(last_xgboost_fit)

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




```
## [1] "Optimal Threshold: 0.01"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 3422    5
##    1 1284   19
##
##                Accuracy : 0.7275
##                  95% CI : (0.7146, 0.7401)
##     No Information Rate : 0.9949
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0189
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.72716
##             Specificity : 0.79167
##          Pos Pred Value : 0.99854
##          Neg Pred Value : 0.01458
##              Prevalence : 0.99493
##          Detection Rate : 0.72347
##    Detection Prevalence : 0.72452
##       Balanced Accuracy : 0.75941
##
##        '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(-.metric, -.estimator, -.config, -mean, -n, -std_err) %>%
  as.list
```

Minutes to run: 1.835

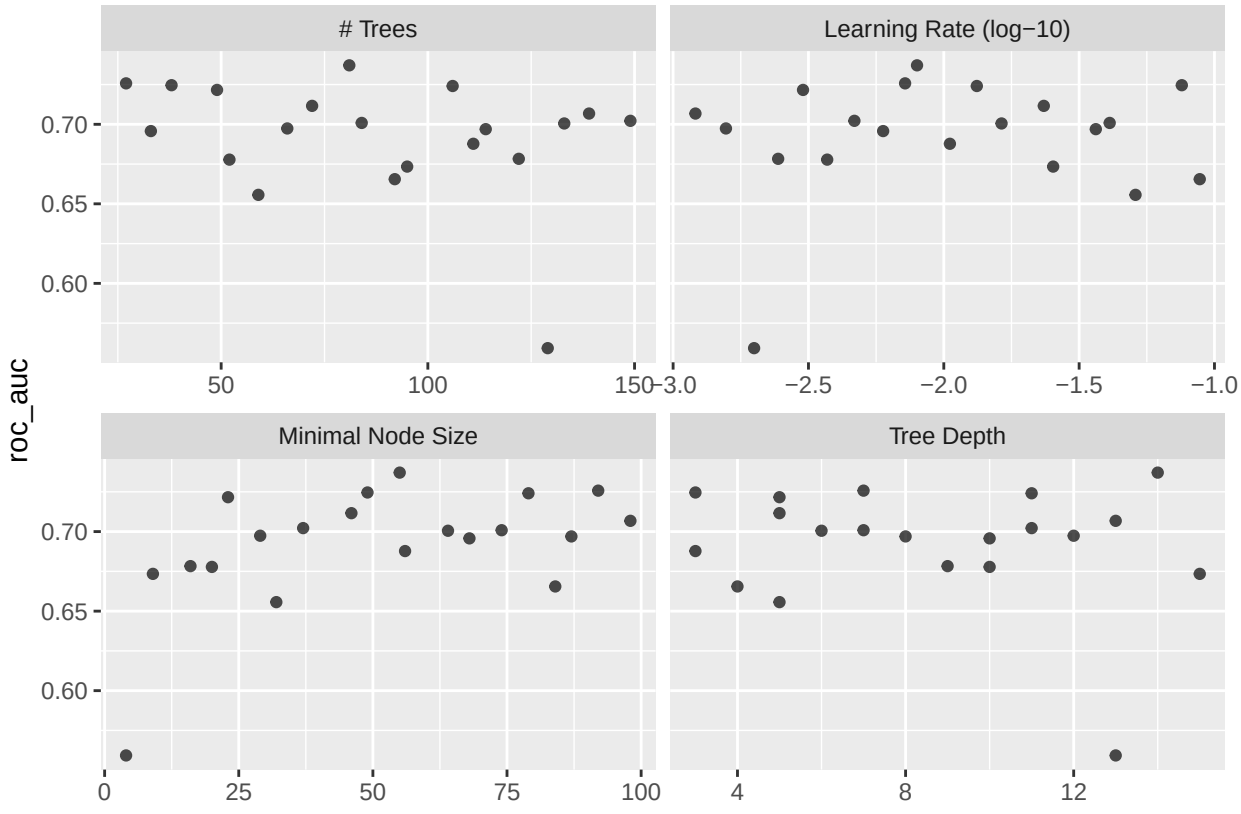
Boosted Tree (LightGBM)

```
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
```

```
)

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")
```

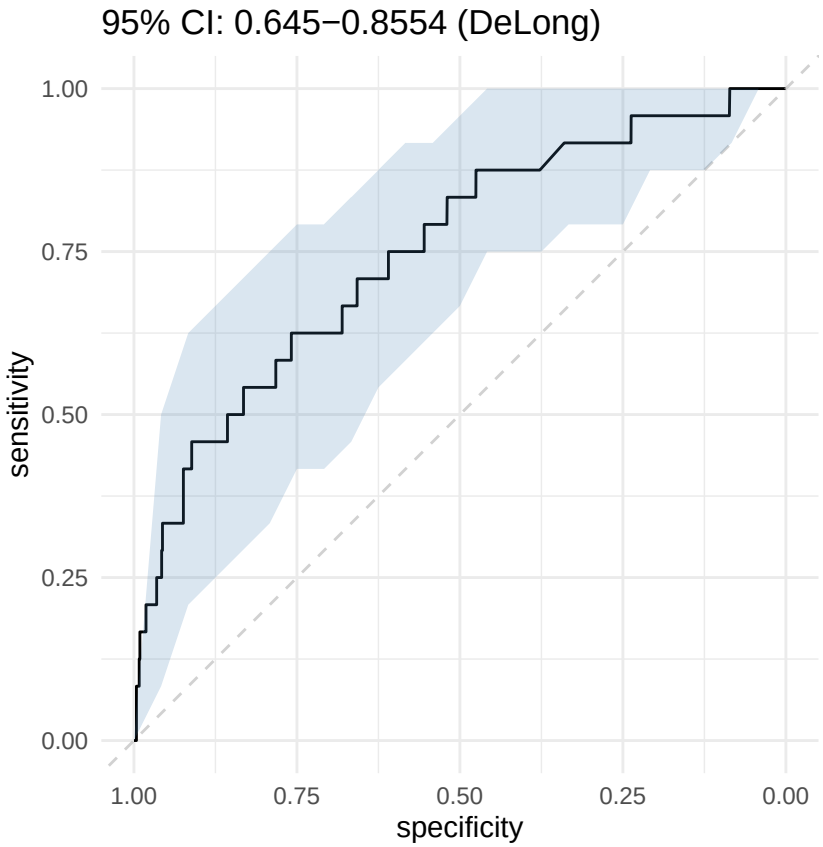
```
## # A tibble: 5 x 10
##   trees min_n tree_depth learn_rate .metric .estimator  mean     n std_err .config
##   <int> <int>      <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    81    55         14    0.00796 roc_auc binary     0.737    10  0.0470 Prepro~
## 2    27    92          7    0.00720 roc_auc binary     0.726    10  0.0631 Prepro~
## 3    38    49          3    0.0757  roc_auc binary     0.725    10  0.0783 Prepro~
## 4   106    79         11    0.0132  roc_auc binary     0.724    10  0.0559 Prepro~
## 5    49    23          5    0.00302 roc_auc binary     0.722    10  0.0518 Prepro~
```

```
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 3570    9
##    1 1136   15
##
##                Accuracy : 0.7579
##                  95% CI : (0.7455, 0.7701)
##     No Information Rate : 0.9949
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0157
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.75861
##             Specificity : 0.62500
##          Pos Pred Value : 0.99749
##          Neg Pred Value : 0.01303
##              Prevalence : 0.99493
##          Detection Rate : 0.75476
##    Detection Prevalence : 0.75666
##       Balanced Accuracy : 0.69180
##
```

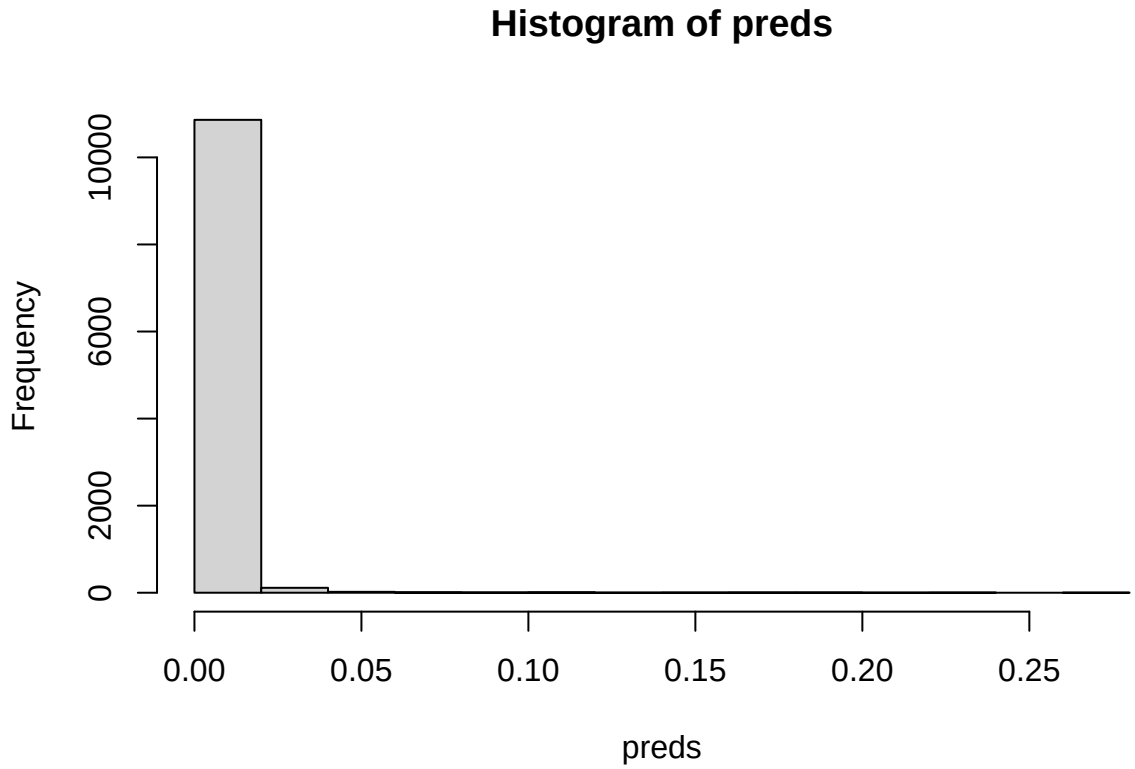
```
##        'Positive' Class : 0
##
```

```
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= 81
## .  min_n = integer 1= 55
## .  tree_depth = integer 1= 14
## .  learn_rate = double 1= 0.0079649
```

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

Minutes to run: 3.17

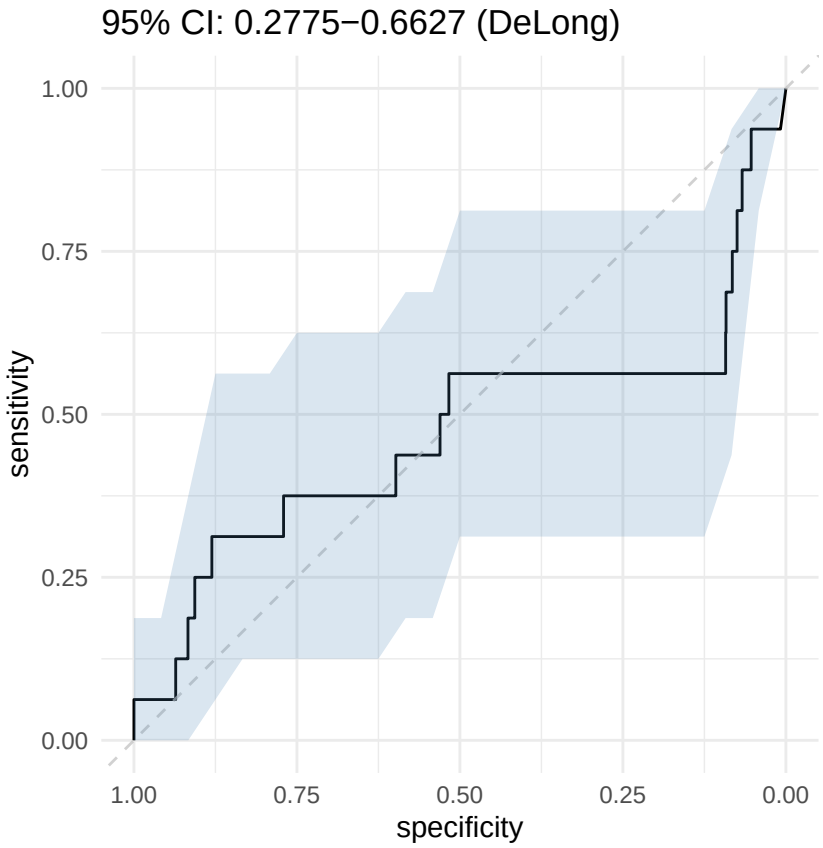


Minutes to run: 0.004

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())
```

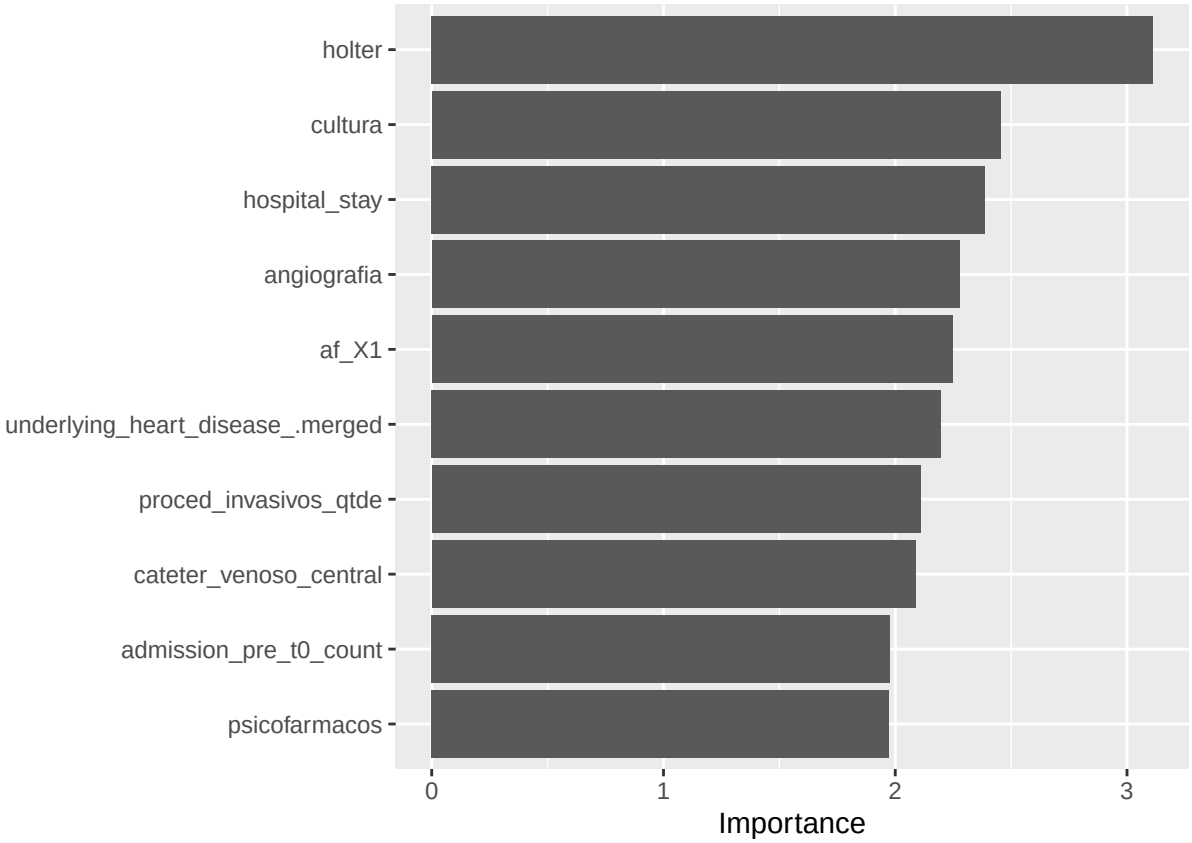
```r
glmnet_spec <-
  logistic_reg(penalty = 0) %>%
  set_mode("classification") %>%
  set_engine("glm")

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: 0.01"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 2858   11
##    1  388    5
##
##                Accuracy : 0.8777
##                  95% CI : (0.8659, 0.8887)
##     No Information Rate : 0.9951
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0152
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.88047
```

```
##             Specificity : 0.31250
##          Pos Pred Value : 0.99617
##          Neg Pred Value : 0.01272
##              Prevalence : 0.99510
##          Detection Rate : 0.87615
##    Detection Prevalence : 0.87952
##       Balanced Accuracy : 0.59648
##
##        '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: 0.191

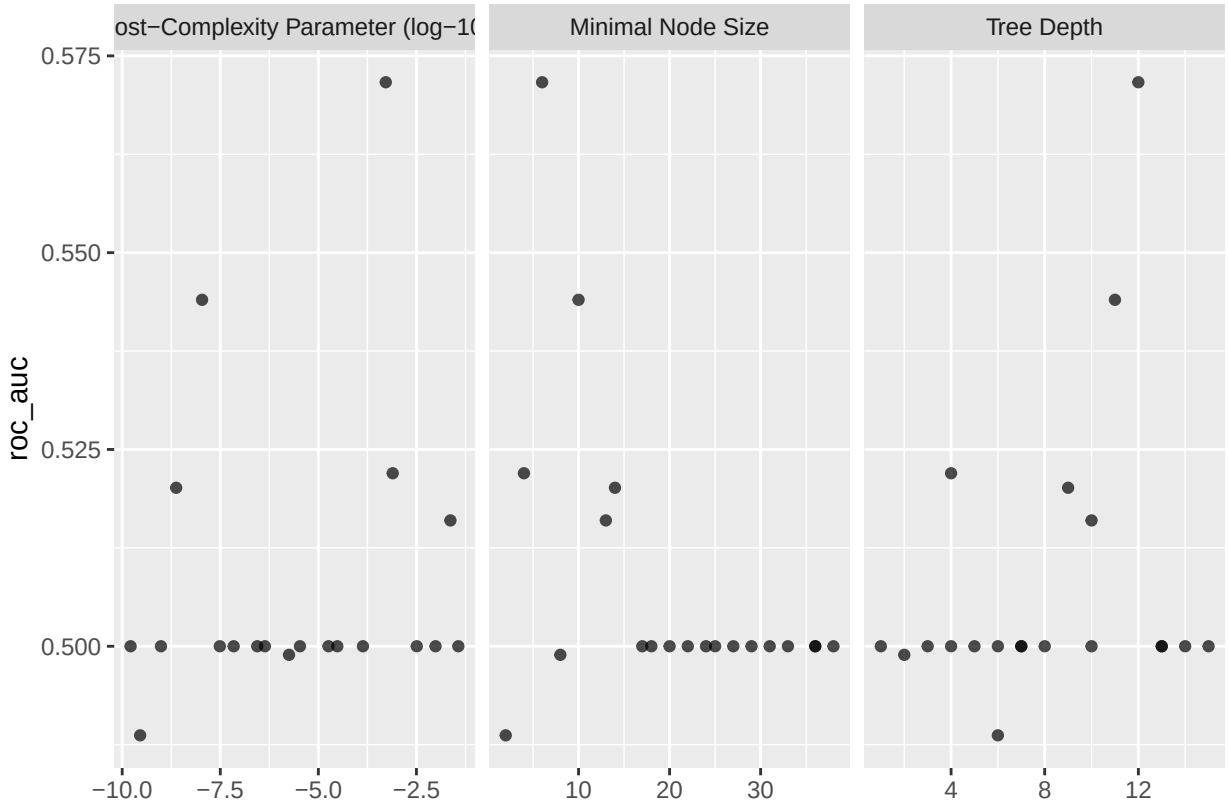
Decision Tree

```
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()
```

```
## # 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   0.0000305              1    18 accura~ binary     0.996    10 6.89e-4 Prepro~
##  2   0.0000305              1    18 roc_auc binary     0.5      10 0       Prepro~
##  3   0.00322               10    36 accura~ binary     0.996    10 6.89e-4 Prepro~
##  4   0.00322               10    36 roc_auc binary     0.5      10 0       Prepro~
##  5   0.00000000238          9    14 accura~ binary     0.996    10 6.94e-4 Prepro~
##  6   0.00000000238          9    14 roc_auc binary     0.520    10 1.65e-2 Prepro~
##  7   0.0000182             13    38 accura~ binary     0.996    10 6.89e-4 Prepro~
##  8   0.0000182             13    38 roc_auc binary     0.5      10 0       Prepro~
##  9   0.00000179             2     8 accura~ binary     0.995    10 6.62e-4 Prepro~
## 10   0.00000179             2     8 roc_auc binary     0.499    10 3.96e-4 Prepro~
## # i 30 more rows
```

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

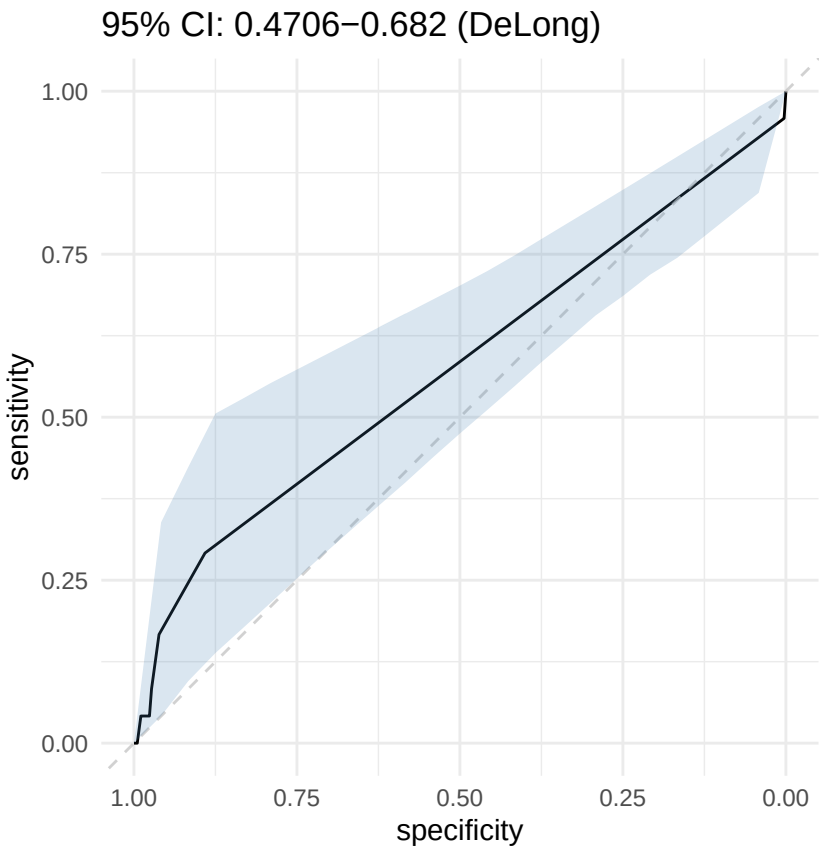



```
tree_tune %>%
  show_best("roc_auc")
```

```
## # 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   0.000523              12     6 roc_auc binary     0.572    10  0.0498 Preproc~
## 2   0.0000000109          11    10 roc_auc binary     0.544    10  0.0357 Preproc~
## 3   0.000786               4     4 roc_auc binary     0.522    10  0.0183 Preproc~
## 4   0.00000000238          9    14 roc_auc binary     0.520    10  0.0165 Preproc~
## 5   0.0232                10    13 roc_auc binary     0.516    10  0.0164 Preproc~
```

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

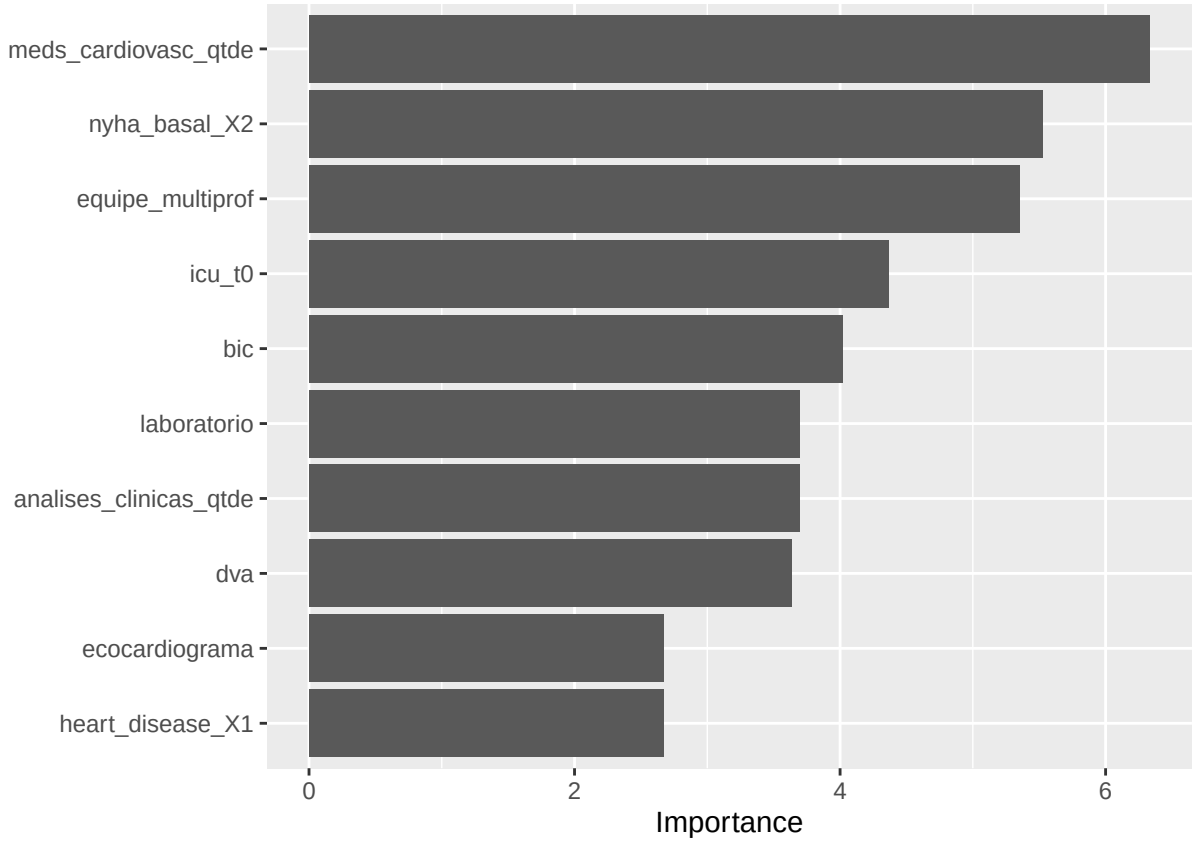
```r
final_tree_workflow <-
  tree_workflow %>%
  finalize_workflow(best_tree)

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: 0.00"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 4193   17
##    1  513    7
##
##                Accuracy : 0.8879
##                  95% CI : (0.8786, 0.8968)
##     No Information Rate : 0.9949
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0162
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.89099
##             Specificity : 0.29167
##          Pos Pred Value : 0.99596
##          Neg Pred Value : 0.01346
```

```
##              Prevalence : 0.99493
##          Detection Rate : 0.88647
##    Detection Prevalence : 0.89006
##       Balanced Accuracy : 0.59133
##
##        'Positive' Class : 0
##
```

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




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

Minutes to run: 1.678

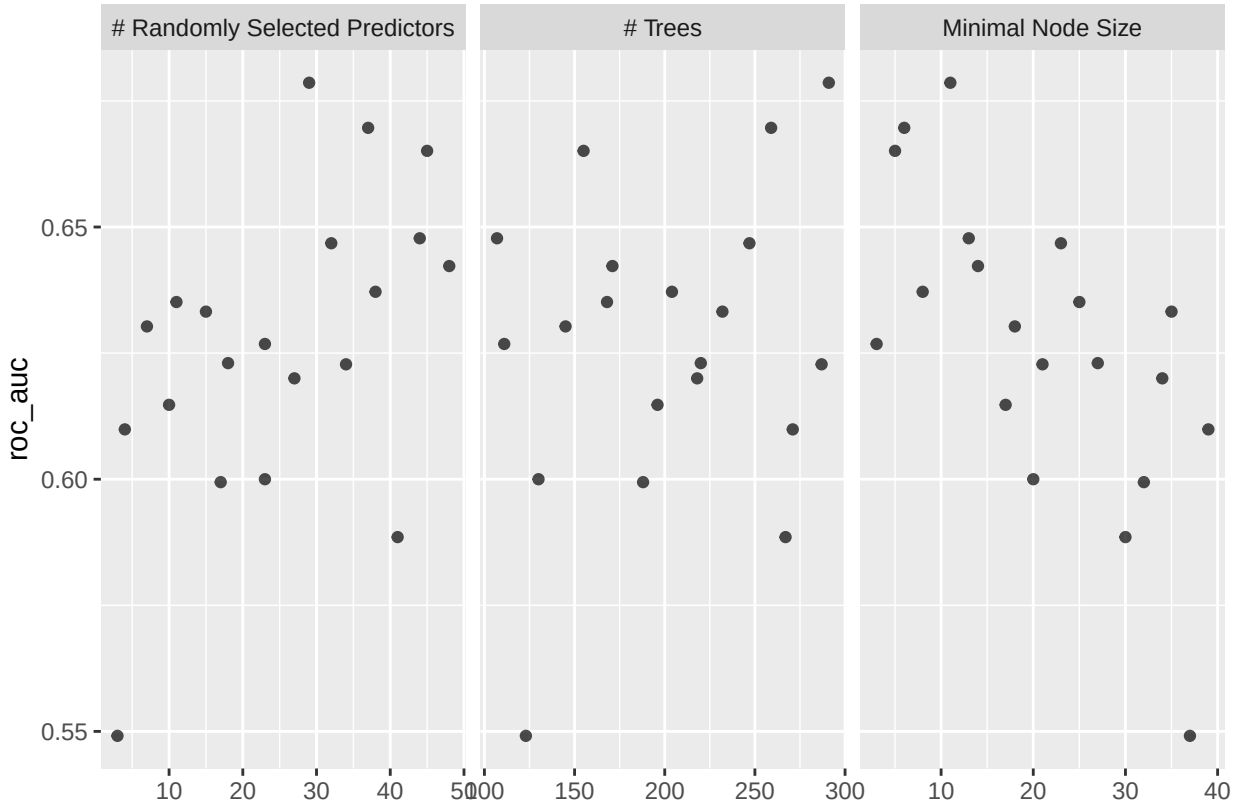
Random Forest

```r
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, 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    45   155     5 accuracy binary     0.996    10 0.000689 Preprocessor1_Model~
##  2    45   155     5 roc_auc  binary     0.665    10 0.0456   Preprocessor1_Model~
##  3    10   196    17 accuracy binary     0.996    10 0.000689 Preprocessor1_Model~
##  4    10   196    17 roc_auc  binary     0.615    10 0.0344   Preprocessor1_Model~
##  5    44   107    13 accuracy binary     0.996    10 0.000689 Preprocessor1_Model~
##  6    44   107    13 roc_auc  binary     0.648    10 0.0456   Preprocessor1_Model~
##  7     3   123    37 accuracy binary     0.996    10 0.000689 Preprocessor1_Model~
##  8     3   123    37 roc_auc  binary     0.549    10 0.0280   Preprocessor1_Model~
##  9    29   291    11 accuracy binary     0.996    10 0.000689 Preprocessor1_Model~
## 10    29   291    11 roc_auc  binary     0.679    10 0.0514   Preprocessor1_Model~
## # i 30 more rows
```

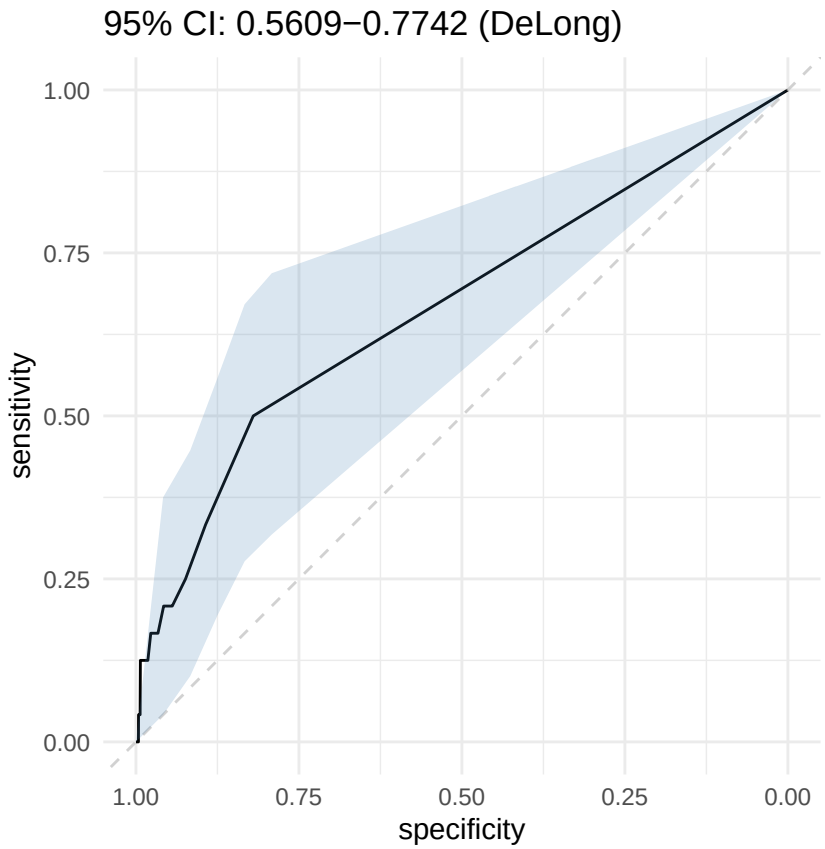
```
autoplot(rf_tune, metric = "roc_auc")
```




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

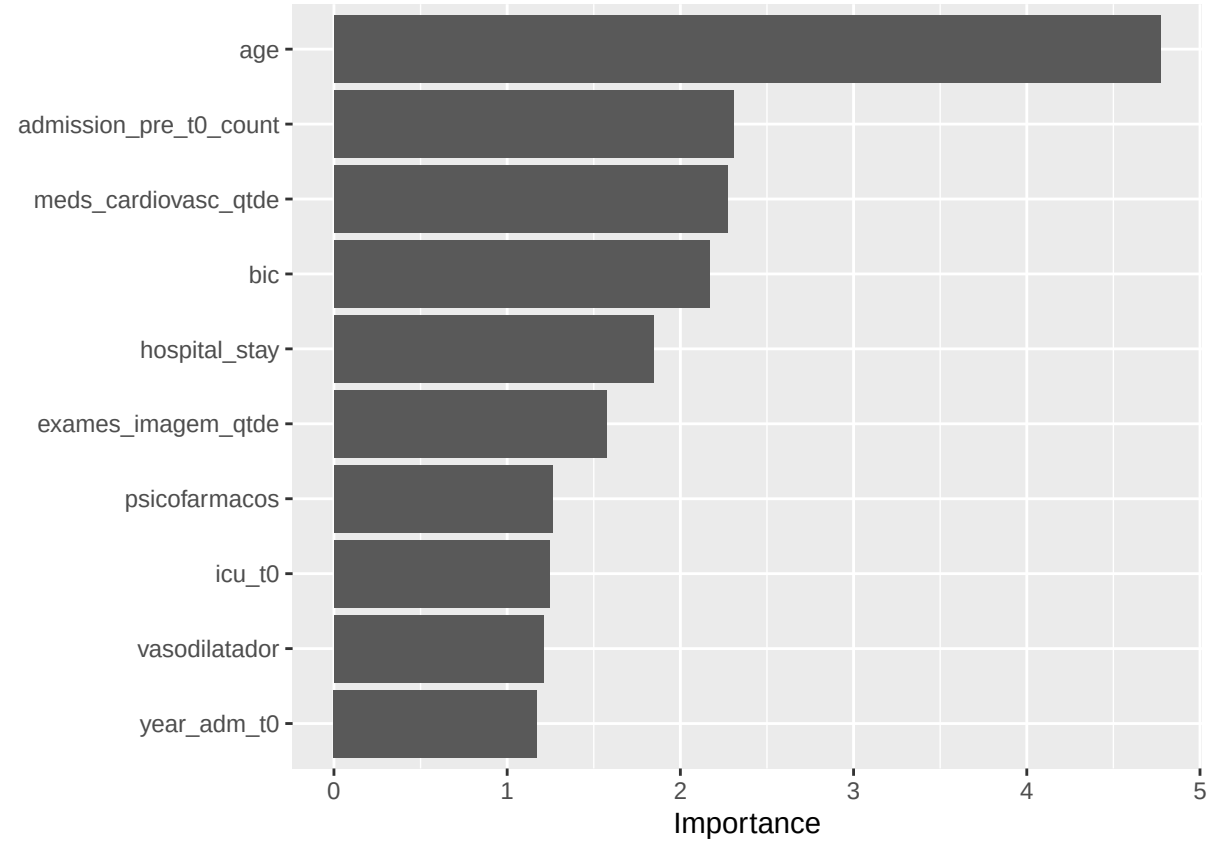
```
## # A tibble: 5 x 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    29   291    11 roc_auc binary     0.679    10  0.0514 Preprocessor1_Model05
## 2    37   259     6 roc_auc binary     0.670    10  0.0519 Preprocessor1_Model20
## 3    45   155     5 roc_auc binary     0.665    10  0.0456 Preprocessor1_Model01
## 4    44   107    13 roc_auc binary     0.648    10  0.0456 Preprocessor1_Model03
## 5    32   247    23 roc_auc binary     0.647    10  0.0420 Preprocessor1_Model16
```

```
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: 0.00"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 3859   12
##    1  847   12
##
##                Accuracy : 0.8184
##                  95% CI : (0.8071, 0.8293)
##     No Information Rate : 0.9949
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0175
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.82002
##             Specificity : 0.50000
##          Pos Pred Value : 0.99690
##          Neg Pred Value : 0.01397
##              Prevalence : 0.99493
##          Detection Rate : 0.81586
##    Detection Prevalence : 0.81839
##       Balanced Accuracy : 0.66001
##
##        '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')
```

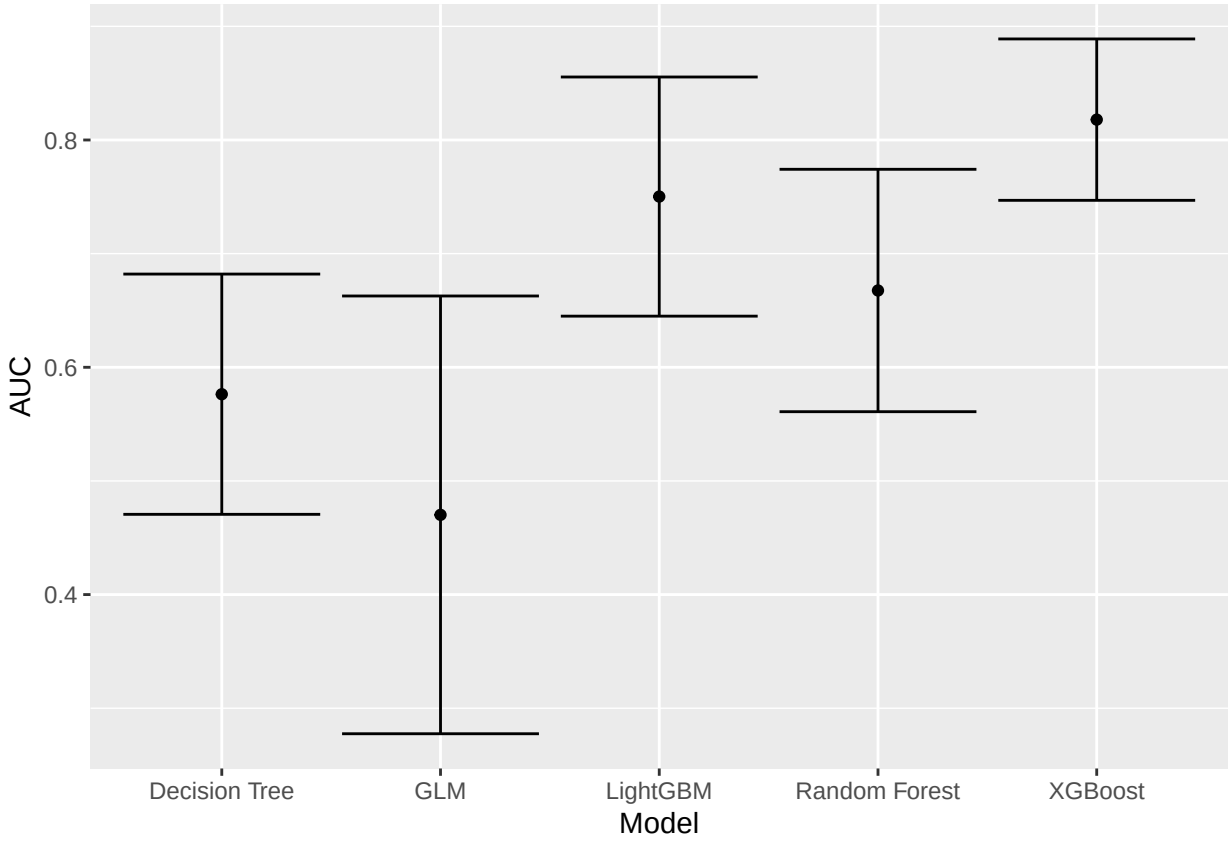



```
# extract_vip(final_rf_fit, pred_wrapper = predict,
#             reference_class = "1", use_matrix = FALSE,
#             method = 'permute')
```

Minutes to run: 36.76

Models Comparison

```
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: 0.002