Model Selection - death_180days

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_180days
## .  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. cardiac_arrest
## 12. valvopathy
## 13. diabetes
```

```
## 14. renal_failure
## 15. hemodialysis
## 16. stroke
## 17. copd
## 18. cancer
## 19. comorbidities_count
## 20. procedure_type_1
## 21. reop_type_1
## 22. procedure_type_new
## 23. cied_final_1
## 24. cied_final_group_1
## 25. admission_pre_t0_count
## 26. admission_pre_t0_180d
## 27. year_adm_t0
## 28. icu_t0
## 29. dialysis_t0
## 30. admission_t0_emergency
## 31. aco
## 32. antiarritmico
## 33. ieca_bra
## 34. dva
## 35. digoxina
## 36. estatina
## 37. diuretico
## 38. vasodilatador
## 39. insuf_cardiaca
## 40. espironolactona
## 41. antiplaquetario_ev
## 42. insulina
## 43. psicofarmacos
## 44. antifungico
## 45. classe_meds_qtde
## 46. meds_cardiovasc_qtde
## 47. meds_antimicrobianos
## 48. vni
## 49. ventilacao_mecanica
## 50. transplante_cardiaco
## 51. outros_proced_cirurgicos
## 52. icp
## 53. cateterismo
## 54. cateter_venoso_central
## 55. proced_invasivos_qtde
## 56. transfusao
## 57. interconsulta
## 58. equipe_multiprof
## 59. holter
## 60. teste_esforco
## 61. metodos_graficos_qtde
## 62. laboratorio
## 63. cultura
## 64. analises_clinicas_qtde
## 65. citologia
## 66. histopatologia_qtde
## 67. angiografia
## 68. aortografia
## 69. arteriografia
## 70. cintilografia
## 71. ecocardiograma
## 72. endoscopia
## 73. ultrassom
## 74. tomografia
```

```
## 75. ressonancia
## 76. exames_imagem_qtde
## 77. bic
## 78. 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: 0.004

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    81     6          6     0.0714 roc_auc binary     0.794    10 0.00832 Prepro~
## 2   117    49          7     0.0345 roc_auc binary     0.777    10 0.00732 Prepro~
## 3   140    63          4     0.0618 roc_auc binary     0.775    10 0.00668 Prepro~
## 4    83    77         15     0.0967 roc_auc binary     0.771    10 0.00763 Prepro~
## 5   131    97         11     0.0307 roc_auc binary     0.761    10 0.00743 Prepro~
```

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

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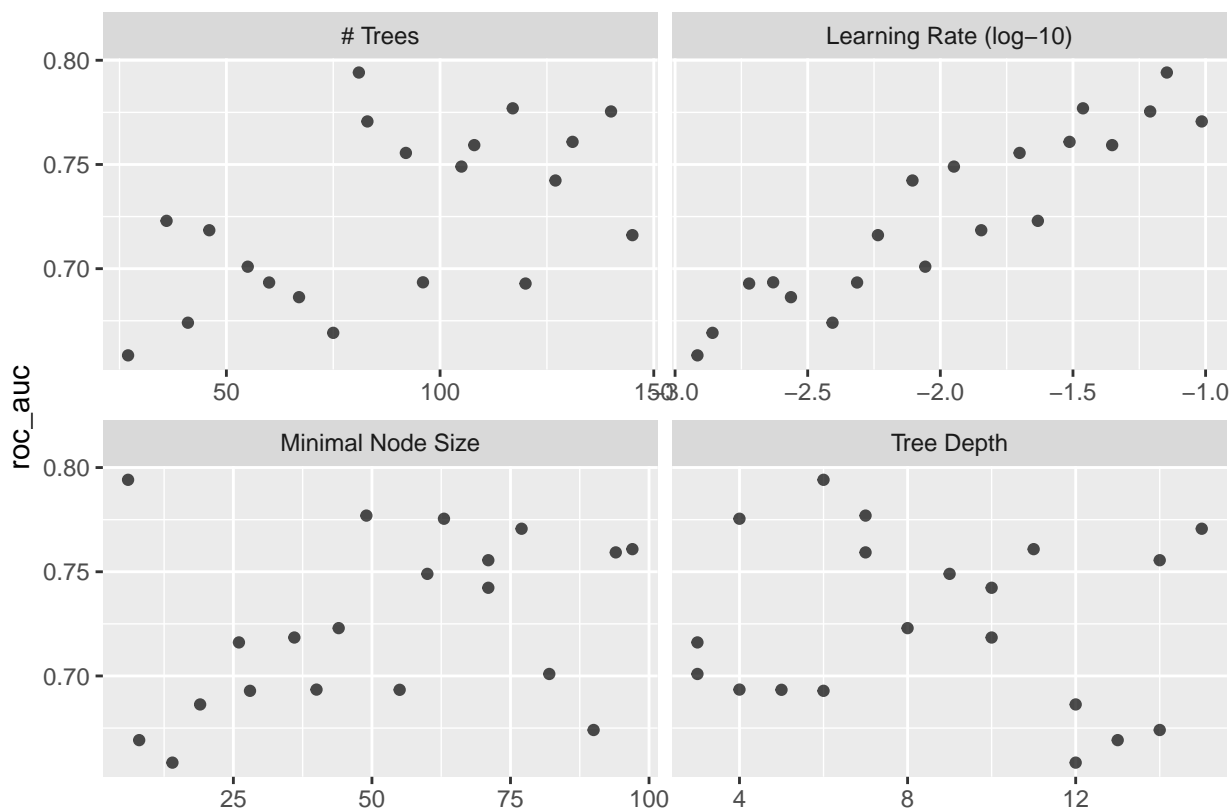


```
final_xgboost_workflow <-
  xgboost_workflow %>%
  finalize_workflow(best_xgboost)

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)
```

95% CI: 0.7995–0.8722 (DeLong)
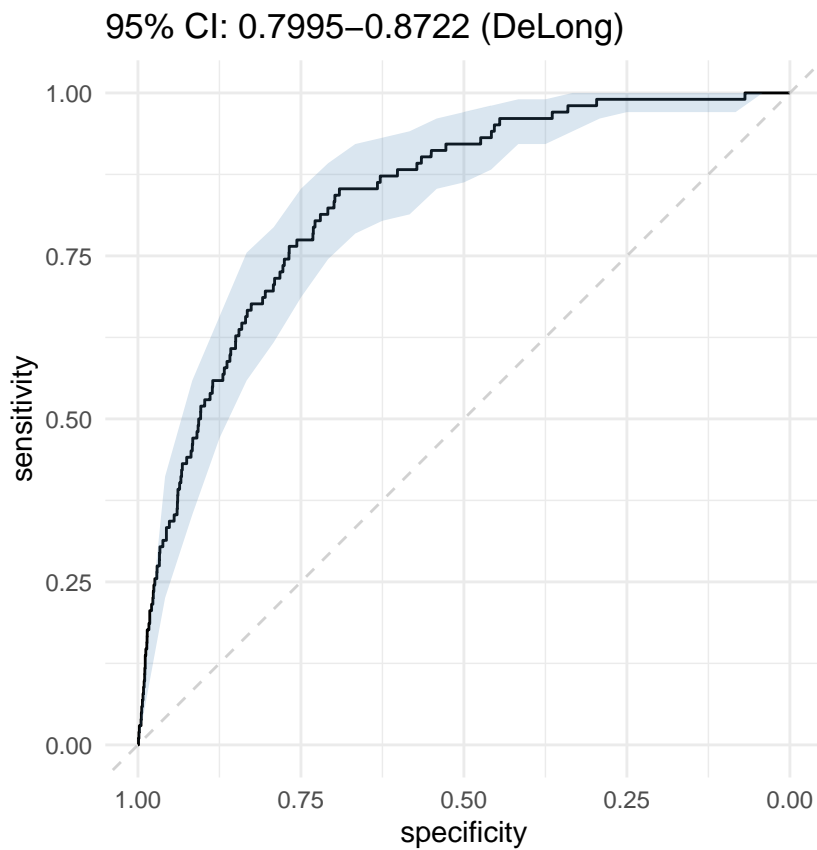

```
## [1] "Optimal Threshold: 0.02"
## Confusion Matrix and Statistics
##
##      reference
## data    0    1
##    0 3198   15
##    1 1430   87
##
##               Accuracy : 0.6945
##                 95% CI : (0.6812, 0.7076)
##    No Information Rate : 0.9784
##    P-Value [Acc > NIR] : 1
##
##                  Kappa : 0.0699
##
##  Mcnemar's Test P-Value : <2e-16
##
##            Sensitivity : 0.69101
##            Specificity : 0.85294
##         Pos Pred Value : 0.99533
##         Neg Pred Value : 0.05735
##             Prevalence : 0.97844
##         Detection Rate : 0.67611
##   Detection Prevalence : 0.67928
##      Balanced Accuracy : 0.77198
##
##       'Positive' Class : 0
##
```

```r
extract_vip(final_xgboost_fit, pred_wrapper = predict,
            reference_class = "0")
```
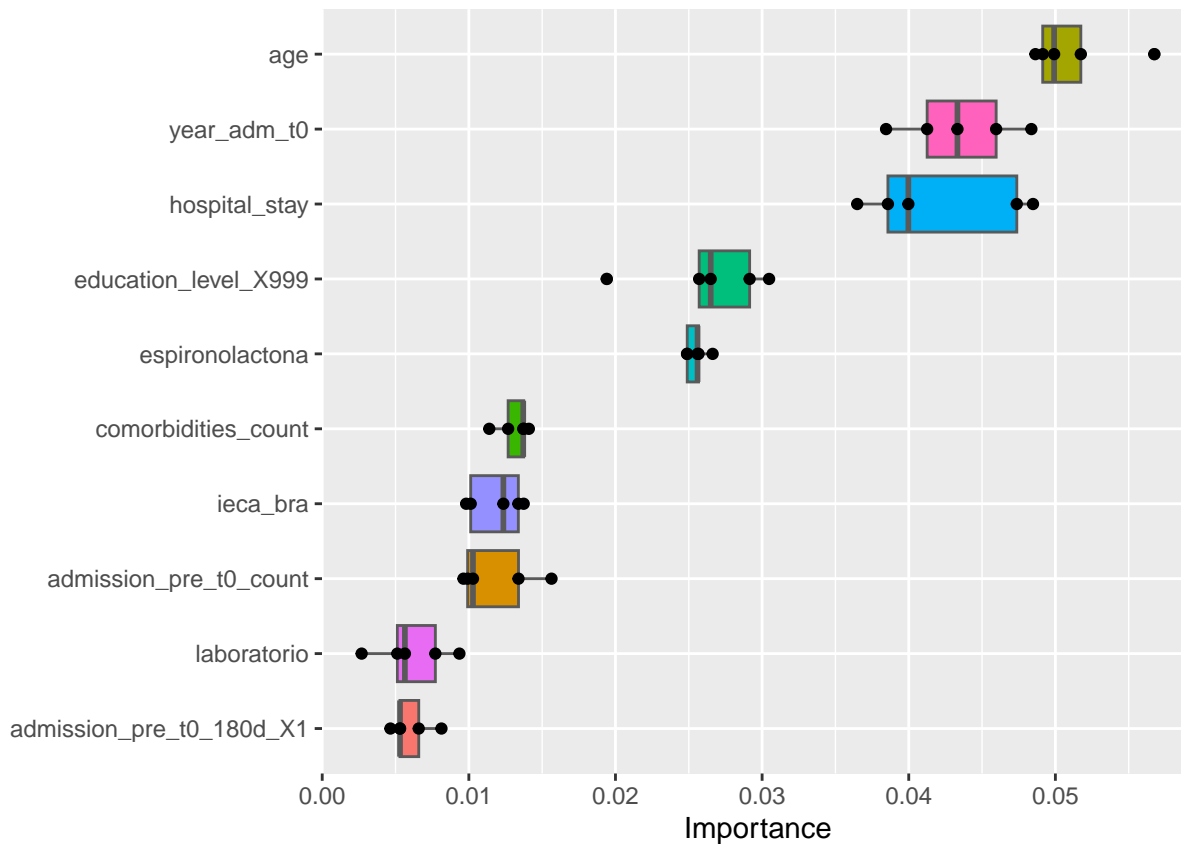
```
xgboost_parameters <- xgboost_tune %>%
  show_best("roc_auc", n = 1) %>%
  select(-.metric, -.estimator, -.config, -mean, -n, -std_err) %>%
  as.list
```

Minutes to run: 3.119

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     98    94          5     0.0213 roc_auc binary     0.795    10 0.00783 Prepro~
## 2     44    26          7     0.0356 roc_auc binary     0.788    10 0.00979 Prepro~
## 3    105    67          6     0.0103 roc_auc binary     0.784    10 0.00593 Prepro~
## 4    140    83         10     0.0130 roc_auc binary     0.784    10 0.00616 Prepro~
## 5     52    53         14     0.0258 roc_auc binary     0.783    10 0.00737 Prepro~
```

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

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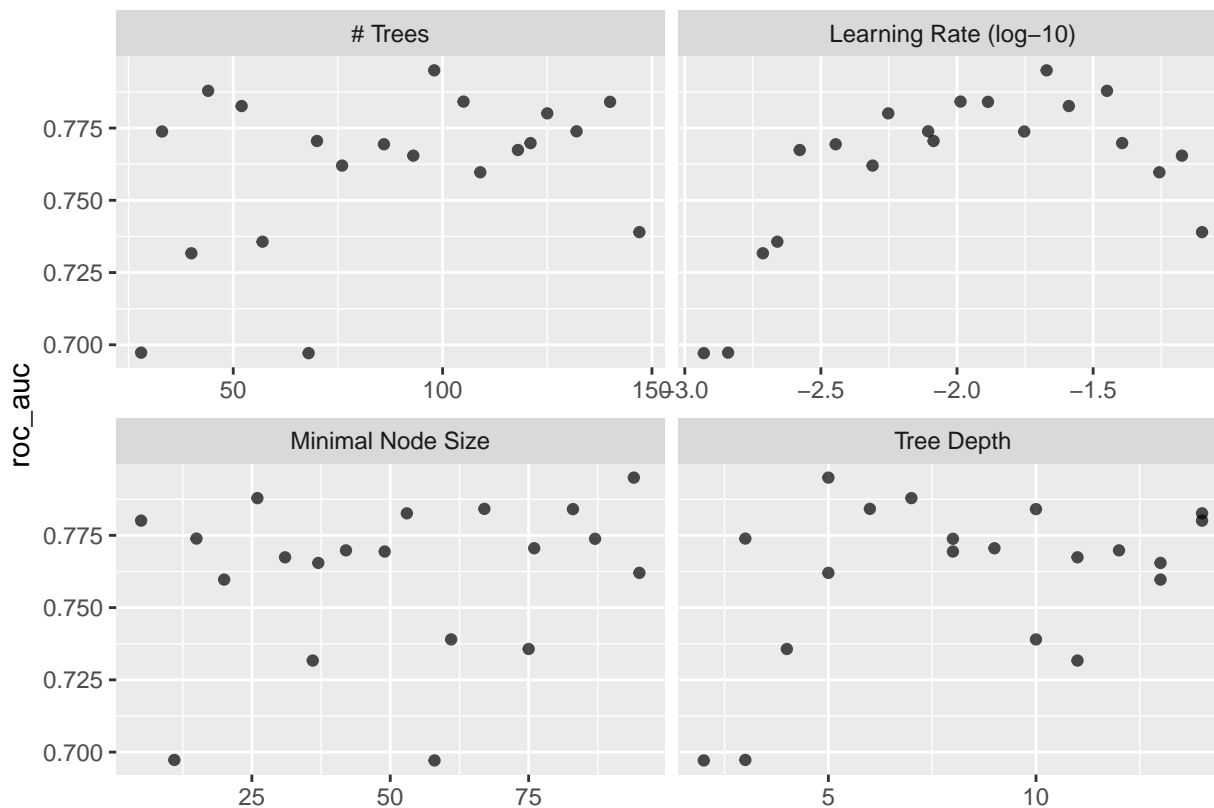


```
final_lightgbm_workflow <-
  lightgbm_workflow %>%
  finalize_workflow(best_lightgbm)

last_lightgbm_fit <-
```
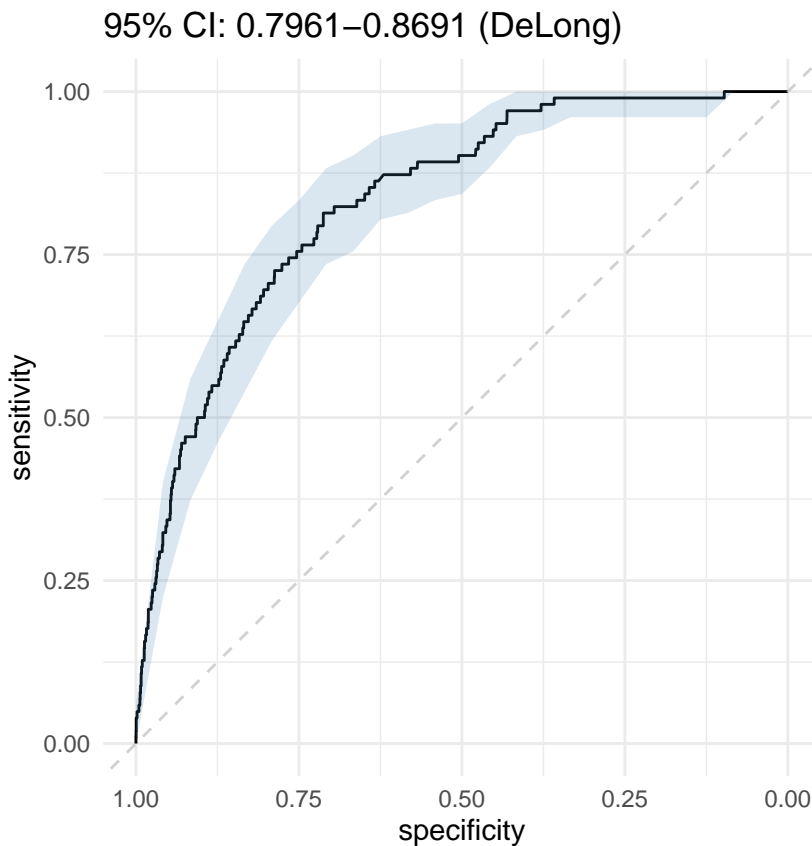
```
  final_lightgbm_workflow %>%
  last_fit(df_split)

final_lightgbm_fit <- extract_workflow(last_lightgbm_fit)

lightgbm_auc <- validation(final_lightgbm_fit, df_test)
```

95% CI: 0.7961–0.8691 (DeLong)


```
## [1] "Optimal Threshold: 0.02"
## Confusion Matrix and Statistics
##
##      reference
## data    0    1
##    0 3298   19
##    1 1330   83
##
##               Accuracy : 0.7148
##                 95% CI : (0.7017, 0.7276)
##    No Information Rate : 0.9784
##    P-Value [Acc > NIR] : 1
##
##                  Kappa : 0.0723
##
##  Mcnemar's Test P-Value : <2e-16
##
##            Sensitivity : 0.71262
##            Specificity : 0.81373
##         Pos Pred Value : 0.99427
##         Neg Pred Value : 0.05874
##             Prevalence : 0.97844
##         Detection Rate : 0.69725
##   Detection Prevalence : 0.70127
##      Balanced Accuracy : 0.76317
##
```

9

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

```r
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)
```
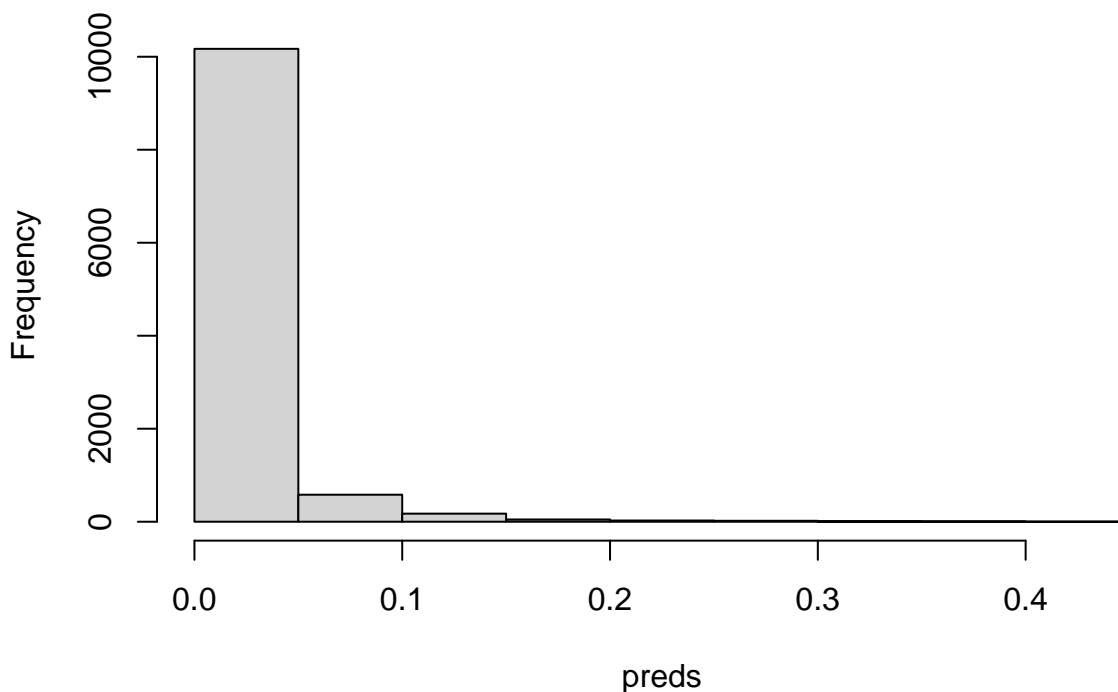
```
## lightgbm_parameters = list 4 (736 bytes)
## .  trees = integer 1= 98
## .  min_n = integer 1= 94
## .  tree_depth = integer 1= 5
## .  learn_rate = double 1= 0.021341
```

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

Minutes to run: 3.368

Histogram of preds

Minutes to run: 0.005

GLM

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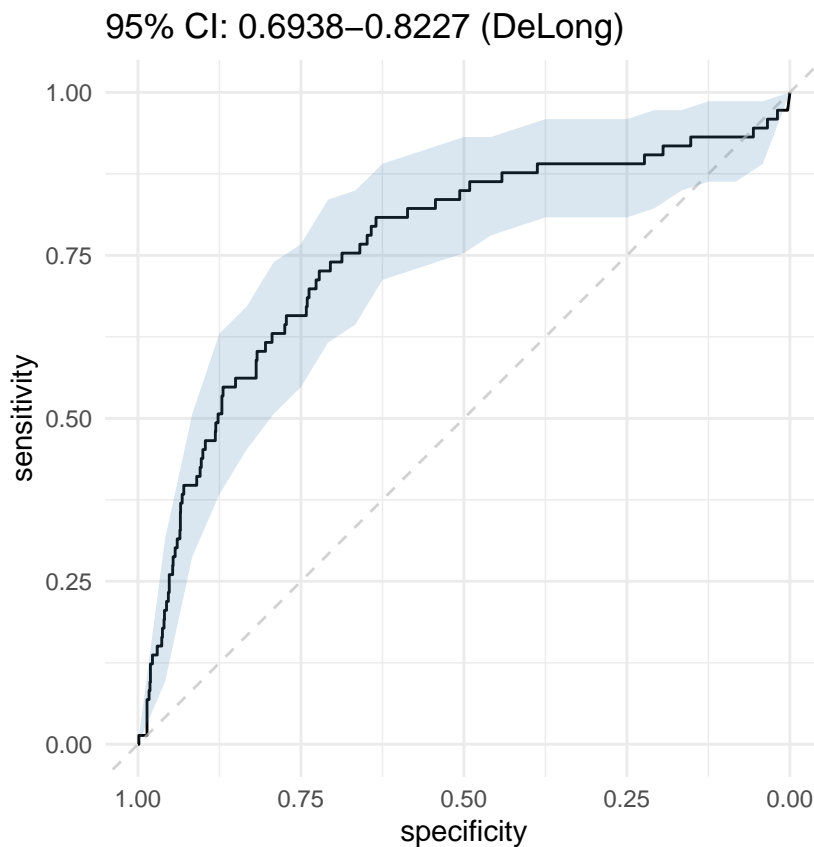
```
glmnet_spec <-
  logistic_reg(penalty = 0) %>%
  set_mode("classification") %>%
  set_engine("glm")

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)
```


95% CI: 0.6938–0.8227 (DeLong)

```
## [1] "Optimal Threshold: 0.02"
## Confusion Matrix and Statistics
##
##      reference
## data    0    1
##    0 2074   20
##    1  799   53
##
##                Accuracy : 0.722
##                  95% CI : (0.7054, 0.7381)
##     No Information Rate : 0.9752
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0722
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.72189
```

```
##             Specificity : 0.72603
##          Pos Pred Value : 0.99045
##          Neg Pred Value : 0.06221
##              Prevalence : 0.97522
##          Detection Rate : 0.70401
##    Detection Prevalence : 0.71079
##       Balanced Accuracy : 0.72396
##
##        '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')
```
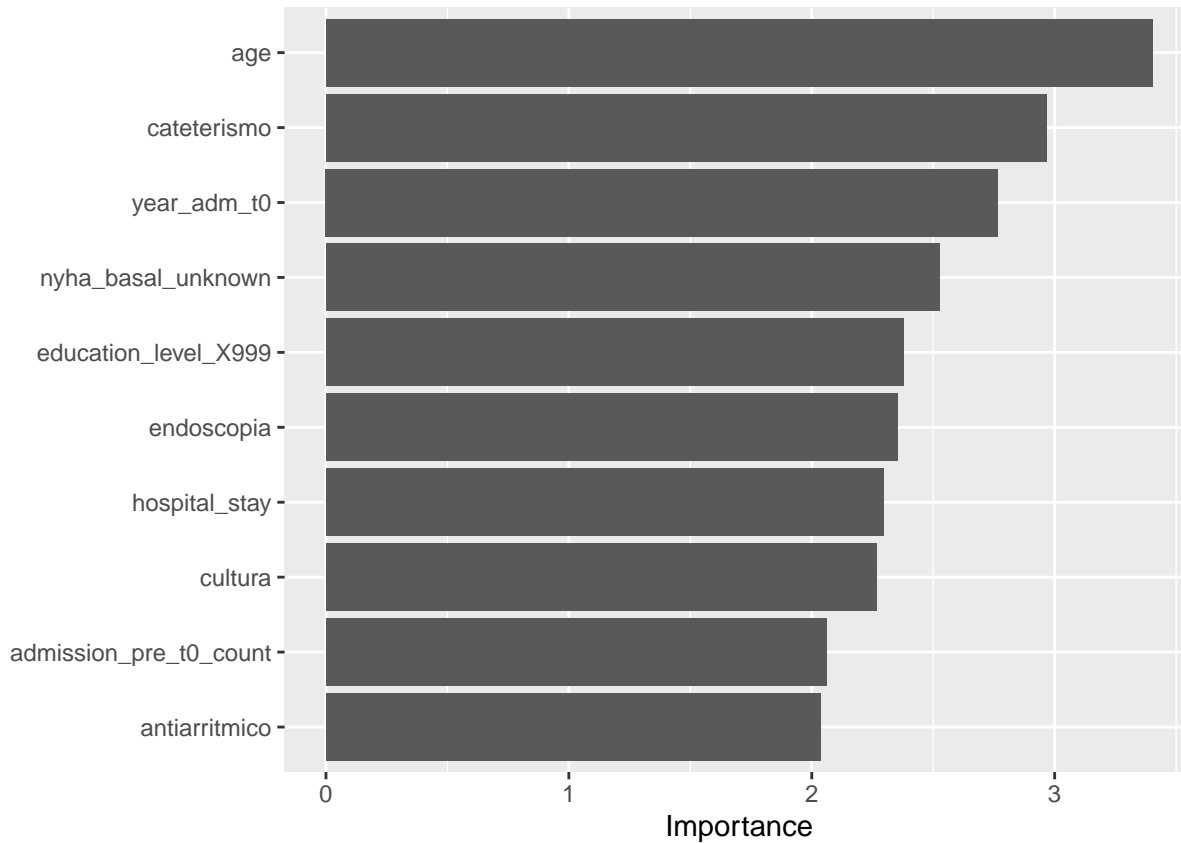


```
extract_vip(glm_fit, pred_wrapper = pfun_glmnet,
            reference_class = "1", method = 'permute')
```
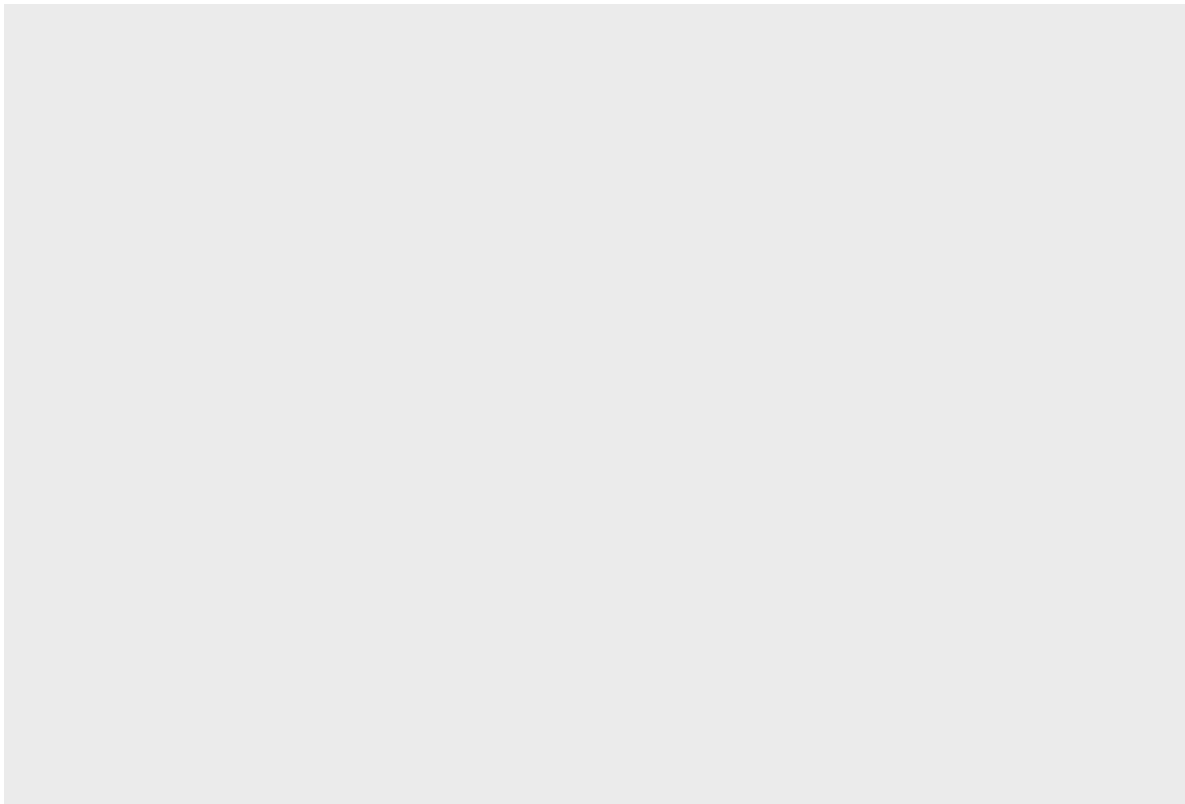
Importance

0.206

Minutes to run:

## Decision Tree

```r
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         1.26e- 7         13    37 accura~ binary     0.979    10 0.00130 Prepro~
##  2         1.26e- 7         13    37 roc_auc binary     0.536    10 0.0242  Prepro~
##  3         1.29e- 6         14    30 accura~ binary     0.979    10 0.00139 Prepro~
##  4         1.29e- 6         14    30 roc_auc binary     0.656    10 0.0154  Prepro~
##  5         1.57e- 5          6    11 accura~ binary     0.978    10 0.00144 Prepro~
##  6         1.57e- 5          6    11 roc_auc binary     0.695    10 0.0131  Prepro~
##  7         4.42e-10         14     9 accura~ binary     0.972    10 0.00128 Prepro~
##  8         4.42e-10         14     9 roc_auc binary     0.683    10 0.0217  Prepro~
##  9         2.11e- 2          4    18 accura~ binary     0.979    10 0.00124 Prepro~
## 10         2.11e- 2          4    18 roc_auc binary     0.5      10 0       Prepro~
## # i 30 more rows
```

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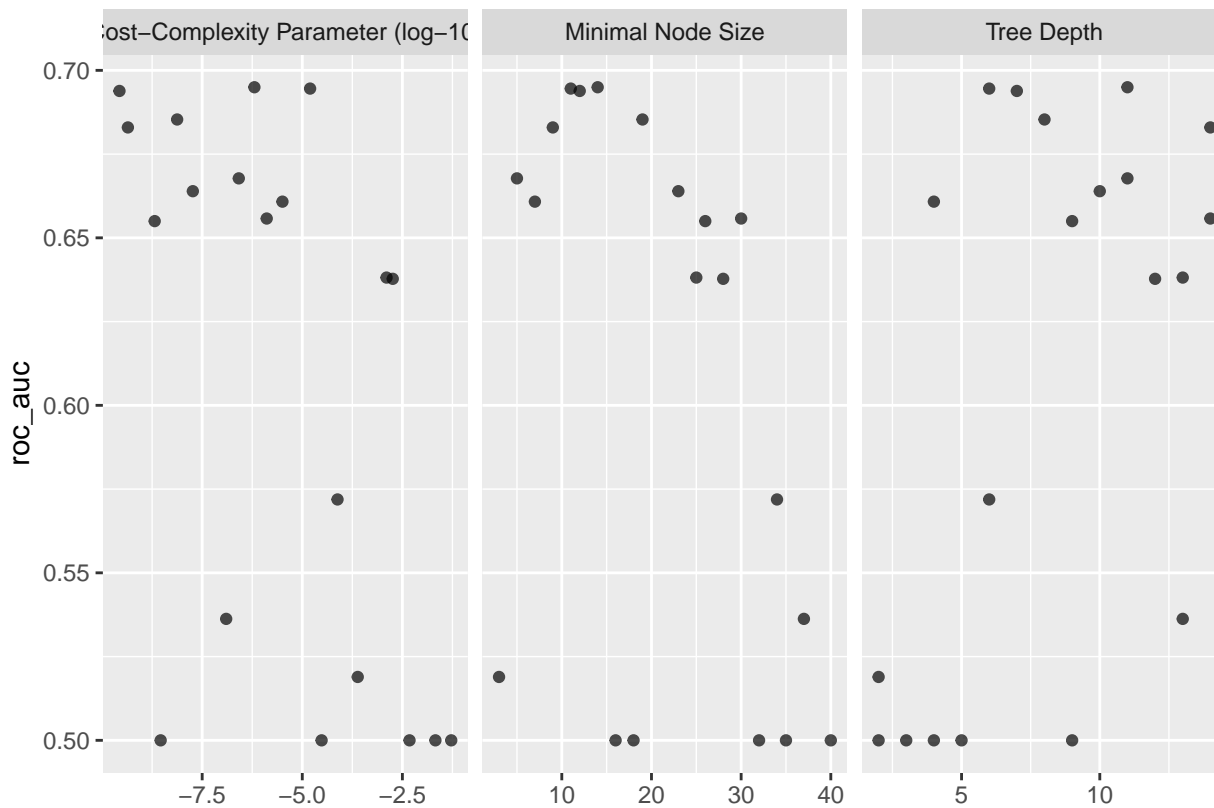


```
tree_tune %>%
  show_best("roc_auc")
```

```
## # 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        6.38e- 7         11    14 roc_auc binary     0.695    10  0.0163 Preproc~
## 2        1.57e- 5          6    11 roc_auc binary     0.695    10  0.0131 Preproc~
## 3        2.72e-10          7    12 roc_auc binary     0.694    10  0.0179 Preproc~
## 4        7.51e- 9          8    19 roc_auc binary     0.685    10  0.0120 Preproc~
## 5        4.42e-10         14     9 roc_auc binary     0.683    10  0.0217 Preproc~
```

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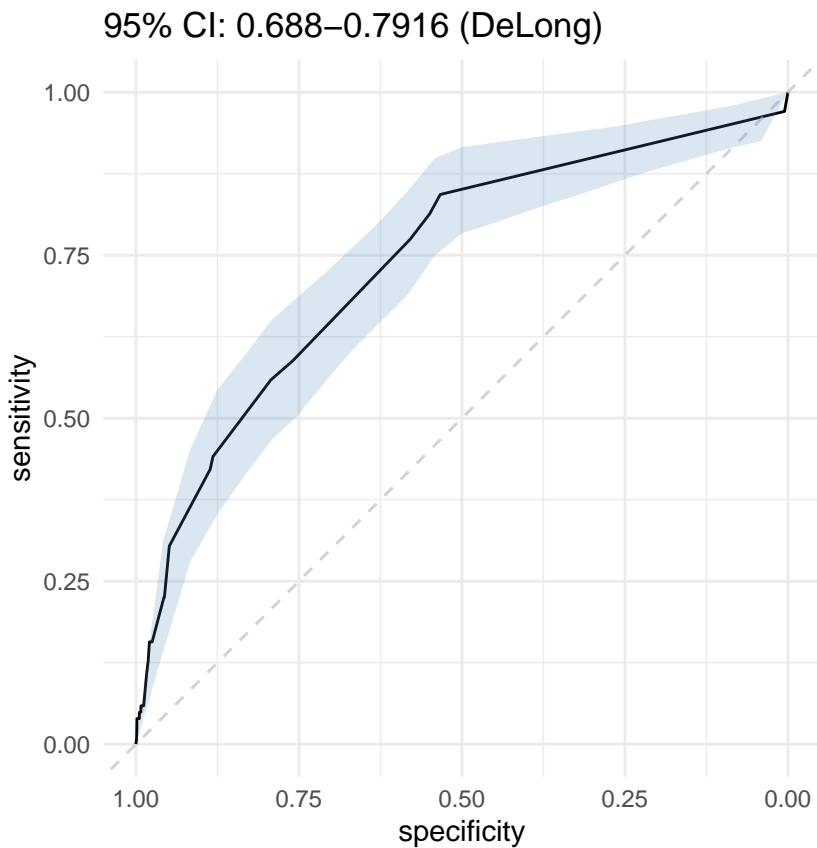
```
final_tree_workflow <-
  tree_workflow %>%
  finalize_workflow(best_tree)

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)
```

## 95% CI: 0.688–0.7916 (DeLong)


```
## [1] "Optimal Threshold: 0.01"
## Confusion Matrix and Statistics
##
##      reference
## data    0    1
##    0 2468   16
##    1 2160   86
##
##               Accuracy : 0.54
##                 95% CI : (0.5256, 0.5542)
##    No Information Rate : 0.9784
##    P-Value [Acc > NIR] : 1
##
##                  Kappa : 0.0334
##
##  Mcnemar's Test P-Value : <2e-16
##
##            Sensitivity : 0.53328
##            Specificity : 0.84314
##         Pos Pred Value : 0.99356
##         Neg Pred Value : 0.03829
```

```
##               Prevalence : 0.97844
##           Detection Rate : 0.52178
##     Detection Prevalence : 0.52516
##        Balanced Accuracy : 0.68821
##
##         'Positive' Class : 0
##
```

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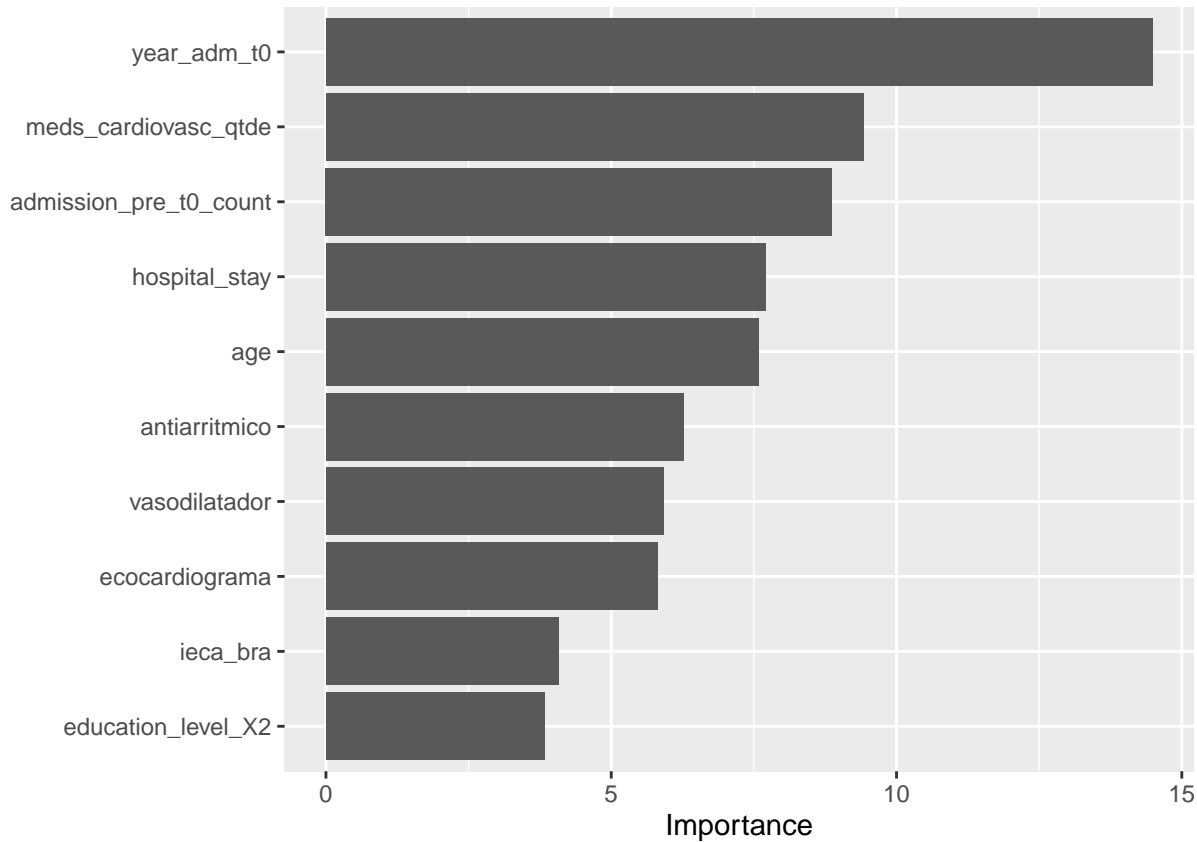


```
# extract_vip(final_tree_fit, pred_wrapper = predict,
#             reference_class = "1", use_matrix = FALSE,
#             method = 'permute')
```

Minutes to run: 3.591

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, 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()
```
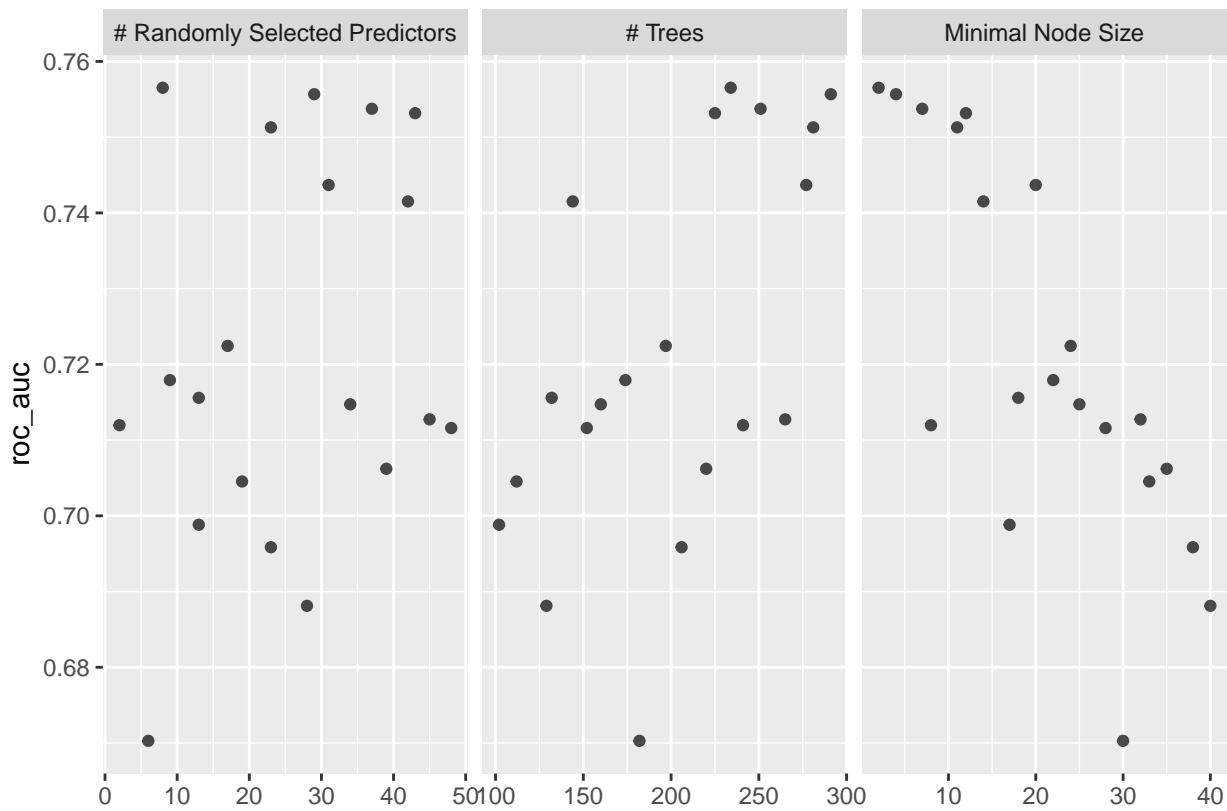
```
## # A tibble: 40 x 9
##      mtry trees min_n .metric  .estimator  mean       n std_err .config
##     <int> <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1       8   234     2 accuracy binary     0.979    10 0.00123 Preprocessor1_Model01
## 2       8   234     2 roc_auc  binary     0.757    10 0.0125  Preprocessor1_Model01
## 3      31   277    20 accuracy binary     0.979    10 0.00124 Preprocessor1_Model02
## 4      31   277    20 roc_auc  binary     0.744    10 0.0151  Preprocessor1_Model02
## 5      23   206    38 accuracy binary     0.979    10 0.00124 Preprocessor1_Model03
## 6      23   206    38 roc_auc  binary     0.696    10 0.0100  Preprocessor1_Model03
## 7       2   241     8 accuracy binary     0.979    10 0.00124 Preprocessor1_Model04
## 8       2   241     8 roc_auc  binary     0.712    10 0.0134  Preprocessor1_Model04
## 9      45   265    32 accuracy binary     0.979    10 0.00124 Preprocessor1_Model05
## 10     45   265    32 roc_auc  binary     0.713    10 0.0172  Preprocessor1_Model05
## # 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     8   234     2 roc_auc binary     0.757    10  0.0125 Preprocessor1_Model01
## 2    29   291     4 roc_auc binary     0.756    10  0.0155 Preprocessor1_Model06
## 3    37   251     7 roc_auc binary     0.754    10  0.0108 Preprocessor1_Model15
## 4    43   225    12 roc_auc binary     0.753    10  0.0130 Preprocessor1_Model20
## 5    23   281    11 roc_auc binary     0.751    10  0.0137 Preprocessor1_Model14
```
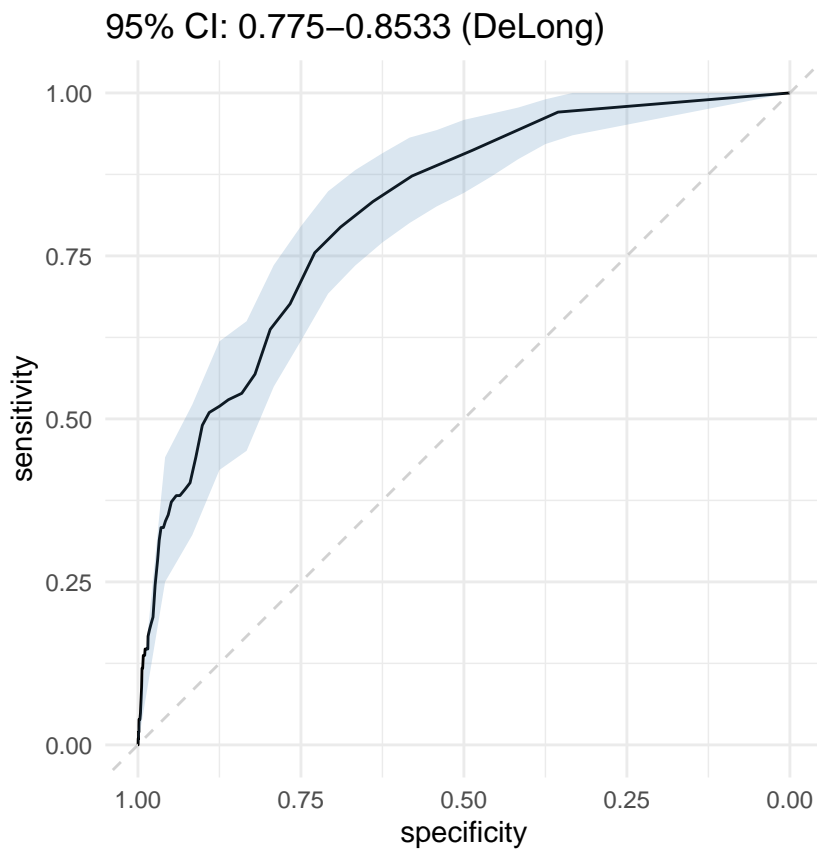
```
best_rf <- rf_tune %>%
  select_best("roc_auc")

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)
```

95% CI: 0.775–0.8533 (DeLong)

```
## [1] "Optimal Threshold: 0.02"
## Confusion Matrix and Statistics
##
##      reference
## data    0    1
##    0 3373   25
##    1 1255   77
##
##                 Accuracy : 0.7294
##                   95% CI : (0.7165, 0.742)
##      No Information Rate : 0.9784
##      P-Value [Acc > NIR] : 1
##
##                    Kappa : 0.0701
##
##   Mcnemar's Test P-Value : <2e-16
##
##              Sensitivity : 0.72882
##              Specificity : 0.75490
##           Pos Pred Value : 0.99264
##           Neg Pred Value : 0.05781
##               Prevalence : 0.97844
##           Detection Rate : 0.71311
##     Detection Prevalence : 0.71839
##        Balanced Accuracy : 0.74186
##
##         'Positive' Class : 0
##
```

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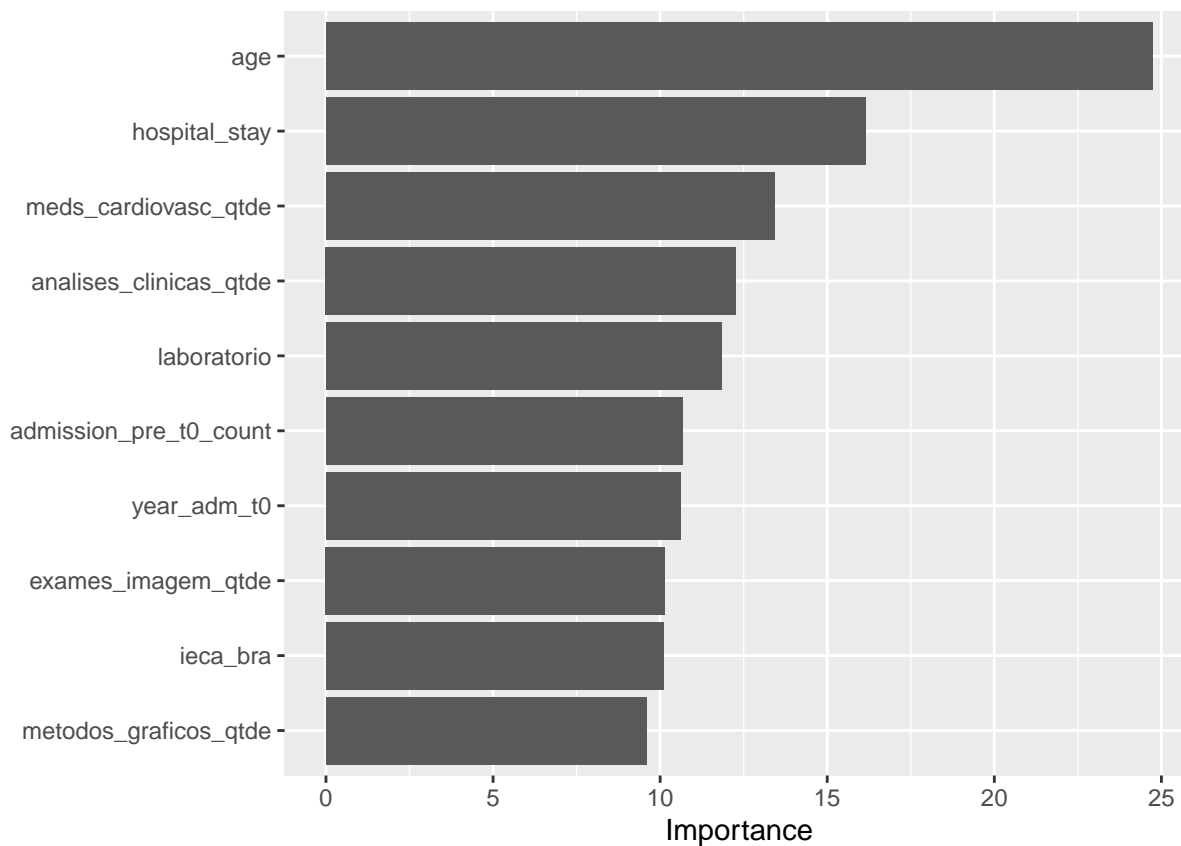
```
# extract_vip(final_rf_fit, pred_wrapper = predict,
#             reference_class = "1", use_matrix = FALSE,
#             method = 'permute')
```

Minutes to run: 56.848

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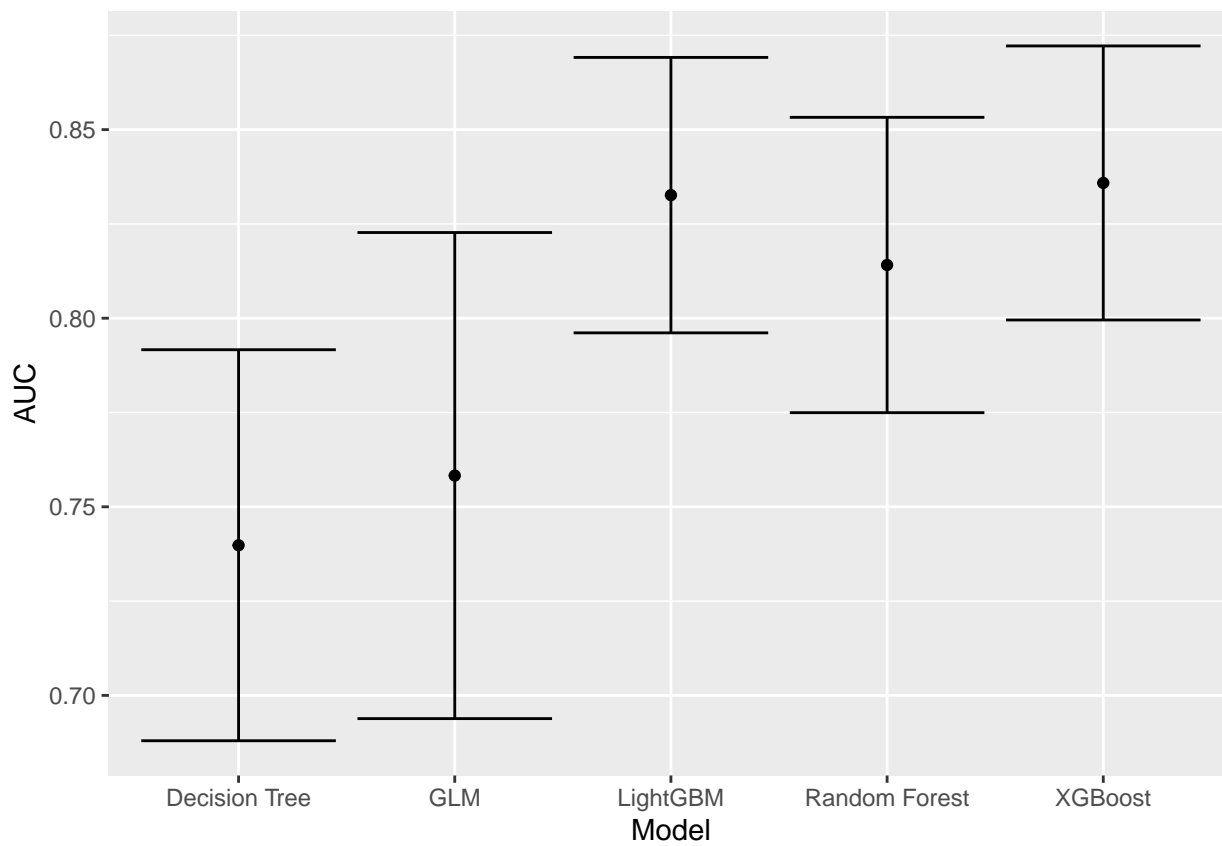
```
write_csv(df_auc, sprintf("./auxiliar/model_selection/performance/%s.csv", outcome_column))
```

Minutes to run: 0.009