Model Selection - death_1year

Eduardo Yuki Yada

Global parameters

```r
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_1year
## .  k = double 1= 10
## .  grid_size = double 1= 20
## .  repeats = double 1= 2
## .  RUN_ALL_MODELS = logical 1= TRUE
```

Minutes to run: 0

Imports

```r
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

```r
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: 0.007

```r
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. antiviral
## 46. classe_meds_qtde
## 47. meds_cardiovasc_qtde
## 48. meds_antimicrobianos
## 49. vni
## 50. ventilacao_mecanica
## 51. transplante_cardiaco
## 52. cir_toracica
## 53. outros_proced_cirurgicos
## 54. icp
## 55. cateterismo
## 56. cateter_venoso_central
## 57. proced_invasivos_qtde
## 58. transfusao
## 59. interconsulta
## 60. equipe_multiprof
## 61. holter
## 62. teste_esforco
## 63. tilt_teste
## 64. metodos_graficos_qtde
## 65. laboratorio
## 66. cultura
## 67. analises_clinicas_qtde
## 68. citologia
## 69. histopatologia_qtde
## 70. angio_tc
## 71. angiografia
## 72. aortografia
## 73. cintilografia
## 74. ecocardiograma
## 75. endoscopia
## 76. flebografia
## 77. pet_ct
## 78. ultrassom
## 79. tomografia
## 80. ressonancia
## 81. exames_imagem_qtde
## 82. bic
## 83. 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))
```

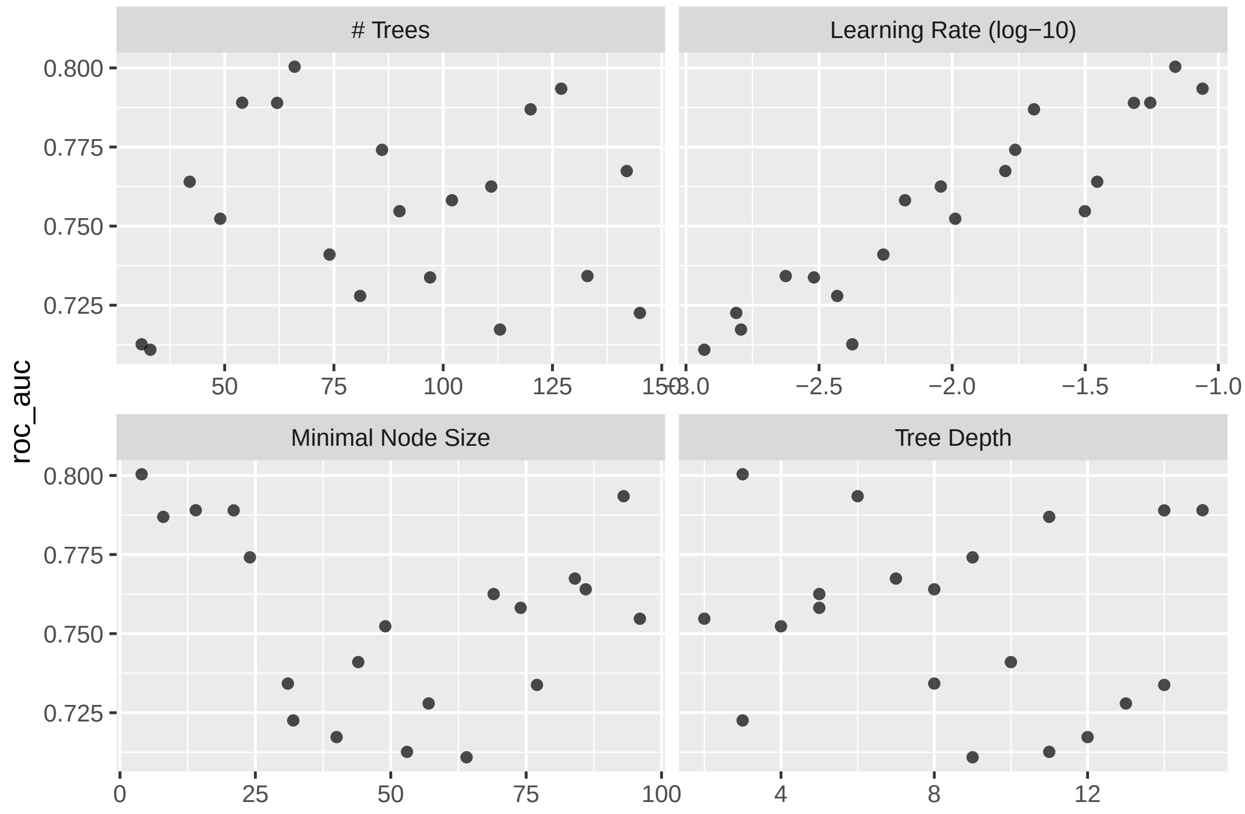
Minutes to run: 0.002

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() %>%
  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")
```

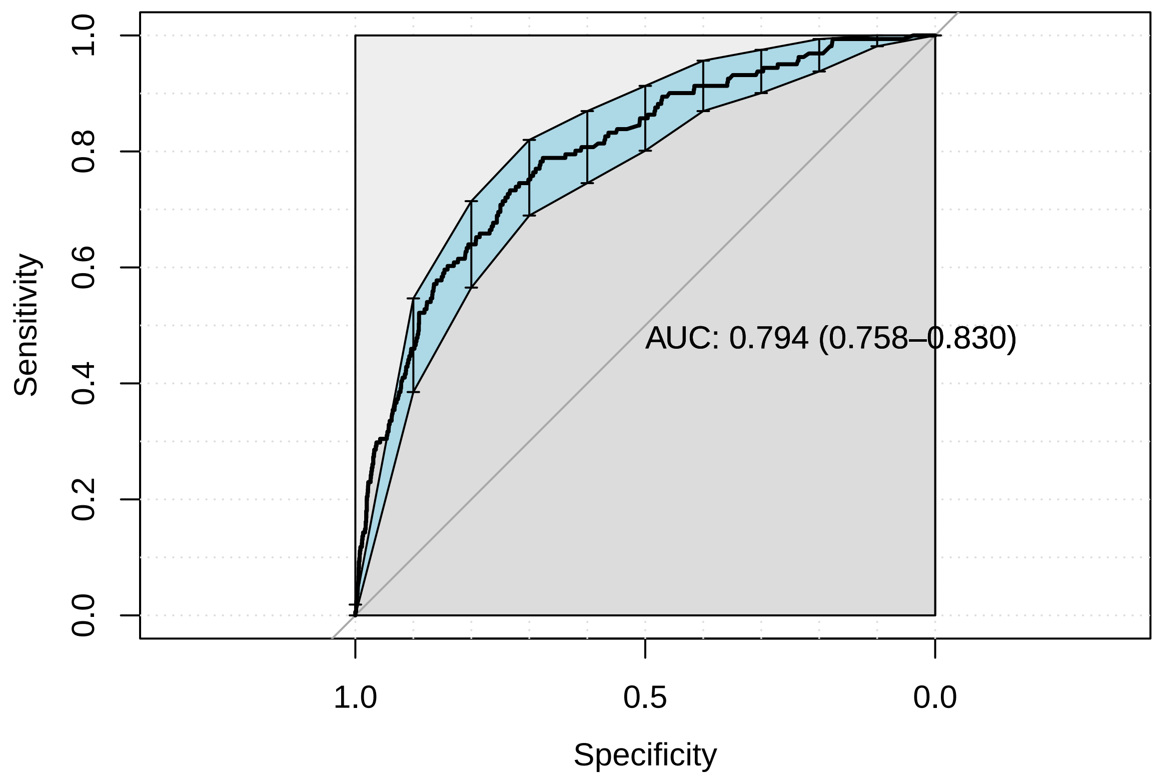
```
## # A tibble: 5 x 10
##   trees min_n tree_depth learn_rate .metric .estimator  mean     n std_err .config
##   <int> <int>      <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    66     4          3     0.0689 roc_auc binary     0.800    10  0.0174 Preprocessor1_Model01
## 2   127    93          6     0.0872 roc_auc binary     0.793    10  0.0155 Preprocessor1_Model19
## 3    54    14         15     0.0555 roc_auc binary     0.789    10  0.0148 Preprocessor1_Model03
## 4    62    21         14     0.0482 roc_auc binary     0.789    10  0.0141 Preprocessor1_Model04
## 5   120     8         11     0.0203 roc_auc binary     0.787    10  0.0154 Preprocessor1_Model02
```

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

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



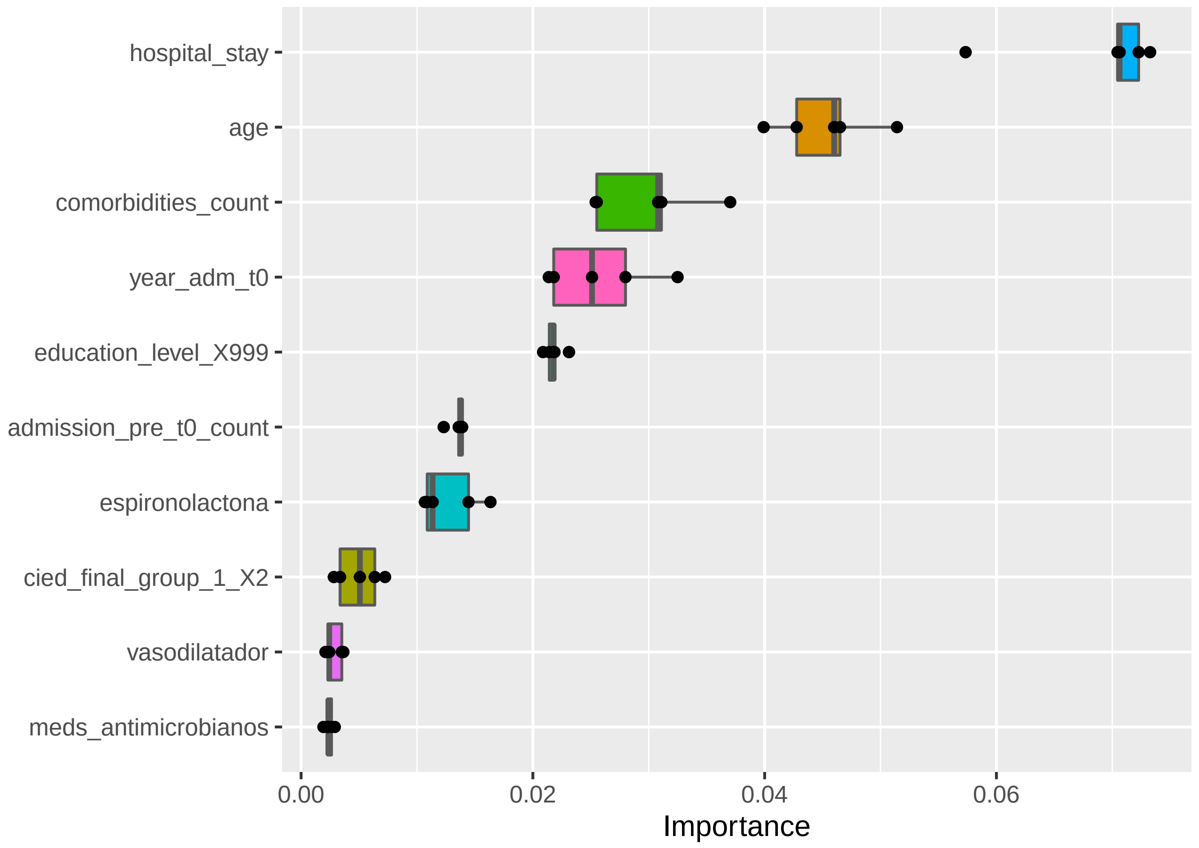
```
final_xgboost_workflow <-
  xgboost_workflow %>%
  finalize_workflow(best_xgboost)

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: 0.04"
## Confusion Matrix and Statistics
##
##      reference
## data    0    1
##    0 3350   43
##    1 1219  118
##
##                Accuracy : 0.7332
##                  95% CI : (0.7203, 0.7458)
##     No Information Rate : 0.966
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.103
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.73320
##             Specificity : 0.73292
##          Pos Pred Value : 0.98733
##          Neg Pred Value : 0.08826
##              Prevalence : 0.96596
##          Detection Rate : 0.70825
##    Detection Prevalence : 0.71734
##       Balanced Accuracy : 0.73306
##
##        'Positive' Class : 0
##
```

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



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

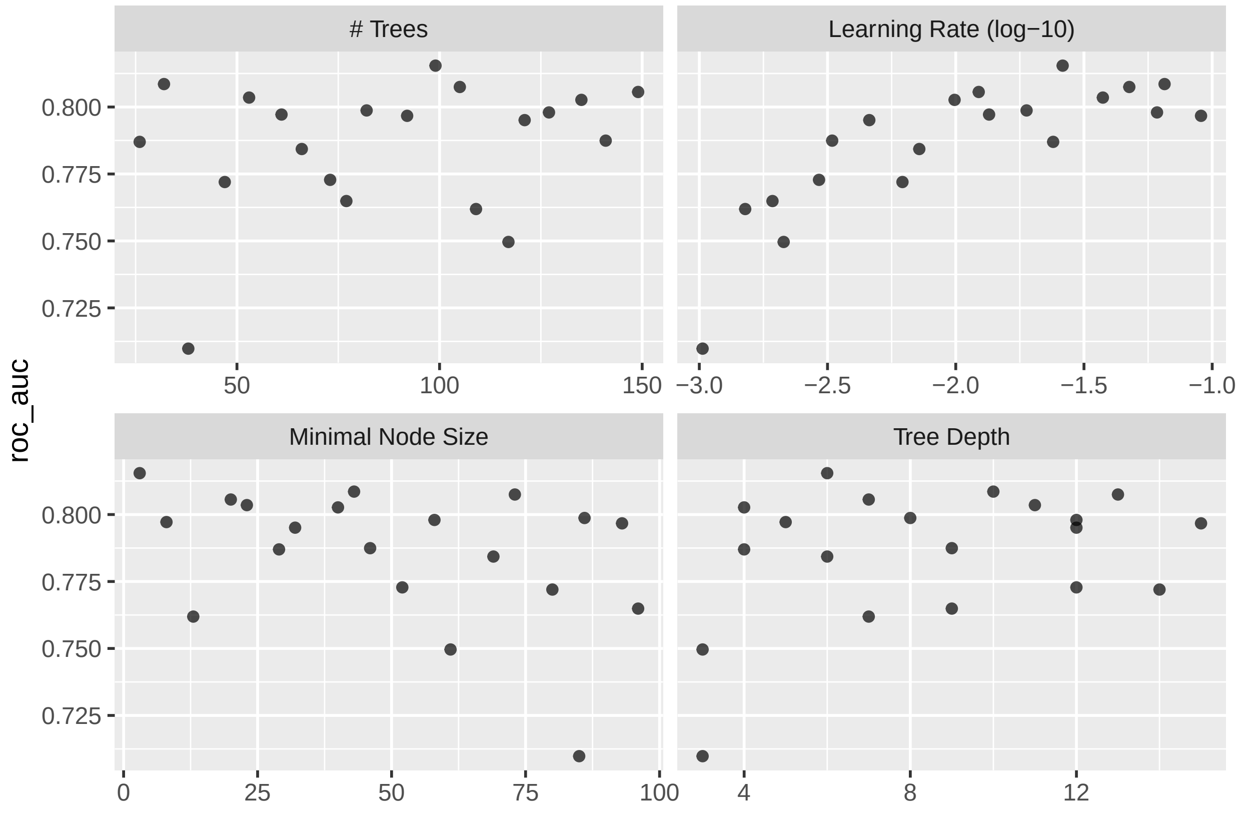
Minutes to run: 4.776

Boosted Tree (LightGBM)

```r
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")
```

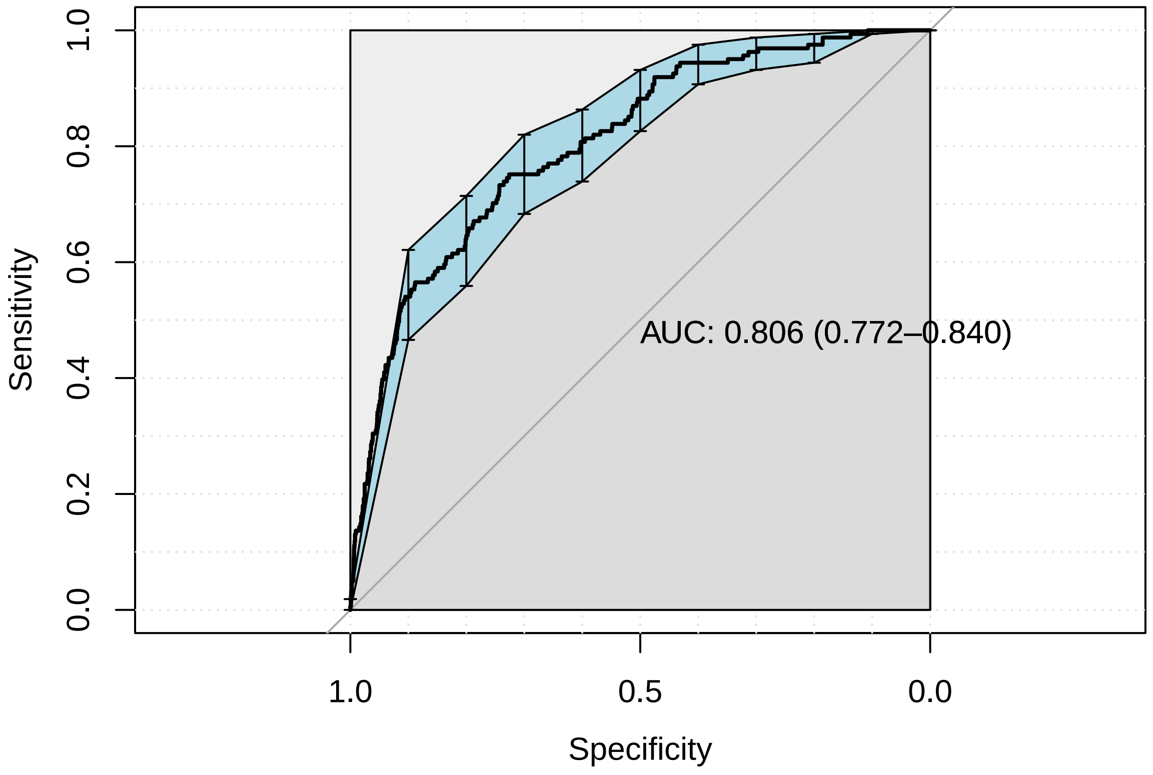
```
## # A tibble: 5 x 10
##   trees min_n tree_depth learn_rate .metric .estimator  mean     n std_err .config
##   <int> <int>      <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    99     3          6     0.0261 roc_auc binary     0.815    10 0.0122  Preprocessor1_Model01
## 2    32    43         10     0.0652 roc_auc binary     0.809    10 0.0131  Preprocessor1_Model09
## 3   105    73         13     0.0475 roc_auc binary     0.807    10 0.00846 Preprocessor1_Model15
## 4   149    20          7     0.0123 roc_auc binary     0.806    10 0.0136  Preprocessor1_Model04
## 5    53    23         11     0.0375 roc_auc binary     0.804    10 0.0136  Preprocessor1_Model05
```

```
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: 0.02"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 3318   40
##    1 1251  121
##
##                Accuracy : 0.7271
##                  95% CI : (0.7141, 0.7397)
##     No Information Rate : 0.966
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1032
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.72620
##             Specificity : 0.75155
##          Pos Pred Value : 0.98809
##          Neg Pred Value : 0.08819
##              Prevalence : 0.96596
##          Detection Rate : 0.70148
##    Detection Prevalence : 0.70994
##       Balanced Accuracy : 0.73888
##
```

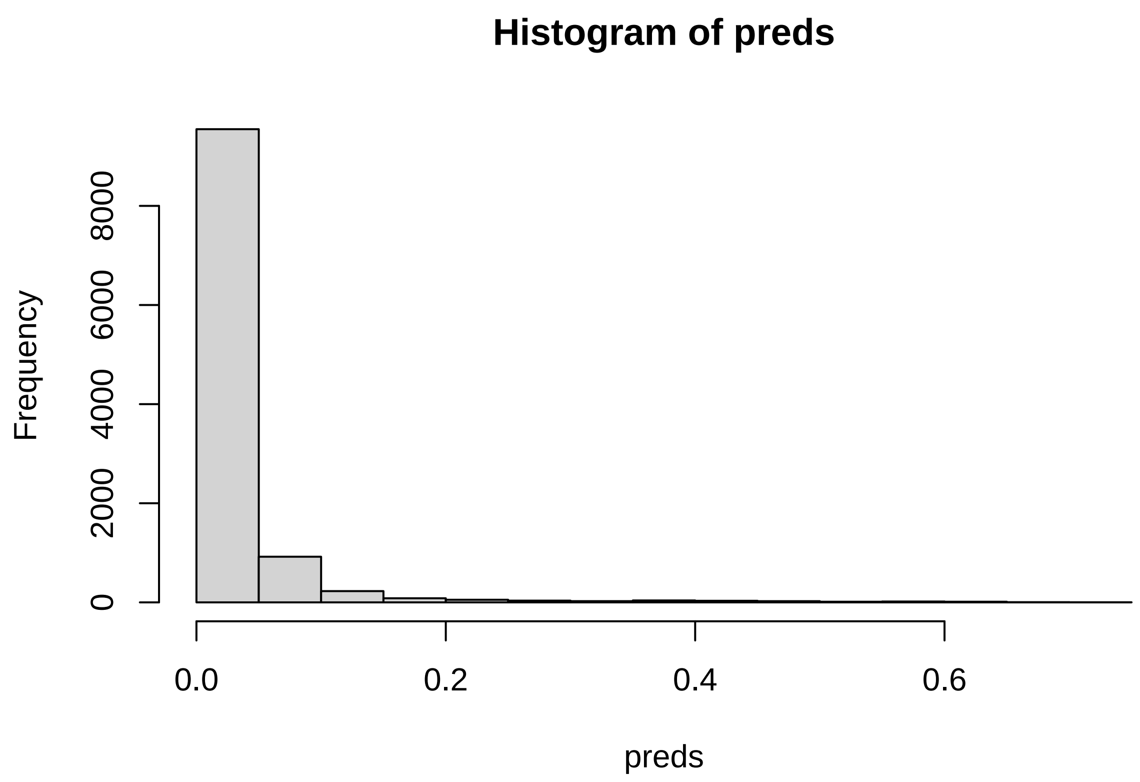
```
##        'Positive' Class : 0
##
```

```
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= 99
## .  min_n = integer 1= 3
## .  tree_depth = integer 1= 6
## .  learn_rate = double 1= 0.026118
```

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

Minutes to run: 4.334

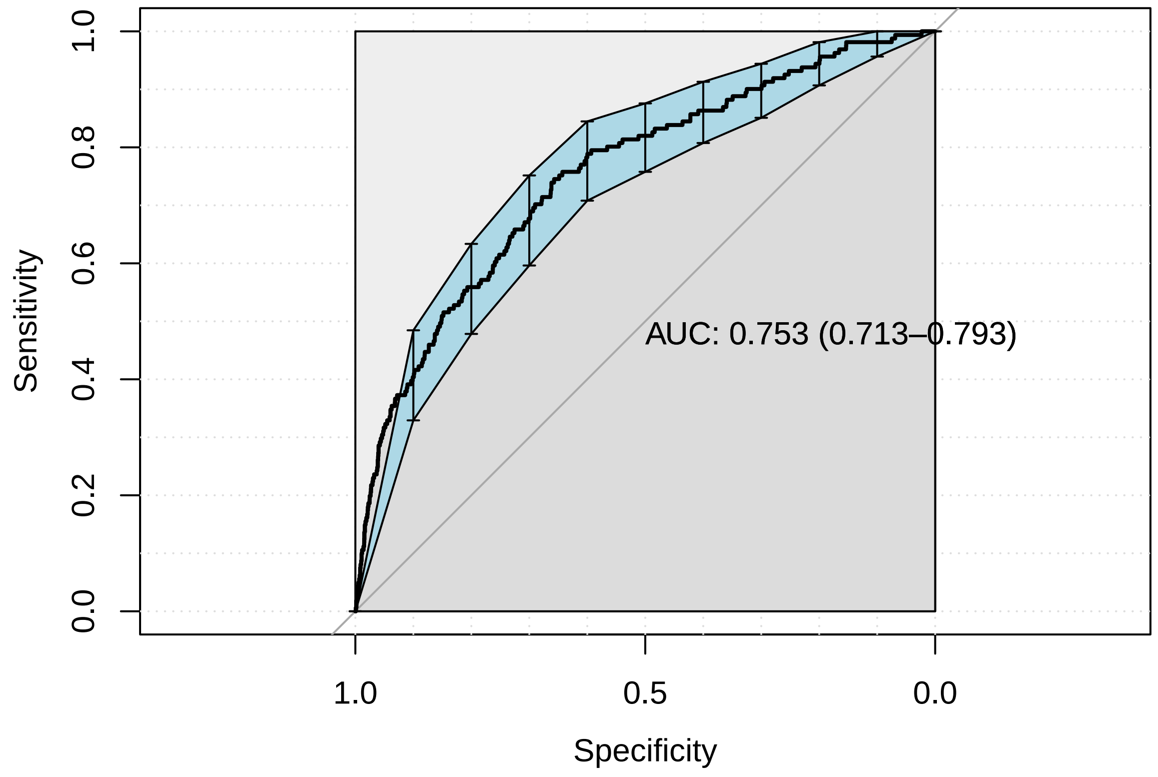


Minutes to run: 0.013

GLM

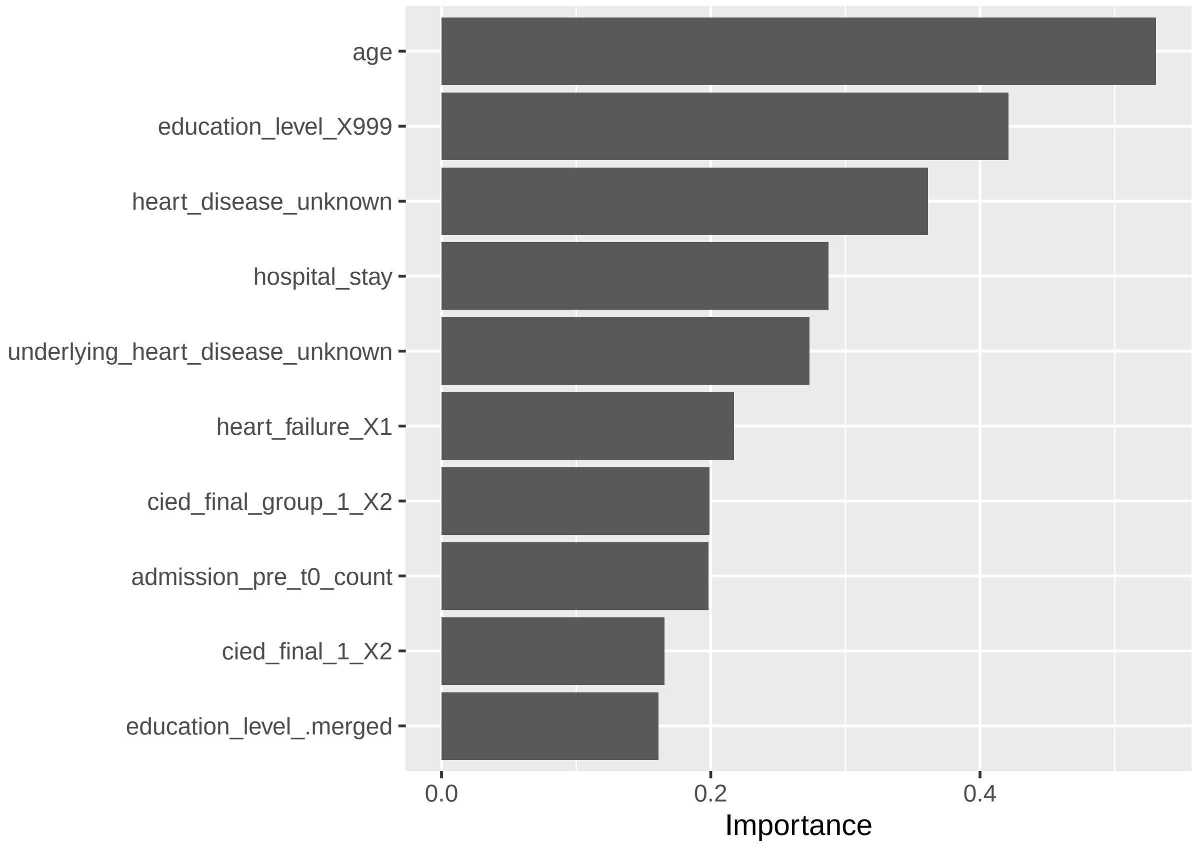
```
glmnet_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_other(all_nominal_predictors(), threshold = 0.05, other = ".merged") %>%
  step_dummy(all_nominal_predictors())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

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

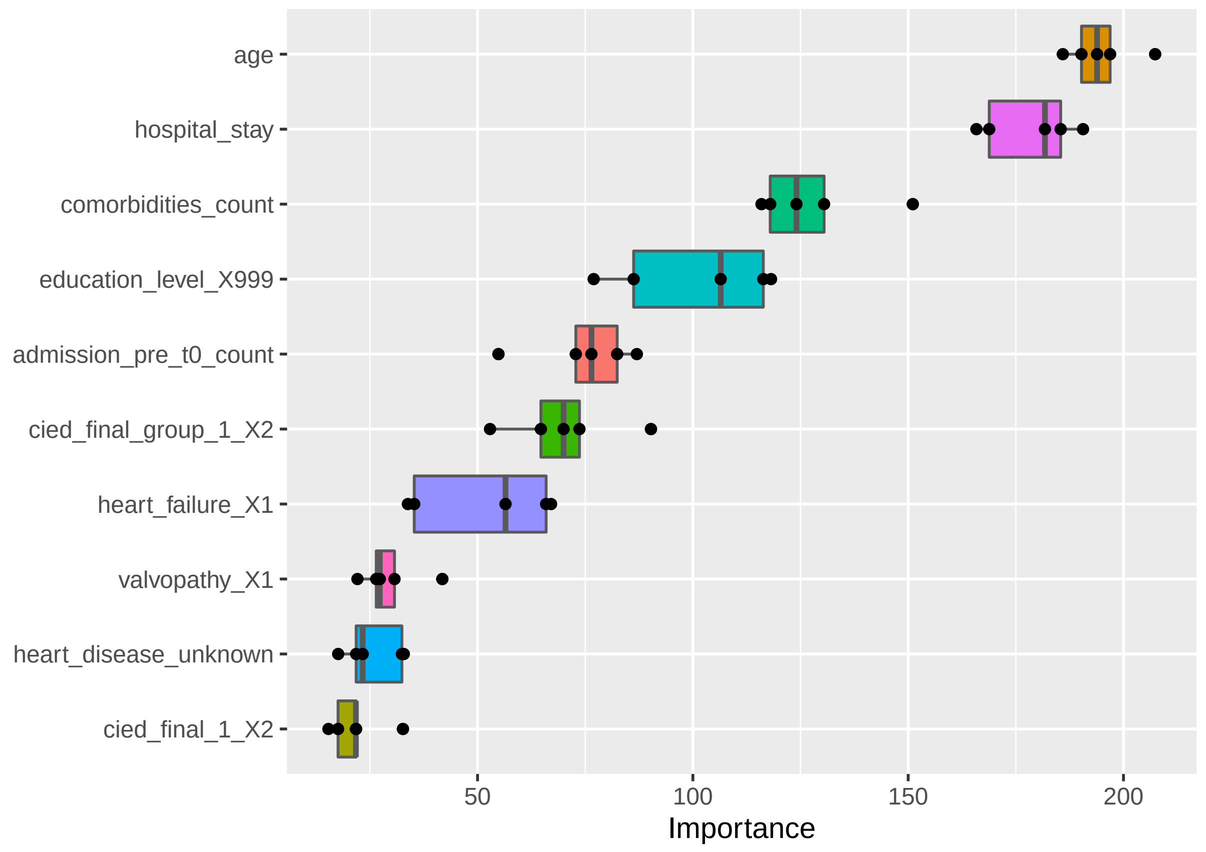


```
## [1] "Optimal Threshold: 0.03"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 3003   41
##    1 1566  120
##
##                Accuracy : 0.6603
##                  95% CI : (0.6466, 0.6738)
##     No Information Rate : 0.966
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0723
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.65726
##              Specificity : 0.74534
##           Pos Pred Value : 0.98653
##           Neg Pred Value : 0.07117
##               Prevalence : 0.96596
##           Detection Rate : 0.63488
##     Detection Prevalence : 0.64355
##        Balanced Accuracy : 0.70130
##
##         '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: 1.923

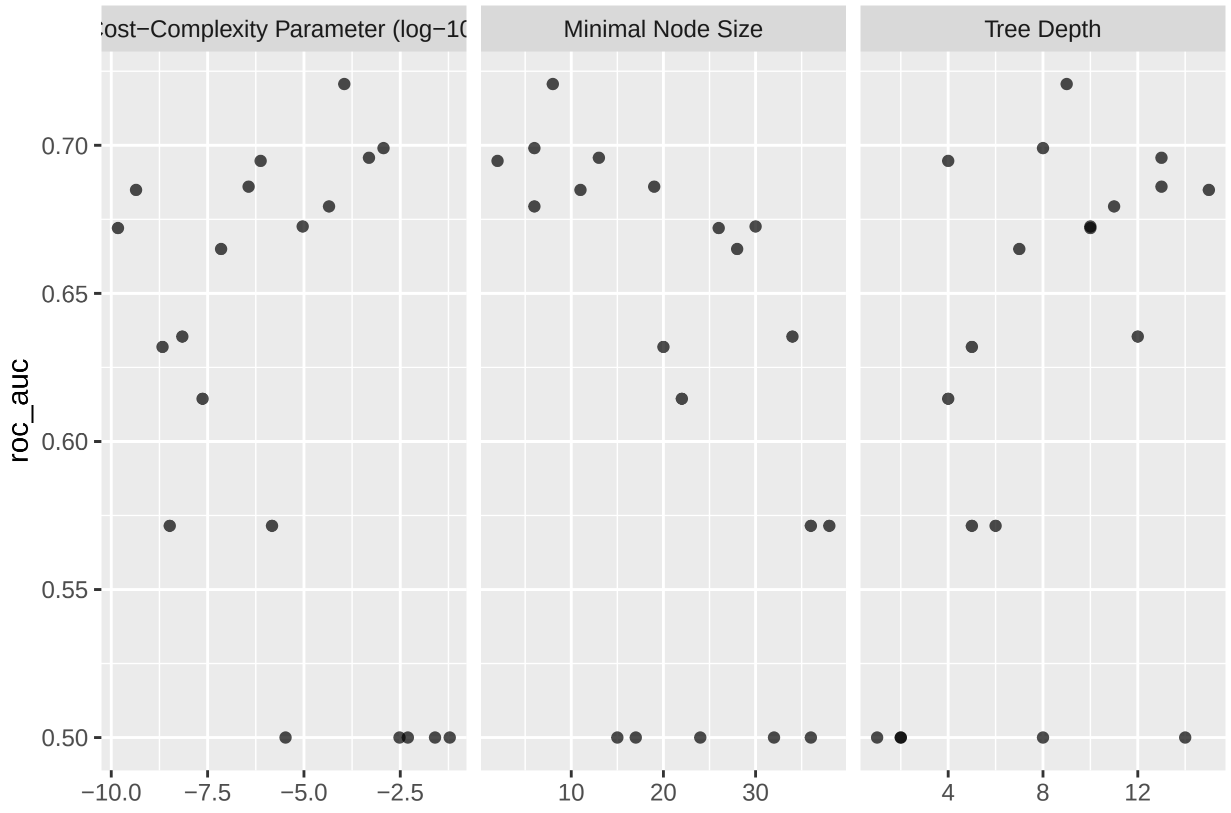
Decision Tree

```
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        1.49e-10         10    26 accuracy binary     0.964    10 0.00166 Preprocessor1_Model01
##  2        1.49e-10         10    26 roc_auc  binary     0.672    10 0.0147  Preprocessor1_Model01
##  3        4.39e-10         15    11 accuracy binary     0.956    10 0.00145 Preprocessor1_Model02
##  4        4.39e-10         15    11 roc_auc  binary     0.685    10 0.0169  Preprocessor1_Model02
##  5        4.48e- 5         11     6 accuracy binary     0.956    10 0.00145 Preprocessor1_Model03
##  6        4.48e- 5         11     6 roc_auc  binary     0.679    10 0.0250  Preprocessor1_Model03
##  7        3.67e- 7         13    19 accuracy binary     0.960    10 0.00162 Preprocessor1_Model04
##  8        3.67e- 7         13    19 roc_auc  binary     0.686    10 0.0168  Preprocessor1_Model04
##  9        9.28e- 6         10    30 accuracy binary     0.966    10 0.00147 Preprocessor1_Model05
## 10        9.28e- 6         10    30 roc_auc  binary     0.673    10 0.0145  Preprocessor1_Model05
## # ... with 30 more rows
## # i Use `print(n = ...)` to see more rows
```

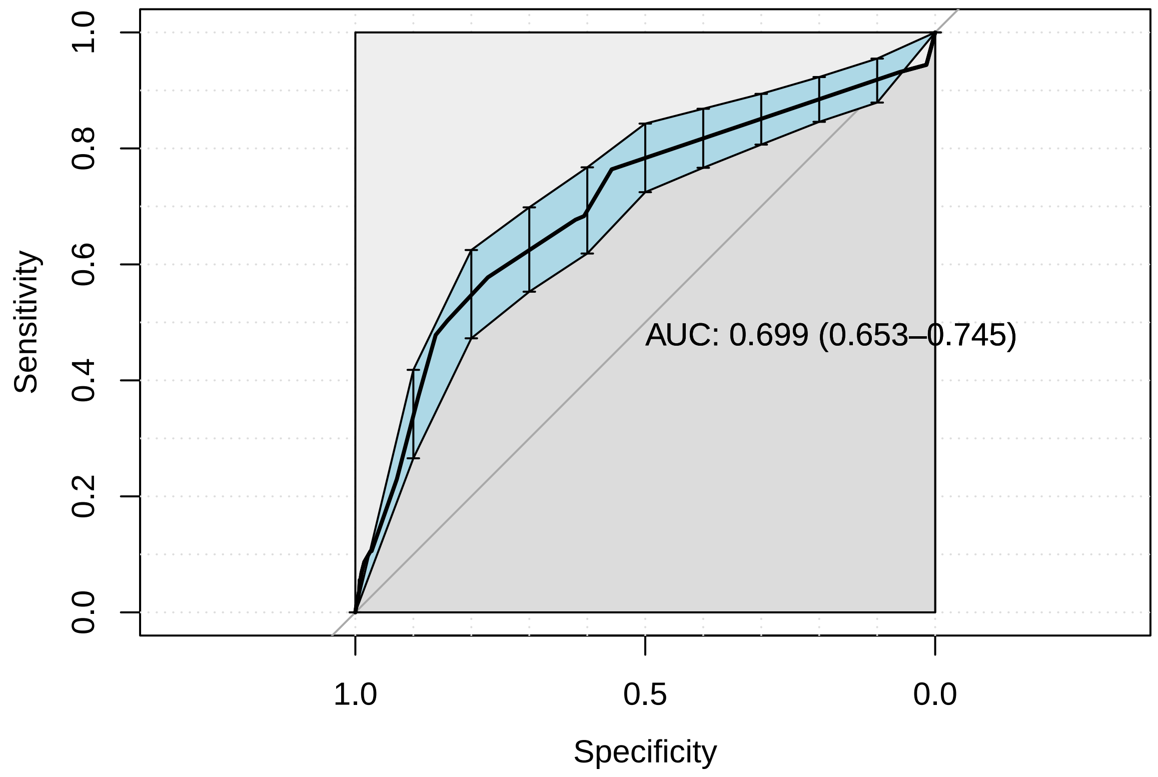
```
autoplot(tree_tune, metric = "roc_auc")
```



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

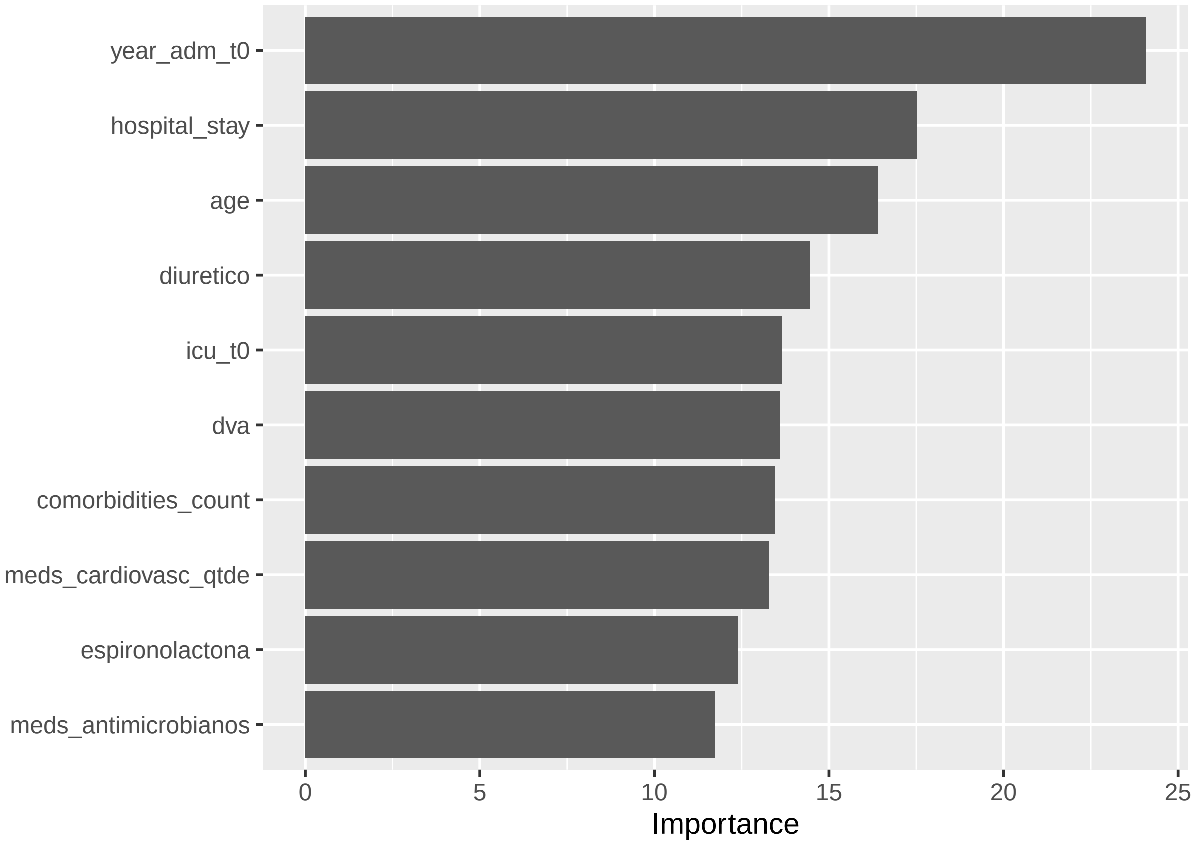
```
## # A tibble: 5 x 9
##   cost_complexity tree_depth min_n .metric .estimator  mean     n std_err .config
##             <dbl>      <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     0.000111             9     8 roc_auc binary     0.721    10  0.0166 Preprocessor1_Model13
## 2     0.00116              8     6 roc_auc binary     0.699    10  0.0167 Preprocessor1_Model10
## 3     0.000488            13    13 roc_auc binary     0.696    10  0.0196 Preprocessor1_Model06
## 4     0.000000752          4     2 roc_auc binary     0.695    10  0.0140 Preprocessor1_Model15
## 5     0.000000367         13    19 roc_auc binary     0.686    10  0.0168 Preprocessor1_Model04
```

```
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)
```



```
## [1] "Optimal Threshold: 0.03"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 3525   68
##    1 1044   93
##
##                Accuracy : 0.7649
##                  95% CI : (0.7526, 0.7769)
##     No Information Rate : 0.966
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.089
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.77150
##               Specificity : 0.57764
##            Pos Pred Value : 0.98107
##            Neg Pred Value : 0.08179
##                Prevalence : 0.96596
##            Detection Rate : 0.74524
##      Detection Prevalence : 0.75962
##         Balanced Accuracy : 0.67457
##
##          '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: 4.538

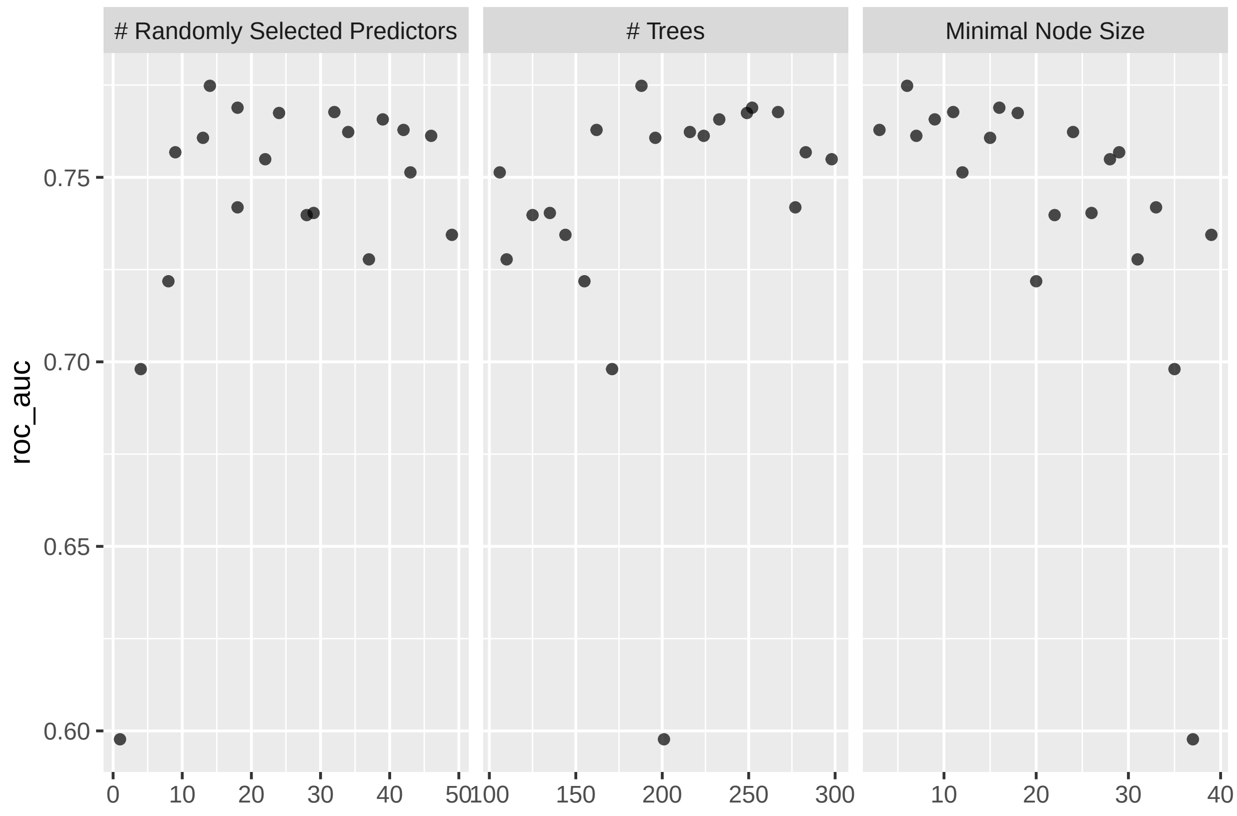
Random Forest

```
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 <-
```

```r
  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    49   144    39 accuracy binary     0.968    10 0.00141 Preprocessor1_Model01
##  2    49   144    39 roc_auc  binary     0.734    10 0.0136  Preprocessor1_Model01
##  3    18   277    33 accuracy binary     0.968    10 0.00143 Preprocessor1_Model02
##  4    18   277    33 roc_auc  binary     0.742    10 0.0181  Preprocessor1_Model02
##  5    46   224     7 accuracy binary     0.967    10 0.00146 Preprocessor1_Model03
##  6    46   224     7 roc_auc  binary     0.761    10 0.0134  Preprocessor1_Model03
##  7    34   216    24 accuracy binary     0.968    10 0.00143 Preprocessor1_Model04
##  8    34   216    24 roc_auc  binary     0.762    10 0.0148  Preprocessor1_Model04
##  9    43   106    12 accuracy binary     0.967    10 0.00139 Preprocessor1_Model05
## 10    43   106    12 roc_auc  binary     0.751    10 0.0131  Preprocessor1_Model05
## # ... with 30 more rows
## # i Use `print(n = ...)` to see more rows
```

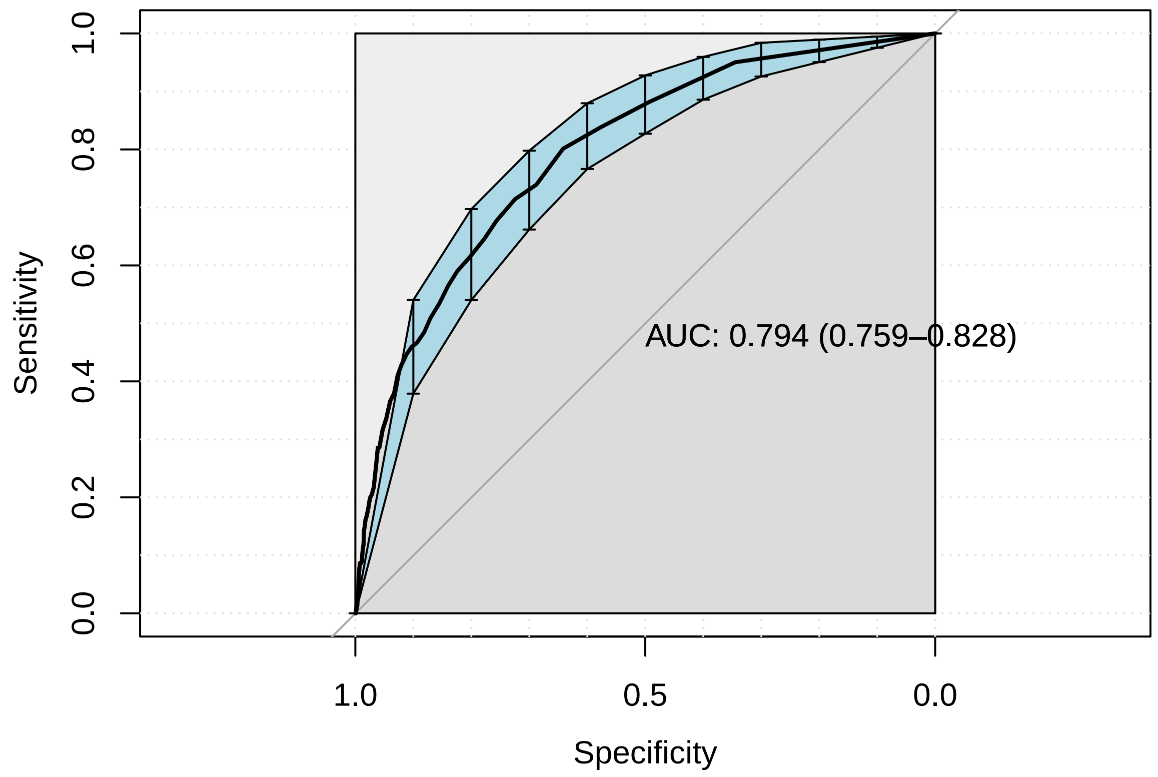
```r
autoplot(rf_tune, metric = "roc_auc")
```



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

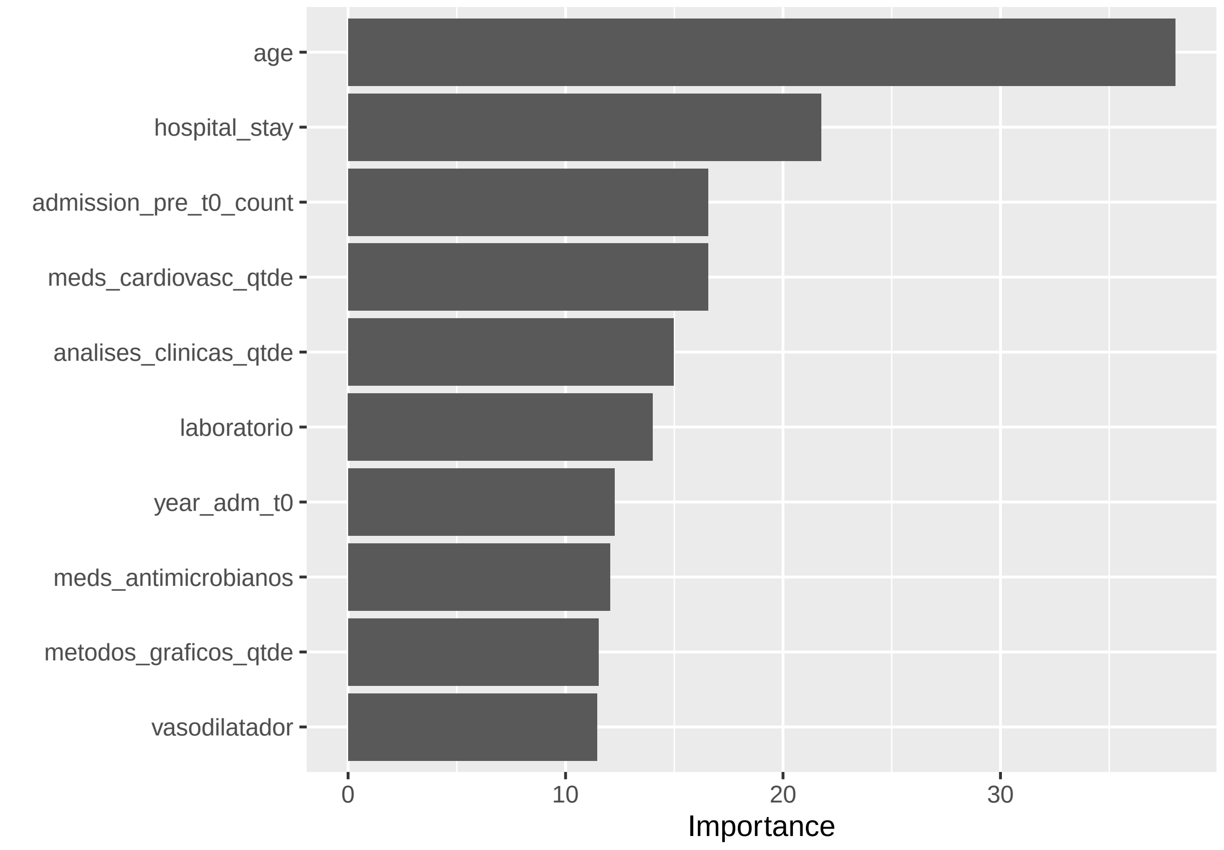
```
## # A tibble: 5 x 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    14   188     6 roc_auc binary     0.775    10  0.0123 Preprocessor1_Model15
## 2    18   252    16 roc_auc binary     0.769    10  0.0151 Preprocessor1_Model08
## 3    32   267    11 roc_auc binary     0.768    10  0.0147 Preprocessor1_Model13
## 4    24   249    18 roc_auc binary     0.767    10  0.0128 Preprocessor1_Model18
## 5    39   233     9 roc_auc binary     0.766    10  0.0141 Preprocessor1_Model19
```

```
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: 0.02"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 2932   32
##    1 1637  129
##
##                Accuracy : 0.6471
##                  95% CI : (0.6333, 0.6608)
##     No Information Rate : 0.966
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0763
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.64172
##             Specificity : 0.80124
##          Pos Pred Value : 0.98920
##          Neg Pred Value : 0.07305
##              Prevalence : 0.96596
##          Detection Rate : 0.61987
##    Detection Prevalence : 0.62664
##       Balanced Accuracy : 0.72148
##
##        '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')
```

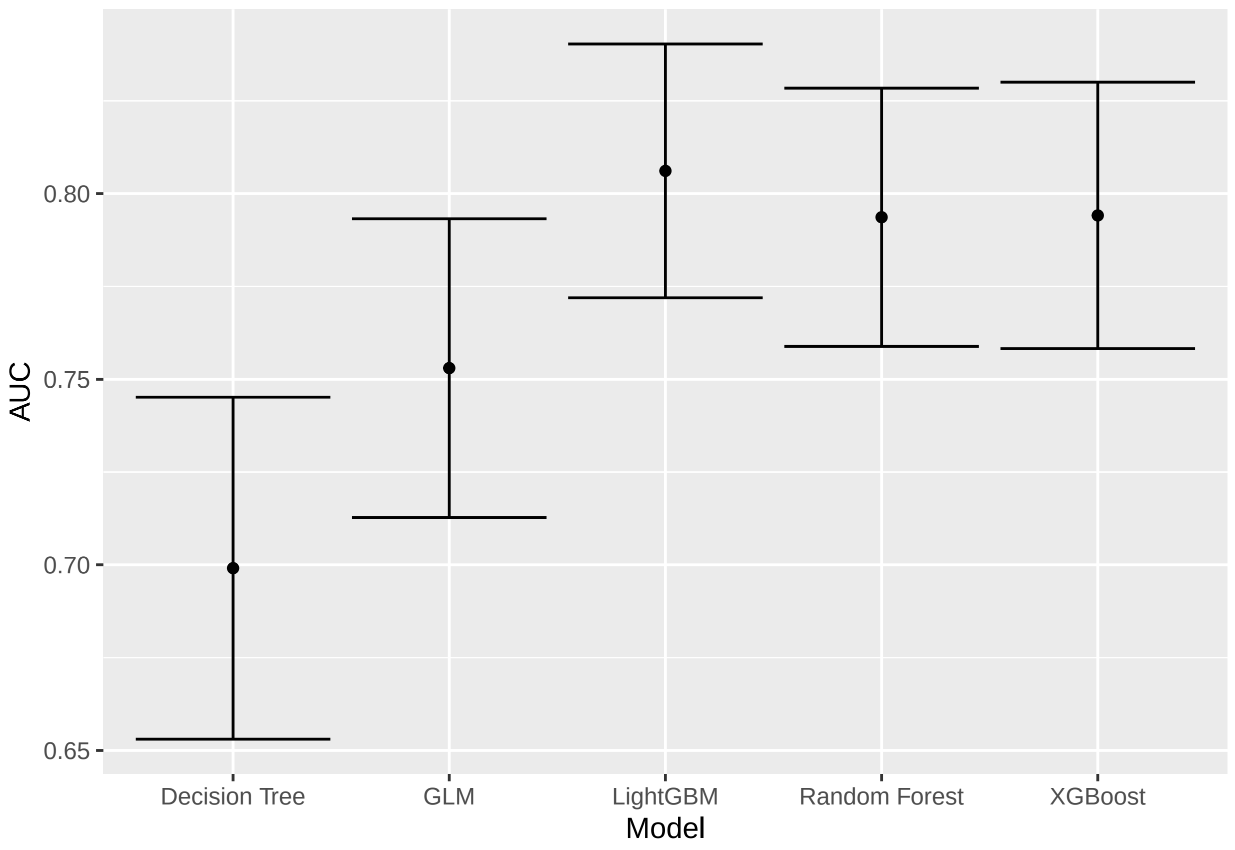


```
# extract_vip(final_rf_fit, pred_wrapper = predict,
#             reference_class = "1", use_matrix = FALSE,
#             method = 'permute')
```

Minutes to run: 62.738

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