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

Minutes to run: 0

Train test split (70%/30%)

```
set.seed(42)

if (outcome_column == 'readmission_30d') {
  df_split <- readRDS("./dataset/split_object.rds")
```

```
} else {
  df_split <- initial_split(df, prop = .7, strata = all_of(outcome_column))
}

df_train <- training(df_split) %>% dplyr::select(all_of(c(features, outcome_column)))
df_test <- testing(df_split) %>% dplyr::select(all_of(c(features, outcome_column)))

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

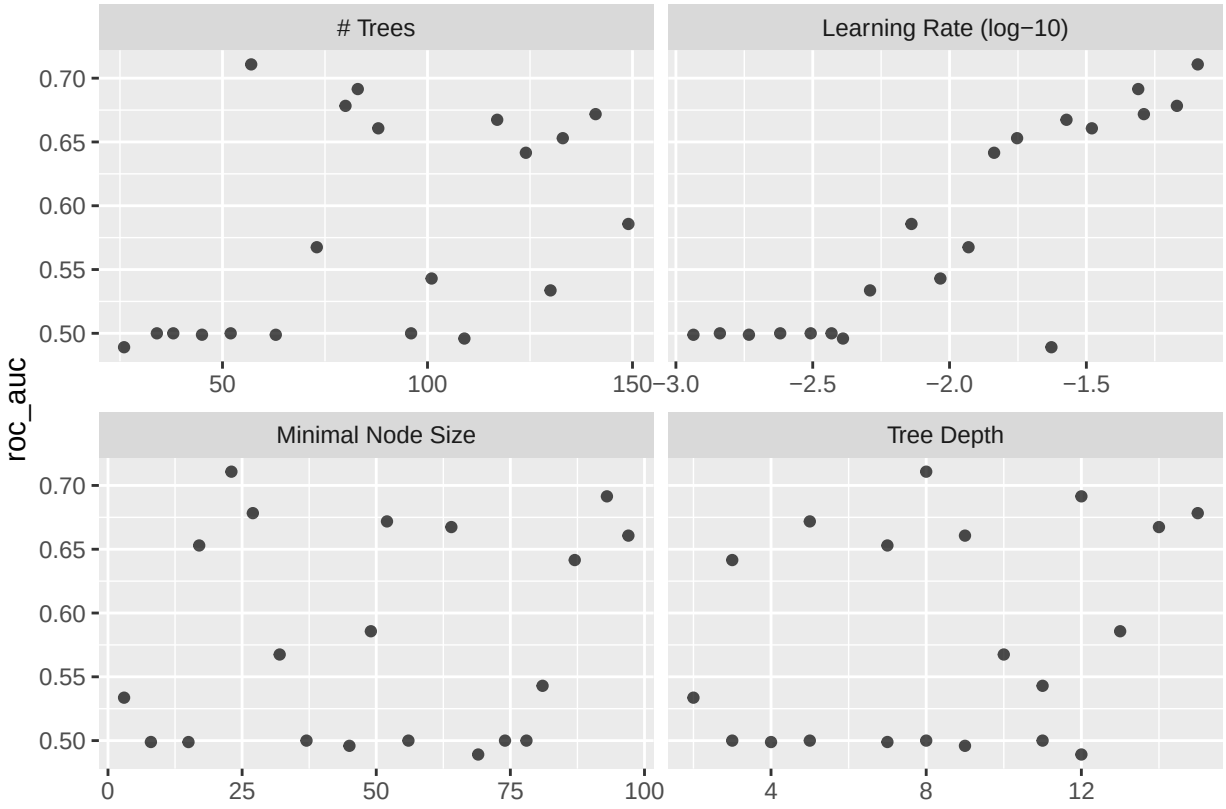
Minutes to run: 0.001

Boosted Tree (XGBoost)

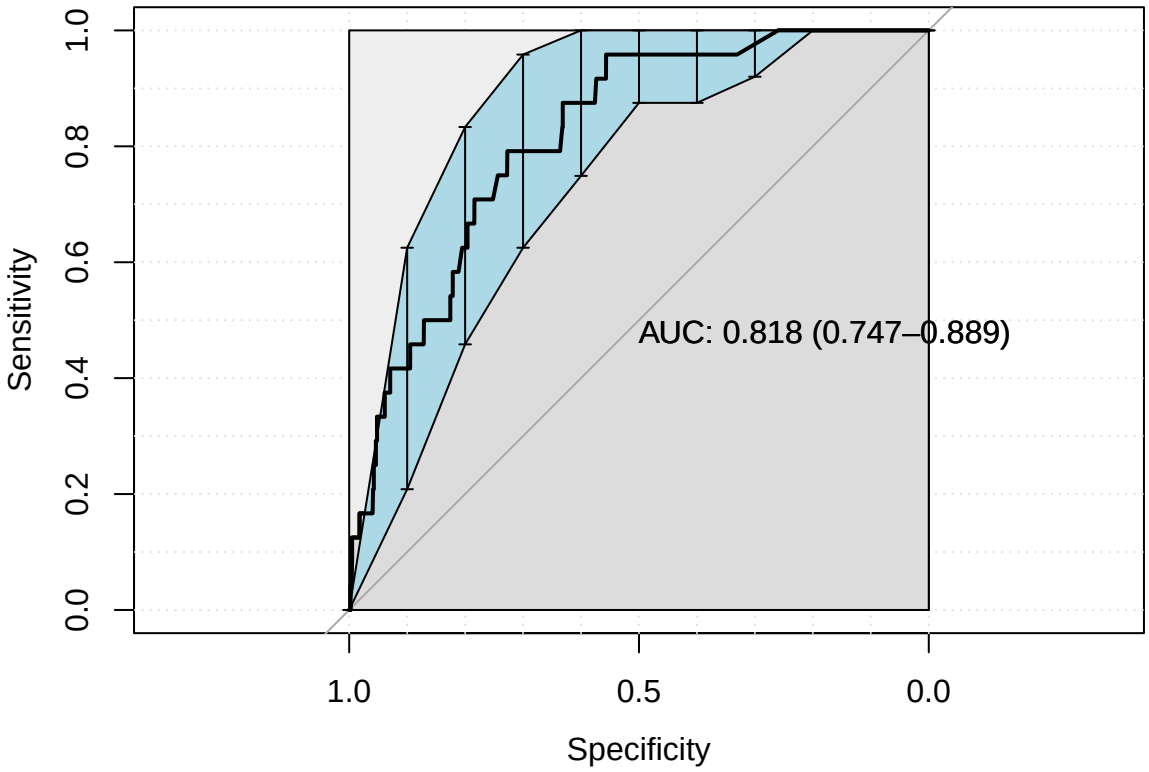
```
xgboost_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_other(all_nominal_predictors(), threshold = 0.05, other = ".merged") %>%
  step_dummy(all_nominal_predictors())

xgboost_spec <- boost_tree(
  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 Preprocessor1_Model05
## 2    83    93         12     0.0489 roc_auc binary     0.691    10  0.0277 Preprocessor1_Model19
## 3    80    27         15     0.0677 roc_auc binary     0.678    10  0.0662 Preprocessor1_Model06
## 4   141    52          5     0.0513 roc_auc binary     0.672    10  0.0479 Preprocessor1_Model11
## 5   117    64         14     0.0268 roc_auc binary     0.667    10  0.0450 Preprocessor1_Model13
```

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

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

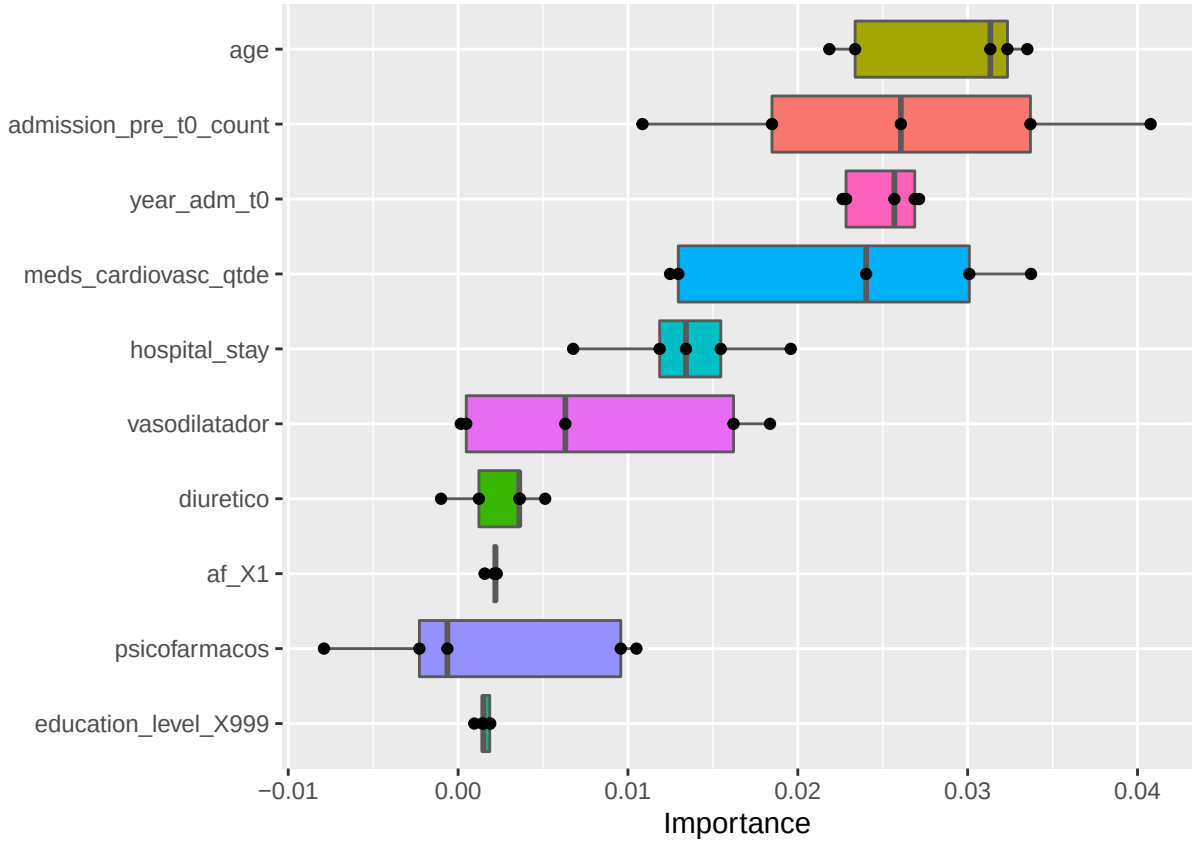



```
final_xgboost_workflow <-
  xgboost_workflow %>%
  finalize_workflow(best_xgboost)

last_xgboost_fit <-
  final_xgboost_workflow %>%
  last_fit(df_split)

final_xgboost_fit <- extract_workflow(last_xgboost_fit)

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




```
## [1] "Optimal Threshold: 0.01"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 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: 2.296

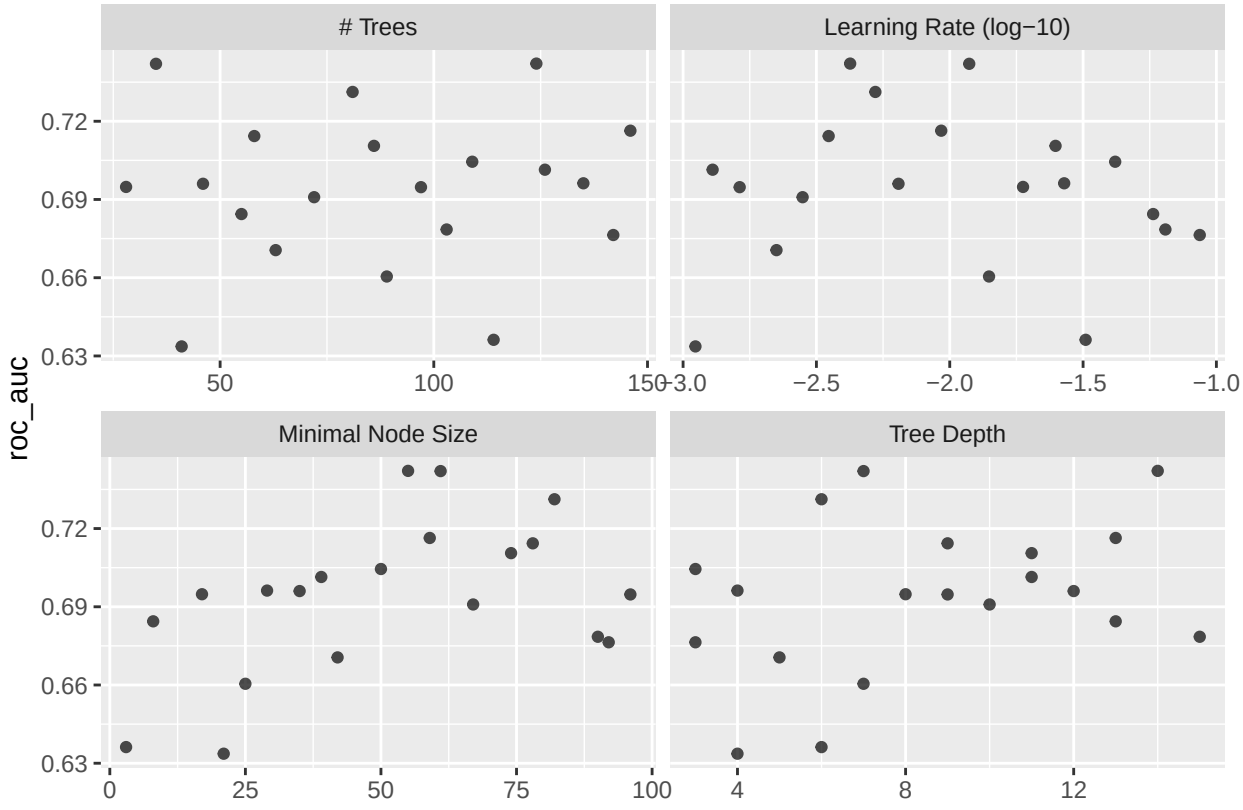
Boosted Tree (LightGBM)

```
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")
```

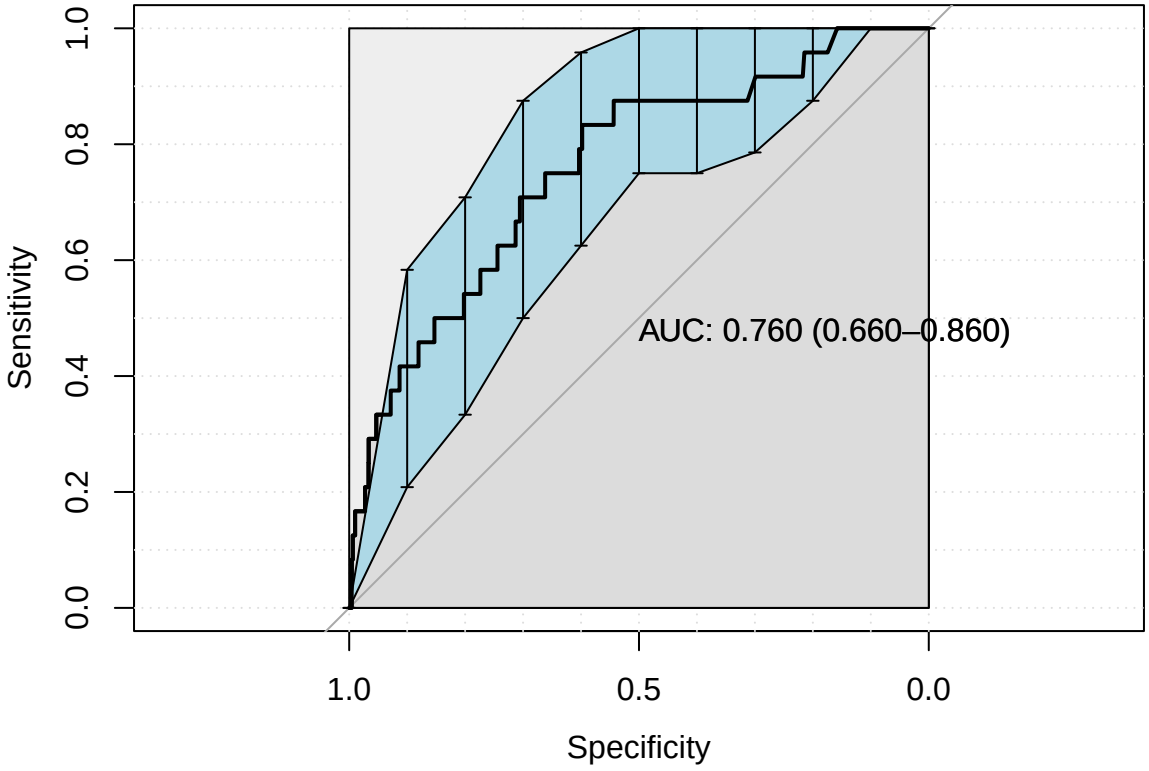
```
## # A tibble: 5 x 10
##   trees min_n tree_depth learn_rate .metric .estimator  mean     n std_err .config
##   <int> <int>      <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1   124    55         14    0.00423 roc_auc binary     0.742    10  0.0416 Preprocessor1_Model11
## 2    35    61          7    0.0118  roc_auc binary     0.742    10  0.0619 Preprocessor1_Model13
## 3    81    82          6    0.00526 roc_auc binary     0.731    10  0.0716 Preprocessor1_Model17
## 4   146    59         13    0.00929 roc_auc binary     0.716    10  0.0477 Preprocessor1_Model12
## 5    58    78          9    0.00352 roc_auc binary     0.714    10  0.0722 Preprocessor1_Model16
```

```
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 2814    4
##    1 1892   20
##
##                Accuracy : 0.5992
##                  95% CI : (0.585, 0.6132)
##     No Information Rate : 0.9949
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0107
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.59796
##             Specificity : 0.83333
##          Pos Pred Value : 0.99858
##          Neg Pred Value : 0.01046
##              Prevalence : 0.99493
##          Detection Rate : 0.59493
##    Detection Prevalence : 0.59577
##       Balanced Accuracy : 0.71565
##
```

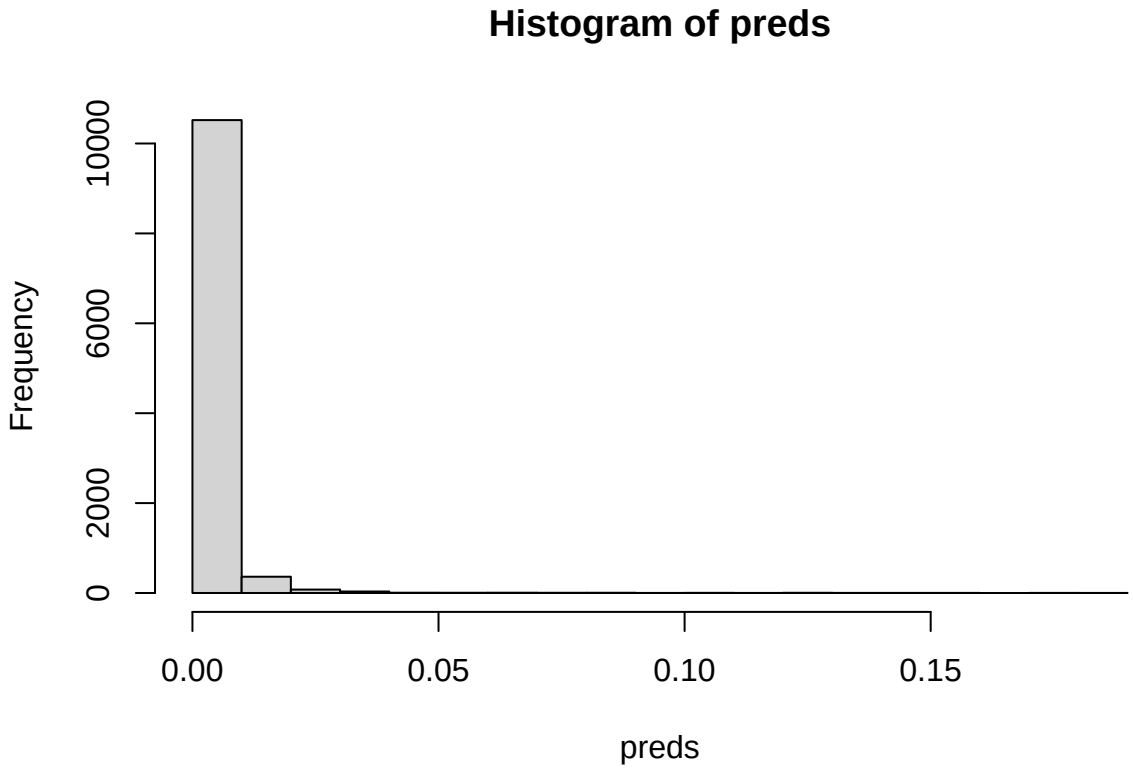
```
##        'Positive' Class : 0
##
```

```
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= 124
## .  min_n = integer 1= 55
## .  tree_depth = integer 1= 14
## .  learn_rate = double 1= 0.0042325
```

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

Minutes to run: 3.52

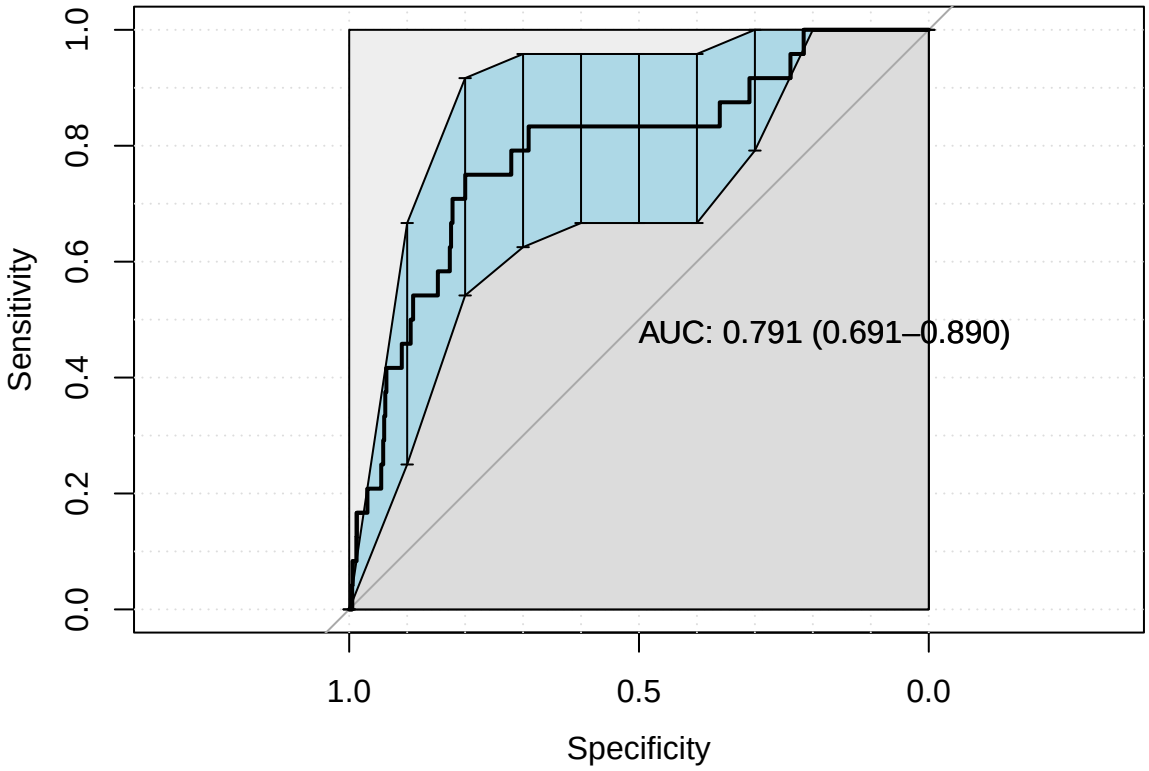


Minutes to run: 0.004

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())
```

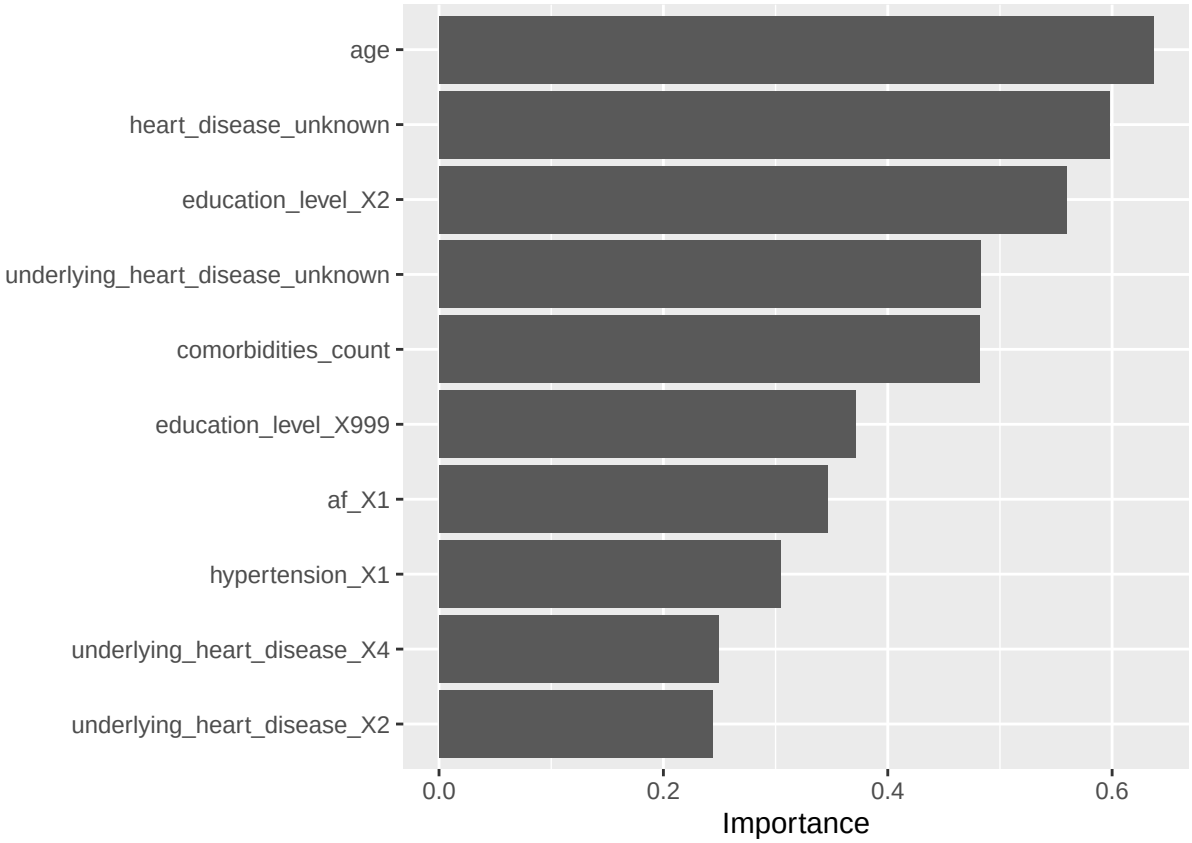
```r
glmnet_spec <-
  logistic_reg(penalty = 0) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

glmnet_workflow <-
  workflow() %>%
  add_recipe(glmnet_recipe) %>%
  add_model(glmnet_spec)

glm_fit <- glmnet_workflow %>%
  fit(df_train)

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



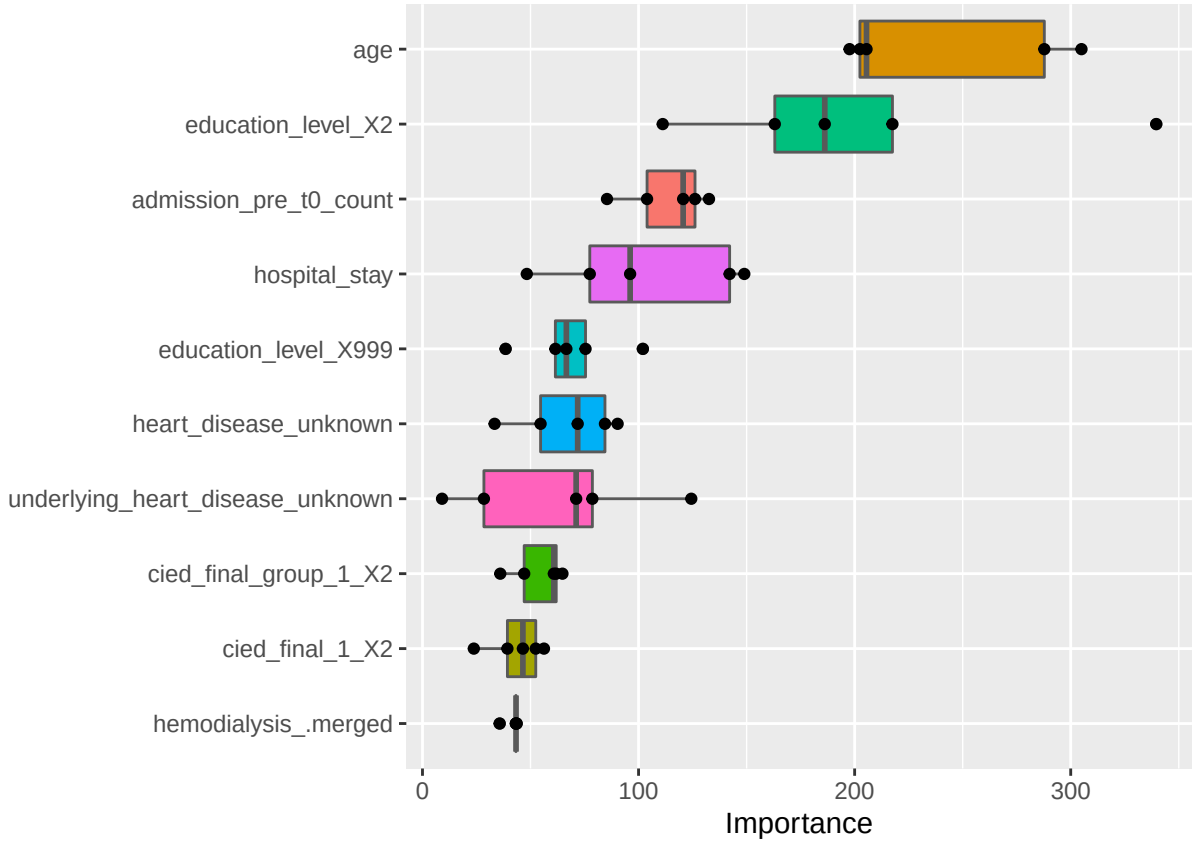

```
## [1] "Optimal Threshold: 0.01"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 3764    6
##    1  942   18
##
##                Accuracy : 0.7996
##                  95% CI : (0.7879, 0.8109)
##     No Information Rate : 0.9949
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.027
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.79983
```

```
##             Specificity : 0.75000
##          Pos Pred Value : 0.99841
##          Neg Pred Value : 0.01875
##              Prevalence : 0.99493
##          Detection Rate : 0.79577
##    Detection Prevalence : 0.79704
##       Balanced Accuracy : 0.77492
##
##        'Positive' Class : 0
##
```

```
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: 1.588

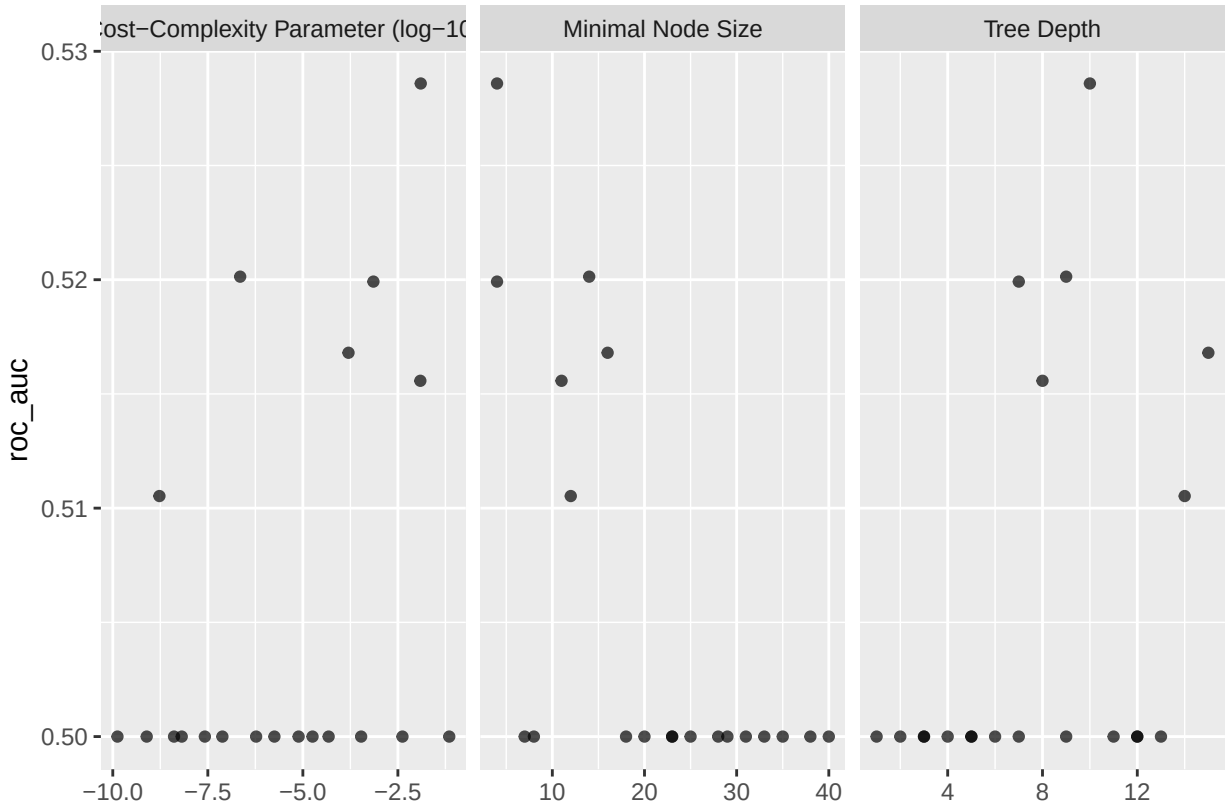
Decision Tree

```
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   0.0000000265           3    33 accuracy binary     0.996    10 0.000689 Preprocessor1_Model01
##  2   0.0000000265           3    33 roc_auc  binary     0.5      10 0        Preprocessor1_Model01
##  3   0.0126                10     4 accuracy binary     0.995    10 0.000832 Preprocessor1_Model02
##  4   0.0126                10     4 roc_auc  binary     0.529    10 0.0176   Preprocessor1_Model02
##  5   0.00000000409          5    20 accuracy binary     0.996    10 0.000689 Preprocessor1_Model03
##  6   0.00000000409          5    20 roc_auc  binary     0.5      10 0        Preprocessor1_Model03
##  7   0.00417                1     8 accuracy binary     0.996    10 0.000689 Preprocessor1_Model04
##  8   0.00417                1     8 roc_auc  binary     0.5      10 0        Preprocessor1_Model04
##  9   0.0000476              2    35 accuracy binary     0.996    10 0.000689 Preprocessor1_Model05
## 10   0.0000476              2    35 roc_auc  binary     0.5      10 0        Preprocessor1_Model05
## # ... with 30 more rows
## # i Use ‘print(n = ...)‘ to see more rows
```

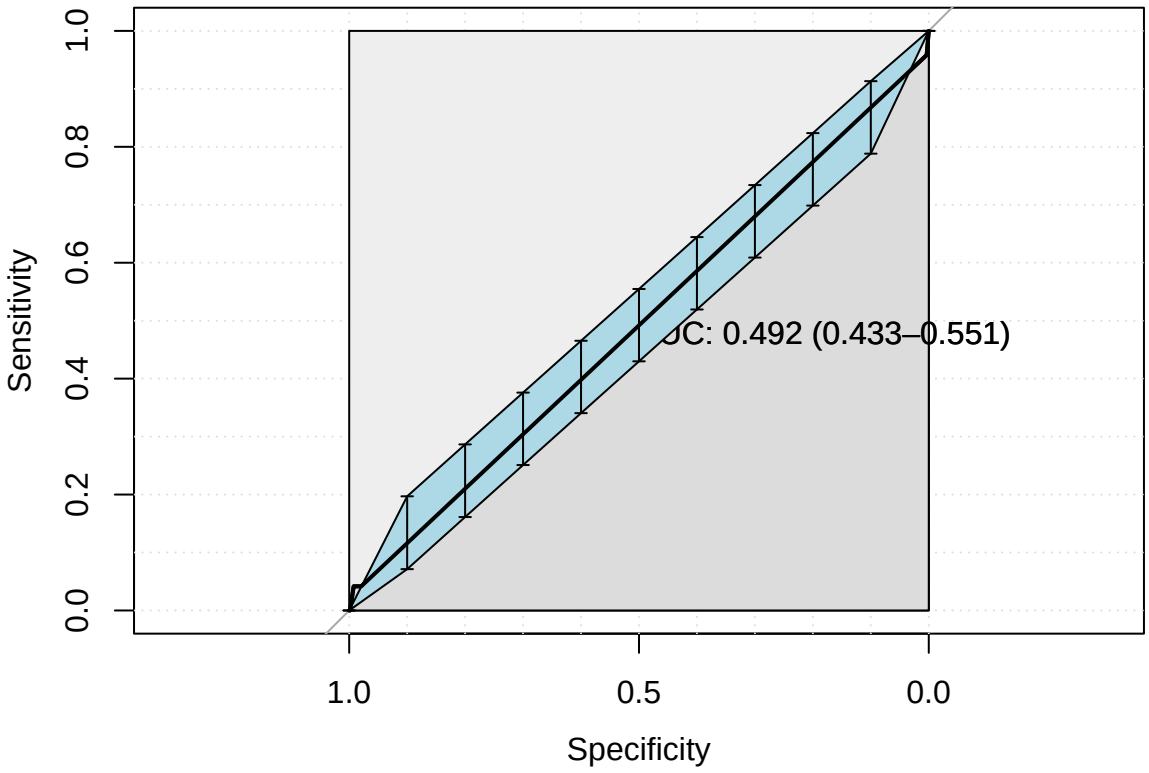
```
autoplot(tree_tune, metric = "roc_auc")
```




```
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     0.0126              10     4 roc_auc binary     0.529    10  0.0176 Preprocessor1_Model02
## 2     0.000000225          9    14 roc_auc binary     0.520    10  0.0165 Preprocessor1_Model12
## 3     0.000718             7     4 roc_auc binary     0.520    10  0.0175 Preprocessor1_Model06
## 4     0.000159            15    16 roc_auc binary     0.517    10  0.0163 Preprocessor1_Model10
## 5     0.0123               8    11 roc_auc binary     0.516    10  0.0164 Preprocessor1_Model19
```

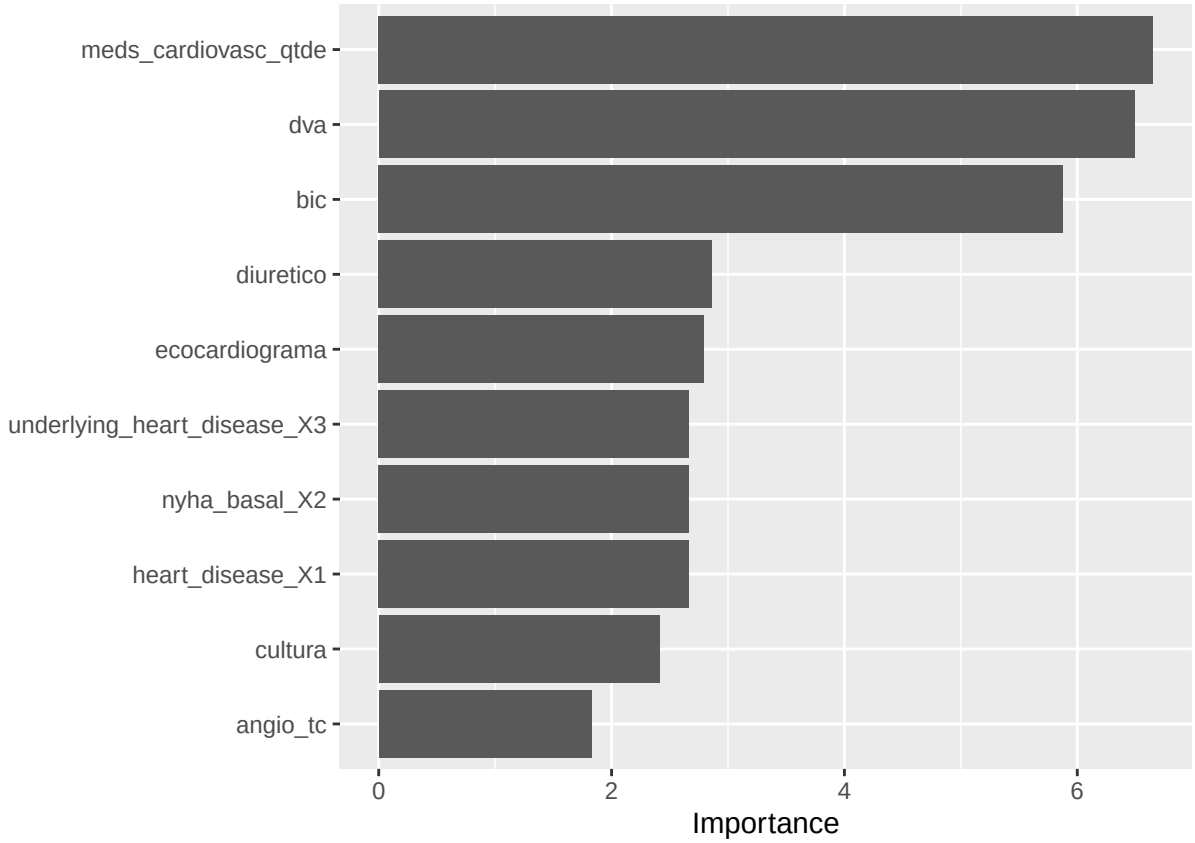
```r
best_tree <- tree_tune %>%
  select_best("roc_auc")

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 4670   23
##    1   36    1
##
##                Accuracy : 0.9875
##                  95% CI : (0.9839, 0.9905)
##     No Information Rate : 0.9949
##     P-Value [Acc > NIR] : 1.0000
##
##                   Kappa : 0.0268
##
##  Mcnemar's Test P-Value : 0.1182
##
##             Sensitivity : 0.99235
```

```
##             Specificity : 0.04167
##          Pos Pred Value : 0.99510
##          Neg Pred Value : 0.02703
##              Prevalence : 0.99493
##          Detection Rate : 0.98732
##    Detection Prevalence : 0.99218
##       Balanced Accuracy : 0.51701
##
##        '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: 2.213

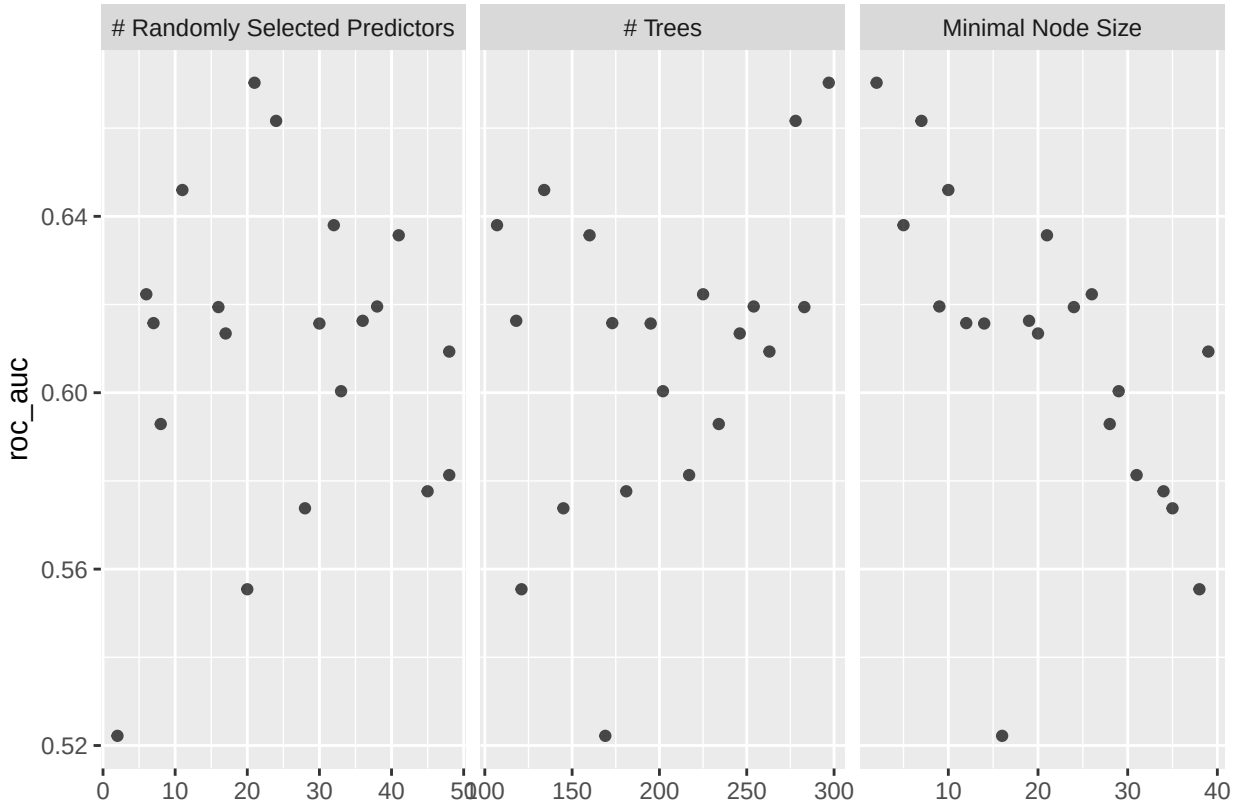
Random Forest

```
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    45   181    34 accuracy binary     0.996    10 0.000689 Preprocessor1_Model01
##  2    45   181    34 roc_auc  binary     0.578    10 0.0390   Preprocessor1_Model01
##  3    21   297     2 accuracy binary     0.996    10 0.000689 Preprocessor1_Model02
##  4    21   297     2 roc_auc  binary     0.670    10 0.0494   Preprocessor1_Model02
##  5     2   169    16 accuracy binary     0.996    10 0.000689 Preprocessor1_Model03
##  6     2   169    16 roc_auc  binary     0.522    10 0.0260   Preprocessor1_Model03
##  7    28   145    35 accuracy binary     0.996    10 0.000689 Preprocessor1_Model04
##  8    28   145    35 roc_auc  binary     0.574    10 0.0386   Preprocessor1_Model04
##  9    17   246    20 accuracy binary     0.996    10 0.000689 Preprocessor1_Model05
## 10    17   246    20 roc_auc  binary     0.613    10 0.0472   Preprocessor1_Model05
## # ... with 30 more rows
## # i Use `print(n = ...)` to see more rows
```

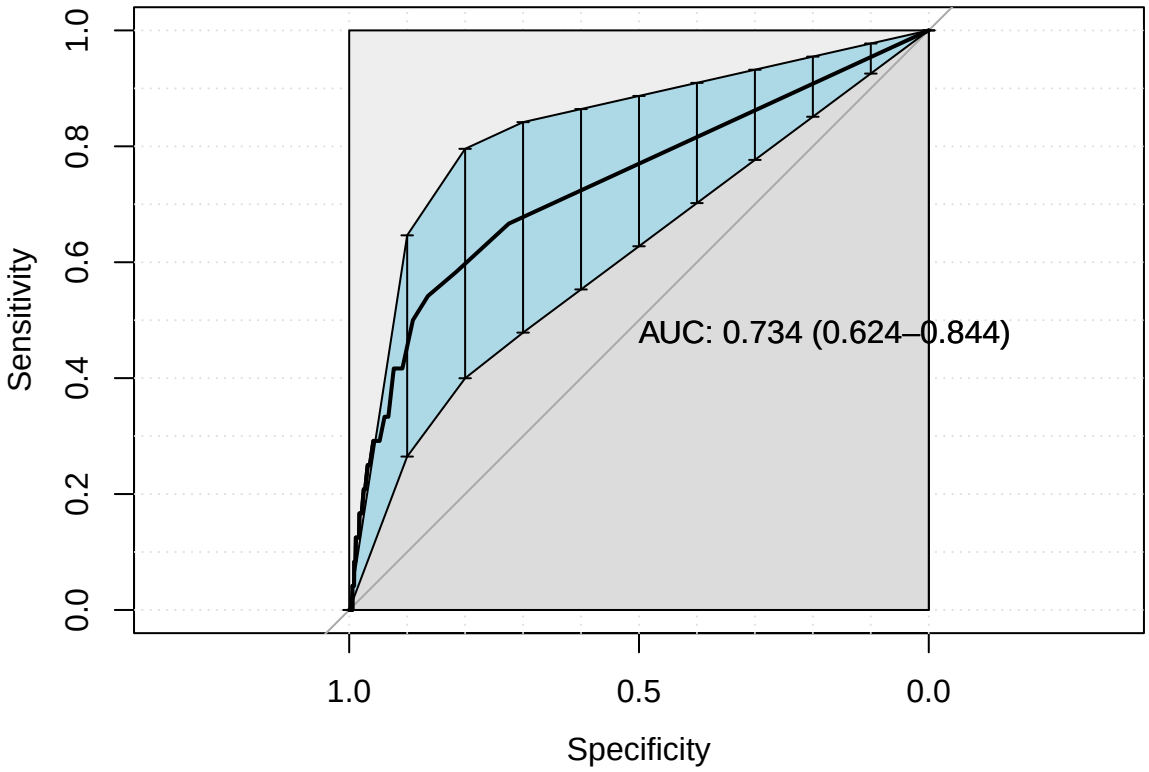
```
autoplot(rf_tune, metric = "roc_auc")
```




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

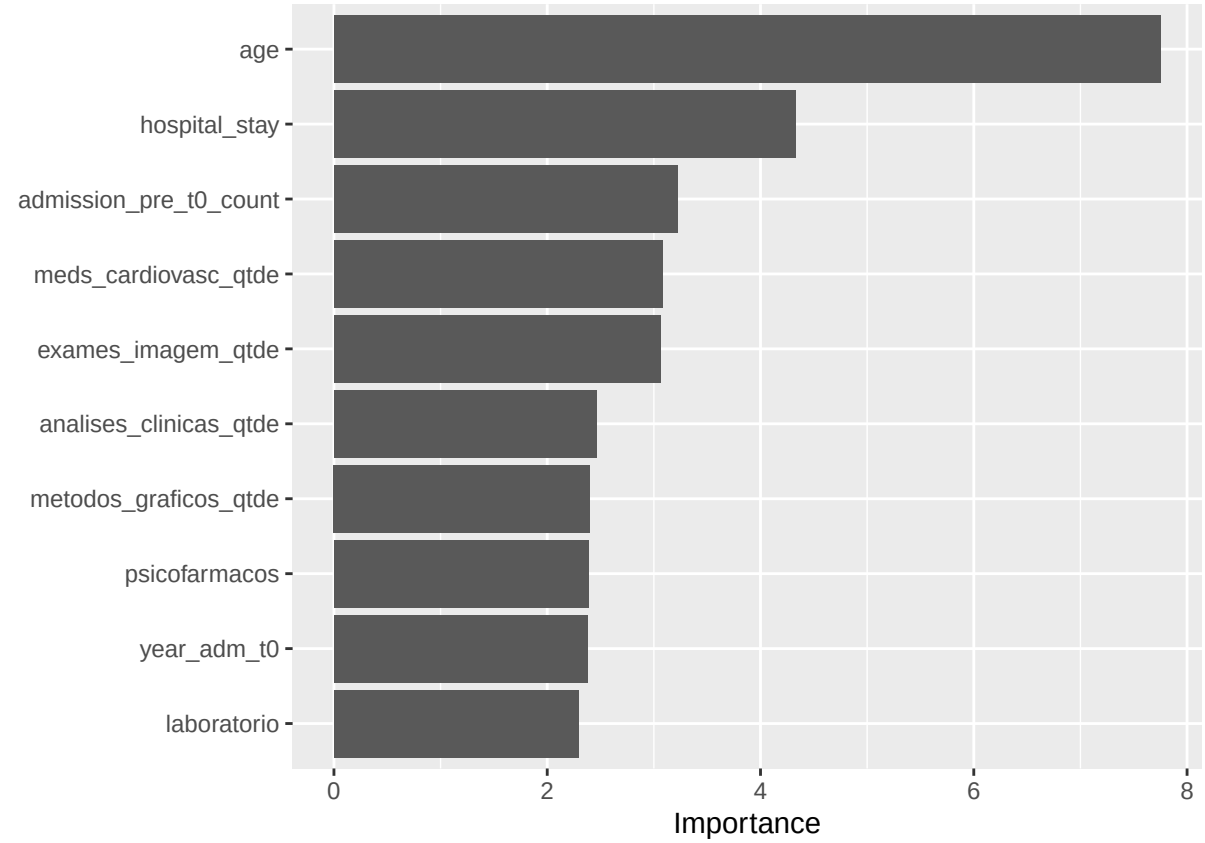
```
## # A tibble: 5 x 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    21   297     2 roc_auc binary     0.670    10  0.0494 Preprocessor1_Model02
## 2    24   278     7 roc_auc binary     0.662    10  0.0568 Preprocessor1_Model17
## 3    11   134    10 roc_auc binary     0.646    10  0.0390 Preprocessor1_Model08
## 4    32   107     5 roc_auc binary     0.638    10  0.0485 Preprocessor1_Model12
## 5    41   160    21 roc_auc binary     0.636    10  0.0527 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: 0.01"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 4067   11
##    1  639   13
##
##                Accuracy : 0.8626
##                  95% CI : (0.8524, 0.8723)
##     No Information Rate : 0.9949
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.029
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.86422
##             Specificity : 0.54167
##          Pos Pred Value : 0.99730
##          Neg Pred Value : 0.01994
##              Prevalence : 0.99493
##          Detection Rate : 0.85983
##    Detection Prevalence : 0.86216
##       Balanced Accuracy : 0.70294
##
##        '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')
```

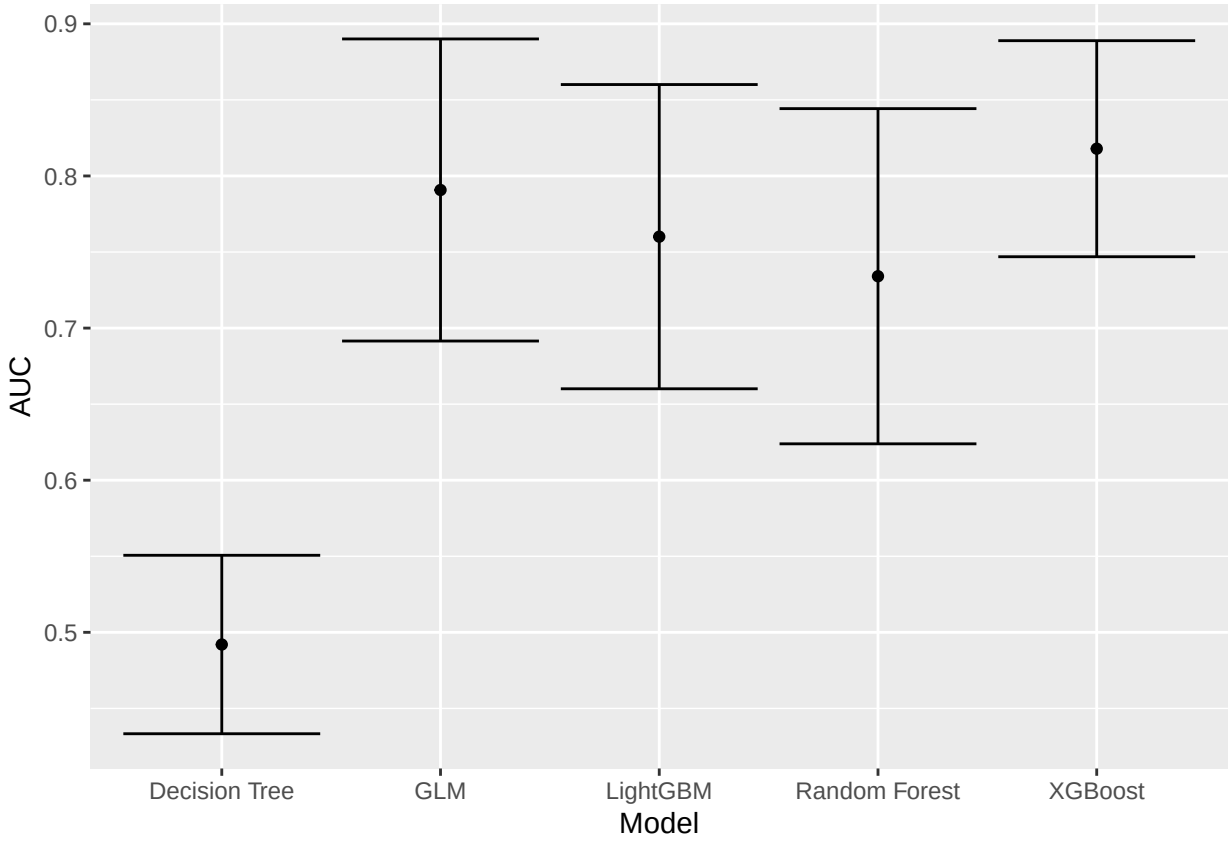



```
# extract_vip(final_rf_fit, pred_wrapper = predict,
#             reference_class = "1", use_matrix = FALSE,
#             method = 'permute')
```

Minutes to run: 37.33

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