

# 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 = readRDS(sprintf(
  "./auxiliar/significant_columns/categorical_%s.rds",
  outcome_column
))

num_features_list = readRDS(sprintf(
  "./auxiliar/significant_columns/numerical_%s.rds",
  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. espirolactona  
## 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: 0.001

## 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,
```

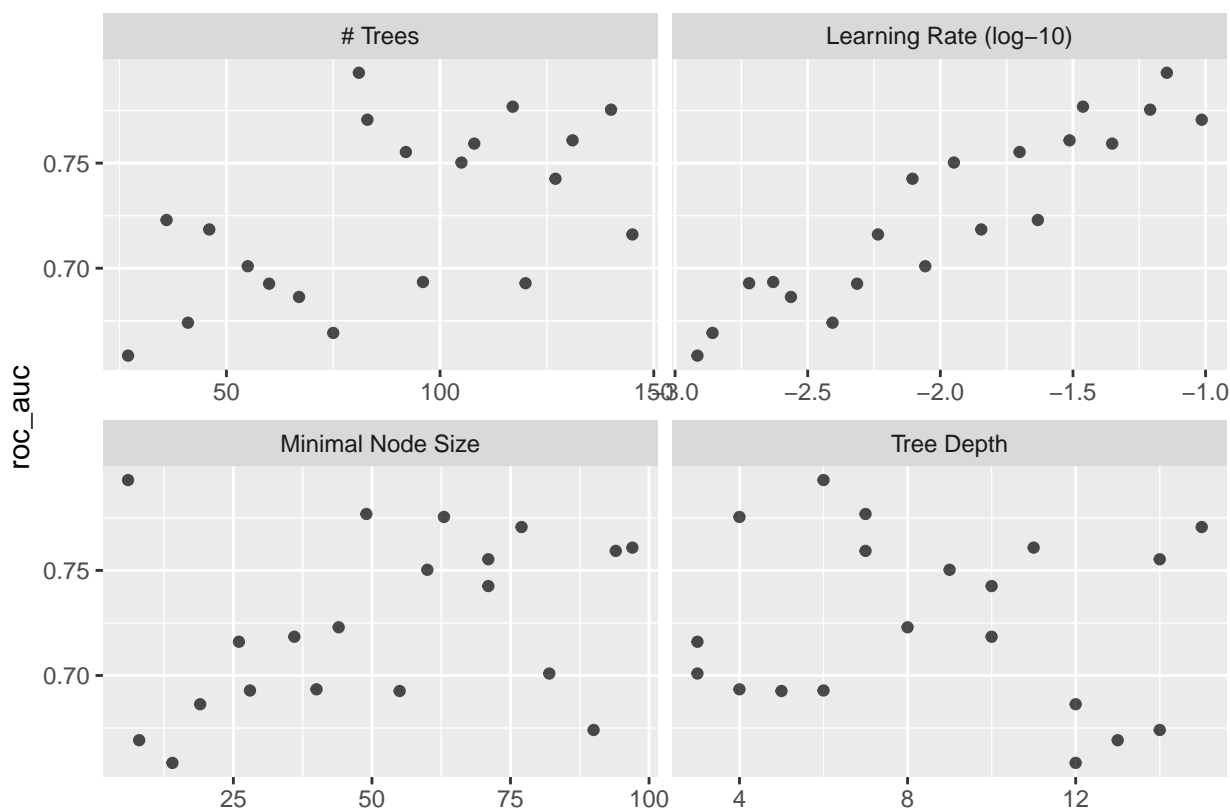
```
grid = xgboost_grid)
```

```
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    81     6        6    0.0714 roc_auc binary   0.793    10 0.00808 Preprocessor1_Model101
## 2   117    49        7    0.0345 roc_auc binary   0.777    10 0.00733 Preprocessor1_Model110
## 3   140    63        4    0.0618 roc_auc binary   0.775    10 0.00668 Preprocessor1_Model113
## 4    83    77       15    0.0967 roc_auc binary   0.771    10 0.00763 Preprocessor1_Model116
## 5   131    97       11    0.0307 roc_auc binary   0.761    10 0.00743 Preprocessor1_Model120
```

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

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

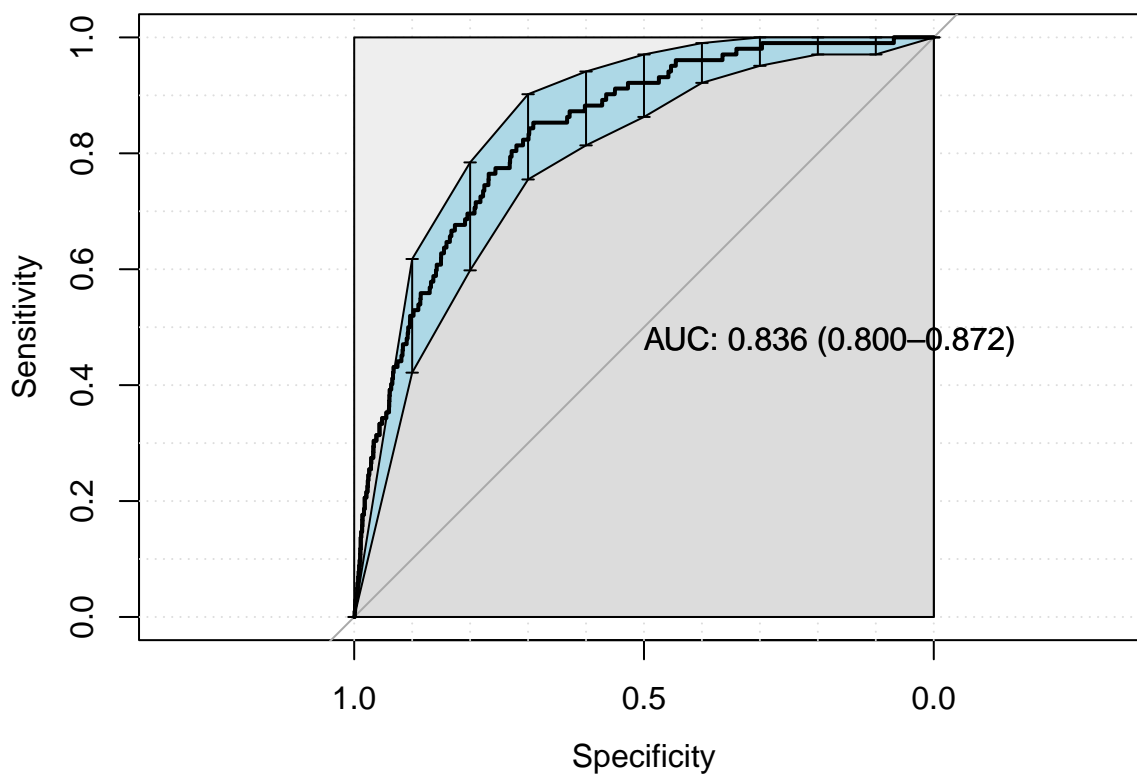


```
final_xgboost_workflow <-
  xgboost_workflow %>%
  finalize_workflow(best_xgboost)

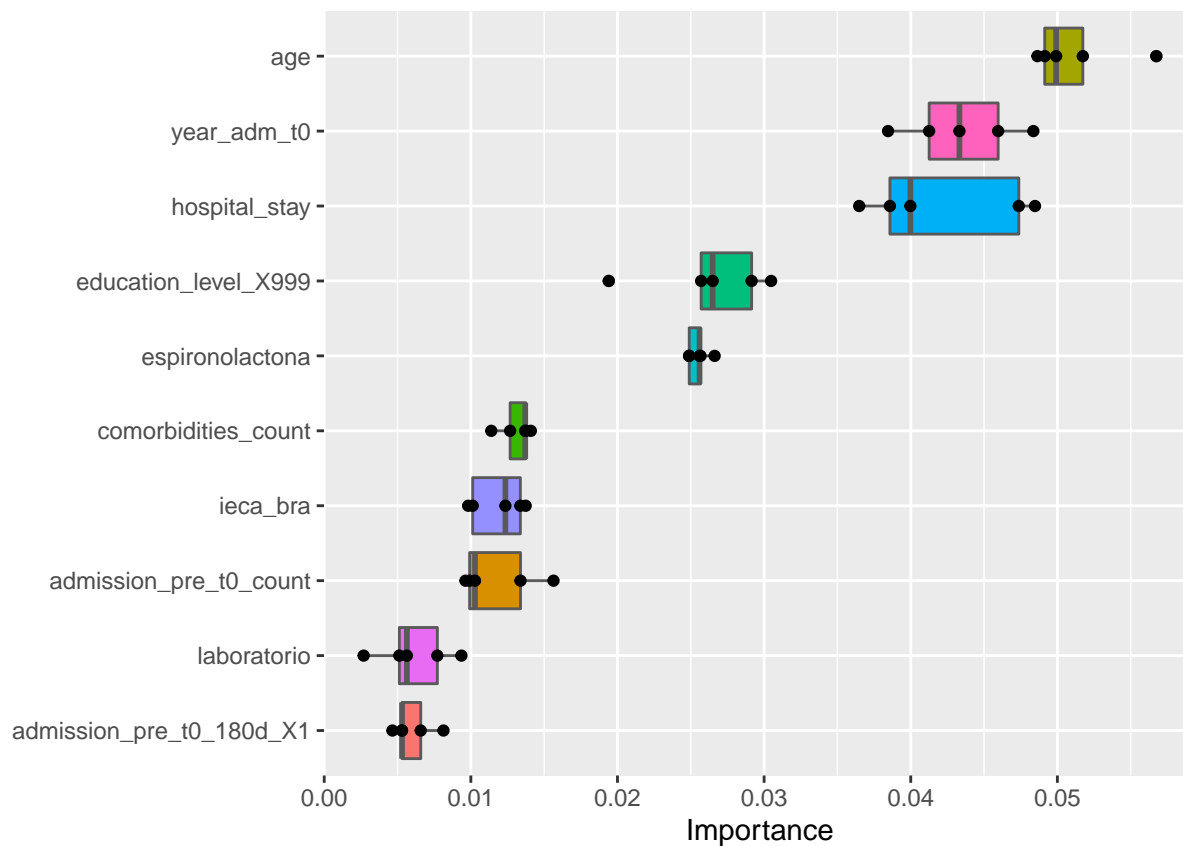
last_xgboost_fit <-
  final_xgboost_workflow %>%
  last_fit(df_split)

final_xgboost_fit <- extract_workflow(last_xgboost_fit)

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



```
## [1] "Optimal Threshold: 0.02"
## 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
##
extract_vip(final_xgboost_fit, pred_wrapper = predict,
            reference_class = "0")
```



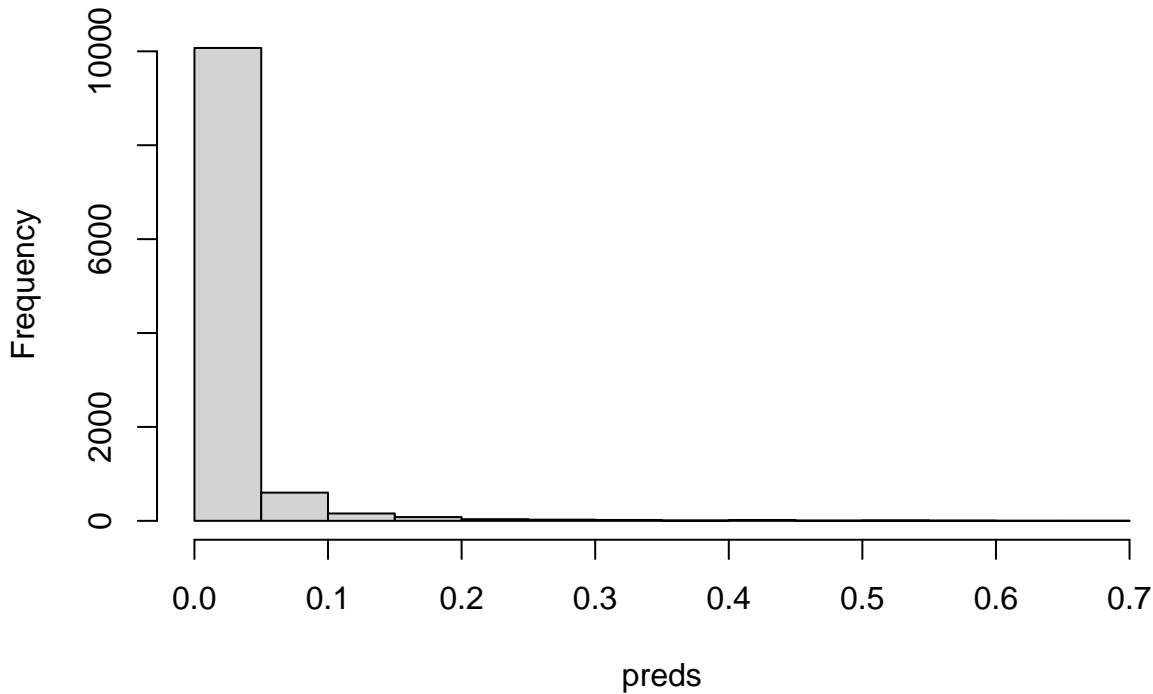
```
xgboost_parameters <- xgboost_tune %>%
  show_best("roc_auc", n = 1) %>%
  select(trees, min_n, tree_depth, learn_rate) %>%
  as.list

saveRDS(
  xgboost_parameters,
  file = sprintf(
    "./auxiliar/model_selection/hyperparameters/xgboost_%s.rds",
    outcome_column
  )
)

preds <- predict(final_xgboost_fit, new_data = df_train, type = "prob") %>%
  rename_at(vars(starts_with(".pred_")), ~ str_remove(., ".pred_")) %>%
  .$`1`

hist(preds)
```

## Histogram of preds



Minutes to run:

4.851

## 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)

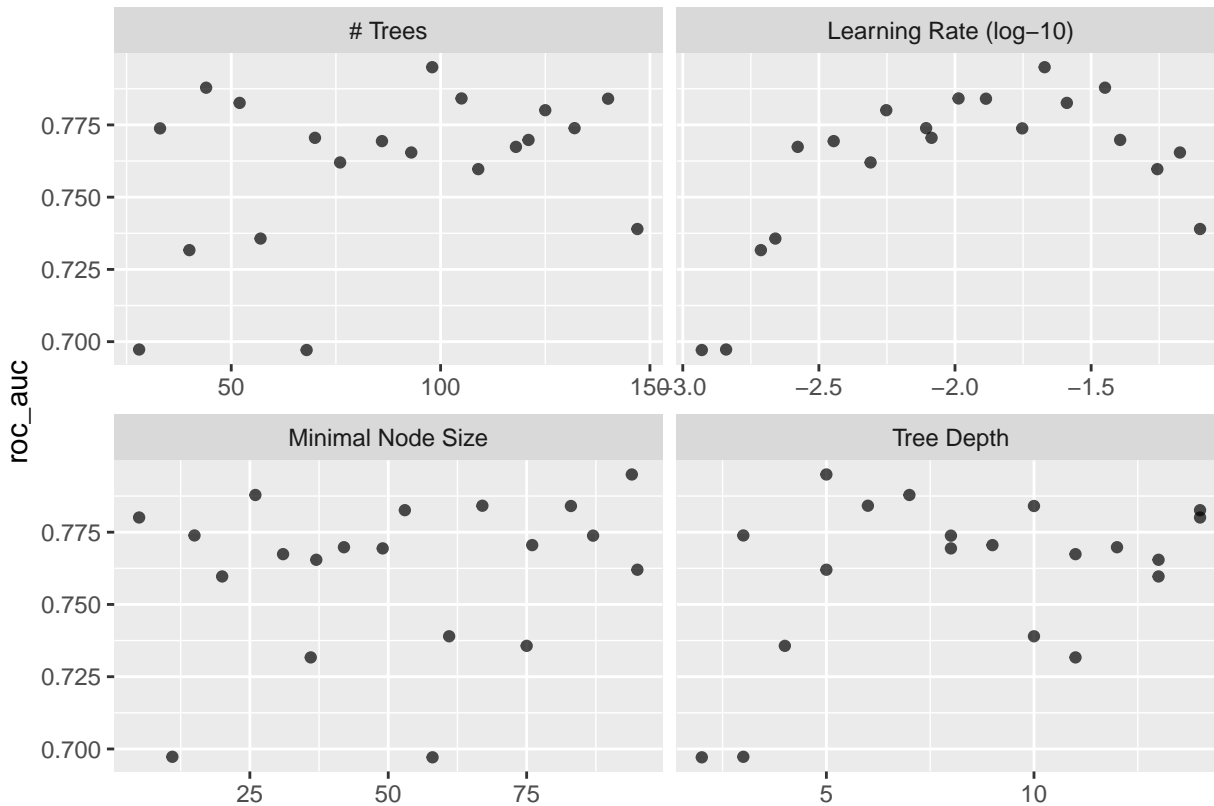
lightgbm_tune %>%
  show_best("roc_auc")

## # 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 Preprocessor1_Model119
## 2    44    26        7    0.0356 roc_auc binary  0.788    10 0.00979 Preprocessor1_Model105
## 3   105    67        6    0.0103 roc_auc binary  0.784    10 0.00593 Preprocessor1_Model114
## 4   140    83       10    0.0130 roc_auc binary  0.784    10 0.00616 Preprocessor1_Model117
## 5    52    53       14    0.0258 roc_auc binary  0.783    10 0.00737 Preprocessor1_Model111

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

autoplot(lightgbm_tune, metric = "roc_auc")

```



```

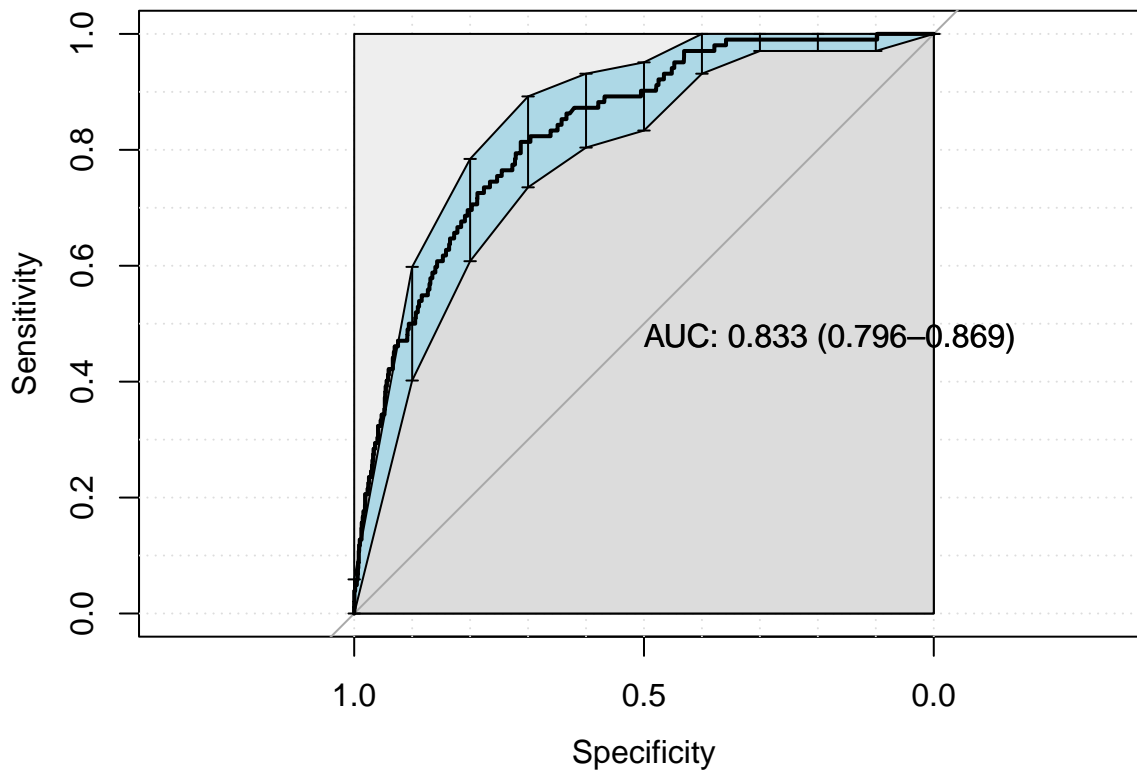
final_lightgbm_workflow <-
  lightgbm_workflow %>%
  finalize_workflow(best_lightgbm)

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 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
##
##              '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)

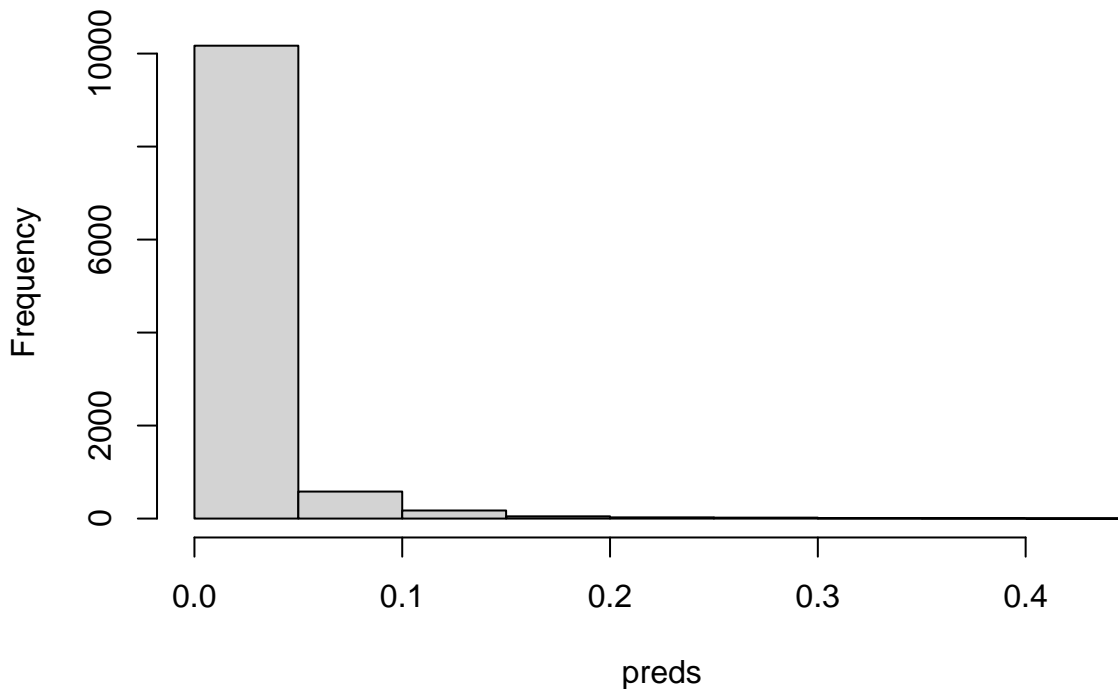
## lightgbm_parameters = list 4 (736 bytes)
## . trees = integer 1= 98
## . min_n = integer 1= 94
## . tree_depth = integer 1= 5
## . learn_rate = double 1= 0.021341

saveRDS(
  lightgbm_parameters,
  file = sprintf(
    "./auxiliar/model_selection/hyperparameters/lightgbm_%s.rds",
    outcome_column
  )
)

```

Minutes to run: 3.576

## Histogram of preds



Minutes to run:

0.006

## 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())

glmnet_spec <-

```

```

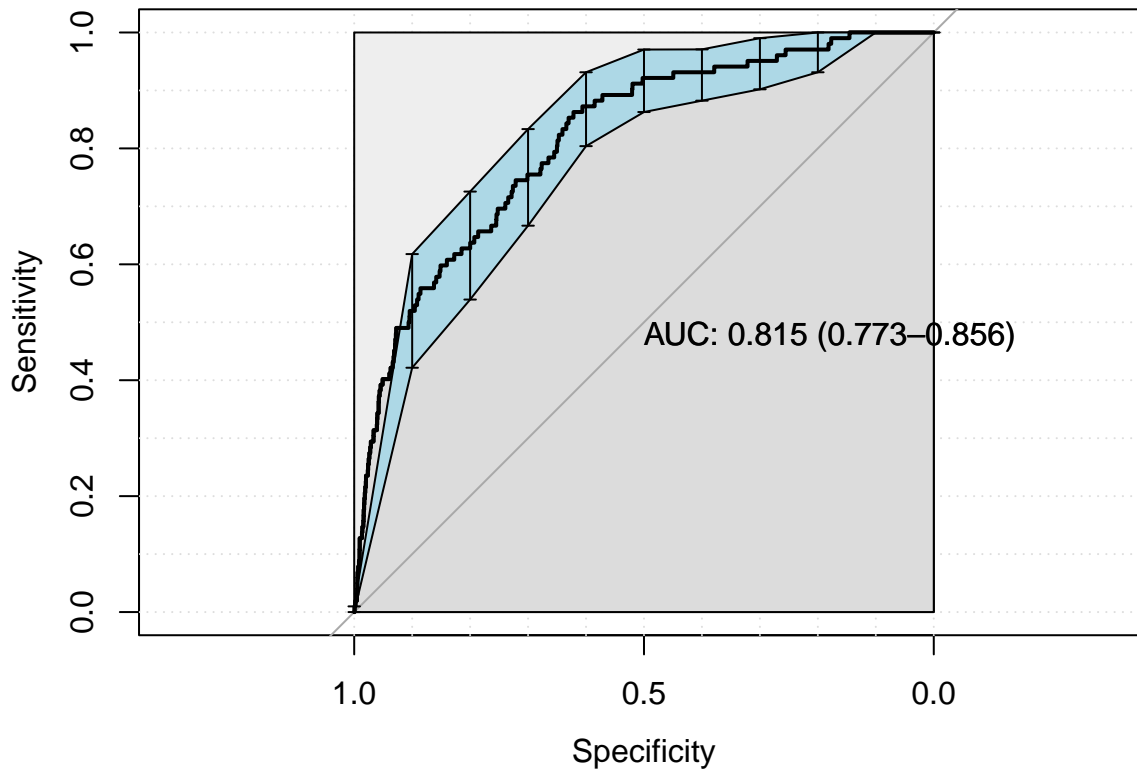
logistic_reg(penalty = 0) %>%
set_mode("classification") %>%
set_engine("glmnet")

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)

```



```

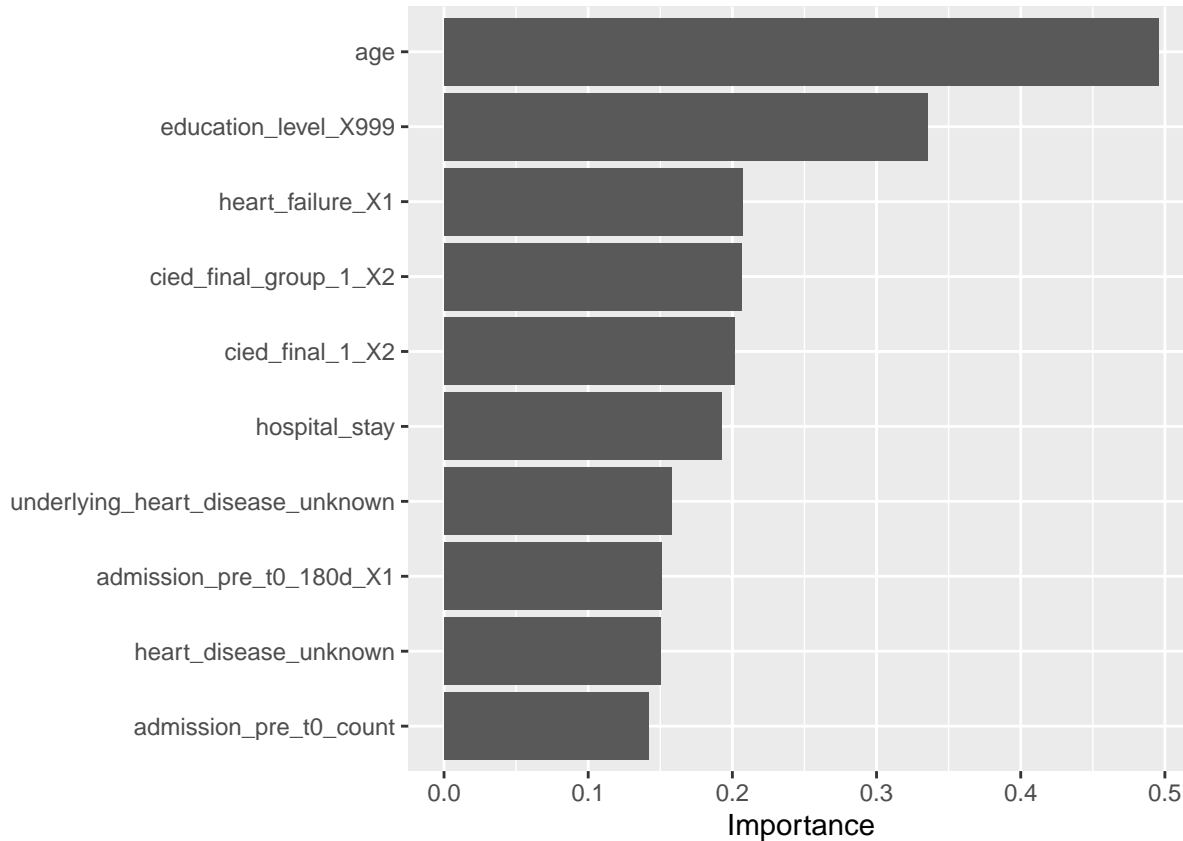
## [1] "Optimal Threshold: 0.02"
## Confusion Matrix and Statistics
##
##      reference
## data    0    1
## 0 2877   14
## 1 1751   88
##
##              Accuracy : 0.6268
##              95% CI   : (0.6129, 0.6407)
##    No Information Rate : 0.9784
##    P-Value [Acc > NIR] : 1
##
##              Kappa   : 0.0519
##
##  Mcnemar's Test P-Value : <2e-16
##
##              Sensitivity : 0.62165
##              Specificity : 0.86275

```

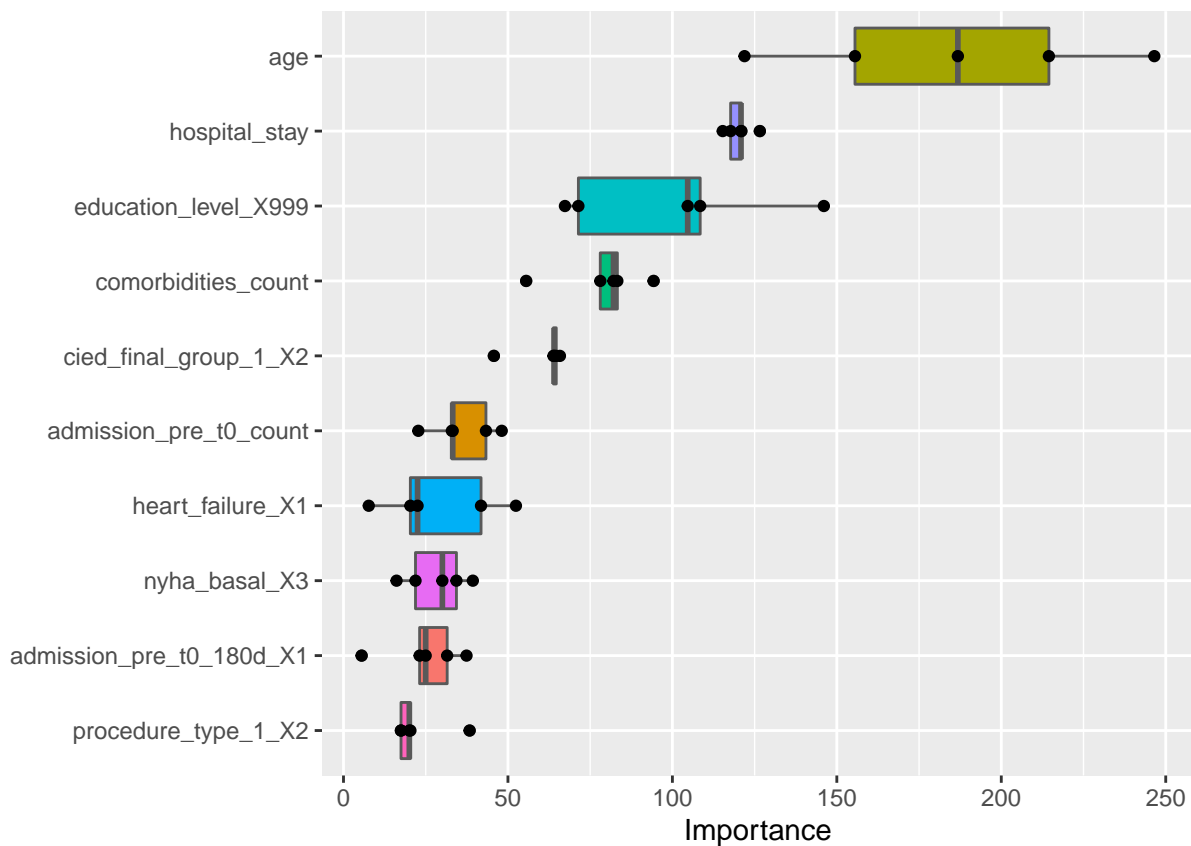
```
##      Pos Pred Value : 0.99516
##      Neg Pred Value : 0.04785
##      Prevalence     : 0.97844
##      Detection Rate  : 0.60825
##      Detection Prevalence : 0.61121
##      Balanced Accuracy : 0.74220
##
##      'Positive' Class : 0
##
```

```
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:

1.914

## 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)

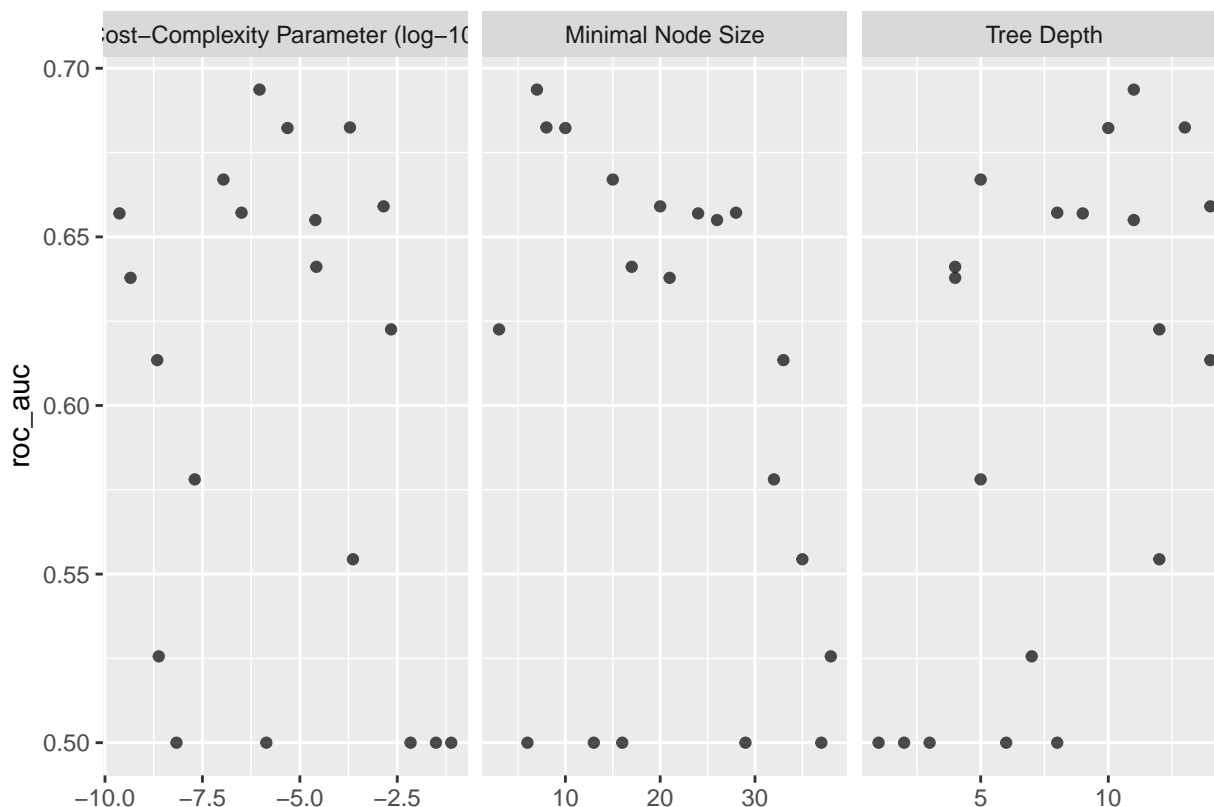
tree_workflow <-
  workflow() %>%
  add_recipe(tree_recipe) %>%
  add_model(tree_spec)

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      2.40e- 9           7    38 accuracy binary    0.979   10 0.00127 Preprocessor1_Model101
## 2      2.40e- 9           7    38 roc_auc  binary    0.526   10 0.0188  Preprocessor1_Model101
## 3      1.10e- 7           5    15 accuracy binary    0.978   10 0.00139 Preprocessor1_Model102
## 4      1.10e- 7           5    15 roc_auc  binary    0.667   10 0.0177  Preprocessor1_Model102
## 5      2.31e- 4          12    35 accuracy binary    0.979   10 0.00137 Preprocessor1_Model103
## 6      2.31e- 4          12    35 roc_auc  binary    0.554   10 0.0277  Preprocessor1_Model103
## 7      2.35e-10           9    24 accuracy binary    0.978   10 0.00130 Preprocessor1_Model104
## 8      2.35e-10           9    24 roc_auc  binary    0.657   10 0.0153  Preprocessor1_Model104
## 9      6.94e- 3           3    13 accuracy binary    0.979   10 0.00124 Preprocessor1_Model105
## 10     6.94e- 3           3    13 roc_auc  binary    0.5     10 0       Preprocessor1_Model105
## # ... with 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      0.000000923          11     7 roc_auc  binary    0.694   10 0.0149 Preprocessor1_Model114
## 2      0.000193           13     8 roc_auc  binary    0.682   10 0.0206 Preprocessor1_Model106
## 3      0.00000483          10    10 roc_auc  binary    0.682   10 0.0213 Preprocessor1_Model120
## 4      0.000000110           5    15 roc_auc  binary    0.667   10 0.0177 Preprocessor1_Model102
## 5      0.00141            14    20 roc_auc  binary    0.659   10 0.0199 Preprocessor1_Model113
```

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

```

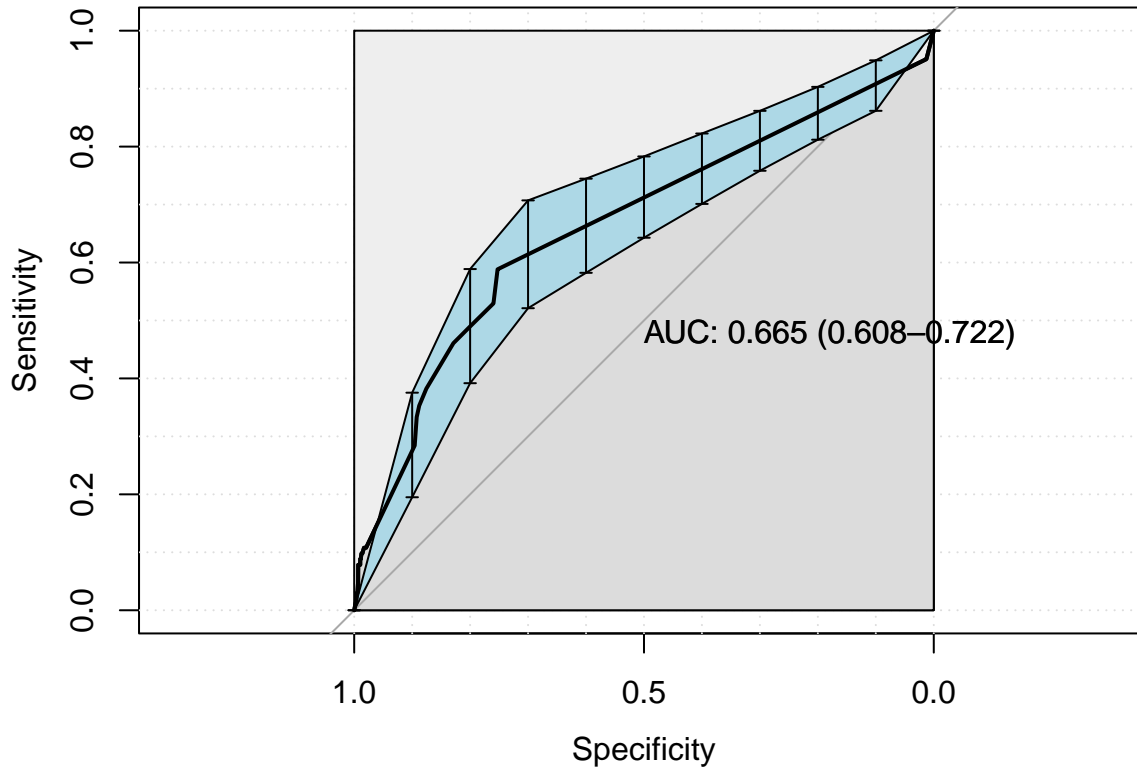
final_tree_workflow <-
  tree_workflow %>%
  finalize_workflow(best_tree)

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)

```



```

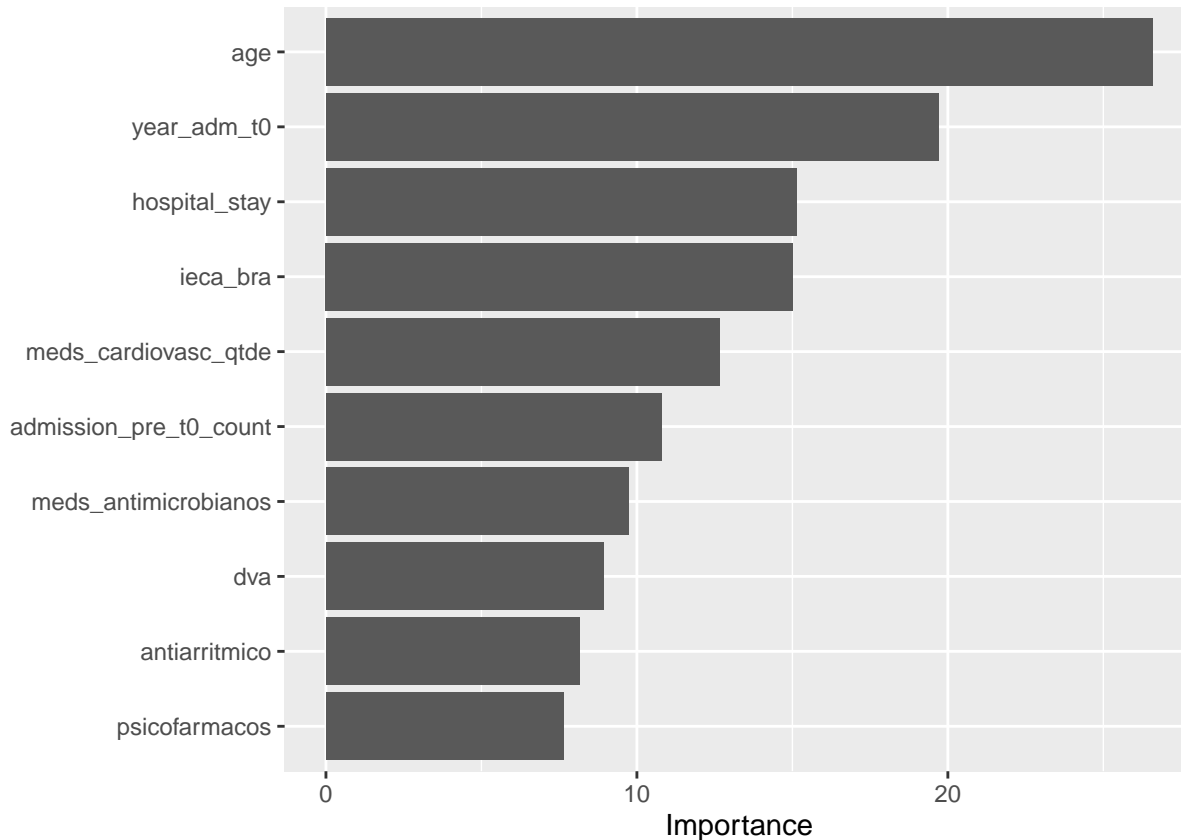
## [1] "Optimal Threshold: 0.01"
## Confusion Matrix and Statistics
##
##      reference
## data    0    1
## 0 3483   42
## 1 1145   60
##
##              Accuracy : 0.749
##              95% CI : (0.7364, 0.7614)
##      No Information Rate : 0.9784
##      P-Value [Acc > NIR] : 1
##
##              Kappa : 0.0542
##
##  Mcnemar's Test P-Value : <2e-16
##
##              Sensitivity : 0.75259
##              Specificity : 0.58824
##      Pos Pred Value : 0.98809
##      Neg Pred Value : 0.04979

```



```
##           Prevalence : 0.97844
##           Detection Rate : 0.73636
##           Detection Prevalence : 0.74524
##           Balanced Accuracy : 0.67041
##
##           '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: 4.234

## 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)

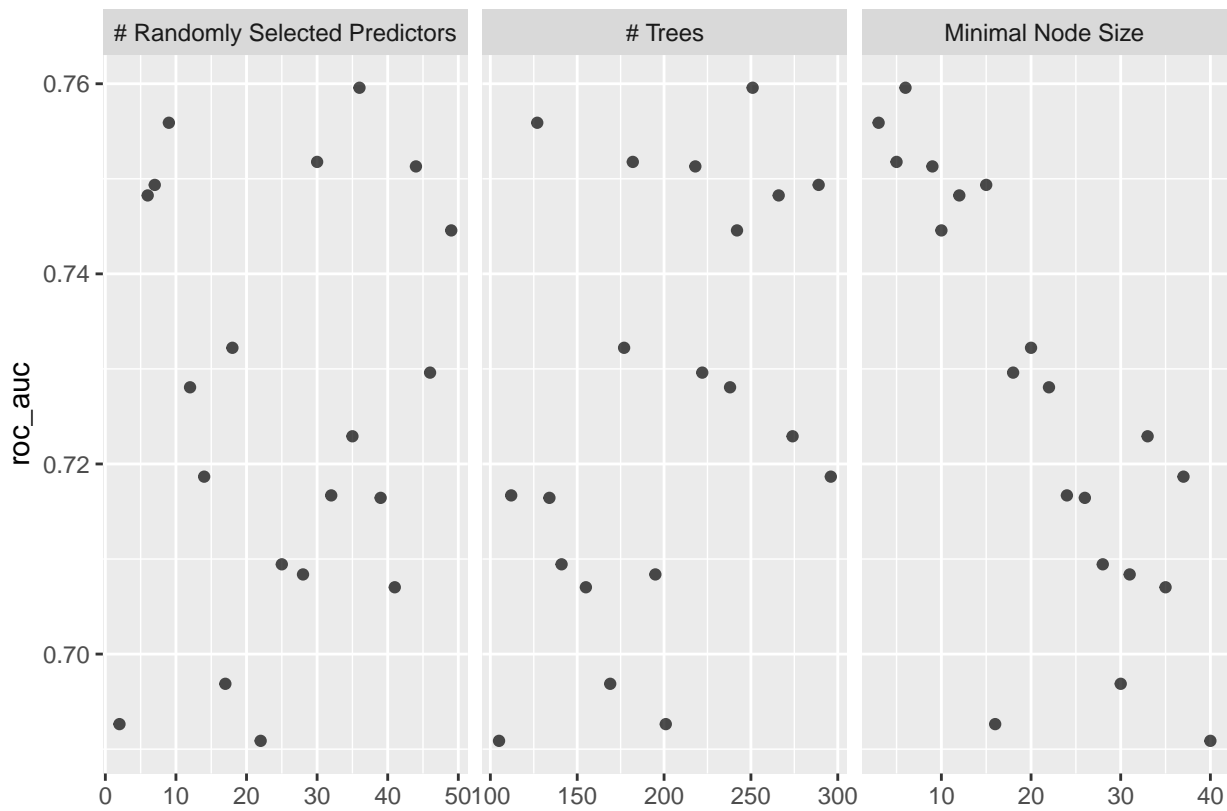
rf_tune <-
  rf_workflow %>%
  tune_grid(resamples = df_folds,
    grid = rf_grid)

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     22    105     40 accuracy binary    0.979     10 0.00124 Preprocessor1_Model01
## 2     22    105     40 roc_auc  binary    0.691     10 0.0165  Preprocessor1_Model01
## 3      7    289     15 accuracy binary    0.979     10 0.00124 Preprocessor1_Model02
## 4      7    289     15 roc_auc  binary    0.749     10 0.0127  Preprocessor1_Model02
## 5      2    201     16 accuracy binary    0.979     10 0.00124 Preprocessor1_Model03
## 6      2    201     16 roc_auc  binary    0.693     10 0.0133  Preprocessor1_Model03
## 7     25    141     28 accuracy binary    0.979     10 0.00124 Preprocessor1_Model04
## 8     25    141     28 roc_auc  binary    0.709     10 0.0163  Preprocessor1_Model04
## 9     46    222     18 accuracy binary    0.979     10 0.00124 Preprocessor1_Model05
## 10    46    222     18 roc_auc  binary    0.730     10 0.0119  Preprocessor1_Model05
## # ... with 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    36   251     6 roc_auc binary   0.760    10  0.0145 Preprocessor1_Model107
## 2     9   127     3 roc_auc binary   0.756    10  0.0123 Preprocessor1_Model112
## 3    30   182     5 roc_auc binary   0.752    10  0.0121 Preprocessor1_Model106
## 4    44   218     9 roc_auc binary   0.751    10  0.0127 Preprocessor1_Model116
## 5     7   289    15 roc_auc binary   0.749    10  0.0127 Preprocessor1_Model102
```

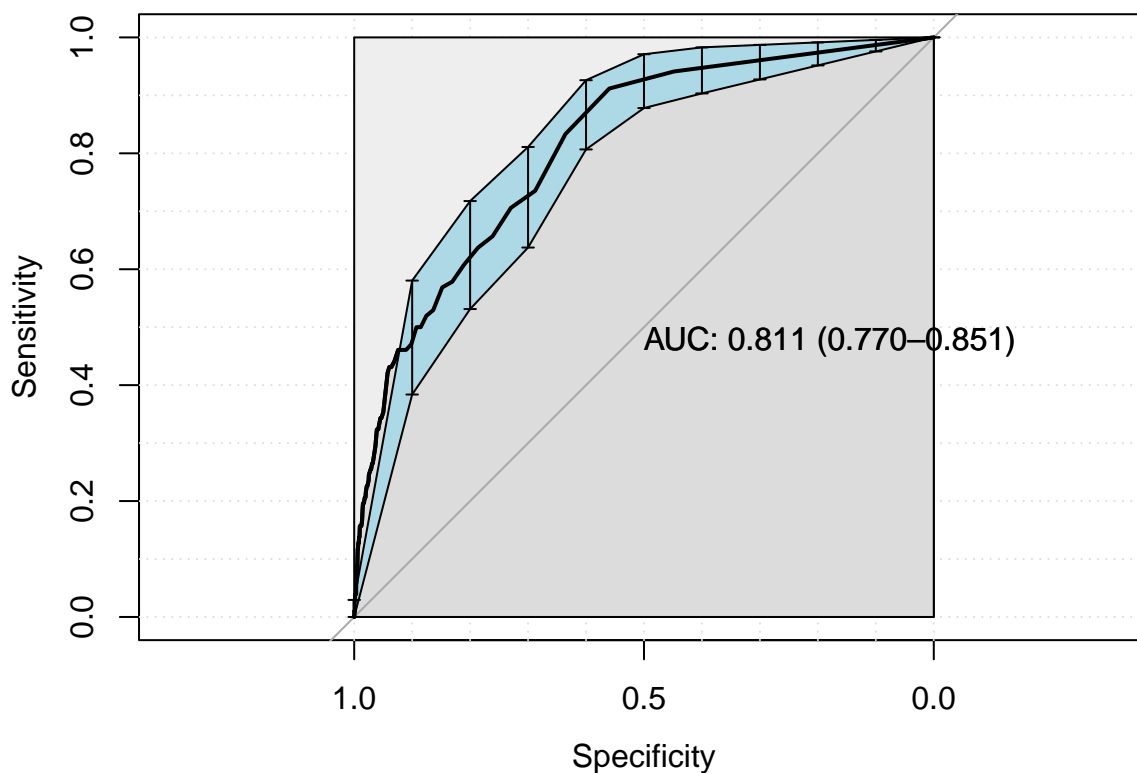
```
best_rf <- rf_tune %>%
  select_best("roc_auc")
```

```
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)
```

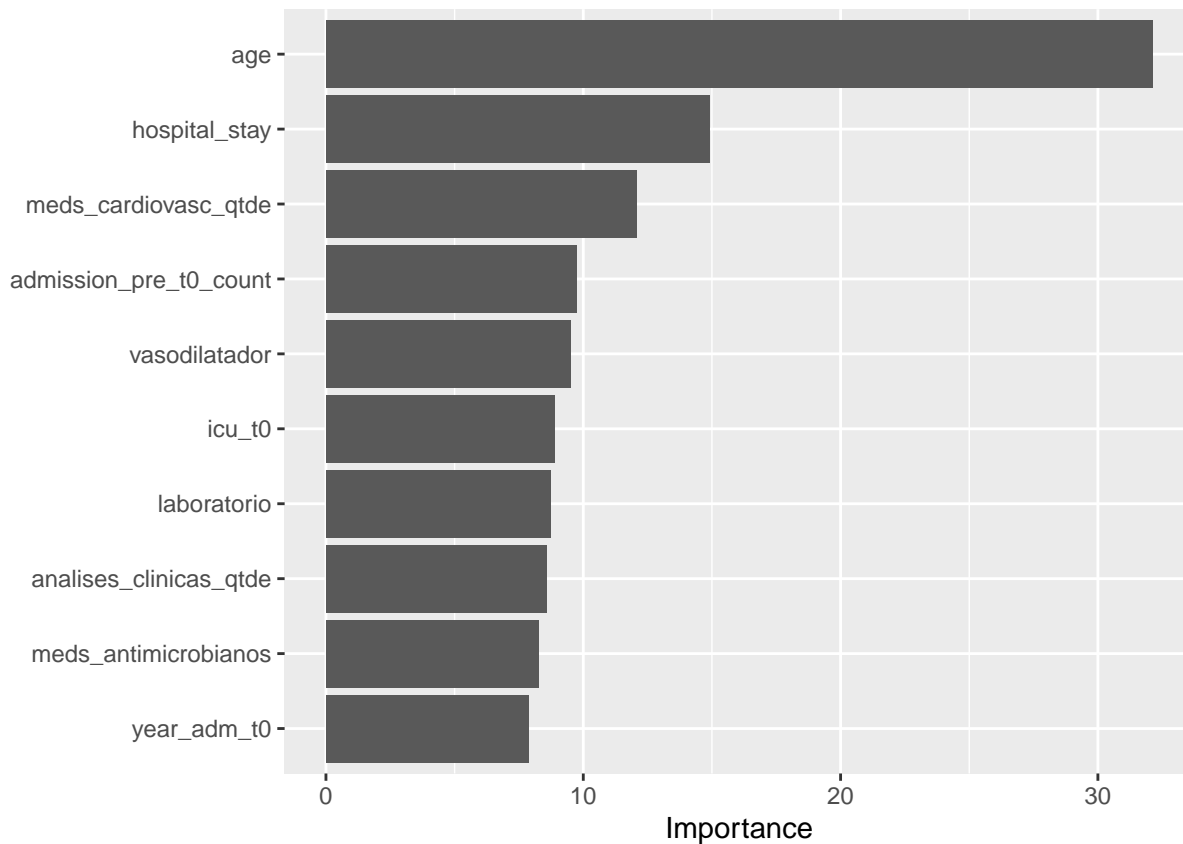
```
rf_auc <- validation(final_rf_fit, df_test)
```



```
## [1] "Optimal Threshold: 0.01"
## Confusion Matrix and Statistics
##
##      reference
## data    0    1
## 0 2591    9
## 1 2037   93
##
##              Accuracy : 0.5674
##              95% CI : (0.5532, 0.5816)
##      No Information Rate : 0.9784
##      P-Value [Acc > NIR] : 1
##
##              Kappa : 0.044
##
##  McNemar's Test P-Value : <2e-16
##
##              Sensitivity : 0.55985
##              Specificity : 0.91176
##      Pos Pred Value : 0.99654
##      Neg Pred Value : 0.04366
##      Prevalence : 0.97844
##      Detection Rate : 0.54778
##      Detection Prevalence : 0.54968
##      Balanced Accuracy : 0.73581
##
##      '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')
```



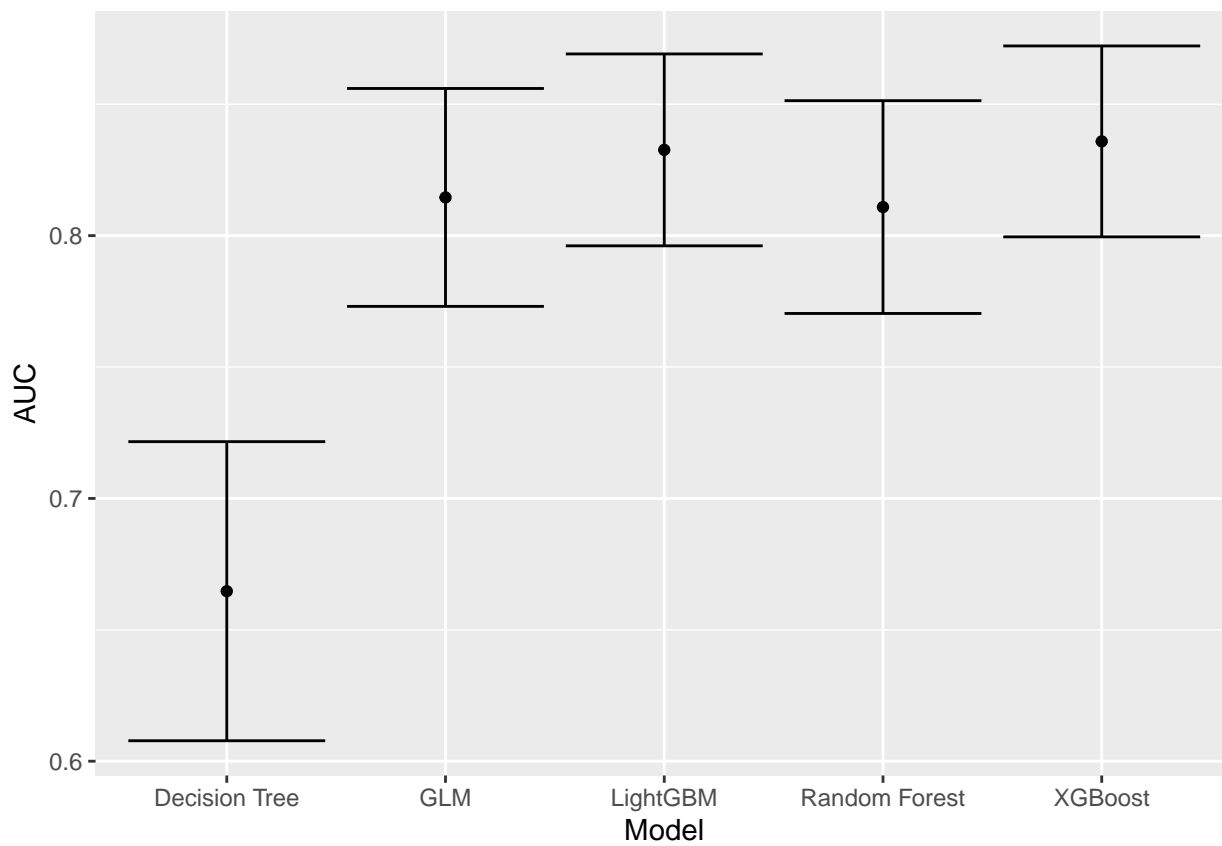
```
# extract_vip(final_rf_fit, pred_wrapper = predict,
#             reference_class = "1", use_matrix = FALSE,
#             method = 'permute')
```

Minutes to run: 59.227

## 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()
```



```
saveRDS(df_auc, sprintf("./auxiliar/model_selection/performance/%s.RData", outcome_column))
```

Minutes to run: 0.002