

# Model Selection

Eduardo Yuki Yada

## Imports

```
library(tidyverse)
library(yaml)
library(tidymodels)
library(usemodels)
library(vip)
```

## 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
```

## Filtering eligible patients

```
df = df %>%
  filter(disch_outcomes_t0 == 0)

df %>% dim
```

```
## [1] 15766 239
```

## Eligible features

```
eligible_columns = df_names %>%
  filter(momento.aquisicao == 'Admissão t0') %>%
  .$variable.name

exception_columns = c('death_intraop', 'death_intraop_1')

correlated_columns = c('year_procedure_1', # com year_adm_t0
  'age_surgery_1', # com age
  'admission_pre_t0_count', # com admission_t0
  'atb', # com meds_antimicrobianos
  'classe_meds_cardio_qtde', # com classe_meds_qtde
  'suporte_hemod' # com proced_invasivos_qtde
)

eligible_features = eligible_columns %>%
  base::intersect(c(columns_list$categorical_columns, columns_list$numerical_columns)) %>%
  setdiff(c(exception_columns, correlated_columns))
```

```

if (is.null(features_list)) {
  features = eligible_features
} else {
  features = base::intersect(eligible_features, features_list)
}

length(features)

## [1] 118
print(features)

## [1] "sex" "age" "race"
## [4] "education_level" "patient_state" "underlying_heart_disease"
## [7] "heart_disease" "nyha_basal" "hypertension"
## [10] "prior_mi" "heart_failure" "af"
## [13] "cardiac_arrest" "transplant" "valvopathy"
## [16] "endocardites" "diabetes" "renal_failure"
## [19] "hemodialysis" "stroke" "copd"
## [22] "cancer" "comorbidities_count" "procedure_type_1"
## [25] "reop_type_1" "procedure_type_new" "cied_final_1"
## [28] "cied_final_group_1" "admission_t0" "admission_pre_t0_180d"
## [31] "year_adm_t0" "icu_t0" "dialysis_t0"
## [34] "disch_outcomes_t0" "n_procedure_t0" "admission_t0_emergency"
## [37] "aco" "antiarritmico" "antihipertensivo"
## [40] "betabloqueador" "ieca_bra" "dva"
## [43] "digoxina" "estatina" "diuretico"
## [46] "vasodilatador" "insuf_cardiaca" "espirolactona"
## [49] "bloq_calcio" "trombolitico" "antiplaquetario_vo"
## [52] "antiplaquetario_ev" "insulina" "hipoglicemiante"
## [55] "hormonio_tireoidiano" "broncodilatador" "anticonvulsivante"
## [58] "psicofarmacos" "antifungico" "antiviral"
## [61] "antiretroviral" "classe_meds_qtde" "meds_cardiovasc_qtde"
## [64] "meds_antimicrobianos" "vni" "ventilacao_mecanica"
## [67] "cec" "cir_cardiovascular" "transplante_cardiaco"
## [70] "cir_toracica" "outros_proced_cirurgicos" "traqueostomia"
## [73] "icp" "intervencao_cv" "stent"
## [76] "angioplastia" "cateterismo" "eletrofisiologia"
## [79] "cateter_venoso_central" "drenagem_torax" "proced_invasivos_qtde"
## [82] "cve_desf" "transfusao" "interconsulta"
## [85] "equipe_multiprof" "ecg" "holter"
## [88] "teste_esforco" "espiro_ergoespiro" "tilt_teste"
## [91] "polissonografia" "metodos_graficos_qtde" "laboratorio"
## [94] "cultura" "analises_clinicas_qtde" "citologia"
## [97] "biopsia" "histopatologia_qtde" "angio_rm"
## [100] "angio_tc" "angiografia" "aortografia"
## [103] "arteriografia" "cavografia" "cintilografia"
## [106] "ecocardiograma" "endoscopia" "flebografia"
## [109] "pet_ct" "ultrassom" "tomografia"
## [112] "radiografia" "ressonancia" "exames_imagem_qtde"
## [115] "dieta_enteral" "dieta_parenteral" "bic"
## [118] "mpp"

```

## Train test split (70%/30%)

```

set.seed(42)

df[columns_list$outcome_columns] <- lapply(df[columns_list$outcome_columns], factor)
df <- mutate(df, across(where(is.character), as.factor))

df_split <- initial_split(df %>% dplyr::select(all_of(c(features, outcome_column))),

```

```

      prop = .7, strata = all_of(outcome_column))
df_train <- training(df_split)
df_test <- testing(df_split)

dim(df_train)[1] / dim(df)[1]

## [1] 0.6999873
dim(df_test)[1] / dim(df)[1]

## [1] 0.3000127

```

## Global parameters

```

k = 4 # Number of folds for cross validation
grid_size = 10 # Number of parameter combination to tune on each model

set.seed(234)
df_folds <- vfold_cv(df_train, v = k,
  strata = all_of(outcome_column))

```

## Functions

```

validation = function(model_fit, new_data) {
  library(pROC)
  library(caret)

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

  pROC_obj <- roc(
    new_data[[outcome_column]],
    test_predictions_prob,
    smoothed = TRUE,
    # arguments for ci
    ci = TRUE,
    ci.alpha = 0.9,
    stratified = FALSE,
    # arguments for plot
    plot = TRUE,
    auc.polygon = TRUE,
    max.auc.polygon = TRUE,
    grid = TRUE,
    print.auc = TRUE,
    show.thres = TRUE
  )

  sens.ci <- ci.se(pROC_obj)
  plot(sens.ci, type = "shape", col = "lightblue")
  plot(sens.ci, type = "bars")

  test_predictions_class <-
    predict(model_fit, new_data = new_data, type = "class") %>%
    rename_at(vars(starts_with(".pred_")), ~ str_remove(., ".pred_")) %>%
    .$class

  conf_matrix = table(test_predictions_class, new_data[[outcome_column]])
}

```

```

confusionMatrix(conf_matrix) %>% print

return(pROC_obj)
}

```

## Boosted Tree (XGBoost)

```

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

xgboost_spec <- boost_tree(
  trees = 100,
  tree_depth = tune(),
  min_n = tune(),
  loss_reduction = tune(),
  sample_size = tune(),
  mtry = tune(),
  learn_rate = tune()
) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

xgboost_grid <- grid_latin_hypercube(
  tree_depth(),
  min_n(),
  loss_reduction(),
  sample_size = sample_prop(),
  finalize(mtry(), df_train),
  learn_rate(),
  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 12
##   mtry min_n tree_depth learn_rate loss_~1 sampl~2 .metric .esti~3 mean      n std_err .config
##   <int> <int>      <int>      <dbl>    <dbl>    <dbl> <chr>    <chr>    <dbl> <int>    <dbl> <chr>
## 1     89     24         15 0.0261    1.11e-5   0.642 roc_auc binary  0.711     4  0.0154 Prepro~
## 2     30      5         14 0.0000266 5.76e-7   0.580 roc_auc binary  0.709     4  0.0140 Prepro~
## 3     71     38         10 0.0000111 1.17e-9   0.877 roc_auc binary  0.707     4  0.0142 Prepro~
## 4     98     30          7 0.00878    2.70e-2   0.343 roc_auc binary  0.705     4  0.0152 Prepro~
## 5     22     18          6 0.00000263 1.34e+0   0.376 roc_auc binary  0.692     4  0.0137 Prepro~
## # ... with abbreviated variable names 1: loss_reduction, 2: sample_size, 3: .estimator

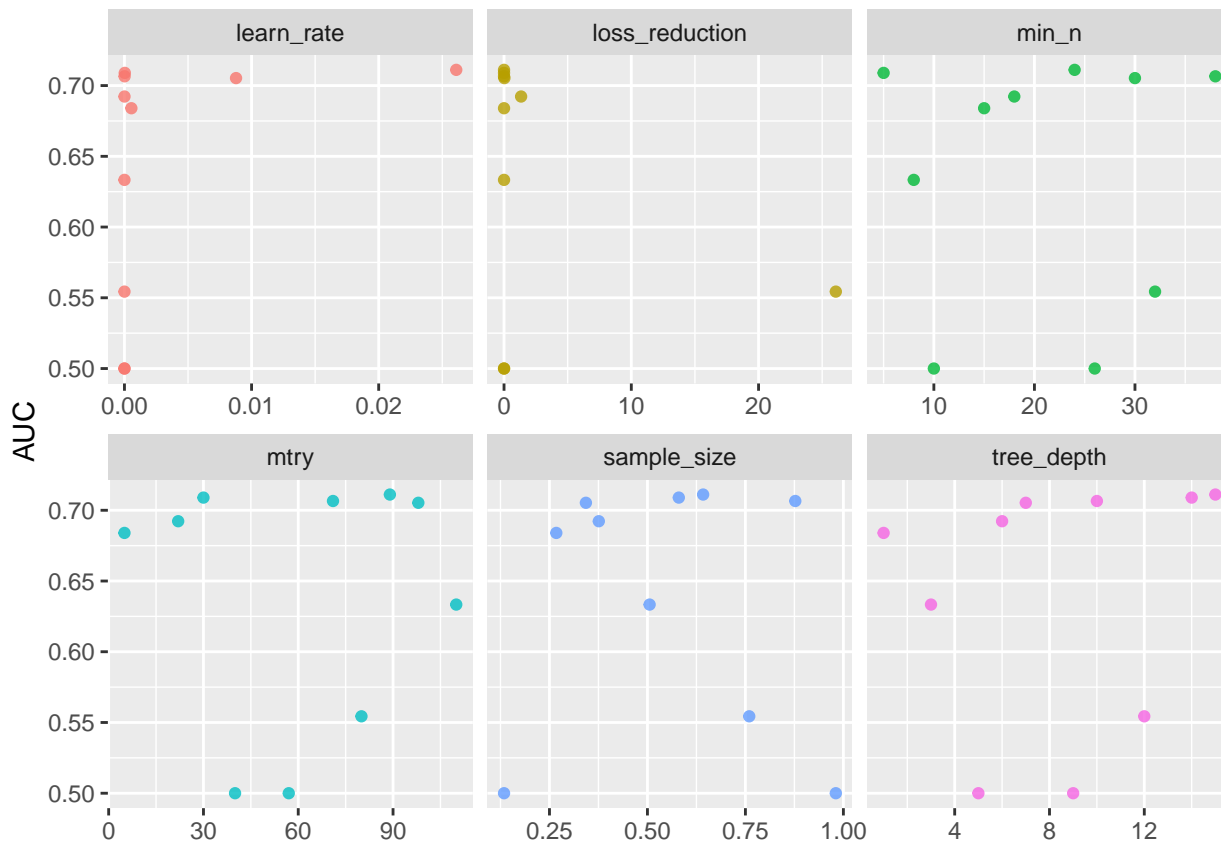
```

```

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

xgboost_tune %>%
  collect_metrics() %>%
  filter(.metric == "roc_auc") %>%
  select(mean, mtry:sample_size) %>%
  pivot_longer(mtry:sample_size,
               values_to = "value",
               names_to = "parameter"
  ) %>%
  ggplot(aes(value, mean, color = parameter)) +
  geom_point(alpha = 0.8, show.legend = FALSE) +
  facet_wrap(~parameter, scales = "free_x") +
  labs(x = NULL, y = "AUC")

```



```

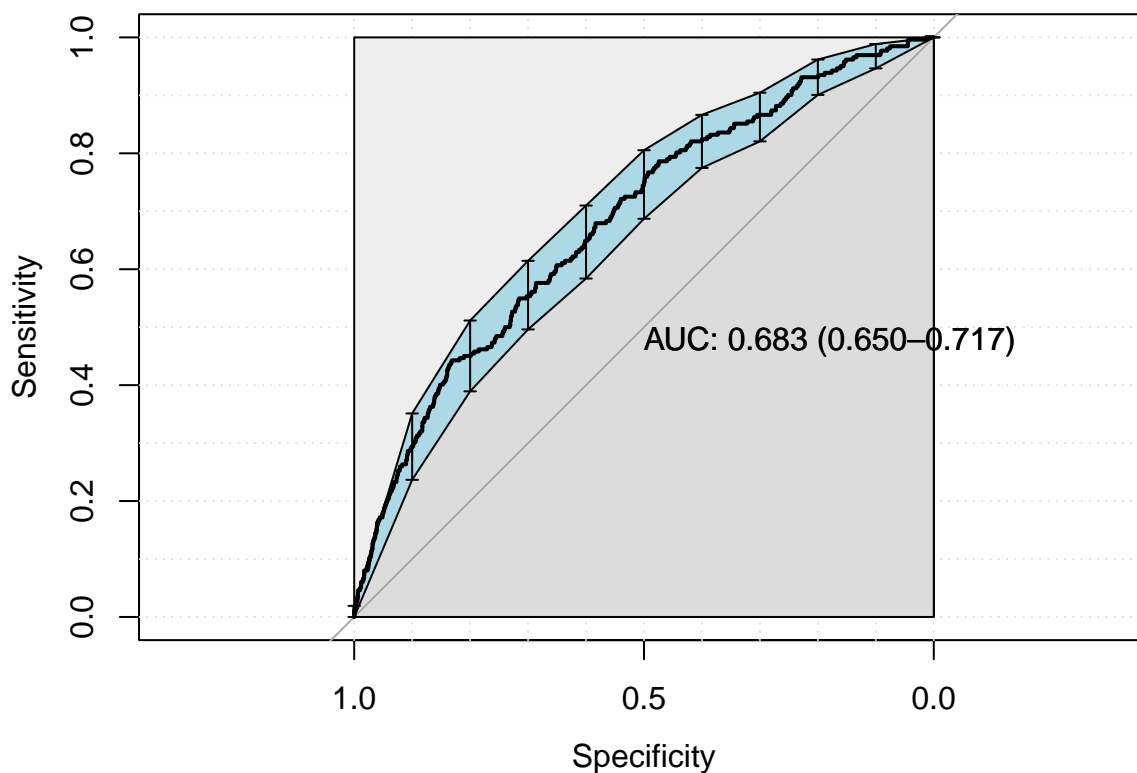
final_xgboost_workflow <-
  xgboost_workflow %>%
  finalize_workflow(best_xgboost)

last_xgboost_fit <-
  final_xgboost_workflow %>%
  last_fit(df_split)

final_xgboost_fit <- extract_workflow(last_xgboost_fit)

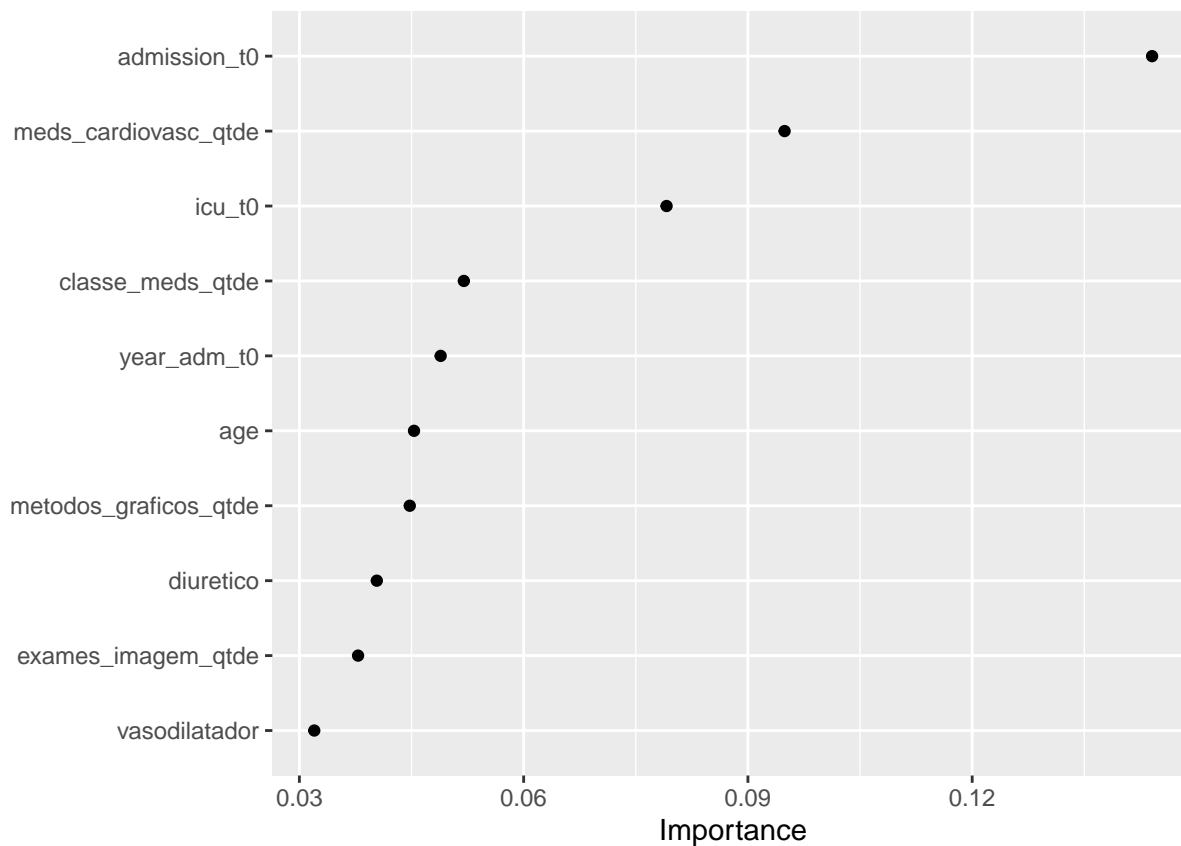
xgboost_auc = validation(final_xgboost_fit, df_test)

```



```
## Confusion Matrix and Statistics
##
##
## test_predictions_class    0    1
##                0 4468  262
##                1    0    0
##
##          Accuracy : 0.9446
##          95% CI   : (0.9377, 0.951)
##    No Information Rate : 0.9446
##    P-Value [Acc > NIR] : 0.5164
##
##          Kappa : 0
##
## McNemar's Test P-Value : <2e-16
##
##          Sensitivity : 1.0000
##          Specificity : 0.0000
##    Pos Pred Value : 0.9446
##    Neg Pred Value :    NaN
##    Prevalence : 0.9446
##    Detection Rate : 0.9446
##    Detection Prevalence : 1.0000
##    Balanced Accuracy : 0.5000
##
##    'Positive' Class : 0
##
```

```
final_xgboost_fit %>%
  fit(data = df_train) %>%
  extract_fit_parsnip() %>%
  vip(geom = "point")
```



## GLM

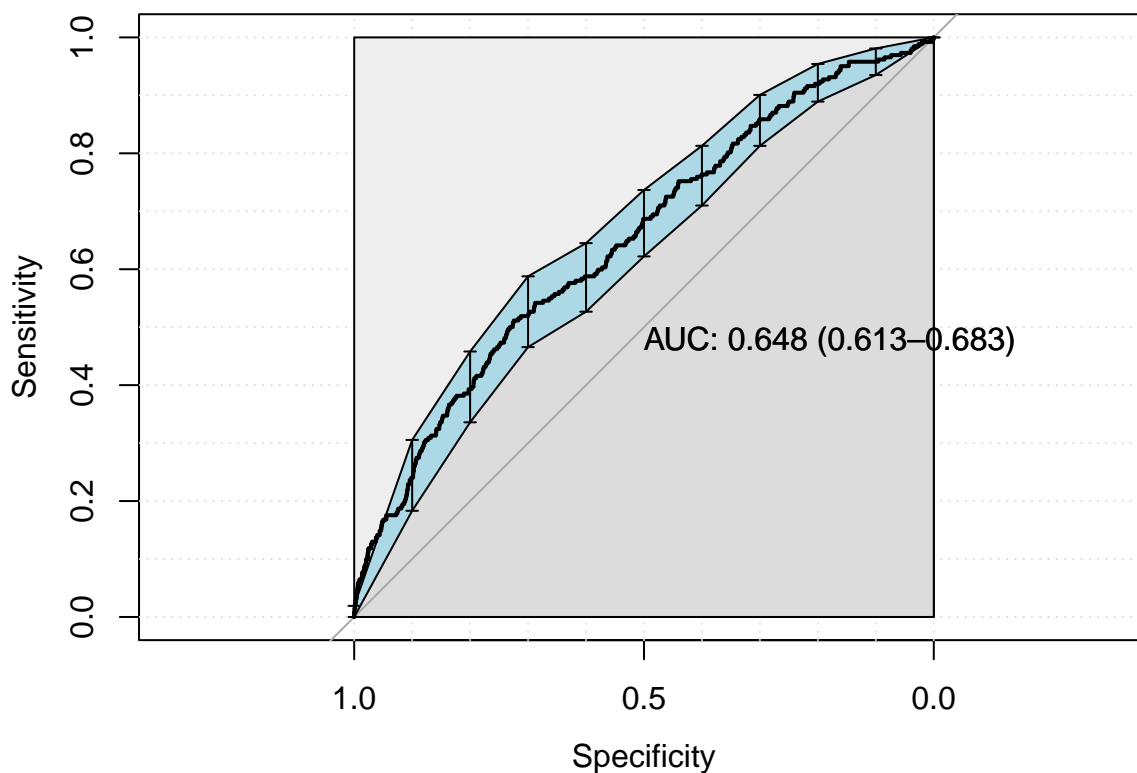
```
glmnet_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_other(all_nominal_predictors(), threshold = 0.05, other=".merged") %>%
  step_dummy(all_nominal_predictors(),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

glm_auc = validation(glm_fit, df_test)
```



```
## Confusion Matrix and Statistics
##
##
## test_predictions_class    0    1
##                0 4465  260
##                1    3    2
##
##              Accuracy : 0.9444
##              95% CI : (0.9375, 0.9508)
##    No Information Rate : 0.9446
##    P-Value [Acc > NIR] : 0.5417
##
##              Kappa : 0.0129
##
##  McNemar's Test P-Value : <2e-16
##
##              Sensitivity : 0.999329
##              Specificity : 0.007634
##    Pos Pred Value : 0.944974
##    Neg Pred Value : 0.400000
##    Prevalence : 0.944609
##    Detection Rate : 0.943975
##    Detection Prevalence : 0.998943
##    Balanced Accuracy : 0.503481
##
##    'Positive' Class : 0
##
```

## 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(), one_hot = TRUE) %>%
step_zv(all_predictors())

tree_spec <-
  decision_tree(cost_complexity = tune(),
                tree_depth = tune()) %>%
  set_mode("classification") %>%
  set_engine("rpart")

tree_grid <- grid_latin_hypercube(cost_complexity(),
                                  tree_depth(),
                                  size = grid_size)

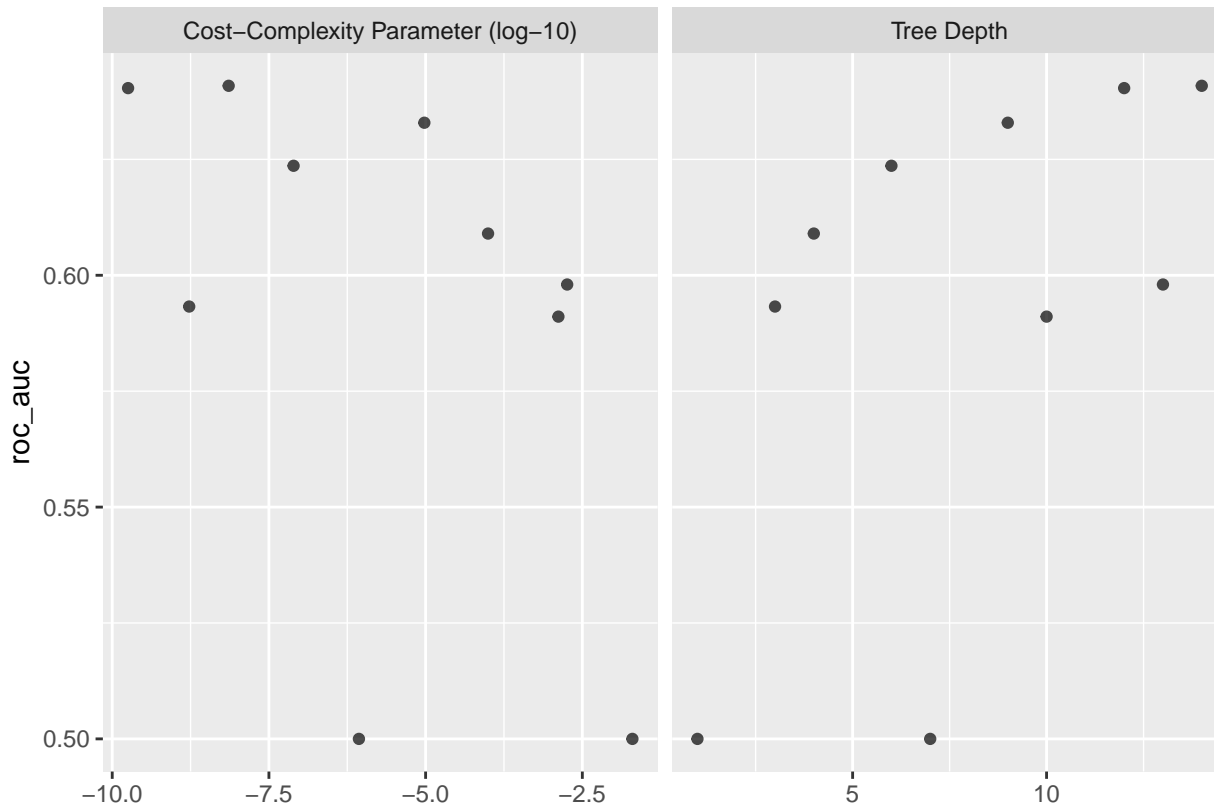
tree_workflow <-
  workflow() %>%
  add_recipe(tree_recipe) %>%
  add_model(tree_spec)

tree_tune <-
  tree_workflow %>%
  tune_grid(resamples = df_folds,
            grid = tree_grid)

tree_tune %>%
  collect_metrics()

autoplot(tree_tune, metric = "roc_auc")

```



```

tree_tune %>%
  show_best("roc_auc")

```

```

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

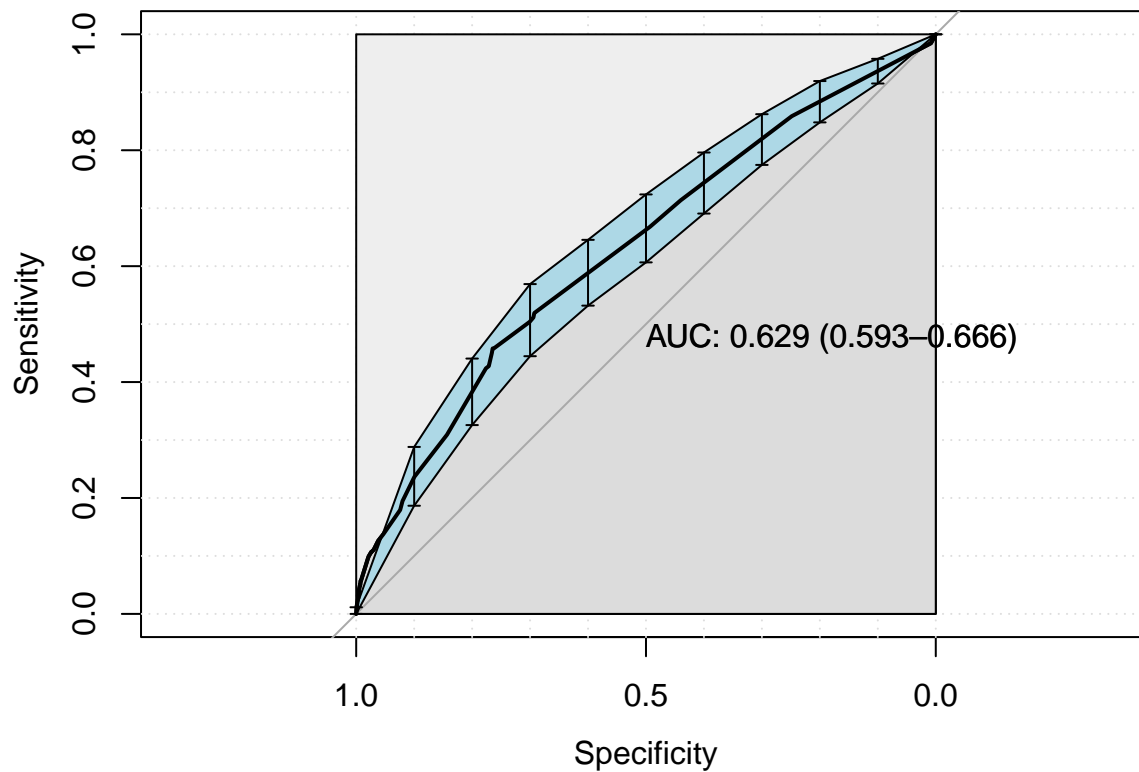
final_tree_workflow <-
  tree_workflow %>%
  finalize_workflow(best_tree)

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)

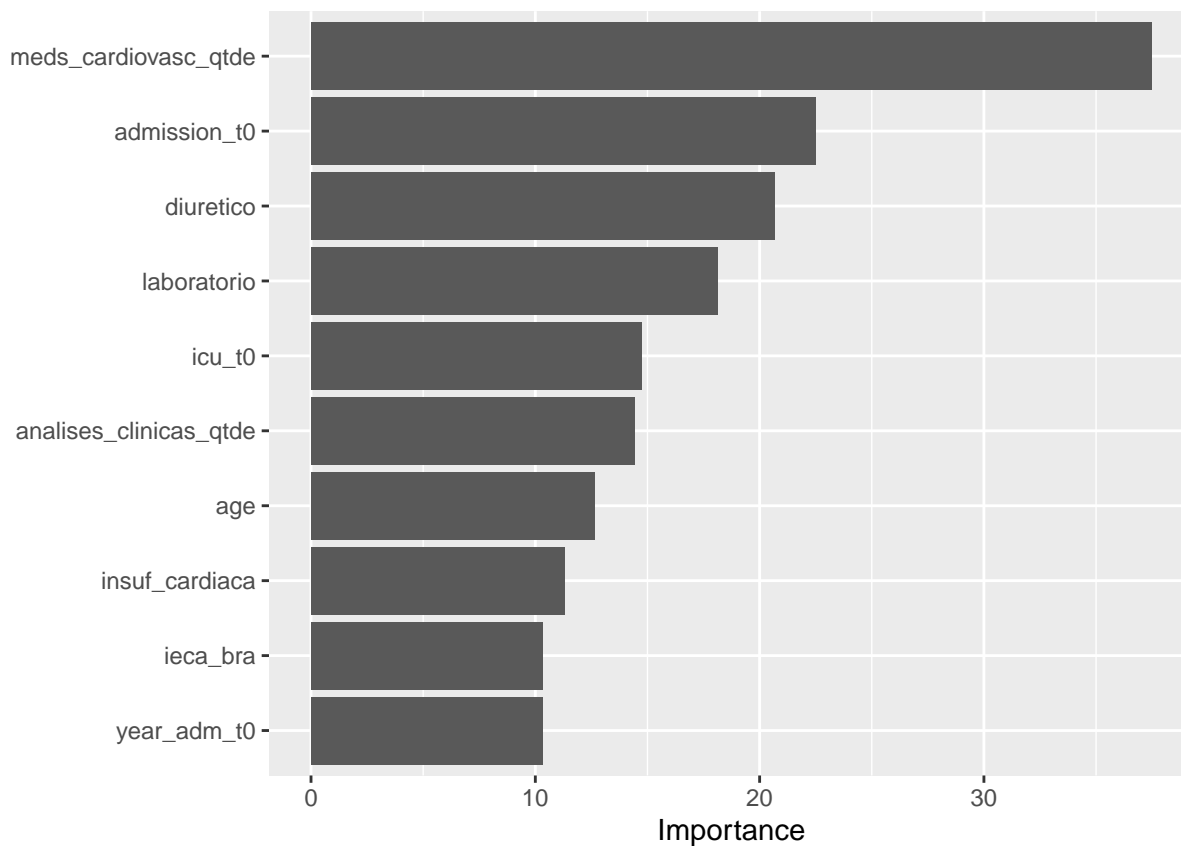
```



```

if (tree_auc$auc > 0.55){
  final_tree_fit %>%
    extract_fit_parsnip() %>%
    vip()
}

```



## 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(), one_hot = TRUE) %>%
  step_zv(all_predictors()) %>%
  step_impute_mean(all_numeric_predictors())

rf_spec <-
  rand_forest(mtry = tune(),
              trees = 100,
              min_n = tune()) %>%
  set_mode("classification") %>%
  set_engine("ranger")

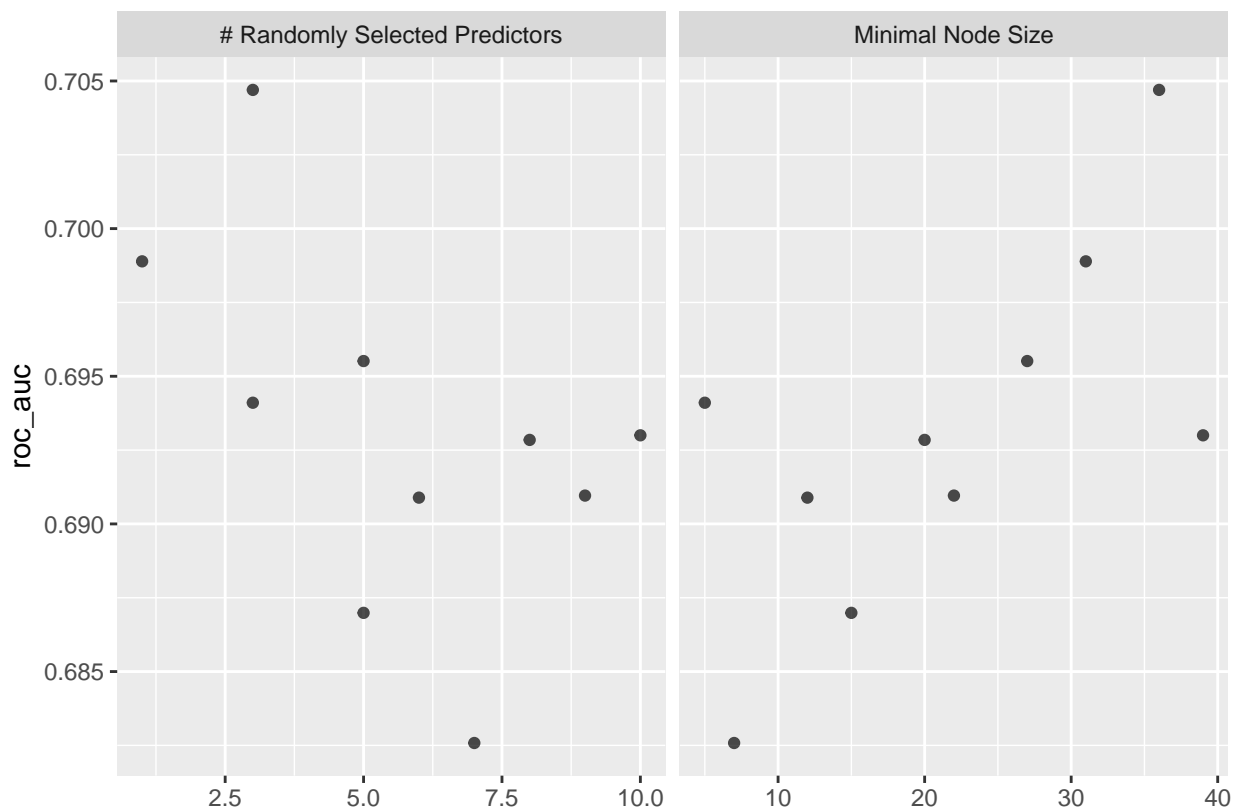
rf_grid <- grid_latin_hypercube(mtry(range = c(1, 10)),
                                min_n(),
                                size = grid_size)

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()

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



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

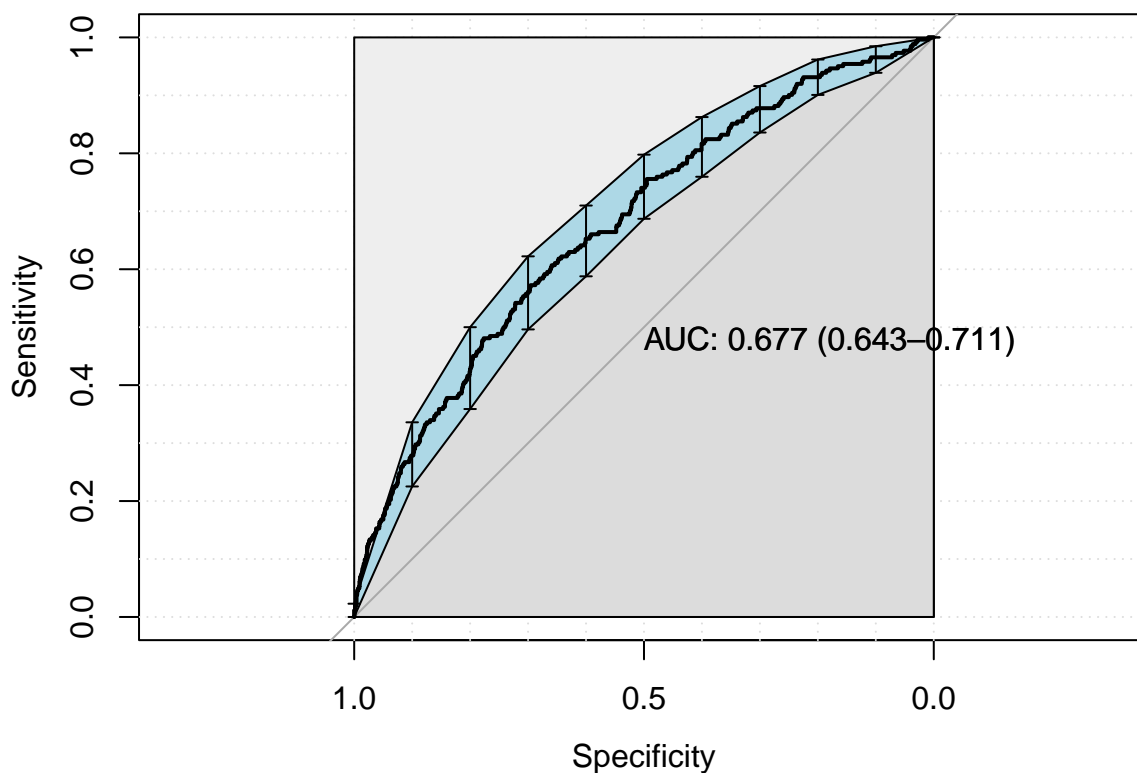
best_rf <- rf_tune %>%
  select_best("roc_auc")

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)
```



## KNN

```
# knn_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(), one_hot = TRUE) %>%
#   step_zv(all_predictors()) %>%
#   step_impute_mean(all_numeric_predictors())
#
# knn_spec <-
#   nearest_neighbor(neighbors = tune(),
#                     weight_func = tune(),
#                     dist_power = tune()) %>%
#   set_mode("classification") %>%
#   set_engine("kknn")
#
# knn_grid <- grid_latin_hypercube(neighbors(),
#                                   weight_func(),
#                                   dist_power(),
#                                   size = 5)
#
# knn_workflow <-
#   workflow() %>%
#   add_recipe(knn_recipe) %>%
#   add_model(knn_spec)
#
# knn_tune <-
#   knn_workflow %>%
#   tune_grid(resamples = df_folds,
#             grid = knn_grid)
```

```

#
# knn_tune %>%
#   collect_metrics()
#
# autoplot(knn_tune, metric = "roc_auc")
#
# knn_tune %>%
#   show_best("roc_auc")
#
# best_knn <- knn_tune %>%
#   select_best("roc_auc")
#
# final_knn_workflow <-
#   knn_workflow %>%
#   finalize_workflow(best_knn)
#
# last_knn_fit <-
#   final_knn_workflow %>%
#   last_fit(df_split)
#
# final_knn_fit <- extract_workflow(last_knn_fit)
#
# knn_auc = validation(final_knn_fit, df_test)

```

## SVM

```

svm_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(), one_hot = TRUE) %>%
  step_zv(all_predictors()) %>%
  step_impute_mean(all_numeric_predictors())

svm_spec <-
  svm_rbf(cost = tune(), rbf_sigma = tune()) %>%
  set_mode("classification") %>%
  set_engine("kernlab")

svm_grid <- grid_latin_hypercube(cost(),
                                rbf_sigma(),
                                size = grid_size)

svm_workflow <-
  workflow() %>%
  add_recipe(svm_recipe) %>%
  add_model(svm_spec)

svm_tune <-
  svm_workflow %>%
  tune_grid(resamples = df_folds,
            grid = 5)

svm_tune %>%
  collect_metrics()

```

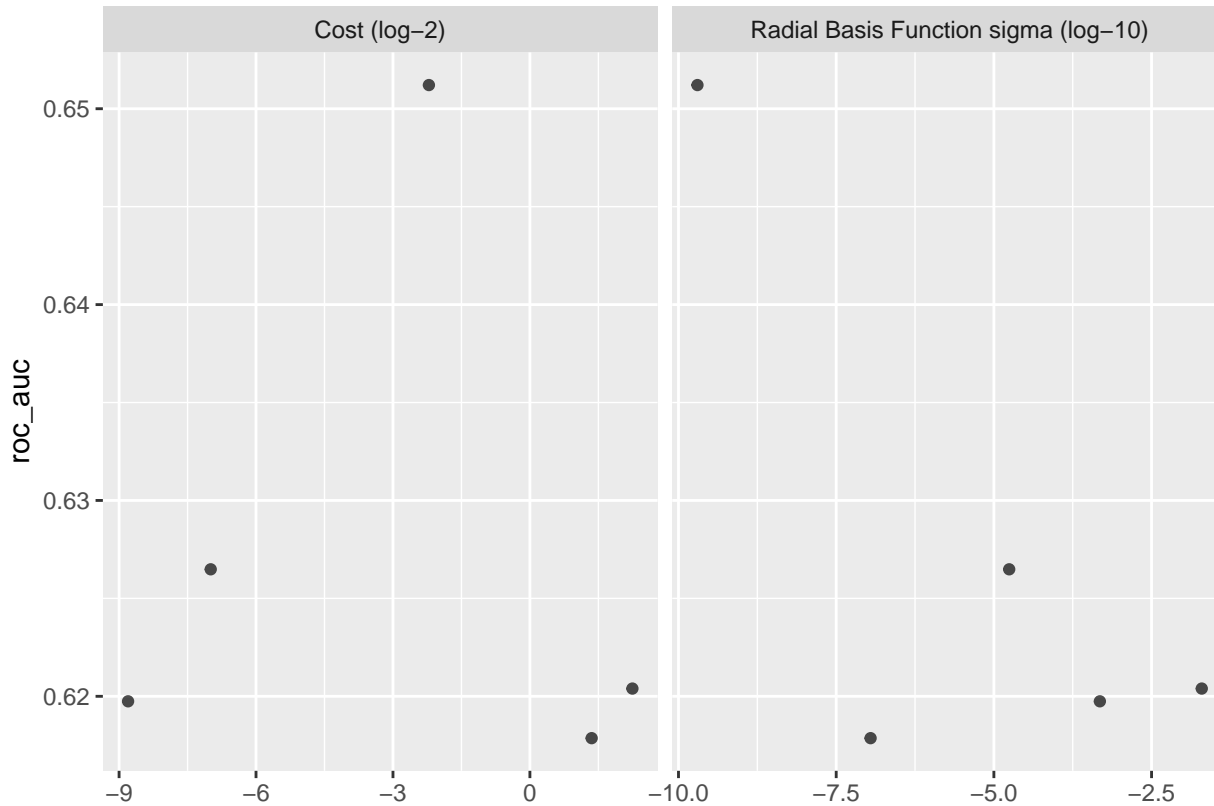
```

## # A tibble: 10 x 8
##       cost rbf_sigma .metric .estimator mean      n std_err .config
##       <dbl>    <dbl> <chr>    <chr>    <dbl> <int>    <dbl> <chr>

```

```
## 1 0.00224 4.79e- 4 accuracy binary 0.942 4 0.00123 Preprocessor1_Model1
## 2 0.00224 4.79e- 4 roc_auc binary 0.620 4 0.0143 Preprocessor1_Model1
## 3 2.55 1.11e- 7 accuracy binary 0.942 4 0.00123 Preprocessor1_Model2
## 4 2.55 1.11e- 7 roc_auc binary 0.618 4 0.00412 Preprocessor1_Model2
## 5 4.74 1.98e- 2 accuracy binary 0.940 4 0.000863 Preprocessor1_Model3
## 6 4.74 1.98e- 2 roc_auc binary 0.620 4 0.00901 Preprocessor1_Model3
## 7 0.00785 1.75e- 5 accuracy binary 0.942 4 0.00123 Preprocessor1_Model4
## 8 0.00785 1.75e- 5 roc_auc binary 0.626 4 0.00311 Preprocessor1_Model4
## 9 0.216 2.00e-10 accuracy binary 0.942 4 0.00123 Preprocessor1_Model5
## 10 0.216 2.00e-10 roc_auc binary 0.651 4 0.0204 Preprocessor1_Model5
```

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



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

```
## # A tibble: 5 x 8
##   cost rbf_sigma .metric .estimator mean n std_err .config
##   <dbl> <dbl> <chr> <chr> <dbl> <int> <dbl> <chr>
## 1 0.216 2.00e-10 roc_auc binary 0.651 4 0.0204 Preprocessor1_Model5
## 2 0.00785 1.75e- 5 roc_auc binary 0.626 4 0.00311 Preprocessor1_Model4
## 3 4.74 1.98e- 2 roc_auc binary 0.620 4 0.00901 Preprocessor1_Model3
## 4 0.00224 4.79e- 4 roc_auc binary 0.620 4 0.0143 Preprocessor1_Model1
## 5 2.55 1.11e- 7 roc_auc binary 0.618 4 0.00412 Preprocessor1_Model2
```

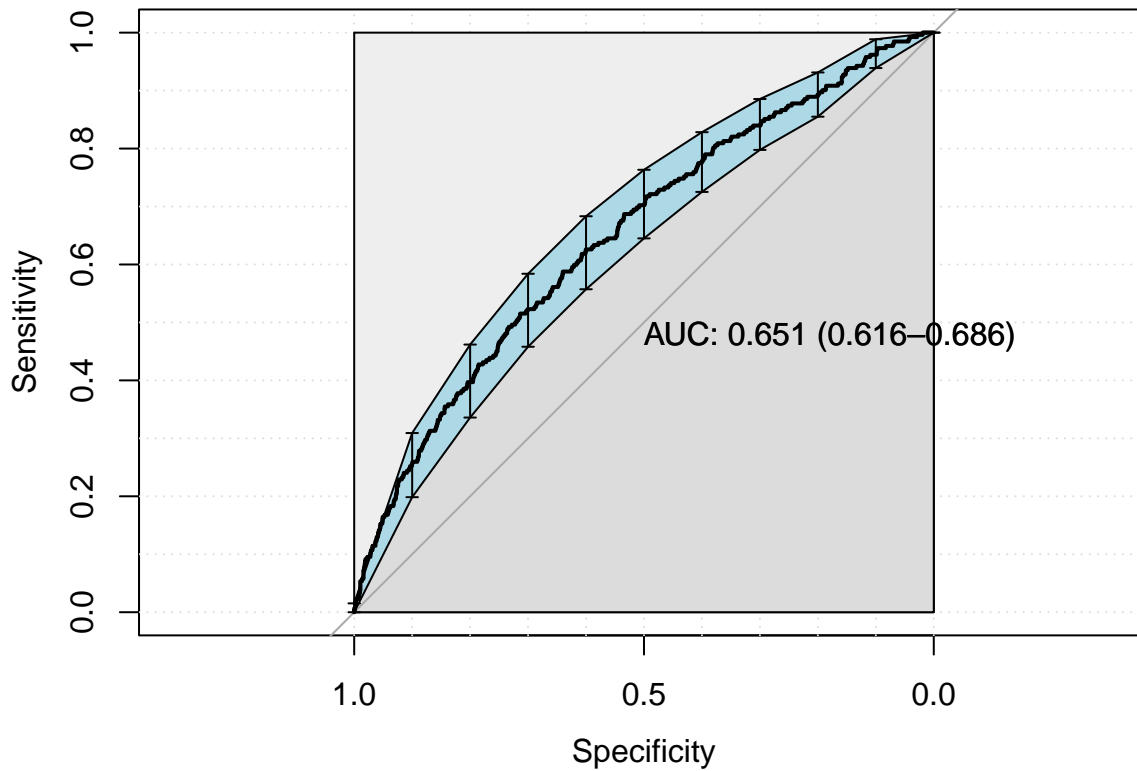
```
best_svm <- svm_tune %>%
  select_best("roc_auc")
```

```
final_svm_workflow <-
  svm_workflow %>%
  finalize_workflow(best_svm)
```

```
last_svm_fit <-
  final_svm_workflow %>%
  last_fit(df_split)
```

```
final_svm_fit <- extract_workflow(last_svm_fit)

svm_auc = validation(final_svm_fit, df_test)
```

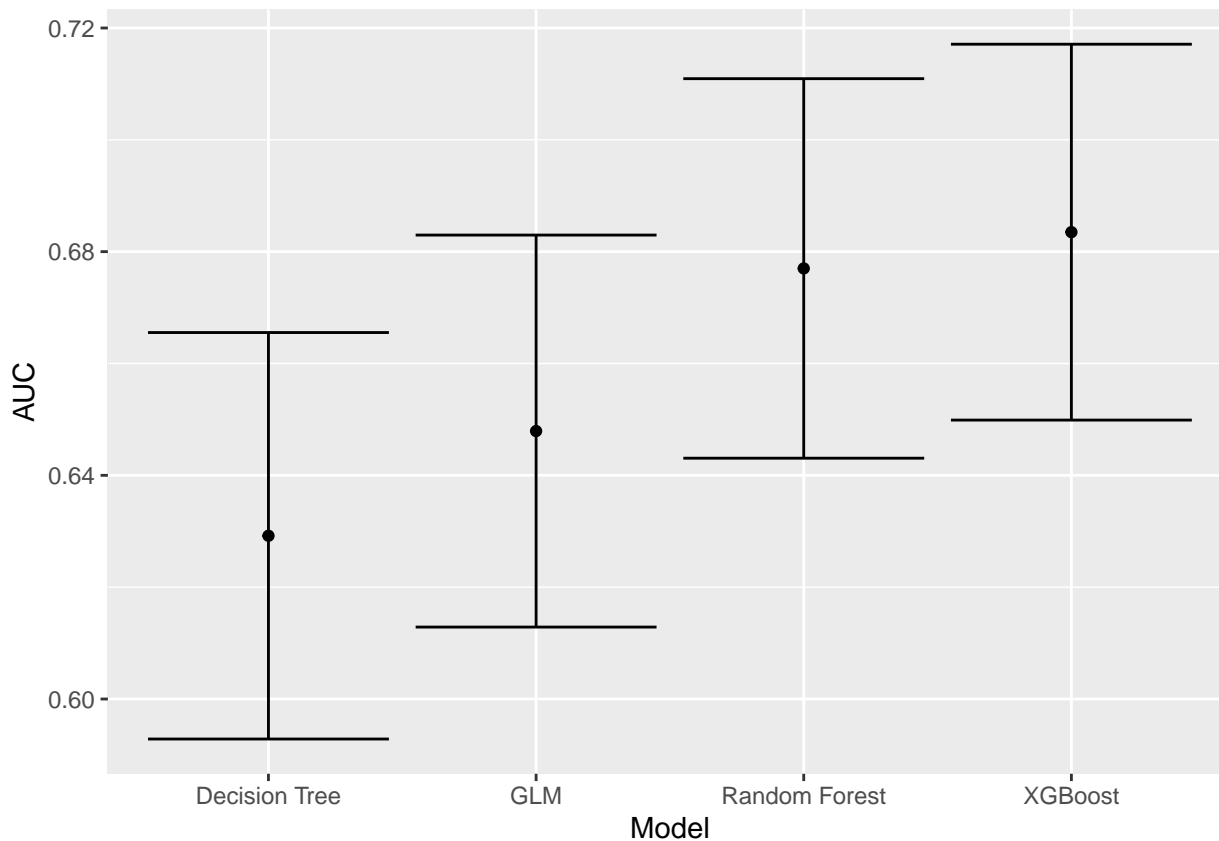


```
## Confusion Matrix and Statistics
##
##
## test_predictions_class    0    1
##          0 4468  262
##          1    0    0
##
##          Accuracy : 0.9446
##          95% CI : (0.9377, 0.951)
##    No Information Rate : 0.9446
##    P-Value [Acc > NIR] : 0.5164
##
##          Kappa : 0
##
## McNemar's Test P-Value : <2e-16
##
##          Sensitivity : 1.0000
##          Specificity : 0.0000
##          Pos Pred Value : 0.9446
##          Neg Pred Value :   NaN
##          Prevalence : 0.9446
##          Detection Rate : 0.9446
##    Detection Prevalence : 1.0000
##          Balanced Accuracy : 0.5000
##
##          'Positive' Class : 0
##
```



## Models Comparison

```
df_auc <- tibble::tribble(  
  ~Model, ~`AUC`, ~`Lower Limit`, ~`Upper Limit`,  
  'XGBoost', as.numeric(xgboost_auc$auc), xgboost_auc$ci[1], xgboost_auc$ci[3],  
  'GLM', as.numeric(glm_auc$auc), glm_auc$ci[1], glm_auc$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)  
  
df_auc %>%  
  ggplot(aes(x = Model, y = AUC, ymin = `Lower Limit`, ymax = `Upper Limit`)) +  
    geom_point() +  
    geom_errorbar()
```



```
saveRDS(df_auc, sprintf("../EDA/auxiliar/performance/%s_auc_result.RData", outcome_column))
```