ZK Models

Zachary Katz (zak2132)

5/12/2022

Contents

Introduction

```
# Read in all data
# source: https://raw.githubusercontent.com/TarekDib03/Analytics/master/Week3%20-%20Logistic%20Regressi
all_df = read_csv("FHS.csv")

# Factor labels for categorical variables and other recoding
cleaned_df = all_df %>%
  mutate(male = factor(male),
         current_smoker = factor(current_smoker),
         bp_meds = factor(bp_meds),
         prevalent_stroke = factor(prevalent_stroke),
         prevalent_hyp = factor(prevalent_hyp),
         diabetes = factor(diabetes),
         ten_year_chd = factor(ten_year_chd))  %>%
  mutate(ten_year_chd = ifelse(ten_year_chd == "1", "CHD_present","CHD_absent") %>%
           fct_relevel("CHD_present", "CHD_absent")) %>%
  dplyr::rename(sex = male) %>%
  mutate(sex = ifelse(sex == "1", "male","female") %>%
           fct_relevel("male", "female")) %>%
  mutate(
    education = case_when(
      education == "1" ~ "some_HS",
      education == "2" ~ "HS_grad",
      education == "3" ~ "some_college",
      education == "4" ~ "college_grad"
    ),
    current_smoker = recode(
      current_smoker,
      "1" = "yes",
      "0" = "no"
```

```
    ),
    bp_meds = recode(
      bp_meds,
      "1" = "yes",
      "0" = "no"
    ),
    prevalent_stroke = recode(
      prevalent_stroke,
      "1" = "yes",
      "0" = "no"
    ),
    prevalent_hyp = recode(
      prevalent_hyp,
      "1" = "yes",
      "0" = "no"
    ),
    diabetes = recode(
      diabetes,
      "1" = "yes",
      "0" = "no"
    ),
    education = factor(education, levels = c("some_HS", "HS_grad", "some_college", "college_grad"))
  )
```

Models

```
set.seed(2022)

# Training/testing partition
index_train = createDataPartition(cleaned_df$ten_year_chd,
                                  p = 0.8,
                                  list = FALSE)

training_df = cleaned_df[index_train, ]
testing_df = cleaned_df[-index_train, ]

# Model matrices
x_train = model.matrix(ten_year_chd ~ ., training_df)[, -1] # Note that if a row has NAs, it is by defa
x_test = model.matrix(ten_year_chd ~ ., testing_df)[, -1]
y_train = training_df$ten_year_chd
y_test = testing_df$ten_year_chd

# Train control with 10-fold cross-validation repeated 5 times
ctrl = trainControl(method = "repeatedcv",
                    repeats = 5,
                    summaryFunction = twoClassSummary,
                    classProbs = TRUE)
```

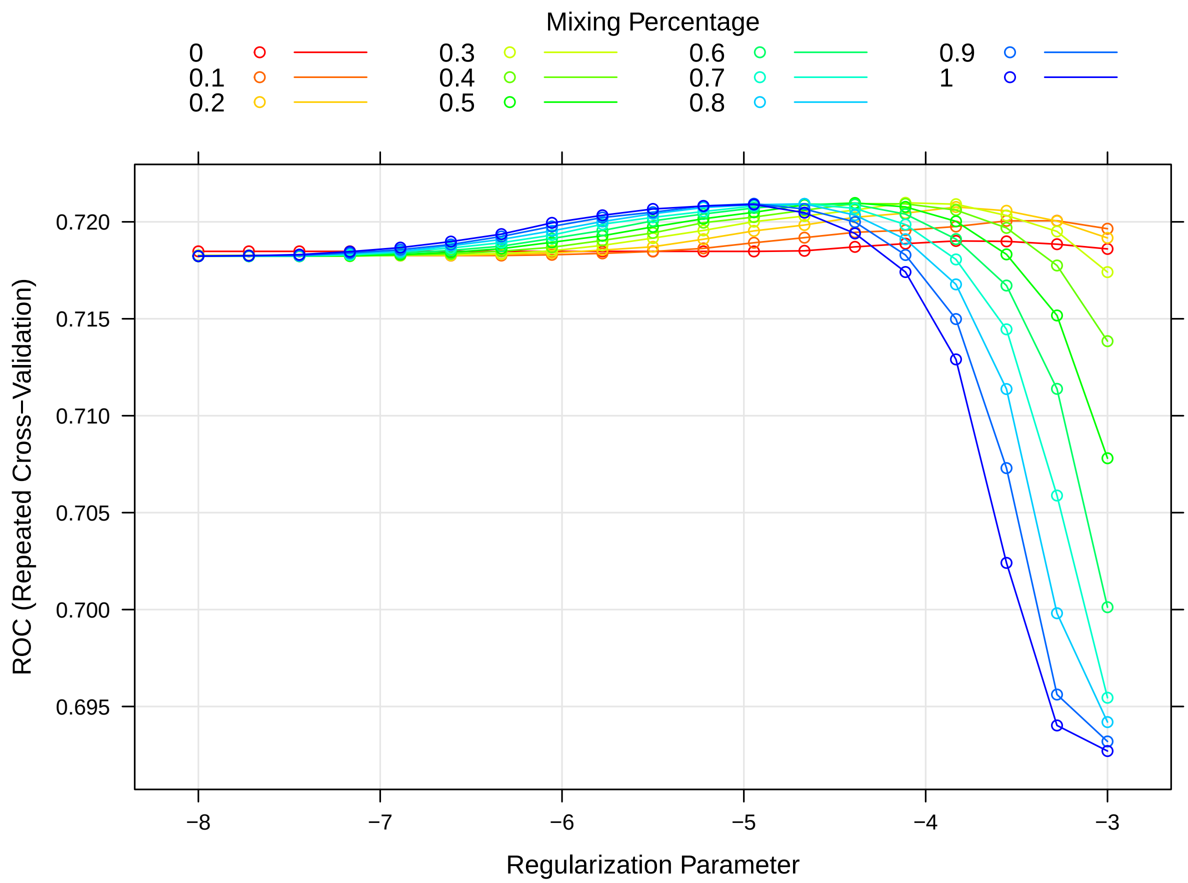
```
# Preprocessing and feature engineering with recipe (including imputation)
# Note: assuming data is MAR
```

```r
# recipe of preprocessing steps
preprocess_recipe = recipe(ten_year_chd ~ ., data = training_df) %>%
  step_impute_knn(all_predictors(), neighbors = 5) %>%  # KNN imputation based on 5 nearest neighbors
  step_BoxCox(all_numeric_predictors()) %>% # transform predictors
  step_center(all_numeric_predictors()) %>% # center and scale numeric predictors
  step_scale(all_numeric_predictors())
```

```r
# Penalized logistic regression with imputation in caret function directly (NOT recipes)
set.seed(2022)

glm_grid = expand.grid(alpha = seq(0, 1, length = 11),
                       lambda = exp(seq(-8, -3, length = 19)))

ctrl_glmnet = trainControl(method = "repeatedcv",
                    repeats = 5,
                    summaryFunction = twoClassSummary,
                    classProbs = TRUE,
                    preProcOptions = list(k = 5))

logit_next = train(ten_year_chd ~ .,
                   data = training_df,
                   na.action = na.pass,
                  method = "glmnet",
                  tuneGrid = glm_grid,
                  metric = "ROC",
                  trControl = ctrl_glmnet,
                  family = "binomial",
                  preProcess = c("knnImpute", "center", "scale", "BoxCox"))

# Optimal tuning parameters
# Alpha = 0.3, Lambda = 0.0164
logit_next$bestTune
```

```
##    alpha lambda
## 72   0.3 0.0164
```

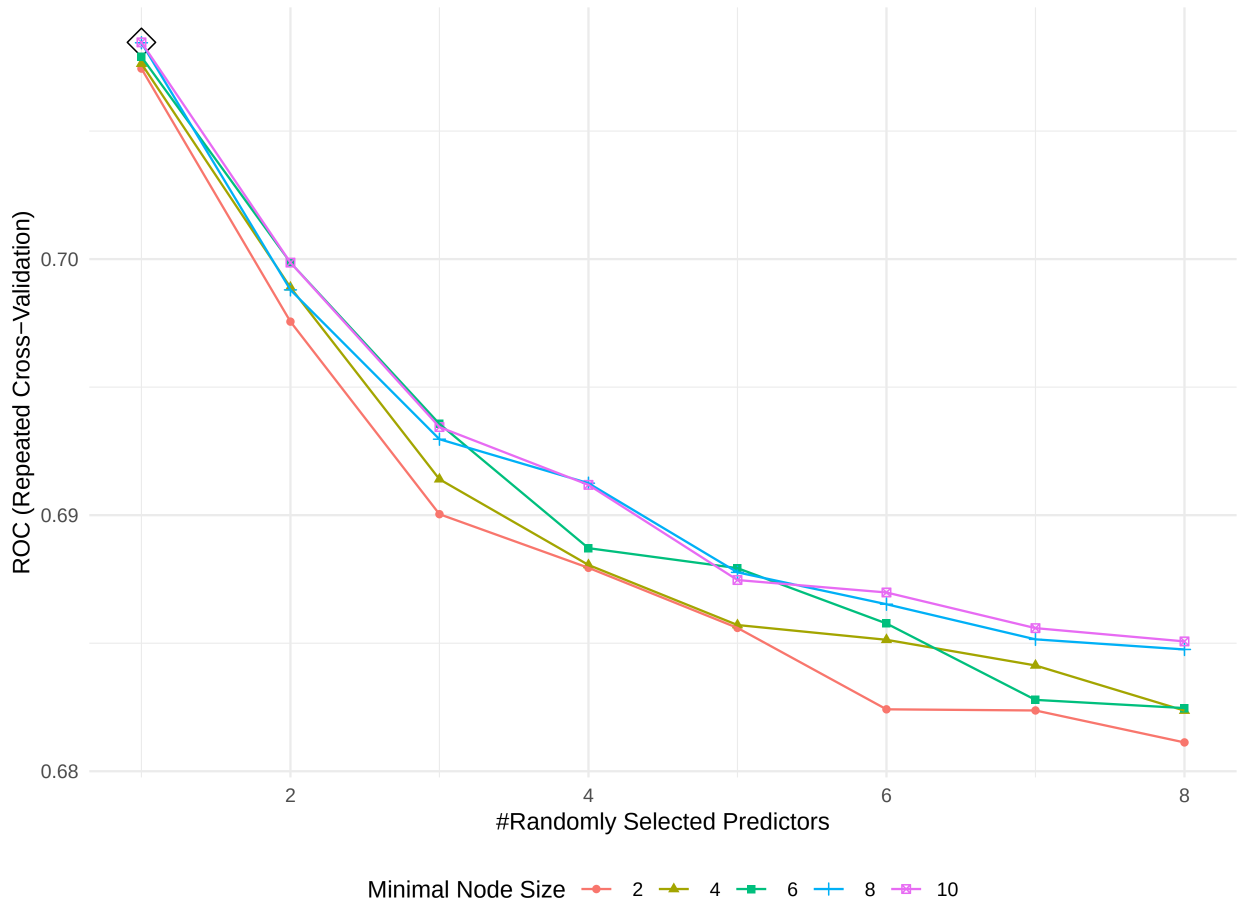
```r
# Plots of optimal tuning parameters
myCol = rainbow(15)
myPar = list(superpose.symbol = list(col = myCol),
             superpose.line = list(col = myCol))

plot(logit_next, par.settings = myPar, xTrans = function(x) log(x))
```



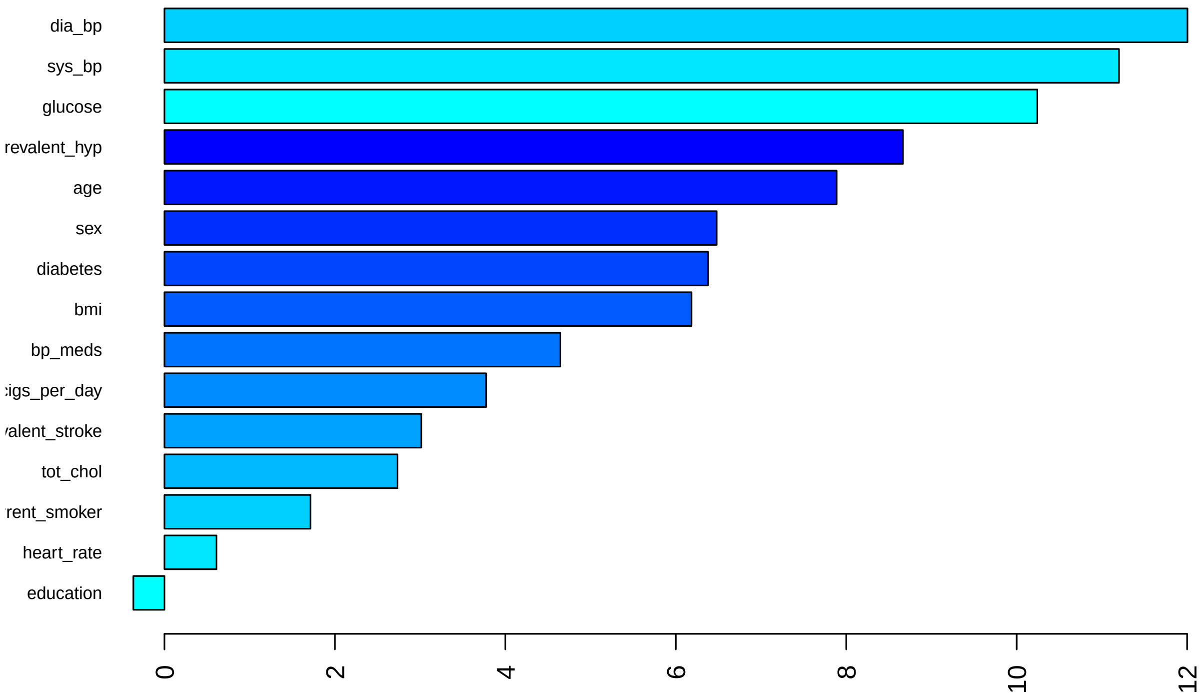

```
logit_tuning_graph = ggplot(logit_next, highlight = T) +
  scale_x_continuous(trans = "log") +
  labs(title = "Penalized Logistic Regression",
       x = "Lambda",
       y = "AUC")

# Variable importance
# Most important variables: age, sys_bp, sexfemale, cigs_per_day
logit_vip_graph = vip(logit_next, num_features = 20, method = "model")

# Test data: predicted probabilities
glmnet_pred_test_probs = predict(logit_next, newdata = testing_df, type = "prob",
                                 na.action = na.pass)[,1]

# Test data: predicted classes
glmnet_pred_test_class = predict(logit_next, newdata = testing_df, type = "raw",
                                 na.action = na.pass)

# Test data: confusion matrix
# Accuracy: 0.854
confusionMatrix(data = glmnet_pred_test_class,
                reference = y_test)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    CHD_present CHD_absent
##   CHD_present           4          0
##   CHD_absent          124        719
##
##                Accuracy : 0.854
##                  95% CI : (0.828, 0.877)
##     No Information Rate : 0.849
##     P-Value [Acc > NIR] : 0.372
##
##                   Kappa : 0.052
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.03125
##             Specificity : 1.00000
##          Pos Pred Value : 1.00000
##          Neg Pred Value : 0.85291
##              Prevalence : 0.15112
##          Detection Rate : 0.00472
##    Detection Prevalence : 0.00472
##       Balanced Accuracy : 0.51562
##
##        'Positive' Class : CHD_present
##
```

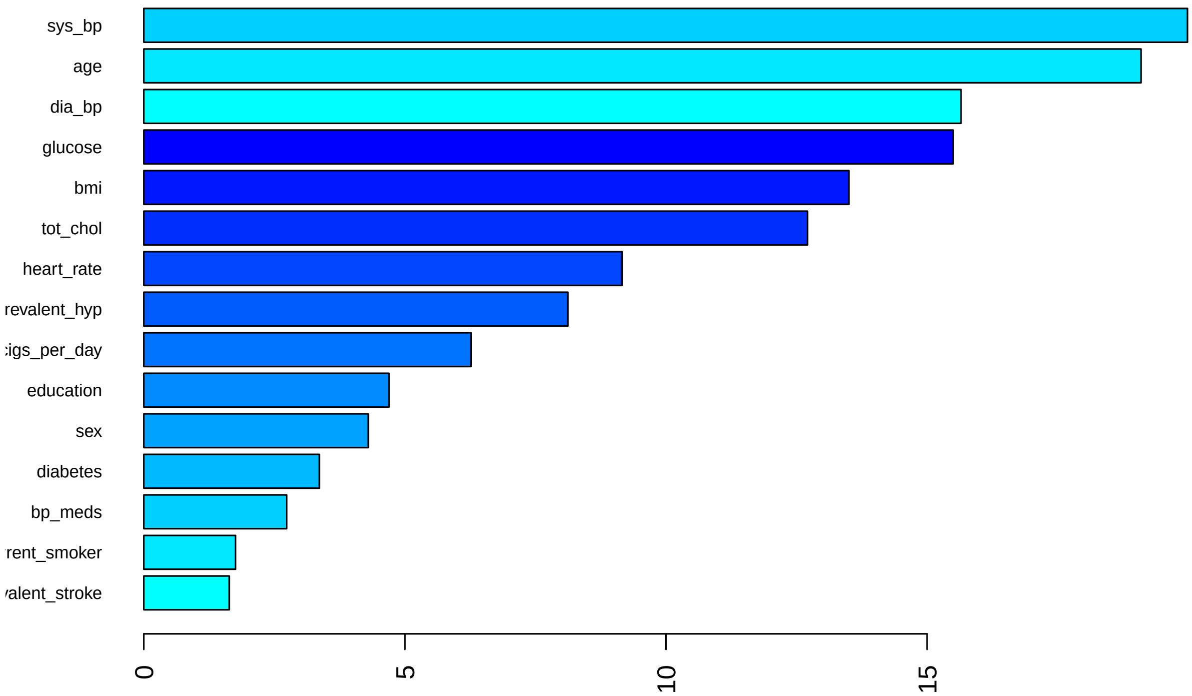
```r
# Random forest with imputation from recipes package
set.seed(2022)

# RF grid
rf_grid = expand.grid(mtry = 1:8,
                      splitrule = "gini",
                      min.node.size = seq(from = 2, to = 10, by = 2))

# Train random forest model
rf_fit = train(preprocess_recipe,
              data = training_df,
              method = "ranger",
              tuneGrid = rf_grid,
              metric = "ROC",
              trControl = ctrl)

# Optimal tuning parameters: 1 randomly selected predictor, min node size = 10
# Note: try tuning parameters > 10 min node size in grid?
ggplot(rf_fit, highlight = TRUE)
```




```
# Variable importance: permutation / gini
# sys_bp, dia_bp, glucose, prevalent_hyp
rf_recipe_var_imp_permutation = ranger(ten_year_chd ~ .,
                           training_df[complete.cases(training_df),],
                           mtry = rf_fit$bestTune[[1]],
                           splitrule = "gini",
                           min.node.size = rf_fit$bestTune[[3]],
                           importance = "permutation",
                           scale.permutation.importance = TRUE)

barplot(sort(ranger::importance(rf_recipe_var_imp_permutation), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("cyan", "blue"))(12))
```




```
# Variable importance: impurity / gini
# sys_bp, age, dia_bp, glucose
rf_recipe_var_imp_impurity = ranger(ten_year_chd ~ .,
                             training_df[complete.cases(training_df),],
                             mtry = rf_fit$bestTune[[1]],
                             splitrule = "gini",
                             min.node.size = rf_fit$bestTune[[3]],
                             importance = "impurity",
                             scale.permutation.importance = TRUE)

barplot(sort(ranger::importance(rf_recipe_var_imp_impurity), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("cyan", "blue"))(12))
```

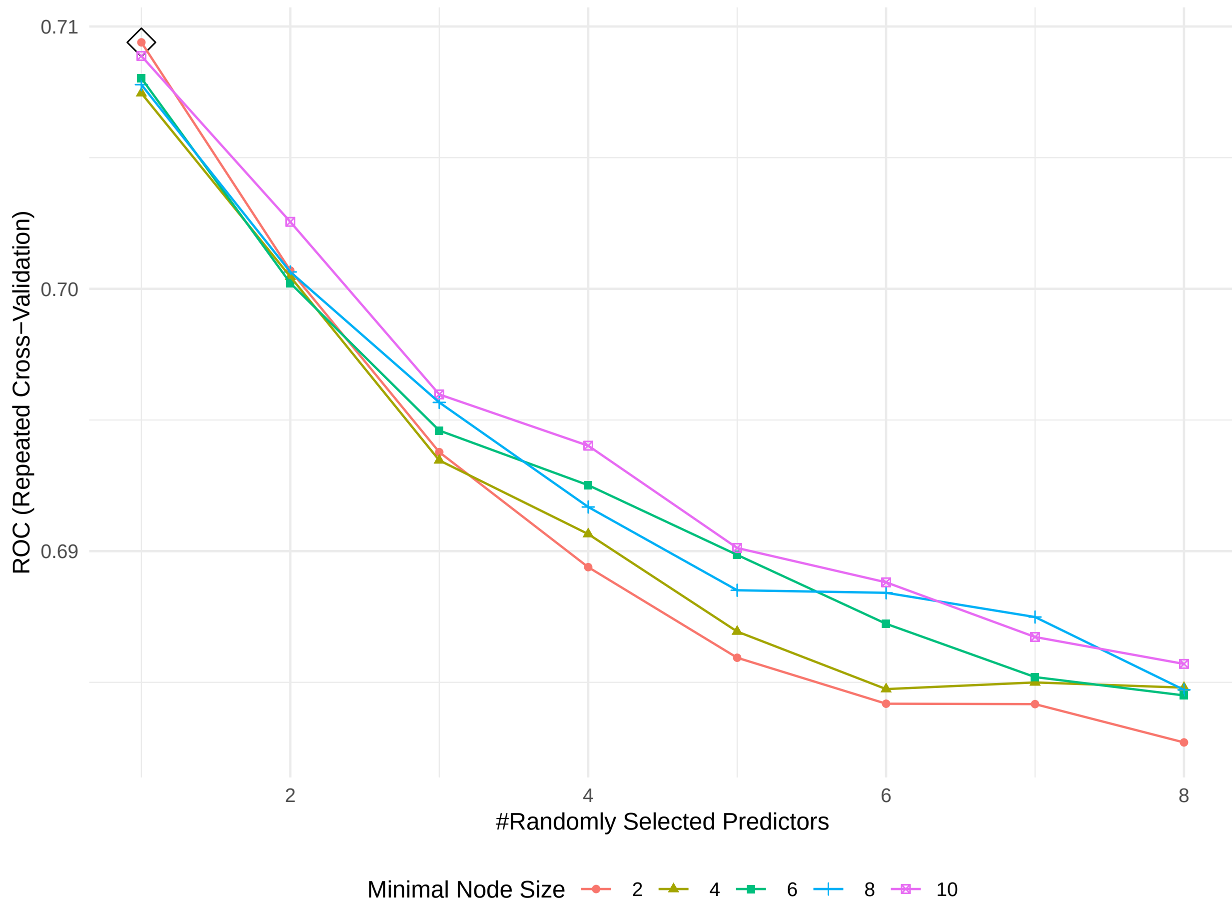



```r
# Test data: predicted probabilities
rf_pred_test_probs = predict(rf_fit, newdata = testing_df, type = "prob")[,1]

# Test data: predicted classes
rf_pred_test_class = predict(rf_fit, newdata = testing_df, type = "raw")

# Test data: confusion matrix
# Accuracy: 0.849
confusionMatrix(data = rf_pred_test_class,
                reference = y_test)
```

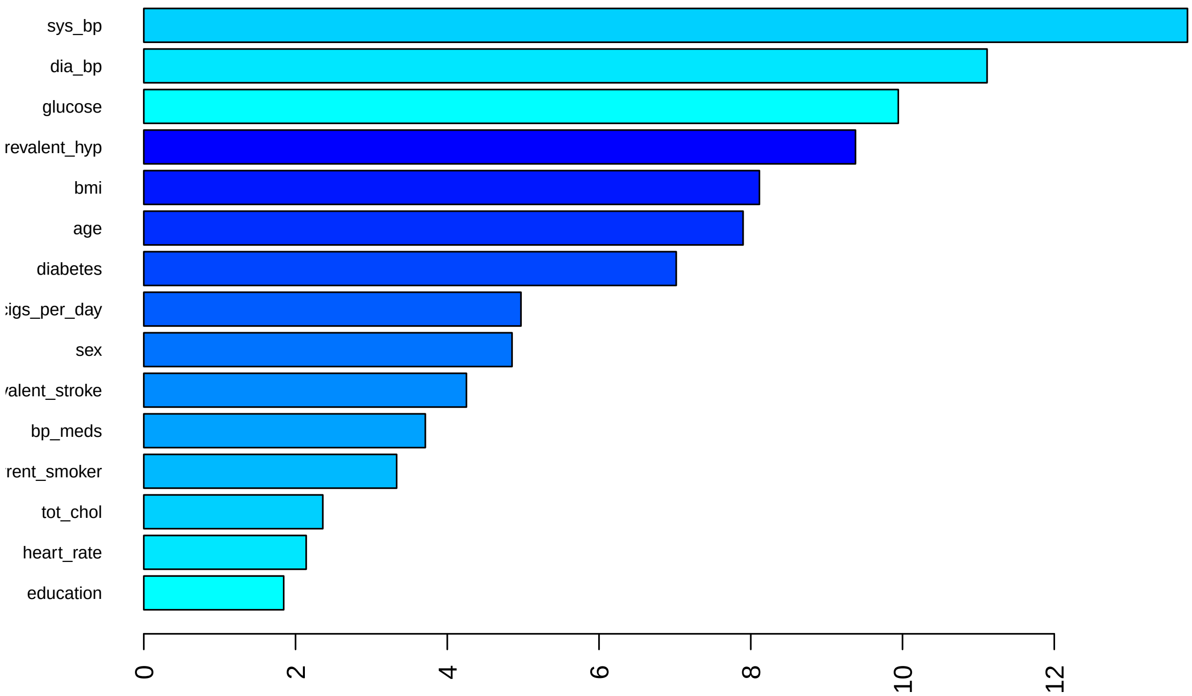
```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    CHD_present CHD_absent
##   CHD_present           0          0
##   CHD_absent          128        719
##
##                Accuracy : 0.849
##                  95% CI : (0.823, 0.872)
##     No Information Rate : 0.849
##     P-Value [Acc > NIR] : 0.524
##
```

```
##                  Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##            Sensitivity : 0.000
##            Specificity : 1.000
##         Pos Pred Value :   NaN
##         Neg Pred Value : 0.849
##             Prevalence : 0.151
##         Detection Rate : 0.000
##   Detection Prevalence : 0.000
##      Balanced Accuracy : 0.500
##
##       'Positive' Class : CHD_present
##
```

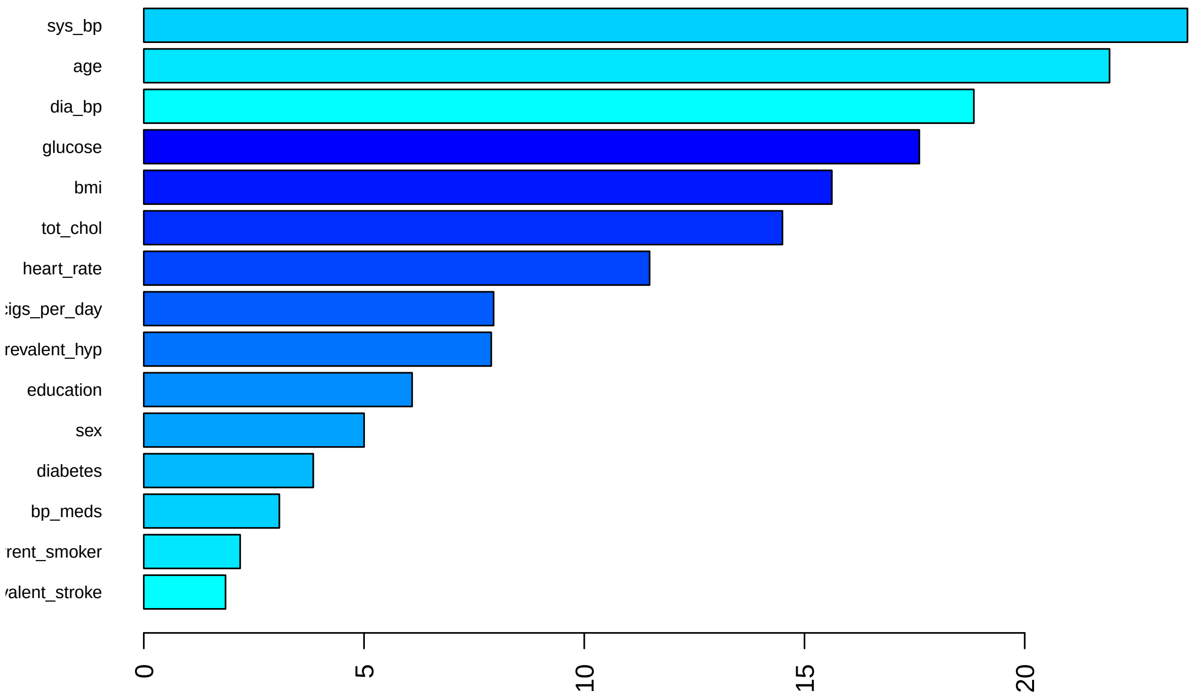
```
# Random forest with imputation from caret function directly (NOT recipes)
set.seed(2022)

ctrl_RF = trainControl(method = "repeatedcv",
                    repeats = 5,
                    summaryFunction = twoClassSummary,
                    classProbs = TRUE,
                    preProcOptions = list(k = 5))

rf_caret = train(ten_year_chd ~ .,
                   data = training_df,
                   na.action = na.pass,
                  method = "ranger",
                  tuneGrid = rf_grid,
                  metric = "ROC",
                  trControl = ctrl_RF,
                  preProcess = c("knnImpute", "center", "scale", "BoxCox"))

# Optimal tuning parameters: 1 randomly selected predictor, min node size = 2
ggplot(rf_caret, highlight = TRUE)
```




```r
# Variable importance: permutation / gini
# sys_bp, dia_bp, prevalent_hyp, glucose
rf_caret_var_imp_permutation = ranger(ten_year_chd ~ .,
                                      training_df[complete.cases(training_df),],
                                      mtry = rf_caret$bestTune[[1]],
                                      splitrule = "gini",
                                      min.node.size = rf_caret$bestTune[[3]],
                                      importance = "permutation",
                                      scale.permutation.importance = TRUE)

barplot(sort(ranger::importance(rf_caret_var_imp_permutation), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("cyan", "blue"))(12))
```




```r
# Variable importance: impurity / gini
# sys_bp, age, dia_bp, glucose
rf_caret_var_imp_impurity = ranger(ten_year_chd ~ .,
                                   training_df[complete.cases(training_df),],
                                   mtry = rf_caret$bestTune[[1]],
                                   splitrule = "gini",
                                   min.node.size = rf_caret$bestTune[[3]],
                                   importance = "impurity",
                                   scale.permutation.importance = TRUE)

barplot(sort(ranger::importance(rf_caret_var_imp_impurity), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("cyan", "blue"))(12))
```

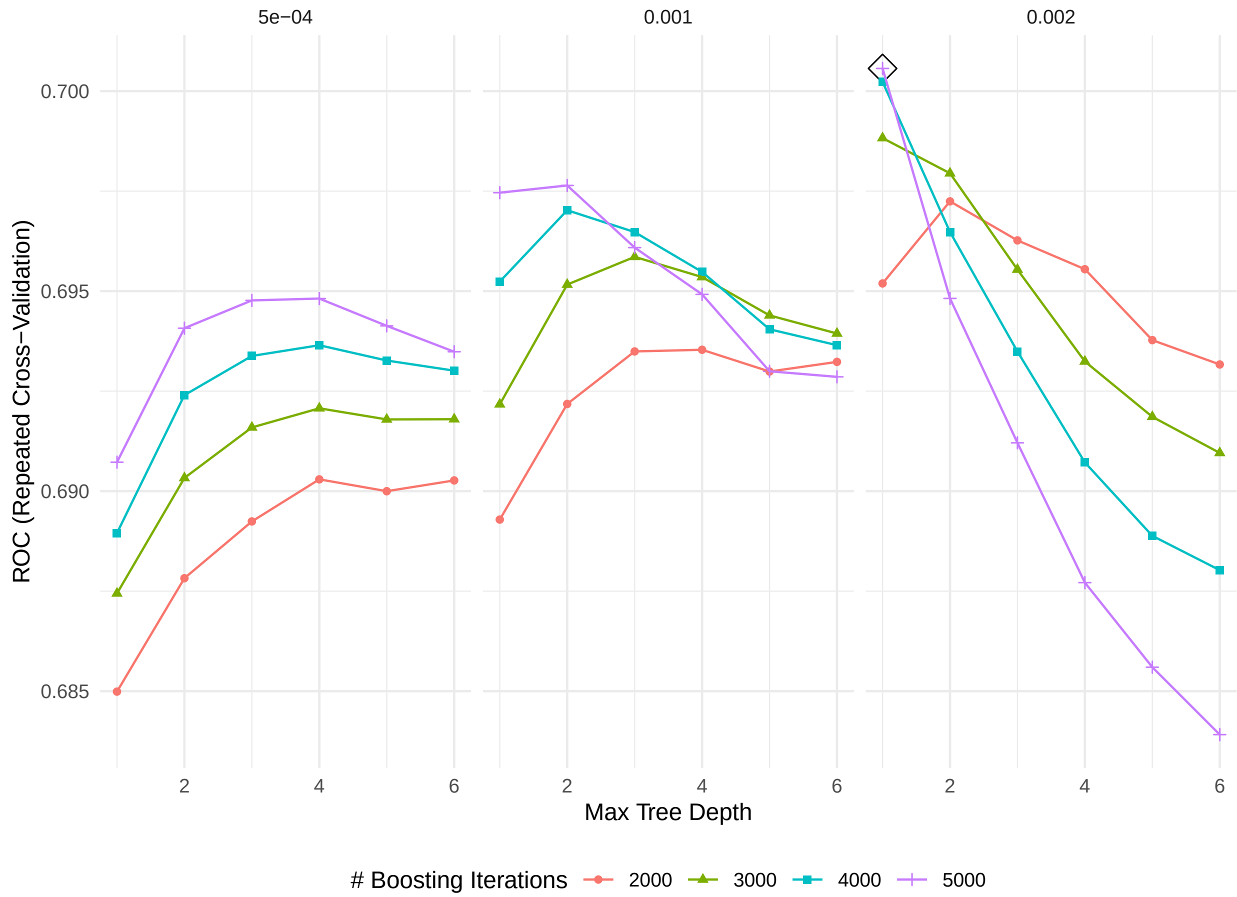



```r
# Test data: predicted probabilities
rf_caret_pred_test_probs = predict(rf_caret, newdata = testing_df, type = "prob",
                                   na.action = na.pass)[,1]

# Test data: predicted classes
rf_caret_pred_test_class = predict(rf_caret, newdata = testing_df, type = "raw",
                                   na.action = na.pass)

# Test data: confusion matrix
# Accuracy: 0.849
confusionMatrix(data = rf_caret_pred_test_class,
                reference = y_test)
```

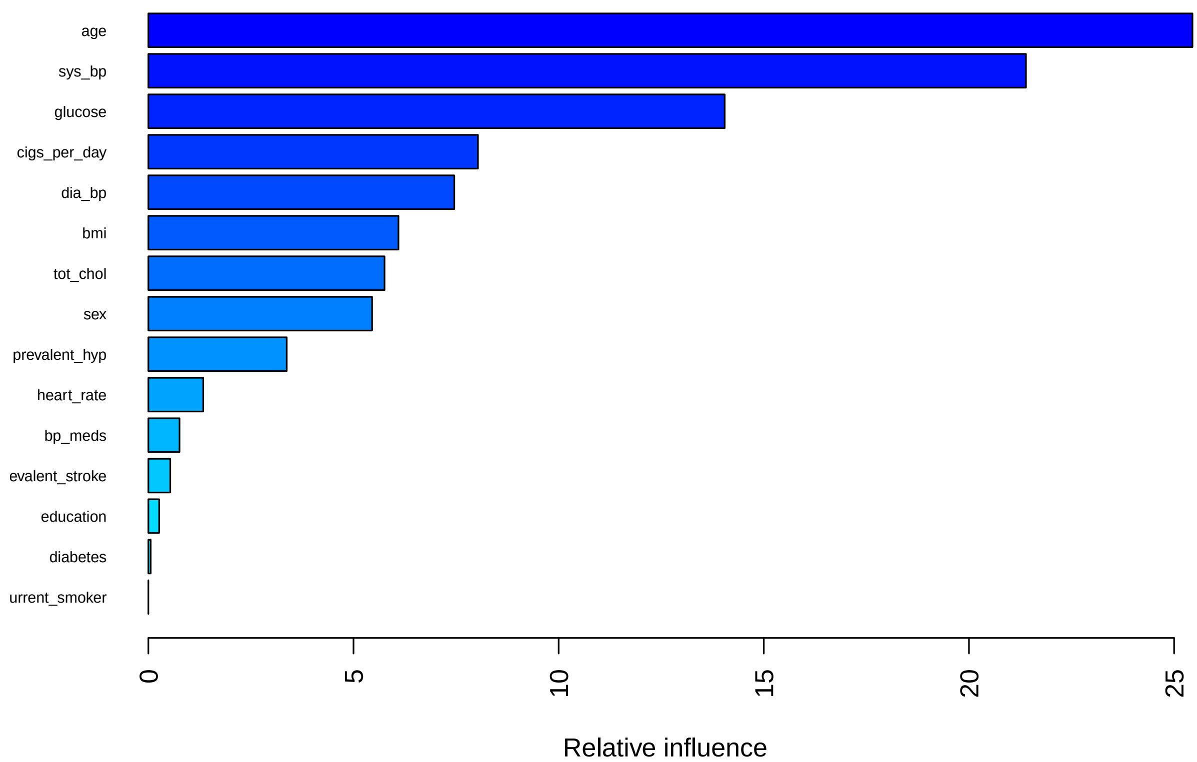
```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    CHD_present CHD_absent
##   CHD_present           0          0
##   CHD_absent          128        719
##
##                Accuracy : 0.849
##                  95% CI : (0.823, 0.872)
##     No Information Rate : 0.849
```

```
##     P-Value [Acc > NIR] : 0.524
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.000
##             Specificity : 1.000
##          Pos Pred Value :   NaN
##          Neg Pred Value : 0.849
##              Prevalence : 0.151
##          Detection Rate : 0.000
##    Detection Prevalence : 0.000
##       Balanced Accuracy : 0.500
##
##        'Positive' Class : CHD_present
##
```

```r
# Boosting with imputation from recipes package
set.seed(2022)

# Grid search for adaboost
adaboost_grid = expand.grid(n.trees = c(2000,3000,4000,5000),
                            interaction.depth = 1:6,
                            shrinkage = c(0.0005,0.001,0.002),
                            n.minobsinnode = 1)

# Train boosting model
boost_fit = train(preprocess_recipe,
                  data = training_df,
                  tuneGrid = adaboost_grid,
                  trControl = ctrl,
                  method = "gbm",
                  distribution = "adaboost",
                  metric = "ROC",
                  verbose = FALSE)

# Optimal tuning parameters: max tree depth = 1, 5000 boosting iterations, shrinkage = 0.002
ggplot(boost_fit, highlight = TRUE)
```




```r
# Variable importance
# age, sys_bp, glucose, cigs_per_day
summary(boost_fit$finalModel, las = 2, cBars = 19, cex.names = 0.6)
```




```
##                                   var rel.inf
## age                               age  25.448
## sys_bp                         sys_bp  21.390
## glucose                       glucose  14.047
## cigs_per_day             cigs_per_day   8.033
## dia_bp                         dia_bp   7.456
## bmi                               bmi   6.095
## tot_chol                     tot_chol   5.756
## sex                               sex   5.453
## prevalent_hyp           prevalent_hyp   3.372
## heart_rate                 heart_rate   1.337
## bp_meds                       bp_meds   0.760
## prevalent_stroke prevalent_stroke   0.534
## education                   education   0.263
## diabetes                     diabetes   0.057
## current_smoker         current_smoker   0.000
```

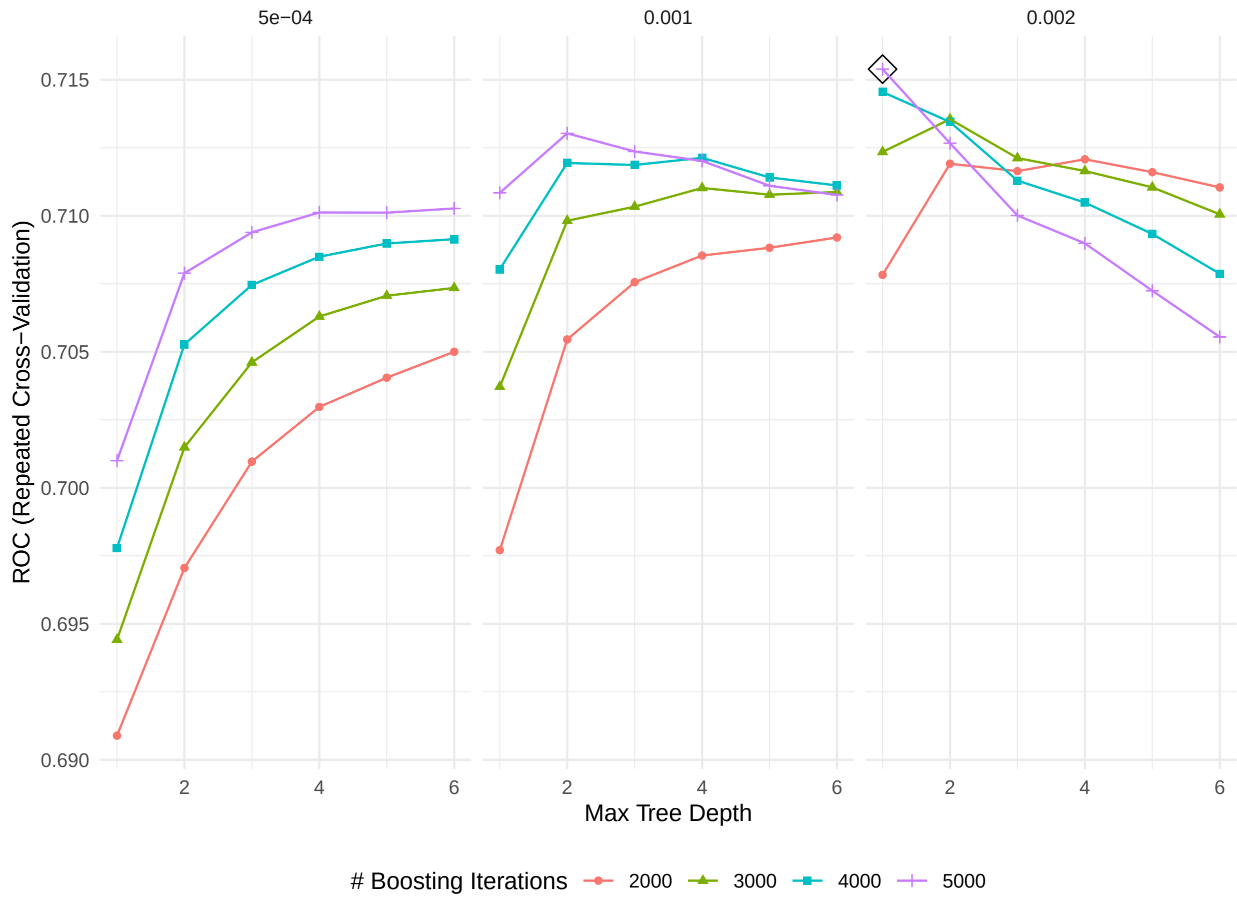
```
# Test data: predicted probabilities
boost_pred_test_probs = predict(boost_fit, newdata = testing_df, type = "prob")[,1]

# Test data: predicted classes
boost_pred_test_class = predict(boost_fit, newdata = testing_df, type = "raw")
```

```r
# Test data: confusion matrix
# Accuracy: 0.854
confusionMatrix(data = boost_pred_test_class,
                reference = y_test)
```

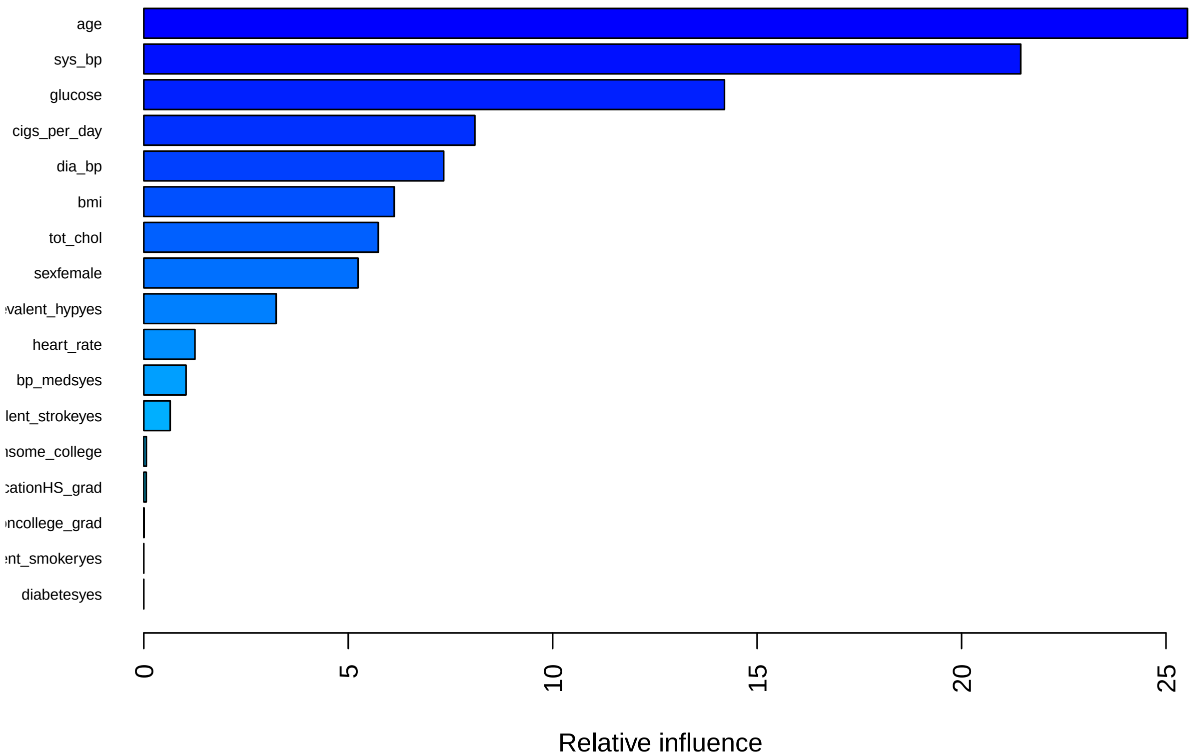
```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    CHD_present CHD_absent
##   CHD_present           7          3
##   CHD_absent          121        716
##
##                Accuracy : 0.854
##                  95% CI : (0.828, 0.877)
##     No Information Rate : 0.849
##     P-Value [Acc > NIR] : 0.372
##
##                   Kappa : 0.081
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.05469
##             Specificity : 0.99583
##          Pos Pred Value : 0.70000
##          Neg Pred Value : 0.85544
##              Prevalence : 0.15112
##          Detection Rate : 0.00826
##    Detection Prevalence : 0.01181
##       Balanced Accuracy : 0.52526
##
##        'Positive' Class : CHD_present
##
```

```r
# Boosting with imputation directly from caret (NOT recipes)
set.seed(2022)

ctrl_boost = trainControl(method = "repeatedcv",
                    repeats = 5,
                    summaryFunction = twoClassSummary,
                    classProbs = TRUE,
                    preProcOptions = list(k = 5))

# Train boosting model
boost_caret = train(ten_year_chd ~ .,
                   data = training_df,
                   na.action = na.pass,
                  tuneGrid = adaboost_grid,
                  trControl = ctrl_boost,
                  method = "gbm",
                  distribution = "adaboost",
                  metric = "ROC",
                  verbose = FALSE,
                  preProcess = c("knnImpute", "center", "scale", "BoxCox"))
```

```r
# Optimal tuning parameters
# Max tree depth = 1, 5000 boosting iterations, shrinkage = 0.002
ggplot(boost_caret, highlight = TRUE)
```




```r
# Variable importance
# age, sys_bp, glucose, cigs_per_day
summary(boost_caret$finalModel, las = 2, cBars = 19, cex.names = 0.6)
```



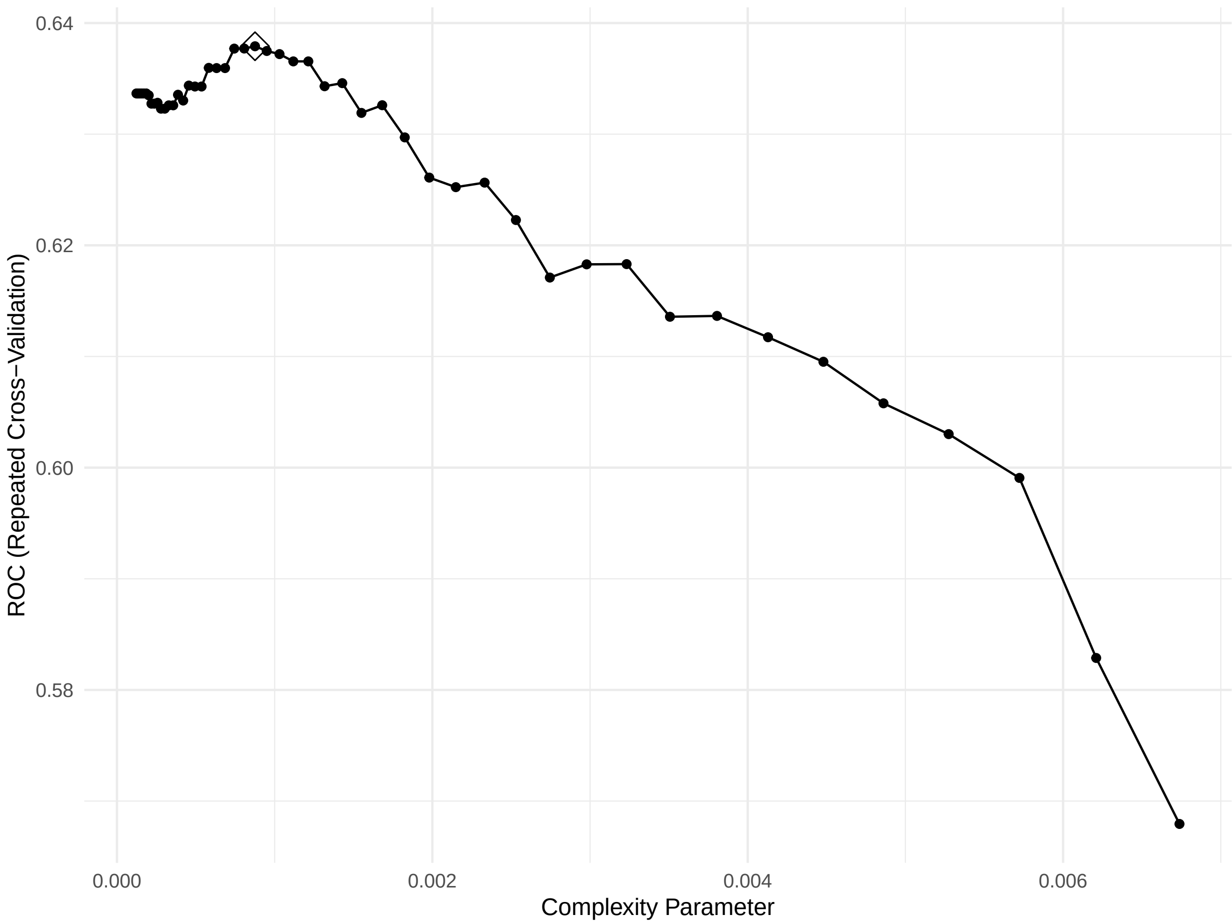

```
##                                         var   rel.inf
## age                                     age 25.52525
## sys_bp                               sys_bp 21.44600
## glucose                             glucose 14.20145
## cigs_per_day                   cigs_per_day  8.09721
## dia_bp                               dia_bp  7.33483
## bmi                                     bmi  6.12448
## tot_chol                           tot_chol  5.73410
## sexfemale                         sexfemale  5.24063
## prevalent_hypyes           prevalent_hypyes  3.23735
## heart_rate                       heart_rate  1.25110
## bp_medsyes                       bp_medsyes  1.03317
## prevalent_strokeyes     prevalent_strokeyes  0.64589
## educationsome_college educationsome_college  0.06271
## educationHS_grad           educationHS_grad  0.05990
## educationcollege_grad educationcollege_grad  0.00594
## current_smokeryes         current_smokeryes  0.00000
## diabetesyes                     diabetesyes  0.00000
```

```
# Test data: predicted probabilities
boost_caret_pred_test_probs = predict(boost_caret, newdata = testing_df, type = "prob",
                                      na.action = na.pass)[,1]
```

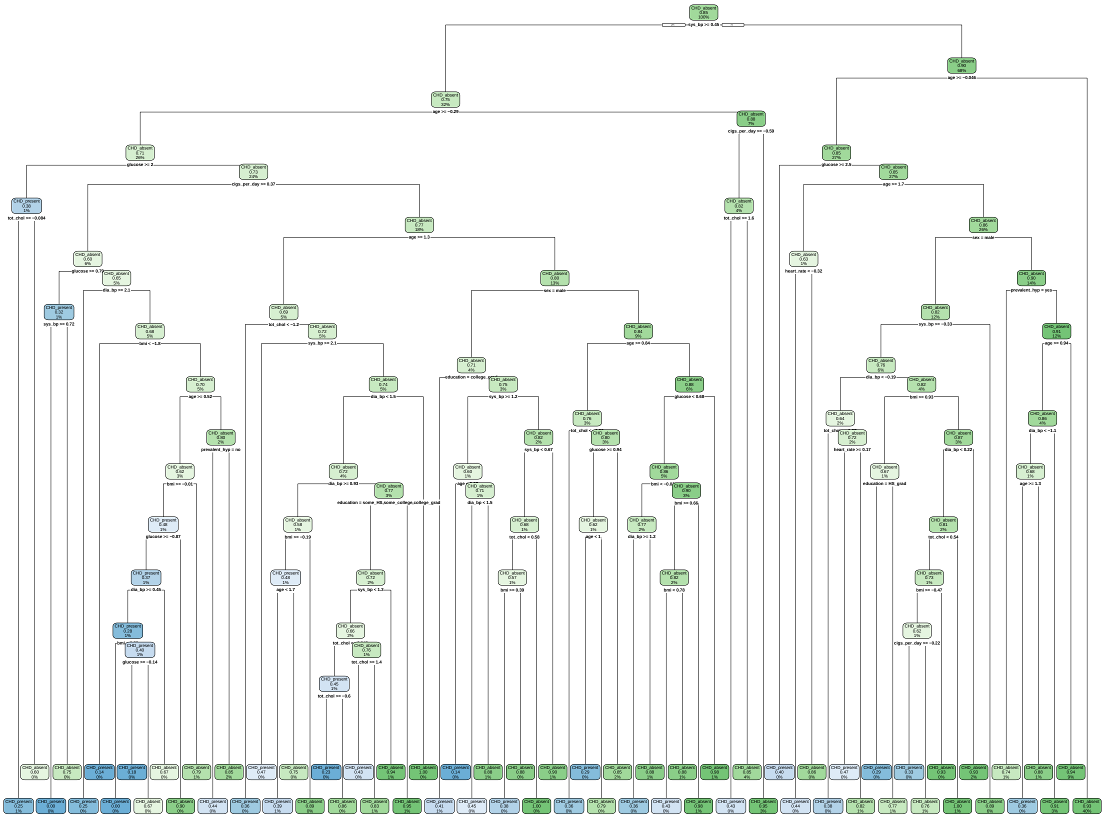
```r
# Test data: predicted classes
boost_caret_pred_test_class = predict(boost_caret, newdata = testing_df, type = "raw",
                                      na.action = na.pass)

# Test data: confusion matrix
# Accuracy: 0.851
confusionMatrix(data = boost_caret_pred_test_class,
                reference = y_test)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    CHD_present CHD_absent
##   CHD_present           4          2
##   CHD_absent          124        717
##
##                Accuracy : 0.851
##                  95% CI : (0.825, 0.875)
##     No Information Rate : 0.849
##     P-Value [Acc > NIR] : 0.447
##
##                   Kappa : 0.047
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.03125
##             Specificity : 0.99722
##          Pos Pred Value : 0.66667
##          Neg Pred Value : 0.85256
##              Prevalence : 0.15112
##          Detection Rate : 0.00472
##    Detection Prevalence : 0.00708
##       Balanced Accuracy : 0.51423
##
##        'Positive' Class : CHD_present
##
```

```r
# CART tree with recipe imputation
set.seed(2022)

cart_fit = train(preprocess_recipe,
                  data = training_df,
                   method = "rpart",
                   tuneGrid = data.frame(cp = exp(seq(-9,-5, len = 50))),
                   trControl = ctrl,
                   metric = "ROC")

# Optimal tuning parameter
ggplot(cart_fit, highlight = TRUE)
```




```
# Final model
rpart.plot(cart_fit$finalModel)
```

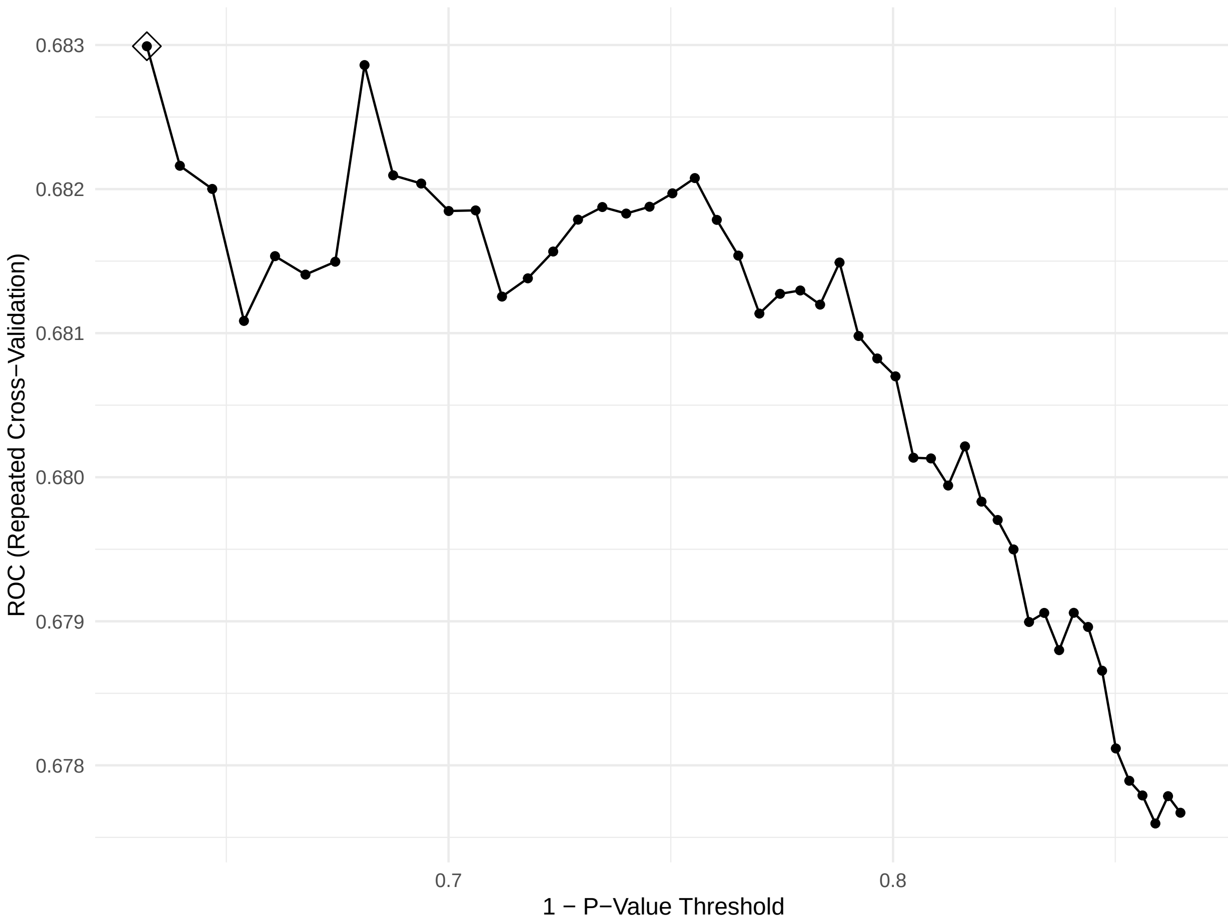


```r
# Test data: predicted probabilities
cart_pred_prob_test = predict(cart_fit, newdata = testing_df,
                              type = "prob")[,1]

# Test data: predicted classes
cart_pred_class_test = predict(cart_fit, newdata = testing_df, type = "raw")

# Test data: confusion matrix
# Accuracy: 0.799
confusionMatrix(data = cart_pred_class_test,
                reference = y_test)
```

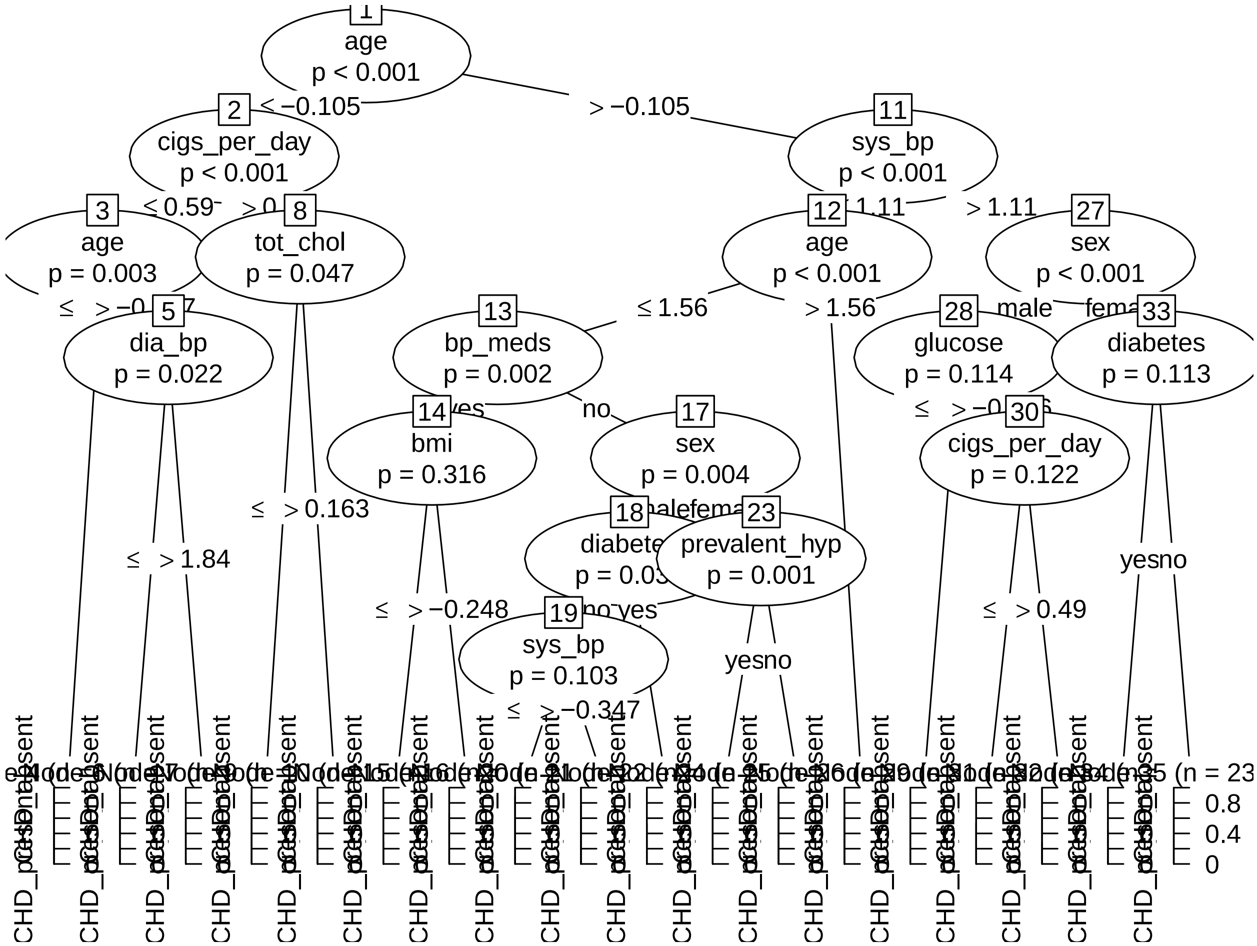
```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    CHD_present CHD_absent
##   CHD_present          24         66
##   CHD_absent          104        653
##
##                Accuracy : 0.799
##                  95% CI : (0.771, 0.826)
##     No Information Rate : 0.849
##     P-Value [Acc > NIR] : 0.99996
```

```
##
##                   Kappa : 0.109
##
##  Mcnemar's Test P-Value : 0.00454
##
##             Sensitivity : 0.1875
##             Specificity : 0.9082
##          Pos Pred Value : 0.2667
##          Neg Pred Value : 0.8626
##              Prevalence : 0.1511
##          Detection Rate : 0.0283
##    Detection Prevalence : 0.1063
##       Balanced Accuracy : 0.5479
##
##        'Positive' Class : CHD_present
##
```

```r
# CIT tree with recipe imputation
set.seed(2022)

cit_fit = train(preprocess_recipe,
                  data = training_df,
                   method = "ctree",
                   tuneGrid = data.frame(mincriterion = 1-exp(seq(-2, -1, length = 50))),
                   metric = "ROC",
                   trControl = ctrl)

# Optimal tuning parameter
ggplot(cit_fit, highlight = TRUE)
```




```
# Final model
plot(cit_fit$finalModel)
```




```
# Test data: predicted probabilities
cit_pred_prob_test = predict(cit_fit, newdata = testing_df,
                        type = "prob")[,1]

# Test data: predicted classes
cit_pred_class_test = predict(cit_fit, newdata = testing_df, type = "raw")

# Test data: confusion matrix
# Accuracy: 0.844
confusionMatrix(data = cit_pred_class_test,
                reference = y_test)
```

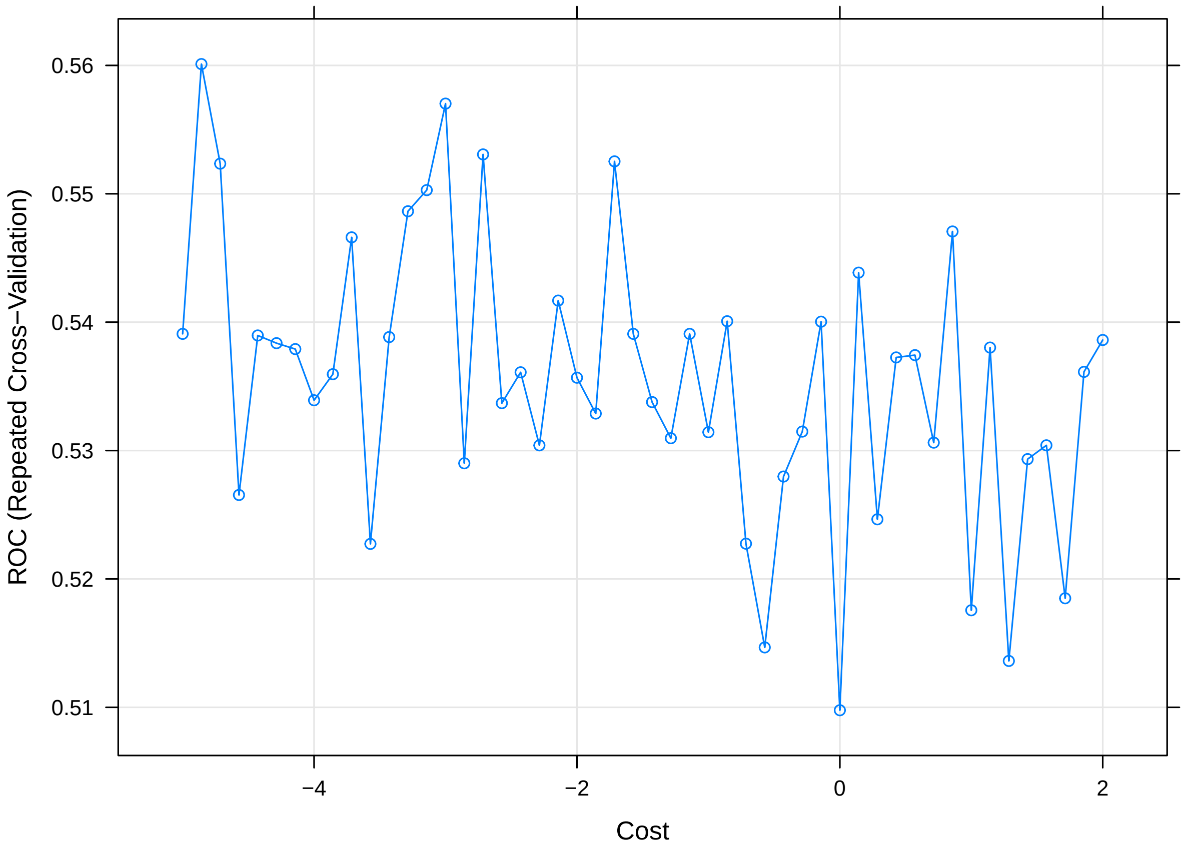
```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction     CHD_present CHD_absent
##   CHD_present            3          7
##   CHD_absent           125        712
##
##                  Accuracy : 0.844
##                    95% CI : (0.818, 0.868)
##       No Information Rate : 0.849
##       P-Value [Acc > NIR] : 0.67
```

```
##
##                   Kappa : 0.022
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.02344
##             Specificity : 0.99026
##          Pos Pred Value : 0.30000
##          Neg Pred Value : 0.85066
##              Prevalence : 0.15112
##          Detection Rate : 0.00354
##    Detection Prevalence : 0.01181
##       Balanced Accuracy : 0.50685
##
##        'Positive' Class : CHD_present
##
```

```
# SVM linear with impute in caret directly
# ROC seems < 0.6, quite poor
set.seed(2022)

ctrl_linear_svm = trainControl(method = "repeatedcv",
                    repeats = 5,
                    summaryFunction = twoClassSummary,
                    classProbs = TRUE,
                    preProcOptions = list(k = 5))

# I know we're told not to do this, but including Platt's probabilistic outputs here just to see...
svm_linear_fit = train(ten_year_chd ~ .,
                       data = training_df,
                       na.action = na.pass,
                       method = "svmLinear",
                       tuneGrid = data.frame(C = exp(seq(-5, 2, len = 50))),
                       trControl = ctrl_linear_svm,
                       prob.model = TRUE,
                   preProcess = c("knnImpute", "center", "scale", "BoxCox")
                       )
```

```
## maximum number of iterations reached 0.00319 -0.00316maximum number of iterations reached 0.00544 -0
```

```
plot(svm_linear_fit, highlight = TRUE, xTrans = log)
```




```r
# Test data: predicted probabilities
svm_linear_pred_prob_test = predict(svm_linear_fit, newdata = testing_df,
                        type = "prob", na.action = na.pass)[,1]

# Test data: predicted classes
svm_linear_pred_class_test = predict(svm_linear_fit, newdata = testing_df, type = "raw",
                                     na.action = na.pass)

# Test data: confusion matrix
# Accuracy: 0.849
confusionMatrix(data = svm_linear_pred_class_test,
                reference = y_test)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    CHD_present CHD_absent
##   CHD_present           0          0
##   CHD_absent          128        719
##
##                Accuracy : 0.849
##                  95% CI : (0.823, 0.872)
##     No Information Rate : 0.849
```

```
##     P-Value [Acc > NIR] : 0.524
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.000
##             Specificity : 1.000
##          Pos Pred Value :   NaN
##          Neg Pred Value : 0.849
##              Prevalence : 0.151
##          Detection Rate : 0.000
##    Detection Prevalence : 0.000
##       Balanced Accuracy : 0.500
##
##        'Positive' Class : CHD_present
##
```

```
# LDA in caret, imputation from caret (recipes gives errors, non-numeric argument to binary operator)
set.seed(2022)

lda_ctrl = trainControl(method = "repeatedcv",
                    repeats = 5,
                    summaryFunction = twoClassSummary,
                    classProbs = TRUE,
                    preProcOptions = list(k = 5))

LDA_model_caret = train(ten_year_chd ~ .,
                  data = training_df,
                  na.action = na.pass,
                  method = "lda",
                  metric = "ROC",
                  trControl = lda_ctrl,
                  preProcess = c("knnImpute", "center", "scale", "BoxCox"))

# Examine results
# AUC: 0.716
LDA_model_caret$results
```

```
##   parameter   ROC   Sens  Spec  ROCSD SensSD  SpecSD
## 1      none 0.716 0.0768 0.984 0.0427  0.034 0.00817
```

```
# Test data: predicted probabilities
lda_pred_prob_test = predict(LDA_model_caret, newdata = testing_df,
                       type = "prob", na.action = na.pass)[,1]

# Test data: predicted classes
lda_pred_class_test = predict(LDA_model_caret, newdata = testing_df, type = "raw",
                                  na.action = na.pass)

# Test data: confusion matrix
# Accuracy: 0.85
confusionMatrix(data = lda_pred_class_test,
                reference = y_test)
```

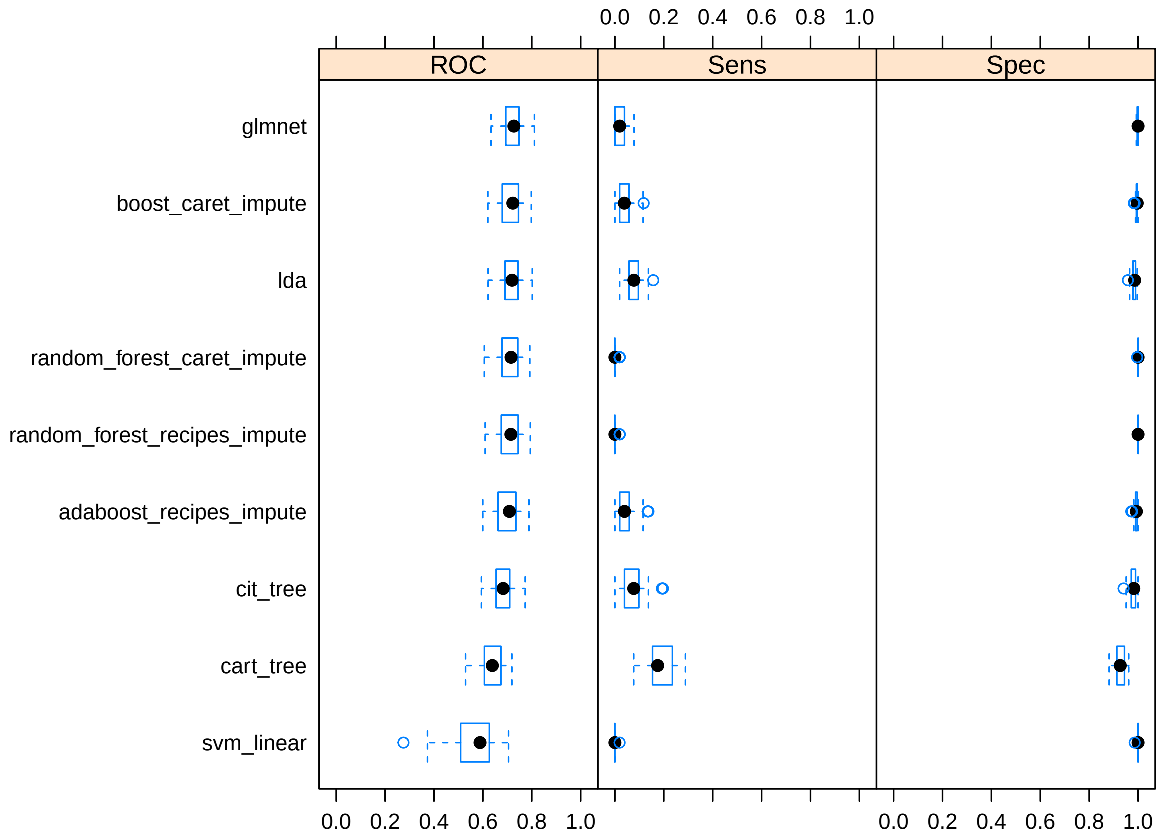
```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    CHD_present CHD_absent
##   CHD_present          10          9
##   CHD_absent          118        710
##
##                Accuracy : 0.85
##                  95% CI : (0.824, 0.873)
##     No Information Rate : 0.849
##     P-Value [Acc > NIR] : 0.485
##
##                   Kappa : 0.101
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0781
##             Specificity : 0.9875
##          Pos Pred Value : 0.5263
##          Neg Pred Value : 0.8575
##              Prevalence : 0.1511
##          Detection Rate : 0.0118
##    Detection Prevalence : 0.0224
##       Balanced Accuracy : 0.5328
##
##        'Positive' Class : CHD_present
##
```

```
# Results from resampling on training data
resamp = resamples(list(random_forest_recipes_impute = rf_fit,
                         adaboost_recipes_impute = boost_fit,
                        glmnet = logit_next,
                        random_forest_caret_impute = rf_caret,
                        boost_caret_impute = boost_caret,
                        cart_tree = cart_fit,
                        cit_tree = cit_fit,
                        svm_linear = svm_linear_fit,
                        lda = LDA_model_caret))

# Median AUC is highest for glmnet (0.727), and boost_caret_impute (0.722)
summary(resamp)
```

```
##
## Call:
## summary.resamples(object = resamp)
##
## Models: random_forest_recipes_impute, adaboost_recipes_impute, glmnet, random_forest_caret_impute, bo
## Number of resamples: 50
##
## ROC
##                               Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's
## random_forest_recipes_impute 0.609   0.677  0.714 0.708   0.743 0.794    0
## adaboost_recipes_impute      0.600   0.662  0.709 0.701   0.734 0.788    0
## glmnet                       0.633   0.694  0.727 0.721   0.747 0.811    0
```

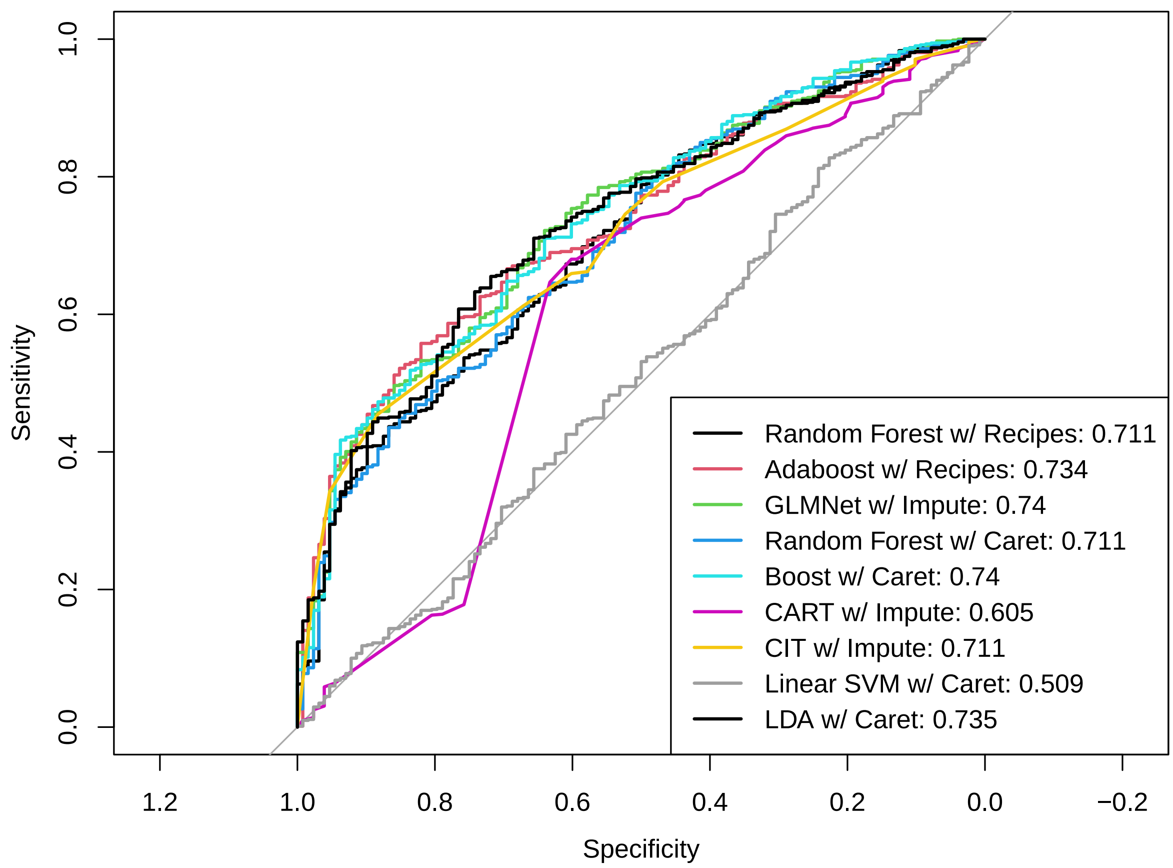
```
## random_forest_caret_impute   0.606   0.678  0.715 0.709   0.743 0.792    0
## boost_caret_impute           0.620   0.679  0.722 0.715   0.745 0.798    0
## cart_tree                    0.529   0.607  0.639 0.638   0.673 0.719    0
## cit_tree                     0.594   0.655  0.683 0.683   0.710 0.773    0
## svm_linear                   0.275   0.509  0.588 0.560   0.626 0.705    0
## lda                          0.621   0.691  0.719 0.716   0.742 0.802    0
##
## Sens
##                                Min. 1st Qu. Median     Mean 3rd Qu.   Max. NA's
## random_forest_recipes_impute 0.0000  0.0000 0.0000 0.001169  0.0000 0.0196    0
## adaboost_recipes_impute      0.0000  0.0196 0.0392 0.048024  0.0588 0.1373    0
## glmnet                       0.0000  0.0000 0.0196 0.025573  0.0390 0.0784    0
## random_forest_caret_impute   0.0000  0.0000 0.0000 0.001938  0.0000 0.0196    0
## boost_caret_impute           0.0000  0.0192 0.0385 0.037217  0.0577 0.1176    0
## cart_tree                    0.0769  0.1538 0.1748 0.185226  0.2353 0.2885    0
## cit_tree                     0.0000  0.0438 0.0769 0.076772  0.0976 0.1961    0
## svm_linear                   0.0000  0.0000 0.0000 0.000385  0.0000 0.0192    0
## lda                          0.0192  0.0577 0.0777 0.076795  0.0962 0.1569    0
##
## Spec
##                               Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's
## random_forest_recipes_impute 1.000   1.000  1.000 1.000   1.000 1.000    0
## adaboost_recipes_impute      0.972   0.990  0.993 0.993   0.997 1.000    0
## glmnet                       0.993   0.997  1.000 0.998   1.000 1.000    0
## random_forest_caret_impute   0.997   1.000  1.000 1.000   1.000 1.000    0
## boost_caret_impute           0.983   0.993  0.997 0.995   0.997 1.000    0
## cart_tree                    0.882   0.914  0.927 0.927   0.944 0.962    0
## cit_tree                     0.941   0.972  0.983 0.980   0.990 1.000    0
## svm_linear                   0.986   1.000  1.000 1.000   1.000 1.000    0
## lda                          0.958   0.979  0.986 0.984   0.990 0.997    0
```

```r
bwplot(resamp, layout = c(3, 1))
```




```r
# ROC curves for fitted models applied to testing data
# AUC is highest for glmnet and boost_caret_impute (0.74 for both)

roc_rf_recipes_impute = roc(y_test, rf_pred_test_probs)
roc_boost_impute = roc(y_test, boost_pred_test_probs)
roc_glmnet = roc(y_test, glmnet_pred_test_probs)
roc_rf_caret_impute = roc(y_test, rf_caret_pred_test_probs)
roc_boost_caret_impute = roc(y_test, boost_caret_pred_test_probs)
roc_cart_impute = roc(y_test, cart_pred_prob_test)
roc_cit_impute = roc(y_test, cit_pred_prob_test)
roc_svm_linear_impute = roc(y_test, svm_linear_pred_prob_test)
roc_lda = roc(y_test, lda_pred_prob_test)

plot(roc_rf_recipes_impute, col = 1)
plot(roc_boost_impute, add = TRUE, col = 2)
plot(roc_glmnet, add = TRUE, col = 3)
plot(roc_rf_caret_impute, add = TRUE, col = 4)
plot(roc_boost_caret_impute, add = TRUE, col = 5)
plot(roc_cart_impute, add = TRUE, col = 6)
plot(roc_cit_impute, add = TRUE, col = 7)
plot(roc_svm_linear_impute, add = TRUE, col = 8)
plot(roc_lda, add = TRUE, col = 9)
```

```r
auc = c(roc_rf_recipes_impute$auc[1], roc_boost_impute$auc[1], roc_glmnet$auc[1],
        roc_rf_caret_impute$auc[1], roc_boost_caret_impute$auc[1], roc_cart_impute$auc[1],
        roc_cit_impute$auc[1], roc_svm_linear_impute$auc[1], roc_lda$auc[1])

model_names = c("Random Forest w/ Recipes", "Adaboost w/ Recipes", "GLMNet w/ Impute",
                "Random Forest w/ Caret", "Boost w/ Caret", "CART w/ Impute", "CIT w/ Impute", "Linear S

legend("bottomright", legend = paste0(model_names, ": ", round(auc, 3)),
       col = 1:9, lwd = 2)
```



Still In Progress

```r
# Trying SVM with radial classifier for fun
# set.seed(2022)
#
# svm_grid = expand.grid(C = exp(seq(-2, 3, len = 50)),
#                        sigma = exp(seq(-3, 0, len = 50)))
#
# svm_fit_impute_classes = train(preprocess_recipe,
#                                data = training_df,
```

```
#                         method = "svmRadialSigma",
#                         tuneGrid = svm_grid,
#                         trControl = ctrl)
#
# # I know we're told not to do this, but including Platt's probabilistic outputs here just to see...
# svm_fit_impute_probs = train(preprocess_recipe,
#                               data = training_df,
#                               method = "svmRadialSigma",
#                               tuneGrid = svm_grid,
#                               trControl = ctrl,
#                               prob.model = TRUE)
```

```
# SVM with linear kernel, imputation from recipe
# Doesn't work; will try impute from train function instead
# set.seed(2022)
#
# svm_linear_fit = train(preprocess_recipe,
#                          data = training_df,
#                          method = "svmLinear",
#                          tuneGrid = data.frame(C = exp(seq(-5, 2, len = 50))),
#                          trControl = ctrl
#                          )
#
# plot(svm_linear_fit, highlight = TRUE, xTrans = log)
```

```
# # Penalized logistic regression
# # Doesn't work
# # https://stackoverflow.com/questions/48179423/error-error-in-lognetx-is-sparse-ix-jx-y-weights-offse
# set.seed(2022)
#
# glm_grid = expand.grid(alpha = seq(0, 1, length = 11),
#                          lambda = exp(seq(-8, -3, length = 19)))
#
# logit_glm = train(preprocess_recipe,
#                    data = training_df,
#                    method = "glmnet",
#                    tuneGrid = glm_grid,
#                    metric = "ROC",
#                    trControl = ctrl,
#                    family = "binomial")
```