Ensemble Methods I

Yifei Sun, Runze Cui

Contents

Regression **2**
Bagging and Random forests 2
Boosting 3
Grid search using `caret` 3
Global interpretation 6

Classification **9**
Bagging and random forests 9
Boosting 10
Grid search using `caret` 11
Global interpretation 13

```
library(tidyverse)
library(ISLR)
library(mlbench)
library(caret)
library(tidymodels)
library(randomForest)
library(ranger)
library(gbm)
library(pdp)
library(pROC)
```

Regression

Predict a baseball player's salary on the basis of various statistics associated with performance in the previous year.

```
data(Hitters)
Hitters <- na.omit(Hitters)

set.seed(2023)
data_split <- initial_split(Hitters, prop = 0.8)

# Extract the training and test data
training_data <- training(data_split)
testing_data <- testing(data_split)
```

Bagging and Random forests

The function `randomForest()` implements Breiman's random forest algorithm (based on Breiman and Cutler's original Fortran code) for classification and regression. `ranger()` is a fast implementation of Breiman's random forests, particularly suited for high dimensional data.

```
set.seed(1)
bagging <- randomForest(Salary ~ . ,
                        data = training_data,
                        mtry = 19)

set.seed(1)
rf <- randomForest(Salary ~ . ,
                   data = training_data,
                   mtry = 6)

# fast implementation
set.seed(1)
rf2 <- ranger(Salary ~ . ,
              data = training_data,
              mtry = 6)

pred.rf <- predict(rf, newdata = testing_data)
pred.rf2 <- predict(rf2, data = testing_data)$predictions

RMSE(pred.rf, testing_data$Salary)
```

```
## [1] 436.8625
```

```
RMSE(pred.rf2, testing_data$Salary)
```

```
## [1] 443.0182
```

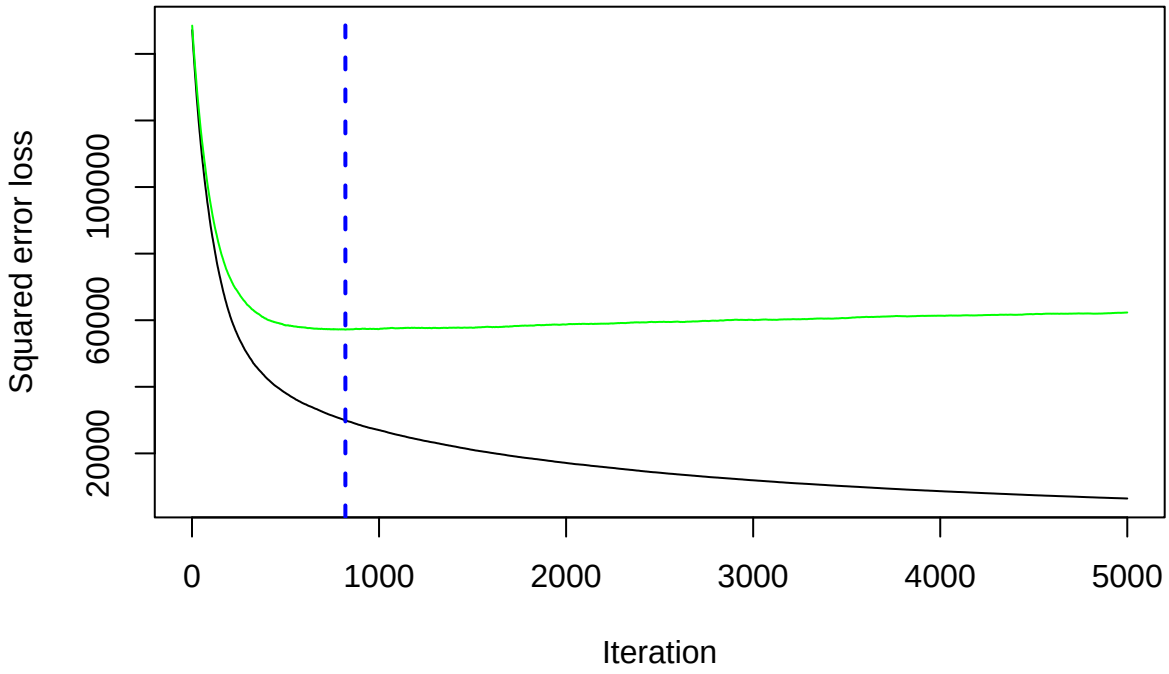
Boosting

We first fit a gradient boosting model with Gaussian loss function.

```
set.seed(1)
bst <- gbm(Salary ~ . ,
           data = training_data,
           distribution = "gaussian",
           n.trees = 5000,
           interaction.depth = 3,
           shrinkage = 0.005,
           cv.folds = 10,
           n.cores = 2)
```

We plot loss function as a result of number of trees added to the ensemble.

```
gbm.perf(bst, method = "cv")
```



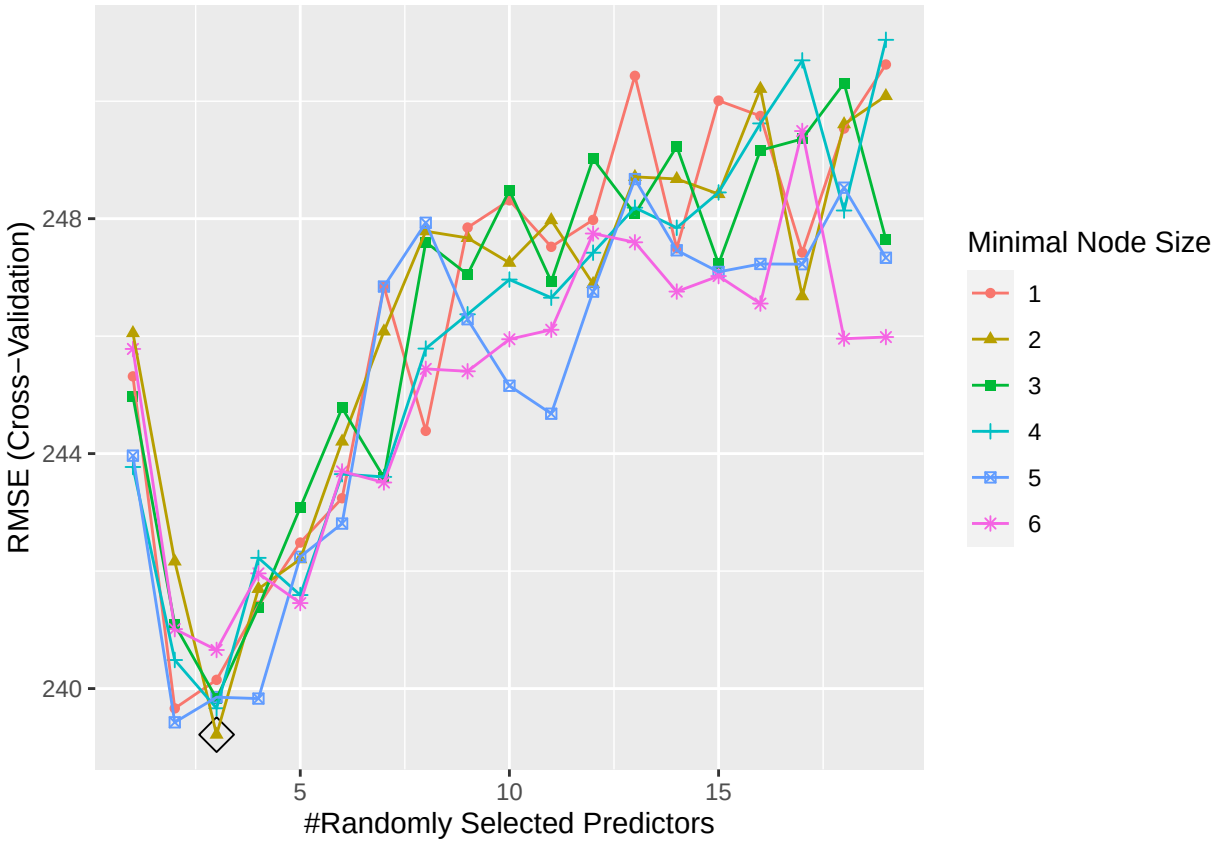

```
## [1] 820
```

Grid search using `caret`

We use the fast implementation of random forest when tuning the model.

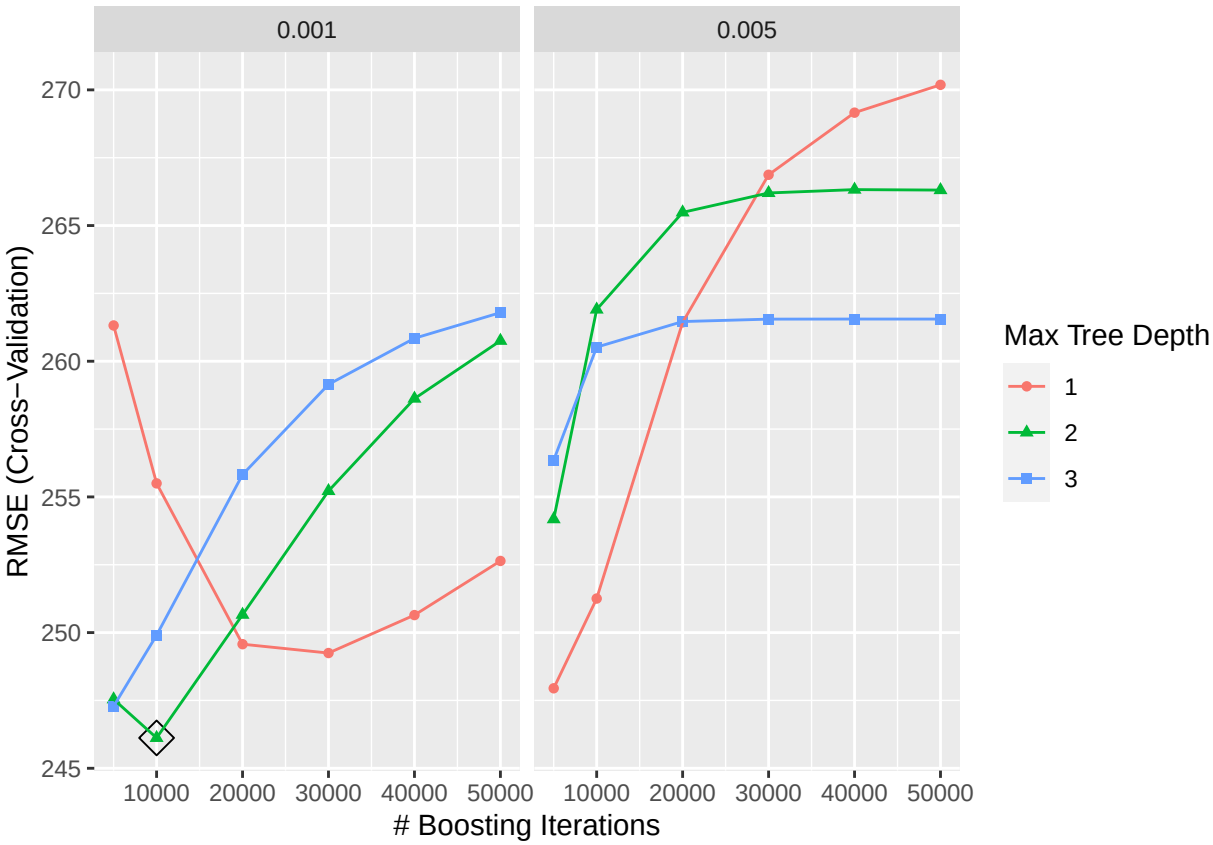
```
ctrl <- trainControl(method = "cv")

# Try more if possible
rf.grid <- expand.grid(mtry = 1:19,
                       splitrule = "variance",
                       min.node.size = 1:6)
set.seed(1)
```

```
rf.fit <- train(Salary ~ . ,
                data = training_data,
                method = "ranger",
                tuneGrid = rf.grid,
                trControl = ctrl)

ggplot(rf.fit, highlight = TRUE)
```



We then tune the `gbm` model.

```
# Try more
gbm.grid <- expand.grid(n.trees = c(5000,10000,20000,30000,40000,50000),
                        interaction.depth = 1:3,
                        shrinkage = c(0.001,0.005),
                        n.minobsinnode = c(1))
set.seed(1)
gbm.fit <- train(Salary ~ . ,
                 data = training_data,
                 method = "gbm",
                 tuneGrid = gbm.grid,
                 trControl = ctrl,
                 verbose = FALSE)

ggplot(gbm.fit, highlight = TRUE)
```



It takes a while to train the `gbm` even with a rough tuning grid. The `xgboost` package provides an efficient implementation of gradient boosting framework (approximately 10x faster than `gbm`). You can find much useful information here: https://github.com/dmlc/xgboost/tree/master/demo.

Compare the cross-validation performance. You can also compare with other models that we fitted before.

```
resamp <- resamples(list(rf = rf.fit, gbm = gbm.fit))
summary(resamp)
```

```
##
## Call:
## summary.resamples(object = resamp)
##
## Models: rf, gbm
## Number of resamples: 10
##
## MAE
##         Min.  1st Qu.  Median     Mean  3rd Qu.     Max. NA's
## rf   88.4984 144.7927 153.446 150.0579 158.7698 190.6592    0
## gbm 127.0154 157.8617 167.379 164.1479 172.0279 197.4903    0
##
## RMSE
##         Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## rf  126.8479 217.0049 240.9261 239.2156 254.5401 342.9996    0
## gbm 167.2814 219.9873 245.4644 246.1187 261.6844 333.5736    0
##
## Rsquared
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
```

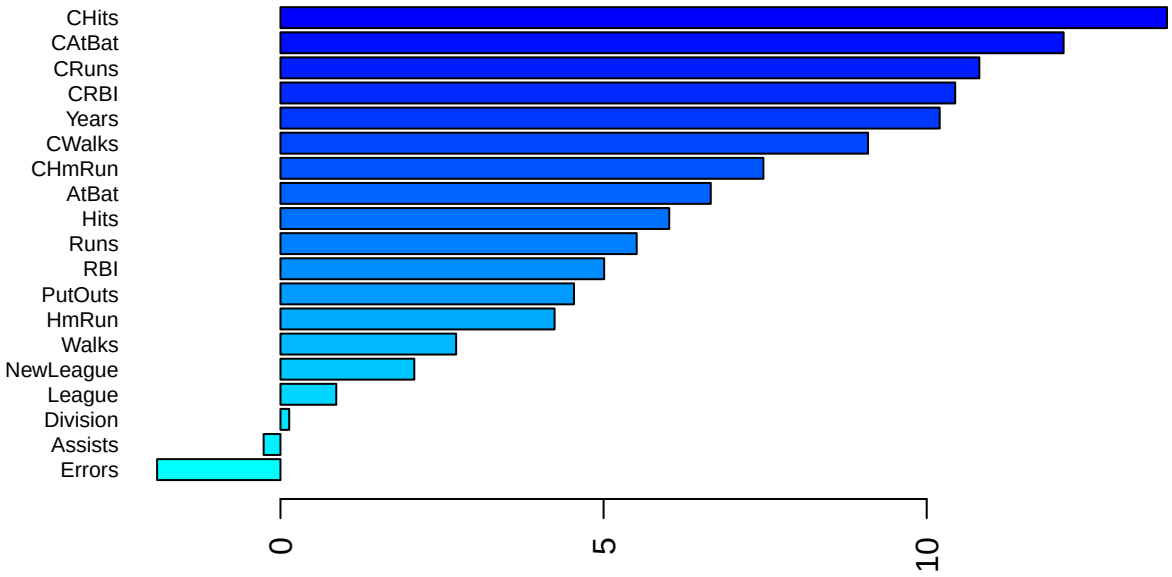
```
## rf   0.3284335 0.6469252 0.6662064 0.6379257 0.7023984 0.8653989    0
## gbm  0.3715395 0.5951801 0.6419476 0.6153784 0.6795782 0.7384643    0
```

Global interpretation

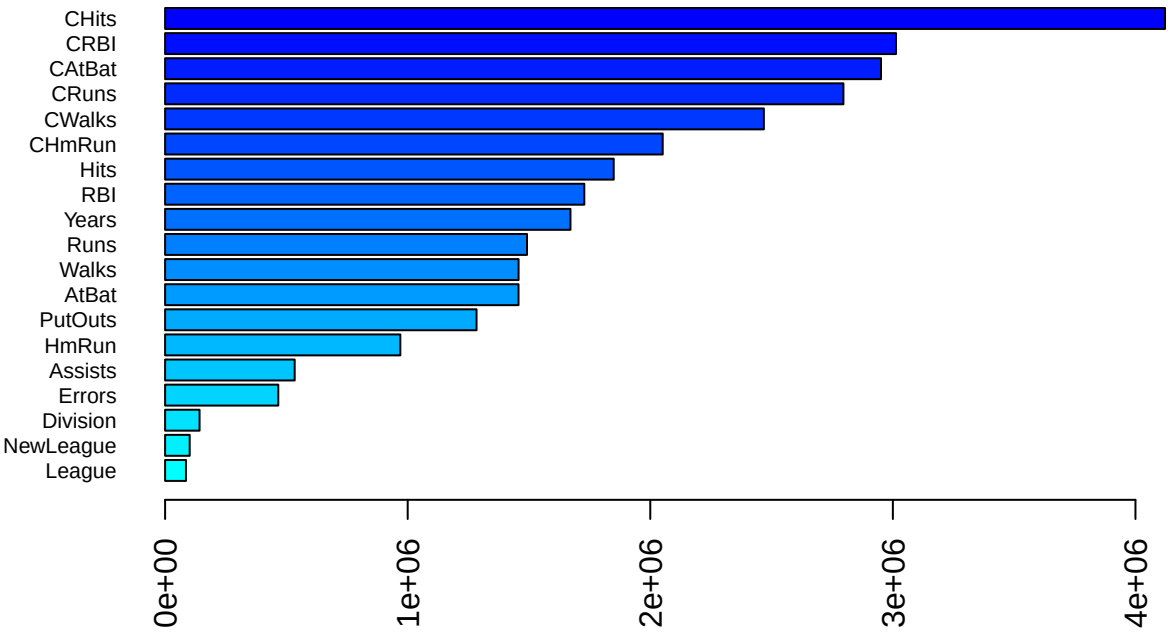
Variable importance

We can extract the variable importance from the fitted models. In what follows, the first measure is computed from permuting OOB data. The second measure is the total decrease in node impurities from splitting on the variable, averaged over all trees. For regression, node impurity is measured by residual sum of squares.

```
set.seed(1)
rf2.final.per <- ranger(Salary ~ . ,
                        data = training_data,
                        mtry = rf.fit$bestTune[[1]],
                        splitrule = "variance",
                        min.node.size = rf.fit$bestTune[[3]],
                        importance = "permutation",
                        scale.permutation.importance = TRUE)

barplot(sort(ranger::importance(rf2.final.per), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("cyan","blue"))(19))
```

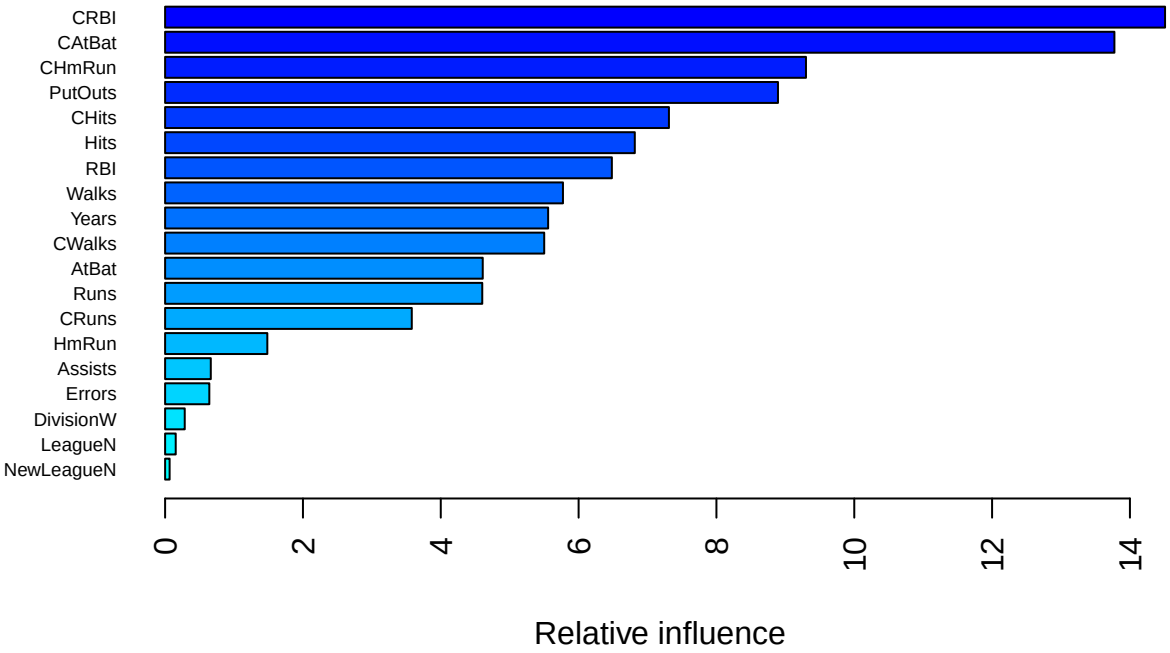



```
set.seed(1)
rf2.final.imp <- ranger(Salary ~ . ,
                        data = training_data,
                        mtry = rf.fit$bestTune[[1]],
                        splitrule = "variance",
                        min.node.size = rf.fit$bestTune[[3]],
                        importance = "impurity")

barplot(sort(ranger::importance(rf2.final.imp), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("cyan","blue"))(19))
```



Variable importance from boosting can be obtained using the `summary()` function.

```
summary(gbm.fit$finalModel, las = 2, cBars = 19, cex.names = 0.6)
```




```
##                 var     rel.inf
## CRBI            CRBI 14.50964876
## CAtBat        CAtBat 13.77487682
## CHmRun        CHmRun  9.29939917
## PutOuts      PutOuts  8.89312612
## CHits          CHits  7.31059318
## Hits            Hits  6.81554966
## RBI              RBI  6.48265148
## Walks          Walks  5.77290041
## Years          Years  5.55782643
```

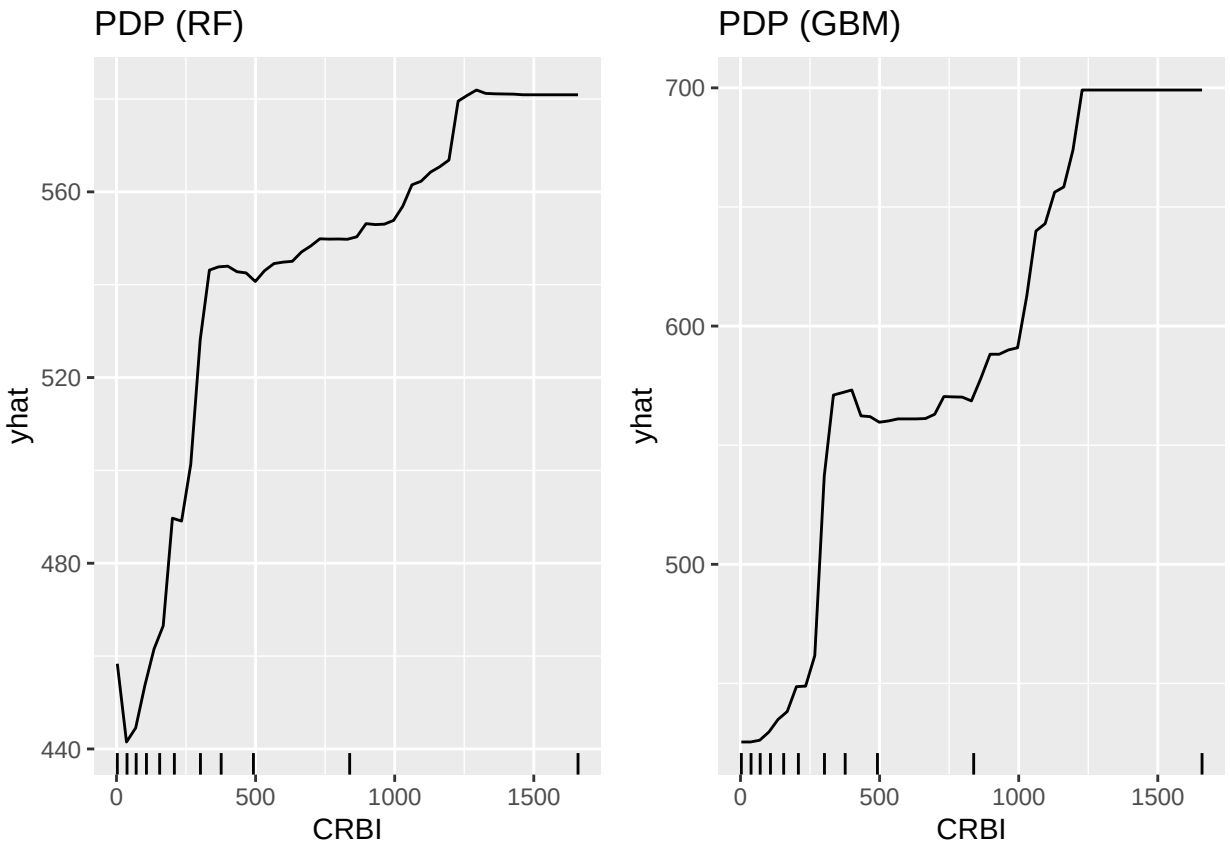
```
## CWalks         CWalks  5.50106649
## AtBat           AtBat  4.60917227
## Runs             Runs  4.60204318
## CRuns           CRuns  3.57918428
## HmRun           HmRun  1.48304732
## Assists       Assists  0.66269596
## Errors         Errors  0.64168589
## DivisionW   DivisionW  0.28492964
## LeagueN       LeagueN  0.15404472
## NewLeagueN NewLeagueN  0.06555822
```

Partial dependence plots

After the most relevant variables have been identified, the next step is to attempt to understand how the response variable changes based on these variables. For this we can use partial dependence plots (PDPs).

PDPs plot the change in the average predicted value as specified feature(s) vary over their marginal distribution. The PDP plot below displays the average change in predicted `Salary` as we vary `CRBI` while holding all other variables constant. This is done by holding all variables constant for each observation in our training data set but then apply the unique values of `CRBI` for each observation. We then average the `Salary` across all the observations.

```
p1 <- partial(rf.fit, pred.var = "CRBI",
              plot = TRUE, rug = TRUE,
              plot.engine = "ggplot") + ggtitle("PDP (RF)")
p2 <- partial(gbm.fit, pred.var = "CRBI",
              plot = TRUE, rug = TRUE,
              plot.engine = "ggplot") + ggtitle("PDP (GBM)")
gridExtra::grid.arrange(p1, p2, nrow = 1)
```



Classification

We use the Pima Indians Diabetes Database for illustration. The data contain 768 observations and 9 variables. The outcome is a binary variable `diabetes`.

```
data(PimaIndiansDiabetes)
dat <- PimaIndiansDiabetes
dat$diabetes <- factor(dat$diabetes, c("pos", "neg"))

set.seed(2022)
data_split <- initial_split(dat, prop = 2/3)

training_data_1 <- training(data_split)
testing_data_1 <- testing(data_split)
```

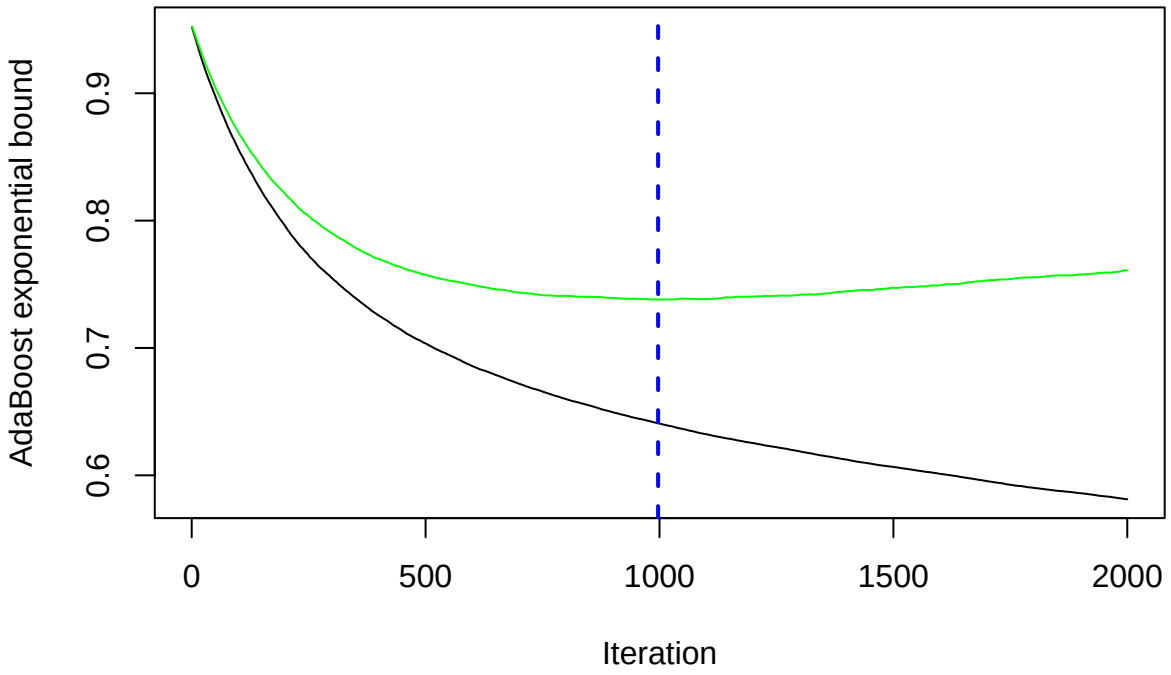
Bagging and random forests

```
set.seed(1)
bagging <- randomForest(diabetes ~ . ,
                        training_data_1,
                        mtry = 8)

set.seed(1)
rf <- randomForest(diabetes ~ . ,
                   training_data_1,
                   mtry = 3)
```

```
set.seed(1)
rf2 <- ranger(diabetes ~ . ,
              training_data_1,
              mtry = 3,
              probability = TRUE)

rf.pred <- predict(rf, newdata = testing_data_1, type = "prob")[,1]
rf2.pred <- predict(rf2, data = testing_data_1, type = "response")$predictions[,1]
```

Boosting

```
training_data_2 <- training_data_1
training_data_2$diabetes <- as.numeric(training_data_1$diabetes == "pos")

set.seed(1)
bst <- gbm(diabetes ~ . ,
           training_data_2,
           distribution = "adaboost",
           n.trees = 2000,
           interaction.depth = 2,
           shrinkage = 0.005,
           cv.folds = 10,
           n.cores = 2)

gbm.perf(bst, method = "cv")
```

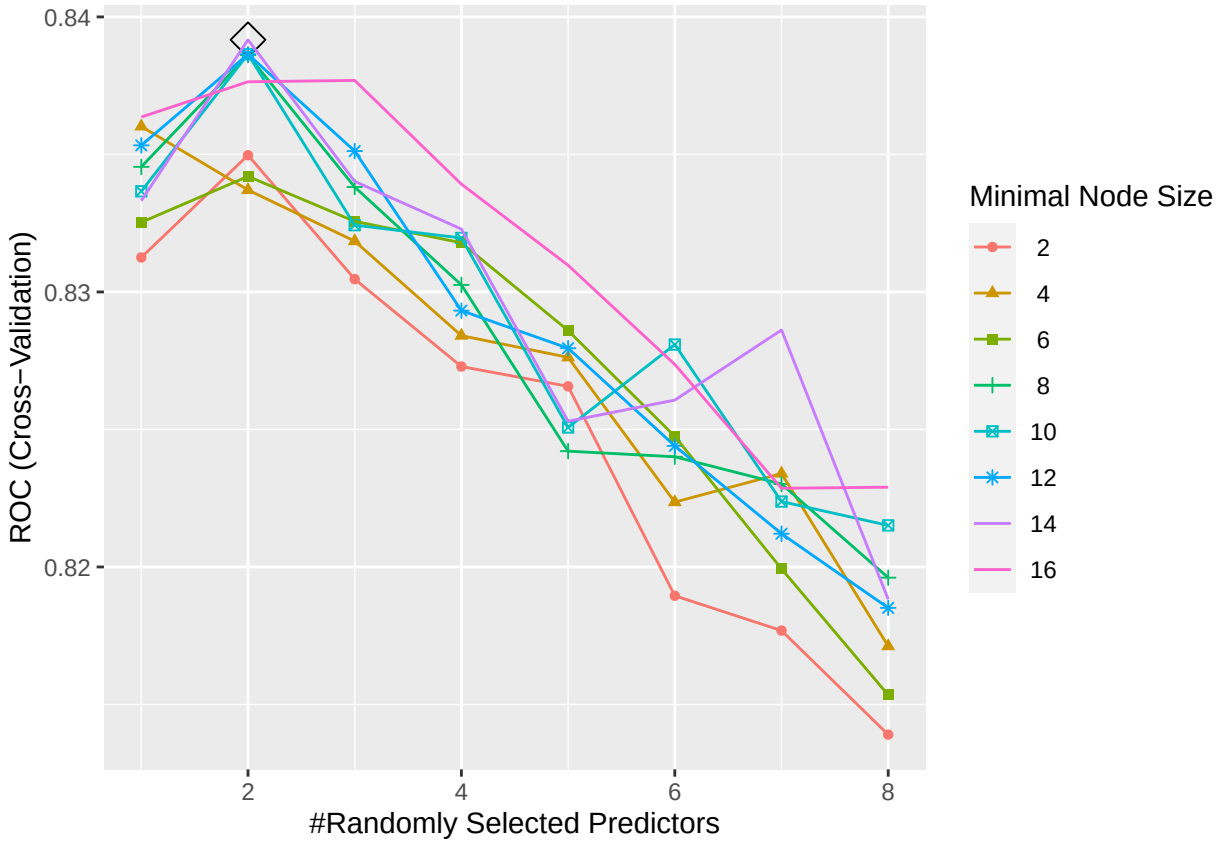



```
## [1] 997
```

Grid search using caret

Random forests

```
ctrl <- trainControl(method = "cv",
                     classProbs = TRUE,
                     summaryFunction = twoClassSummary)

rf.grid <- expand.grid(mtry = 1:8,
                       splitrule = "gini",
                       min.node.size = seq(from = 2, to = 16, by = 2))
set.seed(1)
rf.fit <- train(diabetes ~ . ,
                training_data_1,
                method = "ranger",
                tuneGrid = rf.grid,
                metric = "ROC",
                trControl = ctrl)

ggplot(rf.fit, highlight = TRUE)
```

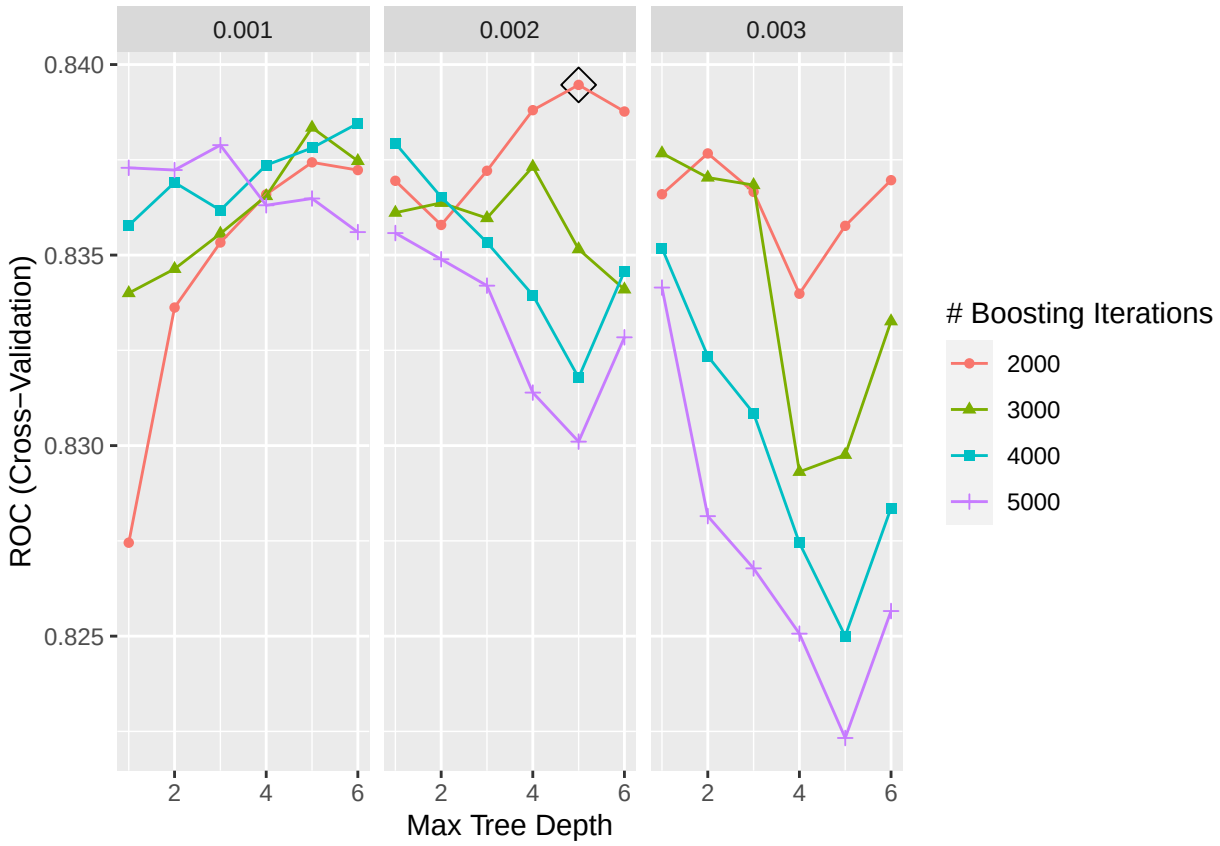



```
rf.pred <- predict(rf.fit, newdata = testing_data_1, type = "prob")[,1]
```

AdaBoost

```
gbmA.grid <- expand.grid(n.trees = c(2000,3000,4000,5000),
                         interaction.depth = 1:6,
```

```
                        shrinkage = c(0.001, 0.002, 0.003),
                        n.minobsinnode = 1)
set.seed(1)
gbmA.fit <- train(diabetes ~ . ,
                  training_data_1,
                  tuneGrid = gbmA.grid,
                  trControl = ctrl,
                  method = "gbm",
                  distribution = "adaboost",
                  metric = "ROC",
                  verbose = FALSE)

ggplot(gbmA.fit, highlight = TRUE)
```




```
gbmA.pred <- predict(gbmA.fit, newdata = testing_data_1, type = "prob")[,1]
```

```
resamp <- resamples(list(rf = rf.fit,
                         gbmA = gbmA.fit))
summary(resamp)
```

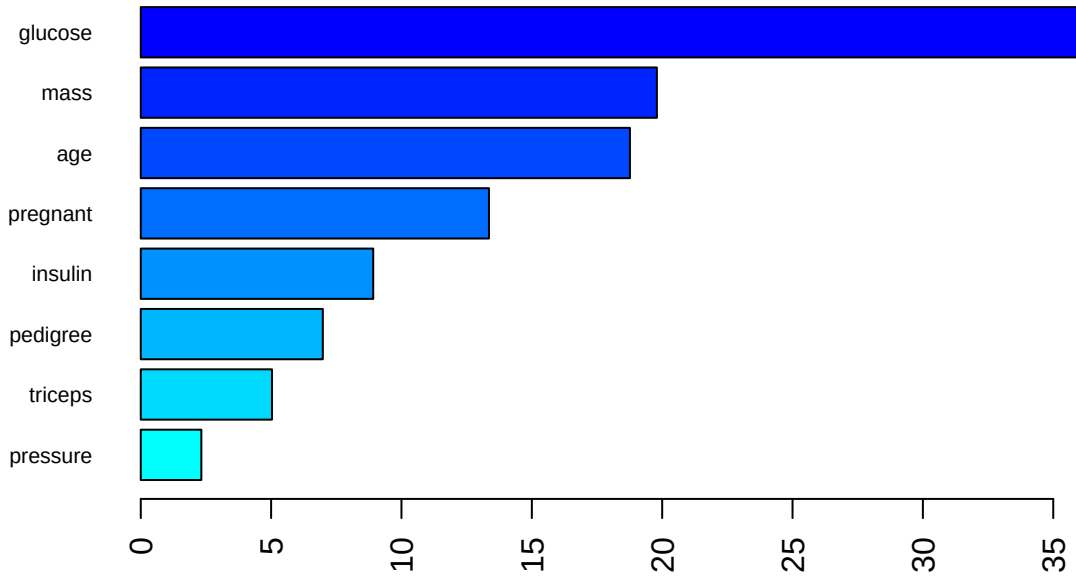
```
##
## Call:
## summary.resamples(object = resamp)
##
## Models: rf, gbmA
## Number of resamples: 10
##
```

```
## ROC
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## rf   0.7255892 0.8130570 0.8484106 0.8391670 0.8754646 0.8954248    0
## gbmA 0.7104377 0.8253119 0.8488562 0.8394656 0.8711684 0.8956229    0
##
## Sens
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## rf   0.3333333 0.5138889 0.5996732 0.5846405 0.7034314 0.7222222    0
## gbmA 0.3888889 0.4583333 0.5996732 0.5846405 0.7034314 0.7777778    0
##
## Spec
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## rf   0.7878788 0.8594029 0.8787879 0.8711230 0.8823529 0.9117647    0
## gbmA 0.7272727 0.8529412 0.8957219 0.8741533 0.9090909 0.9411765    0
```

Global interpretation

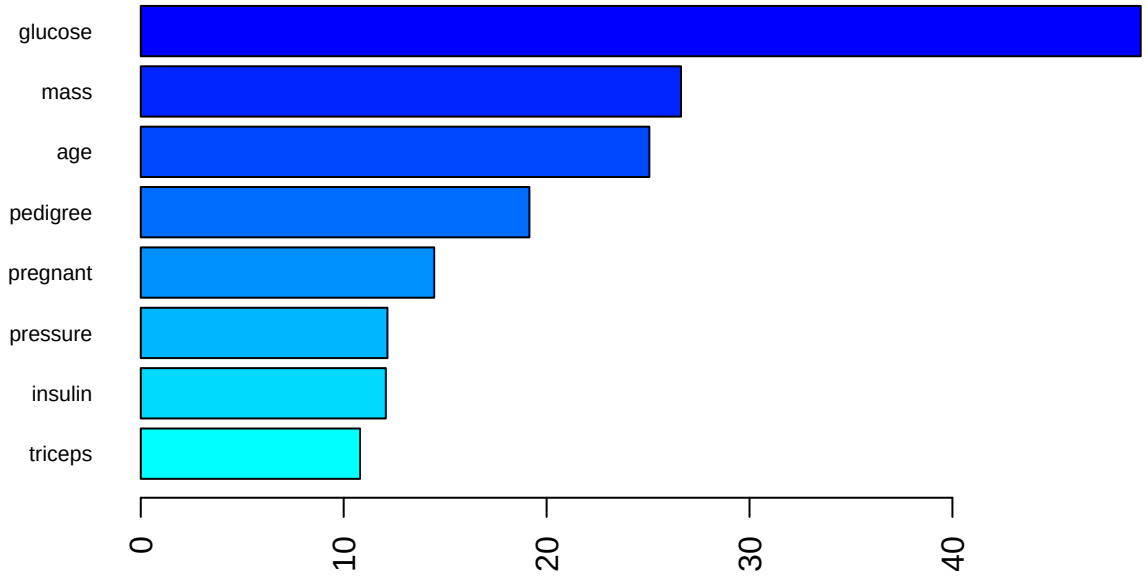
Variable importance

```
set.seed(1)
rf2.final.per <- ranger(diabetes ~ . ,
                        training_data_1,
                        mtry = rf.fit$bestTune[[1]],
                        min.node.size = rf.fit$bestTune[[3]],
                        splitrule = "gini",
                        importance = "permutation",
                        scale.permutation.importance = TRUE)

barplot(sort(ranger::importance(rf2.final.per), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("cyan","blue"))(8))
```

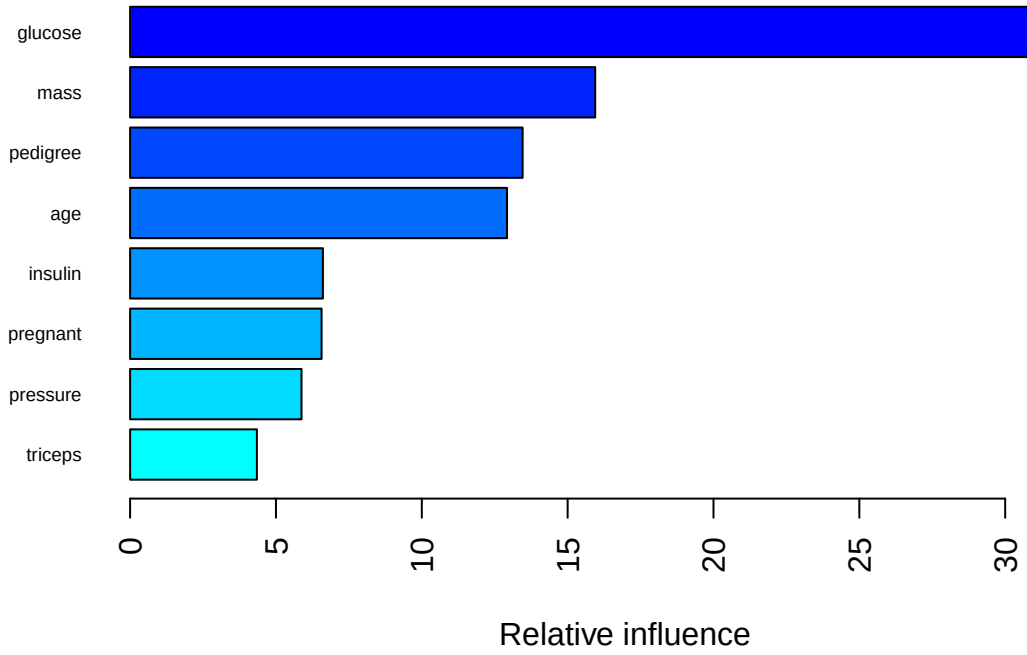



```
set.seed(1)
rf2.final.imp <- ranger(diabetes ~ . , training_data_1,
                        mtry = rf.fit$bestTune[[1]],
                        splitrule = "gini",
```

```
                        min.node.size = rf.fit$bestTune[[3]],
                        importance = "impurity")

barplot(sort(ranger::importance(rf2.final.imp), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("cyan","blue"))(8))
```




```
summary(gbmA.fit$finalModel, las = 2, cBars = 19, cex.names = 0.6)
```

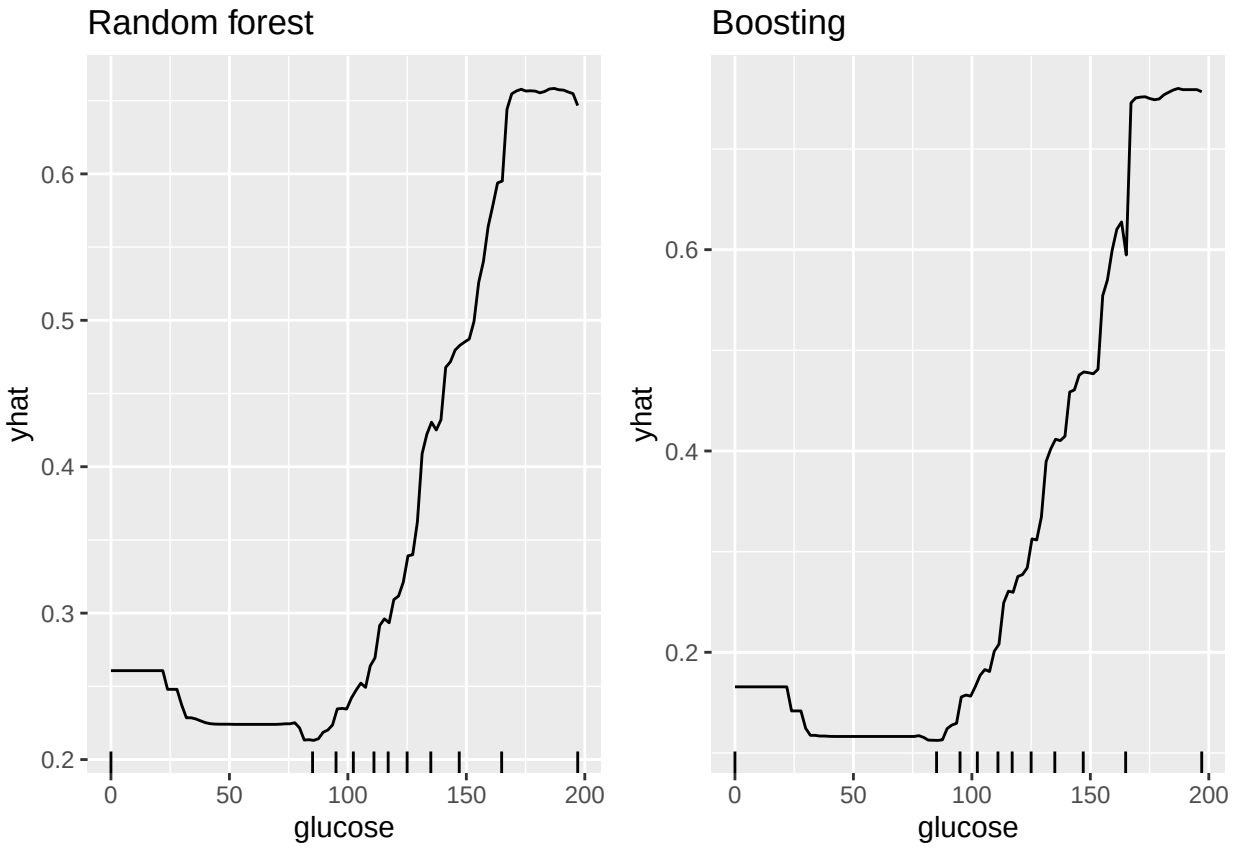



```
##                var   rel.inf
## glucose   glucose 34.281042
## mass         mass 15.945018
## pedigree pedigree 13.456587
## age           age 12.921594
## insulin   insulin  6.609560
```

```
## pregnant pregnant  6.563963
## pressure pressure  5.875685
## triceps   triceps  4.346551
```

PDP

```
pdp.rf <- rf.fit %>%
  pdp::partial(pred.var = "glucose",
          grid.resolution = 100,
          prob = TRUE) %>%
  autoplot(rug = TRUE, train = training_data_1) +
  ggtitle("Random forest")

pdp.gbm <- gbmA.fit %>%
  pdp::partial(pred.var = "glucose",
          grid.resolution = 100,
          prob = TRUE) %>%
  autoplot(rug = TRUE, train = training_data_1) +
  ggtitle("Boosting")

gridExtra::grid.arrange(pdp.rf, pdp.gbm, nrow = 1)
```




```
roc.rf <- roc(testing_data_1$diabetes, rf.pred)
roc.gbmA <- roc(testing_data_1$diabetes, gbmA.pred)

plot(roc.rf, col = 1)
plot(roc.gbmA, add = TRUE, col = 2)
```

```
auc <- c(roc.rf$auc[1], roc.gbmA$auc[1])

modelNames <- c("RF","Adaboost")
legend("bottomright", legend = paste0(modelNames, ": ", round(auc,3)),
       col = 1:2, lwd = 2)
```