NBA 4920/6921 Lecture 18

Ensemble Methods: Random Forest Application

Murat Unal

11/02/2021

```
rm(list=ls())
options(digits = 3, scipen = 999)
library(tidyverse)
library(ISLR)
library(lmtest)
library(sandwich)
library(jtools)
library(caret)
library(rpart)
library(rpart.plot)
library(ROCR)
library(ipred)
library(vip)
library(randomForest)
library(gbm)
library(ranger)
set.seed(2)
```

```r
Hitters <- ISLR::Hitters
Hitters <- na.omit(Hitters)
```

```r
train = sample(1:nrow(Hitters), 0.7*nrow(Hitters))
rmse.tree <- 323
rmse.bag <- 269
```

Random Forest

- The default random forest performs 500 trees and features/3 randomly selected predictor variables at each split.

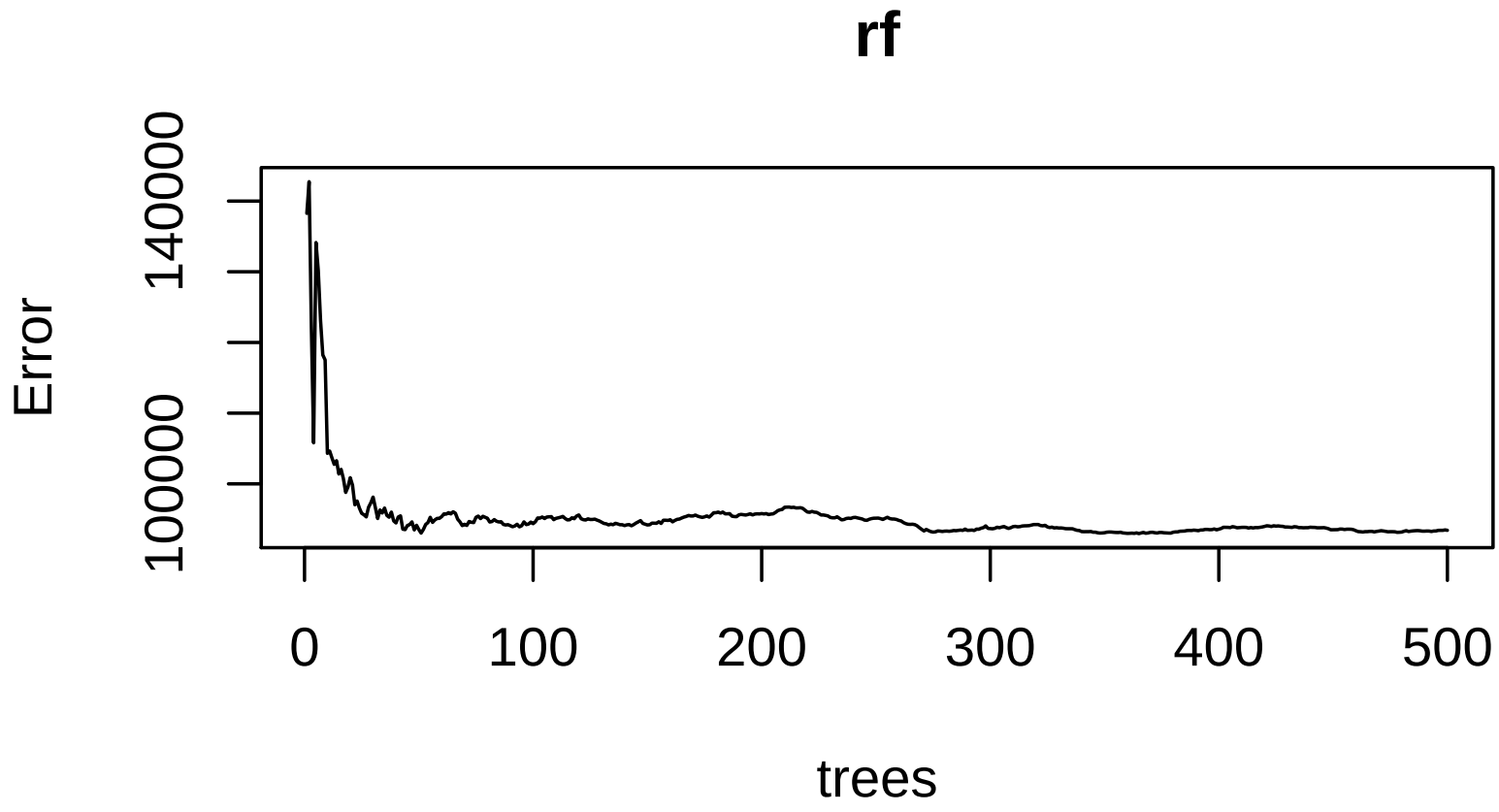
```
rf <-randomForest(Salary ~., data=Hitters[train,])
rf
```

```
Call:
 randomForest(formula = Salary ~ ., data = Hitters[train, ]
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 6

          Mean of squared residuals: 93424
                    % Var explained: 57.4
```

- Our error rate stabilizes with around 300 trees

```
plot(rf)
```



The plotted error rate above is based on the OOB sample error and can be accessed directly at `rf$mse`. Thus, we can find which number of trees providing the lowest error rate.

```
# number of trees with lowest MSE
which.min(rf$mse)
```

```
[1] 365
```

```
# RMSE of this optimal random forest
sqrt(rf$mse[which.min(rf$mse)])
```

```
[1] 305
```

Tuning

- The following hyperparameters should be tuned for optimal ranfom forest performance
- `ntree`: Number of trees. We want enough trees to stabalize the error but using too many trees is unncessarily inefficient, especially when using large data sets.
- `mtry`: The number of variables to randomly sample as candidates at each split. When `mtry`=p the model equates to bagging. When `mtry`=1 the split variable is completely random, so all variables get a chance but can lead to overly biased results.

- `sampsize`: The number of samples to train on. The default sampling scheme for random forests is bootstrapping where 100% of the observations are sampled with replacement. The sample size parameter determines how many observations are drawn for the training of each tree. Assess 3–4 values of sample sizes ranging from 50%–100%.
- `nodesize`: minimum number of samples within the terminal nodes. Controls the complexity of the trees. Smaller node size allows for deeper, more complex trees and smaller node results in shallower trees.
- `maxnodes`: maximum number of terminal nodes. Another way to control the complexity of the trees. More nodes equates to deeper, more complex trees and less nodes result in shallower trees

Full grid search with `ranger`

- To perform a larger grid search across several hyperparameters we'll need to create a grid and loop through each hyperparameter combination and evaluate the model.

- To perform the grid search, first we want to construct our grid of hyperparameters. We're going to search across 96 different models with varying mtry, minimum node size, and sample size.

```r
# hyperparameter grid search
hyper_grid <- expand.grid(
  mtry        = seq(5, 10, by = 1),
  node_size   = seq(2, 8, by = 2),
  sample_size = c(.5, .6, .70, .80),
  OOB_RMSE    = 0
)

# total number of combinations
nrow(hyper_grid)
```

```
[1] 96
```

- We loop through each hyperparameter combination and apply 300 trees since our previous examples illustrated that 300 was plenty to achieve a stable error rate

```
for(i in 1:nrow(hyper_grid)) {
  # train model
  model <- ranger(
    formula         = Salary ~ .,
    data            = Hitters[train,],
    num.trees       = 300,
    mtry            = hyper_grid$mtry[i],
    min.node.size   = hyper_grid$node_size[i],
    sample.fraction = hyper_grid$sample_size[i]  )
  # add OOB error to grid
  hyper_grid$OOB_RMSE[i] <- sqrt(model$prediction.error)
}
```

```
hyper_grid %>%
  arrange(OOB_RMSE) %>% head(10)
```

mtry	node_size	sample_size	OOB_RMSE
10	2	0.7	300
7	4	0.7	300
5	6	0.8	301
9	4	0.7	301
6	2	0.8	301
6	2	0.7	302
9	6	0.7	302
8	2	0.7	302
8	2	0.5	302
9	4	0.8	302

- The best random forest model we have found and uses `mtry = 10`, `node size` of 2 observations, and a `sample size` of 70%.

```
optimal_rf <- ranger(
    formula         = Salary ~ .,
    data            = Hitters[train,],
    num.trees       = 300,
    mtry            = 10,
    min.node.size   = 2,
    sample.fraction = .7,
    importance      = 'impurity')
```

- Once we've identified our preferred model we can use the traditional predict function to make predictions on a new data set.

```
predict_rf <- predict(optimal_rf, Hitters[-train,])$predict
rmse.rf <- sqrt(mean((Hitters[-train,"Salary"] -
                          predict_rf)^2))
rmse.rf
```

```
[1] 255
```

- We could also perform cross validation using the caret package

```
# control <- trainControl(method="oob")
control <- trainControl(method = "cv", number = 10)
tunegrid <- expand.grid(.mtry=c(1:10))
hit_rf <- train(Salary~., data=Hitters[train,],
                method="rf",
                metric="RMSE",
                tuneGrid=tunegrid,
                trControl=control,
                ntree=300)
```

```
hit_rf$bestTune
```

	mtry
4	4

```
hit_rf$finalModel
```

```
Call:
 randomForest(x = x, y = y, ntree = 300, mtry = param$mtry)
               Type of random forest: regression
                     Number of trees: 300
No. of variables tried at each split: 4

          Mean of squared residuals: 89243
                    % Var explained: 59.3
```

```
hit_rf$results
```

mtry	RMSE	Rsquared	MAE	RMSESD	RsquaredSD	MAESD
1	298	0.631	190	102.2	0.223	42.4
2	284	0.643	174	102.9	0.224	46.5
3	285	0.641	172	100.9	0.229	47.3
4	284	0.645	170	101.4	0.231	48.3
5	289	0.636	173	99.5	0.225	45.2
6	286	0.646	172	103.6	0.230	49.6
7	286	0.644	173	100.0	0.231	46.1
8	288	0.643	174	101.2	0.229	47.5
9	286	0.646	173	101.3	0.228	47.6
10	288	0.639	174	98.5	0.225	44.5

- Make predictions on the test data

```
pred.rf <- predict(hit_rf, Hitters[-train,])
rmse.rf2 <- sqrt(mean((Hitters[-train,"Salary"] -
                           pred.rf)^2))
rmse.rf2
```

```
[1] 263
```

- Compare prediction performance

```
rmse.tree
```

```
[1] 323
```

```
rmse.bag
```

```
[1] 269
```

```
rmse.rf
```

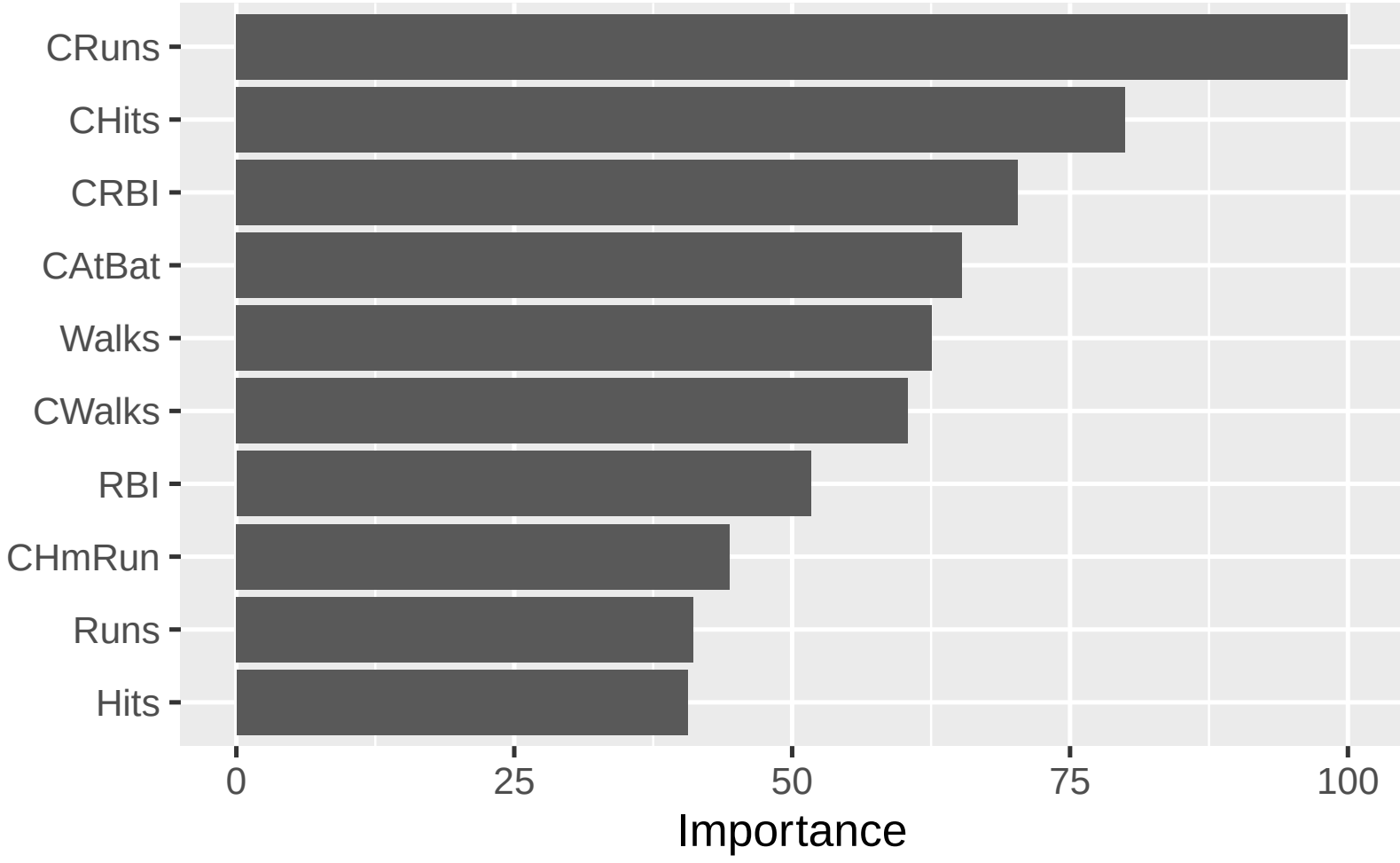
```
[1] 255
```

```
rmse.rf2
```

```
[1] 263
```

- Variable importance plot

```
#vip(optimal_rf)
vip(hit_rf)
```



Exercise

- Implement a single tree with pruning, bagging and random forest on the training data and predict crime rates in the test data.

```
data_test <- read.csv("boston_test.csv")
data_train <- read.csv("boston_train.csv")
```

- Fit single tree

```
bos_tree = rpart(crim ~ .,data = data_train,
method="anova")
bos_tree
```

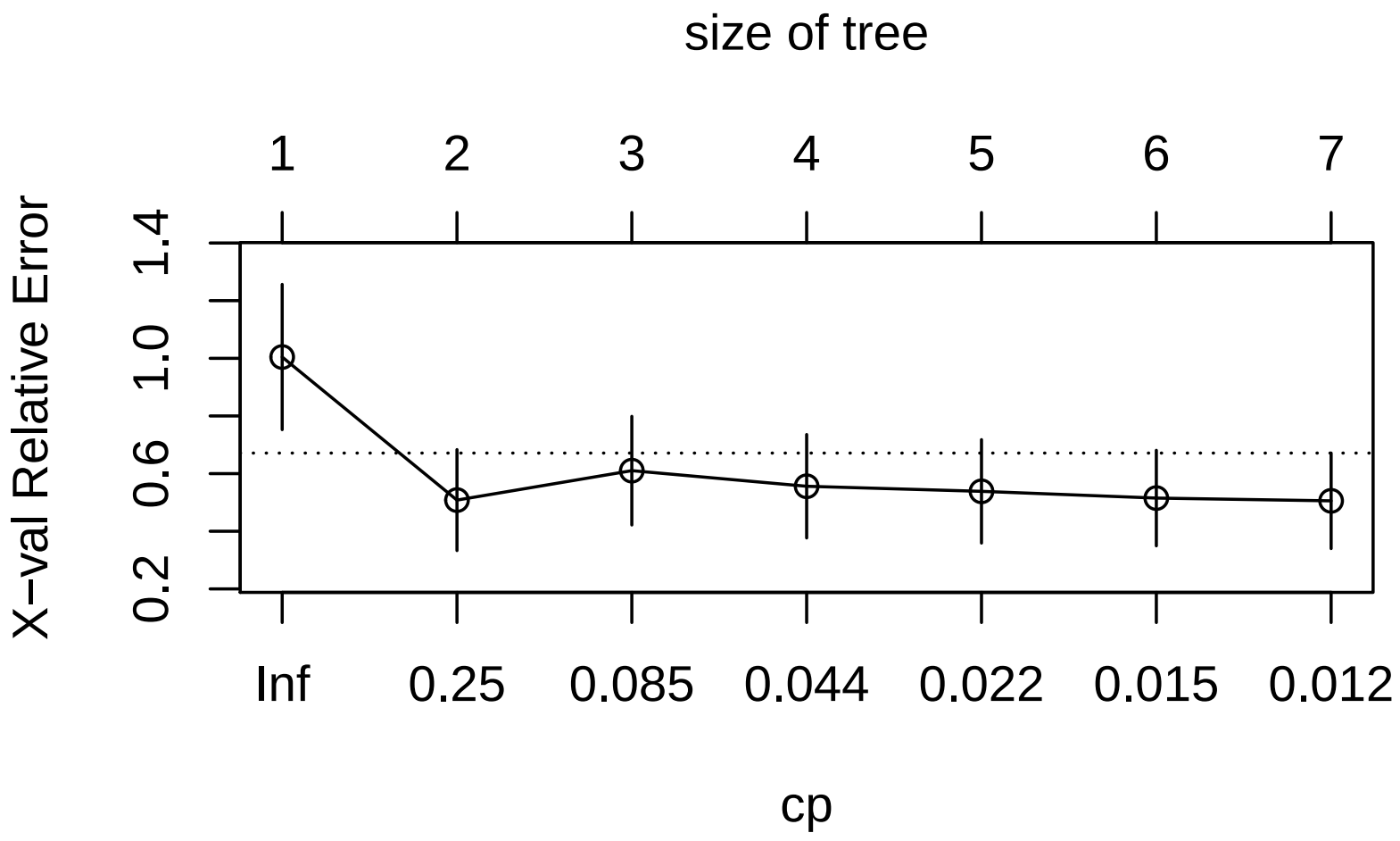
```
n= 405

node), split, n, deviance, yval
      * denotes terminal node

 1) root 405 21800.0  3.37
   2) rad< 16 303    91.0  0.36 *
   3) rad>=16 102 10800.0 12.30
     6) medv>=7.45 94  5400.0 10.80
      12) dis>=1.87 56   750.0  7.82
        24) medv>=12.6 45   378.0  6.66 *
        25) medv< 12.6 11    64.3 12.60 *
      13) dis< 1.87 38  3410.0 15.20
        26) black>=300 26  1010.0 12.30
          52) medv>=12 17   352.0  9.67 *
```

- Plot cp

```
plotcp(bos_tree)
```



- Prune tree

```
bos_pruned <- prune(bos_tree, cp=0.25)
bos_pruned
```

```
n= 405

node), split, n, deviance, yval
      * denotes terminal node

1) root 405 21800  3.37
  2) rad< 16 303    91  0.36 *
  3) rad>=16 102 10800 12.30 *
```

Make predictions on the test data

```
pred.bos.pruned <- predict(bos_pruned, data_test)
rmse.pruned <- sqrt(mean( (data_test$crim-pred.bos.pruned)
rmse.pruned
```

```
[1] 10.7
```

- Apply bagging

```
bos_bag <- bagging(crim ~ .,data = data_train,
nbagg=100,coob=TRUE,
control=rpart.control(cp=0))
bos_bag
```

```
Bagging regression trees with 100 bootstrap replications

Call: bagging.data.frame(formula = crim ~ ., data = data_tr
    coob = TRUE, control = rpart.control(cp = 0))

Out-of-bag estimate of root mean squared error:  4.5
```

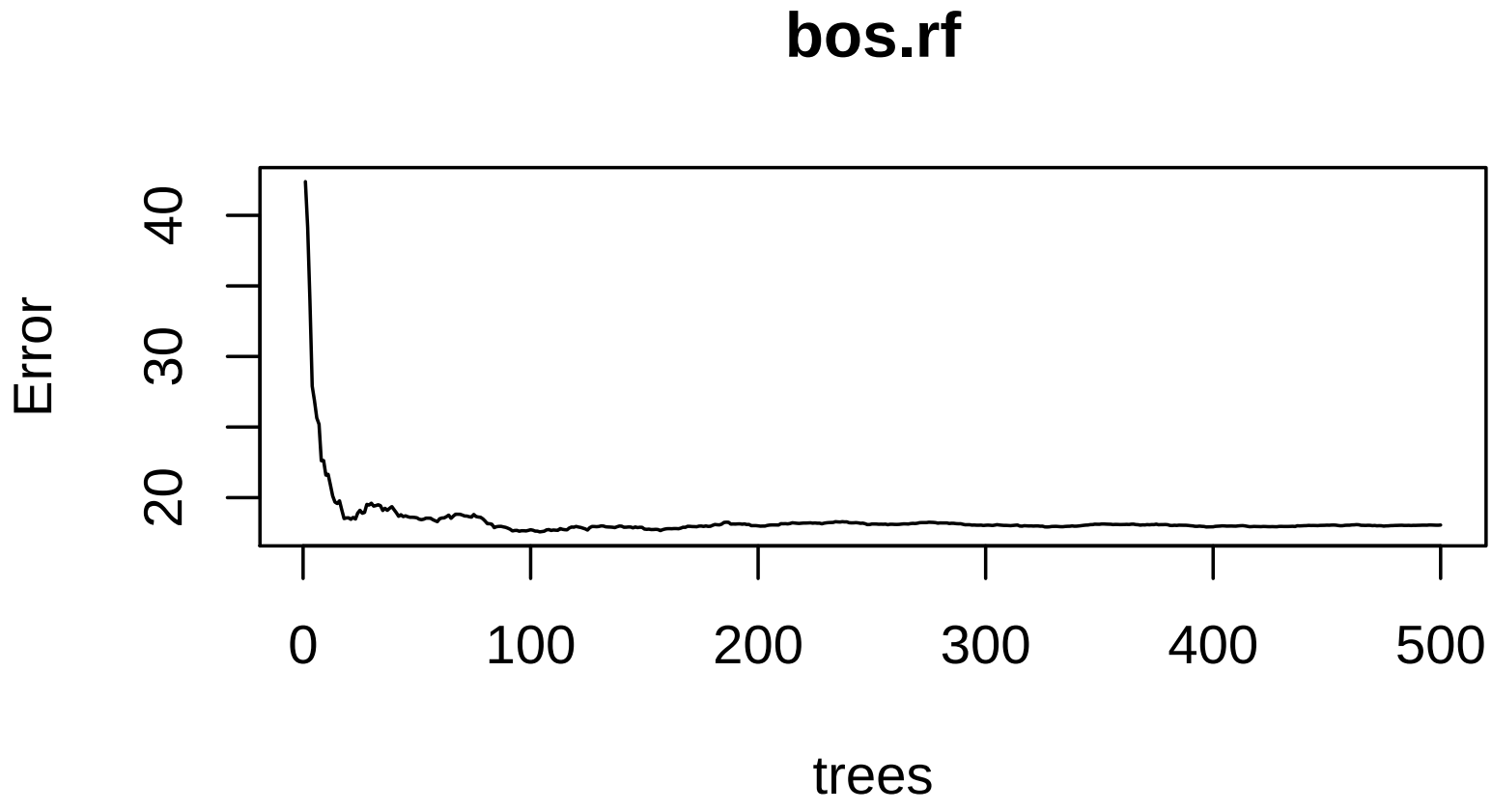
- Make predictions on the test data

```
pred.bos_bag <- predict(bos_bag, data_test)
rmse.bos_bag <- sqrt(mean((data_test$crim-pred.bos_bag)^2))
rmse.bos_bag
```

```
[1] 9.65
```

- Apply random forest

```
bos.rf <-randomForest(crim ~., data=data_train)
plot(bos.rf)
```



- Define tuning grid

```
hyper_grid <- expand.grid(
  mtry        = seq(2, 12, by = 2),
  node_size   = seq(2, 8, by = 2),
  sample_size = c(.5, .70, .80),
  OOB_RMSE    = 0
)
```

_ Apply tuning

```
for(i in 1:nrow(hyper_grid)) {
  # train model
  model <- ranger(
    formula         = crim ~ .,
    data            = data_train,
    num.trees       = 300,
    mtry            = hyper_grid$mtry[i],
    min.node.size   = hyper_grid$node_size[i],
    sample.fraction = hyper_grid$sample_size[i]  )
  # add OOB error to grid
  hyper_grid$OOB_RMSE[i] <- sqrt(model$prediction.error)
}
```

- Show tuning results

```
hyper_grid %>%
  arrange(OOB_RMSE) %>% head(10)
```

mtry	node_size	sample_size	OOB_RMSE
4	6	0.7	4.19
2	4	0.8	4.19
4	2	0.7	4.19
6	6	0.7	4.20
6	6	0.5	4.24
4	4	0.8	4.24
4	4	0.7	4.24
4	8	0.8	4.25
2	2	0.7	4.25
4	8	0.7	4.25

- The best random forest model we have found and uses `mtry = 4`, `node size` of 6 observations, and a `sample size` of 70%.

```
optimal_rf <- ranger(
    formula         = crim ~ .,
    data            = data_train,
    num.trees       = 300,
    mtry            = 4,
    min.node.size   = 6,
    sample.fraction = .7,
    importance      = 'impurity')
```

- Make predictions

```
predict_rf <- predict(optimal_rf, data_test)$predictions
rmse.rf <- sqrt(mean((data_test$crim -
                          predict_rf)^2))
rmse.rf
```

```
[1] 9.9
```

- Compare performance across models

```
rmse.pruned
```

```
[1] 10.7
```

```
rmse.bos_bag
```

```
[1] 9.65
```

```
rmse.rf
```

```
[1] 9.9
```

- Variable importance plot

```
vip(optimal_rf)
```