# Lecture 14

## Bagging, random forest, and boosting

Julian Reif
Fall 2025

# RStudio setup for this lecture

- Log into RStudio on your Amazon EC2 instance
  - Use AMI `FIN550-RStudio` with IAM role `BigDataEC2Role`

```
# Enter this command via RStudio Terminal
aws s3 cp --recursive s3://bigdata-fin550-reif/lecture-14 ~/fin550/lecture-14
```

# Theory

# What's wrong with decision trees?

- Decision trees are easy to explain and interpret

- But, they have high variance, delivering poor predictive performance
  - Pruning helps, but not enough

- Today, we discuss three improvements to standard decision trees:
  1. Bagging (special case of random forest)
  2. Random forest
  3. Boosting

# Statistics review: bootstrapping

# Bootstrapping lets you create more datasets

- Increasing data is useful for both training and testing

- How can you create new datasets without searching for more data?

- Bootstrap: draw randomly **with replacement** from your original dataset
  - Bootstrapped datasets all come from the same data generating process

# Bootstrapping example

```
set.seed(2)
df <- data.frame(X1 = c(1,2,3,4,5,6,7))

df$X1_boot1 <- sample(df$X1, replace = T)
df$X1_boot2 <- sample(df$X1, replace = T)
df$X1_boot3 <- sample(df$X1, replace = T)
df
```

```
#   X1 X1_boot1 X1_boot2 X1_boot3
# 1  1         5         4         6
# 2  2         7         5         2
# 3  3         6         1         3
# 4  4         6         2         7
# 5  5         1         3         7
# 6  6         5         1         7
# 7  7         1         3         1
```

# Bootstrapping properties

- Bootstrapped datasets are created by drawing **with replacement**
  - Some observations get drawn more than once
  - Some observations are not drawn at all

- What is the probability that an observation will not be in a bootstrapped dataset?

$$(1 - 1/n)^n \to 1/e \approx 0.368 \text{ when } n \text{ is large}$$

- Thus, about 1/3 of original observations will not be in a bootstrapped dataset

# Bagging and random forests

# Bootstrap aggregation ("bagging")

- Goal: reduce the variance of decision tree predictions

- If we have more datasets, we can reduce variance by averaging across them
  - Intuition: estimate of an average is better when $n = 1000$ than when $n = 100$

- Bagging uses bootstrap to create more datasets
  - Grow a separate tree on each dataset

# Algorithm for bagging

1. Use bootstrap to create $B$ datasets

2. For each dataset $b = 1, 2, \ldots B$, grow a full tree (no pruning)
   - Denote each tree's prediction as $\hat{f}^b(x)$

3. Calculate the average prediction across all the regression trees

$$\hat{f}^b_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^b(x)$$

   - For classification tree: take a majority vote (i.e., mode of all $\hat{f}^b(x)$))

- Note: $B = 100$ is a good default, but it is fine to use larger values

# Out-of-bag error estimation

- Recall: bootstrapping draws with replacement from the data
  - On average, only 2/3 of the observations are used for tree $b$

- For each observation, form prediction **using trees grown without that observation**
  - On average, there are $B/3$ trees that didn't use this observation

- Take the average (or mode) of those $\approx B/3$ predictions

- "Out-of-bag" error is the mean-squared prediction error from these predictions

# Random forests

- Random forests are identical to bagging, except for one difference
- When considering a split, use **random** sample of $m$ predictors instead of all $p$ predictors
- Example: there are 10 predictors, $X_1 \ldots X_{10}$, and we set $m = 2$
  - Bagging: for all splits, search for optimal $X_j$ and split point $s$ among all 10 predictors
  - Random forest: consider $X_1$ and $X_7$ for first split, $X_4$ and $X_5$ for second split, etc.
- Typical default: $m = \sqrt{p}$
- Note: when $m = p$, random forest is same as bagging

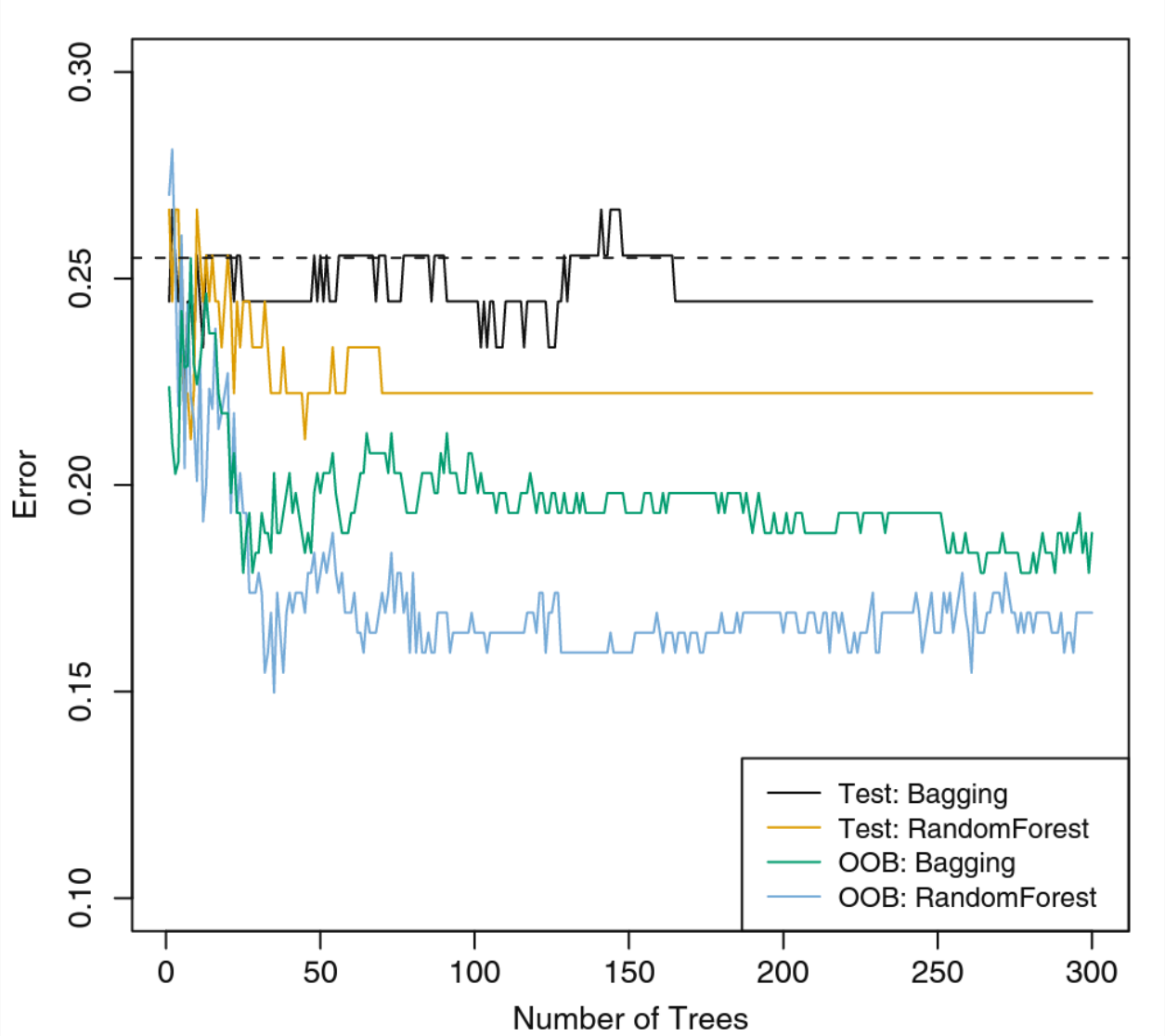
# Why limit the set of predictors??

- Without limiting the predictor set, the $B$ trees used in bagging will be very similar

- Randomly dropping predictors limits the correlation among the $B$ trees
  - Reducing the correlation reduces the variance of the prediction

- Random forest is "bagging + decorrelation"

# Random forest generally outperforms bagging



# Boosting

# Boosting builds trees incrementally

- Build a series of **small**, incremental trees
  - Each tree is grown on a **dataset of residuals** from previous tree: $r_i = y_i - \hat{y}_i$
  - No bootstrapping

- Boosting depends on three tuning parameters:
  1. The number of trees, $B$
     - Chosen by cross-validation (note: very different from random forest!)
  2. The number of splits, $d$, in each incremental tree
     - Typically, we set $d = 1$ and consider a random sample of predictors
  3. The shrinkage parameter, $\lambda$
     - Controls the rate of learning from each incremental tree
     - Typically, $\lambda = 0.01$ or $\lambda = 0.001$

# Algorithm for boosting

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all $i$

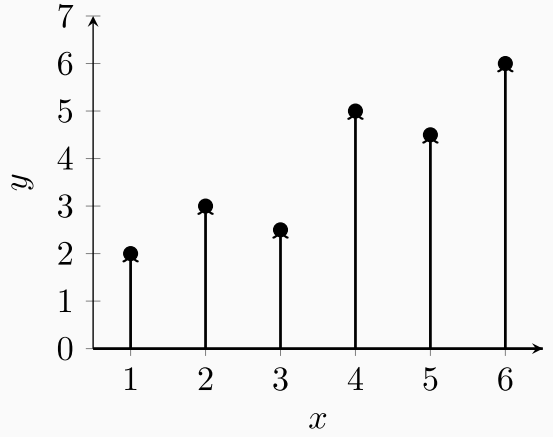
2. For $b = 1, 2, \ldots, B$:
   - Fit a tree $\hat{f}^b$ with $d$ splits to the data $(X, r)$
   - Update $\hat{f}$ using the learning parameter: $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$
   - Update the residuals: $r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$

3. Calculate the boosted prediction across all the incremental trees:

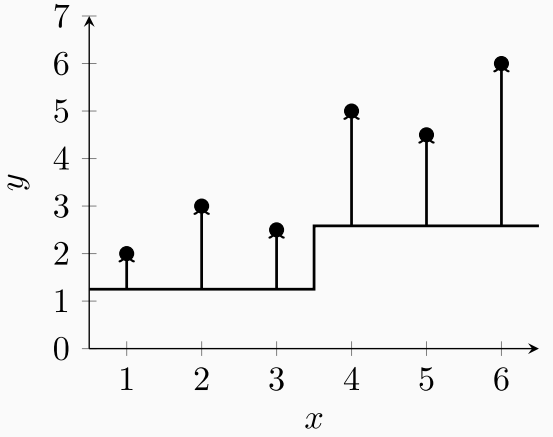
$$\hat{f}_{boost}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

# Example: Boosting predictions for B=3, lambda = 0.5, d = 1

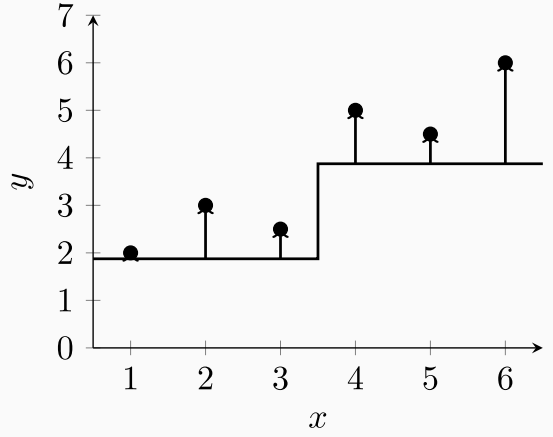
**(1) Init (MSE = 16.8)**



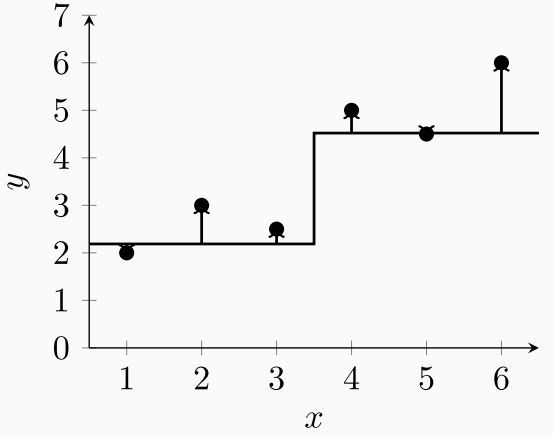

**(2) Split at $x = 3.5$ (MSE = 4.4)**



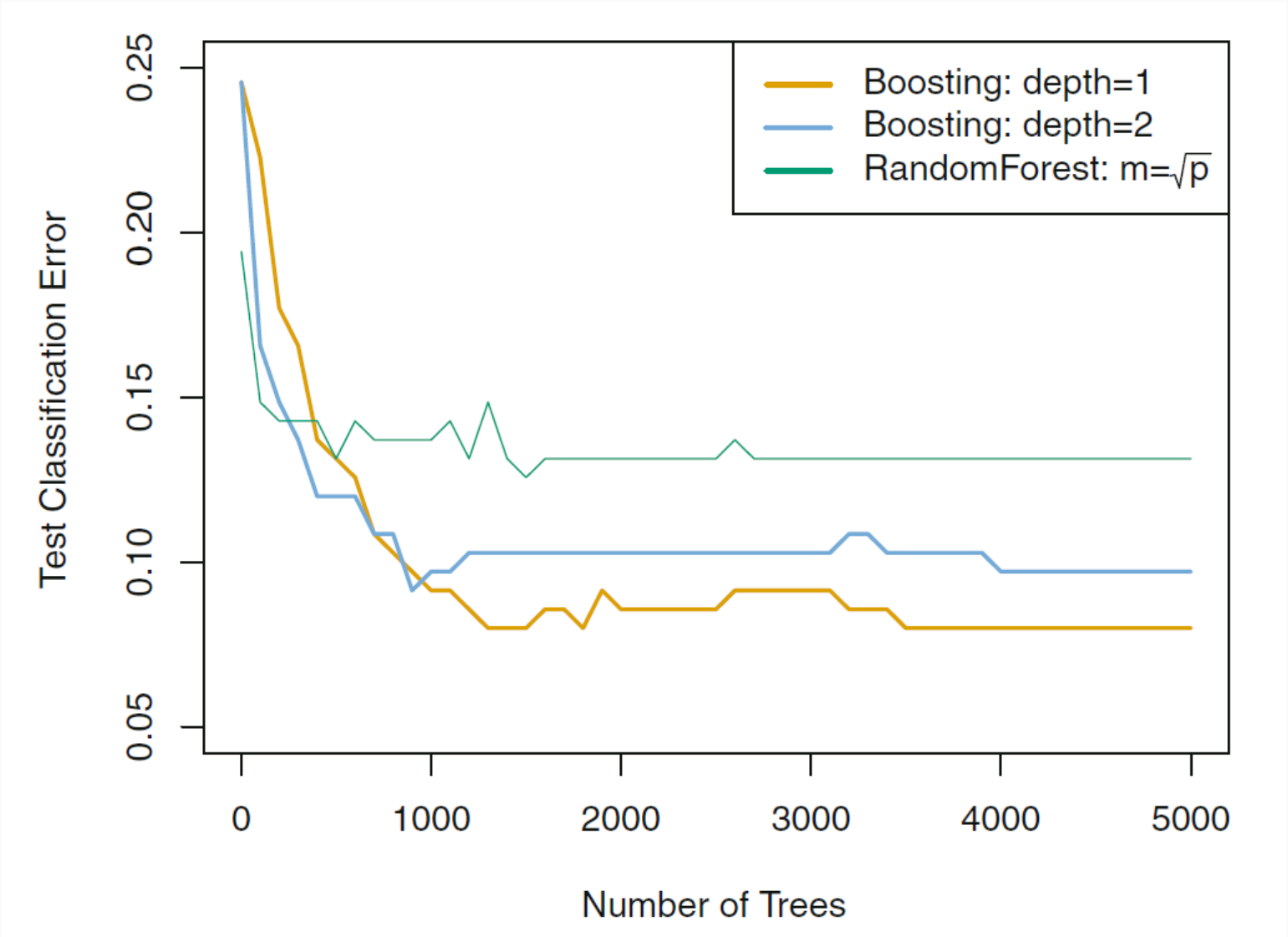

**(3) Split at $x = 3.5$ again (MSE = 1.3)**




**(4) Split at $x = 3.5$ again (MSE = 0.54)**



# Boosting can outperform random forest



# Bagging and random forest in R

# Try it: load and inspect the Boston housing dataset

```
library(tidyverse)
library(ggplot2)

# Housing information for 506 areas around Boston
housing <- read_csv("lecture-14-housing.csv")
nrow(housing)
ncol(housing)

# The data are complete, so no need to remove observations
sum(!complete.cases(housing))
```

```
# [1] 506
# [1] 13
# [1] 0
```

# Variables come from the 1970 Census

| Variable name | Definition |
|---|---|
| MEDV | Median value of owner-occupied homes in $1000's |
| CRIM | Per capita crime rate by census tract |
| ZN | Proportion of residential land zoned for lots over 25,000 sq.ft. |
| INDUS | Proportion of non-retail business acres per town |
| CHAS | Charles River dummy variable (= 1 if tract bounds river; 0 otherwise) |
| NOX | Nitric oxides concentration (parts per 10 million) |
| RM | Average number of rooms per dwelling |
| AGE | Proportion of owner-occupied units built prior to 1940 |
| DIS | Weighted distances to five Boston employment centers |
| RAD | Index of accessibility to radial highways |
| TAX | Full-value property-tax rate per $10,000 |
| PTRATIO | Pupil-teacher ratio by town |
| LSTAT | % lower status of the population |

# Run random forest using the randomForest package

- Library `randomForest`

- Key functions:
  - `myforest <- randomForest()`: grow the forest
  - `myforest$ntree`: B, number of trees
  - `myforest$oob.times`: vector reporting # of times each observation is out-of-bag
  - `myforest$mse`: vector of mean-squared error for OOB observations for $b = 1...B$
  - `importance(myforest)` and `varImpPlot(myforest)`: most important predictors

# Try it: build a model of housing value

```
summary(housing$MEDV)

# How many predictors are available?
num.p <-
```

# Build a model of housing value

```
summary(housing$MEDV)

# How many predictors are available?
num.p <- length(names(housing))-1
cat("\n")
print(paste("Number of predictors is:", num.p))
```

```
#    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#    5.00   17.02   21.20   22.53   25.00   50.00
#
# [1] "Number of predictors is: 12"
```

# Try it: estimate a tree model using bagging

```r
library(randomForest)
set.seed(1)

# Importance = T: calculate how important each variable is
# For bagging, number of splits m=p
bag_housing <- randomForest(MEDV ~ ., data = housing, mtry = num.p, importance = TRUE)
bag_housing
```

```
#
# Call:
#  randomForest(formula = MEDV ~ ., data = housing, mtry = num.p,      importance = TRUE)
#                Type of random forest: regression
#                      Number of trees: 500
# No. of variables tried at each split: 12
#
#           Mean of squared residuals: 10.59575
#                     % Var explained: 87.45
```

# Try it: mean-squared error for "out-of-bag" observations

```
# B (number of trees)
bag_housing$ntree

# Number of times each observation is out-of-bag (OOB)
head(bag_housing$oob.times)

# On average, what fraction of the time is an observation OOB?
```

# Mean-squared error for "out-of-bag" observations

```
# B (number of trees)
bag_housing$ntree

# Number of times each observation is out-of-bag (OOB)
head(bag_housing$oob.times)

# On average, what fraction of the time is an observation OOB?
cat("\n")
mean(bag_housing$oob.times)/500
1/(exp(1))
```

```
# [1] 500
# [1] 188 182 185 181 157 190
#
# [1] 0.3665968
# [1] 0.3678794
```

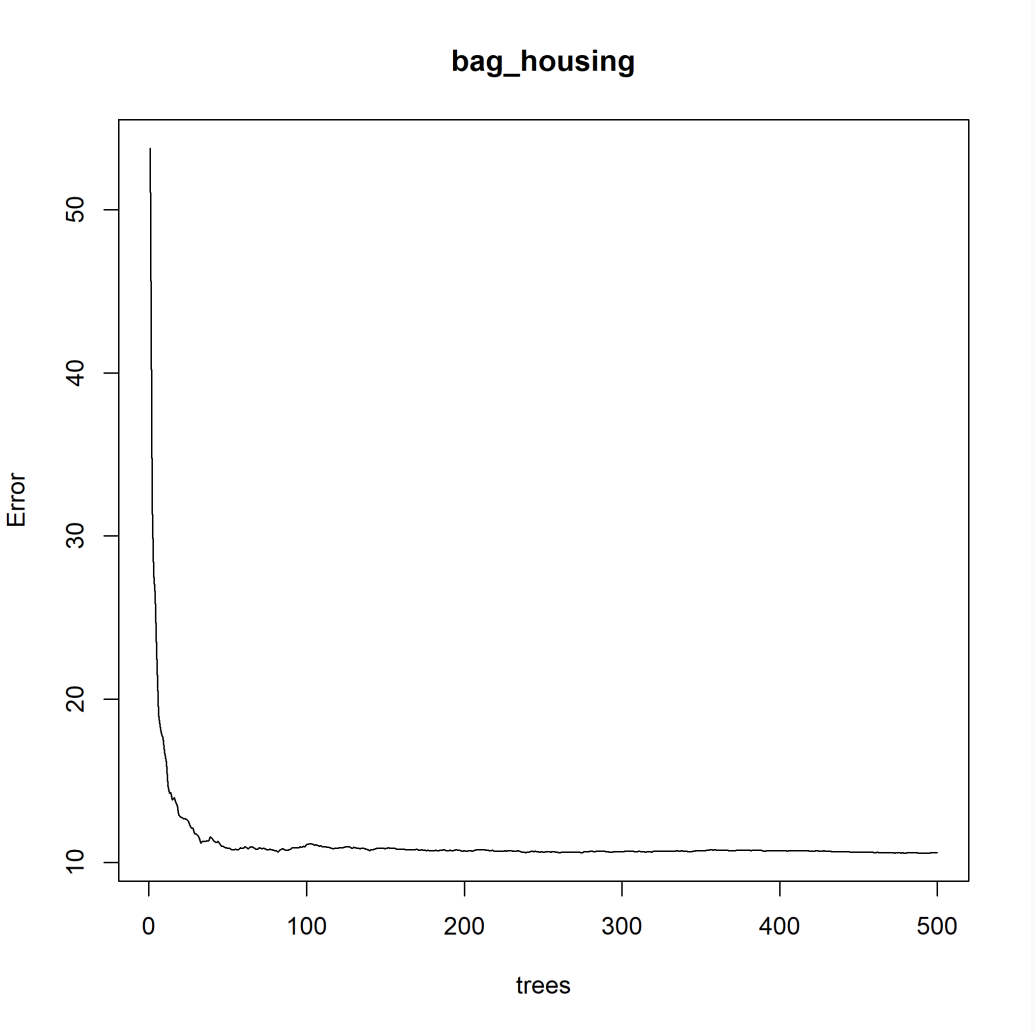
# Average OOB error decreases with the number of trees

```
# Vector of OOB error (length=B)
round(head(bag_housing$mse),1)
```

```
# [1] 53.8 32.1 27.7 26.4 22.9 19.1
```

```
# OOB error vs number of trees
plot(bag_housing)

# Same as:
#   mse <- bag_housing$mse
#   trees <- 1:bag_housing$ntree
#
#   df <- data.frame(x = trees, y=mse)
#   ggplot(df) + geom_line(aes(x=x, y=y))
```



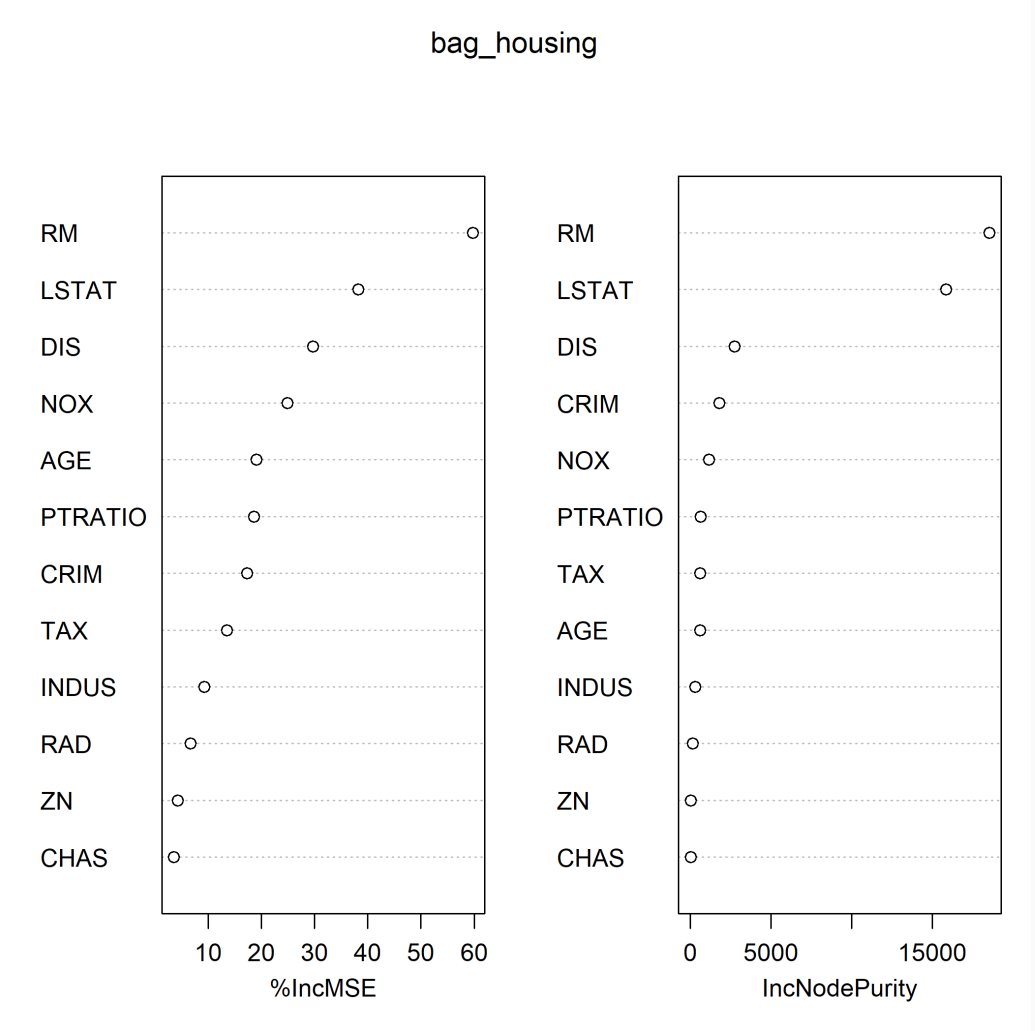
# Bagging: variable importance table

```
# Display in table form
importance(bag_housing)
```

```
#           %IncMSE IncNodePurity
# CRIM    17.320673    1781.32335
# ZN       4.265825      38.98644
# INDUS    9.246720     292.07306
# CHAS     3.555811      32.58706
# NOX     24.936067    1145.53829
# RM      59.780711   18530.57119
# AGE     19.086593     604.67534
# DIS     29.716657    2748.74198
# RAD      6.717248     158.17021
# TAX     13.555102     609.48993
# PTRATIO 18.574731     652.62634
# LSTAT   38.197511   15844.21364
```

# Bagging: variable importance plot

```
# Or, visualize using a plot
varImpPlot(bag_housing)
```



# Try it: estimate a tree model using random forest

```r
# Set number of random splits equal to a value less than 12
rf_housing <- randomForest(MEDV ~ ., data = housing, mtry = 6, importance = TRUE)
rf_housing
```

```
#
# Call:
#  randomForest(formula = MEDV ~ ., data = housing, mtry = 6, importance = TRUE)
#                Type of random forest: regression
#                      Number of trees: 500
# No. of variables tried at each split: 6
#
#           Mean of squared residuals: 9.59256
#                     % Var explained: 88.64
```

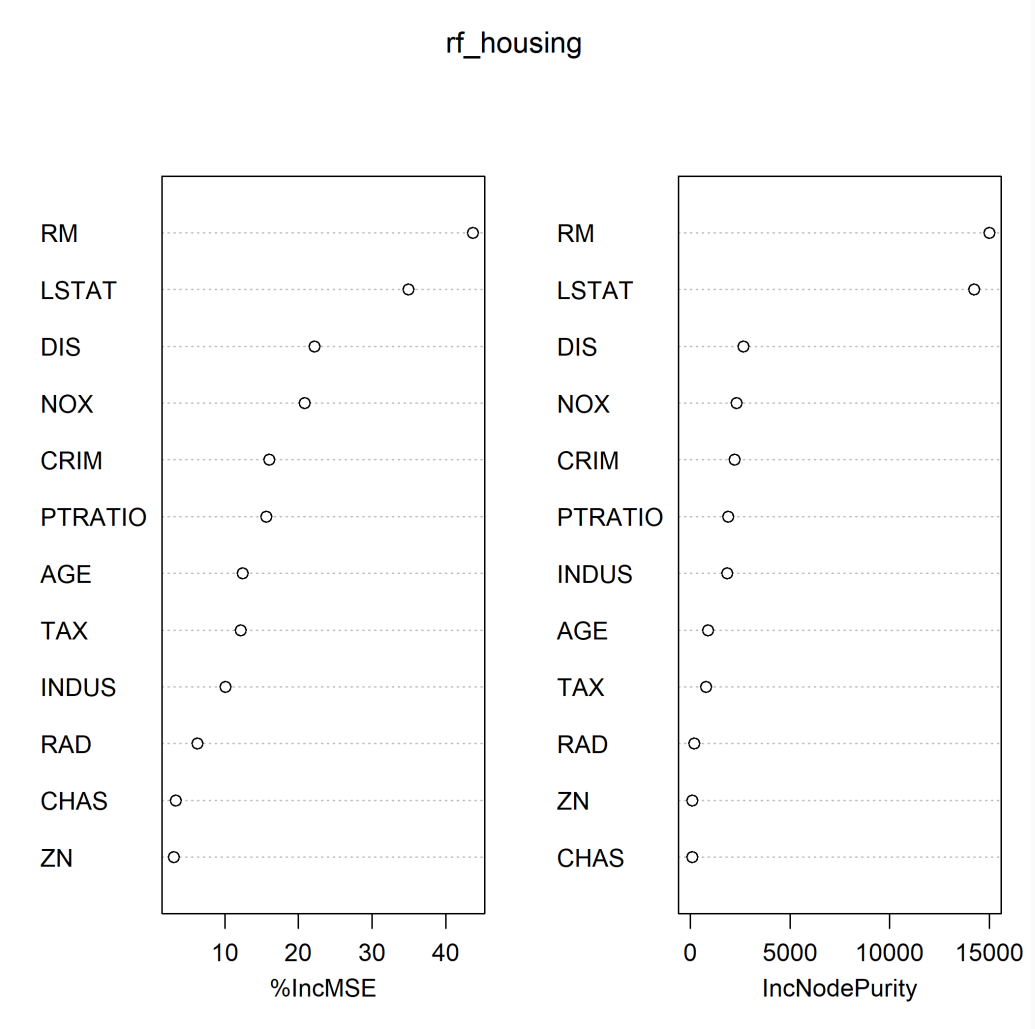
# Random forest: variable importance table

```
importance(rf_housing)
```

```
#           %IncMSE IncNodePurity
# CRIM    16.027037    2228.00483
# ZN       3.094967     102.42133
# INDUS   10.071582    1840.19628
# CHAS     3.380017      88.20743
# NOX     20.848299    2312.70179
# RM      43.661166   15004.01671
# AGE     12.403407     880.33134
# DIS     22.138316    2656.43914
# RAD      6.284539     193.00308
# TAX     12.188322     798.85712
# PTRATIO 15.650162    1895.00131
# LSTAT   34.892093   14246.48393
```

# Random forest: variable importance plot

```
# Or, visualize using a plot
varImpPlot(rf_housing)
```

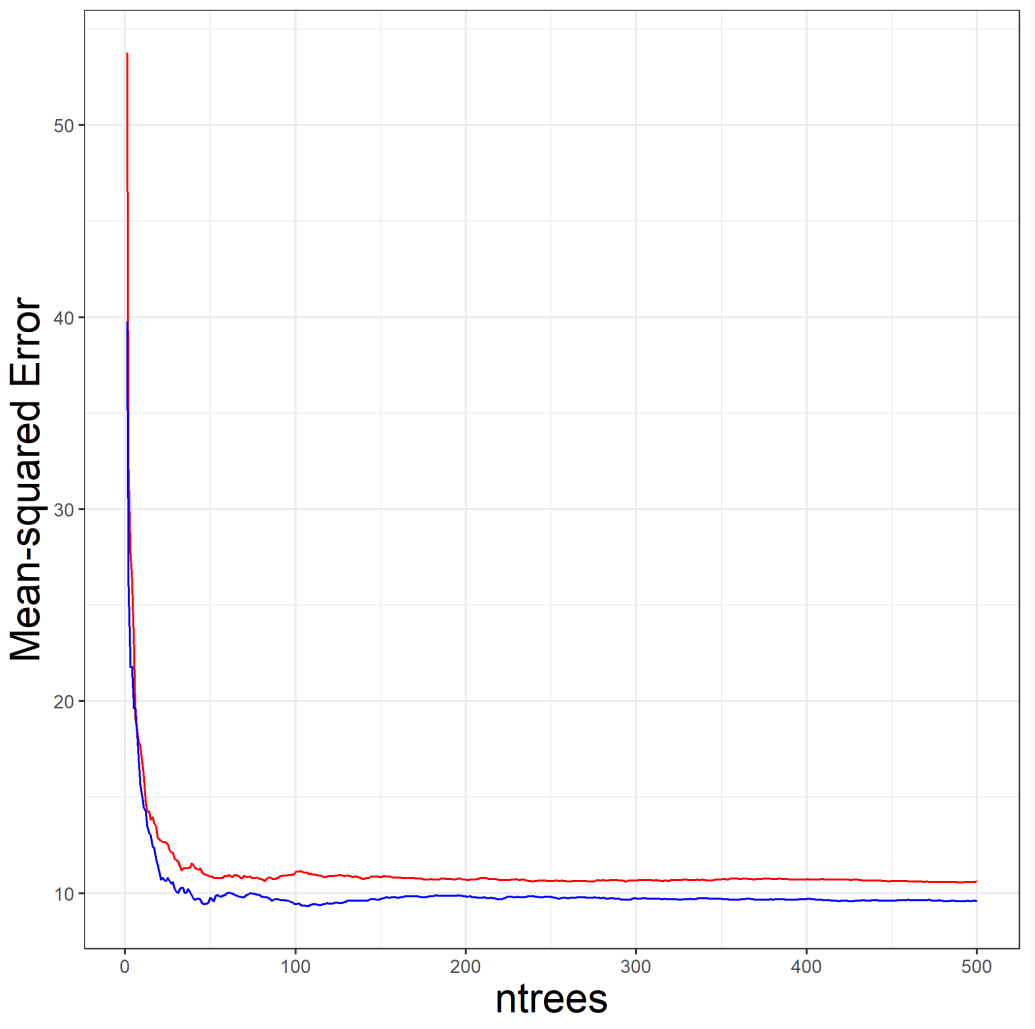


# Our random forest model outperforms the bagging model

```
bag.mse <- bag_housing$mse
rf.mse <- rf_housing$mse
ntrees <- 1:bag_housing$ntree

df <- data.frame(ntrees = ntrees,
                 bag.mse = bag.mse,
                 rf.mse = rf.mse)

ggplot(df, aes(x=ntrees)) +
  geom_line(aes(y=bag.mse),color="red") +
  geom_line(aes(y=rf.mse),color="blue") +
  ylab("Mean-squared Error") +
  theme_bw() +
  theme(axis.title=element_text(size=20))
```



# Making predictions

```
# Note: for classification, specify type="prob" (probability) or type="class" (mode)
housing$yhat.bag <- predict(bag_housing, housing)
housing$yhat.rf <- predict(rf_housing, housing)

# MSE
mean((housing$yhat.bag - housing$MEDV)^2)
mean((housing$yhat.rf  - housing$MEDV)^2)
```

```
# [1] 1.782202
# [1] 1.781661
```

# Summary

- Random forest uses bootstrap to increase the number of trees you can grow
  - Averaging over these trees improves predictive performance
  - "Out-of-bag" observations provide a natural test dataset

- Bagging is a special case of random forest

- Boosting is another way to improve predictive performance of trees

- Lab-14 due Sunday at 11:59pm

- Reminder: midterm is **Wednesday, October 29**