

Big Data Analytics

Session 7

Bagging and Random Forests

Pros and Cons of Decision Trees



- Pros:
 - Trees are very easy to explain to people (probably even easier than linear regression)
 - Trees can be plotted graphically
 - They work fine on both classification and regression problems
- Cons:
 - Trees don't have the same **prediction accuracy** as some of the more complicated approaches that we examine in this course
 - **High variance**
 - By aggregating many decision trees, the predictive performance of trees can be substantially improved. How?
 - using methods like **bagging, random forests, and boosting**

Outline



- Bagging
 - Bootstrapping
 - Bagging for Regression Trees
 - Bagging for Classification Trees
 - Out-of-Bag Error Estimation
 - Variable Importance: Relative Influence Plots
- Random Forests

Problem



- Decision trees discussed earlier suffer from high variance!
 - If we randomly split the training data into 2 parts, and fit decision trees on both parts, the results could be quite different
- We would like to have models with low variance
- To solve this problem, we can use bagging (bootstrap aggregating).

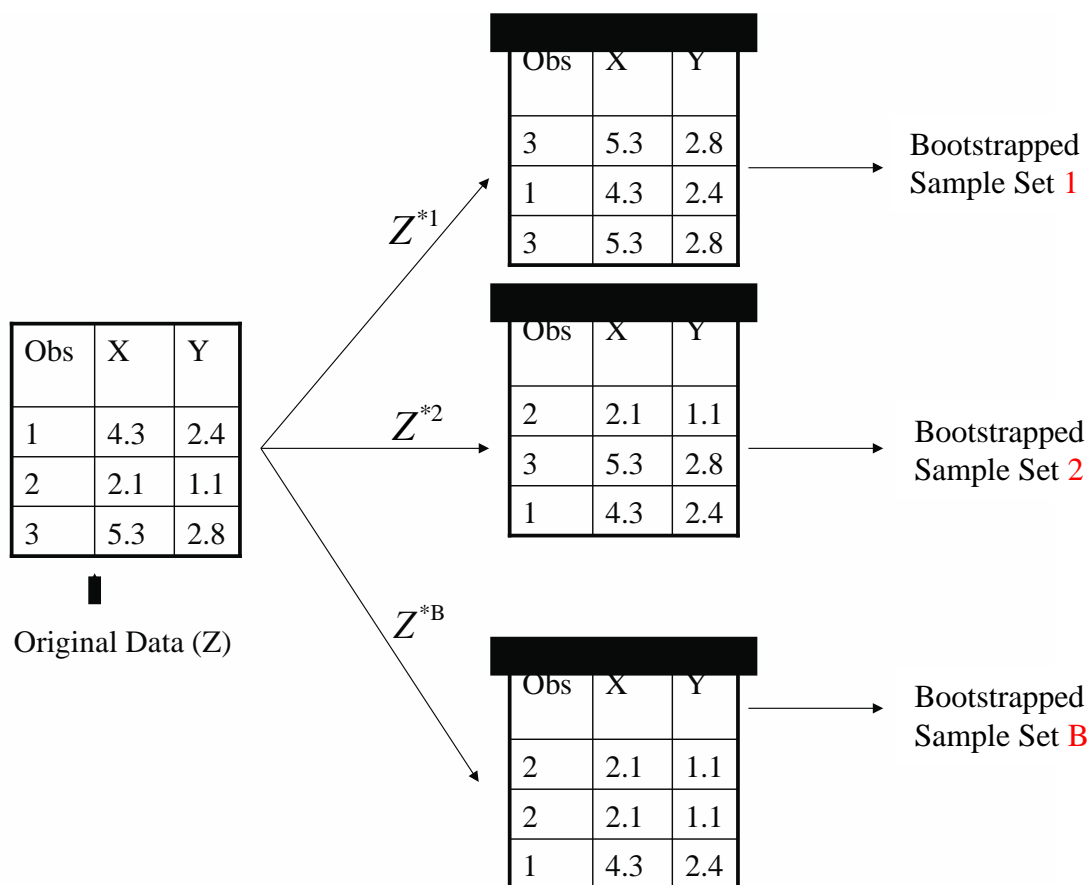
Random Sampling



- Before introducing bootstrapping, we introduce **random sampling with/without replacement**
 - Random sampling **without** replacement
 - One deliberately avoids choosing any member of the population more than once
 - Once a member is chosen, it cannot be chosen again
 - Random sampling **with** replacement
 - The population is “replaced” every time a member is chosen
 - The same member can be chosen more than once

Bootstrapping is simple!

- Resampling of the observed dataset (and of equal size to the observed dataset), each of which is obtained by **random sampling with replacement** from the original dataset.



→ We could have distinct “training sets” by repeatedly sampling from the original data set

→ Distinct test sets are usually there to obtain a **measure of variability**

What is bagging?

- Bagging is an extremely powerful idea based on two things:
 - Bootstrapping: plenty of training datasets!
 - Averaging: reduces variance!
- Why does averaging reduces variance?
 - Averaging a set of observations reduces variance.
 - Recall that given a set of n independent observations Z_1, \dots, Z_n ,
 - each with variance σ^2 ,
 - the variance of the mean \bar{Z} of the observations is given by σ^2/n

How does bagging work?



- Generate B different bootstrapped training datasets
- **Train** the statistical learning method (e.g. a decision tree) on each of the B training datasets, and obtain the prediction
- **For prediction:**
 - **Regression:** average all predictions from all B trees
 - **Classification:** majority vote among all B trees

Outline



- Bagging
 - Bootstrapping
 - Bagging for Regression Trees
 - Bagging for Classification Trees
 - Out-of-Bag Error Estimation
 - Variable Importance: Relative Influence Plots
- Random Forests

Bagging for Regression Trees



- Construct B regression trees using B bootstrapped training datasets
- Average the resulting predictions
- Note: These trees **are not pruned**, so each individual tree has **high variance** but **low bias**
- **Averaging these trees reduces variance**, and thus we end up lowering both **variance** and **bias** 😊

Example: Boston Housing Data



- Apply bagging to the Boston data, using the `randomForest` package in R
 - Later, we will see bagging is a special case of a random forest

```
> library(randomForest)
> library(MASS)
> set.seed(1)
> train=sample(1:nrow(Boston),nrow(Boston)/2)
> set.seed(6)
> bag.boston=randomForest(medv~., data=Boston, subset=train, mtry=13,
  importance=TRUE)
```

#**importance**: Should importance of predictors be assessed?

#**mtry**: number of predictors sampled for splitting at each node. It indicates that **all 13 predictors** should be considered for each split of the tree, this indicates bagging.

#**ntree=500** by default

```
> bag.boston
```

```
Call: randomForest(formula = medv ~ ., data = Boston, mtry = 13, importance = TRUE,
  subset = train)
```

```
      Type of random forest: regression
```

```
      Number of trees: 500
```

```
No. of variables tried at each split: 13
```

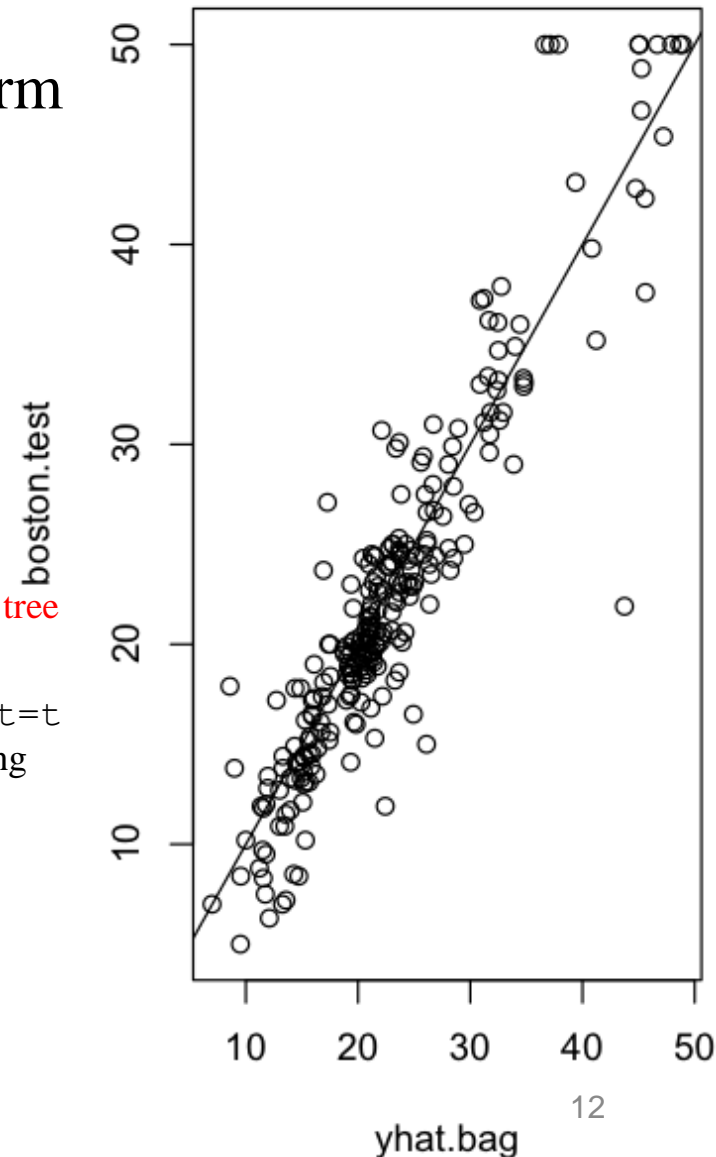
```
      Mean of squared residuals: 10.89212
```

```
      % Var explained: 86.81
```

Example: Boston Housing Data

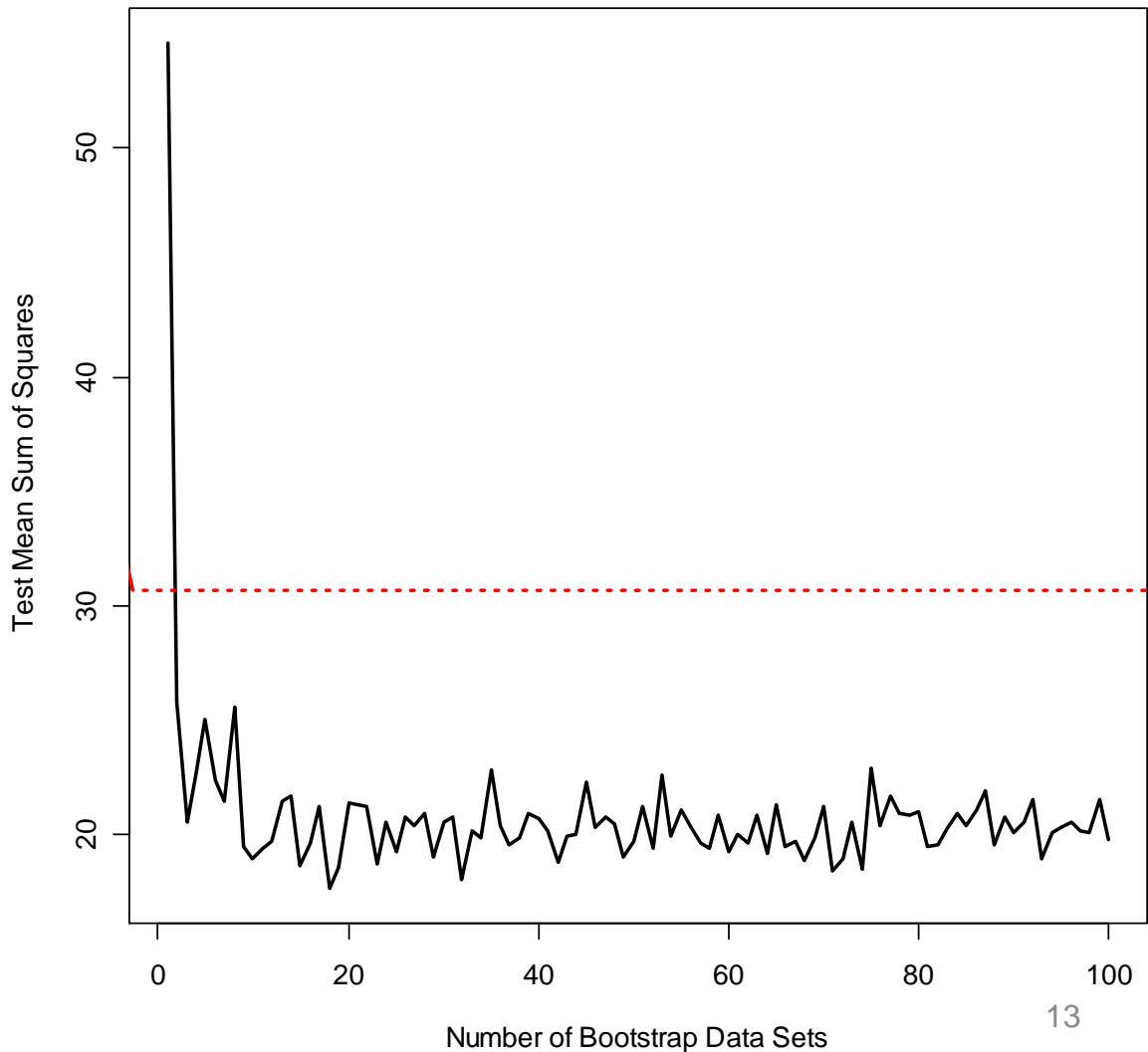
- How well does this bagged model perform on the test set?

```
> yhat.bag=predict(bag.boston,newdata=Boston[-train,])
> boston.test = Boston [-train , "medv"]
> plot(yhat.bag,boston.test)
> abline(0,1)
> mean((yhat.bag-boston.test)^2)
[1] 13.33672 ← the test MSE associated with the bagged regression tree
> set.seed(6)
> bag.boston=randomForest(medv~.,data=Boston,subset=train,mtry=13,ntree=25, importance=TRUE) ← changing the number of trees grown
> yhat.bag=predict(bag.boston,newdata=Boston[-train,])
> mean((yhat.bag-boston.test)^2)
[1] 14.02642 → With less trees, the test MSE increases
```



A Comparison of Error Rates – Boston Housing Data

- The red line represents the test MSE using a single tree.
- The black line corresponds to the bagging error rate



Example: Boston Housing Data



- The code for generating the previous plot:

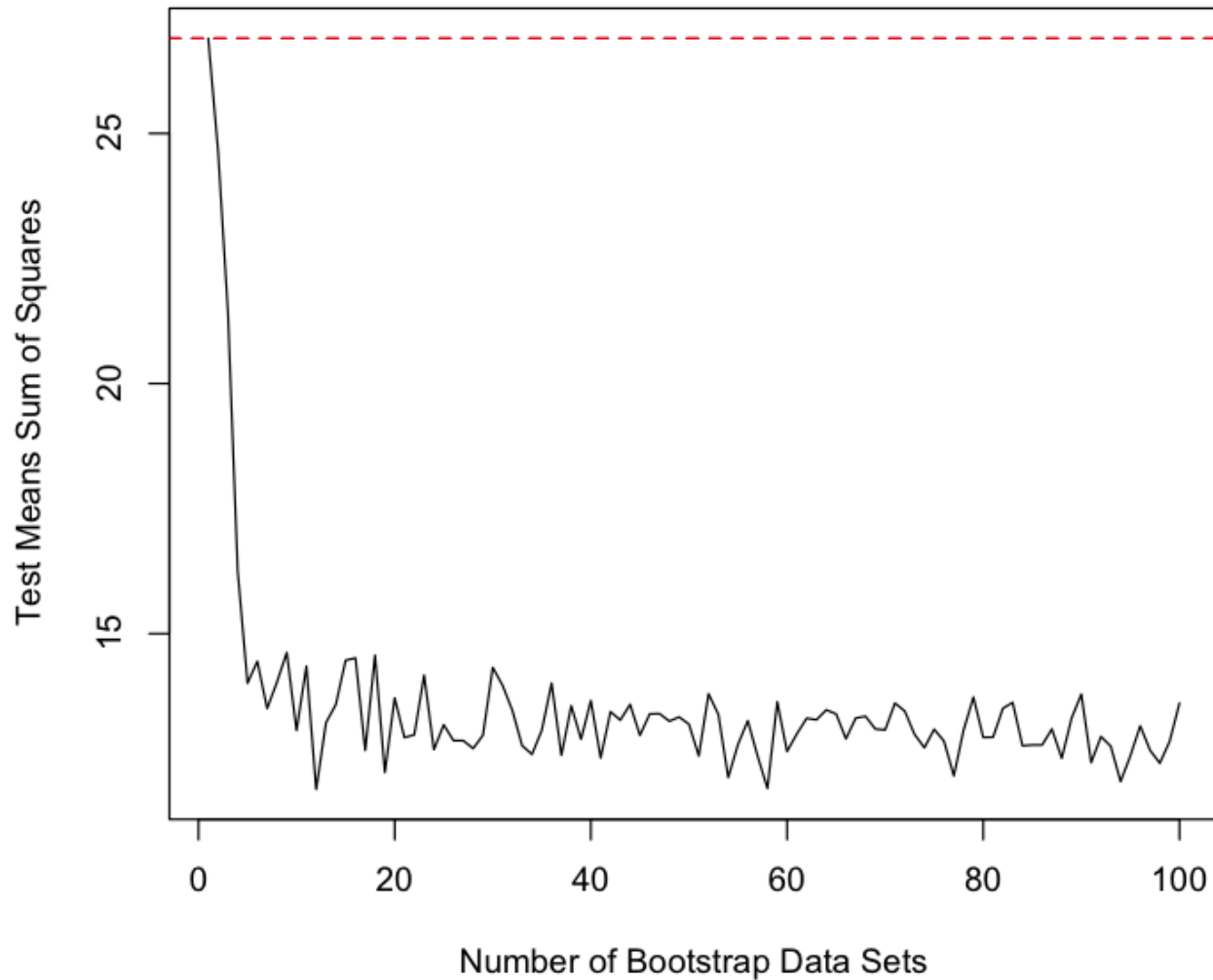
```
> test.mse=rep(0,100)

> for(i in 1:100){
+   set.seed(1)
+   bag.boston=randomForest(medv~., data=Boston, subset=train,
+     mtry=13, ntree=i, importance=TRUE)
+   yhat.bag=predict(bag.boston,newdata=Boston[-train,])
+   test.mse[i]=mean((yhat.bag-boston.test)^2)
+ }

> plot(test.mse,xlab="Number of Bootstrap Data Sets",ylab="Test
  Means Sum of Squares", type="l")
> abline(h=test.mse[1],lty=2,col="red") #h:the y-value(s) for horizontal line(s).
```

The plot obtained is a bit different from the previous plot, see next slide:

Example: Boston Housing Data



Outline



- Bagging
 - Bootstrapping
 - Bagging for Regression Trees
 - Bagging for Classification Trees
 - Out-of-Bag Error Estimation
 - Variable Importance: Relative Influence Plots
- Random Forests

Bagging for Classification Trees



- Construct B classification trees using B bootstrapped training datasets
- For prediction, there are two approaches:
 1. Record the class that each bootstrapped data set predicts and provide an overall prediction to the most commonly occurring one (**majority vote**).
 2. If our classifier produces probability estimates we can just average the probabilities and then predict to the class with the highest probability.
- Both methods work well.

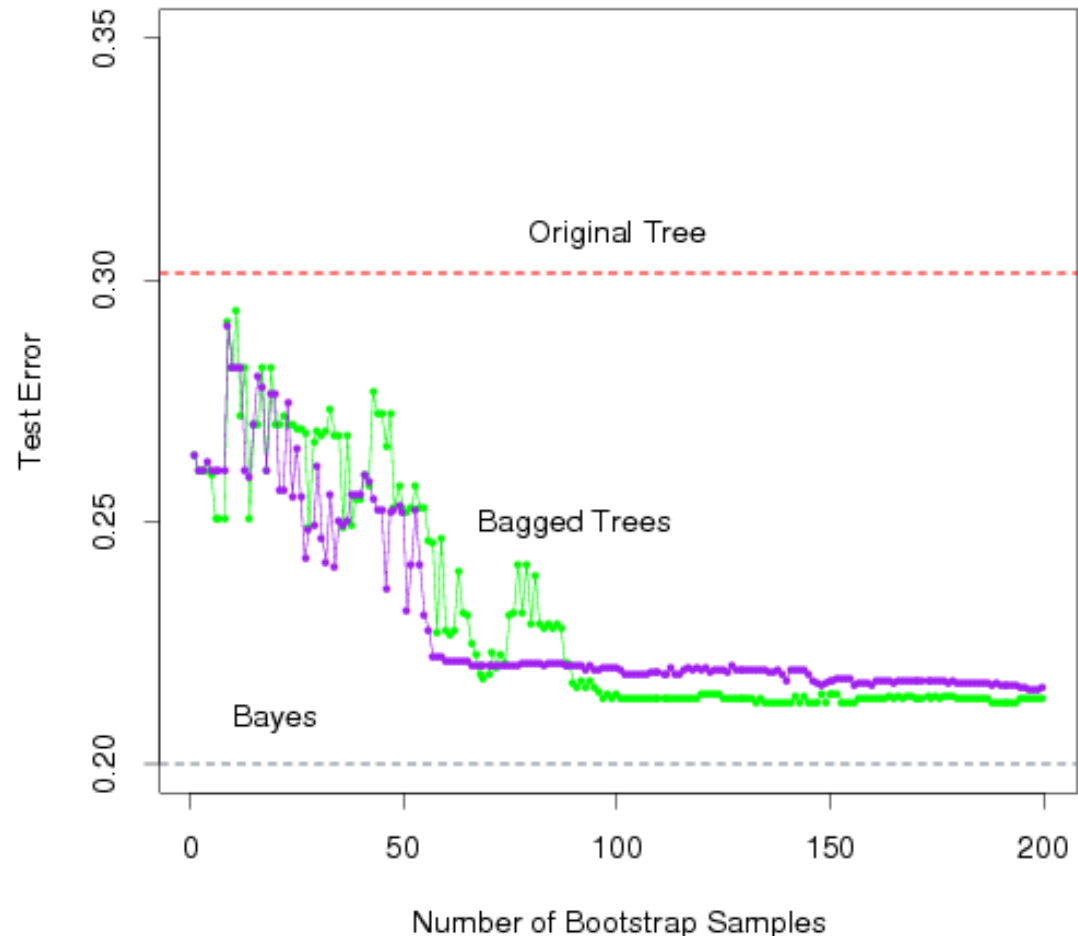
Example: Car Seat Data

- Apply bagging to the `Carseats` data, using the `randomForest` package in R

```
> library(ISLR)
> High=ifelse(Carseats$Sales<=8,"No","Yes")
> Carseats=data.frame(Carseats,High) # add one column High to Carseats
> set.seed(2)
> train=sample(1:nrow(Carseats),200)
> Carseats.test=Carseats[-train,]
> High.test = Carseats[-train, "High"]
> bag.carseats=randomForest(High~.-Sales,Carseats,subset=train,mtry=10)
> yhat.carseats=predict(bag.carseats,newdata=Carseats.test)
> table(yhat.carseats,High.test)
      High.test
yhat.carseats No  Yes
      No    95   17
      Yes   21   67
> (21+17)/200
[1] 0.19
```

A Comparison of Error Rates

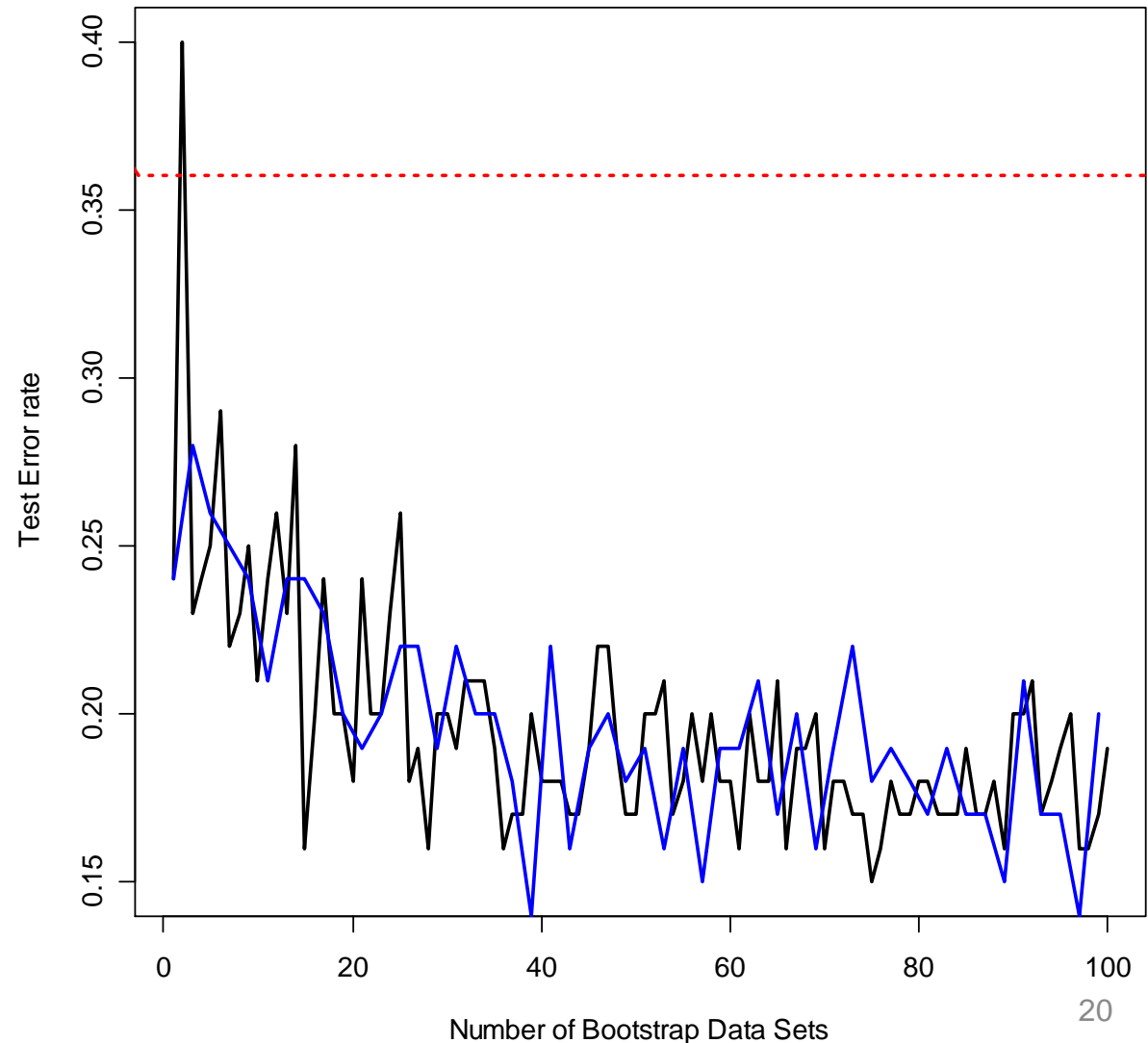
- Here the green line represents a simple majority vote approach
- The purple line corresponds to averaging the probability estimates
- Both do far better than a single tree (dashed red) and get close to the Bayes error rate (dashed grey)



Bayes error rate is the **lowest possible error rate** for a given class of classifier.

Example: Car Seat Data

- The red line represents the test error rate using a single tree.
- The black line corresponds to the bagging error rate using majority vote while the blue line averages the probabilities.



Example: Car Seat Data



- Code to obtain the plot on the last slide:

```
# calculate the black line:
```

```
> yhat.ter.mv=rep(0,200)    # a vector for Test Error Rate using majority vote (by default)
```

```
> for(i in 1:200){
```

```
+ bag.carseats=randomForest(High~.-Sales,Carseats,subset=train,mtry=10,ntree=i)
```

```
+ yhat.carseats=predict(bag.carseats,newdata=Carseats.test)
```

```
+ yhat.ter.mv[i]=(table(yhat.carseats,High.test)[1,2]+table(yhat.carseats,High.test)[2,1])/200
```

```
# or yhat.ter.mv[i] = mean(yhat.carseats!=High.test)
```

```
+ }
```

```
#plot the black line
```

```
> plot(yhat.ter.mv,xlab="Number of Bootstrap Data Sets",ylab="Test Error Rate",  
      type="l",ylim=c(0.10,0.35))
```

```
#plot the red dashed line
```

```
> abline(h=yhat.ter.mv[1],lty=2,col="red")
```

Example: Car Seat Data



#Calculate the blue line

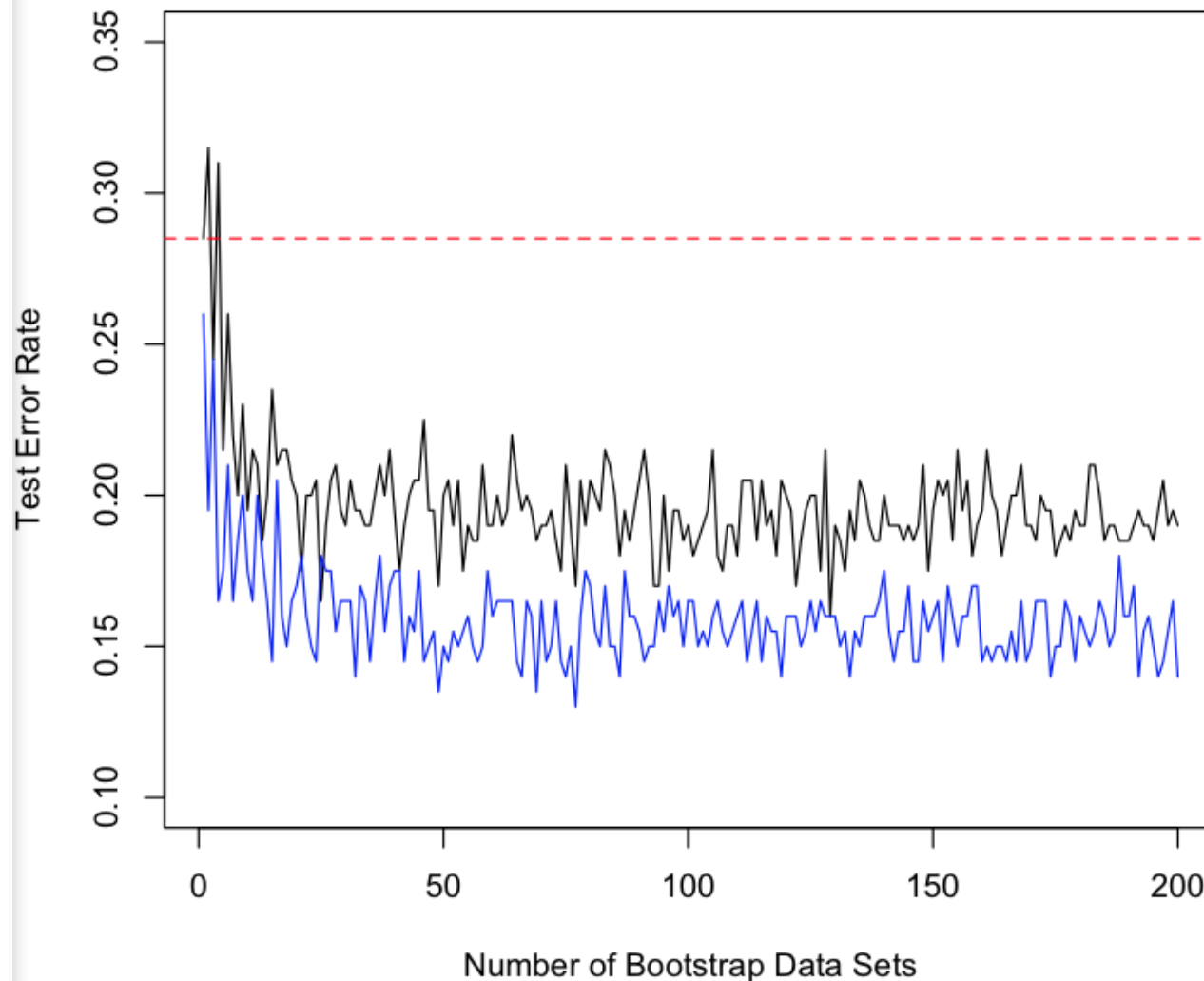
in this case, we need to build a bagging for REGRESSION trees first and discretise the result to “Yes” or “No” later.

```
> yhat.ter.ave=rep(0,200)    # a vector for Test Error Rate using averaging
> for(j in 1:200){
+   bag.carseats.ave=randomForest(Sales~.-High,Carseats,subset=train,mtry=10,ntree=j)
+   yhat.carseats=predict(bag.carseats.ave,newdata=Carseats.test)
+   yhat.carseats.class=ifelse(yhat.carseats<=8,"No","Yes")
+   yhat.ter.ave[j]=(table(yhat.carseats.class,High.test)[1,2]+table(yhat.carseats.class,High.test)[2,1])/200
+ }
```

#plot the blue line:

```
> lines(yhat.ter.ave,col="blue")
```

Example: Car Seat Data



← This is the plot I've got

Outline



- Bagging
 - Bootstrapping
 - Bagging for Regression Trees
 - Bagging for Classification Trees
 - Out-of-Bag Error Estimation
 - Variable Importance: Relative Influence Plots
- Random Forests

Out-of-Bag Error Estimation



- There is a very straightforward way to estimate the test error of a bagged model
 - No need to perform cross validation or the validation set approach
 - Since bootstrapping involves random selection of subsets of observations to build a training data set, the remaining non-selected part could be the testing data.
 - On average, each bagged tree makes use of around $2/3$ of the observations, so we end up having $1/3$ of the observations used for testing.
 - The remaining $1/3$ of the observations are referred to as the out-of-bag (OOB) observations.
 - The estimated test error using the OOB observations is called the **OOB error**.

Out-of-Bag Error Estimation



- When the number of trees B is **sufficiently large**, OOB error is virtually equivalent to LOOCV error.
- The OOB approach for estimating the test error is particularly convenient when performing bagging on **large data sets** for which the CV would be computationally expensive.
- You may find the OOB error rate when printing the summary

```
> print(bag.carseats)
```

```
Call:
```

```
randomForest(formula = High ~ . - Sales, data = Carseats, mtry = 10,  
              subset = train)
```

```
      Type of random forest: classification
```

```
      Number of trees: 500
```

```
No. of variables tried at each split: 10
```

```
      OOB estimate of error rate: 22%
```

```
.....
```

Outline



- Bagging
 - Bootstrapping
 - Bagging for Regression Trees
 - Bagging for Classification Trees
 - Out-of-Bag Error Estimation
 - Variable Importance: Relative Influence Plots
- Random Forests

Variable Importance Measure



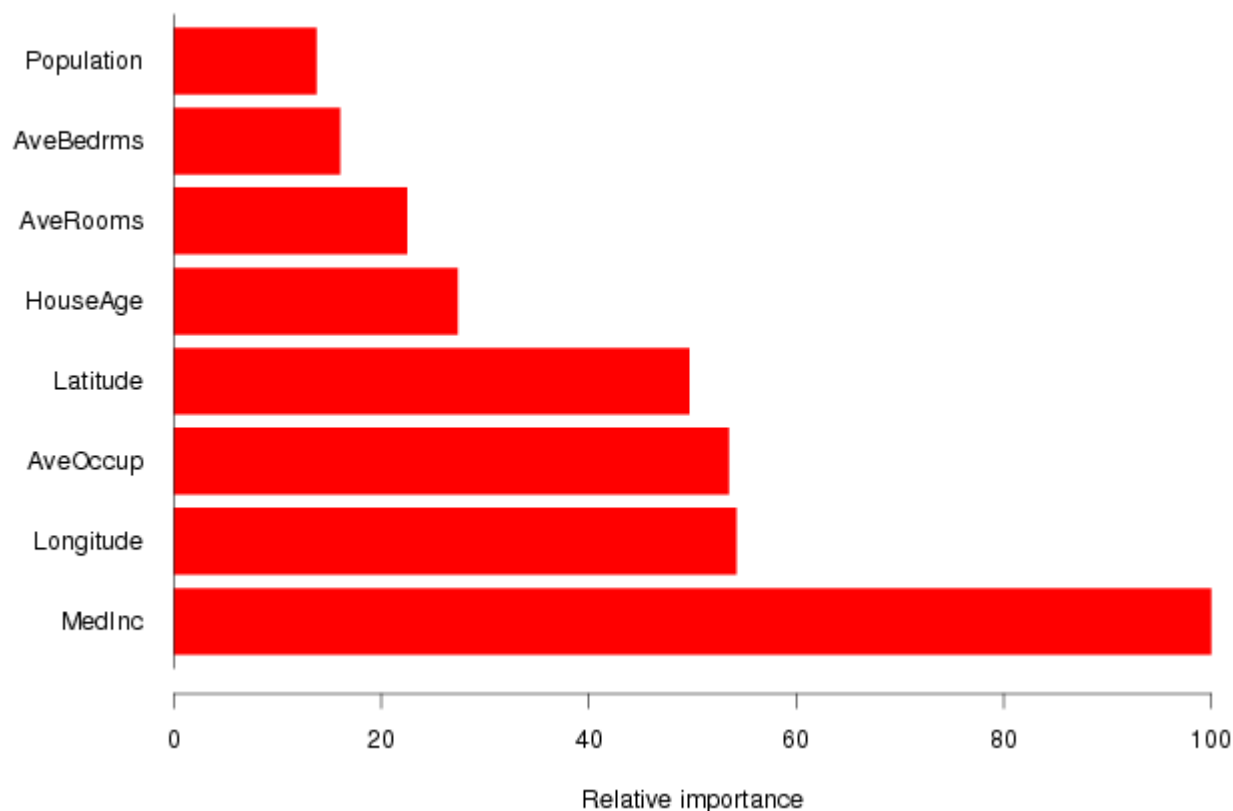
- Bagging typically **improves the accuracy** over prediction using a single tree, but it is now **hard to interpret** the model!
 - We have hundreds of trees, and it is no longer clear which variables are most important to the procedure
 - Thus bagging improves prediction accuracy **at the expense of interpretability**
- But, we can still get an overall summary of the importance of each predictor using **Relative Influence Plots**

Relative Influence Plots

- How do we decide **which variables are most useful** in predicting the response?
 - We can compute something called **relative influence plots**
 - These plots give **a score for each variable**
 - These scores represents **the decrease in MSE** when splitting on a particular variable
 - A number close to zero indicates the variable is not important and could be dropped
 - **The larger the score the more influence the variable has.**

Example: Housing Data

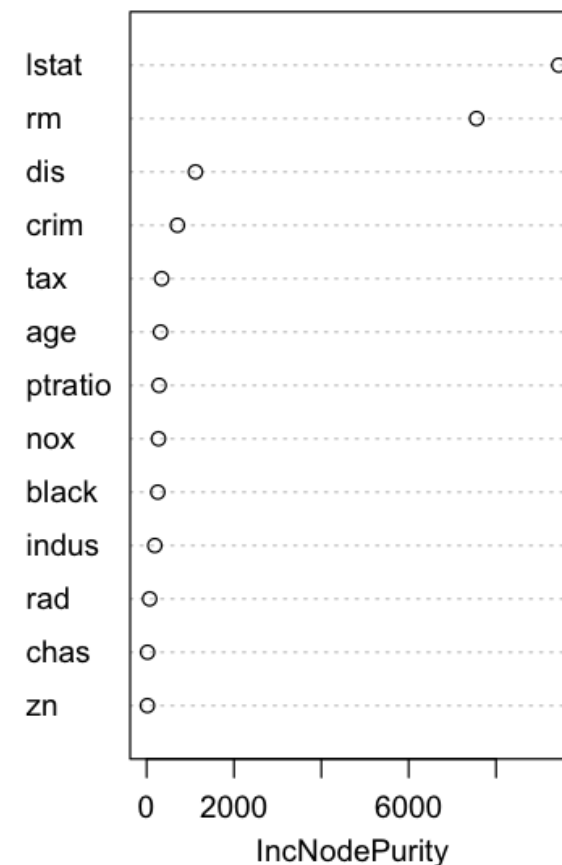
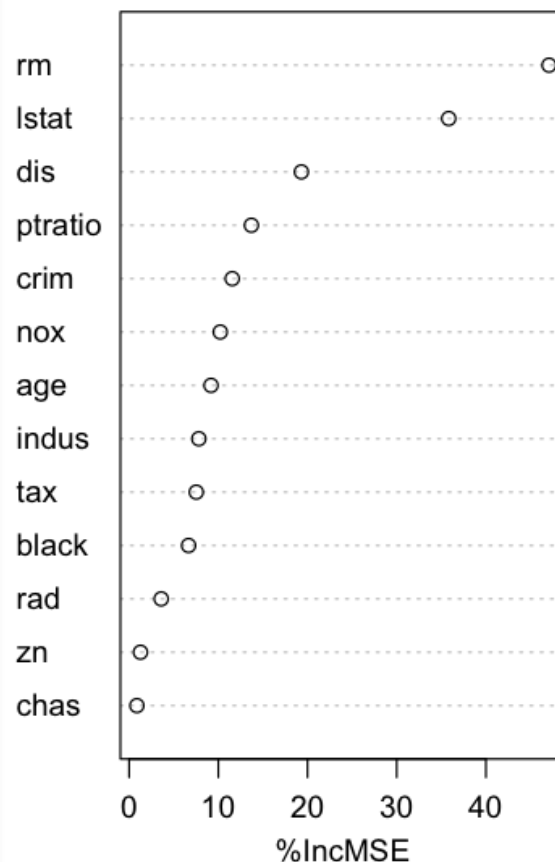
- **Median Income** is by far the most important variable.
- **Longitude**, **Latitude** and **Average occupancy** are the next most important.



Example: Boston Housing Data

```
> importance(bag.boston)
```

	%IncMSE	IncNodePurity
crim	11.5623632	705.52039
zn	1.2742557	17.50011
indus	7.8148780	187.07632
chas	0.8714603	20.51912
nox	10.2193795	271.39076
rm	47.0763407	7555.54873
age	9.1828854	317.74911
dis	19.3127465	1117.95231
rad	3.5974754	68.49749
tax	7.5277047	344.32668
ptratio	13.7071255	285.55153
black	6.6649845	254.79199
lstat	35.8265510	9437.81585



```
> varImpPlot(bag.boston) # find out how to plot in the “bar chart” style
```

Outline



- Bagging
 - Bootstrapping
 - Bagging for Regression Trees
 - Bagging for Classification Trees
 - Out-of-Bag Error Estimation
 - Variable Importance: Relative Influence Plots
- Random Forests

Random Forests



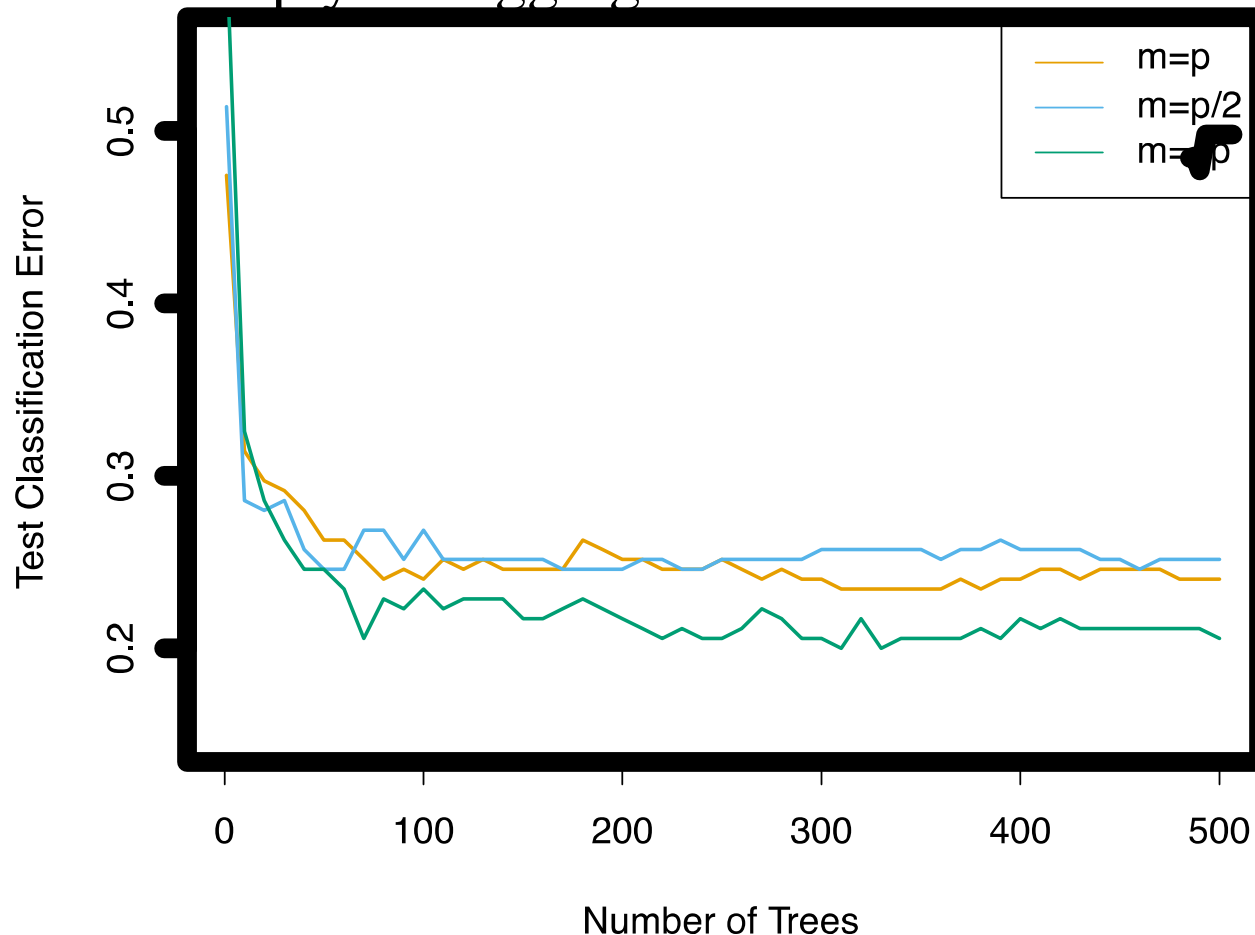
- It is a very efficient statistical learning method
- It builds on the idea of bagging, but it provides an improvement because it **de-correlates the trees**
- How does it work?
 - Build a number of decision trees on bootstrapped training sample, but when building these trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors (Usually $m = \sqrt{p}$ (square root of p))

Why?

- Why are we considering a random sample of m predictors instead of all p predictors for splitting?
 - Suppose that we have a very strong predictor in the data set along with a number of other moderately strong predictors, then in the collection of bagged trees, most or all of them will use the very strong predictor for the first split!
 - All bagged trees will look similar. Hence all the predictions from the bagged trees will be highly correlated
 - Averaging many highly correlated quantities does not lead to a large variance reduction, and thus random forests “de-correlates” the bagged trees leading to more reduction in variance

Random Forest with Different Values of “ m ”

- Notice when random forests are built using $m = p$, then this amounts simply to bagging.

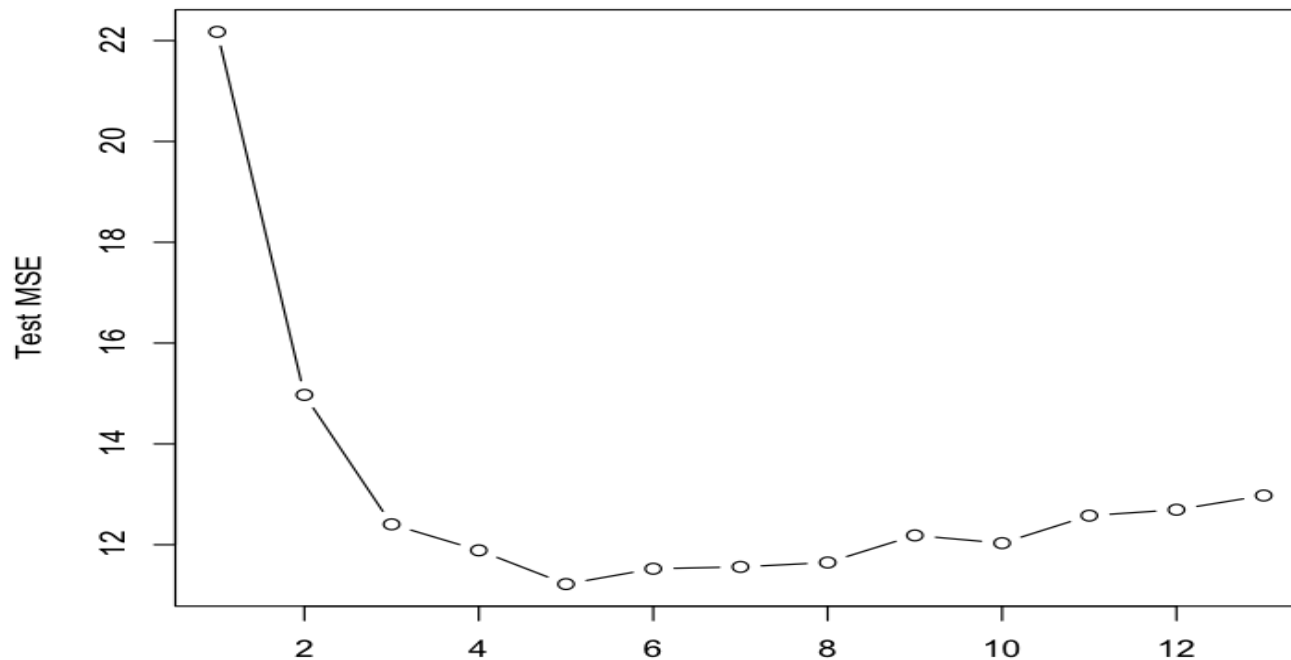


Random Forest

- Growing a random forest proceeds in exactly the same way as in bagging, except that a smaller `mtry` is used
 - By default, `randomForest()` uses
 - $p/3$ variables when building a random forest of regression trees
 - \sqrt{p} variables when building a random forest of classification trees

```
> set.seed(1)
>
  rf.boston=randomForest(medv~., data=Boston, subset=train, mtry
=6, importance=TRUE)
> yhat.rf=predict(rf.boston, newdata=Boston[-train,])
> boston.test=Boston[-train, "medv"]
> mean((yhat.rf-boston.test)^2)
[1] 11.38463 ← the test set MSE, smaller than that derived from a bagged model.
  There is an improvement!
```

Random Forest



- Find the best value of `mtry`:

```
> testMSE=rep(0,13)
> for(i in 1:13){
+   rf.boston=randomForest(medv~.,data=Boston,subset=train,mtry=i,importance=TRUE)
+   yhat.rf=predict(rf.boston,newdata=Boston[-train,])
+   testMSE[i]=mean((yhat.rf-boston.test)^2)
+ }
> plot(testMSE,type="b",xlab="mtry",ylab="Test MSE")
```

Epilogue



- An analogue
 - Decision tree: Validation set approach
 - Use only half of the training set to build a model
 - High variance
 - Bagging: LOOCV
 - A way to utilise almost all observations in the data set to train a model
 - Random Forest: K-fold CV
 - De-correlate the training sets
 - More reduction on variance
 - Bagging is a special case of Random Forest, LOOCV is a special case of K-fold CV

LAB

Hitters



- Explore the Hitters dataset
 - Build a bagged model and a random forest model
 - y: Salary
 - x: all the features other than Salary
 - Play with 1) set.seed 2) mtry and 3) ntree, plot
 - a graph that shows test MSE vs mtry for different seeds
 - a graph that shows test MSE vs ntree for different seeds
 - Find the best/reasonably good mtry and ntree
 - Check the importance of each predictor
 - Check the OOB error estimation
 - Compare the bagged model and RF model with the tree model
 - Compare the test MSE