Module 8: TREE-BASED METHODS

TMA4268 Statistical Learning V2019

Mette Langaas and Thea Roksvåg, Department of Mathematical Sciences, NTNU

week 10 2019

Contents

Last changes: (03.03: broken links, example error, 25.02: first version)

Introduction

Learning material for this module

- James et al (2013): An Introduction to Statistical Learning. Chapter 8.

- Classnotes 04.03.2019
- Classnotes 07.03.2019

Some of the figures in this presentation are taken (or are inspired) from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

What will you learn?

- Decision tree - idea and example
- Regression trees - how to grow
- Classification trees - any changes to the above
- Pruning a tree
- Bagging, with variable importance plots
- Random forests
- Boosting

Running example: Detection of Minor Head Injury

Remark: this data is artificial.

Assume that you work as a statistician at a hospital, and the head of department asks you to develop a

simple method for detecting whether a patient is at risk for having a minor head injury.

The method should be easy to

- interpret for the medical personell that are not skilled in statistics, and
- the method should be fast, such that the medical personell quickly can identify a patient that needs treatment.

- The variable *clinically.important.brain.injury* will be the response of our model: It has value 1 if a person has an acute brain injury, and 0 otherwise.
- In this dataset, 250 (19%) of the patients have a clinically important brain injury.
- 10 variables are used as explanatory variables. These variables describe the state of the patient:
  - Is he/she vomiting?
  - Is the Glasgow Coma Scale (GCS) score after 2 hours equal to 15 (or not)?
  - Has he/she an open scull fracture?
  - Has he/she had a loss of consciousness?
  - and so on.

Comment on GCS: the scale goes back to an article in the Lancet in 1974, and is used to describe the level of consciousness of patients with an acute brain injury. Read more? https://www.glasgowcomascale.org/what-is-gcs/

These are questions the medical staff ask themselves when diagnosing the patient. Our job as a statistician is to systemize these questions in a good way so we can use them to estimate the probability of a brain injury.

This can be done by using tree-based methods.

The dataset includes data about 1321 patients and is a modified and smaller version of the (simulated) dataset *headInjury* from the *DAAG* library. We have made *age* instead of *age.65* giving us the "exact" age of the patient.

```
##    amnesia bskullf GCSdecr GCS.13 GCS.15 risk consc oskullf vomit
## 3        0       0       0      0      0    0     0       0     0
## 9        0       0       0      0      0    1     0       0     0
## 11       0       0       0      0      0    0     0       0     0
## 12       1       0       0      0      0    0     0       0     0
## 14       0       0       0      0      0    0     0       0     0
## 16       0       0       0      0      0    0     0       0     0
##    clinically.important.brain.injury age
## 3                                  0  44
## 9                                  0  67
## 11                                 0  62
## 12                                 0   1
## 14                                 0  55
## 16                                 0  63
```

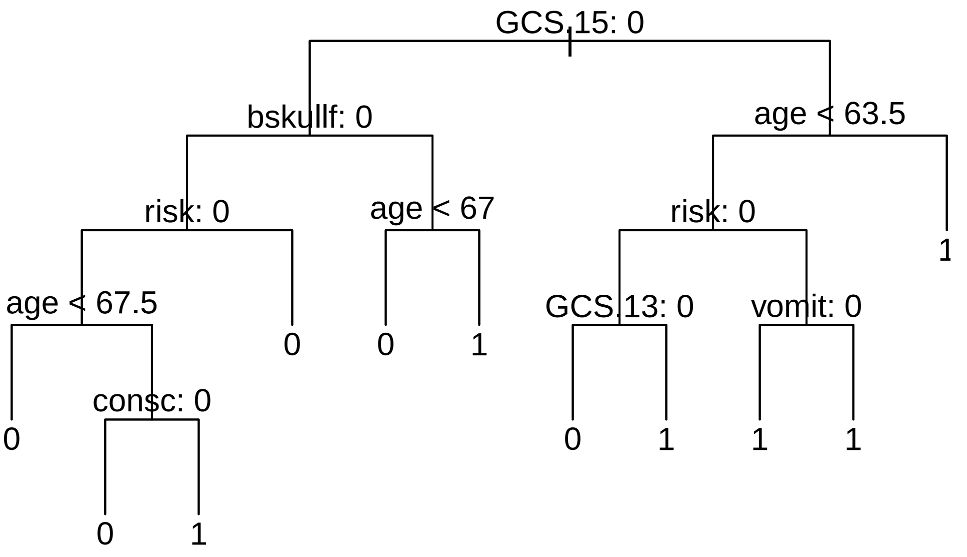
Main idea

The main idea behind tree-based methods is to

- derive a set of decision rules for segmenting the predictor space into a number of regions.
- In order to make a prediction for a new observation, we classify the observation into one of these regions by applying the derived decision rules.
- Then we typically use the mean or a majority vote of the training observations in this region as the prediction.

Below is a clasification tree made from a training set of 850 randomly drawn observations (training set) for the head injury example.

Observe: at splits criterion at node is to the left in the tree. So, root node has "GCS.15:0" so if "GCS.15=0" we go left, and if "GCS.15=1" we go right in the tree.



```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 850 816.00 0 ( 0.8141 0.1859 )
##    2) GCS.15: 0 717 530.00 0 ( 0.8787 0.1213 )
##      4) bskullf: 0 664 407.00 0 ( 0.9081 0.0919 )
##        8) risk: 0 503 222.00 0 ( 0.9423 0.0577 )
##         16) age < 67.5 455 145.00 0 ( 0.9626 0.0374 ) *
##         17) age > 67.5 48  54.00 0 ( 0.7500 0.2500 )
##           34) consc: 0 42  37.80 0 ( 0.8333 0.1667 ) *
##           35) consc: 1 6   5.41 1 ( 0.1667 0.8333 ) *
##        9) risk: 1 161 161.00 0 ( 0.8012 0.1988 ) *
##      5) bskullf: 1 53  73.50 0 ( 0.5094 0.4906 )
##       10) age < 67 46  62.40 0 ( 0.5870 0.4130 ) *
##       11) age > 67 7   0.00 1 ( 0.0000 1.0000 ) *
##    3) GCS.15: 1 133 184.00 1 ( 0.4662 0.5338 )
##      6) age < 63.5 94 128.00 0 ( 0.5745 0.4255 )
##       12) risk: 0 62  72.80 0 ( 0.7258 0.2742 )
##         24) GCS.13: 0 55  55.00 0 ( 0.8000 0.2000 ) *
##         25) GCS.13: 1 7   5.74 1 ( 0.1429 0.8571 ) *
##       13) risk: 1 32  38.00 1 ( 0.2812 0.7188 )
##         26) vomit: 0 22  29.80 1 ( 0.4091 0.5909 ) *
##         27) vomit: 1 10   0.00 1 ( 0.0000 1.0000 ) *
##      7) age > 63.5 39  39.60 1 ( 0.2051 0.7949 ) *
```

By using simple decision rules related to the most important explanatory variables the medical staff can now assess the probability of a brain injury.

For example if the Glasgow Coma Scale of the patient is 15 (*GCS.15.2hours=1*) and the patient is older than 63 year then we should predict a minor injury according to the fitted tree, and the probability of that is estimated to be 0.7949 (node 7 in printout).

- The advantage of such decisions trees is that they are easier to interpret than many of the classification (and regression) methods that we have studied so far
- and they provide an easy way to visualize the data for non-statisticians.

Glossary

- Classification and regression trees are usually drawn upside down, where the top node is called the *root*.
- The *terminal nodes* or *leaf nodes* are the nodes at the bottom, with no splitting criteria. These represent the final predicted class (for classification trees) or predicted response value (for regression trees) and are written symbolically as R_j for $j = 1, 2, ..., J$ - and will be referred to as *non-overlapping regions.*
- *Internal nodes* are all nodes between the root and the terminal nodes. These nodes correspond to the partitions of the predictor space.
- *Branches*: segment of the tree connecting the nodes.

We will consider only binary splits on one variable, but multiway splits and linear combination of variabes are possible - but not so common.

Constructing a decision tree

You can construct decision trees for both classification and regression problems, first we focus on constructing a regression tree.

Regression tree

Assume that we have a dataset consisting of n pairs $(\mathbf{x}_i, Y_i)$, $i = 1, \ldots, n$, and each predictor is $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{ip})$.

Two steps:

1. Divide the predictor space into non-overlapping regions $R_1, R_2, \ldots, R_J$.
2. For every observation that falls into region R_j we make the same prediction - which is the mean of the responses for the training observations that fall into R_j.

How to divide the predictor space into non-overlapping regions $R_1, R_2, \ldots, R_J$?

We could try to minimize the RSS (residual sums of squares) on the training set given by

$$\text{RSS} = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where $\hat{y}_{R_j}$ is the mean response for the training observations in region j. The mean $\hat{y}_{R_j}$ is also the predicted value for a new observations that falls into region j.

To do this we need to consider every partition of the predictor space, and compute the RSS for each partition. An exhaustive search over possible trees is *not* computationally feasible.

Ripley (1996, p 216): Two types of optimality. a) Optimality of the partitioning of the predictor space : only feasible for small dimensions. b) Given partitioning of predictor space, how to represent this by a tree in the best possible way (=minimal expected number of tests) is a NP-complete problem.

A *greedy* approach is taken (aka top-down) - called *recursive binary splitting.*

Recursive binary splitting

We start at the top of the tree and divide the predictor space into two regions, R_1 and R_2 by making a decision rule for one of the predictors $x_1, x_2, ..., x_p$. If we define the two regions by $R_1(j,s) = \{x|x_j < s\}$ and $R_2(j,s) = \{x|x_j \geq s\}$, it means that we need to find the (predictor) j and (splitting point) s that minimize

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2,$$

where $\hat{y}_{R_1}$ and $\hat{y}_{R_2}$ are the mean responses for the training observations in $R_1(j,s)$ and $R_2(j,s)$ respectively. This way we get the two first branches in our decision tree.

- We repeat the process to make branches further down in the tree.
- For every iteration we let each single split depend on *only one of the predictors*, giving us two new branches.
- This is done *successively* and in each step we choose the split that gives the best split at that particular step, i.e the split that gives the smallest RSS.
- We don't consider splits that further down the tree might give a tree with a lower overall RSS.

We continue splitting the predictor space until we reach some *stopping criterion*. For example we stop when no region contains more than 5 observations or when the reduction in the RSS is smaller than a specified limit.

Regression tree: ozone example

Consider the `ozone` data set from the `ElemStatLearn` library. The data set consists of 111 observations on the following variables:

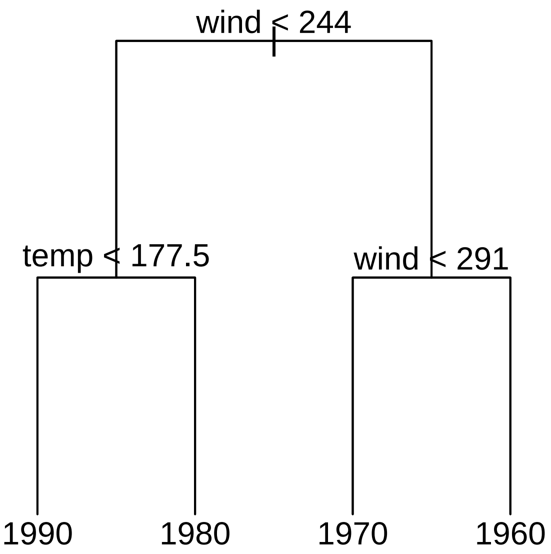
- `ozone` : the concentration of ozone in ppb
- `radiation`: the solar radiation (langleys)
- `temperature` : the daily maximum temperature in degrees F
- `wind` : wind speed in mph

Suppose we want to get an estimate of the ozone concentration based on the measurement of wind speed and the daily maximum temperature.

Year	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	Annual
1956	0	313	311	370	359	334	296	288	274	318
1957	301	284	320	394	347	332	301	280	256	312
1958	0	0	305	349	378	341	328	297	0	333
1959	0	0	302	303	340	322	298	295	0	309
1960	0	287	292	345	375	318	303	304	0	318
1961	0	267	307	332	343	310	297	329	0	312

We can fit a regression tree to the data, with `ozone` as our response variable and `temperature` and `wind` as predictors (not including radiation to make this easier to see). This gives us the following regression tree.

```r
ozone.trainID = sample(1:111, 75)
ozone.train = myozone[ozone.trainID, ]
ozone.test = myozone[-ozone.trainID, ]
ozone.tree = tree(ozone ~ temp + wind, data = ozone.train)
```



```r
summary(ozone.tree)
```

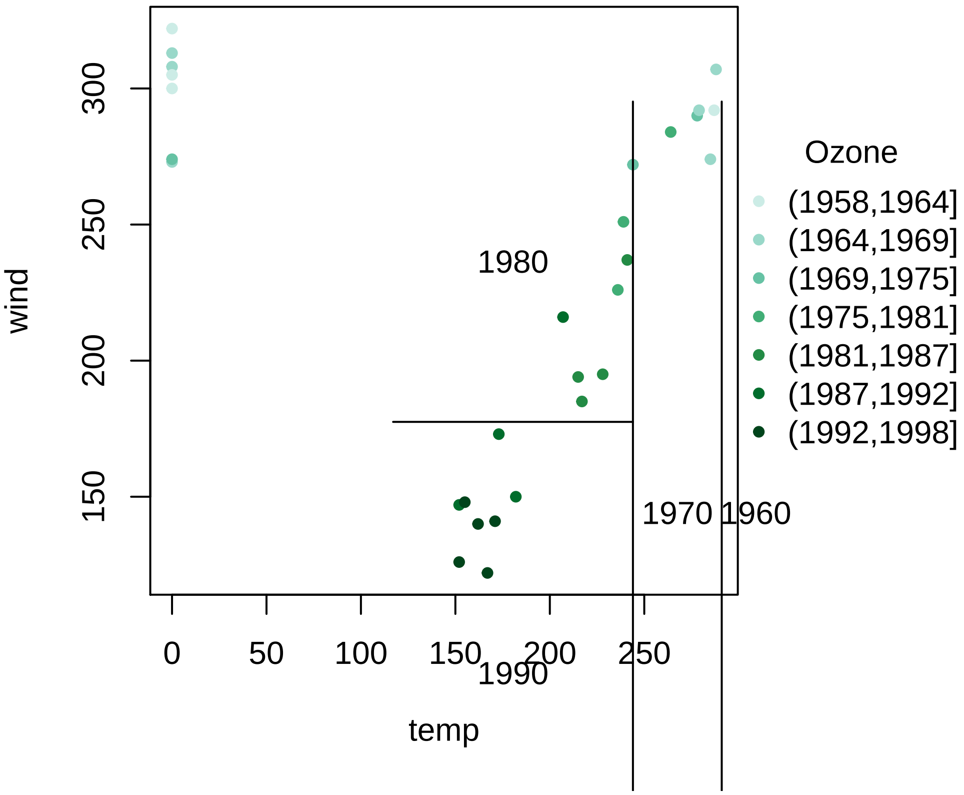
```
##
## Regression tree:
## tree(formula = ozone ~ temp + wind, data = ozone.train)
## Number of terminal nodes:  4
## Residual mean deviance:  12.5 = 313 / 25
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -7.000  -2.290  -0.286   0.000   2.710   6.000
```

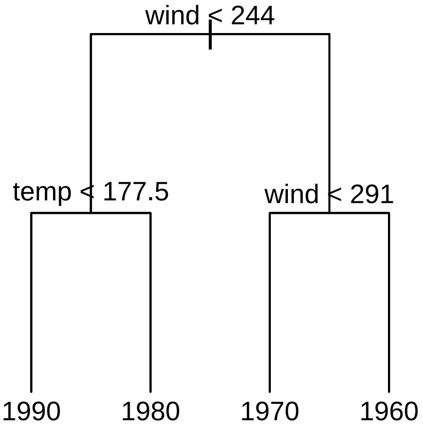
We see that `temperature` is the “most important”" predictor for predicting the ozone concentration. Observe that we can split on the same variable several times.

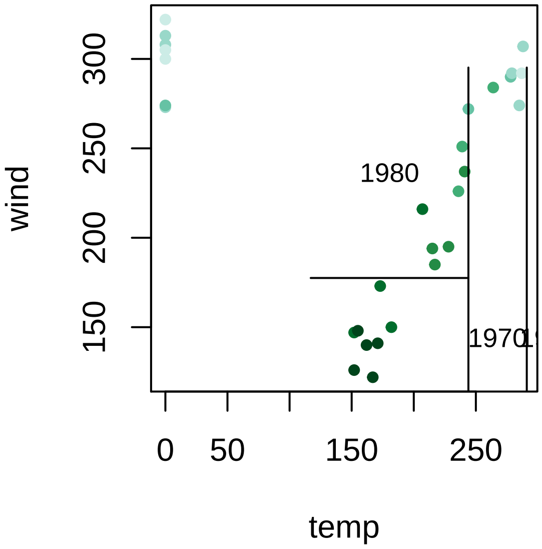
Now, we focus on the regions R_j, $j = 1, \ldots, J$. What is J here? Answer: $J = 6$ (= the number of leaf nodes).

Below we see the partition of the `ozone` data set given in the `ozone.tree`, where the ozone levels have been color-coded, where a darker color corresponds to a higher ozone concentration. Each rectangle corresponds to one leaf node. Each number corresponding to a leaf node has been found by taking an average of all observations (in the training set) in the corresponding region (rectangle). Each line corresponds to an internal node, a binary partition of the predictor space.

```r
par(pty = "s")
# o.class = cut(myozone$ozone, breaks= 7)
o.class = cut(ozone.train$ozone, breaks = 7)
col.oz = brewer.pal(9, "BuGn")
palette(col.oz[3:9])
par(mar = c(5.1, 4.1, 2.1, 8.1), xpd = TRUE)
plot(wind ~ temp, col = o.class, data = ozone.train, pch = 20)
partition.tree(ozone.tree, add = TRUE)
legend("topright", inset = c(-0.4, 0.2), title = "Ozone", legend = levels(o.class),
    col = col.oz[3:9], pch = 20, box.col = NA)
```







Q:

- Explain the connection between the tree and the region plot.
- Why is recursive binary splitting classified as a greedy algorithm?
- Discuss the advantages and disadvantages of letting each single split depend on only one of the predictors.
- Does our tree automatically include interactions between variables?

A:

- Each leaft node correspons to one region with prediction $\hat{y}_{R_j}$
- Recursive splitting does not necessarily give the optimal global solution, but will give the best solution at each split (given what is done previously).
- If the true connection between the response onzone and temperature and wind was so that splits should have been made on the diagonal of the temperature and wind space, that would take many splits on each of temperature and wind to produce. (See more below where we ask the same question for the classification setting.)
- Yes, we may have different effect of wind for different values of temperature.

Tree performance

To test the predictive performance of our regression tree, we have randomly divided our observations into a test and a training set (here 1/3 test).

```
ozone.pred = predict(ozone.tree, newdata = ozone.test)
ozone.MSE = mean((ozone.pred - ozone.test$ozone)^2)
ozone.MSE
```

```
## [1] 30.9177
```

R: function `tree` in library `tree`

by Brian D. Ripley: Fit a Classification or Regression Tree

Description: A tree is grown by binary recursive partitioning using the response in the specified formula and choosing splits from the terms of the right-hand-side.

tree(formula, data, weights, subset, na.action = na.pass, control = tree.control(nobs, …), method = “recursive.partition”, split = c(“deviance”, “gini”), model = FALSE, x = FALSE, y = TRUE, wts = TRUE, …)

- Details: A tree is grown by binary recursive partitioning using the response in the specified formula and choosing splits from the terms of the right-hand-side. Numeric variables are divided into X < a and X > a; the levels of an unordered factor are divided into two non-empty groups. The split which maximizes the reduction in impurity is chosen, the data set split and the process repeated. Splitting continues until the terminal nodes are too small or too few to be split.
- A numerical response gives regression while a factor response gives classifiation.
- The default choice for a function to minimize is the deviance, and for normal data (as we may assume for regression), the deviance is proportional to the RSS. For the interested reader, this is the connection between the deviance and the RSS for regression https://www.math.ntnu.no/emner/TMA4315/2018h/2MLR.html#deviance

---

- Tree growth is limited to a depth of 31 by the use of integers to label nodes.
- Factor predictor variables can have up to 32 levels. This limit is imposed for ease of labelling, but since their use in a classification tree with three or more levels in a response involves a search over 2^(k-1) - 1 groupings for k levels, the practical limit is much less.

A competing R function is `rpart`, explained in https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf

Classification trees

remember the minor head injury example?

Let K be the number of classes for the response.

Building a decision tree in this setting is similar to building a regression tree for a quantitative response, but there are two main differences: *the prediction and the splitting criterion*

1) The prediction:

- In the regression case we use the mean value of the responses in R_j as a prediction for an observation that falls into region R_j.
- For the classification case however, we have two possibilities:
  - Majority vote: Predict that the observation belongs to the most commonly occurring class of the training observations in R_j.

  - Estimate the probability that an observation x_i belongs to a class k, $\hat{p}_{jk}(x_i)$, and then classify according to a threshold value. This estimated probability is the proportion of class k training observations in region R_j, with n_{jk} observations. Region j has N_j observations.

$$\hat{p}_{jk} = \frac{1}{N_j} \sum_{i:x_i \in R_j} I(y_i = k) = \frac{n_{jk}}{N_j}.$$

2) The splitting criterion: We do not use RSS as a splitting criterion for a qualitative variable. Instead we can use some *measure of impurity* of the node. For leaf node j and class $k = 1, \ldots, K$:

Gini index:

$$G = \sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk}),$$

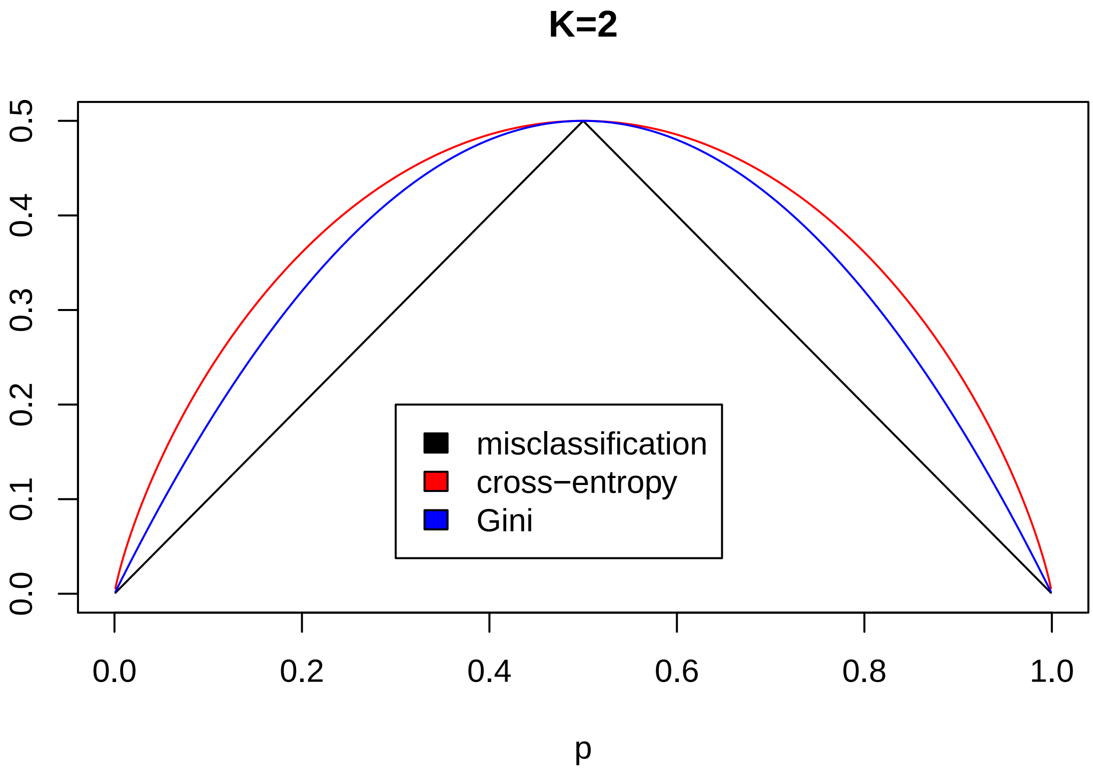
Cross entropy:

$$D = -\sum_{k=1}^{K} \hat{p}_{jk} log \hat{p}_{jk}$$

Here $\hat{p}_{jk}$ is the proportion of training observation in region j that are from class k. Remark: the deviance is a scaled version of the cross entropy. $-2\sum_{k=1}^{K} n_{jk} log \hat{p}_{jk}$ where $\hat{p}_{jk} = \frac{n_{jk}}{N_j}$

When making a split in our classification tree, we want to minimize the Gini index or the cross-entropy.

Q: Why these splitting criteria? Measure of impurity? How? See classnotes.



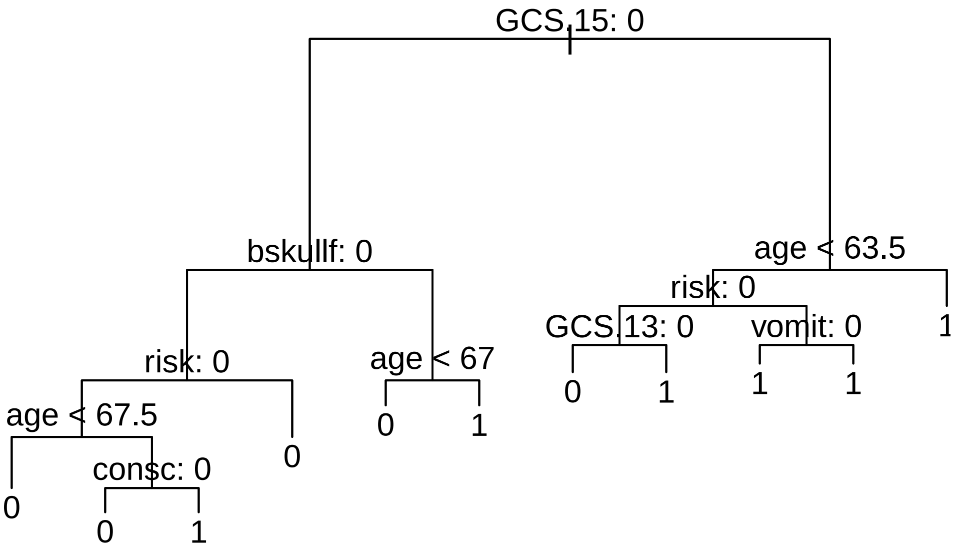
Minor head injury - continued

```
tree.HIClass = tree(clinically.important.brain.injury ~ ., data = headInjury2,
    subset = train, split = "deviance")
summary(tree.HIClass)
```

```
##
## Classification tree:
## tree(formula = clinically.important.brain.injury ~ ., data = headInjury2,
##     subset = train, split = "deviance")
## Variables actually used in tree construction:
## [1] "GCS.15"  "bskullf" "risk"    "age"     "consc"   "GCS.13"  "vomit"
## Number of terminal nodes:  11
## Residual mean deviance:  0.645 = 541 / 839
## Misclassification error rate: 0.124 = 105 / 850
```

Deviance$=-2\sum_j \sum_k n_{jk} log(\hat{p}_{jk})$ for all nodes j and classes k. (Formula on page 255 of Venables and Ripley (2002).)

```
plot(tree.HIClass, type = "proportional")
text(tree.HIClass, pretty = 1)
```



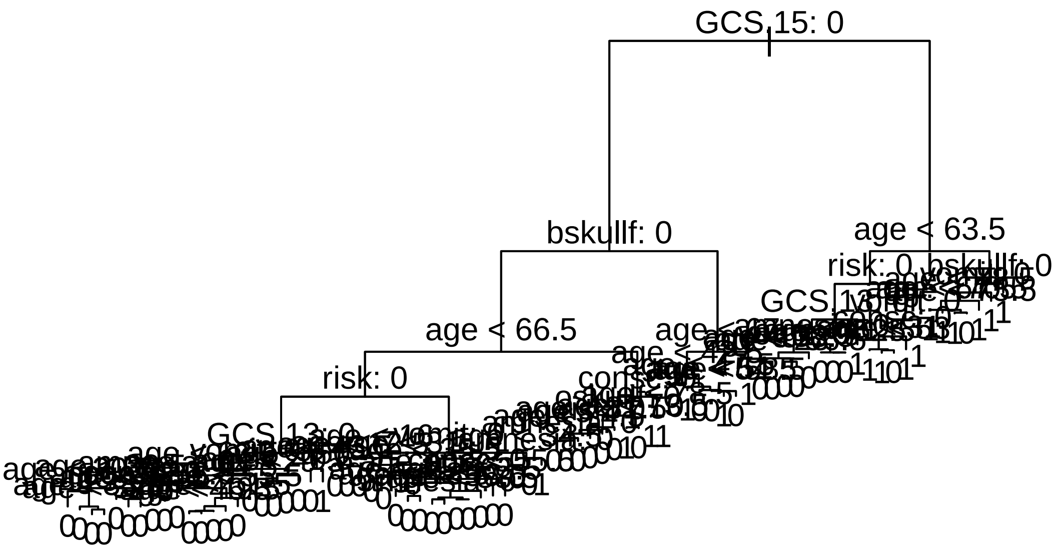
Length of branches are now proportional to the decrease in impurity.

Minor head injury - with Gini index

```
tree.HIClassG = tree(clinically.important.brain.injury ~ ., headInjury2,
    subset = train, split = "gini")
summary(tree.HIClassG)
```

```
##
## Classification tree:
## tree(formula = clinically.important.brain.injury ~ ., data = headInjury2,
##     subset = train, split = "gini")
## Variables actually used in tree construction:
## [1] "GCS.15"  "bskullf" "age"     "risk"    "GCS.13"  "vomit"   "amnesia"
## [8] "consc"   "oskullf"
## Number of terminal nodes:  78
## Residual mean deviance:  0.488 = 377 / 772
## Misclassification error rate: 0.108 = 92 / 850
```

```
tree.HIClassG = tree(clinically.important.brain.injury ~ ., headInjury2,
    subset = train, split = "gini")
plot(tree.HIClassG)
text(tree.HIClassG, pretty = 1)
```



We also use the classification tree to predict the status of the patients in the test set (the one grown with deviance)

```
library(caret)
tree.pred = predict(tree.HIClass, headInjury2[test, ], type = "class")
confusionMatrix(tree.pred, reference = headInjury2[test, ]$clinically.important.brain.injury)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 361  50
##          1  18  42
##
##                Accuracy : 0.856
##                  95% CI : (0.821, 0.886)
##     No Information Rate : 0.805
##     P-Value [Acc > NIR] : 0.00241
##
##                   Kappa : 0.471
##  Mcnemar's Test P-Value : 0.00017
##
##             Sensitivity : 0.953
##             Specificity : 0.457
##          Pos Pred Value : 0.878
##          Neg Pred Value : 0.700
##              Prevalence : 0.805
##          Detection Rate : 0.766
##    Detection Prevalence : 0.873
##       Balanced Accuracy : 0.705
##
##        'Positive' Class : 0
##
```

And for the Gini-grown tree

```
tree.predG = predict(tree.HIClassG, headInjury2[test, ], type = "class")
confusionMatrix(tree.predG, reference = headInjury2[test, ]$clinically.important.brain.injury)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 357  52
##          1  22  40
##
##                Accuracy : 0.843
##                  95% CI : (0.807, 0.875)
##     No Information Rate : 0.805
##     P-Value [Acc > NIR] : 0.018985
##
##                   Kappa : 0.43
##  Mcnemar's Test P-Value : 0.000748
##
##             Sensitivity : 0.942
##             Specificity : 0.435
##          Pos Pred Value : 0.873
##          Neg Pred Value : 0.645
##              Prevalence : 0.805
##          Detection Rate : 0.758
##    Detection Prevalence : 0.868
##       Balanced Accuracy : 0.688
##
##        'Positive' Class : 0
##
```

Questions:

- The classification tree has two terminal nodes with factor "1" originating from the same branch. Why do we get this "unnecessary" split?
- What if we have x_1 and x_2 and the true class boundary (two classes) is linear in x_1, x_2 space. How can we do that with our binary recursive splits?

- What about a rectangular boundary (figure above)?
- Study the above confusion matrices. One type of mistake is more severe than the other. Discuss if it is possible to change the algorithm in order to decrease the number of severe mistakes.

Pruning

Imagine that we have a data set with many predictors, and that we fit a large tree. Then, the number of observations from the training set that falls into some of the regions R_j may be small, and we may be concerned that we have overfitted the training data.

Pruning is a technique for solving this problem.

By *pruning* the tree we reduce the size or depth of the decision tree. When we reduce the number of terminal nodes and regions $R_1, ..., R_J$, each region will probably contain more observations. This way we reduce the probability of overfitting, and we may get better predictions for test data.

If we have a large dataset with many explanatory variables and terminal nodes, we can also prune the tree if we want to create a simpler tree and increase the interpretability of the model.

In the classification tree (grown with deviance) we saw that we got several unnecessary splits. This is also something that can be avoided by pruning.

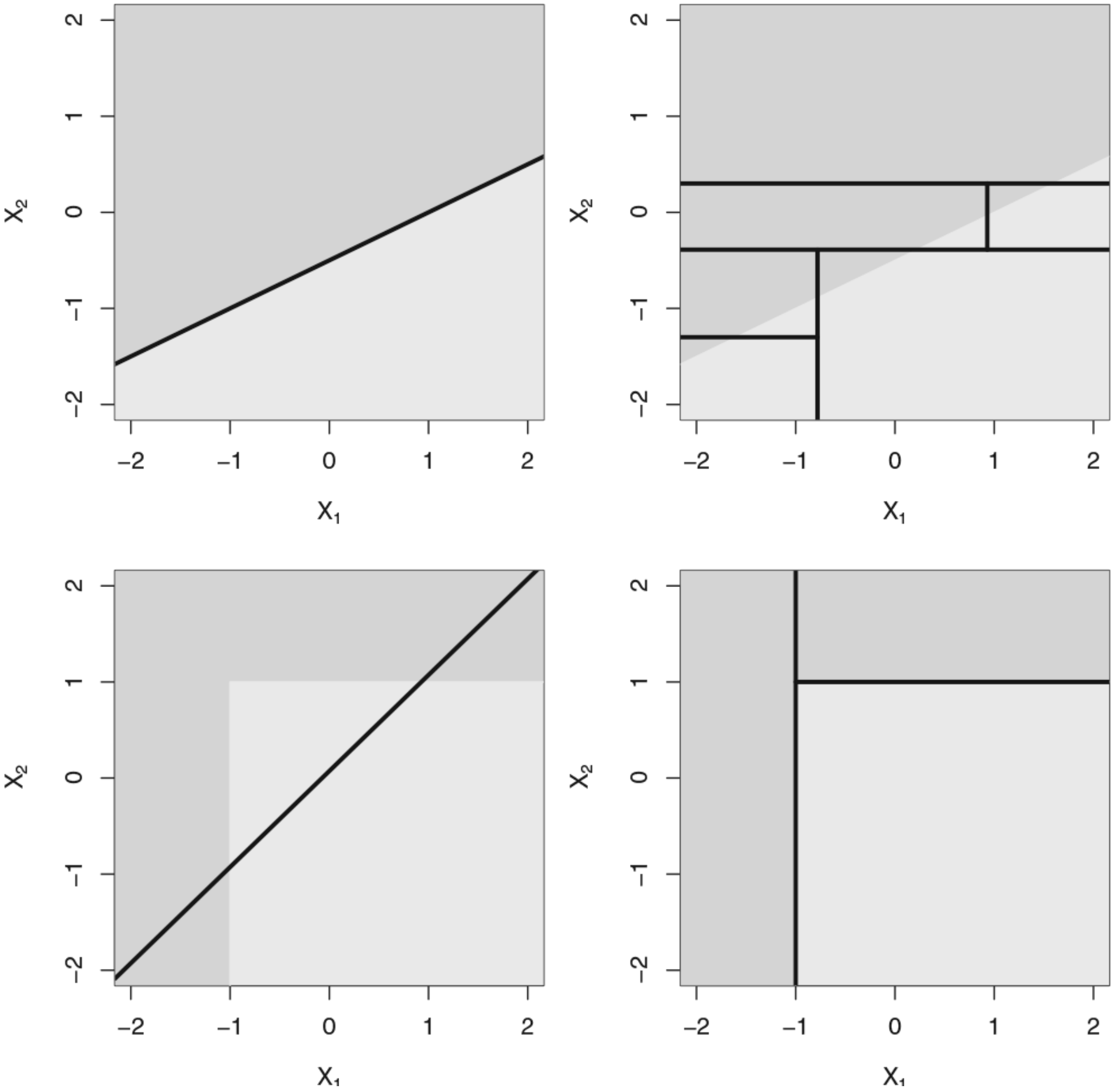


Figure 1: Linear boundary ISL Figure 8.7

In the classification tree (grown with Gini index) there were 78 leaves - maybe we want a more easy interpretable tree?

But, there are many possible pruned versions of our full tree. Can we investigate all of these?

Cost complexity pruning

We can prune the tree by using a algorithm called *cost complexity pruning.* We first build a large tree T_0 by recursive binary splitting. Then we try to find a subtree $T \subset T_0$ that (for a given value of α) minimizes

$$C_\alpha(T) = Q(T) + \alpha|T|,$$

where $Q(T)$ is our cost function, $|T|$ is the number of terminal nodes in tree T. The parameter α is then a parameter penalizing the number of terminal nodes, ensuring that the tree does not get too many branches.

We proceed by repeating the the process for the best subtree T, and this way we get a sequence of smaller of smaller subtrees where each tree is the best subtree of the previous tree.

For regression trees we choose $Q(T) = \sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2$, and or classification trees the entropy (deviance), Gini or misclassification rate.

Given a value of α we get a pruned tree (but the same pruned tree for ranges of α).

For $\alpha = 0$ we get T_0 and as α increases we get smaller and smaller trees.

Please study this note from Bo Lindqvist in MA8701 in 2017 - Advanced topics in Statistical Learning and Inference for an example of how we perform cost complexity pruning in detail. Alterntively, this method, with proofs, are given in B. D. Ripley (1996), Section 7.2.

Building a regression (classification) tree: Algorithm 8.1

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each k $= 1, \dots, K$:

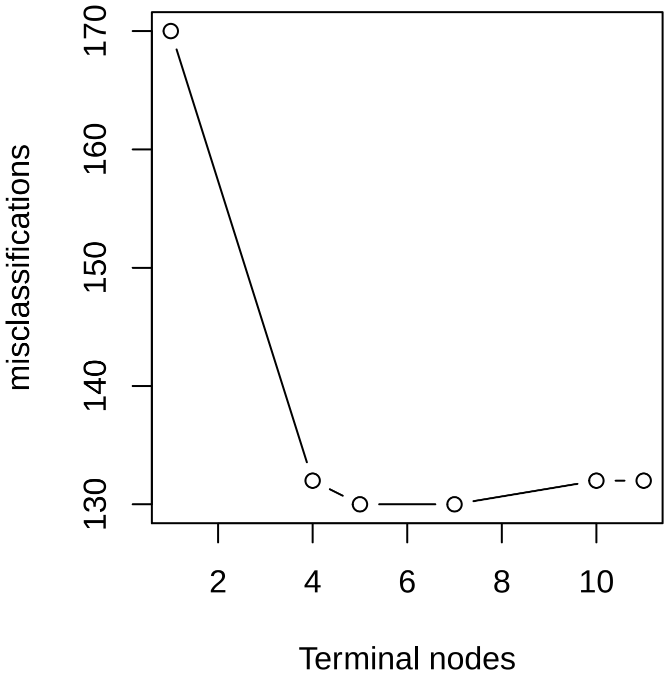
- Repeat Steps 1 and 2 on all but the kth fold of the training data.
- Evaluate the mean squared prediction (misclassification, gini, cross-entropy) error on the data in the left-out kth fold, as a function of α.
- Average the results for each value of α, and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Combining pruning and cross-validation to find optimal tree

We continue using the classification tree.

```
set.seed(1)
cv.head = cv.tree(tree.HIClass, FUN = prune.misclass)
par(pty = "s")
plot(cv.head$size, cv.head$dev, type = "b", xlab = "Terminal nodes",
    ylab = "misclassifications")
```



The function `cv.tree` automatically does 10-fold cross-validation. `dev` is here the number of misclassifications.

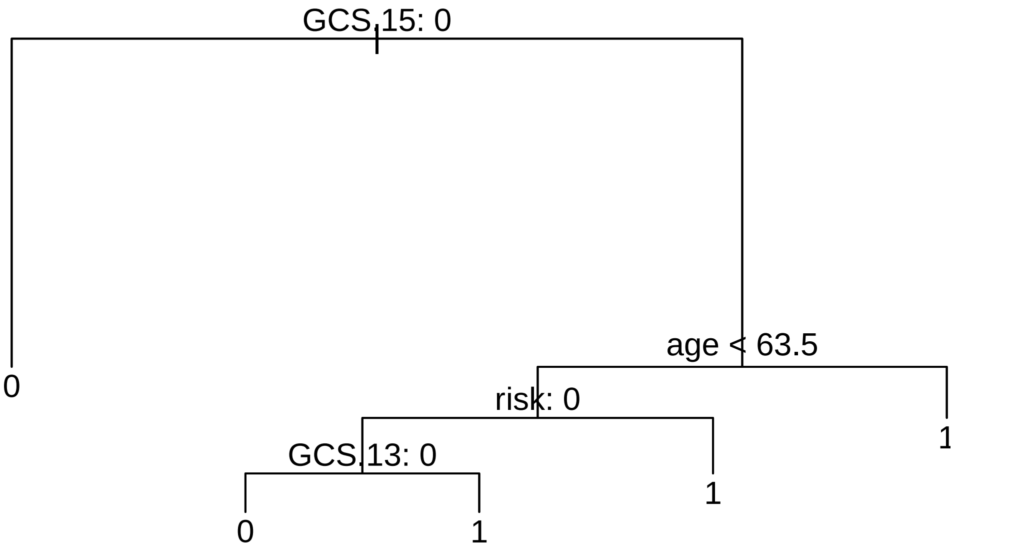
```
print(cv.head)
```

```
## $size
## [1] 11 10  7  5  4  1
##
## $dev
## [1] 132 132 130 130 132 170
##
## $k
## [1]     -Inf  0.00000  1.33333  3.50000  5.00000 12.33333
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

We have done cross-validation on our training set of 850 observations. According to the plot, the number of misclassifications is lowest if we use 5 terminal nodes. Next, we prune the classification tree according to this value:

```
prune.HIClass = prune.misclass(tree.HIClass, best = 5)
# Five node tree.
```

```r
plot(prune.HIClass)
text(prune.HIClass, pretty = 1)
```



We see that the new tree doesn't have any unnecessary splits, and we have a simple and interpretable decision tree. How is the predictive performance of the model affected?

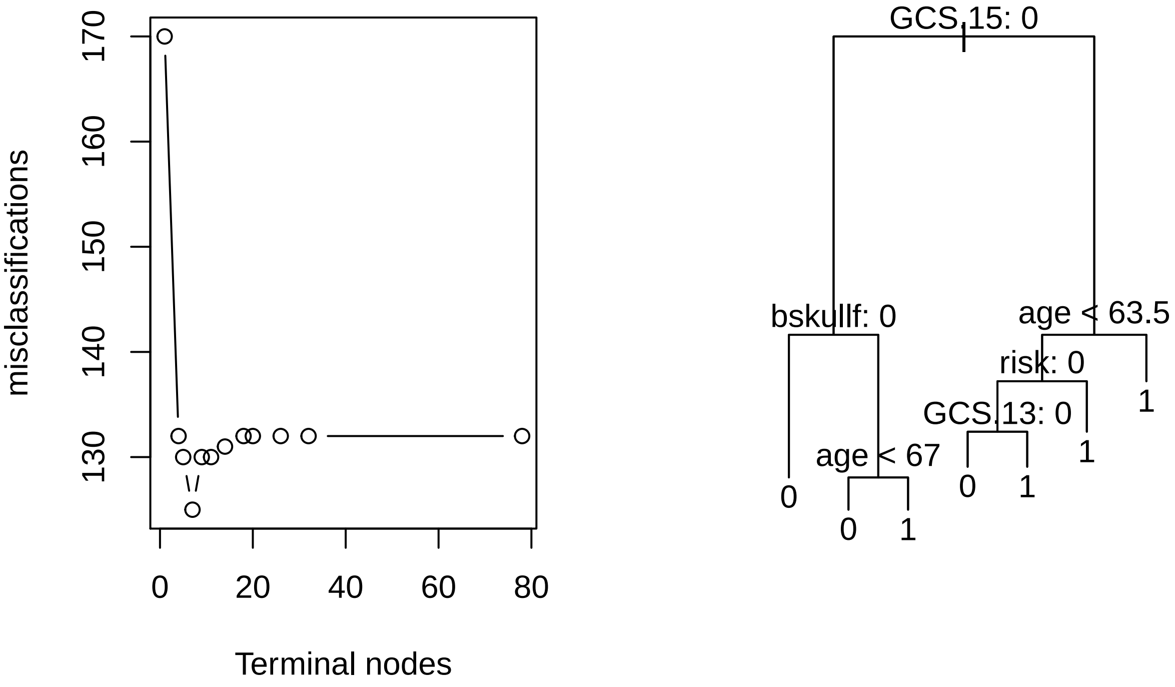
```r
tree.pred.prune = predict(prune.HIClass, headInjury2[test, ], type = "class")
confusionMatrix(tree.pred, headInjury2[test, ]$clinically.important.brain.injury)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 361  50
##          1  18  42
##
##                Accuracy : 0.856
##                  95% CI : (0.821, 0.886)
##     No Information Rate : 0.805
##     P-Value [Acc > NIR] : 0.00241
##
##                   Kappa : 0.471
##  Mcnemar's Test P-Value : 0.00017
##
##             Sensitivity : 0.953
##             Specificity : 0.457
##          Pos Pred Value : 0.878
##          Neg Pred Value : 0.700
##              Prevalence : 0.805
##          Detection Rate : 0.766
##    Detection Prevalence : 0.873
##       Balanced Accuracy : 0.705
##
##        'Positive' Class : 0
##
```

We see that the misclassification rate is as small as before indicating that the pruned tree is as good as the original tree for the test data.

The same repeated for the Gini-grown tree - comment on what is done.

```r
set.seed(1)
cv.headG = cv.tree(tree.HIClassG, FUN = prune.misclass)
par(pty = "m", mfrow = c(1, 2))
plot(cv.headG$size, cv.headG$dev, type = "b", xlab = "Terminal nodes",
    ylab = "misclassifications")

prune.HIClassG = prune.misclass(tree.HIClassG, best = 7)
plot(prune.HIClassG)
text(prune.HIClassG, pretty = 1)
```



```r
print(cv.headG)
```

```
## $size
##  [1] 78 32 26 20 18 14 11  9  7  5  4  1
##
## $dev
##  [1] 132 132 132 132 132 131 130 130 125 130 132 170
##
## $k
##  [1]      -Inf  0.000000  0.166667  0.333333  0.500000  0.750000  1.000000
##  [8]  1.500000  2.000000  3.500000  5.000000 12.333333
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

Questions:

Discuss the bias-variance tradeoff of a regression tree when increasing/decreasing the number of terminal nodes, i.e:

- What happens to the bias?
- What happens to the variance of a prediction if we reduce the tree size?

A:

As the tree size increase the bias will decrease, and the variance will increase. This is the same as any other method when we increase the model complexity.

From trees to forests

Advantages (+)

- Trees automatically select variables
- Tree-growing algorithms scale well to large n, growing a tree greedily
- Trees can handle mixed features (continouos, categorical) seamlessly, and can deal with missing data
- Small trees are easy to interpret and explain to people
- Some believe that decision trees mirror human decision making
- Trees can be displayed graphically

Disadvantages (-)

- Large trees are not easy to interpret
- Trees do not generally have good prediction performance (high variance)
- Trees are not very robust, a small change in the data may cause a large change in the final estimated tree

---

What is next?

- **Bagging**: grow many trees (from bootstrapped data) and average - to get rid of the non-robustness and high variance by averaging
- Variable importance plot - to see which variables make a difference (now that we have many trees).
- **Random forest**: inject more randomness (and even less variance) by just allowing a random selection of predictors to be used for the splits at each node.
- **Boosting**: make one tree, then another based on the residuals from the previous, repeat. The final predictor is a weighted sum of these trees.

But first,

Leo Breiman - the inventor of CART, bagging and random forests

Quotation from Wikipedia

Leo Breiman (January 27, 1928 – July 5, 2005) was a distinguished statistician at the University of California, Berkeley. He was the recipient of numerous honors and awards, and was a member of the United States National Academy of Science.

Breiman's work helped to bridge the gap between statistics and computer science, particularly in the field of machine learning. His most important contributions were his work on classification and regression trees and ensembles of trees fit to bootstrap samples. Bootstrap aggregation was given the name bagging by Breiman. Another of Breiman's ensemble approaches is the random forest.

From Breimans obituary

BERKELEY – Leo Breiman, professor emeritus of statistics at the University of California, Berkeley.

"It is trite to say so, but Leo Breiman was indeed a Renaissance man, and we shall miss him greatly," said Peter Bickel, professor of statistics and chairman this summer of UC Berkeley's statistics department.

Breiman retired in 1993, but as a Professor in the Graduate School, he continued to get substantial National Science Foundation grants and supervised three Ph.D. students. Bickel said that some of Breiman's best work was done after retirement.

"In particular," said Bickel, "he developed one of the most successful state-of-the-art classification programs, 'Random Forest.' This method was based on a series of new ideas that he developed in papers during the last seven years, and it is extensively used in government and industry."

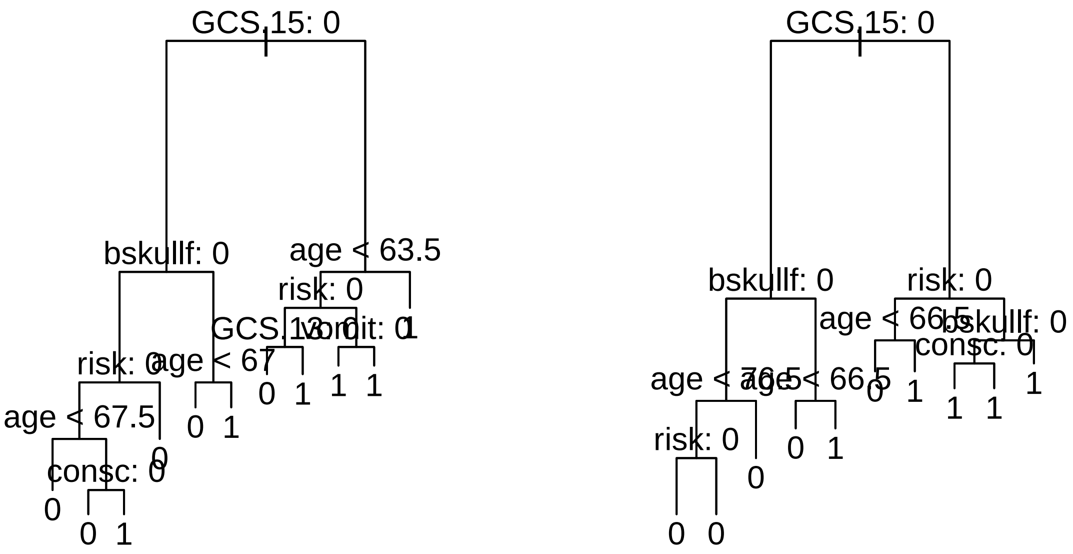
Breiman's best known work is considered to be "Classification and Regression Trees," a work in collaboration with three other scholars that facilitates practical applications, such as the diagnosis of diseases, from a multitude of symptoms.

Bagging

Decision trees often suffer from high variance. By this we mean that the trees are sensitive to small changes in the predictors: If we change the observation set, we may get a very different tree.

Let's draw a new training set for our data and see what happens if we fit our full classification tree (deviance grown).

```r
set.seed(33)
N = dim(headInjury2)[1]
train2 = sample(1:N, 850)  #We draw a new training sample. The number of observations is the same as be
tree.HIClass2 = tree(clinically.important.brain.injury ~ ., data = headInjury2,
    subset = train2)
par(mfrow = c(1, 2))
# short=c('amnesia','bskullf','GCSdecr','GCS.13','GCS.15','risk','consc','oskullf','vomit','brain','age
plot(tree.HIClass)
text(tree.HIClass, pretty = 0)
plot(tree.HIClass2)
text(tree.HIClass2, pretty = 0)
```



This classification tree is constructed by using 850 observations, just like the tree in the classification trees section, but we get two different trees that will give different predictions for a test set.

To reduce the variance of decision trees we can apply *bootstrap aggregating* (*bagging*), invented by Leo Breiman in 1996 (see references).

Independent data sets

Assume we have B i.i.d. observations of a random variable X each with the same mean and with variance σ^2. We calculate the mean $\bar{X} = \frac{1}{B}\sum_{b=1}^{B} X_b$. The variance of the mean is

$$\text{Var}(\bar{X}) = \text{Var}\Big(\frac{1}{B}\sum_{b=1}^{B} X_b\Big) = \frac{1}{B^2}\sum_{b=1}^{B}\text{Var}(X_b) = \frac{\sigma^2}{B}.$$

By averaging we get reduced variance. This is the basic idea!

But, we will not draw random variables - we want to fit decision trees: $\hat{f}_1(\mathbf{x}), \hat{f}_2(\mathbf{x}), \ldots, \hat{f}_B(\mathbf{x})$ and average those.

$$\hat{f}_{avg}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(\mathbf{x})$$

However, we do not have many independent data set - so we use *bootstrapping* to construct B data sets.

Bootstrapping (from Module 5)

Problem: we want to draw samples from a population with distribution f.

But: we do not know f and do not have a population to draw from, we only have our one sample.

Solution: we may use our sample as an empirical estimate for the distribution f - by assuming that each sample point has probability $1/n$ for being drawn.

Therefore: we draw with replacement n observations from our sample - and that is our first *bootstrap sample*.

We repeat this B times and get B bootstrap samples - that we use as our B data sets.

Bootstrap samples and trees

For each bootstrap sample we construct a decision tree, $\hat{f}^{*b}(x)$ with $b = 1, ..., B$, and we then use information from all of the trees to draw inference.

For a regression tree, we take the average of all of the predictions and use this as the final result:

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

For a classification tree we record the predicted class (for a given observation x) for each of the B trees and use the most occurring classification (majority vote) as the final prediction - or alternatively average posterior probabilities for each class.

Originally, Breiman (1996) suggested to prune each tree, but later research has found that it is better to leave the trees at maximal size (a bushy tree), to make the trees as different from each other as possible.

The number B is chosen to be as large as “necessary”. An increase in B will not lead to overfitting, and B is not regarded as a tuning parameter. If a goodness of fit measure is plotted as a function of B (soon) we see that (given that B is large enough) increasing B will not change the goodness of fit measure.

But first, a smart way to avoid doing cross-validation.

Out-of-bag error estimation

- We use a subset of the observations in each bootstrap sample. From Module 5 we know that the probability that an observation is in the bootstrap sample is approximately $1 - e^{-1}$=0.632121 (approximately 2/3).
- when an observation is left out of the bootstrap sample it is not used to build the tree, and we can use this observation as a part of a “test set” to measure the predictive performance and error of the fitted model, $f^{*b}(x)$.

In other words: Since each observation i has a probability of approximately 2/3 to be in a bootstrap sample, and we make B bootstrap samples, then observation i will be outside the bootstrap sample in approximately $B/3$ of the fitted trees.

The observations left out are referred to as the *out-of-bag* observations, and the measured error of the $B/3$ predictions is called the *out-of-bag error*.

Example

We can do bagging by using the function *randomForest()* in the *randomForest* library.

```r
library(randomForest)
set.seed(1)
bag = randomForest(clinically.important.brain.injury ~ ., data = headInjury2,
    subset = train, mtry = 10, ntree = 500, importance = TRUE)
bag$confusion
```

```
##     0  1 class.error
## 0 644 48   0.0693642
## 1  85 73   0.5379747
```

```r
1 - sum(diag(bag$confusion))/sum(bag$confusion[1:2, 1:2])
```

```
## [1] 0.156471
```

The variable *mtry=10* because we want to consider all 10 predictors in each split of the tree. The variable $ntree = 500$ because we want to average over 500 trees.

Predictive performance of the bagged tree on unseen test data:

```r
yhat.bag = predict(bag, newdata = headInjury2[test, ])
misclass.bag = table(yhat.bag, headInjury2[test, ]$clinically.important.brain.injury)
print(misclass.bag)
```

```
##
## yhat.bag   0   1
##        0 351  47
##        1  28  45
```

```r
1 - sum(diag(misclass.bag))/(sum(misclass.bag))
```

```
## [1] 0.159236
```

We note that the misclassification rate has increased slightly for the bagged tree (as compared to our previous full and pruned tree). **In other examples an improvement is very often seen.**

Prediction by majority vote vs. by averaging the probabilities

Consider the case when you have grown B classification tree with a binary response with classes 0 and 1. You might wonder which approach to choose to make a final prediction: majority vote or an average of the probabilities? Or would the prediction be the same in each case?

The difference between these two procedures can be compared to the difference between the mean value and median of a set of numbers. If we average the probabilities and make a classification thereafter, we have the mean value. If we sort all of our classifications, so that the classifications corresponding to one class would be lined up after each other, followed by the classifications corresponding to the other class we obtain the median value.

We examine this by an example:

Suppose we have $B = 5$ (no, B should be higher - this is only for illustration) classification tree and have obtained the following 5 estimated probabilities: $\{0.4, 0.4, 0.4, 0.4, 0.9\}$. If we average the probabilities, we get 0.5, and if we use a cut-off value of 0.5, our predicted class is 1. However, if we take a majority vote, using the same cut of value, the predicted classes will be $\{0, 0, 0, 0, 1\}$. The predicted class, based on a majority vote, would accordingly be 0.

The two procedures thus have their pros and cons: By averaging the predictions no information is lost. We do not only get the final classification, but the probability for belonging to the class 0 or 1. However, this method is not robust to outliers. By taking a majority vote, outliers have a smaller influence on the result.

When should we use bagging?

Bagging can be used for predictors (regression and classification) that are not trees, and according to Breiman (1996)

- the vital element is the instability of the prediction method
- if perturbing the learning set can cause significant changes in the predictor constructed, then bagging can improve accuracy.

Breiman (1996) suggests that these methods should be suitable for bagging:

- neural nets, classification and regression trees, subset selection in linear regression

however not nearest neigbours - since

- the stability of nearest neighbour classification methods with respect to perturbations of the data distinguishes them from competitors such as trees and neural nets.

Variable importance plots

Bagging is an example of an *ensemble method*, so is boosting and random forests (to come next). For all of these methods many trees are grown and combined, and the predictive power can be highly improved. However, this comes at a cost of interpretability. Instead of having one tree, the resulting model consists of B trees, where B often is 300 or 500 (or maybe even 5000 when boosting).

Variable importance plots show *the relative importance of the predictors:* the predictors are sorted according to their importance, such that the top variables have a higher importance than the bottom variables. There are in general two types of variable importance plots:

- variable importance based on decrease in node impurity and
- variable importance based on randomization.

Variable importance based on node impurity

The term *important* relates to *total decrease in the node impurity, over splits for a predictor*, and is defined differently for regression trees and classification trees.

Regression trees:

- The importance of each predictor is calculated using the RSS.
- The algorithm records the total amount that the RSS is decreased due to splits for each predictor (there may be many spits for one predictor for each tree).
- This decrease in RSS is then averaged over the B trees. The higher the decrease, the more important the predictor.

Classification trees:

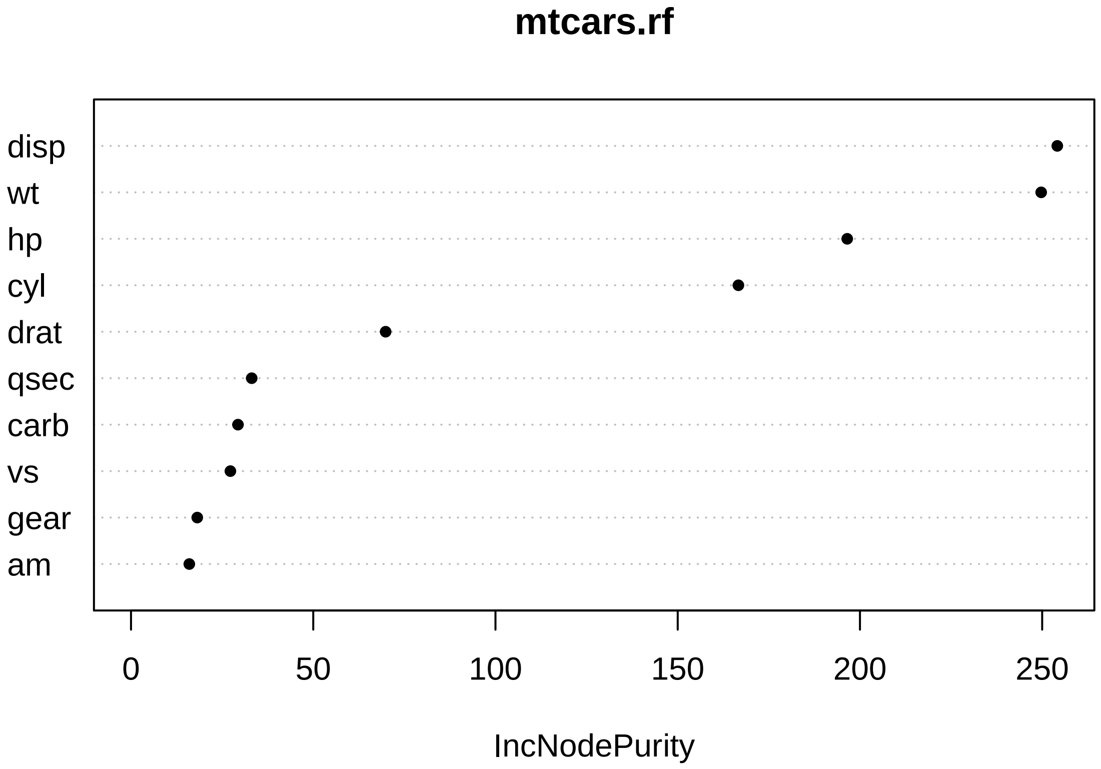
- The importance of each predictor is calculated using the Gini index.
- The importance is the mean decrease (over all B trees) in the Gini index by splits of a predictor.

R: `varImpPlot` (or `importance`) in `randomForest` with `type=2`.

Auto data example

```r
set.seed(4268)
data(mtcars)
mtcars.rf <- randomForest(mpg ~ ., data = mtcars, ntree = 1000, keep.forest = FALSE,
    importance = TRUE)
```

```r
varImpPlot(mtcars.rf, type = 2, pch = 20)
```



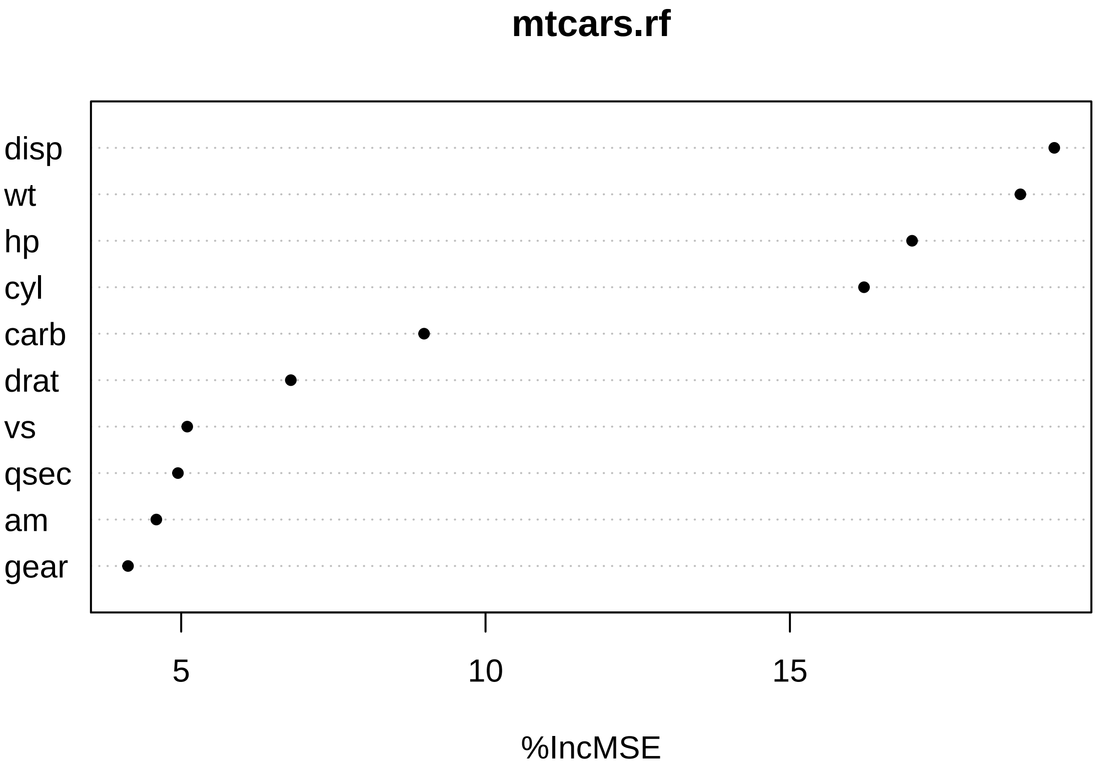
Variable importance based on randomization

Variable importance based on randomization is calculated using the OOB sample.

- Computations are carried out for one bootstrap sample at a time.
- Each time a tree is grown the OOB sample is used to test the predictive power of the tree.
- Then for one predictor at a time, repeat the following:
    - permute the OOB observations for the jth variable x_j and calculate the new OOB error.
    - If x_j is important, permuting its observations will decrease the predictive performance.
- The difference between the two is averaged over all trees (and normalized by the standard deviation of the differences).

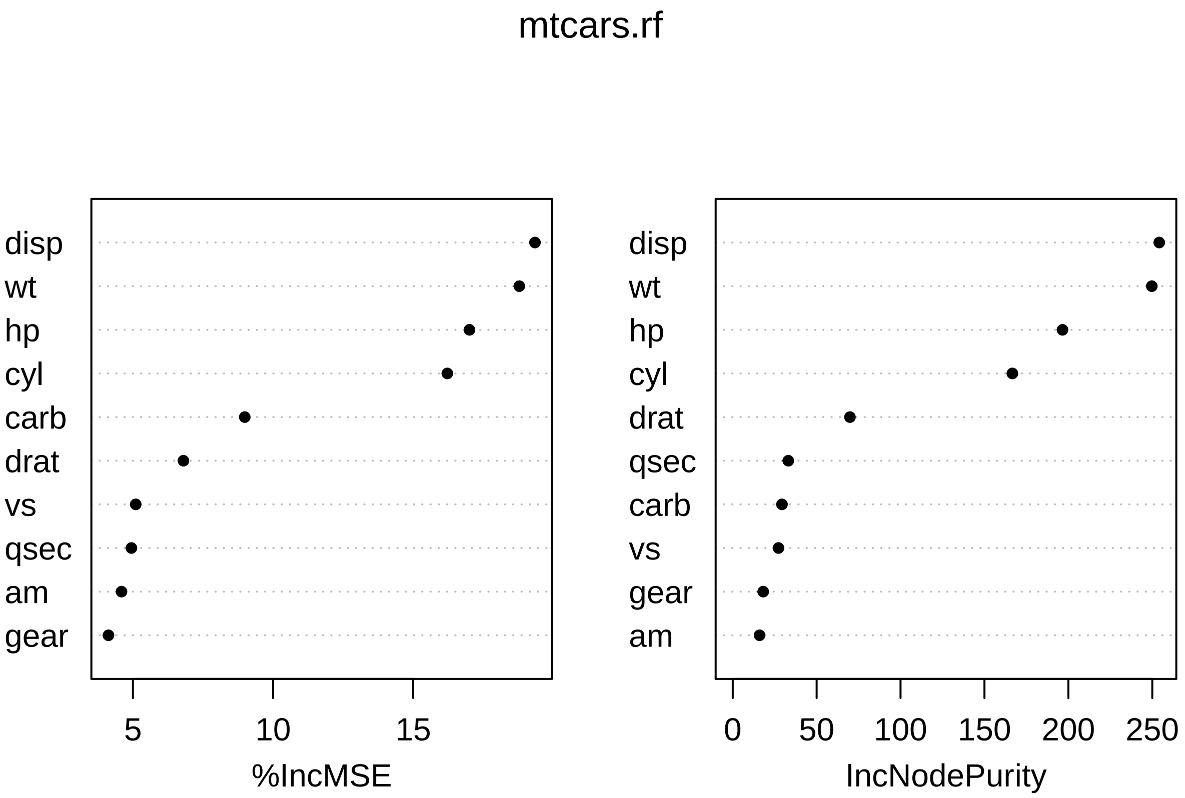
R: `varImpPlot` (or `importance`) in `randomForest` with `type=1`.

```r
varImpPlot(mtcars.rf, type = 1, pch = 20)
```



The two types together: do they agree?

```r
varImpPlot(mtcars.rf, pch = 20)
```



Random Forest

If there is a strong predictor in the dataset, the decision trees produced by each of the bootstrap samples in the bagging algorithm becomes very similar: Most of the trees will use the same strong predictor in the top split.

We have seen this for our example trees for the minor head injury example, the predictor *GCS.15.2hours* was chosen in the top split every time. This is probably the case for a large amount of the bagged trees as well.

This is not optimal, because we get B trees that are highly correlated. We don't get a large reduction in variance by averaging $\hat{f}^{*b}(x)$ when the correlation between the trees is high. In the previous section we actually saw a (marginal) decrease in the predictive performance for the bagged tree compared to the pruned tree and the full tree.

Random forests is a solution to this problem and a method for decorrelating the trees.

The effect of correlation on the variance of the mean

The variance of the average of B observations of i.i.d random variables X, each with variance σ^2 is $\frac{\sigma^2}{B}$. Now, suppose we have B observations of a random variable X which are identically distributed, each with mean μ and variance σ^2, but not independent.

That is, suppose the variables have a positive correlation ρ

$$\text{Cov}(X_i, X_j) = \rho\sigma^2, \quad i \neq j.$$

The variance of the average is

$$
\begin{aligned}
\text{Var}(\bar{X}) &= \text{Var}\Big(\frac{1}{B}\sum_{i=1}^{B} X_i\Big) \\
&= \sum_{i=1}^{B} \frac{1}{B^2}\text{Var}(X_i) + 2\sum_{i=2}^{B}\sum_{j=1}^{i-1}\frac{1}{B}\frac{1}{B}\text{Cov}(X_i, X_j) \\
&= \frac{1}{B}\sigma^2 + 2\frac{B(B-1)}{2}\frac{1}{B^2}\rho\sigma^2 \\
&= \frac{1}{B}\sigma^2 + \rho\sigma^2 - \frac{1}{B}\rho\sigma^2 \\
&= \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \\
&= \frac{1-(1-B)\rho}{B}\sigma^2
\end{aligned}
$$

Check: $\rho = 0$ and $\rho = 1$? (Most negative values of ρ will not give a positive definite covariance matrix. The covariance matrix is positive definite if $\rho > -1/(B-1)$.)

The idea behind random forests is to *improve the variance reduction of bagging* by reducing the correlation between the trees.

The procedure is thus as in bagging, but with the important difference, that

- at each split we are only allowed to consider $m < p$ of the predictors.

A new sample of m predictors is taken at each split and

- typically $m \approx \sqrt{p}$ (classificaton) and $m = p/3$ (regression)

The general idea is at for very correlated predictors m is chosen to be small.

The number of trees, B, is not a tuning parameter, and the best is to choose it large enough.

If B is sufficiently large (three times the number needed for the random forest to stabilize), the OOB error estimate is equvalent to LOOCV (Efron and Hastie, 2016, p 330).

Example

We decorrelate the trees by using the *randomForest()* function again, but this time we set *mtry=3*. This means that the algorithm only considers three of the predictors in each split. We choose 3 because we have 10 predictors in total and $\sqrt{10} \approx 3$.

```
set.seed(1)

rf = randomForest(clinically.important.brain.injury ~ ., data = headInjury2,
    subset = train, mtry = 3, ntree = 500, importance = TRUE)
```

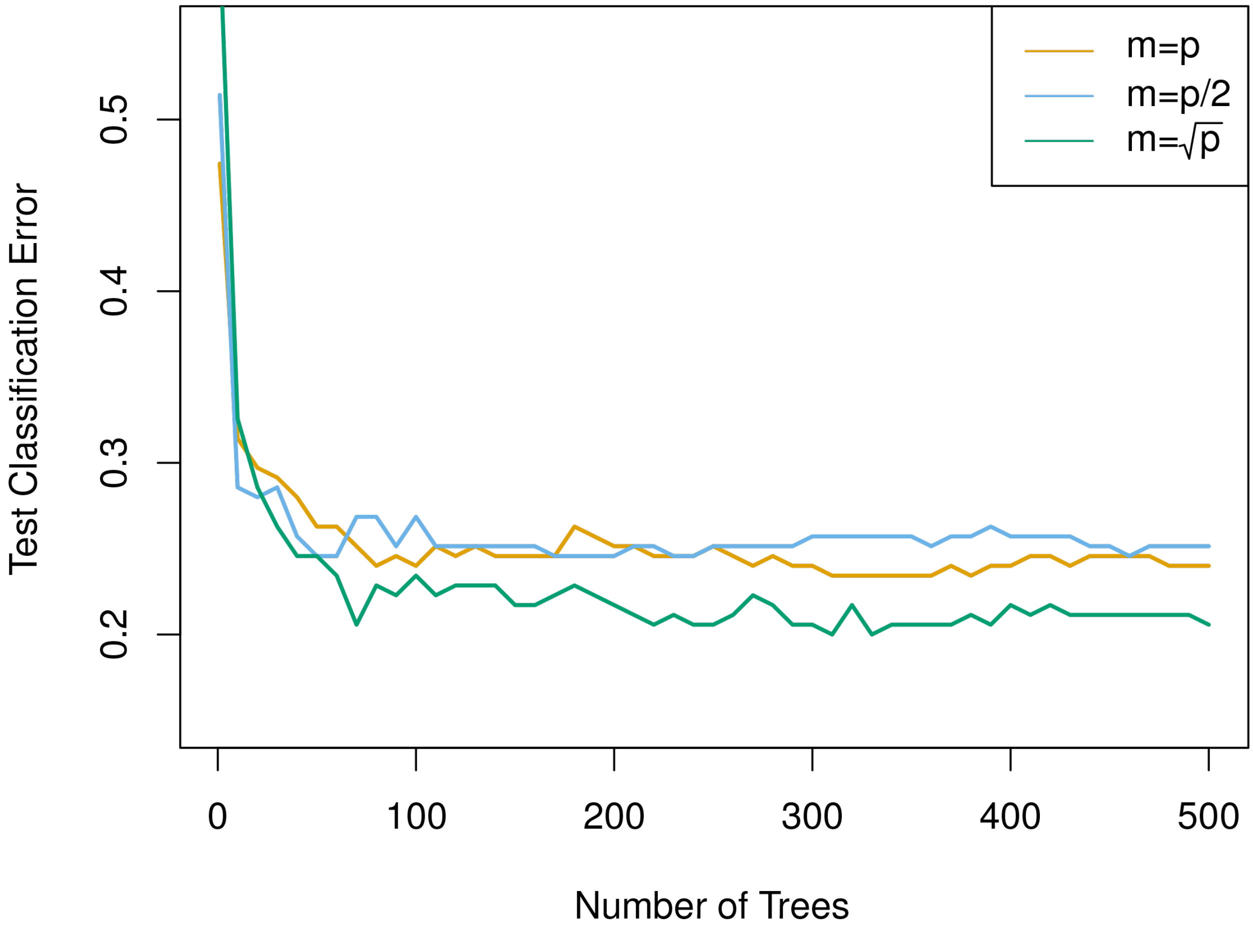


Figure 2: ISLR Figure 8.10, gene expression data set with 15 classes and 500 predictors

We check the predictive performance as before:

```
rf$confusion
```

```
##     0  1 class.error
## 0 674 18   0.0260116
## 1  99 59   0.6265823
```

```
1 - sum(diag(rf$confusion[1:2, 1:2]))/(sum(rf$confusion[1:2, 1:2]))
```

```
## [1] 0.137647
```

```
yhat.rf = predict(rf, newdata = headInjury2[test, ])

misclass.rf = table(yhat.rf, headInjury2[test, ]$clinically.important.brain.injury)
print(misclass.rf)
```

```
##
## yhat.rf   0   1
##       0 368  53
##       1  11  39
```

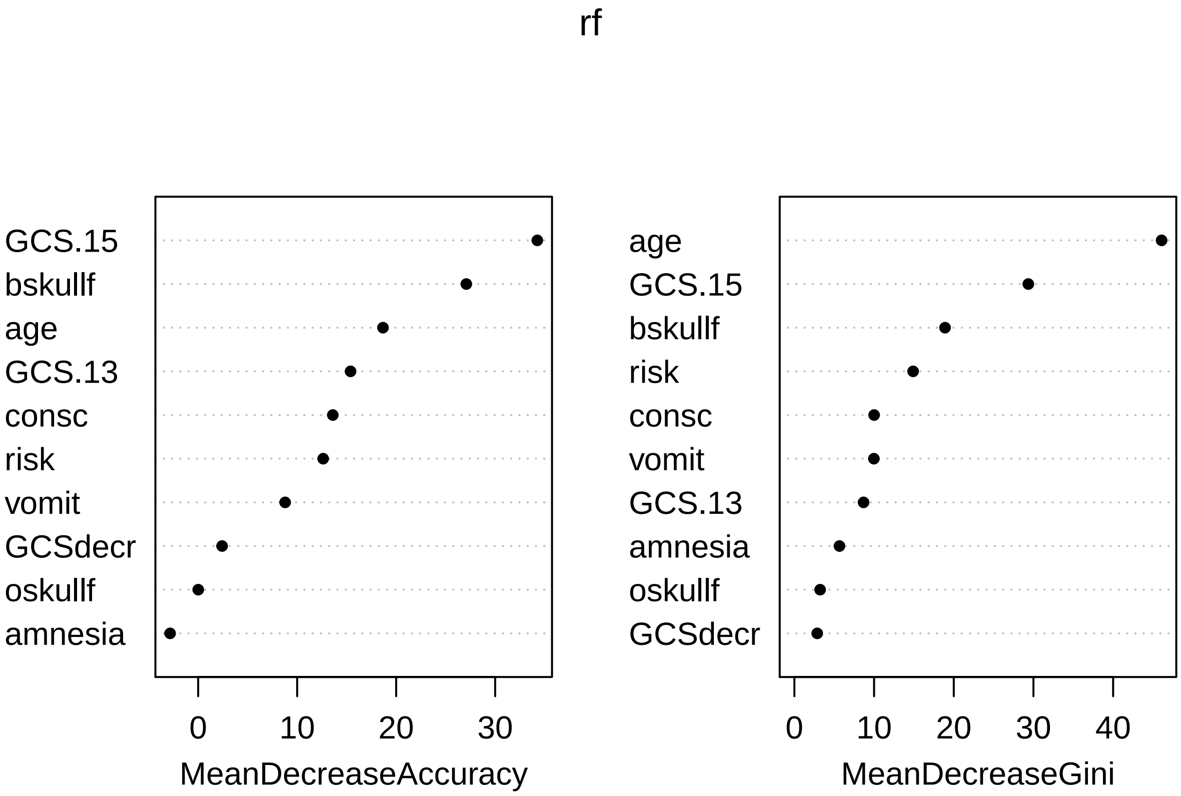
```
1 - sum(diag(misclass.rf))/(sum(misclass.rf))
```

```
## [1] 0.135881
```

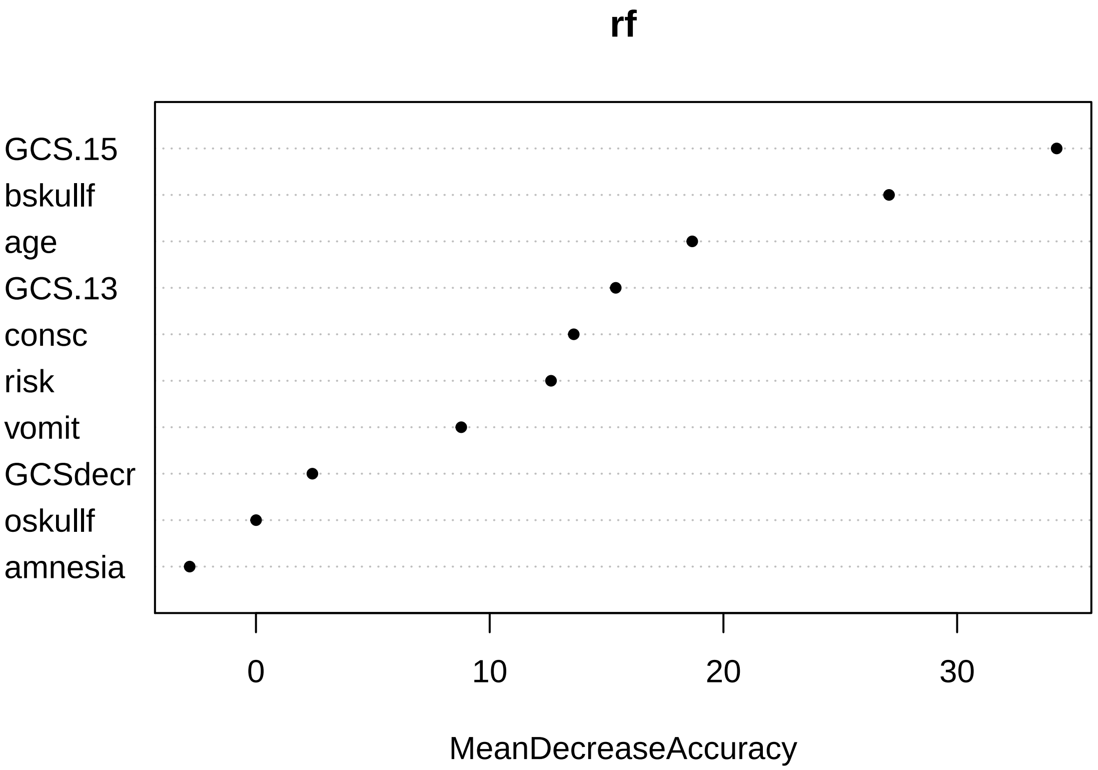
The misclassification rate is slightly decreased compared to the bagged tree (and to the pruned tree).

By using the *varImpPlot()* function we can study the importance of each predictor.

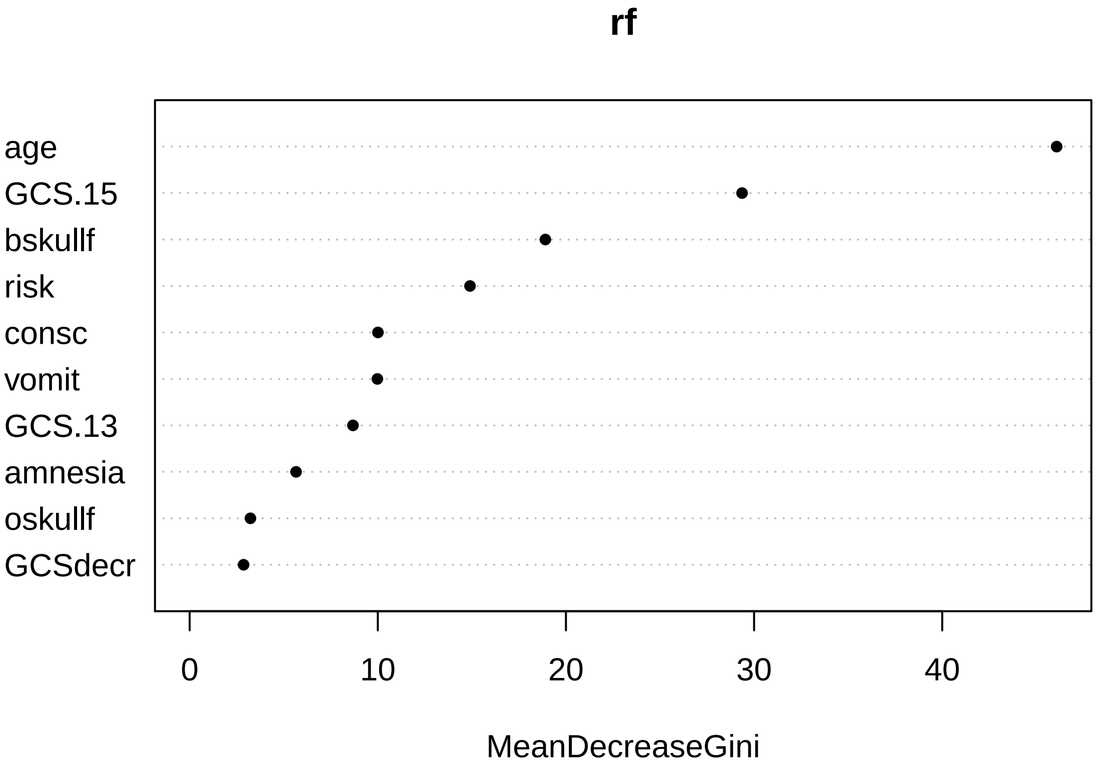
```
varImpPlot(rf, pch = 20)
```



```
varImpPlot(rf, pch = 20, type = 1)
```



```
varImpPlot(rf, pch = 20, type = 2)
```



As expected *GCS.15.2hours* is a strong predictor along with *basal.skull.facture* and *age*. This means that most of the trees will have these predictors in the top split.

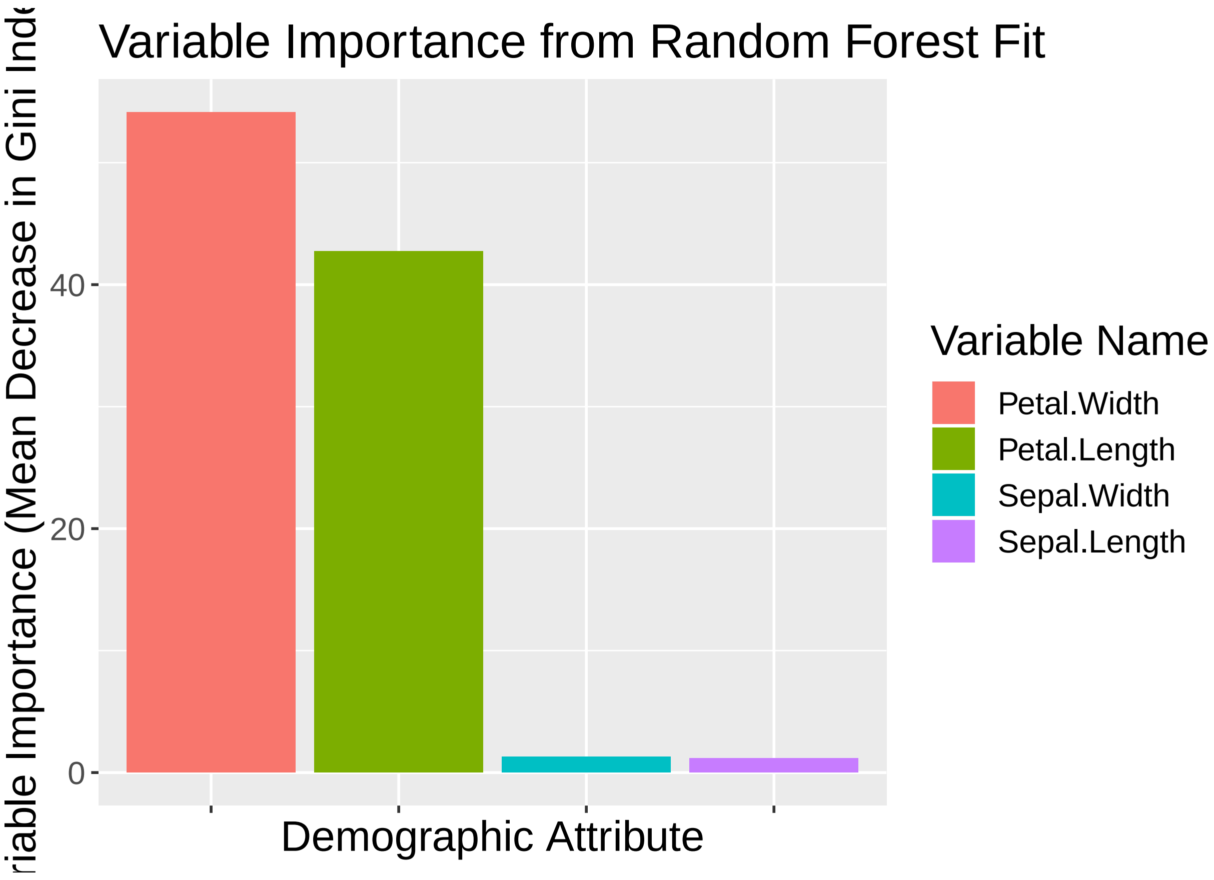
Iris example

Variable importance plot for bagging:

```r
# borrowed from https://gist.github.com/ramhiser/6dec3067f087627a7a85
library(dplyr)
library(ggplot2)
# library(ggpubr)
rf_out <- randomForest(Species ~ ., data = iris, mtry = 4)
rf_out$confusion
```

```
##            setosa versicolor virginica class.error
## setosa         50          0         0        0.00
## versicolor      0         47         3        0.06
## virginica       0          4        46        0.08
```

```r
# Extracts variable importance (Mean Decrease in Gini Index) Sorts by
# variable importance and relevels factors to match ordering
var_importance <- data_frame(variable = setdiff(colnames(iris), "Species"),
    importance = as.vector(importance(rf_out)))
var_importance <- arrange(var_importance, desc(importance))
var_importance$variable <- factor(var_importance$variable, levels = var_importance$variable)

p <- ggplot(var_importance, aes(x = variable, weight = importance, fill = variable))
p <- p + geom_bar() + ggtitle("Variable Importance from Random Forest Fit")
p <- p + xlab("Demographic Attribute") + ylab("Variable Importance (Mean Decrease in Gini Index)")
p <- p + scale_fill_discrete(name = "Variable Name")
p = p + theme(axis.text.x = element_blank(), axis.text.y = element_text(size = 12),
    axis.title = element_text(size = 16), plot.title = element_text(size = 18),
    legend.title = element_text(size = 16), legend.text = element_text(size = 12))
p
```



Iris example

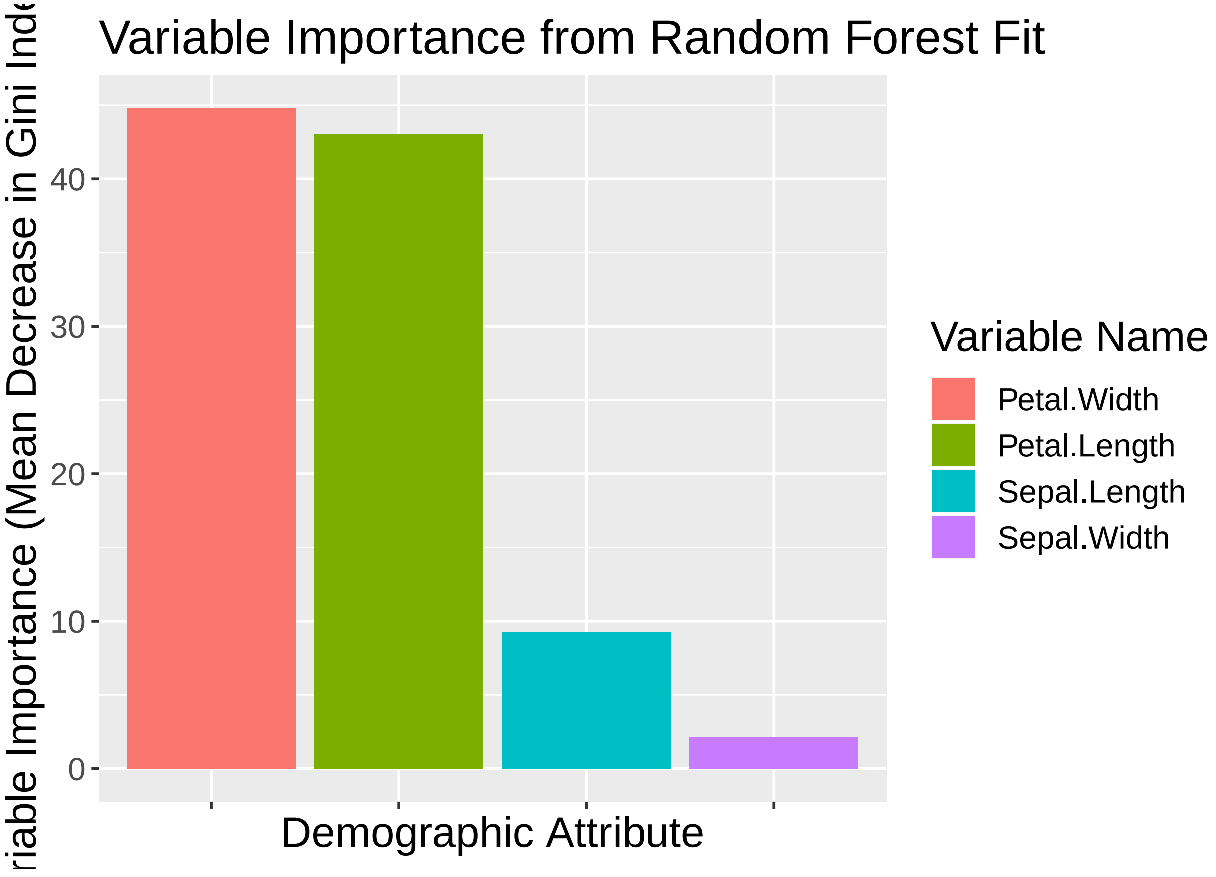
Variable importance plot for random forest:

```
# borrowed from https://gist.github.com/ramhiser/6dec3067f087627a7a85

rf_out = randomForest(Species ~ ., data = iris)
rf_out$confusion
```

```
##            setosa versicolor virginica class.error
## setosa         50          0         0        0.00
## versicolor      0         47         3        0.06
## virginica       0          5        45        0.10
```

```
var_importance <- data_frame(variable = setdiff(colnames(iris), "Species"),
    importance = as.vector(importance(rf_out)))
var_importance <- arrange(var_importance, desc(importance))
var_importance$variable <- factor(var_importance$variable, levels = var_importance$variable)

pp <- ggplot(var_importance, aes(x = variable, weight = importance, fill = variable))
pp <- pp + geom_bar() + ggtitle("Variable Importance from Random Forest Fit")
pp <- pp + xlab("Demographic Attribute") + ylab("Variable Importance (Mean Decrease in Gini Index)")
pp <- pp + scale_fill_discrete(name = "Variable Name")
pp = pp + theme(axis.text.x = element_blank(), axis.text.y = element_text(size = 12),
    axis.title = element_text(size = 16), plot.title = element_text(size = 18),
    legend.title = element_text(size = 16), legend.text = element_text(size = 12))
```

```
pp
```



Boosting

Boosting is an alternative approach for improving the predictions resulting from a decision tree. We will only consider the description of boosting regression trees (and not classification trees) in this course.

In boosting the trees are grown *sequentially* so that each tree is grown using information from the previous tree.

- First build a decision tree with d splits (and $d + 1$ terminal notes).
- Next, improve the model in areas where the model didn't perform well. This is done by fitting a decision tree to the *residuals of the model*. This procedure is called *learning slowly*.
- The first decision tree is then updated based on the residual tree, but with a weight.
- The procedure is repeated until some stopping criterion is reached. Each of the trees can be very small, with just a few terminal nodes (or just one split).

Algorithm 8.2: Boosting for regression trees

```
1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all $i$ in the training set.
2. For $b = 1, 2, ..., B$, repeat:
     a) Fit a tree $\hat{f}^b$ with $d$ splits ($d + 1$ terminal nodes) to the training data.
```

b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$

3. The boosted model is $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$.

Boosting has three tuning parameters which need to set, and can be found using cross-validation.

Tuning parameters

- The number of trees to be grown, B. The value of B could be chosen using cross-validation. A too small value of B would imply that much information is unused (remember that boosting is a slow learner), whereas a too large value of B may lead to overfitting.
- λ: This is a shrinkage parameter and controls the rate at which boosting learns. The role of λ is to scale the new information, when added to the existing tree. We add information from the b-th tree to our existing tree $\hat{f}$, but scaled by the λ. Choosing a small value for λ ensures that the algorithm learns slowly, but will require a larger tree ensemble. Typical values of λ is 0.1 or 0.01.
- Interaction depth d: The number of splits in each tree. This parameter controls the complexity of the boosted tree ensemble (the level of interaction between variables that we may estimate). By choosing $d = 1$ a tree stump will be fitted at each step and this gives an additive model.

Example: Boston data set

First - to get to know the data set we run through trees, bagging and random forests - before arriving at boosting. See also the ISLR book, Section 8.3.4.

Data

```
library(MASS)
library(tree)
set.seed(1)
train = sample(1:nrow(Boston), nrow(Boston)/2)
colnames(Boston)
```

```
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "black"   "lstat"   "medv"
```

```
head(Boston)
```

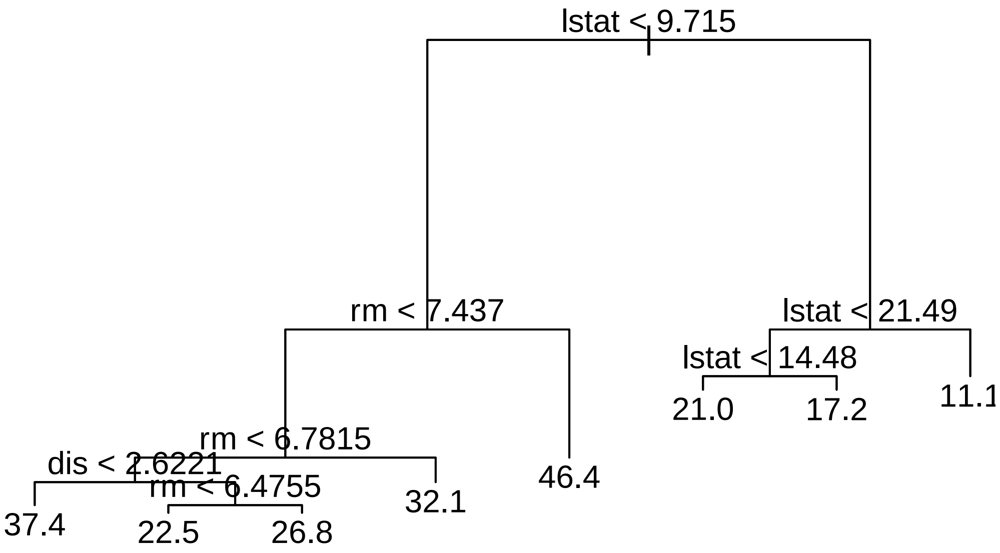
```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12
##   lstat medv
## 1  4.98 24.0
## 2  9.14 21.6
## 3  4.03 34.7
## 4  2.94 33.4
## 5  5.33 36.2
## 6  5.21 28.7
```

Regression tree

```r
tree.boston = tree(medv ~ ., Boston, subset = train)
summary(tree.boston)
```

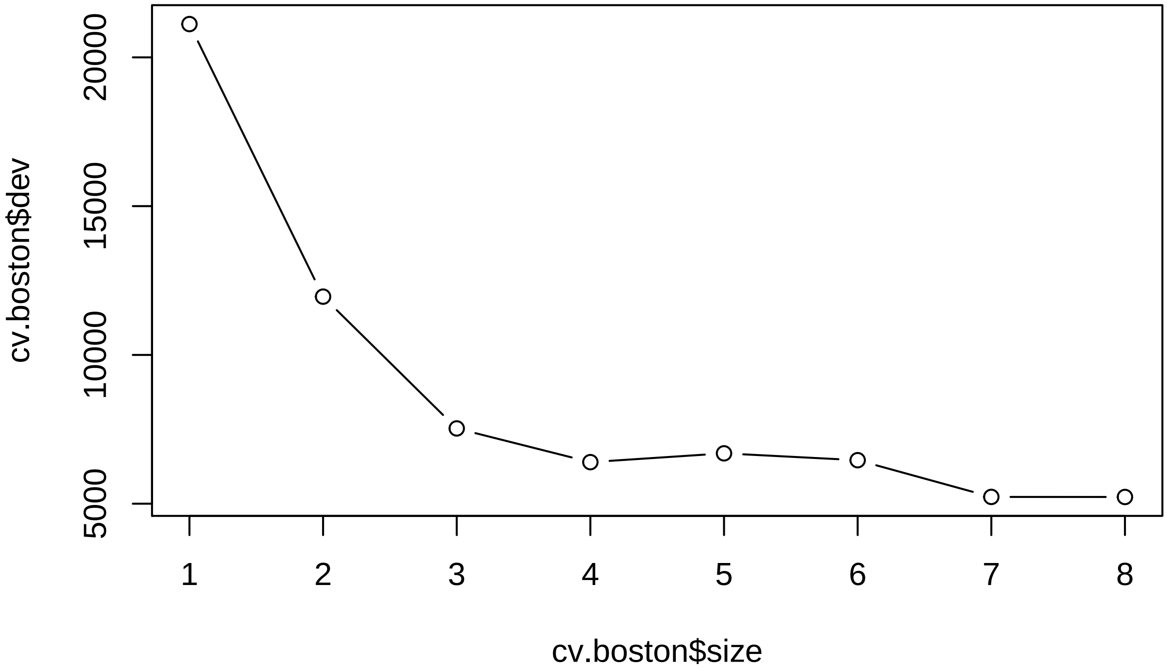
```
##
## Regression tree:
## tree(formula = medv ~ ., data = Boston, subset = train)
## Variables actually used in tree construction:
## [1] "lstat" "rm"    "dis"
## Number of terminal nodes:  8
## Residual mean deviance:  12.6 = 3100 / 245
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -14.1000  -2.0400  -0.0536   0.0000   1.9600  12.6000
```

```r
plot(tree.boston)
text(tree.boston, pretty = 0)
```



Need to prune?

```
cv.boston = cv.tree(tree.boston)
plot(cv.boston$size, cv.boston$dev, type = "b")
```

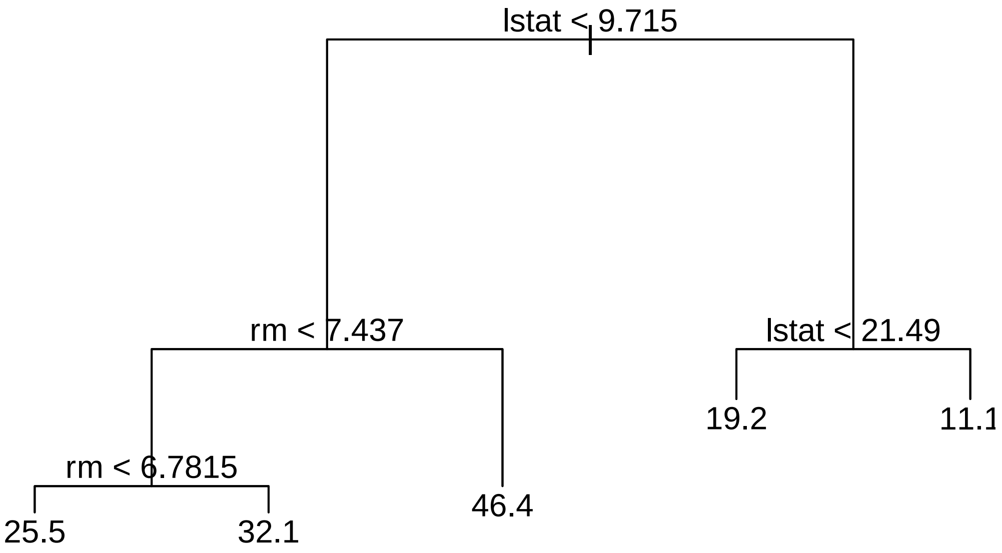


Most complex tree selected.

Pruning

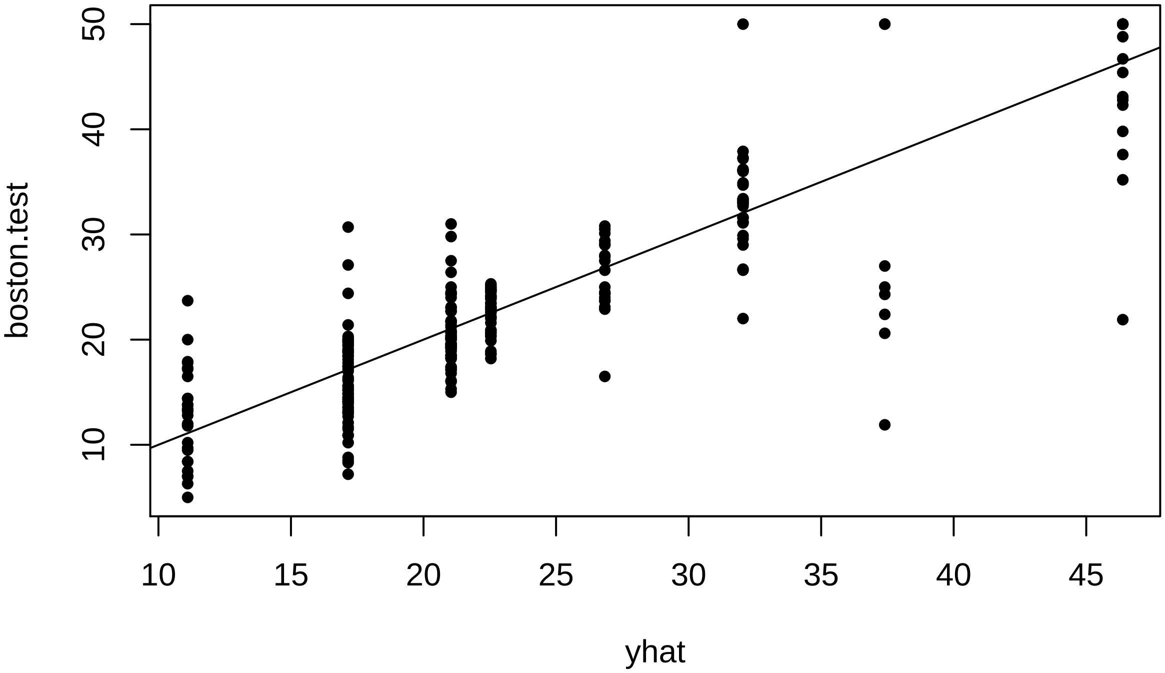
Just to show pruning (even if most complex tree was selected).

```
prune.boston = prune.tree(tree.boston, best = 5)
plot(prune.boston)
text(prune.boston, pretty = 0)
```



Test error for full tree

```
yhat = predict(tree.boston, newdata = Boston[-train, ])
boston.test = Boston[-train, "medv"]
plot(yhat, boston.test, pch = 20)
abline(0, 1)
```



```
mean((yhat - boston.test)^2)
```

```
## [1] 25.0456
```

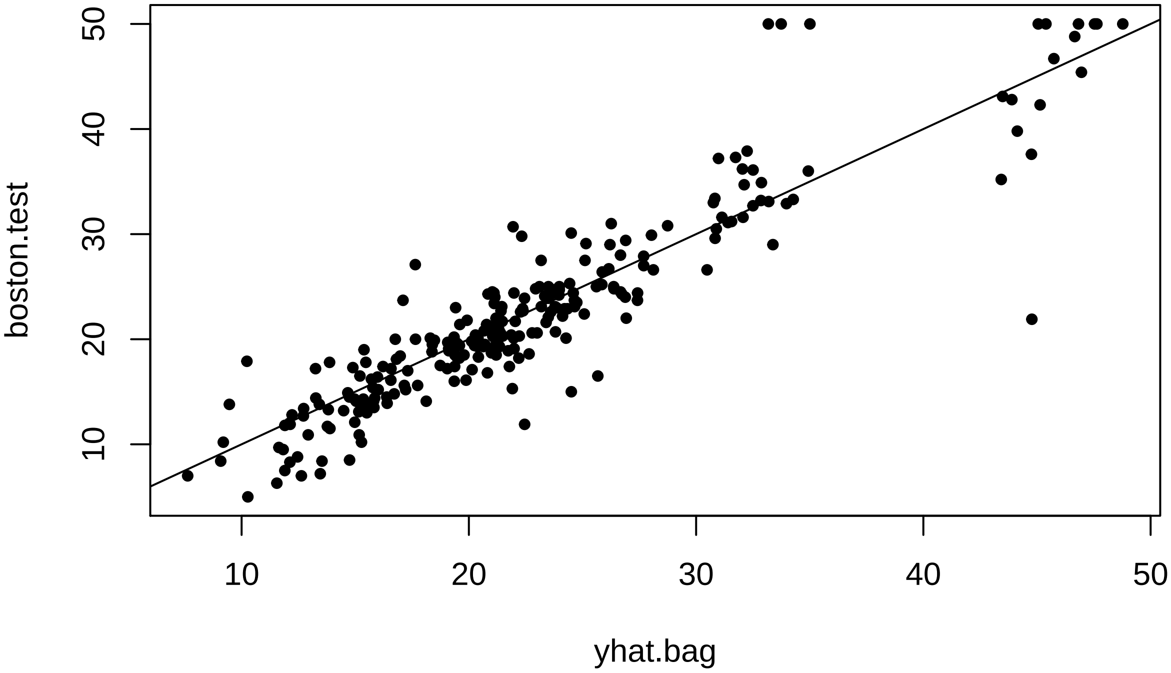
Bagging

```
library(randomForest)
set.seed(1)
bag.boston = randomForest(medv ~ ., data = Boston, subset = train, mtry = 13,
    importance = TRUE)
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: 11.1572
##                     % Var explained: 86.49
```

Plotting predicted test values vs true values.

```
yhat.bag = predict(bag.boston, newdata = Boston[-train, ])
plot(yhat.bag, boston.test, pch = 20)
abline(0, 1)
```



```
mean((yhat.bag - boston.test)^2)
```

```
## [1] 13.5081
```

Error rate on test set for bagging

Random forest

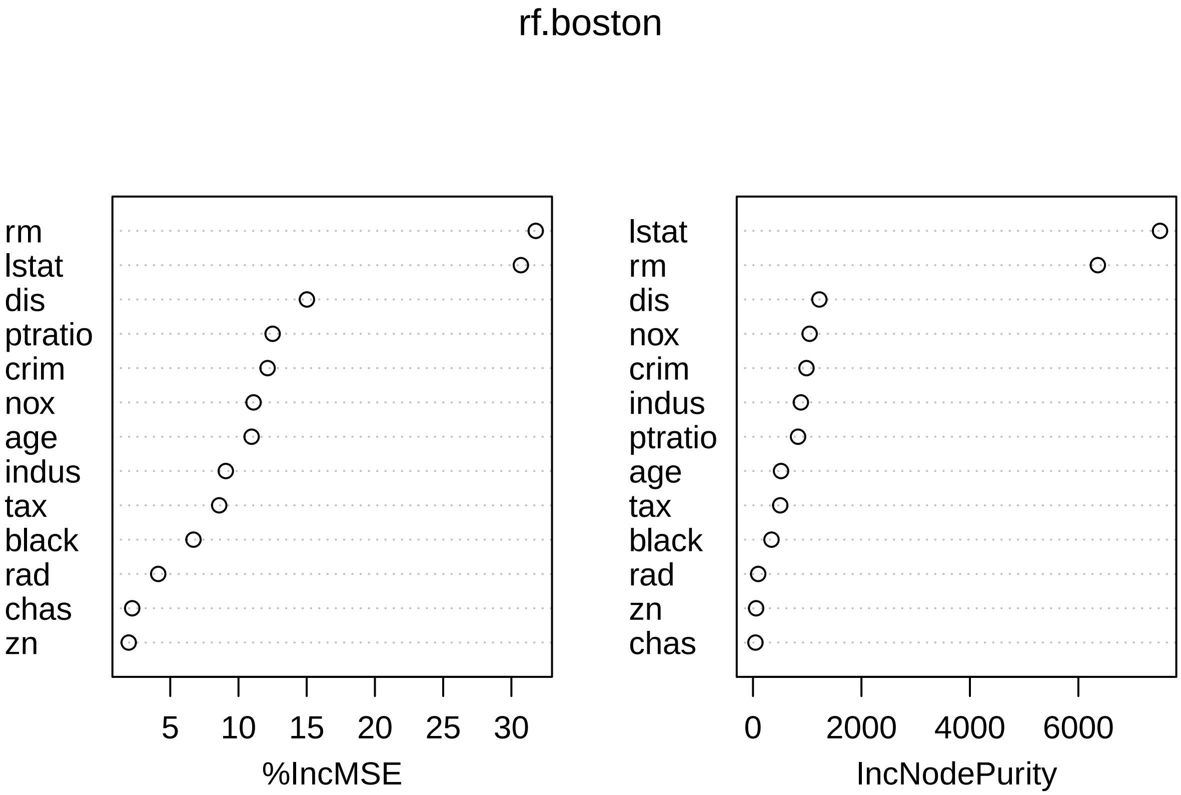
```
set.seed(1)
rf.boston = randomForest(medv ~ ., data = Boston, subset = train, mtry = 6,
    importance = TRUE)
yhat.rf = predict(rf.boston, newdata = Boston[-train, ])
mean((yhat.rf - boston.test)^2)
```

```
## [1] 11.6645
```

```
importance(rf.boston)
```

```
##           %IncMSE IncNodePurity
## crim     12.13232      986.5034
## zn        1.95558       57.9695
## indus     9.06930      882.7826
## chas      2.21083       45.2294
## nox      11.10482     1044.3378
## rm       31.78403     6359.3197
## age      10.96268      516.8297
## dis      15.01524     1224.1161
## rad       4.11801       95.9459
## tax       8.58793      502.9672
## ptratio 12.50390      830.7752
## black     6.70261      341.3036
## lstat    30.69522     7505.7394
```

```r
varImpPlot(rf.boston)
```



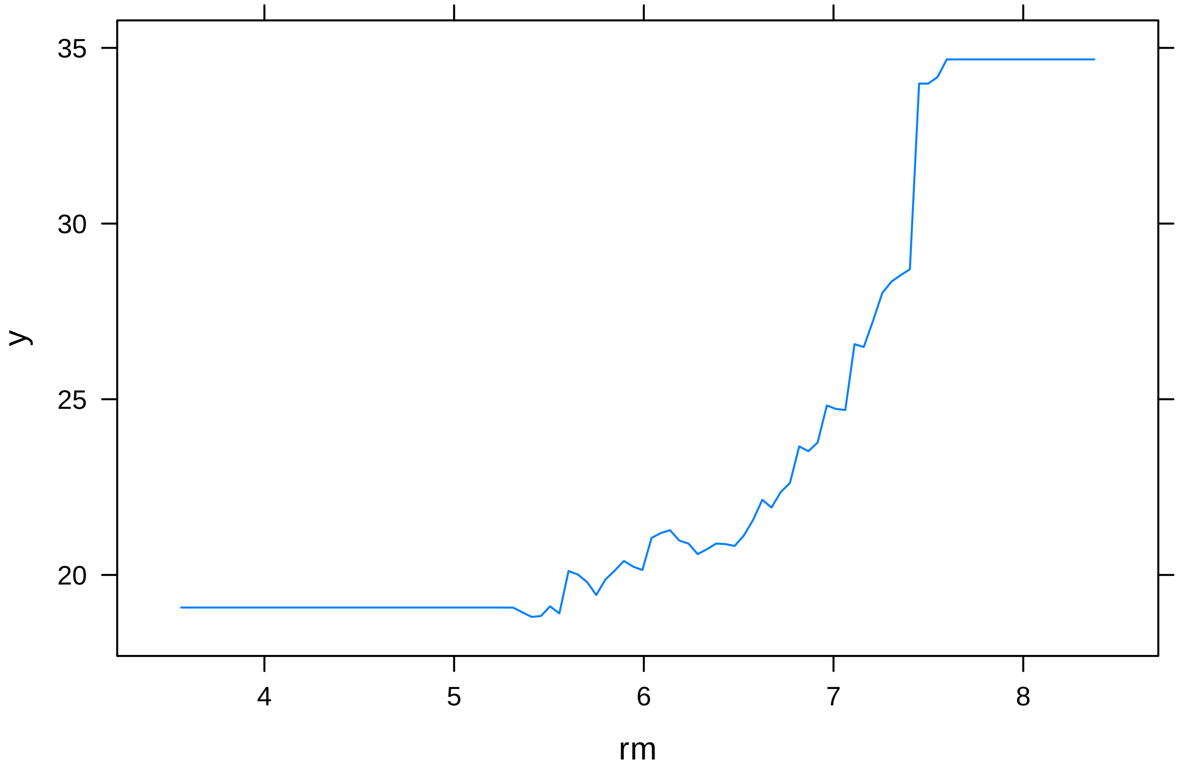
Finally, boosting Boston

```r
library(gbm)
set.seed(1)
boost.boston = gbm(medv ~ ., data = Boston[train, ], distribution = "gaussian",
    n.trees = 5000, interaction.depth = 4)
summary(boost.boston, plotit = FALSE)
```

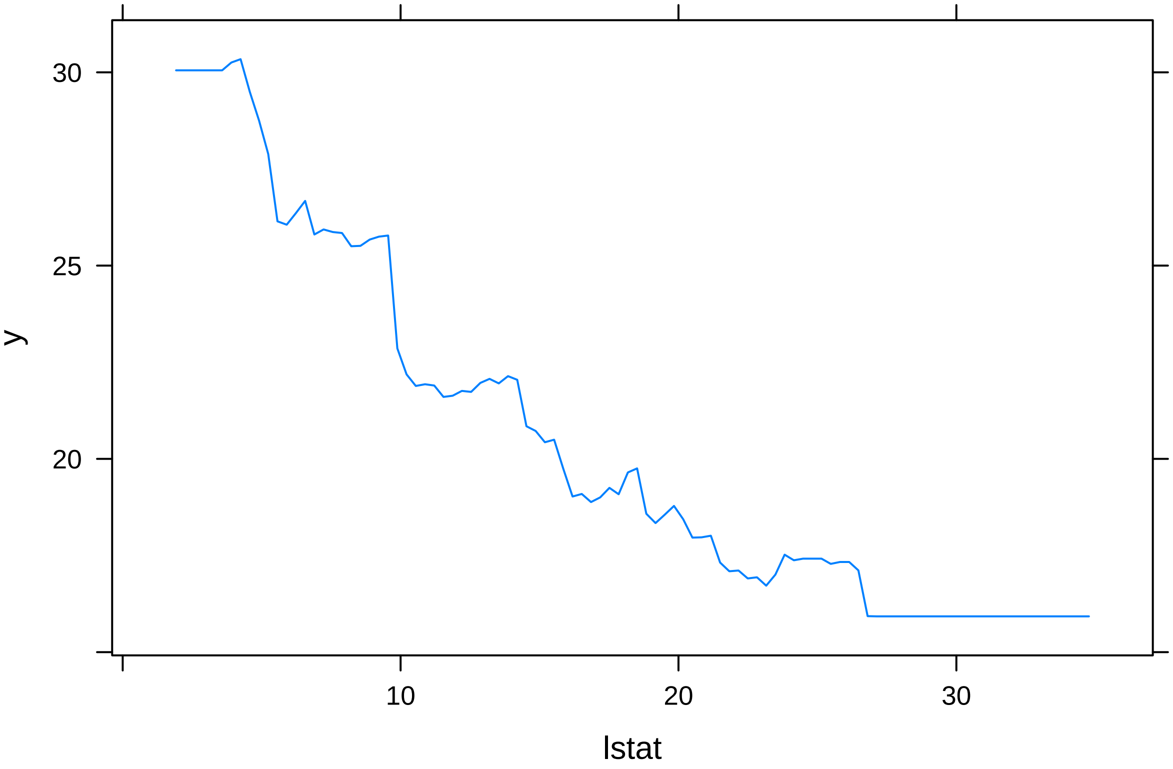
```
##             var   rel.inf
## lstat     lstat 37.066128
## rm           rm 25.353312
## dis         dis 11.790302
## crim       crim  8.038875
## black     black  4.253166
## nox         nox  3.505857
## age         age  3.486872
## ptratio ptratio  2.250039
## indus      indus  1.772507
## tax          tax  1.183659
## chas        chas  0.744132
## rad          rad  0.427431
## zn            zn  0.127721
```

Partial dependency plots - integrating out other variables

```r
par(mfrow = c(1, 2))
plot(boost.boston, i = "rm")
```



```r
plot(boost.boston, i = "lstat")
```



Prediction on test set

First for model with $\lambda = 0.001$ (default), then with $\lambda = 0.2$: MSE on test set. We could have done cross-validation to find the beste λ over a grid.

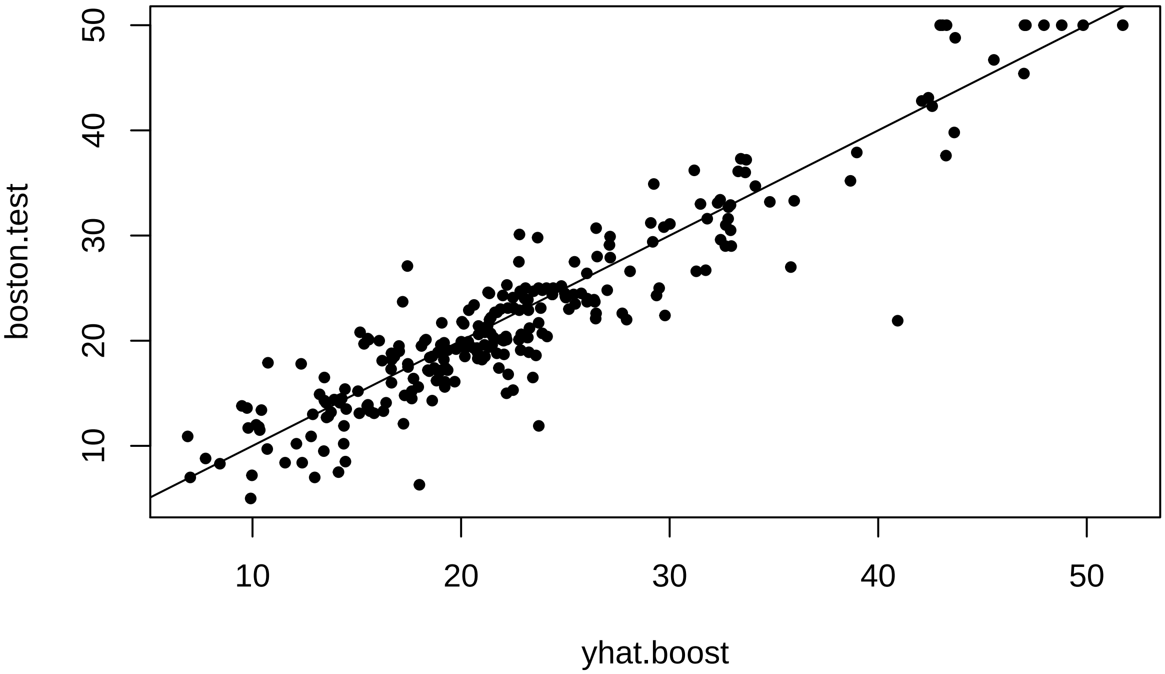
```
yhat.boost = predict(boost.boston, newdata = Boston[-train, ], n.trees = 5000)
mean((yhat.boost - boston.test)^2)
```

```
## [1] 10.8148
```

```
boost.boston = gbm(medv ~ ., data = Boston[train, ], distribution = "gaussian",
    n.trees = 5000, interaction.depth = 4, shrinkage = 0.2, verbose = F)
yhat.boost = predict(boost.boston, newdata = Boston[-train, ], n.trees = 5000)
mean((yhat.boost - boston.test)^2)
```

```
## [1] 11.5111
```

```
plot(yhat.boost, boston.test, pch = 20)
abline(0, 1)
```



```r
mean((yhat.boost - boston.test)^2)
```

```
## [1] 11.5111
```

Summing up

with a Kahoot! quiz in the interactive lecture!

Recommended exercises

1. Theoretical questions:

a) Show that each bootstrap sample will contain on average approximately 2/3 of the observations.

2. Understanding the concepts and algorithms:

a) Do Exercise 1 in our book (page 332)

Draw an example (of your own invention) of a partition of two-dimensional feature space that could result from recursive binary splitting. Your example should contain at least six regions. Draw a decision tree corresponding to this partition. Be sure to label all aspects of your figures, including the regions R1,R2,. . . , the cutpoints t1,t2,. . . , and so forth.

If the class border of the two dimensional space is linear, how can that be done with recursive binary splitting?

b) Do Exercise 4 in the book (page 332).

Suppose that we want to build a regression tree based on the following dataset:

i	(x_{i1}, x_{i2})	y
1	(1,3)	2
2	(2,2)	5
3	(3,2)	3
4	(3,4)	7

Answer the following questions without using *R*:

c) Find the optimal splitting variable and split point for the first binary splitting for these data according to the recursive binary splitting algorithm. *Hint*: Draw a figure and look at possible divisions.

d) Continue the tree construction for the toy data until each terminal node in the tree corresponds to one single observation. Let the resulting tree be denoted T_0.

e) For what values of α in the cost-complexity criterion $C_\alpha(T)$ will the unpruned tree T_0 be the optimal tree? *Hint* : Prune the tree by cost complexity pruning.

f) Suppose that we want to predict the response y for a new observation at $\mathbf{x}$=(2,3). What is the predicted value when using the tree T_0 constructed above?

3. Implementation:

In this exercise you are going to implement a spam filter for e-mails by using tree-based methods. Data from 4601 e-mails are collected and can be uploaded from the kernlab library as follows:

```r
library(kernlab)

data(spam)
```

Each e-mail is classified by *type* (*spam* or *nonspam*), and this will be the response in our model. In addition there are 57 predictors in the dataset. The predictors describe the frequency of different words in the e-mails and orthography (capitalization, spelling, punctuation and so on).

a) Study the dataset by writing *?spam* in R.

b) Create a training set and a test set for the dataset.

c) Fit a tree to the training data with *type* as the response and the rest of the variables as predictors. Study the results by using the *summary()* function. Also create a plot of the tree. How many terminal nodes does it have?

d) Predict the response on the test data. What is the misclassification rate?

e) Use the *cv.tree()* function to find the optimal tree size. Prune the tree according to the optimal tree size by using the *prune.misclass()* function and plot the result. Predict the response on the test data by using the pruned tree. What is the misclassification rate in this case?

f) Create a decision tree by using the bagging approach. Use the function *randomForest()* and consider all of the predictors in each split. Predict the response on the test data and report the misclassification rate.

g) Apply the *randomForest()* function again, but this time consider only a subset of the predictors in each split. This corresponds to the random forest-algorithm. Study the importance of each variable by using the function *importance()*. Are the results as expected based on earlier results? Again, predict the response for the test data and report the misclassification rate.

h) Use *gbm()* to construct a boosted classification tree. Predict the response for the test data and report the misclassification rate.

i) Compare the misclassification rates in d-h. Which method gives the lowest misclassification rate for the test data? Are the results as expected?

Compulsory exercise 3, 2018, Problem 1 - Classification with trees

We will use the *German credit data set* from the UC Irvine machine learning repository. Our aim is to classify a customer as *good* or *bad* with respect to credit risk. A set of 20 covariates (attributes) are available (both numerical and categorical) for 300 customers with bad credit risk and 700 customers with good credit risk.

More information on the 20 covariates are found that the UCI archive data set description

```
library(caret)
# read data, divide into train and test
germancredit = read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/germa
colnames(germancredit) = c("checkaccount", "duration", "credithistory",
    "purpose", "amount", "saving", "presentjob", "installmentrate", "sexstatus",
    "otherdebtor", "resident", "property", "age", "otherinstall", "housing",
    "ncredits", "job", "npeople", "telephone", "foreign", "response")
germancredit$response = as.factor(germancredit$response)  #2=bad
table(germancredit$response)
```

```
##
##   1   2
## 700 300
```

```
str(germancredit)  # to see factors and integers, numerics
```

```
## 'data.frame':    1000 obs. of  21 variables:
##  $ checkaccount   : Factor w/ 4 levels "A11","A12","A13",..: 1 2 4 1 1 4 4 2 4 2 ...
##  $ duration       : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ credithistory  : Factor w/ 5 levels "A30","A31","A32",..: 5 3 5 3 4 3 3 3 3 5 ...
##  $ purpose        : Factor w/ 10 levels "A40","A41","A410",..: 5 5 8 4 1 8 4 2 5 1 ...
##  $ amount         : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ saving         : Factor w/ 5 levels "A61","A62","A63",..: 5 1 1 1 1 5 3 1 4 1 ...
##  $ presentjob     : Factor w/ 5 levels "A71","A72","A73",..: 5 3 4 4 3 3 5 3 4 1 ...
##  $ installmentrate: int  4 2 2 2 3 2 3 2 2 4 ...
##  $ sexstatus      : Factor w/ 4 levels "A91","A92","A93",..: 3 2 3 3 3 3 3 3 1 4 ...
##  $ otherdebtor    : Factor w/ 3 levels "A101","A102",..: 1 1 1 3 1 1 1 1 1 1 ...
##  $ resident       : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ property       : Factor w/ 4 levels "A121","A122",..: 1 1 1 2 4 4 2 3 1 3 ...
##  $ age            : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ otherinstall   : Factor w/ 3 levels "A141","A142",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ housing        : Factor w/ 3 levels "A151","A152",..: 2 2 2 3 3 3 2 1 2 2 ...
##  $ ncredits       : int  2 1 1 1 2 1 1 1 1 2 ...
##  $ job            : Factor w/ 4 levels "A171","A172",..: 3 3 2 3 3 2 3 4 2 4 ...
##  $ npeople        : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ telephone      : Factor w/ 2 levels "A191","A192": 2 1 1 1 1 2 1 2 1 1 ...
##  $ foreign        : Factor w/ 2 levels "A201","A202": 1 1 1 1 1 1 1 1 1 1 ...
##  $ response       : Factor w/ 2 levels "1","2": 1 2 1 1 2 1 1 1 1 2 ...
```

```
set.seed(4268)  #keep this -easier to grade work
in.train <- createDataPartition(germancredit$response, p = 0.75, list = FALSE)
# 75% for training, one split
germancredit.train <- germancredit[in.train, ]
dim(germancredit.train)
```

```
## [1] 750  21
```

```
germancredit.test <- germancredit[-in.train, ]
dim(germancredit.test)
```

```
## [1] 250  21
```

We will now look at classification trees, bagging, and random forests.

Remark: in description of the data set it is hinted that we may use unequal cost of misclassification for the two classes, but we have not covered unequal misclassification costs in this course, and will therefore not address that in this problem set.

a) Full classification tree [1 point]

```
# construct full tree
library(tree)
library(pROC)
fulltree = tree(response ~ ., germancredit.train, split = "deviance")
summary(fulltree)
plot(fulltree)
text(fulltree)
print(fulltree)
fullpred = predict(fulltree, germancredit.test, type = "class")
testres = confusionMatrix(data = fullpred, reference = germancredit.test$response)
print(testres)
1 - sum(diag(testres$table))/(sum(testres$table))
predfulltree = predict(fulltree, germancredit.test, type = "vector")
testfullroc = roc(germancredit.test$response == "2", predfulltree[, 2])
auc(testfullroc)
plot(testfullroc)
```

Run the code and study the output.

- Q1. Explain briefly how `fulltree` is constructed. The explanation should include the words: greedy, binary, deviance, root, leaves.

b) Pruned classification tree [1 point]

```
# prune the full tree
set.seed(4268)
fullcv = cv.tree(fulltree, FUN = prune.misclass, K = 5)
plot(fullcv$size, fullcv$dev, type = "b", xlab = "Terminal nodes", ylab = "misclassifications")
print(fullcv)
prunesize = fullcv$size[which.min(fullcv$dev)]
prunetree = prune.misclass(fulltree, best = prunesize)
plot(prunetree)
text(prunetree, pretty = 1)
predprunetree = predict(prunetree, germancredit.test, type = "class")
prunetest = confusionMatrix(data = predprunetree, reference = germancredit.test$response)
print(prunetest)
1 - sum(diag(prunetest$table))/(sum(prunetest$table))
predprunetree = predict(prunetree, germancredit.test, type = "vector")
testpruneroc = roc(germancredit.test$response == "2", predprunetree[,
    2])
auc(testpruneroc)
plot(testpruneroc)
```

Run the code and study the output.

- Q2. Why do we want to prune the full tree?
- Q3. How is amount of pruning decided in the code?
- Q4. Compare the the full and pruned tree classification method with focus on interpretability and the ROC curves (AUC).

c) Bagged trees [1 point]

```
library(randomForest)
set.seed(4268)
bag = randomForest(response ~ ., data = germancredit, subset = in.train,
    mtry = 20, ntree = 500, importance = TRUE)
bag$confusion
1 - sum(diag(bag$confusion))/sum(bag$confusion[1:2, 1:2])
yhat.bag = predict(bag, newdata = germancredit.test)
misclass.bag = confusionMatrix(yhat.bag, germancredit.test$response)
print(misclass.bag)
1 - sum(diag(misclass.bag$table))/(sum(misclass.bag$table))
predbag = predict(bag, germancredit.test, type = "prob")
testbagroc = roc(germancredit.test$response == "2", predbag[, 2])
auc(testbagroc)
plot(testbagroc)
varImpPlot(bag, pch = 20)
```

Run the code and study the output.

- Q5. What is the main motivation behind bagging?
- Q6. Explain what the importance plots show, and give your interpretation for the data set.
- Q7. Compare the performance of bagging with the best of the full and pruned tree model above with focus on interpretability and the ROC curves (AUC).

d) Random forest [1 point]

```
set.seed(4268)
rf = randomForest(response ~ ., data = germancredit, subset = in.train,
    mtry = 4, ntree = 500, importance = TRUE)
rf$confusion
1 - sum(diag(rf$confusion))/sum(rf$confusion[1:2, 1:2])
yhat.rf = predict(rf, newdata = germancredit.test)
misclass.rf = confusionMatrix(yhat.rf, germancredit.test$response)
print(misclass.rf)
1 - sum(diag(misclass.rf$table))/(sum(misclass.rf$table))
predrf = predict(rf, germancredit.test, type = "prob")
testrfroc = roc(germancredit.test$response == "2", predrf[, 2])
auc(testrfroc)
plot(testrfroc)
varImpPlot(rf, pch = 20)
```

Run the code and study the output.

- Q8. The parameter `mtry=4` is used. What does this parameter mean, and what is the motivation behind choosing exactly this value?
- Q9. The value of the parameter `mtry` is the only difference between bagging and random forest. What is the effect of choosing `mtry` to be a value less than the number of covariates?
- Q10. Would you prefer to use bagging or random forest to classify the credit risk data?

Exam problems

V2018 Problem 4 Classification of diabetes cases

c)

Q20: Explain how we build a bagged set of trees, and why we would want to fit more than one tree.
Q21: Assume we have a data set of size n, calculate the probability that a given observation is in a given bootstrap sample.
Q22: What is an OOB sample?

R packages

These packages needs to be install before knitting this R Markdown file.

```r
install.packages("gamlss.data")
install.packages("tidyverse")
install.packages("GGally")
install.packages("Matrix")
install.packages("tree")
install.packages("randomForest")
install.packages("gbm")
install.packages("caret")
```

References and further reading

- Videoes on YouTube by the authors of ISL, Chapter 8, and corresponding slides
- Solutions to exercises in the book, chapter 8

Acknowledgements

The main part of this module page was developed by Thea Roksvåg. In addition Julia Debik has contributed with example and input.

Breiman, Leo. 1996. “Bagging Predictors.” *Machine Learning* 24: 123–40.

———. 2001. “Random Forest.” *Machine Learning* 45: 5–32.

Efron, Bradley, and Trevor Hastie. 2016. *Computer Age Statistical Inference - Algorithms, Evidence, and*

Data Science. Cambridge University Press.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*. Vol. 1. Springer series in statistics New York.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning*. Vol. 112. Springer.

Ripley, Brian D. 1996. *Pattern Recognicion and Neural Networks*. Cambridge University Press.

Venables, W. N., and B. D. Ripley. 2002. *Modern Applied Statistics with S*. Springer.