

— CLASS NOTE VERSION —

TMA4268 Statistical Learning V2018

Module 8: TREE-BASED METHODS

Mette Langaas, Thea Roksvåg, Julia Debik, Department of
Mathematical Sciences, NTNU

2nd lecture M8: 07.03.2018

week 10, 2018 (version 06.03.2018)

Introduction

What will you learn?

- ▶ Decision tree - idea and example
- ▶ Regression trees
 - ▶ what is a tree
 - ▶ how to grow a tree
- ▶ Classification trees - any changes to the above
- ▶ Pruning a tree
- ▶ Bagging
 - ▶ Variable importance plots
- ▶ Random forests
- ▶ Boosting

L1

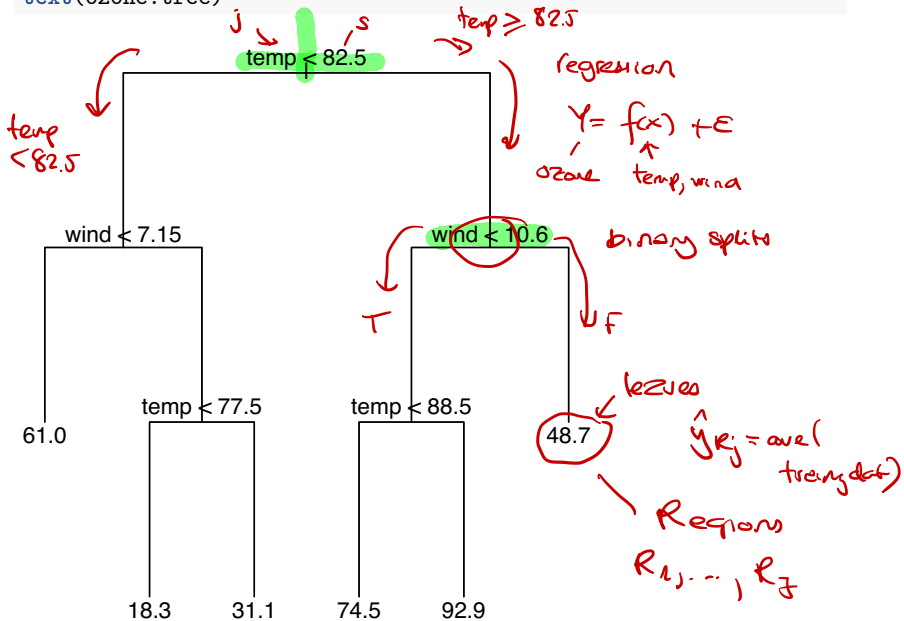


L2

```

ozone.tree = tree(ozone~temp+wind, data=myozone)
plot(ozone.tree,type="uniform")
text(ozone.tree)

```



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

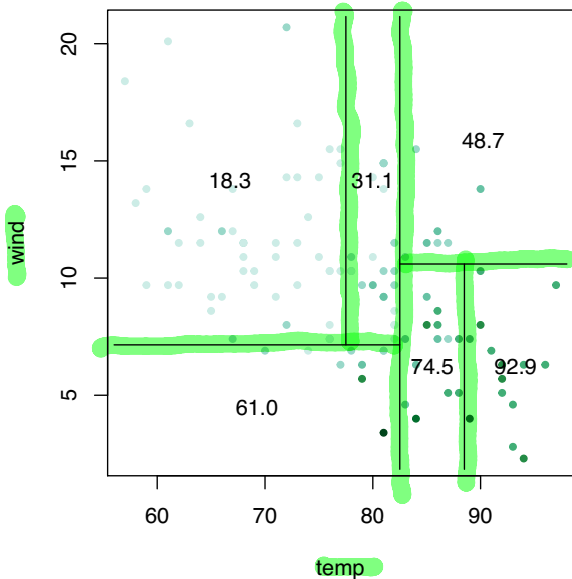
Recursive binary splitting

s, j

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, \dots, 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.



- Ozone
- (0.833,24.9]
 - (24.9,48.7]
 - (48.7,72.6]
 - (72.6,96.4]
 - (96.4,120]
 - (120,144]
 - (144,168]

Partition into

6 regions

R_1, \dots, R_6

↑

18.3

↑
GR

2) The splitting criterion: We can not use RSS as a splitting criterion for a qualitative variable. Instead we can use some *measure of impurity* of the node.

Gini index:

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

Cross entropy:

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

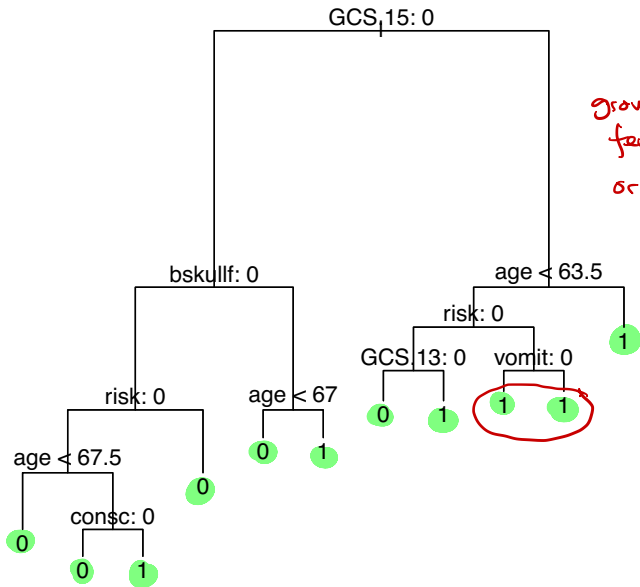
Here \hat{p}_{jk} is the proportion of training observation in region j that are from class k .

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

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 observations in region R_j with N_j observations:

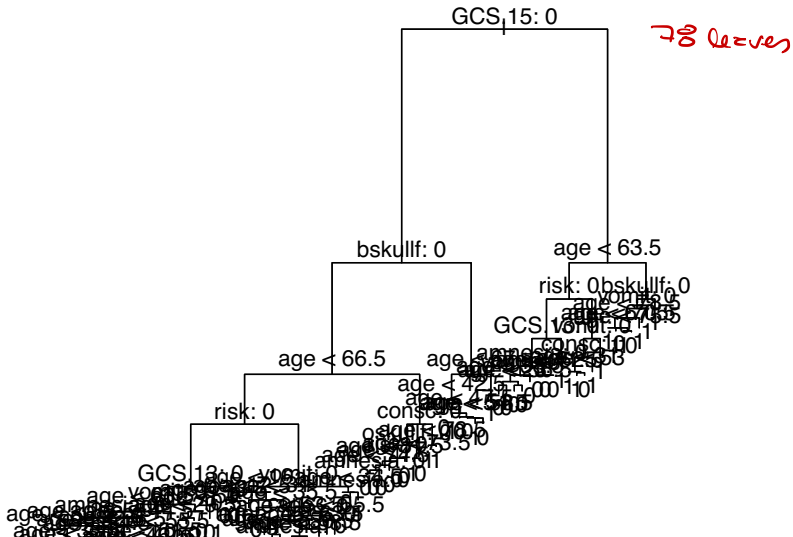
$$\hat{p}_{jk} = \frac{1}{N_j} \sum_{i: x_i \in R_j} I(y_i = k).$$



grow tree until
few obs in leaves
or not enough
described
in
loss

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


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



Tree performance

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

```
set.seed(200)
ozone.trainID = sample(1:111, 75)
ozone.train = ozone[ozone.trainID, ]
ozone.test = ozone[-ozone.trainID,]

ozone.full = tree(ozone~temperature+wind,
                  data=ozone.train)

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

```
## [1] 297.452
```

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, \dots, 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.

- increase interpretability
- avoid unnecessary splits

Cost complexity pruning

We can prune the tree by using an 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 for 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 MA8702 - PhD version of Statistical Learning course for an example of how we perform cost complexity pruning in detail.

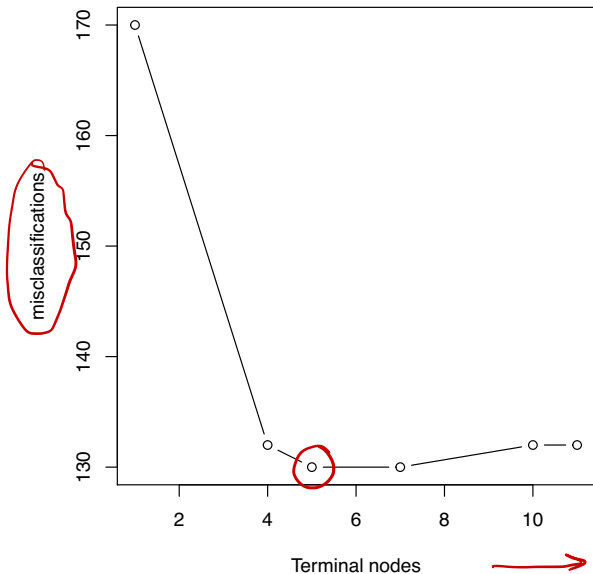
Series of pruned trees.

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 α .

Combining pruning and cross-validation to find optimal tree

We continue using the classification tree.



Minor head injury

5 term. nodes

||

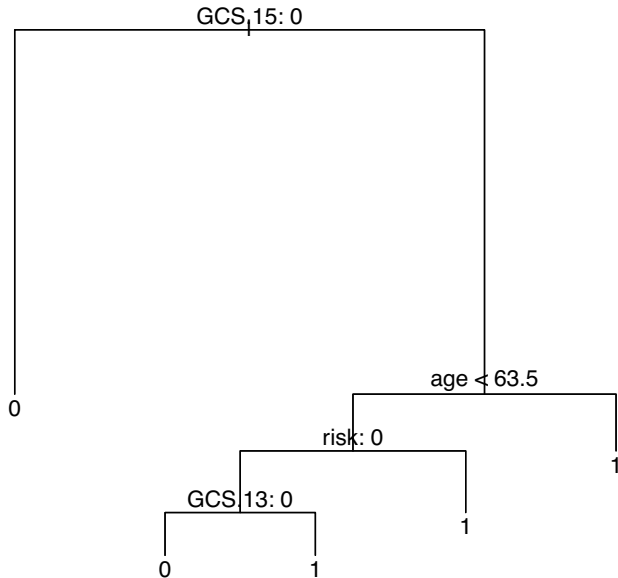
The function `cv.tree` automatically does 10-fold cross-validation. `dev` is here the number of misclassifications.

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

```
## $size
## [1] 11 10 7 5 4 1
##
## $dev miscl.
## [1] 132 132 130 130 132 170
##
## $k  $\leftarrow \alpha$ 
## [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.
```



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?

```
tree.pred.prune=predict(prune.HIClass,headInjury2[test,],ty  
misclass.prune=table(tree.pred,headInjury2[test,]$clinical  
print(misclass.prune)
```

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

```
##
```

```
## tree.pred    0    1
```

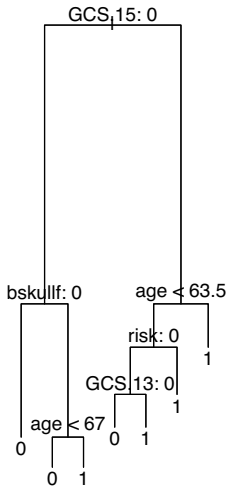
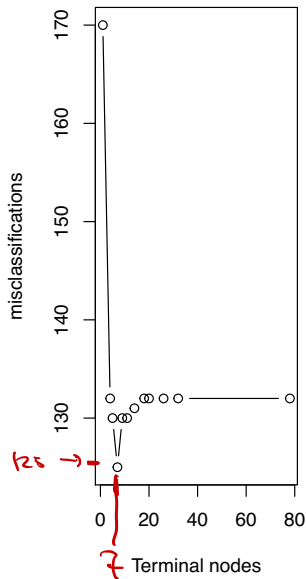
```
##           0 361  50
```

```
##           1  18  42
```

```
## [1] 0.144374
```

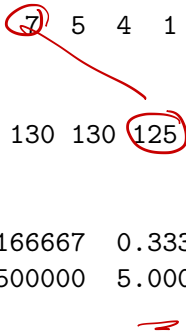
We see that the misclassification rate is as small as before indicating that the pruned tree is as good as the original tree for the test data.

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



```
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
## [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?

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 (continuous, 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

But first,

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

Quotation from Wikipedia

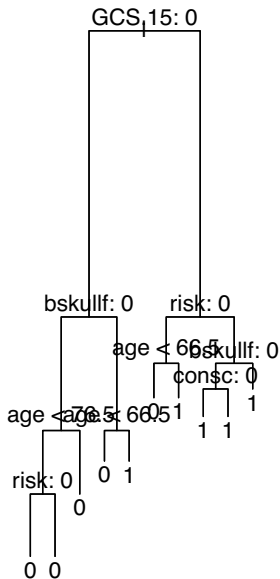
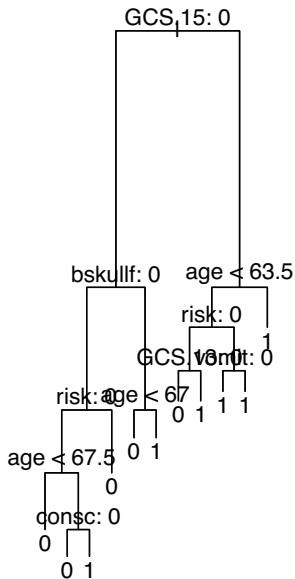
Leo Breiman (January 27, 1928 – July 5, 2005) was a distinguished statistician at the University of California, Berkeley. He was the recipient of numerous honors and awards, and was a member of the United States National Academy of Science.

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.

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).



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

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

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

But, we will not draw random variables - we want to fit decision trees: $\hat{f}_1(\mathbf{x}), \hat{f}_2(\mathbf{x}), \dots, \hat{f}_B(\mathbf{x})$ 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 $b = 1, \dots, B$, 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

$$1 - \left(1 - \frac{1}{n}\right)^n \approx 1 - e^{-1} = 0.632121 \approx \frac{2}{3}$$

- ▶ 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, $f^{*b}(x)$.

An 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.

```
library(randomForest)
set.seed(1)
bag=randomForest(clinically.important.brain.injury~.,
                 data=headInjury2,subset=train,
                 mtry=10,ntree=500,importance=TRUE)
bag$confusion
1-sum(diag(bag$confusion))/sum(bag$confusion[1:2,1:2])
```

```
##      0  1 class.error
## 0 642 50  0.0722543
## 1  82 76  0.5189873
## [1] 0.155294 miscl.error OOB obs.
```

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:

```
yhat.bag=predict(bag,newdata=headInjury2[test,])
misclass.bag=table(yhat.bag,headInjury2[test,]$clinically.)
print(misclass.bag)
1-sum(diag(misclass.bag))/(sum(misclass.bag))
```

```
##
## yhat.bag    0    1
##           0 351  43
##           1  28  49
## [1] 0.150743
```

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.**

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 neighbours - 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.

Method A

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 splits 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`.

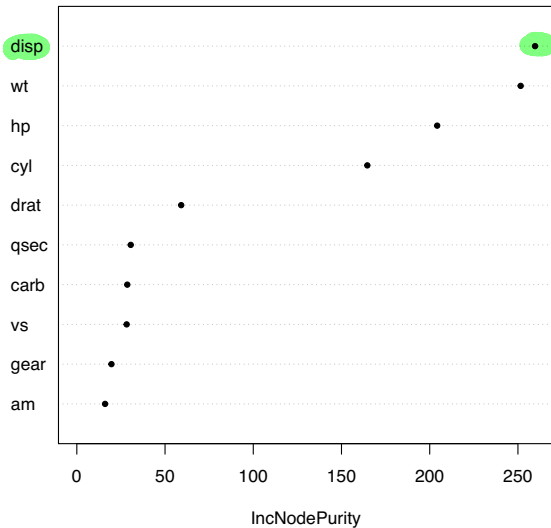
node
importance

Auto data example

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

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

mtcars.rf



Variable importance based on randomization

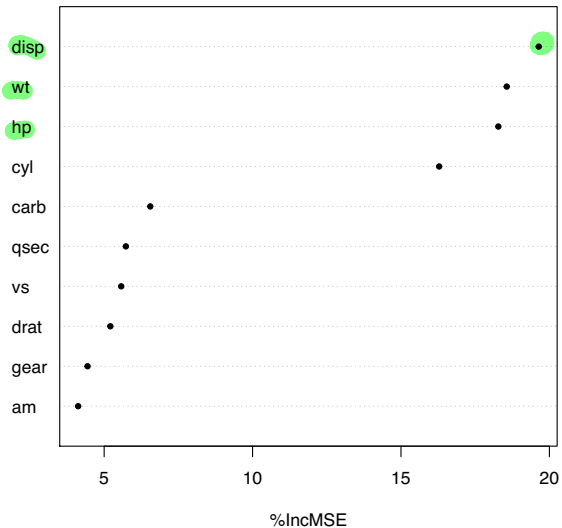
Variable importance based on randomization is calculated using the OOB sample.

- ▶ 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 j th 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`.

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

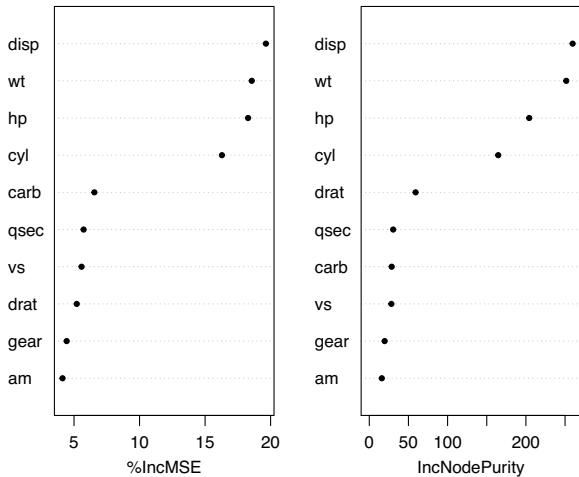
mtcars.rf



The two types together: do they agree?

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

mtcars.rf



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

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

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 $m \approx \sqrt{p}$ (classification) and $m = p/3$ (regression)

The general idea is that 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 equivalent to LOOCV (Efron and Hastie, 2016, p 330).

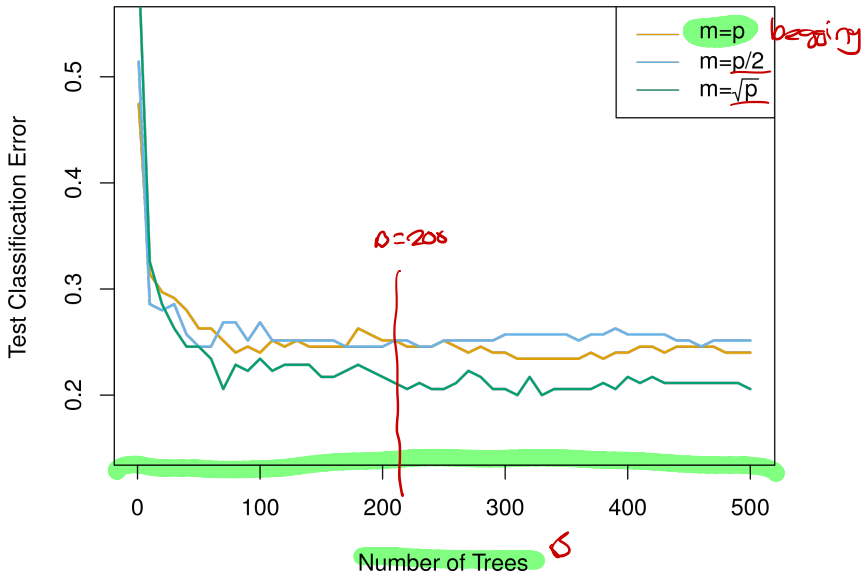


Figure 2: ISLR Figure 8.10, gene expression data set with 15 classes and 500 predictors

Example: Boston data set

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

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

House prices in suburbs in Boston

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

response
median
value

$$l3 = p$$

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

Before we start

Introduction

Constructing a decision tree

Pruning

From trees to forests

Bagging

Random Forest

Boosting

Example: Boston data set

Finally, boosting Boston

Summing up

Recommended exercises

R packages

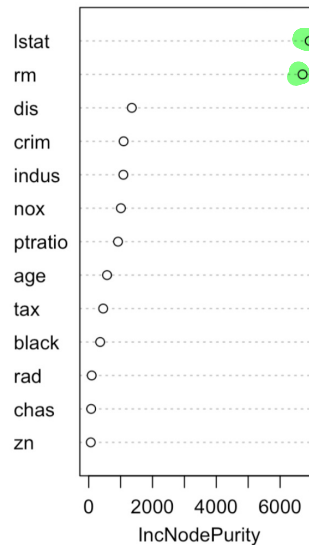
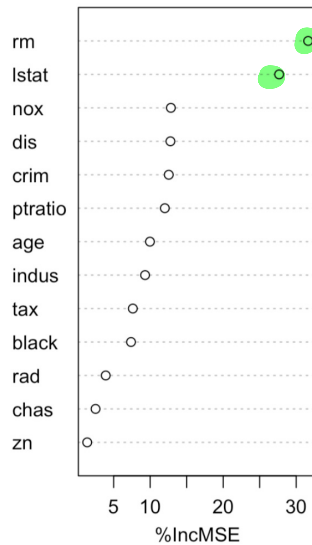
References and further reading

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

rf.boston

m=6 of the predict

Random forest



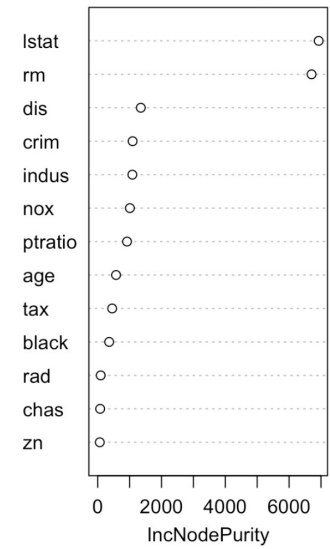
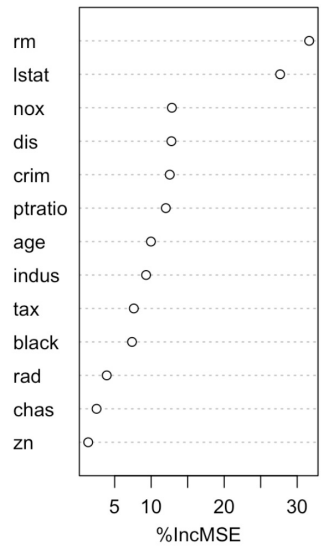
Missing error on test set

Will update web-version of rfs

```
## [1] 11.4802
##           %IncMSE  IncNodePurity
## crim      12.54777    1094.6538
## zn         1.37549      64.4006
## indu       9.30426    1086.0910
## chas       2.51877      76.3680
## nox       12.83561    1008.7370
```



- Before we start
- Introduction
- Constructing a decision tree
- Pruning
- From trees to forests
- Bagging
- Random Forest
- Boosting
 - Example: Boston data set
 - Finally, boosting Boston
- Summing up
- Recommended exercises
- R packages
- References and further reading



```
## [1] 11.4802
##          %IncMSE  IncNodePurity
## crim    12.54777    1094.6538
## zn       1.37549      64.4006
## indus    9.30426    1086.0910
## chas     2.51877      76.3680
## nox     12.83561    1008.7370
## rm      31.64615    6705.0264
## age      9.97024     575.1370
## dis     12.77443    1351.0198
## rad      3.91185      93.7820
## tax      7.62404     453.1947
## ptratio 12.00819     919.0676
## black    7.37602     358.9693
## lstat   27.66690    6927.9848
```

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).

↙ joint effects

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 $b = 1, 2, \dots, B$, repeat:
 - 2.1 Fit a tree \hat{f}^b with d splits ($d + 1$ terminal nodes) to the training data.
 - 2.2 Update \hat{f} by adding in a shrunken version of the new tree:

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

- 2.3 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.

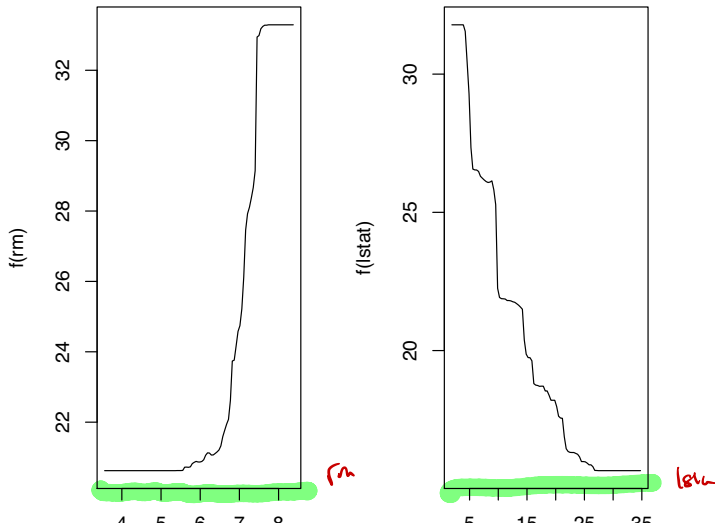
Finally, boosting Boston

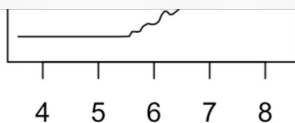
```
library(gbm)
set.seed(1)
boost.boston=gbm(medv~., data=Boston[train, ],
                 distribution="gaussian",
                 n.trees=5000, interaction.depth=4)
summary(boost.boston, plotit=FALSE)
```

```
##           var      rel.inf
## lstat    lstat  45.9627334
## rm       rm    31.2238187
## dis      dis   6.8087398
## crim     crim  4.0743784
## nox      nox   2.5605001
## ptratio  ptratio 2.2748652
## black    black  1.7971159
## age      age   1.6488532
## tax      tax   1.3595005
## indus    indus  1.2705924
## chas     chas  0.8014323
## rad      rad   0.2026619
## zn       zn    0.0148083
```

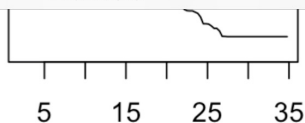

Partial dependency plots - integrating out other variables

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





rm



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 best λ over a grid.

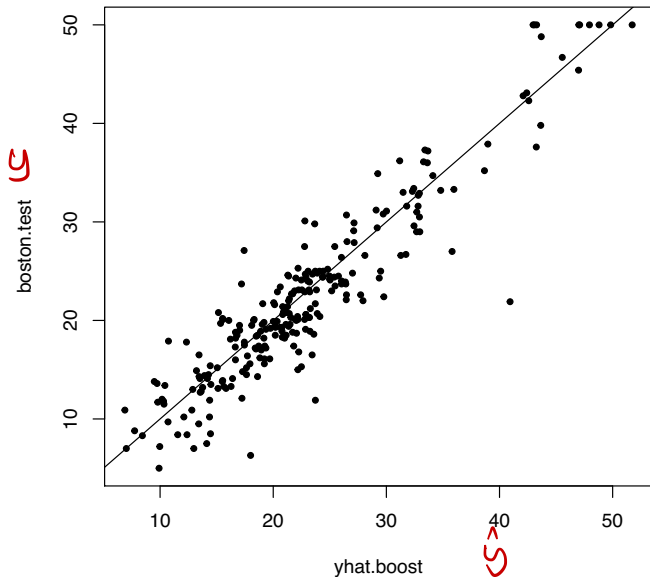
```
yhat.boost=predict(boost.boston,newdata=Boston[-train,],n.trees=5000)
mean((yhat.boost-boston.test)^2)
boost.boston=gbm(medv~.,data=Boston[train,],distribution="gaussian",
  n.trees=5000,interaction.depth=4,shrinkage=0.2,verbose=F)
yhat.boost=predict(boost.boston,newdata=Boston[-train,],n.trees=5000)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

```
## [1] 11.8443
```

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

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

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



~~What is next?~~ What have we done in ML8?

- ▶ **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.

+ Tree construction + pruning

Summing up

with a quiz on Module 8: Tree-based methods - also hosted in Kahoot!

Recommended exercises

1. Theoretical questions:

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

2. Understanding the concepts and algorithms:

1. Do Exercise 1 in our book (page 331?)

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 R_1, R_2, \dots , the cutpoints t_1, t_2, \dots , and so forth.

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

R packages

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

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

References and further reading

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

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.

Rinley, Brian D. 1996. *Pattern Recognition and Neural Networks*