**15 – Tree-based Models for Classification Problems**

**15.1 – Classification Trees**

In classification problems the goal is to develop a tree-based model that will classify observations into one of *C* predetermined classes. The end result of a tree model can be viewed as a series of conditional probabilities (*posterior probabilities*) of class membership given a set of covariate values. For each terminal node we essentially have a probability distribution for class membership, where the probabilities are of the form:

such that .

Here, is a neighborhood or node defined by the set of predictors .

The neighborhoods/nodes are found by a series of binary splits chosen to minimize the overall “loss” of the resulting tree. For classification problems measuring overall “loss” can be a bit complicated. One obvious method is to construct classification trees so that the overall misclassification rate is minimized. In fact, this is precisely what the RPART algorithm does by default. In classification problems it is often the case that we wish to incorporate prior knowledge about likely class membership. This knowledge is represented by prior probabilities of an observation being from class , which will denote by . Naturally the priors must be chosen in such a way that they sum to 1, i.e. . Other information we might want to incorporate into a modeling process is the cost or loss incurred by classifying an object from class as being from class and vice versa. With this information provided we would expect the resulting tree to avoid making the more costly misclassifications on our training data set.

Some notation that is used by Therneau & Atkinson (1999) for determining the loss for a given node :

some node of the tree

number of observations in node A from class

number of observations in training set from class

total number of observations in the training set

number of observations in node A

prior probability of being from class (by default )

probability of future observations being in node A.

true probability of being from class given we are in node A.

loss incurred when classifying a class object as being from class

 predicted class for node A

In general, the loss is specified as a matrix

By default, this is a symmetric matrix with .

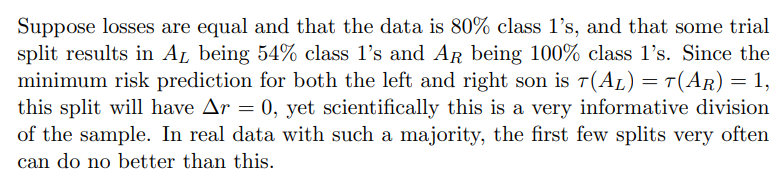
Using the notation and concepts presented above the risk or loss at an arbitrary node is given by

and the risk for the entire tree () is given by

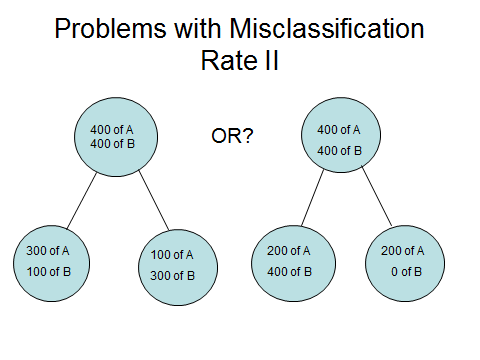
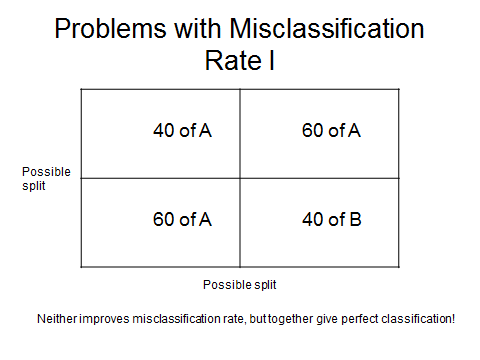
When the priors are set equal to the proportion of observations in each the classes of the response given by then and proportion (or number) of observations misclassified, i.e. the misclassification rate.

It would seem that using misclassification rate (or simply the number misclassified) of the training observations would be a reasonable criterion for constructing a tree, i.e. at each split in the tree we would choose the split which results in the largest decrease in the misclassifications. This however is not what gets used in practice. The following examples illustrate why.

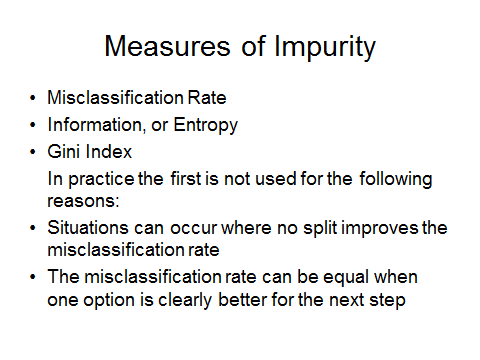
Example 1: (taken from Therneau & Atkinson)



Examples 2 & 3:

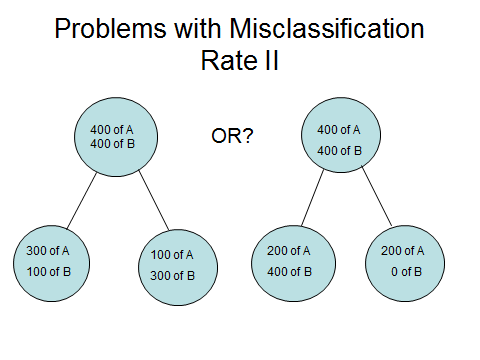


To deal with these potential problems we use alternative measures for building a classification tree model. The general idea is to use a function that measures the impurity of node .



One measure that is used is the Information criterion.

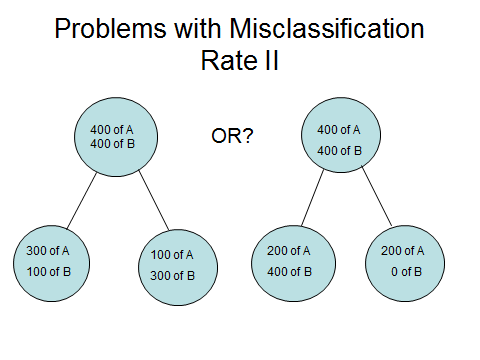
where .

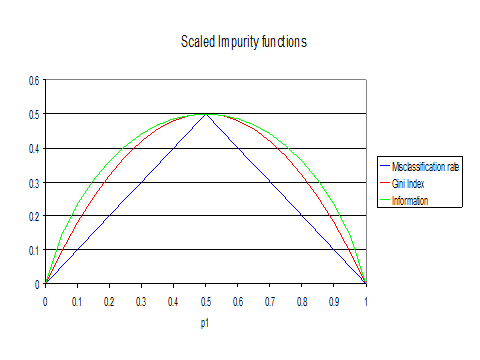


Another measure that can be used to measure impurity is the Gini Index.

For binary classification (i.e. ) the Gini Index for node is given by,

and for multi-class problems (i.e. ) the Gini Index is the given by





**Other metrics**

With other modeling approaches, e.g. boosted classification trees (xgboost), we will see we can choose from a variety of metrics such AUC or log loss. These two metrics are commonly used in Kaggle ® classification problems. AUC is particularly good for highly unbalanced classification problems.

We will now consider some examples using the following packages rpart (comes with R just load it) and partykit (needs to be installed and loaded).

**Example 15.1: Kyphosis Data**

> library(rpart)

> library(partykit)

> data(kyphosis)

> attach(kyphosis)

> names(kyphosis)

[1] "Kyphosis" "Age" "Number" "Start"

> k.default <- rpart(Kyphosis~.,data=kyphosis)

> k.default

n= 81

node), split, n, loss, yval, (yprob)

\* denotes terminal node

**1) root 81 *17* absent (0.7901235 0.2098765)**

2) Start>=8.5 62 6 absent (0.9032258 0.0967742)

4) Start>=14.5 29 0 absent (1.0000000 0.0000000) \*

5) Start< 14.5 33 6 absent (0.8181818 0.1818182)

10) Age< 55 12 0 absent (1.0000000 0.0000000) \*

11) Age>=55 21 6 absent (0.7142857 0.2857143)

22) Age>=111 14 2 absent (0.8571429 0.1428571) \*

23) Age< 111 7 3 present (0.4285714 0.5714286) \*

**3) Start< 8.5 19 *8* present (0.4210526 0.5789474) \***

The basic display of the tree obtained by simply typing its names shows the splits used to define tree with nested splits indented below the parent nodes. Also reported is the number of observations in that node, the “number misclassified” (loss), the predicted response value in that node, and proportion of observations in each class. Terminal nodes are denoted by (\*).

The actual tree is shown on the following page, using functions in the partykit library to create the plot.

> ktree.party = as.party(k.default)

> ktree.party

Model formula:

Kyphosis ~ Age + Number + Start

Fitted party:

[1] root

| [2] Start >= 8.5

| | [3] Start >= 14.5: absent (n = 29, err = 0.0%)

| | [4] Start < 14.5

| | | [5] Age < 55: absent (n = 12, err = 0.0%)

| | | [6] Age >= 55

| | | | [7] Age >= 111: absent (n = 14, err = 14.3%)

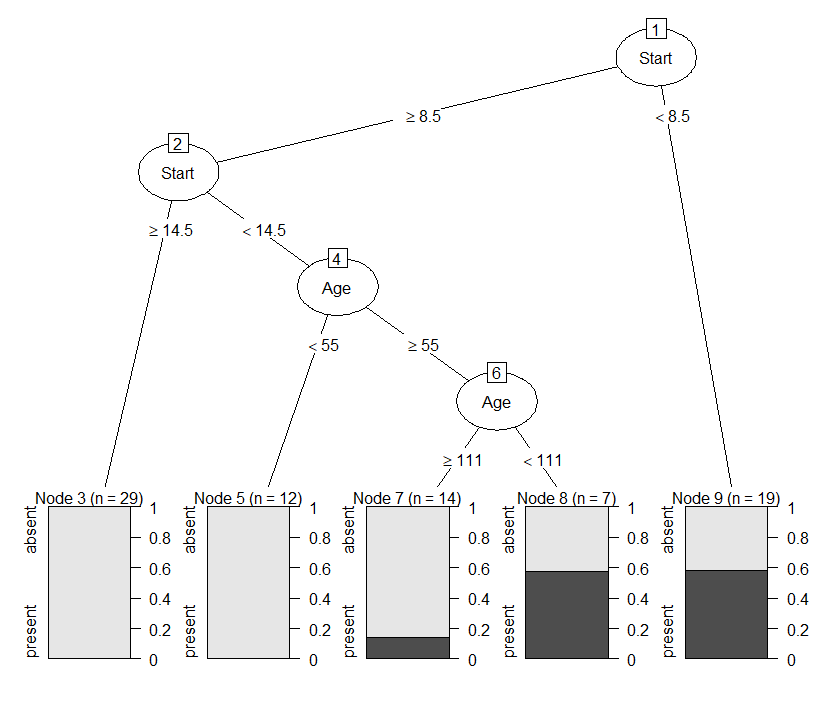
| | | | [8] Age < 111: present (n = 7, err = 42.9%)

| [9] Start < 8.5: present (n = 19, err = 42.1%)

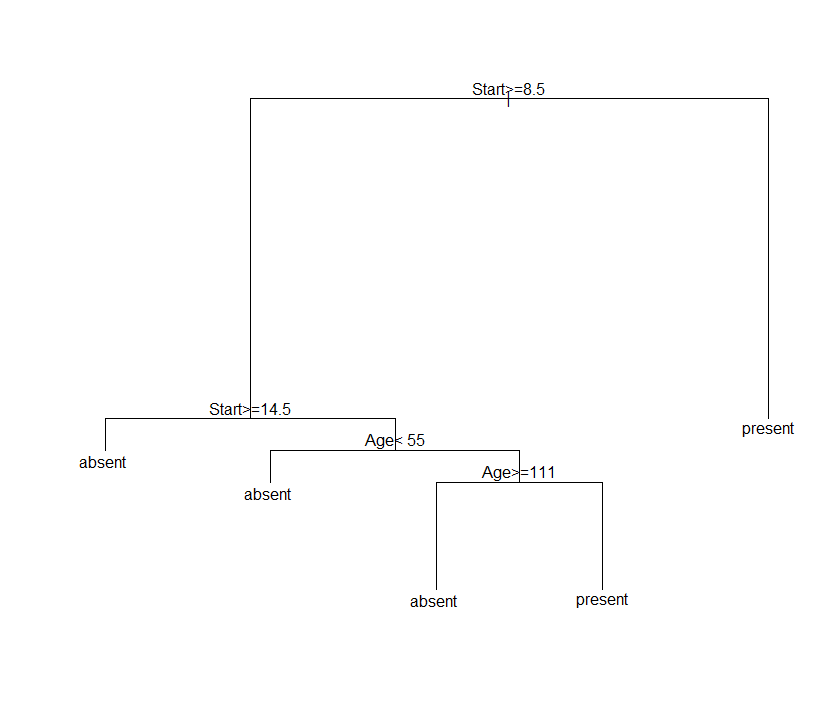
Number of inner nodes: 4

Number of terminal nodes: 5

> plot(ktree.party)



Using the partykit to plot trees produces a much more informative tree than the basic plotting functions in rpart. There are other libraries that have advanced plotting functions for trees, but partykit is the best in my opinion. Below are the base commands for inspecting a tree.



> plot(k.default)

> text(k.default)

To the right is the very basic plot

of the tree using the default

plotting functions, which is   
generally sufficient during the tree

development process.

Suppose now we have prior beliefs that 65% of patients will not have Kyphosis (absent) and 35% of patients will have Kyphosis (present).

> k.priors <-rpart(Kyphosis~.,data=kyphosis,parms=list(prior=c(.65,.35)))

> k.priors

n= 81

node), split, n, loss, yval, (yprob)

\* denotes terminal node

**1) root 81 *28.350000* absent (0.65000000 0.35000000)**

2) Start>=12.5 46 3.335294 absent (0.91563089 0.08436911) \*

3) Start< 12.5 35 16.453130 present (0.39676840 0.60323160)

6) Age< 34.5 10 1.667647 absent (0.81616742 0.18383258) \*

**7) Age>=34.5 25 *9.049219* present (0.27932897 0.72067103) \***

The resulting tree is shown below.

> kprior.party = as.party(k.priors)

> kprior.party

Model formula:

Kyphosis ~ Age + Number + Start

Fitted party:

[1] root

| [2] Start >= 12.5: absent (n = 46, err = 4.3%)

| [3] Start < 12.5

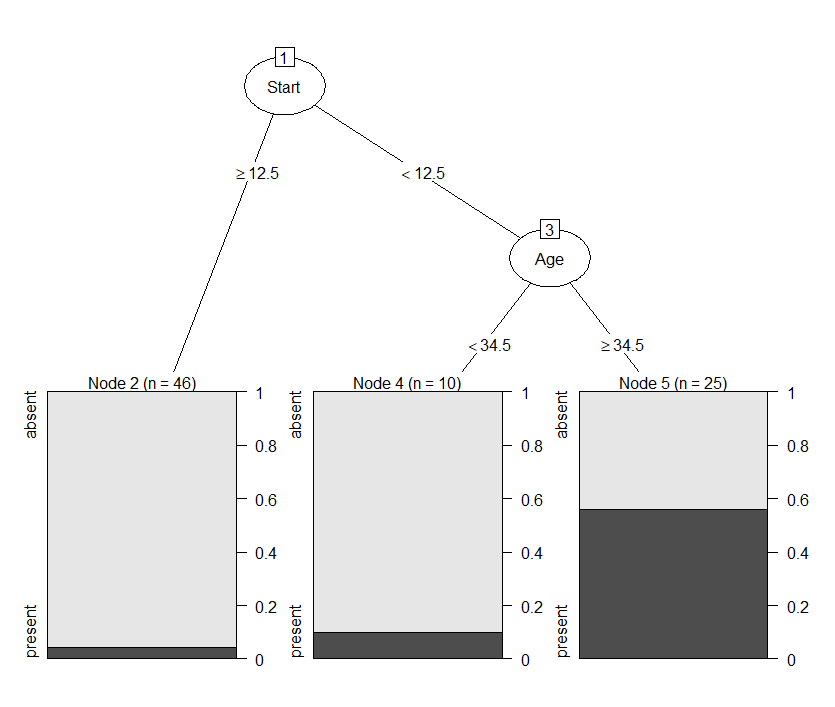
| | [4] Age < 34.5: absent (n = 10, err = 10.0%)

| | [5] Age >= 34.5: present (n = 25, err = 44.0%)

Number of inner nodes: 2

Number of terminal nodes: 3

> plot(kprior.party)



Finally suppose we wish to incorporate the following loss information.

|  |  |  |  |
| --- | --- | --- | --- |
| **Actual** | **Classification** | | |
| *L(i,j)* | present | absent |
| present | 0 | 4 |
| absent | 1 | 0 |

Note: In R the category orderings for the loss matrix are Z 🡪 A in both dimensions.

This says that it is *4* times more serious to misclassify a child that actually has kyphosis (present) as not having it (absent). Again we will use the priors from the previous model (65% - absent, 35% - present).

> lmat <- matrix(c(0,4,1,0),nrow=2,ncol=2,byrow=T)

> k.priorloss <- rpart(Kyphosis~.,data=kyphosis,parms=list(prior=c(.65,.35),loss=lmat))

> k.priorloss

n= 81

node), split, n, loss, yval, (yprob)

\* denotes terminal node

**1) root 81 *52.650000* present (0.6500000 0.3500000)**

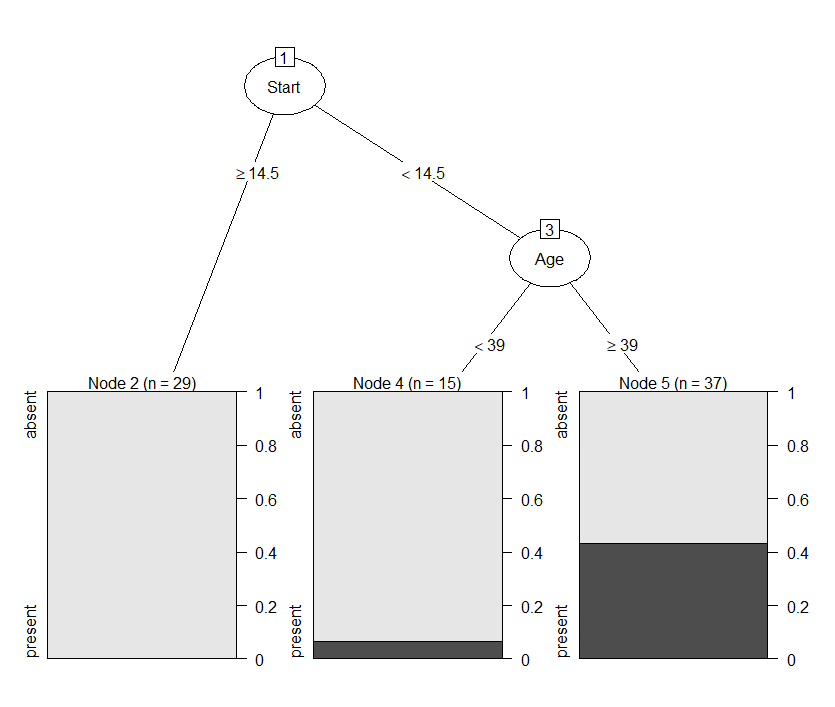
2) Start>=14.5 29 0.000000 absent (1.0000000 0.0000000) \*

3) Start< 14.5 52 28.792970 present (0.5038760 0.4961240)

6) Age< 39 15 6.670588 absent (0.8735178 0.1264822) \*

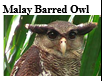
**7) Age>=39 37 *17.275780* present (0.3930053 0.6069947) \***

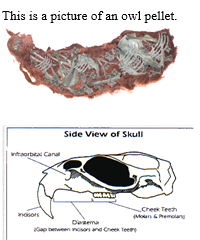
The resulting tree is shown below.



**Example 15.2 – Owl Diet**

Owls, like many other birds, eat their food whole. Since birds do not have teeth, they can't chew their food. Therefore, they use their strong and sharp beaks to rip their prey apart and then swallow large chunks whole. The owl slowly digests its meal by separating the softer materials (such as meat) from the harder material (such as bones). It then regurgitates the harder material along with indigestible items such as feathers and fur in the form of a pellet. Owl pellets can be dissected and used to find out what and how much an owl has been eating. It can also be used to determine in what areas they live etc. These data contain measurements made on skulls of rodents found  
in the owl pellets regurgitated by the Malay Barred Owl. The goal is to classify   
rodent species eaten by the Malay Barred Owl based on skull measurements





from skulls found in regurgitated pellets.

> owl.tree <- rpart(species~.,data=OwlDiet)

> owl.tree

n= 179

node), split, n, loss, yval, (yprob)

\* denotes terminal node

1) root 179 140 tiomanicus (0.14 0.21 0.13 0.089 0.056 0.22 0.16)

2) teeth\_row>=56.09075 127 88 tiomanicus (0.2 0.29 0 0.13 0.079 0.31 0)

4) teeth\_row>=72.85422 26 2 annandalfi (0.92 0.077 0 0 0 0 0) \*

5) teeth\_row< 72.85422 101 62 tiomanicus (0.0099 0.35 0 0.16 0.099 0.39 0)

10) teeth\_row>=64.09744 49 18 argentiventer (0.02 0.63 0 0.27 0.02 0.061 0)

20) palatine\_foramen>=61.23703 31 4 argentiventer (0.032 0.87 0 0.065 0 0.032 0)\*

21) palatine\_foramen< 61.23703 18 7 rajah (0 0.22 0 0.61 0.056 0.11 0) \*

11) teeth\_row< 64.09744 52 16 tiomanicus (0 0.077 0 0.058 0.17 0.69 0)

22) skull\_length>=424.5134 7 1 surifer (0 0.14 0 0 0.86 0 0) \*

23) skull\_length< 424.5134 45 9 tiomanicus (0 0.067 0 0.067 0.067 0.8 0) \*

3) teeth\_row< 56.09075 52 24 whiteheadi (0 0 0.46 0 0 0 0.54)

6) palatine\_foramen>=47.15662 25 1 exulans (0 0 0.96 0 0 0 0.04) \*

7) palatine\_foramen< 47.15662 27 0 whiteheadi (0 0 0 0 0 0 1) \*

The function misclass.rpart constructs a confusion matrix for a classification tree.

> misclass.rpart(owl.tree)

Table of Misclassification

(row = predicted, col = actual)

1 2 3 4 5 6 7

annandalfi 24 2 0 0 0 0 0

argentiventer 1 27 0 2 0 1 0

exulans 0 0 24 0 0 0 1

rajah 0 4 0 11 1 2 0

surifer 0 1 0 0 6 0 0

tiomanicus 0 3 0 3 3 36 0

whiteheadi 0 0 0 0 0 0 27

Misclassification Rate = 0.134

Using equal prior probabilities for each of the groups.

> owl.tree2 <- rpart(species~.,data=OwlDiet,parms=list(prior=c(1,1,1,1,1,1,1)/7))

> misclass.rpart(owl.tree2)

Table of Misclassification

(row = predicted, col = actual)

1 2 3 4 5 6 7

annandalfi 24 2 0 0 0 0 0

argentiventer 1 27 0 2 0 1 0

exulans 0 0 24 0 0 0 1

rajah 0 4 0 11 1 2 0

surifer 0 1 0 1 9 5 0

tiomanicus 0 3 0 2 0 31 0

whiteheadi 0 0 0 0 0 0 27

Misclassification Rate = 0.145

> plot(owl.tree)

> text(owl.tree,cex=.6)



We can see that teeth row and palantine foramen figure prominently in the classification rule. If you examine the group differences across these characteristics using comparative boxplots this fact is not surprising. I should point out that I am not using partykit to plot these trees because the displays are too busy when the response has many categories.

As with regression trees using rpart we can specify parameters such as minbucket, minsplit, and cp to control the size/complexity of the tree. We can use plotcp and printcp to help determine the “optimal” value for the cost complexity parameter (cp).

Cross-validation can also be done in the usual fashion: we leave out a certain percentage of the observations, develop a model from the remaining data, predict back the class of   
  
the observations we set aside, calculate the misclassification rate, and then repeat this process are number of times. The function crpart.sscv leaves out *%* of the data at a time to perform the Monte Carlo cross-validation (MCCV) to estimate the APER.

> results = crpart.sscv(owl.rpart,y=Owldiet$species,data=Owldiet,B=100)

> summary(results)

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.1017 0.1695 0.2034 0.2193 0.2712 0.3559

We can control complexity of the tree by collecting the arguments into a control setting using the function rpart.control. In the tree below we let the tree grow very large by using different tuning parameters.

> owl.overfit = rpart(species~.,data=Owldiet,control=rpart.control(minbucket=1,minsplit=2,

cp=.000001,mindev=0))

The mindev=0 argument specifies the tree should continue growing until there are no misclassification errors, if possible.

> overfit.results = crpart.sscv(owl.overfit,Owldiet$species,data=Owldiet,B=100)

> summary(overfit.results)

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.1356 0.2203 0.2542 0.2600 0.2924 0.3729

We can see that indeed the owl.overfit model is indeed overfitting and does not cross-validate as well as the much simpler model above.

**Monte Carlo Split-Sample Cross-validation function RPART classification models**

> crpart.sscv = function(fit,y,data,B=25,p=.333) {

n = length(y)

These options extract any optional fitting arguments used in the fitting process such as the prior probabilities, loss matrix, complexity parameter (cp), and minsplit/minbucket info. These options are all contained in the tree object fit.

cv <- rep(0,B)

for (i in 1:B) {

ss <- floor(n\*p)

sam <- sample(1:n,ss)

temp <- data[-sam,]

fit2 <- rpart(formula(fit),data=temp,parms=fit$parms,control=fit$control)

ynew <- predict(fit2,newdata=data[sam,],type="class")

tab <- table(y[sam],ynew)

mc <- ss - sum(diag(tab))

cv[i] <- mc/ss

}

cv

}

**Function for calculating the misclassification rate of tree fit using rpart only!**

misclass.rpart = function (tree)

{

temp <- table(predict(tree, type = "class"), tree$y)

cat("Table of Misclassification\n")

cat("(row = predicted, col = actual)\n")

print(temp)

cat("\n\n")

numcor <- sum(diag(temp))

numinc <- length(tree$y) - numcor

mcr <- numinc/length(tree$y)

cat(paste("Misclassification Rate = ", format(mcr,digits = 3)))

cat("\n")

}

**Generic Function for Calculating Misclassification Rate**

This function takes the predicted class from a model (fit) and the actual class for the observations (y).

misclass = function(fit,y) {

temp <- table(fit,y)

cat("Table of Misclassification\n")

cat("(row = predicted, col = actual)\n")

print(temp)

cat("\n\n")

numcor <- sum(diag(temp))

numinc <- length(y) - numcor

mcr <- numinc/length(y)

cat(paste("Misclassification Rate = ",format(mcr,digits=3)))

cat("\n")

}

**15.2 - Bagging for Classification Trees**

As with regression problems bagging (averaging trees built to bootstrap samples of the training data) and random forests (building trees using random selected predictors at each stage) can produce superior predictive performance. We will perform bagging using the bagging() command in the ipred library.

> owl.bag = bagging(species~.,data=Owldiet,coob=T)

> owl.bag

Bagging classification trees with 25 bootstrap replications

Call: bagging.data.frame(formula=species ~ .,data = Owldiet, coob = T)

Out-of-bag estimate of misclassification error: 0.2123

> misclass(predict(owl.bag,newdata=Owldiet),Owldiet$species)

Table of Misclassification

(row = predicted, col = actual)

y

fit annandalfi argentiventer exulans rajah surifer tiomanicus whiteheadi

annandalfi 25 0 0 0 0 0 0

argentiventer 0 37 0 0 0 0 0

exulans 0 0 24 0 0 0 0

rajah 0 0 0 16 0 0 0

surifer 0 0 0 0 10 0 0

tiomanicus 0 0 0 0 0 39 0

whiteheadi 0 0 0 0 0 0 28

Misclassification Rate = 0

The training cases are predicted perfectly with 0% misclassified, however the out-of-bag estimate of the APER is .2123 or 21.23% misclassified. The predict command predicts the class for an observation, if you want posterior probabilities use the argument, type=”prob”, rather than type=”class” which is the default.

**Example 15.3: Italian Olive Oils**

> mod = bagging(Area.name~.,data=Olive.train,coob=T)

> mod

Bagging classification trees with 25 bootstrap replications

Call: bagging.data.frame(formula = Area.name ~ ., data = Olive.train, coob = T)

Out-of-bag estimate of misclassification error: 0.1072

> misclass(predict(mod,newdata=Olive.train),Olive.train$Area.name)

Table of Misclassification

(row = predicted, col = actual)

y

fit Calabria Coastal-Sardinia East-Liguria Inland-Sardinia North-Apulia Sicily South-Apulia Umbria West-Liguria

Calabria 56 0 0 0 0 0 0 0 0

Coastal-Sardinia 0 33 0 0 0 0 0 0 0

East-Liguria 0 0 50 0 0 0 0 0 0

Inland-Sardinia 0 0 0 65 0 0 0 0 0

North-Apulia 0 0 0 0 25 0 0 0 0

Sicily 0 0 0 0 0 36 0 0 0

South-Apulia 0 0 0 0 0 0 206 0 0

Umbria 0 0 0 0 0 0 0 51 0

West-Liguria 0 0 0 0 0 0 0 0 50

Misclassification Rate = 0

Again the training cases are predicted perfectly with 0% misclassified, however the out-of-bag estimate of the APER is .1072 and 10.72% misclassified.

Bagging does improve predictive performance certainly, but the problem is that it does not introduce enough randomness to create markedly different trees which can then be averaged to produce a better predictive model. In the next section we examine random forests for classification which achieve additional randomness by restricting access to the available predictors.

**15.3 - Random Forests for Classification**

The random forest algorithm for classification problems is the same as it was for regression. Trees are fit to bootstrap samples where each split is based on a random sample of the available predictors. The trees are then averaged obtain a final prediction.

**Example 15.3 (cont’d): Italian Olive Oils**

> olive.rf = randomForest(Area.name~.,data=Olive.train,mtry=2,importance=T)

> olive.rf

Call:

randomForest(formula = Area.name ~ ., data = Olive.train, mtry = 2,importance = T)

Type of random forest: classification

Number of trees: 500

No. of variables tried at each split: 2

OOB estimate of error rate: 8.16%

Confusion matrix:

Calabria Coastal-Sardinia East-Liguria Inland-Sardinia North-Apulia Sicily South-Apulia

Calabria 39 0 1 0 0 0 2

Coastal-Sardinia 0 22 0 1 0 0 0

East-Liguria 0 0 35 0 1 1 0

Inland-Sardinia 0 1 0 44 0 0 0

North-Apulia 1 0 3 0 14 1 0

Sicily 6 0 0 0 2 13 6

South-Apulia 0 0 0 0 0 2 150

Umbria 0 0 0 0 0 0 0

West-Liguria 0 0 1 0 0 0 0

Umbria West-Liguria class.error

Calabria 0 0 0.07142857

Coastal-Sardinia 0 0 0.04347826

East-Liguria 1 3 0.14634146

Inland-Sardinia 0 0 0.02222222

North-Apulia 2 0 0.33333333

Sicily 0 0 0.51851852

South-Apulia 0 0 0.01315789

Umbria 43 0 0.00000000

West-Liguria 0 34 0.02857143

Now predicting the growing areas for validation/test olive oils.

> yhat = predict(olive.rf,newdata=Olive.test)  
> misclass(yhat,Olive.test$Area.name)

Table of Misclassification

(row = predicted, col = actual)

y

fit Calabria Coastal-Sardinia East-Liguria Inland-Sardinia North-Apulia Sicily

Calabria 12 0 0 0 0 3

Coastal-Sardinia 0 9 0 0 0 0

East-Liguria 1 0 8 0 0 0

Inland-Sardinia 0 1 0 19 0 0

North-Apulia 0 0 1 1 3 0

Sicily 1 0 0 0 0 4

South-Apulia 0 0 0 0 0 2

Umbria 0 0 0 0 1 0

West-Liguria 0 0 0 0 0 0

y

fit South-Apulia Umbria West-Liguria

Calabria 1 0 0

Coastal-Sardinia 0 0 0

East-Liguria 1 0 1

Inland-Sardinia 1 0 0

North-Apulia 0 0 0

Sicily 0 0 0

South-Apulia 51 0 0

Umbria 0 8 0

West-Liguria 0 0 14

Misclassification Rate = 0.105 or 10.5%

> olive.rf2 = randomForest(Area.name~.,data=Olive.train,mtry=3,importance=T)

> olive.rf2

Call:

randomForest(formula = Area.name ~ ., data = Olive.train, mtry = 3, importance = T)

Type of random forest: classification

Number of trees: 500

No. of variables tried at each split: 3

OOB estimate of error rate: 7.69%

The model with mtry=3 had the same prediction error for the test cases in Olive.test, i.e. 10.5%.

> olive.rf3 = randomForest(Area.name~.,data=Olive.train,mtry=4,importance=T)

> olive.rf3

Call:

randomForest(formula = Area.name ~ ., data = Olive.train, mtry = 4, importance = T)

Type of random forest: classification

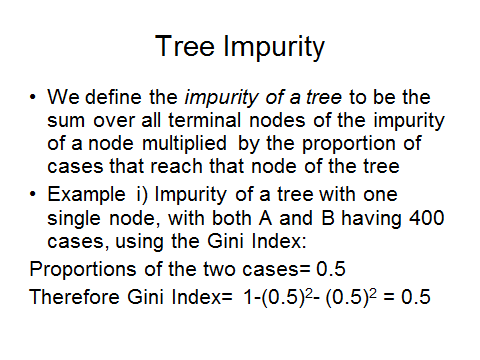
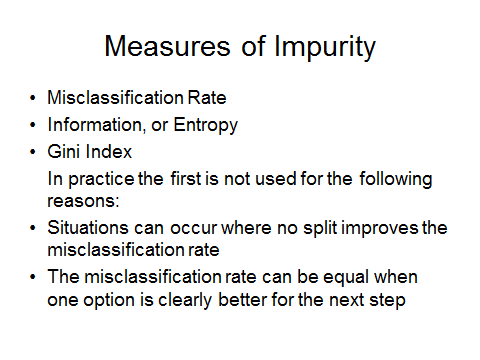
Number of trees: 500

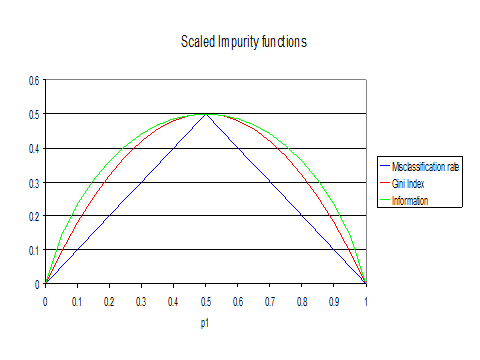
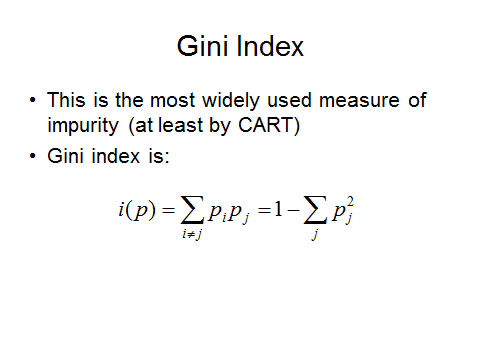
No. of variables tried at each split: 4

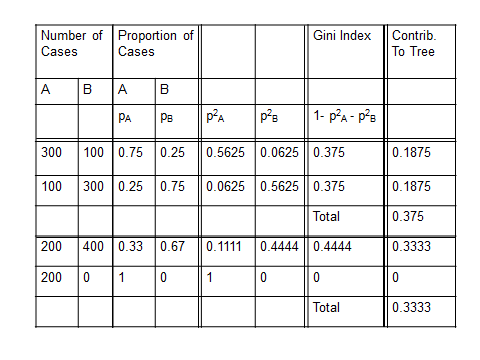
OOB estimate of error rate: 8.86%

The model with mtry=4 had a 14% prediction error for the test/validation cases.

**Measures of Impurity (used to measure variable importance)**







As you can see below the Gini Index decrease is used as a basis for one of the importance measures for random forest models.

> area.rf$importance

Calabria Coastal-Sardinia East-Liguria Inland-Sardinia North-Apulia Sicily South-Apulia Umbria West-Liguria

palmitic 0.13452872 0.073673892 0.02009836 0.063620230 0.219814164 0.02393492 0.035160949 0.18144189 0.07172358

palmitoleic 0.12917763 0.297395878 0.24871128 0.258199312 0.294735044 0.12067000 0.349637005 0.26129882 0.05793994

strearic 0.05485467 0.008206269 0.02237659 0.020664404 0.002113947 0.17771535 0.048116657 0.09939278 0.01070815

oleic 0.11245305 0.282924624 0.23505554 0.355941504 0.230132746 0.06476381 0.165858075 0.69235145 0.24210970

linoleic 0.32933668 0.508924377 0.45492602 0.495736641 0.316481124 0.14388783 0.224371017 0.48554629 0.34768423

eicosanoic 0.23589078 0.031037995 0.02158836 0.007285309 0.012992940 0.02669415 0.008961566 0.05100809 0.16965593

linolenic 0.07932334 0.016359483 0.03856755 0.012864521 0.041563009 0.10790249 0.014568810 0.01612817 0.38582919

eicosenoic 0.28569336 0.189885202 0.47343418 0.166049624 0.307131929 0.25378467 0.080635695 0.39261139 0.23982349

MeanDecreaseAccuracy MeanDecreaseGini

palmitic 0.07276472 28.36385

palmitoleic 0.25472399 102.65161

strearic 0.04839498 23.73749

oleic 0.24516620 96.05318

linoleic 0.33471962 95.28149

eicosanoic 0.05194775 19.67056

linolenic 0.06224676 42.75880

eicosenoic 0.21391468 56.87026

The function below displays a bar graph of a predictor importance measure.

> rfimp.class = function(rffit,measure=1,horiz=T) {

barplot(sort(rffit$importance[,measure]),horiz=horiz,xlab="Importance Measure",main="Variable Importance")

}

> olive.rf$importance

Calabria Coastal-Sardinia East-Liguria Inland-Sardinia North-Apulia Sicily

palmitic 0.16692385 0.18109003 0.06220827 0.25512663 0.23493453 0.05974573

palmitoleic 0.14446453 0.22231111 0.16790365 0.16435717 0.25704015 0.16755492

strearic 0.10512959 0.05339239 0.05916387 0.09467977 0.01429558 0.18388299

oleic 0.15485791 0.31265641 0.27224396 0.35127778 0.17229330 0.07920738

linoleic 0.24935901 0.41369683 0.29939415 0.40484609 0.19038201 0.10259168

eicosanoic 0.27549485 0.08739308 0.09041347 0.04521600 0.06581736 0.13346932

linolenic 0.05099191 0.03691659 0.02475511 0.02722333 0.09766195 0.09593361

South-Apulia Umbria West-Liguria MeanDecreaseAccuracy MeanDecreaseGini

palmitic 0.19570521 0.27925523 0.16628302 0.18386269 50.51496

palmitoleic 0.32799850 0.25562722 0.07215404 0.22913440 67.66979

strearic 0.02356093 0.27073211 0.04895856 0.07983885 32.51946

oleic 0.23570679 0.45225090 0.20585305 0.25328425 68.64811

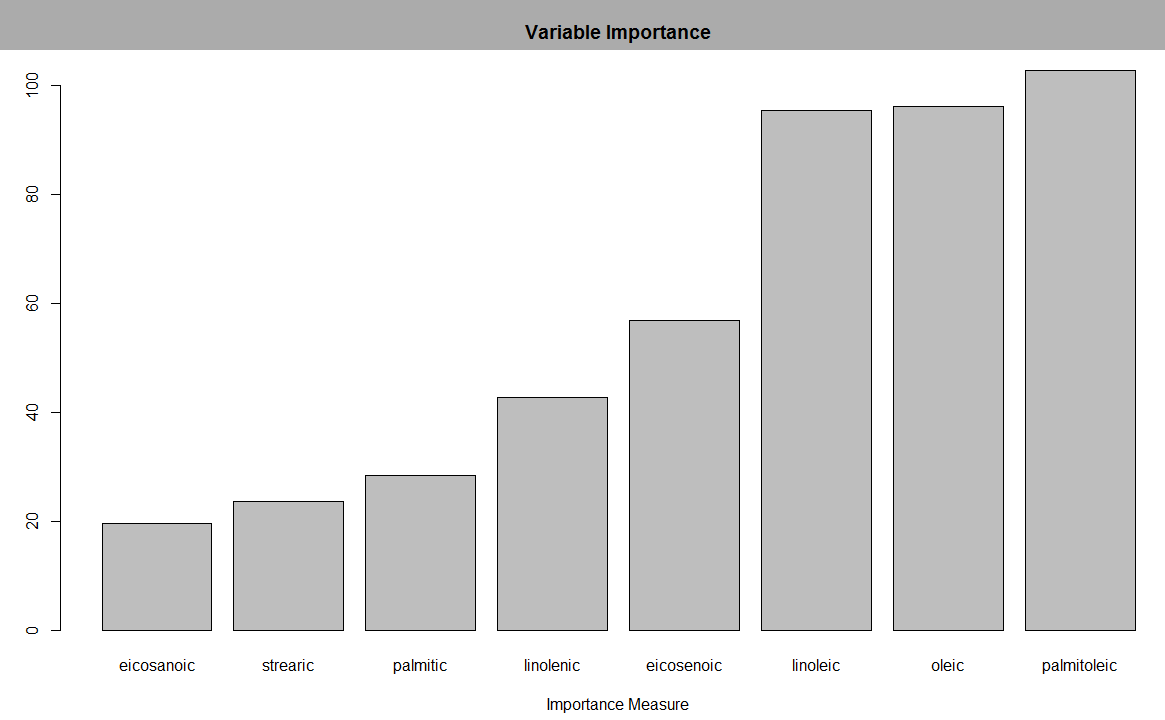
linoleic 0.18042189 0.44868958 0.17454179 0.25365414 64.24496

eicosanoic 0.02456127 0.04709913 0.33404666 0.09664687 33.13443

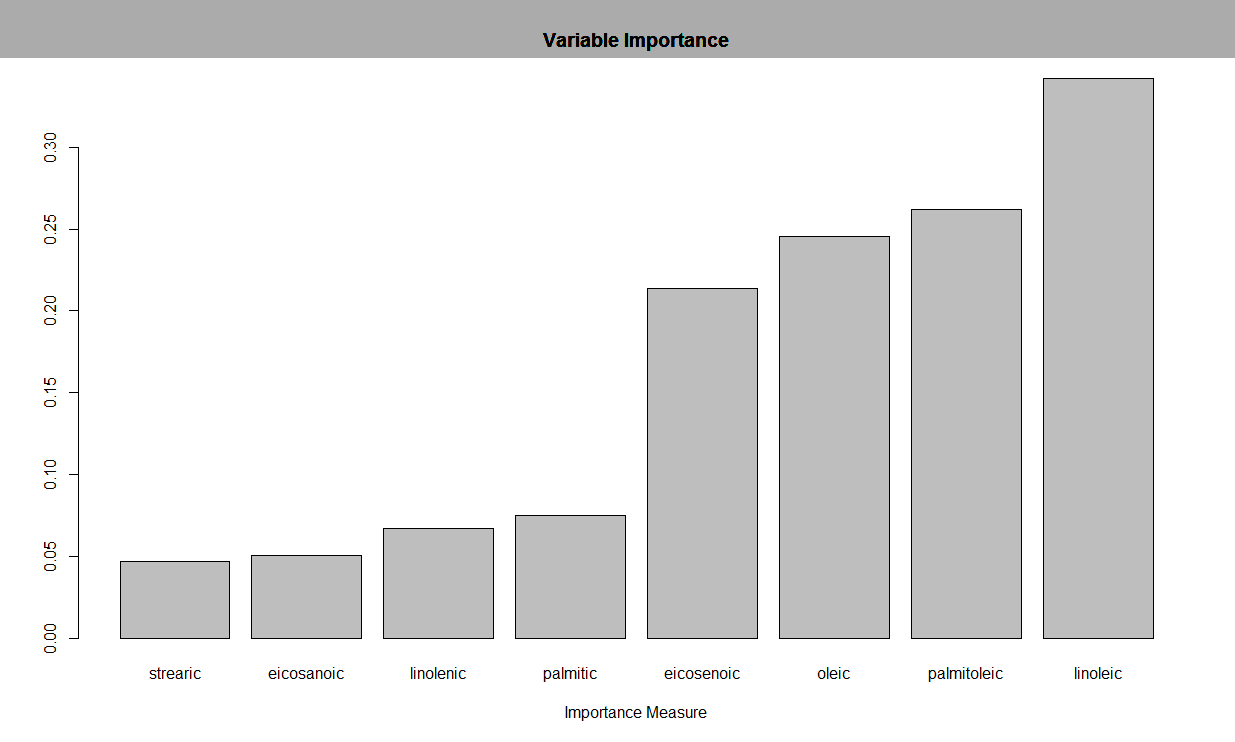
linolenic 0.03241275 0.09179577 0.37615173 0.07351713 33.36514

We have importance measures for classifying into each region separately as well two overall measures in columns 10 and 11 of the importance results.

> rfimp.class(area.rf,measure=11,horiz=F)



> rfimp.class(area.rf,measure=10,horiz=F)



The function below performs MCCV for a Random Forest model for classification.

> crf.sscv = function(fit,y,data,B=25,p=.333,mtry=fit$mtry,ntree=fit$ntree) {

n = length(y)

cv <- rep(0,B)

for (i in 1:B) {

ss <- floor(n\*p)

sam <- sample(1:n,ss)

temp <- data[-sam,]

fit2 <- randomForest(formula(fit),data=temp,mtry=mtry,ntree=ntree)

ynew <- predict(fit2,newdata=data[sam,],type="class")

tab <- table(y[sam],ynew)

mc <- ss - sum(diag(tab))

cv[i] <- mc/ss

}

cv

} **(mtry = 2)**

> results = crf.sscv(olive.rf,Olive.train$Area.name,data=Olive.train)

> summary(results)

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.04930 0.07042 0.09155 0.08563 0.09859 0.12680

**(mtry = 3)**> results = crf.sscv(olive.rf2,Olive.train$Area.name,data=Olive.train)  
> summary(results)

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.03684 0.05263 0.06316 0.06211 0.07368 0.09474

**(mtry = 4)**

> results = crf.sscv(olive.rf3,Olive.train$Area.name,data=Olive.train)

> summary(results)

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.06338 0.07746 0.09859 0.09437 0.11270 0.12680  
 **(mtry = 5)**

> results = crf.sscv(olive.rf3,Olive.train$Area.name,data=Olive.train,mtry=5)

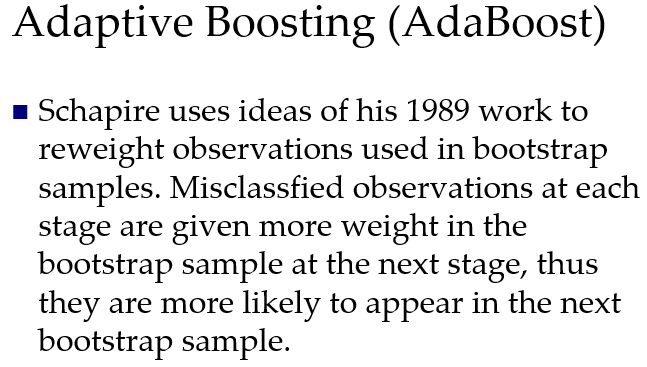
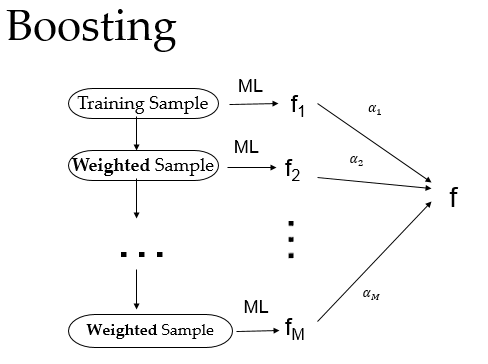
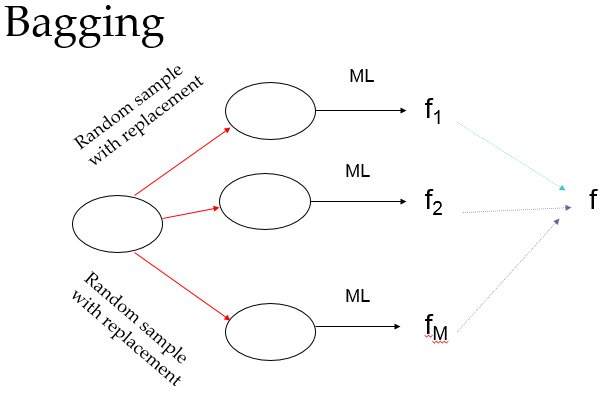
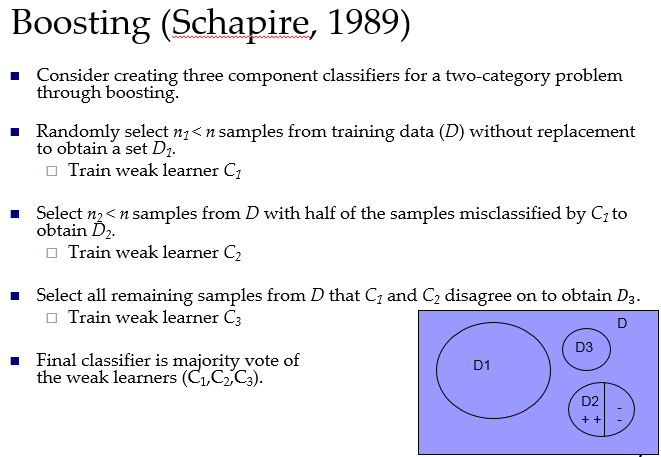
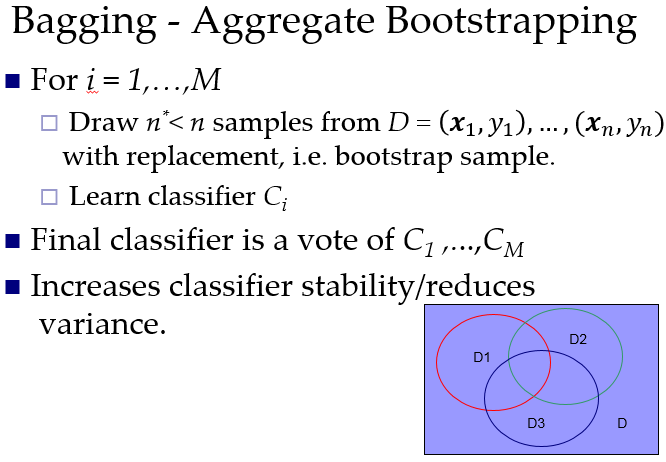
> summary(results)

Min. 1st Qu. Median Mean 3rd Qu. Max.

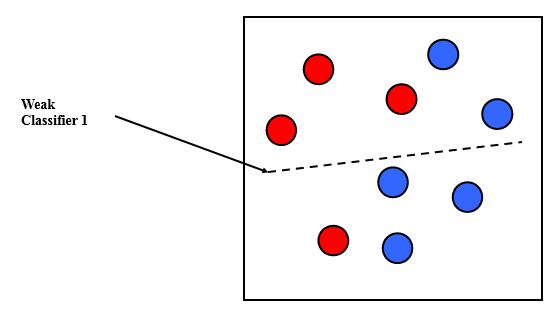
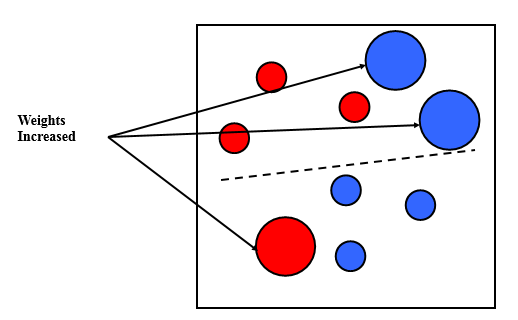
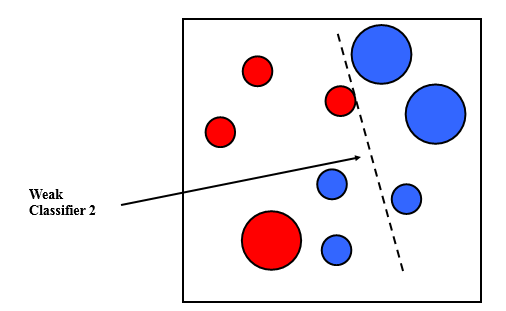
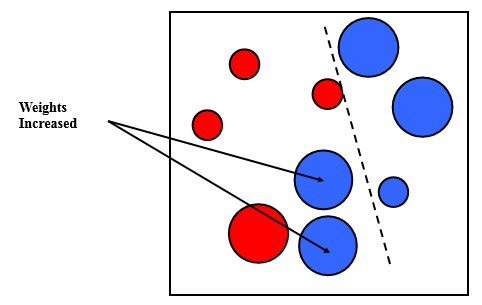
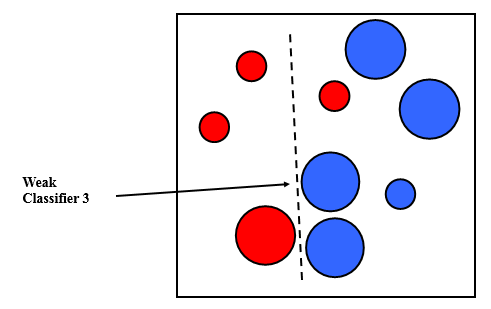
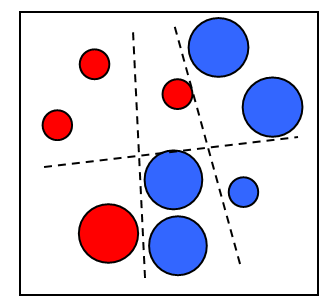
0.02817 0.08451 0.09155 0.09465 0.10560 0.14080

**15.4 – Boosting for Classification Problems**

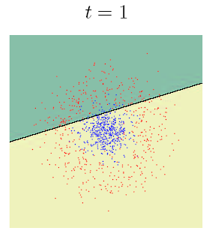
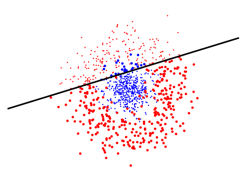
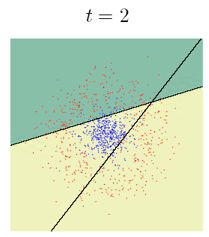
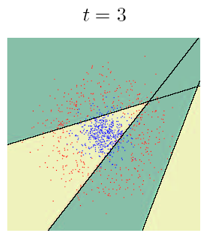
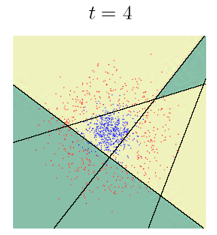
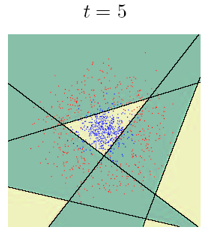
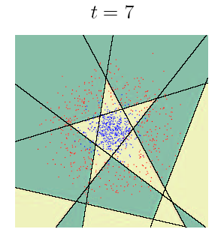
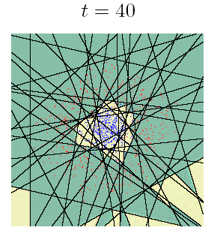
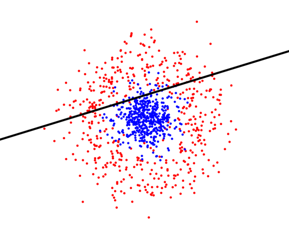
Boosting for classification problems works a little differently than for regression problems. Observations that are misclassified on a given stage in the boosting process are given more weight in next stage. Thus the next model in the sequence will “work harder” to classify those observations correctly. The diagrams below help illustrate this idea and contrast boosting vs. bagging for classification problems.



**The boosting sequence…**

Stage 1 Increase weight on misclassified cases  
   
Stage 2 Increase weight on misclassified cases     
 Final classifier is a combination  
Stage 3 of the three weak classifiers  
 

**Another example:**

Initial classifier

Combination of all 40 classifiers.

Increase weights on misclassified cases