**5.0 - Classification vs. Regression Problems**

In classification problems the response is non-numeric, though it might be coded numerically (e.g. 1 = “Yes”, 0 = “No”). In general the response is either nominal or possibly ordinal, thus the prediction problem is to develop a “model” or “rule” for classifying an what category/level an observation is given a set of predictors . As was the case with regression problems the predictors can be any mixture of data types (continuous/numeric, ordinal, or nominal) and must dealt with accordingly in developing our predictive model.   
  
For some methods we have covered the extension to classification problems should be clear. For example, the nearest neighbor method covered in Section 11 is easily extended to the classification problem. Rather than take the mean or weighted mean of the response for the nearest neighbors we simply classify an observation to category/level determined by it’s nearest neighbors. For example we could simply classify each new/future observation to the class of its nearest neighbor (i.e. ) in the training data. CART and it’s enhancements (Bagging, Random Forest, Boosting, etc.) also easily extend to classification problems. We build the tree the “usual way” and let majority rule in each terminal node. We will have to adjust the “goodness of split” criterion in order to build the tree, but it should be clear that tree-based models are well suited for classification problems.

With the possible exception of nearest neighbor classification and support vector machines (SVM), which we will cover later, the “fitted values” from a classification model are the estimated probability a -level response is in class given a set of predictor values .

That is our model returns, and we can classify an observation to the class with highest estimated conditional probability.  
  
Because our estimated conditional probabilities need to satisfy the following conditions:  
  
   
  
 2)   
  
we will to somehow impose these restrictions into the modeling process.

**5.1 – Measuring Prediction Error for Classification Problems**

The basic measures of prediction quality for classification problem are the ***accuracy*** and the ***misclassification rate*** These defined simply as:

Certainly these make sense intuitively, but they are not ONLY measures that can be used. For example if certain misclassifications are more “expensive” to make, then we would like to take that information into account when developing our classification rule. The problems with Accuracy/Misclassification Rate are summarized below.

1. Accuracy/Misclassification Rate assume equal cost for all errors.
2. Is a 99% Accuracy always good? It could excellent, good, mediocre, poor, or flat out terrible depending on the situation. For example, in a binary classification problem (i.e. the response has two levels – “0” and “1”) where 99% of the training data are 0’s and 1% of the training data are 1’s, a rule that says the predicted class for regardless of will be 99% accurate and have a misclassification rate of 1%. What constitutes a good Accuracy depends on the ***Base Accuracy Rate*** which comes from predicting an observation is from the predominant class. In the above example, the Base Accuracy Rate is 99%.
3. The example in (2) above illustrates the need to consider the percent reduction in error or misclassification rate. As an example suppose the Base Accuracy Rate is 80% and hence the Base Misclassification Rate is 20%. Suppose we develop a statistical learning model the increases accuracy from 80% to 90%, thereby reducing the misclassification rate from 20% to 10%. Thus a statistical learning model reduces error by 50%. Similarly increasing accuracy from 99.9% to 99.99% represents a 90% reduction in error and increasing accuracy from 50% to 80% represents a 60% reduction in error. The idea of percent improvement can be applied to other measures we will consider below as well.

**More Measures of Predictive Performance**

For simplicity we consider the binary classification problem, i.e. the response has two levels (1 = “Yes”, 0 = “No”), when considering prediction accuracy measures in more detail. The accuracy and misclassification rate above can be represented using a ***confusion matrix***.

|  |  |  |  |
| --- | --- | --- | --- |
| Truth | Predicted 1 | Predicted 0 | Row Totals |
| True 1 | a | b | a+b |
| True 0 | c | d | c+d |
|  | a+c | b+d | n = (a+b+c+d) |

Other terminology associated with the confusion matrix,

The ***sensitivity*** is (this is also called the ***Recall*** or True Positive Rate (TPR))

The ***specificity*** of a classification rule is

The ***false positive*** rate (FPR) =

The ***false negative*** rate (FNR) =

What is sometimes of more interest is the predictive value of a classification rule.

The ***positive predictive value (PPV)*** = (this is also called ***Precision***)

The ***negative predictive value (NPV) =***

Usually these come from application of ***Baye’s Rule***:  
  
**PPV**

**NPV**

Calculating these using Baye’s Rule requires that we have some prior knowledge about the probabilities that and in the population of interest, i.e. we need to know or have estimates of and .

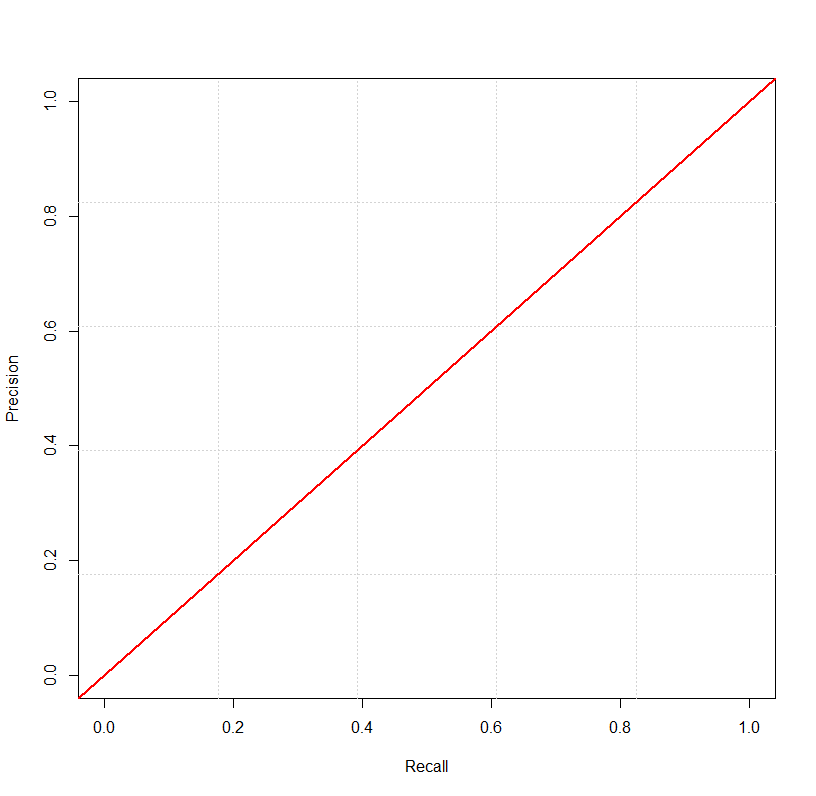
The classification rule for the binary prediction problem will classify an observation given a set of predictor values according to the following rule,

The “logical” choice for , is though in theory we can use any value and there are good reasons for using something other in some cases. As we change the value of the performance of the classification rule change using **any** of the measures above.

Other performance measures that will change with the choice of for a classification rule are the ***F-measure or F1-Score*** and the ***Break Even Point***. The F-measure and Break Even Points are determined by the Precision (PPV) and Recall (Sensitivity or TPR).

and the Break Even Point is the choice of that gives,

The plot below illustrates the Break Even Point.



Break Even Point = .6627

Worse Performance

Better Performance

Remember:

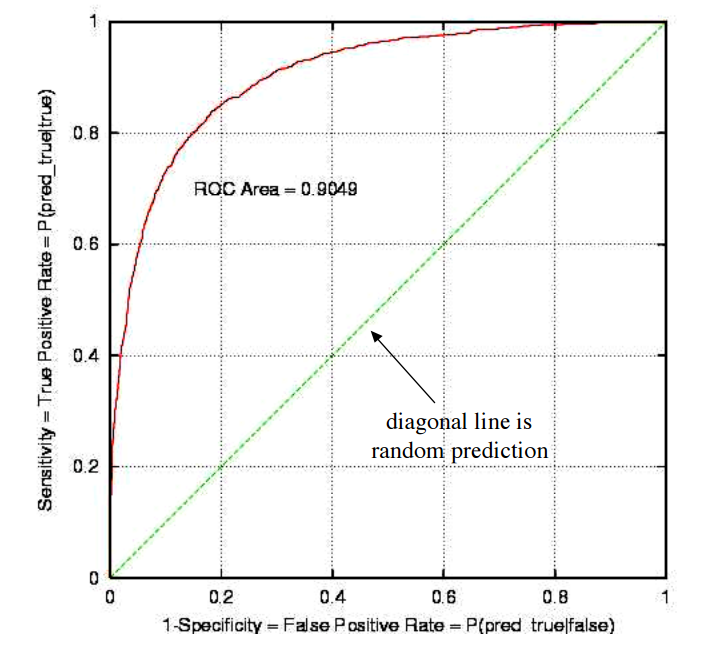
* Recall or TPR or Sensitivity =
* Precision or PPV =

**Receiver Operating Curves (ROC)**

The Receiver Operating Curve (ROC) is a plot of:  
  
 ***Sensitivity*** (Recall) vs. ***1 – Specificity*** (False Positive Rate)

Ideally we would like the sensitivity to be large (near 1) and the 1 – Specificity or the False Positive Rate to be small (near 0). These quantities change as we change the threshold probability in our classification rule which is shown below:

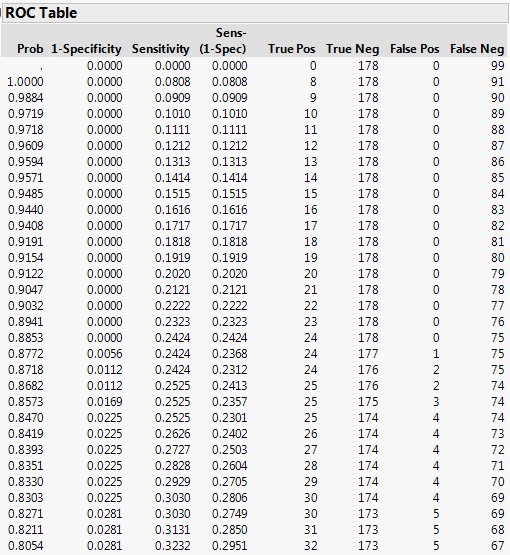
Below is a plot of the ROC for an excellent classification rule for a binary prediction problem. The diagonal line represents the ROC curve for a complete random prediction. The curve represents the predictive performance of a statistical learning model, e.g. from a random forest.



The area beneath the ROC “curve” for a complete random prediction rule is always .50 or 50%. As the predictive performance of classification rule improves the area beneath the ROC curve approaches 1 or 100%. The following adjective scale can be used to discuss the performance of a predictive model in terms the area under the ROC curve, referred to as the AUC (**a**rea **u**nder the **c**urve).

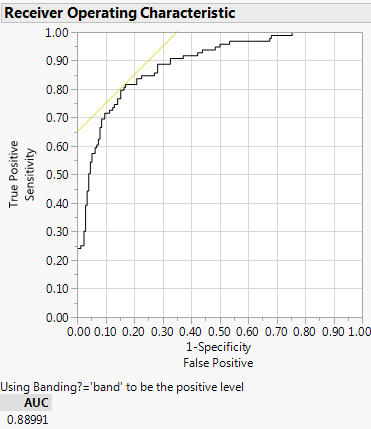
|  |  |
| --- | --- |
| **AUC** | **Predictive Performance** |
| 1.0 or 100% | Perfect Prediction! |
| .90 or 90% | Excellent |
| .80 or 80% | Good |
| .70 or 70% | Mediocre |
| .60 or 60% | Poor |
| .50 or 50% | Random (coin flip) |
| < .50 or 50% | Something is wrong! |

Below is an example of a ROC for a binary prediction problem where we are trying to predict whether or not a printing job will have a problem/defect called “banding”.



The **Prob** column is the table is the cutoff which is used to determine if we predicting banding, i.e.

As decreases sensitivity increases, but so does the false positive rate (FPR) = 1 – specificity.

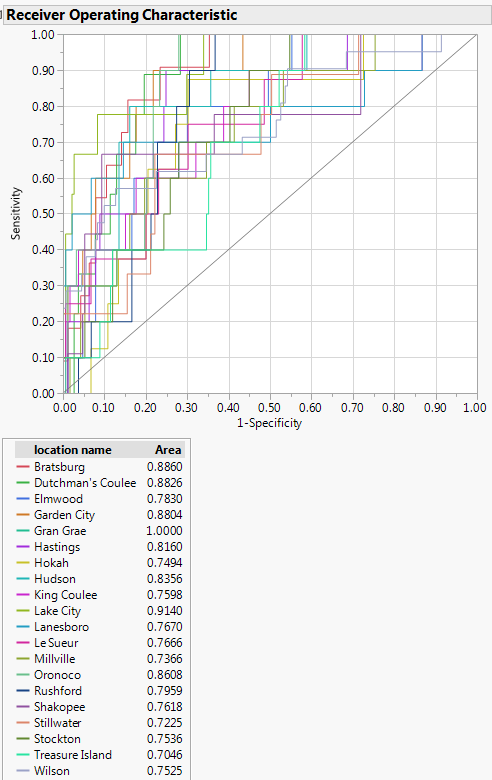


Area Under Curve (AUC) = .8899 or 88.99% nearly excellent prediction!

Small

Large

Good choice for

**Multi-class ROC**Below are the ROC’s for a multi-class prediction problem. Here we are predicting where a soil sample was taken from given the log-concentrations of scandium (Sc), samarium (Sm), uranium (U), and iron (Fe). Some locations are well predicted based on their AUC while others are difficult to classify accurately (e.g. Treasure Island – yes the Casino near Red Wing).  


**Concept of Lift**

Finally we consider another graphical tool for assessing the effectiveness of, or value added by, a statistical learning model for classification problems. We will again focus on the binary prediction problem where the response has two levels.

**Example: Customer Marketing”**Suppose a company wants to do a mail marketing campaign. It costs the company $1 for each mailing. Further suppose they have information on 100,000 potential customers and based on previous mailings they now that approximately 20% of customers positively respond to their mailings. The company has now employed a predictive analyst who knows how to do estimate using data collected from previous mailings. As this information is also available for the 100,000 potential customers being considered for the current mailing the predictive analyst can estimate for these potential customers as well and rank them accordingly.

***Strategy 1***: Mail all 100,000 potential customers

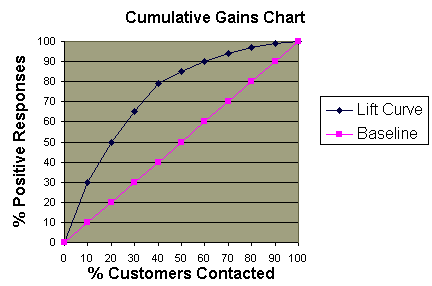
|  |  |  |
| --- | --- | --- |
| Cost  ($) | Total Customers Contacted | Positive Responses |
| 100,000 | 100,000 | 20,000 |

***Strategy 2:*** Use the predictive model to rank customers based upon   
Based upon the model we can estimate the number of positive responses based upon the number of customers contacted ().

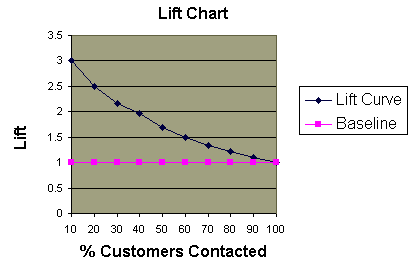
|  |  |  |
| --- | --- | --- |
| Cost ($) | Total Customers Contacted | Expected Positive Responses |
| 10,000 | 10,000 | 6,000 |
| 20,000 | 20,000 | 10,000 |
| 30,000 | 30,000 | 13,000 |
| 40,000 | 40,000 | 15,800 |
| 50,000 | 50,000 | 17,000 |
| 60,000 | 60,000 | 18,000 |
| 70,000 | 70,000 | 18,800 |
| 80,000 | 80,000 | 19,400 |
| 90,000 | 90,000 | 19,800 |
| 100,000 | 100,000 | 20,000 |

Notice the diminishing number of positive responses per 10,000 mailings.

The chart below, called a ***Cumulative Gains Chart***, shows the expected % of positive responses vs. the percentage of the 100,000 customers contacted. The baseline comes from fact that if we mail customers without ranking them we can expect about 10% the 20,000 positive responses per 10% mailed. If we consider the first point on the lift curve we see it has coordinates (10%,30%), which comes from the fact in the first 10,000 **ranked** customers we expect 6,000 positive responses which represents 30% of the 20,000 expected positive responses in the 100,000 potential customers. As the gain diminishes as we move further down our ranked list the lift curve begin taper markedly as the % of customer contacted increases.

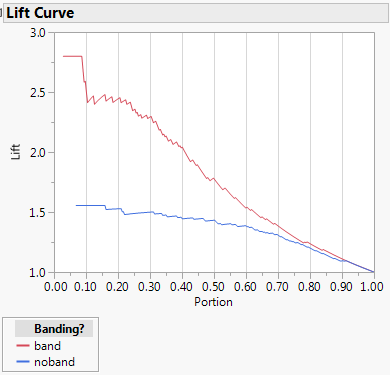


A ***lift chart*** plots the ratio of points on the lift curve relative to the points on the baseline vs. the % of customers contacted.



Again we see the diminishing returns, i..e. decreased lift, as the % of customers contacted increases.

**Example – Lift Curve for the Banding Problem**



By considering only 10% of the ranked process settings based upon our model we will identify about 2.7 times as many scenarios (process settings) where banding is present then if we use no model. Similarly we will identify over 1.5 times as many scenarios where banding is NOT present by using our model.

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

For classification the goals is to a tree-based model that will classify observations into one of *m* 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 terminal node defined by the set of predictors .

The neighborhoods 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. However in classification problems it is often times the case 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 most costly misclassifications on our training data set. We will not delve that deeply into these concepts in this workshop. Our accuracy measure throughout our discussion of tree-based models for classification will be the misclassification rate. The function below will take the role of our PredAcc function for regression problems. This function will take our predicted class membership (fit) and compare it to the actual class (y) in a table and report the proportion of observations misclassified.

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

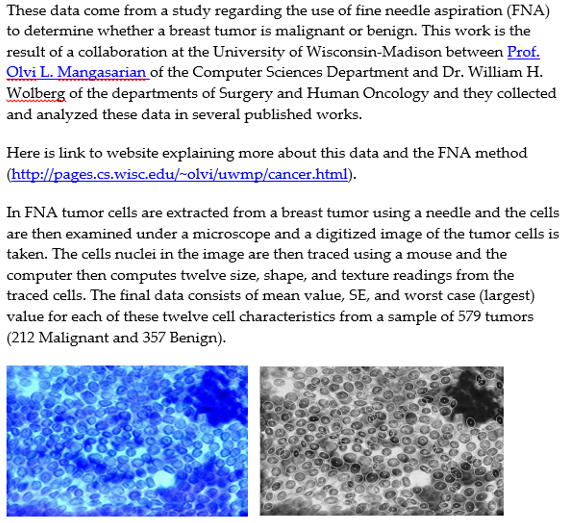
mcr <- numinc/length(y)

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

cat("\n")

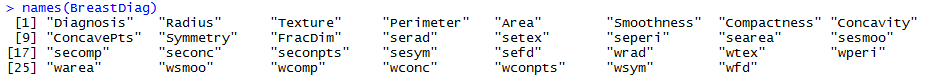
}

**Motivating Example: Breast Cancer Diagnosis Via Fine Needle Aspiration**

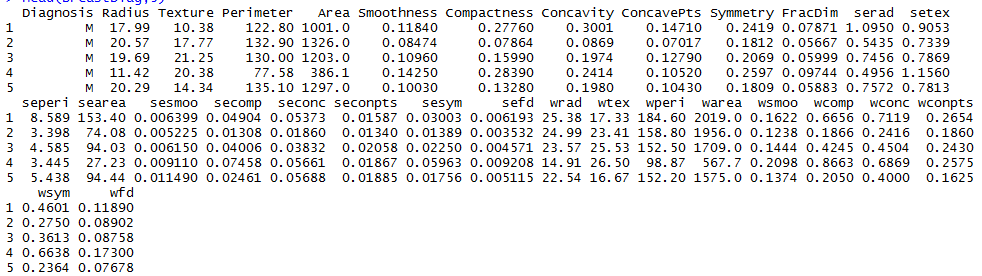


The data collected from this study are contained in the file **BreastDiag.csv**.

> BreastDiag = read.table(file.choose(),header=T,sep=”,”)

> names(BreastDiag)  


> head(BreastDiag,5)



> str(BreastDiag)  
'data.frame': 569 obs. of 31 variables:

$ Diagnosis : Factor w/ 2 levels "B","M": 2 2 2 2 2 2 2 2 2 2 ...

$ Radius : num 18 20.6 19.7 11.4 20.3 ...

$ Texture : num 10.4 17.8 21.2 20.4 14.3 ...

$ Perimeter : num 122.8 132.9 130 77.6 135.1 ...

$ Area : num 1001 1326 1203 386 1297 ...

$ Smoothness : num 0.1184 0.0847 0.1096 0.1425 0.1003 ...

$ Compactness: num 0.2776 0.0786 0.1599 0.2839 0.1328 ...

$ Concavity : num 0.3001 0.0869 0.1974 0.2414 0.198 ...

$ ConcavePts : num 0.1471 0.0702 0.1279 0.1052 0.1043 ...

$ Symmetry : num 0.242 0.181 0.207 0.26 0.181 ...

$ FracDim : num 0.0787 0.0567 0.06 0.0974 0.0588 ...

$ serad : num 1.095 0.543 0.746 0.496 0.757 ...

$ setex : num 0.905 0.734 0.787 1.156 0.781 ...

$ seperi : num 8.59 3.4 4.58 3.44 5.44 ...

$ searea : num 153.4 74.1 94 27.2 94.4 ...

$ sesmoo : num 0.0064 0.00522 0.00615 0.00911 0.01149 ...

$ secomp : num 0.049 0.0131 0.0401 0.0746 0.0246 ...

$ seconc : num 0.0537 0.0186 0.0383 0.0566 0.0569 ...

$ seconpts : num 0.0159 0.0134 0.0206 0.0187 0.0188 ...

$ sesym : num 0.03 0.0139 0.0225 0.0596 0.0176 ...

$ sefd : num 0.00619 0.00353 0.00457 0.00921 0.00511 ...

$ wrad : num 25.4 25 23.6 14.9 22.5 ...

$ wtex : num 17.3 23.4 25.5 26.5 16.7 ...

$ wperi : num 184.6 158.8 152.5 98.9 152.2 ...

$ warea : num 2019 1956 1709 568 1575 ...

$ wsmoo : num 0.162 0.124 0.144 0.21 0.137 ...

$ wcomp : num 0.666 0.187 0.424 0.866 0.205 ...

$ wconc : num 0.712 0.242 0.45 0.687 0.4 ...

$ wconpts : num 0.265 0.186 0.243 0.258 0.163 ...

$ wsym : num 0.46 0.275 0.361 0.664 0.236 ...

$ wfd : num 0.1189 0.089 0.0876 0.173 0.0768 ...

> par(mfrow=c(3,2))

> plot(Radius~Diagnosis,data=BreastDiag,col="red")

> plot(Texture~Diagnosis,data=BreastDiag,col="red")

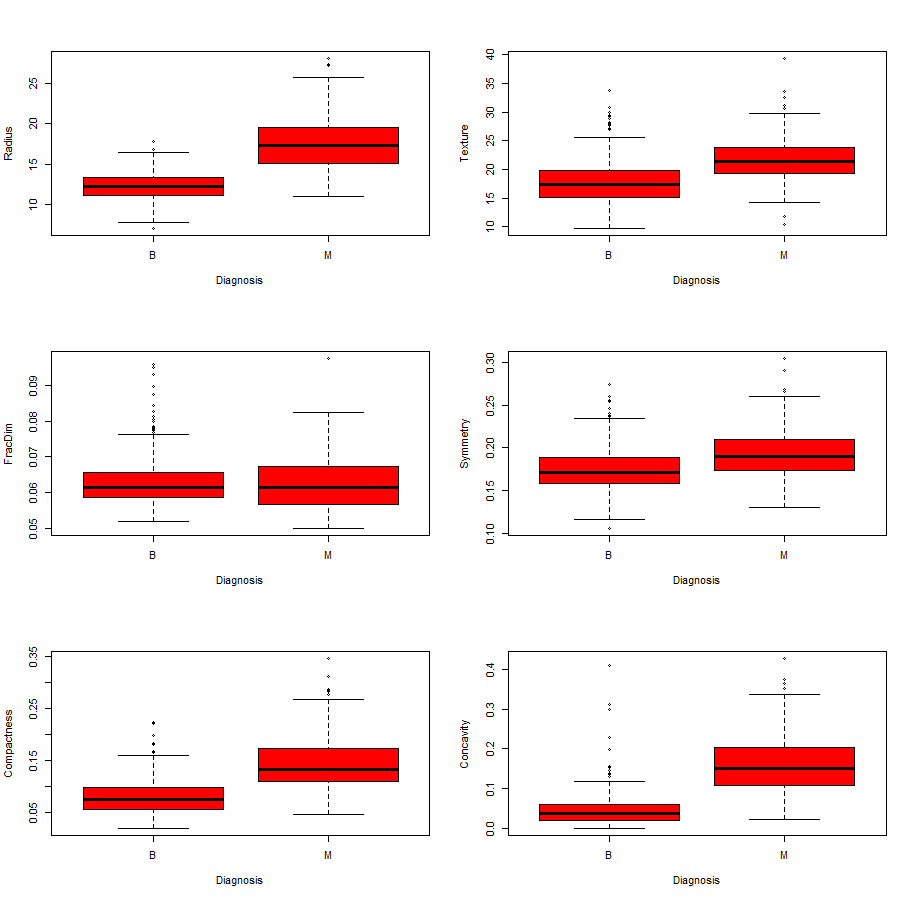
> plot(FracDim~Diagnosis,data=BreastDiag,col="red")

> plot(Symmetry~Diagnosis,data=BreastDiag,col="red")

> plot(Compactness~Diagnosis,data=BreastDiag,col="red")

> plot(Concavity~Diagnosis,data=BreastDiag,col="red")

> par(mfrow=c(1,1))



We could consider many other possibly interesting plots, but let’s begin developing a model for predicting the cancer status of breast tumor patients based upon the cells sampled via FNA.

For tree-based classifiers we have considered RPART, bagged RPART, random forests, boosted trees, and Cubist in the case of regression. The way we fit these models for classification problems is pretty much the same. However, any cross-validation function we develop would need to use misclassification rate (or some other measure e.g. AUC from ROC curve) to measure performance of our model. Also when examining the performance of model using training and validation/test sets we would want to consider similar measures.

Before we fit a basic RPART model to these data, let’s form a training and test set from the original database.

> sam = sample(1:569,floor(569\*.6666),replace=F)

> BCtrain = BreastDiag[sam,]

> BCtest = BreastDiag[-sam,]

> dim(BCtrain)

[1] 379 31

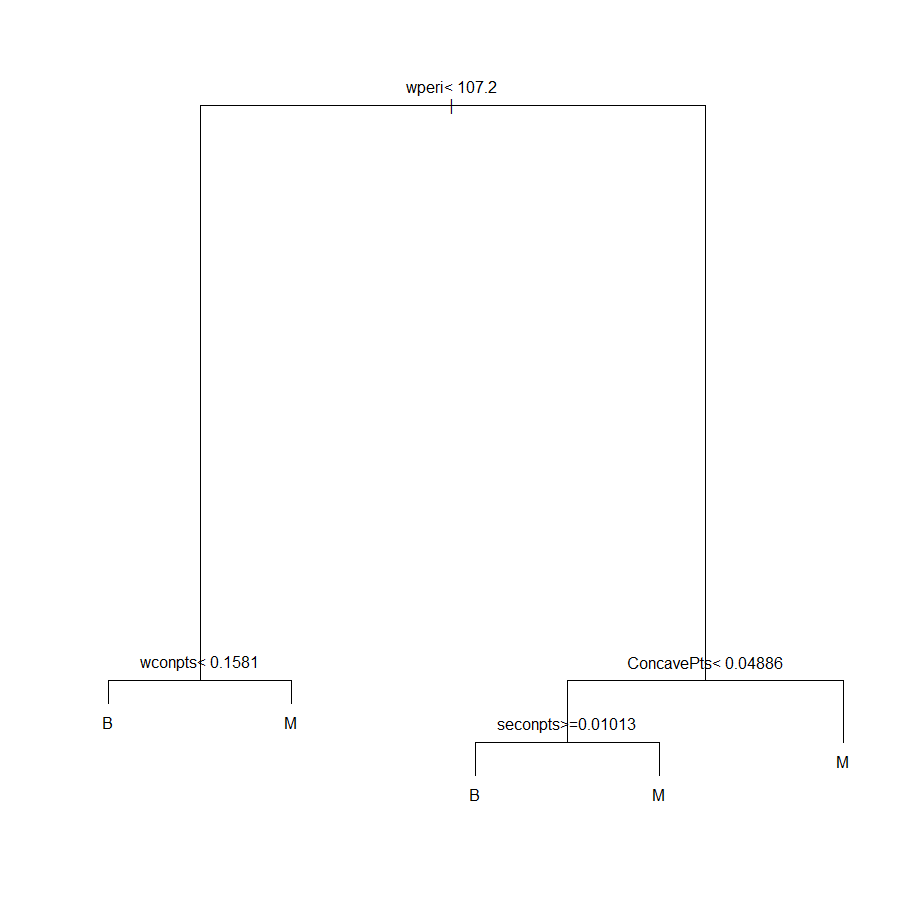
> dim(BCtest)

[1] 190 31

> bc.rpart = rpart(Diagnosis~.,data=BCtrain)

> plot(bc.rpart)

> text(bc.rpart)



The default tree is very simple looking. How well does it fit our training cases?

> yfit = predict(bc.rpart,type="class")

> misclass(yfit,BCtrain$Diagnosis)

Table of Misclassification

(row = predicted, col = actual)

y

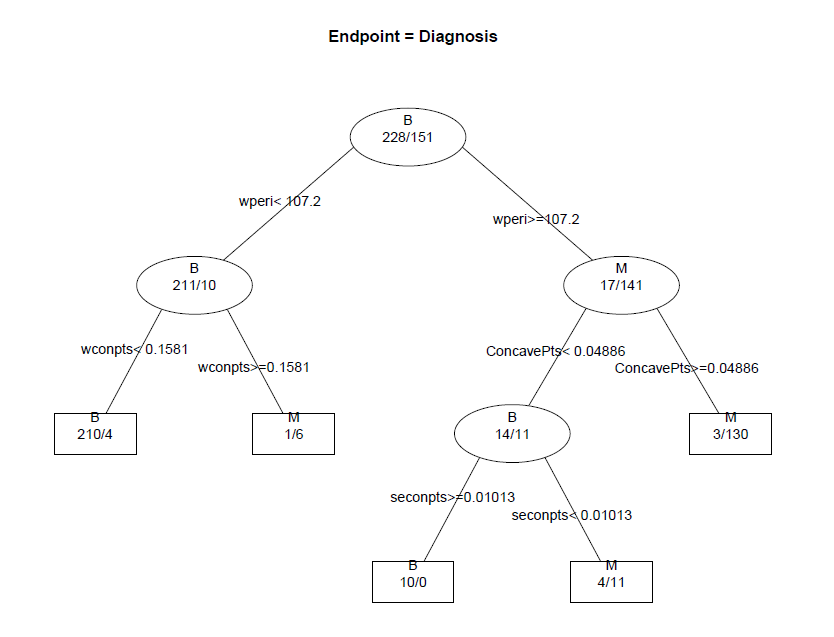
fit B M

B 220 4

M 8 147

Misclassification Rate = 0.0317

> post(bc.rpart)



> yprob = predict(bc.rpart)

> head(yprob,5)

B M

101 0.2666667 0.73333333

148 0.9813084 0.01869159

111 0.9813084 0.01869159

201 0.9813084 0.01869159

562 0.9813084 0.01869159

> summary(bc.rpart)

Call:

rpart(formula = Diagnosis ~ ., data = BCtrain)

n= 379

CP nsplit rel error xerror xstd

1 0.82119205 0 1.0000000 1.0000000 0.06311889

2 0.03311258 1 0.1788079 0.2384106 0.03780087

3 0.01000000 4 0.0794702 0.1456954 0.03014733

Variable importance

wperi warea wrad Perimeter Area Radius wconpts ConcavePts Concavity Compactness

16 16 16 14 14 13 2 2 2 1

secomp seconpts wcomp wsym seconc wconc

1 1 1 1 1 1

Node number 1: 379 observations, complexity param=0.8211921

predicted class=B expected loss=0.3984169 P(node) =1

class counts: 228 151

probabilities: 0.602 0.398

left son=2 (221 obs) right son=3 (158 obs)

Primary splits:

wperi < 107.2 to the left, improve=132.2413, (0 missing)

warea < 868.2 to the left, improve=131.7077, (0 missing)

wrad < 16.79 to the left, improve=131.5531, (0 missing)

wconpts < 0.14235 to the left, improve=127.1001, (0 missing)

ConcavePts < 0.04923 to the left, improve=120.5394, (0 missing)

Surrogate splits:

warea < 784.15 to the left, agree=0.979, adj=0.949, (0 split)

wrad < 16.16 to the left, agree=0.976, adj=0.943, (0 split)

Perimeter < 90.37 to the left, agree=0.937, adj=0.848, (0 split)

Area < 641.7 to the left, agree=0.929, adj=0.829, (0 split)

Radius < 14.43 to the left, agree=0.926, adj=0.823, (0 split)

Node number 2: 221 observations, complexity param=0.03311258

predicted class=B expected loss=0.04524887 P(node) =0.5831135

class counts: 211 10

probabilities: 0.955 0.045

left son=4 (214 obs) right son=5 (7 obs)

Primary splits:

wconpts < 0.1581 to the left, improve=9.530270, (0 missing)

wcomp < 0.43675 to the left, improve=6.471525, (0 missing)

wsym < 0.3617 to the left, improve=5.579764, (0 missing)

wconc < 0.3695 to the left, improve=5.307828, (0 missing)

wsmoo < 0.1701 to the left, improve=4.886427, (0 missing)

Surrogate splits:

wcomp < 0.43375 to the left, agree=0.986, adj=0.571, (0 split)

wsym < 0.3617 to the left, agree=0.986, adj=0.571, (0 split)

Concavity < 0.16185 to the left, agree=0.982, adj=0.429, (0 split)

wconc < 0.53855 to the left, agree=0.982, adj=0.429, (0 split)

ConcavePts < 0.089435 to the left, agree=0.977, adj=0.286, (0 split)

Node number 3: 158 observations, complexity param=0.03311258

predicted class=M expected loss=0.1075949 P(node) =0.4168865

class counts: 17 141

probabilities: 0.108 0.892

left son=6 (25 obs) right son=7 (133 obs)

Primary splits:

ConcavePts < 0.048865 to the left, improve=12.157110, (0 missing)

Concavity < 0.07214 to the left, improve=11.940700, (0 missing)

wperi < 117.45 to the left, improve=10.580390, (0 missing)

wtex < 19.91 to the left, improve= 9.531913, (0 missing)

wconpts < 0.1454 to the left, improve= 9.486784, (0 missing)

Surrogate splits:

Concavity < 0.06578 to the left, agree=0.962, adj=0.76, (0 split)

wconpts < 0.13655 to the left, agree=0.924, adj=0.52, (0 split)

Compactness < 0.085955 to the left, agree=0.918, adj=0.48, (0 split)

seconc < 0.01949 to the left, agree=0.905, adj=0.40, (0 split)

secomp < 0.01462 to the left, agree=0.899, adj=0.36, (0 split)

Node number 4: 214 observations

predicted class=B expected loss=0.01869159 P(node) =0.5646438

class counts: 210 4

probabilities: 0.981 0.019

Node number 5: 7 observations

predicted class=M expected loss=0.1428571 P(node) =0.01846966

class counts: 1 6

probabilities: 0.143 0.857

Node number 6: 25 observations, complexity param=0.03311258

predicted class=B expected loss=0.44 P(node) =0.06596306

class counts: 14 11

probabilities: 0.560 0.440

left son=12 (10 obs) right son=13 (15 obs)

Primary splits:

seconpts < 0.0101255 to the right, improve=6.453333, (0 missing)

secomp < 0.01497 to the right, improve=4.787532, (0 missing)

wtex < 26.375 to the left, improve=4.787532, (0 missing)

wrad < 16.825 to the left, improve=4.555294, (0 missing)

warea < 929.8 to the left, improve=4.435385, (0 missing)

Surrogate splits:

secomp < 0.02017 to the right, agree=0.84, adj=0.6, (0 split)

wrad < 16.635 to the left, agree=0.84, adj=0.6, (0 split)

warea < 838.6 to the left, agree=0.84, adj=0.6, (0 split)

Compactness < 0.097055 to the right, agree=0.80, adj=0.5, (0 split)

sefd < 0.002474 to the right, agree=0.80, adj=0.5, (0 split)

Node number 7: 133 observations

predicted class=M expected loss=0.02255639 P(node) =0.3509235

class counts: 3 130

probabilities: 0.023 0.977

Node number 12: 10 observations

predicted class=B expected loss=0 P(node) =0.02638522

class counts: 10 0

probabilities: 1.000 0.000

Node number 13: 15 observations

predicted class=M expected loss=0.2666667 P(node) =0.03957784

class counts: 4 11

probabilities: 0.267 0.733

Again we see that the graphical version of the tree is much easier to understand but there is some useful information in this output, however we won’t discuss it.

We can increase the size of the classification tree (at the risk of overfitting) by lowering the cp, lowering minsplit and/or minbucket.

> bc.rpart2 = rpart(Diagnosis~.,data=BCtrain,cp=.0001,minsplit=4)

> yfit = predict(bc.rpart2)

> yfit = predict(bc.rpart2,type="class")

> misclass(yfit,BCtrain$Diagnosis)

Table of Misclassification

(row = predicted, col = actual)

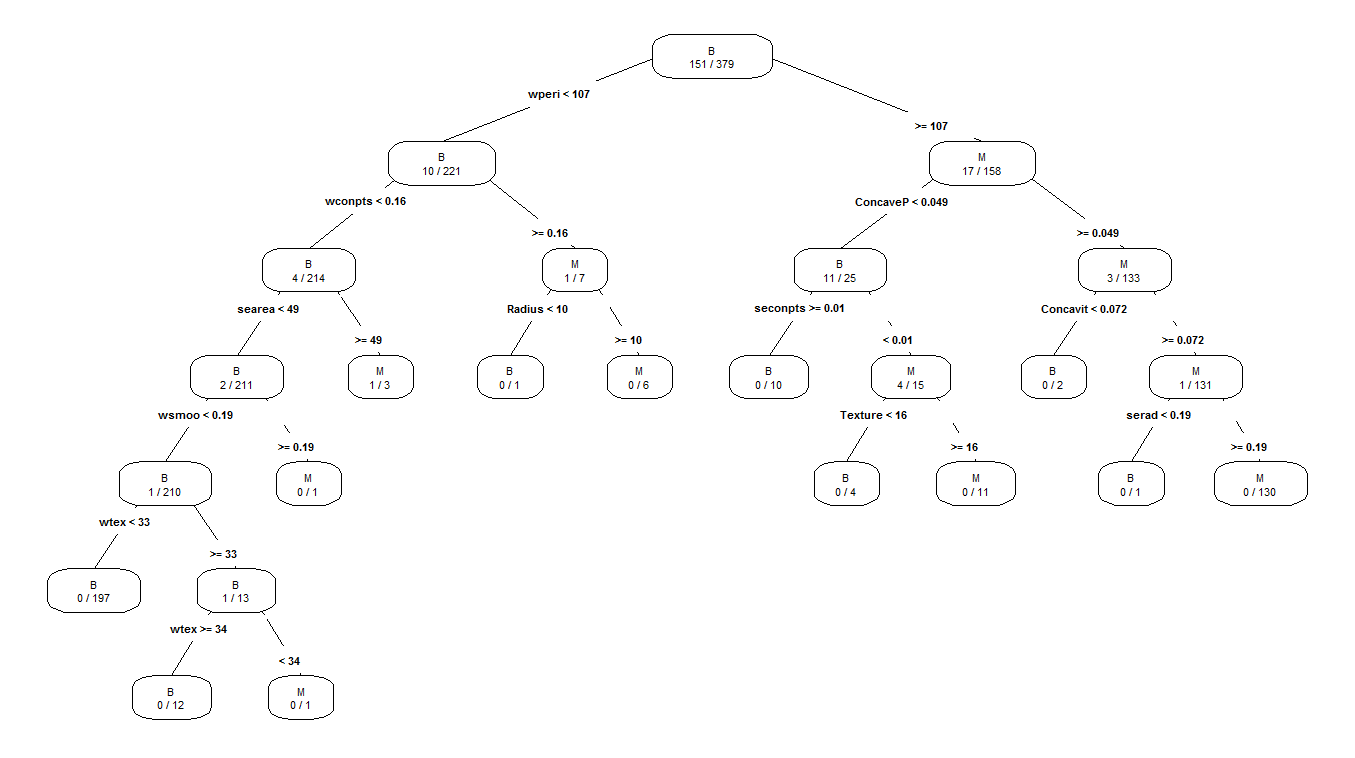
y

fit B M

B 227 0

M 1 151

Misclassification Rate = 0.00264

> prp(bc.rpart2,type=4,extra=3,cex=.7)  


This model fits the training data well with only one subject misclassified! How do these two models compare in terms of the misclassification rate.

> ypred = predict(bc.rpart,newdata=BCtest,type="class")

> misclass(ypred,BCtest$Diagnosis)

Table of Misclassification

(row = predicted, col = actual)

y

fit B M

B 122 5

M 7 56

Misclassification Rate = 0.0632

> ypred2 = predict(bc.rpart2,newdata=BCtest,type="class")

> misclass(ypred2,BCtest$Diagnosis)

Table of Misclassification

(row = predicted, col = actual)

y

fit B M

B 120 7

M 9 54

Misclassification Rate = 0.0842

Here we see that the larger model does not predict as well as the simpler model. Thus it would be wise to use some form of cross-validation to select the optimal RPART model for prediction purposes.

The function below will perform Monte Carlo Split Sample CV for RPART classification tree. It will extract the tuning parameter settings from the fit (i.e. cp, minsplit, minbucket, etc.). It will randomly split our data into 2/3 training and 1/3 validation sets.

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

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 <- 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

}

> bc.rpart3 = rpart(Diagnosis~.,data=BCtrain,cp=.00001,minsplit=5)

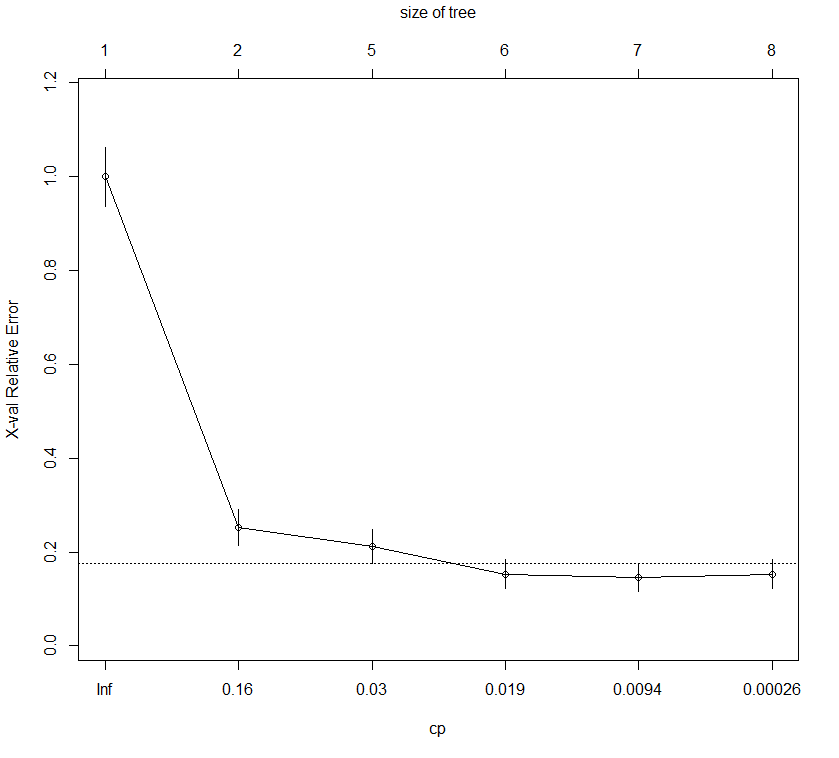
> results = crpart.sscv(bc.rpart3,BCtrain$Diagnosis,data=BCtrain,B=200)

> summary(results)

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

0.03175 0.06349 0.07143 0.07480 0.08730 0.14290

> plotcp(bc.rpart3)



> bc.rpart4 = rpart(Diagnosis~.,data=BCtrain,cp=.019,minsplit=5)

> results = crpart.sscv(bc.rpart4,BCtrain$Diagnosis,data=BCtrain,B=200)

> summary(results)

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

0.01587 0.06349 0.07143 0.07472 0.08730 0.15080

> ypred = predict(bc.rpart4,newdata=BCtest,type="class")

> misclass(ypred,BCtest$Diagnosis)

Table of Misclassification

(row = predicted, col = actual)

y

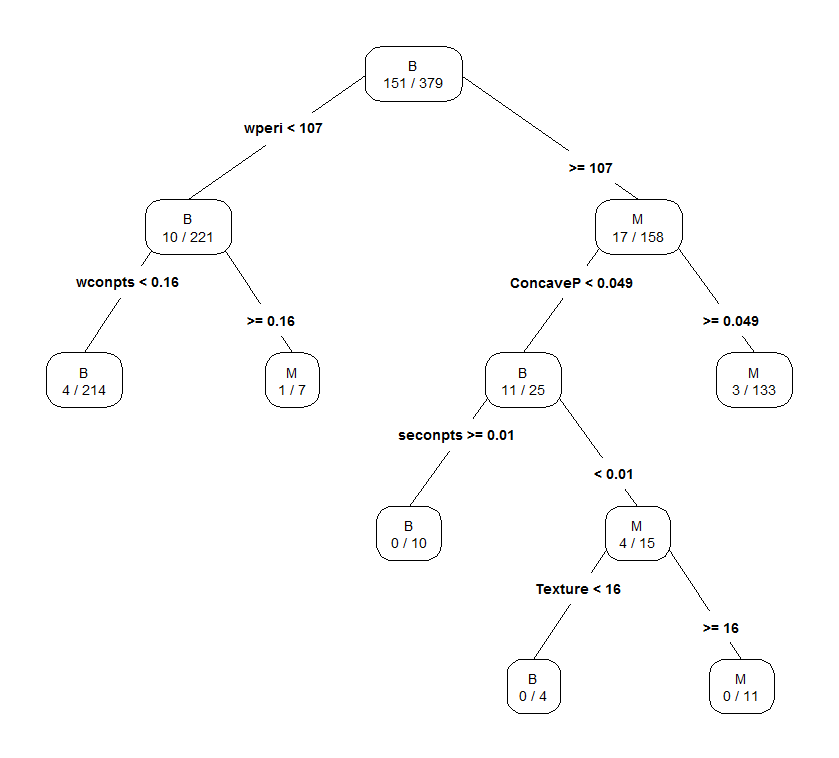
fit B M

B 123 5

M 6 56

Misclassification Rate = 0.0579

This is probably about the best we going to do. Here is a display of the final tree fit to training data.



**Bagging**

Using the same cp and minsplit parameters as a best rpart model we will now use bagging to hopefully improve the predictive performance.

> bc.bag = bagging(Diagnosis~.,data=BCtrain,nbagg=100,coob=T  
control=rpart.control(cp=.019,minsplit=5,xval=0))

> bc.bag

Bagging classification trees with 100 bootstrap replications

Call: bagging.data.frame(formula = Diagnosis ~ ., data = BCtrain, nbagg = 100,

control = rpart.control(cp = 0.019, minsplit = 5, xval = 0),

coob = T)

Out-of-bag estimate of misclassification error: 0.0554

> ypred = predict(bc.bag,newdata=BCtest,type="class")

> misclass(ypred,BCtest$Diagnosis)

Table of Misclassification

(row = predicted, col = actual)

y

fit B M

B 125 6

M 4 55

Misclassification Rate = 0.0526

Slightly better than a single RPART model.

**Random Forest**Using a random forest with the default settings we obtain the following.

> bc.rf = randomForest(Diagnosis~.,data=BCtrain)

> bc.rf

Call:

randomForest(formula = Diagnosis ~ ., data = BCtrain)

Type of random forest: classification

Number of trees: 500

No. of variables tried at each split: 5

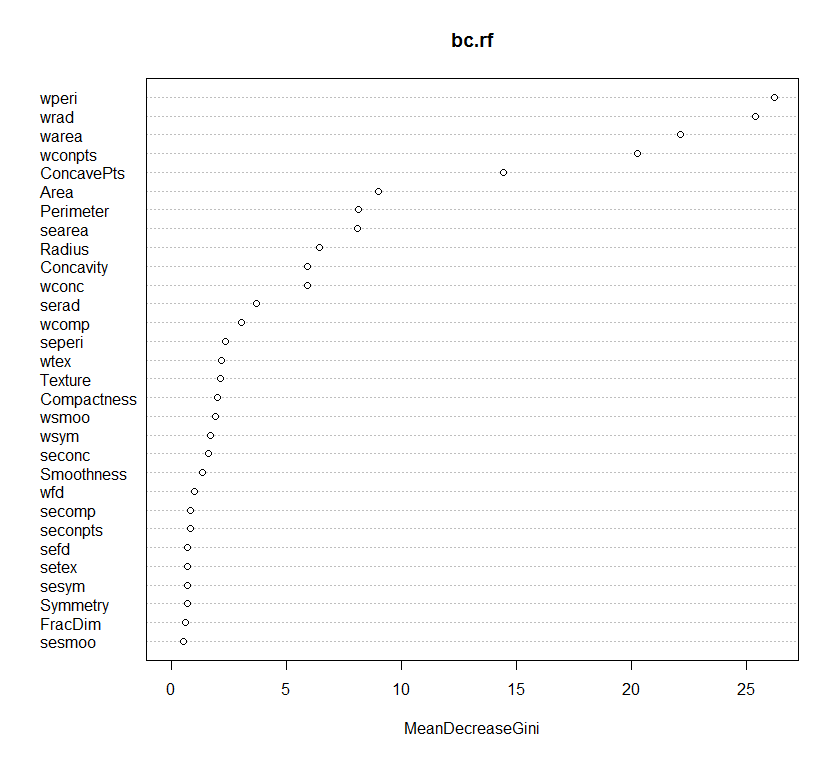
OOB estimate of error rate: 3.69%

Confusion matrix:

B M class.error

B 223 5 0.02192982

M 9 142 0.05960265

> varImpPlot(bc.rf)  


> ypred = predict(bc.rf, newdata=BCtest,type="class")

> misclass(ypred,BCtest$Diagnosis)

Table of Misclassification

(row = predicted, col = actual)

y

fit B M

B 125 3

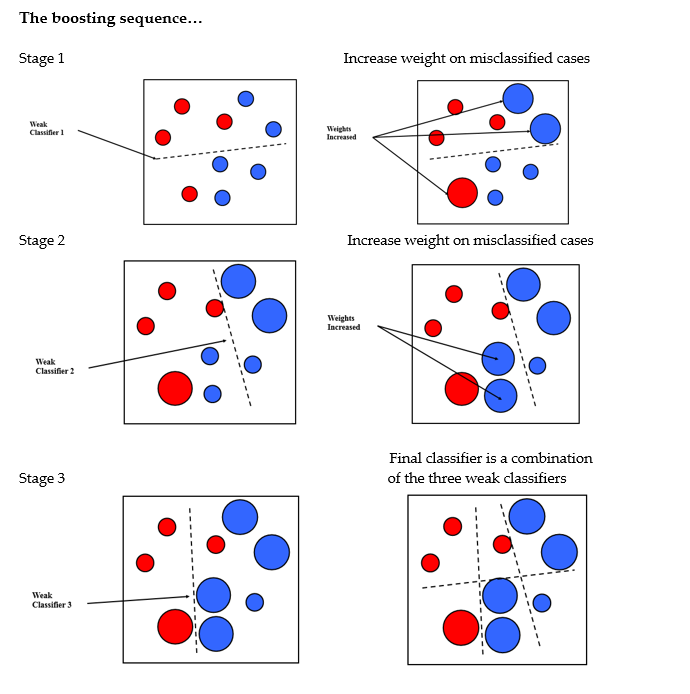
M 4 58

Misclassification Rate = 0.0368

Even with no fine tuning the random forest is superior to the previous models and we also have a nice display of which of the cell sample characteristics are most important.

**Boosting**

For boosting in classification the packages ada and adabag have a better implementation of boosting. Before looking at their use for these data consider the diagram below which illustrates the concept of boosting for a classification problem.



For two level classification problems, like we have here with the breast cancer diagnosis data, the ada package and function of the same name are probably best to use.

> bc.boost = ada(Diagnosis~.,data=BCtrain)

> bc.boost

Call:

ada(Diagnosis ~ ., data = BCtrain)

Loss: exponential Method: discrete Iteration: 50

Final Confusion Matrix for Data:

Final Prediction

True value B M

B 236 0

M 4 139

Train Error: 0.011

Out-Of-Bag Error: 0.016 iteration= 40

Additional Estimates of number of iterations:

train.err1 train.kap1

46 46

> summary(bc.boost)

Call:

ada(Diagnosis ~ ., data = BCtrain)

Loss: exponential Method: discrete Iteration: 50

Training Results

Accuracy: 0.989 Kappa: 0.977

The internal cross-validation results from the boosting fit look very promising, as the OOB error rate is 1.6%.

> ypred = predict(bc.boost,newdata=BCtest)

> attributes(ypred)

$levels

[1] "B" "M"

$class

[1] "factor"

> misclass(ypred,BCtest$Diagnosis)

Table of Misclassification

(row = predicted, col = actual)

y

fit B M

B 118 1

M 3 68

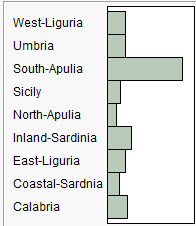
Misclassification Rate = 0.0211

Boosting has the lowest misclassification rate on the test cases, 2.11%.

**Task - Predicting Growing Area for Italian Olive Oils (Datafile: OliveOils.csv)**

The goal here is to classify an olive oil as being from one of nine individual growing areas in Italy (Area Name) – East Liguria, West Liguria, Umbria, North-Apulia, South-Apulia, Sicily, Coastal Sardinia, Inland-Sardinia, and Calabria. The map below should help in your understanding of where these areas are located in Italy.





Puglia = Apulia

Sardegna = Sardinia

Sicilia = Sicily

The bar graph above shows the number of olive oils in these data from each area.

***The fatty acids measured (i.e. the predictors) are as follows:***

Palmitic

Palmitoleic

Strearic

Oleic

Linoleic

Eicosanoic

Linolenic

Reading Data and Forming a Training and Validation/Test Set  
> OliveOils = read.table(file.choose(),header=T,sep=”,”) 🡨 read in **OliveOils.csv**  
> names(OliveOils)[1] "Area.name" "palmitic" "palmitoleic" "strearic" "oleic" "linoleic" [7] "eicosanoic" "linolenic"   
> dim(OliveOils)[1] 572 8> train = sample(1:572,floor(.6666\*572),replace=F)

> Olive.train = OliveOils[train,]

> Olive.test = OliveOils[-train,]

> dim(Olive.train)

[1] 429 8

> dim(Olive.test)

[1] 143 8

**1.** Fit a tuned RPART model to the training data. Predict the growing areas of the olives in the test data set. What is the misclassification rate of your best tree?

**2.** Use bagging to improve your RPART model from the part 1. To predict from your bagged model you have to do the following:

> olive.bag = bagging(Area.name~.,data=Olive.train, *other settings you choose*)

> ypred = predict(olive.bag,newdata=Olive.test,type=”class”)

> attributes(ypred)

$names

[1] "formula" "votes" "prob" "class" "confusion" "error"

> ypred$error

and

> misclass(ypred$class,Olive.test$Area.name)

will both give the error rate for predicting the growing areas of the test cases.

**3.** Grow a random forest for predicting the growing area of the test oils. What fatty acids are   
 most important for classifying olive oils?

4. Use boosting to build a model for classifying olive oils. You will have to use   
 the function boosting in the adabag library to fit a model. Below is the code to fit a base  
 model. The prediction process is identical to that for bagging above.

> olive.boost = boosting(Area.name~.,data=Olive.train,mfinal=100,

control=rpart.control(*change cp, minsplit, etc. here if you want*))

> ypred = predict(olive.boost,newdata=Olive.test)

> attributes(ypred)

$names

[1] "formula" "votes" "prob" "class" "confusion" "error"  
 > misclass(ypred$class,Olive.test$Area.name)

**TASK 2 – Predicting the Edibility of Mushrooms**

**Storytime:**These data were used in the Undergraduate Data Analysis Contest (UDAC) in 2001 which is now defunct. The goal of the problem using these data was to develop a predictive model to classify a mushroom as poisonous or edible on the basis of a set of categorical/nominal characteristics of the mushrooms. Andrea Nibbe, a WSU student, earned 1st place in both aspects of the competition (written summary of her results and the accuracy of her predictions – 100% correct!). She went on to become the first undergraduate to win the prestigious Gertrude Cox Scholarship which is awarded to the best female statistics student in the country.

The mushroom attributes are defined below:

     Poisonous = edibility edible=e,poisonous=p

x1 = cap-shape: bell=b,conical=c,convex=x,flat=f,  
 knobbed=k,sunken=s  
 x2 = cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s  
 x3 = cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r,  
 pink=p,purple=u,red=e,white=w,yellow=y  
 x4 = bruises? bruises=t,no=f  
 x5 = odor: almond=a,anise=l,creosote=c,fishy=y,foul=f,  
 musty=m,none=n,pungent=p,spicy=s  
 x6 = gill-attachment: attached=a,descending=d,free=f,notched=n  
 x7 = gill-spacing: close=c,crowded=w,distant=d  
 x8 = gill-size: broad=b,narrow=n  
 x9 = gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g,  
 green=r,orange=o,pink=p,purple=u,red=e,  
 white=w,yellow=y  
 x10 = stalk-shape: enlarging=e,tapering=t  
 x11 = stalk-root: bulbous=b,club=c,cup=u,equal=e,  
 rhizomorphs=z,rooted=r,missing=? 🡨 LOTS OF MISSING VALUES  
 x12 = stalk-surface-above-ring: ibrous=f,scaly=y,silky=k,smooth=s  
 x13 = stalk-surface-below-ring: ibrous=f,scaly=y,silky=k,smooth=s

x14 = stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,  
 pink=p,red=e,white=w,yellow=y  
 x15 = stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,  
 pink=p,red=e,white=w,yellow=y  
 x16 = veil-type: partial=p,universal=u  
 x17 = veil-color: brown=n,orange=o,white=w,yellow=y  
 x18 = ring-number: none=n,one=o,two=t  
 x19 = ring-type: cobwebby=c,evanescent=e,flaring=f,large=l,  
 none=n,pendant=p,sheathing=s,zone=z  
 x20 = spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r,  
 orange=o,purple=u,white=w,yellow=y  
 x21 = population: abundant=a,clustered=c,numerous=n,  
 scattered=s,several=v,solitary=y  
 x22 = habitat: grasses=g,leaves=l,meadows=m,paths=p,  
 urban=u,waste=w,woods=d

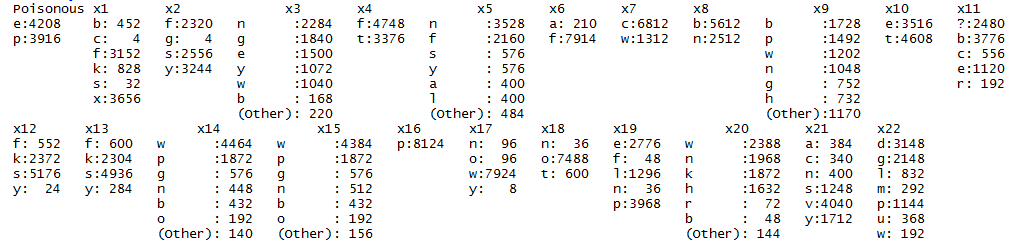
> Mushrooms = read.table(file.choose(),header=T,sep=”,”) 🡨read in **Mushrooms.csv**

> dim(Mushrooms)

[1] 8124 23

> names(Mushrooms)

> summary(Mushrooms)



> str(Mushrooms)

'data.frame': 8124 obs. of 23 variables:

$ Poisonous: Factor w/ 2 levels "e","p": 2 1 1 2 1 1 1 1 2 1 ...

$ x1 : Factor w/ 6 levels "b","c","f","k",..: 6 6 1 6 6 6 1 1 6 1 ...

$ x2 : Factor w/ 4 levels "f","g","s","y": 3 3 3 4 3 4 3 4 4 3 ...

$ x3 : Factor w/ 10 levels "b","c","e","g",..: 5 10 9 9 4 10 9 9 9 10 ...

$ x4 : Factor w/ 2 levels "f","t": 2 2 2 2 1 2 2 2 2 2 ...

$ x5 : Factor w/ 9 levels "a","c","f","l",..: 7 1 4 7 6 1 1 4 7 1 ...

$ x6 : Factor w/ 2 levels "a","f": 2 2 2 2 2 2 2 2 2 2 ...

$ x7 : Factor w/ 2 levels "c","w": 1 1 1 1 2 1 1 1 1 1 ...

$ x8 : Factor w/ 2 levels "b","n": 2 1 1 2 1 1 1 1 2 1 ...

$ x9 : Factor w/ 12 levels "b","e","g","h",..: 5 5 6 6 5 6 3 6 8 3 ...

$ x10 : Factor w/ 2 levels "e","t": 1 1 1 1 2 1 1 1 1 1 ...

$ x11 : Factor w/ 5 levels "?","b","c","e",..: 4 3 3 4 4 3 3 3 4 3 ...

$ x12 : Factor w/ 4 levels "f","k","s","y": 3 3 3 3 3 3 3 3 3 3 ...

$ x13 : Factor w/ 4 levels "f","k","s","y": 3 3 3 3 3 3 3 3 3 3 ...

$ x14 : Factor w/ 9 levels "b","c","e","g",..: 8 8 8 8 8 8 8 8 8 8 ...

$ x15 : Factor w/ 9 levels "b","c","e","g",..: 8 8 8 8 8 8 8 8 8 8 ...

$ x16 : Factor w/ 1 level "p": 1 1 1 1 1 1 1 1 1 1 ... 🡨 ONLY HAS ONE LEVEL!

$ x17 : Factor w/ 4 levels "n","o","w","y": 3 3 3 3 3 3 3 3 3 3 ...

$ x18 : Factor w/ 3 levels "n","o","t": 2 2 2 2 2 2 2 2 2 2 ...

$ x19 : Factor w/ 5 levels "e","f","l","n",..: 5 5 5 5 1 5 5 5 5 5 ...

$ x20 : Factor w/ 9 levels "b","h","k","n",..: 3 4 4 3 4 3 3 4 3 3 ...

$ x21 : Factor w/ 6 levels "a","c","n","s",..: 4 3 3 4 1 3 3 4 5 4 ...

$ x22 : Factor w/ 7 levels "d","g","l","m",..: 6 2 4 6 2 2 4 4 2 4 ...

> Mushrooms2 = Mushrooms[,-c(12,17)]

> train = sample(1:nrow(Mushrooms2),floor(nrow(Mushrooms\*0.5)),replace=F)

> Mush.train = Mushrooms2[train,]

> Mush.test = Mushrooms2[-train,]

> mush.tree = rpart(Poisonous~.,data=Mush.train)

> summary(mush.tree)

> misclass(Mush.train$Poisonous,predict(mush.tree,type="class"))

Table of Misclassification

(row = predicted, col = actual)

y

fit e p

e 4208 0

p 48 3868

Misclassification Rate = 0.00591

> ypred = predict(mush.tree,newdata=Mush.test,type=”class”)

> misclass(Mush.test$Poisonous,ypred)

Your task is build a better model for classifying edibility of mushrooms.