1 Tree

We use \texttt{rpart} package to fit tree model. On the training data set, the default model is presented in Figure 1. Notice that the structure of the tree could be very different with different splits of the data set.

\begin{verbatim}
rm(list=ls(all=TRUE))
spamdata <- read.csv("spam.csv")
N <- nrow(spamdata)
shuffled <- sample(1:N, N)
test.ind <- shuffled[1:ceiling(N*.2)]
train.ind <- shuffled[-(1:ceiling(N*.2))]
spamdata <- spamdata[,c(1,2,7,19,20)]
spamdata.test <- spamdata[test.ind,]
spamdata.train <- spamdata[train.ind,]

library(rpart)

rpart1 <- rpart(spam ~ ., data=spamdata.train)
par(mar=c(1.5,3,1,3))
plot(rpart1,uniform=T)
text(rpart1,use.n=T,pretty=T,cex=.6)
\end{verbatim}

The interpretation of the tree can be confusing, so we want to talk about how to interpret the output from \texttt{rpart} in more details. It helps a lot to look at the numerical output of the tree.

\begin{verbatim}
> print(rpart1)
n= 1736
node), split, n, loss, yval, (yprob)
 * denotes terminal node

1) root 1736 575 no (0.66877880 0.33122120)
   2) local=yes 757 9 no (0.98811096 0.01188904) *
   3) local=no 979 413 yes (0.42185904 0.57814096)
      6) large.text=no 648 280 no (0.56790123 0.43209877)
      12) sucker=no 592 229 no (0.61317568 0.38682432)
\end{verbatim}
For each line, the first item (split) is the split rule, the second item (n) is the number of cases at that node, e.g., there are 757 cases (in our training set) that have local=yes. The third item (loss) is the number of misclassified cases. The fourth item (yval) is the classification decision corresponding to that node, simply based on the proportion of yes and no in that node. This arrangement is quite counter-intuitive, since we need to first know the classification decision to calculate the loss. The last item gives us the probability of no and yes for that node. So to find the terminal nodes that correspond to the highest and lowest probability (the highest and lowest must be terminal nodes), it is most convenient to look at this item. We can see that the lowest probability of being spam (.01189) is local=yes, while the highest probability of being spam (1.0) is local=no, large.text=no, sucker=no, digits< 0.5 and porn=yes.

As for reading the figure, bear in mind that the left branch always corresponds to when the logical expression equals to TRUE, and the right branch FALSE. The labels of the terminal nodes are yval, i.e., the decision for that node, and the two numbers are the number of no and yes (notice whenever the number of no is larger than yes, the label is no, and vice versa.)

To get the cross-validation error, we need to make predictions using predict function on test data. The cross-validation accuracy is actually quite high, around 80 – 90%.

```r
preds<-predict(rpart1,newdata=spamdata.test)
pred.ind<-(preds[,2]>=.5)
table(pred.ind, spamdata.test$spam)
accuracy <- (sum(pred.ind==1&spamdata.test$spam=='yes')+ sum(pred.ind==0&spamdata.test$spam=='no'))/sum(table(spamdata.test$spam,pred.ind))
```

We prune the tree by adjusting the CP (Complexity Parameter) of our model. Note that in the implementation of rpart, different CP levels have already been computed, all we need to do is just to extract the information by using printcp function. It returns a summary of different CP levels with corresponding size of the trees. Then we use prune function with appropriate CP levels to generate new trees. It is important to do these on the same training/test split and report the tree with the smallest cross-validation errors.

```r
printcp(rpart1)
rpart2 <- prune(rpart1, cp=.012)
rpart3 <-
rpart4 <-
rpart5 <-
```
Figure 1: Tree of default size
2 Cluster

Cluster dendrograms by artist and type are in Figures 2 and 3, respectively. First, standardizing the variables is again essential since our distance measure is sensitive to scale. Along with choosing a distance measure, which we discussed in class, we also need to decide which algorithm to use for making the clusters. In our assignment we consider three options. These three differ in how they define the distance between two possible clusters. We then use these distances to combine sets of points into progressively larger clusters. This type of algorithm is called agglomerative clustering. Single-link clustering defines the distance between two clusters as the dissimilarity of the closest pair and then adds the shortest possible link (thus joining the two closest clusters). Complete-link, in contrast, defines the distance between clusters as the maximum of the dissimilarities between members. Wards method defines the distance between two clusters as how much the sums of squares will increase if we merge them. With hierarchical clustering, we start with zero sums of squares (since each point is in its own cluster) and then sums of squares grows as we merge clusters. Wards method keeps this growth as small as possible.

```r
music<-read.csv("music-full.csv")
music.numeric<-music[,,-c(1:4)]
#standardize
for(i in 1:ncol(music.numeric)){
  music.numeric[,i]<-(music.numeric[,i]-mean(music.numeric[,i]))/sd(music.numeric[,i])
}
par(mfrow=c(3,1))
plot(hclust(dist(music.numeric),method='single'),labels=music$Artist,
     main="Clustering method-Single",cex.main=.8,cex=.8)
plot(hclust(dist(music.numeric),method='complete'),labels=music$Artist,
     main="Clustering method-Complete",cex.main=.8,cex=.8)
plot(hclust(dist(music.numeric),method='ward'),labels=music$Artist,
     main="Clustering method-Ward",cex.main=.8,cex=.8)
par(mfrow=c(3,1))
plot(hclust(dist(music.numeric),method='single'),labels=music$Type,
     main="Clustering method-Single",cex.main=.8,cex=.8)
plot(hclust(dist(music.numeric),method='complete'),labels=music$Type,
     main="Clustering method-Complete",cex.main=.8,cex=.8)
plot(hclust(dist(music.numeric),method='ward'),labels=music$Type,
     main="Clustering method-Ward",cex.main=.8,cex=.8)
```

There are several ways to measure how well the clustering works. One possible way to do this is to use the dendrogram to classify the songs into 7 clusters (for artists) or 3 clusters (for types), excluding the unknowns. Then we label the clusters by the most common category in each cluster. Having done that, our problem turns into a typical classification problem, and we can measure the model by calculating (mis)classification rate.
Figure 2: Cluster by Artists
Figure 3: Cluster by Types