Business Data Mining (IDS 572)

Homework 3-Solution

Question 1

(a) 39% of emails are actually spam

```
> sum(spam$spam == “spam”)
[1] 1813
> sum(spam$spam == “spam”)/nrow(spam)
[1] 0.3940448
```

(b) “email” since most letters are not spma.

(c) The error rate is 39%

```
> sum(spam$spam != “email”)/nrow(spam)
[1] 0.3940448
```

Question 2

```
> training.rows = sample(1:nrow(spam), 2301, replace=FALSE)
> training.data = spam[training.rows, ]
> testing.data = spam[-training.rows, ]
> dim(training.data)
[1] 2301 58
> dim(testing.data)
[1] 2300 58
> intersect(rownames(training.data), rownames(testing.data))
character(0)
> sum(training.data$spam=="spam")/2301
[1] 0.4059105
> sum(testing.data$spam=="spam")/2300
[1] 0.3821739
```

In other words, the training data is 41% spam, and the testing data is 38%. This is not statistically significant.

Question 3

```
spam.tr = tree(spam ~., data=training.data)
> summary(spam.tr)
```

```
Classification tree:
tree(formula = spam ~ ., data = training.data)
Variables actually used in tree construction:
[1] "A.53" "A.7"  "A.52" "A.25" "A.5"  "A.56" "A.55" "A.16" "A.27"
Number of terminal nodes:  12
Residual mean deviance:  0.4665 = 1068 / 2289
Misclassification error rate: 0.0804 = 185 / 2301
```

(Notice that summary tells us which variables appear in the tree.)
Using . on the right-hand side of the formula means “include every variable from the data frame other than the response”.

Using cross-validation to prune, and plotting error versus size:

```
> spam.tr.cv = cv.tree(spam.tr, method="misclass")
> plot(spam.tr.cv)
```

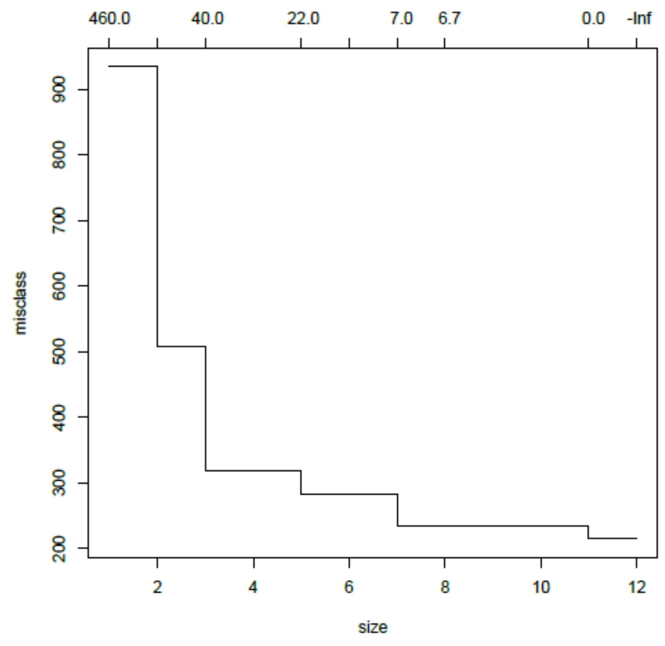


Figure 1: Mis-classification rate versus tree size for prunings of the default tree. (The upper horizontal axis shows the values of the error/size trade-off parameter which give us a tree with a given number of nodes.)

The best tree is, in this case, the largest tree found. In part this is because the default tree-fitting algorithm includes some sensible stopping rules.

```
>plot(spam.tr)
```

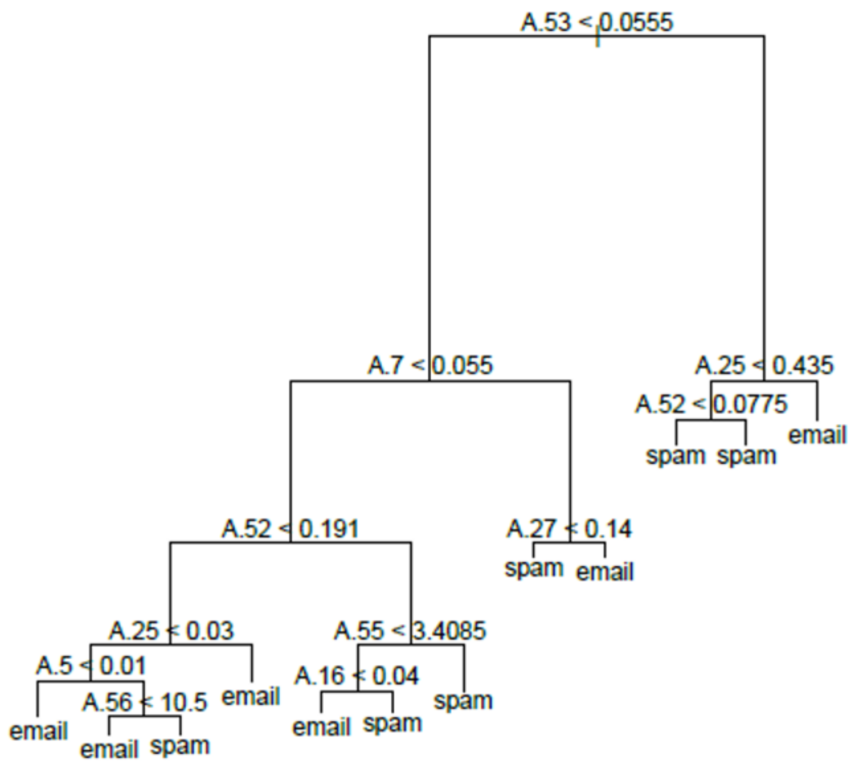


Figure 2: The default tree.

The error rate on the testing data:

```
> spam.tr.predctions = predict(spam.tr,newdata=testing.data,type="class")
> sum(spam.tr.predictions != testing.data$spam)/nrow(testing.data)
[1] 0.1034783
```

This rate is 10%; the difference between this and the 8% rate on the training data is statistically significant, but not substantively huge.

Things look a bit different if we start by fitting a very big tree.

```
> spam.tr.big = tree(spam ~., data=training.data,minsize=1, mindev=0)
> summary(spam.tr.big)
```

```
Classification tree:
tree(formula = spam ~ ., data = training.data, minsize = 1, mindev = 0)
Variables actually used in tree construction:
 [1] "A.53" "A.7"  "A.52" "A.25" "A.5"  "A.27" "A.45" "A.56" "A.55" "A.11"
 "A.20" "A.3"
[13] "A.19" "A.57" "A.46" "A.16" "A.12" "A.49" "A.21" "A.22" "A.29" "A.14"
 "A.28" "A.8"
[25] "A.9"  "A.15" "A.24" "A.1"  "A.44" "A.10" "A.13" "A.33" "A.39" "A.2"
 "A.18" "A.37"
Number of terminal nodes:  125
Residual mean deviance:  0.001274 = 2.773 / 2176
Misclassification error rate: 0.0004346 = 1 / 2301
```

“minsize” is the minimum number of observations to allow in a node, and “mindev” is the minimum improvement in the error needed to create a node. This tells the tree-growing algorithm to ignore such limits.

To get the best size, we look for the point where the error rate bottoms out. Unfortunately, cv.tree lists sizes in decreasing order, and which.min returns the first match. Use rev to reverse vectors.

```
> rev(spam.tr.big.cv$size)[which.min(rev(spam.tr.big.cv$dev))]
[1] 19
```

(Why do we need to use rev twice?)

```
> spam.tr.pruned = prune.tree(spam.tr.big, best=19)
> summary(spam.tr.pruned)
```

```
Classification tree:
snip.tree(tree = spam.tr.big, nodes = c(27, 129, 14, 10, 37,
135, 66, 17, 49, 19, 26, 48, 25, 36, 128, 134))
Variables actually used in tree construction:
 [1] "A.53" "A.7"  "A.52" "A.25" "A.5"  "A.27" "A.45" "A.56" "A.46" "A.55"
 "A.16"
Number of terminal nodes:  19
Residual mean deviance:  0.3911 = 892.6 / 2282
Misclassification error rate: 0.06389 = 147 / 2301
```

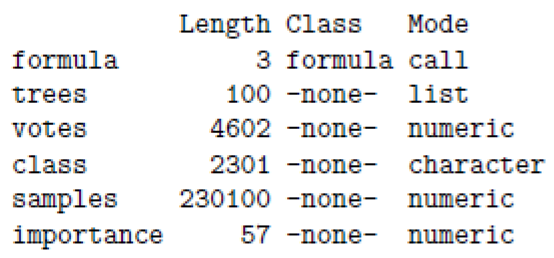
Much smaller than the full tree, this can actually be plotted and labeled.

The error rate on the testing data (9.1%) is a bit better than that of the slightly smaller tree we got at first, but not hugely:

```
> spam.pruned.preds = predict(spam.tr.pruned,newdata=testing.data,type="class")
> sum(spam.pruned.preds != testing.data$spam)/2300
[1] 0.09130435
```

Whether the extra 1% of error is worth having somewhat fewer leaves is up to you.

```
(b) >spam.bag = bagging(spam ~ ., data = training.data, minsplit=1, cp = 0)
> summary(spam.bag)
```



```
           Length Class   Mode
formula         3 formula call
trees         100 -none-  list
votes        4602 -none-  numeric
class        2301 -none-  character
samples    230100 -none-  numeric
importance     57 -none-  numeric
```

With bagging, we'll not worry about individual trees being too over-fit; rather we'll let the averaging take care of that for us. (In fact, here if I use the default control settings bagging sometimes returns the same tree on every bootstrap sample!) The summary method, evidently, is not too informative. In part however this is because there are 100 separate trees to keep track of!

The in-sample and out-of-sample error rates are 5.9% and 7.6%:

```
> sum(spam.bag$class != training.data$spam)/nrow(training.data)
[1] 0.05910474
> predict(spam.bag,newdata=testing.data)$error
[1] 0.07565217
```

(c) They all do

Question 4

The bagged trees with 7.6% misclassification is the model with min error rate.

(a) >testing.spam.rows = (testing.data$spam == “spam”)

```
> predict(spam.bag, newdata=testing.data[testing.spam.rows,])$error
[1] 0.1467577
```

That is, first we check which rows of the testing data have the true class of spam. (Note that testing.spam.rows is a Boolean vector, of length 2300.) Then we then predict only on those data points; all the errors then are false negatives. This gives an error rate of 14.7%.

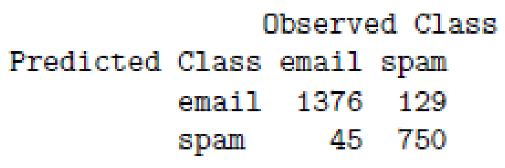
(b) We can re-use the same trick:

```
> predict(spam.bag, newdata=testing.data[!testing.spam.rows,])$error
[1] 0.03166784
```

Notice that we need to logically negate testing.spam.rows — we want the rows which aren’t spam. This gives an error rate of 3.2%.
Quick sanity check: the two error rates together should add up to the over-all error rate. And, indeed, 0.147*0.38+0.032*(1−0.38) = 0.076, as it should.

In this case, the predict method for bagging not only returns the error rate, it will actually return the confusion matrix, the contingency table of actual versus observed classes:

```
> predict(spam.bag, newdata=testing.data)$confusion
```



```
               Observed Class
Predicted Class email spam
          email  1376  129
          spam     45  750
```

From this we can calculate that the false positive rate is 45/(45 + 1376) = 3.2%, and the false negative rate is 129/(129+750) = 14.7%, as we saw above.

(c) From the confusion matrix, 750/(750+ 45) = 94%. Without the confusion matrix, we use Bayes' rule. (To save space, below +1 stands for the class spam, and −1 for the class email.)

$$P\left(Y = +1\middle|\hat{Y} = +1\right)$$

$$= \frac{P\left(\hat{Y} = +1\middle|Y = +1\right)P(Y = +1)}{P\left(\hat{Y} = +1\middle|Y = +1\right)P(Y = +1) + P\left(\hat{Y} = +1\middle|Y = -1\right)P(Y = -1)}$$

$$= \frac{(1 - 0.147) \times 0.38}{(1 - 0.147) \times 0.38 + 0.032 \times (1 - 0.38)} = 0.94$$