PSTAT 131 Homework Two

Lilian Lu and Ted Tinker

10/24/2017

1 Selecting K for K Nearest Neighbors

After loading the dataset and making folds as directed in the preliminary notes, we define a vector of options for K, split *spam.test* into ten folds, and produce a *do.chunk*() function.

```
kvec = c(1, seq(10, 50, length.out=5))

do.chunk <- function(chunkid, folddef, Xdat, Ydat, k){
  train = (folddef!=chunkid)
  Xtr = Xdat[train,]
  Ytr = Ydat[train]
  Xvl = Xdat[!train,]
  Yvl = Ydat[!train]
  ## get classifications for current training chunks
  predYtr = knn(train = Xtr, test = Xtr, cl = Ytr, k = k)
  ## get classifications for current test chunk
  predYvl = knn(train = Xtr, test = Xvl, cl = Ytr, k = k)
  return(data.frame(train.error = calc_error_rate(predYtr, Ytr),
                    val.error = calc_error_rate(predYvl, Yvl)))
}
```

Now we perform cross-validation and print the test and training errors for each value of K.

```
set.seed(1) #the randomness here affects our lowest test error
zeroes = c(0,0,0,0,0,0)
temp <- data.frame(TrainError = zeroes, TestError=zeroes)   # Make 2*6 0 matrix for storage
a <- 1

for(k in kvec) {
  lapply(1:10, function(z) {temp[a,] <<- temp[a,] + .1*
        do.chunk(z,folds,spam.train[,-58],spam.train$y,k)})
        # This runs veeerry slowly. I used the <<- operator to assign outside the scope.
  a = a + 1}
print(temp)
```

```
##   TrainError TestError
## 1  0.0003394    0.1014
## 2  0.0827547    0.1005
## 3  0.0946650    0.1055
## 4  0.1031194    0.1150
## 5  0.1132092    0.1227
## 6  0.1181460    0.1241
```

$k = 10$ seems to yield the lowest test error.

2 Training and Test Errors

Using the value of k chosen in part 1, we calculate the true test error:

```
best.kfold=10
YPred = knn(train = spam.train[,-58], test = spam.train[,-58], cl = spam.train$y, k = best.kfold)
YTest=knn(train = spam.train[,-58],test=spam.test[,-58],cl=spam.train$y,k=best.kfold)
records[1,] <- c(calc_error_rate(YPred,spam.train$y),calc_error_rate(YTest,spam.test$y))
print(records)
```

```
##          train.error test.error
## knn          0.08053      0.099
## tree              NA         NA
## logistic          NA         NA
```

3 Controlling Decision Tree Construction

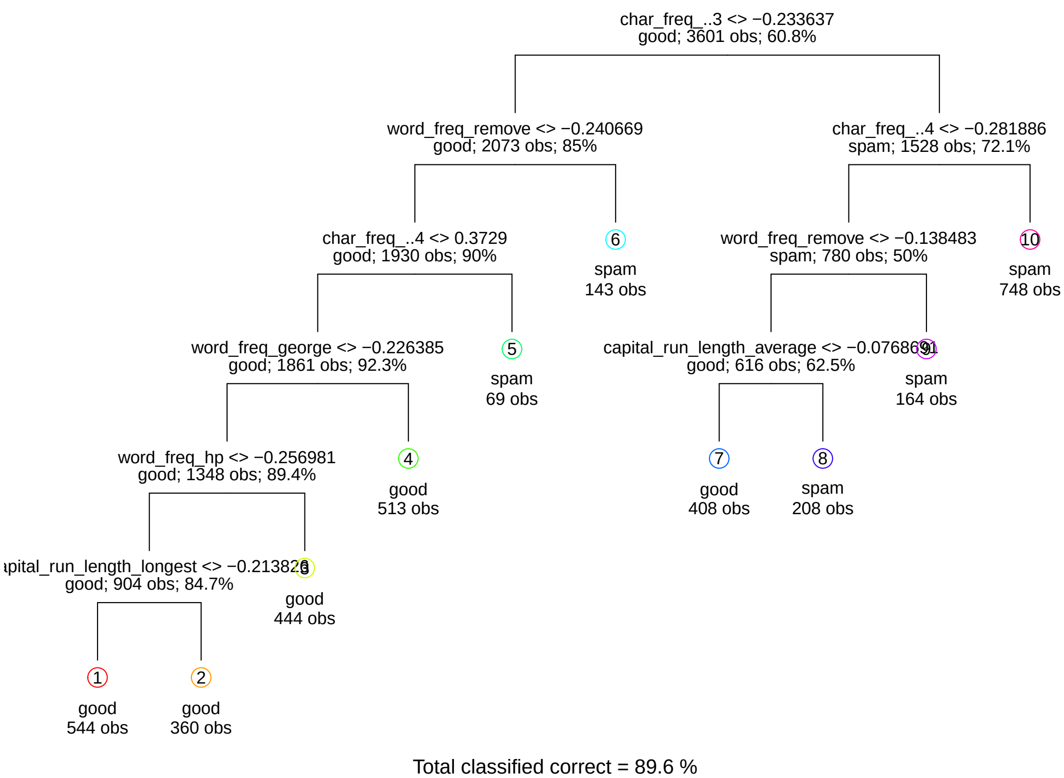
```
spamtree <- tree(y~.,spam.train,control=tree.control(nrow(spam.train),
                    mincut=1,minsize=5,mindev=1e-5),method="recursive.partition")
summary(spamtree)
```

```
##
## Classification tree:
## tree(formula = y ~ ., data = spam.train, control = tree.control(nrow(spam.train),
##     mincut = 1, minsize = 5, mindev = 1e-05), method = "recursive.partition")
## Variables actually used in tree construction:
##  [1] "char_freq_..3"              "word_freq_remove"
##  [3] "char_freq_..4"              "word_freq_george"
##  [5] "word_freq_hp"               "capital_run_length_longest"
##  [7] "word_freq_receive"          "word_freq_free"
##  [9] "word_freq_direct"           "capital_run_length_average"
## [11] "word_freq_re"               "word_freq_you"
## [13] "capital_run_length_total"   "word_freq_credit"
## [15] "word_freq_our"              "word_freq_will"
## [17] "char_freq_..1"              "word_freq_your"
## [19] "word_freq_meeting"          "word_freq_1999"
## [21] "word_freq_make"             "word_freq_hpl"
## [23] "word_freq_order"            "word_freq_telnet"
## [25] "word_freq_mail"             "word_freq_font"
## [27] "word_freq_report"           "word_freq_money"
## [29] "word_freq_address"          "word_freq_data"
## [31] "word_freq_000"              "word_freq_all"
## [33] "word_freq_project"          "word_freq_labs"
## [35] "word_freq_people"           "word_freq_email"
## [37] "word_freq_415"              "word_freq_edu"
## [39] "word_freq_technology"       "word_freq_business"
## [41] "char_freq_..2"              "word_freq_over"
## [43] "word_freq_internet"         "char_freq_..5"
## Number of terminal nodes:  184
## Residual mean deviance:  0.0219 = 74.9 / 3420
## Misclassification error rate: 0.00528 = 19 / 3601
```

The tree generated has 184 terminal nodes, or leaves. Of 3601 observations, only 19 are misclassified.

4 Decision Tree Pruning

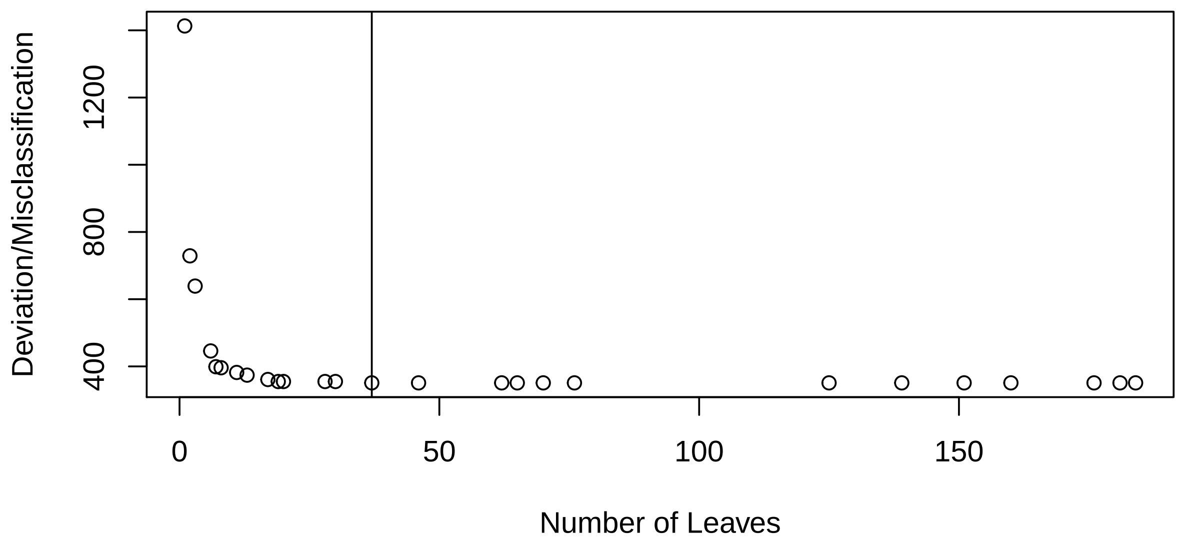
```r
pruned <- prune.tree(spamtree,best=10)     # Prune to 10 leaves
draw.tree(pruned,nodeinfo=TRUE)            # Draw
```



I notice the pair of leaves on the far left are both good. That split branch hardly seems necessary, but the leftmost leaf is slightly purer in content.

5 Pruning Part Two

```
Prunedtr<-cv.tree(spamtree,rand=folds,K=10,method="misclass")        # Make trees
plot(Prunedtr$size,Prunedtr$dev,xlab="Number of Leaves",ylab="Deviation/Misclassification")
              # Plot leaf data
best.size.cv<-min(Prunedtr$size[which(Prunedtr$dev==min(Prunedtr$dev))])
abline(v=best.size.cv) # Add verticle line at best size
```



The most effective tree-size in this case is 37 leaves, which is the smallest tree minimizing the misclassification error.

6a Training and Test Errors

```
spamtree.pruned = prune.tree(spamtree,best=best.size.cv)         # Make tree with 37 leaves
YPred = predict(spamtree.pruned, spam.train, type="class")       # Predict training values
YTest = predict(spamtree.pruned, spam.test,type = "class")       # Predict test values
records[2,] <- c(calc_error_rate(YPred,spam.train$y),calc_error_rate(YTest,spam.test$y))
print(records)
```

```
##          train.error test.error
## knn          0.08053      0.099
## tree         0.05554      0.076
## logistic          NA         NA
```

6b Show the Inverse of the Logistic Function is the Logit Function

To show that $\frac{e^z}{1+e^z}$ is the inverse function of $ln\left(\frac{p}{1-p}\right)$, and vice versa, consider the composition of functions

$$ln\left(\frac{\frac{e^z}{1+e^z}}{1-\frac{e^z}{1+e^z}}\right)$$

By the rules for logorithms of fractions, this composition is equal to

$$ln\left(\frac{e^z}{1+e^z}\right) - ln\left(\frac{1}{1+e^z}\right)$$

Continuting to expand, we find

$$ln\left(e^z\right) - ln\left(1+e^z\right) + ln\left(1+e^z\right) = ln\left(e^z\right) = z.$$

Composing the functions returns the argument z to its original state, so they must be inverses of one-another.

7 Logistic Regression

```
logpre<-glm(y~., data=spam.train, family=binomial)    # Use GLM to model the training data binomially
summary(logpre)
```

```
##
## Call:
## glm(formula = y ~ ., family = binomial, data = spam.train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -4.155  -0.211   0.000   0.120   5.301
##
## Coefficients:
##                            Estimate Std. Error z value Pr(>|z|)
## (Intercept)                -12.8443     2.1072   -6.10  1.1e-09 ***
## word_freq_make              -0.1346     0.0771   -1.75  0.08092 .
## word_freq_address           -0.2027     0.1082   -1.87  0.06115 .
## word_freq_all                0.0596     0.0601    0.99  0.32186
## word_freq_3d                 2.8356     2.2529    1.26  0.20817
## word_freq_our                0.3761     0.0785    4.79  1.6e-06 ***
## word_freq_over               0.2370     0.0753    3.15  0.00164 **
## word_freq_remove             0.9248     0.1452    6.37  1.9e-10 ***
## word_freq_internet           0.2144     0.0776    2.76  0.00570 **
## word_freq_order              0.1288     0.0859    1.50  0.13362
## word_freq_mail               0.0381     0.0474    0.80  0.42158
## word_freq_receive           -0.0385     0.0642   -0.60  0.54869
## word_freq_will              -0.1060     0.0719   -1.47  0.14027
## word_freq_people            -0.0585     0.0770   -0.76  0.44748
## word_freq_report             0.0463     0.0468    0.99  0.32200
## word_freq_addresses          0.4102     0.2346    1.75  0.08038 .
## word_freq_free               0.7808     0.1278    6.11  1.0e-09 ***
## word_freq_business           0.3877     0.1100    3.52  0.00043 ***
```

```
## word_freq_email              0.0635     0.0714    0.89  0.37372
## word_freq_you                0.1602     0.0722    2.22  0.02656 *
## word_freq_credit             0.4368     0.2606    1.68  0.09376 .
## word_freq_your               0.2571     0.0696    3.70  0.00022 ***
## word_freq_font               0.2530     0.2033    1.24  0.21334
## word_freq_000                0.7131     0.1636    4.36  1.3e-05 ***
## word_freq_money              0.2953     0.1183    2.50  0.01257 *
## word_freq_hp                -3.0578     0.5665   -5.40  6.8e-08 ***
## word_freq_hpl               -0.9761     0.4439   -2.20  0.02790 *
## word_freq_george           -37.6621     7.7393   -4.87  1.1e-06 ***
## word_freq_650                0.4055     0.1615    2.51  0.01205 *
## word_freq_lab               -1.5008     1.0499   -1.43  0.15285
## word_freq_labs              -0.1303     0.1489   -0.87  0.38177
## word_freq_telnet            -0.0622     0.1733   -0.36  0.71966
## word_freq_857                0.4322     1.3110    0.33  0.74165
## word_freq_data              -0.5114     0.2088   -2.45  0.01433 *
## word_freq_415               -4.0570     1.3463   -3.01  0.00258 **
## word_freq_85                -1.0813     0.4428   -2.44  0.01460 *
## word_freq_technology         0.3060     0.1436    2.13  0.03310 *
## word_freq_1999              -0.0376     0.0872   -0.43  0.66636
## word_freq_parts             -0.1335     0.1077   -1.24  0.21516
## word_freq_pm                -0.2636     0.1928   -1.37  0.17154
## word_freq_direct            -0.1198     0.1391   -0.86  0.38909
## word_freq_cs               -17.9199     8.9627   -2.00  0.04557 *
## word_freq_meeting           -2.8709     1.0952   -2.62  0.00876 **
## word_freq_original          -0.1897     0.1695   -1.12  0.26297
## word_freq_project           -0.8416     0.3319   -2.54  0.01123 *
## word_freq_re                -0.8851     0.1790   -4.94  7.7e-07 ***
## word_freq_edu               -1.1637     0.2721   -4.28  1.9e-05 ***
## word_freq_table             -0.1418     0.1174   -1.21  0.22697
## word_freq_conference        -1.7300     0.7773   -2.23  0.02603 *
## char_freq_.                 -0.3756     0.1342   -2.80  0.00512 **
## char_freq_..1               -0.1472     0.1015   -1.45  0.14722
## char_freq_..2               -0.0345     0.0787   -0.44  0.66119
## char_freq_..3                0.2178     0.0554    3.93  8.5e-05 ***
## char_freq_..4                1.3849     0.1977    7.01  2.5e-12 ***
## char_freq_..5                1.1893     0.5071    2.35  0.01902 *
## capital_run_length_average   0.5703     0.6477    0.88  0.37854
## capital_run_length_longest   1.2936     0.5236    2.47  0.01348 *
## capital_run_length_total     0.8224     0.1710    4.81  1.5e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4823.9  on 3600  degrees of freedom
## Residual deviance: 1435.1  on 3543  degrees of freedom
## AIC: 1551
##
## Number of Fisher Scoring iterations: 13
```

Here is the summary of our logistic function. Not all the predictors are significant for our logistic model.

```r
prob.training = predict(logpre,spam.train,type="response")        # Predict training values
prob.test=predict(logpre,spam.test, type="response")              # Predict test values
```

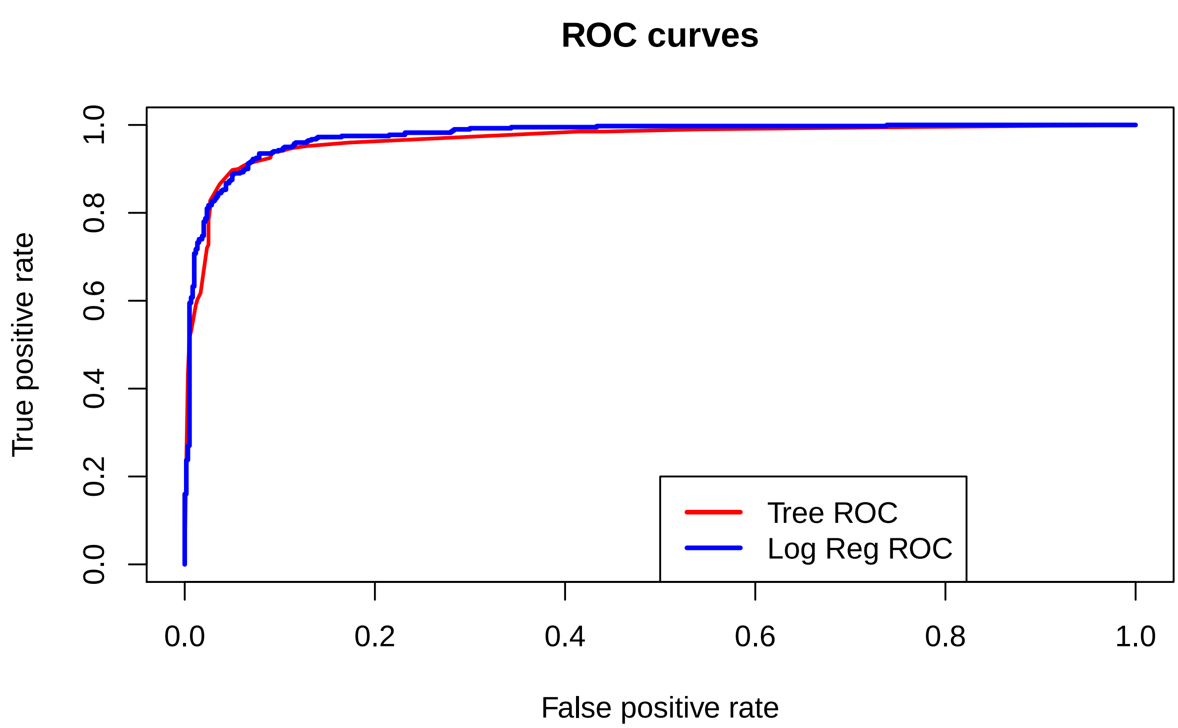
```r
predtrain=as.factor(ifelse(prob.training<=0.5, "good", "spam"))   # Classify as good or spam
predtest=as.factor(ifelse(prob.test<=0.5,"good","spam"))          # at a threshold of 50%
records[3,] <- c(calc_error_rate(predtrain,spam.train$y),calc_error_rate(predtest,spam.test$y))
print(records)
```

```
##          train.error test.error
## knn          0.08053      0.099
## tree         0.05554      0.076
## logistic     0.07081      0.081
```

After testing all three methods, it turns out that the decision tree produces the lowest test error.

8 Receiver Operating Characteristic Curve

```r
pred.prune = predict(spamtree.pruned, spam.test, type="vector") # Predict test values with tree
prob.test=predict(logpre,spam.test, type="response")            # Predict test valyes with logistic reg
pred1<-prediction(pred.prune[,2],spam.test$y)                   # Test Predictions
pred2<-prediction(prob.test,as.numeric(spam.test$y))
perftree= performance(pred1, measure="tpr", x.measure="fpr")    # Measaure performance of tree
perflog=performance(pred2,measure = "tpr",x.measure = "fpr")    # Measure performance of logreg
plot(perftree, col="red", lwd=2, main="ROC curves")
plot(perflog,col="blue",lwd=2.5,add=TRUE)                       # Plot FPR vs TPR
legend(.5,.2, c("Tree ROC","Log Reg ROC"), lty=c(1,1),lwd=c(2.5,2.5),col=c("red","blue"))
```



The red line represents the decision tree model; the blue line represents the logistic regression model.

```r
auctree = performance(pred1, "auc")@y.values
auclog=performance(pred2,"auc")@y.values
print(auctree)
```

```
## [[1]]
## [1] 0.9673
```

```r
print(auclog)
```

```
## [[1]]
## [1] 0.9759
```

Since the AUC of logistic regression is larger than the AUC of the decision tree, we consider the performance of logistic regression to be better.

9 False Positives VS True Positives

Regarding spam, I am most worried about false positives, meaning emails are marked as spam when they are actually important. If the false positive rate is too high, an important memo might fly over my head. With a low true positive rate, I'll have to sort spam from my emails by hand, which is just an inconvenience rather than a career-ending mistake.