Pstat 131 Homework 2

Kendall Brown 8564403

Winter 2018

1.Determining best value of k.

```
kvec = c(1, seq(10, 50, length.out=9))
error.mat=matrix(c(rep(0,20)),nrow=10,ncol=2)
colnames(error.mat)=c("Training Error","Test Error")
rownames(error.mat)=kvec
for(j in 1:10){
  error=ldply(1:9,do.chunk,folds,select(spam.train,-y),spam.train$y,k=kvec[j])
  #mean of the 9 training errors at the value of k
  error.mat[j,1]=mean(error$train.error)
  #mean of the 9 test error at the value of k
  error.mat[j,2]=mean(error$val.error)
}
```

```
#Matrix of mean training and test errors at each value of k
error.mat
```

```
##    Training Error Test Error
## 1    0.0005206923 0.08885841
## 10   0.0806026871 0.09497437
## 15   0.0857748881 0.09747229
## 20   0.0915718721 0.10191050
## 25   0.0973340688 0.10635564
## 30   0.0988962108 0.10746744
## 35   0.1038947573 0.10913688
## 40   0.1073660032 0.11357994
## 45   0.1095529717 0.11441119
## 50   0.1108026463 0.11468828
```

```
#Test errors of each level of k
error.mat[,2]
```

```
##          1         10         15         20         25         30
## 0.08885841 0.09497437 0.09747229 0.10191050 0.10635564 0.10746744
##         35         40         45         50
## 0.10913688 0.11357994 0.11441119 0.11468828
```

```
#Min test error observed
min(error.mat[,2])
```

```
## [1] 0.08885841
```

```
#k value that gives minimum test error
best.kfold=kvec[match(min(error.mat[,2]),error.mat[,2])]
best.kfold
```

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

2.Computing training and test error of the knn method

```
knn.train.train=knn(train = spam.train%>%select(-y),
                    test = spam.train%>%select(-y),
```

```
                    cl = spam.train$y,
                    k=best.kfold)
knn.train.test=knn(train = spam.train%>%select(-y),
                    test = spam.test%>%select(-y),
                    cl = spam.train$y,
                    k=best.kfold)
```

```
records[1,1]=calc_error_rate(knn.train.train,spam.train$y)
records[1,2]=calc_error_rate(knn.train.test,spam.test$y)
```

3.Analysis of initial decision tree.

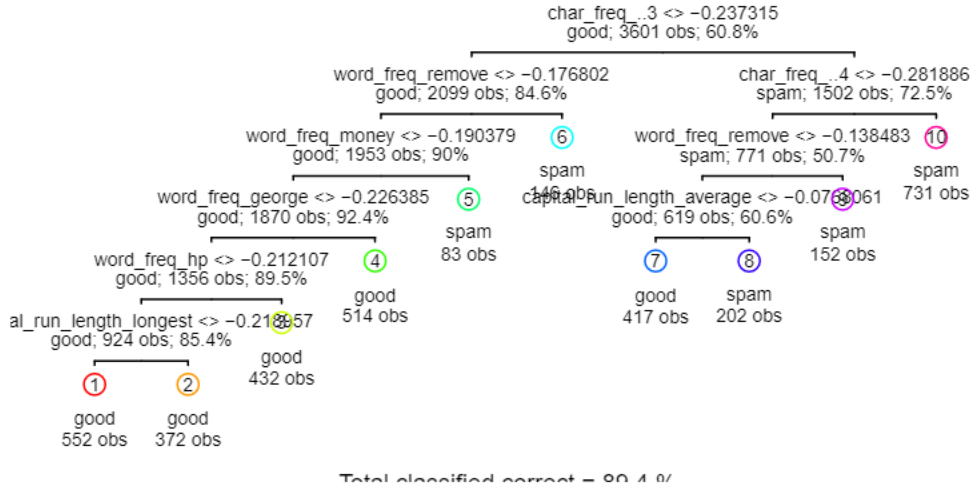
```
tcont=tree.control(nobs=3601,minsize = 6,mindev = 1e-6)
spamtree=tree(y~.,data=spam.train,control = tcont)
summary(spamtree)
```

```
##
## Classification tree:
## tree(formula = y ~ ., data = spam.train, control = tcont)
## Variables actually used in tree construction:
##  [1] "char_freq_..3"              "word_freq_remove"
##  [3] "word_freq_money"            "word_freq_george"
##  [5] "word_freq_hp"               "capital_run_length_longest"
##  [7] "word_freq_receive"          "word_freq_email"
##  [9] "word_freq_free"             "capital_run_length_average"
## [11] "word_freq_re"               "word_freq_you"
## [13] "capital_run_length_total"   "word_freq_your"
## [15] "word_freq_all"              "word_freq_will"
## [17] "word_freq_mail"             "word_freq_edu"
## [19] "word_freq_our"              "char_freq_..4"
## [21] "word_freq_over"             "word_freq_meeting"
## [23] "char_freq_."                "word_freq_85"
## [25] "word_freq_font"             "word_freq_order"
## [27] "word_freq_650"              "char_freq_..1"
## [29] "word_freq_data"             "word_freq_project"
## [31] "word_freq_hpl"              "word_freq_make"
## [33] "word_freq_address"          "word_freq_business"
## [35] "word_freq_1999"             "word_freq_internet"
## Number of terminal nodes:  172
## Residual mean deviance:  0.07285 = 249.8 / 3429
## Misclassification error rate: 0.01861 = 67 / 3601
```

There are 172 leaf nodes and 67 missclassified results.

4.Plotting 10 terminal node decision tree.

```
draw.tree(prune.tree(spamtree,best=10),cex=.6,nodeinfo = T)
```

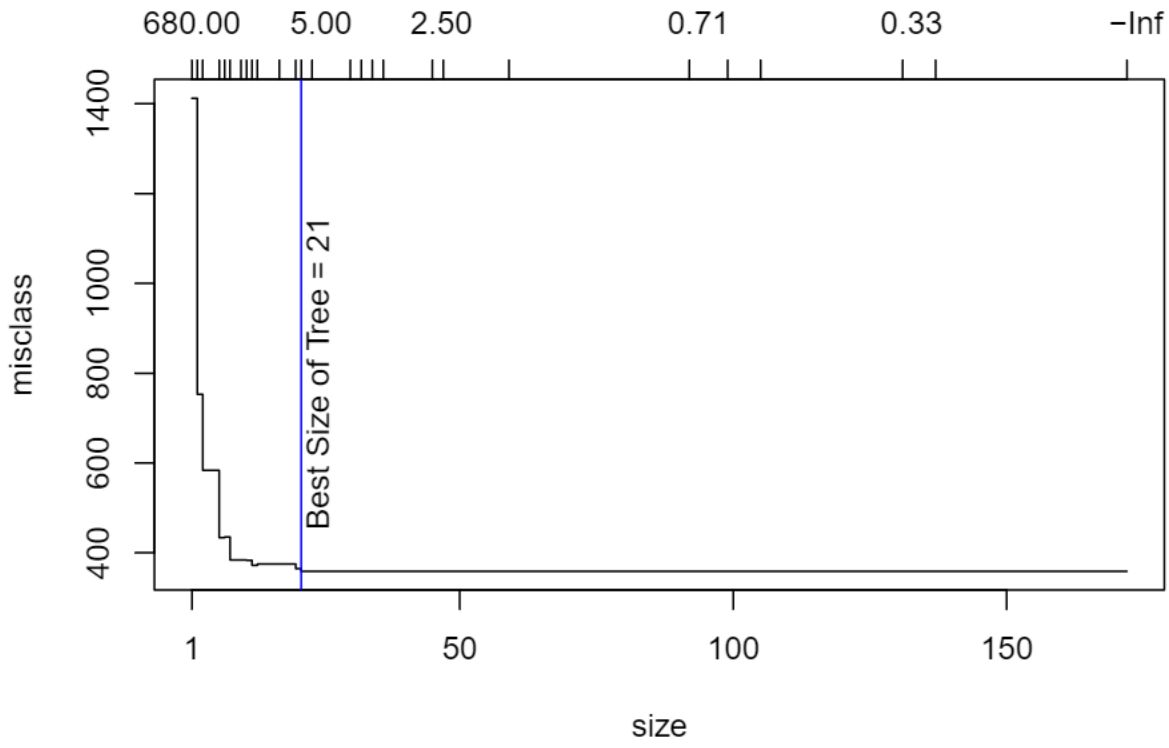


5.Determining optimal tree size and plotting result.

```
cv.spamtree=cv.tree(spamtree,FUN = prune.misclass,rand=folds,K=9)
plot(cv.spamtree)
best.size.cv=min(cv.spamtree$size[which(cv.spamtree$dev==min(cv.spamtree$dev))])
best.size.cv
```

```
## [1] 21
```

```
abline(v=best.size.cv,col="blue")
text(24,800,"Best Size of Tree = 21",srt=90)
```



6.Calculating error rates for the decision tree method.

```
spamtree.pruned=prune.tree(spamtree,best=best.size.cv)
```

```
tree.train=predict(spamtree.pruned,spam.train,type="class")
tree.test=predict(spamtree.pruned,spam.test,type="class")
```

```
records[2,1]=calc_error_rate(tree.train,spam.train$y)
records[2,2]=calc_error_rate(tree.test,spam.test$y)
```

7. If g(y) is an inverse function of f(x), then f(g(y))=y

$z(p)=\ln(\frac{p}{1-p})$

$p(z(p))=\frac{e^{(ln(\frac{p}{1-p}))}}{1+e^{(ln(\frac{p}{1-p})}}$

$=\frac{\frac{p}{1-p}}{1+\frac{p}{1-p}}$

$=\frac{p}{1-p} * \frac{1-p}{1-p+p}$

$=\frac{\frac{p(1-p)}{1-p}}{1-p+p}$

$=\frac{p}{1}$

$=p$

Thus, z(p) is the inverse function of p(z).

8.

```
spam.glm=glm(y~.,data=spam.train,family = quasibinomial())
spam.glm.train=round(predict(spam.glm,type="response"),2)
spam.glm.test=round(predict(spam.glm,spam.test,type="response"),2)
spam.train.glm.pred=spam.train%>%
  mutate(predspam=as.factor(ifelse(spam.glm.train<=.5,"good","spam")))
spam.test.glm.pred=spam.test%>%
  mutate(predspam=as.factor(ifelse(spam.glm.test<=.5,"good","spam")))
```

```
records[3,1]=calc_error_rate(spam.train.glm.pred$predspam,spam.train$y)
records[3,2]=calc_error_rate(spam.test.glm.pred$predspam,spam.test$y)
```

```
records
```

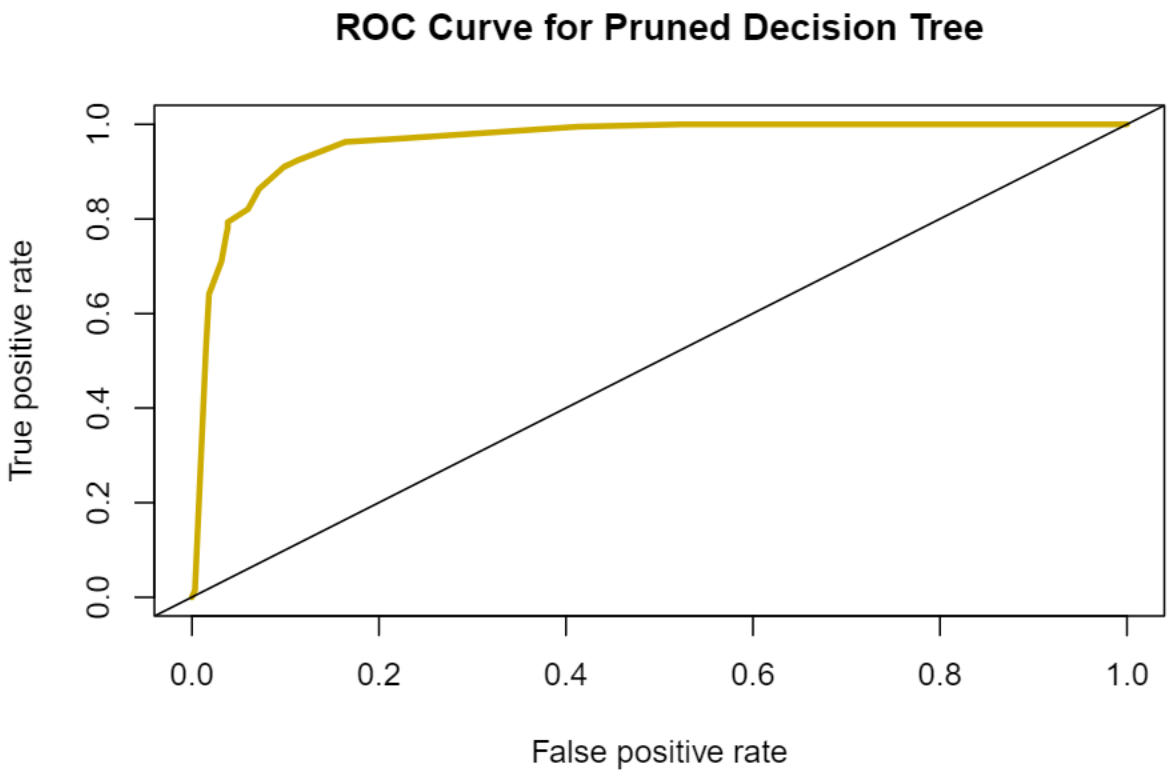
```
##           train.error test.error
## knn      0.0008331019      0.092
## tree     0.0785892808      0.098
## logistic 0.0658150514      0.077
```

Logistic regression had the lowest misclassification rate for test data.

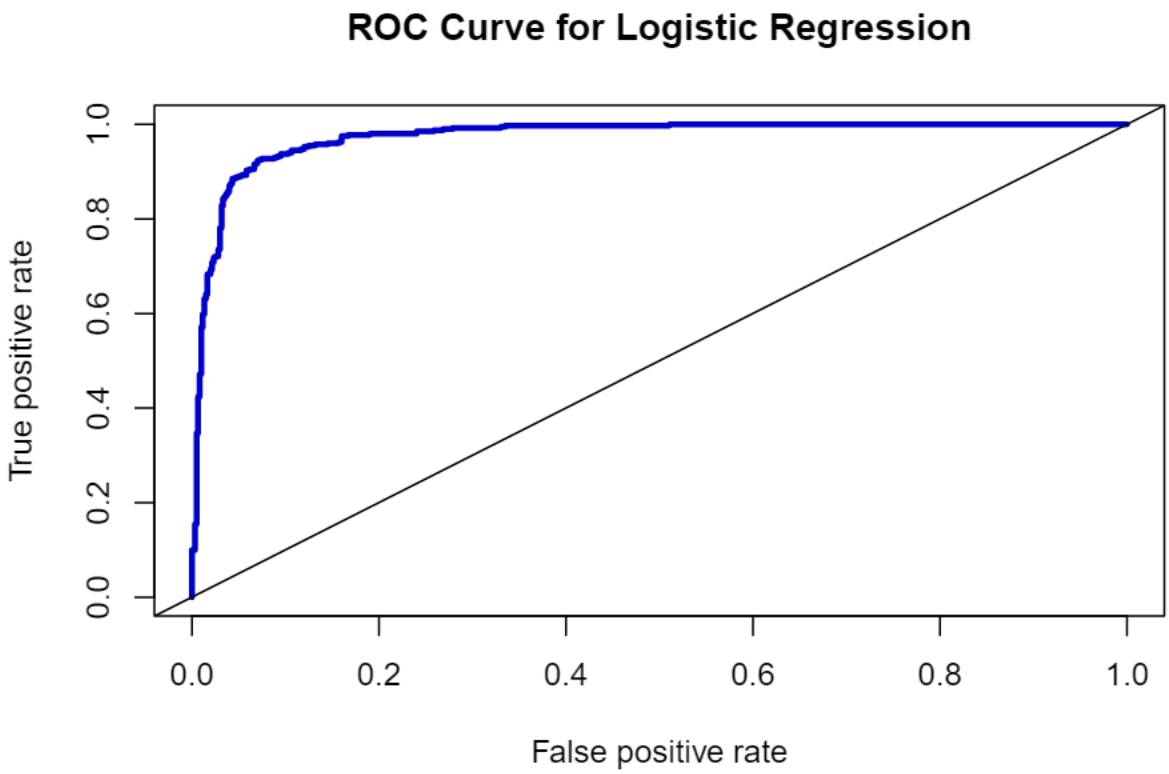
9.ROC curves for the Pruned Decision Tree and Logistic Regression Models.

```
spam.tree.predprob=predict(spamtree.pruned, spam.test, type="vector")[,2]
spam.glm.predprob=predict(spam.glm, spam.test, type="response")
spam.tree.prediction=prediction(spam.tree.predprob,spam.test$y)
spam.glm.prediction=prediction(spam.glm.predprob,spam.test$y)
tree.pref=performance(spam.tree.prediction,measure="tpr",x.measure = "fpr")
glm.pref=performance(spam.glm.prediction,measure="tpr",x.measure = "fpr")
```

```
plot(tree.pref,col="gold3",lwd=3,main="ROC Curve for Pruned Decision Tree")
abline(0,1)
```




```
plot(glm.pref,col="blue3",lwd=3,main="ROC Curve for Logistic Regression")
abline(0,1)
```



10. In this example a false positive result would mean that a non-spam email was incorrectly labeled as spam.

From the ROC curve, as we improve the the true positive rate we run the risk of increasing the false positive rate. Eventually we will hit the point where, in our efforts to increase the true positive rate, we incidentally increase the false positive rate considerably for marginal increases to the true positive rate.

As a designer, if I were to make a decision on how to balance the trade offs I would want to have it be so that the amount of false positive cases is minimized whilst still filtering out as much spam as possible. Reason being is that, as a user of email, I do not want any spam in my inbox and would much rather look for important emails in my spam folder than go through my inbox looking for spam.