Pstat 131 HW2

Kalvin Goode (9454554), Patrick Chen (970890)

2019/4/25

```
spam <- read_table2("spambase.tab", guess_max=2000)
spam <- spam %>%
mutate(y = factor(y, levels=c(0,1), labels=c("good", "spam"))) %>% # label as factors
mutate_at(.vars=vars(-y), .funs=scale) # scale others
```

```
calc_error_rate <- function(predicted.value, true.value)
{
  return(mean(true.value!=predicted.value))
}
```

```
records = matrix(NA, nrow=3, ncol=2)
colnames(records) <- c("train.error","test.error")
rownames(records) <- c("knn","tree","logistic")
```

```
set.seed(1)
test.indices = sample(1:nrow(spam), 1000)
spam.train=spam[-test.indices,]
spam.test=spam[test.indices,]
```

```
nfold = 10
set.seed(1)
folds = seq.int(nrow(spam.train)) %>% ## sequential obs ids
cut(breaks = nfold, labels=FALSE) %>% ## sequential fold ids
sample ## random fold ids
```

1.

```
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)
  data.frame(train.error = calc_error_rate(predYtr, Ytr),
             val.error = calc_error_rate(predYvl, Yvl))
}
```

```
set.seed(1)
kvec=c(1,seq(10,50,length.out=5))
error.folds=NULL
YTrain=spam.train$y
XTrain=spam.train%>% select(-y)
YTest=spam.test$y
XTest=spam.test%>% select(-y)
```

```r
for(j in kvec)
{
  tmp=ldply(1:nfold, do.chunk, folddef=folds, Xdat=XTrain, Ydat=YTrain, k=j)
  tmp$neighbors=j
  error.folds=rbind(error.folds,tmp)
}
```

```r
error_group=error.folds %>%
            group_by(neighbors) %>%
            summarise_at(funs(mean), .var=vars(train.error,val.error))
#error_group
max(error_group$neighbors[error_group$val.error==min(error_group$val.error)])
```

```
## [1] 10
```

We see that the when k=10, it has the smallest estimated test error among other values.

2.

```r
set.seed(1)
XTrain=spam.train%>% select(-y)
YTest=spam.test$y
YTrain=spam.train$y
XTest=spam.test%>% select(-y)
pred.YTest=knn(train=XTrain,test=XTest,cl=YTrain,k=10)
pred.YTrain=knn(train=XTrain,test=XTrain,cl=YTrain,k=10)
test_error=calc_error_rate(pred.YTest,YTest)
train_error=calc_error_rate(pred.YTrain,YTrain)
records[1,]=c(train_error,test_error)
records
```

```
##          train.error test.error
## knn       0.08414329      0.093
## tree              NA         NA
## logistic          NA         NA
```

3.

```r
control=tree.control(nobs=nrow(spam.train),minsize=5,mindev=1e-5)
spamtree=tree(y~.,data=spam.train,control=control)
summary(spamtree)
```

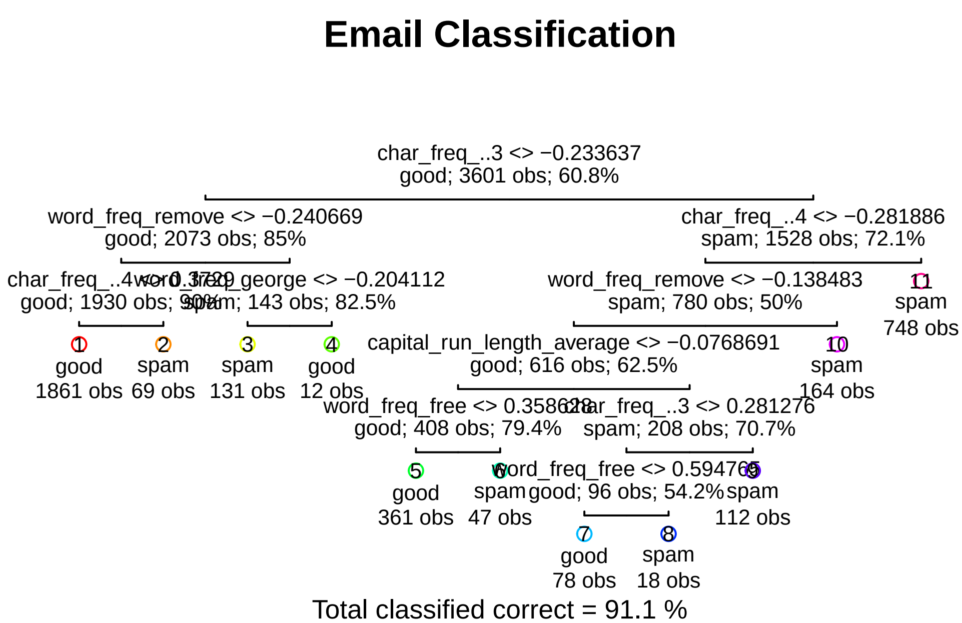
```
##
## Classification tree:
## tree(formula = y ~ ., data = spam.train, control = control)
## Variables actually used in tree construction:
##  [1] "char_freq_..3"              "word_freq_remove"
##  [3] "char_freq_..4"              "word_freq_george"
##  [5] "word_freq_hp"               "capital_run_length_longest"
##  [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_your"
## [17] "word_freq_will"             "char_freq_..1"
## [19] "word_freq_meeting"          "word_freq_1999"
## [21] "word_freq_make"             "word_freq_hpl"
```

```
## [23] "char_freq_."                "word_freq_over"
## [25] "word_freq_font"             "word_freq_report"
## [27] "word_freq_money"            "word_freq_address"
## [29] "word_freq_all"              "word_freq_000"
## [31] "word_freq_data"             "word_freq_project"
## [33] "word_freq_people"           "word_freq_email"
## [35] "word_freq_415"              "word_freq_edu"
## [37] "word_freq_technology"       "word_freq_mail"
## [39] "word_freq_business"         "char_freq_..2"
## [41] "word_freq_order"            "char_freq_..5"
## Number of terminal nodes:  184
## Residual mean deviance:  0.04748 = 162.2 / 3417
## Misclassification error rate: 0.01333 = 48 / 3601
```

We see that there are 48 training observations misclassified and 184 leaf nodes.

4.

```
prtree=prune.tree(spamtree,best=10,method="misclass")
draw.tree(prtree,nodeinfo=TRUE,cex=0.7,size=1)
title("Email Classification")
```

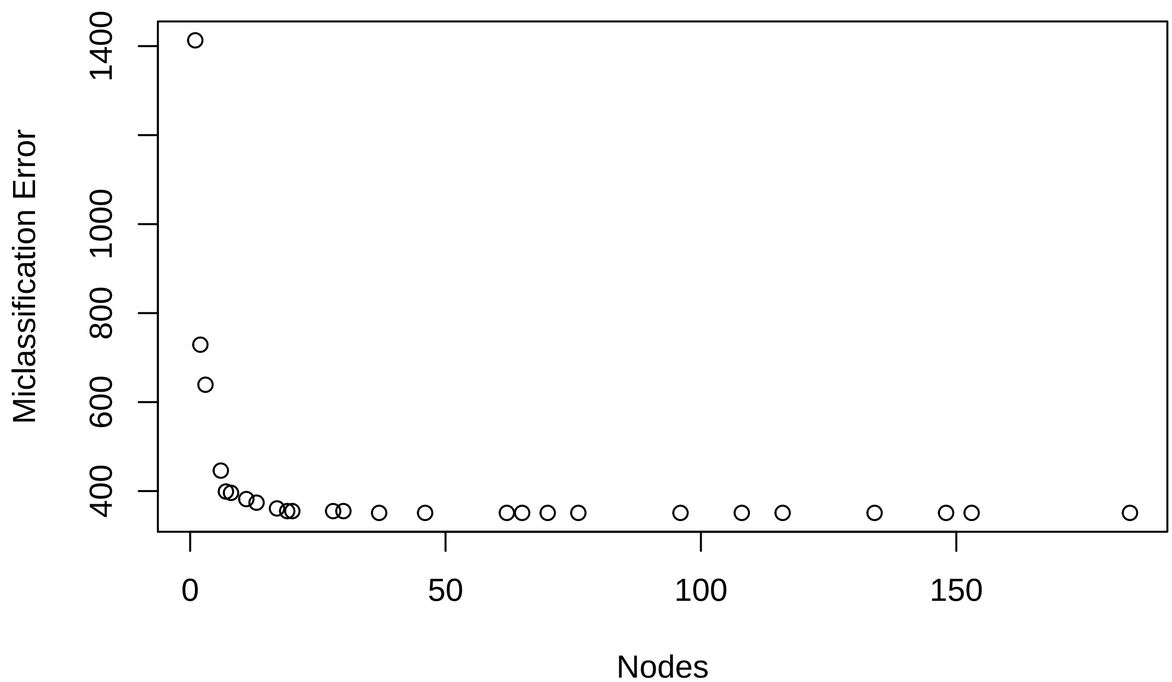


5.

```
crv=cv.tree(spamtree,K=10,rand=folds,method="misclass")
best=min(crv$size[crv$dev==min(crv$dev)])
sprintf("The best size of tree is %i.", best)
```

```
## [1] "The best size of tree is 37."
```

```r
plot(crv$size, crv$dev,xlab="Nodes",ylab="Miclassification Error")
```



6.

```r
prspamtree=prune.misclass(spamtree,best=37)
cvtrain=predict(prspamtree,spam.train,type="class")
cvtest=predict(prspamtree,spam.test,type="class")
errortr=calc_error_rate(cvtrain,YTrain)
errorte=calc_error_rate(cvtest,YTest)
records[2,]=c(errortr,errorte)
records
```

```
##          train.error test.error
## knn       0.08414329      0.093
## tree      0.05165232      0.072
## logistic          NA         NA
```

7.

(a). We have $p(x) = \frac{e^z}{1+e^z}$. Multiply both sides by $(1 + e^z)$, we have $(1 + e^z)p(x) = e^z$. By distributive property of real number, we have $p(x) + e^z p(x) = e^z$. With arrangement, we see that $e^z = \frac{p(z)}{1-p(z)}$. Therefore, by taking natural log on both sides, we have $z(p) = ln(\frac{p}{1-p})$. Thus, the inverse of a logistic function is the logit function.

(b). Let $z = \beta_0 + \beta_1 x_1$ and p=logistic(z), then, the logit function becomes $\beta_0 + \beta_1 x_1 = ln(\frac{p}{1-p})$. Take exponential on both sides, we have $\frac{p}{1-p} = e^{\beta_0+\beta_1 x_1}$. If we increase x_1 by two, we have $\frac{p}{1-p} = e^{\beta_0+\beta_1(x_1+2)} = e^{(\beta_0+\beta_1 x_1)}e^{2\beta_1}$. This means that the odd would change to $e^{2\beta_1}$.

Assume β_1 is negative, then as $x_1 \to \infty$, the value of p approach to 0.

8.

```
lg=glm(y~.,data=spam.train,family="binomial")
fit1=predict(lg,newdata=spam.train,type="response")
fit2=predict(lg,newdata=spam.test,type="response")
tab1=table(Truth=spam.train$y,
            prediction=ifelse(fit1>0.5,"spam","good"))
tab2=table(Truth=spam.test$y,
           prediction=ifelse(fit2>0.5,"spam","good"))
tab1
```

```
##       prediction
## Truth  good spam
##   good 2087  101
##   spam  154 1259
```

```
tab2
```

```
##       prediction
## Truth  good spam
##   good  574   26
##   spam   55  345
```

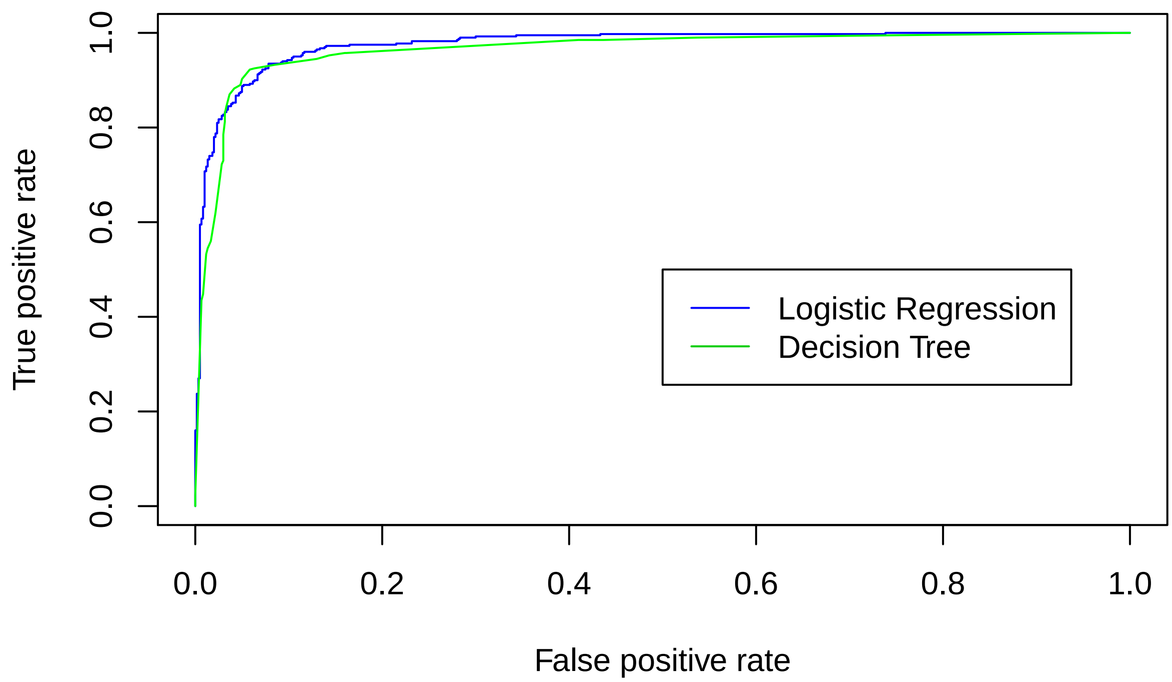
```
train_log_error=(tab1[1,2]+tab1[2,1])/sum(tab1)
test_log_error=(tab2[1,2]+tab2[2,1])/sum(tab2)
records[3,]=c(train_log_error,test_log_error)
records
```

```
##          train.error test.error
## knn       0.08414329      0.093
## tree      0.05165232      0.072
## logistic  0.07081366      0.081
```

9.

```
prediction1=prediction(fit2,spam.test$y)
performance1=performance(prediction1,measure="tpr",x.measure="fpr")
plot(performance1,col="Blue")

prediction2=prediction(predict(prspamtree,
                               spam.test,
                               type="vector")[,2],spam.test$y)
performance2=performance(prediction2,
                         measure="tpr",x.measure="fpr")
plot(performance2,col="Green",add=TRUE)
legend(0.5,0.5,col=c(4,3),
       legend=c("Logistic Regression","Decision Tree"),lwd = c(1,1))
```




```r
performance(prediction1,measure="auc")@y.values #AUC of log reg
```

```
## [[1]]
## [1] 0.9758875
```

```r
performance(prediction2,measure="auc")@y.values #AUC of tree
```

```
## [[1]]
## [1] 0.9647083
```

We see that the area under the ROC curve of Logistic Regression is larger than the area under the ROC curve of Decision Tree. Therefore, Logistic Regression is better in predicting spam emails.

10.

If we were the designer of a spam filter, we would be more concerned about potential of false positive rates that are too large. We don't want to missclassify emails that contain important message to users, which may potenitally ruin the work experience of users. If FPR is too large, users may have to occassionally look through spam folders to make sure emails are not missclassified, which is tedious to do. We would rather let users manually remove spam emails when filter models decide they are legitimate.