PSTAT 131 Homework 2

Tanner Berney 7215445

5/1/2021

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tree)
```

```
## Registered S3 method overwritten by 'tree':
##   method     from
##   print.tree cli
```

```
library(plyr)
```

```
## ------------------------------------------------------------------------------

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)

## ------------------------------------------------------------------------------

##
## Attaching package: 'plyr'

## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## The following object is masked from 'package:purrr':
##
##     compact
```

```
library(class)
library(rpart)
library(maptree)
```

```
## Loading required package: cluster
```

```
library(ROCR)
library(reshape2)
```

```
##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
spam <- read_table2("spambase.tab.txt", guess_max=2000)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double()
## )
## i Use `spec()` for the full column specifications.
```

```
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
```

Question 1

```
set.seed(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(fold = chunkid, train.error = calc_error_rate(predYtr, Ytr), val.error = calc_error_rate(pr
}
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)
set.seed(1)
for (j in kvec){
 tmp = ldply(1:nfold, do.chunk, # apply do.function to each fold
 folddef=folds, Xdat=XTrain, Ydat=YTrain, k=j) # arguments
 tmp$neighbors = j # track each value of neighbors
 error.folds = rbind(error.folds, tmp) # combine the results
}
 errors = melt(error.folds, id.vars=c('fold', 'neighbors'), value.name='error') # Choose the number of
val.error.means = errors %>%
  # Select all rows of validation errors
  filter(variable=='val.error') %>%
  # Group the selected data frame by neighbors
  group_by(neighbors, variable) %>%
  # Calculate CV error rate for each k
  summarise_each(funs(mean), error) %>%
  # Remove existing group
  ungroup() %>%
  filter(error==min(error))
```

```
## Warning: `summarise_each_()` is deprecated as of dplyr 0.7.0.
## Please use `across()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.

## Warning: `funs()` is deprecated as of dplyr 0.8.0.
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
numneighbor = max(val.error.means$neighbors)
numneighbor
```

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

A value of k=10 leads to the smallest estimated test error.

Question 2

```
# Training error
set.seed(1)
pred.YTtrain = knn(train=XTrain, test=XTrain, cl=YTrain, k=10)
knn.training.error <- calc_error_rate(predicted.value=pred.YTtrain, true.value=YTrain)
knn.training.error
```

```
## [1] 0.07803388
```

```
# Test error
set.seed(1)
pred.YTest = knn(train=XTrain, test=XTest, cl=YTrain, k=10)
knn.test.error <- calc_error_rate(predicted.value=pred.YTest, true.value=YTest)
knn.test.error
```

```
## [1] 0.103
```

```
records[1,1:2] <- c(knn.training.error,knn.test.error)
records
```

```
##          train.error test.error
## knn       0.07803388      0.103
## tree              NA         NA
## logistic          NA         NA
```

Question 3

```
spamtree = tree(y ~., data = spam.train, control = tree.control(nobs= nrow(spam.train),minsize = 5, min
summary(spamtree)
```

```
##
## Classification tree:
## tree(formula = y ~ ., data = spam.train, control = tree.control(nobs = nrow(spam.train),
##     minsize = 5, mindev = 1e-05))
## Variables actually used in tree construction:
##  [1] "char_freq_..4"              "word_freq_remove"
##  [3] "char_freq_..3"              "word_freq_free"
##  [5] "word_freq_george"           "word_freq_hp"
##  [7] "capital_run_length_longest" "word_freq_receive"
##  [9] "word_freq_credit"           "capital_run_length_average"
## [11] "word_freq_your"             "word_freq_mail"
## [13] "word_freq_re"               "word_freq_our"
## [15] "word_freq_you"              "capital_run_length_total"
```

```
## [17] "word_freq_make"             "word_freq_all"
## [19] "word_freq_internet"         "word_freq_email"
## [21] "word_freq_project"          "word_freq_money"
## [23] "word_freq_1999"             "word_freq_will"
## [25] "char_freq_..1"              "word_freq_order"
## [27] "char_freq_."                "word_freq_data"
## [29] "word_freq_over"             "word_freq_meeting"
## [31] "word_freq_650"              "word_freq_edu"
## [33] "word_freq_address"          "word_freq_business"
## Number of terminal nodes:  149
## Residual mean deviance:  0.04568 = 157.7 / 3452
## Misclassification error rate: 0.01361 = 49 / 3601
```

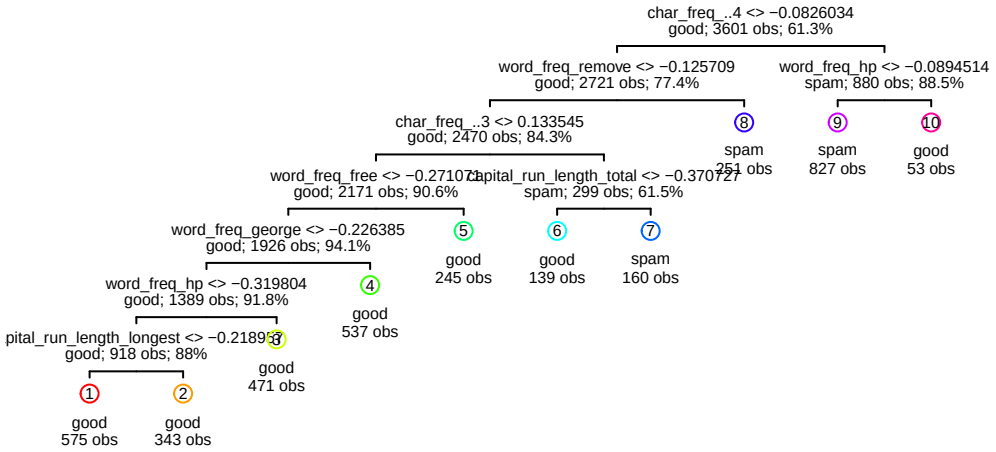
There are 149 leaf nodes. 49 observations are misclassified.

Question 4

```
prune <- prune.tree(spamtree,best=10,method="deviance")
summary(prune)
```

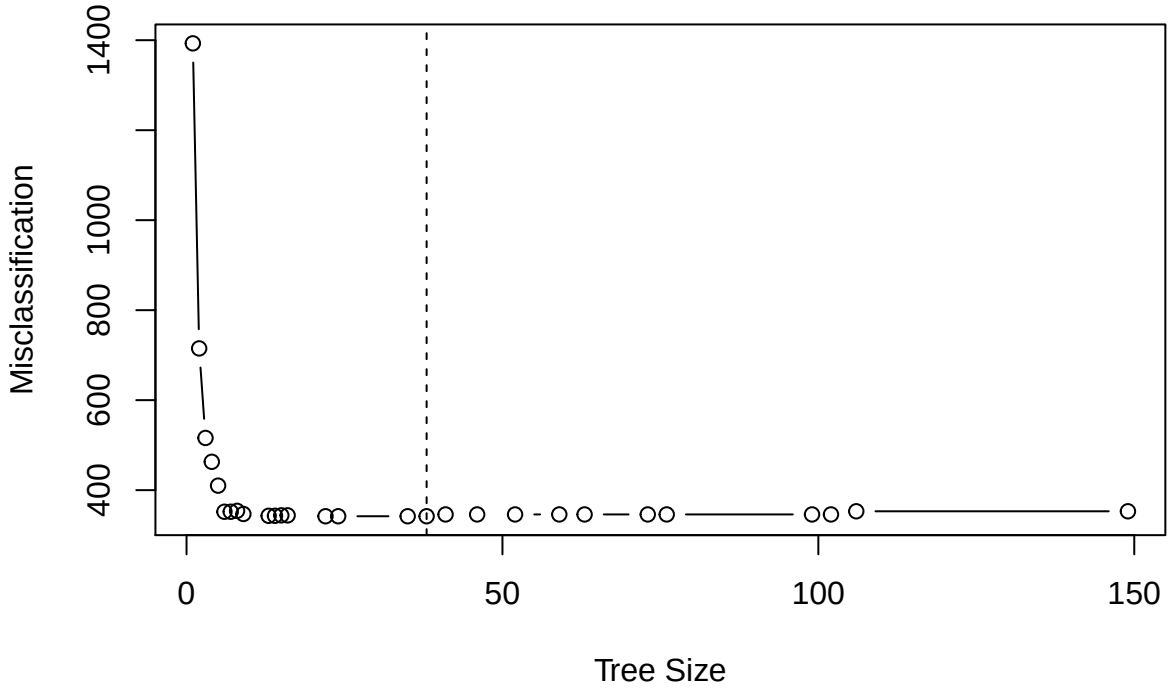
```
##
## Classification tree:
## snip.tree(tree = spamtree, nodes = c(7L, 65L, 128L, 19L, 17L,
## 18L, 129L, 5L, 6L))
## Variables actually used in tree construction:
## [1] "char_freq_..4"              "word_freq_remove"
## [3] "char_freq_..3"              "word_freq_free"
## [5] "word_freq_george"           "word_freq_hp"
## [7] "capital_run_length_longest" "capital_run_length_total"
## Number of terminal nodes:  10
## Residual mean deviance:  0.5089 = 1828 / 3591
## Misclassification error rate: 0.09692 = 349 / 3601
```

```
draw.tree(prune,nodeinfo = TRUE,cex = .5)
```



Question 5

```
set.seed(1)
cv <- cv.tree(spamtree,rand=folds,FUN = prune.misclass,K=10)
best.size.cv = cv$size[which.min(cv$dev)]
plot(cv$size, cv$dev, type="b",xlab="Tree Size",ylab="Misclassification")
abline(v=best.size.cv,lty="dashed")
```



Question 6

```
spamtree.pruned <- prune.misclass(spamtree,best=best.size.cv)
spamtree.pred.train <- predict(spamtree.pruned,spam.train,type="class")
spamtree.pred.test <- predict(spamtree.pruned,spam.test,type="class")
tree.training.error <- calc_error_rate(predicted.value=spamtree.pred.train,true.value =spam.train$y)
tree.training.error
```

```
## [1] 0.0427659
```

```
tree.test.error <- calc_error_rate(predicted.value=spamtree.pred.test,true.value =spam.test$y)
tree.test.error
```

```
## [1] 0.098
```

```
records[2,1:2] <- c(tree.training.error,tree.test.error)
```

Question 7 a-b

Question 7a

$$
\begin{aligned}
p(z) &= \frac{e^z}{1+e^z} \\
\frac{p(z)}{1-p(z)} &= \frac{\frac{e^z}{1+e^z}}{1-\frac{e^z}{1+e^z}} \\
&= \frac{\frac{e^z}{1+e^z}}{1-\frac{e^z}{1+e^z}} * \frac{1+e^z}{1+e^z} \\
&= \frac{e^z}{(1+e^z) * (\frac{1+e^z}{1+e^z} - \frac{e^z}{1+e^z})} \\
&= \frac{e^z}{(1+e^z) * (\frac{1+e^z-e^z}{1+e^z})} \\
&= \frac{e^z}{(1+e^z) * (\frac{1}{1+e^z})} \\
&= e^z \\
&= \ln(e^z) \\
&= z \\
\ln(\frac{p}{1-p}) &= z(p)
\end{aligned}
$$

Question 7b

For every one unit change of x1, the log odds increases by B1. So Increasing x1 by two will result in a increase of B1 two times. As x1 approaches infinity, p approaches 0. As x1 approaches negative infinity p approaches 1.

Question 8

```
glm.fit <- glm(y ~ .,data=spam, family=binomial("logit"))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
train.pred <- ifelse(predict(glm.fit, newdata=spam.train,type = "response") <= 0.5, "good", "spam")
test.pred <- ifelse(predict(glm.fit, newdata = spam.test, type = "response") <= 0.5, "good", "spam")
logistic.training.error <- calc_error_rate(predicted.value=train.pred, true.value=spam.train$y)
logistic.training.error
```

```
## [1] 0.06775896
```

```
logisitc.testing.error <- calc_error_rate(predicted.value=test.pred, true.value=spam.test$y)
logisitc.testing.error
```

```
## [1] 0.072
```

```
records[3,1:2] <- c(logistic.training.error,logisitc.testing.error)
records
```

```
##          train.error test.error
## knn       0.07803388      0.103
## tree      0.04276590      0.098
## logistic  0.06775896      0.072
```

The logistic method had the lowest misclassification error on the test set being 0.072

Question 9

If I was the designer of the Spam filter I would be more concerned about having a large false positive rate. This is because if someone was using this to filter their emails, there is a good chance that a lot of their mail will be marked as spam when it is really not spam. This could than cause them to not read important emails. If I had a small true positive rate, I may be missing more of the spam, however it would allow the user to not have their important emails get marked as spam.