Homework2

Justin Lau

10/28/2020

Libraries

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.0 ──
```

```
## ✓ ggplot2 3.3.2     ✓ purrr   0.3.4
## ✓ tibble  3.0.4     ✓ dplyr   1.0.2
## ✓ tidyr   1.1.2     ✓ stringr 1.4.0
## ✓ readr   1.4.0     ✓ forcats 0.5.0
```

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

```
library(reshape2)
```

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

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

```
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)
```

```
spam <- read_table2("spambase.tab", 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"))) %>%
  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
```

```
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(predYvl, Yvl))
}
```

Question 1

```
error.folds <- NULL
YTrain <- spam.train$y
XTrain <- spam.train %>% select(-y)

YTest <- spam.test$y
XTest <- spam.test %>% select(-y)

set.seed(1)
kvec = c(1, seq(10, 50, length.out=5))
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)
}
error.folds
```

```
##    fold  train.error  val.error neighbors
## 1     1 0.0006172840 0.11080332         1
## 2     2 0.0000000000 0.11944444         1
## 3     3 0.0006170935 0.08055556         1
## 4     4 0.0000000000 0.08055556         1
## 5     5 0.0006170935 0.10833333         1
## 6     6 0.0006170935 0.11111111         1
## 7     7 0.0003085467 0.07777778         1
## 8     8 0.0000000000 0.11666667         1
## 9     9 0.0003085467 0.10000000         1
## 10   10 0.0003085467 0.13055556         1
## 11    1 0.0824074074 0.08864266        10
## 12    2 0.0823819809 0.11111111        10
## 13    3 0.0805307004 0.08888889        10
## 14    4 0.0774452330 0.10000000        10
## 15    5 0.0755939525 0.09722222        10
## 16    6 0.0762110460 0.10277778        10
## 17    7 0.0805307004 0.05833333        10
## 18    8 0.0789879667 0.09444444        10
## 19    9 0.0759024992 0.11111111        10
## 20   10 0.0786794199 0.11388889        10
## 21    1 0.0919753086 0.09418283        20
## 22    2 0.0944153039 0.11944444        20
## 23    3 0.0956494909 0.08055556        20
## 24    4 0.0934896637 0.08888889        20
## 25    5 0.0888614625 0.12500000        20
## 26    6 0.0882443690 0.11111111        20
## 27    7 0.0965751311 0.06944444        20
## 28    8 0.0907127430 0.10555556        20
## 29    9 0.0931811169 0.12777778        20
## 30   10 0.0910212897 0.10000000        20
## 31    1 0.0993827160 0.10249307        30
## 32    2 0.1024375193 0.12500000        30
## 33    3 0.1052144400 0.10000000        30
## 34    4 0.1030546128 0.10555556        30
## 35    5 0.0993520518 0.11666667        30
## 36    6 0.0984264116 0.10833333        30
## 37    7 0.1033631595 0.07777778        30
## 38    8 0.0971922246 0.12777778        30
## 39    9 0.1012033323 0.11944444        30
## 40   10 0.0990435051 0.10833333        30
## 41    1 0.1055555556 0.11357341        40
## 42    2 0.1058315335 0.11666667        40
## 43    3 0.1104597346 0.11111111        40
## 44    4 0.1052144400 0.10555556        40
## 45    5 0.1101511879 0.12222222        40
## 46    6 0.1073742672 0.12222222        40
## 47    7 0.1098426412 0.08055556        40
## 48    8 0.1021289725 0.13055556        40
## 49    9 0.1098426412 0.12500000        40
## 50   10 0.1033631595 0.09722222        40
## 51    1 0.1111111111 0.11080332        50
## 52    2 0.1129281086 0.12222222        50
## 53    3 0.1135452021 0.11388889        50
## 54    4 0.1110768281 0.11111111        50
## 55    5 0.1104597346 0.12222222        50
## 56    6 0.1116939216 0.11944444        50
## 57    7 0.1141622956 0.08055556        50
## 58    8 0.1110768281 0.14722222        50
## 59    9 0.1082999074 0.12222222        50
## 60   10 0.1089170009 0.10555556        50
```

```
errors = melt(error.folds, id.vars=c('fold', 'neighbors'), value.name='error')
val.error.means = errors %>%
  filter(variable=='val.error') %>%
  group_by(neighbors, variable) %>%
  summarise_each(funs(mean), error) %>%
  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
```

```
sum(is.na(spam.test))
```

```
## [1] 0
```

Question 2

```
train_error <- do.chunk(6, folds, Xdat = XTrain, Ydat = YTrain, k = 10)

pred.YTest = knn(train=XTrain, test=XTest, cl=YTrain, k=numneighbor)
test_error <- calc_error_rate(pred.YTest,YTest)

records <- replace(records,1, train_error$train.error)
records <- replace(records,4, test_error)
```

Question 3

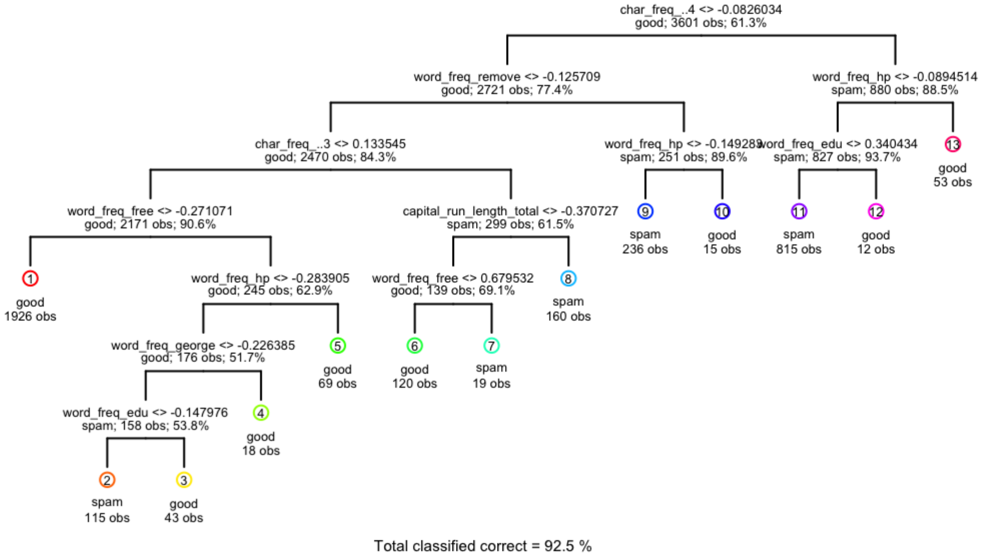
```
controls <- tree.control(nobs= nrow(spam.train),mincut=5, mindev=1e-5)
spamtree <- tree(y~., spam.train, control=controls)
summary(spamtree)
```

```
##
## Classification tree:
## tree(formula = y ~ ., data = spam.train, control = controls)
## 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] "capital_run_length_average" "word_freq_credit"
## [11] "word_freq_your"             "word_freq_mail"
## [13] "word_freq_re"               "word_freq_you"
## [15] "capital_run_length_total"   "word_freq_will"
## [17] "word_freq_edu"              "word_freq_people"
## [19] "word_freq_money"            "word_freq_our"
## [21] "word_freq_1999"             "word_freq_make"
## [23] "char_freq_."                "word_freq_data"
## [25] "word_freq_all"              "word_freq_over"
## [27] "char_freq_..1"              "word_freq_project"
## [29] "word_freq_meeting"          "word_freq_internet"
## [31] "word_freq_650"              "word_freq_hpl"
## [33] "char_freq_..5"              "word_freq_email"
## [35] "word_freq_business"         "word_freq_order"
## [37] "word_freq_address"
## Number of terminal nodes:  129
## Residual mean deviance:  0.1071 = 371.7 / 3472
## Misclassification error rate: 0.02555 = 92 / 3601
```

There was 133 missclassified observations and there are 141 leaf nodes.

Question 4

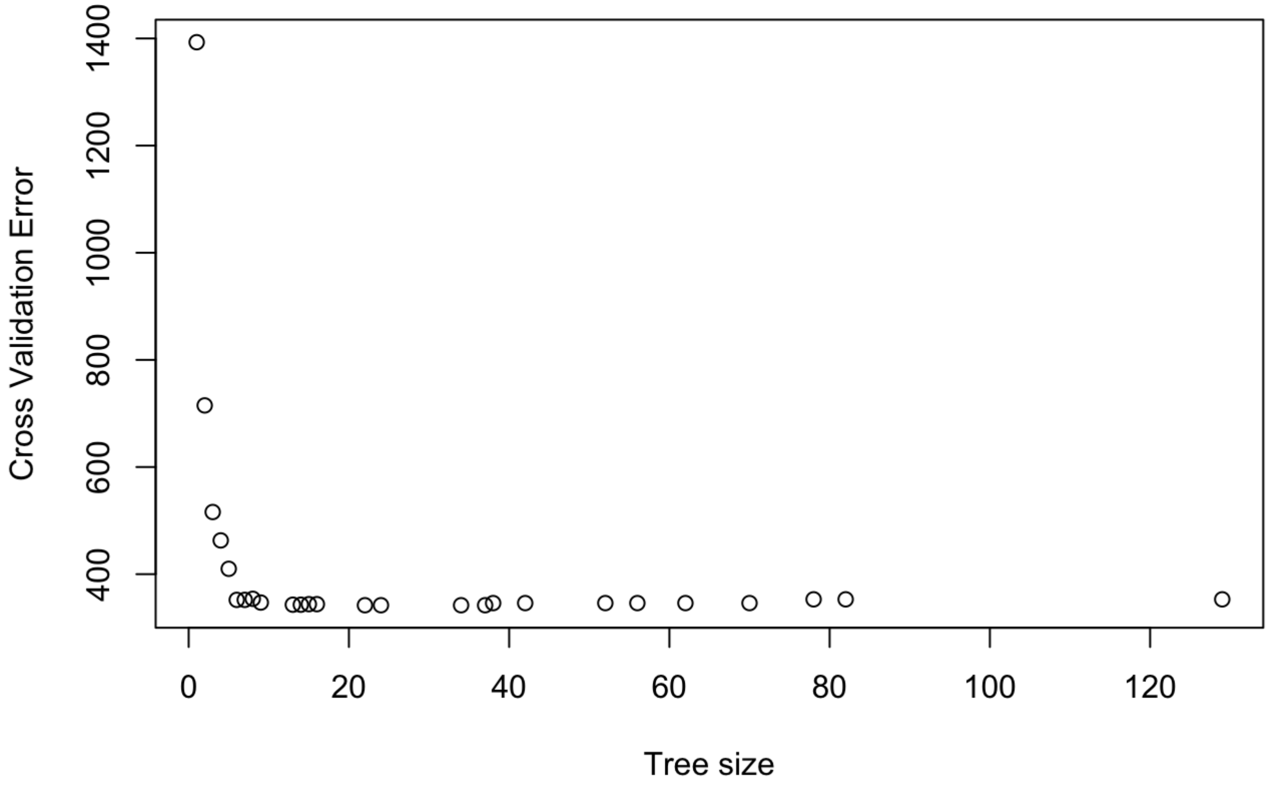
```
prune <- prune.tree(spamtree,best=10, method = 'misclass')
draw.tree(prune, nodeinfo = TRUE, cex = 0.4)
```



Question 5

```
cv <- cv.tree(spamtree, rand=folds, FUN = prune.misclass, K=10)

plot(cv$size, cv$dev, xlab = "Tree size", ylab = "Cross Validation Error")
```



```
best.size.cv = min(cv$size[cv$dev == 351])
```

```
## Warning in min(cv$size[cv$dev == 351]): no non-missing arguments to min;
## returning Inf
```

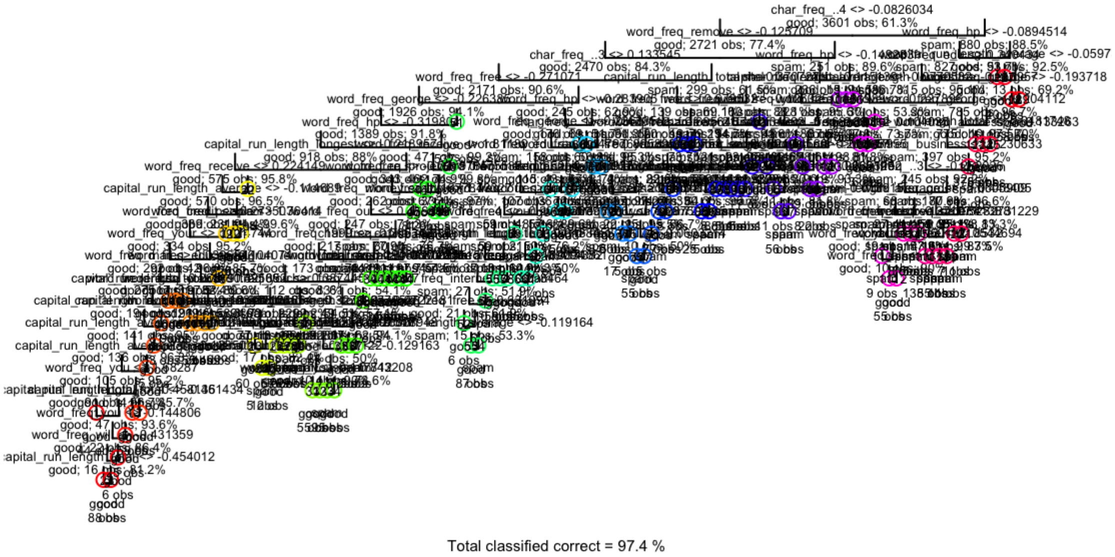
```
best.size.cv
```

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

```
spamtree.pruned <- prune.misclass(spamtree, best = best.size.cv)
```

```
## Warning in prune.tree(tree = spamtree, best = best.size.cv, method =
## "misclass"): best is bigger than tree size
```

```
draw.tree(spamtree.pruned, nodeinfo = TRUE, cex = 0.4)
```



35 is the optimal amount tree size.

Question 6

```
predict.pruned.test <- predict(spamtree.pruned, spam.test, type = 'class')
predict.pruned.train <- predict(spamtree.pruned, spam.train, type = 'class')
prune.test.error <- calc_error_rate(predict.pruned.test, spam.test$y)
prune.train.error <- calc_error_rate(predict.pruned.train, spam.train$y)

records <- replace(records,2, prune.train.error)
records <- replace(records,5, prune.test.error)
```

Question 7 will be written out in a separate segment

Question 8

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

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

```
prob.train = predict(glm.fit, spam.train, type="response")

prob.test = predict(glm.fit, spam.test, type="response")

spam.test <-
  spam.test %>%
  mutate(predSPAM = as.factor(ifelse(prob.test <= .5, "good", "spam")))

spam.train <-
  spam.train %>%
  mutate(predSPAM = as.factor(ifelse(prob.train <= .5, "good", "spam")))
```

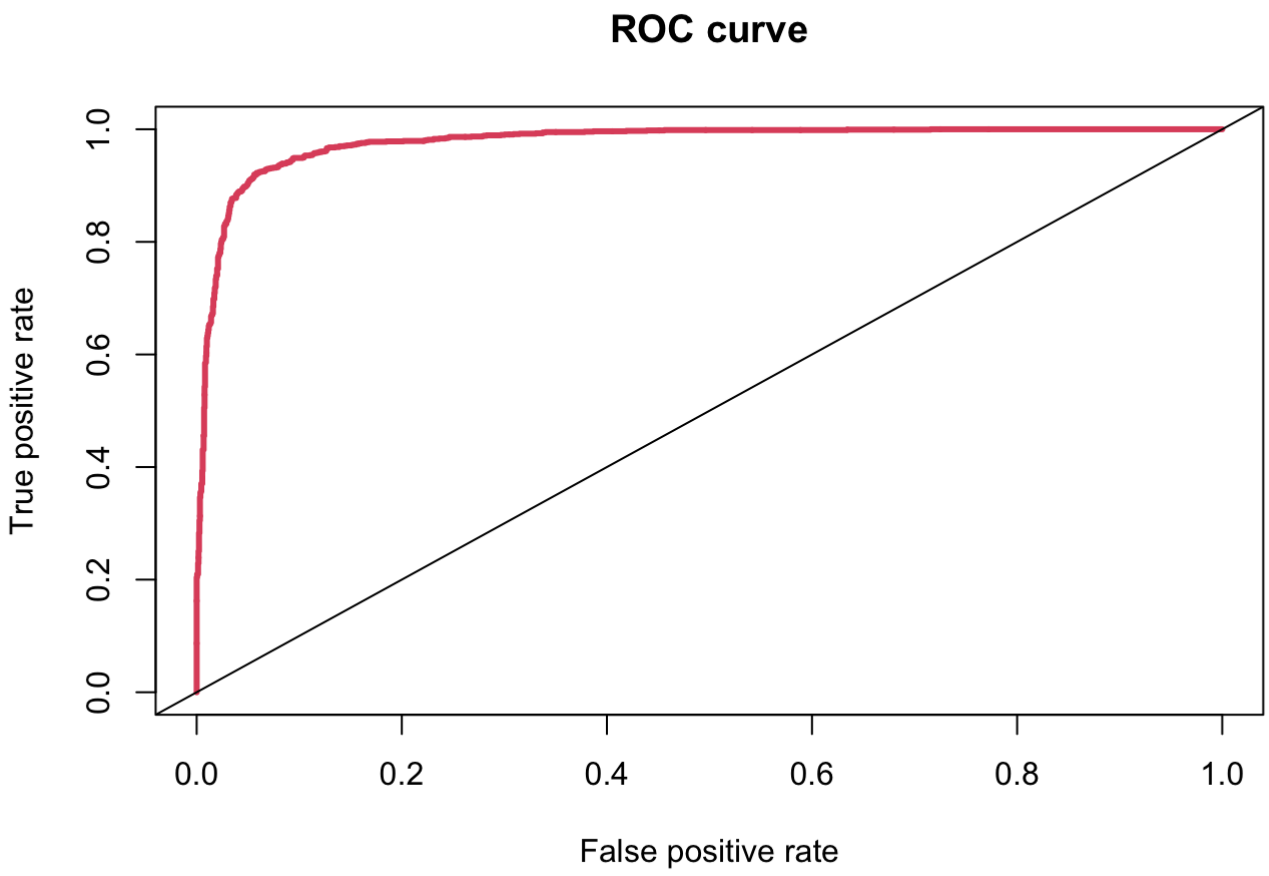
```
log.test.error <- calc_error_rate(spam.test$predSPAM, spam.test$y)
log.train.error <- calc_error_rate(spam.train$predSPAM, spam.train$y)

records <- replace(records,3, log.train.error)
records <- replace(records,6, log.test.error)
print(records)
```

```
##          train.error test.error
## knn       0.08083925      0.103
## tree      0.02471536      0.098
## logistic  0.06803666      0.086
```

Question 9

```
pred <- prediction(prob.train, spam.train$y)
perf = performance(pred, measure="tpr", x.measure="fpr")
plot(perf, col=2, lwd=3, main="ROC curve")
abline(0,1)
```



```
auc = performance(pred, "auc")@y.values
auc
```

```
## [[1]]
## [1] 0.9772759
```

We are more worried with false positive rates that are too large as that would filter out emails that could be potentially important to a client. While having a large true positive rate that is too small would mean that not a lot of spam is being filtered out, almost making the filter worthless, having a large false positive rate would be more of a detriment more than anything else.