HW2_JL

Justin Lee & Brendan Morrison

10/30/2020

```r
library(tidyverse)
library(tree)
library(plyr)
library(class)
library(rpart)
library(maptree)
library(ROCR)
library(dplyr)
```

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

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

```r
spam <- spam %>%
  mutate(spam = as.factor(ifelse(y <= median(y), "good", "spam")))


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
folds = seq.int(nrow(spam.train)) %>%
  cut(breaks = nfold, labels=FALSE) %>%
  sample
```

(1)

```r
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))
}




YTrain = spam.train$spam
XTrain = scale(spam.train %>% select(-spam))

YTest = spam.test$spam
XTest = scale(spam.test %>% select(-spam))


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


for (i in kvec){
  tmp = ldply(1:nfold, do.chunk,
              folddef = folds, Xdat = XTrain, Ydat = YTrain, k =i)
  tmp$folds = seq(1,10,1)
  tmp$neighbors = i
  error.folds = rbind(error.folds,tmp)
}

error.folds
```

```
##    train.error  val.error folds neighbors
## 1   0.00000000 0.03047091     1         1
## 2   0.00000000 0.02500000     2         1
## 3   0.00000000 0.02222222     3         1
## 4   0.00000000 0.01388889     4         1
## 5   0.00000000 0.01388889     5         1
## 6   0.00000000 0.02222222     6         1
## 7   0.00000000 0.02500000     7         1
## 8   0.00000000 0.03055556     8         1
## 9   0.00000000 0.03611111     9         1
## 10  0.00000000 0.01666667    10         1
```

```
## 11  0.02006173 0.01939058     1        10
## 12  0.02005554 0.03055556     2        10
## 13  0.01974699 0.01666667     3        10
## 14  0.02036409 0.02222222     4        10
## 15  0.02190682 0.02500000     5        10
## 16  0.01882135 0.03333333     6        10
## 17  0.01882135 0.03055556     7        10
## 18  0.01727862 0.02777778     8        10
## 19  0.02036409 0.02500000     9        10
## 20  0.02005554 0.02222222    10        10
## 21  0.02314815 0.01939058     1        20
## 22  0.02283246 0.03055556     2        20
## 23  0.02560938 0.02222222     3        20
## 24  0.02437519 0.01388889     4        20
## 25  0.02375810 0.03333333     5        20
## 26  0.02283246 0.02777778     6        20
## 27  0.02344955 0.03333333     7        20
## 28  0.02375810 0.02500000     8        20
## 29  0.02283246 0.02500000     9        20
## 30  0.02375810 0.02777778    10        20
## 31  0.02407407 0.02770083     1        30
## 32  0.02499229 0.03333333     2        30
## 33  0.02375810 0.03055556     3        30
## 34  0.02468374 0.01388889     4        30
## 35  0.02283246 0.03611111     5        30
## 36  0.02283246 0.02500000     6        30
## 37  0.02344955 0.02500000     7        30
## 38  0.02499229 0.02222222     8        30
## 39  0.02746066 0.02222222     9        30
## 40  0.02344955 0.02777778    10        30
## 41  0.02561728 0.03047091     1        40
## 42  0.02746066 0.03888889     2        40
## 43  0.02746066 0.03333333     3        40
## 44  0.02869485 0.01388889     4        40
## 45  0.02622647 0.04166667     5        40
## 46  0.02684357 0.02500000     6        40
## 47  0.02715211 0.02777778     7        40
## 48  0.02622647 0.02222222     8        40
## 49  0.02900339 0.01944444     9        40
## 50  0.02622647 0.02500000    10        40
## 51  0.02654321 0.03324100     1        50
## 52  0.02530083 0.03611111     2        50
## 53  0.02684357 0.03055556     3        50
## 54  0.02591793 0.01111111     4        50
## 55  0.02221537 0.04166667     5        50
## 56  0.02622647 0.02500000     6        50
## 57  0.02591793 0.01944444     7        50
## 58  0.02530083 0.02222222     8        50
## 59  0.02715211 0.01666667     9        50
## 60  0.02776921 0.01666667    10        50
```

```r
#Now obtain the test errors for all the values of k (1,10,20,30,40,50).
error <- as.tibble(error.folds)
```

```
## Warning: `as.tibble()` is deprecated as of tibble 2.0.0.
## Please use `as_tibble()` instead.
## The signature and semantics have changed, see `?as_tibble`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```r
k_1 <- error %>% filter(neighbors == 1) %>% summarise(mean(val.error))
k_10 <- error %>% filter(neighbors == 10) %>% summarise(mean(val.error))
k_20 <- error %>% filter(neighbors == 20) %>% summarise(mean(val.error))
k_30 <- error %>% filter(neighbors == 30) %>% summarise(mean(val.error))
k_40 <- error %>% filter(neighbors == 40) %>% summarise(mean(val.error))
k_50 <- error %>% filter(neighbors == 50) %>% summarise(mean(val.error))

#find the smllaest value for the optimal k

best <- min(k_1, k_10, k_20, k_30, k_40, k_50)
best
```

```
## [1] 0.02360265
```

```r
k_1
```

```
##   mean(val.error)
## 1      0.02360265
```

We can see that using the min() function that when k = 1, we get the smallest estimated test error.

(2)

```r
#training error rate
pred_YTrain = knn(train = XTrain, test = XTrain, cl = YTrain, k = 10)
train_error = calc_error_rate(pred_YTrain,YTrain)
train_error
```

```
## [1] 0.01916134
```

```r
#test error rate
pred_YTest = knn( train = XTrain, test = XTest, cl = YTrain, k = 10)
test_error = calc_error_rate(pred_YTest, YTest)


records = matrix(c(train_error, NA, NA, test_error, NA, NA), nrow = 3, ncol = 2)
colnames(records) <- c("train_error","test_error")
rownames(records) <- c("knn", "tree", "logistic")
records
```

```
##          train_error test_error
## knn       0.01916134      0.033
## tree              NA         NA
## logistic          NA         NA
```

(3)

```r
#Re-read the table due to number 1
spam <- read_table2("spambase.tab", guess_max=2000)
```

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

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

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

YTest = spam.test$y
YTrain = spam.train$y

nrow(spam.train)
```

```
## [1] 3601
```

```r
spamtree = tree(y ~ ., data = spam.train,
                control = tree.control(3601, minsize = 5, mindev = 0.00001))

summary(spamtree)
```

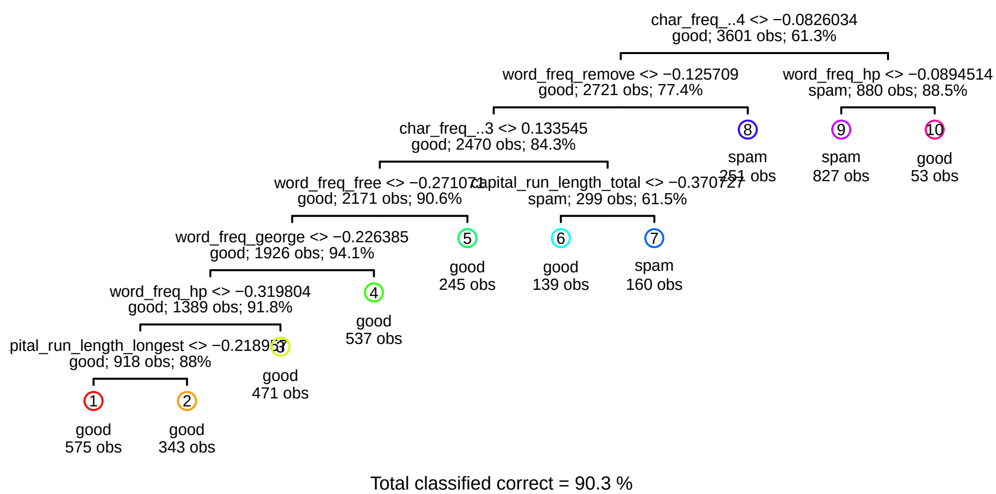
```
##
## Classification tree:
## tree(formula = y ~ ., data = spam.train, control = tree.control(3601,
##     minsize = 5, mindev = 1e-05))
## Variables actually used in tree construction:
##  [1] "char_freq_..4"              "word_freq_remove"
##  [3] "char_freq_..3"              "word_freq_free"
##  [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
```

We can see from our output(summary()) that there are a total of 149 leaf nodes in this tree and that there are 49 training observaitons that are misclassified

(4)

```r
draw.tree(prune.tree(spamtree, best = 10), nodeinfo = TRUE, cex = 0.5)
```



(5)

```r
spam.cv = cv.tree(spamtree, rand = folds, FUN = prune.misclass, K = 10)
spam.cv$size
```

```
##  [1] 149 106 102  99  76  73  63  59  52  46  41  38  35  24  22  16  15  14  13
## [20]   9   8   7   6   5   4   3   2   1
```

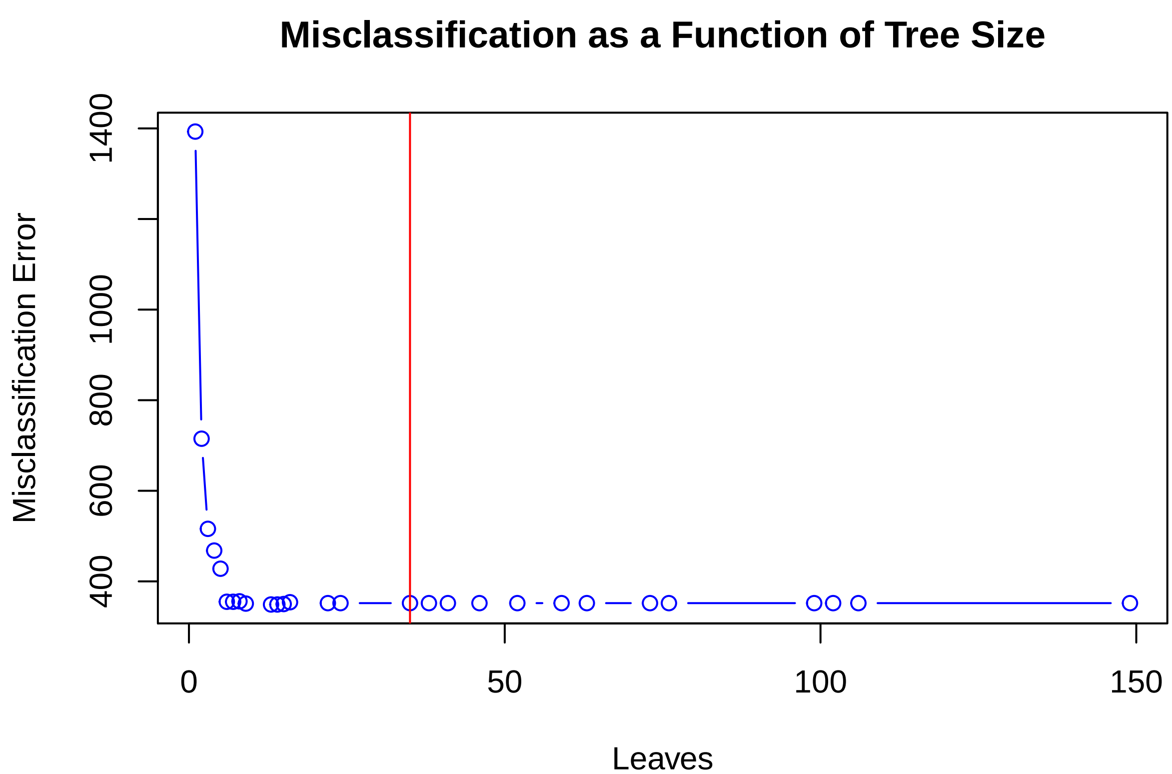
```r
spam.cv$dev
```

```
##  [1]  352  352  352  352  352  352  352  352  352  352  352  352  352  352  352
## [16]  354  350  349  349  351  356  355  355  428  468  516  715 1393
```

```r
best.size.cv = spam.cv$size[which.min(spam.cv$dev)]
best.size.cv
```

```
## [1] 14
```

```r
plot(spam.cv$size, spam.cv$dev, type = "b", xlab = "Leaves", ylab = "Misclassification Error", main = "
abline(v = 35, col = "red")
```



We know from the dev that after the 13th deviation the numbers go up from 353 to 355 and then increases. We set the abline to the 13th value of the size which is 35. This means that the optimal tree size is 35.

(6)

```r
#training error
spamtree.pruned = prune.misclass(spamtree, best = 35)
pred.train.tree = predict(spamtree.pruned, type = "class")
train.error.tree = calc_error_rate(pred.train.tree, YTrain)
train.error.tree
```

```
## [1] 0.04582061
```

```r
#testing error
pred.test.tree = predict(spamtree.pruned, type = "class")
test.error.tree = calc_error_rate(pred.test.tree, YTest)
```

```
## Warning in `!=.default`(true.value, predicted.value): longer object length is
## not a multiple of shorter object length
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```r
test.error.tree
```

```
## [1] 0.490697
```

```r
records[2,2] = test.error.tree
records[2,1] = train.error.tree
records
```

```
##          train_error test_error
## knn       0.01916134   0.033000
## tree      0.04582061   0.490697
## logistic          NA         NA
```