PSTAT 131 Homework Assignment 2

Marissa Santiago and Leticia Cruz

May 02, 2021

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.0     v dplyr   1.0.5
## v tidyr   1.1.3     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()
```

```r
library(tree)
```

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

```r
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
```

```r
library(dplyr)
library(class)
library(rpart)
library(maptree)
```

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

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

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

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

```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(y = factor(y, levels=c(0,1), labels=c("good", "spam"))) %>%
  mutate_at(.vars=vars(-y), .funs=scale)
```

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

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

Training/test sets: Split randomly the data set in a train and a test set:

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

Folds for cv

```r
nfold = 10
set.seed(1)
folds = seq.int(nrow(spam.train)) %>%
  cut(breaks = nfold, labels=FALSE) %>%
  sample
```

Problem 1

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

```r
#check for missing values
sum(is.na(spam.train))
```

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

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

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

```r
YTrain = spam.train$y %>% na.omit()
XTrain = spam.train %>% select(-y) %>% na.omit()
YTest = spam.test$y %>% na.omit()
XTest = spam.test %>% select(-y) %>% na.omit()

error.folds <- NULL
set.seed(1)

kvec = c(1, seq(10, 50, length.out=5))
for (i in kvec){
  tmp <- ldply(1:nfold, do.chunk,
               folddef = folds, Xdat = XTrain,
               Ydat = YTrain, k = i)
  tmp$neighbors <- i
  error.folds <- rbind(error.folds,tmp)
}
error.folds
```

```
##    fold train.error val.error neighbors
## 1     1   0.0006173   0.11080         1
## 2     2   0.0000000   0.11944         1
## 3     3   0.0006171   0.08056         1
## 4     4   0.0000000   0.08056         1
## 5     5   0.0006171   0.10833         1
## 6     6   0.0006171   0.11111         1
## 7     7   0.0003085   0.07778         1
## 8     8   0.0000000   0.11667         1
## 9     9   0.0003085   0.10000         1
## 10   10   0.0003085   0.13056         1
## 11    1   0.0824074   0.08864        10
## 12    2   0.0823820   0.11111        10
## 13    3   0.0805307   0.08889        10
## 14    4   0.0774452   0.10000        10
## 15    5   0.0755940   0.09722        10
## 16    6   0.0762110   0.10278        10
## 17    7   0.0805307   0.05833        10
## 18    8   0.0789880   0.09444        10
## 19    9   0.0759025   0.11111        10
## 20   10   0.0786794   0.11389        10
## 21    1   0.0919753   0.09418        20
## 22    2   0.0944153   0.11944        20
## 23    3   0.0956495   0.08056        20
## 24    4   0.0934897   0.08889        20
## 25    5   0.0888615   0.12500        20
## 26    6   0.0882444   0.11111        20
## 27    7   0.0965751   0.06944        20
## 28    8   0.0907127   0.10556        20
## 29    9   0.0931811   0.12778        20
## 30   10   0.0910213   0.10000        20
## 31    1   0.0993827   0.10249        30
## 32    2   0.1024375   0.12500        30
## 33    3   0.1052144   0.10000        30
## 34    4   0.1030546   0.10556        30
## 35    5   0.0993521   0.11667        30
## 36    6   0.0984264   0.10833        30
## 37    7   0.1033632   0.07778        30
## 38    8   0.0971922   0.12778        30
## 39    9   0.1012033   0.11944        30
## 40   10   0.0990435   0.10833        30
## 41    1   0.1055556   0.11357        40
## 42    2   0.1058315   0.11667        40
## 43    3   0.1104597   0.11111        40
## 44    4   0.1052144   0.10556        40
## 45    5   0.1101512   0.12222        40
## 46    6   0.1073743   0.12222        40
## 47    7   0.1098426   0.08056        40
## 48    8   0.1021290   0.13056        40
## 49    9   0.1098426   0.12500        40
## 50   10   0.1033632   0.09722        40
## 51    1   0.1111111   0.11080        50
## 52    2   0.1129281   0.12222        50
## 53    3   0.1135452   0.11389        50
```

```
## 54    4   0.1110768   0.11111        50
## 55    5   0.1104597   0.12222        50
## 56    6   0.1116939   0.11944        50
## 57    7   0.1141623   0.08056        50
## 58    8   0.1110768   0.14722        50
## 59    9   0.1082999   0.12222        50
## 60   10   0.1089170   0.10556        50
```

```
#Transform the format of error.folds for further convenience
errors = melt(error.folds, id.vars=c('fold', 'neighbors'),value.name='error')

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_()` was deprecated in dplyr 0.7.0.
## Please use `across()` instead.
```

```
## Warning: `funs()` was deprecated in 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))
```

```
val.error.means
```

```
## # A tibble: 1 x 3
##   neighbors variable   error
##       <dbl> <fct>      <dbl>
## 1        10 val.error 0.0966
```

```
best.kfold = max(val.error.means$neighbors)
best.kfold
```

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

Problem 2

```r
set.seed(1)
#training error rate
pred.YTrain = knn(train = XTrain, test = XTrain, cl = YTrain, k = best.kfold)
train_error = calc_error_rate(pred.YTrain,YTrain)

#test error rate
pred.YTest = knn( train = XTrain, test = XTest, cl = YTrain, k = best.kfold)
test_error = calc_error_rate(pred.YTest, YTest)

records[1,1]=train_error
records[1,2]=test_error
records
```

```
##          train.error test.error
## knn          0.07803      0.102
## tree              NA         NA
## logistic          NA         NA
```

Problem 3

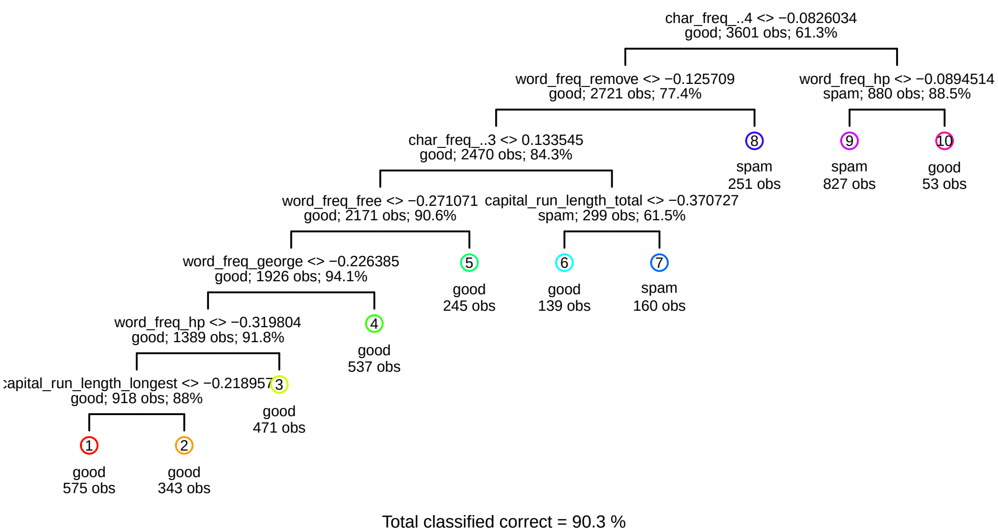
```r
spamtree = tree(y ~ ., data = spam.train,
                control = tree.control(nrow(spam.train), minsize = 5, mindev = 1e-5))
summary(spamtree)
```

```
##
## Classification tree:
## tree(formula = y ~ ., data = spam.train, control = tree.control(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:  0.0457 = 158 / 3450
## Misclassification error rate: 0.0136 = 49 / 3601
```

There is a total of 149 leaf nodes in this tree and there are 49 training observations that are misclassified.

Problem 4

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



Problem 5

```r
set.seed(1)
cv = cv.tree(spamtree, rand = folds, FUN = prune.misclass, K = 10)
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
##
## $dev
##  [1]  353  353  346  346  346  346  346  346  346  346  346  342  342  342  342
## [16]  344  344  343  343  347  354  352  352  410  463  516  715 1393
##
## $k
##  [1]     -Inf   0.0000   0.5000   0.6667   1.0000   1.3333   1.5000   1.7500
```

```
##  [9]   2.0000   2.5000   2.8000   3.0000   3.6667   4.0000   4.5000   5.1667
## [17]   6.0000   7.0000   8.0000   9.7500  11.0000  12.0000  17.0000  45.0000
## [25]  53.0000  69.0000 199.0000 678.0000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

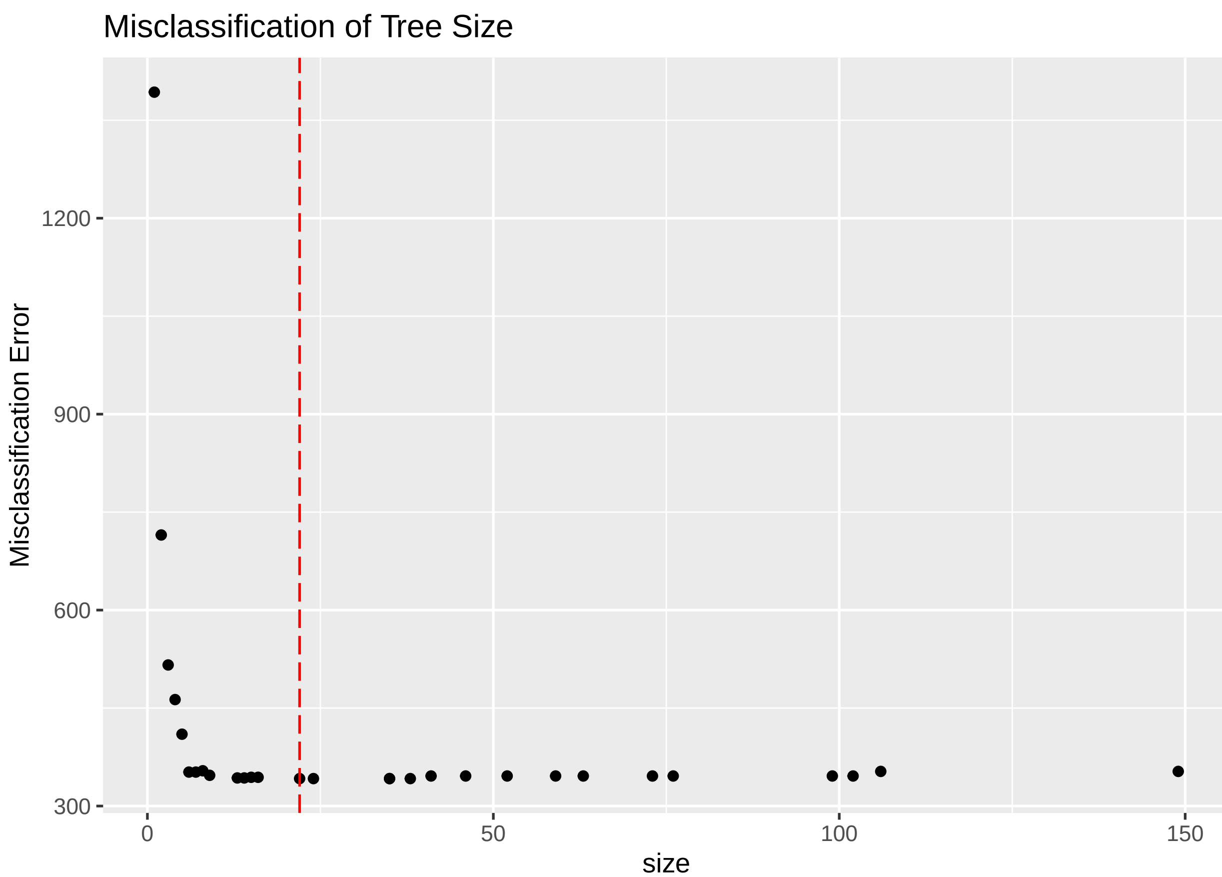
```
tree_data <- data.frame("x"=cv$size,"y"=cv$dev)
tree_data
```

```
##      x    y
## 1  149  353
## 2  106  353
## 3  102  346
## 4   99  346
## 5   76  346
## 6   73  346
## 7   63  346
## 8   59  346
## 9   52  346
## 10  46  346
## 11  41  346
## 12  38  342
## 13  35  342
## 14  24  342
## 15  22  342
## 16  16  344
## 17  15  344
## 18  14  343
## 19  13  343
## 20   9  347
## 21   8  354
## 22   7  352
## 23   6  352
## 24   5  410
## 25   4  463
## 26   3  516
## 27   2  715
## 28   1 1393
```

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

```
## [1] 22
```

```
ggplot(tree_data,aes(x,y)) +geom_point() + geom_point(data=tree_data[13,],aes(x,y)) + geom_vline(xinterc
 ggtitle("Misclassification of Tree Size")+xlab("size")+ylab("Misclassification Error")
```



The optimal tree size is 22.

Problem 6

```
set.seed(1)
#training error
spamtree.pruned = prune.misclass(spamtree, best =  best.size.cv)

pred.train = predict(spamtree.pruned, spam.train, type = "class")
#testing error
pred.test = predict(spamtree.pruned,spam.test, type = "class")

train.error = calc_error_rate(pred.train, YTrain)
test.error = calc_error_rate(pred.test, YTest)

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

```
##          train.error test.error
## knn          0.07803      0.102
## tree         0.06054      0.091
## logistic          NA         NA
```

Logistics Regression

Problem 7

7a

Given,

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

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

7b

#

$$p = \frac{e^{\beta_0+\beta_1 x_1}}{1+e^{\beta_0+\beta_1 x_1}}$$

When $x_1 = x_1 + 2$:

#

$$p = \frac{e^{\beta_0+\beta_1(x_1+2)}}{1+e^{\beta_0+\beta_1(x_1+2)}} = \frac{e^{\beta_0+\beta_1 x_1+2\beta_1}}{1+e^{\beta_0+\beta_1 x_1+2\beta_1}} = \frac{e^{\beta_0+\beta_1 x_1}e^{2\beta_1}}{1+e^{\beta_0+\beta_1 x_1}e^{2\beta_1}}$$

As x increases by 2, the odds is multiplied by $e^{2\beta_1}$.

#

$$\lim_{x\to\infty} p = 0$$

Also, as x goes to infinity, the numerator becomes smaller and the denominator becomes bigger. Therefore, the probability gets closer to 0.

#

$$\lim_{x\to-\infty} p = 1$$

As x goes to negative infinity, the probability goes to 1.

Problem 8

```r
set.seed(1)
#fit logistisic regression
glm.fit = glm(y~.,data=spam.train, family=binomial)
```

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

```r
summary(glm.fit)
```

```
##
## Call:
## glm(formula = y ~ ., family = binomial, data = spam.train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -3.812  -0.198   0.000   0.119   5.551
##
## Coefficients:
##                            Estimate Std. Error z value Pr(>|z|)
## (Intercept)               -1.64e+01   1.98e+02   -0.08  0.93399
## word_freq_make            -1.14e-01   7.73e-02   -1.48  0.13990
## word_freq_address         -1.58e-01   9.42e-02   -1.68  0.09270 .
## word_freq_all              8.79e-02   6.13e-02    1.43  0.15175
## word_freq_3d               3.66e+00   2.41e+00    1.52  0.12789
## word_freq_our              4.83e-01   8.03e-02    6.02  1.7e-09 ***
## word_freq_over             2.91e-01   8.18e-02    3.56  0.00037 ***
## word_freq_remove           7.89e-01   1.30e-01    6.08  1.2e-09 ***
## word_freq_internet         1.92e-01   6.55e-02    2.94  0.00331 **
## word_freq_order            1.57e-01   8.67e-02    1.81  0.07053 .
## word_freq_mail             5.91e-02   4.78e-02    1.24  0.21638
## word_freq_receive         -4.91e-02   6.57e-02   -0.75  0.45412
## word_freq_will            -1.25e-01   7.34e-02   -1.70  0.08919 .
## word_freq_people          -2.35e-03   8.05e-02   -0.03  0.97673
## word_freq_report           1.46e-02   5.20e-02    0.28  0.77940
## word_freq_addresses        3.00e-01   1.83e-01    1.64  0.10177
## word_freq_free             8.92e-01   1.33e-01    6.70  2.1e-11 ***
## word_freq_business         3.53e-01   1.03e-01    3.42  0.00063 ***
## word_freq_email            9.84e-02   6.69e-02    1.47  0.14153
## word_freq_you              1.28e-01   6.95e-02    1.84  0.06519 .
## word_freq_credit           5.14e-01   3.12e-01    1.65  0.09904 .
## word_freq_your             2.61e-01   6.92e-02    3.77  0.00017 ***
## word_freq_font             3.17e-01   2.30e-01    1.38  0.16857
## word_freq_000              8.18e-01   1.85e-01    4.42  9.9e-06 ***
## word_freq_money            1.99e-01   7.42e-02    2.69  0.00721 **
## word_freq_hp              -3.36e+00   6.06e-01   -5.54  3.0e-08 ***
## word_freq_hpl             -7.02e-01   3.93e-01   -1.79  0.07410 .
## word_freq_george          -4.13e+01   8.45e+00   -4.88  1.0e-06 ***
## word_freq_650              2.67e-01   1.87e-01    1.43  0.15378
## word_freq_lab             -1.23e+00   8.37e-01   -1.46  0.14319
## word_freq_labs            -1.80e-01   1.73e-01   -1.04  0.29990
## word_freq_telnet          -4.72e-02   1.50e-01   -0.31  0.75357
```

```
## word_freq_857              -2.52e+01   1.38e+03   -0.02  0.98547
## word_freq_data             -5.96e-01   2.19e-01   -2.72  0.00648 **
## word_freq_415               3.96e-01   5.80e-01    0.68  0.49460
## word_freq_85               -1.08e+00   4.54e-01   -2.38  0.01710 *
## word_freq_technology        2.58e-01   1.43e-01    1.80  0.07114 .
## word_freq_1999              4.45e-02   8.14e-02    0.55  0.58481
## word_freq_parts             3.71e-01   2.13e-01    1.74  0.08205 .
## word_freq_pm               -2.81e-01   1.95e-01   -1.44  0.15065
## word_freq_direct           -1.12e-01   1.33e-01   -0.84  0.39891
## word_freq_cs               -1.68e+01   9.60e+00   -1.75  0.08067 .
## word_freq_meeting          -2.45e+00   8.55e-01   -2.87  0.00414 **
## word_freq_original         -1.57e-01   1.62e-01   -0.97  0.33165
## word_freq_project          -1.14e+00   3.94e-01   -2.88  0.00397 **
## word_freq_re               -7.10e-01   1.54e-01   -4.60  4.2e-06 ***
## word_freq_edu              -1.21e+00   2.59e-01   -4.68  2.9e-06 ***
## word_freq_table            -1.10e-01   1.42e-01   -0.78  0.43734
## word_freq_conference       -1.31e+00   5.56e-01   -2.35  0.01894 *
## char_freq_.                -4.15e-01   1.55e-01   -2.68  0.00739 **
## char_freq_..1              -3.96e-02   8.33e-02   -0.48  0.63461
## char_freq_..2              -6.59e-02   1.16e-01   -0.57  0.56942
## char_freq_..3               1.97e-01   5.54e-02    3.56  0.00037 ***
## char_freq_..4               1.08e+00   1.83e-01    5.89  3.8e-09 ***
## char_freq_..5               1.22e+00   4.99e-01    2.45  0.01443 *
## capital_run_length_average  3.17e-01   6.62e-01    0.48  0.63224
## capital_run_length_longest  1.79e+00   5.60e-01    3.20  0.00139 **
## capital_run_length_total    7.14e-01   1.53e-01    4.68  2.9e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4806.0  on 3600  degrees of freedom
## Residual deviance: 1413.2  on 3543  degrees of freedom
## AIC: 1529
##
## Number of Fisher Scoring iterations: 22
```

```r
#test
prob.training=predict(glm.fit,type="response")
prob.test=predict(glm.fit,newdata=spam.test,type="response")

#save the predicted labels
spamtrain = spam.train%>%
  mutate(predspamtrain=as.factor(ifelse(prob.training<=0.5,"good","spam")))
spamtest = spam.test%>%
  mutate(predspamtest=as.factor(ifelse(prob.test<=0.5,"good","spam")))
d<-calc_error_rate(spamtrain$predspamtrain,YTrain)
e<-calc_error_rate(spamtest$predspamtest,YTest)
records[3,1]=d
records[3,2]=e
records
```

```
##          train.error test.error
## knn          0.07803      0.102
```

```
## tree          0.06054       0.091
## logistic      0.06804       0.086
```

The method with the lowest misclassification error is decision tree method.

Problem 9

We take "positive" here to mean "spam." We would be more concerned with the false positive rate being large. In this case our model would not be correctly classifying the email as "spam" which means that all those spam emails will end up in our inbox; leading to a mixture of spam emails with legitimate emails. With a low true positive rate, the spam email would get filtered into where it belongs so we wouldn't be too concerned with this. The user can always delete spam from their regualar inbox , but cannot easily recover or notice good emails being placed in the spam folder.