PSTAT131_HW2

YuboWei_&HaozeZhu

5/2/2021

Load library and read dataset

```
library(tidyverse)
```

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

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.1     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()
```

```
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)
rm(list = ls())

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)
```

```
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)) %>% cut(breaks = nfold, labels=FALSE) %>% sample
```

K-Nearest Neighbor Method

1. (Selecting number of neighbors)

```
## split train and test data set into two set
YTrain = spam.train$y
XTrain = spam.train %>% select(-y)
XTest = spam.test %>% select(-y)
YTest = spam.test$y
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))
}
kvec = c(1, seq(10, 50, length.out=5))
error.folds = NULL
# Loop through different number of neighbors
for (i in kvec){
tmp = ldply(1:nfold, do.chunk, folddef=folds, Xdat=XTrain, Ydat=YTrain, k=i) # Necessary arguments to b
            tmp$neighbors = i # Keep track of each value of neighors
            error.folds = rbind(error.folds, tmp) # combine results
}
## get mean val.error in each neighbors
x <- data.frame(matrix(ncol = 2, nrow = 0))
colnames(x) <- c('neighbors', 'mean.val.error')
for (i in kvec){
  errors = error.folds %>% filter(neighbors== i) %>% summarise(mean.val.error = mean(val.error))
  x[nrow(x) + 1,] = c(i,errors)
}
# Best number of neighbors
neighbors = x[which.min(x$mean.val.error),1]
neighbors
```

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

2. Training and Test Errors of knn fit

```
## get classifications for current training chunks
predYtr = knn(train = XTrain, test = XTrain, cl = YTrain, k = 10)
## get classifications for current test chunk
predYvl = knn(train = XTrain, test = XTest, cl = YTrain, k = 10)

knn_train_error = calc_error_rate(predYtr, YTrain)
knn_test_error = calc_error_rate(predYvl, YTest)
## Fill in the rst row of records with the train and test error from the knn  t.
records[1,1] = knn_train_error
records[1,2] = knn_test_error
records
```

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

Decision Tree Method

3.(Controlling Decision Tree Construction)

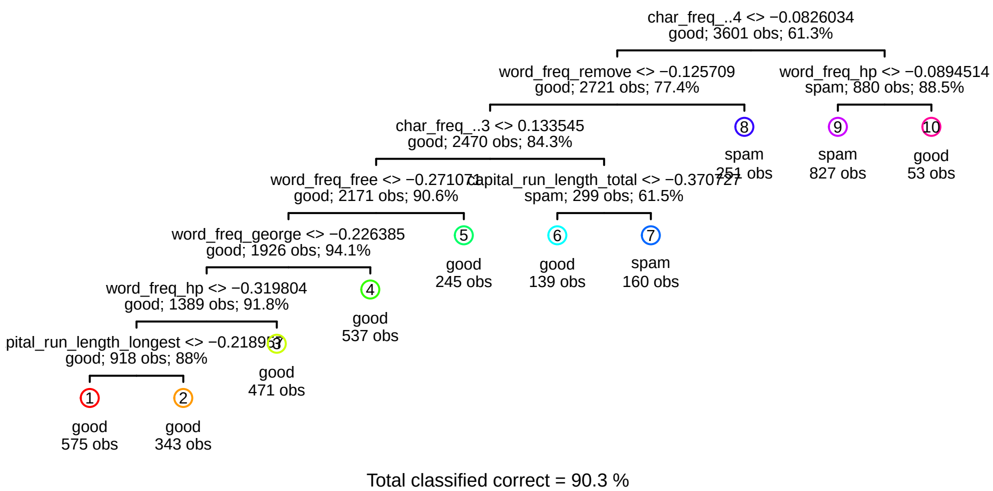
```
spamtree = tree(y~.,control = tree.control(nobs=(nrow(spam.train)), minsize = 5,mindev = 1e-5),data = sp
summary(spamtree)
```

```
##
## 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 in this tree, and 49 training observations are misclassified.

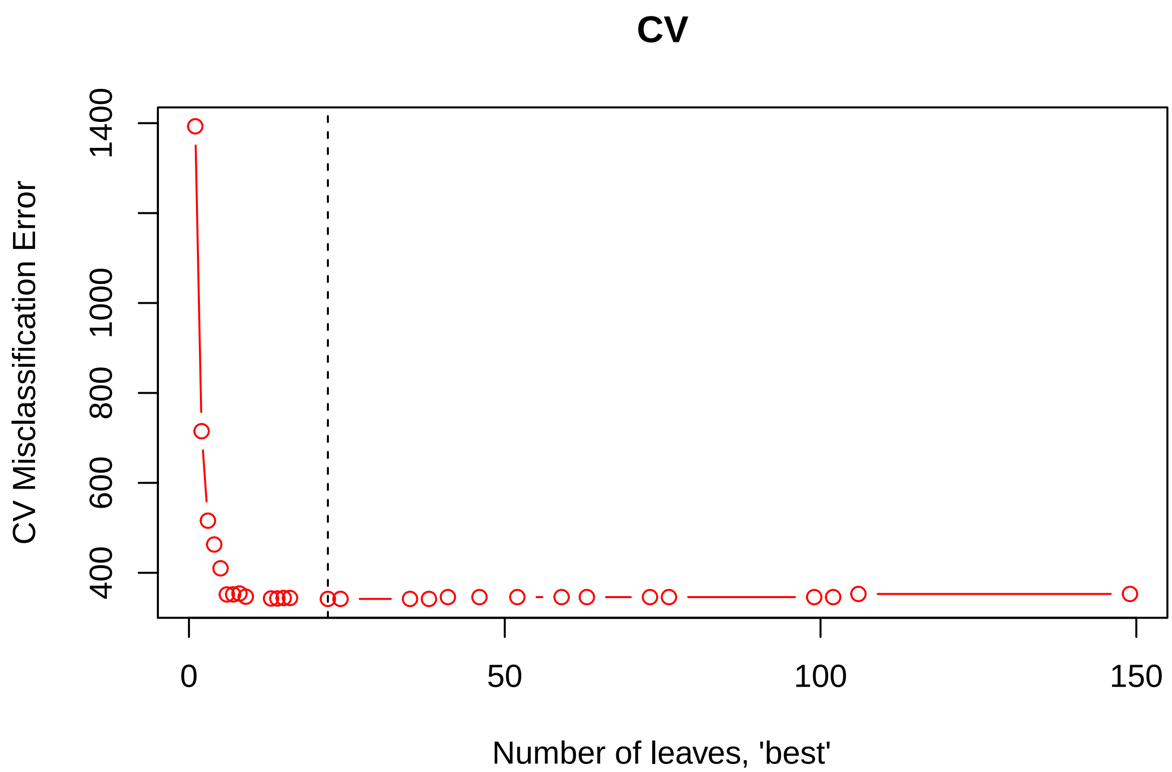
4. (Decision Tree Pruning)

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



5. Use cross validation to prune the tree

```r
# Set random seed
set.seed(1)
cv = cv.tree(spamtree, rand = folds, FUN=prune.misclass, K=10)
best.size.cv = 22
# Plot size vs. cross-validation error rate
plot(cv$size , cv$dev, type="b",
     xlab = "Number of leaves, \'best\'", ylab = "CV Misclassification Error",
     col = "red", main="CV")
  abline(v=best.size.cv, lty=2)
```



6. Training and Test Errors of pruned tree fit

```r
# prune the original tree using the best size in 5
spamtree.pruned = prune.misclass(spamtree, best=best.size.cv)
## get classifications for current training chunks
tree.pred.train = predict(spamtree.pruned, spam.train, type="class")
## get classifications for current test chunk
tree.pred.test = predict(spamtree.pruned, spam.test, type="class")
## get train error rate and test error rate
tree_train_error = calc_error_rate(tree.pred.train, YTrain)
tree_test_error = calc_error_rate(tree.pred.test, YTest)
## record the error rates
records[2,1] = tree_train_error
records[2,2] = tree_test_error
```

Logistic regression

7.a

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

7.b Assume that $z = \beta_0 + \beta_1 x_1$, and p = logistic(z). If we increase $x1$ by 2, we will increase the odds of the outcome multiplicatively by $e^{2\beta_1}$.

$$p = \text{logit}^{-1}(z) = \frac{e^z}{1+e^z} = \frac{e^{\beta_0+\beta_1 x_1}}{1+e^{\beta_0+\beta_1 x_1}}$$

Since we assume β_1 is negative, as $x_1 \to \infty, \beta_1 x_1 \to -\infty$. Therefore,

$$\lim_{x\to\infty} \frac{e^{-\infty}}{1+e^{-\infty}} = \frac{0}{1} = 0$$

The probability converges to 0 . Also,

$$\lim_{x\to-\infty} \frac{e^{\beta_0+\beta_1 x_1}}{1+e^{\beta_0+\beta_1 x_1}} = \lim_{x\to\infty} \frac{e^{\beta_0+|\beta_1 x_1|}}{1+e^{\beta_0+|\beta_1 x_1|}}$$

and, by L'Hopital's rule, the probability converges to 1.

8.Use logistic regression to perform classification

```
## build model using logistic regression
glm.fit = glm(y ~ ., data=spam.train, family=binomial)
```

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

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

```
##
## Call:
## glm(formula = y ~ ., family = binomial, data = spam.train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.8117  -0.1976   0.0000   0.1195   5.5509
##
## Coefficients:
##                              Estimate Std. Error z value Pr(>|z|)
## (Intercept)                -1.639e+01  1.980e+02  -0.083 0.933994
## word_freq_make             -1.141e-01  7.727e-02  -1.476 0.139896
```

```
## word_freq_address          -1.584e-01  9.420e-02  -1.681 0.092696 .
## word_freq_all               8.788e-02  6.131e-02   1.433 0.151749
## word_freq_3d                3.663e+00  2.406e+00   1.522 0.127893
## word_freq_our               4.835e-01  8.025e-02   6.024 1.70e-09 ***
## word_freq_over              2.911e-01  8.179e-02   3.559 0.000372 ***
## word_freq_remove            7.891e-01  1.299e-01   6.076 1.23e-09 ***
## word_freq_internet          1.925e-01  6.553e-02   2.938 0.003308 **
## word_freq_order             1.568e-01  8.669e-02   1.808 0.070530 .
## word_freq_mail              5.907e-02  4.779e-02   1.236 0.216382
## word_freq_receive          -4.915e-02  6.565e-02  -0.749 0.454116
## word_freq_will             -1.247e-01  7.340e-02  -1.700 0.089186 .
## word_freq_people           -2.347e-03  8.046e-02  -0.029 0.976728
## word_freq_report            1.457e-02  5.203e-02   0.280 0.779401
## word_freq_addresses         3.000e-01  1.834e-01   1.636 0.101773
## word_freq_free              8.924e-01  1.332e-01   6.701 2.06e-11 ***
## word_freq_business          3.534e-01  1.034e-01   3.416 0.000635 ***
## word_freq_email             9.839e-02  6.692e-02   1.470 0.141525
## word_freq_you               1.281e-01  6.949e-02   1.844 0.065191 .
## word_freq_credit            5.144e-01  3.119e-01   1.650 0.099038 .
## word_freq_your              2.605e-01  6.916e-02   3.767 0.000165 ***
## word_freq_font              3.171e-01  2.303e-01   1.377 0.168568
## word_freq_000               8.184e-01  1.852e-01   4.420 9.86e-06 ***
## word_freq_money             1.993e-01  7.418e-02   2.687 0.007206 **
## word_freq_hp               -3.355e+00  6.057e-01  -5.540 3.03e-08 ***
## word_freq_hpl              -7.025e-01  3.933e-01  -1.786 0.074096 .
## word_freq_george           -4.126e+01  8.450e+00  -4.883 1.05e-06 ***
## word_freq_650               2.672e-01  1.873e-01   1.426 0.153778
## word_freq_lab              -1.225e+00  8.369e-01  -1.464 0.143189
## word_freq_labs             -1.797e-01  1.733e-01  -1.037 0.299902
## word_freq_telnet           -4.722e-02  1.504e-01  -0.314 0.753565
## word_freq_857              -2.517e+01  1.382e+03  -0.018 0.985470
## word_freq_data             -5.964e-01  2.191e-01  -2.722 0.006483 **
## word_freq_415               3.964e-01  5.804e-01   0.683 0.494604
## word_freq_85               -1.083e+00  4.543e-01  -2.385 0.017096 *
## word_freq_technology        2.583e-01  1.431e-01   1.805 0.071139 .
## word_freq_1999              4.449e-02  8.144e-02   0.546 0.584808
## word_freq_parts             3.706e-01  2.131e-01   1.739 0.082046 .
## word_freq_pm               -2.806e-01  1.952e-01  -1.437 0.150654
## word_freq_direct           -1.120e-01  1.328e-01  -0.844 0.398907
## word_freq_cs               -1.677e+01  9.600e+00  -1.747 0.080673 .
## word_freq_meeting          -2.451e+00  8.549e-01  -2.867 0.004144 **
## word_freq_original         -1.572e-01  1.619e-01  -0.971 0.331654
## word_freq_project          -1.136e+00  3.944e-01  -2.880 0.003971 **
## word_freq_re               -7.105e-01  1.544e-01  -4.601 4.21e-06 ***
## word_freq_edu              -1.211e+00  2.588e-01  -4.678 2.89e-06 ***
## word_freq_table            -1.101e-01  1.418e-01  -0.777 0.437339
## word_freq_conference       -1.305e+00  5.562e-01  -2.347 0.018938 *
## char_freq_.                -4.146e-01  1.548e-01  -2.679 0.007394 **
## char_freq_..1              -3.958e-02  8.328e-02  -0.475 0.634607
## char_freq_..2              -6.594e-02  1.159e-01  -0.569 0.569422
## char_freq_..3               1.973e-01  5.540e-02   3.561 0.000369 ***
## char_freq_..4               1.075e+00  1.825e-01   5.891 3.84e-09 ***
## char_freq_..5               1.221e+00  4.990e-01   2.446 0.014434 *
## capital_run_length_average  3.169e-01  6.621e-01   0.479 0.632243
```

```
## capital_run_length_longest  1.791e+00  5.603e-01   3.196 0.001392 **
## capital_run_length_total    7.143e-01  1.528e-01   4.675 2.94e-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: 1529.2
##
## Number of Fisher Scoring iterations: 22
```

```
## get tranning error rate
prob.training = predict(glm.fit, type="response")
log_pred_train=as.factor(ifelse(prob.training<=0.5, "good", "spam"))
log_train_error = calc_error_rate(log_pred_train, YTrain)
## get test error rate
prob.test = predict(glm.fit,spam.test, type="response")
log_pred_test=as.factor(ifelse(prob.test<=0.5, "good", "spam"))
log_test_error = calc_error_rate(log_pred_test, YTest)
## record the error rates
records[3,1] = log_train_error
records[3,2] = log_test_error
records
```

```
##          train.error test.error
## knn       0.07803388      0.103
## tree      0.06053874      0.091
## logistic  0.06803666      0.086
```

logistic regression method had the lowest misclassification error on the test set

9. If I am the designer of a spam filter,I will be more concerned about the potential for false positive rates that are too large than true positive rates that are too small. A false positive rates that are too large means some important emails are listed as spam emails and filered by the algorithm. This would casue more damage to the users than a few spam emails passes through the algorithm.