Solutions to HW3

Shaoyi Zhang

April 30th, 2016

Question 1

```
# set up data frame
setwd("/Users/Shawn/Desktop/PSTAT 231/PSTAT-231/assign3")
spam = read.table("spambase.dat",header=T,sep="")
#summary(spam)
spam$y = factor(spam$y,levels=c(0,1),labels=c("good","spam"))

# partition the data set
# train set size = sample size - 1000
# test set size = 1000
train_size <- floor(nrow(spam)-1000)

# set the seed to make your partition reproductible
set.seed(1)
train_index <- sample(seq_len(nrow(spam)), size = train_size)

train <- spam[train_index, ]
test <- spam[-train_index, ]
```

Now, we can start build the decision tree

```
require(tree)
```

```
## Loading required package: tree
```

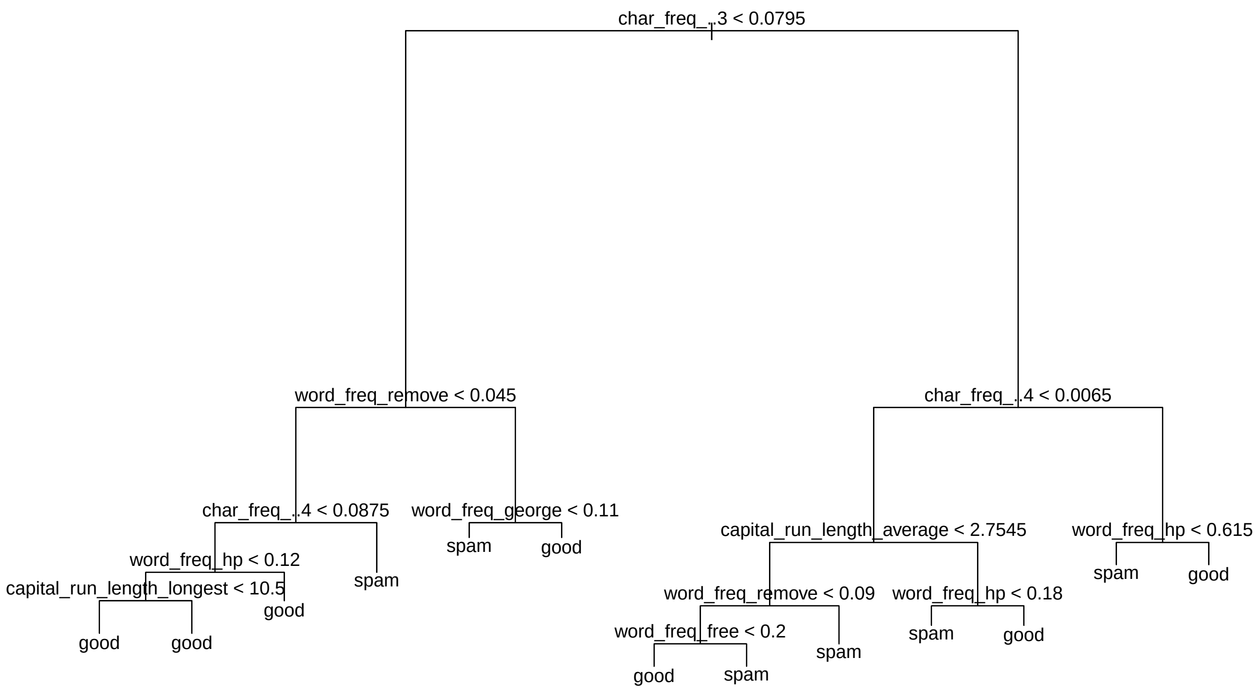
```
spam.tree = tree(y~.,data=train)

cv.tree(spam.tree,FUN=prune.misclass)
```

```
## $size
##  [1] 13 11 10  9  8  7  6  5  3  2  1
##
## $dev
##  [1]  343  343  356  356  384  396  417  509  690  721 1420
##
## $k
##  [1] -Inf    0   11   12   17   19   22   55   94   98  674
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

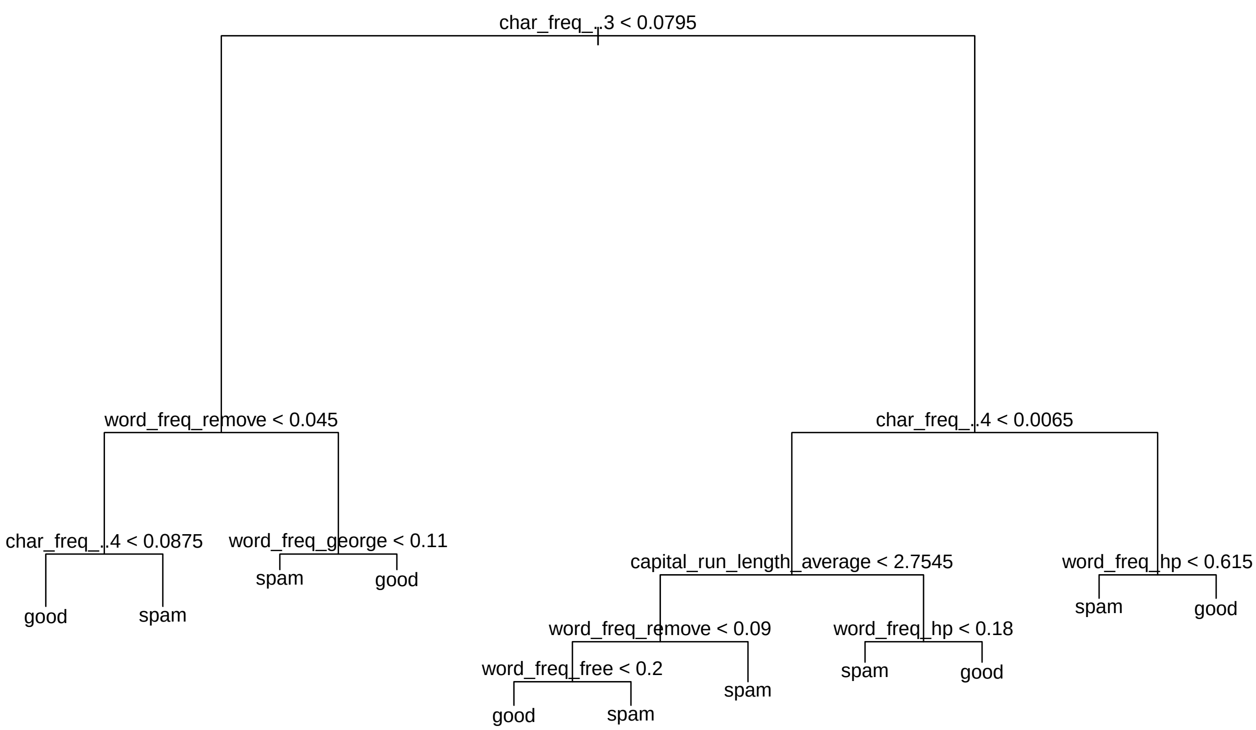
The optimal tree size is 11

```r
prune.spam.tree = prune.misclass(spam.tree,best = 11)
```

unpruned decision tree without option



pruned decision tree without option



```r
# make prediction on test set(unpruned)
spam.tree.pred = predict(spam.tree,test,type="class")
conti.table = table(spam.tree.pred,test$y)

# make prediction on test set(pruned)
prune.pred = predict(prune.spam.tree,test,type="class")
prune.conti.table = table(prune.pred,test$y)

# construct error rate vector
test.error.rates = vector()
model.index = 1

# compute the test error rate
test.error.rates[model.index] = (prune.conti.table[3] + prune.conti.table[2])/nrow(test)
model.index = model.index + 1
test.error.rates
```

```
## [1] 0.114
```

Then, let's try a decision tree with options

```r
# decision tree with option
spam.tree.option = tree(y~.,data=train,control=tree.control(nrow(spam),mincut=2,minsize=5,mindev=0.001))
cv.tree(spam.tree.option,FUN=prune.misclass)
```

```
## $size
##  [1] 133 116 112 110  77  72  67  64  46  43  41  32  30  27  25  19  13
## [18]  11  10   9   8   7   6   5   3   2   1
##
## $dev
##  [1]  341  341  341  341  341  341  341  341  341  341  341  341  338  338
## [15]  338  338  338  349  348  380  385  388  399  485  694  694 1420
##
## $k
##  [1]       -Inf   0.000000   0.250000   0.500000   1.000000   1.200000
##  [7]   1.600000   1.666667   2.000000   2.333333   2.500000   3.000000
## [13]   3.500000   4.000000   5.000000   5.500000   6.000000  11.000000
## [19]  12.000000  17.000000  18.000000  19.000000  22.000000  55.000000
## [25]  94.000000  98.000000 674.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```r
# the optimal tree size is 91
prune.spam.option = prune.misclass(spam.tree.option,best = 91)


spam.option.pred = predict(prune.spam.option,test,type="class")
conti.table = table(spam.option.pred,test$y)
```

```r
test.error.rates[model.index] = (conti.table[3] + conti.table[2])/nrow(test)
model.index = model.index + 1
test.error.rates
```

```
## [1] 0.114 0.080
```

Tree bagging model

```r
#install.packages("randomForest")
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```r
set.seed(1)
bag.spam = randomForest(y~.,data=train,mtry=(ncol(train)-1),importance=T)
bag.spam
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = train, mtry = (ncol(train) -      1), importance = T)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 57
##
##         OOB estimate of  error rate: 5.64%
## Confusion matrix:
##      good spam class.error
## good 2088   93  0.04264099
## spam  110 1310  0.07746479
```

```r
bag.pred = predict(bag.spam,test,type="class")
bag.conti.table = table(bag.pred,test$y)

test.error.rates[model.index] = (bag.conti.table[3] + bag.conti.table[2])/nrow(test)
model.index = model.index + 1
test.error.rates
```

```
## [1] 0.114 0.080 0.056
```

Random Forest model

```r
set.seed(1)
rand.spam = randomForest(y~.,data=train,mtry=floor(sqrt(ncol(test)-1)),importance=T)
rand.spam
```

```
##
## Call:
```

```
##  randomForest(formula = y ~ ., data = train, mtry = floor(sqrt(ncol(test) -      1)), importance = T)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 4.64%
## Confusion matrix:
##      good spam class.error
## good 2118   63  0.02888583
## spam  104 1316  0.07323944
```

```
rand.pred = predict(rand.spam,test,type="class")
rand.conti.table = table(rand.pred,test$y)

test.error.rates[model.index] = (conti.table[3] + conti.table[2])/nrow(test)
model.index = model.index + 1
test.error.rates
```

```
## [1] 0.114 0.080 0.056 0.080
```

k-NN classification

```
require(class)
```

```
## Loading required package: class
```

```
require(boot)
```

```
## Loading required package: boot
```

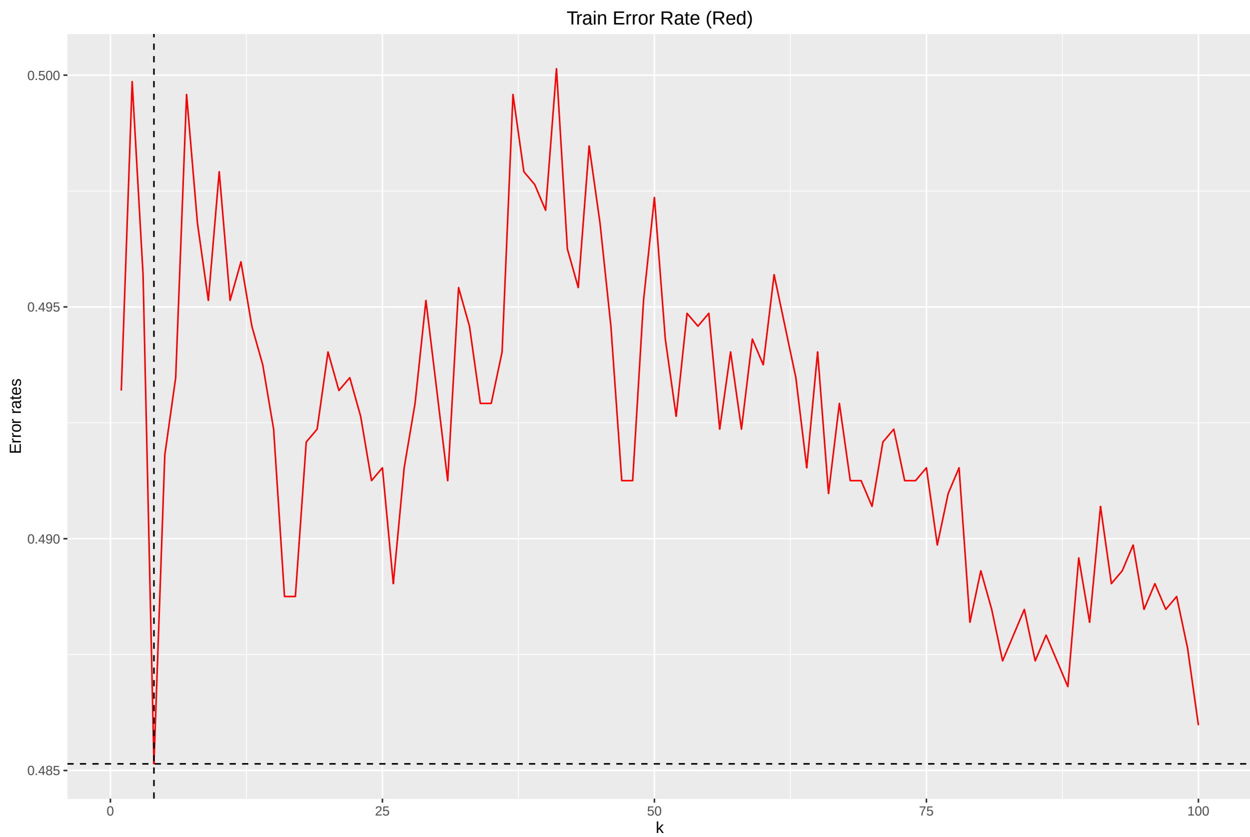
```
require(ggplot2)
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

```
p.YTrain = NULL
train.error.rate = NULL
for(i in 1:100){
  set.seed(3)
  # use test to determine optimal knn ??
  p.YTrain = knn.cv(train = test[,1:(ncol(test)-1)], cl = test$y, k = i)
  train.error.rate[i] = mean(train$y != p.YTrain)
}

gg4<-ggplot(data.frame(x = 1:100,y = train.error.rate))+geom_line(aes(x=x,y=y), color="Red")+xlab("k")+y
gg4
```



Part 3

```r
require(ROCR)
```

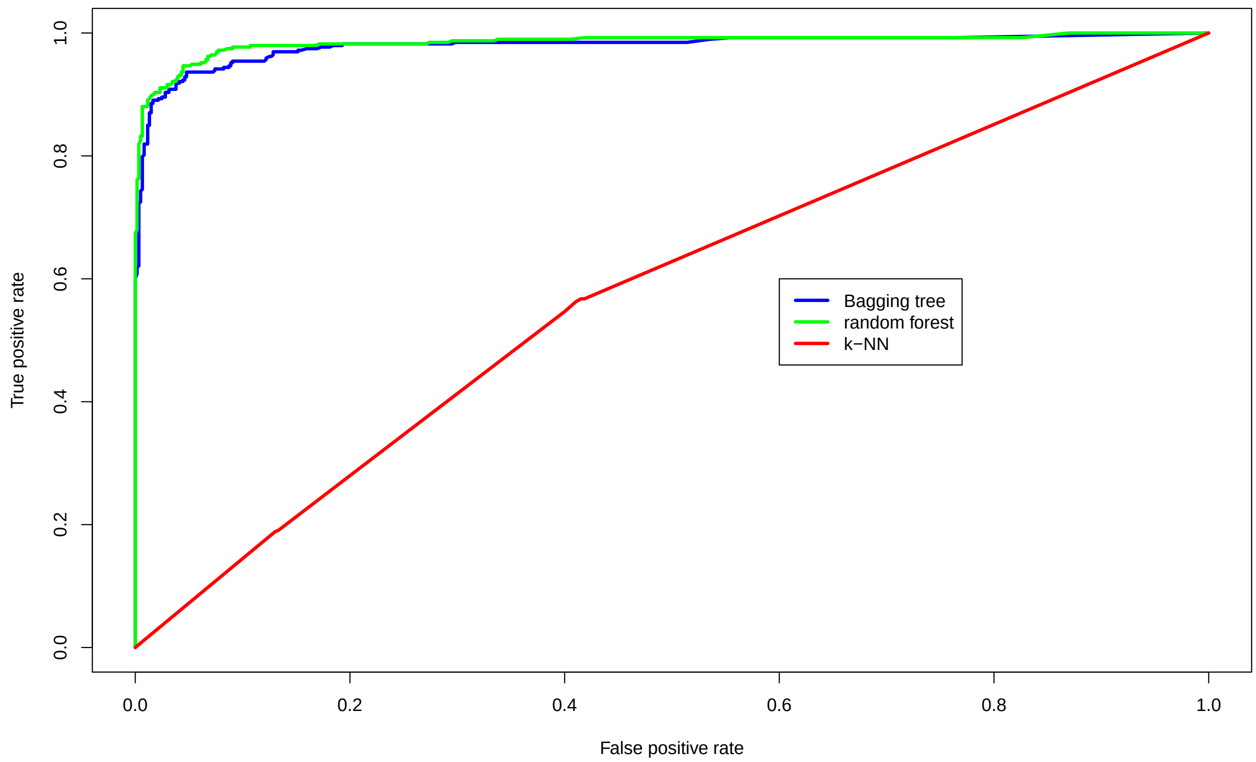
```
## Loading required package: ROCR

## Loading required package: gplots

##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess
```

```r
require(data.table)
```

```
## Loading required package: data.table
```

```r
bag.pred = data.table(predict(bag.spam,test,type="prob")[,-1])
pred.bag = prediction(bag.pred,test$y)
perf.bag = performance(pred.bag,measure="tpr",x.measure="fpr")
```

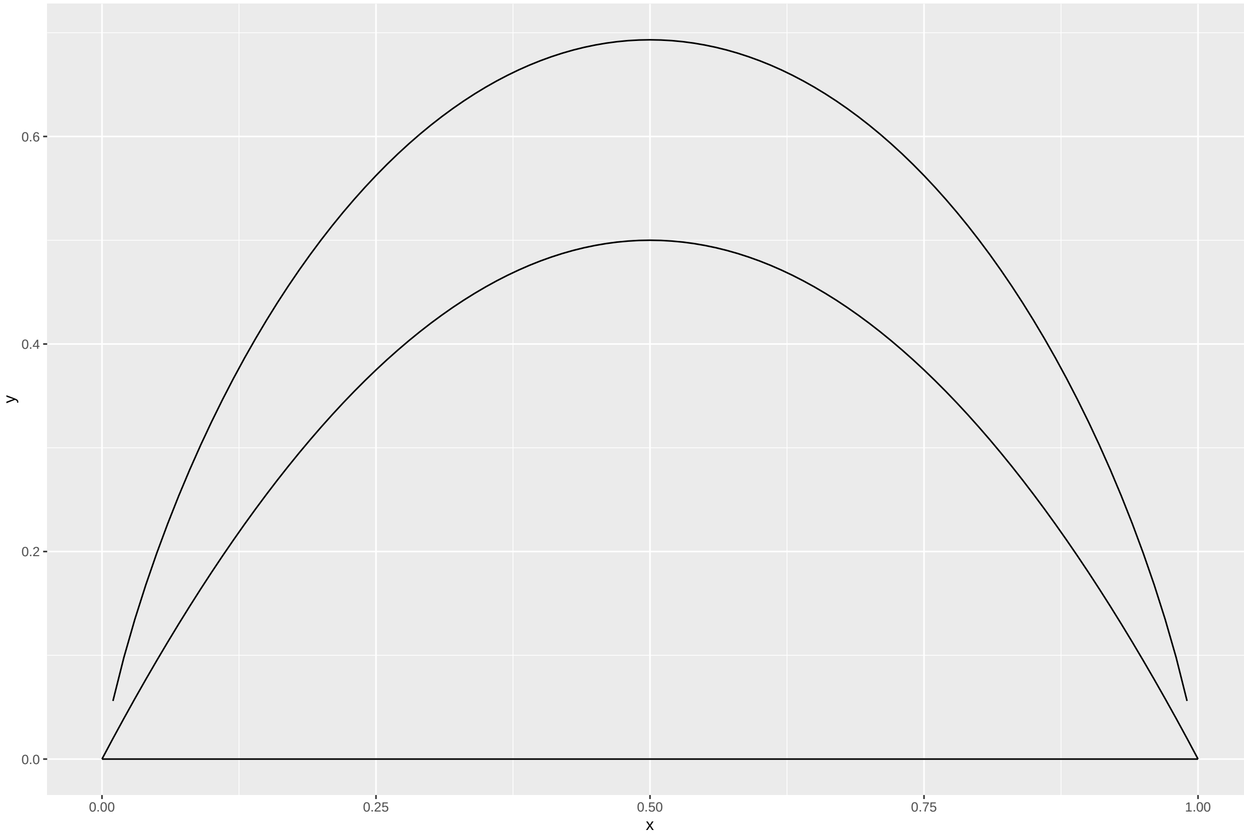
```
plot(perf.bag,col="blue",lwd=3)

rf.pred = data.table(predict(rand.spam,test,type="prob"))
#rf.pred
pred.rf = prediction(rf.pred[,c(spam)],test$y)
perf.rf = performance(pred.rf,measure = "tpr",x.measure = "fpr")
plot(perf.rf,lwd=3,add=T,col="green")

knn.pred = knn(train[,-ncol(train)],test[,-ncol(test)],train$y,k=4,prob=T)
knn.p=1-attributes(knn.pred)$prob
knn.p[knn.p==0]=0
#knn.p
pred.knn = prediction(knn.p,test$y)
#pred.knn
perf.knn = performance(pred.knn,measure = "tpr",x.measure = "fpr")
plot(perf.knn,lwd=3,add=T,col="red")
legend(0.6,0.6,c('Bagging tree','random forest','k-NN'),col=c('blue','green','red'),lwd=3)
```



Question 2

```
require(ggplot2)
funcs = ggplot(data.frame(x=c(0,1)),aes(x))
GiniIndex = function(p){2*p*(1-p)}
class.error = function(p) {min(p,(1-p))}
# class error rate measuere is wierd
```

```
cross.entropy = function(p) {-p*log(p)-(1-p)*log(1-p)}
funcs + stat_function(fun=GiniIndex) + stat_function(fun=class.error) + stat_function(fun=cross.entropy)
```



Question 3

Majority vote approach: Among the 10 estimates, we have 4 prediction that has a probability less than 0.5. Since we have more estimates indicating "class is red", we conclude that the class is red.

Average probabiltiy approach: The average probability among the 10 estimates is:

```
(0.1+0.15+0.2+0.2+0.55+0.6+0.6+0.65+0.7+0.75)/10
```

```
## [1] 0.45
```

Since the average probability is below 0.5, we conclude that the class is NOT red.