Statistical Learning HW4

Yaniv Bronshtein

11/28/2021

Import the necessary libraries

```
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```
library(tree)
library(rpart)
library(e1071)
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-2
```

```
library(ISLR)
library(superml) #for grid search cv for Q2
```

```
## Loading required package: R6
```

```
library(caret)
```

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

```
## Loading required package: lattice
```

Question 1

1. Write a program to implement AdaBoost with trees (Algorithm 10.1). [Hint. The rpart() func- tion has a argument weights, which you need to supply for Step 2(a) of the algorithm. Also, use the control=rpart.control(maxdepth=1) so that a stump is added in each step.] Do the following using your program.

Create a function to compute e

```
compute_e <- function(w, y, y_pred){
  return(sum(w*(y != y_pred)) / sum(w))
}
```

Create a function to compute alpha

```
compute_alpha <- function(e){
  return(log((1-e)/e))
}
```

Create a function to update the weights

```r
update_weights <- function(w, alpha, y, y_pred){
  return(w*exp(alpha*(y != y_pred)))
}
```

Create a function to compute the test error

```r
compute_test_errors <- function(y, output, n_rounds){
  test_error <- NULL
    for(i in 1:n_rounds){
      test_error <- c(test_error, sum(y != output[i,]) / length(y))
    }
  return(test_error)
}
```

Ada-boost function

```r
adaboost <- function(X_train, X_test, y_train, y_test, tree_depth, n_rounds){
  #Define the weights. Start with uniform weights
  w <- rep(1/nrow(X_train), nrow(X_train))
  #This is used for prediction
  classifier <- matrix(0, n_rounds, nrow(X_test))
  #Convert X_train and X_test to dataframes
  X_train <- data.frame(X_train)
  X_test <- data.frame(X_test)
  #Train
  alphas <- NULL
  for(i in 1:n_rounds){

    tree <- rpart(y_train ~ ., data=X_train, weights=w, method='class',
                  control=rpart.control(maxdepth=tree_depth))

    pred_train <- as.integer(as.character(predict(tree, X_train, type='class')))
    classifier[i,] <- as.integer(as.character(predict(tree, X_test, type='class')))

    #compute_error() function
    e <- compute_e(w=w, y=y_train, y_pred=pred_train)

    #compute_alpha() function
    alpha <- compute_alpha(e)
    alphas <- c(alphas, alpha)

    #update_weights() function
    w <- update_weights(alpha=alpha,w, y=y_train, y_pred=pred_train)
  }

  #Multiply predicted classifier by alpha.
  for(i in 1:n_rounds){
    classifier[i,] <- classifier[i,] * alphas[i]
  }

  #take colsum for each. then take sign()
  output <- ifelse(classifier[1,] < 0, -1, 1)
  for(i in 2:n_rounds){
    output <- rbind(output, ifelse(colSums(classifier[1:i,]) < 0, -1, 1))
  }
```

```
  #get the test classification error
  test_errors <- compute_test_errors(y=y_test, output=output, n_rounds=n_rounds)

  return(list('Predicted Class' = output, 'Test_Error' = test_errors))

}
```

Simulate the data

```
set.seed(123)
n.tr=2000; n.te=10000; p=10

X.tr=matrix(rnorm(n.tr*p),nrow=n.tr)
y=apply(X.tr^2,MAR=1,FUN="sum")
y=y>=9.34
y=as.factor(as.numeric(y))
ex1.tr=data.frame(X.tr,y)

X.te=matrix(rnorm(n.te*p),nrow=n.te)
y=apply(X.te^2,MAR=1,FUN="sum")
y=y>=9.34
y=as.factor(as.numeric(y))
ex1.te=data.frame(X.te,y)
```

Create the trees

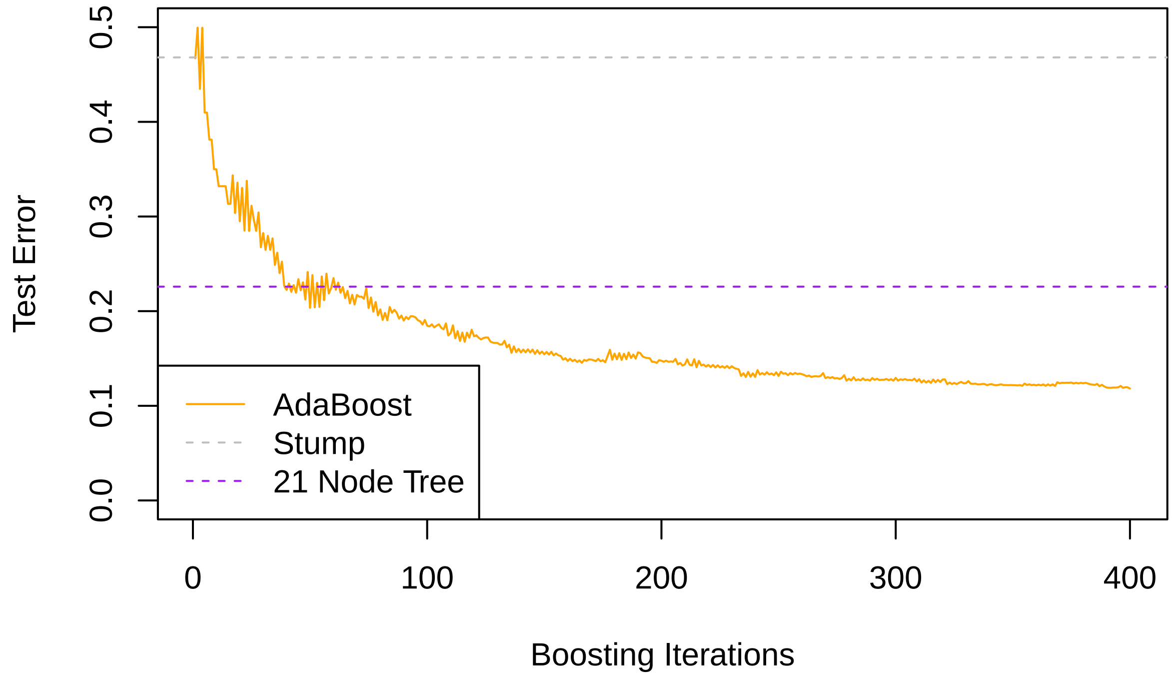
```
ex1.tree=tree(y~.,ex1.tr)
ex1.tree.pred=predict(ex1.tree,ex1.te,type="class")
test.error.tree=sum(ex1.tree.pred!=ex1.te$y)/n.te

ex1.stump=prune.tree(ex1.tree,best=2,method="deviance")
ex1.stump.pred=predict(ex1.stump,ex1.te,type="class")
test.error.stump=sum(ex1.stump.pred!=ex1.te$y)/n.te
```

Perform boosting

```
ex1.tr$y=as.numeric(ex1.tr$y)-1
ex1.te$y=as.numeric(ex1.te$y)-1
ntree=400
ex1.boost=adaboost(X_train=as.matrix(ex1.tr[,1:p]), X_test=as.matrix(ex1.te[,1:p]),
                   y_train=ifelse(ex1.tr$y==0, -1, 1), y_test=ifelse(ex1.te$y==0, -1, 1),
                   tree_depth=1, n_rounds=ntree)
```

Generate the adaboost plots

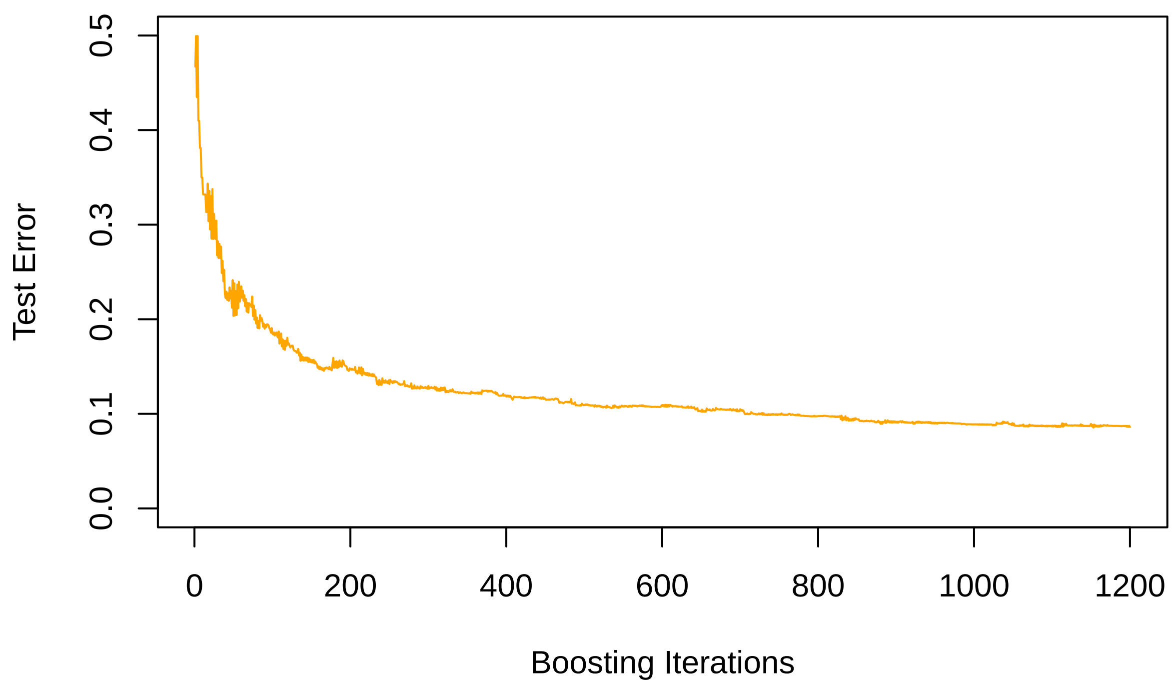
```
plot(1:ntree, ex1.boost$Test_Error, type="l", col="orange", xlab="Boosting Iterations", ylab="Test Erro
abline(h=test.error.tree,lty=2,col="purple")
abline(h=test.error.stump,lty=2,col="gray")
legend("bottomleft", c("AdaBoost","Stump","21 Node Tree"),
col=c("orange","gray","purple"), lty=c(1,2,2))
```



(b) Investigate the number of iterations needed to make the test error start to rise in the figure above.

Result of investigation led to choosing 1200 rounds

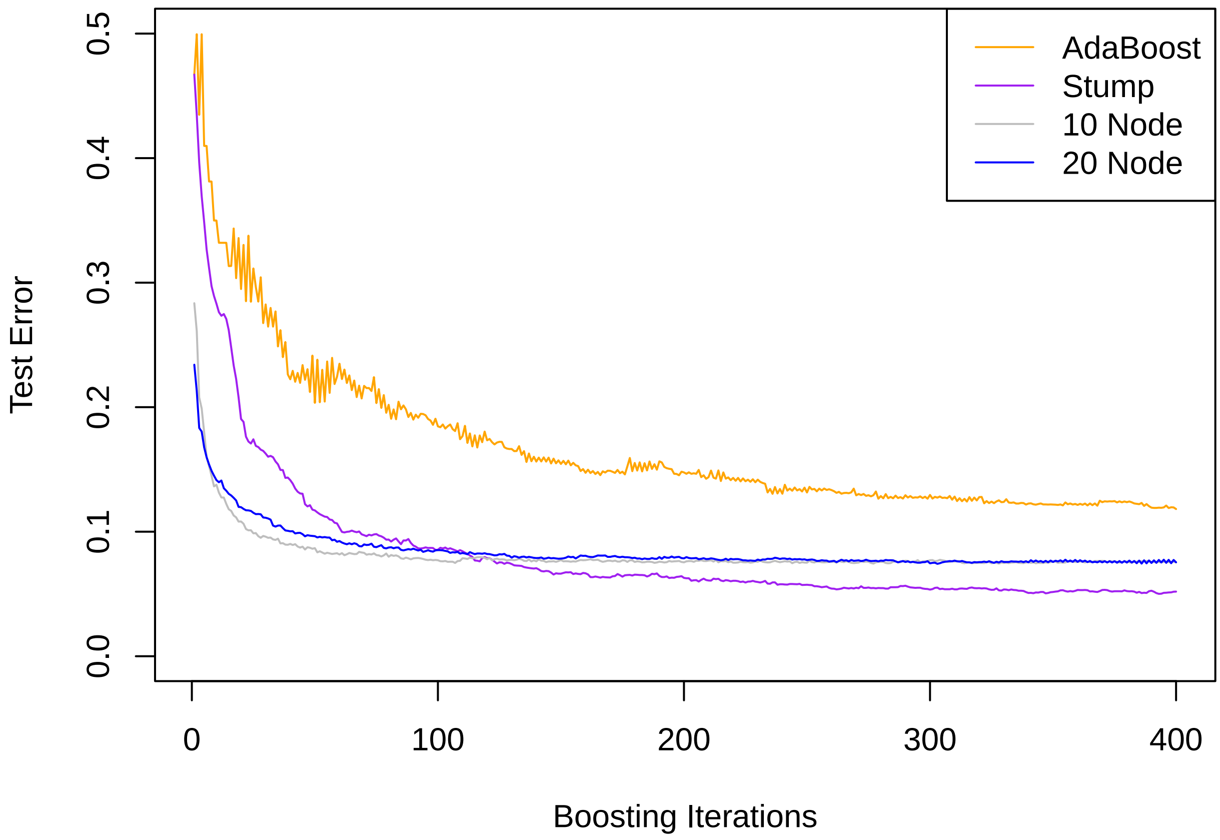
```
ex1.boost.model=adaboost(X_train=as.matrix(ex1.tr[,1:p]), X_test=as.matrix(ex1.te[,1:p]),
                         y_train=ifelse(ex1.tr$y==0, -1, 1), y_test=ifelse(ex1.te$y==0, -1, 1),
                         tree_depth=1, n_rounds=1200)

plot(1:1200, ex1.boost.model$Test_Error, type="l", col="orange", xlab="Boosting Iterations", ylab="Test
```



Part c). **Function to compute the error rate**

```r
error.rate=function(m,newdata,ntree){
  err=array(0,c(3,ntree))
  rownames(err)=c("Mis","Exp","Dev")
  for (i in 1:ntree){
    p=dim(newdata)[2]-1
    pp=predict(m,newdata=newdata[,1:p], n.trees=i)
    err[2,i]=mean(exp(-pp*(2*newdata$y-1)))
    err[3,i]=mean(log(1+exp(-2*pp*(2*newdata$y-1))))
    pp=pp>=0
    pp=as.numeric(pp)
    err[1,i]=mean(pp!=newdata$y)
  }
  err
}
```

```r
m1=gbm(y~.,data=ex1.tr,distribution="bernoulli",n.trees=ntree,
interaction.depth=1, shrinkage=1, bag.fraction=1)
m2=gbm(y~.,data=ex1.tr,distribution="bernoulli",n.trees=ntree,
interaction.depth=9, shrinkage=1, bag.fraction=1)
m3=gbm(y~.,data=ex1.tr,distribution="bernoulli",n.trees=ntree,
interaction.depth=19, shrinkage=1, bag.fraction=1)
err1=error.rate(m1,newdata=ex1.te,ntree=ntree)
err2=error.rate(m2,newdata=ex1.te,ntree=ntree)
err3=error.rate(m3,newdata=ex1.te,ntree=ntree)
par(mar=c(4.5,4.5,.5,.4))
plot(1:ntree, ex1.boost$Test_Error, type="l", col="orange", xlab="Boosting Iterations",
ylab="Test Error" ,ylim=c(0,0.5))
lines(1:ntree,err1[1,],type="l",col="purple")
lines(1:ntree,err2[1,],type="l",col="gray")
lines(1:ntree,err3[1,],type="l",col="blue")
legend("topright", c("AdaBoost","Stump","10 Node", "20 Node"),
col=c("orange","purple","gray","blue"), lty=rep(1,4))
```



Question 2

Read in data

```
spam <- read.table('../data/spam.txt')
spam_ind <- read.table('../data/spam_ind.txt')
```

Perform 50-50 train test split

```
set.seed(123)

train <- sample(1:nrow(spam), 0.5*nrow(spam))
spam_train <- spam[train,]
spam_test <- spam[-train,]
spam_ind_train <- spam_ind[[1]][train]
spam_ind_test <- spam_ind[[1]][-train]
```

Helper function for tuning

```
get_test_err <- function(gbm_model, spam_test, spam_ind_test){
  gbm_prob <- predict(gbm_model, spam_test, type='response') #get prediction probabilities
  gbm_pred <- rep('0', nrow(spam_test)) #Default 0 classification
  gbm_pred[gbm_prob > .5] <- '1' #Use 0.5 threshold to get 1 classification

  confusion_matrix <- table(gbm_pred, spam_ind_test) #Generate confusion matrix
  err <- 1-(confusion_matrix[1]+confusion_matrix[4])/nrow(spam_test) #Get the test error
  return(err)
}
```

Experiment with various interaction depth values.

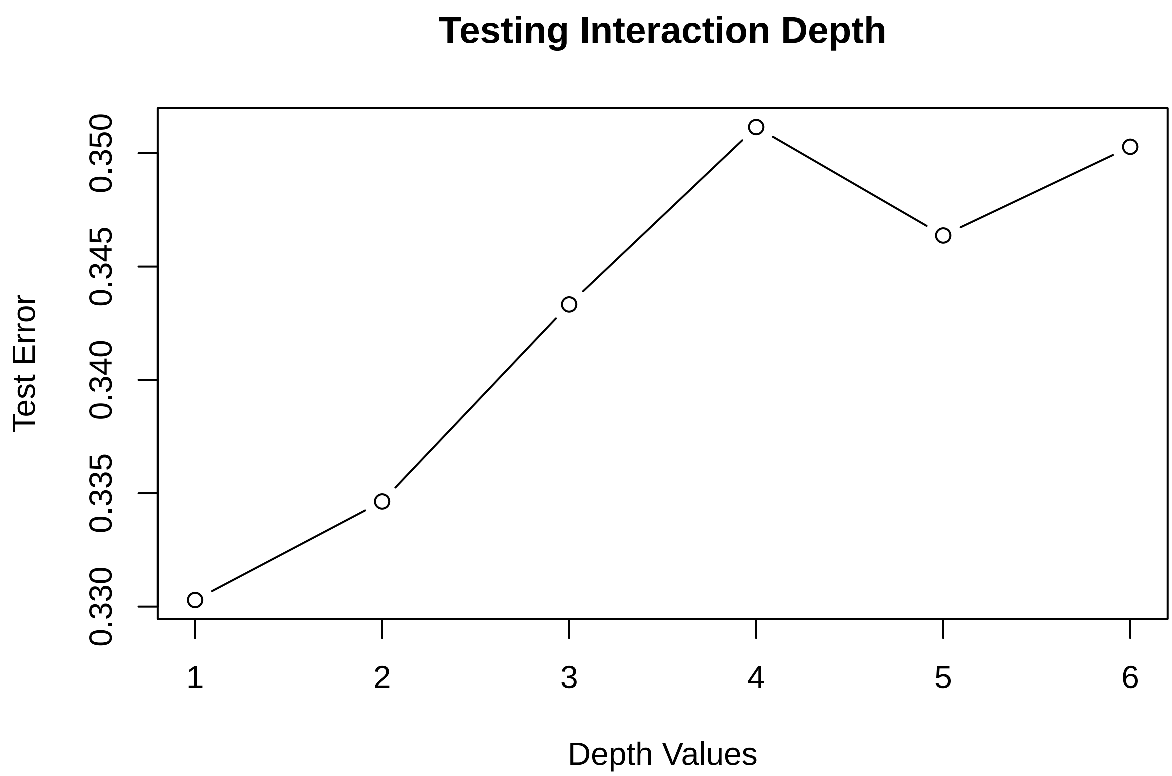
```
tune_interaction_depth <- function(){
  test_error <- NULL

  for(i in 1:6){
    set.seed(123)

    gbm_model <- gbm(formula=spam_ind_train ~ .,data=spam_train,interaction.depth=i)

    err <- get_test_err(gbm_model, spam_test, spam_ind_test)
    test_error <- c(test_error, err)
  }
  return(test_error)
}
```

Plot the test Error for interaction depth

```
test_error <- tune_interaction_depth()
```

```
## Distribution not specified, assuming bernoulli ...
## Distribution not specified, assuming bernoulli ...
## Distribution not specified, assuming bernoulli ...
## Distribution not specified, assuming bernoulli ...
## Distribution not specified, assuming bernoulli ...
## Distribution not specified, assuming bernoulli ...
```

```
plot(test_error,
     type='b',
     xaxt = 'n',
     xlab='Depth Values',
     main='Testing Interaction Depth',
     ylab='Test Error')
axis(1, at=1:6, 1:6)
```



We note from the above plot that the minimum test error happens at an interaction depth of 1. Thus we will continue using the default value of 1 during our testing of shrinkage

Now let us test shrinkage

```
tune_shrinkage <- function(){
  test_error <- NULL
  for(i in 0:5){
    set.seed(123)
    gbm_model <- gbm(spam_ind_train ~ ., data=spam_train, shrinkage=0.1 - 0.02*i)
    err <- get_test_err(gbm_model, spam_test, spam_ind_test)
    test_error <- c(test_error, err)
  }
  return(test_error)
}
```

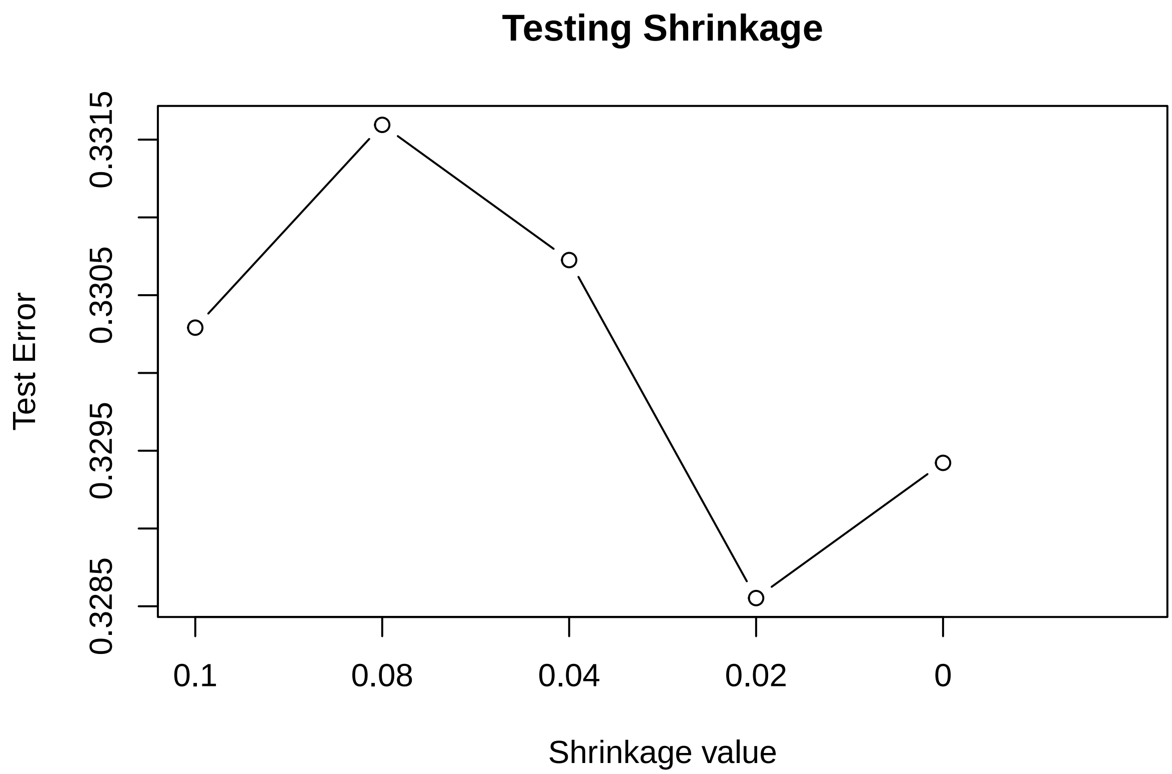
Get test errors and plot for varying shrinkage

```
test_error <- tune_shrinkage()
```

```
## Distribution not specified, assuming bernoulli ...

## Using 100 trees...

## Distribution not specified, assuming bernoulli ...

## Using 100 trees...

## Distribution not specified, assuming bernoulli ...

## Using 100 trees...

## Distribution not specified, assuming bernoulli ...

## Using 100 trees...
```

```
## Distribution not specified, assuming bernoulli ...

## Using 100 trees...

## Distribution not specified, assuming bernoulli ...

## Using 100 trees...
```

```
plot(test_error,
     type='b',
     xaxt = 'n',
     xlab='Shrinkage value',
     ylab='Test Error',
     main='Testing Shrinkage')
axis(1, at=1:5, labels=c(0.1, 0.08, 0.04, 0.02, 0))
```



Based on the graph of the test error, the best shrinkage should be 0 but I will use 0.02

Now let us test bag.fraction

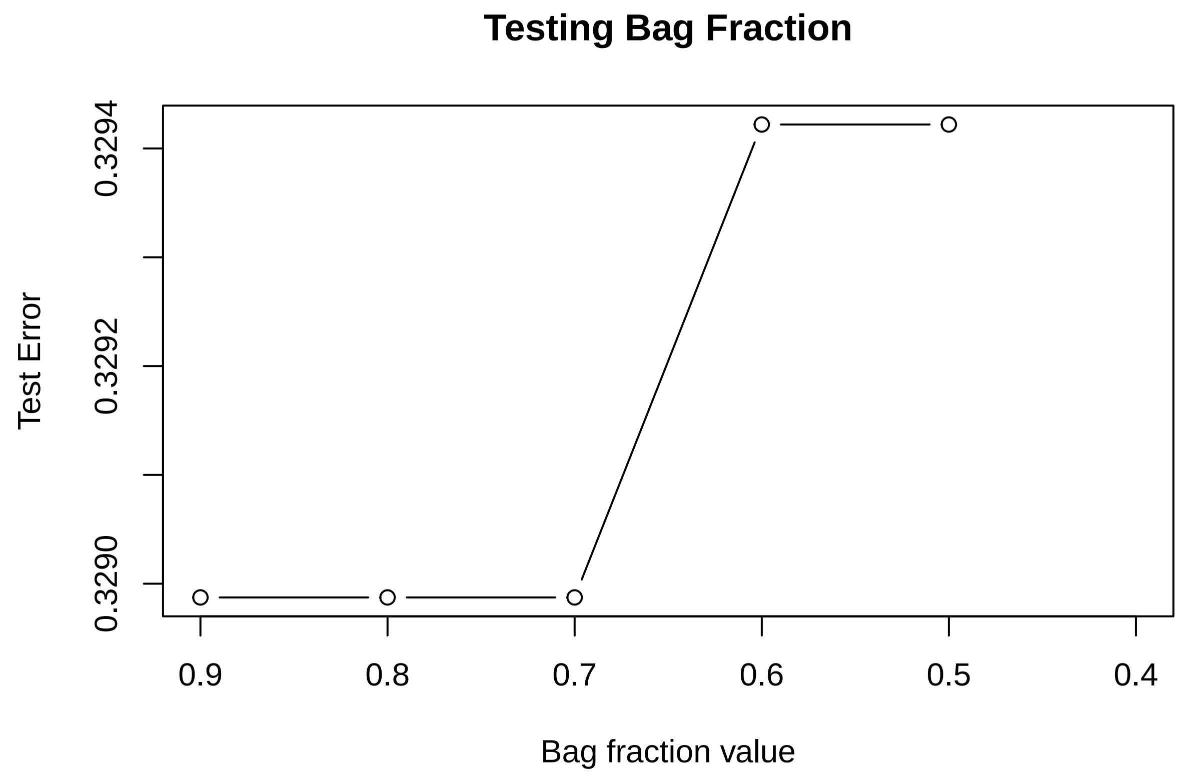
```
tune_bag_fraction <- function(){
  test_error <- NULL
  for(i in 1:6){
    set.seed(123)
    gbm_model <- gbm(spam_ind_train ~ .,data=spam_train, shrinkage=0.02,bag.fraction=1-0.1*i)
    err <- get_test_err(gbm_model, spam_test, spam_ind_test)
    test_error <- c(test_error, err)
  }
  return(test_error)
}
```

Get test errors and plot for varying bag.fraction

```
test_error <- tune_bag_fraction()
```

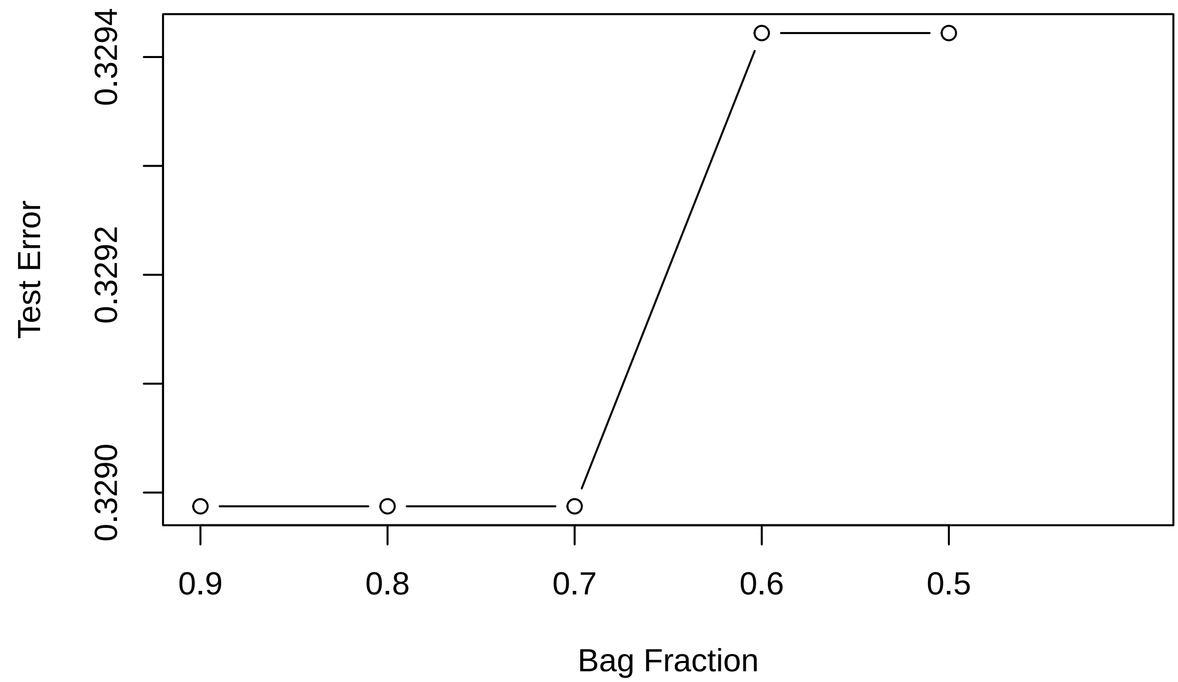
```
## Distribution not specified, assuming bernoulli ...
## Using 100 trees...
## Distribution not specified, assuming bernoulli ...
## Using 100 trees...
## Distribution not specified, assuming bernoulli ...
## Using 100 trees...
## Distribution not specified, assuming bernoulli ...
## Using 100 trees...
## Distribution not specified, assuming bernoulli ...
## Using 100 trees...
## Distribution not specified, assuming bernoulli ...
## Using 100 trees...
```

```
plot(test_error,
     type='b',
     xaxt = 'n',
     xlab='Bag fraction value',
     ylab='Test Error',
     main='Testing Bag Fraction')
axis(1, at=1:6, labels=c(0.9, 0.8, 0.7, 0.6, 0.5, 0.4))
```



generate plot

```r
plot(test_error, type='b', xaxt = 'n', xlab='Bag Fraction', ylab='Test Error')
axis(1, at=1:5, labels=c(0.9, 0.8, 0.7, 0.6, 0.5))
```



Based on the plot, I will set the bag.fraction to 0.9

Now let us test the final parameter of n.trees

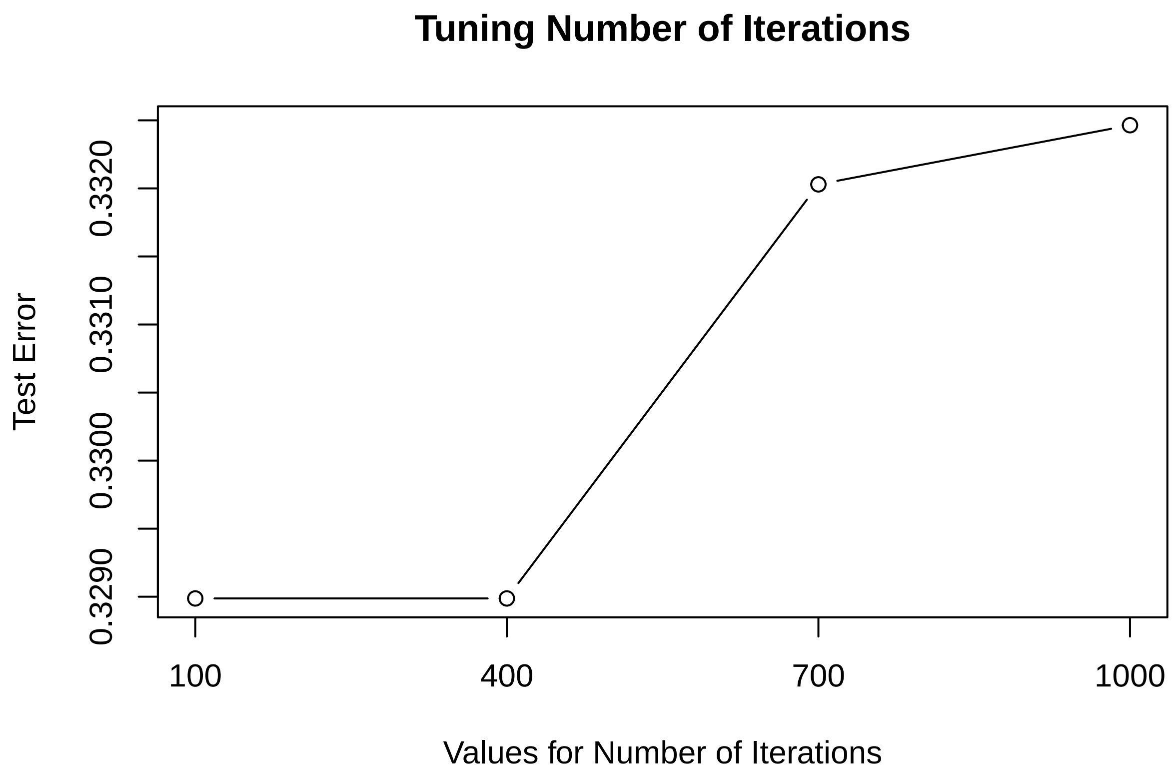
```r
tune_ntrees <- function(){
  test_error <- NULL
  for(i in 0:3){
    set.seed(123)
    gbm_model <- gbm(spam_ind_train ~ .,
                     data=spam_train,
                     shrinkage=0.02,
                     bag.fraction=0.9,
                     n.trees=100 + 300*i
                     )
    err <- get_test_err(gbm_model, spam_test, spam_ind_test)
    test_error <- c(test_error, err)
  }
  return(test_error)
}
```

```r
test_error <- tune_ntrees()
```

```
## Distribution not specified, assuming bernoulli ...

## Using 100 trees...

## Distribution not specified, assuming bernoulli ...

## Using 400 trees...

## Distribution not specified, assuming bernoulli ...

## Using 700 trees...
```

```
## Distribution not specified, assuming bernoulli ...
## Using 1000 trees...
```

```
plot(test_error,
     type='b',
     xaxt = 'n',
     xlab='Values for Number of Iterations',
     ylab='Test Error',
     main='Tuning Number of Iterations')
axis(1, at=1:4, labels=c(100, 400, 700, 1000))
```



Based on the results of the plot, I will stick to using 100 trees

Let us fit the final optimal model

```
set.seed(123)

gbm_model_optimal <- gbm(spam_ind_train ~.,
                   data=spam_train,
                   distribution='bernoulli',
                   n.trees=100,
                   shrinkage=0.02,
                   interaction.depth=1,
                   bag.fraction=0.9)

optimal_test_err <- get_test_err(gbm_model_optimal, spam_test, spam_ind_test)
optimal_test_err
```

```
## [1] 0.3289874
```

Now let's try logistic regression

```
lr_model <- glm(spam_ind_train ~ ., data=spam_train, family='binomial')
lr_test_err <- get_test_err(lr_model, spam_test, spam_ind_test)
lr_test_err
```

```
## [1] 0.3311604
```

And now SVM

```
svm_model <- glm(spam_ind_train ~ ., data=spam_train, family='binomial')
lr_test_err <- get_test_err(lr_model, spam_test, spam_ind_test) #can recycle helper from before
lr_test_err
```

```
## [1] 0.3311604
```

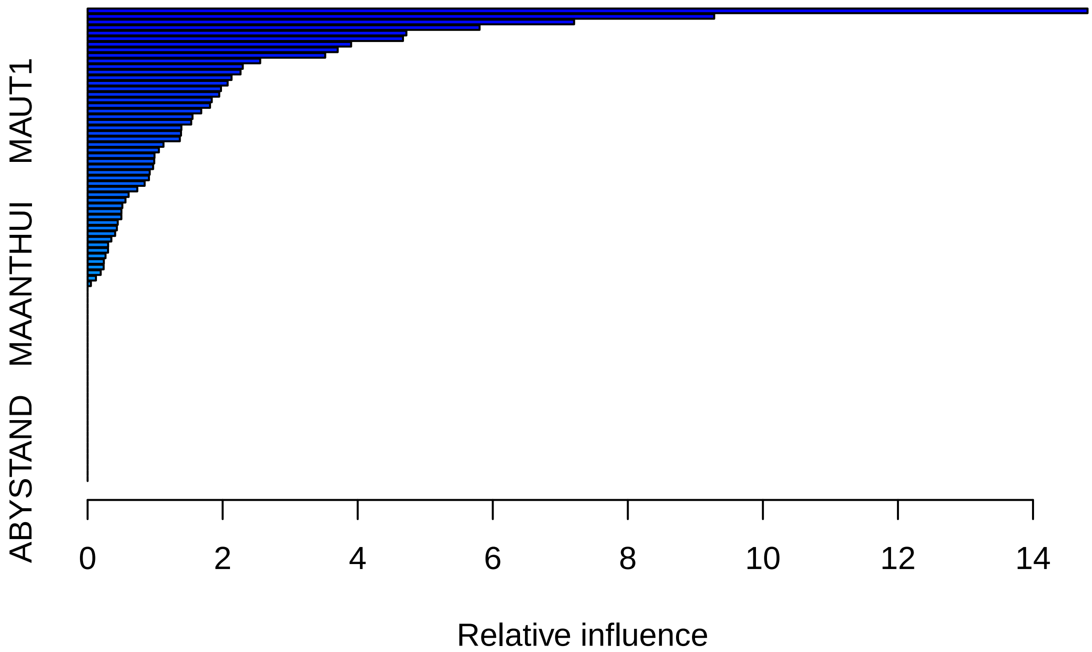
```
svm_model <- svm(spam_ind_train ~ .,
                data=spam_train,
                kernel='radial',
                probability=TRUE)
svm_prob <- as.numeric(predict(svm_model, spam_test, probability=TRUE))
svm_pred <- rep('0', nrow(spam_test))
svm_prob[svm_prob > .5] <- '1'
confusion_matrix <- table(svm_pred, spam_ind_test)

svm_test_err <- 1-((confusion_matrix[1] + confusion_matrix[4])/nrow(spam_test))
svm_test_err
```

```
## [1] NA
```

Question 3

Note:code similar to textbook

```
data(Caravan)

tr <- Caravan[1:1000,]
te <- Caravan[1001:nrow(Caravan),]

gbm_model <- suppressWarnings(gbm(ifelse(tr$Purchase == 'No', 0, 1) ~ ., data=tr,
                                distribution='bernoulli', n.trees=1000, shrinkage=0.01))
summary(gbm_model)
```




```
##                 var     rel.inf
## PPERSAUT PPERSAUT 14.80774876
## MKOOPKLA MKOOPKLA  9.27926304
## MOPLHOOG MOPLHOOG  7.20458021
## MBERMIDD MBERMIDD  5.80450799
## ABRAND     ABRAND  4.72082026
## PBRAND     PBRAND  4.67070835
## MGODGE     MGODGE  3.90249875
## MOSTYPE   MOSTYPE  3.70400708
## MINK3045 MINK3045  3.51848111
## PWAPART   PWAPART  2.55586639
## MAUT2       MAUT2  2.29814565
## MSKC         MSKC  2.26497001
## MSKA         MSKA  2.13157597
## MGODPR     MGODPR  2.07393754
## MSKB1       MSKB1  1.97512491
## MBERARBG MBERARBG  1.94931343
## MINKGEM   MINKGEM  1.83874748
## MAUT0       MAUT0  1.81299626
## MAUT1       MAUT1  1.68260965
## PBYSTAND PBYSTAND  1.55339478
## MGODOV     MGODOV  1.53431349
## MRELOV     MRELOV  1.38890523
## MBERHOOG MBERHOOG  1.38336457
## MFWEKIND MFWEKIND  1.36597373
## MOPLMIDD MOPLMIDD  1.12384966
## MFGEKIND MFGEKIND  1.05566262
## MINK4575 MINK4575  0.99080443
## MINK7512 MINK7512  0.98653289
## MRELGE     MRELGE  0.97034935
## MHHUUR     MHHUUR  0.92027440
## MGODRK     MGODRK  0.90829746
## APERSAUT APERSAUT  0.84499956
```

```
## MHKOOP      MHKOOP  0.73585266
## PMOTSCO    PMOTSCO  0.60794972
## MINKM30    MINKM30  0.56090709
## MBERBOER MBERBOER  0.51425164
## MZFONDS    MZFONDS  0.50093585
## MOSHOOFD MOSHOOFD  0.49999606
## MFALLEEN MFALLEEN  0.44680656
## MGEMLEEF MGEMLEEF  0.43344020
## MBERARBO MBERARBO  0.40783627
## MSKD          MSKD  0.35170003
## PLEVEN      PLEVEN  0.30407487
## MGEMOMV    MGEMOMV  0.30297317
## MSKB2        MSKB2  0.26525383
## MINK123M MINK123M  0.24063375
## MZPART      MZPART  0.23795909
## MRELSA      MRELSA  0.19430348
## MOPLLAAG MOPLLAAG  0.12361145
## MAANTHUI MAANTHUI  0.04888925
## MBERZELF MBERZELF  0.00000000
## PWABEDR    PWABEDR  0.00000000
## PWALAND    PWALAND  0.00000000
## PBESAUT    PBESAUT  0.00000000
## PVRAAUT    PVRAAUT  0.00000000
## PAANHANG PAANHANG  0.00000000
## PTRACTOR PTRACTOR  0.00000000
## PWERKT      PWERKT  0.00000000
## PBROM        PBROM  0.00000000
## PPERSONG PPERSONG  0.00000000
## PGEZONG    PGEZONG  0.00000000
## PWAOREG    PWAOREG  0.00000000
## PZEILPL    PZEILPL  0.00000000
## PPLEZIER PPLEZIER  0.00000000
## PFIETS      PFIETS  0.00000000
## PINBOED    PINBOED  0.00000000
## AWAPART    AWAPART  0.00000000
## AWABEDR    AWABEDR  0.00000000
## AWALAND    AWALAND  0.00000000
## ABESAUT    ABESAUT  0.00000000
## AMOTSCO    AMOTSCO  0.00000000
## AVRAAUT    AVRAAUT  0.00000000
## AAANHANG AAANHANG  0.00000000
## ATRACTOR ATRACTOR  0.00000000
## AWERKT      AWERKT  0.00000000
## ABROM        ABROM  0.00000000
## ALEVEN      ALEVEN  0.00000000
## APERSONG APERSONG  0.00000000
## AGEZONG    AGEZONG  0.00000000
## AWAOREG    AWAOREG  0.00000000
## AZEILPL    AZEILPL  0.00000000
## APLEZIER APLEZIER  0.00000000
## AFIETS      AFIETS  0.00000000
## AINBOED    AINBOED  0.00000000
## ABYSTAND ABYSTAND  0.00000000
```

Fitting gbm() with n.trees = 1000 shrinkage 0.01, we get predictors appear to be most PPERSAUT, MKOOP-KLA, MOPLHOOG, and MBERMIDD deemed most important

```
gbm_prob <- predict(gbm_model, te, type='response')
gbm_pred <- rep('No', nrow(Caravan)-1000)
gbm_pred[gbm_prob > .2] <- 'Yes'
```

```
confusion_matrix <- table(gbm_pred, te$Purchase)
confusion_matrix
```

```
##
## gbm_pred   No  Yes
##      No  4407  254
##      Yes  126   35
```

```
cat('Precision:', confusion_matrix[4]/(confusion_matrix[2] + confusion_matrix[4]))
```

```
## Precision: 0.2173913
```

Fraction of people who made the purchase is around 20%

```
glm_model <- glm(tr$Purchase ~ ., data=tr, family=binomial)
```

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

```
glm_prob <- predict(glm_model, te, type='response')
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
glm_pred <- rep('No', nrow(Caravan)-1000)
glm_pred[glm_prob > .2] <- 'Yes'
confusion_matrix <- table(glm_pred, te$Purchase)
cat('Precision:', confusion_matrix[4]/(confusion_matrix[2] + confusion_matrix[4]))
```

```
## Precision: 0.1421569
```

Logistic Regression is not nearly as high as boosting. Around half as precise

4. Derive expression (10.12) of ESL for the update of β in AdaBoost

10.11 in the text shows that

$$(e^{\beta}-e^{-\beta})\sum_{i=1}^{N} w_i^{(m)} + e^{\beta} \cdot \sum_{y_i \neq G(x_i)} w_i^{(m)}$$

We are given in (10.9) that:

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} w_i^{(m)} \exp(-\beta y_i G(x_i))$$

$$G_m = \arg\min_{G} \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i))$$

We take partial derivatives and set to 0:

$$\sum_{i=1}^{N} w_i^{(m)} y_i G(x_i) \exp(-\beta y_i G(x_i)) = 0$$

Separate:

$$\sum_{y_i = G(x_i)} w_i^{(m)} \exp(-\beta) - \sum_{y_i \neq G(x_i)} w_i^{(m)} \exp(\beta) = 0$$

$$\sum_{y_i = G(x_i)} w_i^{(m)} \exp(-\beta) = \sum_{y_i \neq G(x_i)} w_i^{(m)} \exp(\beta)$$

Multiplying Both Sides by e^{β} and through a series of steps:

$$\exp(2\beta) = \frac{\sum_{y_i = G(x_i)} w_i^{(m)}}{\sum_{y_i \neq G(x_i)} w_i^{(m)}}$$

$$= \frac{1 - err_m}{err_m}$$

*We define errm as the minimized weighted error rate

$$err_m = \frac{\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i^{(m)}}$$

We arrive at 10.12: $\beta = \frac{1}{2} \log \frac{1 - err_m}{err_m}$ ∎