# Task 2

## Introduction

For task 2, we are going to deal with a dataset containing information about webmails. We have variables describing the frequency of some specific words like *“remove”* in each observation, variables describing the frequency of some specific chars like *“$”* in one observation, and three variables, *capital\_run\_length\_longest*, *capital\_run\_length\_average* and *capital\_run\_length\_total*, describing the length of the longest uninterrupted sequence of capital letters, the average length of uninterrupted sequences of capital letters, and the total number of capital letters in each observation respectively. We also have a variable called *spam*, which indicates whether this webmail is a spam with and , where for spam, and for not spam. Here all our variables are numeric type. Our task is to use these attribute variables to classify whether a webmail is spam.

## Methodology

In order to validate the accuracy of our methods, we firstly divide our dataset into a train set, which contains 2500 observations, and a test set, which contains 2101 observations. We use the train set to train our models, and then apply it to the test set to validate its accuracy.

## Results

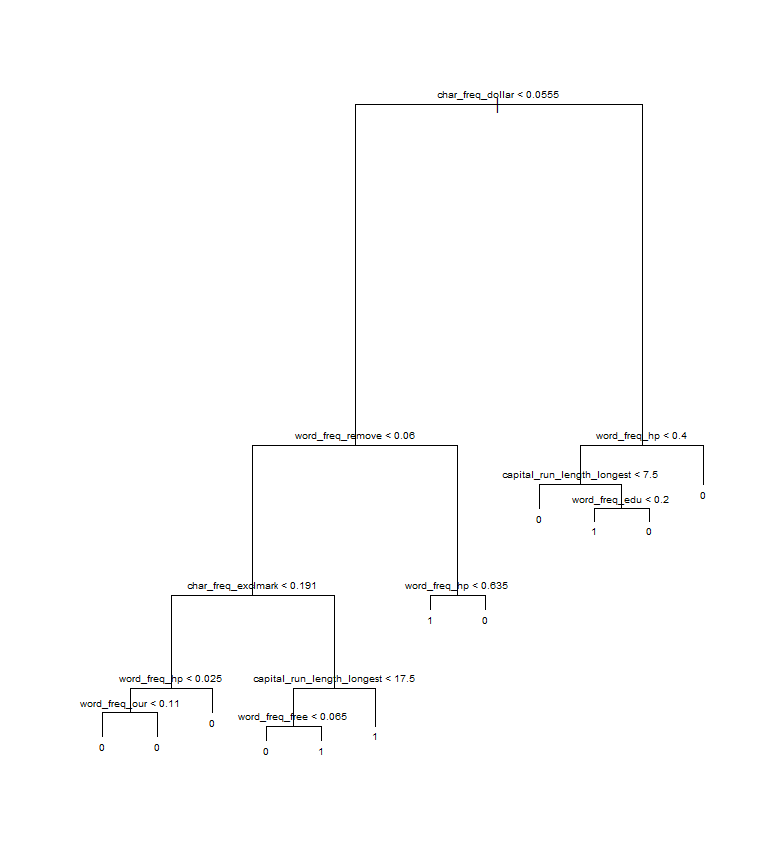
### 1. Classification Trees

In this part, we are going to discuss the results obtained by complex tree model and pruned tree model. We begin with construct a complex tree model by dividing our observation into small non-overlapping regions according to some numerical criteria. Here we split our dataset until each leaf of our classification tree contains only less then 2 observations. The method used here is recursive binary splitting.

#grow complex tree using deviance as criterion  
tree.mod = tree(factor(spam) ~ . ,data.train,   
 control = tree.control(nobs = 2500, minsize = 2, mincut = 1),  
 split = "deviance")  
  
summary(tree.mod)

Classification tree:  
tree(formula = factor(spam) ~ ., data = data.train, control = tree.control(nobs = 2500,   
 minsize = 2, mincut = 1), split = "deviance")  
Variables actually used in tree construction:  
[1] "char\_freq\_dollar" "word\_freq\_remove"   
[3] "char\_freq\_exclmark" "word\_freq\_hp"   
[5] "word\_freq\_our" "capital\_run\_length\_longest"  
[7] "word\_freq\_free" "word\_freq\_edu"   
Number of terminal nodes: 12   
Residual mean deviance: 0.4883 = 1215 / 2488   
Misclassification error rate: 0.084 = 210 / 2500

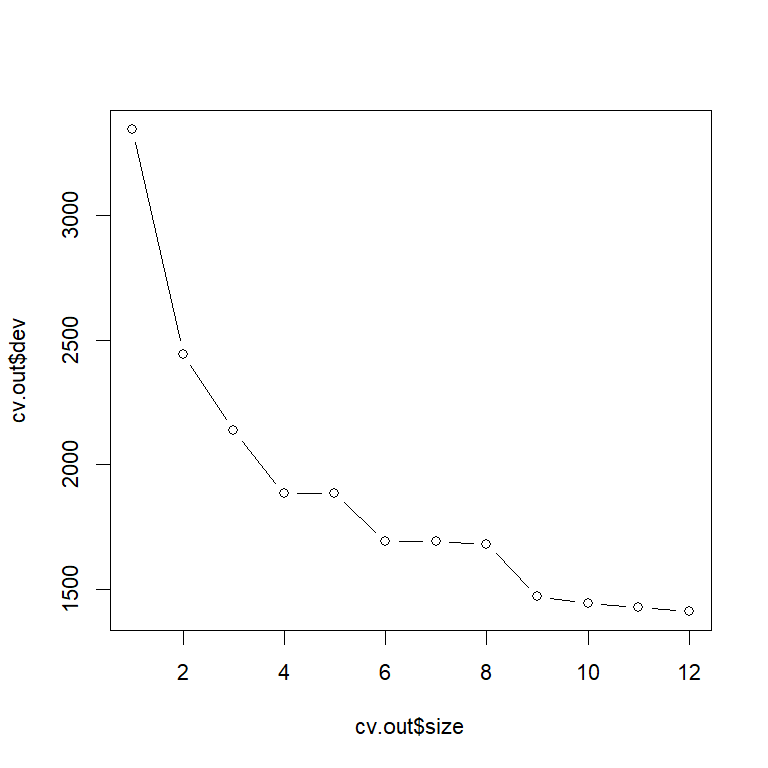
#plot tree  
plot(tree.mod)  
text(4, 7, expression(bar(x) == sum(frac(x[i], n), i==1, n)))  
#rpart.plot(tree.mod)  
text(tree.mod,pretty=10,cex=0.65)



We can see clearly here that criteria concerning the frequency of “$”, “remove”, “!”, “hp”, “our”, “free”, “edu” as well as the length of the longest uninterrupted sequence of capital letters are used for splitting. It’s actually quite reasonable, because from our own experience, spam webmails are always advertisements on money related topics or education related topics, and are always filled with words in capital letters, together with exclamation symbols, to draw attention.

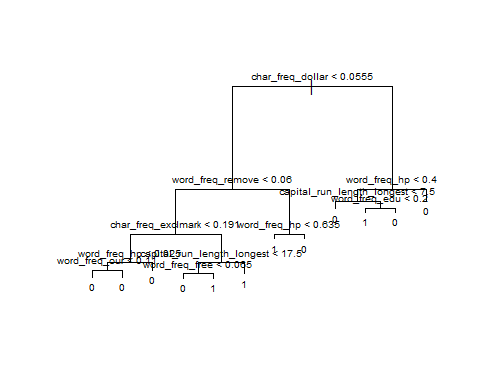
Since the process of recursive binary splitting may lead to overfitting, where we obtain a complex tree with a good fit on the training data, but with a poor performance on test data, we introduce the process of tree pruning. Actually, a smaller tree with fewer splits may have a lower variance at the cost of acceptable little bias. In order to decide the optimal tuning parameter which leads to both much lower variance and acceptable bias, we use cross-validation to make a selection. In fact, the least cross-validation error implies the least probability of overfitting, as it’s also a train-test process.

#use cross-validation to select tuning parameter for pruning the tree  
 set.seed(0829539)  
 cv.out=cv.tree(tree.mod,K=5)  
 par(cex=1.4)  
 plot(cv.out$size,cv.out$dev,type='b')



However, in this specific task, we can see that the cross-validation error is monotonously decreasing, so the optimal fold number is the original fold number. We choose best size=12 here such that the pruned tree model here is exactly the same as our complex tree model.

#prune the tree  
prune.mod=prune.tree(tree.mod,best=12)  
plot(prune.mod)  
text(prune.mod,pretty=10, cex=0.65)



Now we validate the accuracy of our classification tree model with the test data.

#make predictions on training and test set using the unpruned tree  
pred.train<- predict(tree.mod,newdata=data.train)  
classif.train<-ifelse(pred.train[,2]>=pred.train[,1],1,0)  
t26 <- err(data.train$spam,classif.train)  
kbl(t26$tab)

0

1

0

1460

67

1

143

830

#make predictions on training and test set using the unpruned tree  
pred.test<-predict(tree.mod,newdata=data.test)  
classif.test<-ifelse(1\*pred.test[,2]>=pred.test[,1],1,0)  
t27 <- err(data.test$spam,classif.test)  
kbl(t27$tab)

0

1

0

1209

52

1

153

687

#make predictions on training and test set using the pruned tree  
pred.train<-predict(prune.mod,newdata=data.train)  
classif.train<-ifelse(pred.train[,2]>=pred.train[,1],1,0)  
t28 <- err(data.train$spam,classif.train)  
kbl(t28$tab)

0

1

0

1460

67

1

143

830

#make predictions on training and test set using the pruned tree  
pred.test<-predict(prune.mod,newdata=data.test)  
classif.test<-ifelse(1\*pred.test[,2]>=pred.test[,1],1,0)  
t29 <- err(data.test$spam,classif.test)  
kbl(t29$tab)

0

1

0

1209

52

1

153

687

We can conclude that our classification model performs very well. since the complex tree is the same as pruned tree here, we can see that the test error is just very slightly higher than the train error. There is not much overfitting here.

### 2. Different Classification Models

Now for this part, we are going to compare different classification models using two different classification scenarios.

Firstly, we consider the scenario where we have different prior probabilities and equal classification costs. We use the entire dataset to figure out the prior probability.

#(a)Account for different prior probabilities and equal classification costs   
# (1) linear discriminant analysis  
set.seed(0829539)  
ldaS.out<-lda(spam ~ .,data = spamdata)  
print(ldaS.out$prior)

0 1   
0.6059552 0.3940448

So, we take (is spam)=0.394, P (not spam) =0.606 as our prior probability, and we can verify that their sum equals 1.

The four classification models we are going to compare is 1) Linear Discriminant Analysis, 2) Bagging, 3) Random Forests and 4) Gradient Boosting. Since the dimension here is very high and the mathematical assumption of our dataset is very weak, we do not suppose that Linear Discriminant Analysis without Principal Component analysis here will have very good performance. We just take it as a baseline. The three other methods are all methods used to improve the classification tree model, as tree models are always with high variance, thus high probability to cause overfitting.

The results are listed as below:

#### 1) Linear Discriminant Analysis

lda.out<-lda(spam~.,prior=c(0.606,0.394),data=data.train)  
  
scaling <- lda.out$scaling %>% as.data.frame() %>% mutate(name = rownames(lda.out$scaling))  
scaling %>% filter(name %in% c("char\_freq\_dollar", "word\_freq\_remove", "word\_freq\_table",  
 "word\_freq\_conference", "char\_freq\_dotcomma",   
 "char\_freq\_parenthesis", "char\_freq\_bracket", "char\_freq\_exclmark" ))

LD1 name  
1 0.8727936 word\_freq\_remove  
2 -0.7164016 word\_freq\_table  
3 -0.2623494 word\_freq\_conference  
4 -0.6216711 char\_freq\_dotcomma  
5 -0.3637729 char\_freq\_parenthesis  
6 -0.2686609 char\_freq\_bracket  
7 0.3970286 char\_freq\_exclmark  
8 0.8798683 char\_freq\_dollar

Here we can see that the three attributes highly correlated with spam webmails the frequency of “$”, “!” and “remove”, are exactly the same as what we have got from our classification tree model. What is interesting here is we can also see that the frequency of “table”, “;” and “parenthesis” shows strong negative correlation with spam webmails.

pred.train<-predict(lda.out,data.train)  
t212 <- err(data.train$spam,pred.train$class)  
kbl(t212$tab)

0

1

0

1460

67

1

202

771

pred.test<-predict(lda.out,data.test)  
t213 <- err(data.test$spam,pred.test$class)  
kbl(t213$tab)

0

1

0

1205

56

1

195

645

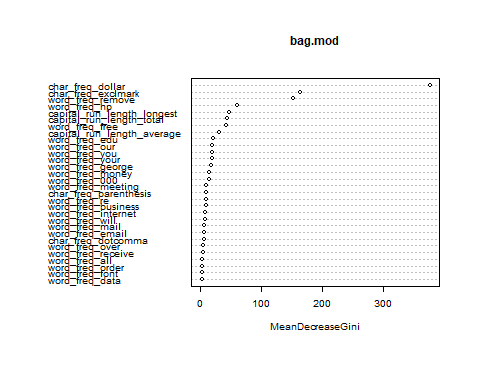
#### 2) Bagging

just some text

#(2) Bagging  
set.seed(0829539)  
bag.mod=randomForest(spam ~ ., data = data.train,   
 classwt=c(0.606,0.394),  
 mtry=57, ntree=nrotree, importance=TRUE)  
bag.mod

Call:  
 randomForest(formula = spam ~ ., data = data.train, classwt = c(0.606, 0.394), mtry = 57, ntree = 5000, importance = TRUE)   
 Type of random forest: classification  
 Number of trees: 5000  
No. of variables tried at each split: 57  
  
 OOB estimate of error rate: 6.76%  
Confusion matrix:  
 0\_no 1\_yes class.error  
0\_no 1463 64 0.04191225  
1\_yes 105 868 0.10791367

#importance(bag.mod,plot=TRUE)  
varImpPlot(bag.mod, type = 2, cex = 0.6)



pred.train<-predict(bag.mod,newdata=data.train)  
t218 <- err(data.train$spam,pred.train)  
kbl(t218$tab)

0\_no

1\_yes

0

1527

0

1

0

973

pred.test<-predict(bag.mod,newdata=data.test)  
t219 <- err(data.test$spam,pred.test)  
#kbl(t219$tab)

Actually, with the MeanDecreaseGini graph we can see that the frequency of “$”, “!” and “remove” made evidently the most contribution. This is the same conclusion we have drawn from our classification tree model and Linear Discriminant Analysis. Here Out Of Bag error rate is 6.76%, which is acceptable.

#### 3) Random Forests

just some text.

# (3) random forests  
set.seed(0829539)  
rf.mod=randomForest(spam ~ ., data = data.train,   
 classwt=c(0.606,0.394), mtry=5,   
 ntree=5000, importance=TRUE)  
rf.mod

Call:  
 randomForest(formula = spam ~ ., data = data.train, classwt = c(0.606, 0.394), mtry = 5, ntree = 5000, importance = TRUE)   
 Type of random forest: classification  
 Number of trees: 5000  
No. of variables tried at each split: 5  
  
 OOB estimate of error rate: 5.12%  
Confusion matrix:  
 0\_no 1\_yes class.error  
0\_no 1484 43 0.02815979  
1\_yes 85 888 0.08735868

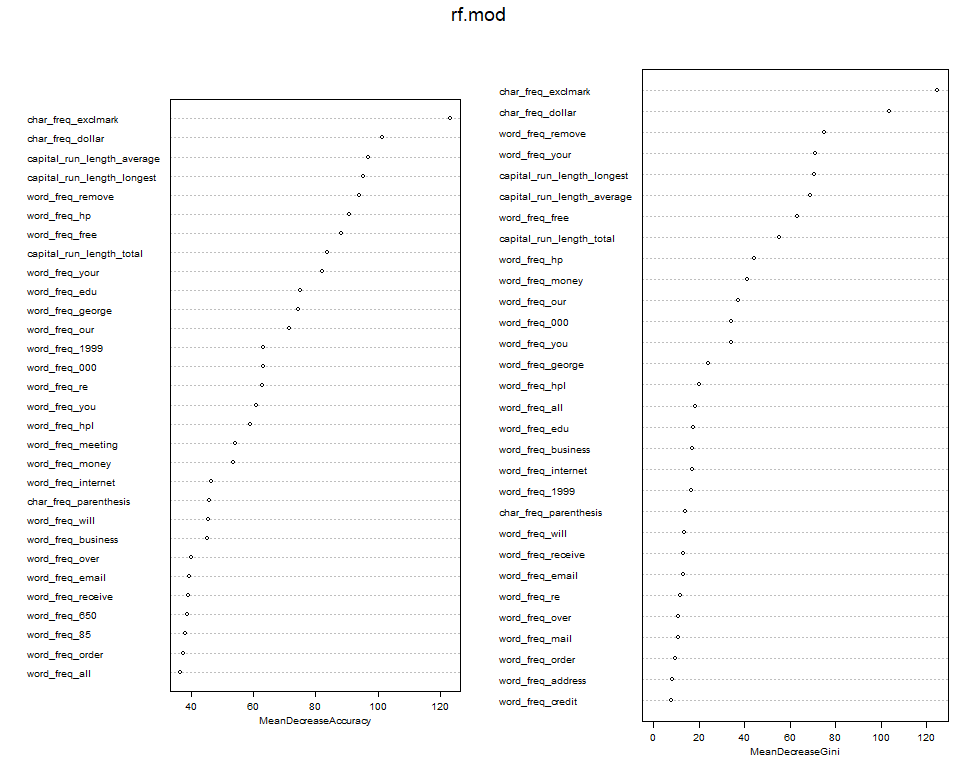
pred.train<-predict(rf.mod,newdata=data.train)  
t2\_21Y1 <- err(data.train$spam,pred.train)  
t2\_21Y1$tab

predicted  
observed 0\_no 1\_yes  
 0 1527 0  
 1 9 964

pred.test<-predict(rf.mod,newdata=data.test)  
t2\_21Y2 <-err(data.test$spam,pred.test)  
t2\_21Y2$tab

predicted  
observed 0\_no 1\_yes  
 0 1227 34  
 1 72 768

varImpPlot(rf.mod, cex = 0.6)



#varImpPlot(bag.mod, type = 2, cex = 0.6)

Random Forests Model is also based on the idea of decorrelating trees. Here we choose m=5 for the size of our predictor set. can see that although there is slightly difference, “$”, “!” and “remove” still stand for spam webmails, as well as the length of the longest uninterrupted sequence of capital letters.

#### 4) Gradient Boosting

Different from Bagging and Random Forests, the number of trees for Gradient Boosting is not expected to be as large as possible, since using too many trees may cause overfitting for Gradient Boosting. We start with deciding the optimal number of trees by cross-validation, for the same reason explained before.

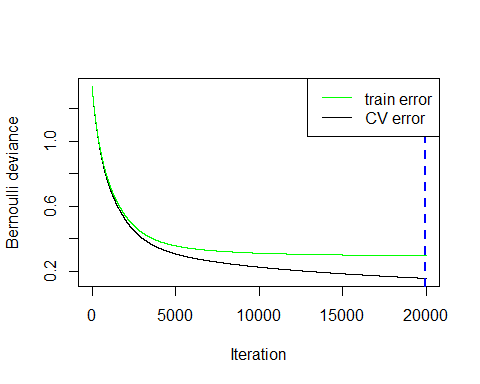
Note that since we are going to suppose Bernoulli distribution in our gbm function, we should firstly make sure that our variable for classification is numeric type.

# (4) gradient boosting  
set.seed(0829539)  
data.train$spam<-ifelse(data.train$spam=="1\_yes",1,0)  
data.test$spam<-ifelse(data.test$spam=="1\_yes",1,0)  
spamdata$spam<-ifelse(spamdata$spam=="1\_yes",1,0)

load(paste0(base\_url,"T2/T2\_gbm\_3.Rdata"))  
boost.mod<-t2\_gbm\_3

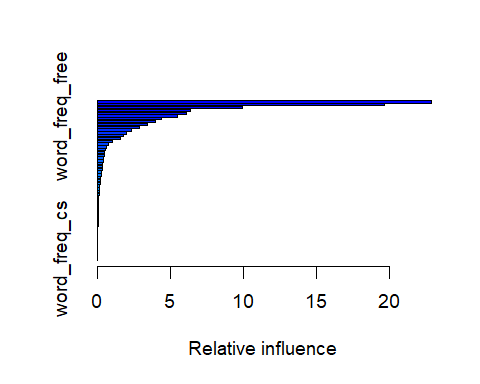
#use distribution="bernoulli" for binary target  
#interaction.depth=4, means that we fit a tree that uses 4 splits   
#(and that includes at most a 4-th order interaction)  
boost.mod=gbm(spam ~ . ,data = data.train, distribution = "bernoulli",   
 n.trees = 20000, interaction.depth = 4, shrinkage = 0.001, cv.folds = 5)  
gbm.perf(boost.mod,method="cv")  
legend("topright",c("train error","CV error"),col=c("green","black"),lty=c(1,1))

[1] 19932



So here we get that the optimal number of trees is 19932. We use this parameter to continue our analysis.

#relative influence plot  
par(cex=1.2)  
summary(boost.mod,n.trees=19932)



var rel.inf  
char\_freq\_dollar char\_freq\_dollar 2.285917e+01  
char\_freq\_exclmark char\_freq\_exclmark 1.965558e+01  
word\_freq\_remove word\_freq\_remove 9.933912e+00  
word\_freq\_hp word\_freq\_hp 6.371358e+00  
word\_freq\_your word\_freq\_your 6.075841e+00  
word\_freq\_free word\_freq\_free 5.428856e+00  
capital\_run\_length\_longest capital\_run\_length\_longest 4.344714e+00  
capital\_run\_length\_average capital\_run\_length\_average 3.937332e+00  
capital\_run\_length\_total capital\_run\_length\_total 3.391273e+00  
word\_freq\_george word\_freq\_george 2.866851e+00  
word\_freq\_edu word\_freq\_edu 2.308666e+00  
word\_freq\_our word\_freq\_our 1.976375e+00  
word\_freq\_money word\_freq\_money 1.743456e+00  
word\_freq\_000 word\_freq\_000 1.559673e+00  
word\_freq\_you word\_freq\_you 1.013728e+00  
word\_freq\_meeting word\_freq\_meeting 7.223353e-01  
word\_freq\_re word\_freq\_re 6.227953e-01  
word\_freq\_internet word\_freq\_internet 5.493848e-01  
word\_freq\_1999 word\_freq\_1999 4.530588e-01  
word\_freq\_business word\_freq\_business 4.244831e-01  
word\_freq\_over word\_freq\_over 3.985102e-01  
word\_freq\_email word\_freq\_email 3.710928e-01  
word\_freq\_receive word\_freq\_receive 3.484809e-01  
word\_freq\_will word\_freq\_will 3.377431e-01  
char\_freq\_dotcomma char\_freq\_dotcomma 3.321253e-01  
char\_freq\_parenthesis char\_freq\_parenthesis 2.853891e-01  
word\_freq\_report word\_freq\_report 2.587867e-01  
word\_freq\_technology word\_freq\_technology 2.084485e-01  
word\_freq\_mail word\_freq\_mail 1.903978e-01  
word\_freq\_hpl word\_freq\_hpl 1.677002e-01  
word\_freq\_650 word\_freq\_650 1.504262e-01  
word\_freq\_3d word\_freq\_3d 1.131915e-01  
word\_freq\_all word\_freq\_all 1.063984e-01  
word\_freq\_font word\_freq\_font 8.440753e-02  
word\_freq\_project word\_freq\_project 7.595445e-02  
word\_freq\_make word\_freq\_make 6.275894e-02  
word\_freq\_pm word\_freq\_pm 4.042293e-02  
word\_freq\_address word\_freq\_address 3.613297e-02  
word\_freq\_order word\_freq\_order 3.496306e-02  
word\_freq\_parts word\_freq\_parts 2.876464e-02  
word\_freq\_conference word\_freq\_conference 2.842824e-02  
word\_freq\_credit word\_freq\_credit 2.467152e-02  
word\_freq\_people word\_freq\_people 2.197801e-02  
word\_freq\_85 word\_freq\_85 1.685307e-02  
char\_freq\_pound char\_freq\_pound 1.468910e-02  
word\_freq\_data word\_freq\_data 1.128843e-02  
word\_freq\_labs word\_freq\_labs 4.584238e-03  
char\_freq\_bracket char\_freq\_bracket 3.254895e-03  
word\_freq\_direct word\_freq\_direct 2.550972e-03  
word\_freq\_lab word\_freq\_lab 4.378566e-04  
word\_freq\_original word\_freq\_original 1.606371e-04  
word\_freq\_addresses word\_freq\_addresses 9.088811e-05  
word\_freq\_table word\_freq\_table 7.843543e-05  
word\_freq\_telnet word\_freq\_telnet 0.000000e+00  
word\_freq\_857 word\_freq\_857 0.000000e+00  
word\_freq\_415 word\_freq\_415 0.000000e+00  
word\_freq\_cs word\_freq\_cs 0.000000e+00

We can actually get nearly the same result as we obtained from all the former models concerning the variable most correlated to spam webmails here, that the frequency of “$”, “!” and “remove” come top.

pred.train<-predict(boost.mod,n.trees=19932,newdata=data.train,type="response")  
classif.train<-ifelse(pred.train>=1-pred.train,1,0)  
t225 <- err(data.train$spam,classif.train)  
t225$tab

predicted  
observed 0 1  
 0 1547 953

pred.test<-predict(boost.mod,n.trees=19932,newdata=data.test,type="response")  
classif.test<-ifelse(pred.test>=1-pred.test,1,0)  
t226 <- err(data.test$spam,classif.test)  
t226$tab

predicted  
observed 0 1  
 0 1287 814

Then, we consider the scenario where we have different prior probabilities and unequal classification costs: C(spam|email)=10\*C(email|spam). Actually, the models here remain the same, and we just need to do some reclassification based on the posterior probabilities obtained from our built models and a criterion taking the classification costs as weights for posterior probabilities.

##### 1) Linear Discriminant Analysis

**some text here**

pred.train<-predict(lda.out,data.train)  
classif.train<-ifelse(1\*pred.train$posterior[,2]>=10\*pred.train$posterior[,1],1,0)  
t227Y2 <- err(data.train$spam, classif.train)  
t227Y2$tab

predicted  
observed 0 1  
 0\_no 1878 622

pred.test<-predict(lda.out,data.test)  
classif.test<-ifelse(1\*pred.test$posterior[,2]>=10\*pred.test$posterior[,1],1,0)  
t227Y3 <- err(data.test$spam,classif.test)  
t227Y3$tab

predicted  
observed 0 1  
 0\_no 1601 500

##### 2) bagging

**some text here**

pred.train<-predict(bag.mod,type="prob",newdata=data.train)  
classif.train<-ifelse(1\*pred.train[,2]>=10\*pred.train[,1],1,0)  
t228Y1 <- err(data.train$spam,classif.train)  
t228Y1$tab

predicted  
observed 0 1  
 0\_no 1702 798

pred.test<-predict(bag.mod,type="prob",newdata=data.test)  
classif.test<-ifelse(1\*pred.test[,2]>=10\*pred.test[,1],1,0)  
t228Y2 <- err(data.test$spam,classif.test)  
t228Y2$tab

predicted  
observed 0 1  
 0\_no 1512 589

##### 3) Random Forests

**some text here**

pred.train<-predict(rf.mod,type="prob",newdata=data.train)  
classif.train<-ifelse(1\*pred.train[,2]>=10\*pred.train[,1],1,0)  
t229Y1 <- err(data.train$spam,classif.train)  
t229Y1$tab

predicted  
observed 0 1  
 0\_no 1745 755

pred.test<-predict(rf.mod,type="prob",newdata=data.test)  
classif.test<-ifelse(1\*pred.test[,2]>=10\*pred.test[,1],1,0)  
t229Y2 <- err(data.test$spam,classif.test)  
t229Y2$tab

predicted  
observed 0 1  
 0\_no 1645 456

1. Gradient Boosting **some text here**

pred.train<-predict(boost.mod,n.trees=19932,newdata=data.train,type="response")  
classif.train<-ifelse(1\*pred.train>=10\*(1-pred.train),1,0)  
t230Y1 <- err(data.train$spam,classif.train)  
t230Y1$tab

predicted  
observed 0 1  
 0\_no 1693 807

pred.test<-predict(boost.mod,n.trees=19932,newdata=data.test,type="response")  
classif.test<-ifelse(1\*pred.test>=10\*(1-pred.test),1,0)  
t230Y2 <- err(data.test$spam,classif.test)  
t230Y2$tab

predicted  
observed 0 1  
 0\_no 1467 634

We arrange all the training errors and test errors into a whole table below, and calculate the sensitivities as well as the false positive rates for the test set, using the probability tables we printed out:

We can see that as we claimed at the beginning, Linear Discriminant Analysis performs the worst in both scenarios, Random Forests performs the best in scenario a), where we have different prior probabilities but equal classification costs, and Gradient Boosting performs the best in scenario b), where we have different prior probabilities and unequal classification costs. Generally, for scenario a), the performance of Random Forests and Gradient Boosting is equally good, and Bagging also performs good, just with slightly higher test error than Random Forests and Gradient Boosting. However, for scenario b), the accuracy of Random Forests decreases much faster than Bagging.

What’s more, as is expected, in scenario b), since misclassifying a not spam webmail into spam ones costs much more than misclassifying a spam webmail into not spam ones, our models all tend to classify more cases as not spam ones to minimize the cost. As a result, the false positive rate is very low for all models, but the accuracy and sensitivity decrease significantly for all models, since a number of real positive cases are classified as not spam webmails. Gradient Boosting and Bagging are relatively more stable here compared with the other two models.