Task 2

Introduction

For task 2, we are going to deal with a dataset containing information about 4601 webmails. We have 48 variables describing the frequency of some specific words like *"remove"* in each observation, 6 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 0 and 1, where 1 for spam, and 0 for not spam. Here all our variables are numeric type. Our task is to use these 57 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

Classification Trees

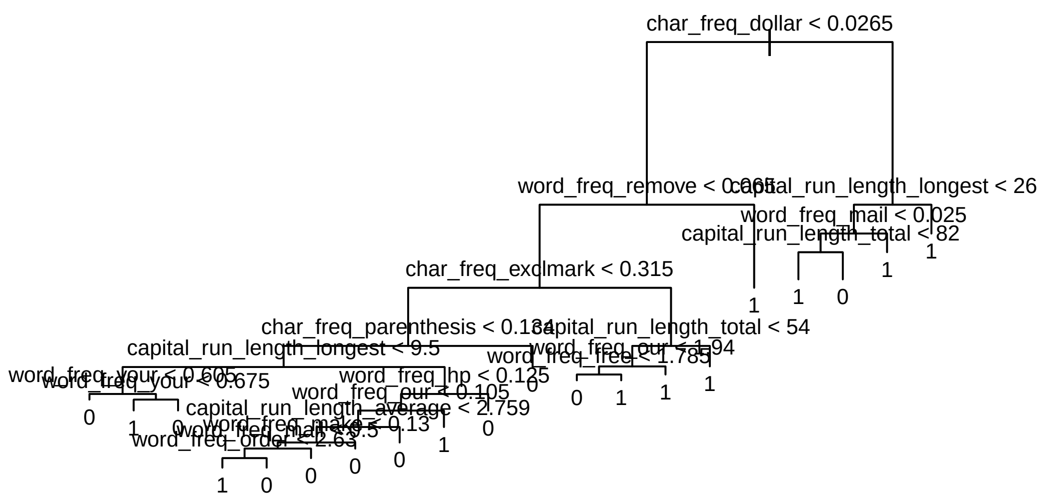
In this part, we are going to discuss the results obtained by complex tree model and pruned tree model. We begin with construct a complex tree model by dividing our observation into small non-overlapping regions according to some numerical criteria. Here we split our dataset until each leaf of our classification 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")

#tree.mod = rpart(spam ~ . ,data.train,
#                 control = tree.control(nobs = 2500, minsize = 2, mincut = 1))

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"         "char_freq_parenthesis"
 [5] "capital_run_length_longest" "word_freq_your"
 [7] "word_freq_hp"               "word_freq_our"
 [9] "capital_run_length_average" "word_freq_make"
[11] "word_freq_mail"             "word_freq_order"
[13] "capital_run_length_total"   "word_freq_free"
Number of terminal nodes:  20
Residual mean deviance:  0.02621 = 6.028 / 230
Misclassification error rate: 0.004 = 1 / 250
```

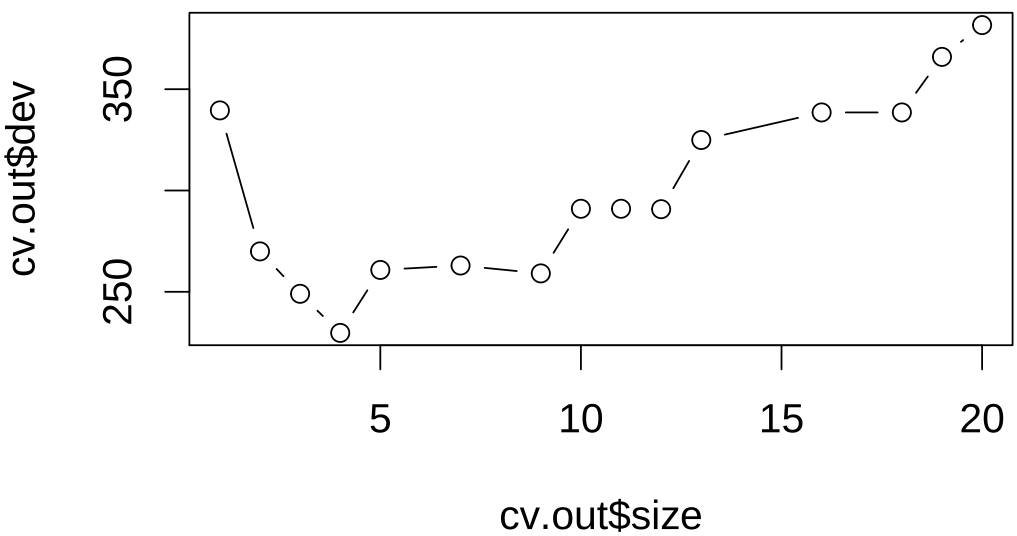
```r
#plot tree
plot(tree.mod)
#rpart.plot(tree.mod)
text(tree.mod,pretty=3,cex=0.75)
```



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

```r
#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.

```r
#prune the tree
prune.mod=prune.tree(tree.mod,best=12)
plot(prune.mod)
text(prune.mod,pretty=0)
```

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

```r
#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)
err(data.train$spam,classif.train)

pred.test<-predict(tree.mod,newdata=data.test)
classif.test<-ifelse(1*pred.test[,2]>=pred.test[,1],1,0)
err(data.test$spam,classif.test)
```

```r
#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)
err(data.train$spam,classif.train)
pred.test<-predict(prune.mod,newdata=data.test)
classif.test<-ifelse(1*pred.test[,2]>=pred.test[,1],1,0)
err(data.test$spam,classif.test)
```

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.

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.

```r
#(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 P (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

```r
lda.out<-lda(spam~.,prior=c(0.606,0.394),data=data.train)
```

```
Warning in lda.default(x, grouping, ...): variables are collinear
```

```r
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  0.6815563      word_freq_remove
2 -0.4296151       word_freq_table
3 -0.4725607  word_freq_conference
4 -0.5919005    char_freq_dotcomma
5 -0.4811911 char_freq_parenthesis
6 -1.3138634     char_freq_bracket
7  0.2859002    char_freq_exclmark
8  0.9359106      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)
err(data.train$spam,pred.train$class)
```

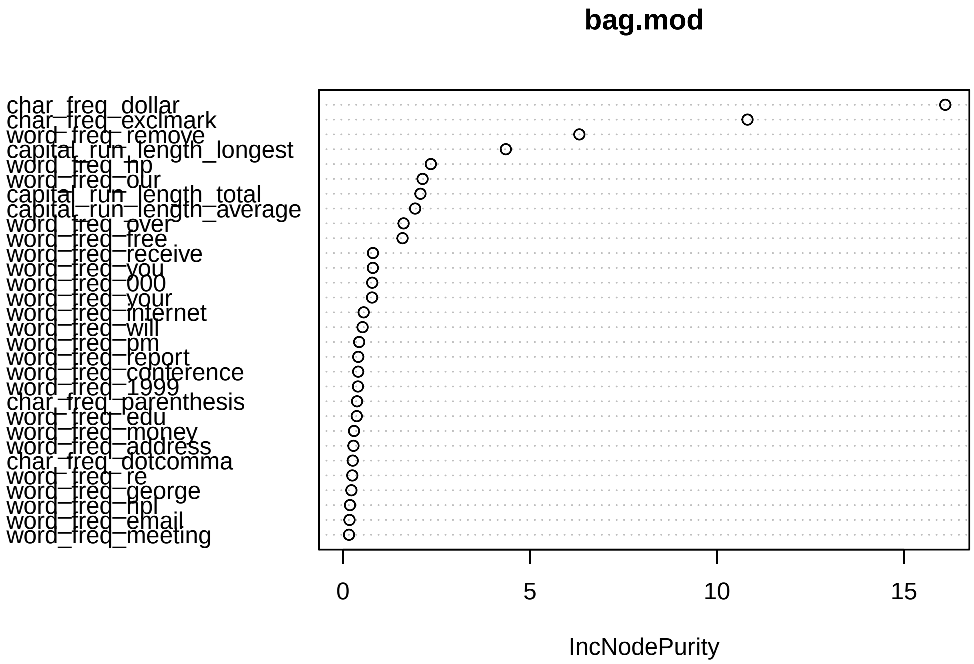
```
pred.test<-predict(lda.out,data.test)
err(data.test$spam,pred.test$class)
```

2) Bagging just some test

```
#(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 =
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 57

          Mean of squared residuals: 0.07989729
                    % Var explained: 66.82
```

```
#plot variable importance
varImpPlot(bag.mod,type=2,cex=.8)
```




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

pred.test<-predict(bag.mod,newdata=data.test)
err(data.test$spam,pred.test)
```

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=nrotree,importance=TRUE)
rf.mod
```

```
Call:
 randomForest(formula = spam ~ ., data = data.train, classwt = c(0.606,      0.394), mtry = 5, ntree = n
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 5

          Mean of squared residuals: 0.07646554
                    % Var explained: 68.24
```

```r
pred.train<-predict(rf.mod,newdata=data.train)
err(data.train$spam,pred.train)

pred.test<-predict(rf.mod,newdata=data.test)
err_1 <- err(data.test$spam,pred.test)

kable(err_1$table)
```

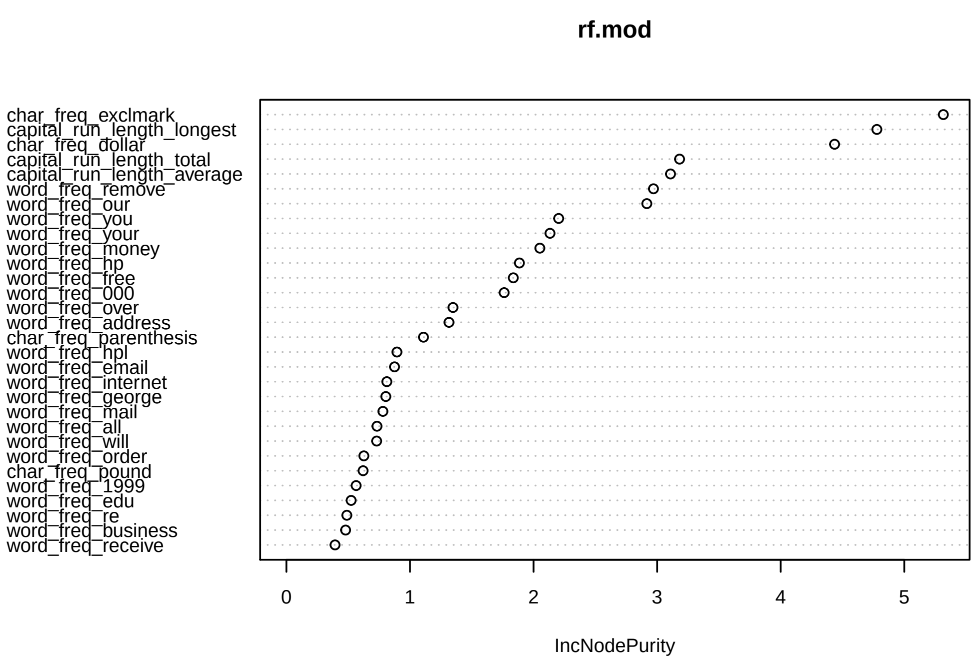
```
Warning in kable_pipe(x = structure(character(0), .Dim = c(0L, 0L), .Dimnames =
list(: The table should have a header (column names)
```

```r
kable(err_1$numeric)
```

```
Warning in kable_pipe(x = structure(character(0), .Dim = c(0L, 0L), .Dimnames =
list(: The table should have a header (column names)
```

```r
#variable importance plot
#two measures are reported:
#increase in MSE computed on OOB samples when removing variable
#decrease in node impurity (or increase in node purity) resulting from splits on the variable
importance(rf.mod)
```

```r
varImpPlot(rf.mod,type=2,cex=.7)
```



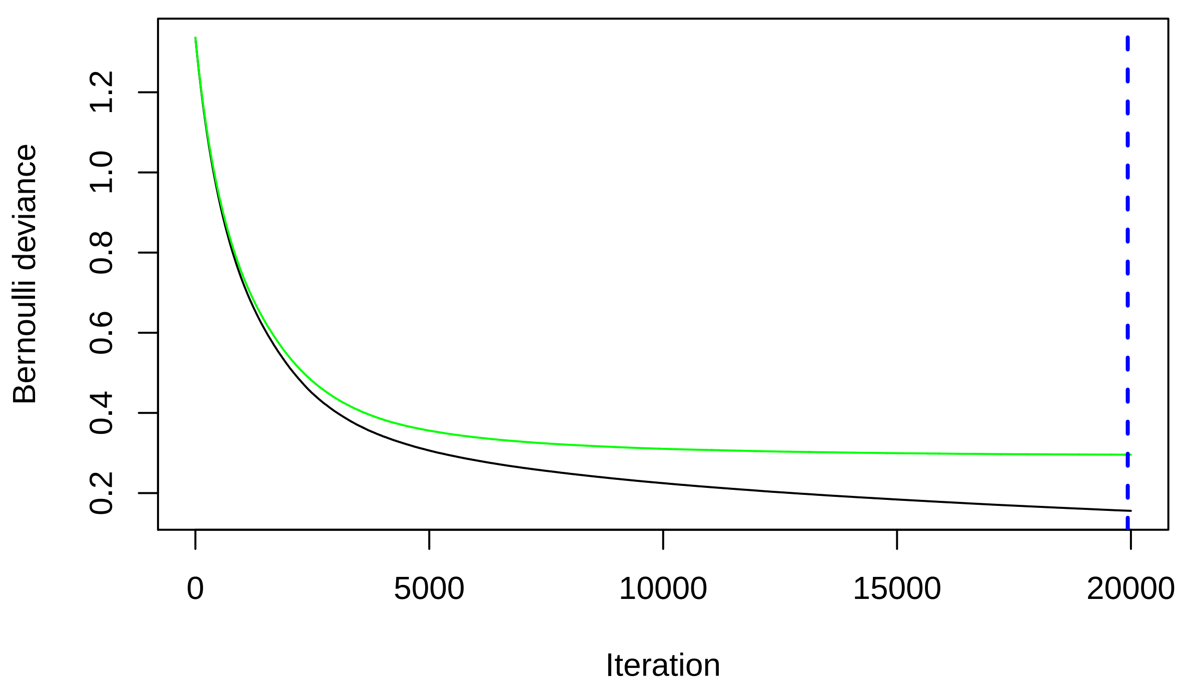
Random Forests Model is also based on the idea of decorrelating trees. Here we choose m=5 for the size of our predictor set. can see that although there is slightly difference, "$", "!" and "remove" still stand for spam webmails, as well as the length of the longest uninterrupted sequence of capital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

#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)
```

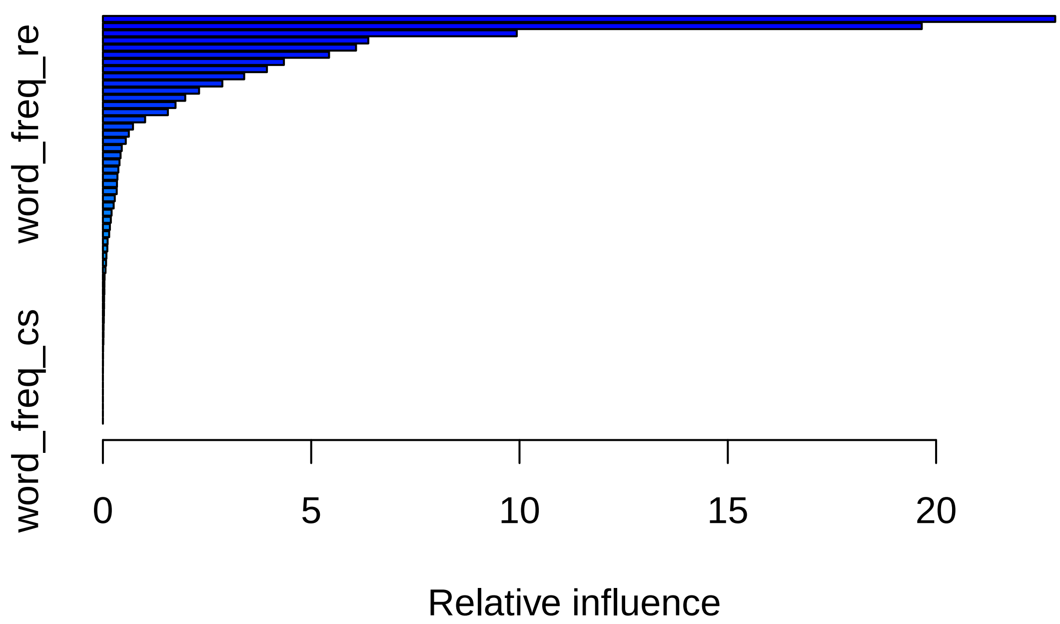
```
#data.train$spam <- as.factor(data.train$spam)
#boost.mod=gbm(spam~., distribution="bernoulli", data=data.train, n.trees=20000, interaction.depth=4,
#              shrinkage=0.001,cv.folds=5)
#boost.mod=gbm::gbm(spam~., distribution="bernoulli", data=data.train, n.trees=20000, interaction.depth
#              shrinkage=0.001,cv.folds=1)
load("~/GitHub/kul-multivariate-a2/save_objects.Rdata")
gbm.perf(boost.mod_save,method="cv")
```



```
## [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_save,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_save,n.trees=19932,newdata=data.train,type="response")
classif.train<-ifelse(pred.train>=1-pred.train,1,0)
err(data.train$spam,classif.train)
pred.test<-predict(boost.mod_save,n.trees=19932,newdata=data.test,type="response")
classif.test<-ifelse(pred.test>=1-pred.test,1,0)
err(data.test$spam,classif.test)
```

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

```
#(b)Account for different prior probabilities and unequal classification costs: C(spam|email)=10*C(emai
# (1) linear discriminant analysis
set.seed(0829539)
data.train$spam<-factor(ifelse(data.train$spam==1,"1_yes","0_no"))
data.test$spam<-factor(ifelse(data.test$spam==1,"1_yes","0_no"))
spamdata$spam<-factor(ifelse(spamdata$spam==1,"1_yes","0_no"))
print(data.train$spam)
```

```
  [1] 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no
 [16] 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no
 [31] 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no
 [46] 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no
 [61] 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no
 [76] 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no
 [91] 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no
[106] 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no
[121] 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no
[136] 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no
[151] 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no
[166] 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no
[181] 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no
[196] 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no
[211] 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no
[226] 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no
[241] 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no 0_no
Levels: 0_no
```

```
#lda.out<- MASS::lda(spam~.,prior=c(0.606,0.394),data=data.train)
#print(lda.out)

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

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