Assignment3

QINGTAN

18/10/2022

Question 1

1.(a)

Firstly, we need to load the train and test datasets:

```
library(MASS)
trainRain = read.table('/Users/apple/Desktop/MAST90138/XGtrainRain.txt', sep=",", header = T)
testRain = read.table('/Users/apple/Desktop/MAST90138/XGtestRain.txt', sep=",", header = T)
```

Then we do the logistic regression on our training dataset:

```
logistic_model = glm(G ~., data=trainRain, family=binomial(link='logit'))
#summary(logistic_model)
logistic_model$coefficients[151:365]
```

```
## X150 X151 X152 X153 X154 X155 X156 X157 X158 X159 X160 X161 X162 X163 X164 X165
##   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## X166 X167 X168 X169 X170 X171 X172 X173 X174 X175 X176 X177 X178 X179 X180 X181
##   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## X182 X183 X184 X185 X186 X187 X188 X189 X190 X191 X192 X193 X194 X195 X196 X197
##   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## X198 X199 X200 X201 X202 X203 X204 X205 X206 X207 X208 X209 X210 X211 X212 X213
##   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## X214 X215 X216 X217 X218 X219 X220 X221 X222 X223 X224 X225 X226 X227 X228 X229
##   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## X230 X231 X232 X233 X234 X235 X236 X237 X238 X239 X240 X241 X242 X243 X244 X245
##   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## X246 X247 X248 X249 X250 X251 X252 X253 X254 X255 X256 X257 X258 X259 X260 X261
##   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## X262 X263 X264 X265 X266 X267 X268 X269 X270 X271 X272 X273 X274 X275 X276 X277
##   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## X278 X279 X280 X281 X282 X283 X284 X285 X286 X287 X288 X289 X290 X291 X292 X293
##   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## X294 X295 X296 X297 X298 X299 X300 X301 X302 X303 X304 X305 X306 X307 X308 X309
##   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## X310 X311 X312 X313 X314 X315 X316 X317 X318 X319 X320 X321 X322 X323 X324 X325
##   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## X326 X327 X328 X329 X330 X331 X332 X333 X334 X335 X336 X337 X338 X339 X340 X341
```

```
##   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## X342 X343 X344 X345 X346 X347 X348 X349 X350 X351 X352 X353 X354 X355 X356 X357
##   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## X358 X359 X360 X361 X362 X363 X364
##   NA   NA   NA   NA   NA   NA   NA
```

The logistic regression algorithm is not converge, I saw the summary of the logistic_model, while the output is too long to show, I got that the coefficients from X150 to X364 are NA, because the number of variables is greater than the number of our training samples.

Then we try the QDA classifier algorithm:

```
#qda_model = qda(G ~., data=trainRain)
```

When using quadratic discriminant algorithm, I got an error said some groups is too small for this algorithm. It is also because we have more variables than the training samples.

In conclusion, I do not recommond these two methods, because we have more variables than training samples, which will not converge in logistic regression, and some group is too small for discriminant algorithm.

1.(b)

Firstly, we get PCA by a function:

```
library(pls)
```

```
##
## Attaching package: 'pls'
```

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

```
PCA = prcomp(trainRain[, 1:365], retx=T)

# Then we check the first five elements value
PCA$x[1:5]
```

```
## [1] -151.1118 1013.6886  480.6649  439.7710 -157.1982
```

Then we get PCA by hand:

```
PCA_vec = PCA$rotation
train = as.matrix(trainRain)
test = as.matrix(testRain)
Xbar = matrix(rep(1,150), nrow=150) %*% colMeans(train[,1:365])
train_center = train[,-366] - Xbar
PCA_hand = train_center %*% PCA_vec

# Then we check the first five elements value
PCA_hand[1:5]
```

```
## [1] -151.1118 1013.6886  480.6649  439.7710 -157.1982
```

Because the first five values in PCA generated by function and by hand are exactly the same, so the manually calculation is successful.

Then we comes to the PLS operation:

```
# by function
PLS = plsr(G ~., data=trainRain)

# by hand
PLS_proj = PLS$projection
PLS_hand = train_center %*% PLS_proj
```

Then the first five elements generated by function is:

```
PLS$scores[1:5]
```

```
## [1]  102.0531 -971.5203 -368.5665 -342.0121  132.9166
```

Then the first five elements generated by hand is:

```
PLS_hand[1:5]
```

```
## [1]  102.0531 -971.5203 -368.5665 -342.0121  132.9166
```

The first five elements are the same, so our PLS calculated by hand is successful.

1.(c)

Firstly, we consider PCA method on QDA algorithm, use the PCA we calculated by hand

```
goal = train[,366]
PCA_data = as.data.frame(PCA_hand)
CV_value = c(rep(0, 50))
for (m in 2:50){
  num = 0
  for (n in 1:150){
    cv_train = PCA_data[-n, 1:m]
    cv_goal = goal[-n]
    cv_traindata = data.frame(cbind(cv_train, cv_goal))
    cv_test = data.frame(cbind(PCA_data[n, 1:m], goal[n]))
    cv_qda = qda(cv_goal ~., data=cv_traindata)
    qda_pred = predict(cv_qda, newdata=cv_test)$class
    if (qda_pred != goal[n]){
      num = num + 1
    }
  }
  CV_value[m] = num
}
best_one = min(which(CV_value[2:50] == min(CV_value[2:50])))+1
```

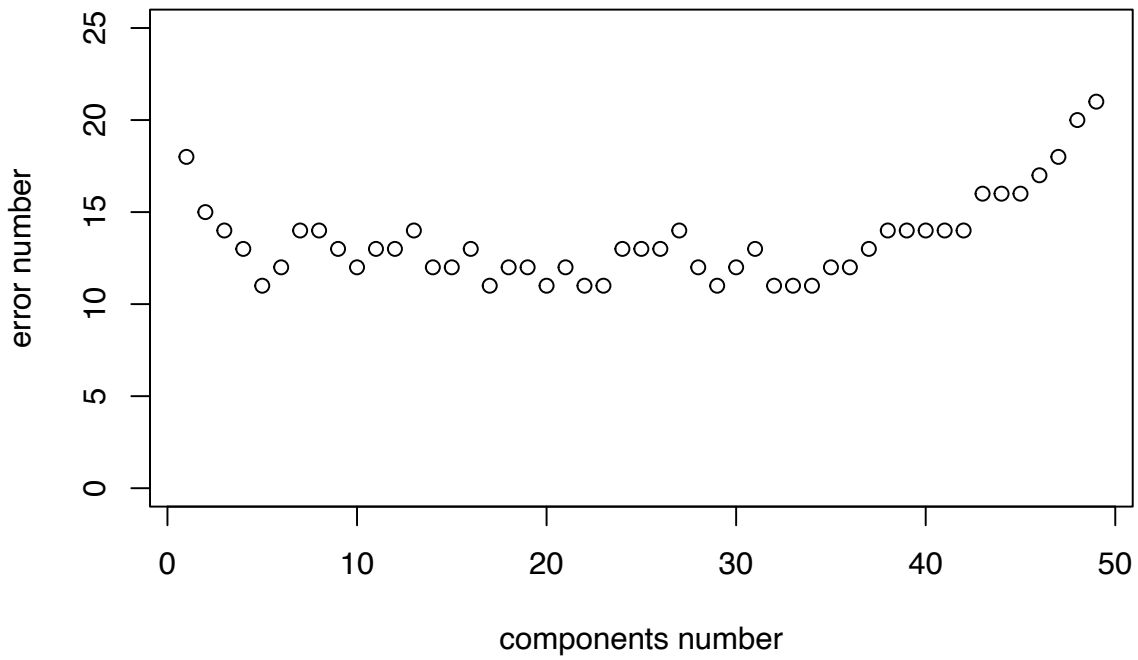
Then we have the best result at this position

```
best_one
```

```
## [1] 6
```

Then we draw the error number plot:

```
plot(CV_value[2:50], ylim=c(0,25), xlab = "components number",ylab="error number")
```



So as in the plot shown, we choose 6 components to get the best model.

Then we consider PLS method on QDA algorithm, use the PCA we calculated by hand

```
PLQACV_value = c(rep(0, 50))
PLS_data = as.data.frame(cbind(PLS_hand, goal))
for (m in 2:50){
  num = 0
  for (n in 1:150){
    plcv_train = PLS_data[-n, 1:m]
    plcv_goal = goal[-n]
    plcv_traindata = data.frame(cbind(plcv_train, plcv_goal))
    plcv_test = data.frame(PLS_data[n, 1:m])
    plcv_qda = qda(plcv_goal ~., data=plcv_traindata)
    plqda_pred = predict(plcv_qda, plcv_test)$class
    if (plqda_pred != goal[n]){
```

```
      num = num + 1
    }
  }
  PLQACV_value[m] = num
}
best_one = min(which(PLQACV_value[2:50] == min(PLQACV_value[2:50])))+1
```

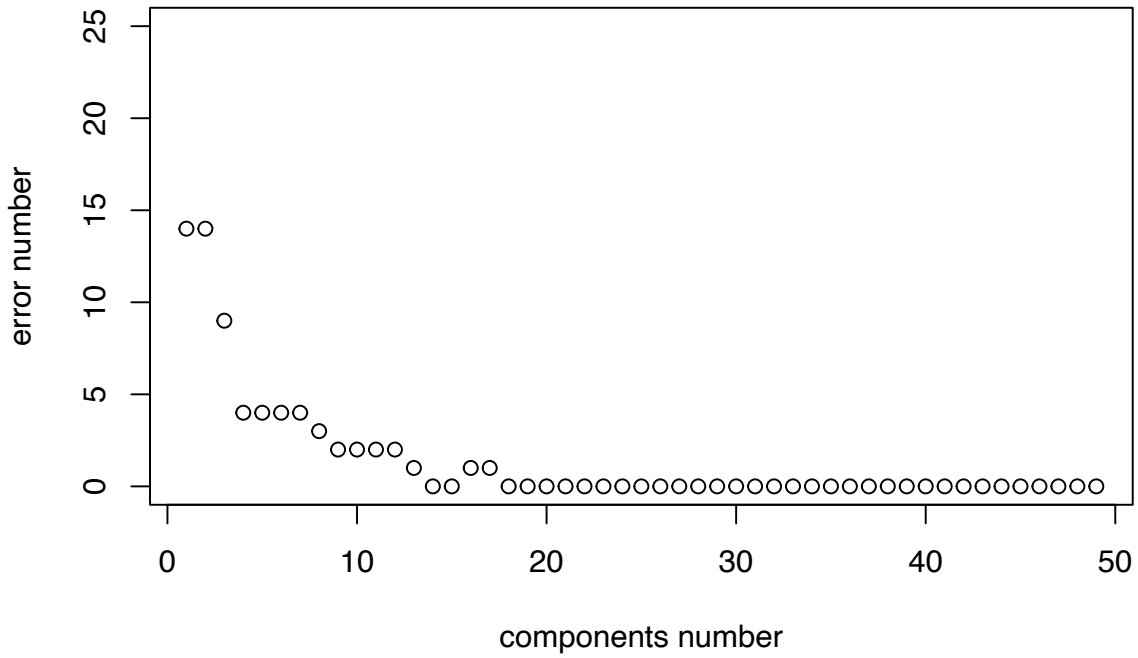
Then we have the best result at this position

```
best_one
```

```
## [1] 15
```

Then we draw the error number plot:

```
plot(PLQACV_value[2:50], ylim=c(0,25), xlab = "components number",ylab="error number")
```



So as in the plot shown, we choose 15 components to get the best model.

Then we use PCA method in the logistic regression algorithm.

```
PCLOCV_value = c(rep(0, 50))
for (m in 2:50){
  num = 0
  for (n in 1:150){
```

```
    precision = 1
    cv_train = PCA_data[-n, 1:m]
    cv_goal = goal[-n]
    cv_traindata = as.data.frame(cbind(cv_train, cv_goal))
    cv_test = as.data.frame(PCA_data[n, 1:m])
    logPCA = glm(cv_goal ~., data=cv_traindata, family = binomial(link="logit"))
    b = logPCA$coefficients[-1]
    b_0 = logPCA$coefficients[1]
    result = b_0 + b %*% t(cv_test)
    if (result <= 0){
      precision = 0
    }
    if (precision != goal[n]){
      num = num + 1
    }
  }
  PCLOCV_value[m] = num
}
best_one = min(which(PCLOCV_value[2:50] == min(PCLOCV_value[2:50])))+1
```

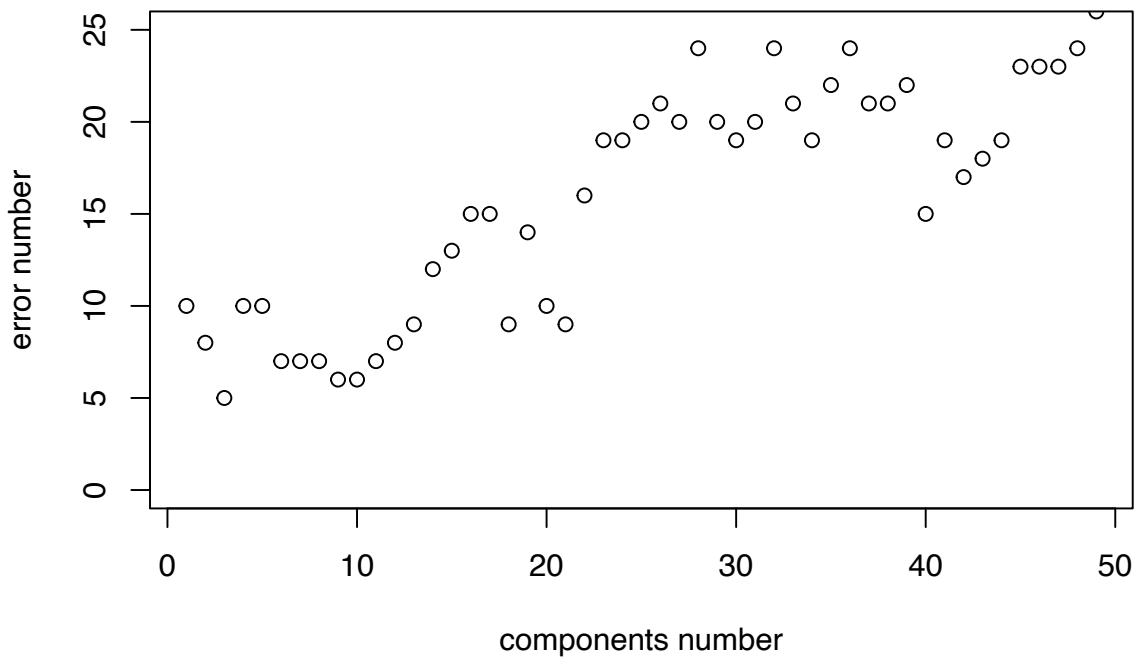
Then we have the best result at this position

```
best_one
```

```
## [1] 4
```

Then we draw the error number plot:

```
plot(PCLOCV_value[2:50], ylim=c(0,25), xlab = "components number",ylab="error number")
```



So as in the plot shown, we choose 4 components to get the best model.

Then we use PLS method in the logistic regression algorithm.

```
PLLOCV_value = c(rep(0, 50))
for (m in 2:50){
  num = 0
  for (n in 1:150){
    precision = 1
    pllcv_train = PLS_data[-n, 1:m]
    pllcv_goal = goal[-n]
    pllcv_traindata = as.data.frame(cbind(pllcv_train, pllcv_goal))
    pllcv_test = as.data.frame(PLS_data[n, 1:m])
    logPL = glm(pllcv_goal ~., data=pllcv_traindata, family = binomial(link="logit"))
    b = logPL$coefficients[-1]
    b_0 = logPL$coefficients[1]
    result = b_0 + b %*% t(pllcv_test)
    if (result <= 0){
      precision = 0
    }
    if (precision != goal[n]){
      num = num + 1
    }
  }
  PLLOCV_value[m] = num
}
best_one = min(which(PLLOCV_value[2:50] == min(PLLOCV_value[2:50])))+1
```

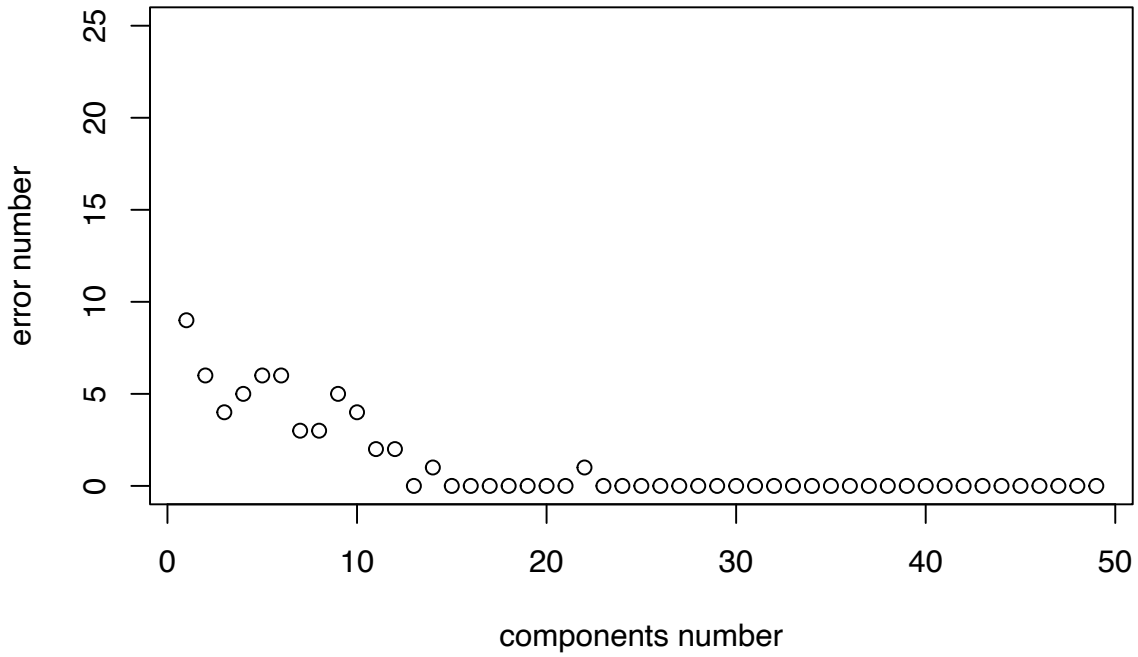
Then we have the best result at this position

```
best_one
```

```
## [1] 14
```

Then we draw the error number plot:

```
plot(PLLOCV_value[2:50], ylim=c(0,25), xlab = "components number",ylab="error number")
```



So as in the plot shown, we choose 14 components to get the best model.

1.(d)

I prefer PLS than PCA method in both QDA and logistic regression. From the plots we generated above, when checking the difference between PLS and PCA, we found that the leave one out error is smaller when using PLS than using PCA on both QDA and logistic regression algorithm. The leave one out error approaches to zero at most time. Another reason is that QDA is not a linear model, while PCA is linear transformation. So PLS is much better when using QDA method. Also PCA is unsupervised technique, while PLS is supervised technique. Because logistic regression is supervised algorithm. Therefore PLS is more suitable for logistic regression.

1.(e)

Firstly, we preprocess our test dataset

```
testdata = as.matrix(testRain)
testPCA = as.data.frame((testdata[,-366] - Xbar[1:41,])%*%PCA_vec)
testPLS = as.data.frame((testdata[,-366] - Xbar[1:41,])%*%PLS$projection)
testdim = 41
test_goal = testdata[, 366]
```

After that we apply PCA to logistic regression:

```
logm1 = glm(goal ~., data = PCA_data[,1:4], family = binomial(link="logit"))
b0 = logm1$coefficients[1]
b = logm1$coefficients[-1]
result = b%*%t(testPCA[, 1:4]) + b0
getPCAlog = c(rep(1,41))
getPCAlog[result < 0] = 0
PCAlogE = sum(getPCAlog != test_goal)
# get the percentage
PCAlogEper = PCAlogE / testdim
PCAlogEper
```

```
## [1] 0.02439024
```

So when using PCA method on logistic regression, the test error rate is 0.0244

Then we apply PCA to QDA approach:

```
QDAm1 = qda(goal ~.,data = PCA_data[,1:6])
getPCAqda = predict(QDAm1, testPCA[,1:6])$class
PCAqdaE = sum(getPCAqda != test_goal)
PCAqdaper = PCAqdaE / testdim
PCAqdaper
```

```
## [1] 0.09756098
```

So when using PCA method on QDA approach, the test error rate is 0.0976

Then we apply PLS to logistic regression:

```
logm2 = glm(goal ~., data = PLS_data[,1:14], family = binomial(link="logit"))
b0 = logm2$coefficients[1]
b = logm2$coefficients[-1]
result = b%*%t(testPLS[, 1:14]) + b0
getPLSlog = c(rep(1,41))
getPLSlog[result < 0] = 0
PLSlogE = sum(getPLSlog != test_goal)
# get the percentage
PLSlogEper = PLSlogE / testdim
PLSlogEper
```

```
## [1] 0.1219512
```

So when using PLS method on logistic regression, the test error rate is 0.1220

Then we apply PLS to QDA approach:

```
QDAm2 = qda(goal ~.,data = PLS_data[,1:15])
getPLSqda = predict(QDAm2, testPLS[,1:15])$class
PLSqdaE = sum(getPLSqda != test_goal)
PLSqdaper = PLSqdaE / testdim
PLSqdaper
```

```
## [1] 0.09756098
```

So when using PLS method on logistic regression, the test error rate is 0.0976

2.

2.(a)

Firstly, we set a seed to 50:

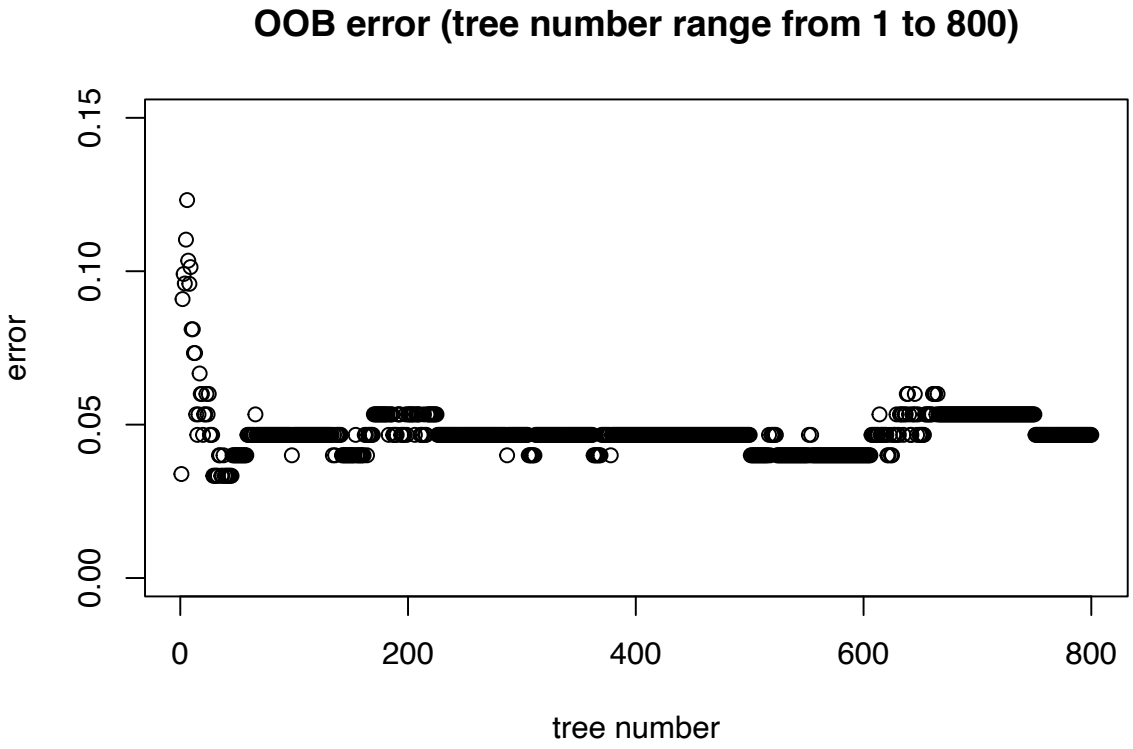
```
set.seed(100)
library(randomForest)
```

```
## randomForest 4.7-1.1
```

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

Then we set tree number from 1 to 800, to get a approximate range for the best tree number

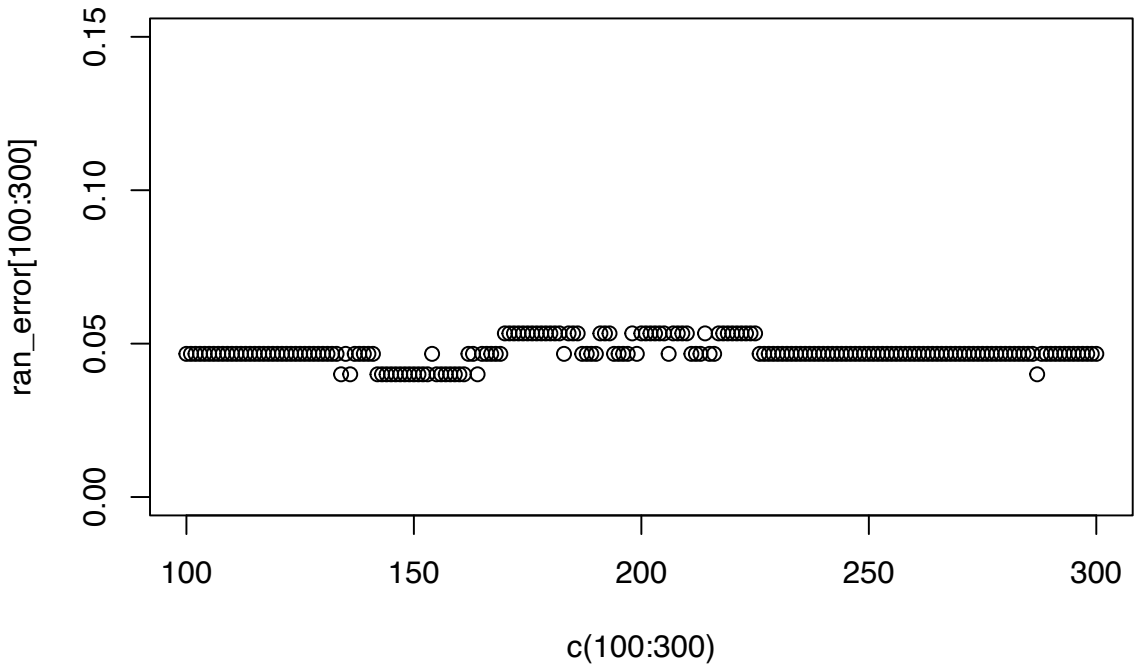
```
randomF = randomForest(as.factor(goal) ~., data=train, ntree=800, importance=TRUE)
ran_error = randomF$err.rate[, 1]
plot(1:800, ran_error, xlab='tree number', ylab='error', ylim=c(0,0.15), main='OOB error (tree number ra
```



OOB error is the percentage that the highest vote is not equal to the correct class divided by the total cases.

So from this plot, we found that if the tree number is too small, it will cause an underfitting problem. And if the tree number is too large, it will cause an overfitting problem. It is shown that when the number of trees is greater than 100, the error decreases smoothly. Also the tree number cannot be large, so we believe that the best tree number between 100 and 300, then we get a more clear plot:

```
plot(c(100:300),ran_error[100:300],ylim=c(0,0.15))
```

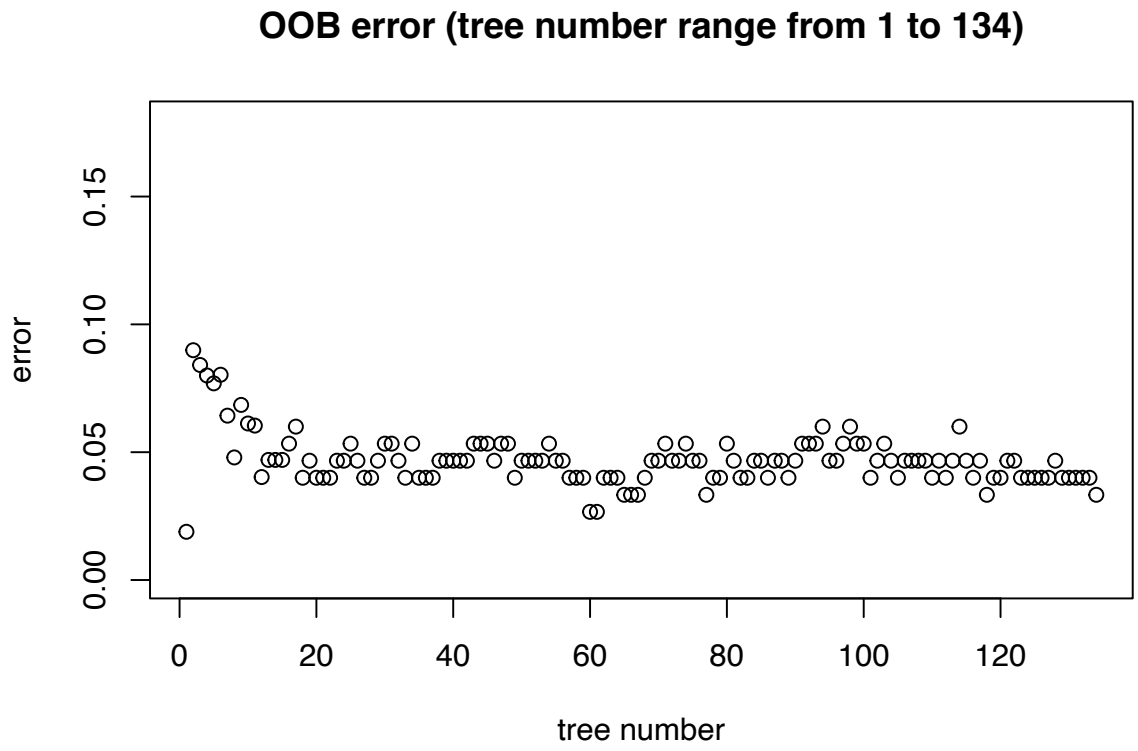



```
best_num = which((ran_error[100:300] == min(ran_error[100:300])))+100-1
best_num = best_num[1]
```

We find when we obtain the minimum error value, the tree number has more than one value, for example, it can be 134, 136 or 142, etc. We just choose the smallest one, so the best tree number is 134.

Then we apply 134 tree number to our final plot:

```
randomF1 = randomForest(as.factor(goal) ~., data=train, ntree=best_num, importance=TRUE)
ran_error1 = randomF1$err.rate[, 1]
plot(1:134, ran_error1, xlab='tree number', ylab='error', ylim=c(0,0.18), main='OOB error (tree number
```

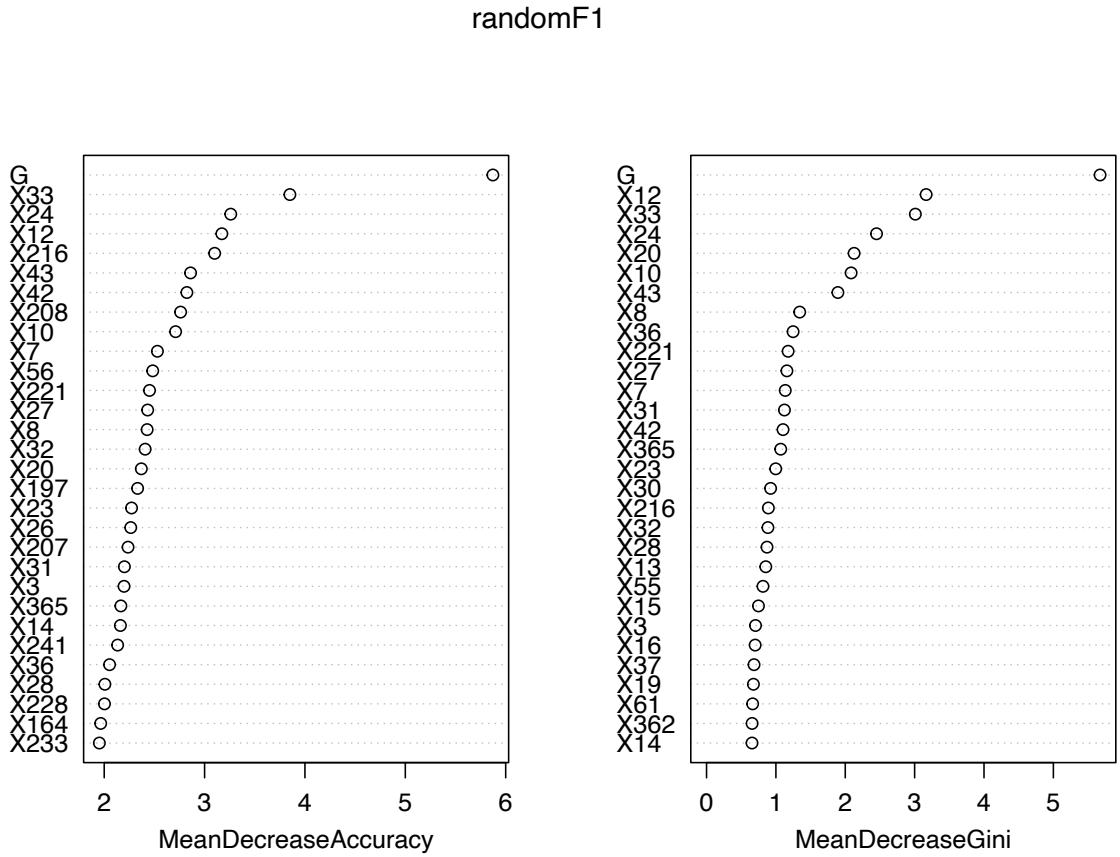


So here is the final plot with our best tree number, which is 134

2.(b)

Here we show the plots:

```
varImpPlot(randomF1)
```



This Mean Decrease Accuracy plot shows the information about the model accuracy value losses by deleting each variable. If the accuracy only changes a little after deleting one variable, it means that this variable is not important for the successful classification. Another condition is the accuracy changes a lot after deleting one variable, it means that this variable is important for the successful classification. Mean Decrease Gini plot shows the average of the total decrease of a variable in node impurity. Also after deleting one important variable, the losses will be large. After deleting an unimportant variable, the losses will be small. X_j is the rainfall quantity in jth day. The plot shows that days in summer is more important than others. Due to the different climate, there are some differences in rainfall quantity between north and south in summer. Also we can find if the rain remains several days, the first day of this rain is more important than other days.

2.(c)

We run our code for 50 times to get 50 error values:

```
set.seed(50)
error_list = c(rep(0, 50))

for(i in 1:50){
  RandomFm = randomForest(as.factor(goal) ~., data=train, ntree=best_num, importance=TRUE)
  RandomPre = predict(RandomFm, newdata=test)
  error_list[i] = sum(RandomPre != test[,366])/41
}
error_list
```

```
##  [1] 0.02439024 0.04878049 0.02439024 0.02439024 0.02439024 0.02439024
##  [7] 0.04878049 0.02439024 0.04878049 0.02439024 0.02439024 0.02439024
## [13] 0.02439024 0.04878049 0.07317073 0.02439024 0.04878049 0.02439024
## [19] 0.04878049 0.07317073 0.02439024 0.02439024 0.02439024 0.04878049
## [25] 0.02439024 0.02439024 0.02439024 0.02439024 0.02439024 0.02439024
## [31] 0.04878049 0.02439024 0.04878049 0.02439024 0.02439024 0.04878049
## [37] 0.02439024 0.02439024 0.02439024 0.02439024 0.02439024 0.02439024
## [43] 0.02439024 0.02439024 0.02439024 0.04878049 0.04878049 0.02439024
## [49] 0.02439024 0.02439024
```

From these error list, we found that we cannot get the same classfication error all the time. Because each time our random forest classifier can be different. The method to make the random forest more stable is that we can increase the number of trees. Because adding trees means adding classification. So I can get more accurate estimates by using OOB predictions if the tree number increases.

3.

Logistic + PCA error:

```
PCAlogEper
```

```
## [1] 0.02439024
```

Logistic + PLS error:

```
PLSlogEper
```

```
## [1] 0.1219512
```

QDA + PCA error:

```
PCAqdaper
```

```
## [1] 0.09756098
```

QDA + PLS error:

```
PLSqdaper
```

```
## [1] 0.09756098
```

Random Forest error (we use the average value of the 50 errors):

```
RF_error = (sum(error_list) / 50)
RF_error
```

```
## [1] 0.03219512
```

The smallest test error is (Logistic + PCA), the value is 0.0244, which is the best one of these five methods. The test error of (Logistic + PLS) is the biggest one, the value is 0.1220, which is the worst method. As we said in Q1, PCA is linear transformation. And our data is linearly separable. The second best one is the Random Forest method, whose error is 0.0322, just a little larger than the error of (Logistic + PCA), So Random Forest method is also suitable for this dataset. Another reason is that we has already chosen the reliable tree number from a large range (from 1 to 800). The (QDA + PCA) and (QDA + PLS) methods have the same error, which is 0.0976. Choosing PCA or PLS in QDA method do not effect its error result obviously, neither of them are good. Also QDA generates a non-linear quadratic decision boundary, while logistic regression generates a linear decision boundary. So when the dataset has linear boundaries in reality, the linear boundary generation approach, such as logistic regression, will show a better performance.