Lab 02

Group A20

2022-12-03

Assignment 1

```r
set.seed(12345)
data1 <- read.csv("tecator.csv")
n <- nrow(data1)
sample50 <- sample(n, round(0.5 * n))

data1_train <- data1[sample50,]
data1_test <- data1[-sample50,]
# Divide data
```

Task 1

```r
lm_fat <- glm(Fat ~ . -Sample -Protein -Moisture,
              data = data1_train, family = gaussian())

pred_train <- predict(lm_fat, newdata = data1_train)
pred_test <- predict(lm_fat, newdata = data1_test)

err_train <- sum((pred_train - data1_train$Fat)^2) / (2*length(pred_train))
err_test <- sum((pred_test - data1_test$Fat)^2) / (2*length(pred_test))

err_train
```

```
## [1] 0.003554318
```

```r
err_test
```

```
## [1] 317.8356
```

- Overfitted! Works well on train set but can't fit the test set at all. We need a better method to predict Fat.

Task 2

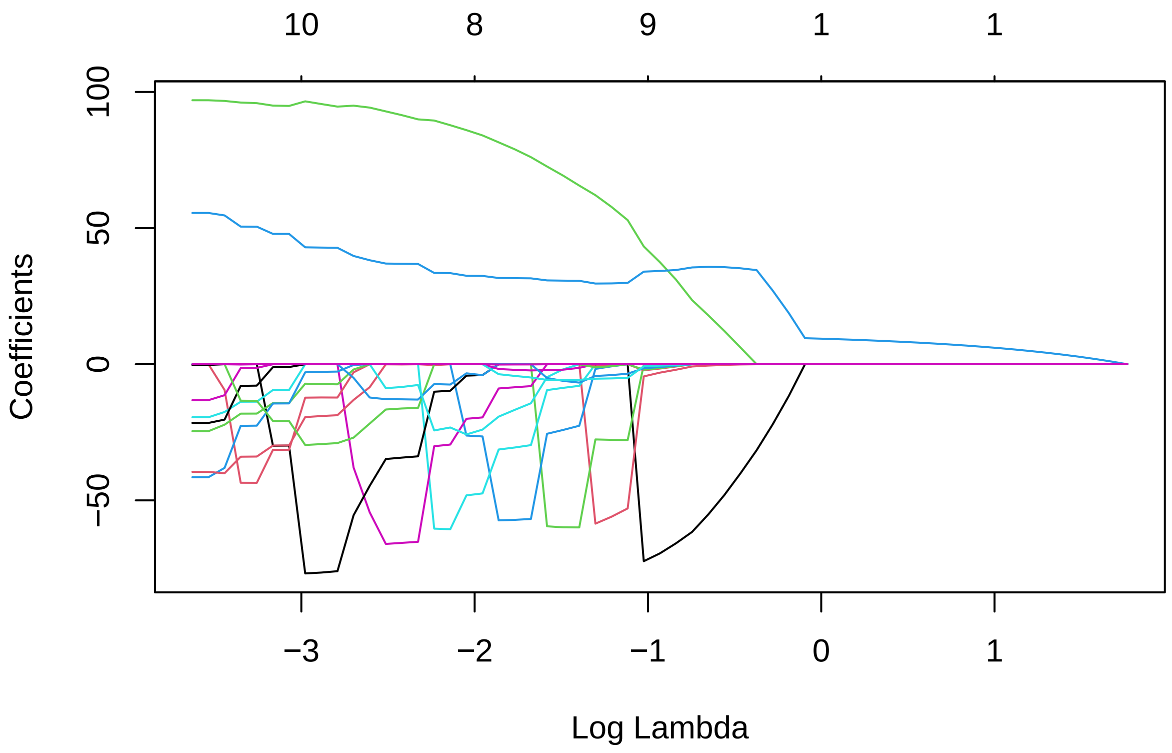
```r
theta_hat_cost <- function(X, y, lambda, theta){
  # WITH intercept
  theta_hat <- sqrt(sum((X %*% theta - y)^2)) / length(theta) + lambda * sum(theta)
}
```

- cost function(with squared error loss):

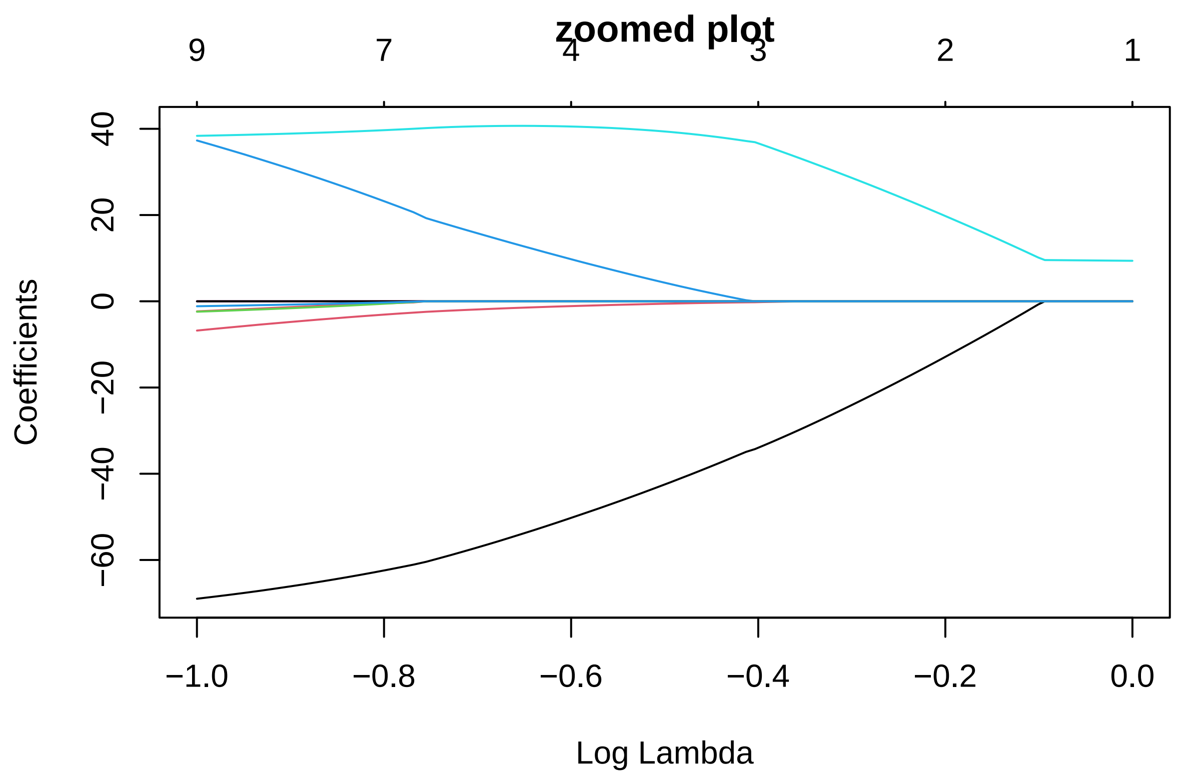
$$\hat{\theta} = \frac{1}{n}||y - X\theta||_2^2 + \lambda||\theta||_1$$

Task 3

```
glm_train <- glmnet(data1_train[,2:101], data1_train[,102],
                    family = "gaussian",alpha = 1)
plot(glm_train, xvar = "lambda")
```



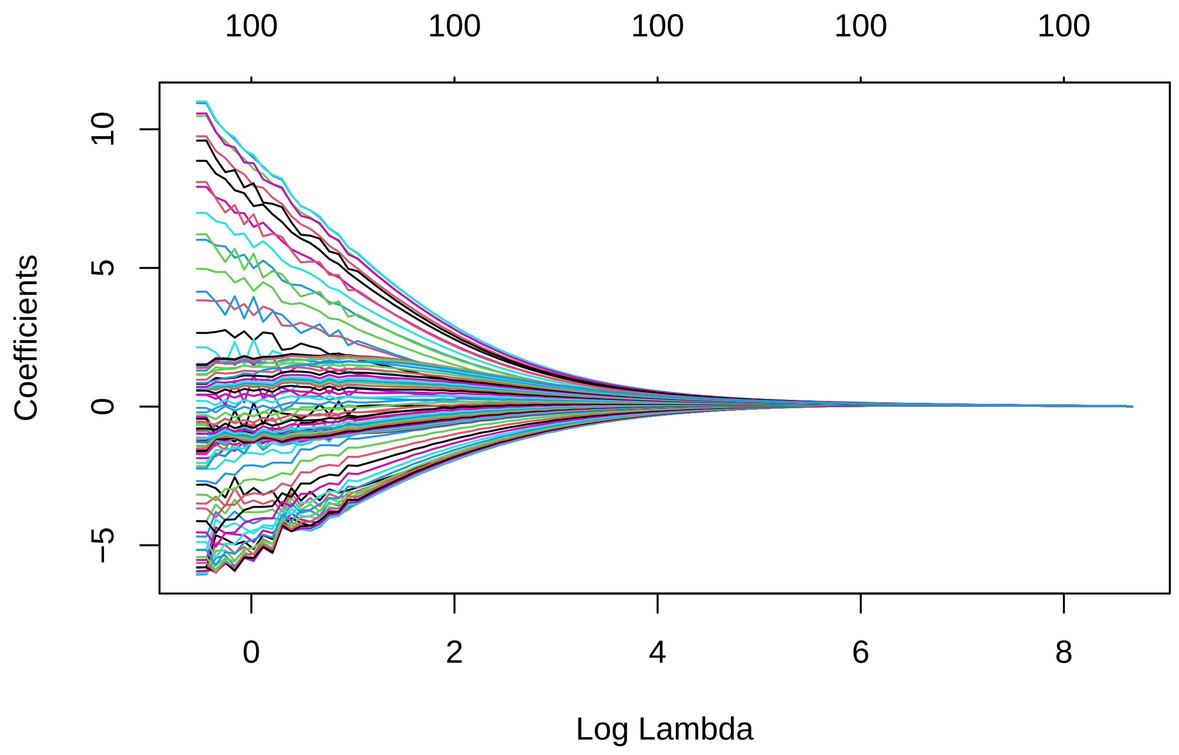
```
zglm_train <- glmnet(data1_train[,2:101], data1_train[,102],
                     family = "gaussian",alpha = 1,
                     lambda = seq(exp(-1),1, length.out = 100))
plot(zglm_train, xvar = "lambda", main = "zoomed plot")
```



- when λ goes greater, the coefficients will all reduce to 0.While most coefficients is or close to 0 all the way. From the plot, we can choose λ which $ln(\lambda)$ close to 0.5 if we want to select a model with only three features.

Task 4

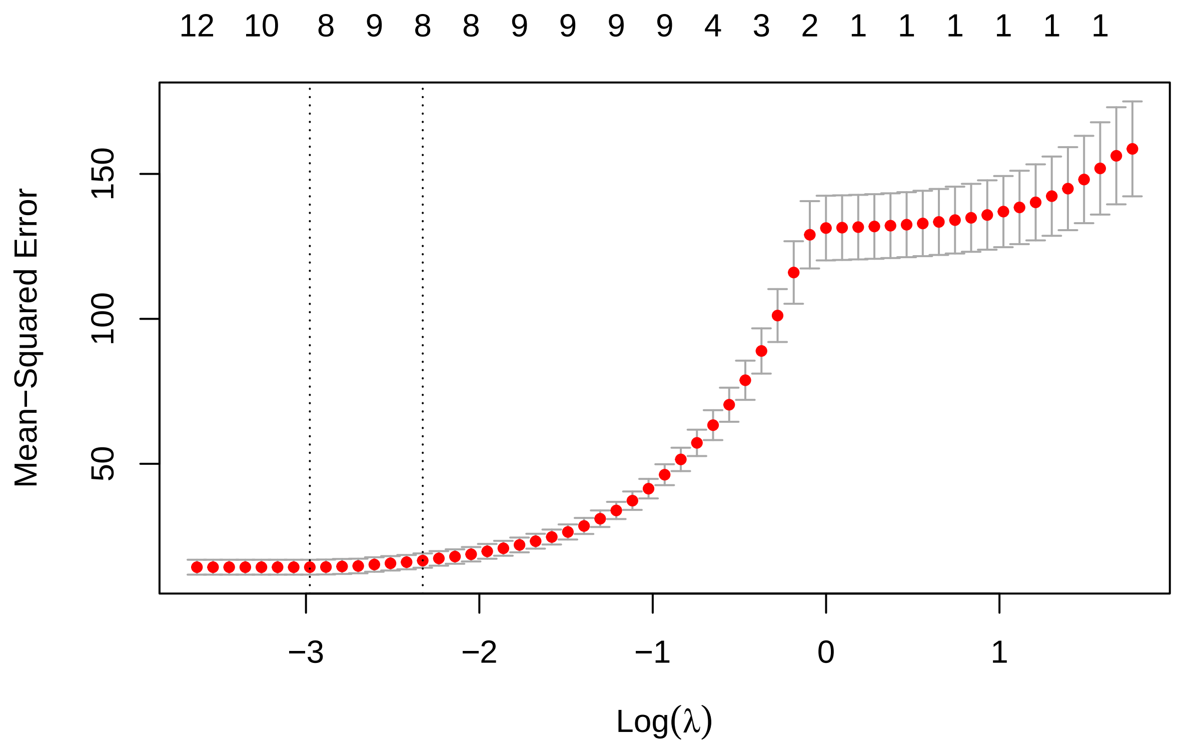
```
glm_train_ridge <- glmnet(data1_train[,2:101], data1_train[,102],
                          family = "gaussian", alpha = 0)
plot(glm_train_ridge, xvar = "lambda")
```



- greater lambda value of ridge will also converge all coefficients to 0, but ridge don not tend to filter some coefficients to 0 when lambda value is small, instead it scales them in relatively similar rates.

Task 5

```
glm_train_lassoCV <- cv.glmnet(as.matrix(data1_train[,2:101]),
                               as.matrix(data1_train[,102]),
                               family = "gaussian",alpha = 1)
plot(glm_train_lassoCV)
```



```r
nlambda.1se <- which(glm_train_lassoCV$glmnet.fit$lambda == glm_train_lassoCV$lambda.1se)
nlambda.min <- which(glm_train_lassoCV$glmnet.fit$lambda == glm_train_lassoCV$lambda.min)
glm_train_lassoCV[["glmnet.fit"]][["df"]][c(nlambda.1se, nlambda.min)]
```

```
## [1]  8 10
```

```r
glm_train_lassoCV$lambda.1se
```

```
## [1] 0.09763481
```

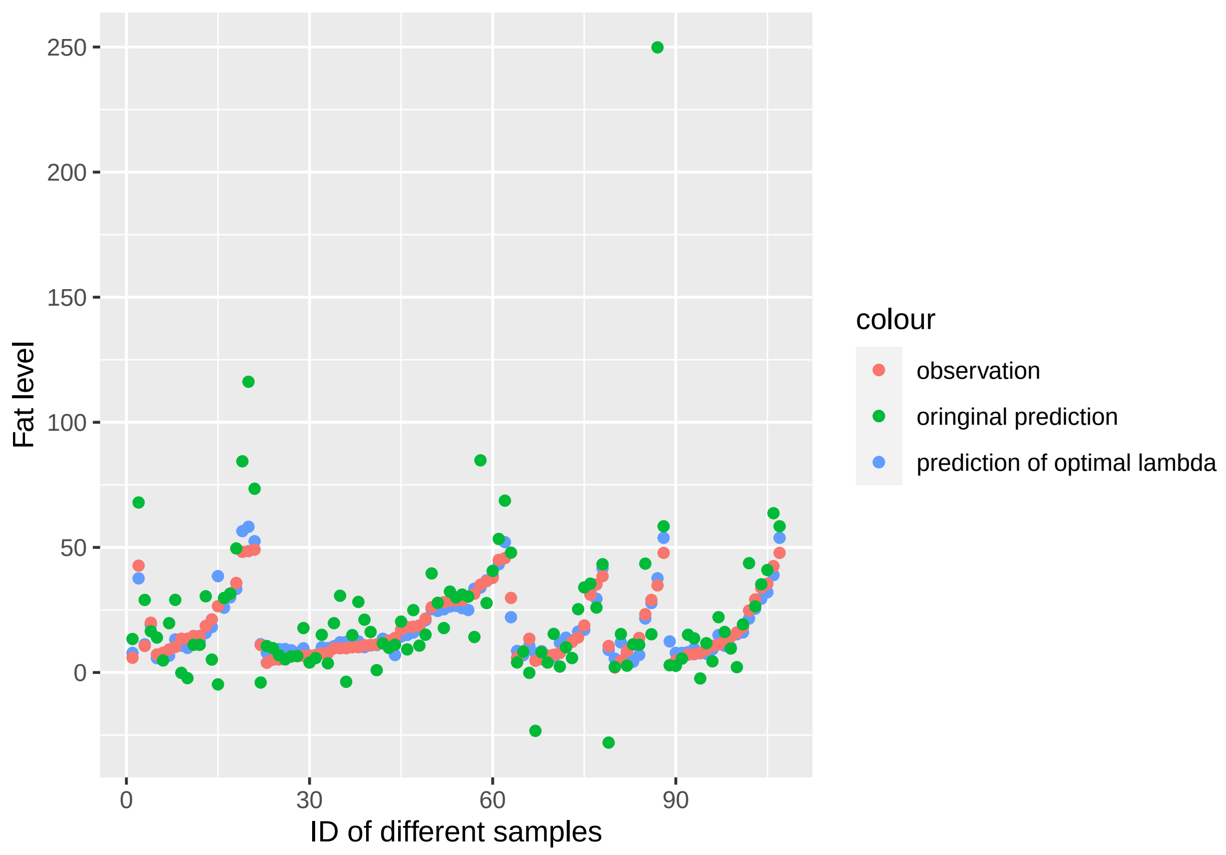
```r
glm_train_lassoCV$lambda.min
```

```
## [1] 0.05090687
```

- the value of lambda we found that gives minimum cvm is 0.06593502, but it is not significantly better than $log(\lambda) = -4$, the MSE doesn't seems have much difference when $log(\lambda) \leq -2$. So we choose lambda.1se as optimal λ value which maximize the non-zero coefficients number.

```r
pred_test_1se <- predict(glm_train_lassoCV, newx = as.matrix(data1_test[,2:101]))

ggplot(data = data.frame(pred_test_1se, obs = data1_test[,102], origin = pred_test))+
  geom_point(aes(x = 1:nrow(data1_test), y = lambda.1se, color = "prediction of optimal lambda"))+
  geom_point(aes(x = 1:nrow(data1_test), y = obs, color = "observation"))+
  geom_point(aes(x = 1:nrow(data1_test), y = origin, color = "oringinal prediction"))+
  labs(x  = "ID of different samples", y = "Fat level")
```



- We can see that the differences between points of observation and optimized prediction is close, and much more better than the origin predictions. So we consider the optimized one a good and better prediction.

Assignment 2. Decision trees and logistic regression for bank marketing

task1

The code is given in APPENDIX

task2

misclassification rates for the training and validation data

```
##                 train_error valid_error
## default          0.10484406   0.1092679
## nodesize_7000    0.10484406   0.1092679
## mindev_0.0005    0.09400575   0.1119221
```

According to the misclassification rates above we can see that the first and second models can be regarded as the best ones. This is becauses for these two models, the valid_error are pretty small than the third model.Obviously, the third model overfits the training data set(the training error is relatively smaller) and thus cause valide_error larger.

task3

Use training and validation sets to choose the optimal tree depth in the model 2c: study the trees up to 50 leaves. Present a graph of the dependence of deviances for the training and the validation data on the number of leaves and interpret this graph in terms of bias-variance tradeoff.

From the plot we can see that as the number of numbers increases, the deviance of the training data decreases constantly. But it is not the case for deviance of validation data set.we can notice that at first, the deviance of validation data set decreases but after about 20 it begins to increase. This is because as the number of leaves increase, the tree model becomes more and more complex. When the complexity of a model increase, the bias square for this model decreases constantly, but the variance for this model decreases first but then increases.We know that Enew = bias^2 + variance, so it is reasonable to get the plot above.

optimal amount of leaves:

```
## [1] 22
```

variables seem to be most important for decision making in this tree

```
##
## Classification tree:
## snip.tree(tree = tree_third, nodes = c(581L, 17L, 577L, 79L,
## 37L, 77L, 14L, 576L, 153L, 580L, 6L, 1157L, 15L, 16L, 5L, 1156L,
## 156L, 152L, 579L))
## Variables actually used in tree construction:
## [1] "poutcome" "month"    "contact"  "pdays"    "age"      "day"      "balance"
## [8] "housing"  "job"
## Number of terminal nodes:  22
## Residual mean deviance:  0.5698 = 10290 / 18060
## Misclassification error rate: 0.1039 = 1879 / 18084
```

"poutcome" "month" "contact" "pdays" "age" "day" "balance" "housing" "job" are variables used in tree construction and thus they are the most important.

task4

Estimate the confusion matrix, accuracy and F1 score for the test data by using the optimal model from step 3.

```
##      predict_test
##          no   yes
##   no  11872   107
##   yes  1371   214
```

```
## F1 score: 0.224554
## accuracy: 0.8910351
```

Comment whether the model has a good predictive power and which of the measures (accuracy or F1-score) should be preferred here.

From the results we can see accuracy is about 0.9. It seems this model has a good predictive power. But in fact it does not. This model can tell us precisely which customer are not willing to subscribe, but for those who are interested in subscribing it cannot make precise prediction. Unfortunately, this ability of model is what we really want. So here, F1 score is a better measure.

task5 Perform a decision tree classification of the test data with the following loss matrix,and report the confusion matrix for the test data. Compare the results with the results from step 4 and discuss how the rates has changed and why.

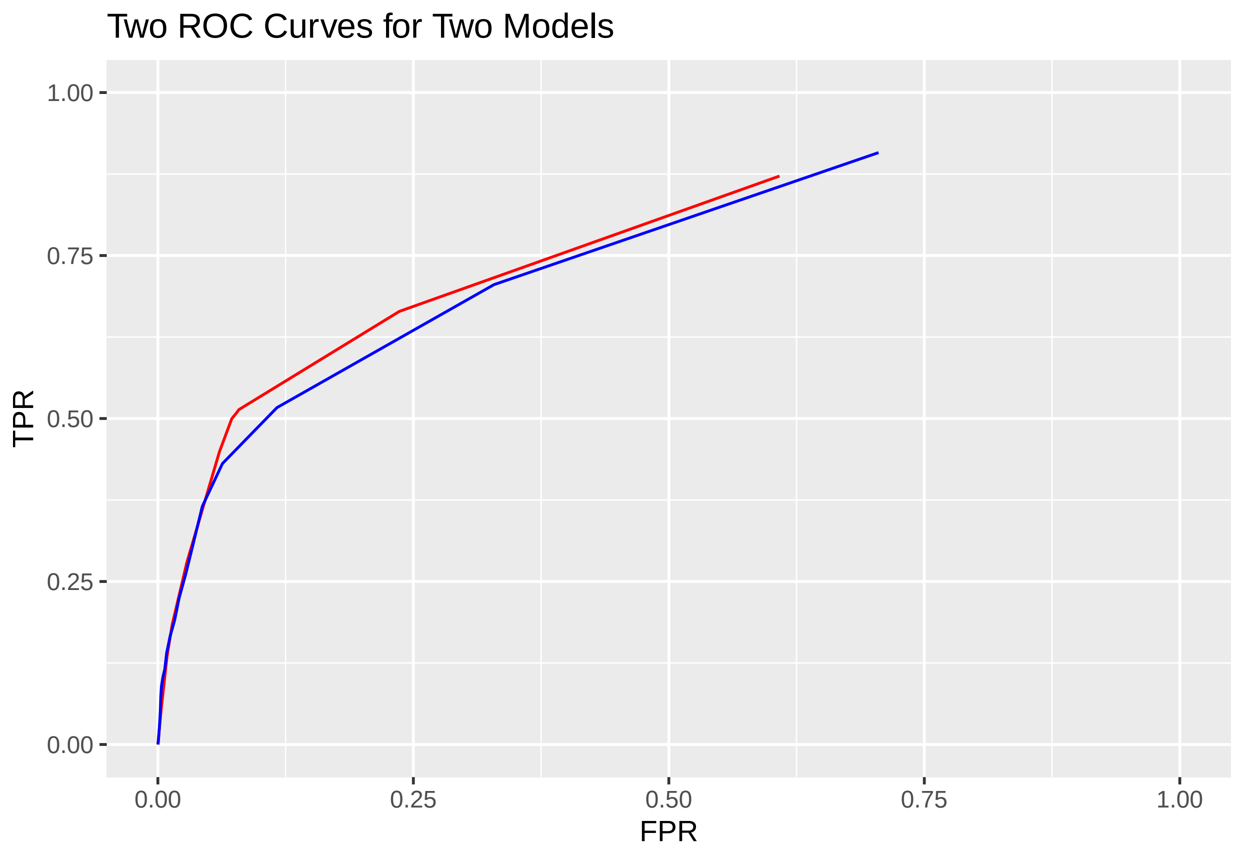
```
##      predict_test_2
##          no   yes
##   no  10880  1099
##   yes   807   778
```

```
## F1 score: 0.4494512
## accuracy: 0.859481
```

According to the data above we can see that F1 scores increases, which means the predicted results are not as unbiased as before. The main reason for this when using this loss matrix, we increase the cost false negative and decrease the cost of false positive. This change will make it harder for the model to make prediction about false negative and easier for the model to make prediction about false positive. And this can be shown in these two results.

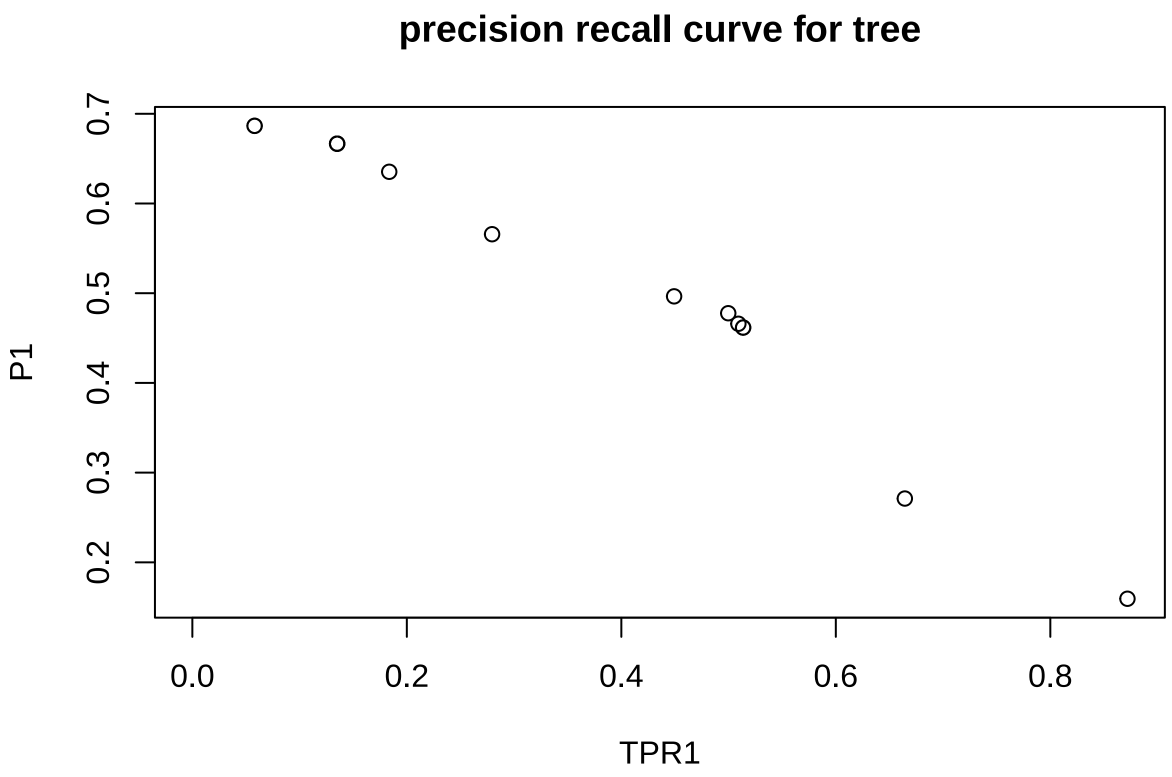
task6

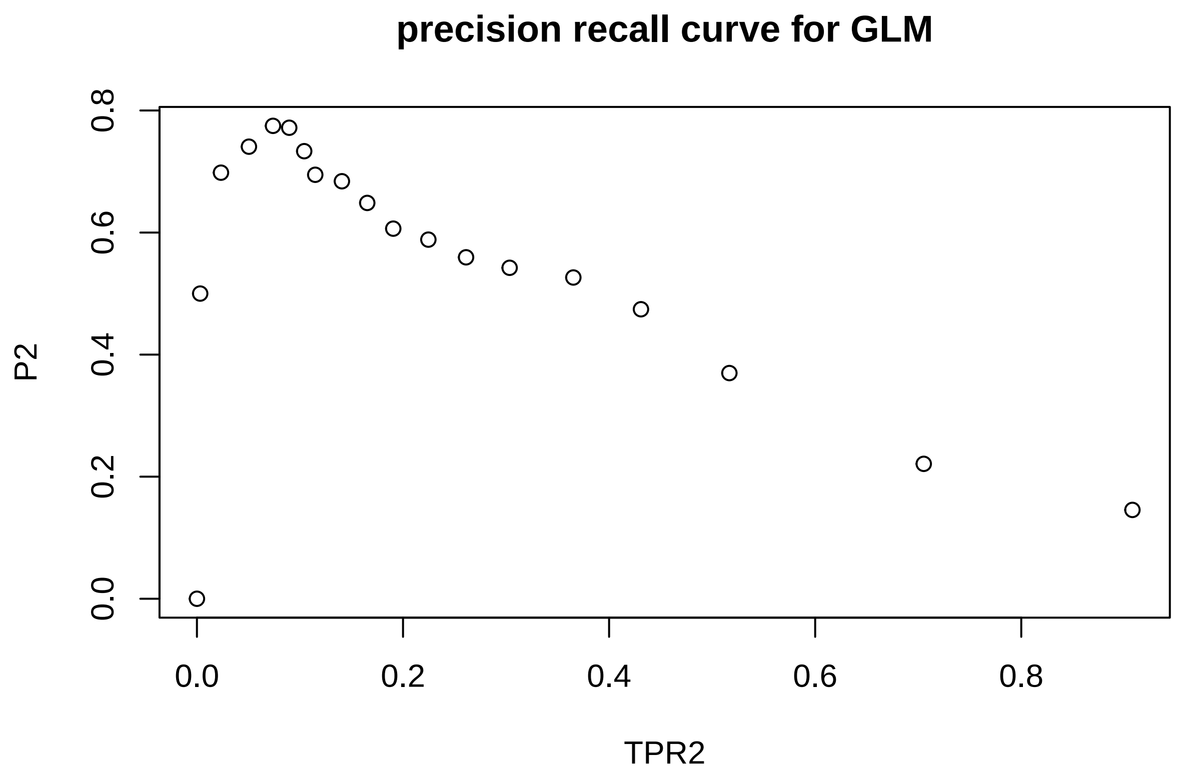
Compute the TPR and FPR values for the two models and plot the corresponding ROC curves. Conclusion?



By comparing two plots for two models, we find that the AUC of the tree model (red curve)is larger than that of glm(blue curve), and thus tree model has better performance.

Why precision recall curve could be a better option here?


precision recall curve for tree
P1
0.2
0.3
0.4
0.5
0.6
0.7
TPR1
0.0
0.2
0.4
0.6
0.8



From the ROC curves, we may think all of these models perform better, when we lower the threshold for predicting "positive". Although FPR increases because of this, the TPR increases more. But what we really care about is the precision of making a "positive" prediction. From the RP curves, we can see as the threshold lower and lower, the decreases in general. This is not what we want. So, we can draw a conclusion RP curves can reflect what we really care about and therefore a better choice compared to ROC.

Assignment 3

Task 1

```r
data3 <- read.csv("communities.csv")
set.seed(12345)

data3_full <- data3

data3_full[-ncol(data3_full)] <-
    data3_full %>%
      select( !ViolentCrimesPerPop) %>%
      apply( 2, scale)

data3_X <- data3_full[-ncol(data3_full)]
# no ViolentCrimesPerPop

S <- cov(data3_X)
eigenS <- eigen( S)
```

```
sumvar <- sum(eigenS$values)
for (i in 1:length(eigenS$values)) {
  if (sum(eigenS$values[1:i]) >= 0.95 * sumvar) {
    break;
  }
}
i
```

```
## [1] 35
```

```
# i:how many components are needed to obtain at least 95% of variance in the data

eigenS$values[c(1,2)] / sumvar
```

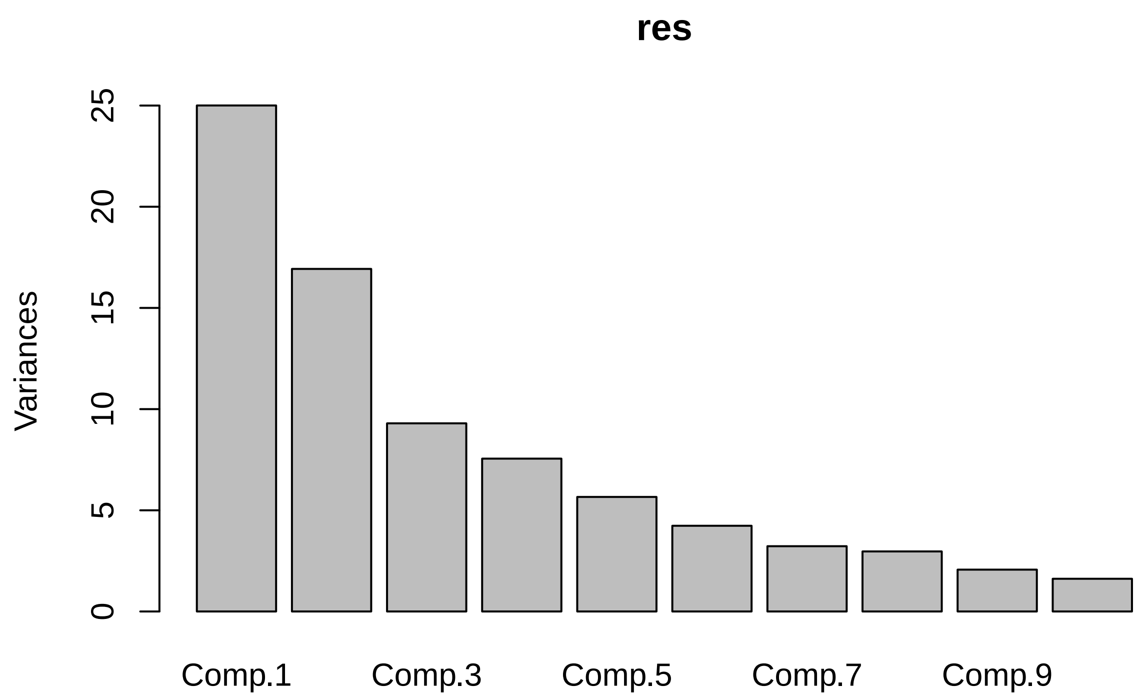
```
## [1] 0.2501699 0.1693597
```

```
# What is the proportion of variation explained by each of the first two principal components
```

- we can see that first 35 components obtain 95%+ of variance, and PC1 took 25% variation, PC2 took 17%

Task 2

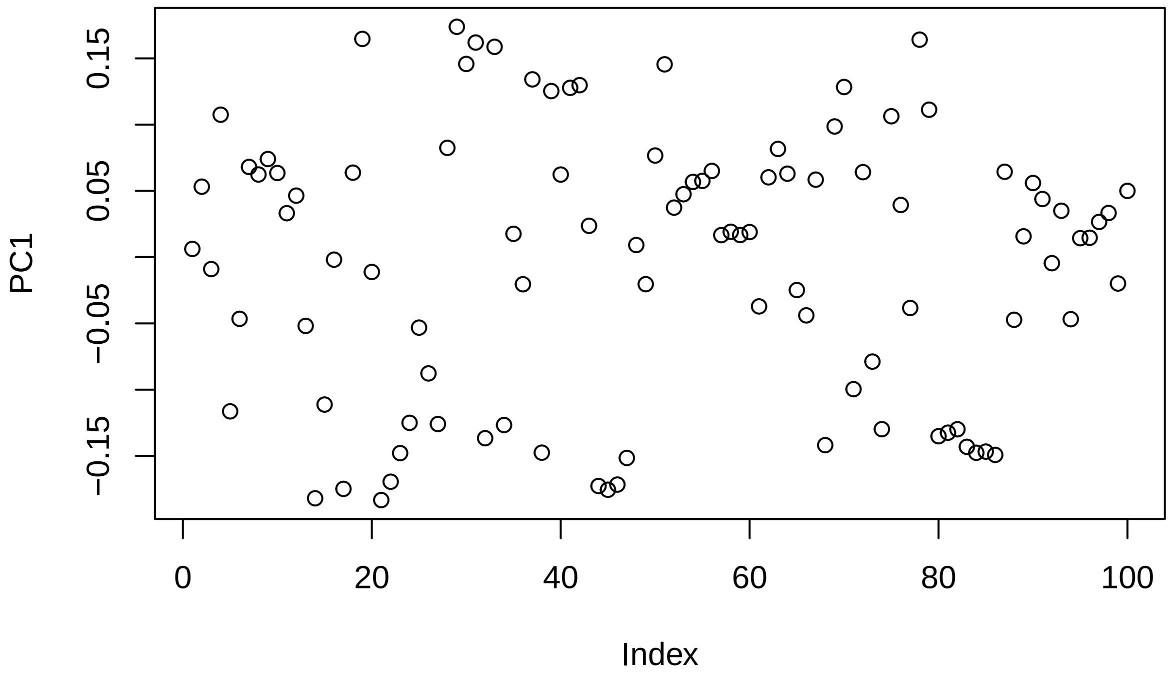
```
res <- princomp(data3_X)
lambda <- res$sdev^2
screeplot(res)
```



```
sort(abs(res$loadings[,1]), decreasing = TRUE)[1:5]
```

```
##      medFamInc      medIncome     PctKids2Par      pctWInvInc PctPopUnderPov
##      0.1833080      0.1819830       0.1755423       0.1748683      0.1737978
```

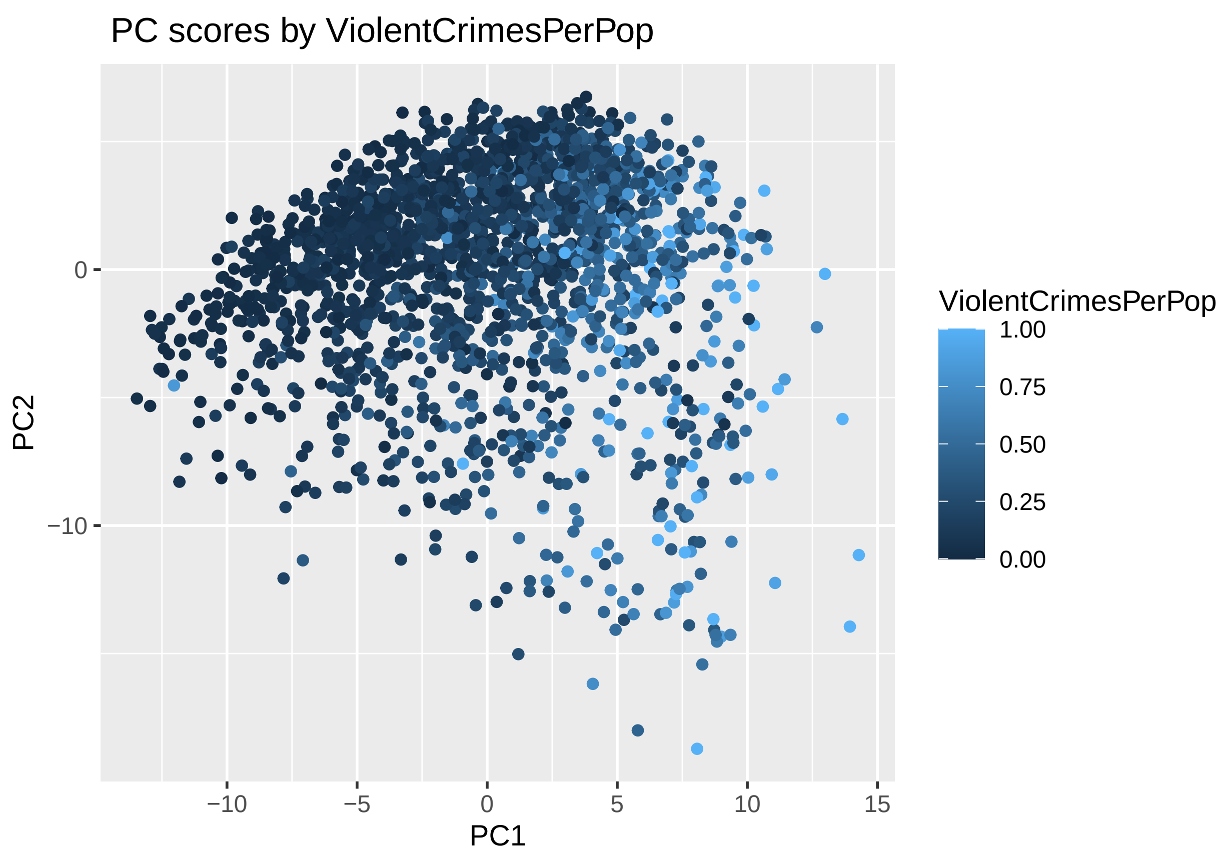
```
plot(eigenS[["vectors"]][,1],ylab = "PC1")
```



- Only 5 features have a variance which is greater than 5 and 2 features over 10 as we can seen in the plot, so we can consider that there isn't much features have a notable contribution. And from the trace plot of PC1, we can see that it's rather evenly distributed and few features contributes more then an absolute value of 0.15.
  5 features contribute mostly:
  medFamInc; medIncome; PctKids2Par; pctWInvInc; PctPopUnderPov
  We can see that all these 5 features highly related to economic level, which may positively of negatively contrubute to the crime level.

```
df_plotPCScores <- data.frame(PC1 = res$scores[,1], PC2 = res$scores[,2],
                              ViolentCrimesPerPop = data3$ViolentCrimesPerPop)

ggplot(df_plotPCScores, aes(x = PC1, y = PC2),)+
  geom_point(aes(colour = ViolentCrimesPerPop))+
  labs(title = " PC scores by ViolentCrimesPerPop")
```



- pop with a higher PC1 score tend to have a higher violent crimes rate. While PC2 doesn't have the same visibly enough contribute to the crimes rate in comparison with PC1

Task 3

```
n <- nrow(data3)
r_train <- sample(1:n , floor(0.5 * n) )
data3_train <- data3[r_train,]
data3_test <- data3[-r_train,]


data3_train_scale <- as.data.frame(scale(data3_train, scale = FALSE))
data3_test_scale <- as.data.frame(scale(data3_test, scale = FALSE))

lm_q3 <- glm(ViolentCrimesPerPop ~.-1 , data = data3_train_scale, family = gaussian)

err_train <- mean(lm_q3$residuals^2)

fit_test <- predict.glm(lm_q3, newdata = data3_test_scale)
err_test <- mean((fit_test - data3_test_scale$ViolentCrimesPerPop)^2)

err_train # train errors
```

```
## [1] 0.01406865
```

```
err_test # test errors
```

```
## [1] 0.02170806
```

- This model has a 0.01906891 test error,and hard to be considered having a good quality

Task 4

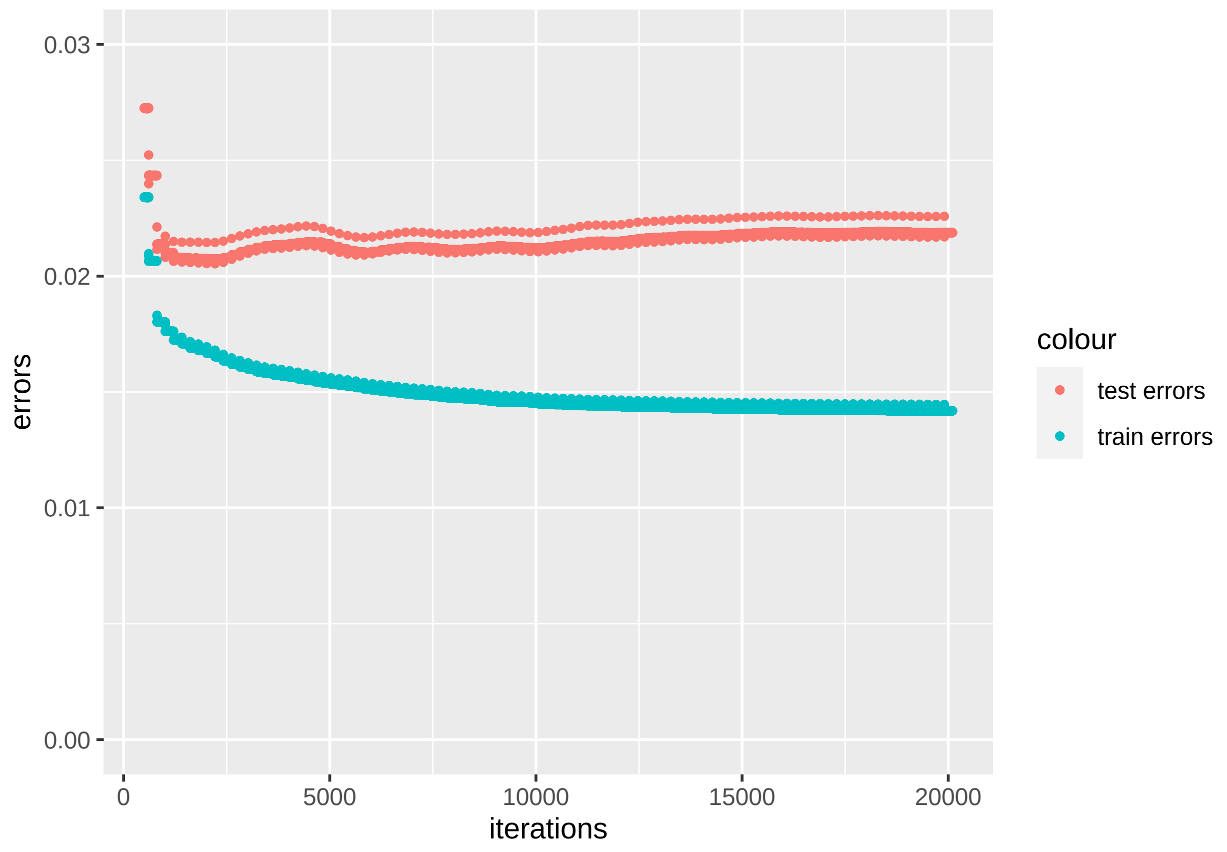
```
err_train_liklihood_optimed <- c()
err_test_liklihood_optimed <- c()

fun_cost <- function(theta, dataIn){
  y_hat <- theta %*% t(as.matrix(dataIn[,1:100]))
  y <- dataIn[,101]
  cost <- sum((y_hat - y)^2) / n


  pred_train_liklihood <- theta %*% t(data3_train_scale[,1:100])
  pred_test_liklihood <- theta %*% t(data3_test_scale[,1:100])
  i <<- i+1
  err_train_liklihood_optimed[i] <<- mean(
    (pred_train_liklihood - data3_train_scale$ViolentCrimesPerPop)^2)
  err_test_liklihood_optimed[i] <<- mean(
    (pred_test_liklihood - data3_test_scale$ViolentCrimesPerPop)^2)
  return(cost)
}

theta0 <- rep(0,100)
i <- 0
theta_hat_optim <- optim(theta0, fun_cost, dataIn = data3_train_scale, method = "BFGS")

ggplot(data = data.frame(err_train_liklihood_optimed, err_test_liklihood_optimed) )+
  geom_point(aes(x = 1:i, y = err_train_liklihood_optimed, color = 'train errors'),
             size = 1) +
  geom_point(aes(x = 1:i, y = err_test_liklihood_optimed, color = 'test errors'),
             size = 1) +
  xlim(500, i) +
  ylim(0, 0.03) +
  labs(
    main = "errors by iterations",
    x = "iterations",
    y = "errors",
  )
```



```
err_train_liklihood_optimed[7500]
```

```
## [1] 0.01483463
```

```
err_test_liklihood_optimed[7500]
```

```
## [1] 0.02119672
```

- from the plot we can choose iteration 7500 as an optimal choice, and the corresponding error is 0.01721628 for training set and 0.01871308 for testing set, it's slightly better than results in step 3 (0.01906891 test error) but not significant.

APPENDIX PART1

```
# ```{r library, include=FALSE}
library(tidyverse)
library(ggplot2)
library(glmnet)
<!-- ``` -->

# ```{r 1.1 preparation}
set.seed(12345)
data1 <- read.csv("tecator.csv")
n <- nrow(data1)
sample50 <- sample(n, round(0.5 * n))

data1_train <- data1[sample50,]
data1_test <- data1[-sample50,]
```

```
# Divide data
<!-- ``` -->
## Task 1

# ```{r 1.1}
lm_fat <- glm(Fat ~ . -Sample -Protein -Moisture, data = data1_train, family = gaussian())

pred_train <- predict(lm_fat, newdata = data1_train)
pred_test <- predict(lm_fat, newdata = data1_test)

err_train <- sum((pred_train - data1_train$Fat)^2) / (2*length(pred_train))
err_test <- sum((pred_test - data1_test$Fat)^2) / (2*length(pred_test))

err_train
err_test
<!-- ``` -->

- Overfitted! Works well on train set but can't fit the test set at all. We need
a better method to predict Fat.

## Task 2

# ```{r 1.2}
theta_hat_cost <- function(X, y, lambda, theta){
  # WITH intercept
  theta_hat <- sqrt(sum((X %*% theta - y)^2)) / length(theta) + lambda * sum(theta)
}
<!-- ``` -->

- cost function(with squared error loss):
$$\hat\theta = \frac1n||y-X\theta||^2_2 +   \lambda||\theta||_1$$

## Task 3

# ```{r 1.3}
glm_train <- glmnet(data1_train[,2:101], data1_train[,102], family = "gaussian",alpha = 1)
plot(glm_train, xvar = "lambda")
<!-- ``` -->
# ```{r 1.3 plotzoom}
zglm_train <- glmnet(data1_train[,2:101], data1_train[,102], family = "gaussian",
alpha = 1, lambda = seq(exp(-1),1, length.out = 100))
plot(zglm_train, xvar = "lambda", main = "zoomed plot")
<!-- ``` -->

- when $\lambda$ goes greater, the coefficients will all reduce to 0.While most
coefficients is or close to 0 all the way. From the plot, we can
choose $\lambda$ which   $ln(\lambda)$ close to 0.5 if we want to
select a model with only three features.

# ```{r 1.3 findlambda, eval=FALSE, include=FALSE}
for (i in 1:glmdata1$dim[2]) {
  row_coef <- which(glmdata1$beta[,i] != 0)
  if (length(row_coef) >= 3) {
```

```
    break
  }
}
glmdata1$lambda[i]
<!-- ``` -->

## Task 4

# ```{r 1.4}
glm_train_ridge <- glmnet(data1_train[,2:101], data1_train[,102],
family = "gaussian", alpha = 0)
plot(glm_train_ridge, xvar = "lambda")
<!-- ``` -->

- greater lambda value of ridge will also converge all coefficients to 0, but
ridge don not tend to filter some coefficients to 0 when lambda value is small,
instead it scales them in relatively similar rates.

## Task 5

# ```{r 1.5}

glm_train_lassoCV <- cv.glmnet(as.matrix(data1_train[,2:101]),
as.matrix(data1_train[,102]), family = "gaussian",alpha = 1)
plot(glm_train_lassoCV)

nlambda.1se <- which(glm_train_lassoCV$glmnet.fit$lambda == glm_train_lassoCV$lambda.1se)
nlambda.min <- which(glm_train_lassoCV$glmnet.fit$lambda == glm_train_lassoCV$lambda.min)
glm_train_lassoCV[["glmnet.fit"]][["df"]][c(nlambda.1se, nlambda.min)]
glm_train_lassoCV$lambda.1se
glm_train_lassoCV$lambda.min
<!-- ``` -->

- the value of lambda we found that gives minimum cvm is 0.06593502, but it is not
significantly better than $log(\lambda) = -4$, the MSE doesn't seems have much
difference when $log(\lambda) \le -2$. So we choose lambda.1se as optimal $\lambda$
  value which maximize the non-zero coefficients number.

# ```{r 1.5 scatterplot}
pred_test_1se <- predict(glm_train_lassoCV, newx = as.matrix(data1_test[,2:101]))

ggplot(data = data.frame(pred_test_1se, obs = data1_test[,102], origin = pred_test))+
  geom_point(aes(x = 1:nrow(data1_test), y = lambda.1se, color = "prediction of optimal lambda"))+
  geom_point(aes(x = 1:nrow(data1_test), y = obs, color = "observation"))+
  geom_point(aes(x = 1:nrow(data1_test), y = origin, color = "oringinal prediction"))+
  labs(x  = "ID of different samples", y = "Fat level")

<!-- ``` -->

- We can see that the differences between points of observation and optimized prediction
is close, and much more better than the origin predictions. So we consider the optimized
one a good and better prediction.
```

PART2

```r
# task1
mydata <- read.csv("bank-full.csv", sep = ";",stringsAsFactors = TRUE)
mydata <- mydata[,-12]

n <- dim(mydata)[1]
set.seed(12345)
id <- sample(1:n,floor(n * 0.4))
data_train <- mydata[id,]

id1 <- setdiff(1:n,id)
set.seed(12345)
id2 <- sample(id1, floor(0.3 * n))
data_validation <- mydata[id2,]

id3 <- setdiff(id1,id2)
data_test <- mydata[id3,]

# task2

# the first decision tree without any limitation
library(tree)
tree_first <- tree(y~.,data = data_train)
predict_train_1 <- predict(tree_first,type = "class")
mis_train_1 <- 1- sum(diag(table(data_train$y,predict_train_1))) /
  nrow(data_train)


predict_valid_1 <- predict(tree_first, newdata = data_validation,type = "class")
mis_valid_1 <- 1- sum(diag(table(data_validation$y,predict_valid_1))) /
  nrow(data_validation)
# plot(tree_first,mai = c(0.5, 0.1, 0.5, 0.1))


# the second decision tree with smallest allowed node size equal to 7000.
tree_second <- tree(y~.,data = data_train,minsize = 7000)
predict_train_2 <- predict(tree_second,type = "class")
mis_train_2 <- 1- sum(diag(table(data_train$y, predict_train_2))) /
  nrow(data_train)

predict_valid_2 <- predict(tree_second, newdata = data_validation,type = "class")
mis_valid_2 <- 1- sum(diag(table(data_validation$y, predict_valid_2))) /
  nrow(data_validation)
# plot(tree_second,mai = c(0.5, 0.1, 0.5, 0.1))



# the third decision tree(minimum deviance to 0.0005)
tree_third <- tree(y~.,data = data_train,mindev = 0.0005)
predict_train_3 <- predict(tree_third,type = "class")
mis_train_3 <- 1- sum(diag(table(data_train$y, predict_train_3))) /
  nrow(data_train)

predict_valid_3 <- predict(tree_third,newdata = data_validation, type = "class")
```

```
mis_valid_3 <- 1- sum(diag(table(data_validation$y, predict_valid_3))) /
  nrow(data_validation)
# plot(tree_third,mai = c(0.5, 0.1, 0.5, 0.1))

result_table <- data.frame(train_error = c(mis_train_1,mis_train_2,mis_train_3),
                           valid_error = c(mis_valid_1,mis_valid_2,mis_valid_3))
rownames(result_table) <- c("default", "nodesize_7000", "mindev_0.0005")
result_table



# task3
leave_numbers <- NULL
train_de <- NULL
valid_de <- NULL

i <- 1
while(i < 50){
  tree_pruned <- prune.tree(tree_third,best = i + 1)
  predict_train_new <- predict(tree_pruned, type = "tree",newdata = data_train)
  predict_valid_new <- predict(tree_pruned, type = "tree", newdata = data_validation)

  leave_numbers[i] <- i + 1
  train_de[i] <- deviance(predict_train_new)
  valid_de[i] <- deviance(predict_valid_new)

  i <- i + 1
}

data_deviance <- data.frame(leave_numbers, train_de, valid_de)
index <- order(valid_de)[1]
leave_number <- leave_numbers[index]


# plot
library(reshape2)
melt_map <- melt(data_deviance, id.vars = "leave_numbers")

library(ggplot2)
ggplot(data = melt_map, aes(x = leave_numbers, y = value, color = variable)) +
  geom_line() + geom_point() + xlab("leave numbers") + ylab("deviance")



optimal_tree <- prune.tree(tree_third,best = leave_number)
summary(optimal_tree)



# task4
predict_test <- predict(optimal_tree,newdata = data_test, type = "class")
confusion_matrix <- table(data_test$y,predict_test)
```

```r
true_positive <- confusion_matrix[2,2]
false_negative <- confusion_matrix[2,1]
false_positive <- confusion_matrix[1,2]
recall <- true_positive / (true_positive + false_negative)
precision <- true_positive / (true_positive + false_positive)
F1 <- 2 * precision * recall / (precision + recall)

accuracy <- 1 - mean(data_test$y != predict_test)
print(confusion_matrix)
cat("F1 score:",F1,"\naccuracy:",accuracy)


# task5
library(rpart)
loss_matrix <- matrix(c(0,1,5,0), nrow = 2, ncol = 2, byrow = TRUE)

# another function to create a tree with loss_matrix
new_tree <- rpart(y~.,data_train,method = "class", parms = list(loss = loss_matrix) )

predict_test_2 <- predict(new_tree, newdata = data_test, type = "class" )

confusion_matrix2 <- table(data_test$y,predict_test_2)
confusion_matrix2

true_positive2 <- confusion_matrix2[2,2]
false_negative2 <- confusion_matrix2[2,1]
false_positive2 <- confusion_matrix2[1,2]
recall2 <- true_positive2 / (true_positive2 + false_negative2)
precision2 <- true_positive2 / (true_positive2 + false_positive2)
F1_2 <- 2 * precision2 * recall2 / (precision2 + recall2)


accuracy_2 <- 1 - mean(data_test$y != predict_test_2)
print(confusion_matrix2)
cat("F1 score:",F1_2,"\naccuracy:",accuracy_2)


# task6


lr <- glm(y~.,data = data_train,family = "binomial")
predict_test_3 <- predict(lr, newdata = data_test, type = "response")
predict_test_4_temp <- predict(optimal_tree, newdata = data_test,
                          type = "vector")
predict_test_4 <- as.data.frame(predict_test_4_temp)$yes

threshold <- seq(0.05,0.95,0.05)

# TREE part
TPR1 <- NULL
FPR1 <- NULL
P1 <- NULL
```

```r
for (i in threshold) {

  predicted <- ifelse(predict_test_4 > rep(i,13564), "yes","no")
  TP1 <- length(which((predicted  == data_test$y) & predicted == "yes" ))
  TN1 <-length(which((predicted  == data_test$y) & predicted == "no" ))
  FP1 <- length(which((predicted != data_test$y) & predicted == "yes" ))
  FN1 <- length(which((predicted != data_test$y) & predicted == "no" ))

  index <- i * 20

  TPR1[index] <- TP1 / (TP1 + FN1)
  FPR1[index] <- FP1 / (TN1 + FP1)
  P1[index] <- TP1 / (TP1 + FP1)

}

# GLM part

TPR2 <- NULL
FPR2 <- NULL
P2 <- NULL

for (i in threshold) {

  predicted_GLM <- ifelse(predict_test_3 > rep(i,13564), "yes","no")
  TP2 <- length(which(predicted_GLM == data_test$y & predicted_GLM == "yes"))
  TN2 <- length(which(predicted_GLM == data_test$y & predicted_GLM == "no"))
  FP2<- length(which(predicted_GLM != data_test$y & predicted_GLM == "yes"))
  FN2 <- length(which(predicted_GLM != data_test$y & predicted_GLM == "no"))

  index <- i * 20

  TPR2[index] <- TP2/ (TP2 + FN2)
  FPR2[index] <- FP2 / (TN2 + FP2)
  P2[index] <- TP2 / (TP2 + FP2)
}



data_roc_tree <- data.frame(FPR1,TPR1)
data_roc_glm <- data.frame(FPR2,TPR2)
ggplot(data_roc_tree,aes(x = FPR1, y = TPR1)) +
  geom_line(color = "red")  +
  xlim(0,1) + ylim(0,1) + labs(title = "Two ROC Curves for Two Models",
                               x = "FPR", y = "TPR")+
  geom_line(data=data_roc_glm, aes(x = FPR2, y = TPR2), color = "blue")



plot(TPR1,P1, main = "precision recall curve for tree")
plot(TPR2,P2, main = "precision recall curve for GLM")
```

PART3

```
## Task 1

# ```{r 3.1}
data3 <- read.csv("communities.csv")
set.seed(12345)

data3_full <- data3

data3_full[-ncol(data3_full)] <-
    data3_full %>%
      select( !ViolentCrimesPerPop) %>%
      apply( 2, scale)

data3_X <- data3_full[-ncol(data3_full)]
# no ViolentCrimesPerPop

S <- cov(data3_X)
eigenS <- eigen( S)

sumvar <- sum(eigenS$values)
for (i in 1:length(eigenS$values)) {
  if (sum(eigenS$values[1:i]) >= 0.95 * sumvar) {
    break;
  }
}
i
# i:how many components are needed to obtain at least 95% of variance in the data

eigenS$values[c(1,2)] / sumvar
# What is the proportion of variation explained by each of the first two principal components
<!-- ``` -->
- we can see that first 35 components obtain 95%+ of variance,
and PC1 took 25% variation, PC2 took 17%

## Task 2

# ```{r 3.2}
res <- princomp(data3_X)
lambda <- res$sdev^2
screeplot(res)

sort(abs(res$loadings[,1]), decreasing = TRUE)[1:5]

plot(eigenS[["vectors"]][,1],ylab = "PC1")
<!-- ``` -->

- Only 5 features have a variance which is greater than 5 and 2 features over
10 as we can seen in the plot, so we can consider that there isn't much features
have a notable contribution. And from the trace plot of PC1, we can see that it's
rather evenly distributed and few features contributes more then an absolute
value of 0.15.
5 features contribute mostly:
medFamInc; medIncome; PctKids2Par; pctWInvInc; PctPopUnderPov
```

```
We can see that all these 5 features highly related to economic level, which
may positively of negatively contrubute to the crime level.

# ```{r 3.2 plot}

df_plotPCScores <- data.frame(PC1 = res$scores[,1], PC2 = res$scores[,2],
                              ViolentCrimesPerPop = data3$ViolentCrimesPerPop)

ggplot(df_plotPCScores, aes(x = PC1, y = PC2),)+
  geom_point(aes(colour = ViolentCrimesPerPop))+
  labs(title = " PC scores by ViolentCrimesPerPop")


<!-- ``` -->

- pop with a higher PC1 score tend to have a higher violent crimes rate.
While PC2 doesn't have the same visibly enough contribute to the crimes rate
in comparison with PC1


## Task 3

# ```{r 3.3}
n <- nrow(data3)
r_train <- sample(1:n , floor(0.5 * n) )
data3_train <- data3[r_train,]
data3_test <- data3[-r_train,]


data3_train_scale <- as.data.frame(scale(data3_train, scale = FALSE))
data3_test_scale <- as.data.frame(scale(data3_test, scale = FALSE))

lm_q3 <- glm(ViolentCrimesPerPop ~.-1 , data = data3_train_scale, family = gaussian)

err_train <- mean(lm_q3$residuals^2)

fit_test <- predict.glm(lm_q3, newdata = data3_test_scale)
err_test <- mean((fit_test - data3_test_scale$ViolentCrimesPerPop)^2)

err_train # train errors
err_test # test errors
<!-- ``` -->

- This model has a 0.01906891 test error,and hard to be considered having
a good quality

## Task 4

# ```{r 3.4, warning=FALSE}
err_train_liklihood_optimed <- c()
err_test_liklihood_optimed <- c()

fun_cost <- function(theta, dataIn){
```

```
  y_hat <- theta %*% t(as.matrix(dataIn[,1:100]))
  y <- dataIn[,101]
  cost <- sum((y_hat - y)^2) / n


  pred_train_liklihood <- theta %*% t(data3_train_scale[,1:100])
  pred_test_liklihood <- theta %*% t(data3_test_scale[,1:100])
  i <<- i+1
  err_train_liklihood_optimed[i] <<- mean(
(pred_train_liklihood - data3_train_scale$ViolentCrimesPerPop)^2)
  err_test_liklihood_optimed[i] <<- mean(
(pred_test_liklihood - data3_test_scale$ViolentCrimesPerPop)^2)
  return(cost)
}

theta0 <- rep(0,100)
i <- 0
theta_hat_optim <- optim(theta0, fun_cost, dataIn = data3_train_scale, method = "BFGS")

ggplot(data = data.frame(err_train_liklihood_optimed, err_test_liklihood_optimed) )+
  geom_point(aes(x = 1:i, y = err_train_liklihood_optimed, color = 'train errors'),
             size = 1) +
  geom_point(aes(x = 1:i, y = err_test_liklihood_optimed, color = 'test errors'),
             size = 1) +
  xlim(500, i) +
  ylim(0, 0.03) +
  labs(
    main = "errors by iterations",
    x = "iterations",
    y = "errors",
  )

err_train_liklihood_optimed[7500]
err_test_liklihood_optimed[7500]

<!-- ``` -->

- from the plot we can choose iteration 7500 as an optimal choice,
and the corresponding error is 0.01721628 for training set and 0.01871308
for testing set, it's slightly better than results in step 3
(0.01906891 test error) but not significant.
```