Assignment 2, STK-INF 4300

He Gu

heg@mail.uio.no

Problem 1

Task 1

At the very first, we could check the distribution of each modality for all categorical variables.

CODE:

```
data <- read.table("C:/Users/he322/Desktop/Oblig2STK/ozone_496obs_25vars.txt", head
er = TRUE)
header <- names(data)
variables <- header[-25]
cata_vari <- c("ADHEU","SEX","HOCHOZON","AMATOP","AVATOP","ADEKZ","ARAUCH",
               "FSNIGHT","FMILB","FTIER","FPOLL","FLTOTMED","FSPT","FSATEM",
               "FSAUGE","FSPFEI","FSHLAUF")
num_vari <- c("ALTER","AGEBGEW","FLGROSS","FNOH24","FO3H24","FTEH24","FLGEW")
#1
#get and split and standardize the data
##check the distribution for catagorical variables
print(c("==1","==0"))
for (v in cata_vari){
  a0 <- nrow(data[data[v] == "1",])
  a1 <- nrow(data[data[v] == "0",])
  print(c(a0,a1))
}
```

RESULT:

```
[1] "==1" "==0"
[1]  37 459
[1] 251 245
[1] 315 181
[1] 132 364
[1] 104 392
[1]  97 399
[1] 150 346
[1]  55 441
[1]  79 417
[1]  56 440
[1] 120 376
[1]  47 445
[1] 154 342
[1]  26 470
[1]  68 428
[1]  26 470
[1]  33 463
```

Obviously, we see that there are clearly some unbalance, but since we are having 496 observations in total, it is nearly impossible that our training set contains only one modality even if we just randomly split the data. And we could test it.

We then simply split the data randomly and check the distribution of each modality.

CODE:

```
##split the data randomly
n_random = sample(1:496,496)
train_set <- data[c(sort(n_random[1:248])),]
test_set <- data[c(sort(n_random[249:496])),]

##check the distribution for catagorical variables of both test and train sets
for (v in cata_vari){
  a0 <- nrow(train_set[train_set[v] == "1",])
  a1 <- nrow(train_set[train_set[v] == "0",])
  print(c(a0,a1))
}
```

RESULT:

```
[1] "==1" "==0"
[1]  21 227
[1] 126 122
[1] 157  91
```

```
[1]  57 191
[1]  46 202
[1]  47 201
[1]  73 175
[1]  29 219
[1]  41 207
[1]  29 219
[1]  64 184
[1]  29 218
[1]  83 165
[1]  10 238
[1]  38 210
[1]  12 236
[1]  15 233
```

We see that the training set does contain all modalities.

And we shall then standardize the data. Notice that there are categorical variables, but according to our tasks' description, we do not need to interpret our model, which means that these variables could also be standardized, although it might be pretty meaningless. But notice that we need to find the "most relevant" variable in further tasks, and it means that we are supposed to standardize also the categorical variables, if not, the relative "importance" of these categorical variables could be higher than they really are.

Thus, we are standardizing categorical variables by the same mechanism as standardizing the other variables, i.e. $Z - score\ normalization$

Since we are assuming that we have no idea about the test set, we need to standardize both training and test set by using the status of training set, i.e. both by

$$new\ data = \frac{data - mean(training\ set)}{std(training\ set)}$$

CODE:

```
##standardize the training data and test set BOTH by mean(train) and std(train)
for (v in variables){
  mean_train <- mean(train_set[,v])
  sd_train <- sd(train_set[,v])
  train_set[v] <- (train_set[v] - mean_train)/ sd_train
  test_set[v] <- (test_set[v] - mean_train)/ sd_train
}
```

Task 2

CODE:

```
#2
#fit the data with gaussian
model <- lm(train_set$FFVC ~ ., data = train_set)
summary(model)
```

RESULT:

```
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.2727419  0.0128286 177.162  < 2e-16 ***
ALTER        0.0506641  0.0157764   3.211  0.00152 **
ADHEU       -0.0243432  0.0159593  -1.525  0.12859
SEX         -0.1029461  0.0136156  -7.561 1.03e-12 ***
HOCHOZON    -0.0091874  0.0180470  -0.509  0.61120
AMATOP       0.0007830  0.0138945   0.056  0.95511
AVATOP       0.0008852  0.0135605   0.065  0.94801
ADEKZ       -0.0124366  0.0140509  -0.885  0.37705
ARAUCH      -0.0029869  0.0139206  -0.215  0.83030
AGEBGEW      0.0046095  0.0139171   0.331  0.74080
FSNIGHT     -0.0017677  0.0149890  -0.118  0.90623
FLGROSS      0.1515287  0.0222518   6.810 8.96e-11 ***
FMILB       -0.0108766  0.0191260  -0.569  0.57015
FNOH24      -0.0176679  0.0179608  -0.984  0.32633
FTIER       -0.0066234  0.0159679  -0.415  0.67869
FPOLL       -0.0484411  0.0279185  -1.735  0.08411 .
FLTOTMED    -0.0126885  0.0138921  -0.913  0.36204
FO3H24       0.0431497  0.0286550   1.506  0.13352
FSPT         0.0260766  0.0314917   0.828  0.40853
FTEH24      -0.0428080  0.0268900  -1.592  0.11281
FSATEM       0.0125539  0.0164625   0.763  0.44652
FSAUGE      -0.0028839  0.0153242  -0.188  0.85090
FLGEW        0.0820594  0.0205223   3.999 8.67e-05 ***
FSPFEI       0.0453508  0.0162609   2.789  0.00575 **
FSHLAUF      0.0012133  0.0143145   0.085  0.93253
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.202 on 223 degrees of freedom
Multiple R-squared:  0.6853,    Adjusted R-squared:  0.6514
F-statistic: 20.23 on 24 and 223 DF,  p-value: < 2.2e-16
```

COMMENTS:

Apparently, we see that **FLGROSS** has the strongest association with the forced vital capacity in such case, since the absolute value of its estimate is higher than others.

As for the **standard error**, we see that many variables have a relatively small standard error, which means that their estimates could be stable. But there also exists some variables which have a relatively large standard error, and it means that the estimates of these variables could be quite **unstable**, e.g. **ADHEU**, **HOCHOZON**, and etc.

Besides, according to the p-values of our variables, some of them are significantly non-zero, i.e. **ALTER, SEX, FLGROSS, FLGEW** and **FSPFEI**, while others are not.

Task 3

At first, we are using **AIC** as the stopping criterion.

CODE:

```
library(MASS)
null_model <- lm(train_set$FFVC ~ 1, data = train_set)

backward_AIC <- stepAIC(model, direction = "backward", k = 2, trace = FALSE)
forward_AIC <- stepAIC(null_model, direction = "forward", k = 2, trace = FALSE,
                       scope = list(lower = null_model, upper = model))
backward_AIC
forward_AIC
```

RESULT:

```
> (backward_AIC)

Call:
lm(formula = FFVC ~ ALTER + SEX + HOCHOZON + ARAUCH + AGEBGEW +
    FLGROSS + FLGEW + FSPFEI, data = train_set)

Coefficients:
(Intercept)  ALTER   SEX   HOCHOZON   ARAUCH   AGEBGEW   FLGROSS   FLGEW   FSPFEI
2.29084  0.02265  -0.09708  -0.02270  0.02954   0.02528   0.19083   0.06379  0.03864

> (forward_AIC)

Call:
lm(formula = train_set$FFVC ~ FLGROSS + SEX + FLGEW + FSPFEI +
    ARAUCH + HOCHOZON + AGEBGEW + ALTER, data = train_set)
```

```
16.
17. Coefficients:
18. (Intercept)  FLGROSS   SEX   FLGEW   FSPFEI   ARAUCH   HOCHOZON   AGEBGEW   ALTER
19. 2.29084   0.19083  -0.09708  0.06379  0.03864  0.02954  -0.02270  0.02528  0.02265
```

COMMENTS:

Obviously, we see that the models selected by backward and forward AIC selection are the same, but in fact, they are not always the same.

Sometimes, the model selected by backward AIC may contain more variables than forward AIC. This is pretty reasonable, since the selection made by AIC may converge at a local minimum, which means that backward and forward selections might converge earlier than they supposed to be, and thus, the backward AIC contains more variables than backward AIC selection.

But in such case, it seems that our selection performs pretty well, since both backward AIC and forward AIC converge at the same model.

We then change the stopping criterion into **BIC**, i.e. change $k = 2$ into $k = \log(n) = \log(24)$ in such case, and run the similar code again.

CODE:

```
1. backward_BIC <- stepAIC(model, direction = "backward", k = log(24),trace = FALSE)
2. forward_BIC <- stepAIC(null_model, direction = "forward", k = log(24), trace = FALS
   E, scope = list(lower = null_model, upper = model))
3. backward_BIC
4. forward_BIC
```

RESULT:

```
1. > (backward_BIC)
2.
3. Call:
4. lm(formula = FFVC ~ SEX + FLGROSS + FPOLL + FLTOTMED + FO3H24 +
5.     FTEH24 + FSATEM + FLGEW, data = train_set)
6.
7. Coefficients:
8. (Intercept)  SEX    FLGROSS    FPOLL   FLTOTMED   FO3H24   FTEH24  FSATEM  FLGEW
9.   2.28480  -0.09755  0.17628  -0.03219 -0.02928 0.06308 -0.07099  0.04508 0.10742
10.
11. > (forward_BIC)
12.
13. Call:
14. lm(formula = train_set$FFVC ~ FLGROSS + SEX + FLGEW + FSATEM +
15.     FLTOTMED + FPOLL, data = train_set)
16.
```

```
Coefficients:
(Intercept)    FLGROSS        SEX       FLGEW      FSATEM     FLTOTMED       FPOLL
   2.28480    0.17377    -0.09719    0.10694    0.04752    -0.03165    -0.03201
```

COMMENTS:

Following the same logic we used to analyze AIC selection, we see that, this time, the backward selection does contain more variables than the forward selection.

We know that the forward selection starts from a null model while the backward one starts from a full model, and in such case, they converge at different local minimums, and this is why the selected models are different and backward selection contains more variables.

OVERALL COMMENTS:

In such case, we see that the backward selection contains more variables than the forward selection, and the AIC selection contains more variables than the BIC selection. This is exactly the same as we expected.

According to the formulas of AIC and BIC, we see that BIC penalizes the amount of variables, especially when the amount is relatively small, while AIC does not penalize the amount.

In addition, we know that AIC tends to find a model which performs better in prediction, since AIC assumes that there is no correct model among the candidates, and it shall only find the "closest" model to the true model. But meanwhile, BIC tends to find a sparser model, since it shall find the model which is "most likely" to be true.

In fact, it is hard to say which stopping criterion and which kind of procedure is better, but we could just simply test their performance with using the test set, and it shall be fulfilled in Task 7.

REPORT FOR THESE 4 MODELS:

CODE:

```
summary(backward_AIC)
summary(forward_AIC)
summary(backward_BIC)
summary(forward_BIC)
```

RESULT:

```
> summary(backward_AIC)

Call:
lm(formula = FFVC ~ SEX + AMATOP + FLGROSS + FPOLL + FLTOTMED +
    FO3H24 + FTEH24 + FSATEM + FLGEW + FSPFEI, data = train_set)

Residuals:
```

```
      Min       1Q   Median       3Q      Max 
-0.47336 -0.13900 -0.00203  0.13173  0.55450 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  2.28480    0.01266 180.534  < 2e-16 ***
SEX         -0.09756    0.01321  -7.386 2.51e-12 ***
AMATOP      -0.02142    0.01324  -1.618  0.10703    
FLGROSS      0.18060    0.01895   9.532  < 2e-16 ***
FPOLL       -0.03229    0.01352  -2.388  0.01770 *  
FLTOTMED    -0.02719    0.01298  -2.095  0.03725 *  
FO3H24       0.05688    0.02537   2.242  0.02585 *  
FTEH24      -0.06978    0.02516  -2.774  0.00598 ** 
FSATEM       0.03480    0.01580   2.202  0.02860 *  
FLGEW        0.10461    0.01915   5.463 1.18e-07 ***
FSPFEI       0.02331    0.01541   1.513  0.13172    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2001 on 239 degrees of freedom
Multiple R-squared:  0.6957,    Adjusted R-squared:  0.683 
F-statistic: 54.65 on 10 and 239 DF,  p-value: < 2.2e-16

> summary(forward_AIC)

Call:
lm(formula = train_set$FFVC ~ FLGROSS + SEX + FLGEW + FSATEM + 
    FLTOTMED + FPOLL + AMATOP + FTEH24 + FO3H24 + FSPFEI, data = train_set)

Residuals:
      Min       1Q   Median       3Q      Max 
-0.47336 -0.13900 -0.00203  0.13173  0.55450 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  2.28480    0.01266 180.534  < 2e-16 ***
FLGROSS      0.18060    0.01895   9.532  < 2e-16 ***
SEX         -0.09756    0.01321  -7.386 2.51e-12 ***
FLGEW        0.10461    0.01915   5.463 1.18e-07 ***
FSATEM       0.03480    0.01580   2.202  0.02860 *  
FLTOTMED    -0.02719    0.01298  -2.095  0.03725 *  
FPOLL       -0.03229    0.01352  -2.388  0.01770 *  
AMATOP      -0.02142    0.01324  -1.618  0.10703    
FTEH24      -0.06978    0.02516  -2.774  0.00598 ** 
```

```
FO3H24       0.05688    0.02537   2.242  0.02585 *
FSPFEI       0.02331    0.01541   1.513  0.13172
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2001 on 239 degrees of freedom
Multiple R-squared:  0.6957,    Adjusted R-squared:  0.683
F-statistic: 54.65 on 10 and 239 DF,  p-value: < 2.2e-16

> summary(backward_BIC)

Call:
lm(formula = FFVC ~ SEX + FLGROSS + FPOLL + FLTOTMED + FO3H24 +
    FTEH24 + FSATEM + FLGEW, data = train_set)

Residuals:
     Min       1Q   Median       3Q      Max
-0.46964 -0.14614  0.00336  0.13200  0.56232

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.28480    0.01273 179.537  < 2e-16 ***
SEX         -0.09755    0.01320  -7.390 2.40e-12 ***
FLGROSS      0.17628    0.01892   9.318  < 2e-16 ***
FPOLL       -0.03219    0.01355  -2.375 0.018312 *
FLTOTMED    -0.02928    0.01297  -2.257 0.024919 *
FO3H24       0.06308    0.02529   2.494 0.013291 *
FTEH24      -0.07099    0.02520  -2.817 0.005243 **
FSATEM       0.04508    0.01338   3.369 0.000878 ***
FLGEW        0.10742    0.01900   5.653 4.44e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2012 on 241 degrees of freedom
Multiple R-squared:  0.6898,    Adjusted R-squared:  0.6795
F-statistic: 66.98 on 8 and 241 DF,  p-value: < 2.2e-16

> summary(forward_BIC)

Call:
lm(formula = train_set$FFVC ~ FLGROSS + SEX + FLGEW + FSATEM +
    FLTOTMED + FPOLL, data = train_set)

Residuals:
```

```
     Min       1Q   Median       3Q      Max
-0.49700 -0.13431 -0.00183  0.12576  0.53832

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.28480    0.01288 177.375  < 2e-16 ***
FLGROSS      0.17377    0.01912   9.091  < 2e-16 ***
SEX         -0.09719    0.01334  -7.287 4.42e-12 ***
FLGEW        0.10694    0.01922   5.565 6.91e-08 ***
FSATEM       0.04752    0.01346   3.531 0.000495 ***
FLTOTMED    -0.03165    0.01310  -2.416 0.016423 *
FPOLL       -0.03201    0.01366  -2.343 0.019957 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2037 on 243 degrees of freedom
Multiple R-squared:  0.6795,    Adjusted R-squared:  0.6716
F-statistic: 85.87 on 6 and 243 DF,  p-value: < 2.2e-16
```

PREDICTIONS ON THEIR PERFORMANCE:

In my opinion, the reduced model shall perform better than the full model.

At first, it is easy to find out that AIC/BIC of these models shall be lower than the full model, which indicates that the reduced models are better. And besides, according to the Std. Error of our estimates, nearly all estimates of the reduced model are more stable than the full model. In addition, notice that the p-values of our reduced models show that nearly all estimates are significantly not 0, while the p-values of the full model do not. This may result in an overfitting in the full model.

Thus, we could conclude that the reduced models are supposed to be better than the full model, i.e. has a lower prediction error.

Task 4

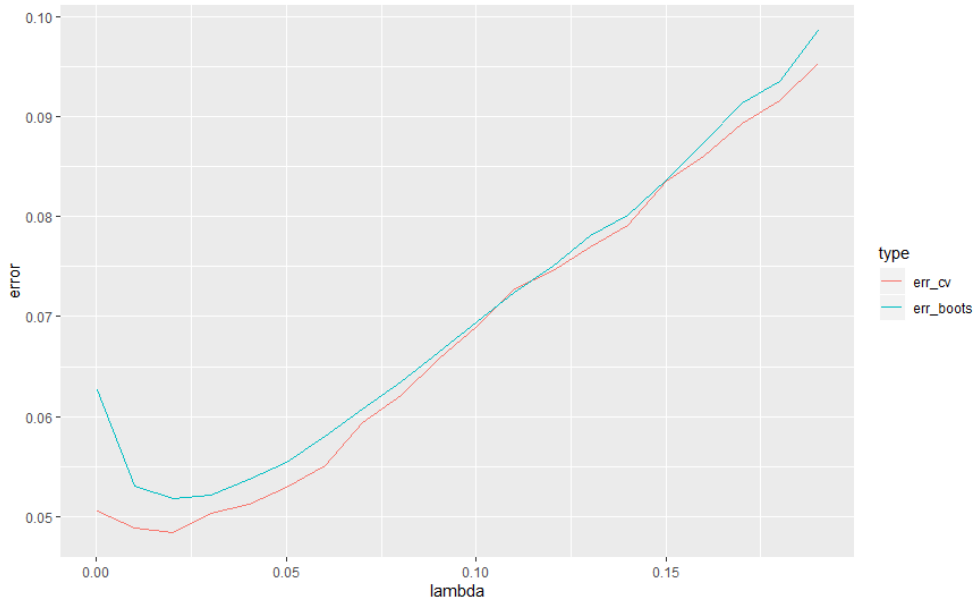
CODE:

```r
#4
#select lasso
##split data
library(glmnet)
n_random = sample(1:496,496)
train_set <- data[c(sort(n_random[1:250])),]
X <- train_set[-25]
y <- train_set[25]

```

```r
##setting range of parameters
lambdas <- seq(0.0001, 0.2, by = 0.01)
n_lambdas = length(lambdas)

##5 fold cv
err_cv = numeric(n_lambdas)
for (k in 1:n_lambdas){
  n_random_cv = sample(1:250,250)
  n_cv = c(sort(n_random_cv[1:50]), sort(n_random_cv[51:100]),
           sort(n_random_cv[101:150]), sort(n_random_cv[151:200]),
           sort(n_random_cv[201:250]))
  for (i in 1:5){
    a <- 50*i-49
    b <- 50*i
    X_train <- data.matrix(X[-n_cv[a:b], ])
    y_train <- data.matrix(y[-n_cv[a:b], ])
    X_test <- data.matrix(X[n_cv[a:b], ])
    y_test <- data.matrix(y[n_cv[a:b], ])
    lasso_reg <- glmnet(X_train, y_train, alpha = 1, lambda = lambdas[k])
    err_cv[k] <- err_cv[k] + sum((predict(lasso_reg, newx = X_test , s = lambdas[k]
) - y_test)^2)/50
  }
  err_cv[k] <- err_cv[k]/5
}

##bootstrap
err_boots = numeric(n_lambdas)
for (k in 1:n_lambdas){
  for (i in 1:10){
    n_random_boots = sample(1:250,250, replace = TRUE)
    n_boots = c(sort(n_random_boots))
    X_train <- data.matrix(X[-n_boots, ])
    y_train <- data.matrix(y[-n_boots, ])
    X_test <- data.matrix(X[n_boots, ])
    y_test <- data.matrix(y[n_boots, ])
    lasso_reg <- glmnet(X_train, y_train, alpha = 1, lambda = lambdas[k])
    err_boots[k] <- err_boots[k] + sum((predict(lasso_reg, newx = X_test , s = lamb
das[k]) - y_test)^2)/length(y_test)
  }
  err_boots[k] <- err_boots[k]/10
}

##plot
library(ggplot2)
```

```
df1 <- data.frame(lambda = lambdas, error = err_cv, type = "err_cv")
df2 <- data.frame(lambda = lambdas, error = err_boots, type = "err_boots")
df = rbind(df1, df2)
ggplot(df) + geom_line(aes(lambda,error,colour=type))
```

RESULT:



COMMENTS:

We see that, for Bootstrap procedure, the optimal lambda is around 0.02, while the optimal lambda for 5-fold cross-validation is also around 0.02.

By repeating the test several times (not shown here), although the result of Bootstrap procedure could vary, it shall still locate in $(0.01, 0.03)$, while the result of 5-fold CV also has a similar bound, i.e. $(0.01, 0.03)$.

Thus, we could conclude that the results of these 2 procedures agree with each other.

Task 5

Obviously, those categorical variables are not supposed to contain non-linear effects, since they are actually binary variables, i.e. 1 or 0.

Although we only need to consider the numerical variables, we are not knowing which variables contain non-linear effects. And thus, we just fit a GAM with splines for all numerical variables. Afterwards, we could check the ANOVA and the plots to justify which variables may contain non-linear effects.

We are using **mgcv** package to implement the GAM model, since such a package is able to automatically select the smoothness parameter, it shall use GCV-C_p by default, but other methods, such

as AIC and etc., are also available.

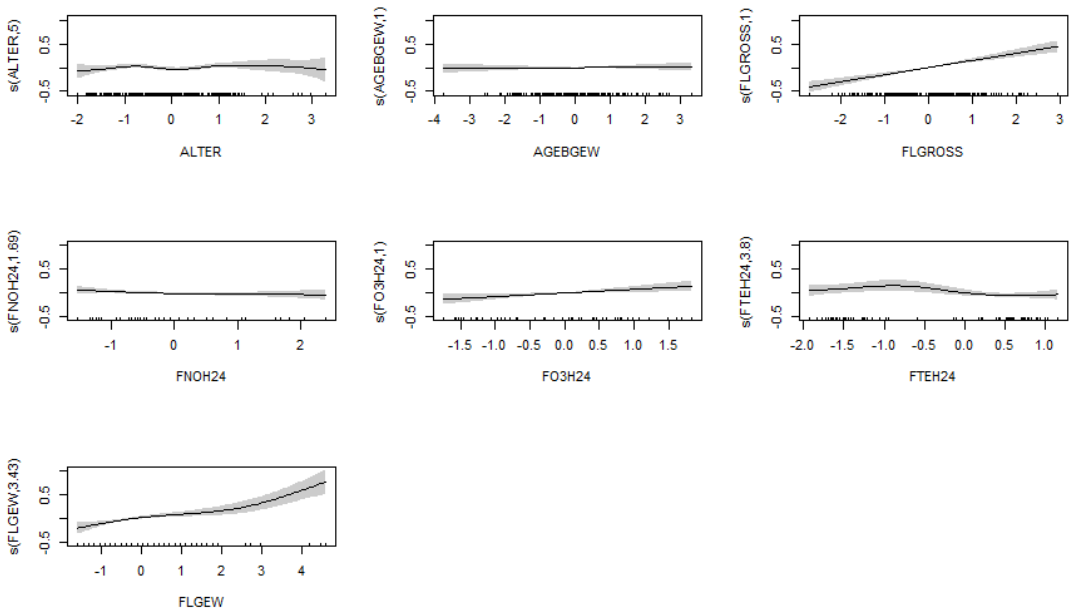
CODE:

```
#5
##GAM
library(mgcv)
model_gam_test <- gam(train_set$FFVC~s(ALTER)+s(AGEBGEW)+s(FLGROSS)+s(FNOH24)+s(FO3H24)+s(FTEH24)+s(FLGEW)+ADHEU+SEX+HOCHOZON+AMATOP+AVATOP+ADEKZ+ARAUCH+FSNIGHT+FMILB+FTIER+FPOLL+FLTOTMED+FSPT+FSATEM+FSAUGE+FSPFEI+FSHLAUF, data = train_set)
summary(model_gam_test)
plot(model_gam_test, pages = 1, scheme = 1, all.terms = TRUE)
```

RESULT:

```
Approximate significance of smooth terms:
              edf Ref.df      F  p-value
s(ALTER)   5.004  6.099  1.191   0.3323
s(AGEBGEW) 1.000  1.000  0.146   0.7027
s(FLGROSS) 1.000  1.000 52.219 6.34e-12 ***
s(FNOH24)  1.687  2.100  1.440   0.2493
s(FO3H24)  1.000  1.000  6.025   0.0149 *
s(FTEH24)  3.797  4.664  1.812   0.1192
s(FLGEW)   3.433  4.259 10.098 1.03e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-sq.(adj) =  0.715   Deviance explained = 75.3%
GCV = 0.04159  Scale est. = 0.03578   n = 250
```

PLOT:



By analyzing the "Approximate significance of smooth terms" and the plots above, it is clear that only the **FLGEW** may have a significant non-linear effect in such case. We then just implement a GAM which only applies splines to **FLGEW**.

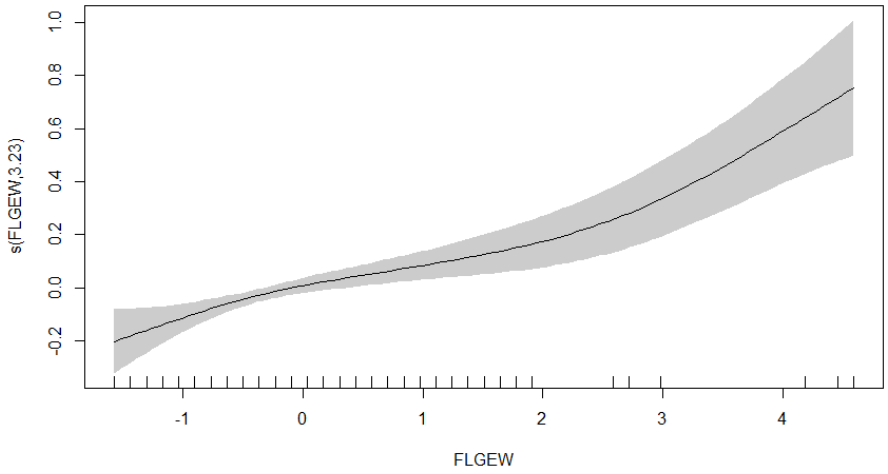
CODE:

```
1. model_gam <- gam(train_set$FFVC~ALTER+AGEBGEW+FLGROSS+FNOH24+FO3H24+FTEH24+s(FLGEW)
   +ADHEU+SEX+HOCHOZON+AMATOP+AVATOP+ADEKZ+ARAUCH+FSNIGHT+FMILB+FTIER+FPOLL+FLTOTMED+
   FSPT+FSATEM+FSAUGE+FSPFEI+FSHLAUF, data = train_set)
2. summary(model_gam)
3. plot(model_gam, pages = 1, scheme = 1, all.terms = FALSE)
```

RESULT:

```
1. Approximate significance of smooth terms:
2.            edf Ref.df     F  p-value
3. s(FLGEW) 3.226  4.028 10.54 6.85e-08 ***
4. ---
5. Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
6.
7. R-sq.(adj) =  0.699   Deviance explained = 73.1%
8. GCV = 0.042305  Scale est. = 0.037698  n = 250
```

PLOT:



COMMENTS:

The statistics and the plot above are the same as we expected. The p-value shows that our smooth terms are significantly not 0, and besides, the adjusted R-square is much higher than the original linear model which we implemented before. It means that this model fits better to the data, and therefore the non-linearity could be captured.

As for the plot above, it is clearly shown that the non-linearity of **FLGEW** does exist. And we then just try to implement a linear model with adding polynomial terms to this variable. We add the terms from degree 3 and keep increasing the degree until the ANOVA result is not proper any more.

CODE:

```
1. ##GAM polynomial
2. model_pol <- lm(train_set$FFVC ~ I(FLGEW^2) + I(FLGEW^3) + ., data = train_set)
3. summary(model_pol)
4. model_pol <- lm(train_set$FFVC ~ I(FLGEW^2) + I(FLGEW^3) + I(FLGEW^4) + ., data = t
   rain_set)
5. summary(model_pol)
```

RESULT:

```
1. Call:
2. lm(formula = train_set$FFVC ~ I(FLGEW^2) + I(FLGEW^3) + ., data = train_set)
3.
4. Residuals:
5.     Min      1Q  Median      3Q     Max
6. -0.5054 -0.1296  0.0074  0.1166  0.4875
7.
8. Coefficients:
9.               Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept)  2.3029252  0.0173051 133.078  < 2e-16 ***
I(FLGEW^2)  -0.0539251  0.0164036  -3.287  0.00117 **
I(FLGEW^3)   0.0118474  0.0040024   2.960  0.00341 **
ALTER        0.0192996  0.0146568   1.317  0.18927
ADHEU        0.0002489  0.0139112   0.018  0.98574
SEX         -0.1018458  0.0127167  -8.009 6.38e-14 ***
HOCHOZON    -0.0367559  0.0180419  -2.037  0.04281 *
AMATOP       0.0172865  0.0138462   1.248  0.21317
AVATOP      -0.0118461  0.0131629  -0.900  0.36911
ADEKZ       -0.0162862  0.0134587  -1.210  0.22752
ARAUCH      -0.0076738  0.0130251  -0.589  0.55635
AGEBGEW      0.0051167  0.0136240   0.376  0.70760
FSNIGHT     -0.0129540  0.0148930  -0.870  0.38534
FLGROSS      0.1473639  0.0214269   6.878 6.06e-11 ***
FMILB       -0.0083075  0.0173360  -0.479  0.63226
FNOH24      -0.0150290  0.0172761  -0.870  0.38527
FTIER        0.0119805  0.0144093   0.831  0.40661
FPOLL       -0.0232736  0.0248676  -0.936  0.35034
FLTOTMED    -0.0122225  0.0133976  -0.912  0.36260
FO3H24       0.0205912  0.0287983   0.715  0.47535
FSPT        -0.0046885  0.0292319  -0.160  0.87272
FTEH24      -0.0045167  0.0264904  -0.171  0.86477
FSATEM       0.0425879  0.0169139   2.518  0.01251 *
FSAUGE       0.0046250  0.0139578   0.331  0.74069
FLGEW        0.1054947  0.0239645   4.402 1.66e-05 ***
FSPFEI       0.0205858  0.0176614   1.166  0.24503
FSHLAUF     -0.0261818  0.0148196  -1.767  0.07865 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.1919 on 223 degrees of freedom
Multiple R-squared:  0.7213,    Adjusted R-squared:  0.6888
F-statistic:  22.2 on 26 and 223 DF,  p-value: < 2.2e-16

Call:
lm(formula = train_set$FFVC ~ I(FLGEW^2) + I(FLGEW^3) + I(FLGEW^4) +
    ., data = train_set)

Residuals:
      Min       1Q   Median       3Q      Max
-0.51798 -0.12306  0.00294  0.11571  0.47478

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept)  2.3011418  0.0172555 133.357  < 2e-16 ***
I(FLGEW^2)  -0.0461002  0.0169284  -2.723  0.00698 **
I(FLGEW^3)  -0.0119088  0.0141377  -0.842  0.40050
I(FLGEW^4)   0.0043722  0.0024965   1.751  0.08127 .
ALTER        0.0182333  0.0146020   1.249  0.21310
ADHEU       -0.0005823  0.0138553  -0.042  0.96652
SEX         -0.1032830  0.0126847  -8.142 2.78e-14 ***
HOCHOZON    -0.0385558  0.0179883  -2.143  0.03317 *
AMATOP       0.0182188  0.0137927   1.321  0.18789
AVATOP      -0.0125921  0.0131092  -0.961  0.33782
ADEKZ       -0.0157439  0.0134003  -1.175  0.24129
ARAUCH      -0.0055920  0.0130195  -0.430  0.66797
AGEBGEW      0.0043560  0.0135683   0.321  0.74848
FSNIGHT     -0.0145044  0.0148509  -0.977  0.32980
FLGROSS      0.1473226  0.0213283   6.907 5.14e-11 ***
FMILB       -0.0032768  0.0174936  -0.187  0.85159
FNOH24      -0.0163209  0.0172124  -0.948  0.34406
FTIER        0.0089142  0.0144494   0.617  0.53792
FPOLL       -0.0233969  0.0247532  -0.945  0.34558
FLTOTMED    -0.0108870  0.0133578  -0.815  0.41593
FO3H24       0.0219318  0.0286759   0.765  0.44519
FSPT        -0.0042973  0.0290982  -0.148  0.88273
FTEH24      -0.0041095  0.0263695  -0.156  0.87630
FSATEM       0.0408472  0.0168654   2.422  0.01624 *
FSAUGE       0.0047144  0.0138936   0.339  0.73469
FLGEW        0.1457747  0.0331363   4.399 1.69e-05 ***
FSPFEI       0.0220139  0.0175990   1.251  0.21230
FSHLAUF     -0.0286810  0.0148203  -1.935  0.05423 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.191 on 222 degrees of freedom
Multiple R-squared:  0.7251,    Adjusted R-squared:  0.6917
F-statistic: 21.69 on 27 and 222 DF,  p-value: < 2.2e-16
```

COMMENTS:

According to the results above, we see that, for a polynomial of degree 3, both **FLGEW^2** and **FLGEW^3** are significantly not 0. And besides, recall the R^2 for the original linear model we implemented before, we find that our new model has a much higher R^2, i.e. a better fitness.

Now, we could conclude that our new models really capture some of the non-linearity, and both GAM with splines and the GLM with polynomial terms are supposed to perform well. But as for the further comparisons, we could not guarantee anything more until we really test the performance of our models, i.e. Task 7.

Task 6

FYI, since all our data are standardized before fitting, we are then using "center = TRUE" for glmboost, i.e. **linear model** for component-wise boosting. And according to what we have stated before, only the numerical variables shall be applied a component-wise boosting with **splines** as base learners. And since the response of our data, i.e. FFVC, is numerical, so we are using default Gaussian family for all following functions:

CODE:

```
1. library(mboost)
2. compo_boosting_lin <- glmboost(FFVC ~ ., data = train_set, center = TRUE)
3. compo_boosting_spl <- mboost(FFVC ~ ALTER+AGEBGEW+FLGROSS+FNOH24+FO3H24+FTEH24+FLGE
   W, data = train_set, baselearner = 'bbs')
4. compo_boosting_tree <- mboost(FFVC ~ ., data = train_set, baselearner = "btree")
5. summary(compo_boosting_lin)
6. summary(compo_boosting_spl)
7. summary(compo_boosting_tree)
```

Thus, we are able to produce the **variable selection frequencies:**

Linear Models:

```
1. Selection frequencies:
2. SEX  FLGROSS  FLGEW FLTOTMED  FSATEM  FSPFEI  FPOLL  ADHEU  ALTER  FNOH24 HOCHOZON   FMILB
3. 0.21    0.17    0.12    0.10    0.08    0.08    0.07    0.05    0.04    0.04    0.03    0.01
```

Splines:

```
1. Selection frequencies:
2.  bbs(FTEH24) bbs(FLGROSS)   bbs(FLGEW)   bbs(ALTER)  bbs(FNOH24) bbs(AGEBGEW)  bbs(FO3H24)
3.      0.26         0.21         0.15         0.14         0.12         0.09         0.03
```

Trees:

```
1. Selection frequencies:
2. btree(FLGROSS)btree(FLGEW) btree(SEX) btree(ALTER) btree(AGEBGEW)btree(HOCHOZON) btree(FO3H24)
3.      0.30         0.20         0.15         0.08         0.06          0.05           0.05
4. btree(FNOH24)   btree(ADHEU)  btree(FTEH24)  btree(FSATEM)   btree(FPOLL)  btree(FLTOTMED)
5.      0.03           0.02           0.02           0.02           0.01           0.01
```

And the **regression coefficients** of linear models:

```
1. Coefficients:
2. (Intercept)       ALTER        ADHEU        SEX        HOCHOZON
3. -4.2850504427  0.0100680434 -0.0333030854 -0.1967860125 0.0080105588
```

```
FLGROSS      FMILB       FNOH24
0.0291114253 -0.0032612557 -0.0002388067
FPOLL       FLTOTMED        FSATEM          FLGEW        FSPFEI
-0.0268506402 -0.0418009858  0.0822904467  0.0097438394  0.0613377048
```

Task 7

CODE:

```
#7
##train error
train_error <- NULL
train_error['full'] <- mean(residuals(model)^2)

train_error['backAIC'] <- mean(residuals(backward_AIC)^2)
train_error['forAIC'] <- mean(residuals(forward_AIC)^2)
train_error['backBIC'] <- mean(residuals(backward_BIC)^2)
train_error['forBIC'] <- mean(residuals(forward_BIC)^2)

train_error['lasso_cv'] <- mean((data.matrix(y_task4) - predict(lasso_cv, newx = data.matrix(X_task4), s = lambda_cv))^2)
train_error['lasso_boots'] <- mean((data.matrix(y_task4) - predict(lasso_boots, newx = data.matrix(X_task4), s = lambda_boots))^2)

train_error['gam'] <- mean(residuals(model_gam)^2)
train_error['pol'] <- mean(residuals(model_pol)^2)

train_error['com_boost_lin'] <- mean(residuals(compo_boosting_lin)^2)
train_error['com_boost_spl'] <- mean(residuals(compo_boosting_spl)^2)
train_error['com_boost_tree'] <- mean(residuals(compo_boosting_tree)^2)

##test error
X_task7 <- test_set[-25]
y_task7 <- data.matrix(test_set[25])
test_error <- NULL

test_error['full'] <- mean((y_task7 - predict(model, newdata = X_task7))^2)
test_error['backAIC']<-mean((y_task7 - predict(backward_AIC, newdata = X_task7))^2)
test_error['forAIC']<-mean((y_task7 - predict(forward_AIC, newdata = X_task7))^2)
test_error['backBIC']<-mean((y_task7 - predict(backward_BIC, newdata = X_task7))^2)
test_error['forBIC'] <- mean((y_task7 - predict(forward_BIC, newdata = X_task7))^2)

```

```
test_error['lasso_cv'] <- mean((y_task7 - predict(lasso_cv, newx = data.matrix(X_task7), s = lambda_cv))^2)
test_error['lasso_boots'] <- mean((y_task7 - predict(lasso_boots, newx = data.matrix(X_task7), s = lambda_boots))^2)

test_error['gam'] <- mean((y_task7 - predict(model_gam, newdata = X_task7))^2)
test_error['pol'] <- mean((y_task7 - predict(model_pol, newdata = X_task7))^2)

test_error['com_boost_lin'] <- mean((y_task7 - predict(compo_boosting_lin, newdata = X_task7))^2)
test_error['com_boost_spl'] <- mean((y_task7 - predict(compo_boosting_spl, newdata = X_task7[num_vari]))^2)
test_error['com_boost_tree'] <- mean((y_task7 - predict(compo_boosting_tree, newdata = X_task7))^2)

print("train error")
print(train_error)
print("test error")
print(test_error)
```

RESULT:

```
[1] "train error"
     full        backAIC         forAIC        backBIC         forBIC
0.03950484     0.04123226     0.04123226     0.04123226     0.04123226
    lasso_cv    lasso_boots           gam         pol       com_boost_lin
0.10981893     0.11133505     0.03864648    0.03879739     0.04236932
com_boost_spl  com_boost_tree
0.04686207     0.03688197

[1] "test error"
     full          backAIC         forAIC        backBIC         forBIC
0.04831460     0.04789635     0.04789635     0.04789635     0.04789635
    lasso_cv    lasso_boots           gam         pol         com_boost_lin
0.14082055     0.14424095     0.04735161    0.04607857     0.04687056
com_boost_spl  com_boost_tree
0.05514283     0.05273946
```

COMMENTS:

We see overall that the training error are smaller than test error, which is exactly the same as we expected, since we are accessing the training set when we are fitting our model, while the test set keeps unseen.

We notice that both the training error and the prediction error for lasso regression are much larger

than others. This could be a result of standardizing. We have standardized both numerical variables and categorical variables into mean$= 0$ and sd.$= 1$, and it means that the corresponding **estimates** of these variables could **be shrunk/zoomed**, and this shall influence the LASSO.

Recall our prediction in **Task 3**, we see that the reduced models have larger training errors, but lower test errors. It is the same as we predicted. Although the full model fits the training data better, at the same time, it also brings more risk of overfitting. But as for the reduced models, they did not fit the training data as well as the full model, but they could take uncertainty of new data into account and avoid the overfitting issues, and therefore, they have better performance when being tested.

Problem 2

Before all tasks: split the data

By using the same method in **Problem 1**, we at first check the distribution for diabetes (positive/negative)

CODE:

```
1.  library(mlbench)
2.  library(kknn)
3.  data(PimaIndiansDiabetes)
4.  dat <- PimaIndiansDiabetes
5.
6.  a0 <- nrow(dat[dat['diabetes'] == "pos",])
7.  a1 <- nrow(dat[dat['diabetes'] == "neg",])
8.  print(c("pos","neg"))
9.  print(c(a0,a1))
```

RESULT:

```
1.  [1] "pos" "neg"
2.  [1] 268 500
```

It seems that we could just divide the data randomly, since there is no extreme distribution here.

Thus, we just split the train set and test set randomly and check the distribution of **diabetes** again to ensure that the training set and test set have similar distributions.

CODE:

```
1.  a0 <- nrow(train_set[train_set['diabetes'] == "pos",])
2.  a1 <- nrow(train_set[train_set['diabetes'] == "neg",])
```

```
print(c("pos","neg"))
print(c(a0,a1))

a0 <- nrow(test_set[test_set['diabetes'] == "pos",])
a1 <- nrow(test_set[test_set['diabetes'] == "neg",])
print(c("pos","neg"))
print(c(a0,a1))
```

RESULT:

```
[1] "pos" "neg"
[1] 183 327

[1] "pos" "neg"
[1]  85 173
```

Obviously, our train set and test set follow similar distributions of **diabetes.**

Task 1

CODE:

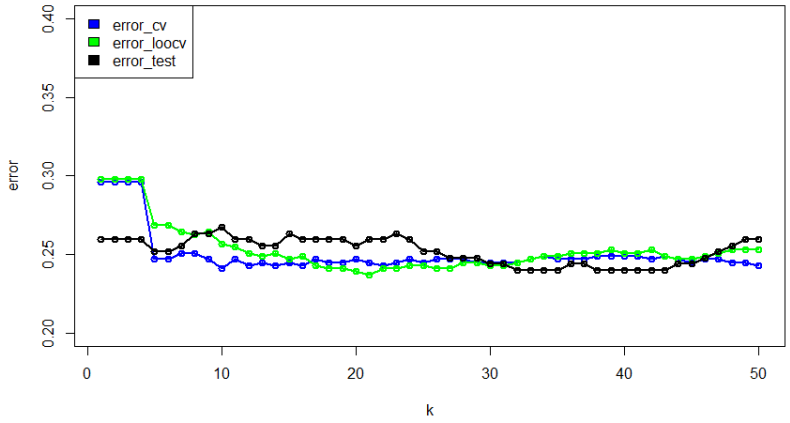
```
#task 1
##5-fold cv
err_cv <- numeric(k_range)
n_random_cv = sample(1:510,510)
n = c(sort(n_random_cv[1:102]), sort(n_random_cv[103:204]), sort(n_random_cv[205:306]),
      sort(n_random_cv[307:408]), sort(n_random_cv[409:510]))
k_range = 50

for (i in 1:k_range){
  for(j in 1:5){
    a <- 102*j-101
    b <- 102*j
    fit <- kknn(diabetes ~ ., train = train_set[-n[a:b], ], test = train_set[n[a:b], ], k = i)
    err_cv[i] <- err_cv[i] + sum(fitted.values(fit)!=train_set[n[a:b],9])/102
  }
  err_cv[i] <- err_cv[i]/5
}

##LOOCV
```

```
err_LOOCV <- numeric(k_range)
for (i in 1:k_range){
  for(j in 1:510){
    fit <- kknn(diabetes ~ ., train = train_set[-
j, ], test = train_set[j, ], k = i)
    err_LOOCV[i] <- err_LOOCV[i] + sum(fitted.values(fit)!=train_set[j, 9])
  }
  err_LOOCV[i] <- err_LOOCV[i]/510
}

##test
err_test <- numeric(k_range)
for (i in 1:k_range){
  fit <- kknn(diabetes ~ ., train = train_set, test = test_set, k = i)
  err_test[i] <- sum(fitted.values(fit)!=test_set[,9])/258
}

##print&plot
which.min(err_cv)
which.min(err_LOOCV)
which.min(err_test)

plot(1:k_range, err_cv, ylim=c(0.2, 0.4), xlab="k", ylab="error",col="blue", type="
o",lwd=2)
lines(1:k_range, err_LOOCV, col="green",type="o",lwd=2)
lines(1:k_range, err_test, col="black",type="o",lwd=2)
legend("topleft", c("error_cv","error_LOOCV","error_test"),fill=c("blue","green","b
lack"))
```

RESULT:

```
> which.min(err_cv)
[1] 24
> which.min(err_LOOCV)
[1] 24
> which.min(err_test)
[1] 19
```

PLOT:



COMMENTS:

Obviously, we see that all the 3 types of errors have similar tendencies, and it shows that both 5-fold CV and LOOCV could represent the distribution of the whole data set to some degree.

Task 2

By using the package **mgcv** with setting "select = TRUE", we could penalize some of the variables into 0. Such a procedure is pretty much similar with a LASSO regression. And it could automatically process the variable selection.

CODE:

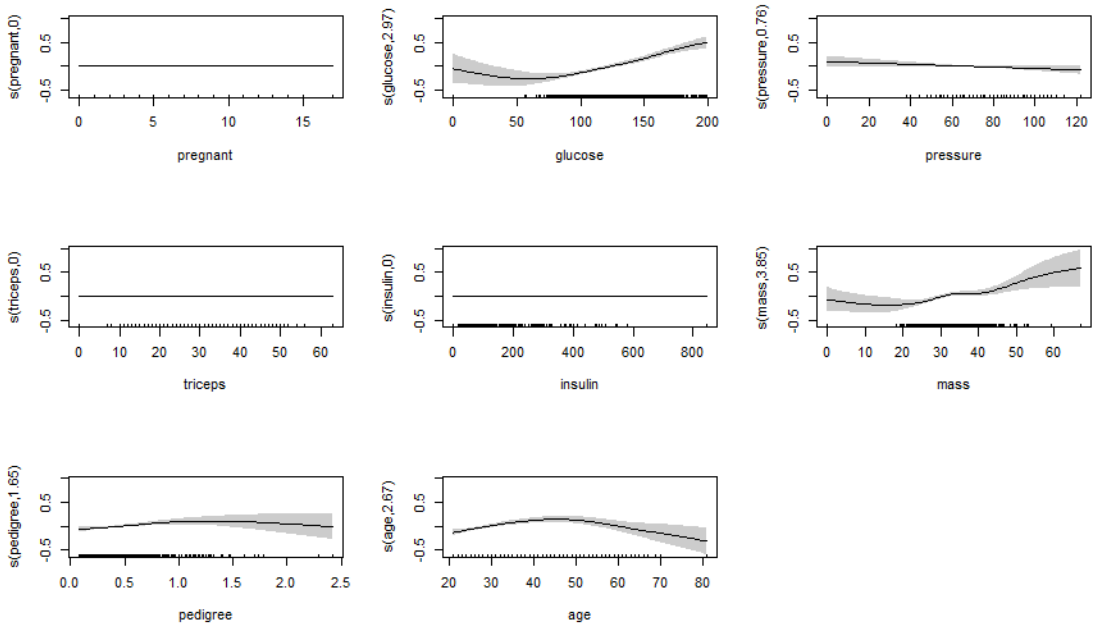
```
model_gam <- mgcv::gam(as.numeric(diabetes)~s(pregnant)+s(glucose)+s(pressure)+s(triceps)+s(insulin)+s(mass)+s(pedigree)+s(age), data = train_set, select = TRUE)
summary(model_gam)
plot(model_gam, pages = 1, scheme = 1, all.terms = FALSE)
```

RESULT:

```
Approximate significance of smooth terms:
                edf Ref.df      F  p-value
s(pregnant) 4.863e+00      9  1.534 0.009605 **
s(glucose)  2.463e+00      9 14.331  < 2e-16 ***
s(pressure) 9.717e-01      9  1.270 0.000587 ***
s(triceps)  4.408e-08      9  0.000 0.851817
s(insulin)  8.560e-08      9  0.000 0.604047
s(mass)     1.241e+00      9  2.272 2.07e-06 ***
s(pedigree) 9.033e-01      9  0.337 0.039042 *
s(age)      6.087e+00      9  3.056 5.93e-05 ***
---
```

```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-sq.(adj) =  0.395   Deviance explained = 41.5%
GCV = 0.14292  Scale est. = 0.13801    n = 510
```

PLOT:



According to the result and plot above, we see that some variables are reduced into 0, which is equivalent with removing these variables. Besides, all the **Effective Degrees of Freedom** are also automatically decided by penalizing.

And thus, this model could now be considered as the best model after processing subset selection.

Task 3

TREE:

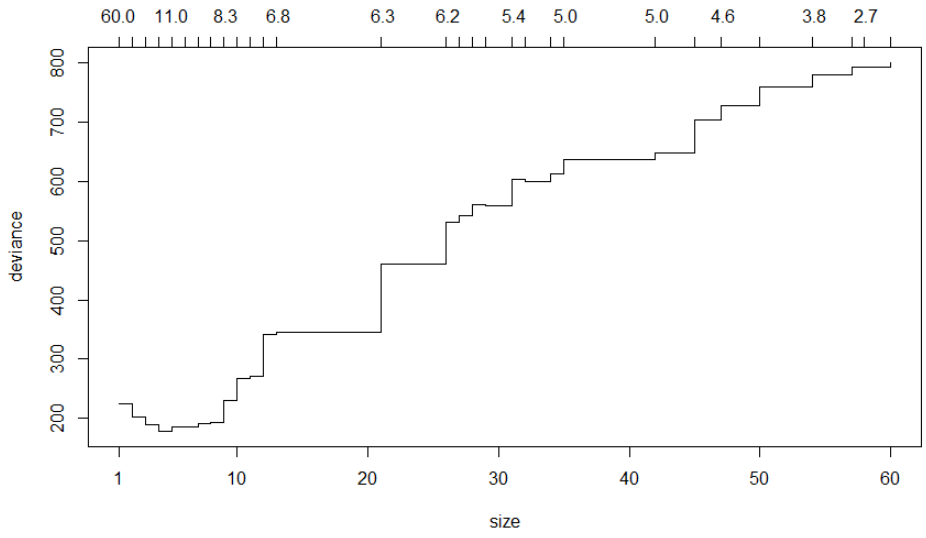
In order to generate a proper classification tree, we need to generate a huge tree at first and then try to prune it into the best one.

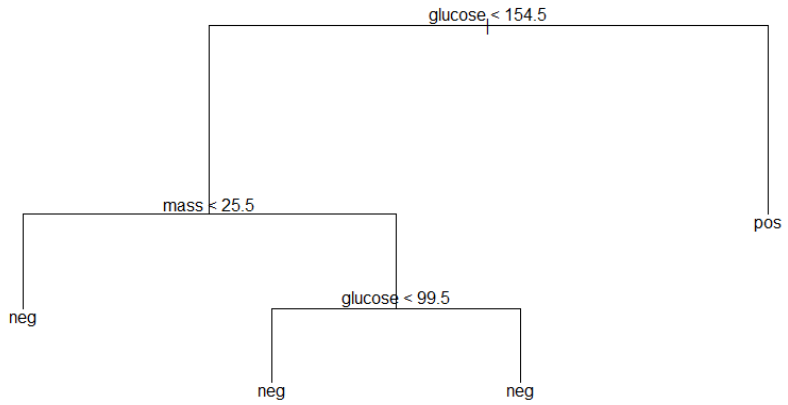
CODE:

```
library(tree)
n_random = sample(1:510,510)
tree_train_set <- dat[c(sort(n_random[1:340])),]
tree_test_set <- dat[c(sort(n_random[341:510])),]

```

```
6.  t1<- tree(diabetes~., data = tree_train_set, control = tree.control(nobs = 340, min
    size = 0, mindev = 0))
7.  plot(t1)
8.  text(t1)
9.
10. trees<- prune.tree(t1, newdata = tree_test_set)
11. plot(trees)
12.
13. tree_best <- prune.tree(t1, best = trees$size[trees$dev == min(trees$dev)])
14. plot(tree_best)
15. text(tree_best)
```

RESULT:





BAGGING:

By directly using the function **bagging** in the package **ipred**, we could simply implement a bagging trees model. In such case, there is not clearly improvement even if we increase the number of baggings, i.e. **nbagg**. Thus, we just use an empirical number 100.

CODE:

```
1. library(ipred)
2. bagging(diabetes~., nbagg=100, method="standard", data = train_set)
```

RESULT:

```
1. Bagging classification trees with 100 bootstrap replications
2.
3. Call: bagging.data.frame(formula = diabetes ~ ., data = train_set,
4.     nbagg = 100, method = "standard", coob = TRUE)
5.
6. Out-of-bag estimate of misclassification error:  0.2588
```

RANDOM FOREST:

According to the lectures, there might be pretty little improvement with tuning parameters, thus, we just directly implement the model by using package **randomForest** with default parameters. But notice that we need to at least ensure the parameter **ntree** is not too small, i.e. $1, 2$ and etc., in order to generate enough amount of trees.

We just simply use an empirical number **ntree**$= 25.$

CODE:

```
1. library(randomForest)
2. randomForest(diabetes~., ntree = 25, data = train_set)
```

RESULT:

```
1.         OOB estimate of  error rate: 26.08%
2. Confusion matrix:
3.     neg pos class.error
4. neg 271  57   0.1737805
5. pos  76 106   0.4175824
```

NEURAL NETWORK:

By using package **neuralnet**, we could simply implement a neural network model for our data. Since the package is using **rprop**+, so we do not need to tune the learning rate any more, and instead, it could be automatically adjusted.

CODE:

```
library(neuralnet)
model_nn <- neuralnet(diabetes~., data = train_set)
```

RESULT:

The details of neural networks are too much to be shown here.

ADABOOST:

In fact, this algorithm is pretty simple to be implemented, but in order to make the model more universal, i.e. could be directly used with **predict** and etc., we just use the package **fastAdaboost**, and since this package uses C++, as it is called, it runs faster, so we could just choose a larger amount of classifiers, **nIter**$= 200$**.**

CODE:

```
library(fastAdaboost)
model_ada <- adaboost(diabetes~., data = train_set, nIter = 200)
```

Task 4

Anyway, the numbers are reliable, thus, we just use the test set to test the performance at first.

CODE:

```
#task4
###test
error_tree <- sum((predict(model_tree, newdata = test_set[,-9])[,2]>0.5)!=
                    (test_set[,9]=="pos"))/258
error_bagging <- sum(predict(model_bagging, newdata = test_set[,-9])!=
                       test_set[,9])/258
error_rf <- sum(predict(model_rf, newdata = test_set[,-9])!=
                  test_set[,9])/258
error_nn <- sum((predict(model_nn, newdata = test_set[,-9])[,2]>0.5)!=
                  (test_set[,9]=="pos"))/258
error_ada <- sum((predict(model_ada, newdata = test_set[,-9])$prob[,2]>0.5)!=
                   (test_set[,9]=="pos"))/258
error_knn <- sum(fitted.values(model_knn)!=
                   test_set[,9])/258
error_gam <- sum((predict(model_gam, newdata = test_set[,-9])>1.5)!=
                   (test_set[,9]=="pos"))/258
print(c(error_tree,error_bagging,error_rf,error_nn,error_ada,error_knn,error_gam))
```

RESULT:

```
>print(c(error_tree,error_bagging,error_rf,error_nn,error_ada,error_knn,error_gam))
[1] 0.2558140 0.2480620 0.2325581 0.3449612 0.2596899 0.2790698 0.2403101
```

It seems that nearly all models have similar performance except **neural networks**, since the amount of data is too small to apply such an algorithm.

According to their performance and their interpretability, **I would like to choose classification tree** to analyse the data because of its **high interpretability**, although it does not have the best performance.

We are considering the reality.

Notice that the ability of predictions in such case is not so important, since it could pretty easy to check whether a person has diabetes. One could just go to hospital, and within one day, he/she could get the result of whether he/she has diabetes.

But instead, in reality, we are eager to figure out the relationship between the covariates and the response, which means that the interpretability are much more important than the prediction performance.

Then the choice could be pretty easy, since **classification tree** is much more interpretable than other models, especially when **bagging**, **random forest**, **neural network**, **Adaboost** and **knn** are nearly impossible to be interpreted and **GAM** are much harder to be interpreted than **classification tree**.

Task 5

By following the requirements of this task, we could easily change the data into that new dataset.

CODE:

```
data(PimaIndiansDiabetes2)
dat<-PimaIndiansDiabetes2[complete.cases(PimaIndiansDiabetes2),] #remove all NA obs
```

Then, we could just run the same code as before, and finally, the errors of each models could be:

```
>print(c(error_tree,error_bagging,error_rf,error_nn,error_ada,error_knn,error_gam))
[1] 0.2348485 0.1893939 0.1893939 0.3257576 0.2500000 0.2196970 0.2348485
```

COMMENTS:

Compared to the result of Task 4, we could easily find out that all errors are reduced with the new data.

This could be pretty understandable, since with changing the missing values into 0, we were actually creating wrong data and adding “noise” to the dataset. It could strongly influence the implementation of our model, since we are using biased data. But now, since we have no more “noise”, we are having a more precise result.

____END____