1. Setting1(MCAR)

Under the MCAR missing mechanism, we respectively consider the calculation of multi-source functional principal component score and canonical score of 2-source data (image and curve), as well as the corresponding classification evaluation index.

We have four main modules as follows:

setting_1_gamma(CCA): main program using the FR-CCA method in Setting 1 Case 1;

setting_1_gamma(PCA): main program using the FR-PCA method in Setting 1 Case 1;

setting_1_rho(CCA): main program using the FR-CCA method in Setting 1 Case 2;

setting_1_rho(PCA): main program using the FR-PCA method in Setting 1 Case 2.

The functions used in the main program are all listed after the four main programs.

###Module 1: setting_1_gamma(CCA) ###

```
library(stargazer)
library(funData)
library(caret)
library(MASS)
library(nnet)
N<-300        ## The sample size
M1<-25        ## The number of PC scores of x(1)
T1<-200       ## The number of sample point of curve x(1)
M2<-25        ## The number of PC scores of x(2)
T2<-200       ## The number of sample point of curve x(2)
k1<-10        ## The number of truncated PC scores of x(1) in Ma's paper
k2<-10        ## The number of truncated PC scores of x(2) in Ma's paper
## Regression coefficient vector
alpha1<-rep(0,10)
alpha2<-2*c(0.972,0.734,0.691,0.541,0.480,0.424,0.331,0.271,0.123,0.0405)
alpha3<-4*c(0.934,0.903,0.815,0.604,0.517,0.447,0.392,0.370,0.345,0.3)
M<-length(alpha1)            ## The number of mulrivariate truncated PC scores
sd<-0.2                      ## The standard deviation of error in the regression model
gamma<-0.3                   ## The coefficient in generating MFPC curves
miss_ratio<-0                ## Missing ratio .2 .6 .9
NN<-c()
NN[1]<-N*(1-miss_ratio)      ## The number of complete-data
NN[2]<-N*miss_ratio/2        ## The number of missing subjects in x(2)
NN[3]<-N*miss_ratio/2        ## The number of missing subjects in X(1)
Iter.times<-100
setwd("C:/Users/pc/Desktop/Logistic/Setting1(MCAR)/")
#####################################################################
######   Generating the eigenfunctions of setting 1 ######
source("generate_eigenfunction_setting1.R")
temp1 <- generate_eigenfunction_setting1(N, M1, T1, M2, T2)
t.1       <- temp1[[1]]
t.2       <- temp1[[2]]
phi.1     <- temp1[[3]]
phi.2     <- temp1[[4]]
t.1h      <- temp1[[5]]
t.2h      <- temp1[[6]]
#######   The end of Generating the eigenfunctions of setting 1 ######
#####################################################################
######## generating data ###################
psi.1_std<-psi.2_std<-rho_std<-f.hat_com<-y<-x_std<-vector('list',Iter.times)
source("generate_data_setting1_gamma.R")
for (times in 1:Iter.times){
  temp1 <- generate_data_setting1_gamma(phi.1, phi.2, alpha1, alpha2, alpha3, sd, gamma)
```

```
  rho_std[[times]]       <- temp1[[1]]
  x_std[[times]]         <- temp1[[2]]
  y[[times]]             <- temp1[[3]]
  psi.1_std[[times]]   <- temp1[[4]]
  psi.2_std[[times]]   <- temp1[[5]]
}
######## The end of generating data ########
M0=10
##################################################################
######### missing rate = 0 ###################
source("MCCA.R")
for (times in 1:Iter.times){
  temp2 <- MCCA(x_std[[times]], k1, k2, t.1h, t.2h)
  f.hat_com[[times]]     <- temp2[[1]]
}
Accuracy_com<-Precision_com<-Recall_com<-F1_com<-AIC_com<-array()
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
  temp5 <- Factor_regression_Imputed_CCA(f.hat_com[[times]], y[[times]][1:NN[1]])
  Accuracy_com[times]           <- temp5[[1]]
  Precision_com[times]          <- temp5[[2]]
  Recall_com[times]             <- temp5[[3]]
  F1_com[times]                 <- temp5[[4]]
  AIC_com[times]                <- temp5[[5]]
}
result_com <- round(c(mean(Accuracy_com),sd(Accuracy_com),mean(Precision_com),mean(Recall_com),mean(F1_com)),4)
print(result_com)
stargazer(result_com, title = "Evaluation",   align = F, type = "latex")
######### The end of missing rate = 0 ###################
# NN=c(60,270,270) # N=600,missing rate=0.9
# NN=c(30,135,135) # N=300,missing rate=0.9
M0=10
##################################################################
######### MCCA_complete_data #########
psi.1.hat<-psi.2.hat<-x.1.score_NA<-x.2.score_NA<-f.hat_com<-vector('list',Iter.times )
source("MCCA_complete_data.R")
for (times in 1:Iter.times){
  same<-MCCA_complete_data(NN, N, T1, T2, x_std[[times]],t.1h,t.2h)
  psi.1.hat[[times]]              <- same[[1]]
  psi.2.hat[[times]]              <- same[[2]]
  x.1.score_NA[[times]]           <- same[[3]]
  x.2.score_NA[[times]]           <- same[[4]]
  f.hat_com[[times]]              <- same[[5]]
}
######### The end of MFPCA_complete_data #########
##################################################################
######## Factor regression based on the CMI method #########
Accuracy_CMI<-Precision_CMI<-Recall_CMI<-F1_CMI<-AIC_CMI<-array()
x.1.score.hat<-x.2.score.hat<-f.hat_CMI<-vector('list',Iter.times)
source("Imputing_data_CMI.R")
for (times in 1:Iter.times){
  temp1 <- Imputing_data_CMI(NN, x_std[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]])
  x.1.score.hat[[times]]        <- temp1[[1]]
```

```
  x.2.score.hat[[times]]        <- temp1[[2]]
}
source("Construct_factor_CCA.R")
for (times in 1:Iter.times){
  temp2    <-    Construct_factor_CCA(x_std[[times]],    psi.1.hat[[times]],    psi.2.hat[[times]],    x.1.score.hat[[times]],
x.2.score.hat[[times]])
  f.hat_CMI[[times]] <- temp2[[1]]
}
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_CCA(f.hat_CMI[[times]][,1:M0], y[[times]])
  Accuracy_CMI[times]            <- temp3[[1]]
  Precision_CMI[times]          <- temp3[[2]]
  Recall_CMI[times]              <- temp3[[3]]
  F1_CMI[times]                  <- temp3[[4]]
  AIC_CMI[times]                 <- temp3[[5]]
}
######## The end of factor regression based on the CMI method #########
##################################################################
######## Factor regression based on the MBI method #########
Accuracy_MBI<-Precision_MBI<-Recall_MBI<-F1_MBI<-AIC_MBI<-array()
x.1.score.hat<-x.2.score.hat<-f.hat_MBI<-vector('list',Iter.times)
source("Imputing_data_MBI.R")
for (times in 1:Iter.times){
  temp1 <- Imputing_data_MBI(NN, x_std[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]])
  x.1.score.hat[[times]]        <- temp1[[1]]
  x.2.score.hat[[times]]        <- temp1[[2]]
}
source("Construct_factor_CCA.R")
for (times in 1:Iter.times){
  temp2    <-    Construct_factor_CCA(x_std[[times]],    psi.1.hat[[times]],    psi.2.hat[[times]],    x.1.score.hat[[times]],
x.2.score.hat[[times]])
  f.hat_MBI[[times]]            <- temp2[[1]]
}
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_CCA(f.hat_MBI[[times]][,1:M0], y[[times]])
  Accuracy_MBI[times]            <- temp3[[1]]
  Precision_MBI[times]          <- temp3[[2]]
  Recall_MBI[times]              <- temp3[[3]]
  F1_MBI[times]                  <- temp3[[4]]
  AIC_MBI[times]                 <- temp3[[5]]
}
######## The end of factor regression based on the MBI method #########
#result----
result                                                                                                              <-
round(rbind(   c(mean(Accuracy_CMI),sd(Accuracy_CMI),mean(Precision_CMI),mean(Recall_CMI),mean(na.omit(F1_CMI))),
c(mean(Accuracy_MBI),sd(Accuracy_MBI),mean(Precision_MBI),mean(Recall_MBI),mean(na.omit(F1_MBI)))),4)
print(result)
stargazer(result, title = "Evaluation",   align = F, type = "latex")
```

Module 2: setting_1_gamma(PCA)

```
library(stargazer)
library(funData)
```

```
library(caret)
library(MASS)
library(nnet)
N<-300        ## The sample size
M1<-25        ## The number of PC scores of x(1)
T1<-200       ## The number of sample point of curve x(1)
M2<-25        ## The number of PC scores of x(2)
T2<-200       ## The number of sample point of curve x(2)
k1<-10        ## The number of truncated PC scores of x(1) in Ma's paper
k2<-10        ## The number of truncated PC scores of x(2) in Ma's paper
## Regression coefficient vector
alpha1<-rep(0,10)
alpha2<-2*c(0.972,0.734,0.691,0.541,0.480,0.424,0.331,0.271,0.123,0.0405)
alpha3<-4*c(0.934,0.903,0.815,0.604,0.517,0.447,0.392,0.370,0.345,0.3)
M<-length(alpha1)              ## The number of mulrivariate truncated PC scores
sd<-0.2                        ## The standard deviation of error in the regression model
gamma<-0.3                       ## The coefficient in generating MFPC curves
miss_ratio<-0                  ## Missing ratio .2 .6 .9
NN<-c()
NN[1]<-N*(1-miss_ratio)     ## The number of complete-data
NN[2]<-N*miss_ratio/2       ## The number of missing subjects in x(2)
NN[3]<-N*miss_ratio/2       ## The number of missing subjects in X(1)
Iter.times<-300
setwd("C:/Users/pc/Desktop/Logistic/Setting1(MCAR)/")
######################################################################
######   Generating the eigenfunctions of setting 1 ######
source("generate_eigenfunction_setting1.R")
temp1 <- generate_eigenfunction_setting1(N, M1, T1, M2, T2)
t.1        <- temp1[[1]]
t.2        <- temp1[[2]]
phi.1      <- temp1[[3]]
phi.2      <- temp1[[4]]
t.1h       <- temp1[[5]]
t.2h       <- temp1[[6]]
#######   The end of Generating the eigenfunctions of setting 1 ######
######################################################################
######## generating data ###################
psi.1_std<-psi.2_std<-rho_std<-rho_std_com<-y<-x_std<-vector('list',Iter.times)
source("generate_data_setting1_gamma.R")
for (times in 1:Iter.times){
  temp1 <- generate_data_setting1_gamma(phi.1, phi.2, alpha1, alpha2, alpha3, sd, gamma)
  rho_std[[times]]      <- temp1[[1]]
  x_std[[times]]        <- temp1[[2]]
  y[[times]]            <- temp1[[3]]
  psi.1_std[[times]]    <- temp1[[4]]
  psi.2_std[[times]]    <- temp1[[5]]
}
######## The end of generating data ###################
M0=10
######################################################################
######### missing rate = 0 ###################
source("MFPCA.R")
for (times in 1:Iter.times){
```

```
  temp2 <- MFPCA(x_std[[times]], k1, k2, t.1h, t.2h)
  rho_std_com[[times]]     <- temp2[[1]]
}
Accuracy_com<-Precision_com<-Recall_com<-F1_com<-AIC_com<-array()
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
  temp5 <- Factor_regression_Imputed_PCA(rho_std_com[[times]], y[[times]][1:NN[1]])
  Accuracy_com[times]           <- temp5[[1]]
  Precision_com[times]          <- temp5[[2]]
  Recall_com[times]             <- temp5[[3]]
  F1_com[times]                 <- temp5[[4]]
  AIC_com[times]                <- temp5[[5]]
}
result_com <- round(c(mean(Accuracy_com),sd(Accuracy_com),
                      mean(Precision_com),mean(Recall_com),mean(F1_com)),4)
print(result_com)
stargazer(result_com, title = "Evaluation",   align = F, type = "latex")
######### The end of missing rate = 0 ####################
# NN=c(60,270,270) # N=600,missing rate=0.9
# NN=c(30,135,135) # N=300,missing rate=0.9
M0=10
##################################################################
######### MFPCA_complete_data #########
psi.1.hat<-psi.2.hat<-x.1.score_NA<-x.2.score_NA<-rho.hat_std_com<-vector('list',Iter.times )
source("MFPCA_complete_data.R")
for (times in 1:Iter.times){
  same<-MFPCA_complete_data(NN, N, T1, T2, x_std[[times]], t.1h, t.2h)
  psi.1.hat[[times]]            <- same[[1]]
  psi.2.hat[[times]]            <- same[[2]]
  x.1.score_NA[[times]]         <- same[[3]]
  x.2.score_NA[[times]]         <- same[[4]]
  rho.hat_std_com[[times]]      <- same[[5]]
}
######### The end of MFPCA_complete_data #########
##################################################################
######## Factor regression based on the CMI method #########
Accuracy_CMI<-Precision_CMI<-Recall_CMI<-F1_CMI<-AIC_CMI<-array()
x.1.score.hat<-x.2.score.hat<-rho.hat_std_CMI<-vector('list',Iter.times)
source("Imputing_data_CMI.R")
for (times in 1:Iter.times){
  temp1 <- Imputing_data_CMI(NN, x_std[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]])
  x.1.score.hat[[times]]        <- temp1[[1]]
  x.2.score.hat[[times]]        <- temp1[[2]]
}
source("Construct_factor_PCA.R")
for (times in 1:Iter.times){
  temp2 <- Construct_factor_PCA(x_std[[times]], psi.1.hat[[times]], psi.2.hat[[times]], x.1.score.hat[[times]],
x.2.score.hat[[times]])
  rho.hat_std_CMI[[times]]      <- temp2[[1]]
}
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_PCA(rho.hat_std_CMI[[times]], y[[times]])
```

```
  Accuracy_CMI[times]             <- temp3[[1]]
  Precision_CMI[times]           <- temp3[[2]]
  Recall_CMI[times]                <- temp3[[3]]
  F1_CMI[times]                      <- temp3[[4]]
  AIC_CMI[times]                    <- temp3[[5]]
}
######## The end of factor regression based on the CMI method #########
##################################################################
######## Factor regression based on the MBI method #########
Accuracy_MBI<-Precision_MBI<-Recall_MBI<-F1_MBI<-AIC_MBI<-array()
x.1.score.hat<-x.2.score.hat<-rho.hat_std_MBI<-vector('list',Iter.times)
source("Imputing_data_MBI.R")
for (times in 1:Iter.times){
  temp1 <- Imputing_data_MBI(NN, x_std[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]])
  x.1.score.hat[[times]]        <- temp1[[1]]
  x.2.score.hat[[times]]        <- temp1[[2]]
}
source("Construct_factor_PCA.R")
for (times in 1:Iter.times){
  temp2      <-      Construct_factor_PCA(x_std[[times]],      psi.1.hat[[times]],      psi.2.hat[[times]],      x.1.score.hat[[times]],
x.2.score.hat[[times]])
  rho.hat_std_MBI[[times]]     <- temp2[[1]]
}
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_PCA(rho.hat_std_MBI[[times]], y[[times]])
  Accuracy_MBI[times]             <- temp3[[1]]
  Precision_MBI[times]           <- temp3[[2]]
  Recall_MBI[times]                <- temp3[[3]]
  F1_MBI[times]                      <- temp3[[4]]
  AIC_MBI[times]                    <- temp3[[5]]
}
######## The end of factor regression based on the MBI method #########
#result----
result <- round(rbind(
c(mean(Accuracy_CMI),sd(Accuracy_CMI),mean(Precision_CMI),mean(Recall_CMI),mean(na.omit(F1_CMI))),
c(mean(Accuracy_MBI),sd(Accuracy_MBI),mean(Precision_MBI),mean(Recall_MBI),mean(na.omit(F1_MBI)))),4)
print(result)
stargazer(result, title = "Evaluation",   align = F, type = "latex")
```

###Module 3: setting_1_rho(CCA) ###

```
library(stargazer)
library(funData)
library(caret)
library(MASS)
library(nnet)
N<-300        ## The sample size
M1<-25        ## The number of PC scores of x(1)
T1<-200       ## The number of sample point of curve x(1)
M2<-25        ## The number of PC scores of x(2)
T2<-300       ## The number of sample point of curve x(2)
k1<-10        ## The number of truncated PC scores of x(1) in Ma's paper
k2<-10        ## The number of truncated PC scores of x(2) in Ma's paper
## Regression coefficient vector
```

```
alpha1<-rep(0,10)
alpha2<-2*c(0.972,0.734,0.691,0.541,0.480,0.424,0.331,0.271,0.123,0.0405)
alpha3<-4*c(0.934,0.903,0.815,0.604,0.517,0.447,0.392,0.370,0.345,0.3)
M<-length(alpha1)              ## The number of mulrivariate truncated PC scores
sd<-0.2                        ## The standard deviation of error in the regression model
rho_0<-0.4                     ## The correlation among data sources
miss_ratio<-0                  ## Missing ratio .2 .6 .9
NN<-c()
NN[1]<-N*(1-miss_ratio)     ## The number of complete-data
NN[2]<-N*miss_ratio/2       ## The number of missing subjects in x(2)
NN[3]<-N*miss_ratio/2       ## The number of missing subjects in X(1)
Iter.times<-300
setwd("C:/Users/pc/Desktop/Logistic/Setting1(MCAR)/")
####################################################################
######   Generating the eigenfunctions of setting 1 ######
source("generate_eigenfunction_setting1.R")
temp1 <- generate_eigenfunction_setting1(N, M1, T1, M2, T2)
t.1       <- temp1[[1]]
t.2       <- temp1[[2]]
phi.1     <- temp1[[3]]
phi.2     <- temp1[[4]]
t.1h      <- temp1[[5]]
t.2h      <- temp1[[6]]
#######   The end of Generating the eigenfunctions of setting 1 ######
####################################################################
######## generating data ###################
psi.1_std<-psi.2_std<-rho_std<-f.hat_com<-y<-x_std<-vector('list',Iter.times)
source("generate_data_setting1_rho.R")
for (times in 1:Iter.times) {
  temp1 <- generate_data_setting1_rho(rho_0, phi.1, phi.2, alpha1, alpha2, alpha3, sd)
  rho_std[[times]]      <- temp1[[1]]
  x_std[[times]]        <- temp1[[2]]
  y[[times]]            <- temp1[[3]]
  psi.1_std[[times]]   <- temp1[[4]]
  psi.2_std[[times]]   <- temp1[[5]]
}
######## The end of generating data ########
M0=10
####################################################################
######### missing rate = 0 ###################
source("MCCA.R")
for (times in 1:Iter.times){
  temp2 <- MCCA(x_std[[times]], k1, k2, t.1h, t.2h)
  f.hat_com[[times]]     <- temp2[[1]]
}
Accuracy_com<-Precision_com<-Recall_com<-F1_com<-AIC_com<-array()
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
  temp5 <- Factor_regression_Imputed_CCA(f.hat_com[[times]][1:NN[1],1:M0], y[[times]][1:NN[1]])
  Accuracy_com[times]          <- temp5[[1]]
  Precision_com[times]         <- temp5[[2]]
  Recall_com[times]            <- temp5[[3]]
  F1_com[times]                <- temp5[[4]]
```

```
  AIC_com[times]                    <- temp5[[5]]
}
result_com <- round(c(mean(Accuracy_com),sd(Accuracy_com),
                                    mean(Precision_com),mean(Recall_com),mean(F1_com)),4)
print(result_com)
stargazer(result_com, title = "Evaluation",    align = F, type = "latex")
######### The end of missing rate = 0 ###################
# NN=c(60,270,270) # N=600,missing rate=0.9
# NN=c(30,135,135) # N=300,missing rate=0.9
M0=10
##################################################################
######### MCCA_complete_data #########
psi.1.hat<-psi.2.hat<-x.1.score_NA<-x.2.score_NA<-f.hat_com<-vector('list',Iter.times )
source("MCCA_complete_data.R")
for (times in 1:Iter.times){
  same<-MCCA_complete_data(NN, N, T1, T2, x_std[[times]],t.1h,t.2h)
  psi.1.hat[[times]]              <- same[[1]]
  psi.2.hat[[times]]              <- same[[2]]
  x.1.score_NA[[times]]           <- same[[3]]
  x.2.score_NA[[times]]           <- same[[4]]
  f.hat_com[[times]]              <- same[[5]]
}
######### The end of MFPCA_complete_data #########
##################################################################
######## Factor regression based on the CMI method #########
Accuracy_CMI<-Precision_CMI<-Recall_CMI<-F1_CMI<-AIC_CMI<-array()
x.1.score.hat<-x.2.score.hat<-f.hat_CMI<-vector('list',Iter.times)
source("Imputing_data_CMI.R")
for (times in 1:Iter.times){
  temp1 <- Imputing_data_CMI(NN, x_std[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]])
  x.1.score.hat[[times]]        <- temp1[[1]]
  x.2.score.hat[[times]]        <- temp1[[2]]
}
source("Construct_factor_CCA.R")
for (times in 1:Iter.times){
  temp2    <-    Construct_factor_CCA(x_std[[times]],    psi.1.hat[[times]],    psi.2.hat[[times]],    x.1.score.hat[[times]],
x.2.score.hat[[times]])
  f.hat_CMI[[times]]              <- temp2[[1]]
}
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_CCA(f.hat_CMI[[times]][,1:M0], y[[times]])
  Accuracy_CMI[times]             <- temp3[[1]]
  Precision_CMI[times]            <- temp3[[2]]
  Recall_CMI[times]               <- temp3[[3]]
  F1_CMI[times]                   <- temp3[[4]]
  AIC_CMI[times]                  <- temp3[[5]]
}
######## The end of factor regression based on the CMI method #########
##################################################################
######## Factor regression based on the MBI method #########
Accuracy_MBI<-Precision_MBI<-Recall_MBI<-F1_MBI<-AIC_MBI<-array()
x.1.score.hat<-x.2.score.hat<-f.hat_MBI<-vector('list',Iter.times)
```

```
source("Imputing_data_MBI.R")
for (times in 1:Iter.times){
  temp1 <- Imputing_data_MBI(NN, x_std[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]])
  x.1.score.hat[[times]]        <- temp1[[1]]
  x.2.score.hat[[times]]        <- temp1[[2]]
}
source("Construct_factor_CCA.R")
for (times in 1:Iter.times){
  temp2 <- Construct_factor_CCA(x_std[[times]], psi.1.hat[[times]], psi.2.hat[[times]], x.1.score.hat[[times]],
x.2.score.hat[[times]])
  f.hat_MBI[[times]]            <- temp2[[1]]
}
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_CCA(f.hat_MBI[[times]][,1:M0], y[[times]])
  Accuracy_MBI[times]           <- temp3[[1]]
  Precision_MBI[times]          <- temp3[[2]]
  Recall_MBI[times]             <- temp3[[3]]
  F1_MBI[times]                 <- temp3[[4]]
  AIC_MBI[times]                <- temp3[[5]]
}
######## The end of factor regression based on the MBI method #########
#result----
result <- round(rbind(
c(mean(Accuracy_CMI),sd(Accuracy_CMI),mean(Precision_CMI),mean(Recall_CMI),mean(na.omit(F1_CMI))),
c(mean(Accuracy_MBI),sd(Accuracy_MBI),mean(Precision_MBI),mean(Recall_MBI),mean(na.omit(F1_MBI)))),4)
print(result)
stargazer(result, title = "Evaluation",  align = F, type = "latex")
### Module 4: setting_1_rho(PCA) ###
library(stargazer)
library(funData)
library(caret)
library(MASS)
library(nnet)
N<-300        ## The sample size
M1<-25        ## The number of PC scores of x(1)
T1<-200       ## The number of sample point of curve x(1)
M2<-25        ## The number of PC scores of x(2)
T2<-300       ## The number of sample point of curve x(2)
k1<-10        ## The number of truncated PC scores of x(1) in Ma's paper
k2<-10        ## The number of truncated PC scores of x(2) in Ma's paper
## Regression coefficient vector
alpha1<-rep(0,10)
alpha2<-2*c(0.972,0.734,0.691,0.541,0.480,0.424,0.331,0.271,0.123,0.0405)
alpha3<-4*c(0.934,0.903,0.815,0.604,0.517,0.447,0.392,0.370,0.345,0.3)
M<-length(alpha1)              ## The number of mulrivariate truncated PC scores
sd<-0.2                        ## The standard deviation of error in the regression model
rho_0<-0.4                     ## The correlation among data sources
miss_ratio<-0                  ## Missing ratio .2 .6 .9
NN<-c()
NN[1]<-N*(1-miss_ratio)     ## The number of complete-data
NN[2]<-N*miss_ratio/2       ## The number of missing subjects in x(2)
NN[3]<-N*miss_ratio/2       ## The number of missing subjects in X(1)
```

```
Iter.times<-300
setwd("C:/Users/pc/Desktop/Logistic/Setting1(MCAR)/")
#####################################################################
######    Generating the eigenfunctions of setting 1 ######
source("generate_eigenfunction_setting1.R")
temp1 <- generate_eigenfunction_setting1(N, M1, T1, M2, T2)
t.1          <- temp1[[1]]
t.2          <- temp1[[2]]
phi.1        <- temp1[[3]]
phi.2        <- temp1[[4]]
t.1h         <- temp1[[5]]
t.2h         <- temp1[[6]]
#######    The end of Generating the eigenfunctions of setting 1 ######
#####################################################################
######## generating data ####################
psi.1_std<-psi.2_std<-rho_std<-rho_std_com<-y<-x_std<-vector('list',Iter.times)
source("generate_data_setting1_rho.R")
for (times in 1:Iter.times) {
  temp1 <- generate_data_setting1_rho(rho_0, phi.1, phi.2, alpha1, alpha2, alpha3, sd)
  rho_std[[times]]       <- temp1[[1]]
  x_std[[times]]         <- temp1[[2]]
  y[[times]]             <- temp1[[3]]
  psi.1_std[[times]]   <- temp1[[4]]
  psi.2_std[[times]]   <- temp1[[5]]
}
######## The end of generating data ########
M0=10
#####################################################################
######### missing rate = 0 ####################
source("MFPCA.R")
for (times in 1:Iter.times){
  temp2 <- MFPCA(x_std[[times]], k1, k2, t.1h, t.2h)
  rho_std_com[[times]]     <- temp2[[1]]
}
Accuracy_com<-Precision_com<-Recall_com<-F1_com<-AIC_com<-array()
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
  temp5 <- Factor_regression_Imputed_PCA(rho_std_com[[times]], y[[times]][1:NN[1]])
  Accuracy_com[times]           <- temp5[[1]]
  Precision_com[times]          <- temp5[[2]]
  Recall_com[times]             <- temp5[[3]]
  F1_com[times]                 <- temp5[[4]]
  AIC_com[times]                <- temp5[[5]]
}
result_com <- round(c(mean(Accuracy_com),sd(Accuracy_com),mean(Precision_com),mean(Recall_com),mean(F1_com)),4)
print(result_com)
stargazer(result_com, title = "Evaluation",    align = F, type = "latex")
######### The end of missing rate = 0 ####################
# NN=c(60,270,270) # N=600,missing rate=0.9
# NN=c(30,135,135) # N=300,missing rate=0.9
M0=10
#####################################################################
######### MFPCA_complete_data #########
```

```
psi.1.hat<-psi.2.hat<-x.1.score_NA<-x.2.score_NA<-rho.hat_std_com<-vector('list',Iter.times )
source("MFPCA_complete_data.R")
for (times in 1:Iter.times){
  same<-MFPCA_complete_data(NN, N, T1, T2, x_std[[times]], t.1h, t.2h)
  psi.1.hat[[times]]              <- same[[1]]
  psi.2.hat[[times]]              <- same[[2]]
  x.1.score_NA[[times]]           <- same[[3]]
  x.2.score_NA[[times]]           <- same[[4]]
  rho.hat_std_com[[times]]     <- same[[5]]
}
######### The end of MFPCA_complete_data #########
##################################################################
######## Factor regression based on the CMI method #########
Accuracy_CMI<-Precision_CMI<-Recall_CMI<-F1_CMI<-AIC_CMI<-array()
x.1.score.hat<-x.2.score.hat<-rho.hat_std_CMI<-vector('list',Iter.times)
source("Imputing_data_CMI.R")
for (times in 1:Iter.times){
  temp1 <- Imputing_data_CMI(NN, x_std[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]])
  x.1.score.hat[[times]]        <- temp1[[1]]
  x.2.score.hat[[times]]        <- temp1[[2]]
}
source("Construct_factor_PCA.R")
for (times in 1:Iter.times){
  temp2  <-  Construct_factor_PCA(x_std[[times]],  psi.1.hat[[times]],  psi.2.hat[[times]],  x.1.score.hat[[times]],
x.2.score.hat[[times]])
  rho.hat_std_CMI[[times]]     <- temp2[[1]]
}
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_PCA(rho.hat_std_CMI[[times]], y[[times]])
  Accuracy_CMI[times]            <- temp3[[1]]
  Precision_CMI[times]           <- temp3[[2]]
  Recall_CMI[times]              <- temp3[[3]]
  F1_CMI[times]                  <- temp3[[4]]
  AIC_CMI[times]                 <- temp3[[5]]
}
######## The end of factor regression based on the CMI method #########
##################################################################
######## Factor regression based on the MBI method #########
Accuracy_MBI<-Precision_MBI<-Recall_MBI<-F1_MBI<-AIC_MBI<-array()
x.1.score.hat<-x.2.score.hat<-rho.hat_std_MBI<-vector('list',Iter.times)
source("Imputing_data_MBI.R")
for (times in 1:Iter.times){
  temp1 <- Imputing_data_MBI(NN, x_std[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]])
  x.1.score.hat[[times]]        <- temp1[[1]]
  x.2.score.hat[[times]]        <- temp1[[2]]
}
source("Construct_factor_PCA.R")
for (times in 1:Iter.times){
  temp2  <-  Construct_factor_PCA(x_std[[times]],  psi.1.hat[[times]],  psi.2.hat[[times]],  x.1.score.hat[[times]],
x.2.score.hat[[times]])
  rho.hat_std_MBI[[times]]     <- temp2[[1]]
}
```

```
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_PCA(rho.hat_std_MBI[[times]], y[[times]])
  Accuracy_MBI[times]          <- temp3[[1]]
  Precision_MBI[times]         <- temp3[[2]]
  Recall_MBI[times]            <- temp3[[3]]
  F1_MBI[times]                <- temp3[[4]]
  AIC_MBI[times]               <- temp3[[5]]
}
######## The end of factor regression based on the MBI method #########
#result----
result <- round(rbind(
c(mean(Accuracy_CMI),sd(Accuracy_CMI),mean(Precision_CMI),mean(Recall_CMI),mean(na.omit(F1_CMI))),
c(mean(Accuracy_MBI),sd(Accuracy_MBI),mean(Precision_MBI),mean(Recall_MBI),mean(na.omit(F1_MBI)))),4)
print(result)
stargazer(result, title = "Evaluation",  align = F, type = "latex")
```

Functions: all the functions used in the main program are listed below.

The function "Construct_factor_CCA" is used to construct canonical scores as factors.

```
Construct_factor_CCA = function(x_std, psi.1.hat, psi.2.hat, x.1.score_NA, x.2.score_NA){
  ### Calculate the estimate of x ###
  score<-cbind(x.1.score_NA,x.2.score_NA)
  z.hat<-t(score) %*% score /(N-1)
  rho.hat_std_temp<-x.1.score_NA%*% eigen(z.hat)$vec[1:k1,1:(k1+k2)] + x.2.score_NA%*% eigen(z.hat)$vec[(k1+1):(k1+k2),1:(k1+k2)]  ### Estimated MFPC scores with order N*(k1+k2) based on the correlation matrix
  x.hat_std<-rho.hat_std_temp %*% cbind(psi.1.hat,psi.2.hat)
  rho.hat_std<-scale(rho.hat_std_temp)[,1:M0]
  x_std[(sum(NN[1:2])+1):N,1:T1]<-x.hat_std[(sum(NN[1:2])+1):N,1:T1]
x_std[(NN[1]+1):sum(NN[1:2]),(T1+1):(T1+T2)]<-x.hat_std[(NN[1]+1):sum(NN[1:2]),(T1+1):(T1+T2)]
  ### Construct canonical scores as factor ###
  Z1 = x.1.score_NA
  Z2 = x.2.score_NA
  Omega1 = Z1%*%solve(t(Z1)%*%Z1)%*%t(Z1)
  Omega2 = Z2%*%solve(t(Z2)%*%Z2)%*%t(Z2)
  evd = eigen(Omega1 + Omega2)
  f.hat = evd$vectors[,1:M0]
  return(list(f.hat, x_std))
}
```

The function "Construct_factor_PCA" is used to construct multi-source functional principal component scores as factors.

```
Construct_factor_PCA = function(x_std, psi.1.hat, psi.2.hat, x.1.score_NA, x.2.score_NA){
  score<-cbind(x.1.score_NA,x.2.score_NA)
  z.hat<-t(score) %*% score /(N-1)
  rho.hat_std_temp<-x.1.score_NA%*% eigen(z.hat)$vec[1:k1,1:(k1+k2)] + x.2.score_NA%*% eigen(z.hat)$vec[(k1+1):(k1+k2),1:(k1+k2)]  ### Estimated MFPC scores with order N*(k1+k2) based on the correlation matrix
  x.hat_std<-rho.hat_std_temp %*% cbind(psi.1.hat,psi.2.hat)
  rho.hat_std<-scale(rho.hat_std_temp)[,1:M0]
  x_std[(sum(NN[1:2])+1):N,1:T1]<-x.hat_std[(sum(NN[1:2])+1):N,1:T1]
x_std[(NN[1]+1):sum(NN[1:2]),(T1+1):(T1+T2)]<-x.hat_std[(NN[1]+1):sum(NN[1:2]),(T1+1):(T1+T2)]
  return(list(rho.hat_std, x_std))
}
```

Function "Factor_regression_Imputed_CCA" was used to build Logistic regression model with canonical scores as factors.

```
Factor_regression_Imputed_CCA<-function(f, y){
  data = data.frame(cbind(f[,1:M0]), y)
  data$y = factor(data$y)
  mult.model = multinom(y ~ .-1, data = data, MaxNWts = 10000)
  predlab = predict(mult.model, newdata = data, type = "class")
  summary = multiClassSummary(
    data.frame(obs = data$y, pred = predlab), lev = levels(data$y))
  Accuracy = summary[1]
  F1 = summary[3]
  Precision = summary[8]
  Recall = summary[9]
  AIC = mult.model$AIC
  return(list(Accuracy, Precision, Recall, F1, AIC))
}
```

Function "Factor_regression_Imputed_PCA" was used to build Logistic regression model with multi-source functional principal component scores as factors.

```
Factor_regression_Imputed_PCA<-function(rho.hat_std, y){
  data = data.frame(cbind(rho.hat_std[,1:M0]), y)
  data$y = factor(data$y)
  mult.model = multinom(y ~ .-1, data = data, MaxNWts = 10000)
  predlab = predict(mult.model, newdata = data, type = "class")
  summary = multiClassSummary(
    data.frame(obs = data$y, pred = predlab), lev = levels(data$y))
  Accuracy = summary[1]
  F1 = summary[3]
  Precision = summary[8]
  Recall = summary[9]
  AIC = mult.model$AIC
  return(list(Accuracy, Precision, Recall, F1, AIC))
}
```

The function "generate_data_setting1_gamma" generates data under Setting1: Case 1.

```
generate_data_setting1_gamma<-function(phi.1, phi.2, alpha1, alpha2, alpha3, sd, gamma){
  psi.1 <- phi.1*sqrt(gamma)                ## (M11 * M12) * (T11 * T12) tensor
  psi.2 <- phi.2*sqrt(1-gamma)              ## M2 * T2 matrix
  ## Generating the 2-source functional data ##
  rho = apply(matrix(1:M2,nrow=M2,ncol=1), 1, function(x){return(rnorm(N,0,(exp(-(1+x)/2))^0.5))})    ### Multivariate Functional PC scores matrix with order N*M2 based on Covariance matrix
  x.1 = rho %*% psi.1                        # N * T1 matrix  of 1st data source: Image x(1)
  x.2 = rho %*% psi.2                        # N * T2  matrix of 2nd data source: curve x(2)
  x = cbind(x.1,x.2)                         # N * (T1+T2)  matrix of 2-source data x
  ## The end of Generating the 2-source functional data ##
  rho_std = scale(rho)
  p.1=exp(rho_std[,1:M]%*% alpha1)/(exp(rho_std[,1:M]%*% alpha1)+exp(rho_std[,1:M]%*% alpha2)+exp(rho_std[,1:M]%*% alpha3))
  p.2 = exp(rho_std[,1:M]%*% alpha2)/(exp(rho_std[,1:M]%*% alpha1)+exp(rho_std[,1:M]%*% alpha2)+exp(rho_std[,1:M]%*% alpha3))
  p.3 = exp(rho_std[,1:M]%*% alpha3)/(exp(rho_std[,1:M]%*% alpha1)+exp(rho_std[,1:M]%*% alpha2)+exp(rho_std[,1:M]%*% alpha3))
  p.0 = cbind(p.1, p.2, p.3)
  y=c()
  for(i in 1:N){
```

```
    y0 = which.max(rmultinom(1, size = 1, prob = p.0[i,]))
    y=c(y,y0)
  }
  return(list(rho_std, x, y, psi.1, psi.2))
}
### The function "generate_data_setting1_rho" generates data under Setting1: Case 2. ###
generate_data_setting1_rho<-function(rho_0, phi.1, phi.2, alpha1, alpha2, alpha3, sd){
  # generating the covariance-variance matrix of two data sources.
  sigma<-matrix(NA,M1+M2,M1+M2)
  sigma[1:M1,1:M1]<-diag(exp(-(2:(M1+1))/2))
  sigma[(M1+1):(M1+M2),(M1+1):(M1+M2)]<-diag(exp(-(2:(M2+1))/2))
  R<-matrix(0,M1,M2)
  for (i in 1:M1){
    for (j in 1:M2){
      R[i,j]<-rho_0^(abs(i-j)+1)
    }
  }
  sigma[1:M1,(M1+1):(M1+M2)]<-sigma[1:M1,1:M1]^0.5 %*% R %*% sigma[(M1+1):(M1+M2),(M1+1):(M1+M2)]^0.5
  sigma[(M1+1):(M1+M2),1:M1]<-t(sigma[1:M1,(M1+1):(M1+M2)])
  rr2<-eigen(sigma%*%sigma)
  sigma0.5<-rr2$vectors%*%diag(rr2$values^(0.5))%*%t(rr2$vectors)
  # The end of generating the covariance-variance matrix of two data sources.

  ## Generating the 2-source functional data ########
  ksi<-mvrnorm(N,rep(0,M1+M2),sigma0.5)     # ksi is an N*(M1+M2) matrix
  ksi.1<-ksi[,1:M1]                          # The PC socres of x.1
  ksi.2<-ksi[,(M1+1):(M1+M2)]                # The PC socres of x.2
  x.1<-ksi.1 %*% phi.1
  x.2<-ksi.2 %*% phi.2
  Z<-t(ksi) %*% ksi/(N-1)
  psi.1<-t(eigen(Z)$vec[1:M1,]) %*% phi.1              # MFPC curves of order (M1+M2)*T1
  psi.2<-t(eigen(Z)$vec[(M1+1):(M1+M2),]) %*%phi.2     # MFPC curves of order (M1+M2)*T2
  rho<-ksi.1 %*% eigen(Z)$vec[1:M1,1:(M1+M2)] + ksi.2 %*% eigen(Z)$vec[(M1+1):(M1+M2),1:(M1+M2)] # MFPC scores
matrix with order N*(M1+M2)
  x<-rho %*% cbind(psi.1,psi.2)
  ## The end of Generating the 2-source functional data ########
  rho_std = scale(rho)
  p.1 = exp(rho_std[,1:M]%*% alpha1)/(exp(rho_std[,1:M]%*% alpha1)+exp(rho_std[,1:M]%*% alpha2)+exp(rho_std[,1:M]%*%
alpha3))
  p.2 = exp(rho_std[,1:M]%*% alpha2)/(exp(rho_std[,1:M]%*% alpha1)+exp(rho_std[,1:M]%*% alpha2)+exp(rho_std[,1:M]%*%
alpha3))
  p.3 = exp(rho_std[,1:M]%*% alpha3)/(exp(rho_std[,1:M]%*% alpha1)+exp(rho_std[,1:M]%*% alpha2)+exp(rho_std[,1:M]%*%
alpha3))
  p = cbind(p.1, p.2, p.3)
  y=c()
  for(i in 1:N){
    y0 = which.max(rmultinom(1, size = 1, prob = p[i,]))
    y=c(y,y0)
  }
  return(list(rho_std, x, y, psi.1, psi.2))
}
### The function "generate_eigenfunction_setting1" generates eigenfunctions under Setting1. ###
generate_eigenfunction_setting1<-function(N, M1, T1, M2, T2){
```

```
  fai.1<-eFun(seq(0,4,length.out = T1), M = M1, type = "Fourier")   # the first one is the Fourier basis defined on the interval [0,4]
  fai.2<-eFun(seq(-1,1,length.out = T2), M = M2, type = "Wiener")   # the second one is the Wiener basis defined on the interval [-1,1]
  t.1<-fai.1@argvals[[1]]
  t.1h<-t.1[T1]-t.1[1]
  t.2<-fai.2@argvals[[1]]
  t.2h<-t.2[T2]-t.2[1]
  phi.1<-fai.1@X   # phi.1 is an M1*T1 matrix
  phi.2<-fai.2@X   # phi.2 is an M2*T2 matrix
  return(list(t.1, t.2, phi.1, phi.2,   t.1h, t.2h))
}
```

The function "Imputing_data_CMI" is using the conditional mean imputation method for univariate principal component scores.

```
Imputing_data_CMI<-function(NN, x_std, x.1.score_NA, x.2.score_NA){
  x.2.score_NA[!complete.cases(x.2.score_NA),] = mean(na.omit(x.2.score_NA))
  x.1.score_NA[!complete.cases(x.1.score_NA),] = mean(na.omit(x.1.score_NA))
  n_iter = 10
  for(j in 1:n_iter)
  {
    x.2.score.hat = matrix(0,NN[2],k2)
    for (i in 1:k2) {
      fit = lm(x.2.score_NA[,i]~., data = as.data.frame(x.1.score_NA))
      x.2.score.hat[,i] = cbind(rep(1,NN[2]),x.1.score_NA[(NN[1]+1):sum(NN[1:2]),])%*%fit$coefficients
    }
    x.2.score_NA[(NN[1]+1):sum(NN[1:2]),] = x.2.score.hat
    x.1.score.hat = matrix(0,NN[3],k1)
    for (i in 1:k1) {
      fit = lm(x.1.score_NA[,i]~., data = as.data.frame(x.2.score_NA))
      x.1.score.hat[,i] = cbind(rep(1,NN[3]),x.2.score_NA[(sum(NN[1:2])+1):N,])%*%fit$coefficients
    }
    x.1.score_NA[(sum(NN[1:2])+1):N,] = x.1.score.hat
  }
  return(list(x.1.score_NA, x.2.score_NA))
}
```

The function "Imputing_data_MBI" is using the multiple block-wise imputation method for univariate principal component scores.

```
Imputing_data_MBI<-function(NN, x_std, x.1.score_NA, x.2.score_NA){
  x.2.score.hat = matrix(0,NN[2],k2)
  for (i in 1:k2) {
    fit = lm(x.2.score_NA[1:NN[1],i]~., data = as.data.frame(x.1.score_NA[1:NN[1],]))
    x.2.score.hat[,i] = cbind(rep(1,NN[2]),x.1.score_NA[(NN[1]+1):sum(NN[1:2]),])%*%fit$coefficients
  }
  x.2.score_NA[(NN[1]+1):sum(NN[1:2]),] = x.2.score.hat
  x.1.score.hat = matrix(0,NN[3],k1)
  for (i in 1:k1) {
    fit = lm(x.1.score_NA[1:NN[1],i]~., data = as.data.frame(x.2.score_NA[1:NN[1],]))
    x.1.score.hat[,i] = cbind(rep(1,NN[3]),x.2.score_NA[(sum(NN[1:2])+1):N,])%*%fit$coefficients
  }
  x.1.score_NA[(sum(NN[1:2])+1):N,] = x.1.score.hat
  return(list(x.1.score_NA, x.2.score_NA))
}
```

Function "MCCA" is used to conduct multi-set canonical correlation analysis on all data and construct canonical scores.

```
###
MCCA <- function(x_co, k1, k2, t.1h, t.2h){
  ###################################################
  ##### Computing the FPC scores and curves for each data source by SVD method #############
  s1<-svd(x_co[,1:T1]/sqrt(N) )                                                  ## SVD for x.1 on the all data
  phi.1.hat<-t(s1$v[,1:k1]) * (t.1h/T1)^(-0.5)                                  ## The univariate FPC curves of x.1 with order k1*T1
  x.1.score<-x_co[,1:T1] %*% t(phi.1.hat) * (t.1h/T1)                           ## The univariate FPC scores of x.1 with order N*k1
  s2<-svd(x_co[,(T1+1):(T1+T2)]/sqrt(N) )                                         ## SVD for x.2 on the all data
  phi.2.hat<-t(s2$v[,1:k2]) * (t.2h/T2)^(-0.5)                                    ## The univariate FPC curves of x.2 with orderk2*T2
  x.2.score<-x_co[,(T1+1):(T1+T2)] %*% t(phi.2.hat) * (t.2h/T2)              ## The univariate FPC score of x.2 with order N*k2
  ##### The end of Computing the FPC scores and curves for each data source by SVD method #############
  ###################################################
  ##### Computing the Canonical scores #############
  Z1 = x.1.score
  Z2 = x.2.score
  Omega1 = Z1%*%solve(t(Z1)%*%Z1)%*%t(Z1)
  Omega2 = Z2%*%solve(t(Z2)%*%Z2)%*%t(Z2)
  evd = eigen(Omega1 + Omega2)
  f.hat = evd$vectors[,1:M0]
  #####The end of    Computing the Canonical scores #############
  return(list(f.hat))
}
### Function "MCCA_complete_data" is used to conduct multi-set canonical correlation analysis on complete data and construct canonical scores. ###
MCCA_complete_data<-function(NN, N, T1, T2, x_std, t.1h, t.2h){
  ##### Computing the FPC scores and curves for each data source by SVD method #############
  s1<-svd(x_std[1:sum(NN[1:2]),1:T1]/sqrt(sum(NN[1:2])))          ## SVD for x.1 on the observed data
  phi.1.hat<-t(s1$v[,1:k1]) * (t.1h/T1)^(-0.5)                            ## The univariate FPC curves of x.1 with order k1*T1
  x.1.score<-x_std[1:sum(NN[1:2]),1:T1] %*% t(phi.1.hat) * (t.1h/T1)     ## The univariate FPC scores of x.1 with order (N1+N2)*k1
  x.1.score_NA<-matrix(NA,N,k1)
  x.1.score_NA[1:sum(NN[1:2]),]<- x.1.score
  s2<-svd(x_std[c(1:NN[1],(sum(NN[1:2])+1):N),(T1+1):(T1+T2)]/sqrt(N-NN[2]))   ## SVD for x.2 on the observed data
  phi.2.hat<-t(s2$v[,1:k2]) * (t.2h/T2)^(-0.5)                       ## The univariate FPC curves of x.2 with order k2*T2
  x.2.score<-x_std[c(1:NN[1],(sum(NN[1:2])+1):N),(T1+1):(T1+T2)] %*% t(phi.2.hat) * (t.2h/T2)          ##    The univariate FPC score of x.2 with order (N-N2)*k2
  x.2.score_NA<-matrix(NA,N,k2)
  x.2.score_NA[1:NN[1],]<- x.2.score[1:NN[1],]
  x.2.score_NA[(sum(NN[1:2])+1):N,]<- x.2.score[(NN[1]+1):(NN[1]+NN[3]),]
  ##### The end of Computing the FPC scores and curves for each data source by SVD method #############
  score<-cbind(x.1.score[1:NN[1],],x.2.score[1:NN[1],])      ##    N1*(k1+k2)
  z.hat<-t(score) %*% score /(NN[1]-1)
  psi.1.hat<-t(eigen(z.hat)$vec[1:k1,1:(k1+k2)]) %*% phi.1.hat     ## The estimated Multivariate FPC eigenfunctions of x.1 with order (k1+k2)*T1
  psi.2.hat<-t(eigen(z.hat)$vec[(k1+1):(k1+k2),1:(k1+k2)]) %*% phi.2.hat ## The estimated Multivariate FPC eigenfunctions of x.2 with order (k1+k2)*T2
  ######### The Canonical scores of x.1 and x.2 on the complete-data #########
  Z1 = x.1.score[1:NN[1],]
  Z2 = x.2.score[1:NN[1],]
  Omega1 = Z1%*%solve(t(Z1)%*%Z1)%*%t(Z1)
  Omega2 = Z2%*%solve(t(Z2)%*%Z2)%*%t(Z2)
```

```
  evd = eigen(Omega1 + Omega2)
  f_com = evd$vectors[,1:M0]
  return(list(psi.1.hat,psi.2.hat,x.1.score_NA,x.2.score_NA,f_com))
}
```

Function "MFPCA" is used to conduct multi-source functional principal component analysis on all data and construct multi-source principal component scores.

```
MFPCA <- function(x_co, k1, k2, t.1h, t.2h){
  ################################################
  ##### Computing the FPC scores and curves for each data source by SVD method ############
  s1<-svd(x_co[,1:T1]/sqrt(N) )                                    ## SVD for x.1 on the all data
  phi.1.hat<-t(s1$v[,1:k1]) * (t.1h/T1)^(-0.5)                       ## The univariate FPC curves of x.1 with order k1*T1
  x.1.score<-x_co[,1:T1] %*% t(phi.1.hat) * (t.1h/T1)                ## The univariate FPC scores of x.1 with order N*k1
  s2<-svd(x_co[,(T1+1):(T1+T2)]/sqrt(N) )                            ## SVD for x.2 on the all data
  phi.2.hat<-t(s2$v[,1:k2]) * (t.2h/T2)^(-0.5)                       ## The univariate FPC curves of x.2 with orderk2*T2
  x.2.score<-x_co[,(T1+1):(T1+T2)] %*% t(phi.2.hat) * (t.2h/T2)      ## The univariate FPC score of x.2 with order N*k2
  ##### The end of Computing the FPC scores and curves for each data source by SVD method ############
  ################################################
  ##### Computing the Multi-source FPC scores and curves by the method in Ma's paper############

  score<-cbind(x.1.score,x.2.score)
  z.hat<-t(score) %*% score /(N-1)
  psi.1.hat<-t(eigen(z.hat)$vec[1:k1,1:(k1+k2)]) %*% phi.1.hat        ## The estimated Multivariate FPC eigenfunctions of x.1 with order (k1+k2)*T1
  psi.2.hat<-t(eigen(z.hat)$vec[(k1+1):(k1+k2),1:(k1+k2)]) %*% phi.2.hat   ## The estimated Multivariate FPC eigenfunctions of x.2 with order (k1+k2)*T2
  rho_std_temp<-x.1.score%*% eigen(z.hat)$vec[1:k1,1:(k1+k2)] + x.2.score %*% eigen(z.hat)$vec[(k1+1):(k1+k2),1:(k1+k2)] ## The estimated multivariate FPC scores with order N*(k1+k2) of the complete-data
  rho_std<-scale(rho_std_temp)[,1:M0]
  #####The end of  Computing the Multi-source FPC scores and curves by the method in Ma's paper############
  ################################################
  return(list(rho_std, psi.1.hat, psi.2.hat))
}
```

Function "MFPCA_complete_data" is used to conduct multi-source functional principal component analysis on complete data and construct multi-source principal component scores.

```
MFPCA_complete_data<-function(NN, N, T1, T2, x_std, t.1h, t.2h){
  ##### Computing the FPC scores and curves for each data source by SVD method ############
  s1<-svd(x_std[1:sum(NN[1:2]),1:T1]/sqrt(sum(NN[1:2])))              ## SVD for x.1 on the observed data
  phi.1.hat<-t(s1$v[,1:k1]) * (t.1h/T1)^(-0.5)                        ## The univariate FPC curves of x.1 with order k1*T1
  x.1.score<-x_std[1:sum(NN[1:2]),1:T1] %*% t(phi.1.hat) * (t.1h/T1)   ## The univariate FPC scores of x.1 with order (N1+N2)*k1
  x.1.score_NA<-matrix(NA,N,k1)
  x.1.score_NA[1:sum(NN[1:2]),]<- x.1.score
  s2<-svd(x_std[c(1:NN[1],(sum(NN[1:2])+1):N),(T1+1):(T1+T2)]/sqrt(N-NN[2])) ## SVD for x.2 on the observed data
  phi.2.hat<-t(s2$v[,1:k2]) * (t.2h/T2)^(-0.5)                        ## The univariate FPC curves of x.2 with order k2*T2
  x.2.score<-x_std[c(1:NN[1],(sum(NN[1:2])+1):N),(T1+1):(T1+T2)] %*% t(phi.2.hat) * (t.2h/T2)     ##  The univariate FPC score of x.2 with order (N-N2)*k2
  x.2.score_NA<-matrix(NA,N,k2)
  x.2.score_NA[1:NN[1],]<- x.2.score[1:NN[1],]
  x.2.score_NA[(sum(NN[1:2])+1):N,]<- x.2.score[(NN[1]+1):(NN[1]+NN[3]),]
  ##### The end of Computing the FPC scores and curves for each data source by SVD method ############
  ######### The multivariate FPCA of x.1 and x.2 on the complete-data #########
```

```
  score<-cbind(x.1.score[1:NN[1],],x.2.score[1:NN[1],])    ##   N1*(k1+k2)
  z.hat<-t(score) %*% score /(NN[1]-1)
  psi.1.hat<-t(eigen(z.hat)$vec[1:k1,1:(k1+k2)]) %*% phi.1.hat    ## The estimated Multivariate FPC eigenfunctions of x.1 with order (k1+k2)*T1
  psi.2.hat<-t(eigen(z.hat)$vec[(k1+1):(k1+k2),1:(k1+k2)]) %*% phi.2.hat ## The estimated Multivariate FPC eigenfunctions of x.2 with order (k1+k2)*T2
  rho.hat_std_com_t<-x.1.score[1:NN[1],]%*%  eigen(z.hat)$vec[1:k1,1:(k1+k2)]  +  x.2.score[1:NN[1],]  %*% eigen(z.hat)$vec[(k1+1):(k1+k2),1:(k1+k2)]   # The estimated multivariate FPC scores with order N1*(k1+k2) of the complete-data
  rho.hat_std_com<-scale(rho.hat_std_com_t)[,1:M0]
  return(list(psi.1.hat,psi.2.hat,x.1.score_NA,x.2.score_NA,rho.hat_std_com))
}
```

2. Setting1(MNAR)

Under the MNAR missing mechanism, we respectively consider the calculation of multi-source functional principal component score and canonical score of 2-source data (image and curve), as well as the corresponding classification evaluation index. “N” represents the sample size.

We have four main modules as follows:

"setting_1_gamma(CCA)" represents the main program using the FR-CCA method in Setting 1: Case 1;

"setting_1_gamma(PCA)" represents the main program using the FR-PCA method in Setting 1: Case 1;

"setting_1_rho(CCA)" represents the main program using the FR-CCA method in Setting 1: Case 2;

"setting_1_rho(PCA)" represents the main program using the FR-PCA method in Setting 1: Case 2.

Module 1: setting_1_gamma(CCA)

```
library(stargazer)
library(funData)
library(caret)
library(MASS)
library(nnet)
N<-300        ## The sample size
M1<-25        ## The number of PC scores of x(1)
T1<-200       ## The number of sample point of curve x(1)
M2<-25        ## The number of PC scores of x(2)
T2<-200       ## The number of sample point of curve x(2)
k1<-10        ## The number of truncated PC scores of x(1) in Ma's paper
k2<-10        ## The number of truncated PC scores of x(2) in Ma's paper
## Regression coefficient vector
alpha1<-rep(0,10)
alpha2<-2*c(0.972,0.734,0.691,0.541,0.480,0.424,0.331,0.271,0.123,0.0405)
alpha3<-4*c(0.934,0.903,0.815,0.604,0.517,0.447,0.392,0.370,0.345,0.3)
M<-length(alpha1)              ## The number of mulrivariate truncated PC scores
sd<-0.2                        ## The standard deviation of error in the regression model
gamma<-0.3                     ## The coefficient in generating MFPC curves
## miss_ratio<-0 ##
gam1.1 = rep(1, (T1+T2))
gam1.2 = gam1.3 = rep(0, (T1+T2))
gam2.1 = rep(1, (T1+T2))
gam2.2 = gam2.3 = rep(0, (T1+T2))
gam3.1 = rep(1, (T1+T2))
gam3.2 = gam3.3 = rep(0, (T1+T2))
## miss_ratio<-0.2 ##
gam1.1 = rep(1, (T1+T2))
gam1.2 = gam1.3 = rep(7/51, (T1+T2))
gam2.1 = rep(1, (T1+T2))
gam2.2 = gam2.3 = rep(1/18, (T1+T2))
```

```
gam3.1 = rep(1, (T1+T2))
gam3.2 = gam3.3 = rep(7/51, (T1+T2))
## miss_ratio<-0.6 ##
gam1.1 = rep(1, (T1+T2))
gam1.2 = gam1.3 = rep(21/23, (T1+T2))
gam2.1 = rep(1, (T1+T2))
gam2.2 = gam2.3 = rep(3/14, (T1+T2))
gam3.1 = rep(1, (T1+T2))
gam3.2 = gam3.3 = rep(21/23, (T1+T2))
## miss_ratio<-0.9 ##
gam1.1 = rep(1, (T1+T2))
gam1.2 = gam1.3 = rep(63/4, (T1+T2))
gam2.1 = rep(1, (T1+T2))
gam2.2 = gam2.3 = rep(9/22, (T1+T2))
gam3.1 = rep(1, (T1+T2))
gam3.2 = gam3.3 = rep(63/4, (T1+T2))
Iter.times<-300
setwd("C:/Users/pc/Desktop/Logistic/Setting1(MNAR)/")
####################################################################
######    Generating the eigenfunctions of setting 1 ######
source("generate_eigenfunction_setting1.R")
temp1 <- generate_eigenfunction_setting1(N, M1, T1, M2, T2)
t.1          <- temp1[[1]]
t.2          <- temp1[[2]]
phi.1        <- temp1[[3]]
phi.2        <- temp1[[4]]
t.1h         <- temp1[[5]]
t.2h         <- temp1[[6]]
#######    The end of Generating the eigenfunctions of setting 1 ######
######## generating data ####################
psi.1_std<-psi.2_std<-rho_std<-f.hat_com<-y<-x_std<-vector('list',Iter.times)
source("generate_data_setting1_gamma.R")
for (times in 1:Iter.times){
   temp1 <- generate_data_setting1_gamma(phi.1, phi.2, alpha1, alpha2, alpha3, sd, gamma)
   rho_std[[times]]       <- temp1[[1]]
   x_std[[times]]         <- temp1[[2]]
   y[[times]]             <- temp1[[3]]
   psi.1_std[[times]]   <- temp1[[4]]
   psi.2_std[[times]]   <- temp1[[5]]
}
######## The end of generating data ########
######## generating missing pattern ####################
source("generate_missing_pattern.R")
for (times in 1:Iter.times){
   temp2 <- generate_missing_pattern(rho_std[[times]], x_std[[times]], y[[times]])
   NN                          <- temp2[[1]]
   rho_std[[times]]       <- temp2[[2]]
   x_std[[times]]         <- temp2[[3]]
   y[[times]]             <- temp2[[4]]
}
######## The end of generating missing pattern ########
M0=10
######### missing rate = 0 ###################
```

```
source("MCCA.R")
for (times in 1:Iter.times){
   temp2 <- MCCA(x_std[[times]], k1, k2, t.1h, t.2h)
   f.hat_com[[times]]     <- temp2[[1]]
}
Accuracy_com<-Precision_com<-Recall_com<-F1_com<-AIC_com<-array()
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
   temp5 <- Factor_regression_Imputed_CCA(f.hat_com[[times]][1:NN[1],1:M0], y[[times]][1:NN[1]])
   Accuracy_com[times]           <- temp5[[1]]
   Precision_com[times]          <- temp5[[2]]
   Recall_com[times]             <- temp5[[3]]
   F1_com[times]                 <- temp5[[4]]
   AIC_com[times]                <- temp5[[5]]
}
result_com <- round(c(mean(Accuracy_com),sd(Accuracy_com),
                      mean(Precision_com),mean(Recall_com),mean(F1_com)),4)
print(result_com)
stargazer(result_com, title = "Evaluation",   align = F, type = "latex")
######### The end of missing rate = 0 ####################
M0=10
######### MCCA_complete_data #########
psi.1.hat<-psi.2.hat<-x.1.score_NA<-x.2.score_NA<-f.hat_com<-vector('list',Iter.times )
source("MCCA_complete_data.R")
for (times in 1:Iter.times){
   same<-MCCA_complete_data(NN, N, T1, T2, x_std[[times]],t.1h,t.2h)
   psi.1.hat[[times]]            <- same[[1]]
   psi.2.hat[[times]]            <- same[[2]]
   x.1.score_NA[[times]]         <- same[[3]]
   x.2.score_NA[[times]]         <- same[[4]]
   f.hat_com[[times]]            <- same[[5]]
}
######### The end of MFPCA_complete_data #########
######## Factor regression based on the CMI method #########
Accuracy_CMI<-Precision_CMI<-Recall_CMI<-F1_CMI<-AIC_CMI<-array()
x.1.score.hat<-x.2.score.hat<-f.hat_CMI<-vector('list',Iter.times)
source("Imputing_data_CMI.R")
for (times in 1:Iter.times){
   temp1 <- Imputing_data_CMI(NN, y[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]])
   x.1.score.hat[[times]]      <- temp1[[1]]
   x.2.score.hat[[times]]      <- temp1[[2]]
}
source("Construct_factor_CCA.R")
for (times in 1:Iter.times){
   temp2    <-    Construct_factor_CCA(x_std[[times]],    psi.1.hat[[times]],    psi.2.hat[[times]],    x.1.score.hat[[times]],
x.2.score.hat[[times]])
   f.hat_CMI[[times]]            <- temp2[[1]]
}
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
   temp3 <- Factor_regression_Imputed_CCA(f.hat_CMI[[times]][,1:M0], y[[times]])
   Accuracy_CMI[times]           <- temp3[[1]]
   Precision_CMI[times]          <- temp3[[2]]
```

```
  Recall_CMI[times]              <- temp3[[3]]
  F1_CMI[times]                  <- temp3[[4]]
  AIC_CMI[times]                 <- temp3[[5]]
}
######## The end of factor regression based on the CMI method #########
#####################################################################
######## Factor regression based on the MBI method #########
Accuracy_MBI<-Precision_MBI<-Recall_MBI<-F1_MBI<-AIC_MBI<-array()
x.1.score.hat<-x.2.score.hat<-f.hat_MBI<-vector('list',Iter.times)
source("Imputing_data_MBI.R")
for (times in 1:Iter.times){
  temp1 <- Imputing_data_MBI(NN, y[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]])
  x.1.score.hat[[times]]      <- temp1[[1]]
  x.2.score.hat[[times]]      <- temp1[[2]]
}
source("Construct_factor_CCA.R")
for (times in 1:Iter.times){
  temp2    <-    Construct_factor_CCA(x_std[[times]],    psi.1.hat[[times]],    psi.2.hat[[times]],    x.1.score.hat[[times]],
x.2.score.hat[[times]])
  f.hat_MBI[[times]]           <- temp2[[1]]
}
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_CCA(f.hat_MBI[[times]][,1:M0], y[[times]])
  Accuracy_MBI[times]          <- temp3[[1]]
  Precision_MBI[times]         <- temp3[[2]]
  Recall_MBI[times]            <- temp3[[3]]
  F1_MBI[times]                <- temp3[[4]]
  AIC_MBI[times]               <- temp3[[5]]
}
######## The end of factor regression based on the MBI method #########
#result----
result <- round(rbind(
  c(mean(Accuracy_CMI),sd(Accuracy_CMI),mean(Precision_CMI),mean(Recall_CMI),mean(na.omit(F1_CMI))),
  c(mean(Accuracy_MBI),sd(Accuracy_MBI),mean(Precision_MBI),mean(Recall_MBI),mean(na.omit(F1_MBI)))),4)
print(result)
stargazer(result, title = "Evaluation",   align = F, type = "latex")
```

###Module 2: setting_1_gamma(PCA) ###

```
library(stargazer)
library(funData)
library(caret)
library(MASS)
library(nnet)
N<-300       ## The sample size
M1<-25       ## The number of PC scores of x(1)
T1<-200      ## The number of sample point of curve x(1)
M2<-25       ## The number of PC scores of x(2)
T2<-200      ## The number of sample point of curve x(2)
k1<-10       ## The number of truncated PC scores of x(1) in Ma's paper
k2<-10       ## The number of truncated PC scores of x(2) in Ma's paper
## Regression coefficient vector
alpha1<-rep(0,10)
alpha2<-2*c(0.972,0.734,0.691,0.541,0.480,0.424,0.331,0.271,0.123,0.0405)
```

```
alpha3<-4*c(0.934,0.903,0.815,0.604,0.517,0.447,0.392,0.370,0.345,0.3)
M<-length(alpha1)               ## The number of mulrivariate truncated PC scores
sd<-0.2                         ## The standard deviation of error in the regression model
gamma<-0.3                        ## The coefficient in generating MFPC curves
## miss_ratio<-0 ##
gam1.1 = rep(1, (T1+T2))
gam1.2 = gam1.3 = rep(0, (T1+T2))
gam2.1 = rep(1, (T1+T2))
gam2.2 = gam2.3 = rep(0, (T1+T2))
gam3.1 = rep(1, (T1+T2))
gam3.2 = gam3.3 = rep(0, (T1+T2))
## miss_ratio<-0.2 ##
gam1.1 = rep(1, (T1+T2))
gam1.2 = gam1.3 = rep(7/51, (T1+T2))
gam2.1 = rep(1, (T1+T2))
gam2.2 = gam2.3 = rep(1/18, (T1+T2))
gam3.1 = rep(1, (T1+T2))
gam3.2 = gam3.3 = rep(7/51, (T1+T2))
## miss_ratio<-0.6 ##
gam1.1 = rep(1, (T1+T2))
gam1.2 = gam1.3 = rep(21/23, (T1+T2))
gam2.1 = rep(1, (T1+T2))
gam2.2 = gam2.3 = rep(3/14, (T1+T2))
gam3.1 = rep(1, (T1+T2))
gam3.2 = gam3.3 = rep(21/23, (T1+T2))
## miss_ratio<-0.9 ##
gam1.1 = rep(1, (T1+T2))
gam1.2 = gam1.3 = rep(63/4, (T1+T2))
gam2.1 = rep(1, (T1+T2))
gam2.2 = gam2.3 = rep(9/22, (T1+T2))
gam3.1 = rep(1, (T1+T2))
gam3.2 = gam3.3 = rep(63/4, (T1+T2))
Iter.times<-300
setwd("C:/Users/pc/Desktop/Logistic/Setting1(MNAR)/")
#####################################################################
######    Generating the eigenfunctions of setting 1 ######
source("generate_eigenfunction_setting1.R")
temp1 <- generate_eigenfunction_setting1(N, M1, T1, M2, T2)
t.1          <- temp1[[1]]
t.2          <- temp1[[2]]
phi.1        <- temp1[[3]]
phi.2        <- temp1[[4]]
t.1h         <- temp1[[5]]
t.2h         <- temp1[[6]]
#######    The end of Generating the eigenfunctions of setting 1 ######
#####################################################################
######## generating data ####################
psi.1_std<-psi.2_std<-rho_std<-rho_std_com<-y<-x_std<-vector('list',Iter.times)
source("generate_data_setting1_gamma.R")
for (times in 1:Iter.times){
  temp1 <- generate_data_setting1_gamma(phi.1, phi.2, alpha1, alpha2, alpha3, sd, gamma)
  rho_std[[times]]       <- temp1[[1]]
  x_std[[times]]         <- temp1[[2]]
```

```r
  y[[times]]                  <- temp1[[3]]
  psi.1_std[[times]]    <- temp1[[4]]
  psi.2_std[[times]]    <- temp1[[5]]
}
######## The end of generating data ########
####################################################################
######## generating missing pattern ####################
source("generate_missing_pattern.R")
for (times in 1:Iter.times){
  temp2 <- generate_missing_pattern(rho_std[[times]], x_std[[times]], y[[times]])
  NN                              <- temp2[[1]]
  rho_std[[times]]       <- temp2[[2]]
  x_std[[times]]           <- temp2[[3]]
  y[[times]]                  <- temp2[[4]]
}
######## The end of generating missing pattern ########
M0=10
####################################################################
######### missing rate = 0 ####################
source("MFPCA.R")
for (times in 1:Iter.times){
  temp2 <- MFPCA(x_std[[times]], k1, k2, t.1h, t.2h)
  rho_std_com[[times]]      <- temp2[[1]]
}
Accuracy_com<-Precision_com<-Recall_com<-F1_com<-AIC_com<-array()
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
  temp5 <- Factor_regression_Imputed_PCA(rho_std_com[[times]], y[[times]][1:NN[1]])
  Accuracy_com[times]            <- temp5[[1]]
  Precision_com[times]          <- temp5[[2]]
  Recall_com[times]               <- temp5[[3]]
  F1_com[times]                      <- temp5[[4]]
  AIC_com[times]                    <- temp5[[5]]
}
result_com <- round(c(mean(Accuracy_com),sd(Accuracy_com),
                                 mean(Precision_com),mean(Recall_com),mean(F1_com)),4)
print(result_com)
stargazer(result_com, title = "Evaluation",    align = F, type = "latex")
######### The end of missing rate = 0 ####################
M0=10
####################################################################
######### MFPCA_complete_data #########
psi.1.hat<-psi.2.hat<-x.1.score_NA<-x.2.score_NA<-rho.hat_std_com<-vector('list',Iter.times )
source("MFPCA_complete_data.R")
for (times in 1:Iter.times){
  same<-MFPCA_complete_data(NN, N, T1, T2, x_std[[times]], t.1h, t.2h)
  psi.1.hat[[times]]               <- same[[1]]
  psi.2.hat[[times]]               <- same[[2]]
  x.1.score_NA[[times]]          <- same[[3]]
  x.2.score_NA[[times]]          <- same[[4]]
  rho.hat_std_com[[times]]     <- same[[5]]
}
######### The end of MFPCA_complete_data #########
```

```
##################################################################
######## Factor regression based on the CMI method #########
Accuracy_CMI<-Precision_CMI<-Recall_CMI<-F1_CMI<-AIC_CMI<-array()
x.1.score.hat<-x.2.score.hat<-rho.hat_std_CMI<-vector('list',Iter.times)
source("Imputing_data_CMI.R")
for (times in 1:Iter.times){
  temp1 <- Imputing_data_CMI(NN, y[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]])
  x.1.score.hat[[times]]       <- temp1[[1]]
  x.2.score.hat[[times]]       <- temp1[[2]]
}
source("Construct_factor_PCA.R")
for (times in 1:Iter.times){
  temp2     <-     Construct_factor_PCA(x_std[[times]],     psi.1.hat[[times]],     psi.2.hat[[times]],     x.1.score.hat[[times]],
x.2.score.hat[[times]])
  rho.hat_std_CMI[[times]]     <- temp2[[1]]
}
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_PCA(rho.hat_std_CMI[[times]], y[[times]])
  Accuracy_CMI[times]             <- temp3[[1]]
  Precision_CMI[times]            <- temp3[[2]]
  Recall_CMI[times]               <- temp3[[3]]
  F1_CMI[times]                   <- temp3[[4]]
  AIC_CMI[times]                  <- temp3[[5]]
}
######## The end of factor regression based on the CMI method #########
##################################################################
######## Factor regression based on the MBI method #########
Accuracy_MBI<-Precision_MBI<-Recall_MBI<-F1_MBI<-AIC_MBI<-array()
x.1.score.hat<-x.2.score.hat<-rho.hat_std_MBI<-vector('list',Iter.times)
source("Imputing_data_MBI.R")
for (times in 1:Iter.times){
  temp1 <- Imputing_data_MBI(NN, y[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]])
  x.1.score.hat[[times]]        <- temp1[[1]]
  x.2.score.hat[[times]]        <- temp1[[2]]
}
source("Construct_factor_PCA.R")
for (times in 1:Iter.times){
  temp2     <-     Construct_factor_PCA(x_std[[times]],     psi.1.hat[[times]],     psi.2.hat[[times]],     x.1.score.hat[[times]],
x.2.score.hat[[times]])
  rho.hat_std_MBI[[times]]     <- temp2[[1]]
}
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_PCA(rho.hat_std_MBI[[times]], y[[times]])
  Accuracy_MBI[times]             <- temp3[[1]]
  Precision_MBI[times]            <- temp3[[2]]
  Recall_MBI[times]               <- temp3[[3]]
  F1_MBI[times]                   <- temp3[[4]]
  AIC_MBI[times]                  <- temp3[[5]]
}
######## The end of factor regression based on the MBI method #########
#result----
```

```
result <- round(rbind(
   c(mean(Accuracy_CMI),sd(Accuracy_CMI),mean(Precision_CMI),mean(Recall_CMI),mean(na.omit(F1_CMI))),
c(mean(Accuracy_MBI),sd(Accuracy_MBI),mean(Precision_MBI),mean(Recall_MBI),mean(na.omit(F1_MBI)))),4)
print(result)
stargazer(result, title = "Evaluation",   align = F, type = "latex")
### Module 3: setting_1_rho(CCA) ###
library(stargazer)
library(funData)
library(caret)
library(MASS)
library(nnet)
N<-300        ## The sample size
M1<-25        ## The number of PC scores of x(1)
T1<-200       ## The number of sample point of curve x(1)
M2<-25        ## The number of PC scores of x(2)
T2<-300       ## The number of sample point of curve x(2)
k1<-10        ## The number of truncated PC scores of x(1) in Ma's paper
k2<-10        ## The number of truncated PC scores of x(2) in Ma's paper
## Regression coefficient vector
alpha1<-rep(0,10)
alpha2<-2*c(0.972,0.734,0.691,0.541,0.480,0.424,0.331,0.271,0.123,0.0405)
alpha3<-4*c(0.934,0.903,0.815,0.604,0.517,0.447,0.392,0.370,0.345,0.3)
M<-length(alpha1)              ## The number of mulrivariate truncated PC scores
sd<-0.2                        ## The standard deviation of error in the regression model
rho_0<-0.4                     ## The correlation among data sources
## miss_ratio<-0 ##
gam1.1 = rep(1, (T1+T2))
gam1.2 = gam1.3 = rep(0, (T1+T2))
gam2.1 = rep(1, (T1+T2))
gam2.2 = gam2.3 = rep(0, (T1+T2))
gam3.1 = rep(1, (T1+T2))
gam3.2 = gam3.3 = rep(0, (T1+T2))
## miss_ratio<-0.2 ##
gam1.1 = rep(1, (T1+T2))
gam1.2 = gam1.3 = rep(7/51, (T1+T2))
gam2.1 = rep(1, (T1+T2))
gam2.2 = gam2.3 = rep(1/18, (T1+T2))
gam3.1 = rep(1, (T1+T2))
gam3.2 = gam3.3 = rep(7/51, (T1+T2))
## miss_ratio<-0.6 ##
gam1.1 = rep(1, (T1+T2))
gam1.2 = gam1.3 = rep(21/23, (T1+T2))
gam2.1 = rep(1, (T1+T2))
gam2.2 = gam2.3 = rep(3/14, (T1+T2))
gam3.1 = rep(1, (T1+T2))
gam3.2 = gam3.3 = rep(21/23, (T1+T2))
## miss_ratio<-0.9 ##
gam1.1 = rep(1, (T1+T2))
gam1.2 = gam1.3 = rep(63/4, (T1+T2))
gam2.1 = rep(1, (T1+T2))
gam2.2 = gam2.3 = rep(9/22, (T1+T2))
gam3.1 = rep(1, (T1+T2))
gam3.2 = gam3.3 = rep(63/4, (T1+T2))
```

```
Iter.times<-300
setwd("C:/Users/pc/Desktop/Logistic/Setting1(MNAR)/")
##################################################################
######    Generating the eigenfunctions of setting 1 ######
source("generate_eigenfunction_setting1.R")
temp1 <- generate_eigenfunction_setting1(N, M1, T1, M2, T2)
t.1           <- temp1[[1]]
t.2           <- temp1[[2]]
phi.1         <- temp1[[3]]
phi.2         <- temp1[[4]]
t.1h          <- temp1[[5]]
t.2h          <- temp1[[6]]
#######    The end of Generating the eigenfunctions of setting 1 ######
##################################################################
######## generating data ####################
psi.1_std<-psi.2_std<-rho_std<-f.hat_com<-y<-x_std<-vector('list',Iter.times)
source("generate_data_setting1_rho.R")
for (times in 1:Iter.times) {
   temp1 <- generate_data_setting1_rho(rho_0, phi.1, phi.2, alpha1, alpha2, alpha3, sd)
   rho_std[[times]]       <- temp1[[1]]
   x_std[[times]]         <- temp1[[2]]
   y[[times]]             <- temp1[[3]]
   psi.1_std[[times]]    <- temp1[[4]]
   psi.2_std[[times]]    <- temp1[[5]]
}
######## The end of generating data ########
##################################################################
######## generating missing pattern ####################
source("generate_missing_pattern.R")
for (times in 1:Iter.times){
   temp2 <- generate_missing_pattern(rho_std[[times]], x_std[[times]], y[[times]])
   NN                              <- temp2[[1]]
   rho_std[[times]]       <- temp2[[2]]
   x_std[[times]]         <- temp2[[3]]
   y[[times]]             <- temp2[[4]]
}
######## The end of generating missing pattern ########
M0=10
##################################################################
######### missing rate = 0 ####################
source("MCCA.R")
for (times in 1:Iter.times){
   temp2 <- MCCA(x_std[[times]], k1, k2, t.1h, t.2h)
   f.hat_com[[times]]      <- temp2[[1]]
}
Accuracy_com<-Precision_com<-Recall_com<-F1_com<-AIC_com<-array()
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
   temp5 <- Factor_regression_Imputed_CCA(f.hat_com[[times]][1:NN[1],1:M0], y[[times]][1:NN[1]])
   Accuracy_com[times]          <- temp5[[1]]
   Precision_com[times]         <- temp5[[2]]
   Recall_com[times]            <- temp5[[3]]
   F1_com[times]                <- temp5[[4]]
```

```
  AIC_com[times]                    <- temp5[[5]]
}
result_com <- round(c(mean(Accuracy_com),sd(Accuracy_com),
mean(Precision_com),mean(Recall_com),mean(na.omit(F1_com))),4)
print(result_com)
stargazer(result_com, title = "Evaluation",    align = F, type = "latex")
######### The end of missing rate = 0 ###################
M0=10
##################################################################
######### MCCA_complete_data #########
psi.1.hat<-psi.2.hat<-x.1.score_NA<-x.2.score_NA<-f.hat_com<-vector('list',Iter.times )
source("MCCA_complete_data.R")
for (times in 1:Iter.times){
  same<-MCCA_complete_data(NN, N, T1, T2, x_std[[times]],t.1h,t.2h)
  psi.1.hat[[times]]              <- same[[1]]
  psi.2.hat[[times]]              <- same[[2]]
  x.1.score_NA[[times]]           <- same[[3]]
  x.2.score_NA[[times]]           <- same[[4]]
  f.hat_com[[times]]              <- same[[5]]
}
######### The end of MFPCA_complete_data #########
##################################################################
######## Factor regression based on the CMI method #########
Accuracy_CMI<-Precision_CMI<-Recall_CMI<-F1_CMI<-AIC_CMI<-array()
x.1.score.hat<-x.2.score.hat<-f.hat_CMI<-vector('list',Iter.times)
source("Imputing_data_CMI.R")
for (times in 1:Iter.times){
  temp1 <- Imputing_data_CMI(NN, y[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]])
  x.1.score.hat[[times]]       <- temp1[[1]]
  x.2.score.hat[[times]]       <- temp1[[2]]
}
source("Construct_factor_CCA.R")
for (times in 1:Iter.times){
  temp2    <-    Construct_factor_CCA(x_std[[times]],    psi.1.hat[[times]],    psi.2.hat[[times]],    x.1.score.hat[[times]],
x.2.score.hat[[times]])
  f.hat_CMI[[times]]              <- temp2[[1]]
}
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_CCA(f.hat_CMI[[times]][,1:M0], y[[times]])
  Accuracy_CMI[times]             <- temp3[[1]]
  Precision_CMI[times]           <- temp3[[2]]
  Recall_CMI[times]                <- temp3[[3]]
  F1_CMI[times]                     <- temp3[[4]]
  AIC_CMI[times]                    <- temp3[[5]]
}
######## The end of factor regression based on the CMI method #########
##################################################################
######## Factor regression based on the MBI method #########
Accuracy_MBI<-Precision_MBI<-Recall_MBI<-F1_MBI<-AIC_MBI<-array()
x.1.score.hat<-x.2.score.hat<-f.hat_MBI<-vector('list',Iter.times)
source("Imputing_data_MBI.R")
for (times in 1:Iter.times){
```

```
  temp1 <- Imputing_data_MBI(NN, y[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]])
  x.1.score.hat[[times]]      <- temp1[[1]]
  x.2.score.hat[[times]]      <- temp1[[2]]
}
source("Construct_factor_CCA.R")
for (times in 1:Iter.times){
  temp2    <-    Construct_factor_CCA(x_std[[times]],    psi.1.hat[[times]],    psi.2.hat[[times]],    x.1.score.hat[[times]],
x.2.score.hat[[times]])
  f.hat_MBI[[times]]           <- temp2[[1]]
}
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_CCA(f.hat_MBI[[times]][,1:M0], y[[times]])
  Accuracy_MBI[times]          <- temp3[[1]]
  Precision_MBI[times]         <- temp3[[2]]
  Recall_MBI[times]            <- temp3[[3]]
  F1_MBI[times]                <- temp3[[4]]
  AIC_MBI[times]               <- temp3[[5]]
}
######## The end of factor regression based on the MBI method #########
#result----
result <- round(rbind(
c(mean(Accuracy_CMI),sd(Accuracy_CMI),mean(Precision_CMI),mean(Recall_CMI),mean(na.omit(F1_CMI))),
c(mean(Accuracy_MBI),sd(Accuracy_MBI),mean(Precision_MBI),mean(Recall_MBI),mean(na.omit(F1_MBI)))),4)
print(result)
stargazer(result, title = "Evaluation",   align = F, type = "latex")
### Module 4: setting_1_rho(PCA) ###
library(stargazer)
library(funData)
library(caret)
library(MASS)
library(nnet)
N<-300       ## The sample size
M1<-25       ## The number of PC scores of x(1)
T1<-200      ## The number of sample point of curve x(1)
M2<-25       ## The number of PC scores of x(2)
T2<-300      ## The number of sample point of curve x(2)
k1<-10       ## The number of truncated PC scores of x(1) in Ma's paper
k2<-10       ## The number of truncated PC scores of x(2) in Ma's paper
## Regression coefficient vector
alpha1<-rep(0,10)
alpha2<-2*c(0.972,0.734,0.691,0.541,0.480,0.424,0.331,0.271,0.123,0.0405)
alpha3<-4*c(0.934,0.903,0.815,0.604,0.517,0.447,0.392,0.370,0.345,0.3)
M<-length(alpha1)              ## The number of mulrivariate truncated PC scores
sd<-0.2                        ## The standard deviation of error in the regression model
rho_0<-0.4                     ## The correlation among data sources
## miss_ratio<-0 ##
gam1.1 = rep(1, (T1+T2))
gam1.2 = gam1.3 = rep(0, (T1+T2))
gam2.1 = rep(1, (T1+T2))
gam2.2 = gam2.3 = rep(0, (T1+T2))
gam3.1 = rep(1, (T1+T2))
gam3.2 = gam3.3 = rep(0, (T1+T2))
```

```
## miss_ratio<-0.2 ##
gam1.1 = rep(1, (T1+T2))
gam1.2 = gam1.3 = rep(7/51, (T1+T2))
gam2.1 = rep(1, (T1+T2))
gam2.2 = gam2.3 = rep(1/18, (T1+T2))
gam3.1 = rep(1, (T1+T2))
gam3.2 = gam3.3 = rep(7/51, (T1+T2))
## miss_ratio<-0.6 ##
gam1.1 = rep(1, (T1+T2))
gam1.2 = gam1.3 = rep(21/23, (T1+T2))
gam2.1 = rep(1, (T1+T2))
gam2.2 = gam2.3 = rep(3/14, (T1+T2))
gam3.1 = rep(1, (T1+T2))
gam3.2 = gam3.3 = rep(21/23, (T1+T2))
## miss_ratio<-0.9 ##
gam1.1 = rep(1, (T1+T2))
gam1.2 = gam1.3 = rep(63/4, (T1+T2))
gam2.1 = rep(1, (T1+T2))
gam2.2 = gam2.3 = rep(9/22, (T1+T2))
gam3.1 = rep(1, (T1+T2))
gam3.2 = gam3.3 = rep(63/4, (T1+T2))
Iter.times<-300
setwd("C:/Users/pc/Desktop/Logistic/Setting1(MNAR)/")
####################################################################
######   Generating the eigenfunctions of setting 1 ######
source("generate_eigenfunction_setting1.R")
temp1 <- generate_eigenfunction_setting1(N, M1, T1, M2, T2)
t.1         <- temp1[[1]]
t.2         <- temp1[[2]]
phi.1       <- temp1[[3]]
phi.2       <- temp1[[4]]
t.1h        <- temp1[[5]]
t.2h        <- temp1[[6]]
#######   The end of Generating the eigenfunctions of setting 1 ######
####################################################################
######## generating data ####################
psi.1_std<-psi.2_std<-rho_std<-rho_std_com<-y<-x_std<-vector('list',Iter.times)
source("generate_data_setting1_rho.R")
for (times in 1:Iter.times) {
  temp1 <- generate_data_setting1_rho(rho_0, phi.1, phi.2, alpha1, alpha2, alpha3, sd)
  rho_std[[times]]      <- temp1[[1]]
  x_std[[times]]        <- temp1[[2]]
  y[[times]]            <- temp1[[3]]
  psi.1_std[[times]]    <- temp1[[4]]
  psi.2_std[[times]]    <- temp1[[5]]
}
######## The end of generating data ########
####################################################################
######## generating missing pattern ###################
source("generate_missing_pattern.R")
for (times in 1:Iter.times){
  temp2 <- generate_missing_pattern(rho_std[[times]], x_std[[times]], y[[times]])
  NN                         <- temp2[[1]]
```

```
    rho_std[[times]]       <- temp2[[2]]
    x_std[[times]]          <- temp2[[3]]
    y[[times]]                  <- temp2[[4]]
}
######## The end of generating missing pattern ########
M0=10
####################################################################
######### missing rate = 0 ####################
source("MFPCA.R")
for (times in 1:Iter.times){
    temp2 <- MFPCA(x_std[[times]], k1, k2, t.1h, t.2h)
    rho_std_com[[times]]     <- temp2[[1]]
}
Accuracy_com<-Precision_com<-Recall_com<-F1_com<-AIC_com<-array()
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
    temp5 <- Factor_regression_Imputed_PCA(rho_std_com[[times]], y[[times]][1:NN[1]])
    Accuracy_com[times]            <- temp5[[1]]
    Precision_com[times]          <- temp5[[2]]
    Recall_com[times]              <- temp5[[3]]
    F1_com[times]                      <- temp5[[4]]
    AIC_com[times]                    <- temp5[[5]]
}
result_com <- round(c(mean(Accuracy_com),sd(Accuracy_com),
                                    mean(Precision_com),mean(Recall_com),mean(F1_com)),4)
print(result_com)
stargazer(result_com, title = "Evaluation",    align = F, type = "latex")
######### The end of missing rate = 0 ####################
M0=10
####################################################################
######### MFPCA_complete_data #########
psi.1.hat<-psi.2.hat<-x.1.score_NA<-x.2.score_NA<-rho.hat_std_com<-vector('list',Iter.times )
source("MFPCA_complete_data.R")
for (times in 1:Iter.times){
    same<-MFPCA_complete_data(NN, N, T1, T2, x_std[[times]], t.1h, t.2h)
    psi.1.hat[[times]]               <- same[[1]]
    psi.2.hat[[times]]               <- same[[2]]
    x.1.score_NA[[times]]          <- same[[3]]
    x.2.score_NA[[times]]          <- same[[4]]
    rho.hat_std_com[[times]]     <- same[[5]]
}
######### The end of MFPCA_complete_data #########
####################################################################
######## Factor regression based on the CMI method #########
Accuracy_CMI<-Precision_CMI<-Recall_CMI<-F1_CMI<-AIC_CMI<-array()
x.1.score.hat<-x.2.score.hat<-rho.hat_std_CMI<-vector('list',Iter.times)
source("Imputing_data_CMI.R")
for (times in 1:Iter.times){
    temp1 <- Imputing_data_CMI(NN, y[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]])
    x.1.score.hat[[times]]       <- temp1[[1]]
    x.2.score.hat[[times]]       <- temp1[[2]]
}
source("Construct_factor_PCA.R")
```

```
for (times in 1:Iter.times){
  temp2 <- Construct_factor_PCA(x_std[[times]], psi.1.hat[[times]], psi.2.hat[[times]], x.1.score.hat[[times]],
x.2.score.hat[[times]])
  rho.hat_std_CMI[[times]]    <- temp2[[1]]
}
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_PCA(rho.hat_std_CMI[[times]], y[[times]])
  Accuracy_CMI[times]          <- temp3[[1]]
  Precision_CMI[times]         <- temp3[[2]]
  Recall_CMI[times]            <- temp3[[3]]
  F1_CMI[times]                <- temp3[[4]]
  AIC_CMI[times]               <- temp3[[5]]
}
######## The end of factor regression based on the CMI method #########
####################################################################
######## Factor regression based on the MBI method #########
Accuracy_MBI<-Precision_MBI<-Recall_MBI<-F1_MBI<-AIC_MBI<-array()
x.1.score.hat<-x.2.score.hat<-rho.hat_std_MBI<-vector('list',Iter.times)
source("Imputing_data_MBI.R")
for (times in 1:Iter.times){
  temp1 <- Imputing_data_MBI(NN, y[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]])
  x.1.score.hat[[times]]     <- temp1[[1]]
  x.2.score.hat[[times]]     <- temp1[[2]]
}
source("Construct_factor_PCA.R")
for (times in 1:Iter.times){
  temp2 <- Construct_factor_PCA(x_std[[times]], psi.1.hat[[times]], psi.2.hat[[times]], x.1.score.hat[[times]],
x.2.score.hat[[times]])
  rho.hat_std_MBI[[times]]    <- temp2[[1]]
}
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_PCA(rho.hat_std_MBI[[times]], y[[times]])
  Accuracy_MBI[times]          <- temp3[[1]]
  Precision_MBI[times]         <- temp3[[2]]
  Recall_MBI[times]            <- temp3[[3]]
  F1_MBI[times]                <- temp3[[4]]
  AIC_MBI[times]               <- temp3[[5]]
}
######## The end of factor regression based on the MBI method #########
#result----
result <- round(rbind(
c(mean(Accuracy_CMI),sd(Accuracy_CMI),mean(Precision_CMI),mean(Recall_CMI),mean(na.omit(F1_CMI))),
c(mean(Accuracy_MBI),sd(Accuracy_MBI),mean(Precision_MBI),mean(Recall_MBI),mean(na.omit(F1_MBI)))),4)
print(result)
stargazer(result, title = "Evaluation",   align = F, type = "latex")
```

Functions: the following is the function "generate_missing_pattern" used in the main program. The rest of the functions used are the same as in Setting1(MCAR).

The function "generate_missing_pattern" means to generate data according to the missing data mechanism of MNAR.

```
generate_missing_pattern<-function(rho, x, y){
  id1 = which(y == 1)
```

```
id2 = which(y == 2)
id3 = which(y == 3)
x.1 = x[,1:T1]
x.2 = x[,(T1+1):(T1+T2)]
## Divide the missing block when y = 1 ##
g1.1 = x.1[id1,]%*%gam1.1[1:T1]/T1 + x.2[id1,]%*%gam1.1[(T1+1):(T1+T2)]/T2
g1.2 = x.1[id1,]%*%gam1.2[1:T1]/T1 + x.2[id1,]%*%gam1.2[(T1+1):(T1+T2)]/T2
g1.3 = x.1[id1,]%*%gam1.3[1:T1]/T1 + x.2[id1,]%*%gam1.3[(T1+1):(T1+T2)]/T2
p1.1 = g1.1 / (g1.1+g1.2+g1.3)
p1.2 = g1.2 / (g1.1+g1.2+g1.3)
p1.3 = g1.3 / (g1.1+g1.2+g1.3)
p1 = cbind(p1.1, p1.2, p1.3)
miss.1 = c()
for(i in 1:length(id1)){
  miss0.1 = which.max(rmultinom(1, size = 1, prob = p1[i,]))
  miss.1 = c(miss.1, miss0.1)
}
id1.1 = which(miss.1 == 1)
id1.2 = which(miss.1 == 2)
id1.3 = which(miss.1 == 3)
## The end of dividing the missing block when y = 1 ##
## Divide the missing block when y = 2 ##
g2.1 = x.1[id2,]%*%gam2.1[1:T1]/T1 + x.2[id2,]%*%gam2.1[(T1+1):(T1+T2)]/T2
g2.2 = x.1[id2,]%*%gam2.2[1:T1]/T1 + x.2[id2,]%*%gam2.2[(T1+1):(T1+T2)]/T2
g2.3 = x.1[id2,]%*%gam2.3[1:T1]/T1 + x.2[id2,]%*%gam2.3[(T1+1):(T1+T2)]/T2
p2.1 = g2.1 / (g2.1+g2.2+g2.3)
p2.2 = g2.2 / (g2.1+g2.2+g2.3)
p2.3 = g2.3 / (g2.1+g2.2+g2.3)
p2 = cbind(p2.1, p2.2, p2.3)
miss.2 = c()
for(i in 1:length(id2)){
  miss0.2 = which.max(rmultinom(1, size = 1, prob = p2[i,]))
  miss.2 = c(miss.2, miss0.2)
}
id2.1 = which(miss.2 == 1)
id2.2 = which(miss.2 == 2)
id2.3 = which(miss.2 == 3)
## The end of dividing the missing block when y = 2 ##
## Divide the missing block when y = 3 ##
g3.1 = x.1[id3,]%*%gam3.1[1:T1]/T1 + x.2[id3,]%*%gam3.1[(T1+1):(T1+T2)]/T2
g3.2 = x.1[id3,]%*%gam3.2[1:T1]/T1 + x.2[id3,]%*%gam3.2[(T1+1):(T1+T2)]/T2
g3.3 = x.1[id3,]%*%gam3.3[1:T1]/T1 + x.2[id3,]%*%gam3.3[(T1+1):(T1+T2)]/T2
p3.1 = g3.1 / (g3.1+g3.2+g3.3)
p3.2 = g3.2 / (g3.1+g3.2+g3.3)
p3.3 = g3.3 / (g3.1+g3.2+g3.3)
p3 = cbind(p3.1, p3.2, p3.3)
miss.3 = c()
for(i in 1:length(id3)){
  miss0.3 = which.max(rmultinom(1, size = 1, prob = p3[i,]))
  miss.3 = c(miss.3, miss0.3)
}
id3.1 = which(miss.3 == 1)
id3.2 = which(miss.3 == 2)
```

```
    id3.3 = which(miss.3 == 3)
    ## The end of dividing the missing block when y = 3 ##
    ## Divide the missing block ##
    id.1 = c(id1[id1.1], id2[id2.1], id3[id3.1])
    id.2 = c(id1[id1.2], id2[id2.2], id3[id3.2])
    id.3 = c(id1[id1.3], id2[id2.3], id3[id3.3])
    NN = c()
    NN[1] = length(id.1)
    NN[2] = length(id.2)
    NN[3] = length(id.3)
    y.new = c(y[id.1], y[id.2], y[id.3])
    x.new = rbind(x[id.1,], x[id.2,], x[id.3,])
    rho.new = rbind(rho[id.1,], rho[id.2,], rho[id.3,])
    ## The end of dividing the missing block ##

    return(list(NN, rho.new, x.new, y.new))
}
```

3. Setting 2(MCAR)

Under the MCAR missing mechanism, we respectively consider the calculation of multi-source functional principal component score and canonical score of 3-source data (image, curve, curve), as well as the corresponding classification evaluation index.

We have two main modules as follows:

"setting_2(CCA)" represents the main program using the FR-CCA method in Setting 2;

"setting_2(PCA)" represents the main program using the FR-PCA method in Setting 2.

The functions used in the main program are all listed after the four main programs.

Module 1: setting_2(CCA)

```
library(stargazer)
library(funData)
library(caret)
library(MASS)
library(nnet)
N<-300             ## The sample size
T11<-100           ## Image x(1): the sample number of x-axis
T12<-50            ## Image x(1): the sample number of y-axis
T1<-T11*T12        ## The number of sample point of Image x(1)
M11<-5             ## The number of PC scores of x(1) along x-axis
M12<-5             ## The number of PC scores of x(1) along y-axis
M1<-M11*M12          ## The number of PC scores of x(1)
T2<-200            ## The number of sample point of Image x(2)
M2<-25             ## The number of PC scores of x(2)
T3<-200            ## The number of sample point of Image x(3)
M3<-25             ## The number of PC scores of x(3)
k1=10              ## The number of truncated PC scores of x(1) in Ma's paper
k2=10              ## The number of truncated PC scores of x(2) in Ma's paper
k3=10              ## The number of truncated PC scores of x(3) in Ma's paper
gamma1<-0.3         ## The coefficient in generating MFPC curves
gamma2<-0.3
## Regression coefficient vector
alpha1<-rep(0,10)
alpha2<-2*c(0.972,0.734,0.691,0.541,0.480,0.424,0.331,0.271,0.123,0.0405)
alpha3<-4*c(0.934,0.903,0.815,0.604,0.517,0.447,0.392,0.370,0.345,0.3)
M<-length(alpha1)          ## The number of mulrivariate truncated PC scores in Happ's paper
sd<-0.2                     ## The standard deviation of error in the regression model
```

```
miss_ratio<-0                    ## Missing ratio .2 .6 .9
NN<-array()
NN[1]<-N*(1-miss_ratio) ## The number of complete-data
NN[2:7]<-N*miss_ratio/6 ## The number of missing subjects
Iter.times<-300
setwd("C:/Users/pc/Desktop/Logistic/Setting2(MCAR)/")
#################################################################
######    Generating the eigenfunctions of setting 3 ######
source("generate_eigenfunction_setting2.R")
temp1 <- generate_eigenfunction_setting2(N, M11, T11, M12, T12, M2, T2, M3, T3)
t.11                  <- temp1[[1]]
t.12                  <- temp1[[2]]
t.2                   <- temp1[[3]]
t.3                   <- temp1[[4]]
phi.1                 <- temp1[[5]]
phi.2                 <- temp1[[6]]
phi.3                 <- temp1[[7]]
t.1h                  <- temp1[[8]]
t.2h                  <- temp1[[9]]
t.3h                  <- temp1[[10]]
#######    The end of Generating the eigenfunctions of setting 3 ######
#################################################################
######## generating data ####################
psi.1_std<-psi.2_std<-psi.3_std<-rho_std<-f.hat_com<-y<-x_std<-vector('list',Iter.times)
source("generate_data_setting2.R")
for (times in 1:Iter.times) {
   temp1 <- generate_data_setting2(phi.1, phi.2, phi.3, alpha1, alpha2, alpha3, sd, gamma1, gamma2)
   rho_std[[times]]                <- temp1[[1]]
   x_std[[times]]          <- temp1[[2]]
   y[[times]]                 <- temp1[[3]]
   psi.1_std[[times]]    <- temp1[[4]]
   psi.2_std[[times]]    <- temp1[[5]]
   psi.3_std[[times]]    <- temp1[[6]]
}
######## The end of generating data ########
M0=10
#################################################################
######### missing rate = 0 ###################
source("MCCA_setting2.R")
for (times in 1:Iter.times){
   temp2 <- MCCA_setting2(x_std[[times]], k1, k2, k3, t.1h, t.2h, t.3h)
   f.hat_com[[times]]      <- temp2[[1]]
}
Accuracy_com<-Precision_com<-Recall_com<-F1_com<-AIC_com<-array()
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
   temp5 <- Factor_regression_Imputed_CCA(f.hat_com[[times]][1:NN[1],1:M0], alpha2, alpha3, y[[times]][1:NN[1]])
   Accuracy_com[times]           <- temp5[[1]]
   Precision_com[times]          <- temp5[[2]]
   Recall_com[times]             <- temp5[[3]]
   F1_com[times]                    <- temp5[[4]]
   AIC_com[times]                  <- temp5[[5]]
}
```

```
result_com <- round(c(mean(Accuracy_com),sd(Accuracy_com),
                      mean(Precision_com),mean(Recall_com),mean(F1_com)),6)
print(result_com)
stargazer(result_com, title = "Evaluation",   align = F, type = "latex")
######### The end of missing rate = 0 ###################
# NN=c(30,45,45,45,45,45,45) # N=300,missing rate=0.9
# NN=c(60,90,90,90,90,90,90) # N=600,missing rate=0.9
M0=10
##################################################################
######### MCCA_complete_data #########
psi.1.hat<-psi.2.hat<-psi.3.hat<-
  x.1.score<-x.2.score<-x.3.score<-
  x.1.score_NA<-x.2.score_NA<-x.3.score_NA<-
  f.hat_com<-vector('list',Iter.times )
source("MCCA_complete_data_setting2.R")
for (times in 1:Iter.times){
  same<-MCCA_complete_data_setting2(NN, N, T1, T2, T3, x_std[[times]], t.1h, t.2h, t.3h)
  psi.1.hat[[times]]            <-same[[1]]
  psi.2.hat[[times]]            <-same[[2]]
  psi.3.hat[[times]]            <-same[[3]]
  x.1.score_NA[[times]]         <-same[[4]]
  x.2.score_NA[[times]]         <-same[[5]]
  x.3.score_NA[[times]]         <-same[[6]]
  x.1.score[[times]]            <-same[[7]]
  x.2.score[[times]]            <-same[[8]]
  x.3.score[[times]]            <-same[[9]]
  f.hat_com[[times]]            <-same[[10]]
}
######### The end of MCCA_complete_data #########
##################################################################
######## Factor regression based on the CMI method #########
Accuracy_CMI<-Precision_CMI<-Recall_CMI<-F1_CMI<-AIC_CMI<-array()
x.1.score.hat<-x.2.score.hat<-x.3.score.hat<-f.hat_CMI<-vector('list',Iter.times)
source("Imputing_data_CMI_setting2.R")
for (times in 1:Iter.times) {
  temp1 <- Imputing_data_CMI_setting2(NN, x_std[[times]], x.1.score[[times]], x.2.score[[times]], x.3.score[[times]])
  x.1.score.hat[[times]]      <- temp1[[1]]
  x.2.score.hat[[times]]      <- temp1[[2]]
  x.3.score.hat[[times]]      <- temp1[[3]]
}
source("Construct_factor_CCA.R")
for (times in 1:Iter.times){
  temp2 <- Construct_factor_CCA(x_std[[times]], psi.1.hat[[times]], psi.2.hat[[times]],psi.3.hat[[times]],
                                x.1.score.hat[[times]], x.2.score.hat[[times]], x.3.score.hat[[times]])
  f.hat_CMI[[times]]            <- temp2[[1]]
}
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_CCA(Re(f.hat_CMI[[times]][,1:M0]), alpha2, alpha3, y[[times]])
  Accuracy_CMI[times]           <- temp3[[1]]
  Precision_CMI[times]          <- temp3[[2]]
  Recall_CMI[times]             <- temp3[[3]]
  F1_CMI[times]                 <- temp3[[4]]
```

```
  AIC_CMI[times]                    <- temp3[[5]]
}
######## The end of factor regression based on the CMI method #########
##################################################################
######## Factor regression based on the MBI method #########
Accuracy_MBI<-Precision_MBI<-Recall_MBI<-F1_MBI<-AIC_MBI<-array()
x.1.score.hat<-x.2.score.hat<-x.3.score.hat<-f.hat_MBI<-vector('list',Iter.times)
source("Imputing_data_MBI_setting2.R")
for (times in 1:Iter.times) {
  temp1    <-    Imputing_data_MBI_setting2(NN,    x_std[[times]],    x.1.score_NA[[times]],    x.2.score_NA[[times]],
x.3.score_NA[[times]])
  x.1.score.hat[[times]]      <- temp1[[1]]
  x.2.score.hat[[times]]      <- temp1[[2]]
  x.3.score.hat[[times]]      <- temp1[[3]]
}
source("Construct_factor_CCA.R")
for (times in 1:Iter.times){
  temp2 <- Construct_factor_CCA(x_std[[times]], psi.1.hat[[times]], psi.2.hat[[times]],psi.3.hat[[times]],
                                           x.1.score.hat[[times]], x.2.score.hat[[times]], x.3.score.hat[[times]])
  f.hat_MBI[[times]]            <- temp2[[1]]
}
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_CCA(Re(f.hat_MBI[[times]][,1:M0]), alpha2, alpha3, y[[times]])
  Accuracy_MBI[times]          <- temp3[[1]]
  Precision_MBI[times]         <- temp3[[2]]
  Recall_MBI[times]            <- temp3[[3]]
  F1_MBI[times]                <- temp3[[4]]
  AIC_MBI[times]               <- temp3[[5]]
}
######## The end of factor regression based on the MBI method #########
#result----
result <- round(rbind(
  c(mean(Accuracy_CMI),sd(Accuracy_CMI),mean(Precision_CMI),mean(Recall_CMI),mean(F1_CMI)),
  c(mean(Accuracy_MBI),sd(Accuracy_MBI),mean(Precision_MBI),mean(Recall_MBI),mean(F1_MBI))),4)
print(result)
stargazer(result, title = "Evaluation",   align = F, type = "latex")
### Module 2: setting_2(PCA) ###
library(stargazer)
library(funData)
library(caret)
library(MASS)
library(nnet)
N<-300              ## The sample size
T11<-100            ## Image x(1): the sample number of x-axis
T12<-50             ## Image x(1): the sample number of y-axis
T1<-T11*T12         ## The number of sample point of Image x(1)
M11<-5              ## The number of PC scores of x(1) along x-axis
M12<-5              ## The number of PC scores of x(1) along y-axis
M1<-M11*M12         ## The number of PC scores of x(1)
T2<-200             ## The number of sample point of Image x(2)
M2<-25              ## The number of PC scores of x(2)
T3<-200             ## The number of sample point of Image x(3)
```

```
M3<-25                ## The number of PC scores of x(3)
k1=10                 ## The number of truncated PC scores of x(1) in Ma's paper
k2=10                 ## The number of truncated PC scores of x(2) in Ma's paper
k3=10                 ## The number of truncated PC scores of x(3) in Ma's paper
gamma1<-0.3       ## The coefficient in generating MFPC curves
gamma2<-0.3
## Regression coefficient vector
alpha1<-rep(0,10)
alpha2<-2*c(0.972,0.734,0.691,0.541,0.480,0.424,0.331,0.271,0.123,0.0405)
alpha3<-4*c(0.934,0.903,0.815,0.604,0.517,0.447,0.392,0.370,0.345,0.3)
M<-length(alpha1)           ## The number of mulrivariate truncated PC scores
sd <- 0.2                       ## The standard deviation of error in the regression model
miss_ratio<-0                 ## Missing ratio .2 .6 .9
NN<-array()
NN[1]<-N*(1-miss_ratio) ## The number of complete-data
NN[2:7]<-N*miss_ratio/6 ## The number of missing subjects
Iter.times<-300
setwd("C:/Users/pc/Desktop/Logistic/Setting2(MCAR)/")
####################################################################
######    Generating the eigenfunctions of setting 3 ######
source("generate_eigenfunction_setting2.R")
temp1 <- generate_eigenfunction_setting2(N, M11, T11, M12, T12, M2, T2, M3, T3)
t.11                 <- temp1[[1]]
t.12                 <- temp1[[2]]
t.2                  <- temp1[[3]]
t.3                  <- temp1[[4]]
phi.1                <- temp1[[5]]
phi.2                <- temp1[[6]]
phi.3                <- temp1[[7]]
t.1h                 <- temp1[[8]]
t.2h                 <- temp1[[9]]
t.3h                 <- temp1[[10]]
#######    The end of Generating the eigenfunctions of setting 3 ######
####################################################################
######## generating data ####################
psi.1_std<-psi.2_std<-psi.3_std<-rho_std<-rho_std_com<-y<-x_std<-vector('list',Iter.times)
source("generate_data_setting2.R")
for (times in 1:Iter.times) {
   temp1 <- generate_data_setting2(phi.1, phi.2, phi.3, alpha1, alpha2, alpha3, sd, gamma1, gamma2)
   rho_std[[times]]       <- temp1[[1]]
   x_std[[times]]         <- temp1[[2]]
   y[[times]]                <- temp1[[3]]
   psi.1_std[[times]]    <- temp1[[4]]
   psi.2_std[[times]]    <- temp1[[5]]
   psi.3_std[[times]]    <- temp1[[6]]
}
######## The end of generating data ########
M0=10
####################################################################
######### missing rate = 0 ###################
source("MFPCA_setting2.R")
for (times in 1:Iter.times){
   temp2 <- MFPCA_setting2(x_std[[times]], k1, k2, k3, t.1h, t.2h, t.3h)
```

```
  rho_std_com[[times]]     <- temp2[[1]]
}
Accuracy_com<-Precision_com<-Recall_com<-F1_com<-AIC_com<-mse_alpha2<-mse_alpha3<-array()
alpha2.hat<-alpha3.hat<-vector('list',Iter.times )
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
  temp5 <- Factor_regression_Imputed_PCA(rho_std_com[[times]], alpha2, alpha3, y[[times]][1:NN[1]])
  Accuracy_com[times]           <- temp5[[1]]
  Precision_com[times]          <- temp5[[2]]
  Recall_com[times]             <- temp5[[3]]
  F1_com[times]                 <- temp5[[4]]
  AIC_com[times]                <- temp5[[5]]
  alpha2.hat[[times]]           <- temp5[[6]]
  alpha3.hat[[times]]           <- temp5[[7]]
  mse_alpha2[times]             <- temp5[[8]]
  mse_alpha3[times]             <- temp5[[9]]
}
result_com <- round(c(mean(Accuracy_com),sd(Accuracy_com),
                      mean(Precision_com),mean(Recall_com),mean(F1_com),
                      mean(mse_alpha2),sd(mse_alpha2),
                      mean(mse_alpha3),sd(mse_alpha3)),6)
print(result_com)
stargazer(result_com, title = "Evaluation",   align = F, type = "latex")
######### The end of missing rate = 0 ####################
# NN=c(30,45,45,45,45,45,45) # N=300,missing rate=0.9
# NN=c(60,90,90,90,90,90,90) # N=600,missing rate=0.9
M0=10
#################################################################
######### MFPCA_complete_data #########
psi.1.hat<-psi.2.hat<-psi.3.hat<-
  x.1.score<-x.2.score<-x.3.score<-
  x.1.score_NA<-x.2.score_NA<-x.3.score_NA<-
  rho.hat_std_com<-vector('list',Iter.times )
source("MFPCA_complete_data_setting2.R")
for (times in 1:Iter.times){
  same<-MFPCA_complete_data_setting2(NN, N, T1, T2, T3, x_std[[times]], t.1h, t.2h, t.3h)
  psi.1.hat[[times]]              <-same[[1]]
  psi.2.hat[[times]]              <-same[[2]]
  psi.3.hat[[times]]              <-same[[3]]
  x.1.score_NA[[times]]           <-same[[4]]
  x.2.score_NA[[times]]           <-same[[5]]
  x.3.score_NA[[times]]           <-same[[6]]
  x.1.score[[times]]              <-same[[7]]
  x.2.score[[times]]              <-same[[8]]
  x.3.score[[times]]              <-same[[9]]
  rho.hat_std_com[[times]]     <-same[[10]]
}
######### The end of MFPCA_complete_data #########
#################################################################
######## Factor regression based on the CMI method #########
Accuracy_CMI<-Precision_CMI<-Recall_CMI<-F1_CMI<-AIC_CMI<-
  mse_alpha2_CMI<-mse_alpha3_CMI<-array()
x.1.score.hat<-x.2.score.hat<-x.3.score.hat<-rho.hat_std_CMI<-
```

```
  alpha2.hat_CMI<-alpha3.hat_CMI<-vector('list',Iter.times )
source("Imputing_data_CMI_setting2.R")
for (times in 1:Iter.times) {
  temp1 <- Imputing_data_CMI_setting2(NN, x_std[[times]], x.1.score[[times]], x.2.score[[times]], x.3.score[[times]])
  x.1.score.hat[[times]]      <- temp1[[1]]
  x.2.score.hat[[times]]      <- temp1[[2]]
  x.3.score.hat[[times]]      <- temp1[[3]]
}
source("Construct_factor_PCA.R")
for (times in 1:Iter.times){
  temp2 <- Construct_factor_PCA(x_std[[times]], psi.1.hat[[times]], psi.2.hat[[times]], psi.3.hat[[times]], x.1.score.hat[[times]],
x.2.score.hat[[times]], x.3.score.hat[[times]])
  rho.hat_std_CMI[[times]]     <- temp2[[1]]
}
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_PCA(rho.hat_std_CMI[[times]], alpha2, alpha3, y[[times]])
  Accuracy_CMI[times]           <- temp3[[1]]
  Precision_CMI[times]          <- temp3[[2]]
  Recall_CMI[times]             <- temp3[[3]]
  F1_CMI[times]                 <- temp3[[4]]
  AIC_CMI[times]                <- temp3[[5]]
  alpha2.hat_CMI[[times]]      <- temp3[[6]]
  alpha3.hat_CMI[[times]]      <- temp3[[7]]
  mse_alpha2_CMI[times]         <- temp3[[8]]
  mse_alpha3_CMI[times]         <- temp3[[9]]
}
######## The end of factor regression based on the CMI method #########
####################################################################
######## Factor regression based on the MBI method #########
Accuracy_MBI<-Precision_MBI<-Recall_MBI<-F1_MBI<-AIC_MBI<-
  mse_alpha2_MBI<-mse_alpha3_MBI<-array()
x.1.score.hat<-x.2.score.hat<-x.3.score.hat<-rho.hat_std_MBI<-
  alpha2.hat_MBI<-alpha3.hat_MBI<-vector('list',Iter.times)
source("Imputing_data_MBI_setting2.R")
for (times in 1:Iter.times) {
  temp1 <- Imputing_data_MBI_setting2(NN, x_std[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]],
x.3.score_NA[[times]])
  x.1.score.hat[[times]]      <- temp1[[1]]
  x.2.score.hat[[times]]      <- temp1[[2]]
  x.3.score.hat[[times]]      <- temp1[[3]]
}
source("Construct_factor_PCA.R")
for (times in 1:Iter.times){
  temp2 <- Construct_factor_PCA(x_std[[times]], psi.1.hat[[times]], psi.2.hat[[times]], psi.3.hat[[times]], x.1.score.hat[[times]],
x.2.score.hat[[times]], x.3.score.hat[[times]])
  rho.hat_std_MBI[[times]]   <- temp2[[1]]
}
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_PCA(rho.hat_std_MBI[[times]], alpha2, alpha3, y[[times]])
  Accuracy_MBI[times]          <- temp3[[1]]
  Precision_MBI[times]         <- temp3[[2]]
```

```
  Recall_MBI[times]              <- temp3[[3]]
  F1_MBI[times]                  <- temp3[[4]]
  AIC_MBI[times]                 <- temp3[[5]]
  alpha2.hat_MBI[[times]]     <- temp3[[6]]
  alpha3.hat_MBI[[times]]     <- temp3[[7]]
  mse_alpha2_MBI[times]        <- temp3[[8]]
  mse_alpha3_MBI[times]        <- temp3[[9]]
}
######## The end of factor regression based on the MBI method #########
#result----
result <- round(rbind(
c(mean(Accuracy_CMI),sd(Accuracy_CMI),mean(Precision_CMI),mean(Recall_CMI),mean(F1_CMI),
mean(mse_alpha2_CMI),sd(mse_alpha2_CMI),mean(mse_alpha3_CMI),sd(mse_alpha3_CMI)),
c(mean(Accuracy_MBI),sd(Accuracy_MBI),mean(Precision_MBI),mean(Recall_MBI),mean(F1_MBI),
mean(mse_alpha2_MBI),sd(mse_alpha2_MBI),mean(mse_alpha3_MBI),sd(mse_alpha3_MBI))),4)
print(result)
stargazer(result, title = "Evaluation",   align = F, type = "latex")
```

Functions: all the functions used in the main program are listed below.

The function "Construct_factor_CCA" is used to construct canonical scores as factors.

```
Construct_factor_CCA = function(x_std, psi.1.hat, psi.2.hat, psi.3.hat,
                                x.1.score_NA, x.2.score_NA, x.3.score_NA){
  score<-cbind(x.1.score_NA, x.2.score_NA, x.3.score_NA)
  z.hat<-t(score) %*% score /(N-1)
  rho.hat_std_temp<-x.1.score_NA%*% eigen(z.hat)$vec[1:k1,1:(k1+k2+k3)] +
    x.2.score_NA%*% eigen(z.hat)$vec[(k1+1):(k1+k2),1:(k1+k2+k3)] +
    x.3.score_NA%*%  eigen(z.hat)$vec[(k1+k2+1):(k1+k2+k3),1:(k1+k2+k3)]  #  Estimated  MFPC  scores  with  order
N*(k1+k2+k3) based on the correlation matrix
  x.hat_std<-rho.hat_std_temp %*% cbind(psi.1.hat,psi.2.hat,psi.3.hat)
  rho.hat_std<-scale(rho.hat_std_temp)[,1:M0]
  x_std[c((sum(NN[1:3])+1):sum(NN[1:4]),(sum(NN[1:5])+1):sum(NN[1:7])),1:T1]<-
    x.hat_std[c((sum(NN[1:3])+1):sum(NN[1:4]),(sum(NN[1:5])+1):sum(NN[1:7])),1:T1]

x_std[c((sum(NN[1:2])+1):sum(NN[1:3]),(sum(NN[1:4])+1):sum(NN[1:5]),(sum(NN[1:6])+1):sum(NN[1:7])),(T1+1):(T1+T2)]<-

x.hat_std[c((sum(NN[1:2])+1):sum(NN[1:3]),(sum(NN[1:4])+1):sum(NN[1:5]),(sum(NN[1:6])+1):sum(NN[1:7])),(T1+1):(T1+T2)]
  x_std[c((NN[1]+1):sum(NN[1:2]),(sum(NN[1:4])+1):sum(NN[1:6])),(T1+T2+1):(T1+T2+T3)]<-
    x.hat_std[c((NN[1]+1):sum(NN[1:2]),(sum(NN[1:4])+1):sum(NN[1:6])),(T1+T2+1):(T1+T2+T3)]
  Z1 = x.1.score_NA
  Z2 = x.2.score_NA
  Z3 = x.3.score_NA
  Omega1 = Z1%*%solve(t(Z1)%*%Z1)%*%t(Z1)
  Omega2 = Z2%*%solve(t(Z2)%*%Z2)%*%t(Z2)
  Omega3 = Z3%*%solve(t(Z3)%*%Z3)%*%t(Z3)
  evd = eigen(Omega1 + Omega2 + Omega2)
  f.hat = evd$vectors[,1:M0]
  return(list(f.hat, x_std))
}
```

The function "Construct_factor_PCA" is used to construct multi-source functional principal component scores as factors.

```
Construct_factor_PCA = function(x_std, psi.1.hat, psi.2.hat, psi.3.hat,
                                x.1.score_NA, x.2.score_NA, x.3.score_NA){
  score<-cbind(x.1.score_NA, x.2.score_NA, x.3.score_NA)
```

```
  z.hat<-t(score) %*% score /(N-1)
  rho.hat_std_temp<-x.1.score_NA%*% eigen(z.hat)$vec[1:k1,1:(k1+k2+k3)] +
    x.2.score_NA%*% eigen(z.hat)$vec[(k1+1):(k1+k2),1:(k1+k2+k3)] +
    x.3.score_NA%*% eigen(z.hat)$vec[(k1+k2+1):(k1+k2+k3),1:(k1+k2+k3)] # Estimated MFPC scores with order N*(k1+k2+k3) based on the correlation matrix
  x.hat_std<-rho.hat_std_temp %*% cbind(psi.1.hat,psi.2.hat,psi.3.hat)
  rho.hat_std<-scale(rho.hat_std_temp)[,1:M0]

  x_std[c((sum(NN[1:3])+1):sum(NN[1:4]),(sum(NN[1:5])+1):sum(NN[1:7])),1:T1]<-
    x.hat_std[c((sum(NN[1:3])+1):sum(NN[1:4]),(sum(NN[1:5])+1):sum(NN[1:7])),1:T1]

x_std[c((sum(NN[1:2])+1):sum(NN[1:3]),(sum(NN[1:4])+1):sum(NN[1:5]),(sum(NN[1:6])+1):sum(NN[1:7])),(T1+1):(T1+T2)]<-
x.hat_std[c((sum(NN[1:2])+1):sum(NN[1:3]),(sum(NN[1:4])+1):sum(NN[1:5]),(sum(NN[1:6])+1):sum(NN[1:7])),(T1+1):(T1+T2)]
x_std[c((NN[1]+1):sum(NN[1:2]),(sum(NN[1:4])+1):sum(NN[1:6])),(T1+T2+1):(T1+T2+T3)]<-x.hat_std[c((NN[1]+1):sum(NN[1:2]),(sum(NN[1:4])+1):sum(NN[1:6])),(T1+T2+1):(T1+T2+T3)]
  return(list(rho.hat_std, x_std))
}
### Function "Factor_regression_Imputed_CCA" was used to build Logistic regression model with canonical scores as factors. ###
Factor_regression_Imputed_CCA<-function(f, alpha2, alpha3, y){
  data = data.frame(cbind(f[,1:M0]), y)
  data$y = factor(data$y)
  mult.model = multinom(y ~ .-1, data = data, MaxNWts = 10000)
  predlab = predict(mult.model, newdata = data, type = "class")
  summary = multiClassSummary(
    data.frame(obs = data$y, pred = predlab), lev = levels(data$y))
  alpha2.hat = abs(summary(mult.model)$coefficients[1,])
  alpha3.hat = abs(summary(mult.model)$coefficients[2,])
  mse_alpha2 = mean(na.omit((alpha2-alpha2.hat[1:M]))^2)
  mse_alpha3 = mean(na.omit((alpha3-alpha3.hat[1:M]))^2)
  Accuracy = summary[1]
  F1 = summary[3]
  Precision = summary[8]
  Recall = summary[9]
  AIC = mult.model$AIC
  return(list(Accuracy, Precision, Recall, F1, AIC,
              alpha2.hat, alpha3.hat, mse_alpha2, mse_alpha3))
}
### Function "Factor_regression_Imputed_PCA" was used to build Logistic regression model with multi-source functional principal component scores as factors. ###
Factor_regression_Imputed_PCA<-function(rho.hat_std, alpha2, alpha3, y){
  data = data.frame(cbind(rho.hat_std[,1:M0]), y)
  data$y = factor(data$y)
  mult.model = multinom(y ~ .-1, data = data, MaxNWts = 10000)
  predlab = predict(mult.model, newdata = data, type = "class")
  summary = multiClassSummary(
    data.frame(obs = data$y, pred = predlab), lev = levels(data$y))
  alpha2.hat = abs(summary(mult.model)$coefficients[1,])
  alpha3.hat = abs(summary(mult.model)$coefficients[2,])
  mse_alpha2 = mean(na.omit((alpha2-alpha2.hat[1:M]))^2)
  mse_alpha3 = mean(na.omit((alpha3-alpha3.hat[1:M]))^2)
  Accuracy = summary[1]
  F1 = summary[3]
```

```
  Precision = summary[8]
  Recall = summary[9]
  AIC = mult.model$AIC
  return(list(Accuracy, Precision, Recall, F1, AIC,
              alpha2.hat, alpha3.hat, mse_alpha2, mse_alpha3))
}
### The function " generate_data_setting2" generates data under Setting2. ###
generate_data_setting2<-function(phi.1,phi.2,phi.3,alpha1,alpha2,alpha3,sd,gamma1,gamma2){
  dim(phi.1)<-c((M11*M12),(T11*T12))                    ## Transfoming the tensor into matrix
  psi.1 <- phi.1*sqrt(1-gamma1-gamma2)                  ### (M11 * M12) * (T11 * T12) tensor
  psi.2 <- phi.2*sqrt(gamma1)                           ### M2 * T2 matrix
  psi.3 <- phi.3*sqrt(gamma2)                           ### M3 * T3 matrix
  rho<-apply(matrix(1:M2,nrow=M2,ncol=1), 1,
             function(x){return(rnorm(N,0,(exp(-(1+x)/2))^0.5))})
  x.1<-rho %*% psi.1                                    ## N * T1 matrix   of 1st data source: Image x(1)
  x.2<-rho %*% psi.2                                    ## N * T2 matrix   of 2nd data source: curve x(2)
  x.3<-rho %*% psi.3                                    ## N * T3 matrix   of 3rd data source: curve x(3)
  x<-cbind(x.1,x.2,x.3)                                 ## N * (T1+T2+T3)   matrix of 3-source data x
  rho_std = scale(rho)
  p1 = exp(rho_std[,1:M]%*% alpha1)/(exp(rho_std[,1:M]%*% alpha1)+exp(rho_std[,1:M]%*% alpha2)+exp(rho_std[,1:M]%*%
alpha3))
  p2 = exp(rho_std[,1:M]%*% alpha2)/(exp(rho_std[,1:M]%*% alpha1)+exp(rho_std[,1:M]%*% alpha2)+exp(rho_std[,1:M]%*%
alpha3))
  p3 = exp(rho_std[,1:M]%*% alpha3)/(exp(rho_std[,1:M]%*% alpha1)+exp(rho_std[,1:M]%*% alpha2)+exp(rho_std[,1:M]%*%
alpha3))
  p = cbind(p1, p2, p3)
  y=c()
  for(i in 1:N){
    y0 = which.max(rmultinom(1, size = 1, prob = p[i,]))
    y=c(y,y0)
  }
  return(list(rho_std, x, y, psi.1, psi.2, psi.3))
}
### The function "generate_eigenfunction_setting2" generates eigenfunctions under Setting2. ###
generate_eigenfunction_setting2<-function(N, M11, T11, M12, T12, M2, T2, M3, T3){
  tensor.product<-function(V,W){
    V.W<-array(0,dim=c(nrow(V@X)*nrow(W@X), ncol(V@X), ncol(W@X)))
    for (i in 1:nrow(V@X)) {
      for (j in 1:nrow(W@X)) {
        V.W[(i-1)*nrow(V@X)+j,,]<-V@X[i,] %*% t(W@X[j,])
      }
    }
    return(V.W)
  }
  # Fourier basis
  V<-eFun(seq(0,1,length.out = T11), M = M11, type = "Fourier")     # T11 * M11 matrix
  W<-eFun(seq(0,0.5,length.out = T12), M = M12, type = "Fourier") # T12 * M12 matrix
  phi.1<-tensor.product(V, W)     #(M11 * M12) * T11 * T12 tensor
  t.11<-seq(0,1,length.out = T11)
  t.12<-seq(0,0.5,length.out = T12)
  t.1h<-(t.11[T11]-t.11[1])*(t.12[T12]-t.12[1])

  fai.2<-eFun(seq(-1,1,length.out = T2), M = M2, type = "Poly")
```

```
  t.2<-fai.2@argvals[[1]]
  t.2h<-t.2[T2]-t.2[1]
  phi.2<-fai.2@X # phi.2 is an M2*T2 matrix

  fai.3<-eFun(seq(0,2,length.out = T3), M = M3, type = "Fourier")
  t.3<-fai.2@argvals[[1]]
  t.3h<-t.3[T3]-t.3[1]
  phi.3<-fai.3@X # phi.3 is an M3*T3 matrix
  return(list(t.11, t.12, t.2, t.3, phi.1, phi.2, phi.3, t.1h, t.2h, t.3h))
}
### The function "Imputing_data_CMI_setting2" is using the conditional mean imputation method for univariate principal component scores. ###
Imputing_data_CMI_setting2<-function(NN, x_std, x.1.score, x.2.score, x.3.score){
  x.3.score.hat<-t(cov(x.3.score[1:NN[1],],cbind(x.1.score[1:NN[1],],x.2.score[1:NN[1],])) %*%
solve(var(cbind(x.1.score[1:NN[1],],x.2.score[1:NN[1],]))) %*%
t(cbind(x.1.score[(NN[1]+1):sum(NN[1:2]),],x.2.score[(NN[1]+1):sum(NN[1:2]),])))
  x.2.score.hat<-t(cov(x.2.score[1:NN[1],],cbind(x.1.score[1:NN[1],],x.3.score[1:NN[1],])) %*%
solve(var(cbind(x.1.score[1:NN[1],],x.3.score[1:NN[1],]))) %*%
t(cbind(x.1.score[(sum(NN[1:2])+1):sum(NN[1:3]),],x.3.score[(NN[1]+1):(NN[1]+NN[3]),])))
  x.1.score.hat<-t(cov(x.1.score[1:NN[1],],cbind(x.2.score[1:NN[1],],x.3.score[1:NN[1],])) %*%
solve(var(cbind(x.2.score[1:NN[1],],x.3.score[1:NN[1],]))) %*%
t(cbind(x.2.score[(sum(NN[1:2])+1):(sum(NN[1:2])+NN[4]),],x.3.score[(NN[1]+NN[3]+1):(NN[1]+sum(NN[3:4])),])))
x.23.score.hat<-t(cov(cbind(x.2.score[1:NN[1],],x.3.score[1:NN[1],]),x.1.score[1:NN[1],]) %*%
solve(var(x.1.score[1:NN[1],])) %*% t(x.1.score[(sum(NN[1:3])+1):(sum(NN[1:3])+NN[5]),]))
x.13.score.hat<-t(cov(cbind(x.1.score[1:NN[1],],x.3.score[1:NN[1],]),x.2.score[1:NN[1],]) %*%
solve(var(x.2.score[1:NN[1],])) %*%
t(x.2.score[(sum(NN[1:2])+NN[4]+1):(sum(NN[1:2])+NN[4]+NN[6]),]))
x.12.score.hat<-t(cov(cbind(x.1.score[1:NN[1],],x.2.score[1:NN[1],]),x.3.score[1:NN[1],]) %*%
solve(var(x.3.score[1:NN[1],])) %*%
t(x.3.score[(NN[1]+sum(NN[3:4])+1):(NN[1]+sum(NN[3:4])+NN[7]),]))
x.1.score_NA<-rbind(x.1.score[1:sum(NN[1:3]),],x.1.score.hat,x.1.score[(sum(NN[1:3])+1):(sum(NN[1:3])+NN[5]),],x.13.score.hat[,1:k1],x.12.score.hat[,1:k1])
x.2.score_NA<-rbind(x.2.score[1:sum(NN[1:2]),],x.2.score.hat,x.2.score[(sum(NN[1:2])+1):(sum(NN[1:2])+NN[4]),],x.23.score.hat[,1:k2],
x.2.score[(sum(NN[1:2])+NN[4]+1):(sum(NN[1:2])+NN[4]+NN[6]),],x.12.score.hat[,(k2+1):(k1+k2)])x.3.score_NA<-rbind(x.3.score[1:NN[1],],x.3.score.hat,x.3.score[(NN[1]+1):(NN[1]+sum(NN[3:4])),],x.23.score.hat[,(k2+1):(k2+k3)],x.13.score.hat[,(k1+1):(k1+k3)],x.3.score[(NN[1]+sum(NN[3:4])+1):(NN[1]+sum(NN[3:4])+NN[7]),])

  n_iter = 5
  for(j in 1:n_iter)
  {
    x.3.score.hat<-t(cov(x.3.score_NA,cbind(x.1.score_NA,x.2.score_NA)) %*%
solve(var(cbind(x.1.score_NA,x.2.score_NA))) %*% t(cbind(x.1.score_NA,x.2.score_NA)[(NN[1]+1):sum(NN[1:2]),]))
    x.2.score.hat<-t(cov(x.2.score_NA,cbind(x.1.score_NA,x.3.score_NA)) %*%
                         solve(var(cbind(x.1.score_NA,x.3.score_NA))) %*%
                         t(cbind(x.1.score_NA,x.3.score_NA)[(sum(NN[1:2])+1):sum(NN[1:3]),]))
    x.1.score.hat<-t(cov(x.1.score_NA,cbind(x.2.score_NA,x.3.score_NA)) %*%
                         solve(var(cbind(x.2.score_NA,x.3.score_NA))) %*%
t(cbind(x.2.score_NA,x.3.score_NA)[(sum(NN[1:3])+1):sum(NN[1:4]),]))
    x.23.score.hat<-t(cov(cbind(x.2.score_NA,x.3.score_NA),x.1.score_NA) %*%
                          solve(var(x.1.score_NA)) %*%
                          t(x.1.score_NA[(sum(NN[1:4])+1):sum(NN[1:5]),]))
    x.13.score.hat<-t(cov(cbind(x.1.score_NA,x.3.score_NA),x.2.score_NA) %*%
```

```
                                    solve(var(x.2.score_NA)) %*%
                                    t(x.2.score_NA[(sum(NN[1:5])+1):sum(NN[1:6]),]))
    x.12.score.hat<-t(cov(cbind(x.1.score_NA,x.2.score_NA),x.3.score_NA) %*%
                                    solve(var(x.3.score_NA)) %*%
                                    t(x.3.score_NA[(sum(NN[1:6])+1):sum(NN[1:7]),]))
    x.1.score_NA<-rbind(x.1.score[1:sum(NN[1:3]),],
                                    x.1.score.hat,
                                    x.1.score[(sum(NN[1:3])+1):(sum(NN[1:3])+NN[5]),],
                                    x.13.score.hat[,1:k1],
                                    x.12.score.hat[,1:k1])
    x.2.score_NA<-rbind(x.2.score[1:sum(NN[1:2]),],
                                    x.2.score.hat,
                                    x.2.score[(sum(NN[1:2])+1):(sum(NN[1:2])+NN[4]),],
                                    x.23.score.hat[,1:k2],
                                    x.2.score[(sum(NN[1:2])+NN[4]+1):(sum(NN[1:2])+NN[4]+NN[6]),],
                                    x.12.score.hat[,(k2+1):(k1+k2)])
    x.3.score_NA<-rbind(x.3.score[1:NN[1],],
                                    x.3.score.hat,
                                    x.3.score[(NN[1]+1):(NN[1]+sum(NN[3:4])),],
                                    x.23.score.hat[,(k2+1):(k2+k3)],
                                    x.13.score.hat[,(k1+1):(k1+k3)],
                                    x.3.score[(NN[1]+sum(NN[3:4])+1):(NN[1]+sum(NN[3:4])+NN[7]),])
  }
  return(list(x.1.score_NA, x.2.score_NA, x.3.score_NA))
}
```

The function "Imputing_data_MBI_setting2" is using the multiple block-wise imputation method for univariate principal component scores.

```
Imputing_data_MBI_setting2<-function(NN, x_std, x.1.score_NA, x.2.score_NA, x.3.score_NA){
  ### X_{m(2)} = X_{2,3} ###
  x.3.score.hat1 = matrix(0,NN[2],k3)
  for(i in 1:k3){
    fit = lm(x.3.score_NA[1:NN[1],i]~.,
data = as.data.frame(cbind(x.1.score_NA[1:NN[1],],x.2.score_NA[1:NN[1],])))
    x.3.score.hat1[,i]                                                                    =
cbind(rep(1,NN[2]),x.1.score_NA[(NN[1]+1):sum(NN[1:2]),],x.2.score_NA[(NN[1]+1):sum(NN[1:2]),])%*%fit$coefficients
  }
  x.3.score.hat2 = matrix(0,NN[2],k3)
  for(i in 1:k3){
    fit = lm(cbind(x.3.score_NA[c(1:NN[1],(sum(NN[1:2])+1):sum(NN[1:3])),i])~.,
data = as.data.frame(cbind(x.1.score_NA[c(1:NN[1],(sum(NN[1:2])+1):sum(NN[1:3])),])))
    x.3.score.hat2[,i] = cbind(rep(1,NN[2]),x.1.score_NA[(NN[1]+1):sum(NN[1:2]),])%*%fit$coefficients
  }
  x.3.score.hat3 = matrix(0,NN[2],k3)
  for(i in 1:k3){
    fit = lm(cbind(x.3.score_NA[c(1:NN[1],(sum(NN[1:3])+1):sum(NN[1:4])),i])~.,
data = as.data.frame(cbind(x.2.score_NA[c(1:NN[1],(sum(NN[1:3])+1):sum(NN[1:4])),])))
    x.3.score.hat3[,i] = cbind(rep(1,NN[2]),x.2.score_NA[(NN[1]+1):sum(NN[1:2]),])%*%fit$coefficients
  }
  x.3.score_NA[(NN[1]+1):sum(NN[1:2]),] =
    (x.3.score.hat1 + x.3.score.hat2 + x.3.score.hat3) / 3

  ### X_{m(3)} = X_{3,2} ###
  x.2.score.hat1 = matrix(0,NN[3],k2)
```

```
  for(i in 1:k2){
    fit = lm(x.2.score_NA[1:NN[1],i]~.,
data = as.data.frame(cbind(x.1.score_NA[1:NN[1],],x.3.score_NA[1:NN[1],])))
    x.2.score.hat1[,i]                                                                                                    =
cbind(rep(1,NN[3]),x.1.score_NA[(sum(NN[1:2])+1):sum(NN[1:3]),],x.3.score_NA[(sum(NN[1:2])+1):sum(NN[1:3]),])%*%fit$co
efficients
  }
  x.2.score.hat2 = matrix(0,NN[3],k2)
  for(i in 1:k2){
    fit = lm(cbind(x.2.score_NA[1:sum(NN[1:2]),i])~.,
data = as.data.frame(cbind(x.1.score_NA[1:sum(NN[1:2]),])))
    x.2.score.hat2[,i] = cbind(rep(1,NN[3]),x.1.score_NA[(sum(NN[1:2])+1):sum(NN[1:3]),])%*%fit$coefficients
  }
  x.2.score.hat3 = matrix(0,NN[3],k2)
  for(i in 1:k2){
    fit = lm(cbind(x.2.score_NA[c(1:NN[1],(sum(NN[1:3])+1):sum(NN[1:4])),i])~.,
data = as.data.frame(cbind(x.3.score_NA[c(1:NN[1],(sum(NN[1:3])+1):sum(NN[1:4])),])))
    x.2.score.hat3[,i] = cbind(rep(1,NN[3]),x.3.score_NA[(sum(NN[1:2])+1):sum(NN[1:3]),])%*%fit$coefficients
  }
  x.2.score_NA[(sum(NN[1:2])+1):sum(NN[1:3]),] =
    (x.2.score.hat1 + x.2.score.hat2 + x.2.score.hat3) / 3
  ### X_{m(4)} = X_{4,1} ###
  x.1.score.hat1 = matrix(0,NN[4],k1)
  for(i in 1:k1){
    fit = lm(x.1.score_NA[1:NN[1],i]~.,
data = as.data.frame(cbind(x.2.score_NA[1:NN[1],],x.3.score_NA[1:NN[1],])))
x.1.score.hat1[,i]                                                                                                        =
cbind(rep(1,NN[4]),x.2.score_NA[(sum(NN[1:3])+1):sum(NN[1:4]),],x.3.score_NA[(sum(NN[1:3])+1):sum(NN[1:4]),])%*%fit$co
efficients
  }
  x.1.score.hat2 = matrix(0,NN[4],k1)
  for(i in 1:k1){
    fit = lm(cbind(x.1.score_NA[1:sum(NN[1:2]),i])~.,
data = as.data.frame(cbind(x.2.score_NA[1:sum(NN[1:2]),])))
    x.1.score.hat2[,i] = cbind(rep(1,NN[4]),x.2.score_NA[(sum(NN[1:3])+1):sum(NN[1:4]),])%*%fit$coefficients
  }
  x.1.score.hat3 = matrix(0,NN[4],k1)
  for(i in 1:k1){
    fit = lm(cbind(x.1.score_NA[c(1:NN[1],(sum(NN[1:2])+1):sum(NN[1:3])),i])~.,
           data = as.data.frame(cbind(x.3.score_NA[c(1:NN[1],(sum(NN[1:2])+1):sum(NN[1:3])),])))
    x.1.score.hat3[,i] = cbind(rep(1,NN[4]),x.3.score_NA[(sum(NN[1:3])+1):sum(NN[1:4]),])%*%fit$coefficients
  }
  x.1.score.hat                                                                                                           =
t(cov(x.1.score_NA[c(1:NN[1],(sum(NN[1:2])+1):sum(NN[1:3])),],cbind(x.3.score_NA[c(1:NN[1],(sum(NN[1:2])+1):sum(NN[1:3
])),]))) %*%
                        solve(var(cbind(x.3.score_NA[c(1:NN[1],(sum(NN[1:2])+1):sum(NN[1:3])),]))) %*%
                        t(cbind(x.3.score_NA[(sum(NN[1:3])+1):sum(NN[1:4]),]))
  x.1.score_NA[(sum(NN[1:3])+1):sum(NN[1:4]),] =
    (x.1.score.hat1 + x.1.score.hat2 + x.1.score.hat3) / 3
  ### X_{m(5)} = X_{5,2} and x_{5,3} ###
  x.23.score.hat = matrix(0,NN[5],(k2+k3))
  for(i in 1:k2){
    fit = lm(cbind(x.2.score_NA[1:NN[1],i])~.,
```

```
            data = as.data.frame(cbind(x.1.score_NA[1:NN[1],])))
    x.23.score.hat[,i] = cbind(rep(1,NN[5]),x.1.score_NA[(sum(NN[1:4])+1):sum(NN[1:5]),])%*%fit$coefficients
  }
  for(i in 1:k3){
    fit = lm(cbind(x.3.score_NA[1:NN[1],i])~.,
            data = as.data.frame(cbind(x.1.score_NA[1:NN[1],])))
    x.23.score.hat[,(k2+i)] = cbind(rep(1,NN[5]),x.1.score_NA[(sum(NN[1:4])+1):sum(NN[1:5]),])%*%fit$coefficients
  }
  x.2.score_NA[(sum(NN[1:4])+1):sum(NN[1:5]),] = x.23.score.hat[,1:k2]
  x.3.score_NA[(sum(NN[1:4])+1):sum(NN[1:5]),] = x.23.score.hat[,(k2+1):(k2+k3)]
  ### X_{m(6)} = X_{6,1} and x_{6,3} ###
  x.13.score.hat = matrix(0,NN[6],(k1+k3))
  for(i in 1:k1){
    fit = lm(cbind(x.1.score_NA[1:NN[1],i])~.,
            data = as.data.frame(cbind(x.2.score_NA[1:NN[1],])))
    x.13.score.hat[,i] = cbind(rep(1,NN[6]),x.2.score_NA[(sum(NN[1:5])+1):sum(NN[1:6]),])%*%fit$coefficients
  }
  for(i in 1:k3){
    fit = lm(cbind(x.3.score_NA[1:NN[1],i])~.,
            data = as.data.frame(cbind(x.2.score_NA[1:NN[1],])))
    x.13.score.hat[,(k1+i)] = cbind(rep(1,NN[6]),x.2.score_NA[(sum(NN[1:5])+1):sum(NN[1:6]),])%*%fit$coefficients
  }
  x.1.score_NA[(sum(NN[1:5])+1):sum(NN[1:6]),] = x.13.score.hat[,1:k1]
  x.3.score_NA[(sum(NN[1:5])+1):sum(NN[1:6]),] = x.13.score.hat[,(k1+1):(k1+k3)]
  ### X_{m(7)} = X_{7,1} and x_{7,2} ###
  x.12.score.hat = matrix(0,NN[7],(k1+k2))
  for(i in 1:k1){
    fit = lm(cbind(x.1.score_NA[1:NN[1],i])~.,
            data = as.data.frame(cbind(x.3.score_NA[1:NN[1],])))
    x.12.score.hat[,i] = cbind(rep(1,NN[7]),x.3.score_NA[(sum(NN[1:6])+1):sum(NN[1:7]),])%*%fit$coefficients
  }
  for(i in 1:k2){
    fit = lm(cbind(x.2.score_NA[1:NN[1],i])~.,
            data = as.data.frame(cbind(x.3.score_NA[1:NN[1],])))
    x.12.score.hat[,(k1+i)] = cbind(rep(1,NN[7]),x.3.score_NA[(sum(NN[1:6])+1):sum(NN[1:7]),])%*%fit$coefficients
  }
  x.1.score_NA[(sum(NN[1:6])+1):sum(NN[1:7]),] = x.12.score.hat[,1:k1]
  x.2.score_NA[(sum(NN[1:6])+1):sum(NN[1:7]),] = x.12.score.hat[,(k1+1):(k1+k2)]
  return(list(x.1.score_NA, x.2.score_NA, x.3.score_NA))
}
```

Function "MCCA_complete_data_setting2" is used to conduct multi-set canonical correlation analysis on complete data and construct canonical scores.

```
MCCA_complete_data_setting2<-function(NN, N, T1, T2, T3, x_std, t.1h, t.2h, t.3h){
  ##### Computing the FPC scores and curves for each data source by SVD method #############
s1<-svd(x_std[c(1:sum(NN[1:3]),(sum(NN[1:4])+1):sum(NN[1:5])),1:T1]/sqrt(sum(NN[c(1:3,5)])))    ## SVD for x.1 on the observed data NN[1,2,3,5]
  phi.1.hat<-t(s1$v[,1:k1]) * (t.1h/T1)^(-0.5)                                ## The univariate FPC curves of x.1 with order k1*T1
  x.1.score<-x_std[c(1:sum(NN[1:3]),(sum(NN[1:4])+1):sum(NN[1:5])),1:T1] %*% t(phi.1.hat) * (t.1h/T1)## The univariate FPC scores of x.1
  x.1.score_NA <- matrix(NA,N,k1)
  x.1.score_NA[1:sum(NN[1:3]),] <- x.1.score[1:sum(NN[1:3]),]
  x.1.score_NA[(sum(NN[1:4])+1):(sum(NN[1:5])),] <- x.1.score[(sum(NN[1:3])+1):(sum(NN[1:3])+NN[5]),]
```

```
s2<-svd(x_std[c(1:sum(NN[1:2]),(sum(NN[1:3])+1):sum(NN[1:4]),(sum(NN[1:5])+1):sum(NN[1:6])),(T1+1):(T1+T2)]/sqrt((sum(NN[c(1:2,4,6)]))))## SVD for x.2 on the observed data NN[1,2,4,6]
  phi.2.hat<-t(s2$v[,1:k2]) * (t.2h/T2)^(-0.5)                    ## The univariate FPC curves of x.2 with order k2*T2
x.2.score<-x_std[c(1:sum(NN[1:2]),(sum(NN[1:3])+1):sum(NN[1:4]),(sum(NN[1:5])+1):sum(NN[1:6])),(T1+1):(T1+T2)]     %*% t(phi.2.hat) * (t.2h/T2) #The univariate FPCA of x.2
  x.2.score_NA <- matrix(NA, N, k2)
  x.2.score_NA[1:sum(NN[1:2]),] <- x.2.score[1:sum(NN[1:2]),]
  x.2.score_NA[(sum(NN[1:3])+1):(sum(NN[1:4])),] <- x.2.score[(sum(NN[1:2])+1):(sum(NN[1:2])+NN[4]),]
  x.2.score_NA[(sum(NN[1:5])+1):(sum(NN[1:6])),] <- x.2.score[(sum(NN[1:2])+NN[4]+1):(sum(NN[1:2])+NN[4]+NN[6]),]
s3<-svd(x_std[c(1:NN[1],(sum(NN[1:2])+1):sum(NN[1:4]),(sum(NN[1:6])+1):sum(NN[1:7])),(T1+T2+1):(T1+T2+T3)]/sqrt((NN[1]+sum(NN[3:4])+NN[7])))## SVD for x.3 on the observed data NN[1,3,4,7]
  phi.3.hat<-t(s3$v[,1:k3]) * (t.3h/T3)^(-0.5)                    ## The univariate FPC curves of x.3 with order k3*T3
x.3.score<-x_std[c(1:NN[1],(sum(NN[1:2])+1):sum(NN[1:4]),(sum(NN[1:6])+1):sum(NN[1:7])),(T1+T2+1):(T1+T2+T3)]     %*% t(phi.3.hat) * (t.3h/T3) #The univariate FPCA of x.3
  x.3.score_NA <- matrix(NA, N, k3)
  x.3.score_NA[1:NN[1],] <- x.3.score[1:NN[1],]
  x.3.score_NA[(sum(NN[1:2])+1):(sum(NN[1:4])),] <- x.3.score[(NN[1]+1):(NN[1]+sum(NN[3:4])),]
  x.3.score_NA[(sum(NN[1:6])+1):(sum(NN[1:7])),] <- x.3.score[(NN[1]+sum(NN[3:4])+1):(NN[1]+sum(NN[3:4])+NN[7]),]
  ##### The end of Computing the FPC scores and curves for each data source by SVD method ############
  ######### The multivariate FPCA of x.1,x.2 and x.3 on the complete-data #########
  score<-cbind(x.1.score[1:NN[1],],x.2.score[1:NN[1],],x.3.score[1:NN[1],])     ##   N1*(k1+k2+k3)
  z.hat<-t(score) %*% score /(NN[1]-1)
  psi.1.hat<-t(eigen(z.hat)$vec[1:k1,1:(k1+k2+k3)]) %*% phi.1.hat                 ## The estimated Multivariate FPC eigenfunctions of x.1 with order (k1+k2+k3)*T1
  psi.2.hat<-t(eigen(z.hat)$vec[(k1+1):(k1+k2),1:(k1+k2+k3)]) %*% phi.2.hat          ## The estimated Multivariate FPC eigenfunctions of x.1 with order (k1+k2+k3)*T2
  psi.3.hat<-t(eigen(z.hat)$vec[(k1+k2+1):(k1+k2+k3),1:(k1+k2+k3)]) %*% phi.3.hat      ## The estimated Multivariate FPC eigenfunctions of x.1 with order (k1+k2+k3)*T3

  ######### The multivariate FPCA of x.1 and x.2 on the complete-data #########
  Z1 = x.1.score[1:NN[1],]
  Z2 = x.2.score[1:NN[1],]
  Z3 = x.3.score[1:NN[1],]
  Omega1 = Z1%*%solve(t(Z1)%*%Z1)%*%t(Z1)
  Omega2 = Z2%*%solve(t(Z2)%*%Z2)%*%t(Z2)
  Omega3 = Z3%*%solve(t(Z3)%*%Z3)%*%t(Z3)
  evd = eigen(Omega1 + Omega2 + Omega3)
  f.hat = evd$vectors[,1:M0]
  return(list(psi.1.hat,psi.2.hat,psi.3.hat,
              x.1.score_NA,x.2.score_NA,x.3.score_NA,
              x.1.score,x.2.score,x.3.score,f.hat))
}
```

Function "MCCA_setting2" is used to conduct multi-set canonical correlation analysis on all data and construct canonical scores.

```
MCCA_setting2 <- function(x, k1, k2, k3, t.1h, t.2h, t.3h){
  ##################################################
  ##### Computing the FPC scores and curves for each data source by SVD method ############
  s1<-svd(x[,1:T1]/sqrt(N))
  phi.1.hat<-t(s1$v[,1:k1]) * (t.1h/T1)^(-0.5)
  x.1.score<-x[,1:T1] %*% t(phi.1.hat) * (t.1h/T1) #The univariate FPCA of x.1
  s2<-svd(x[,(T1+1):(T1+T2)]/sqrt(N))
```

```
  phi.2.hat<-t(s2$v[,1:k2]) * (t.2h/T2)^(-0.5)
  x.2.score<-x[,(T1+1):(T1+T2)] %*% t(phi.2.hat) * (t.2h/T2) #The univariate FPCA of x.2
  s3<-svd(x[,(T1+T2+1):(T1+T2+T3)]/sqrt(N))
  phi.3.hat<-t(s3$v[,1:k3]) * (t.3h/T3)^(-0.5)
  x.3.score<-x[,(T1+T2+1):(T1+T2+T3)] %*% t(phi.3.hat) * (t.3h/T3) #The univariate FPCA of x.3
  ##### The end of Computing the FPC scores and curves for each data source by SVD method ############
  #################################################
  #################################################
  ##### Computing the Canonical scores ############
  Z1 = x.1.score
  Z2 = x.2.score
  Z3 = x.3.score
  Omega1 = Z1%*%ginv(t(Z1)%*%Z1)%*%t(Z1)
  Omega2 = Z2%*%ginv(t(Z2)%*%Z2)%*%t(Z2)
  Omega3 = Z3%*%ginv(t(Z3)%*%Z3)%*%t(Z3)
  evd = eigen(Omega1 + Omega2 + Omega3)
  f.hat = evd$vectors[,1:M0]
  #####The end of   Computing the Canonical scores ############
  #################################################
  return(list(f.hat))
}
```

Function "MFPCA_complete_data_setting2" is used to conduct multi-source functional principal component analysis on complete data and construct multi-source principal component scores.

```
MFPCA_complete_data_setting2<-function(NN, N, T1, T2, T3, x_std, t.1h, t.2h, t.3h){
  ##### Computing the FPC scores and curves for each data source by SVD method ############
s1<-svd(x_std[c(1:sum(NN[1:3]),(sum(NN[1:4])+1):sum(NN[1:5])),1:T1]/sqrt(sum(NN[c(1:3,5)])))    ## SVD for x.1 on the observed data NN[1,2,3,5]
  phi.1.hat<-t(s1$v[,1:k1]) * (t.1h/T1)^(-0.5)                                    ## The univariate FPC curves of x.1 with order k1*T1
  x.1.score<-x_std[c(1:sum(NN[1:3]),(sum(NN[1:4])+1):sum(NN[1:5])),1:T1] %*% t(phi.1.hat) * (t.1h/T1)## The univariate FPC scores of x.1
  x.1.score_NA <- matrix(NA,N,k1)
  x.1.score_NA[1:sum(NN[1:3]),] <- x.1.score[1:sum(NN[1:3]),]
  x.1.score_NA[(sum(NN[1:4])+1):(sum(NN[1:5])),] <- x.1.score[(sum(NN[1:3])+1):(sum(NN[1:3])+NN[5]),]
s2<-svd(x_std[c(1:sum(NN[1:2]),(sum(NN[1:3])+1):sum(NN[1:4]),(sum(NN[1:5])+1):sum(NN[1:6])),(T1+1):(T1+T2)]/sqrt((sum(NN[c(1:2,4,6)]))))## SVD for x.2 on the observed data NN[1,2,4,6]
  phi.2.hat<-t(s2$v[,1:k2]) * (t.2h/T2)^(-0.5)                        ## The univariate FPC curves of x.2 with order k2*T2
  x.2.score<-x_std[c(1:sum(NN[1:2]),(sum(NN[1:3])+1):sum(NN[1:4]),(sum(NN[1:5])+1):sum(NN[1:6])),(T1+1):(T1+T2)]  %*% t(phi.2.hat) * (t.2h/T2) #The univariate FPCA of x.2
  x.2.score_NA <- matrix(NA, N, k2)
  x.2.score_NA[1:sum(NN[1:2]),] <- x.2.score[1:sum(NN[1:2]),]
  x.2.score_NA[(sum(NN[1:3])+1):(sum(NN[1:4])),] <- x.2.score[(sum(NN[1:2])+1):(sum(NN[1:2])+NN[4]),]
  x.2.score_NA[(sum(NN[1:5])+1):(sum(NN[1:6])),] <- x.2.score[(sum(NN[1:2])+NN[4]+1):(sum(NN[1:2])+NN[4]+NN[6]),]
s3<-svd(x_std[c(1:NN[1],(sum(NN[1:2])+1):sum(NN[1:4]),(sum(NN[1:6])+1):sum(NN[1:7])),(T1+T2+1):(T1+T2+T3)]/sqrt((NN[1]+sum(NN[3:4])+NN[7])))## SVD for x.3 on the observed data NN[1,3,4,7]
  phi.3.hat<-t(s3$v[,1:k3]) * (t.3h/T3)^(-0.5)                        ## The univariate FPC curves of x.3 with order k3*T3
x.3.score<-x_std[c(1:NN[1],(sum(NN[1:2])+1):sum(NN[1:4]),(sum(NN[1:6])+1):sum(NN[1:7])),(T1+T2+1):(T1+T2+T3)]    %*% t(phi.3.hat) * (t.3h/T3) #The univariate FPCA of x.3
  x.3.score_NA <- matrix(NA, N, k3)
  x.3.score_NA[1:NN[1],] <- x.3.score[1:NN[1],]
  x.3.score_NA[(sum(NN[1:2])+1):(sum(NN[1:4])),] <- x.3.score[(NN[1]+1):(NN[1]+sum(NN[3:4])),]
```

```
  x.3.score_NA[(sum(NN[1:6])+1):(sum(NN[1:7])),] <- x.3.score[(NN[1]+sum(NN[3:4])+1):(NN[1]+sum(NN[3:4])+NN[7]),]
  ##### The end of Computing the FPC scores and curves for each data source by SVD method ############
  ######### The multivariate FPCA of x.1,x.2 and x.3 on the complete-data #########
  score<-cbind(x.1.score[1:NN[1],],x.2.score[1:NN[1],],x.3.score[1:NN[1],])      ##   N1*(k1+k2+k3)
  z.hat<-t(score) %*% score /(NN[1]-1)
  psi.1.hat<-t(eigen(z.hat)$vec[1:k1,1:(k1+k2+k3)]) %*% phi.1.hat                       ## The estimated Multivariate FPC eigenfunctions of x.1 with order (k1+k2+k3)*T1
  psi.2.hat<-t(eigen(z.hat)$vec[(k1+1):(k1+k2),1:(k1+k2+k3)]) %*% phi.2.hat             ## The estimated Multivariate FPC eigenfunctions of x.1 with order (k1+k2+k3)*T2
  psi.3.hat<-t(eigen(z.hat)$vec[(k1+k2+1):(k1+k2+k3),1:(k1+k2+k3)]) %*% phi.3.hat       ## The estimated Multivariate FPC eigenfunctions of x.1 with order (k1+k2+k3)*T3
  rho.hat_std_com_t<-x.1.score[1:NN[1],]%*% eigen(z.hat)$vec[1:k1,1:(k1+k2+k3)] +
                     x.2.score[1:NN[1],] %*% eigen(z.hat)$vec[(k1+1):(k1+k2),1:(k1+k2+k3)] +
                     x.3.score[1:NN[1],]  %*%  eigen(z.hat)$vec[(k1+k2+1):(k1+k2+k3),1:(k1+k2+k3)]#  The  estimated multivariate FPC scores   of the complete-data
  rho.hat_std_com<-scale(rho.hat_std_com_t)[,1:M0]
  return(list(psi.1.hat,psi.2.hat,psi.3.hat,
              x.1.score_NA,x.2.score_NA,x.3.score_NA,
              x.1.score,x.2.score,x.3.score,rho.hat_std_com))
}
```

Function "MFPCA_setting2" is used to conduct multi-source functional principal component analysis on all data and construct multi-source principal component scores.

```
MFPCA_setting2 <- function(x_co, k1, k2, k3, t.1h, t.2h, t.3h){
  ##################################################
  ##### Computing the FPC scores and curves for each data source by SVD method ############
  s1<-svd(x_co[,1:T1]/sqrt(N))
  phi.1.hat<-t(s1$v[,1:k1]) * (t.1h/T1)^(-0.5)
  x.1.score<-x_co[,1:T1] %*% t(phi.1.hat) * (t.1h/T1) #The univariate FPCA of x.1
  s2<-svd(x_co[,(T1+1):(T1+T2)]/sqrt(N))
  phi.2.hat<-t(s2$v[,1:k2]) * (t.2h/T2)^(-0.5)
  x.2.score<-x_co[,(T1+1):(T1+T2)] %*% t(phi.2.hat) * (t.2h/T2) #The univariate FPCA of x.2
  s3<-svd(x_co[,(T1+T2+1):(T1+T2+T3)]/sqrt(N))
  phi.3.hat<-t(s3$v[,1:k3]) * (t.3h/T3)^(-0.5)
  x.3.score<-x_co[,(T1+T2+1):(T1+T2+T3)] %*% t(phi.3.hat) * (t.3h/T3) #The univariate FPCA of x.3
  ##### The end of Computing the FPC scores and curves for each data source by SVD method ############
  ##################################################
  ##################################################
  ##### Computing the Multi-source FPC scores and curves by the method in Ma's paper############
  score<-cbind(x.1.score,x.2.score,x.3.score)
  z.hat<-t(score) %*% score /(NN[1]-1)
  psi.hat.1<-t(eigen(z.hat)$vec[1:k1,1:(k1+k2+k3)]) %*% phi.1.hat ## The estimated Multivariate FPC eigenfunctions of x.1 with order (k1+k2+k3)*T1
  psi.hat.2<-t(eigen(z.hat)$vec[(k1+1):(k1+k2),1:(k1+k2+k3)])   %*%   phi.2.hat##  The  estimated  Multivariate  FPC eigenfunctions of x.2 with order (k1+k2+k3)*T2
  psi.hat.3<-t(eigen(z.hat)$vec[(k1+k2+1):(k1+k2+k3),1:(k1+k2+k3)])  %*%  phi.3.hat##  The  estimated  Multivariate  FPC eigenfunctions of x.3 with order (k1+k2+k3)*T3
  rho_std_temp<-x.1.score%*% eigen(z.hat)$vec[1:k1,1:(k1+k2+k3)] +
                x.2.score %*% eigen(z.hat)$vec[(k1+1):(k1+k2),1:(k1+k2+k3)] +
                x.3.score %*% eigen(z.hat)$vec[(k1+k2+1):(k1+k2+k3),1:(k1+k2+k3)]
  ## The estimated multivariate FPC scores with order N*(k1+k2+k3) of the complete-data
  rho_std<-scale(rho_std_temp)[,1:M0]
  #####The end of   Computing the Multi-source FPC scores and curves by the method in Ma's paper############
  ##################################################
```

```
  return(list(rho_std, psi.hat.1, psi.hat.2, psi.hat.3))
}
```

4. Setting2(MNAR)

Under the MNAR missing mechanism, we respectively consider the calculation of multi-source functional principal component score and canonical score of 3-source data (image, curve, curve), as well as the corresponding classification evaluation index. "N" represents the sample size.

We have two main modules as follows:

"setting_2(CCA)" represents the main program using the FR-CCA method in Setting 2;

"setting_2(PCA)" represents the main program using the FR-PCA method in Setting 2.

The functions used in the main program are all listed after the four main programs.

```
### Module 1: setting_2(CCA) ###
library(stargazer)
library(funData)
library(caret)
library(MASS)
library(nnet)
N<-300              ## The sample size
T11<-100            ## Image x(1): the sample number of x-axis
T12<-50             ## Image x(1): the sample number of y-axis
T1<-T11*T12         ## The number of sample point of Image x(1)
M11<-5              ## The number of PC scores of x(1) along x-axis
M12<-5              ## The number of PC scores of x(1) along y-axis
M1<-M11*M12         ## The number of PC scores of x(1)
T2<-200             ## The number of sample point of Image x(2)
M2<-25              ## The number of PC scores of x(2)
T3<-200             ## The number of sample point of Image x(3)
M3<-25              ## The number of PC scores of x(3)
k1=10               ## The number of truncated PC scores of x(1) in Ma's paper
k2=10               ## The number of truncated PC scores of x(2) in Ma's paper
k3=10               ## The number of truncated PC scores of x(3) in Ma's paper
gamma1<-0.3         ## The coefficient in generating MFPC curves
gamma2<-0.3
## Regression coefficient vector
alpha1<-rep(0,10)
alpha2<-2*c(0.972,0.734,0.691,0.541,0.480,0.424,0.331,0.271,0.123,0.0405)
alpha3<-4*c(0.934,0.903,0.815,0.604,0.517,0.447,0.392,0.370,0.345,0.3)
M<-length(alpha1)         ## The number of mulrivariate truncated PC scores in Happ's paper
sd <- 0.2                 ## The standard deviation of error in the regression model
## miss_ratio<-0 ##
gam1.1 = rep(1, (T1+T2+T3))
gam1.2 = gam1.3 = gam1.4 = gam1.5 = gam1.6 = gam1.7 = rep(0, (T1+T2+T3))
gam2.1 = rep(1, (T1+T2+T3))
gam2.2 = gam2.3 = gam2.4 = gam2.5 = gam2.6 = gam2.7 = rep(0, (T1+T2+T3))
gam3.1 = rep(1, (T1+T2+T3))
gam3.2 = gam3.3 = gam3.4 = gam3.5 = gam3.6 = gam3.7 = rep(0, (T1+T2+T3))
## miss_ratio<-0.2 ##
gam1.1 = rep(1, (T1+T2+T3))
gam1.2 = gam1.3 = gam1.4 = gam1.5 = gam1.6 = gam1.7 = rep(7/153, (T1+T2+T3))
gam2.1 = rep(1, (T1+T2+T3))
gam2.2 = gam2.3 = gam2.4 = gam2.5 = gam2.6 = gam2.7 = rep(1/54, (T1+T2+T3))
gam3.1 = rep(1, (T1+T2+T3))
gam3.2 = gam3.3 = gam3.4 = gam3.5 = gam3.6 = gam3.7 = rep(7/153, (T1+T2+T3))
## miss_ratio<-0.6 ##
```

```
gam1.1 = rep(1, (T1+T2+T3))
gam1.2 = gam1.3 = gam1.4 = gam1.5 = gam1.6 = gam1.7 = rep(7/23, (T1+T2+T3))
gam2.1 = rep(1, (T1+T2+T3))
gam2.2 = gam2.3 = gam2.4 = gam2.5 = gam2.6 = gam2.7 = rep(1/14, (T1+T2+T3))
gam3.1 = rep(1, (T1+T2+T3))
gam3.2 = gam3.3 = gam3.4 = gam3.5 = gam3.6 = gam3.7 = rep(7/23, (T1+T2+T3))
## miss_ratio<-0.9 ##
gam1.1 = rep(1, (T1+T2+T3))
gam1.2 = gam1.3 = gam1.4 = gam1.5 = gam1.6 = gam1.7 = rep(21/4, (T1+T2+T3))
gam2.1 = rep(1, (T1+T2+T3))
gam2.2 = gam2.3 = gam2.4 = gam2.5 = gam2.6 = gam2.7 = rep(3/22, (T1+T2+T3))
gam3.1 = rep(1, (T1+T2+T3))
gam3.2 = gam3.3 = gam3.4 = gam3.5 = gam3.6 = gam3.7 = rep(21/4, (T1+T2+T3))
Iter.times<-300
setwd("C:/Users/pc/Desktop/Logistic/Setting2(MNAR)/")
####################################################################
######    Generating the eigenfunctions of setting 3 ######
source("generate_eigenfunction_setting2.R")
temp1 <- generate_eigenfunction_setting2(N, M11, T11, M12, T12, M2, T2, M3, T3)
t.11                  <- temp1[[1]]
t.12                  <- temp1[[2]]
t.2                   <- temp1[[3]]
t.3                   <- temp1[[4]]
phi.1                 <- temp1[[5]]
phi.2                 <- temp1[[6]]
phi.3                 <- temp1[[7]]
t.1h                  <- temp1[[8]]
t.2h                  <- temp1[[9]]
t.3h                  <- temp1[[10]]
#######    The end of Generating the eigenfunctions of setting 3 ######
####################################################################
######## generating data ####################
psi.1_std<-psi.2_std<-psi.3_std<-rho_std<-f.hat_com<-y<-x_std<-vector('list',Iter.times)
source("generate_data_setting2.R")
for (times in 1:Iter.times) {
  temp1 <- generate_data_setting2(phi.1, phi.2, phi.3, alpha1, alpha2, alpha3, sd, gamma1, gamma2)
  rho_std[[times]]       <- temp1[[1]]
  x_std[[times]]         <- temp1[[2]]
  y[[times]]             <- temp1[[3]]
  psi.1_std[[times]]    <- temp1[[4]]
  psi.2_std[[times]]    <- temp1[[5]]
  psi.3_std[[times]]    <- temp1[[6]]
}
######## The end of generating data ########
####################################################################
######## generating missing pattern ####################
source("generate_missing_pattern.R")
for (times in 1:Iter.times){
  temp2 <- generate_missing_pattern(rho_std[[times]], x_std[[times]], y[[times]])
  NN                          <- temp2[[1]]
  rho_std[[times]]       <- temp2[[2]]
  x_std[[times]]         <- temp2[[3]]
  y[[times]]             <- temp2[[4]]
```

```
}
######## The end of generating missing pattern ########
M0=10
####################################################################
######### missing rate = 0 ####################
source("MCCA_setting2.R")
for (times in 1:Iter.times){
  temp2 <- MCCA_setting2(x_std[[times]], k1, k2, k3, t.1h, t.2h, t.3h)
  f.hat_com[[times]]     <- temp2[[1]]
}
Accuracy_com<-Precision_com<-Recall_com<-F1_com<-AIC_com<-array()
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
  temp5 <- Factor_regression_Imputed_CCA(f.hat_com[[times]][1:NN[1],1:M0], alpha2, alpha3, y[[times]][1:NN[1]])
  Accuracy_com[times]          <- temp5[[1]]
  Precision_com[times]         <- temp5[[2]]
  Recall_com[times]            <- temp5[[3]]
  F1_com[times]                <- temp5[[4]]
  AIC_com[times]               <- temp5[[5]]
}
result_com <- round(c(mean(Accuracy_com),sd(Accuracy_com),
                      mean(Precision_com),mean(Recall_com),mean(F1_com)),6)
print(result_com)
stargazer(result_com, title = "Evaluation",   align = F, type = "latex")
######### The end of missing rate = 0 ####################
M0=10
####################################################################
######### MCCA_complete_data #########
psi.1.hat<-psi.2.hat<-psi.3.hat<-
  x.1.score<-x.2.score<-x.3.score<-
  x.1.score_NA<-x.2.score_NA<-x.3.score_NA<-
  f.hat_com<-vector('list',Iter.times )
source("MCCA_complete_data_setting2.R")
for (times in 1:Iter.times){
  same<-MCCA_complete_data_setting2(NN, N, T1, T2, T3, x_std[[times]], t.1h, t.2h, t.3h)
  psi.1.hat[[times]]           <-same[[1]]
  psi.2.hat[[times]]           <-same[[2]]
  psi.3.hat[[times]]           <-same[[3]]
  x.1.score_NA[[times]]        <-same[[4]]
  x.2.score_NA[[times]]        <-same[[5]]
  x.3.score_NA[[times]]        <-same[[6]]
  x.1.score[[times]]           <-same[[7]]
  x.2.score[[times]]           <-same[[8]]
  x.3.score[[times]]           <-same[[9]]
  f.hat_com[[times]]           <-same[[10]]
}
######### The end of MCCA_complete_data #########
####################################################################
######## Factor regression based on the CMI method #########
Accuracy_CMI<-Precision_CMI<-Recall_CMI<-F1_CMI<-AIC_CMI<-array()
x.1.score.hat<-x.2.score.hat<-x.3.score.hat<-f.hat_CMI<-vector('list',Iter.times)
source("Imputing_data_CMI_setting2.R")
for (times in 1:Iter.times) {
```

```
  temp1 <- Imputing_data_CMI_setting2(NN, y[[times]], x.1.score[[times]], x.2.score[[times]], x.3.score[[times]])
  x.1.score.hat[[times]]      <- temp1[[1]]
  x.2.score.hat[[times]]      <- temp1[[2]]
  x.3.score.hat[[times]]      <- temp1[[3]]
}
source("Construct_factor_CCA.R")
for (times in 1:Iter.times){
  temp2 <- Construct_factor_CCA(x_std[[times]], psi.1.hat[[times]], psi.2.hat[[times]],psi.3.hat[[times]],
                               x.1.score.hat[[times]], x.2.score.hat[[times]], x.3.score.hat[[times]])
  f.hat_CMI[[times]]           <- temp2[[1]]
}
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_CCA(Re(f.hat_CMI[[times]][,1:M0]), alpha2, alpha3, y[[times]])
  Accuracy_CMI[times]          <- temp3[[1]]
  Precision_CMI[times]         <- temp3[[2]]
  Recall_CMI[times]            <- temp3[[3]]
  F1_CMI[times]                <- temp3[[4]]
  AIC_CMI[times]               <- temp3[[5]]
}
######## The end of factor regression based on the CMI method #########
####################################################################
######## Factor regression based on the MBI method #########
Accuracy_MBI<-Precision_MBI<-Recall_MBI<-F1_MBI<-AIC_MBI<-array()
x.1.score.hat<-x.2.score.hat<-x.3.score.hat<-f.hat_MBI<-vector('list',Iter.times)
source("Imputing_data_MBI_setting2.R")
for (times in 1:Iter.times) {
  temp1 <- Imputing_data_MBI_setting2(NN, y[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]], x.3.score_NA[[times]])
  x.1.score.hat[[times]]      <- temp1[[1]]
  x.2.score.hat[[times]]      <- temp1[[2]]
  x.3.score.hat[[times]]      <- temp1[[3]]
}
source("Construct_factor_CCA.R")
for (times in 1:Iter.times){
  temp2 <- Construct_factor_CCA(x_std[[times]], psi.1.hat[[times]], psi.2.hat[[times]],psi.3.hat[[times]],
                               x.1.score.hat[[times]], x.2.score.hat[[times]], x.3.score.hat[[times]])
  f.hat_MBI[[times]]          <- temp2[[1]]
}
source("Factor_regression_Imputed_CCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_CCA(Re(f.hat_MBI[[times]][,1:M0]), alpha2, alpha3, y[[times]])
  Accuracy_MBI[times]         <- temp3[[1]]
  Precision_MBI[times]        <- temp3[[2]]
  Recall_MBI[times]           <- temp3[[3]]
  F1_MBI[times]               <- temp3[[4]]
  AIC_MBI[times]              <- temp3[[5]]
}
######## The end of factor regression based on the MBI method #########
#result----
result <- round(rbind(
c(mean(Accuracy_CMI),sd(Accuracy_CMI),mean(Precision_CMI),mean(Recall_CMI),mean(F1_CMI)),
c(mean(Accuracy_MBI),sd(Accuracy_MBI),mean(Precision_MBI),mean(Recall_MBI),mean(F1_MBI))),4)
print(result)
```

```
stargazer(result, title = "Evaluation",    align = F, type = "latex")
### Module 2: setting_2(PCA) ###
library(stargazer)
library(funData)
library(caret)
library(MASS)
library(nnet)
N<-300                ## The sample size
T11<-100              ## Image x(1): the sample number of x-axis
T12<-50               ## Image x(1): the sample number of y-axis
T1<-T11*T12           ## The number of sample point of Image x(1)
M11<-5                ## The number of PC scores of x(1) along x-axis
M12<-5                ## The number of PC scores of x(1) along y-axis
M1<-M11*M12           ## The number of PC scores of x(1)
T2<-200               ## The number of sample point of Image x(2)
M2<-25                ## The number of PC scores of x(2)
T3<-200               ## The number of sample point of Image x(3)
M3<-25                ## The number of PC scores of x(3)
k1=10                 ## The number of truncated PC scores of x(1) in Ma's paper
k2=10                 ## The number of truncated PC scores of x(2) in Ma's paper
k3=10                 ## The number of truncated PC scores of x(3) in Ma's paper
gamma1<-0.3           ## The coefficient in generating MFPC curves
gamma2<-0.3
## Regression coefficient vector
alpha1<-rep(0,10)
alpha2<-2*c(0.972,0.734,0.691,0.541,0.480,0.424,0.331,0.271,0.123,0.0405)
alpha3<-4*c(0.934,0.903,0.815,0.604,0.517,0.447,0.392,0.370,0.345,0.3)
M<-length(alpha1)           ## The number of mulrivariate truncated PC scores in Happ's paper
sd <- 0.2                   ## The standard deviation of error in the regression model
## miss_ratio<-0 ##
gam1.1 = rep(1, (T1+T2+T3))
gam1.2 = gam1.3 = gam1.4 = gam1.5 = gam1.6 = gam1.7 = rep(0, (T1+T2+T3))
gam2.1 = rep(1, (T1+T2+T3))
gam2.2 = gam2.3 = gam2.4 = gam2.5 = gam2.6 = gam2.7 = rep(0, (T1+T2+T3))
gam3.1 = rep(1, (T1+T2+T3))
gam3.2 = gam3.3 = gam3.4 = gam3.5 = gam3.6 = gam3.7 = rep(0, (T1+T2+T3))
## miss_ratio<-0.2 ##
gam1.1 = rep(1, (T1+T2+T3))
gam1.2 = gam1.3 = gam1.4 = gam1.5 = gam1.6 = gam1.7 = rep(7/153, (T1+T2+T3))
gam2.1 = rep(1, (T1+T2+T3))
gam2.2 = gam2.3 = gam2.4 = gam2.5 = gam2.6 = gam2.7 = rep(1/54, (T1+T2+T3))
gam3.1 = rep(1, (T1+T2+T3))
gam3.2 = gam3.3 = gam3.4 = gam3.5 = gam3.6 = gam3.7 = rep(7/153, (T1+T2+T3))
## miss_ratio<-0.6 ##
gam1.1 = rep(1, (T1+T2+T3))
gam1.2 = gam1.3 = gam1.4 = gam1.5 = gam1.6 = gam1.7 = rep(7/23, (T1+T2+T3))
gam2.1 = rep(1, (T1+T2+T3))
gam2.2 = gam2.3 = gam2.4 = gam2.5 = gam2.6 = gam2.7 = rep(1/14, (T1+T2+T3))
gam3.1 = rep(1, (T1+T2+T3))
gam3.2 = gam3.3 = gam3.4 = gam3.5 = gam3.6 = gam3.7 = rep(7/23, (T1+T2+T3))
## miss_ratio<-0.9 ##
gam1.1 = rep(1, (T1+T2+T3))
gam1.2 = gam1.3 = gam1.4 = gam1.5 = gam1.6 = gam1.7 = rep(21/4, (T1+T2+T3))
```

```
gam2.1 = rep(1, (T1+T2+T3))
gam2.2 = gam2.3 = gam2.4 = gam2.5 = gam2.6 = gam2.7 = rep(3/22, (T1+T2+T3))
gam3.1 = rep(1, (T1+T2+T3))
gam3.2 = gam3.3 = gam3.4 = gam3.5 = gam3.6 = gam3.7 = rep(21/4, (T1+T2+T3))
Iter.times<-300
setwd("C:/Users/pc/Desktop/Logistic/Setting2(MNAR)/")
####################################################################
######   Generating the eigenfunctions of setting 3 ######
source("generate_eigenfunction_setting2.R")
temp1 <- generate_eigenfunction_setting2(N, M11, T11, M12, T12, M2, T2, M3, T3)
t.11            <- temp1[[1]]
t.12            <- temp1[[2]]
t.2             <- temp1[[3]]
t.3             <- temp1[[4]]
phi.1           <- temp1[[5]]
phi.2           <- temp1[[6]]
phi.3           <- temp1[[7]]
t.1h            <- temp1[[8]]
t.2h            <- temp1[[9]]
t.3h            <- temp1[[10]]
#######   The end of Generating the eigenfunctions of setting 3 ######
####################################################################
######## generating data ####################
psi.1_std<-psi.2_std<-psi.3_std<-rho_std<-rho_std_com<-y<-x_std<-vector('list',Iter.times)
source("generate_data_setting2.R")
for (times in 1:Iter.times) {
  temp1 <- generate_data_setting2(phi.1, phi.2, phi.3, alpha1, alpha2, alpha3, sd, gamma1, gamma2)
  rho_std[[times]]      <- temp1[[1]]
  x_std[[times]]        <- temp1[[2]]
  y[[times]]            <- temp1[[3]]
  psi.1_std[[times]]    <- temp1[[4]]
  psi.2_std[[times]]    <- temp1[[5]]
  psi.3_std[[times]]    <- temp1[[6]]
}
######## The end of generating data ########
####################################################################
######## generating missing pattern ####################
source("generate_missing_pattern.R")
for (times in 1:Iter.times){
  temp2 <- generate_missing_pattern(rho_std[[times]], x_std[[times]], y[[times]])
  NN                    <- temp2[[1]]
  rho_std[[times]]      <- temp2[[2]]
  x_std[[times]]        <- temp2[[3]]
  y[[times]]            <- temp2[[4]]
}
######## The end of generating missing pattern ########
M0=10
####################################################################
######### missing rate = 0 ###################
source("MFPCA_setting2.R")
for (times in 1:Iter.times){
  temp2 <- MFPCA_setting2(x_std[[times]], k1, k2, k3, t.1h, t.2h, t.3h)
  rho_std_com[[times]]    <- temp2[[1]]
```

```
}
Accuracy_com<-Precision_com<-Recall_com<-F1_com<-AIC_com<-mse_alpha2<-mse_alpha3<-array()
alpha2.hat<-alpha3.hat<-vector('list',Iter.times )
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
  temp5 <- Factor_regression_Imputed_PCA(rho_std_com[[times]], alpha2, alpha3, y[[times]][1:NN[1]])
  Accuracy_com[times]            <- temp5[[1]]
  Precision_com[times]           <- temp5[[2]]
  Recall_com[times]              <- temp5[[3]]
  F1_com[times]                  <- temp5[[4]]
  AIC_com[times]                 <- temp5[[5]]
  alpha2.hat[[times]]            <- temp5[[6]]
  alpha3.hat[[times]]            <- temp5[[7]]
  mse_alpha2[times]              <- temp5[[8]]
  mse_alpha3[times]              <- temp5[[9]]
}
result_com <- round(c(mean(Accuracy_com),sd(Accuracy_com),
                      mean(Precision_com),mean(Recall_com),mean(F1_com),
                      mean(mse_alpha2),sd(mse_alpha2),
                      mean(mse_alpha3),sd(mse_alpha3)),6)
print(result_com)
stargazer(result_com, title = "Evaluation",   align = F, type = "latex")
######### The end of missing rate = 0 ####################
M0=10
####################################################################
######### MFPCA_complete_data #########
psi.1.hat<-psi.2.hat<-psi.3.hat<-
  x.1.score<-x.2.score<-x.3.score<-
  x.1.score_NA<-x.2.score_NA<-x.3.score_NA<-
  rho.hat_std_com<-vector('list',Iter.times )
source("MFPCA_complete_data_setting2.R")
for (times in 1:Iter.times){
  same<-MFPCA_complete_data_setting2(NN, N, T1, T2, T3, x_std[[times]], t.1h, t.2h, t.3h)
  psi.1.hat[[times]]             <-same[[1]]
  psi.2.hat[[times]]             <-same[[2]]
  psi.3.hat[[times]]             <-same[[3]]
  x.1.score_NA[[times]]          <-same[[4]]
  x.2.score_NA[[times]]          <-same[[5]]
  x.3.score_NA[[times]]          <-same[[6]]
  x.1.score[[times]]             <-same[[7]]
  x.2.score[[times]]             <-same[[8]]
  x.3.score[[times]]             <-same[[9]]
  rho.hat_std_com[[times]]       <-same[[10]]
}
######### The end of MFPCA_complete_data #########
####################################################################
######## Factor regression based on the CMI method #########
Accuracy_CMI<-Precision_CMI<-Recall_CMI<-F1_CMI<-AIC_CMI<-
  mse_alpha2_CMI<-mse_alpha3_CMI<-array()
x.1.score.hat<-x.2.score.hat<-x.3.score.hat<-rho.hat_std_CMI<-
  alpha2.hat_CMI<-alpha3.hat_CMI<-vector('list',Iter.times)
source("Imputing_data_CMI_setting2.R")
for (times in 1:Iter.times) {
```

```
  temp1 <- Imputing_data_CMI_setting2(NN, y[[times]], x.1.score[[times]], x.2.score[[times]], x.3.score[[times]])
  x.1.score.hat[[times]]      <- temp1[[1]]
  x.2.score.hat[[times]]      <- temp1[[2]]
  x.3.score.hat[[times]]      <- temp1[[3]]
}
source("Construct_factor_PCA.R")
for (times in 1:Iter.times){
  temp2 <- Construct_factor_PCA(x_std[[times]], psi.1.hat[[times]], psi.2.hat[[times]], psi.3.hat[[times]],
                                x.1.score.hat[[times]], x.2.score.hat[[times]], x.3.score.hat[[times]])
  rho.hat_std_CMI[[times]]     <- temp2[[1]]
}
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_PCA(rho.hat_std_CMI[[times]], alpha2, alpha3, y[[times]])
  Accuracy_CMI[times]          <- temp3[[1]]
  Precision_CMI[times]         <- temp3[[2]]
  Recall_CMI[times]            <- temp3[[3]]
  F1_CMI[times]                <- temp3[[4]]
  AIC_CMI[times]               <- temp3[[5]]
  alpha2.hat_CMI[[times]]      <- temp3[[6]]
  alpha3.hat_CMI[[times]]      <- temp3[[7]]
  mse_alpha2_CMI[times]        <- temp3[[8]]
  mse_alpha3_CMI[times]        <- temp3[[9]]
}
######## The end of factor regression based on the CMI method #########
###################################################################
######## Factor regression based on the MBI method #########
Accuracy_MBI<-Precision_MBI<-Recall_MBI<-F1_MBI<-AIC_MBI<-
  mse_alpha2_MBI<-mse_alpha3_MBI<-array()
x.1.score.hat<-x.2.score.hat<-x.3.score.hat<-rho.hat_std_MBI<-
  alpha2.hat_MBI<-alpha3.hat_MBI<-vector('list',Iter.times)
source("Imputing_data_MBI_setting2.R")
for (times in 1:Iter.times) {
  temp1 <- Imputing_data_MBI_setting2(NN, y[[times]], x.1.score_NA[[times]], x.2.score_NA[[times]], x.3.score_NA[[times]])
  x.1.score.hat[[times]]      <- temp1[[1]]
  x.2.score.hat[[times]]      <- temp1[[2]]
  x.3.score.hat[[times]]      <- temp1[[3]]
}
source("Construct_factor_PCA.R")
for (times in 1:Iter.times){
  temp2 <- Construct_factor_PCA(x_std[[times]], psi.1.hat[[times]], psi.2.hat[[times]], psi.3.hat[[times]],
                                x.1.score.hat[[times]], x.2.score.hat[[times]], x.3.score.hat[[times]])
  rho.hat_std_MBI[[times]]     <- temp2[[1]]
}
source("Factor_regression_Imputed_PCA.R")
for (times in 1:Iter.times){
  temp3 <- Factor_regression_Imputed_PCA(rho.hat_std_MBI[[times]], alpha2, alpha3, y[[times]])
  Accuracy_MBI[times]          <- temp3[[1]]
  Precision_MBI[times]         <- temp3[[2]]
  Recall_MBI[times]            <- temp3[[3]]
  F1_MBI[times]                <- temp3[[4]]
  AIC_MBI[times]               <- temp3[[5]]
  alpha2.hat_MBI[[times]]      <- temp3[[6]]
```

```
   alpha3.hat_MBI[[times]]       <- temp3[[7]]
   mse_alpha2_MBI[times]          <- temp3[[8]]
   mse_alpha3_MBI[times]          <- temp3[[9]]
}
######## The end of factor regression based on the MBI method #########
#result----
result <- round(rbind(
c(mean(Accuracy_CMI),sd(Accuracy_CMI),mean(Precision_CMI),mean(Recall_CMI),mean(F1_CMI),
mean(mse_alpha2_CMI),sd(mse_alpha2_CMI),mean(mse_alpha3_CMI),sd(mse_alpha3_CMI)),
   c(mean(Accuracy_MBI),sd(Accuracy_MBI),mean(Precision_MBI),mean(Recall_MBI),mean(F1_MBI),
      mean(mse_alpha2_MBI),sd(mse_alpha2_MBI),mean(mse_alpha3_MBI),sd(mse_alpha3_MBI))),4)
print(result)
stargazer(result, title = "Evaluation",    align = F, type = "latex")
```

The following is the function "generate_missing_pattern" used in the main program. The rest of the functions used are the same as in Setting2(MCAR).

The function "generate_missing_pattern" means to generate data according to the missing data mechanism of MNAR.

```
### generate_missing_pattern ###
generate_missing_pattern<-function(rho, x, y){
   id1 = which(y == 1)
   id2 = which(y == 2)
   id3 = which(y == 3)
   x.1 = x[,1:T1]
   x.2 = x[,(T1+1):(T1+T2)]
   x.3 = x[,(T1+T2+1):(T1+T2+T3)]
   ## Divide the missing block when y = 1 ##
   g1.1 = x.1[id1,]%*%gam1.1[1:T1]/T1 + x.2[id1,]%*%gam1.1[(T1+1):(T1+T2)]/T2 + x.3[id1,]%*%gam1.1[(T1+T2+1):(T1+T2+T3)]/T3
   g1.2 = x.1[id1,]%*%gam1.2[1:T1]/T1 + x.2[id1,]%*%gam1.2[(T1+1):(T1+T2)]/T2 + x.3[id1,]%*%gam1.2[(T1+T2+1):(T1+T2+T3)]/T3
   g1.3 = x.1[id1,]%*%gam1.3[1:T1]/T1 + x.2[id1,]%*%gam1.3[(T1+1):(T1+T2)]/T2 + x.3[id1,]%*%gam1.3[(T1+T2+1):(T1+T2+T3)]/T3
   g1.4 = x.1[id1,]%*%gam1.4[1:T1]/T1 + x.2[id1,]%*%gam1.4[(T1+1):(T1+T2)]/T2 + x.3[id1,]%*%gam1.4[(T1+T2+1):(T1+T2+T3)]/T3
   g1.5 = x.1[id1,]%*%gam1.5[1:T1]/T1 + x.2[id1,]%*%gam1.5[(T1+1):(T1+T2)]/T2 + x.3[id1,]%*%gam1.5[(T1+T2+1):(T1+T2+T3)]/T3
   g1.6 = x.1[id1,]%*%gam1.6[1:T1]/T1 + x.2[id1,]%*%gam1.6[(T1+1):(T1+T2)]/T2 + x.3[id1,]%*%gam1.6[(T1+T2+1):(T1+T2+T3)]/T3
   g1.7 = x.1[id1,]%*%gam1.7[1:T1]/T1 + x.2[id1,]%*%gam1.7[(T1+1):(T1+T2)]/T2 + x.3[id1,]%*%gam1.7[(T1+T2+1):(T1+T2+T3)]/T3
   p1.1 = g1.1 / (g1.1+g1.2+g1.3+g1.4+g1.5+g1.6+g1.7)
   p1.2 = g1.2 / (g1.1+g1.2+g1.3+g1.4+g1.5+g1.6+g1.7)
   p1.3 = g1.3 / (g1.1+g1.2+g1.3+g1.4+g1.5+g1.6+g1.7)
   p1.4 = g1.4 / (g1.1+g1.2+g1.3+g1.4+g1.5+g1.6+g1.7)
   p1.5 = g1.5 / (g1.1+g1.2+g1.3+g1.4+g1.5+g1.6+g1.7)
   p1.6 = g1.6 / (g1.1+g1.2+g1.3+g1.4+g1.5+g1.6+g1.7)
   p1.7 = g1.7 / (g1.1+g1.2+g1.3+g1.4+g1.5+g1.6+g1.7)
   p1 = cbind(p1.1, p1.2, p1.3, p1.4, p1.5, p1.6, p1.7)
   miss.1 = c()
   for(i in 1:length(id1)){
      miss0.1 = which.max(rmultinom(1, size = 1, prob = p1[i,]))
      miss.1 = c(miss.1, miss0.1)
   }
```

```
id1.1 = which(miss.1 == 1)
id1.2 = which(miss.1 == 2)
id1.3 = which(miss.1 == 3)
id1.4 = which(miss.1 == 4)
id1.5 = which(miss.1 == 5)
id1.6 = which(miss.1 == 6)
id1.7 = which(miss.1 == 7)
## The end of dividing the missing block when y = 1 ##
## Divide the missing block when y = 2 ##
g2.1 = x.1[id2,]%*%gam2.1[1:T1]/T1 + x.2[id2,]%*%gam2.1[(T1+1):(T1+T2)]/T2 + x.3[id2,]%*%gam2.1[(T1+T2+1):(T1+T2+T3)]/T3
g2.2 = x.1[id2,]%*%gam2.2[1:T1]/T1 + x.2[id2,]%*%gam2.2[(T1+1):(T1+T2)]/T2 + x.3[id2,]%*%gam2.2[(T1+T2+1):(T1+T2+T3)]/T3
g2.3 = x.1[id2,]%*%gam2.3[1:T1]/T1 + x.2[id2,]%*%gam2.3[(T1+1):(T1+T2)]/T2 + x.3[id2,]%*%gam2.3[(T1+T2+1):(T1+T2+T3)]/T3
g2.4 = x.1[id2,]%*%gam2.4[1:T1]/T1 + x.2[id2,]%*%gam2.4[(T1+1):(T1+T2)]/T2 + x.3[id2,]%*%gam2.4[(T1+T2+1):(T1+T2+T3)]/T3
g2.5 = x.1[id2,]%*%gam2.5[1:T1]/T1 + x.2[id2,]%*%gam2.5[(T1+1):(T1+T2)]/T2 + x.3[id2,]%*%gam2.5[(T1+T2+1):(T1+T2+T3)]/T3
g2.6 = x.1[id2,]%*%gam2.6[1:T1]/T1 + x.2[id2,]%*%gam2.6[(T1+1):(T1+T2)]/T2 + x.3[id2,]%*%gam2.6[(T1+T2+1):(T1+T2+T3)]/T3
g2.7 = x.1[id2,]%*%gam2.7[1:T1]/T1 + x.2[id2,]%*%gam2.7[(T1+1):(T1+T2)]/T2 + x.3[id2,]%*%gam2.7[(T1+T2+1):(T1+T2+T3)]/T3
p2.1 = g2.1 / (g2.1+g2.2+g2.3+g2.4+g2.5+g2.6+g2.7)
p2.2 = g2.2 / (g2.1+g2.2+g2.3+g2.4+g2.5+g2.6+g2.7)
p2.3 = g2.3 / (g2.1+g2.2+g2.3+g2.4+g2.5+g2.6+g2.7)
p2.4 = g2.4 / (g2.1+g2.2+g2.3+g2.4+g2.5+g2.6+g2.7)
p2.5 = g2.5 / (g2.1+g2.2+g2.3+g2.4+g2.5+g2.6+g2.7)
p2.6 = g2.6 / (g2.1+g2.2+g2.3+g2.4+g2.5+g2.6+g2.7)
p2.7 = g2.7 / (g2.1+g2.2+g2.3+g2.4+g2.5+g2.6+g2.7)
p2 = cbind(p2.1, p2.2, p2.3, p2.4, p2.5, p2.6, p2.7)
miss.2 = c()
for(i in 1:length(id2)){
  miss0.2 = which.max(rmultinom(1, size = 1, prob = p2[i,]))
  miss.2 = c(miss.2, miss0.2)
}
id2.1 = which(miss.2 == 1)
id2.2 = which(miss.2 == 2)
id2.3 = which(miss.2 == 3)
id2.4 = which(miss.2 == 4)
id2.5 = which(miss.2 == 5)
id2.6 = which(miss.2 == 6)
id2.7 = which(miss.2 == 7)
## The end of dividing the missing block when y = 2 ##
## Divide the missing block when y = 3 ##
g3.1 = x.1[id3,]%*%gam3.1[1:T1]/T1 + x.2[id3,]%*%gam3.1[(T1+1):(T1+T2)]/T2 + x.3[id3,]%*%gam3.1[(T1+T2+1):(T1+T2+T3)]/T3
g3.2 = x.1[id3,]%*%gam3.2[1:T1]/T1 + x.2[id3,]%*%gam3.2[(T1+1):(T1+T2)]/T2 + x.3[id3,]%*%gam3.2[(T1+T2+1):(T1+T2+T3)]/T3
g3.3 = x.1[id3,]%*%gam3.3[1:T1]/T1 + x.2[id3,]%*%gam3.3[(T1+1):(T1+T2)]/T2 + x.3[id3,]%*%gam3.3[(T1+T2+1):(T1+T2+T3)]/T3
g3.4 = x.1[id3,]%*%gam3.4[1:T1]/T1 + x.2[id3,]%*%gam3.4[(T1+1):(T1+T2)]/T2 + x.3[id3,]%*%gam3.4[(T1+T2+1):(T1+T2+T3)]/T3
```

```
  g3.5 = x.1[id3,]%*%gam3.5[1:T1]/T1 + x.2[id3,]%*%gam3.5[(T1+1):(T1+T2)]/T2 + x.3[id3,]%*%gam3.5[(T1+T2+1):(T1+T2+T3)]/T3
  g3.6 = x.1[id3,]%*%gam3.6[1:T1]/T1 + x.2[id3,]%*%gam3.6[(T1+1):(T1+T2)]/T2 + x.3[id3,]%*%gam3.6[(T1+T2+1):(T1+T2+T3)]/T3
  g3.7 = x.1[id3,]%*%gam3.7[1:T1]/T1 + x.2[id3,]%*%gam3.7[(T1+1):(T1+T2)]/T2 + x.3[id3,]%*%gam3.7[(T1+T2+1):(T1+T2+T3)]/T3
  p3.1 = g3.1 / (g3.1+g3.2+g3.3+g3.4+g3.5+g3.6+g3.7)
  p3.2 = g3.2 / (g3.1+g3.2+g3.3+g3.4+g3.5+g3.6+g3.7)
  p3.3 = g3.3 / (g3.1+g3.2+g3.3+g3.4+g3.5+g3.6+g3.7)
  p3.4 = g3.4 / (g3.1+g3.2+g3.3+g3.4+g3.5+g3.6+g3.7)
  p3.5 = g3.5 / (g3.1+g3.2+g3.3+g3.4+g3.5+g3.6+g3.7)
  p3.6 = g3.6 / (g3.1+g3.2+g3.3+g3.4+g3.5+g3.6+g3.7)
  p3.7 = g3.7 / (g3.1+g3.2+g3.3+g3.4+g3.5+g3.6+g3.7)
  p3 = cbind(p3.1, p3.2, p3.3, p3.4, p3.5, p3.6, p3.7)
  miss.3 = c()
  for(i in 1:length(id3)){
    miss0.3 = which.max(rmultinom(1, size = 1, prob = p3[i,]))
    miss.3 = c(miss.3, miss0.3)
  }
  id3.1 = which(miss.3 == 1)
  id3.2 = which(miss.3 == 2)
  id3.3 = which(miss.3 == 3)
  id3.4 = which(miss.3 == 4)
  id3.5 = which(miss.3 == 5)
  id3.6 = which(miss.3 == 6)
  id3.7 = which(miss.3 == 7)
  ## The end of dividing the missing block when y = 3 ##
  ## Divide the missing block ##
  id.1 = c(id1[id1.1], id2[id2.1], id3[id3.1])
  id.2 = c(id1[id1.2], id2[id2.2], id3[id3.2])
  id.3 = c(id1[id1.3], id2[id2.3], id3[id3.3])
  id.4 = c(id1[id1.4], id2[id2.4], id3[id3.4])
  id.5 = c(id1[id1.5], id2[id2.5], id3[id3.5])
  id.6 = c(id1[id1.6], id2[id2.6], id3[id3.6])
  id.7 = c(id1[id1.7], id2[id2.7], id3[id3.7])
  NN = c()
  NN[1] = length(id.1)
  NN[2] = length(id.2)
  NN[3] = length(id.3)
  NN[4] = length(id.4)
  NN[5] = length(id.5)
  NN[6] = length(id.6)
  NN[7] = length(id.7)
  y.new = c(y[id.1], y[id.2], y[id.3], y[id.4], y[id.5], y[id.6], y[id.7])
  x.new = rbind(x[id.1,], x[id.2,], x[id.3,], x[id.4,], x[id.5,], x[id.6,], x[id.7,])
  rho.new = rbind(rho[id.1,], rho[id.2,], rho[id.3,], rho[id.4,], rho[id.5,], rho[id.6,], rho[id.7,])
  ## The end of dividing the missing block ##
  return(list(NN, rho.new, x.new, y.new))
}
```

5. Empirical analysis

The following program uses the FRI-CCA method for 4-class and 3-class classification tasks.

```
### Accuracy(FRI-CCA) ###
```

```
library(data.table)
library(tidyverse)
library(caret)
library(pROC)
library(nnet)
######### Read data #########
rho.hat_PET = fread("C:/Users/pc/Desktop/Logistic/Empirical analysis/2-source(0.95)/rho.hat_PET.csv") %>%
as.matrix() %>% .[,1:100]
rho.hat_MRI = fread("C:/Users/pc/Desktop/Logistic/Empirical analysis/2-source(0.95)/rho.hat_MRI.csv") %>%
as.matrix() %>% .[,1:235]
MMSE = fread("C:/Users/pc/Desktop/Logistic/Empirical analysis/2-source(0.95)/MMSE.csv", header = F) %>% .$V1
researchGroup = fread("C:/Users/pc/Desktop/Logistic/Empirical analysis/2-source(0.95)/researchGroup.csv", header =
F) %>% .$V1
# researchGroup[researchGroup == "EMCI"] = "MCI"
# researchGroup[researchGroup == "LMCI"] = "MCI"
M1 = rho.hat_PET %>% ncol()
M2 = rho.hat_MRI %>% ncol()
NN = c(327, 315, 72); N = sum(NN); M1; M2
### Constructing missing principal components ###
rho.hat_PET_NA = matrix(NA, N, M1)
rho.hat_PET_NA[1:sum(NN[1:2]),] = rho.hat_PET
rho.hat_MRI_NA = matrix(NA, N, M2)
rho.hat_MRI_NA[1:NN[1],] = rho.hat_MRI[1:NN[1],]
rho.hat_MRI_NA[(sum(NN[1:2])+1):N,] = rho.hat_MRI[(NN[1]+1):(NN[1]+NN[3]),]
### MBI imputation ###
rho.hat_MRI_hat = matrix(0,NN[2],M2)
for (i in 1:M2) {
  fit = lm(rho.hat_MRI_NA[1:NN[1],i]~., data = as.data.frame(rho.hat_PET_NA[1:NN[1],]))
  rho.hat_MRI_hat[,i] = cbind(rep(1,NN[2]),rho.hat_PET_NA[(NN[1]+1):sum(NN[1:2]),])%*%fit$coefficients
}
rho.hat_MRI_NA[(NN[1]+1):sum(NN[1:2]),] = rho.hat_MRI_hat
rho.hat_PET_hat = matrix(0,NN[3],M1)
for (i in 1:M1) {
  fit = lm(rho.hat_PET_NA[1:NN[1],i]~., data = as.data.frame(rho.hat_MRI_NA[1:NN[1],]))
  rho.hat_PET_hat[,i] = cbind(rep(1,NN[3]),rho.hat_MRI_NA[(sum(NN[1:2])+1):N,])%*%fit$coefficients
}
rho.hat_PET_NA[(sum(NN[1:2])+1):N,] = rho.hat_PET_hat
### CMI imputation ###
# Zero-mean imputation as initial value #
rho.hat_MRI_hat<-t(matrix(apply(rho.hat_MRI_NA[c(1:NN[1],(sum(NN[1:2])+1):N),], 2, mean), M2, NN[2]))
rho.hat_MRI_NA[(NN[1]+1):sum(NN[1:2]),] = rho.hat_MRI_hat
rho.hat_PET_hat<-t(matrix(apply(rho.hat_PET_NA[1:sum(NN[1:2]),], 2, mean), M1, NN[3]))
rho.hat_PET_NA[(sum(NN[1:2])+1):N,] = rho.hat_PET_hat
# Iterative regression #
n_iter = 50
for(j in 1:n_iter)
{
  rho.hat_MRI_hat = matrix(0,NN[2],M2)
  for (i in 1:M2) {
    fit = lm(rho.hat_MRI_NA[,i]~., data = as.data.frame(rho.hat_PET_NA))
    rho.hat_MRI_hat[,i] = cbind(rep(1,NN[2]),rho.hat_PET_NA[(NN[1]+1):sum(NN[1:2]),])%*%fit$coefficients
  }
  rho.hat_MRI_NA[(NN[1]+1):sum(NN[1:2]),] = rho.hat_MRI_hat
```

```
    rho.hat_PET_hat = matrix(0,NN[3],M1)
    for (i in 1:M1) {
      fit = lm(rho.hat_PET_NA[,i]~., data = as.data.frame(rho.hat_MRI_NA))
      rho.hat_PET_hat[,i] = cbind(rep(1,NN[3]),rho.hat_MRI_NA[(sum(NN[1:2])+1):N,])%*%fit$coefficients
    }
    rho.hat_PET_NA[(sum(NN[1:2])+1):N,] = rho.hat_PET_hat
}
### f.hat_CMI as factor ###
Z1 = rho.hat_PET_NA
Z2 = rho.hat_MRI_NA
Omega1 = Z1%*%solve(t(Z1)%*%Z1)%*%t(Z1)
Omega2 = Z2%*%solve(t(Z2)%*%Z2)%*%t(Z2)
f.hat_CMI = eigen(Omega1 + Omega2)$vectors
###### rep is the number of repetitions ###
###### prob is the training rate              ###
###### M is the number of factors             ###
###### N is the sample size                     ###
M = c(1:max(M1,M2))
rep = 100; prob = 0.8
Accuracy = F1 = Sensitivity = Specificity = Precision = Recall = Mm = c()
for(l in 1:rep){
    ### divide the dataset ###
    folds = list(); accuracy_test = YY = YT = c(); iter = 5
    f.hat_CMI_train = f.hat_CMI_test = matrix(nrow = 0, ncol = N)
    for(i in 1:iter){
      folds[[i]] = createDataPartition(researchGroup, p = prob, list = FALSE)
      f.hat_CMI_train0 = f.hat_CMI %>% .[folds[[i]],]
      f.hat_CMI_train = rbind(f.hat_CMI_train, f.hat_CMI_train0)
      f.hat_CMI_test0 = f.hat_CMI %>% .[-folds[[i]],]
      f.hat_CMI_test = rbind(f.hat_CMI_test, f.hat_CMI_test0)
      YY0 = c(researchGroup %>% .[folds[[i]]])
      YY = c(YY, YY0)

      YT0 = c(researchGroup %>% .[-folds[[i]]])
      YT = c(YT, YT0)
    }
    AIC_model = c()
    for(m in 1:length(M)){
      ### train data ###
      Group = YY
      data = data.frame(cbind(f.hat_CMI_train[,1:M[m]]), Group)
      data$Group = factor(data$Group)
      mult.model = multinom(Group ~ ., data = data, MaxNWts = 10000)
      AIC_model[m] = mult.model$AIC
    }
    m = which.min(AIC_model)
    Mm[l] = M[m]
    ### train data ###
    Group = YY
    data = data.frame(cbind(f.hat_CMI_train[,1:M[m]]), Group)
    data$Group = factor(data$Group)
    mult.model = multinom(Group ~ ., data = data, MaxNWts = 10000)
    predlab = predict(mult.model, newdata = data, type = "class")
```

```
  confum_train = confusionMatrix(data = predlab, reference = data$Group, mode = "everything")
  accuracy_train = confum_train$overall[1]
  ### test data ###
  Group = YT
  data = data.frame(cbind(f.hat_CMI_test[,1:M[m]]), Group)
  data$Group = factor(data$Group)
  predlab = predict(mult.model, newdata = data, type = "class")
  confum_test = confusionMatrix(data = predlab, reference = data$Group, mode = "everything")
  accuracy_test = confum_test$overall[1]
  confumat_test = as.data.frame(confum_test$table)
  ggplot(confumat_test, aes(x = Reference, y = Prediction)) +
    geom_tile(aes(fill = Freq)) +
    geom_text(aes(label = Freq)) +
    scale_fill_gradient(low = "steelblue", high = "lightgreen", guide = "colorbar") +
    ggtitle("mlogit")
  summary_test = multiClassSummary(
    data.frame(obs = data$Group, pred = predlab), lev = levels(data$Group))
  Accuracy[l] = summary_test[1]
  F1[l] = summary_test[3]
  Precision[l] = summary_test[8]
  Recall[l] = summary_test[9]
}
Accuracy_mean = mean(Accuracy)
Precision_mean = mean(Precision)
Recall_mean = mean(Recall)
F1_mean = mean(F1)
print(c(Accuracy_mean, Precision_mean, Recall_mean, F1_mean))
```

The following program uses the FRI-PCA method for 4-class and 3-class classification tasks.

```
### Accuracy(FRI-PCA) ###
library(data.table)
library(tidyverse)
library(caret)
library(pROC)
library(nnet)
######### Read data #########
rho.hat_PET = fread("C:/Users/pc/Desktop/Logistic/Empirical analysis/2-source(0.95)/rho.hat_PET.csv") %>% as.matrix() %>% .[,1:100]
rho.hat_MRI = fread("C:/Users/pc/Desktop/Logistic/Empirical analysis/2-source(0.95)/rho.hat_MRI.csv") %>% as.matrix() %>% .[,1:235]
MMSE = fread("C:/Users/pc/Desktop/Logistic/Empirical analysis/2-source(0.95)/MMSE.csv", header = F) %>% .$V1
researchGroup = fread("C:/Users/pc/Desktop/Logistic/Empirical analysis/2-source(0.95)/researchGroup.csv", header = F) %>% .$V1
# researchGroup[researchGroup == "EMCI"] = "MCI"
# researchGroup[researchGroup == "LMCI"] = "MCI"
M1 = rho.hat_PET %>% ncol()
M2 = rho.hat_MRI %>% ncol()
NN = c(327, 315, 72); N = sum(NN); M1; M2
### Constructing missing principal components ###
rho.hat_PET_NA = matrix(NA, N, M1)
rho.hat_PET_NA[1:sum(NN[1:2]),] = rho.hat_PET
rho.hat_MRI_NA = matrix(NA, N, M2)
rho.hat_MRI_NA[1:NN[1],] = rho.hat_MRI[1:NN[1],]
rho.hat_MRI_NA[(sum(NN[1:2])+1):N,] = rho.hat_MRI[(NN[1]+1):(NN[1]+NN[3]),]
```

```
### MBI imputation ###
rho.hat_MRI_hat = matrix(0,NN[2],M2)
for (i in 1:M2) {
  fit = lm(rho.hat_MRI_NA[1:NN[1],i]~., data = as.data.frame(rho.hat_PET_NA[1:NN[1],]))
  rho.hat_MRI_hat[,i] = cbind(rep(1,NN[2]),rho.hat_PET_NA[(NN[1]+1):sum(NN[1:2]),])%*%fit$coefficients
}
rho.hat_MRI_NA[(NN[1]+1):sum(NN[1:2]),] = rho.hat_MRI_hat
rho.hat_PET_hat = matrix(0,NN[3],M1)
for (i in 1:M1) {
  fit = lm(rho.hat_PET_NA[1:NN[1],i]~., data = as.data.frame(rho.hat_MRI_NA[1:NN[1],]))
  rho.hat_PET_hat[,i] = cbind(rep(1,NN[3]),rho.hat_MRI_NA[(sum(NN[1:2])+1):N,])%*%fit$coefficients
}
rho.hat_PET_NA[(sum(NN[1:2])+1):N,] = rho.hat_PET_hat
### CMI imputation ###
# Zero-mean imputation as initial value #
rho.hat_MRI_hat<-t(matrix(apply(rho.hat_MRI_NA[c(1:NN[1],(sum(NN[1:2])+1):N),], 2, mean), M2, NN[2]))
rho.hat_MRI_NA[(NN[1]+1):sum(NN[1:2]),] = rho.hat_MRI_hat
rho.hat_PET_hat<-t(matrix(apply(rho.hat_PET_NA[1:sum(NN[1:2]),], 2, mean), M1, NN[3]))
rho.hat_PET_NA[(sum(NN[1:2])+1):N,] = rho.hat_PET_hat
# Iterative regression #
n_iter = 50
for(j in 1:n_iter)
{
  rho.hat_MRI_hat = matrix(0,NN[2],M2)
  for (i in 1:M2) {
    fit = lm(rho.hat_MRI_NA[,i]~., data = as.data.frame(rho.hat_PET_NA))
    rho.hat_MRI_hat[,i] = cbind(rep(1,NN[2]),rho.hat_PET_NA[(NN[1]+1):sum(NN[1:2]),])%*%fit$coefficients
  }
  rho.hat_MRI_NA[(NN[1]+1):sum(NN[1:2]),] = rho.hat_MRI_hat
  rho.hat_PET_hat = matrix(0,NN[3],M1)
  for (i in 1:M1) {
    fit = lm(rho.hat_PET_NA[,i]~., data = as.data.frame(rho.hat_MRI_NA))
    rho.hat_PET_hat[,i] = cbind(rep(1,NN[3]),rho.hat_MRI_NA[(sum(NN[1:2])+1):N,])%*%fit$coefficients
  }
  rho.hat_PET_NA[(sum(NN[1:2])+1):N,] = rho.hat_PET_hat
}
### rho.hat_CMI as factor ###
score = cbind(rho.hat_PET_NA, rho.hat_MRI_NA)
z.hat = cov(score)
rho.hat_CMI = (rho.hat_PET_NA %*% eigen(z.hat)$vec[1:M1,1:(M1+M2)] +
                  rho.hat_MRI_NA %*% eigen(z.hat)$vec[(M1+1):(M1+M2),1:(M1+M2)]) %>% scale()
###### rep is the number of repetitions ###
###### prob is the training rate        ###
###### M is the number of factors       ###
###### N is the sample size             ###
M = c(1:(M1+M2))
rep = 100; prob = 0.8
Accuracy = F1 = Sensitivity = Specificity = Precision = Recall = Mm = c()
for(l in 1:rep){
  ### divide the dataset ###
  folds = list(); accuracy_test = YY = YT = c(); iter = 5
  rho.hat_CMI_train = rho.hat_CMI_test = matrix(nrow = 0, ncol = (M1+M2))
  for(i in 1:iter){
```

```
      folds[[i]] = createDataPartition(researchGroup, p = prob, list = FALSE)
        rho.hat_CMI_train0 = rho.hat_CMI %>% .[folds[[i]],]
      rho.hat_CMI_train = rbind(rho.hat_CMI_train, rho.hat_CMI_train0)
      rho.hat_CMI_test0 = rho.hat_CMI %>% .[-folds[[i]],]
      rho.hat_CMI_test = rbind(rho.hat_CMI_test, rho.hat_CMI_test0)
      YY0 = c(researchGroup %>% .[folds[[i]]])
      YY = c(YY, YY0)
      YT0 = c(researchGroup %>% .[-folds[[i]]])
      YT = c(YT, YT0)
    }
    AIC_model = c()
    for(m in 1:length(M)){
      ### train data ###
      Group = YY
      data = data.frame(cbind(rho.hat_CMI_train[,1:M[m]]), Group)
      data$Group = factor(data$Group)
      mult.model = multinom(Group ~ ., data = data, MaxNWts = 10000)
      AIC_model[m] = mult.model$AIC
    }
    m = which.min(AIC_model)
    Mm[l] = M[m]
    ### train data ###
    Group = YY
    data = data.frame(cbind(rho.hat_CMI_train[,1:M[m]]), Group)
    data$Group = factor(data$Group)
    mult.model = multinom(Group ~ ., data = data, MaxNWts = 10000)
    predlab = predict(mult.model, newdata = data, type = "class")
    confum_train = confusionMatrix(data = predlab, reference = data$Group, mode = "everything")
    accuracy_train = confum_train$overall[1]
    ### test data ###
    Group = YT
    data = data.frame(cbind(rho.hat_CMI_test[,1:M[m]]), Group)
    data$Group = factor(data$Group)
    predlab = predict(mult.model, newdata = data, type = "class")
    confum_test = confusionMatrix(data = predlab, reference = data$Group, mode = "everything")
    accuracy_test = confum_test$overall[1]
    confumat_test = as.data.frame(confum_test$table)
    ggplot(confumat_test, aes(x = Reference, y = Prediction)) +
      geom_tile(aes(fill = Freq)) +
      geom_text(aes(label = Freq)) +
      scale_fill_gradient(low = "steelblue", high = "lightgreen", guide = "colorbar") +
      ggtitle("mlogit")
    summary_test = multiClassSummary(
      data.frame(obs = data$Group, pred = predlab), lev = levels(data$Group))
    Accuracy[l] = summary_test[1]
    F1[l] = summary_test[3]
    Precision[l] = summary_test[8]
    Recall[l] = summary_test[9]
  }
  Accuracy_mean = mean(Accuracy)
  Precision_mean = mean(Precision)
  Recall_mean = mean(Recall)
  F1_mean = mean(F1)
```

```
print(c(Accuracy_mean, Precision_mean, Recall_mean, F1_mean))
```

The following program uses the FRI-CCA method for 2-class classification tasks.

```
### Accuracy(FRI-CCA)(2-class) ###
library(data.table)
library(tidyverse)
library(caret)
library(pROC)
library(nnet)
######### Read data #########
rho.hat_PET = fread("C:/Users/pc/Desktop/Logistic/Empirical analysis/2-source(0.95)/rho.hat_PET.csv") %>% as.matrix() %>% .[,1:100]
rho.hat_MRI = fread("C:/Users/pc/Desktop/Logistic/Empirical analysis/2-source(0.95)/rho.hat_MRI.csv") %>% as.matrix() %>% .[,1:235]
MMSE = fread("C:/Users/pc/Desktop/Logistic/Empirical analysis/2-source(0.95)/MMSE.csv", header = F) %>% .$V1
researchGroup = fread("C:/Users/pc/Desktop/Logistic/Empirical analysis/2-source(0.95)/researchGroup.csv", header = F) %>% .$V1
researchGroup[researchGroup == "EMCI"] = "MCI"
researchGroup[researchGroup == "LMCI"] = "MCI"
M1 = rho.hat_PET %>% ncol()
M2 = rho.hat_MRI %>% ncol()
NN = c(327, 315, 72); N = sum(NN); M1; M2
### Constructing missing principal components ###
rho.hat_PET_NA = matrix(NA, N, M1)
rho.hat_PET_NA[1:sum(NN[1:2]),] = rho.hat_PET
rho.hat_MRI_NA = matrix(NA, N, M2)
rho.hat_MRI_NA[1:NN[1],] = rho.hat_MRI[1:NN[1],]
rho.hat_MRI_NA[(sum(NN[1:2])+1):N,] = rho.hat_MRI[(NN[1]+1):(NN[1]+NN[3]),]
### CMI imputation ###
# Zero-mean imputation as initial value #
rho.hat_MRI_hat<-t(matrix(apply(rho.hat_MRI_NA[c(1:NN[1],(sum(NN[1:2])+1):N),], 2, mean), M2, NN[2]))
rho.hat_MRI_NA[(NN[1]+1):sum(NN[1:2]),] = rho.hat_MRI_hat
rho.hat_PET_hat<-t(matrix(apply(rho.hat_PET_NA[1:sum(NN[1:2]),], 2, mean), M1, NN[3]))
rho.hat_PET_NA[(sum(NN[1:2])+1):N,] = rho.hat_PET_hat
# Iterative regression #
n_iter = 50
for(j in 1:n_iter)
{
  rho.hat_MRI_hat = matrix(0,NN[2],M2)
  for (i in 1:M2) {
    fit = lm(rho.hat_MRI_NA[,i]~., data = as.data.frame(rho.hat_PET_NA))
    rho.hat_MRI_hat[,i] = cbind(rep(1,NN[2]),rho.hat_PET_NA[(NN[1]+1):sum(NN[1:2]),])%*%fit$coefficients
  }
  rho.hat_MRI_NA[(NN[1]+1):sum(NN[1:2]),] = rho.hat_MRI_hat
  rho.hat_PET_hat = matrix(0,NN[3],M1)
  for (i in 1:M1) {
    fit = lm(rho.hat_PET_NA[,i]~., data = as.data.frame(rho.hat_MRI_NA))
    rho.hat_PET_hat[,i] = cbind(rep(1,NN[3]),rho.hat_MRI_NA[(sum(NN[1:2])+1):N,])%*%fit$coefficients
  }
  rho.hat_PET_NA[(sum(NN[1:2])+1):N,] = rho.hat_PET_hat
}
### f.hat_CMI as factor ###
Z1 = rho.hat_PET_NA
```

```
Z2 = rho.hat_MRI_NA
Omega1 = Z1%*%solve(t(Z1)%*%Z1)%*%t(Z1)
Omega2 = Z2%*%solve(t(Z2)%*%Z2)%*%t(Z2)
f.hat_CMI = eigen(Omega1 + Omega2)$vectors
### Extract two categories for classification ###
researchGroup = researchGroup[which(researchGroup != "MCI")]
f.hat_CMI = f.hat_CMI[which(researchGroup != "MCI"),]
###### rep is the number of repetitions ###
###### prob is the training rate             ###
###### M is the number of factors            ###
###### N is the sample size                       ###
M = c(1:max(M1,M2))
rep = 100; prob = 0.5
Accuracy = F1 = Sensitivity = Specificity = Precision = Recall = Mm = c()
for(l in 1:rep){
  ### divide the dataset ###
  folds = list(); accuracy_test = YY = YT = c(); iter = 3
  f.hat_CMI_train = f.hat_CMI_test = matrix(nrow = 0, ncol = N)
  for(i in 1:iter){
    folds[[i]] = createDataPartition(researchGroup, p = prob, list = FALSE)
    f.hat_CMI_train0 = f.hat_CMI %>% .[folds[[i]],]
    f.hat_CMI_train = rbind(f.hat_CMI_train, f.hat_CMI_train0)
    f.hat_CMI_test0 = f.hat_CMI %>% .[-folds[[i]],]
    f.hat_CMI_test = rbind(f.hat_CMI_test, f.hat_CMI_test0)
    YY0 = c(researchGroup %>% .[folds[[i]]])
    YY = c(YY, YY0)
    YT0 = c(researchGroup %>% .[-folds[[i]]])
    YT = c(YT, YT0)
  }
  AIC_model = c()
  for(m in 1:length(M)){
    ### train data ###
    Group = YY
    data = data.frame(cbind(f.hat_CMI_train[,1:M[m]]), Group)
    data$Group = factor(data$Group)
    mult.model = multinom(Group ~ ., data = data, MaxNWts = 10000)
    AIC_model[m] = mult.model$AIC
  }
  m = which.min(AIC_model)
  Mm[l] = M[m]
  ### train data ###
  Group = YY
  data = data.frame(cbind(f.hat_CMI_train[,1:M[m]]), Group)
  data$Group = factor(data$Group)
  mult.model = multinom(Group ~ ., data = data, MaxNWts = 10000)
  predlab = predict(mult.model, newdata = data, type = "class")
  confum_train = confusionMatrix(data = predlab, reference = data$Group, mode = "everything")
  accuracy_train = confum_train$overall[1]
  ### test data ###
  Group = YT
  data = data.frame(cbind(f.hat_CMI_test[,1:M[m]]), Group)
  data$Group = factor(data$Group)
  predlab = predict(mult.model, newdata = data, type = "class")
```

```
  confum_test = confusionMatrix(data = predlab, reference = data$Group, mode = "everything")
  accuracy_test = confum_test$overall[1]
  confumat_test = as.data.frame(confum_test$table)
  ggplot(confumat_test, aes(x = Reference, y = Prediction)) +
    geom_tile(aes(fill = Freq)) +
    geom_text(aes(label = Freq)) +
    scale_fill_gradient(low = "steelblue", high = "lightgreen", guide = "colorbar") +
    ggtitle("mlogit")
  summary_test = multiClassSummary(
    data.frame(obs = data$Group, pred = predlab), lev = levels(data$Group))
  Accuracy[l] = summary_test[1]
  Sensitivity[l] = summary_test[4]
  Specificity[l] = summary_test[5]
}
Accuracy_mean = mean(Accuracy)
Sensitivity_mean = mean(Sensitivity)
Specificity_mean = mean(Specificity)
print(c(Accuracy_mean, Sensitivity_mean, Specificity_mean))
```

The following program uses the FRI-PCA method for 2-class classification tasks.

```
### Accuracy(FRI-PCA)(2-class) ###
library(data.table)
library(tidyverse)
library(caret)
library(pROC)
library(nnet)
######### Read data #########
rho.hat_PET = fread("C:/Users/pc/Desktop/Logistic/Empirical analysis/2-source(0.95)/rho.hat_PET.csv") %>% as.matrix() %>% .[,1:100]
rho.hat_MRI = fread("C:/Users/pc/Desktop/Logistic/Empirical analysis/2-source(0.95)/rho.hat_MRI.csv") %>% as.matrix() %>% .[,1:235]
MMSE = fread("C:/Users/pc/Desktop/Logistic/Empirical analysis/2-source(0.95)/MMSE.csv", header = F) %>% .$V1
researchGroup = fread("C:/Users/pc/Desktop/Logistic/Empirical analysis/2-source(0.95)/researchGroup.csv", header = F) %>% .$V1
researchGroup[researchGroup == "EMCI"] = "MCI"
researchGroup[researchGroup == "LMCI"] = "MCI"
M1 = rho.hat_PET %>% ncol()
M2 = rho.hat_MRI %>% ncol()
NN = c(327, 315, 72); N = sum(NN); M1; M2
### Constructing missing principal components ###
rho.hat_PET_NA = matrix(NA, N, M1)
rho.hat_PET_NA[1:sum(NN[1:2]),] = rho.hat_PET
rho.hat_MRI_NA = matrix(NA, N, M2)
rho.hat_MRI_NA[1:NN[1],] = rho.hat_MRI[1:NN[1],]
rho.hat_MRI_NA[(sum(NN[1:2])+1):N,] = rho.hat_MRI[(NN[1]+1):(NN[1]+NN[3]),]
### CMI imputation ###
# Zero-mean imputation as initial value #
rho.hat_MRI_hat<-t(matrix(apply(rho.hat_MRI_NA[c(1:NN[1],(sum(NN[1:2])+1):N),], 2, mean), M2, NN[2]))
rho.hat_MRI_NA[(NN[1]+1):sum(NN[1:2]),] = rho.hat_MRI_hat
rho.hat_PET_hat<-t(matrix(apply(rho.hat_PET_NA[1:sum(NN[1:2]),], 2, mean), M1, NN[3]))
rho.hat_PET_NA[(sum(NN[1:2])+1):N,] = rho.hat_PET_hat
# Iterative regression #
n_iter = 50
for(j in 1:n_iter)
```

```
{
  rho.hat_MRI_hat = matrix(0,NN[2],M2)
  for (i in 1:M2) {
    fit = lm(rho.hat_MRI_NA[,i]~., data = as.data.frame(rho.hat_PET_NA))
    rho.hat_MRI_hat[,i] = cbind(rep(1,NN[2]),rho.hat_PET_NA[(NN[1]+1):sum(NN[1:2]),])%*%fit$coefficients
  }
  rho.hat_MRI_NA[(NN[1]+1):sum(NN[1:2]),] = rho.hat_MRI_hat
  rho.hat_PET_hat = matrix(0,NN[3],M1)
  for (i in 1:M1) {
    fit = lm(rho.hat_PET_NA[,i]~., data = as.data.frame(rho.hat_MRI_NA))
    rho.hat_PET_hat[,i] = cbind(rep(1,NN[3]),rho.hat_MRI_NA[(sum(NN[1:2])+1):N,])%*%fit$coefficients
  }
  rho.hat_PET_NA[(sum(NN[1:2])+1):N,] = rho.hat_PET_hat
}
### rho.hat_CMI as factor ###
score = cbind(rho.hat_PET_NA, rho.hat_MRI_NA)
z.hat = cov(score)
rho.hat_CMI = (rho.hat_PET_NA %*% eigen(z.hat)$vec[1:M1,1:(M1+M2)] +
                 rho.hat_MRI_NA %*% eigen(z.hat)$vec[(M1+1):(M1+M2),1:(M1+M2)]) %>% scale()
### Extract two categories for classification ###
researchGroup = researchGroup[which(researchGroup != "MCI")]
rho.hat_CMI = rho.hat_CMI[which(researchGroup != "MCI"),]
###### rep is the number of repetitions ###
###### prob is the training rate          ###
###### M is the number of factors         ###
###### N is the sample size               ###
M = c(1:(M1+M2))
rep = 100; prob = 0.5
Accuracy = F1 = Sensitivity = Specificity = Precision = Recall = Mm = c()
for(l in 1:rep){
  ### divide the dataset ###
  folds = list(); accuracy_test = YY = YT = c(); iter = 3
  rho.hat_CMI_train = rho.hat_CMI_test = matrix(nrow = 0, ncol = (M1+M2))
  for(i in 1:iter){
    folds[[i]] = createDataPartition(researchGroup, p = prob, list = FALSE)
     rho.hat_CMI_train0 = rho.hat_CMI %>% .[folds[[i]],]
    rho.hat_CMI_train = rbind(rho.hat_CMI_train, rho.hat_CMI_train0)
    rho.hat_CMI_test0 = rho.hat_CMI %>% .[-folds[[i]],]
    rho.hat_CMI_test = rbind(rho.hat_CMI_test, rho.hat_CMI_test0)
    YY0 = c(researchGroup %>% .[folds[[i]]])
    YY = c(YY, YY0)
    YT0 = c(researchGroup %>% .[-folds[[i]]])
    YT = c(YT, YT0)
  }
  AIC_model = c()
  for(m in 1:length(M)){

    ### train data ###
    Group = YY
    data = data.frame(cbind(rho.hat_CMI_train[,1:M[m]]), Group)
    data$Group = factor(data$Group)
    mult.model = multinom(Group ~ ., data = data, MaxNWts = 10000)
    AIC_model[m] = mult.model$AIC
```

```
    }
    m = which.min(AIC_model)
    Mm[l] = M[m]
    ### train data ###
    Group = YY
    data = data.frame(cbind(rho.hat_CMI_train[,1:M[m]]), Group)
    data$Group = factor(data$Group)
    mult.model = multinom(Group ~ ., data = data, MaxNWts = 10000)
    predlab = predict(mult.model, newdata = data, type = "class")
    confum_train = confusionMatrix(data = predlab, reference = data$Group, mode = "everything")
    accuracy_train = confum_train$overall[1]
    ### test data ###
    Group = YT
    data = data.frame(cbind(rho.hat_CMI_test[,1:M[m]]), Group)
    data$Group = factor(data$Group)
    predlab = predict(mult.model, newdata = data, type = "class")
    confum_test = confusionMatrix(data = predlab, reference = data$Group, mode = "everything")
    accuracy_test = confum_test$overall[1]
    confumat_test = as.data.frame(confum_test$table)
    ggplot(confumat_test, aes(x = Reference, y = Prediction)) +
      geom_tile(aes(fill = Freq)) +
      geom_text(aes(label = Freq)) +
      scale_fill_gradient(low = "steelblue", high = "lightgreen", guide = "colorbar") +
      ggtitle("mlogit")
    summary_test = multiClassSummary(
      data.frame(obs = data$Group, pred = predlab), lev = levels(data$Group))
    Accuracy[l] = summary_test[1]
    Sensitivity[l] = summary_test[4]
    Specificity[l] = summary_test[5]
  }
  Accuracy_mean = mean(Accuracy)
  Sensitivity_mean = mean(Sensitivity)
  Specificity_mean = mean(Specificity)
  print(c(Accuracy_mean, Sensitivity_mean, Specificity_mean))
```