07-31-2023

Wenjuan Bian,Caleb Simiyu Khaemba and Alice Wangui Wachira

2023-07-31

Abstract

This project explores the application of FDA to temporal gene expression data, implementing two distinct methodologies. The first approach employs Multinomial Logistic Regression, controlled by Leave-One-Out Cross-Validation (LOOCV), following the methodology of an existing study, and achieving an overall accuracy rate of 73.3% and a G1 classification accuracy of 87.5% in our implementation. The second approach leverages Penalized Functional Regression (PFR) using both Functional Principal Components and raw expression profiles for selected genes as predictors, resulting in an overall classification accuracy rate of 90% for G1 and non-G1. The report also emphasizes the random selection of training and testing datasets, a crucial step due to the absence of specific gene information in the original paper. In addition to successfully reproducing Figures 1, 2, 3, 4, 5, and 7 from the original paper, our Figure 6 diverges from the original by plotting pairwise predicted values against observations, providing a distinct analytical perspective.

Introduction

Functional Data Analysis (FDA) is a powerful statistical approach that deals with data represented as curves or functions, making it particularly well-suited for the analysis of temporal gene expression data. In this report, we aim to explore the application of FDA in the classification of gene expression profiles over time through two distinct approaches. The first employs Multinomial Logistic Regression with Leave-One-Out Cross-Validation (LOOCV), as described in a published paper. The second leverages Penalized Functional Regression (PFR), a method we learned in our course, using both Functional Principal Components and raw expression profiles for the selected 90 genes as predictors. Our primary objective is to replicate and reproduce key classification results from the original work, while also introducing innovations through the application of methods we learned in the course. The original article served as the foundation for our investigation, guiding our methodology and design. We successfully replicated the methods used to create Figures 1, 2, 3, 4, 5, and 7 in the original paper. While the specific plots are not exactly identical due to differences in the data, they maintain the essential structure and analytical approach of the original figures. By using Multinomial Logistic Regression controlled by LOOCV to classify five types of genes, we achieved an overall accuracy rate of 73.3% and a G1 classification accuracy rate of 87.5%. When applying Penalized Functional Regression (PFR) for the classification of G1 and non-G1, we attained an overall classification accuracy of 90%. In addition to this replication, we contributed our innovations through Table 2, showing the prediction results by the first approach, and Figure 6, where we plotted pairwise predicted values against observations for both approaches. These successes and distinctions underline both the robustness of the original paper's methodology and our innovative contributions in applying FDA techniques learned during our course. By following the approach proposed by the authors, while incorporating these two complementary techniques, we aim to ensure the reproducibility and application of what we learned. The selected data used in our project differs from what was employed in the original paper. In the given context, 90 genes have been identified by traditional methods as being related to the yeast cell cycle, with specific associations to different phases such as G1, S, S/G2, G2/M, and M/G1 [1]. However, it's important to note that we do not have detailed information about which specific genes were identified by these traditional methods and have data available. Given this lack of specific information, the selection of our training and testing datasets had to be conducted randomly. Without insight into the genes identified by traditional methods, we were unable to segregate or prioritize them based on their known characteristics or associations. As a result, random selection was the most unbiased approach available to partition the data into training and testing sets. The implications of this difference should be carefully

considered when comparing the results and conclusions drawn from our analysis with those from the original study. Our approach aims to practice what we have learned in the FDA course, while the original paper has more resources for data.

Methodology

Multinomial Logistic Regression (Model 1) with control of LOOCV: Following the methodology mentioned in the original paper, we employed multinomial logistic regression using the “multinom” method in the train function from the ‘caret’ package. This approach is consistent with the original paper and targets classification problems with a categorical response variable having more than two levels. The predictors in this model are the Functional Principal Components (FPCs), capturing the major patterns of variation in the functional data. The R code for this method is:

model1 <- train(Y ~ ., data = data1, method = “multinom”, trace = FALSE, trControl = train_control)

Here, Y represents the five different gene types, and data1 contains the predictors, which are the FPCs of raw expression profiles. The number of FPCs in data 1 is selected by minimizing the LOOCV classification error rate. The trace = FALSE argument prevents additional information from being printed during the training process, and train_control defines the resampling method used, such as LOOCV (Leave-One-Out Cross-Validation).

Genalized Functional Regression (PFR) with Logit Link (Model 2): For the second approach, we explored penalized functional regression (PFR) using both Functional Principal Components (FPCs) and raw expression profiles for the selected 90 genes in the training dataset as predictors. The aim was to compare these two types of input features within the framework of PFR. The code snippet for using raw expression profiles is:

pfr.fit.final <- pfr(Y ~ lf(X, k = k_opt, fx = FALSE), family = binomial(link = “logit”))

Here, Y is the binary response variable of G1 and non-G1, and X represents the chosen predictors, either the raw expression profiles or FPCs of them. The term lf(X, k = k_opt, fx = FALSE) represents a linear functional form, and k_opt is the selected number of basis functions by minimizing AIC. We also experimented with the “probit” link, achieving similar results to the “logit” link. Interestingly, both methods, utilizing raw expression profiles and FPCs, yielded very close results. This congruence showcases the robustness of the PFR method and offers multiple perspectives on the relationships between gene expression patterns and the response variable. By employing these two different types of predictors, we were able to validate our findings and gain comprehensive insights into the underlying structure of the data.

```
library(refund)
library(fda)
```

```
## Loading required package: splines
```

```
## Loading required package: fds
```

```
## Loading required package: rainbow
```

```
## Loading required package: MASS
```

```
## Loading required package: pcaPP
```

```
## Loading required package: RCurl
```

```
## Loading required package: deSolve
```

```
##
## Attaching package: 'fda'
```

```
## The following object is masked from 'package:graphics':
##
##     matplot
```

```
library(Ecdat)
```

```
## Loading required package: Ecfun
```

```
##
## Attaching package: 'Ecfun'
```

```
## The following object is masked from 'package:base':
##
##     sign
```

```
##
## Attaching package: 'Ecdat'
```

```
## The following object is masked from 'package:MASS':
##
##     SP500
```

```
## The following object is masked from 'package:datasets':
##
##     Orange
```

```
library(boot)
```

```
##
## Attaching package: 'boot'
```

```
## The following object is masked from 'package:fda':
##
##     melanoma
```

```
## The following object is masked from 'package:refund':
##
##     cd4
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:MASS':
##
##     select
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'lattice'
```

```
## The following object is masked from 'package:boot':
##
##     melanoma
```

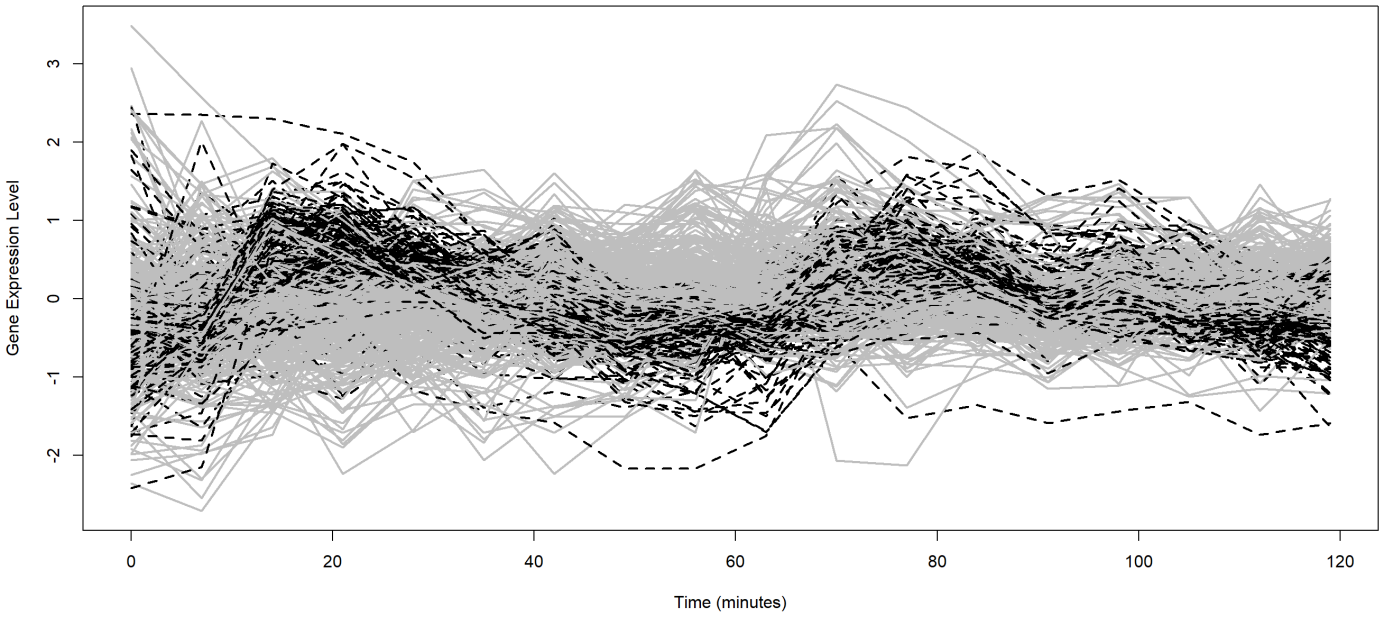
```
## The following object is masked from 'package:fda':
##
##     melanoma
```

```
library(caTools)
library(stepPlr)
library(plot3D)
library(gridExtra)
```

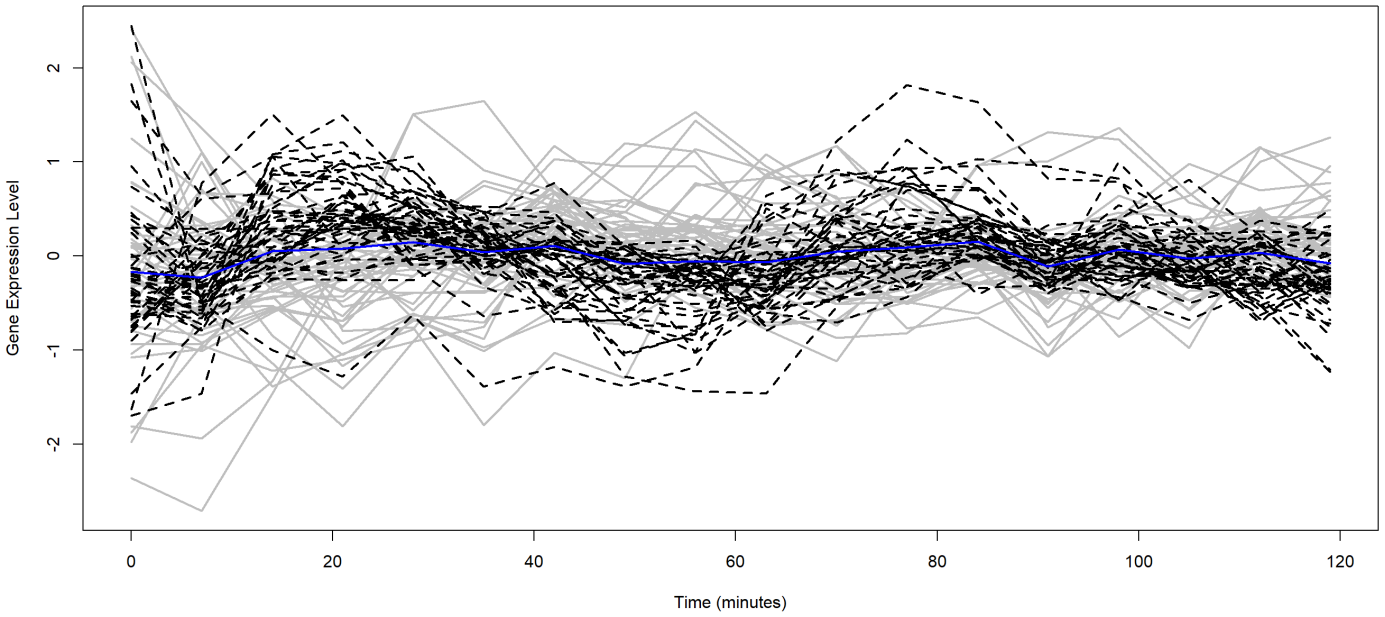
```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(scatterplot3d)

file_path <- "C:/Users/fwjbi/OneDrive - Bowling Green State University/Summer 2023/6820-Functio
n/Final Project/Data/cdcdata.txt"

# Read the file into a data frame
dtcdc <- read.csv(file_path, header = TRUE, sep = "\t", na.strings = "")


file_path2 <- "C:/Users/fwjbi/OneDrive - Bowling Green State University/Summer 2023/6820-Functio
n/Final Project/Data/orf_data.txt"

# Read the file into a data frame
dtorf <- read.csv(file_path2, header = TRUE, sep = "\t", na.strings = "")


# Find column indices containing 'alpha' or 'cdc' in dtcdc
alphancols <- grep("alpha", colnames(dtcdc))
columns1 <- c(1, alphancols)
subdtcdc <- dtcdc[, columns1]
names(subdtcdc)[names(subdtcdc) == "X"] <- "NAME"


# Merge subdtcdc and dtorf data frames by the 'NAME' column
merdata <- merge(subdtcdc, dtorf, by = "NAME", all.x = TRUE)

merdata<- merdata[complete.cases(merdata), ]



# Create a vector of line types based on the X.1 column
lines = ifelse(merdata$X.1 == "G1", 2, 1)  # use 2 for 'dashed' line and 1 for 'solid' line

# Create a vector of colors based on the X.1 column
colors = ifelse(merdata$X.1 == "G1", "black", "gray")  # use 'black' for G1 and 'gray' otherwise



time = seq(0, (ncol(merdata[, 2:19])-1)*7, by=7)

matplot(time, t(merdata[, 2:19]), type='l', lwd=2,lty=lines, col=colors,
        xlab="Time (minutes)",
        ylab="Gene Expression Level")
```


Gene Expression Level
3
2
1
0
-1
-2
0
20
40
60
80
100
120
Time (minutes)

```
# Randomly select 44 rows with X.1 equal to "G1"
set.seed(13)
subg1 <- subset(merdata, X.1 == "G1")
subg1 <- subg1[sample(nrow(subg1), 44), ]


# Randomly select 46 rows with X.1 not equal to "G1"
subnong1 <- subset(merdata, X.1 != "G1")
subnong1 <- subnong1[sample(nrow(subnong1), 46), ]

# Combine subset_g1 and subset_non_g1 into one data frame
gendata <- rbind(subnong1,subg1)

## Create gentest with the remaining rows from merdata
gentest <- merdata[!(rownames(merdata) %in% rownames(gendata)), ]

subg2 <- subset(gentest, X.1 == "G1")
subg2 <- subg2[sample(nrow(subg2), 44), ]
subnong2 <- subset(gentest, X.1 != "G1")
subnong2 <- subnong2[sample(nrow(subnong2), 46), ]

gendata1 <- rbind(subnong2,subg2)


################################## Figure 1
#Plot Figure1#
# Create a vector of line types based on the X.1 column
lines1 = ifelse(gendata$X.1 == "G1", 2, 1)  # use 2 for 'dashed' line and 1 for 'solid' line

# Create a vector of colors based on the X.1 column
colors1 = ifelse(!gendata$X.1 == "G1", "gray", "black")  # use 'black' for G1 and 'gray' otherwi
se

time1 = seq(0, (ncol(gendata[, 2:19])-1)*7, by=7)
matplot(time1, t(gendata[, 2:19]), type='l', lwd=2,lty=lines1, col=colors1,
        xlab="Time (minutes)",
        ylab="Gene Expression Level")

meangene <- rowMeans(t(gendata[, 2:19]))
lines(time1, meangene, col="blue", lwd=2, lty=1)
```




```
################################### Figure 2
#Plot figure 2
#Define the phase-specific groups
G1_data <- gendata[gendata$X.1 == "G1", 2:19]
S_data <- gendata[gendata$X.1 == "S", 2:19]
G2_data <- gendata[gendata$X.1 == "G2", 2:19]
M_data <- gendata[gendata$X.1 == "M", 2:19]
MG1_data <- gendata[gendata$X.1 == "M/G1", 2:19]


#Compute the mean function for each phase
basis <- create.bspline.basis(c(0,119), nbasis = 12)
G1_fd <- Data2fd(time1,t(G1_data),basis)
G1_mu <- mean.fd(G1_fd)

S_fd <-  Data2fd(time1,t(S_data),basis)
S_mu <- mean.fd(S_fd)

G2_fd <-  Data2fd(time1,t(G2_data),basis)
G2_mu <- mean.fd(G2_fd)

M_fd <-  Data2fd(time1,t(M_data),basis)
M_mu <- mean.fd(M_fd)

MG1_fd <-  Data2fd(time1,t(MG1_data),basis)
MG1_mu <- mean.fd(MG1_fd)

# Plot the gene expression profiles with mean functions
par(mfrow = c(2, 3))
matplot(time1, t(G1_data), type = "l", lty = 1, col = "gray", lwd = 1.25,xlab="",ylab="",ylim =
c(-2, 4), xlim = c(0, 120),main="G1")
plot(G1_mu,lwd =1.5, add= TRUE)
```

```
## [1] "done"
```

```
matplot(time1, t(S_data), type = "l", lty = 1, col = "gray", lwd = 1.25,xlab="",ylab="",ylim = c
(-2, 4), xlim = c(0, 120),main="S")
plot(S_mu,lwd =1.5, add= TRUE)
```

```
## [1] "done"
```

```
matplot(time1, t(G2_data), type = "l", lty = 1, col = "gray", lwd = 1.25,xlab="",ylab="",ylim =
c(-2, 4), xlim = c(0, 120),main="G2" )
plot(G2_mu,lwd =1.5, add= TRUE)
```

```
## [1] "done"
```

```
matplot(time1, t(M_data), type = "l", lty = 1, col = "gray", lwd = 1.25,xlab="",ylab="",ylim = c
(-2, 4), xlim = c(0, 120),main="M" )
plot(M_mu,lwd =1.5, add= TRUE)
```

```
## [1] "done"
```

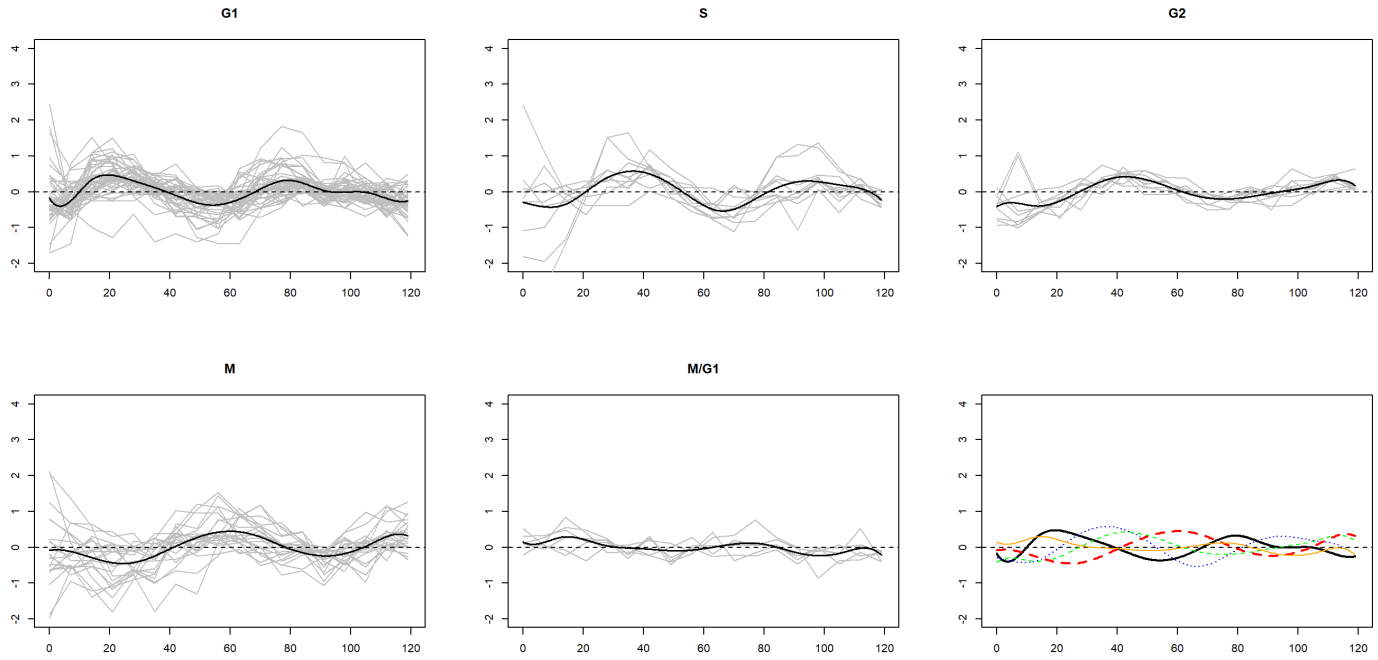
```
matplot(time1, t(MG1_data), type = "l", lty = 1, col = "gray", lwd = 1.25,xlab="",ylab="",ylim =
c(-2, 4), xlim = c(0, 120),main="M/G1")
plot(MG1_mu,lwd =1.5, add= TRUE)
```

```
## [1] "done"
```

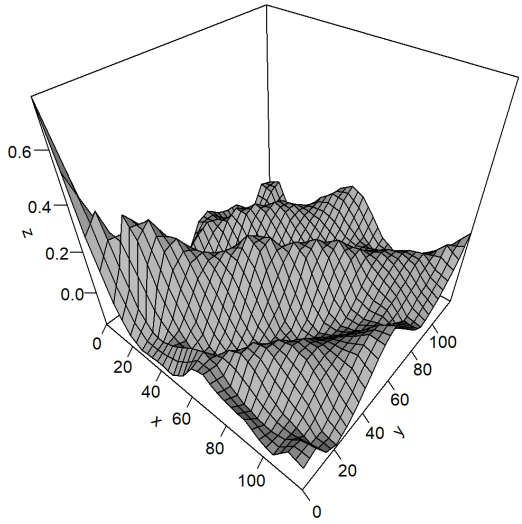
```
plot(G1_mu, col= "black", lwd= 2, ylim = c(-2, 4), xlim = c(0, 120),xlab="",ylab="")
```

```
## [1] "done"
```

```
lines(S_mu, col = "blue", lwd = 1.25, lty = "dotted")
lines(G2_mu, col = "green", lwd = 1.25, lty = "dashed")
lines(M_mu, col = "red", lwd = 2, lty = "44")
lines(MG1_mu, col = "orange", lwd = 1, lty = "solid")
```




```
##################################### Figure 3
#Plot figure 3
#Compute the covariance function
gendata_fd <- Data2fd(time1,t(gendata[ ,2:19]),basis)
gendata_cov <- var.fd(gendata_fd)
gendata_cov_mat <- eval.bifd(time1, time1,gendata_cov)


par(mfrow = c(1, 1))

persp3D(time1, time1, gendata_cov_mat,lphi=0, ltheta = 30, ticktype="detailed",
        border= "black", col="grey", shade = .5,asp = c(1, 1, 0.75),inttype = 3, resfac = 2, zmi
n = TRUE)
```




```
##################################### Figure 4
#Plot Figure 4

X=gendata[,2:19]
X2=gendata1[,2:19]
dim(X)
```

```
## [1] 90 18
```

```
# gendata$YG1 <- ifelse(gendata$X.1 == "G1", 1, 0)
Y=gendata$X.1
Y2=gendata1$X.1
X = as.matrix(t(X))
X2 = as.matrix(t(X2))
X = t(X)
X2 = t(X2)
dim(X)
```

```
## [1] 90 18
```

```
Y <- factor(gendata$X.1)
Y2 <-factor(gendata1$X.1)


grid <- seq(0, 119, by = 7)

# Create the Fourier basis with a reduced number of basis functions (nbasis = 5)
mybasis <- create.bspline.basis(rangeval = range(grid), nbasis = 12)

# Convert the data to functional data object using the Fourier basis
Xfdtrain <- Data2fd(argvals = grid, y = t(X), mybasis)
Xfdtrain2 <- Data2fd(argvals = grid, y = t(X2), mybasis)


error_rates <- numeric(0)
nharm_all = 3:12 # k = number of basis functions
suppressWarnings(
  # Loop through nharm values
  for (nharm in nharm_all) {
    # Perform PCA with current nharm value
    FPCtrain <- pca.fd(Xfdtrain, nharm = nharm)

    # Extract the scores from PCA
    Sctrain <- FPCtrain$score
    Sctrain_df <- as.data.frame(Sctrain)

    # Combine the scores with the Y variable
    data1 <- cbind(Y, Sctrain_df)

    # Set up trainControl for LOOCV
    train_control <- trainControl(method = "LOOCV")

    # Build the multinomial model using train function
    model <- train(Y ~ ., data = data1,
                   method = "multinom",trace = FALSE,
                   trControl = train_control)

    error_rate <- 1 -model$results$Accuracy[1]
    error_rates <- c(error_rates, error_rate)
  }
)


# Print the error rates for different nharm values
par(mfrow = c(1, 2))
print(error_rates)
```

```
##  [1] 0.3000000 0.2888889 0.3222222 0.3000000 0.3555556 0.3555556 0.3555556
##  [8] 0.4222222 0.4111111 0.4111111
```

```
plot(nharm_all, error_rates, cex = 1.5, pch = 2, ylab = "Classification error rate", xlab = "Num
ber of eigenfuntions", main="Error Rate vs FPCs, train (LOOCV)")

FPCtrain$varprop
```

```
##  [1] 3.580857e-01 2.918024e-01 1.683861e-01 1.010213e-01 2.142393e-02
##  [6] 1.723013e-02 1.176647e-02 1.129491e-02 9.135514e-03 6.945051e-03
## [11] 2.906461e-03 2.122647e-06
```

```
M_OPT <- nharm_all[which.min(error_rates)]
print(M_OPT)
```

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

```
FPCtrain <- pca.fd(Xfdtrain, nharm = M_OPT)
FPCtrain2 <- pca.fd(Xfdtrain2, nharm = M_OPT)
var.explained <- cumsum(FPCtrain$varprop)
n.components <- which(var.explained >= 0.90)[1]
n.components
```

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

```
Sctrain <- FPCtrain$score
Sctrain2 <- FPCtrain2$score
Sctrain_df <- as.data.frame(Sctrain)
Sctrain_df2 <- as.data.frame(Sctrain2)

data1 <- cbind(Y, Sctrain_df)
data2 <- cbind(Y2, Sctrain_df2)

# Set up trainControl for LOOCV
train_control <- trainControl(method = "LOOCV")

suppressWarnings(
model1 <- train(Y ~ ., data = data1,
                method = "multinom", trace = FALSE,
                trControl = train_control)
)
errorrate1 <- 1- model1$results$Accuracy
print(errorrate1)
```

```
## [1] 0.2888889 0.2888889 0.2888889
```

```
P1 <- model1$pred$pred
O1 <- model1$pred$obs
Error_rate <- num_differences <- sum(P1 != O1)/270
Error_rate
```

```
## [1] 0.2888889
```

```
print(model1$results$Accuracy)
```

```
## [1] 0.7111111 0.7111111 0.7111111
```

```
print(model1$results$Accuracy[1])
```

```
## [1] 0.7111111
```

```
pred1 <- predict(model1, data2)


Y2obs <- factor(data2$Y2)
unique(Y2)
```

```
## [1] M/G1 M    S    G2   G1
## Levels: G1 G2 M M/G1 S
```

```
# Compute the confusion matrix
conf_matrix <- confusionMatrix(pred1, Y2obs)

# Calculate the accuracy and error rate
accuracy <- conf_matrix$overall["Accuracy"]
error_rate <- 1 - accuracy

# Print the error rate
print(error_rate)
```

```
##  Accuracy
## 0.2666667
```

```
# Get the table of the confusion matrix
cm_table <- as.table(conf_matrix$table)

# Compute accuracy for each class
class_accuracies <- diag(cm_table) / rowSums(cm_table)

# Compute error rates for each class
class_error_rates <- 1 - class_accuracies
G1.C <- sum(gendata$X.1=="G1")
M.C <- sum(gendata$X.1 == "M")
S.C <- sum(gendata$X.1 == "S")
G2.C <- sum(gendata$X.1 == "G2")
MG1.C <- sum(gendata$X.1=="M/G1")

G1.C1 <- sum(gendata1$X.1=="G1")
M.C1 <- sum(gendata1$X.1 == "M")
S.C1 <- sum(gendata1$X.1 == "S")
G2.C1 <- sum(gendata1$X.1== "G2")
MG1.C1 <- sum(gendata1$X.1=="M/G1")

# Create a data frame
counts <- data.frame(
  Train_set = c(G1.C,  G2.C, M.C,MG1.C, S.C),
  Test_set = c(G1.C1,  G2.C1, M.C1,MG1.C1, S.C1),
  accuracy = class_accuracies,
  Error = class_error_rates
)
class_accuracies
```

```
##        G1        G2         M      M/G1         S
## 0.8750000 0.6153846 0.6250000       NaN 0.2000000
```

```
# Print the data frame
print(counts)
```

```
##      Train_set Test_set  accuracy     Error
## G1          44       44 0.8750000 0.1250000
## G2           9       15 0.6153846 0.3846154
## M           21       19 0.6250000 0.3750000
## M/G1         6        8       NaN       NaN
## S           10        4 0.2000000 0.8000000
```

```
sum(pred1=="M/G1")
```

```
## [1] 0
```

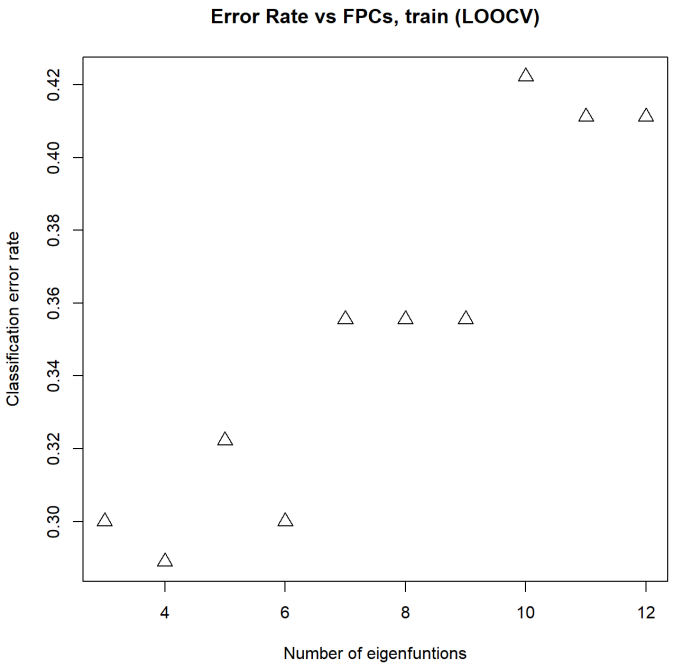
```
####################
mybasis <- create.bspline.basis(rangeval = range(grid), nbasis = 12)
# Convert the data to functional data object using the Fourier basis
Xfdtrain <- Data2fd(argvals = grid, y = t(X), mybasis)
Xfdtrain2 <- Data2fd(argvals = grid, y = t(X2), mybasis)


# Initialize empty vector to store AIC values
aic_values <- numeric(0)

# Initialize an empty data frame for results
result <- data.frame()

# Initialize variables to track the best parameters
min_aic <- Inf
best_k <- NA
best_nharm <- NA


  Y <- ifelse(gendata$X.1=="G1",1,0)
  Y2 <- ifelse(gendata1$X.1=="G1",1,0)

  # Compute AIC for different values of k
  aic_all = numeric(0)
k_all = 4:18
for(k in k_all){
  pfr.fit <- pfr(Y ~ lf(X, k = k, fx = FALSE), family = binomial(link = "logit"))
  tmp = extractAIC(pfr.fit)[2]
  aic_all = c(aic_all,tmp)
}

# Find the minimum AIC for this value of nharm
k_opt <- k_all[which.min(aic_all)]

aic_all
```

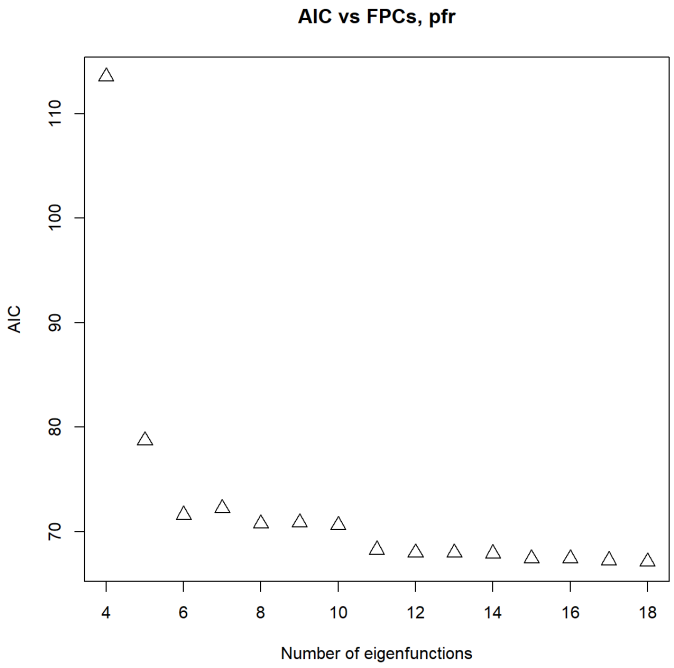
```
##  [1] 113.53855  78.72992  71.60561  72.21537  70.75303  70.84954  70.61562
##  [8]  68.25819  67.98803  67.98864  67.90441  67.40899  67.43468  67.25338
## [15]  67.10972
```

```
k_opt
```

```
## [1] 18
```

```
plot(k_all,aic_all, cex = 1.5, pch = 2, xlab="Number of eigenfunctions", ylab="AIC", main="AIC v
s FPCs, pfr")
```






```
length(k_all)
```

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

```
length(aic_all)
```

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

```
par(mfrow = c(1, 1))



pfr.fit.final <- pfr(Y ~ lf(X, k =  k_opt, fx = FALSE), family = binomial(link = "logit"))
# Make predictions
predictions <- predict(pfr.fit.final, newdata = list(X = X2))# type = "response")
pred2 <- ifelse(predictions > 0.5, 1, 0)

error.rate2 <- 1 - mean(pred2 == Y2)

# Print the error rate
print(error.rate2)
```

```
## [1] 0.1
```

```
################################### Figure 5
#Plot Figure 5

par(mfrow = c(2,2))
plot(FPCtrain$harmonics[1], ylab = "", lwd=3)
```

```
## [1] "done"
```

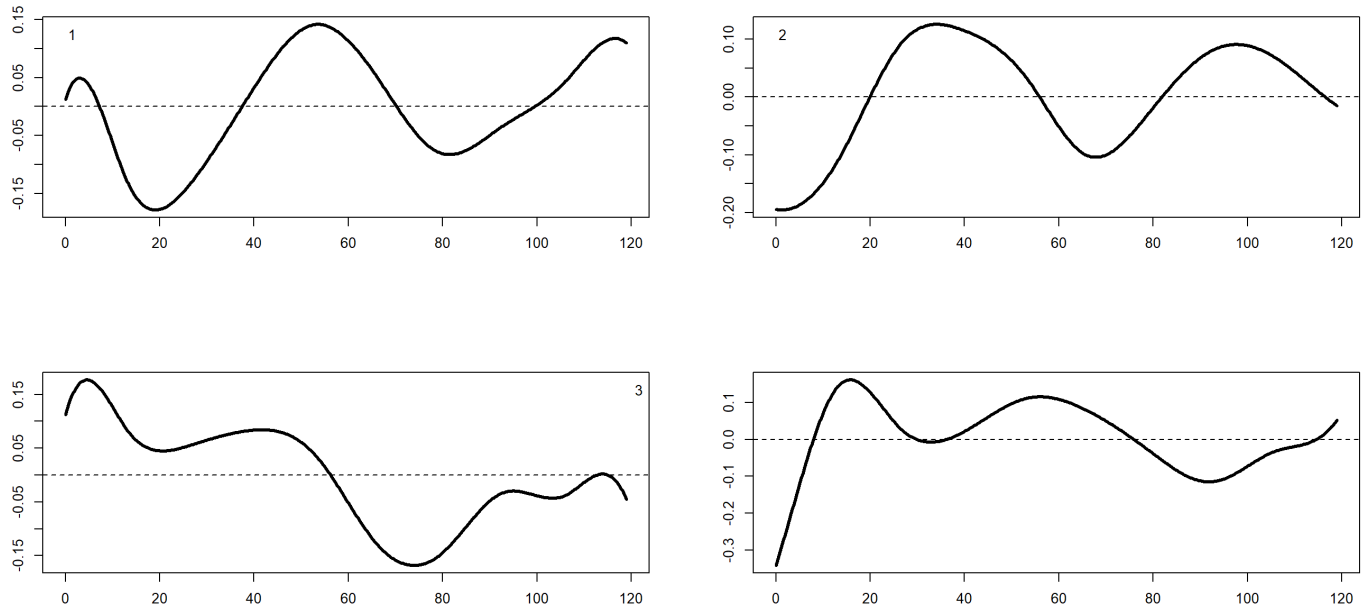
```
legend("topleft", legend = "1", bty = "n")
plot(FPCtrain$harmonics[2], ylab = "", lwd=3)
```

```
## [1] "done"
```

```
legend("topleft", legend = "2", bty = "n")
plot(FPCtrain$harmonics[3], ylab = "", lwd=3)
```

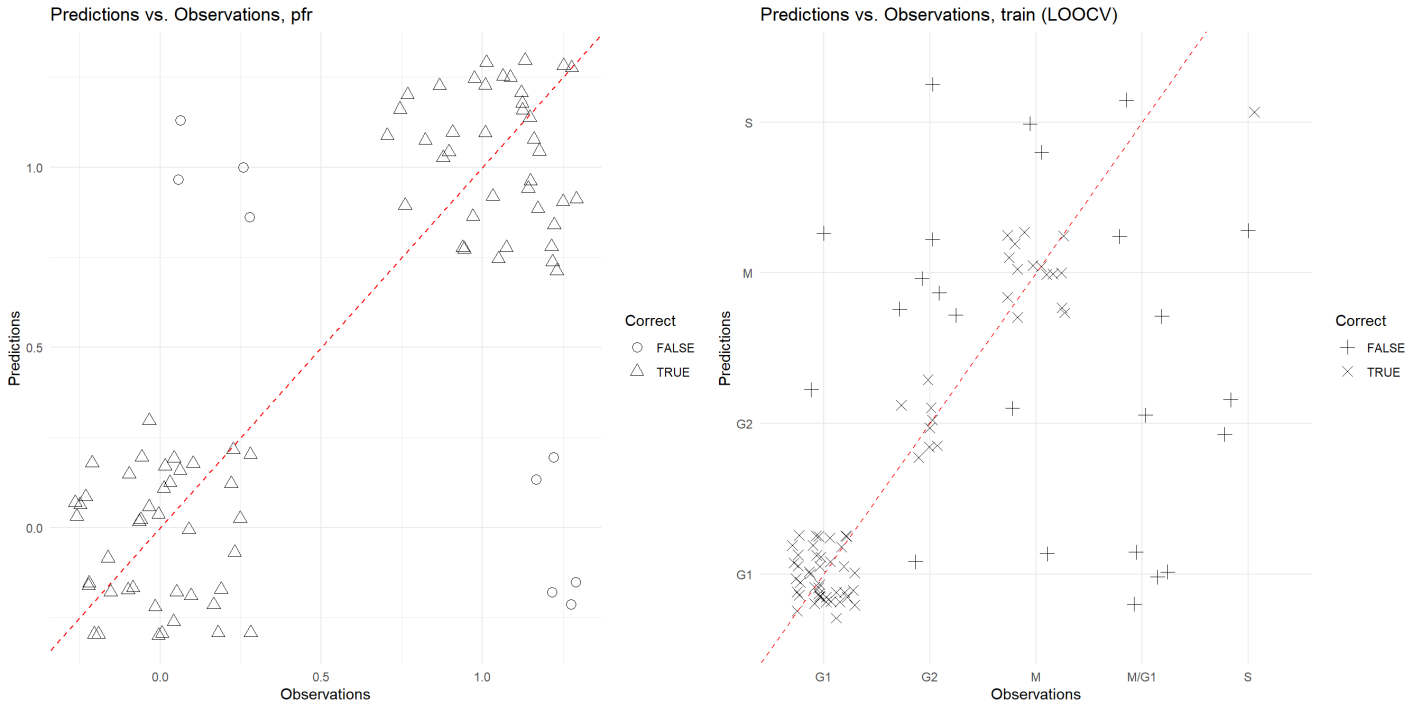
```
## [1] "done"
```

```
legend("topright", legend = "3", bty = "n")
plot(FPCtrain$harmonics[4], ylab = "", lwd=3)
```

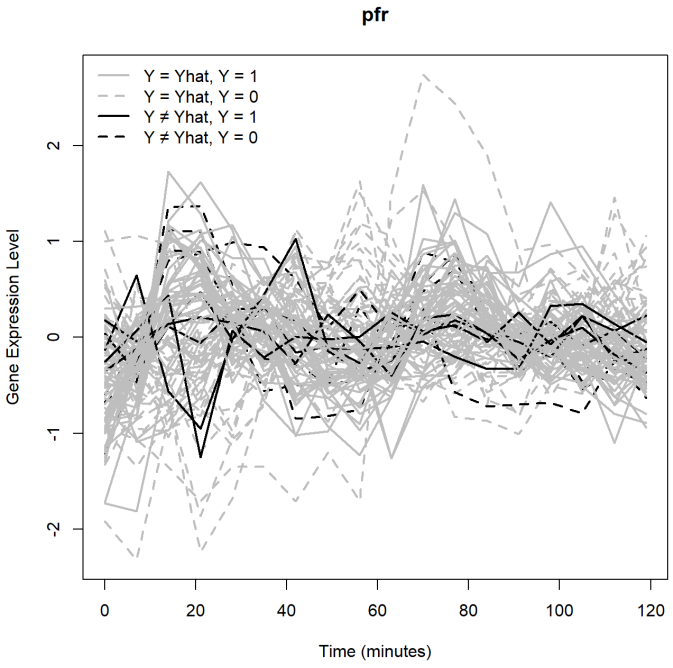


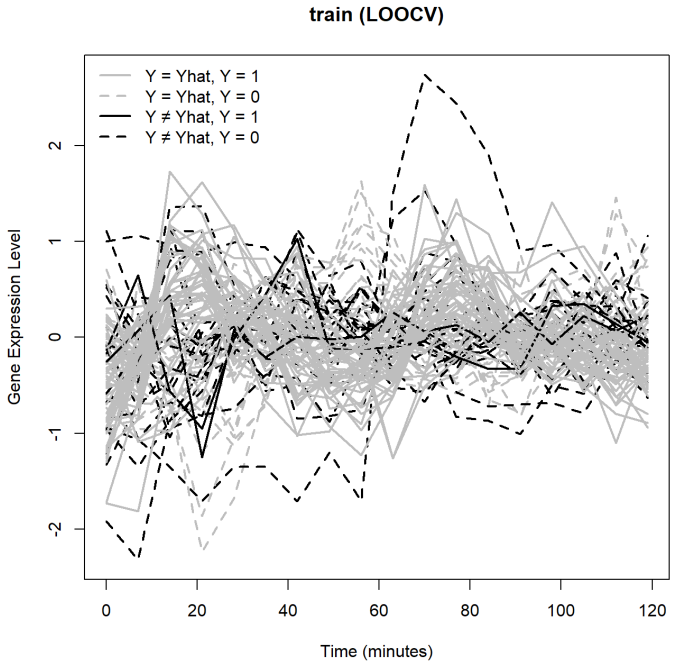

```
## [1] "done"
```

```
par(mfrow = c(1,1))

################################### Figure 7
#Plot Figure 7

datap <- data.frame(Predictions = pred1, Actual = Y2obs)
datap$Correct = datap$Predictions == datap$Actual

plot2 <- ggplot(datap, aes(x = Actual, y = Predictions)) +
  geom_jitter(aes(shape = Correct), size = 3, width = 0.3, height = 0.3) +
  scale_shape_manual(values = c(3, 4)) +
  geom_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") +
  xlab("Observations") +
  ylab("Predictions") +
  ggtitle("Predictions vs. Observations, train (LOOCV)") +
  theme_minimal()

datap1 <- data.frame(Predictions = pred2, Actual = Y)
datap1$Correct = datap1$Predictions == datap1$Actual

plot1 <- ggplot(datap1, aes(x = Actual, y = Predictions)) +
  geom_jitter(aes(shape = Correct), size = 3, width = 0.3, height = 0.3) +
  scale_shape_manual(values = c(1, 2)) +
  geom_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") +
  xlab("Observations") +
  ylab("Predictions") +
  ggtitle("Predictions vs. Observations, pfr") +
  theme_minimal()

grid.arrange(plot1, plot2, ncol = 2)
```



```
par(mfrow = c(1,2))

colors3 = ifelse(datap1$Correct == "TRUE", "gray", "black")

lines3 = ifelse(Y2 == 1, 1, 2)


matplot(time, t(X2), type='l', lwd=2,lty=lines3, col=colors3,
       xlab="Time (minutes)", ylab="Gene Expression Level", main = "pfr")

legend("topleft",
       legend = c("Y = Yhat, Y = 1", "Y = Yhat, Y = 0", "Y ≠ Yhat, Y = 1", "Y ≠ Yhat, Y = 0"),
       lty = c(1, 2, 1, 2), bty = "n",
       lwd = 2,
       col = c("gray", "gray", "black", "black"))


colors4 = ifelse(datap$Correct == "TRUE", "gray", "black")

lines4 = ifelse(Y2 == 1, 1, 2)


matplot(time, t(X2), type='l', lwd=2,lty=lines4, col=colors4,
        xlab="Time (minutes)", ylab="Gene Expression Level", main = "train (LOOCV)")

legend("topleft",
       legend = c("Y = Yhat, Y = 1", "Y = Yhat, Y = 0", "Y ≠ Yhat, Y = 1", "Y ≠ Yhat, Y = 0"),
       lty = c(1, 2, 1, 2), bty = "n",
       lwd = 2,
       col = c("gray", "gray", "black", "black"))
```





```
par(mfrow = c(1,1))
```