QoE Prediction for Streaming Video

Jean Steve TAMO

2024-06-06

Import the useful libraries

```
library("dplyr")
library("plyr")
library("lme4")
library("fda")
library("gridExtra")
library("mgcv")
library("refund")
library("mvtnorm")
library("flexmix")
library("latex2exp")
library("ggplot2")
library("e1071")
```

Read files

```
root_path <- "~/QoE_Data/LIVE_NFLX/data"
ldf <- list.files(path = root_path, pattern = ".csv")
raw.X <- lapply(ldf[-c(2:3,7,10:11)], function(var){

  print(var)

  ## Read data
  data <- read.csv(file = paste(root_path, var, sep = "/"))[,-1]

  ## fill data
  m <- ncol(data)
  for (i in 1:nrow(data)) {
    mi <- length(na.omit(as.vector(unlist(data[i,]))))
    if (mi < m){
      data[i,(mi+1):m] <- rev(as.vector(unlist(data[i,(2*mi-m):(mi-1)])))
    }
  }
  data
})
```

```
## [1] "buffer_evolution_sec.csv"
## [1] "MSSIM.csv"
## [1] "per_segment_encoding_height.csv"
## [1] "per_segment_encoding_QP.csv"
```

```
## [1] "playout_bitrate.csv"
## [1] "PSNR.csv"
## [1] "scene_cuts.csv"
## [1] "selected_streams.csv"
## [1] "SSIM.csv"
## [1] "STRRED.csv"
## [1] "throughput_trace_kbps.csv"
## [1] "VMAF.csv"
```

Logarithm Transformation of variables

```
# "buffer_evolution_sec.csv" : log(x) transformation
raw.X[[1]] <- log(raw.X[[1]])

# "MSSIM.csv : log(1-x) transformation
raw.X[[2]] <- log(1-raw.X[[2]])

# "playout_bitrate.csv : log(x) transformation
raw.X[[5]] <- log(1+raw.X[[5]])

# "selected_streams.csv : log(x) transformation
raw.X[[8]] <- log(1+raw.X[[8]])

# "SSIM.csv : log(1-x) transformation
raw.X[[9]] <- log(1-raw.X[[9]])

# "STRRED.csv : log(x) transformation
raw.X[[10]] <- log(1+raw.X[[10]])

# "throughput_trace_kbps.csv : log(x) transformation
raw.X[[11]] <- log(raw.X[[11]])
```

functional expansion of functional predictors

```
source("R/my_functrep.R")
X_fd.list <- lapply(1:length(raw.X), function(l){
  data <- raw.X[[l]]
  my_functrep(data = data, nbasis = min(100, ncol(data)),
              n.order = 4)
})

var <- "MSSIM.csv"
data <- read.csv(file = paste(root_path, var, sep = "/"))[,-1]
m <- ncol(data)
for (i in 1:nrow(data)) {
  mi <- length(na.omit(as.vector(unlist(data[i,]))))
  if (mi < m){
    data[i,(mi+1):m] <- rev(as.vector(unlist(data[i,(2*mi-m):(mi-1)])))
  }
}
X_fd.list[[length(X_fd.list)+1]] <- my_functrep(data = data,
```

```
                                                nbasis = min(100, ncol(data)),
                                                n.order = 4)

X_cut <- read.csv(file = paste(root_path, ldf[11], sep = "/"))[,-1]
X_cut <- sapply(1:nrow(X_cut), function(i){
  length(as.vector(na.omit(unlist(X_cut[i,]))))
})
X_cut <- X_cut/max(X_cut)

X_cut <- c(X_cut, rep(0,28))/max(X_cut)
X_cut2 <- read.csv(file = paste(root_path, ldf[11], sep = "/"))[,2]
X_cut2 <- c(X_cut2, rep(0,28))

X_cut3 <- read.csv(file = paste(root_path, ldf[11], sep = "/"))[,3]
X_cut3[is.na(X_cut3)] <- 0
X_cut3 <- c(X_cut3, rep(0,28))

X_cut4 <- read.csv(file = paste(root_path, ldf[11], sep = "/"))[,4]
X_cut4[is.na(X_cut4)] <- 0
X_cut4 <- c(X_cut4, rep(0,28))
```

Compute the derivatives

```
X_fd.list2 <- lapply(1:length(X_fd.list), function(l){
  deriv.fd(X_fd.list[[l]], 1)
})

X_fd.list3 <- lapply(1:length(X_fd.list), function(l){
  deriv.fd(X_fd.list[[l]], 2)
})

X_fd.list4 <- lapply(1:length(X_fd.list), function(l){
  deriv.fd(X_fd.list[[l]], 3)
})

X_fd.list <- unlist(list(X_fd.list, X_fd.list2, X_fd.list3, X_fd.list4), recursive = F)
```

Read the scalar predictors

```
X.scal <- read.csv(file = paste(root_path, ldf[10], sep = "/"))[,-c(1,6,7)]
X.scal <- cbind(X.scal, scene_cut = X_cut, first_cut = X_cut2,
                sec_cut = X_cut3, thi_sec = X_cut4)
```

Read the MOS scores

```
Y.mat <- read.csv(file = paste(root_path, ldf[2], sep = "/"))[,-1]
n <- nrow(Y.mat)
m <- sapply(1:n, function(i){length(as.vector(na.omit(unlist(Y.mat[i,]))))})
m2 <- ncol(Y.mat)
Y.mat2 <- Y.mat
```

```r
for (i in 1:n) {
  if (m[i] < m2){
    Y.mat2[i,(m[i]+1):m2] <- rev(as.vector(unlist(Y.mat[i,(2*m[i]-m2):(m[i]-1)])))
  }
}
```

Build the time matrix

```r
t.mat <- data.frame(t(data.frame(seq(0, 1, length.out = m[1]))))
for (i in 2:n) {
  tmp <- data.frame(t(data.frame(seq(0, 1, length.out = m[i]))))
  t.mat <- rbind.fill(t.mat, tmp)
}
```

Model

```r
source("R/penffr1.R")
d <- length(X_fd.list)
trace_name <- unique(X.scal$throughput_trace_name)
ind.test <- which(X.scal$throughput_trace_name %in% c("Train_vestby_oslo",
                                                      "Tram_jernbanetorget_ljabru"))
model <- penffr1(Y.mat = as.matrix(Y.mat2)[-ind.test,],
                 X_fd.list = lapply(1:d, function(l){
                   X_fd.list[[l]][-ind.test]
                 }), X.scal = X.scal[-ind.test,-c(3,10,11)],
                 pen = F, nbasis = 25)
```

Prediction

```r
pred <- pred.penffr1(model = model,
                     #t.mat = t.mat[ind.test, ],
                     newX_fd.list = lapply(1:d, function(l){
                       X_fd.list[[l]][ind.test]
                     }), newX.scal = X.scal[ind.test,-c(3,10,11)])

pred2 <- matrix(pred, ncol = m2, byrow = T)
```

Build the confident interval for predictions

```r
source("R/utils.R")
err <- t(sapply(1:120, function(i){
  abs(Y.mat2[ind.test[i],] - pred2[i,])
}))
err <- matrix(unlist(err), nrow = 120, byrow = F)

# Univariate conformal prediction
emp_quant <- function(err, tau = 0){
  if (tau == 0) {
```

```
    tau <- 0.95*(1+1/nrow(err))
  }
  sapply(1:ncol(err), function(j){
    quantile(unlist(err[,j]), probs = tau, names  = F)
  })
}
R <- t(sapply(1:nrow(err), function(i){
  emp_quant(err = err[-c(i,sample((1:nrow(err))[-i],29)),])
}))
```

```
## Useful preds
for (i in 1:length(ind.test)) {
  j <- ind.test[i]
  R[i,(m[j]+1):m2] <- rep(NA, m2-m[j])
  pred2[i,(m[j]+1):m2] <- rep(NA, m2-m[j])
}
```

```
conf.mat <- data.frame(fit = na.omit(as.vector(t(pred2))),
                       lwr = na.omit(as.vector(t(pred2))) - na.omit(as.vector(t(R))),
                       upr = na.omit(as.vector(t(pred2))) + na.omit(as.vector(t(R))))
pred3 <- na.omit(as.vector(t(pred2)))
Y.mat3 <- na.omit(as.vector(t(Y.mat[ind.test,])))
```

RMSE

```
sqrt((norm(matrix(Y.mat3 - pred3, nrow = 1), type = "F")^2)/length(pred3))
```

```
## [1] 0.2904745
```

OR

```
sum(sapply(1:length(pred3), function(i){
  j <- ind.test[i]
  1-as.numeric(between(Y.mat3[i], conf.mat$lwr[i], conf.mat$upr[i]))
}))/length(pred3)
```

```
## [1] 0.04488439
```