BIOS 7720: Applied Functional Data Analysis

Lecture 11: Function on Scalar Regression (cont)

Andrew Leroux

April 15, 2021

FoSR

$$
\begin{aligned}
\mathrm{y}_i(s) &= f_0(s) + f_1(s)\mathrm{x}_i + b_i(s) + \epsilon_i(s) \\
b_i(s) &\sim \mathrm{GP}(0, \boldsymbol{\Sigma}_b) \\
\epsilon_i(s) &\overset{\text{iid}}{\sim} N(0, \sigma_\epsilon^2)
\end{aligned}
$$

- Last time we saw how to estimate using an iterative procedure
- Applied to the NHANES data

FoSR

- Iterative procedure:
  1. Estimate the working model under independence
  2. Fit fpca to the residuals
  3. Extract the eigenfunctions
  4. Fit the “weak" oracle model
  5. Repeat 1-4 as necessary

FoSR: NHANES Physical Activity vs Age

- Consider the model

$$\mathrm{lAC}_i(s) = f_0(s) + f_1(s)\mathrm{Age}_i + b_i(s) + \epsilon_i(s)$$

- Problem: data size is huge

FoSR: NHANES Physical Activity vs Age

```r
df <- readr::read_rds(here::here("data","data_processed","NHANES_AC_processed.rds"))
## extract the PA data
lX <- log(1+as.matrix(df[,paste0("MIN",1:1440)]))
lX[is.na(lX)] <- 0
N  <- nrow(lX)
## bin the data into 60 minute intervals
tlen <- 60
nt   <- ceiling(1440/tlen)
inx_cols <- split(1:1440, rep(1:nt, each=tlen)[1:1440])
lX_bin   <- vapply(inx_cols, function(x) rowMeans(lX[,x], na.rm=TRUE), numeric(N))
## get subject average curves
inx_rows   <- split(1:N, factor(df$SEQN, levels=unique(df$SEQN)))
lX_bin_ind <- t(vapply(inx_rows, function(x) colMeans(lX_bin[x,], na.rm=TRUE), numeric(nt)))
nid <- nrow(lX_bin_ind)
# get a data frame for model fitting
sind <- seq(0,1,len=nt)
df_fit <-
  data.frame(lAC=as.vector(t(lX_bin_ind)),
             sind = rep(sind, nid),
             SEQN = rep(unique(df$SEQN), each=nt)) %>%
  left_join(dplyr::select(df[!duplicated(df$SEQN),], SEQN, Age), by="SEQN") %>%
  mutate(id = factor(SEQN)) %>%
  filter(!is.na(Age))
```

FoSR: NHANES Physical Activity vs Age

```
## subset the data to just 500 participants
set.seed(10110)
nid_samp <- 500
id_samp  <- sample(unique(df_fit$id), size=nid_samp, replace=FALSE)
df_fit_sub <- subset(df_fit, id %in% id_samp)

## fit the naive model
fit_naive <- bam(lAC ~ s(sind, bs="cc",k=20) + s(sind, by=Age, bs="cc",k=20),
                 method="fREML",data=df_fit_sub, discrete=TRUE)
## extract the resiudals
resid_mat <- matrix(fit_naive$residuals,
                    nid_samp, nt,byrow=TRUE)
## fit fpca
fpca_fit  <- fpca.face(resid_mat, knots=15)
## add in eigenfunctiosn
for(k in 1:length(fpca_fit$evalues)){
    df_fit_sub[[paste0("Phi",k)]] <- rep(fpca_fit$efunctions[,k],nid_samp)
}
```

FoSR: NHANES Physical Activity vs Age

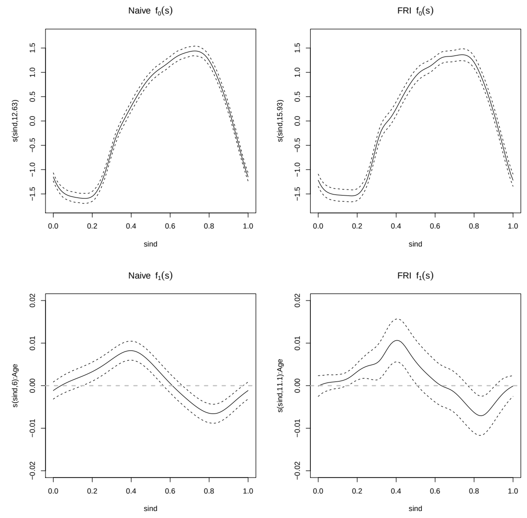
```r
## fit the fri model using mgcv::bam()
## using the first four eigenfunctions only
system.time({
    fit_fri <- bam(lAC ~
                     s(sind, bs="cc",k=20) +
                     s(sind, by=Age, bs="cc",k=20) +
                     s(id, by=Phi1, bs="re") + s(id, by=Phi2, bs="re") +
                     s(id, by=Phi3, bs="re") + s(id, by=Phi4, bs="re"),
                   method="fREML",data=df_fit_sub, discrete=TRUE)
})

##    user  system elapsed
##  48.315   0.778  49.101
```

FoSR: NHANES Physical Activity vs Age

- Plot the estimated $\hat{f}_0(s)$ and $\hat{f}_1(s)$ from the naive and FRI fits
- Compare the shapes and CIs
- Do these results make sense?

FoSR: NHANES Physical Activity vs Age



FoSR: NHANES Physical Activity vs Age

- Problem: functional random intercept models computationally expensive
    - RAM
    - Time
- One possible solution
    - Choose smoothing parameter(s) for fixed effects using leave-several-out CV MSE estimated using the naive model
    - Estimate standard errors by bootstrapping curves
        - Fix λ at the value obtained from using CV MSE
        - Estimate λ at each stage

In-Class Exercise

1. Obtain optimal smoothing parameters for the varying coefficient model using leave-one-function out CV MSE (you may use a relatively coarse grid for $\log(\lambda)$ for faster computation time). If you find your implementation is too slow for whatever reason, you can implement leave-several-function out CV MSE instead.
2. You may fix λ_{f_0} at it's estimated value from the FRI fit (otherwise you'd need to do a bivariate grid search)
3. Given the smoothing parameter you found, estimate $\text{SE}(\hat{f}_1(s))$ by bootstrapping curves (you may use a relatively small number of bootstrap samples, 50-100)
4. Plot the results, compare to the naive and FRI model results

In-Class Exercise: Some Helpful Code

```r
## fitting the model using mgcv with fixed smoothing parameters
sp_fri       <- fit_fri$sp[1:2]
lambda_f1    <- 5
fit_naive_sp <- bam(lAC ~ s(sind, bs="cc",k=20,sp=sp_fri[1]) +
                         s(sind, by=Age, bs="cc",k=20,sp=candidate_lambda,
                    method="fREML",data=df_fit_sub, discrete=TRUE)
```

In-Class Exercise: Some Helpful Code

```r
## get indices for the leave-function(s)-out CV
# get unqiue participant IDs
uid <- unique(df_fit_sub$SEQN)
nid <- length(uid)
# number of folds to split the participants
nfolds <- 5
# set the seed
set.seed(1010)
# create a vector of fold indicators
fold_vec <- sample(rep(1:nfolds, ceiling(nid/nfolds))[1:nid])
# combine with IDs
df_folds <- data.frame(SEQN=uid, fold=sample(1:nfolds))
# merge with the data frame for fitting
df_fit_sub <-
  left_join(df_fit_sub, df_folds, by="SEQN")
# create a list with each element as the vector of row indices
# associated with each fold
inx_ls <- split(1:nrow(df_fit_sub), df_fit_sub$fold)
```

In-Class Exercise: Some Helpful Code

```r
## set up grid search (here do a coarse search for computation time)
nlambda   <- 25
loglambda <- seq(5,20,len=nlambda)
lambda    <- exp(loglambda)
## set up progress bar
pb  <- txtProgressBar(min=0,max=nlambda*nfolds,style=3)
inx_pb <- 1
## loop over candidate smoothing parameters
mse <- rep(NA, nlambda)
## get smoothing parameters from the FRI model
sp_fri <- fit_fri$sp[1:2]
for(l in 1:nlambda){
  ## loop over "folds"
  mse_l <- rep(NA, nfolds)
  for(i in 1:nfolds){
    # get the training and test data
    inx_li   <- inx_ls[[i]]
    df_train <- df_fit_sub[-inx_li,]
    df_test  <- df_fit_sub[inx_li,]
    # fit the model using the training data
    fit_li   <- bam(lAC ~ s(sind, bs="cc",k=20,sp=sp_fri[1]) +
                         s(sind, by=Age, bs="cc",k=20,sp=lambda[l]),
                   data=df_train)
    # predict on the test data
    fhat_test <- predict(fit_li, newdata=df_test, type='response')
    mse_l[i]  <- mean((fhat_test-df_test$lAC)^2)
    ## update progress bar
    setTxtProgressBar(pb, inx_pb)
    inx_pb <- inx_pb + 1
  }
  # average over the folds for this candidate smoothing parameter
  mse[l] <- mean(mse_l)
}
```