vsFlcm： Variable Selection in FLCM (Functional Linear Concurrent Model)

Author: Hongming Pu (phmhappier@163.com)

Version: 0.2.0

This package implements two statistical methods for selecting variables in the functional linear concurrent model. These methods are described later in this article.

Methodology

The functional linear concurrent model can be written as

$Y_i(t) = \Sigma_{k=1}^{p} X_{ik}(t)\beta_k(t) + \delta_i(t) + \epsilon_i(t)$

,where t denotes the time, i denotes the subject, Y denotes the response, k denotes the index of predictor function, β denotes the coefficient function, δ denotes the individual residual and ϵ denotes the error.

We expand $\beta_k(t)$ with a spline basis.

$\Theta(t) = [\theta_1(t), \ldots, \theta_{K_\theta}(t)]^T$

$\beta^{(k)} = [\beta_{k1}, \ldots, \beta_{kK_\theta}]^T$

$\beta_k(t) = \Theta(t)^T \beta^{(k)}$

Since different subjects have different individual effects, we expand individual residual with functional principal component basis functions.

$\Theta_\delta(t)^T = \Theta(t)^T A$

$R_i = [R_{i1}, \ldots, R_{iK_0}]^T$

$\delta_i(t) = \Theta_\delta(t)^T R_i$

where A is a $K_\theta \times K_0$ matrix. Note $K_0 < K_\theta$

Then we can compute $\hat{Y}$ with all the parameters above including $\beta^{(k)}, 1 \leq k \leq p; A; R_i, 1 \leq i \leq n,$

$\hat{Y}_i(t_{i_j}) = \Sigma_{k=1}^{p} X_{ik}(t_{i_j})\hat{\beta}_k(t_{i_j}) + \hat{\delta}_i(t_{i_j})$

We use group Lasso to select useful predictors. Since $\beta_{k1}, \ldots, \beta_{kK_\theta}$ are all parameters of β_k and our goal is to select useful β_ks from $\beta_1, \ldots, \beta_p$, $\beta_{k1}, \ldots, \beta_{kK_\theta}$ are a natural group. The penalty term of β_k is

$$P_k = \sqrt{\int_{t.min}^{t.max} \beta_k(t)^2 dt} = \sqrt{{\beta^{(k)}}^T \Theta_{int} \beta^{(k)}}$$

$$\Theta_{int}(i,j) = \int_{t.min}^{t.max} \theta_i(t)\theta_j(t)dt$$

Thus the methodology is to minimize the sum of penalties and residuals.

minimize $F_1 = ||Y - \hat{Y}||_2^2 + \lambda \Sigma_{k=1}^{p} P_k$

where λ is the parameter of group Lasso. This is the first method implemented in my package. Choose *method.obj='nonconvex'* to use it.

```
vsflcm(...,method.obj='nonconvex',...)
```

However, F_1 is **not** a convex function. To solve this problem, we propose an alternative methodology. We define

$$R = [R_1, \ldots, R_n]$$

$$M_{AR} = AR$$

$$\beta = [\beta^{(1)}, \ldots, \beta^{(p)}]$$

where M_{AR} is a $K_\theta \times n$ matrix whose rank is not larger than K_0. It is easy to verify that F_1 is a convex function of M_{AR} and β. To model the low-rank attribute of M_{AR}, we add a nuclear norm penalty.

$$P_M = ||M_{AR}||_*$$

Then we have the new object function which is convex:

minimize $F_2(M_{AR}, \beta s) = F_1 + \lambda_* P_M = |Y - \hat{Y}||_2^2 + \lambda \Sigma_{k=1}^{p} P_k + \lambda_* P_M$

where λ_* is the parameter of nuclear norm. This is the second method we implemented in this R package. Choose *method.obj='nuclear'* to use it.

```
vsflcm(...,method.obj='nuclear',...)
```

Typically the second method is better.

Installation

You can install this package from GitHub with devtools:

```
library(devtools)
devtools::install_github("Hongming-Pu/vsFlcm")
```

Example of Use

The code below simulates a dataset under the functional linear concurrent model. For each of 30 subjects, observations of 30 predictor functions and a response function are observed over times between 0 and 1. Different subjects have different individual effects.

```
library(vsFlcm)
set.seed(1)

#generate parameters
n.sub<-30
n.time<-30
n.var<-30
fpc.rate<-0.2
n.total<-n.sub*n.time
train.rate<-0.5
n.train<-floor(n.total*train.rate)
n.test<-n.total-n.train
train.set<-sample(1:n.total,n.train)
test.set<-c(1:n.total)[-train.set]

#functions
f1<-function(x){return(sin(2*pi*x))}
f2<-function(x){return(cos(2*pi*x))}
f.fpc<-function(pa,x){return(fpc.rate*sin(2*pi*(x+pa)))}

#generate the data
data.time = runif(n.total,0,1)
data.var<-
matrix(rnorm(n.total*n.var),nrow=n.total,ncol=n.var)
res1<-
f1(data.time)*data.var[,1]+f2(data.time)*data.var[,2]
subs<-as.vector(rep(1,n.time)%*%t(c(1:n.sub)))
fpc.par<-as.vector(rep(1,n.time)%*%t(runif(n.sub)))
res.fpc<-f.fpc(fpc.par,data.time)
```

```
n1<-length(res1)
sd<-0.2
res.error<-rnorm(n1)*sd
res<-res1+res.fpc+res.error
data.fin<-cbind(res,subs,data.time,data.var)
vars<-paste0('V',1:n.var)
colnames(data.fin)<-
c('res','sub','time',paste0('V',1:n.var))
data.fra<-as.data.frame(data.fin)
data.train<-data.fra[train.set,]
data.test<-data.fra[test.set,]
formula = as.formula( paste("res~", paste(vars, collapse
= "+")) )

#fit the model
res<-vsflcm(formula,data =
data.train,id.time='time',t.min=0,t.max=1,
              id.sub =
'sub',lambda=10,method.optim='BFGS',method.obj =
'nuclear',
              delta = 0.01,times = 1,fpc.on =
TRUE,lam.nuc = 3)
pred<-predict(res,data.test,sub.reg = FALSE)
error<-pred-data.test$res
print(mean(error*error))
```