Computational Methods for Functional Data Analysis: Bootstrap Smoothing, Dimension Reduction and L1 Regularized Prediction

Author: Bingfan Liu

Abstract

In the real world, many data are collected in the form of curves when each observation is recorded densely along the time domain. Many researchers have found out that treating this type of data as discretized vectors as in the classical machine learning studies will not generate a good prediction result. What's more, when the number of the observations are far fewer than the dimensionalities of the data, the classical approaches are not feasible for solving the problems.

In this project, I implemented a bootstrap smoothing method and two dimension reduction methods that helped to address these problems. And finally four non-regularized/l1-regularized models combined with the methods above were used for comparing the prediction result regarding the power of smoothing and dimension reduction.

1. Introduction

In many practical applications, the data should be treated as curves. There are potential correlations between coordinates when the observation is recorded densely along the time domain. As a result, functional data analysis is used to address the issues and discover the underlying nature of the data. The typical applications, just to name a few, are in the fields of brain signal processing (Lila et al., 2017), spectrometry (Rossi et al., 2006), speech recognition(Biau et al., 2005) and etc.

In this study, I adapted a bootstrap sampling method for data smoothing. A sampling method and a projection kernel method is implemented for addressing the high dimension issue. Combined with the non-regularized/l1-regularized logistic regression and support vector machine, the methods were examed on a diffusion tensor imaging (DTI) dataset to illustrate their power in predicting patient's water diffusion status using fractional anisotropy (FA) measure.

In the following sections of this study, section 2 introduced the DTI dataset used in the analysis. Section 3 specified the methodologies. Section 4 explained the prediction results from the experiments. Section 5 is the discussion. References are summarized in section 6. Section 7 which is the appendix contains the code.

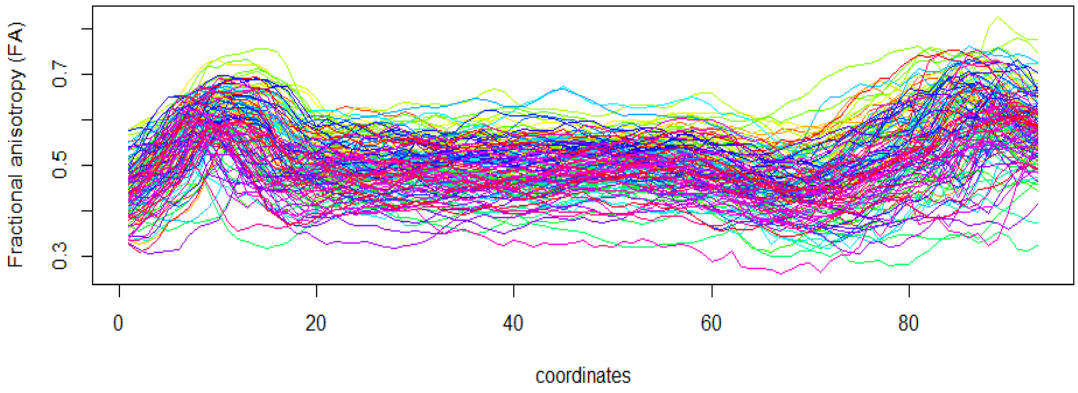
2. Data

The data used in this study is the Diffusion Tensor Imaging (DTI) data collected at Johns Hopkins University and the Kennedy-Krieger Institute presented in the R package refund (Goldsmith et al., 2010).

DTI is a method which measures proxies of demyelination. A common proxy is fractional anisotropy (FA) which is sensitive to microstructural changes and can be used to analyze the water diffusivity in the brain. (Alexander et al., 2007)

In this study, I used FA profiles of 143 patients' corpus callosum (CCA) water diffusion status at their first visit. And the aim is to use the different computational methods and predict the water diffusion status of the patients.

Fig.1: FA measures of water diffusion for all patients



3. Methodologies

3.1 Smoothing using bootstrap

In the literature, people believe that there are underlying functions that generate the curves which we observe. Using the smoothed curves, based on the observed discrete data points, to be the approximates of the underlying functions will have many advantages in statistical analysis.

There are three main reasons that we may want to consider using smoothing techniques for functional data:

- The functional data collected in practice are not perfectly continuous. The curves are observed and recorded at some certain time points as a dense discrete vector. Smoothing makes it possible for people evaluate a curve at any given time point.
- Due to the measurement error, it is common that the observed curves are measured with nose. Smoothing is a technique that performs denoising.
- What's more, in some study, people observe multiple curves with time lag. Using smoothing to capture the true underlying structure of the curve and then perform registering is a popular method for handling such problem.

The popular smoothing methods in the literatures are smoothing using roughness penalty (Ramsay et al.,2005). That is the smoothing methods selecting the optimal number of basis functions while controlling the complexity of the curvatures. In this study, I will use bootstrap sampling combined with basis expansion for smoothing the observed data.

In this study, the observed data are denoted to be

$$Y_i(t_j) \; or \; Y_{ji \; j=1,\ldots,m \; i=1,\ldots,n}$$

and the smoothed data to be

$$X_i(t_j) \; or \; X_{ji \; j=1,\ldots,m \; i=1,\ldots,n}$$

where n is the number of the observations and m is the length of the curves. Each of them represents a column vector (if i=1) or a column matrix, ie. each column of the matrix (or vector) is an observation. The relation between them is

$$Y_i(t_j) = X_i(t_j) + \epsilon_i(t_j)$$

where the ϵ_{ij} is assumed to be the white noise for i-th observation at the time point j. The aim of smoothing is to estimate $X_i(t) \; \forall i$.

It is well known that in the literature of non-parametrics, a curve can be represented using a set of basis functions denoted by $\phi_k; k \geq 1$ where each basis function is orthogonal to the others. Then a smooth function can be written as

$$X_i(t) \approx \sum_{k=1}^{K} c_{ik} \; \phi_k(t) = \Phi_i C_i$$

where C_i is a column vector of c_{ik}'s, and Φ_i is the m by K matrix whose each column is a basis function $\phi_i(t)$'s. So a set of n curves with length of m each, $Y_{m\times n}$, can be represented by a coefficient matrix,$C_{m\times n}$, and a set of basis functions Φ as:

$$Y \approx \Phi C + \epsilon$$

So that coefficient matrix C can be estimated using OLS as

$$C = (\Phi^T \Phi)^{-1} \Phi^T Y$$

Using more basis functions will have a good approximate of the data, but likely to overfit $X(t)$. While using less basis functions will have less flexibility and represent the data poorly. In order to have a good representation of the data while excluding the potential noise in the observation, we need to choose an optimal number of basis functions, K, that balances this trade off.

Here, I implemented a bootstrap method combined with least square estimations for finding K and smoothing the data. The evaluation of the goodness of fit was based on the MSE.

The bootstrap algorithm implemented in this project is as follows:

---

1. Start with smoothing a single curve $y(t)$ using OLS with some small and randomly chosen K to obtain the smoothed fitted curve $x(t)$ and residual $\epsilon(t) = y(t) - x(t)$. The fitted curve $x(t)$ will be treated as a benchmark.
2. For a single curve, using simple random sampling with replacement on its estimated residuals to obtain the sampled residuals $\epsilon'(t)$. Then add them to the corresponding benchmark curve $x(t)$ to obtain a simulated observed curve $y'(t)$ where $y'(t) = x(t) + \epsilon'(t)$.
3. Smooth the curve with various choices of K and calculate each MSE in L2 space.
4. Repeat step 2 and 3 for a large number of B times. Find the K that costs a smallest MSE over B samples.
5. Apply the above steps for every curve.

Notice that in the functional data framework the MSE is a curve:

$$MSE[\hat{X}(t)] = E[\hat{X}(t) - E[X(t)]]^2 = Bias^2\left[\hat{X}(t)\right] + Var[\hat{X}(t)]$$

where

$$Bias[\hat{X}(t)] = E[\hat{X}(t)] - X(t)$$

$$Var[\hat{X}(t)] = E[\hat{X}^2(t)] - [E[X(t)]]^2$$

We need to transform it into L2 space in order to do the comparison. The MSE_{L2} is calculated by doing the following integration:

$$MSE_{L2} = \int_T MSE[\hat{X}(t)]dt \approx w \sum_{t^*=\frac{T}{n}}^{T} MSE[\hat{X}(t^*)]$$

which is approximated by right Riemann Sum, where $w = \frac{T}{n}$.

Fig.2: Patient 1's FA measures

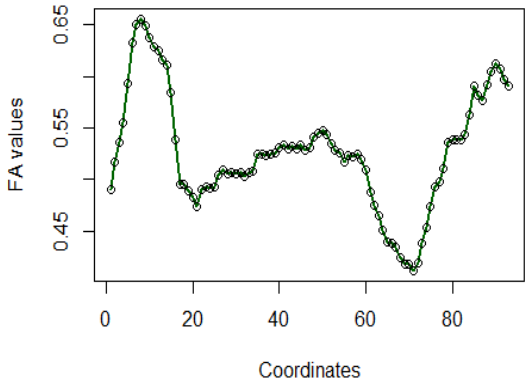
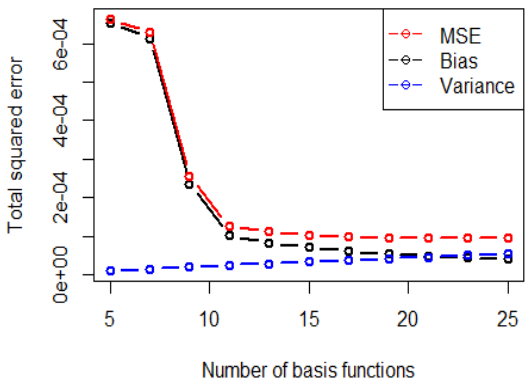


Fig.3: Patient 1's Bias Variance trade off



Using the first patient in the dataset for illustration, the patient's FA measures were visualized in figure 2. As we can see that the raw observation was a little bit noisy. Directly using the raw data might add some noise into the analysis.

The results of bootstrap sampling, the estimated MSE, Bias and Variance, were shown in the figure 3. One can easily see that with more basis functions added into smoothing the observed curve, the Bias decreased but variance increased. At some point, MSE reached its minimum, and the corresponding K value is the optimal number of basis to use.

Using the optimal K value and we can obtain the smoothed data for patient 1 or every patient shown in figure 4 and 5.

Fig.4: Patient 1's smoothed FA

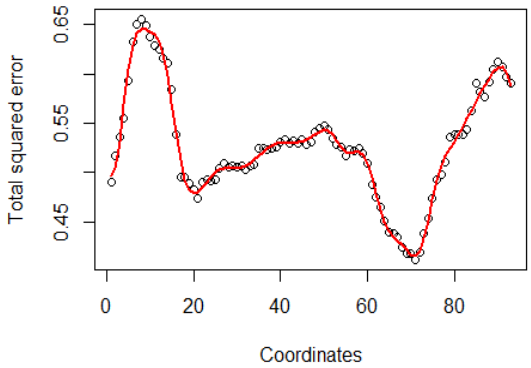
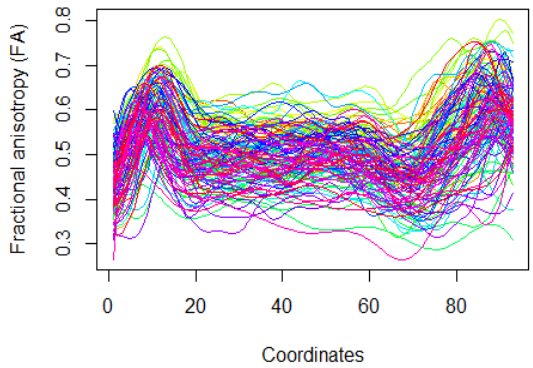


Fig.5: Smoothed FA measures, 143 patients



3.2 Dimension reduction

Due to the high dimensionality of the data, researchers usually seek different methods for selecting important features. There are many popular dimension reduction methods exist. For example, the functional principal component analysis (fPCA) is a popular and powerful tool to extract information from variations. (Besse P. et al., 1986)

In this section, I implemented two dimension reduction methods. One is sampling method and the other one is projection method which is suggested in Rossi et al. (2006).

3.2.1 sampling method

This is a naive approach that performs a random sampling without replacement with uniform probability. For each curve, this method tends to sample fewer time points from the curve for dimension reduction. In this case, each curve after sampling will be a lot shorter such that it can be used as multivariate data for regular classification problem.

The graphs illustrate the patient 1's FA data after sampling method been applied to. Both of patent 1's pre-smoothed FA data and unsmoothed FA data were sampled. Judging from their shapes, the curves would still remain some of the important information if the remaining dimensions of the curves are carefully chosen.

Fig.6: Patient 1 FA measures

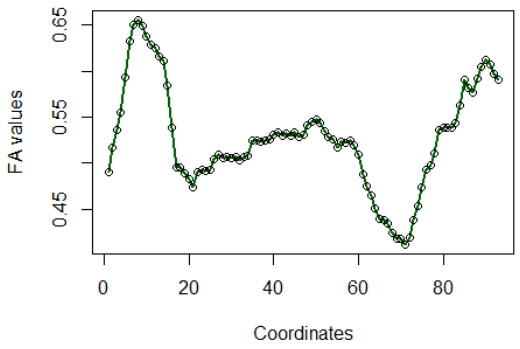
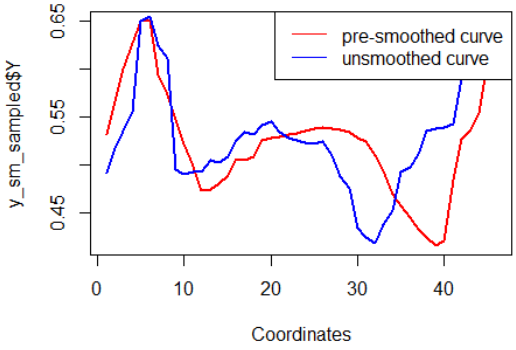


Fig.7: The curves after sampling



3.2.2 projection kernel

The idea of projection kernel is to project the infinite dimensional curves into a finite dimensional space such that the dimensionality is reduced significantly. If we define a d-dimensional subspace of Hilbert space to be V_d, then the orthogonal projection on V_d can be defined as:

$$P_{V_d}(X(t)) = \sum_{j=1}^{k} < X(t), \phi_k(t) > \phi_k(t)$$

where $< X(t), \phi_k(t) >= \int_T X(t)\phi(t)dt$ is the projection score and $\phi_k(t)$ is the basis function as defined before.

When the kernel is applied to every curve in the dataset, we can transfer the functional data problem into a multivariate analysis problem. With a carefully chosen dimension of the subspace V_d, each curve will remain informative information for an accurate prediction.

3.3 Prediction models

There are many machine learning models that performs classification in the literatures. In this project, I chose logistic regression and support vector machine along with their L1 regularized version to predict whether a patient has water diffusion using her FA measures. The intention for using L1 regularization is for further dimension reductions.

3.3.1 Logistic regression

Logistic regression model the log odds of probability of water diffusion using a linear function

$$log\frac{p_i}{1-p_i} = \beta_0 + \sum_{k=1}^{p} \beta_k x_{ik}$$

The coefficients are estimated by minimizing the negative log likelihood function

$$C(\beta) = -\sum_{i=1}^{N} y_i \, logp(x_i, \beta) + (1-y_i)log(1-p(x_i, \beta))$$

With L1 penalty,

$$\hat{\beta} = argmin_\beta C(\beta) + \lambda \sum_{k}^{p} |\beta_k|$$

3.3.2 Support Vector Machine

The model predict value based on the decision rule:

$$\hat{f}(x_i) = \hat{\beta}_0 + \sum_{k=1}^{p} \hat{\beta}_k x_{ik}$$

And the coefficients are estimated by minimizing the squared error loss, ie.

$$\hat{\beta} = argmin \sum_{i=1}^{n} [\, max(0, 1 - y_i(\beta_0 + \sum_{k=1}^{p} \beta_k x_{ik})]^2$$

With L1 penalty,

$$\hat{\beta} = argmin \sum_{i=1}^{n} [\, max(0, 1 - y_i(\beta_0 + \sum_{k=1}^{p} \beta_k \, x_{ik})]^2 + \lambda \sum_{k=1}^{p} |\, \beta_k|$$

4. Experiment Results

The raw dataset was splitted into 70% of training set and 30% of test set. And a smoothed duplicate of each was made for comparing the power of smoothing. Then two version of the dimension reduction techniques, sampling and projection kernel, were separately applied to the curves in each pair of training set and testing set. The dimensions of the curves after using sampling method or projection kernel and regularization strength were tuning parameters. Massive grid searches were performed for tuning each model.

Using the corresponding testing sets, the final prediction accuracies for patients' water diffusion statuses were summarized in the tables below.

Type of data	logistic with sampling	logistic with projection
Unsmoothed	0.76	0.78
Smoothed	0.78	0.85

Type of data	l1-logistic with sampling	l1-logistic with projection
Unsmoothed	0.85	0.88
Smoothed	0.85	0.85

Type of data	svm with sampling	svm with projection
Unsmoothed	0.80	0.76
Smoothed	0.76	0.80

Type of data	l1-svm with sampling	l1-svm with projection
Unsmoothed	0.88	0.88
Smoothed	0.90	0.90

After comparing the prediction results of using the smoothed data and unsmoothed data, one can easily spot that the bootstrap smoothing techniques will improve the prediction accuracy for each model.

For the same model, projection kernels did not necessarily outperform the sampling methods. Evidence was that the logistic regression, l1-logistic regression and l1-svm models combined with projection kernel achieved higher prediction accuracies, but non-regularized SVM models didn't reveal such a property. That is likely due to the

coordinates on the curves are sampled with equal probabilities. When the number of the remaining coordinates are relatively large as shown in Fig.7, each reduced curve is informative enough for prediction.

L1-regularized models were generally outperforming the non-regularized models. This is mainly due to the models can self-select the coordinates rather than taking in all of the variables generated from the sampling or from the projection kernels.

5. Discussion

In this study, I implemented a bootstrap smoothing method that can smooth the raw input data and generate better prediction accuracy. A sampling method and projection kernel method were also implemented for effect dimension reduction.

Their effect on producing prediction result were tested using four non-regularized/regularized models. The l1-svm model combined with projection kernel produced the highest accuracy that 90% of the patients were correctly diagnosed.

However, some of the limitations in this study are that, for example, the smoothing and projection kernels used pre-specified basis functions, in this case, B-spline basis. One cannot tell which type of basis is the best basis to use. What's more, the prediction models used are linear predictors. One can try using different models which can incorporate complex interactions and non-linear relationships of the input data to seek better prediction results.

6. References

Biau G.,Bunea F.,Wegkamp M.H.(2005). Functional classification in Hilbert spaces. IEEE Transactions on Information Theory. 51, 6.

Besse P., Ramsay J.(1986) Principle component analysis of sampled curves. Psychometrica 51,258-311.

Rossi F., Villa N. (2006). Support vector machine for functional data classification. Neurocomputing. 69(7-9), 730-742 (2006b).

Lila E., Aston J.A.D., Sangalli L.M. (2017) Functional data analysis of neuroimaging signals associated with cerebral activity in the brain cortex. In: Aneiros G., G. Bongiorno E., Cao R., Vieu P. (eds) Functional Statistics and Related Fields. Contributions to Statistics. Springer, Cham

Goldsmith J., Scheipl F., Huang l., Wrobel J., Di C., Gellar J., Harezlak J., McLean M.W., Swihart B., Xiao L., Crainiceanu C., Reiss P.T., Chen Y., Greven S., Huo L., Kundu M.G., Park S.Y., Miller D L., Staicu A.(2019). refund: Regression with Functional Data. R package version 0.1-21. https://CRAN.R-project.org/package=refund

Alexander A., Lee J.E., Lazar M., Field A.S. (2007). Diffusion Tensor Imaging of the Brain. Neurotherapeutics. 2007 Jul; 4(3): 316–329.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. (2008). LIBLINEAR: A library for large linear classification Journal of Machine Learning Research 9, 1871-1874.

Ramsay J.O., Silverman B.W. (2005). Functional Data Analysis (Second Edition). Springer Series in Statistics.

7. Appendix

The codes used in this study are listed here.

```
library(refund)
library(fda)
library(future)
library(caret)
library(fdp)
library(e1071)
library(glmnet)
library(LiblineaR)
data(DTI)
attach(DTI)
DTI <- subset(DTI, visit == 1)
row_missing <- apply(is.na(DTI$cca),MARGIN=1,any)
DTI <- subset(DTI, !row_missing)
outcome <- DTI$case
# Y is m by n, n obs, each obs is of length m
Y <- t(DTI$cca)
```

```
n <- length(unique(DTI$ID))
m <- length(1:dim(DTI$cca)[2])

matplot(1:m, t(DTI$cca),
      type='l', lty=1,
      col=rainbow(n),
      main = "Fig.1: FA measures of water diffusion for all patients",
      xlab="coordinates", ylab="Fractional anisotropy (FA)")

par(mfrow=c(1,2))

eval1_result <- evaluate_y(Y[,1])

plot(1:length(Y[,1]), Y[,1], type="l", cex=2,
   xlab="Coordinates",
   ylab="FA values",
   main="Fig.2: Patient 1's FA measures",
   lwd=2,
   col='darkgreen',
   ylim=range(Y[,1]))
points(1:length(Y[,1]), Y[,1])



plot(eval1_result$K_pool, eval1_result$Bias2, type="n", cex=2,
   xlab="Number of basis functions",
   ylab="Total squared error",
   main="Fig.3: Patient 1's Bias Variance trade off",
   ylim=range(cbind(eval1_result$Bias2, eval1_result$MSE,eval1_result$Var)))
```

```
points(eval1_result$K_pool, eval1_result$Bias2, type='b', col="black", lwd=2)
points(eval1_result$K_pool, eval1_result$Var, type='b', col="blue", lwd=2)
points(eval1_result$K_pool, eval1_result$MSE, type='b', col="red", lwd=2)
legend("topright", legend = c("MSE", "Bias", "Variance"), col =
c('red','black',"blue"),lty=5, pch=1)



y_sm <- smooth_one_y(Y[,1])
Y_sm <- smooth_all(Y)

par(mfrow=c(1,2))
plot(1:length(Y[,1]), Y[,1], cex=1,
    xlab="Coordinates",
    ylab="Total squared error",
    main="Fig.4: Patient 1's smoothed FA",
    ylim=range(Y[,1]))
points(1:length(Y[,1]),y_sm, type='l', col="red", lwd=2)

matplot(1:length(Y[,1]), Y_sm,
       type='l', lty=1,
       col=rainbow(n),
       main = "Fig.5: Smoothed FA measures, 143 patients",
       xlab="Coordinates",
       ylab="Fractional anisotropy (FA)")

y_sm <- smooth_one_y(Y[,1])
y_sm_sampled <- sample_functional_data(y_sm)
```

```
y_sampled <- sample_functional_data(Y[,1])

par(mfrow=c(1,2))
plot(1:length(Y[,1]), Y[,1], type="l", cex=2,
    xlab="Coordinates",
    ylab="FA values",
    main="Fig.6: Patient 1 FA measures",
    lwd=2,
    col='darkgreen',
    ylim=range(Y[,1]))
points(1:length(Y[,1]), Y[,1])

plot(1:length(y_sm_sampled$Y),
    y_sm_sampled$Y,
    type='l',lwd=2,col='red',
    xlim = range(c(1:length(y_sampled$Y)),
                c(1:length(y_sm_sampled$Y))),
    main='Fig.7: The curves after sampling',
    xlab="Coordinates")
points(1:length(y_sampled$Y), y_sampled$Y, type='l',col='blue', lwd=2)
legend("topright",
      legend = c("pre-smoothed curve", "unsmoothed curve"),
      col = c('red',"blue"),lty=1)
set.seed(20679485)
options(warning=-1)

d_choices <- 20:80
```

```
result_lr_unsm_sampled <- rep(NA, length(d_choices))
result_l1lr_unsm_sampled <- rep(NA, length(d_choices))
result_svm_unsm_sampled <- rep(NA, length(d_choices))
result_l1svm_unsm_sampled <- rep(NA, length(d_choices))

result_lr_sm_sampled <- rep(NA, length(d_choices))
result_l1lr_sm_sampled <- rep(NA, length(d_choices))
result_svm_sm_sampled <- rep(NA, length(d_choices))
result_l1svm_sm_sampled <- rep(NA, length(d_choices))

result_lr_unsm_projected <- rep(NA, length(d_choices))
result_l1lr_unsm_projected <- rep(NA, length(d_choices))
result_svm_unsm_projected <- rep(NA, length(d_choices))
result_l1svm_unsm_projected <- rep(NA, length(d_choices))

result_lr_sm_projected <- rep(NA, length(d_choices))
result_l1lr_sm_projected <- rep(NA, length(d_choices))
result_svm_sm_projected <- rep(NA, length(d_choices))
result_l1svm_sm_projected <- rep(NA, length(d_choices))

i=0
for(d in d_choices){
i=i+1
# unsmoothed , sampled data
Y_sampled <- sample_functional_data(Y,d=d)$Y
dat_unsm_sam = data.frame(t(Y_sampled), y = as.factor(outcome))
train_index <- createDataPartition(dat_unsm_sam$y, p = .7, list = FALSE)
train_unsm_sampled <- dat_unsm_sam[train_index,] # 100 by (46 + 1)
```

```
test_unsm_sampled  <- dat_unsm_sam[-train_index,] # 41 by 47


# unsmoothed , projected data
Y_projected <- project_functional_data(Y,d=d)$scores
dat_unsm_proj = data.frame(t(Y_projected), y = as.factor(outcome))
train_index <- createDataPartition(dat_unsm_proj$y, p = .7, list = FALSE)
train_unsm_projected <- dat_unsm_proj[train_index,] # 100 by (46 + 1)
test_unsm_projected  <- dat_unsm_proj[-train_index,] # 41 by 47
#head(train_unsm_projected)


# smoothed , sampled data
Y_smoothed_sampled <- sample_functional_data(Y_sm,d=d)$Y
dat_smoothed_sam = data.frame(t(Y_smoothed_sampled), y = as.factor(outcome))
train_smoothed_index <- createDataPartition(dat_smoothed_sam$y, p = .7, list =
FALSE)
train_sm_sampled <- dat_smoothed_sam[train_smoothed_index,]
test_sm_sampled  <- dat_smoothed_sam[-train_smoothed_index,]


# smoothed , projected data
Y_smoothed_projected <- project_functional_data(Y_sm,d=d)$scores


dat_smoothed_proj = data.frame(t(Y_smoothed_projected), y = as.factor(outcome))
train_smoothed_index <- createDataPartition(dat_smoothed_proj$y, p = .7, list =
FALSE)
train_sm_projected <- dat_smoothed_proj[train_smoothed_index,]
test_sm_projected  <- dat_smoothed_proj[-train_smoothed_index,]


###### lr
```

```
# unsmoothed sampling with lr


lrfit_unsm_sampled = glm(y ~ .,family=binomial, data = train_unsm_sampled)

predict_lr_unsm_sampled <- predict(lrfit_unsm_sampled, test_unsm_sampled,
type="response")

predict_lr_unsm_sampled[predict_lr_unsm_sampled < 0.5] <- 0

predict_lr_unsm_sampled[predict_lr_unsm_sampled >= 0.5] <- 1

# accuracy

acc <- sum(predict_lr_unsm_sampled ==
test_unsm_sampled$y)/length(test_unsm_sampled$y)

result_lr_unsm_sampled[i] <- acc


# unsmoothed sampling with l1-lr

d <- dim(train_unsm_sampled)[2]

l1lrfit_unsm_sampled <- glmnet(x=as.matrix(train_unsm_sampled[,1:(d-1)])
,y=train_unsm_sampled[,d],family=c('binomial'), alpha=1 )

predict_l1lr_unsm_sampled <- predict(l1lrfit_unsm_sampled,
newx=as.matrix(test_unsm_sampled[,1:(d-1)]), type="response", s= 0.05)

predict_l1lr_unsm_sampled[predict_l1lr_unsm_sampled < 0.5] <- 0

predict_l1lr_unsm_sampled[predict_l1lr_unsm_sampled >= 0.5] <- 1

# accuracy

acc <- sum(predict_l1lr_unsm_sampled ==
test_unsm_sampled$y)/length(test_unsm_sampled$y)

result_l1lr_unsm_sampled[i] <- acc


# smoothed sampling with lr


lrfit_sm_sampled = glm(y ~ .,family=binomial, data = train_sm_sampled)

predict_lr_sm_sampled <- predict(lrfit_sm_sampled, test_sm_sampled)

predict_lr_sm_sampled[predict_lr_sm_sampled < 0.5] <- 0
```

```
predict_lr_sm_sampled[predict_lr_sm_sampled >= 0.5] <- 1

# accuracy

acc <- sum(predict_lr_sm_sampled ==
test_sm_sampled$y)/length(test_sm_sampled$y)

result_lr_sm_sampled[i] <- acc


# smoothed sampling with l1-lr

d <- dim(train_sm_sampled)[2]

l1lrfit_sm_sampled <- glmnet(x=as.matrix(train_sm_sampled[,1:(d-1)])
,y=train_sm_sampled[,d],family=c('binomial'), alpha=1)


predict_l1lr_sm_sampled <- predict(l1lrfit_sm_sampled,
newx=as.matrix(test_sm_sampled[,1:(d-1)]), type="response", s= 0.05)

predict_l1lr_sm_sampled[predict_l1lr_sm_sampled < 0.5] <- 0

predict_l1lr_sm_sampled[predict_l1lr_sm_sampled >= 0.5] <- 1

# accuracy

acc <- sum(predict_l1lr_sm_sampled ==
test_sm_sampled$y)/length(test_sm_sampled$y)

result_l1lr_sm_sampled[i] <- acc


# unsmoothed projection with lr


lrfit_unsm_projected = glm(y ~ .,family=binomial, data = train_unsm_projected)

predict_lr_unsm_projected <- predict(lrfit_unsm_projected, test_unsm_projected,
type="response")

predict_lr_unsm_projected[predict_lr_unsm_projected < 0.5] <- 0

predict_lr_unsm_projected[predict_lr_unsm_projected >= 0.5] <- 1

# accuracy

acc <- sum(predict_lr_unsm_projected ==
test_unsm_projected$y)/length(test_unsm_projected$y)
```

```
result_lr_unsm_projected[i] <- acc


# unsmoothed projection with l1-lr

d <- dim(train_unsm_projected)[2]

l1lrfit_unsm_projected <- glmnet(x=as.matrix(train_unsm_projected[,1:(d-1)])
,y=train_unsm_projected[,d],family=c('binomial'), alpha=1 )

predict_l1lr_unsm_projected <- predict(l1lrfit_unsm_projected,
newx=as.matrix(test_unsm_projected[,1:(d-1)]), type="response", s= 0.05)

predict_l1lr_unsm_projected[predict_l1lr_unsm_projected < 0.5] <- 0

predict_l1lr_unsm_projected[predict_l1lr_unsm_projected >= 0.5] <- 1

# accuracy

acc <- sum(predict_l1lr_unsm_projected ==
test_unsm_projected$y)/length(test_unsm_projected$y)

result_l1lr_unsm_projected[i] <- acc


# smoothed projection with lr


lrfit_sm_projected = glm(y ~ .,family=binomial, data = train_sm_projected)

predict_lr_sm_projected <- predict(lrfit_sm_projected, test_sm_projected)

predict_lr_sm_projected[predict_lr_sm_projected < 0.5] <- 0

predict_lr_sm_projected[predict_lr_sm_projected >= 0.5] <- 1

# accuracy

acc <- sum(predict_lr_sm_projected ==
test_sm_projected$y)/length(test_sm_projected$y)

result_lr_sm_projected[i] <- acc


# smoothed projection with l1-lr

d <- dim(train_sm_projected)[2]

l1lrfit_sm_projected <- glmnet(x=as.matrix(train_sm_projected[,1:(d-1)])
,y=train_sm_projected[,d],family=c('binomial'), alpha=1)
```

```
predict_l1lr_sm_projected <- predict(l1lrfit_sm_projected,
newx=as.matrix(test_sm_projected[,1:(d-1)]), type="response", s= 0.05)

predict_l1lr_sm_projected[predict_l1lr_sm_projected < 0.5] <- 0

predict_l1lr_sm_projected[predict_l1lr_sm_projected >= 0.5] <- 1

# accuracy

acc <- sum(predict_l1lr_sm_projected ==
test_sm_projected$y)/length(test_sm_projected$y)

result_l1lr_sm_projected[i] <- acc


# svm


# unsmoothed sampling with svm

svmfit_unsm_sampled = svm(y ~ ., data = train_unsm_sampled, cost = 10, scale =
F)#radial basis

predict_svm_unsm_sampled <- predict(svmfit_unsm_sampled, test_unsm_sampled)

# accuracy

acc <- sum(predict_svm_unsm_sampled ==
test_unsm_sampled$y)/length(test_unsm_sampled$y)

result_svm_unsm_sampled[i] <- acc


# unsmoothed sampling with l1-svm

l1svmfit_unsm_sampled  <- LiblineaR(data=as.matrix(train_unsm_sampled[,1:(d-
1)]),
                        target=train_unsm_sampled$y,
                        type=5, cost = 10)#radial basis


predict_l1svm_unsm_sampled <- predict(l1svmfit_unsm_sampled,
test_unsm_sampled[,1:(d-1)])$predictions

# accuracy

acc <- sum(predict_l1svm_unsm_sampled ==
test_unsm_sampled$y)/length(test_unsm_sampled$y)
```

```
result_l1svm_unsm_sampled[i] <- acc


# smoothed sampling with svm

svmfit_sm_sampled = svm(y ~ ., data = train_sm_sampled, cost = 10, scale =
F)#radial basis

predict_svm_sm_sampled <- predict(svmfit_sm_sampled, test_sm_sampled)

# accuracy

acc <- sum(predict_svm_sm_sampled ==
test_sm_sampled$y)/length(test_sm_sampled$y)

result_svm_sm_sampled[i] <- acc


# smoothed sampling with l1-svm

l1svmfit_sm_sampled  <- LiblineaR(data=as.matrix(train_sm_sampled[,1:(d-1)]),

                    target=train_sm_sampled$y,

                    type=5, cost = 10)#radial basis

predict_l1svm_sm_sampled <- predict(l1svmfit_sm_sampled, test_sm_sampled[,1:(d-
1)])$predictions

# accuracy

acc <- sum(predict_l1svm_sm_sampled ==
test_sm_sampled$y)/length(test_sm_sampled$y)

result_l1svm_sm_sampled[i] <- acc


# unsmoothed projection with svm

svmfit_unsm_projected = svm(y ~ ., data = train_unsm_projected, cost = 10, scale =
F)#radial basis

predict_svm_unsm_projected <- predict(svmfit_unsm_projected,
test_unsm_projected)

# accuracy

acc <- sum(predict_svm_unsm_projected ==
test_unsm_projected$y)/length(test_unsm_projected$y)

result_svm_unsm_projected[i] <- acc
```

```
# unsmoothed projection with l1-svm

l1svmfit_unsm_projected  <- LiblineaR(data=as.matrix(train_unsm_projected[,1:(d-1)]),

                        target=train_unsm_projected$y,

                        type=5, cost = 10)#radial basis

predict_l1svm_unsm_projected <- predict(l1svmfit_unsm_projected, test_unsm_projected[,1:(d-1)])$predictions

# accuracy

acc <- sum(predict_l1svm_unsm_projected == test_unsm_projected$y)/length(test_unsm_projected$y)

result_l1svm_unsm_projected[i] <- acc


# smoothed projection with svm

svmfit_sm_projected = svm(y ~ ., data = train_sm_projected, cost = 10, scale = F)#radial basis

predict_svm_sm_projected <- predict(svmfit_sm_projected, test_sm_projected)

# accuracy

acc <- sum(predict_svm_sm_projected == test_sm_projected$y)/length(test_sm_projected$y)

result_svm_sm_projected[i] <- acc


# smoothed projection with l1-svm

l1svmfit_sm_projected  <- LiblineaR(data=as.matrix(train_sm_projected[,1:(d-1)]),

                        target=train_sm_projected$y,

                        type=5, cost = 10)#radial basis


predict_l1svm_sm_projected <- predict(l1svmfit_sm_projected, test_sm_projected[,1:(d-1)])$predictions

# accuracy
```

```
acc <- sum(predict_l1svm_sm_projected ==
test_sm_projected$y)/length(test_sm_projected$y)

result_l1svm_sm_projected[i] <- acc

}


# results

max(result_lr_unsm_sampled)

max(result_l1lr_unsm_sampled)

max(result_svm_unsm_sampled)

max(result_l1svm_unsm_sampled)

max(result_lr_sm_sampled)

max(result_l1lr_sm_sampled)

max(result_svm_sm_sampled)

max(result_l1svm_sm_sampled)

max(result_lr_unsm_projected)

max(result_l1lr_unsm_projected)

max(result_svm_unsm_projected)

max(result_l1svm_unsm_projected)

max(result_lr_sm_projected)

max(result_l1lr_sm_projected)

max(result_svm_sm_projected)

max(result_l1svm_sm_projected)
```