M.S. THESIS

Classification for Functional Data using Functional Principal Component Analysis

함수형 주성분분석을 활용한 함수형 데이터 분류 문제 해결

BY

Seungyeop Hyun

AUGUST 2023

DEPARTMENT OF STATISTICS
COLLEGE OF SCIENCE
SEOUL NATIONAL UNIVERSITY

Classification for Functional Data using Functional Principal Component Analysis

함수형 주성분분석을 활용한 함수형 데이터 분류 문제 해결

지도교수 PARK JUN YONG

이 논문을 이학석사 학위논문으로 제출함

2023년 6월

서울대학교 대학원

통계학과

현 승 엽

현 승 엽의 이학석사 학위논문을 인준함

2023년 8월

위 원 장	이 재 용	(인)
부위원장	PARK JUN YONG	(인)
위 원	박 건 웅	(인)

Abstract

Functional data refers to multivariate data with an ordering on the dimensions. In dealing with such data, it is more effective to apply methods that consider the high dimensionality and inter-variable correlations characteristic of functional data, rather than conventional multivariate analysis approaches. FPCA, as a method of Functional Data Analysis (FDA), is a principal component analysis technique specifically tailored for functional data. It considers the order of variables and the continuity of data, making it an effective approach for analyzing functional datasets. In this thesis, we will compare the performance of several machine learning models for solving the binary classification problem on three different functional datasets. This comparison will shed light on the effectiveness of machine learning models and the application of FPCA when addressing binary classification problems with functional data.

Keywords: Classification, Functional Data Analysis, Functional Principal Component Analysis

Student Number: 2021-22845

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Functional data is defined as "multivariate data with an ordering on the dimensions"[10]. It refers to data in the form of curves represented by functions on a continuum, such as time, frequency, or wavelength. Due to the inherent high-dimensionality of functional data, which can approach infinity, there are typically fewer observations than the number of observed time points. Additionally, considering the measurement process, it is evident that there are high correlations between variables.

Therefore, applying traditional multivariate analysis methods poses challenges, and Functional Data Analysis (FDA) becomes necessary for analyzing functional data. Functional Principal Component Analysis (FPCA) is a method that applies PCA to functional data and is one of the techniques in FDA. Unlike conventional PCA, FPCA considers the order between variables and calculates principal components that account for the variations in functional data. Moreover, FPCA takes into account the continuity of data collected discretely, making it a more effective analysis method. FPCA has found extensive appli-

cations in diverse domains, such as finance and medical research. Moreover, in recent times, it has also been used as a method for feature extraction in image and video analysis.

In this thesis, we will address the binary classification problem with functional data using various machine learning algorithms. Next, we will compare the performance of Logistic Regression, Random Forest, XGBoost, and Fused Lasso models for functional data classification. Finally, we will apply FPCA to summarize the data and investigate how the performance of the three models 'Logistic Regression, Random Forest, and XGBoost' changes with the reduced data, excluding Fused Lasso, which inherently considers the order of adjacent variables and is fitted with the original data. We will conduct this analysis on three distinct datasets: Sleeping Energy Expenditure (SEE) [9], Earthquake, and Strawberry data[1], and compare the results to draw meaningful conclusions.

Chapter 2 explains the models used to solve the classification problem. Moving on to Chapter 3, we provide the analysis results for each of the three datasets. Chapter 4 provides the conclusions drawn from our study and the appendix includes the R and Python code utilized in the analysis.

Chapter 2

Methodology

2.1 Logistic Regression

Logistic regression is a commonly used model for solving binary classification problems. For a binary response variable $y \in \{0, 1\}$, we can assume that

$$y_i \overset{\text{ind}}{\sim} B(n, p_i) \qquad \text{for } i = 1, 2, \cdots, n \tag{2.1}$$

where $p_i \in [0, 1]$

The logit parameter λ is defined as

$$\lambda = \log\left(\frac{p}{1-p}\right) = \text{logit}(p) \tag{2.2}$$

with λ increasing from $-\infty$ to ∞ as p increases from 0 to 1. For a k-dimensional explanatory variable $\mathbf{x} \in \mathbb{R}^k$, logistic regression model can be described by the following equation:

$$\lambda(\mathbf{x}) = \text{logit}(p(\mathbf{x})) = \beta_0 + \boldsymbol{\beta}^{\mathbf{T}}\mathbf{x}, \quad \boldsymbol{\beta} \in \mathbb{R}^k \tag{2.3}$$

and it equals to

$$p(\mathbf{x}) = \frac{e^{\beta_0 + \boldsymbol{\beta}^{\mathbf{T}}\mathbf{x}}}{1 + e^{\beta_0 + \boldsymbol{\beta}^{\mathbf{T}}\mathbf{x}}}. \tag{2.4}$$

In this model, the logit function is called a 'link function' which models the probability of the event as a function of $\mathbf{x}$. There is an advantage of using logit transformation that λ isn' t restricted to the range $[0, 1]$, ensuring that the above model never approaches limited territory.

We can obtain β_0 and $\boldsymbol{\beta}$ using a maximum likelihood estimation. From (2.1), a log likelihood function is defined as

$$\log \mathcal{L}(\mathbf{y}; \mathbf{x}, \beta_0, \boldsymbol{\beta}) = \sum_{i=1}^{n} \Big[p_i \log y_i + (1 - y_i) \log(1 - p_i) \Big] \tag{2.5}$$

where $\mathbf{y} = (y_1, y_2, \cdots, y_n)$ and β_0, $\boldsymbol{\beta}$ are calculated by optimization methods like iteratively reweighted least squares (IRLS) as follows:

$$(\hat{\beta}_0, \hat{\boldsymbol{\beta}}) = \underset{\beta_0, \boldsymbol{\beta}}{\arg\max} \sum_{i=1}^{n} \Big[p_i \log y_i + (1 - y_i) \log(1 - p_i) \Big] \tag{2.6}$$

where $p_i = p(\mathbf{x}_i) = \frac{\exp(\beta_0 + \boldsymbol{\beta}^{\mathbf{T}}\mathbf{x}_i)}{1 + \exp(\beta_0 + \boldsymbol{\beta}^{\mathbf{T}}\mathbf{x}_i)}$

2.2 Random Forest

Breiman proposed a bootstrap aggregating(bagging)[5] that learns several decision trees with a bootstrap data and returns the result by aggregating the output of trees. Bagging aims to train different individual decision trees. But it has a limitation: if there are important features that can divide the data well, they can be repeatedly used in most of the decision trees.

Afterward, he suggested a more powerful model, random forest[6], which decreases the correlation of each tree and increases the strength of individual trees. Random forest uses some random elements to solve the problem. The

method selects the variables from a part of the whole feature group which is randomly chosen. If there are p features in a data set, the group of candidates typically consists of $m = \sqrt{p}$ or $m = p/3$ number of features. This solution helps the individual trees not to be too correlated and prevents overfitting. The algorithm[8] for training the random forest model is below.

Algorithm 1 Random Forest for Binary Classification

1. Given training data set $\boldsymbol{d} = (\mathbf{X}, \mathbf{y})$. Fix $m \leq p$ and the number of trees B.

2. For $b = 1, 2, ..., B$, do the following.

   (a) Create a bootstrap version of the training data $\boldsymbol{d}_b^*$, by randomly sampling n observations with replacement n.

   (b) Select m candidate variables of the p features at random.

   (c) Grow a maximal-depth tree $\hat{\mathbf{r}}_b(x)$ using the data in $\boldsymbol{d}_b^*$.

3. For binary classification, perform a majority voting. Count the number of trees and output k which has a larger value between N_0 and N_1

$$N_k = (\text{the number of trees predicting class k}), \qquad k = 0, 1$$

And the ensemble method has the disadvantage of being difficult to interpret. Random forest clears it up with variable importance using the measures like the gini index[4], permutation importance[2] etc.

2.3 Fused Lasso

Tibshirani proposed a regularized regression 'least absolute shrinkage and selection operator' called lasso[11].

A standard linear model is

$$y_i = \sum_j x_{ij}\beta_j + \epsilon_i \tag{2.7}$$

where $\epsilon_i \sim (0, \sigma^2)$, $y_j \in \mathbb{R}$ and $x_i = (x_{i1}, \cdots, x_{ip})$. The predictors are assumed to have a mean of 0 and a variance of 1 and the targets y_i have a mean 0.

Lasso finds the solution $\hat{\boldsymbol{\beta}} = (\hat{\beta}_1, \hat{\beta}_2, \cdots, \hat{\beta}_p)$ satisfying the conditions below

$$\hat{\boldsymbol{\beta}} = \text{argmin} \left\{ \sum_i \left(y_i \sum_j x_{ij}\beta_j \right)^2 \right\} \quad \text{subject to} \sum_j |\beta_j| \leq s \tag{2.8}$$

where the bound s is a tuning parameter.

Unlike other regularized regression models such as ridge and principal component regression, lasso chooses some important values and drops the others giving 0 to the coefficients. It has the effect of variable selection and helps to interpret the fitted model. And it works well for high dimensional settings where p is large.

A weak point of the lasso is that it overlooks the ordering of the features. Tibshirani presented the fused lasso improving the lasso[12]. It has an additional penalty term that causes the sparsity in differences between the coefficients. Specifically, it encourages the piecewise constant solution

$$\begin{gathered} \hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\text{argmin}} \left\{ \sum_i \left(y_i \sum_j x_{ij}\beta_j \right)^2 \right\} \\ \text{subject to} \sum_j |\beta_j| \leq s_1 \text{ and } \sum_{j=2}^{p} |\beta_j - \beta_{j-1}| \leq s_2. \end{gathered} \tag{2.9}$$

Fused lasso particularly assumes the case where $p >> n$.

2.4 XGBoost

XGBoost, short for 'extreme gradient boosting', was proposed by Tianqi Chen and Carlos Guestrin[7]. It is a faster and more powerful method that improves the shortcomings of previous algorithms.

Boosting is an ensemble technique that makes a strong classifier using a number of weak classifiers. A tree model is usually used as the weak classifier and the boosting algorithm sequentially learns each weak tree. We use the weak trees learned up to previous steps to fit the next one.

A tree ensemble model uses the summation of K trees to predict the result. For a data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$ ($|\mathcal{D}| = n,\ \mathbf{x}_i \in \mathbb{R}^m.\ y_i \in \mathbb{R}$) with n observations and m features, the model is expressed by

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F} \tag{2.10}$$

where $\mathcal{F} = \{f(\mathbf{x}) = w_{q(\mathbf{x})}\}(q : \mathbb{R}^m \to T, w \in \mathbb{R}^T)$ is the space of basic tree model.

XGBoost uses the following *regularized* objective function and it helps to fix the over-fitting problem in the former methods[7]

$$\begin{aligned} \mathcal{L}^{(t)}(\phi) = \sum_i l(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t) \\ \text{where } \Omega(f_t) = \gamma T + \frac{1}{2}\lambda ||w||^2. \end{aligned} \tag{2.11}$$

l is a differentiable convex loss function that measures the difference between the prediction and the target. And the Ω imposes a penalty on the complexity of the tree model. It prevents the tree model to have complex structures by controlling the number of leaf nodes, T. And it also shrinks the weight of each leaf node to avoid the over-fitting problem. When the penalty term goes to zero, the objective is the same as the existing gradient tree boosting.

However, the objective (2.11) cannot be optimized using existing optimization methods in Euclidean space. The second-order approximation is a solution to the optimization problem

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \Omega(f_t) \tag{2.12}$$

where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ and $h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ are the first and second derivative of the loss function.

We can take a simpler objective by deleting the constant term which leads to

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \Omega(f_t). \tag{2.13}$$

Set $I_j = \{i | q(\mathbf{x}_i) = j\}$ as the observation set of the leaf j. Considering the structure of the trees, the equation (2.13) can be changed by

$$\begin{aligned} \tilde{\mathcal{L}}^{(t)} &= \sum_{i=1}^{n} \left[g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T. \end{aligned} \tag{2.14}$$

For the fixed tree structure $q(\mathbf{x})$, we can take the optimal weight w_j^* and calculate the optimal value of $\tilde{\mathcal{L}}^{(t)}$ which are

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} + \lambda}, \tag{2.15}$$

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T. \tag{2.16}$$

All the possible candidates for the tree structure q cannot be considered. So we can use a greedy algorithm that iteratively adds branches to the tree starting from a single leaf node. Then we can use the equation (2.7) as the scoring function to evaluate the quality of q.

Let I_L, I_R be the instance sets of the left and right leaf nodes after the split. Assuming $I = I_L + I_R$, the loss reduction after the split is

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{2.17}$$

and we assess the split candidates.

In addition, XGBoost fits the more accurate model in a shorter time using a few split finding algorithms and system design for parallel computing.

2.5 Functional Principal Component Analysis

Now we think about principal component analysis (PCA) for multivariate situations. In this case, we select the weights emphasizing the variation which are represented in the data. Then we define the principal components analysis through a stepwise procedure and it chooses the sets of normalized weights maximizing the variation in the f_i's below.

1. Find the weights $\boldsymbol{\xi}_1 = (\xi_{11}, \xi_{21}, \cdots, \xi_{p1})'$ for the linear combinations

$$f_{i1} = \sum_j \xi_{j1} x_{ij} = \boldsymbol{\xi}_1' x_i$$

   which have the largest mean square $N^{-1}\sum_i f_{i1}^2$ subject to

$$\sum_j \xi_{j1}^2 = ||\boldsymbol{\xi}_1||^2 = 1.$$

2. Implement the second and next steps up to the number of p. On the m step, calculate a new weight vector $\boldsymbol{\xi}_m = (\xi_{1m}, \xi_{2m}, \cdots, \xi_{pm})$ and a new value $f_{im} = \boldsymbol{\xi}_m' x_i$. And the f_{im} has maximum mean square which is subject to the $||\boldsymbol{\xi}_m||^2$ and the additional $m-1$ constraints

$$\sum_j \xi_{jk}\xi_{jm} = \boldsymbol{\xi}_k'\boldsymbol{\xi}_m = 0,\ k < m.$$

Let's see the first step of the procedure. We find the most important variation in the variables by maximizing the mean square. And the constraint on the weights helps the problem be well defined. The values of the mean squares could be larger and larger without the unit sum of squares constraint.

In the following steps, we also look for the most important modes of variation. And we take the weights which are orthogonal to those found previously.

The linear combinations f_{im} are called *principal component scores*. And they describe what the components mean in terms of representing the variation in the data.

For a general case, let the matrix $\mathbf{X} \in \mathbb{R}^{N \times p}$ and $\boldsymbol{\xi} \in \mathbb{R}^p$. We find the $\boldsymbol{\xi}$ that satisfies,

$$\max_{\boldsymbol{\xi}'\boldsymbol{\xi}=1} N^{-1}\boldsymbol{\xi}'\mathbf{X}'\mathbf{X}\boldsymbol{\xi}.$$

And it can be rewritten by using the sample variance-covariance matrix $\mathbf{V} = N^{-1}\mathbf{X}'\mathbf{X}$,

$$\max_{\boldsymbol{\xi}'\boldsymbol{\xi}=1} \boldsymbol{\xi}'\mathbf{V}\boldsymbol{\xi}.$$

Finally, the problem of finding $\boldsymbol{\xi}$ is equivalent to the problem of finding $\boldsymbol{\xi}$ with largest eigen value ρ for

$$\mathbf{V}\boldsymbol{\xi} = \rho\boldsymbol{\xi}.$$

Now, let's see how PCA works for the functional data. For the functions $\xi(s)$ and $x(s)$, we can combine them by integrating over s to define the inner product,

$$\int \xi x = \int \xi(s)x(s)ds.$$

And then, we can express the principal component scores matching the weight ξ are

$$f_i = \int \xi x_i = \int \xi(s)x_i(s)ds.$$

At first, we choose the weight function $\xi_1(s)$ maximizing $N^{-1}\sum_i f_{i1}^2$. And it is equivalent to maximizing $N^{-1}\sum_i(\int \xi_1 x_i)^2$. Additionally, it has the unit sum of squares constraint $\int \xi_1(s)^2 ds$. ξ_i would be selected similarly for $i <= p$.

As for the multivariate case, ξ_m have to satisfy the orthogonality condition,

$$\int \xi_k \xi_m = 0,\ k < m.$$

The weight function $\xi(s)$ can be found in the way for multivariate cases. Let the covariance function $v(s,t)$,

$$v(s,t) = N^{-1}\sum_{i=1}^{N} x_i(s)x_i(t).$$

Now we can say $\xi(s)$ are the functions that satisfy

$$\int v(s,t)\xi(t)dt = \rho\xi(s). \tag{2.18}$$

The left side of the above equation is *integral transform* V of the weight function ξ defined by

$$V\xi = \int v(\cdot,t)\xi(t)dt$$

and the equation (2.12) can be represented below

$$V\xi = \rho\xi$$

which looks the same as the multivariate case.

Finally, we are going to use the principal component scores f_i as the new variables from the original data.

Chapter 3

Application

In this chapter, we use three real datasets to compare the accuracy of our proposed methods. We are going to analyze Sleeping Energy Expenditure(SEE) dataset, strawberry, and earthquake datasets. For each dataset, we compare the method in various conditions.

3.1 SEE dataset

SEE of participants, 109 children, and adolescents whose ages range from 5 to 18, were measured by every minute. Energy expenditure was guessed from a measured respiratory exchange of carbon dioxide for oxygen. And the topic of sufficient sleep is very important during childhood for metabolic health and physical and cognitive development. There are 63 boys and 46 girls and the 3 observations are deleted due to insufficient SEE data.

We want to classify the two groups, obese and non-obese, based on Body Mass Index(BMI). A person having a BMI larger than the 95th percentile is

classified as obese. In SEE the dataset, there are 44 obese and 62 non-obese participants in total 106. SEE was measured every minute from the starting points of sleep for participants. We have the data for $T = 405$ time points for each participant. The starting point was defined by monitoring activity and heart rate.

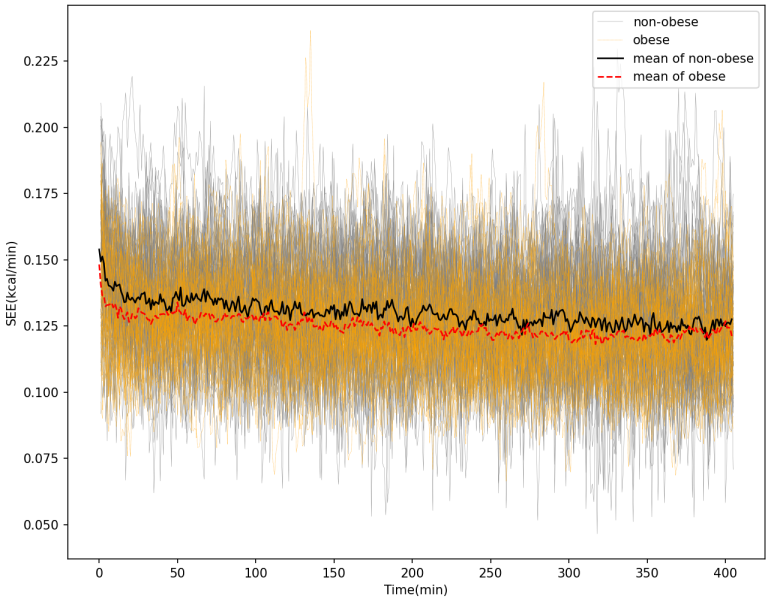

Figure 3.1: Plot of all observations and their pointwise means for obese and non-obese classes.

In Figure 3.1, there is the visualization of SEE data for all the participants and the point-wise mean for two groups. We can find the non-obese group has higher fluctuations than obese group. But the higher fluctuation means that there is larger noise, and it makes our classification tasks difficult.

We used functional principal component analysis(FPCA) for dimension re-

duction of functional data. And we adopt Logistic Regression, Random Forest, and XGBoost for both the original dataset and the reduced one. Considering the characteristics of the fused lasso, we use original SEE data for it. Accuracy was computed by 10-fold cross validation with an average of 200 iterations. Table 3.1 is the result of the analysis.

Model	Full	FPCA*	FPCA†
Logistic Reg	0.5945	0.6327	0.6236
Random Forest	0.6101	0.5965	0.5982
XGBoost	0.6120	0.6102	0.6078
Fused Lasso	0.5714	-	-

Table 3.1: Classification accuracy for SEE based on each model.
* monomial basis, † bspline basis

In the SEE dataset, logistic regression achieves a higher accuracy when using the data with FPCA applied. With the monomial basis in FPCA, the highest score recorded is 63.27%. In contrast, for Random Forest and XGBoost, using the original data without applying FPCA results in better performance. We can see the logistic regression model with FPCA is better than the other machine learning models in this case. And fused lasso shows the lowest accuracy in this case.

3.2 Earthquake dataset

Earthquake data consists of measurement data for major earthquakes from 1967 to 2003, taken from Northern California Earthquake Data Center. Each data is an averaged reading for one hour. It defines a major event as any reading of over 5 on the Rictor scale

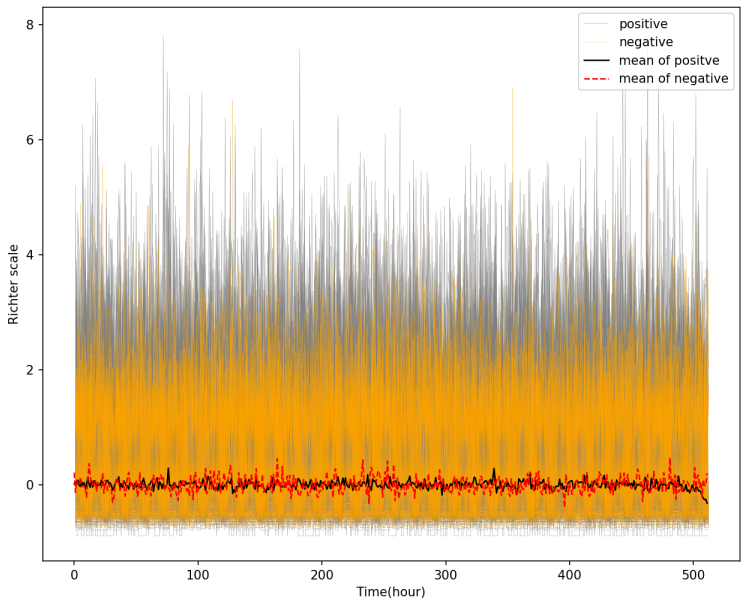

Figure 3.2: Plot of all observations and their pointwise means for positive and negative classes.

The objective is to predict 'whether a major event is about to occur based on the most recent readings in the surrounding area'[1]. Major events are often followed by aftershocks. Positive case means 'one where a major event is not preceded by another major event for at least 512 hours' and the negative case

is considered as 'instances where there is a reading below 4 that is preceded by at least 20 readings in the previous 512 hours that are non-zero'[1].

Training and testing data set consist of each 322 and 139 observations. Training data has 58 positive cases and 264 negative and testing has 35, 104 each. All the samples are measured at $T = 512$ time points. And as in Figure 3.2, we can see a larger fluctuation in positive cases than in negetive.

Model	Full	FPCA*	FPCA†
Logistic Reg	0.7645	0.8229	0.8229
Random Forest	0.8312	0.7593	0.7343
XGBoost	0.8468	0.6812	0.7406
Fused Lasso	0.7396	-	-

Table 3.2: Classification accuracy for Earthquake based on each model.
* monomial basis, † bspline basis

In the earthquake dataset, logistic regression also shows performance improvment when applying FPCA. Additionally, Random Forest and XGBoost recorded better accuracy when using the original data, with XGBoost achieving the highest accuracy of 84.68%. On the other hand, Fused lasso exhibited the lowest performance compared to other methods.

3.3 Strawberry dataset

A food spectrograph is a method used for classifying food types to ensure food safety and quality assurance. The Strawberry dataset contains measurements

obtained from performing spectroscopy on strawberry purees. Using the spectroscopy results, we can classify whether the composition of the puree is strawberry or non-strawberry. The objective is to classify the authenticity of the purees based on this data.

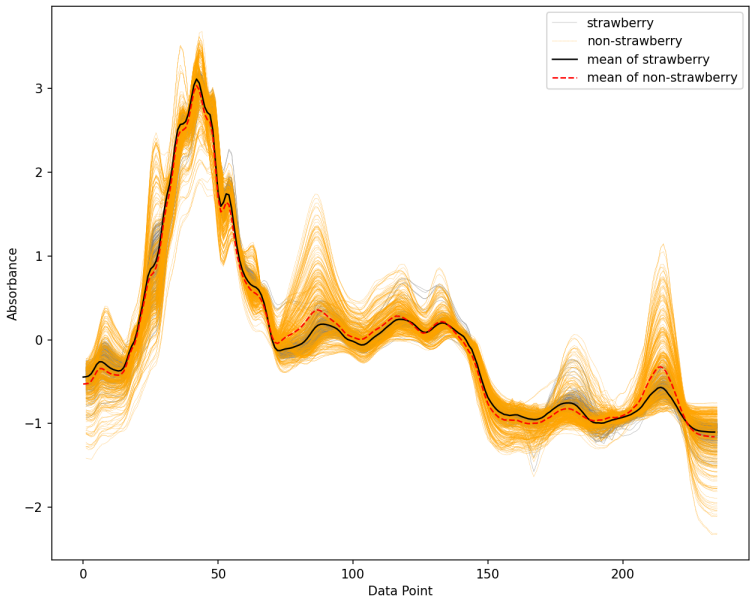

Figure 3.3: Plot of all observations and their pointwise means for strawberry and non-strawberry classes.

As measured in $T = 235$ time points, training data consists of 219 strawberry class and 394 non-strawberry class. For testing data, we have 370 observations which are 132 strawberries and 238 non-strawberries. There is a plot in Figure 3.3, we can see the mean functions for each class of purees. Both classes show a similar trend in their mean values, but it can be observed that non-strawberry exhibits larger fluctuations.

Model	Full	FPCA*	FPCA†
Logistic Reg	0.9167	0.9392	0.9252
Random Forest	0.9307	0.8087	0.8153
XGBoost	0.9516	0.8431	0.8478
Fused Lasso	0.9032	-	-

Table 3.3: Classification accuracy for Strawberry based on each model.
* monomial basis, † bspline basis

Similarly, when using data with FPCA applied, Logistic Regression showed an improvement in performance. Random Forest and XGBoost performed better when using the original data, with XGBoost achieving the highest accuracy of 95.16%. Fused lasso exhibited better performance than FPCA-applied Random Forest and XGBoost, but it still showed the lowest performance among all four models.

Chapter 4

Conclusion

The binary classification problem with functional data was solved using several machine learning methods, along with the application of Functional Principal Component Analysis(FPCA). Functional data is characterized by high dimensionality and strong correlations between variables, making it challenging to handle using simple multivariate approaches. FPCA was employed to effectively reduce the dimensionality of the functional data, allowing it to be processed more efficiently. After applying FPCA to summarize the functional data into lower dimensions, Logistic Regression, Random Forest, and XGBoost were fitted for classification. On the other hand, the fused lasso method, which takes into account the sequential order of adjacent variables, was fitted with the original data.

In particular, applying FPCA to Logistic Regression proved to be effective. When utilizing data with FPCA applied, the accuracy consistently increased compared to the non-FPCA preprocessed data. Furthermore, the performance either surpassed or remained on par with other machine learning models. We

can think that FPCA successfully summarized the functional data into a lower-dimensional representation. However, for XGBoost and Random Forest, using the original data resulted in better performance. This can be attributed to the characteristic of these models as ensemble techniques, where they combine results from multiple independent individual trees. As a result, they have a relative robustness to correlations between variables.

In reality, a multitude of high-dimensional functional datasets exists. There are some examples such as time series data and image data and these types of data are often subject to functional data analysis. Moreover, for data accumulating based on continuous parameters, such as time, we can also apply functional data analysis to it. In subsequent research, besides FPCA, alternative approaches like the Generalized Functional Linear Model and Dynamic Time Warping will be explored and applied to enhance the performance of functional data analysis models. This pursuit aims to advance the understanding of methods for elevating the efficacy of modeling when dealing with functional datasets.

Appendix A

Codes

A.1 Python Code for Random Forest

```
from sklearn.ensemble import RandomForestClassifier

import skfda
from skfda.preprocessing.dim_reduction import FPCA
from skfda.representation.grid import FDataGrid
from skfda.representation.basis import BSpline, Monomial

## Full model
n_estimators = [200, 300, 400]
max_depth = [6, 8, 10, 12]
min_samples_leaf = [8,12,18]
min_samples_split = list(range(30, 50, 2))

# repeat M
acc_mean_rf_full_SEE = []
for i in tqdm(range(M)):
```

```
        rf_clf = RandomForestClassifier(
                        random_state = seed * i,
                        n_jobs = -1,
                        n_estimators=random.choice(n_estimators),
                        max_depth=random.choice(max_depth),
                        min_samples_leaf=random.choice(min_samples_leaf),
                        min_samples_split=random.choice(min_samples_split)
                )
        scores = cross_val_score(rf_clf, X, y, cv = 10)
        acc_mean_rf_full_SEE.append(scores.mean())


    ## FPCA (Monomial basis)
    n_components = 5
    n_basis = 5

    # X to grid-data
    X_mat = X.to_numpy()
    grid_points = np.arange(1, X_mat.shape[1]+1)
    fd = FDataGrid(X_mat, grid_points)
    fd_M = fd.to_basis(Monomial(n_basis = n_basis))

    F = FPCA(n_components)
    pc_M_SEE = F.fit(fd_M)  # compute principal components
    pc_score_M_SEE = pc_M_SEE.transform(fd_M)   # and their scores
    X_fpca_M = pd.DataFrame(pc_score_M_SEE)

    n_estimators = [100, 200, 300]
    max_depth = [3,5,7]
    min_samples_leaf = [6,8,10,12,14]
    min_samples_split = [4, 6, 8,16]

    # repeat M
```

```
acc_mean_rf_monomial_SEE = []
for i in tqdm(range(M)):
    rf_clf = RandomForestClassifier(
                    random_state = seed * i,
                    n_jobs = -1,
                    n_estimators=random.choice(n_estimators),
                    max_depth=random.choice(max_depth),
                    min_samples_leaf=random.choice(min_samples_leaf),
                    min_samples_split=random.choice(min_samples_split)
            )
    scores = cross_val_score(rf_clf, X_fpca_M, y, cv = 10)
    acc_mean_rf_monomial_SEE.append(scores.mean())

## FPCA (Bspline basis)
n_components = 5
n_basis = 5

# X to grid-data
X_mat = X.to_numpy()
grid_points = np.arange(1, X_mat.shape[1]+1)

fd = FDataGrid(X_mat, grid_points)
fd_B = fd.to_basis(BSpline(n_basis = n_basis))

F = FPCA(n_components)
pc_B_SEE = F.fit(fd_B)  # compute principal components
pc_score_B_SEE = pc_B_SEE.transform(fd_B)   # and their scores
X_fpca_B = pd.DataFrame(pc_score_B_SEE)

# repeat M
acc_mean_rf_bspline_SEE = []
for i in tqdm(range(M)):
    rf_clf = RandomForestClassifier(
```

```
                            random_state = seed * i,
                            n_jobs = -1,
                            n_estimators=random.choice(n_estimators),
                            max_depth=random.choice(max_depth),
                            min_samples_leaf=random.choice(min_samples_leaf),
                            min_samples_split=random.choice(min_samples_split)
                    )
            scores = cross_val_score(rf_clf, X_fpca_M, y, cv = 10)
            acc_mean_rf_bspline_SEE.append(scores.mean())
```

A.2 Python Code for XGBoost

```
from xgboost import XGBClassifier

import skfda
from skfda.preprocessing.dim_reduction import FPCA
from skfda.representation.grid import FDataGrid
from skfda.representation.basis import BSpline, Monomial

## Full Model
n_estimators = [200, 250, 300]
max_depth = [3,5,7]

# repeat M
acc_mean_xgb_full_SEE = []
for i in tqdm(range(M)):
    xgb_clf = XGBClassifier(random_state = seed * i,
                            n_jobs = -1,
                            learning_rate = 0.1,
                            n_estimators = random.choice(n_estimators),
                            max_depth = random.choice(max_depth))
    scores = cross_val_score(xgb_clf, X, y, cv = 10)
    acc_mean_xgb_full_SEE.append(scores.mean())
```

```python


## FPCA (Monomial basis)
X_fpca_M = pd.DataFrame(pc_score_M_SEE)

n_estimators = [100, 200, 300]
max_depth = [3,5,7]

# repeat M
acc_mean_xgb_monomial_SEE = []
for i in tqdm(range(M)):
    xgb_clf = XGBClassifier(random_state = seed * i,
                            n_jobs = -1,
                            learning_rate = 0.1,
                            n_estimators = random.choice(n_estimators),
                            max_depth = random.choice(max_depth))
    scores = cross_val_score(xgb_clf, X_fpca_M, y, cv = 10)
    acc_mean_xgb_monomial_SEE.append(scores.mean())


## FPCA (B-spline basis)
X_fpca_B = pd.DataFrame(pc_score_B_SEE)

n_estimators = [100, 200, 300]
max_depth = [3,5,7]

# repeat M
acc_mean_xgb_bspline_SEE = []
for i in tqdm(range(M)):
    xgb_clf = XGBClassifier(random_state = seed * i,
                            n_jobs = -1,
                            learning_rate = 0.1,
                            n_estimators = random.choice(n_estimators),
```

```
                                    max_depth = random.choice(max_depth))
            scores = cross_val_score(xgb_clf, X_fpca_B, y, cv = 10)
            acc_mean_xgb_bspline_SEE.append(scores.mean())
```

A.3 R Code for Classification based on Fused Lasso

```
# Packages
library(genlasso)
library(caret)

### Fused lasso on a custom graph
n = nrow(data); p = ncol(data)-2
D = matrix(0, p-1, p)
for (i in 1:p-1) {
D[i,i] = 1
D[i,i+1] = -1
}

### fused lasso with 10-fold cross validation
gamma = seq(0, .5, 0.1)
cutoff = seq(0.3, 0.7, 0.05)

best_F1 = 0
best_lambda = 0
best_gamma = 0
best_cutoff = 0
best_metrics = 0

k = 10 # k-fold
idx = createFolds(y, k=k, list=TRUE, returnTrain = FALSE)
updated = 0
for (i in 1:k) {
i = unlist(idx[i])
```

```
        X_train <- as.matrix(X[-i,]); X_val <- as.matrix(X[i,])
        y_train <- y[-i]; y_val <- y[i]

        for (g in 1:length(gamma)) {
            f = fusedlasso(y_train, X_train, D, gamma = gamma[g],
                           approx = FALSE, maxsteps = 1000, minlam = 0)
            coef0 = coef(f, nlam=15)

            for (l in 1:length(coef0$lambda)) {
            pred = predict(f, lambda = coef0$lambda[l], Xnew = X_val)$fit
            for (c in 1:length(cutoff)) {
                pred_bin = ( (exp(pred) / (1 + exp(pred))) >= cutoff[c] )
                metrics = confusionMatrix(
                            factor(pred_bin, levels = c("TRUE", "FALSE")),
                            factor(y_val == 1, levels = c("TRUE", "FALSE"))
                          )

                F1 = metrics$byClass['F1']

                if (F1 > best_F1) {
                    updated = updated + 1
                    best_F1 = F1
                    best_lambda = coef0$lambda[l]
                    best_gamma = gamma[g]
                    best_cutoff = cutoff[c]
                    best_metrics = metrics
                }
            }
            }
        }
    }
```

Bibliography

[1] A. Bagnall, J. Lines, W. Vickers, and E Keogh. *The UEA and UCR Time Series Classification Repository*, 2019. http://www.timeseriesclassification.com.

[2] André Altmann, Laura Toloşi, Oliver Sander, and Thomas Lengauer. Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10):1340–1347, 04 2010.

[3] Seungchul Baek, Yewon Kim, Junyong Park, and Jong Soo Lee. Revisit to functional data analysis of sleeping energy expenditure. *Journal of Applied Statistics*, 49(4):988–1002, 2022.

[4] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[5] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[6] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[7] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference*

on Knowledge Discovery and Data Mining, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[8] Bradley Efron and Trevor Hastie. *Computer Age Statistical Inference: Algorithms, Evidence, and Data Science*. Institute of Mathematical Statistics Monographs. Cambridge University Press, 2016.

[9] Jong Soo Lee, Issa F Zakeri, and Nancy F Butte. Functional data analysis of sleeping energy expenditure. *PLoS One*, 12(5):e0177286, May 2017.

[10] James Ramsay and Giles Hooker. *Dynamic Data Analysis*. 01 2017.

[11] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society* (*Series B*), 58:267–288, 1996.

[12] Robert Tibshirani, Michael Saunders, S Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B* (*Statistical Methodology*), 67(1):91–108, 2005.

초록

함수형 데이터는 각 차원의 순서가 있는 다변량 데이터를 말한다. 함수형 데이터를 다룰 때는 기존의 다변량 분석 접근 방식보다 함수형 데이터의 고차원성과 변수 간 상관관계를 고려하는 방법을 적용하면 효과적이다. 함수형 주성분 분석은 함수형 데이터 분석의 한 방법으로서, 변수의 순서와 데이터의 연속성을 고려하여 함수형 데이터에 적용할 수 있게 고안된 주성분 분석 방법이다. 이 논문에서는 여러 가지 머신러닝 모델을 이용하여 세 가지의 함수형 데이터셋에 대한 이진 분류 문제를 해결하고 각 모형의 성능을 비교한다. 이를 통해 함수형 데이터에 대한 머신러닝 모델의 성능과 함수형 주성분 분석의 효과를 확인한다.

주요어: 분류, 함수형 데이터 분석, 함수형 주성분 분석

학번: 2021-22845