UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO FIN DE GRADO

Análisis de datos funcionales: representación en bases y regresión funcional en scikit-fda

Autor: Pablo Pérez Manso
Tutor: Alberto Suárez González

Junio 2019

Pablo Pérez Manso
Análisis de datos funcionales: representación en bases y regresión funcional en scikit-fda

Pablo Pérez Manso

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A Blanca,
el amor de mi corazón.

Talk is cheap.
Show me the code.

Linus Torvalds

Agradecimientos

En primer lugar, me gustaría agradecer a todas las personas con las que he compartido este proyecto. A Alberto por proponerme y acompañarme en uno de los retos más duros de mi vida. A Carlos, por dejarlo todo en cada momento de necesidad para empujar con toda su fuerza, esto es gracias a ti. A Jose Luis, por tantas dudas resueltas. A Pablo y Amanda, por hacer piña y correr de la mano juntos en este proyecto.

A todas las personas que he conocido en la universidad, que no han sido pocas en todo este tiempo. A Cris, Mikel, Jorge, Rubén, Marcos, Jose Luis, Guille... y toda esa gente que comparte tantas horas contigo que son ya familia. A mis amigos del barrio, que me picaron una y otra vez hasta que he conseguido sacarlo, os debo una ronda.

A Andrea, la mejor amiga que me ha dado la vida, por tener siempre razón y enseñarme todo lo bueno que hay en mí.

A mi familia, por estar siempre a mi lado. A mi madrina Diana y a mi tía Macu por preocuparse por mí y por esta carrera más de lo que yo lo he hecho y porque siempre creyeron que podía hacer cosas imposibles, y no se equivocaron. A mi hermano Miguel, a Almu y a Blanca, por enseñarme que la familia es lo más importante en este mundo.

A mis abuelos, mis segundos padres, que perdí durante esta aventura y me hubiera gustado que hoy me pudieran acompañar. Ojalá os sintáis tan orgullosos de mi hoy como yo estoy de todo lo que me habéis enseñado. Lo que soy hoy es gracias a vosotros.

Y sobre todo a mis padres, que lo han dado todo por mí y confiaron en lo que valía, aunque a veces ni yo lo hiciera, muchísimas gracias. Y sobre todo por aguantarme diariamente. Yo me aguanto a mí mismo, y sé lo duro que ha sido.

Resumen

El análisis de datos funcionales es una rama de la estadística cuyo objetivo es el estudio de datos que dependen de parámetros continuos, como curvas (por ejemplo, series temporales) o superficies (por ejemplo, cantidades que dependen de coordenadas geográficas). Dada la abundancia de aplicaciones en los que este tipo de datos son relevantes (por ejemplo, la medicina, la meteorología o la geoestadística) existe un gran interés por el desarrollo de herramientas estadísticas y computacionales para su tratamiento y análisis. Es en este ámbito en el que se enmarca el proyecto actual, cuyo objetivo es el desarrollo de métodos de regresión funcional que han sido integrados en el paquete Python *scikit-fda*.

La regresión es un procedimiento estadístico cuyo objetivo es modelizar la relación que existe entre una o más variables explicativas con una variable de respuesta. En este trabajo, las variables explicativas consideradas pertenecen a un espacio de funciones. La variable de respuesta puede ser tanto escalar como funcional. Se considerará únicamente el caso en el que la relación entre las variables explicativas y las de respuesta es lineal. El objetivo principal en un problema de regresión consiste en estimar los parámetros que caracterizan dicha relación. Para problemas regresión multivariante estos parámetros son escalares o matrices. Sin embargo, en el caso de la regresión funcional los correspondientes parámetros pueden ser funciones; es decir tienen una dimensión infinita. Con el fin de abordar esta dificultad, las funciones son aproximadas mediante desarrollos en una base de tamaño finito. De esta manera, se obtiene una representación de los datos de partida como vectores de coeficientes, cuyo tamaño es finito. En este trabajo se han ampliado las operaciones que se pueden realizar entre funciones en este sistema de representación. En concreto se han desarrollado operaciones aritméticas como la suma, resta y multiplicación, el producto interno y la derivación.

La técnica utilizada para realizar la regresión es el método de mínimos cuadrados, que busca minimizar la suma de cuadrados de la diferencia entre el valor real de la variable respuesta y su valor predicho. Así tras plantear el modelo a resolver y aplicar el producto interno para medir la influencia que tienen las diferentes funciones de las bases unas sobre otras, se reduce finalmente el problema a la resolución de un sistema de ecuaciones en el que las incógnitas son los coeficientes de los parámetros en su correspondiente expansión en una base.

La implementación de esta funcionalidad ha requerido modificaciones en el diseño de funciones y estructuras existentes en el paquete. Además, la implementación realizada de los métodos es compatible con las diferentes herramientas que integra el paquete *scikit-learn*. Asimismo, se ha realizado un diseño que permite incluir de manera sencilla otros métodos de regresión funcional en el paquete *scikit-fda* en el futuro.

Palabras clave

Análisis de datos funcionales, representación en bases, regresión funcional, Python

Abstract

Functional data analysis is a branch of statistics whose objective is the study of data that depend on continuous parameters, such as curves (e.g. time series) or surfaces (e.g. quantities that depend on geographical coordinates). Given the abundance of applications in which this type of data is relevant (for example, medicine, meteorology or geostatistics), there is great interest in developing statistical and computational tools for processing and analysis this data. It is in this field that the current project is framed, whose objective is the development of functional regression methods that have been integrated into the Python scikit-fda package.

Regression is a statistical procedure whose objective is to model the relationship between one or more explanatory variables and a response variable. In this work, the explanatory variables considered belong to a space of functions. The response variable can be either scalar or functional. Only the case in which the relationship between the explanatory variables and the response variables is linear will be considered. The main objective in a regression problem is to estimate the parameters that characterize this relationship. For multivariate regression problems these parameters are scalars or matrices. However, in the case of functional regression, the corresponding parameters can be functions; that is, they have an infinite dimension. In order to address this difficulty, the functions are approximated by finite size basis. In this way, a representation of the starting data is obtained as vectors of coefficients, whose size is finite. In this work, the operations that can be carried out between functions in this representation system have been extended. Specifically, arithmetic operations such as addition, subtraction and multiplication, internal product and derivation have been developed.

The technique used to perform the regression is the method of least squares, which seeks to minimize the sum of squares of the difference between the real value of the response variable and its predicted value. After proposing the model to be solved and applying the inner product to measure the influence that the different functions of the bases have on each other, the problem is finally reduced to the resolution of a system of equations in which the solution are the coefficients of the parameters in their corresponding expansion in a base.

The development of this functionality has required modifications to the design of existing functions and structures in the package. In addition, the implementation of the methods is compatible with the different tools that integrate the package *scikit-learn*. It has also been designed to easily include other functional regression methods in the package *scikit-fda* in the future.

Keywords

Functional data analysis, basis representation, functional regression, Python

Índice

Listas

Lista de figuras

1

Introducción

El análisis de datos funcionales es una rama de la estadística que se encarga del estudio de la información contenida en curvas, superficies y en general datos cuyo valor depende de un parámetro continuo. Este área busca desarrollar herramientas que nos permitan el trabajo con este tipo de datos, el diseño de métodos de representación que facilitan su entendimiento, el estudio de sus características o el análisis de su variabilidad.

La literatura sobre el análisis de datos funcionales es extensa, siendo la principal referencia el libro de J. O. Ramsay y B. W. Silverman *Functional data analysis* [1]. Otras referencias a tener en cuenta son el libro de P. Kokoszka y M. Reimherr, *Introduction to functional data analysis* [2] y el libro F. Ferraty and P. Vieu, *Nonparametric functional data analysis: theory and practice* [3]. A pesar de ello, como nos indican los propios Ramsay y Silverman en *Applied functional data analysis* [4], nos encontramos ante un área que está todavía en su infancia, por lo que el camino por recorrer es todavía largo.

Junto a las publicaciones que encontramos sobre el tema, se han desarrollado diferentes soluciones software para el trabajo con este tipo de datos. El principal en este área es el paquete *fda* [5] desarrollado por J. O. Ramsay e implementado en los lenguajes de R y Matlab, principales lenguajes orientados a entornos matemáticos. Además, extendiendo el trabajo realizado en éste, encontramos paquetes que complementan su funcionalidad, destacando otros como son *fda.usc* [6] o *refund* [7] también desarrollados en R.

Durante los últimos años, Python ha experimentado un aumento notable en su uso comparado con otros lenguajes de programación. Índices como *TIOBE* hacen muestra de este crecimiento, alcanzando Python los primeros puestos y sobrepasando a otros lenguajes de ámbito científico como R o Matlab. También *Github*, principal plataforma de almacenamiento de código, muestra en sus estadísticas anuales *The State of the Octoverse*, cómo Python se ha consolidado durante los últimos años como uno de los principales lenguajes de la plataforma, siendo actualmente el tercero en número total de repositorios.

Además, se ha colocado como referencia del ámbito científico gracias a una sintaxis clara, una curva de aprendizaje poco pronunciada y a librerías como *Numpy*, que facilita el trabajo de almacenamiento y operaciones con grandes cantidades de datos o *Scipy*, una colección de diferentes herramientas y

algoritmos de áreas matemáticas como la estadística y el álgebra. Su posicionamiento ha sido logrado gracias a su facilidad de uso y a la buena eficiencia que consiguen, ya que gran parte de las rutinas están implementadas en lenguajes como *C*, *C++* o *Fortran*.

A fin de complementar estas librerías se han desarrollado los *scikits*, diferentes paquetes que extienden la funcionalidad de éstos siendo el más importante *scikit-learn* [8] que desarrolla métodos de aprendizaje automático. Otros paquetes relativos a ámbitos más específicos son *scikit-bio* para bioinformática, *scikit-aero* para aeronáutica o *scikit-allel* para genética. Éstos cubren multitud de áreas, pero no existía ningún software específico para el trabajo con datos funcionales por lo que durante año 2018 se comenzó el proyecto de desarrollar un paquete en este área llevado a cabo por Miguel Carbajo [9].

1.1. Objetivos

El objetivo principal de este trabajo ha sido el desarrollo de técnicas de regresión, herramienta estadística que mide la relación que tiene un conjunto de variables sobre otra dada, aplicadas a datos funcionales. Además, se buscó desde su diseño la integración de los diferentes métodos implementados con las herramientas de la librería de referencia en este ámbito *scikit-learn* [8].

También se extendió la funcionalidad del paquete ampliando las herramientas para la representación de los datos funcionales mediante bases. Los principales retos en este área fueron la ampliación del número de bases con las que representar la información, desarrollar métodos como la suma, multiplicación o el producto interno para poder operar con nuestros datos o implementar herramientas como la derivación para obtener más información sobre nuestras muestras.

El proyecto se diseñó desde su inicio siguiendo la filosofía de código abierto para facilitar la colaboración en su desarrollo. Para poder llevar a cabo esta colaboración de forma que no se vea comprometida la integridad del proyecto, se configuraron herramientas encargadas de evaluar las diferentes aportaciones y asegurar su compatibilidad con el resto del código y el cumplimiento de los estándares definidos para el paquete.

1.2. Estructura del documento

Este trabajo está estructurado en tres partes. En el capítulo 2 se explica en qué consiste el problema de la regresión y diferentes métodos para la regresión con datos funcionales que se implementarán en el paquete. Para ellos se explican las diferentes formas de representarlos, qué ventajas tienen y se detallan todas las operaciones que son necesarias desarrollar para este propósito.

El capítulo 3 explica el proceso llevado a cabo para la implementación de estos métodos. Se co-

mienza por un análisis del estado del paquete y los requisitos del mismo, mostrando los diferentes módulos en los que se ha trabajado y que funcionalidad se ha añadido a cada uno. Se comentarán las decisiones llevadas a cabo a la hora de desarrollar la funcionalidad explicada, y cómo se ha documentado y probado la misma. Además, se comenta el modo en que se ha organizado el trabajo dentro del equipo de desarrollo y las herramientas utilizadas para llevar a cabo el mismo.

Finalmente, el capítulo 4 resume los objetivos que tenía este proyecto, los conocimientos adquiridos y se plantean los diferentes retos que éste deja para el futuro.

2

Análisis de datos funcionales: regresión

En este capítulo trataremos el problema de la regresión aplicado al análisis de datos funcionales que ya definimos en el capítulo anterior y el desarrollo de diferentes herramientas necesarias para llevarlo a cabo.

Comenzaremos viendo los procesos que existen para registrar este tipo de datos y las diferentes formas de representarlos. Estudiaremos la representación en bases, qué ventajas aporta frente a otras y las bases con las que trabaja. Además, se definen las diferentes operaciones que podemos realizar con esta representación como la suma o el producto interno y se desarrolla la derivación y su aplicación como operador lineal.

A continuación, se explica el problema de la regresión, cuál es su objetivo y las soluciones a la misma, para su posterior extensión al caso funcional. Para ello se estudian los casos en que la parte funcional es una variable explicativa (respuesta escalar) o una variable respuesta (respuesta funcional), explicando su motivación y desarrollando la técnica de mínimos cuadrados para cada uno de ellos.

2.1. Representación de Datos Funcionales en Bases

Imaginemos que queremos estudiar el clima de nuestro país. Para recabar datos, se reparten estaciones meteorológicas por diferentes partes del territorio que registran mediciones de temperatura, precipitaciones, velocidad del viento y otro tipo de indicadores. Si tratáramos estas mediciones desde el punto de vista del análisis de datos multivariante estándar, podríamos estudiarlas para dar respuesta a preguntas como cuál es la región más cálida o cuál tiene más probabilidad de sufrir una sequía debido a sus bajas precipitaciones.

Pero este punto de vista no puede estudiar otras como cuál es la variación de las temperaturas cuanto más al norte viajamos, cómo depende la velocidad media del viento en función de la altura a la que nos encontremos o relacionar el descenso de las precipitaciones con la proximidad de los meses de verano. Todas estas preguntas dependen de parámetros continuos, como el tiempo en el que se registran cada una de las mediciones o la latitud y altitud de las diferentes estaciones meteorológicas.

Actualmente, tenemos un número finito de estaciones que registran las mediciones y no podemos obtener información en cada punto del territorio de los diferentes valores. Pero, ¿qué ocurre con el tiempo? ¿Podemos registrar para cada tiempo t los diferentes parámetros que vamos a estudiar? Como comentan Kokoszka y Reimherr [2], el tiempo es un parámetro continuo y por tanto las mediciones de los diferentes indicadores conforman un conjunto infinito imposible de almacenar. Realizar una medición cada segundo, hora o día, proporciona una cantidad finita de observaciones, y aunque no se tenga registro de cada instante de tiempo t se sabe que los datos están definidos para los intervalos que hay entre cada una de ellas. El intervalo de tiempo entre mediciones nos lo define la naturaleza del problema a estudiar, por ejemplo, será interesante recoger la información cada minuto si se estudia la evolución a lo largo del día o diarias para un análisis del clima anual.

Para aclarar la notación a lo largo del trabajo, denotaremos por X a una muestra cualquiera dentro de un conjunto de muestras $\mathcal{X} = \cup\{X_n\}_{i=1}^{N}$ donde X_n es la n-ésima muestra.

Una vez registrados los datos, hay que estudiar cómo representarlos para poder trabajar con ellos. La forma natural es almacenar las observaciones como pares clave-valor, tiempo-observación en nuestro caso. A esta forma de registrar nuestros datos se la conoce como representación discreta. Así los datos obtenidos por todas las estaciones se representan de la siguiente manera

$$X_n = \{(t_j, x_j)\}_{j=1}^{J_n}, \quad T_{inicial} \leq t_1 < \ldots < t_{J_n} \leq T_{final}, \quad n = 1, \ldots, N.$$

Para cada una de las N muestras hay un número J_n de observaciones entre un $T_{inicial}$ y un T_{final}, que pueden estar registradas en valores del parámetro t diferentes.

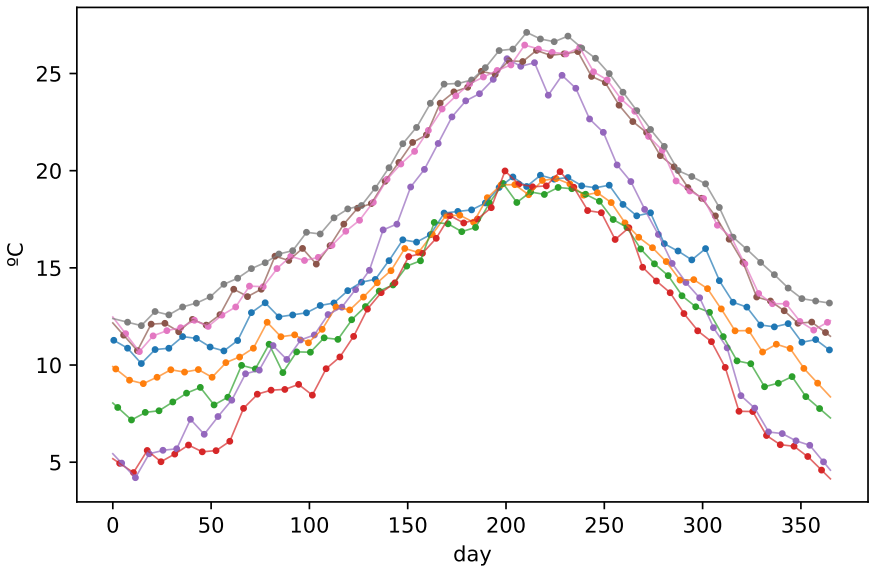


Figura 2.1: Representación discreta de la media de la temperatura semanal entre 1980-2009 realizadas por ocho estaciones meteorológicas en España - *Fuente*: Aemet.

Como observamos en la figura 2.1, esta representación registra un numero discreto de puntos para cada muestra, haciendo que cada gráfica sea una consecución de rectas interpoladas entre cada par de observaciones. Las mediciones son tomadas una vez por semana, pero observando la gráfica podemos comprobar que no todas están tomadas en el mismo día de la semana. Además, si queremos conocer información entre cada par de puntos, necesitaremos aumentar el número de observaciones que tenemos que registrar y almacenar, pudiendo llegar a ser grande el volumen de datos si aumentara también el número de muestras.

Existe otra forma de representar nuestros datos mediante la utilización de un sistema de bases de funciones. Para evitar el problema que ya comentamos para la representación discreta, estas bases serán truncadas a un número finito de funciones, pasando a representarse las muestras como combinaciones lineales de la siguiente manera

$$X_n(t) = \sum_{m=0}^{M} c_{nm}\phi_m(t) = \boldsymbol{c_n'}\boldsymbol{\phi_m}(t) \quad t \in [T_{inicial}, T_{final}] \quad n = 1, \ldots, N.$$

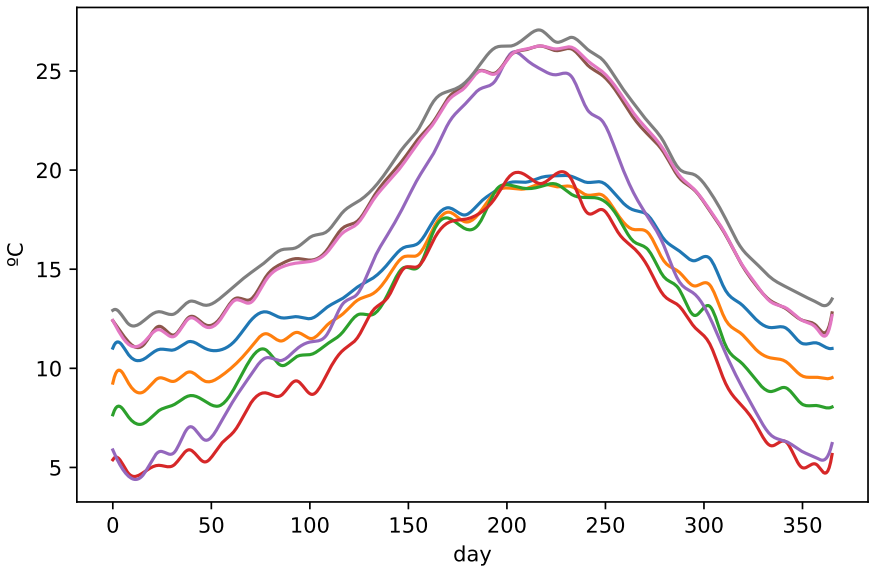


Figura 2.2: Representación en bases de funciones de la media de la temperatura semanal entre 1980-2009 realizadas por ocho estaciones meteorológicas en España - *Fuente*: Aemet.

A la base de funciones escogida se le supone cierta suavidad como podemos ver en la 2.2. El número de bases M escogido para representar nuestros datos es habitualmente menor que el número de observaciones J_n, reduciendo a un vector de coeficientes $\mathbf{c} = (c_1, \ldots, c_M)'$ la representación de nuestras muestras. Además, si cada muestra tiene observaciones en valores de t diferentes, el paso a un sistema de bases colocará las muestras en un dominio común, solucionando los problemas anteriormente descritos.

Para la representación en bases se utilizan diferentes familias de funciones, como por ejemplo la base de los monomios,

$$X(t) = \sum_{m=0}^{M} c_m t^m = c_0 + c_1 t + c_2 t^2 + \cdots + c_M t^M,$$

la base de Fourier, formada por funciones $\sin$ y $\cos$,

$$\phi_0 = \frac{1}{\sqrt{2}} \quad \phi_{2k-1} = sin\left(\frac{k}{\omega}\, t\right) \quad \phi_{2k} = cos\left(\frac{k}{\omega}\, t\right) \quad k = 1, \ldots, \frac{M-1}{2}$$

o la base B-spline de orden k. Los *splines* son curvas diferenciables definidas en intervalos por polinomios. Al conjunto de puntos ordenados $T = \{t_i\}_{i=0}^{l-1}$ que delimitan estos intervalos se los denomina nodos. Un spline se define como

$$X(t) = \sum_{m=0}^{k+l-2} c_m \phi_{m,k,T},$$

donde $\phi_{m,k,T}(t)$ es la función definida de forma recursiva como

$$\phi_{m,k,T}(t) = \frac{t - t_m}{t_{m+k} - t_m}\phi_{m,k-1,T}(t) + \frac{t_{m+k+1} - t}{t_{m+k+1} - t_{m+1}}\phi_{m+1,k-1,T}(t),$$

$$\phi_{m,1,T}(t) = \begin{cases} 1 \text{ si } t_m < t < t_{m+1} \\ 0 \text{ si } t \notin (t_m, t_{m+1}) \end{cases}$$

Una vez tenemos representados nuestros datos puede surgir la necesidad, por ejemplo, de sumar los datos de dos muestras para verlas en conjunto o multiplicar cada una de ellas por un factor dependiendo de la importancia que tenga cada una dentro del conjunto para poder compararlas. A continuación, veremos cómo llevar a cabo estas operaciones con nuestros datos representados en bases.

2.1.1. Operaciones: suma, producto, producto interno

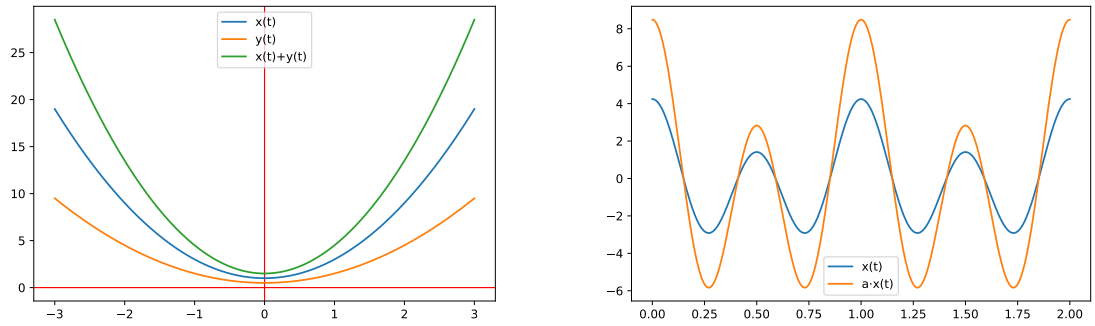
Una vez almacenados y representadas mediante bases las diferentes observaciones de las estaciones comenzamos a trabajar con los datos. Por ejemplo, queremos calcular las precipitaciones medias diarias a lo largo del año. Necesitamos definir las operaciones necesarias para poder sumar las precipitaciones de cada una de las estaciones para calcular el total o ponderar cada una de ellas para realizar el promedio.

Como vimos en la sección anterior nuestras muestras están representadas por un vector de coeficientes $\mathbf{c} = (c_1, \ldots, c_M)'$ y una base de funciones $\{\phi_1, \ldots, \phi_M\}$. Variando los coeficientes del vector

podemos expresar diferentes funciones con la misma base. Todas las posibles funciones que puede llegar a representar nuestra base forman un espacio vectorial.

Un espacio vectorial $(V, +, \cdot)$ sobre un cuerpo $\mathbb{K}$ es un conjunto no vacío dotado de una operación interna llamada suma ($V \times V \to V$) y otra externa llamada producto por escalar ($\mathbb{K} \times V \to V$), para los cuales es cerrado.

De esta manera podemos para dos funciones $x(t) = \sum_{i=0}^{M} x_i \phi_i(t)$ e $y(t) = \sum_{i=0}^{M} y_i \phi_i(t)$ definimos la suma como $x(t)+y(t) = \sum_{i=0}^{M} (x_i+y_i)\phi_i(t)$ y el producto por escalar como $a \cdot x(t) = \sum_{i=0}^{M} (a \cdot x_i)\phi_i(t)$ para $a \in \mathbb{R}$. En la figura 2.3 podemos ver representaciones gráficas de las operaciones que hemos descrito, la suma de dos funciones representadas en la base de los monomios (figura 2.3(a)) y el producto de una función de Fourier por un escalar (figura 2.3(b)),



(a) Suma

(b) Producto por escalar

Figura 2.3: Ejemplos de operaciones con datos representados en bases de funciones.

Además podemos definir sobre nuestras funciones una nueva operación, el producto interno, que extiende el espacio en el que están definidas al espacio prehilbertiano o espacio prehilbert, definido como un espacio vectorial V dotado de un producto interno $(V, \langle \cdot, \cdot \rangle)$.

El producto interno $\langle \cdot, \cdot \rangle$ es una operación binaria ($V \times V \to \mathbb{K}$) tal que dados los elementos u, v y w de un espacio vectorial cumple las siguientes propiedades:

1. $\langle \alpha u + \beta v, w \rangle = \alpha \langle u, w \rangle + \beta \langle v, w \rangle \quad \alpha, \beta \in \mathbb{K}$
2. $\langle v, w \rangle = \langle w, v \rangle$
3. $\langle v, v \rangle \geq 0$

   a) $\langle v, v \rangle = 0 \iff v = 0$

Para cada par de funciones $x(t)$ e $y(t)$ de nuestro espacio definimos su producto interno como

$$\langle x, y \rangle = \int_a^b x(t)y(t)dt$$

para un intervalo cerrado $[a, b]$.

Como vimos en la sección anterior, nuestras funciones están definidas como vectores sobre una base $\{\phi_1, \ldots, \phi_M\}$ de funciones. Decimos que una base es ortogonal si $\forall i, j$ tal que $i \neq j$ entonces $\langle \phi_i, \phi_j \rangle = 0$ con $i, j = 1, \ldots, M$. Si además se cumple que $\langle \phi_i, \phi_i \rangle = 1, \ \forall i = 1, \ldots, M$, entonces diremos que la base es ortonormal.

Para las bases que hemos visto antes, podemos comprobar que la base de los monomios no es ortogonal. Sin embargo, la base de Fourier, si el periodo de las funciones coincide con el intervalo de definición, la base es ortonormal.

Para cada par de elementos del espacio vectorial $x(t)$ e $y(t)$ definidos sobre la misma base $\{\phi_1(t), \ldots, \phi_n(t)\}$ podemos ver que su producto interno resulta

$$\langle x(t), y(t) \rangle = \langle \sum_i x_i \phi_i(t), \sum_j y_j \phi_j(t) \rangle = \sum_{i,j} \langle x_i \phi_i(t), y_j \phi_j(t) \rangle = \sum_{i,j} x_i y_j \langle \phi_i(t), \phi_j(t) \rangle.$$

Observamos que el producto interno de dos funciones depende del producto interno de los elementos de su base.

La matriz de Gram

Dado un espacio prehilbertiano $(V, \langle \cdot, \cdot \rangle)$ y una base del mismo $\{\phi_1, \phi_2, \ldots, \phi_M\}$ definimos la matriz de Gram G como la matriz cuadrada de dimensión $M \times M$

$$G = \begin{pmatrix} \langle \phi_1, \phi_1 \rangle & \ldots & \langle \phi_1, \phi_M \rangle \\ \vdots & \ddots & \vdots \\ \langle \phi_M, \phi_1 \rangle & \ldots & \langle \phi_M, \phi_M \rangle \end{pmatrix}.$$

Para cada par de elementos $x(t)$ e $y(t)$ del espacio vectorial, podemos definir su producto interno en base esta matriz como

$$\langle x(t), y(t) \rangle = \sum_{i,j} x_i y_j \langle \phi_i(t), \phi_j(t) \rangle = \mathbf{x}' \mathbf{G} \mathbf{y}.$$

La matriz de Gram tiene varias propiedades:

- Es una matriz simétrica, $\langle \phi_i, \phi_j \rangle = \langle \phi_j, \phi_i \rangle$.
- Es diagonal si la base es ortogonal, $\langle \phi_i, \phi_j \rangle = 0$.
- Es la matriz identidad si la base es ortonormal, $\langle \phi_i, \phi_i \rangle = 1$.

Además, podemos extender el producto interno mediante una matriz a todas las funciones, aunque

estén definidas sobre diferentes bases. Dadas las bases ϕ y $\boldsymbol{\theta}$, definimos la matriz asociada de los productos internos $J_{\boldsymbol{\phi\theta}}$ como $(J_{\boldsymbol{\phi\theta}})_{ij} = \langle \phi_i, \theta_j \rangle$. Así el producto interno de $x = \boldsymbol{x'}\boldsymbol{\phi}$ e $y = \boldsymbol{y'}\boldsymbol{\theta}$ se calcula como $\langle x, y \rangle = \boldsymbol{x'}\boldsymbol{J}_{\boldsymbol{\phi\theta}}\boldsymbol{y}$.

Ya hemos visto los diferentes métodos que tenemos, por ejemplo, para combinar la información de dos muestras o poder cambiar la escala para poder estudiar mejor nuestras observaciones. La diferenciación nos ayudará a evaluar cómo cambian las diferentes muestras a lo largo del tiempo, por lo que a continuación veremos cómo se aplica a nuestra representación en bases.

2.1.2. Diferenciación

Una vez ya tenemos preparados nuestros datos, estudiaremos cómo cambian estos a lo largo del tiempo. Evaluar en qué época del año crece más rápido el volumen de precipitaciones nos puede ayudar a prevenir posibles inundaciones, así como las bajadas repentinas de temperatura pueden indicar posibles olas de frio.

La diferenciación nos ayuda a estudiar cómo cambian nuestras muestras a lo largo del tiempo. Para nuestras funciones representadas en bases definimos la diferenciación u operador derivada de orden k como

$$X_n^{(k)}(t) = \sum_{k=1}^{M} c_{nm}\phi_m^{(k)}(t) = \mathbf{c_n'}\boldsymbol{\phi}^{(\mathbf{k})}(\mathbf{t})$$

Para poder realizar el cálculo de la k-ésima derivada se ha de tener en cuenta que las funciones que conforman la base tienen que ser también k veces derivables. La base de Fourier compuesta por senos y cosenos, es derivable infinitas veces con derivadas continuas. Sin embargo, la elección de la base de splines nos permite derivar tantas veces como el orden - 1 de la misma.

Un uso clásico de la derivada, por ejemplo, es el cálculo de máximos y mínimos de una función. En la figura 2.4 podemos ver una muestra y su derivada, señalando por líneas verticales rojas los puntos de corte de la derivada con el eje horizontal, que nos indican los puntos máximos y mínimos de nuestra muestra.

Una generalización de la operación de derivación son los operadores diferenciales. El operador diferencial más sencillo es la diferenciación que hemos visto, definido como $Dx(t) = x'(t)$. Estos operadores pueden ser aplicados de forma iterada ($D^2x(t) = D(Dx(t)) = D(x'(t)) = x''(t)$) pudiendo realizar el cálculo n veces, asumiendo que $x(t)$ es diferenciable ese número de veces. Además, el operador diferencial cumple una propiedad importante, $D^nD^m = D^{n+m}$.

Consideremos la ecuación diferencial ordinaria homogénea

$$x^{(n)}(t) + a_{n-1}x^{(n-1)}(t) + \cdots + a_0x(t) = 0.$$

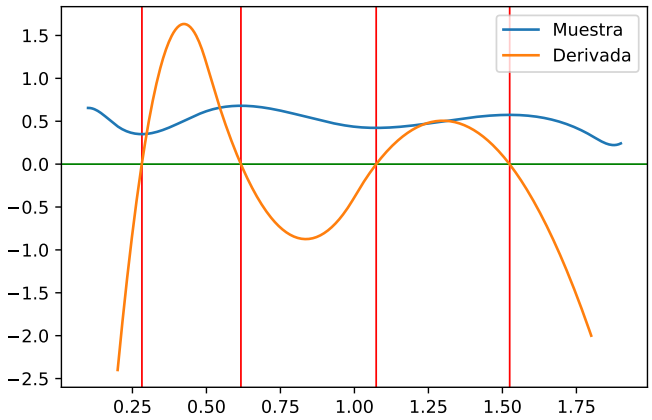


Figura 2.4: Ejemplo de una muestra y su derivada, resaltado los máximos y mínimos de la muestra identificados por la derivada.

Usando el operador diferencial D que hemos definido previamente podemos reescribir la ecuación como $L(D)x(t) = 0$ donde $L(D)$ es el operador lineal diferencial definido como

$$L(D) = D^n + a_{n-1}D^{n-1} + \cdots + a_0$$

En este caso el operador $L(D)$ es un polinomio donde el operador diferencial cumple el rol de variable. Estos operadores nos permiten definir nuevas funciones operando sobre otras dadas.

Además, estos operadores cumplen ciertas propiedades:

1. Linealidad: $L(D)[ax(t) + by(t)] = aL(D)x(t) + bL(D)y(t)$
2. Regla de la suma: $[L_1(D) + L_2(D)]x(t) = L_1(D)x(t) + L_2(D)x(t)$
3. Regla del producto: $[L_1(D)L_2(D)]x(t) = [L_2(D)L_1(D)]x(t) = L_1(D)[L_2(D)x(t)]$

2.2. Regresión

La regresión lineal es una herramienta estadística que estudia la relación que hay entre una variable dependiente y una o más variables independientes, que busca estimar los parámetros que modelan esta relación.

Como nos indican Su, Yan y Tsai en *Linear Regression* [10], dado un conjunto de datos $\mathcal{D} = \{(y_i, \mathbf{x}_i) : i = 1, \ldots, n\}$, donde $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip})^{\mathrm{t}} \in \mathbb{R}^p$ es la i-ésimas muestra para nuestras variables independientes e y_i es el valor de la variable dependiente asociada a esta muestra, consideramos

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \varepsilon_i \quad \text{with } \varepsilon_i \overset{\text{iid}}{\sim} \mathcal{N}\left(0, \sigma^2\right)$$

como el modelo lineal asociado que visto en forma matricial resulta

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}.$$

Una vez construido el modelo que siguen nuestros datos, el paso natural que continua es la búsqueda de métodos para el cálculo los parámetros que modelizan esa relación ($\beta_j : j = 1, \ldots, p$). Una de las técnicas más utilizadas es el método de mínimos cuadrados (SSE por sus siglas en inglés), que busca minimizar la suma de los cuadrados de la diferencia entre los valores predichos con los datos observados:

$$SSE(\beta) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\mathrm{t}} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

Si diferenciamos con respecto de $\boldsymbol{\beta}$ obtenemos

$$-2\left(\mathbf{X}^{\mathrm{t}}\mathbf{y} - \mathbf{X}^{\mathrm{t}}\mathbf{X}\boldsymbol{\beta}\right) = 0.$$

La solución a nuestro modelo inicialmente planteado son los valores de $\boldsymbol{\beta}$ que cumplen $\mathbf{X}^{\mathrm{t}}\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^{\mathrm{t}}\mathbf{y}$. Si suponemos que todas sus columnas son linealmente independientes, la matriz $\mathbf{X^tX}$ será definida positiva y por ello la ecuación anterior tiene como única solución

$$\boldsymbol{\beta} = (\boldsymbol{X}^t\boldsymbol{X})^{-1}\boldsymbol{X}^t\boldsymbol{y}.$$

Después de ver el caso multivariante estándar, nos gustaría aplicar la regresión al contexto funcional. Hay dos formas de extender estos modelos:

- Uno o más de las variables independientes son funcionales.
- La variable dependiente es funcional.

En las siguientes secciones estudiaremos estos modelos y las soluciones a los mismos que nos plantean Ramsay y Silverman [1].

2.2.1. Regresión con Respuesta Escalar

Continuando con el estudio de las diferentes mediciones de las estaciones meteorológicas, es normal preguntarse ¿depende el total de las precipitaciones de una estación de las mediciones de temperatura en ese periodo de tiempo? Como hemos visto previamente, para este ejemplo tenemos nuestra variable dependiente $y_i = \int_0^T Prec_i(t)dt$ que nos indica las precipitaciones de cada estación hasta el instante T. Nuestro problema consiste en encontrar la influencia que tienen las mediciones de

la temperatura $x_i(t)$ a lo largo de ese periodo tiempo. De este modo, definimos el modelo como

$$y_i = \alpha + \int_0^T x_i(t)\beta(t)dt$$

En este modelo, definimos el valor α con el término constante que nos ajusta el valor de la variable dependiente. Nuestro interés reside en estimarlo junto con el término $\beta(t)$. Definiremos nuestra función $\beta(t)$ escogiendo una base de funciones $\phi_k(t)$ de modo que nuestra función resulta

$$\beta(t) = \sum_{k=1}^{K_\beta} b_k \phi_k(t) = \boldsymbol{\phi' b}$$

pasando nuestro problema de regresión a estimar los valores del vector $\boldsymbol{b}$. Escogeremos un valor de K_β de modo que sea lo suficientemente grande como para no perder información, pero lo suficientemente pequeño como para poder entender la información contenida en la función. Si este ejemplo lo hubiéramos estudiado desde un punto de vista multivariante, hubiera sido necesario el cálculo de tantos valores de β como observaciones de temperatura se tuvieran, lo que muestra una ventaja aproximar el problema desde el punto de vista funcional.

Del mismo modo nuestra variable independiente $x_i(t)$ queda definida en una base de funciones, no siendo necesario que sea la misma que la de β.

$$x_i(t) = \sum_{j=1}^{K_x} x_{i,j}\psi_j(t) \qquad x(t) = \boldsymbol{C\psi}$$

donde $\boldsymbol{C}$ es la matriz de coeficientes de dimensiones $N \times K_x$. Ahora nuestra integral puede ser expresada como

$$\int_0^T x(t)\beta(t)dt = \int_0^T \boldsymbol{C\psi(t)\phi'(t)b}\, dt = \boldsymbol{CJ_{\psi\phi}b}$$

donde la matriz $\boldsymbol{J_{\psi\phi}}$ es la matriz de dimensiones $K_x \times K_\beta$ definida como

$$\boldsymbol{J_{\psi\phi}} = \int_0^T \boldsymbol{\psi(t)\phi'(t)}\, dt,$$

es decir, la matriz de los productos internos de las bases que ya vimos previamente.

Buscando simplificar aún más la notación definimos el vector $\boldsymbol{\zeta} = (\alpha, b_1, \ldots, b_{K_\beta})$ con $K_\beta + 1$ componentes y la matriz $\boldsymbol{Z}$ de dimensiones $N \times (K_\beta + 1)$ definida como $\boldsymbol{Z} = [\mathbf{1} \quad \boldsymbol{CJ_{\psi\phi}}]$, simplificando nuestro modelo a

$$\boldsymbol{\hat{y} = Z\zeta}$$

Estimando por el método de mínimos cuadrados que vimos previamente, el vector $\boldsymbol{\zeta}$ es la solución de la ecuación

$$\boldsymbol{Z'Z\zeta = Z'y}$$

que se resuelve igual que el problema de regresión multivariante.

Para evitar los problemas de localidad que podemos llegar a tener debido al muestreo de los datos, podemos definir una penalización de rugosidad

$$PEN(\beta) = \lambda \int_0^T (L\beta(t))^2 dt = \lambda \boldsymbol{b'Rb}.$$

donde L es un operador lineal diferencial como los vistos en secciones anteriores, por ejemplo, el operador de la curvatura $L = D^2$ o el de la aceleración armónica $L = (\frac{2\pi}{T})^2 D + D^3$. De este modo, el método de mínimos cuadrados penalizado (PENSSE por sus siglas en inglés) de nuestro modelo a minimizar es

$$PENSSE_\lambda(\alpha, \beta) = \|\boldsymbol{y} - \alpha - \boldsymbol{CJ_{\psi\phi}b}\|^2 + \lambda \boldsymbol{b'Rb}$$

Podemos definir la matriz $\boldsymbol{R_0}$ como la matriz $\boldsymbol{R}$ añadiendo una primera columna y una primera fila de ceros, resultando nuestro modelo a minimizar

$$PENSSE_\lambda(\boldsymbol{\zeta}) = \|\boldsymbol{y} - \boldsymbol{Z\zeta}\|^2 + \lambda \boldsymbol{\zeta' R_0 \zeta}$$

Así la solución por el método de mínimos cuadrados a nuestro modelo es el vector $\boldsymbol{\zeta}$ que satisface la ecuación

$$\boldsymbol{(Z'Z + \lambda R_0)\zeta = Z'y}.$$

2.2.2. Regresión con Respuesta Funcional

El ejemplo de regresión que hemos visto nos relaciona las mediciones de la temperatura durante un periodo de tiempo con el total de precipitaciones en ese periodo. Pero, ¿y si en vez del total queremos estimar las precipitaciones a lo largo del tiempo?

Podemos plantear un modelo en el que las precipitaciones de un instante de tiempo pueden estar relacionadas con las mediciones de temperatura de todo el periodo

$$y_i(t) = \alpha(t) + \int_0^T x_i(s)\beta(s,t)ds + \epsilon_i(t).$$

De este modo, al ser β bivariante se puede estimar cuanto influye la temperatura a cada tiempo s sobre las precipitaciones a tiempo t. Otra forma de ver el problema es mirar la relación que existe a corto plazo, si la temperatura en instantes previos afecta a las precipitaciones o la previa y la posterior,

$$y_i(t) = \alpha(t) + \int_{t-\delta}^{t} x(s)\beta(s,t)ds + \epsilon_i(t) \quad \text{o} \quad y_i(t) = \alpha(t) + \int_{t-\delta}^{t+\delta} x(s)\beta(s,t)ds + \epsilon_i(t).$$

El problema con esta aproximación del modelo es que se contemplan muestras de tiempo fuera del intervalo donde están definidas. También podemos trabajar en un modelo concurrente, la relación entre temperatura y precipitación es solo dependiente del mismo instante de tiempo

$$y_i(t) = \alpha(t) + x_i(t)\beta(t) + \epsilon_i(t).$$

Como nos indican Ramsay y Silverman [1], todos los modelos lineales funcionales pueden ser reducidos al problema concurrente.

Supongamos que tenemos más de una variable independiente y para cada una de ella definiremos una función β_j distinta. El modelo resultante es

$$y_i(t) = \sum_{j=1}^{q} x_{i,j}(t)\beta_j(t) + \epsilon_i(t) \quad \text{o} \quad \boldsymbol{y}(t) = \boldsymbol{Z}(t)\boldsymbol{\beta}(t) + \boldsymbol{\epsilon}(t)$$

en forma matricial. Definiremos nuestras funciones

$$\beta_j(t) = \sum_{k=1}^{K_j} b_{k,j}\phi_{k,j}(t) = \boldsymbol{\phi}_j'(t)\boldsymbol{b}_j$$

de modo que cada función β_j puede tener una base distinta e incluso un número de funciones distintas que componen la base. Así, el total de coeficientes que tenemos que estimar es $K_\beta = \sum_{j=1}^{q} K_j$ definiendo el vector de coeficientes de las funciones β como

$$\boldsymbol{b} = (\boldsymbol{b_1'}, \boldsymbol{b_2'}, \ldots, \boldsymbol{b_q'})$$

Para simplificar más nuestro modelo, definimos la matriz de las bases $\boldsymbol{\Phi}$ de dimensiones $q \times K_\beta$ del siguiente modo

$$\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\phi_1'} & 0 & \ldots & 0 \\ 0 & \boldsymbol{\phi_2'} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \boldsymbol{\phi_q'} \end{pmatrix}.$$

Así podemos expresar nuestro modelo como

$$\boldsymbol{y}(t) = \boldsymbol{Z}(t)\boldsymbol{\Phi}(t)\boldsymbol{b} + \boldsymbol{\epsilon}(t) = \boldsymbol{Z^*}(t)\boldsymbol{b} + \boldsymbol{\epsilon}(t).$$

Es posible, como en el caso de respuesta escalar, que queramos aplicar cierto suavizado a nuestras funciones aplicando una matriz de penalización de rugosidad. Como cada variable independiente tiene su propia función β_j podemos definir una penalización diferente para cada base

$$PEN_j(\beta_j) = \lambda_j \int_0^T (L_j\beta_j)^2 dt.$$

A la hora de introducir las penalizaciones de rugosidad en nuestro modelo, definiremos la matriz

$$\boldsymbol{R} = \begin{pmatrix} \boldsymbol{R_1'} & 0 & \dots & 0 \\ 0 & \boldsymbol{R_2'} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \boldsymbol{R_q'} \end{pmatrix} \text{ donde } \boldsymbol{R_j} = \lambda_j \int_0^T \boldsymbol{\phi_j}(t)\boldsymbol{\phi_j'}(t)dt.$$

Finalmente, mediante el método de mínimos cuadrados podremos calcular los coeficientes de $\boldsymbol{b}$ que resuelven la ecuación

$$\left(\int_0^T \boldsymbol{\Phi'}(t)\boldsymbol{Z'}(t)\boldsymbol{Z}(t)\boldsymbol{\Phi}(t)dt + \boldsymbol{R}\right)\boldsymbol{b} = \left(\int_0^T \boldsymbol{\Phi'}(t)\boldsymbol{Z'}(t)\boldsymbol{y}(t)dt\right)$$

Una vez presentadas la representación de datos funcionales con bases de funciones, las diferentes operaciones entre muestras y la aplicación del problema de la regresión sobre datos funcionales, en el siguiente capítulo se comentará el proceso para su implementación en el paquete *Python scikit-fda*. Se detallará en qué modulo se ha desarrollado cada una, las diferentes decisiones que se han tomado en cuanto a su diseño y las pruebas que se han realizado para la validación de los resultados.

3

Análisis y desarrollo

Una vez estudiados los diferentes métodos de regresión que se quieren implementar y las herramientas necesarias, en este capítulo se explica el proceso de desarrollo que llevó su implementación. Comenzamos con la definición de todos los requisitos que tiene que cumplir nuestro proyecto, desde ser un proyecto de código abierto o la compatibilidad que debe tener con otros paquetes.

A continuación, se mostrarán los diferentes módulos en los que se ha contribuido y las contribuciones realizadas en cada uno de ellos. Además, se explicarán las decisiones técnicas tomadas en su implementación, la documentación generada y las pruebas realizadas. Por último, se explicarán las diferentes herramientas utilizadas y los procesos de integración continua implementados para asegurar la calidad del código.

3.1. Análisis

Como se comentó en los objetivos del capítulo 1, la finalidad de este trabajo es la continuación del desarrollo del paquete Python iniciado por Miguel Carbajo [9] en 2018, por lo que se ha buscado dar continuidad a los requisitos que ya fueron definidos en su momento:

- El software será desarrollado en Python.
- El código deberá seguir los estándares de desarrollo PEP 8 y PEP 257 definidos para el código y documentación.
- Debe tratarse de un proyecto de código abierto.
- La documentación debe ser completa y detallada para facilitar el uso del software.
- Debe ser escalable para facilitar la posibilidad de contribución al proyecto.
- El uso de algoritmos ya programados en las librerías *numpy* [11] y *scipy* [12], referencia en el ámbito científico dentro de Python.

Además de los requisitos anteriores, según avanzaba el desarrollo del proyecto, se han ido añadiendo otros nuevos:

- Implementar métodos que permitan la distribución del paquete, como su inclusión en repositorios de código como *PyPI*.
- Desarrollar el paquete buscando maximizar la compatibilidad de nuestro código con otras librerías como *scikit-learn* [8].

3.2. Diseño

Durante el desarrollo del proyecto, se produjo una reestructuración del paquete en diferentes módulos agrupados por funcionalidad como vemos en la figura 3.1. En ella, la intensidad del color representa la madurez de cada uno de los módulos en el momento de la presentación del paquete que se realizó en el *III International Workshop on Advances in Functional Data Analysis* [13].

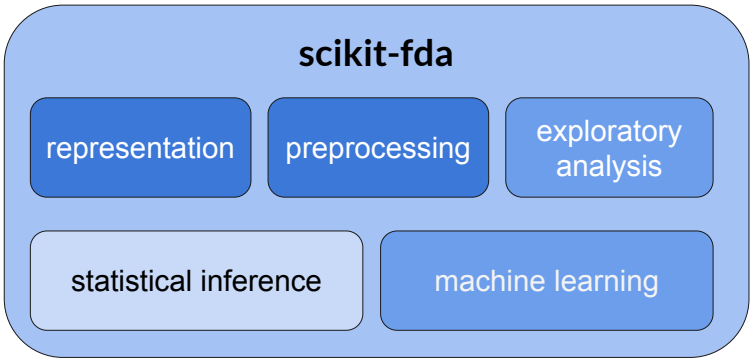


Figura 3.1: Estructura del paquete scikit-fda

Durante la realización de este trabajo se colaboró con los módulos de representación, extendiendo la representación en bases e implementando diferentes operaciones que vimos en la sección 2.1, y en el módulo de machine learning, desarrollando los métodos de regresión funcional que ya presentamos en la sección 2.2. A continuación se detallan las diferentes aportaciones.

3.2.1. Representación

Este módulo engloba las diferentes representaciones que vimos en la sección anterior distribuidas en dos clases *FDataGrid* para la discreta y *FDataBasis* para la representación en bases. Además, contiene diferentes implementaciones de la clase *Basis* para cada una ellas: *Monomial*, *Fourier* y *BSpline*.

A la lista de bases ya implementadas se ha diseñado la clase *Constant*, que representa los datos que no varían con el parámetro. Ésta nos ayudará a representar datos como el intercepto que ya mencionamos en la regresión con respuesta escalar.

Las diferentes operaciones para la representación en bases que se explicaron en la sección 2.1.1 se han desarrollado siguiendo el diseño escogido para estas en la representación discreta. Se han desarrollado los métodos *add* para la suma, *mul* para el producto por escalar, *inner_product* para el producto interno y *derivative* para la derivación.

También se han desarrollado otros métodos que facilitan el trabajo con este tipo de representaciones como *get_item* para seleccionar determinadas muestras dentro de nuestro conjunto o *copy* para hacer clonado de objetos. Para las diferentes bases se facilitan métodos para su conversión a otros tipos, *to_basis* convierte una base en un conjunto de muestras donde cada una es un elemento de la base o *to_scipy* y *from_scipy* que convierten la representación propia de *scikit-fda* de B-spline y monomios a la representación usada por *scipy*.

3.2.2. Operadores Lineales Diferenciales

Los operadores lineales diferenciales se han incluido dentro del paquete miscelánea que engloba diferentes herramientas. Además, se ha diseñado de forma que pueda trabajar indistintamente con ambas representaciones. A la hora de construirlos, se han implementado dos formas, mediante un entero que indica el orden del operador (D^n) o como una lista de coeficientes ($a_0, \ldots, a_n$) representando su posición ella como el orden del operador, resultando la combinación lineal de ellos ($a_0 D^0 + \cdots + a_n D^n$).

La evaluación de los operadores lineales se ha desarrollado de dos formas diferentes, usando la evaluación de las representaciones e indicando el operador a aplicar o con el método *evaluate* de la clase y que evalúa los datos que recibe. Para poder comparar diferentes operadores se ha implementado un método *equal*.

3.2.3. Regresión

Los métodos de regresión han sido implementados dentro del módulo creado tras la reestructuración del paquete *machine learning*. El principal requisito a la hora de desarrollar estos algoritmos fue la compatibilidad con la *API* de *scikit-learn* [14], sobrecargando las diferentes clases que provee a este efecto, *BaseEstimator* y *RegressorMixin*. Para ello se han desarrollado dos clases para cada una de los métodos regresión descritos, *LinearScalarRegression* para la respuesta escalar (sección 2.2.1) y *LinearFunctionalRegression* para la respuesta funcional (sección 2.2.2).

Así los regresores deben de tener implementados los métodos *fit* para el entrenamiento del modelo y *predict* para la evaluación del modelo. En nuestras implementaciones, el método *fit* es el encargado del cálculo de los coeficientes para las funciones β de nuestro modelo dadas las variables independientes y la variable dependiente, y el método *predict* se encargará de utilizar el modelo para obtener los valores de la variable dependiente para cada muestra.

Una de las características más interesantes de la integración con *scikit-learn* es la integración con sus *pipelines*, flujos de trabajo para el procesado de los datos. De este modo se pueden definir diferentes métodos para la preparación de nuestros datos (alineamiento, suavizado, reducción de dimensionalidad, etc.) previa a la aplicación de los métodos de regresión.

A continuación, se explican las diferentes decisiones que se tomaron durante la fase de codificación de las funcionalidades descritas en este capítulo, las metodologías utilizadas para documentar los desarrollos y las diferentes pruebas realizadas para la validación de los métodos.

3.3. Codificación, documentación y pruebas

Uno de los puntos más importantes a la hora de llevar a cabo el desarrollo, dado su enfoque código abierto, ha sido la implementación de los estándares definidos para el lenguaje. Con ello, se buscaba unificar el desarrollo de todo el paquete y tener estilo similar al de los paquetes de referencia para su mejor integración con la comunidad.

Python define sus *PEP*, estándares que se deben seguir a la hora de llevar a cabo los desarrollos en el mismo. Como se indicaba en el análisis, todo el código ha sido implementado en base a las normas indicadas en *PEP 8* referentes a la guía de estilo de código. La tabulación de líneas con espacios, su longitud máxima definida, definida como 79 caracteres o el número de saltos de línea entre funciones o clases vienen definidas por este estándar.

Python permite implementar los diferentes operadores para nuestras clases a través de los métodos conocidos como *Magic Methods*. De esta manera cuando ejecutamos la sentencia $a + b$ el intérprete llamará al método *__add__* de la clase a con b como parámetro. Pero es posible que un array de *numpy* no tenga implementada la suma con nuestras representaciones. En esos casos, éste avisa de que no sabe cómo operar y el propio intérprete llama al método *__radd__* de la clase b con a como parámetro. El uso de un método distinto sirve a dos propósitos: evitar las llamadas circulares y para los casos en los que la suma no sea conmutativa. Existen otros métodos aritméticos como *__sub__* para la resta, *__mul__* para la multiplicación o *__truediv__* para la división. También se han implementado otros como la comparación *__eq__* o la selección de subconjuntos dentro de una secuencia de datos *__getitem__*.

Para las operaciones de derivación y producto interno se ha utilizado el mismo nombre de función y parámetros que ya estaban definidos por la representación discreta, a fin de unificar su uso. Esto facilitó el desarrollo de la clase abstracta *FData* desarrollada por Pablo Marcos [15] y que provee de una *API* común a ambas representaciones, facilitando la abstracción de los métodos a la hora de trabajar con éstas.

Para el caso del producto interno, se optó por el uso de múltiples implementaciones internas como

el cálculo a través de la integración numérica o la matriz de Gram, optando por uno u otro automáticamente al llamarlo, de modo que se minimice el tiempo de computación.

Una vez terminado cada desarrollo, se siguieron los estándares definidos por la *PEP 257* para realizar el proceso de documentación, que describe a través de los *docstring* la forma de llevarlo a cabo. Los *docstring* son cadenas de texto al principio de cada función, clase o módulo que los definen, dan una explicación de su funcionamiento, las propiedades que tiene o parámetros de ejecución y muestran diferentes formas de uso. En la figura 3.2, se muestra el docstring de la función derivada para datos representados en bases, en el que se ve una explicación del método y diferentes casos de uso con el retorno esperado.

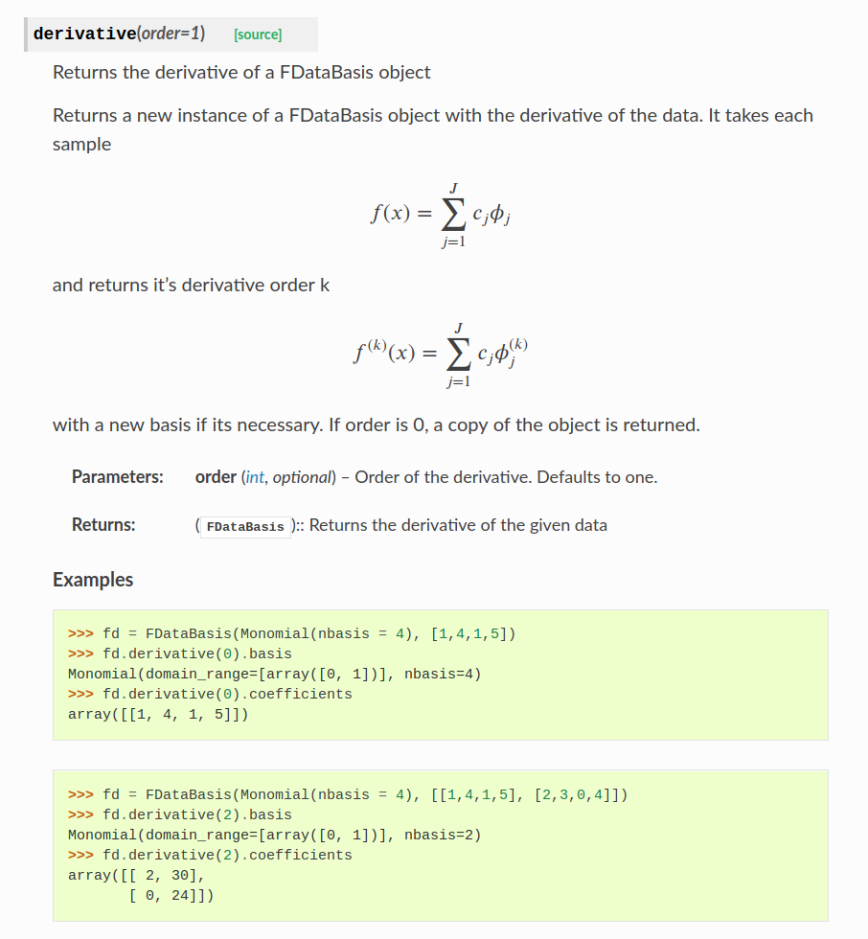
```
class FDataBasis(FData):
    def derivative(self, order=1):
        r"""Returns the derivative of a FDataBasis object

        It takes each sample

        .. math::
            f(x) = \sum_{j=1}{J}c_j\phi_j

        and calculates the derivative order k

        .. math::
            f^{(k)}(x) = \sum_{j=1}{J}c_j\phi_j^{(k)}

        with a new basis if it's necessary. If order is 0, a copy of the object
        is returned.


        Args:
            order (int, optional): Order of the derivative. Defaults to one.

        Returns:
           (:obj:`FDataBasis`):: Returns the derivative of the given data

        Examples:
            >>> fd = FDataBasis(Monomial(nbasis = 4), [1,4,1,5])
            >>> fd.derivative(0).basis
            Monomial(domain_range=[array([0, 1])], nbasis=4)
            >>> fd.derivative(0).coefficients
            array([[1, 4, 1, 5]])

            >>> fd = FDataBasis(Monomial(nbasis = 4), [[1,4,1,5], [2,3,0,4]])
            >>> fd.derivative(2).basis
            Monomial(domain_range=[array([0, 1])], nbasis=2)
            >>> fd.derivative(2).coefficients
            array([[ 2, 30],
                   [ 0, 24]])
        """
```

Figura 3.2: Ejemplo de docstring para el cálculo de derivadas de un FDataBasis

Además, toda esta documentación puede ser procesada por la herramienta *Sphinx* que la convierte a formatos accesibles como *PDF* o *HTML*. Una de las características más importantes de ésta, dado el carácter científico del paquete, es la posibilidad de la inserción de fórmulas LaTeX como la de la figura 3.2. Una vez compiladas, serán mostradas mediante su representación gráfica como podemos ver en la figura 3.3

A la hora de validar la funcionalidad implementada se ha hecho uso de la herramienta *pytest* utilizada para el desarrollo de test unitarios en *Python*. Esta herramienta valida inicialmente todos los

ejemplos que han sido desarrollados en los *docstring* y compara la salida con lo esperado en el ejemplo.



derivative(*order=1*) [source]

Returns the derivative of a FDataBasis object

Returns a new instance of a FDataBasis object with the derivative of the data. It takes each sample

$$f(x) = \sum_{j=1}^{J} c_j \phi_j$$

and returns it's derivative order k

$$f^{(k)}(x) = \sum_{j=1}^{J} c_j \phi_j^{(k)}$$

with a new basis if its necessary. If order is 0, a copy of the object is returned.

Parameters: **order** (*int, optional*) – Order of the derivative. Defaults to one.

Returns: (`FDataBasis`):: Returns the derivative of the given data

Examples

```
>>> fd = FDataBasis(Monomial(nbasis = 4), [1,4,1,5])
>>> fd.derivative(0).basis
Monomial(domain_range=[array([0, 1])], nbasis=4)
>>> fd.derivative(0).coefficients
array([[1, 4, 1, 5]])
```

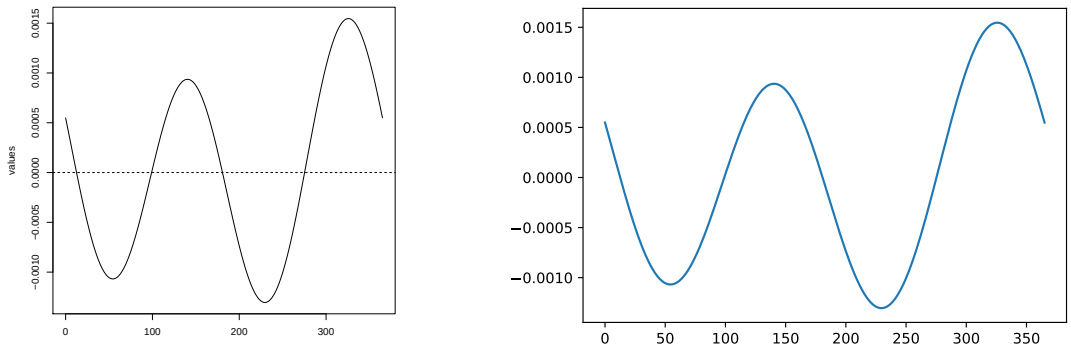
```
>>> fd = FDataBasis(Monomial(nbasis = 4), [[1,4,1,5], [2,3,0,4]])
>>> fd.derivative(2).basis
Monomial(domain_range=[array([0, 1])], nbasis=2)
>>> fd.derivative(2).coefficients
array([[ 2, 30],
       [ 0, 24]])
```

Figura 3.3: Ejemplo de documentación generada por Sphinx

Por otro lado, se creó un módulo propio denominado *tests* y que contiene una clase por cada una dentro del paquete. Así para cada función del módulo, se evaluaban las diferentes casuísticas posibles y se diseñaban diferentes tests encargados de ello. Por ejemplo, el producto interno se puede calcular entre objetos del tipo *Basis* y *FDataBasis* y además éste por las propiedades vistas en la sección 2.1.1 debe ser conmutativo. Por ello se diseñaron test para probar todas esas casuísticas, así como otros que evaluaran los diferentes errores que puede generar la función, como el recibir parámetros con tipos de datos incompatibles.

Para la comprobación de los resultados, se desarrollaron diferentes métodos que pudieran exportar nuestros datos para usarlos con la librería *fda* [5]. De este modo, se podía hacer una validación cruzada de los resultados que obtenían nuestros métodos con los obtenidos por sus homólogos en R. Por ejemplo, en el caso de la regresión funcional con respuesta escalar con el dataset Canadian Weather [5] se buscaba estudiar la relación entre las temperaturas anuales y el total de precipitaciones anuales en cada estación. Aplicando el mismo problema en ambos paquetes se comprobó que los coeficientes

obtenidos para las funciones beta eran iguales, y su representación podemos verla en la figura 3.4.



(a) R: fda

(b) Python: scikit-fda

Figura 3.4: Ejemplos de funciones beta calculadas por los diferentes paquetes para el mismo dataset

Para comprobar que la mayor cantidad de código era cubierta por los test se ha implementó el uso de la herramienta *coverage*, encargada de identificar que partes del código eran evaluadas en los test y cuáles no. De este modo era posible identificar partes de la funcionalidad que no estaban siendo revisadas y diseñar nuevos test para asegurar el correcto funcionamiento de los métodos.

3.4. Control de versiones e integración continua

Los proyectos de código abierto a menudo involucran el trabajo conjunto de muchas personas. Proyectos antes citados como *numpy* o *scipy* son desarrollados y mantenidos por más de medio millar de personas que mejoran y corrigen el software de forma constante, para lo que es vital elegir un control de versiones adecuado. En nuestro caso, se utiliza git como herramienta para el control de versiones dada su sencillez de uso y la integración con *Github* que es el servicio elegido para albergar el código.

Github es la plataforma más utilizada globalmente para almacenar código y un referente para toda la comunidad de software libre. Facilita la administración de estos proyectos con diferentes herramientas para la revisión de código, planificación del trabajo o incluso el registro de incidencias o sugerencias para su solución o implementación por los desarrolladores.

Para la organización del modo de trabajo con el control de versiones se ha optado por usar el método *gitflow*. Cada nueva característica a desarrollar es introducida en una rama *feature* durante su fase de codificación y pruebas, y una vez finalizada estos cambios son enviados a la rama *develop*, que irá guardando las diferentes características que saldrán con la nueva versión. Una vez todos los cambios que saldrán con la nueva versión están en develop, se copian a una rama release en la que

se realizarán todas las pruebas de integración necesarias previas a la versión definitiva. En esta rama, se pueden añadir nuevas características desde la rama *develop* o corregir errores de las diferentes funcionalidades. Terminadas las pruebas, los cambios realizados se copian a la rama *develop* (para almacenar las correcciones) y a la rama *master*, donde se guardan las versiones definitivas del código.

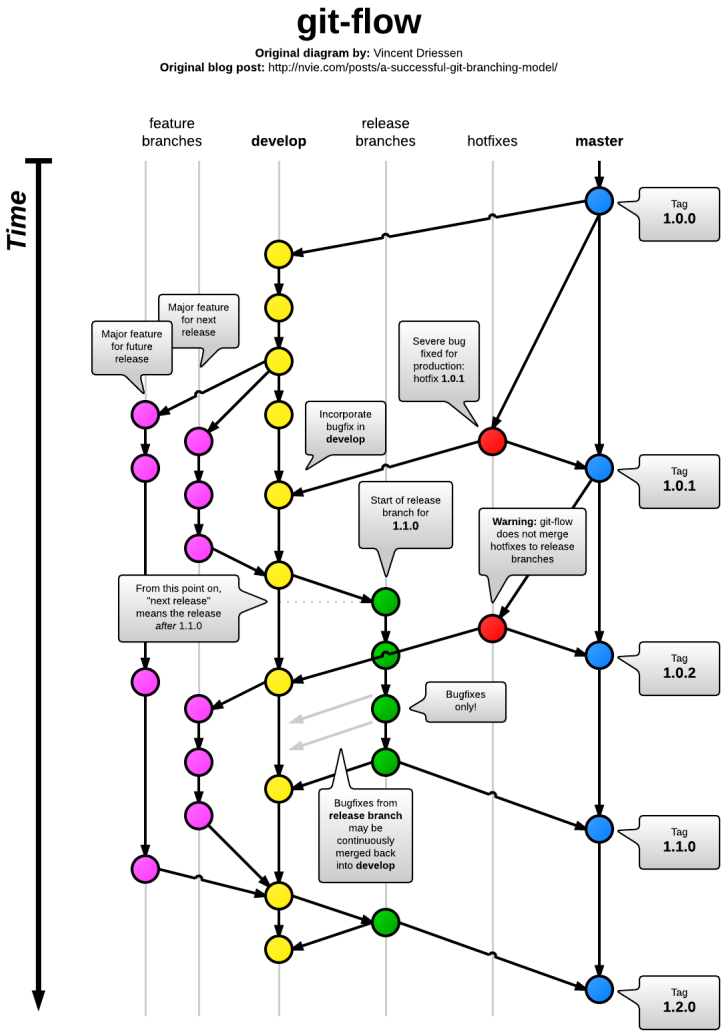


Figura 3.5: Esquema de organización de las ramas de gitflow

Para asegurar la calidad del código y una correcta generación de la documentación se ha implementado un sistema de integración continua, procesos que automatizan todas estas tareas. Para ello se ha utilizado la herramienta *Travis CI*, que permite a través de un sencillo fichero de configuración alojado en el mismo repositorio parametrizar todos estos procesos y ejecutarlos cada vez que se realiza un cambio. Uno de los procesos más importantes es la validación de test, una comprobación de que el código cumple con los estándares de calidad definidos por estos. De esta forma, es necesaria una validación satisfactoria de todos los tests para poder llevar código de una rama a otra.

Uno de los requisitos que definimos al principio de este capítulo fue la implementación de diferentes estándares *PEP* de *Python*. Se ha añadido al proceso de *Travis CI* la validación de las normas definidas en *PEP 8* a través de la herramienta *flake8*, que analiza el código del paquete e indica los errores con respecto a lo definido por la guía de estilo.

También se ha automatizado el proceso que evalúa la cobertura de código realizada por la herramienta *coverage*. Ésta realiza un informe que se envía automáticamente a la plataforma codecov y muestra el porcentaje del código que es cubierto por los tests y señala qué partes del código no son probadas.

Por último, otro de los requerimientos de este desarrollo es la disponibilidad de una documentación accesible y completa. *Readthedocs* es una herramienta que recoge los cambios que se realizan en la documentación del paquete, y aloja en su web la misma tras compilarla con *Sphinx*. Así, toda la documentación referente al paquete y diferentes ejemplos de uso están accesibles en *https://fda.readthedocs.io*.

4

Conclusiones y trabajo futuro

El principal objetivo de este trabajo era la implementación de métodos de regresión funcional con respuesta tanto escalar y funcional en la librería scikit-fda. La integración dentro de su equipo de desarrollo y la utilización de las dinámicas de trabajo de la filosofía de código abierto fueron también objetivos del proyecto.

Con esta finalidad, una de las primeras necesidades fue el aprendizaje de Python, para familiarizarse con su sintaxis y librerías. Además, también ha sido necesario conocer R para poder estudiar las diferentes soluciones al problema de la regresión que ya estaban desarrolladas.

Durante el transcurso del proyecto, se realizaron reuniones regulares del equipo del proyecto, con el fin de explicar y compartir la teoría tras los diferentes desarrollos, por lo que se ha obtenido un conocimiento sobre datos funcionales más allá de la regresión.

Después de finalizado este proyecto, todavía hay más trabajo posible para las áreas que se han cubierto. Para la representación sería importante la implementación de nuevas bases de funciones como la exponencial o bases bivariantes para la representación de superficies, que actualmente solo permite la representación discreta.

Además, en el área de la regresión se debería continuar con el desarrollo de nuevos métodos, incorporando otros como los desarrollados en la librería *refund* [7], basados en el análisis de componentes principales (PCA por sus siglas en inglés) o la regresión de mínimos cuadrados parciales (PLS por sus siglas en inglés).

Por último, sería interesante fomentar la participación de los usuarios finales del paquete en partes del desarrollo, reportando errores en el software o haciendo sugerencias de funcionalidad nueva que pueda orientar la hoja de ruta del equipo de la librería.

Bibliografía

[1] J. O. Ramsay and B. W. Silverman, *Functional data analysis*. Springer-Verlag New York, 2005.

[2] P. Kokoszka and M. Reimherr, *Introduction to functional data analysis*. CRC, Taylor and Francis Group, 2017.

[3] F. Ferraty and P. Vieu, *Nonparametric functional data analysis: theory and practice*. Springer, 2011.

[4] J. O. Ramsay and B. W. Silverman, *Applied functional data analysis: methods and case studies*. Springer, 2002.

[5] J. O. Ramsay, H. Wickham, S. Graves, and G. Hooker, *fda: Functional Data Analysis*, 2018. R package version 2.4.8.

[6] M. Febrero-Bande and M. de la Fuente, "Statistical computing in functional data analysis: The r package fda.usc," *Journal of Statistical Software, Articles*, vol. 51, no. 4, pp. 1–28, 2012.

[7] J. Goldsmith, F. Scheipl, L. Huang, J. Wrobel, J. Gellar, J. Harezlak, M. W. McLean, B. Swihart, L. Xiao, C. Crainiceanu, P. T. Reiss, Y. Chen, S. Greven, L. Huo, M. G. Kundu, S. Y. Park, D. L. Miller, and A. M. Staicu, *refund: Regression with Functional Data*, 2018. R package version 0.1-17.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[9] M. Carbajo Berrocal, *FDA-PY: desarrollo de un paquete Python para el análisis de datos funcionales*. Universidad Autónoma de Madrid, 2018.

[10] X. Su, X. Yan, and C.-L. Tsai, "Linear regression," *WIREs Comput Stat*, vol. 4, pp. 275–294, 2012.

[11] O. Travis E, "NumPy: A guide to NumPy." USA: Trelgol Publishing, 2006. `http://www.numpy.org/`, [Online; accessed 2019-06-20].

[12] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python," 2001–. `http://www.scipy.org/`, [Online; accessed 2019-06-20].

[13] C. Ramos-Carreño, "Scikit-fda: A Python package for Functional Data Analysis." III International Workshop on Advances in Functional Data Analysis, 05 2019.

[14] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[15] P. Marcos, "Functional data analysis: interpolation, registration, and nearest neighbors in scikit-fda," 06 2019.