Maestría en Ciencia de Datos

Departamento de Matemática y Ciencias

Density Functions Time Series Analysis

With Application to the Aerospace Sector

Nicolás Lupi **2023**

Directora: Marcela Svarc

Universidad de
SanAndrés

Abstract

The objective of this work is to compare different methods for forecasting the evolution of a probability density function over time. Specifically, given a set of observations generated by a random process over time, our interest lies in characterizing its temporal evolution to make future predictions. The motivation behind this analysis is the ability to generate samples from a new distribution conditioned on the passage of time, although it could be conditioned on another variable depending on the problem. The trigger for this analysis is a study that aims to investigate the changes over the years in the characteristics of satellite launches into orbit, particularly concerning their mass. From the data, for each year, we can obtain an estimator of the mass density of satellites. Based on these estimates, we seek to forecast the densities for the upcoming years.

Three alternative methods are proposed. The first approach is parametric and assumes that the sequence of distributions comes from a parametric family, and that future observations will also belong to the same family. Consequently, the problem reduces to modeling the parameters of these distributions through time series. The second approach is non-parametric and is based on Functional Principal Component Analysis (FPCA): considering densities as a sequence of functional data, they will be represented in a convenient basis of functions weighted by scalars that vary over time. With these scalars, a time series will be constructed for forecasting. Finally, the method of Conditional Generative Adversarial Networks (cGAN) is an innovative approach that aims to train a model to generate new samples similar to real ones, taking into account the time period for which they will be generated. Like the second method, it makes no assumptions about the distributions, but unlike the previous one, it does not require constructing or forecasting any time series.

This work follows the following structure: Chapter 1 introduces the problem's motivation, and Chapter 2 describes the three methodological proposals. Chapter 3 is devoted to analyzing their performance on simulated datasets. In the fourth chapter, the data related to the weights of the satellites are analyzed. Finally, the conclusions of the work are presented.

Acknowledgments

I would like to express my gratitude to Marcela Svarc for her patience, suggestions, and assistance.

Dedicated to Flor, Fue en Defensa Propia, and friends.

"Si esto va de a tres, a cada paso haremos cien . . . "

Contents

CHAPTER 1

Introduction

Often, we encounter data for which we are interested in predicting future behavior. A common instance of this problem involves having samples from a stochastic process recorded at different time points, which we group into windows, summarizing the information by obtaining typical statistics such as the mean, median, standard deviation, percentiles, maximum, minimum, among others.

For example, we can have meteorological data such as the temperature of a city, data that we might have on an hourly basis. It might be of interest to predict the evolution of this variable over time, for which one option would be to summarize the observations on a daily basis, taking, for instance, the daily average, and constructing a time series with one data point per day for subsequent forecasts.

Another example of a different nature could be data concerning the orbits to which satellites have been launched over time. For these satellites, we might be interested in their masses, to determine whether they have become lighter or heavier over the years, or in the parameters that define their orbits, such as altitude, inclination, among others, to identify any temporal patterns. Once again, one approach could be to summarize all this data into the average launch for each year and then construct a time series based on this average.

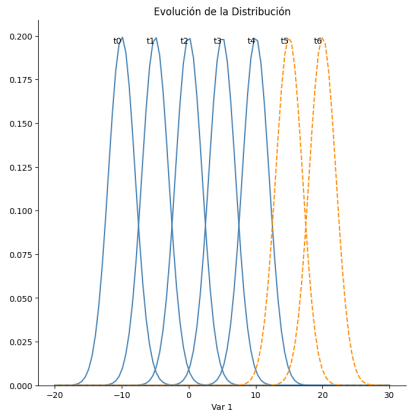
Often, for the analysis of this type of data, it suffices to construct a time series of the statistics of interest and model the series in a conventional manner. However, this approach can be insufficient, for example, if our data exhibit more than one mode with distinct behaviors that are hidden when using classical statistics. In the case of satellites, we could summarize the information by the average mass and observe that it did not vary from one year to another, when in reality what was happening was that we had two modes drifting apart, which could be recording some temporal pattern that we would be missing. This concept can be better understood by observing Figure 1.1. Hence, the need arises to adopt another approach that allows us to understand the distribution law of our data and how it varies over time.

This work involves applying methods to the temporal analysis of data that enable the extraction of meaningful insights from observations as a whole. This way, we can not only predict how temperature or the average launch will behave but also draw conclusions about the distribution that generated this data and how it changed over time. Special emphasis will be placed on methods that allow the generation of new observations for future periods.

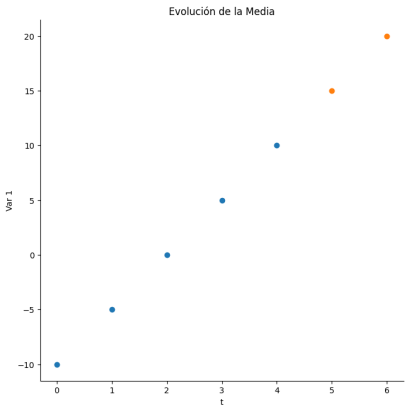
Our problem might involve conducting simulations for a complex system

that depends on weather conditions, such as the case of an entity interested in determining whether a city's drainage network could handle the rainfall in the coming year. It would be of little use to know that the average precipitation level for the next period will be so many millimeters, even if our hypothetical network could handle it if we overlook the fact that intolerable levels are highly likely based on historical rainfall data.

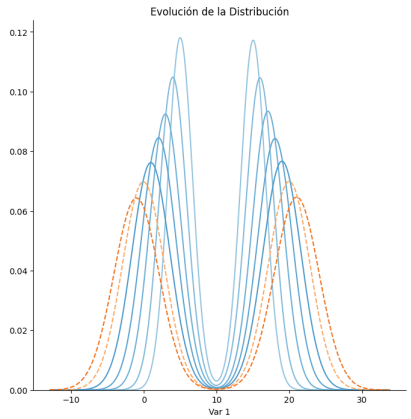
We could also be an entity responsible for operating or launching satellites, and we need to have an idea about what characteristics these satellites will have in the future. Once again, if we work solely with summary statistics, we would be missing valuable information, which could lead us to make misguided decisions.



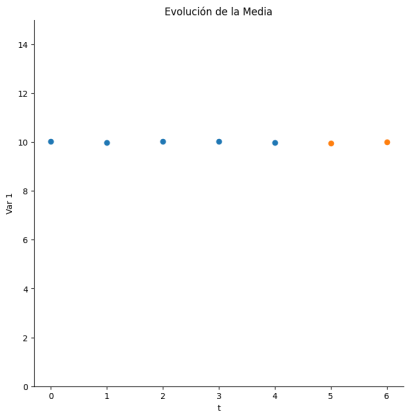
(a) Shift in the distribution



(b) Increase in the mean



(c) Separation of modes



(d) Constant mean

Figure 1.1: In the first row, distributions shift over time resulting in an increase in the mean; in this case, we would not lose information by predicting only the mean. In the second row, we have bimodal distributions with modes drifting apart over time; in this case, the mean is not a good reflection of the behavior of the distributions.

CHAPTER 2

Theoretical Framework

2.1 Problem Approach

We begin with a set of observations, where $X_{i_t,t}$ represents observation i at time period t. In our case, the variable t represents the time period in which the observation originated and varies from period 1 to T. However, this can be adapted to any problem where we want to estimate a density function conditional on some variable. Also, observe the subscript $i_t = 1, \ldots, n_t$, to emphasize that the samples at different time periods are unrelated (they are not panel data), and even the sample sizes are not necessarily the same at different points in time.

We assume that these observations were generated by some distribution with a density function f_t that varies over time (or according to what t represents). This function is unknown to us, so we estimate it for each t based on our observations, obtaining $\hat{f}_t$. The problem then revolves around, based on $\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_T$, predicting $\hat{f}_{T+h}$ for the desired value of h.

To achieve this, we will introduce three alternative methods and test them on simulated data as well as real data from satellite launches. First, we will begin with the scenario in which we assume that the f_t belong to a known parametric distribution family, either due to some domain-specific knowledge or because the data suggests it. This way, we can estimate the distribution parameters for each time period t and predict them using standard time series techniques.

The second alternative involves utilizing elements from the field of Functional Data Analysis (FDA). Without the need to assume that the functions come from any particular family, we will estimate the $\hat{f}_t$ for each t, whether in a parametric or non-parametric manner. Once these functions are estimated, we will apply Functional Principal Components, breaking them down into a series of time-invariant underlying functions. These functions, weighted with different weights, will give rise to the original functions. It's these weights that vary over time, and with them, we will construct the time series to forecast the functions.

Lastly, we will consider an alternative using neural networks, training a conditional GAN model. One model will attempt to generate new observations that closely resemble real data for a given time point, while the other will try to distinguish this observation from the original samples, taking into account the time period it is conditioned on. Unlike the previous methods, this approach will not allow us to explicitly obtain the density function for either periods $1, \ldots, T$ or beyond. The output will be a generative model capable of generating

samples for future periods, upon which we will adjust the densities $\hat{f}_{T+h}$ as the final step.

2.2 Methodology

Parametric Method

A straightforward approach is to assume that the observations come from a parametric distribution $f(\cdot, \theta_t), \theta_t \in \Theta$. Furthermore, in each period, the distribution belongs to the same family, and what varies over time are the values of the parameters that characterize it.

Since, in practice, the parameters θ_t are not observed, the first step involves fitting the distributions for each period based on the available data, estimating the parameters using a method such as maximum likelihood, resulting in $f(\cdot, \hat{\theta}_t)$.

Using the estimates, we construct the series $\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_T$ from which we will obtain $\hat{\theta}_{T+h}$ for the desired h. By forecasting the parameters and maintaining the assumption that the functions will continue to belong to the same family, we construct the function $f(\cdot, \hat{\theta}_{T+h})$.

While the method is straightforward, some of its disadvantages are already evident. For instance, it requires having some knowledge or making assumptions about the family from which the distributions originate. Additionally, certain parameters require careful handling when forecasting them: if we forecast a standard deviation, we must impose the constraint that it is always positive, or if we deal with a covariance matrix, it must always be positive semi-definite. The method would be suitable only if the data fits a distribution family correctly. If an appropriate parametric fit is not found, or if doing so requires estimating an increasing number of parameters, this approach becomes less practical, and we would then consider a non-parametric alternative.

Non-Parametric Method - Functional PCA

Secondly, we consider the alternative proposed by Sen and Ma [11], where instead of forecasting the estimated parameters of a function, they estimate and forecast the function as a whole, drawing on the field of Functional Data Analysis (FDA). The idea, much like before, starts with the series of density functions for each period $f_1, f_2, \ldots, f_T$ to forecast f_{T+h}. The difference now is that this method begins by estimating the functions f_t without the need to assume the family from which they originate. Following Horta and Ziegelmann [5], we assume that the series $f_1, f_2, \ldots$ is stationary and treat the f_t as elements of the Hilbert space $L^2(I)$ defined on a compact interval I and equipped with the inner product $\langle f, g \rangle := \int_I f(x)g(x)dx$.

Since f_t is not directly observable, in practice, we must first estimate $\hat{f}_t$ based on observations $x_{1,t}, \ldots, x_{n_t,t}$. In the cited application cases, the authors propose estimating densities using kernels, but in principle, any method could be used (provided it maintains a certain level of quality). In our case, we follow this approach for both simulations and the application case, estimating densities using a Gaussian kernel:

$$\hat{f}_t(s) = \frac{1}{n_t h_t} \sum_{i=1}^{n_t} K\left(\frac{s - x_{i,t}}{h_t}\right),$$

where $K(s) = (\sqrt{2\pi})^{-1} \exp(-s^2/2)$ and h_t is the bandwidth, for which we use the Silverman rule [12] $h_t = 1.06\,\hat{\sigma}_t\, n_t^{-1/5}$ with $\hat{\sigma}_t$ representing the standard deviation calculated for the observations in period t.

The next step involves applying FPCA to the T density functions fitted in the previous step. To better explain this point, let's first draw an analogy with traditional Principal Component Analysis (PCA), following Ramsay and Silverman [10] and Nicol [9]. Given the random vector $X = (X_1, \ldots, X_p)$[1], PCA aims to find a lower-dimensional linear subspace such that the variance of the data projected onto this subspace is maximized. First, we seek a unit vector ξ_1 such that the projection of X onto this vector has the highest possible variance:

$$\max_{\xi_1} \operatorname{Var}\left(\xi_1^{'} X\right) = \xi_1^{'} S \xi_1 \quad \text{subject to} \quad \xi_1^{'} \xi_1 = 1,$$

where S is the covariance matrix of X. The solution to the above problem is the eigenvector of S associated with the largest eigenvalue. The next step is to find the unit vector ξ_2 that maximizes the variance of the data projected onto it, while also ensuring that ξ_2 is orthogonal to ξ_1. This process can be repeated, selecting the eigenvectors of S associated with the largest eigenvalues at each step, with the added variation diminishing in each subsequent step.

From a sample perspective, starting with a random sample x_i with $i = 1, \ldots, n$, the first step of PCA involves finding the vector ξ_1 such that the expression $y_{i,1} = \xi_1^{'} x_i$ - the projection onto the direction of the first principal component - has the highest possible variance $N^{-1} \sum_i y_{i,1}^2$, subject to the condition that the vector ξ_1 has unit norm. This step can be repeated to find new ξ_j, ensuring in each step that ξ_j is orthogonal to ξ_k for $k = 1, \ldots, j-1$. The matrix S is replaced by its sample counterpart, Σ_n of dimension p x p, where the element $a_{i,j} = \frac{1}{n} \sum_k^n x_k^i x_k^j$, and the solution consists of retaining the eigenvectors of this latter matrix, ordered by their eigenvalues.

In the case of FPCA, instead of starting with a finite-dimensional random vector, we begin with a functional variable $f(x)$; instead of matrices, we deal with linear operators; instead of summations, we work with integrals; and rather than the traditional inner product, we employ the inner product in the L^2 space. In this context, we use the following definitions:

$$\mu(x) = \mathbb{E}\left[f(x)\right],$$

$$\operatorname{Cov}_f(x, y) = \sigma(x, y) = \mathbb{E}\left[f(x) f(y)\right] - \mathbb{E}\left[f(x)\right] \mathbb{E}\left[f(y)\right],$$

$$\operatorname{Var}_f(x) = \sigma^2(x) = \mathbb{E}\left[f(x)^2\right] - \mathbb{E}\left[f(x)\right]^2$$

We also define the following covariance operator:

$$\Gamma : L^2 \longrightarrow L^2, \quad \Gamma(f) = \int_I \sigma(x, y) f(x) dx$$

[1] For simplicity, let's assume $\mathbb{E}[X] = 0$ for the explanation, but it could be otherwise

Given our random element $f(x)$, we can formulate the PCA problem in functional terms, where we aim to project this random function onto a set of orthogonal functions ξ_i in such a way that the variance of $\langle \xi_i, f \rangle$ is maximized:

$$\max_{\xi_i \in L^2} \text{Var}\left(\langle \xi_i, f \rangle\right) \quad \text{s.a.} \quad \langle \xi_i, \xi_k \rangle = \delta_{ik},\ k \leq i,\ i = 1, 2, \ldots$$

Continuing the analogy with traditional PCA, the solution to the above problem arises when $\xi_i(x)$ are the eigenfunctions (instead of eigenvectors) of the covariance operator $\Gamma(f)$. These ξ_i are the functional principal components, and the new variables $y_i = \langle f - \mu, \xi_i \rangle$ are the projections of f in the direction of ξ_i, or the scores of the principal components. Subsequently, we can express $f(x)$ using the Karhunen-Loève expansion as follows:

$$f(x) = \mu(x) + \sum_{k=1}^{\infty} y_k \xi_k(x)$$

In practice, instead of the vectors x_i, our sample consists of a series of n functions f_i originating from the same random process. In sample terms, we now aim to maximize $\frac{1}{n}\sum_{j=1}^{n}\langle f_j, \xi_k \rangle^2$ while imposing orthogonality among the ξ_k. The solution will be found for the eigenfunctions of the sample version of the covariance operator $\hat{\Gamma}_n(f) = \frac{1}{n}\sum_{j=1}^{n}\langle f_j, f \rangle f_j$.

Given our series of functions $f_1, f_2, \ldots, f_T$, we define the *score* $y_{t,k} = \langle f_t - \hat{\mu}, \xi_k \rangle$. Finally, our empirical version of the Karhunen-Loève expansion is expressed as follows:

$$f_t(x) = \hat{\mu}(x) + \sum_{k=1}^{T} y_{t,k} \hat{\xi}_k(x)$$

Just like in traditional PCA, the explained variance decreases with each added ξ_k, so it is expected that a small number $K < T$ of components should suffice to represent the functions adequately. In a way, what we are doing is projecting the data into a lower dimension. Now we can express our density functions as a combination of a series of K functions (the elements of our new basis), each weighted by its corresponding weights, the $y_{t,k}$. What's interesting about this is that once we have found the functions $\hat{\xi}_k$, we can analyze our densities in the space of scalars $y_{t,k}$:

$$\hat{f}_t(x) \approx \hat{\mu}(x) + \sum_{k=1}^{K} y_{t,k} \hat{\xi}_k(x)$$

Returning to the context of a time series of T densities, what we do is apply FPCA to represent them as a combination of K functions, which must be weighted by a weight vector y_t in each period to reconstruct the original function. The crucial point is that, given this analysis, the only thing that varies over time is the vector y_t. Therefore, we manage to transform the problem from a time series of functions to one of a time series of scalars that can be addressed with traditional tools. This way, we can forecast the vector y_{T+h} for future periods, which will be used to weight the K functions in our basis, resulting in the estimated density $\hat{f}_{T+h}$.

In other words, and following the analogy with traditional PCA, this method is analogous to having data in $\mathbb{R}^n$ generated at different time points. The method

involves reducing the dimension to $\mathbb{R}^K$ and then performing time series analysis with these projections.

The steps described above would be sufficient for a time series of functions. However, in our case, we deal with probability density functions, which come with constraints of non-negativity and unit integral. To enforce the non-negativity constraint, instead of applying FPCA directly to the densities, we perform it on the logarithm of the densities. Then, when we want to reconstruct f_t, we take the exponential of the expression we have [11]. To enforce the unit integral constraint, we divide the resulting function by its integral.

Conditional GANs

Finally, as an alternative, we will consider using conditional GANs. A GAN is an unsupervised learning procedure where the goal is to determine patterns within a dataset and then generate new samples that could have been part of the original set of observations. The idea behind this method is to create a generative model capable of generating new observations of future distributions based on observations from previous periods.

To train a GAN, we start with a dataset and two models: a generator model (G) and a discriminator model (D). The generator model aims to generate observations that capture the main characteristics of the original data, starting from a noise vector. The discriminator model's role is to distinguish between real and generated data. In this way, GANs typically tackle the problem introduced by Goodfellow et al. [3]:

$$\min_G \max_D \mathbb{E}_{x\sim p_X(x)} \left[log\, D(x)\right] + \mathbb{E}_{z\sim p_z(z)} \left[log\left(1 - D(G(z))\right)\right], \qquad (2.1)$$

where $D(\cdot)$ represents the discriminator model's output (indicating the probability of an observation being real), $G(\cdot)$ represents the generator model's output (a vector in the sample observation space), and p_X and p_z are the population distributions of real data and noise, respectively. In other words, we have a minimax problem between two players: one, the discriminator, aims to maximize the function (2.1), essentially detecting real data from "fake" data. The other, the generator, seeks the opposite by minimizing the function (2.1), making the discriminator classify the generated data as real.

As the discriminator becomes better at distinguishing between real and fake data, the value of the objective function approaches zero. Conversely, if the generator manages to train itself to deceive the discriminator, leaving it with no better option than to "guess" whether a sample is real or fake, the objective function would reach a value of $-log(4)$. This can be understood by first considering that, with a fixed generator that produces the distribution over the data domain $p_g(x)$, the minimum value the objective function could reach is when the discriminator produces $D_G(x) = \frac{p_X(x)}{p_X(x)+p_g(x)}$ (see [3]). If the generator can train itself to achieve a perfect mapping between Z and X, then we have $p_g = p_X$, so $D_G = 0.5$. In other words, if the generator trains perfectly, when an optimal discriminator encounters a new observation, it will have no way of knowing whether it is fake or real, and it would have to choose randomly, with a 50% chance of being correct.

Training a GAN largely involves maintaining synchronization between the learning progress of the discriminator and the generator. It's crucial to balance

the rate at which both models are trained to avoid undesirable equilibriums, such as the "Helvetica Scenario" [3], where a generator that is trained far beyond its rival discriminator ends up learning only to collapse the Z space to a single point in the X domain where the discriminator cannot determine if the observation is real or fake. In our use case of GANs, which we will detail later, this problem would be equivalent to the generator model learning to generate only the mode of our distribution. These anomalies are known to occur with these models, and it's important to look out for them.

In general, adversarial networks are often trained with the aim of having a generative model for observations that reside in highly complex spaces, as seen in image generation, for instance. Building upon this, we have conditional GANs, developed by Mirza and Osindero [8]. As the name suggests, cGANs tackle the problem of generating samples conditional on one or more variables, denoted as y. The discriminator must take into account the condition of variable y to discern whether a sample is real or fake given the context, and the generator must consider it to produce samples that are as realistic as possible given that same context. Consequently, the objective function becomes:

$$\min_G \max_D \mathbb{E}_{x \sim p_X(x)} \left[log\, D(x|y) \right] + \mathbb{E}_{z \sim p_z(z)} \left[log\, (1 - D(G(z|y))) \right]$$

For instance, suppose we want to train a GAN to generate images of animals. A standard GAN would be trained on images of various types of animals, and when generating new observations, we would have no control over the resulting images. In contrast, with a cGAN, we could instruct the generator to create either a dog or a cat image, and we would guide the discriminator to judge whether the image is real or fake, considering that it should be either a dog or a cat. In other words, we would be labeling the dataset so that the generator learns how to adjust the mapping from Z to X based on the values of these labels. Additionally, the discriminator would somehow learn to determine how these labels should appear to detect generator-made observations correctly.

Our proposal is to use cGANs not for image generation as commonly done but for generating observations conditioned on the selected time step. Once the training is complete, we will have a generative model to which we will provide the variable y representing the chosen time step. In return, it will produce random samples that, according to the model, could have occurred at that time. Then, assuming that the generator has learned how to adjust the mapping from Z to X based on changes in t, we can request it to generate samples for $T + h$, thereby obtaining a model capable of producing future observations based on what it learned from past observations.

One difference from the first two approaches is that, in this case, we will not predict the density. Instead, we will generate new samples $x_{i,t}$ from which we can subsequently fit a distribution using traditional methods.

Another consideration to keep in mind is that training a GAN with the traditional objective function (2.1) often comes with some issues, such as the mode collapse mentioned earlier or model instability in the face of subtle parameter changes. This is because (2.1) is typically not continuous in the generator's weights. To address this problem, Arjovsky et al. propose using the Wasserstein distance for probability distributions $W(q, p)$, leading to the objective function:

$$\min_G \max_{D \in \mathbb{D}} \mathbb{E}_{x \sim p_X(x)} [D(x)] - \mathbb{E}_{z \sim p_z(z)} [D(G(z))] ,$$

where $\mathbb{D}$ is the set of Lipschitz-1 functions. To enforce the constraint that the Discriminator belongs to $\mathbb{D}$, they suggest clipping the weights of D to the range $[-c, c]$. This alternative is known as Wasserstein GAN (WGAN) [1].

Gulrajani et al. build upon this previous contribution and add that the way of imposing the Lipschitz-1 constraint can lead to some complications, such as driving the gradient to extreme or null values and underutilizing the GAN's capacity by limiting the discriminator to very simple functions. They suggest, instead, to impose the constraint in a "soft" manner by incorporating it directly into the objective function, which would now be:

$$\mathbb{E}_{x \sim p_X(x)} [D(x|y)] - \mathbb{E}_{z \sim p_z(z)} [D(G(z|y))] - \lambda \mathbb{E}_{\hat{x}} \left[(||\nabla_{\hat{x}} D(\hat{x}|y)||_2 - 1)^2 \right] \quad (2.2)$$

The term on the right penalizes the discriminator when its gradient with respect to the inputs (in our case, the observations) deviates from 1. $\hat{x}$ represents convex combinations of observations from $G(Z)$ and X, and λ is a parameter that determines how this term will impact the function. This alternative is known as Wasserstein GAN with Gradient Penalty (WGAN-GP) [4]. It's important to note that Equation 2.2 is the loss function proposed by these authors, adapted to our problem, conditioning both the generator and the discriminator on the labels y, which in our case represent time. Fabbri [2] explored the application of WGAN-GP in a conditional setting, experimenting with both categorical and numerical labels, albeit for image generation.

One thing to consider is that since both the Generator and the Discriminator will take an observation or noise and the time moment as inputs, we normalize the data so that the time variable's values fall within the range of the other inputs. For instance, our generator could be taking as input both a noise vector with a mean of zero and a year, which could be, for example, 1995.

To conclude, an interesting point to consider is the one discussed by Zaheer et al. [13], who point out the difficulty that GANs face when trying to learn simple one-dimensional distributions, even when using the Wasserstein distance. In their paper, the authors conclude that these models tend to collapse onto the mode in such cases and fail to generate samples faithful to the entire distribution. We will pay attention to this when evaluating the model on our data.

CHAPTER 3

Simulations and Implementation

We implemented the three alternatives and tested them on model datasets, knowing in advance what to expect, in order to evaluate their performance. The entire implementation was carried out in Python, using open-source packages[2]. For each case, we attempted to predict the evolution of normal distributions or mixtures of normals, from which we took 1000 samples for each time period. We used 20 periods as the training set and aimed to predict the next 5. For each method, we visualized its predictions by comparing them with the distributions of the training period. We estimated these distributions using Gaussian kernels based on the samples we took from the original distributions (the same samples used for fitting). Finally, we evaluated the prediction quality by fitting each alternative 100 times and averaging the Kullback-Leibler divergence [7] from the true densities. We tried to use the same configurations and hyperparameters for each case. In the following sections, we will describe the general implementation of the three alternatives for all simulations and the case of application.

Parametric

For each observed period in the sample, we fit the corresponding distributions to the generated observations. In cases with only one mode, we fit one distribution per period, while in cases with more than one mode, we fit mixtures of distributions. After making these estimates, we forecast the parameter series using different models depending on the case. With these forecasts, we then reconstruct the distributions at $T + h$ and evaluate them on the grid.

FPCA

As a first step, we fit a distribution for each period on the generated samples using kernels, with the Silverman's rule for bandwidth selection. Next, we take the log-density and evaluate it on a grid. On this discretization, we re-express the log-densities in a B-spline basis. The number of B-splines was chosen as the minimum number such that the functions still visually resemble the originals. For all cases, we chose a number of 30 B-splines. Then, we apply FPCA, projecting the functions onto a basis of K orthogonal functions. In general, for K=2, we can explain more than 99% of the variability in the functions,

[2]Code available at https://github.com/nicolaslupi/densities-time-series

although in some cases, we decided to take higher values of K to improve the quality of the results.

Once the eigenfunctions are fitted, we obtain their scores series and forecast them with linear, polynomial, or exponential trends, or an ARIMA model depending on the case, to reconstruct the log-densities at $T + h$. The final step is to exponentiate the functions and normalize their integrals to 1.

GAN

For the GAN, we aimed for a parsimonious architecture compared to the usual architectures available[3]. We understand that these architectures are designed to train on much more complex domains than what we will be dealing with. We ended up using a Generator with four fully connected hidden layers, each with 16 nodes and Leaky ReLU activation, except for the last one (Figure 3.1). The noise vector is one-dimensional (a scalar), sampled from a normal distribution $\mathcal{N}(0, 1)$.

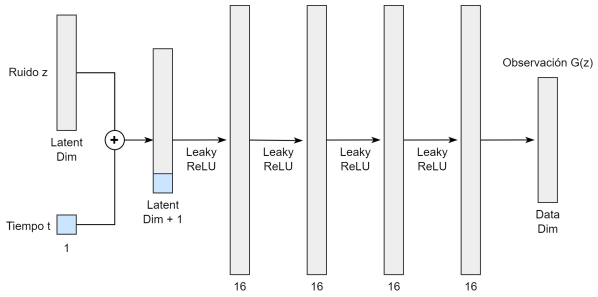


Figure 3.1: Generator

Regarding the Discriminator (Figure 3.2), it consists of two fully connected hidden layers, each with 64 nodes, including dropout and Leaky ReLU activation, except for the last layer, which outputs a scalar with a sigmoid activation function, representing the probability that the input observation is real.

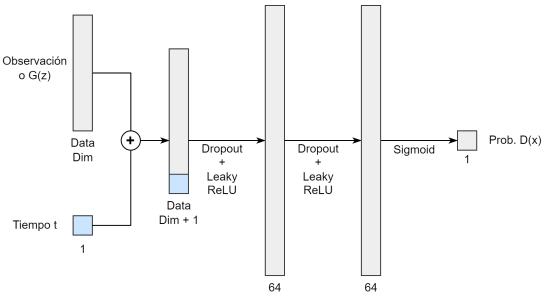


Figure 3.2: Discriminator

Regarding the hyperparameters, we kept the standard values used in previous implementations. In particular, we used an Adam optimizer with $\beta_1 = 0.5$,

[3]We based our implementation on the code available at https://github.com/eriklindernoren/PyTorch-GAN, making the necessary changes to adapt it to our problem.

$\beta_2 = 0.999$, and a learning rate of $2e-4$ for both networks. Concerning training, we set $ncritic = 5$ as the number of steps between each update of the generator. The only hyperparameter for which we deviated from typical values is λ in the loss function 2.2, which is usually set to 10 in previous implementations. In our case, we obtained better results with lower values, as low as 0.5.

When training the GANs, the criterion used to determine when to stop training was how visually similar the distributions generated by the generator for different time points were to the real distributions. In general, we let the model train for up to 2000 epochs. While in some cases training for fewer epochs might not significantly affect performance, we preferred to keep the training duration the same for all cases.

Once the model is trained, the final step is to reconstruct the functions for $T + h$. To do this, we sample a noise vector that the generator takes as input along with the corresponding labels for the periods h we want to generate. The generator returns the samples conditioned on each moment, and we fit the distributions $\hat{f}_{T+h}$ using kernels on these samples.

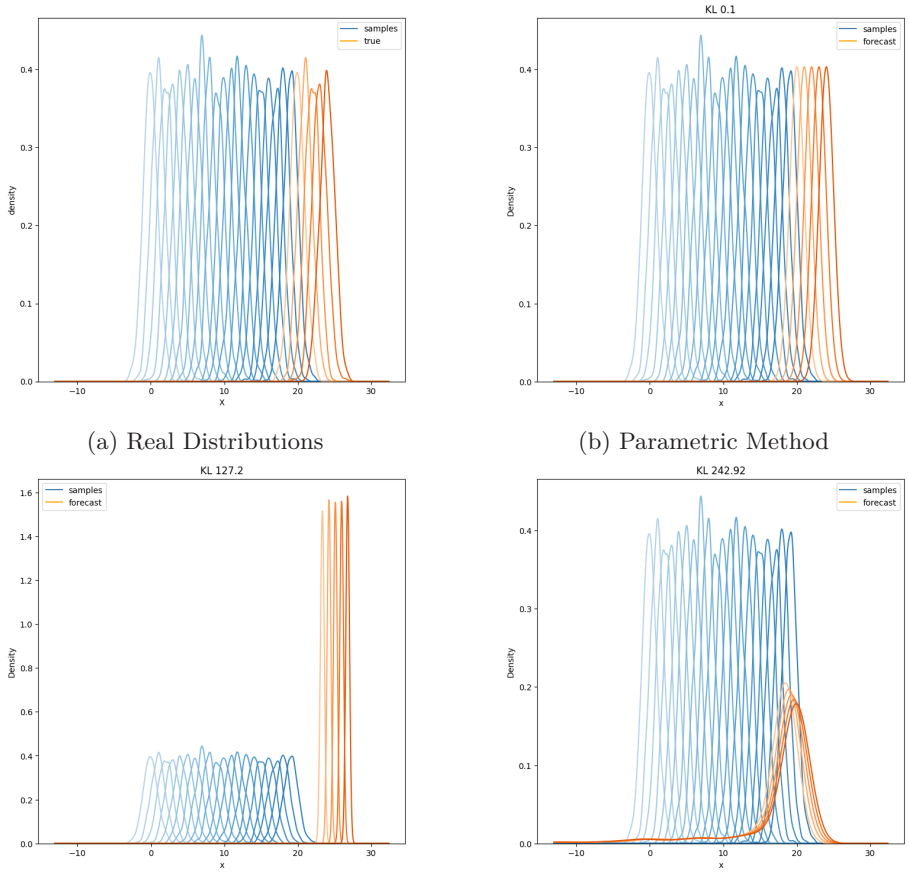
3.1 Uniform Shift

The first case we simulated is that of a one-dimensional normal distribution with a standard deviation of 1, whose mean increases uniformly by one unit over time, as shown in Figure 3.3a. The alternatives should capture this shift and continue it over time.

For the parametric case, we fit a normal distribution for each period using the Scipy implementation and then forecast the means with a linear trend while keeping the standard deviations constant at their sample period mean. After a brief fit, the method achieves the expected results, as seen in Figure 3.3b, with an average Kullback-Leibler (KL) divergence of 0.1 from the true densities. We emphasize that this good result is due to our prior knowledge of the data-generating distribution, which is why we sought to fit normal distributions.

For the FPCA method, we used the Scikit-FDA library. We fit two eigenfunctions, forecasting their scores with linear and quadratic trends, respectively. After a brief fit, the method manages to detect the displacement of the distributions, although it exaggerates the jump between the sample period and the prediction period and underestimates the data's dispersion (Figure 3.3c). The fit is worse than the parametric method, both visually and quantitatively, with an average KL divergence of 127.2.

Regarding the GAN, we trained the model for 2000 epochs with the parameters described above using the PyTorch library. This was the most time-consuming alternative to train, although we could have stopped training earlier. If the FPCA method exaggerated the displacement, with the GAN, we observed an underestimation instead (Figure 3.3d), with the predictions for $T + h$ overlapping the densities of the sample period. Furthermore, the densities have a heavy left tail that is not present in the sample distributions. Of the three alternatives, it produces the worst fit with an average KL divergence of 242.92.



(a) Real Distributions

(b) Parametric Method

(c) FPCA Method

(d) GAN Method

Figure 3.3: Results - Linear Trend. In blue, the sample densities, and in orange, the test predictions. Average KL divergence on test data is displayed on each figure. The best fit is achieved by the parametric method, largely because we fit normals, knowing beforehand that this is the true distribution of the data.

3.2 Uniform Shift with Two Modes

Similar to the previous example, we now aim to see if the methods can capture the shift in the distribution when it consists of two normals separated from each other. Again, we have 20 sample periods and 5 to predict; out of the 1000 samples in each period, 500 will come from a normal with a standard deviation of 1, and 500 from an identical distribution but shifted by 10 units, as shown in Figure 3.4a.

For the parametric case, now that we have two modes, we fit a mixture of two normals per period by estimating the parameters (mean and standard deviation of both) and the weights of each one, using the implementation from the *pomegranate* library. Once the parameters are estimated, we only forecast the means using an ARIMA model for each mean. We could have used a linear trend, but we preferred to use ARIMA since the results do not differ, and we will also use this model later for other series that are not as straightforward.

The standard deviations and weights of the normals in the mixtures are kept constant as their mean in the sample period. Like in the case of a single mode, the predictions are correct, capturing to a great extent the shift of both modes (Figure 3.4b). The only noticeable issue is that the dispersion of the distributions is lower than in the sample period. Also, just like with one mode, the parametric method achieves the best fit with an average KL divergence of 3.69.

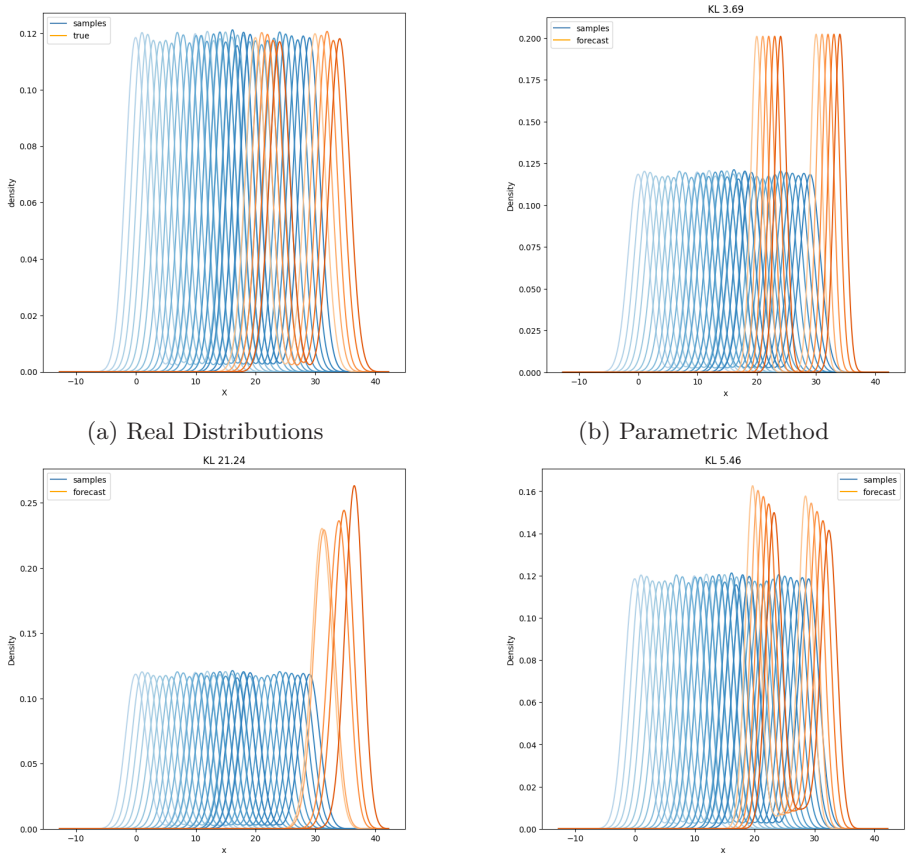
For the FPCA method, we repeat the same steps as in the single-mode case, adjusting two eigenfunctions, but now we forecast the scores using an ARIMA model. When predicting for T+h, we notice significantly worse results than with the parametric method, with an average KL divergence of 21.24 (Figure 3.4c). Of the two modes, the fit only manages to capture the shift of the larger one, while the smaller mode is completely lost in the prediction. We conducted tests with different ARIMA specifications and fitting different numbers of eigenfunctions, but in no case did we manage to achieve a shift for both modes. We suspect that this is partly due to the fact that, like in the case of a single mode shift, we are expecting the method to generate distributions that are increasingly distant from the domain it was trained on. It should be noted that this method is designed for stationary series of functions that are defined on a compact interval, as detailed in Section 2.2; given the nature of the simulation we constructed, the densities would continue to shift indefinitely, eventually falling outside the interval where the f_t are defined, breaking the stationarity assumption. Therefore, this method would not be suitable if the data is non-stationary, potentially leading to mismatches as observed.

Finally, the adversarial network model is trained without changes compared to the single mode case. After training for 2000 epochs, the model shows encouraging results by identifying the shift for both modes and even closely matching the spread of the distributions (Figure 3.4d). This time, the neural network model outperforms the FPCA method with an average KL divergence of 5.46.

3.3 Crossing of Two Modes

The next test we will perform is similar to the previous one, with the difference that now the modes will start separated, come closer with time, until they cross at a certain point and then move away from each other. The methods should capture this and continue to separate them. As in the previous case, both normals have a standard deviation of 1, and we sample 500 observations from each of them per period (having a total of 1000 per period). The difference is that the smaller mode will shift to the right and vice versa, ending up in opposite positions at T (Figure 3.8a).

For the parametric case, we once again fit a mixture of two normals for each period and forecast their parameters. Something to keep in mind, and this only happens with this method, is that we will have to make an assumption about the identification of each component of the mixture at each moment. In other words, for each period, we fit $g_{1,t}(x)$ and $g_{2,t}(x)$, which, when combined with the weights $w_{1,t}$ and $w_{2,t}$, give rise to our estimate of $f_t(x)$. However, the problem is that g_1 and g_2 are not ordered, and when we estimate them, we must somehow order them, and then we can construct the time series of the



(a) Real Distributions

(b) Parametric Method

(c) FPCA Method

(d) GAN Method

Figure 3.4: Results - Linear Trend with Two Modes. In blue are the sample densities, and in orange are the test predictions. Average KL divergence on each figure. Once again, the parametric method achieves the best fit. This time, the neural network model performs better than the FPCA fit, which only detects one mode.

parameters. In the case of Section 3.2, this didn't seem like such a big issue - we had two modes equally separated at each moment, moving together in parallel[4]. However, now it is necessary to make some assumption since, at one point, the modes cross. The most reasonable assumption would be either to think that one mode is decreasing while the other is increasing (which is what is actually happening), or that the modes start getting closer and then separate again. In terms of the time series of the means, this would imply one of the cases indicated in Figure 3.5:

In this scenario, this doesn't seem to be a problem either since, in any case, for $T+1$ we will be stating that one mode increased and the other one decreased. However, for other types of functions where we need to estimate mixtures of more components or where they don't have such a stable behavior over time,

[4]This may not be the case. How do we know if there are two populations alternating with each other underneath?

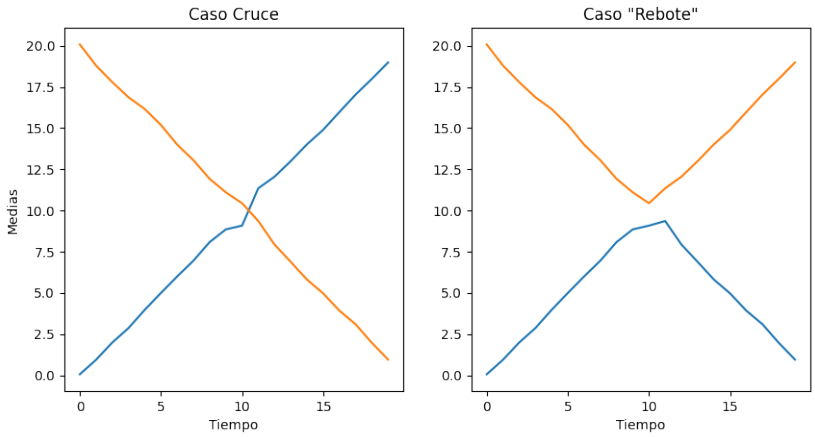


Figure 3.5: Different assignments lead to different time series.

the identification issue could become relevant. This time we will assume the first case, that is, the modes cross each other.

Resuming, after making this assumption, we forecast the means of both normals with ARIMA. We once again keep the standard deviations and weights constant as the mean in the sample period. When evaluating the estimated functions for $T + h$, the fit correctly identifies the separation of both normals, although it again underestimates their standard deviation (Figure 3.8b), resulting in an average KL of 16.87.

With the functional method, in this particular case, we were unable to reconstruct the functions for $T + h$ using only 2 components. The time series of the scores become more complicated when adding less significant components. We had to experiment with various ARIMA specifications and even leave some series as constants to obtain reasonable results. This required visually analyzing the eigenfunctions and interpreting what each one was capturing.

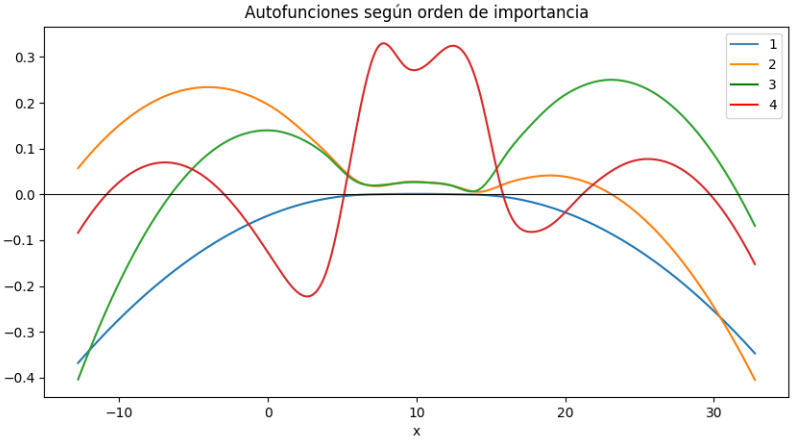


Figure 3.6: Eigenfunctions for the mode crossing in order of importance. Logarithmic scale.

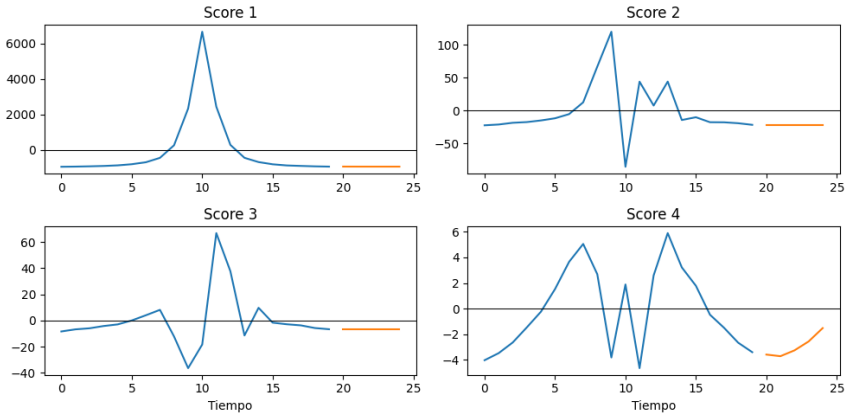


Figure 3.7: Score series for the mode crossing. In blue, the training period; in orange, forecasts for $T + h$.

The eigenfunctions we see in Figure 3.6 are based on the log-densities, so negative values on this scale would translate to values between 0 and 1 on the original scale of the functions. The first eigenfunction, which explains the most variability, determines how centered within the domain of the samples the functions will be. Analyzing its score series in Figure 3.7, we see that at the beginning and end of the period, it is weighted by negative values, giving more importance to values toward the domain's extremes. It becomes positive around the half of the period when the two modes cross in the center. The continuation of this series seems correct with negative values and a slight tendency toward smaller values, indicating that the series will move further away from the center over time. To capture this, we fit an exponential trend only to the descending part of the series.

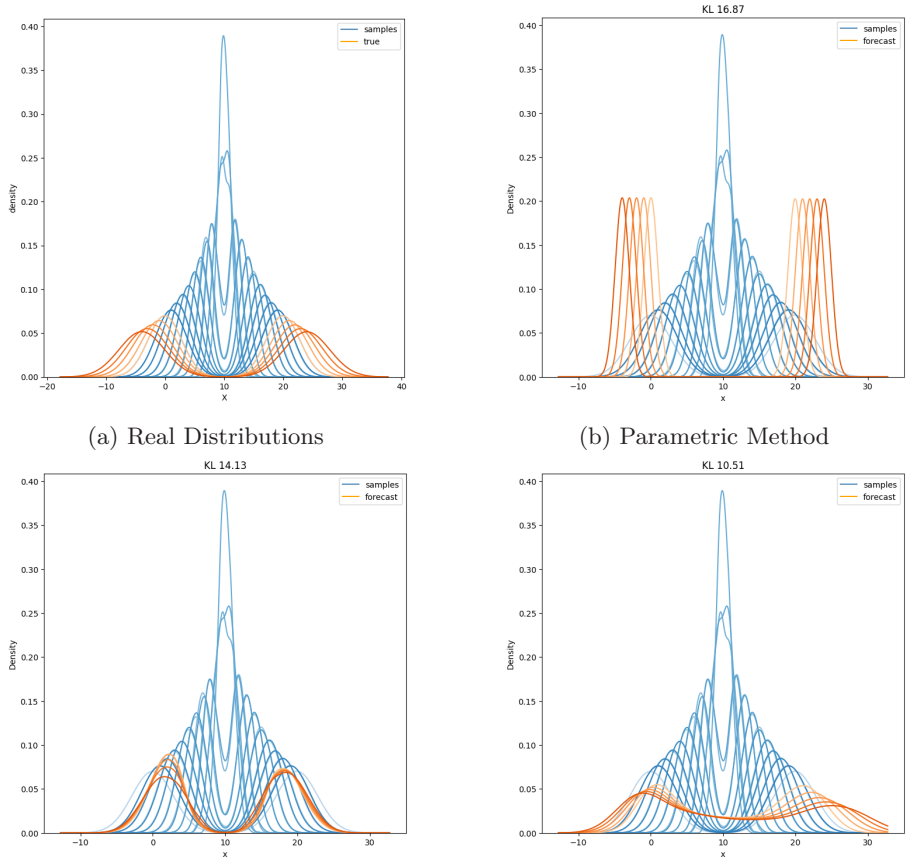
The second and third eigenfunctions govern the extremes of the domain, with each favoring one end while penalizing the other. The ARIMA forecasts leave both series relatively constant; nevertheless, we had to force the second one to remain constant at the last recorded value because ARIMA was overestimating it, causing the predictions to be weighted more toward the left.

The fourth eigenfunction once again emphasizes the center of the domain and partially penalizes the extremes. However, this component contributes very little to the reconstruction of the functions (notice the magnitude of the scores compared to those of the first eigenfunction). We chose to include it as it slightly improved the results, but interpreting the function and its score series becomes challenging.

With these considerations, we managed to make the densities for $T + h$ from the FPCA method look reasonable. However, it still doesn't capture the displacement completely, as all the predictions end up around the same location (Figure 3.8c). For this specific simulation, with a KL of 14.13, FPCA outperforms the parametric method, although it's not capturing the displacement as effectively.

Lastly, for the GAN method, we trained the model exactly the same as in the previous cases, and judging by the results in the case of the crossing of two modes, it is the alternative that best captured the change in distributions over

time and continued it into the future, with a KL of 10.51 (Figure 3.8d). Despite this, it doesn't completely separate the modes well, assigning high density to the values between them (when that region should have a density close to zero). Additionally, it produces an asymmetry that is not present in the samples, especially for the larger mode.



(a) Real Distributions

(b) Parametric Method

(c) FPCA Method

(d) GAN Method

Figure 3.8: Results - Crossing of Modes. In blue are the sample densities, and in orange are the test predictions. Average KL in the test on each figure. This time, the neural network model proves to be the best at capturing the movement of both modes, although it produces an asymmetry that was not present in the original densities.

3.4 Change of Mode

For the last test we conducted, instead of having modes that shift, they will always be in the same position and their proportion will change over time. What we do is generate 1000 samples, taking a fraction with a smaller mode and the other fraction with a larger one. For each period, the fraction of one subpopulation increases at the expense of the other (Figure 3.9a).

For the parametric method, we again fit a mixture of two normals for each period. The main difference from the previous examples is that now we will forecast the time series of the weights w_0 and w_1 in addition to the means. Since these weights range between 0 and 1, we decided to model this series as a linear trend of the *logit* of w_0 over time, and then we take w_1 as the complement of w_0. As for the means, we again use ARIMA.

When predicting the functions for $T + h$ using the parametric method, as seen in Figure 3.9b, we notice that the change from the smaller mode to the larger one is correctly captured. However, the densities are out of scale as their variances are again underestimated, resulting in a KL divergence of 5.0.

Regarding the non-parametric FPCA method, we decided to fit three eigenfunctions. While the first eigenfunction explains the weighting of the modes to a large extent, we incorporated the other two to ensure that the densities for $T+h$ not only reflect the shift from one mode to the other but also accommodate themselves concerning the sample distributions. In general, as observed in Figure 3.9c, we note that the forecasted densities resemble the true ones, despite the smaller mode being penalized a bit more than necessary. Of the three alternatives, this procedure achieves the best fit with a KL divergence of 0.33 for this simulation.

Finally, the conditional GANs method broadly captures the change in the functions without the need to modify hyperparameters and by training for the same number of epochs as in the previous cases. However, we noticed some shortcomings in the fit, such as a slight shift in the modes compared to the original ones (Figure 3.9d), placing this method in second place with a KL divergence of 3.68.

3.5 Results

Next, we present the results after 100 fits of each alternative on each dataset. We conducted only 100 replications due to the computational cost of the procedure, especially the training of the GAN.

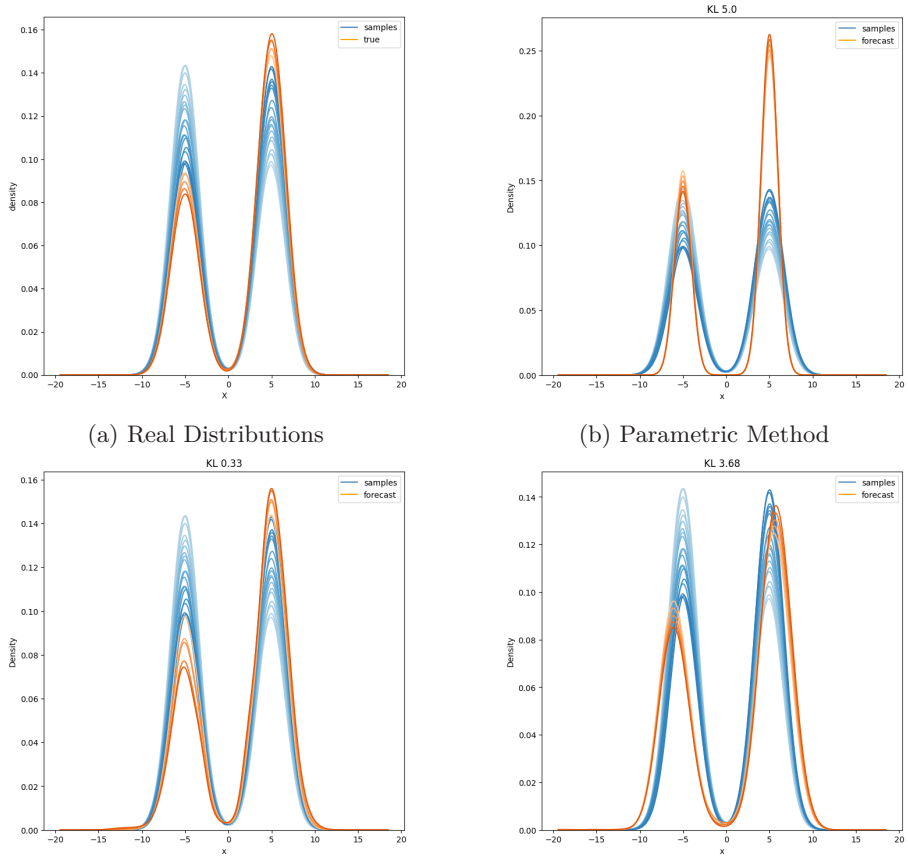
To display the results, in each iteration, we generated new samples for the four scenarios described previously and fitted the three procedures in each case using exactly the same parameters and configurations. For example, for FPCA, we fitted the same number of eigenfunctions and used the same models for the time series. Likewise, for the adversarial networks, we employed the same parameters and training epochs.

Next, we present the average Kullback-Leibler divergences between the predicted densities and the actual ones, calculated over the 100 iterations.

Method	**1 Mode**	**2 Modes**	**Mode Crossing**	**Mode Switch**
Parametric	**0,11**	**3,62**	16,55	4,91
FPCA	92,84	22,95	14,03	**0,20**
WGAN	78,93	9,57	**9,42**	3,31

Table 3.1: Average Kullback-Lebiler divergence.

Table 3.1 shows that for both the single and double mode shift scenarios, the parametric method performed the best, which aligns with our visual assessments.



(a) Real Distributions

(b) Parametric Method

(c) FPCA Method

(d) GAN Method

Figure 3.9: Results - Mode Change. In blue, the sample densities; in orange, the test predictions. Average KL divergence on test data for each figure. The FPCA method produces the best fit, with forecasted densities very similar to the real ones. It is closely followed by the neural network model.

In both cases, apart from the parametric method, the GAN model exhibited better results than the FPCA procedure. We believe this is due to the non-stationary nature of the density series in both simulations, where predictions needed to be made further away from the domain in which the samples were located. This situation is not ideal for the FPCA method, as discussed in Section 3.2.

For the mode crossing and mode switch scenarios, the alternatives that best captured these behaviors were the GAN model and the FPCA, respectively. It's worth noting that, although it takes longer to train, the GAN model was exactly the same for all cases, while the parametric and FPCA methods always required careful adjustments when fitting densities and forecasting time series. Additionally, the GAN model never turned out to be the worst of the three in any of the four cases we examined. The dataset where it faced the greatest challenges was the single-mode shift, which is somewhat surprising as it would seem a simpler case to predict a priori compared to the others. Nonetheless, in

that scenario, it performed better than the FPCA alternative.

CHAPTER 4

Application Case

In this section, we apply the three methods to the dataset that triggered this work. The goal was to analyze the evolution of the characteristics of satellites launched in recent years for a satellite service provider. Some examples of these services include providing internet coverage to satellites, debris removal, in-orbit refueling, power supply, and towing, among others. Specifically, a debris removal provider is responsible for deorbiting obsolete or damaged satellites to free up space and prevent collisions with other satellites. This can be achieved with vehicles equipped with their own propulsion system that would dock with the satellite to be removed and propel it out of orbit. One question that arises when developing this vehicle is related to the mass of the satellites it will have to remove. This is essential to evaluate, for example, what propulsion system to develop or acquire; the vehicle will have different characteristics if it plans to provide services to satellites weighing in the tens of kilograms or to satellites in the tonnage range.

A first approach to answer this question would be to study the evolution of some measure of central tendency for the mass of satellites launched year by year. This way, one could see, for example, if the average satellite is becoming heavier or lighter over time and forecast this measure for the years in which the service is planned. However, this measure of central tendency could hide useful information, such as a subgroup of the market moving against the average, which could be a niche that the provider might miss out on. To conduct a more comprehensive analysis, it would be advisable to study the entire distribution of satellite masses launched and forecast its evolution. With these forecasted distributions, not only could any measure of central tendency be recovered to conduct an analysis like the example above, but the entire distribution would be available, allowing for a deeper analysis. For example, these distributions could be used to sample and conduct simulations, something that was not possible before.

The data we used was provided by Teri Grimwood of the Union of Concerned Scientists (UCS). Quoting the UCS website[5], it provides a dataset with "details on satellites currently orbiting the Earth, including their country of origin, purpose, and other operational details." These operational details include satellite orbit characteristics (parameters like apogee, perigee, or inclination), and the mass of the satellites, a variable of interest in this case.

[5] https://www.ucsusa.org/resources/satellite-database

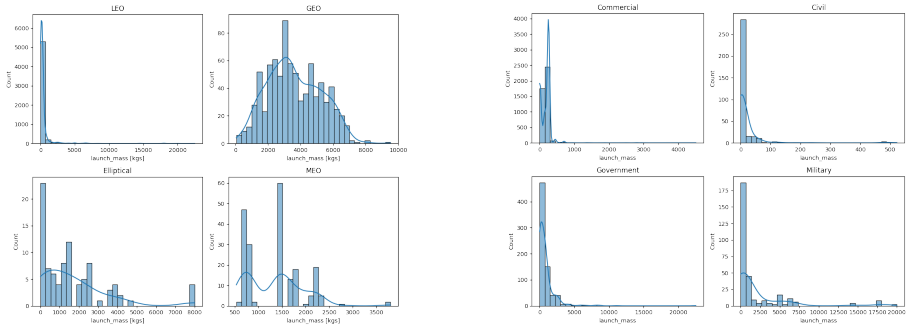
The dataset they provide is public, with the caveat that it is limited to currently orbiting satellites. In other words, a satellite launched in the past that is no longer in orbit will not be available in the sample. This is why we asked UCS to provide access to datasets published in the past, which they made available to us. With these datasets, we were able to obtain a more representative sample of satellites launched in recent years.

The dataset we have consists of 7,137 observations, which means 7,137 satellites launched into orbit from 1967 to 2022. From this original dataset, we first removed observations for which we did not have the variable *launch_mass* (mass at the time of launch in kilograms), leaving us with 6,827 observations.

Something to keep in mind at this point is that satellites and their orbits can vary widely. In particular, a significant distinction can be made between those in Low Earth Orbits (LEO) and those in Geostationary Orbits (GEO). LEO comprises orbits up to one thousand kilometers in altitude and it is the most popular destination for commercial satellites with purposes such as Earth observation or telecommunications. These satellites are usually small, with the majority weighing less than 100 kilograms (Figure 4.1a), and they orbit the Earth approximately every 90 minutes.

On the other hand, GEO orbits are located at 36,000 kilometers in altitude. Satellites launched into GEO tend to be more complex, expensive, and heavier (up to 10,000 kilograms), making these orbits less popular for the private sector. Satellites in GEO orbit the Earth every 24 hours, so from the perspective of an Earth observer, a satellite in this orbit would always appear to be in the same place (hence the name geostationary), making them ideal for meteorology, navigation (GPS), and communications services.

With such a marked separation between these groups of observations, we decided to narrow our study to one of these families, focusing on satellites launched into LEO due to their commonality (they represent 83% of our data).



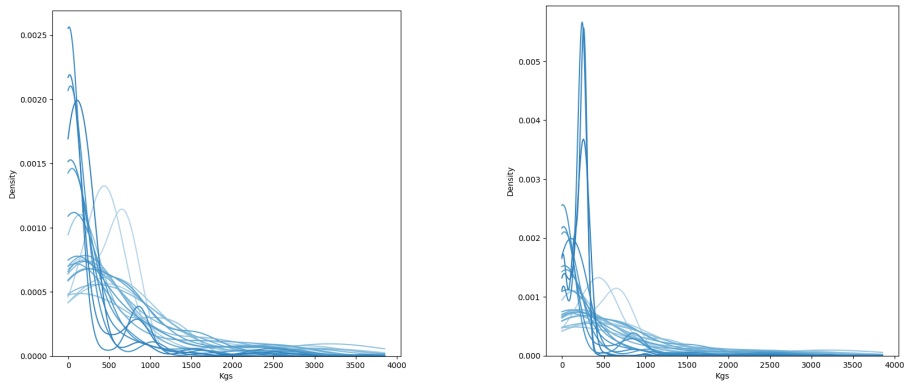
(a) According to orbit class (b) According to user (LEO)

Figure 4.1: Mass distributions. Left panel: the most significant separation is between LEO with satellites mainly under 1000 kilograms, and GEO with high density even up to 8000 kilograms. Right panel: within LEO, the main operators are of commercial and civil type, where the concentration towards lightweight satellites is even more pronounced.

Within the LEO group, we can observe that satellites exhibit different mass distributions based on user type. In Figure 4.1b, commercial and civilian satellites rarely exceed 500 kilograms. On the other hand, there are several

satellites launched by government users that reach up to 5,000 kilograms, and many for military purposes that weigh up to 20,000 kilograms. We chose to discard observations above 4,000 kilograms, as they constitute a very small fraction of the sample (around 1%) and, apart from being in LEO, they do not align with the characteristics of the other satellites. The group of observations we discarded included space station modules like the Chinese space station[6] and other military satellites that exhibited different behavior compared to the rest of the observations.

Returning to our analysis, the next step is to group the observations into time windows. We chose to group the observations by the year of launch and discarded years for which we had few data points (arbitrarily setting a threshold of 10 observations), keeping in mind that the next step is to fit a density function for each time window. Thus, summarizing, we focus on satellites launched into LEO from 1998 onwards, weighing below 4,000 kilograms, and for which we have their mass variable.



(a) 1998-2019 (b) 1998-2022

Figure 4.2: Annual mass distributions. In recent years, there has been a strong concentration around 200 kilograms, driven by the launch of constellations like Starlink.

Through a visual inspection of the data, in Figure 4.2, we can see that LEO satellites have generally remained below 1000 kilograms. What stands out the most is the pronounced peak centered between 0 and 500 kilograms, which has become even more pronounced in the last 3 years. This is because, out of the remaining 5538 data points, 2391 belong to SpaceX and were launched between 2019 and 2022, with the particularity that they are all nearly identical observations (between 230 and 260 kilograms[7]), part of the Starlink constellation[8].

Once the sample set was narrowed down and we had an understanding of how our data looked, we proceeded to apply the three methods described above. We used the distributions from 1998 to 2021 as the training set and reserved 2022 for testing.

[6] https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=2021-035A
[7] https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=2019-074D
[8] https://www.starlink.com/technology

The first step is to generate our functional observations by estimating a density function for each year based on the available data. For this, we employ Gaussian kernels using the Silverman rule [12] for bandwidth selection. It should be noted that these functions will serve as input for the FPCA procedure; both the parametric method and the neural network method will take the raw samples as input. After fitting each method, we will compare the predicted density function for 2022 with the one fitted based on the actual observations.

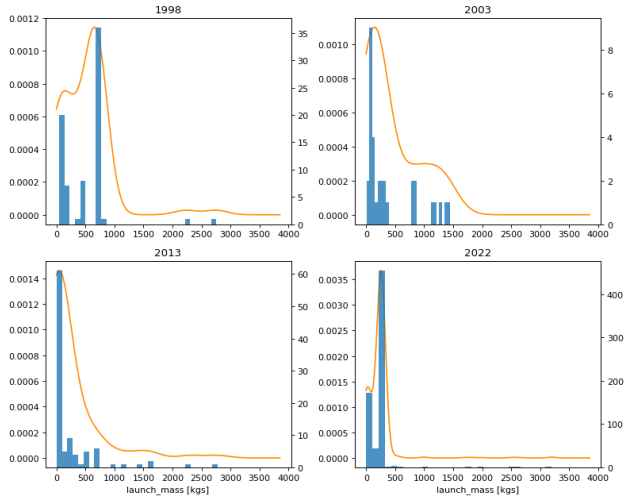


Figure 4.3: Estimated distributions for some years in the sample set.

4.1 Parametric Method

For the parametric method, we chose to fit a mixture of two lognormals for each period. The choice of the lognormal is partly because our densities have support only on positive real numbers and partly because they visually resemble this distribution (they exhibit a peak for values close to zero and a right-skewed tail). The decision to use a mixture of two lognormals instead of a single distribution was made to provide greater flexibility in the fitting process. Additionally, after visual inspection, the distributions often show a primary group of satellites with low mass and a smaller group with fewer observations of heavier satellites.

After fitting the mixtures for each year, we were left with a time series of 5 parameters (two means, two standard deviations, and the weight of one of the mixtures). We assumed that the standard deviations would remain constant, and for $T + h$, we forecasted their mean from the training set. We used ARIMA for the meeans time series, and for the mixture weight, we fitted a generalized linear model with a logit link function with time as the independent variable.

The Kullback-Leibler divergence between the predicted function and the one fitted based on the data is 0.27. In Figure 4.7a, we can see that the estimated density correctly concentrates around values close to zero, although it doesn't completely capture the peak around 200 kilograms, where the Starlink constellation is located.

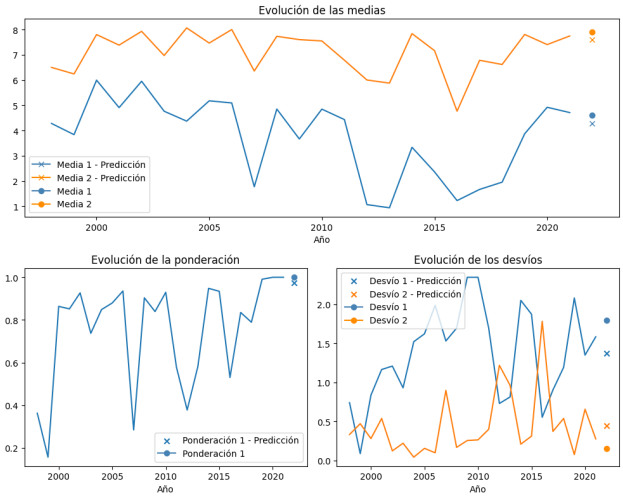


Figure 4.4: Evolution of the parameters. The means are modeled using ARIMA; the weight is forecasted using a generalized linear model with a logit link function, with time as the independent variable; the standard deviations are taken as their mean across the training periods.

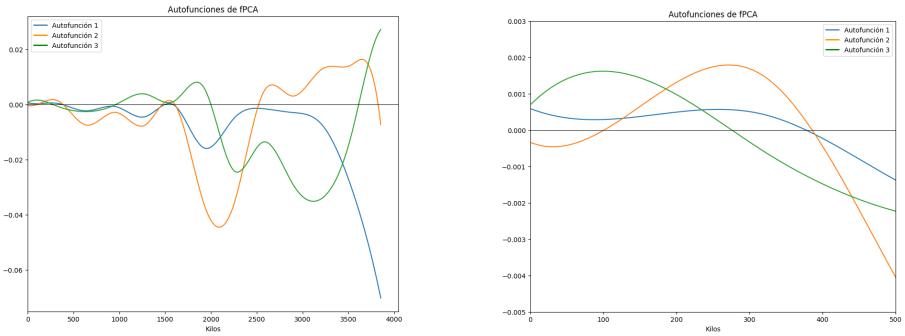
Observing the parameters that make up the mixtures in Figure 4.4, we can see that, in broad terms, the forecasts are around the expected values. What we did was compare the parameters we predicted for 2022 with those obtained by fitting a mixture for the launches of that year. With the exception of the standard deviations, the forecasted parameters appear to be around the expected values. Therefore, the mismatch with the original density should be attributed to the fact that the choice of two lognormals may not be the best for this data.

4.2 Non-Parametric Method

For the functional PCA method, we start with the kernel density estimations (or, more precisely, the log-densities, as discussed in Section 2.2). One difference from the simulations is that, in this case, we represent the log-densities using a basis of 15 B-splines instead of 30. Then, we project them onto their first three principal components, explaining over 99% of the variability. In this way, we transform the problem from a time series of functions into a traditional time series composed of the weights of the eigenfunctions. We fit an ARIMA model for each of the three score series, and with the predictions, we reconstruct the densities $\hat{f}_{T+h}$. We first take the exponential and then divide by the integral to ensure that it is a density function.

With the reconstructed density for 2022, we obtain a KL divergence of 0.06. In Figure 4.7b, we can see a representation that is quite close to the actual density, correctly capturing the peak around 200 kilograms.

Analyzing the eigenfunctions (Figure 4.5) and their weights (Figure 4.6), we see that for 2022, positive weights are assigned to all three eigenfunctions, with the first one having the highest weight and the third one the lowest. Visually,



(a) Entire domain (b) From 0 to 500 kg

Figure 4.5: Inspection of the eigenfunctions. Focusing on the region below 500 kilograms, the second eigenfunction (orange) appears to represent the Starlink peak around 200-300 kilograms, while the third (green) gives more weight to lighter satellites around 0 kilograms. The first eigenfunction (blue) positively weights the entire region, becoming negative starting at 400 kilograms.

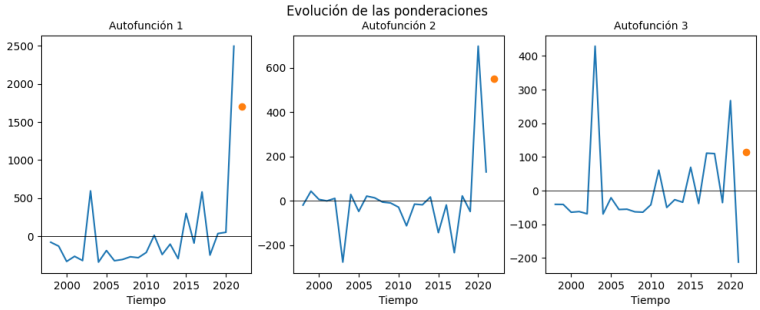


Figure 4.6: Evolution of the weights. The behavior of the score associated with the first eigenfunction is interesting. In Figure 4.5b, it can be seen that the eigenfunction takes positive values below 400 kilograms and negative values for higher masses. Observing the time series (left panel), we note that this function was generally weighted negatively for almost all years in the sample, becoming positive in recent years. This would reflect the significant recent surge in popularity of small satellites at the expense of heavier ones.

the first eigenfunction strongly "penalizes" observations above 3000 kilograms. Focusing on the region between 0 and 500 kilograms, in Figure 4.5b, we observe that the second eigenfunction has a peak between 200 and 300 kilograms that is not counteracted by the others (the first eigenfunction takes positive values in this region as well, and the third eigenfunction becomes close to zero). Figure 4.6 shows that the second eigenfunction has a strong positive weight, suggesting it is responsible for capturing the Starlink peak.

Looking at the results in Figure 4.7b, the predicted function also captures the peak around 0 kilograms. Returning to Figure 4.5b, in that region, both the first and third eigenfunctions take positive values and are positively weighted, so they would be responsible for representing the group of lighter satellites. On the other hand, their effect is dampened, in part, by the second eigenfunction, which takes negative values, causing this peak to have less relevance in the final density than the peak around 200 kilograms.

4.3 Conditional GANs

Lastly, for the method based on Conditional GANs, we trained the model using the same hyperparameters as we did in the previous simulations. We decided to train, just like before, for 2000 iterations, even though past the halfway point of training, there were no additional improvements in the results.

Once the model was trained, we generated new samples conditional on the desired period, in our case for 2022. Based on these samples, we fitted a kernel density function to compare it with the one fitted based on the real data.

The resulting density has a KL divergence of 0.05 compared to the test density. In Figure 4.7c, we observe that the method successfully captures the peak around 200 kilograms with greater precision than the FPCA procedure, although it seems to underestimate the region around 0 to some extent.

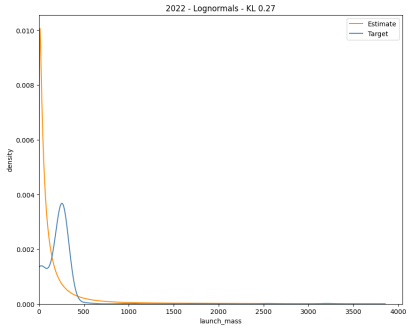
4.4 Results

In Figure 4.7, we present the densities generated by each method and contrast them with the density fitted to the launches of 2022. The parametric method based on a mixture of two lognormals generates the worst of the three fits with a Kullback-Leibler divergence of 0.27. Although it succeeds in concentrating the density below 500 kilograms, it fails to capture the shape of the real distribution, overlooking the main mode (200 kilograms) and instead concentrating around zero.

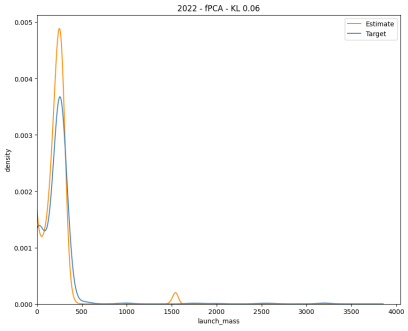
Second in terms of performance, the FPCA procedure achieves a better fit with a KL of 0.06, reconstructing the density as a combination of the first three eigenfunctions. It not only captures the peak at 200 kilograms but also assigns weight to the region around 0. However, the density it assigns to the 200-kilogram peak is higher than it should be, and it also leads to a slight concentration at 1500 kilograms not observed in the real density.

Lastly, the best fit was generated by the GAN model, although it's very close to the FPCA alternative, with a KL of 0.05. The GAN identifies and assigns the right weight to the 200-kilogram peak, although there are some shortcomings, such as lower density than expected in the region around 0. We emphasize that the hyperparameters and architectures of both the generator

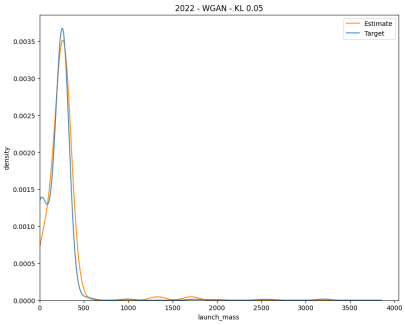
and discriminator are the same as those used throughout this work, indicating the method's potential robustness and versatility.



(a) Parametric Method



(b) Non-Parametric Method - FPCA



(c) Conditional GANs

Figure 4.7: 2022 Predictions. In blue, the density fitted on the 2022 launches; in orange, the forecasted densities. KL between both densities above each figure. The neural network method achieves the best prediction, obtaining the lowest Kullback-Leibler divergence from the true density, mainly capturing the mode around 200 kilograms.

CHAPTER 5

Conclusion

Three alternative methods were presented to address the problem of forecasting time series of density functions, showcasing their theoretical development and performance on simulated and real datasets.

Firstly, the parametric method provides a simple and relatively quick-to-implement alternative, which can serve as a benchmark for comparison with other methods. As we have seen, it can perform very well if the chosen functions to fit coincide with the real functions. However, it suffers from certain disadvantages, especially when attempting to fit more complex functions that require more parameters, leading to more time series to handle. Something we didn't explore in this work but presents as an advantage of this method over the others is that it can be useful when modeling data generated by a family of discrete distributions, such as Poisson or binomial. No changes would be necessary to adapt the method to this problem, as the steps to follow would be the same: fit the parameters on the available data and forecast them for future periods.

The second alternative we presented is based on Functional Principal Component Analysis and is one of the most popular in the literature. For the application case, although it ranked second in terms of performance, it achieved a similar fit to the GAN model. However, it requires special care when interpreting and predicting the eigenfunctions and scores. While it is a relatively quick method to implement, it demands a certain amount of time and some manual work when forecasting the time series, as even a slight change in them can significantly worsen the results. Additionally, although it is not a parametric method, it requires forecasting a number of time series that can scale with the complexity of the functions. This could potentially become an issue, for example, when trying to work with multivariate functions. Furthermore, we observed that the method encounters difficulties when attempting to forecast a non-stationary series of densities, as in the cases we discussed in Sections 3.1 and 3.2, where the functions were not defined on a bounded domain.[9].

Finally, the alternative of generative adversarial networks achieved the best performance of the three in the application case and in some of the simulated sets. The use of this procedure for forecasting functions is innovative, and there is not much literature on its application to this specific use case. Although it takes the longest of the three methods to fit, it is still relatively easy to implement.

[9]Based on comments from the jury, it could be explored to apply Dynamic Time Warping to the densities before the FPCA decomposition to address the issues related to unbounded domains.

We obtained satisfactory results, both in simulated and real data, without deviating significantly from the architectures used in other implementations. Every GAN fitted for this work had the same set of hyperparameters and the same architectures for both the generator and discriminator, giving the impression that it is a versatile method. Unlike what Zaheer et al. [13] propose, we did not encounter issues using this method to learn univariate distributions. On the contrary, in some cases, it even achieved a better fit and future prediction than the other methods. Additionally, as a point to explore in future work, it should be straightforward to scale the method to learn more complex and higher-dimensional distributions, something that could be challenging with the other methods. However, one of its flaws is common to neural network models: it remains a black box. Unlike the FPCA alternative, where we could, for example, interpret each eigenfunction, this third method does not offer much in terms of interpretability.

In Table 5.1 we present a comparison of the three alternatives, highlighting their main advantages and disadvantages:

Method	**Parametric**	**FPCA**	**GAN**
Advantages	Simple and quick to train.	Good performance on stationary series.	Good performance even without changing hyperparameters.
	Suitable for discrete distributions.	Interpretability of eigenfunctions.	Does not require time series modeling.
	Not limited to stationary series.		Scalable to higher dimensions.
Disadvantages	Low flexibility.	Relies on the stationarity assumption.	Training time.
	Limited scalability to more complex functions and larger dimensions.	Limited scalability to higher dimensions.	Limited control and interpretability of results.
	Requires time series modeling.	Requires time series modeling.	

Table 5.1: Comparison of alternatives.

As points to explore, we can mention the application of the methods to forecast the evolution of multivariate densities. Of the three alternatives we developed, the best candidate for distributions in higher dimensions is the one based on generative adversarial networks, considering that they were originally designed to generate complex data.

On the other hand, it would be possible to expand the analysis we conducted on the univariate setting by also exploring the stability of the three methods. In both the simulated data and the application case, we focused on the quality of the fit that the methods provided, but we did not analyze their variability. Hyndman and Shang cover this topic by proposing the use of bootstrap methods

to construct confidence intervals for the estimated functions, although they do not do so specifically for density functions [6].

Bibliography

[1] Arjovsky, M., Chintala, S. and Bottou, L. (2017). "Wasserstein GAN," arXiv preprint arXiv:1701.07875.

[2] Fabbri, C. (2017). "Conditional Wasserstein Generative Adversarial Networks," https://cameronfabbri.github.io/papers/conditionalWGAN.pdf

[3] Goodfellow, I., Pouget-Abadie J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio Y. (2014). "Generative adversarial nets," in Advances in neural information processing systems, pp. 2672–2680.

[4] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V. and Courville, A. (2017). "Improved Training of Wasserstein GANs," in NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems, December 2017, Pages 5769–5779.

[5] Horta, E. and Ziegelmann, F. (2018). Dynamics of financial returns densities: A functional approach applied to the Bovespa intraday index. International Journal of Forecasting 34 (2018) 75–88.

[6] Hyndman, R.J. and Shang, H.L. (2009). "Forecasting functional time series," in Journal of the Korean Statistical Society 38 (2009) 199-211.

[7] Kullback, S. and Leibler, R.A. (1951). "On information and sufficiency". Annals of Mathematical Statistics. 22 (1): 79–86. doi:10.1214/aoms/1177729694.

[8] Mirza, M. and Osindero, S. (2014). "Conditional generative adversarial nets," arXiv preprint arXiv:1411.1784.

[9] Nicol, F. (2013). Functional Principal Component Analysis of Aircraft Trajectories. [Research Report] RR/ENAC/2013/02, ENAC. hal-01349113.

[10] Ramsay, J.O. and Silverman, B.W. (2005). Functional Data Analysis. Springer, New York. http://dx.doi.org/10.1002/0470013192.bsa239.

[11] Sen, R. and Ma, C. (2015). Forecasting Density Function: Application in Finance. Journal of Mathematical Finance, 5, 433-447. doi: 10.4236/jmf.2015.55037.

[12] Silverman B.W. (1986). Density Estimation for Statistics and Data Analysis. Chapman & Hall, London.

[13] Zaheer, M., Li, C., Póczos, B. and Salakhutdinov, R. (2017). "GAN Connoisseur: Can GANs Learn Simple 1D Parametric Distributions?," In NIPS Workshop on Deep Learning: Bridging Theory and Practice.