




## Article

# Kernel Learning by Spectral Representation and Gaussian Mixtures

Luis R. Pena-Llamas <sup>1</sup> , Ramon O. Guardado-Medina <sup>2,\*</sup> , Arturo Garcia <sup>2</sup>  and Andres Mendez-Vazquez <sup>1</sup>

<sup>1</sup> Department of Computer Science, El Centro de Investigación y de Estudios Avanzados (CINVESTAV), Ciudad de Mexico 44960, Mexico

<sup>2</sup> Department of Research, Escuela Militar de Mantenimiento y Abastecimiento, Universidad del Ejercito y Fuerza Aerea, Zapopan 45200, Mexico

\* Correspondence: osvaldomedina2@gmail.com

**Abstract:** One of the main tasks in kernel methods is the selection of adequate mappings into higher dimension in order to improve class classification. However, this tends to be time consuming, and it may not finish with the best separation between classes. Therefore, there is a need for better methods that are able to extract distance and class separation from data. This work presents a novel approach for learning such mappings by using locally stationary kernels, spectral representations and Gaussian mixtures.

**Keywords:** non-parametric kernel learning; approximating kernel; kernel spectral representation; locally stationary kernel



**Citation:** Pena-Llamas, L.; Guardado-Medina, R.; Garcia, A.; Mendez-Vazquez, A. Kernel Learning by Spectral Representation and Gaussian Mixtures. *Appl. Sci.* **2023**, *13*, 2473. <https://doi.org/10.3390/app13042473>

Academic Editor: Lidia Jackowska-Strumillo

Received: 18 December 2022

Revised: 25 January 2023

Accepted: 30 January 2023

Published: 14 February 2023



**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

During the 90's, the use of kernels [1–5] in Machine Learning received a considerable attention for their ability to improve the performance of linear classifiers. By using kernels, Support Vector Machines and other kernel methods [6,7] can classify complex data sets by mapping them into high dimensional spaces. However, an underlying issue exists, summarized by a simple question: Which kernel should be used? [8].

Kernel selection is not a small task and would highly depend on the problem to be solved. A first idea to select the best kernel could be evaluating different kernels from a small set using leave-one-out cross-validation and selecting the kernel with better classification properties. Nevertheless, this can become a time-consuming task when the number of samples range in the thousands. A better idea is to use a combination of kernels to create kernels with better classification properties. Methods using this type of techniques are called Multiple Kernel Learning (MKL) [9]. For example, Lanckriet [10] uses a Semi-Definite Programming (SDP) to find the best conic combination of multiple kernels. However, these methods still require some pre-selected set of kernels as inputs. A better plan will be to use the distance information at the class data sets. For example, Hoi [11] tries to find a kernel Gram matrix by building the Laplacian Graph [12] of the data. Then, an SDP is applied to find the best combination of kernels.

However, none of these methods are scalable given that their Gram matrix needs to be built the computational complexity of building a Gram matrix is  $O(N^2)$  where  $N$  is the number of samples. As a possible solution, it has been proposed to use Sequential Minimal Optimization (SMO) [13] to reduce complexity. This allows to move the quadratic programming problem to a quadratic programming sub-problems. For example, Bach [14] uses an SDP setup and solves the problem by using an SMO algorithm. Other techniques [15] use a random sample of the training set, and a possible approximation to the Gram Matrix to reduce complexity  $O(m^2N + m^2 + mN)$ ,  $m$  sub-problem size). Expanding on this idea, Rahimi [16] approximates kernel functions using samples of the distribution, but only for stationary kernels. On the other hand, Ghiasi-Shirazi [17] proposes a method for learning

$m$  number of stationary kernels in the approach of MKL. The method has a main advantage, its ability to learn  $m$  number of kernels in an unsupervised way by reducing the complexity of the output function. Furthermore, it reduces the complexity of the classifier output from  $O(mxNxN_SV)$  to  $O(mxN)$  by using  $m$  kernels. Finally, Oliva [18], makes use of Bayesian methods to learn a stationary kernel in a non-parametric way.

On this work, we propose to learn locally stationary kernel from data, given that stationary kernels are a subset of the locally stationary kernel, by using a spectral representation and Gaussian Mixtures [19]. This allows to improve the classification and regression task by looking at the kernel as the result of a sampling process on a spectral representation. This paper is structured in the following way: in Section 2, we show the basic theory to understand the idea of stationary and locally stationary kernels. In Section 3, the proposed algorithm is developed by using Fourier Basis and sampling. Additionally, a theorem is given about the performance of the spectral representation. In Section 4, we review the experiments for classification and regression tasks to test the robustness of the proposed algorithm. Finally, we present an analysis of the advantages of the proposed algorithm and possible venues of research in the Section 5.

## 2. The Concept of Kernels

The main idea of using kernel methods is to obtain the distance between samples on higher dimensional. Thus, avoid mapping the samples to higher dimensional spaces and using the inner product for such process. In other words, let  $\mathcal{X} \subseteq \mathbb{R}^D$  be the input set, where  $D \in \mathbb{N}$ , let  $\mathcal{K}$  be a feature space, and suppose the feature mapping function is defined as  $\varphi: \mathcal{X} \rightarrow \mathcal{K}$ . Hence, the kernel function,  $\kappa: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ , has the following property:

$$\kappa(\mathbf{x}, \mathbf{x}') \equiv \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle,$$

where  $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ . Thus,  $\varphi$  and the feature space can be implicitly defined. Now, let  $\{\mathbf{x}_i\}_{i=1}^N$  be the set of samples,  $\kappa: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  be a valid kernel, and  $\langle \cdot, \cdot \rangle$  be a well defined inner product. Then, the elements at the Gram Matrix,  $K \in \mathbb{R}^{N \times N}$ , are computed using the  $\kappa$  mapping,  $K_{ij} := \kappa(\mathbf{x}_i, \mathbf{x}_j)$ . Given this definition, Genton [20] makes an in-depth study of the class of kernel from a statistics perspective, i.e., the kernel functions as a co-variance function. He pointed out that kernels have a spectral representation which can be used to represent their Gram matrix. Based on this representation, the proposed algorithm learns the Gram matrix by using a Gibbs sampler to obtain the structure of such matrix.

### 2.1. Stationary Kernels

Stationary kernels [20] are defined as  $\kappa(\mathbf{x}, \mathbf{x}') = \kappa_s(\mathbf{x} - \mathbf{x}')$ . An important factor in such definition is its dependency on the lag vector which can be interpreted as generalizations of the Gaussian probability distribution functions which are used to represent distributions [15]. Additionally, Bochner [21] proved that a symmetric function  $\kappa_s$  is a positive definite in  $\mathbb{R}^D$ , if and only if, it has the form:

$$\kappa_s(\mathbf{x} - \mathbf{x}') = \int_{\mathbb{R}^D} e^{2\pi i \boldsymbol{\omega}^T (\mathbf{x} - \mathbf{x}')} d\mu(\boldsymbol{\omega}), \quad (1)$$

where  $\mu$  is a positive finite measure. Equation (1) is called the *spectral representation* of  $\kappa_s$ . Now, suppose  $\mu$  has a density  $F(\boldsymbol{\omega})$  and  $\boldsymbol{\tau} = \mathbf{x} - \mathbf{x}'$ . Thus, it is possible to obtain:

$$\begin{aligned} \kappa(\boldsymbol{\tau}) &= \int F(\boldsymbol{\omega}) e^{2\pi i \boldsymbol{\omega}^T \boldsymbol{\tau}} d\boldsymbol{\omega}, \\ F(\boldsymbol{\omega}) &= \int \kappa(\boldsymbol{\tau}) e^{-2\pi i \boldsymbol{\omega}^T \boldsymbol{\tau}} d(\boldsymbol{\tau}). \end{aligned}$$

In other words, the kernel function  $\kappa_s$  and its spectral density  $F$  are Fourier dual of each other. Furthermore, if  $\kappa(\mathbf{0}) = \int F(\boldsymbol{\omega}) d\boldsymbol{\omega}$  and  $F$  is a probability measure, the unique

condition to define a **valid** Gaussian process is  $\kappa(\mathbf{0}) = 1$ . In other words, we need this condition to ensure that the kernel  $\kappa$  and the function  $f$  are correctly correlated.

## 2.2. Locally Stationary Kernels

Extending on the previous concept, Silverman [22] defines the locally stationary kernels as:

$$\kappa(\mathbf{x}, \mathbf{x}') := \kappa_1\left(\frac{\mathbf{x} + \mathbf{x}'}{2}\right) \kappa_s(\mathbf{x} - \mathbf{x}'), \quad (2)$$

where  $\kappa_1$  is a non negative function and  $\kappa_s$  is a stationary kernel. This type of kernels increase the power of the representation by introducing a possible variance into the final calculated similarity through the use of  $\kappa_1$ . Furthermore, we can see from Equation (2) that the Locally Stationary Kernels include all stationary kernels. In order to see this, we make  $\kappa_1(\cdot) = c$ , where  $c$  is a positive constant, then  $\kappa(\mathbf{x}, \mathbf{x}') := c\kappa_s(\mathbf{x} - \mathbf{x}')$ , a multiple of all stationary kernels. Furthermore, the variance of locally stationary kernels is given by  $\mathbf{x} = \mathbf{x}'$ , thus, the variance is defined as:

$$\kappa(\mathbf{x}, \mathbf{x}) = \kappa(\mathbf{x})\kappa(\mathbf{0}) = \kappa_1(\mathbf{x}),$$

This means that the variance of the Locally Stationary Kernels relies in the positive definite function  $\kappa_1$ .

The spectral representation of a locally stationary kernel is also given by [22], and it is defined as:

$$\kappa_1\left(\frac{\mathbf{x} + \mathbf{x}'}{2}\right) \kappa_s(\mathbf{x} - \mathbf{x}') = \int_{\mathcal{X}} \int_{\mathcal{X}} \exp\left(i\frac{\boldsymbol{\omega}_1^T \mathbf{x} - \boldsymbol{\omega}_2^T \mathbf{x}'}{2}\right) f_1\left(\frac{\boldsymbol{\omega}_1 + \boldsymbol{\omega}_2}{2}\right) f_2(\boldsymbol{\omega}_1 - \boldsymbol{\omega}_2) d\boldsymbol{\omega}_1 d\boldsymbol{\omega}_2.$$

Furthermore, by setting  $\mathbf{x} = \mathbf{x}' = \mathbf{0}$ , we can get:

$$\kappa(\mathbf{0}, \mathbf{0}) = \int_{\mathcal{X}} \int_{\mathcal{X}} f_1\left(\frac{\boldsymbol{\omega}_1 + \boldsymbol{\omega}_2}{2}\right) f_2(\boldsymbol{\omega}_1 - \boldsymbol{\omega}_2) d\boldsymbol{\omega}_1 d\boldsymbol{\omega}_2.$$

Consequently, in order to define a locally stationary kernel,  $f_1$  and  $f_2$  must be integrable functions. Additionally, an important fact is that the kernel  $\kappa$  has a defined inverse, given by:

$$f_1\left(\frac{\boldsymbol{\omega}_1 + \boldsymbol{\omega}_2}{2}\right) f_2(\boldsymbol{\omega}_1 - \boldsymbol{\omega}_2) = \frac{1}{(2\pi)^2} \int_{\mathcal{X}} \int_{\mathcal{X}} \kappa_1\left(\frac{\mathbf{x}_1 + \mathbf{x}'}{2}\right) \kappa_2(\mathbf{x} - \mathbf{x}') d\mathbf{x} d\mathbf{x}'.$$

Moreover,  $f_2$  is the Fourier transform of  $\kappa_1$  and  $f_1$  is the Fourier transform of  $\kappa_2$ . Thus, if we introduce two dummy variables  $\mathbf{u} = (\mathbf{x} + \mathbf{x}')/2$  and  $\mathbf{v} = \mathbf{x} - \mathbf{x}'$ , it is possible to obtain:

$$\begin{aligned} f_1(\boldsymbol{\omega}_1) &= \frac{1}{2\pi} \int_{\mathcal{X}} \exp(-i\mathbf{v}^T \boldsymbol{\omega}_1) \kappa_2(\mathbf{v}) d\mathbf{v} \\ f_2(\boldsymbol{\omega}_2) &= \frac{1}{2\pi} \int_{\mathcal{X}} \exp(-i\mathbf{u}^T \boldsymbol{\omega}_2) \kappa_1(\mathbf{u}) d\mathbf{u} \end{aligned}$$

and

$$\begin{aligned} \kappa_1(\mathbf{u}) &= \int_{\mathcal{X}} \exp(i\mathbf{u}^T \boldsymbol{\omega}_2) f_2(\boldsymbol{\omega}_2) d\boldsymbol{\omega}_2 \\ \kappa_2(\mathbf{v}) &= \int_{\mathcal{X}} \exp(i\mathbf{v}^T \boldsymbol{\omega}_1) f_1(\boldsymbol{\omega}_1) d\boldsymbol{\omega}_1, \end{aligned}$$

with this in mind, it is possible to use the ideas in [16] to approximate the locally stationary kernels.

### 3. Approximating Stationary Kernels

Rahimi [16] makes use of (1) to approximate stationary kernels. This is, if we define  $\zeta_{\omega} = \exp(i\omega^T \mathbf{x})$ ; then, Equation (1) becomes:

$$\kappa(\mathbf{x} - \mathbf{x}') = \int_{\mathcal{X}} f(\omega) \exp(i\omega^T (\mathbf{x} - \mathbf{x}')) d\omega = \mathbb{E}_{\omega} [\zeta_{\omega}(\mathbf{x}) \zeta_{\omega}^*(\mathbf{x}')] ]$$

where  $\omega \sim f$ . Now, using Monte Carlo integration and taking  $\omega_j \sim f$ , the kernel can be approximated as

$$\kappa(\mathbf{x} - \mathbf{x}') \approx \frac{1}{M_1} \sum_{j=1}^{M_1} \zeta_{\omega_j}(\mathbf{x}) \zeta_{\omega_j}^*(\mathbf{x}'). \quad (3)$$

In particular, if the kernel is real-valued; then, Equation (3) becomes

$$\kappa(\mathbf{x} - \mathbf{x}') \approx \frac{1}{M_1} \boldsymbol{\phi}^T(\mathbf{x}) \boldsymbol{\phi}(\mathbf{x}'), \quad (4)$$

where  $\boldsymbol{\phi}(\mathbf{s}) = [\cos(\omega_1^T \mathbf{s}), \dots, \cos(\omega_{M_1}^T \mathbf{s}), \sin(\omega_1^T \mathbf{s}), \dots, \sin(\omega_{M_1}^T \mathbf{s})]$ . A side effect of (4) is that we can compute  $f(\mathbf{x})$  as  $f(\mathbf{x}) = \sum_{i=1}^n \alpha_i \kappa(\mathbf{x}_i - \mathbf{x})$ . This means that function  $f$  can be approximated as

$$f(\mathbf{x}) \approx \frac{1}{M} \sum_{i=1}^n \alpha_i \boldsymbol{\phi}(\mathbf{x}_i)^T \boldsymbol{\phi}(\mathbf{x}) = \boldsymbol{\gamma}^T \boldsymbol{\phi}(\mathbf{x})$$

where  $\boldsymbol{\gamma} = \frac{1}{M} \sum_{i=1}^n \alpha_i \boldsymbol{\phi}(\mathbf{x}_i)$  is a constant. This constant makes possible to avoid some of the operations to obtain the Gram matrix.

#### 3.1. Approximating Locally Stationary Kernel

As we know,  $\kappa_2$  is a stationary kernel which allows to approximate  $\kappa_2$  as presented in Section 3. Now, to obtain the locally stationary kernel, we would like to approximate  $\kappa_1$ . For this, we define  $\zeta'_{\mathbf{v}}(\mathbf{x}) = \exp(i\mathbf{v}^T \mathbf{x}/2)$ :

$$\kappa_1\left(\frac{\mathbf{x} + \mathbf{x}'}{2}\right) = \int_{\mathbb{R}^D} \exp\left(i\mathbf{v}^T \left(\frac{\mathbf{x} + \mathbf{x}'}{2}\right)\right) f_2(\mathbf{v}) d\mathbf{v} = \mathbb{E}_{\mathbf{v}} [\zeta'_{\mathbf{v}}(\mathbf{x}) \zeta'_{\mathbf{v}}(\mathbf{x}')],$$

where  $\mathbf{v} \sim f_2$ . Using Monte Carlo integration and taking  $\mathbf{v}_k \sim f_2$ , for  $k = 1, 2, \dots, M_2$ , it is possible to approximate  $\kappa_1$  as:

$$\kappa_1\left(\frac{\mathbf{x} + \mathbf{x}'}{2}\right) \approx \frac{1}{M_2} \sum_{k=1}^{M_2} \zeta'_{\mathbf{v}_k}(\mathbf{x}) \zeta'_{\mathbf{v}_k}(\mathbf{x}'). \quad (5)$$

To approximate the output of the locally stationary kernel, we can use Equations (3) and (5) as follows:

$$\begin{aligned} \kappa(\mathbf{x}, \mathbf{x}') &= \kappa_1\left(\frac{\mathbf{x} + \mathbf{x}'}{2}\right) \kappa_2(\mathbf{x} - \mathbf{x}') \\ &\approx \frac{1}{M_1 M_2} \left( \sum_{k=1}^{M_2} \exp\left(i \frac{\mathbf{v}_k^T \mathbf{x}}{2}\right) \exp\left(i \frac{\mathbf{v}_k^T \mathbf{x}'}{2}\right) \right) \left( \sum_{n=1}^{M_1} \exp\left(i \omega_n^T \mathbf{x}\right) \exp\left(-i \omega_n^T \mathbf{x}'\right) \right) \\ &= \frac{1}{M_1 M_2} \sum_{n=1}^{M_1} \sum_{k=1}^{M_2} \exp\left(i \left(\frac{\mathbf{v}_k}{2} + \omega_n\right)^T \mathbf{x}\right) \exp\left(i \left(\frac{\mathbf{v}_k}{2} - \omega_n\right)^T \mathbf{x}'\right) \end{aligned}$$

where  $\omega_n \sim f_1$  and  $\mathbf{v}_k \sim f_2$ . In particular, if our kernel is real-valued, then previous equation becomes

$$\kappa_1\left(\frac{\mathbf{x} + \mathbf{x}'}{2}\right) \kappa_2(\mathbf{x} - \mathbf{x}') \approx \frac{1}{M_1 M_2} \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}^-(\mathbf{x}') \quad (6)$$

where

$$\begin{aligned}\boldsymbol{\varphi}(\mathbf{s}) &= \left[ \cos\left(\left(\frac{\mathbf{v}_1}{2} + \boldsymbol{\omega}_1\right)^T \mathbf{s}\right), \dots, \cos\left(\left(\frac{\mathbf{v}_{M_2}}{2} + \boldsymbol{\omega}_{M_1}\right)^T \mathbf{s}\right), \sin\left(\left(\frac{\mathbf{v}_1}{2} + \boldsymbol{\omega}_1\right)^T \mathbf{s}\right), \dots, \sin\left(\left(\frac{\mathbf{v}_{M_2}}{2} + \boldsymbol{\omega}_{M_1}\right)^T \mathbf{s}\right) \right] \\ \boldsymbol{\varphi}^-(\mathbf{s}) &= \left[ \cos\left(\left(\frac{\mathbf{v}_1}{2} - \boldsymbol{\omega}_1\right)^T \mathbf{s}\right), \dots, \cos\left(\left(\frac{\mathbf{v}_{M_2}}{2} - \boldsymbol{\omega}_{M_1}\right)^T \mathbf{s}\right), -\sin\left(\left(\frac{\mathbf{v}_1}{2} - \boldsymbol{\omega}_1\right)^T \mathbf{s}\right), \dots, -\sin\left(\left(\frac{\mathbf{v}_{M_2}}{2} - \boldsymbol{\omega}_{M_1}\right)^T \mathbf{s}\right) \right]\end{aligned}$$

and  $\boldsymbol{\omega}_n \sim f_1$ ,  $\mathbf{v}_k \sim f_2$ . Thus, the advantage of representing the locally stationary kernel as Equation (6) is the possibility of computing  $f(\mathbf{x})$  as:

$$f(\mathbf{x}) = \sum_{j=1}^N \alpha_i \kappa(\mathbf{x}_j, \mathbf{x}) \approx \frac{1}{M_1 M_2} \sum_{j=1}^N \alpha_i \boldsymbol{\varphi}^T(\mathbf{x}_j) \boldsymbol{\varphi}^-(\mathbf{x}) = \boldsymbol{\psi}^T \boldsymbol{\varphi}^-(\mathbf{x})$$

where  $\boldsymbol{\psi} = \frac{1}{M_1 M_2} \sum_{j=1}^N \alpha_i \boldsymbol{\varphi}^T(\mathbf{x}_j)$ . Given this representation, we only need to compute  $\boldsymbol{\psi}$  once, avoiding the use of total Gram Matrix representation.

Now, it is necessary to remark an interesting property of using this representation. It is possible to say that  $|\boldsymbol{\varphi}^T(\mathbf{x}) \boldsymbol{\varphi}^-(\mathbf{x}') - \kappa(\mathbf{x}, \mathbf{x}')| \leq C$  (using the Hoedding's inequality [23]) almost everywhere. Given this, it is possible to obtain the following inequality: given any  $\epsilon > 0$ , and taking samples  $M_1$  and  $M_2$  from  $\kappa_2$  and  $\kappa_1$  respectively; then

$$P\left(|\boldsymbol{\varphi}^T(\mathbf{x}) \boldsymbol{\varphi}^-(\mathbf{x}') - \kappa(\mathbf{x}, \mathbf{x}')| \geq \epsilon\right) \leq 2 \exp\left(-\frac{M_1 M_2 \epsilon^2}{2(\sigma^2 + 2\epsilon^2/3)}\right).$$

Therefore, the proposed representation of the kernel allows to obtain a good approximation to  $\boldsymbol{\varphi}^T(\mathbf{x}) \boldsymbol{\varphi}^-(\mathbf{x}')$ . Furthermore, the following theorem gives a tighter bound making possible to say: Given a larger  $\epsilon$ , the less likely is the possibility of having a larger  $|\boldsymbol{\varphi}^T(\mathbf{x}) \boldsymbol{\varphi}^-(\mathbf{x}') - \kappa(\mathbf{x}, \mathbf{x}')|$ .

**Theorem 1.** *Approximation of a locally stationary kernel.*

Let  $\mathcal{M}$  be a compact subset of  $\mathbb{R}^D$  with diameter  $\text{Diam}(\mathcal{M})$ , and  $\sigma^2 > \epsilon/2$  then the approximation of the kernel is given by:

$$P\left(\sup_{\mathbf{x}, \mathbf{x}'} |\boldsymbol{\varphi}^T(\mathbf{x}) \boldsymbol{\varphi}^-(\mathbf{x}') - \kappa(\mathbf{x}, \mathbf{x}')| \geq \epsilon\right) < \frac{2\sigma^2}{\epsilon} \left(1 + 4\text{Diam}(\mathcal{M}) \exp\left(-\frac{M_1 M_2 \epsilon^2}{4D(\sigma^2 + 2\epsilon^2/3)}\right)\right)$$

**Proof of Theorem 1.** Define  $s(\mathbf{x}, \mathbf{x}') \equiv \boldsymbol{\varphi}^T(\mathbf{x}) \boldsymbol{\varphi}^-(\mathbf{x}')$ , a locally stationary kernel  $\kappa(\mathbf{x}, \mathbf{x}') \equiv \kappa_1\left(\frac{\mathbf{x} + \mathbf{x}'}{2}\right) \kappa_2(\mathbf{x} - \mathbf{x}')$  and  $f(\mathbf{x}, \mathbf{x}') = |s(\mathbf{x}, \mathbf{x}') - \kappa(\mathbf{x}, \mathbf{x}')| \leq 2$ . Then, it is possible to say that  $\mathbb{E}[f(\mathbf{x}, \mathbf{x}')] = 0$ . Given that  $\kappa_2$  is shift invariant, it is possible to define  $\Delta_- \equiv \mathbf{x} - \mathbf{x}' \in \mathcal{M}_-$ . Now, given that  $\kappa_1$  can be interpreted as the mean, it is possible to define  $\Delta_+ \equiv \frac{\mathbf{x} + \mathbf{x}'}{2} \in \mathcal{M}_+$ . Consequently, it is possible to define  $\kappa(\Delta_-, \Delta_+) = \kappa_1(\Delta_+) \kappa_2(\Delta_-)$ . Let  $\mathcal{M} \subset \mathbb{R}^D$  a compact bounded subset, it is known that  $\text{Diam}(\mathcal{M}_-) \leq 2\text{Diam}(\mathcal{M})$  and  $\text{Diam}(\mathcal{M}_+) \leq 2\text{Diam}(\mathcal{M})$ . With this in mind, it is possible to define  $\epsilon$ -net that covers  $\mathcal{M}_- \times \mathcal{M}_+$  at most  $T = \left(\frac{4\text{Diam}(\mathcal{M})}{r}\right)^{2D}$  balls of radius  $r$ . Let  $\{\Delta_{-,i}, \Delta_{+,i}\}_{i=1}^T$  denote the center of these  $T$  balls, and let  $L_f$  be the Lipschitz constant of  $f$ . Therefore, we have that  $|f(\Delta_-, \Delta_+)| < \epsilon$  for all  $\Delta_-, \Delta_+ \in \mathcal{M}_- \times \mathcal{M}_+$ . Then,  $f(\Delta_{-,i}, \Delta_{+,i}) < \frac{\epsilon}{2}$  and  $L_f < \frac{\epsilon}{2r}$  for all  $i$ . Now,  $L_f = \|\nabla f(\Delta_{+,i}^*, \Delta_{-,i}^*)\|$ , where  $(\Delta_+^*, \Delta_-^*) = \max_{\Delta_+, \Delta_- \in \mathcal{M}_+ \times \mathcal{M}_-} \|\nabla f(\Delta_+, \Delta_-)\|$ . Additionally, we know  $\mathbb{E}[\nabla s(\Delta_+, \Delta_-)] = \nabla \kappa(\Delta_+, \Delta_-)$ . Thus, it is possible to say:

$$\mathbb{E}[L_f^2] = \mathbb{E}[\|\nabla s(\Delta_+^*, \Delta_-^*) - \kappa(\Delta_+^*, \Delta_-^*)\|^2] = \mathbb{E}[\|s(\Delta_+^*, \Delta_-^*)\|^2] + \mathbb{E}[\|\kappa(\Delta_+^*, \Delta_-^*)\|^2]$$

Now, given that  $\mathbb{E}[\|s(\Delta_+^*, \Delta_-^*)\|^2]$  and  $\mathbb{E}[\|\kappa(\Delta_+^*, \Delta_-^*)\|^2]$  are positive,

$$\mathbb{E}[\|s(\Delta_+^*, \Delta_-^*)\|^2] - \mathbb{E}[\|\kappa(\Delta_+^*, \Delta_-^*)\|^2] \leq \mathbb{E}[\|s(\Delta_+^*, \Delta_-^*)\|^2] = \sigma^2$$

with  $\mathbb{E}[L_f^2] = \sigma^2$  and  $\mathbb{E}[L_f] = \sigma$ , where  $\sigma^2$  is the second momentum of Fourier transform of  $\kappa$ . Thus, using the Markov's inequality,

$$P\left(L_f \geq t\right) \leq \frac{\mathbb{E}[L_f]}{t},$$

$$P\left(L_f \geq \frac{\epsilon}{2r}\right) \leq \frac{2r\sigma^2}{\epsilon}$$

Finally, using the Boole's inequality we have

$$P\left(\cup_{i=1}^T |f(\Delta_{+,i}^*, \Delta_{-,i}^*)| \geq \frac{\epsilon}{2}\right) \leq 2T \exp\left(-\frac{M_1 M_2 \epsilon^2}{2(\sigma^2 + 2\epsilon^2/3)}\right)$$

With this at hand, it is possible to say:

$$P\left(\sup_{\mathbf{x}, \mathbf{x}'} |\boldsymbol{\varphi}^T(\mathbf{x}) \boldsymbol{\varphi}(\mathbf{x}') - \kappa(\mathbf{x}, \mathbf{x}')| \geq \epsilon\right) \leq 1 - 2\left(\frac{4Diam(\mathcal{M})}{r}\right)^{2D} \exp\left(-\frac{M_1 M_2 \epsilon^2}{2(\sigma^2 + 2\epsilon^2/3)}\right) - \frac{2r\sigma^2}{\epsilon}$$

Meaning that we need to solve the following equation

$$1 - k_1 r^{-2D} - k_2 r, \quad (7)$$

where

$$k_1 = 2(4Diam(\mathcal{M}))^{2D} \exp\left(-\frac{M_1 M_2 \epsilon^2}{2(\sigma^2 + 2\epsilon^2/3)}\right),$$

$$k_2 = \frac{2\sigma^2}{\epsilon}$$

The solution of (7) is given by  $r = \left(\frac{k_1}{k_2}\right)^{\frac{1}{2D}}$ . Then, plugging back this result,

$$1 - k_1 \left(\left(\frac{k_1}{k_2}\right)^{\frac{1}{2D}}\right)^{-2D} - k_2 \left(\frac{k_1}{k_2}\right)^{\frac{1}{2D}}.$$

After some development, it is possible to obtain:

$$k_1 \left(\left(\frac{k_1}{k_2}\right)^{\frac{1}{2D}}\right)^{-2D} = \frac{2\sigma^2}{\epsilon}, \quad k_2 \left(\frac{k_1}{k_2}\right)^{\frac{1}{2D}} = 2\left(\frac{\sigma^2}{\epsilon}\right)^{\frac{2D-1}{2D}} 4Diam(\mathcal{M}) \exp\left(-\frac{M_1 M_2 \epsilon^2}{4D(\sigma^2 + 2\epsilon^2/3)}\right)$$

Using this equality, we get (8) and (9).

$$P\left(\sup_{\mathbf{x}, \mathbf{x}'} |\boldsymbol{\varphi}^T(\mathbf{x}) \boldsymbol{\varphi}(\mathbf{x}') - \kappa(\mathbf{x}, \mathbf{x}')| \leq \epsilon\right) \geq 1 - \frac{2\sigma^2}{\epsilon} - 2\left(\frac{\sigma^2}{\epsilon}\right)^{\frac{2D-1}{2D}} 4Diam(\mathcal{M}) \exp\left(-\frac{M_1 M_2 \epsilon^2}{4D(\sigma^2 + 2\epsilon^2/3)}\right) \quad (8)$$

$$P\left(\sup_{\mathbf{x}, \mathbf{x}'} |\boldsymbol{\varphi}^T(\mathbf{x}) \boldsymbol{\varphi}(\mathbf{x}') - \kappa(\mathbf{x}, \mathbf{x}')| \geq \epsilon\right) \leq \frac{2\sigma^2}{\epsilon} + 2\left(\frac{\sigma^2}{\epsilon}\right)^{\frac{2D-1}{2D}} 4Diam(\mathcal{M}) \exp\left(-\frac{M_1 M_2 \epsilon^2}{4D(\sigma^2 + 2\epsilon^2/3)}\right) \quad (9)$$

Now, if  $\sigma^2 > \epsilon/2$ , then  $\frac{2\sigma^2}{\epsilon} + \frac{2\sigma^2}{\epsilon}^{\frac{2D-1}{2D}} x < \frac{2\sigma^2}{\epsilon}(1+x)$ . Finally:

$$P\left(\sup_{\mathbf{x}, \mathbf{x}'} |\boldsymbol{\varphi}^T(\mathbf{x}) \boldsymbol{\varphi}(\mathbf{x}') - \kappa(\mathbf{x}, \mathbf{x}')| \geq \epsilon\right) < \frac{2\sigma^2}{\epsilon} \left(1 + 4Diam(\mathcal{M}) \exp\left(-\frac{M_1 M_2 \epsilon^2}{4D(\sigma^2 + 2\epsilon^2/3)}\right)\right)$$

□

### 3.2. Learning Locally Stationary Kernel, GaBaSR

In this section, we explain how to learn the proposed stationary kernel. This learning algorithm is based on the work presented in [18], named Bayesian Nonparametric Kernel (BaNK) algorithm. However, given its greatest representation capabilities, we propose learning a Gaussian mixture distribution to improve the performance of the algorithm. For this reason, we name this model as Gaussian Mixture Bayesian Nonparametric Kernel Learning using Spectral Representation (GaBaSR). Furthermore, to learn the Gaussian mixture, the proposed algorithm uses ideas proposed in [15], together with a different way to learn the kernel in the classification task. Additionally, one of its main advantages is the use of vague/non-informative priors, [15,24], as well as having fewer hyperparameters for learning the kernels.

#### 3.2.1. GaBaSR Algorithm

Based in the previous ideas, we have the following high level description of the algorithm.

1. Learn all the parameters for the Gaussian mixture  $\rho(\omega)$ :
  - Let  $\{\pi_k, \mu_k, \Sigma_k\}_{k=1}^K$  be the current parameters of the Gaussian Mixture Model (GMM), where  $\pi_k$  is the prior probability of the  $k$ th component,  $\mu_k$  is the mean and  $\Sigma_k$  is the covariance matrix of the  $k$ th component, then the GMM is given by  $\rho(\omega) = \sum_{k=1}^K \pi_k \mathcal{N}(\omega | \mu_k, \Sigma_k)$ , here the output will be the new sample parameters for the GMM  $\rho(\omega)$ .
2. Take  $M$  samples from  $\rho(\omega)$ , i.e.,  $\omega_i \sim \rho(\omega)$ ,  $i = 1, 2, \dots, M$  for the spectral representation.
  - Here the input are the parameters of the GMM, and the frequencies  $\omega_i$ ,  $i = 1, \dots, M$ , and the output will be the new frequencies sampled.
3. Approximate the kernel as

$$\kappa_1\left(\frac{\mathbf{x} + \mathbf{x}'}{2}\right) \kappa_2(\mathbf{x} - \mathbf{x}') \approx \frac{1}{M_1 M_2} \boldsymbol{\varphi}(\mathbf{x})^T \boldsymbol{\varphi}^-(\mathbf{x}')$$

4. Predict the new samples:
  - (a) If the task is a regression use:

$$f(\mathbf{x}) = \mathcal{N}(\boldsymbol{\beta}^T \boldsymbol{\varphi}(\mathbf{x}), \sigma^2)$$

- (b) If the task is a classification use:

$$f(\mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\beta}^T \boldsymbol{\varphi}(\mathbf{x}))}$$

In this work, we use a Markov Chain Monte Carlo (MCMC) algorithm, the Gibbs sampler [25], to learn and predict new inputs. The final entire process is described in the following subsections.

#### 3.2.2. Learning the Gaussian Mixture

In order to learn the parameters  $Z_i$ ,  $\mu_k$ , and  $\Sigma_k$  of the Gaussian mixture, we take the following steps:

1. **First sample  $Z_i$ :**  
 $Z_i$  indicates the component of the Gaussian Mixture from which the random frequency  $\omega_i$  is drawn.  
 For  $i = 1, 2, \dots, M$  do:
  - (a) The element  $\omega_i$  belongs to class  $k = 1, 2, \dots, K$  with probability:

$$p(z_i = k | \mathbf{Z}_{-i}, \alpha, \mu_k, \Lambda_k) \propto \frac{N_{-i,k}}{N - 1 + \alpha} |\Lambda_k|^{1/2} e^{-1/2(\omega_i - \mu_k)^T \Lambda_k (\omega_i - \mu_k)}$$



- (b) The element  $\omega_i$  belongs to an unrepresented class, with probability:

$$p(z_i = k' | \alpha, \mu_{k'}, \Lambda_{k'}) \propto \frac{\alpha}{N-1+\alpha} |\Lambda_{k'}|^{1/2} e^{-1/2(\omega_i - \mu_{k'})^T \Lambda_{k'} (\omega_i - \mu_{k'})},$$

where the parameters  $\mu_{k'}$  and  $\Lambda_{k'}$  are sampled from their priors,

$$\begin{aligned}\mu_k &\sim \mathcal{N}(\lambda, R^{-1}), \\ \Lambda_k &\sim \mathcal{W}(\beta, W^{-1}),\end{aligned}$$

where  $\lambda, R^{-1}, \beta$  and  $W^{-1}$  are vague/non-informative priors.

## 2. Second sample $\mu_k$ and $\Sigma_k$ :

For  $k = 1, 2, \dots, K$ , sample  $\mu_k$  and  $\Sigma_k$  from:

$$\begin{aligned}\mu_k &\sim \mathcal{N}\left((N_k \bar{\omega}_k \Lambda_k + \lambda R)(N_k \Lambda_k + R)^{-1}, (N_k \Lambda_k + R)^{-1}\right), \\ \Lambda_k &\sim \mathcal{W}\left(\beta + N_k + D - 1, \left((\beta W)^{-1} + \sum_{i=1}^N \left(\delta(z_i = k)(\omega_i - \mu_k)(\omega_i - \mu_k)^T\right)\right)^{-1}\right).\end{aligned}$$

## 3.2.3. Sampling to approximate the kernel

As we established earlier, the kernel can be represented by:

$$\kappa(\mathbf{x} - \mathbf{x}') \approx \frac{1}{M} \boldsymbol{\varphi}^T(\mathbf{x}) \boldsymbol{\varphi}(\mathbf{x}'),$$

where  $\boldsymbol{\varphi}(\mathbf{s}) = [\cos(\omega_1^T \mathbf{s}), \dots, \cos(\omega_M^T \mathbf{s}), \sin(\omega_1^T \mathbf{s}), \dots, \sin(\omega_M^T \mathbf{s})]$ , and  $\omega_i, i = 1, 2, 3, \dots, M$  is a sampled from the learned Gaussian Mixture. In order to approximate the kernel, for each random representation, we take a candidate frequency with probability  $r = \min(1, \alpha)$ , where

$$\alpha = \frac{P(\mathbf{y}|X, W_{-j}, \omega_j^*)}{P(\mathbf{y}|X, W)}$$

Now, if the task is a regression, then Equation (10) is used. With classification, Equation (11) is used. Then, we take a random number  $u \sim U(0, 1)$  and accept  $\omega_j^*$  if  $u < r$ ; otherwise reject  $\omega_j^*$ . For this, it is clear that we need to sample  $\omega_j^*$  from:

$$\omega_j^* \sim \mathcal{N}(\omega_j^* | \mu_{Z_j}, \Sigma_{Z_j})$$

In order to compute  $P(\mathbf{y}|X, \Omega)$ , it is necessary to identify what type of task is being solved, regression or classification.

## 1. In the case of a regression:

$$P(\mathbf{y}|X, \Omega) \propto \sqrt{\frac{|V_N|}{|V_0|}} \frac{b_0^{a_0}}{b_N^{a_N}} \frac{\Gamma(a_N)}{\Gamma(a_0)}, \quad (10)$$

where

$$\begin{aligned}w_N &= V_N(V_0^{-1}w_0 + \Phi(X)^T \mathbf{y}) \\ V_N &= (V_0 + \Phi(X)^T \Phi(X))^{-1} \\ a_N &= a_0 + \frac{N}{2} \\ b_N &= b_0 + \frac{1}{2}(w_0^T V_0^{-1} w_0 + \mathbf{y}^T \mathbf{y} - w_N^T V_N^{-1} w_N) \\ \Phi(X) &= (\boldsymbol{\varphi}(\mathbf{x}_1)^T, \dots, \boldsymbol{\varphi}(\mathbf{x}_N)^T)^T\end{aligned}$$



2. In the **classification** task, an approximation of the logistic regression is used,

$$p(y_i = C_1 | \mathbf{x}, X, W) \approx \text{sigm}(\mathbf{w}_N^T \boldsymbol{\varphi}(\mathbf{x})),$$

where  $\text{sigm}(a) = \frac{1}{1 + \exp(-a)}$ . Thus, the likelihood is approximated by:

$$\begin{aligned} p(\mathbf{y} | X, W) &= \prod_{i=1}^N p(y_i = C_1 | \mathbf{x}, X, W)^{y_i} (1 - p(y_i = C_1 | \mathbf{x}, X, W))^{1-y_i} \\ &\approx \prod_{i=1}^N \left( \text{sigm}(\mathbf{w}_N^T \boldsymbol{\varphi}(\mathbf{x}_i)) \right)^{y_i} \left( 1 - \text{sigm}(\mathbf{w}_N^T \boldsymbol{\varphi}(\mathbf{x}_i)) \right)^{1-y_i} \end{aligned} \quad (11)$$

where,

$$\begin{aligned} \mathbf{w}_N &= V_N (V_0^{-1} \mathbf{w}_0 + \Phi(X)^T \mathbf{y}) \\ V_N &= (V_0 + \Phi(X)^T \Phi(X))^{-1} \end{aligned}$$

3. **Computing  $\alpha$** : the following criteria is used to accept a sample  $\boldsymbol{\omega}_j^*$  with probability  $r$ :

$$r = \min(1, \alpha),$$

### 3.2.4. Learning Locally Stationary Kernels

In order to learn locally stationary kernels, we use a similar process, but we compute  $\boldsymbol{\varphi}(x)$  by using Equation (6) instead of Equation (4). Equation (6) needs the variables  $\boldsymbol{\omega}_i, i = 1, 2, \dots, M_1$  (approximating the  $\kappa_2$ ) and  $\mathbf{v}_k, k = 1, 2, \dots, M_2$  (approximation for  $\kappa_1$ ). To learn the variables in  $\kappa_2$  we use the algorithm showed; however to learn the variables to approximate  $\kappa_1$ , we approximate  $\kappa_1$  as a infinite Gaussian mixture. This means that we need to learn the variables  $Z'_j, \boldsymbol{\mu}'_k, \Sigma'_k$  and  $\mathbf{v}_k$  that approximate the function  $\kappa_1$ . Learning these variables is very similar on how we learn them from the stationary kernel with a slight modification:

1. **Sample  $Z'_j$** : Sampling  $Z'_j$  is analogous to learning the stationary kernel but with  $\mathbf{v}_k$  instead of  $\boldsymbol{\omega}_k$ .
2. **Sample  $\boldsymbol{\mu}'_k, \Sigma'_k$** : This sample is analogous to the previous section but with  $\mathbf{v}_k$  instead of  $\boldsymbol{\omega}_k$ .
3. **Sample  $\mathbf{v}_k$** : To sample  $\mathbf{v}_k$ , we sample from

$$\mathbf{v}_k^* \sim \mathcal{N}(\mathbf{v}_k^* | \boldsymbol{\mu}'_{Z'_j}, \Sigma'_{Z'_j}).$$

In order to compute  $P(\mathbf{y} | X, W, V)$ , we use  $\boldsymbol{\varphi}(x)$  as a locally stationary kernel instead of a stationary kernel. This simple change allows to add more learning capabilities to GaBaSR.

### 3.2.5. Complexity of GaBaSR

- **Complexity of sampling all  $Z_i$** : The complexity of sampling one  $Z_i$  is  $O(KMd)$ . Thus, the complexity of sampling all  $Z_i$  for  $i = 1, 2, \dots, M$  is bounded by  $O(KM^2d)$ , where  $d$  is the dimension of the input vectors,  $M$  is the number of samples to approximate the kernel and  $K$  is the number of Gaussian's found by the algorithm.
- **Complexity of sampling all  $\boldsymbol{\mu}_k$  and  $\Sigma_k$** :
  - **Complexity of computing  $\boldsymbol{\mu}_k$** : To take a sample we need to compute  $(N_k \bar{\mathbf{w}}_k \Lambda_j + \lambda R)(N_k \Lambda_k + R)^{-1}$  which takes  $O(d + d^2)$ . Also, we need to compute  $(N_k \Lambda_k + R)^{-1}$  which takes  $O(d^3)$ , so the complexity is bounded by  $O(d^3)$ .

- **Complexity of computing  $\Lambda_k$ :** To take a sample we need to compute  $(\beta W)^{-1}$  which takes  $O(d^3)$ . After that we need to compute the inverse of a matrix of  $d \times d$  which takes  $O(d^3)$ , so this step is bounded by  $O(d^3)$ .
- **Complexity of computing both  $\mu_k$  and  $\Lambda_k$**  is bounded by  $O(d^3)$ .

We need to take  $K$  samples, so sample all  $\mu_k$  and  $\Sigma_k, k = 1, 2, \dots, K$  is bounded by  $O(Kd^3)$ .

- **Complexity of  $P(\mathbf{y}|X, W_{-j}, \omega_j^*)$ :** The complexity of computing  $P(\mathbf{y}|X, W_{-j}, \omega_j^*)$  (doesn't matter if it is a regression or classification task) is bounded by  $O(NdM + M^2N + M^3)$ . Since the complexity of computing the matrix  $\Phi(X)$  is  $O(NdM)$ ; then, the complexity of computing  $V_N$  is  $O(M^2N + M^3)$ . This means that the complexity of taking  $M$  samples  $(\omega_1, \omega_2, \dots, \omega_M)$  is bounded by  $O(NdM^2 + M^3N + M^4)$ .
- **Complexity of one swap (loop) of the algorithm:** We sum the three complexities and we have:  $O(KM^2d + Kd^3 + NdM^2 + M^3N + M^4) = O(M^2d(K + N) + Kd^3 + M^3N + M^4) = O(M^3N)$  because  $N \gg M$ .
- **Complexity of  $s$  swaps (loops) of the algorithm:** If we make  $s$  swaps, then the complexity of all the GaBaSR is bounded by  $O(sM^3N)$ .

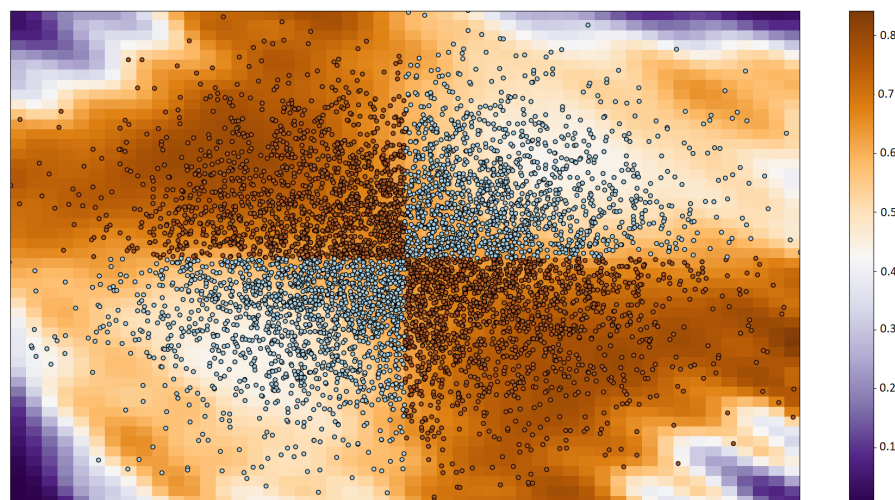
#### 4. Experiments

In this work, the experiments are performed without data cleaning i.e. no normalization or removal of outliers is done. Additionally, we use vague/non-informative priors to test the robustness of GaBaSR. We use the following variables  $a_0 = 0.001, b_0 = 0.001, V_0 = 0.000001 * I_{2M}$ , where  $I_{2M}$  is the identity matrix of dimension  $2M \times 2M$ .

Using non-informative priors together with the fact that there is no need of prepossessing the data, can be seen as one of the many the advantage of GaBaSR. Finally, the main idea of the kernel methods is to give more power to linear machines via the kernel trick. For this reason, we designed the experiments to compare GaBaSR with pure linear machines. Unfortunately, when trying to collect the original datasets by Oliva et al. [18], we found that they are no longer available online. Thus, the comparison between GaBaSR and Oliva's algorithm could not be performed.

##### 4.1. Classification

The first dataset is the XOR problem in 2D. We set the number of samples to  $N = 6000$ . After five swaps with 300 frequencies ( $M$ ), the proposal got an AUC of 0.98. The result of this experiment is shown in Figure 1.



**Figure 1.** The XOR problem with the probability of belonging to class 1 (orange). If the probability is more than 0.5, then the sample belongs to class 1 otherwise belongs to class 2.

All the results of the GaBaSR algorithm uses 500 samples ( $M$ ) and 5 swaps each one. For classification problem we use some small datasets, Breast Cancer, Credit-g, Blood Transfusion, Electricity, Egg-eye-state and Kr vs Kp. The breast cancer dataset it comes from the UCI repository [26] dataset, this dataset is the breast cancer wisconsin dataset. The Credit-g dataset comes from the UCI repository [26] and classifies people by a set of attributes as good or bad credit risk. The Electricity Dataset we downloaded from openml.org and contains data from the Australian New South Wales. Dataset egg-eye-state we downloaded from UCI, this describes if the eye is closed (1) or open (0). Kr Vs Kp dataset was downloaded from UCI, is the King Rook vs King Pawn and it's from the King+Rook's side to move and the classification is see if win or not win.

Tables 1–3 show the results when we try to solve the problem using perceptron, SVM and GaBaSR, respectively. From those tables, we can see that the in the problems of Kr vs. Kp and Electricity GaBaSR has "similar" AUC to SVM but it is important to notice that in blood transfusion GaBaSR performs better than the perceptron and SVM.

In other words, if we compare Table 3 with Table 2, it is possible to observe that the results, in general, for SVM Classification are better than for GaBaSR Classification. AUC equal to 0.5 indicates that the classifier is random, so it does not fulfil its function. Comparing Table 3 with Table 2, we can conclude that SVM Classification works properly for all tested datasets except Blood Transfusion, and GaBaSR Classification works properly only for Kr vs. Kp and Electricity. Results for Blood Transfusion obtained by GaBaSR are better than for SVM but still not satisfactory. A bad result is also a result that has its value.

In general we had a good accuracy, in most of the cases we had an accuracy above 0.8. For example in the dataset for the breast cancer, we had an accuracy of 0.89, which in general is a good accuracy. In the dataset credit-g we had 0.91 of accuracy using only 500 frequencies in 5 swaps.

**Table 1.** Results of the Perceptron.

Dataset	N	M	Swaps	AUC
Breast Cancer [26]	569	500	5	0.96480
Credit-g [26]	1000	500	5	0.44576
Blood Transfusion [27]	748	500	5	0.37572
Electricity [28]	45,312	500	5	0.68576
Egg-eye-state [26]	14,980	500	5	0.61635
Kr vs Kp [26]	3196	500	5	0.99357

**Table 2.** Results SVM Classification.

Dataset	N	M	Swaps	AUC
Breast Cancer [26]	569	500	5	0.934656
Credit-g [26]	1000	500	5	0.85282
Blood Transfusion [27]	748	500	5	0
Electricity [28]	45,312	500	5	0.76076
Egg-eye-state [26]	14,980	500	5	0.692924
Kr vs Kp [26]	3196	500	5	0.96970

**Table 3.** Results of GaBaSR Classification.

Dataset	N	M	Swaps	AUC
Breast Cancer [26]	569	500	5	0.51348
Credit-g [26]	1000	500	5	0.5142
Blood Transfusion [27]	748	500	5	0.54063
Electricity [28]	45,312	500	5	0.74991
Egg-eye-state [26]	14,980	500	5	0.519743
Kr vs Kp [26]	3196	500	5	0.9045

#### 4.2. Regression

For the regression experiments, we use a synthetic data set in our regression attempt. For this, samples are taken from the Gaussian Mixture distribution shown in Equation (12). After that,  $M = 250$ , and  $\beta \sim \mathcal{N}(0, I_{501})$  are set. In fact, the number 501 is because the extended vector is used, which takes samples from  $y_i \sim \mathcal{N}(\phi_\rho(x_i)^T \beta)$ , where  $\beta_\rho$  represents the random features from  $\rho$ , i.e.  $\omega_i \sim \rho(\omega)$ ,  $i = 1, 2, \dots, M$ . Furthermore, an instance of this problem is shown in Figure 2. Also, 250 samples and vague/non-informative prior to learn the function are used, and the result is shown in Figure 3.

$$p(\omega) = \pi_1 p\left(\omega \middle| 0, \frac{1}{2^2}\right) + \pi_2 p\left(\omega \middle| \frac{3\pi}{4}, \frac{1}{2^2}\right) + \pi_3 p\left(\omega \middle| \frac{11\pi}{8}, \frac{1}{4^2}\right) \quad (12)$$

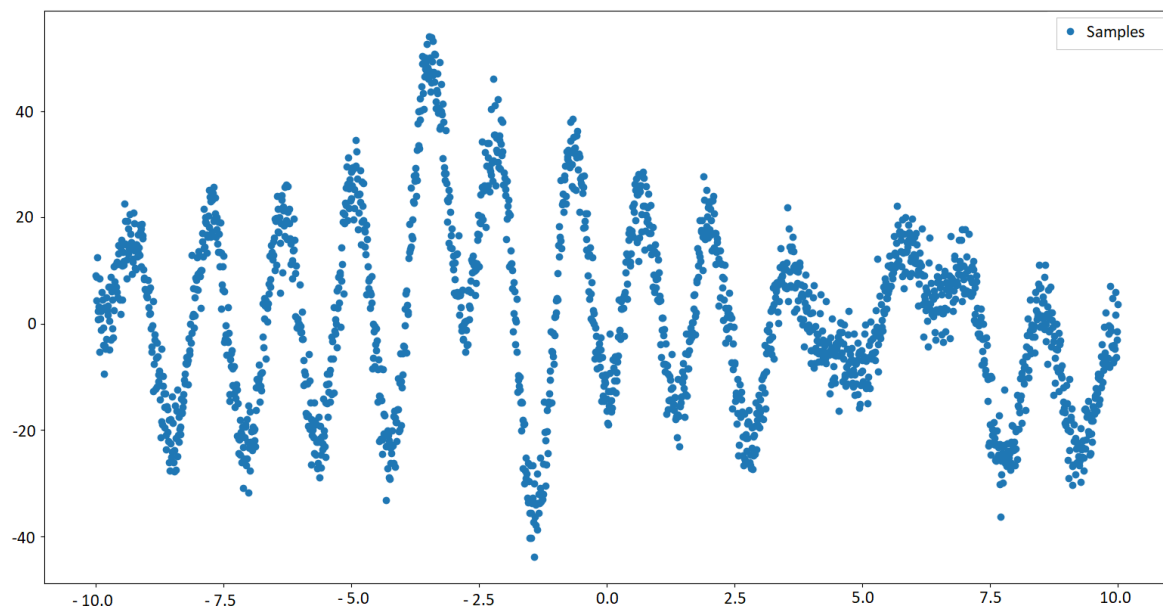
$$\text{with } \pi_1 = \pi_2 = \pi_3 = \frac{1}{3} \quad (13)$$

All the results of the GaBaSR algorithm uses 500 samples ( $M$ ) and 5 swaps each one. For regression problems we use some small datasets, Mauna LOA CO<sub>2</sub>, California Houses, Boston house-price, and Diabetes. The Mauna LOA CO<sub>2</sub> [29] from the global monitoring laboratory this collects the information of the monthly mean CO<sub>2</sub>, as we can see from Figure 4, this data it is stationary, has some repetitions and increments. The California Houses is a set of 20,640 rows with 8 columns. The Boston house price dataset was collected in 1978 from various suburbs in Boston. Diabetes Dataset has ten variables and the progression of the disease one year after.

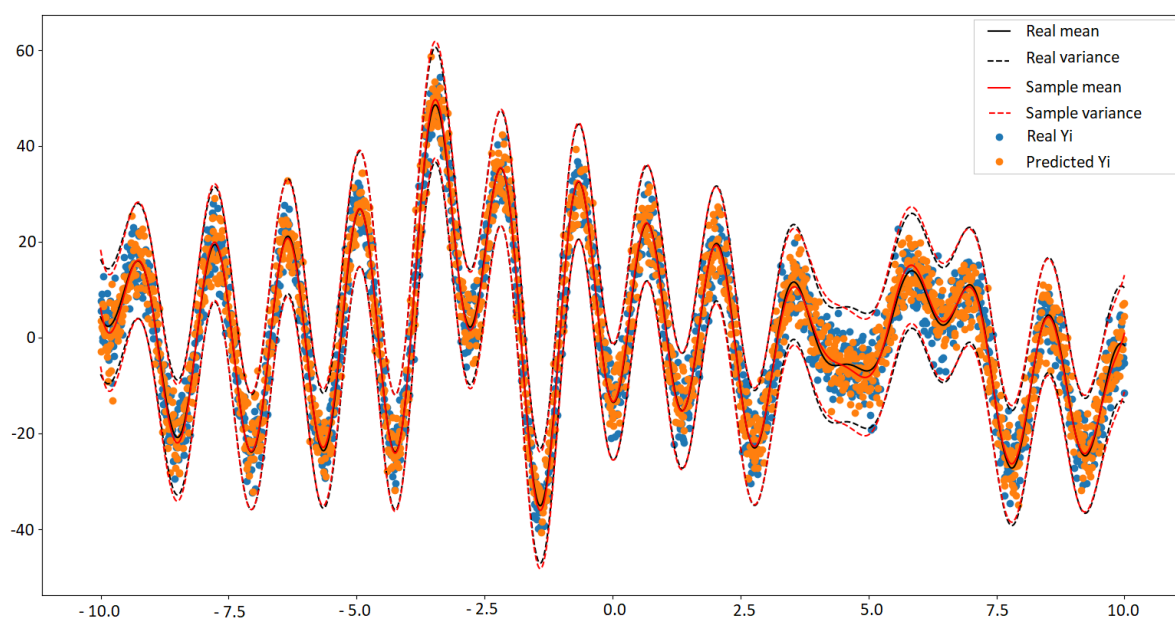
In this section we show three tables of results, Table 4 shows the results of our algorithm. Tables 5 and 6 we present the results of the linear regression and the Support Vector Machine with linear kernel, respectively.

In this subsection, the experiments are performed with simple data and the result of a simple linear regression are shown. The results are shown in Tables 4 and 5.

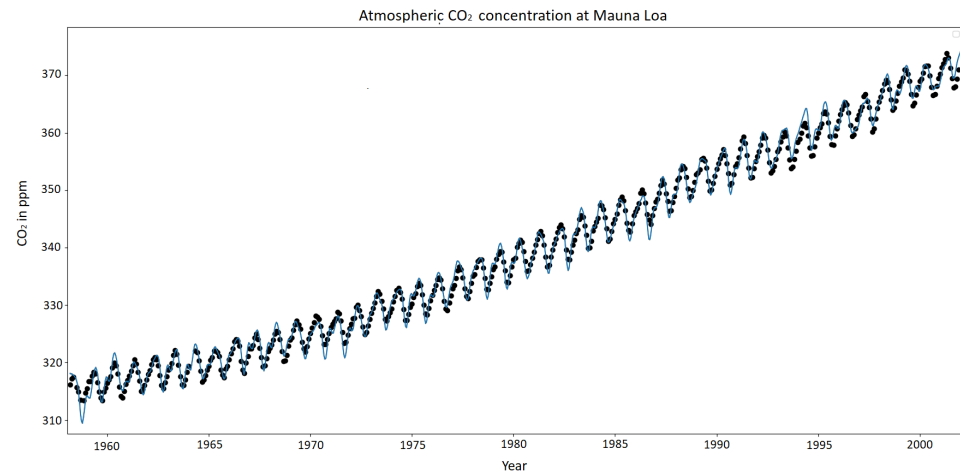
As it can be seen, an important result is the given by the Mauna LOA CO<sub>2</sub> dataset. This dataset contains data from the year 1958 to 2001. Thus, for this experiment the algorithm is trained with  $M = 250$  and performing five swaps. After the model has been trained, to learn stationary kernels, it is possible to asses the performance of the model. For example, the achieved MSE is 0.6052 which helps at the estimation of the CO<sub>2</sub> outputs. For example, at the sample 2002.13, we have prediction 376.873 where the real measure for this value is 373.08. The total results of Mauna LOA CO<sub>2</sub> are shown in Figure 4 and Table 4.



**Figure 2.** An instance of the samples taken from Equation (12).



**Figure 3.** The real vs. predicted using GaBaSR.



**Figure 4.** Mauna LOA CO<sub>2</sub> from 1958 to 2001 and the prediction.

We use the following datasets: (1) Mauna LOA CO<sub>2</sub> from [29], (2) California houses from [30], (3) Boston house-price from [30] and Diabetes from [30]

**Table 4.** Results GaBaSR Regression.

Dataset	M	Swaps	MSE	R <sup>2</sup>
Mauna LOA CO <sub>2</sub>	250	5	0.60522	0.99789
California houses	250	5	2.91313	−1.16892
Boston house-price	250	5	3.10931	0.97060
Diabetes	250	5	5918263.63330	−869.13210

**Table 5.** Results of Linear Regression.

Dataset	MSE	R <sup>2</sup>
Mauna LOA CO <sub>2</sub>	6.86	0.98
California houses	0.55	0.59
Boston house-price	18.92	0.78
Diabetes	3141.62	0.51

**Table 6.** Results Linear Regression SVM.

Dataset	MSE	R <sup>2</sup>
Mauna LOA CO <sub>2</sub>	758.08513	−1.60705
California houses	2.00777	−0.50784
Boston house-price	47.66904	0.43533
Diabetes	8094.08752	−0.36496

## 5. Conclusions

Although GaBaSR's result are promising, there are still quite a lot of work to do. For example, sampling the  $\omega_j$  is quite slow, and there is a need to update the matrix  $\Phi$  for only one sample. Oliva et al. [18] states that this can be done using low-rank updates. However, he does not present any procedure to perform such task, the low-rank updates are being consider for the next phase of GaBaSR. Thus, it is necessary to research how many samples  $M$  are required in order to obtain a low rank approximation.



In the experiments, it is possible to observe that GaBaSR is more accurate when performing a classification task rather than a regression task. This is an opportunity to improve the regression model. It means that the regression model needs more research to improve its performance or perhaps that a different model to learn the regression task is needed.

**Author Contributions:** Conceptualization, L.R.P.-L. and A.M.-V.; Methology, L.R.P.-L. and R.O.G.-M.; software L.R.P.-L. and A.G.; validation, R.O.G.-M.; formal analysis, L.R.P.-L. and A.M.-V.; resources, A.M.-V.; data curation, A.G.; writing—original draft preparation, L.R.P.-L. and R.O.G.-M.; writing—review and editing, A.M.-V. and A.G.; supervision, A.M.-V. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** All the data was cited and can be downloaded with their respective citation.

**Acknowledgments:** The authors wish to thank the The National Council for Science and Technology (CONACyT) in Mexico and Escuela Militar de Mantenimiento y Abastecimiento, Fuerza Aérea Mexicana Zapopan.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

AUC	Area Under the Curve
MSE	Mean Squared Error
SVM	Support Vector Machine
$R^2$	Coefficient of determination
MKL	Multiple Kernel Learning
SDP	Semi-Definite Programming
SMO	Sequential Minimal Optimization
BaNK	Bayesian Nonparametric Kernel
GaBaSR	Gaussian Mixture Bayesian Nonparametric Kernel Learning using Spectral Representation
GMM	Gaussian Mixture Model
MCMC	Markov Chain Monte Carlo
UCI	University of California, Irvine

## References

- Smola, A.J.; Schölkopf, B. *Learning with Kernels*; Citeseer: Princeton, NJ, USA, 1998; Volume 4.
- Soentpiet, R. *Advances in Kernel Methods: Support Vector Learning*; MIT Press: Cambridge, MA, USA, 1999.
- Anand, S.S.; Scotney, B.W.; Tan, M.G.; McClean, S.I.; Bell, D.A.; Hughes, J.G.; Magill, I.C. Designing a kernel for data mining. *IEEE Expert* **1997**, *12*, 65–74. [[CrossRef](#)]
- Schölkopf, B.; Smola, A.; Müller, K.R. Kernel principal component analysis. In Proceedings of the International Conference on Artificial Neural Networks, Lausanne, Switzerland, 8–10 October 1997; pp. 583–588.
- Zien, A.; Rätsch, G.; Mika, S.; Schölkopf, B.; Lengauer, T.; Müller, K.R. Engineering support vector machine kernels that recognize translation initiation sites. *Bioinformatics* **2000**, *16*, 799–807. [[CrossRef](#)] [[PubMed](#)]
- Tipping, M.E. The relevance vector machine. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2000; pp. 652–658.
- Junli, C.; Licheng, J. Classification mechanism of support vector machines. In Proceedings of the WCC 2000-ICSP 5th International Conference on Signal Processing Proceedings 16th World Computer Congress, Beijing, China, 21–25 August 2000; Volume 3, pp. 1556–1559.
- Bennett, K.P.; Campbell, C. Support vector machines: Hype or hallelujah? *Acm Sigkdd Explor. Newsl.* **2000**, *2*, 1–13. [[CrossRef](#)]
- Gönen, M.; Alpaydın, E. Multiple kernel learning algorithms. *J. Mach. Learn. Res.* **2011**, *12*, 2211–2268.
- Lanckriet, G.R.; Cristianini, N.; Bartlett, P.; Ghaoui, L.E.; Jordan, M.I. Learning the Kernel Matrix with Semidefinite Programming. *J. Mach. Learn. Res.* **2004**, *5*, 27–72.
- Hoi, S.C.; Jin, R.; Lyu, M.R. Learning Nonparametric Kernel Matrices from Pairwise Constraints. In Proceedings of the 24th International Conference on Machine Learning ACM, Corvallis, OR, USA, 20–24 June 2007; pp. 361–368.



12. Cvetkovic, D.M.; Doob, M.; Sachs, H. *Spectra of Graphs*; Academic Press, New York, NY, USA, 1980; Volume 10.
13. Platt, J. *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*; Microsoft: Redmond, WA, USA, 1998.
14. Bach, F.R.; Lanckriet, G.R.; Jordan, M.I. Multiple Kernel Learning, Conic Duality, and the SMO Algorithm. In Proceedings of the Twenty-First international Conference on Machine Learning ACM, Banff, AB, Canada, 4–8 July 2004; p. 6.
15. Williams, C.K.; Rasmussen, C.E. *Gaussian Processes for Machine Learning*; MIT Press: Cambridge, MA, USA, 2006.
16. Rahimi, A.; Recht, B. Random features for large-scale kernel machines. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 3–5 December 2007; pp. 1177–1184.
17. Ghiasi-Shirazi, K.; Safabakhsh, R.; Shamsi, M. Learning translation invariant kernels for classification. *J. Mach. Learn. Res.* **2010**, *11*, 1353–1390.
18. Oliva, J.B.; Dubey, A.; Wilson, A.G.; Póczos, B.; Schneider, J.; Xing, E.P. Bayesian Nonparametric Kernel-Learning. In Proceedings of the Artificial Intelligence and Statistics, Cadiz, Spain, 9–11 May 2016; pp. 1078–1086.
19. Rasmussen, C. The infinite Gaussian mixture model. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2000; pp. 554–560.
20. Genton, M. Classes of kernels for machine learning: A statistics perspective. *J. Mach. Learn. Res.* **2001**, *2*, 299–312.
21. Bochner, S. *Harmonic Analysis and the Theory of Probability*; California University Press: Berkeley, CA, USA, 1955.
22. Silverman, R. Locally stationary random processes. *IRE Trans. Inf. Theory* **1957**, *3*, 182–187. [[CrossRef](#)]
23. Hoeffding, W. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*; Springer: Berlin, Germany, 1994; pp. 409–426.
24. Gelman, A.; Carlin, J.B.; Stern, H.S.; Dunson, D.B.; Vehtari, A.; Rubin, D.B. *Bayesian Data Analysis*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2013.
25. Geman, S.; Geman, D. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.* **1984**, *6*, 721–741. [[CrossRef](#)] [[PubMed](#)]
26. Dua, D.; Graff, C. UCI Machine Learning Repository. *Open J. Stat.* **2017**, *10*.
27. Yeh, I.C.; Yang, K.J.; Ting, T.M. Knowledge discovery on RFM model using Bernoulli sequence. *Expert Syst. Appl.* **2009**, *36*, 5866–5871. [[CrossRef](#)]
28. Gama, J. Electricity Dataset. 2004. Available online: [http://www.inescporto.pt/~{j}gama/ales/ales\\_5.html](http://www.inescporto.pt/~{j}gama/ales/ales_5.html). (accessed on 6 August 2019).
29. Carbon, D. Mauna Loa CO<sub>2</sub>. 2004. Available online: [https://cdiac.ess-dive.lbl.gov/ftp/trends/CO2/sio-keel-flask/maunaloa\\_c.dat](https://cdiac.ess-dive.lbl.gov/ftp/trends/CO2/sio-keel-flask/maunaloa_c.dat). (accessed on 6 August 2019).
30. Buitinck, L.; Louppe, G.; Blondel, M.; Pedregosa, F.; Mueller, A.; Grisel, O.; Niculae, V.; Prettenhofer, P.; Gramfort, A.; Grobler, J.; et al. API design for machine learning software: Experiences from the scikit-learn project. In Proceedings of the ECML PKDD Workshop: Languages for Data Mining and Machine Learning, Prague, Czech Republic, 23–27 September 2013; pp. 108–122.

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.