Neurocomputing 462 (2021) 426–439

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Evolving Gaussian process kernels from elementary mathematical expressions for time series extrapolation

Ibai Roman [a,b,*], Roberto Santana [a], Alexander Mendiburu [a], Jose A. Lozano [a,c]

[a] Intelligent Systems Group, University of the Basque Country, UPV/EHU, Paseo Manuel de Lardizabal 1, Donostia 20018, Spain
[b] Software and System Engineering Group, Mondragon University, Loramendi, 4, Arrasate 20500, Spain
[c] Basque Center for Applied Mathematics, BCAM Alameda de Mazarredo 14, Bilbao 48009, Spain

ARTICLE INFO

Article history:
Received 18 August 2020
Revised 20 July 2021
Accepted 8 August 2021
Available online 12 August 2021
Communicated by Zidong Wang

Keywords:
Evolutionary search
Gaussian processes
Genetic programming
Kernel learning
Time series extrapolation

ABSTRACT

Choosing the best kernel is crucial in many Machine Learning applications. Gaussian Processes are a state-of-the-art technique for regression and classification that heavily relies on a kernel function. However, in the Gaussian Processes literature, kernels have usually been either ad hoc designed, selected from a predefined set, or searched for in a space of compositions of kernels which have been defined a priori. In this paper, we propose a Genetic Programming algorithm that represents a kernel function as a tree of elementary mathematical expressions. By means of this representation, a wider set of kernels can be modeled, where potentially better solutions can be found, although new challenges also arise. The proposed algorithm is able to overcome these difficulties and find kernels that accurately model the characteristics of the data. This method has been tested in several real-world time series extrapolation problems, improving the state-of-the-art results while reducing the complexity of the kernels.

1. Introduction

Gaussian Processes (GPs) [1] are one of the most used techniques in Machine Learning for regression and classification tasks. Furthermore, they have also been applied to optimization tasks under the umbrella of Bayesian optimization [2]. A GP is a collection of random variables, any finite set of which has a joint Gaussian distribution. It is completely defined by a mean function and a covariance function described in terms of a Positive Semidefinite (PSD) kernel. The assumption in a GP is that, as the similarity between two solutions increases, so does the similarity of the output function at these solutions. The kernel function encodes the particular manner in which the similarity between any two solutions is defined, which makes it a key element in any application of GPs.

While there is a repertoire of kernel functions available in the literature [1,3,4], the choice of the most appropriate kernel for a given problem is not straightforward. Moreover, kernels usually have some parameters that need to be adjusted, which hardens the kernel selection problem [5]. These parameters, often called hyperparameters, are usually tuned by maximizing a given metric (e.g., the marginal likelihood) [6].

In early applications of GPs, the kernel function used to be designed by an expert [1], or selected from a predefined set [7]. However, some recent works tackle the question of automating the choice of the kernel [8–10]. Compositional kernel search is one of the most used techniques when automating the kernel choice. In this technique, the kernel is always the combination of a limited number of a priori defined kernels, and the kernel selection is reframed as a search in the space of possible kernel compositions. The compositional kernel search approaches take advantage of some operands (e.g., sum, product, ...) that guarantee the composed kernel is PSD as long as its components are also PSD. For example, in [8,10], the kernel search is carried out by means of a greedy search procedure. A similar approach is presented in [9], although in this work the search is guided by Genetic Programming (GenProg)[1] [11]. The solutions proposed by these methods have shown their ability to capture function properties such as smoothness, trends and periodicity [8,10]. In addition, as the behavior of the base kernels and the operands is well-known, the behavior of their composition may be guessed by an expert [8]. On the contrary, these kernel search methods usually end up with very complicated kernels, including many hyperparameters, which make them expensive to optimize. Moreover, it must be noted that these approaches rely on kernels that have already been proposed in the literature.

* Corresponding author at: Software and System Engineering Group, Mondragon University, Loramendi, 4, Arrasate 20500, Spain
E-mail address: iroman@mondragon.edu (I. Roman).

[1] Note that we keep the acronym GP to refer to Gaussian Process, and GenProg to refer to Genetic Programming.

https://doi.org/10.1016/j.neucom.2021.08.020
0925-2312/

There is no reason to believe that kernels obtained by composing a limited set of human-designed kernels are optimal for arbitrary problems. Furthermore, using previously designed kernels as building blocks could bias the search and prevent the exploration of more promising candidates.

In this paper, instead of considering a reduced set of base kernels as building blocks, we propose using a set of elementary mathematical expressions (e.g., product, sum, exponent, etc.) to serve as components of a wider set of kernels. Our hypothesis is that, by enlarging the space of possible solutions, the representation capability of GPs is expanded, allowing more accurate kernels with a lower number of hyperparameters to be found.

Searching for the most appropriate solution in the space of mathematical expressions is very challenging due to the vast number of kernels that can be generated and the lack of guarantee that these kernels satisfy the PSD property. Note that, before evaluating a kernel, its hyperparameters should be optimized, which limits the number of kernels that can be explored due to the computational effort required to find these hyperparameters. We propose a novel GenProg method, EvoCov, which is able to overcome these challenges and learn adequate kernel functions for each problem. This method does not rely on previously proposed kernels, and thus, new kernels may naturally arise.

Although in this work we focus on time series extrapolation problems, our contribution can be extended to other GP applications, such as classification. Moreover, some of the components designed in EvoCov could be applied to other GenProg applications. Further work is certainly needed to test whether or not they can outperform existing kernels in these other domains.

The remainder of the paper is structured as follows: In the next section, a background on GP regression is provided, including the presentation of the best known GP kernel functions. In Section 3, we present our kernel representation approach, based on elementary mathematical expressions. In Section 4, a novel GenProg method, EvoCov, is proposed to search for GP kernel functions based on such grammar. In Section 5, a review on related work is provided. Next, in Section 6, we present an empirical validation of the algorithm and comparisons with other methods. Finally, in Section 7, the conclusions and the future work are presented.

2. Gaussian process regression

A Gaussian Process (GP) is a stochastic process, defined by a collection of random variables, any finite number of which have a joint Gaussian distribution [1]. A GP can be interpreted as a distribution over functions, and each sample of a GP is a function. GPs can be completely defined by a mean function $m(\mathbf{x})$ and a covariance function. GP models use a PSD kernel to define the covariance between any two function values $cov(f(\mathbf{x}),f(\mathbf{x}\prime)) = k(\mathbf{x},\mathbf{x}\prime)$ [3]. Given that, a GP can be expressed as follows:

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}), k(\mathbf{x},\mathbf{x}\prime)) \tag{1}$$

where we assume that $\mathbf{x} \in \mathbb{R}^d$ and $d \in \mathbb{N}$.

GPs can be used for regression by obtaining their conditional distribution given some (training) data, also known as the posterior distribution. The joint distribution between the training outputs $\mathbf{f} = (f_1, f_2, \ldots, f_n)$ (where $f_i \in \mathbb{R}, i \in \{1,\ldots,n\}$ and $n \in \mathbb{N}$) and the test outputs $\mathbf{f}_* = (f_{n+1}, f_{n+2}, \ldots, f_{n+n_*})$ is given by:

$$\begin{bmatrix} \mathbf{f}^T \\ \mathbf{f}_*^T \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} M(X) \\ M(X_*) \end{bmatrix}, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix} \right) \tag{2}$$

where $N(\mu,\Sigma)$ is a multivariate Gaussian distribution, $X = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ ($\mathbf{x}_i \in \mathbb{R}^d, i \in \{1,\ldots,n\}$ and $n \in \mathbb{N}$) corresponds to the training inputs, and $X_* = (\mathbf{x}_{n+1}, \mathbf{x}_{n+2}, \ldots, \mathbf{x}_{n+n_*})$ to the test inputs. $K(X,X_*)$ denotes the $n \times n_*$ matrix of the covariances evaluated for all the (X,X_*) pairs.

The predictive Gaussian distribution can be found by obtaining the conditional distribution given the training data and the test inputs:

$$\begin{aligned} &\mathbf{f}_*|X_*,X,\mathbf{f} \sim \mathcal{N}\left(\widehat{M}(X_*), \widehat{K}(X_*,X_*)\right) \\ &\widehat{M}(X_*) = M(X_*) + K(X_*,X)K(X,X)^{-1}\left(\mathbf{f}^T - M(X)\right) \\ &\widehat{K}(X_*,X_*) = K(X_*,X_*) - K(X_*,X)K(X,X)^{-1}K(X,X_*) \end{aligned} \tag{3}$$

As in many previous works [12,7,13], we consider an a priori equal-to-zero mean function ($m(\mathbf{x}) = 0$).

2.1. Kernel function

A Positive Semi-definite (PSD) kernel k is a symmetric function $\mathcal{S} \times \mathcal{S} \to \mathbb{R}$ on the set $\mathcal{S}$, such that, the matrix M, where $m_{ij} = k(x_i,x_j), \forall x_1,\ldots,x_n \in \mathcal{S}$ and $\forall n \in \mathbb{N}$, is a PSD matrix. A matrix is PSD if $\mathbf{u}M\mathbf{u}^T \geqslant 0$ for all real vectors $\mathbf{u} \in \mathbb{R}^n$, which is equivalent to saying that all its eigenvalues are non-negative.

2.2. Standard kernel functions

In this section, we introduce some of the best-known kernel functions. These kernels can be divided into two main families: stationary and non-stationary kernels [4].

A stationary kernel is translation invariant. Among the stationary kernels, we focus on isotropic kernels, as they are the most used kernel functions in the literature. Such kernels can be defined by the following equation:

$$\begin{aligned} &k(\mathbf{x},\mathbf{x}\prime) = \widehat{k}(r) \\ &r = \| \tfrac{\mathbf{x}}{\theta_l} - \tfrac{\mathbf{x}\prime}{\theta_l} \| \end{aligned} \tag{4}$$

where $\widehat{k}$ is a function that guarantees that the kernel is PSD and θ_l is the lengthscale hyperparameter. The lengthscale hyperparameter can be also a vector that expresses the relevance of each dimension d, as suggested in Automatic Relevance Determination (ARD) approaches [14,15].

On the contrary, in non-stationary kernels, the output of the kernel may vary with translation transformations of the input space. Within this family, the most common ones are those that depend on the dot product of the input vectors, and they are usually referred to as dot-product kernels:

$$\begin{aligned} &k(\mathbf{x},\mathbf{x}\prime) = \widehat{k}(s) \\ &s = \left(\tfrac{\mathbf{x}-\theta_s\mathbf{1}}{\theta_l}\right)\left(\tfrac{\mathbf{x}\prime-\theta_s\mathbf{1}}{\theta_l}\right)^T \end{aligned} \tag{5}$$

where θ_l is again the lengthscale hyperparameter, θ_s is the shift hyperparameter and $\mathbf{1}$ is a vector of ones.

Table 1 shows eleven standard kernels used in different applications of GPs [1,3]. One of the most popular kernel choices, the Squared Exponential (SE) kernel (also known as Radial Basis Function (RBF) or Exponentiated Quadratic) is described as k_{SE} in the table. This kernel is known to capture the smoothness property of the data.

2.3. Model selection

The choice of the kernel function and its hyperparameters has a critical influence on the behavior of the model, and it is crucial to achieve good results in any application of GPs. This selection has usually been made by choosing one kernel a priori, and then adjusting the hyperparameters of the kernel function in order to optimize a given metric for the data.

Table 1
Standard kernel functions. θ_0 and θ_p are the kernel hyperparameters, called amplitude and period respectively. δ is the Kronecker delta, while r and s are described in Eqs. (4) and (5) respectively.

Kernel function expressions	
Constant	$k_{CON}(\mathbf{x},\mathbf{x}\prime) = \theta_0$
White Noise	$k_{WN}(\mathbf{x},\mathbf{x}\prime) = \theta_0\ \delta(\mathbf{x},\mathbf{x}\prime)$
Exponential	$k_E(r) = \theta_0^2\ exp(-r)$
γ exponential	$k_{E^\gamma}(r) = \theta_0^2\ exp(-r^\gamma)$
Squared Exponential	$k_{SE}(r) = \theta_0^2\ exp(-\frac{1}{2}r^2)$
Matern12	$k_{M12}(r) = \theta_0^2 exp(-r)$
Matern32	$k_{M32}(r) = \theta_0^2\left(1+\sqrt{3}r\right)exp\left(-\sqrt{3}r\right)$
Matern52	$k_{M52}(r) = \theta_0^2\left(1+\sqrt{5}r+\frac{5}{3}r^2\right)exp\left(-\sqrt{5}r\right)$
Rational Quadratic	$k_{RQ}(r) = \theta_0^2\left(1+\frac{1}{2\alpha}r^2\right)^{-\alpha}$
Periodic	$k_{PER}(r) = \theta_0^2 \exp\left(-\frac{2\sin^2(\pi r)}{\theta_p^2}\right)$
Linear	$k_{LIN}(s) = s$

Although a variety of methods have been also proposed to optimize the hyperparameters [16–19], the most common approach is to find the hyperparameter set that maximizes the Log Marginal Likelihood (LML):

$$\begin{aligned} &log\, p(\mathbf{f}|X,\boldsymbol{\theta}) = -\tfrac{1}{2}\mathbf{m}_a K_a^{-1}\mathbf{m}_a^T - \tfrac{1}{2}log\,|K_a| - \tfrac{n}{2}\,log\,2\pi \\ &with \\ &\mathbf{m}_a^T = \mathbf{f}^T - M(X) \\ &K_a = K(X,X) \end{aligned} \tag{6}$$

where $\boldsymbol{\theta}$ is the set of hyperparameters of the kernel and n is the length of X.

Alternatively, a Leave-one-out Cross Validation (LOOCV) metric was proposed by [1]. In this case, the likelihood of each sample of the training data is measured given the rest of the data. Then, these probabilities are added as follows:

$$\begin{aligned} &L_{LOOCV}(X,\mathbf{f},\boldsymbol{\theta}) = \sum_{i=1}^{n} log\, p(f_i|X,\mathbf{f}_{-i},\boldsymbol{\theta}) \\ &log\, p(f_i|X,\mathbf{f}_{-i},\boldsymbol{\theta}) = -\frac{(f_i-\mu_i)^2}{2\sigma_i^2} - \tfrac{1}{2}log\,\sigma_i^2 - \tfrac{1}{2}\,log\,2\pi \end{aligned} \tag{7}$$

where μ_i and σ_i are the posterior mean and variance for $\mathbf{x}_i$ given X_{-i} and $\mathbf{f}_{-i}$ [2].

The selection of the right set of hyperparameters is known to be a hard problem, particularly when few observations are available [13,20,21]. Although in most cases the gradient of the LML and the LOOCV has a closed-form expression, depending on the problem, these functions can be multi-modal and a greedy search procedure may lead to suboptimal results.

2.4. Kernel composition

When creating new kernels, it is usually difficult to prove whether they are PSD or not. However, kernels that are PSD can be combined to create new ones keeping the positive semi-definiteness of its components [3,22] by means of the following operations:

- Sum: $k(\mathbf{x},\mathbf{x}\prime) = k_1(\mathbf{x},\mathbf{x}\prime) + k_2(\mathbf{x},\mathbf{x}\prime)$.
- Product: $k(\mathbf{x},\mathbf{x}\prime) = k_1(\mathbf{x},\mathbf{x}\prime) \times k_2(\mathbf{x},\mathbf{x}\prime)$.
- Polynomial: $k(\mathbf{x},\mathbf{x}\prime) = p(k_1(\mathbf{x},\mathbf{x}\prime))$, where p is a polynomial function with non-negative coefficients.
- Exponential: $k(\mathbf{x},\mathbf{x}\prime) = exp(k_1(\mathbf{x},\mathbf{x}\prime))$.
- Composition with a function: $k(\mathbf{x},\mathbf{x}\prime) = f(\mathbf{x})k_1(\mathbf{x},\mathbf{x}\prime)f(\mathbf{x}\prime)$, with $f : \mathbb{R}^d \to \mathbb{R}$.
- Mapping: $k(\mathbf{x},\mathbf{x}\prime) = k_1(\Psi(\mathbf{x}),\Psi(\mathbf{x}\prime))$, with $\Psi : \mathbb{R}^d \to \mathbb{R}^d$.

3. Gaussian Process kernel representation as elementary mathematical expression trees

While previous approaches have proposed the composition of kernel functions, in this work we break down the standard kernels of Table 1 into basic mathematical expressions, in order to use them as the building blocks for new kernels. Thus, these standard kernel functions and their sum/product compositions are a subset of our search space.

The kernel functions are described as an expression tree, composed by the set of the terminals and non-terminals shown in Table 2. The expression tree is strongly-typed [23], as the output of each node matches the input type of its ancestor.

A *scalar* non-terminal is used as the start symbol of the grammar. The $+$, $\times$, and $\hat{}$ arithmetic operators, with their usual meanings (addition, product and power, respectively), can be used to combine *scalar* non-terminals. Also note that the exponent in the power operator is a hyperparameter (*hp*) terminal. On the other hand, in order to allow division operations over *scalar*s, we introduce the negative power. The *scalar* elements can be also obtained by means of the square distance or dot product of two vectors ($\mathbf{vec}, \mathbf{vec}\prime$), as described in Section 2.2. Similarly, the constant non-terminal is included, which value depends on a hyperparameter terminal. Pairs of vectors ($\mathbf{vec}, \mathbf{vec}\prime$) can be obtained directly using the input vectors of the kernel ($\mathbf{x}, \mathbf{x}\prime$) or applying the spectral transformation to them, in order to allow periodic kernels as suggested by [24]. Finally, among hyperparameter terminals we can find the input parameters of the kernel (θ) or the constant values indicated in Table 2.

4. Evolving kernel functions based on the new grammar

Once the grammar has been introduced, we present our GenProg [11] approach for GP kernel search, EvoCov. GenProg is an

Table 2
Proposed grammar for GPs. t indicates the number of different hyperparameters allowed in the grammar (in this work it is set to $t = 20$).

kernel:	*scalar*	start symbol
scalar:		
	$scalar^{hp}$	power
	$scalar + scalar$	add
	$scalar \times scalar$	multiply
	$scalar^{-1}$	div
	e^{scalar}	exp
	$\sqrt{scalar}$	sqrt
	$scalar^2$	square
	$\|\frac{\mathbf{vec}}{hp} - \frac{\mathbf{vec}\prime}{hp}\|^2$	sq_distance
	$\left(\frac{\mathbf{vec}-\mathbf{1}hp_1}{hp_0}\right).\left(\frac{\mathbf{vec}\prime-\mathbf{1}hp_1}{hp_0}\right)^T$	dot_product
	hp	constant
	;	
($\mathbf{vec}, \mathbf{vec}\prime$):		
	$\left(\left[sin\left(\frac{2\pi\mathbf{x}}{hp}\right)\ \ cos\left(\frac{2\pi\mathbf{x}}{hp}\right)\right],$ $\left[sin\left(\frac{2\pi\mathbf{x}\prime}{hp}\right)\ \ cos\left(\frac{2\pi\mathbf{x}\prime}{hp}\right)\right]\right)$	spectral
	$(\mathbf{x}, \mathbf{x}\prime)$	input
	;	
hp:		
	$\mid -1 \mid -0.5 \mid 1 \mid 2 \mid 3 \mid 5$ $\mid \theta_0 \mid \theta_1 \mid \ldots \mid \theta_t$	
	;	

[2] X_{-i} notation is used to indicate that $\mathbf{x}_i$ was removed from X.

Evolutionary Algorithm (EA) that allows to search in a set of possible computer programs by encoding each program as a gene. In our case, the computer programs are the kernel functions.

EvoCov, shown in Algorithm 1, takes into account two challenges related to this problem: The cost of evaluating the fitness function (mainly due to hyperparameter optimization) and the fact that many of the kernels generated during the search are not PSD.

Algorithm 1: EvoCov algorithm

```
1: procedure EvoCov(N, G, S, p_m, p_cx, β, d_min, d_max)
2:   pop = GENRANDPOP(N, d_min, d_max)
3:   bestfit_{-1} = ∞
4:   all = pop
5:   i = 0
6:   while i < G − 1 do
7:     EVALUATE(pop)
8:     best = SELECT(pop, 1)
9:     bestfit_i = GETFITNESS(best)
10:    relimprov = (bestfit_{i−1} − bestfit_i) / |bestfit_i|
11:    if β < relimprov then
12:      sel = SELECT(pop, S)
13:      offspring = VARIATE(sel, N − S, p_m, p_cx)
14:    else                          ▷ Restart procedure
15:      sel = ∅
16:      offspring = GENRANDPOP(N, d_min, d_max)
17:      bestfit_i = ∞
18:    end if
19:    pop = sel ∪ offspring
20:    all = all ∪ offspring
21:    i = i + 1
22:  end while
23:  EVALUATE(pop)
24:  best = SELECT(all, 1)
25:  return best
26: end procedure
```

First, an initial population of N kernels is generated. In order to do so, each individual is created at random, limited by a maximum (d_{max}) and a minimum (d_{min}) depth. At each generation, the whole population is evaluated. Next, the relative improvement (*relimprov*) is calculated, which measures the improvement from the best fitness value of the previous generation to the best fitness value in the current generation. If the relative improvement in the current population is greater than a threshold β, a new population is generated through selection and variation. After selecting the S best individuals, a mutation or a crossover operator is randomly applied with probability p_m and p_{cx} respectively (where $p_{cx} = 1 - p_m$) to generate the offspring population. Since, due to the hyperparameter inheritance (as explained in Section 4.3.2), we want to re-evaluate the selected individuals and evaluate N kernels at each generation, the offspring population consists of $N - S$ new individuals. When the relative improvement is lower than or equal to the threshold, the current population is replaced by a randomly generated one. This procedure is repeated for $G - 1$ generations. Finally, the last population is evaluated and the best individual found during the whole process is returned.

In this section we describe each of the methods used by Algorithm 1. First, we address the issue of randomly generating new kernels for the initial population. The distinguished characteristic of our proposal for generating the random kernels is that it does not take into account any kernel proposed in the literature, while guaranteeing a minimum depth and a maximum depth. Then, we provide the variation operators conceived to generate kernels that are likely to inherit useful properties from the selected ones. A method to control the depth of the trees created by the variation operators is also introduced. Next, we explain how the GP kernels are evaluated. Finally, for comparison proposes, we include two simpler methods: Random Search and Go With The First.

4.1. Initial population

We generate kernel functions at random and discard the non-PSD ones until the desired population size is reached.

4.1.1. Random kernel generation

In order to randomly generate kernel expressions, we propose a strongly-typed grow method based on the work presented by [11]. This approach creates kernels from scratch, without any knowledge of previously proposed kernels. This is achieved by a recursive process where, at each step, a random terminal or a random operator is added.

When generating random solutions, some of the solutions may be too complex in terms of the number of terms in the expression, and others too simple or trivial. Thus, we propose a method to control the depth of the generated expressions by setting a minimum (d_{min}) and a maximum depth (d_{max}). As can be seen in Table 2, some of the non-terminals have the same symbol on both sides of the production rule. These non-terminals guarantee that, once selected, the iterative procedure can continue growing this branch, i.e., they are recursive. During the creation process, we select a uniformly random production rule depending on the current symbol. If the minimum depth has not been reached, only recursive non-terminals are used. Then, until the maximum depth is reached, any non-terminal can be selected. Finally, when the maximum depth is reached, only the terminals and the non-recursive non-terminals are used, limiting the depth of the expression.

4.1.2. Dealing with non-PSD kernels

In order to mitigate the evaluation of non-PSD kernels, we have applied to GPs the approach used by [25] in the field of SVMs: check the positive definiteness of the matrix generated by a kernel for some random data and attempt the generation of the kernel again if this matrix is not PSD.

As mentioned in Section 2.1, any matrix generated by a PSD kernel has to be symmetric and also PSD. To identify non-PSD kernels, we generate w random uniformly distributed datasets $X = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ (where $\mathbf{x}_i \in \mathbb{R}^d, i \in \{1, \ldots, n\}$ and $n \in \mathbb{N}$) and check the M matrix produced by the kernel for each dataset. If any M matrix does not match any of the following conditions, the generation process of the kernel is repeated:

- $M = M^T$: As previously mentioned, the matrix given by a PSD kernel should be symmetric.
- None of the elements in the main diagonal (m_{ii}) is negative: It has been proved [26] that, if any of the elements in the main diagonal are negative, the matrix is not PSD.
- None of the eigenvalues of M is negative: Similarly, all the eigenvalues of the matrix should be non-negative.

Compliance with all these conditions is necessary but not sufficient for a kernel to be PSD, and if the kernel is not PSD, the covariance matrix can not be inverted for GP inference. Thus, if after passing the PSD check, we find out that the kernel is not PSD during the evaluation step, the fitness value of the kernel is penalized. Fortunately, this validity check is severe enough to avoid most of the false positives. Among the kernels that were generated and validated during the preliminary experiments, only 0.67% were not PSD.

4.2. Variation operators for kernel generation

Our kernel search method is based on perturbation or variation methods that modify previous solutions to obtain new ones. We use two variation operators which are randomly selected at every VARIATE function call in Algorithm 1: A crossover operator, which combines two kernel functions to generate a new one that inherits some of the features of its parents, and a mutation operator, which introduces slight modifications to the original kernel to obtain a new individual. We also explain how the algorithm controls the depth of the trees generated by these variation methods.

4.2.1. Crossover

A purely random crossover operator hardly ever produces PSD kernels. Since kernel function evaluation is a computationally costly process, we would like to avoid non-PSD kernels. As explained in Section 2.4, the product or the sum of two PSD kernels is also PSD. Hence, a crossover method could just combine two PSD kernels with any of these operators to generate a new PSD kernel.

However, this procedure rapidly increases the depth of the expressions. Therefore, we propose a crossover operator that randomly selects a sub-expression from each kernel and combines them with the sum or the product operator. As this method does not guarantee that the resulting kernel is PSD, this operation must be repeated if a non-PSD kernel is found. Nevertheless, the method increases the chance of obtaining a PSD kernel, since, if both of the sub-expressions are PSD, the result is guaranteed to be also PSD.

4.2.2. Mutation

Based on [27], the algorithm applies a mutation operator that randomly selects one of the following methods in a type-safe manner:

Insert: Inserts a randomly chosen recursive non-terminal (see Section 4.1.1) at a random position in the kernel expression, as long as its output and the output of the one at the selected position agree. The sub-expression that was at the chosen position is used as an argument of the newly created non-terminal. If this non-terminal requires more arguments, a new sub-expression is randomly generated using the random generation method described in Section 4.1.1.

Shrink: This method shrinks the kernel expression by randomly choosing a non-terminal and replacing it with one of its arguments (also randomly chosen) of the same type.

Uniform: Selects a position uniformly at random in the kernel expression and replaces the sub-expression at that point by a randomly generated one. Note that the output type of the new sub-expression must match the output type of the replaced one.

Replacement: Replaces a randomly chosen non-terminal from the kernel expression by another non-terminal with the same number of arguments and types, also randomly chosen.

As these methods do not guarantee that the generated kernels are PSD, mutations are repeated if a non-PSD kernel is detected (see Section 4.1.2).

4.2.3. Bloat control

None of the variation methods described above limits the depth of the kernel expression. Depending on the operators, the depth of the expressions may increase without any limit during the search, making the resulting kernel functions highly complex and useless for practical applications. This is a well-known problem in GenProg literature, known as bloating [11]. In our work, when the depth of a kernel expression becomes larger than o_{max}, we discard the expression. In this case, the mutation or the crossover method is repeated until a kernel with the desired depth is obtained, or a limit of ρ_{max} trials is reached. If this number of trials is exceeded, one of the parent kernels is returned unchanged.

4.3. Evaluation

In our approach, in contrast to other GenProg applications, the solutions do not encode all the necessary information to be evaluated. In order to evaluate each kernel, we need to set the value of the hyperparameters. Thus, the fitness of the solutions depends on the results of the hyperparameter optimization. Both search procedures, the selection of the best hyperparameters for each kernel and the selection of the best kernel given these hyperparameters, are illustrated in Fig. 1.

Following the work done in [10], we use the Bayesian Information Criterion (BIC) [28] as a quality metric for each kernel. BIC is a metric for model selection which adds a regularization term to the LML to penalize the complexity of the kernels. This metric serves as the fitness function of our GenProg algorithm and it can be expressed as follows:

$$BIC(k_i) = 2\ log\ p(\mathbf{f}|X, k_i, \boldsymbol{\theta}_{i,best}) - q\ log\ n \tag{8}$$

where q is the number of hyperparameters of the kernel and n is the number of data points in X. $\boldsymbol{\theta}_{i,best}$ is the best hyperparameter set for the kernel k_i according to a given metric.

Before computing the BIC associated to a given kernel, the hyperparameters have to be optimized. As we have discussed in Section 2.3, several metrics (LML, LOOCV, ...) can be used to measure the quality of each hyperparameter set. Thus, we find the best hyperparameter set for kernel i as follows:

$$\boldsymbol{\theta}_{i,best} = argmax_j\ \mathrm{METRIC}(\mathbf{f}, X, k_i, \boldsymbol{\theta}_{i,j}) \tag{9}$$

4.3.1. Hyperparameter optimization algorithm

The hyperparameters are optimized by means of *Powell*'s local search algorithm [29]. As this algorithm does not deal with boundaries, the search space has to be constrained by penalizing non-feasible hyperparameter sets. Moreover, as the function to optimize might be multi-modal, a multi-start approach was used, performing a restart every time the stopping criteria of *Powell*'s algorithm are met, and getting the best overall result. Note that, as a result of the inclusion of the randomized restarts, the hyperparameters found for a certain kernel in two independent evaluations may not be the same. In fact, this implies that the fitness function optimized by the GenProg algorithm, i.e., BIC, is stochastic.

4.3.2. Random restarts and hyperparameter inheritance

The initial solutions for the restarts of the hyperparameter optimization algorithm are sampled from two different distributions depending on the origin of the kernel. In the randomly generated kernels, the initial hyperparameters for these restarts are sampled from a uniform distribution within the search bounds. On the other hand, if a kernel is generated through any of the variation methods, we take advantage of the information gathered in previous hyperparameter optimization procedures by adapting the inheritance technique described by [10,8] to the particularities of GenProg. Instead of restarting the multi-start optimization from a uniform distribution, each restart is sampled from a Gaussian distribution centered on the hyperparameter values of the parent individuals and with a pre-defined variance (σ_0). This inheritance method is particularly useful when the variation performs few changes to the expression.

Note that, in Algorithm 1, the selected individuals are kept for the next population and the whole population is evaluated at each generation. Thus, some individuals may be evaluated several times

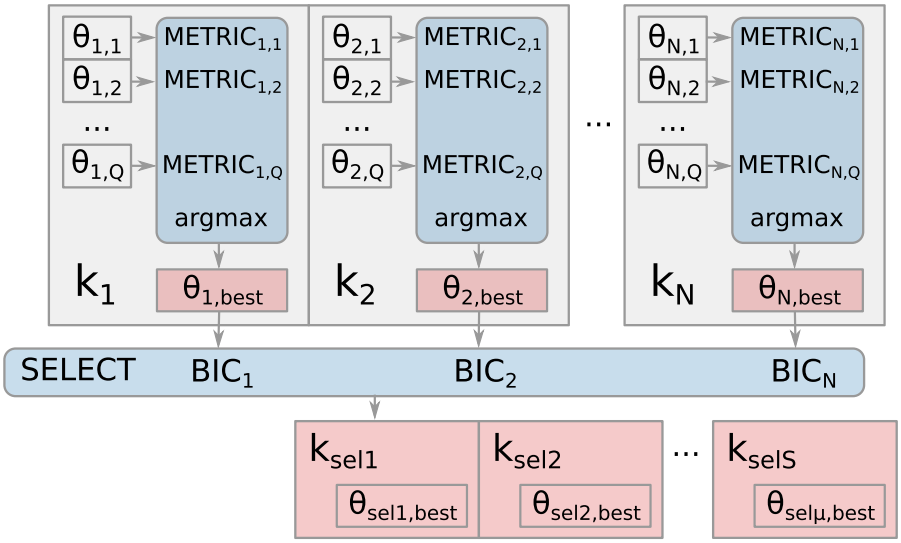


Fig. 1. Two nested search procedures: The selection of the best hyperparameters for each kernel is made according to a given METRIC, such as LML or LOOCV, and the selection of the best kernels according to the BIC.

during the search. This procedure, along with the hyperparameter inheritance, allows the selected individuals to keep optimizing their hyperparameters across generations, and compete fairly with the individuals in the offspring population, which inherit the hyperparameters.

4.4. Selection

We perform a search in the kernel function space to find the kernel that maximizes the BIC. Thus, the selection operator shown in Algorithm 1 selects the S best kernels according to the BIC metric by applying truncation selection.

5. Related work

In this section, we review the work carried out in the literature related to this paper. First, we discuss works that design ad hoc kernels based on expert knowledge. Subsequently, we review the works which propose an automatic design of kernels.

In ad hoc kernel approaches [1,30], the authors assume that the choice of the kernel function is clear from a priori knowledge about the problem. Then, the hyperparameters are optimized to adjust each kernel. In [1], an ad hoc kernel is introduced to fit the Mauna Loa Atmospheric CO_2 time series, which is a well-known problem in the GP literature due to its several periodic patterns. On the other hand, in [31], the authors propose a product of a Squared Exponential and a periodic kernel to construct a control signal. Similarly, the authors of [30] designed an ad hoc kernel to predict the number of occurrences of certain hashtags in Twitter, given the past records. Finally, the authors of [32] took advantage of the Bochner Theorem [33] to design kernels that were able to model the periodical patterns of time series. Regarding the hyperparameter optimization, Deep Learning methods have also been applied to pre-train and fine-tune the hyperparameters of the covariance functions [34].

Regarding the automatic design of kernels, many works have followed the kernel composition approach by using the properties shown in Section 2.4 to generate new kernels [9,10]. Authors of [9] propose a GenProg method for compositional kernel search, using the standard kernels shown in Table 1 as building blocks. They also consider the sum, product and scale as primitives, along with a dimension mask. The hyperparameters where not included in the grammar, as only the hyperparameters present in the standard kernels are considered. The experimentation of this work was limited to the Mauna Loa Atmospheric CO_2 time series and some synthetic 2-dimensional datasets. In addition to GenProg, other search methods have been proposed to search for kernel composition structures in GPs, such as the greedy search procedure proposed in [10]. In that work, the best kernel function in terms of BIC is searched in the space of possible compositions (sums and products) of simpler kernels. In [8], the authors improve the previous approach by adding change-point and change-window kernels. Similarly, multiple kernel learning (MKL) approaches have tried to create new kernels by adding the standard ones [35].

The idea of using elementary mathematical expressions as building blocks of kernel functions has been applied to other fields, such as Support Vector Machines (SVMs) [36] and Relevance Vector Machines (RVMs) [37]. Some of these approaches [37–39] do not guarantee that kernels are PSD. In SVM and RVM, although kernels theoretically have to be PSD in order to do the kernel-trick, in practice, many kernels can be used even if they are not of this kind. Some other approaches, such as [40,41], guarantee that the kernels are PSD by means of the kernel composition properties shown in Section 2.4, similar to the compositional kernel search methods in the GP literature. Finally, in [25,36,42] the non-PSD kernels are penalized or discarded as in our approach. The authors of [36,42] propose a method to penalize (giving the worst possible fitness) the non-PSD kernels in evaluation time. On the other hand, in [25], if during the random generation a non-PSD kernel is found, it is discarded and the creation is retried. However, as stated by the authors, their approach was not able to improve the results of the standard kernels in SVM. The above mentioned approaches deal with hyperparameters by means of optimizing small grids or adding random constants to the GenProg grammar. To the best of our knowledge, more complex techniques such as the hyperparameter inheritance have not been applied in this context.

6. Experiments

In this section, we describe the experiments we carried out to analyze the performance of our proposal. We solve extrapolation problems from real-world time series and compare our proposal to the main methods discussed in Section 5 (compositional kernel

methods and ad hoc kernel approaches) in such tasks. The goal of our experiments is threefold:

- To compare EvoCov to state-of-the-art methods that rely on kernel composition.
- To compare our proposal to the ad hoc kernels proposed in the literature.
- To study the influence of the metric used to optimize the hyperparameters in time series extrapolation problems.

First, an introduction to the time series extrapolation problem is given, before describing the experimental setup. Then, three experiments are shown, one for each objective of the experimentation.

6.1. Time series extrapolation problems

Although time series extrapolation can be implemented in different ways, in our problem, data points are associated with a time index variable and the extrapolation problem consists in producing a prediction for each time index for a time interval in the future. While properties like the smoothness of the data have been extensively studied in GP literature for interpolation problems, extrapolation problems require additional data patterns to be exploited, such as periodicities and trends, which have not been studied to the same extent.

Real-world time series extrapolation problems have been considered for the evaluation of our methods, being more realistic than synthetic time series benchmarks. These problems are characterized by a limited amount of generally noisy data, with strong variations from the training set to the test set due to the temporal bias between both sets.

Following the work done in [8], in the first two experiments, we use the real-world time series described in Table 3[3]. We trained all the algorithms on the first 90% of the data, and predicted the remaining 10%.

6.2. Alternative search methods

In order to verify that every component of the GenProg algorithm introduced in the previous section is providing benefits to the kernel search, we include two algorithms to be used as a baseline in the experiments. In order to test the joint contribution of the mutation and crossover operators, a Random Search algorithm [43] is proposed, which only uses the same random generation and selection methods as EvoCov. Also, in order to measure the gain produced by the crossover operator, we propose an algorithm which does not depend on this operator, the GoWithTheFirst algorithm, inspired by the "go with the winner" methods [44].

6.2.1. Random Search algorithm

This Random Search method generates a random population by iteratively following the random generation method described in Section 4.1.1 until the desired population size is achieved (N). Next, it chooses the best solution according to the selection criterion described in Section 4.4.

6.2.2. GoWithTheFirst algorithm

The GoWithTheFirst algorithm, (i) generates an initial population of size N, (ii) applies a hill-climbing procedure for H evaluations for each individual, by generating a random mutation, and keeping the best solution between the original and the mutated one, and (iii) discards the worst individual among this optimized population, according to the selection criteria.

Table 3
Description of the time series used in the experiments. The visually identifiable periodic patterns are described with the letter P, and if many periods are present, P + is shown. The ascendant trends are represented by AT and exponential growths by EXP. N denotes the presence of noise, while C indicates a trend change at some point in the time series.

Name	Size	Properties
Airline	144	P, AT
Solar	402	P, C
Mauna Loa Atmospheric CO_2	545	P, AT
Beveridge Wheat Price Index	370	P, C
Daily minimum temperatures in Melbourne	1000	P N
Internet traffic data (bits)	1000	P+
Monthly average daily calls	180	N, C
Monthly critical radio frequencies	240	P+
Monthly production of gas in Australia	476	P, AT
Monthly prod. of sulphuric acid in Australia	462	N
Monthly U.S. male unemployment	408	P, AT
Number of daily births in Quebec	1000	N
Real daily wages in England (£)	735	EXP

The steps (ii) and (iii) are repeated until only one kernel is left.

6.3. Experimental setup

For the GP regression, a noisy approach was used, by adding a GP with a white noise kernel to the model, and including its noise hyperparameter to the hyperparameter optimization process. For the random generation method, $d_{min} = 5$ and $d_{max} = 15$ were used to limit the size of each expression tree. In order to discard the non-PSD kernels, the positive semi-definiteness conditions described in Section 4.1.2 were checked in $w = 20$ random datasets. Besides, to avoid bloating, a maximum depth of $o_{max} = 40$ was allowed, and the number of attempts was limited to $\rho_{max} = 250$ in each variation operator.

In the Random Search algorithm, $N = 20000$ was set to generate the initial population. Similarly, in the GoWithTheFirst algorithm, $N = 13$ initial individuals and $H = 200$ local search evaluations were used in order to have a comparable evaluation budget. Finally, after some preliminary experiments with the EvoCov approach in the Mauna Loa Atmospheric CO_2 dataset, we set the parameters to the following values: $N = 141, G = 141$, $S = 14, p_m = 0.4, p_{cx} = 0.6$ and $\beta = 1e-5$. These parameters are fixed for all the datasets and experiments.

The proposed algorithms were coded in Python, based on the EA software *DEAP*[4] [27], and made publicly available in the Python Package Index[5]. Due to the stochastic nature of our algorithms, each kernel search process was repeated 10 times in all the experiments.

Regarding the hyperparameter optimization, in every restart of Powell's optimization algorithm [29], a Gaussian noise with $\sigma_\theta = 0.1$ was added to the inherited hyperparameters. Since the computational time required to evaluate the hyperparameters increases significantly with the size of the time series, and due to computational constraints, we decided to adjust the number of evaluations allowed in the hyperparameter optimization depending on the length of the time series, in order to establish a similar run time for each data set. Thus, we allow $Q = ref\_fun\_call \times \frac{ref\_ts\_len^2}{current\_ts\_len^2}$ evaluations. *ref_fun_call* and *ref_ts_len* are parameters of the algorithm describing the desired number of evaluations (*ref_fun_call*) for a reference time series size (*ref_ts_len*), and *current_ts_len* is the length of the current time series. In the experiments, the reference time series size was set to *ref_ts_len* = 350, for which *ref_fun_call* = 300 evaluations were allowed.

[3] The time series data can be found at https://pkg.yangzhuoranyang.com/tsdl/

[4] https://deap.readthedocs.io

[5] https://pypi.org/project/evocov/

6.4. Metric comparison for hyperparameter optimization

In GP literature, hyperparameter optimization is considered a crucial task. Most of the works carried out in this field rely on the LML for hyperparameter optimization [9,10]. However, it has been reported that the LML may lead to suboptimal results under certain conditions, where LOOCV could be more robust[45]. Regarding the kernel optimization, having a consistent method to optimize the hyperparameters helps to obtain more reliable evaluations of the individuals. Hence, before evaluating the differences among the kernel search algorithms introduced in the previous sections, we decided to perform an experiment to test if other alternative metrics to LML and LOOCV can improve the results in time series extrapolation problems.

Apart from the well-known LML and LOOCV, we tested other metrics specifically designed to optimize the hyperparameters in extrapolation problems. While LML measures the probability of the training data given the prior GP model, the goal in extrapolation is to increase the probability of the test data given the posterior GP model. Therefore, along with the prior LML, we also measure the posterior LML, by splitting the training set at a given point in time into the training-training and training-test sets. Thus, the probability of the training-test set given a GP model conditioned to the training-training set, i.e., the posterior LML, can be measured as follows:

$$\begin{aligned} &log\, p(\mathbf{f}|X,X_*,\boldsymbol{\theta}) = -\tfrac{1}{2}\widehat{\mathbf{m}}_a\widehat{K}_a^{-1}\widehat{\mathbf{m}}_a^T - \tfrac{1}{2}log\,|\widehat{K}_a| - \tfrac{n}{2}\,log\,2\pi \\ &with \\ &\widehat{\mathbf{m}}_a^T = \mathbf{f}^T - \widehat{M}(X_*) \\ &\widehat{K}_a = \widehat{K}(X_*,X_*) \end{aligned} \tag{10}$$

where $\widehat{M}(X_*)$ and $\widehat{K}(X_*,X_*)$ can be obtained as in Eq. (3).

Furthermore, we also use the Negative Log Predictive Densities (NLPD) [46] as a extrapolation oriented version of the LOOCV. NLPD adds the likelihood of each prediction in the training-test set, given some hyperparameters and the training-training data as follows:

$$L_{NLPD}(X_*,\mathbf{f}_*,X,\mathbf{f},\boldsymbol{\theta}) = -\frac{1}{n_*}\sum_{i=1}^{n_*} log\, p(f_{*i}|X_{*i},X,\mathbf{f},\boldsymbol{\theta}) \tag{11}$$

where n_* is the number of test samples in X_*, and $\mathbf{f}_*$ corresponds to their function values.

Finally, we also measure the Root Mean Squared Error (RMSE) in the training-test set as a measure of the quality of the hyperparameters.

Having such a variety of metrics to choose from, it is unclear which of these metrics is most suitable for time series extrapolation tasks. We would like to find the best metric to optimize the hyperpameters of some of the most competitive kernel structures in the time series described in Table 3. Thus, we considered the best kernels found for each of the problems in [10]. For each metric and time series, we carried out an optimization process with Powell's algorithm, in order to find which one leads to the best results in terms of the RMSE in the test set. Each optimization process starts from a random hyperparameter set and stops when $Q = 5000$ samples have been taken. Due to the randomness of the process, 10 trials were carried out.

In Table 4, the results in the test set are shown for the thirteen time series in terms of average RMSE. The NLPD outperforms the rest of the metrics in five of the problems, while LML gets the best overall results in four time series and RMSE is the best choice in three. These three metrics show better average results compared to posterior LML and LOOCV, as the former is only able to obtain the best results in the *Daily Minimum Temperatures in Melbourne* time series, and the latter is not able to beat other metrics in any of the problems. As expected, in these extrapolation problems, LOOCV is the metric with the worst performance for hyperparameter optimization.

Statistical tests were used to determine if there is a metric that is more robust than the others in time series extrapolation problems[6]. First, we averaged all the RMSE results for each metric and time series, and then applied Friedman's test [48]. We found significant differences between all the metrics ($\alpha = 0.05$. p-value = 8.454e − 5). Then, we applied a post hoc test based on Friedman's test, and adjusted the results with Shaffer's correction [49]. In Fig. 2, the critical differences between the metrics are shown. As can be seen, there are no significant differences between NLPD, LML, RMSE and posterior LML. Similarly, the results between LOOCV and posterior LML do not differ significantly.

Overall, it can be seen that there is no metric best suited for guiding the hyperparameter optimization for all the time series, and the choice of the best metric depends on the problem. However, it is clear that some of these metrics, such as LOOCV, do not produce competitive results for any dataset.

6.5. Testing the proposed grammar

Once we have selected a metric to optimize the hyperparameters, we test whether better kernels can be found with our grammar, compared to kernel composition approaches. Particularly, we would like to know:

1. Whether it is possible to improve the composed kernels by means of manipulating elementary mathematical expressions, as we propose in this work.
2. Whether the proposed mutation operator allows such an improvement.

Considering the best kernels found by [10] for each time series, we generated 200 random mutations from each kernel, according to the method described in Section 4.2.2. Note that those original kernels had already been optimized by the cited kernel composition approach. Next, we performed a hyperparameter optimization process using the LML metric for each mutation, departing from the best hyperparameter values found in the original work. Finally, we measured the RMSE in the test set for all these mutations against the RMSE provided by [10].

Table 5 shows the results of this experiment for the 13 time series previously introduced. In 12 out of 13 time series, among the 200 randomly generated kernels, there are some kernels that have better results than the original one in terms of RMSE with fewer hyperparameters.

As we have shown in this experiment, we conclude that it is possible to improve the results achieved by kernels optimized through kernel composition, by manipulating their elementary mathematical expressions. We can also confirm that the mutation operator presented in this work allows such improvements.

6.6. Time series extrapolation benchmark

In view of the results obtained in the previous section, we evaluated EvoCov, along with the proposed alternative search methods, in the benchmark presented by [8]. To the best of our knowledge, this work provides the most extensive comparison in the literature in time series extrapolation with GPs. In this benchmark the following algorithms can be found:

Eureqa: A Symbolic Regression engine that uses genetic algo-

[6] The tests were carried out using the SCMAMP R package [47]

Table 4
Hyperparameter optimization metrics compared across different time series. The average RMSE in the test set is shown for each metric. The best results for each time series are shown in bold.

	LML	LOOCV	Post. LML	NLPD	RMSE
Airline	**37.00**	230.27	157.96	57.10	97.34
Solar	269.12	539.64	925.25	279.24	**132.04**
Mauna Loa	4.40	37.61	3.94	**2.34**	3.19
Wheat Price	54.13	99.64	67.94	**52.58**	266.37
Mel. Temps.	4.92	6.63	**4.62**	5.62	4.67
Traffic data	49352.14	4994073.33	38259.26	**23756.89**	25970.23
Avg. calls.	212.22	43078.20	1460.47	844.32	**55.08**
Radio freqs.	2.14	2.01	1.27	**0.74**	1.63
Gas Aus.	**13791.03**	179944.80	26403.55	18066.63	50771.39
Acid Aus.	**39.58**	1979.48	56.56	53.42	67.89
U.S. unemp.	**142.30**	4436.13	265.76	192.34	219.65
Births Queb.	44.78	16203.27	49.84	**44.54**	46.40
Wages Eng.	23.26	40.97	15.16	15.38	**13.61**

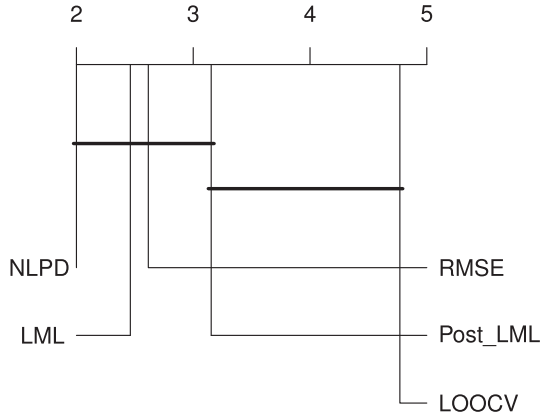


Fig. 2. Critical differences diagram between hyperparameter optimization metrics. The metrics are ordered following their rankings. The metrics with no significant differences between them are matched with a straight line.

Table 5
Results of the random mutation experiment. In the second column, the ratio of mutated kernels that are better fitted, in terms of RMSE, than the best kernel achieved by [10] is shown. The ratio of kernels that are simpler than the original one, according to the number of hyperparameters, is illustrated in the third column. In the last column, the ratio of kernels that are both better fitted and simpler can be found.

Name	Better fitted	Simpler	Both
Airline	0.38	0.43	0.13
Solar	0.55	0.32	0.21
Mauna Loa	0.18	0.34	0.05
Wheat Price	0.40	0.19	0.11
Mel. Temps.	0.23	0.27	0.00
Traffic data	0.24	0.38	0.07
Avg. calls.	0.50	0.48	0.20
Radio freqs.	0.36	0.30	0.05
Gas Aus.	0.71	0.41	0.30
Acid Aus.	0.56	0.43	0.23
U.S. unemp.	0.47	0.38	0.15
Births Queb.	0.48	0.32	0.13
Wages Eng.	0.38	0.28	0.08

rithms to search in the space of the possible equations [50]. Although this approach may seem similar to our work, Eureqa learns the predictive function itself, while our approach provides a probabilistic prediction by means of GPs.

Linear Regression (LIN): The basic linear regression is approximated by a GP model with a Linear kernel. The hyperparameters are learned by the LML optimization.

Squared Exponential (SE): A GP model with the SE kernel shown in Table 1 is used. The hyperparameters are also learned by optimizing the LML.

Bayesian variant of multiple kernel learning (MKL): A weighted sum of base kernels is used to construct more complex ones [51].

Change Point (CP) Modeling: A GP based approach allowing changepoints in kernels, that is, a combination of kernels where the weight of each of the components depends on the inputs [52–54].

Spectral Mixture Kernels (SK): These kernels model the spectral density with a Gaussian mixture [32].

Trend-Cyclical-Irregular (TCI) Models: The statistical model described by [55] is approximated by means of GPs and combining the Periodic kernels with Linear ones as covariance function.

GPSS: The greedy GP kernel search method described by [10] is used.

ABCD accuracy (ABCDa): An improvement of GPSS, introduced by [8], which includes the ChangeWindow and ChangePoint kernels.

ABCD interpretability (ABCDi): A modification of the previous approach that focuses on interpretability. This approach favors additive components as they are more interpretable by the practitioners. Similarly, the authors decided to remove the Rational Quadratic kernel as it is more difficult to describe automatically [8].

All the compositions of kernels that are included in GPSS can be represented in our grammar. Thus, the search space of EvoCov is a superset of the search space of GPSS. On the other hand, ABCD approaches include ChangeWindow and ChangePoint kernels that cannot be modeled with the current grammar of EvoCov. Hence, different kernels can be found by these approaches.

EvoCov is presented in two variants, one using LML to optimize the hyperparameters (EvoCov-LML), and the other variant using the NLPD metric (EvoCov-NLPD). LML is one of the most widely used metrics in the GP literature, and it is also the metric used by the competing GP methods. Although the differences between LML and NLPD were not significant in the experiment of the Section 6.4, we also include a EvoCov variant using this metric, in order to measure its performance when evolving kernels.

Table 6 shows the numerical results of the experimentation for each time series, while in Fig. 3, the overall results are illustrated. Note that RMSE is standardized by dividing by the smallest RMSE achieved in the experiments for each dataset, so that the best per-

Table 6
Standardized RMSE for each extrapolation problem and algorithm is shown. In our approaches, the best, the worst and the mean values are illustrated. The best results on average for each time series are shown in bold, while the best results among all methods are highlighted with an asterisk. Finally, in order to represent the distribution of the results over all problems, the mean and the median statistics are shown.

	ABCDa	GPSS	ABCDi	CP	LIN	MKL	SE	SP-bic	TCI	eureqa	Random Search			Go With The First			EvoCov LML			EvoCov NLPD		
											Best	Mean	Worst	Best	Mean	Worst	Best	Mean	Worst	Best	Mean	Worst
Airline	1.34	1.35	2.47	5.63	5.80	5.55	37.80	**1.17**	1.69	3.80	1.18	3.31	4.81	1.02	1.45	2.04	1.0*	1.31	2.01	1.11	1.56	2.27
Solar	1.66	2.13	2.08	1.71	1.77	1.65	2.66	1.99	1.70	4.36	1.0*	2.39	2.79	1.08	1.84	2.76	1.07	1.84	3.91	1.01	**1.60**	2.28
Mauna Loa	3.46	1.46	2.47	4.29	7.87	4.30	4.56	3.26	3.18	6.28	8.53	23.44	51.22	1.0*	1.50	2.81	1.07	**1.35**	2.08	1.08	2.50	7.94
Wheat Price	1.13	1.11	1.26	**1.06**	1.08	3.19	3.23	3.19	3.19	1.39	1.01	2.08	3.24	1.0*	2.37	8.37	1.04	1.29	2.03	1.32	1.77	2.87
Mel. Temps.	1.01	**1.0***	1.01	1.35	1.52	1.35	2.73	1.03	1.00	1.29	1.00	1.03	1.14	1.00	1.01	1.01	1.00	1.02	1.17	1.01	1.02	1.05
Traffic data	**1.46**	1.70	2.96	6.10	7.21	5.98	6.04	4.94	3.13	9.15	5.32	10.78	13.68	1.64	3.89	6.74	1.0*	3.63	5.92	1.46	4.82	6.63
Avg. calls.	2.98	2.26	**1.0***	3.54	28.76	1.80	22.63	11.04	1.80	493.30	1.21	7.76	22.72	1.33	6.82	20.89	1.16	2.81	5.78	1.45	11.69	34.33
Radio freqs.	5.61	3.32	3.40	5.65	7.79	5.65	15.29	1.81	4.17	2.55	1.31	2.69	4.51	1.0*	**1.35**	2.12	1.07	1.68	2.25	1.78	4.08	6.78
Gas Aus.	**1.01**	3.33	2.06	3.16	2.66	2.91	2.87	1.62	1.52	2.74	1.79	5.02	9.74	1.0*	2.75	6.09	1.06	2.50	4.65	1.39	2.34	4.89
Acid Aus.	**1.11**	1.82	1.60	2.49	3.89	1.78	1.20	1.58	1.99	2.18	1.68	2.04	2.96	1.0*	1.52	2.22	1.08	1.43	2.23	2.03	2.33	2.67
U.S. unemp.	2.91	1.66	2.73	2.24	**1.35**	2.30	2.54	4.81	3.01	3.01	1.32	5.25	28.20	1.30	1.59	2.02	1.0*	1.74	3.23	1.17	1.84	3.87
Births Queb.	1.17	1.25	**1.11**	2.04	2.17	2.05	1.84	1.71	1.73	2.14	1.30	1.74	1.98	1.0*	1.13	1.30	1.05	1.21	1.74	1.14	1.45	1.76
Wages Eng.	3.03	3.24	4.00	5.84	4.25	3.16	5.35	3.63	3.12	**1.0***	2.61	3.04	3.83	2.99	3.60	4.12	2.88	3.55	4.10	1.28	2.56	3.42
Mean	2.14	1.97	2.16	3.47	5.85	3.21	8.36	3.21	2.40	41.01		5.43			2.36			**1.95**			3.00	
Median	**1.46**	1.70	2.08	3.16	3.89	2.91	3.23	1.99	1.99	2.74		2.68			1.56			1.48			1.83	

formance on each dataset has a value of 1. Also, it is worth mentioning that, in the experiments conducted by [8], only one trial for each time series and algorithm was carried out[7], and, for our algorithms, the mean and the best of ten trials are shown.

As we can see in Table 6, EvoCov-LML achieves the best average result across all problems. On the other hand, its median result is the second best, very close to ABCDa, which obtains the best median result. Nevertheless, Fig. 3 indicates that the results obtained by EvoCov-LML have a lower interquartile range than those achieved by ABCDa. GoWithTheFirst algorithm scored the third best median RMSE, although its average performance is not as good as EvoCov-LML's. GPSS also delivers a solid performance, with the lowest interquartile range, achieving the second best average result, although it also shows a higher median than the aforementioned algorithms. Regarding the proposed NLPD EvoCov variant, its results are poorer than those of EvoCov-LML, indicating that this metric is not as good at evolving kernels as it is at optimizing hyperparameters. The Random Search also shows a worse performance than EvoCov approaches, with a very high variance, confirming the contribution of the mutation and crossover operators. Out of the GP approaches, although Eureqa outperforms the rest of the approaches in one time series, this symbolic regression engine is outperformed by EvoCov-LML in the rest of the time series.

Table 7 shows the number of hyperparameters of the best kernel found for each algorithm in each problem. It can be seen that the compositional kernel approaches (ABCDi, ABCDa and GPSS) have always more hyperparameters than any of our approaches on average. The only exception can be found in *Number of daily births in Quebec* time series, where ABCDi uses 6 hyperparameters, and EvoCov-LML and EvoCov-NLPD use 6.8 and 7.0 hyperparameters respectively.

We also conducted the tests of significance in this experiment. In order to compare the single evaluations found in [8], with the 10 trials conducted in our experiments, the median values of the latter were computed. Again, we applied Friedman's test [48], finding significant differences between the methods ($\alpha = 0.05$. p-value = $1.5e-7$). Then, we applied the post hoc test, and adjusted the results with Shaffer's correction [49], to produce the critical differences diagram shown in Fig. 4. As shown in the figure, the EvoCov-LML approach obtains the best ranking on average, and significantly better results that CP, Random Search, eureqa LIN and SE approaches.

Overall, EvoCov-LML is a competitive approach compared to the current state-of-the-art compositional kernel search approaches. In spite of not including the ChangePoint and ChangeWindow kernels, this approach is able to obtain comparable results to ABCDa, with kernels that have fewer hyperparameters, which makes them easier to optimize. Unfortunately, having only one execution of the methods compared to does not allow more sound conclusions to be provided.

6.7. Comparing our proposal to ad hoc kernel approaches

As we have mentioned in Section 2.4, many works in GPs propose a human-designed specific kernel for each particular problem. In the work carried out by [30], the number of tweets that contain a given hashtag in the Twitter timeline was predicted by means of GPs. The authors show that this time series information is also useful to predict the hashtags that a tweet has given its content. They propose an ad hoc kernel, Periodic Spikes (PS), that captures the periodicities of these hashtag time series. For example, the *#goodmorning* hashtag shows a clear periodic pattern, as it is more frequently tweeted in the mornings. On the other hand, there are some hashtags, such as #np (now playing), that do not follow the periodic pattern mentioned above, and according to the authors, there are kernels better suited than PS for these problems. Our proposal should be able to identify these situations, and offer the best possible kernel without human intervention.

Hence, we carried out the experiment introduced by [30], where *#goodmorning*, *#breakfast*, *#confessionhour*, *#fail*, *#fyi* and *#raw* hashtags were predicted[8]. For each hashtag, the number of tweets per hour was collected, using one month for training and the next one for testing, except for the *#goodmorning* hashtag, as in the original paper, where only 3 weeks were gathered, having 2 weeks to train and the last one to test.

In Fig. 5, an example of the hashtag prediction is illustrated, where the best model given by our approach in a single run is shown. In this problem, a periodic trend can be appreciated, which is successfully captured by our model.

Table 8 shows a comparison between EvoCov-LML, EvoCov-NLPD and the PS kernel. The experiments with the PS kernel were

[7] The results were gathered from the supplementary material of [8].

[8] Data can be found at https://web.sas.upenn.edu/danielpr/resources/

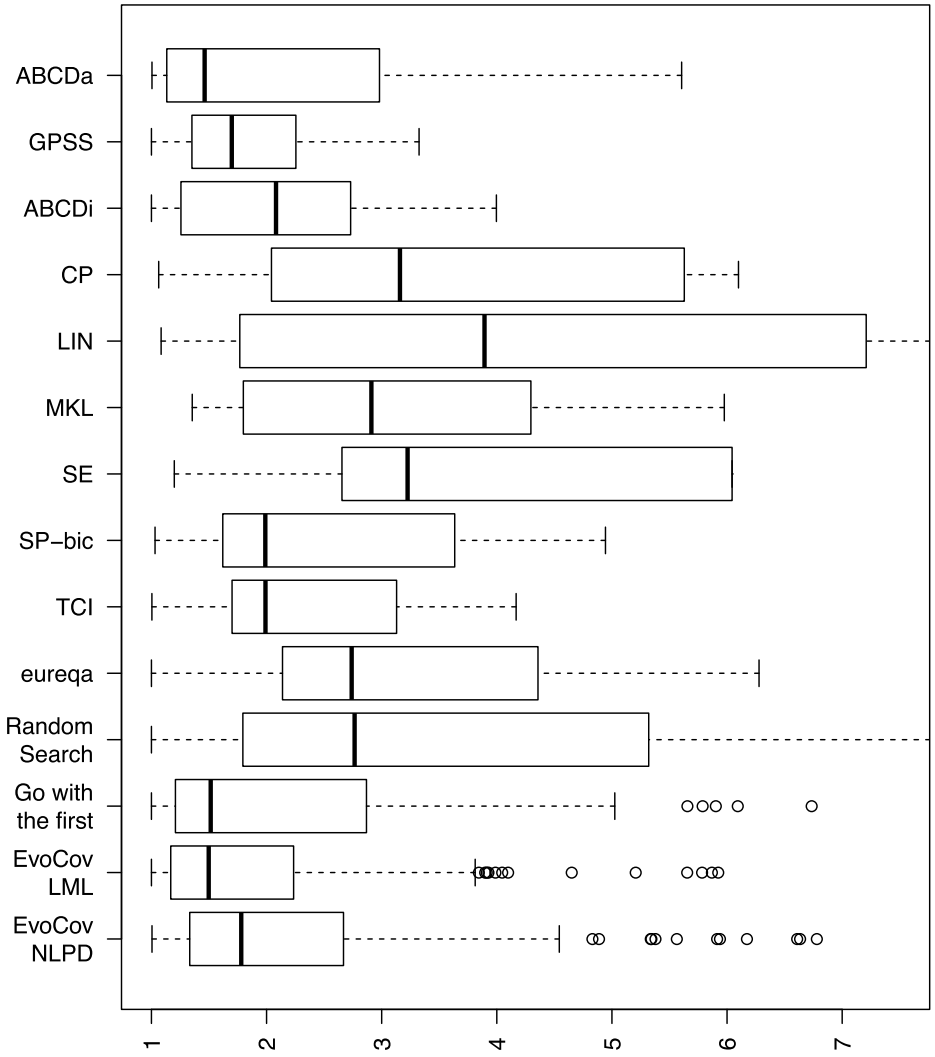


Fig. 3. Standardized RMSE of each algorithm over all problems. Note that the results of our algorithms have more observations due to the 10 trials.

Table 7
Number of hyperparameters for each extrapolation problem and algorithm. The noise hyperparameter is also considered. In our approaches, the average number of hyperparameters is shown. In ABCDa some data is missing as it could not be found in the supplementary material of the work done by [8].

	ABCDa	GPSS	ABCDi	CP	LIN	MKL	SE	SP-bic	TCI	Random Search	Go With The First	EvoCov LML	EvoCov NLPD
Airline	12	15	12	8	4	5	3	16	8	6.1	6.3	10.2	8.5
Solar	23	13	19	18	4	7	3	13	10	4.8	5.4	7.0	6.3
Mauna Loa	10	11	12	5	4	5	3	12	8	6.7	4.9	7.0	10.8
Wheat Price	14	7	13	13	4	5	3	7	5	5.4	4.9	6.6	5.8
Mel. Temps.	9	9	8	6	4	6	3	9	7	4.5	4.5	4.2	5.3
Traffic data	18	15	26	13	4	5	3	13	8	4.1	6.0	10.1	6.3
Avg. calls.	18	19	16	11	4	7	3	7	7	5.1	8.2	11.3	7.6
Radio freqs.	15	9	13	5	4	5	3	12	8	6.0	6.1	7.6	8.0
Gas Aus.	28	21	21	18	4	5	3	13	11	5.8	8.0	13.4	8.7
Acid Aus.	19	17	17	13	4	7	3	12	9	5.4	6.5	7.5	7.2
U.S. unemp.	13	15	10	9	4	4	3	18	8	6.4	4.8	7.1	8.3
Births Queb.	-	11	6	5	4	5	3	9	6	5.0	5.6	6.8	7.0
Wages Eng.	-	13	19	18	4	7	3	10	7	3.0	5.0	6.3	5.6
Mean	16.3	13.5	14.8	10.9	4	5.6	3	11.6	7.8	5.3	5.9	8.1	7.3

carried out using our software, and $Q = 5000$ samples were allowed to find the best hyperparameters for this kernel. As can be seen, EvoCov-NLPD is the best approach on average, obtaining the best results in the *#confessionhour* problem. EvoCov-LML is able to get the best score for the *#fail* and *#fyi* hashtags, finding a complex periodic pattern. It is also worth mentioning that the best trials of this algorithm obtain the best results in four out of five problems. However, in simpler periodic time series, such as *#goodmorning*, *#breakfast* and *#raw*, the PS kernel is the best choice, getting the best median result.

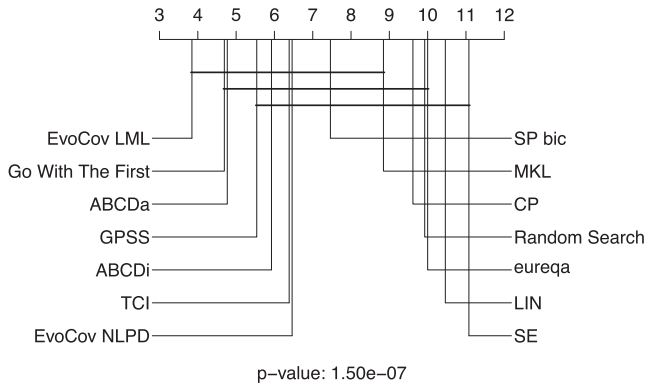


Fig. 4. Critical differences diagram of the Benchmark. The metrics are ordered following their rankings. The metrics with no significant differences between them are matched with a straight line.

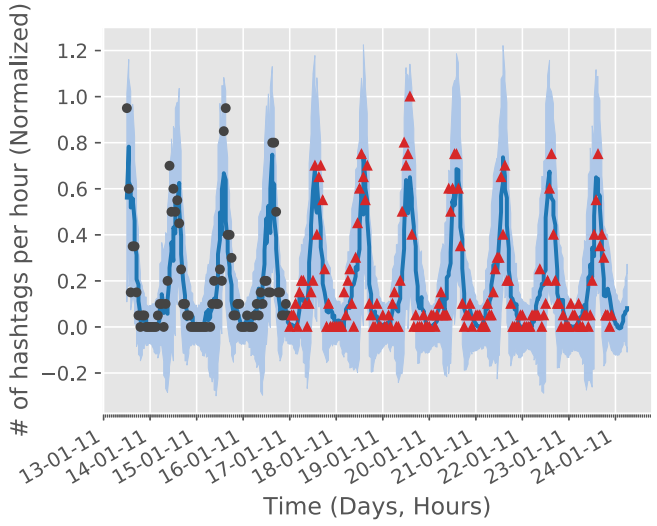


Fig. 5. Extrapolation of *#goodmorning* hashtag time series. The dots represent the last samples of the training set, while the triangles show the samples of the test set. The prediction given by a GP model with a kernel learned by the EvoCov-LML method is illustrated with a continuous blue curve for the mean, and the light blue shadow shows 3 times the standard deviation.

Table 9 shows the number of hyperparameters used by the different approaches. As expected, our approaches use more hyperparameters than the PS kernel, as this kernel is specifically designed for these problems.

All in all, the PS kernel is able to hold the best results in three out of six problems. On the other hand, the EvoCov approaches, without any knowledge about the problem, are able to obtain similar predictions to PS, even EvoCov-LML improving the results of the PS kernel in *#fyi*. Also note that the EvoCov approaches are able to produce better average results than the PS kernel, showing a more adaptable behavior.

Table 9
Number of hyperparameters for each extrapolation problem and algorithm. The noise hyperparameter is also considered. In our approaches, the average number of hyperparameters is shown.

	PS	EvoCov-NLPD	EvoCov-LML
#breakfast	3.0	4.8	5.7
#confessionhour	3.0	5.3	10.7
#fail	3.0	7.0	6.5
#fyi	3.0	4.3	2.9
#goodmorning	3.0	7.2	7.9
#raw	3.0	5.5	11.2
Mean	3.0	5.7	7.5

7. Conclusions

Kernel functions are widely used in several Machine Learning methods. GPs are one of these techniques, where a PSD kernel is used as a covariance function. This kernel function has to be carefully selected to achieve good results in any GP application. Although initial approaches used to rely on predefined kernels or ad hoc solutions for specific problems, there is an increasing interest in automatically learning these kernels. In this work, we have presented an evolutionary approach to learn kernel functions for GPs. While other approaches are based on kernel composition, in our approach, kernels are modeled by means of basic mathematical expressions.

This work has made the following contributions:

- Basic mathematical expressions as building blocks for GP kernels: We propose to bring the progress made in other Machine Learning areas to the GPs by considering its covariance function as a program that can be learned.
- Fast PSD check for GP kernels: Although some of the kernels generated by this new random method are not PSD, we have defined a kernel validation procedure that rapidly discards most non-PSD expression trees based on the properties of the covariance matrix.
- Hyperparameter inheritance: We have incorporated hyperparameter inheritance within GenProg, improving the efficiency of the algorithm.
- Metric comparison for hyperparameter optimization: We provide valuable insights about the suitability and performance of several metrics for hyperparameter optimization in extrapolation problems.

Table 8
The PS ad hoc kernel compared to EvoCov-LML and EvoCov-NLPD, in hashtag prediction problems. Standardized RMSE for each extrapolation problem and algorithm is shown. In all the approaches, the best, the worst and the mean results are illustrated. The best results on average are shown in bold, while the best results among all methods are highlighted with an asterisk.

	PS			EvoCov-NLPD			EvoCov-LML		
	Best	Mean	Worst	Best	Mean	Worst	Best	Mean	Worst
#breakfast	1.019	**1.045**	1.093	1.00*	1.09	1.319	1.06	1.148	1.537
#confessionhour	320.167	320.19	320.243	1.00*	**97.229**	320.339	1.017	197.542	320.472
#fail	1.008	1.363	4.300	1.00*	**1.034**	1.168	1.007	1.096	1.401
#fyi	1.019	1.05	1.075	1.024	1.041	1.056	1.00*	**1.001**	1.008
#goodmorning	1.00*	**1.058**	1.103	1.317	1.527	1.786	1.003	1.103	1.299
#raw	1.004	**1.365**	1.820	1.155	1.754	1.837	1.00*	1.677	1.833
Mean		54.345			**17.279**			33.928	
Median		1.211			1.309			**1.125**	

- Extensive benchmark in realistic problems: We have evaluated our proposal in an extensive benchmark of realistic problems, showing that our agnostic algorithm is competitive to a wide range of methods.

Altogether, these contributions enabled the design of a GenProg variant which is able to improve the state-of-the-art results in the application of GP to time series extrapolation. We can conclude that there is no need to rely on a priori defined kernels for GP time series extrapolation problems. According to the results of the experiment conducted in Section 6.5 it is possible to learn simpler and better kernels by evolving mathematical expression trees.

Further research in the grammar is suggested, extending it to ChangePoint and ChangeWindow kernels. We also suggest to investigate alternative regularization schemes as for the BIC metric, taking into account the correlation of the time series data. On the other hand, we propose continuing the work carried out to measure the performance of the hyperparameter optimization metrics for GP extrapolation problems. Finally, the application of the proposed technique to real-time applications must be carefully analyzed. In an scenario in which a training procedure can be executed with some preliminary data, our approach could be used to find the best performing kernel. Then, this (previously trained) kernel could be evaluated at real-time with little computational effort, comparable to that required by standard kernels.

CRediT authorship contribution statement

Ibai Roman: Writing - original draft, Software. **Roberto Santana:** Conceptualization, Writing - review & editing. **Alexander Mendiburu:** Investigation, Validation. **Jose A. Lozano:** Software, Methodology.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has been supported by the Spanish Ministry of Science and Innovation (project PID2019-104966 GB-I00), and the Basque Government (projects KK-2020/00049 and IT1244-19, and ELKARTEK program). Jose A. Lozano is also supported by BERC 2018–2021 (Basque government) and BCAM Severo Ochoa accreditation SEV-2017-0718 (Spanish Ministry of Science and Innovation).

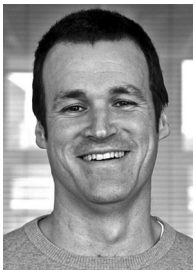
References

[1] C.E. Rasmussen, C.K. Williams, Gaussian processes for machine learning, MIT Press, 2006.
[2] J. Močkus, V. Tiesis, A. Zilinskas, The application of Bayesian methods for seeking the extremum, in: Towards Global Optimization, Vol. 2, Elsevier, 1978, pp. 117–129..
[3] D. Duvenaud, Automatic model construction with Gaussian processes, University of Cambridge, Thesis, 2014.
[4] M.G. Genton, Classes of Kernels for Machine Learning: A Statistics Perspective, J. Mach. Learn. Res. 2 (2002) 299–312.
[5] S. Ali, K.A. Smith-Miles, A meta-learning approach to automatic kernel selection for support vector machines, Neurocomputing 70 (1) (2006) 173–186, https://doi.org/10.1016/j.neucom.2006.03.004.
[6] M. Blum, M. Riedmiller, Optimization of Gaussian Process Hyperparameters using Rprop, in: European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, 2013, pp. 339–344..
[7] E. Brochu, V.M. Cora, N. de Freitas, A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning, arXiv:1012.2599 [cs]ArXiv: 1012.2599 (Dec. 2010)..
[8] J.R. Lloyd, D. Duvenaud, R. Grosse, J.B. Tenenbaum, Z. Ghahramani, Automatic Construction and Natural-Language Description of Nonparametric Regression Models, in: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, AAAI'14, AAAI Press, 2014, pp. 1242–1250, event-place: Québec City, Québec, Canada..
[9] G. Kronberger, M. Kommenda, Evolution of Covariance Functions for Gaussian Process Regression Using Genetic Programming, in: Computer Aided Systems Theory - EUROCAST 2013, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2013, pp. 308–315. doi:10.1007/978-3-642-53856-8_39..
[10] D. Duvenaud, J. Lloyd, R. Grosse, J. Tenenbaum, Z. Ghahramani, Structure Discovery in Nonparametric Regression through Compositional Kernel Search, in: Proceedings of The 30th International Conference on Machine Learning, 2013, pp. 1166–1174.
[11] J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 1992.
[12] W. Chu, Z. Ghahramani, Gaussian Processes for Ordinal Regression, J. Mach. Learn. Res. 6 (Jul) (2005) 1019–1041.
[13] Z. Wang, N. de Freitas, Theoretical Analysis of Bayesian Optimisation with Unknown Gaussian Process Hyper-Parameters, arXiv:1406.7758 [cs, stat] ArXiv: 1406.7758 (Jun. 2014)..
[14] D.J.C. MacKay, Bayesian Methods for Backpropagation Networks, in: Models of Neural Networks III, Physics of Neural Networks, Springer, New York, NY, 1996, pp. 211–254. doi:10.1007/978-1-4612-0723-8_6..
[15] R.M. Neal, Bayesian Learning for Neural Networks, Springer-Verlag, New York, 1996.
[16] S. Sundararajan, S.S. Keerthi, Predictive Approaches for Choosing Hyperparameters in Gaussian Processes, Neural Comput. 13 (5) (2001) 1103–1118, https://doi.org/10.1162/08997660151134343.
[17] D.J.J. Toal, N.W. Bressloff, A.J. Keane, Kriging Hyperparameter Tuning Strategies, Am. Inst. Aeronautics Astronautics J. 46 (5) (2008) 1240–1252, https://doi.org/10.2514/1.34822.
[18] D.J.J. Toal, N.W. Bressloff, A.J. Keane, C.M.E. Holden, The development of a hybridized particle swarm for kriging hyperparameter tuning, Eng. Optim. 43 (6) (2011) 675–699, https://doi.org/10.1080/0305215X.2010.508524.
[19] R. Garnett, M.A. Osborne, P. Hennig, Active Learning of Linear Embeddings for Gaussian Processes, in: Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence, AUAI Press, Arlington, Virginia, United States, 2014, pp. 230–239.
[20] R. Benassi, J. Bect, E. Vazquez, Robust Gaussian Process-Based Global Optimization Using a Fully Bayesian Expected Improvement Criterion, in: C. A.C. Coello (Ed.), Learning and Intelligent Optimization, no. 6683 in Lecture Notes in Computer Science, Springer, Berlin Heidelberg, 2011, pp. 176–190.
[21] A.D. Bull, Convergence Rates of Efficient Global Optimization Algorithms, J. Mach. Learn. Res. 12 (2011) 2879–2904.
[22] N. Durrande, D. Ginsbourger, O. Roustant, Additive covariance kernels for high-dimensional Gaussian process modeling, Annales de la Faculté de Sciences de Toulouse Tome 21 (numéro 3) (2012) p. 481–499..
[23] D.J. Montana, Strongly Typed Genetic Programming, Evol. Comput. 3 (2) (1995) 199–230, https://doi.org/10.1162/evco.1995.3.2.199.
[24] N. Haji Ghassemi, M. Deisenroth, Analytic Long-Term Forecasting with Periodic Gaussian Processes, Proc. Mach. Learn. Res. (2014) 303–311.
[25] P. Koch, B. Bischl, O. Flasch, T. Bartz-Beielstein, C. Weihs, W. Konen, Tuning and evolution of support vector kernels, Evol. Intel. 5 (3) (2012) 153–170, https://doi.org/10.1007/s12065-012-0073-8.
[26] F. Zhang, Positive Semidefinite Matrices, in: Matrix Theory, Universitext, Springer, New York, NY, 2011, pp. 199–252. doi:10.1007/978-1-4614-1099-7_7..
[27] F.-A. Fortin, F.-M.D. Rainville, M.-A. Gardner, M. Parizeau, C. Gagné, DEAP: Evolutionary Algorithms Made Easy, J. Mach. Learn. Res. 13 (Jul) (2012) 2171–2175.
[28] G. Schwarz, Estimating the Dimension of a Model, Ann. Stat. 6 (2) (1978) 461–464, https://doi.org/10.1214/aos/1176344136.
[29] M.J.D. Powell, An efficient method for finding the minimum of a function of several variables without calculating derivatives, Computer J. 7 (2) (1964) 155–162, https://doi.org/10.1093/comjnl/7.2.155.
[30] D. Preoţiuc-Pietro, T. Cohn, A temporal model of text periodicities using Gaussian Processes, in: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 2013, pp. 977–988.
[31] E.D. Klenske, M.N. Zeilinger, B. Schölkopf, P. Hennig, Nonparametric dynamics estimation for time periodic systems, in: 2013 51st Annual Allerton Conference on Communication, Control, and Computing (Allerton), 2013, pp. 486–493. doi:10.1109/Allerton.2013.6736564..
[32] A. Wilson, R. Adams, Gaussian Process Kernels for Pattern Discovery and Extrapolation, in: Proceedings of The 30th International Conference on Machine Learning, 2013, pp. 1067–1075.
[33] S. Bochner, Lectures on Fourier Integrals. (AM-42), Princeton University Press, 1959, google-Books-ID: O1jQCwAAQBAJ..
[34] G.E. Hinton, R.R. Salakhutdinov, Using Deep Belief Nets to Learn Covariance Kernels for Gaussian Processes, in: J.C. Platt, D. Koller, Y. Singer, S.T. Roweis (Eds.), Advances in Neural Information Processing Systems 20, Curran Associates Inc, 2008, pp. 1249–1256.
[35] Z. Zhu, J. Zhang, J. Zou, C. Deng, Multi-Kernel Gaussian Process Latent Variable Regression Model for High-dimensional Sequential Data Modeling, Neurocomputing (2018), https://doi.org/10.1016/j.neucom.2018.07.082.
[36] L. Dioşan, A. Rogozan, J.P. Pecuchet, Evolving kernel functions for SVMs by genetic programming, in: Sixth International Conference on Machine Learning

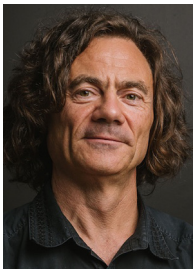
and Applications (ICMLA 2007), 2006, pp. 19–24, https://doi.org/10.1109/ICMLA.2007.70.
[37] W. Bing, Z. Wen-qiong, C. Ling, L. Jia-hong, A GP-based kernel construction and optimization method for RVM, in: 2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE), Vol. 4, 2010, pp. 419–423. doi:10.1109/ICCAE.2010.5451646..
[38] T. Howley, M.G. Madden, The Genetic Kernel Support Vector Machine: Description and Evaluation, Artif. Intell. Rev. 24 (3–4) (2005) 379–395, https://doi.org/10.1007/s10462-005-9009-3.
[39] C. Gagné, M. Schoenauer, M. Sebag, M. Tomassini, Genetic Programming for Kernel-Based Learning with Co-evolving Subsets Selection, in: Parallel Problem Solving from Nature - PPSN IX, in: Lect. Notes Comput. Sci., Springer, Berlin, Heidelberg, 2006, pp. 1008–1017, https://doi.org/10.1007/11844297_102.
[40] L. Dioşan, A. Rogozan, J.-P. Pecuchet, Improving classification performance of Support Vector Machine by genetically optimising kernel shape and hyper-parameters, Appl. Intell. 36 (2) (2012) 280–294, https://doi.org/10.1007/s10489-010-0260-1.
[41] K.M. Sullivan, S. Luke, Evolving Kernels for Support Vector Machine Classification, in: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO '07, ACM, New York, NY, USA, 2007, pp. 1702–1707. doi:10.1145/1276958.1277292..
[42] T. Howley, M.G. Madden, An Evolutionary Approach to Automatic Kernel Construction, in: Artificial Neural Networks - ICANN 2006, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2006, pp. 417–426. doi:10.1007/11840930_43..
[43] L. Rastrigin, The convergence of the random search method in the extremal control of a many parameter system, Automaton Remote Control 24 (1963) 1337–1342.
[44] D. Aldous, U. Vazirani, "Go with the winners" algorithms, in: Proceedings 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 492–501, https://doi.org/10.1109/SFCS.1994.365742.
[45] F. Bachoc, Cross Validation and Maximum Likelihood estimations of hyper-parameters of Gaussian processes with model misspecification, Comput. Statistics Data Anal. 66 (2013) 55–69, https://doi.org/10.1016/j.csda.2013.03.016.
[46] J. Quiñonero-Candela, C.E. Rasmussen, F. Sinz, O. Bousquet, B. Schölkopf, Evaluating Predictive Uncertainty Challenge, in: J. Quiñonero-Candela, I. Dagan, B. Magnini, F. d'Alché Buc (Eds.), Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment, Lect. Notes Comput. Sci., Springer, Berlin Heidelberg, 2006, pp. 1–27.
[47] B. Calvo, G. Santafé, scmamp: Statistical Comparison of Multiple Algorithms in Multiple Problems, The R Journal 8 (1) (2016) 248–256.
[48] M. Friedman, The use of ranks to avoid the assumption of normality implicit in the analysis of variance, Journal of the American Statistical Association 32 (200) (1937) 675–701.
[49] J.P. Shaffer, Modified Sequentially Rejective Multiple Test Procedures, J. Am. Stat. Assoc. (2012).
[50] M. Schmidt, H. Lipson, Eureqa (version 0.98 beta)[software], Nutonian, Somerville, Mass, USA (2013)..
[51] F.R. Bach, G.R.G. Lanckriet, M.I. Jordan, Multiple Kernel Learning, Conic Duality, and the SMO Algorithm, in: Proceedings of the Twenty-first International Conference on Machine Learning, ICML '04, ACM, New York, NY, USA, 2004, pp. 6–. doi:10.1145/1015330.1015424..
[52] R. Garnett, M.A. Osborne, S. Reece, A. Rogers, S.J. Roberts, Sequential Bayesian Prediction in the Presence of Changepoints and Faults, Computer J. 53 (9) (2010) 1430–1446, https://doi.org/10.1093/comjnl/bxq003.
[53] Y. Saatçi, R. Turner, C.E. Rasmussen, Gaussian Process Change Point Models, in: Proceedings of the 27th International Conference on International Conference on Machine Learning, 2010, pp. 927–934.
[54] E.B. Fox, D.B. Dunson, Multiresolution Gaussian Processes, in: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates Inc, 2012, pp. 737–745.
[55] D. Lind, W. Marchal, S. Wathen, Basic Statistics for Business & Economics, McGraw-Hill/Irwin series Business statistics, McGraw-Hill/Irwin, 2006.

Ibai Roman received his bachelor degree in Computer Science from Mondragon University in 2010. In 2020, he obtained a Ph.D. in Computer Science from the University of the Basque Country UPV/EHU. He is currently a lecturer and researcher at the Mondragon University. His research interests are Bayesian Optimization and Gaussian Process.

Roberto Santana received and M.Sc. degree in Computer Science from the University of Havana, Cuba, in 1996. He received a Ph.D. in Mathematics from the University of Havana in 2005 and a Ph.D. in Computer Science from the University of the Basque Country, in Spain, in 2006. He is a Tenured researcher at the Intelligent Systems Group (ISG), Department of Computer Science and Artificial Intelligence, University of the Basque Country UPV/EHU, Spain. Roberto Santana's research interests comprise the use of probabilistic graphical models in evolutionary algorithms and the application of Machine Learning methods to problems from bioinformatics and neuroinformatics. He has published over 25 papers in international journals, more than 60 papers in international conferences.



Alexander Mendiburu is an associate professor at the department of Computer Architecture and Technology, school of Computer Science, University of the Basque Country UPV/EHU. His main research areas are Evolutionary Computation, Probabilistic Graphical Models, Time Series, and Parallel Computing.



Jose A. Lozano His research interests are in the field of Statistical Machine Learning and Combinatorial Optimization. Particularly in Machine Learning, he pursues the design and evaluation of new classification paradigms and algorithms which are able to produce predictive models which can be applied in different fields. On the other hand, in the Combinatorial Optimization field he has contributed to developing new heuristics and metaheuristic algorithms which are able to find a balance between the quality of the solutions and computational time, study their theoretical properties and apply them in the solution of real problems.