Contributions to automatic learning of kernel functions

Ibai Roman Txopitea

Supervised by Roberto Santana, Alexander Mendiburu and Jose A. Lozano

2020

eman ta zabal zazu

Konputazio Zientziak eta Adimen Artifizialaren Saila

Departamento de Ciencias de la Computación e Inteligencia Artificial

Contributions to automatic learning of kernel functions

by

Ibai Roman Txopitea

Supervised by Roberto Santana, Alexander Mendiburu and Jose A. Lozano

Dissertation submitted to the Department of Computer Science and Artificial Intelligence of the University of the Basque Country (UPV/EHU) as partial fulfilment of the requirements for the PhD degree in Computer Science

Donostia - San Sebastián, May 2020

Zuriñe, familia
eta lagunei

Acknowledgments

First of all, I would like to thank my advisors Alexander Mendiburu, Roberto Santana, and Jose A. Lozano. Without their wise guidance, this dissertation would never have been possible. Your patience and encouragement, as well as the scientific discussions, have been essential in the development of this dissertation. I hope to continue working together in the future. Thank you very much.

I also owe a debt of gratitude to Prof. Lu for allowing me to complete a stay as a visitor in the Brain-like Computing and Machine Intelligence center at the Shanghai Jiao Tong University.

Special thanks go to to my friends in the Intelligent Systems Group with whom I have shared these years, and also to the members of other research groups at the Faculty of Computer Science. They are part of this work.

I would like to acknowledge the great work of John Kennedy, whose comments on English writing have improved the readability of the published contributions.

I am grateful to the Basque Government for the financial support (PRE-2013-1-919). In addition, this work has been supported by the Spanish Ministry of Science and Innovation (project TIN2016-78365-R), and the Basque Government (IT1244-19 and ELKARTEK programs).

Finally, I would like to thank the support of Zuriñe, my family and friends. This thesis is also for them.

Abstract

Many Machine Learning algorithms rely on kernel functions to solve the proposed tasks. Among these algorithms, we can find kernel methods, including Support Vector Machines (SVMs), or Bayesian inference methods, such as Gaussian Processes (GPs). However, the validity of these algorithms to solve problems depends largely on the kernel function, and there is no kernel function that is optimal for all application domains. Some standard kernels, which are invalid under certain conditions, have been proposed in the literature. Researchers have also manually designed problem-specific kernels aiming to improve the performance of those standard kernels. However, choosing among the set of standard kernels or creating a new one requires expert knowledge of the algorithm and the application domain. Therefore, there is a great interest in automating this process.

The work done in the SVM literature suggests that using Genetic Programming is a valid solution to learn kernel functions in an automatic manner. However, SVMs consist of several components that interact with each other and there is not a complete understanding of the implications that these interactions have in the evolution of kernel functions.

In this research direction, in the kernel search for SVMs, two types of grammars have been proposed to define the search space, kernel composition methods and approaches based on basic mathematical expressions; while in GPs, most of the work has focused only on the combination of kernels. In fact, we have not found any approach based on basic mathematical expressions in the GP field.

In this dissertation we study the use of Genetic Programming to learn kernels for both SVMs and GPs. First, we describe the analysis made in the field of SVMs, where we have studied the different interactions between the components of SVMs during the evolution of kernels, and provide some guidelines to improve the kernel learning process. Next, we propose a method based on basic mathematical expressions to learn kernels for GPs through Genetic Programming, and test the validity of this method in various applications in time series prediction and Natural Language Processing.

Contents

Symbols and Notation

Symbol	Meaning
$\mathbf{x}$	vector
$\mathbf{x}^T$	transpose of a vector
$\mathbf{1}$	vector of ones
M	matrix
$\|M\|$	determinant of M matrix
$\\|\mathbf{w}\\|$	Euclidean norm
$\delta(\mathbf{x}, \mathbf{x}')$	Kronecker delta. 1 if $\mathbf{x}$ equals $\mathbf{x}'$ and 0 otherwise
$\langle \cdot, \cdot \rangle_{\mathcal{V}}$	inner product in Space $\mathcal{V}$
$\mathbb{N}$	natural numbers
$\mathbb{R}$	real numbers
n	number of input samples
d	dimension of the input space
X	d x n matrix of the input samples
X_*	test input samples
$\mathbf{x}_i$	ith input sample
X_{-i}	$\mathbf{x}_i$ removed from X
θ	vector of kernel hyperparameters

Glossary

AML Automatic Machine Learning. 4, 16, 23
ARD Automatic Relevance Determination. 8

BIC Bayesian Information Criterion. 18, 45–47, 54–56, 91, 100, 108

CBOW Continuous Bags of Words. 82

GenProg Genetic Programming. IX, 4, 18, 23, 32–39, 45–47, 49, 51, 54–57, 59, 76, 82, 86, 94, 107, 108, 110
GP Gaussian Process. IX, 4, 5, 7–9, 12–16, 18–20, 49–51, 56–58, 63, 64, 68, 69, 73, 75–77, 79, 80, 82–84, 86, 92, 94, 96, 97, 101–103, 107–110, 117, 118

HTER Human Translation Edit Rate. 80, 90, 91, 93, 95, 96, 98, 100

LML Log Marginal Likelihood. 13, 14, 54, 55, 63–66, 68–73, 75–77, 80, 84, 86, 94
LOOCV Leave-one-out Cross Validation. 13, 14, 54, 55, 63–65
LSTM Long Short-term Memory. 83, 94, 95, 103

NLP Natural Language Processing. IX, 5, 49, 79, 80, 82, 89, 101, 102, 108–110
NLPD Negative Log Predictive Densities. 64, 65, 69–73, 75–77, 80, 84, 86–89, 92

PCC Pearson's Correlation Coefficient. 79, 84, 86–89, 91
PMLB Penn Machine Learning Benchmarks. 26, 27, 30, 32–36, 38, 41, 43, 44, 46
PS Periodic Spikes. 73, 75–77
PSD Positive semi-definite. 3, 5, 7, 12, 15, 16, 18, 20, 29, 30, 49, 51–53, 56, 63, 109

QE Quality Estimation. 79, 80, 82, 83, 89, 91–94, 97, 100–103

RBF Radial Basis Function. 8, 25–27, 35, 36, 39, 41

RMSE Root Mean Squared Error. 64–67, 69, 70, 72, 75, 76, 95–100, 102
RNN Recurrent Neural Network. 83, 94, 103

SA Sentiment Analysis. 5, 79, 80, 82–85, 94, 101, 102, 109, 110
SE Squared Exponential. 8, 9, 13–16, 19, 25, 68, 70, 71, 79, 86, 87, 95–97, 103, 113
SVM Support Vector Machine. IX, 3–5, 7–12, 15, 16, 18–20, 23–26, 28–30, 33, 39–47, 49–51, 56, 107–109

Part I

Introduction

1 Introduction

Machine Learning is a field devoted to designing algorithms and techniques which are able to complete specific tasks. Based on sample data, mathematical models are built for different purposes, such as classification, regression, clustering, and so on. Many machine learning algorithms use kernel functions in order to encode the particular manner in which the similarity between any pairs of data points is defined. Introduced by Mercer (1909), kernel functions hold Mercer's condition, which, in the discrete case, is analogous to the Positive semi-definite (PSD) matrix property.

Among the machine learning algorithms that are based on kernel functions, the kernel methods are particularly noteworthy. These methods are characterized by the use of the "kernel trick", which consists of turning a linear model into a non-linear one by replacing the inner product between the features with a kernel function. These functions compute the dot product of the projections of two data points into a feature space (Shawe-Taylor et al., 2004), avoiding the explicit computation of the feature mapping. Kernel methods include the kernel perceptron, kernel principal component analysis, or kernel canonical correlation analysis, their best known example being the Support Vector Machine (SVM).

SVMs (Vapnik, 1963) have long been the reference paradigm in supervised classification and regression. Although the field is nowadays overwhelmed by the application of deep learning approaches, SVMs remain one of the best alternatives whenever the requirements for using deep neural networks are not met. When applied to binary classification problems, SVMs separate samples from the two different classes by means of a hyperplane that maximizes the gap to the nearest samples in order to ensure a proper generalization. SVMs can even handle non-linearly-separable problems by means of a kernel function (Boser et al., 1992), and when this kernel meets Mercer's condition (Mercer, 1909), the optimal hyperplane can be found.

In spite of the fact that SVMs are adequate tools to solve classification problems, the choice of the kernel heavily influences their performance, and there is no rule of thumb to select it. In addition, most of the kernels have some parameters that need to be adjusted, which hardens the kernel selection problem. These parameters, often called hyperparameters, are usually tuned by optimizing a given metric.

While some standard kernels proposed in the literature are straightforwardly used in different applications, tailored kernels produce generally better results (Duvenaud et al., 2013; Howley and Madden, 2006), since each problem has its own specific characteristics. Several works in the literature pose the selection of the kernel as a search problem in the space of kernels with no human intervention (Howley and Madden, 2006; Dioşan et al., 2012; Koch et al., 2012). In order to achieve an Automatic Machine Learning (AML) approach, complex search methods, such as Genetic Programming (GenProg) (Koza, 1992), have been used in the literature to find kernels that improve the performance of standard kernels (Howley and Madden, 2006; Dioşan et al., 2008; Kronberger and Kommenda, 2013).

In the SVM literature, two main approaches have been proposed to define the kernel function space: kernel composition methods and basic mathematical expression based approaches. In kernel composition methods (Dioşan et al., 2012; Sullivan and Luke, 2007), some composition rules that preserve Mercer's condition are used to combine predefined kernels, guaranteeing that the newly created kernels also meet this condition. Alternatively, basic mathematical expression based methods (Howley and Madden, 2005; Dioşan et al., 2007; Koch et al., 2012) use simpler expressions as building-blocks for the kernel functions. While these approaches do not guarantee that the kernels satisfy Mercer's condition, they allow a richer and broader set of kernels to be explored.

There is extensive literature about kernel search methods for SVMs. However, in addition to the kernel, SVMs involve several components that must be adjusted in order to obtain a good performance. In the effort to study the search methods themselves, little attention has been paid to these other SVM components, and there are still some open questions regarding their influence when it comes to selecting the best possible kernel.

Besides the application of kernel functions in kernel methods, they are also required in some Bayesian inference methods, such as Gaussian Processes (GPs) (Rasmussen and Williams, 2006). Despite being studied by different communities, it is widely known that kernel methods and GPs are closely related (Kanagawa et al., 2018).

GPs (Rasmussen and Williams, 2006) are one of the most studied techniques in Machine Learning for regression tasks and they have also been extensively applied for function approximation. In comparison to other regression methods, GPs not only provide a prediction of a given function, but also estimate the uncertainty of these predictions. Furthermore, GPs have been used for optimization tasks under the umbrella of Bayesian optimization (Močkus et al., 1978), as this model relies on strong Bayesian inference foundations and can be updated when new evidence becomes available.

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 PSD kernel, i.e., a Mercer kernel. The assumption in GPs is that, as the similarity between two solutions increases, so does the similarity of the function value at these solutions, which makes the kernel a key element in any application of GPs.

Any application of GPs requires a kernel function to be defined and their hyperparameters to be adjusted to the data. In early applications of GPs, the kernel function was often selected from a predefined set (Brochu et al., 2010), or designed by an expert (Rasmussen and Williams, 2006). Then, the search for the hyperparameters was approached as an optimization process. Although some recent works tackle the question of automating the choice of the kernel (Lloyd et al., 2014; Kronberger and Kommenda, 2013; Duvenaud et al., 2013), automatic kernel search has not been investigated in GPs to the same extent as in kernel methods. Most works have focused on kernel combination, and, to the best of our knowledge, no basic mathematical expression approaches have been proposed.

Taking into account that the kernels play a main role in this kind of methods, the objective of this dissertation is to address the issues and limitations of static kernels and contribute with automated kernel learning techniques to, both, SVM and GP fields. Regarding SVMs, we study in depth the components involved in SVM kernel learning beyond the search method itself. We address their interactions with the kernel learning process in order to shed some light on the issues found in the literature. As for the Bayesian inference methods, we propose using a basic mathematical expression based method for GPs, bringing the advances made in the SVM literature to the GPs. By exploring this wider search space of kernels that also contains non-Mercer ones, this method is able to find kernels that improve the state-of-the-art results of time series regression tasks with GPs. We have also successfully applied this technique to solve various Natural Language Processing (NLP) tasks by means of GPs, first, tackling Sentiment Analysis (SA) tasks by means of a multi-objective extension of our proposal, and secondly, improving the performance of the most used kernels in translation effort prediction.

This dissertation is organized as follows: First, in Chapter 2, a background about the automatic learning of kernel functions is provided. Next, the main contributions of the dissertation are presented. In Chapter 3, the influence of different SVM components in the kernel learning process is analyzed. Chapter 4 introduces our proposal to learn kernels for GPs, which is applied to time series problems in Chapter 5. In Chapter 6, this GP kernel learning proposal is also applied to SA and translation effort prediction NLP problems. Finally, in Chapter 7, the concluding remarks and the lines for future work are presented.

2

Background

This chapter presents the background of the dissertation. First, in Section 2.1, we introduce the definition of the kernel functions. Next, the foundations of the SVM and GP models are explained in Section 2.2. Finally, the literature regarding automatic kernel learning is reviewed in Section 2.3.

2.1 Kernel functions

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

In the SVM literature PSD kernels are often addressed as Mercer kernels, since a PSD kernel holds Mercer's condition (Mercer, 1909), as it satisfies:

$$\int_{\mathcal{S}}\int_{\mathcal{S}} g(x)k(x,x')g(x')\,dx\,dx' \geq 0 \tag{2.1}$$

for any square integrable function $g(x)$.

Kernel functions can be also interpreted as inner products in some Hilbert space (Hilbert, 1906). Given a mapping function $\phi : \mathcal{X} \rightarrow \mathcal{V}$ a PSD kernel can be written as:

$$k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{V}} \tag{2.2}$$

Although the use of kernels enables the algorithms to be applied to other types of data, in this dissertation we work with real vector spaces.

2.1.1 Standard kernel functions

The kernel functions can be divided into two main families: stationary and non-stationary kernels (Genton, 2002).

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:

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= \widehat{k}(r) \\ r &= \left\| \frac{\mathbf{x}}{\theta_l} - \frac{\mathbf{x}'}{\theta_l} \right\| \end{aligned} \tag{2.3}$$

where $\widehat{k}$ is a function that guarantees that the kernel satisfies Mercer's condition 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 (MacKay, 1996; Neal, 1996).

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:

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= \widehat{k}(s) \\ s &= \left(\frac{\mathbf{x} - \theta_s \mathbf{1}}{\theta_l} \right) \left(\frac{\mathbf{x}' - \theta_s \mathbf{1}}{\theta_l} \right)^T \end{aligned} \tag{2.4}$$

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

Table 2.1 shows eleven standard kernels used in different applications of SVMs and GPs (Howley and Madden, 2006; Koch et al., 2012; Dioşan et al., 2012). One of the most popular kernel choices, the Squared Exponential (SE) kernel (also known as Radial Basis Function (RBF) in the SVM literature or the Exponentiated Quadratic) can be represented as shown in Figure 2.1. This kernel is known to capture the smoothness property of the data.

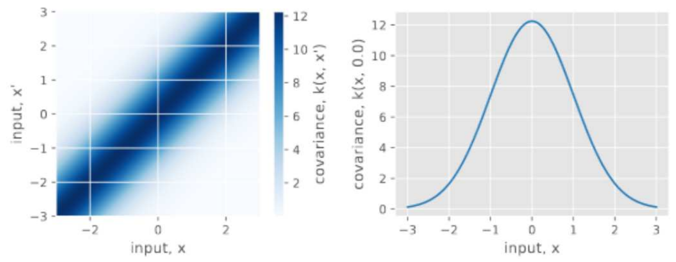


Fig. 2.1. SE kernel. On the left side, the actual outcome of the kernel function is shown according to the values of the input vectors. On the right, the same function is shown, when $x' = 0$.

Kernel function expressions	
Constant	$k_{CON}(\mathbf{x},\mathbf{x}') = \theta_0$
White Noise	$k_{WN}(\mathbf{x},\mathbf{x}') = \theta_0\ \delta(\mathbf{x},\mathbf{x}')$
Exponential	$k_E(r) = \theta_0^2\ exp\,(-r)$
γ exponential	$k_{E\gamma}(r) = \theta_0^2\ exp\,(-r^\gamma)$
Squared Exponential	$k_{SE}(r) = \theta_0^2\ exp\left(-\frac{1}{2}r^2\right)$
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	$k_{LIN}(s) = s$

Table 2.1. Standard kernel functions. θ_0 and θ_p are the kernel hyperparameters, called amplitude and period respectively. r is described in Equation (2.3), while s is presented in Equation (2.4).

2.2 Models

The kernel function is the key component of several machine learning algorithms. Among them, SVMs and GPs are the most representative. Thus, in this chapter, a gentle introduction to both methods is provided.

2.2.1 Support Vector Machines

Support Vector Machines (SVMs) were introduced by Vapnik in 1963 as non-probabilistic linear classifiers to solve binary classification problems. Later, probabilistic variants (Platt, 1999) of SVMs and extensions to multi-class problems (Crammer and Singer, 2001) were proposed.

In a supervised binary classification scenario, linear classifiers, such as SVMs, classify these samples by means of a hyperplane. Nevertheless, there are many ways to position this hyperplane. SVMs are characterized by the use of a hyperplane that maximizes the separation, or margin, between classes, as can be seen in Figure 2.2.

In the most trivial case, where the samples are linearly-separable, the hard-margin formulation can be used. Then, the margin from the plane to the solutions of each class is maximized to achieve a better generalization. Given some data $D = \{\mathbf{x}_i, y_i\}_{i=1}^n$ ($n \in \mathbb{N}$), where $y_i \in \{-1, +1\}$ indicates to which class $\mathbf{x}_i \in \mathbb{R}^d$ ($d \in \mathbb{N}$) belongs, the maximal separating hyperplane can be found by solving the following optimization problem:

$$\begin{aligned} & min\left(\frac{1}{2}\mathbf{w}\mathbf{w}^T\right) \\ \text{subject to} & \\ & y_i(\mathbf{w}\mathbf{x}_i^T + b) \geq 1, \forall i \in \{1,2,...,n\} \end{aligned} \tag{2.5}$$

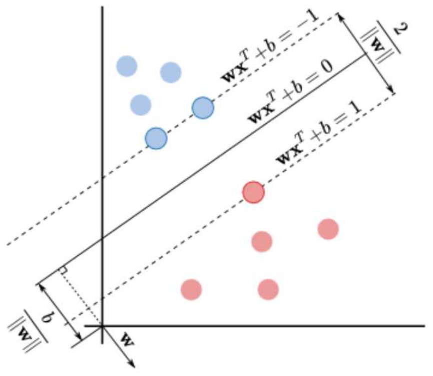


Fig. 2.2. SVM diagram. The blue and red dots represent the data samples belonging to each class. The straight line represents the hyperplane, while the dashed lines represent the support vectors.

where $\mathbf{w}$ is the normal vector of the hyperplane, n refers to the number of samples in the dataset and b corresponds to a special parameter in SVMs, often called bias.

The label assigned to each new sample $\mathbf{x}_*$ is determined by the following function:

$$y_* = sgn(\mathbf{wx}_*^T + b) \tag{2.6}$$

where $sgn(a)$ returns $+1$ if a is positive, and -1 otherwise.

On the contrary, the soft-margin formulation allows linear SVMs to be used with non-linearly-separable data by introducing the hinge loss function ($L(\mathbf{w}, b) = max(0, 1 - y_i(\mathbf{wx}_i^T + b))$) with the error variable ζ_i:

$$\begin{aligned} & min\left(\frac{1}{2}\mathbf{ww}^T + C\sum_{i=1}^{n}\zeta_i\right) \\ \text{subject to} \quad & \\ & y_i(\mathbf{wx}_i^T + b) \geq 1 - \zeta_i \text{ and} \\ & \zeta_i \geq 0, \forall i \in \{1, 2, ..., n\} \end{aligned} \tag{2.7}$$

where C is the regularization parameter. If its value is large, having a small hinge loss will be more important than having large margins. Therefore, SVMs will reduce the margin of the hyperplane in order to classify as many training points as possible correctly. On the other hand, if the value of C is small, increasing margins will be more important than reducing the hinge loss. Thus, SVMs will assume some classification errors to have large margins. In the extreme case, when C is tiny, SVMs will behave similarly to the hard-margin case.

Equation (2.7) can be simplified by solving its Lagrangian dual:

$$
\begin{aligned}
&max \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j^T \right) \\
&\text{subject to} \\
&\sum_{i=1}^{n} \alpha_i y_i = 0 \text{ and} \\
&C \geq \alpha_i \geq 0, \forall i \in \{1, 2, ..., n\}
\end{aligned}
\tag{2.8}
$$

In this dual formulation, α_i can be found by means of quadratic programming methods (Joachims, 1998). Consequently, $\mathbf{w}$ and b can be calculated as follows:

$$
\begin{aligned}
\mathbf{w} =& \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \\
b =& y_B - \mathbf{w} \mathbf{x}_B^T
\end{aligned}
\tag{2.9}
$$

where $\mathbf{x}_B$ and y_B are the values for a sample on the boundary of the margin.

2.2.1.1 Kernel trick

When data is not linearly-separable in the original space, there might be some feature space $\mathcal{V}$ where a hyperplane can classify the data. Boser et al. (1992) proposed a mapping of the data to a higher dimensional space, called the *kernel trick*.

As mentioned in Section 2.1, a kernel function k can be defined as an inner product in some Hilbert space $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_{\mathcal{V}}$, given a feature map $\phi : \mathbb{R}^d \rightarrow \mathcal{V}$. Thus, replacing the dot product operations in Equation (2.8) with the kernel function k is equivalent to mapping the data to the feature space $\mathcal{V}$ and computing the SVM in such space, which allows non-linearly-separable classification problems to be solved. In addition, the quadratic programming problem required to find the optimal hyperplane is convex as long as the kernel function satisfies Mercer's condition (Burges and Crisp, 2000).

2.2.1.2 Kernel and parameter setting for SVMs

As explained in the previous section, the application of SVMs requires optimizing the weights (w) and bias (b), as well as setting the C parameter. Apart from these general parameters of SVMs, in kernel learning approaches, the kernel structure and its hyperparameters (Θ) must also be searched for. All these components are depicted in Figure 2.3.

In order to find the best kernel structure for a certain problem, all the components must be properly set. In Chapter 3 we describe the work done, as part of this dissertation, to analyze these components and their interplay.

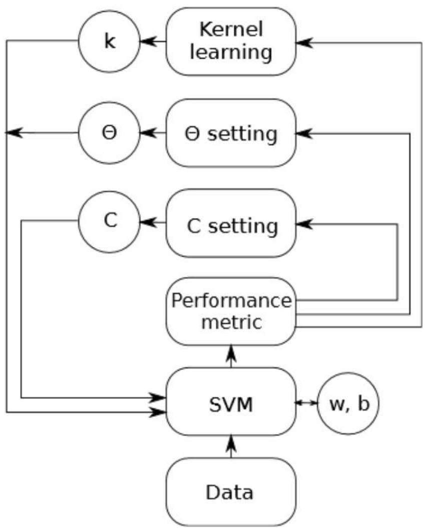


Fig. 2.3. Kernel search diagram for SVMs. The elements that take part in the kernel search are shown in rectangles, while the associated parameters are displayed in circles.

2.2.2 Gaussian Processes

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 (Rasmussen and Williams, 2006). 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 $cov\left(f(\mathbf{x}), f(\mathbf{x}')\right) = k(\mathbf{x}, \mathbf{x}')$ (Duvenaud, 2014). Given that, a GP can be expressed as follows:

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{2.10}$$

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 $\mathbf{f} = (f_1, f_2, ..., f_n)$ (where $f_i \in \mathbb{R}$, $i \in \{1, ..., n\}$ and $n \in \mathbb{N}$) and the test outputs $\mathbf{f}_* = (f_{n+1}, f_{n+2}, ..., f_{n+n_*})$ is given by:

$$\begin{bmatrix} \mathbf{f}^T \\ \mathbf{f}_*^T \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} M(X) \\ M(X_*) \end{bmatrix}, \begin{bmatrix} K(X, X) & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix} \right) \tag{2.11}$$

where $N(\mu, \Sigma)$ is a multivariate Gaussian distribution, $X = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$ ($\mathbf{x}_i \in \mathbb{R}^d$, $i \in \{1, ..., n\}$ and $n \in \mathbb{N}$) corresponds to the training inputs, and

$X_* = (\mathbf{x}_{n+1}, \mathbf{x}_{n+2}, ..., \mathbf{x}_{n+n_*})$ 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:

$$\begin{aligned} \mathbf{f}_*|X_*, X, \mathbf{f} &\sim \mathcal{N}(\widehat{M}(X_*), \widehat{K}(X_*, X_*)) \\ \widehat{M}(X_*) &= M(X_*) + K(X_*, X)K(X, X)^{-1}\left(\mathbf{f}^T - M(X)\right) \\ \widehat{K}(X_*, X_*) &= K(X_*, X_*) - K(X_*, X)K(X, X)^{-1}K(X, X_*) \end{aligned} \tag{2.12}$$

As in many previous works (Chu and Ghahramani, 2005; Brochu et al., 2010; Wang and de Freitas, 2014), we consider an a priori equal-to-zero mean function ($m(\mathbf{x}) = 0$).

2.2.2.1 Importance of the kernel function in GPs

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.

To illustrate the influence of both the covariance function and its hyperparameters, Figure 2.4 shows several GP models learned from the same input data by using different kernel functions and/or hyperparameters. It can be seen that the functions described by the SE kernel (bottom) are smoother than those drawn by the Matern32 kernel (top). Note also that the shape of the functions depends on the hyperparameters.

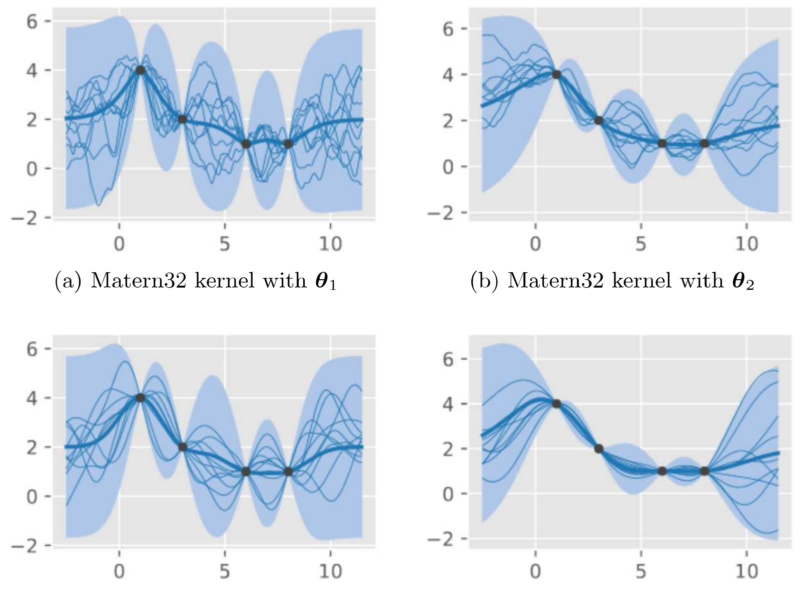
2.2.2.2 Kernel and parameter setting for GPs

Although a variety of methods have been also proposed to optimize the hyperparameters (Sundararajan and Keerthi, 2001; Toal et al., 2008, 2011; Garnett et al., 2014), the most common approach is to find the hyperparameter set that maximizes the Log Marginal Likelihood (LML):

$$\begin{aligned} L_{LML}(X, \mathbf{f}, \boldsymbol{\theta}) = log\, p\left(\mathbf{f}|X, \boldsymbol{\theta}\right) = -\frac{1}{2}\mathbf{m}_a K_a^{-1}\mathbf{m}_a^T - \frac{1}{2}log\,|K_a| - \frac{n}{2}\,log\,2\pi \\ with \\ \mathbf{m}_a^T = \mathbf{f}^T - M(X) \\ K_a = K(X, X) \end{aligned} \tag{2.13}$$

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 Rasmussen and Williams (2006). In this case, the likelihood of each



(a) Matern32 kernel with $\boldsymbol{\theta}_1$ (b) Matern32 kernel with $\boldsymbol{\theta}_2$

(c) SE kernel with $\boldsymbol{\theta}_1$ (d) SE kernel with $\boldsymbol{\theta}_2$

Fig. 2.4. GP model variations on the same input data (black dots) depending on the kernels and their hyperparameters. The Matern32 and SE kernels are shown with different hyperparameter values. The continuous blue curve represents the mean of the posterior GP, while the light blue shadow shows 3 times its standard deviation. The thin blue curves are 10 samples of the posterior GP.

sample of the training data is measured given the rest of the data. Then, these probabilities are added as follows:

$$\begin{aligned} L_{LOOCV}(X, \mathbf{f}, \boldsymbol{\theta}) &= \sum_{i=1}^{n} log\, p\,(f_i|X, \mathbf{f}_{-i}, \boldsymbol{\theta}) \\ log\, p\,(f_i|X, \mathbf{f}_{-i}, \boldsymbol{\theta}) &= -\frac{(f_i - \mu_i)^2}{2\sigma_i^2} - \frac{1}{2} log\, \sigma_i^2 - \frac{1}{2}\, log\, 2\pi \end{aligned} \tag{2.14}$$

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

The selection of the right set of hyperparameters is known to be a hard problem, particularly when few observations are available (Wang and de Freitas, 2014; Benassi et al., 2011; Bull, 2011). 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.

2.3 Proposals for kernel function design

In the early stages of SVM and GP research, the standard kernel functions introduced in Table 2.1 were applied (Mohandes et al., 2004; Zhou and Wang, 2005; Lin and Chen, 2011; Xu et al., 2019). In order to select the most suitable kernel among them, cross-validation techniques were used (Hussain et al., 2011). Similarly, decision trees were also proposed to find the most appropriate kernel depending on the characteristics of the problem (Ali and Smith-Miles, 2006).

Later, methodological advances allowed these standard kernels to be combined to create more complex structures. First, problem-specific kernel functions were manually developed, although, later, automatic techniques were proposed for learning these new kernels.

2.3.1 Composed ad hoc kernel functions

These proposals depart from the basis that there are certain kernel operations which guarantee that, if the source kernels are PSD, the result will also keep the positive semi-definiteness of its components (Duvenaud, 2014; Durrande et al., 2012). Here, some of these operations are presented:

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

These composed kernels can be also described by using a tree structure, as shown in Figure 2.5.

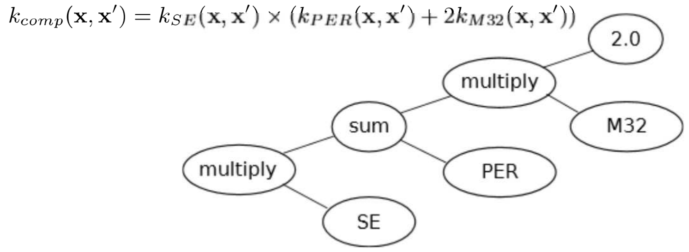

$k_{comp}(\mathbf{x}, \mathbf{x}') = k_{SE}(\mathbf{x}, \mathbf{x}') \times (k_{PER}(\mathbf{x}, \mathbf{x}') + 2k_{M32}(\mathbf{x}, \mathbf{x}'))$
2.0
multiply
sum
M32
multiply
PER
SE

Fig. 2.5. Composed kernel represented as a tree.

To illustrate the effect of the kernel composition, in Figure 2.6 we show the kernel function that results from the addition and product of a Linear kernel and a Periodic one. In the composed kernel functions, clear periodic patterns and decreasing slopes can be observed.

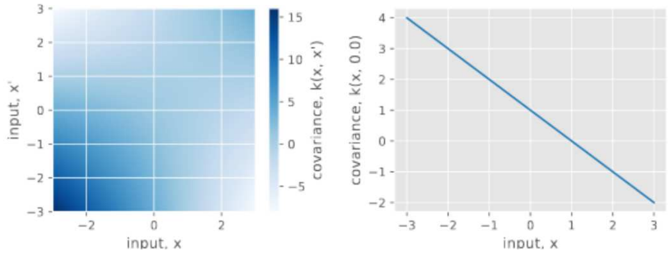
These basic operations can be used to create complex kernel functions by combining the standard kernels (Kronberger and Kommenda, 2013; Lloyd et al., 2014; Howley and Madden, 2005). In the GP literature, some authors use their expertise on the problem to design custom kernel functions based on these kernel composition techniques (Rasmussen and Williams, 2006; Preoţiuc-Pietro and Cohn, 2013). For example, Rasmussen and Williams (2006) introduced an ad hoc kernel 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 (this time series problem is discussed in detail in Section 5.2). Furthermore, Klenske et al. (2013) propose a product of a SE and a Periodic kernel to control periodic errors in astrophotography systems. Similarly, Preoţiuc-Pietro and Cohn (2013) designed an ad hoc kernel to predict the number of occurrences of certain hashtags in Twitter, given the past records. Finally, Wilson and Adams (2013) took advantage of the Bochner Theorem (Bochner, 1959) to design kernels that were able to model the periodical patterns of several time series.

2.3.2 Automatic kernel learning methods

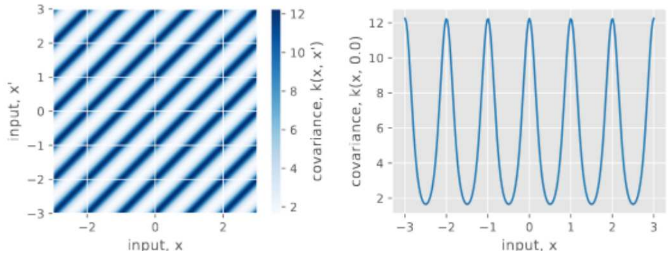
In order to achieve an AML approach, several search methods have been proposed for learning kernels with no human intervention. This is usually approached as a constructive procedure in which some kernel components are brought together in order to find the combination that provides the best performance. To do so, these approaches define a grammar that includes the modules to be used and the rules to combine them, as well as a search algorithm that defines the way the search is conducted.

Note that these grammars may include the kernel hyperparameters, which are usually optimized during the kernel learning process, as elements in the grammar. There might be some benefits in separating the hyperparameter learning from the structure search, as two different search methods can be used, one to learn the kernel structure and other to find the best hyperparameters. Thus, each search method can better exploit the regularities of each search space.

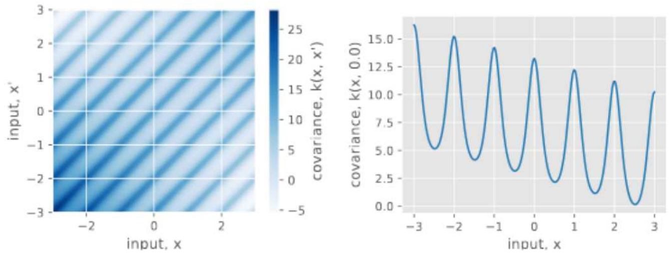
In the case of SVMs, it is desirable to obtain a Mercer kernel, as this condition ensures the convergence to the global optimum of the algorithm used to solve Equation (2.8) (Howley and Madden, 2005), while in GPs, the kernel function must be PSD in order to have closed formulas. Nevertheless, defining a grammar by means of which all Mercer kernels can be composed and all kernels that can be composed are Mercer is not an easy task. In fact, to the best of our knowledge, there is no proposal in the literature in this sense.



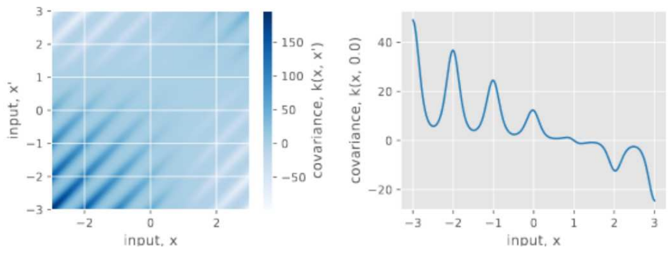
(a) Linear kernel $k_{LIN}(\mathbf{x}, \mathbf{x}')$



(b) Periodic kernel $k_{PER}(\mathbf{x}, \mathbf{x}')$



(c) $k_{LIN}(\mathbf{x}, \mathbf{x}') + k_{PER}(\mathbf{x}, \mathbf{x}')$



(d) $k_{LIN}(\mathbf{x}, \mathbf{x}') \times k_{PER}(\mathbf{x}, \mathbf{x}')$

Fig. 2.6. Example of a composed kernel function. On the left side, the outcome of the kernel is shown. On the right, the same function is shown when $x' = 0$.

Thus, automatic kernel learning methods can be classified depending on how they deal with Mercer's condition: kernel composition approaches and methods based on basic mathematical expressions. In this section, we describe the work done in the literature devoted to the study of both approaches.

2.3.2.1 Automatic kernel learning approaches based on kernel composition

These approaches are based on the kernel composition properties introduced in Section 5.8. The grammar is composed of a set of kernels like those shown in Table 2.1, along with the composition rules presented in Section 2.3.1. Thus, in these approaches, Mercer's condition is guaranteed by the grammar itself.

In addition, the kernels obtained by using these methods have shown their ability to capture function properties such as smoothness, trend and periodicity (Duvenaud et al., 2013; Lloyd et al., 2014). Furthermore, as the behavior of the standard kernels and the operators is well-known, the behavior of their compositions may be guessed by an expert (Lloyd et al., 2014).

On the contrary, although all the solutions created by means of kernel composition are guaranteed to be PSD, not all Mercer kernels can be created through this method. Furthermore, depending on the application domain, the search may end up with extremely complex structures that can be too cost-intensive to evaluate (Sullivan and Luke, 2007). Also note that these approaches rely on kernels that have already been proposed in the literature. There is no reason to assume that kernels obtained by composing a limited set of human-designed kernels are optimal for arbitrary problems. In fact, using previously designed kernels as building-blocks could bias the search and prevent the exploration of more promising candidates.

Genetic Programming (GenProg) (Koza, 1992) has been one of the most used methods to automatically find kernel combinations that improve the performance of standard kernels in SVM literature (Howley and Madden, 2005; Dioşan et al., 2008). GenProg is an evolutionary algorithm designed to search in a predefined space of computer programs, i.e., kernel functions in our case. Most of these GenProg approaches report some sort of accuracy gains over standard kernel functions (Howley and Madden, 2005; Koch et al., 2012).

In the GP literature, Kronberger and Kommenda (2013) proposed GenProg as a method for compositional kernel search. Nevertheless, the experimentation of this work was limited to the Mauna Loa Atmospheric CO_2 time series and some synthetic two-dimensional datasets. In addition to GenProg, other search methods have been proposed to search for kernel composition in GPs. Duvenaud et al. (2013) proposed a greedy search procedure, where the best kernel function in terms of Bayesian Information Criterion (BIC) was searched in the space of possible compositions (sums and products) of the standard kernels. Later, Lloyd et al. (2014) improved the previous approach by adding change-point and change-window kernels. Similarly, Malkomes et al.

(2016) used Bayesian Optimization, while Hinton and Salakhutdinov (2008) applied Deep Belief Nets to search in the model space. Finally, Deep Kernel Learning has been also proposed based on spectral mixture kernels (Wilson et al., 2016).

2.3.2.2 Basic mathematical expression based automatic approaches

The second type of approaches are based on basic mathematical expressions as building blocks. An example is shown in Figure 2.7, where the SE kernel is represented as a tree composed of basic mathematical expressions.

$$k_{SE}(\mathbf{x}, \mathbf{x}') = \theta_0^2 \ exp\left(-\frac{1}{2}\left\|\frac{\mathbf{x}}{\theta_1} - \frac{\mathbf{x}'}{\theta_1}\right\|^2\right)$$

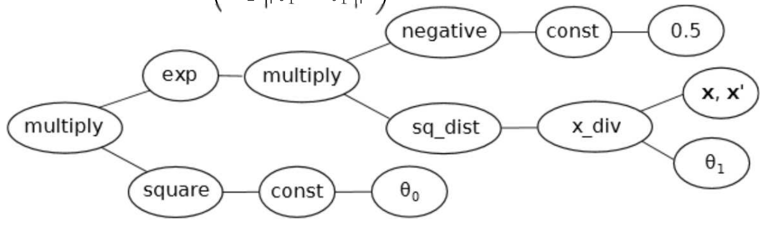


Fig. 2.7. SE kernel represented as a tree composed of basic mathematical expressions.

These proposals are more flexible and potentially better because they allow a richer and a wider set of kernels to be explored, built from scratch, without any previous bias. This types of grammars allow more compact kernel expressions to be found than those designed by an expert or automatically learned by composing kernels. In addition, note that a basic mathematical expression based grammar can be designed for every kernel composition grammar, where all the kernels that can be created from the former are a superset of all the kernels that can be composed with the latter.

In contrast, the wider search space derived from the basic mathematical expression based approaches may hinder the task of finding good-performing kernels.

Another undesirable downside of being able to compose more compact and flexible kernels is that these methods may generate non-Mercer kernels during the search. The simplest way to deal with this problem is not to guarantee that kernels meet Mercer's condition (Bing et al., 2010; Howley and Madden, 2005; Gagné et al., 2006; Thadani et al., 2006). Although, this option is not valid for GPs, it can be used for SVMs, assuming that the optimization algorithm used to find the optimal hyperplane may not converge to the global optimum. Other proposals in the literature (Howley and Madden, 2006; Dioşan et al., 2007; Koch et al., 2012) check Mercer's condition for every kernel and those that do not meet this condition are penalized or discarded. Howley and Madden

(2006), and Dioşan et al. (2007), propose a method to penalize (assigning the worst possible fitness to them) the non-PSD kernels in evaluation time, while Koch et al. (2012) suggest discarding the non-PSD kernels and repeat the kernel creation process. Although these methods have been proposed for SVMs, they are also applicable for GPs.

Part II

Methodological Contributions

3

An analysis of SVM kernel learning

3.1 Introduction

Selecting the kernel function that will be used is an important choice in the SVM setting discussed in Section 2.2.1.2. Towards an AML approach of SVMs, many research outputs have been produced dealing with the challenge of automatic learning of good-performing kernels. However, these works have been carried out without a thorough analysis of the set of components that influence the behavior of SVMs and their interaction with the kernel. These components are related in an intricate manner and it is difficult to provide a comprehensible analysis of their joint effect. This chapter is devoted to filling this gap introducing the necessary steps in order to understand these interactions and provide clues for the research community to know where to place the emphasis.

To start with, a search space which contains all (and only) the Mercer kernels has not been described yet. Instead, previously proposed methods pose some sort of limitations in the search space of kernels, whether only considering a subset of all the Mercer kernels or also including some which are non-Mercer. Several challenges have to be dealt with in relation to this topic: How does the selected search space of kernels influence the results of SVMs? According to the characteristics of the kernels, which are the regions of the search space on which the search efforts should focus?

Once the space of possible kernels has been defined, the next relevant question is the selection of a strategy to carry out the search. Most of the works in the literature have proposed various heuristic algorithms to solve this search problem, GenProg being one of the most used (Howley and Madden, 2006; Dioşan et al., 2012; Koch et al., 2012). However, there is a lack of knowledge about many aspects related to the specific characteristics of the kernel function optimization problem, and in particular, about how these characteristics relate to the way the GenProg search for optimal solutions is accomplished. Furthermore, there is no clear understanding of the relative performance of GenProg compared to other simpler search strategies, since other optimization methods have rarely been applied to this problem. Relevant questions in this area are: What is the relevance of the search method with respect to the characteristics of the chosen search space? Which characteristics should a search algorithm have in order to efficiently explore the kernel space?

Apart from the choice of the kernel, there are other components of SVMs that interact in a complex manner, which hinders the identification of the essential elements that are necessary to obtain a good performance in the classification task. However, in most of the previous works little attention has been paid to the rationale behind the choice of those components of SVMs and how these choices influence the dynamics and results of the kernel search.

The learning of the kernel itself is often divided into the kernel structure search and the tuning of its hyperparameters. This tuning process is one of the steps involved in the SVM learning, whose essential role is usually overlooked in the literature. The key questions are: What is the relevance of finding the right hyperparameters? Which is the best method for finding them? How much computational effort needs to be used to optimize the hyperparameters?

Beyond the setting of the kernel and its hyperparameters, the commonly used flexible variant of SVMs has its own parameter (C), whose role is to deal with the overfitting of the model. Although the choice of C strongly influences the effectiveness of the final classifier, it is not clear in the literature what the interactions of this parameter are with the rest of the components. For instance, what is the contribution of the C parameter to the performance of SVMs with a particular choice of the kernel? Which is the interplay between the kernel hyperparameter setting and the C parameter setting?

Finally, for an automated kernel search, we not only need to assess the quality of the solution on the training data but also to implicitly capture how it will generalize to new data. If a wrong evaluation measure is chosen, then, an apparently good solution (in terms of the measure) may overfit the data and produce poor results at the prediction stage. The choice of the objective function used to evaluate the quality of the kernel also has an impact on the roughness of the kernel search space, and therefore on the performance of the search methods. Most of the previous works in the literature have used the classifier accuracy as the metric of choice. However, are there better metrics to guide the search for optimal kernels?

Trying to shed some light on these issues, in this chapter we analyze the components involved in the structural learning of kernels. We start by considering each component independently, and we then proceed by addressing the way they interact to influence the behavior of SVMs. In the study of these components and their interactions, we introduce some guidelines to improve the performance of SVMs.

3.2 The search space

Depending on the classification problem that is being solved by means of SVMs, some elements of the grammar might be crucial. However, most of the kernel learning approaches do not single out the grammar nor the choice of the elements that compose it as relevant issues.

3.2.1 Relevance of periodic elements

In order to illustrate the importance of including the appropriate elements in the grammar, we compare the classic RBF kernel[1] to the Periodic kernel in a subset of well-known problems. The RBF kernel is known for capturing the smoothness property of the data, as elements close to each other in terms of Euclidean distance have a high kernel value and it smoothly decreases as the distance increases. The elements needed to compose the RBF kernel are present in most of the grammars reported in the kernel learning literature, regardless of the grammar type choice, whether they use kernel composition or are based on basic mathematical expressions. On the other hand, the Periodic kernel is based on the RBF kernel, however it adds a spectral transformation to the space (HajiGhassemi and Deisenroth, 2014) in order to model periodic patterns in the data. Periodic elements, such as the spectral transformation, have been overlooked in the kernel search literature, and only some approaches (Bing et al., 2010) include them in their grammars.

If the Periodic kernel obtains a better result than the RBF, it may indicate that some periodic patterns are present in the data. If so, having the spectral transformation in the grammar is essential to achieve good results.

Although some of the kernel search approaches have been used to solve particular problems, most works in the literature test their proposals in the UCI classification datasets (Dua and Graff, 2017), shown in Table 3.1, in order to compare their results with those reported in previous works.

Classification problem	Samples	Variables	Classes
pima	768	8	2
ionosphere	351	34	2
heart statlog	270	13	2
glass2	163	9	2
liver disorder	345	6	2
breast cancer Wisconsin	569	30	2

Table 3.1. Characteristics of the UCI problems studied in this work.

In these UCI datasets, the RBF kernel has a reasonably good performance, close to the state-of-the-art kernels created by composition (Dioşan et al.,

[1] In this section we refer to the SE kernel as RBF, as it is the name most commonly used in the SVM literature.

2012). It might indicate that modeling the smoothness property of the data is enough to achieve good classification results (Duvenaud et al., 2013).

To widen the scope of the analysis, we searched for other types of classification problems, where new kernel properties, apart from smoothness, could be necessary to obtain more accurate results. In a preliminary experiment, we used the Penn Machine Learning Benchmarks (PMLB) (Olson et al., 2017) to find datasets where RBF does not perform so well, possibly indicating that other kernel properties are needed.

After evaluating the SVMs with the RBF kernel in all the PMLB databases, we selected the 8 problems where the lowest accuracy was obtained. The characteristics of these datasets are shown in Table 3.2. In the non-binary PMLB problems, the one-vs-one approach was used.

Classification problem	Samples	Variables	Classes
calendarDOW	399	31	5
contraceptive	1473	9	3
GAMETES Epistasis 0.1H	1600	19	2
GAMETES Epistasis 0.4H	1600	19	2
GAMETES Heterogeneity 50	1600	19	2
GAMETES Heterogeneity 75	1600	19	2
parity5+5	1124	10	2
Hill Valley with noise	1212	100	2

Table 3.2. Characteristics of the PMLB problems for which the RBF kernel obtained the worst accuracies.

In order to compare the results of the RBF kernel to the Periodic kernel (PER), we ran a second experiment with the Periodic and RBF kernels in the previously shown UCI and PMLB databases.

The dataset was partitioned twice. A random fold of 20% of the data is selected as the test set, and a 4-fold cross-validation was used to set C and the hyperparameters for each kernel. The SVMs were fitted in each fold of the training set for each combination of C (2^{-5} to 2^{15}, at powers of 2^2 as proposed by Sousa et al. (2017)) and the hyperparameters (2^{-5} to 2^4, at powers of 2), with a limit of 1000 evaluations. Next, the combination with the best average accuracy was selected to be evaluated in the test set.

In Table 3.3 the results of this experimentation are shown. Although in the UCI datasets the results of both kernels are similar, important performance gains can be obtained in the *GAMETES* PMLB datasets when using the periodic kernel instead of the RBF.

In spite of being very similar kernels, there are remarkable performance differences between the RBF and Periodic kernels depending on the database. The limited capacity of the RBF kernel to model the *GAMETES* databases restricts the classification performance of the SVMs. This suggests that including the elements of the Periodic kernel in the kernel search grammar is

	Classification problem	RBF	PER
UCI	pima	**0.772**	0.758
	ionosphere	0.939	**0.944**
	heart statlog	0.826	**0.837**
	glass2	**0.809**	0.773
	liver disorder	**0.743**	0.726
	breast cancer Wisconsin	0.972	**0.974**
PMLB	calendarDOW	0.621	**0.624**
	contraceptive	0.547	**0.548**
	GAMETES Epistasis 0.1H	0.562	**0.668**
	GAMETES Epistasis 0.4H	0.709	**0.797**
	GAMETES Heterogeneity 50	0.660	**0.714**
	GAMETES Heterogeneity 75	0.669	**0.701**
	parity5+5	**0.931**	0.905
	Hill Valley with noise	**0.815**	0.801

Table 3.3. Mean accuracies in the test set for the RBF kernel and the Periodic kernel in UCI and PMLB classification problems. The numbers in bold indicate the best result for each problem. UCI databases are shown in the top 6 rows of the table, while PBML problems are at the bottom of the table.

crucial in some problems, and highlights the importance of a careful selection of the elements that compose this grammar.

3.2.2 Proposed grammar

In order to investigate the importance of the selected search space, and taking into account the grammars proposed in the literature, we designed a grammar based on basic mathematical expressions by means of which all the kernels of Table 2.1 can be composed. The production rules of this grammar are shown in Table 3.4.

The *scalar* non-terminal is the start symbol of the grammar. It also includes the $+$, $\times$, and $\hat{}$ arithmetic operators, with their usual meanings (addition, product and power, respectively). Note that we only allow hyperparameters as the exponent in the power operator. The same interpretation is given to the unary operators. The power to the minus one is also added as a unary operator in order to allow division operations. Then, the input vectors are converted into scalars by means of the square distance and dot product non-terminals, as described in Section 2.1.1. Similarly, constant and noise non-terminals, whose values depend on the input hyperparameter, are included. The subtraction and the division of an input vector by a hyperparameter are also incorporated. In addition, the grammar also contains the spectral transformation, in order to allow periodic kernels, as suggested by HajiGhassemi and Deisenroth (2014). Finally, $(\mathbf{x}, \mathbf{x}')$ (the input vectors of the kernel) and θ (the hyperparameters) are the terminals of this grammar.

$kernel :$	$scalar$	start symbol
$scalar :$		
	$\mid scalar^{hp}$	power
	$\mid scalar + scalar$	add
	$\mid scalar \times scalar$	multiply
	$\mid scalar^{-1}$	div
	$\mid e^{scalar}$	exp
	$\mid \sqrt{scalar}$	sqrt
	$\mid scalar^2$	square
	$\mid -scalar$	negative
	$\mid tanh(scalar)$	tanh
	$\mid \lVert \mathbf{invec} - \mathbf{invec}' \rVert^2$	sq_distance
	$\mid \mathbf{invec}.\mathbf{invec}'^{T}$	dot_product
	$\mid hp$	constant
	$\mid hp \times \delta(\mathbf{x}, \mathbf{x}')$	noise
	;	
$(\mathbf{invec}, \mathbf{invec}') :$		
	$\mid \left(\left[sin(\mathbf{invec})\ cos(\mathbf{invec})\right], \left[sin(\mathbf{invec}')\ cos(\mathbf{invec}')\right]\right)$	spectral
	$\mid \left(\frac{\mathbf{invec}}{hp}, \frac{\mathbf{invec}'}{hp}\right)$	x_div
	$\mid (\mathbf{invec} - \mathbf{1}hp, \mathbf{invec}' - \mathbf{1}hp)$	x_rest
	$\mid (\mathbf{x}, \mathbf{x}')$	input
	;	
$hp :$		
	$\mid 0.5 \mid 1 \mid 2 \mid 3 \mid 5$	
	$\mid \theta_0 \mid \theta_1 \mid ... \mid \theta_t$	
	;	

Table 3.4. Proposed grammar for SVMs. t indicates the number of different hyperparameters allowed in the grammar (in this work, it is set to $t = 20$).

3.2.2.1 Random kernel generation

In order to randomly generate kernel expressions, we propose a strongly-typed grow method based on the work presented by Koza (1992). 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 3.4, 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.

3.2.2.2 Dealing with non-Mercer kernels

In order to mitigate the evaluation of non-Mercer kernels, we followed the approach used by Koch et al. (2012): 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 Mercer kernel has to be symmetric and also PSD. To identify non-PSD kernels, we generate w random uniformly distributed datasets $X = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$ (where $\mathbf{x}_i \in \mathbb{R}^d$, $i \in \{1, ..., n\}$ and $n \in \mathbb{N}$) and check the M matrix produced by the kernel for each dataset. If any M matrix matches the following cases, the generation process of the kernel is repeated:

- $M \neq M^T$: As previously mentioned, the matrix given by a Mercer kernel should be symmetric.
- Any m_{ii} is negative: It has been proved (Zhang, 2011) that, if any of the elements in the main diagonal are negative, the matrix is not PSD.
- Any of the eigenvalues of M is negative: Similarly, all the eigenvalues of the matrix should be non-negative.

With this method, and using the grammar shown in Table 3.4, 9 trials were required on average in the experiments conducted in this chapter to create a kernel that generates a PSD matrix. Note that meeting these conditions is not sufficient for a kernel to be Mercer. Although some non-Mercer kernels may pass the PSD check, it was not a problem during the experimentation, as the SVM optimization did not converge in only 3.16% of the kernels that passed the PSD check.

3.2.3 Increasing grammar experiment

Following the experiment introduced in Section 3.2.1, we conducted a more in-depth experiment to observe the influence of the grammar in the search of the best kernel for the SVM classification. Particularly, we wanted to measure the effect that the addition of certain elements to the grammar has in randomly generated kernels and in the performance of the SVMs that use these kernels.

In order to carry out this experiment, we obtain a series of grammars that are able to create kernels that can generate PSD matrices and each grammar in the sequence comprises all the elements from the previous one. We start with the minimum possible grammar, including the input vector and the dot product operator from which only a simplified version of the Linear kernel can be created. To create the next grammar, we add one random element to the first one, and test whether a random kernel generated with this new grammar, which contains this new element, passes the PSD check. $d_{min} = 5$ and $d_{max} = 15$ were set to control the depth of the generated expressions. If the randomly generated kernel fails the test, we try adding another element to the grammar. If, after testing all the elements, none of the new grammars is able to generate a random kernel that passes the PSD check, we try to add pairs of elements to the grammar. On the other hand, if the new grammar is able to generate random kernels that pass the PSD check, this new grammar is added to the sequence, and the process is repeated adding a new element to it. We repeat this experiment obtaining 10 series of 27 grammars for each UCI and PMLB database. Next, for each grammar in the sequence, we randomly generate 18 kernels and evaluate them. The evaluation of these kernels is carried out following the same setting used in Section 3.2.1.

Figure 3.1 illustrates one of these experiments in the *GAMETES Epistasis 0.4H* database. It can be seen that, when including certain elements in the grammar, the accuracy of SVMs increases. For example, when the *spectral* non-terminal is included, in combination with the exponential, 60% accuracy can be achieved. Moreover, in the 16th iteration, the inclusion of the *multiplication* non-terminal improves the accuracy up to 74%. In the following iterations, the *sq_distance* non-terminal slightly increases the performance when selected. Overall, the performance of the best kernel of each iteration (created by means of a richer grammar) shows an increasing trend.

To obtain a general view of the experiment, in Figure 3.2 the average accuracy for each iteration and database is shown. In some problems, such as *pima*, *heart statlog* and *breast cancer Wisconsin* the results of the first iteration, i.e., the simplified Linear kernel, can not be improved with the addition of new elements. This is consistent with the results of Table 3.3, where the rest of the standard kernels barely outperform the results of the Linear kernel. However, for the rest of the problems, there is a clear increase in accuracy when new elements are added to the grammar. This is especially visible in the *glass2*, *parity5+5* and *GAMETES* databases.

In summary, this experiment shows that including a wide set of elements in the grammar is beneficial for any kernel structure search attempt. If some of these elements are missing, the performance of SVMs is limited, regardless of the number of evaluations or the search method. Thus, framing an appropriate search space is essential to achieve good results. Our first recommendation would be to include as many elements described in the literature as possible. Particularly, we have observed in the experiment of Section 3.2.1 that the spectral element is essential for some particular problems. A possible drawback derived from a very rich grammar would be a wider search space, which makes it more difficult for a search algorithm to find a good performing combination (kernel). However, as discussed in Section 3.3, even a basic random search algorithm with a limited budget is able to provide competitive results.

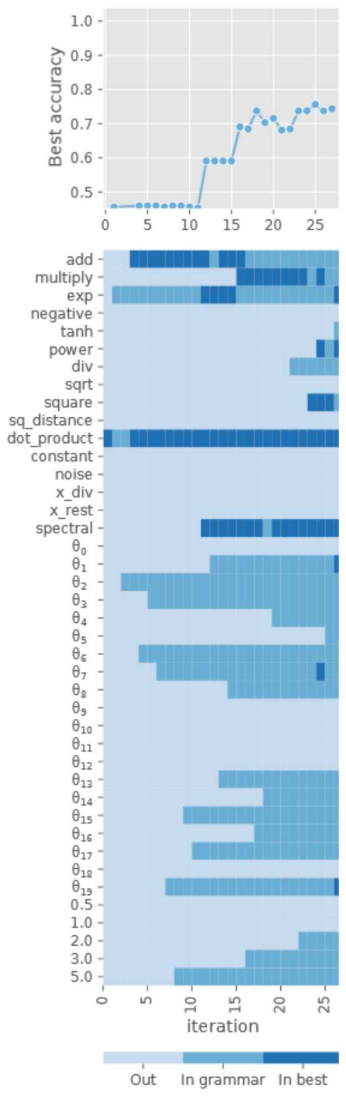
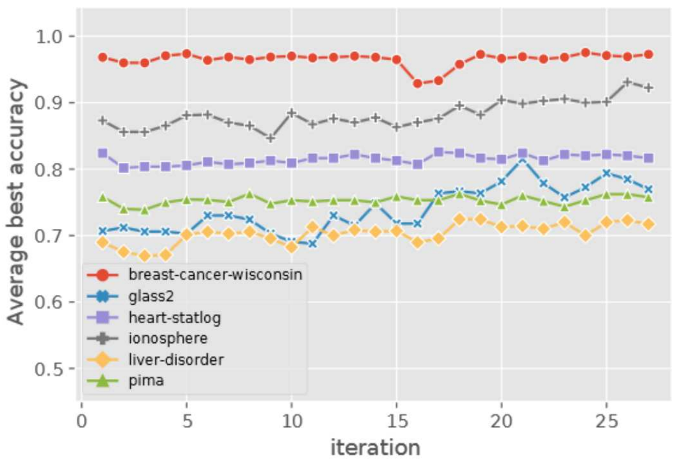
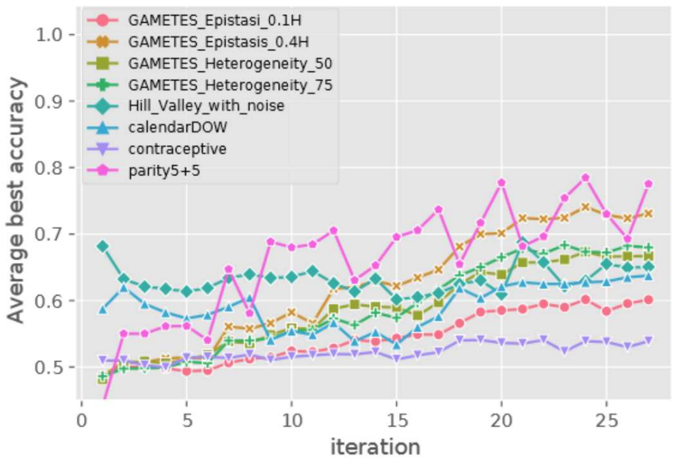


Fig. 3.1. Increasing grammar experiment in the *GAMETES Epistasis 0.4H* dataset. The figure at the bottom shows the elements present in the grammar at each iteration. The lightest blue color indicates that an element is out of the grammar, while the darker blue color expresses that this element is included. If the best kernel of that iteration contains a certain element the darkest blue color is shown. At the top, the accuracy on the test set of the best random kernel (selected according to the training set) is plotted.



(a) UCI



(b) PMLB

Fig. 3.2. Average accuracy on the test set of the best kernel (selected according to the training set) at each iteration.

3.3 Kernel structure search

Once the importance of the grammar and the search space has been introduced, we then focus on the kernel structure search step. For this purpose, we designed an experiment to compare the performance of GenProg with other kernel structure search approaches, also including the standard kernels shown in Table 2.1, for the UCI and PMLB databases.

The GenProg method studied in this experiment is based on the mathematical expression grammar introduced in Section 3.2.2. As shown in Algorithm 1, in this approach, an initial population of N random kernels is generated with a minimum (d_{min}) and a maximum depth (d_{max}), and evaluated. After selecting the S best individuals, the algorithm randomly chooses between a mutation or a crossover operator (with probability p_m and p_{cx} respectively, where $p_{cx} = 1 - p_m$) to generate an offspring population of N new individuals. After evaluating all the individuals in this offspring population, the previously selected individuals are added to generate the next population that consists of $N + S$ individuals. This procedure is repeated for G generations, until the last population is evaluated and the best individual found during the whole process is returned.

Algorithm 1 GenProg algorithm for SVM kernel structure search

```
 1: procedure GenProg($N$, $G$, $S$, $p_m$, $p_{cx}$, $\beta$, $d_{min}$, $d_{max}$)
 2:     $pop$ = GenRandPop($N$, $d_{min}$, $d_{max}$)
 3:     Evaluate($pop$)
 4:     $all = pop$
 5:     $i = 0$
 6:     while $i < G - 1$ do
 7:         $sel$ = Select($pop$, $S$)
 8:         $offspring$ = Variate($sel$, $N$, $p_m$, $p_{cx}$)
 9:         Evaluate($offspring$)
10:         $all = all \cup offspring$
11:         $pop = sel \cup offspring$
12:         $i = i + 1$
13:     end while
14:     $best$ = Select($all$, 1)
15:     return $best$
16: end procedure
```

In order to assess the contribution that each component of the proposed GenProg algorithm makes to the kernel search, we introduce three algorithms to be used as a baseline in the experiments.

First, we describe a random search algorithm that generates N kernels following the method described in Section 3.2.2.1. Then, the best solution is chosen according to the cross validated accuracy in the training set.

Secondly, in order to measure the gain produced by the crossover operator in the GenProg setting, we use a hill-climbing algorithm (Davis, 1991), which does not depend on this operator. This procedure generates an initial kernel, from which a second kernel is created by applying a random mutation. Next, the best kernel in terms of accuracy is selected. This procedure is repeated for N evaluations.

Finally, we also introduce a GenProg variant without the spectral element in the grammar (sGenProg), in order to develop the results of the experiment shown in Section 3.2.1.

In all these approaches, before the evaluation of every kernel, the hyperparameters and C were optimized as in the experiment in Section 3.2.1. In the kernel structure search approaches, the hyperparameters were optimized in a grid of 2^{-5} to 2^{4}, at powers of 2, with a limit of 1000 evaluations. For a fair comparison, we used a more exhaustive grid search to find the hyperparameters for the standard kernels: 2^{-5} to 2^{4}, at powers of $2^{0.1}$, with a limit of 486000 evaluations. In the GenProg approach, in each of the $G = 27$ generations, $N = 18$ kernels were created and the best $S = 4$ kernels were chosen as seeds for new individuals. The mutation and crossover probabilities were set to $p_m = 0.4$ and $p_{cx} = 0.6$ respectively. Similarly, $N = 486$ evaluations were carried out in the random search and hill-climbing methods. Each configuration was repeated 10 times.

As can be seen in Table 3.5, the GenProg approach improves the training set results of the standard kernels in all the UCI and PMLB databases, except in the *GAMETES* problems, where it is not able to achieve the same accuracy as the Periodic kernel. Notable performance gains were achieved in the *glass2*, *parity5+5* and *Hill Valley with noise* problems by using the GenProg method. The hill-climbing and the random search methods obtain results similar to the best standard kernel in most of the databases. Comparing GenProg to the random search and the hill-climbing methods, the best average accuracies are achieved by GenProg.

The results obtained in the training set by the different models are used to determine the best performing model in the test set. However, the performance of the models changes when applied to the test set, probably due to the overfitting effect. Thus, in spite of obtaining good accuracy values in the training set, the GenProg approach is not able to maintain those results in the test set. Table 3.6 shows that this issue is clearly visible in most databases where slight differences were observed in the training set between the GenProg and the best standard kernels. However, in other datasets, such as *glass2*, *parity5+5* and *Hill Valley with noise*, the GenProg kernel learning method is clearly a better choice in the training set, and these results are also visible in the test set.

In the *contraceptive* and *GAMETES Epistasis 0.4H* databases, GenProg achieves better average accuracy values in the test set than the other kernel structure search methods. In some other problems, such as *pima*, *heart statlog*, *liver disorder* and *GAMETES Epistasis 0.4H* problems, the exploitation oriented mutation operator of the hill-climbing algorithm generates the best kernels for the test set. On the other hand, the exploration oriented behavior of the random search seems to be less prone to overfitting, achieving the best results in the *breast cancer Wisconsin*, *calendarDOW* and two *GAMETES Heterogeneity* problems.

	Classification problem	LIN	M32	M52	RBF	PER	Random	HC	sGenProg	GenProg
UCI	pima	0.779	0.781	0.782	0.783	0.785	0.788	0.788	0.793	**0.793**
	ionosphere	0.889	0.956	0.957	0.956	0.958	0.955	0.958	**0.962**	**0.962**
	heart statlog	0.859	0.859	0.859	0.860	0.869	0.876	**0.878**	**0.878**	**0.878**
	glass2	0.715	0.809	0.813	0.813	0.855	0.860	0.862	0.836	**0.909**
	liver disorder	0.697	0.737	0.742	0.743	0.747	0.745	0.750	0.756	**0.763**
	breast cancer Wisconsin	0.981	0.984	0.984	0.983	0.984	0.984	0.985	0.986	**0.986**
PMLB	calendarDOW	0.592	0.622	0.625	0.627	0.651	0.651	0.660	0.642	**0.665**
	contraceptive	0.516	0.564	0.565	0.565	0.572	0.569	0.570	0.569	**0.573**
	GAMETES_Epistasi_0.1H	0.501	0.576	0.578	0.582	**0.684**	0.679	0.680	0.596	0.681
	GAMETES Epistasis 0.4H	0.503	0.678	0.686	0.690	**0.794**	0.791	0.793	0.700	0.793
	GAMETES Heterogeneity 50	0.502	0.641	0.647	0.650	**0.716**	0.704	0.706	0.667	0.709
	GAMETES Heterogeneity 75	0.517	0.659	0.664	0.665	**0.736**	0.725	0.732	0.673	0.733
	parity5+5	0.497	0.555	0.563	0.632	0.722	0.957	0.938	0.990	**0.999**
	Hill Valley with noise	0.866	0.833	0.828	0.820	0.835	0.796	0.817	0.886	**0.905**

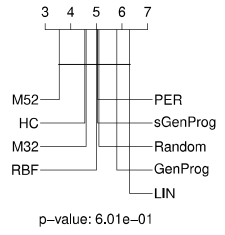
Table 3.5. Results of the Kernel search experiment in the training set. The mean accuracy achieved by each kernel search method is shown. The numbers in bold indicate the best result for each problem. sGenProg indicates the spectral-less GenProg approach.

	Classification problem	LIN	M32	M52	RBF	PER	Random	HC	sGenProg	GenProg
UCI	pima	0.759	0.766	**0.773**	0.766	0.765	0.762	0.763	0.756	0.759
	ionosphere	0.875	0.942	0.945	0.942	0.944	0.945	0.945	0.944	**0.946**
	heart statlog	**0.830**	0.820	0.824	0.820	0.815	0.806	0.811	0.815	0.802
	glass2	0.710	0.818	0.800	0.794	0.800	0.773	0.803	0.776	**0.836**
	liver disorder	0.684	0.722	0.732	**0.735**	0.713	0.712	0.719	0.714	0.719
	breast cancer Wisconsin	0.973	0.969	0.971	0.970	0.971	**0.975**	0.969	0.972	0.973
PMLB	calendarDOW	0.577	0.620	0.621	0.623	0.619	**0.626**	0.619	0.609	0.616
	contraceptive	0.521	**0.555**	0.553	0.550	0.548	0.547	0.547	0.553	0.552
	GAMETES_Epistasi_0.1H	0.467	0.561	0.562	0.559	0.675	**0.676**	0.675	0.560	0.675
	GAMETES Epistasis 0.4H	0.489	0.697	0.708	0.717	**0.797**	0.796	0.796	0.722	0.796
	GAMETES Heterogeneity 50	0.482	0.648	0.651	0.653	**0.721**	0.719	0.719	0.665	0.712
	GAMETES Heterogeneity 75	0.488	0.665	0.670	0.668	**0.720**	0.714	0.713	0.672	0.708
	parity5+5	0.474	0.542	0.718	0.892	0.865	0.985	0.965	0.998	**1.000**
	Hill Valley with noise	0.819	0.849	0.841	0.825	0.837	0.786	0.818	0.865	**0.910**

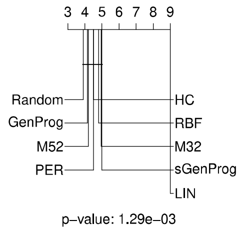
Table 3.6. Results of the Kernel search experiment in the test set. The mean accuracy achieved by each kernel search method is shown. The numbers in bold indicate the best result for each problem. sGenProg indicates the spectral-less GenProg approach.

Besides, the GenProg approach with the spectral element in the grammar shows a better performance than the spectral-less variant in the training set in all the problems, especially in the *glass2*, *calendarDOW*, *Hill Valley with noise* and *GAMETES* problems. In Table 3.6, it can be seen that these differences are also notable in the test set. These problems probably include some periodic patterns that can be better modeled when the spectral element is present.

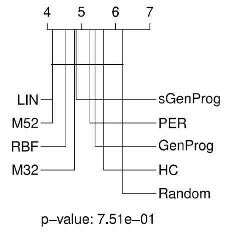
We conducted a statistical test to assess the existence of significant differences among the methods in the test set. For each database, we applied Friedman's test (Friedman, 1937) and we found significant differences ($\alpha = 0.05$) in the *ionosphere*, *glass2*, *contraceptive*, *parity5+5* and all *GAMETES* databases (p-values can be seen in Figures 3.3 and 3.4). Then, for each configuration, we applied a post-hoc test based on Friedman's test as in the work done by Demšar (2006), and adjusted the results with Shaffer's correction (Shaffer, 2012). The results are shown in Figures 3.3 and 3.4, and in Table 3.7, where a summary of the statistical tests is presented.



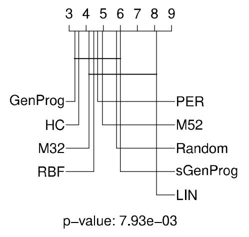
(a) pima



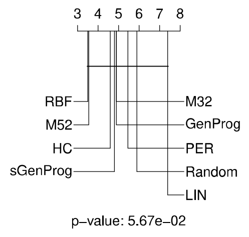
(b) ionosphere



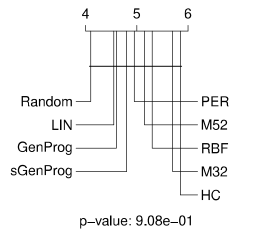
(c) heart statlog



(d) glass2

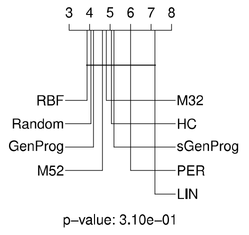


(e) liver disorder

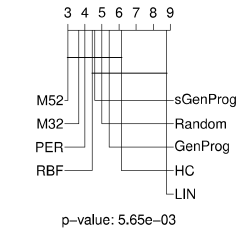


(f) breast cancer Wisconsin

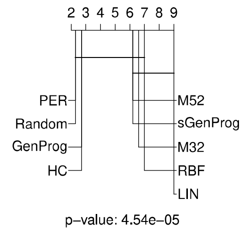
Fig. 3.3. Critical difference diagrams in UCI datasets. Search methods are ordered following their rankings. The methods with no significant differences among them are matched with a straight line.



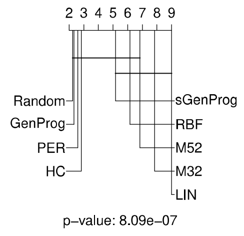
(a) calendarDOW



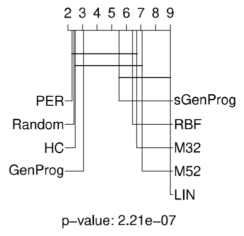
(b) contraceptive



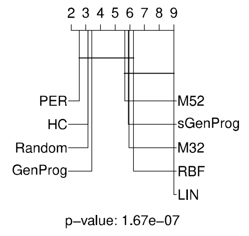
(c) GAMETES Ep. 0.1H



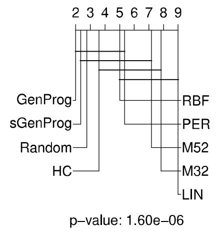
(d) GAMETES Ep. 0.4H



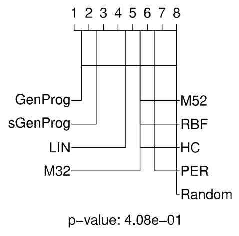
(e) GAMETES Het. 50



(f) GAMETES Het. 75



(g) parity5+5



(h) Hill Valley with noise

Fig. 3.4. Critical difference diagrams in PMLB datasets. Search methods are ordered following their rankings. The methods with no significant differences among them are matched with a straight line.

Overall, the GenProg method is the best performing approach, obtaining significantly better results than the Linear (LIN), Matern32 (M32) and Matern52 (M52) kernels in some problems. On the contrary, there are not many statistical differences between the structure search methods. Among the standard kernels, the periodic kernel shows significantly better results than the Linear kernel in 6 databases, and improves the performance of Matern32 in *GAMETES Epistasis 0.4H*, and Matern52 in *GAMETES Heterogeneity 50*.

	LIN	RBF	M32	M52	sGenProg	PER	HC	Random	GenProg	Worse
LIN	0	1	2	2	2	6	7	6	7	33
RBF	0	0	0	0	0	0	0	0	0	0
M32	0	0	0	0	1	1	1	2	2	7
M52	0	0	0	0	0	1	0	1	1	3
sGenProg	0	0	0	0	0	0	0	0	0	0
PER	0	0	0	0	0	0	0	0	0	0
HC	0	0	0	0	0	0	0	0	0	0
Random	0	0	0	0	0	0	0	0	0	0
GenProg	0	0	0	0	0	0	0	0	0	0
Better	0	1	2	2	3	8	8	9	10	

Table 3.7. Summary table of the statistical testing. The number of databases where the method in the column is significantly better than the method in the row is shown.

Although the GenProg approach improves the results of the standard kernels in the training set, and it has a better exploration-exploitation balance than random search and hill-climbing, these results cannot be transferred to the test set, probably due to overfitting issues. In the test set, there are no significant differences between the GenProg approach and the simpler kernel structure search methods. It is also important to notice that a single change in the grammar can produce a greater impact in the results than the search method itself, as in the *GAMETES* problems, where the average differences between the results of the GenProg with and without the spectral element are higher than the gap between the hill-climbing and the standard GenProg approach.

As a final note, we can question the importance of the kernel search method compared to the importance of selecting an appropriate search space. According to the experiments, the GenProg method shows the best results. Nevertheless, the absence of statistical differences with the random search suggests that the efforts of the practitioners should focus on the design of an adequate search space rather than on the design of the best possible search algorithm. It is also worth mentioning the small differences we found between the training and test results. Not having a measure of complexity of the models in the kernel learning approaches has probably generated models that are too dependent on the training set.

3.4 Hyperparameter and C optimization

We have shown that in SVMs there are several variables to optimize apart from the kernel structure, such as the kernel hyperparameters and the C parameter. Hyperparameters, being part of the kernel, change the transformed space, while the C parameter balances the trade-off between increasing the margin and assuming greater hinge loss. In this section, we review the literature about the C parameter and hyperparameter setting and investigate the interplay of these variables for several kernels.

3.4.1 Hyperparameter setting

During the kernel learning process, kernel hyperparameters must be carefully set. A change in the hyperparameters can be as relevant as a change in the structure of the kernel. These hyperparameters clearly influence the results of the kernel function, and therefore, the performance of SVMs.

In the initial kernel learning approaches, the hyperparameters were not even included in the grammar (Howley and Madden, 2005, 2006; Dioşan et al., 2007; Thadani et al., 2006). In other methods, random constants were incorporated to the grammar, which can be interpreted as hyperparameters that are learned together with the structure (Sullivan and Luke, 2007; Phienthrakul and Kijsirikul, 2007; Gijsberts et al., 2010; Alizadeh and Ebadzadeh, 2011; Gagné et al., 2006; Sousa et al., 2017). Alternatively, hyperparameters can be also learned apart from the structure in a secondary optimization procedure. The most common hyperparameter optimization method is grid search (Girdea and Ciortuz, 2007; Dioşan et al., 2012; Koch et al., 2012; Mezher and Abbod, 2014; Dioşan et al., 2008), although more complex methods have also been tried, such as particle swarm optimization (Schuh et al., 2012). As can be seen, choosing the right hyperparameters for the kernel remains an open question.

3.4.2 C parameter setting

The value of C also influences the evaluation of the quality of the kernels generated during the learning process. The simplest approach to fairly compare the kernels is to set a constant value for the C parameter ($C = c$) (Dioşan et al., 2007; Girdea and Ciortuz, 2007). However, this approach also creates a bias in the kernel selection process to that constant value of C.

The opposite approach is to run an exhaustive search in a reduced set of values (Koch et al., 2012). Here, a kernel-performance-maximizing C is selected for each of the visited kernels, increasing the computational cost of the search. We can classify these approaches depending on the method used to deal with the optimization of C, along with the search of a kernel and its hyperparameters: (i) the approaches that use a nested search procedure, where we optimize C for each kernel structure and hyperparameters visited in the search (ii) the methods that optimize C together with the kernel hyperparameters (Koch et al., 2012) and (iii) the algorithms that optimize C together with the kernel as a parameter of the kernel itself (Sousa et al., 2017).

Finally, the C parameter can be selected based on the characteristics of the evaluation of the kernel in the data as proposed by Chapelle (2002) (represented as c_h in this chapter). For each kernel, a good value of C is approximated while reducing the evaluation cost in a similar way to the fixed case.

Analogous to the hyperparameter tuning problem, the C parameter setting poses many questions when searching for the most suitable kernel. There is a clear interplay between these parameters, and also a trade-off between quality and computational cost.

3.4.3 Interplay between optimization procedures

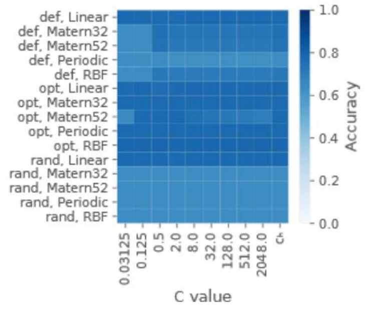
By means of the following experiment, we would like to investigate the interaction between the kernel hyperparameters and the C parameter. Particularly, we analyze the performance and the overfitting of SVMs with different kernel hyperparameters and C parameter values for several kernel structures in the mentioned UCI and PMLB datasets.

We have selected some of the standard kernels shown in Table 2.1. Three hyperparameter configurations are tested for each kernel: a set of default hyperparameters ($\theta_i = 1$), a random set of hyperparameters, and an optimized set of hyperparameters according to a grid search (2^{-5} to 2^4, at powers of 2, with a limit of 1000 evaluations) that maximizes the accuracy in the training set when $C = c_h$. Furthermore, for each kernel and hyperparameter configuration, 20 values (2^{-5} to 2^{15}, at powers of 2^2) for C are tried apart from c_h.

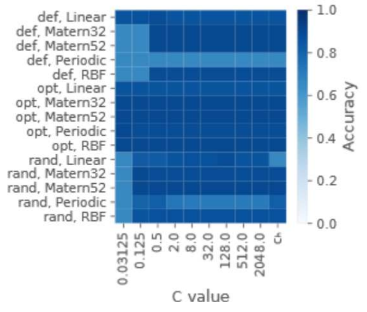
In Figures 3.5, 3.6 and 3.7, the results of the experiment are shown. For each database, the accuracy in the test set is represented by a heatmap. In the X axis the different values of C are shown, while in the Y axis the different kernels and their hyperparameters can be seen.

The best results are achieved with the optimized hyperparameters. This can be clearly seen in the *parity5+5* database, where the Periodic, RBF and Matern52 kernels obtain their best results when optimizing their hyperparameters. Regarding the influence of C, note that lower values of C show a lower performance in almost every configuration. Although there are some databases, such as *breast cancer Wisconsin* or *contraceptive*, where the C parameter has very little influence, in the rest of the problems certain C values are required to achieve the best possible performance. Also, it is worth mentioning that configurations with optimized hyperparameters show a more consistent performance for different values of C. Finally, it can be seen that c_h shows good accuracy values overall.

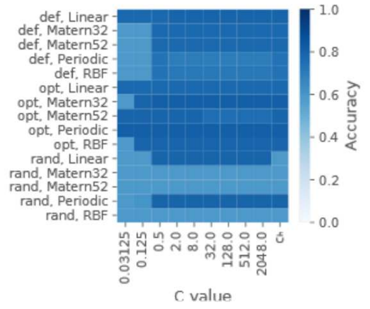
On the whole, there is a clear influence of the kernel structure and hyperparameters in the results, but also the C parameter can drastically change the quality of the prediction. In order to set the values of the hyperparameters and the C parameter, a grid search is highly recommended due to their strong interactions. Searching the hyperparameters on a grid and using the data based approach suggested by Chapelle (2002) to set the C parameter could be a good approximation in the cases where the exhaustive search is not computationally affordable.



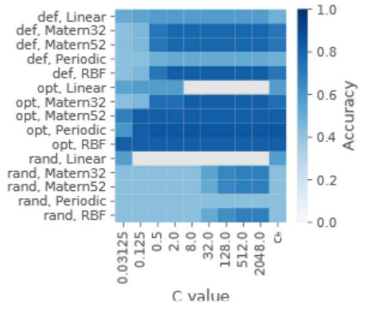
(a) pima



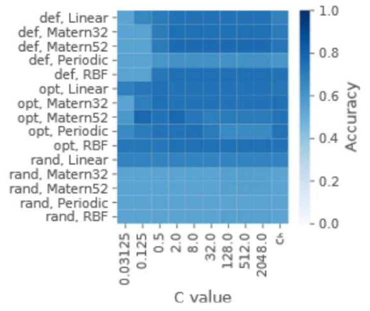
(b) ionosphere



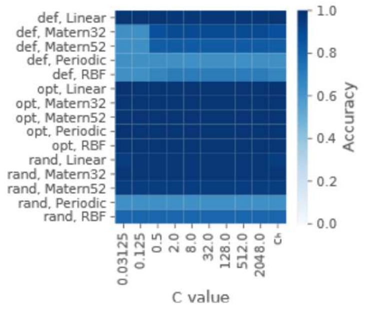
(c) heart statlog



(d) glass2

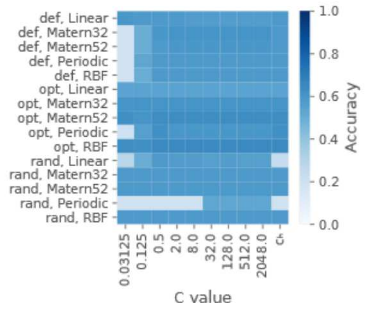


(e) liver disorder

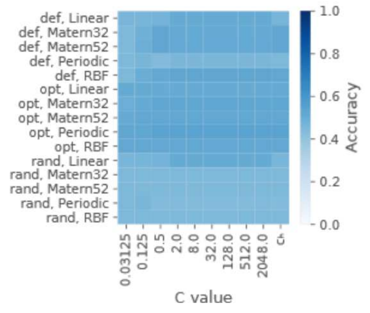


(f) breast cancer Wisconsin

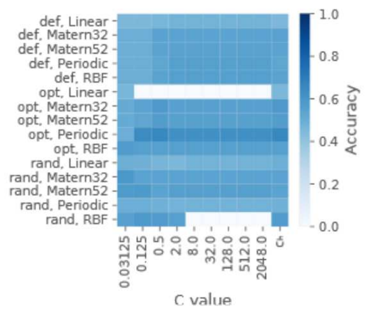
Fig. 3.5. Accuracy in the test set for different values of C per kernel and hyperparameter optimization methods for UCI datasets. c_h indicates the C value proposed by Chapelle (2002). *def* indicates the default set of hyperparameters, while *opt* and *rand* refer to the optimized and random hyperparameters respectively. The gray areas indicate the combinations for which the SVM could not be computed.



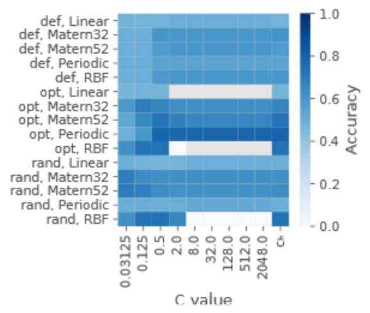
(a) calendarDOW



(b) contraceptive



(c) GAMETES Ep. 0.1H



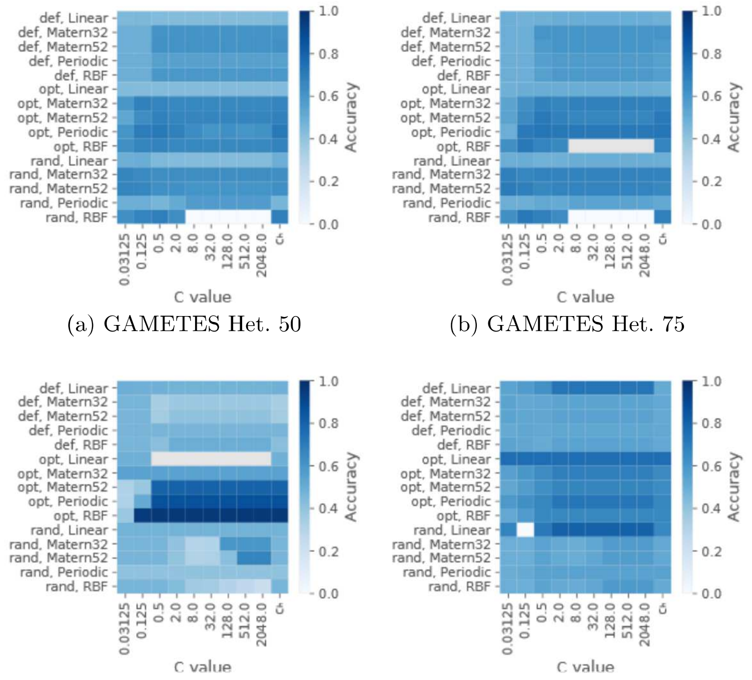
(d) GAMETES Ep. 0.4H

Fig. 3.6. Accuracy in the test set for different values of C per kernel and hyperparameter optimization methods for PMLB datasets. c_h indicates the C value proposed by Chapelle (2002). *def* indicates the default set of hyperparameters, while *opt* and *rand* refer to the optimized and random hyperparameters respectively. The gray areas indicate the combinations for which the SVM could not be computed.

3.5 Metrics

Another important aspect of the kernel learning is the metric used to evaluate its performance. The selected metric should be informative about the performance of the kernels in the training set, but it also needs to provide some clues as to the generalization ability of the kernel. Furthermore, it has a direct influence on the search method, as the roughness of the search landscape heavily depends on this choice.

In the SVM kernel search literature, almost every proposal uses accuracy (Gagné et al., 2006; Howley and Madden, 2005; Sousa et al., 2017; Mezher and Abbod, 2014; Alizadeh and Ebadzadeh, 2011; Sullivan and Luke, 2007; Dioşan et al., 2008; Schuh et al., 2012; Dioşan et al., 2012; Thadani et al., 2006; Girdea and Ciortuz, 2007) or classification error related metrics (Koch



(a) GAMETES Het. 50 (b) GAMETES Het. 75

(c) parity5+5 (d) Hill Valley with noise

Fig. 3.7. Accuracy in the test set for different values of C per kernel and hyperparameter optimization methods for PMLB datasets. c_h indicates the C value proposed by Chapelle (2002). *def* indicates the default set of hyperparameters, while *opt* and *rand* refer to the optimized and random hyperparameters respectively. The gray areas indicate the combinations for which the SVM could not be computed.

et al., 2012; Howley and Madden, 2006; Gijsberts et al., 2010) to measure the goodness of the kernel. As these metrics are discrete, some of the approaches include tiebreakers to deal with the same results when comparing similar kernels (Howley and Madden, 2006; Thadani et al., 2006). There is very little knowledge about the performance of other metrics. For example, Valerio and Vilalta (2014) tried to estimate the goodness of the matrix generated by the kernel measuring the intra/extra class similarity ratio instead of evaluating the SVMs.

The accuracy may seem the best choice according to the literature, but it also has some drawbacks. As previously mentioned, this measure requires some methods to deal with tie results. It produces a search landscape where we can not directly obtain the gradient of this metric, disallowing the usage of many search methods that exploit this feature to optimize the C parameter, the kernel or its hyperparameters. Besides, the accuracy does not include any information about the generalization ability of the model and requires some k-fold cross-validation to mitigate overfitting, which is very computationally demanding.

3.5.1 Likelihood based metric for the SVM kernel selection

Considering the issues derived from using the accuracy as a metric to guide the selection of kernels, there appears to be a need to investigate alternative methods to measure the performance of the kernels. In order to provide a more robust measure than the accuracy, we propose a Bayesian Information Criterion (BIC) (Schwarz, 1978) based measure for SVMs, SVMBIC. This measure uses Platt scaling (Platt, 1999) to obtain probabilistic predictions of the SVM model and includes a complexity penalization according to the number of hyperparameters. It can be described as follows:

$$SVMBIC(k_i) = -2 \sum_{i=0}^{n} log\, p\,(y_i|\mathbf{x}_i, k_i, \boldsymbol{\theta}_{i,best}) + q\; log\; n \tag{3.1}$$

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 structure k_i.

Being a continuous measure, SVMBIC provides a smoother landscape than the accuracy, as small changes in the SVM parameters produce little variations in their values. Therefore, gradient based approaches can be used to optimize the hyperparameters or C. Besides, it includes an explicit complexity penalization.

The following experimental scenario was designed as a test for this new measure against the commonly used accuracy metric. By using GenProg as described in Section 3.3, we search for new kernels in the UCI datasets. In order to take advantage of properties of the BIC measure, we have optimized the hyperparameters based on a multi-start variation of Powell's conjugate direction method (Powell, 1964) for every kernel, while for the experimental setting with the accuracy measure, we perform a grid search to find the best parameters. In both cases the same amount of evaluations was allowed (1000). The C parameter was set following a grid search as in the experiment described in Section 3.2.1.

The results of the experiment are summarized in Table 3.8. As expected, in all the problems, the kernel structures optimized using accuracy as metric achieve better results in the training set than those learned by means of the SVMBIC metric, being particularly noticeable in the *GAMETES Heterogeneity 50* problem. However, when compared in the test set, the performance gap between these two methods is undoubtedly lower for most of the problems. In fact, for the *heart statlog* and *contraceptive* problems, the SVMBIC method outperforms the results obtained by using accuracy as the metric.

	Classification problem	# of HPs Ac.	# of HPs BIC	Training set Ac.	Training set BIC	Test set Ac.	Test set BIC
UCI	pima	4.2	0.1	**0.793**	0.770	**0.759**	0.758
	ionosphere	4.8	0.4	**0.962**	0.939	**0.946**	0.942
	heart statlog	3.4	0.0	**0.878**	0.847	0.802	**0.820**
	glass2	4.8	0.7	**0.909**	0.835	**0.836**	0.803
	liver disorder	5.4	0.3	**0.763**	0.727	**0.719**	0.712
	breast-cancer-wisco.	4.6	0.0	**0.986**	0.978	**0.973**	**0.973**
PMLB	calendarDOW	5.7	0.7	**0.665**	0.592	**0.616**	0.594
	contraceptive	4.5	0.8	**0.57**3	0.552	0.552	**0.553**
	GAMETES Epistasis 0.1H	4.8	1.6	**0.681**	0.674	**0.675**	0.670
	GAMETES Epistasis 0.4H	4.1	1.9	**0.793**	0.734	**0.796**	0.739
	GAMETES Heterogeneity 50	4.4	1.1	**0.709**	0.576	**0.712**	0.576
	GAMETES Heterogeneity 75	5.3	1.0	**0.733**	0.681	**0.708**	0.675
	parity5+5	3.8	1.7	**0.999**	0.987	**1.000**	**1.000**
	Hill_Valley_w._noise	4.0	1.8	**0.905**	0.863	**0.910**	0.884

Table 3.8. SVMBIC (BIC) measure compared to accuracy (Ac.). The average number of hyperparameters of the best kernels is shown on the left. The mean accuracy achieved by each kernel search metric (in the training and test sets) is shown in the right-most columns. The numbers in bold indicate the best result for each problem.

Moreover, in Table 3.8, the average number of hyperparameters of the best kernels obtained in each search procedure is shown. As can be seen, the SVMBIC approach clearly penalizes the number of hyperparameters, showing a lower average number of hyperparameters per kernel than the accuracy guided approach.

According to the results of our experimentation, SVMBIC can be used to obtain simpler kernels than those obtained with accuracy, close to them in terms of performance, even outperforming the latter in some of the runs. The reduction of the overfitting of the solutions, together with the continuous nature of the SVMBIC measure, can contribute to improving the performance of a GenProg-like search.

3.6 Conclusions

Kernel functions are a key element of SVMs, as their performance strongly depends on them. Although the previous works in automated kernel search for SVMs have focused on the search algorithm itself, there are other components of the method that influence and condition the performance of SVMs that have not received the same attention. In this chapter, we have identified those components and analyzed the interactions between them, with the aim of obtaining a more general view about the kernel search for SVMs, making the following contributions:

- Identification of the components that influence the performance of SVMs: Apart from the weights of the hyperplane and the bias of the SVM, the kernel structure, its hyperparameters and the C parameter are also important for the efficiency of the method. The practitioner should consider all these components as a whole, instead of focusing on just one of them.
- The intrinsic limitation of using a reduced set of databases to evaluate kernel search strategies has been exposed: We have identified a number of datasets where the behavior of standard kernels is far from being optimal. We highlight the need to extend the benchmark of datasets and increase the variety of characteristics that the datasets exhibit.
- Analysis of the kernel space: The kernel space where the search is carried out is a key element for the automated kernel search. We have proposed a basic mathematical expression based grammar, and proved the influence of including different elements in the performance of SVMs. All in all, it is worth expanding the search space by adding new elements to the grammar.
- Insights about the kernel structure search have been provided: Several methods have been proposed to learn the structure of the kernel for SVMs. We have compared the performance of the GenProg to other simpler search methods, obtaining marginal gains over them. Including the right elements in the grammar can be more important than the search method itself when trying to find the best kernel structure.
- Study the interplay of the hyperparameter tuning and C parameter setting: We have shown that the value of the hyperparameters and the value of C can drastically change the behavior of SVMs. We have also provided guidance on setting those parameters during the kernel search.
- A novel metric for the SVM kernel search has been proposed: Although the accuracy has been the standard measure in the SVM classification problems, it also presents some challenges. We propose a metric based on the BIC measure, which can overcome these problems.

In conclusion, in this work we have provided several ideas to improve the setting and, consequently, the performance of SVMs. We have shown the importance of understanding the characteristics of the kernel space beyond the search method itself. We have also provided some clues about the best practices to set the hyperparameters and the C parameter in order to better balance the computational effort used during the kernel search. Finally, we have proposed a measure to evaluate the efficiency of the kernel, which should be further studied.

4

GenProg approach to learn GP kernels based in basic mathematical expressions

4.1 Introduction

Kernel learning methods based on basic mathematical expressions have their own advantages, although, as we have seen in SVMs, finding the most appropriate solution can be 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. On the GP side, having a grammar that allows non-PSD kernels adds some additional challenges. For example, although SVMs can deal with non-Mercer kernels, the exact inference of GPs requires the kernels to be PSD. Therefore, some procedure to exclude the non-PSD kernels should be included during the search in the space of mathematical expressions. Furthermore, the performance of GPs is very sensitive to the values of kernel hyperparameters. Although most SVM approaches (Koch et al., 2012; Dioşan et al., 2007; Howley and Madden, 2006) deal with hyperparameters by means of optimizing small grids or using GenProg with random constants in the grammar, more advanced techniques are required to adapt the hyperparameters during the kernel search for GPs.

In this chapter, 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. In order to deal with non-PSD kernels, we bring the approach proposed by Koch et al. (2012) in SVMs to GPs, where the kernel creation is retried if a non-PSD kernel is found. Moreover, we add a hyperparameter inheritance method to GenProg in order to adapt the hyperparameters.

This method is extended and applied in Chapters 5 and 6, where kernel functions for GPs are learned to solve time series extrapolation and NLP problems respectively.

4.2 Elementary mathematical expression grammar for GPs

First, we introduce a new elementary mathematical expression grammar for GPs, shown in Table 4.1, similar to the work done for SVMs in Section 3.2.2. After some preliminary experiments, we concluded that, contrary to the SVMs, a greater number of elements in the grammar was detrimental to the search. Thus, we use a simpler grammar in order to reduce the number of possible kernels and improve the performance of the search. First, -1 and -0.5 terminals are used instead of the *minus* operator. Also, *tanh* is not included since it was hardly ever used in the literature of GPs, as well as the *noise* operator. Finally, *x_div* and *x_rest* are merged with *sq_distance*, *dot_product* and *spectral* operands, in order to reduce the complexity of vector operators. Despite having simplified the grammar, the standard kernel functions in Table 2.1 and their sum/product compositions remain a subset of our search space.

$kernel :$	$scalar$	start symbol
$scalar :$		
	$\mid scalar^{hp}$	power
	$\mid scalar + scalar$	add
	$\mid scalar \times scalar$	multiply
	$\mid scalar^{-1}$	div
	$\mid e^{scalar}$	exp
	$\mid \sqrt{scalar}$	sqrt
	$\mid scalar^2$	square
	$\mid \left\Vert \frac{\mathbf{invec}}{hp} - \frac{\mathbf{invec}'}{hp} \right\Vert^2$	sq_distance
	$\mid \left(\frac{\mathbf{invec}-\mathbf{1}hp_1}{hp_0}\right) \cdot \left(\frac{\mathbf{invec}'-\mathbf{1}hp_1}{hp_0}\right)^T$	dot_product
	$\mid hp$	constant
	;	
$(\mathbf{invec}, \mathbf{invec}') :$		
	$\mid \left(\left[sin(\frac{2\pi\mathbf{x}}{hp})\ cos(\frac{2\pi\mathbf{x}}{hp})\right],\right.$	
	$\left.\left[sin(\frac{2\pi\mathbf{x}'}{hp})\ cos(\frac{2\pi\mathbf{x}'}{hp})\right]\right)$	spectral
	$\mid (\mathbf{x}, \mathbf{x}')$	input
	;	
$hp :$		
	$\mid -1 \mid -0.5 \mid 1 \mid 2 \mid 3 \mid 5$	
	$\mid \theta_0 \mid \theta_1 \mid ... \mid \theta_t$	
	;	

Table 4.1. Proposed grammar for GPs. t indicates the number of different hyperparameters allowed in the grammar (in this work it is set to $t = 20$).

4.3 Evolving kernel functions for GP based on the new grammar

Once the grammar has been introduced, we present our GenProg approach for GP kernel search, EvoCov. This algorithm 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.

Our GenProg approach is shown in Algorithm 2. 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. As shown in the GenProg method studied in Section 3.3, 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 4.3.3.2), 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 generations. Finally, the last population is evaluated and the best individual found during the whole process is returned.

In the following sections, we describe each of the methods used by Algorithm 2. First, we address the issue of randomly generating new kernels for the initial population. Then, we provide the variation operators conceived to generate kernels that are likely to inherit useful characteristics from the selected ones. A method to control the depth of the expressions created by the variation operators is also introduced. Next, we explain how the GP kernels are evaluated.

4.3.1 Initial population

We generate kernel functions at random until the desired population size is reached, using the same procedure described in Section 3.2.2.1. Similarly, we identify the non-PSD kernels as in Section 3.2.2.2, and generate a new one if any of the described conditions are met, as non-PSD kernels are not valid for GPs. However, non-compliance with all these conditions is necessary but not sufficient for a kernel to be PSD. Although in SVMs the evaluation of the kernel can be carried out despite being non-Mercer, the covariance matrix can not be inverted for GP inference. Thus, if after passing the PSD check, we find

Algorithm 2 EvoCov algorithm

```
1: procedure EvoCov($N$, $G$, $S$, $p_m$, $p_{cx}$, $\beta$, $d_{min}$, $d_{max}$)
2:     $pop$ = GenRandPop($N$, $d_{min}$, $d_{max}$)
3:     $bestfit_{-1} = \infty$
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:        $relimprov = \frac{bestfit_{i-1} - bestfit_i}{|bestfit_i|}$
11:        if $\beta < relimprov$ then
12:            $sel$ = Select($pop$, $S$)
13:            $offspring$ = Variate($sel$, $N - S$, $p_m$, $p_{cx}$)
14:        else                                              ▷ Restart procedure
15:            $sel = \emptyset$
16:            $offspring$ = GenRandPop($N$, $d_{min}$, $d_{max}$)
17:            $bestfit_i = \infty$
18:        end if
19:        $pop = sel \cup offspring$
20:        $all = all \cup offspring$
21:        $i = i + 1$
22:    end while
23:    Evaluate($pop$)
24:    $best$ = Select($all$, 1)
25:    return $best$
26: end procedure
```

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 conducted in Chapter 5, only 0.67% were not PSD.

4.3.2 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 2: 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.

4.3.2.1 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 2.3.1, 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.

4.3.2.2 Mutation

Based on the work done by Fortin et al. (2012), the algorithm applies a mutation operator that randomly selects one of the following methods in a type-safe manner:

Insert: Inserts an elementary mathematical expression (see Table 4.1) at a random position in the kernel expression, as long as both the output of the operator at the selected position and the output of the expression to be inserted agree. The operator at the chosen position is used as the input of the newly created expression. If more inputs are required by the new operator, new terminals are chosen at random.

Shrink: This operator shrinks the kernel expression by randomly choosing an operator and replacing it with one of its arguments (also randomly chosen) of the same type.

Uniform: Selects a point uniformly at random in the kernel expression and replaces the sub-expression at that point by a randomly generated sub-expression, using the random generation method described in Section 3.2.2.1. Note that the output type of the new sub-expression must match the output type of the replaced one.

Replacement: Replaces a randomly chosen operator from the kernel expression by an operator with the same number of inputs 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 3.2.2.2).

4.3.2.3 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 (Koza, 1992). 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.

4.3.3 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 Figure 4.1.

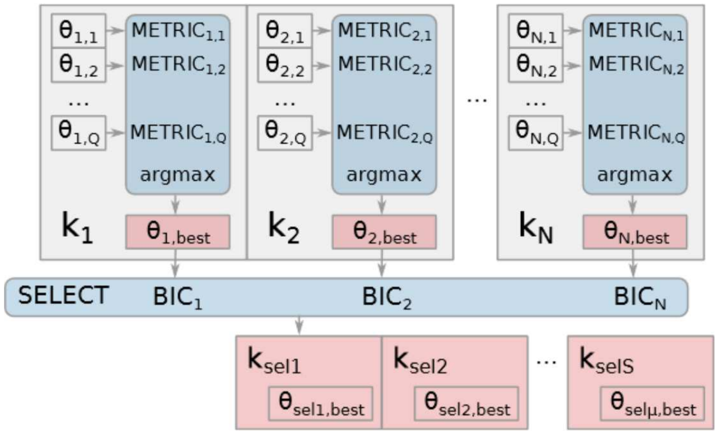


Fig. 4.1. Two nested search procedures: The selection of the best hyperparameters for each kernel is made according to a given METRIC, such as LML or LOOCV, and the selection of the best kernels according to the BIC.

Following the work done by Duvenaud et al. (2013), we use the Bayesian Information Criterion (BIC) (Schwarz, 1978) 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:

$$BIC(k_i) = 2\ log\ p\left(\mathbf{f}|X, k_i, \boldsymbol{\theta}_{i,best}\right) - q\ log\ n \tag{4.1}$$

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 2.2.2.2, 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:

$$\boldsymbol{\theta}_{i,best} = argmax_j\ \text{METRIC}\left(\mathbf{f}, X, k_i, \boldsymbol{\theta}_{i,j}\right) \tag{4.2}$$

4.3.3.1 Hyperparameter Optimization Algorithm

The hyperparameters are optimized by means of *Powell*'s local search algorithm (Powell, 1964). 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.

4.3.3.2 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 Duvenaud et al. (2013) and Lloyd et al. (2014) 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_θ). This inheritance method is particularly useful when the variation performs few changes to the expression.

Note that, in Algorithm 2, 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 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.

4.3.4 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 2 selects the S best kernels according to the BIC metric by applying truncation selection.

4.4 Conclusions

In this chapter, we have presented an evolutionary approach to learn kernel functions for GPs. While other GP approaches are based on kernel composition, in our approach, kernels are modeled by means of basic mathematical expressions, making the following contributions:

- Basic mathematical expressions as building blocks for GP kernels: We propose bringing 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, following the approach proposed by Koch et al. (2012) in SVMs: 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 expressions based on the properties of the covariance matrix.
- Adapt the hyperparameter inheritance to GenProg: We have incorporated the hyperparameter inheritance technique described by Duvenaud et al. (2013) and Lloyd et al. (2014) within GenProg, improving the efficiency of the algorithm.

5

Automatic GP kernel learning for time series extrapolation

5.1 Introduction

The objective in time series extrapolation is to predict future time-stamp values given some previous data. While properties such as the smoothness of the data have been extensively studied in GP literature for interpolation problems, other properties required in extrapolation, including periodicities and trends, have not been studied to the same extent.

In this chapter, we solve real-world time series prediction problems by using the EvoCov algorithm to learn kernels for GP, showing that these models are also valuable in extrapolation tasks. In order to validate this approach, we have conducted an extensive experimentation. In particular, the goals of the experiments carried out in this chapter are:

- To compare the proposed mathematical expression based grammar to the kernel composition approaches (Duvenaud et al., 2013).
- To benchmark EvoCov to state-of-the-art methods for time series extrapolation tasks (Lloyd et al., 2014).
- To compare our proposal to the ad hoc kernels proposed in the literature (Preoţiuc-Pietro and Cohn, 2013).
- To study the influence of the metric used to optimize the hyperparameters in time series extrapolation problems.

The chapter has been organized in the following way: first, an introduction to the time series extrapolation problem is given, before describing the experimental setup. In order to validate the different components of the proposed GenProg algorithm, we include two simpler kernel learning methods for GP: Random Search and GoWithTheFirst. Next, four experiments are shown, one for each objective of the experimentation.

5.2 Time series extrapolation problems

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.

One of the datasets most frequently used to test GP approaches is the Mauna Loa Atmospheric CO_2 time series (Rasmussen and Williams, 2006; Kronberger and Kommenda, 2013; Duvenaud et al., 2013). This time series, shown in Figure 5.1, was analyzed in detail by Rasmussen and Williams (2006), who also proposed a hand-tuned kernel function. Duvenaud et al. (2013) also used this time series to investigate the structure discovery for kernel functions, improving the results shown by the kernel proposed by Rasmussen and Williams (2006).

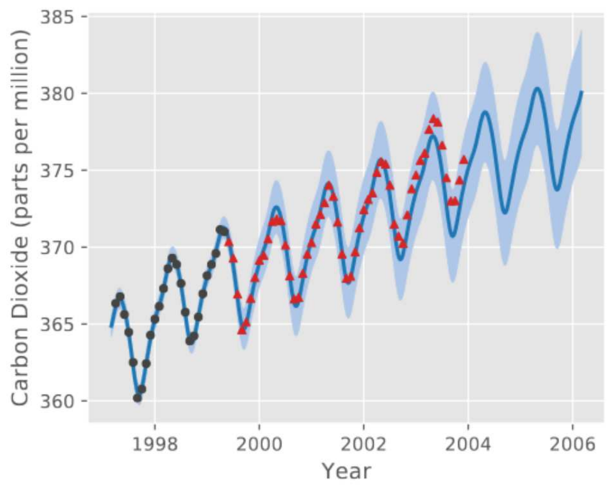


Fig. 5.1. Time series extrapolation in the Mauna Loa Atmospheric CO_2 dataset. The dots represent the last samples of the training set (the first samples of the training set are not shown), while the triangles show the samples of the test set. The prediction given by the ad hoc kernel proposed by Rasmussen and Williams (2006) is illustrated with a continuous blue curve for the mean of the GP model and the light blue shadow shows 3 times the standard deviation.

In addition to the mentioned time series, other common datasets have been included in this study. Following the work done by Lloyd et al. (2014), in the first two experiments, we use the real-world time series described in

Table 5.1, which are also illustrated in Figures 5.2 and 5.3[1]. We trained all the algorithms on the first 90% of the data, and predicted the remaining 10%.

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

Table 5.1. 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.

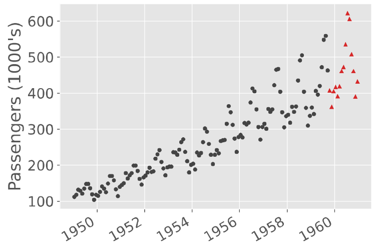
5.3 Alternative search methods

In order to verify that every component of the GenProg algorithm introduced in the previous chapter 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 (Rastrigin, 1963) 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 (Aldous and Vazirani, 1994).

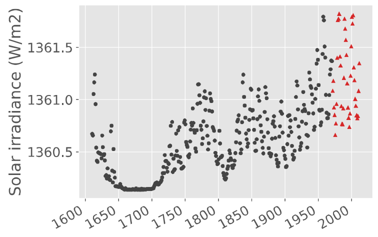
5.3.1 Random Search algorithm

As shown in Algorithm 3, this Random Search method generates a random population by iteratively following the random generation method described in Section 3.2.2.1 until the desired population size is achieved (N). Next,

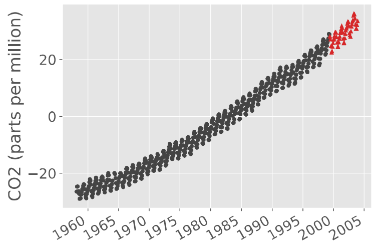
[1] The time series data can be found at `https://pkg.yangzhuoranyang.com/tsdl/`



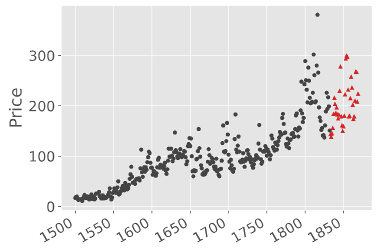
(a) Airline



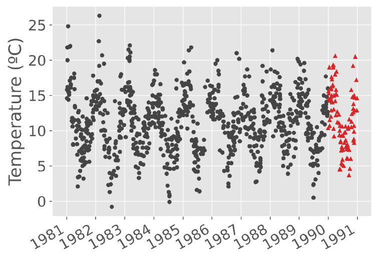
(b) Solar



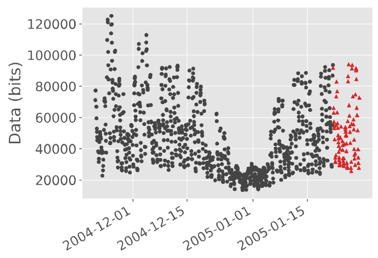
(c) Mauna Loa



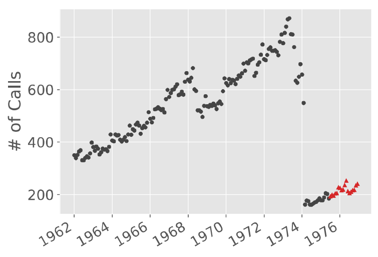
(d) Wheat Price



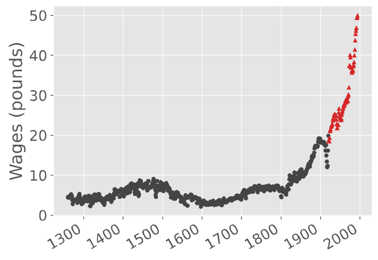
(e) Mel. Temps.



(f) Traffic data

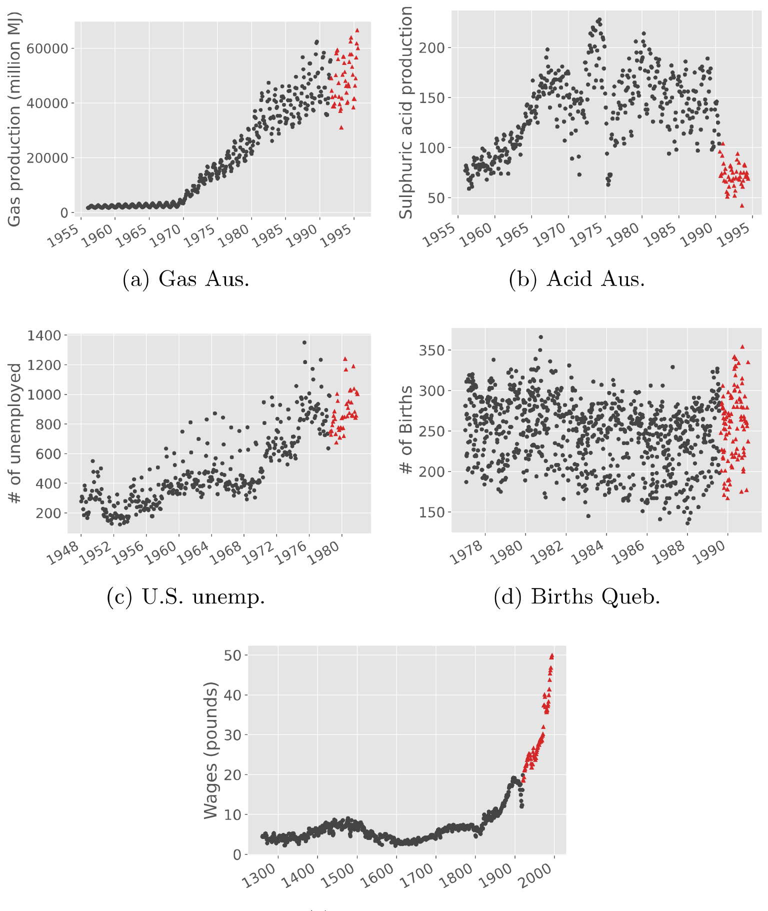


(g) Avg. calls.



(h) Radio freqs.

Fig. 5.2. Time series extrapolation problems. The dots represent the samples of the training set, while the triangles show the samples of the test set.



(a) Gas Aus. (b) Acid Aus. (c) U.S. unemp. (d) Births Queb. (e) Wages Eng.

Fig. 5.3. Time series extrapolation problems. The dots represent the samples of the training set, while the triangles show the samples of the test set.

it chooses the best solution according to the selection criterion described in Section 4.3.4.

5.3.2 GoWithTheFirst algorithm

The GoWithTheFirst algorithm, shown in Algorithm 4, (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

Algorithm 3 Random Search Algorithm

```
1: procedure RANDOM SEARCH($N$, $d_{min}$, $d_{max}$)
2:     $pop$ = GenRandPop($N$, $d_{min}$, $d_{max}$)
3:     $best$ = Select($pop$, 1)                    ▷ Best kernel is selected
4:     return $best$
5: end procedure
```

worst individual among this optimized population, according to the selection criteria. The steps (ii) and (iii) are repeated until only one kernel is left.

Algorithm 4 GoWithTheFirst algorithm

```
 1: procedure GO-WITH-THE-FIRST($N$, $H$, $d_{min}$, $d_{max}$)
 2:     $pop$ = GenRandPop($N$, $d_{min}$, $d_{max}$)
 3:     $i = 0$
 4:     while $i < (N - 1)$ do                         ▷ Generation loop
 5:         $j = 0$
 6:         $best\_pop = \emptyset$
 7:         while $j < (N - i)$ do                     ▷ Population loop
 8:             $best = pop_j$
 9:             $k = 0$
10:             while $k < H$ do                       ▷ Local search loop
11:                 $mutation$ = Mutate($best$)
12:                 $best$ = Select($mutation \cup best$, 1)
13:                 $k = k + 1$
14:             end while
15:             $best\_pop = best\_pop \cup best$
16:             $j = j + 1$
17:         end while
18:         $pop$ = Select($best\_pop$, $N - i - 1$)
19:         $i = i + 1$
20:     end while
21:     return $pop_0$             ▷ Return the only individual in the population
22: end procedure
```

5.4 Experimental Setup

In order to validate our approach, we designed the following experimental setup. In Section 5.5, we compare several metrics to optimize the hyperparameters, in order to find the most appropriate one in this context of time series extrapolation problems. Then, we validate the proposed basic mathematical expression based grammar in Section 5.6, by introducing random mutations to some of the best performing kernels in these problems, learned by means of kernel composition. Section 5.7 is devoted to comparing the EvoCov algorithm to various state-of-the-art methods in an extensive benchmark. Finally, in Section 5.8, we compare our proposal to an ad hoc kernel specifically designed to predict trends in the Twitter timeline.

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 3.2.2.2 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, we set the parameters to the following values: $N = 141$, $G = 141$, $S = 14$, $p_m = 0.4$, $p_{cx} = 0.6$ and $\beta = 1\text{e}{-5}$. All these algorithms were also coded in Python, based on the EA software *DEAP*[2] (Fortin et al., 2012). 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 (Powell, 1964), a Gaussian noise with $\sigma_\theta = 0.1$ was added to the inherited hyperparameters. Since the computational time required to evaluate the hyperparameters increases quadratically with the size of the time series, in order to keep a reasonable computational cost to optimize these hyperparameters in all the problems, we decided to adjust the number of evaluations allowed in the hyperparameter optimization depending on the length of the time series. Thus, we allow $Q = ref\_fun\_call \times \frac{350^2}{current\_ts\_len^2}$ evaluations, where $current\_ts\_len$ is the length of the current time series and ref_fun_call is a parameter of the algorithm. In the experiment shown in Section 5.5, we allow $ref\_fun\_call = 5000$ evaluations, and for the rest of the experiments $ref\_fun\_call = 300$.

5.5 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 (Kronberger and Kommenda, 2013; Duvenaud et al., 2013). However, it has been reported that the LML may lead to suboptimal results under certain conditions, where LOOCV could be more robust(Bachoc, 2013). Regarding the kernel optimization, having a consistent method to optimize the

[2] `https://deap.readthedocs.io`

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.

Furthermore, we also use the Negative Log Predictive Densities (NLPD) (Quiñonero-Candela et al., 2006) 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:

$$L_{NLPD}(X_*, \mathbf{f}_*, X, \mathbf{f}, \boldsymbol{\theta}) = -\frac{1}{n_*}\sum_{i=1}^{n_*} log\, p\,(f_{*i}|X_{*i}, X, \mathbf{f}, \boldsymbol{\theta}) \tag{5.1}$$

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 5.1. Thus, we considered the best kernels found for each of the problems in the work done by Duvenaud et al. (2013). 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 5000 samples have been taken. Due to the randomness of the process, 10 trials were carried out.

In Table 5.2, 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.

	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**

Table 5.2. 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.

Statistical tests were used to determine if there is a metric that is more robust than the others in time series extrapolation problems[3]. First, we averaged all the RMSE results for each metric and time series, and then applied Friedman's test (Friedman, 1937). We found significant differences between all the metrics ($\alpha = 0.05$. p-value $= 8.454\mathrm{e}{-5}$). Then, we applied a post-hoc test based on Friedman's test, and adjusted the results with Shaffer's correction (Shaffer, 2012). In Figure 5.4, 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.

As the differences between LML and NLPD are not significant, in the following experiments, we used two variants of EvoCov for optimizing the parameters: one using LML (EvoCov-LML) and the other using NLPD (EvoCov-NLPD).

[3] The tests were carried out using the SCMAMP R package (Calvo and Santafé, 2016)

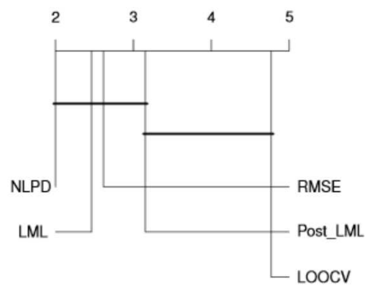


Fig. 5.4. Critical differences diagram between hyperparameter optimization metrics. The metrics are ordered following their rankings. The metrics with no significant differences between them are matched with a straight line.

5.6 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 chapter.
2. Whether the proposed mutation operator allows such an improvement.

Given the best kernels found by Duvenaud et al. (2013) for all the time series, we generated 200 random mutations according to the method described in Section 4.3.2.2. 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 Duvenaud et al. (2013).

Table 5.3 shows the results of this experiment for the 13 time series previously introduced. In addition, for illustrative purposes, the results for the Mauna Loa Atmospheric CO_2 time series are detailed in Figure 5.5.

Regarding the results in Table 5.3, 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.

In Figure 5.5 the RMSE and the number of hyperparameters of the 200 random mutations for the Mauna Loa Atmospheric CO_2 time series are illustrated. There are 36 mutated kernels that obtain a better RMSE than the original one for the Mauna Loa Atmospheric CO_2 time series. Moreover, some of those kernels have a slightly lower number of hyperparameters (9 instead of 10) than the best kernel achieved by Duvenaud et al. (2013).

As we have shown in this experiment, we conclude that it is possible to improve the results obtained by compositional kernel search approaches by

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

Table 5.3. 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 Duvenaud et al. (2013) 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.

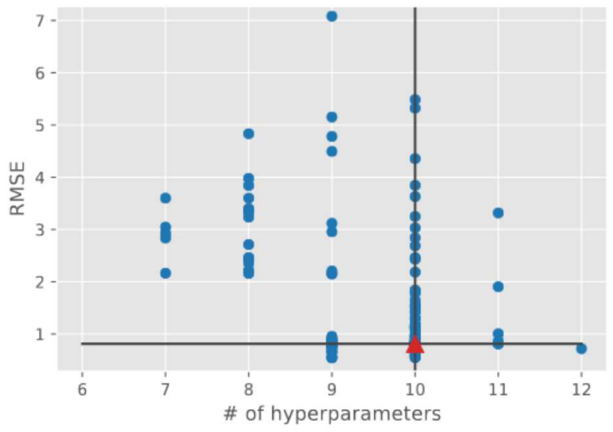


Fig. 5.5. RMSE and number of hyperparameters of the 200 random mutations for the Mauna Loa Atmospheric CO_2 time series. The triangle represents the original kernel, and the vertical and horizontal straight lines represent its number of hyperparameters and RMSE respectively.

means of manipulating elementary mathematical expressions. We can also confirm that the mutation operator presented in this work allows such improvements.

5.7 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 Lloyd et al. (2014). 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 algorithms to search in the space of the possible equations (Schmidt and Lipson, 2013). 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 2.1 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 (Bach et al., 2004).

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 (Garnett et al., 2010; Saatçi et al., 2010; Fox and Dunson, 2012).

Spectral Mixture Kernels (SK): These kernels model the spectral density with a Gaussian mixture (Wilson and Adams, 2013).

Trend-Cyclical-Irregular (TCI) Models: The statistical model described by Lind et al. (2006) 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 Duvenaud et al. (2013) is used.

ABCD accuracy (ABCDa): An improvement of GPSS, introduced by Lloyd et al. (2014), 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 (Lloyd et al., 2014).

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.

Table 5.4 shows the numerical results of the experimentation for each time series, while in Figure 5.6, the overall results are illustrated. EvoCov is presented in two variants, one using LML to optimize the hyperparameters, and the other variant using the NLPD metric. Note that RMSE is standardized by dividing by the smallest RMSE achieved in the experiments for each dataset, so that the best performance on each dataset has a value of 1. Also, it is worth mentioning that, in the experiments conducted by Lloyd et al. (2014), only one trial for each time series and algorithm was carried out[4], and, for our algorithms, the mean and the best of ten trials are shown.

As we can see in Table 5.4, EvoCov-LML achieves the best average result across all problems and beats the rest of the algorithms in the *Mauna Loa Atmospheric CO_2* time series. Moreover, its overall median result is the second best, very close to the best one. The best trials of this algorithm outperform the rest of the algorithms in three other problems. Our GoWithTheFirst algorithm is the best choice in *Monthly critical radio frequencies* time series, and its best trials were better than the other algorithms in 6 problems. Although its average performance is not as good as EvoCov-LML, it was able to achieve the third best median RMSE. EvoCov-NLPD shows the best behavior in *Solar* time series. On the contrary, this approach obtains poor results in *Monthly average daily calls*, compromising its average result. The Random Search was able to get the best result for the *Solar* time series in its best trial. However, this algorithm has a worse performance than those already mentioned, confirming the contribution of the mutation and crossover operators. Regarding the compositional kernel search approaches, GPSS approach gets the second best mean result, and is able to beat the rest of the algorithms in *Daily minimum temperatures in Melbourne* time series. ABCDa obtains the third best mean result. This algorithm gets the best median result and holds the best results in three of the time series. ABCDi, with worse results than ABCDa according to the mean, is the best approach in *Monthly average daily calls* and *Number of daily births in Quebec* problems. On the other hand, there are some other time series, such as *Beveridge Wheat Price Index*, *Monthly U.S. male unemployment* and *Airline*, where the algorithms based on the CP, Linear and Spectral kernels, produce better results than compositional kernel search approaches. Finally, out of the GP approaches, Eureqa outperforms the rest of the approaches in *Real daily wages in England* time series. However, this symbolic regression engine is worse than EvoCov-LML in the rest of the time series.

Table 5.5 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

[4] The results were gathered from the supplementary material of the work done by Lloyd et al. (2014).

	ABCDa	GPSS	ABCDi	CP	LIN	MKL	SE	SP-bic	TCI	eureqa	Random Search		Go With The First		EvoCov LML		EvoCov NLPD	
											Best	Mean	Best	Mean	Best	Mean	Best	Mean
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	1.02	1.45	1.0*	1.31	1.11	1.56
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	1.08	1.84	1.07	1.84	1.01	**1.60**
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	1.0*	1.50	1.07	**1.35**	1.08	2.50
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	1.0*	2.37	1.04	1.29	1.32	1.77
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	1.00	1.01	1.00	1.02	1.01	1.02
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	1.64	3.89	1.0*	3.63	1.46	4.82
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	1.33	6.82	1.16	2.81	1.45	11.69
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	1.0*	**1.35**	1.07	1.68	1.78	4.08
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	1.0*	2.75	1.06	2.50	1.39	2.34
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	1.0*	1.52	1.08	1.43	2.03	2.33
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	1.30	1.59	1.0*	1.74	1.17	1.84
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	1.0*	1.13	1.05	1.21	1.14	1.45
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	2.99	3.60	2.88	3.55	1.28	2.56
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

Table 5.4. Standardized RMSE for each extrapolation problem and algorithm is shown. In our approaches, the mean and the best 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.

	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

Table 5.5. 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 Lloyd et al. (2014).

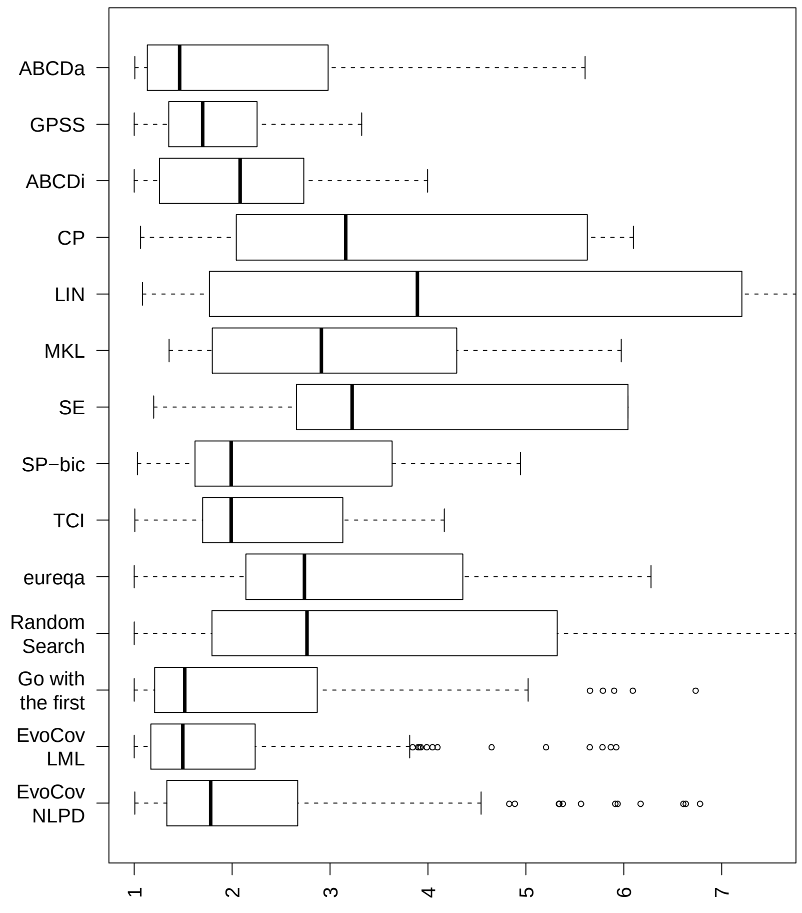


Fig. 5.6. Standardized RMSE of each algorithm in each problem. Note that the results of our algorithms have more observations due to the 10 trials.

hyperparameters, and EvoCov-LML and EvoCov-NLPD use 6.8 and 7.0 hyperparameters respectively. An example of the difference in the complexity between kernels learned by composition approaches and those based on basic mathematical expressions can be found in Appendix A.

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.

5.8 Comparing our proposal to ad hoc kernel approaches

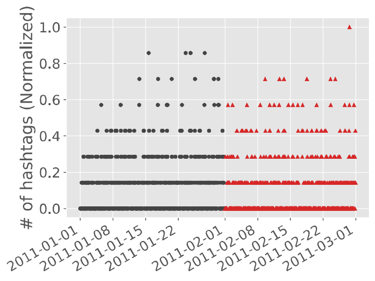
As we have mentioned in Section 2.3.1, many works in GPs propose a human-designed specific kernel for each particular problem. In the work carried out by Preoţiuc-Pietro and Cohn (2013), the number of tweets that contain a given hashtag in the Twitter timeline was predicted by means of GPs. The authors 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 Preoţiuc-Pietro and Cohn (2013), where *#goodmorning*, *#breakfast*, *#confessionhour*, *#fail*, *#fyi* and *#raw* hashtags were predicted[5]. 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. These time series are illustrated in Figure 5.7, where the number of occurrences of each hashtag is shown.

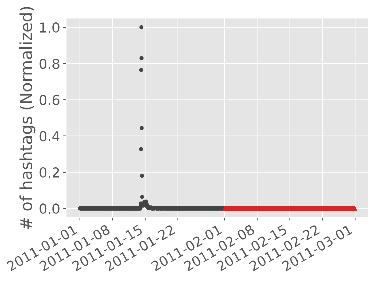
In Figure 5.8, an example of the hashtag prediction is illustrated, where the best model given by our approach in a single run is shown. In this problem, a periodic trend can be appreciated, which is successfully captured by our model.

Table 5.6 shows a comparison between EvoCov-LML, EvoCov-NLPD and the PS kernel. The experiments with the PS kernel were 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

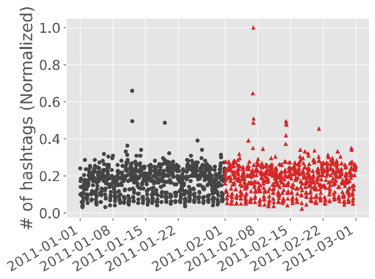
[5] Data can be found at `https://web.sas.upenn.edu/danielpr/resources/`



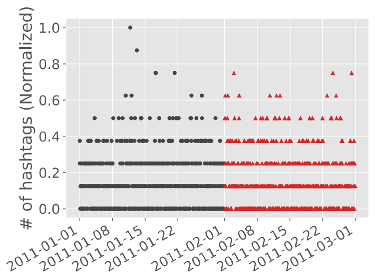
(a) #breakfast



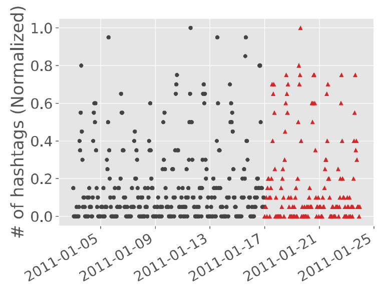
(b) #confessionhour



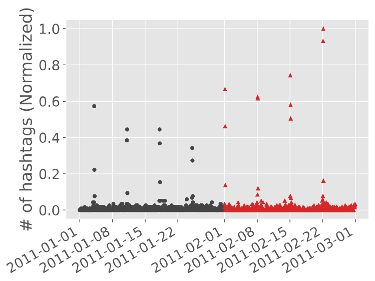
(c) #fail



(d) #fyi.



(e) #goodmorning



(f) #raw

Fig. 5.7. Hashtag time series problems. The dots represent the samples of the training set, while the triangles show the samples of the test set.

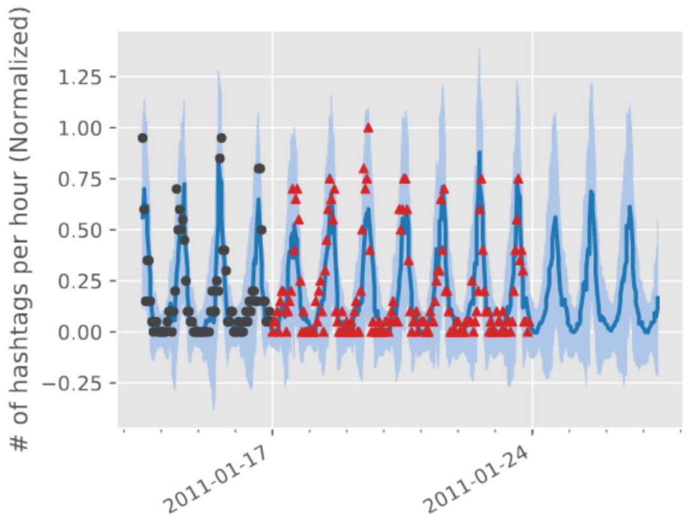


Fig. 5.8. Extrapolation of *#goodmorning* hashtag time series. The dots represent the last samples of the training set, while the triangles show the samples of the test set. The prediction given by a GP model with a kernel learned by the EvoCov-LML method is illustrated with a continuous blue curve for the mean, and the light blue shadow shows 3 times the standard deviation.

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.

	PS		EvoCov-NLPD		EvoCov-LML	
	Best	Mean	Best	Mean	Best	Mean
#breakfast	1.019	**1.045**	1.00*	1.090	1.060	1.148
#confessionhour	320.167	320.190	1.00*	**97.229**	1.017	197.542
#fail	1.008	1.363	1.00*	**1.034**	1.007	1.096
#fyi	1.019	1.050	1.024	1.041	1.00*	**1.001**
#goodmorning	1.00*	**1.058**	1.317	1.527	1.003	1.103
#raw	1.004	**1.365**	1.155	1.754	1.00*	1.677
Mean		54.345		**17.279**		33.928
Median		1.211		1.309		**1.125**

Table 5.6. 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 mean and the best results are illustrated. The best results on average are shown in bold, while the best results among all methods are highlighted with an asterisk.

Table 5.7 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.

	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

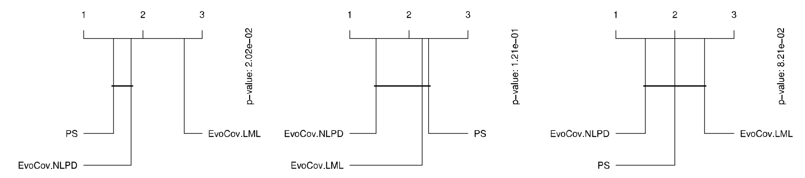
Table 5.7. Number of hyperparameters for each extrapolation problem and algorithm. The noise hyperparameter is also considered. In our approaches, the average number of hyperparameters is shown.

In order to assess whether there are significant differences between the ad hoc PS kernel and EvoCov approaches, we applied Friedman's test to their RMSE results in each problem. In *#breakfast*, *#fyi* and *#goodmorning* hashtags, significant differences can be observed ($\alpha = 0.05$. p-values are shown in Figure 5.9). Following the same procedure as in Section 5.5, we applied a post-hoc test based on Friedman's test, and adjusted the results with Shaffer's correction (Shaffer, 2012). In Figure 5.9, the results of this post-hoc testing are shown. While in *#breakfast*, the PS kernel and EvoCov-NLPD present significantly better results than EvoCov-LML, in *#goodmorning* hashtag, the PS kernel and EvoCov-LML outperform the EvoCov-NLPD approach. Finally, it is worth mentioning that EvoCov-LML is the best choice in *#fyi*.

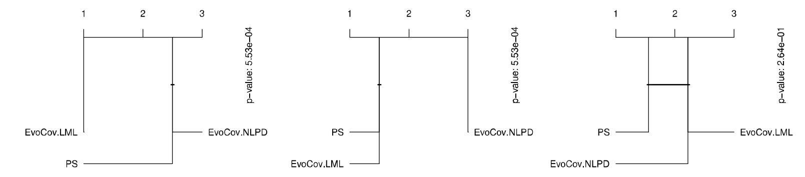
The PS kernel is able to hold the best results in three out of six problems, although the results only show statistical differences with EvoCov-LML in *#breakfast*, and with EvoCov-NLPD in *#goodmorning*. Moreover, the EvoCov approaches, without any knowledge about the problem, are able to obtain similar 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.

5.9 Conclusions

Overall, these experiments enabled the evaluation of the GenProg variant proposed in this dissertation, being able to improve the state-of-the-art results in the application of GPs to time series extrapolation. We have also provided valuable insights about the suitability and performance of several metrics for hyperparameter optimization in extrapolation problems.



(a) breakfast (b) confessionhour (c) fail



(d) fyi (e) goodmorning (f) raw

Fig. 5.9. Critical difference diagrams for the PS ad hoc kernel compared to EvoCov-LML and EvoCov-NLPD in hashtag prediction problems. The methods are ordered following the results in their ranking. The metrics with no significant differences among them are matched with a straight line.

We can conclude that it is neither necessary to use kernels designed by an expert nor to create kernels by composition of a priori defined ones to solve time series extrapolation problems with GPs. It is possible to learn simpler and better kernels than those that rely on previous knowledge through the evolution of expressions based on basic mathematical operations.

6

Automatic GP kernel learning for Natural Language Processing

6.1 Introduction

In the previous chapter, our proposal for evolving kernels was tested on time series extrapolation. In this chapter, we extend our experiments to another field of interest, Natural Language Processing (NLP), extending the functionality of the EvoCov algorithm to deal with multi-objective problems.

As in many other areas, GPs have been successfully applied to different NLP tasks. For instance, in Sentiment Analysis (SA), which consists of identifying or classifying the opinions contained in written or spoken language in an automated manner, several authors have shown promising results inferring the latent sentiments by means of GPs (Beck et al., 2015; Beck and Cohn, 2017; Beck, 2017b). Similarly, in Machine Translation Quality Estimation (QE), many studies have relied on GPs to provide post-editing work estimations (Cohn and Specia, 2013; Shah et al., 2013; Beck et al., 2016). GPs have been also used in text classification (Polajnar et al., 2011), where the goal is to assign predefined labels to the text.

Nevertheless, the use of GPs to solve NLP problems also poses some challenges. First, as stated throughout this dissertation, choosing the best performing kernel is not an easy task. In most of the applications of GPs to NLP, the choice of the most appropriate kernel was made a-priori, based on preliminary experiments or previous results. In early studies, the SE kernel was the most commonly applied one. In the work done by Specia et al. (2013), post-editing effort was predicted by using this kernel. Shah et al. (2013) also relied on the SE kernel to select the most relevant features for QE using GPs. In more recent studies, although Beck et al. (2016) pointed out that the results of Matern52 and Matern32 kernels were very similar to those of the SE kernel in Machine Translation QE, in SA, Matern kernels showed better results than the SE (Beck, 2017a).

Secondly, regarding the metric used to measure the quality of the kernels, there is no single criterion. While in the SA literature Pearson's Correlation Coefficient (PCC) (Pearson, 1895) has been used, this metric does not provide insights about the uncertainty that GPs are able to model. As the GPs offer a probabilistic prediction about emotions, metrics such as the LML (Rasmussen and Williams, 2006), or the NLPD (Quiñonero-Candela et al., 2006), could seem better suited to choose between GP models. Similarly, Machine Translation QE tasks also present a similar issue, where different metrics have been described to measure the quality of the translation (Specia, 2011), such as, post-editing score, Human Translation Edit Rate (HTER) or post-edit time.

A third aspect to consider in NLP tasks is the method selected to extract features from the text. Specia et al. (2013) proposed a Machine Translation QE framework where some features were extracted relying on expert knowledge. However, as feature engineering can be highly time-consuming (Specia et al., 2013), several works have tried to automate the feature extraction process. For example, Beck et al. (2015) applied structural kernels to emotion analysis and QE (Beck, 2017a). Other authors have proposed sentence embeddings to extract the features from the text. Yankovskaya et al. (2019) used this type of vector representation of the source and automatically translated texts to successfully predict the post-editing effort based on a neural-based regression model.

Taking into account these three issues, in this chapter, we extend the previously introduced EvoCov algorithm in order to efficiently solve this kind of task. Thus, we automatically evolve ad hoc kernels by means of a multi-objective approach of EvoCov that considers multiple NLP metrics simultaneously, with the aim of producing a diverse set of kernels that is also balanced according to the criteria represented by these metrics. In addition, sentence embedding techniques are used for extracting the features from the text, taking a step towards a fully automated framework for solving SA and QE tasks.

The chapter is organized as follows. First, the proposed algorithm is presented in Section 6.2. Then, in Section 6.3, the automatic techniques commonly used to extract sentence embeddings are described. In Sections 6.4 and 6.5, we show the experiments conducted regarding SA and QE problems respectively. Finally, in Section 6.6, the conclusions of the chapter are detailed.

6.2 Evolving multi-objective kernel functions

As mentioned in the introduction, in some cases the use of more than one metric can be advisable or even necessary. Thus, in this chapter we develop a multi-objective variant of the EvoCov algorithm (introduced in Section 4.3), called MOECov.

The description of MOECov is shown in Algorithm 5. As in EvoCov, first, an initial population of N kernels is generated, limiting the depth of the kernel expressions (maximum d_{max}, minimum d_{min}). At each generation, the whole population is evaluated and the relative improvement ($relimprov$) is calculated. If the relative improvement in the current population is greater than a threshold β, the S best individuals are selected and an offspring population of

$N - S$ new individuals is randomly generated by applying a mutation (with probability p_m) or a crossover (with probability p_{cx}) operator for each individual. Otherwise, the current population is replaced by a randomly generated one. After repeating this procedure for G generations, the last population is evaluated and the best individuals, taking into account the balance between all the metrics, are returned.

Algorithm 5 MOECov algorithm

```
 1: procedure MOECov($N$, $G$, $O$, $S$, $p_m$, $p_{cx}$, $\beta$, $d_{min}$, $d_{max}$)
 2:     $pop$ = GenRandPop($N$, $d_{min}$, $d_{max}$)
 3:     for $j$ in $O$ do
 4:         $bestfit_{-2,j} = \infty$
 5:         $bestfit_{-1,j} = \infty$
 6:     end for
 7:     $all = pop$
 8:     $i = 0$
 9:     while $i < G - 1$ do
10:         Evaluate($pop$)
11:         for $j$ in $O$ do
12:             $bestfit_{i,j}$ = GetBestFitness($pop$, $j$)
13:             $relimprov_j = \frac{\frac{1}{2}(bestfit_{i-2,j} + bestfit_{i-1,j}) - bestfit_{i,j}}{|bestfit_{i,j}|}$
14:         end for
15:         if $\beta < MAX(relimprov)$ then
16:             $sel$ = Select($pop$, $S$)
17:             $offspring$ = Variate($sel$, $N - S$, $p_m$, $p_{cx}$)
18:         else                                              ▷ Restart procedure
19:             $sel = \emptyset$
20:             $offspring$ = GenRandPop($N$, $d_{min}$, $d_{max}$)
21:             for $j$ in $O$ do
22:                 $bestfit_{i-1,j} = \infty$
23:                 $bestfit_{i,j} = \infty$
24:             end for
25:         end if
26:         $pop = sel \cup offspring$
27:         $all = all \cup offspring$
28:         $i = i + 1$
29:     end while
30:     Evaluate($pop$)
31:     $best$ = SelectBest($all$)
32:     return $best$
33: end procedure
```

MOECov algorithm differs from EvoCov in several aspects. First, the evaluation step has been adapted to consider more than one metric. In Sections 6.4.1 and 6.5.1, the metrics used for each particular problem are introduced.

Secondly, the SELECT function in Algorithm 5, that selects the most promising individuals, has been adapted by adding Pareto operators (Pareto set). Inspired by the NSGA2 algorithm (Deb et al., 2000), the population is divided into groups of non-dominated solutions corresponding to the Pareto fronts by iteratively selecting the Pareto set, excluding these solutions from the general set, and repeating the operation with the rest of individuals. The individuals in each group are sorted by crowding distance, and finally, the best S individuals are chosen. At the end of the evolution process, the SELECTBEST function selects the non-dominated individuals among all the expressions created during the whole process. Thus, the practitioner is provided with a set of Pareto optimal solutions to choose from depending on the requirements of the problem.

Also note that the restart procedure has been altered. A restart will be made only if none of the objectives have been improved. In addition, this improvement is now checked taking into account the previous 2 generations, reducing the number of restarts.

Regarding the grammar, the kernel functions are encoded according to the basic mathematical expression grammar described in Section 4.2. However, we limited the search to stationary kernels (see Equation (2.3)), as in the preliminary experiments we conducted, we found out that, when sentence embeddings are used as the input of the GPs, the kernels that include the *dot_product* operator are hardly ever chosen by the GenProg algorithm.

6.3 Sentence embeddings

For the problems presented in this chapter, SA and QE, as well as for NLP problems in general, a previous step is needed: extract usable features from the input data.

In the initial NLP approaches, expert domain knowledge was used to extract relevant information from text and produce vector representations (Callison-Burch et al., 2011). For instance, Specia (2011) extracted 80 shallow and Machine Translation independent features from the source sentences, their corresponding automatic (before post-editing) translations, and monolingual and parallel corpora. A selected set of these features, comprising only 17 features, was used by Specia et al. (2013) to predict translation-effort.

Recent advances in artificial neural networks have allowed the feature engineering process to be automated, reducing expert knowledge dependency. In word embeddings, each word is assigned a vector of continuous values using a neural network. Some of the first embedding methods, and still among those most used, are the Continuous Bags of Words (CBOW) and the Skip-gram models (Mikolov et al., 2013; Pennington et al., 2014).

Nevertheless, in the SA and QE tasks presented in this dissertation, sentence vector representations are required. In order to obtain these vector representations, there are two main approaches: word embedding aggregation (Le and Mikolov, 2014) and the direct sentence embedding encoding (Kiros et al., 2015).

In word embedding aggregation methods, vector representations of each word in the sentence are obtained. Then, a function that combines the embedding representations of all the words in a single vector is computed to generate the sentence embedding. The usual approach when combining embeddings from words in a sentence is to compute the average. This is the procedure used by Beck (2017a), who used 100-dimensional GloVe embeddings (Pennington et al., 2014) as the representation of choice for mapping texts to emotion scores using GP regression.

In addition to the word embedding aggregation methods, direct sentence embedding encodings can be also obtained by means of pre-trained bidirectional Long Short-term Memory (LSTM) artificial Recurrent Neural Networks (RNNs). This approach was successfully applied in Machine Translation QE using a neural-based regression model (Yankovskaya et al., 2019).

6.4 Sentiment Analysis

Once the general framework has been explained, the next two sections are devoted to each of the problems described initially: Sentiment Analysis (SA) (this section), and Quality Estimation (QE) (Section 6.5).

Sentiment Analysis (SA) is an automated process that infers the opinion or feeling from a piece of text. It can be considered as a particular type of semantic annotation of the text. SA is a very complex problem due to several factors, including the ambiguity of human language, the large variability in the use of terms across individuals, and the complexity of grammatical rules. However, the availability of large text corpora and the usefulness of mining these corpora, e.g., for opinion mining related to products, services or politics (Pang and Lee, 2008), have contributed to developing more advanced machine learning algorithms for this task.

One of the directions of extending SA methods is to go beyond text categorization in positive or negative classes, to a more fine-grained emotion annotation (Ortigosa-Hernández et al., 2012; Strapparava and Mihalcea, 2007). This could be done by extending the number of classes in which a text is classified, but also by allowing a continuous value of the strength in which the sentiment is manifested in the text. A subject is requested to evaluate, in a range $[0, 100]$ several sentiments (e.g., joy, fear, etc.) in a text. In the sentence "Alonso would be happy to retire with three titles", for example, joy and sadness feelings are mixed.

The problem we address in this chapter is to automatically estimate the intensity of a given sentiment from the analysis of the text. This problem is posed as a supervised regression problem in which a number of text samples are available together with their corresponding sentiment value. After extracting the sentence embedding of each sample using a word embedding aggregation method, we use a GP model to predict the sentiment value present in the text.

In SA tasks, Pearson's Correlation Coefficient (PCC) is one of the most used metrics to measure the quality of the prediction. This metric divides the covariance of two variables by the product of their standard deviations. In the case of GPs, PCC can be measured as follows:

$$PCC(\boldsymbol{\mu}_*, \mathbf{f}_*) = \frac{\sum_{i=1}^{n_*}(f_{*i} - \hat{f}_*)(\mu_{*i} - \hat{\mu}_*)}{\sqrt{\sum_{i=1}^{n_*}(f_{*i} - \hat{f}_*)^2 \sum_{i=1}^{n_*}(\mu_{*i} - \hat{\mu}_*)^2}} \tag{6.1}$$

where n_* is the number of test samples, and $\mathbf{f}_*$ corresponds to their function values. $\boldsymbol{\mu}_*$ indicates the predictive mean of the GP model. $\hat{f}_*$ and $\hat{\mu}_*$ are the mean values of their corresponding $\mathbf{f}_*$ and $\boldsymbol{\mu}_*$ vectors.

Note that PCC does not depend on the variance of the GP model, and thus, it does not take advantage of all the information provided by the probabilistic prediction. On the other hand, the NLPD, introduced in Section 5.5, is more informative about the performance of the GP model, as it measures the likelihood of the predictive Gaussian distribution to the data.

6.4.1 Fitness evaluation for Sentiment Analysis

In the SA setting of MOECov, it is desirable to ensure a high correlation between the prediction and the actual data, as it has been already investigated with other models with the PCC metric. At the same time, we want to take advantage of the GPs to provide an accurate probabilistic prediction in terms of NLPD. As can be seen, although PCC and NLPD metrics are related, they evaluate the quality of the kernel from different perspectives. In order to balance the kernel learning process between these two objectives, when applying to SA tasks, the EVALUATE function in Algorithm 5 measures both the PCC and NLPD metrics.

In addition, we have observed that the complexity of the kernels, in terms of number of primitives or depth, increases the evaluation time and the risk to overfit. Thus, in order to obtain simple kernels that are efficient to evaluate, we also take the evaluation time as the third objective, indirectly discouraging kernels with large expressions.

The fitness evaluation of these metrics can be represented as shown in Figure 6.1. Before carrying out these evaluations, the hyperparameters of the kernel function are set according to the LML as explained in Section 2.2.2.2. Then, PCC and the NLPD are evaluated by dividing the training set into 3 cross-validation folds. The results in each fold are averaged to obtain the actual fitness of each kernel. Finally, the time spent computing both metrics in each fold is added as the third objective.

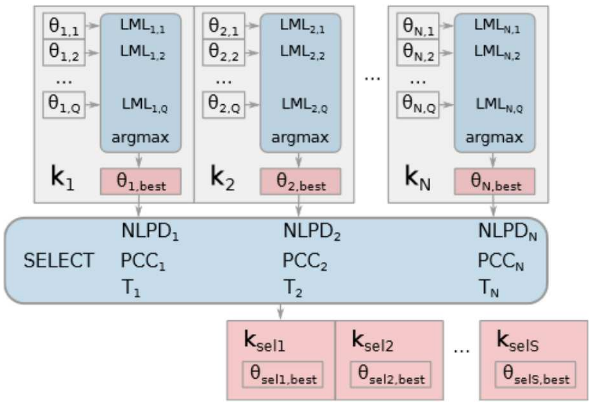


Fig. 6.1. Fitness evaluation in MOECov for SA.

6.4.2 Sentiment Analysis experiments

The goal of this experiment is to evaluate the performance of the proposed MOECov algorithm for the task of sentiment prediction from text. First, we introduce the problem benchmark and word embeddings. Then, we describe the parameters used by the algorithms and explain the characteristics of the experimental framework. Finally, we present the numerical results obtained from the experiments and discuss these results.

6.4.2.1 Benchmark problems and word-embeddings

We use the SemEval2007 Affective Text shared task dataset (Strapparava and Mihalcea, 2007)[1], following the work carried out by Beck (2017a). In this dataset, news headlines were manually annotated by experts, assigning to each text a degree of presence for each six Eckman (Ekman, 1993) emotions: anger, disgust, fear, joy, sadness and surprise. In the original work where this dataset was introduced, texts were divided into “dev” and “test” datasets. For the experiments presented by Beck (2017a), the two sets were combined and further divided into 10 folds used for cross-validation. We also use this 10-fold partition to evaluate our algorithms.

In order to compute a representation for each text, punctuations in each headline were removed, tokenized (Bird et al., 2009), and case ignored. From the resulting text, the word-vector representation of each word was obtained using the 100-dimensional GloVe embeddings (Pennington et al., 2014)[2]. After deleting the words that were not found in the embedding, the representation of each headline was computed as the average of the words.

[1] Available at `https://web.eecs.umich.edu/~mihalcea/downloads.html#affective`

[2] Available at `https://nlp.stanford.edu/projects/glove/`

6.4.2.2 Experimental setup

Our experiments consist of learning a kernel for a GP regressor that predicts a particular emotion based on sentence embeddings. We compare the MOECov algorithm with different variants of a-priori defined kernel methods, which were shown by Beck (2017a) to produce good prediction results for the six emotions previously described.

The parameters used by the evolutionary algorithm were chosen after some preliminary experimentation. $N = 38$ kernels are generated in each of the $G = 65$ generations. These newly generated kernels are evaluated and the best $S = 9$ kernels are selected. In order to apply the restart procedure, a threshold $\beta = 1\text{e}{-5}$ is set. Besides, variation operators are applied with a mutation probability $p_m = 0.6$ and a crossover probability $p_{cx} = 1 - p_m = 0.4$.

In spite of having a non-dominated set of kernels as a result of the MOECov algorithm, in order to illustrate of the results of this experimentation, we select the best kernel in terms of the LML metric during the training step.

The hyperparameters of the standard kernels are learned according to the LML metric. Then, for each of these kernels and selected kernel in MOECov, we evaluate the PCC and NLPD on the test dataset and measure the time to compute this evaluation. Due to the stochastic nature of the MOECov algorithm and the hyperparameter optimization of the standard kernels, for every configuration, the kernel search process was repeated 30 times along 10 random cross-validation folds.

6.4.2.3 Comparison between MOECov and standard kernels

Table 6.1 shows the average PCC metric of the best solution obtained by the algorithms in the 30 experiments. We remark that these values have been computed on the test data, which was not used for learning the GenProg programs. The results for the NLPD metric are shown in Table 6.2. In both tables, the best average value obtained for each of the sentiments are highlighted.

The analysis of the tables reveals that, in terms of the average fitness, MOECov improves all the other kernels for both metrics in most of the cases. For the PCC metric, only in the *disgust* dataset, MOECov was not able to outperform the Matern52 (M52) kernel. For the NLPD metric, MOECov outperforms all the algorithms for all the sentiments.

Among the standard kernels, Matern52 seems a better choice than the others, as it gets the second best result on average in the rest of the problems according to the PCC metric. Matern52 is also the second best choice in NLPD, only surpassed by the SE kernel in the *fear* dataset, while the results of the Matern32 (M32) and the SE kernels are similar. As can be appreciated in the tables, the Linear kernel (LIN) is the worst performing kernel according to the average results.

We conducted a statistical test to assess the existence of significant differences among the algorithms. For each metric and emotion, we applied Friedman's test (Friedman, 1937) and we found significant differences in every

	LIN	M32	M52	SE	MOECov
anger	0.58592	0.63556	0.64010	0.62629	**0.64690**
disgust	0.44828	0.52492	**0.52782**	0.50111	0.52456
fear	0.68056	0.72810	0.73059	0.72737	**0.73555**
joy	0.53832	0.55775	0.57459	0.56341	**0.59158**
sadness	0.63625	0.67148	0.68205	0.67876	**0.69710**
surprise	0.40311	0.45416	0.45647	0.43758	**0.46751**

Table 6.1. Mean results for PCC metric (the higher the better). The best results are shown in bold.

	LIN	M32	M52	SE	MOECov
anger	3.94141	3.92037	3.91162	3.93041	**3.91084**
disgust	3.81476	3.78148	3.77491	3.80068	**3.77419**
fear	4.16636	4.10615	4.10060	4.09860	**4.07623**
joy	4.34633	4.32588	4.30362	4.32737	**4.29549**
sadness	4.31082	4.28845	4.27618	4.28176	**4.24454**
surprise	4.06292	4.04524	4.04519	4.05110	**4.02712**

Table 6.2. Mean results for NLPD metric (the lower the better). The best results are shown in bold.

comparison ($\alpha = 0.05$. p-values are shown in Figure 6.2 and Figure 6.3). Then, for each configuration, we applied a post-hoc test based on Friedman's test as suggested by Demšar (2006), and adjusted the results with Shaffer's correction (Shaffer, 2012).

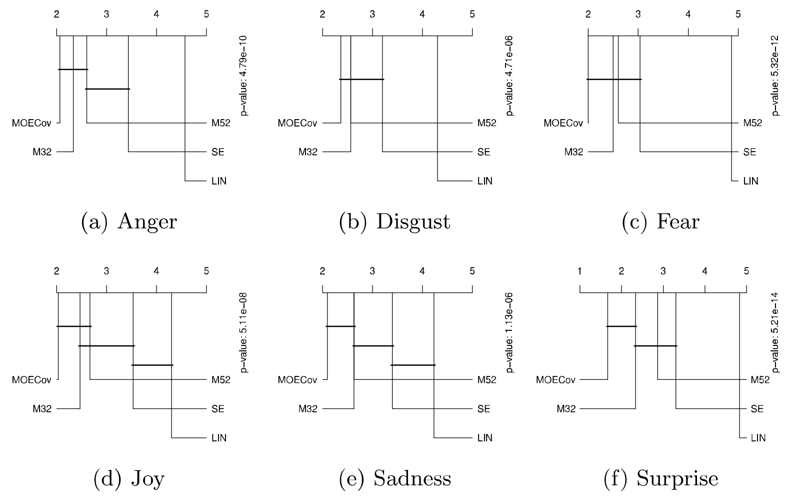
The results are shown in Figure 6.2 and Figure 6.3. They confirm a coherent pattern where MOECov is the best performing algorithm for most of the datasets. However, according to this test, the differences between MOECov and Matern52 are not significant for most of the datasets.

In evaluating these results it is important to take into account that the kernels produced by MOECov have been generated completely from scratch, with no prior knowledge of the existing kernels. The algorithm is able to evolve a well performing kernel starting from elementary mathematical components.

The computational overhead introduced by MOECov should be also considered. Most of the effort is spent evaluating kernels. The rest of the calculations required by the algorithm are negligible. Therefore, in this experiment, MOECov has required approximately 2500 times more computational time than fixed kernels. In terms of the evaluation time, once the kernel has been learned, the effort required to compute the kernels generated by MOECov is similar to the standard kernels.

6.4.2.4 Analysis of the MOECov evolution

One distinctive feature of our approach is that we simultaneously optimize different characteristics of the kernels. In order to determine whether the



(a) Anger (b) Disgust (c) Fear

(d) Joy (e) Sadness (f) Surprise

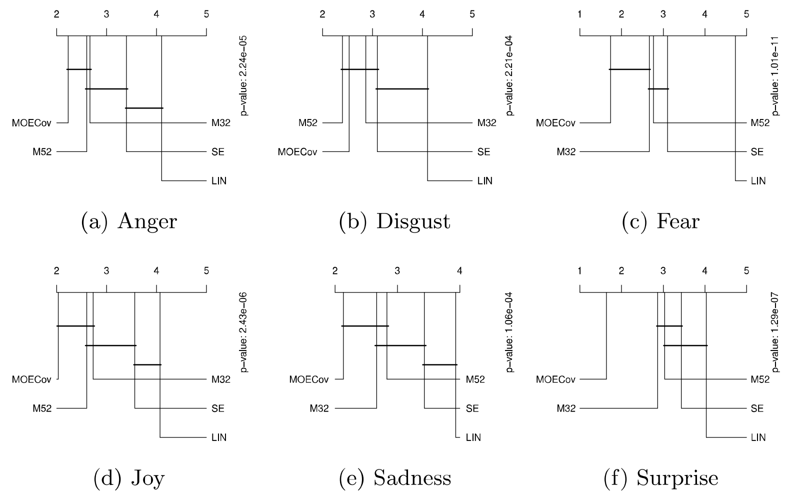
Fig. 6.2. Critical difference diagrams for the PCC metric. The kernels are ordered following their rankings. The metrics with no significant differences between them are matched with a straight line.

multi-objective approach effectively leads to the creation of more efficient kernels in terms of the accuracy for the prediction task and in terms of efficiency, we analyze the fitness distribution of the solutions in the first and last population of MOECov for the *anger* dataset. These results are shown in Figure 6.4, where, for the sake of the visualization, only one (illustrative) execution of the algorithm is shown.

In Figure 6.4 we represent the scatter plots for each possible pair of objectives. As can be appreciated in the figures, from the first to the last population there is an improvement in the values of the objective values for the PCC and NLPD metrics. However, the computational time actually increases from the first to the last population. This result is not surprising since it is expected that the increase in accuracy of the trees is achieved by also augmenting their complexity. In this scenario, using the time as a third objective can serve to counteract the useless complexity gain of the programs.

6.4.2.5 Transferability of the evolved kernels

An important question to analyze is whether the kernels evolved by MOECov are only valid for the sentiment datasets in which they have been learned or they can also be used in other datasets. This question can be framed on the general research that investigates the transferability of solutions produced by evolutionary algorithms (Iqbal et al., 2016; Garciarena et al., 2018; Santana et al., 2012). In order to answer this question, we have used the best kernels



(a) Anger (b) Disgust (c) Fear

(d) Joy (e) Sadness (f) Surprise

Fig. 6.3. Critical difference diagrams for the NLPD metric. The kernel are ordered following their rankings. The metrics with no significant differences between them are matched with a straight line.

learned for the *anger* dataset to make predictions in other datasets. This can be considered as a transfer learning scenario in which the *anger* dataset is the source domain and all the other datasets serve as target domains. Notice, that in this particular example we do not recompute the hyperparameter values for the kernels. We simply apply the kernels *as they are* to the target datasets.

Figure 6.5 and Figure 6.6 respectively show the distributions of the objectives values for the PCC and NLPD metrics. In the figures, *MOECov_anger* indicates the kernels learned using the *anger* dataset. Notice that all the other algorithms have been learned using (training) data for each target dataset. The analysis of the figures indicate that the transferability of the kernels depends on the type of metric used. Results for PCC are at least as good as those obtained with the other kernels. However, for the NLPD metric, the results are slightly worse.

The main conclusion from these experiments is that the kernels evolved for predicting some sentiment can be also useful to predict other sentiments. This means that the type of transformations that make a kernel a good predictor are similar across sentiment domains.

6.5 Machine Translation Quality Estimation

Machine Translation QE has attracted a lot of attention within NLP literature. Depending on the quality of the automatic translation, human post-

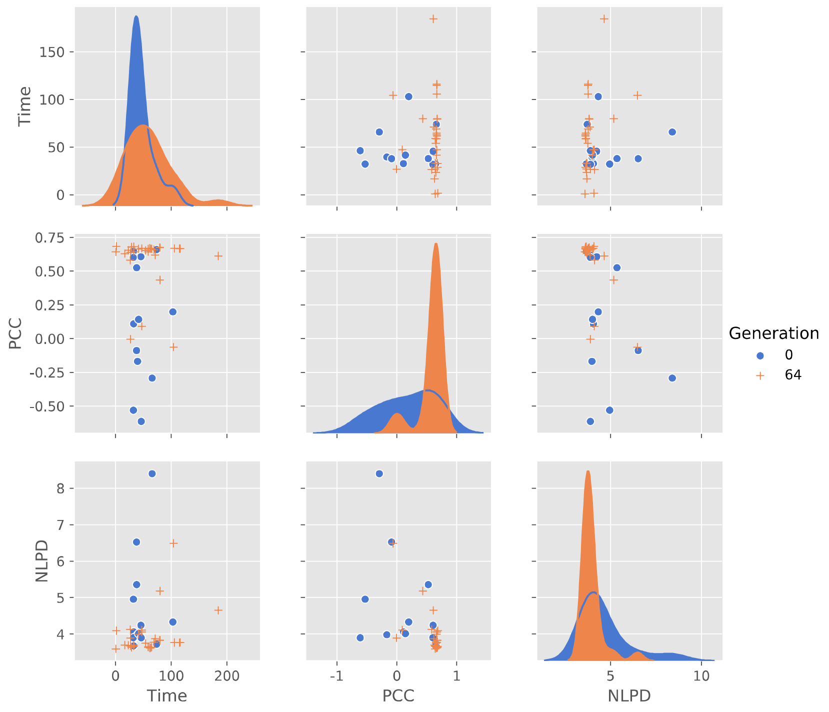


Fig. 6.4. Distribution of the objective values in the first and last population of MOECov for one execution of the algorithm in the *anger* dataset.

editing work is often required, and it is desirable to have an estimation of the cost of the editing process (in terms of time, effort, or editing distance). Based on the work done by Specia (2011), we measure the post-editing effort in terms of the following metrics: post-editing score, Human Translation Edit Rate (HTER) and post-edit time.

In order to score the post-editing effort, Specia (2011) asked the translators to post-edit each sentence and rate the post-editing effort according to the following options:

1. Requires complete retranslation.
2. Requires some retranslation, but post-editing is still quicker than retranslation.
3. Very little post-editing needed.
4. Fit for purpose.

Another metric used to evaluate the translation quality is the edit distance between the automatic translation and its post-edited version. This

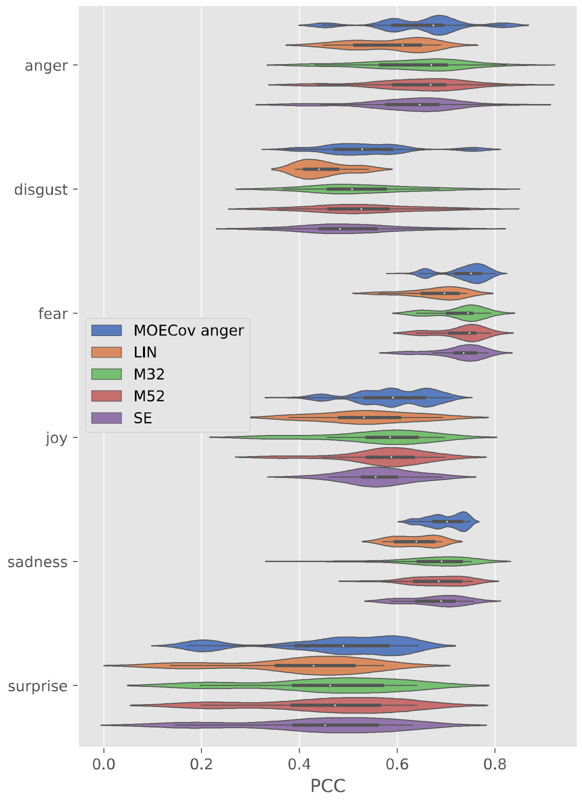


Fig. 6.5. Results of the transferability experiments for the PCC metric. Each colored shape shows a kernel density estimation of the distribution. A boxplot of the results is shown inside. Higher is better.

is computed using the HTER (Snover et al., 2006). HTER is defined as $HTER = \frac{e}{pe_w}$, where pe_w is the number of words in the sentence and e refers to the number of edits, which can be: standard insertion, deletion and substitution of single words, as well as shifting of word sequences.

Finally, Specia (2011) computed the post-edit time using the average number of seconds required by two translators to post-edit each word in the sentence, i.e., the number of seconds to post-edit the sentence normalized by the number of words in that sentence.

6.5.1 Fitness evaluation for Quality Estimation

In order to apply the MOECov algorithm in QE, for each solution (kernel), the BIC value related to the prediction of each of the QE metrics is computed. As a

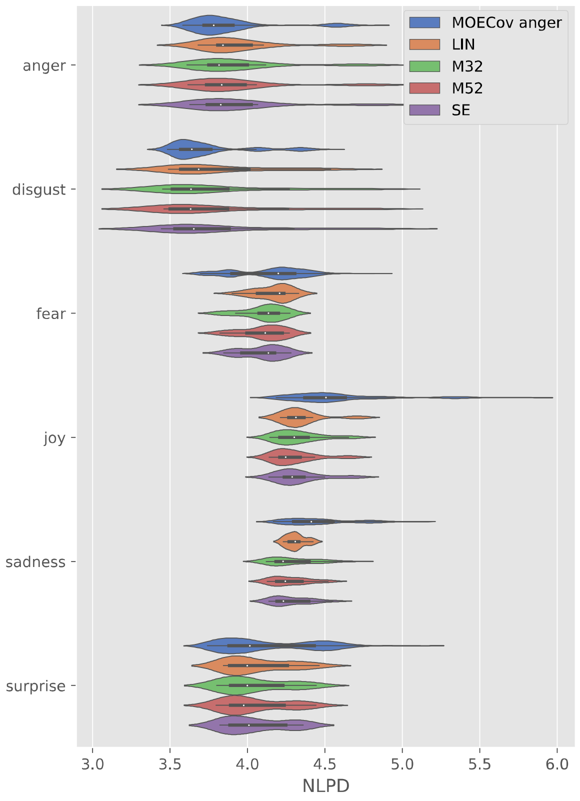


Fig. 6.6. Results of the transferability experiments for the NLPD metric. Similar to the previous figure, a boxplot is shown inside each colored shape that shows the estimation of the distribution. Lower is better.

consequence, the hyperparameters are optimized three times, one for each QE metric. Both learning procedures, the selection of the best hyperparameters for each metric, and the selection of the best kernel given these hyperparameters, are illustrated in Figure 6.7.

6.5.2 Quality Estimation experiments

The experiments for the QE problem consist of learning a GP regressor based on a combination of source and automatically translated embeddings, in order to predict one or several metrics. Particularly, we would like to know if the evolution of GP kernels by means of the EvoCov algorithm can improve the results of standard stationary kernels. In addition, we test the ability of the MOECov algorithm to learn kernels that can be adapted to several Quality

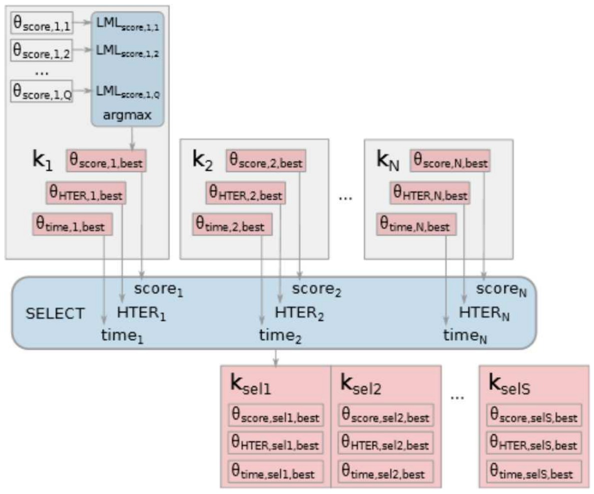


Fig. 6.7. Fitness evaluation for QE. In the top left of the figure the hyperparameter optimization for k_1 and score metric is shown. This optimization is carried out for each kernel and objective. Then, the best kernels are selected as shown in Section 6.2, taking into account the performance of the kernel in each of the objectives.

Estimation (QE) metrics. Finally, we also investigate the best approach to conform the sentence embeddings for translation post-editing effort estimation, comparing the word-vector aggregation methods to the direct computations of the sentence embedding.

6.5.2.1 Datasets and embeddings

To carry out these experiments, we use the datasets originally proposed by Specia (2011):

1. en-es news-test2010: 1000 English news sentences and their translations into Spanish using Moses software (Koehn et al., 2007). We decided to use 600 sentences for training and 400 for testing.
2. fr-en news-test2009: 2525 French news sentences and their Moses translations into English. In order to have similar computation times as in the *en-es news-test2010* dataset, 600 sentences were used for training and those remaining for testing.

Punctuation was removed from the sentences, the text was tokenized, and case was ignored. A zero-vector representation was assigned to the words missing in the dictionary. For the HTER metric, an equal cost was used for all edits (Specia, 2011).

For each sentence and language, several sentence and word embeddings were extracted. For the first dataset, we used English GloVe word embeddings

(Pennington et al., 2014) of 50 and 300 dimensions. To represent sentences in the target language, we applied the Spanish word embeddings of size 100 computed from the CoNLL17 corpus and available at the NLPL word embeddings repository[3]. For the second dataset, French word embeddings of sizes 300 and 52 were used. The 300-dimensional embeddings[4] were trained on data from Common Crawl and Wikipedia, using fastText (Grave et al., 2018), while the 52-dimensional word embeddings[5] were trained using Twitter messages (Deriu et al., 2017). The 100-dimensional word embeddings used for the target language are those provided by GloVe.

We computed the sentence embeddings for the source and target languages (before post-edition). In order to create the sentence embeddings based on the word-vectors, we examined several methods to aggregate them. Apart from the mean (*mean*) function, we used another two functions: the maximum (*max*) of all word embeddings (maximum value of each embedding component across all word embeddings), and the standard deviation (*std*) of word embeddings. The rationale behind these two choices is to determine whether "extreme values" or "amount of sentence variability" are better predictors of QE effort.

Apart from the sentence embeddings computed by aggregating word-vectors, we also computed the sentence embeddings using LASER sentence embeddings (Artetxe and Schwenk, 2019). As these bidirectional LSTM artificial RNN were pre-trained for different languages, we applied them for source and automatically translated sentences with 1024 dimensions.

Finally, all these sentence embeddings from the source and target languages were concatenated to produce the vector representations that are used by the GP model. The embedding combinations tried in the experiments are detailed in Table 6.3.

6.5.2.2 Experimental setup

In the GenProg algorithms used in these experiments, EvoCov and MOECov, $N = 38$ kernels were generated in each of the $G = 65$ generations. We used a similar setting to the one followed in the SA experiments, selecting the best $S = 9$ kernels each generation, using a threshold of $\beta = 1\text{e}{-4}$ to restart the procedure, and applying the variation operators with a mutation probability $p_m = 0.6$ and a crossover probability $p_{cx} = 1 - p_m = 0.4$.

Learning a GP regressor implies optimizing the hyperparameters of the kernel. In order to carry out this optimization, $Q = 350$ evaluations of the LML were allowed for all the kernels evaluated during the experimentation, including the evolved and standard kernels. Since the local optimizer used for this optimization is a stochastic process, we ran 30 executions of the fitting process using the training data. For the standard kernels, this amounts to

[3] `http://vectors.nlpl.eu/repository/`
[4] `https://fasttext.cc/docs/en/crawl-vectors.html`
[5] `https://www.spinningbytes.com/resources/wordembeddings/`

Embedding types		Dataset	Source sentence	Automatically translated sentence
LASER embeddings	0-1024	*en-es*	Not used	1024-dim LASER
		fr-en	Not used	1024-dim LASER
	1024-0	*en-es*	1024-dim LASER	Not used
		fr-en	1024-dim LASER	Not used
	1024-1024	*en-es*	1024-dim LASER	1024-dim LASER
		fr-en	1024-dim LASER	1024-dim LASER
mean/max/std word-vector aggregations	0-100	*en-es*	Not used	100-dim NLPL
		fr-en	Not used	100-dim GloVe
	300-0	*en-es*	300-dim GloVe	Not used
		fr-en	300-dim fastText	Not used
	300-100	*en-es*	300-dim GloVe	100-dim NLPL
		fr-en	300-dim fastText	100-dim GloVe
	50/52-0	*en-es*	50-dim GloVe	Not used
		fr-en	52-dim tweet	Not used
	50/52-100	*en-es*	50-dim GloVe	100-dim NLPL
		fr-en	52-dim tweet	100-dim GloVe

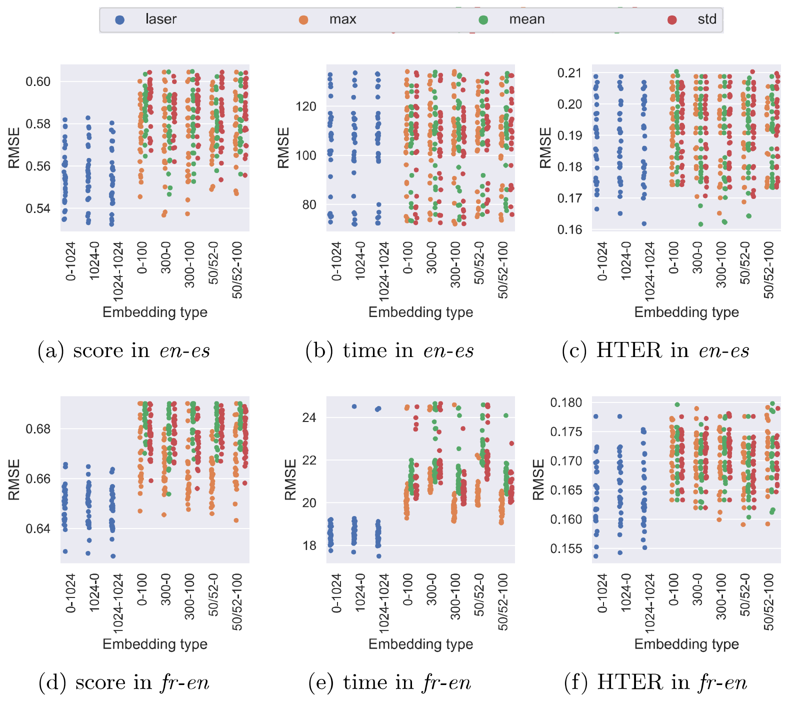
Table 6.3. Embedding types used in the experiments.

learning 30 different hyperparameter configurations of the same kernel. For the kernels evolved by means of EvoCov, this means obtaining 30 different kernels. Also, in order to evaluate the performance of the multi-objective variant of the algorithm, MOECov, we obtained 30 different non-dominated sets of kernels with their hyperparameters optimized for each metric. Finally, for each kernel, the quality of the prediction was measured by computing the RMSE between the known true metric values and the predictions.

6.5.2.3 Embedding type comparison

The aim of this experiment is to investigate whether the different ways to compute sentence embeddings influences the quality of the prediction. This experiment was carried out with the SE kernel, as the best-known representative of the standard kernels. Figure 6.8 shows the RMSE as computed in the test set with the *max*, *mean*, and *std* functions of the word-vectors, and the sentence embeddings provided by the LASER bidirectional LSTM. For each embedding type, several dimensions were tried, also testing the possibility of not including the source or the automatically translated sentence. Each dot in the plot represents the result in terms of RMSE for each of the 30 executions.

The analysis of Figure 6.8 reveals that there are no major differences in sentence embeddings for the time and HTER metrics in the *en-es news-test2010* dataset. However, for the score metric, LASER embeddings produce lower errors in the predictions than the word-vector aggregation functions. This effect is more pronounced in the *fr-en news-test2009* dataset for all metrics, where the LASER embeddings produce better predictions. Among the word-vector aggregation functions, obtaining the maximum value of the vector is the best option according to the results shown in the *fr-en news-test2009* dataset for the score and time metrics.



(a) score in *en-es* (b) time in *en-es* (c) HTER in *en-es*

(d) score in *fr-en* (e) time in *fr-en* (f) HTER in *fr-en*

Fig. 6.8. Comparison between the embedding types for a GP model with the SE kernel. Each dot represents the RMSE of each of the 30 executions for each embedding type. The embedding types are shown along the x axis, using colors to distinguish between LASER embeddings and mean/max/std word-vector aggregation methods. At the top, the results for the *en-es news-test2010* dataset for each metric can be found. At the bottom of the figure, the results for the *fr-en news-test2009* dataset are shown.

The numeric results of the experiment are presented in Table 6.4, where the average RMSE of all the 30 executions is shown. According to these average results, the configurations based on the LASER embeddings obtain the best results in all the metrics and datasets. The best results in the *en-es news-test2010* dataset for score and time metrics and in the *fr-en news-test2009* dataset for score metric are obtained by using both source and automatically translated embeddings. Regarding the word-vector aggregation methods, the influence of the number of dimensions of the selected embeddings is less noticeable than the choice of the aggregation function.

Embedding type		en-es HTER	en-es score	en-es time	fr-en HTER	fr-en score	fr-en time
LASER	0-1024	0.1883	0.5560	108.2696	**0.1642**	0.6495	19.1413
	1024-0	**0.1879**	0.5562	106.1376	0.1653	0.6487	**19.0268**
	1024-1024	0.1882	**0.5545**	**105.2971**	0.1645	**0.6476**	19.2748
max	0-100	0.1930	0.5899	108.4599	0.1845	0.6781	21.1592
	300-0	0.2212	0.5804	110.3525	0.1715	0.6642	21.9848
	300-100	0.2538	0.5796	107.4330	0.1887	0.6669	20.0665
	50/52-0	0.2555	0.5761	110.6683	0.1711	0.6612	20.9603
	50/52-100	0.2681	0.5826	106.6381	0.1946	0.6721	19.8703
mean	0-100	0.1929	0.5913	111.3924	0.1737	0.6862	21.5264
	300-0	0.1910	0.5817	111.2523	0.1693	0.6794	22.7502
	300-100	0.1897	0.5847	110.1628	0.1718	0.6858	21.6260
	50/52-0	0.1904	0.5852	112.0467	0.1686	0.6837	22.9924
	50/52-100	0.1912	0.5858	111.5610	0.1747	0.6940	21.8793
std	0-100	0.1946	0.5948	109.9629	0.1731	0.6767	21.8624
	300-0	0.1926	0.5906	107.9739	0.1711	0.6814	22.4847
	300-100	0.1930	0.5909	107.1841	0.1721	0.6740	20.7808
	50/52-0	0.1932	0.5871	112.1964	0.1694	0.6822	22.4158
	50/52-100	0.1929	0.5887	111.7536	0.1757	0.6770	21.2571

Table 6.4. Comparison between the embedding types for a GP model with SE kernel (mean RMSE from 30 executions). The best results are shown in bold.

In conclusion, the LASER embeddings are the best choice to compute the sentence embeddings for translation-effort prediction with GP models using the SE kernel. None of the word-vector aggregation methods has been able to obtain competitive results on average in the tested datasets and metrics. Also, including the source and automatically translated embeddings as inputs of the GP model is preferable according to the results of this experiment.

6.5.2.4 Learning kernels for QE

In this second experiment regarding QE, we compare the classical kernel models to the evolved kernels learned by the EvoCov algorithm. Figure 6.9 shows the RMSE on the test set of the GP models with these kernels. As they are the sentence representations that showed the best results in the previous experiment, we selected LASER embeddings (including the information of the automatically translated sentences) to carry out this second experiment. In addition, we also included the mean aggregation function since it is one of the most commonly reported aggregation methods in the literature.

The first remarkable fact in Figure 6.9 is that GP kernels show a smaller variance compared to the standard kernels, which can be a sign of convergence of the algorithm to the optimal kernel. Furthermore, it can be seen that the results in this experiment are consistent with the experiment shown in Section 6.5.2.3, as the LASER embeddings outperform the word-vector approaches, which is particularly evident for the *fr-en news-test2009* dataset. While the performance of the standard kernels is similar in all the databases and metrics, the evolved kernels show a better performance in the word-vector approaches.

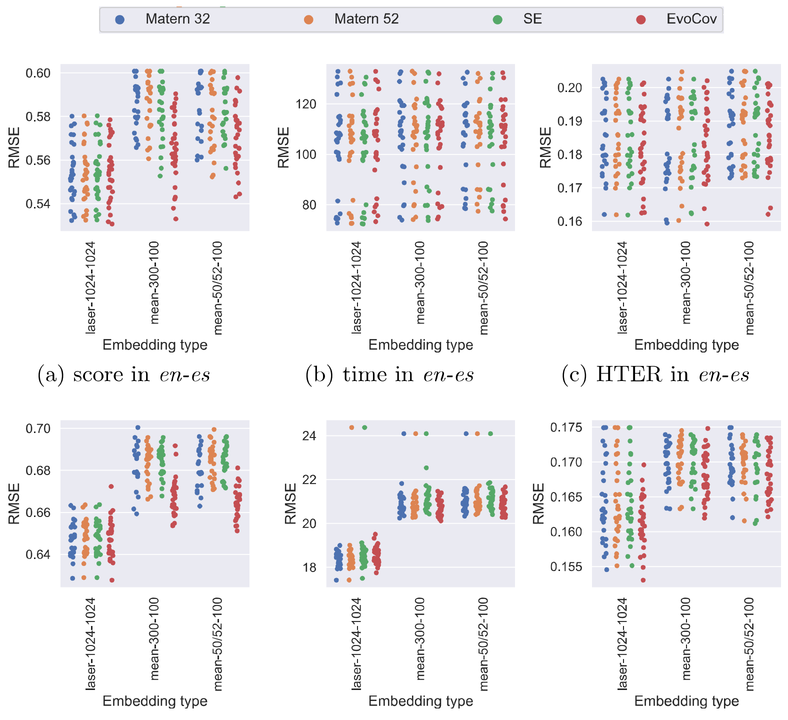


Fig. 6.9. Comparison between the EvoCov kernel learning method and the standard kernels. At the top, the results for the *en-es news-test2010* dataset for each metric can be found. At the bottom of the figure, the results for the *fr-en news-test2009* dataset are shown.

In Table 6.5 the average RMSE results are shown. It can be seen that the kernels learned by the EvoCov algorithm outperform the standard kernels on average in 4 out of 6 problems using the LASER embeddings. Nevertheless, for the score and time metrics in the *en-es news-test2010* dataset, the results are favorable to the Matern32 kernel. On the configurations where the sentence embedding were produced by aggregating the word-vectors, the best average results are achieved by the EvoCov algorithm in all the cases.

		en-es			fr-en		
Emb. type	Method	HTER	score	time	HTER	score	time
LASER 1024-1024	EvoCov	**0.1824**	0.5552	105.9512	**0.1615**	**0.6471**	**18.5265**
	M32	0.1883	**0.5542**	**105.1188**	0.1645	0.6474	19.6751
	M52	0.1880	0.5545	105.5599	0.1645	0.6475	19.6734
	SE	0.1882	0.5545	105.2971	0.1645	0.6476	19.2748
mean 300-100	EvoCov	0.1838	0.5656	108.8462	0.1682	0.6671	20.7017
	M32	0.1873	0.5931	109.6791	0.1708	0.6987	21.5774
	M52	0.1883	0.5959	109.1189	0.1715	0.6929	21.3227
	SE	0.1897	0.5847	110.1628	0.1718	0.6858	21.6260
mean 50/52-100	EvoCov	0.1861	0.5709	109.7424	0.1681	0.6650	20.8455
	M32	0.1909	0.5927	111.0449	0.1713	0.7010	21.2932
	M52	0.1904	0.5887	111.2285	0.1724	0.6915	21.4581
	SE	0.1912	0.5858	111.5610	0.1747	0.6940	21.8793

Table 6.5. Comparison between the EvoCov kernel learning method and the standard kernels (mean RMSE from 30 executions). The best results for each embedding type are underlined, while the best results overall are shown in bold.

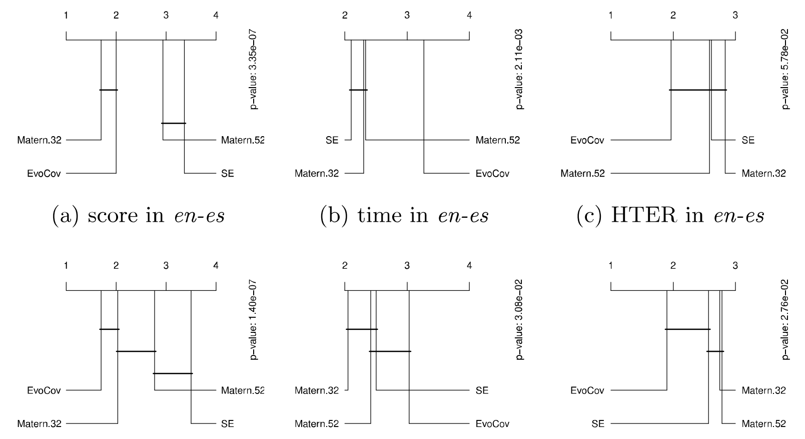
In order to assess the existence of significant differences among the standard kernels and the evolved ones, we conducted a statistical test based on the results of the LASER embeddings. For each dataset and metric, we applied Friedman's test (Friedman, 1937) and we found significant differences in 4 out of 6 comparisons ($\alpha = 0.05$. p-values can be seen in Figure 6.10). Then, for each configuration, we applied a post-hoc test based on Friedman's test, and adjusted the results with Shaffer's correction (Shaffer, 2012).

The results of the statistical tests are shown in Figure 6.10. They confirm a coherent pattern, where EvoCov is the best performing kernel for 3 of the configurations. However, according to this test, it can also be appreciated that for these configurations the differences between the kernels learned by EvoCov and Matern32 are not significant. On the other hand, the Matern32 kernel obtains significantly better results than the EvoCov kernels on both datasets for the time metric.

Overall, the kernels learned by the EvoCov algorithm have shown a solid performance in most of the sentence embedding types. These kernels outperform the standard ones in all the databases and metrics for the word-vector embeddings, although they obtain mixed results when using the LASER embeddings.

6.5.2.5 Learning multi-objective kernels

Another relevant issue is to analyze if an approach similar to the one presented can be adapted to different metrics by only optimizing their hyperparameters.



(a) score in *en-es* (b) time in *en-es* (c) HTER in *en-es*

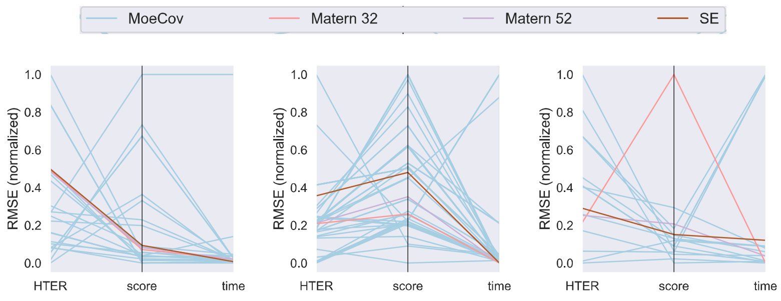
(d) score in *fr-en* (e) time in *fr-en* (f) HTER in *fr-en*

Fig. 6.10. Critical difference diagrams for the standard kernels and EvoCov with $1024 - 1024$-dimensional LASER sentence-vectors. The methods are ordered following their rankings. The metrics with no significant differences among them are matched with a straight line. At the top, the results for the *en-es news-test2010* dataset can be found, while at the bottom of the figure, the results for the *fr-en news-test2009* dataset are shown.

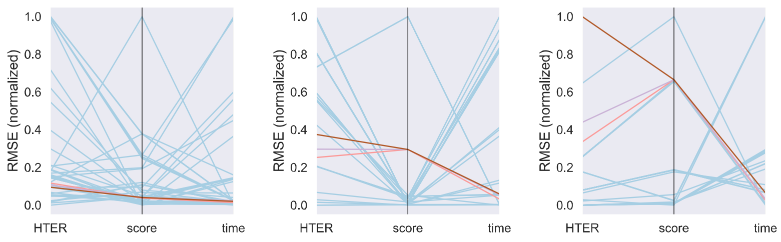
Thus, we conducted a third experiment using the MOECov algorithm and compared the results to those obtained by the standard kernels.

Being a multi-objective approach, the outcome of the MOECov algorithm is a set of non-dominated kernels per execution. For illustrative purposes, in Figure 6.11, we show a parallel coordinates diagram for one of the experiment runs, where all the kernels evolved by MOECov are compared to the standard kernels. It can be seen that most of the kernels selected by MOECov obtain good results in HTER and score metrics, although their performance is slightly worse than the standard kernels for the time metric in most types of embeddings.

In addition, a summary of the results is shown in Table 6.6. There, the final kernels (those non-dominated according to the BIC measure during the training step) provided by MOECov for each of the executions are compared to the standard kernels. This comparison is made over the test set, in terms of RMSE for each of the QE metrics. This table shows the ratio of executions in which the standard kernels are not dominated by any of the evolved kernels. We also measure the average ratio of MOECov kernels that dominate, draw with, or are dominated by the standard kernels for each type of sentence embedding. We see that for the LASER embeddings, the Matern52 is the most competitive choice as it is not dominated in 13 out of 30 executions



(a) LASER 1024-1024 in *en-es* (b) mean 50-100 in *en-es* (c) mean 300-100 in *en-es*



(d) LASER 1024-1024 in *fr-en* (e) mean 50-100 (f) mean 300-100 in *fr-en*

Fig. 6.11. Parallel coordinate plot for one of the executions comparing the results of MOECov with standard kernels. The results for the *en-es news-test2010* dataset can be found at the top, while the results for the *fr-en news-test2009* dataset are shown at the bottom.

in the *en-es news-test2010* dataset and in 10 executions in the *fr-en news-test2009* dataset. Nevertheless, in most executions there is a kernel in the non-dominated solutions proposed by MOECov that dominates each standard kernel.

All in all, the kernels learned by the MOECov algorithm have shown their ability to predict several QE metrics, dominating the performance of the standard kernels in most cases.

6.6 Conclusions

The application of GPs on NLP tasks has grown in interest in recent years. Some of the most relevant NLP research fields where GPs have been used include SA and QE of automatic translation. Moreover, recent advances in sentence embeddings have facilitated automatic feature extraction procedures that traditionally required some domain expertise. However, similar to the time series problems shown in Chapter 5, the kernel selection remains a key

	Emb. type	Standard kernels	% of executions the standard kernel is non-dominated	Avg. % of MOECov kernels that *dominate/draw with/are dominated by* the standard kernel
en-es	LASER 1024-1024	M32	30.00	13.23 / 79.04 / 7.74
		M52	43.33	13.47 / 79.78 / 6.75
		SE	36.67	11.58 / 81.91 / 6.51
	mean 300-100	M32	40.00	15.06 / 81.71 / 3.23
		M52	40.00	17.09 / 80.65 / 2.26
		SE	46.67	19.54 / 78.13 / 2.33
	mean 50/52-100	M32	36.67	20.89 / 75.44 / 3.67
		M52	43.33	14.54 / 77.07 / 8.39
		SE	33.33	18.61 / 77.31 / 4.08
fr-en	LASER 1024-1024	M32	23.33	7.39 / 77.50 / 15.11
		M52	33.33	7.99 / 76.43 / 15.58
		SE	30.00	7.52 / 78.02 / 14.47
	mean 300-100	M32	16.67	28.74 / 70.24 / 1.02
		M52	6.67	31.00 / 67.25 / 1.75
		SE	0.00	43.55 / 55.51 / 0.94
	mean 50/52-100	M32	6.67	30.95 / 68.48 / 0.57
		M52	6.67	38.60 / 60.92 / 0.48
		SE	0.00	53.89 / 45.94 / 0.17

Table 6.6. Comparison of the standard kernels and the kernels proposed by MOECov in terms of RMSE for each execution. The percentage of executions where the standard kernel is not dominated is shown. The average percentage of MOECov kernels that *dominate*, *draw with* and *are dominated by* the standard kernel is represented in the right-most column.

issue in the application of GPs to these NLP problems. Moreover, in SA and QE, it is desirable that such models are suitable for various tasks, and several metrics have been proposed to measure the quality of the GP models. Thus, we have proposed a multi-objective extension of the EvoCov algorithm, MOECov, that is able to learn kernels that are adequate according to several metrics. To the best of our knowledge, this is the first work that uses evolved GP kernels for multi-objective problems. Neither could we find previous studies that try to optimize the hyperparameters of fixed kernels simultaneously considering two or more metrics.

By addressing the creation of kernels as a multi-objective problem, we have been able to generate kernels that simultaneously optimize two of the SA metrics proposed, along with the computational complexity. Furthermore, the kernels evolved for predicting some sentiment can be also useful to predict other sentiments. We have also demonstrated that, by using MOECov, kernels that are able to predict several QE metrics can be learned. Moreover, these kernels evolved by means of EvoCov improve the performance of the standard kernels in most cases.

In terms of the sentence embeddings used, LASER embeddings have proved to be more valuable compared to the sentence embeddings computed by aggregating word-vectors. Once the bidirectional LSTM RNN has been pre-trained for several languages, the LASER embeddings are the recommended option for translation-effort prediction with GP models using the SE kernel. Among the functions used to aggregate the word-vectors, *max* function has showed a slight advantage over the extensively used *mean*. In most of the QE experiments carried out using these sentence embedding methods, including the information from the automatically translated text has helped to improve the results.

Part III

Conclusions

7

Conclusions

Kernel functions are an essential component of many machine learning algorithms. The kernel methods are a good example of this, where the kernel function plays a key role. One of the best-known kernel methods is the SVM model, where the kernel function allows non-linear classification problems to be dealt with. Beyond kernel methods, kernel functions are also needed in Bayesian inference methods, such as GPs.

In general, and particularly for SVMs and GPs, the choice of the kernel heavily influences the performance of the methods, and there is no rule of thumb for selecting the most appropriate one. Although different standard kernels have been proposed, the best results are commonly obtained by using kernels specifically designed for solving the problem at hand. Furthermore, due to the complexity of this procedure and the expert knowledge required, there is an increasing interest in developing algorithms which are able to learn kernels without human intervention.

In this regard, several methods, such as GenProg, have been proposed in the SVM literature to automatically find kernels that improve the performance of standard kernels. Apart from the selected method to represent the kernel functions and the chosen search algorithm, learning kernel functions for SVMs involves several components that need to be adjusted in order to obtain good performance, such as the hyperparameters of the kernel, the C parameter, and the selected metric to evaluate the performance of the kernel.

On the other hand, the kernel search problem has not attracted that much attention in the GP literature. While for SVMs, two types of grammars have been proposed to define the search space of kernel functions, kernel composition methods and basic mathematical expression based approaches, in GPs, most of the works have only focused on combining kernels. To the extent of our knowledge, approaches based on basic mathematical expressions, which allow a wider search space of kernels, have not been investigated in the GP field.

Next, we describe the contributions made during this dissertation, the ideas for future work, and finally the publications accepted in international conferences and journals.

7.1 Contributions

In this dissertation we have investigated the issue of learning kernels for two important Machine Learning methods, SVMs and GPs. We have analyzed the work done in SVMs in depth and identified areas for improvement, and brought to the GPs the advances made in the SVM literature, adapting these techniques to the particularities of the GPs. We have also demonstrated the usefulness of this GP approach in time series and NLP tasks, even extending the method to multi-objective problems.

We summarize the contributions made during this dissertation as follows:

- In-depth study of the components that influence the kernel search for SVMs: We have identified the different elements of SVMs, from weights and the bias of the hyperplane, to the C parameter, paying attention to the kernels and its hyperparameters, and investigated their interplay during the search. We have focused our efforts on basic mathematical expression based grammars to define the search space, and on GenProg as the search method. We emphasize the importance of having the right elements in the grammar, and question the performance of GenProg over other simpler search methods. Including the correct elements in the grammar can be more important than the search method itself when trying to find the best kernel structure for SVMs. We have also proposed some methods to set the hyperparameters and C during the kernel search. Finally, we have introduced a metric based on the BIC measure, which can overcome the challenges presented by the accuracy.
- Proposal of basic mathematical expressions as building blocks for GP kernels: We have proposed bringing the progress made in SVMs to the GPs by considering its covariance function as a program that can be learned by means of GenProg. The usage of basic mathematical expressions as building blocks provides much simpler kernels than kernel combination approaches, even improving their performance. We have also incorporated hyperparameter inheritance into GenProg, improving the efficiency of the algorithm. Some of the components designed in EvoCov could be extended to other GenProg applications beyond GPs.
- Application of the method to time series regression tasks: By means of this technique, we have improved the state-of-the-art results of time series regression tasks. We have also investigated several metrics for kernel hyperparameter optimization, providing valuable insights into their influence on the performance of the GPs in time series extrapolation.

- Application of the method to various NLP tasks: By addressing the creation of kernels as a multi-objective problem, we have been able to generate kernels that simultaneously optimize different accuracy metrics. We have also researched the best methods for feature extraction to be used in GPs, comparing several types of sentence embeddings. Thus, we have improved the performance of the most used kernels in translation effort prediction with GPs, and have shown that this method can be also useful in SA tasks.
- Development of a software for GP and SVM kernel learning: The proposed kernel learning algorithms have been coded in Python and made publicly available in the Python Package Index[1]. Moreover, the software developed to carry out the experimentation related to GPs, GPlib, has also been published[2]. These pieces of software can be useful to further develop the issues discussed in this dissertation or in other application areas.

7.2 Future work

In this section the future research directions derived from this dissertation are presented.

First, further research on grammar definition is suggested, as possible performance gains have been reported when certain elements are added to the grammar. These are the improvement areas we have identified:

- Extend the grammar to ChangePoint and ChangeWindow kernels: Lloyd et al. (2014) showed that this type of kernels are very useful in some time series extrapolation tasks. New elements could be added to the grammar to be able to create kernels of this kind.
- Create a basic mathematical expression based grammar that is able to reproduce the standard kernels, where any composition of the building-blocks is guaranteed to be PSD: The search space described by this grammar would be able to merge the benefits of kernel composition approaches and basic mathematical expression based grammars. The next step in grammar definition could be to investigate the existence of a grammar that could be used to describe a search space that contains all the PSD kernels and only these PSD kernels.

In addition, it is also interesting to investigate more sophisticated metrics to better evaluate the kernel and its hyperparameters:

- Improve the performance of the SVMBIC metric: We have seen that SVMBIC introduces some interesting properties compared to the traditional accuracy. New methods for finding kernels and hyperparameters can be designed so that SVMBIC metric can take advantage of their properties.

[1] `https://pypi.org/project/evocov/`
[2] `https://pypi.org/project/gplib/`

- Better metrics to evaluate the performance of the hyperparameters in GPs: We propose continuing the work done to measure the performance of the hyperparameter optimization metrics for GP extrapolation problems.

Finally, there are various possibilities to extend our studies on the applications of the proposed method in the NLP field:

- Extend the work done in SA to other domains: Other sentiment datasets, possibly in other languages, could be considered.
- Improve the experimentation done to validate the proposed multi-objetive GenProg approach: In the SA experiments, we noticed that, at the time of stopping the MOECov algorithm, the quality of the solutions was still improving. Therefore, more fitness evaluations are likely to produce better results of the algorithm.
- Apply the proposed methods to classification tasks within NLP: Although we focus on regression problems in this work, our contribution can be easily extended to other NLP applications, such as text classification.
- Better understand the interplay between the sentence embeddings and the GP models: We noticed that *dot_product* elements were hardly ever selected when evolving kernels for GPs based on sentence embedding data. The relation of the elements in the selected kernels with the sentence embedding type should be further studied.

7.3 Publications

In this section, the publications and submissions are presented. First, we present the publications directly derived from the thesis:

1. I. Roman, A. Mendiburu, R. Santana, and J. A. Lozano, "Sentiment analysis with genetically evolved Gaussian kernels," in Proceedings of the Genetic and Evolutionary Computation Conference, Prague, Czech Republic, Jul. 2019, pp. 1328–1337, doi: 10.1145/3321707.3321779.
2. I. Roman, R. Santana, A. Mendiburu, and J. A. Lozano, "Evolving Gaussian Process Kernels for Translation Editing Effort Estimation," in Proceedings of the 13th International Conference on Learning and Intelligent Optimization, Chania, Greece, 2020, pp. 304–318, doi: 10.1007/978-3-030-38629-0_25.

Next, the submissions that are in a revision process are shown:

1. "Evolution of Gaussian Process kernels for machine translation post-editing effort estimation," submitted to Annals of Mathematics and Artificial Intelligence.
2. "Evolving Gaussian Process kernels from elementary mathematical expressions," submitted to IEEE Transactions on Cybernetics.
3. "SVM kernel learning revisited," submitted to Neural Computing and Applications journal.

Finally, in spite of not being strictly related to this dissertation, we also list the publications produced during this thesis, as their work has contributed to some extent to the contents of this dissertation:

1. I. Roman, R. Santana, A. Mendiburu, and J. A. Lozano, "An Experimental Study in Adaptive Kernel Selection for Bayesian Optimization," IEEE Access, vol. 7, pp. 184294–184302, 2019, doi: 10.1109/ACCESS.2019.2960498.
2. I. Roman, A. Mendiburu, R. Santana, and J. A. Lozano, "Bayesian Optimization Approaches for Massively Multi-modal Problems," in Proceedings of the 13th International Conference on Learning and Intelligent Optimization, Chania, Greece, 2020, pp. 383–397, doi: 10.1007/978-3-030-38629-0_31.
3. I. Roman, J. Ceberio, A. Mendiburu, and J. A. Lozano, "Bayesian optimization for parameter tuning in evolutionary algorithms," in IEEE Congress on Evolutionary Computation, CEC 2016, Vancouver, Canada, Jul. 2016, pp. 4839–4845, doi: 10.1109/CEC.2016.7744410.
4. I. Roman, R. Santana, A. Mendiburu, and J. A. Lozano, "Kernel hautapen dinamikoa Optimizazio Bayesiarrean," in I. Ikergazte: Nazioarteko ikerketa euskaraz. Kongresuko artikulu-bilduma, Durango, 2015, p. 842.
5. I. Roman, R. Santana, A. Mendiburu, and J. A. Lozano, "Dynamic Kernel Selection Criteria for Bayesian Optimization," in BayesOpt 2014: NIPS Workshop on Bayesian Optimization, Montreal, Canada, 2014.

A

Examples of evolved kernel functions

In order to illustrate the characteristics of the kernels learned through composition approaches in contrast to kernels composed of basic mathematical expressions, we compare the ad hoc kernel designed by Rasmussen and Williams (2006) for the Mauna Loa Atmospheric CO_2 time series (shown in Equation (A.1)) and the kernel automatically composed by the approach proposed by Duvenaud et al. (2013) (Equation (A.2)), to a kernel learned by means of EvoCov (Equation (A.3)).

$$\begin{aligned} k_{RW}(\mathbf{x},\mathbf{x}') =& k_{SE}(\mathbf{x},\mathbf{x}') \times k_{PER}(\mathbf{x},\mathbf{x}') + k_{SE}(\mathbf{x},\mathbf{x}') + k_{RQ}(\mathbf{x},\mathbf{x}') + \\ & k_{SE}(\mathbf{x},\mathbf{x}') + k_{WN}(\mathbf{x},\mathbf{x}') \end{aligned} \tag{A.1}$$

$$\begin{aligned} k_{GPSS}(\mathbf{x},\mathbf{x}') =& (k_{SE}(\mathbf{x},\mathbf{x}') + k_{PER}(\mathbf{x},\mathbf{x}')) \times (k_{SE}(\mathbf{x},\mathbf{x}') + k_{RQ}(\mathbf{x},\mathbf{x}')) + \\ & k_{WN}(\mathbf{x},\mathbf{x}') \end{aligned} \tag{A.2}$$

$$\begin{aligned} k_{EvoCov}(\mathbf{x},\mathbf{x}') =& hp_{17}^2 \delta(\mathbf{x},\mathbf{x}') + 5 \times \left(e^{\left(e^{\left(3 \left\| \frac{\mathbf{x}}{hp16} - \frac{\mathbf{x}'}{hp16} \right\|^2\right)^{0.5}} \right)^{-1}} + \right. \\ & \left(hp_3 + \left\| \frac{\mathbf{x}}{hp_{16}} - \frac{\mathbf{x}'}{hp_{16}} \right\|^2 \right)^{-0.5} \times \left(5^{-1} \times hp18 + \right. \\ & \left. \left. e^{-\left\| \frac{(\left[sin(\frac{2\pi\mathbf{x}}{hp})\ cos(\frac{2\pi\times\mathbf{x}}{hp})\right]}{hp} - \frac{\left[sin(\frac{2\pi\mathbf{x}'}{hp})\ cos(\frac{2\pi\times\mathbf{x}'}{hp})\right]}{hp} \right\|^2} \right) \right) \end{aligned} \tag{A.3}$$

In Figures A.1, A.2 and A.3, these three kernels are represented as basic mathematical expression trees. It can be seen that the tree corresponding to the kernel created by EvoCov has a more compact structure.

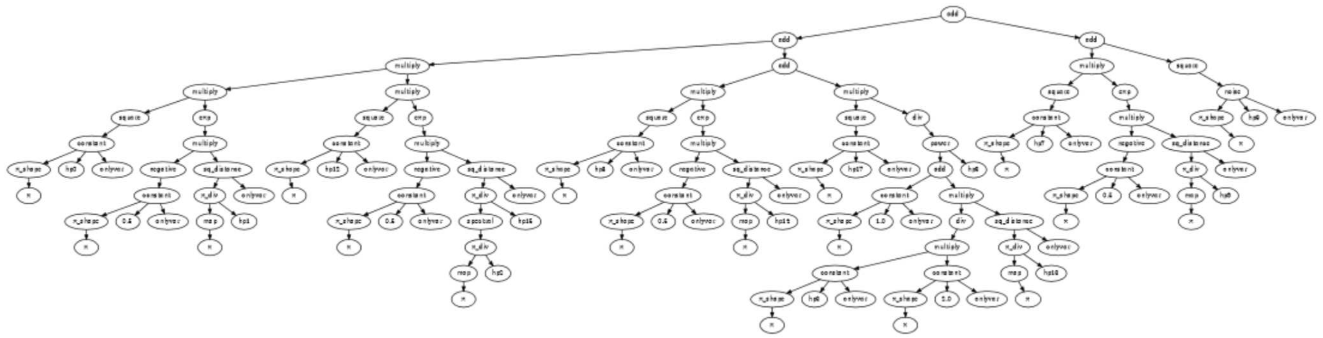


Fig. A.1. Kernel designed by Rasmussen and Williams (2006) for the Mauna Loa Atmospheric CO_2 time series, illustrated as a tree composed of basic mathematical expressions.

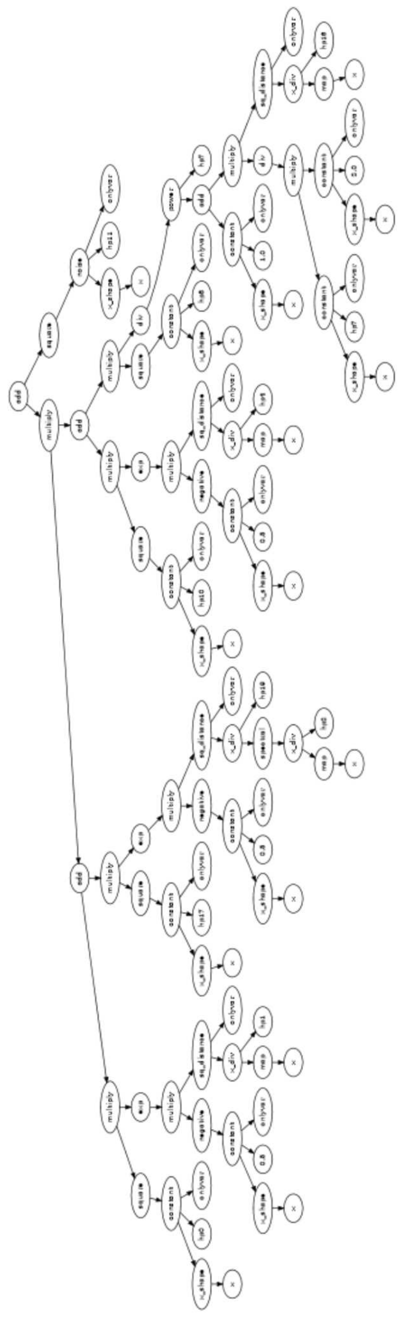


Fig. A.2. Kernel learned by means of the algorithm proposed by Duvenaud et al. (2013) for the Mauna Loa Atmospheric CO_2 time series, illustrated as a tree composed of basic mathematical expressions.

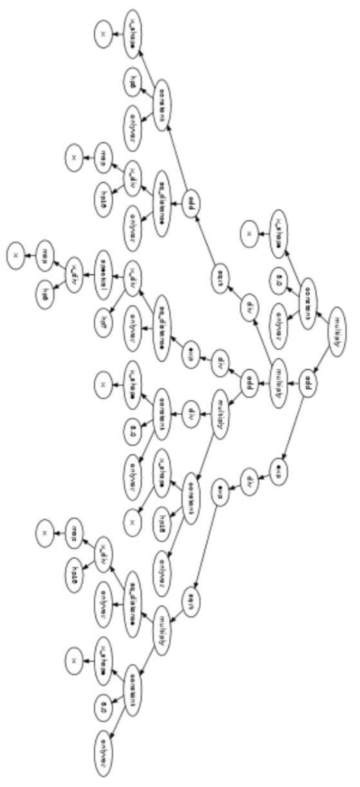
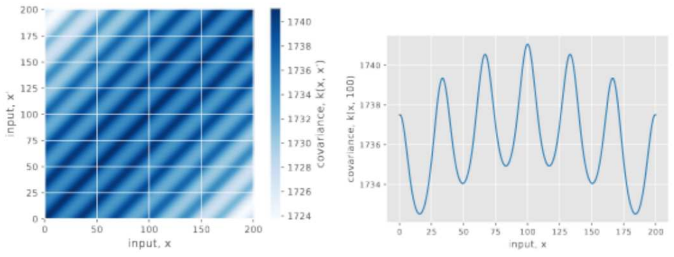


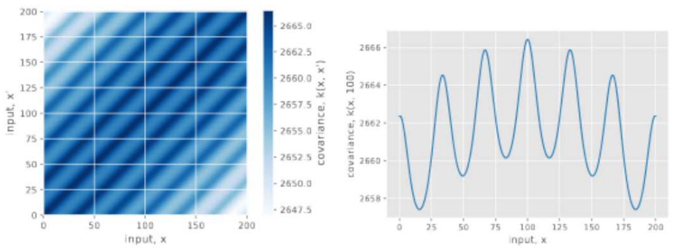
Fig. A.3. Kernel learned by means of EvoCov for the Mauna Loa Atmospheric CO_2 time series, illustrated as a tree composed of basic mathematical expressions.

While the kernel designed by Rasmussen and Williams (2006) and the one learned by means of the algorithm proposed by Duvenaud et al. (2013) are composed by 77 and 64 primitives respectively, the EvoCov kernel uses only 41 primitives.

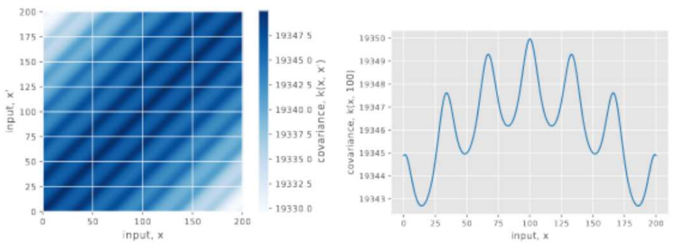
Figure A.4 shows the outcome of these kernels. In spite of having such different expressions, it can be appreciated that their outcome is similar.



(a) Rasmussen and Williams (2006)



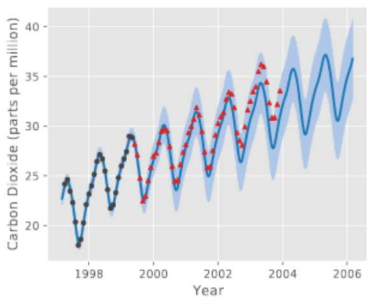
(b) Duvenaud et al. (2013)



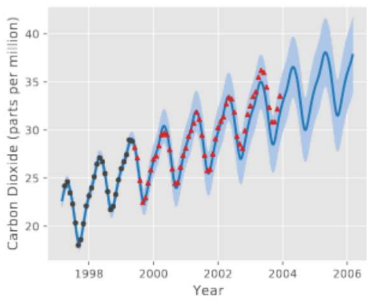
(c) EvoCov

Fig. A.4. Output of the kernels designed for the Mauna Loa Atmospheric CO_2 time series. On the left side, the actual outcome of the kernel function is shown, according to the values of the input vectors. On the right, the same function is shown, when $x' = 0$.

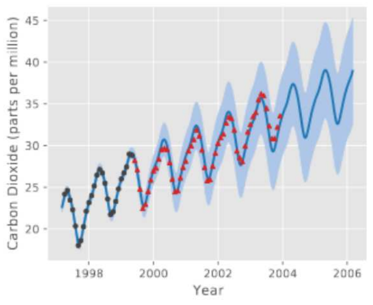
Finally, in Figure A.5, the predictions of the GP model with each of the kernels are illustrated for the Mauna Loa Atmospheric CO_2 time series. The kernel designed by Rasmussen and Williams (2006) shows the highest error in this regression task, the EvoCov kernel being the one with the best performance among these three kernels.



(a) Rasmussen and Williams (2006)



(b) Duvenaud et al. (2013)



(c) EvoCov

Fig. A.5. GP predictions of the kernels designed for the Mauna Loa Atmospheric CO_2 time series. The dots represent the last samples of the training set (the first samples of the training set are not shown), while the triangles show the samples of the test set. The predictions are illustrated with a continuous blue curve for the mean of the GP model and the light blue shadow shows 3 times the standard deviation.

References

Aldous, D. and Vazirani, U. (1994). "Go with the winners" algorithms. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 492–501.

Ali, S. and Smith-Miles, K. A. (2006). A meta-learning approach to automatic kernel selection for support vector machines. *Neurocomputing*, 70(1):173–186.

Alizadeh, M. and Ebadzadeh, M. M. (2011). Kernel evolution for support vector classification. In *2011 IEEE Workshop on Evolving and Adaptive Intelligent Systems (EAIS)*, pages 93–99.

Artetxe, M. and Schwenk, H. (2019). Massively Multilingual Sentence Embeddings for Zero-Shot Cross-Lingual Transfer and Beyond. *Transactions of the Association for Computational Linguistics*, 7:597–610. Publisher: MIT Press.

Bach, F. R., Lanckriet, G. R. G., and Jordan, M. I. (2004). Multiple Kernel Learning, Conic Duality, and the SMO Algorithm. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 6–, New York, NY, USA. ACM.

Bachoc, F. (2013). Cross Validation and Maximum Likelihood estimations of hyper-parameters of Gaussian processes with model misspecification. *Computational Statistics & Data Analysis*, 66:55–69.

Beck, D. (2017a). Modelling Representation Noise in Emotion Analysis using Gaussian Processes. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 140–145, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Beck, D. and Cohn, T. (2017). Learning Kernels over Strings using Gaussian Processes. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 67–73, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Beck, D., Cohn, T., Hardmeier, C., and Specia, L. (2015). Learning Structural Kernels for Natural Language Processing. *Transactions of the Association for Computational Linguistics*, 3:461–473.

Beck, D., Specia, L., and Cohn, T. (2016). Exploring Prediction Uncertainty in Machine Translation Quality Estimation. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 208–218.

Beck, D. E. (2017b). *Gaussian Processes for Text Regression.* phd, University of Sheffield.

Benassi, R., Bect, J., and Vazquez, E. (2011). Robust Gaussian Process-Based Global Optimization Using a Fully Bayesian Expected Improvement Criterion. In Coello, C. A. C., editor, *Learning and Intelligent Optimization*, number 6683 in Lecture Notes in Computer Science, pages 176–190. Springer Berlin Heidelberg.

Bing, W., Wen-qiong, Z., Ling, C., and Jia-hong, L. (2010). A GP-based kernel construction and optimization method for RVM. In *2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE)*, volume 4, pages 419–423.

Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".

Bochner, S. (1959). *Lectures on Fourier Integrals. (AM-42).* Princeton University Press. Google-Books-ID: O1jQCwAAQBAJ.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA. ACM. event-place: Pittsburgh, Pennsylvania, USA.

Brochu, E., Cora, V. M., and de Freitas, N. (2010). 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.

Bull, A. D. (2011). Convergence Rates of Efficient Global Optimization Algorithms. *Journal of Machine Learning Research*, 12:2879–2904.

Burges, C. J. and Crisp, D. J. (2000). Uniqueness of the SVM solution. In *Advances in Neural Information Processing Systems*, pages 223–229.

Callison-Burch, C., Koehn, P., Monz, C., and Zaidan, O. F. (2011). Findings of the 2011 Workshop on Statistical Machine Translation. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, WMT '11, pages 22–64, Stroudsburg, PA, USA. Association for Computational Linguistics. event-place: Edinburgh, Scotland.

Calvo, B. and Santafé, G. (2016). scmamp: Statistical Comparison of Multiple Algorithms in Multiple Problems. *The R Journal*, 8(1):248–256.

Chapelle, O. (2002). *Support Vector Machines: Induction Principle, Adaptive Tuning and Prior Knowledge.* PhD thesis, LIP6.

Chu, W. and Ghahramani, Z. (2005). Gaussian Processes for Ordinal Regression. *Journal of Machine Learning Research*, 6(Jul):1019–1041.

Cohn, T. and Specia, L. (2013). Modelling annotator bias with multi-task gaussian processes: An application to machine translation quality estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 32–42.

Crammer, K. and Singer, Y. (2001). On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. *Journal of Machine Learning Research*, 2(Dec):265–292.

Davis, L. (1991). Bit-climbing, representational bias, and test suit design. In *Proc. Intl. Conf. Genetic Algorithm, 1991*, pages 18–23.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II. In Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J. J., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature PPSN VI*, Lecture Notes in Computer Science, pages 849–858. Springer Berlin Heidelberg.

Demšar, J. (2006). Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7:1–30.

Deriu, J., Lucchi, A., De Luca, V., Severyn, A., Müller, S., Cieliebak, M., Hofmann, T., and Jaggi, M. (2017). Leveraging Large Amounts of Weakly Supervised Data for Multi-Language Sentiment Classification. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 1045–1052, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee. event-place: Perth, Australia.

Dioşan, L., Rogozan, A., and Pecuchet, J. P. (2007). Evolving kernel functions for SVMs by genetic programming. In *Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*, pages 19–24.

Dioşan, L., Rogozan, A., and Pecuchet, J.-P. (2008). Optimising Multiple Kernels for SVM by Genetic Programming. In *Evolutionary Computation in Combinatorial Optimization*, Lecture Notes in Computer Science, pages 230–241. Springer, Berlin, Heidelberg.

Dioşan, L., Rogozan, A., and Pecuchet, J.-P. (2012). Improving classification performance of Support Vector Machine by genetically optimising kernel shape and hyper-parameters. *Applied Intelligence*, 36(2):280–294.

Dua, D. and Graff, C. (2017). *UCI Machine Learning Repository*. University of California, Irvine, School of Information and Computer Sciences.

Durrande, N., Ginsbourger, D., and Roustant, O. (2012). Additive covariance kernels for high-dimensional Gaussian process modeling. *Annales de la Faculté de Sciences de Toulouse*, Tome 21(numéro 3):p. 481–499.

Duvenaud, D. (2014). *Automatic model construction with Gaussian processes*. Thesis, University of Cambridge.

Duvenaud, D., Lloyd, J., Grosse, R., Tenenbaum, J., and Zoubin, G. (2013). Structure Discovery in Nonparametric Regression through Compositional Kernel Search. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1166–1174.

Ekman, P. (1993). Facial expression and emotion. *American Psychologist*, 48(4):384.

Fortin, F.-A., Rainville, F.-M. D., Gardner, M.-A., Parizeau, M., and Gagné, C. (2012). DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13(Jul):2171–2175.

Fox, E. B. and Dunson, D. B. (2012). Multiresolution Gaussian Processes. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 737–745. Curran Associates, Inc.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701.

Gagné, C., Schoenauer, M., Sebag, M., and Tomassini, M. (2006). Genetic Programming for Kernel-Based Learning with Co-evolving Subsets Selection. In *Parallel Problem Solving from Nature - PPSN IX*, Lecture Notes in Computer Science, pages 1008–1017. Springer, Berlin, Heidelberg.

Garciarena, U., Santana, R., and Mendiburu, A. (2018). Evolved GANs for Generating Pareto Set Approximations. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 434–441, New York, NY, USA. ACM. event-place: Kyoto, Japan.

Garnett, R., Osborne, M. A., and Hennig, P. (2014). Active Learning of Linear Embeddings for Gaussian Processes. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, UAI'14, pages 230–239, Arlington, Virginia, United States. AUAI Press.

Garnett, R., Osborne, M. A., Reece, S., Rogers, A., and Roberts, S. J. (2010). Sequential Bayesian Prediction in the Presence of Changepoints and Faults. *The Computer Journal*, 53(9):1430–1446.

Genton, M. G. (2002). Classes of Kernels for Machine Learning: A Statistics Perspective. *Journal of Machine Learning Research*, 2:299–312.

Gijsberts, A., Metta, G., and Rothkrantz, L. (2010). Evolutionary Optimization of Least-Squares Support Vector Machines. In *Data Mining*, Annals of Information Systems, pages 277–297. Springer, Boston, MA.

Girdea, M. and Ciortuz, L. (2007). A Hybrid Genetic Programming and Boosting Technique for Learning Kernel Functions from Training Data. In *Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2007)*, pages 395–402.

Grave, E., Bojanowski, P., Gupta, P., Joulin, A., and Mikolov, T. (2018). Learning Word Vectors for 157 Languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*, pages 3483–3487, Miyazaki, Japan. European Languages Resources Association (ELRA).

HajiGhassemi, N. and Deisenroth, M. (2014). Analytic Long-Term Forecasting with Periodic Gaussian Processes. In *Proceedings of Machine Learning Research*, pages 303–311.

Hilbert, D. (1906). Grundzüge einer allgemeinen Theorie der linearen Integralgleichungen. Vierte Mitteilung. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, 1906:157–228.

Hinton, G. E. and Salakhutdinov, R. R. (2008). Using Deep Belief Nets to Learn Covariance Kernels for Gaussian Processes. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T., editors, *Advances in Neural Information Processing Systems 20*, pages 1249–1256. Curran Associates, Inc.

Howley, T. and Madden, M. G. (2005). The Genetic Kernel Support Vector Machine: Description and Evaluation. *Artificial Intelligence Review*, 24(3-4):379–395.

Howley, T. and Madden, M. G. (2006). An Evolutionary Approach to Automatic Kernel Construction. In *Artificial Neural Networks – ICANN 2006*, Lecture Notes in Computer Science, pages 417–426. Springer, Berlin, Heidelberg.

Hussain, M., Wajid, S. K., Elzaart, A., and Berbar, M. (2011). A Comparison of SVM Kernel Functions for Breast Cancer Detection. In *Imaging and Visualization 2011 Eighth International Conference Computer Graphics*, pages 145–150.

Iqbal, M., Zhang, M., and Xue, B. (2016). Improving classification on images by extracting and transferring knowledge in genetic programming. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 3582–3589.

Joachims, T. (1998). Making large-scale SVM learning practical. Technical Report 1998,28, Technical Report.

Kanagawa, M., Hennig, P., Sejdinovic, D., and Sriperumbudur, B. K. (2018). Gaussian Processes and Kernel Methods: A Review on Connections and Equivalences. *arXiv:1807.02582 [cs, stat]*. arXiv: 1807.02582.

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Skip-Thought Vectors. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 3294–3302. Curran Associates, Inc.

Klenske, E. D., Zeilinger, M. N., Schölkopf, B., and Hennig, P. (2013). Nonparametric dynamics estimation for time periodic systems. In *2013 51st Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 486–493.

Koch, P., Bischl, B., Flasch, O., Bartz-Beielstein, T., Weihs, C., and Konen, W. (2012). Tuning and evolution of support vector kernels. *Evolutionary Intelligence*, 5(3):153–170.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. (2007). Moses: Open Source Toolkit for Statistical Machine Translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings* *of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic. Association for Computational Linguistics.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press.

Kronberger, G. and Kommenda, M. (2013). Evolution of Covariance Functions for Gaussian Process Regression Using Genetic Programming. In *Computer Aided Systems Theory - EUROCAST 2013*, Lecture Notes in Computer Science, pages 308–315. Springer, Berlin, Heidelberg.

Le, Q. and Mikolov, T. (2014). Distributed Representations of Sentences and Documents. In *International Conference on Machine Learning*, pages 1188–1196. ISSN: 1938-7228 Section: Machine Learning.

Lin, K.-P. and Chen, M.-S. (2011). On the Design and Analysis of the Privacy-Preserving SVM Classifier. *IEEE Transactions on Knowledge and Data Engineering*, 23(11):1704–1717.

Lind, D., Marchal, W., and Wathen, S. (2006). *Basic Statistics for Business & Economics.* McGraw-Hill/Irwin series Business statistics. McGraw-Hill/Irwin.

Lloyd, J. R., Duvenaud, D., Grosse, R., Tenenbaum, J., and Ghahramani, Z. (2014). Automatic Construction and Natural-Language Description of Nonparametric Regression Models. In *Twenty-Eighth AAAI Conference on Artificial Intelligence.*

MacKay, D. J. C. (1996). Bayesian Methods for Backpropagation Networks. In *Models of Neural Networks III*, Physics of Neural Networks, pages 211–254. Springer, New York, NY.

Malkomes, G., Schaff, C., and Garnett, R. (2016). Bayesian optimization for automated model selection. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 2900–2908. Curran Associates, Inc.

Mercer, J. (1909). XVI. Functions of positive and negative type, and their connection the theory of integral equations. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 209(441-458):415–446.

Mezher, M. A. and Abbod, M. F. (2014). Genetic folding for solving multiclass SVM problems. *Applied Intelligence*, 41(2):464–472.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

Mohandes, M. A., Halawani, T. O., Rehman, S., and Hussain, A. A. (2004). Support vector machines for wind speed prediction. *Renewable Energy*, 29(6):939–947.

Močkus, J., Tiesis, V., and Zilinskas, A. (1978). The application of Bayesian methods for seeking the extremum. In *Towards Global Optimization*, volume 2, pages 117–129. Elsevier.

Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Lecture Notes in Statistics. Springer-Verlag, New York.

Olson, R. S., La Cava, W., Orzechowski, P., Urbanowicz, R. J., and Moore, J. H. (2017). PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(1):36.

Ortigosa-Hernández, J., Rodríguez, J. D., Alzate, L., Lucania, M., Inza, I., and Lozano, J. A. (2012). Approaching Sentiment Analysis by using semi-supervised learning of multi-dimensional classifiers. *Neurocomputing*, 92:98–115.

Pang, B. and Lee, L. (2008). Opinion Mining and Sentiment Analysis. *Foundations and Trends in Information Retrieval*, 2(1–2):1–135.

Pearson, K. (1895). Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 58:240–242.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Phienthrakul, T. and Kijsirikul, B. (2007). GPES: An algorithm for evolving hybrid kernel functions of Support Vector Machines. In *2007 IEEE Congress on Evolutionary Computation*, pages 2636–2643.

Platt, J. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in Large-Margin Classifiers*, 10(3):61–74.

Polajnar, T., Rogers, S., and Girolami, M. (2011). Protein interaction detection in sentences via Gaussian Processes: a preliminary evaluation. *International Journal of Data Mining and Bioinformatics*, 5(1):52–72.

Powell, M. J. D. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162.

Preoţiuc-Pietro, D. and Cohn, T. (2013). A temporal model of text periodicities using Gaussian Processes. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 977–988.

Quiñonero-Candela, J., Rasmussen, C. E., Sinz, F., Bousquet, O., and Schölkopf, B. (2006). Evaluating Predictive Uncertainty Challenge. In Quiñonero-Candela, J., Dagan, I., Magnini, B., and d'Alché Buc, F., editors, *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, Lecture Notes in Computer Science, pages 1–27. Springer Berlin Heidelberg.

Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian processes for machine learning*. MIT Press.

Rastrigin, L. (1963). The convergence of the random search method in the extremal control of a many parameter system. *Automaton & Remote Control*, 24:1337–1342.

Saatçi, Y., Turner, R., and Rasmussen, C. E. (2010). Gaussian Process Change Point Models. In *Proceedings of the 27th International Conference on Inter-*

national Conference on Machine Learning, ICML'10, pages 927–934. Omnipress.

Santana, R., Mendiburu, A., and Lozano, J. A. (2012). Structural transfer using EDAs: An application to multi-marker tagging SNP selection. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8.

Schmidt, M. and Lipson, H. (2013). Eureqa (version 0.98 beta)[software]. *Nutonian, Somerville, Mass, USA*.

Schuh, M. A., Angryk, R. A., and Sheppard, J. (2012). Evolving Kernel Functions with Particle Swarms and Genetic Programming. In Youngblood, G. M. and McCarthy, P. M., editors, *Proceedings of the Twenty-Fifth International Florida Artificial Intelligence Research Society Conference, 2012*, pages 80–85, Marco Island, Florida. AAAI Press.

Schwarz, G. (1978). Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461–464.

Shaffer, J. P. (2012). Modified Sequentially Rejective Multiple Test Procedures. *Journal of the American Statistical Association*.

Shah, K., Cohn, T., and Specia, L. (2013). An investigation on the effectiveness of features for translation quality estimation. In *Proceedings of the Machine Translation Summit*, volume 14, pages 167–174.

Shawe-Taylor, D. o. C. S. R. H. J., Shawe-Taylor, J., and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press. Google-Books-ID: 9i0vg12lti4C.

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Makhoul, J. (2006). A study of translation edit rate with targeted human annotation. In *In Proceedings of Association for Machine Translation in the Americas*, pages 223–231.

Sousa, A. D. M., Lorena, A. C., and Basgalupp, M. P. (2017). GEEK: Grammatical Evolution for Automatically Evolving Kernel Functions. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 941–948.

Specia, L. (2011). Exploiting objective annotations for measuring translation post-editing effort. In *Proceedings of the 15th Conference of the European Association for Machine Translation*, pages 73–80.

Specia, L., Shah, K., de Souza, J. G., and Cohn, T. (2013). QuEst - A translation quality estimation framework. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 79–84, Sofia, Bulgaria. Association for Computational Linguistics.

Strapparava, C. and Mihalcea, R. (2007). SemEval-2007 Task 14: Affective Text. In *Proceedings of the 4th International Workshop on Semantic Evaluations*, SemEval '07, pages 70–74, Stroudsburg, PA, USA. Association for Computational Linguistics.

Sullivan, K. M. and Luke, S. (2007). Evolving Kernels for Support Vector Machine Classification. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, pages 1702–1707, New York, NY, USA. ACM.

Sundararajan, S. and Keerthi, S. S. (2001). Predictive Approaches for Choosing Hyperparameters in Gaussian Processes. *Neural Computation*, 13(5):1103–1118.

Thadani, K., Ashutosh, Jayaraman, V. K., and Sundararajan, V. (2006). Evolutionary Selection of Kernels in Support Vector Machines. In *2006 International Conference on Advanced Computing and Communications*, pages 19–24.

Toal, D. J. J., Bressloff, N. W., and Keane, A. J. (2008). Kriging Hyperparameter Tuning Strategies. *American Institute of Aeronautics and Astronautics Journal*, 46(5):1240–1252.

Toal, D. J. J., Bressloff, N. W., Keane, A. J., and Holden, C. M. E. (2011). The development of a hybridized particle swarm for kriging hyperparameter tuning. *Engineering Optimization*, 43(6):675–699.

Valerio, R. and Vilalta, R. (2014). Kernel selection in support vector machines using gram-matrix properties. In *Proceedings of the 27th International Conference on Advances in Neural Information Processing Systems. Workshop on Modern Nonparametrics: Automating the Learning Pipeline, NIPS*, volume 14, pages 2–4.

Vapnik, V. (1963). Pattern recognition using generalized portrait method. *Automation and remote control*, 24:774–780.

Wang, Z. and de Freitas, N. (2014). Theoretical Analysis of Bayesian Optimisation with Unknown Gaussian Process Hyper-Parameters. *arXiv:1406.7758 [cs, stat]*. arXiv: 1406.7758.

Wilson, A. and Adams, R. (2013). Gaussian Process Kernels for Pattern Discovery and Extrapolation. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1067–1075.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. (2016). Deep Kernel Learning. In *Artificial Intelligence and Statistics*, pages 370–378.

Xu, J., Zeng, W., Lan, Y., Guo, J., and Cheng, X. (2019). Modeling the Parameter Interactions in Ranking SVM with Low-Rank Approximation. *IEEE Transactions on Knowledge and Data Engineering*, 31(6):1181–1193.

Yankovskaya, E., Tättar, A., and Fishel, M. (2019). Quality Estimation and Translation Metrics via Pre-trained Word and Sentence Embeddings. In *Proceedings of the Fourth Conference on Machine Translation (Volume 3: Shared Task Papers, Day 2)*, pages 101–105, Florence, Italy. Association for Computational Linguistics.

Zhang, F. (2011). Positive Semidefinite Matrices. In *Matrix Theory*, Universitext, pages 199–252. Springer, New York, NY.

Zhou, S. and Wang, K. (2005). Localization site prediction for membrane proteins by integrating rule and SVM classification. *IEEE Transactions on Knowledge and Data Engineering*, 17(12):1694–1705.