Kernel Adaptive Filtering Algorithms with Improved Tracking Ability

Jad Kabbara

Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

April 2014

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Engineering.

In the Name of Allah, the Most Gracious, the Most Merciful

Abstract

In recent years, there has been an increasing interest in kernel methods in areas such as machine learning and signal processing as these methods show strong performance in classification and regression problems. Interesting "kernelized" extensions of many well-known algorithms in artificial intelligence and signal processing have been presented, particularly, kernel versions of the popular online recursive least squares (RLS) adaptive algorithm, namely kernel RLS (KRLS). These algorithms have been receiving significant attention over the past decade in statistical estimation problems, among which those problems involving tracking time-varying systems. KRLS algorithms obtain a non-linear least squares (LS) regressor as a linear combination of kernel functions evaluated at the elements of a carefully chosen subset, called a dictionary, of the received input vectors. As such, the number of coefficients in that linear combination, i.e., the weights, is equal to the size of the dictionary. This coupling between the number of weights and the dictionary size introduces a trade-off. On one hand, a large dictionary would accurately capture the dynamics of the input-output relationship over time. On the other, it has a detrimental effect on the algorithms ability to track changes in that relationship because having to adjust a large number of weights can significantly slow down adaptation. In this thesis, we present a new KRLS algorithm designed specifically for the tracking of time-varying systems. The key idea behind the proposed algorithm is to break the dependency of the number of weights on the dictionary size. In the proposed method, the number of weights K is fixed and is independent from the dictionary size.

Particularly, we use a novel hybrid approach for the construction of the dictionary that employs the so-called surprise criterion for admitting data samples along with a simple pruning method ("remove-the-oldest") that imposes a hard limit on the dictionary size. Then, we propose to construct a K-sparse LS regressor tracking the relationship of the most recent training input-output pairs using the K dictionary elements that provide the best approximation of the output values. Identifying those dictionary elements is a combinatorial optimization problem with a prohibitive computational complexity. To overcome this, we extend the Subspace Pursuit algorithm (SP) which, in essence, is a low complexity method to obtain LS solutions with a pre-specified sparsity level, to non-linear regression problems and introduce a kernel version of SP, which we call Kernel SP (KSP). The standard KRLS is used to recursively update the weights until a new dictionary element selection is triggered

by the admission of a new input vector to the dictionary. Simulations show that that the proposed algorithm outperforms existing KRLS-type algorithms in tracking time-varying systems and highly chaotic time series.

Sommaire

Au cours des dernières années, il y a eu un intérêt accru pour les méthodes à noyau dans des domaines tels que l'apprentissage automatique et le traitement du signal, puisque ces méthodes démontrent une performance supérieure dans la résolution des problèmes de classification et de régression. D'intéressantes extensions à noyau de plusieurs algorithmes connus en intelligence artificielle et en traitement du signal ont été introduites, particulièrement, les versions à noyau du fameux algorithme d'apprentissage incrémental des moindres carrés récursifs (en anglais, Recursive Least Squares (RLS)), nommées KRLS. Ces algorithmes ont reçu une attention considérable durant la dernière décennie dans les problèmes d'estimation statistique, particulièrement ceux de suivi des systèmes variant dans le temps. Les algorithmes KRLS forment le régresseur aux moindres carrés non-linéaires en utilisant une combinaison linéaire de noyaux évalués aux membres d'un sous-ensemble, appelé dictionnaire, des données d'entrée. Le nombre des coefficients dans la combinaison linéaire, c'est à dire les poids, est égal à la taille du dictionnaire. Ce couplage entre le nombre de poids et la taille du dictionnaire introduit un compromis. D'une part, un dictionnaire de grande taille reflète avec précision la dynamique de la relation entre les données d'entrée et les sorties à travers le temps. De l'autre part, un tel dictionnaire diminue la capacité de l'algorithme à suivre les variations dans cette relation, car ajuster un grand nombre de poids ralentit considérablement l'adaptation de l'algorithme aux variations du système. Dans cette thèse, nous présentons un nouvel algorithme KRLS conçu précisément pour suivre les systèmes variant dans le temps. L'idée principale de l'algorithme est d'enlever la dépendance du nombre de poids sur la taille du dictionnaire. Ainsi, nous proposons de fixer le nombre de poids indépendamment de la taille du dictionnaire.

Particulièrement, nous présentons une nouvelle approche hybride pour la construction du dictionnaire qui emploie le test de la surprise pour l'admission des données d'entrées avec une méthode simple d'élagage (l'élimination du membre le plus ancien du dictionnaire) qui impose une limite stricte sur la taille du dictionnaire. Nous proposons ainsi de construire un régresseur K-creux (en anglais, K-sparse) aux moindres carrés qui suit la relation des paires de données d'entrées et sorties les plus récentes en utilisant les K membres du dictionnaire qui approximent le mieux possible les sorties. L'identification de ces membres est un problème d'optimisation combinatoire ayant une complexité prohibitive. Pour surmonter cet obstacle, nous étendons l'algorithme Subspace Pursuit (SP), qui est une méthode à

complexité réduite pour le calcul des solutions aux moindres carrés ayant un niveau préfixé de parcimonie, aux problèmes de régression non-linéaire. Ainsi, nous introduisons une version à noyau de SP qu'on appelle Kernel Subspace Pursuit (KSP). L'algorithme standard KRLS est utilisé pour l'ajustement récursif des poids jusqu'à ce qu'un nouveau vecteur de donnée soit admis au dictionnaire. Les simulations démontrent que la performance de notre algorithme dans le cadre du suivi des systèmes variant dans le temps surpasse celle d'autres algorithmes KRLS.

Acknowledgments

First and foremost, all praise is due to God. It is only through His immense bounty and constant guidance that I was able to go on through my journey at McGill. I thank Him for blessing me with the opportunity to attend McGill and for giving me the strength and drive to finish my graduate studies.

I have also been blessed and very fortunate to have had the opportunity to work under the supervision of Professor Ioannis Psaromiligkos. In my two years at McGill, I got to know Prof. Psaromiligkos as an instructor, advisor and older brother. His dedication and hard work are simply inspiring. His advice and mentoring helped shaping me not only as a researcher, but as a well-rounded individual. His cheerful spirit and amiable character were always a source of comfort and optimism. I can go on talking about Prof. Psaromiligkos, but I am certain that words would fall short of expressing how grateful I am to have been able to work with him. He is simply an inspiring advisor and a great individual. Learning from him and with him was truly an enjoyable experience that I shall forever remember.

I would like to thank Professor Mark Coates for reviewing my thesis and for his insightful feedback and valuable comments.

I would like to thank my lab mates Ahmad, Ardavan, Dinos, François, Ivan, Mahmoud, Stefanos and Saeed for their wonderful company. I am particularly grateful for Saeed's immense support throughout all the stages of my graduate studies. I am also grateful to Ardavan and François for all the instrumental discussions we had and for sharing with me their insight into the intricacies of machine learning. I want to also thank Ahmad for all the good times we spent in the lab, for all the valuable conversations we shared and for being a source of support and motivation.

I am truly blessed to have been able to share my journey at McGill with my best friend Fadel. Despite the distance, Fadel was always there to provide me with all the support I needed and to motivate me to always outdo myself. His unwavering support, constant motivation and dedication in his own work were a source of continued inspiration. His friendship has always been a source of happiness, and it is one that I will always cherish dearly.

I was also fortunate to share those two past years with amazing friends that really made Montreal embody the meaning of home away from home. I would like to first thank Mohamad and Moataz for making my stay in Montreal enjoyable and unique, for supporting

me at all times, for sharing with me those happy moments and for being there for me in those other frustrating moments. I would also like to thank Ahmed Youssef for his friendship and particular support in the last two months leading up to the culmination of this thesis. I am also grateful to Fawzi and Seif for being great friends; I thank Fawzi for sharing with me many memorable moments that I will always remember, and I thank Seif for sparking conversations and debates that range from engineering to psychology and that feed the mind, soul and heart. I also thank Amir, Hani, Mohamed Abdelghany, Mohamed Khairy, Muhammad Elhusseini and Nazem for their friendships. I am thankful to Alaa for truly being an older brother to me. I am grateful to Ahmed Masmoudi, Chris, Dung Ho, Gowdemy, Jules, Ishaan, Rawan and Yue for all the good times we spent in my first year. I would also like to thank Mahvish for being a great friend, for her light spirit and warm company, and for her continuous support.

Finally, I cannot genuinely express my gratitude towards my parents Ezzat and Ghiwa, and my brothers Nouhad and Abdallah. Without my parents, I wouldn't be where I am today. Their endless sacrifices and tremendous encouragement have always been a source of inspiration, and making them proud was and will always be an incentive for my hard work. I will always be indebted to them, and it is to them that my heartfelt appreciation and unconditional love go.

Contents

List of Figures

List of Tables

List of Acronyms

ALD	Approximate Linear Dependence
ANN	Artificial Neural Networks
BP	Basis Pursuit
FB	Fixed Budget
KBP	Kernel Basis Pursuit
KLMS	Kernel Least Mean Square
KMP	Kernel Matching Pursuit
KRLS	Kernel Recursive Least Squares
KSP	Kernel Subspace Pursuit
LARS	Least Angle Regression
LASSO	Least Absolute Shrinkage and Selection Operator
LMS	Least Mean Square
LS	Least Squares
LS-SVM	Least Squares-Support Vector Machine
MAP	Maximum A Posteriori
OBD	Optimal Brain Damage
OBS	Optimal Brain Surgeon
PCA	Principal Component Analysis
RKHS	Reproducing Kernel Hilbert Space
RLS	Recursive Least Squares
SC	Surprise Criterion
SP	Subspace Pursuit
SVM	Support Vector Machine
SW	Sliding Window

Chapter 1

Introduction

Over the past few decades, the area of machine learning has attracted widespread interest from researchers: On one hand, the area showed a promising prospect for a vital role in many areas of science (e.g., biology and medicine), finance, and industry as many recurring problems in those areas involve making predictions about new data given past data. Examples include diagnosing prostate cancer volume and stage from various measurements, studying patterns in DNA datasets in an attempt to link them to certain disorders, forecasting stock market movements and identifying people and recognizing faces. On the other hand, it lies at the intersection of the fields of engineering, computer science and statistics - thus drawing the interest of researchers from all those fields. This multitude and diversity of the problems in context made machine learning appealing to researchers from different areas and backgrounds.

1.1 From Linear Methods to Kernel Methods

Many problems in machine learning can be expressed as a simple classification or regression problem. In classification, data is typically represented by points in a given space, and classes are ideally different regions of that space grouping data points that have similar characteristics. The simplest form of classes is that of linearly separable classes, i.e., those that can be separated by hyperplanes (or, lines in the case of 2-dimensional spaces). Regression is more general in the sense that the desired output consists of one or more continuous variables (as opposed to discrete labels in the case of classification). The simplest of regression models are linear models that can be represented in the form of linear functions of the

input vectors. Due to their inherent simplicity, such problems have been studied first, and as a result, linear classifiers and linear regression models were the mainstay of statistical learning for much of that period. However, in most real problems, data patterns are more complex. Classes may not be easily separated by hyperplanes and data cannot be always fitted by simple linear regression models. Hence, more sophisticated models were needed to learn complex data. These include non-linear models which are models that involve a non-linear transformation of data into a high-dimensional space where the transformed data is more likely to be separable. As such, non-linear learning methods became increasingly popular and were studied extensively in the past two decades. Examples include Artificial Neural Networks (ANNs) [1], Decision Trees [1] and Support Vector Machines (SVMs) [2].

In recent years, kernel methods [3] have received major attention as they presented non-linear versions of conventional linear supervised and unsupervised learning algorithms, yielding impressive classification and regression performance. Simply put, a kernel function, applied to pairs of input vectors, can be interpreted as an inner product between the pair of vectors mapped to a high dimensional Hilbert space, the map being the non-linear transformation mentioned earlier. In the computer science, signal processing and machine learning literature, the substitution of inner products by kernels has been referred to as the kernel trick, and has led to interesting kernelized extensions of many well-known algorithms. These include kernel SVMs [3], kernel Principle Component Analysis (PCA) [4] and kernel Fisher discriminant analysis [5]. In the field of adaptive filtering, several algorithms were kernelized, e.g., the Recursive Least-Square (RLS) algorithm [6], the Least-Mean Square (LMS) algorithm [7] and the Affine Projection algorithm [8].

1.2 KRLS, Sparsification and Pruning

The RLS algorithm is a well-known and widely-used algorithm in adaptive signal processing and communications. This extension of the classical least-square (LS) approach addresses the issue that, in many applications, data arrives sequentially to the system, and so, the solution of the LS problem needs to be recomputed with each new data sample. RLS, an online algorithm, is presented as an answer to that need. It recursively finds linear LS predictors by updating the LS solution at each instance when a new data sample arrives.

By virtue of the kernel trick, Engel et al. present in [9] an online kernel version of the RLS algorithm, namely KRLS, which is able to efficiently and recursively solve non-

linear LS prediction problems. The algorithm solves linear regression problems in a high-dimensional feature space (induced by the kernel) where regression models of the transformed data are expected to be more accurate. As is typical with kernel-based regression methods, the number of parameters that need to be calculated to obtain the LS solution is equal to the number of input vectors, and this number grows without bound as the iterations progress, which burdens the algorithm with an increasing complexity and a need for an increasing amount of computational resources. To address this issue, a selection procedure, referred to as "sparsification", is commonly used in conjunction with KRLS to form the LS regressor using a carefully chosen subset, termed dictionary, of the input vectors. In the past two decades, sparsification has been well studied in the fields of machine learning (specifically LS-SVMs [10] and neural networks [1]) and image processing, and has been increasingly popular in kernel adaptive filtering algorithms. Examples of sparsification techniques used with KRLS include the Approximate Linear Dependence technique (ALD) [9] and the Surprise Criterion (SC) [11]: In ALD, an incoming input is added to the dictionary only if its feature representation can't be written as an approximate linear combination of the feature representations of the input vectors already admitted to the dictionary. In SC, Liu et al. approach the sparsification issue from an information theoretic perspective by adopting as the admission criterion the "surprise" which measures how relevant the information that a new input-output pair can contribute to the already accumulated "knowledge" of the learning system is.

Despite the improvement in computational consumption and memory usage, sparsification techniques often fall short of satisfying the need to impose a limit on the number of samples that can be stored in the dictionary. This is indeed crucial for practical considerations when algorithms need to be implemented, for example, on DSP chips with finite memory and limited computational resources. The family of Fixed-Budget (FB) algorithms addresses this requirement: when the number of dictionary samples reaches the predefined limit, one sample is removed from the dictionary to accommodate the newest incoming sample. This process, known as pruning, has been well studied in the context of neural networks [1] and has been recently receiving attention in the field of kernel adaptive filtering algorithms (e.g., [12] and [13]).

1.3 Tracking Time-Varying Systems

One of the key advantages of sparsification and pruning is that they enable tracking of non-stationary systems. In many applications, such as mobile communications and acoustic echo cancellation, the statistics of signals change across time, thus, it is important to design algorithms that possess the ability to effectively track time-varying systems.

In standard KRLS problems, the LS solution is given in the form of a weighted linear combination of kernels evaluated at the input vectors. As a result, in all KRLS algorithms to this date, the vector consisting of those weights is of equal size to that of the dictionary. This coupling of the parameter vector length to the dictionary size introduces an interesting trade-off. While a large dictionary is favorable as it would represent all the dynamics of the input-output relationship over time, it has a detrimental effect on the algorithm's ability to track changes in that relationship; it is well known that having to adjust a large number of weights significantly slows down convergence and adaptation as the algorithm would need an increasing time to react to the changes in the inputs/outputs. This trade-off highlights the need to decouple the size of the weight vector from the dictionary size and motivates the work presented in this thesis.

In this thesis, we introduce a new KRLS algorithm designed for the specific purpose of tracking time-varying systems. Data samples are admitted to the dictionary only after passing the SC test, however, the maximum size K of the weight vector is fixed and independent from the dictionary size. As such, we gain the best of both worlds: on one hand, we are allowed to have a large dictionary that can accurately represent the dynamics of the input-output relationship; on the other, the weight vector retains a limited size which leads to an improved tracking ability. Our algorithm, when the dictionary size is less than K, is identical to SC-KRLS [11]. However, when the dictionary size exceeds K, we choose, from the dictionary elements, the K elements that will track best the N most recent received data samples. To this end, we extend the Subspace Pursuit (SP) algorithm [14] to non-linear regression problems and introduce the Kernel Subspace Pursuit (KSP) used as a building block of the proposed KRLS algorithm termed SP-KRLS.

KSP is, in essence, a method that learns a regression function by means of sparse approximation using a finite subset of elements selected from a kernel-based dictionary. KSP is an iterative method whereby the subset of selected dictionary elements is initialized in the first iteration of the method and then refined throughout a limited number of iterations

to minimize the approximation error. The selected dictionary subset from the last KSP iteration specifies the dictionary elements that will be used in the KRLS computations (until a new subset has been selected in future iterations of SP-KRLS).

1.4 Contributions

The contributions of the work presented in this thesis are three-fold:

- First, we present an online algorithm that is capable of tracking efficiently time-varying systems, by decoupling the dictionary size and weight vector size, the equality between which was encountered in all previous KRLS algorithms.
- Second, to the best of our knowledge, our work is the first to present a kernel version of the Subspace Pursuit algorithm and to use it in the context of tracking of time-varying systems.
- Third, we present a KRLS algorithm that employs a sparsification criterion for admitting data samples to the dictionary along with imposing a hard limit on the size of the dictionary. Previous KRLS algorithms only considered one of sparsification or pruning in their algorithm design.

1.5 Thesis Organization

This thesis is organized as follows: In Chapter 2, a primer on kernel methods is presented: Kernels are defined and their most important properties are presented, as well as relevant theory on reproducing kernel Hilbert spaces and the Representer theorem. In Chapter 3, we review the related work on sparsification techniques and pruning strategies. In Chapter 4, we motivate the need for decoupling the size of the weight vector from the dictionary size, then introduce the Kernel Subspace Pursuit algorithm that will be used as a building block of the KRLS algorithm proposed subsequently in the chapter. The results of simulation studies are presented in Chapter 5. Finally, Chapter 6 concludes the thesis and outlines promising directions for future research.

Chapter 2

A Primer on Kernel Methods

In this chapter, we present a review of kernels which are at the heart of one of the most popular classes of non-linear methods used in machine learning, namely the class of kernel methods.

We begin by reviewing some of the relevant mathematical structures and notations. Then we define kernels and present their most important properties. Next, we introduce reproducing kernel Hilbert spaces. Afterwards, we define kernel methods before presenting finally one of the main theorems for kernel methods, the representer theorem.

2.1 Mathematical Prerequisites

In this section, we review basic mathematical definitions that will be needed later in this thesis. We start by defining the inner product space, the general form of a more special space, the Hilbert space, which, as we will see later in this chapter, is at the core of the research presented in this thesis.

Definition 1. (Inner Product Space [15]) *An inner product space is a vector space that admits a structure called inner product. Let $\mathcal{V}$ be a vector space and $\mathbb{C}$ the space of complex numbers. An inner product is a mapping $\langle \cdot, \cdot \rangle : \mathcal{V}^2 \mapsto \mathbb{C}$ that satisfies the following conditions:*

1. Conjugate Symmetry*:*

$$\langle x, x' \rangle = \langle x', x \rangle^* \ \ \forall x, x' \in \mathcal{V} \tag{2.1}$$

where $(\cdot)^*$ *denotes the complex conjugation operation. Note that if the inner product maps to the real space* $\mathbb{R}$*, this property becomes symmetry.*

2. Linearity*:*

$$\langle \alpha x + \beta x', x'' \rangle = \alpha \langle x, x'' \rangle + \beta \langle x', x'' \rangle \quad \forall x, x', x'' \in \mathcal{V}, \forall \alpha, \beta \in \mathbb{C} \tag{2.2}$$

i.e., the inner product is linear in its first argument. Note that if the inner product maps to $\mathbb{R}$*, this property becomes bilinearity. An inner product is said to be bilinear if it is linear in each argument, that is,*

$$\langle x, \alpha x' + \beta x'' \rangle = \alpha \langle x, x' \rangle + \beta \langle x, x'' \rangle \quad \forall x, x', x'' \in \mathcal{V}, \forall \alpha, \beta \in \mathbb{R}. \tag{2.3}$$

3. Positive definiteness*:*

$$\langle x, x \rangle \geq 0, \quad \forall x \in \mathcal{V} \tag{2.4}$$

i.e., the inner product of an element with itself is positive with the equality holding only when $x = 0$*.*

Next, we introduce the metric space.

Definition 2. (Metric Space [16]) *A metric space* $\mathcal{M}$ *is a set such that any two elements* x *and* x' *of* $\mathcal{M}$ *can be associated with a real number* $d(x, x')$*, called the distance from* x *to* x'*, that satisfies the following properties:*

$$\begin{aligned} d(x, x') &\geq 0 \ , \forall x \neq x' && (2.5) \\ d(x, x) &= 0 && (2.6) \\ d(x, x') &= d(x', x) && (2.7) \\ d(x, x') &\leq d(x, x'') + d(x'', x), \quad \forall x'' \in \mathcal{M} && (2.8) \end{aligned}$$

Any function satisfying the aforementioned properties is called a distance function, or simply a metric. One example of a metric space is the two-dimensional Euclidean space $\mathbb{R}^2$ where the distance function between elements of $\mathbb{R}^2$ is:

$$d(x, x') = ||x - x'||, \quad \forall x, x' \in \mathbb{R}^2 \tag{2.9}$$

where $||\cdot||$ is the Euclidean distance, also called Euclidean norm, defined for a vector $x = [x_1, x_2] \in \mathbb{R}^2$ as:

$$||x|| = \sqrt{x_1^2 + x_2^2}. \tag{2.10}$$

In addition to the aforementioned definitions, we need to define the Cauchy sequence and notion of completeness to be able to define the Hilbert space.

Definition 3. (Cauchy Sequence [16]) *A sequence p_n (with $n = 1, \ldots$) in a metric space $\mathcal{M}$ is said to be a Cauchy sequence if, as the sequence progresses, its elements become arbitrarily close to each other (in a way that the sequence converges eventually to a limit), that is, if for every $\varepsilon > 0$, there is an integer N such that $d(p_n, p_m) < \varepsilon$ if $n, m \geq N$.*

For a better understanding of the notion of a Cauchy sequence, consider a sequence of real numbers $x_1, x_2, \ldots$. The absolute value $|x_n - x_m|$, $\forall n, m \in \{1, \ldots, \}$, reflects how far from each other the two real numbers x_n and x_m are so it can be thought of as a distance metric.[1] For this sequence to be Cauchy, there must exist a positive integer N for every positive real number ε such that for all natural numbers $n, m > N$, $|x_n - x_m| < \varepsilon$ $(\forall n, m \in \{1, \ldots\})$.

The definition of a Cauchy sequence allows us to define the notion of complete space as follows.

Definition 4. (Complete Space [16]) *A metric space $\mathcal{M}$ in which every Cauchy sequence converges to a point in $\mathcal{M}$ is called a complete space.*

Having defined all the necessary structures and notions, we finally define the Hilbert space.

Consider an inner product space $\mathcal{H}$ as defined in Definition 1. The norm induced by the inner product of $\mathcal{H}$ is:

$$\|x\| = \sqrt{\langle x, x \rangle} \tag{2.11}$$

The norm allows then the definition of a distance metric d such that, if x and y are two elements in $\mathcal{H}$, their distance is given by:

$$d(x, y) := ||x - y||. \tag{2.12}$$

[1] The absolute value $|x_n - x_m|$ does indeed satisfy all 4 properties for a distance metric that were introduced in Definition 2.

Definition 5. (Hilbert Space [15]) *A Hilbert space $\mathcal{H}$ is an inner product space that is complete with respect to the norm induced by the inner product of $\mathcal{H}$.*

To better understand the meaning of completeness with respect to the norm in this context, first note that an inner product space as defined in Definition 1 (without any further assumptions about completeness) is called a pre-Hilbert space. To turn it into a Hilbert space, one *completes* it in the norm (2.11) corresponding to its inner product. This is done by adding to the space all limit points of those Cauchy sequences that are convergent in that norm but outside the space. Accordingly, by including in that pre-Hilbert space $\mathcal{H}$ the aforementioned limit points, all Cauchy sequences in $\mathcal{H}$ will converge inside $\mathcal{H}$, thus turning it into a complete space.

The following are two examples of Hilbert spaces:

1. The vector space $\mathbb{R}^n$ with $\langle x, y\rangle = x^T y$, the vector dot product of x and y.
2. The space ℓ_2 of square summable sequences[2] with inner product $\langle x, y\rangle = \sum_{i=1}^{\infty} x_k y_k$ where $x = [x_k]_{k\in\mathbb{N}}$ and $y = [y_k]_{k\in\mathbb{N}}$.

2.2 Kernels and Their Properties

In this section, we define kernels and present their most important properties. We focus on the class of positive definite kernels as this class allows the use of the kernel trick.

2.2.1 Definition

Consider a training set

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n) \in \mathcal{X} \times \mathcal{Y} \tag{2.13}$$

with $\mathcal{X}$ a non-empty domain containing the inputs $\mathbf{x}_i$, called the input domain, and $\mathcal{Y}$ the target domain, i.e., the domain containing the targets y_i.

The objective of a learning algorithm is to be able to predict what a target y_{n+1} is for a new input $\mathbf{x}_{n+1}$ given what this algorithm has learned from the training set. This objective

[2] A square summable sequence is a sequence such that the series of the squares of its terms converges to a finite sum.

can also be expressed as being able to choose y_{n+1} in a way that the pair $(\mathbf{x}_{n+1}, y_{n+1})$ is in some sense *similar* to the training examples. In 1964, Aizman et al. introduced the concept of a *kernel* to the field of pattern recognition in [17] as a tool to solve binary classification problems. In such problems, the objective is to assign to a new input $\mathbf{x}_{n+1}$ a label $y_{n+1} \in \{\pm 1\}$. For this purpose, similarity measures in each of $\mathcal{X}$ and $\mathcal{Y} = \{\pm 1\}$ are required. In $\mathcal{Y}$, this is rather simple as two target values can be either equal or non-equal. However, the former requires a function, say k, that, given two inputs $\mathbf{x}$ and $\mathbf{x}'$, returns a real number characterizing their similarity [3], that is,

$$k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}, \qquad (\mathbf{x}, \mathbf{x}') \mapsto k(\mathbf{x}, \mathbf{x}'). \tag{2.14}$$

This function k is referred to as kernel. One particular class of kernels is the class of positive definite kernels which is introduced in the next section.

2.2.2 Positive Definite Kernels

Before defining positive definite kernels, we first need to define the Gram matrix of a kernel and recall the definition of positive definite matrices.

Definition 6. (Gram Matrix) *Consider a kernel k and inputs $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n \in \mathcal{X}$. The $n \times n$ matrix K with entries*

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) \tag{2.15}$$

is called the Gram matrix (also referred to as the kernel matrix) of k with respect to $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$.

Definition 7. (Positive Definite Matrix) *A real $n \times n$ symmetric matrix $\mathbf{K}$ with entries K_{ij} satisfying*

$$\sum_{i,j} c_i c_j K_{ij} \geq 0 \tag{2.16}$$

for all $c_i \in \mathbb{R}$ is called positive definite. The matrix is called strictly positive definite if the equality in (2.16) only holds for $c_1 = ... = c_n = 0$.

Definition 8. (Positive Definite Kernel) *Consider a non-empty set $\mathcal{X}$ and a kernel $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$. The kernel k is called a positive definite kernel if, for all $n \in \mathbb{N}$ and $\mathbf{x}_i \in \mathcal{X}, i = 1, \ldots, n$, k leads to the creation of a positive definite Gram matrix. Moreover,*

if for all $n \in \mathbb{N}$ and distinct $\mathbf{x}_i \in \mathcal{X}, i = 1, \ldots, n$, k leads to the creation of a strictly positive definite Gram matrix, then k is called a strictly positive definite kernel. Following the definition of a positive definite matrix, a positive definite kernel is symmetric.

The class of positive definite kernels has particularly gained a lot of attention in the area of machine learning because kernels belonging to this class can be written in the following form [18]:

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle \quad \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X} \tag{2.17}$$

where $\mathcal{X}$ is the input space (as defined in Section 2.2.1) and $\phi : \mathcal{X} \mapsto \mathcal{H}$ maps input vectors from $\mathcal{X}$ into a Hilbert space $\mathcal{H}$, sometimes referred to as the feature space.[3] Since $\phi(\cdot)$ maps the input space to the feature space, it is called the feature map. As we will see in the next section, (2.17) constitutes the foundation of kernel-based algorithms, hence the aforementioned interest in positive definite kernels.

It is important to note that not all kernels are positive definite; these kernels which are not, cannot be written in the form (2.17). An example is the Sigmoid kernel [3] defined as:

$$k(\mathbf{x}, \mathbf{x}') = \tanh(\alpha \langle \mathbf{x}, \mathbf{x}' \rangle + \beta), \quad \alpha > 0, \beta < 0. \tag{2.18}$$

The Sigmoid kernel is not positive definite for certain values of its parameters, but is quite popular nonetheless. Since positive definite kernels are the kernels of interest in the work presented in this thesis, they will, henceforth, simply be referred to as kernels.

2.2.3 Properties of Kernels

The following are some of the properties of kernels [19]:

1. A linear combination of kernels is also a kernel, i.e., given two kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following is a kernel:

$$k(\mathbf{x}, \mathbf{x}') = \alpha k_1(\mathbf{x}, \mathbf{x}') + \beta k_2(\mathbf{x}, \mathbf{x}'), \quad \alpha, \beta \geq 0. \tag{2.19}$$

Proof Both $k_1(\cdot, \cdot)$ and $k_2(\cdot, \cdot)$ are kernels, so using (2.17), we can write them as $k_1(\mathbf{x}, \mathbf{x}') = \langle \phi_1(\mathbf{x}), \phi_1(\mathbf{x}') \rangle$ and $k_2(\mathbf{x}, \mathbf{x}') = \langle \phi_2(\mathbf{x}), \phi_2(\mathbf{x}') \rangle$. By multiplying $k_1(\cdot, \cdot)$ and

[3] We will specify the map $\phi(\cdot)$ later in Section 2.3.1.

$k_2(\cdot,\cdot)$ by α and β respectively, we get:

$$\alpha k_1(\mathbf{x},\mathbf{x}') = \langle\sqrt{\alpha}\phi_1(\mathbf{x}), \sqrt{\alpha}\phi_1(\mathbf{x}')\rangle \text{ and } \beta k_2(\mathbf{x},\mathbf{x}') = \langle\sqrt{\beta}\phi_2(\mathbf{x}), \sqrt{\beta}\phi_2(\mathbf{x}')\rangle.$$

Thus, we get the following:

$$\begin{aligned} k(\mathbf{x},\mathbf{x}') &= \alpha k_1(\mathbf{x},\mathbf{x}') + \beta k_2(\mathbf{x},\mathbf{x}') \\ &= \langle\sqrt{\alpha}\phi_1(\mathbf{x}), \sqrt{\alpha}\phi_1(\mathbf{x}')\rangle + \langle\sqrt{\beta}\phi_2(\mathbf{x}), \sqrt{\beta}\phi_2(\mathbf{x}')\rangle \\ &= \left\langle \left[\sqrt{\alpha}\phi_1(\mathbf{x}) \;\; \sqrt{\beta}\phi_2(\mathbf{x})\right], \left[\sqrt{\alpha}\phi_1(\mathbf{x}') \;\; \sqrt{\beta}\phi_2(\mathbf{x}')\right] \right\rangle \end{aligned}$$

where the notation $[\mathbf{u}\ \mathbf{v}]$ denotes the concatenation of the elements $\mathbf{u}$ and $\mathbf{v}$. This shows that $k(\mathbf{x},\mathbf{x}')$ can also be expressed as an inner dot product in the form (2.17), and, thus, is a (positive definite) kernel. ■

2. The product of two kernels is also a kernel, i.e., given two kernels $k_1(\mathbf{x},\mathbf{x}')$ and $k_2(\mathbf{x},\mathbf{x}')$, the following is a kernel:

$$k(\mathbf{x},\mathbf{x}') = k_1(\mathbf{x},\mathbf{x}')k_2(\mathbf{x},\mathbf{x}') \tag{2.20}$$

3. Consider a function $f : \mathcal{X} \mapsto \mathcal{X}$ and kernel $k_1(\mathbf{x},\mathbf{x}')$, then the following is a kernel:

$$k(\mathbf{x},\mathbf{x}') = k_1(f(\mathbf{x}), f(\mathbf{x}')) \tag{2.21}$$

Proof Using (2.17), the kernel $k_1(\cdot,\cdot)$ can be written as:

$$k_1(f(\mathbf{x}), f(\mathbf{x}')) = \langle\phi(f(\mathbf{x})), \phi(f(\mathbf{x}'))\rangle\,.$$

Since f is a transformation in the same space $\mathcal{X}$, then $k(\cdot,\cdot)$ can be simply thought of as a different kernel in the same space, i.e.:

$$k(\mathbf{x},\mathbf{x}') = k_1(f(\mathbf{x}), f(\mathbf{x}')) = \langle\phi(f(\mathbf{x})), \phi(f(\mathbf{x}'))\rangle = \langle\phi_f(\mathbf{x}), \phi_f(\mathbf{x}')\rangle$$

where $\phi_f : \mathcal{X} \mapsto \mathcal{H}'$ is a new feature map that maps input vectors from $\mathcal{X}$ into a Hilbert space $\mathcal{H}'$. Accordingly, $k(\mathbf{x},\mathbf{x}')$ can also be expressed as an inner dot product in the form (2.17), and, thus, is a (positive definite) kernel. ■

4. Consider a function $g : \mathcal{X} \mapsto \mathbb{R}$, then the following is a kernel:

$$k(\mathbf{x}, \mathbf{x}') = g(\mathbf{x})g(\mathbf{x}') \tag{2.22}$$

5. Consider a polynomial function f with positive coefficients and a kernel k_1, then the following is a kernel:

$$k(\mathbf{x}, \mathbf{x}') = f(k_1(\mathbf{x}, \mathbf{x}')) \tag{2.23}$$

Proof Since each polynomial term is a product of kernels with a positive coefficient, the proof follows by applying properties 1 (2.19) and 2 (2.20). ■

6. Consider a kernel k_1, then the following is a kernel:

$$k(\mathbf{x}, \mathbf{x}') = \exp(k_1(\mathbf{x}, \mathbf{x}')) \tag{2.24}$$

Proof By definition,

$$\exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots \tag{2.25}$$

From this we see that $\exp(x)$ is a sum of polynomial functions. Accordingly,

$$\exp(k_1(\mathbf{x}, \mathbf{x}')) = 1 + k_1(\mathbf{x}, \mathbf{x}') + \frac{k_1(\mathbf{x}, \mathbf{x}')^2}{2!} + \frac{k_1(\mathbf{x}, \mathbf{x}')^3}{3!} + \frac{k_1(\mathbf{x}, \mathbf{x}')^4}{4!} + \cdots \tag{2.26}$$

is a sum of polynomial functions of $k_1(\mathbf{x}, \mathbf{x}')$. By properties 1 (2.19) and 5 (2.23), $k(\mathbf{x}, \mathbf{x}')$ is a kernel. ■

7. (Cauchy-Schwartz inequality) If k is a positive definite kernel then

$$k(\mathbf{x}, \mathbf{x}')^2 \leq k(\mathbf{x}, \mathbf{x})k(\mathbf{x}', \mathbf{x}') \tag{2.27}$$

Proof In what follows, we will modify the notation for more simplicity and use $\mathbf{x}_1$ and $\mathbf{x}_2$ to denote $\mathbf{x}$ and $\mathbf{x}'$ respectively. Since k is a positive definite kernel, the 2×2 Gram matrix with entries $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ is positive definite. This implies that both its eigenvalues are non-negative, and so is their product, the determinant of the

Gram matrix K, i.e.:

$$0 \leq K_{11}K_{22} - K_{12}K_{21}$$

Next, substitute K_{ij} by $k(\mathbf{x}_i, \mathbf{x}_j)$ to get:

$$0 \leq k(\mathbf{x}_1, \mathbf{x}_1)k(\mathbf{x}_2, \mathbf{x}_2) - k(\mathbf{x}_1, \mathbf{x}_2)k(\mathbf{x}_2, \mathbf{x}_1)$$

Since $k(\cdot, \cdot)$ is symmetric (see Section 2.2.2), $k(\mathbf{x}_1, \mathbf{x}_2) = k(\mathbf{x}_2, \mathbf{x}_1)$, thus:

$$0 \leq k(\mathbf{x}_1, \mathbf{x}_1)k(\mathbf{x}_2, \mathbf{x}_2) - k(\mathbf{x}_1, \mathbf{x}_2)k(\mathbf{x}_1, \mathbf{x}_2)$$

By rearranging terms, we get the Cauchy-Schwartz inequality (2.27). ■

2.3 Reproducing Kernel Hilbert Spaces

Having formally defined the Hilbert space, we now introduce the Reproducing Kernel Hilbert Space (RKHS) [18] which is widely used in the area of statistical learning.

2.3.1 Construction of the RKHS

Consider a non-empty input space $\mathcal{X}$ and a map

$$\phi : \mathcal{X} \mapsto \mathbb{R}^{\mathcal{X}}$$

from $\mathcal{X}$ into the space of functions mapping $\mathcal{X}$ into $\mathbb{R}$, denoted as $\mathbb{R}^{\mathcal{X}}$. Here, $\phi(\cdot) = k(\cdot, \mathbf{x})$ denotes the function that assigns the value $k(\mathbf{x}', \mathbf{x})$ to $\mathbf{x}' \in \mathcal{X}$ where k is a symmetric positive definite kernel. Assume that k is real-valued (although the discussion can be extended for complex-valued k as well). In the remainder of this section, we will show how to construct a feature space associated with ϕ by proceeding as follows: First, construct a vector space containing the images of the inputs under ϕ. Second, define an inner product. Finally, show that this inner product satisfies (2.17): $\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle = k(\mathbf{x}, \mathbf{x}')$, $\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}$.

Step 1. We begin by defining a vector space containing the following linear combinations:

$$f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\cdot, \mathbf{x}_i) \tag{2.28}$$

where $n \in \mathbb{N}$, $\alpha_i \in \mathbb{R}$ and $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$ are arbitrary.

Step 2. Next, consider the function $g(\cdot) = \sum_{j=1}^{n'} \beta_j k(\cdot, \mathbf{x}'_j)$ with $n' \in \mathbb{N}$, $\beta_i \in \mathbb{R}$ and $\mathbf{x}'_1, \ldots, \mathbf{x}'_n \in \mathcal{X}$ and define an inner product between f and g as:

$$\langle f, g \rangle := \sum_{i=1}^{n} \sum_{j=1}^{n'} \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{x}'_j) \tag{2.29}$$

Notice that

$$\langle f, g \rangle = \sum_{i=1}^{n} \alpha_i g(\mathbf{x}_i) \tag{2.30}$$

and that

$$\langle f, g \rangle = \sum_{j=1}^{n'} \beta_j f(\mathbf{x}'_j). \tag{2.31}$$

We can see by the last two equations that $\langle f, g \rangle$ can be written as a linear combination of both its arguments, which proves that the defined product $\langle \cdot, \cdot \rangle$ is bilinear. Moreover, since the kernel $k(\cdot, \cdot)$ in (2.29) is symmetric, we have that $\langle f, g \rangle = \sum_{i,j} \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{x}'_j) = \sum_{i,j} \beta_j \alpha_i k(\mathbf{x}'_j, \mathbf{x}_i) = \langle g, f \rangle$, which proves that the defined product is symmetric. In addition, we have by assumption that k is a positive definite kernel which implies that, for any function f of the form (2.28), we have:

$$\langle f, f \rangle = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0 \tag{2.32}$$

(see Section 2.2.2 Definition 2). This proves that the defined product $\langle \cdot, \cdot \rangle$ is positive definite.

Using (2.30) with $g(\cdot) = k(\cdot, \mathbf{x})$, we have:

$$\langle k(., \mathbf{x}), f \rangle = \sum_{i=1}^{n} \alpha_i g(\mathbf{x}_i) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_i) = f(\mathbf{x})$$

where the last equality resulted from (2.28). This elegant property

$$\langle k(\cdot, \mathbf{x}), f \rangle = f(\mathbf{x}) \tag{2.33}$$

is known as the *reproducing property*. Using (2.33) with $f(\cdot) = k(\cdot, \mathbf{x}')$, we get:

$$\langle k(\cdot, \mathbf{x}), k(\cdot, \mathbf{x}') \rangle = k(\mathbf{x}, \mathbf{x}') \tag{2.34}$$

By virtue of what preceded, k is called a reproducing kernel, a term that was coined by Aronszajin in 1950 [20].

We have proved so far that the defined product $\langle \cdot, \cdot \rangle$ is 1) bilinear, 2) symmetric and 3) positive definite but still have to prove that the equality in (2.4) holds only for $f = 0$ (to be able to prove that the defined product is indeed an inner product). To this end, we note that, using (2.33) along with the Cauchy-Schwartz inequality (2.27), we have:

$$|f(\mathbf{x})|^2 = |\langle k(\cdot, \mathbf{x}), f \rangle|^2 \leq k(\mathbf{x}, \mathbf{x}) \cdot \langle f, f \rangle \tag{2.35}$$

which shows that $\langle f, f \rangle = 0$ directly implies that $f = 0$. By this, we have proved that all properties required for an inner product (according to Definition 1) are satisfied by the defined product $\langle \cdot, \cdot \rangle$.

Step 3. The above reasoning, particularly (2.34), illustrates what we already mentioned in Section 2.2.2, that a positive definite kernel k can be thought of as an inner product in another space. Using the definition of ϕ, (2.34) can be written as:

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle \tag{2.36}$$

which is the property of positive definite kernels that we saw in Section 2.2.2. Accordingly, the inner product space constructed in this way is a feature space associated with the kernel k. Given that k is a reproducing kernel, the feature space is called a Reproducing Kernel Hilbert Space (RKHS), defined formally next.

Definition 9. (Reproducing Kernel Hilbert Space [3]) *Let $\mathcal{X}$ be a nonempty set and $\mathcal{H}$ a Hilbert space of functions $f : \mathcal{X} \mapsto \mathbb{R}$. Then $\mathcal{H}$ is called a reproducing kernel Hilbert space endowed with the inner product $\langle \cdot, \cdot \rangle$ (and the norm $\|f\| := \sqrt{\langle f, f \rangle}$) if there exists a function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ with the following properties:*

1. *k has the reproducing property*

$$\langle k(\cdot, \mathbf{x}), f \rangle = f(\mathbf{x}), \ \ \forall f \in \mathcal{H}, \tag{2.37}$$

in particular,

$$\langle k(\cdot, \mathbf{x}), k(\cdot, \mathbf{x}') \rangle = k(\mathbf{x}, \mathbf{x}') \tag{2.38}$$

2. *k spans $\mathcal{H}$, i.e., $\mathcal{H} = \overline{span\{k(\cdot, \mathbf{x}) | \mathbf{x} \in \mathcal{X}\}}$ where $\overline{\mathcal{A}}$ denotes the completion of the space $\mathcal{A}$.*

2.4 Kernel Methods

Kernel methods, that is, techniques and algorithms that utilize kernels, are powerful non-linear techniques based on a non-linear transformation (namely $\phi(\cdot)$) of data $\mathbf{x}$ into a high-dimensional RKHS (the feature space) in which the transformed data $\phi(\mathbf{x})$ is more likely to be linearly separable. In the computer science and machine learning literature, the substitution of a high-dimensional dot product, $\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$, by $k(\mathbf{x}, \mathbf{x}')$, allowed by (2.36), has been referred to as the "kernel trick". By making use of this property, we have the ability to run inner product-based algorithms implicitly in the feature space by substituting the inner products by the corresponding kernels. This has far-reaching consequences as there are examples of positive definite kernels which can be evaluated efficiently even though, via (2.17), they correspond to inner products in infinite-dimensional feature spaces. One example is the Gaussian kernel [3]:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right), \quad \sigma > 0 \tag{2.39}$$

Using the kernel trick, "kernelized" extensions of many well-known algorithms were presented. These include kernel SVMs [3], kernel PCA [4] and kernel Fisher discriminant analysis [5], etc. The general idea is that, if we have an algorithm formulated in such a way that the input vector $\mathbf{x}$ enters only in the form of scalar products, then we can replace that scalar product with a kernel. Thus, we have the ability to develop non-linear generalizations of any algorithm that can be cast in terms of inner products.

2.5 The Representer Theorem

The Representer Theorem was first introduced by Kimeldorf and Wahba in 1971 [21] as an approach to solving optimization problems using kernels. Simply put, the theorem

shows that the solution of a certain class of optimization problems can be expressed as a finite linear combination of kernels evaluated at the input vectors. As we will see, this theorem proves to be quite useful from a practical standpoint as it dramatically simplifies the regularized risk minimization problem by providing a theoretical basis for the reduction of such problems to simple algorithms that we are able to solve efficiently. In this thesis, a more generalized version of the Representer Theorem is presented [22].

Theorem 1. (The Representer Theorem) *Let $\Omega : [0, \infty) \mapsto \mathbb{R}$ be a strictly monotonic increasing function, $\mathcal{X}$ be a non-empty set, $c : (\mathcal{X} \times \mathbb{R}^2)^n \mapsto \mathbb{R} \cup \{\infty\}$ be an arbitrary loss function and $\mathcal{H}$ be the RKHS associated with a kernel k. Then any minimizer $f \in \mathcal{H}$ of the regularized cost function*

$$c((\mathbf{x}_1, y_1, f(\mathbf{x}_1)), (\mathbf{x}_2, y_2, f(\mathbf{x}_2)), ..., (\mathbf{x}_n, y_n, f(\mathbf{x}_n))) + \Omega(\|f\|_{\mathcal{H}}^2) \tag{2.40}$$

admits a representation of the form

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_i), \;\; \alpha_i \in \mathbb{R} \tag{2.41}$$

Proof We begin by projecting f on the subspace:

$$\text{span}\{k(\cdot, \mathbf{x}_i) | 1 \leq i \leq n\}. \tag{2.42}$$

This results in a decomposition of f into two parts: one contained in the span of the kernel functions $k(\cdot, \mathbf{x}_1), k(\cdot, \mathbf{x}_2), \ldots, k(\cdot, \mathbf{x}_n)$ and one in its orthogonal complement, i.e.,

$$f(\mathbf{x}) = f_{\|}(\mathbf{x}) + f_{\perp}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_i) + f_{\perp}(\mathbf{x}). \tag{2.43}$$

This leads to

$$\|f\|^2 = \|f_{\|}\|^2 + \|f_{\perp}\|^2 \geq \|f_{\|}\|^2. \tag{2.44}$$

Since Ω is strictly monotonic increasing, we have that

$$\Omega(\|f\|^2) \geq \Omega(\|f_{\|}\|^2) \tag{2.45}$$

which implies that the second term in the cost function (2.40) will be minimized if the function f lies in the subspace (2.42) (i.e., the span of the kernels).

Next, we note that, since $f_\perp$ is the part of f contained in the orthogonal complement to the span of the kernels, then we have:

$$\langle f_\perp, k(\cdot, \mathbf{x}_i)\rangle = 0, \quad \forall f_\perp \in \mathcal{H}, \forall i \in \{1, ..., n\}. \tag{2.46}$$

Using (2.33), we can write $f(\mathbf{x}_j)$ for all $j \in \{1, ..., n\}$ as:

$$f(\mathbf{x}_j) = \langle f(\cdot), k(\cdot, \mathbf{x}_j)\rangle = \left\langle \left(f_\parallel(\cdot) + f_\perp(\cdot)\right), k(\cdot, \mathbf{x}_j)\right\rangle \tag{2.47}$$

where the second equality holds from (2.43). Since the inner product is linear in its first argument, we have:

$$f(\mathbf{x}_j) = f_\parallel(\mathbf{x}_j) + \langle f_\perp(\cdot), k(\cdot, \mathbf{x}_j)\rangle = f_\parallel(\mathbf{x}_j) \tag{2.48}$$

where the second equality is obtained using (2.46). This implies that the first term of the cost function (2.40) depends only on the component of f lying in the subspace (2.42) (i.e., the span of the kernels). And as already shown, the second term in the cost function (2.40) is minimized if the function f lies in that same subspace. Hence, the cost function in (2.40) will be minimized if the function f lies in that subspace as well which allows expressing the minimizer as the form in (2.41). This concludes the proof. ■

2.6 Summary

In this chapter, we presented a review of kernels which are at the heart of one of the popular classes of non-linear methods used in machine learning, namely the class of kernel methods. We began by reviewing some of the relevant mathematical background. Then we defined kernels as functions that measure the similarity of any two input vectors, and we presented their most important properties. We focused on the class of positive kernels as it allows using the kernel trick, which we defined as the substitution of a high-dimensional dot product by a kernel. Next, we introduced the reproducing kernel Hilbert space (RKHS) and showed how to construct it. We then defined kernel methods as techniques based on a non-linear transformation of data to a high-dimensional RKHS. In many learning problems, data patterns are complex, and so, simple classification or regression techniques perform

poorly in such learning problems. Kernel methods are thus presented as powerful tools to solve such problems. Finally, we presented the representer theorem which is widely used in conjunction with kernel methods as it shows that the solution to certain learning problems can be expressed as a finite linear combination of kernels evaluated at the input vectors. As such, the representer theorem presents a theoretical basis for the reduction of complex learning problems to simple algorithms that we are able to solve efficiently.

Chapter 3

Literature Review

In this chapter, we first review the classical Recursive Least Squares (RLS) algorithm. Then, we introduce its kernel extension, the kernel RLS (KRLS) algorithm presented by Engel et al. in [9]. KRLS obtains a non-linear least squares (LS) regressor as a linear combination of kernel functions evaluated at the elements of a carefully chosen subset, termed dictionary, of the received input vectors. Next, we review sparsification techniques which are the techniques by which we construct our dictionary, and which are widely used in conjunction with kernel adaptive filtering algorithms. Finally, we review pruning strategies which impose a hard limit on the size of the dictionary.

3.1 Recursive Least-Squares

This section presents the classical Recursive Least-Squares (RLS) algorithm [6]. Consider an online prediction setup where input-output pairs $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots\}$, are sequentially given, where we assume the inputs $\mathbf{x}_i \in \mathbb{R}^M$, and the desired outputs $y_i \in \mathbb{R}$. The objective is to predict the desired output using a function $f(\mathbf{x}) = \mathbf{w}_n^T \mathbf{x}$. The weight vector $\mathbf{w}_n \in \mathbb{R}^M$ is obtained by minimizing the cost function:

$$\mathcal{E}_n = \sum_{i=1}^{n} \lambda^{n-i} |e_i|^2 = \sum_{i=1}^{n} \lambda^{n-i} |y_i - \mathbf{w}_n^T \mathbf{x}_i|^2. \tag{3.1}$$

The constant λ, $0 < \lambda \leq 1$, is called the forgetting factor, and it provides the filter with the ability to follow the statistical variations of the observed data when it operates in a

non-stationary environment, by putting more emphasized on recent data, thus ensuring that data in the distant past are forgotten. Let $\mathbf{R}_n$ be the M-by-M correlation matrix defined by

$$\mathbf{R}_n \triangleq \sum_{i=0}^{n} \lambda^{n-i} \mathbf{x}_i \mathbf{x}_i^H, \tag{3.2}$$

and, assuming $\mathbf{R}_n$ is invertible, define $\mathbf{Q}_n \triangleq \mathbf{R}_n^{-1}$. Thus [6], given $\mathbf{Q}_{n-1}$ and $\mathbf{w}_{n-1}$, and with a new sample $\{\mathbf{x}_n, y_n\}$, the matrices $\mathbf{Q}_n$ and $\mathbf{w}_n$ can be computed using the following recursive procedure:

$$\mathbf{k}_n = \frac{\lambda^{-1}\mathbf{Q}_{n-1}\mathbf{x}_n}{1 + \lambda^{-1}\mathbf{x}_n^H \mathbf{Q}_{n-1}\mathbf{x}_n} \tag{3.3}$$

$$\xi_n = y_n - \mathbf{w}_{n-1}^H \mathbf{x}_n \tag{3.4}$$

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \mathbf{k}_n \xi_n^* \tag{3.5}$$

$$\mathbf{Q}_n = \lambda^{-1}\mathbf{Q}_{n-1} - \lambda^{-1}\mathbf{k}_n \mathbf{x}_n^H \mathbf{Q}_{n-1} \tag{3.6}$$

The vector $\mathbf{k}_n$ is called the gain vector and ξ_n is called the a priori estimation error. The procedure, given collectively by equations (3.3) to (3.6), constitutes the RLS algorithm.

3.2 Kernel Recursive Least-Squares

By virtue of the kernel trick, Engel et al. present in [9] an online kernel version of the RLS algorithm, namely kernel RLS (KRLS), which is able to efficiently and recursively solve non-linear LS prediction problems. The algorithm solves linear[1] regression problems in a high-dimensional feature space induced by the kernel where linear regression models of the transformed data are expected to be more accurate.

Consider again the prediction setup from Section 3.1. At each time step n, KRLS minimizes the sum of squared errors given by:

$$\mathcal{L} = \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2. \tag{3.7}$$

Let k be a kernel and $\mathcal{H}$ the RKHS associated with it. According to the Representer

[1] Linear in the high-dimensional space, but non-linear in the original input space.

Theorem [22], any minimizer $f \in \mathcal{H}$ of $\mathcal{L}$ in (3.7) admits a representation of the form

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_i), \;\; \alpha_i \in \mathbb{R}. \tag{3.8}$$

The importance of this theorem is that it ensures that the minimizer of (3.7) can be expressed as a finite linear combination of kernels evaluated at the input vectors, and therefore, identifying $f \in \mathcal{H}$ reduces to identifying the coefficients α_i, $i = 1, \ldots, n$. Indeed, substituting (3.8) in (3.7), $\mathcal{L}$ can be written as

$$\mathcal{L} = \|\mathbf{K}_n \boldsymbol{\alpha}_n - \mathbf{y}_n\|^2 \tag{3.9}$$

where $\mathbf{y}_n = [y_1, \ldots, y_n]^T$, $\mathbf{K}_n$ is the Gram matrix of the kernel k with entries $[\mathbf{K}_n]_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$ and $\boldsymbol{\alpha}_n = [\alpha_1, \ldots, \alpha_n]^T$ is a weight vector. Minimizing the loss function (3.9) gives the optimal vector

$$\boldsymbol{\alpha}_n = \mathbf{K}_n^{\dagger} \mathbf{y}_n \tag{3.10}$$

where

$$\mathbf{K}_n^{\dagger} = (\mathbf{K}_n^T \mathbf{K}_n)^{-1} \mathbf{K}_n^T \tag{3.11}$$

denotes the Moore-Penrose pseudo-inverse of $\mathbf{K}_n$. For a large number n of input vectors, storing $\mathbf{K}_n$ in memory and carrying out computations on it could prove cumbersome. Moreover, the size of the vector $\boldsymbol{\alpha}_n$ will be equal to the number of training samples, which would lead to severe overfitting [9]. Finally, inverting $\mathbf{K}_n$ might cause numerical instability given that the eigenvalues of $\mathbf{K}_n$ often decay rapidly to 0 as the number of received samples increases [9].

To overcome these shortcomings, one approach would be to limit the size of the matrix $\mathbf{K}_n$. Indeed, naively storing every received input vector in memory leads to an ever-growing kernel matrix and algorithm complexity as more input vectors points are received. For that, a sparsification procedure is commonly employed in conjunction with KRLS, that limits the size of $\mathbf{K}_n$ by forming it using a carefully chosen subset, termed dictionary, of the input vectors. This curbs its growth and helps in alleviating the aforementioned issues.

In the past two decades, sparsification techniques have received substantial attention in the fields of machine learning (specifically LS-SVMs [10] and neural networks [1]) and image

processing [23–26]. Sparsification techniques are also common in kernel adaptive filtering algorithms other than KRLS (e.g., KLMS [27] and [28]). In Section 3.3, we review recent work on sparsification techniques in kernel adaptive filtering algorithms. These techniques are also summarized in Table 3.1.

While these sparsification techniques curb the growth of the kernel matrix, they fall short of satisfying the often-arising need to fix the size of the kernel matrix, or in other words, to fix the number of input vectors that can be stored in the dictionary. This is important from a practical standpoint, e.g., when algorithms need to be implemented on DSP chips with finite memory and limited computational resources. Fixed-budget algorithms were presented as an answer to this need. In this type of algorithms, when the number of dictionary samples reaches a predefined limit, one sample is removed to accommodate the newest incoming sample. This process, known as pruning, requires selecting the specific sample for removal from the dictionary. Section 3.4 reviews pruning strategies used in different areas in machine learning, and Table 3.2 outlines those strategies. It is interesting to note that in all of the cited papers, the newest incoming sample is always added to the dictionary without any further verification about whether it would be useful to add this sample (e.g., in case a very similar sample is already stored in memory). This motivates the need for pre-processing incoming samples (e.g., by means of a sparsification method), a need that will be addressed in this research.

While sparsification and pruning techniques are important for the aforementioned reasons, they are also vital for equipping adaptive filtering algorithms, particularly KRLS algorithms, with the ability to track non-stationary systems. To describe well the variations of the input-output relationship over time, a large dictionary is appealing. However, since in all previous KRLS algorithms, the size of the weight vector was equal to that of the dictionary, allowing the dictionary to be large damages substantially the algorithm's ability to track changes in that input-output relationship. We can understand that from the fact that having to adjust a large number of weights significantly slows down adaptation [6], which weakens the algorithm's tracking ability. Accordingly, techniques such as sparsification and pruning that curb down the growth of the dictionary are useful for equipping adaptive filtering algorithms with a tracking ability. However, on their own, they might not be sufficient for obtaining a superior tracking ability as we will see later in this thesis.

3.3 Sparsification Techniques

Sparsification techniques are very popular in kernel adaptive filtering algorithms, especially in online prediction problems as they provide a solution to the problem of the growing size of the kernel matrix. In this section, we will review recent work on sparsification techniques in kernel adaptive filtering algorithms.

3.3.1 Approximate Linear Dependence

Consider again the online prediction setup from Section 2.2.1 where input-output pairs $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots\}$, $\mathbf{x}_i \in \mathcal{X}$, $y_i \in \mathbb{R}$, are sequentially given. Assume that at time step n, after having received $n-1$ training pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n-1}$, a dictionary $\mathcal{D}_{n-1} = \{\tilde{\mathbf{x}}_i\}_{i=1}^{m_{n-1}}$ has been constructed from a subset (of size m_{n-1}) of the received input vectors[2]. We emphasize that $\tilde{\mathbf{x}}_i$ should not be confused with $\mathbf{x}_i$ as $\tilde{\mathbf{x}}_i$ denotes the specific input vectors that have been admitted (to the dictionary) from all the received input vectors, $\mathbf{x}_i$.

The online sparsification method in [9] examines the relation between a new input vector and input vectors that were admitted to the dictionary. Particularly, when given a new input vector $\mathbf{x}_n$, we are presented with two cases:

1. If $\phi(\mathbf{x}_n)$ can be written approximately as a linear combination of $\{\phi(\tilde{\mathbf{x}}_i)\}_{i=1}^{m_{n-1}}$, then the new sample $\mathbf{x}_n$ is not admitted to the dictionary, with the weights corresponding to those previous samples being updated appropriately to reflect the dependence of the new sample on them.

2. If $\phi(\mathbf{x}_n)$ cannot be written approximately as a linear combination of $\{\phi(\tilde{\mathbf{x}}_i)\}_{i=1}^{m_{n-1}}$, the sample $\mathbf{x}_n$ is is added to the dictionary. The weight vector grows to include a new weight corresponding to $\mathbf{x}_n$.

For a new sample $\mathbf{x}_n$, we decide whether it should be added to the dictionary of samples by calculating:

$$\delta_n \triangleq \min_{\boldsymbol{a}} \left\| \sum_{i=1}^{m_{n-1}} a_i \phi(\tilde{\mathbf{x}}_i) - \phi(\mathbf{x}_n) \right\|^2 \tag{3.12}$$

[2]In this thesis, the terms "sample" and "input vector" will be used interchangeably. This flexibility in the terminology is particularly useful when we want to talk about the input vectors that were already admitted to the dictionary, and which can be simply referred to as "dictionary samples".

where $\boldsymbol{a} = [a_i, \ldots, a_{m_{n-1}}]^T$ and determining whether $\mathbf{x}_n$ satisfies the Approximate Linear Dependence (ALD) condition:

$$\delta_n \leq \nu \tag{3.13}$$

where ν is an accuracy parameter determining the level of sparsity. Indeed, the smaller ν is, the smaller the tolerable error in approximating the samples, thus the more accurate the KRLS prediction of the real outputs. Expanding (3.12) and making use of the kernel trick, we obtain:

$$\delta_n = \min_{\boldsymbol{a}} \left\{ \boldsymbol{a}^T \tilde{\mathbf{K}}_{n-1} \boldsymbol{a} - 2\boldsymbol{a}^T \tilde{\mathbf{k}}_{n-1} + k_{nn} \right\} \tag{3.14}$$

where $k_{nn} = k(\mathbf{x}_n, \mathbf{x}_n)$, and $\tilde{\mathbf{K}}_{n-1}$ and $\tilde{\mathbf{k}}_{n-1}$ are the matrices whose entries are respectively given by:

$$[\tilde{\mathbf{K}}_{n-1}]_{i,j} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) \tag{3.15}$$

and

$$[\tilde{\mathbf{k}}_{n-1}]_i = k(\tilde{\mathbf{x}}_i, \mathbf{x}_n) \tag{3.16}$$

with $i, j = 1, \ldots, m_{n-1}$. Solving (3.14) yields the optimal vector $\boldsymbol{a}_n$ given by:

$$\boldsymbol{a}_n = \tilde{\mathbf{K}}_{n-1}^{-1} \tilde{\mathbf{k}}_{n-1}. \tag{3.17}$$

By substituting (3.17) in (3.14), the ALD condition δ_n can be expressed as:

$$\delta_n = k_{nn} - \tilde{\mathbf{k}}_{n-1}^T \boldsymbol{a}_n \leq \nu. \tag{3.18}$$

KRLS and Sparsification

Define the matrix $\mathbf{P}_n \triangleq (\mathbf{A}_n^T \mathbf{A}_n)^{-1}$ where $\mathbf{A}_n$ is the matrix formed by the row-by-row concatenation of coefficient vectors $\boldsymbol{a}_n$ (3.17) found at specific iterations (as will be discussed in the two cases below). We note that it is very important that the coefficients $\boldsymbol{a}_n$ are not confused with the weights $\boldsymbol{\alpha}_n$ (which are also referred to as coefficients in the literature). For a given dictionary sample $\tilde{\mathbf{x}}_i$, the value a_i is the coefficient associated with $\phi(\tilde{\mathbf{x}}_i)$, that is, the coefficient that is attributed to that dictionary sample in the approximate linear combination that is checked for a new input vector $\mathbf{x}_n$ (to determine whether it should be admitted to the dictionary or not). On the other hand, the weight vector $\boldsymbol{\alpha}_n$ is the solution to the KRLS minimization problem (3.7).

Recall that $\boldsymbol{a}_n = \tilde{\mathbf{K}}_{n-1}^{-1}\tilde{\mathbf{k}}_{n-1}$ with $[\tilde{\mathbf{K}}_{n-1}]_{i,j} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$ and $[\tilde{\mathbf{k}}_{n-1}]_i = k(\tilde{\mathbf{x}}_i, \mathbf{x}_n)$ for $i, j = 1, \ldots, m_{n-1}$. Since the matrix $\mathbf{A}_n$ is formed by the vectors $\boldsymbol{a}_n$, and given the expression of $\boldsymbol{a}_n$, we can see that the matrix $\mathbf{P}_n$ is analogous –in the feature space– to the inverse correlation matrix $\mathbf{Q}_n$ (3.6) seen in the classical RLS. We note that in the actual algorithm, the matrix $\mathbf{A}_n$ is never computed explicitly, it is the matrix $\mathbf{P}_n$ that we actually calculate.

In Section 3.2, we expressed the solution of the KRLS minimization problem (3.9) $\mathcal{L} = \|\mathbf{K}_n\boldsymbol{\alpha}_n - \mathbf{y}_n\|^2$ as $\boldsymbol{\alpha}_n = (\mathbf{K}_n)^\dagger\mathbf{y}_n$. Making use of the kernel trick, we have: $\mathbf{K}_n = \boldsymbol{\Phi}_n^T\boldsymbol{\Phi}_n$ where $\boldsymbol{\Phi}_n = [\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)]$. Thus, we can express (3.9) as:

$$\mathcal{L} = \left\| \boldsymbol{\Phi}_n^T\mathbf{w}_n - \mathbf{y}_n \right\|^2 \tag{3.19}$$

where $\mathbf{w}_n = \boldsymbol{\Phi}_n\boldsymbol{\alpha}_n$.

As a result of applying the sparsifcation rule, the KRLS computations are in fact being carried out on a smaller $m_n \times m_n$ kernel matrix $\tilde{\mathbf{K}}_n$ (to which corresponds $\tilde{\boldsymbol{\Phi}}_n$) and we are solving for a vector $\tilde{\boldsymbol{\alpha}}_n$ of m_n weights (i.e., the weights resulting from having admitted only m_n samples). In [9], it is shown that

$$\boldsymbol{\Phi}_n = \tilde{\boldsymbol{\Phi}}_n\mathbf{A}_n^T. \tag{3.20}$$

Accordingly,

$$\mathbf{w}_n = \boldsymbol{\Phi}_n\boldsymbol{\alpha} = \tilde{\boldsymbol{\Phi}}_n\mathbf{A}_n^T\boldsymbol{\alpha} = \tilde{\boldsymbol{\Phi}}_n\tilde{\boldsymbol{\alpha}}_n, \tag{3.21}$$

where we define $\tilde{\boldsymbol{\alpha}}_n := \mathbf{A}_n^T\boldsymbol{\alpha}_n$. The vector $\tilde{\boldsymbol{\alpha}}_n$ is the "reduced" weight vector introduced earlier. Plugging (3.21) in (3.19), the loss becomes:

$$\mathcal{L} = \left\| \boldsymbol{\Phi}_n^T\tilde{\boldsymbol{\Phi}}_n\tilde{\boldsymbol{\alpha}}_n - \mathbf{y}_n \right\|^2. \tag{3.22}$$

Plugging (3.20) in (3.22) and making use of the kernel trick, the loss can be expressed as:

$$\mathcal{L} = \left\| \mathbf{A}_n\tilde{\mathbf{K}}_n\tilde{\boldsymbol{\alpha}}_n - \mathbf{y}_n \right\|^2, \tag{3.23}$$

and its minimizer is

$$\tilde{\boldsymbol{\alpha}}_n = (\mathbf{A}_n\tilde{\mathbf{K}}_n)^\dagger\mathbf{y}_n = \tilde{\mathbf{K}}_n^{-1}(\mathbf{A}_n^T\mathbf{A}_n)^{-1}\mathbf{A}_n^T\mathbf{y}_n. \tag{3.24}$$

Formulation of ALD-KRLS

At each time step n, a new input vector $\mathbf{x}_n$ is observed along with the corresponding target value y_n. After calculating $\boldsymbol{a}_n$ using (3.17) and δ_n using (3.18), the ALD condition is checked and we are presented with one of two cases:

CASE 1: $\delta_n > \nu$, that is, $\phi(\mathbf{x}_n)$ is not ALD on $\mathcal{D}_{n-1}$.

In this case, $\mathbf{x}_n$ is added to the dictionary so $\mathcal{D}_n = \mathcal{D}_{n-1} \cup \{\mathbf{x}_n\}$, $m_n = m_{n-1} + 1$ and $\tilde{\mathbf{K}}_n$ gets "upsized", that is, a new row and a new column corresponding to the newly admitted dictionary sample are added to $\tilde{\mathbf{K}}_n$. A recursive formula for its inverse $\tilde{\mathbf{K}}_n^{-1}$ is derived as:

$$\tilde{\mathbf{K}}_n^{-1} = \frac{1}{\delta_n} \begin{bmatrix} \delta_n \tilde{\mathbf{K}}_{n-1}^{-1} + \boldsymbol{a}_n \boldsymbol{a}_n^T & -\boldsymbol{a}_n \\ -\boldsymbol{a}_n^T & 1 \end{bmatrix}. \tag{3.25}$$

Since $\mathbf{x}_n$ has just been admitted to the dictionary, no other dictionary sample is ALD on it. In other words, $\phi(\mathbf{x}_n)$ is exactly represented by itself, and at this stage, is not expressed as an approximate linear combination of any other dictionary samples (except itself). Therefore, the corresponding coefficient a_n to $\phi(\mathbf{x}_n)$ is equal to 1. Accordingly, the matrix $\mathbf{A}_n$ of coefficients is expanded by adding a coefficient of 1 for the newly admitted sample at the (n,n)-th entry of matrix $\mathbf{A}_n$, and setting the other entries in the new row and column to zero, i.e.,

$$\mathbf{A}_n = \begin{bmatrix} \mathbf{A}_{n-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \implies \mathbf{A}_n^T \mathbf{A}_n = \begin{bmatrix} \mathbf{A}_{n-1}^T \mathbf{A}_{n-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}. \tag{3.26}$$

A recursive formula for $\mathbf{P}_n$ is thus given by:

$$\mathbf{P}_n = (\mathbf{A}_n^T \mathbf{A}_n)^{-1} = \begin{bmatrix} \mathbf{P}_{n-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{3.27}$$

where $\mathbf{0}$ is a vector of zeros of appropriate length. By adding an element to the dictionary, the weight vector $\tilde{\boldsymbol{\alpha}}_n$ also grows, and its recursive update rule is derived as:

$$\tilde{\boldsymbol{\alpha}}_n = \begin{bmatrix} \tilde{\boldsymbol{\alpha}}_{n-1} - \frac{\boldsymbol{a}_n}{\delta_n} \left(y_n - \tilde{\mathbf{k}}_{n-1}^T \tilde{\boldsymbol{\alpha}}_{n-1} \right) \\ \frac{1}{\delta_n} \left(y_n - \tilde{\mathbf{k}}_{n-1}^T \tilde{\boldsymbol{\alpha}}_{n-1} \right) \end{bmatrix}. \tag{3.28}$$

CASE 2: $\delta_n \leq \nu$, that is, $\phi(\mathbf{x}_n)$ is ALD on $\mathcal{D}_{n-1}$.

In this case, the sample is not added to the dictionary so $\mathcal{D}_n = \mathcal{D}_{n-1}$, $m_n = m_{n-1}$ and $\tilde{\mathbf{K}}_n = \tilde{\mathbf{K}}_{n-1}$. Being the matrix of coefficients, $\mathbf{A}_n$ changes as follows:

$$\mathbf{A}_n = [\mathbf{A}_{n-1}^T, \boldsymbol{a}_n]^T, \tag{3.29}$$

thus leading to a change in $\mathbf{P}_n = (\mathbf{A}_n^T \mathbf{A}_n)^{-1}$. Using the matrix inversion lemma (See [6]: Page 745, Lemma A.1), a recursive formula for $\mathbf{P}_n$ is obtained:

$$\mathbf{P}_n = \mathbf{P}_{n-1} - \frac{\mathbf{P}_{n-1}\boldsymbol{a}_n\boldsymbol{a}_n^T\mathbf{P}_{n-1}}{1 + \boldsymbol{a}_n^T\mathbf{P}_{n-1}\boldsymbol{a}_n}. \tag{3.30}$$

Defining $\mathbf{q}_n \triangleq \frac{\mathbf{P}_{n-1}\boldsymbol{a}_n}{1+\boldsymbol{a}_n^T\mathbf{P}_{n-1}\boldsymbol{a}_n}$, we also obtain a recursive update rule for $\tilde{\boldsymbol{\alpha}}_n$:

$$\tilde{\boldsymbol{\alpha}}_n = \tilde{\boldsymbol{\alpha}}_{n-1} + \tilde{\mathbf{K}}_n^{-1}\mathbf{q}_n \left(y_n - \tilde{\mathbf{k}}_{n-1}^T\tilde{\boldsymbol{\alpha}}_{n-1}\right). \tag{3.31}$$

Algorithm 1 presents the complete algorithm denoted by ALD-KRLS.

Algorithm 1 ALD-KRLS

Parameter: ν

Initialize: $\tilde{\mathbf{k}}_1 = [k_{11}]$, $\tilde{\mathbf{k}}_1^{-1} = [1/k_{11}]$, $\tilde{\boldsymbol{\alpha}}_n = (y_1/k_{11})$, $\mathbf{P}_1 = [1]$, $m = 1$.

for $n = 2, 3, \ldots$ **do**

1. Get new sample: $(\mathbf{x}_n, y_n)$

2. Compute: $\tilde{\mathbf{k}}_{n-1}$ using (3.16)

3. ALD Test:

$\boldsymbol{a}_n = \tilde{\mathbf{K}}_{n-1}^{-1}\tilde{\mathbf{k}}_{n-1}$

$\delta_n = k_{nn} - \tilde{\mathbf{k}}_{n-1}^T\boldsymbol{a}_n$

if $\delta_n > \nu$ **then** {*Add* $\mathbf{x}_n$ *to dictionary*}

$\mathcal{D}_n = \mathcal{D}_{n-1} \cup \{\mathbf{x}_n\}$

Compute $\tilde{\mathbf{K}}_n^{-1}$ using (3.25)

Compute $\mathbf{P}_n$ using (3.27)

Compute $\tilde{\boldsymbol{\alpha}}_n$ using (3.28)

$m := m + 1$

else {*dictionary unchanged*}

$\mathcal{D}_n = \mathcal{D}_{n-1}$

$\mathbf{q}_n = \frac{\mathbf{P}_{n-1}\boldsymbol{a}_n}{1+\boldsymbol{a}_n^T\mathbf{P}_{n-1}\boldsymbol{a}_n}$

Compute $\mathbf{P}_n$ using (3.30)

Compute $\tilde{\boldsymbol{\alpha}}_n$ using (3.31)

end if

end for

Output: $\mathcal{D}_t$, $\tilde{\boldsymbol{\alpha}}_n$

3.3.2 Surprise Criterion

In [11], Liu et al. approach the issue of the growing size of the kernel matrix from an information theoretic perspective. They propose a criterion that measures the information a point can contribute to the knowledge of the learning system, based on Gaussian processes theory.

The criterion, called the "surprise," quantifies how much information a new sample point contains relatively to the already accumulated knowledge of the learning system. If, for example, that information is redundant, the new sample point leads to little surprise,

and so, it is not admitted into the dictionary.

Definition of Surprise

Surprise, denoted by $\mathcal{S}_{\mathcal{T}}(\mathbf{x}, y)$, is a subjective information measure of a pair $\{\mathbf{x}, y\}$ with respect to a learning system $\mathcal{T}$ which consists of the dictionary of admitted inputs and the dictionary of their corresponding outputs. Surprise is defined as:

$$\mathcal{S}_{\mathcal{T}}(\mathbf{x}, y) = -\ln p(\mathbf{x}, y|\mathcal{T}) \tag{3.32}$$

where $p(\mathbf{x}, y|\mathcal{T})$ is the posterior probability distribution of $\{\mathbf{x}, y\}$ given $\mathcal{T}$.

Intuitively, if the probability of an incoming sample given the dictionary is very high, the sample is not "surprising" for the learning system (the value of the surprise is small). However, if that probability is low, then the surprise is high, which suggests that the new sample point has new information for the system or that it's abnormal (an outlier) if the surprise is too high. Accordingly, based on the measure of surprise, we can define three categories for the sample points:

- *abnormal*: $\mathcal{S}_{\mathcal{T}}(\mathbf{x}, y) > T_1$
- *learnable*: $T_1 \geq \mathcal{S}_{\mathcal{T}}(\mathbf{x}, y) \geq T_2$
- *redundant*: $\mathcal{S}_{\mathcal{T}}(\mathbf{x}, y) < T_2$

where T_1 and T_2 are problem-dependent parameters.

Evaluation of Surprise

In Gaussian Processes Regression [29], the predicted outputs, $f(\mathbf{x}_i), i = 1, \ldots, n$ (with n denoting the current iteration), are assumed to have a jointly Gaussian prior distribution, that is,

$$[f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)]^T \sim \mathcal{N}(0, \sigma^2\mathbf{I} + \mathbf{G}_n), \ \ \forall n \tag{3.33}$$

where

$$\mathbf{G}_n = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_n, \mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_1, \mathbf{x}_n) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}, \tag{3.34}$$

and σ^2 is the variance of the noise contained in the samples. With this assumption, the posterior probability distribution $p(y_n|\mathbf{x}_n, \mathcal{T}_{n-1})$ of the predicted output y_n given $\mathbf{x}_n$ and $\mathcal{T}_{n-1}$ is also normally distributed with mean

$$\bar{y}_n = \mathbf{h}_n^T[\sigma^2\mathbf{I} + \mathbf{G}_{n-1}]^{-1}\mathbf{y}_{n-1} \tag{3.35}$$

and variance

$$\sigma_n^2 = \sigma^2 + k(\mathbf{x}_n, \mathbf{x}_n) - \mathbf{h}_n^T[\sigma^2\mathbf{I} + \mathbf{G}_{n-1}]^{-1}\mathbf{h}_n \tag{3.36}$$

where $\mathbf{h}_n = [k(\mathbf{x}_n, \mathbf{x}_1), \ldots, k(\mathbf{x}_n, \mathbf{x}_{n-1})]^T$ and $\mathbf{y}_{n-1} = [y_1, \ldots, y_{n-1}]^T$. The posterior joint probability distribution $p(\mathbf{x}_n, y_n|\mathcal{T}_{n-1})$ from (3.32) becomes:

$$\begin{aligned} p(\mathbf{x}_n, y_n|\mathcal{T}_{n-1}) &= p(y_n|\mathbf{x}_n, \mathcal{T}_{n-1})p(\mathbf{x}_n|\mathcal{T}_{n-1}) \\ &= \frac{1}{\sqrt{2\pi}\sigma_n}\exp\left(-\frac{(y_n - \bar{y}_n)^2}{2\sigma_n^2}\right)p(\mathbf{x}_n|\mathcal{T}_{n-1}). \end{aligned} \tag{3.37}$$

Taking the negative logarithm of (3.37), the surprise is given by:

$$\mathcal{S}_{\mathcal{T}_{n-1}}(\mathbf{x}_n, y_n) = \ln(\sqrt{2\pi}) + \ln(\sigma_n) + \frac{(y_n - \bar{y}_n)^2}{2\sigma_n^2} - \ln p(\mathbf{x}_n|\mathcal{T}_{n-1}). \tag{3.38}$$

For simplicity, $\mathcal{S}_{\mathcal{T}_{n-1}}(\mathbf{x}_n, y_n)$ will be denoted as $\mathcal{S}_n$ henceforth. The distribution $p(\mathbf{x}_n|\mathcal{T}_{n-1})$ is problem-dependent. In the regression literature, it is often assumed that $p(\mathbf{x}_n|\mathcal{T}_{n-1}) = p(\mathbf{x}_n)$, i.e., the distribution of $\mathbf{x}_n$ is independent of the previous inputs and corresponding outputs. In general, we can assume the distribution $\mathbf{x}_n$ is uniform if no a priori information is available [9].

We can see from equation (3.38) why a "rare" observation leads to a high value for the surprise. This is due to (the negative of the logarithm of) the small $p(\mathbf{x}_n|\mathcal{T}_{n-1})$ that leads to a larger $\mathcal{S}_n$. Also, $e_n = y_n - \bar{y}_n$ can be seen as the prediction error since $\bar{y}_n$ is the maximum a posteriori (MAP) estimate of y_n by the current learning system [11]. Predicting y_n well means e_n^2 is small, which explains, in light of (3.38), why we get a little surprise when the system is predicting well near $\mathbf{x}_n$. On the other hand, for a small variance and a large prediction error, the third term in (3.38) becomes very large and dominates $\mathcal{S}_n$ which becomes consequently very large. This can be understood as an abnormality which leads to the corresponding sample being discarded.

KRLS with Surprise Criterion

It can be verified that $\bar{y}_n$ in (3.35) and σ_n^2 in (3.36) are in fact equal to $f(\mathbf{x}_n) = \tilde{\mathbf{k}}_n^T \tilde{\boldsymbol{\alpha}}_n$ (see (3.16) and (3.28)) and to δ_n (see (3.18)), respectively. Therefore, from (3.38), we see that the surprise criterion for KRLS is given by:

$$\mathcal{S}_n = \frac{1}{2} \ln \delta_n + \frac{(y_n - \tilde{\mathbf{k}}_n^T \tilde{\boldsymbol{\alpha}}_n)^2}{2\delta_n} - \ln p(\mathbf{x}_n | \mathcal{T}_{n-1}) \tag{3.39}$$

where $p(\mathbf{x}_n|\mathcal{T}_{n-1})$ can be assumed to be constant if no a priori information is available. Note that the first (constant) term in (3.38) was ignored since the thresholds, T_1 and T_2, against which $\mathcal{S}_n$ is compared can be adjusted accordingly.

As a result, the algorithm for KRLS with Surprise Criterion, denoted as SC-KRLS, is the same as Algorithm 1 with the only change being the condition for deciding whether to add a sample to the dictionary or not. Specifically, here, $\mathcal{S}_n$ will be used instead of δ_n to determine whether the sample is learnable, abnormal or redundant. If the sample is abnormal, it is discarded and no update is carried out for the algorithm variables since we have no interest in extracting information from an abnormal sample. If it is redundant, the sample is not added to the dictionary but the update rules (3.30)-(3.31) are executed. If the sample is learnable, it is is added to the dictionary and the corresponding update rules (3.25), (3.27) and (3.28) are executed.

3.3.3 Quantization Technique

In [28], a quantization approach is presented to reduce the growth of the kernel matrix of the KLMS algorithm, but can be used in conjunction with KRLS as well. The basic idea behind the quantization method is to quantize and hence compress the input space by selecting specific samples to be admitted to the dictionary: a sample is admitted to the dictionary only if it is not close enough to the dictionary samples.

Given a new sample $\mathbf{x}_n$, the sparsification technique is carried out by first computing the distance between $\mathbf{x}_n$ and every sample in the dictionary $\mathcal{D}_{n-1}$. Denote by $\tilde{\mathbf{x}}_q$ the element in $\mathcal{D}_{n-1}$ that has the smallest distance to $\mathbf{x}_n$, and call $d_{\min}$ that distance, i.e.,

$$d_{\min} = ||\mathbf{x}_n - \tilde{\mathbf{x}}_q|| = \min_{1 \le i \le m_{n-1}} ||\mathbf{x}_n - \tilde{\mathbf{x}}_i|| \tag{3.40}$$

where $||\cdot||$ denotes the Euclidean distance. Then, $d_{\min}$ is checked against a quantization parameter ε. If $d_{\min}$ is larger than ε, then the new sample $\mathbf{x}_n$ is added to the dictionary (and, obviously, a new coefficient is appended in the weight vector for that newest entry to $\mathcal{D}_n$). If that distance is smaller than ε, i.e., the new sample $\mathbf{x}_n$ is close enough to $\tilde{\mathbf{x}}_q$, the new sample is not added to the dictionary and the coefficient of $\tilde{\mathbf{x}}_q$ (that contributes to the KLMS solution) is updated to reflect that a new sample has been quantized to it as follows:

$$\boldsymbol{\alpha}_q = \boldsymbol{\alpha}_q + \mu e_n \tag{3.41}$$

where μ is an algorithm parameter (learning step) and $e_n = y_n - \hat{y}_n$, y_n being the real value of the output and $\hat{y}_n$ the KLMS-predicted value of the output.

3.3.4 Novelty Sparsification Rule

In [30], a two-fold sparsification rule is presented to determine if a sample is "novel" enough to be added to the dictionary. Given a new input-output pair $\{\mathbf{x}_n, y_n\}$, first, we determine the distance between $\mathbf{x}_n$ and the dictionary as defined in (3.40). Second, we measure the difference between the real output y_n and the predicted output $\hat{y}_n$ for that sample. Accordingly, the incoming sample is added to the dictionary if the two following conditions are satisfied:

$$\min_{1\leq i\leq m_{n-1}} ||\mathbf{x}_n - \tilde{\mathbf{x}}_i|| > \varepsilon_x \quad \text{and} \quad |y_n - \hat{y}_n| > \varepsilon_y \tag{3.42}$$

where ε_x and ε_y are sparsification thresholds that determine the level of sparsity of the dictionary. If at least one of the two conditions in (3.42) is not satisfied, the sample is discarded.

3.3.5 Significance Criterion

In [31], Fan and Song consider the "significance" of each incoming input sample $\mathbf{x}_i$ by evaluating the impact of a sample on the loss function (that is minimized in the KRLS problem). They consider a modified version of the loss function (3.19) (used in ALD-KRLS [9]) by including a forgetting factor β as follows:

$$\mathcal{L}_n = \sum_{i=1}^{n} \beta^{n-i} \left| y_i - \mathbf{w}_n^T \phi(\mathbf{x}_i) \right|^2. \tag{3.43}$$

The sparsification criterion they adopt is the change in $\mathcal{L}_n$ which would occur if a new sample added to the dictionary, and which can be expressed at iteration n as:

$$\Delta\mathcal{L}_n = \frac{1}{2}\Delta\boldsymbol{\alpha}_n^T\mathbf{H}_n\Delta\boldsymbol{\alpha}_n \tag{3.44}$$

where $\mathbf{H}_n$ is the Hessian matrix of the loss function (3.43) at iteration n and $\Delta\boldsymbol{\alpha}_n$ is the difference in the value of the weight vector $\boldsymbol{\alpha}$ in the two cases: (1) when the sample is not added and (2) if the sample is added. The difference $\Delta\boldsymbol{\alpha}_n$ is derived as:

$$\Delta\boldsymbol{\alpha}_n = \begin{bmatrix} -\frac{\mathbf{P}_n\tilde{\mathbf{k}}_n}{z_n}e_n \\ \frac{1}{z_n}e_n \end{bmatrix} \tag{3.45}$$

where e_n is the prediction error $e_n = y_n - \hat{y} = y_n - \tilde{\mathbf{k}}_n^T\tilde{\boldsymbol{\alpha}}_n$ and $z_n = 1 - \tilde{\mathbf{k}}_n^T\mathbf{P}_n\tilde{\mathbf{k}}_n$, with $\tilde{\mathbf{k}}_n$ and $\tilde{\boldsymbol{\alpha}}_n$ as defined in Section 3.3.1. Here, $\mathbf{P}_n$ is a modified version of the matrix considered in the ALD-KRLS case [9] in which a forgetting factor was not considered in the initial minimization problem (3.19). Hence, $\mathbf{P}_n$ becomes $(\mathbf{A}_n^T\boldsymbol{\Lambda}_n\mathbf{A}_n)^{-1}$ where $\boldsymbol{\Lambda}_n$ is the diagonal matrix with diagonal entries: $\beta^n, \beta^{n-1}, \ldots, 1$. By taking the second derivative of the loss function (3.43) with respect to the kernel weight $\boldsymbol{\alpha}_n$, the $\mathbf{H}_n$ matrix can be obtained as:

$$\mathbf{H}_n = \frac{\partial^2\mathcal{L}_n}{\partial^2\boldsymbol{\alpha}_n} = \mathbf{R}_n \tag{3.46}$$

where $\mathbf{R}_n = \tilde{\mathbf{k}}_n\mathbf{y}_n$. The change in the loss function $\Delta\mathcal{L}_n$ measures the impact, or significance, of an incoming input sample on the loss function. Accordingly, if the significance of a specific input sample exceeds a predefined threshold, the contribution of such sample is substantial enough, and it is added to the dictionary. Otherwise, the sample is discarded.

3.3.6 Mutual Information Criterion

In [32], Fan et al. propose a sparsification method based on the estimated instantaneous mutual information between the incoming input sample and the corresponding predicted output. The instantaneous mutual information is a measure of the amount of information shared between the input $\mathbf{x}_n$ and the predicted output $\hat{y}_n$. It is formally defined as:

$$I(\mathbf{x}_n; \hat{y}_n|\mathcal{T}_n) = H(\hat{y}_n|\mathcal{T}_n) - H(\hat{y}_n|\mathbf{x}_n, \mathcal{T}_n) \tag{3.47}$$

where $\mathcal{T}_n$ is the learning system, which consists, as before, of the dictionary of admitted inputs and the dictionary of their corresponding outputs, at iteration n and the entropies $H(\hat{y}_n|\mathcal{T}_n)$ and $H(\hat{y}_n|\mathbf{x}_n, \mathcal{T}_n)$ are respectively given by:

$$\begin{aligned} H(\hat{y}_n|\mathcal{T}_n) &= -\log p(\hat{y}_n|\mathcal{T}_n) \\ H(\hat{y}_n|\mathbf{x}_n, \mathcal{T}_n) &= -\log p(\hat{y}_n|\mathbf{x}_n, \mathcal{T}_n), \end{aligned} \tag{3.48}$$

with $p(\hat{y}_n|\mathcal{T}_n)$ being the system output probability density and $p(\hat{y}_n|\mathbf{x}_n, \mathcal{T}_n)$ being the conditional probability density of system output $\hat{y}_n$ given the input $\mathbf{x}_n$ and the dictionary $\mathcal{T}_n$.

Kernel density estimators [33] are non-parametric probability density function estimation methods that exhibit fast convergence in the mean-square sense. The kernel density estimator with Gaussian kernel was used in [32] to estimate $p(\hat{y}_n|\mathcal{T}_n)$ and $p(\hat{y}_n|\mathbf{x}_n, \mathcal{T}_n)$. Accordingly, the entropies can be obtained as:

$$\begin{aligned} H(\hat{y}_n|\mathcal{T}_n) &= -\log\left(\frac{1}{m_{n-1}} \sum_{i=1}^{m_{n-1}} k(\hat{y}_n, \hat{y}_i)\right) \\ H(\hat{y}_n|\mathbf{x}_n, \mathcal{T}_n) &= -\log\left(\frac{1}{b} \sum_{i=1}^{b} k(e_n, e_i)\right) \end{aligned} \tag{3.49}$$

where $k(\cdot, \cdot)$ is the Gaussian kernel defined in (2.39), the $\hat{y}_i$'s are the predicted outputs of the samples $\{\tilde{\mathbf{x}}_i, y_i\}$ stored in the dictionary with m_{n-1} being the number of samples stored in the dictionary, and $e_i = y_i - \hat{y}_i$ is the prediction error (between the real output and the predicted output) computed for the b nearest input samples. Similarly to the Surprise Criterion test (Section 3.3.2), the usefulness of incoming samples is determined by measuring the value of a sample's mutual information using (3.47) and (3.49) and comparing that value to pre-specified thresholds. Accordingly, incoming samples can be characterized as redundant if the mutual information is very high (exceeding a certain threshold) and abnormal if the mutual information is too small (in that case, the input $\mathbf{x}_n$ and the estimated output $\hat{y}_n$ are almost irrelevant to the former training samples). If the mutual information lies within a certain range between the two former cases, the incoming sample is deemed to be informative and thus added to the dictionary.

3.3.7 Coherence Criterion

In [34], Richard et al. present a coherence-based sparsification rule in the context of online prediction of time series using kernels. They define coherence as:

$$\mu = \max_{i \neq j} |k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)|, \ \ \forall \tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j \in \mathcal{D}_n \tag{3.50}$$

where $|\cdot|$ denotes absolute value and $k(\cdot, \cdot)$ is a unit-norm kernel, i.e., $k(\mathbf{x}, \mathbf{x}) = 1$ for all $\mathbf{x}$. By observing the definition, it can be seen that μ is the largest absolute value of the off-diagonal entries in the Gram matrix. Thus, coherence reflects the largest cross-correlations in the dictionary. A dictionary is said to be incoherent when μ is small. Based on this definition of coherence, when an incoming sample $\{\mathbf{x}_n, y_n\}$ is received, the following sparsification rule is checked:

$$\max_i |k(\mathbf{x}_n, \tilde{\mathbf{x}}_i)| \leq \mu_0, \ \ \forall \tilde{\mathbf{x}}_i \in \mathcal{D}_{n-1} \tag{3.51}$$

where $\mu_0 \in [0, 1)$ is a parameter that determines the level of sparsity and coherence of the dictionary. Accordingly, if the incoming sample has a small enough coherence with the dictionary (satisfying condition (3.51)), it is added to the dictionary. Otherwise, it is discarded with the weight coefficients being updated appropriately.

3.3.8 Prediction Error Criterion

In [35], a sparsification rule based on the prediction error is presented. When an incoming sample $\{\mathbf{x}_n, y_n\}$ is received, its predicted output $\hat{y}_n$ is computed. Let the prediction error e_n be the error between the real output y_n and the predicted output $\hat{y}$. Based on this definition, the sparsification rule is as follows:

$$|e_n| = |y_n - \hat{y}_n| \leq \delta, \tag{3.52}$$

that is, if the absolute of value of the error $|e_n|$ is less than a given threshold δ, then the system is accurate enough and does not need to be improved. Thus, the sample is discarded with the weight coefficients being updated appropriately. Otherwise, the sample is added.

Table 3.1 summarizes the sparsification techniques presented in Section 3.3.

Table 3.1 Different Sparsification Criteria

Notation:	$\mathbf{x}_n$	Input sample at iteration n
	y_n	Real output at iteration n
	$\hat{y}_n$	Predicted output at iteration n
	$\mathcal{D}_n$	Dictionary of samples at iteration n
	$\tilde{\mathbf{x}}_i$	i-th entry in the dictionary at iteration n
	$\boldsymbol{\alpha}_n$	Weight vector at iteration n
	m_{n-1}	Size of dictionary at iteration $n-1$

Criterion	**Expression**	**Adopted in**
Approximate Linear Dependence	$\delta_n = \min_{\boldsymbol{a}} \left\|\sum_{i=1}^{m_{n-1}} a_i \phi(\tilde{\mathbf{x}}_i) - \phi(\mathbf{x}_n)\right\|^2$ where a_i: coefficients	[9], [36], [37]
Surprise Criterion	$\mathcal{S}_{\mathcal{D}}(\mathbf{x}, y) = -\ln p(\mathbf{x}, y\|\mathcal{D})$	[11]
Quantization Criterion	$d_{\min} = \min_{1 \le i \le m_{n-1}} \|\|\mathbf{x}_n - \tilde{\mathbf{x}}_i\|\|$	[28]
Novelty Criterion	$d_{\min} = \min_{1 \le i \le m_{n-1}} \|\|\mathbf{x}_n - \tilde{\mathbf{x}}_i\|\|$ and $\|\|y_n - \hat{y}_n\|\|$	[30]
Significance Criterion	$\Delta\mathcal{L}_n = \frac{1}{2}\Delta\boldsymbol{\alpha}_n^T \mathbf{H}_n \Delta\boldsymbol{\alpha}_n$ where $\mathbf{H}_n = \frac{\partial^2 \mathcal{L}_n}{\partial^2 \boldsymbol{\alpha}_n}$	[31]
Mutual Information Criterion	$I(\mathbf{x}_n; \hat{y}_n\|\mathcal{D}_n) = H(\hat{y}_n\|\mathcal{D}_n) - H(\hat{y}_n\|\mathbf{x}_n, \mathcal{D}_n)$ where $H(U\|V) = -\log p(U\|V)$	[32]
Coherence Criterion	$\mu = \max_{1 \le i \le m_{n-1}} \|k(\mathbf{x}_n, \tilde{\mathbf{x}}_i)\|$	[34]
Prediction Error Criterion	$\|e_n\| = \|y_n - \hat{y}_n\|$	[35]

3.4 Pruning Strategies

The sparsification techniques presented in Section 3.3 are quite useful in curbing the growth of the dictionary and the corresponding Gram matrix. However, often, a more aggressive approach is needed particularly when practical considerations are taken into account. For example, algorithm implementations on DSP processors would impose more restrictive limitations, e.g., limited memory size and computational resources. Fixed-budget algorithms provide a solution to this problem as they entail a more constraining approach than the sparsification approach: A bound, namely M, is imposed on the number of samples that can be stored in the dictionary of samples at any point in time. This is also particularly important for tracking time-varying systems as a large size of the dictionary has a detrimental effect on the algorithm's tracking ability. As a result of imposing this hard bound in fixed-budget algorithms, whenever the number of dictionary samples exceeds the limit, a sample must be discarded. This is known as pruning and in this section, we will be reviewing pruning strategies from different areas in machine learning.

3.4.1 Remove-the-Oldest Strategy

One of the simplest pruning approaches is to remove the oldest sample in the dictionary. This means that when the number of dictionary samples reaches the limit M, the oldest sample is removed from the dictionary. This criterion was first used in the sliding-window KRLS algorithm [12] and later in the Forgetron algorithm [38] which presents a kernel version of the Perceptron algorithm on a fixed budget.

3.4.2 Optimal Brain Damage

One of the earliest pruning techniques for neural networks is proposed by Le Cun et al. in [39]. The technique, called Optimal Brain Damage (OBD), consists in removing elements from the network with small saliency, i.e., those whose removal from the network will affect the least the training error.

Saliency is formally defined as the change in the error function E, i.e., the objective function of the learning (minimization) problem. Let $\mathbf{u}$ be the parameter vector from which elements will be removed. Using the Taylor series expansion, a change $\Delta\mathbf{u}$ in the parameter

vector will lead to the following change in the objective function:

$$\Delta E = \sum_i g_i \Delta u_i + \frac{1}{2} \sum_i h_{ii} \Delta u_i^2 + \frac{1}{2} \sum_{i \neq j} h_{ij} \Delta u_i \Delta u_j + O(||\Delta \mathbf{u}||^3) \tag{3.53}$$

where $O(\cdot)$ denotes the big O notation, the Δu_i's are the components of $\Delta \mathbf{u}$, the g_i's are the components of the gradient $\mathbf{G}$ of E with respect to $\mathbf{u}$ and the h_{ij}'s are the elements of the Hessian matrix $\mathbf{H}$ of E with respect to $\mathbf{u}$, i.e.,

$$g_i = \frac{\partial E}{\partial u_i} \quad \text{and} \quad h_{ij} = \frac{\partial^2 E}{\partial u_i \partial u_j}. \tag{3.54}$$

The goal is to find a set of parameters from $\mathbf{u}$ whose removal will lead to the least increase of E. With some approximations and assumptions, particularly that the Hessian matrix $\mathbf{H}$ is assumed to be diagonal, (3.53) reduces to

$$\Delta E = \frac{1}{2} \sum_i h_{ii} \delta u_i^2 \tag{3.55}$$

with one of the approximations implying that the ΔE caused by the removal of several parameters is the sum of the ΔE's caused by the removal of each parameter individually. This leads to the following definition of the saliency of a given parameter:

$$s_i = h_{ii} u_i^2 / 2. \tag{3.56}$$

Accordingly, a predefined number of parameters with the lowest saliency is removed. For the KRLS problem, (3.56) amounts to $s_i = k_{ii} \alpha_i^2 / 2$ where $k_{ii} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_i)$ and the α_i's are the weights to be solved for (see (3.10)).

3.4.3 Optimal Brain Surgeon

In [40], Hassibi et al. consider the same problem as that in OBD [39]. However, while OBD assumes that the Hessian matrix $\mathbf{H}$ of the error function E is diagonal, which would lead OBD to sometimes prune the wrong weights, the method in [40], namely Optimal Brain Surgeon (OBS), makes no assumptions about the form of the Hessian matrix. Accordingly, without any assumptions on $\mathbf{H}$ and following a Lagrangian approach, we obtain

the following definition of the saliency of a given parameter u_i:

$$L_i = \frac{1}{2}\frac{u_i^2}{[\mathbf{H}^{-1}]_{ii}}. \tag{3.57}$$

Elements with the smallest saliencies (3.57) will be removed. For the KRLS problem, (3.57) amounts to: $L_i = \frac{1}{2}\frac{\alpha_i^2}{[\mathbf{K}^{-1}]_{ii}}$ where the α_i's are the weights to be solved for (see (3.10)) and $[\mathbf{K}^{-1}]_{ii}$ is the i-th element along the diagonal of the inverse of the Kernel matrix.

3.4.4 Minimal Introduced Error Criterion

In this section, we present the Fixed-Budget KRLS algorithm [13], denoted as FB-KRLS, which employs the minimal introduced error criterion [41] as the pruning strategy. The algorithm consists of adding a new sample point to the dictionary at each iteration, then determining the sample to be discarded when the limit M is reached, and subsequently removing that point from the dictionary. Similarly to ALD-KRLS, the goal is to find a recursive solution $\boldsymbol{\alpha}$ to a regularized version of the minimization problem (3.9) given by:

$$\arg\min_{\boldsymbol{\alpha}} \|\mathbf{y} - \mathbf{K}\boldsymbol{\alpha}\|^2 + \lambda \boldsymbol{\alpha}^T \mathbf{K}\boldsymbol{\alpha} \tag{3.58}$$

where λ is a regularization parameter that penalizes the solutions of the minimization problem $\arg\min_{\boldsymbol{\alpha}} \|\mathbf{y} - \mathbf{K}\boldsymbol{\alpha}\|^2$ that have large norms. The solution of (3.58) is given by:

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}. \tag{3.59}$$

Adding a New Sample Point to the Dictionary

When a new sample point $(\mathbf{x}_n, y_n)$ is received, it is first added to the dictionary. This corresponds to adding a new row and a new column to the kernel matrix $\mathbf{K}_{n-1}$. The result of this operation –termed matrix "upsizing"– is denoted by $\breve{\mathbf{K}}_n$ and given by:

$$\breve{\mathbf{K}}_n = \begin{bmatrix} \mathbf{K}_{n-1} & \mathbf{b} \\ \mathbf{b}^T & d \end{bmatrix} \tag{3.60}$$

where $\mathbf{K}_{n-1}$ is the matrix whose entries are given by $[\mathbf{K}_{n-1}]_{i,j} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$, $\mathbf{b} = \mathbf{k}_{n-1}$ where $\mathbf{k}_{n-1}$ is the matrix whose entries are given by $[\mathbf{k}_{n-1}]_i = k(\tilde{\mathbf{x}}_i, \mathbf{x}_n)$ and $d = k(\mathbf{x}_n, \mathbf{x}_n) + \lambda$.

Given the inverse matrix $\mathbf{K}_{n-1}^{-1}$, the inverse of the upsized matrix $\breve{\mathbf{K}}_n^{-1}$ can be obtained as follows:

$$\breve{\mathbf{K}}_n^{-1} = \begin{bmatrix} \mathbf{K}_{n-1}^{-1} + g\mathbf{e}\mathbf{e}^T & -g\mathbf{e} \\ -g\mathbf{e}^T & g \end{bmatrix} \tag{3.61}$$

where $\mathbf{e} = \mathbf{K}_{n-1}^{-1}\mathbf{b}$ and $g = (d - \mathbf{b}^T\mathbf{e})^{-1}$.

Selecting a Sample Point for Pruning

With the addition of a new sample point, the dictionary of samples may now consist of $M+1$ stored samples. In the next step, we determine the point that needs to be discarded in order to preserve only M sample points in the dictionary.

The adopted pruning technique was introduced by De Kruif and De Vries in [41] for least-squares support vector machines (LS-SVMs) with the underlying goal of minimizing the approximation error of LSSVM by discarding the sample which would introduce the smallest additional approximation error after its removal from the dataset. It is closely related to the OBS technique (3.57) which was derived for neural networks. The increase in the error when sample $\tilde{\mathbf{x}}_i$ is discarded was derived as:

$$d_i = \frac{|\boldsymbol{\alpha}_i|}{[\breve{\mathbf{K}}_n^{-1}]_{i,i}}. \tag{3.62}$$

This is easy to evaluate given that $\boldsymbol{\alpha}_n$ and $\breve{\mathbf{K}}_n^{-1}$ are quantities that would be calculated in the previous iteration. The minimal introduced error criterion (3.62) was also used in other KRLS-type algorithms (e.g., [42]), KLMS-type algorithms (e.g., [43]), kernel-based perceptron algorithms (e.g., [36]) in addition to other kernel algorithms (e.g., [35]).

Discarding the Selected Sample

In this step, the selected dictionary sample, say the i-th one, will be removed from the dictionary along with the corresponding i-th row and column in the (upsized) kernel matrix. This is done by first exchanging the first and i-th row and column of the kernel matrix, removing the new first row and column, computing the inverse of the "downsized" kernel matrix and finally, moving back the $(i-1)$-th row and column back to their initial position.

To this end, consider the following matrices:

$$\mathbf{\Pi}_i = \begin{bmatrix} 0 & \mathbf{0} & 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{i-2} & \mathbf{0} & \mathbf{0} \\ 1 & \mathbf{0} & 0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_{M-i+1} \end{bmatrix}, \quad \mathbf{\Psi}_i = \begin{bmatrix} \mathbf{0} & \mathbf{I}_{i-1} & \mathbf{0} \\ 1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{M-i} \end{bmatrix} \tag{3.63}$$

where $\mathbf{I}_j$ is the unit matrix of size j and $\mathbf{0}$ is the all-zeros matrix of appropriate dimensions. In the first step, pre- and post-multiplying by $\mathbf{\Pi}_i$ corresponds to an exchange of the first and i-th row and column. Now, we introduce the matrix operations for removing the first row and column of the upsized kernel matrix $\breve{\mathbf{K}}_n$ which results in the downsized matrix $\mathbf{K}_n$ whose inverse can be computed with the knowledge of $\breve{\mathbf{K}}_n^{-1}$ as follows:

$$\breve{\mathbf{K}}_n = \begin{bmatrix} q & \mathbf{u}^T \\ \mathbf{u} & \mathbf{K}_n \end{bmatrix}, \quad \breve{\mathbf{K}}_n^{-1} = \begin{bmatrix} r & \mathbf{v}^T \\ \mathbf{v} & \mathbf{G} \end{bmatrix} \Rightarrow \mathbf{K}_n^{-1} = \mathbf{G} - \mathbf{v}\mathbf{v}^T/r \tag{3.64}$$

where the are no underlying assumptions regarding the vectors $\mathbf{u}$ and $\mathbf{v}$, the scalar values q and r and the matrix $\mathbf{G}$. In the last step, pre- and post-multiplying by $\mathbf{\Psi}_i$ corresponds to moving the $(i-1)$-th row and column back to the first position. Algorithm (2) presents the steps to compute the inverse of the downsized matrix.

Algorithm 2 Procedure to obtain the inverse of a matrix $\breve{\mathbf{K}}_n$ whose i-th row and column are removed

Compute $\breve{\mathbf{K}}_n^i = \mathbf{\Pi}_i \breve{\mathbf{K}}_n \mathbf{\Pi}_i$ and $(\breve{\mathbf{K}}_n^i)^{-1} = \mathbf{\Pi}_i \breve{\mathbf{K}}_n^{-1} \mathbf{\Pi}_i$.
Remove the first row and column of $\breve{\mathbf{K}}_n^i$ to obtain $\mathbf{K}_n^i$.
Calculate the inverse $(\mathbf{K}_n^i)^{-1}$ using (3.64).
Obtain $\mathbf{K}_n^{-1} = \mathbf{\Psi}_i (\mathbf{K}_n^i)^{-1} \mathbf{\Psi}_i$.

Memory Update for Tracking Time-Varying Mappings

In what preceded, it was assumed that the goal is to model a static non-linear mapping that allows predicting labels for incoming inputs. However, if the non-linear mapping changes over time, it is likely that the stored samples do not reflect well the current mapping. To

achieve tracking capability, with each incoming sample $(\mathbf{x}_n, y_n)$, the update

$$y_i \leftarrow y_i - \mu(y_i - y_n)k(\tilde{\mathbf{x}}_i, \mathbf{x}_n), \quad \forall i, \tag{3.65}$$

where $\mu \in [0, 1]$ is a step-size parameter, is proposed for all labels y_i. The update rule in (3.65) considers the similarities in both the inputs and outputs, measured respectively by the kernel and the difference $y_i - y_n$. The update rule affects mostly pairs whose input vector $\tilde{\mathbf{x}}_i$ is similar to $\mathbf{x}_n$, that is, if $\tilde{\mathbf{x}}_i$ is similar to $\mathbf{x}_n$, the value of $k(\tilde{\mathbf{x}}_i, \mathbf{x}_n)$ will be large and so the update on the output will be more significant. At the same time, the update will also be proportional to $y_i - y_n$. Hence, the update rule balances between the similarities (or lack thereof) in the input space and those in the output space. FB-KRLS is summarized in Algorithm 3.

Algorithm 3 Fixed-Budget KRLS

initialize: Memory $= \{(\tilde{\mathbf{x}}_1, y_1)\}$.
Calculate $\mathbf{K}_1^{-1}$ and $\boldsymbol{\alpha}$ with (3.59).
for $n = 2, 3, \ldots$ **do**
  Update all stored labels y_i with (3.65).
  Add $(\mathbf{x}_n, y_n)$ to memory and obtain $\breve{\mathbf{K}}_n^{-1}$ with (3.61).
  if memory size $> M$ **then**
    Determine least significant point $(\tilde{\mathbf{x}}_L, y_L)$ using (3.62).
    Prune $(\tilde{\mathbf{x}}_L, y_L)$ from memory and obtain $\mathbf{K}_n^{-1}$ (Algorithm 2).
  end if
  Get KRLS solution based on updated memory using (3.59).
end for

3.4.5 Minimal Dependence Strategy

In [37], Nguyen-Tuong et al. propose a pruning strategy based on the dependency of a given sample on other samples in the dictionary. Recall the ALD criterion (3.12) from Section 3.3.1:

$$\delta_n \triangleq \min_{\boldsymbol{a}} \left\| \sum_{i=1}^{m_{n-1}} a_i \phi(\tilde{\mathbf{x}}_i) - \phi(\mathbf{x}_n) \right\|^2 \tag{3.66}$$

which we also expressed as:

$$\delta_n = k_{nn} - \tilde{\mathbf{k}}_{n-1}^T \boldsymbol{a}_n. \tag{3.67}$$

The pruning strategy consists in selecting for removal the element with the largest dependence on other samples, i.e., we want to remove samples that are more dependent on other dictionary samples. To that end, Nguyen-Tuong et al. consider the following adaptation of (3.67):

$$\delta_i = k_{ii} - \tilde{\mathbf{k}}_{n-1}^T \boldsymbol{a}_n. \tag{3.68}$$

Accordingly, when a new sample $\{\mathbf{x}_n, y_n\}$ is received, $\tilde{\mathbf{k}}_{n-1}^T$ and $\boldsymbol{a}_n$ are computed as in Section 3.3.1; however, the linear independence measure δ_i is computed for each element in the dictionary using (3.68). Moreover, in the proposed pruning strategy, temporal considerations are also taken in account by imposing a time-dependent forgetting factor $\lambda_i \in [0, 1]$. Consequently, the pruning strategy considers the independence value δ_i weighted by the forgetting factor λ_i as a criterion for removal. Thus, the proposed strategy will lead to the removal of the dictionary sample with the smallest $\lambda_i \delta_i$. The forgetting factor λ_i is defined to be:

$$\lambda_i = e^{-\frac{(n-i)^2}{2h}} \tag{3.69}$$

where n is the current iteration, i is the iteration at which that specific point (being considered for pruning) was introduced to the dictionary and h is a parameter representing the intensity of the forgetting factor.

3.4.6 Soft Pruning for Kernel-based Anomaly Detection

In [44], Ahmed et al. use a form of soft pruning to remove elements from the dictionary. While the technique does not impose a hard limit on the dictionary size, it aims at limiting the dictionary size by removing elements that become "obsolete" after a certain number of iterations. More specifically, at iteration n, the usefulness of each dictionary element over the previous L measurements is evaluated by checking the following criterion:

$$\zeta_i = \sum_{j=n-L+1}^{n} \mathbb{1}\{k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) > d\}, \;\; i = n - L + 1, \ldots, n, \tag{3.70}$$

where $\mathbb{1}\{\cdot\}$ denotes the indicator function and d is a threshold parameter. If criterion (3.70) evaluates to 0 for a vector $\tilde{\mathbf{x}}_i$, then $\tilde{\mathbf{x}}_i$ is deemed obsolete and is subsequently removed from

the dictionary.

Table 3.2 outlines the different pruning strategies presented in Section 3.4.

Table 3.2 Different Pruning Strategies

n	Current iteration
$\tilde{\mathbf{x}}_i$	Input sample stored at index i in the dictionary
$\mathbf{x}_n$	Incoming input sample at iteration n
α_i	Weight corresponding to input sample $\tilde{\mathbf{x}}_i$
$\mathbf{K}_n^{-1}$	Inverse of kernel matrix at iteration n

Strategy	Expression	Adopted in
Optimal Brain Damage	$s_i = k_{ii}\alpha_i^2/2$ where $k_{ii} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_i)$	[39]
Optimal Brain Surgeon	$L_i = \frac{1}{2}\frac{\alpha_i^2}{[\mathbf{K}_n^{-1}]_{i,i}}$	[40]
Minimal Introduced Error	$d_i = \frac{\lvert\alpha_i\rvert}{[\mathbf{K}_n^{-1}]_{i,i}}$	[41], [13], [42], [43], [36], [35]
Remove-the-Oldest	–	[12], [38]
Minimal Dependence Strategy	$\lambda_i\delta_i$ where $\lambda_i = \exp(-\frac{(n-i)^2}{2h})$ h: intensity of forgetting factor λ $\delta_i = k_{ii} - \mathbf{k}_{n-1}^T\mathbf{K}_{n-1}^{-1}\mathbf{k}_{n-1}$ with $[\mathbf{k}_{n-1}]_i = k(\tilde{\mathbf{x}}_i, \mathbf{x}_n)$	[37]
Soft Pruning for Kernel-based Anomaly Detection	$\zeta = \sum_{j=n-L+1}^{n} \mathbb{1}\{k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) > d\},\ i = n-L+1, \ldots, n$ where $\mathbb{1}\{\cdot\}$: the indicator function d: threshold parameter	[44]

3.5 Conclusion

In this chapter, we first presented an overview of the classical Recursive Least Squares (RLS) algorithm. Engel et al. presented in [9] an online kernel version of the RLS algorithm, namely Kernel RLS (KRLS), which is able to recursively solve non-linear regression problems. KRLS is sequentially presented with input-output pairs and iteratively calculates a linear LS regressor in the high-dimensional feature space induced by the employed

kernel. As is typical with kernel-based regression methods, the number of parameters that need to be calculated to obtain the LS solution is equal to the number of input vectors that grows without bound as the iterations progress. To address this issue, a "sparsification" procedure is proposed in [9] to be used in conjunction with KRLS, that forms the LS regressor using a carefully chosen subset, termed dictionary, of the input vectors. Sparsification techniques have been increasingly used in conjunction with kernel adaptive filtering algorithms as they are quite useful in curbing the growth of the dictionary and the Gram matrix. However, often, a more aggressive approach is needed, particularly when practical considerations are taken into account, e.g., algorithm implementations on DSP processors with limited memory size and computational resources. Fixed-budget algorithms provide a solution to this problem by imposing a hard limit on the number of samples that can be stored in the dictionary. As a result, reaching that limit necessitates the removal, known as pruning, of a sample from the dictionary. While sparsification and pruning techniques are useful in curbing the growth of the dictionary, which is crucial for equipping an adaptive filtering algorithm with the tracking ability, they are not, on their own, sufficient for the algorithm to have a good tracking ability.

Chapter 4

Proposed Method

4.1 Background

In this section, we first present the Subspace Pursuit (SP) algorithm [14] which was introduced in the context of compressive sensing as a method to reconstruct unknown sparse signals. Then, we extend SP to non-linear regression problems in the spirit of two kernel methods, Kernel Matching Pursuit (KMP) [45] and Kernel Basis Pursuit (KBP) [46] which we present subsequently in the section. KMP and KBP are of particular interest to our work as they attempt to solve a similar problem to ours, that of learning a regression function by means of sparse approximation.

4.1.1 Subspace Pursuit

Subspace Pursuit (SP) was first introduced by Dai et al. in [14] in the context of compressive sensing (see also [47] for a similar method). It was originally proposed as an iterative method for the reconstruction of an unknown signal $\mathbf{u} = [u_1, \ldots, u_N]$ from a possibly noisy observation, $\mathbf{v} \in \mathbb{R}^{N'}$, of $\mathbf{u}$ via $N' \ll N$ linear measurements, that is,

$$\mathbf{v} = \Phi\mathbf{u} \tag{4.1}$$

where $\Phi \in \mathbb{R}^{N' \times N}$ is referred to as the sampling matrix and $\mathbf{v}$ as the measurement vector. The unknown signal $\mathbf{u}$, having $K \ll N$ non-zero elements, is said to be K-sparse.

SP as an ℓ_1-minimization technique

In compressive sensing, we are mainly interested in sparse signals as sparsity encapsulates the idea that the signal contains far less information than what its actual dimension suggests. Thus, in reconstructing signals in the compressive sensing context, SP seeks to find the sparsest vector $\mathbf{u}$ that satisfies (4.1). This amounts to solving the following ℓ_0-minimization problem

$$\min ||\mathbf{u}||_0 \text{ subject to } \mathbf{v} = \Phi\mathbf{u} \tag{4.2}$$

as the ℓ_0-norm of a vector is the number of non-zero elements in that vector, which is the sparsity of the vector. However, as solving the minimization problem (4.2) is NP-hard [14], it is relaxed to the following ℓ_1-minimization problem that can be solved efficiently by linear programming techniques:

$$\min ||\mathbf{u}||_1 \text{ subject to } \mathbf{v} = \Phi\mathbf{u} \tag{4.3}$$

where $||\mathbf{u}||_1 = \sum_{i=1}^{N} |u_i|$ denotes the ℓ_1-norm of the vector $\mathbf{u}$.

SP from a least-squares perspective

In practice, the measurement vector $\mathbf{v}$ contains noise. To handle the case of noisy linear measurements, the minimization problem (4.3) is written in the form of a least-squares constrained minimization problem as follows:

$$\min_{\mathbf{u}} \|\mathbf{\Phi u} - \mathbf{v}\|^2 \text{ s.t. } \mathbf{u} \text{ is } K\text{-sparse.} \tag{4.4}$$

Preliminaries

Before discussing the details of the SP algorithm, we first need to define the notions of projection and residue. Let $I \subset \{1, \ldots, N\}$. The matrix Φ_I consists the columns of Φ with indices $i \in I$. The space spanned by the columns of Φ_I is denoted by span(Φ_I). Suppose Φ_I is full-rank, i.e., $\Phi_I^T\Phi_I$ is invertible.

Definition 10. (Projection) *The projection of* $\mathbf{v}$ *onto span*(Φ_I) *is defined as:*

$$\mathbf{v}_p = \text{proj}(\mathbf{v}, \Phi_I) := \Phi_I\Phi_I^{\dagger}\mathbf{v} \tag{4.5}$$

where $\Phi_I^{\dagger}$ *denotes the Moore-Penrose pseudo-inverse of* Φ_I.

Definition 11. (Residue) *The residue vector of the projection is defined as*

$$\mathbf{r} = \text{resid}(\mathbf{v}, \Phi_I) := \mathbf{v} - \mathbf{v}_p. \tag{4.6}$$

Summary of SP algorithm

To reconstruct the unknown K-sparse signal $\mathbf{u}$, SP identifies the set of K columns of the sampling matrix Φ that approximate best the signal $\mathbf{u}$. This set of K columns of Φ is referred to as the support set.

SP starts by computing the correlation vector between the sampling matrix Φ and the measurement vector $\mathbf{v}$. The support set T is initialized to be T_0 given by

$$T_0 = \{\text{Indices of the } K \text{ largest magnitude entries in the correlation vector } \Phi^T\mathbf{v}\}, \tag{4.7}$$

with the resulting residue being initialized to

$$\mathbf{r}_0 = \text{resid}(\mathbf{v}, \Phi_{T_0}) = \mathbf{v} - \text{proj}(\mathbf{v}, \Phi_{T_0}). \tag{4.8}$$

The support set T is then iteratively refined by discarding and adding columns according to the value of their correlation with the residue. Specifically, at iteration ℓ, we compute the correlation vector between the sampling matrix Φ and the residue of the previous iteration, and then identify the vector's K largest magnitude entries. The set of newly identified indices is then merged with the support set of the previous iteration, i.e.,

$$\tilde{T}_\ell = T_{\ell-1} \cup \{\text{Indices of the } K \text{ largest magnitude entries in the vector } \mathbf{\Phi}^T\mathbf{r}_{\ell-1}\}. \tag{4.9}$$

Using the merged set (4.9), we project the measurement vector $\mathbf{v}$ onto $\text{span}(\Phi_{\tilde{T}_\ell})$. The resulting projection coefficients are then used to produce a new support set

$$T_\ell = \{\text{Indices of the } K \text{ largest magnitude entries in the vector } \mathbf{\Phi}^\dagger_{\tilde{T}_\ell}\mathbf{v}\}. \tag{4.10}$$

Finally, we compute the new residue

$$\mathbf{r}_\ell = \text{resid}(\mathbf{v}, \Phi_{T_\ell}) = \mathbf{v} - \text{proj}(\mathbf{v}, \Phi_{T_\ell}). \tag{4.11}$$

This procedure is repeated until the stopping condition

$$||\mathbf{r}_\ell||_2 > ||\mathbf{r}_{\ell-1}||_2 \tag{4.12}$$

is met, in which case we quit the iterative process and set T_ℓ=$T_{\ell-1}$.

4.1.2 Kernel Matching Pursuit

Vincent et al. presented in [45] a method, the Kernel Matching Pursuit (KMP), that constructs a regression function as a linear combination of functions selected from a kernel-based dictionary. Particularly, we are given L noisy observations $\{y_1, \ldots, y_L\}$ of a target function $f \in \mathcal{H}$ at the input vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_L\}$ where $\mathbf{x}_i \in \mathbb{R}^d$ (for a given d) and $y_i \in \mathbb{R}, \forall i$. We are also given a finite dictionary centered on the input vectors, that is,

$$\mathcal{G} = \{g_i = k(\cdot, \mathbf{x}_i) | i = 1, \ldots, L\} \tag{4.13}$$

of L functions in a Hilbert space $\mathcal{H}$ where k is a kernel. The goal is to approximate f using a finite number B of functions, called basis functions, selected from $\mathcal{G}$. Given that B is finite, we talk of a sparse approximation of f. In other words, we are interested in constructing a sparse approximation, also referred to as expansion, f_B of f as a linear combination of functions $g_{\gamma_i}, i = 1, \ldots, B$ selected from $\mathcal{G}$:

$$f_B(\mathbf{x}) = \sum_{i=1}^{B} \alpha_i g_{\gamma_i}(\mathbf{x}) = \sum_{i=1}^{B} \alpha_i k(\mathbf{x}, \mathbf{x}_{\gamma_i}) \tag{4.14}$$

where B is the number of basis functions in the approximation, $\{\gamma_1, \ldots, \gamma_B\}$ are the indices of the B functions selected from $\mathcal{G}$ and $\{\alpha_1, \ldots, \alpha_B\}$ is the set of coefficients corresponding to those functions.

The approximation vector $\hat{\mathbf{y}}_B$ is the vector consisting of the evaluation of f_B on the L input vectors, i.e., $\hat{\mathbf{y}}_B = [f_B(\mathbf{x}_1), \ldots, f_B(\mathbf{x}_L)]^T$. We define the residue as the approximation error between the target vector $\mathbf{y} = [y_1, \ldots, y_L]^T$ and the approximation vector $\hat{\mathbf{y}}_B$, i.e.,

$$\mathbf{r}_B = \mathbf{y} - \hat{\mathbf{y}}_B. \tag{4.15}$$

Theoretically, the B selected functions $\{g_{\gamma_1}, \ldots, g_{\gamma_B}\}$ and their corresponding coefficients

$\{\alpha_1, \ldots, \alpha_B\}$ should be chosen such that they minimize the squared norm of the residue

$$||\mathbf{r}_B||^2 = ||\mathbf{y} - \hat{\mathbf{y}}_B||^2. \tag{4.16}$$

However, finding the optimal set of functions $\{g_{\gamma_1}, \ldots, g_{\gamma_B}\}$ requires an exhaustive search over all $\frac{K!}{B!(K-B)!}$ possible choices of B basis functions among the K dictionary functions. As carrying out such search would be computationally prohibitive, Vincent et al. propose a suboptimal, greedy, iterative approach in which, at each iteration, the optimal function is first selected, then the corresponding weight is computed based on the selected function. Particularly, KMP starts with the expansion being initialized to zero, and then builds it by adding to it, at each iteration, one new function from the dictionary $\mathcal{G}$.

At iteration $n < B$, the approximation of f is not fully constructed yet, and it is given by:

$$f_n(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_{\gamma_i}). \tag{4.17}$$

The corresponding approximation vector is $\hat{\mathbf{y}}_n = [f_n(\mathbf{x}_1), \ldots, f_n(\mathbf{x}_L)]^T$. The coefficient α_n that minimizes the residue at iteration n, that is,

$$||\mathbf{r}_n||^2 = ||\mathbf{y} - \hat{\mathbf{y}}_n||^2 = ||\mathbf{y} - (\hat{\mathbf{y}}_{n-1} + \alpha_n \mathbf{g}_n)||^2 = ||\mathbf{r}_{n-1} - \alpha_n \mathbf{g}_n||^2 \tag{4.18}$$

where $\mathbf{g}_n = [g_{\gamma_n}(\mathbf{x}_1), \ldots, g_{\gamma_n}(\mathbf{x}_L)]^T$, is given by:

$$\alpha_n = \frac{\langle \mathbf{g}_n, \mathbf{r}_{n-1} \rangle}{||\mathbf{g}_n||^2}. \tag{4.19}$$

Plugging (4.19) in (4.18), we get:

$$||\mathbf{r}_n||^2 = ||\mathbf{r}_{n-1}||^2 - \left(\frac{\langle \mathbf{g}_n, \mathbf{r}_{n-1} \rangle}{||\mathbf{g}_n||} \right)^2 \tag{4.20}$$

So the function $g_{\gamma_n} \in \mathcal{G}$ that minimizes (4.18) is the one that maximizes $\left| \frac{\langle \mathbf{g}_n, \mathbf{r}_{n-1} \rangle}{||\mathbf{g}_n||} \right|$. Accordingly, at iteration n, KMP selects from $\mathcal{G}$ the function that maximizes $\left| \frac{\langle \mathbf{g}_n, \mathbf{r}_{n-1} \rangle}{||\mathbf{g}_n||} \right|$ and then computes the corresponding coefficient using (4.19). KMP is summarized in Algorithm 4.

Algorithm 4 Kernel Matching Pursuit Algorithm

Input:

Dataset $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_L, y_L)\}$

B: Number of basis functions desired in the expansion

Dictionary $\mathcal{G}$ of functions with $\mathcal{G}_{ij} = g_j(\mathbf{x}_i) = k(\mathbf{x}_i, \mathbf{x}_j), i = 1, \ldots, L, j = 1, \ldots, K$ with $\mathbf{g}_j = [g_j(\mathbf{x}_1), \ldots, g_j(\mathbf{x}_L)]^T$.

Initialization:

$\mathbf{r} = [y_1, \ldots, y_L]^T$

For $n = 1, \ldots, B$

$$\gamma_n = \arg \max_{k=1,\ldots K} \left| \frac{\langle \mathbf{g}_k, \mathbf{r} \rangle}{||\mathbf{g}_k||} \right|$$

$$\alpha_n = \frac{\langle \mathbf{g}_{\gamma_n}, \mathbf{r} \rangle}{||\mathbf{g}_{\gamma_n}||^2}$$

$$\mathbf{r} = \mathbf{r} - \alpha_n \mathbf{g}_{\gamma_n}$$

Output:

$$f_B(\mathbf{x}) = \sum_{i=1}^{B} \alpha_i g_{\gamma_i}(\mathbf{x}) = \sum_{i=1}^{B} \alpha_i k(\mathbf{x}, \mathbf{x}_{\gamma_i})$$

4.1.3 Kernel Basis Pursuit

Consider again the setup from Section 4.1.2. Kernel Basis Pursuit (KBP) [46] addresses the same problem as KMP [45], that is, building a sparse approximation of the target function f using a finite number of functions selected from a kernel-based dictionary. However, there are several key differences between the two approaches.

While KMP attempts to minimize the mean squared error (4.16) with respect to both the basis function and the corresponding coefficient, KBP solves a regularized version of the problem with respect to the coefficient vector $\boldsymbol{\alpha}$ only:

$$\min_{\boldsymbol{\alpha}} ||\mathbf{y} - \mathbf{G}\boldsymbol{\alpha}||^2 \quad \text{s.t. } ||\boldsymbol{\alpha}||_1 \leq K \tag{4.21}$$

where K is a regularization parameter and $\mathbf{G}$ is the Gram matrix. Essentially, both KBP and KMP attempt to solve the same problem but they address the question of the sparsity of the solution in different ways. Indeed, KMP guarantees that the solution is K-sparse by constructing it using a finite number K of basis functions. On the other hand, KBP uses a regularization term (in the minimization problem) that imposes the sparsity of the

solution.

Problem (4.21) corresponds to the well-known Least Absolute Shrinkage and Selection Operator (LASSO) formulation [48] (in the feature space) which combines an ℓ_2-loss function (squared error) and ℓ_1-regularization. The original Basis Pursuit (BP) algorithm [49] finds the optimal solution to the minimization problem as a linear combination of all the dictionary functions using costly and complex linear programming techniques [46]. Instead, KBP uses the Least Angle Regression (LARS) technique [50] which also finds the exact solution of the LASSO but in an iterative and efficient way.

4.2 Motivation

In designing an adaptive filtering algorithm for tracking purposes, the choice of the weight vector length should be carefully considered. In principle, the weight vector length should be chosen to be as short as possible to improve the algorithm's tracking ability, but long enough to adequately model the system.

In all KRLS algorithms to this date, the weight vector $\boldsymbol{\alpha}$ that is the solution of the minimization problem (3.9) is of equal size to that of the dictionary. This coupling of the weight vector length to the dictionary size introduces a trade-off. While a large dictionary is favorable as it would represent all the dynamics of the input-output relationship over time, it has a detrimental effect on the algorithm's ability to track changes in that relationship; it is well known [6] that having to adjust a large number of weights significantly slows down adaptation. This trade-off highlights the need to decouple the size of the weight vector from the dictionary size and motivates the proposed method presented in this thesis. We illustrate this by the following experiment.

In this experiment, we investigate the effect of changing the size of the weight vector on the tracking performance of the Fixed-Budget KRLS (FB-KRLS) algorithm [13] and the Sliding-Window KRLS (SW-KRLS) algorithm [12]. Particularly, two versions of each of FB-KRLS and SW-KRLS are run along 3000 iterations, one with a budget of $M = 2000$, the other with a budget of $M = 200$.

We consider a system composed of a time-varying linear filter followed by a static nonlinearity. The input signal, whose elements x_i are drawn i.i.d. from a normal distribution with mean 0 and variance 0.5, is passed through a linear filter that varies in time as follows:

During the first 1500 iterations, its impulse response is given by:

$$h_1(n) = \delta(n) - 0.37\delta(n-1) - 0.48\delta(n-2) + 0.81\delta(n-3),$$

where $\delta(n)$ is the unit impulse (also known as Kronecker delta). On iteration 1501, the filter is abruptly changed. Its impulse response becomes

$$h_2(n) = \delta(n) - 0.83\delta(n-1) + 0.67\delta(n-2) + 0.72\delta(n-3)$$

which remains constant for the next 1500 iterations. The output of the linear filter is then passed through the static non-linear function

$$f(x) = \tanh(x). \tag{4.22}$$

The resulting signal is finally corrupted with 20 dB[1] of white Gaussian noise. We use 400 sample points as a test set (different in each phase of the scenario according to the linear filter) and a time embedding of 4, i.e., $\mathbf{x}_n = [x_n, x_{n-1}, x_{n-2}, x_{n-3}]^T$. For FB-KRLS, the step-size parameter is set to $\mu = 0.01$, and the regularization parameter is set to $\lambda = 0.001$. For both algorithms, a Gaussian kernel is used with a width $\sigma = 0.8$. The values of the parameters were chosen via 5-fold cross-validation.

The results, averaged over 15 Monte-Carlo simulations, are shown in Figure 4.1. Each version of the two algorithms with a budget of $M = 200$ outperforms the version with $M = 2000$ in tracking the system changes. Indeed, following changes in the filter (depicted by the vertical blue line), the MSE curve of each of the algorithms with $M = 200$ is faster to change and converge, thus capturing faster the change in the system. As in FB-KRLS and SW-KRLS the dictionary size is equal to the weight vector size, we can conclude that a large dictionary size has indeed a detrimental effect on the tracking ability. We note however that the algorithms with a smaller budget do converge to a worse steady-state value. This is expected since, as we discussed earlier, a larger dictionary represents better the dynamics of the input-output relationship over time, and as such, it is expected that an algorithm with a larger dictionary would converge to a smaller steady-state error.

[1] This value represents the SNR equal to $10\log_{10}(\frac{\text{Signal Power}}{\text{Noise Power}})$ where the signal power is assumed to be equal to 1.

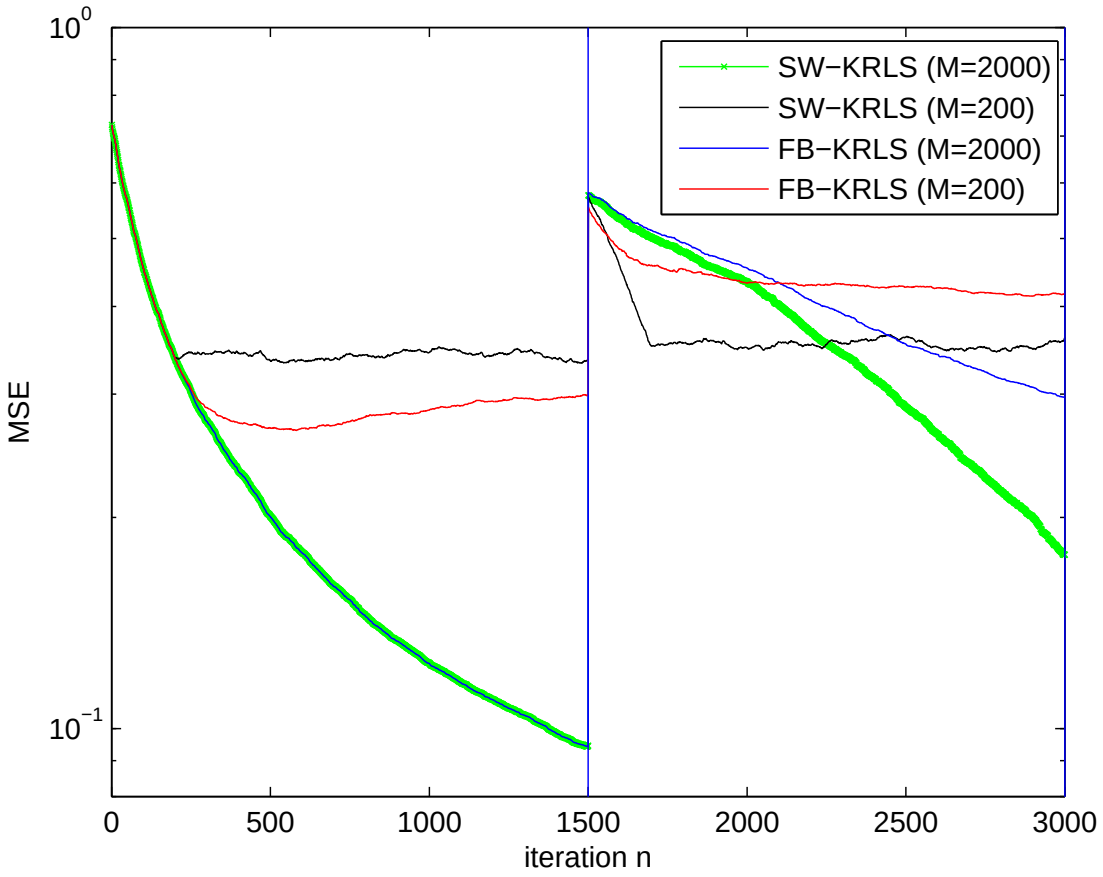

Fig. 4.1 Effect of changing the dictionary size/weight vector size on the tracking performance of SW-KRLS and FB-KRLS on a time-varying system.

4.3 Kernel Subspace Pursuit

In this section, we extend SP to non-linear regression problems and introduce Kernel SP (KSP) in the spirit of KMP [45] and KBP [46] which were presented in Section 4.1. In Section 4.4, we will use KSP in conjunction with KRLS to improve the latter's performance in tracking time-varying systems.

4.3.1 Problem Formulation

Consider again the setup introduced in Section 3.1. Let $\mathcal{D}_n = [\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_{m_n}]$ be the dictionary containing a subset of input vectors up to time n. As in KRLS, we are interested in

obtaining a minimizer of (3.7) of the form

$$f(\mathbf{x}) = \sum_{i=1}^{m_n} \alpha_i k(\mathbf{x}, \tilde{\mathbf{x}}_i) \tag{4.23}$$

but we now add the constraint that the vector of coefficients $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_{m_n}]^T$ is K-sparse, i.e., it contains exactly K non-zero coefficients.

Let $\mathbf{G}_n$ be the $m_n \times n$ Gram matrix obtained by evaluating the functions $k(\cdot, \tilde{\mathbf{x}}_i)$, $i = 1, \ldots, m_n$ at the input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$, i.e.,

$$[\mathbf{G}_n]_{i,j} = k(\mathbf{x}_i, \tilde{\mathbf{x}}_j). \tag{4.24}$$

Using $\mathbf{G}_n$, we can formulate the constrained minimization of (3.7) considered here as:

$$\min_{\boldsymbol{\alpha}} \|\mathbf{G}_n\boldsymbol{\alpha} - \mathbf{y}_n\|^2 \text{ s.t. } \boldsymbol{\alpha} \text{ is } K - \text{sparse}. \tag{4.25}$$

To simplify presentation, for the rest of this section we will drop the subscript n of $\mathbf{G}_n$. Denote by $\mathbf{G}_T$, with T being a set of column indices, the matrix consisting of the columns of $\mathbf{G}$ with indices in T. Then (4.25) can be rewritten as

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^K, T} \|\mathbf{G}_T\boldsymbol{\alpha} - \mathbf{y}_n\|^2 \text{ s.t. cardinality of } T \text{ is } K. \tag{4.26}$$

4.3.2 The KSP Procedure

KSP solves (4.26) by, initially selecting the K columns of $\mathbf{G}$ that exhibit the highest correlation with the vector $\mathbf{y}_n$, and then iteratively refining this set by discarding and adding columns. The refinement process is carried out by retaining "informative" columns of $\mathbf{G}$, i.e., the columns of $\mathbf{G}$ that better approximate $\mathbf{y}_n$, and discarding the less informative ones.

Particularly, KSP starts by computing the correlation vector between the matrix $\mathbf{G}$ and the vector $\mathbf{y}_n$, then identifies the K elements of the correlation vector with the largest magnitudes. An initial estimate, T_0, of the support set is thus obtained:

$$T_0 = \{\text{Indices of the } K \text{ largest magnitude entries in the correlation vector } \mathbf{G}\mathbf{y}_n\}. \tag{4.27}$$

Given T_0, we obtain an initial estimate, $\hat{\mathbf{y}}_n$, of $\mathbf{y}_n$ by projecting the vector $\mathbf{y}_n$ onto the span of the columns of $\mathbf{G}_{T_0}$:

$$\hat{\mathbf{y}}_n = \text{proj}(\mathbf{y}_n, \mathbf{G}_{T_0}).^2 \tag{4.28}$$

The approximation error of $\mathbf{y}_n$, also referred to as residue, is then computed:

$$\mathbf{r}_0 = \mathbf{y}_n - \hat{\mathbf{y}}_0. \tag{4.29}$$

At a given iteration[3] ℓ, we begin by computing the correlation vector $\mathbf{G}^T\mathbf{r}_{\ell-1}$ between $\mathbf{G}$ and the approximation error of $\mathbf{y}_n$ from the previous iteration, and then identify the columns of $\mathbf{G}$ that correspond to that vector's K elements with the largest magnitudes. The new set of K columns is then merged with the set $T_{\ell-1}$ of columns of the previous iteration's approximation:

$$\tilde{T}_\ell = T_{\ell-1} \cup \{\text{Indices of the } K \text{ largest magnitude entries in the vector } \mathbf{G}^T\mathbf{r}_{\ell-1}\}. \tag{4.30}$$

We then project $\mathbf{y}_n$ onto the span of the columns of $\mathbf{G}$ in the merged set:

$$\text{proj}(\mathbf{y}_n, \mathbf{G}_{\tilde{T}_\ell}) = \mathbf{G}_{\tilde{T}_\ell}\mathbf{G}^\dagger_{\tilde{T}_\ell}\mathbf{y}_n \tag{4.31}$$

and use the projection coefficients to obtain a new set of indiced, T_ℓ, by identifying the columns corresponding to those K coefficients having the largest magnitude:

$$T_\ell = \{\text{Indices of the } K \text{ largest magnitude entries in the vector } \mathbf{G}^\dagger_{\tilde{T}_\ell}\mathbf{y}_n\}. \tag{4.32}$$

Finally, a new LS estimate of $\mathbf{y}_n$, $\hat{\mathbf{y}}_n$, is acquired by projecting the vector $\mathbf{y}_n$ onto the span of the columns of T_ℓ:

$$\hat{\mathbf{y}}_n = \text{proj}(\mathbf{y}_n, \mathbf{G}_{\tilde{T}}) = \mathbf{G}_{T_\ell}\mathbf{G}^\dagger_{T_\ell}\mathbf{y}_n \tag{4.33}$$

and a new residue $\mathbf{r}_\ell = \mathbf{y}_n - \hat{\mathbf{y}}_n$ is obtained. This procedure is repeated until the maximum

[2] See (4.5) for a definition of projection.

[3] We note that iteration ℓ should not be confused with iteration n. Indeed, as mentioned in the setup introduced in Section 3.1, training pairs are sequentially received. At iteration n, after having received the n-th training pair $(\mathbf{x}_n, y_n)$, we run up to $\ell_{\max}$ iterations of KSP, indexed by ℓ.

number $\ell_{\max}$ of iterations is reached or until the stopping condition

$$||\mathbf{r}_\ell||_2 > ||\mathbf{r}_{\ell-1}||_2 \tag{4.34}$$

is met, in which case we quit the iterative process and set T_ℓ=$T_{\ell-1}$. Algorithm 5 summarizes KSP.

Algorithm 5 Kernel Subspace Pursuit Algorithm

Input: K, $\mathbf{y}_n$, $\mathcal{D}$.

Initialization:

Compute $\mathbf{G}$ using (4.24).

$T_0 = \{\text{Indices of the } K \text{ largest magnitude entries in the vector } \mathbf{Gy}\}$

$\mathbf{r}_0 = \mathbf{y}_n - \text{proj}(\mathbf{y}_n, \mathbf{G}_{T_0})$

Loop: At iteration ℓ $(1 \leq \ell \leq \ell_{\max})$, execute the following:

$\tilde{T}_\ell = T_{\ell-1} \cup \{\text{Indices of the } K \text{ largest magnitude entries in the vector } \mathbf{G}^T\mathbf{r}_{\ell-1}\}$

$T_\ell = \{\text{Indices of the } K \text{ largest magnitude entries in } \mathbf{G}^\dagger_{\tilde{T}_\ell}\mathbf{y}_n\}$

$\mathbf{r}_\ell = \mathbf{y}_n - \text{proj}(\mathbf{y}_n, \mathbf{G}_{T_\ell})$

If $||\mathbf{r}_\ell||_2 > ||\mathbf{r}_{\ell-1}||_2$, let $T_\ell = T_{\ell-1}$ and exit the loop.

Output: T_ℓ

4.3.3 KSP versus KMP and KBP

While KSP, KMP and KBP are all methods that learn a regression function by means of sparse approximation, there exists a substantial difference in the reconstruction method that each employs. Indeed, both KMP and KBP start by intializing the regression function to zero, then iteratively build it by selecting, at each iteration, one new entry from the dictionary of functions. To build the regression function, KMP needs a number of iterations equal to the desired number of basis functions in the expansion, and KBP needs a number of iterations equal to the number of dictionary functions.[4] KSP, on the other hand, always maintains an estimate of the regression function built using a pre-specified number of basis functions, and refines the estimate through a usually limited number of iterations. Thus, KSP would normally require a smaller number of iterations than that required by KMP

[4]Using the LARS implementation, KBP will require a number of iterations equal to the desired number of basis functions in the expansion only.

and KBP.

In Chapter 5, we present simulation results that show that KSP outperforms both KMP and KBP in learning non-linear functions. We also provide a comparison of the computational burden of each of these algorithms. This comparison will show that KSP is indeed less computationally expensive than both KMP and KBP.

4.4 Proposed Algorithm

In this section, we present a new KRLS algorithm that is able to efficiently track time-varying systems. In our proposed method, we are particularly interested in learning a regression function that approximates the most recent $\mathcal{N}$ target values in order to track the changes of a time-varying system. As such, we consider constructing a K-sparse approximation of the target function (representing the input-output relationship of the most recent $\mathcal{N}$ training pairs) using the K dictionary elements that track best those $\mathcal{N}$ target values. To that end, we propose the use of KSP in conjunction with KRLS. KSP selects the K dictionary elements that will be used to form the LS regressor. The proposed algorithm, called Subspace Pursuit(SP)-KRLS, decouples the weight vector length from the dictionary size. Thus, the weight vector $\boldsymbol{\alpha}$ has a fixed maximum size K, which is independent from the size of the dictionary.

4.4.1 Dictionary Construction

For constructing our sparse dictionary, we adopt the surprise criterion introduced in Section 3.3.2. In what follows, we will modify the notation previously used to denote the dictionary (at iteration n) in order to distinguish between the dictionary of inputs, henceforth denoted as $\mathcal{D}_n^x$, and the dictionary of corresponding real outputs, henceforth denoted as $\mathcal{D}_n^y$.

We begin by recalling that in Section 3.3.1, the ALD measure was given by:

$$\delta_n = k_{nn} - \mathbf{k}_{n-1}^T \mathbf{K}_{n-1}^{-1} \mathbf{k}_{n-1} \tag{4.35}$$

where $k_{nn} = k(\mathbf{x}_n, \mathbf{x}_n)$ and $\mathbf{K}_{n-1}$ and $\mathbf{k}_{n-1}$ are the matrices whose entries are respectively given by:

$$[\mathbf{K}_{n-1}]_{i,j} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) \tag{4.36}$$

and

$$[\mathbf{k}_{n-1}]_i = k(\tilde{\mathbf{x}}_i, \mathbf{x}_n) \tag{4.37}$$

with $i, j = 1, \ldots, m_{n-1}$.

When we receive a new training pair $\mathbf{x}_n, y_n$, we compute the value of the surprise given by:

$$\mathcal{S}_n = \frac{1}{2} \ln \delta_n + \frac{(y_n - \mathbf{k}_{n-1}^T \boldsymbol{\alpha}_{n-1})^2}{2\delta_n} \tag{4.38}$$

where $\boldsymbol{\alpha}_{n-1}$ is the KRLS weight vector from the previous iteration. Based on the value of $\mathcal{S}_n$, there are two possible scenarios for the evolution of the dictionary:

- If $\mathcal{S}_n$ falls within the range of the pre-specified thresholds T_1 and T_2, that is,

$$T_1 \geq \mathcal{S}_n \geq T_2, \tag{4.39}$$

  then the elements of the training pair are admitted to the respective dictionaries:

$$\mathcal{D}_n^x = \mathcal{D}_{n-1}^x \cup \{\mathbf{x}_n\} \text{ and } \mathcal{D}_n^y = \mathcal{D}_{n-1}^y \cup \{\mathbf{y}_n\}. \tag{4.40}$$

- Otherwise, the dictionary remains unchanged:

$$\mathcal{D}_n^x = \mathcal{D}_{n-1}^x \text{ and } \mathcal{D}_n^y = \mathcal{D}_{n-1}^y. \tag{4.41}$$

Unlike other sparsification techniques, the surprise criterion not only examines the relation between the inputs of the incoming data samples and those of the dictionary samples, but it also takes into account how close the predicted output is to the real output (by considering the approximation error). Accordingly, in the very likely scenario where the input samples continue to arrive according to a specific distribution while the corresponding real outputs undergo a sudden change in their statistical properties, the SC will still admit new samples into the dictionary. ALD, for example, falls short of doing so because we would be still able to write the corresponding inputs in terms of previously admitted dictionary elements as there was no change in the statistical distribution of the inputs.

4.4.2 Imposing a hard limit on the size of the dictionary

As discussed in the previous section, in our algorithm, we employ the surprise criterion for admitting data samples to the dictionary. Unlike all previous KRLS algorithms which only considered one of sparsification or pruning, we also impose a hard limit on the size of the dictionary. Indeed, when the dictionary size exceeds αK ($\alpha > 1$), we start pruning dictionary elements using the remove-the-oldest criterion (see Section 3.4.1), that is, at each iteration after the dictionary size has exceeded αK, the oldest entry in each of $\mathcal{D}_n^x$ and $\mathcal{D}_n^y$ are removed. That way, on one hand, we are equipping our algorithm with a sparsification technique for building the dictionary, thus reducing the redundancy by not admitting training pairs that do not add to the knowledge of the system. On the other hand, we are reinforcing the algorithm's tracking ability by imposing a hard limit on the number of dictionary elements from which we will select the best dictionary elements for tracking as we will see in the next section.

To prune elements from the dictionary, we could have chosen a pruning criterion among other existing alternatives such as those we presented in Section 3.4. In a time-varying scenario, recent samples have more relevant information about the input-output relationship. Given that such relationship is constantly changing, older samples become less representative of it. The remove-the-oldest criterion is intuitively an appropriate pruning criterion to forget this old information and track changes in the input-output relationship. The criteria presented in Sections 3.4.2, 3.4.3 and 3.4.4 allow older dictionary elements to stay in the dictionary, which has a counteractive effect on our main goal, that is, the tracking of time-varying systems. Another alternative is to remove the oldest element under the condition that it is not among the K dictionary elements that were selected to approximate the regression function. However, we believe that for tracking purposes the remove-the-oldest criterion is a more appropriate criterion for the best representation of the time-varying dynamics of the inputs and outputs. We are motivated by our observation that the K selected elements can vary drastically between iterations, and even change completely from iteration to iteration in certain cases.

4.4.3 Choosing the best K elements for tracking using KSP

If, at the nth iteration, the input sample passes the SC test and is admitted to the dictionary $\mathcal{D}_n^x$ and the dictionary size exceeds K, we use KSP to select K elements in order to form

the LS regressor.

More specifically, we consider the αK ($\alpha > 1$) most recent entries in the dictionary $\mathcal{D}_n^x = [\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_{\alpha K}]$,[5] from which we want to select the K elements that will lead to the best approximation of the most recent $\mathcal{N}$ received target values. In this context, the KSP gram matrix $\mathbf{G}_n$ is obtained by evaluating $k(\cdot, \tilde{\mathbf{x}}_i),\ i = 1, \ldots, \alpha K$ at the $\mathcal{N}$ most recent inputs $\mathbf{x}_{n-\mathcal{N}+1}, \ldots, \mathbf{x}_n$. The vector $\mathbf{y}_n$ consists of the most recent $\mathcal{N}$ target values, i.e.,

$$\mathbf{y}_n = [y_{n-\mathcal{N}+1}, \ldots, y_n]^T. \tag{4.42}$$

The set T of indices obtained from KSP determines which subset of the dictionary is used to carry out the KRLS computations:

$$\breve{\mathcal{D}}_n^x = \{\mathcal{D}_n^x \text{ entries with indices given by } T\} \tag{4.43}$$

$$\breve{\mathcal{D}}_n^y = \{\mathcal{D}_n^y \text{ entries with indices given by } T\}. \tag{4.44}$$

It is important to emphasize that KSP does not need be run at every iteration after the dictionary size has exceeded K. Indeed, KSP is only "triggered" when a new element is added to the dictionary. We view the admission of a new element in the dictionary as an indication of a possible change in the input-output relationship being tracked, thus, requiring an update to the K-sized subset of the dictionary used in the KRLS computations.

Using $\breve{\mathcal{D}}_n^x$, we compute the new kernel matrix $\mathbf{K}_n$ and its inverse $\mathbf{K}_n^{-1}$ where

$$[\mathbf{K}_n]_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j),\ \ \forall \mathbf{x}_i, \mathbf{x}_j \in \breve{\mathcal{D}}_n^x, i, j = 1, \ldots, K.$$

Finally, we compute the weight vector:

$$\boldsymbol{\alpha}_n = \mathbf{K}_n^{-1} \breve{\mathbf{y}}_n^{\mathcal{D}} \tag{4.45}$$

where $\breve{\mathbf{y}}_n^{\mathcal{D}}$ is the vector $\breve{\mathbf{y}}_n^{\mathcal{D}} = [y_1, \ldots, y_i, \ldots, y_K]^T$, $\underset{1 \le i \le K}{y_i} \in \breve{\mathcal{D}}_n^y$. The computed weight vector $\boldsymbol{\alpha}_n$ will be used in future KRLS iterations until a new subset of $\mathcal{D}_n^x$ has been selected by KSP, thus requiring the update of the weight vector $\boldsymbol{\alpha}$. Accordingly, following the admission of a new input-output pair $(\mathbf{x}_n, y_n)$, SP-KRLS will find the best K dictionary

[5] We assume that the elements of the dictionary are sorted by the time they were admitted to the dictionary - oldest first.

elements to track the most recent received data samples.

4.4.4 Effect of the parameter α

It is interesting to note that we can leverage the parameter α to have control over the number of dictionary elements from which we can choose the best K dictionary elements for tracking. Indeed, we might be presented with scenarios (resulting from specific changes in the system itself or changes in the statistics of the received data) where a large α is needed to be able to capture better the dynamics of the input-output relationship in the set of elements from which the subset will be selected by KSP. On the other hand, in other scenarios, the tracking performance of the algorithm might actually improve when using a smaller α if, for instance, recently received input-output pairs are not very different (despite being admitted by the SC criterion). This implies that there would be no need for a large α, instead, focusing on a smaller subset αK of the most recent dictionary entries, which would result in a better tracking in such scenarios.

4.4.5 Summary of Proposed Algorithm

SP-KRLS is summarized in Algorithm 6. When the system receives a new pair $(\mathbf{x}_n, y_n)$, it is checked against the SC test (4.35). We are presented with two possible scenarios:

1. If it does not pass the SC test, the input vector is not added to the dictionary and the weight vector is updated appropriately via the KRLS recursions [9].

2. If the input vector is admitted to the dictionary, we are further presented with two cases:

   (a) If the size of the dictionary is less than or equal to αK the weight vector is updated appropriately via the KRLS recursions .

   (b) If the size of the dictionary is larger than αK, we use KSP (Algorithm 5) to identify the K input vectors that will be used by the KRLS.

Thus, by using KSP as a building block of SP-KRLS, we decouple the size of the weight vector from the dictionary size and equip KRLS with the ability to predict well the most recent N target values in order to efficiently track the changes of a time-varying system.

Algorithm 6 SP-KRLS

Parameters: T_1, T_2, K, $\mathcal{N}$, α.

Initialize: $\mathbf{k}_1 = [k_{11}]$, $\mathbf{k}_1^{-1} = [1/k_{11}]$, $\boldsymbol{\alpha}_n = (y_1/k_{11})$, $\mathbf{P}_1 = [1]$, $m = 1$.

for $n = 2, 3, \ldots$ **do**

Get new sample: $(\mathbf{x}_n, y_n)$

$\mathbf{k}_{n-1}$ with $[\mathbf{k}_{n-1}]_i = k(\tilde{\mathbf{x}}_i, \mathbf{x}_n), i = 1, \ldots, m$.

$k_{nn} = k(\mathbf{x}_n, \mathbf{x}_n)$

SC Test:

$\boldsymbol{a}_n = \mathbf{K}_{n-1}^{-1}\mathbf{k}_{n-1}$

$\delta_n = k_{nn} - \mathbf{k}_{n-1}^T \boldsymbol{a}_n$

$\mathcal{S}_n = \frac{1}{2}\ln\delta_n + \frac{(y_n - \mathbf{k}_{n-1}^T\boldsymbol{\alpha}_{n-1})^2}{2\delta_n}$

if $(T_1 \geq \mathcal{S}_n \geq T_2)$ **then** {*Add* $\mathbf{x}_n$ *to dictionary*}

$\mathcal{D}_n^x = \mathcal{D}_{n-1}^x \cup \{\mathbf{x}_n\}$, $\mathcal{D}_n^y = \mathcal{D}_{n-1}^y \cup \{\mathbf{y}_n\}$

$m = m + 1$

if $m \leq K$ **then** {*SC-KRLS approach*}

$$\mathbf{K}_n^{-1} = \frac{1}{\delta_n}\begin{bmatrix} \delta_n\mathbf{K}_{n-1}^{-1} + \boldsymbol{a}_n\boldsymbol{a}_n^T & -\boldsymbol{a}_n \\ -\boldsymbol{a}_n^T & 1 \end{bmatrix}$$

$$\mathbf{P}_n = \begin{bmatrix} \mathbf{P}_{n-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

$$\boldsymbol{\alpha}_n = \begin{bmatrix} \boldsymbol{\alpha}_{n-1} - \frac{\boldsymbol{a}_n}{\delta_n}\left(y_n - \mathbf{k}_{n-1}^T\boldsymbol{\alpha}_{n-1}\right) \\ \frac{1}{\delta_n}\left(y_n - \mathbf{k}_{n-1}^T\boldsymbol{\alpha}_{n-1}\right) \end{bmatrix}$$

else {*KSP approach*}

if $(m > \alpha K)$, remove the oldest element in $\mathcal{D}_n^x$ and $\mathcal{D}_n^y$.

Find the set, T, of indices of the best K elements in $\mathcal{D}_n^x$ using Algorithm 5.

$\breve{\mathcal{D}}_n^x = \{\mathcal{D}_n^x$ entries with indices given by $T\}$.

$\breve{\mathcal{D}}_n^y = \{\mathcal{D}_n^y$ entries with indices given by $T\}$.

Find $\mathbf{K}_n^{-1}$ where $[\mathbf{K}_n]_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j), \ \forall \mathbf{x}_i, \mathbf{x}_j \in \breve{\mathcal{D}}_n^x, i, j = 1, \ldots, K$.

$\boldsymbol{\alpha}_n = \mathbf{K}_n^{-1}\breve{\mathbf{y}}_n^{\mathcal{D}}$ where $\breve{\mathbf{y}}_n^{\mathcal{D}} = [y_1, \ldots, y_i, \ldots, y_K]^T, \underset{1\leq i\leq K}{y_i} \in \breve{\mathcal{D}}_n^y$.

end if

else {*SC-KRLS approach: Case "dictionary unchanged"*}

$\mathcal{D}_n^x = \mathcal{D}_{n-1}^x$, $\mathcal{D}_n^y = \mathcal{D}_{n-1}^y$

$\mathbf{q}_n = \frac{\mathbf{P}_{n-1}\boldsymbol{a}_n}{1 + \boldsymbol{a}_n^T\mathbf{P}_{n-1}\boldsymbol{a}_n}$

$\mathbf{P}_n = \mathbf{P}_{n-1} - \mathbf{q}_n\boldsymbol{a}_n^T\mathbf{P}_{n-1}$

$\boldsymbol{\alpha}_n = \boldsymbol{\alpha}_{n-1} + \mathbf{K}_n^{-1}\mathbf{q}_n\left(y_n - \mathbf{k}_{n-1}^T\boldsymbol{\alpha}_{n-1}\right)$

end if

end for

Output: $\mathcal{D}_n^x$, $\mathcal{D}_n^y$, $\boldsymbol{\alpha}_n$.

4.4.6 Computational Considerations

Given that SP-KRLS does not need to run at every iteration, the computational complexity of SP-KRLS is significantly reduced. Indeed, in Cases 1 and 2a of the algorithm (see Section 4.4.5), we only need to update the KRLS matrices using operations (addition, subtraction and multiplication) of order $O(m_n^2)$, without having to perform any matrix inversion that would be of order $O(m_n^3)$. We do emphasize, however, that even in Case 2b of SP-KRLS (see Section 4.4.5) where KSP is executed, the matrix inversion is of order $O(K^3)$ with $K \ll m_n$. Other KRLS-type algorithms have a complexity of $O(m_n^2)$ (e.g., ALD-KRLS [9], SW-KRLS [12] and FB-KRLS [13]). Given the computational considerations that we just stated, we believe that a general order of complexity that would take into account which cases of SP-KRLS are actually being executed at each iteration would depend on the nature of the scenarios being considered and the data that is being used and would be very difficult to obtain.

4.5 Conclusion

In this chapter, we first presented the Subspace Pursuit algorithm which was introduced in the context of compressive sensing as a method to reconstruct unknown sparse signals. We also introduced the Kernel Matching Pursuit and Kernel Basis Pursuit algorithms which are of particular interest to our work as they attempt to solve a similar problem to ours, that of learning a regression function by means of sparse approximation. We argued that, unlike all previous KRLS algorithms, we need to decouple the weight vector size from the dictionary size to design an algorithm that can track well non-stationary systems. This motivated us to propose the Kernel Subspace Pursuit (KSP) algorithm. We presented KSP as a tool to learn a regression function by means of sparse approximation using a finite number of basis functions selected from a kernel-based dictionary. Finally, we use KSP in conjunction with KRLS to propose a new KRLS algorithm for tracking time-varying changes. KSP allows us to approximate the most recent $\mathcal{N}$ target values by constructing a K-sparse approximation of the target function (representing the input-output relationship of the most recent $\mathcal{N}$ training pairs) using the K dictionary elements that track best those $\mathcal{N}$ target values.

Chapter 5

Simulations

In this chapter, we first experimentally compare the performance of KSP in learning non-linear functions to that of KMP [45] and KBP [46]. Then, we test SP-KRLS and compare its performance in tracking a time-varying system having a normally distributed input to that of the following algorithms: ALD-KRLS [9], SW-KRLS [12] and FB-KRLS [13]. Next, we test the performance of SP-KRLS in predicting the highly chaotic Mackey-Glass time series [51]. Finally, we investigate the effect of the algorithm parameters on the performance of SP-KRLS.

5.1 Performance of KSP

In this experiment, we test KSP in learning different non-linear functions and compare its performance to that of KMP and KBP. Given a function $g(x)$ that is corrupted by noise, our aim is to learn $g(x)$ from a function $h(x)$ as follows:

$$h(x) = g(x) + \mathcal{N}(0, \sigma^2) \tag{5.1}$$

where $\mathcal{N}$ represents white Gaussian noise with variance σ^2. In this experiment, we consider the following functions:

1. $g(x) = \cos(\exp(\omega x))$,
2. $g(x) = \sin(\exp(\omega x))$,
3. $g(x) = \tanh(\omega x)$,

4. $g(x) = \tan(\omega x)$,

where ω is a pre-specified real parameter. We also test our method using synthetic data described by Donoho and Johnstone [52]. These data were generated using the following functions:

1. Doppler function: $g(x) = [x(1-x)]^{0.5} \sin(2\pi \frac{1+\epsilon}{x+\epsilon})$ with $\epsilon = 0.5$,
2. Blocks function: $g(x) = \sum_{j=1}^{11} a_j p(x - b_j)$ where:
   - $p(x) = \{1 + \text{sgn}((x)\}/2$,
   - a_j is the j-th element of $[4, -5, 3, -4, 5, -4.2, 2.1, 4.3, -3.1, 2.1, -4.2]$ and
   - b_j is the j-th element of $[0.1, 0.13, 0.15, 0.23, 0.25, 0.40, 0.44, 0.65, 0.76, 0.78, 0.81]$.

The input is formed as follows: Each input $\mathbf{x}_i$ is drawn i.i.d. from the interval $[0, 1]$ according to a uniform distribution. The input is then passed through one of the non-linear functions mentioned above. The result is finally corrupted with 20 dB of white Gaussian noise. We use a training set of size 400 and a test set of size 100. A Gaussian kernel is used with width $\sigma = 3$ for all of the testing scenarios and $\omega = 0.3$ for the corresponding functions $g(x)$. The parameters ω and σ are chosen via 10-fold cross-validation. The results, averaged out over 50 Monte-Carlo simulations, are shown in Table 5.1 where, following each function, we identify the number B of basis functions used in the construction of the approximation function. We use the same value for B that was used in [46]. Results show that KSP outperforms both KMP and KBP in learning 5 of the 6 functions shown in Table 5.1 with only KBP performing better than KSP in the case of the Doppler function. It is interesting to note that in all of the first 4 cases, KSP has an error that is at least 9 times smaller than one of the two other algorithms.

To understand better the computational burden of these algorithms, we note that both KSP and KMP are of order $O(m_n n)$ per iteration and KBP using the LARS implementation is of order $O(m_n K)$ per iteration. Our simulations showed that, for all the different learning scenarios presented above, increasing the maximum number, $\ell_{\max}$, of KSP iterations beyond 5 iterations (we considered values of $\ell_{\max}$ up to 50) led to a very minimal improvement in the MSE performance of KSP, and such an improvement was practically negligible. Hence, the number of KSP iterations that were executed in order to build the regression function was fixed at 5. Since, by design, both KMP and KBP need a number of iterations equal

to the desired number of basis functions in the expansion, KMP and KBP required 95 iterations in each of the first 4 learning scenarios and 50 and 70 respectively in the last two scenarios. Given the comparable complexity per iteration of the three algorithms, the computational burden ensuing from running KSP for this small number of iterations is much smaller than that resulting from running KMP and KBP for the corresponding large number of iterations.

Table 5.1 MSE Performance of KSP, KMP and KBP in learning synthetic data using a Gaussian kernel

	Algorithms		
Functions	KSP	KMP	KBP
$\cos(\exp(\omega\mathbf{x}))$, B=95	0.01	0.0896	0.0237
$\sin(\exp(\omega\mathbf{x}))$, B=95	0.00974	0.0115	0.0913
$\tanh(\omega\mathbf{x})$, B=95	0.00981	0.0394	0.114
$\tan(\omega\mathbf{x})$, B=95	0.00976	0.0661	0.159
Doppler, B=50	0.0861	0.0941	0.0841
Blocks, B=70	0.294	0.366	0.524

5.2 Tracking of a Time-Varying System

In this experiment, we experimentally test SP-KRLS and compare its tracking performance to that of the following algorithms: ALD-KRLS [9], SW-KRLS [12] and FB-KRLS [13]. For this purpose, we consider a time-varying system composed of a time-varying linear filter followed by a static non-linearity.

The input signal, whose elements x_i are drawn i.i.d. from a normal distribution with mean 0 and variance 0.5, is passed through a linear filter that varies in time along 3200 iterations as follows: During the first 1500 iterations, its impulse response is given by:

$$h_1(n) = \delta(n) - 0.37\delta(n-1) - 0.48\delta(n-2) + 0.81\delta(n-3). \tag{5.2}$$

On iteration 1501, the filter is abruptly changed and its impulse response becomes

$$h_2(n) = \delta(n) - 0.83\delta(n-1) + 0.67\delta(n-2) + 0.72\delta(n-3) \tag{5.3}$$

which remains constant for 1200 iterations. At iteration 2701, the filter starts linearly

changing, throughout 500 iterations, from $h_2(n)$ to

$$h_3(n) = \delta(n) - 0.5\delta(n-1) - 0.25\delta(n-2) + 0.4\delta(n-3).^1 \tag{5.4}$$

The output of the linear filter is then passed through the static non-linear function

$$f(x) = \tanh(x). \tag{5.5}$$

The resulting signal is finally corrupted with 20 dB of white Gaussian noise. We use 400 sample points as a test set (different in each phase of the scenario according to the linear filter) and a time embedding of 4 , i.e., $\mathbf{x}_n = [x_n, x_{n-1}, x_{n-2}, x_{n-3}]^T$. The input signal is normally distributed with mean 0 and variance 0.5.

For SP-KRLS, the thresholds for learnable data are set to $T_1 = 3$ and $T_2 = -3$. For SP-KRLS, FB-KRLS and SW-KRLS, the parameters M and K are set to $M = K = 200$. In addition for SP-KRLS, $N = 10$ and $\alpha = 1.5$. For FB-KRLS, the step-size parameter is set to $\mu = 0.01$. The regularization parameter for SP-KRLS and FB-KRLS is set to $\lambda = 0.001$. The accuracy parameter for ALD-KRLS is set to $\nu = 0.001$. For all algorithms, a Gaussian kernel is used with a width $\sigma = 0.8$. Algorithm parameters were chosen via 5-fold cross-validation.

The results, averaged over 50 Monte-Carlo simulations, are shown in Figure 5.1. The performance of ALD-KRLS is the worst as it is not designed to be a tracking algorithm. SP-KRLS outperforms both SW-KRLS and FB-KRLS in tracking the system: Following changes in the linear filter (indicated by the vertical lines in the plot), the MSE curve of SP-KRLS is the fastest to change, thus capturing the fastest the change in the system. Moreover, the MSE of SP-KRLS converges in each phase to the smallest value among all three algorithms. Finally, Table 5.2 displays the MSE values averaged out over the last two phases. SP-KRLS achieves 15.6% improvement over FB-KRLS and 17.6% improvement over SW-KRLS.

[1]The linear transition occurs at the rate 1/500 as follows. Denote by $(\rho_k)_{k=0}^{499}$ the sequence given by $\rho_0 = 1$ and $\rho_{k+1} = \rho_k - 0.002$ for $0 \leq k \leq 498$. Thus, at iteration n ($2701 \leq n \leq 3200$), the filter is given by: $\rho_{n-2701}\, h_2(n) + (1 - \rho_{n-2701}) h_3(n)$.

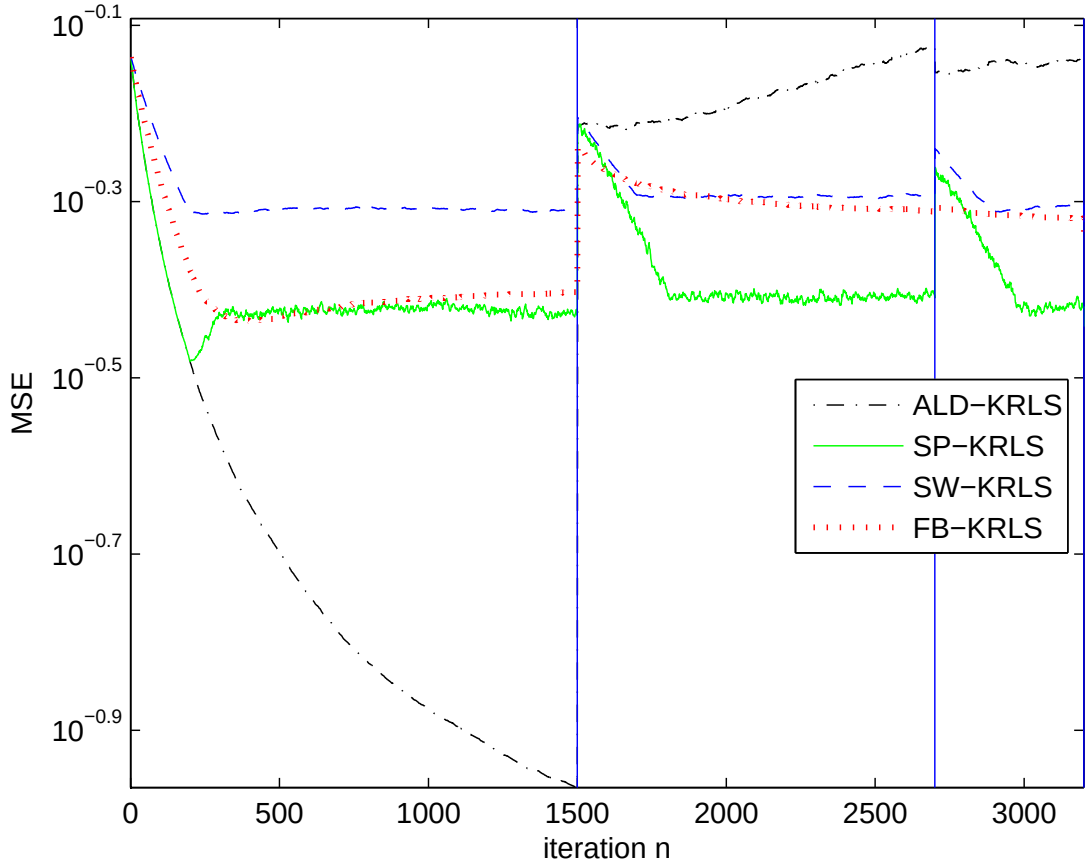

Fig. 5.1 Performance of SP-KRLS vs other KRLS algorithms on a time-varying Wiener system.

5.3 Prediction of the Mackey-Glass Time Series

In this experiment, we perform one-step prediction on the highly chaotic Mackey-Glass time series [51]. The algorithm is trained online on 800 sample points, and the MSE performance is calculated at each iteration on a test set of 200 sample points. The Gaussian kernel used in this experiment has a width[2] $\sigma = 8$. For SP-KRLS, the thresholds for learnable data are set to $T_1 = 200$ and $T_2 = -4$. A large T_1 was used to disable abnormality detection, T_2 was chosen via 5-fold cross-validation. The rest of the algorithm parameters are the same as in

[2]In this experiment, we had to use a σ with a large width as all algorithms performed very poorly if the width was small (e.g., as in the previous experiment).

Table 5.2 Performance comparison of average MSE values

Algorithm	MSE
ALD-KRLS	0.677
SP-KRLS	0.431
SW-KRLS	0.523
FB-KRLS	0.511

the first experiment. In Figure 5.2, we can clearly see that our algorithm outperforms the other algorithms in predicting a Mackey-Glass time series as the MSE curve for SP-KRLS converges to a lower steady-state value.

5.4 Effect of changing K on the performance of SP-KRLS

In this section, we investigate the effect of changing K on the performance of SP-KRLS. We consider again the time-varying system from Section 5.2. The first three phases (iterations 1 to 3200) are the same, but we add a 4-th phase where the impulse response of the filter remains constant at $h_3(n)$ for the last 800 iterations. We run SP-KRLS for 4 different values of K: 20, 50, 100 and 200. The rest of the algorithm parameters are the same as in Section 5.2. The results, averaged over 50 Monte-Carlo simulations, are shown in Figure 5.3.

Following the first change in the linear filter, all 4 algorithms start adapting (tracking) to the change at roughly the same rate with the rates being ordered from fastest to slowest according to the following order of K: 20, 50, 100 and 200. This can be particularly seen for the blue ($K = 50$), green ($K = 100$), black ($K = 200$) curves as the line where the curve represents the tracking phase is steepest for $K = 50$, less steep for $K = 100$ and least steep for $K = 200$ (notice how the green and black lines intersect due to the difference in slopes).

We do note, however, that the larger K is, the lowest the steady-state MSE to which the algorithms converge. This goes in line with the trade-off that we already discussed in Chapter 4, specifically, when we mentioned that a larger dictionary represents better the input-output relationship and, thus, leads the algorithm to converge to a smaller steady-state MSE. We do emphasize though that, here, the notion of dictionary is thought of in terms of the "subset" of the dictionary chosen by KSP and which is of equal size to K, that

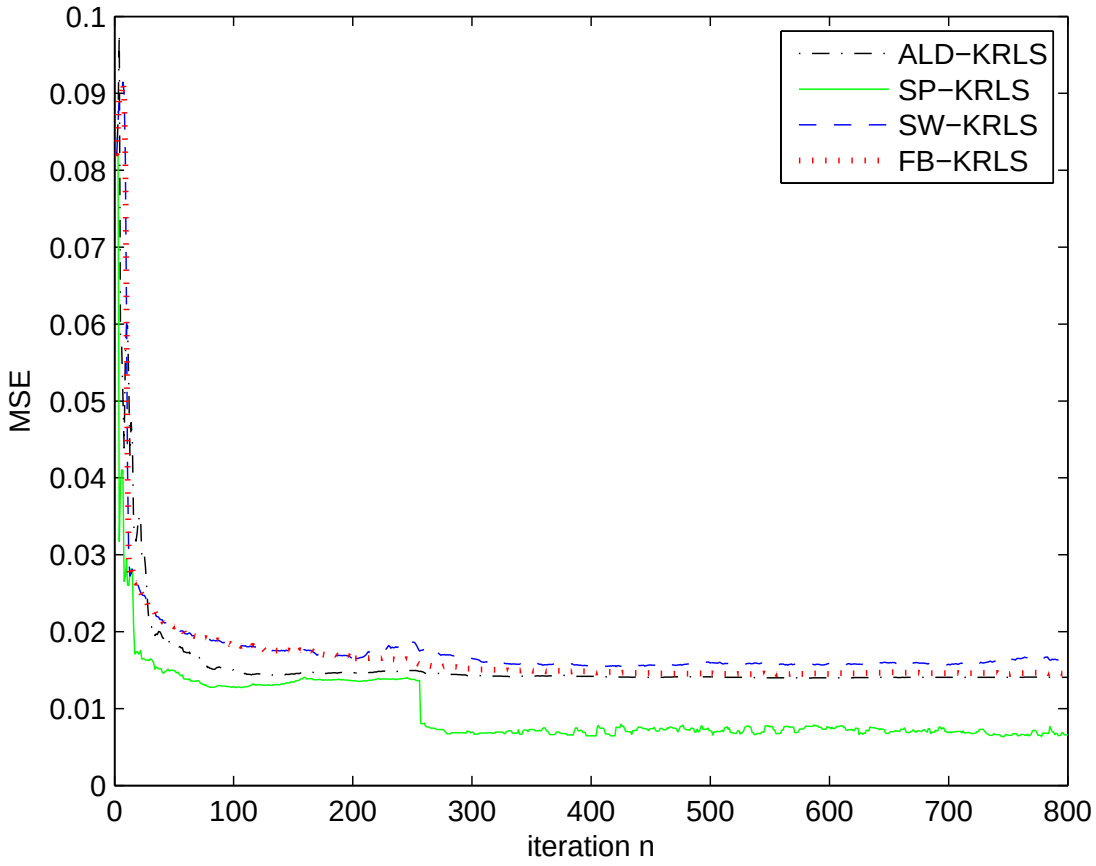

Fig. 5.2 Performance of SP-KRLS vs other KRLS algorithms in predicting a Mackey-Glass time series.

is, the larger the K, the larger the subset of the dictionary, thus, the better representation of the system and the lower the steady-state MSE to which the algorithm converges.

Following the second change, the algorithms exhibit a similar behavior in terms of speed of tracking to that seen following the first change. Finally, we note that, following the final change, the MSE curve of each of the 4 algorithms becomes almost flat. This implies that following that last change, the rate of adaptation (tracking speed) of the algorithms became slower following yet another change, thus the algorithms needed more time to start converging.

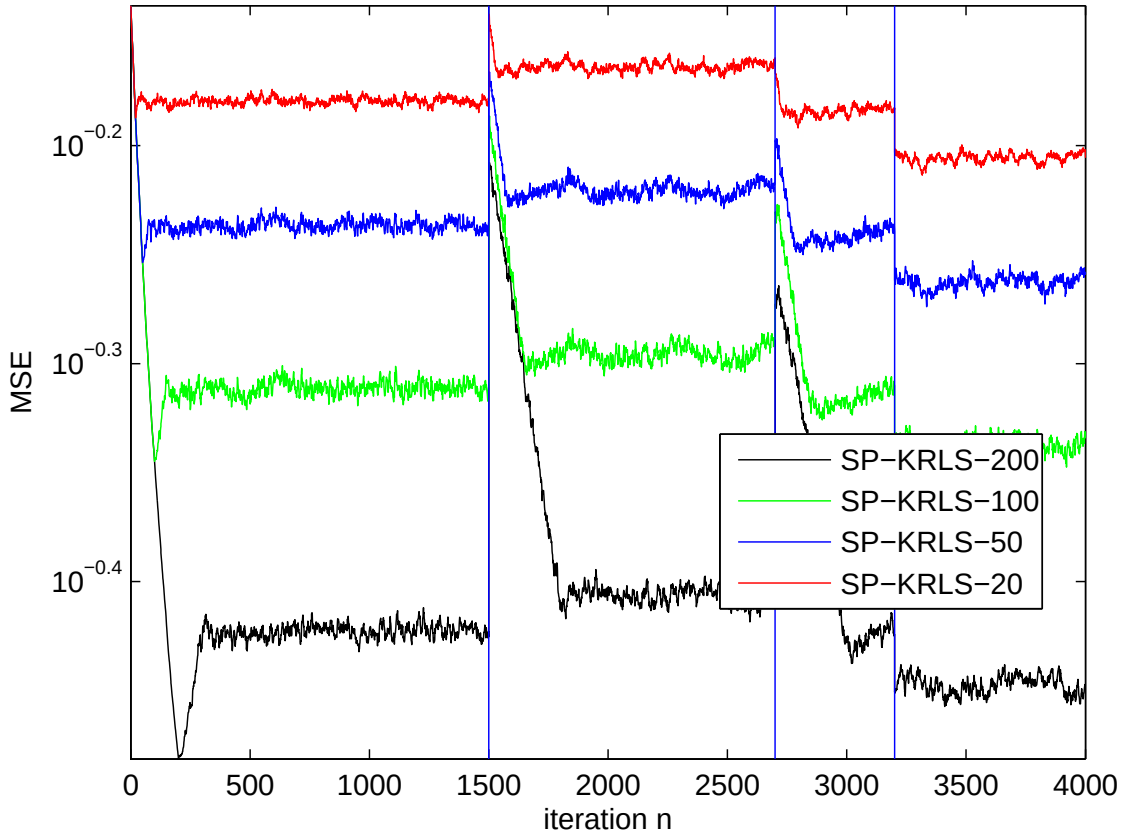

Fig. 5.3 Effect of changing K on the performance of SP-KRLS.

5.5 Effect of changing α on the performance of SP-KRLS

In this section, we investigate the effect of changing α on the performance of SP-KRLS. We consider again the system from Section 5.2. In this experiment, we fix $K = 200$ and run SP-KRLS for 4 different values of α: 1.25, 1.5, 2 and 5. The rest of the algorithm parameters are not changed from Section 5.2. The results, averaged out over 25 Monte-Carlo simulations, are shown in Figure 5.4.

In this experiment, by changing α, we control the number of dictionary elements that we consider in choosing the best K elements for tracking (via KSP). Accordingly, for example, for $\alpha = 1.25$, KSP will choose the best K elements from the most recent $\alpha K = 250$ dictionary elements, while for $\alpha = 5$, the choice is done among the most recent $\alpha K = 1000$

dictionary elements.

Following the first change, we can see that the algorithm with the smallest α tracks the change the fastest (steepest line) and converges to the smallest steady-state MSE. On the other hand, the algorithm with $\alpha = 5$ is the slowest in tracking and by the time the second filter change takes place, it hasn't converged yet to a steady-state MSE. This can be understood from the fact that for a larger α, the number of dictionary elements from which KSP selects the K-sized support set becomes larger, and thus, contains a larger number of older elements (as opposed to a small α which leads the support set being chosen from a smaller number of more recent dictionary elements). As the algorithm is sub-optimal, some of the elements selected by KSP might be older elements. This results in slowing down the tracking performance of the algorithm for an increasing α. Finally, we note that, following the second change, the algorithms exhibit a similar behavior in terms of speed of tracking to that seen following the first change.

It is important to note that we ran this experiment for different values of $\ell_{\max}$, the maximum number of KSP iterations, going up to 50. Results showed that increasing the number of iterations beyond $\ell_{\max} = 5$ iterations led to a very minimal improvement in the MSE performance, and such an improvement was practically negligible. Thus, we decided to adopt $\ell_{\max} = 5$.

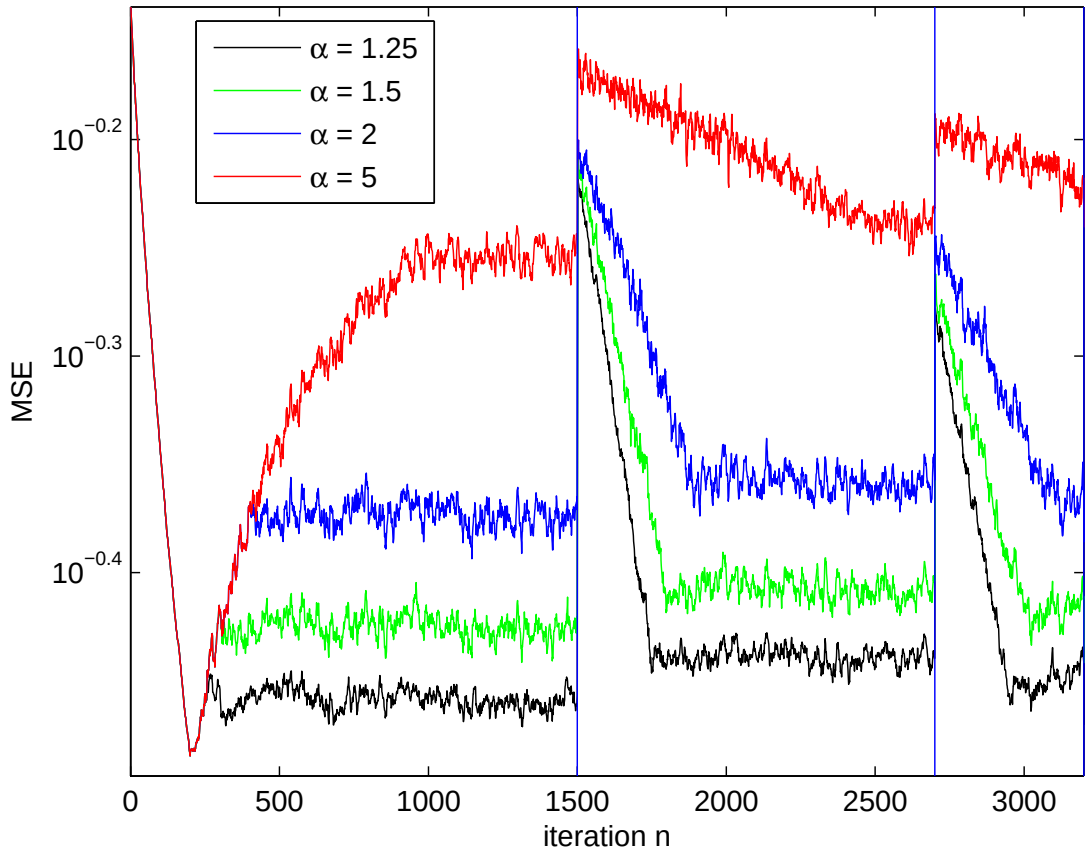

Fig. 5.4 Effect of changing α on the performance of SP-KRLS.

Chapter 6

Conclusions and Future Research

6.1 Concluding Remarks

In this thesis, we presented a new KRLS algorithm, SP-KRLS, that is able to efficiently track time-varying systems. Indeed, in many applications, the statistics of signals change across time, thus, it is important to design algorithms that retain the ability to effectively track time-varying systems.

In designing our algorithm, we considered both a sparsification technique and a pruning strategy to construct our dictionary unlike all previous KRLS algorithms which considered one of the two. Among the various various sparsification and pruning techniques discussed in the literature and which we presented in this thesis, we adopted the surprise criterion as the sparsification and the remove-the-oldest criterion as the pruning criterion.

Previous KRLS algorithms also suffered from the coupling between the weight vector size and the dictionary size. If we were to limit the size of the weight vector to favor the algorithm's tracking ability, the dictionary, thus small-sized, would model poorly the dynamics of the input-output relationship over time. On the other hand, if we were to allow a large dictionary, the size of the weight vector would be large. As such the algorithm would need to adapt a large number of weights, which slows down adaptation and damages the algorithm's tracking ability. To address this hindering trade-off, we decoupled in our algorithm the dictionary size from the weight vector size, which we fix independently from the dictionary size.

Motivated by the need to decouple the dictionary size from the weight vector size, we introduced Kernel Subspace Pursuit (KSP) as a tool to construct a regression function

by means of a sparse approximation using a finite number of functions selected from a kernel-based dictionary. We were particularly interested in predicting well the most recent $\mathcal{N}$ target values in order to efficiently track the changes of a time-varying system. As such, we considered constructing a K-sparse approximation of the target function (representing the input-output relationship of the most recent $\mathcal{N}$ training pairs) using the K dictionary elements that track best those $\mathcal{N}$ target values.

To that end, we adopted KSP in conjunction with KRLS in our proposed method. KSP selects the K dictionary elements that will be used to form the LS regressor. Those elements are used in the subsequent KRLS computations until a new training pair has been added to the dictionary which which we regard as an indication of a potential change in the input-output relationship being tracked, thus, requiring an update to the K-sized subset of the dictionary used in the KRLS computations.

Simulation results presented showed that our algorithm outperformed other well-known KRLS algorithms in both tracking time-varying systems and predicting highly chaotic time series.

6.2 Future Research

The research presented in this thesis can be further extended in several directions.

In the algorithm presented in Chapter 4, KSP needs not be run at every iteration. Instead, it is triggered only when a new element is added to the dictionary after having passed the surprise criterion test. As future work, we intend to examine further the changes in the dictionary subsets between KSP iterations in order to possibly reduce even more the number of times KSP needs to be run.

Another promising direction for future work is investigating the merits of the convex combination of adaptive filters which has been extensively studied for classical adaptive filtering algorithms (e.g., [53] and [54]). This theory considers the problem of combining the outputs of different adaptive filtering algorithms or the same algorithms but with varying algorithms parameters in order to obtain adaptive filtering algorithms with superior tracking capability. As such, it would be interesting to investigate the tracking performance of combined versions of SP-KRLS with varying weight vector sizes or varying values of the algorithm parameter α.

Finally, SP-KRLS could be further extended by adapting the size of the subset of the

dictionary that is selected by KSP (which is equal to the effective size of the weight vector), instead of fixing that size. This might prove to be particularly beneficial for scenarios with substantial and frequent changes. It would be indeed interesting to study the possible gains that could be achieved, for instance, by adapting the size of the selected subset of the dictionary following consecutive changes in the system that lead to the algorithm's inability to track well with the given size of the weight vector.

References

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, vol. 1. Springer New York, 2001.

[2] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, Sept. 1995.

[3] A. Smola and B. Schölkopf, *Learning with kernels*. MIT Press, Cambridge, MA, 1 ed., 2002.

[4] B. Schölkopf, A. Smola, and K.-R. Müller, "Kernel principal component analysis," in *International Conference on Artificial Neural Networks*, pp. 583–588, 1997.

[5] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K. Muller, "Fisher discriminant analysis with kernels," in *IEEE Signal Processing Society Workshop on Neural Networks for Signal Processing*, pp. 41–48, 1999.

[6] D. Manolakis, V. Ingle, and S. Kogon, *Statistical and Adaptive Signal Processing: Spectral Estimation, Signal Modeling, Adaptive Filtering, and Array Processing*, vol. 46. Artech House, 2005.

[7] B. Widrow and M. Hoff, "Adaptive switching circuits," *The IRE Western Electronic Show and Convention Records*, pp. 96–104, Aug. 1960.

[8] K. Ozeki and T. Umeda, "An adaptive filtering algorithm using an orthogonal projection to an affine subspace and its properties," *The IEICE Transactions*, vol. 67, pp. 126–132, May 1984.

[9] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least-squares algorithm," *IEEE Transactions on Signal Processing*, vol. 52, pp. 2275–2285, Aug. 2004.

[10] J. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Processing Letters*, vol. 9, pp. 293–300, June 1999.

[11] W. Liu, I. Park, and J. Príncipe, "An information theoretic approach of designing sparse kernel adaptive filters," *IEEE Transactions on Neural Networks*, vol. 20, pp. 1950–1961, Dec. 2009.

[12] S. Van Vaerenbergh, J. Via, and I. Santamana, "A sliding-window kernel rls algorithm and its application to nonlinear channel identification," in *IEEE International Conference on Acoustics Speech and Signal Processing*, vol. 5, p. V, 2006.

[13] S. Van Vaerenbergh, I. Santamaría, W. Liu, and J. Príncipe, "Fixed-budget kernel recursive least-squares," in *IEEE International Conference on Acoustics Speech and Signal Processing*, pp. 1882–1885, 2010.

[14] W. Dai and O. Milenkovic, "Subspace pursuit for compressive sensing signal reconstruction," *IEEE Transactions on Information Theory*, vol. 55, pp. 2230–2249, May 2009.

[15] A. Friedman, *Foundations of Modern Analysis*. Dover publications, 2010.

[16] W. Rudin, *Principles of Mathematical Analysis*, vol. 3. McGraw-Hill New York, 1964.

[17] M. Aizerman, E. Braverman, and L. Rozonoer, "Theoretical foundations of the potential function method in pattern recognition learning," *Automation and Remote Control*, vol. 25, pp. 821–837, 1964.

[18] T. Hofmann, B. Schölkopf, and A. Smola, "Kernel methods in machine learning," *The Annals of Statistics*, vol. 36, pp. 1171–1220, June 2008.

[19] A. Berlinet and C. Thomas-Agnan, *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Kluwer Academic, 1 ed., 2004.

[20] N. Aronszajn, "Theory of reproducing kernels," *Transactions of the American Mathematical Society*, vol. 68, pp. 337–404, May 1950.

[21] G. Kimeldorf and G. Wahba, "Some results on tchebycheffian spline functions," *Journal of Mathematical Analysis and Applications*, vol. 33, pp. 82–95, Jan. 1971.

[22] B. Schölkopf, R. Herbrich, and A. Smola, "A generalized representer theorem," in *Computational Learning Theory*, pp. 416–426, 2001.

[23] L. Fatone, M. C. Recchioni, and F. Zirilli, "Wavelet bases made of piecewise polynomial functions: Theory and applications," *Applied Mathematics*, vol. 2, no. 2, pp. 196–216, 2011.

[24] M. Bronstein, A. Bronstein, M. Zibulevsky, and Y. Zeevi, "Blind deconvolution of images using optimal sparse representations," *IEEE Transactions on Image Processing*, vol. 14, pp. 726–736, June 2005.

[25] J. M. Duarte-Carvajalino and G. Sapiro, "Learning to sense sparse signals: Simultaneous sensing matrix and sparsifying dictionary optimization," *IEEE Transactions on Image Processing*, vol. 18, pp. 1395–1408, July 2009.

[26] S. Ravishankar and Y. Bresler, "Mr image reconstruction from highly undersampled k-space data by dictionary learning," *IEEE Transactions on Medical Imaging*, vol. 30, pp. 1028–1041, May 2011.

[27] W. Liu, P. P. Pokharel, and J. C. Principe, "The kernel least-mean-square algorithm," *IEEE Transactions on Signal Processing*, vol. 56, pp. 543–554, Feb. 2008.

[28] B. Chen, S. Zhao, P. Zhu, and J. Príncipe, "Quantized kernel least mean square algorithm," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, pp. 22–32, Jan. 2012.

[29] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, 2006.

[30] J. Platt, "A resource-allocating network for function interpolation," *Neural computation*, vol. 3, pp. 213–225, Mar. 1991.

[31] H. Fan and Q. Song, "A sparse kernel algorithm for online time series data prediction," *Expert Systems with Applications*, vol. 40, pp. 2174–2181, May 2013.

[32] H. Fan, Q. Song, and Z. Xu, "An information theoretic kernel algorithm for robust online learning," in *IEEE International Joint Conference on Neural Networks*, pp. 1–8, 2012.

[33] Y.-I. Moon, B. Rajagopalan, and U. Lall, "Estimation of mutual information using kernel density estimators," *Physical Review E*, vol. 52, p. 2318, Sept. 1995.

[34] C. Richard, J. Bermudez, and P. Honeine, "Online prediction of time series data with kernels," *IEEE Transactions on Signal Processing*, vol. 57, pp. 1058–1067, Mar. 2009.

[35] Y. Liu, H. Wang, J. Yu, and P. Li, "Selective recursive kernel learning for online identification of nonlinear systems with narx form," *Journal of Process Control*, vol. 20, no. 2, pp. 181–194, 2010.

[36] W. He and S. Wu, "A kernel-based perceptron with dynamic memory," *Neural Networks*, vol. 25, pp. 106–113, Jan. 2012.

[37] D. Nguyen-Tuong and J. Peters, "Incremental online sparsification for model learning in real-time robot control," *Neurocomputing*, vol. 74, pp. 1859–1867, May 2011.

[38] O. Dekel, S. Shalev-Shwartz, and Y. Singer, "The forgetron: A kernel-based perceptron on a budget," *SIAM Journal on Computing*, vol. 37, pp. 1342–1372, Jan. 2008.

[39] Y. Le Cun, J. Denker, and S. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, pp. 598–605, 1989.

[40] B. Hassibi, D. Stork, and G. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE International Conference on Neural Networks*, pp. 293–299, 1993.

[41] B. De Kruif and T. De Vries, "Pruning error minimization in least squares support vector machines," *IEEE Transactions on Neural Networks*, vol. 14, pp. 696–702, May 2003.

[42] M. Lázaro-Gredilla, S. Van Vaerenbergh, and I. Santamaría, "A bayesian approach to tracking with kernel recursive least-squares," in *IEEE International Workshop on Machine Learning for Signal Processing*, pp. 1–6, 2011.

[43] D. Rzepka, "Fixed-budget kernel least mean squares," in *IEEE Conference on Emerging Technologies Factory Automation*, pp. 1–4, 2012.

[44] T. Ahmed, M. Coates, and A. Lakhina, "Multivariate online anomaly detection using kernel recursive least squares," in *26th IEEE International Conference on Computer Communications*, pp. 625–633, 2007.

[45] P. Vincent and Y. Bengio, "Kernel matching pursuit," *Machine Learning*, vol. 48, no. 1-3, pp. 165–187, 2002.

[46] V. Guigue, A. Rakotomamonjy, and S. Canu, "Kernel basis pursuit," in *European Conference on Machine Learning*, pp. 146–157, october 2005.

[47] D. Needell and J. Tropp, "Cosamp: Iterative signal recovery from incomplete and inaccurate samples," *Applied and Computational Harmonic Analysis*, vol. 26, pp. 301–321, May 2009.

[48] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society (Series B)*, pp. 267–288, Jan 1996.

[49] S. Chen, D. Donoho, and M. Saunders, "Atomic decomposition by basis pursuit," *SIAM Journal on Scientific Computing*, vol. 20, pp. 33–61, Jan. 1998.

[50] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression," *The Annals of Statistics*, vol. 32, pp. 407–499, Apr. 2004.

[51] M. Mackey and L. Glass, "Oscillation and chaos in physiological control systems," *Journal of Science*, vol. 197, pp. 287–289, July 1977.

[52] D. Donoho and J. Johnstone, "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, no. 3, pp. 425–455, 1994.

[53] M. Martínez-Ramón, J. Arenas-Garcia, A. Navia-Vázquez, and A. Figueiras-Vidal, "An adaptive combination of adaptive filters for plant identification," in *IEEE International Conference on Digital Signal Processing*, vol. 2, pp. 1195–1198, 2002.

[54] M. Silva and V. Nascimento, "Improving the tracking capability of adaptive filters via convex combination," *IEEE Transactions on Signal Processing*, vol. 56, pp. 3137–3149, July 2008.