Contents lists available at ScienceDirect

Pattern Recognition Letters

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

Kernel subspace pursuit for sparse regression[☆,☆☆]

Jad Kabbara, Ioannis N. Psaromiligkos[*]

Department of Electrical and Computer Engineering, McGill University, 3480 University Street, Montreal, QC, H3A 0E9, Canada

A R T I C L E   I N F O

Article history:
Received 15 December 2014
Available online 22 October 2015

Keywords:
Kernel methods
Sparse function approximation
Regression
Subspace pursuit

A B S T R A C T

Recently, results from sparse approximation theory have been considered as a means to improve the generalization performance of kernel-based machine learning algorithms. In this paper, we present Kernel Subspace Pursuit (KSP), a new method for sparse non-linear regression. KSP is a low-complexity method that iteratively approximates target functions in the least-squares sense as a linear combination of a limited number of elements selected from a kernel-based dictionary. Unlike other kernel methods, by virtue of KSP's algorithmic design, the number of KSP iterations needed to reach the final solution does not depend on the number of basis functions used nor that of elements in the dictionary. We experimentally show that, in many scenarios involving learning synthetic and real data, KSP is less complex computationally and outperforms other kernel methods that solve the same problem, namely, Kernel Matching Pursuit and Kernel Basis Pursuit.

1. Introduction

For decades, non-linear regression models have been extensively studied in the area of statistics, econometrics and machine learning. Quite often these models involve a non-linear transformation of data into a high-dimensional space in which linear regression models are expected to be more accurate compared to the original space. Examples include Artificial Neural Networks [9], Decision Trees [9] and Support Vector Machines (SVMs) [3].

An important family of non-linear regression methods is kernel methods [19] that have received major attention in the past two decades, as they allowed non-linear versions of conventional linear supervised and unsupervised learning algorithms, yielding impressive regression performance. Using the kernel trick, interesting "kernelized" extensions of many well-known algorithms were presented, including kernel SVMs [19], kernel Principle Component Analysis (PCA) [18] and kernel Fisher discriminant analysis [11].

The generalization, or "out-of-sample," performance of a learning method, including kernel methods considered in this work, quantifies its ability to predict on independent never-seen-before data. From a learning-theoretic perspective, controlling the "capacity" of learning algorithms is necessary to guarantee good generalization performance [23]. This in turn is related to the issue of overfitting: the more complex a model is, the more likely it is to overfit the data. Traditional non-parametric kernel regression methods such as the Nadaraya-Watson method [12,25] affect generalization performance through tuning parameters (such as the kernel bandwidth), which could be viewed as a way of controlling model complexity. Recently, results from sparse approximation theory [7] have been considered as another means to directly control the model complexity and, consequently, limit overfitting. Sparse approximation refers to estimating a vector (or function) as a linear combination of a small number of elements selected from a larger set, called dictionary, of vectors (or functions). In our regression context, we can control model complexity by restricting the regression function to be sparse, i.e., that it is a linear combination of a fixed (but small) number of functions selected from a given dictionary.

1.1. Related work

Previous work has discussed interesting connections between non-linear kernel-based learning and pursuit algorithms [10]. Pursuit algorithms, are a family of greedy, iterative approaches to obtain sparse approximations of a function. Poggio and Girosi relate in their work [15] the basis pursuit algorithm [2] to kernel SVMs. The work of Smola and Schölkopf [20] presents ties between the Matching Pursuit (MP) algorithm [10] and kernel PCA, and shows how such ties can be used to compress the kernel matrix in SVMs to allow dealing with large datasets. Also, Smola and Bartlett present in [21] a greedy MP-like technique that approximates Maximum A Posteriori (MAP) estimates of Gaussian Processes by expressing the MAP estimate as an expansion in terms of a small subset of pre-specified kernel functions.

[☆] This paper has been recommended for acceptance by G. Moser.

[☆☆] This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC), grant RGPIN 262017.

[*] Corresponding author. Tel.: +1 514 398 2465; fax: +1 514 398 4470.
E-mail addresses: jad.kabbara@mail.mcgill.ca (J. Kabbara), yannis@ece.mcgill.ca (I. N. Psaromiligkos).

http://dx.doi.org/10.1016/j.patrec.2015.09.018
0167-8655/© 2015 Elsevier B.V. All rights reserved.

Following this previous work, Kernel Matching Pursuit (KMP) [24] and Kernel Basis Pursuit (KBP) [8] were presented as kernel methods that learn target functions by means of sparse approximation. KMP adopts a greedy suboptimal iterative approach to construct a sparse linear approximation of the target regression function. It starts with the approximation being initialized to zero, and then builds it by adding to it, at each iteration, a new term consisting of an appropriately weighted function from the dictionary. The function is chosen according to a correlation-based criterion and then, the corresponding weight is computed so that the approximation error at that iteration is minimized. KBP, on the other hand, solves a relaxed version of the same problem by incorporating ℓ_1-regularization on the minimization of the approximation error. In doing so, KBP controls the sparsity of the solution. The problem formulation in KBP corresponds to the well-known Least Absolute Shrinkage and Selection Operator (LASSO) formulation [22] in the feature space which combines an ℓ_2-loss function (squared error) with ℓ_1-regularization. While previous work (e.g., [2] that inspired KBP) considered finding the optimal solution to the minimization problem through costly and complex linear programming techniques, KBP uses the Least Angle Regression (LARS) technique [6] which also finds the exact solution of the LASSO but in an iterative and efficient way.

In summary, KBP and KMP attempt to solve similar problems while addressing the sparsity of the solution in different ways. Indeed, KMP guarantees that the solution is K-sparse by imposing a pre-specified finite number K of basis functions that will be used to construct the approximation function. On the other hand, KBP, in its original formulation, uses a regularization term that controls the sparsity of the solution. In addition, both KMP and KBP suffer, computationally, from the same drawback: the number of iterations that they have to run directly depends on the intended number of basis functions in the final solution.

Other work in the literature has addressed similar problems. In [16], a stochastic version of KMP is presented for large datasets. In this sub-optimal version of KMP, at a given iteration, the search space from which basis functions are selected is reduced. More specifically, a basis function is selected from a randomly chosen subset of the available basis functions. The work in [13] presents a family of greedy algorithms for building sparse kernel-based regression and classification models. An ℓ_2-loss function is iteratively minimized until a specified stopping criterion is satisfied. Different greedy criteria for basis selection from the literature are discussed and two numerical schemes are presented for updating the weights and residue (approximation error), the first based on residual minimization and the other based on QR factorization.

1.2. Main contribution

Our research effort aims at identifying an alternative framework to KMP and KBP, one that would: (1) address the need to dissociate (to the extent possible) kernel-based learning algorithms from the computational constraints from which KMP and KBP suffer (i.e., the dependency of the number of iterations on the number of basis functions), and (2) still provide us with the control of the sparsity of the solution. Thus, we apply the Subspace Pursuit (SP) algorithm of Dai et al. [4] to non-linear regression problems, and introduce the Kernel SP (KSP) algorithm. SP was first introduced in the context of compressive sensing. It was originally proposed as an iterative method for the reconstruction of an unknown sparse signal from a set of linear measurements. In our work, we capitalize on the fact that SP is, in essence, a low complexity method to obtain least-squares solutions with a pre-specified sparsity level.

As in KMP and KBP, the proposed algorithm iteratively learns a regression function with a predefined sparsity level. In contrast to both KMP and KBP that start by initializing the regression function to zero, and then iteratively expand it until it reaches the desired sparsity level, KSP always maintains an estimate of the regression function built using the pre-specified number K of dictionary elements, and refines the estimate through a usually limited number of iterations. To build the regression function, KMP and KBP (using the LARS implementation) need a number of iterations equal to the desired number of basis functions in the expansion. However, the number of KSP iterations needed to reach the final solution does not depend on the required number of basis functions and is normally smaller than in KMP and KBP. We experimentally show that in various scenarios that involve learning synthetic and real data, this required number of KSP iterations is indeed much smaller than that required for KMP and KBP, and that in many of these scenarios, KSP is less computationally intensive than KMP and KBP. We further present experimental validation that shows that, in most of these learning scenarios, KSP outperforms both KMP and KBP in the task of learning real and synthetic data.

The remainder of this paper is organized as follows: In Section 2, we formulate the problem considered in this work. In Section 3, we introduce the Kernel Subspace Pursuit algorithm. In Section 4, we compare the computational overhead for running KSP, KMP and KBP in various scenarios involving learning synthetic and real data. We also present the results of simulations showing that our algorithm outperforms both KMP and KBP in most of these learning scenarios. Finally, Section 5 concludes the paper.

2. Problem formulation

We are given L noisy observations $\{y_1, \dots, y_L\}$ of an unknown target function $f : \mathbb{R}^d \mapsto \mathbb{R}$ at the inputs $\{\mathbf{x}_1, \dots, \mathbf{x}_L\}$ where $\mathbf{x}_i \in \mathbb{R}^d$ (for a given d) and $y_i \in \mathbb{R}, \forall i$. Let $k : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ be a positive definite kernel and let $\mathcal{H}$ be the associated kernel Hilbert space whose norm is denoted by $\| \cdot \|^2_{\mathcal{H}}$. We are interested in identifying a function $\hat{f} \in \mathcal{H}$ that is a good (in some sense) approximation of f. According to the Representer Theorem [17], given a strictly increasing function $\Omega : [0, \infty) \mapsto \mathbb{R}$ and an arbitrary cost function $c : (\mathbb{R}^d \times \mathbb{R}^2)^L \mapsto \mathbb{R} \cup \{\infty\}$, any minimizer $\hat{f} \in \mathcal{H}$ of the regularized cost $c((\mathbf{x}_1, y_1, \hat{f}(\mathbf{x}_1)), \dots, (\mathbf{x}_L, y_L, \hat{f}(\mathbf{x}_L))) + \Omega(\|\hat{f}\|^2_{\mathcal{H}})$ has the form

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{L} \alpha_i k(\mathbf{x}, \mathbf{x}_i), \quad \alpha_i \in \mathbb{R}. \tag{1}$$

That is, $\hat{f}$ can be written as a linear combination of the elements of the set $\mathcal{G}$ containing L functions in $\mathcal{H}$:

$$\mathcal{G} = \{g_i = k(\cdot, \mathbf{x}_i) | i = 1, \dots, L\} \subset \mathcal{H}. \tag{2}$$

Borrowing terminology from Sparse Approximation theory [7], we refer to $\mathcal{G}$ as the dictionary and to its elements as atoms.

In this paper, the goal is to approximate f using a given number $K < L$ of dictionary atoms, called basis functions, that is, we want our solution to admit the form in (1) but we now add the constraint that only K of the coefficients α_i are non-zero. In other words, we are interested in constructing an approximation $\hat{f}_K$ of f as a linear combination of K atoms $g_{\gamma_i}, i = 1, \dots, K$:

$$\hat{f}_K(\mathbf{x}) = \sum_{i=1}^{K} \alpha_i g_{\gamma_i}(\mathbf{x}) = \sum_{i=1}^{K} \alpha_i k(\mathbf{x}, \mathbf{x}_{\gamma_i}) \tag{3}$$

where $\{\gamma_1, \dots, \gamma_K\}$ are the indices of the selected atoms and $\alpha_1, \dots, \alpha_K$ are the corresponding coefficients. Given that K is smaller than L, we talk of a sparse approximation of f.

Let $\hat{\mathbf{y}}$ be the vector consisting of the evaluation of $\hat{f}_K$ at the L input vectors, i.e., $\hat{\mathbf{y}} = [\hat{f}_K(\mathbf{x}_1), \dots, \hat{f}_K(\mathbf{x}_L)]^T$. We also define the residue as the approximation error between the target vector $\mathbf{y} = [y_1, \dots, y_L]^T$ and $\hat{\mathbf{y}}$, i.e.,

$$\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}}. \tag{4}$$

One way to choose the K functions $\{g_{\gamma_1}, \ldots, g_{\gamma_K}\}$ and the coefficients $\{\alpha_1, \ldots, \alpha_K\}$ is to minimize the squared norm of the residue

$$\|\mathbf{r}\|^2 = \|\mathbf{y} - \hat{\mathbf{y}}\|^2. \tag{5}$$

Let $\mathbf{G}$ be the $n \times n$ Gram matrix of k whose (i, j)th element is given by

$$[\mathbf{G}]_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j). \tag{6}$$

Then, the problem under consideration is equivalent to

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^L} \|\mathbf{y} - \mathbf{G}\boldsymbol{\alpha}\|^2 \quad \text{s.t.} \quad \boldsymbol{\alpha} \text{ is } K - \text{sparse}, \tag{7}$$

which can be more formally stated as:

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^L} \|\mathbf{y} - \mathbf{G}\boldsymbol{\alpha}\|^2 \quad \text{s.t.} \quad \|\boldsymbol{\alpha}\|_0 = K \tag{8}$$

where $\|\boldsymbol{\alpha}\|_0$ denotes the ℓ_0 norm, i.e., the number of non-zero elements of $\boldsymbol{\alpha}$.

Solving (8) involves finding the optimal set of functions $\{g_{\gamma_1}, \ldots, g_{\gamma_K}\}$ to use in forming $\hat{f}_K$ that, in turn, requires an exhaustive search over all $\frac{L!}{K!(L-K)!}$ possible choices of K basis functions among the L dictionary atoms. As carrying out such search would be computationally prohibitive, several suboptimal procedures such as KMP and KBP have been proposed.

3. Proposed method

3.1. Subspace pursuit

Subspace Pursuit (SP) was first introduced in [4] in the context of compressive sensing (see also [14] 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 (possibly noisy) observations, $\mathbf{v} \in \mathbb{R}^M$, of $\mathbf{u}$ obtained via $M < N$ linear measurements, that is,

$$\mathbf{v} = \boldsymbol{\Phi}\mathbf{u} \tag{9}$$

where $\boldsymbol{\Phi} \in \mathbb{R}^{M\times N}$ is referred to as the measurement or sampling matrix and $\mathbf{v}$ as the measurement vector. The unknown signal $\mathbf{u}$ is assumed to be K-sparse, i.e., it is assumed to have $K \ll N$ non-zero elements. In compressive sensing, sparsity encapsulates the idea that the signal contains far less information than what its actual dimension suggests and thus, it is possible to reconstruct $\mathbf{u}$ from its measurements $\mathbf{v}$ even though $K \ll N$.

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

$$\min_{\mathbf{u}\in\mathbb{R}^N} \|\mathbf{v} - \boldsymbol{\Phi}\mathbf{u}\|^2 \text{ s.t. } \|\mathbf{u}\|_0 = K. \tag{10}$$

Conceptually, the main task in SP is to identify which subspace, generated by K columns of the sampling matrix $\boldsymbol{\Phi}$, can provide the best least squares approximation of the measurement vector $\mathbf{v}$. To do so, SP iteratively tests carefully chosen subsets of columns of $\boldsymbol{\Phi}$; at each iteration, columns are removed or added to the subset aiming at reducing the least squares approximation error of $\mathbf{v}$.

3.2. Kernel subspace pursuit

In this section, we solve the problem presented in Section 2 by extending SP to non-linear regression problems and introduce Kernel SP (KSP). In KSP, we treat the sampling matrix $\boldsymbol{\Phi}$ as the Gram matrix $\mathbf{G}$. Consider again the setup introduced in Section 2. 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. Thus, our minimization problem can be rewritten as

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^K, T} \|\mathbf{G}_T\boldsymbol{\alpha} - \mathbf{y}\|^2 \text{ s.t. cardinality of } T \text{ is } K. \tag{11}$$

KSP solves (11) by initially selecting the K columns of $\mathbf{G}$ that exhibit the highest correlation with the vector $\mathbf{y}$, 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 $\mathbf{y}$, and discarding the less informative ones.

In the initialization step, we form a column set T_0 by selecting the K columns of $\mathbf{G}$ that exhibit the highest correlation with the vector $\mathbf{y}$, i.e., the columns that correspond to the largest-in-magnitude elements of $\mathbf{G}^T\mathbf{y}$. We then obtain an initial estimate $\hat{\mathbf{y}}_0$ of $\mathbf{y}$ using only the columns of $\mathbf{G}$ in T_0 by projecting the vector $\mathbf{y}$ onto the span of the columns of $\mathbf{G}_{T_0}$, that is,

$$\hat{\mathbf{y}}_0 = \text{proj}(\mathbf{y}, \mathbf{G}_{T_0}) := \mathbf{G}_{T_0}\mathbf{G}_{T_0}^{\dagger}\mathbf{y}, \tag{12}$$

where $\mathbf{G}_{T_0}^{\dagger}$ denotes the Moore-Penrose pseudo-inverse of $\mathbf{G}_{T_0}$. The initial residue is finally calculated as follows:

$$\mathbf{r}_0 = \mathbf{y} - \hat{\mathbf{y}}_0. \tag{13}$$

Subsequently, the following procedure is repeated until a stopping criterion is met. At a given iteration, say ℓ, we begin by computing the correlations between the columns of $\mathbf{G}$ and the residue from the previous iteration, and retaining the K columns of $\mathbf{G}$ that correspond to the largest correlations (in magnitude). This set of K columns is then merged with the set of columns used to obtain the previous iteration's approximation of $\mathbf{y}$. As a result, we now have a set containing up to $2K$ columns of $\mathbf{G}$, say $\tilde{T}_\ell$. These columns are the candidates among which we will select the K columns to be used to obtain the current iteration's estimate of $\mathbf{y}$.

The selection process proceeds as follows. We first project $\mathbf{y}$ onto the span of the columns of $\mathbf{G}$ in the merged set $\tilde{T}_\ell$ and we obtain a new set of indices, T_ℓ, by identifying the columns corresponding to the K largest-in-magnitude projection coefficients, i.e., to the K largest elements of the vector $\mathbf{G}_{\tilde{T}_\ell}^{\dagger}\mathbf{y}$. The LS estimate of $\mathbf{y}$ at the ℓth iteration, $\hat{\mathbf{y}}_\ell$, is calculated by projecting the vector $\mathbf{y}$ onto the span of the columns of $\mathbf{G}_{T_\ell}$:

$$\hat{\mathbf{y}}_\ell = \text{proj}(\mathbf{y}, \mathbf{G}_{T_\ell}). \tag{14}$$

and a new residue is obtained as $\mathbf{y} - \hat{\mathbf{y}}_\ell$. Finally, the residue computed at the current iteration is compared to the one from the previous iteration. If the current residue is larger or the maximum number $\ell_{\max}$ of iterations is reached, the refinement process is ended and the loop is exited.

Once the final column set T_ℓ has been identified, the K-sparse kernel regressor is obtained by

$$\hat{f}_K(\mathbf{x}) = \sum_{i=1}^{K} \alpha_i k(\mathbf{x}, \mathbf{x}_{\gamma_i}) \tag{15}$$

where γ_i denotes the ith element of T_ℓ and α_i is the corresponding coefficient, i.e., α_i is the ith element of the vector $\mathbf{G}_{T_\ell}^{\dagger}\mathbf{y}$.

Algorithm 1 summarizes KSP.

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 exist substantial differences in the reconstruction methods that they employ. Indeed, both KMP and KBP start by initializing 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 and KBP (using the LARS implementation) need a number of iterations equal to the desired number of basis functions in the expansion. KSP, on the other hand, always maintains an estimate of the

Algorithm 1 Kernel Subspace Pursuit Algorithm.

```
Input: $k$, $\{\mathbf{x}_1, \dots, \mathbf{x}_L, \mathbf{y}\}$, $K$.
Initialization:
    Compute $\mathbf{G}$ using (6).
    $T_0 = \{$Indices of the $K$ largest magnitude entries in the vector $\mathbf{G}^T\mathbf{y}\}$
    $\mathbf{r}_0 = \mathbf{y} - \mathrm{proj}(\mathbf{y}, \mathbf{G}_{T_0})$
Loop: At iteration $\ell$ ($1 \le \ell \le \ell_{\max}$), execute the following:
    Compute the vector $\mathbf{G}^T\mathbf{r}_{\ell-1}$
    Identify the set $T$ of indices of the $K$ largest magnitude entries in that vector
    Find the set of candidate columns $\tilde{T}_\ell = T_{\ell-1} \cup T$
    Compute the vector $\mathbf{G}^\dagger_{\tilde{T}_\ell}\mathbf{y}$
    Identify the set $T_\ell$ of indices of the $K$ largest magnitude entries in that vector
    Compute the residue $\mathbf{r}_\ell = \mathbf{y} - \mathrm{proj}(\mathbf{y}, \mathbf{G}_{T_\ell})$
    If $||\mathbf{r}_\ell||_2 > ||\mathbf{r}_{\ell-1}||_2$, let $T_\ell = T_{\ell-1}$ and exit the loop.
Output: $K$-sparse kernel regressor $\hat{f}_K(\mathbf{x})$ (obtained using (15))
```

regression function built using the pre-specified number K 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 and KBP.

Furthermore, as we stated in Section 1.1, KMP employs a greedy iterative approach in constructing the approximation function. At each iteration, one atom is selected from the dictionary and appended to the expansion. When such a selection is made at a specific iteration, it cannot be reverted in later iterations as KMP lacks the capability to reassess later whether a selected atom could have been replaced with one that is more informative. KSP does not suffer from this deficiency as with each iteration, any selected atom can be dynamically suppressed if it were determined that it is less informative – at that particular iteration – than K other atoms.

In the following section, we present simulation results that show that KSP outperforms both KMP and KBP in learning non-linear functions as well as real data. 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 in many learning scenarios.

4. Simulations

In this section, we experimentally test KSP and compare its performance to that of KMP and KBP[1].

4.1. Synthetic data

In this experiment, we test KSP in learning different non-linear functions. Our aim is to learn a function $g(x)$ by observing L noisy samples:

$$g(\mathbf{x}_i) + \mathcal{N}(0, \sigma^2_{\mathcal{N}}), i = 1, \dots, L \tag{16}$$

obtained at known input points $\mathbf{x}_i \in \mathbb{R}$, $i = 1, \dots, L$. In (16), $\mathcal{N}(0, \sigma^2_{\mathcal{N}})$ represents Gaussian noise with mean 0 and variance $\sigma^2_{\mathcal{N}}$.

In this experiment, we consider the following functions:

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

[1] KBP was implemented using the vanilla version of the LARS algorithm (Section 2 in [6]) stopped after K iterations.

Table 1
MSE performance of KSP, KMP and KBP in learning synthetic data using a Gaussian kernel. For each entry, the first row presents the error alongside the corresponding error margin while the second row shows the values (K, σ) used in each case.

	Algorithms		
Functions	KSP	KMP	KBP
$\cos(\exp(\mathbf{x}))$	0.00214 ± 0.00096	$\mathbf{0.00212 \pm 0.00070}$	0.049 ± 0.012
	(80,0.9)	(80,0.5)	(20,0.1)
$\sin(\exp(\mathbf{x}))$	0.00255 ± 0.00105	$\mathbf{0.00216 \pm 0.00088}$	0.7911 ± 0.0197
	(60,0.9)	(70,0.5)	(20,0.01)
$\tanh(\mathbf{x})$	$\mathbf{0.00291 \pm 0.00112}$	0.00369 ± 0.00111	0.2223 ± 0.0151
	(48,0.6)	(36,0.3)	(42,0.01)
$\tan(\mathbf{x})$	$\mathbf{0.00348 \pm 0.00133}$	0.00658 ± 0.00153	0.3107 ± 0.1050
	(60,0.6)	(62,0.3)	(26,0.06)
Heavisine	$\mathbf{0.06323 \pm 0.0183}$	0.11921 ± 0.01790	9.5013 ± 0.565
	(44,0.1)	(92,0.1)	(24,0.6)
Doppler	$\mathbf{0.0235 \pm 0.0047}$	0.04025 ± 0.00489	0.0881 ± 0.00679
	(66,0.1)	(70,0.1)	(86,0.8)
Blocks	$\mathbf{1.4258 \pm 0.25}$	1.6615 ± 0.235	5.8548 ± 0.693
	(40,0.1)	(94,0.1)	(40,0.2)

We also test our method using synthetic data described by Donoho and Johnstone [5]. These data were generated using the following functions:

1. Heavisine function: $g(x) = 4\sin(4\pi x) - \mathrm{sgn}(x - 0.3) - \mathrm{sgn}(0.72 - x)$,
2. Doppler function: $g(x) = [x(1-x)]^{0.5}\sin(2\pi\frac{1+\epsilon}{x+\epsilon})$ with $\epsilon = 0.5$,
3. Blocks function: $g(x) = \sum_{j=1}^{11} \mathbf{a}_j p(x - \mathbf{b}_j)$ where:
   - $p(x) = \{1 + \mathrm{sgn}((x)\}/2$,
   - $\mathbf{a} = [4, -5, 3, -4, 5, -4.2, 2.1, 4.3, -3.1, 2.1, -4.2]$,
   - $\mathbf{b} = \frac{1}{100}[10, 13, 15, 23, 25, 40, 44, 65, 76, 78, 81]$.

The inputs $\mathbf{x}_i \in \mathbb{R}$ are drawn i.i.d. from the interval [0, 1] according to a uniform distribution. Each input is then passed through one of the non-linear functions mentioned above. The result is finally corrupted with white Gaussian noise with variance $\sigma^2_{\mathcal{N}} = 0.15$. We use a training set of size $L = 400$ and a test set of size 200. For each experiment, we optimize, for each algorithm, the kernel width σ and the sparsity K (number of basis functions in the final solution) using 5-fold cross-validation. The results, averaged out over 50 Monte-Carlo simulations, are shown in Table 1 where we also specify the values for K and σ used in each case. The error figures are shown along with the corresponding error margin. For all cases, we use $\ell_{\max} = 5$.[2] Results show that KSP outperforms both KMP and KBP in learning 5 of the 7 functions shown in Table 1 while being outperformed in the first two experiments by only one algorithm, namely, KMP.

Next, we repeat the first experiment but instead of a Gaussian kernel, we use a polynomial kernel given by: $k(\mathbf{u}, \mathbf{v}) = (\mathbf{uv} + 1)^2$. Again, for each experiment, we optimize, for each algorithm, the kernel width σ and the sparsity K using 5-fold cross-validation. The results, averaged out over 50 Monte-Carlo simulations, are shown in Table 2. The error figures are shown along with the corresponding error margin. Results show that KSP outperforms KMP and KBP in learning 6 of the 7 functions, with KSP being outperformed in the first experiment by one algorithm only, namely, KMP.

4.2. Real data

In this section, we test KSP using real data obtained from three regression datasets available in the UCI machine learning repository [1]. In the first dataset, the task consists in predicting the prices of houses in suburbs of Boston based on different demographic, economic and social factors. In this dataset, the first 800 samples are used as a training set and the next 200 samples are used as a test set. In the second

[2] See Section 4.3 for an important observation related to $\ell_{\max}$.

Table 2
MSE performance of KSP, KMP and KBP in learning synthetic data using a polynomial kernel. For each entry, the error is shown alongside the corresponding error margin and corresponding value for K in parenthesis.

	Algorithms		
Functions	KSP	KMP	KBP
$\cos(\exp(\mathbf{x}))$	0.00116 ± 0.000872	**$0.00088 \pm$ 0.00073**	0.24158 ± 0.141
	(40)	(58)	(80)
$\sin(\exp(\mathbf{x}))$	**0.00173 ± 0.00100**	0.011966 ± 0.00134	0.81667 ± 0.05910
	(20)	(24)	(76)
$\tanh(\mathbf{x})$	**0.00112 ± 0.00090**	0.00267 ± 0.00079	0.54034 ± 0.08680
	(30)	(36)	(70)
$\tan(\mathbf{x})$	**0.00151 ± 0.00076**	0.00213 ± 0.00084	0.46754 ± 0.07472
	(34)	(44)	(76)
Heavisine	**6.93244 ± 0.528**	7.72686 ± 0.62700	8.57339 ± 0.601
	(20)	(20)	(20)
Doppler	**0.07833 ± 0.00579**	0.07937 ± 0.00589	0.08588 ± 0.00668
	(50)	(50)	(138)
Blocks	**3.27133 ± 0.295**	3.65594 ± 0.304	6.07591 ± 0.576
	(20)	(30)	(56)

Table 3
MSE performance of KSP, KMP and KBP in learning real data using a Gaussian kernel. For each entry, the first row presents the error while the second row shows the values (K, σ) used in each case.

	Algorithms		
Data	KSP	KMP	KBP
Housing	**106.0016**	119.4577	268.31759
	(288,14.5)	(250,20)	(300,4)
Abalone	**0.05389**	0.0626	8.8319
	(500,0.7)	(540,0.6)	(450,0.2)
Stocks	**0.00002**	0.00003	0.00199
	(94,0.1)	(72,0.1)	(100,4.9)

dataset, the task consists in predicting the age of Abalone species (a type of sea snails) based on various physical measurements. In this dataset, the first 400 samples are used as a training set and the next 100 samples are used as a test set. In the last dataset, the task consists in predicting the values of Istanbul's stock exchange given different attributes. In this dataset, the first 800 samples are used as a training set and the next 100 samples are used as a test set. For all three datasets, we use $\ell_{\max} = 5$, and for each dataset, we optimize the kernel width σ and the sparsity K for each algorithm using 5-fold cross-validation. The results are presented in Table 3 and show that KSP outperforms both KMP and KBP in all of the 3 learning scenarios.

Next, we repeat the same experiment but instead of a Gaussian kernel, we use a polynomial kernel given by: $k(\mathbf{u}, \mathbf{v}) = \mathbf{uv} + 1$. Again, for each dataset, we optimize the kernel width σ and the sparsity K for each algorithm using 5-fold cross-validation. The results are presented in Table 4 and show that KSP outperforms both KMP and KBP in all of the learning scenarios.

4.3. Computational considerations

To understand better the computational burden of the three algorithms, KSP, KMP and KBP, we note that KMP and KBP are of order $O(L^2)$ per iteration. KSP is of order $O(L^2 + LK^2)$ per iteration. Regarding the number of iterations, recall that the number of iterations in KSP does not depend on K. The extent to which our algorithm depends on K is the computational cost per iteration. Therefore, KSP does not suffer from the more detrimental dependency that binds the number of iterations to K. In fact, our simulations showed that, for all learning scenarios presented in Section 4, increasing the maximum number, $\ell_{\max}$, of KSP iterations beyond 5 iterations (we considered values of $\ell_{\max}$ up to 50) led to a minimal (practically negligible) improvement in the MSE performance of KSP. Hence, the maximum number of KSP iterations that were executed in order to build the regression function was fixed at 5.

Table 4
MSE performance of KSP, KMP and KBP in learning real data using a polynomial kernel. For each entry, the error is shown alongside the corresponding value for K in parenthesis.

	Algorithms		
Data	KSP	KMP	KBP
Housing	**24.4105** (11)	69.5966.1 (30)	26.4025 (162)
Abalone	**9.6848** (50)	15.0165 (80)	71.70987 (50)
Stocks	**0.00002** (40)	0.00015 (50)	0.00015 (126)

Table 5
CPU running times for KSP, KMP and KBP in milliseconds.

	Algorithms		
Functions	KSP	KMP	KBP
$\cos(\exp(\mathbf{x}))$	146.64	287.04	**124.80**
$\sin(\exp(\mathbf{x}))$	**102.96**	254.28	115.44
$\tanh(\mathbf{x})$	**85.80**	124.80	235.56
$\tan(\mathbf{x})$	**96.72**	226.20	143.52
Heavisine	**38.22**	322.61	140.40
Doppler	**89.86**	242.74	447.41
Blocks	**32.45**	336.03	219.96
Housing	274.44	**171.93**	613.72
Abalone	2229.91	**1726.39**	3110.85
Stocks	52.53	**40.75**	157.09

We further investigate the computational burden ensuing from running KSP, KMP and KBP in different learning scenarios. We record the CPU time needed to run one Monte-Carlo iteration (i.e., one complete run) of each of the 3 algorithms in the 10 different learning scenarios that use the Gaussian kernel. All parameters are kept the same as in Section 4.1. The three algorithms were implemented in MATLAB, and all tests were run on an 8-GB, 2.00-GHz Intel Core i7 workstation running a 64-bit Windows 7 OS. We report the corresponding CPU running times (in seconds) for each of the three algorithms in Table 5 for the scenarios involving synthetic data (first 7 rows) and real data (last 3 rows).

Results show that despite the higher complexity-per-iteration of KSP, KSP's CPU running time is the smallest in 6 out of the 10 learning scenarios. This can be explained by the fact that, despite KSP's higher complexity per iteration, a maximum of 5 KSP iterations were needed (as explained earlier in the section) while the other two algorithms required a larger number of iterations (equal to the number of basis functions in the final solution).

4.4. Effect of *K* on KSP's performance

In this section, we investigate how KSP's MSE performance and CPU time vary with the increase of K. We report in Table 6

Table 6
Effect of K on KSP's MSE Performance and CPU Running Times (in milliseconds).

	K			
Functions	10	50	100	200
$\cos(\exp(\mathbf{x}))$	0.01029	0.01027	0.01014	0.01009
	12.00	54.00	124.80	577.20
$\sin(\exp(\mathbf{x}))$	0.01048	0.01041	0.01040	0.01039
	21.00	78.00	187.20	717.60
$\tanh(\mathbf{x})$	0.01012	0.00993	0.00989	0.00981
	13.20	79.00	249.60	639.60
$\tan(\mathbf{x})$	0.01053	0.01035	0.01032	0.01024
	17.40	62.40	253.60	811.21
Heavisine	0.6760	0.5870	0.5770	0.556
	4.16	61.9	241.00	836.00
Doppler	0.0865	0.0858	0.0853	0.0844
	9.88	60.3	264	852
Blocks	3.205	3.131	2.789	2.769
	10.80	54.60	204.36	683.60

the MSE performance and CPU running times for learning the 7 different synthetic data using a Gaussian kernel of width $\sigma = 3$ for $K = 10, 50, 100,$ and 200. The value in the first row in each cell represents the MSE while the value in the second row represents the CPU running time in milliseconds. It is interesting to note that for this choice of σ the MSE performance is not as sensitive to the choice of K as the CPU running time.

5. Summary and conclusion

In this paper, we presented a new kernel method that approximates target functions in the least-squares sense by means of sparse approximation. To approximate the target functions, KSP uses a linear combination of a finite number of elements selected from a kernel-based dictionary. KSP is an iterative approach that starts by initializing the subset of selected dictionary functions to be used in the approximation and then refines this subset throughout a limited number of iterations to minimize the approximation error. By specifying the number of basis functions, KSP is able to guarantee the sparsity of the solution, and thus offer improved generalization performance.

KSP belongs in the same family as KMP and KBP: they all solve the same regression problem and they share many attractive properties. For example, they all allow the dictionary of functions to include different types of kernels, and thus enable learning using mixtures of different kernels or kernels of the same type but having different parameters. Also, they are all based on pursuit methods but they differ on the exact pursuit method employed to enforce the sparsity constraint (SP, MP and BP). In a sense, our work completes the group of kernel regression methods that are based on pursuit algorithms, by introducing its most powerful member (both in terms of learning and computational performance). Therefore, part of this paper's contribution is to demonstrate for the first time that SP is the better pursuit method to address the problem at hand.

We presented 20 different scenarios involving learning different synthetic and real data. We experimentally showed that, in these scenarios, KSP requires a number of iterations that is much smaller than that required by KMP and KBP. As a result, KSP is less computationally intensive than KMP and KBP in many of those learning scenarios. We further presented experimental validation that shows that KSP outperforms KMP and KBP in the task of learning real and synthetic data in 17 out of different 20 learning scenarios.

References

[1] K. Bache, M. Lichman, UCI machine learning repository, 2013,http://archive.ics.uci.edu/ml.
[2] S. Chen, D. Donoho, M. Saunders, Atomic decomposition by basis pursuit, SIAM J. Sci. Comput. 20 (1) (1998) 33–61.
[3] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297.
[4] W. Dai, O. Milenkovic, Subspace pursuit for compressive sensing signal reconstruction, IEEE Trans. Inf. Theory 55 (5) (2009) 2230–2249, doi:10.1109/TIT.2009.2016006.
[5] D. Donoho, J. Johnstone, Ideal spatial adaptation by wavelet shrinkage, Biometrika 81 (3) (1994) 425–455.
[6] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani, Least angle regression, The Annals of Stat. 32 (2) (2004) 407–499.
[7] M. Elad, Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing, Springer, 2010.
[8] V. Guigue, A. Rakotomamonjy, S. Canu, Kernel basis pursuit, in: Proceedings of European Conference on Machine Learning, 2005, pp. 146–157.
[9] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, vol. 1, Springer New York, 2001.
[10] S.G. Mallat, Z. Zhang, Matching pursuits with time-frequency dictionaries, IEEE Trans. on Signal Process. 41 (12) (1993) 3397–3415.
[11] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, K. Muller, Fisher discriminant analysis with kernels, in: IEEE Signal Processing Society Workshop on Neural Networks for Signal Processing, 1999, pp. 41–48, doi:10.1109/NNSP.1999.788121.
[12] E.A. Nadaraya, On estimating regression, Theory of Probab. Its Appl. 9 (1) (1964) 141–142.
[13] P.B. Nair, A. Choudhury, A.J. Keane, Some greedy learning algorithms for sparse regression and classification with mercer kernels, J. Mach. Learn. Res. 3 (2002) 781–801.
[14] D. Needell, J. Tropp, CoSaMP: iterative signal recovery from incomplete and inaccurate samples, Appl. Comput. Harmonic Anal. 26 (3) (2009) 301–321.
[15] T. Poggio, F. Girosi, A sparse representation for function approximation, Neural Comput. 10 (6) (1998) 1445–1454.
[16] V. Popovici, S. Bengio, J.-P. Thiran, Kernel matching pursuit for large datasets, Pattern Recognit. 38 (12) (2005) 2385–2390.
[17] B. Schölkopf, R. Herbrich, A. Smola, A generalized representer theorem, in: Computational Learning Theory, 2001, pp. 416–426.
[18] B. Schölkopf, A. Smola, K.-R. Müller, Kernel principal component analysis, in: International Conference on Artificial Neural Networks, 1997, pp. 583–588.
[19] A. Smola, B. Schölkopf, Learning with Kernels, 1 edition, MIT Press, Cambridge, MA, 2002.
[20] A.J. Smola, B. Schökopf, Sparse greedy matrix approximation for machine learning, in: Proceedings of the Seventeenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc., 2000, pp. 911–918.
[21] A.J. Smola, P.L. Bartlett, Sparse greedy Gaussian process regression, in: Advances in Neural Information Processing Systems, 2001, pp. 619–625.
[22] R. Tibshirani, Regression shrinkage and selection via the lasso, J. R. Stat. Soc. (Series B) (1996) 267–288.
[23] V. Vapnik, Statistical Learning Theory, vol. 2, Wiley New York, 1998.
[24] P. Vincent, Y. Bengio, Kernel matching pursuit, Mach. Learn. 48 (1–3) (2002) 165–187.
[25] G.S. Watson, Smooth regression analysis, Sankhyā: The Indian J. Stat. Ser. A (1964) 359–372.