Kernel Polytope Faces Pursuit

Tom Diethe and Zakria Hussain

Department of Computer Science, University College London
{t.diethe,z.hussain}@cs.ucl.ac.uk

Abstract. Polytope Faces Pursuit (PFP) is a greedy algorithm that approximates the sparse solutions recovered by ℓ_1 regularised least-squares (Lasso) [4,10] in a similar vein to (Orthogonal) Matching Pursuit (OMP) [16]. The algorithm is based on the geometry of the polar polytope where at each step a basis function is chosen by finding the maximal vertex using a path-following method. The algorithmic complexity is of a similar order to OMP whilst being able to solve problems known to be hard for (O)MP. Matching Pursuit was extended to build kernel-based solutions to machine learning problems, resulting in the sparse regression algorithm, Kernel Matching Pursuit (KMP) [17]. We develop a new algorithm to build sparse kernel-based solutions using PFP, which we call Kernel Polytope Faces Pursuit (KPFP). We show the usefulness of this algorithm by providing a generalisation error bound [7] that takes into account a natural regression loss and experimental results on several benchmark datasets.

Keywords: Polytope Faces Pursuit, Orthogonal Matching Pursuit, Pseudo-dimension, Sample Compression Bounds, Regression, Kernel methods.

1 Introduction

Estimating a sparsely represented function from a set of training examples is a classical problem in regression. Sparsity of representation is an important issue, for reasons of computational efficiency and for its influence on generalisation performance [5,6]. Suppose we are given a sequence of observations $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_m], \mathbf{x}_i \in \mathbb{R}^n$ with corresponding outputs $\mathbf{y} \in \mathbb{R}$. We would like to find a sparse set of weights $\mathbf{w} \in \mathbb{R}^n$ such that the regression loss (e.g. $\|\mathbf{y} - \mathbf{X}'\mathbf{w}\|_2^2$) is minimised. It is theoretically possible to directly enforce sparsity through the ℓ_0 optimisation problem

$$\min_{\mathbf{w}} \|\mathbf{w}\|_0 \qquad (1)$$
$$\text{s.t.} \quad \mathbf{y} = \mathbf{X}'\mathbf{w}$$

Finding this ℓ_0 solution is known to be $NP-$hard. However the equivalent ℓ_1 optimisation problem

$$\min_{\mathbf{w}} \|\mathbf{w}\|_1 \qquad (2)$$
$$\text{s.t.} \quad \mathbf{y} = \mathbf{X}'\mathbf{w}$$

is a convex optimisation problem and can be solved using general purpose solvers. A reformulation of this that directly minimises the regression loss is the Least Absolute Shrinkage and Selection Operator (LASSO) [15], which is given by

$$\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}'\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_1, \tag{3}$$

i.e. a form of ℓ_1-penalised least squares. Basis pursuit [3] uses the Least Angle Regression Solver (LARS) to solve the LASSO problem. The algorithm requires the computation of the full regularisation path through the LARS. However, for large scale problems, this optimisation becomes inefficient and in some cases intractible.

Matching Pursuit was proposed in the signal-processing community as an algorithm that decomposes any signal into a linear expansion of waveforms that are selected from a redundant dictionary of functions [8]. It is a general, greedy, sparse function approximation scheme with the squared error loss, which iteratively adds new functions (i.e. basis functions) to the linear expansion.

If we take as dictionary of functions of the form $K(\cdot, x_i)$ where x_i is the input part of a training example, then the linear expansion has essentially the same form as a Support Vector Machine. Matching Pursuit and its variants were developed primarily in the signal-processing and wavelets community, but there are many interesting links with the research on kernel-based learning algorithms developed in the machine learning community.

Kernel Matching Pursuit (KMP) [17] was developed as a method to estimate a function from training examples in the presence of noise in the context of a Reproducing Kernel Hilbert Space (RKHS). The general idea is to decompose the function to learn on a sparse-optimal set of spanning functions. Unlike Basis Pursuit (BP), which finds the exact ℓ_1 solution, the implementation does not rely on the LASSO formulation or the LARS. We are then effectively finding an approximation to the regression problem in the RKHS defined by the kernel function,

$$\min_{\boldsymbol{\alpha}} \|\boldsymbol{\alpha}\|_1 \tag{4}$$
$$\text{s.t.} \quad \mathbf{y} = \mathbf{K}\boldsymbol{\alpha}$$

where $\boldsymbol{\alpha}$ resembles the dual weight vector found by other kernel methods such as Kernel Ridge Regression [12].

More recently, connections have been made between Matching Pursuit, Kernel-PCA, Sparse Kernel Feature analysis, and how greedy algorithms of this kind can be used to compress the design matrix in SVMs to allow handling of very large data sets [14,13].

Further investigation of the criteria under which ℓ_0/ℓ_1 equivalence holds led to consideration of the $d-$dimensional *polytope* (the $d-$dimensional generalisation of a polygon) [4]. Using this geometric interpretation, a greedy algorithm called Polytope Faces Pursuit (PFP) has been proposed [9] which adopts a path-following approach through the relative interior faces of the polar polytope. In order to

generalise this to its kernelised form, we begin by converting (4) into its standard form

$$\min_{\boldsymbol{\alpha}} \|\tilde{\boldsymbol{\alpha}}\|_1 \tag{5}$$
$$\text{s.t.} \quad \mathbf{y} = \tilde{\mathbf{K}}\tilde{\boldsymbol{\alpha}}, \quad \tilde{\boldsymbol{\alpha}} \geq 0$$

where $\tilde{\mathbf{K}} = [\mathbf{K}, -\mathbf{K}]$ and $\tilde{\boldsymbol{\alpha}}$ has $2m$ nonnegative components, with the standard weight vector recoverable by $\alpha_i = \tilde{\boldsymbol{\alpha}}_i - \tilde{\boldsymbol{\alpha}}_{i+m}$ [2]. The corresponding *dual* of this linear program is

$$\max_{\mathbf{c}} \mathbf{y}'\mathbf{c} \tag{6}$$
$$\text{s.t.} \quad \tilde{\mathbf{K}}'\mathbf{c} \leq 1$$

which has an optimal dual weight vector $\mathbf{c}$ which coincides with the optimum $\boldsymbol{\alpha}$ of the primal formulation. Note that in the sense of kernel methods, the formulation (5) is already in the dual space, so the weight vector $\mathbf{c}$ is in fact the dual of the dual. The greedy approach to the solution of (6) then follows the same approach as taken in standard PFP [9]. This will be described further in section 2.2

We present experimental results on real world datasets, which show that KPFP is competitive with the KMP, Kernel Ridge Regression (KRR) and the LARS solver for LASSO on datasets derived from the UCI and STATLOG repositories.

2   Kernel Polytope Faces Pursuit

2.1   Preliminaries

Assume we have a sample S containing examples as paired inputs $\mathbf{x} \in \mathbb{R}^n$ and outputs $y \in \mathbb{R}$. Let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_m)'$ be the input vectors stored in matrix $\mathbf{X}$ as row vectors, where $'$ denotes the transpose of vectors or matrices. For simplicity we always assume that the examples are already projected into the kernel defined feature space, so that the kernel matrix $\mathbf{K}$ has entries $\mathbf{K}[i,j] = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$. In the analysis section we will explicitly denote the feature map $\phi(\mathbf{x})$ for some vector $\mathbf{x}$. The notation $\mathbf{K}[:, i]$ will denote the ith column of the matrix $\mathbf{K}$. When given a set of indices $\mathbf{i} = \{i_1, \ldots, i_k\}$ (say) then $\mathbf{K}[\mathbf{i}, \mathbf{i}]$ denotes the square matrix defined solely by the index set $\mathbf{i}$.

For analysis purposes we assume that the training examples are generated i.i.d. according to an unknown but fixed probability distribution that also governs the generation of the test data. Expectation over the training examples (empirical average) is denoted by $\hat{\mathbb{E}}[\cdot]$, while expectation with respect to the underlying distribution is denoted $\mathbb{E}[\cdot]$.

2.2   Algorithm

We now derive the Kernel Polytope Faces Pursuit (KPFP) algorithm, which is a generalisation of Polytope Faces Pursuit (PFP) to Reproducing Kernel Hilbert Spaces (RKHS).

<div align="center">**Table 1.** Notation</div>

$\mathbf{X}$	Input matrix
$\mathbf{y}$	Output vector
$\hat{\mathbf{y}}$	Estimates of outputs
$\mathbf{K}$	Kernel matrix with entries $\mathbf{K}_{i,j} =< \phi(x_i), \phi(x_j) >$
$\tilde{\mathbf{K}}$	$[\mathbf{K}, -\mathbf{K}]$
$\boldsymbol{\alpha}$	Dual (sparse) weight vector
$\mathbf{r}$	Vector of residuals
$\mathbf{c}$	Double-dual weight vector
k_{max}	Sparsity parameter
$\mathbf{A}^{\dagger}$	pseudo-inverse of matrix $\mathbf{A}$
$\mathbf{0}, \mathbf{1}$	Vector of zeroes and ones respectively
$\mathbf{I}$	Identity matrix

At each step the approach to the solution of this problem is to identify the optimal vertex which is the maximiser of $\mathbf{y}'\mathbf{c}$, which is similar to the way in which KMP builds up its solution. However the difference is that at each step, the path is constrained on the polytope face F given by the vertex of the previous step. This is achieved by projecting $\mathbf{y}$ into a subspace parallel to F to give $\mathbf{r} = (\mathbf{I} - \mathbf{Q})\mathbf{y}$ where $\mathbf{Q} = \frac{\mathbf{K}[:,i]\mathbf{K}[:,i]'}{|\mathbf{K}[:,i]|^2}$. Since $\boldsymbol{\alpha} = \mathbf{K}[:,i]'^{\dagger}\mathbf{y}$ and $\hat{y} = \mathbf{K}[:,i]\boldsymbol{\alpha}$ we have $\mathbf{r} = \mathbf{y} - \mathbf{K}[:,i]\boldsymbol{\alpha} = \mathbf{y} - \hat{\mathbf{y}}$ meaning that $\mathbf{r}$ is the residual from the approximation at step i.

The second step, which is where the main difference between (O)MP and PFP arises, involves projecting within the face F that has just been found, rather than from the origin. This is done by projecting along the residual $\mathbf{r}$. Therefore to find the next face at each step we find the maximum *scaled* correlation

$$\mathbf{i}_i = \arg \max_{i \notin \mathbf{i}} \tilde{\mathbf{K}}[:,i]'\mathbf{r}/(1 - \tilde{\mathbf{K}}[:,i]'\mathbf{c}) \tag{7}$$

where we only consider bases such that $\tilde{\mathbf{K}}[:,i]'\mathbf{r} > 0$.

We then proceed by removing any constraints that violate the condition that $\tilde{\boldsymbol{\alpha}}$ contains any negative entries. This is achieved by finding $j \in \mathbf{i}$ such that $\tilde{\boldsymbol{\alpha}}_j < 0$, removing j from $\mathbf{i}$ and removing the face from the current solution. We then recaluclate $\tilde{\boldsymbol{\alpha}}$ and continue until $\boldsymbol{\alpha}_j \geq 0, \forall j$. Although this step is necessary to provide exact solutions to (6), it may be desirable in some circumstances to remove this step due to the fact that our primal space is in fact the dual space of an RKHS. This would result in faster iterations but less sparse solutions. In section 3 we compare the performance of the algorithm with and without this step. The full algorithm is given in Algorithm 1.

2.3   Generalisation Error Bound

For the generalisation error bound we assume that the data are generated iid from a fixed but unknown probability distribution P over the joint space $\mathcal{X} \times \mathcal{Y}$. Given the *true error* of a function f:

$$\text{err}(f) = \mathbb{E}_{(x,y) \sim P} L(f(x), y),$$

Require: kernel $\mathbf{K}$, sparsity parameter $k > 0$, training outputs $\mathbf{y}$
1: Initialise $\tilde{\mathbf{K}} = [\mathbf{K}, -\mathbf{K}]$, $\tilde{\boldsymbol{\alpha}} = [\,]$, $\boldsymbol{\alpha} = [\,]$, $\hat{\mathbf{y}} = \mathbf{0}$, $\tilde{A} = [\,]$, $\mathbf{r} = \mathbf{y}$, $\mathbf{c} = \mathbf{0}$
2: **for** $i = 1$ to k **do**
3:     Find face $\mathbf{i}_i = \arg \max_{i \notin \mathbf{i}} \tilde{\mathbf{K}}[:, i]' \mathbf{r} / (1 - \tilde{\mathbf{K}}[:, i]' \mathbf{c})$ where $\tilde{\mathbf{K}}[:, i]' \mathbf{r} > 0$
4:     Add constraint: $\tilde{\mathbf{A}} = [\tilde{\mathbf{A}}, \tilde{\mathbf{K}}[:, \mathbf{i}_i]]$
5:     Update $\mathbf{B} = (\tilde{\mathbf{A}})^{\dagger}$, $\tilde{\boldsymbol{\alpha}} = \mathbf{B}\mathbf{y}$
6:     (Optional) Release violating constraints:
7:     **while** $\exists \, \tilde{\boldsymbol{\alpha}}_j < 0, \forall \, j$ **do**
8:         Remove face j: $\tilde{\mathbf{A}} = \tilde{\mathbf{A}} \smallsetminus \tilde{\mathbf{K}}[:, j]$, $\mathbf{i} = \mathbf{i} \smallsetminus \{j\}$
9:         Update $\mathbf{B} = \tilde{\mathbf{K}}[:, \mathbf{i}_i]^{\dagger}$, $\boldsymbol{\alpha} = \mathbf{B}\mathbf{y}$
10:    **end while**
11:    Set $\mathbf{c} = \mathbf{B}'\mathbf{1}$, $\hat{\mathbf{y}} = \tilde{\mathbf{A}}\tilde{\boldsymbol{\alpha}}$, $\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}}$
12: **end for**
13: Calculate $\boldsymbol{\alpha}_i = \tilde{\boldsymbol{\alpha}}_i + \tilde{\boldsymbol{\alpha}}_{i+m}$
Ensure: final set $\mathbf{i}$, (sparse) dual weight vector $\boldsymbol{\alpha}$, predicted outputs $\hat{y}$

Algorithm 1. Kernel Polytope Faces Pursuit

where $L(\hat{y}, y)$ is the loss between the predicted $\hat{y}$ and true y, the *empirical error* of f given S:

$$\mathrm{err}_S(f) = \frac{1}{m} \sum_{i=1}^{m} L_S(f(x_i), y_i)$$

and the estimation error $\mathrm{est}(f)$

$$\mathrm{est}(f) = |\mathrm{err}(f) - \mathrm{err}_S(f)|,$$

we would like to find an upper bound for $\mathrm{est}(f)$.

We use Theorem 17.1 from [1] (using the ℓ_∞ covering number as opposed to the ℓ_1 covering number) which states that:

$$P^m\{\exists f \in F : |\mathrm{err}(f) - \mathrm{err}_S(f)| \geq \epsilon\} \leq 4\mathcal{N}_\infty(\epsilon/16, F, 2m) \exp(-\epsilon^2 m/32),$$

where $\mathcal{N}_\infty(\epsilon, F, m)$ is the ℓ_∞ *covering number*. [1] This covering number can be upper bounded using Theorem 12.2 from [1]:

$$\mathcal{N}_\infty(\epsilon, F, m) \leq \left(\frac{emR}{\epsilon d}\right)^d,$$

where R is the support of the distribution and d denotes the *pseudo-dimension*. As with KMP [7] the KPFP also has VC-dimension (pseudo-dimension) k, when k is the number of basis vectors chosen. However, in contrast to the KMP bound of [7] we use the pseudo-dimension to apply a natural regression loss function, the so-called squared error:

$$L(f(x), y) = (f(x) - y)^2.$$

---

[1] Note that the $\ell_\infty \geq \ell_1$ covering number is always an upper bound on the ℓ_1 covering number.

Therefore there is no need to fix a bandwidth parameter as was the case with the bound of [7] *i.e.*, no need to map the regression loss into a classification one. We follow the proof technique of [7] but instead apply the sample compression technique over pseudo-dimension bounds, resulting in a slightly more involved proof.

Theorem 1. *Let $f \in F : \mathbf{X} \mapsto [0, 1]$ be the function output by any* sparse (dual) kernel regression *algorithm which builds regressors using basis vectors, m the size of the training set S and k the size of the chosen basis vectors* $\mathbf{i}$. *Let $\bar{S} = S \setminus S_{\mathbf{i}}$ denote the examples outside of the set $S_{\mathbf{i}}$. Assume without loss of generality that the last k examples in S form the set $S_{\mathbf{i}}$. Let R be the radius of the ball containing the support of S, then with $1 - \delta$ confidence we can upper bound the true error* $\mathrm{err}(f)$ *of function f given any training set S by,*

$$\mathrm{err}(f) \leq \mathrm{err}_{\bar{S}}(f) +$$
$$\frac{\sqrt{32^2 + 128(m-k)\left(k \ln \frac{em}{k} + k \ln 32e(m-k)R + 1 + \ln \frac{4km}{\delta}\right)} - 32}{2(m-k)}.$$

Proof. First consider a fixed k for the indices $\mathbf{i}$. Assume that the first $m - k$ points from S are drawn independently and apply Theorem 17.1 (and Theorem 12.2) from [1] to obtain the bound

$$P^m\{\bar{S} : |\mathrm{err}(f) - \mathrm{err}_{\bar{S}}(f)| \geq \epsilon\} \leq 4 \left(\frac{32e(m-k)R}{\epsilon k}\right)^k \exp\left(\frac{-\epsilon^2(m-k)}{32}\right). \quad (8)$$

Given that we would like to choose k basis vectors from m choices we have $\binom{m}{k}$ different ways of selecting them. Multiplying the rhs of Equation 8 by $\binom{m}{k}$ and setting it equal to δ we get:

$$P^m\{S : \exists f \in \mathrm{span}\{S_{\mathbf{i}}\} \text{ s.t. } |\mathrm{err}(f) - \mathrm{err}_{\bar{S}}(f)| \geq \epsilon\}$$
$$\leq 4\binom{m}{k}\left(\frac{32e(m-k)R}{\epsilon k}\right)^k \exp\left(\frac{-\epsilon^2(m-k)}{32}\right). \quad (9)$$

Next by setting the rhs of Equation (9) to δ, taking logarithms and rearranging we get

$$\frac{\epsilon^2(m-k)}{32} = k \ln \frac{em}{k} + k \ln 32e(m-k)R - \ln \epsilon + \ln k + \ln \frac{4}{\delta}.$$

We would like to write this bound in terms of ϵ and use the following result [11] which states that for any $\alpha > 0$, $\ln \epsilon \leq \ln \frac{1}{\alpha} - 1 + \alpha \epsilon$. Substituting this result with $\alpha = 1$ (a smaller α can be used but would make the bound less neat) we get

$$\epsilon^2(m-k) = 32\left(k \ln \frac{em}{k} + k \ln 32e(m-k)R - \ln 1 + 1 - \epsilon + \ln k + \ln \frac{4}{\delta}\right),$$

which yields the following quadratic equation:

$$(m - k)\epsilon^2 + 32\epsilon - 32 \left(k \ln \frac{em}{k} + k \ln 32e(m - k)R + 1 + \ln \frac{4k}{\delta} \right) = 0.$$

Therefore, solving for ϵ gives the result[2] when we further apply the bound m times.   □

This bound can be specialised to a Gaussian kernel[3] that uses the mean squared error loss.

Corollary 1. *For a Gaussian kernel and using all the definitions from Theorem 1 we can upper bound the loss of KPFP by:*

$$\text{err}(f) \leq \frac{1}{m - k} \sum_{i=1}^{m-k} L_{\bar{S}}(f(x_i), y_i) +$$

$$\frac{\sqrt{32^2 + 128(m - k) \left(k \ln \frac{em}{k} + k \ln 32e(m - k) + 1 + \ln \frac{4km}{\delta} \right)} - 32}{2(m - k)}.$$

Remark 1. The consequences of Theorem 1 (and Corollary 1) is that although the pseudo-dimension can be infinite even in cases where learning is successful[4], we will generate a bound that is *always* finite. Also, this is the first bound for KMP and KPFP (proposed in this paper) to use *the natural regression loss* in order to upper bound generalisation error. The bound is naturally trading off empirical error with complexity – as the training error decreases the bound gets smaller, and as the number of basis vectors (complexity) increase the bound gets larger. A good trade-off is to find small training error whilst using a small number of basis vectors. Clearly, the KMP and KPFP algorithms try and optimise this trade-off, and the bound suggests that this will result in good generalisation.

It is quite obvious that the output of the function class $F : \mathbf{X} \mapsto [0, 1]$ is not bounded between 0 and 1 in most 'real world' regression scenarios. Therefore, we can give a more practically useful bound for a function class $F : \mathbf{X} \mapsto [-B, B]$ where the outputs are bounded in the range of $[-B, B] \in \mathbb{R}$.

Corollary 2. *Let $\|\mathbf{w}\|_2 \leq B \in \mathbb{R}$ and $\|\mathbf{x}_i\|_2 \leq 1, i = 1, \ldots, m$. Let $f \in F : \mathbf{X} \mapsto [-B, B]$ be the function output by any sparse (dual) kernel regression algorithm which builds regressors using basis vectors, m the size of the training set S and k the size of the chosen basis vectors $\mathbf{i}$. Let $\bar{S} = S \setminus S_{\mathbf{i}}$ denote the examples outside of the set $S_{\mathbf{i}}$. Assume without loss of generality that the last k examples in S form the set $S_{\mathbf{i}}$. Let R be the radius of the ball containing the support of S,*

---

[2] Only solve the quadratic equation for the positive quadrant.
[3] We use the Gaussian kernel in the experiments.
[4] Note that the pseudo-dimension is a generalisation of the VC-dimension and hence the same problems of infinite VC-dimension also apply to the pseudo-dimension.

then with $1 - \delta$ confidence we can upper bound the true error $\mathrm{err}(f)$ of function f given any training set S by,

$$\mathrm{err}(f) \leq \mathrm{err}_{\bar{S}}(f) +$$

$$2B\frac{\sqrt{32^2 + 128(m-k)\left(k\ln\frac{em}{k} + k\ln 32e(m-k)R + 1 + \ln\frac{4km}{\delta}\right)} - 32}{2(m-k)}.$$

Proof. Denote the function class $\tilde{F} = \left\{\frac{f \pm B}{2B} : f \in F\right\} : \mathbf{X} \mapsto [0,1]$. Therefore, given any function $\tilde{f} \in \tilde{F}$ Theorem 1 holds. Furthermore, for any function class $F : \mathbf{X} \mapsto [-B, B]$ we have:

$$2B\mathrm{err}(\tilde{f}) \leq 2B\mathrm{err}_{\bar{S}}(\tilde{f}) +$$

$$2B\frac{\sqrt{32^2 + 128(m-k)\left(k\ln\frac{em}{k} + k\ln 32e(m-k)R + 1 + \ln\frac{4km}{\delta}\right)} - 32}{2(m-k)},$$

which completes the proof when we make the substitutions $\mathrm{err}(f) = 2B\mathrm{err}(\tilde{f})$ and $\mathrm{err}_{\bar{S}}(f) = 2B\mathrm{err}_{\bar{S}}(\tilde{f})$. $\square$

3 Experiments

We present a comparison on 9 benchmark datasets derived from the UCI, StatLib, and Delve benchmark repositories. Details of the datasets are given in Table 2. We analyse the performance of KPFP, KMP, Kernel Ridge Regression (KRR) and LASSO using the Least Angle Regression Solver (LARS) using Radial Basis Function (RBF) kernels. We used 10 randomised splits into training and test sets. For each of the datasets we used cross-validation (c.v.) to select the optimal RBF kernel width parameter for KRR. We then used this kernel as input to the KMP, LARS and KPFP algorithms. For both KMP and KPFP the initial sparsity level k was set in training by a heuristic method to the lesser of 100 or the number of training examples. Since the train and test error curves for both KMP and KPFP tend to follow each other well, we used the index of minimum training error as the final sparsity value. The means and standard deviations of the generalisation error for each method and dataset are given in Table 3.

The results show that overall the sparse methods (KMP,KPFP,LARS) all perform better than KRR. It is interesting to compare the performance of KPFP with and without the release of violating constraints (KPFPv and KPFP respectively). KPFPv performs nearly as well as KMP on all datasets except for `cpusmall`, whilst requiring fewer bases in the final solutions. On the other hand, KPFPv results in solutions that are the least sparse of the three methods, but results in the lowest generalisation error. LARS which gives an exact solution to the LASSO problem performs the worst here, showing that the exact solution is not necessarily the optimal one for generelisation. The key to the performance of all of these methods is in selecting the appropriate stopping point k. This is

Table 2. Number of examples and dimensions of each of the 9 benchmark datasets

Dataset	# examples	# dimensions
abalone	4177	8
bodyfat	252	14
cpusmall	8192	12
housing	506	13
mpg	392	7
mg	1385	6
pyrim	74	27
space_ga	3107	6
triazines	186	60

Table 3. Mean MSE (μ) and standard deviations (σ) for 9 benchmark datasets for Kernel Ridge Regression (KRR), Kernel Matching Pursuit (KMP), LASSO using the Least Angle Regression Solver (LARS) and Kernel Polytope Faces Pursuit with and without violation release (KPFPv,KPFP). The total number of wins over all splits of the data for each algorithm is given in the last row.

Dataset	KRR		KMP			LARS			KPFPv			KPFP		
	μ	σ	μ	σ	k	μ	σ	k	μ	σ	k	μ	σ	k
abalone	8.70	1.79	5.70	2.56	49.2	21.64	28.80	5.4	6.07	1.16	7.3	**4.82**	0.24	37.7
bodyfat	0.00	0.00	0.00	0.00	49.1	0.01	0.02	5.7	0.00	0.00	30.1	**0.00**	0.00	129.7
cpusmall	216.35	64.04	15.66	2.51	24.0	519.06	95.45	10.3	69.97	2.51	13.4	**12.50**	1.51	54.2
housing	72.19	19.59	**21.93**	7.17	50.3	56.84	19.35	8.9	34.16	8.19	21.9	23.22	6.67	150.8
mpg	39.47	24.57	20.70	14.37	50.6	42.05	48.27	7.7	13.11	3.35	11.5	**10.98**	1.97	161.1
mg	0.04	0.01	**0.02**	0.00	49.0	0.11	0.19	4.4	0.02	0.00	7.6	0.02	0.00	48.7
pyrim	0.02	0.01	0.02	0.02	24.3	0.02	0.01	11.6	0.02	0.01	17.8	**0.01**	0.01	39.0
space_ga	0.03	0.01	0.02	0.00	49.9	0.05	0.05	4.8	0.02	0.00	6.0	**0.02**	0.00	38.2
triazines	0.02	0.01	0.03	0.02	50.9	0.02	0.00	11.3	**0.02**	0.00	34.4	0.02	0.00	109.7
wins	3		34			6			9			39		

quite difficult to achieve in KMP, as the algorithm tends to overfit quite quickly, and there is no obvious criteria for stopping. For example, if cross-validation were used to select k, the resulting value would be too low, as the number of bases would selected from a smaller validation set. In our experiments we found that by selecting an intial k through a heuristic method and then choosing the minimiser of the training error resulted in the best compromise. In KPFP and KPFPv the optimal value for k is more easily achieved, as the training and test error curves tend to follow each other quite well, and we also have an (optional) stopping parameter θ_{max}. In fact, the value of θ to which θ_{max} is compared also follows the error curves. We found that by taking the minimiser of θ as the number of bases was a reliable way of estimating k.
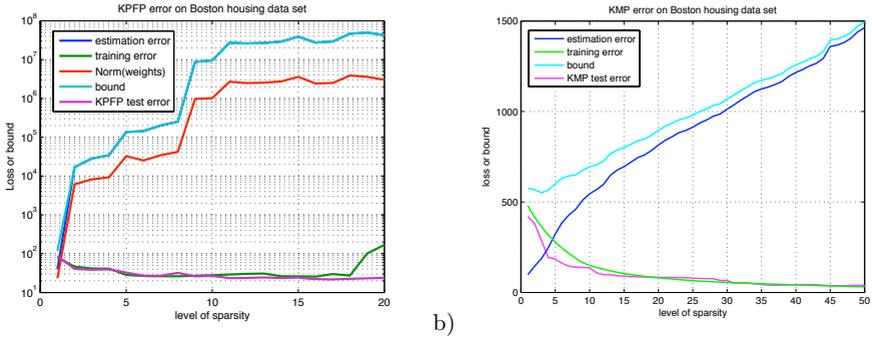
Fig. 1. a) Plot of generalisation error bound for different values of k using RBF kernels for the 'Boston housing' data set. The log of the generalisation error is shown on the y axis. The plot shows the empirical error of the set $\bar{S}$ (denoted training error, in green), the estimation error (in blue), the norm of the weight vector (in red), the bound value which is calculated from these three values (in cyan), and the generalisation (true) error (in magenta). Note that the empirical error follows the true error very well, which justifies its' use in the setting of the sparsity parameter. However the bound value is swamped by the norm of the weight vector (needed according to Corrollory 2), and as such is not useful. b) The bound values for the KMP algorithm. Note that in this case the bound (which is valid for this algorithm too) is more useful, simply because the norm of the weight vector does not blow up as quickly.

3.1   Bound Experiments

Finally we present results of the performance of the bound. Figure 3.1 shows typical plots of the bound. For Figure 3.1 (b) the number of training examples chosen were 450 and the number of test examples were 56, with the Gaussian width parameter set to $\sigma = 0.035$. The bound values tend to fall as basis vectors are added, before rising again as the complexity of the solution rises. Hence the first minimum of the bound value could serve as an appropriate point to stop the algorithm. This is clearly much more efficient than using cross-validation to select the value of k, the number of basis vectors to use. However in our experiments this resulted in stopping too early, resulting in underfitting. Further refinement of the bound may improve its performance in this respect.

4   Conclusions

Polytope Faces Pursuit (PFP) is a greedy algorithm that approximates the sparse solutions recovered by ℓ_1 regularised least-squares (LASSO) [4,10] in a similar vein to (Orthogonal) Matching Pursuit (OMP) [16]. The algorithm is based on the geometry of the polar polytope where at each step a basis function is chosen by finding the maximal vertex using a path-following method. The algorithmic complexity is of a similar order to OMP whilst being able to solve problems known to be hard for (O)MP.

We extended the PFP algorithm to a kernel verison, which we called Kernel Polytope Faces Pursuit (KPFP). We showed the utility of this algorithm by providing a novel generalisation error bound which used the natural regression loss and pseduo-dimension in order to upper bound its loss. The experimental results were also encouraging and showed that KPFP was competitive against the KMP and Kernel Ridge Regression.

A future research direction is to tighten the bound and use it to find the number of basis vectors during training.

Acknowledgements

References

1. Anthony, M., Bartlett, P.: Neural Network Learning: Theoretical Foundations. Cambridge University Press, Cambridge (1999)
2. Chen, S.: Basis Pursuit. PhD thesis, Department of Statistics, Stanford University (November 1995)
3. Chen, S.S., Donoho, D.L., Saunders, M.A.: Atomic decomposition by basis pursuit. SIAM Journal on Scientific Computing 20(1), 33–61 (1998)
4. Donoho, D.: Neighborly polytopes and sparse solution of underdetermined linear equations. Technical report, Department of Statistics, Stanford Univ., Stanford, CA (2005)
5. Floyd, S., Warmuth, M.: Sample compression, learnability, and the Vapnik-Chervonenkis dimension. Machine Learning 21(3), 269–304 (1995)
6. Graepel, T., Herbrich, R., Shawe-Taylor, J.: Generalisation error bounds for sparse linear classifiers. In: Proceedings of the Thirteenth Annual Conference on Computational Learning Theory, pp. 298–303 (2000)
7. Hussain, Z., Shawe-Taylor, J.: Theory of matching pursuit. In: Koller, D., Schuurmans, D., Bengio, Y., Bottou, L. (eds.) Advances in Neural Information Processing Systems, vol. 21, pp. 721–728 (2009)
8. Mallat, S., Zhang, Z.: Matching pursuit with time-frequency dictionaries. IEEE Transactions on Signal Processing 41(12), 3397–3415 (1993)
9. Plumbley, M.D.: Polar polytopes and recovery of sparse representations (2005)
10. Plumbley, M.D.: Recovery of sparse representations by polytope faces pursuit. In: Rosca, J.P., Erdogmus, D., Príncipe, J.C., Haykin, S. (eds.) ICA 2006. LNCS, vol. 3889, pp. 206–213. Springer, Heidelberg (2006)
11. Shawe-Taylor, J., Anthony, M., Biggs, N.L.: Bounding sample size with the Vapnik-Chervonenkis dimension. Discrete Applied Mathematics 42(1), 65–73 (1993)
12. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge (2004)

13. Smola, A.J., Bartlett, P.: Sparse greedy gaussian process regression. Advances in neural information processing systems 13 (2001)
14. Smola, A.J., Schölkopf, B.: Sparse greedy matrix approximation for machine learning. In: Proceedings of 17th International Conference on Machine Learning, pp. 911–918. Morgan Kaufmann, San Francisco (2000)
15. Tibshirani, R.: Regression selection and shrinkage via the lasso. Technical report, Department of Statistics, University of Toronto (June 1994),
    `ftp://utstat.toronto.edu/pub/tibs/lasso.ps`
16. Tropp, J.A., Gilbert, A.C., Strauss, M.J.: Algorithms for simultaneous sparse approximation. part i: Greedy pursuit. Signal Processing, special issue Sparse approximations in signal and image processing 86, 572–588 (2006)
17. Vincent, P., Bengio, Y.: Kernel matching pursuit. Machine Learning 48(1-3), 165–187 (2002)