Large-Scale Nonlinear Variable Selection via Kernel Random Features

Magda Gregorová[1,2(✉)], Jason Ramapuram[1,2], Alexandros Kalousis[1,2], and Stéphane Marchand-Maillet[2]

[1] Geneva School of Business Administration, HES-SO, Geneva, Switzerland
magda.gregorova@hesge.ch
[2] University of Geneva, Geneva, Switzerland

Abstract. We propose a new method for input variable selection in nonlinear regression. The method is embedded into a kernel regression machine that can model general nonlinear functions, not being a priori limited to additive models. This is the first kernel-based variable selection method applicable to large datasets. It sidesteps the typical poor scaling properties of kernel methods by mapping the inputs into a relatively low-dimensional space of random features. The algorithm discovers the variables relevant for the regression task together with learning the prediction model through learning the appropriate nonlinear random feature maps. We demonstrate the outstanding performance of our method on a set of large-scale synthetic and real datasets. Code related to this paper is available at: https://bitbucket.org/dmmlgeneva/srff_pytorch.

1 Introduction

It has been long appreciated in the machine learning community that learning sparse models can bring multiple benefits such as better interpretability, improved accuracy by reducing the curse of dimensionality, computational efficiency at prediction times, reduced costs for gathering and storing measurements, etc. A plethora of sparse learning methods has been proposed for linear models [16]. However, developing similar methods in the nonlinear setting proves to be a challenging task.

Generalized additive models [15] can use similar sparse techniques as their linear counterparts. However, the function class of linear combinations of nonlinear transformations is too limited to represent general nonlinear functions. Kernel methods [29] have for long been the workhorse of nonlinear modelling. Recently, a substantial effort has been invested into developing kernel methods with feature selection capabilities [5]. The most successful approaches within the filter methods are based on mapping distributions into the reproducing kernel Hilbert spaces (RKHS) [23]. Amongst the embedded methods, multiple algorithms use the feature-scaling weights proposed in [32]. The authors in [28] follow an alternative strategy based on the function and kernel partial derivatives.

All the kernel-based approaches above suffer from a common problem: they do not scale well for large data sets. The kernel methods allow for nonlinear

M. Berlingerio et al. (Eds.): ECML PKDD 2018, LNAI 11052, pp. 177–192, 2019.
https://doi.org/10.1007/978-3-030-10928-8_11

modelling by applying high dimensional (possibly infinite-dimensional) nonlinear transformations $\phi : \mathcal{X} \to \mathcal{H}$ to the input data. Due to what is known as the kernel trick, these transformations do not need to be explicitly evaluated. Instead, the kernel methods operate only over the inner products between pairs of data points that can be calculated quickly by the use of positive definite kernel functions $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, $k(\mathbf{x}, \tilde{\mathbf{x}}) = \langle \phi(\mathbf{x}), \phi(\tilde{\mathbf{x}}) \rangle$. Given that these inner products need to be calculated for all data-point pairs, the kernel methods are generally costly for datasets with a large number n of training points both in terms of computation and memory. This is further exacerbated for the kernel variable selection methods, which typically need to perform the $\mathcal{O}(n^2)$ kernel evaluations multiple times (per each input dimension, or with each iterative update).

In this work we propose a novel kernel-based method for input variable selection in nonlinear regression that can scale to datasets with large numbers of training points. The method builds on the idea of approximating the kernel evaluations by Fourier random features [24]. Instead of fixing the distributions generating the random features a priori, it learns them together with the predictive model such that they degenerate for the irrelevant input dimensions. The method falls into the category of embedded approaches that seek to improve predictive accuracy of the learned models through sparsity [18]. This is the first kernel-based variable selection method for general nonlinear functions that can scale to large datasets of tens of thousands of training data.

2 Background

We formulate the problem of nonlinear regression as follows: given a training set of n input-output pairs $\mathcal{S}_n = \{(\mathbf{x}_i, y_i) \in (\mathcal{X} \times \mathcal{Y}) : \mathcal{X} \subseteq \mathbb{R}^d, \mathcal{Y} \subseteq \mathbb{R}, i \in \mathbb{N}_n\}$ sampled i.i.d. according to some unknown probability measure ρ, our task is to estimate the regression function $f : \mathcal{X} \to \mathcal{Y}$, $f(\mathbf{x}) = \mathbb{E}(y|\mathbf{x})$ that minimizes the expected squared error loss $\mathcal{L}(f) = \mathbb{E}\left(y - f(\mathbf{x})\right)^2 = \int \left(y - f(\mathbf{x})\right)^2 d\rho(\mathbf{x}, y)$.

In the variable selection setting, we assume that the regression function does not depend on all the d input variables. Instead, it depends only on a subset $\mathcal{I}$ of these of size $l < d$, so that $f(\mathbf{x}) = f(\tilde{\mathbf{x}})$ if $x^s = \tilde{x}^s$ for all dimensions $s \in \mathcal{I}$.

We follow the standard theory of regularised kernel learning and estimate the regression function as the solution to the following problem

$$\widehat{f} = \arg\min_{f \in \mathcal{H}} \widehat{\mathcal{L}}(f) + \lambda ||f||^2_{\mathcal{H}}. \tag{1}$$

Here the function hypothesis space $\mathcal{H}$ is a reproducing kernel Hilbert space (RKHS), $||f||_{\mathcal{H}}$ is the norm induced by the inner product in that space, and $\widehat{\mathcal{L}}(f) = \frac{1}{n} \sum_i^n \left(y_i - f(\mathbf{x}_i)\right)^2$ is the empirical loss replacing the intractable expected loss above.

From the standard properties of the RKHS, the classical result (e.g. [29]) states that the evaluation of the minimizing function $\widehat{f}$ at any point $\tilde{\mathbf{x}} \in \mathcal{X}$

can be represented as a linear combination of the kernel functions k over the n training points

$$\widehat{f}(\tilde{\mathbf{x}}) = \sum_{i}^{n} c_i \, k(\mathbf{x}_i, \tilde{\mathbf{x}}). \tag{2}$$

The parameters $\mathbf{c}$ are obtained by solving the linear problem

$$(\mathbf{K} + \lambda \mathbf{I}_n)\, \mathbf{c} = \mathbf{y}, \tag{3}$$

where $\mathbf{K}$ is the $n \times n$ kernel matrix with the elements $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ for all $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{S}_n$.

2.1 Random Fourier Features

Equations (2) and (3) point clearly to the scaling bottlenecks of the kernel regression. In principal, at training it needs to construct and keep in memory the $(n \times n)$ kernel matrix and solve an n dimensional linear system ($\propto \mathcal{O}(n^3)$). Furthermore, the whole training set $\mathcal{S}_n$ needs to be stored and accessed at test time so that the predictions are of the order $\mathcal{O}(n)$.

To address these scaling issues, the authors in [24] proposed to map the data into a low-dimensional Euclidean space $\mathbf{z} : \mathcal{X} \to \mathbb{R}^D$ so that the inner products in this space are close approximations of the corresponding kernel evaluation $\langle \mathbf{z}(\mathbf{x}), \mathbf{z}(\tilde{\mathbf{x}}) \rangle_{\mathbb{R}^D} \approx \langle \phi(\mathbf{x}), \phi(\tilde{\mathbf{x}}) \rangle_{\mathcal{H}} = k(\mathbf{x}, \tilde{\mathbf{x}})$. Using the nonlinear features $\mathbf{z}(\mathbf{x}) \in \mathbb{R}^D$ the evaluations of the minimising function can be approximated by

$$\widehat{f}(\tilde{\mathbf{x}}) \approx \langle \mathbf{z}(\tilde{\mathbf{x}}), \mathbf{a} \rangle_{\mathbb{R}^D}, \tag{4}$$

where the coefficients $\mathbf{a}$ are obtained from solving the linear system

$$(\mathbf{Z}^T \mathbf{Z} + \lambda \mathbf{I}_D)\, \mathbf{a} = \mathbf{Z}^T \mathbf{y}, \tag{5}$$

where $\mathbf{Z}$ is the $(n \times D)$ matrix of the random features for all the data points. The above approximation requires the construction of the $\mathbf{Z}$ matrix and solving the D-dimensional linear problem, hence significantly reducing the training costs if $D \ll n$. Moreover, access to training points is no longer needed at test time and the predictions are of the order $\mathcal{O}(D) \ll \mathcal{O}(n)$.

To construct well-approximating features, the authors in [24] called upon Bochner's theorem which states that a continuous function $g : \mathbb{R}^d \to \mathbb{R}$ with $g(\mathbf{0}) = 1$ is positive definite if and only if it is a Fourier transform of some probability measure on $\mathbb{R}^d$. For translation-invariant positive definite kernels we thus have

$$k(\mathbf{x}, \tilde{\mathbf{x}}) = g(\mathbf{x} - \tilde{\mathbf{x}}) = g(\boldsymbol{\lambda}) = \int_{\mathbb{R}^d} e^{i \boldsymbol{\omega}^T \boldsymbol{\lambda}} \, \mathrm{d}\mu(\boldsymbol{\omega}), \tag{6}$$

where $\mu(\boldsymbol{\omega})$ is the probability measure on $\mathbb{R}^d$. In the above, g is the characteristic function of the multivariate random variable $\boldsymbol{\omega}$ defined by the expectation $g(\boldsymbol{\lambda}) = \mathbb{E}_{\boldsymbol{\omega}}(e^{i \boldsymbol{\omega}^T \boldsymbol{\lambda}}) = \mathbb{E}_{\boldsymbol{\omega}}(e^{i \boldsymbol{\omega}^T (\mathbf{x} - \tilde{\mathbf{x}})}) = \mathbb{E}_{\boldsymbol{\omega}}(e^{i \boldsymbol{\omega}^T \mathbf{x}} e^{-i \boldsymbol{\omega}^T \tilde{\mathbf{x}}}) = k(\mathbf{x}, \tilde{\mathbf{x}})$.

It is straightforward to show that the expectation over the complex exponential can be decomposed into an expectation over an inner product $\mathbb{E}_{\boldsymbol{\omega}}(e^{i\boldsymbol{\omega}^T(\mathbf{x}-\tilde{\mathbf{x}})}) = \mathbb{E}_{\boldsymbol{\omega}}\langle\psi_{\boldsymbol{\omega}}(\mathbf{x}), \psi_{\boldsymbol{\omega}}(\tilde{\mathbf{x}})\rangle$ where the nonlinear mappings are defined as $\psi_{\boldsymbol{\omega}} : \mathcal{X} \to \mathbb{R}^2$, $\psi_{\boldsymbol{\omega}}(\mathbf{x}) = [\cos(\boldsymbol{\omega}^T\mathbf{x})\ \sin(\boldsymbol{\omega}^T\mathbf{x})]^T$. In [24] the authors proposed an even lower-dimensional transformation $\varphi_{\boldsymbol{\omega},b}(\mathbf{x}) : \mathcal{X} \to \mathbb{R}$

$$\varphi_{\boldsymbol{\omega},b}(\mathbf{x}) = \sqrt{2}\cos(\boldsymbol{\omega}^T\mathbf{x} + b), \tag{7}$$

where b is sampled uniformly from $[0, 2\pi]$ and that satisfies the expectation equality $\mathbb{E}_{\boldsymbol{\omega}}(e^{i\boldsymbol{\omega}^T(\mathbf{x}-\tilde{\mathbf{x}})}) = \mathbb{E}_{\boldsymbol{\omega},b}\langle\varphi_{\boldsymbol{\omega},b}(\mathbf{x}), \varphi_{\boldsymbol{\omega},b}(\tilde{\mathbf{x}})\rangle$. We chose to work with the mapping φ (dropping the subscripts $\boldsymbol{\omega}, b$ when there is no risk of confusion) in the remainder of the text. The approximating nonlinear feature $\mathbf{z}(\mathbf{x})$ for each data-point $\mathbf{x}$ is obtained by concatenating D instances of the random mappings $\mathbf{z}(\mathbf{x}) = [\varphi^1(\mathbf{x}), \ldots, \varphi^D(\mathbf{x})]^T$ with $\boldsymbol{\omega}$ and b sampled according to their probability distribution so that the expectation is approximated by the sample sum.

2.2 Variable Selection Methods

In this section we position our research with respect to other nonlinear methods for variable selection with an emphasis on kernel methods.

In the class of generalized additive models, lessons learned from the linear models can be reused to construct sparse linear combinations of the nonlinear functions of each variable or, taking into account also possible interactions, of all possible pairs, triplets, etc., e.g. [20,26,31,34]. Closely related to these are the multiple kernel learning methods that seek to learn a sparse linear combination of kernel bases, e.g. [2,3,19]. While these methods have shown some encouraging results, their simplifying additive assumption and the fast increasing complexity when higher-order interactions shall be considered (potentially 2^d additive terms for d input variables) clearly present a serious limitation.

Recognising these shortcomings, multiple alternative approaches for general nonlinear functions were explored in the literature. They can broadly be grouped into three categories [18]: filters, wrappers and embedded methods.

The filter methods consider the variable selection as a preprocessing step that is then followed by an independent algorithm for learning the predictive model. Many traditional methods based on information-theoric or statistical measures of dependency (e.g. information gain, Fisher-score, etc.) fall into this category [6]. More recently, significant advancement has been achieved in formulating criteria more appropriate for non-trivial nonlinear dependencies [8,9,12,27,30,33]. These are based on the use of (conditional) cross-covariance operators arising from embedding probability measures into the reproducing kernel Hilbert spaces (RKHS) [23]. However, they are still largely disconnected from the predictive model learning procedure and oblivious of the effects the variable selection has on the final predictive performance.

The wrapper methods perform variable selection on top of the learning algorithm treating it as a black box. These are practical heuristics (such as greedy

forward or backward elimination) for the search in the 2^d space of all possible subsets of the d input variables [18]. Classical example in this category is the SVM with Recursive Feature Elimination [14]. The wrappers are universal methods that can be used on top of any learning algorithm but they can become expensive for large dimensionalities d, especially if the complexity of the underlying algorithm is high.

Finally, the embedded methods link the variable selection to the training of the predictive model with the view to achieve higher predictive accuracy stemming from the learned sparsity. Our method falls into this category. There are essentially just two branches of kernel-based methods here: methods based on feature rescaling [1,10,21,25,32], and derivative-based methods [11,28]. We discuss the feature rescaling methods in more detail in Sect. 3.2. The derivative based methods use regularizers over the partial derivatives of the function and exploit the derivative reproducing property [36] to arrive at an alternative finite-dimensional representation of the function. Though theoretically intriguing, these methods scale rather badly as in addition to the $(n \times n)$ kernel matrix they construct also the $(nd \times n)$ and $(nd \times nd)$ matrices of first and second order partial kernel derivatives and use their concatenations to formulate the sparsity constrained optimisation problem.

There exist two other large groups of *sparse* nonlinear methods. These address the sparsity in either the latent feature representation, e.g. [13], or in the data instances, e.g. [7]. While their motivation partly overlaps with ours (control of overfitting, lower computational costs at prediction), their focus is on a different notion of sparsity that is out of the scope of our discussion.

3 Towards Sparsity in Input Dimensions

As stated above, our objective in this paper is learning a regression function that is sparse with respect to the input variables. Stated differently, the function shall be insensitive to the values of the inputs in the $d - l$ large complement set $\mathcal{I}^{\mathsf{c}}$ of the irrelevant dimensions so that $f(\mathbf{x}) = f(\tilde{\mathbf{x}})$ if $x^s = \tilde{x}^s$ for all $s \in \mathcal{I}$.

From Eq. (4) we observe that the function evaluation is a linear combination of the D random features φ. The random features (7) are in turn constructed from the input $\mathbf{x}$ through the inner product $\boldsymbol{\omega}^T\mathbf{x}$. Intuitively, if the function $\widehat{f}$ is insensitive to an input dimension s, the value of the corresponding input x^s shall not enter the inner product $\boldsymbol{\omega}^T\mathbf{x}$ generating the D random features. Formally, we require $\omega^s x^s = 0$ for all $s \in \mathcal{I}^{\mathsf{c}}$ which is obviously achieved by $\omega^s = 0$. We therefore identify the problem of learning sparse predictive models with sparsity in vectors $\boldsymbol{\omega}$.

3.1 Learning Through Random Sampling

Though in Eq. (7) $\boldsymbol{\omega}$ appears as a parameter of the nonlinear transformation φ, it cannot be learned directly as it is the result of random sampling from the probability distribution $\mu(\boldsymbol{\omega})$. In order to ensure the same sparse pattern in the

D random samples of $\boldsymbol{\omega}$, we use a procedure similar to what is known as the reparametrization trick in the context of variational auto-encoders [17].

We begin by expanding Eq. (6) of the Bochner's theorem into the marginals across the d dimensions[1]

$$g(\boldsymbol{\lambda}) = \int_{\mathbb{R}^d} e^{i\,\boldsymbol{\omega}^T\boldsymbol{\lambda}}\,\mathsf{d}\mu(\boldsymbol{\omega}) = \int_{\mathbb{R}} e^{i\,\omega^1\lambda^1}\mathsf{d}\mu(\omega^1)\ldots\int_{\mathbb{R}} e^{i\,\omega^d\lambda^d}\mathsf{d}\mu(\omega^d). \tag{8}$$

To ensure that $\omega^s = 0$ when $s \in \mathcal{I}^{\mathsf{c}}$ in all the D random samples, the corresponding probability measure (distribution) $\mu(\omega^s)$ needs to degenerate to $\delta(\omega^s)$. The distribution $\delta(\omega^s)$ has all its mass concentrated at the point $\omega^s = 0$, and has the property $\int_{\mathcal{X}} h(\omega^s)\,\mathsf{d}\delta(\omega^s) = h(0)$. In particular for h the complex exponential we have $\int_{\mathcal{X}} e^{i\,\omega^s\lambda^s}\,\mathsf{d}\delta(\omega^s) = 1$ so that the value of λ^s has no impact on the product in Eq. (8), and therefore no impact on $g(\boldsymbol{\lambda})$.[2]

Reparametrization Trick. To ensure that all the D random samples of $\boldsymbol{\omega}$ have the same sparse pattern we need to be able to optimise through its random sampling. For each element ω of the vector $\boldsymbol{\omega}$, we parametrize the sampling distributions $\mu_\gamma(\omega)$ by its scale γ so that $\lim_{\gamma\to 0}\mu_\gamma(\omega) = \delta(\omega)$. We next express each of the univariate random variables ω as a deterministic transformation of the form $\omega = q_\gamma(\epsilon) = \gamma\epsilon$ (scaling) of an auxiliary random variable ϵ with a fixed probability distribution $\mu_1(\epsilon)$ with the scale parameter $\gamma = 1$. For example, for the Gaussian and Laplace kernels the auxiliary distribution $\mu_1(\epsilon)$ are the standard Gaussian and Cauchy respectively.

By the above reparametrization of the random variable $\boldsymbol{\omega}$ we disconnect the sampling operation over $\boldsymbol{\epsilon}$ from the rescaling operation $\boldsymbol{\omega} = \mathbf{q}_{\boldsymbol{\gamma}}(\boldsymbol{\epsilon}) = \boldsymbol{\epsilon}\odot\boldsymbol{\gamma}$ with a deterministic parameter vector $\boldsymbol{\gamma}$. Sparsity in $\boldsymbol{\omega}$ (and therefore the learned model) can now be achieved by learning sparse parameter vector $\boldsymbol{\gamma}$.

Though in principle it would be possible to learn the sparsity in the sampled $\boldsymbol{\omega}$'s directly, this would mean sparsifying instead of one vector $\boldsymbol{\gamma}$ the D sampled vectors $\boldsymbol{\omega}$. Moreover, the procedure would need to cater for the additional constraint that all the samples have the same sparse pattern. While theoretically possible, we find our reparametrization approach more elegant and practical.

3.2 Link to Feature Scaling

In the previous section we have built our strategy for sparse learning using the inverse Fourier transform of the kernels and the degeneracy of the associated probability measures. When we plug the rescaling operation into the random feature mapping (7)

$$\varphi(\mathbf{x}) = \sqrt{2}\cos(\boldsymbol{\omega}^T\mathbf{x} + b) = \sqrt{2}\cos((\boldsymbol{\epsilon}\odot\boldsymbol{\gamma})^T\mathbf{x} + b) = \sqrt{2}\cos(\boldsymbol{\epsilon}^T(\boldsymbol{\gamma}\odot\mathbf{x}) + b), \tag{9}$$

we see that the parameters $\boldsymbol{\gamma}$ can be interpreted as weights scaling the input variables $\mathbf{x}$. This makes a link to the variable selection methods based on feature

[1] This is possible due to the independence of the d dimensions of the r.v. $\boldsymbol{\omega}$.

[2] And from (6) and (4) it neither impacts the kernel and regression function evaluation.

scaling. These are rather straightforward when the kernel is simply linear, or when the nonlinear transformations $\phi(\mathbf{x})$ can be evaluated explicitly (e.g. polynomial) [10,32]. In essence, instead of applying the weights to the input features, they are applied to the associated model parameters and suitably constrained to approximate the zero-norm problem.

More complex kernels, for which the nonlinear features $\phi(\mathbf{x})$ cannot be directly evaluated (may be infinite dimensional), are considered in [1,21,25]. Here the scaling is applied within the kernel function $k(\boldsymbol{\gamma} \odot \mathbf{x}, \boldsymbol{\gamma} \odot \tilde{\mathbf{x}})$. The methods typically apply a two-step iterative procedure: they fix the rescaling parameters $\boldsymbol{\gamma}$ and learn the corresponding n-long model parameters vector $\mathbf{c}$ (Eq. (2)); fix $\mathbf{c}$ and learn the d-long rescaling vector $\boldsymbol{\gamma}$ under some suitable zero-norm approximating constraint. The naive formulation for $\boldsymbol{\gamma}$ is a nonconvex problem that requires calculating derivatives of the kernel functions with respect to $\boldsymbol{\gamma}$ (which depending on the kernel function may become rather expensive). In [1], the author proposed a convex relaxation based on linearization of the kernel function. Nevertheless, all the existing methods applying the feature scaling within the kernel functions scale badly with the number of instances as they need to recalculate the $(n \times n)$ kernel matrix and solve the corresponding optimisation (typically $\mathcal{O}(n^3)$) with every update of the weights $\boldsymbol{\gamma}$.

4 Sparse Random Fourier Features Algorithm

In this section we present our algorithm for learning with Sparse Random Fourier Features (SRFF).

Input : training data $(\mathbf{X}, \mathbf{y})$; hyper-parameters λ, D, size of Δ simplex
Output : model parameters $\mathbf{a}$, scale vector $\boldsymbol{\gamma}$
Initialise : $\boldsymbol{\gamma}$ evenly on simplex Δ, $\boldsymbol{\epsilon}_j \sim \mu_I(\boldsymbol{\epsilon})$ and $b_j \sim U[0, 2\pi]$, $\forall j \in \mathbb{N}_D$
Objective: $J(\mathbf{a}, \boldsymbol{\gamma}) = ||\mathbf{y} - \mathbf{Z}\mathbf{a}||_2^2 + \lambda ||\mathbf{a}||_2^2$

repeat `// Alternating descent`
  begin Step 1: Solve for $\mathbf{a}$
    rescalings $\boldsymbol{\omega}_j = \boldsymbol{\gamma} \odot \boldsymbol{\epsilon}_j$, $\forall j \in \mathbb{N}_D$
    random features $\mathbf{z}(\mathbf{x}) = [\varphi^1(\mathbf{x}), \ldots, \varphi^D(\mathbf{x})]$, $\forall \mathbf{x} \in \mathcal{S}_n$ `// equation (9)`
    $\mathbf{a} \leftarrow \arg\min_a ||\mathbf{y} - \mathbf{Z}\mathbf{a}||_2^2 + \lambda ||\mathbf{a}||_2^2$ `// equation (5)`
  end
  begin Step 2: Solve for $\boldsymbol{\gamma}$
    $\boldsymbol{\gamma} \leftarrow \arg\min_{\boldsymbol{\gamma} \in \Delta} ||\mathbf{y} - \mathbf{Z}\mathbf{a}||_2^2$ `// projected gradient descent`
  end
until *objective convergence*;

Algorithm 1. Sparse Random Fourier Features (SRFF) algorithm

Similarly to the feature scaling methods we propose a two-step alternative procedure to learn the model parameters $\mathbf{a}$ and the distribution scalings $\boldsymbol{\gamma}$. For a fixed $\boldsymbol{\gamma}$ we generate the random features for all the input training points $\mathcal{O}(nD)$, and solve the linear problem (5) $\mathcal{O}(D^3)$ to get the D-long model parameters $\mathbf{a}$.

Given that in our large-sample settings we assume $D \ll n$, this step is significantly cheaper than the corresponding step for learning the $\mathbf{c}$ parameters in the existing kernel feature scaling methods described in Sect. 3.2.

In the second step, we fix the model parameters $\mathbf{a}$ and learn the d-long vector of the distribution scalings $\boldsymbol{\gamma}$. We formulate the optimisation problem as the minimisation of the empirical squared error loss with $\boldsymbol{\gamma}$ constrained on the probability simplex Δ to encourage the sparsity.

$$\arg\min_{\boldsymbol{\gamma}\in\Delta} J(\boldsymbol{\gamma}), \qquad J(\boldsymbol{\gamma}) := ||\mathbf{y} - \mathbf{Z}\mathbf{a}||_2^2 \tag{10}$$

Here the $(n \times D)$ matrix $\mathbf{Z}$ is constructed by concatenating the D random features φ with the $\boldsymbol{\gamma}$ rescaling (9).

We solve problem (10) by the projected gradient method with accelerated FISTA line search [4]. The gradient can be constructed from the partial derivatives as follows

$$\begin{aligned} \frac{\partial J(\boldsymbol{\gamma})}{\gamma^s} &= -(\mathbf{y} - \mathbf{Z}\,\mathbf{a})^T \frac{\partial \mathbf{Z}}{\partial \gamma^s}\,\mathbf{a} \qquad\qquad \forall s \in \mathbb{N}_d \\ \frac{\partial Z_{ij}}{\partial \gamma^s} &= -\sqrt{2}\sin(\boldsymbol{\epsilon}^T(\boldsymbol{\gamma}\odot\mathbf{x}) + b)\,\epsilon^s x^s, \qquad \epsilon^s = \omega_s/\gamma_s. \end{aligned} \tag{11}$$

Unlike in the other kernel feature scaling methods, the form of the gradient (11) is always the same irrespective of the choice of the kernel. The particular kernel choice is reflected only in the probability distribution from which the auxiliary variable $\boldsymbol{\epsilon}$ is sampled and has no impact on the gradient computations. In our implementation (https://bitbucket.org/dmmlgeneva/srff_pytorch), we leverage the automatic differentiation functionality of pytorch in order to obtain the gradient values directly from the objective formulation.

5 Empirical Evaluation

We implemented our algorithm in pytorch and made it executable optionally on CPUs or GPUs. All of our experiments were conducted on GPUs (single p100). The code including the settings of our experiments amenable for replication is publicly available at https://bitbucket.org/dmmlgeneva/srff_pytorch.

In our empirical evaluation we compare to multiple baseline methods. We included the nonsparse random Fourier features method (RFF) of [24] in our main SRFF code as a call option. For the naive mean and ridge regression we use our own matlab implementation. For the linear lasso we use the matlab PASPAL package [22]. For the nonlinear Sparse Additive Model (SPAM) [26] we use the R implementation of [35]. For the Hilbert-Schmidt Independence Criterion lasso method (HSIC) [33], and the derivative-based embedded method of [28] (Denovas) we use the authors' matlab implementation.

Except SPAM, all of the baseline sparse learning methods use a two step procedure for arriving at the final model. They first learn the sparsity using either predictive-model-dependent criteria (lasso, Denovas) or in a completely

disconnected fashion (HSIC). In the second step (sometimes referred to as debiasing [28]), they use a base non-sparse learning method (ridge, or kernel ridge) to learn a model over the selected variables (including hyper-parameter search and cross-validation). For HSIC, which is a filter method that does not natively predict the regression outputs, we use as the second step our implementation of the RFF. It searches through the candidate sparsity patterns HSIC produces and uses the validation mean square error as a criteria for the final model selection. In contrast to these, our SRFF method is a *single step procedure* that does not necessitate this extra re-learning phase.

Experimental Protocol. In all our experiments we use the same protocol. We randomly split the data into three independent subsets: train, validation and test. We use the train subset for training the models, we use the validation subset to perform the hyper-parameter search, and we use the test set to evaluate the predictive performance. We repeat all the experiments 30 times, each with a different random train/validation/test split.

We measure the predictive performance in terms of the root mean squared error (RMSE) over the test samples, averaged over the 30 random replications of the experiments. The regularization hyper-parameter λ (exists in ridge, lasso, Denovas, HSIC, RFF and SRFF) is searched within a 50-long data-dependent grid (automatically established by the methods). The smoothing parameter in Denovas is fixed to 10 following the authors' default [28]. We use the Gaussian kernel for all the experiments with the width σ set as the median of the Euclidean distances amongst the 20 nearest neighbour instances. We use the same kernel in all the kernel methods and the corresponding scale parameter $\gamma = 1/\sigma$ in the random feature methods for comparability of results. We fix the number of random features to $D = 300$ for all the experiments in both RFF and SRFF.

We provide the results of the baseline nonlinear sparse methods (SPAM, HSIC, Denovas) only for the smallest experiments. As explained in the previous sections, the motivation for our paper is to address the poor scaling properties of the existing methods. Indeed, none of the baseline kernel sparse methods scales to the problems we consider here. HSIC [33] creates a $(n \times n)$ kernel matrix per each dimension d and solves a linear lasso problem over the concatenated vectorization of these with memory requirements $(n^2 \times d)$ and complexity $\mathcal{O}(n^4)$. In our tests, it did not finish within 24 h running on 20 CPUs (Dual Core Intel Xeon E5-2680 v2/2.8 GHz) for the smallest training size of 1000 instances in our SE3 experiment. Within the same time span it did not arrive at a solution for any of the experiments with $n > 1000$. Denovas constructs, stores in memory, and operates over the $(n \times n)$, $(nd \times n)$ and $(nd \times nd)$ kernel matrix and the matrices of the first and second order derivatives. In our tests the method finished with an out-of-memory error (with 32 GB RAM) for the SE1 with 5k training samples and for SE2 problem already with 1k training instances. SPAM finished with errors for most of the real-data experiments.

5.1 Synthetic Experiments

We begin our empirical evaluation by exploring the performance over a set of synthetic experiments. The purpose of these is to validate our method under controlled conditions when we understand the true sparsity of the generating model. We also experiment with various nonlinear functions and increasing data sizes in terms of both the sample numbers n and the dimensionality d.

Table 1. Summary of synthetic experiments

Exp code	Train size	Test size	Total dims	Relevant dims	Generative function
SE1	1k–50k	1k	18	5	$y = \sin\left((x_1 + x_3)^2\right) \sin(x_7 x_8 x_9) + N(0, 0.1)$
SE2	1k–50k	1k	100	5	$y = \log\left((\sum_{s=11}^{15} x_s)^2\right) + N(0, 0.1)$
SE3	1k–50k	10k	1000	10	$y = 10(z_1^2 + z_3^2)e^{-2(z_1^2+z_3^2)} + N(0, 0.01)$

We use the same size for the test and validation samples. In all the experiments, the data instances are generated from a standard normal distribution. In the functions, subscripts are dimensions, superscripts are exponents. For more detailed description of the generative function of SE3 see the appropriate section in the text.

SE1: The very first of our experiments is a rather small problem with only $d = 18$ input dimensions of which only 5 are relevant for the regression function. In Table 2 we compare our SRFF method to the baselines for the smallest sample setting with $n = 1000$. Most of the methods (linear, additive or non-sparse) do not succeed in learning a model for the complex nonlinear relationships between the inputs and outputs and fall back to predicting simple mean.

The general nonlinear models with sparsity (HSIC, Denovas and SRFF) divert from the simple mean prediction. They all discover and use in the predictive model the same sparse pattern (see Fig. 1 for SRFF). Denovas and SRFF achieve almost identical results which confirms that our method is competitive with the state of the art methods in terms of predictive accuracy and variable selection.[3]

In Table 3 we document how the increasing training size contributes to improving the predictive performance even in the case of several thousands instances. The performance of the SRFF method for the largest 50k sample is by about 6% better than for the 1k training size. For the other methods the problem remains out of their reach[4] and they stick to the mean prediction even

[3] The low predictive performance of HSIC is the result of the 2nd model fitting step. It could potentially be improved with an additional kernel learning step. However, as we keep the kernel fixed for all the other methods, we do not perform the kernel search for HSIC either.

[4] The class of linear functions is too limited and the nonlinear function with all the variables considered by RFF is too complex.

Table 2. SE1 - Test RMSE for $n = 1000$

	Mean	Ridge	Lasso	RFF	SPAM	HSIC	Denovas	SRFF
RMSE	0.287	0.287	0.287	0.287	0.0287	0.341	**0.272**	**0.272**
Std	0.009	0.009	0.009	0.009	0.009	0.060	0.009	0.009

Predictive performance in terms of root mean squared error (RMSE) over independent test sets for the SE1 dataset with training size $n = 1000$. The std line is the standard deviation of the RMSE across the 30 resamples.

for higher training sizes.[5] We do not provide any comparisons with the nonlinear sparse methods because, as explained above, they do not scale to the sample sizes we consider here.

Table 3. SE1 - Test RMSE for increasing train size n

n	Mean	Ridge	Lasso	RFF	SRFF
1k	0.287 (0.009)	0.287 (0.009)	0.287 (0.009)	0.287 (0.009)	**0.272** (0.009)
5k	0.284 (0.011)	0.284 (0.011)	0.284 (0.011)	0.284 (0.011)	**0.263** (0.010)
10k	0.285 (0.010)	0.285 (0.010)	0.285 (0.010)	0.286 (0.010)	**0.261** (0.011)
50k	0.283 (0.010)	0.283 (0.010)	0.283 (0.010)	0.283 (0.010)	**0.255** (0.009)

Predictive performance in terms of root mean squared error (RMSE) over independent test sets for the SE1 dataset with increasing training size n. The standard deviation of the RMSE across the 30 resamples is in the brackets.

The improved predictive performance for the larger training sizes goes hand in hand with the variable selection, Fig. 1. For the smallest 1k training sample, SRFF identifies only the 7th, 8th and 9th relevant dimensions. They enter the sine in the generative function in a product and therefore have a larger combined effect on the function outcome than the squared sum of dimensions 1 and 3. These two dimensions are picked up by the method from the larger training sets and this contributes to the increase in the predictive performance.

SE2: In the second experiment we increase the dimensionality to $d = 100$ and change the nonlinear function (see Table 1). The overall outcomes are rather similar to the SE1 experiment. Again, it's only the nonlinear sparse models that predict something else than mean, SPAM marginally better, HSIC marginally worse. Our SRFF method clearly outperforms all the other methods in the predictive accuracy. It also correctly discovers the 5 relevant variables with the median value of γ for these dimensions between 0.92–1.04 while the maximum

[5] The small variations in the error stem from using different training sets to estimate the mean.

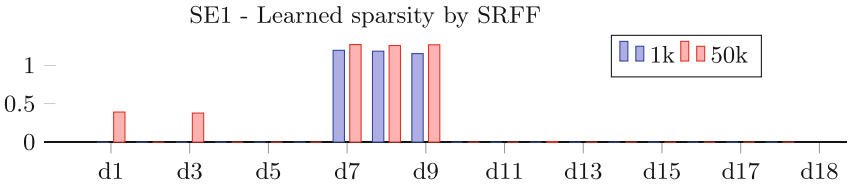

Fig. 1. Learned sparsity pattern $\boldsymbol{\gamma}$ by the SRFF method for the 1k and 50k training size in the SE1 experiment (the median of the 30 replications). The other nonlinear sparse methods learn the same pattern for the 1k problem but cannot solve the 50k problem.

for all the irrelevant variables is 0.06.[6] The advantage of SRFF over the baselines for large sample sizes (Tables 4 and 5) is even more striking than in the SE1 experiment (Tables 3 and 4).

Table 4. SE2 - Test RMSE for $n = 1000$

	Mean	Ridge	Lasso	RFF	SPAM	HSIC	SRFF
RMSE	2.216	2.216	2.216	2.216	2.162	2.357	**1.603**
Std	0.105	0.105	0.105	0.104	0.110	0.141	0.104

Predictive performance in terms of root mean squared error (RMSE) over independent test sets for the SE2 dataset with training size $n = 1000$. The std line is the standard deviation of the RMSE across the 30 resamples.

Table 5. SE2 - Test RMSE for increasing train size n

n	Mean	Ridge	Lasso	RFF	SRFF
1k	2.216 (0.105)	2.216 (0.105)	2.216 (0.105)	2.216 (0.105)	**1.603** (0.104)
5k	2.211 (0.079)	2.211 (0.079)	2.211 (0.079)	2.211 (0.079)	**1.278** (0.076)
10k	2.224 (0.115)	2.224 (0.115)	2.224 (0.115)	2.224 (0.115)	**1.272** (0.138)
50k	2.224 (0.082)	2.224 (0.082)	2.224 (0.082)	2.224 (0.082)	**1.273** (0.080)

Predictive performance in terms of root mean squared error (RMSE) over independent test sets for the SE2 dataset with increasing training size n. The standard deviation of the RMSE across the 30 resamples is in the brackets.

SE3: In this final synthetic experiment we increase the dimensionality to $d = 1000$ to further stretch our SRFF method. There are only 10 relevant input variables in this problem. The first 5 were generated as random perturbations of the random variable z_1, e.g. $x_1 = z_1 + N(0, 0.1)$, the second 5 by the same procedure from z_2, e.g. $x_5 = z_2 + N(0, 0.1)$. The remaining 990 input variables were generated by the same process from the other 198 standard normal z's.

[6] SPAM and HSIC discover the correct patterns as well but it does not help their predictive accuracy.

We summarise the results for the 1k and 50k training instances in Table 6. As in the other synthetic experiments, the baseline methods are not able to capture the nonlinear relationships of this extremely sparse problem and instead predict a simple mean. Our SRFF method achieves significantly better accuracy for the 1k training set, and it further considerably improves with 50k samples to train on. These predictive gains are possible due to SRFF correctly discovering the set of relevant variables. In the 1k case, the medians across the 30 data resamples of the learned $\boldsymbol{\gamma}$ parameters are between 0.37–0.71 for the 10 relevant variables and maximally 0.05 for the remaining 990 irrelevant variables. In the 50k case, the differences are even more clearly demarcated: 1.19–1.64 for the relevant, and 0.03 maximum for the irrelevant (bearing in mind that the total sum over the vector $\boldsymbol{\gamma}$ is the same in both cases).

Table 6. SE3 - Test RMSE for increasing train size n

n	Mean	Ridge	Lasso	RFF	SRFF
1k	0.676 (0.002)	0.676 (0.002)	0.676 (0.002)	0.676 (0.002)	**0.478** (0.031)
50k	0.677 (0.002)	0.677 (0.002)	0.677 (0.002)	0.677 (0.002)	**0.206** (0.004)

Predictive performance in terms of root mean squared error (RMSE) over independent test sets for the SE3 dataset with increasing training size n. The standard deviation of the RMSE across the 30 resamples is in the brackets.

5.2 Real Data Experiments

We experiment on four real datasets (Table 7): three from the LIACC[7] regression repository, and one Kaggle dataset[8]. The summary of these is presented in Table 8. The RFF results illustrate the advantage nonlinear modelling has over simple linear models. Our sparse nonlinear SRFF method clearly outperforms all the linear as well as the non-sparse nonlinear RFF method. Moreover, it is the only nonlinear sparse learning method that can handle problems of these large-scale datasets.

Table 7. Summary of real-data experiments

Data source	Dataset name	Exp code	Train size	Test size	Total dims
LIAC	Computer Activity	RCP	6k	1k	21
LIAC	F16 elevators	REL	6k	1k	17
LIAC	F16 ailernos	RAI	11k	1k	39
Kaggle	Ore mining impurity	RMN	50k	10k	21

We use the same size for the test and validation samples.

[7] http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html.
[8] https://www.kaggle.com/edumagalhaes/quality-prediction-in-a-mining-process.

Table 8. Real-data experiments - Test RMSE for increasing train size n

n	Mean	Ridge	Lasso	RFF	SRFF
RCP	18.518 (0.988)	9.686 (0.705)	9.689 (0.711)	8.194 (0.635)	**2.516** (0.184)
REL	1.044 (0.050)	0.514 (0.210)	0.468 (0.178)	0.446 (0.036)	**0.314** (0.032)
RAI	1.013 (0.034)	0.430 (0.018)	0.430 (0.017)	0.498 (0.038)	**0.407** (0.022)
RMN	1.014 (0.006)	0.987 (0.006)	0.987 (0.006)	0.856 (0.009)	**0.716** (0.008)

Predictive performance in terms of root mean squared error (RMSE) over independent test sets for the real datasets. The standard deviation of the RMSE across the 30 resamples is in the brackets.

6 Summary and Conclusions

We present here a new kernel-based method for learning nonlinear regression function with relevant variable subset selection. The method is unique amongst the state of the art as it can scale to tens of thousands training instances, way beyond what any of the existing kernel-based methods can handle. For example, while none of the tested sparse method worked over datasets with more than 1k instances, the CPU version of our SRFF finished the *full* validation search over 50 hyper-parameters λ in the 50k SE1 experiment within two hours on a laptop with a Dual Intel Core i3 (2nd Gen) 2350M/2.3 GHz and 16 GB RAM.

We focus here on nonlinear regression but the extension to classification problems is straightforward by replacing appropriately the objective loss function. We used the Gaussian kernel for our experiments as one of the most popular kernels in practice. But the principals hold for other shift-invariant kernels as well, and the method and the algorithm can be applied to them directly as soon as the corresponding probability measure $\mu(\boldsymbol{\omega})$ is recovered and the reparametrization of $\boldsymbol{\omega}$ explained in Sect. 3.1 can be applied.

Acknowledgements. This work was partially supported by the research projects HSTS (ISNET) and RAWFIE #645220 (H2020). The computations were performed at University of Geneva on the Baobab and Whales clusters. We specifically wish to thank Yann Sagon, the Baobab administrator, for his excellent work and continuous support.

References

1. Allen, G.I.: Automatic feature selection via weighted kernels and regularization. J. Comput. Graph. Stat. **22**(2), 284–299 (2013)
2. Bach, F.: Consistency of the group lasso and multiple kernel learning. J. Mach. Learn. Res. **9**, 1179–1225 (2008)
3. Bach, F.: High-dimensional non-linear variable selection through hierarchical kernel learning. ArXiv arXiv:0909.0844 (2009)
4. Beck, A., Teboulle, M.: A fast iterative shrinkage-thresholding algorithm for linear inverse problems. SIAM J. Imaging Sci. **2**(1), 183–202 (2009)

5. Bolón-Canedo, V., Sánchez-Maroño, N., Alonso-Betanzos, A.: A review of feature selection methods on synthetic data. Knowl. Inf. Syst. **34**(3), 483–519 (2013)
6. Bolón-Canedo, V., Sánchez-Maroño, N., Alonso-Betanzos, A.: Recent advances and emerging challenges of feature selection in the context of big data. Knowl. Based Syst. **86**, 33–45 (2015)
7. Chan, A.B., Vasconcelos, N., Lanckriet, G.R.G.: Direct convex relaxations of sparse SVM. In: International Conference on Machine Learning (2007)
8. Chen, J., Stern, M., Wainwright, M.J., Jordan, M.I.: Kernel feature selection via conditional covariance minimization. In: Advances in Neural Information Processing Systems (NIPS) (2017)
9. Fukumizu, K., Leng, C.: Gradient-based kernel method for feature extraction and variable selection. In: Advances in Neural Information Processing Systems (NIPS) (2012)
10. Grandvalet, Y., Canu, S.: Adaptive scaling for feature selection in SVMs. In: Advances in Neural Information Processing Systems (NIPS) (2002)
11. Gregorová, M., Kalousis, A., Marchand-Maillet, S.: Structured nonlinear variable selection. In: Conference on Uncertainty in Artificial Intelligence (UAI) (2018)
12. Gretton, A., Fukumizu, K., Teo, C.H., Song, L., Schölkopf, B., Smola, A.J.: A kernel statistical test of independence. In: Advances in Neural Information Processing Systems (NIPS) (2008)
13. Gurram, P., Kwon, H.: Optimal sparse kernel learning in the empirical kernel feature space for hyperspectral classification. IEEE J. Sel. Top. Appl. Earth Observ. Remote Sens. **7**(4), 1217–1226 (2014)
14. Guyon, I., Weston, J., Barnhill, S., Vapnik, V.: Gene selection for cancer classification using support vector machines. Mach. Learn. **46**(1–3), 389–422 (2002)
15. Hastie, T., Tibshirani, R.: Generalized Additive Models. Chapman and Hall, London (1990)
16. Hastie, T., Tibshirani, R., Wainwright, M.: Statistical Learning with Sparsity: The Lasso and Generalizations. CRC Press, Boca Raton (2015)
17. Kingma, D.P., Welling, M.: Auto-encoding variational Bayes. In: International Conference on Learning Representations (ICLR) (2014)
18. Kohavi, R., John, G.H.: Wrappers for feature subset selection. Artif. Intell. **97**(1–2), 273–324 (1997)
19. Koltchinskii, V., Yuan, M.: Sparsity in multiple kernel learning. Ann. Stat. **38**(6), 3660–3695 (2010)
20. Lin, Y., Zhang, H.H.: Component selection and smoothing in multivariate nonparametric regression. Ann. Stat. **34**(5), 2272–2297 (2006)
21. Maldonado, S., Weber, R., Basak, J.: Simultaneous feature selection and classification using kernel-penalized support vector machines. Inf. Sci. **181**(1), 115–128 (2011)
22. Mosci, S., Rosasco, L., Santoro, M., Verri, A., Villa, S.: Solving structured sparsity regularization with proximal methods. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD 2010. LNCS (LNAI), vol. 6322, pp. 418–433. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15883-4_27
23. Muandet, K., Fukumizu, K., Sriperumbudur, B., Schölkopf, B.: Kernel mean embedding of distributions: a review and beyond. Found. Trends Mach. Learn. **10**(1–2), 1–141 (2017)
24. Rahimi, A., Recht, B.: Random features for large-scale kernel machines. In: Advances in Neural Information Processing Systems (NIPS) (2007)
25. Rakotomamonjy, A.: Variable selection using SVM-based criteria. J. Mach. Learn. Res. **3**, 1357–1370 (2003)

26. Ravikumar, P., Liu, H., Lafferty, J., Wasserman, L.: Spam: sparse additive models. In: Advances in Neural Information Processing Systems (NIPS) (2007)
27. Ren, S., Huang, S., Onofrey, J.A., Papademetris, X., Qian, X.: A scalable algorithm for structured kernel feature selection. In: Aistats (2015)
28. Rosasco, L., Villa, S., Mosci, S.: Nonparametric sparsity and regularization. J. Mach. Learn. Res. **14**(1), 1665–1714 (2013)
29. Schölkopf, B., Smola, A.J.: Learning with Kernels. The MIT Press, Cambridge (2002)
30. Song, L., Smola, A., Gretton, A., Borgwardt, K.M., Bedo, J.: Supervised feature selection via dependence estimation. In: Proceedings of the 24th International Conference on Machine Learning - ICML 2007 (2007)
31. Tyagi, H., Krause, A., Eth, Z.: Efficient sampling for learning sparse additive models in high dimensions. In: International Conference on Artificial Intelligence and Statistics (AISTATS) (2016)
32. Weston, J., Elisseeff, A., Scholkopf, B., Tipping, M.: Use of the zero-norm with linear models and kernel methods. J. Mach. Learn. Res. **3**, 1439–1461 (2003)
33. Yamada, M., Jitkrittum, W., Sigal, L., Xing, E.P., Sugiyama, M.: High-dimensional feature selection by feature-wise kernelized lasso. Neural Comput. **26**(1), 185–207 (2014)
34. Yin, J., Chen, X., Xing, E.P.: Group sparse additive models. In: International Conference on Machine Learning (ICML) (2012)
35. Zhao, T., Li, X., Liu, H., Roeder, K.: CRAN - Package SAM (2014)
36. Zhou, D.X.: Derivative reproducing properties for kernel methods in learning theory. J. Comput. Appl. Math. **220**(1–2), 456–463 (2008)