The Kernel Recursive Least Squares Algorithm

Yaakov Engel
Interdisciplinary Center for Neural Computation
Hebrew University
Jerusalem 91904, Israel
yaki@alice.nc.huji.ac.il

Shie Mannor
Lab. for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, MA 02139
shie@mit.edu

Ron Meir
Dept. of Electrical Engineering
Technion Institute of Technology
Haifa 32000, Israel
rmeir@ee.technion.ac.il

Abstract

We present a non-linear kernel-based version of the Recursive Least Squares (RLS) algorithm. Our Kernel-RLS (KRLS) algorithm performs linear regression in the feature space induced by a Mercer kernel, and can therefore be used to recursively construct the minimum mean-squared-error regressor. Sparsity of the solution is achieved by a sequential sparsification process that admits into the kernel representation a new input sample only if its feature space image cannot be sufficiently well approximated by combining the images of previously admitted samples. This sparsification procedure is crucial to the operation of KRLS, as it allows it to operate on-line, and by effectively regularizing its solutions. A theoretical analysis of the sparsification method reveals its close affinity to kernel PCA, and a data-dependent loss bound is presented, quantifying the generalization performance of the KRLS algorithm. We demonstrate the performance and scaling properties of KRLS and compare it to a state-of-the-art Support Vector Regression algorithm, using both synthetic and real data. We additionally test KRLS on two signal processing problems in which the use of traditional least-squares methods is commonplace: Time series prediction and channel equalization.

Keywords: on-line learning, kernel methods, non-linear regression, sparse representations, recursive least-squares, signal processing

Table of Contents

1 Introduction 5

2 On-Line Sparsification 7

2.1 The Sparsification Procedure 8

2.2 Properties of the Sparsification Method 9

2.3 On-Line Sparsification as Approximate PCA 12

2.4 Comparison to Other Sparsification Schemes 13

3 The Kernel RLS Algorithm 15

4 A Generalization Bound 18

5 Experiments 21

5.1 Non-Linear Regression 21

5.2 Time Series Prediction 24

5.2.1 Mackey–Glass Time Series 25

5.2.2 Santa Fe Laser Time Series 26

5.3 Channel Equalization 28

6 Discussion and Conclusion 30

List of Tables

1 The Kernel RLS Algorithm. On the right we bound the number of operations per time-step for each line of the pseudo-code. The overall per time-step computational cost is bounded by $O(m^2)$ (we assume that kernel evaluations require $O(1)$ time) 16

2 Results on the synthetic *Friedman* data-sets. The columns, from left to right, list the average R.M.S. test-set error, its standard deviation, average percentage of support/dictionary vectors, and the average CPU time in seconds used by the respective algorithm. 24

3 Results on the real-world *Comp-activ* and *Boston* data-sets. 24

4 1-step and 200-step iterated prediction results on the Mackey–Glass time series with $\tau = 17$ (MG(17)) and $\tau = 30$ (MG(30)) 26

5 Results of the non-linear channel equalization experiment 30

List of Figures

1 Scaling properties of KRLS and SVMTorch with respect to sample size (top) and noise magnitude (bottom), on the *Sinc-Linear* function. Error bars mark one standard deviation above and below the mean. 23

2 Multi-step iterated predictions for the Mackey–Glass time series with $\tau = 17$ (top) and $\tau = 30$ (bottom). 26

3 The Santa Fe competition laser training series (data set A) 27

4 KRLS predicting 100 steps into the future (dashed line) on the laser time series. The true continuation is shown as a solid line. Note that on the first 60 steps the prediction error is hardly noticeable. 29

1 Introduction

The celebrated recursive least-squares (RLS) algorithm (e.g. [16, 14, 25]) is a popular and practical algorithm used extensively in signal processing, communications and control. The algorithm is an efficient on-line method for finding linear predictors minimizing the mean squared error over the training data. We consider the classic system identification setup (e.g. [17]), where we assume access to a recorded sequence of input and output samples

$$Z^t = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_t, y_t)\}$$

arising from some unknown source. In the classic regression (or function approximation) framework the input-output pairs $(\mathbf{x}_i, y_i)$ are assumed to be independent identically distributed (IID) samples from some distribution $p(Y, \mathbf{X})$. In signal processing applications the inputs more typically consist of lagged values of the outputs y_i, as would be the case for autoregressive (AR) sources, and/or samples of some other signal (ARMA and MA models, respectively). In the prediction problem, one attempts to find the best predictor $\hat{y}_t$ for y_t given $Z^{t-1} \cup \{\mathbf{x}_t\}$. In this context, one is often interested in on-line applications, where the predictor is updated following the arrival of each new sample. On-line algorithms are useful in learning scenarios where input samples are observed sequentially, one at a time (e.g. data mining, time series prediction, reinforcement learning). In such cases there is a clear advantage to algorithms that do not need to relearn from scratch when new data arrive. In many of these applications there is an additional requirement for *real-time* operation, meaning that the algorithm's computational cost per time-step should be bounded by a constant independent of time, for it is assumed that new samples arrive at a roughly constant rate.

Standard approaches to the prediction problem usually assume a simple parametric form, e.g. $\hat{y}(t|\mathbf{w}) = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_t) \rangle$, where $\mathbf{w}$ is a vector of parameters and $\boldsymbol{\phi}$ is a *fixed, finite dimensional mapping.* In the classic least-squares approach, one then attempts to find the value of $\mathbf{w}$ that minimizes the squared error $\sum_{i=1}^{t} \left(y_i - \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_i) \rangle \right)^2$. The RLS algorithm is used to recursively solve this least-squares problem for $\mathbf{w}$. Given a new sample $(\mathbf{x}_t, y_t)$, the number of computations performed by RLS to derive a new minimum least-squares estimate of $\hat{\mathbf{w}}$ is independent of t, making it suitable for real-time applications.

Kernel machines are a relatively new class of learning algorithms utilizing Mercer kernels to produce non-linear and non-parametric versions of conventional supervised and unsupervised learning algorithms. The basic idea behind kernel machines is that a Mercer kernel function, applied to pairs of input vectors, can be interpreted as an inner product in a high dimensional Hilbert space $\mathcal{H}$ (the feature space), thus allowing inner products in feature space to be computed without making direct reference to feature vectors. This idea, commonly known as the "kernel trick", has been used extensively in recent years, most notably in classification and regression [5, 15, 27]. Focusing on regression, several kernel based algorithms have been proposed, most prominently Support Vector Regression (SVR) [37] and Gaussian Process Regression (GPR) [42].

Kernel methods present an alternative to the parametric approach. Solutions attained by these methods are non-parametric in nature and are typically of the form

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{t} \alpha_i k(\mathbf{x}_i, \mathbf{x}) \ , \tag{1.1}$$

where $\{\mathbf{x}_i\}_{i=1}^t$ are the training data points. The Representer Theorem [40] assures us that in most practical cases, we need not look any further than an expression of the form (1.1). Since the number of of tunable parameters in kernel solutions equals the size of the training data-set, one must introduce some form of regularization. For instance, in SVR regularization is attained by using the so-called "ε-insensitive" error tolerant cost function, in conjunction with an additional regularization (penalty) term encouraging "flat" solutions. The end effect of this form of regularization is that SVR solutions are typically sparse – meaning that many of the α_i variables vanish in the SVR solution (1.1).

In SVR, and more generally in regularization networks [10], sparsity is achieved by *elimination*. This means that, at the outset, these algorithms consider all training samples as potential contributing members of the expansion (1.1); and upon solving the optimization problem they eliminate those samples whose coefficients vanish. An alternative approach is to obtain sparsity by *construction*. Here the algorithm starts with an empty representation, in which all coefficients vanish, and gradually adds samples according to some criterion. Constructive sparsification is normally used off-line (e.g. [39]), in which case the algorithm is free to choose any one of the training samples at each step of the construction process. Due to the intractability of finding the best subset of samples [22], these algorithms usually resort to employing various *greedy* selection strategies, in which at each step the sample selected is the one that maximizes the amount of increase (or decrease) its addition induces in some fitness (or error) criterion.

In a nutshell, the major obstacles in applying kernel methods to on-line algorithms are: (i) Many kernel methods require random/multiple access to training samples, (ii) their computational cost (both in time and space) is super-linear in the size of the training set, and (iii) their prediction (query) time often scales linearly with the training set size.

Clearly, non of the methods described above is suitable for on-line applications. In such applications the algorithm is presented at each time step with a single training sample and a simple dichotomic decision has to be made: Either add the next sample into the representation, or discard it. In [9] we proposed a solution to this problem by an on-line constructive sparsification method based on sequentially admitting into the kernel representation only samples that cannot be approximately represented by linear combinations of previously admitted samples. In [9] our sparsification method was used to construct an on-line SVR-like algorithm [1].

The main contribution of this report is the introduction of KRLS – a kernel based version of the RLS algorithm – that is capable of efficiently and recursively solving non-linear least-squares prediction problems, and that is therefore particularly useful in applications requiring

[1] We comment that several approaches to sparsification in the context of kernel methods in general, and kernel regression in particular, have been suggested recently. We compare our method to these in Section 6.

on-line or real-time operation. In addition, we discuss with further detail our on-line kernel sparsification procedure, which possesses a merit of its own in the context of kernel based learning methods and signal processing.

2 On-Line Sparsification

Sparse solutions for kernel algorithms are desirable for two main reasons. First, instead of storing information pertaining to the entire history of training instances, sparsity allows the solution to be stored in memory in a compact form and to be easily used later. The sparser is the solution of a kernel algorithm, the less time and memory are consumed in both the learning and the operation (query) phases of the kernel machine. Second, sparsity is related to generalization ability, and is considered a desirable property in learning algorithms (see, e.g. [27, 15]) as well as in signal processing (e.g. [19]). The ability of a kernel machine to correctly generalize from its learned experience to new data can be shown to improve as the number of its free variables decreases (as long as the training error does not increase), which means that sparsification may be used as a regularization instrument.

In the classic SVM framework sparsity is achieved by making use of error-tolerant cost functions in conjunction with an additional regularization term encouraging "flat" solutions by penalizing the squared norm of the weight vector [27]. For SV classification, it has been shown in [38] that the expected number of of SVs is bounded below by $(t-1)E(p_{err})$ where t is the number of training samples and $E(p_{err})$ is the expectation of the error probability on a test sample. In spite of claims to the contrary [33], it has been shown, both theoretically and empirically [2, 8], that the solutions provided by SVMs are not always maximally sparse. It also stands to reason that once a sufficiently large training set has been learned, any additional training samples would not contain much new information and therefore should not cause a linear increase in the size of the solution.

We take a somewhat different approach toward sparsification that is based on the following observation: Although the dimension of the feature space $\mathcal{H}$ is usually very high, or even infinite; the *effective* dimensionality of the manifold spanned by the training feature vectors may be significantly lower. Consequently, the solution to any optimization problem conforming to the conditions required by the Representer Theorem [40], may be expressed, to arbitrary accuracy, by a set of linearly independent feature vectors that *approximately span* this manifold.

As already mentioned above, kernel methods typically output a predictor which is a function of the training set. This implies that by time t the estimate of f will have the form (1.1), which as discussed before, is in fact a linear predictor in the Hilbert space $\mathcal{H}$:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{t} \alpha_i \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}) \rangle.$$

A naive approach would be to admit all training points $\mathbf{x}_1, \ldots, \mathbf{x}_t$ into an expression of the form (1.1) and solve the least-squares problem for the coefficients α_i. This, however, would lead to an algorithm whose complexity grows as more data points are added, and would additionally result in severe overfitting. Essentially, the problem is that as more points are considered, t increases and the dimensionality of the space spanned by $\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_t)$ increases. We note that in the "classic" linear RLS algorithm, the complexity of the update step depends on the dimensionality of the input samples. While in linear RLS the dimension of the input samples does not change in time, the dimension of $\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_t)$ may increase at every time step, depending on the data points and the specific kernel being used. If, however, the point $\mathbf{x}_t$ satisfies $\boldsymbol{\phi}(\mathbf{x}_t) = \sum_{i=1}^{t-1} a_i \boldsymbol{\phi}(\mathbf{x}_i)$, then in the estimate $\hat{f}$ at time t (and any subsequent time) there is no need to have a non-zero coefficient for $\boldsymbol{\phi}(\mathbf{x}_t)$ as it can be absorbed in the other terms. This idea would work when $\mathcal{H}$ is low dimensional, or if the data happens to belong to a low dimensional subspace of the feature space. However, for many kernels $\mathcal{H}$ is high dimensional, or even infinite dimensional. For example, if k is a Gaussian kernel then $\dim(\mathcal{H}) = \infty$ (e.g. [27].) Moreover, in this case unless $\mathbf{x}_t = \mathbf{x}_i$ for some $i < t$, the feature vector $\boldsymbol{\phi}(\mathbf{x}_t)$ will be linearly independent of $\{\boldsymbol{\phi}(\mathbf{x}_i)\}_{i=1}^{t-1}$. The solution we propose is to relax the requirement that $\boldsymbol{\phi}(\mathbf{x}_t)$ can be exactly written as a sum of $\{\boldsymbol{\phi}(\mathbf{x}_i)\}_{i=1}^{t-1}$ and to consider instead *approximate* linear dependency. Given a new sample $\mathbf{x}_t$, we will distinguish between two cases. In the first case, the sample is approximately dependent of past samples. Such a sample will be considered only through its effect on the estimate $\hat{f}$. A sample whose feature vector is not approximately dependent on past samples will be admitted into a "dictionary." From the RLS point of view, our approach is based on an on-line projection of the feature vectors encountered during training to a low dimensional subspace spanned by a small subset of the training samples – the dictionary.

Next, in Section 2.1 we describe the on-line sparsification algorithm in detail. Section 2.2 proves and discusses some desirable theoretical properties of the algorithm, while Section 2.3 illuminates its connection to kernel PCA. In Section 2.4 we overview current sparsification methods and compare our method to them.

2.1 The Sparsification Procedure

The on-line prediction setup assumes we sequentially sample a stream of input/output pairs $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots\}$, $\mathbf{x}_i \in \mathcal{X}$, $y_i \in \mathbb{R}$. Assume that at time step t, after having observed $t-1$ training samples $\{\mathbf{x}_i\}_{i=1}^{t-1}$, we have collected a dictionary consisting of a subset of the training samples $\mathcal{D}_{t-1} = \{\tilde{\mathbf{x}}_j\}_{j=1}^{m_{t-1}}$, where by construction $\{\boldsymbol{\phi}(\tilde{\mathbf{x}}_j)\}_{j=1}^{m_{t-1}}$ are linearly independent feature vectors. Now we are presented with a new sample $\mathbf{x}_t$. We test whether $\boldsymbol{\phi}(\mathbf{x}_t)$ is approximately linearly dependent on the dictionary vectors. If not, we add it to the dictionary. Consequently, all training samples up to time t can be approximated as linear combinations of the vectors in $\mathcal{D}_t$.

To avoid adding the training sample $\mathbf{x}_t$ to the dictionary, we need to find coefficients $\boldsymbol{a} = (a_1, \ldots, a_{m_{t-1}})^\top$ satisfying the approximate linear dependence (ALD) condition

$$\delta_t \overset{\text{def}}{=} \min_{\boldsymbol{a}} \left\| \sum_{j=1}^{m_{t-1}} a_j \boldsymbol{\phi}(\tilde{\mathbf{x}}_j) - \boldsymbol{\phi}(\mathbf{x}_t) \right\|^2 \leq \nu \ , \tag{2.2}$$

where ν is an accuracy parameter determining the level of sparsity. If the ALD condition in (2.2) holds, $\boldsymbol{\phi}(\mathbf{x}_t)$ can be approximated within a squared error ν by some linear combination of current dictionary members. Performing the minimization in (2.2) we can simultaneously check whether this condition is satisfiable and obtain the optimal coefficient vector $\tilde{\boldsymbol{a}}_t$. Expanding (2.2) we note that it may be written entirely in terms of inner products (in $\mathcal{H}$) between feature vectors $\boldsymbol{\phi}(\cdot)$,

$$\delta_t = \min_{\boldsymbol{a}} \Big\{ \sum_{i,j=1}^{m_{t-1}} a_i a_j \big\langle \boldsymbol{\phi}(\tilde{\mathbf{x}}_i), \boldsymbol{\phi}(\tilde{\mathbf{x}}_j) \big\rangle - 2 \sum_{j=1}^{m_{t-1}} a_j \big\langle \boldsymbol{\phi}(\tilde{\mathbf{x}}_j), \boldsymbol{\phi}(\mathbf{x}_t) \big\rangle + \big\langle \boldsymbol{\phi}(\mathbf{x}_t), \boldsymbol{\phi}(\mathbf{x}_t) \big\rangle \Big\}.$$

We employ the kernel trick by replacing the inner product between feature space vectors with the kernel defined over pairs of vectors in input space. We therefore make the substitution $\big\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}') \big\rangle = k(\mathbf{x}, \mathbf{x}')$, obtaining

$$\delta_t = \min_{\boldsymbol{a}} \Big\{ \boldsymbol{a}^\top \tilde{\mathbf{K}}_{t-1} \boldsymbol{a} - 2\boldsymbol{a}^\top \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) + k_{tt} \Big\}, \tag{2.3}$$

where $[\tilde{\mathbf{K}}_{t-1}]_{i,j} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$, $(\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t))_i = k(\tilde{\mathbf{x}}_i, \mathbf{x}_t)$, $k_{tt} = k(\mathbf{x}_t, \mathbf{x}_t)$, with $i, j = 1, \ldots, m_{t-1}$. Solving (2.3) yields the optimal $\tilde{\boldsymbol{a}}_t$ and the ALD condition

$$\tilde{\boldsymbol{a}}_t = \tilde{\mathbf{K}}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t), \quad \delta_t = k_{tt} - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top \tilde{\boldsymbol{a}}_t \le \nu, \tag{2.4}$$

respectively. If $\delta_t > \nu$ then we must expand the current dictionary by augmenting it with $\mathbf{x}_t$: $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}$ and $m_t = m_{t-1} + 1$. Using the expanded dictionary, $\boldsymbol{\phi}(\mathbf{x}_t)$ may be exactly represented (by itself) and δ_t is therefore set to zero.

Consequently, for every time-step i up to t we have

$$\boldsymbol{\phi}(\mathbf{x}_i) = \sum_{j=1}^{m_i} a_{i,j} \boldsymbol{\phi}(\tilde{\mathbf{x}}_j) + \boldsymbol{\phi}_i^{res}, \quad (\|\boldsymbol{\phi}_i^{res}\|^2 \le \nu), \tag{2.5}$$

and $\boldsymbol{\phi}_i^{res}$ denotes the residual component vector. By choosing ν to be sufficiently small we can make the approximation error in $\boldsymbol{\phi}(\mathbf{x}_i) \approx \sum_{j=1}^{m_i} a_{i,j} \boldsymbol{\phi}(\tilde{\mathbf{x}}_j)$ correspondingly small. The corresponding approximation in terms of kernel matrices is $\mathbf{K}_t \approx \mathbf{A}_t \tilde{\mathbf{K}}_t \mathbf{A}_t^\top$, where $[\mathbf{K}_t]_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$ with $i, j = 1, \ldots, t$ is the full kernel matrix and $[\mathbf{A}_t]_{i,j} = a_{i,j}$. Note that due to the sequential nature of the algorithm, $[\mathbf{A}_t]_{i,j} = 0$ for $j > m_i$. In practice, we will freely make the substitution $\mathbf{K}_t = \mathbf{A}_t \tilde{\mathbf{K}}_t \mathbf{A}_t^\top$ with the understanding that the resulting expressions are approximate whenever $\nu > 0$.

2.2 Properties of the Sparsification Method

Let us now study some properties of the sparsification method. We first show that under mild conditions on the data and the kernel function, the dictionary is finite. We then discuss the question how good the approximation really is, by showing that the sensitivity parameter ν controls how well the true kernel matrix is approximated.

Recall that the data points $\mathbf{x}_1, \mathbf{x}_2, \ldots$ are assumed to belong to some input set $\mathcal{X}$. Let $\boldsymbol{\phi}(\mathcal{X}) = \{\boldsymbol{\phi}(\mathbf{x}) : \mathbf{x} \in \mathcal{X}\} \subset \mathcal{H}$. The following theorem holds regardless of the dimensionality of $\mathcal{H}$, and essentially says that as long as $\mathcal{X}$ is compact, the set of dictionary vectors is finite.

Theorem 2.1. *Assume that (i) k is a continuous Mercer kernel and (ii) $\mathcal{X}$ is a compact subset of a Banach space. Then for any training sequence $\{\mathbf{x}_i\}_{i=1}^{\infty}$, and for any $\nu > 0$, the number of dictionary vectors is finite.*

Proof First, we claim that $\boldsymbol{\phi}$ is continuous. Given a sequence $\mathbf{z}_1, \mathbf{z}_2, \ldots$ of points in $\mathcal{X}$ such that $\mathbf{z}_i \to \mathbf{z}^*$ we have that $\|\boldsymbol{\phi}(\mathbf{z}_i) - \boldsymbol{\phi}(\mathbf{z}^*)\|_2^2 = \langle \boldsymbol{\phi}(\mathbf{z}_i), \boldsymbol{\phi}(\mathbf{z}_i)\rangle + \langle \boldsymbol{\phi}(\mathbf{z}^*), \boldsymbol{\phi}(\mathbf{z}^*)\rangle - \langle \boldsymbol{\phi}(\mathbf{z}_i), \boldsymbol{\phi}(\mathbf{z}^*)\rangle - \langle \boldsymbol{\phi}(\mathbf{z}^*), \boldsymbol{\phi}(\mathbf{z}_i)\rangle$. Writing this in terms of kernels we have $\|\boldsymbol{\phi}(\mathbf{z}_i) - \boldsymbol{\phi}(\mathbf{z}^*)\|_2^2 = k(\mathbf{z}_i, \mathbf{z}_i) + k(\mathbf{z}^*, \mathbf{z}^*) - k(\mathbf{z}_i, \mathbf{z}^*) - k(\mathbf{z}^*, \mathbf{z}_i)$. Since k itself is continuous we have that $\|\boldsymbol{\phi}(\mathbf{z}_i) - \boldsymbol{\phi}(\mathbf{z}^*)\|_2^2 \to 0$, so $\boldsymbol{\phi}$ is continuous.
We now recall several ideas from functional analysis. See [1] for precise definitions and applications in learning. An ℓ_2-norm based cover of $\boldsymbol{\phi}(\mathcal{X})$ at a scale of ε (an ε-cover) is a collection of ℓ_2-balls of radius ε, whose union contains $\boldsymbol{\phi}(\mathcal{X})$. The covering number at scale ε is the *minimal* number of balls of radius ε needed to cover the set. The packing number of the set is the *maximal* number of points from the set which are separated by an ℓ_2 distance larger than ε. Since $\mathcal{X}$ is compact and $\boldsymbol{\phi}$ is continuous, we conclude that $\boldsymbol{\phi}(\mathcal{X})$ is compact, implying that for any $\varepsilon > 0$ a finite ε-cover exists. Recall that the covering number is finite if, and only if, the packing number is finite (e.g. [1]), implying that the maximal number of ε-separated points in $\boldsymbol{\phi}(\mathcal{X})$ is finite. Next, observe that by construction of the algorithm, any two points $\boldsymbol{\phi}(\tilde{\mathbf{x}}_i)$ and $\boldsymbol{\phi}(\tilde{\mathbf{x}}_j)$ in the dictionary obey $\|\boldsymbol{\phi}(\tilde{\mathbf{x}}_i) - \boldsymbol{\phi}(\tilde{\mathbf{x}}_j)\|_2 > \sqrt{\nu}$, i.e. are $\sqrt{\nu}$-separated. Since the packing number of $\boldsymbol{\phi}(\mathcal{X})$ at scale $\sqrt{\nu}$ is finite we conclude that the number of such separated balls is also finite, implying that the dictionary is finite. ▮

Theorem 2.1 implies that after an initial period, the computational cost per time-step of the algorithm becomes independent of time and depends only on the dictionary size. It is this property that makes our framework practical for on-line real-time learning. Precise values for the size of the cover, and thus for the maximal size of the dictionary, can be obtained by making further assumptions on $\boldsymbol{\phi}$ (or the kernel k).

The next proposition assumes that $\mathcal{X} \subseteq \mathbb{R}^d$, and bounds the size of the dictionary as a function of ν.

Proposition 2.2. *Assume that (i) k is a Lipschitz continuous Mercer kernel on $\mathcal{X}$ and (ii) $\mathcal{X}$ is a compact subset of $\mathbb{R}^d$. Then there exist a constant C that depends on $\mathcal{X}$ and on the kernel function such that for any training sequence $\{\mathbf{x}_i\}_{i=1}^{\infty}$, and for any $\nu > 0$, the number of dictionary vectors N, satisfies $N \leq C\nu^{-d}$.*

Proof Since $\mathcal{X}$ is compact it is contained in $B_M(\mathbf{0})$ where $B_M(\mathbf{x})$ is the ℓ_2 ball of radius M around $\mathbf{x}$ and $M > 0$. The covering number of $\mathcal{X}$ at scale ε is bounded by $C_1\varepsilon^{-d}$ (e.g. [1]) for some C_1. The kernel k is Lipschitz continuous function of $\mathcal{X} \times \mathcal{X}$ implying that there exists a positive constant L such that $|k(\mathbf{x}, \mathbf{y}) - k(\mathbf{x}', \mathbf{y}')| \leq L\|\mathbf{x} - \mathbf{x}'\|_2 + L\|\mathbf{y} - \mathbf{y}'\|_2$. Now, suppose that a point $\mathbf{x} \in \mathcal{X}$ is admitted to the dictionary. As in Theorem 2.1, we know that a point $\mathbf{x}' \in \mathcal{X}$ will not be admitted to the dictionary if $\|\boldsymbol{\phi}(\mathbf{x}) - \boldsymbol{\phi}(\mathbf{x}')\|_2 \leq \sqrt{\nu}$. We claim that all the points in $B_{\nu/2L}(\mathbf{x})$ will not be admitted to the dictionary. Indeed, if $\mathbf{x}' \in B_{\nu/2L}(\mathbf{x})$ then $\|\boldsymbol{\phi}(\mathbf{x}) - \boldsymbol{\phi}(\mathbf{x}')\|_2^2 = |k(\mathbf{x},\mathbf{x}) + k(\mathbf{x}',\mathbf{x}') - k(\mathbf{x},\mathbf{x}') - k(\mathbf{x}',\mathbf{x})| \leq |k(\mathbf{x},\mathbf{x}) - k(\mathbf{x},\mathbf{x}')| + |k(\mathbf{x}',\mathbf{x}') - k(\mathbf{x}',\mathbf{x})| \leq 2L\|\mathbf{x} - \mathbf{x}'\|_2 \leq \nu$. We therefore have that the size of the dictionary, N, for a given ν cannot exceed the packing number of $\mathcal{X}$ at scale $\nu/2L$. Since the packing number at scale ε is upper bounded by the covering number at scale $\varepsilon/2$ ([1]) we have that $N \leq C_1(\nu/4L)^{-d} = C\nu^{-d}$, for $C = C_1(4L)^d$. ▮

We comment that similar results can be obtained if $\mathcal{X}$ cannot be embedded in $\mathbb{R}^d$, but a bound on the covering number is available. Furthermore, tighter bounds can be obtained by considering the eigenvalues of the kernel function. Since the behavior of the size of the dictionary is not fundamentally different, this was not pursued further.

After establishing that the dictionary is finite we turn to study the effect of the approximation level ν on the approximation of the kernel matrix $\mathbf{K}$. In order to simplify the analysis we first consider an *off-line* version of the algorithm on a finite data set of size t. In this case, the dictionary is first constructed in the usual manner, and then the optimal expansion coefficient matrix $\mathbf{A}$ is computed for the entire data set at once. We use essentially the same notation as before except that now every quantity depending on an incomplete dictionary is redefined to depend on the entire dictionary. In order to remind us of this change we omit the time index in each such quantity. For instance, $\mathcal{D}$, m, and $\mathbf{K}$ denote the dictionary, its size, and the full kernel matrix, respectively.

Defining the matrices $\mathbf{\Phi} = [\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_t)]$, $\tilde{\mathbf{\Phi}} = [\boldsymbol{\phi}(\tilde{\mathbf{x}}_1), \ldots, \boldsymbol{\phi}(\tilde{\mathbf{x}}_m)]$, and $\mathbf{\Phi}^{res} = [\boldsymbol{\phi}_1^{res}, \ldots, \boldsymbol{\phi}_t^{res}]$, we may write (2.5) for all samples concisely as

$$\mathbf{\Phi} = \tilde{\mathbf{\Phi}}\mathbf{A}^\top + \mathbf{\Phi}^{res}. \tag{2.6}$$

As before, the optimal expansion coefficients for the sample $\mathbf{x}_i$ are found by $\boldsymbol{a}_i = \tilde{\mathbf{K}}^{-1}\tilde{\mathbf{k}}(\mathbf{x}_i)$, and $[\mathbf{A}]_{i,j} = a_{i,j}$. Pre-multiplying (2.6) by its transpose we get

$$\mathbf{K} = \mathbf{A}\tilde{\mathbf{K}}\mathbf{A}^\top + \mathbf{\Phi}^{res\top}\mathbf{\Phi}^{res}. \tag{2.7}$$

The cross term $\mathbf{A}\tilde{\mathbf{\Phi}}^\top\mathbf{\Phi}^{res}$ and its transpose vanish due to the ℓ_2-orthogonality of the residuals to the subspace spanned by the dictionary vectors. Defining the residual kernel matrix by $\mathbf{R} = \mathbf{K} - \mathbf{A}\tilde{\mathbf{K}}\mathbf{A}^\top$ it can be easily seen that, in both the on-line and the off-line cases, $(\operatorname{diag}\mathbf{R})_i = \delta_i$. Further, in the off-line case $\mathbf{R} = \mathbf{\Phi}^{res\top}\mathbf{\Phi}^{res}$ and is therefore positive semi-definite. As a consequence, in the off-line case we may bound the norm of $\mathbf{R}$, thus justifying our approximation. Recall that for a matrix $\mathbf{R}$ the matrix 2-norm is defined as $\|\mathbf{R}\|_2 = \max_{\mathbf{u}:\|\mathbf{u}\|_2=1} \|\mathbf{R}\mathbf{u}\|_2$.

Proposition 2.3. *In the off-line case* $\|\mathbf{R}\|_2 \leq t\nu$.

Proof Let $\lambda_i^{\mathbf{R}}$ be the i-th eigenvalue of $\mathbf{R}$. We recall from linear algebra that $\|\mathbf{R}\|_2 = \max_i |\lambda_i^{\mathbf{R}}|$. Since $(\operatorname{diag}\mathbf{R})_i = \delta_i$, we have that $\sum_{i=1}^t \lambda_i^{\mathbf{R}} = \operatorname{Tr}\mathbf{R} = \sum_{i=1}^t \delta_i$. Moreover, since $\mathbf{R}$ is positive semi-definite $\lambda_i^{\mathbf{R}} \geq 0$ for all i. Therefore $\|\mathbf{R}\|_2 = \max_i \lambda_i^{\mathbf{R}} \leq \sum_{i=1}^t \delta_i \leq t\nu$. ▮

As a corollary, a similar bound can be placed on the covariance of the residuals $\|\mathbf{C}^{res}\| = \frac{1}{t}\|\mathbf{\Phi}^{res}\mathbf{\Phi}^{res\top}\| = \max_i \lambda_i^{\mathbf{R}}/t \leq \sum_{i=1}^t \delta_i/t \leq \nu$.

Considering the on-line case, it is clear that once the dictionary stops growing, for the remaining samples the algorithm behaves exactly like its off-line counterpart. Using Theorem 2.1 we know that if $\mathcal{X}$ is compact for any $\nu > 0$ the dictionary is finite, and therefore a bound similar to that of Proposition 2.3 holds beyond some finite time. More specifically, in the on-line case we have $\|\mathbf{R}\|_2 \leq t\nu + B$, where B is a random positive constant accounting for the size of the residuals up to the point when the dictionary reaches its final size. The corresponding bound for the residual covariance is therefore $\|\mathbf{C}^{res}\| \leq \nu + \frac{B}{t}$.

2.3 On-Line Sparsification as Approximate PCA

The on-line sparsification method described above has close connections to kernel PCA [28]. PCA is the optimal unsupervised dimensionality reduction method, in the mean squared error sense, and is used in signal processing, machine learning and other fields involving high dimensional data. PCA is often used to remove noise from data. In such applications the noisy data is projected onto the first principal directions (i.e. on the subspace spanned by the first eigenvectors of the data's covariance matrix, where the eigenvectors are ordered by non-increasing eigenvalues), with the implicit assumption that the variance in the remaining directions is due to noise. In [28] it was shown that solving the eigenvalue problem of the kernel (Gram) matrix $\mathbf{K}_t = \mathbf{\Phi}_t^\top \mathbf{\Phi}_t$, is essentially equivalent to performing PCA of the data in the feature space $\mathcal{H}$. In this section we show that our sparsification method is essentially an approximate form of PCA in the feature space.

We first describe the optimal dimensionality reduction procedure. Let the covariance matrix at time t, be $\mathbf{C}_t \stackrel{\text{def}}{=} \frac{1}{t}\mathbf{\Phi}_t\mathbf{\Phi}_t^\top$. $\mathbf{C}_t$ has (at most) t positive eigenvalues $\{\lambda_1, \ldots, \lambda_t\}$ and a corresponding orthonormal set of eigenvectors $\{\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_t\}$, forming an orthogonal basis for the subspace of $\mathcal{H}$ that contains $\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_t)$. Defining a projection operator $P_\mathcal{S}$ onto the span of the set $\Psi_\mathcal{S} = \{\boldsymbol{\psi}_i : i \in \mathcal{S} \subseteq \{1, \ldots, t\}\}$, it is well known that projecting the training data onto the subspace spanned by the eigenvectors in $\Psi_\mathcal{S}$, entails a mean squared error of $\frac{1}{t}\sum_{i=1}^t \|(I - P_\mathcal{S})\boldsymbol{\phi}(x_i)\|^2 = \sum_{i \notin \mathcal{S}} \lambda_i$. An immediate consequence of this is that an optimal m dimensional projection, with respect to the mean squared error criterion, is one in which $\Psi_\mathcal{S}$ consists of the first m eigenvectors (i.e. $\mathcal{S} = \{1, \ldots, m\}$).

Similarly, we can also define a projection operator onto the span of the dictionary vectors found by our sparsification method. Let $P_\mathcal{D} \stackrel{\text{def}}{=} \tilde{\mathbf{\Phi}}\left(\tilde{\mathbf{\Phi}}^\top\tilde{\mathbf{\Phi}}\right)^{-1}\tilde{\mathbf{\Phi}}^\top$ denote this projection operator. Using the ALD condition (2.4) we can place a bound on the mean squared error due to the projection $P_\mathcal{D}$: $\frac{1}{t}\sum_{i=1}^t \|\boldsymbol{\phi}(\mathbf{x}_i) - P_\mathcal{D}(\boldsymbol{\phi}(\mathbf{x}_i))\|^2 = \frac{1}{t}\sum_{i=1}^t \|(I - P_\mathcal{D})\boldsymbol{\phi}(\mathbf{x}_i)\|^2 = \frac{1}{t}\sum_{i=1}^t \delta_i \leq ((t - m_t)/t)\nu \leq \nu$, where the first inequality is due to the fact that for the m_t dictionary vectors the error is zero. However, since the size of the dictionary is not known a priori, the second inequality provides a more useful bound.

In the preceding paragraph we showed that the mean squared error of the projection performed by the on-line sparsification procedure is bounded by ν. We now show that the projection performed by the sparsification method, essentially keeps all the "important" eigenvectors that are used in the optimal PCA projection.

Theorem 2.4. *The i-th normalized eigenvector* $\boldsymbol{\psi}_i$ *of the empirical covariance matrix* $\mathbf{C} = \frac{1}{t}\mathbf{\Phi}\mathbf{\Phi}^\top$*, with eigenvalue* $\lambda_i > 0$*, satisfies* $\|P_\mathcal{D}\boldsymbol{\psi}_i\|^2 \geq 1 - \nu/\lambda_i$*.*

Proof Any $\boldsymbol{\psi}_i$ for which $\lambda_i > 0$ may be expanded in terms of the feature vectors corresponding to the data points. It is well known (e.g. [28]) that every respective expansion coefficient vector is itself an eigenvector of the kernel matrix $\mathbf{K}$ with eigenvalue $t\lambda_i$. Therefore we may write $\boldsymbol{\psi}_i = \frac{1}{\sqrt{t\lambda_i}}\mathbf{\Phi}\mathbf{u}_i$ where $\mathbf{K}\mathbf{u}_i = t\lambda_i\mathbf{u}_i$ and $\|\mathbf{u}_i\| = 1$. Substituting into $\|P_\mathcal{D}\boldsymbol{\psi}_i\|^2$ we have $\|P_\mathcal{D}\boldsymbol{\psi}_i\|^2 = \boldsymbol{\psi}_i^\top P_\mathcal{D}\boldsymbol{\psi}_i = \frac{1}{t\lambda_i}\mathbf{u}_i^\top\mathbf{\Phi}^\top P_\mathcal{D}\mathbf{\Phi}\mathbf{u}_i$. Recalling that $P_\mathcal{D}$ is the projection operator onto the span of the dictionary we have $P_\mathcal{D}\boldsymbol{\phi}(\mathbf{x}_i) = \tilde{\mathbf{\Phi}}\boldsymbol{a}_i$ and $P_\mathcal{D}\mathbf{\Phi} = \tilde{\mathbf{\Phi}}\mathbf{A}^\top$. Therefore, $\mathbf{\Phi}^\top P_\mathcal{D}\mathbf{\Phi} = \mathbf{A}\tilde{\mathbf{K}}\mathbf{A}^\top$ and

$$\begin{aligned}\|P_\mathcal{D}\boldsymbol{\psi}_i\|^2 &= \frac{1}{t\lambda_i}\mathbf{u}_i^\top\mathbf{A}\tilde{\mathbf{K}}\mathbf{A}^\top\mathbf{u}_i = \frac{1}{t\lambda_i}\mathbf{u}_i^\top(\mathbf{K}-\mathbf{R})\mathbf{u}_i \\ &= 1 - \frac{1}{t\lambda_i}\mathbf{u}_i^\top\mathbf{R}\mathbf{u}_i \geq 1 - \frac{\|\mathbf{R}\|_2}{t\lambda_i} \geq 1 - \frac{\nu}{\lambda_i},\end{aligned}$$

where the last inequality is due to Proposition 2.3. ∎

Theorem 2.4 establishes the connection between our sparsification algorithm and kernel PCA, since it implies that eigenvectors whose eigenvalues are significantly larger than ν are projected almost in their entirety onto the span of the dictionary vectors, and the quality of the approximation improves linearly with the ratio λ_i/ν. In comparison, kernel PCA projects the data solely onto the span of the first few eigenvectors of $\mathbf{C}$. We are therefore led to regard our sparsification method as an approximate form of kernel PCA, with the caveat that our method does not diagonalize $\mathbf{C}$, and so cannot extract individual orthogonal features, as kernel PCA does. Computationally however, our method is significantly cheaper ($O(m^2)$ memory and $O(tm^2)$ time) than exact kernel PCA ($O(t^2)$ memory and $O(t^2p)$ time where p is the number of extracted components).

2.4 Comparison to Other Sparsification Schemes

Several approaches to sparsification of kernel-based solutions have been proposed in the literature. As already mentioned above, SVMs for classification and regression achieve sparsity by utilizing error insensitive cost functions. The solution produced by an SVM consists of a linear combination of kernel evaluations, one per training sample, where typically a (sometimes large) fraction of the combination coefficients vanish. This approach has several major disadvantages. First, with a training set of t samples, during learning the SVM algorithm must maintain a matrix of size $t \times t$ and update a full set of t coefficients. This means that, even if the end result turns out to be very sparse, the training algorithm will not be able to take full advantage of this sparsity in terms of efficiency. As a consequence, even the current state-of-the art SVM algorithm scales super-linearly in t [4]. Second, in SVMs the solution's sparsity depends on the level of noise in the training data; this effect is especially pronounced in the case of regression. Finally, SVM solutions are known to be non-maximally sparse. This is due to the special form of the SVM quadratic optimization problem, in which the constraints limit the level of sparsity attainable [23].

Shortly after the introduction of SVMs to the machine learning community it was realized [2] that the solutions provided by SVMs, both for classification and regression, may often be made significantly sparser, without altering the solutions' weight vectors. It was also shown in [2] and later in [3] that additional sparsity may be attained by allowing small changes to be made in the SVM solution, with little or no degradation in generalization ability. Burges' idea is based on using a "reduced-set" of feature space vectors to approximate the weight vector of the original solution. In Burges' method the reduced-set of feature vectors, apart from its size, is virtually unconstrained and therefore the algorithmic complexity of finding reduced-set solutions is rather high, posing a major obstacle to the widespread use of this method. In [26] and [8] it was suggested that restricting the reduced set to be a subset of the training samples would help alleviate the computational cost associated with the original reduced-set method. This was backed by empirical results on several problems including handwritten digit recognition. All of these reduced-set methods achieve sparsity by elimination, meaning that they are best used as a post-processing stage, after a kernel solution is obtained from some main algorithm (e.g. SVM) whose level of sparsity is deemed unsatisfactory by the user. For a more thorough account of reduced-set methods see Chapter 18 of [27].

Ultimately, reduced-set methods are based on the identification of (approximate) linear dependencies between feature space vectors and their subsequent elimination. Based on the same underlying principle, another class, of "sparse-greedy" methods, aim at greedily constructing a non-redundant set of feature vectors starting with an initially empty set, rather than a full solution [31, 30, 44] (see also Chapter 10 of [27]). These methods are also closely related to the kernel Matching Pursuit algorithm [39]. It should be noted that the reason why greedy strategies are resorted to is due to a general hardness result regarding the problem of finding the best subset of samples in a sparse approximation framework [22], along with some positive results concerning the convergence rates for sparse greedy algorithms [22, 44].

Greedy methods represent the opposite extreme to reduced-set methods along the elimination-construction axis of sparsification methods. Another (orthogonal) dimension along which sparsification methods may be measured is the one differentiating between supervised and unsupervised sparsification. Supervised sparsification is geared toward optimizing a supervised error criterion (e.g. the mean-squared error in regression tasks), while unsupervised sparsification attempts to faithfully reproduce the the images of input samples in feature space. Examples for supervised sparsification are [2, 34, 30, 39], of which [34] is unique in that it aims at achieving sparsity by taking a Bayesian approach in which a prior favoring sparse solutions is employed [2]. In [30] a greedy sparsification method is suggested that is

[2] It should be noted that support vector machines may also be cast within a probabilistic Bayesian framework, see [32].

specific to Gaussian Process regression and is similar to kernel Matching Pursuit [39].

Examples of unsupervised sparsification are [31, 43, 11, 35]. In [31] a randomized-greedy selection strategy is used to reduce the rank of the kernel matrix $\mathbf{K}$ while [43] uses a purely random strategy based on the Nyström method to achieve the same goal. In [11] the incomplete Cholesky factorization algorithm is used to yield yet another reduced-rank approximation to $\mathbf{K}$. Employing low-rank approximations to $\mathbf{K}$ is essentially equivalent to using low-dimensional approximations of the feature vectors corresponding to the training samples. As principal component analysis (PCA) is known to deliver the optimal unsupervised dimensionality reduction for the mean-squared reconstruction error criterion, it is therefore natural to turn to kernel PCA [28] as a sparsification device. Indeed, many of the unsupervised methods mentioned above are closely related to kernel PCA. In [35] a sparse variant of kernel PCA is proposed, based on a Gaussian generative model. The general idea in both cases is to project the entire feature space onto the low dimensional manifold spanned by the first eigenvectors of the sample covariance in feature space, corresponding to the leading/non-zero eigenvalues.

While many of the sparsification methods discussed above are constructive in nature, progressively building increasingly richer representations with time, they are not applicable to the on-line setting. In this setting input samples are not randomly accessible, instead they are given as a stream of data in which only one sample may be observed at any given time. This imposes an additional constraint on any sparsification method that attempts to represent the entire training history using some representative sample. Namely, at any point in time the algorithm must decide whether to add the current sample to its representation, or discard it. This problem of on-line sparsification is not as well studied in the kernel-methods community, and is the one we address with our sparsification algorithm.

Our method is most closely related to a sparsification method used By Csató and Opper [6, 7] in the context of learning with Gaussian Processes [13, 42]. Csató and Opper's method also incrementally constructs a dictionary of input samples on which all other data are projected (with the projection performed in the feature space $\mathcal{H}$). However, while in our method the criterion used to decide whether a sample should be added to the dictionary is based only on the distance (in $\mathcal{H}$) between the new sample and the span of previously stored dictionary samples, their method also takes into account the estimate of the regressor (or classifier) on the new point and its difference from the target value. Consequently, the dictionary constructed by their method depends on the function being estimated and on the sample noise, while our method disregards these completely. We defer further discussion of the differences between these two methods to Section 6.

3 The Kernel RLS Algorithm

The RLS algorithm is used to incrementally train a linear regression model, parameterized by a weight vector $\mathbf{w}$, of the form $\hat{f}(\mathbf{x}) = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle$, where, as before, $\boldsymbol{\phi}(\mathbf{x})$ is the feature vector associated with the input $\mathbf{x}$. We assume that a standard preprocessing step has been

Table 1: The Kernel RLS Algorithm. On the right we bound the number of operations per time-step for each line of the pseudo-code. The overall per time-step computational cost is bounded by $O(m^2)$ (we assume that kernel evaluations require $O(1)$ time)

```
Parameter: $\nu$                                                                         Cost
Initialize: $\tilde{\mathbf{K}}_1 = [k_{11}]$, $\tilde{\mathbf{K}}_1^{-1} = [1/k_{11}]$,
        $\boldsymbol{\alpha}_1 = (y_1/k_{11})$, $\mathbf{P}_1 = [1]$, $m = 1$
for $t = 2, 3 \ldots$
  1. Get new sample: $(\mathbf{x}_t, y_t)$
  2. Compute $\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$                                     $O(m)$
  3. ALD test:
    $\boldsymbol{a}_t = \tilde{\mathbf{K}}_{t-1}^{-1}\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$                        $O(m^2)$
    $\delta_t = k_{tt} - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top \boldsymbol{a}_t$                        $O(m)$
    if $\delta_t > \nu$   % add $\mathbf{x}_t$ to dictionary                                   $O(1)$
      $\mathcal{D}_t = \{\mathcal{D}_{t-1} \cup \{\tilde{\mathbf{x}}_t\}\}$                                      $O(1)$
      Compute $\tilde{\mathbf{K}}_t^{-1}$ (3.14) with $\tilde{\boldsymbol{a}}_t = \boldsymbol{a}_t$              $O(m^2)$
      $\boldsymbol{a}_t = (\mathbf{0}, \ldots, 1)^\top$                                          $O(m)$
      Compute $\mathbf{P}_t$ (3.15)                                                        $O(m)$
      Compute $\boldsymbol{\alpha}_t$ (3.16)                                               $O(m)$
      $m := m + 1$                                                                       $O(1)$
    else   % dictionary unchanged
      $\mathcal{D}_t = \mathcal{D}_{t-1}$
      $\mathbf{q}_t = \frac{\mathbf{P}_{t-1}\boldsymbol{a}_t}{1+\boldsymbol{a}_t^\top \mathbf{P}_{t-1}\boldsymbol{a}_t}$                   $O(m^2)$
      Compute $\mathbf{P}_t$ (3.12)                                                        $O(m^2)$
      Compute $\boldsymbol{\alpha}_t$ (3.13)                                               $O(m^2)$
Output: $\mathcal{D}_t$, $\boldsymbol{\alpha}_t$
```

performed in order to absorb the bias term into the weight vector $\mathbf{w}$ (i.e. by redefining $\mathbf{w}$ as $(\mathbf{w}^\top, b)^\top$ and $\boldsymbol{\phi}$ as $(\boldsymbol{\phi}^\top, 1)^\top$); see [9] for details. In the simplest form of the RLS algorithm we minimize at each time step t the sum of the squared errors:

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{t} \left(\hat{f}(\mathbf{x}_i) - y_i\right)^2 = \|\boldsymbol{\Phi}_t^\top \mathbf{w} - \mathbf{y}_t\|^2, \tag{3.8}$$

where we have defined the vector $\mathbf{y}_t = (y_1, \ldots, y_t)^\top$. Ordinarily, we would minimize (3.8) with respect to $\mathbf{w}$ and obtain $\mathbf{w}_t = \operatorname{argmin}_{\mathbf{w}} \|\boldsymbol{\Phi}_t^\top \mathbf{w} - \mathbf{y}_t\|^2 = \left(\boldsymbol{\Phi}_t^\top\right)^\dagger \mathbf{y}_t$, where $\left(\boldsymbol{\Phi}_t^\top\right)^\dagger$ is the pseudo-inverse of $\boldsymbol{\Phi}_t^\top$. The classic RLS algorithm [17], based on the *matrix inversion lemma* allows one to minimize the loss $\mathcal{L}(\mathbf{w})$ recursively and on-line without recomputing the matrix $\left(\boldsymbol{\Phi}_t^\top\right)^\dagger$ at each step.

As mentioned above, the feature space may be of very high dimensionality, rendering the handling and manipulation of matrices such as $\boldsymbol{\Phi}_t$ and $\boldsymbol{\Phi}_t\boldsymbol{\Phi}_t^\top$ prohibitive. Fortunately, as can be easily verified[3], we may express the optimal weight vector as

$$\mathbf{w}_t = \sum_{i=1}^{t} \alpha_i \boldsymbol{\phi}(\mathbf{x}_i) = \mathbf{\Phi}_t \boldsymbol{\alpha} \ , \tag{3.9}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_t)^\top$. Substituting into (3.8), slightly abusing notation, we have

$$\mathcal{L}(\boldsymbol{\alpha}) = \|\mathbf{K}_t \boldsymbol{\alpha} - \mathbf{y}_t\|^2 \ . \tag{3.10}$$

Theoretically, the minimizer of (3.10) is given by $\boldsymbol{\alpha}_t = \mathbf{K}_t^\dagger \mathbf{y}_t$, which can be computed recursively using the classic RLS algorithm. The problem with this approach is threefold. First, for large datasets simply maintaining $\mathbf{K}$ in memory, estimating the coefficient vector $\boldsymbol{\alpha}$ and evaluating new points could prove prohibitive both in terms of space and time. Second, the order of the model produced (i.e. the size of the vector $\boldsymbol{\alpha}$, which will be dense in general), would be equal to the number of training samples, causing severe overfitting. Third, in many cases the eigenvalues of the matrix $\mathbf{K}_t$ decay rapidly to 0, which means that inverting it would be numerically unstable.

By making use of the sparsification method as described in the preceding section we can overcome these shortcomings. The basic idea is to use the smaller (sometimes much smaller) set of dictionary samples $\mathcal{D}_t$ in the expansion of the weight vector $\mathbf{w}_t$, instead of the entire training set.

Using (2.6) in (3.9) we have $\mathbf{w}_t = \mathbf{\Phi}_t \boldsymbol{\alpha}_t \approx \tilde{\mathbf{\Phi}}_t \mathbf{A}_t^\top \boldsymbol{\alpha}_t = \tilde{\mathbf{\Phi}}_t \tilde{\boldsymbol{\alpha}}_t$, where $\tilde{\boldsymbol{\alpha}}_t \overset{\text{def}}{=} \mathbf{A}_t^\top \boldsymbol{\alpha}_t$ is a vector of m "reduced" coefficients. The loss becomes

$$\mathcal{L}(\tilde{\boldsymbol{\alpha}}) = \|\mathbf{\Phi}_t^\top \tilde{\mathbf{\Phi}}_t \tilde{\boldsymbol{\alpha}} - \mathbf{y}_t\|^2 = \|\mathbf{A}_t \tilde{\mathbf{K}}_t \tilde{\boldsymbol{\alpha}} - \mathbf{y}_t\|^2 \ , \tag{3.11}$$

and its minimizer is $\tilde{\boldsymbol{\alpha}}_t = (\mathbf{A}_t \tilde{\mathbf{K}}_t)^\dagger \mathbf{y}_t = \tilde{\mathbf{K}}_t^{-1} \mathbf{A}_t^\dagger \mathbf{y}_t$.

In the on-line scenario, at each time step t we are faced with either one of the following two cases:

1. $\boldsymbol{\phi}(\mathbf{x}_t)$ is ALD on $\mathcal{D}_{t-1}$, i.e. $\delta_t \le \nu$ and $\boldsymbol{a}_t$ are given by (2.4). In this case $\mathcal{D}_t = \mathcal{D}_{t-1}$, and consequently $m_t = m_{t-1}$ and $\tilde{\mathbf{K}}_t = \tilde{\mathbf{K}}_{t-1}$.

2. $\delta_t > \nu$, therefore $\boldsymbol{\phi}(\mathbf{x}_t)$ is not ALD on $\mathcal{D}_{t-1}$. $\mathbf{x}_t$ is added to the dictionary, i.e. $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}$, and therefore $m_t = m_{t-1} + 1$ and $\tilde{\mathbf{K}}_t$ grows accordingly.

We now derive the KRLS update equations for each of these cases.

Case 1 Here, only $\mathbf{A}$ changes between time steps: $\mathbf{A}_t = [\mathbf{A}_{t-1}^\top, \boldsymbol{a}_t]^\top$. Therefore $\mathbf{A}_t^\top \mathbf{A}_t = \mathbf{A}_{t-1}^\top \mathbf{A}_{t-1} + \boldsymbol{a}_t \boldsymbol{a}_t^\top$, and $\mathbf{A}_t^\top \mathbf{y}_t = \mathbf{A}_{t-1}^\top \mathbf{y}_{t-1} + \boldsymbol{a}_t y_t$. Note that $\tilde{\mathbf{K}}_t$ is unchanged. Defining $\mathbf{P}_t = (\mathbf{A}_t^\top \mathbf{A}_t)^{-1}$ we apply the matrix inversion Lemma (e.g. [25]) to obtain a recursive formula for $\mathbf{P}_t$:

$$\mathbf{P}_t = \mathbf{P}_{t-1} - \frac{\mathbf{P}_{t-1} \boldsymbol{a}_t \boldsymbol{a}_t^\top \mathbf{P}_{t-1}}{1 + \boldsymbol{a}_t^\top \mathbf{P}_{t-1} \boldsymbol{a}_t} \ . \tag{3.12}$$

[3] Simply add to $\mathbf{w}_t$ some vector $\bar{\mathbf{w}}$ orthogonal to $\{\boldsymbol{\phi}(\mathbf{x}_i)\}_{i=1}^t$ and substitute into (3.8).

Defining $\mathbf{q}_t = \frac{\mathbf{P}_{t-1}\boldsymbol{a}_t}{1+\boldsymbol{a}_t^\top \mathbf{P}_{t-1}\boldsymbol{a}_t}$ we derive the KRLS update rule for $\tilde{\boldsymbol{\alpha}}$:

$$\begin{aligned}\tilde{\boldsymbol{\alpha}}_t &= \tilde{\mathbf{K}}_t^{-1}\mathbf{P}_t\mathbf{A}_t^\top \mathbf{y}_t \\ &= \tilde{\mathbf{K}}_t^{-1}\left(\mathbf{P}_{t-1} - \mathbf{q}_t\boldsymbol{a}_t^\top \mathbf{P}_{t-1}\right)\left(\mathbf{A}_{t-1}^\top \mathbf{y}_{t-1} + \boldsymbol{a}_t y_t\right) \\ &= \tilde{\boldsymbol{\alpha}}_{t-1} + \tilde{\mathbf{K}}_t^{-1}\Big(\mathbf{P}_t\boldsymbol{a}_t y_t - \mathbf{q}_t\boldsymbol{a}_t^\top \tilde{\mathbf{K}}_t\tilde{\boldsymbol{\alpha}}_{t-1}\Big) \\ &= \tilde{\boldsymbol{\alpha}}_{t-1} + \tilde{\mathbf{K}}_t^{-1}\mathbf{q}_t\Big(y_t - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top \tilde{\boldsymbol{\alpha}}_{t-1}\Big), \qquad (3.13)\end{aligned}$$

where the last equality is based on $\mathbf{q}_t = \mathbf{P}_t\boldsymbol{a}_t$, and $\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) = \tilde{\mathbf{K}}_t\boldsymbol{a}_t$.

<u>Case 2</u> Here, $\mathbf{K}_t \neq \mathbf{K}_{t-1}$, but a recursive formula for $\tilde{\mathbf{K}}_t^{-1}$ is easily derived:

$$\begin{aligned}\tilde{\mathbf{K}}_t &= \begin{bmatrix} \tilde{\mathbf{K}}_{t-1} & \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) \\ \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top & k_{tt} \end{bmatrix} \Rightarrow \\ \tilde{\mathbf{K}}_t^{-1} &= \frac{1}{\delta_t}\begin{bmatrix} \delta_t\tilde{\mathbf{K}}_{t-1}^{-1} + \tilde{\boldsymbol{a}}_t\tilde{\boldsymbol{a}}_t^\top & -\tilde{\boldsymbol{a}}_t \\ -\tilde{\boldsymbol{a}}_t^\top & 1 \end{bmatrix}, \qquad (3.14)\end{aligned}$$

where $\tilde{\boldsymbol{a}}_t = \tilde{\mathbf{K}}_{t-1}^{-1}\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$. Note that $\tilde{\boldsymbol{a}}_t$ equals the $\boldsymbol{a}_t$ computed for the ALD test (2.4), so there is no need to recompute $\tilde{\mathbf{K}}_{t-1}^{-1}\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$. Furthermore, $\boldsymbol{a}_t = (0,\ldots,1)^\top$ since $\boldsymbol{\phi}(\mathbf{x}_t)$ is exactly representable by itself. Therefore

$$\begin{aligned}\mathbf{A}_t &= \begin{bmatrix} \mathbf{A}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix}, \ \mathbf{A}_t^\top \mathbf{A}_t = \begin{bmatrix} \mathbf{A}_{t-1}^\top \mathbf{A}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix}, \\ \mathbf{P}_t &= \begin{bmatrix} \mathbf{P}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix}, \qquad (3.15)\end{aligned}$$

where $\mathbf{0}$ is a vector of zeros of appropriate length. The KRLS update rule for $\tilde{\boldsymbol{\alpha}}$ is:

$$\begin{aligned}\tilde{\boldsymbol{\alpha}}_t &= \tilde{\mathbf{K}}_t^{-1}(\mathbf{A}_t^\top \mathbf{A}_t)^{-1}\mathbf{A}_t^\top \mathbf{y}_t \\ &= \tilde{\mathbf{K}}_t^{-1}\begin{bmatrix} (\mathbf{A}_{t-1}^\top \mathbf{A}_{t-1})^{-1}\mathbf{A}_{t-1}^\top \mathbf{y}_{t-1} \\ y_t \end{bmatrix} \\ &= \begin{bmatrix} \tilde{\boldsymbol{\alpha}}_{t-1} - \frac{\tilde{\boldsymbol{a}}_t}{\delta_t}(y_t - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top \tilde{\boldsymbol{\alpha}}_{t-1}) \\ \frac{1}{\delta_t}(y_t - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top \tilde{\boldsymbol{\alpha}}_{t-1}) \end{bmatrix}, \qquad (3.16)\end{aligned}$$

where for the final equality we used $\tilde{\boldsymbol{a}}_t^\top \tilde{\mathbf{K}}_{t-1} = \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top$. The algorithm in pseudo-code form is described in Table 1.

4 A Generalization Bound

In this section we present a data-dependent generalization bound whose generality makes it applicable to the KRLS algorithm, as well as to many other regression and classification algorithms. Most of the currently available bounds for classification and regression assume a bounded loss function. While this assumption is often acceptable for classification, this is clearly not the case for regression. Recently a generalization error bound for unbounded loss functions was established in [20], where the boundedness assumption is replaced by a moment condition. Theorem 4.1 below is a slightly revised version of Theorem 8 in [20].

We quote the general theorem, and then apply it to the specific case of the squared loss studied in this work. For each function f, consider the loss function

$$\ell_f(\mathbf{x}, y) = \ell(y, f(\mathbf{x}))$$

viewed as a function from $\mathcal{X} \times \mathcal{Y}$ to $\mathbb{R}$. In our the context of the present paper we may take $\ell(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$. Moreover, the function f is given by $f(\mathbf{x}) = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle$. Let $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ be a set of n IID samples drawn from some distribution $P(X, Y)$. Set $L(f) = \mathbf{E}_{X,Y}\ell_f(X, Y)$ and $\hat{L}(f) = \frac{1}{n}\sum_{i=1}^n \ell(y_i, f(\mathbf{x}_i))$. Furthermore, let

$$\mathcal{L}_{\mathcal{F}} = \{\ell_f(\mathbf{x}, y) : \ f \in \mathcal{F}\}$$

be a class of functions (defined on $\mathcal{X} \times \mathcal{Y}$).

In order to establish useful generalization bounds we introduce a classic complexity measure for a class of functions $\mathcal{F}$. Let $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$ be a sequence of independent and identically distributed $\{\pm 1\}$-valued random variables, such that $\mathrm{Prob}(\sigma_i = 1) = 1/2$. The *empirical* Rademacher complexity of $\mathcal{F}$ (e.g. [36]) is defined as

$$\hat{R}_n(\mathcal{F}) = \mathbf{E}_{\boldsymbol{\sigma}} \sup_{f \in \mathcal{F}} \left\{ \frac{1}{n} \sum_{i=1}^n \sigma_i f(\mathbf{x}_i) \right\}.$$

The Rademacher complexity $R_n(\mathcal{F})$ is given by $R_n(\mathcal{F}) = \mathbf{E}\hat{R}_n(\mathcal{F})$, where the expectation is taken with respect to the marginal product distribution P^n over $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$.

Theorem 4.1 ([20]). *Let $\mathcal{F}$ be a class of functions mapping from a domain $\mathcal{X}$ to $\mathbb{R}$, and let $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, $x_i \in \mathcal{X}$, $y_i \in \mathbb{R}$, be independently selected according to a probability measure P. Assume there exists a positive real number M such that for all positive λ*

$$\log \mathbf{E}_{X,Y} \sup_{f \in \mathcal{F}} \cosh(2\lambda\ell(Y, f(X))) \leq \lambda^2 M^2/2. \tag{4.17}$$

Then with probability at least $1 - \delta$ over samples of length n, every $f \in \mathcal{F}$ satisfies

$$L(f) \leq \hat{L}(f) + 2R_n(\mathcal{L}_{\mathcal{F}}) + M\sqrt{\frac{2\log(1/\delta)}{n}}.$$

Note that the condition (4.17) replaces the standard uniform boundedness used in many approaches. In order for this result to be useful we need to present an upper bound on the Rademacher complexity $R_n(\mathcal{L}_{\mathcal{F}})$. Assume initially that $(1/2)\|\mathbf{w}\|^2 \leq A$ (where $\|\mathbf{w}\|^2 = \langle \mathbf{w}, \mathbf{w} \rangle$). We quote the following result from [20] (see Eq. (13)).

Lemma 4.2. *Consider the class of functions* $\mathcal{F}_A = \{f(\mathbf{x}) = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle : \ (1/2)\|\mathbf{w}\|^2 \le A\}$. *Then*

$$\hat{R}_n(\mathcal{F}_A) \le \sqrt{\frac{2A}{n}\left(\frac{1}{n}\sum_{i=1}^n \|\boldsymbol{\phi}(\mathbf{x}_i)\|^2\right)}. \tag{4.18}$$

In the present context $\ell_f(\mathbf{x}, y) = (y - f(\mathbf{x}))^2$. We assume for simplicity that $k(\mathbf{x}, \mathbf{x}') \le B$ for all $\mathbf{x}$ and $\mathbf{x}'$. In the following sequence of inequalities we use the notation $\mathbf{w} \in \Omega_A$ to indicate that $(1/2)\|\mathbf{w}\|^2 \le A$. Denoting the expectation with respect to the product distribution P^n over the sample S by $\mathbf{E}_S$, and keeping in mind that $\mathbf{E}\sigma_i = 0$ for any i we have that

$$\begin{aligned}
R_n(\mathcal{L}_{\mathcal{F}_A}) &= \mathbf{E}_S \mathbf{E}_{\boldsymbol{\sigma}} \sup_{\mathbf{w}\in\Omega_A} \frac{1}{n}\sum_{i=1}^n \sigma_i (y_i - \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_i)\rangle)^2 \\
&= \mathbf{E}_S \mathbf{E}_{\boldsymbol{\sigma}} \sup_{\mathbf{w}\in\Omega_A} \frac{1}{n}\sum_{i=1}^n \sigma_i [\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_i)\rangle^2 - 2\langle \mathbf{w}, y_i\boldsymbol{\phi}(\mathbf{x}_i)\rangle] \\
&\le \mathbf{E}_S \mathbf{E}_{\boldsymbol{\sigma}} \sup_{\mathbf{w}\in\Omega_A} \frac{1}{n}\sum_{i=1}^n \sigma_i (\sqrt{2AB} + 2|y_i|)\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_i)\rangle \\
&\overset{(a)}{\le} \frac{\sqrt{2AB}}{n}\mathbf{E}_S \sqrt{\sum_{i=1}^n (\sqrt{2AB} + 2|y_i|)^2} \\
&\overset{(b)}{\le} \frac{\sqrt{2AB}}{n}\mathbf{E}_S \sqrt{\sum_{i=1}^n (4AB + 8y_i^2)} \\
&\overset{(c)}{\le} \frac{\sqrt{2AB}}{n}\sqrt{4nAB + 8n\mathbf{E}[Y^2]} \\
&\overset{(d)}{\le} \frac{2\sqrt{2}AB + 4\sqrt{AB\mathbf{E}[Y^2]}}{\sqrt{n}} .
\end{aligned}$$

where (a) used (4.18) with $\boldsymbol{\phi}(\mathbf{x}_i)$ replaced by $(\sqrt{2AB}+2|y_i|)\boldsymbol{\phi}(\mathbf{x}_i)$ and the fact that $k(\mathbf{x}, \mathbf{x}) = \|\boldsymbol{\phi}(\mathbf{x}_i)\|^2 \le B$, (b) used $(a+b)^2 \le 2a^2 + 2b^2$, (c) made use of Jensen's inequality, and (d) is based on $\sqrt{x+y} \le \sqrt{x} + \sqrt{y}$.

The derived bound on $R_n(\mathcal{L}_{\mathcal{F}_A})$ is formulated in terms of the parameter A. In order to remove this dependence we use a (by now) standard trick based on the union bound. Let $\{A_i\}_{i=1}^\infty$ and $\{p_i\}_{i=1}^\infty$ be sets of positive numbers such that $\limsup A_i = \infty$ and $\sum_i p_i = 1$ (for example, $p_i = 1/i(i+1)$). We apply Theorem 4.1 for each value of A_i replacing δ by $p_i\delta$. A simple utilization of the union bound, and some algebraic manipulations (described in the proof of Theorem 10 in [20]) allow one to establish the following bound, where all reference to the parameter A has been eliminated.

Theorem 4.3. *Let* $\mathcal{F}$ *be a class of functions of the form* $f_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \phi(\mathbf{x})\rangle$, *and let* $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, $\mathbf{x}_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$, *be independently selected according to a probability measure* P.

Assume further that (4.17) holds. Fix any number g_0 and set $\tilde{g}(\mathbf{w}) = 2\max((1/2)\|\mathbf{w}\|^2, g_0)$ Then with probability at least $1-\delta$ over samples of length n, every $f_w \in \mathcal{F}$ satisfies

$$\mathbf{E}\,(Y - f_{\mathbf{w}}(X))^2 \leq \frac{1}{n}\sum_{i=1}^{n}(y_i - f_{\mathbf{w}}(\mathbf{x}_i))^2 + \frac{4\sqrt{2}B\tilde{g}(\mathbf{w}) + 8\sqrt{B\tilde{g}(\mathbf{w})\mathbf{E}[Y^2]}}{\sqrt{n}}$$
$$+ M\sqrt{\frac{4\log\log_2(2\tilde{g}(\mathbf{w})/g_0) + 2\log(1/\delta)}{n}}\ .$$

The essential feature of the bound in Theorem 4.3 is that it holds uniformly for any $\mathbf{w}$, and thus in particular to the weight vector $\mathbf{w}$ obtained using the KRLS algorithm. Observe that the price paid for eliminating the parameter A is an extra term of order $\log\log\tilde{g}(\mathbf{w})$. Note also that using the dual representation $\mathbf{w} = \sum_i \alpha_i \boldsymbol{\phi}(\mathbf{x}_i)$ the bound can be phrased in terms of the coefficients α_i using $\|\mathbf{w}\|^2 = \boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha}$.

5 Experiments

In this section we experimentally demonstrate the potential utility and efficacy of the KRLS algorithm in a range of machine learning and signal processing applications. We begin by exploring the scaling behavior of KRLS on a simple non-linear static regression problem. We then move on to test KRLS on several well-known benchmark regression problems, both synthetic and real. As a point of reference we use the highly efficient SVM package SVMTorch [4], with which we compare our algorithm. Next, we move to the domain of time series prediction (TSP). The most common approach to the TSP problem is based on identifying a generally non-linear auto-regressive model of the series. This approach essentially reduces the TSP problem to a regression problem with the caveat that samples can no longer be assumed to be IID. Numerous learning architectures and algorithms have been thrown at this problem with mixed results (see e.g. [41]). One of the more successful general purpose algorithms tested on TSP is again the SVM [21]; however SVMs are inherently limited by their off-line (batch) mode of training, and their poor scaling properties. We argue that KRLS is a more appropriate tool in this domain and to support this claim we test KRLS on two well known and difficult time series prediction problems. Finally, we apply KRLS to a non-linear channel equalization problem, on which SVMs were reported to perform well [29]. All tests were run on a 256Mb, 667MHz Pentium 3 Linux workstation.

5.1 Non-Linear Regression

We report the results of experiments comparing the KRLS algorithm (coded in C) to the state-of-the-art SVR implementation SVMTorch [4]. Throughout, the best parameter values (for C, ε, ν)[4] were found by using a 10-fold cross-validation procedure in which we looked for

[4] See Chapter 9 of [27] for the definition of C and ε.

the minimum average root-mean-squared error (RMSE) over a range of parameter values, spaced logarithmically. The kernel we use throughout this section is the Gaussian kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right), \tag{5.19}$$

and a similar procedure was used to find the value of the kernel width parameter σ for which SVMTorch performed best. This value was then used by both SVMTorch and KRLS. All the results reported below are averaged over 50 independent randomly generated training sets.

We first used KRLS for learning the 2-dimensional *Sinc-Linear* function $\sin(x_1)/x_1 + x_2/10$ defined on the domain $[-10, 10]^2$. The kernel width parameter is $\sigma = 4.25$. The SVR parameters are $C = 10^5$ and $\varepsilon = 0.1$, while the KRLS parameter is $\nu = 10^{-3}$. Learning was performed on a random set of samples corrupted by additive IID zero-mean Gaussian noise. Testing was performed on an independent random sample of 1000 noise-free points. Figure 1 depicts the results of two tests. In the first we fixed the noise level (noise STD 0.1) and varied the number of training samples from 5 to 50000, with each training set drawn independently. We then plotted the test-set error (top left), the number of support/dictionary vectors as a percentage of the training set (top-center), and the CPU time (top-right) for each algorithm. As can be seen, the solution produced by KRLS significantly improves upon the SVR solution, both in terms of generalization performance and sparsity (with a maximum of 75 dictionary samples) for training set sizes of 500 samples or more. In terms of speed, KRLS outperforms SVMTorch over the entire range of training set sizes, by one to two orders of magnitude. In fact, looking at the asymptotic slopes of the two CPU-time graphs we observe that, while SVMTorch exhibits a super-linear dependence on the sample size (slope 1.82), KRLS scales linearly (slope 1.00) as required of a real-time algorithm.

In the second test we fixed the training sample size at 1000 and varied the level of noise in the range 10^{-6} to 10. We note that SVMTorch suffers from an incorrect estimation of the noise level by its ε parameter, in all respects. Most notably, in the presence of high noise, the sparsity of its solution deteriorates drastically. In contrast, KRLS produces a sparse solution with complete disregard of the level of noise. Moreover, in terms of generalization, the KRLS solution is at least as good as the SVR solution.

We tested our algorithm on three additional synthetic data-sets, *Friedman 1,2* and *3*, due to [12]. Both training and test sets were 1000 samples long, and introduced noise was zero-mean Gaussian with a standard deviation of 0.1. For these data-sets a simple preprocessing step was performed which consisted of scaling the input variables to the unit hyper-cube, based on their minimum and maximum values. The results are summarized in Table 2.

We also tested KRLS on two real-world data-sets - *Boston housing* and *Comp-activ*, both from Delve[5]. The task in the *Boston* data-set (506 samples, 13 dimensions) is to predict median value of owner-occupied homes in \$1000's for various Boston neighborhoods, based on census data. In the *Comp-activ* (cpuSmall) data-set (8192 samples, 12 dimensions) the

[5] http://www.cs.toronto.edu/~delve/data/datasets.html

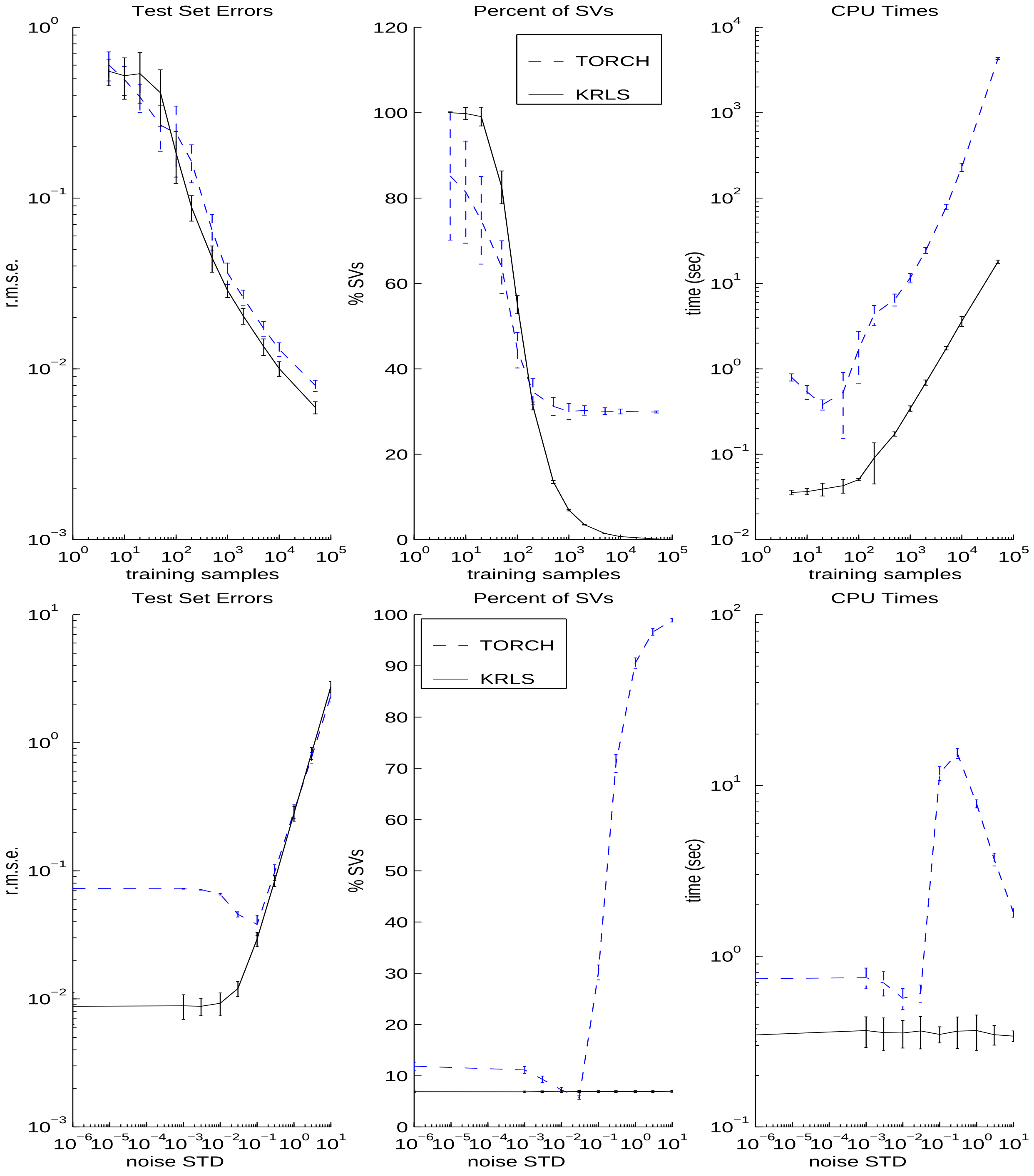

Figure 1: Scaling properties of KRLS and SVMTorch with respect to sample size (top) and noise magnitude (bottom), on the *Sinc-Linear* function. Error bars mark one standard deviation above and below the mean.

Parameters: $t = 1000$, $d = 10$, $C = 10^4$, $\varepsilon = 0.01$, $\sigma = 0.8$, $\nu = 10^{-3}$	**Friedman-1**	RMSE	STD	% SV	CPU
	SVMTorch	0.61	0.03	91.9	24.63
	KRLS	0.55	0.02	100.0	26.97

Parameters: $t = 1000$, $d = 4$, $C = 10^6$, $\varepsilon = 10^{-4}$, $\sigma = 0.5$, $\nu = 10^{-4}$	**Friedman-2**	RMSE	STD	% SV	CPU
	SVMTorch	0.20	0.01	97.3	725.22
	KRLS	0.18	0.02	36.1	10.24

Parameters: $t = 1000$, $d = 4$, $C = 10^5$, $\varepsilon = 0.1$, $\sigma = 0.5$, $\nu = 10^{-3}$	**Friedman-3**	RMSE	STD	% SV	CPU
	SVMTorch	0.09	0.01	37.6	2.32
	KRLS	0.09	0.01	23.3	4.74

Table 2: Results on the synthetic *Friedman* data-sets. The columns, from left to right, list the average R.M.S. test-set error, its standard deviation, average percentage of support/dictionary vectors, and the average CPU time in seconds used by the respective algorithm.

task is to predict the users' CPU utilization percentage in a multi-processor, multi-user computer system, based on measures of system activity. A preprocessing step, like the one used on the *Friedman* data-sets, was performed here as well. Generalization performance was checked using 106 (*Boston*) and 2192 (*Comp-activ*) left-out test samples. The results are summarized in Table 3.

Parameters: $t = 6000$, $d = 12$, $C = 10^6$, $\varepsilon = 0.5$, $\sigma = 0.9$, $\nu = 0.001$	**Comp-activ**	RMSE	STD	% SV	CPU
	SVMTorch	3.13	0.07	81.9	130.4
	KRLS	3.09	0.08	3.4	19.52

Parameters: $t = 400$, $d = 13$, $C = 10^6$, $\varepsilon = 1$, $\sigma = 1.3$, $\nu = 0.001$	**Boston**	RMSE	STD	% SV	CPU
	SVMTorch	3.16	0.49	58.0	5.57
	KRLS	3.52	0.48	38.6	0.53

Table 3: Results on the real-world *Comp-activ* and *Boston* data-sets.

5.2 Time Series Prediction

Digital signal processing is a rich application domain in which the classic RLS algorithm has been applied extensively. On-line and real-time constraints are often imposed upon signal processing algorithms, making many of the recently developed kernel-based machine learning algorithms irrelevant for such tasks. The development of KRLS was aimed at filling this gap, indeed, it is in such tasks that the recursive on-line operation of KRLS becomes particularly useful, if not essential.

An important and well studied problem in both the machine learning and the signal processing communities is the prediction of time series. We tested KRLS on both synthetically generated and real-world time series – The well studied Mackey–Glass time series, and the Laser (A) time series from the Santa Fe time series prediction competition [41]. The Mackey–Glass series is synthetically generated by numerical integration of a time-delay differential equation. The Laser time series is taken from real measurements of the intensity of a far-infrared NH3 laser. Both of these series exhibit chaotic behavior, making the task of multi-step prediction exponentially difficult as a function of the prediction horizon. Nevertheless, short-term multi-step prediction is feasible, depending on the intrinsic predictability of the dynamics. In the Mackey–Glass series it is possible to make accurate predictions 100's of time steps into the future, while for the Laser series the useful limit seems to be about 100. Throughout the time-series experiments we used the Gaussian kernel (5.19).

5.2.1 Mackey–Glass Time Series

Our first experiment is with the Mackey–Glass chaotic time series. This time series may be generated by numerical integration of a time-delay differential equation that was proposed as a model of white blood cell production [18]:

$$\frac{dy}{dt} = \frac{ay(t-\tau)}{1+y(t-\tau)^{10}} - by(t), \tag{5.20}$$

where $a = 0.2$, $b = 0.1$. For $\tau > 16.8$ the dynamics become chaotic; we therefore conducted our tests using two value for τ, corresponding to weakly chaotic behavior at $\tau = 17$ and a more difficult case at $\tau = 30$. Eq. 5.20 was numerically integrated using the Euler method and uniformly distributed initial conditions $\mathbf{x}_0 \in [0.1, 2]$ and $\mathbf{x}_t = 0$ for $t < 0$.

Training sets 1000 samples long were generated by sampling the series at 1 time-unit intervals, with the respective test sets consisting of the subsequent 200 samples. The embedding dimension was fixed at 10 with an embedding delay of 4 samples, i.e. $\mathbf{x}_t = (y(t-4), y(t-8), \ldots, y(t-40))$. The parameters for each algorithm were selected by searching for the minimum 200-step RMS iterated prediction error, averaged over 10 independent training and validation sets. The parameters found for the Mackey–Glass(17) series are, for SVMTorch $\sigma = 0.7$, $C = 10^3$, $\varepsilon = 10^{-5}$, while for KRLS they are $\sigma = 0.5$, $\nu = 10^{-4}$. For the Mackey–Glass(30) series the SVMTorch parameters were unchanged, while for KRLS ν remained the same and $\sigma = 0.6$. The test results for SVMTorch and KRLS on both series, averaged over 50 independent trials, are given in Table 4.

On the Mackey–Glass(30) series KRLS and SVM perform comparably, both in terms of prediction accuracy and in terms of sparsity. However, on the Mackey–Glass(17) series KRLS significantly outperforms SVM in its prediction accuracy; why this should be the case remains to be clarified. Fig. (2) shows examples of two test sets, one for each series, and the iterated predictions produced by the two algorithms.

MG(17)	RMSE(1)	STD(1)	RMSE(200)	STD(200)	% SV	CPU
SVMTorch	0.0023	0.0003	0.0187	0.0080	30.48	4.18
KRLS	0.0004	0.0002	0.0027	0.0016	27.05	6.85
MG(30)	RMSE(1)	STD(1)	RMSE(200)	STD(200)	% SV	CPU
SVMTorch	0.006	0.003	0.034	0.036	50.2	7.23
KRLS	0.006	0.004	0.033	0.028	51.9	12.00

Table 4: 1-step and 200-step iterated prediction results on the Mackey–Glass time series with $\tau = 17$ (MG(17)) and $\tau = 30$ (MG(30))

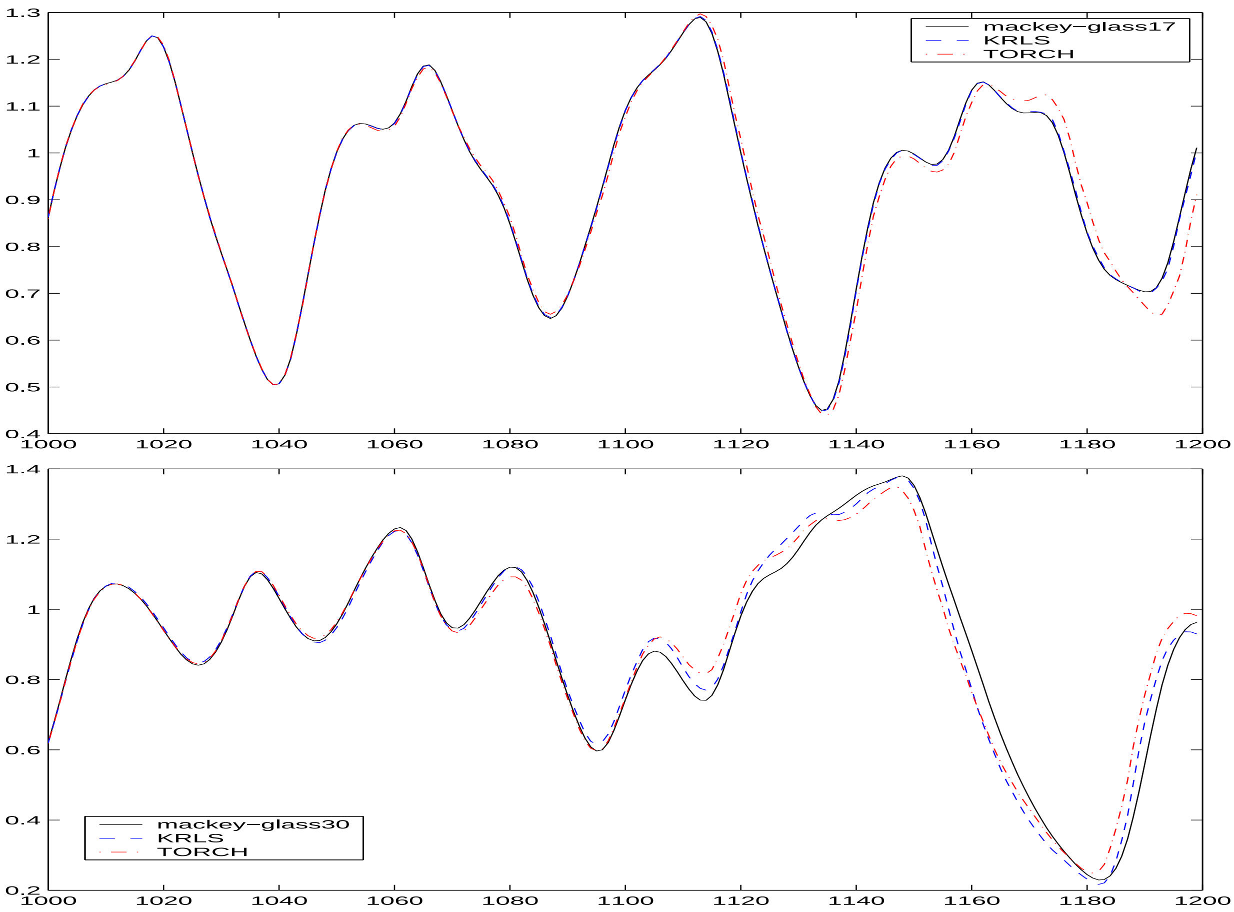

Figure 2: Multi-step iterated predictions for the Mackey–Glass time series with $\tau = 17$ (top) and $\tau = 30$ (bottom).

5.2.2 Santa Fe Laser Time Series

Our next experiment is with the chaotic laser time series (data set A) from the Santa Fe time series competition [41] [6] This is a particularly difficult time series to predict, due both

[6] http://www-psych.stanford.edu/ãndreas/Time-Series/SantaFe.html

to its chaotic dynamics and to the fact that only three “intensity collapse” events occur in the training set. The accurate prediction of these events is crucial to achieving a low prediction error on the test set. The training data consists of 1000 samples, with the test data being the subsequent 100 samples (see Fig. 3). The task is to predict the test series with minimum mean squared error. Looking at the competition results the difficulty of this task becomes apparent, as only two of the 14 contestants achieved prediction accuracies that are significantly better than simply predicting the mean of the training series. The winning entry achieved a normalized mean squared error (NMSE - the mean squared error divided by the series’ variance) of 0.028, by utilizing a complex and highly specialized neural network architecture adapted by a temporal version of the backpropagation algorithm. Moreover, the final 25 steps of the predicted sequence were hand-picked by adjoining to the initial 75–step prediction a similar sequence taken from the training set. The second place entry used an approach based on local linear models and achieved a NMSE of 0.080 on the test set.

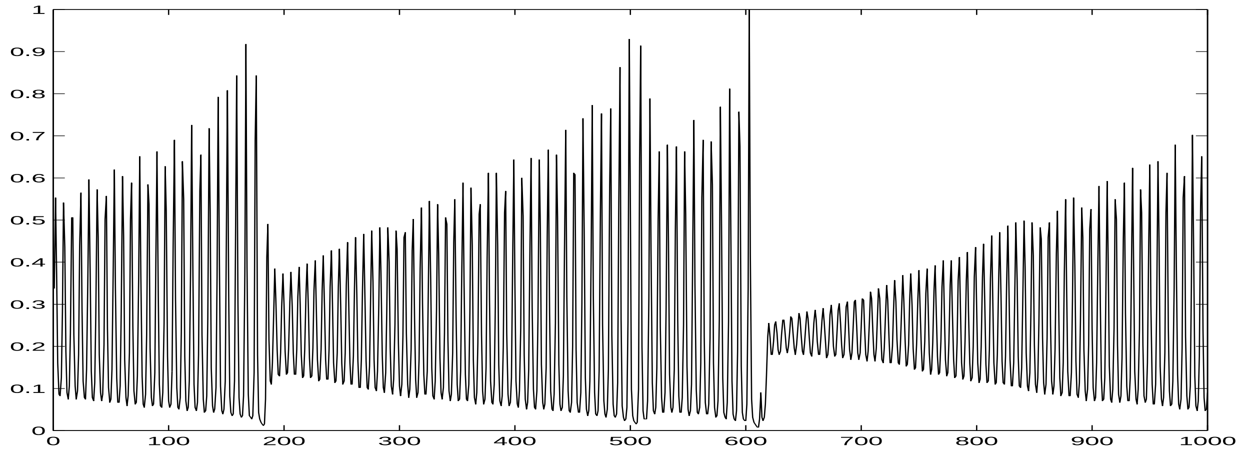

Figure 3: The Santa Fe competition laser training series (data set A)

We attempted to learn this series with the KRLS algorithm. The naive approach, as used above for the Mackey–Glass series, would be to minimize the 1-step prediction error by defining the training samples as $\{(\mathbf{x}_i, y_i)\}_{i=1}^t$ with $\mathbf{x}_i = (y_{i-1}, y_{i-2}, \ldots, y_{i-d})$, where d is the model order and t the number of training samples [7]. Predicting the test series is performed by *iterated prediction* in which successive 1-step predictions are fed back into the predictor as inputs for the prediction of subsequent series values. Unfortunately this naive approach works well for this series only if we are really interested in 1-step prediction. Since our goal is to provide multi-step predictions, a more sophisticated method is called for. The method we used is as follows: Run KRLS to minimize the 1-step prediction MSE, as in the naive approach. After this pass over the training set is complete, we compute the optimal 1-step estimates $(\hat{y}_1^1, \ldots, \hat{y}_t^1)$. Next, we continue running KRLS on a new data set, which is of the same length as the original data, and in which $\mathbf{x}_t = (\hat{y}_{t-1}^1, y_{t-2}, \ldots, y_{t-d})$. We now

[7] $y_0, \ldots, y_{1-d}$ are assumed to be equal to zero.

have 2-step estimates $(\hat{y}_1^2, \ldots, \hat{y}_t^2)$, based on the real data and the 1-step estimates obtained in the previous step. On the i'th step of this iterative process the appended training set consists of samples of the form $\mathbf{x}_t = (\hat{y}_{t-1}^{i-1}, \hat{y}_{t-2}^{i-2}, \hat{y}_{t-3}^{i-3}, \ldots, \hat{y}_{t-i+1}^{1}, y_{t-i}, \ldots, y_{t-d})$ if $i \leq d$; or $\mathbf{x}_t = (\hat{y}_{t-1}^{i-1}, \hat{y}_{t-2}^{i-2}, \hat{y}_{t-3}^{i-3}, \ldots, \hat{y}_{t-d}^{i-d})$ if $i > d$.

We may iterate this process until some prespecified prediction horizon n is reached. Assuming that the dictionary ceases to grow at an early stage, The end result is a good approximation to the minimizer of the mean squared error over the entire $n \times t$ - long dataset, in which equal weight is given to 1-step through n-step prediction accuracy. The idea behind this somewhat complex scheme is to improve the stability of the iterative multi-step predictor with respect to small errors in its predicted values, since these are fed back into the predictor as inputs for predictions at subsequent time steps. Note that this scheme relies heavily on the recursiveness of KRLS; the implementation of such a scheme using a batch algorithm such as SVM would be considerably more difficult and costly.

The free parameters of the algorithm were tuned as follows. First we normalized the series, so that its values lie in the range $[0, 1]$. We then performed hold-out tests to determine the values of σ, ν, d and n, where ν is the ALD threshold parameter, d is the model order and n is the prediction horizon for the iterations described above. The training sets we used were a. samples 1-700, b. samples 1-800, and c. samples 1-900. Their respective held-out sets were a. 701-800, b. 801-900 and c. 901-1000. The parameters were optimized with respect to the mean squared error of the multi-step iterated prediction on the held out sets. When insignificant differences in prediction accuracy were observed, the parameter value incurring a lower computational cost was preferred (i.e. preference to high values of ν and σ and low values of d and n). The values found are: $\sigma = 0.9$, $\nu = 0.01$, $d = 40$ and $n = 6$. The NMSE prediction error on the competition test set (samples 1001-1100) achieved by KRLS is 0.026, which is slightly better than the winning entry in the competition. The KRLS prediction and the true continuation are shown in Fig. 4. [8]

5.3 Channel Equalization

In [29] SVMs were applied to non-linear channel equalization problems with considerable success. One of the reservations made by the authors concerning the use of SVMs in this application domain was due to the inability of SVMs to be trained on-line. We suggest KRLS as a viable alternative that performs similarly in terms of error rates, but may be trained on-line, and often produces solutions that are much sparser than SVM solutions in less time, especially for large amounts of data.

Let us briefly state the channel equalization problem. A binary signal is fed into a generally non-linear channel. At the receiver end of the channel the signal is further corrupted by additive IID (usually Gaussian) noise, and is then observed as $(y_1, \ldots, y_t)$. The aim of channel equalization is to construct an "inverse" filter that reproduces (possibly with some delay) the original signal with as low an error rate as possible. In order to attain

[8] In this experiment we used a Matlab version of KRLS, for which the entire training session took less than 5 minutes. The C implementation typically runs 5–10 times faster.

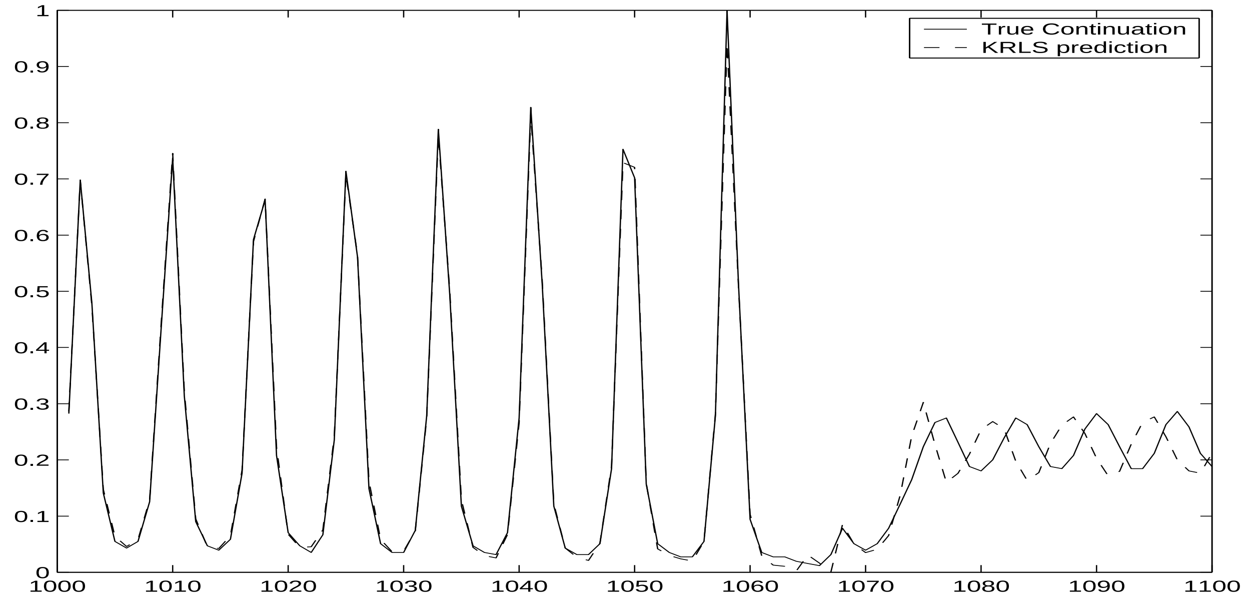

Figure 4: KRLS predicting 100 steps into the future (dashed line) on the laser time series. The true continuation is shown as a solid line. Note that on the first 60 steps the prediction error is hardly noticeable.

this goal a known random binary signal $(u_1, \dots, u_t)$ is sent through the channel and the corresponding noisy observations $(y_1, \dots, y_t)$ are used to adapt the equalizer. Just as the time-series prediction problem can be cast as a regression problem, it is easy to see that channel equalization may be reduced to a classification problem, with samples $\{(\mathbf{x}_i, u_i)\}_{i=1}^t$, $\mathbf{x}_i = (y_{i+D}, y_{i+D-1}, \dots, y_{i+D-d+1})$, where d is the model order and D is the equalization time lag. As KRLS minimizes the MSE, its choice as a learning algorithm for equalization seems questionable, since equalization aims at reducing the bit error rate (BER), which is the number of misclassifications. Nevertheless we show that the performance of KRLS solutions, as measured by the BER criterion, is at least as good as SVM classification solutions.

In this experiment we replicate the setup used in the first simulation in [29]. The nonlinear channel model is defined by $x_t = u_t + 0.5u_{t-1}$, $y_t = x_t - 0.9x_t^3 + n_\sigma$, where n_σ is white Gaussian noise with variance 0.2. As in [29] a SVM with a third degree polynomial kernel $k(\mathbf{x}, \mathbf{x}') = \left(\langle \mathbf{x}, \mathbf{x}' \rangle + 1\right)^3$ was used, the other parameters being $t = 500$ and $C/t = 5$. Testing was performed on a 5000 samples long random test sequence. The results are presented in Table 5; each entry is the result of averaging over 50 repeated independent tests.

The results for the SVM as reported in [29] are significantly better, especially for $D = 0$; we currently have no explanation to this discrepancy. The most remarkable result of this experiment are the differences in the sparsities of the solutions provided by the respective algorithms. KRLS uses 2% of the dataset for its dictionary throughout while the SVM uses anywhere between 10% to 68% of the data as support vectors. Still, KRLS outperforms the SVM.

Table 5: Results of the non-linear channel equalization experiment

	SVM	KRLS
D=0 BER Sparsity (%)	0.302 ± 0.013 63.8 ± 3.5	0.279 ± 0.017 2.0± 0.0
D=1 BER Sparsity (%)	0.087 ± 0.020 19.6 ± 2.4	0.070 ± 0.004 2.0 ± 0.0
D=2 BER Sparsity (%)	0.047 ± 0.006 12.2 ± 2.2	0.043 ± 0.004 2.0 ± 0.0

6 Discussion and Conclusion

We have presented a non-linear Kernel based version of the popular RLS algorithm. The algorithm requires two key ingredients: expressing all RLS related operations in terms of inner products in the feature space (which can be calculated using the kernel function in the input space), and sparsifying the data so that the dimension of the samples in the feature space remains bounded.

As mentioned in Section 2.4, the approach closest to ours is probably that of Csató and Opper [7], which introduced an on-line algorithm for sparse Gaussian Process (GP) regression. Since in GP regression the posterior moments are evaluated, the method described in [7] requires an additional parameter for the estimated measurement noise variance that is not needed in our method. The distinction between sparse GP regression and our sparsification method becomes apparent if this parameter is given a lower value than the true noise variance. In this case, given some fixed dictionary, sparse GP regression would favor fitting the dictionary points at the price of increased error on the other points. In the limit, GP regression would fit only the dictionary points and ignore completely all other points. This means that to compensate for an erroneous noise model, sparse GP regression would need to increase the size of its dictionary, sacrificing sparsity much like the observed behavior of SVR. In comparison, our sparsification method weights all points equally and maintains the same level of sparsity irrespective of the level of noise. Further, the sparsification algorithm used in [7] also differs in that it considers the error performed in estimating each new sample's *target value* (y_t) if that sample is not added to the dictionary, while our method considers the error incurred in approximating the sample point $\boldsymbol{\phi}(\mathbf{x}_t)$ itself. This is essentially the distinction between *supervised* and *unsupervised* sparsification. By taking the unsupervised path we were able to prove the finiteness of the dictionary and derive the PCA-like properties of our framework.

Let us summarize the main contributions of this work. We presented a computationally efficient on-line algorithm possessing performance guarantees similar to those of the RLS algorithm (e.g. [25]). Essentially, this means that the information content of each sample is fully extracted before that sample is disposed, since a second iteration over a previously learned training set would cause no change, while on-line gradient-based algorithms usually benefit from data recycling. Due to the unsupervised nature of our sparsification mechanism the sparsity of the solution is immune both to increase in noise and in training set size. Moreover, we formally proved the relationship of our on-line sparsification approach to kernel PCA with its known optimality properties. We have also been able to present data-dependent generalization bounds that use only the dictionary obtained. Finally, experiments indicate that the algorithm compares very favorably with the state-of-the-art SVR algorithm SVMTorch in both generalization performance and computation time. In some cases the KRLS algorithm produces much sparser solutions with higher robustness to noise. The usefulness of KRLS was also demonstrated in the RLS algorithm's traditional application domain – signal processing. Here too, our algorithm compares favorably with current state-of-the-art algorithms and results

An important future research direction would be to employ our on-line sparsification method, in conjunction with the kernel trick, to "kernelize" other algorithms that are recursive in nature, such as the Kalman filter (e.g. [14]). A non-linear kernelized version of the Kalman filter may be able to circumvent the inherent problem of handling non-linearities, which was only partially resolved by the extended Kalman filter.

As far as the KRLS algorithm is concerned, many directions for modification and improvement are open. Further exploration into the connection of KRLS with maximum likelihood estimation is in order. Other directions include the utilization of specialized kernels tailored to specific problems, such as time series prediction; optimization of the kernel function by tuning its hyper-parameters; noise estimation (in the spirit of adaptive RLS); an exponentially weighted version of KRLS; and adaptive model order identification, as in [24].

References

[1] M. Anthony and P.L. Bartlett. *Neural Network Learning; Theoretical Foundations.* Cambridge University Press, 1999.

[2] C.J.C. Burges. Simplified support vector decision rules. In *International Conference on Machine Learning*, pages 71–77, 1996.

[3] C.J.C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector machines. In *Advances in Neural Information Processing Systems*, volume 9. MIT Press, 1997.

[4] R. Collobert and S. Bengio. SVMTorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1:143–160, 2001.

[5] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines.* Cambridge University Press, Cambridge, England, 2000.

[6] L. Csató and M. Opper. Sparse representation for Gaussian process models. In *Advances in Neural Information Processing Systems 13*, 2001.

[7] L. Csató and M. Opper. Sparse on-line Gaussian processes. *Neural Computation*, 14:641–668, 2002.

[8] T. Downs, K. Gates, and A. Masters. Exact simplification of support vector solutions. *Journal of Machine Learning Research*, 2:293–297, December 2001.

[9] Y. Engel, S. Mannor, and R. Meir. Sparse online greedy support vector regression. In *13th European Conference on Machine Learning*, 2002.

[10] T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13(1):1–50, 2000.

[11] S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representation. *JMLR (Special Issue on Kernel Methods)*, pages 243–264, 2001.

[12] J.H. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19(1):1–141, 1991.

[13] M. Gibbs and D. MacKay. Efficient implementation of Gaussian processes, 1997. Draft.

[14] S. Haykin. *Adaptive Filter Theory.* Prentice Hall, 3rd edition, 1996.

[15] R. Herbrich. *Learning Kernel Classifiers.* MIT Press, Cambridge, MA, 2002.

[16] T. Kailath, A.H. Sayed, and B. Hassibi. *Linear Estimation.* Prentice Hall, 2000.

[17] L. Ljung. *System Identification: Theory for the User.* Prentice Hall, New Jersey, second edition, 1999.

[18] M. Mackey and L. Glass. Oscillation and chaos in physiological control systems. *Science*, 197:287–289, 1977.

[19] S. Mallat. *A Wavelet Tour of Signal Processing.* Academic Press, New York, 1998.

[20] R. Meir and T. Zhang. Generalization bounds for Bayesian mixture algorithms. Technical Report CCIT # 421, Technion, 2003. Submitted for publication.

[21] K.-R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Using support vector machines for time series prediction. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Proceedings ICANN'97, International Conference on Artificial Neural Networks*, volume 1327 of *LNCS*, pages 999–1004, Berlin, 1997. Springer.

[22] B.K. Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 24(2):227–234, 1995.

[23] E. Osuna and F. Girosi. Reducing the run-time complexity of support vector machines. In *International Conference on Pattern Recognition*, 1998.

[24] A.H. Sayed and T. Kailath. A state-space approach to adaptive RLS filtering. *IEEE Signal Processing Magazine*, 11(3):18–60, 1994.

[25] L.L. Scharf. *Statistical Signal Processing.* Addison-Wesley, 1991.

[26] B. Schlkopf, S. Mika, C.J.C. Burges, P. Knirsch, K.-R. Mller, G. Rtsch, and A.J. Smola. Input space vs. feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, 10(5):1000–1017, 1999.

[27] B. Schölkopf and A. Smola. *Learning with Kernels.* MIT Press, Cambridge, MA, 2002.

[28] B. Schölkopf, A. Smola, and K.-R. Muller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

[29] D.J. Sebald and J.A. Bucklew. Support vector machine techniques for nonlinear equalization. *IEEE Transactions on Signal Processing*, 48(11):3217–3226, 2000.

[30] A.J. Smola and P.L. Bartlett. Sparse greedy Gaussian process regression. In *Advances in Neural Information Processing Systems 13*, pages 619–625. MIT Press, 2001.

[31] A.J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proc. 17th International Conference on Machine Learning*, pages 911–918. Morgan Kaufmann, San Francisco, CA, 2000.

[32] P. Sollich. Bayesian methods for support vector machines: Evidence and predictive class probabilities. *Machine Learning*, 46(1-3):21–52, 2002.

[33] N. Syed, H. Liu, and K. Sung. Incremental learning with support vector machines. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999.

[34] M.E. Tipping. Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, June 2001.

[35] M.E. Tipping. Sparse kernel principal component analysis. In *Advances in Neural Information Processing Systems 135*, pages 633–639. MIT Press, 2001.

[36] A.W. van der Vaart and J.A. Wellner. *Weak Convergence and Empirical Processes.* Springer Verlag, New York, 1996.

[37] V. Vapnik, S. Golowich, and A. Smola. Support vector method for function approximation, regression estimation, and signal processing. In *Advnaces in Neural Information Processing Systems*, pages 281–287, 1997.

[38] V.N. Vapnik. *The Nature of Statistical Learning Theory.* Springer Verlag, New York, 1995.

[39] P. Vincent and Y. Bengio. Kernel matching pursuit. *Machine Learning*, 48:165–187, 2002.

[40] G. Wahba. *Spline Models for Observational Data.* SIAM, 1990.

[41] A.S. Weigend and N.A. Gershenfeld eds. *Time Series Prediction.* Addison Wesley, 1994.

[42] C. Williams. Prediction with Gaussian processes: From linear regression to linear prediction and beyond. In M. I. Jordan, editor, *Learning and Inference in Graphical Models.* Kluwer, 1998.

[43] C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*, pages 682–688, 2001.

[44] T. Zhang. Sequential greedy approximation for certain convex optimization problems. *IEEE Transactions on Information Theory*, 49(3):682–691, 2003.