Kernel Methods and their Potential Use in Signal Processing

Fernando Pérez-Cruz[†]*and Olivier Bousquet[‡]

[†]Dept. of Signal Theory and Communications, Universidad Carlos III de Madrid. Avda. de la Universidad 30, 28911 Leganés, Madrid, Spain, and;
Gatsby Computational Neuroscience Unit. University College London. Alexandra House, 17 Queen Square, London, WC1N 3AR UK. fernandop@ieee.org

[‡]Max Planck Institute for Biological Cybernetics, Spemannstr. 38, D-72076 Tübingen, Germany olivier.bousquet@tuebingen.mpg.de

Abstract

The notion of kernels, recently introduced, has drawn a lot of interest as it allows to obtain nonlinear algorithms from linear ones in a simple and elegant manner. This, in conjunction with the introduction of new linear classification methods such as the Support Vector Machines has produced significant progress in machine learning and related research topics. The success of such algorithms is now spreading as they are applied to more and more domains. Signal processing procedures can benefit from a kernel perspective, making them more powerful and applicable to nonlinear processing in a simpler and nicer way. We present an overview of kernel methods and provide some guidelines for future development in kernel methods, as well as, some perspectives to the actual signal processing problems in which kernel methods are being applied.

1 Introduction

Kernel-based algorithms have been recently developed in the machine learning community, they were first used to solve binary classification problems, the so-called Support Vector Machine [1]. And there is now an extensive literature on SVM [2, 3] and the family of kernel-based algorithms [4]. In a nutshell, a kernel-based algorithm is a nonlinear version of a linear algorithm where the data has been previously (nonlinearly) transformed to a higher dimensional space in which we only need to be able to compute inner products (via a kernel function). The attractiveness of such algorithms stems from their elegant treatment of nonlinear problems and their efficiency in high-dimensional problems.

*This work has been partially supported by CAM 07T/0016/2003.

It is clear that many problems arising in signal processing are of statistical nature and require automatic data analysis methods. Furthermore, the algorithms used in signal processing are usually linear and their transformation for nonlinear processing is sometimes unclear. Signal processing practitioners can benefit from a deeper understanding of kernel methods, because they provide a different way of taking into account nonlinearities without loosing the original properties of the linear method. Another aspect is dealing with the amount of available data in a space of a given dimensionality, one needs methods that can use little data and avoid the curse of dimensionality. This is what Vapnik's approach aims at [2] and explains why using kernel methods and Vapnik's ideas may allow to efficiently handle data from nonlinear signal processing problems.

We will start in Section 2 with an intuitive explanation on how the data, after being put through a nonlinear transformation, can be more easily linearly explained, and we will draw a simple example on the transformation of a linear algorithm into a nonlinear one using kernels. We will review in Section 3 and 4 two of the most widely use kernel algorithms, Support Vector Machines for binary classification and Kernel Principal Component Analysis (K-PCA), to illustrate how they can be used for nonlinear knowledge discovery. We will then have sufficient information about kernel method development to be able to extend it to any signal processing algorithm that can be expressed using inner products.

Kernel methods need to operate with the so-called kernel matrix, which is formed by all the inner products of the input samples in the transformed space (also known as feature space). This matrix is regularly constructed using a known kernel function, but in many applications these typical kernels do not provide significant results. There is a new trend in machine learning in which the kernel matrix is not computed but it is instead learnt from the samples, so it directly captures the nonlinear relations within the input data. We will present this approach in Section 5, together with its lines for future development.

We will continue in Section 6 with an outline of some applications in which kernel methods and support vector machines are being applied with success and describe some of them. We will conclude the paper with a discussion in which some open questions still remain to be answered and give final thoughts about future work for kernel methods.

2 Kernel Methods

Many algorithms for data analysis are based on the assumption that the data can be represented as vectors in a finite dimensional vector space. These algorithms, such as linear discrimination, principal component analysis, or least squares regression, make extensive use of the linear struc-

ture. Roughly speaking, kernels allow to naturally derive nonlinear versions of them.

The general idea is the following. Given a linear algorithm (i.e. an algorithm which works in a vector space), one first maps the data living in a space $\mathcal{X}$ (the input space) to a vector space $\mathcal{H}$ (the *feature space*) via a nonlinear mapping $\boldsymbol{\phi}(\cdot) : \mathcal{X} \to \mathcal{H}$, and then runs the algorithm on the vector representation $\boldsymbol{\phi}(\mathbf{x})$ of the data. In other words, one performs nonlinear analysis of the data using a linear method.

The purpose of the map $\boldsymbol{\phi}(\cdot)$ is to translate nonlinear structures of the data into linear ones in $\mathcal{H}$. Consider the following discrimination problem (see Figure 1) where the goal is to separate two sets of points. In the input space, the problem is nonlinear, but after applying the transformation $\boldsymbol{\phi}(\cdot)$ which maps each vector to the three monomials of degree 2 formed by its coordinates, the separation boundary becomes linear. We have just transformed the data and

(a) Input space $\mathcal{X}$ (b) Feature space $\mathcal{H}$

Figure 1: Effect of the map $\boldsymbol{\phi}(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$

we hope that in the new representation, linear structures will emerge. However, we may need to add a lot of dimensions to really make this happen and it may be hard to 'guess' the right $\boldsymbol{\phi}(\cdot)$.

Here is where the so-called kernel comes into the game. We shall first restrict ourselves to algorithms that work in vector spaces endowed with an inner product. In this case, $\boldsymbol{\phi}(\cdot)$ has to map the input space to a Hilbert space.

If, in the execution of the algorithms, only inner products between data vectors are considered, i.e. the data appears only in expressions like $\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}') \rangle = \boldsymbol{\phi}^T(\mathbf{x})\boldsymbol{\phi}(\mathbf{x}')$, we can make use of the fact that for certain specific maps $\boldsymbol{\phi}(\cdot)$ this inner product can be computed directly from $\mathbf{x}$ and $\mathbf{x}'$ without explicitly computing $\boldsymbol{\phi}(\mathbf{x})$ and $\boldsymbol{\phi}(\mathbf{x}')$. This computational trick is known as the 'kernel trick'. More precisely, a kernel is a symmetric function of two variables that satisfies the following condition: for all $n \in \mathbb{N}$, and all $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$, the kernel matrix, i.e. the matrix whose elements are $k(\mathbf{x}_i, \mathbf{x}_j)$ is positive semi-definite. The main property of functions satisfying this condition is that they implicitly define a mapping $\boldsymbol{\phi}(\cdot)$ from $\mathcal{X}$ to a Hilbert space $\mathcal{H}$ such that $k(\mathbf{x}, \mathbf{x}') = \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}') \rangle$ and can thus be used in algorithms using inner products,

introducing nonlinearities via a straightforward modification.

Because they correspond to inner products in some space, kernels can be considered as measures of similarity between two data points. Many different types of kernels are known [4] and among them, the polynomial kernel of degree d, defined as $k(\mathbf{x}, \mathbf{x}') = (a + \langle \mathbf{x}, \mathbf{x}' \rangle)^d$ and the so-called Gaussian kernel $k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2 / 2\sigma^2)$, are quite useful in practice. However, kernels are not limited to vector spaces and can be defined on graphs, sequences, groups, among others (see [4]). This is a key feature of kernels: they allow to work in a simple (linear) way on data of various types (discrete or not, fixed or variable length).

Another issue is knowing why transforming the data nonlinearly to a higher dimensional space ensures that a linear algorithm can be used over it to obtain a linear explanation of the data. For classification, an intuitive explanation can be provided by Cover's theorem [5] which states the probability that n data points can be linearly separated in a d dimensional space is:

$$P(n, d) = \begin{cases} 1, & n \leq d + 1 \\ \dfrac{1}{2^{n-1}} \displaystyle\sum_{i=0}^{d} \begin{pmatrix} n-1 \\ i \end{pmatrix}, & n \geq d + 1 \end{cases}$$

The direct consequence of this theorem is that the probability of linear separability of n data points for any distribution and class label increases with growing d, so we hold to this belief to expect that after transforming the data we will have more chances to have a linear explanation.

Another explanation can be provided based on the expression power of kernels like the Gaussian kernel. It has been proved [6] for example that this kernel is *universal* which means that linear combinations of this kernel computed at the data points can approximate any (continuous) function. The corresponding feature space is actually infinite dimensional and intuitively, this means that given any labelled data set (where points with different labels have different positions), there exists a linear hyperplane which correctly separates them in the Gaussian feature space. This hyperplane may correspond to a very complex non-linear decision surface in the input space. So the expression power is basically unlimited, but at the same time, as we will see in section 3, the complexity of the solution can be controlled to avoid overfitting of the data.

2.1 An Example: Time Series Prediction

Now, we will illustrate how a linear algorithm can be described using kernels, obtaining a linear algorithm in the feature space and a nonlinear algorithm in the input or original space. We have selected a time series prediction problem in which the actual output is predicted using the previous samples of the process, i.e $\hat{y}[k] = w_1 y[k-1] + w_2 y[k-2] + \cdots + w_d y[k-d] + b$. Our goal

is to find the coefficients w_i and b that minimise the error (mean square error) between the true output of the process $y[k]$ and its estimate $\hat{y}[k]$ for a given number of samples from the time series. We can describe the estimation process, given a labelled dataset $((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$, where $\mathbf{x}_i = [y[d+i], \ldots, y[i]]^T \in \mathcal{X}$ and $y_i = y[d+1+i] \in \mathbb{R}$, as the minimization of $\sum_{i=1}^{n}(y_i - (x_i^T\mathbf{w}+b))^2$ with respect to $\mathbf{w} = [w_1, \ldots, w_d]^T$ and b. To transform this least square estimation into a kernel method algorithm, we only need to apply a nonlinear mapping $\boldsymbol{\phi}(\cdot)$ over $\mathbf{x}_i$. In this case, we will need to solve the following linear system:

$$\begin{array}{c} \boldsymbol{\Phi}^T(\boldsymbol{\Phi}\mathbf{w} + \mathbf{1}b) = \boldsymbol{\Phi}^T\mathbf{y} \\ \mathbf{1}\boldsymbol{\Phi}^T\mathbf{w} + b = \mathbf{1}^T\mathbf{y} \end{array} \quad \Longrightarrow \quad \begin{bmatrix} \boldsymbol{\Phi}^T\boldsymbol{\Phi} & \mathbf{1} \\ \mathbf{1}^T\boldsymbol{\Phi} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Phi}^T\mathbf{y} \\ \mathbf{1}^T\mathbf{y} \end{bmatrix}$$

which was obtained equating to zero the gradient of the mean square error with respect to $\mathbf{w}$ and b, where $\boldsymbol{\Phi}^T = [\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_n)]$, $\mathbf{y} = [\mathbf{y}_1, \ldots \mathbf{y}_n]^T$ and $\mathbf{1}$ is a column vector of n ones. For solving this linear system of equations we need to know the nonlinear mapping $\boldsymbol{\phi}(\cdot)$. To work with kernels, we can rely on the Representer theorem [7, 4] that, under fairly general conditions, states that the best solution can be constructed as a linear combination of the training samples in the feature space, i.e. $\mathbf{w} = \sum_{i=1}^{n} \boldsymbol{\phi}(\mathbf{x}_i)\gamma_i = \boldsymbol{\Phi}^T\boldsymbol{\gamma}$. If we replace this definition into the above system, it can be reduced to:

$$\begin{bmatrix} \mathbf{H} & \mathbf{1} \\ \mathbf{1}^T\mathbf{H} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{\gamma} \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ \mathbf{1}^T\mathbf{y} \end{bmatrix}$$

where $(\mathbf{H})_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ is a matrix formed by inner products in the feature space, the so-called kernel matrix. The predictions of the time series can be done, as well, using kernels: $\hat{y}[k] = \boldsymbol{\phi}^T(\mathbf{x})\mathbf{w} + b$, where $\mathbf{x} = [y[k-1], \ldots, y[k-d]]^T$. If we replace the definition of $\mathbf{w}$ as a linear combination of the training samples, each prediction can be obtained as follows $\hat{y}[k] = \sum_{i=1}^{n} k(\mathbf{x}_i, \mathbf{x})\gamma_i + b$. Therefore, we can find the parameters of the model and make predictions using kernels, without ever needing to explicitly compute the nonlinear mapping $\boldsymbol{\phi}$.

This simple, yet illustrative, example shows the two basic ingredients to transform a linear algorithm into a nonlinear one using kernels. This algorithm for regression estimation is known as Gaussian Processes for regression [8], in which the kernel matrix plays the role of a Gaussian covariance matrix. But before going any further, let us make two comments about the procedure to construct kernel algorithms. First, the Representer theorem tells us how the best solution can be constructed in a given feature space, but it is not telling that the chosen space is optimal in any sense. The search for such space is problem dependant and the practitioner would have to find it for each given application. In some cases, it can be as simple as tuning a hyper-parameter, i.e. σ for the RBF kernel, or can be as complicated as learning the kernel matrix,

as we will describe in Section 4, or constructing the whole nonlinear mapping and deriving the kernel afterwards.

Second, transforming a linear algorithm into a nonlinear one, does not guarantee immediate success. Linear algorithms work for linear problems, because they usually have sufficient data for the given input space. But with a nonlinear version using kernels, in which the feature space can be larger than the number of training samples (or even infinite dimensional), one cannot rely only on the data to obtain good estimates. At this point is when VC (Vapnik-Chervonenkis) Theory [2] and Regularization [9] play a fundamental role, to be able to provide meaningful results even in an infinite dimensional feature space. We will show a practical example in the following section, the support vector machine, in which these theories can be incorporated into nonlinear algorithms to guarantee good performance.

3 Kernel Based Algorithms

We will present two kinds of learning algorithms. The first ones, supervised learning algorithms, take as inputs a set of *labeled examples*, $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ where $\mathbf{x}_i$ are in some *input space* $\mathcal{X}$ and y_i are typically in $\mathbb{R}$, and they produce as an output, a function $f : \mathcal{X} \to \mathbb{R}$ which (hopefully) is able to predict the label y of new examples $\mathbf{x}$. The second ones, work on unlabeled examples ($\mathbf{x}_i$ only) and try to describe the structure of the data (e.g. its distribution).

3.1 Support Vector Machines

The support vector machine (SVM) is a nonlinear algorithm to perform binary classification ($y_i \in \{\pm 1\}$) proposed in [1] in 1992. The SVM is a nonlinear version of a linear algorithm, the optimal hyperplane decision rule (also known as the generalized portrait algorithm), which was first described in the early sixties [10, 11]. The idea is, given a labelled data set linearly separable in the feature space, find the "optimal" linear decision function. In terms of training error, we have many (some would prefer the word infinite) solutions that are optimal, i.e. that are able to discriminate the samples without error. Therefore, we will need to find another criteria to obtain the optimal solution among those with null training error. As our final goal is to construct a decision function which is able to generalize for future unseen patterns, we would have to find a criteria to match this purpose.

If we knew the density of each class and their relative frequencies, we could construct the optimal decision rule using maximum a posteriori (MAP) criteria. But estimating the density from samples is a very hard learning problem [2] especially in high dimension. Thus, in most practical pattern recognition problems, is not an affordable task. But, we can translate the

idea behind MAP to obtain a criteria for selecting the "optimal" hyperplane without knowing the densities and relative frequencies of the classes. We have a linearly separable set, therefore there must be a gap between the samples of each class, where the optimal hyperplane must lie. Not knowing their densities nor their relative frequencies, it would be advisable to set the linear decision function in the middle of the gap between the classes, correctly classifying all the training samples and putting the decision functions as far apart from the given samples as possible. Therefore, having situated the linear decision function as far apart from the samples as possible, we will have less chance of making an incorrect prediction over the unseen patterns (that are supposed to be generated by the same distribution which generated the training data). The distance from the decision boundary to the closest training example is called the margin and the SVM algorithm, which finds an hyperplane which maximizes this distance is called a maximum margin classifier.

Maximizing the margin seems reasonable if the only available information is the given data points and, the more dimensions the data has, the more suitable this principle is, because the less information about the exact location of the decision boundary is carried by the data. This is an intuitive explanation of why it has shown very good results in high dimensional problems (handwritten digit recognition [12] and face detection in images [13]). However, optimality in the sense of maximum margin and minimum training error, does not imply optimality in the sense of minimum test (i.e. expected) error. But the flexibility of the kernel and the simplicity of the algorithm together with the robustness provided by the maximum margin principle makes SVM a good starting point for any binary classification problem and in many cases it turns out to yield the best available solution.

We mentioned earlier than one cannot only rely on the data to find a good decision function, but needs to control the complexity of the obtained solution. This is exactly what the maximum margin principle intends to do: among the solutions that present zero empirical error, we choose the simplest one (where simple here means linear and with a large margin, but this is purely a convenience choice). The rationale is that we expect to be able to better explain the future unseen patterns with a simple solution since it may then capture the informative structure of the data. This is what VC theory is basically about, minimizing the empirical error and controlling the complexity (or capacity) of the obtained classifier.

To solve an SVM, we will have to find a hyperplane that correctly classifies the data, i.e. satisfies $\boldsymbol{\phi}^T(\mathbf{x}_i)\mathbf{w} + b > 0$ for every $\mathbf{x}_i$ with class label $y_i = 1$ and $\boldsymbol{\phi}^T(\mathbf{x}_i)\mathbf{w} + b < 0$ for every $\mathbf{x}_i$ with class label $y_i = -1$. These restrictions are usually expressed as: $y_i(\boldsymbol{\phi}^T(\mathbf{x}_i)\mathbf{w} + b) \geq \rho$, where $\rho/\|\mathbf{w}\|$ is the minimal distance of the samples of each class to the linear decision function

in the feature space ($\boldsymbol{\phi}^T(\mathbf{x})\mathbf{w} + b = 0$). To maximize the margin, we should maximize $\rho/\|\mathbf{w}\|$. As any multiplicative factor over ρ and the norm of $\mathbf{w}$ would lead to the same solution, we can fix one of them ($\rho = 1$) and maximize $1/\|\mathbf{w}\|$. The maximum of $1/\|\mathbf{w}\|$ can be obtained as the minimum of $\|\mathbf{w}\|$ or equivalently as the minimum of $\|\mathbf{w}\|^2$. The SVM optimisation problem is:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \tag{1}$$

subject to:

$$y_i(\boldsymbol{\phi}^T(\mathbf{x}_i)\mathbf{w} + b) \geq 1 \tag{2}$$

which is a quadratic problem (convex), that can be easily solved. A typical approach for solving such a problem is to first convert it into the Wolfe *dual* problem [14] (using the Lagrange multipliers method) which is an optimization problem expressed in terms of auxiliary variables but which is equivalent. It has the form

$$\max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{3}$$

subject to $\sum_{i=1}^{n} \alpha_i y_i = 0$ and $\alpha_i > 0$, where the SVM solution is obtained as $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{\phi}^T(\mathbf{x}_i)$. Predicting the label of unseen data can be made, as well, using kernels, once the definition of $\mathbf{w}$ as a linear combination of the samples in the feature space is used, i.e. $f(\mathbf{x}) = sign\left(\sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b\right)$. There exist other approaches to solve the SVM optimization problem, for example using an Iterative Re-Weighted Least Square procedure [15].

The name of the algorithm is due to the fact that only a few samples have a non-zero α_i, and they are called the support vectors. They fulfill the restriction in (2) with equality and, consequently, are the samples closest to the linear decision function. We show a simple example in Figure 2 in a 2 dimensional space, in which the support vectors are represented as stars. There, one can readily check that the obtained solution is the furthest apart from all the samples maximizing the margin between the samples and the linear decision function.

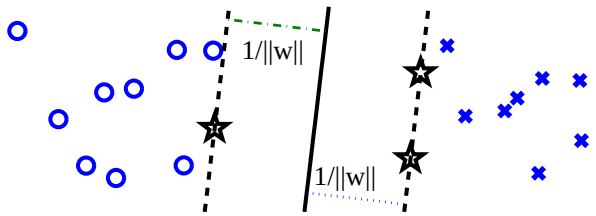

Figure 2: Maximal margin hyperplane

The SVM has been extended to solve problems that are not linearly separable in the feature space. This is done by softening the correct classification constraint by allowing some violation of (2). The constraint is replaced by $(y_i(\boldsymbol{\phi}^T(\mathbf{x}_i)\mathbf{w} + b) \geq 1 - \xi_i)$ where the ξ are non-negative *slack*

variables. Of course, one has to pay a penalty for misclassifying an example. The one-norm of the slacks is added to the objective functional weighted by a parameter C ($\min \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$) which controls the trade-off between misclassification of the training data and achieved margin. The corresponding dual to this so-called 'soft-margin' formulation is the same as (3) with the additional constraint $\alpha_i \leq C$ whose effect is to limit the influence an outlier can have on the final decision boundary.

To illustrate how SVMs work we are going to classify handwritten letters from the Spanish alphabet. We have gathered 580 samples of each letter (upper and lower cases) from 580 different people, summing up to 274320 inputs. Each letter is a 28×28 dimension 8-bit-per-pixel image, we show some examples in Figure 3. To classify the letters we are going to train an individual classifier for each letter, confronting it with the rest of the letters, any new letter will be classified using the largest output of the binary classifiers (this is an standard procedure to construct multiclass SVMs [2]). We have preprocesses the samples so each input value is between 0 and 1 and we have solve this problem using a gaussian kernel with $\sigma = 20$ and $C = 10$, which have been set by cross validation. 500 instances of each letter has been used for training and the remaining 80 have been used for testing purposes. The obtained classifier has an error rate of 13.7%. We have also meassure the human error rate over 24 subjects, each of them looked at 180 samples, obtaining an error rate of 6.6%.

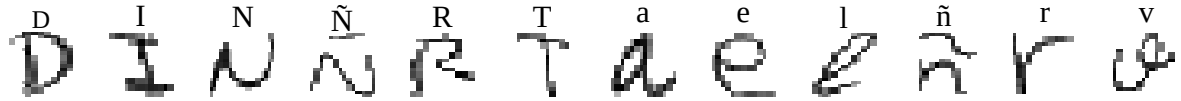

Figure 3: We show 12 examples of the letters in the training set.

Besides binary classification, the SVM algorithm has been generalized for solving regression estimation [16, 17], for multiclass [18, 19] and for multiple output problems [20, 21].

It is important at this point to mention that the SVM optimization problem can be written as a *regularization* problem:

$$\min_{f,b} \frac{1}{n}\sum_{i=1}^{n} L(y_i(f(\mathbf{x}_i) + b)) + \lambda\|f\|^2 \,,$$

where $f(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x})$ and $\|f\| := \|\mathbf{w}\|$ and L is a *loss function*. Choosing $L(z) = \max(0, 1 - z)$ gives the soft-margin SVM (also called 1-norm soft margin), while other choices give variants, like the 2-norm soft margin SVM with $L(z) = \max(0, 1 - z)^2$ [3] or the least-squares SVM with $L(z) = (1 - z)^2$ [22].

In the regularization setting, $\|f\|^2$ is called the regularizer and λ the regularization parameter. The role of the regularizer is to restrict the space in which the solution is searched, or equivalently

to control its capacity. One advantage of the SVM algorithm is that the regularizer has a direct geometric interpretation in terms of margin of separation of the classes. This is no longer the case in the regression estimation setting. In a sense, this lack of interpretability of the regularizer for regression may be a reason for the limited development of kernel methods in regression.

Finally, let us note that the parameters C or λ above have no direct interpretation and this limits the use of heuristics for choosing them (the only way of tuning them is by trying several possible values). It is thus relevant to mention the ν-SVM [23] which is, at first, a less intuitive parametrization of the SVM, but more meaningful for practitioners because the new parameter ν roughly represents the fraction of expected support vectors.

3.2 Unsupervised Learning Algorithms

One of the kernel algorithm that has been known from the mid-nineties is the Kernel Principal Component Analysis (K-PCA) [24]. This algorithm performs the well-known PCA in the feature space, so it looks for directions of largest variance that yields nonlinear directions in the input space. We show a toy example in Figure 4, where PCA is performed in both the input and

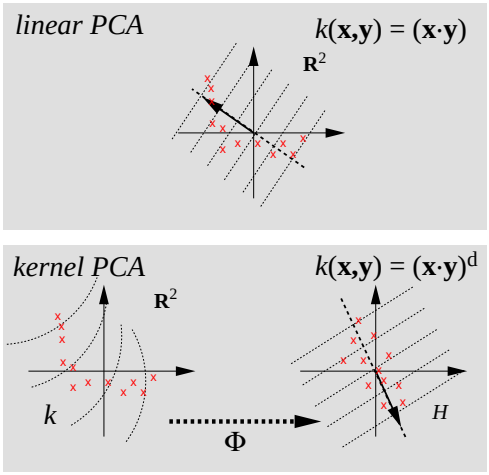

Figure 4: Linear vs. kernel PCA

feature space produced by a polynomial kernel. The result of the kernel PCA is to extract nonlinear components which describe the non-ellipsoidal shape of the data cloud. Besides, the interest K-PCA presents in itself, for feature extraction [4] or for obtaining nonlinear version of algorithms that cannot be readily kernelized [25], its development is also relevant, because it is not straightforward to express the PCA in terms of scalar products. Therefore, presenting this development can give some ideas on how to construct kernel algorithms.

The Principal Component Analysis solves the eigenvalue equation:

$$\lambda \mathbf{v} = \mathbf{C}\mathbf{v} \tag{4}$$

where λ and $\mathbf{v}$ are, respectively, the eigenvalues and eigenvectors of $\mathbf{C}$, which is the covariance matrix of the samples $\mathbf{x}_1 \ldots, \mathbf{x}_n$ in the feature space:

$$\mathbf{C} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{\phi}(\mathbf{x}_i) \boldsymbol{\phi}^T(\mathbf{x}_i) \tag{5}$$

for $\sum_{i=1}^{n} \boldsymbol{\phi}(\mathbf{x}_i) = 0$ [1].

As a covariance matrix, $\mathbf{C}$ is positive semidefinite, so that the resolution of the eigenvalue equation (4) will give only non-negative eigenvalues ($\lambda \geq 0$) for non zero eigenvectors $\mathbf{v} \in \mathcal{H} - \{\mathbf{0}\}$. However, the problem is expressed in terms of the covariance matrix, which is formed by outer products of the samples, not by inner products. To be able to introduce kernels, we need to obtain a representation of $\mathbf{v}$ in terms of $\boldsymbol{\phi}(\mathbf{x}_i)$. We replace in (4) the definition of $\mathbf{C}$ in (5) to get

$$\lambda \mathbf{v} = \mathbf{C}\mathbf{v} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{\phi}(\mathbf{x}_i) \boldsymbol{\phi}^T(\mathbf{x}_i) \mathbf{v}$$

where one can realise that for nonzero eigenvalues the eigenvector is constructed as a linear combination of the input vectors in the feature space (this is another form of the Representer theorem mentioned earlier):

$$\mathbf{v} = \sum_{i=1}^{n} \boldsymbol{\phi}(\mathbf{x}_i) \alpha_i = \boldsymbol{\Phi}^T \boldsymbol{\alpha}$$

where $\alpha_i = \frac{\boldsymbol{\phi}^T(\mathbf{x}_i)\mathbf{v}}{\lambda n}$. We can use this definition of $\mathbf{v}$ to solve the PCA problem with kernels, leading to the following eigenvalue equation:

$$\lambda \boldsymbol{\Phi}^T \boldsymbol{\alpha} = \frac{1}{n} \boldsymbol{\Phi}^T \boldsymbol{\Phi} \boldsymbol{\Phi}^T \boldsymbol{\alpha} \tag{6}$$

If we premultiply it by the pseudoinverse of $\boldsymbol{\Phi}^T$, we are left with:

$$n\lambda (\boldsymbol{\Phi}\boldsymbol{\Phi}^T)^{-1} \boldsymbol{\Phi}\boldsymbol{\Phi}^T \boldsymbol{\alpha} = (\boldsymbol{\Phi}\boldsymbol{\Phi}^T)^{-1} \boldsymbol{\Phi}\boldsymbol{\Phi}^T \boldsymbol{\Phi}\boldsymbol{\Phi}^T \boldsymbol{\alpha}$$

which can be simplified to:

$$n\lambda \boldsymbol{\alpha} = \mathbf{H}\boldsymbol{\alpha} \tag{7}$$

which is an eigenvalue equation for the kernel matrix $\mathbf{H}$, where the eigenvectors $\boldsymbol{\alpha}$ are the expansions coefficients of the eigenvectors $\mathbf{v}$ for the covariance matrix $\mathbf{C}$.

Once it has been solved, we will have n eigenvalues λ_j ($\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ corresponding to the $n\lambda$ in (7)). As the eigenvalue equation is solved for $\boldsymbol{\alpha}^j$ instead of $\mathbf{v}^j$, we will have to apply a normalization post processing to the $\boldsymbol{\alpha}^j$ to ensure that the eigenvalues $\mathbf{v}^j$ have unit norm in the feature space, therefore $\boldsymbol{\alpha}^j = \boldsymbol{\alpha}^j / \sqrt{\lambda_j}$, see [26] for further details.

[1] We will deal with non-centered data at the end of this section.

The expansion of any vector $\boldsymbol{\phi}(\mathbf{x})$ in the feature space can be calculated, as well, using kernels for the nonzero λ_j:

$$\langle v^j, \boldsymbol{\phi}(\mathbf{x}) \rangle = \sum_{i=1}^{n} \alpha_i^j \boldsymbol{\phi}^T(\mathbf{x}_i)\boldsymbol{\phi}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i^j k(\mathbf{x}_i, \mathbf{x}) \qquad \forall j = 1, \ldots, m \tag{8}$$

where m is the number of nonzero λ_j.

We will now deal with non-centered data in the feature space. In this case the data would have to be centered in the feature space and then the kernel matrix would have to be computed, this operation can be directly done using the kernel matrix of the data as follows: $\widetilde{\mathbf{H}} = \mathbf{H} - \mathbf{1}_n\mathbf{H} - \mathbf{H}\mathbf{1}_n + \mathbf{1}_n\mathbf{H}\mathbf{1}_n$, where $\mathbf{1}_n$ is a $n \times n$ matrix with all its entries equal to $1/n$ and being $\widetilde{\mathbf{H}}$ the kernel matrix of the centered data and the one to be used in the previous development.

We have plotted in Figure 5 the output of the KPCA algorithm for a 2D data set. We have run the KPCA with a gaussian kernel ($\sigma = 0.5$), the contour plots show the output of the KPCA to the eigenvectors with largest eigenvalue. It can be seen that the first three plots concentrate on distinguishing between clusters and the fourth it considers the intercluster structure.

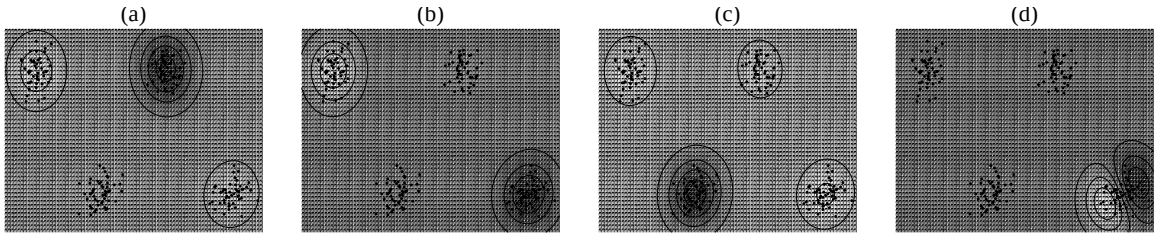

Figure 5: We show the contour lines of the KPCA algorithm for the first four eigenvectors.

Another type of unsupervised learning algorithm is the so-called *one-class* SVM whose goal is to estimate the support of the data distribution. It works by applying a variant of the SVM algorithm with all y_i set to 1. The result is a nonlinear boundary that encloses the data (or a prescribed fraction), which can be applied to the outliers or novelty detection problem [27].

Finally, other types of algorithms that are often used in signal processing have been recently extended to the nonlinear case via kernels. These include Vector Quantization [28], Independent Component Analysis [29, 30] and Canonical Correlation Analysis [31].

4 Learning the Kernel Matrix

We have already mentioned in Section 2 that given a feature space the best strategy one can follow is maximizing the margin between the classes. The problem that still needs to be addressed is finding the optimal feature space and, as we only know it by its kernel, it reduces to choosing the best kernel for the actual learning.

Each kernel corresponds to a space of functions along with a preference relation on this space. Indeed, the space of functions associated to a kernel is the space of linear hyperplanes in the corresponding feature space $f(\mathbf{x}) = \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}) + b$ and the preference is given to those functions whose weight vector has a small norm $\|\mathbf{w}\|$. If the desired (target) function can be represented by a weight vector with a small norm, the kernel algorithm will have a good performance since a small amount of data will be needed to identify the correct function.

This shows that the performance of the kernel algorithm will be highly dependent on the kernel and the best kernel depends directly on the problem at hand. This has two implications:

- If one has some prior knowledge about the problem and more precisely about the shape of the solution, it is possible to introduce this knowledge into the kernel in a principled way. This will significantly improve the performance of the kernel algorithm.
- On the other hand, if the kernel is not appropriate to the learning problem, the performance can be arbitrarily bad, so that one needs to tune the kernel.

So the bad news is that one needs prior knowledge to have good performance (there exist no universally good algorithm) but the good news is that some of this knowledge can be extracted from the data.

One approach is to tune the hyperparameter of a given family of functions, i.e. set the kernel width σ for the Gaussian Kernel $K(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|/2\sigma^2)$. This tuning can either be done using cross validation approaches [32] or using a performance bound [33]. However, this parametric approach has limited flexibility especially with non-vectorial data such as strings [34], in which popular kernels are not able to provide meaningful results. For this kind of problems, i.e. text categorization, gene expression or protein classification, people have mainly devised their own kernel incorporating prior knowledge. Further information about these techniques can be found in [35, 36, 37].

The second and more promising approach consists in trying to learn the kernel matrix directly from the data, instead of tuning a hyperparameter of a given parametric family of kernels. The goal this approach seeks is two-fold. First, not to be restricted to a family of kernels that might not be able to capture precisely the required nonlinear mapping for the problem at hand. Second, to stop tuning any hyperparameter, obtaining a robust learning procedure which is able to provide a kernel matrix without setting any learning parameter for any given problem.

The first of such methods was introduced by Cristianini et al. [38], in which the objective is to align the kernel matrix with the samples labels, using a normalized Frobenious norm:

$$\hat{A} = \frac{\langle \mathbf{H}, \mathbf{y}\mathbf{y}^T \rangle_F}{\sqrt{\langle \mathbf{H}, \mathbf{H} \rangle \langle \mathbf{y}\mathbf{y}^T, \mathbf{y}\mathbf{y}^T \rangle}} = \frac{\mathbf{y}^T\mathbf{H}\mathbf{y}}{n \sum_{i,j=1}^{n} (\mathbf{H})_{ij}^2}$$

The larger this norm is, the larger the alignment would be between the label vector $\mathbf{y}$ and the kernel matrix $\mathbf{H}$. The proposed algorithm decompose $\mathbf{H}$ as a sum of kernel matrices, $\mathbf{H} = \sum_i \alpha_i \mathbf{H}_i$ and maximize the kernel target alignment with respect to α_i. To ensure the obtained kernel matrix is positive semi-definite, the α_i are restricted to be nonnegative. This work has been extended by Lanckriet et al. [39] in which the kernel target alignment was maximized using a semi-definite programming procedure [40], which ensure the kernel matrix being positive semi-definite. In this approach the overaligment (overfitting) is controlled using transductive learning [2], in which the labels have to be predicted over a known test set, instead of inductive learning, in which the test set can be potentially any point in the input space.

Two recent proposals [41] and [42] have come to provide a better understanding on learning the kernel matrix. In [41], a regularized functional is constructed to solve the learning problem, in which a term is added to control the kernel complexity, in the same way as it is done for controlling the decision function complexity. They use an extension of the Representer theorem to construct kernels for kernels, which they name superkernels. Initially, one can think that in this case one would need to set the hyperparameters of the superkernels instead of the kernels parameters, so we have just translated the problem to another tuning procedure. The relevance of this approach is the relative insensitivity of the obtained solution to the hyperparameters of the proposed regularized functional, so the tuning has limited influence on the solution.

Finally, in [42], inspired by the work of [39], the authors propose a different measure for the learning kernel matrix, instead of maximizing the kernel alignment, minimizing $\frac{\sqrt{tr\mathbf{H}}}{Mn}$, where $tr\mathbf{H}$ is the trace of the kernel matrix (after centering), M is the margin of the classifier and n the number of training samples. They propose to fix the trace of the kernel matrix and maximize the margin of the classifier with respect to the kernel matrix. The space of possible matrices is chosen as the convex hull of a finite number of given matrices which yields a significantly simpler algorithm than the semi-definite programming approach proposed in [39].

Several questions remain open with regard to learning of the kernel matrix. It is not clear yet what is the best criterion to optimize. Also computationally efficient procedures have to be designed. Finally the choice of the class of kernel matrices to be considered is an important aspect of these methods. Therefore, important developments can be expected in this domain.

5 Signal Processing Applications

Kernel algorithms have presented a steady growth in the recent years. The SVM has been applied to several signal processing and communications applications. Some of them are direct application of the standard algorithms for detection or estimation and others incorporate

prior knowledge to the learning process, either incorporating virtual training samples or by constructing a relevant kernel for the given problem. The SVM has been applied to: speech and audio processing (speech recognition [43], speaker identification [44], extraction of audio features [45]), image processing and computer vision (face detection and recognition [13], image coding [28], handwritten digit recognition [12]), communications (channel equalization [46, 47], non-stationary channel models [48], multi-user detection [49]), time series estimation (chaotic time series [50], financial series [51]), and others (text [34, 37] and protein [35] classification) in which string kernels have been defined and used with great success. This list does not intend to be exhaustive but shows the diversity of problems that can be addressed with an algorithm and its variants. We will now revisit some of these applications.

5.1 Handwritten digit recognition

Handwritten digit recognition was the first real application solved using SVMs [1], and it can be considered the SVM killer application, because it improved the best previous result using the SVM directly over the unprocessed images of the handwritten digits [12] (the input vector are the pixels of the images). This application has been further improved by using virtual training samples [52], which consists of constructing new samples from the initial ones. The virtual samples are formed by making the training digit thinner/coarser and by slightly translating/rotating it. These extra samples will incorporate prior knowledge to the problem, because of a "5" written with a thinner pen is still a five and an "8" rotated 5 or 10 degrees is still an eight. The SVM trained with 12 virtual digit per digit is actually the best available classifier for the MNIST digit recognition with a misclassification rate of 0.6%. Also a new method for incorporating invariances into the kernel matrix, taking into account the positioning of the training samples has further improved the classification rate [53].

5.2 Speech recognition

Speech recognition has been usually solved applying Hidden Markov Models (HMM) trained using maximum likelihood estimates. The limitation of this approach is that the probability density function of the models is unknown and any assumption could be misleading. There is a proposal in which the SVM has been incorporated into the HMM decision process [43], to be able to process the voice as it is generated, but at the same time make the decisions according to the maximum margin rule that will ensure best separability under any density model.

5.3 Communications

Channel equalization and estimation is one of the key issues in digital communication because it involves linear (Inter-Symbolic Interference) and nonlinear (amplifiers and converters) distortions and, in many situation, the training sequence need to be short, not to reduce the payload bits. Channel equalization is needed, because the communication channels introduces inter-symbolic interference, each transmitted symbol is spread between some contiguous received symbols. Therefore a linear transversal filter, that contains several consecutive symbols, is used to estimate the incoming symbol. Least squares are frequently employed to compute the weights of these filters. SVMs have been used with great success over nonlinear channel directly using linear and nonlinear kernel with very short training sequences [46] and some modifications using hidden Markov models have been also proposed [47]. One of the most relevant features in equalization is that the channel does not need to be stationary, needing to change the decision function without new training sequence. The SVM has been originally proposed for i.i.d. training sample, and consequently it has to be modified for using it over time varying communication channels. This has been obtained by incorporating the prior information about the evolution with time in the SVM cost and margin [48].

6 Discussion and Perspectives

In this paper, we have tried to give an overview of kernel methods focusing on their aspects that we consider most relevant and that can be expected to have a great development in the forthcoming years. We have explained that the maximization of the margin in classification problems provides an intuitively reasonable measure for generalization in a given feature space. We have also explained how this feature space can be constructed from the data. These are two basic elements in the success of support vector machines and they can be extended to other settings to construct meaningful kernel methods from linear algorithms.

We would like to say that this paper reflects our own point of view on kernel methods and the views that we expressed regarding their justification or their future developments may not be shared by the whole community. However, we hope that this will encourage people to use these methods and to express their own views in turn, to the benefit of all.

Acknowledgment

We would like to thank Bernhard Schölkopf for providing Figures 1 and 4 and Patrice López-Ortega for providing the Figures in 3 and the shown results for the handwritten letter recognition.

References

[1] B. Boser, I. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proc. 5th COLT*, D. Haussler, Ed. 1992, pp. 144–152, ACM Press.

[2] V. N. Vapnik, *Statistical Learning Theory*, Wiley, 1998.

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

[4] B. Schölkopf and A. Smola, *Learning with kernels*, MIT Press, 2002.

[5] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with application in pattern recognition," *IEEE Transactions on Electronic Computers*, vol. 14, pp. 304–314, 1965.

[6] I. Steinwart, "On the influence of the kernel on the consistency of support vector machines," *J. of Machine Learning Research*, vol. 2, pp. 67–93, 2001.

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

[8] C. K. I. Williams and C. E. Rasmussen, "Advances in neural processing systems," in *NIPS*, D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds., Cambridge, MA, 1996, pp. 598–604, M.I.T. Press.

[9] A. N. Tikhonov and V. Y. Arsenin, *Solution of Ill-posed Problems*, Wiston, 1977.

[10] V. N. Vapnik and A. Lerner, "Pattern recognition using generalized portrait method," *Automation and Remote Control*, vol. 24, 1963.

[11] V. N. Vapnik, *Estimation of Dependences Based on Empirical Data*, Springer–Verlag, 1982.

[12] B. Schölkopf, K.-K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik, "Comparing support vector machines with Gaussian kernels to radial basis function classifiers," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2758–2765, Nov. 1997.

[13] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: An application to face detection," in *Proc. IEEE Conf. CVPR*, 1997, pp. 130–136.

[14] R. Fletcher, *Practical Methods of Optimization*, Wiley, Chichester, 2 edition, 1987.

[15] F. Pérez-Cruz, P. L. Alarcón-Diana, A. Navia-Vázquez, and A. Artés-Rodríguez, "Fast training of support vector classifiers," in *NIPS 13*, T. Leen, T. Dietterich, and V. Tresp, Eds., Cambridge, MA, Nov. 2000, pp. 734–740, M.I.T. Press.

[16] V. N. Vapnik, S. Golowich, and A. Smola, "Support vector method for function approximation, regression estimation, and signal processing," in *NIPS*, M. Mozer, M. Jordan, and T. Petsche, Eds., Cambridge, MA, 1997, pp. 169–184, M.I.T. Press.

[17] H. Drunker, C. Burges, L. Kaufman, A. Smola, and V. N. Vapnik, "Support vector regression machines," in *NIPS*, M. Mozer, M. Jordan, and T. Petsche, Eds., Cambridge, MA, 1997, pp. 155–161, M.I.T. Press.

[18] J. Weston and C. Watkins, "Multi-class support vector machines," in *ESANN*, 1999.

[19] E. L. Allwein, R. E. Schapire, and Y. Singer, "Reducing multiclass to binary: A unifying approach for margin classifiers," *J. of Machine Learning Research*, pp. 113–141, 2000.

[20] A. Elisseeff and J. Weston, "kernel method for multi-labelled classification," in *NIPS 14*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds., Cambridge, MA, 2001, MIT Press.

[21] F. Pérez-Cruz, G. Camps, E. Soria, J. Pérez, A.R. Figueiras-Vidal, and A. Artés-Rodríguez, "Multi-dimensional function approximation and regression estimation," in *ICANN02*, Madrid, Spain, 2002, Springer.

[22] J. A. K. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Processing Letters*, vol. 9, no. 3, pp. 293–300, 1999.

[23] B. Schölkopf, A. Smola, R. Williamson, and P. L. Bartlett, "New support vector algorithms," *Neural Computation*, vol. 12, no. 5, pp. 1207–1245, May 2000.

[24] B. Schölkopf and A. Smola K. R. Müller, "Kernel principal component analysis," in *ICANN97*, Berlin, Germany, 1997, Springer.

[25] J. Weston, O. Chapelle, A. Elisseeff, B. Schölkopf, and V. Vapnik, "Kernel dependency estimation," in *NIPS 15*, S. Becker, S. Thrun, and K. Obermayer, Eds. 2002, MIT Press.

[26] B. Schölkopf, A. J. Smola, and K.-R. Müller, "Kernel principal component analysis," in *Advances in Kernel Methods— Support Vector Learning*, B. Schölkopf, C. J. C. Burges, and A. J. Smola, Eds., pp. 327–352. M.I.T. Press, Cambridge, (MA), 1999.

[27] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt, "Support vector method for novelty detection," in *NIPS 12*, S. A. Solla, T. K. Leen, and K.-R. Müller, Eds. 1999, pp. 582–588, MIT Press.

[28] M. Tipping and B. Schölkopf, "A kernel approach for vector quantization with guaranteed distortion bounds," in *Artificial Intelligence and Statistics*, T. Jaakkola and T. Richardson, Eds. 2001, pp. 129–134, Morgan Kaufmann.

[29] F. Bach and M. Jordan, "Kernel independent component analysis," *J. of Machine Learning Research*, vol. 3, pp. 1–48, 2002.

[30] A. Gretton, R. Herbrich, and A. Smola, "Independence and kernel correlation," Preprint.

[31] M. Kuss and T. Graepel, "The geometry of kernel canonical correlation analysis," Preprint.

[32] G. Wahba, "Spline models for observational data," *CBMS-NFS Regional conference series in Applied Mathematics*, vol. 69, 1990.

[33] O. Chapelle, V. N. Vapnik, O. Bousquete, and S. Mukherjee, "Choosing multiple parameters for support vector machines," *Machine Learning*, vol. 46, no. 1, pp. 131–159, 2002.

[34] T. Joachims, "Text categorization with support vector machines," Tech. Rep. LS VIII, 23, University of Dortmund, Informatik, AI-Unit Collaborative Research Center on 'Complexity Reduction in Multivariate Data'., 1997.

[35] C. Leslie, E. Eskin, J. Weston, and W. S. Noble, "Mismatch string kernels for SVM protein classification," in *NIPS 15*, S. Becker, S. Thrun, and K. Obermayer, Eds. 2002, MIT Press.

[36] D. Haussler, "Convolutional kernels on discrete structures," Tech. Rep. UCSC-CRL-99-10, Computer science Department, University of California at Santa Cruz, 1999.

[37] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins, "Text classification using string kernels," *J. of Machine Learning Research*, pp. 419–444, 2002.

[38] N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and j. Kandola, "On kernel target alignment," in *NIPS 14*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds. 2001, MIT Press.

[39] G.R.G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, and M.I. Jordan, "Learning the kernel matrix with semi-definite programming," in *19th ICML*, Sidney, 2002, pp. 57–64.

[40] L. Vandenberghe and S. Boyd, "Semidefinite programming," vol. 38, pp. 40–95, 1996.

[41] C. S. Ong, A. J. Smola, and R. C. Williamson, "Supperkernels," in *NIPS 15*, S. Becker, S. Thrun, and K. Obermayer, Eds. 2002, MIT Press.

[42] O. Bousquet and D. J.L. Herrmann, "On the complexity of learning the kernel matrix," in *NIPS 15*, S. Becker, S. Thrun, and K. Obermayer, Eds. 2002, MIT Press.

[43] N. Smith and M. Gales, "Speech recognition using SVMs," in *NIPS 14*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds. 2001, MIT Press.

[44] V. Wan and S. Renals, "Evaluation of kernel methods for speaker verification and identification," in *Proc. ICASSP 2002*, Orlando, FL, 2002.

[45] C.J.C. Burges, J.C. Platt, and S. Jana, "Extracting noise-robust features from audio data," in *Proc. ICASSP 2002*, Orlando, FL, 2002.

[46] F. Pérez-Cruz, P. Alarcón-Diana, A. Navia-Vázquez, and A. Artés-Rodríguez, "SVC-based equalizer for burst TDMA transmissions," *Signal Processing*, vol. 81, pp. 1681–1693, 2001.

[47] S. Chakrabartty and G. Cauwenberghs, "Sequence estimation qnd channel equalization using forward decoding kernel machines," in *Proc. ICASSP 2002*, Orlando, FL, 2002.

[48] F. Pérez-Cruz and A. Artés-Rodríguez, "Adaptive SVC for nonlinear channel equalization," in *Proc. of EUSIPCO'02*, Toulouse, France, 2002.

[49] S. Chen, A. K. Samingan, and L. Hanzo, "Support vector machine multiuser receiver for DS-CDMA signal in multipath channels," *IEEE Transactions on Neural Networks*, vol. 12, no. 3, pp. 604–611, 2001.

[50] D. Mattera and S. Haykin, "Support vector machines for dynamic reconstruction of a chaotic system," in *Advances in Kernel Methods— Support Vector Learning*, B. Schölkopf, C. J. C. Burges, and A. J. Smola, Eds., pp. 211–242. MIT Press, 1998.

[51] F. Pérez-Cruz, J. Afonso-Rodríguez, and J. Giner, "Estimating GARCH models using support vector machines," *Quantitative Finance*, pp. 1–10, 2003.

[52] D. DeCoste and B. Schoelkopf, "Training invariant support vector machines," *Machine Learning*, vol. 46, 2002.

[53] O. Chapelle and B. Scholkopf, "Incorporating invariances in non-linear SVMs," in *NIPS 14*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds. 2001, MIT Press.