Incorporating Invariances in Nonlinear Support Vector Machines

Olivier Chapelle
ochapell@ens-lyon.fr
École Normale Supérieure de Lyon

Bernhard Schölkopf
bs@conclu.de
Biowulf Technologies, New York

June 20, 2001

Abstract

The choice of an SVM kernel corresponds to the choice of a representation of the data in a feature space and, to improve performance, it should therefore incorporate prior knowledge such as known transformation invariances. We propose a technique which extends earlier work and aims at incorporating invariances in nonlinear kernels. We show on a digit recognition task that the proposed approach is superior to the Virtual Support Vector method, which previously had been the method of choice.

1 Introduction

In some classification tasks, an a priori knowledge is known about the invariances related to the task. For instance, in image classification, we know that the label of a given image should not change after a small translation or rotation.

More generally, we assume we know a local transformation $\mathcal{L}_t$ depending on a parameter t (for instance, a vertical translation of t pixels) such that any point $\mathbf{x}$ should be considered equivalent to $\mathcal{L}_t\mathbf{x}$, the transformed point. Ideally, the output of the learned function should be constant when its inputs are transformed by the desired invariance.

It has been shown [1] that one can not find a non-trivial kernel which is globally invariant. For this reason, we consider here local invariances and

for this purpose we associate at each training point $\mathbf{x}_i$ a *tangent vector* $d\mathbf{x}_i$,

$$\begin{aligned} d\mathbf{x}_i &= \lim_{t \to 0} \frac{1}{t} \left(\mathcal{L}_t \mathbf{x}_i - \mathbf{x}_i \right) \\ &= \left. \frac{\partial}{\partial t} \right|_{t=0} \mathcal{L}_t \mathbf{x}_i \end{aligned}$$

In practice $d\mathbf{x}_i$ can be either computed by finite difference or by differentiation. Note that generally one can consider more than one invariance transformation. The multi invariance case is usually a straight forward extension of the single one.

A common way of introducing invariances in a learning system is to add the perturbed examples $\mathcal{L}_t \mathbf{x}_i$ in the training set [7]. Those points are often called *virtual examples*. In the SVM framework, when applied only to the SVs, it leads to the *Virtual Support Vector* (VSV) method [10].

An alternative to this is to modify directly the cost function in order to take into account the tangent vectors. This has been successfully applied to neural networks [14] and linear Support Vector Machines [12]. The aim of the present work is to extend these methods to the case of nonlinear SVMs which will be achieved mainly by using the kernel PCA trick [13].

The paper is organized as follows. After introducing the basics of Support Vector Machines in section 2, we recall the method proposed in [12] to train invariant linear SVMs (section 3). In section 4, we show how to extend it to the nonlinear case and finally experimental results are provided in section 5.

2 Support Vector Learning

We introduce some standard notations for SVMs; for a complete description, see [17]. Let $\{(\mathbf{x}_i, y_i)\}_{1 \leq i \leq \ell}$ be a set of training examples, $\mathbf{x}_i \in \mathcal{X}$, belonging to classes labeled by $y_i \in \{-1, 1\}$. The domain $\mathcal{X}$ is often, but not necessarily, taken to be $\mathbb{R}^d$, $d \in \mathbb{N}$. In kernel methods, we map these vectors into a feature space using a kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ that defines an inner product in this feature space. The decision function given by an SVM is the maximal margin hyperplane in this space,

$$g(\mathbf{x}) = \operatorname{sign}(f(\mathbf{x})), \text{ where } f(\mathbf{x}) = \left(\sum_{i=1}^{\ell} \alpha_i^0 y_i K(\mathbf{x}_i, \mathbf{x}) + b \right). \qquad (1)$$

The coefficients α_i^0 are obtained by maximizing the functional

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j K_(\mathbf{x}_i, \mathbf{x}_j) \tag{2}$$

under the constraints

$$\sum_{i=1}^{\ell} \alpha_i y_i = 0 \quad \text{and} \quad \alpha_i \geq 0.$$

This formulation of the SVM optimization problem is called the *hard margin* formulation since no training errors are allowed. Every training point satisfies the inequality $y_i f(\mathbf{x}_i) \geq 1$; points $\mathbf{x}_i$ with $\alpha_i > 0$ satisfy it as an equality. These points are called *support vectors*.

Dealing with non-separability For the non-separable case, one needs to allow training errors which results in the so called *soft margin* SVM algorithm [3]. It can be shown that soft margin SVMs with quadratic penalization of errors can be considered as a special case of the hard margin version with the modified kernel [3, 4]

$$\mathbf{K} \leftarrow \mathbf{K} + \frac{1}{C}\mathbf{I}, \tag{3}$$

where $\mathbf{I}$ is the identity matrix and C a parameter penalizing the training errors. However, in the remainder of the paper, we will focus on the hard margin SVM.

3 Invariances for Linear SVMs

For linear SVMs, one wants to find a hyperplane whose normal vector $\mathbf{w}$ is as orthogonal as possible to the tangent vectors. This can be easily understood from the equality

$$f(\mathbf{x}_i + d\mathbf{x}_i) - f(\mathbf{x}_i) = \mathbf{w} \cdot d\mathbf{x}_i.$$

For this purpose, it has been suggested [12] to minimize the functional

$$(1-\gamma)\mathbf{w}^2 + \gamma \sum_{i=1}^{n} (\mathbf{w} \cdot d\mathbf{x}_i)^2 \tag{4}$$

subject to the constraints

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1.$$

The parameter γ trades off between normal SVM training ($\gamma = 0$) and full enforcement of the orthogonality between the hyperplane and the invariance directions ($\gamma \to 1$).

Let us introduce

$$C_\gamma = \left((1-\gamma) I + \gamma \sum_i d\mathbf{x}_i d\mathbf{x}_i^\top \right)^{1/2}, \tag{5}$$

the square root of the regularized covariance matrix of the tangent vectors. The functional (4) then reads

$$\mathbf{w}^\top C_\gamma^2 \mathbf{w}.$$

Introducing $\tilde{\mathbf{w}} = C_\gamma \mathbf{w}$ and $\tilde{\mathbf{x}}_i = C_\gamma^{-1} \mathbf{x}_i$, we obtain the equivalent optimization problem:

$$\min_{\tilde{\mathbf{w}}} \tilde{\mathbf{w}}^2 \tag{6}$$

under constraints

$$y_i (\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}_i + b) \geq 1.$$

Here, we made use of the equality $C_\gamma^{-1} \mathbf{w} \cdot \mathbf{x}_i = \mathbf{w} \cdot C_\gamma^{-1} \mathbf{x}_i$, which is valid because C_γ is symmetric. Note also that C_γ is strictly positive definite (and thus invertible) if $\gamma < 1$. For this reason, in the rest of the paper, we will assume $\gamma < 1$.

The optimization problem (6) is the standard SVM one where the training points $\mathbf{x}_i$ have been linearly preprocessed using the matrix C_γ^{-1}.

The output value of the decision function on a test point is :

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{w} \cdot \mathbf{x}_i + b \\ &= \tilde{\mathbf{w}} \cdot C_\gamma^{-1} \mathbf{x} + b \\ &= \sum_{i=1}^{n} \alpha_i y_i \; C_\gamma^{-1} \mathbf{x}_i \cdot C_\gamma^{-1} \mathbf{x} + b \end{aligned}$$

We also obtain the standard formulation of an SVM output, but where the test point is first multiplied by C_γ^{-1}.

To conclude, one could say that a linear invariant SVM is equivalent to a standard SVM where the input space has been transformed through the linear mapping

$$\mathbf{x} \to C_\gamma^{-1} \mathbf{x}.$$

In [12], it was shown that this method leads to significant improvements in linear SVMs, and to small improvements when used as a linear preprocessing step in nonlinear SVMs. The latter, however, was a hybrid system

with unclear theoretical foundations. In the next section we show how to deal with the nonlinear case in a principled way.

4 Extension to the nonlinear case

In the nonlinear case, data are first mapped into a high-dimensional feature space where a linear decision boundary is computed. To extend directly the previous analysis to the nonlinear case, one would need to compute the matrix C_γ in feature space,

$$C_\gamma = \left((1-\gamma) I + \gamma \sum_i d\Phi(\mathbf{x}_i) d\Phi(\mathbf{x}_i)^\top \right)^{1/2} \tag{7}$$

and the new kernel function

$$\tilde{K}(\mathbf{x}, \mathbf{y}) = C_\gamma^{-1}\Phi(\mathbf{x}) \cdot C_\gamma^{-1}\Phi(\mathbf{y}) = \Phi(\mathbf{x})^\top C_\gamma^{-2}\Phi(\mathbf{y}) \tag{8}$$

However, due to the high dimension of the feature space, it is impossible to do it directly. We propose two different ways for overcoming this difficulty. First, we need to introduce the so-called *kernel PCA map*.

4.1 The kernel PCA map

Even in the case of a high dimensional feature space $\mathcal{H}$, a training set $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ of size n when mapped to this feature space spans a subspace $E \subset \mathcal{H}$ whose dimension is at most n. The *kernel PCA map*, introduced in [13] and extended in [16] makes use of this idea.

Let $(\mathbf{v}_1, \ldots, \mathbf{v}_n) \in E^n$ be an orthonormal basis of E with each $\mathbf{v}_i$ being a principal axis of $\{\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_n)\}$. The kernel PCA map $\psi : \mathcal{X} \to \mathbb{R}^n$ is defined coordinatewise as

$$\psi_p(\mathbf{x}) = \Phi(\mathbf{x}) \cdot \mathbf{v}_p, \quad 1 \leq p \leq n.$$

Note that by definition, for all i and j, $\Phi(\mathbf{x}_i)$ and $\Phi(\mathbf{x}_j)$ lie in E and thus

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) = \psi(\mathbf{x}_i) \cdot \psi(\mathbf{x}_j). \tag{9}$$

This reflects the fact that if we retain all principal components, kernel PCA is just a basis transform in E, leaving the dot product of training points invariant.

Let us next give the decomposition of the principal directions $(\mathbf{v}_1, \cdots, \mathbf{v}_n)$. For $1 \leq p \leq n$, $\mathbf{v}_p$ can be written as

$$\mathbf{v}_p = \sum_{i=1}^{n} a_{ip} \Phi(\mathbf{x}_i). \tag{10}$$

The fact that $(\mathbf{v}_1, \cdots, \mathbf{v}_n)$ are the principal directions in feature space of the training set means that they are eigenvectors of the covariance matrix,

$$C\mathbf{v}_p = \lambda_p \mathbf{v}_p, \quad 1 \leq p \leq n \tag{11}$$

where[1]

$$C = \sum_{i=1}^{n} \Phi(\mathbf{x}_i) \Phi(\mathbf{x}_i)^\top. \tag{12}$$

Combining equations (10), (11) and (12), and multiplying on the left side by $\Phi(\mathbf{x}_k)^\top$, we get

$$\Phi(\mathbf{x}_k)^\top \left(\sum_{i=1}^{n} \Phi(\mathbf{x}_i) \Phi(\mathbf{x}_i)^\top \right) \sum_{j=1}^{n} a_{jp} \Phi(\mathbf{x}_j) = \lambda_p \Phi(\mathbf{x}_k)^\top \sum_{j=1}^{n} a_{jp} \Phi(\mathbf{x}_j),$$

which can be written as

$$\sum_{i,j=1}^{n} K(\mathbf{x}_i, \mathbf{x}_k) K(\mathbf{x}_i, \mathbf{x}_j) a_{jp} = \lambda_p \sum_{j=1}^{n} a_{jp} K(\mathbf{x}_k, \mathbf{x}_j), \quad 1 \leq p, k \leq n$$

In matrix notation, this last equality reads

$$K^2 a_p = \lambda_p K a_p,$$

which is satisfied whenever

$$K a_p = \lambda_p a_p. \tag{13}$$

Thus if a_p is an eigenvector of the kernel matrix K, its components can be used in the expansion (10). Note that the eigenvalues of K are the same as the ones of the covariance matrix (12).

We have now to enforce the constraint that $\mathbf{v}_p$ has unit length. Equations (10) and (13) yield

$$\mathbf{v}_p^2 = a_p^\top K a_p = \lambda_p a_p^2.$$

[1] We make the assumption that the data are centered in feature space. It is actually always possible to center them (cf [13] for details on how to do it)

Writing the eigendecomposition of K as

$$K = U\Lambda U^\top,$$

with U being an orthonormal matrix and Λ a diagonal one, we have $a_{ip} = U_{ip}/\sqrt{\lambda_p}$, and the kernel PCA map reads

$$\psi(\mathbf{x}) = \Lambda^{-1/2} U^\top \mathbf{k}(\mathbf{x}), \tag{14}$$

where

$$\mathbf{k}(\mathbf{x}) = (K(\mathbf{x}, \mathbf{x}_1), \ldots, K(\mathbf{x}, \mathbf{x}_n))^\top.$$

4.2 Decomposition of the Gram matrix of the tangent vectors

In order to be able to compute the new kernel (8), we propose to diagonalize the matrix C_γ (eq 7) using the same approach as the one described in section 4.1. Remember that in that section, we showed how it was possible to diagonalize the feature space covariance matrix in the PCA subspace (spanned by a training sample) by computing the eigendecomposition of the Gram matrix of those points. Presently, instead of having a set of training points $\{\Phi(\mathbf{x}_i)\}$, we have a set of tangent vectors $\{d\Phi(\mathbf{x}_i)\}$ and a tangent covariance matrix

$$C^t = \sum_{i=1}^{n} d\Phi(\mathbf{x}_i) d\Phi(\mathbf{x}_i)^\top. \tag{15}$$

Let us consider the Gram matrix K^t of the tangent vectors:

$$\begin{aligned} K^t_{ij} &= d\Phi(\mathbf{x}_i) \cdot d\Phi(\mathbf{x}_j) \\ &= K(\mathbf{x}_i + d\mathbf{x}_i, \mathbf{x}_j + d\mathbf{x}_j) - K(\mathbf{x}_i + d\mathbf{x}_i, \mathbf{x}_j) \\ &\quad - K(\mathbf{x}_i, \mathbf{x}_j + d\mathbf{x}_j) + K(\mathbf{x}_i, \mathbf{x}_j) \qquad (16) \\ &= d\mathbf{x}_i^\top \frac{\partial^2 K(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{x}_i \partial \mathbf{x}_j} d\mathbf{x}_j \qquad (17) \end{aligned}$$

This matrix K^t can be computed either by finite differences (equation 16) or with the analytical derivative expression given by equation (17). Note that for a linear kernel, $K(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{y}$, and (17) reads $K^t_{ij} = d\mathbf{x}_i^\top d\mathbf{x}_j$, which is a standard dot product between the tangent vectors.

As in section 4.1, we write the eigendecomposition of K^t as $K^t = U\Lambda U^\top$, which enables us to find the expansion of the eigenvectors $(\mathbf{v}_1, \ldots, \mathbf{v}_n)$ of C^t

corresponding to non-zero eigenvalues.

$$\mathbf{v}_p = \frac{1}{\sqrt{\lambda_p}} \sum_{i=1}^{n} U_{ip} d\Phi(\mathbf{x}_i). \tag{18}$$

We can complete the orthonormal family $(\mathbf{v}_1, \ldots, \mathbf{v}_n)$ to an orthonormal basis of $\mathcal{H}$, $(\mathbf{v}_1, \ldots, \mathbf{v}_n, \mathbf{v}_{n+1}, \ldots, \mathbf{v}_{d_\mathcal{H}})$ with $d_\mathcal{H} = \dim \mathcal{H} \leq \infty$.

In this basis, C^t is diagonal

$$C^t = V \mathrm{D}(\lambda_1, \ldots, \lambda_n, 0, \ldots, 0) V^\top,$$

V being an orthonormal matrix whose columns are the $\mathbf{v}_i$. Remember from equation (8) that we are actually interested in C_γ^{-2}

$$C_\gamma^{-2} = V \mathrm{D} \left(\frac{1}{\gamma\lambda_1 + 1 - \gamma}, \ldots, \frac{1}{\gamma\lambda_n + 1 - \gamma}, \frac{1}{1-\gamma}, \ldots, \frac{1}{1-\gamma} \right) V^\top. \tag{19}$$

To be able to compute the new kernel function (8), we need to determine the projection of a point in feature space $\Phi(\mathbf{x}_i)$ onto one of the vectors $\mathbf{v}_p$. From equation (18), we have, for $1 \leq p \leq n$,

$$\begin{aligned} \Phi(\mathbf{x}) \cdot \mathbf{v}_p &= \frac{1}{\sqrt{\lambda_p}} \sum_{i=1}^{n} U_{ip} (K(\mathbf{x}_i + d\mathbf{x}_i, \mathbf{x}) - K(\mathbf{x}_i, \mathbf{x})) \\ &= \frac{1}{\sqrt{\lambda_p}} \sum_{i=1}^{n} U_{ip} d\mathbf{x}_i^\top \frac{\partial K(\mathbf{x}_i, \mathbf{x})}{\partial \mathbf{x}_i}. \end{aligned} \tag{20}$$

For $p > n$, however, the dot product $\Phi(\mathbf{x}) \cdot \mathbf{v}_p$ is unknown. The trick is to rewrite equation (8) as

$$\tilde{K}(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x})^\top \left(C_\gamma^{-2} - \frac{1}{1-\gamma} I \right) \Phi(\mathbf{y}) + \frac{1}{1-\gamma} K(\mathbf{x}, \mathbf{y}) \tag{21}$$

From equation (19), one can easily see that the eigenvalues of $C_\gamma^{-2} - \frac{1}{1-\gamma} I$ associated with the eigenvectors $\mathbf{v}_p$ are zero for $p > n$.

Combining equations (19), (20) and (21) we finally get

$$\begin{aligned} \tilde{K}(\mathbf{x}, \mathbf{y}) &= \frac{1}{1-\gamma} K(\mathbf{x}, \mathbf{y}) + \\ &\quad \sum_{p=1}^{n} \Phi(\mathbf{x}) \cdot \mathbf{v}_p \left(\frac{1}{\gamma\lambda_p + 1 - \gamma} - \frac{1}{1-\gamma} \right) \Phi(\mathbf{y}) \cdot \mathbf{v}_p \end{aligned}$$

$$= \frac{1}{1-\gamma}K(\mathbf{x},\mathbf{y}) + \sum_{p=1}^{n}\frac{1}{\lambda_p}\left(\frac{1}{\gamma\lambda_p+1-\gamma}-\frac{1}{1-\gamma}\right)$$
$$\left(\sum_{i=1}^{n}U_{ip}\ d\mathbf{x}_i^\top\frac{\partial K(\mathbf{x}_i,\mathbf{x})}{\partial\mathbf{x}_i}\right)\left(\sum_{i=1}^{n}U_{ip}\ d\mathbf{x}_i^\top\frac{\partial K(\mathbf{x}_i,\mathbf{y})}{\partial\mathbf{x}_i}\right)$$

4.3 Decomposition of the Gram matrix of the input vectors

A drawback of the previous approach appears when one wants to deal with multiples invariances (i.e. more than one tangent vector per training point). Indeed, it requires to diagonalize the matrix K^t (cf eq 16) whose size is equal to the number of invariances.

If one wants to introduce multiple invariances, we propose an alternative method. The idea is to use directly the empirical kernel map described in section 4.1. Remember that a training sample of size n span a subspace E whose dimension is at most n. The empirical kernel map ψ (cf eq 14) gives in an orthonormal basis the components of a point $\Phi(\mathbf{x})$ projected on E.

From equation (9), it is easy to see that

training a nonlinear SVM on $\{\mathbf{x}_1,\ldots,\mathbf{x}_n\}$ is equivalent to training a linear SVM on $\{\psi(\mathbf{x}_1),\ldots,\psi(\mathbf{x}_n)\}$ and thus, thanks to the nonlinear mapping ψ, we can work directly in the linear space E and use exactly the technique described for invariant *linear* SVMs (section 3). However the invariance directions $d\Phi(\mathbf{x}_i)$ do not necessarily belong to E. By projecting them onto E, some information might be lost. The hope is that this approximation will give a similar decision function to the exact one obtained in section 4.2.

Finally, the proposed algorithm consists in training an invariant linear SVM as described in section 3 with training set $\{\psi(\mathbf{x}_1),\ldots,\psi(\mathbf{x}_n)\}$ and with invariance directions $\{d\psi(\mathbf{x}_1),\ldots,d\psi(\mathbf{x}_n)\}$, where

$$d\psi(\mathbf{x}_i) = \psi(\mathbf{x}_i + d\mathbf{x}_i) - \psi(\mathbf{x}_i),$$

which can be expressed from equation (14) as

$$d\psi(\mathbf{x}_i)_p = \frac{d\mathbf{x}_i^\top}{\sqrt{\lambda_p}}\sum_{j=1}^{n}U_{jp}\frac{\partial K(\mathbf{x}_i,\mathbf{x}_j)}{\partial\mathbf{x}_i}.$$

Note that we can try to have an idea of how much information has been lost by projecting $d\Phi(\mathbf{x}_i)$ on E with the ratio

$$\frac{||d\psi(\mathbf{x}_i)||}{||d\Phi(\mathbf{x}_i)||} \leq 1 \tag{22}$$

4.4 Comparisons with the VSV method

One might wonder what is the difference between enforcing an invariance and just adding the virtual examples $\mathcal{L}_t \mathbf{x}_i$ in the training set. Indeed the two approaches are related and some equivalence can be shown [6].

So why not just add virtual examples ? This is the idea of the Virtual Support Vector (VSV) method [10]. The reason is the following: if a training point $\mathbf{x}_i$ is far from the margin, adding the virtual example $\mathcal{L}_t \mathbf{x}_i$ will not change the decision boundary since neither of the points can become a support vector. Hence adding virtual examples in the SVM framework enforces invariance *only around the decision boundary*, which, as an aside, is the main reason why the virtual SV method only adds virtual examples generated from points that were support vectors in the earlier iteration.

One might argue that the points which are far from the decision boundary do not provide any information anyway. On the other hand, there is some merit in not only keeping the output label invariant under the transformation $\mathcal{L}_t$, but also the *real-valued output*. This can be justified by seeing the distance of a given point to the margin as an indication of its class-conditional probability [8]. It appears reasonable that an invariance transformation should not affect this probability too much.

The situation can be understood from the toy picture of figure 1. The point at the bottom right corner might not be a support vector, but it is still worth enforcing that the level curves around that point follow the direction given by its tangent vector.

4.5 Speeding-up

The complexity of computing the kernel PCA map $\psi(\mathbf{x})$ (equation 14) described in section 4.1 is of the order n^2: E is an n-dimensional space and each basis vector is expressed as a combination of n training points (cf equation 18). There are thus two ways of reducing the complexity:

- Reduce the dimension of E by keeping only the leading principal components
- Express each basis vector $\mathbf{v}_p$ as a sparse linear combination of the training points.

The interested reader might also look at [18] for more details.

A straightforward way to implement this idea for the method described in section 4.3 is to select a random subset of the training points of size

$m << n$ and to work in the subspace E' spanned by those m points. In practice, this means that the kernel PCA map (14) can be computed only from those m training examples. Of course, there is a loss of information by projecting on E' instead of E, but if the eigenvalues of the kernel function decay quickly enough and m is large enough, then one can hope that with high probability $||P_E(\mathbf{x}) - P_{E'}(\mathbf{x})||$ is small (P being the projection operator) [15].

The method described in section 4.2 amounts to finding the linear invariant hyperplane in the subspace spanned by the tangent vectors (in feature space) whereas the one presented in section 4.3 amounts to doing the same thing but in the subspace spanned by the training points. More generally, one could consider an hybrid method where the linear invariant hyperplane is constructed in the linear space spanned by a subset of the tangent vectors and a subset of the training points. This will be the topic of further research.

5 Experiments

In our experiments, we compared a standard SVM with several methods taking into account invariances:

- Standard SVM with virtual examples (cf. the VSV method [10]) [VSV]
- Invariant SVM as described in section 4.2 [ISVM]
- Invariant hyperplane in kernel PCA coordinates as described in section 4.3 [IH_{KPCA}]
- SVM trained with points preprocessed through the linear invariant hyperplane method [LIH]

The last method refers to [12] where the authors describe the invariant linear SVM (see also section 3). In their experiments, they linearly preprocessed the training and test points using C_γ^{-1} (equation 5). After, instead of training a linear SVM, they trained a nonlinear SVM. Even though there is no guarantee for such an approach to work, experimental results on the NIST set showed a slight improvement over standard nonlinear SVM.

Experiments have been carried on a toy problem and a digit recognition dataset.

5.1 Choice of γ

In all the methods described for training an invariant SVM, there is a parameter γ to choose, which is a trade-off between maximization of the margin and enforcement of the invariances (cf equation 4). One can set this parameter using a validation set or by gradient descent [2].

In order to have a functional which is scale invariant with respect to the size of $d\mathbf{x}_i$ (in other words, the parametrization of $\mathcal{L}_t$, instead of minimizing the functional (4), we actually minimize

$$(1-\gamma)\mathbf{w}^2 + \gamma \sum_{i=1}^{n} \frac{(\mathbf{w} \cdot d\mathbf{x}_i)^2}{S},$$

with

$$S = \sum_{i=1}^{n} d\mathbf{x}_i^2.$$

This equivalent to minimizing functional (4) where γ is replaced by

$$\gamma \longleftarrow \frac{\gamma}{S + \gamma(1-S)}.$$

5.2 Toy problem

The toy problem we considered is the following: the training data has been generated uniformly from $[-1,1]^2$. The true decision boundary is a circle centered at the origin:

$$f(\mathbf{x}) = \text{sign}(\mathbf{x}^2 - 0.7).$$

The a priori knowledge we want to encode in this toy problem is *local invariance under rotations*. Therefore, the output of the decision function on a given training point $\mathbf{x}_i$ and on its image $R(\mathbf{x}_i, \varepsilon)$ obtained by a small rotation should be as similar as possible. To each training point, we associate a tangent vector $d\mathbf{x}_i$ which is actually orthogonal to $\mathbf{x}_i$(see figure 1).

A training set of 30 points was generated and the experiments were repeated 100 times. A Gaussian kernel

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{||\mathbf{x}-\mathbf{y}||^2}{2\sigma^2}\right)$$

was chosen.

The results are summarized in figure 2 and the following observations can be made:

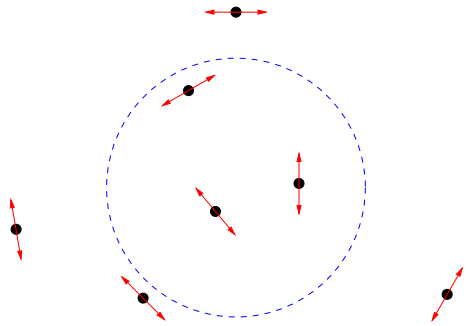


Figure 1: Toy problem: the true decision function is the circle. For each point we plot a tangent vector corresponding to a small rotation.

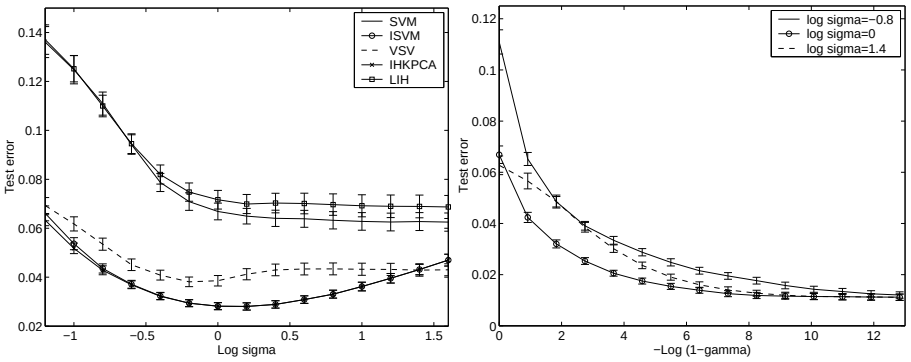


Figure 2: Left: test error for different learning algorithms plotted against the width of a RBF kernel and γ fixed to 0.9. Right: test error of $\mathrm{IH}_{\mathrm{KPCA}}$ across γ and for different values of σ. The test errors are averaged over the 100 splits and the error bars correspond to the standard deviation of the means.

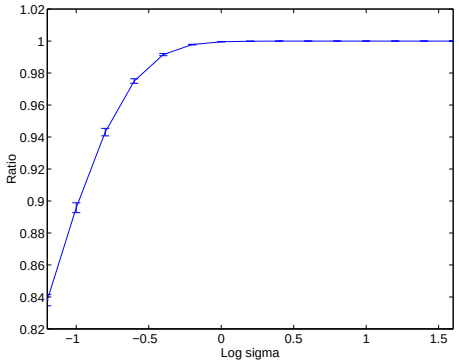


Figure 3: Approximation ratio (22) for different values of σ. When the ratio is near 1, there is almost no loss of information using $\mathrm{IH}_{\mathrm{KPCA}}$ and thus in this case ISVM and $\mathrm{IH}_{\mathrm{KPCA}}$ are identical.

- For each σ, adding virtual training examples reduces the test error.
- The ISVM and $\mathrm{IH}_{\mathrm{KPCA}}$ yield almost identical performances, especially for large values of σ. This can be explained by figure 3. The ratio (22) has been computed for different values of σ. For large values, this ratio is almost 1, which means that there is almost no loss of information by projecting $d\Phi(\mathbf{x}_i)$ on E (see section 4.3) and thus ISVM and $\mathrm{IH}_{\mathrm{KPCA}}$ produce the same decision boundary.
- The method LIH actually slightly impaired the performance. As pointed out above, this method has no guarantee to improve the performances. Actually, in this situation, we expected to get almost the same results as a standard SVM. Indeed, the covariance matrix of tangent vectors in input space is in this toy example is expected to be roughly a constant times the identity matrix:

$$\sum_{i=1}^{n} d\mathbf{x}_i d\mathbf{x}_i^\top \approx C\ I_2.$$

  Different values of γ have been tried for this method, but none of them led to an improvement (figure 4).

Some of the results presented in figure 2 are somehow surprising, but might be just particular to this problem:

- The test error of a normal SVM does not increase when σ has large values. We do not have here the traditional "bowl" shape.

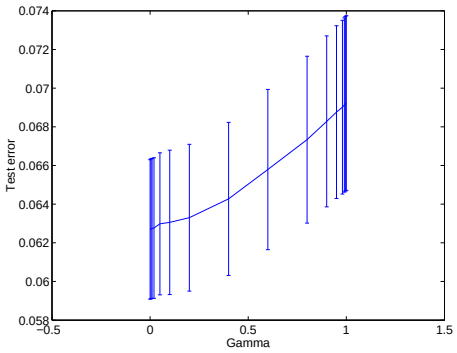


Figure 4: Test error versus γ for LIH. Trying to incorporate invariance using this method fails on this toy example.

- While the test error for SVM and VSV do not increase for large σ, the one of ISVM and IH_{KPCA} do. Maybe a larger value of γ (more invariance) should be applied in this case. This can be seen from figure 5, where the following quantity has been plotted

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\mathbf{w}\cdot d\Phi(\mathbf{x}_i))^2} \tag{23}$$

  However, the reason for a larger value of γ remains unclear. Note that equation (23) gives a rule of thumb on how to choose a good value for γ. Indeed, after training a reasonable value for (23) should be in the range [0.2 - 0.4]. It represents the average difference between the output of a training point and its transformed $\mathcal{L}_t\mathbf{x}_i$.

Some comments about the right hand-side of figure 2.

- The more invariances, the better (and it converges to the same value for different σ). However, it might be due to the nature of this toy problem.

- When comparing $\log\sigma = 1.4$ and $\log\sigma = 0$, one notices that the decrease in the test error does not have the same speed. This is actually the dual of the phenomenon observed on the left side of figure 2 : for a same value of gamma, the test error tends to increase, when σ is larger.

Finally table 1 summarizes the test error of the different algorithms (using optimal parameter settings).

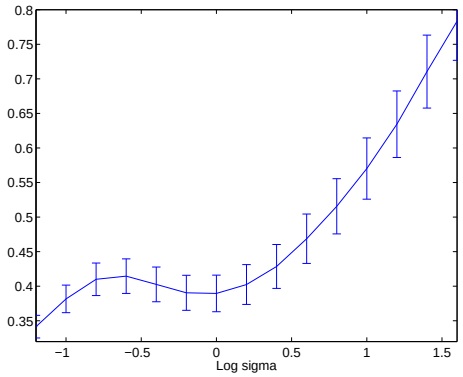


Figure 5: Value of the invariance enforcement part (eq 23) of the objective function across σ. A small value amounts to a better enforcement of the invariances.

SVM	VSV	LIH	ISVM	IH_{KPCA}
6.25	3.81	6.87	1.11	1.11

Table 1: Summary of the test error of the different learning algorithms

5.3 Handwritten digit recognition

As a real world experiment, we tried to incorporate invariances for a handwritten digit recognition task. The USPS dataset have been used extensively in the past for this purpose, especially in the SVM community. It consists of 7291 training and 2007 test examples.

According to [9], the best performance has been obtained for a polynomial kernel of degree 3,

$$K(\mathbf{x}, \mathbf{y}) = \left(\frac{\mathbf{x} \cdot \mathbf{y}}{256}\right)^3.$$

The factor 256, equaling the dimension of the data, has been included to avoid numerical instabilities. Since the choice of the base kernel is not our main concern here, all the results described in this section were performed using this kernel. The local transformations we considered are translations (horizontal and vertical). All the tangent vectors have been computed by a finite difference between the original digit and its 1-pixel translated(see figure 6).

Following the experimental protocol described in [11], we split the train-

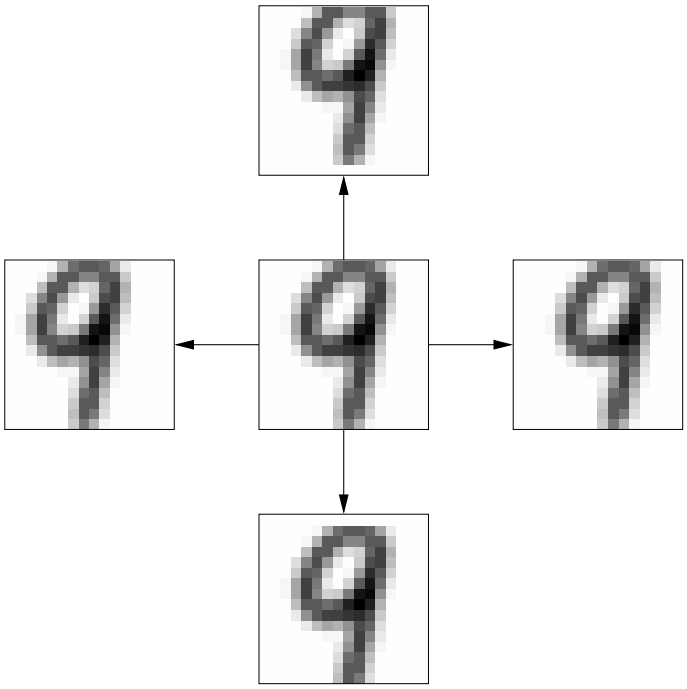
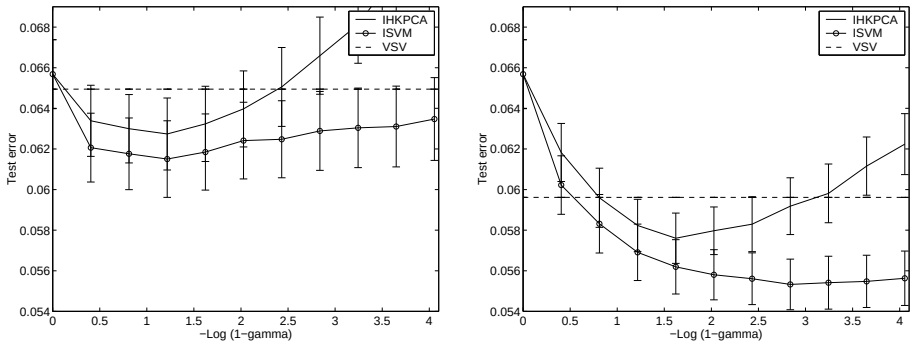


Figure 6: The original digit in the center and its translated from 1 pixel on the left, right, up and down



Vertical translation (to the top) Horizontal translation (to the right)

Figure 7: Comparison of ISVM, IH_{KPCA} and VSV on the USPS dataset. The left of the plot ($\gamma = 0$) corresponds to standard SVM whereas the right part of the plot ($\gamma \to 1$) means that a lot of emphasis is put on the enforcement of the constraints. The test errors are averaged over the 23 splits and the error bars correspond to the standard deviation of the means.

ing set into 23 subsets of 317 training examples after a random permutation of the training and test set. Also we concentrated on a binary classification problem, namely separating digits 0 to 4 against 5 to 9. The gain in performance should also be valid for the multiclass case.

Figure 7 compares ISVM, IH_{KPCA} and VSV for different values of γ. From those figures, it can be seen that the difference between ISVM (the original method) and IH_{KPCA} (the approximation) is much larger than in the toy example. The quality of the approximation can be estimated through the approximation ratio (22). It is 0.2 for the vertical translation and 0.33 for the horizontal one. Those number are far from being 1, which explains the difference in performance between the 2 methods (especially when $\gamma \to 1$). The difference to the toy example is probably due to the input dimensionality. In 2 dimensions, with an RBF kernel, the 30 examples of the toy problem "almost span" the whole feature space, whereas with 256 dimensions, this is no longer the case.

What is noteworthy in these experiments is that our proposed method is much better than the standard VSV. As explained in section 4.4, the reason for this might be that invariance is enforced around *all* training points and not only around support vectors. Note that what we call VSV here is a standard SVM with a *double size* training set containing the original data points and their translates.

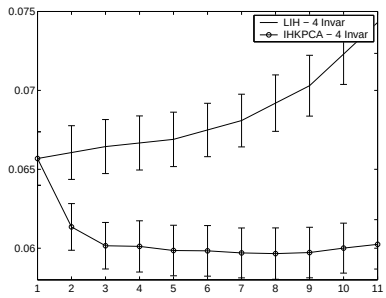


Figure 8: Results for 4 invariances and different values of γ. Left is standard SVM ($\gamma = 0$)

The horizontal invariance yields larger improvements than the vertical one. One of the reason might be that the digits in the USPS database are already centered vertically(see figure 6).

After having studied single invariances, we wanted to see what kind of gain one can expect with multiple invariances. For the digit recognition task, we thus considered the 4 invariances (one pixel in each direction).

We compared $\mathrm{IH}_{\mathrm{KPCA}}$ (which scales better than ISVM for multiple invariances, as mentioned at the beginning of section 4.3) with LIH in figure 8

There is again a significant improvement in performance while LIH fails.

6 Conclusion

We have extended a method for constructing invariant hyperplanes to the nonlinear case. We have shown results that are superior to the virtual SV method. The latter has recently broken the record on the NIST database which is the "gold standard" of handwritten digit benchmarks [5], therefore it appears promising to also try the new system on that task. For this propose, a large scale version of this method needs to be derived. The first idea we tried is to compute the kernel PCA map using only a subset of the training points. Encouraging results have been obtained on the 10-class USPS database (with the whole training set), but other methods are also currently under study.

References

[1] C. J. C. Burges. Geometry and invariance in kernel based methods. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*. MIT Press, 1999.

[2] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 2001. in press.

[3] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273 – 297, 1995.

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

[5] D. DeCoste and B. Schölkopf. Training invariant support vector machines. *Machine Learning*, 2001. In press.

[6] Todd K. Leen. From data distributions to regularization in invariant learning. In *Nips*, volume 7. The MIT Press, 1995.

[7] P. Niyogi, T. Poggio, and F. Girosi. Incorporating prior information in machine learning by creating virtual examples. *IEEE Proceedings on Intelligent Signal Processing*, 86(11):2196–2209, November 1998.

[8] John Platt. Probabilities for support vector machines. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, Cambridge, MA, 2000.

[9] B. Schölkopf, C. Burges, and V. Vapnik. Extracting support data for a given task. In U. M. Fayyad and R. Uthurusamy, editors, *First International Conference on Knowledge Discovery & Data Mining*. AAAI Press, 1995.

[10] B. Schölkopf, C. Burges, and V. Vapnik. Incorporating invariances in support vector learning machines. In *Artificial Neural Networks — ICANN'96*, volume 1112, pages 47–52, Berlin, 1996. Springer Lecture Notes in Computer Science.

[11] B. Schölkopf, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Generalization bounds via eigenvalues of the Gram matrix. Technical Report 99-035, NeuroColt, 1999.

[12] B. Schölkopf, P. Y. Simard, A. J. Smola, and V. N. Vapnik. Prior knowledge in support vector kernels. In MIT Press, editor, *NIPS*, volume 10, 1998.

[13] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1310, 1998.

[14] P. Simard, Y. LeCun, J. Denker, and B. Victorri. Transformation invariance in pattern recognition, tangent distance and tangent propagation. In G. Orr and K. Muller, editors, *Neural Networks: Tricks of the trade*. Springer, 1998.

[15] A. J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In P. Langley, editor, *Proceedings of the 17th International Conference on Machine Learning*, pages 911–918, San Francisco, 2000. Morgan Kaufman.

[16] K. Tsuda. Support vector classifier with asymmetric kernel function. In M. Verleysen, editor, *Proceedings of ESANN'99*, pages 183–188, 1999.

[17] V. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.

[18] 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. MIT Press, 2001.