Data Centering in Feature Space

Marina Meilă
Department of Statistics
University of Washington
Seattle, WA 98195-4322
mmp@stat.washington.edu

Abstract

This paper presents a family of methods for shifting the data in feature space to be used in conjunction with kernel machines. The shift can be performed using only kernel evaluations in input space. The methods are used to improve the numerical properties of kernel machines and to generate new families of string kernels. Experiments on real and artificial data show the beneficial effects of the centering methods and reveal further insights into the geometry of origin shifts in feature space.

1 Introduction

Support vector machines (SVMs for short) classify data by mapping it into a high (possibly infinite) dimensional *feature* space and constructing a maximum margin hyperplane to separate the classes in that space. Operations in the feature space are rendered independent of its dimension by what is commonly called now the "kernel trick", the use of an efficiently computable *kernel function* for the scalar product in feature space.

The SVM classifier is learned from data by means of the *Gram matrix* K consisting of the pairwise scalar products of the data points in feature space. If, in the feature space, the origin is far away from the convex hull of the data, then the elements of K have about the same value and, as a result, the matrix K is ill-conditioned. Figure 1 illustrates such a situation, showing that the performance of the resulting classifier degrades.

The present work sets out to correct this problem, by shifting the data such that the origin is located in the convex hull of the data. While this is almost trivial for a linear classifier, it is not so for non-linear SVMs where the data are mapped non-linearly into a high-dimensional space that is not explicitly represented. Thus, the challenge is to perform the shift and to compute the resulting SVM using only "allowed" operations, that is applying the kernel to points in the input space.

This paper presents kernel tricks that allow one to perform a large variety of origin translations in the feature space of a non-linear kernel machine. In its simplest version, this method of data centering in feature space has been in use for a long time (see e.g [5]). Here we show that one can formulate data centering in the form of criterion to be optimized by a translation in feature space, and that this translation can be performed entirely by "allowed" kernel operations.

There has been previous work on adapting kernels to the data, notably that of [1, 4, 6]. The

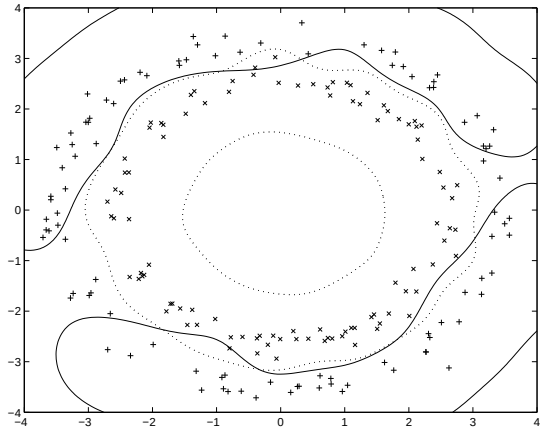

Figure 1: Original SVM classifier (dotted line) and classifier obtained after translating the origin in feature space by ≈ 500 units (full line). The two classes are represented by "$+$" and "$\times$" respectively. The kernel used is the RBF kernel.

current idea differs from the above in that it does not affect the geometry of the problem, it only affects its translation into a numerical task.

Here we use origin translations primarily for data centering, in order to improve the numerical properties of the kernel machine in the training phase. Experiments on real and artificial data show that these operations are indeed effective.

The second goal is to obtain new composite kernels. Composite kernels, typically used for string data, are formed from other, simpler, kernels, called *element kernels*. Translation in the feature space of each of the element kernels followed by composition does not in general preserve the feature space geometry of the composite kernel. Therefore the "shifted" classifier will in general differ from the original, unshifted, classifier.

An origin placed far away from the data is not the only cause of numerical problems in SVMs. Another common problem is the inappropriate choice of kernel width (typical for RBF kernels) which can result into overfitting/underfitting [12]. A related problem known as "ridge effect" occurs in string kernels [14]: the data are almost orthogonal to each other in feature space. This can lead to both numerical problems and overfitting. While a translation of the data may alleviate the numerical problems, the geometry of the classifier cannot be influenced by origin shift in feature space. Therefore we will not address these problems in the current paper.

We start with a short introduction to support vector machines, then we introduce a simple method of data centering in section 3 and explain how to perform classification with shifted support vectors in section 4. We present a general method of shifting in feature space in order to optimize some given centering criterion in section 5. Section 6 presents another method for data translation in feature space, that is not related to an optimization problem, and section 7 presents some preliminary ideas about origin shifts in string kernels. Experiments are presented in section 8 and the discussion in section 9 concludes the paper.

2 Support Vector Machines

We start by briefly introducing the reader to SVMs; for more details the reader is invited to consult [12]. In a classification task, we are given a set of n *data points* $\{\tilde{x}_1, \tilde{x}_2, \ldots \tilde{x}_n\}$ elements of an *input space* $\tilde{\mathcal{X}}$. Each data point $\tilde{x}_i$ is labeled by $y_i \in \{\pm 1\}$. The task is to

use the data and labels to construct a classifier, i.e. to find a function f that predicts the label y of a new point $\tilde{x}$.

The SVM constructs a classifier in two steps: First, the data are mapped into a space called the *feature space*. We assume that the data is *linearly separable* in that space, meaning that a hyperplane that separates the two classes exists. In the second step, the SVM finds the hyperplane that optimally separates the classes. Once a new input $\tilde{x}$ is presented, the hyperplane is used to classify its mapping x into the feature space. We now further describe each of the steps.

Input space and feature space. We call the space of the original data the *input space*. For all but the easiest problems, the data are not linearly separable in the input space. The goal of the mapping is to obtain a data set that is linearly separable or almost. Another reason for the mapping is to enable us to construct non-linear classifiers, as it will be shown below.

We map the data to feature space by a function ϕ and we denote[1]

$$x \;=\; \phi(\tilde{x}) \quad \text{for all } \tilde{x} \in \tilde{\mathcal{X}} \tag{1}$$

The feature space is a Hilbert space whose dimension d is commonly much larger than n the number of data points and it can be infinity (e.g in the *RBF* kernel [12]). The input space need not be a Hilbert space, it can be any set. The trick that makes SVMs work is never to explicitly represent points in feature space or ϕ itself. The SVM only makes use of scalar products of points in feature space, which are computed by the function

$$K(\tilde{x},\tilde{z}) \;\equiv < \phi(\tilde{x}), \phi(\tilde{z}) > = < x, z > \tag{2}$$

called the *kernel* associated with ϕ. It is assumed that $K(\tilde{x},\tilde{z})$ can be computed efficiently for any pair of inputs. To be represent a scalar product, a kernel K is subject to the Mercer condition [12], namely that it induces a positive definite integral operator on $\tilde{\mathcal{X}}$.

Finding the optimal hyperplane. The separating hyperplane is described by the equation $< w, x > + b = 0$, with w a vector in feature space and b a real number. The maximum margin separating hyperplane is found by solving the following optimization problem in the variables $\alpha_i,\ i = 1, 2, \ldots n$.

$$\max_{x} \mathcal{V}(\alpha) \quad \text{s.t. } \alpha_i \geq 0 \ \text{ and } \sum_{i=1}^{n} \alpha_i y_i \;=\; 0 \tag{3}$$

with

$$\mathcal{V}(\alpha) \;=\; \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j < x_i, x_j > \tag{4}$$

The optimal hyperplane is given by

$$w \;=\; \sum_{i=1}^{n} \alpha_i y_i x_i \tag{5}$$

$$b \;=\; y_i - \sum_{j=1}^{n} \alpha_j y_j < x_i, x_j > \quad \text{for some } i \text{ s.t } \alpha_i \neq 0 \tag{6}$$

[1]Note that here we depart from the standard notation in which x is a point in input space and its image is $\tilde{x}$. Since most of our calculations will be carried on in feature space the change will keep the notation simple.

Note that although the optimization problem involves the data images in feature space, the data points enter $\mathcal{V}$ only via the pairwise scalar products $< x_i, x_j >= K(\tilde{x}_i, \tilde{x}_j)$ which can be computed using the kernel function. This is the celebrated "kernel trick" of support vector machines. The matrix

$$K \;=\; [K(\tilde{x}_i, \tilde{x}_j)]_{i,j=1,\ldots n} \tag{7}$$

is called the *Gram matrix*[2]. Throughout the rest of the paper, we assume that a function SVM-SOLVER is given. The function SVM-SOLVER takes as input a Gram matrix K and a set of labels y returns the parameters $b,\ \alpha_i,\ i = 1, \ldots n$ of an SVM.

Classifying with SVMs. When a new point $\tilde{x}$ is presented for classification, its label is computed by

$$y \;=\; f(\tilde{x}) \;=\; \text{sign}(< w, x > +b) \tag{8}$$

which amounts to computing

$$f(\tilde{x}) \;=\; \text{sign}\left[\sum_i \alpha_i y_i < x, x_i > +b\right] \tag{9}$$

$$\;=\; \text{sign}\left[\sum_i \alpha_i y_i K(\tilde{x}, \tilde{x}_i) + b\right] \tag{10}$$

Although $w,\ x,\ x_i$ are vectors in feature space, the function f can be computed explicitly because it uses only kernel computations. If the kernel K is a non-linear function in each argument, then the resulting classifier f is a non-linear classifier.

The theory of SVMs [12] predicts that a (usually) large proportion of the coefficients α_i will be 0. The data points for which $\alpha_i > 0$ are called *support vectors*. Because it uses only a subset of the points in constructing the classifier, the SVM solution is called *sparse*.

SVM extensions For the sake of simplicity, we have presented here only the most basic version of SVM. Many other version exist that build upon the basics, some meant to deal with the case of non separable data (C-SVM [3], ν-SVM [11]), others adapted for classification from positive examples only [9], and others meant to deal with more than two classes [10]. All SVM versions cited here have in common the use of the Gram matrix as the only vehicle by which the data enter the SVM training. Therefore, the methods for data translation we present here should apply to them as well.

3 A simple centering method

As shown in section 1, if in feature space the origin lies far away from the data, then the matrix K will have almost equal elements and will be ill-conditioned.

How can we establish if, in the high-dimensional feature space, the origin lies "between" the classes or far-away from them? One way is to look at $K(\tilde{x}_i, \tilde{x}_j)$ when data points i, j belong to different classes. If this scalar product is negative, it means that the points are seen from the origin under an obtuse angle, in other words the origin lies approximately between the two points. If we denote by K_a the kernel representing the scalar product with the origin shifted in a

$$K_a(\tilde{x}, \tilde{z}) \;\stackrel{\Delta}{=}\; < x, z >_a \;=\; < x - a, z - a > \tag{11}$$

[2]We shall use the same notation K for both the Gram matrix and the kernel; the distinction will be evident from the context.

then we can define the optimal position of the origin to be the location a that minimizes

$$J(a) \;=\; \sum_{y_i=1} \sum_{y_j=-1} K_a(\tilde{x}_i, \tilde{x}_j) \tag{12}$$

Using

$$< x, z >_a = < x, z > - < x, a > - < z, a > + < a, a > \tag{13}$$

and letting $n^+(n^-)$ denote the number of data points with $+1(-1)$ labels, we can rewrite $J(a)$ as

$$J(a) \;=\; \sum_{y_i=1} \sum_{y_j=-1} < x_i, x_j > - \sum_{y_i=1} n^- < x_i, a > - \sum_{y_i=-1} n^+ < x_i, a > + n^+ n^- < a, a > \tag{14}$$

This is a quadratic criterion in a whose minimum is at

$$a \;=\; \frac{1}{2n^+} \sum_{y_i=1} x_i + \frac{1}{2n^-} \sum_{y_i=-1} x_i \tag{15}$$

Thus the optimal a according to (12) is positioned halfway between the centers of gravity of the two classes in feature space. The kernel K_a for this position of the origin may not be computable in closed form; nevertheless, we can obtain the Gram matrix K_a necessary to solve the SVM optimization problem using only calls to the original kernel K. This is a consequence of the fact that a is a linear combination of data points in feature space. Denote

$$\gamma_i \;=\; \begin{cases} (2n^+)^{-1}, & y_i = 1 \\ (2n^-)^{-1}, & y_i = -1 \end{cases} \qquad \overline{\gamma} \;=\; [\gamma_1 \;\ldots\; \gamma_n]^T \qquad \Gamma = [\overline{\gamma} \ldots \overline{\gamma}] \tag{16}$$

Then the "centered" Gram matrix is given by

$$K_a \;\equiv\; [K_a(\tilde{x}_i, \tilde{x}_j)]_{ij} \;=\; K - \Gamma^T K - K\Gamma + \Gamma^T K \Gamma \tag{17}$$

Having obtained K_a, a call to SVM-SOLVER(K_a, y) will output the parameters $b,\ \alpha_i,\ i = 1, \ldots n$ of an SVM.

The optimal a obtained by the centering as in (15) may not belong to the data manifold in feature space, hence it is generally not representable by a point $\tilde{a}$ in input space. This fact does not preclude us from performing the centering by (17) as we just showed. Nor does it preclude from solving the SVM optimization (3) with the new kernel K_a. In the next section we show that classifying new data points is also tractable in this setup.

The criterion (12) and its corresponding data centering operation solution can be generalized to problems that involve more than two classes. Let the set of classes be C (finite) . We define J as

$$J(a) \;=\; \sum_{c \in C} \sum_{y_i = c} \sum_{y_j \neq c} K_a(\tilde{x}_i, \tilde{x}_j) \tag{18}$$

Denoting by n_c the number of data points in class $c \in C$, we obtain

$$a \;=\; \frac{\sum_c (n - n_c) \sum_{y_i = c} x_i}{\sum_c n_c (n - n_c)} \tag{19}$$

and

$$\gamma_i \;=\; \frac{n - n_c}{2 \sum_{c > c'} n_c n_{c'}} \quad \text{if } y_i = c \qquad \text{for all } i = 1, \ldots n \tag{20}$$

In other words, the center a is a convex combination of the centers of mass of each of the classes. The centering operation is performed according to (17) with coefficients γ_i from (20).

Both criteria guarantee that the origin is contained within the convex hull of the data after the shift. They are very similar to the data centering method of [5] which moves the origin at the center of gravity of the data. Henceforth we call this method the *standard* centering method. In terms of the γ_i coefficients above, moving the origin to the center of gravity of all the data amounts to setting $\gamma_i = \frac{1}{n}$.

4 SVM classification with centered data

We explain now how to perform classification with centered data. Denote by $b,\ \alpha_i$ the output of SVM-SOLVER$(K_a,\ y)$. According to equation (9), classifying a new data point $\tilde{x}$ is done by

$$f_a(\tilde{x}) \ = \ \text{sign}\left[\sum_i \alpha_i y_i K_a(\tilde{x}, \tilde{x}_i) + b\right] \tag{21}$$

Here, of course, we don't have $K_a(\tilde{x}, \tilde{x}_i)$ in closed form. This apparent obstacle can be overcome by rewriting the expression inside the brackets in the following way

$$\begin{aligned}
\sum_i \alpha_i y_i K_a(\tilde{x}, \tilde{x}_i) + b \ &= \ \sum_i \alpha_i y_i < x, x_i >_a + b && (22)\\
&= \ \sum_i \alpha_i y_i [< x, x_i > - < a, x_i > - < a, x > + < a, a >] + b \\
&= \ \sum_i \alpha_i y_i < x, x_i > - \sum_i \alpha_i y_i < a, x_i > \\
&\quad + b + \underbrace{\sum_i \alpha_i y_i}_{0} (- < a, x > + < a, a >) && (23)\\
&= \ \sum_i \alpha_i y_i K(\tilde{x}, \tilde{x}_i) + b - \sum_i \alpha_i y_i < a, x_i > && (24)
\end{aligned}$$

Denote

$$h_i \ = < a, x_i > \qquad \text{for all } \ i = 1, \ldots, n, \qquad \overline{h} = [h_1 \ h_2 \ \ldots h_n]^T \tag{25}$$

The values of h_i are obtained by

$$h_i \ = < a, x_i > = \ \sum_{j=1}^{n} \gamma_j < x_j, x_i > = \ \sum_{j=1}^{n} \gamma_j K(\tilde{x}_j, \tilde{x}_i) \tag{26}$$

Hence, a shift in the origin amounts to a constant correction term in the classifier, having the value

$$\Delta b \ = \ \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i y_i \gamma_j K(\tilde{x}_i, \tilde{x}_j) \tag{27}$$

The above result can be explained geometrically in the following way, illustrated in figure 2. In feature space, classification is equivalent to projecting the vector $\vec{Ox}$ from the origin to the image of x the new point onto the normal w to the separating hyperplane , then comparing it to the (signed) distance $-b$ from O to the hyperplane. When the origin is move to a, the vector to be projected is $\vec{ax} = \vec{Ox} - \vec{Oa}$ and the distance to the hyperplane becomes $-b + Oa'$, where Oa' is the projection of $\vec{Oa}$ onto w. The normal to the separating hyperplane is invariant to translation so w is the same before and after the shift.

This result is a direct consequence of the known fact that the maximum margin hyperplane is *invariant to translations in feature space*. In other words, after shifting the origin we

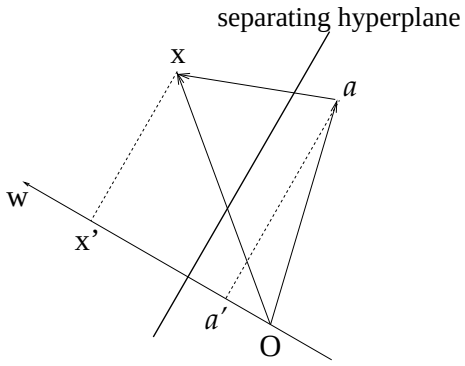

Figure 2: The geometry of origin shift and its effect on b. O, a, x, are respectively the old origin, the new origin and a data point; x', a' are the projection of x, a on the normal w to the separating hyperplane (this direction is invariant to translation). The classification threshold b is the distance between the origin (old or new) to the hyperplane, and the change Δb due to the change in origin equals Oa' the projection of Oa on w. The correction Δb accounts for the fact that the origin was shifted during training while the new data are classified with the original, unshifted, kernel K.

obtain the same classifier as we would from the original kernel, numeric stability issues aside.

The computation required to obtain Δb is proportional to the number of non-zero α_i's times the total number of data points. Hence, if the SVM solution is sparse, the computation of Δb is subquadratic. Note that Δb is in general non-zero for $\gamma_i = 1/n$, i.e in the case of the "standard" centering method. This means that after centering the Gram matrix by the standard method, one needs to make a correction to the final classifier. This fact is sometimes overlooked in the literature [5]. The correction Δb is proportional to the norm of a, implying that if the origin is initially far away from the data, the value of Δb can be quite large. The effect of ignoring the correction Δb is illustrated by figure 8 in section 8.5.

The SVM classification with centered data can then be summarized as follows:

1. **Preprocessing:**
   (a) Compute the Gram matrix K using definition (7)
   (b) Compute the centered Gram matrix K_a by (11)
   (c) Compute the scalar products h_i using (26)
2. **Training:** Call SVM-SOLVER(K_a, y). This outputs $b,\ \alpha_i,\ i = 1, \ldots n$.
3. **Postprocessing:** $b \leftarrow b - \Delta b$
4. **Classification:** Classify any new data points just as for an unmodified SVM classifier, i.e using $b,\ \alpha_i,\ i = 1, \ldots n$ and the support vectors according to formula (9).

In conclusion, centering in feature space only adds extra work in the SVM training phase, being essentially transparent in the classification phase.

5 A general centering method

In the above we have shown how to perform an origin shift that optimizes the criterion (12). Now we proceed to generalize the method to optimizing any criterion that is a function of the Gram matrix only. Examples of such criteria are

- Minimizing the sum of the cosines between all pairs of examples in different classes
$$\sum_{y_i=1} \sum_{y_j=-1} \cos \angle(x_i, x_j) \tag{28}$$
Since the cosine between two points in feature space is a valid kernel (which places all the data points on the unit sphere), this criterion is equivalent to simple centering in a different feature space. Note however that the relationship between the two feature spaces is not straightforward.
- Maximizing the *kernel alignment* of [6] defined as
$$A(K_a) \;=\; \frac{y^T K_a y}{n|K_a|_F} \tag{29}$$
with $|K_a|_F$ being the Frobenius norm [6] of K_a.
- Another apparently useful criterion is maximizing the "unnormalized" alignment
$$y^T K y \tag{30}$$
equivalent to maximizing the sum of K_{ij} within each class minus the sum of K_{ij} between classes. At a closer inspection however, it can be seen that, unless $n^+ = n^- = n/2$ this criterion has a maximum for $a \;\rightarrow\; \infty$ in any direction. Therefore we do not recommend its usage.

Assume that our centering criterion is

$$\max_a J(K_a) \tag{31}$$

We assume that J is a suitably smooth function of elements the Gram matrix, and in particular that its gradient is well defined. We show how to optimize J by gradient ascent.

For this purpose, we first need to compute the gradient of J with respect to a.

$$\nabla_a J(K_a) \;=\; \sum_{ij} \frac{\partial J}{\partial K_a(\tilde{x}_i, \tilde{x}_j)} \nabla_a K_a(\tilde{x}_i, \tilde{x}_j) \tag{32}$$

From equation (11)

$$\nabla_a K_a(\tilde{x}_i, \tilde{x}_j) \;=\; -x_i - x_j + 2a \tag{33}$$

The gradient is a vector in feature space and, by combining the two formulas above, one can easily see that the gradient is a linear combination of a and the data vectors. Taking a step in the direction $\nabla_a J$ with step size η means

$$a \;\leftarrow\; a + \eta \nabla_a J \tag{34}$$

The step $\delta_a = \eta \nabla_a J$ is itself a linear combination of a and the data vectors, hence

$$\delta_a \;=\; \gamma_0 a + \sum_i \gamma_i x_i \tag{35}$$

with

$$\gamma_0 \;=\; 2\eta \sum_{ij} \frac{\partial J}{\partial K_a(\tilde{x}_i, \tilde{x}_j)} \tag{36}$$

$$\gamma_i \;=\; -2\eta \sum_{j=1}^{n} \frac{\partial J}{\partial K_a(\tilde{x}_i, \tilde{x}_j)} \qquad \text{for } \; i = 1, \ldots n \tag{37}$$

All the coefficients γ above can be easily computed using only kernel evaluations.

At each step of the iteration, we update: the Gram matrix K_a, the scalar products $h_i =< a, x_i >$ for $i = 1, \ldots n$ and the square length of a, $h_0 =< a, a >$.

$$< x_i, x_j >_{a+\delta_a} = < x_i, x_j >_a - < x_i, \delta_a > - < x_j, \delta_a > + 2 < \delta_a, a > + < \delta_a, \delta_a > \quad (38)$$

$$< \delta_a, x_i > = \gamma_0 < x_i, a > + \sum_{i'} \gamma_{i'} < x_i, x_{i'} > \quad (39)$$

$$< \delta_a, a > = \gamma_0 h_0 + \sum_i \gamma_i < a, x_i > \quad (40)$$

$$< \delta_a, \delta_a > = \sum_{i'j'} \gamma_{i'} \gamma_{j'} < x_{i'}, x_{j'} > + 2\gamma_0 \sum_{i'} \gamma_{i'} < x_{i'}, a > + \gamma_0^2 < a, a > \quad (41)$$

$$< a + \delta_a, x_i > = < a, x_i > + < \delta_a, x_i > \quad (42)$$

$$< a + \delta_a, a + \delta_a > = < a, a > + 2 < a, \delta_a > + < \delta_a, \delta_a > \quad (43)$$

If we denote $\overline{\gamma} = [\gamma_1 \ \ldots \ \gamma_n]^T$, $\Gamma = [\overline{\gamma} \ \overline{\gamma} \ \ldots \ \overline{\gamma}]$ we can summarize the gradient ascent algorithm as follows

1. Initialize $K_a = K$, $\overline{h} = 0$, $h_0 = 0$.
2. Compute γ_i for $i = 0, \ldots n$ by (36, 37)
3. Update K_a, $\overline{h}$ and h_0 by (38-43)
4. Go to step 2 until convergence

From the computational point of view, each gradient step requires order n^2 computations: order n^2 derivative evaluations in step 2 and order n^2 update operations in step 3. This is of the same order of growth with one whole evaluation of the Gram matrix and affects only the training phase of the SVM classification. In our experiments we found that the gradient algorithm converges reasonably fast, in 50 iterations or less.

For large data sets, evaluating the whole K matrix is prohibitive and state-of-the-art SVM implementations evaluate only a subset of rows of K. In that case, the centering algorithms presented here would be prohibitive as well. We can easily fix this problem by using only a subsample of the data for centering, in a way similar to [6, 16]. For the simple centering method, we would sample e.g. $n'/2$ data points from each class and represent a as the arithmetic mean of the subsample. This would still ensure that the new position of the origin falls inside the convex hull of the data, but the extra amount of computation per row of K_a will be of order n'^2.

We can also use sampling to reduce the computational complexity of the general centering method. In this case the solution is to redefine the optimality criterion J to involve only a submatrix of K_a, depending on a subset of $n' << n$ points. While the solution may not work for any criterion, it is a reasonable approximation in the case of e.g optimizing the kernel alignment. For this case, [13] shows that the sampled alignment is concentrated around the true value.

6 An arbitrary shift in feature space

The previous sections showed how to obtain K_a and the SVM classifier if the new origin a is chosen so as to optimize some function $J(K_a)$. Now we show a kernel trick that would have the same effect when a is defined "explicitly", i.e by its coordinates in a basis of the feature space.

Let us define the problem more precisely. From the previous sections, we know that what we need to obtain K_a and the shift in b that defines the "shifted" SVM classifier are the values $h_i =< a, x_i >$ for $i = 1, \ldots n$ and $h_0 =< a, a >$. Assume for a moment that the dimension of the feature space is $d \leq n$. This is often the case for a polynomial kernel of low degree. Define a basis for $\mathcal{X}$ by the orthonormal $d \times d$ matrix V whose columns form the basis vectors. Then, a is completely defined by its coordinates w.r.t V:

$$a_j \ =< a, V_{:j} > \tag{44}$$

where $V_{:j}$ denotes the j-th basis vector. In matrix notation

$$a \ = \ V\overline{a} \tag{45}$$

where $\overline{a} \ = \ [a_1 \ldots a_d]^T$ is a's representation in the basis V. Similarly, the data can be represented in the basis V by

$$X \ = \ VB \quad \text{for} \quad X = [x_1 \ \ldots \ x_n] \tag{46}$$

with B a $d \times n$ coordinate matrix. It follows that

$$\overline{h} \ = \ X^T a \ = \ B^T V^T V \overline{a} \ = \ B\overline{a} \tag{47}$$

and

$$< a, a > = \ \sum_{j=1}^{d} a_j^2 \tag{48}$$

We have now everything we need to compute the shifted Gram matrix.

Let us now examine the more interesting case $d > n$ which includes the case $d = \infty$. In this case one cannot require a complete representation of a. We will show that $n+1$ numbers are sufficient to define a for our purpose. We assume again that we have an orthonormal basis in feature space. Denote by V the $d \times n$ matrix whose columns are the first n basis vectors[3]. W.l.o.g we assume that these n vectors span the data. (This can always be achieved by a simple rotation.) Denote by a_X and $a_\perp$ respectively the projections of a on the space spanned by V and on its orthogonal complement. Then,

$$< a, a > \ = \ < a_X, a_X > + < a_\perp, a_\perp > \tag{49}$$

$$< a, x_i > \ = \ < a_X, x_i > \quad \text{for all} \ \ i = 1, \ldots n \tag{50}$$

Hence, for our purpose, it suffices to know the projections of a on the vectors in V and the length of $a_\perp$, denoted $|a_\perp|$. Assume a_j define as above, with j ranging now from 1 to n and X represented again by (46). Then

$$\overline{h} \ = \ Xa \ = \ B^T \overline{a} \tag{51}$$

There is an easy way to obtain B, by noting that

$$XX^T \ = \ K \ = \ B^T V^T V B \ = \ B^T B \tag{52}$$

Therefore, $B \ = \ \sqrt{K}$ is a square root of K. This method was used in the simulations performed in section 8.

Finally, h_0 is obtained by

$$h_0 \ =< a, a > = \ \sum_{j=1}^{n} a_j^2 + |a_\perp|^2 \tag{53}$$

[3]As it will become evident shortly, the computations we perform here are symbolic, so working with infinite dimensional vectors should not be an obstacle.

7 Obtaining new string kernels

In this section we turn to using data centering in feature space to create new kernels. String kernels are kernels used for input data that are strings over a finite alphabet S. Some common application domains of string kernels are: text documents, where the input data are strings of letters, spaces and punctuation (i.e documents), genetics, where the data are DNA sequences over the four letter alphabet $S = \{A, C, G, T\}$, and molecular biology where the data are proteins over a 20 letter alphabet of amino-acids. The works of [7, 14] have pioneered the study of kernels for such data. Most string kernels are *composite* kernels, meaning that they are obtained from simple kernels called *element* kernels by compositions that are guaranteed to produce a valid kernel as result. These techniques are presented for example in [7, 14]. The element kernels are typically defined over sequences of length 1. The simplest element kernel is the "equality" kernel, defined over the symbols of the alphabet by

$$k(s, s') \;=\; \delta_{ss'} \qquad \text{for all } s, s' \in S \tag{54}$$

where $\delta_{ss'}$ is the Kronecker symbol.

Here, we illustrate the effect of the simple centering described in section 3 on the above element kernel. Assume that we have n examples consisting of single symbols labeled ± 1 according to their class. In practice, these could represent the values of the string data at one fixed location. We introduce the notation n_s^+ (n_s^-) to denote the number of occurrences of symbol $s \in S$ in class $+1$, (-1) respectively. Let n^+, (n^-) denote, as before, the total number of examples in each class. We denote by ν_s the average frequency of symbol s in the training set.

$$\nu_s \;=\; \frac{1}{2}\left(\frac{n_s^+}{n^+} + \frac{n_s^-}{n^-}\right) \tag{55}$$

Then, the centered element kernel, obtained after some straightforward calculations, is

$$k_a(s, s') \;=\; \delta_{ss'} - \nu_s - \nu_{s'} + \sum_{t \in S} \nu_t^2 \tag{56}$$

The centering makes $k_a(s, s)$ and $k_a(s, s')$ decrease with ν_s, giving thus larger weight to rarely seen symbols. One can envisage replacing the sample frequencies with different frequency estimates to a similar effect. Another effect, whose importance will be discussed shortly is the fact that centering the kernel makes it occasionally take negative values.

By centering the element kernels before composition, one obtains a new composite kernel and SVM classifier. In our experiments (see section 8.4) we use the Gap Penalty ANOVA kernel (GPAK) of [15]. This kernel is defined as follows: Let $\tilde{x} = (\tilde{x}^1, \ldots, \tilde{x}^m)$, $\tilde{z} = (\tilde{z}^1, \ldots, \tilde{z}^m)$ two strings of length m and let $\mathbf{i} = (i_1, \ldots, i_p)$ with $1 \leq i_1 < \ldots i_p \leq m$ a *tuple* of length p defining a (possibly noncontiguous) substring in each of $\tilde{x}, \tilde{z}$. Then, the GPAK of order p is defined as

$$K(\tilde{x}, \tilde{z}) \;=\; \sum_{\mathbf{i}:|\mathbf{i}|=p} w(\mathbf{i}) \prod_{j \in \mathbf{i}} k(\tilde{x}^j, \tilde{z}^j) \tag{57}$$

where $w(\mathbf{i})$ is a positive weight that depends on the number n_g and sum $l_g = i_p - i_1 - p + 1$ of gaps in $\mathbf{i}$.

$$w(\mathbf{i}) \;=\; \lambda^{n_g} \beta^{l_g}, \quad 0 \leq \lambda, \beta \leq 1 \tag{58}$$

This kernel can be computed by a dynamic programming type algorithm in time proportional to pm [15].

In (57), k represents the element kernel. The composition rule guarantees [7] that the composite kernel K is a valid kernel whenever k is a valid kernel, positive or not. However, a K constructed from element kernels with negative values does not make sense intuitively

(two negative values, multiplied, will make a positive overall contribution to K). In order to avoid such situations, we exponentiate the element kernels (a legal operation [7]) after centering. The process is summarized below

equality kernel		centered kernel		exponentiation		GPAK
k	$\longrightarrow$	k_a	$\longrightarrow$	$k_e = \exp(k_a/\sigma^2)$	$\longrightarrow$	K

8 Experiments

In the following experiments we used the SVMLIB [2] source code, that we modified in order to accept a user-defined Gram matrix.

8.1 Shifting and re-centering in feature space

This set of experiments aims to show that (1) drastic origin shifts in feature space harm the performance of an SVM classifier and (2), that the simple centering algorithm is able to restore the effects of the shift.

First, we generated data normally distributed around two concentric circles as in figure 1 and computed its Gram matrix K. Then we shifted the data in a random direction in feature space by a predetermined distance $|a|$ and computed the "shifted" Gram matrix K_s. We then centered the shifted data by the simple centering method described in section 3 and computed the "centered" Gram matrix K_c. Finally we trained an SVM using each of the three Gram matrices and evaluated it on test data from the same distribution.

The experiment was repeated 10 times with different samples and shift directions for every value of $|a|$. We used the degree 2 polynomial kernel and the RBF kernel. The training set size was 300 and the test set size was 200 in all cases. The results are shown in figure 3.

The second set of experiments was similar to the first, except that now we used real data sets from the UCI repository. The data set descriptions and SVM parameters are given in table 1. The shift length was 1000 for all data sets. For each data set, the experiment was run 10 times with different randomly sampled training sets. The results are shown in figure 4.

From the two experiments we see first that a large shift is detrimental to classification performance. The re-centered and original classifiers are practically identical for all the artificial data experiments and for all but one of the real data sets (wdbc). This shows that re-centering has indeed a restorative effect on data placed far away from the origin in feature space. The figures also show the effect on shifting and re-centering on kernel alignment: the alignment of the shifted kernel is practically 0 for the artificial data and drastically reduced for the real data. Re-centering brings alignment back to near or above the original values. The number of support vectors, an indirect indicator of generalization performance, grows with the size of the shift in the polynomial kernel and for all but the largest shift in the RBF kernel, but drops elsewhere. The drop is an artifact of the SVM-SOLVER software that was used in the experiments for very ill-posed problems (note that a shift of size 1000 is extremely large in the case of the RBF kernel).

8.2 Comparison with the standard centering method

Here we compared the centering method described in section 3 with shifting the origin in the center of weight of the data. To maximize the difference between the two methods, in these experiments the number of examples in one class was 20 times larger than the number of examples in the other class. Testing was done on data generated from the same distribution as the training data. We used only the polynomial kernel. The experimental

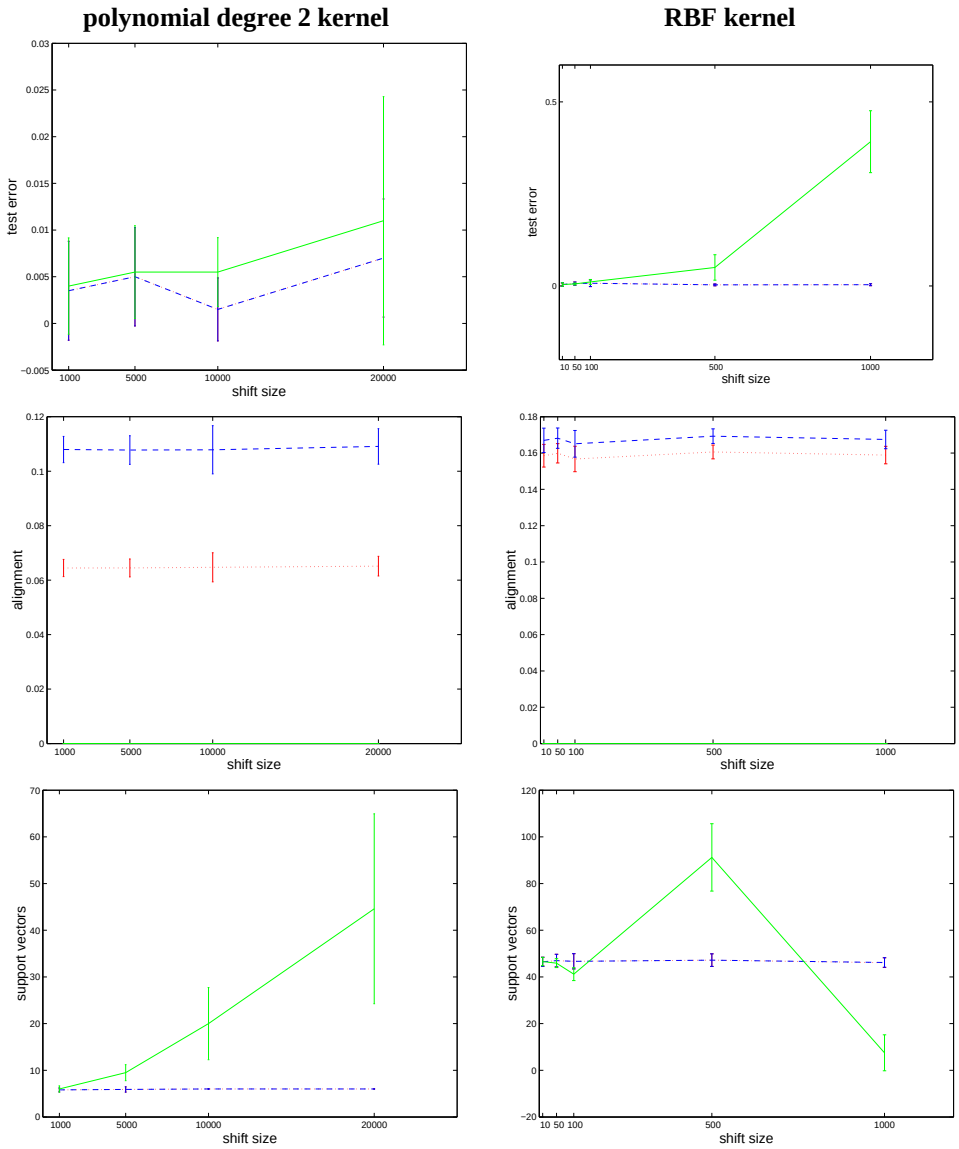

Figure 3: SVM classification with original (dotted line), shifted (full line) and re-centered (dashed line) data for different values of the shift length $|a|$. The original data are generated from the distribution shown in figure 1 (two concentric circles). These data are shifted in a random direction by an amount $|a|$ in feature space to obtain the shifted data. The shifted data are then re-centered as in section 3 to obtain the re-centered data. The figures depict the test error, kernel alignment and number of support vectors for the resulting SVMs, in the case of the polynomial degree 2 kernel (left) and of the RBF kernel (right). Results are averaged over 10 randomly sampled training sets of size 300. The original and centered SVMs are identical in all cases. The alignment of the shifted kernel is practically 0.

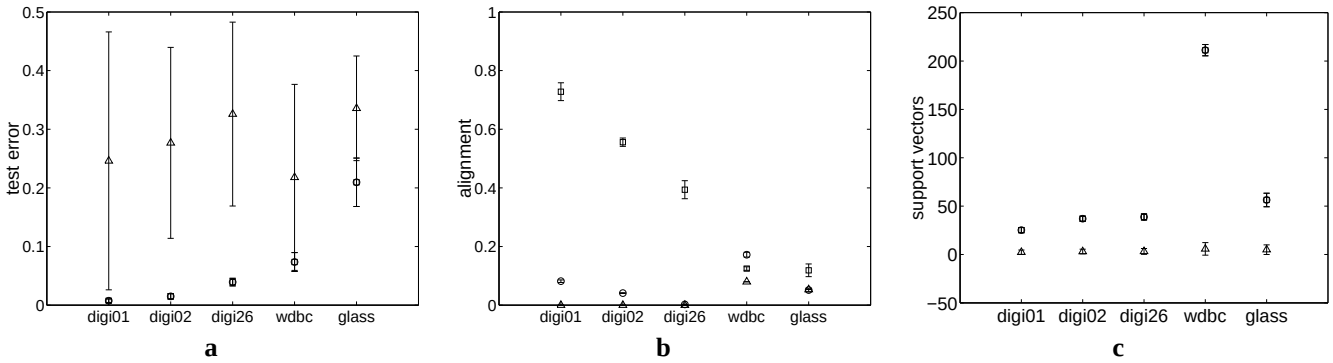

Figure 4: Shifting and re-centering on real data sets: **(a)** test error, **(b)** alignment, **(c)** number support vectors. Circles represent original data, triangles – shifted data, squares – re-centered data. The data sets are described in table 1. Each experiment was repeated 10 times with random direction shifts. The shift length $|a|$ was 1000 and the kernel was the RBF kernel in all cases. Note that the original and centered results are superimposed in **a, c**.

setup replicated the one for the first experiment, with the results shown in figure 5. We also tested on data generated with equal probability from the two classes, with similar results.

For both centering methods, the re-centered and the original classifier are essentially the same for the whole range of shifts. Therefore we can safely conclude that there is no practical difference between the two centering methods.

An interesting aspect is revealed by the alignment plots: unlike figure 3 the alignment is maximum for the *shifted* data, while centering drastically *reduces* it. A quick analysis reveals the cause of this behavior: for a sufficiently large shift in feature space, the value of the alignment tends to

$$A(K_a) \;\longrightarrow\; \frac{(n^+ - n^-)^2}{n^2} \tag{59}$$

In our case, n^+ is 20 times larger than n^- which yields the value $A(K_s) = 0.82$, in perfect agreement with the experiments. This strongly cautions us that optimizing the kernel alignment may not always produce the best classifier.

8.3 Centering real data

In this set of experiments, we applied the simple centering algorithm to real data from the UCI repository. We computed the Gram matrices before and after centering, denoted by K and K_a respectively), trained an SVM for each of them, and evaluated its performance on an independent test data set. The data sets, training and test set sizes, kernel types and parameters are given in table 1. The SVM parameters were chosen so as to produce reasonable but not necessarily optimal classification results on the original data. This was done before the centering experiments, with one random training/validation split of the original data.

The results are summarized in figure 6. Each point in the figure represents the average of 10 random training/test splits. The test error plot shows that, as expected, centering has no effect in most cases but it improves performance occasionally (here, in 5 out of 11 cases). In none of the experiments did data centering hurt performance. In most cases where performance wasn't improved, the SVM classifiers from the centered and original data were virtually identical.

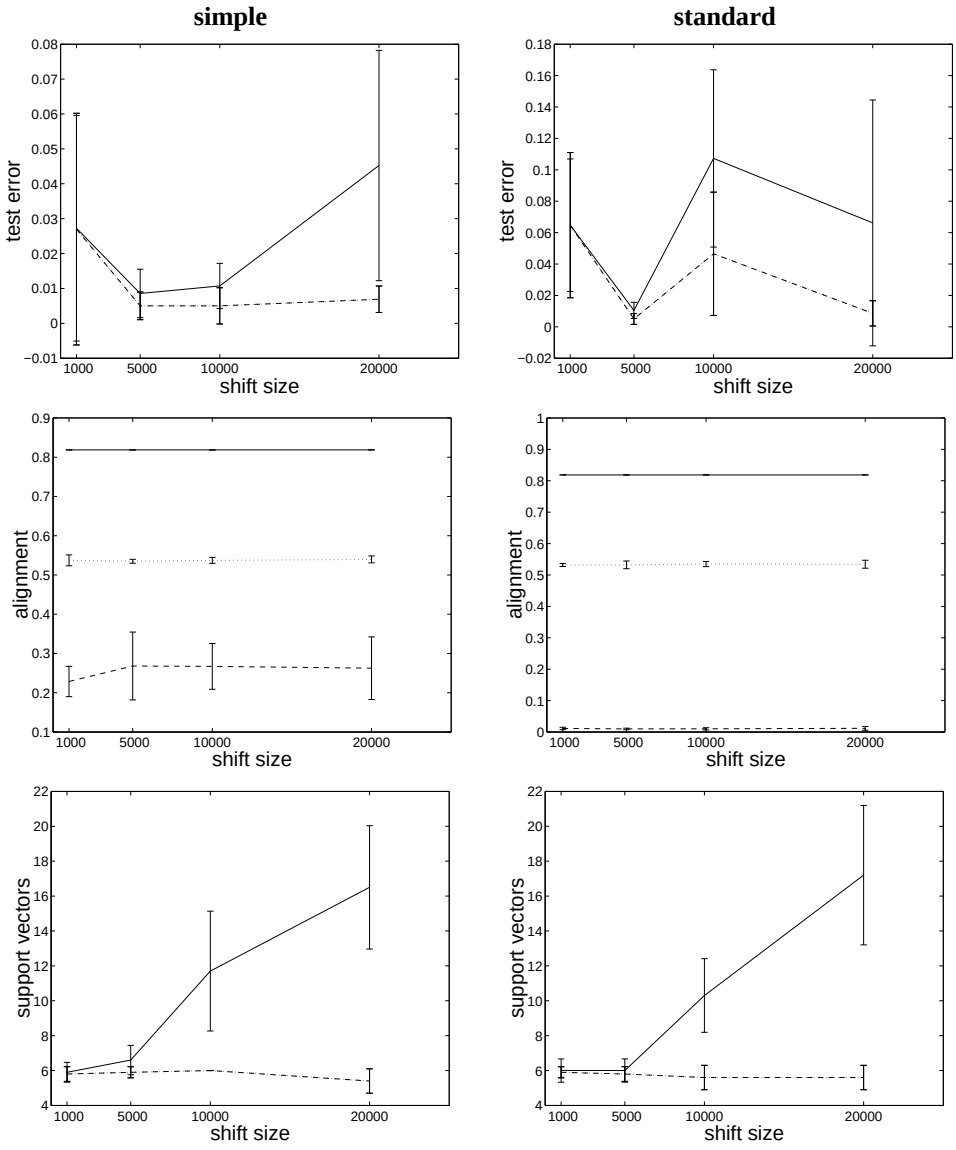

Figure 5: SVM classification with original (dotted line), shifted (full line) and re-centered (dashed line) data for different values of the shift length $|a|$. The original data are generated from the distribution shown in figure 1 (two concentric circles). These data are shifted in a random direction by an amount $|a|$ in feature space to obtain the shifted data. The shifted data are then re-centered as in section 3 (**simple**) or by moving the origin at the center of weight (**standard**) to obtain the re-centered data. The figures depict the test error, kernel alignment and number of support vectors for the resulting SVMs. Results are show average and standard deviation over 10 randomly sampled training sets of size 210. The original and centered SVMs are identical in all cases.

Table 1: The data sets used in the experiments.

Name	Description	# inputs	train set	test set	SVM parameters
cmc	Contraceptive data from the UCI repository [8] (class 1 vs all others)	9	400	523	RBF($\sigma^2 = 100$), $C = 1000$
glass	Glass data from the UCI repository (class 2 vs all others)	9	130	84	poly2, poly4, RBF($\sigma^2 = 2$), $C = 1000$
wdbc	Wisconsin breast cancer data from the UCI repository	30	312	257	poly2, RBF($\sigma^2 = 1000$), $C = 1000$
digi*ab*	Handwritten digits from the USPS (digit *a* vs digit *b* where *a,b* $\in \{0, 1, 2, 6\}$)	64	200	400	RBF($\sigma^2 = 1000$), $C = 1000$

The kernel alignment is slightly reduced in 2 of the 12 experiments and dramatically increased in 8 others. Again, we notice that improving the alignment per se does not necessarily guarantee an improvement in the classification performance.

8.4 Experiments with string kernels

This series of experiments investigates the properties of the new string kernels obtained as described in section 7. We used the equality element kernel with simple centering, followed by exponentiation ($\sigma^2 = 1$)as outlined at the end of section 7, and we compared the results with the uncentered equality kernel. The composite kernel is the GPAK in all cases.

The task was splice IE (intron-exon) junction classification. The data were DNA sequences of length 25. The positive examples had a splice junction between positions 12 and 13. Such a junction is marked by two fixed code letters in the immediately preceding positions 11 and 12. The negative examples were DNA sequences not containing a splice junction but containing the same symbols in positions 11, 12. Thus, any classifier trained on this data is forced to ignore the "obvious" junction information and to discover more subtle features to base its decision upon. The training (test) set comprised 300 (respectively 100) examples from each class.

The results, shown in figure 7 show that the centered GPA kernel is comparable as classification error with the original GPA kernel, without having a definite advantage. We will study better origin shift methods in the future. Another surprising result that requires further investigation is shown in figure 7, b. The number of support vectors for the centered GPAK is consistently and significantly reduced in comparison to the original GPAK across all experiments.

8.5 Experiments with the correction term in standard centering

The following experiments illustrate the effect of ignoring the correction Δb in standard centering. To simulate the various positions of the origin w.r.t the data, the data set was first shifted in a random direction by a fixed amount $|a|$ and the resulting kernel K_s was computed. We use this as the "original" kernel, and center it by the standard method to obtain K_a. We compute the centered but **uncorrected** SVM (b_u, α^u) by calling SVM-SOLVER(K_a, y). We compute the corrected $b_c = b_u - \Delta b$ obtaining the **corrected** classifier (b_c, α^u). We evaluate the performance of the **corrected** and **uncorrected** classifier on test data, by using the kernel K_s (the original kernel in our setting), α^u and b_c, b_u respectively. We also evaluate, for comparison, the performance of the **original** classifier produced by SVM-SOLVER(K_s, y).

We ran this experiment on the artificial data generated from two concentric distributions.

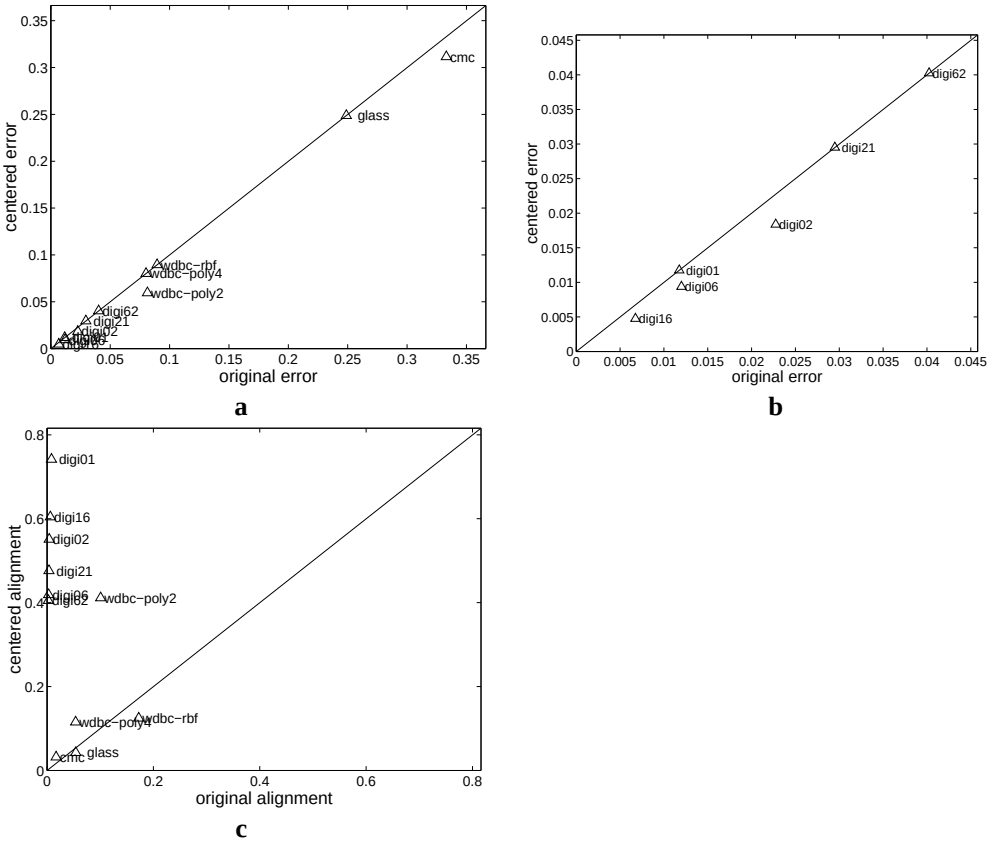

Figure 6: SVM classification of original versus centered data in 11 experiments with 9 data sets: **(a)** test error, **(b)** enlargement of the lower left corner of **(a)**, **(c)** kernel alignment. Each data point is the average of 10 random training/test splits. The data sets are described in table 1.

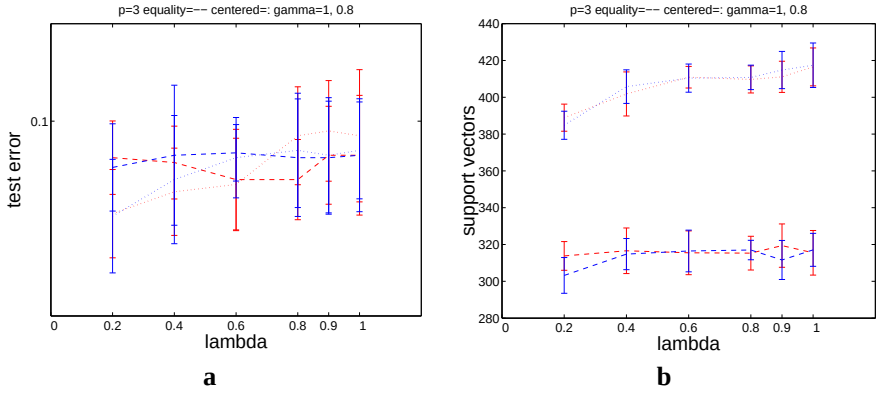

Figure 7: DNA splice junction classification with string kernels. Test error (**a**) and number of support vectors (**b**) versus λ for the original kernel (dotted) and for the centered kernel (dashed); $\gamma = .8$ (blue) or $\gamma = 1$ (red); $p = 3$ in all experiments. The results are averaged over 10 runs.

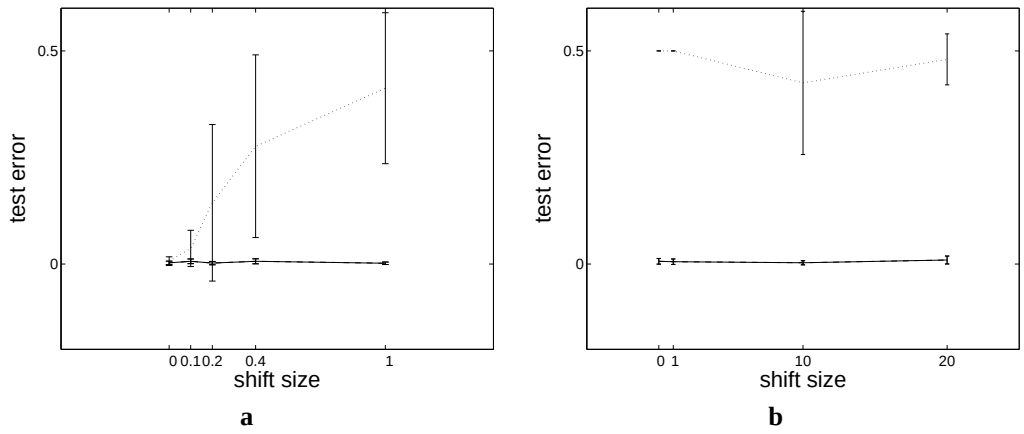

Figure 8: Classification error of the **original** (full), **corrected** (dashed), and **uncorrected** (dotted) on the artificial data from the distribution in figure 1 (the first two overlapping). The shift size denotes the amount of shift in the original data. The kernel used was RBF with $\sigma^2 = 1.7$ in **a**, and degree 2 polynomial in **b**; $C = 1000$ for all experiments. The plots represent means and standard deviations of the classification error over 40 (**a**), respectively 10 (**b**) experiments.

The training (test) set had 200 examples, 100 from each class. Figure 8,a shows the test error for the RBF kernel, at different shift values. It can be seen that even for very small shift values, the effect of neglecting the Δb correction is visible. These values of the shift are too small to cause numerical problems, so that the **original** and **corrected** classifiers are virtually identical in all experiments. The **uncorrected** classifier performs worse due to its wrong threshold value, showing that an erroneous data centering will have a negative effect on classification error.

The effects are even more dramatic if the degree 2 polynomial kernel is used for the same task. Then even if the original data are not shifted at all, standard centering without correction completely compromises the classification performance, as figure 8,b demonstrates.

8.6 Experiments with the kernel alignment

In the following we experimented with using data centering in conjunction with the kernel alignment method of [6]. We constructed 4 SVM classifiers as described below, and compared their classification performance. We also computed the alignment of the Gram matrix to the training data and the number of support vectors for all four classifiers.

The first SVM classifier, called **original** was constructed from the data with no centering in feature space. The second, **centered** included the simple centering algorithm in section 3. The classifier called **aligned** is the classifier that maximizes the kernel alignment as in [6]. The fourth, **centered+aligned** first centers the data in feature space, then applies the alignment optimization to the resulting Gram matrix.

We evaluated all classifiers on the artificial task of separating two concentric data sets and on some of the real classification problems described in table 1. For the real problems, the training (test) set sizes are those given in 1. The artificial data was generated from the same distribution as in section 8.2 and the training and test set sizes were both equal to 210. To evaluate the aligned kernel on new data points, we devised the method described below. The results are shown in figure 9.

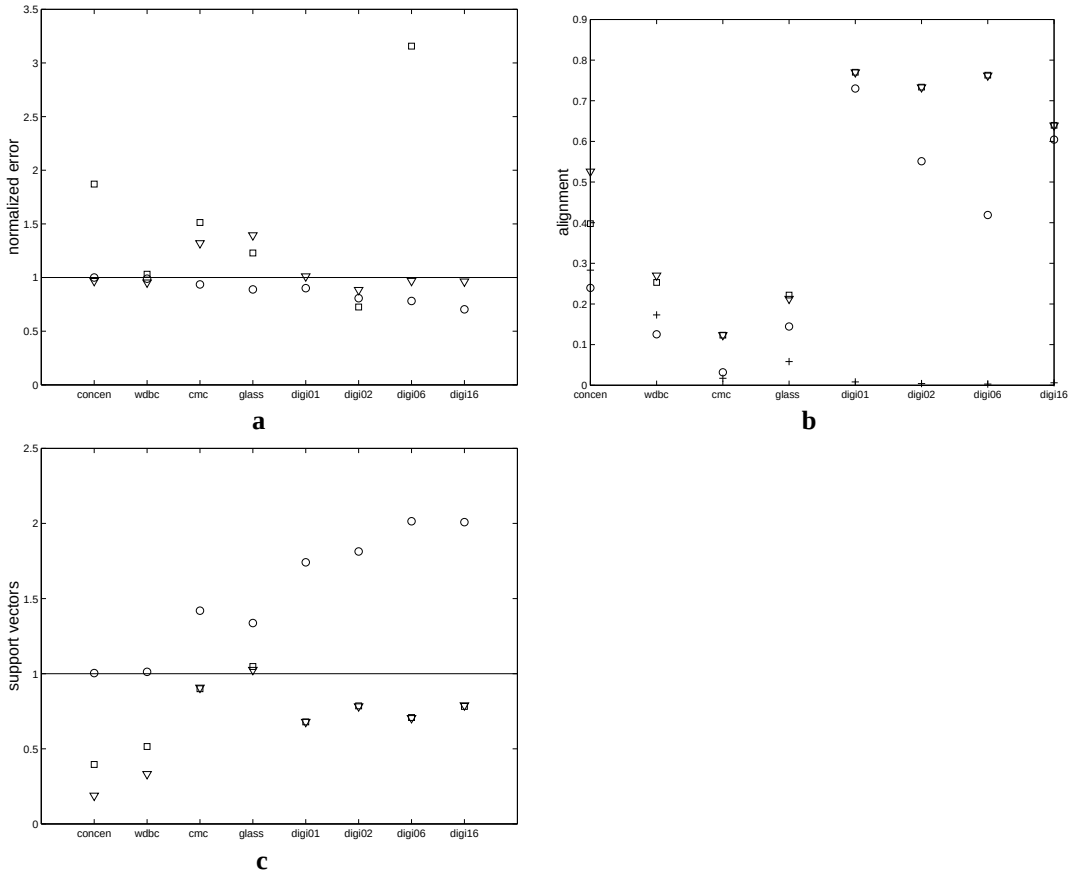

Figure 9: Centering and maximizing alignment on several data sets: **(a)** test error (divided by test error of original SVM), **(b)** kernel alignment, **(c)** number of support vectors (divided by the number of support vectors of the original SVM). Circles represent the **centered** SVM, squares represent the **aligned** SVM, triangles the **centered+aligned** SVM and pluses (in **b**) represent the original SVM. Shown are averages over 20 random runs. The error bars are small and have been omitted for clarity. The runs are on random training/test set splits for the real data sets, and randomly generated data sets for the concentric data. The concentric data is generated from the same distribution as for figure 5. In **a**, the normalized error of the **centered+aligned** SVM for `digi01` and `digi16` exceeds 3.5 and is not shown.

The alignment plots show, not surprisingly, that the **aligned** SVM has a consistently larger alignment than the **original** and **centered** SVM. The **centered+aligned** SVM's alignment is in most cases as least as good as that of the **aligned** SVM. The number of support vectors is decreased relative to the **original** SVM in the SVM's that use alignment, and is increased in the **centered** SVM.

The test error plots show, however, that maximizing the kernel alignment does not generally improve the classification error. While the **centered** SVM never increases the average classification error, and produces slight improvements in some cases, the **aligned** SVM improves the average error in one out of 8 classification problems and does not change it in another one. In the other 6 cases, the **aligned** SVM performs significantly worse than the **original** SVM (sometimes by an order of magnitude). The situation is a little better for the **centered+aligned** SVM: of the 8 problems, classification is improved in 1, degraded in 2 and about the same as the **original** SVM in the remaining 5 problems. Therefore, we strongly recommend data centering as a preprocessing step for any alignment maximization algorithm.

8.6.1 Classifying new data points with an aligned kernel

Changing the kernel by optimizing the alignment causes the same problem as shifting the data in feature space; namely, the kernel is not given by an explicit function any more. Here we show how to circumvent this problem and compute the value of the kernel for new data points $\tilde{x}$ after the SVM is obtained.

Let S be the set of support vectors of the aligned SVM. Denote by $< .,. >_A$ the scalar product associated with the aligned kernel and by K_A the corresponding Gram matrix restricted to the indices in S. Let $< .,. >$ and K have similar meanings for the original SVM. We need the value of $< x_i, x >_A$ for x_i a support vector and x a test point. Assume for simplicity that the support vectors are linearly independent. Note also that any component of x that is not in the span of the support vectors can be ignored since its contribution to f will be 0. Under these conditions, there is a unique representation of x in terms of the support vectors

$$x = \sum_{j \in S} c_j x_j$$

Using this representation, with a few calculations, we obtain:

$$< x_i, x > = \sum_{j \in S} c_j K(\tilde{x}_i, \tilde{x}_j) \tag{60}$$

$$< x_i, x >_A = \sum_{j \in S} c_j K_A(\tilde{x}_i, \tilde{x}_j) \tag{61}$$

Denote by h the vector $[h_i]_{i \in S},\ h_i =< x_i, x >$ and by $\overline{\alpha}$ the vector $[\alpha_i y_i]_{i \in S}$. Then, from equations (5, 60, 61) we obtain

$$< w, x >_A = \overline{\alpha}^T (K^{-1} K_A) h \tag{62}$$

For comparison, with the present vector notation, the value of $< w, x >$ for the original SVM is

$$< w, x > = \overline{\alpha}^T h \tag{63}$$

The solution above can be extended in a straightforward manner to the case when the support vectors are not linearly independent.

9 Discussion

This paper has presented a family of methods for data shifting and centering in feature space. They can be used in conjunction with any kernel machine that incorporates the

information from the data in a Gram matrix. Data centering in feature space does not, in theory, affect the resulting classifier. We have shown that in practice, it can have a beneficial effect when the Gram matrix is ill conditioned due to a poor position of the origin relative to the data in feature space. We have found no instances where data centering hurt the classification performance.

When used for data centering, translation in feature space requires extra work only in the training stage of the SVM. The extra computations are of the order n^2, but can be reduced by standard sampling schemes.

There have been many previous studies on kernel adaptation [1, 4, 6]. Our centering methods differ from the previous as they do not attempt to obtain a more appropriate kernel and they do not change the geometry of the problem. The aim of data centering is merely to hand the SVM-SOLVER a problem instance with better numerical properties.

Additionally, the paper has shown that the "standard" centering method present in the literature requires a correction term for b and has introduced a previously unpublished method for using the "aligned" kernel on test data. The experiments have also illustrated interesting aspects of the (lack of) relationship between the kernel alignment and classification performance in practice. In particular, translation in feature space can greatly change the alignment with no effect on the classifier performance.

It is interesting to compare centering in feature space as described here with centering the data in input space, advocated occasionally as a method of getting improved classifier performance. While data translation (and in particular centering) in feature space is transparent from the point of view of the final classifier, shifting the data in input space results in a non-linear transformation in feature space and thus can strongly influence the final classifier. Here we point out that centering the data around the origin in input space can have as result a kind of centering in feature space as well. If the input space is Euclidean, then centering by definition reduces the average norm (in $\tilde{\mathcal{X}}$) of the data. If in addition the kernel is such that to a small $|\tilde{x}|$ corresponds a small $K(\tilde{x}, \tilde{x})$, a condition satisfied by the majority of off-the-shelf kernels, then the diagonal elements K_{ii} of the Gram matrix cannot be too large, implying that the origin cannot be too far away from the data.

There is growing interest, however, in constructing kernels for non-Euclidean data (text data for example) where centering in input space is not possible. Data centering in feature space may become a standard preprocessing step transparent to the user, but potentially important for obtaining a good solution.

The results of our theoretical investigation into data translation in feature space were used mainly for data centering. We envisage however a more interesting realm of applications: shifting the data in order to obtain new kernels, parametrized by the shift. Obtaining a new kernel by origin shifts is possible with composite kernel, such as the ones used in the classification of string data (see e.g [14]). We have experimented with the simple centering method for the GPAK and have obtained promising results. In the future we are planning to investigate optimizing the shift of the element kernels according to a criterion related to the classification performance.

Acknowledgments

The author thanks Deepak Verma for discussions and some help with the code, Chris Watkins for sharing his knowledge of string kernels and Chih-Chung Chang and Chih-Jen Lin for writing and making available the **LIBSVM** software.

References

[1] S. Amari and S. Wu. Improving support vector machines by modifying kernel functions. *Neural Networks*, pages 783–789, 1999.

[2] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

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

[4] N. Cristianini, C. Campbell, and J. Shawe-Taylor. Dynamically adapting kernels in support vector machines. NeuroCOLT Technical Report NC-TR-98-017, Royal Holloway College, University of London, UK, 1998.

[5] Nello Cristianini. Support vector and kernel machines. Tutorial at ICML, 2001.

[6] Nello Cristianini, John Shawe-Taylor, Andre Elisseeff, and Jaz Kandola. On kernel target alignment. In Tom Dietterich, Sue Becker, and David Cohn, editors, *Neural Information Processing Systems*, number 14, Cambridge, MA, 2002. MIT Press.

[7] David Haussler. Convolution kernels on discrete structures. Technical Report USCS-CRL-99-10, University of California at Santa Cruz, Department of Computer Science, Santa Cruz, CA, 1999.

[8] P. M. Murphy and D. W. Aha. U. C. Irvine Machine Learning Repository. ftp://ftp.ics.uci.edu/pub/machine-learning-databases/.

[9] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. Technical Report 99-87, Microsoft Research, 1999. To appear in *Neural Computation*, 2001.

[10] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998. Technical Report No. 44, 1996, Max Planck Institut für biologische Kybernetik, Tübingen.

[11] B. Schölkopf, A. Smola, R. Williamson, and P. L. Bartlett. New support vector algorithms. NeuroCOLT Technical Report NC-TR-98-031, Royal Holloway College, University of London, UK, 1998. Published in *Neural Computation* **12**(5):1207–1245, 2000.

[12] Bernhard Schölkopf. Statistical learning and kernel methods. Technical Report MSR-TR-2000-23, Microsoft Research, Cambridge, UK, 2000.

[13] John Shawe-Taylor, Nello Cristianini, and Jaz Kandola. On the concentration of spectral properties. In Sue Becker Tom Dietterich and David Cohn, editors, *Neural Information Processing Systems*, number 14, Cambridge, MA, 2002. MIT Press.

[14] Christopher J. C. H. Watkins. Dynamic alignment kernels. Technical Report CSD-TR-98-11, Royal Holloway, University of London, 1999.

[15] Christopher J. C. H. Watkins. Penalty gap anova kernels. personal communication, June 2002.

[16] C Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *International Conference on Machine Learning*, number 17, 2000.