Training Invariant Support Vector Machines

Dennis DeCoste decoste@aig.jpl.nasa.gov
Jet Propulsion Laboratory, MS 126-347, 4800 Oak Grove Drive, Pasadena, CA 91109, USA and California Institute of Technology.

Bernhard Schölkopf bs@conclu.de
Microsoft Research Limited, 1 Guildhall Street, Cambridge, UK

Abstract. Practical experience has shown that in order to obtain the best possible performance, prior knowledge about invariances of a classification problem at hand ought to be incorporated into the training procedure. We describe and review all known methods for doing so in support vector machines, provide experimental results, and discuss their respective merits. One of the significant new results reported in this work is our recent achievement of the lowest reported test error on the well-known MNIST digit recognition benchmark task, with SVM training times that are also significantly faster than previous SVM methods.

Keywords: support vector machines, invariance, prior knowledge, image classification, pattern recognition

1. Introduction

In 1995, LeCun et al. published a pattern recognition performance comparison noting the following:

> "The optimal margin classifier has excellent accuracy, which is most remarkable, because unlike the other high performance classifiers, it does not include *a priori* knowledge about the problem. In fact, this classifier would do just as well if the image pixels were permuted by a fixed mapping. [...] However, improvements are expected as the technique is relatively new."

Two things have changed in the years since this statement was made. First, optimal margin classifiers, or support vector machines (SVMs) (Boser et al., 1992; Cortes and Vapnik, 1995; Vapnik, 1995), have turned into a mainstream method which is part of the standard machine learning toolkit. Second, methods for incorporating prior knowledge into optimal margin classifiers have become part of the standard SV methodology.

These two things are actually closely related. Initially, SVMs had been considered a theoretically elegant spin-off of the general but, allegedly, largely useless VC-theory of statistical learning. In 1996, using the first methods for incorporating prior knowledge (Schölkopf et al., 1996), SVMs became competitive with the state of the art in the handwritten digit classification benchmarks (LeCun et al., 1995) that were popularized in the machine learning community by AT&T and Bell Labs. At that point, practitioners who are not interested in theory, but in results, could no longer ignore SVMs. In this sense, the methods to be described below actually helped pave the way to make SVM a widely used machine learning tool.

The present paper tries to provide a snapshot of the state of the art in methods to incorporate prior knowledge about invariances of classification problems. It gathers material from various sources. Partly, it reviews older material that has appeared elsewhere (Schölkopf et al., 1996; Schölkopf et al., 1998a), but never in a journal paper, and partly, it presents new techniques, applications, and experimental results. It is organized as follows. The next section introduces the concept of prior knowledge, and discusses what types of prior knowledge are used in pattern recognition. The present paper focuses on prior knowledge about invariances, and Section 3 describes methods for incorporating invariances into SVMs. Section 4 introduces a specific way which combines a method for invariant SVMs with the widely used SMO training algorithm. Section 5 reports experimental results on all presented methods, and Section 6 discusses our findings.

2. Prior Knowledge in Pattern Recognition

By *prior knowledge* we refer to all information about the learning task which is available in addition to the training examples. In this most general form, only prior knowledge makes it possible to generalize from the training examples to novel test examples.

For instance, many classifiers incorporate general *smoothness assumptions* about the problem. A test patterns which is similar to one of the training examples will thus tend to be assigned to the same class. For SVMs, it can be shown that using a kernel function K amounts to enforcing smoothness with a regularizer $\|Pf\|^2$, where f is the estimated function, and K is a Green's function of P^*P, where P^* is the adjoint operator of P ((Smola et al., 1998; Girosi, 1998), cf. also Poggio and Girosi (1989)). In a Bayesian maximum-a-posteriori setting, this corresponds to a smoothness prior of $\exp(-\|Pf\|^2)$ (Kimeldorf and Wahba, 1970).

A second method for incorporating prior knowledge, already somewhat more specific, consists of *selecting features* which are thought to be particularly informative or reliable for the task at hand. For instance, in handwritten character recognition, correlations between image pixels that are nearby tend to be more reliable than the ones of distant pixels. The intuitive reason for this is that variations in writing style tends to leave the local structure of a handwritten digit fairly unchanged, while the global structure is usually quite variable. In the case of SVMs, this type of prior knowledge is readily incorporated by designing polynomial kernels which compute mainly products of nearby pixels (Schölkopf et al., 1998a) (cf. Table II). While the example given here applies to handwritten character recognition, it is clear that a great deal of problems have a similar *local* structure, for instance, problems of computational biology involving nucleotide sequences. Zien et al. (2000) have used this analogy to engineer kernels that outperform other approaches in the problem of recognizing translation initiation sites on DNA or mRNA sequences, i.e., positions which mark regions coding proteins. Haussler (1999) has gone further than that and constructed convolutional kernels which compute features targeted at problems such as those of bioinformatics.

One way to look at feature selection is that it changes the representation of the data, and in this, it is not so different from another method for incorporating prior knowledge in SVMs that has recently attracted attention. In this method, it is assumed that we have knowledge about probabilistic models generating the data. Specifically, let $p(x|\Theta)$ be a generative model that characterizes the probability of a pattern x given the underlying parameter Θ. It is possible to construct a class of kernels which are invariant with respect to reparametrizations of Θ and which, loosely speaking, have the property that $K(x, x')$ is the similarity of x and x' subject to the assumption that they both stem from the generative model. These kernels are called Fisher kernels (Jaakkola and Haussler, 1999). It can be shown that they induce regularizers which, in turn, take account of the underlying model $p(x|\Theta)$ (Oliver et al., 2000). A different approach to designing kernels based on probabilistic models is the one of Watkins (2000).

Finally, we get to the type prior knowledge that the present paper will deal with: prior knowledge about *invariances*. For instance, in image classification tasks, there exist transformations which leave class membership invariant (e.g. translations).

3. Incorporating Invariances into SVMs

In the present section, we describe methods for incorporating invariances into SVMs. We distinguish three types of methods, to be addressed in the following three subsections:

- engineer kernel functions which lead to invariant SVMs

– generate artificially transformed examples from the training set, or subsets thereof (e.g. the set of SVs)

– combine the two approaches by making the transformation of the examples part of the kernel definition

This paper will mainly focus on the latter two methods. However, for completeness, we also include some material on the first method, which we will presently describe.

3.1. Invariant Kernel Functions

This section briefly reviews a method for engineering kernel functions leading to invariant SVMs. Following Schölkopf et al. (1998a), we consider linear decision functions $f(\mathbf{x}) = \operatorname{sgn}(g(\mathbf{x}))$, where

$$g(\mathbf{x}) := \sum_{i=1}^{\ell} v_i (B\mathbf{x} \cdot B\mathbf{x}_i) + b. \tag{1}$$

To get local invariance under transformations forming a Lie group $\{\mathcal{L}_t\}$, [1] one can minimize the regularizer

$$\frac{1}{\ell} \sum_{j=1}^{\ell} \left(\frac{\partial}{\partial t}\Big|_{t=0} g(\mathcal{L}_t \mathbf{x}_j) \right)^2 . \tag{2}$$

One can show that this is equivalent to performing the usual SVM training after preprocessing the data with

$$B = C^{-\frac{1}{2}}, \tag{3}$$

where

$$C = \frac{1}{\ell} \sum_{j=1}^{\ell} \left(\frac{\partial}{\partial t}\Big|_{t=0} \mathcal{L}_t \mathbf{x}_j \right) \left(\frac{\partial}{\partial t}\Big|_{t=0} \mathcal{L}_t \mathbf{x}_j \right)^{\top} . \tag{4}$$

In other words: the modification of the kernel function is equivalent to whitening the data with respect to the covariance matrix of the vectors

[1] The reader who is not familiar with this concept may think of it as a group of transformations where each element is labelled by a set of continuously variable parameters, cf. also (Simard et al., 1992; Simard et al., 1993). Such a group may be considered also to be a manifold, where the parameters are the coordinates. For Lie groups, it is required that all group operations are smooth maps. It follows that one can, for instance, compute derivatives with respect to the parameters. Examples of Lie groups are the translation group, the rotation group, or the Lorentz group; for details, see e.g. (Crampin and Pirani, 1986).

$\pm\frac{\partial}{\partial t}|_{t=0}\mathcal{L}_t\mathbf{x}_i$. To get a clearer understanding of what this means, let us diagonalize B using the matrix D containing the eigenvalues of C, as

$$B = C^{-\frac{1}{2}} = SD^{-\frac{1}{2}}S^{\top}. \tag{5}$$

The preprocessing of the data thus consists of projecting $\mathbf{x}$ onto the Eigenvectors of C, and then dividing by the square roots of the Eigenvalues. In other words, the directions of main variance of $\pm\frac{\partial}{\partial t}|_{t=0}\mathcal{L}_t\mathbf{x}$ — the directions that are most affected by the transformation — are scaled back. Note that the leading S in (5) can actually be omitted, since it is unitary (since C is symmetric), and it thus does not affect the value of the dot product.

Moreover, in practice, it is advisable to use $C_\lambda := (1-\lambda)C+\lambda I$ rather than C. This way, we can balance the two goals of maximizing the margin and of getting an invariant decision function by the parameter $\lambda \in [0, 1]$.

Schölkopf et al. (1998a) have obtained performance improvements by applying the above technique in the linear case. However, these improvements are not nearly as large as the ones that we will get with the methods to be described in the next section,[2] and thus we put the main emphasis in this study on the latter.

In the nonlinear case, the above method still applies, using $D^{-\frac{1}{2}}S^{\top}\Phi(\mathbf{x})$. Here, Φ is a feature map corresponding to the kernel chosen, i.e., a (usually nonlinear) map satisfying $K(\mathbf{x}, \mathbf{x}') = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}'))$. To compute projections of the mapped data $\Phi(\mathbf{x})$ onto the eigenvectors of S, one needs to essentially carry out kernel PCA on S (cf. (Schölkopf, 1997; Chapelle and Schölkopf, 2000)). For further material on methods to construct invariant kernel functions, cf. Vapnik (1998) and Burges (1999).

3.2. Virtual Examples and Support Vectors

One way to make classifiers invariant is to generate artificial training examples, or *virtual examples*, by transforming the training examples accordingly (Baird, 1990; Poggio and Vetter, 1992). It is then hoped that the learning machine will extract the invariances from the artificially enlarged training data.

Simard et al. (1992) report that, not surprisingly, training a neural network on the artificially enlarged data set is significantly slower, due to both correlations in the artificial data and the increase in training set size. If the size of a training set is multiplied by a number of desired invariances (by generating a corresponding number of artificial

[2] We think that this is because that study focused on the linear case. The nonlinear case has recently been studied by Chapelle and Schölkopf (2000).

examples for each training pattern), the resulting training sets can get rather large (as the ones used by Drucker et al. (1993)). However, the method of generating virtual examples has the advantage of being readily implemented for all kinds of learning machines and symmetries, even discrete ones such as reflections (which do not form Lie groups). It would thus be desirable to construct a method which has the advantages of the virtual examples approach without its computational cost. The *Virtual SV* method (Schölkopf et al., 1996), to be described in the present section, retains the flexibility and simplicity of virtual examples approaches, while cutting down on their computational cost significantly.

In Schölkopf et al. (1995) and Vapnik (1995), it was observed that the SV set contains all information necessary to solve a given classification task. It particular, it was possible to train any one of three different types of SV machines solely on the SV set extracted by another machine, with a test performance not worse than after training on the full database — SV sets seemed to be fairly robust characteristics of the task at hand. This led to the conjecture that it might be sufficient to generate virtual examples from the Support Vectors only. After all, one might hope that it does not add much information to generate virtual examples of patterns which are not close to the boundary. In high-dimensional cases, however, care has to be exercised regarding the validity of this intuitive picture. Experimental tests on high-dimensional real-world problems hence were imperative, and they have confirmed that the method works very well (Schölkopf et al., 1996). It proceeds as follows (cf. Figure 1):

1. train a Support Vector machine to extract the Support Vector set

2. generate artificial examples, termed *virtual support vectors*, by applying the desired invariance transformations to the support vectors

3. train another Support Vector machine on the generated examples[3].

If the desired invariances are incorporated, the curves obtained by applying Lie symmetry transformations to points on the decision surface should have tangents parallel to the latter (Simard et al., 1992). If we use small Lie group transformations to generate the virtual examples, this implies that the Virtual Support Vectors should be approximately as close to the decision surface as the original Support Vectors. Hence,

[3] Clearly, the scheme can be iterated; however, care has to exercised, since the iteration of local invariances would lead to global ones which are not always desirable — cf. the example of a '6' rotating into a '9' (Simard et al., 1993).

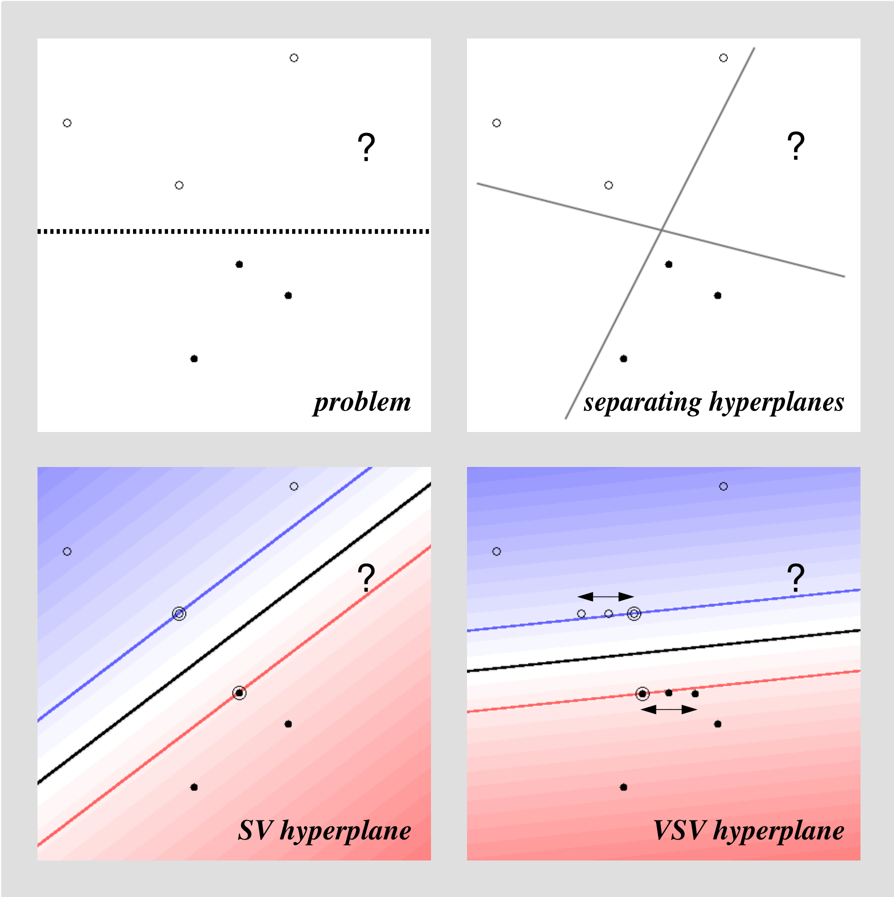

Figure 1. Suppose we have prior knowledge indicating that the decision function should be invariant with respect to horizontal translations. The true decision boundary is drawn as a dotted line (*top left*); however, as we are just given a limited training sample, different separating hyperplanes are conceivable (*top right*). The SV algorithm finds the unique separating hyperplane with maximal margin (*bottom left*), which in this case is quite different from the true boundary. For instance, it would lead to wrong classification of the ambiguous point indicated by the question mark. Making use of the prior knowledge by generating Virtual Support Vectors from the Support Vectors found in a first training run, and retraining on these, yields a more accurate decision boundary (*bottom right*). Note, moreover, that for the considered example, it is sufficient to train the SV machine *only* on virtual examples generated from the Support Vectors (from (Schölkopf, 1997)).

they are fairly likely to become Support Vectors after the second training run; in this sense, they add a considerable amount of information to the training set.

However, as noted above, the method is also applicable if the invariance transformations do not form Lie groups. For instance, performance improvements in object recognition of bilaterally symmetric objects have been obtained using reflections (Schölkopf, 1997).

3.3. Kernel Jittering

An alternative to the VSV approach of pre-expanding a training set by applying various transformations is to perform those transformations inside the kernel function itself (DeCoste and Burl, 2000; Simard et al., 1993).

In particular, we have recently explored the notion of *jittering kernels*, in which any two given examples are explicitly jittered around in the space of discrete invariance distortions until the closest match between them is found.

For any kernel $K(x_i, x_j) \equiv K_{ij}$ suitable for traditional use in a SVM, consider a jittering kernel form $K^J(x_i, x_j) \equiv K^J_{ij}$, defined procedurally as follows:

1. consider all jittered forms [4] of example x_i (including itself) in turn, and select the one (x_q) "closest" to x_j; specifically, select x_q to minimize the metric distance between x_q and x_j in the space induced by the kernel. This distance is given by: [5]

$$\sqrt{K_{qq} - 2K_{qj} + K_{jj}}.$$

2. let $K^J_{ij} = K_{qj}$.

Multiple invariances (e.g., both translations and rotations) could be considered as well, by considering cross-products (i.e. all valid combinations) during the generation of all "jittered forms of x_i" in step 1 above. However, we have not yet investigated the practicality of such multiple invariance via kernel jittering in any empirical work to date.

For some kernels, such as radial-basis functions (RBFs), simply selecting the maximum K_{qj} value to be the value for K^J_{ij} suffices, since

[4] For example, Figure 4 shows some jittered forms of a particular example of digit "5".

[5] This corresponds to Euclidian distance in the feature space corresponding to the kernel, using the definition of the two-norm:

$$||Z_i - Z_j||^2 = (Z_i \cdot Z_i) - 2\ (Z_i \cdot Z_j) + (Z_j \cdot Z_j).$$

the K_{qq} and K_{jj} terms are constant (e.g. 1) in that case. This similarly holds for translation jitters, as long as sufficient padding exists so that no image pixels fall off the image after translations. In general, a jittering kernel may have to consider jittering either or both examples. However, for symmetric invariances such as translation, it suffices to jitter just one.

We refer to the use of jittering kernels as the *JSV* approach. A major motivation for considering JSV approachs is that VSV approaches scale at least quadratically in the number (J) of jitters considered, whereas under favorable conditions jittering kernel approaches can scale only linearly in J, as discussed below.

Jittering kernels are J times more expensive to compute than non-jittering kernels, since each K^J_{ij} computation involves finding a minimum over J K_{ij} computations. However, the potential benefit is that the training set can be J times smaller than the corresponding VSV method, since the VSV approach can expand the training set by a factor of J. Known SVM training methods scale at least quadratically in the number of training examples. Thus, the potential net gain is that JSV training may only scale linearly in J instead of quadratically as in VSV.

Furthermore, through comprehensive use of kernel caching, as is common in modern practical SVM implementations, even the factor of J slow-down in kernel computations using jittering kernels may be largely amortized away, either over multiple trainings (e.g. for different regularization C values) or within one training (simply because the same kernel value is often requested many times during training).

As long as the distance metric conditions are satisfied under such jittering (i.e. non-negative, symmetry, triangularity inequality), the kernel matrix defined by K^J_{ij} will be positive definite and suitable for traditional use in SVM training (i.e. it will satisfy Mercer's conditions, at least *for the given set of training examples*).

In practice, those conditions seem to almost always be satisfied and we rarely detect the symptoms of a non-positive definite kernel matrix during SVM training with jittering kernels. The SMO method seems to tolerate such minor degrees of non-positivity, although, as discussed in (Platt, 1999) this still somewhat reduces convergence speed. This rarity seems particularly true for simple translation invariances which we have focused on to date in empirical work. The one exception is that the triangular inequality is sometimes violated. For example, imagine three simple images A, B, and C consisting of a single row of three binary pixels, with A =(1,0,0), B=(0,1,0), and C=(0,0,1). The minimal jittered distances (under 1-pixel translation) between A and B and between B and C will be 0. However, the distance between A and C will be positive (e.g. d(A,C) $= \sqrt{2}$ for a linear kernel). Thus, the triangular inequality $d(A,B) + d(B,C) \geq d(A,C)$ is violated in that example.

Note that with a sufficiently large jittering set (i.e. such as one including both 1-pixel and 2-pixel translations for the above example), the triangularity inequality is not violated. We suspect that for reasonably complete jittering sets, the odds of triangularity inequality violations will become small. Based on our limited preliminary experiments with kernel jittering to date, it is still unclear how much impact any such violations will typically have on generalization performance in practice.

Jittering kernels have one other potential disadvantage compared to VSV approaches: the kernels must continue to jitter at test time. In contrast, the VSV approach effectively compiles the relevant jittering into the final set of SVs it produces. In cases where the final JSV SV set size is much smaller than the final VSV SV set size, the JSV approach can actually be as fast or even faster at test time. We have indeed observed this result to be frequent, at least for the relatively small tests tried to date. We do not yet know how common this case is for large-scale applications.

4. Efficient Invariance Training

Recent developments in decomposition methods for SVM training have demonstrated significant improvements in both space and time costs for training SVMs on many tasks. In particular, the SMO (Platt, 1999) and SVM^{light} (Joachims, 1999) implementations have become popular practical methods for large-scale SVM training. In this section, we discuss some improvements to the style of SVM training represented by approaches such as SMO and SVM^{light}. We have found these improvements to lead to significantly faster training times, in some cases by more than an order of magnitude. Our specific implementation and tests are based on enhancements to the variant of SMO described in (Keerthi et al., 1999). However, these ideas should be applicable to the original SMO specification, as well as SVM^{light}. For simplicity, we will cast our discussion below in terms of the well-known SMO specification (Platt, 1999).

For the sake of the rest of this section, we assume the SVM training task consists of the following Quadratic Programming (QP) dual formulation:

$$
\begin{array}{ll}
\textit{maximize:} & \\
& \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\
\textit{subject to:} & \\
& 0 \leq \alpha_i \leq C, \\
& \sum_{i=1}^{\ell} \alpha_i y_i = 0,
\end{array}
$$

where ℓ is the number of training examples, y_i is the label (+1 for positive example, -1 for negative) for the i-th training example (x_i), and $K(x_i, x_j)$ denotes the value of the SVM kernel function for i-th and j-th examples. The output prediction of the SVM, for any example x is

$$f(x) = sign(\sum_{i=1}^{\ell} \alpha_i y_i K(x, x_i) + b),$$

where scalar b (bias) and vector of alphas α (of length ℓ) are the variables determined by the above QP optimization problem.

4.1. Key Efficiency Issue: Maximizing Cache Reuse

Employing general QP solvers to the SVM optimization problem typically involves $O(\ell^2)$ space and $O(\ell^3)$ time complexities. SMO can avoid the $O(\ell^2)$ space cost by avoiding the need for an explicit full kernel matrix — it computes (and recomputes) elements as needed. Evaluating the current SVM outputs during training for each of the ℓ examples involves ℓ summations, each over all current support vectors. Thus, SMO (and any other approach which similarly checks KKT conditions during training) necessarily suffers time complexity of at least $O(L \cdot \ell)$, where L is the final number of support vectors, since it would require at least one final full pass over all ℓ examples simply to check that all KKT conditions are satisfied.

In practice, computing elements of the kernel matrix often dominates training time. The same element will typically be required many times, since many (e.g. tens or hundreds) of passes over the training set may be required before convergence to the final solution. Thus, modern SMO implementations try to cache as many kernel computations as possible.

However, caching all computed kernels is generally not possible, for two key reasons:

1. L is often a significant fraction of ℓ (e.g. 5% or more) and thus the space required to avoid any kernel recomputations would often prohibitively be $O(\ell^2)$.

2. The intermediate (working) set of (candidate) support vectors is typically much larger than the final size L. [6]

Since full passes over all examples are required in SMO whenever the working set of candidate support vectors becomes locally optimal, the kernel cache is often implemented simply as the most frequently or recently accessed rows of the implicit full kernel matrix. For instance, for the 60,000 training examples of the MNIST data set (to be discussed in Section 5), 512 megabytes of computer memory reserved for the kernel cache, and IEEE single precision (4 byte) representations of the kernel values, only about 2230 rows can be cached at one time (e.g. only about half of all the support vectors for some of the MNIST binary recognition tasks).

Of particular concern is that during SMO training, when the final set of support vectors is not yet known, it turns out to be common for examples which will eventually not be support vectors to end up grabbing and hoarding cache rows first, requiring other kernel rows (including those of final support vectors) to be continually recomputed until those examples cease to be support vectors and release their cache memory. We will refer to this problem as *intermediate support vector bulge*. This problem is particularly critical for VSV methods, since they greatly enlarge the number of effective training examples. Addressing this key issue will be the subject of Section 4.2.

4.2. Digestion: Reducing Intermediate SV Bulge

SMO alternates between "full" and "inbounds" iterations, in which either all examples or just the examples with alphas between 0 and C are considered, respectively. In each inbounds stage, optimization continues until not further alpha changes occur. In each full stage, each example is optimized only once (with some other heuristically-selected example), to see if it becomes part of the working inbounds set and triggers new work for the next inbounds stage. During the course of SMO full iterations, it is not uncommon for the number of working candidate support vectors to become orders of magnitude larger than the number that will result at the end of the following inbounds iteration.

When the kernel cache is sufficiently large (i.e. R near ℓ) then such SV bulge behavior is not critically problematic. However, even in that

[6] Incidentally, this is why we expect, for large-scale problems, that JSV approaches may be able to effectively amortize away much of their extra jittering kernel costs (J times slower than VSV's standard kernels). That is, the JSV working set will be J times smaller than for the corresponding VSV, and thus might reuse a limited cache more effectively.

case, the computation of the SVM outputs for each example can be quite excessive, since the time cost of computing an output is linear in the number of current support vectors. So, it would be generally useful to keep the the size of the support vector set closer to minimal.

A key problem arises when the size of the working candidate support vector set has already exceeded R. In that case, any additional support vectors will not be able to cache their kernel computations.

To address this key problem, we have extended the SMO algorithm to include a concept we call *digestion*. The basic idea is to jump out of full SMO iterations early, once the working candidate support vector set grows by a large amount. This switches SMO into an "inbounds" iteration in which it fully "digests" the working candidate SV set, reducing it to a locally minimal size, before switching back to a full iteration (and returning first to the example at which it had previously jumped out).

Digestion allows us to better control the size of the intermediate SV bulge, so as to best tradeoff the cost of overflowing the kernel cache against the cost of doing more inbounds iterations that standard SMO would.

Digestion is similar to other (non-SMO) chunking methods, in that it performs full optimizations over various large subsets of the data. However, it differs significantly in that these full optimizations are triggered by heuristics based on maximizing the likely reuse of the kernel cache, instead of using some predetermined and fixed chunk size. We suspect that digestion would be even more useful and distinct when using sparse kernel caches (e.g. hash tables), although our current implementation caches entire rows of the kernel matrix (much like other published implementations, such as SVM^{light}).

Our current approach involves user-defined settings to reflect tradeoff heuristics. We expect that future work could provide meta-learning methods for identifying the best values for these settings, tuned for the particular nature of specific domains (e.g. costs of each kernel computation), over the course of multiple training sessions. Our settings were choosen to be particularly reasonable for training set sizes on the order of ℓ = 10,000 to 100,000 and a kernel cache of about 800 megabytes, which is the case for our MNIST experiments (see Section 5.1.1). Our heuristics were (highest-priority first):

1. Once digestion has been performed, wait until at least some number (default = 1000) of new (currently uncached) kernel values are computed before digesting again.

2. After the current full iteration has already required some (default = 100,000) new kernel computations, force a digestion.

3. Otherwise, wait at least as long between digestions as some factor (default = 2) times the time spent in the previous inbounds iteration. The idea is that if the inbounds iterations are expensive themselves, one must tolerate longer full iterations as well.

4. When the kernel cache is full (i.e. working candidate SV set size $> R$), digest as soon as the net number of new working candidate SVs grows past some threshold (default = 200). The idea here is that the kernel values computed for each such new SV cannot be cached, so it is becomes critical to free rows in the kernel cache, if at all possible.

5. Digest whenever the net number of new SVs grows past some threshold (default = 1000). The intuition behind this is simply that output computations involve summations over all candidate SVs in the working set. Thus, it is often worthwhile to periodically make sure that this set is not many times larger than it should be.

One can easily imagine promising alternative heuristics as well, such as ones based on when the working candidate SV set grows by more than some percentage of the working candidate SV size at the start of a full iteration. The key point is that heuristics to induce digestion have proven to be useful in significantly reducing the complexity of SMO training on particularly difficult large-scale problems. For example, we observed speedups of over 5 times for training some of the SVMs in the MNIST examples of Section 5.1.1 when using the above heuristics and default settings.

5. Experiments

5.1. Handwritten Digit Recognition

We start by reporting results on two widely known handwritten digit recognition benchmarks, the USPS set and the MNIST set. Let us first described the databases. Both of them are available from http://www.kernel-machines.org/data.html.

The *US Postal Service (USPS)* database (see Figure 2) contains 9298 handwritten digits (7291 for training, 2007 for testing), collected from mail envelopes in Buffalo (LeCun et al., 1989). Each digit is a 16×16 image, represented as a 256-dimensional vector with entries between -1 and 1. Preprocessing consisted of smoothing with a Gaussian kernel of width $\sigma = 0.75$.

It is known that the USPS test set is rather difficult — the human error rate is 2.5% (Bromley and Säckinger, 1991). For a discussion, see

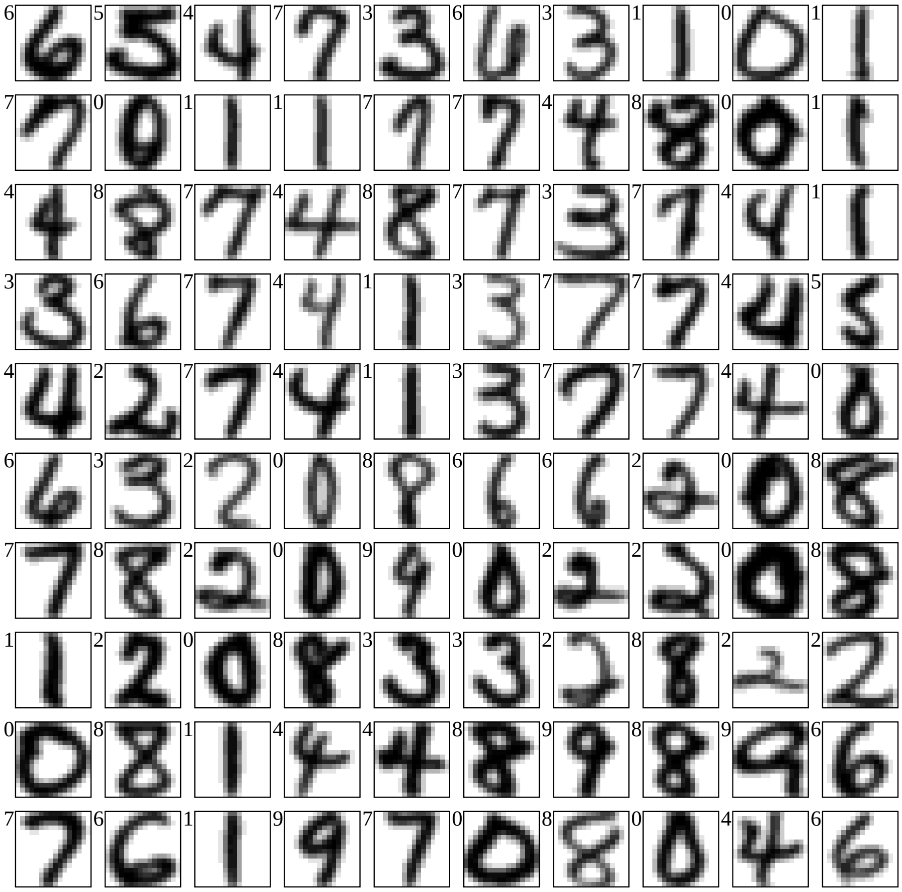

Figure 2. The first 100 USPS training images, with class labels.

(Simard et al., 1993). Note, moreover, that some of the results reported in the literature for the USPS set have been obtained with an enhanced training set. For instance, Drucker et al. (1993) used an enlarged training set of size 9709, containing some additional machine-printed digits, and note that this improves the accuracy on the *test* set. Similarly, Bottou and Vapnik (1992) used a training set of size 9840. Since there are no machine-printed digits in the test set that is commonly used (size 2007), this addition distorts the original learning problem to a situation where results become somewhat hard to interpret. For our experiments, we only had the original 7291 training examples at our disposal.

The *MNIST* database (Figure 3) contains 120000 handwritten digits, equally divided into training and test set. The database is a modified version of *NIST Special Database 3* and *NIST Test Data 1*. Training and test set consist of patterns generated by different writers. The

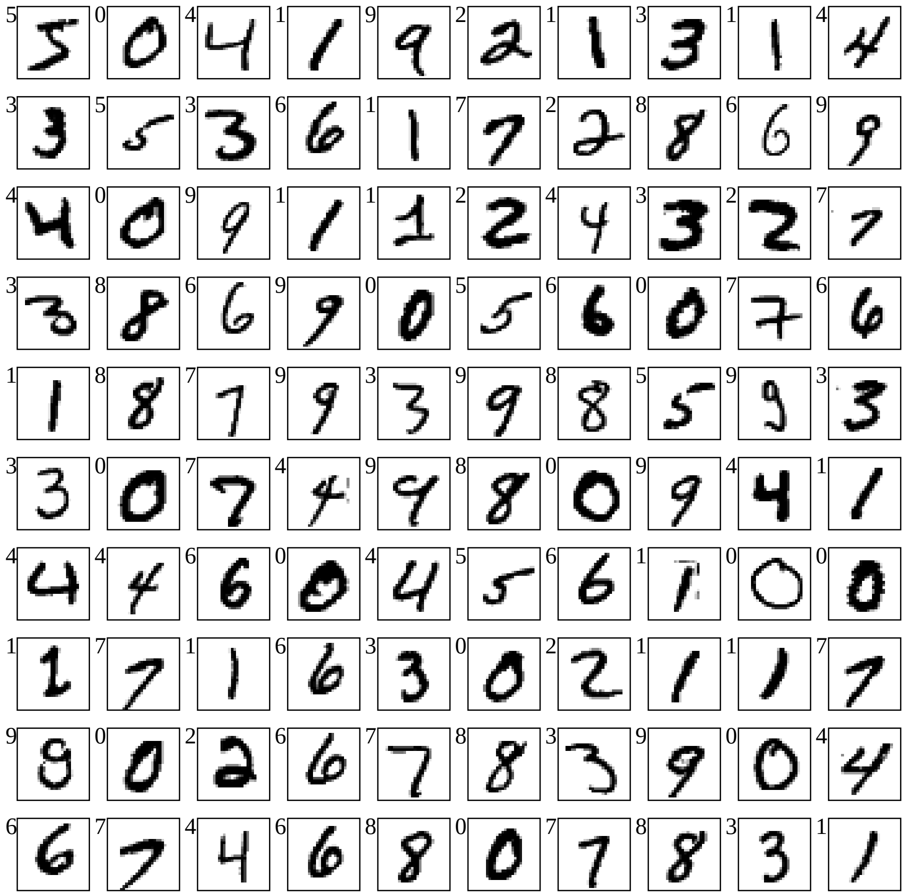

Figure 3. The first 100 MNIST training images, with class labels.

images were first size normalized to fit into a 20×20 pixel box, and then centered in a 28×28 image (LeCun et al., 1998).

Test results on the MNIST database which are given in the literature (e.g. (LeCun et al., 1995; LeCun et al., 1998)) commonly do not use the full MNIST test set of 60000 characters. Instead, a subset of 10000 characters is used, consisting of the test set patterns from 24476 to 34475. To obtain results which can be compared to the literature, we also use this test set, although the larger one would be preferable from the point of view of obtaining more reliable test error estimates. The MNIST benchmark data is available from http://www.research.att.com/~yann/exdb/mnist/index.html.

5.1.0.1. *Virtual SV Method — USPS Database* The first set of experiments was conducted on the USPS database. This database has been used extensively in the literature, with a LeNet1 Convolutional

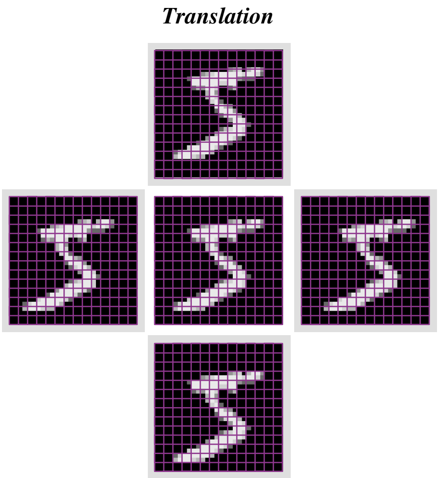

Figure 4. Different invariance transformations in the case of handwritten digit recognition. In all three cases, the central pattern, taken from the MNIST database, is the original which is transformed into "virtual examples" (marked by grey frames) with the same class membership by applying small transformations.

Table I. Comparison of Support Vector sets and performance for training on the original database and training on the generated Virtual Support Vectors. In both training runs, we used a polynomial classifier of degree 3. Virtual Support Vectors were generated by simply shifting the images by one pixel in the four principal directions. Adding the unchanged Support Vectors, this leads to a training set of the second classifier which has five times the size of the first classifier's overall Support Vector set (i.e. the union of the 10 Support Vector sets of the binary classifiers, of size 1677 — note that due to some overlap, this is smaller than the sum of the ten support set sizes). Note that training on virtual patterns generated from *all* training examples does not lead to better results than in the Virtual SV case; moreover, although the training set in this case is much larger, it hardly leads to more SVs.

classifier trained on	size	av. no. of SVs	test error
full training set	7291	274	4.0%
overall SV set	1677	268	4.1%
Virtual SV set	8385	686	3.2%
Virtual patterns from full DB	36455	719	3.4%

Network achieving a test error rate of 5.0% (LeCun et al., 1989). We used the regularization constant $C = 10$. This value was taken from earlier work (Schölkopf et al., 1995), no additional model selection was performed.

Virtual Support Vectors were generated for the set of all different Support Vectors of the ten classifiers. Alternatively, one can carry out

Table II. Summary of results on the USPS set. Note that two variants of this database have been used in the literature; one of them (denoted by USPS$^+$) has been enhanced by a set of machine-printed characters which have been found to improve the test error. Note that the virtual SV systems perform best out of all systems trained on the original USPS set.

Classifier	train set	test err	reference
nearest-neighbor	USPS$^+$	5.9%	(Simard et al., 1993)
LeNet1	USPS$^+$	5.0%	(LeCun et al., 1989)
Optimal margin classifier	USPS	4.6%	(Boser et al., 1992)
SVM	USPS	4.0%	(Schölkopf et al., 1995)
Linear Hyperplane on KPCA features	USPS	4.0%	(Schölkopf et al., 1998b)
Local learning	USPS$^+$	3.3%	(Bottou and Vapnik, 1992)
Virtual SVM	USPS	3.2%	(Schölkopf et al., 1996)
Virtual SVM, local kernel	USPS	3.0%	(Schölkopf, 1997)
Boosted Neural Nets	USPS$^+$	2.6%	(Drucker et al., 1993)
Tangent distance	USPS$^+$	2.6%	(Simard et al., 1993)
Human error rate	—	2.5%	(Bromley and Säckinger, 1991)

the procedure separately for the ten binary classifiers, thus dealing with smaller training sets during the training of the second machine. Table I shows that incorporating only translational invariance already improves performance significantly, from 4.0% to 3.2% error rate. For other types of invariances (Figure 4), we also found improvements, albeit smaller ones: generating Virtual Support Vectors by rotation or by the line thickness transformation of Drucker et al. (1993), we constructed polynomial classifiers with 3.7% error rate (in both cases). For purposes of comparison, we have summarized the main results on the USPS database in Table II.

Note, moreover, that generating Virtual examples from the full database rather than just from the SV sets did not improve the accuracy, nor did it enlarge the SV set of the final classifier substantially. This finding was confirmed using Gaussian RBF kernels (Schölkopf, 1997): in that case, similar to Table I, generating Virtual examples from the full database led to identical performance, and only slightly increased SV set size. From this, we conclude that for the considered recognition task, it is sufficient to generate Virtual examples only from the SVs — Virtual examples generated from the other patterns do not add much useful information.

5.1.0.2. *Virtual SV Method — MNIST Database.* The larger a database, the more information about invariances of the decision function is already contained in the differences between patterns of the same class.

Table III. Summary of results on the MNIST set. At 0.6% (0.56% before rounding), the system described in Section 5.1.1 performs best.

Classifier	test err. (60k)	test err. (10k)	reference
3-nearest-neighbor	—	2.4%	(LeCun et al., 1998)
2-layer MLP	—	1.6%	(LeCun et al., 1998)
SVM	1.6%	1.4%	(Schölkopf, 1997)
Tangent distance	—	1.1%	(Simard et al., 1993) (LeCun et al., 1998)
LeNet4	—	1.1%	(LeCun et al., 1998)
LeNet4, local learning	—	1.1%	(LeCun et al., 1998)
Virtual SVM	1.0%	0.8%	(Schölkopf, 1997)
LeNet5	—	0.8%	(LeCun et al., 1998)
dual-channel vision model	—	0.7%	(Teow and Loe, 2000)
Boosted LeNet4	—	0.7%	(LeCun et al., 1998)
Virtual SVM, 2-pixel translation	—	0.6%	*this paper; see Section 5.1.1*

To show that it is nevertheless possible to improve classification accuracies with our technique, we applied the method to the MNIST database of 60000 handwritten digits. This database has become the standard for performance comparisons at AT&T and Bell Labs.

Using Virtual Support Vectors generated by 1-pixel translations, we improved a degree 5 polynomial SV classifier from 1.4% to 1.0% error rate on the 10000 element test set. In this case, we applied our technique separately for all ten Support Vector sets of the binary classifiers (rather than for their union) in order to avoid having to deal with large training sets in the retraining stage. Note, moreover, that for the MNIST database, we did not compare results of the VSV technique to those for generating Virtual examples from the whole database: the latter is computationally exceedingly expensive, as it entails training on a very large training set. [7]

After retraining, the number of SVs more than doubled. Thus, although the training sets for the second set of binary classifiers were substantially smaller than the original database (for four Virtual SVs per SV, four times the size of the original SV sets, in our case amounting to around 10^4), we concluded that the amount of data in the region

[7] We did, however, make a comparison for a subset of the MNIST database, utilizing only the first 5000 training example. There, a degree 5 polynomial classifier was improved from 3.8% to 2.5% error by the Virtual SV method, with an increase of the average SV set sizes from 324 to 823. By generating Virtual examples from the full training set, and retraining on these, we obtained a system which had slightly more SVs (939), but an unchanged error rate.

of interest, close to the decision boundary, had more than doubled. Therefore, we reasoned that it should be possible to use a more complex decision function in the second stage (note that typical VC risk bounds depends on the *ratio* of VC-dimension and training set size (Vapnik, 1995)). Indeed, using a degree 9 polynomial led to an error rate of 0.8%, very close to the record performance of 0.7%. The main results on the MNIST set are summarized in Table III. Prior to the present work, the best system on the MNIST set was a boosted ensemble of LeNet4 neural networks, trained on a huge database of artificially generated virtual examples. Note that a difference in error rates on the MNIST set which is at least 0.1% may be considered significant (LeCun et al., 1998).

It should be noted that while it is much slower in training, the LeNet4 ensemble also has the advantage of a faster runtime speed. Especially when the number of SVs is large, SVMs tend to be slower at runtime than neural networks of comparable capacity. This is particularly so for virtual SV systems, which work by increasing the number of SV. Hence, if runtime speed is an issue, the systems have to be sped up. Burges and Schölkopf (1997) have shown that one can reduce the number of SVs to about 10% of the original number with very minor losses in accuracy. In a study on the MNIST set, they started with a VSV system performing at 1.0% test error rate, and sped it up by a factor of 10 with the accuracy degrading only slightly to 1.1%.[8] Note that there are neural nets which are still faster than that (cf., LeCun et al. (1998)).

5.1.1. *SMO VSV experiments*

In this section we summarize our newest results on the MNIST data set.

Following the last section, we used a polynomial kernel of degree 9. We normalized so that dot-products giving values within [0,1] yield kernel values within [0,1]; specifically:

$$K(u,v) \equiv \frac{1}{512}(u \cdot v + 1)^9.$$

This ensures that kernel values of 1 and 0 have the same sort of canonical meaning that holds for others, such as radial-basis function (RBF) kernels. Namely, a kernel value of 1 corresponds to the minimum distance between (identical) examples in the kernel-induced metric distance and 0 corresponds to the maximum distance.

[8] Note this study was done before VSV systems with higher accuracy were obtained.

We ensured any dot-product was within [0,1] by normalizing each example by their 2-norm scalar value (i.e. such that each example dot-producted against itself gives a value of 1). We used this normalization (a form of brightness or "amount of ink" equalization) by default because it is the sort of normalization that we routinely use in our NASA space image detection applications. This also appears to be a good normalization for the MNIST domain.

Since our polynomial kernel value normalization gives a kernel value of 1 special significance, we suspected that a SVM regularization parameter setting of C=1 would probably be too low. We also determined, by trying a large value (C=10) for training a binary recognizer for digit "8", that no training example reached an alpha value above 7.0. By looking at the histogram of the 60,000 alpha values for that case, we realized that only a handful of examples in each of the 10 digit classes had alpha values above 2.0. Under the assumption that only a few training examples in each class are particularly noisy and that digit "8" is one of the harder digits to recognize, for simplicity we used C=2 for training each SVM. We have not yet attempted to find better C values, and it is most likely that our simplistic choice was relatively suboptimal. In future work, we will determine the effect of more careful selection of C, such as via cross-validation and approximations based on generalization error upper-bound estimates.

Experiment 1: 1-Pixel Translations

Tables IV, V, and VI summarize our VSV results on the MNIST data sets using 1-pixel translations in all 8 directions.

These experiments employed our new SMO-based methods, as described in Section 4 (including our digestion technique). To see whether our faster training times could be put to good use, we tried the VSV method with more invariance than was practical in previous experiments. Specifically, we used "3x3 box jitter" of the image centers, which corresponds to translation of each image for a distance of one pixel in any of the 8 directions (horizontal, vertical, or both). Total training time, for obtaining 10 binary recognizers for each of the base SV and the VSV stages, was about 2 days (50 hours) on an Sun Ultra60 workstation with a 450Mhz processor and 2 Gigabytes of RAM (allowing an 800Mb kernel cache). This training time compares very favorably to other recently published results, such as learning just a single binary recognizer even without VSV (e.g. training digit "8" took about 8 hours in (Platt, 1999)) [9]. The VSV stage is also significantly

[9] The difference in CPU's (i.e. our Sparc Ultra-II 450Mhz versus Platt's Intel Pentium II 266Mhz) seems not as significant as one might imagine. Experiments on

Table IV. Errors on MNIST (10,000 test examples), using VSV with 1-pixel translation.

Misclassifications

	digit misclassifications										10-class
digit	0	1	2	3	4	5	6	7	8	9	test error rate
SV	6	8	15	17	13	10	8	15	9	21	1.22%
VSV	2	3	7	7	7	8	7	8	7	12	0.68%

Table V. Binary-Recognition Errors on MNIST (10,000 test examples), using VSV with 1-pixel translation.

Errors for each binary recognizer

SVM for digit	0	1	2	3	4	5	6	7	8	9
SV										
false negatives	9	12	22	21	18	22	20	33	27	40
false positives	6	3	11	6	8	4	5	11	9	14
VSV										
false negatives	5	6	11	11	6	12	12	16	14	21
false positives	4	4	9	5	5	3	7	10	7	8

more expensive that the base SV stage (averaging about 4 hours versus 1 hour) — a majority of examples given to VSV training typically end up being support vectors.

Table VI. Number of support vectors for MNIST (60,000 training examples), using VSV with 1-pixel translation.

Number of support vectors for each binary recognizer

digit	0	1	2	3	4	5	6	7	8	9
SV	1955	1354	3784	4157	3231	3977	2352	3090	4396	4130
$0<\alpha_i<C$	1848	1210	3630	3915	3051	3791	2237	2886	4060	3730
$\alpha_i = C$	107	144	154	242	180	186	115	204	336	400
VSV	8294	5439	15186	16697	12575	15858	9587	13165	18749	18215
$0<\alpha_i<C$	7907	4724	14648	15794	11784	15124	9081	12133	17315	16322
$\alpha_i = C$	387	715	538	903	791	734	506	1032	1434	1893

Experiment 2: Deslanted Images and Additional Translations

Some previous work on the MNIST data have achieved their best results by *deslanting* each image before training and testing. Deslanting

both CPU's indicate that the Sparc is effectively only 50% faster than the Pentium in dot-product computations, which dominate the training time in our experiments.

Table VII. Errors on deslanted MNIST (10,000 test examples), using VSV with 2-pixel translation.

Misclassifications

	digit misclassifications										10-class
digit	0	1	2	3	4	5	6	7	8	9	test error rate
SV	5	5	14	12	13	10	13	13	12	25	1.22%
VSV	3	4	6	4	8	7	8	7	8	13	0.68%
VSV2	3	3	5	3	6	7	7	6	5	11	**0.56%**

Table VIII. Binary-Recognition Errors on deslanted MNIST (10,000 test examples), using VSV with 2-pixel translation.

Errors for each binary recognizer

SVM for digit	0	1	2	3	4	5	6	7	8	9
SV										
false negatives	9	11	21	22	15	21	22	24	21	47
false positives	7	6	7	5	8	6	9	9	10	8
VSV										
false negatives	5	5	13	11	5	9	12	13	13	21
false positives	3	6	11	4	9	3	11	6	3	7
VSV2										
false negatives	5	5	12	11	4	9	13	12	11	18
false positives	4	5	9	4	9	4	11	9	3	7

Table IX. Number of support vectors for deslanted MNIST (60,000 training examples), using VSV with 2-pixel translation.

Number of support vectors for each binary recognizer

digit	0	1	2	3	4	5	6	7	8	9
SV	1859	1057	3186	3315	2989	3311	2039	2674	3614	3565
$0 < \alpha_i < C$	1758	927	3023	3090	2830	3124	1934	2475	3301	3202
$\alpha_i = C$	101	130	163	225	159	187	105	199	313	363
VSV	8041	4534	13462	13628	12115	13395	8558	11731	15727	16341
$0 < \alpha_i < C$	7619	3762	12837	12609	11146	12584	8031	10538	14181	14332
$\alpha_i = C$	422	772	625	1019	969	811	527	1193	1546	2009
VSV2	11305	6298	18549	18680	16800	18550	11910	16489	21826	23003
$0 < \alpha_i < C$	10650	5117	17539	17141	15329	17294	11084	14645	19382	19898
$\alpha_i = C$	655	1181	1010	1539	1471	1256	826	1844	2444	3105

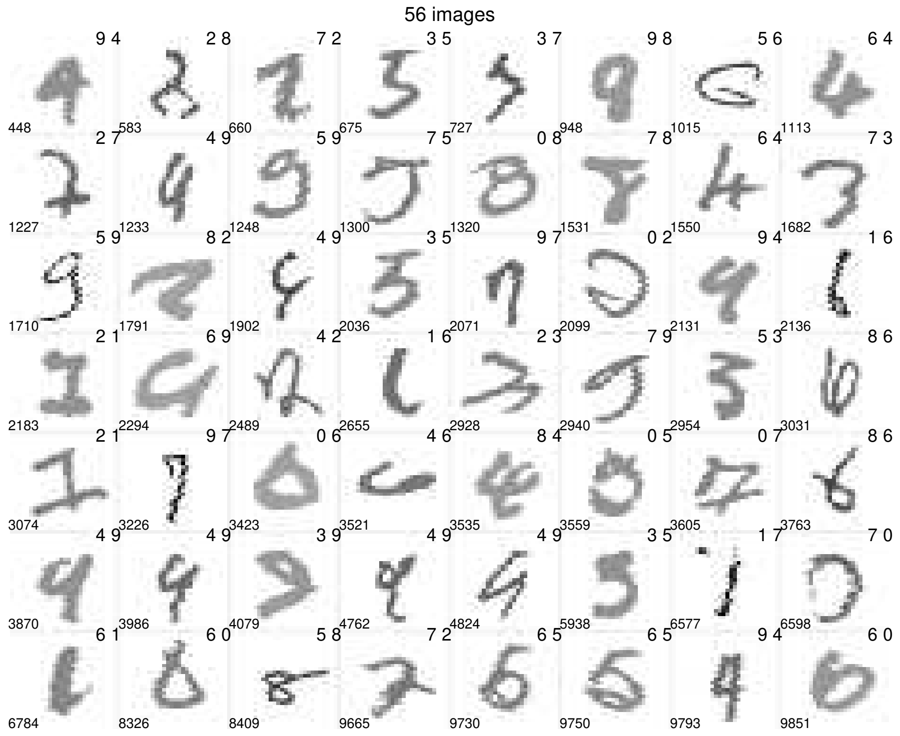

Figure 5. The 56 errors for deslanted 10,000 MNIST test data (VSV2). The number in the lower-left corner of each image box indicates the test example number (1 thru 10,000). The first number in the upper-right corner indicates the predicted digit label and the second indicates the true label.

is performed by computing each image's principal axis and horizontally translating each row of pixels to best approximate the process of rotating each image to make its principal axis become vertical. For example, under such deslanting, tilted "1" digits all become very similar and all close to vertical.

Tables VII, VIII, and IX summarize our newest results, using *deslanted* versions of the MNIST training and test sets. Other than using deslanted data, the same training conditions were used as the previous experiments (i.e. same kernel, same C=2, same 3x3 box jitter for VSV).

Interestingly, using deslanted images did not improve the test error rates for our experiments. In fact, the same test error rate was reached using either version of the data set for either SV (122 errors) or VSV (68 errors), although the specific test examples with errors varied somewhat. This result seems to provide evidence that the polynomial kernel is already doing a good job in capturing some of the rotational invariance from the training set alone. Nevertheless, comparing Table VI with

Table IX shows that deslanted data does lead to significantly fewer SVs and VSVs, with a particularly high percentage of reduction for digit "1" (as one might expect, since deslanted "1" digits are especially similar to each other).

To investigate whether further jitters would lead to even better results (or perhaps worse ones), we tried the VSV method using 3x3 box jitter combined with four additional translations by 2-pixels (i.e. horizontally or vertically, but not both), using the same training conditions as the other experiments. Although this only enlarged the number of training examples by about 50% (from 9 to 13 per original SV), it require approximately 4 times more training time, due to the large number of VSVs that resulted.

The results of this experiment are labelled as "VSV2" in Tables VII, VIII, and IX. Figure 5 shows the 56 misclassified test examples that resulted. It is interesting to note that the largest number of VSVs (23,003) is still only about a third of the size of the full (60,000) training set, despite the large number of translation jitters explored in this case.

Other distortions, such as scale, rotation, and line thickness, have all been reported to also help significantly in this domain. It seems clear from Figure 5 that many of the remaining misclassifications are due to failure to fully account for such other invariances.

5.1.2. *SMO Jittering Kernels Experiments*

Table X summarizes some initial experiments in comparing VSV and JSV methods on a small subset of the MNIST training set. Specifically, 200 "3" digit examples and 200 "8" digit examples from the training set and all 1984 "3" and "8" digits from the 10,000 example test set. We have not yet run more conclusion comparisons using the entire data set. We again used a polynomial kernel of degree 9.

These experiments illustrate typical relative behaviors, such as the JSV test times being much faster than the worst case (of J times slower than VSV test times), even though JSV must jitter at test time, due to JSV having many fewer final SVs than for VSV. Furthermore, both JSV and VSV typically beat standard SVMs (i.e. no invariance). They also both typically beat *query jitter* as well, in which the test examples are jittered inside the kernels during SVM output computations. Query jitter effectively uses jittering kernels at test time, even though the SVM was not specifically trained for a jittering kernel. We tested that case simply as a control in such experiments, to verify that training the SVM with the actual jittering kernel used at test time is indeed important.

Relative test errors between VSV and JSV vary — sometimes VSV is substantially better (as in the 5x5 box jitter example) and sometime

Table X. "3 vs 8" results for MNIST (all 1984 test examples, first 200 3's and 200 8's from training set).

Using 3x3 box jitter

method	test errors	SV / N	task	time (secs)
base		207 / 400	train	2.3
	61 / 1984		test	11.0
	50 / 1984		query jitter test	119.4
VSV		1050 / 1863	train	51.7
	34 / 1984		test	60.5
JSV		171 / 400	train	18.8
	30 / 1984		test	98.6

Using 5x5 box jitter and deslant preprocessing (principal axis vertical)

method	test errors	SV / N	task	time (secs)
base		179 / 400	train	1.8
	33 / 1984		test	9.9
	50 / 1984		query jitter test	480.5
VSV		2411/ 4475	train	385.0
	8 / 1984		test	>400
JSV		151 / 400	train	31.7
	28 / 1984		test	396.0

it is somewhat worse. Although our results to date are preliminary, it does seem that VSV methods are substantially more robust than JSV ones, in terms of variance in generalization errors.

5.2. NASA Volcanoe Recognition

We are currently applying these invariance approaches to several difficult NASA object detection tasks. In particular, we are focusing on improving known results in volcanoe detection (Burl et al., 1998) and in enabling useful initial results for a crater detection application (Burl, 2001). These NASA applications have in fact motivated much of our recent work on efficient training of invariant SVMs.

In this section we present some preliminary results on a subset of the NASA volcanoe data, specifically that which is currently publicly available (the UCI KDD Archive "volcanoes" data (Burl, 2000)). Figure 6 shows some examples from this data set. As described in Burl et al. (1998), all examples were determined from a focus of attention (FOA) matched filter that was designed to quickly scan large images and select 15-pixel-by-15-pixel subimages with relatively high false alarm rates but low miss rates (about 20:1).

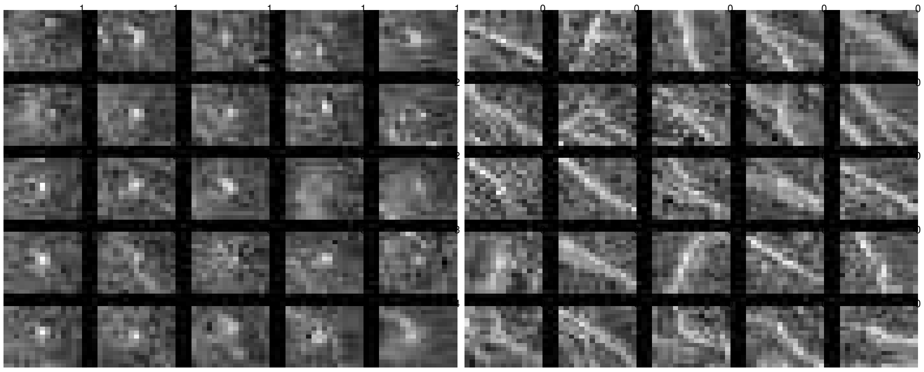

Figure 6. First 25 positive (left) and first 25 negative (right) test examples of volcanoe data set `HOM4` (partition A1) (Burl, 2000).

The previous best overall results on this data were achieved using a quadratic classifier, modeling the two classes with one Gaussian for each class (Burl et al., 1998). To overcome the high dimensionality of the 225-pixel example vectors, PCA was performed to find the six principal axes of only the positive examples and then all examples were projected to those axes. Mean and covariance matrices were computed over those six dimensions for each class.

Figures 7 and 8 compare ROC [10] curves for our initial SVMs (without any explicit invariance) versus the best previous Gaussian models. [11] There are five experiments, varying from homogeneous collections of volcanoes from the same regions of Venus to heterogeneous collections of examples from across the planet, with various k-fold partitionings into training and test sets for each experiment. The ROC curves represent the leave-out-fold cross-validation performances for each experiment.

The number of example images vary from thousands to tens of thousands across the five experiments. Due to the much greater number of negative examples than positive examples in each experiment (by factors ranging from 10 to 20), we trained SVMs used SVM^{light}, employing its implemented ability to use two regularization parameters, C^- and C^+, instead of a single C. That was done to reflect the importance of a high detection rate despite the relatively scarcity of positive

[10] As in Burl et al. (1998), these are actually *FROC* (Free-response ROC) curves, showing probability of detection versus the number of false alarms per unit area.

[11] The volcanoe data from the UCI KDD Archive (Burl, 2000) are slightly different from the data used in Burl et al. (1998), due to recent changes in their FOA mechanism. The author of those experiments (Burl) reran them for us, so that all our ROC plots here involve the same data. This accounts for our slight differences from the ROC curves shown in Burl et al. (1998).

training examples. Specifically, we used $C^- = 1$ and $C^+ = 3C^-$ and the polynomial kernel $K(u, v) \equiv \frac{1}{512}(u \cdot v + 1)^9$, based solely on some manual tuning on data set `HOM4` (experiment A). Burl et al. (1998) similarly used `HOM4` to determine settings for the remaining four experiments. It is interesting to note that `HOM4` is the only one of the experiments for which our (SVM) results seem to be very similar to theirs. We also adopted the same normalization (of the 8-bit pixel values) used in their original experiments, namely so that the mean pixel value of each example image is 0 and the standard deviation is 1.

To extract ROC curves from each trained SVM, we added a sliding offset value to the SVM output, with an offset of 0 giving the trained response. Interesting, our SVMs beat the previous best (Gaussian) model for each of the five experiments when offset=0, i.e. at the operating point for which the SVM was specifically trained. This suggests that the SVM-derived ROC curves might be even higher overall if a SVM was explicitly trained for multiple operating points, instead of just one. It is also interesting that the SVM approach does well without the sort of feature extraction effort (i.e. PCA) that the previous work had required. One disadvantage of our SVM approach was that training time for all five experiments was somewhat slower — about 15 minutes, versus 3 minutes for the Gaussian model, on the same machine (a 450Mhz Sun Ultra60).

This NASA volcanoe detection problem is somewhat challenging for VSV techniques to make further progress, since many potential invariances have already been normalized away. For example, the Sun light source is from the same angle and intensity for each example considered, so rotation transformations would not be useful. Furthermore, the focus of attention mechanism already centers each volcanoe example quite well. Nevertheless, the volcanoe examples (e.g. Figure 6) do indicate some slight variance in the location of the characteristic bright middle of each volcanoe, so we have been investigating whether VSV translations can still help.

Due to the relatively large number of negative examples, we first explored whether performing VSV transformations only on the positive examples might suffice. This was inspired by the previous work succeeding with PCA limited to capturing the variance only in the positive examples. Furthermore, it seemed possible that the large set of negative examples might already contain considerable translation variation. However, we found that this lead to significantly worse performance than using no VSV. The standard VSV approach of transforming both negative and positive SVs indeed seems critical, perhaps because otherwise the new positive instances have free reign to claim large

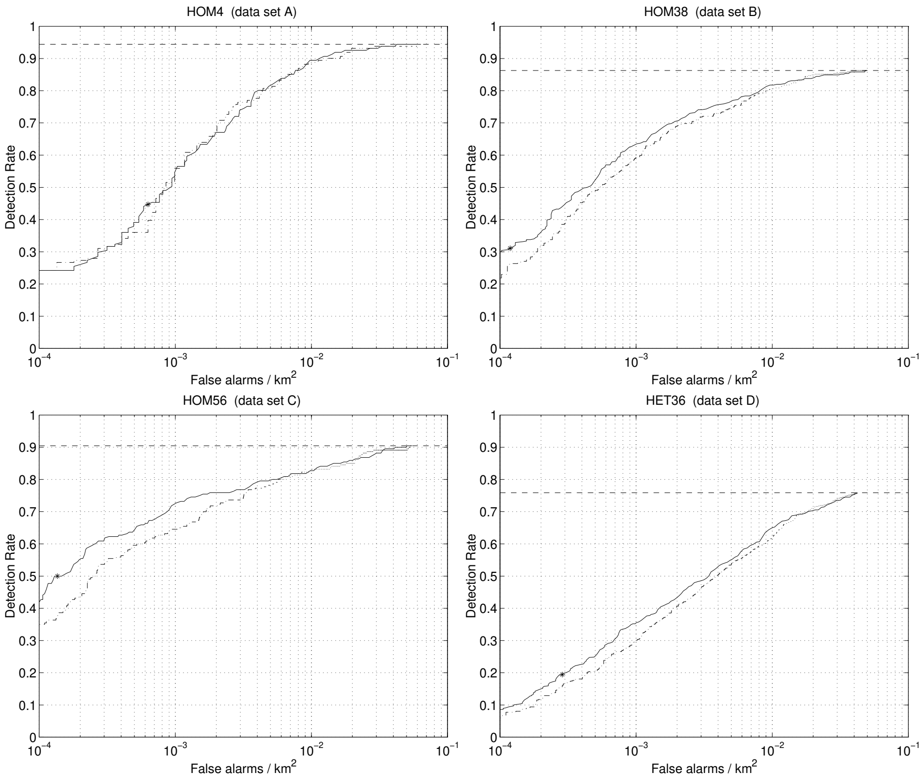

Figure 7. ROC curves for the first four experiments described by Burl et al. (1998). The solid curve is the ROC for our SVM and the dashed curve is the ROC for the best two-Gaussian model from Burl et al. (1998). The mark "*" on the SVM curve indicates the operating point of the trained SVM (i.e. offset=0). The dashed line near the top of each plot box indicates the best performance possible, which is less than a detection rate of 1.0 due to the focus of attention mechanism having a non-zero miss rate.

areas of kernel feature space for which it (incorrectly) appears negative instances would not occur.

Figure 9 shows a comparison of ROC curves for a baseline SVM and its VSV SVM, using four 1-pixel translations (horizontal or vertical) for all SVs of the baseline SVM and the same training conditions as above. However, unlike for the MNIST domain, in this domain replacing a pixel which moves away from an edge is not as simple as shifting in a constant background (e.g. zero) pixel. For simplicity, we first tried replicating the shifted row (or column), but found this did not work well, presumably because this conflicted with the higher-order correlations modeled by the nonlinear kernel. So, instead for each image we did the following: 1) apply the translation to each raw 8-bit-pixel image (without concern for shift-in values), 2) shrink from 15-by-15 to 13-by-13 images (i.e. ignore

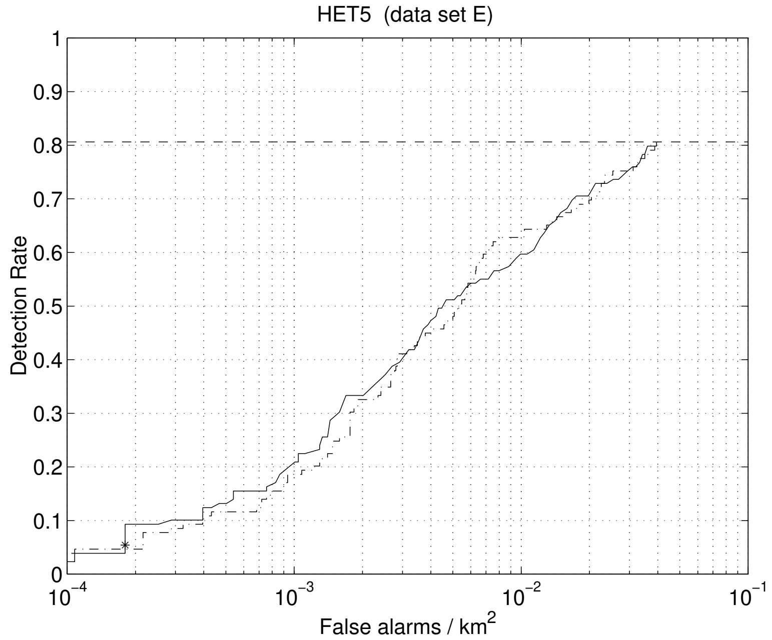

Figure 8. ROC curves for the fifth experiment (as in Figure 7).

the borders), 3) normalize each 13x13 example. This shrinking was done for the baseline SVM as well, which apparently did not influence the ROC curve significantly (as comparison of `HOM4` in Figures 7 and 9 indicates).

Histograms of the SVM outputs suggest that a key reason for the overall improvements in the ROC curve for the VSV SVM is that it is significantly more confident in its outputs (e.g. they are much larger in magnitude), especially for negative examples.

For comparison, we have also tried translating all examples (not just SVs). As in earlier work, we found the results to be essentially identical, although much slower to train.

Our work with VSV on the volcanoe data is still preliminary, so we are still focussing on the `HOM4` data and remaining blind to VSV performance on the others. To date we have observed that VSV does not seem to hurt overall ROC performance, as long as the above warnings are heeded, and sometimes helps. We suspect that with significant further work (e.g. employing scale transforms, starting with larger 17-by-17 images to shrink after translations, using more comprehensive model selection, etc.) significantly better results can still be achieved in this domain.

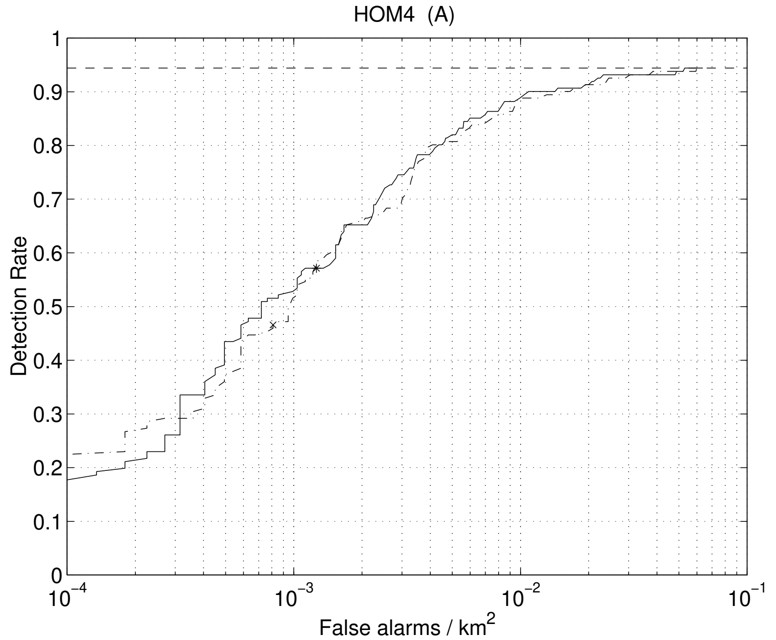

Figure 9. Dashed ROC curve is from baseline SVM (mark "x" shows threshold=0 operating point). Solid ROC curve is from SVM including VSV with four 1-pixel translations (mark "*" shows threshold=0 operating point).

6. Conclusion

This paper has described a variety of methods for incorporating prior knowledge about invariances into SVMs. In experimental work, we have demonstrated that these techniques allow state-of-the-art performance, both in terms of generalization error and in terms of SVM training time.

As mentioned throughout this paper, there are several promising lines of future work. These include experiments with a wider assortment of distortions (e.g. rotations, scale, line thickness) and across multiple domains (e.g. further NASA space applications, in addition to the traditional digit recognition tasks).

An interesting issue is under what conditions applying additional distortions to training data leads to significantly worse test errors, even when using careful cross-validation during training. One would expect that obviously-excessive distortions, such as rotating "6" digits so far as to confuse them with "9" digits, would be easily detected as counter-productive during such cross-validations. Thus, an open issue is to what extent distortions reflecting known invariances can be safely applied during training whenever computationally feasible and to what extent their use makes controlling generalization error more difficult.

Given our demonstrated success in training invariant SVMs that generalize very well, a key issue for future work is in lowering their test time costs. For example, the best neural networks (e.g. (LeCun et al., 1998)) still appear to be much faster at test time than our best SVMs, due to the large number of (virtual) support examples that are each compared to each test example during kernel computations. Further work in reducing the number of such comparisons per test example (with minimal increase in test errors), perhaps along the lines of reduced sets (Burges and Schölkopf, 1997), would seem particularly useful at this point.

Acknowledgements

Thanks to Chris Burges, Michael C. Burl, Olivier Chapelle, Sebastian Mika, Patrice Simard, Alex Smola, and Vladimir Vapnik for useful discussions.

Parts of this research were carried out by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

Baird, H. (1990). Document image defect models. In *Proceedings, IAPR Workshop on Syntactic and Structural Pattern Recognition*, pages 38 – 46, Murray Hill, NJ.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In Haussler, D., editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA. ACM Press.

Bottou, L. and Vapnik, V. N. (1992). Local learning algorithms. *Neural Computation*, 4(6):888–900.

Bromley, J. and Säckinger, E. (1991). Neural-network and k-nearest-neighbor classifiers. Technical Report 11359–910819–16TM, AT&T.

Burges, C. J. C. (1999). Geometry and invariance in kernel based methods. In Schölkopf, B., Burges, C. J. C., and Smola, A. J., editors, *Advances in Kernel Methods — Support Vector Learning*, pages 89–116, Cambridge, MA. MIT Press.

Burges, C. J. C. and Schölkopf, B. (1997). Improving the accuracy and speed of support vector learning machines. In Mozer, M., Jordan, M., and Petsche, T., editors, *Advances in Neural Information Processing Systems 9*, pages 375–381, Cambridge, MA. MIT Press.

Burl, M. C. (2000). NASA volcanoe data set at UCI KDD Archive. (See http://kdd.ics.uci.edu/databases/volcanoes/volcanoes.html).

Burl, M. C. (2001). Mining large image collections: Architecture and algorithms. In Grossman, R., Kamath, C., Kumar, V., and Namburu, R., editors, *Data Mining for Scientific and Engineering Applications*, Series in Massive Computing, Cambridge, MA. Kluwer Academic Publishers.

Burl, M. C., Asker, L., Smyth, P., Fayyad, U., Perona, P., Crumpler, L., and Aubele, J. (1998). Learning to recognize volcanoes on Venus. *Machine Learning*, 30:165–194.

Chapelle, O. and Schölkopf, B. (2000). Incorporating invariances in nonlinear SVMs. *in preparation*. Presented at the NIPS2000 workshop on Learning with Kernels.

Cortes, C. and Vapnik, V. (1995). Support vector networks. *Machine Learning*, 20:273 – 297.

Crampin, M. and Pirani, F. A. E. (1986). *Applicable Differential Geometry*. Cambridge University Press, Cambridge, UK.

DeCoste, D. and Burl, M. C. (2000). Distortion-invariant recognition via jittered queries. In *Computer Vision and Pattern Recognition (CVPR-2000)*.

DeCoste, D. and Wagstaff, K. (2000). Alpha seeding for support vector machines. In *International Conference on Knowledge Discovery and Data Mining (KDD-2000)*.

Drucker, H., Schapire, R., and Simard, P. (1993). Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4):705 – 719.

Girosi, F. (1998). An equivalence between sparse approximation and support vector machines. *Neural Computation*, 10(6):1455–1480.

Haussler, D. (1999). Convolutional kernels on discrete structures. Technical Report UCSC-CRL-99-10, Computer Science Department, University of California at Santa Cruz.

Jaakkola, T. S. and Haussler, D. (1999). Exploiting generative models in discriminative classifiers. In Kearns, M. S., Solla, S. A., and Cohn, D. A., editors, *Advances in Neural Information Processing Systems 11*, Cambridge, MA. MIT Press.

Joachims, T. (1999). Making large-scale support vector machine learning practical. In *Advances in Kernel Methods: Support Vector Machines* (Schölkopf et al., 1999).

Keerthi, S., Shevade, S., Bhattacharyya, C., and Murthy, K. (1999). Improvements to Platt's SMO algorithm for SVM classifier design. Technical Report CD-99-14, Dept. of Mechanical and Production Engineering, National University of Singapore.

Kimeldorf, G. S. and Wahba, G. (1970). A correspondence between Bayesian estimation on stochastic processes and smoothing by splines. *Annals of Mathematical Statistics*, 41:495–502.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. J. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541 – 551.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278 – 2324.

LeCun, Y., Jackel, L. D., Bottou, L., Brunot, A., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Müller, U. A., Säckinger, E., Simard, P., and Vapnik, V. (1995). Comparison of learning algorithms for handwritten digit recognition. In Fogelman-Soulié, F. and Gallinari, P., editors, *Proceedings ICANN'95 — International Conference on Artificial Neural Networks*, volume II, pages 53 – 60, Nanterre, France. EC2.

Oliver, N., Schölkopf, B., and Smola, A. J. (2000). Natural regularization in SVMs. In Smola, A. J., Bartlett, P. L., Schölkopf, B., and Schuurmans, D., editors, *Advances in Large Margin Classifiers*, pages 51 – 60, Cambridge, MA. MIT Press.

Platt, J. (1999). Fast training of support vector machines using sequential minimal optimization. In Schölkopf, B., Burges, C. J. C., and Smola, A. J., editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA. MIT Press.

Poggio, T. and Girosi, F. (1989). A theory of networks for approximation and learning. Technical Report AIM-1140, Artificial Intelligence Laboratory, Massachusetts Institute of Technology (MIT), Cambridge, Massachusetts.

Poggio, T. and Vetter, T. (1992). Recognition and structure from one 2D model view: observations on prototypes, object classes and symmetries. A.I. Memo No. 1347, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Schölkopf, B. (1997). *Support Vector Learning*. R. Oldenbourg Verlag, München. Doktorarbeit, TU Berlin. Download: http://www.kernel-machines.org.

Schölkopf, B., Burges, C., and Smola, A. (1999). *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA.

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

Schölkopf, B., Burges, C., and Vapnik, V. (1996). Incorporating invariances in support vector learning machines. In von der Malsburg, C., von Seelen, W., Vorbrüggen, J. C., and Sendhoff, B., editors, *Artificial Neural Networks — ICANN'96*, pages 47 – 52, Berlin. Springer Lecture Notes in Computer Science, Vol. 1112.

Schölkopf, B., Simard, P., Smola, A., and Vapnik, V. (1998a). Prior knowledge in support vector kernels. In Jordan, M., Kearns, M., and Solla, S., editors, *Advances in Neural Information Processing Systems 10*, pages 640 – 646, Cambridge, MA. MIT Press.

Schölkopf, B., Smola, A., and Müller, K.-R. (1998b). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319.

Simard, P., LeCun, Y., and Denker, J. (1993). Efficient pattern recognition using a new transformation distance. In Hanson, S. J., Cowan, J. D., and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5. Proceedings of the 1992 Conference*, pages 50–58, San Mateo, CA. Morgan Kaufmann.

Simard, P., Victorri, B., LeCun, Y., and Denker, J. (1992). Tangent prop — a formalism for specifying selected invariances in an adaptive network. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*, San Mateo, CA. Morgan Kaufmann.

Smola, A., Schölkopf, B., and Müller, K.-R. (1998). The connection between regularization operators and support vector kernels. *Neural Networks*, 11:637–649.

Teow, L.-N. and Loe, K.-F. (2000). Handwritten digit recognition with a novel vision model that extracts linearly separable features. In *Computer Vision and Pattern Recognition (CVPR-2000)*.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer, NY.

Vapnik, V. (1998). *Statistical Learning Theory*. Wiley, NY.

Watkins, C. (2000). Dynamic alignment kernels. In Smola, A. J., Bartlett, P. L., Schölkopf, B., and Schuurmans, D., editors, *Advances in Large Margin Classifiers*, pages 39 – 50, Cambridge, MA. MIT Press.

Zien, A., Rätsch, G., Mika, S., Schölkopf, B., Lengauer, T., and Müller, K.-R. (2000). Engineering Support Vector Machine Kernels That Recognize Translation Initiation Sites. *Bioinformatics*, 16(9):799–807.