Journal of Machine Learning Research 7 (2006) 1493–1515 Submitted 10/05; Revised 3/06; Published 7/06

Building Support Vector Machines with Reduced Classifier Complexity

S. Sathiya Keerthi SELVARAK@YAHOO-INC.COM
Yahoo! Research
3333 Empire Avenue, Building 4
Burbank, CA 91504, USA

Olivier Chapelle CHAPELLE@TUEBINGEN.MPG.DE
MPI for Biological Cybernetics
72076 Tübingen, Germany

Dennis DeCoste DECOSTED@YAHOO-INC.COM
Yahoo! Research
3333 Empire Avenue, Building 4
Burbank, CA 91504, USA

Editors: Kristin P. Bennett and Emilio Parrado-Hernández

Abstract

Support vector machines (SVMs), though accurate, are not preferred in applications requiring great classification speed, due to the number of support vectors being large. To overcome this problem we devise a primal method with the following properties: (1) it decouples the idea of basis functions from the concept of support vectors; (2) it greedily finds a set of kernel basis functions of a specified maximum size ($d_{\max}$) to approximate the SVM primal cost function well; (3) it is efficient and roughly scales as $O(nd_{\max}^2)$ where n is the number of training examples; and, (4) the number of basis functions it requires to achieve an accuracy close to the SVM accuracy is usually far less than the number of SVM support vectors.

Keywords: SVMs, classification, sparse design

1. Introduction

Support Vector Machines (SVMs) are modern learning systems that deliver state-of-the-art performance in real world pattern recognition and data mining applications such as text categorization, hand-written character recognition, image classification and bioinformatics. Even though they yield very accurate solutions, they are not preferred in online applications where classification has to be done in great speed. This is due to the fact that a large set of basis functions is usually needed to form the SVM classifier, making it complex and expensive. In this paper we devise a method to overcome this problem. Our method incrementally finds basis functions to maximize accuracy. The process of adding new basis functions can be stopped when the classifier has reached some limiting level of complexity. In many cases, our method efficiently forms classifiers which have an order of magnitude smaller number of basis functions compared to the full SVM, while achieving nearly the same level of accuracy.

SVM solution and post-processing simplification Given a training set $\{(x_i, y_i)\}_{i=1}^n, y_i \in \{1, -1\}$, the Support Vector Machine (SVM) algorithm with an L_2 penalization of the training errors consists of solving the following primal problem

$$\min \frac{\lambda}{2}\|w\|^2 + \frac{1}{2}\sum_{i=1}^{n}\max(0, 1 - y_i w \cdot \phi(x_i))^2. \tag{1}$$

Computations involving ϕ are handled using the kernel function, $k(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$. For convenience the bias term has not been included, but the analysis presented in this paper can be extended in a straightforward way to include it. The quadratic penalization of the errors makes the primal objective function continuously differentiable. This is a great advantage and becomes necessary for developing a primal algorithm, as we will see below.

The standard way to train an SVM is to introduce Lagrange multipliers α_i and optimize them by solving a dual problem. The classifier function for a new input x is then given by the sign of $\sum_i \alpha_i y_i k(x, x_i)$. Because there is a flat part in the loss function, the vector α is usually sparse. The x_i for which $\alpha_i \neq 0$ are called *support vectors (SVs)*. Let n_{SV} denote the number of SVs for a given problem. A recent theoretical result by Steinwart (Steinwart, 2004) shows that n_{SV} grows as a linear function of n. Thus, for large problems, this number can be large and the training and testing complexities might become prohibitive since they are respectively, $O(n\, n_{\text{SV}} + {n_{\text{SV}}}^3)$ and $O(n_{\text{SV}})$.

Several methods have been proposed for reducing the number of support vectors. Burges and Schölkopf (1997) apply nonlinear optimization methods to seek sparse representations after building the SVM classifier. Along similar lines, Schölkopf et al. (1999) use L_1 regularization on β to obtain sparse approximations. These methods are expensive since they involve the solution of hard non-convex optimization problems. They also become impractical for large problems. Downs et al. (2001) give an exact algorithm to prune the support vector set after the SVM classifier is built. Thies and Weber (2004) give special ideas for the quadratic kernel. Since these methods operate as a post-processing step, an expensive standard SVM training is still required.

Direct simplification via basis functions and primal Instead of finding the SVM solution by maximizing the dual problem, one approach is to *directly minimize the primal form* after invoking the representer theorem to represent w as

$$w = \sum_{i=1}^{n} \beta_i \phi(x_i). \tag{2}$$

If we allow $\beta_i \neq 0$ for all i, substitute (2) in (1) and solve for the β_i's then (assuming uniqueness of solution) we will get $\beta_i = y_i \alpha_i$ and thus we will precisely retrieve the SVM solution (Chapelle, 2005). But our aim is to obtain approximate solutions that have as few non-zero β_i's as possible. For many classification problems there exists a small subset of the basis functions[1] suited to the complexity of the problem being solved, irrespective of the training size growth, that will yield pretty much the same accuracy as the SVM classifier. The evidence for this comes from the empirical performance of other sparse kernel classifiers: the Relevance Vector Machine (Tipping, 2001), Informative Vector Machine (Lawrence et al., 2003) are probabilistic models in a Bayesian setting; and Kernel Matching Pursuit (Vincent and Bengio, 2002) is a discriminative method that is mainly developed for the least squares loss function. These recent non-SVM works have laid the claim that they can match the accuracy of SVMs, while also bringing down considerably, the number of basis functions as well as the training cost. Work on simplifying SVM solution has not caught up well

1. Each $k(x, x_i)$ will be referred to as a basis function.

with those works in related kernel fields. The method outlined in this paper makes a contribution to fill this gap.

We deliberately use the variable name, β_i in (2) so as to interpret it as a basis weight as opposed to viewing it as $y_i\alpha_i$ where α_i is the Lagrange multiplier associated with the i-th primal slack constraint. While the two are (usually) one and the same at exact optimality, they can be very different when we talk of sub-optimal primal solutions. There is a lot of freedom when we simply think of the β_i's as basis weights that yield a good suboptimal w for (1). First, we do not have to put any bounds on the β_i. Second, we do not have to think of a β_i corresponding to a particular location relative to the margin planes to have a certain value. Going even one more step further, we do not even have to restrict the basis functions to be a subset of the training set examples.

Osuna and Girosi (1998) consider such an approach. They achieve sparsity by including the L_1 regularizer, $\lambda_1\|\beta\|_1$ in the primal objective. But they do not develop an algorithm (for solving the modified primal formulation and for choosing the right λ_1) that scales efficiently to large problems.

Wu et al. (2005) write w as

$$w = \sum_{i=1}^{l} \beta_i \phi(\tilde{x}_i)$$

where l is a chosen small number and optimize the primal objective with the β_i as well as the $\tilde{x}_i$ as variables. But the optimization can become unwieldy if l is not small, especially since the optimization of the $\tilde{x}_i$ is a hard non-convex problem.

In the RSVM algorithm (Lee and Mangasarian, 2001; Lin and Lin, 2003) a random subset of the training set is chosen to be the $\tilde{x}_i$ and then only the β_i are optimized.[2] Because basis functions are chosen randomly, this method requires many more basis functions than needed in order to achieve a level of accuracy close to the full SVM solution; see Section 3.

A principled alternative to RSVM is to use a greedy approach for the selection of the subset of the training set for forming the representation. Such an approach has been popular in Gaussian processes (Smola and Bartlett, 2001; Seeger et al., 2003; Keerthi and Chu, 2006). Greedy methods of basis selection also exist in the boosting literature (Friedman, 2001; Rätsch, 2001). These methods entail selection from a continuum of basis functions using either gradient descent or linear programming column generation. Bennett et al. (2002) and Bi et al. (2004) give modified ideas for kernel methods that employ a set of basis functions fixed at the training points.

Particularly relevant to the work in this paper are the kernel matching pursuit (KMP) algorithm of Vincent and Bengio (2002) and the growing support vector classifier (GSVC) algorithm of Parrado-Hernández et al. (2003). KMP is an effective greedy discriminative approach that is mainly developed for least squares problems. GSVC is an efficient method that is developed for SVMs and uses a heuristic criterion for greedy selection of basis functions.

Our approach *The main aim of this paper is to give an effective greedy method SVMs which uses a basis selection criterion that is directly related to the training cost function and is also very efficient.* The basic theme of the method is forward selection. It starts with an empty set of basis functions and greedily chooses new basis functions (from the training set) to improve the primal objective function. We develop efficient schemes for both, the greedy selection of a new basis function, as well as the optimization of the β_i for a given selection of basis functions. For choosing upto $d_{\max}$ basis functions, the overall compuational cost of our method is $O(nd_{\max}^2)$. The different

2. For convenience, in the RSVM method, the SVM regularizer is replaced by a simple L_2 regularizer on β.

	SpSVM-2		**SVM**	
Data Set	TestErate	#Basis	TestErate	n_{SV}
Banana	10.87 (1.74)	17.3 (7.3)	10.54 (0.68)	221.7 (66.98)
Breast	29.22 (2.11)	12.1 (5.6)	28.18 (3.00)	185.8 (16.44)
Diabetis	23.47 (1.36)	13.8 (5.6)	23.73 (1.24)	426.3 (26.91)
Flare	33.90 (1.10)	8.4 (1.2)	33.98 (1.26)	629.4 (29.43)
German	24.90 (1.50)	14.0 (7.3)	24.47 (1.97)	630.4 (22.48)
Heart	15.50 (1.10)	4.3 (2.6)	15.80 (2.20)	166.6 (8.75)
Ringnorm	1.97 (0.57)	12.9 (2.0)	1.68 (0.24)	334.9 (108.54)
Thyroid	5.47 (0.78)	10.6 (2.3)	4.93 (2.18)	57.80 (39.61)
Titanic	22.68 (1.88)	3.3 (0.9)	22.35 (0.67)	150.0 (0.0)
Twonorm	2.96 (0.82)	8.7 (3.7)	2.42 (0.24)	330.30 (137.02)
Waveform	10.66 (0.99)	14.4 (3.3)	10.04 (0.67)	246.9 (57.80)

Table 1: Comparison of *SpSVM-2* and *SVM* on benchmark data sets from (Rätsch). For TestErate, #Basis and n_{SV}, the values are means over ten different training/test splits and the values in parantheses are the standard deviations.

components of the method that we develop in this paper are not new in themselves and are inspired from the above mentioned papers. However, from a practical point of view, it is not obvious how to combine and tune them in order to get a very efficient SVM training algorithm. That is what we achieved in this paper through numerous and careful experiments that validated the techniques employed.

Table 1 gives a preview of the performance of our method (called *SpSVM-2* in the table) in comparison with SVM on several UCI data sets. As can be seen there, our method gives a competing generalization performance while reducing the number of basis functions very significantly. (More specifics concerning Table 1 will be discussed in Section 4.)

The paper is organized as follows. We discuss the details of the efficient optimization of the primal objective function in Section 2. The key issue of selecting basis functions is taken up in Section 3. Sections 4-7 discuss other important practical issues and give computational results that demonstrate the value of our method. Section 8 gives some concluding remarks. The appendix gives details of all the data sets used for the experiments in this paper.

2. The Basic Optimization

Let $J \subset \{1,\ldots,n\}$ be a given index set of basis functions that form a subset of the training set. We consider the problem of minimizing the objective function in (1) over the set of vectors w of the form[3]

$$w = \sum_{j \in J} \beta_j \phi(x_j). \tag{3}$$

3. More generally, one can consider expansion on points which do not belong to the training set.

2.1 Newton Optimization

Let $K_{ij} = k(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$ denote the generic element of the $n \times n$ kernel matrix K. The notation K_{IJ} refers to the submatrix of K made of the rows indexed by I and the columns indexed by J. Also, for a n-dimensional vector p, let p_J denote the $|J|$ dimensional vector containing $\{p_j : j \in J\}$.

Let $d = |J|$. With w restricted to (3), the primal problem (1) becomes the d dimensional minimization problem of finding β_J that solves

$$\min_{\beta_J} f(\beta_J) = \frac{\lambda}{2} \beta_J^\top K_{JJ} \beta_J + \frac{1}{2} \sum_{i=1}^{n} \max(0, 1 - y_i o_i)^2 \tag{4}$$

where $o_i = K_{i,J} \beta_J$. Except for the regularizer being more general, i.e., $\beta_J^\top K_{JJ} \beta_J$ (as opposed to the simple regularizer, $\|\beta_J\|^2$), the problem in (4) is very much the same as in a linear SVM design. Thus, the Newton method and its modification that are developed for linear SVMs (Mangasarian, 2002; Keerthi and DeCoste, 2005) can be used to solve (4) and obtain the solution β_J.

Newton Method

1. Choose a suitable starting vector, β_J^0. Set $k = 0$.
2. If β_J^k is the optimal solution of (4), stop.
3. Let $I = \{i : 1 - y_i o_i \geq 0\}$ where $o_i = K_{i,J} \beta_J^k$ is the output of the i-th example. Obtain $\bar{\beta}_J$ as the result of a Newton step or equivalently as the solution of the regularized least squares problem,
$$\min_{\beta_J} \frac{\lambda}{2} \beta_J^\top K_{JJ} \beta_J + \frac{1}{2} \sum_{i \in I} (1 - y_i K_{i,J} \beta_J)^2. \tag{5}$$
4. Take β_J^{k+1} to be the minimizer of f on L, the line joining β_J^k and $\bar{\beta}_J$. Set $k := k + 1$ and go back to step 2 for another iteration.

The solution of (5) is given by

$$\bar{\beta}_J = \beta_J^k - P^{-1} g, \quad \text{where} \quad P = \lambda K_{JJ} + K_{JI} K_{JI}^\top \quad \text{and} \quad g = \lambda K_{JJ} \beta_J - K_{JI}(y_I - o_I). \tag{6}$$

P and g are also the (generalized) Hessian and gradient of the objective function (4).

Because the loss function is piecewise quadratic, Newton method converges in a finite number of iterations. The number of iterations required to converge to the exact solution of (4) is usually very small (less than 5). Some Matlab code is available online at `http://www.kyb.tuebingen.mpg.de/bs/people/chapelle/primal`.

2.2 Updating the Hessian

As already pointed out in Section 1, we will mainly need to solve (4) in an incremental mode:[4] with the solution β_J of (4) already available, solve (4) again, but with one more basis function added, i.e., J incremented by one. Keerthi and DeCoste (2005) show that the Newton method is very efficient

4. In our method basis functions are added one at a time.

for such seeding situations. Since the kernel matrix is dense, we maintain and update a Cholesky factorization of P, the Hessian defined in (6). Even with J fixed, during the course of solving (4) via the Newton method, P will undergo changes due to changes in I. Efficient rank one schemes can be used to do the updating of the Cholesky factorization (Seeger, 2004). The updatings of the factorization of P that need to be done because of changes in I are not going to be expensive because such changes mostly occur when J is small; when J is large, I usually undergoes very small changes since the set of training errors is rather well identified by that stage. Of course P and its factorization will also undergo changes (their dimensions increase by one) each time an element is added to J. This is a routine updating operation that is present in most forward selection methods.

2.3 Computational Complexity

It is useful to ask: what is the complexity of the incremental computations needed to solve (4) when its solution is available for some J, at which point one more basis element is included in it and we want to re-solve (4)? In the best case, when the support vector set I does not change, the cost is mainly the following: computing the new row and column of K_{JJ} ($d+1$ kernel evaluations); computing the new row of K_{JI} (n kernel computations);[5] computing the new elements of P ($O(nd)$ cost); and the updating of the factorization of P ($O(d^2)$ cost). Thus the cost can be summarized as: $(n+d+1)$ kernel evaluations and $O(nd)$ cost. Even when I does change and so the cost is more, it is reasonable to take the above mentioned cost summary as a good estimate of the cost of the incremental work. Adding up these costs till $d_{\max}$ basis functions are selected, we get a complexity of $O(nd_{\max}^2)$. Note that this is the basic cost given that we already know the sequence of $d_{\max}$ basis functions that are to be used. Thus, $O(nd_{\max}^2)$ is also the complexity of the method in which basis functions are chosen randomly. In the next section we discuss the problem of selecting the basis functions systematically and efficiently.

3. Selection of New Basis Element

Suppose we have solved (4) and obtained the minimizer β_J. Obviously, the minimum value of the objective function in (4) (call it f_J) is greater than or equal to $f^\star$, the optimal value of (1). If the difference between them is large we would like to continue on and include another basis function. Take one $j \notin J$. How do we judge its value of inclusion? The best scoring mechanism is the following one.

3.1 Basis Selection Method 1

Include j in J, optimize (4) fully using (β_J, β_j), and find the improved value of the objective function; call it $\tilde{f}_j$. Choose the j that gives the least value of $\tilde{f}_j$. We already analyzed in the earlier section that the cost of doing one basis element inclusion is $O(nd)$. So, if we want to try all elements outside J, the cost is $O(n^2 d)$; the overall cost of such a method of selecting $d_{\max}$ basis functions is $O(n^2 d_{\max}^2)$, which is much higher than the basic cost, $O(nd_{\max}^2)$ mentioned in the previous section. Instead, if we work only with a random subset of size κ chosen from outside J, then the cost in one basis selection step comes down to $O(\kappa nd)$, and the overall cost is limited to $O(\kappa nd_{\max}^2)$. Smola and Bartlett (2001) have successfully tried such random subset choices for Gaussian process regression, using $\kappa = 59$. However, note that, even with this scheme, the cost of new basis selection ($O(\kappa nd)$)

5. In fact this is not n but the size of I. Since we do not know this size, we upper bound it by n.

is still disproportionately higher (by κ times) than the cost of actually including the newly selected basis function ($O(nd)$). Thus we would like to go for cheaper methods.

3.2 Basis Selection Method 2

This method computes a score for a new element j in $O(n)$ time. The idea has a parallel in Vincent and Bengio's work on Kernel Matching Pursuit (Vincent and Bengio, 2002) for least squares loss functions. They have two methods called *prefitting* and *backfitting*; see equations (7), (3) and (6) of Vincent and Bengio (2002).[6] Their *prefitting* is parallel to *Basis Selection Method 1* that we described earlier. The cheaper method that we suggest below is parallel to their *backfitting* idea. Suppose β_J is the solution of (4). Including a new element j and its corresponding variable, β_j yields the problem of minimizing

$$\frac{\lambda}{2}(\beta_J^\top \;\; \beta_j)\begin{pmatrix} K_{JJ} & K_{Jj} \\ K_{jJ} & K_{jj} \end{pmatrix}\begin{pmatrix} \beta_J \\ \beta_j \end{pmatrix} + \frac{1}{2}\sum_{i=1}^{n} \max(0, 1 - y_i(K_{iJ}\beta_J + K_{ij}\beta_j)^2, \tag{7}$$

We fix β_J and optimize (7) using only the new variable β_j and see how much improvement in the objective function is possible in order to define the score for the new element j.

This one dimensional function is piecewise quadratic and can be minimized exactly in $O(n \log n)$ time by a dichotomy search on the different breakpoints. But, a very precise calculation of the scoring function is usually unnecessary. So, for practical solution we can simply do a few Newton-Raphson-type iterations on the derivative of the function and get a near optimal solution in $O(n)$ time. Note that we also need to compute the vector K_{Jj}, which requires d kernel evaluations. Though this cost is subsumed in $O(n)$, it is a factor to remember if kernel evaluations are expensive.

If all $j \notin J$ are tried, then the complexity of selecting a new basis function is $O(n^2)$, which is disproportionately large compared to the cost of including the chosen basis function, which is $O(nd)$. Like in *Basis Selection Method 1*, we can simply choose κ random basis functions to try. If $d_{\max}$ is specified, one can choose $\kappa = O(d_{\max})$ without increasing the overall complexity beyond $O(nd_{\max}^2)$. More complex schemes incorporating a kernel cache can also be tried.

3.3 Kernel Caching

For upto medium size problems, say $n < 15,000$, it is a good idea to have cache for the entire kernel matrix. If additional memory space is available and, say a Gaussian kernel is employed, then the values of $\|x_i - x_j\|^2$ can also be cached; this will help significantly reduce the time associated with the tuning of hyperparameters. For larger problems, depending on memory space available, it is a good idea to cache as many as possible, full kernel rows corresponding to j that get tried, but do not get chosen for inclusion. It is possible that they get called in a later stage of the algorithm, at which time, this cache can be useful. It is also possible to think of variations of the method in which full kernel rows corresponding to a large set (as much that can fit into memory) of randomly chosen training basis is pre-computed and only these basis functions are considered for selection.

3.4 Shrinking

As basis functions get added, the SVM solution w and the margin planes start stabilizing. If the number of support vectors form a small fraction of the training set, then, for a large fraction of

6. For least squares problems, Adler et al. (1996) had given the same ideas as Vincent and Bengio in earlier work.

(well-classified) training examples, we can easily conclude that they will probably never come into the active set I. Such training examples can be left out of the calculations without causing any undue harm. This idea of shrinking has been effectively used to speed-up SVM training (Joachims, 1999; Platt, 1998).

3.5 Experimental Evaluation

We now evaluate the performance of basis selection methods 1 and 2 (we will call them as *SpSVM-1*, *SpSVM-2*) on some sizable benchmark data sets. A full description of these data sets and the kernel functions used is given in the appendix. The value of $\kappa = 59$ is used. To have a baseline, we also consider the method, *Random* in which the basis functions are chosen randomly. This is almost the same as the RSVM method (Lee and Mangasarian, 2001; Lin and Lin, 2003), the only difference being the regularizer ($\beta_J^\top K_{J,J}\beta_J$ in (4) versus $\|\beta_J\|^2$ in RSVM). For another baseline we consider the (more systematic) unsupervised learning method in which an incomplete Cholesky factorization with pivoting (Meijerink and van der Vorst, 1977; Bach and Jordan, 2005) is used to choose basis functions.[7] For comparison we also include the GSVC method of Parrado-Hernández et al. (2003). This method, originally given for SVM hinge loss, uses the following heuristic criterion to select the next basis function $j^* \notin J$:

$$j^* = \arg \min_{j\in I, j\notin J} \max_{l\in J} |K_{jl}| \tag{8}$$

with the aim of encouraging new basis functions that are far from the basis functions that are already chosen; also, j is restricted only to the support vector indices (I in (5)). For a clean comparison with our methods, we implemented GSVC for SVMs using quadratic penalization, $\max(0, 1-y_i o_i)^2$. We also tried another criterion, suggested to us by Alex Smola, that is more complex than (8):

$$j^* = \arg \max_{j\in I, j\notin J} (1-y_j o_j)^2 d_j^2 \tag{9}$$

where d_j is the distance (in feature space) of the j-th training point from the subspace spanned by the elements of J. This criterion is based on an upper bound on the improvement to the training cost function obtained by including the j-th basis function. It also makes sense intuitively as it selects basis functions that are both not well approximated by the others (large d_j) and for which the error incurred is large.[8] Below, we will refer to this criterion as *BH*. It is worth noting that both (8) and (9) can be computed very efficiently.

Figures 1 and 2 compare the six methods on six data sets.[9] Overall, *SpSVM-1* and *SpSVM-2* give the best performance in terms of achieving good reduction of test error rate with respect to the number of basis functions. Although *SpSVM-2* slightly lags *SpSVM-1* in terms of performance in the early stages, it does equally well as more basis functions are added. Since *SpSVM-2* is significantly less expensive, it is the best method to use. Since *SpSVM-1* is quite cheap in the early stages, it is also appropriate to think of a hybrid method in which *SpSVM-1* is used in the early stages and, when it becomes expensive, switch to *SpSVM-2*. The other methods sometimes do well, but, overall, they are inferior in comparison to *SpSVM-1* and *SpSVM-2*. Interestingly, on the *IJCNN* and *Vehicle* data

7. We also tried the method of Bach and Jordan (2005) which uses the training labels, but we noticed little improvement.
8. Note that when the set of basis functions is not restricted, the optimal β satisfies $\lambda\beta_i y_i = \max(0, 1-y_i o_i)$.
9. Most figures given in this paper appear in pairs of two plots. One plot gives test error rate as a function of the number of basis functions, to see how effective the compression is. The other plot gives the test error rate as a function of CPU time, and is used to indicate the efficiency of the method.

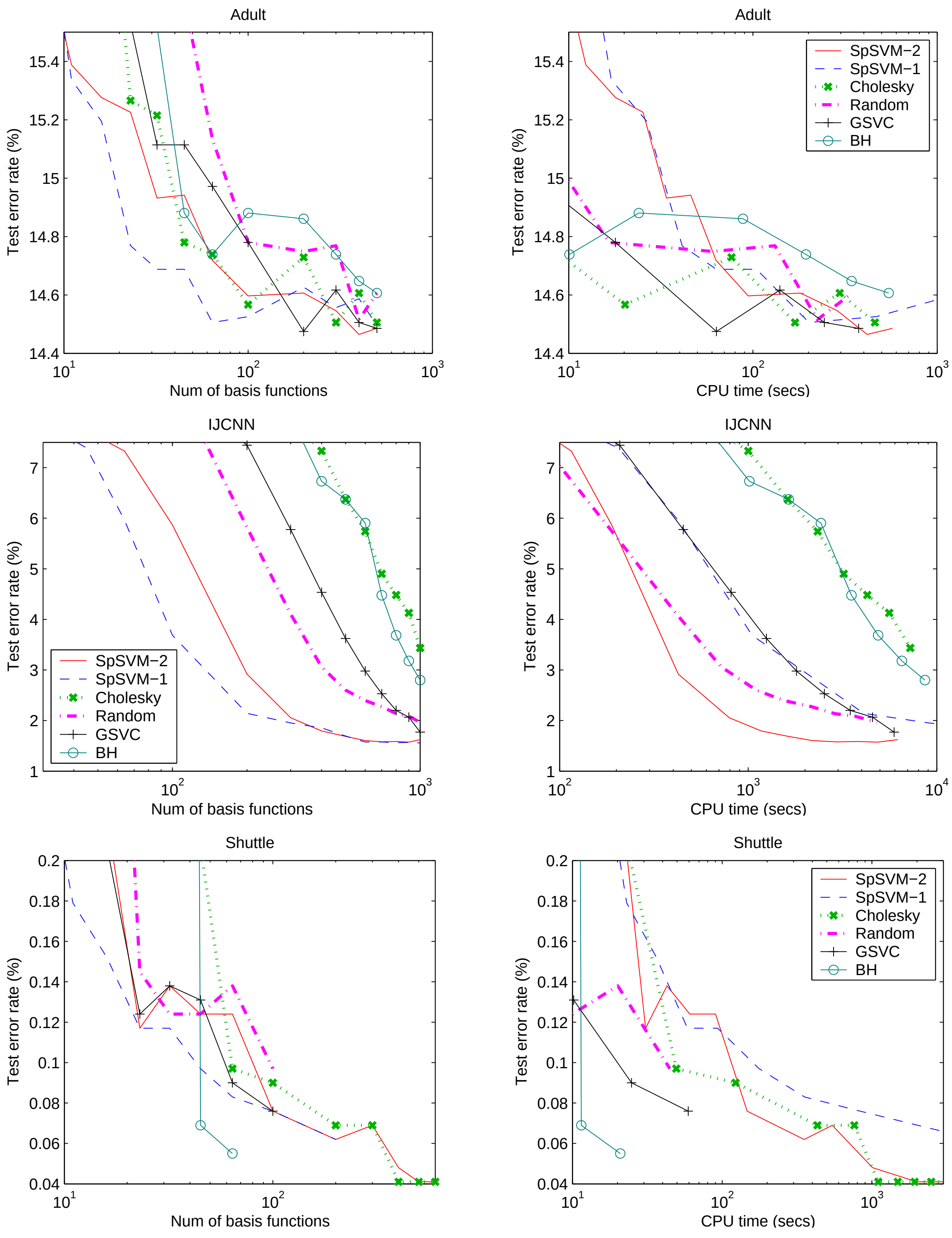

Figure 1: Comparison of basis selection methods on *Adult*, *IJCNN* & *Shuttle*. On *Shuttle* some methods were terminated because of ill-conditioning in the matrix P in (6).

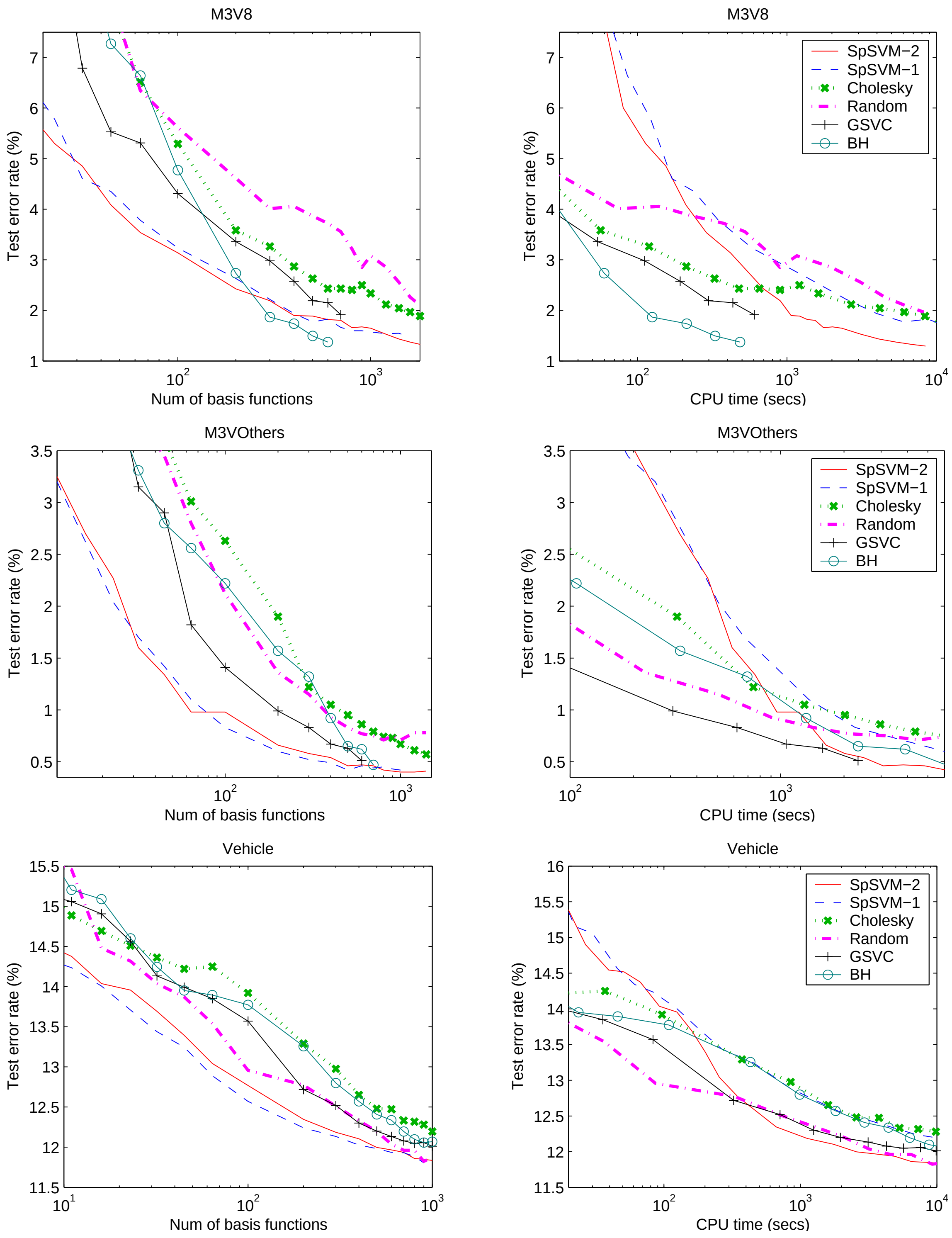

Figure 2: Comparison of basis selection methods on *M3V8*, *M3VOthers* & *Vehicle*.

sets, *Cholesky, GSVC* and *BH* are even inferior to *Random*. A possible explanation is as follows: these methods give preference to points that are furthest away in feature space from the points already selected. Thus, they are likely to select points which are outliers (far from the rest of the training points); but outliers are probably unsuitable points for expanding the decision function.

As we mentioned in Section 1, there also exist other greedy methods of kernel basis selection that are motivated by ideas from boosting. These methods are usually given in a setting different from that we consider: a set of (kernel) basis functions is given and a regularizer (such as $\|\beta\|_1$) is directly specified on the multiplier vector β. The method of Bennett et al. (2002) called MARK is given for least squares problems. It is close to the kernel matching pursuit method. We compare *SpSVM-2* with kernel matching pursuit and discuss MARK in Section 5. The method of Bi et al. (2004) uses column generation ideas from linear and quadratic programming to select new basis functions and so it requires the solution of, both, the primal and dual problems.[10] Thus, the basis selection process is based on the sensitivity of the primal objective function to an incoming basis function. On the other hand, our *SpSVM* methods are based on computing an estimate of the decrease in the primal objective function due to an incoming basis function; also, the dual solution is not needed.

4. Hyperparameter Tuning

In the actual design process, the values of hyperparameters need to be determined. This can be done using k-fold cross validation. Cross validation (CV) can also be used to choose d, the number of basis functions. Since the solution given by our method approaches the SVM solution as d becomes large, there is really no need to choose d at all. One can simply choose d to be as big a value as possible. But, to achieve good reduction in the classifier complexity (as well as computing time) it is a good idea to track the validation performance as a function of d and stop when this function becomes nearly flat. We proceed as follows. First an appropriate value for $d_{\max}$ is chosen. For a given choice of hyperparameters, the basis selection method (say, *SpSVM-2*) is then applied on each training set formed from the k-fold partitions till $d_{\max}$ basis functions are chosen. This gives an estimate of the k-fold CV error for each value of d from 1 to $d_{\max}$. We choose d to be the number of basis functions that gives the lowest k-fold CV error. This computation can be repeated for each set of hyperparameter values and the best choice can be decided.

Recall that, at stage d, our basis selection methods choose the $(d+1)$-th basis function from a set of κ random basis functions. To avoid the effects of this randomness on hyperparameter tuning, it is better to make this κ-set to be dependent only on d. Thus, at stage d, the basis selection methods will choose the same set of κ random basis functions for all hyperparameter values.

We applied the above ideas on 11 benchmark data sets from (Rätsch) using *SpSVM-2* as the basis selection method. The Gaussian kernel, $k(x_i, x_j) = 1 + \exp(-\gamma\|x_i - x_j\|^2)$ was used. The hyperparameters, λ and γ were tuned using 3-fold cross validation. The values, 2^i, $i = -7, \cdots, 7$ were used for each of these parameters. Ten different train-test partitions were tried to get an idea of the variability in generalization performance. We used $\kappa = 25$ and $d_{\max} = 25$. (The *Titanic* data set has three input variables, which are all binary; hence we set $d_{\max} = 8$ for this data set.)

Table 1 (already introduced in Section 1) gives the results. For comparison we also give the results for the SVM (solution of (1)); in the case of SVM, the number of support vectors (n_{SV}) is the

10. The CPLEX LP/QP solver is used to obtain these solutions.

number of basis functions. Clearly, our method achieves an impressive reduction in the number of basis functions, while yielding test error rates comparable to the full SVM.

5. Comparison with Kernel Matching Pursuit

Kernel matching pursuit (KMP) (Vincent and Bengio, 2002) was mainly given as a method of greedily selecting basis functions for the non-regularized kernel least squares problem. As we already explained in Section 3, our basis selection methods can be viewed as extensions of the basic ideas of KMP to the SVM case. In this section we empirically compare the performances of these two methods. For both methods we only consider *Basis Selection Method 2* and refer to the two methods simply as *KMP* and *SpSVM*. It is also interesting to study the effect of the regularizer term ($\lambda\|w\|^2/2$ in (1)) on generalization. The regularizer can be removed by setting $\lambda = 0$. The original KMP formulation of Vincent and Bengio (2002) considered such a non-regularized formulation only. In the case of SVM, when perfect separability of the training data is possible, it is improper to set $\lambda = 0$ without actually rewriting the primal formulation in a different form; so, in our implementation we brought in the effect of no-regularization by setting λ to the small value, 10^{-5}. Thus, we compare 4 methods: *KMP-R*, *KMP-NR*, *SpSVM-R* and *SpSVM-NR*. Here "*R*" and "*NR*" refer to regularization and no-regularization, respectively.

Figures 3 and 4 compare the four methods on six data sets. Except on *M3V8*, SpSVM gives a better performance than KMP. The better performance of *KMP* on *M3V8* is probably due to the fact that the examples corresponding to each of the digits, 3 and 8, are distributed as a Gaussian, which is suited to the least squares loss function. Note that in the case of *M3VOthers* where the "Others" class (corresponding to all digits other than 3) is far from a Gaussian distribution, SVM does better than KMP.

The no-regularization methods, *KMP-NR* and *SpSVM-NR* give an interesting performance. In the initial stages of basis addition we are in the underfitting zone and so they perform as well (in fact, a shade better) than their respective regularized counterparts. But, as expected, they start overfitting when many basis functions are added. See, for example the performance on *Adult* data set given in Figure 3. Thus, when using these non-regularized methods, a lot of care is needed in choosing the right number of basis functions. The number of basis functions at which overfitting sets-in is smaller for *SpSVM-NR* than that of *KMP-NR*. This is because of the fact that, while KMP has to concentrate on reducing the residual on all examples in its optimization, SVM only needs to concentrate on the examples violating the margin condition.

It is also useful to mention the method, MARK[11] of Bennett et al. (2002) which is closely related to KMP. In this method, a new basis function (say, the one corresponding to the j-th training example) is evaluated by looking at the magnitude (larger the better) of the gradient of the primal objective function with respect to β_j evaluated at the current β_J. This gradient is the dot product of the kernel column containing K_{ij} and the residual vector having the elements, $o_i - y_i$. The computational cost as well as the performance of MARK are close to those of KMP. MARK can also be easily extended to the SVM problem in (1): all that we need to do is to replace the residual vector mentioned above by the vector having the elements, $y_i \max\{0, 1 - y_i o_i\}$. This modified method (which uses our Newton optimization method as the base solver) is close to our *SpSVM-2* in terms of computational cost as well as performance. Note that, if we optimize (7) for β_j using only a

11. The basis selection idea used in MARK is also given in the earlier papers, Mallat and Zhang (1993) and Adler et al. (1996) under the name, *Basic Matching Pursuit*.

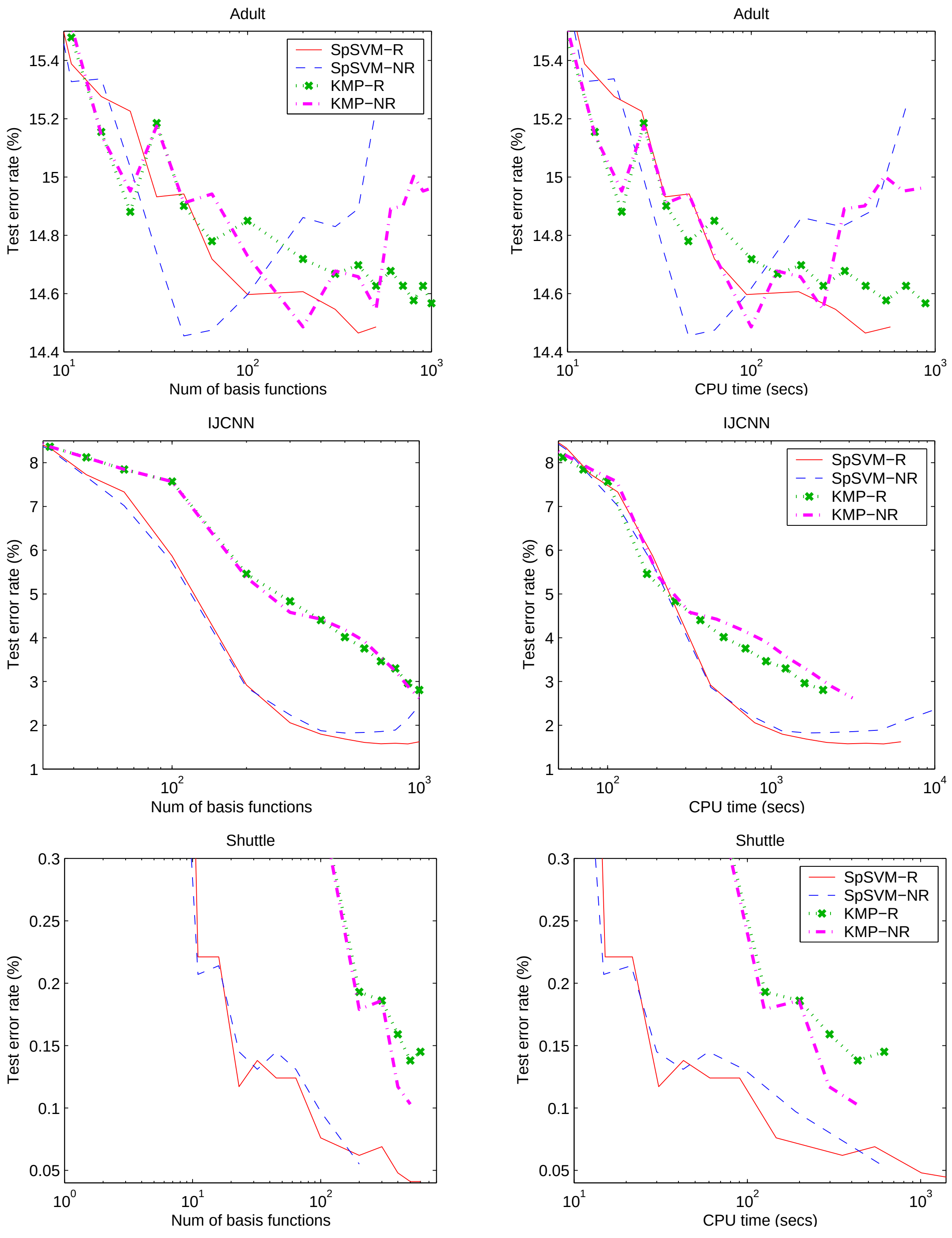

Figure 3: KMP vs SpSVM (with/without regularization) on *Adult*, *IJCNN* & *Shuttle*.

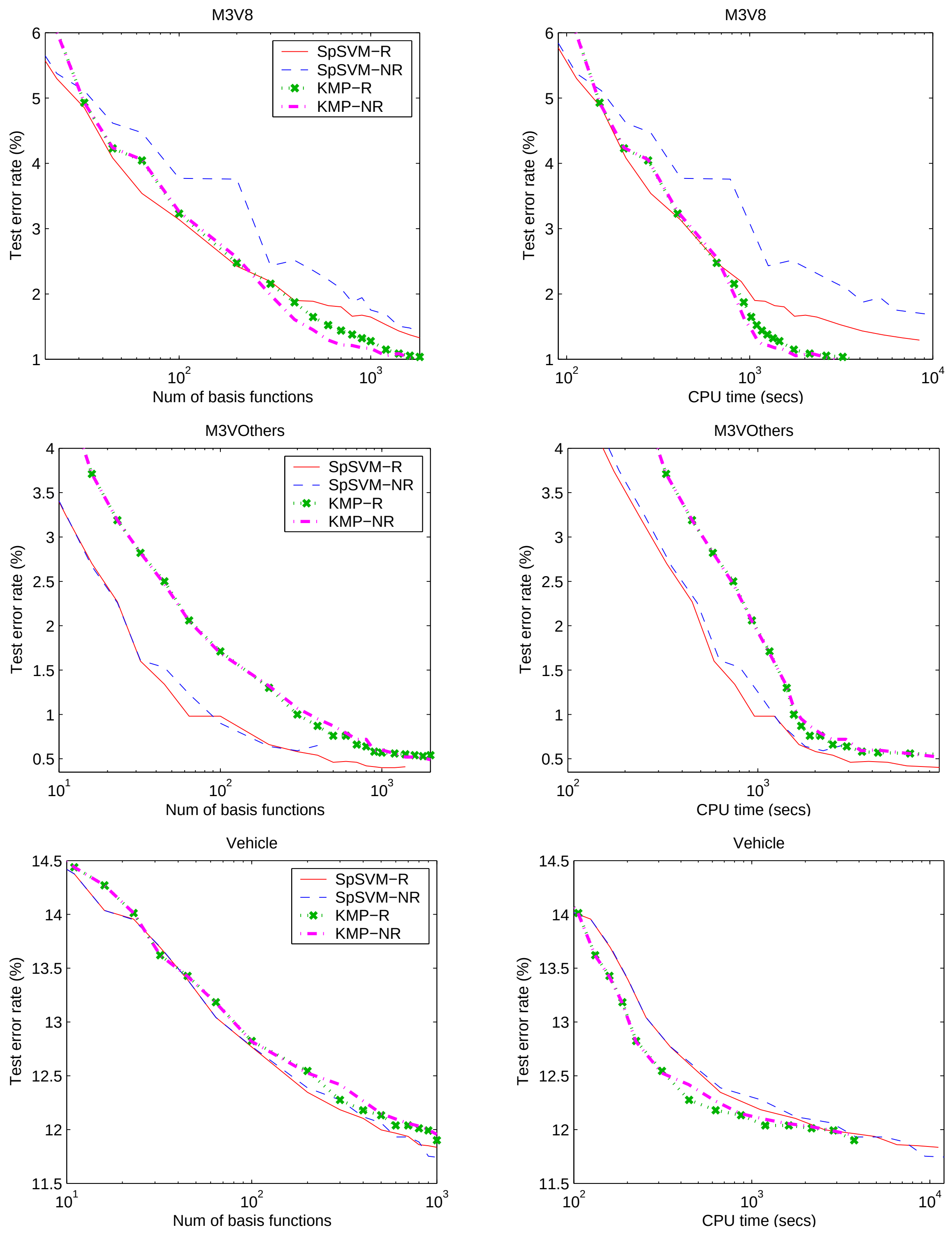

Figure 4: KMP vs SpSVM (with/without regularization) on *M3V8, M3VOthers* & *Vehicle*.

single Newton step, the difference between MARK (as adapted above to SVMs) and *SpSVM-2* is only in the use of the second order information.

6. Additional Tuning

We discuss in this section the choice of κ for *SpSVM* as well as the possibility of not solving (4) every time a new basis function is added.

6.1 Few Retrainings

It might be a bit costly to perform the Newton optimization described in Section 2.1 each time a new basis function is added. Indeed, it is unlikely that the set of support vectors changes a lot after each addition. Therefore, we investigate the possibility of retraining only from time to time.

We first tried to do retraining only when $|J| = 2^p$ for some $p \in \mathbb{N}$, the set of positive integers. It makes sense to use an exponential scale since we expect the solution not to change too much when J is already relatively large. Note that the overall complexity of the algorithm does not change since the cost of adding one basis function is still $O(nd)$. It is only the constant which is better, because fewer Newton optimizations are performed.

The results are presented in figure 5. For a given number of basis functions, the test error is usually not as good as if we always retrain. But on the other hand, this can be much faster. We found that a good trade-off is to retrain whenever $|J| = \lfloor 2^{p/4} \rfloor$ for $p \in \mathbb{N}$. This is the strategy we will use for the experiments in the rest of the paper.

6.2 Influence of κ

The parameter κ is the number of candidate basis functions that are being tried each time a new basis function should be added: we select a random set of κ examples and the best one (as explained in Section 3.2) among them is chosen. If $\kappa = 1$, this amounts to choosing a random example at each step (i.e. the *Random* method on figures 1 and 2) .

The influence of κ is studied in figure 6. The larger κ is, the better the test error for a given number of basis functions, but also the longer the training time. We found that $\kappa = 10$ is a good trade-off and that is the value that we will keep for the experiments presented in the next section.

Finally, an interesting question is how to choose appropriately a good value for κ and an efficient retraining strategy. Both are likely to be problem dependent, and even though $\kappa = 59$ was suggested by Smola and Schölkopf (2000), we believe that there is no universal answer. The answer would for instance depend on the cost associated with the computation of the kernel function, on the number of support vectors and on the number of training points. Indeed, the basic cost for one iteration is $O(nd)$ and the number of kernel calculations is $\kappa n_{\mathrm{SV}} + n$: the first term corresponds to trying different basis function, while the second one correspond to the inclusion of the chosen basis function. So κ should be chosen such that the kernel computations takes about the same tame as the training itself.

Ideally, an adaptive strategy should be designed to find automatically κ and to adjust the retraining schedule. The decay rate of the objective function as well as the variance of the scores produced by the basis selection scoring function would be two key quantities helpful to adjust them.

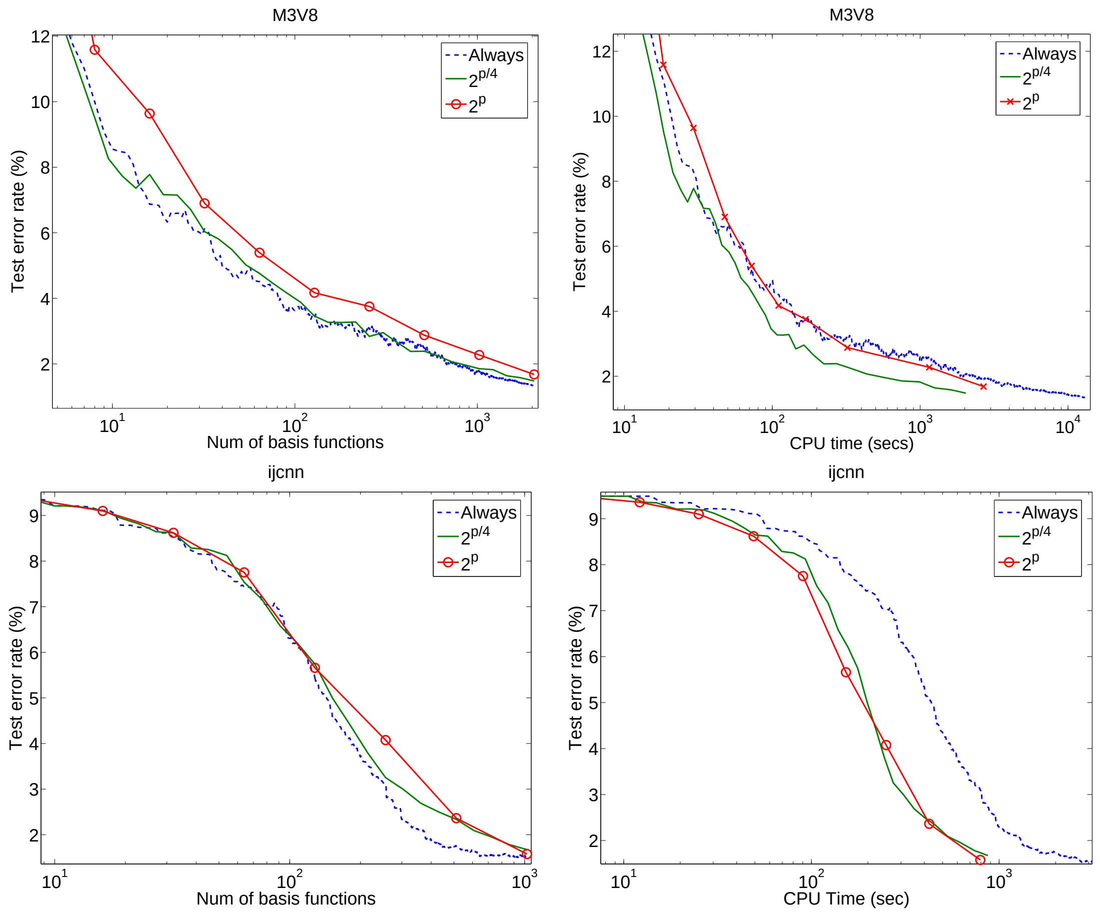

Figure 5: Three different possible retraining strategies showing a different trade-off between accuracy and time: always retraining is too time consumming; on the other hand retraining not often enough can lead to sub-optimal performances (see the top left plot). For these experiments, $\kappa = 100$ was used.

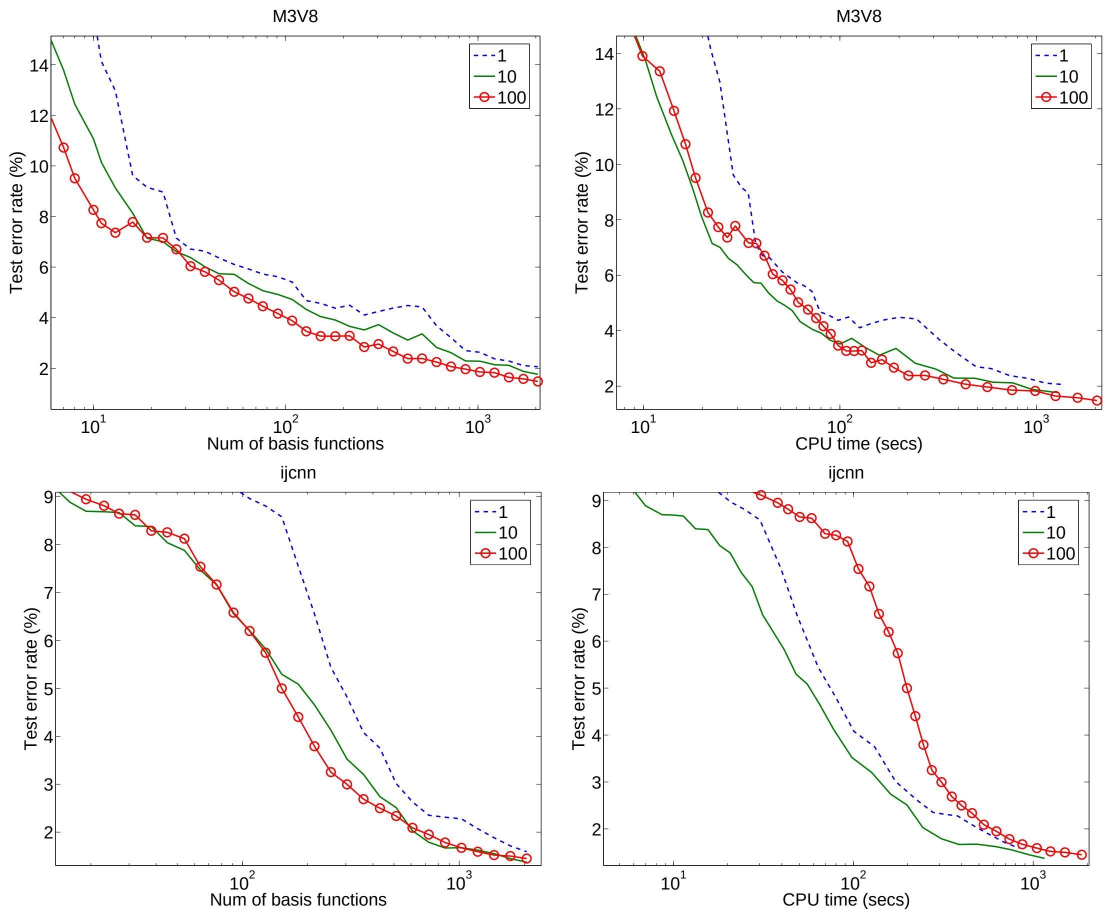

Figure 6: Influence of the paramter κ: when it is large, a good reduction is achieved (left column), but the computaional cost is larger (right column). $\kappa = 10$ seems a good trade-off.

7. Comparison with Standard SVM Training

We conclude the experimental study by comparing our method with the well known SVM solver, *SVMLight* (Joachims, 1999).[12] For this solver, we selected random training subsets of sizes from $2^{-10}n, 2^{-9}n, \ldots, n/4, n/2, n$. For each training set size, we measure the test error, the training time and the number of support vectors. The L_2 version (quadratic penalization of the slacks) is the one relevant for our study since it is the same loss function as the one we used; note that, when the number of basis functions increases towards n, the *SpSVM* solution will converge to the L_2 SVM solution. For completeness, we also included experimental results of an SVM trained with a L_1 penalization of the slacks. Finally, note that for simplicity we kept the same hyperparameters for the different sizes, but that both methods would certainly gain by additional hyerparameter tuning (for instance when the number of basis functions is smaller, the bandwith of the RBF kernel should be larger).

Results are presented in figures 7 and 8. In terms of compression (left columns), our method is usually able to reach the same accuracy as a standard SVM using less than one-tenth the number of basis functions (this confirms the results of table 1).

From a time complexity point of view also, our method is very competitive and can reach the same accuracy as an SVM in less time. The only disappointing performance is on the *M3V8* data set. A possible explanation is that for this data set, the number of support vectors is very small and a standard SVM can compute the exact solution quickly.

Finally, note that when the number of basis functions is extremely small compared to the number of training examples, *SpSVM* can be slower than a SVM trained on a small subset (left part of the right column plots). It is because solving (4) using n training examples while there are only few parameters to estimate is an overkill. It would be wiser to choose n as a function of d, the number of basis functions.

8. Conclusion

In this paper we have given a fast primal algorithm that greedily chooses a subset of the training basis functions to approximate the SVM solution. As the subset grows the solution converges to the SVM solution since choosing the subset to be the entire training set is guaranteed to yield the exact SVM solution. The real power of the method lies in its ability to form very good approximations of the SVM classifier with a clear control on the complexity of the classifier (number of basis functions) as well as the training time. In most data sets, performance very close to that of the SVM is achieved using a set of basis functions whose size is a small fraction of the number of SVM support vectors. The graded control over the training time offered by our method can be valuable in large scale data mining. Many a times, simpler algorithms such as decision trees are preferred over SVMs when there is a severe constraint on computational time. While there is no satisfactory way of doing early stopping with SVMs, our method enables the user to control the training time by choosing the number of basis functions to use.

Our method can be improved and modified in various ways. Hyperparameter tuning time can be substantially reduced by using gradient-based methods on a differentiable estimate of the generalization performance formed using k-fold cross validation and posterior probabilities. Improved methods of choosing the κ-subset of basis functions in each step can also make the method more ef-

12. The default optimization options of *SVMLight (Version 6.0)* have been used.

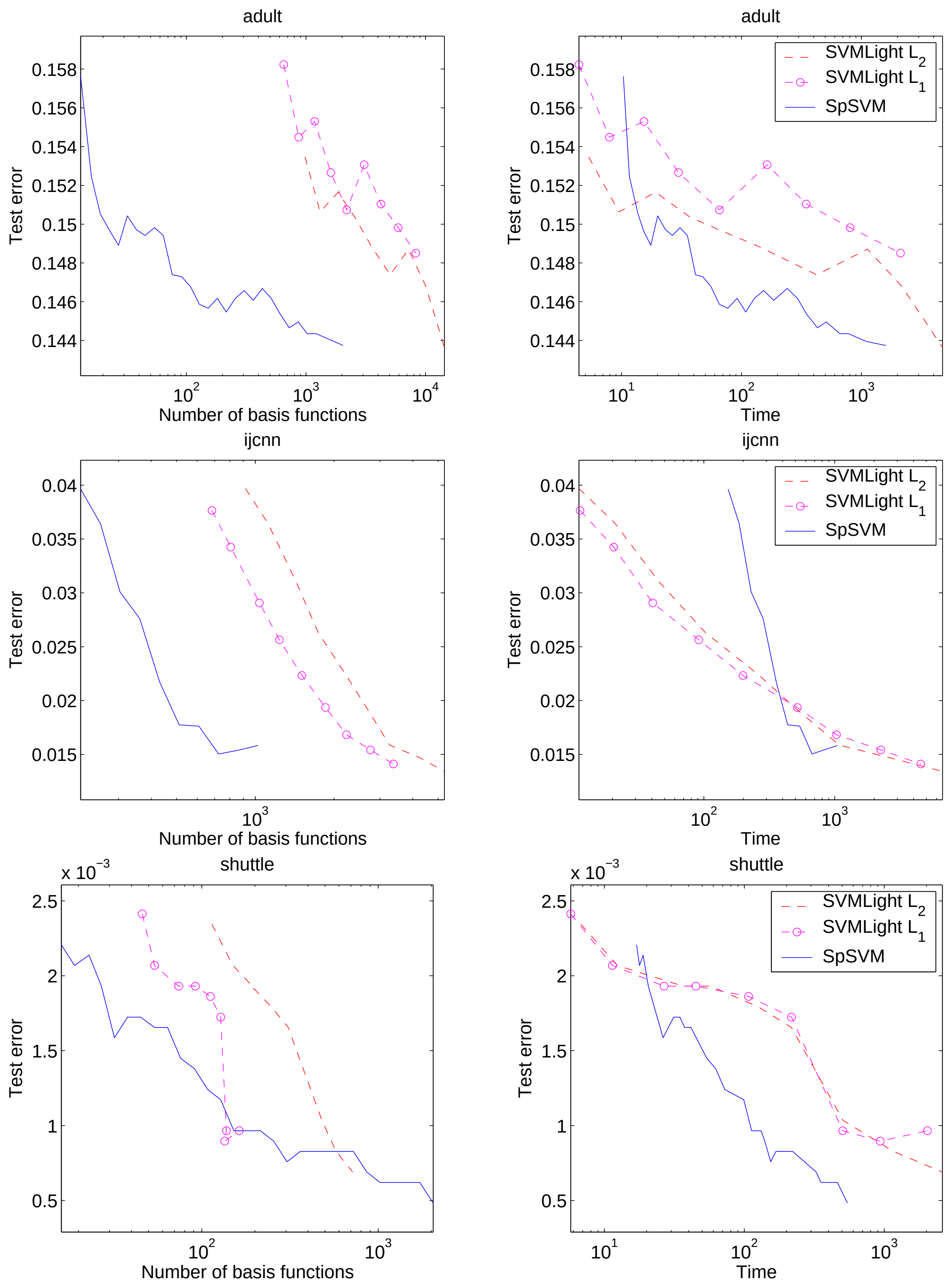

Figure 7: Comparison of *SpSVM* with SVMLight on *Adult*, *IJCNN*, *Shuttle*. For SVMLight, “Num of basis functions” should be understood as number of support vectors.

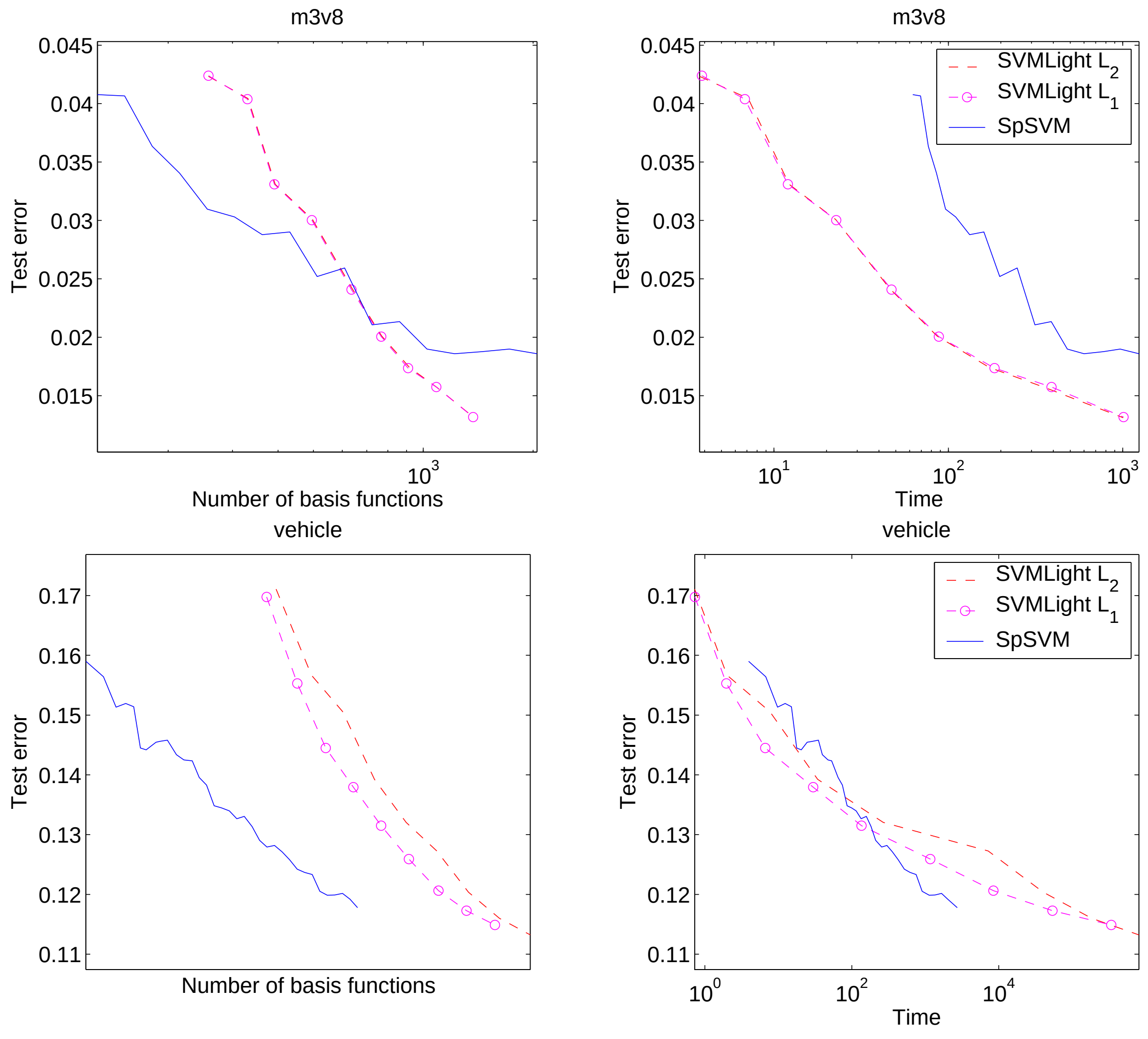

Figure 8: Comparison of *SpSVM* with SVMLight on *M3V8* and *Vehicle*. For SVMLight, “Num of basis functions” should be understood as number of support vectors.

fective. Also, all the ideas described in this paper can be easily extended to the Huber loss function using the ideas in Keerthi and DeCoste (2005).

Appendix: A Description of Data Sets Used

As in the main paper, let n denote the number of training examples. The six data sets used for the main experiments of the paper are: *Adult*, *IJCNN*, *M3V8*, *M3VOthers*, *Shuttle* and *Vehicle*. For *M3V8* and *M3VOthers* we go by the experience in (DeCoste and Schölkopf, 2002) and use the polynomial kernel, $k(x_i,x_j) = 1+(1+x_i \cdot x_j)^9$ where each x_i is normalized to have unit length. For all other data sets, we use the Gaussian kernel, $k(x_i,x_j) = 1+\exp(-\gamma\|x_i-x_j\|^2)$. The values of γ are given below.[13] In each case, the values chosen for γ and λ are ballpark values such that the methods considered in the paper give good generalization performance.

Adult data set is the version given by Platt in his *SMO* web page: `http://www.research.microsoft.com/~jplatt/smo.html`. Platt created a sequence of data sets with increasing number of examples in order to study the scaling properties of his *SMO* algorithm with respect to n. For our experiments we only used *Adult-8* which has 22,696 training examples and 9865 test examples. Each example has 123 binary features, of which typically only 14 are non-zero. We used $\gamma = 0.05$ and $\lambda = 1$.

The next five data sets are available from the following LIBSVM-Tools page: `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

IJCNN data set has 49,990 training examples and 91,701 test examples. Each example is described by 22 features. We used $\gamma = 4$ and $\lambda = 1/16$.

Shuttle data set has 43,500 training examples and 14,500 test examples. Each example is described by 9 features. This is a multiclass data set with seven classes. We looked only at the binary classification problem of differentiating class 1 from the rest. We used $\gamma = 16$ and $\lambda = 1/512$.

M3V8 data set is the binary classification problem of *MNIST* corresponding to classifying digit 3 from digit 8. The original data set has 11,982 training examples and 1984 test examples for this problem. Since the original test data set could not clearly show a distinction between several closely competing methods, we formed an extended test set by applying invariances like translation and rotation to create an extended test set comprising of 17,856 examples. (This data set can be obtained from the authors.) We used $\lambda = 0.1$.

M3VOthers data set is another binary classification problem of *MNIST* corresponding to differentiating digit 3 from all the other digits. The data set has 60,000 training examples and 10,000 test examples. We used $\lambda = 0.1$.

Vehicle data set corresponds to the “vehicle (combined, scaled to [-1,1])” version in the LIBSVM-Tools page mentioned above. It has 78,823 training examples and 19,705 test examples. Each example is described by 100 features. This is a multiclass data set with three classes. We looked only at the binary classification problem of differentiating class 3 from the rest. We used $\gamma = 1/8$ and $\lambda = 1/32$.

Apart from the above six large data sets, we also used modified versions of UCI data sets as given in (Rätsch). These data sets were used to show the sparsity that is achievable using our method; see Table 1 of Section 1 and the detailed discussion in Section 4.

13. For both, the polynomial and Gaussian kernels, the additive term “1” gives the effect of including the threshold term in the classifier and regularizing it.

References

J. Adler, B. D. Rao, and K. Kreutz-Delgado. Comparison of basis selection methods. In *Proceedings of the 30th Asilomar conference on signals, systems and computers*, pages 252–257, 1996.

F. Bach and M. Jordan. Predictive low-rank decomposition for kernel methods. In *Proceedings of the Twenty-second International Conference on Machine Learning (ICML)*, 2005.

K. P. Bennett, M. Momma, and M. J. Embrechts. MARK: A boosting algorithm for heterogeneous kernel models. In *Proceedings of SIGKDD'02*, 2002.

J. Bi, T. Zhang, and K. P. Bennet. Column generation boosting methods for mixture of kernels. In *Proceedings of SIGKDD'04*, 2004.

C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector learning machines. In *Proceedings of the* 9th *NIPS Conference*, pages 375–381, 1997.

O. Chapelle. Training a support vector machine in the primal. *Journal of Machine Learning Research*, 2005. submitted.

D. DeCoste and B. Schölkopf. Training invariant support vector machines. *Machine Learning*, 46: 161–190, 2002.

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

J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 29:1180, 2001.

T. Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, Cambridge, Massachussetts, 1999.

S. S. Keerthi and W. Chu. A matching pursuit approach to sparse Gaussian process regression. In *Proceedings of the* 18th *NIPS Conference*, 2006.

S. S. Keerthi and D. DeCoste. A modified finite Newton method for fast solution of large scale linear svms. *Journal of Machine Learning Research*, 6:341–361, 2005.

N. Lawrence, M. Seeger, and R. Herbrich. Fast sparse Gaussian process methods: The Informative Vector Machine. In *Proceedings of the* 15th *NIPS Conference*, pages 609–616, 2003.

Y. J. Lee and O. L. Mangasarian. RSVM: Reduced support vector machines. In *Proceedings of the SIAM International Conference on Data Mining*. SIAM, Philadelphia, 2001.

K. M. Lin and C. J. Lin. A study on reduced support vector machines. *IEEE TNN*, 14:1449–1459, 2003.

S. G. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on ASSP*, 41:3397–3415, 1993.

O. L. Mangasarian. A finite Newton method for classification. *Optimization Methods and Software*, 17:913–929, 2002.

J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of Computation*, 31:148–162, 1977.

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

E. Parrado-Hernández, I. Mora-Jimenéz, J. Arenas-García, A. R. Figueiras-Vidal, and A. Navia-Vázquez. Growing support vector classifiers with controlled complexity. *Pattern Recognition*, 36:1479–1488, 2003.

J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, Microsoft Research, Redmond, 1998.

G. Rätsch. *Robust boosting via convex optimization*. PhD thesis, University of Potsdam, Department of Computer Science, Potsdam, Germany, 2001.

G. Rätsch. Benchmark repository. `http://ida.first.fraunhofer.de/~raetsch/`.

B. Schölkopf, S. Mika, C. J. C. Burges, P. Knirsch, K. R. Muller, G. Raetsch, and A. J. Smola. Input space vs. feature space in kernel-based methods. *IEEE TNN*, 10:1000–1017, 1999.

M. Seeger. Low rank updates for the Cholesky decomposition. Technical report, University of California, Berkeley, 2004.

M. Seeger, C. Williams, and N. Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In *Proceedings of the Workshop on AI and Statistics*, 2003.

A. J. Smola and P. L. Bartlett. Sparse greedy Gaussian process regression. In *Proceedings of the* 13^{th} *NIPS Conference*, pages 619–625, 2001.

A. J. Smola and Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proceedings of the* 17^{th} *International Conference on Machine Learning*, pages 911–918, 2000.

I. Steinwart. Sparseness of support vector machines - some asymptotically sharp bounds. In *Proceedings of the* 16^{th} *NIPS Conference*, pages 169–184, 2004.

T. Thies and F. Weber. Optimal reduced-set vectors for support vector machines with a quadratic kernel. *Neural Computation*, 16:1769–1777, 2004.

M. E. Tipping. Sparse Bayesian learning and the Relevance Vector Machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

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

M. Wu, B. Schölkopf, and G. Bakir. Building sparse large margin classifiers. In *Proceedings of the 22nd International Conference on Machine Learning*, 2005.