Constructing Support Vector Machines with Reduced Classifier Complexity

Ankur M. Bobade, N. N. Khalsa, S. M. Deshmukh, Ph.D

ABSTRACT
Support vector machines (SVMs), though perfect, are not chosen in applications requiring great classification speed, due to the number of support vectors being large. To conquer this problem we devise a primitive method with the following properties: (1) it decouples the idea of basis functions from the concept of support vectors; (2) it materialistically finds a set of kernel basis functions of a specified maximum size (dmax) to approximate the SVM primitive cost function well; (3) it is efficient and roughly scales as O(ndmax2)where n is the number of training examples; and, (4) the number of basis functions it requires to accomplish an accuracy close to the SVM accuracy is usually far less than the number of SVM support vectors.

Keywords
Support vectors (SVs), SVMs, classification, sparse design.

1. INTRODUCTION
The 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.

2. POST PROCESSING SIMPLIFICATION AND SVM SOLUTION
Given a training set $\{(x_i,y_i)\}_{i=1}^{n}$, $y_i \in \{1, -1\}$, the SVM algorithm with an L_2penalization of the training errors consists of solving the following primitive problem.

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

Computations involving φ are handled using the kernel function, $k(x_i,x_j) = \varphi(x_i)$ $\varphi(x_j)$. 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 primitive objective function continuously differentiable. This is a great advantage and becomes necessary for developing a primitive 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 $\sum_i \alpha_i y_i k(x,xi)$. 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 give 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, $O(n\ n_{SV} + n_{SV}^3)$ and $O(n_{SV})$.

Several methods have been proposed for reducing the number of support vectors. Burges and Sch¨olkopf (1997) apply nonlinear optimization methods to seek sparse representations after building the SVM classifier. Along similar lines, Sch¨olkopf et al. (1999) use $L1$ 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.

2.1 Simplification via Basis Functions and Primitive
Instead of finding the SVM solution by maximizing the dual problem, one approach is to *directly minimize the primitive form* after invoking the representer theorem to represent w as

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

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, which will yield pretty much the same accuracy as the SVM classifier. Kernel Matching Pursuit (Vincent and Bengio, 2002) is a discriminative method that is mainly developed for the least squares loss function. Work on simplifying SVM solution has not caught up well 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 $\alpha_i$is the Lagrange multiplier associated with the i-th primitive slack constraint. While the two are one and the same at exact optimality, they can be very different when we talk of sub-optimal primitive 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.

Consider such an approach. They achieve sparsity by including the L_1 regularizer, $\lambda_1\|\boldsymbol{\beta}\|_1$in the primitive objective. But they do not develop an algorithm (for solving the modified primitive formulation and for choosing the right λ_1) that scales efficiently to large problems. write *w* as

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

where *l* is a chosen small number and optimize the primitive 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. 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; R¨atsch, 2001).

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-Hern´andez et al. (2003). KMP is an effective greedy discriminative approach that is mainly developed for least squares problems

3. PROPOSED 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 primitive objective function. We develop efficient schemes for both, the greedy selection of a new basis function, as well as the optimization of the $\beta_i$for a given selection of basis functions. For choosing upto *d*max basis functions, the overall compuational cost of our method is $O(nd_{\max}2)$.

Table 1: Comparison of *SpSVM-2* and *SVM* on benchmark

	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)

data sets from (R$\ddot{a}$tsch). For TestErate, #Basis and n_{SV}, the values are means over ten different training/test splits and the values in parentheses are the standard deviations.

The different components of the method that we develop in this paper are not new in themselves and are inspired from the above mentioned papers.

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.)

3.1 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]

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

3.2 Newton Optimization

Let $K_{ij} = k(xi, x\,j) = \varphi(xi)\varphi(x\,j)$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 primitive problem (1) becomes the *d* dimensional minimization problem of finding $\beta_J$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., $\beta^{\top}{}_J K_{JJ}\beta_J$ (as opposed to the simple regularizer, $\|\boldsymbol{\beta J}\|^2$), 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.

3.3 Newton method

1. Select a suitable starting vector, $\beta 0J$. Set $k = 0$.

2. If $\beta kJ$is the optimal solution of (4), stop.

3. Let $I = \{i : 1 - yioi \geq 0\}$ where $oi = Ki,J\ \beta kJ$is the output of the *i*-th example. Obtain $\bar{\boldsymbol{\beta}}J$as the result of a

Newton step or equivalently as the solution of the regularized least squares problem,

$$\min_{\beta_J} \frac{\lambda}{2}\beta_J^\top K_{JJ}\beta_J + \frac{1}{2}\sum_{i\in I}(1 - y_i K_{i,J}\beta_J)^2. \quad (5)$$

4. Take $\beta k+1J$ to be the minimizer of f on L, the line joining $\beta kJ$and $\bar{\beta}J$. Set $k := k + 1$ and go back to step 2 for another iteration.

The solution of (5) is given by

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

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).

Computational Complexity

It is useful to inquire: 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.

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 fJ) 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 6∈ J. How do we judge its value of inclusion? The best scoring mechanism is the following one.

Basis selection method

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).6Their 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 $\beta_J$is the solution of (4). Including a new element j and its corresponding variable, β_j yields the problem of minimizing

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

We fix $\beta_J$and optimize (7) using only the new variable $\beta_j$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 $K_{J\ j}$, 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 6∈ 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 κ = $O(d_{\max})$ without increasing the overall complexity beyond $O(nd^2\max)$. More complex schemes incorporating a kernel cache can also be tried.

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 $\|xi - xj\|^2$can also be cached. 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.

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 (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).

For comparison we also include the GSVC method of Parrado-Hern´andez et al. (2003). This method, originally given for SVM hinge loss, uses the following heuristic criterion to select the next basis function $j*$ 6∈ J:

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

We also tried another criterion, suggested to us by Alex Smola, that is more complex than (8):

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

where d_j is the distance (in feature space) of the j-th training point from the subspace spanned by the elements of J.

4. COMPARISON OF KMP AND SPSVM

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 primitive 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 1 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 primitive 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

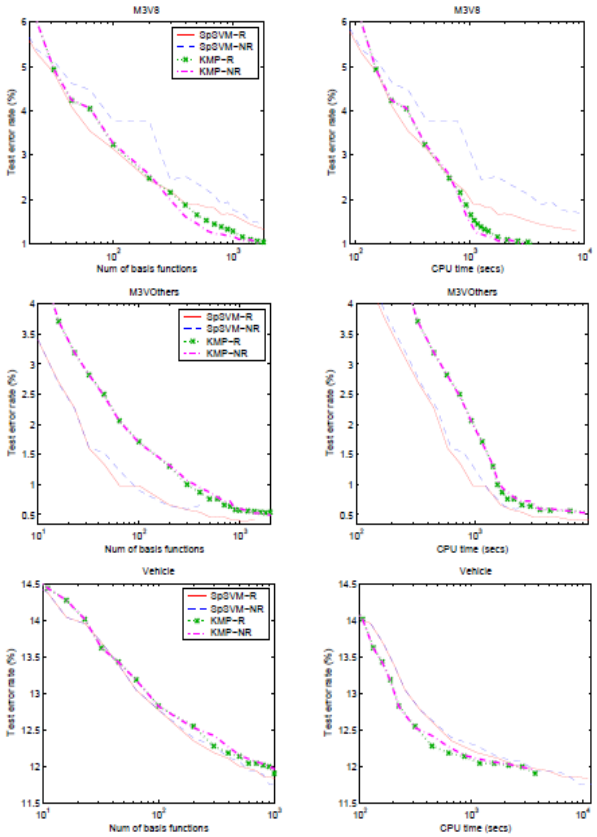

Figure 1: KMP vs SpSVM (with/without regularization)

5. CONCLUSION

In this paper we have given a fast primitive 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. 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 effective.

6. REFERENCES

[1] J. Adler, B. D. Rao, and K. Kreutz-Delgado. Comparison of basis selection methods,1996.

[2] F. Bach and M. Jordan. Predictive low-rank decomposition for kernel methods, 2005.

[3] K. P. Bennett, M. Momma, and M. J. Embrechts. MARK: A boosting algorithm for heterogeneous kernel models,2002.

[4] J. Bi, T. Zhang, and K. P. Bennet. Column generation boosting methods for mixture of kernels. 2004.

[5] C. J. C. Burges and B. Sch¨olkopf. Improving the accuracy and speed of support vector learning machines,1997.

[6] O. Chapelle. Training a support vector machine in the primitive. *Journal of Machine Learning Re-search*, 2005.

[7] D. DeCoste and B. Sch¨olkopf. Training invariant support vector machines. *Machine Learning*, 2002.

[8] T. Downs, K. E. Gates, and A. Masters. Exact simplification of support vector solutions, 2001.

[9] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 2001.

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

[11] S. S. Keerthi and W. Chu. A matching pursuit approach to sparse Gaussian process regression.

[12] S. S. Keerthi and D. DeCoste. A modified finite Newton method for fast solution of large scale linear svms, 2005.