Learning Sparse Kernel Classifiers in the Primal

Zhouyu Fu[1], Guojun Lu[2], Kai-Ming Ting[2], and Dengsheng Zhang[2]

[1] School of Computing, University of Western Sydney, Penrith, NSW 2750, Australia
[2] Gippsland School of IT, Monash University, Churchill, VIC 3842, Australia
z.fu@uws.edu.au, {guojun.lu,kaiming.ting,dengsheng.zhang}@monash.edu

Abstract. The increasing number of classification applications in large data sets demands that efficient classifiers be designed not only in training but also for prediction. In this paper, we address the problem of learning kernel classifiers with reduced complexity and improved efficiency for prediction in comparison to those trained by standard methods. A single optimisation problem is formulated for classifier learning which optimises both classifier weights and eXpansion Vectors (XVs) that define the classification function in a joint fashion. Unlike the existing approach of Wu et al, which performs optimisation in the dual formulation, our approach solves the primal problem directly. The primal problem is much more efficient to solve, as it can be converted to the training of a linear classifier in each iteration, which scales *linearly* to the size of the data set and the number of expansions. This makes our primal approach highly desirable for large-scale applications, where the dual approach is inadequate and prohibitively slow due to the solution of *cubic-time* kernel SVM involved in each iteration. Experimental results have demonstrated the efficiency and effectiveness of the proposed primal approach for learning sparse kernel classifiers that clearly outperform the alternatives.

1 Introduction

Kernel classifiers have been widely used in pattern classification applications due to its superior predictive performance. The major issue with kernel classifiers is the heavy computational cost involved in both training and prediction. While existing methods have mainly focused on reducing the training cost for kernel classifiers especially the Support Vector Machine (SVM) [2], one should not overlook the issue of prediction cost. The reason is that a kernel classifier takes a linear expansion of kernel evaluations for prediction, where the number of expansion terms is usually determined by the size of training data. This makes kernel classifiers quite inefficient for large-scale applications where prediction speed is a main concern, such as classifying pictures, music and web documents in a large collection.

A few methods have been proposed in the literature for learning sparse kernel classifiers with fewer expansion terms in the learned classifiers [3,4,1,5]. The fewer expansions, the sparser the classifier, the smaller number of kernel function evaluations needed, and hence the more efficient the prediction stage is. While early methods such as the Reduced Set (RS) [4] and Reduced SVM (RSVM) [3] focused

G.L. Gimel' farb et al. (Eds.): SSPR & SPR 2012, LNCS 7626, pp. 60–69, 2012.

on pre- and post-processing steps for building sparse kernel classification models, they do not explicitly take into account label information and can result in the loss of discriminant information in fitting the kernel expansions. More recent methods [1,5] adopted a discriminant approach by searching for the expansion vectors (XV) which form the kernel function terms in the resulting classifier so as to maximise the margin and minimise the mis-classification cost. Specifically, Wu et al [1] proposed the Sparse Kernel Learning Algorithm (SKLA), a direct approach for building kernel classifiers with significantly smaller number of XVs and comparable predictive performance to the standard SVM. Compared to the greedy incremental algorithm in [5], SKLA formulates sparse kernel learning in a single optimisation problem and is able to select XVs at arbitrary locations, making it possible for further reduction of the classifier.

A major issue with SKLA is its training complexity. It is an iterative algorithm which involves training a full kernel SVM model in each iteration. This makes it extremely inefficient for large data sets, where SVM training becomes quite costly. On the other hand, it is more likely to have a over-complex kernel classifier with thousands of SVs when we apply standard SVM methods to large data sets. Hence producing sparser classifiers to improve prediction efficiency becomes a real issue for problems with large data sets. For these problems, one needs to have an effective algorithm for sparse kernel classifier learning with low computational cost in *both training and testing*.

The main contribution of this paper is a solution to the above problem that scales well to large data sets and produces sparse prediction models for testing. A similar formulation to [1] is developed using a differentiable loss function. This allows us to tackle the resulting optimisation problem directly in its primal form instead of converting it to the dual form for solution as in [1]. Moreover, with transformation of variables, we are able to convert the primal problem into a standard linear SVM. An iterative technique [6] is then employed to solve the formulated problem, where each iteration only involves solving a linear SVM with significant computational savings particularly for large data. The resulting algorithm, dubbed Primal Sparse Kernel Classifier (PSKC), is shown to perform competitively with SKLA and SVM while being more efficient.

2 Learning Sparse Kernel Classifiers

We focus on kernel SVM in this paper, but the proposed algorithm can easily be adapted to other kernel classifiers. Consider a binary classification problem with data set $(\mathcal{X}, \mathcal{Y}) = \{(\mathbf{x}_i, y_i)|i = 1, \ldots, N, x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}\}$, kernel SVM training can be cast as the following optimisation problem

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i}^{N} \ell(f(\mathbf{x}_i), y_i) \\ & f(\mathbf{x}_i) = \mathbf{w}^T\varphi(\mathbf{x}_i) + b \end{aligned} \tag{1}$$

where φ specifies an implicit feature mapping function. The exact form of φ is unknown but the inner product between two feature maps $\varphi(\mathbf{x}_i)$ and $\varphi(\mathbf{x}_j)$ is

well defined by the kernel function κ such that $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \rangle$ holds for any $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^d$. And $\ell(f_i, y_i) = \max(0, 1 - f_i y_i)$ is the Hinge loss for SVM and varnishes whenever the margin is greater than 1 ($f_i y_i \geq 1$).

Despite the simple form, it is difficult to directly solve problem (1) without knowing φ in closed form. Hence the following dual problem is solved instead

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j \kappa(\mathbf{x}_i, \mathbf{x}_j) - \sum_i \alpha_i y_i \\ s.t. \quad & \sum_i \alpha_i = 0 \qquad \text{and} \qquad 0 \leq \alpha_i y_i \leq C \end{aligned} \tag{2}$$

where α_i is the dual variable for input example i and the primal variable $\mathbf{w}$ and the classifier f in (1) can be expressed in terms of dual variables

$$\mathbf{w} = \sum_{i, \alpha_i \neq 0} \alpha_i \varphi(\mathbf{x}_i) \tag{3}$$

$$f(\mathbf{x}) = \sum_{i, \alpha_i \neq 0} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}) + b \tag{4}$$

Note that only examples with nonzero dual variables are included in the expansion above. These are called Support Vectors (SV), referring to examples incurring positive Hinge losses. The complexity of kernel SVM depends on the number of SVs in the expansion (4). With larger training data set, it is likely to produce a classifier with a large number of SVs. The purpose for sparse kernel classifiers is to reduce the number of expansion terms in (3) and (4) without affecting the performance. The weight vector takes a similar form below

$$\mathbf{w} = \sum_{j=1}^{m} \beta_j \varphi(\mathbf{z}_j) \tag{5}$$

where $\mathbf{z}_i$'s are the eXpansion Vectors (XV) that form the bases of $\mathbf{w}$. Unlike SVs in kernel SVM, XVs do not necessarily overlap with input data and thus provide more flexibility in fitting the weight $\mathbf{w}$. The number of XVs m is much smaller than the number of SVs in standard SVM, making predictions more efficient.

Various strategies can be used for selecting XVs here, such as random selection from the input examples (RSVM, [3]) or fitting a trained SVM classifier with a fixed number of XVs that minimises the reconstruction error (RS, [4]). A more principled approach would account for the cost function to be minimised and embed XV selection into the optimisation process. Wu et al [1] added Equation (5) as an explicit constraint into the SVM formulation in (1). They then showed that a new dual problem can be formulated for sparse SVM, resembling the standard formulation in (2) with a modified kernel function

$$\hat{\kappa}_z(\mathbf{x}_i, \mathbf{x}_j) = \psi_i^T (\mathbf{K}^z)^{-1} \psi_j \tag{6}$$

$$\psi_i = [\kappa(\mathbf{x}_i, \mathbf{z}_1), \ldots, \kappa(\mathbf{x}_i, \mathbf{z}_m)]^T \tag{7}$$

where $\mathbf{K}^z$ is a $m \times m$ Gram matrix whose (i, j)th entry is given by the kernel evaluation $\kappa(\mathbf{z}_i, \mathbf{z}_j)$. Each entry $\hat{\kappa}_z(\mathbf{x}_i, \mathbf{x}_j)$ depends on the XVs $\mathbf{z}_i$'s. This suggests

the use of a perturbed optimisation technique [6] to solve the formulated problem iteratively. In each iteration, the gradient of the dual objective function with respect to $\mathbf{z}_i$ is computed by solving the dual SVM problem with modified kernel (6) and fixing the dual variables to their optimal values in gradient computation as if they do not depend on the XVs. Then a line search is pursued in the direction of the negative gradient for sufficient decrease of cost function value. Each linear search updates the values of $\mathbf{z}_i$'s and hence involves retraining of the kernel SVM. The validity of this approach is established by a theorem for optimal value functions in [6].

3 Primal Sparse Kernel Classifier Learning

Despite the effectiveness of SKLA [1] for learning sparse kernel classifiers, it is extremely expensive in training, making it impractical for large-scale applications. The complexity of SKLA arises mainly in two aspects. Firstly, being an algorithm of iterative nature, SKLA involves repeatedly retraining of the kernel SVM problem for each function evaluation involved in gradient computation and line search. Note the complexity of kernel SVM training is at best cubic in the number of SVs, which is roughly proportional to the training data size. In addition, for difficult problems, there is a high probability of failure for line searches. This would greatly increase the number of times for SVM retraining and the overall training time. Secondly, the gradient computation is also very costly, as it involves taking the derivative of each $\hat{\kappa}_z(\mathbf{x}_i, \mathbf{x}_j)$ in the modified kernel w.r.t. $\mathbf{z}_j$'s. The time complexity for computing the gradient of a single XV is $O(N^2m^2d)$, quadratic in both the training data size and the number of XVs. This is also undesirable for large-scale applications.

The main hurdle to the efficiency of SKLA is the optimisation of dual variables, which are then used to compute the β variables that define the weight vector in (5). A more direct approach would aim to solve β_i's directly. This motivates the development of the PSKC algorithm, which provides a primal optimisation framework for sparse kernel classifier learning. By substituting weight vector $\mathbf{w}$ in (5) into the primal problem in (1) and rewriting the cost function in terms of β and $\mathbf{z}_j$'s, we have

$$\min_{\beta,b;\mathbf{Z}} \qquad f(\beta, b; \mathbf{Z}) = \frac{1}{2}\beta^T \mathbf{K}^z \beta + C \sum_i^N \ell(\beta^T \psi_i + b, y_i) \tag{8}$$

Here $\beta = [\beta_1, \ldots, \beta_m]^T$ is the vector of expansion coefficients, ψ_i is defined in (7), and $\mathbf{Z}$ is a concatenation of XVs $\mathbf{z}_j$'s. Moreover, the squared Hinge loss is used here $\ell(f_i, y_i) = \max(0, 1 - f_i y_i)^2$ instead of the Hinge loss for standard SVM. The reason for utilising the squared Hinge loss will become apparent shortly.

Two sets of variables need to be optimised for the above problem, the expansion coefficient vector β and the XVs $\mathbf{Z}$. Hence, we adopt an efficient approach to solve this joint optimisation problem. Specifically, we convert the original problem in (8) into the following problem which depends on variable $\mathbf{Z}$ only

$$\min_{\mathbf{Z}} g(\mathbf{Z}) \qquad \text{with} \qquad g(\mathbf{Z}) = \min_{\beta,b} f(\beta, b; \mathbf{Z}) \tag{9}$$

The new cost function $g(\mathbf{Z})$ is special because itself is the optimal value of f minimised over variables (β, b). In our case, $g(\mathbf{Z})$ not only exists, but is also differentiable at each $\mathbf{Z}$. This result can be established by applying Theorem 4.1 of [6], which provides sufficient conditions for the existence of derivatives for optimal value functions like $g(\mathbf{Z})$. According to the theorem, $g(\mathbf{Z})$ is differentiable if $f(\beta, b; \mathbf{Z})$ is differentiable w.r.t. β and b and has unique optimal value over variables β and b for each given $\mathbf{Z}$. The first condition is guaranteed by the use of squared Hinge loss discussed before, which is differentiable everywhere. Note this condition is not true for the standard Hinge loss, as it is non-differentiable if $y_i f_i = 1$. The uniqueness condition is true because f is a quadratic function over β with positive definite Hessian $\mathbf{K}^z$. Thus f is convex in β and ensures the optimal solution is unique. Let $(\overline{\beta}, \overline{b}) = \arg\min f(\beta, b; \mathbf{Z})$ be the minimiser of f at given $\mathbf{Z}$, the derivative of $g(\mathbf{Z})$ w.r.t. each XV $\mathbf{z}_j$ can then be computed by substituting $\overline{\beta}$ and $\overline{b}$ into (8) and taking the corresponding derivative as if $g(\mathbf{Z})$ does not depend on $\overline{\beta}$ and $\overline{b}$

$$\frac{\partial g}{\partial \mathbf{z}_j} = \sum_{i=1} m\overline{\beta}_i \frac{\partial \kappa(\mathbf{z}_i, \mathbf{z}_j)}{\partial \mathbf{z}_j} \overline{\beta}_j + 2C \sum_{i \in \mathcal{S}} (\beta^T \psi_i + b - y_i) \beta_j \frac{\partial \kappa(\mathbf{x}_i, \mathbf{z}_j)}{\partial \mathbf{z}_j} \tag{10}$$

where $\mathcal{S} = \{i | \ell(y_i, f_i) > 0\}$ denotes the index set of examples with positive loss terms. The partial derivative of the kernel function depends on the choice of the kernel. In this paper, we have adopted the following Gaussian kernel, but the algorithm works for all differentiable kernel functions

$$\begin{aligned} \kappa(\mathbf{x}, \mathbf{z}) &= \exp\left(-\gamma \|\mathbf{x} - \mathbf{z}\|^2\right) \\ \frac{\partial \kappa(\mathbf{x}, \mathbf{z})}{\partial \mathbf{z}} &= 2\gamma \kappa(\mathbf{x}, \mathbf{z})(\mathbf{x} - \mathbf{z}) \end{aligned} \tag{11}$$

With the derivatives of $g(\mathbf{Z})$, we can use a gradient descent algorithm with back-tracing line search to iteratively optimise the values of $\mathbf{Z}$. The only problem left is how to evaluate the value of $g(\mathbf{Z})$ for each $\mathbf{Z}$, which equals solving the minimisation problem in (8) for β and b with fixed $\mathbf{Z}$. This problem is in fact equivalent to a linear SVM with the one-to-one mapping of variables below.

$$\vartheta = (\mathbf{K}^z)^{\frac{1}{2}} \beta \qquad \Longleftrightarrow \qquad \beta = (\mathbf{K}^z)^{-\frac{1}{2}} \vartheta \tag{12}$$

By substituting β in (8), we can rewrite it as a cost function over ϑ

$$f'(\vartheta, b; \mathbf{Z}) = \vartheta^T \vartheta + C \sum_{i}^{N} \ell(\vartheta^T (\mathbf{K}^z)^{-\frac{1}{2}} \psi_i + b, y_i) \tag{13}$$

The above is the same as the cost function for a linear SVM. ϑ is the weight vector of the linear SVM, and $(\mathbf{K}^z)^{-\frac{1}{2}} \psi_i$'s are the input vectors. Let $(\overline{\vartheta}, \overline{b})$ be the solution of the linear SVM trained with transformed feature vectors, the minimiser $(\overline{\beta}, \overline{b})$ for $f(\beta, b; \mathbf{Z})$ can be easily obtained by mapping the solution $\overline{\vartheta}$ back to $\overline{\beta}$ via (12).

Same as the case of SKLA, the complexity of PSKC depends mainly on function evaluation and gradient computation. We have shown above that function evaluation is equal to solving a linear SVM with transformed features. Both linear SVM training and feature transformation has linear time complexity $O(N)$ with input data size N. This is much better than SKLA with time complexity of $O(N^3)$ for kernel SVM training. Since m is a small number compared to N, the computation of $(\mathbf{K}^z)^{-\frac{1}{2}}$ and map from ϑ to β is negligible. From (10), we can see that the cost of gradient computation for each XV $\mathbf{z}_j$ is roughly $O(Nmd)$, a factor-$O(Nm)$ saving compared to that in SKLA.

4 Experimental Results

We first tested our PSKC algorithm on a synthetic example to showcase its interesting properties. The synthetic data set shown in Figure 1. has four classes, each occupying a separate cluster generated from a Gaussian distribution. Points from different classes are marked with different symbols. For this example, a minimum of 4 XVs overlapping with the centroid of each cluster is sufficient to distinguish the four classes. We deliberately initialise PSKC with poor initial locations of the XVs far from their respective cluster centroids, as denoted by squares in the top-left plot. By running PSKC and recording the locations of XVs over each iteration, a trajectory is created for each XV which keeps track of its evolution during optimisation. It can be seen that eventually all XVs have converged to locations close to the cluster centroids, as denoted by the circles in the same plot. The improvement on XVs locations is a natural consequence of the reduction in cost function values over each iteration, as shown in the top-right plot. These plots have empirically demonstrated the effectiveness of PSKC in finding good XVs for sparse kernel classifiers.

We have obtained similar results with the dual SKLA algorithm [1]. However, SKLA is much more costly than PSKC in training. To show this, we conducted two experiments on the same data distributions. The first experiment compares the training speed of PSKC and SKLA by increasing the size of the training data while the second one focuses on the effect of increasing feature dimensions. The results are shown on the second row of Figure 1, with the number of seconds spent on training over different training data sizes in the left plot and increasing feature dimensions on the right for both PSKC ((in solid lines) and SKLA (in broken lines). It can be easily seen that PSKC has superior scalability in comparison to SKLA in both cases. This is especially true with large training data sizes. Whereas SKLA has cubic time complexity with the sample size, PSKC scales linearly and is well suited for large-scale applications.

We now turn our attention to real-world data sets. 12 data sets from the UCI Machine Learning Repository[1] were used in our experiment. A summary of data sets used is given in Table 1, including the size of the training (Ntr) and testing (Nts) sets, feature dimension (Dim), number of classes (Cls). We then applied the

[1] `http://archive.ics.uci.edu/ml/`

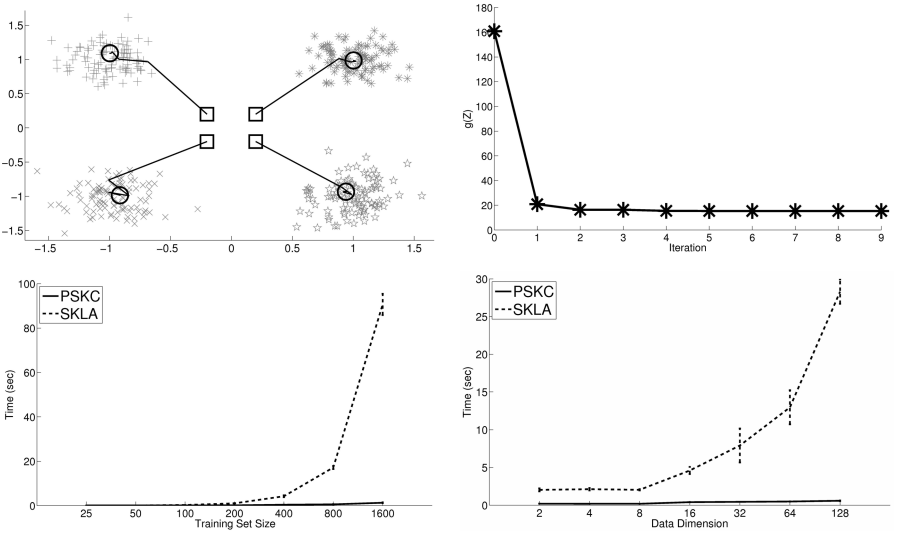

Fig. 1. Results on synthetic data. Top row: trajectories for the locations of each XV (left) and cost function values over the iterations (right); Bottom row: training speed for PSKC and SKLA with increasing data sizes (left) and feature dimensions (right).

standard SVM with the Gaussian kernel on each data set and compared the results of SVM with four candidate sparse kernel classifiers - RS [4], RSVM [3], the dual SKLA [1] and the proposed PSKC. Input data were scaled to have unit variance for each coordinate. The parameter γ for the Gaussian kernel was empirically set to the inverse of the feature dimension and the SVM parameter C was 10 for all methods under comparison. We have used the LibSVM package[2] for training kernel SVM classifiers and the LibLinear package[3] for training linear SVMs for RSVM and PSKC. The algorithms was implemented in Octave on a MacBook Pro with Intel Core i5 CPU and 4Gb memory. We have repeated the experiment 10 times over random partitions of training and testing data. The classification performance in terms of average accuracy values and their standard deviations are reported in Table 2.

For the SVM classifier, we have also recorded the number of SVs produced on average for each data set, which are the numbers in the brackets on the second column of Table 2. For each sparse kernel classifier, we have used a fixed number of expansions equal to 1% of the training data size capped at a minimum value of 10 and a maximum value of 100. The exact number for each data set is shown inside the brackets following the names on the first column. For each data set, we have highlighted the accuracy value corresponding to the best-performing sparse kernel classifier. More than one values could be highlighted in cases of ties, which are determined by the results of paired t-tests at the confidence interval of 95%.

[2] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`
[3] `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`

Table 1. Statistics of the real-world data sets used in the comparison experiment

Data	Ntr	Nts	Dim	Cls	Data	Ntr	Nts	Dim	Cls
australian	346	344	14	2	letter	15000	5000	16	26
breast-cancer	342	341	10	2	connect-4	33780	33777	126	3
dna	1001	999	180	3	shuttle	43500	14500	9	7
segment	1155	1155	19	7	ijcnn1	49990	91701	22	2
satimage	4435	2000	36	6	mnist	60000	10000	784	10
usps	7291	2007	256	10	SensIT	78823	19705	100	3

We can clearly see from Table 2 that PSKC and SKLA are the most competitive sparse kernel classifiers. Their performances could approach that of kernel SVM albeit with a significantly smaller number of kernel expansions. In contrast, RS and RSVM do not perform as well as PSKC and SKLA. Specifically, among the four sparse kernel classifiers under comparison, PSKC is the exclusive winner for 6 out of 12 data sets, and winning 10 of them including tied cases. Moreover, for the majority of data sets, PSKC beats RSVM by a large margin in performance. This is clear evidence for the effectiveness of the PSKC optimisation algorithm, as RSVM is the special case of PSKC without any optimisation.

Table 2. Comparison of classification performance for SVM and various sparse kernel classifiers on real-world data sets

	SVM	RS	RSVM	SKLA	PSKC
australian (10)	84.2 ± 1.4(156)	82.2 ± 3.1	77.9 ± 2.6	$\mathbf{85.7 \pm 2.1}$	$\mathbf{85.8 \pm 1.6}$
breast-cancer (10)	95.2 ± 0.8(56)	94.6 ± 1.6	$\mathbf{96.4 \pm 0.7}$	$\mathbf{96.1 \pm 0.9}$	96.0 ± 0.8
dna (10)	95.3 ± 0.6(821)	$\mathbf{94.7 \pm 0.4}$	67.0 ± 2.5	94.3 ± 0.6	94.2 ± 0.5
segment (12)	94.4 ± 0.6(384)	71.3 ± 7.7	85.7 ± 2.5	91.7 ± 1.1	$\mathbf{92.9 \pm 0.9}$
satimage (44)	91.0 ± 0.7(1425)	$\mathbf{89.8 \pm 0.8}$	88.5 ± 1.0	$\mathbf{90.2 \pm 0.8}$	$\mathbf{90.4 \pm 0.6}$
usps (73)	97.8 ± 0.8(2175)	96.1 ± 1.0	94.4 ± 1.0	$\mathbf{96.9 \pm 0.8}$	$\mathbf{97.0 \pm 0.9}$
letter (100)	96.5 ± 0.4(6260)	76.9 ± 1.9	88.3 ± 0.3	-	$\mathbf{94.6 \pm 0.3}$
connect-4 (100)	83.7 ± 0.1(16721)	$\mathbf{80.3 \pm 0.7}$	74.5 ± 0.3	-	$\mathbf{80.1 \pm 0.5}$
shuttle (100)	99.8 ± 0.0(605)	98.8 ± 1.1	99.8 ± 0.0	-	$\mathbf{99.8 \pm 0.0}$
ijcnn1 (100)	99.0 ± 0.1(2683)	$\mathbf{98.5 \pm 0.2}$	96.4 ± 0.2	-	$\mathbf{98.5 \pm 0.2}$
mnist (100)	97.7 ± 0.1(9883)	94.3 ± 0.4	90.5 ± 0.4	-	$\mathbf{95.8 \pm 0.2}$
SensIT (100)	84.0 ± 0.2(21249)	82.0 ± 0.8	80.0 ± 0.6	-	$\mathbf{83.6 \pm 0.4}$

Though the performances of PSKC and SKLA are quite comparable, there is a huge difference in training cost. SKLA is very inefficient and scales poorly to large data, hence we only have the results for SKLA in Table 2 for the first six data sets. For the remaining data sets with over $10,000$ training examples, the training cost is prohibitive for SKLA, which involves many iterations of kernel SVM training and heavy gradient computations. The training times for each method are reported in Table 3. It is evident that SKLA is very inefficient in training. On the other hand, PSKC scales much better with increasing training sizes. After all,

Table 3. Comparison of training time for different methods

	SVM	RS	RSVM	SKLA	PSKC
australian	0.02 sec	0.35 sec	0.01 sec	1.0 sec	0.2 sec
breast-cancer	0.01 sec	0.39 sec	0.01 sec	0.4 sec	0.1 sec
dna	1.2 sec	2.9 sec	0.01 sec	$\approx$ 4.2 min	0.5 sec
segment	0.2 sec	2.5 sec	0.02 sec	31.1 sec	0.6 sec
satimage	2.5 sec	27.9 sec	0.2 sec	$\approx$ 47.0 min	9.0 sec
usps	32.9 sec	$\approx$ 4.7 min	0.9 sec	$>$ 3 hour	38.4 sec
letter	52.1 sec	$\approx$ 7.0 min	6.2 sec	-	$\approx$ 3.7 min
connect-4	$\approx$ 13.4 min	$\approx$ 21.0 min	3.0 sec	-	$\approx$ 4.7 min
shuttle	9.7 sec	$\approx$ 1.0 min	5.3 sec	-	$\approx$ 2.3 min
ijcnn1	31.2 sec	$\approx$ 1.3 min	1.9 sec	-	$\approx$ 1.3 min
mnist	$\approx$ 1.5 hour	$\approx$ 2 hour	12.3 sec	-	$\approx$ 12.8 min
SensIT	$\approx$ 2 hour	$\approx$ 2.5 hour	18.6 sec	-	$\approx$ 11.4 min

it only involves repeated training of linear SVM during optimisation. Although it is not as efficient as SVM for small and median data sets, the asymptotic complexity for PSKC is better. This is empirically justified by the less amount of time spent for training on two largest data sets - "mnists" and "SensIT" in Table 3. In addition, PSKC uses only 10% to 1% of XVs compared to the number of SVs in SVM. This reduces prediction time by a factor of 10 to 100, which depends linearly on the number of XVs in the classifier. The improvement in efficiency for PSKC is more pronounced for large data sets compared to SVM. This makes PSKC particularly suited for large-scale applications.

5 Conclusions

In this paper we presented PSKC, an efficient and effective algorithm for learning sparse kernel classifiers. Training PSKC is quite efficient and only involves solving linear classifiers repeatedly. Experiments show that PSKC outperforms other sparse kernel classifiers and is comparable with kernel SVM in predictive accuracy at lower training and prediction costs for large data sets.

Acknowledgment. The work was partly supported by the Australian Research Council under the Discovery Project "Automatic music feature extraction, classification and annotation" (DP0986052).

References

1. Wu, M., Scholkopf, B., Bakir, G.: A direct method for building sparse kernel learning algorithms. Journal of Machine Learning Research 7, 603–624 (2006)
2. Platt, J.: Fast training of support vector machines using sequential minimal optimization. In: Advances in Kernel Methods - Support Vector Learning (1998)

3. Lee, Y.J., Mangasarian, O.L.: Rsvm: Reduced support vector machines. In: Siam Data Mining Conf. (2001)
4. Scholkopf, B., Smola, A.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press (2002)
5. Keerthi, S., Chapelle, O., DeCoste, D.: Building support vector machines with reduced classifier complexity. Journal Machine Learning Res. 7, 1493–1515 (2006)
6. Bonnans, J.F., Shapiro, A.: Optimization problems with pertubation: A guided tour. SIAM Review 40(2), 202–227 (1998)