Sparse Kernel SVMs via Cutting-Plane Training

Thorsten Joachims and Chun-Nam John Yu

Cornell University, Dept. of Computer Science, Ithaca, NY 14853 USA.
{tj|cnyu}@cs.cornell.edu

Abstract. We explore an algorithm for training SVMs with Kernels that can represent the learned rule using arbitrary basis vectors, not just the support vectors (SVs) from the training set. This results in two benefits. First, the added flexibility makes it possible to find sparser solutions of good quality, substantially speeding-up prediction. Second, the improved sparsity can also make training of Kernel SVMs more efficient, especially for high-dimensional and sparse data (e.g. text classification). This has the potential to make training of Kernel SVMs tractable for large training sets, where conventional methods scale quadratically due to the linear growth of the number of SVs. In addition to a theoretical analysis of the algorithm, we also present an empirical evaluation.

1 Introduction

While Support Vector Machines (SVMs) with kernels offer great flexibility and prediction performance on many application problems, their practical use is often hindered by the following two problems. Both problems can be traced back to the number of Support Vectors (SVs), which is known to generally grow linearly with the data set size [1]. First, training is slower than other methods and linear SVMs, where recent advances in training algorithms vastly improved training time. Second, since the prediction rule takes the form $h(\boldsymbol{x}) = \text{sign}\left[\sum_{i=1}^{\#SV} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x})\right]$, it is too expensive to evaluate in many applications when the number of SVs is large.

This paper tackles these two problems by generalizing the notion of Support Vector to arbitrary points in input space, not just training vectors. Unlike Wu et al. [2], who explore making the location of the points part of a large non-convex optimization problem, we propose an algorithm that iteratively constructs the set of basis vectors from a cutting-plane model. This makes our algorithm, called *Cutting-Plane Subspace Pursuit* (CPSP), efficient and modular. We analyze the training efficiency and the solution quality of the CPSP algorithm both theoretically and empirically. We find that its classification rules can be orders of magnitude sparser than the conventional support-vector representation while providing comparable prediction accuracy. The sparsity of the CPSP representation not only makes predictions substantially more efficient, it also allows the user to control training time. Especially for large datasets with sparse feature vectors (e.g. text classification), the CPSP methods is substantially faster than methods that only consider basis vectors from the training set.

2 Related Work

Most existing algorithms for training kernel SVMs follow the Representer Theorem and search for the optimal weight vector in the span of the training vectors $\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i \phi(\boldsymbol{x}_i)$. This includes decomposition methods [3, 4] and all other dual approaches. To overcome the problems resulting from the growing number of support vectors, Burges and Schölkopf [5] propose to post-process their solution and replace the support vector expansion with an approximation that is more sparse. Clearly, this can improve only the prediction efficiency, while it is still necessary to compute a full solution during training. For large datasets, this is intractable.

An alternative to post-processing are methods for selecting a set of basis vectors a priori. This includes sampling randomly from the training set in the NYSTROM method [6], greedily minimizing reconstruction error [7], and variants of the Incomplete Cholesky factorization [8, 9]. However, these selection methods are not part of the optimization process, which makes a goal-directed choice of basis vectors difficult. In fact, all but [9] ignore label information, and all methods are limited to selecting basis vectors from the training set.

Methods like the Core Vector Machine (CVM) [10], the Ball Vector Machine (BVM) [11], and the active selection strategy of the LASVM method [12] greedily select which basis vectors to include in the classification rule. While they allow the user to sacrifice solution quality to gain sparsity and training efficiency, they are also limited to selecting basis vectors from the training set.

Another set of methods are basis pursuit approaches [13, 14]. They repeatedly solve the optimization problem for a given set of basis vector, and then greedily search for vectors to add or remove. The Cutting-Plane Subspace Pursuit method we propose is similar in the respect that it iteratively constructs the basis set. However, the construction of the basis set is part of the optimization algorithm itself, and the cutting-plane model makes it straightforward to add basis vectors that are not in the training set. It is not clear how to efficiently add such general basis vectors in other basis pursuit approaches.

The method most closely related to ours was proposed in [2]. They treat the basis vectors as variables in the SVM optimization problem, and solve the resulting non-convex program via gradient descent to a local optimum. However, training efficiency is a bottleneck in this approach and they focus only on small datasets in their evaluation. We will consider datasets that are several orders of magnitude larger. Furthermore, we will provide theoretical results giving insight into the quality of the CPSP solution.

3 Cutting-Plane Algorithm for SVMs

We first introduce the Cutting-Plane Algorithm for training SVMs [15, 16], since it is the basis for the CPSP algorithm proposed in this paper. For a training sample, $(x_1, y_1), ..., (x_n, y_n)$, the following is a general formulation of the large-margin training problem for learning a rule $h : \mathcal{X} \rightarrow \mathcal{Y}$ mapping from some input

space $\mathcal{X}$ to some output space $\mathcal{Y}$ [17]. For different choices of the joint feature map $\Psi(x, y)$ and the loss function $\Delta(y, \hat{y})$, it can be specialized to classification SVMs, to Maximum-Margin Markov Networks, or various structured prediction problems.

$$\min_{\boldsymbol{w},\xi} \frac{1}{2}\langle \boldsymbol{w}, \boldsymbol{w}\rangle + \frac{C}{n}\sum_{i=1}^{n}\xi_i \quad s.t. \forall i, \forall \hat{y} \in \mathcal{Y}\colon \langle \boldsymbol{w}, \Psi(x_i, y_i) - \Psi(x_i, \hat{y})\rangle \geq \Delta(y_i, \hat{y}) - \xi_i \quad (1)$$

$\langle ., . \rangle$ denotes an inner product. For the sake of simplicity, this paper only deals with the special case of binary classification with $\mathcal{X} = \Re^N$ and $\mathcal{Y} = \{-1, +1\}$, where the joint feature map is $\Psi(\boldsymbol{x}, y) = \frac{1}{2} y \boldsymbol{x}$ (or $\Psi(\boldsymbol{x}, y) = \frac{1}{2} y \phi(\boldsymbol{x})$ for the kernel $K(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}')\rangle$) and where the loss function $\Delta(y, \hat{y})$ is the zero/one-loss. In this case, it is easy to verify that (1) is equivalent to the following program, which corresponds to a binary classification SVM without explicit offset.

$$\min_{\boldsymbol{w},\xi} \frac{1}{2}\langle \boldsymbol{w}, \boldsymbol{w}\rangle + \frac{C}{n}\sum_{i=1}^{n}\xi_i \quad s.t. \forall i : y_i \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle \geq 1 - \xi_i$$

3.1 Linear SVMs

Instead of solving (1) directly, [15] proposes to solve the following equivalent program.

$$\min_{\boldsymbol{w},\xi} \frac{1}{2}\langle \boldsymbol{w}, \boldsymbol{w}\rangle + C\xi \quad (2)$$

$$s.t. \ \forall \hat{y}_1 ... \hat{y}_n \in \mathcal{Y}^n : \left\langle \boldsymbol{w}, \frac{1}{n}\sum_{i=1}^{n}(\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)) \right\rangle \geq \frac{1}{n}\sum_{i=1}^{n}\Delta(y_i, \hat{y}_i) - \xi$$

This program has only a single slack variable ξ, and is therefore called the 1-slack formulation. It is shown in [15] that any $\boldsymbol{w}$ solving (2) is also a solution of (1), and that $\xi = \frac{1}{n}\sum_{i=1}^{n}\xi_i$. While (2) has a huge number of constraints, Algorithm 1 is a cutting-plane procedure that always constructs a solution of precision ϵ with at most $O(\frac{C}{\epsilon})$ active constraints [15, 18, 16]. In the experiments from Section 6, the number of active constraints was typically around 30 – independent of the size of the training set.

Algorithm 1 maintains a working set of m constraints $\left\langle \boldsymbol{w}, \bar{\boldsymbol{\Psi}}_i \right\rangle \geq \bar{\Delta}_i - \xi$ over which it solves the QP in Line 4. In each iteration I, the algorithm finds the most violated constraint from (2) (Lines 5-7) and adds it to the working set, so $m \leq I$. Typically, however, $m << I$, since constraints become inactive in later iterations and can be removed from the working set (Line 8). Therefore, the size m of the working set is roughly equal to the number of active constraints (i.e. $m \approx 30$). The algorithm is known to need at most $I \in O(\frac{C}{\epsilon})$ iterations to converge to an ϵ-accurate solution [15, 18, 16]. This means that the number of iterations is independent of the number of training examples n and the number of features N.

Algorithm 1 Cutting-Plane Algorithm for Structural SVM (primal)

1: Input: $S = ((x_1, y_1), \ldots, (x_n, y_n))$, C, ϵ
2: $\bar{\Delta} \leftarrow \mathbf{0}$, $\bar{\boldsymbol{\Psi}} \leftarrow \mathbf{0}$, $m \leftarrow 0$
3: **repeat**
4: $(\boldsymbol{w}, \xi) \leftarrow \operatorname{argmin}_{\boldsymbol{w}, \xi \geq 0} \frac{1}{2} \langle \boldsymbol{w}, \boldsymbol{w} \rangle + C\xi$
s.t. $\forall i : \left\langle \boldsymbol{w}, \bar{\boldsymbol{\Psi}}_i \right\rangle \geq \bar{\Delta}_i - \xi$
5: **for** i=1,...,n **do**
6: $\hat{y}_i \leftarrow \operatorname{argmax}_{\hat{y} \in \mathcal{Y}} \{\Delta(y_i, \hat{y}) + \boldsymbol{w}^T \Psi(x_i, \hat{y})\}$
7: **end for**
8: $(\bar{\boldsymbol{\Psi}}, \bar{\Delta}, m) = \mathit{remove\_inactive}(\bar{\boldsymbol{\Psi}}, \bar{\Delta}, \boldsymbol{w}, \xi)$
9: $m \leftarrow m + 1$
10: $\bar{\boldsymbol{\Psi}}_m \leftarrow \frac{1}{n} \sum_{i=1}^{n} [\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)]$
11: $\bar{\Delta}_m \leftarrow \frac{1}{n} \sum_{i=1}^{n} \Delta(y_i, \hat{y}_i)$
12: **until** $\left\langle \boldsymbol{w}, \bar{\boldsymbol{\Psi}}_m \right\rangle \geq \bar{\Delta}_m - \xi - \epsilon$
13: return($\boldsymbol{w}$,ξ)

3.2 SVMs with Kernels

While originally proposed for linear SVMs, the cutting-plane method can be extended to the non-linear case with kernels. Since $\boldsymbol{w}$ now lies in the Reproducing Kernel Hilbert Space (RKHS) of the kernel $K(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle$, we need to move to the dual representation. Algorithm 2 is this dual variant of Algorithm 1 and specialized to the case of binary classification as implemented in the SVM^{perf} software. It replaces the primal QP with its Wolfe dual in Line 5, and the solution vector in the RKHS is $\boldsymbol{w} \equiv \sum_i \alpha_i \bar{\boldsymbol{\Psi}}_i$. Note that sums cannot be computed efficiently in the RKHS. Therefore, the assignment operator $\leftarrow$ is replaced with a rewrite operator $\equiv$ where appropriate. However, it is easy to verify that all inner products (Lines 4, 9, 15) can be computed as sums of kernel evaluations. The $O(\frac{C}{\epsilon})$ bound on the number of iterations [15, 16] holds independent of whether a kernel is used or not, but how does the time complexity per iteration change when moving from a linear to a kernelized SVM?

Without kernels, any iteration in Algorithm 2 takes at most $O(m^3)$ for solving the QP, $O(m^2)$ for ξ, $O(mN)$ for $\boldsymbol{w}$, $O(nN)$ for computing the most violated constraint, $O(n)$ for $\bar{\Delta}$, $O(nN)$ for $\bar{\boldsymbol{\Psi}}$, and $O(mN)$ for adding a row/column to $\boldsymbol{H}$. So the overall time complexity is $O(m^3 + mN + nN)$.

When using a kernel, however, computing $\langle \boldsymbol{w}, \phi(\boldsymbol{x}_i) \rangle$ and $\boldsymbol{H}$ becomes more expensive than in the linear case. Denote with $\boldsymbol{Y}$ the matrix with $\boldsymbol{Y}_{ij} = (y_i - \hat{y}_i)$ for the j-the constraint in $\bar{\boldsymbol{\Psi}}$. To find the most violated constraint, for each example one now needs to evaluate

$$\langle \boldsymbol{w}, \phi(\boldsymbol{x}_i) \rangle = \sum_{k=1}^{n} \left(\sum_{j=1}^{m} \boldsymbol{\alpha}_j \boldsymbol{Y}_{kj} \right) K(\boldsymbol{x}_i, \boldsymbol{x}_k). \tag{3}$$

Algorithm 2 Cutting-Plane Algorithm for Classification SVM (dual)

Input: $S = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n))$, C, ϵ, $K(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle$
$\bar{\Delta} \leftarrow \mathbf{0}$, $\bar{\boldsymbol{\Psi}} \leftarrow \mathbf{0}$, $\boldsymbol{H} \leftarrow \mathbf{0}$,$m \leftarrow 0$
repeat
  $\boldsymbol{H} \leftarrow (\boldsymbol{H}_{ij})_{1 \le i,j \le m}$, where $\boldsymbol{H}_{ij} = \left\langle \bar{\boldsymbol{\Psi}}_i, \bar{\boldsymbol{\Psi}}_j \right\rangle$
  $\boldsymbol{\alpha} \leftarrow \text{argmax}_{\boldsymbol{\alpha} \ge 0} \boldsymbol{\alpha}^T \bar{\Delta} - \frac{1}{2} \boldsymbol{\alpha}^T \boldsymbol{H} \boldsymbol{\alpha}$ s.t. $\boldsymbol{\alpha}^T \mathbf{1} \le C$
  $\xi \leftarrow \frac{1}{C}(\boldsymbol{\alpha}^T \bar{\Delta} - \boldsymbol{\alpha}^T \boldsymbol{H} \boldsymbol{\alpha})$
  $\boldsymbol{w} \equiv \sum_i \alpha_i \bar{\boldsymbol{\Psi}}_i$
  for i=1,...,n **do**
    $\hat{y}_i \leftarrow \text{sign}(\langle \boldsymbol{w}, \phi(\boldsymbol{x}_i) \rangle - y_i)$
  end for
  $(\bar{\boldsymbol{\Psi}}, \bar{\Delta}, m) = remove\_inactive(\bar{\boldsymbol{\Psi}}, \bar{\Delta}, \boldsymbol{\alpha})$
  $m \leftarrow m + 1$
  $\bar{\boldsymbol{\Psi}}_m \equiv \frac{1}{2n} \sum_{i=1}^{n} (y_i - \hat{y}_i) \phi(\boldsymbol{x}_i)$
  $\bar{\Delta}_m \leftarrow \frac{1}{2n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$
until $\left\langle \boldsymbol{w}, \bar{\boldsymbol{\Psi}}_m \right\rangle \ge \bar{\Delta}_m - \xi - \epsilon$
return($\boldsymbol{w}$,ξ)

Over all n examples, this has a cost of $O(n^2+mn)$. Similarly, adding a row/column for the new $\bar{\boldsymbol{\Psi}}_m$ to the Gram matrix $\boldsymbol{H}$ now requires computing

$$\forall i : \boldsymbol{H}_{mi} = \boldsymbol{H}_{im} = \left\langle \bar{\boldsymbol{\Psi}}_i, \bar{\boldsymbol{\Psi}}_m \right\rangle = \sum_{k=1}^{n} \sum_{l=1}^{n} \boldsymbol{Y}_{ki} \boldsymbol{Y}_{lm} K(\boldsymbol{x}_k, \boldsymbol{x}_l) \tag{4}$$

This takes time $O(mn^2)$, counting a single kernel evaluation as $O(1)$. So, the overall time complexity of an iteration when kernels are used is $O(m^3 + mn^2)$. This $O(n^2)$ scaling is not practical for any reasonably-sized dataset, and the algorithm has worse constants than decomposition methods like SVM^{light} that also typically scale $O(n^2)$. However, Algorithm 2 does provide a path to a substantially more efficient algorithm that is explored in the next section.

4 Cutting-Plane Subspace Pursuit

Is it possible to remove the $O(n^2)$ scaling behavior? Here is the intuition for the approach we take. A property of the cutting-plane algorithm is that it iteratively constructs a low-dimensional subspace $\mathcal{W} = \text{span}(\bar{\boldsymbol{\Psi}}_1, ..., \bar{\boldsymbol{\Psi}}_m) = \{\sum_{i=1}^{m} \beta_i \bar{\boldsymbol{\Psi}}_i : \boldsymbol{\beta} \in \Re^m\}$ in which the final solution

$$\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i \bar{\boldsymbol{\Psi}}_i \tag{5}$$

is guaranteed to lie. Instead of using the Representer Theorem and considering the larger subspace $\mathcal{F} = \text{span}(\phi(\boldsymbol{x}_1), ..., \phi(\boldsymbol{x}_n))$ to express the optimal weight

vector as $\boldsymbol{w} = \sum_{i=1}^{n} \alpha'_i \phi(\boldsymbol{x}_i)$, the cutting-plane method tells us that we only need to consider the subspace $\mathcal{W} \subset \mathcal{F}$ in each iteration, where $m << n$ and m does not grow with n. Our core idea is to find a small set of basis vectors $\boldsymbol{b}_1, ..., \boldsymbol{b}_k$ so that

$$\mathcal{W}' = \text{span}(\phi(\boldsymbol{b}_1), ..., \phi(\boldsymbol{b}_k)) \approx \mathcal{W}, \tag{6}$$

which means that we can express the final solution from (5) as

$$\boldsymbol{w} \approx \sum_{i=1}^{k} \alpha''_i \phi(\boldsymbol{b}_i). \tag{7}$$

This enables efficient prediction using the rule $h(\boldsymbol{x}) = \text{sign}[\sum_{i=1}^{k} \alpha''_i K(\boldsymbol{b}_i, \boldsymbol{x})]$, given that k is small. Furthermore, we will elaborate in the following how projecting into the subspace $\mathcal{W}'$ allows computing $\boldsymbol{H}$ and $\langle \boldsymbol{w}, \phi(x) \rangle$ in time independent of n.

To understand the intuition behind our approach, consider the ideal case where for every $\bar{\boldsymbol{\Psi}}_i$ there exists a vector $\boldsymbol{b}_i$ in input space (not necessarily from the training set) so that $\bar{\boldsymbol{\Psi}}_i = \phi(\boldsymbol{b}_i)$ (as it does in the linear case, where $\boldsymbol{b}_i = \frac{1}{n} \sum_{j=1}^{n} (y_j - \hat{y}_j) \boldsymbol{x}_j$). Then we could replace each $\bar{\boldsymbol{\Psi}}_i$ with $\phi(\boldsymbol{b}_i)$, and it is easy to verify that the time complexity of an iteration goes down to $O(m^3 + mn)$ – almost like in the linear case. Furthermore, the resulting classifier would only have $k = m \approx 30$ "Support Vectors" – or, more generally named, "Basis Vectors" –, making it much faster than conventional SVM classifiers that often have 10000's of SVs.

Unfortunately, in most cases there will be no single pre-image $\boldsymbol{b}$ so that $\bar{\boldsymbol{\Psi}} = \phi(\boldsymbol{b})$. However, in any iteration it suffices to find a set of pre-image vectors so that $\bar{\boldsymbol{\Psi}}_1, ..., \bar{\boldsymbol{\Psi}}_m$ lie (approximately) in their span. In particular, we are looking for a set of basis vectors $\boldsymbol{B} = (\boldsymbol{b}_1, ..., \boldsymbol{b}_k)$, $\boldsymbol{b}_i \in \Re^N$, so that for every $\bar{\boldsymbol{\Psi}}_i$ in $\bar{\boldsymbol{\Psi}}$

$$\min_{\boldsymbol{\beta}} ||\bar{\boldsymbol{\Psi}}_i - \sum_{j=1}^{k} \boldsymbol{\beta}_j \phi(\boldsymbol{b}_j)||^2 \leq \delta \tag{8}$$

for some (small) $\delta \geq 0$. When computing $\boldsymbol{H}$ and $\langle \boldsymbol{w}, \phi(\boldsymbol{x}_i) \rangle$ in Algorithm 2, we then replace $\bar{\boldsymbol{\Psi}}_i$ with its projection $\hat{\boldsymbol{\Psi}}_i \equiv \sum_{j=1}^{k} \boldsymbol{\beta}_j \phi(\boldsymbol{b}_j)$. This is summarized in Algorithm 3, which we call the *Cutting-Plane Subspace Pursuit* (CPSP) algorithm. It is easy to verify that $\boldsymbol{H}$ and all $\langle \boldsymbol{w}, \phi(\boldsymbol{x}_i) \rangle$ can now be computed in time $O(m^2 k^2)$ (or $O(k^3 + m^2 k + m k^2)$) and $O(mk + kn)$, respectively.

Using $\hat{\bar{\boldsymbol{\Psi}}}$ instead of $\bar{\boldsymbol{\Psi}}$ in Algorithm 3 is straightforward. However, we still have to define how the function $\mathit{extend\_basis}(\boldsymbol{B}, \bar{\boldsymbol{\Psi}}_m)$ (Line 15) computes the set of basis vectors $\boldsymbol{B} = (\boldsymbol{b}_1, ..., \boldsymbol{b}_k)$ and how the function $\mathit{project}(\bar{\boldsymbol{\Psi}}_i, \boldsymbol{B})$ (Line 17) computes the approximate cutting-planes $\hat{\boldsymbol{\Psi}}_i$. This is addressed in the following.

4.1 Projecting Cutting-Planes $\bar{\boldsymbol{\Psi}}$ onto $\boldsymbol{B}$

For a given subspace $\text{span}(\phi(\boldsymbol{b}_1), ..., \phi(\boldsymbol{b}_k))$, the function `project(`$\bar{\boldsymbol{\Psi}}_i$`,`$\boldsymbol{B}$`)` computes the projection $\hat{\boldsymbol{\Psi}}_i$ of a cutting-plane $\bar{\boldsymbol{\Psi}}_i$ via the following least-squares

Algorithm 3 Cutting-Plane Subspace Pursuit (CPSP) Algorithm

Input: $S = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n))$, C, ϵ, k_{max}, $K(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle$
$\bar{\Delta} \leftarrow \mathbf{0}$, $\bar{\boldsymbol{\Psi}} \leftarrow \mathbf{0}$, $\hat{\boldsymbol{\Psi}} \leftarrow \mathbf{0}$, $\boldsymbol{H} \leftarrow \mathbf{0}$,$\boldsymbol{B} \leftarrow \emptyset$, $m \leftarrow 0$
repeat
  $\boldsymbol{H} \leftarrow (\boldsymbol{H}_{ij})_{1 \le i,j \le m}$, where $\boldsymbol{H}_{ij} = \left\langle \hat{\boldsymbol{\Psi}}_i, \hat{\boldsymbol{\Psi}}_j \right\rangle$
  $\boldsymbol{\alpha} \leftarrow \text{argmax}_{\boldsymbol{\alpha} \ge 0}\, \boldsymbol{\alpha}^T \bar{\Delta} - \frac{1}{2} \boldsymbol{\alpha}^T \boldsymbol{H} \boldsymbol{\alpha}$ s.t. $\boldsymbol{\alpha}^T \mathbf{1} \le C$
  $\xi \leftarrow \frac{1}{C}(\boldsymbol{\alpha}^T \bar{\Delta} - \boldsymbol{\alpha}^T \boldsymbol{H} \boldsymbol{\alpha})$
  $\boldsymbol{w} \equiv \sum_i \alpha_i \hat{\Psi}_i$
  for i=1,...,n **do**
    $\hat{y}_i \leftarrow \text{sign}(\langle \boldsymbol{w}, \phi(\boldsymbol{x}_i) \rangle - y_i)$
  end for
  $(\bar{\boldsymbol{\Psi}}, \bar{\Delta}, m) = \mathit{remove\_inactive}(\bar{\boldsymbol{\Psi}}, \bar{\Delta}, \boldsymbol{\alpha})$
  $m \leftarrow m + 1$
  $\bar{\boldsymbol{\Psi}}_m \equiv \frac{1}{2n} \sum_{i=1}^{n} (y_i - \hat{y}_i) \phi(\boldsymbol{x}_i)$
  $\bar{\Delta}_m \leftarrow \frac{1}{2n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$
  if $|\boldsymbol{B}| < k_{max}$ **then** $\boldsymbol{B} \leftarrow \mathit{extend\_basis}(\boldsymbol{B}, \bar{\boldsymbol{\Psi}}_m)$
  for i=1,...,k **do**
    $\hat{\boldsymbol{\Psi}}_i \equiv \mathit{project}(\bar{\boldsymbol{\Psi}}_i, \boldsymbol{B})$
  end for
until $\left\langle \boldsymbol{w}, \bar{\boldsymbol{\Psi}}_m \right\rangle \ge \bar{\Delta}_m - \xi - \epsilon$
return($\boldsymbol{w}$,ξ)

problem:

$$\hat{\boldsymbol{\Psi}}_i = \sum_{j=1}^{k} \boldsymbol{\beta}_j \phi(\boldsymbol{b}_j) \;\text{ where }\; \boldsymbol{\beta} = \min_{\boldsymbol{\beta}} ||\bar{\boldsymbol{\Psi}}_i - \sum_{j=1}^{k} \boldsymbol{\beta}_j \phi(\boldsymbol{b}_j)||^2 \tag{9}$$

To accomodate kernels, we maintain the $k \times k$-matrix $\boldsymbol{G}$ with $\boldsymbol{G}_{ij} = K(\boldsymbol{b}_i, \boldsymbol{b}_j)$ and the $k \times n$-matrix $\boldsymbol{K}$ with $\boldsymbol{K}_{ij} = K(\boldsymbol{b}_i, \boldsymbol{x}_j)$. The solution of the least-squares problem can then be written as $\boldsymbol{\beta} = \frac{1}{2n} \boldsymbol{G}^{-1} \boldsymbol{K} \boldsymbol{Y}_{*i}$. It is more efficient, however, to use the Cholesky decomposition $\boldsymbol{L}_G$ of $\boldsymbol{G}$ (i.e. $\boldsymbol{G} = \boldsymbol{L}_G \boldsymbol{L}_G^T$). With $\boldsymbol{L}_G$, the solution can be computed via forward and back-substitution from $\boldsymbol{L}_G \boldsymbol{\gamma} = \frac{1}{2n} \boldsymbol{K} \boldsymbol{Y}_{*i}$ and $\boldsymbol{L}_G^T \boldsymbol{\beta} = \boldsymbol{\gamma}$ in time $O(k^2 + kn)$. This excludes the time for computing $\boldsymbol{K}$, $\boldsymbol{G}$, and its Cholesky decomposition $\boldsymbol{L}_G$, since these need to be computed only once and can then be used until $\boldsymbol{B}$ changes. This is further discussed in the next section.

4.2 Constructing the Set of Basis Vectors $\boldsymbol{B}$

The method for constructing the set of basis vectors $\boldsymbol{B}$ is the final part of Algorithm 3 that still needs to be specified. The goal is to find a set of basis vectors $\boldsymbol{B} = (\boldsymbol{b}_1, ..., \boldsymbol{b}_k)$ such that for some small $\delta \ge 0$, all $\bar{\boldsymbol{\Psi}}_i$ that are active in the current iteration fulfill (8). Recomputing $\boldsymbol{B}$ in each iteration would be costly,

but fortunately it is unnecessary. Only $\bar{\boldsymbol{\Psi}}_m$ is new and all other $\bar{\boldsymbol{\Psi}}_i$ are already well approximated by the set of basis vectors from the previous iteration. The function `extend_basis(`$\boldsymbol{B}$`,`$\bar{\boldsymbol{\Psi}}_m$`)` therefore only adds some new basis vectors to $\boldsymbol{B}$ that are required to fit $\bar{\boldsymbol{\Psi}}_m$. Note that this can only improve the fit for the other $\bar{\boldsymbol{\Psi}}_i$.

To decide which basis vectors to add, we follow [5] and take a greedy approach. We search for the basis vector $\boldsymbol{b}_{k+1}$ that minimizes the residual error for $\bar{\boldsymbol{\Psi}}_m$, where $\hat{\boldsymbol{\Psi}}_m$ is the projection for the current $\boldsymbol{B}$.

$$(\beta', \boldsymbol{b}') = \operatorname*{argmin}_{\beta_{k+1}, \boldsymbol{b}_{k+1}} ||\bar{\boldsymbol{\Psi}}_m - \hat{\boldsymbol{\Psi}}_m - \beta_{k+1}\phi(\boldsymbol{b}_{k+1})||^2 \tag{10}$$

This optimization problem is commonly referred to as the "preimage" problem. While exact solutions are difficult to obtain, approximate solutions can be found with gradient-based methods [19, 20] or randomized search. In this paper, we use the fix-point iteration approach described in [20, Sec. 18.2.2] for the RBF kernel to solve (10) to a local optimum. In this way we can efficiently produce arbitrary vectors as basis vectors to add to $\boldsymbol{B}$. We refer to the Cutting-Plane Subspace Pursuit algorithm with this preimage method as "CPSP" in the following.

To evaluate in how far general basis vectors improve sparsity, we also explore a second preimage method that is restricted to using basis vectors from the training set. We refer to this method as "CPSP(tr)". As proposed by Smola and Schölkopf [7] (and used by most of the methods we compare against), we randomly sample 59 feature vectors from the training set and pick the one with maximum objective value in (10). Note that this alternative strategy is introduced only to evaluate the benefit of selecting "support vectors" outside the training set.

The number of new basis vectors to add for each $\bar{\boldsymbol{\Psi}}_m$ is a design choice. One could either use a fixed number, or keep adding until a certain δ is achieved. In the following experiments, we use the simplest choice and add exactly one basis vector for each $\bar{\boldsymbol{\Psi}}_m$ until the maximum size k_{max} specified by the user has been reached. At that point, no further vectors are added and `extend_basis(`$\boldsymbol{B}$`,`$\bar{\boldsymbol{\Psi}}_m$`)` returns $\boldsymbol{B}$ unchanged.

After a new $\boldsymbol{b}_{k+1}$ is added to $\boldsymbol{B}$, a column/row needs to be added to the kernel matrices $\boldsymbol{G}$ and $\boldsymbol{K}$. This takes $O(n + k)$ kernel evaluations, and the Cholesky factorization of $\boldsymbol{G}$ can be updated in time $O(k^2)$.

5 Theoretical Analysis

Before evaluating the CPSP algorithm empirically, we first give a theoretical characterization of the quality of its solutions and the number of iterations it takes until convergence.

The following theorem gives an upper bound on the number of iterations of Algorithm 3. It extends the general results [15, 18, 16] for cutting-plane training of SVMs to the CPSP algorithm.

Theorem 1. *For parameter C, precision ϵ, training-set size n, and basis-set size k_{max}, Algorithm 3 terminates after at most $O(k_{max} + \frac{C}{\epsilon})$ iterations.*

Proof. After the first k_{max} iterations, the basis $\boldsymbol{B}$ becomes fixed, and from then on we are essentially solving the optimization problem:

$$\min_{\boldsymbol{w},\xi} \ \frac{1}{2}\|\boldsymbol{w}\|^2 + C\xi \tag{11}$$
$$s.t. \ \forall \hat{y} \in \{-1,1\}^n : \left\langle \boldsymbol{w}, \sum_{i=1}^{n} (y_i - \hat{y}_i)\phi(\boldsymbol{x}_i) \right\rangle \geq \sum_{i=1}^{n} \Delta(y_i, \hat{y}_i) - \xi \ \text{ and } \ \boldsymbol{w} \in \sum_{\boldsymbol{b}_i \in \boldsymbol{B}} \boldsymbol{\beta}_i \phi(\boldsymbol{b}_i)$$

Let $P_{\boldsymbol{B}}$ be the orthogonal projection operator onto the subspace spanned by $\boldsymbol{B}$. Such an orthogonal projection operator always exists in a Hilbert Space. After folding the subspace constraint into the objective by replacing $\boldsymbol{w}$ with $P_{\boldsymbol{B}}\boldsymbol{w}$, the above optimization problem can be re-written as (using the self-adjointness and linearity of $P_{\boldsymbol{B}}$):

$$\min_{\boldsymbol{w},\xi} \frac{1}{2}\|P_{\boldsymbol{B}}\boldsymbol{w}\|^2 + C\xi$$
$$s.t. \ \forall \hat{y} \in \{-1,1\}^n : \left\langle \boldsymbol{w}, \sum_{i=1}^{n} (y_i - \hat{y}_i) P_{\boldsymbol{B}}\phi(\boldsymbol{x}_i) \right\rangle \geq \sum_{i=1}^{n} \Delta(y_i, \hat{y}_i) - \xi$$

Finally the operator $P_{\boldsymbol{B}}$ in the objective can be dropped since if $\boldsymbol{w}$ contains any component in $\boldsymbol{B}^{\perp}$, it will only increase the objective without changing value of the LHS of the constraints. This is in the form of the general Structural SVM optimization problem solved by Algorithm 1, with the feature space changed from being spanned by $\phi(\boldsymbol{x}_i)$ to being spanned by $P_B\phi(\boldsymbol{x}_i)$. The $O(\frac{C}{\epsilon})$ iteration bound from [15, 18, 16] therefore applies. □

The time complexity of each iteration was already discussed in Section 4, but can be summarized as follows. In iterations where no new basis vector is added to $\boldsymbol{B}$, the time complexity is $O(m^3 + mk^2 + kn)$, since only the new $\bar{\boldsymbol{\Psi}}_m$ needs to be projected and the respective column be added to $\boldsymbol{H}$. In iterations where $\boldsymbol{B}$ is extended, the time complexity is $O(m^3 + k^2m + km^2 + kmn)$ plus the time it takes to solve the preimage problem (10). Note that typical values are $m \approx 30$, $k \in [10..1000]$, and $n > 10000$.

The following theorem describes the quality of the solution at termination, accounting for the error incurred by projecting on an imperfect $\boldsymbol{B}$. Most importantly, the theorem justifies our use of (10) for deciding which basis vectors to add.

Theorem 2. *When Algorithm 3 terminates with $||\bar{\boldsymbol{\Psi}}_i - \hat{\boldsymbol{\Psi}}_i|| \leq \delta$ for all $\bar{\boldsymbol{\Psi}}_i$ and $\hat{\boldsymbol{\Psi}}_i$, then the primal objective value o of the solution found does not exceed the exact solution o^* by more than $o - o^* \leq C(\delta\sqrt{2C} + \epsilon)$.*

Proof. Let $\boldsymbol{w}^*$ be the optimal solution with value o^*. We know that the optimal $\boldsymbol{w}^*$ satisfies $\|\boldsymbol{w}^*\| \leq \sqrt{2C}$. Hence for all i,

$$|\left\langle \boldsymbol{w}, \bar{\boldsymbol{\Psi}}_i \right\rangle - \left\langle \boldsymbol{w}, \hat{\Psi}_i \right\rangle| \leq \ \|\boldsymbol{w}\|\|\bar{\boldsymbol{\Psi}}_i - \hat{\boldsymbol{\Psi}}_i\| \leq \ \delta\sqrt{2C}$$

Let $P_{\boldsymbol{B}}$ be the orthogonal projection on the subspace spanned by $\phi(\boldsymbol{b}_i)$ in the final basis $\boldsymbol{B}$. Let $\boldsymbol{v}^*$ be the optimal solution to the optimization problem (12) restricted to the subspace $\boldsymbol{B}$, we have:

$$
\begin{aligned}
o &\leq \frac{1}{2}\|\boldsymbol{v}^*\|^2 + C(\xi+\epsilon) \\
&= \frac{1}{2}\|\boldsymbol{v}^*\|^2 + C \max_{1\leq i\leq m}[\bar{\Delta}_i - \left\langle \boldsymbol{v}^*, \hat{\boldsymbol{\Psi}}_i \right\rangle] + C\epsilon \\
&\leq \frac{1}{2}\|P_{\boldsymbol{B}}\boldsymbol{w}^*\|^2 + C \max_{1\leq i\leq m}[\bar{\Delta}_i - \left\langle P_{\boldsymbol{B}}\boldsymbol{w}^*, \hat{\boldsymbol{\Psi}}_i \right\rangle] + C\epsilon \\
&\quad [\text{since } \boldsymbol{v}^* \text{ is the optimal solution wrt the basis } \boldsymbol{B}] \\
&= \frac{1}{2}\|P_{\boldsymbol{B}}\boldsymbol{w}^*\|^2 + C \max_{1\leq i\leq m}[\bar{\Delta}_i - \left\langle \boldsymbol{w}^*, P_{\boldsymbol{B}}\hat{\boldsymbol{\Psi}}_i \right\rangle] + C\epsilon \\
&= \frac{1}{2}\|P_{\boldsymbol{B}}\boldsymbol{w}^*\|^2 + C \max_{1\leq i\leq m}[\bar{\Delta}_i - \left\langle \boldsymbol{w}^*, \hat{\boldsymbol{\Psi}}_i \right\rangle] + C\epsilon \\
&\leq \frac{1}{2}\|\boldsymbol{w}^*\|^2 + C \max_{1\leq i\leq m}[\bar{\Delta}_i - \left\langle \boldsymbol{w}^*, \hat{\boldsymbol{\Psi}}_i \right\rangle] + C\epsilon \\
&\leq \frac{1}{2}\|\boldsymbol{w}^*\|^2 + C \max_{1\leq i\leq m}[\bar{\Delta}_i - \langle \boldsymbol{w}^*, \bar{\boldsymbol{\Psi}}_i \rangle + \delta\sqrt{2C}] + C\epsilon \\
&\leq o^* + C(\delta\sqrt{2C} + \epsilon) \qquad \square
\end{aligned}
$$

6 Experimental Analysis

The following experiments are designed to evaluate how the CPSP method compares to conventional training methods in terms of sparsity (i.e. prediction efficiency) and training efficiency. In particular, they explore whether the use of general basis vectors outside the training set improves prediction accuracy and training efficiency, and how both quantities scale with basis set size k_{max}.

Our implementation of the CPSP algorithm is available for download at `http://svmlight.joachims.org/svm_perf.html`.

We compare the CPSP algorithm with the exact solution computed by SVM^{light}, as well as approximate solutions of the Nystrom method (Nystrom) [6], the Incomplete Cholesky Factorization (IncChol) [8], the Core Vector Machine (CVM) [10], the Ball Vector Machine (BVM) [11], and LASVM with margin-based active selection and finishing [12]. Both the Nystrom method and the Incomplete Cholesky Factorization are implemented in SVM^{perf} as described in [16]. We use the RBF-Kernel $K(\boldsymbol{x}, \boldsymbol{x}') = exp(-\gamma||\boldsymbol{x}-\boldsymbol{x}'||^2)$ in all experiments. The cache sizes of SVM^{light}, CVM, BVM, and LASVM were set to 1GB.

We compare on the following five binary classification tasks, each split into training/validation/test set. If not mentioned otherwise, parameters (i.e. C and γ) are selected to maximize performance on the validation set for each method and k_{max} individually. Both C and γ are explored on a log-scale. The first dataset is Adult as compiled by John Platt with 123 features and using a train/validation/test split of 20000/6281/6280. Second is the Reuters RCV1

Table 1. Prediction accuracy with $k_{max} = 1000$ basis vectors (except SVM^{light}, where the number of SVs is shown in the third line) using the RBF kernel (except linear).

	ADULT	CCAT	OCR0	OCR*	IJCNN
SVM-light (linear)	84.4	94.2	99.4	87.6	92.2
SVM-light (RBF)	84.4	95.1	99.8	98.6	99.4
#SV	7125	28748	2786	19309	9243
CPSP	**84.5**	**95.0**	**99.8**	**98.5**	**99.3**
CPSP(tr)	84.1	93.5	**99.8**	97.9	99.2
NYSTROM	84.3	92.5	99.7	97.0	99.1
INCCHOL	84.0	92.1	99.7	97.0	98.9
CVM	78.4	88.1	**99.8**	96.9	98.2
BVM	77.1	56.1	**99.8**	89.1	97.7
LASVM	83.8	91.7	**99.8**	97.2	97.5

CCAT text-classification dataset with 47236 features. We use 78127 examples from the original test set for training and split the original training set into validation and test sets of sizes 11575 and 11574 respectively. Third and fourth, we classify the digit "0" against the rest (OCR0), as well as classify the digits "01234" against the digits "56789" (OCR*) on the MNIST dataset. The MNIST datasets have 780 features and we use a training/validation/test split of 50000/5000/5000. Finally, we use the IJCNN (task 1) dataset as pre-processed by Chih-Jen Lin. It has 22 features and we use a training/validation/test split of 113533/14169/14169.

How Accurate are the Solutions for a given Sparsity Budget? We first explore a scenario where the application demands an upper bound on the number of support vectors to achieve a desired computational efficiency at prediction time. Table 1 summarizes the results. The first two lines show the performance of SVM^{light} for the linear kernel and SVM^{light} for the RBF kernel as baselines to compare against. All but the Adult dataset show substantial non-linear structure, and the RBF kernel outperforms a linear SVM. The number of SVs when using the RBF kernel is given in the third line. The remaining lines in Table 1 are for the "sparse" methods, all of which use $k_{max} = 1000$ basis vectors. Note that this is well below the 2786 to 28748 support vectors required by the exact SVM.

The CPSP algorithm with the general preimage method matches the accuracy of SVM^{light} up to ± 0.1. This means that the prediction accuracy is roughly the same as for the exact method, while speeding up prediction by a factor between 2.7 to 28. We will see in Section 6 that far fewer than $k_{max} = 1000$ basis vectors would have sufficed on some of the tasks for the CPSP algorithm, leading to an even larger speedup.

Random sampling of the basis vectors in the NYSTROM method and the Incomplete Cholesky factorization (INCCHOL) perform consistently worse than the CPSP method, except on the OCR0 dataset where all methods do well with $k_{max} = 1000$ basis vectors. The Core Vector Machine (CVM), the Ball Vector Machine (BVM), and the LASVM algorithm with active selection are not competitive on most datasets.

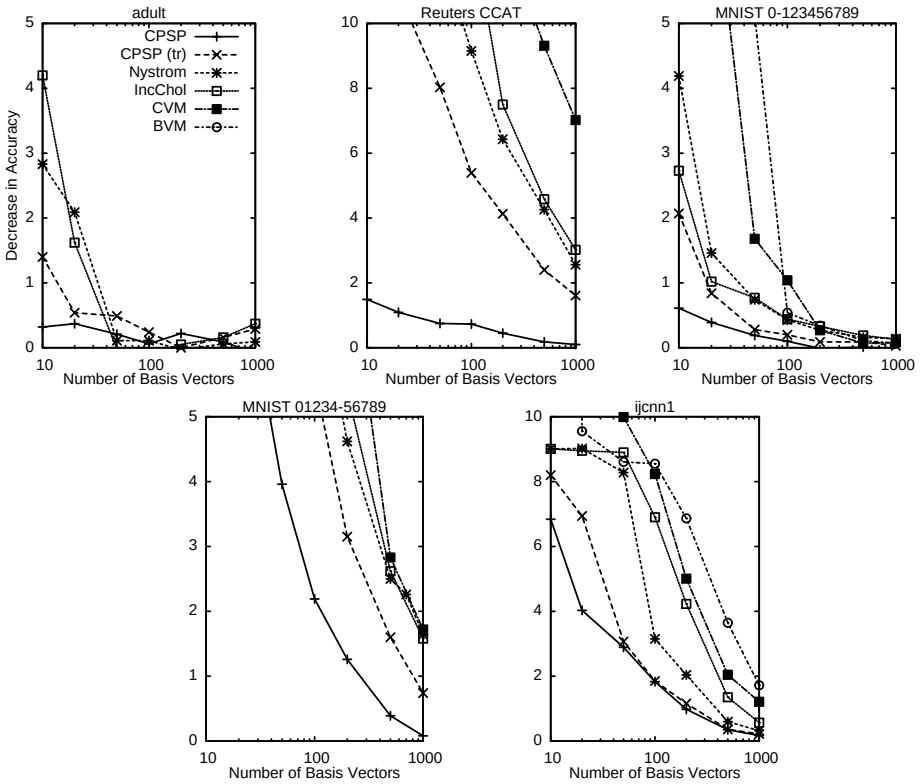

Fig. 1. Decrease in accuracy w.r.t. exact SVM for different basis-set sizes k_{max}.

How does Accuracy Scale with Basis-Set Size? As mentioned above, a lower number of basis vectors $k_{max} << 1000$ could have sufficed to get reasonable accuracy on some datasets. The plots in Figure 1 investigate this question and show by how much the test accuracies for a given k_{max} are lower than the accuracy of the exact SVM solution. In each plot, 0 corresponds to the accuracy of the exact SVM solution.

Figure 1 shows that CPSP dominates all other methods not only for $k_{max} = 1000$, but over the whole range. For all datasets, the CPSP method using general preimages outperforms the other methods especially for small numbers of basis vectors. In particular, on three of the five datasets, CPSP already performs within 1% of the exact solution with only 50 basis vectors. Similarly, on all five datasets does CPSP perform equivalent or better than the linear SVM when using 50 basis vectors or more. Especially on ADULT, CCAT, and OCR0 far fewer than $k_{max} = 1000$ basis vectors would have sufficed to reach an acceptable level of performance.

What is the Benefit of using General Basis Vectors? A key premise of the paper is that using basis vectors outside the training set is beneficial. To test its validity, Figure 1 and Table 1 include the performance of the CPSP(tr) algorithm, which is identical to CPSP except for selecting basis vectors only

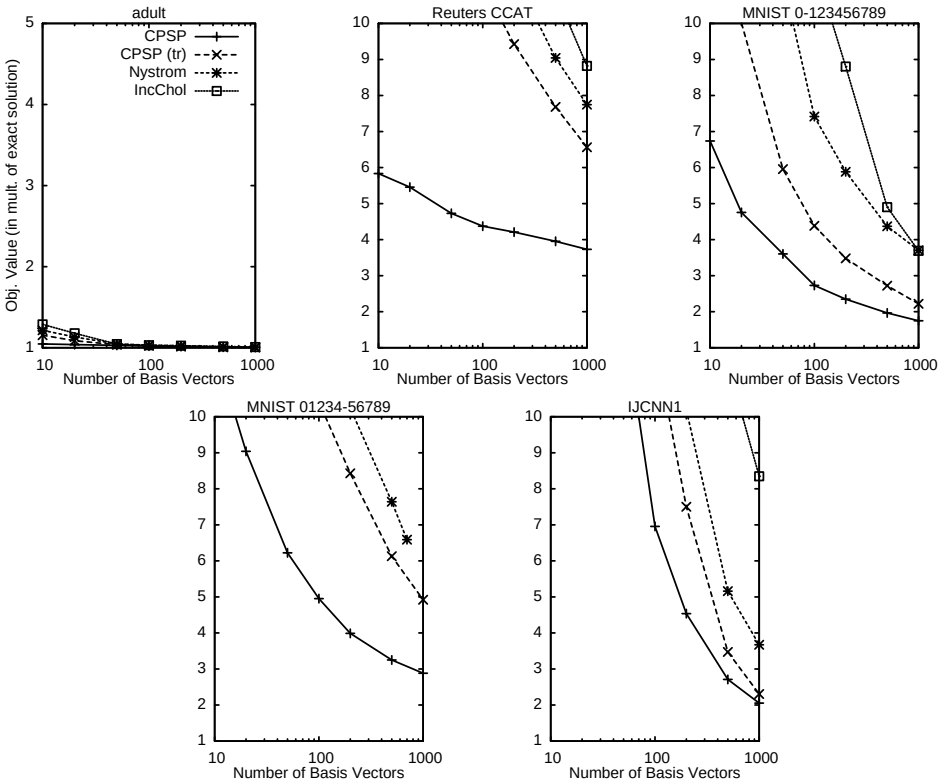

Fig. 2. Primal objective value of the approximate solutions expressed as multiples of the exact SVM solution.

from the training set. Consistently over all dataset, Figure 1 shows that the general CPSP algorithm provides improved prediction accuracy over CPSP(tr) especially for small numbers of basis vectors. The difference is largest on the CCAT dataset, where the general CPSP algorithm with 10 basis vectors already performs at an accuracy for which CPSP(tr) requires about 1000 basis vectors. This confirms our hypothesis that basis vectors outside the training set can lead to more accurate solutions at a given level of sparsity.

How Accurate is the Objective Value? The four methods CPSP, CPSP(tr), NYSTROM, and INCCHOL all optimize the same objective function as a regular SVM. How well do they manage to minimize this objective? The plots in Figure 6 show by what factor their primal objective value is higher than the exact SVM solution. All methods use the same parameters (i.e. C and γ), which are picked to optimize validation set accuracy of the exact SVM.

Again, CPSP dominates the other methods, and the curves in Figure 6 very much resemble the curves in Figure 1. This verifies that finding a subspace that contains a solution of low objective value is indeed crucial for good prediction accuracy, and that the subspaces found by CPSP are of superior fidelity (also compared to CPSP(tr)).

Table 2. Number of SV (left) and training time (right) to reach an accuracy that is not more than 0.5% below the accuracy of the exact solution of SVM-light (see Table 1). The RBF kernel is used for all methods. '>' indicates that the largest tractable solution did not achieve the target accuracy.

	Number of SV					Training Time (CPU-Seconds)				
	ADULT	CCAT	OCR0	OCR*	IJCNN	ADULT	CCAT	OCR0	OCR*	IJCNN
SVM-light	7125	28748	2786	19309	9243	56	9272	400	4629	1175
CPSP	**10**	**200**	**20**	**500**	**500**	**6**	**225**	11	465	2728
CPSP(tr)	50	5000	50	2000	**500**	30	88873	57	8967	2178
NYSTROM	50	>5000	100	5000	1000	10	>2281	37	2270	1572
INCCHOL	50	>2000	100	>2000	2000	14	>21673	66	>12330	59454
CVM	5000	20000	200	5000	2000	43	23730	**2**	497	**29**
BVM	5000	20000	200	5000	5000	67	11004	**2**	538	229
LASVM	2000	10000	100	2000	5000	51	3433	5	**295**	705

What is the Training and Test Efficiency? While efficiency at test time may be the dominant criterion for many applications, training has to be tractable as well. Since CPSP does more work in each iteration (e.g. solve a pre-image problem), one superficial concern might be that the training process is slow. However, the following shows that the increased sparsity observed in Figure 1 not only improves prediction efficiency, but also speeds up training. This is a key difference to the Reduced Set method [5]. The Reduced Set method requires solving an exact SVM, making it intractable for large training sets.

Table 2 compares the training time and number of basis vectors that each method needs to reach a certain prediction accuracy. The experiment simulates how a user may chose to trade prediction accuracy for improved training and test efficiency. In particular, Table 2 shows the number of basis vectors (left) and the training time (right) to reach a test accuracy that is not more than 0.5% below the test accuracy of the exact SVM. Basis set sizes $k_{max} \in \{10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000, 20000\}$ were tried for each method and the results for the smallest k_{max} that achieved the target accuracy are shown.

Table 2 shows that the solutions found by CPSP are typically substantially more sparse than those of the other methods. Compared to the exact solution, they lead to an 18 to 712 fold speed-up at prediction time. Compared to the other approximate methods, the speed-up is still typically between 5 and 10.

The increased sparsity also leads to very efficient training times for CPSP. While it is difficult to rank methods by aggregated training times, CPSP is clearly among the fastest methods in the comparison, especially on those tasks where general basis vectors provide a substantial gain in sparsity. On the CCAT text-classification dataset, it is orders of magnitude faster than any of the other methods (and CPSP(tr)). For such large and sparse data, there simply does not appear to be a small subset of training vectors that can represent an accurate

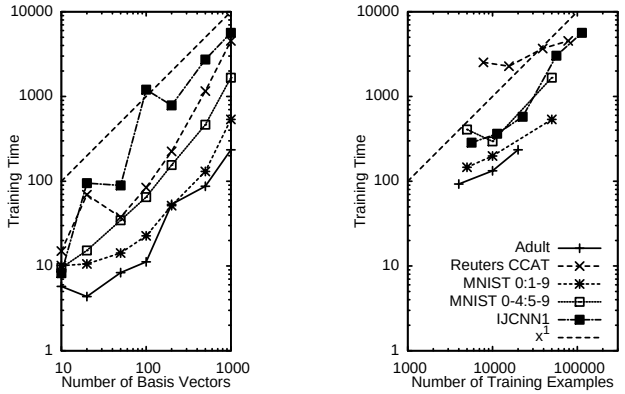

Fig. 3. Training times of CPSP for varying basis-set sizes (left) and training-set sizes with $k_{max} = 1000$ (right).

classifier, and the increased sparsity from allowing general basis vectors greatly improves training efficiency. More generally, all training methods for SVMs scale super-linearly with the number of SVs, so that improving sparsity is the key to making large-scale training tractable. The scaling properties of CPSP are explored in more detail in the following section.

Comparing to the results published in [2], our method is substantially faster. They focus mostly on small training sets with less than 1000 examples. The USPS OCR dataset with 7291 and 256 features is their largest dataset, and they report and average training time of ∼2400 seconds. This dataset roughly compares to our OCR0 task. However, the OCR0 dataset is an order of magnitude larger.

How does Training Time Scale with Basis-Set Size? Finally, let us investigate the efficiency of CPSP in more detail. The left-hand plots in Figure 3 show training time for different values of k_{max}. Parameters (regularization C, RBF γ) are individually picked via CV for each method and k_{max}. While the theoretical time complexity is $O({k_{max}}^3)$, the actual scaling shown in Figure 3 (left) is much more benign. For $k_{max} < 1000$, the time contribution of the cubic parts of the algorithm (e.g. repeatedly updating the Cholesky factorization $\boldsymbol{L}_G$) is still rather small, and the scaling behavior is only modestly super-linear.

How does Training Time Scale with Training-Sample Size? Finally, the right-hand plot in Figure 3 shows training time of CPSP for different training set sizes. Parameters (regularization C, RBF γ) are individually picked via CV for each method and training set size. As expected from the theoretical analysis, the scaling behavior is roughly linear, making CPSP particularly attractive for large datasets.

7 Conclusions

We presented a training algorithm for kernel SVMs that constructs a sparse set of basis vectors as part of the cutting-plane optimization process. The algorithm's

efficiency and effectiveness is characterized theoretically, and an experimental comparison shows that is produces solutions of a sparsity that is superior to Nystrom, IncChol, CVM, BVM, and LASVM. We find that the ability to use basis vectors outside the training set substantially contributes to this gain in sparsity and efficiency, especially on large datasets with sparse feature vectors.

Acknowledgments This work was funded in part under NSF awards IIS-0713483.

References

1. Steinwart, I.: Sparseness of support vector machines. JMLR **4** (2003) 1071–1105
2. Wu, M., Schölkopf, B., Bakir, G.H.: A direct method for building sparse kernel learning algorithms. JMLR **7** (2006) 603–624
3. Platt, J.: Using analytic QP and sparseness to speed training of support vector machines. In: NIPS. (1999) 557–563
4. Joachims, T.: Making large-scale SVM learning practical. In Schölkopf, B., Burges, C., Smola, A., eds.: Advances in Kernel Methods - Support Vector Learning. MIT Press (1999) 169–184
5. Burges, C., Schölkopf, B.: Improving the accuracy and speed of support vector learning machines. In: NIPS. Volume 9. (1997) 375–381
6. Williams, C., Seeger, M.: Using the Nystrom method to speed up kernel machines. In: NIPS. (2001)
7. Smola, A., Schölkopf, B.: Sparse greedy matrix approximation for machine learning. In: ICML. (2000) 911–918
8. Fine, S., Scheinberg, K.: Efficient SVM training using low-rank kernel representations. JMLR **2** (2001) 243–264
9. Bach, F., Jordan, M.: Predictive low-rank decomposition for kernel methods. In: ICML. (2005) 33–40
10. Tsang, I., Kwok, J., Cheung, P.M.: Core vector machines: Fast svm training on very large data sets. JMLR **6** (2005) 363–392
11. Tsang, I.W., Kocsor, A., Kwok, J.T.: Simpler core vector machines with enclosing balls. In: ICML. (2007) 911–918
12. Bordes, A., Ertekin, S., Weston, J., Bottou, L.: Fast kernel classifiers with online and active learning. JMLR **6** (2005) 1579–1619
13. Keerthi, S., Chapelle, O., DeCoste, D.: Building support vector machines with reduced classifier complexity. JMLR **7** (2006) 1493–1515
14. Vincent, P., Bengio, Y.: Kernel matching pursuit. Machine Learning **48**(1-3) (2002) 165–187
15. Joachims, T.: Training linear SVMs in linear time. In: SIGKDD. (2006) 217–226
16. Joachims, T., Finley, T., Yu, C.N.: Cutting-plane training of structural SVMs. Machine Learning **76**(1) (2009)
17. Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y.: Large margin methods for structured and interdependent output variables. JMLR **6** (September 2005) 1453–1484
18. Teo, C.H., Smola, A., Vishwanathan, S.V., Le, Q.V.: A scalable modular convex solver for regularized risk minimization. In: SIGKDD. (2007) 727–736
19. Burges, C.: Simplified support vector decision rules. In: ICML. (1996) 71–77
20. Schölkopf, B., Smola, A.J.: Learning with Kernels. MIT Press (2002)