Learning Optimally Sparse Support Vector Machines

Andrew Cotter COTTER@TTIC.EDU
Toyota Technological Institute at Chicago, 6045 S. Kenwood Ave., Chicago, IL 60637 USA

Shai Shalev-Shwartz SHAIS@CS.HUJI.AC.IL
John S. Cohen SL in CS, The Hebrew University of Jerusalem, Israel

Nathan Srebro NATI@TTIC.EDU
Toyota Technological Institute at Chicago, 6045 S. Kenwood Ave., Chicago, IL 60637 USA

Abstract

We show how to train SVMs with an optimal guarantee on the number of support vectors (up to constants), and with sample complexity and training runtime bounds matching the best known for kernel SVM optimization (i.e. without any additional asymptotic cost beyond standard SVM training). Our method is simple to implement and works well in practice.

1. Introduction

An important aspect of kernel SVMs is that, despite being non-parametric, the learned predictor can be expressed in terms of only a subset of the training points, known as "support vectors". The number of support vectors determines the memory footprint of the learned predictor, as well as the computational cost of using it. In order for SVMs to be practical in large scale applications, it is therefore important to have only a small number of support vectors. This is particularly true when, as is the case in many applications, the training is done only once, on a powerful compute cluster that can handle large data, but the predictor then needs to be used many times, possibly in real time, perhaps on a small low-powered device.

However, when training a SVM in the non-separable setting, all incorrectly classified points will be support vectors—e.g. with 10% error, the solution of the SVM empirical optimization problem will necessarily have at least 10% of the training points as support vectors. For data sets with many thousands or even millions of points, this results in very large predictors that are expensive to store and use. Even for some separable problems, the number of support vectors in the SVM solution (the margin-maximizing classifier) may increase linearly with the training set size (e.g. when all the points are on the margin). And so, even though minimizing the empirical SVM objective might be good in terms of generalization ability (and this is very well studied), as we argue here, it might be bad in terms of obtaining sparse solutions.

In this paper, we ask how sparse a predictor we can expect, and show how to learn a SVM predictor with an optimal guarantee on its sparsity, without increasing the required sample complexity nor (asymptotically) the training runtime, and for which the worst-case generalization error has the same bound as the best known for (non-sparse) kernel SVM optimization.

2. Background: SVM Learning

SVM predictors take the form $x \mapsto \mathrm{sign}(g_w(x))$, where $g_w(x) = \langle w, \Phi(x) \rangle$ is the (real-valued) prediction on example x, $\Phi(x)$ is a (possibly implicit) mapping into some Hilbert space $\mathcal{H}$, and the predictor is specified by the vector $w \in \mathcal{H}$. SVM training amounts to finding an "empirically optimal" predictor that minimizes a balance between its norm $\|w\|$ and the training error, measured through the average hinge loss on the training examples $\hat{\mathcal{L}}_{hinge}(g_w) = \frac{1}{n}\sum_{i=1}^{n} \ell_{hinge}(y_i g_w(x_i))$, where $\ell_{hinge}(z) = \max\{0, 1-z\} = [1-z]_+$ and $(x_1, y_1), \ldots, (x_n, y_n)$ is the training set. The Representer Theorem ensures that this predictor can be expressed as $w = \sum_{i=1}^{n} \alpha_i \Phi(x_i)$. The training vectors x_i corresponding to non-zero α_i are the *support vectors*. Here, we seek predictors w with "sparse" representations, i.e. with a small number of support vectors.

We are mostly interested in Kernel SVMs, for which Φ

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

is specified implicitly via a kernel function $K(x,x') = \langle \Phi(x), \Phi(x') \rangle$. It will be convenient for us to derive and analyze our method in terms of $\Phi(\cdot)$, and then observe that it can be "kernelized", i.e. implemented in terms of $K(\cdot,\cdot)$. To simplify the presentation, we assume without loss of generality that $K(x,x) \leq 1$.

Using SVMs is theoretically justified as follows: suppose that there exists some (unknown) predictor u with low norm and low expected hinge loss $\mathcal{L}_{hinge}(g_u) = \mathbb{E}_{x,y}[\ell_{hinge}(y g_u(x))]$, where the expectation is with respect to some unknown distribution $(x,y) \sim \mathcal{D}$. Then the SVM predictor w, minimizing the appropriate balance between norm and training error, and based on enough training examples drawn i.i.d. from $\mathcal{D}$, will have (with high probability) a small expected misclassification error rate $\mathcal{L}_{0/1}(g_w) = \mathbb{E}\left[\ell_{0/1}(y g_w(x))\right]$, where $\ell_{0/1}(z) = \mathbf{1}(z \leq 0)$. More specifically, the following number of examples:

$$n \geq \tilde{O}\left(\left(\frac{\mathcal{L}_{hinge}(g_u) + \epsilon}{\epsilon}\right)\frac{\|u\|^2}{\epsilon}\log\frac{1}{\delta}\right)$$

are enough to ensure that, with probability of at least $1-\delta$, the SVM predictor will satisfy $\mathcal{L}_{0/1}(g_w) \leq \mathcal{L}_{hinge}(g_u) + \epsilon$ (this follows from Srebro et al. (2010)—see Appendix E for details[1]). We will show that one can learn a *sparse* predictor with essentially the same sample complexity.

3. Sparse SVMs

The question we consider here is whether, given that there exists a reference classifier u as in Section 2, it is possible to efficiently find a w based on a training sample which not only generalizes well, but *also* has a small support set.

If the reference classifier u separates the data with a margin, namely $\mathcal{L}_{hinge}(g_u) = 0$, then one can run the kernelized Perceptron algorithm (see for example Freund & Schapire (1999)). The Perceptron processes the training examples one by one and adds a support vector only when it makes a prediction mistake. Therefore, a bound on the number of prediction mistakes (i.e. a mistake bound) translates to a bound on the sparsity of the learned predictor. A classic result shows that if the data is separable with a margin 1 by some vector u, then the Perceptron will make at most $\|u\|^2$ prediction mistakes. Combining this with a generalization bound based on compression of representation (or with an online-to-batch conversion) we can conclude that with $n \geq \tilde{O}(\|u\|^2/\epsilon)$, the generalization error of w will be at most ϵ.

[1] Appendices are in the supplementary material

The non-separable case is more tricky. If we somehow obtained a vector v which makes a small number of *margin* violations on the training set, i.e. $\epsilon_v = \frac{1}{n}\sum_{i=1}^{n}\mathbf{1}(y_i \langle v, \Phi(x_i)\rangle < 1)$ is small, then we can find a w with $\|v\|^2$ support vectors which satisfies $\hat{\mathcal{L}}_{0/1}(g_w) \leq \epsilon_v$ by simply ignoring the examples on which $y_i \langle v, \Phi(x_i)\rangle < 1$ and running the Perceptron on the remainder. Again using a compression bound, we can show that $\mathcal{L}_{0/1}(g_w)$ is little larger than $\hat{\mathcal{L}}_{0/1}(g_w)$.

However, we cannot, in general, efficiently find a predictor v with a low margin error rate, even if we know that such a predictor exists. Instead, in learning SVMs, we minimize the empirical *hinge* loss. It is not clear how to relate the margin error rate ϵ_v to the hinge loss of u. One option is to note that $\mathbf{1}(z < 1) \leq 2[1 - z/2]_+$, hence $\epsilon_v \leq 2\hat{\mathcal{L}}_{hinge}(g_{v/2})$. Since $2\hat{\mathcal{L}}_{hinge}(g_{v/2})$ is a convex function of v, we can minimize it as a convex surrogate to ϵ_v. Unfortunately, this approach would lead to a dependence on the quantity $2\mathcal{L}_{hinge}(g_{u/2})$, which might be significantly larger than $\mathcal{L}_{hinge}(g_u)$. This is the main issue we address in Section 4, in which we show how to construct an efficient sparsification procedure which depends on $\mathcal{L}_{hinge}(g_u)$ and has the same error guarantees and sample complexity as the vanilla SVM predictor.

Before moving on, let us ask whether we could hope for sparsity *less* then $\Theta(\|u\|^2)$. As the following Lemma establishes, we cannot:

Lemma 3.1. *Let $R, \mathcal{L}^*, \epsilon \geq 0$ be given, with $\mathcal{L}^* + \epsilon \leq 1/4$ and with R^2 being an integer. There exists a data distribution $\mathcal{D}$ and a reference vector u such that $\|u\| = R$, $\mathcal{L}_{hinge}(g_u) = \mathcal{L}^*$, and any w which satisfies:*

$$\mathcal{L}_{0/1}(g_w) \leq \mathcal{L}^* + \epsilon$$

must necessarily be supported on at least $R^2/2$ vectors.[2]

Proof. The idea of the proof is to construct $\mathcal{D}$ so that the distribution over the instances is uniform over some orthonormal basis of $\mathbb{R}^{R^2}$. Then, any w which is supported on less than $R^2/2$ vectors must have $g_w(x) = 0$ with probability of at least $1/2$. For full details see Appendix A. □

4. Learning Sparse SVMs

In the previous section we showed that having a good low-norm predictor u often implies there exists also a good sparse predictor, but the existence proof required

[2] This also holds for randomized classification rules that predict 1 with probability $\psi(g_u(x))$ for some $\psi : \mathbb{R} \to [0,1]$

low *margin error*. We will now consider the problem of efficiently finding a good sparse predictor, given the existence of low-norm reference predictor u which suffers low *hinge loss* on a finite training sample.

Our basic approach will be broadly similar to that of Section 3, but instead of relying on an unknown u, we will start by using any SVM optimization approach to learn a (possibly dense) w. We will then learn a sparse classifier $\tilde{w}$ which mimics w.

In Section 3 we removed margin violations and dealt with an essentially separable problem. But when one considers not margin error, but hinge loss, the difference between "correct" and "wrong" is more nuanced, and we must take into account the numeric value of the loss:

1. If $y\langle w, \Phi(x)\rangle \leq 0$ (i.e. w is wrong), then we can ignore the example, as in Section 3.
2. If $0 < y\langle w, \Phi(x)\rangle < 1$ (i.e. w is correct, but there is a margin violation), then we allow $\tilde{w}$ to make a margin violation at most $1/2$ worse then the margin violation made by w, i.e. $y\langle \tilde{w}, \Phi(x)\rangle \geq y\langle w, \Phi(x)\rangle - 1/2$.
3. If $y\langle w, \Phi(x)\rangle \geq 1$ (i.e. w is correct and classifies this example outside the margin), we would like $\tilde{w}$ to also be correct, though we require a smaller margin: $y\langle \tilde{w}, \Phi(x)\rangle \geq 1/2$.

These are equivalent to finding a solution with value at most $1/2$ to the following optimization problem:

$$\begin{aligned} \text{minimize }: & f(\tilde{w}) = \max_{i:h_i>0} \left(h_i - y_i\langle \tilde{w}, \Phi(x_i)\rangle\right) \qquad (4.1) \\ \text{where }: & h_i = \min\left(1, y_i\langle w, \Phi(x_i)\rangle\right) \end{aligned}$$

We will show that a randomized classifier based on a solution to Problem 4.1 with $f(\tilde{w}) \leq 1/2$ has empirical 0/1 error bounded by the empirical hinge loss of w; that we can efficiently find such solution based on at most $4\|w\|^2$ support vectors; and that such a sparse solution generalizes as well as w itself.

4.1. The slant-loss

The key to our approach is our use of the *randomized* classification rule $\tilde{g}_{\tilde{w}}(x) = \langle \tilde{w}, \Phi(x)\rangle + Z$ for $Z \sim \text{Unif}[-1/2, 1/2]$, rather than the standard linear classification rule g_w defined in Section 2. The effect of the randomization is to "spread out" the expected loss of the classifier. We define the loss function:

$$\ell_{slant}(z) = \min\left(1, \max\left(0, 1/2 - z\right)\right) \qquad (4.2)$$

which we call the "slant-loss" (Figure 4.1), using $\mathcal{L}_{slant}(g)$ and $\hat{\mathcal{L}}_{slant}(g)$ to denote its expectation and empirical average, analogously to the 0/1 and hinge losses. It is easy to see that $\mathbb{E}_Z\left[\ell_{0/1}(\tilde{g}_{\tilde{w}})\right] = \ell_{slant}(g_{\tilde{w}})$—hence, the 0/1 loss of $\tilde{g}_{\tilde{w}}$ is the same as the slant-loss of $g_{\tilde{w}}$. Equally importantly, $\ell_{slant}(z - 1/2) \leq \ell_{hinge}(z)$, from which the following Lemma follows:

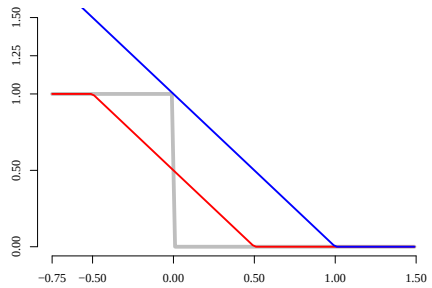

Figure 1. Illustration of how the slant-loss (red) relates to the 0/1 (gray) and hinge (blue) losses. Notice that, if the slant-loss is shifted by $1/2$, then it is still upper bounded by the hinge loss.

Lemma 4.1. *For any w, and any $\tilde{w}$ for which Problem 4.1 has value $f(\tilde{w}) \leq 1/2$, we have that*

$$\mathbb{E}_Z\left[\hat{\mathcal{L}}_{0/1}\left(\tilde{g}_{\tilde{w}}\right)\right] = \hat{\mathcal{L}}_{slant}\left(g_{\tilde{w}}\right) \leq \hat{\mathcal{L}}_{hinge}\left(g_w\right)$$

Proof. It remains only to establish that $\hat{\mathcal{L}}_{slant}(g_{\tilde{w}}) \leq \hat{\mathcal{L}}_{hinge}(g_w)$. For every x_i, y_i, consider the following three cases:

1. If $y_i\langle w, \Phi(x_i)\rangle \leq 0$, then $\ell_{slant}(y_i g_{\tilde{w}}(x_i)) \leq 1 \leq \ell_{hinge}(y_i g_w(x_i))$.
2. If $0 < y_i\langle w, \Phi(x_i)\rangle < 1$, then $\ell_{slant}(y_i g_{\tilde{w}}(x_i)) \leq \ell_{slant}(y_i g_w(x_i) - 1/2) \leq \ell_{hinge}(y_i g_w(x_i))$.
3. If $y_i\langle w, \Phi(x_i)\rangle \geq 1$, then $\ell_{slant}(y_i g_{\tilde{w}}(x_i)) \leq \ell_{slant}(1/2) = 0 = \ell_{hinge}(y_i g_w(x_i))$.

Hence, $\ell_{slant}(y_i g_{\tilde{w}}(x_i)) \leq \ell_{hinge}(y_i g_w(x_i))$, which completes the proof. □

4.2. Finding sparse solutions

To find a sparse $\tilde{w}$ with value $f(\tilde{w}) \leq 1/2$, we apply subgradient descent to Problem 4.1. The algorithm is extremely straightforward to understand and implement. We initialize $\tilde{w}^{(0)} = 0$, and then proceed iteratively, performing the following at the tth iteration:

1. Find the training index $i_t : y_{i_t}\langle w, \Phi(x_{i_t})\rangle > 0$ which maximizes $h_{i_t} - y_{i_t}\left\langle \tilde{w}^{(t-1)}, \Phi(x_{i_t})\right\rangle$.
2. Take the subgradient step $\tilde{w}^{(t)} \leftarrow \tilde{w}^{(t-1)} + \eta y_{i_t}\Phi(x_{i_t})$.

The convergence rate of this algorithm is characterized in the following lemma:

Lemma 4.2. *After $T \leq 4\|w\|^2$ iterations of subgradient descent, we obtain a solution of the form*

$\tilde{w} = \frac{1}{2}\sum_{t=1}^{T} y_{i_t}\Phi(x_{i_t})$ *which has value* $f(\tilde{w}) \leq 1/2$.

Proof. First note that $f(w) \leq 0$. Relying on this possible solution w, the Lemma follows from standard convergence bounds of subgradient descent (see e.g. Section 1.2 of Nesterov (2009)): with the step size $\eta = \epsilon$, after performing $\|w\|^2/\epsilon^2$ iterations, at least one iterate $\tilde{w}^{(t)}$ will have an objective function value no greater than ϵ. Choosing $\epsilon = 1/2$ gives the desired result. □

Because each iteration adds at most one new element to the support set, the support size of the solution will likewise be bounded by $4\|w\|^2$.

4.3. Generalization guarantee

The fact that the optimization procedure outlined in the previous section results in a sparse predictor of a particularly simple structure enables us to bound its generalization error using a compression bound:

Lemma 4.3. *With probability at least* $1-\delta$ *over the training set, for all* $\tilde{w}$ *of the form* $\tilde{w} = \frac{1}{2}\sum_{t=1}^{T} y_{i_t}\Phi(x_{i_t})$*, where* (x_{i_t}, y_{i_t}) *is any sequence of training examples, we have:*

$$\mathcal{L}_{slant}(g_{\tilde{w}}) \leq \hat{\mathcal{L}}_{slant}(g_{\tilde{w}}) + O\left(\frac{T\log n + \log\frac{1}{\delta}}{n} + \sqrt{\hat{\mathcal{L}}_{slant}(g_{\tilde{w}})\frac{T\log n + \log\frac{1}{\delta}}{n}}\right)$$

provided that $n \geq 8T$.

Proof. This follows directly from compression bounds, e.g. Theorem B.1 (included in Appendix B for completeness), since $\tilde{w}$ is a fixed function (namely a sum) of at most T sample points. □

Instead of using a compression bound, it is also possible to use uniform concentration arguments to obtain an almost identical guarantee (up to log factors) which holds for any $\tilde{w}$ (not necessarily sparse) with norm $\|\tilde{w}\| \leq \|w\|$ and value $f(\tilde{w}) \leq 1/3$ (see Appendix C). This could be used to justify other optimization approaches to Problem 4.1. However, for the method presented, the compression bound suffices.

4.4. Putting it together

Now that all of the pieces are in place, we can state our final procedure, start-to-finish:

1. Train a SVM to obtain w with norm $\|w\| \leq O(\|u\|)$ and $\hat{\mathcal{L}}_{hinge}(g_w) \leq \hat{\mathcal{L}}_{hinge}(g_u) + O(\epsilon)$.
2. Run subgradient descent on Problem 4.1 until we find a predictor $\tilde{w}$ with value $f(\tilde{w}) \leq 1/2$.
3. Predict using $\tilde{g}_{\tilde{w}}$.

Theorem 4.4. *For an (unknown)* u*, with probability at least* $1-2\delta$ *over a training set of size:*

$$n = \tilde{O}\left(\left(\frac{\mathcal{L}_{hinge}(g_u)+\epsilon}{\epsilon}\right)\frac{\|u\|^2}{\epsilon}\log\frac{1}{\delta}\right)$$

the procedure above finds a predictor $\tilde{w}$ *supported on at most* $O(\|u\|^2)$ *training vectors and error* $\mathcal{L}_{0/1}(\tilde{g}_{\tilde{w}}) \leq \mathcal{L}_{hinge}(g_u) + O(\epsilon)$

Proof. First, note that with the specified sample size, applying Bernstein's inequality to the fixed predictor u, we have that with probability at least $1-\delta$,

$$\hat{\mathcal{L}}_{hinge}(u) \leq \mathcal{L}_{hinge}(u) + \epsilon. \tag{4.3}$$

Combining Equation 4.3 with the SVM training goal (Step 1) and Lemma 4.1, we have that $\hat{\mathcal{L}}_{slant}(g_{\tilde{w}}) \leq \mathcal{L}_{hinge}(u) + O(\epsilon)$. Following Lemma 4.2 we can apply Lemma 4.3 with $T = O(\|u\|^2)$, and plugging in the specified sample complexity, we have $\mathcal{L}_{slant}(g_{\tilde{w}}) \leq \hat{\mathcal{L}}_{slant}(g_{\tilde{w}})$. Combining the two inequalities, and recalling that the slant-loss of $g_{\tilde{w}}$ is the same as the expected 0/1 error of $\tilde{g}_{\tilde{w}}$, we obtain $\mathcal{L}_{0/1}(\tilde{g}_{\tilde{w}}) \leq \mathcal{L}_{hinge}(g_u) + O(\epsilon)$. Lemma 4.2 also establishes the desired bound on the number of support vectors. □

The procedure is efficient, and aside from initial SVM optimization, requires at most $O(\|u\|^2)$ iterations.

4.5. Kernelization

To this point, we have worked in the explicit kernel Hilbert space $\mathcal{H}$, even though we are interested primarily in the kernelized case, where $\mathcal{H}$ and $\Phi(\cdot)$ are specified only implicitly through $K(x,x') = \langle\Phi(x),\Phi(x')\rangle$. We now show how our procedure can be "kernelized" and implemented using only $K(\cdot,\cdot)$.

As is typical, we shall rely on the representations $w = \sum_{i=1}^{n}\alpha_i y_i\Phi(x_i)$ and $\tilde{w} = \sum_{i=1}^{n}\tilde{\alpha}_i y_i\Phi(x_i)$, keeping track of the coefficients α and $\tilde{\alpha}$. We will also maintain an up-to-date vector of "responses" $\tilde{c}$:

$$\tilde{c}_j = y_j\langle\tilde{w},\Phi(x_j)\rangle = \sum_{i=1}^{n}\tilde{\alpha}_i y_i y_j K(x_i,x_j)$$

Notice that the values of these responses provide sufficient knowledge to find the update index i at every iteration. We can then perform the subgradient descent update $\tilde{w} \leftarrow \tilde{w} + \eta y_i\Phi(x_i)$ by adding η to $\tilde{\alpha}_i$, and updating the responses as $\tilde{c}_j \leftarrow \tilde{c}_j + \eta y_i y_j K(x_i,x_j)$, at

a total cost of n kernel evaluations. These kernel evaluations dominate the computational cost of our gradient descent procedure (all other operations can be performed in $O(n)$ per iteration). As is standard for kernel SVM training, we will therefore analyze runtime in terms of the number of kernel evaluations required.

With $O(\|u\|^2)$ iterations, and $O(n)$ kernel evaluations per iteration, the overall number of required kernel evaluations for the gradient descent procedure is (ignoring the δ dependence):

$$O\left(n\|u\|^2\right) = \tilde{O}\left(\left(\frac{\hat{\mathcal{L}}_{hinge}(g_u)+\epsilon}{\epsilon}\right)\frac{\|u\|^4}{\epsilon}\right)$$

This is *less* then the best known runtime bound for kernel SVM optimization, so we do not expect the sparsification step to be computationally dominant (i.e. it is in a sense "free"). In order to complete the picture and understand the entire runtime of our method, we must also consider the runtime of the SVM training (Step 1). The best kernelized SVM optimization guarantee of which we are aware is achieved by the Stochastic Batch Perceptron (SBP, Cotter et al. (2012a)). Using the SBP, we can find w with $\|w\| \leq 2\|u\|$ and $\hat{\mathcal{L}}_{hinge}(g_w) \leq \hat{\mathcal{L}}_{hinge}(g_u) + \epsilon$ using:

$$O\left(\left(\frac{\hat{\mathcal{L}}_{hinge}(g_u)+\epsilon}{\epsilon}\right)^2\|u\|^2 n\right)$$

kernel evaluations, yielding (with the δ-dependence):

Corollary 1. *If using the SBP for Step 1 and the sample size required by Theorem 4.4, the procedure in Section 4.4 can be performed with:*

$$\tilde{O}\left(\left(\frac{\hat{\mathcal{L}}_{hinge}(g_u)+\epsilon}{\epsilon}\right)^3\frac{\|u\|^4}{\epsilon}\log\frac{1}{\delta}\right)$$

kernel evaluations.

Because the SBP finds a w with $\|w\| \leq 2\|u\|$, and our subgradient descent algorithm finds a $\tilde{w}$ supported on $4\|w\|^2$ training vectors, it follows that the support size of $\tilde{w}$ is bounded by $16\|u\|^2$. The runtime of Step 2 (the sparsification procedure) is asymptotically negligible compared to Step 1 (initial SVM training), so the overall runtime is the same as for stand-alone SBP. Overall, our procedure finds an optimally sparse SVM predictor, and at the same time matches the best known sample and runtime complexity guarantees for SVM learning (up to small constant factors).

4.6. Unregularized bias

Frequently, SVM problems contain an unregularized bias term—rather than the classification function being the sign of $g_w(x) = \langle w, \Phi(x)\rangle$, it is the sign of $g_{w,b}(x) = (\langle w, \Phi(x)\rangle + b)$ for a weight vector w and a bias b, where the bias is unconstrained, being permitted to take on the value of any real number.

When optimizing SVMs, the inclusion of an unregularized bias introduces some additional complications which typically require special treatment. Our subgradient descent procedure, however, is essentially unchanged by the inclusion of a bias (although the SVM solver which we use to find w and b must account for it). Indeed, we need only redefine:

$$h_i = \min\left(1, y_i\left(\langle w, \Phi(x_i)\rangle + b\right)\right) - y_i b$$

in Problem 4.1, and then find $\tilde{w}$ as usual. The resulting sparse classifier is parameterized by $\tilde{w}$ and b, with b being that of the initial SVM solution.

5. Related Algorithms

The tendency of SVM training algorithms to find solutions with large numbers of support vectors has been recognized as a shortcoming of SVMs since their introduction, and many approaches for finding solutions with smaller support sizes have been proposed, of varying levels of complexity and effectiveness.

We group these approaches into two categories: those which, like ours, start with a non-sparse solution to the SVM problem, and then find a sparse approximation; and those which either modify the SVM objective so as to result in sparse solutions, or optimize it using an algorithm specifically designed to maximize sparsity.

In this section, we will discuss previous work of both of these types. None of these algorithms have performance guarantees which can be compared to that of Theorem 4.4, so we will also discuss some algorithms which do not optimize the SVM objective (even approximately), but do find sparse solutions, and for some of which generalization bounds have been proven. In section 7, we also report on empirical comparisons to some of the methods discussed here.

5.1. Post-hoc approximation approaches

One of the earliest proposed methods for finding sparse SVM solutions was that of Osuna & Girosi (1998), who suggest that one first solve the kernel SVM optimization problem to find w, and then, as a post-processing step, find a sparse approximation $\tilde{w}$ using support vector regression (SVR), minimizing the aver-

age ϵ-insensitive loss, plus a regularization penalty:

$$\text{minimize} : f_{OG}(\tilde{w}) = \frac{1}{2}\|\tilde{w}\|^2 + \tilde{C}\sum_{i=1}^{n}\max\left(0, |\langle w, \Phi(x_i)\rangle - \langle \tilde{w}, \Phi(x_i)\rangle| - \epsilon\right) \tag{5.1}$$

Optimizing this problem results in a $\tilde{w}$ for which the numerical values of $\langle w, \Phi(x)\rangle$ and $\langle \tilde{w}, \Phi(x)\rangle$ must be similar, even for examples which w misclassifies. This is an unnecessarily stringent condition—because the underlying problem is one of classification, we need only find a solution which gives roughly the same classifications as w, without necessarily matching the value of the classification function g_w. It is in this respect that our objective function, Problem 4.1, differs.

Osuna and Girosi's work was later used as a key component of the work of Zhan & Shen (2005), who first solve the SVM optimization problem, and then exclude a large number of support vectors from the training set based on a "curvature heuristic". They then retrain the SVM on this new, smaller, training set, and finally apply the technique of Osuna and Girosi to the result.

5.2. Alternative optimization strategies

Another early method for finding sparse approximate SVM solutions is RSVM (Lee & Mangasarian, 2001), which randomly samples a subset of the training set, and then searches for a solution supported only on this sample, minimizing the loss on the *entire* training set.

So-called "reduced set" methods (Burges & Schölkopf, 1997; Wu et al., 2005) address the problem of large support sizes by removing the constraint that the SVM solution be supported on the training set. Instead it is now supported on a set of "virtual training vectors" $z_1, \ldots, z_k$ with $k \ll n$, while having the same form as the standard SVM solution: $\text{sign}\left(\sum_{i=1}^{k}\beta_i y_i K(x, z_i)\right)$. One must optimize over not just the coefficients β_i, but *also* the virtual training vectors z_i. Because the support set is not a subset of the training set, our lower bound (Lemma 3.1) does not apply. However, the resulting optimization problem is non-convex, and is therefore difficult to optimize.

More recently, techniques such as those of Joachims & Yu (2009) and Nguyen et al. (2010) have been developed which, rather than explicitly including the search for good virtual training vectors in the objective function, instead find such vectors heuristically during optimization. These approaches have the significant advantage of not explicitly relying on the optimization of a non-convex problem, although in a sense this difficulty is being "swept under the rug" through the use of heuristics.

Another approach is that of Keerthi et al. (2006), who optimize the standard SVM objective function while explicitly keeping track of a support set S. At each iteration, they perform a greedy search over all training vectors $x_i \notin S$, finding the x_i such that the optimal solution supported on $S \cup \{x_i\}$ is best. They then add x_i to S, and repeat. This is another extremely well-performing algorithm, but while the authors propose a clever method for improving the computational cost of their approach, it appears that it is still too computationally expensive to be used on very large datasets.

5.3. Non-SVM algorithms

Collobert et al. (2006) modify the SVM objective to minimize not the convex hinge loss, but rather the non-convex "ramp loss", which differs from our slant-loss only in that the ramp covers the range $[-1, 1]$ instead of $[-1/2, 1/2]$. Because the resulting objective function is non-convex, it is difficult to find a global optimum, but the experiments of Collobert et al. (2006) show that local optima achieve essentially the same performance with smaller support sizes than solutions found by "standard" SVM optimization.

Another approach for learning kernel-based classifiers is to use online learning algorithms such as the Perceptron (e.g. Freund & Schapire (1999)). The Perceptron processes the training example one by one and adds a support vector only when it makes a prediction mistake. Therefore, a bound on the number of prediction mistakes (i.e. a mistake bound) translates to a bound on the sparsity of the learned predictor.

As was discussed in Section 3, if the data are separable with margin 1 by some vector u, then the Perceptron can find a very sparse predictor with low error. However, in the non-separable case, the Perceptron might make a number of mistakes that grows linearly with the size of the training sample.

To address this, online learning algorithms for which the support size is bounded by a *budget parameter* have been proposed. Notable examples include the Forgetron (Dekel et al., 2005) and the Randomized Budget Perceptron (RBP, Cavallanti et al. (2007)). Such algorithms discard support vectors when their number exceeds the budget parameter—for example, the RBP discards an example chosen uniformly at random from the set of support vectors, whenever needed.

Both of these algorithms have been analyzed, but the resulting mistake bounds are inferior to that of the Perceptron, leading to worse generalization bounds than the one we achieve for our proposed procedure,

for the same support size. For example, the generalization bound of the Forgetron is at least $4\hat{\mathcal{L}}_{hinge}(g_u)$. The bound of the RBP is more involved, but it is possible to show that in order to obtain a support size of $16\|u\|^2$, the generalization bound would depend on at least $(5/3)\hat{\mathcal{L}}_{hinge}(g_u)$. In contrast, the bound we obtain only depends on $\hat{\mathcal{L}}_{hinge}(g_u)$.

6. Practical Variants

While the algorithm described in Section 4 gives asymptotically optimal theoretical performance, slight variations of it give better empirical performance.

The analysis of Theorem 4.4 bounds the performance of the *randomized* classifier $\tilde{g}_{\tilde{w}}$, but we have found that randomization *hurts* performance in practice, and that one is better off predicting using $\text{sign}(g_{\tilde{w}})$. Our technique relies on finding an approximate solution $\tilde{w}$ to Problem 4.1 with $f(\tilde{w}) \leq 1/2$, but this $1/2$ threshold is a relic of our use of randomization. Since randomization does not help in practice, there is little reason to expect there to be anything "special" about $1/2$—one may achieve a superior sparsity/generalization tradeoff at different levels of convergence, and with values of the step-size η other than the suggested value of $1/2$. For this reason, we suggest experimenting with different values of these parameters, and choosing the best based on cross-validation.

Another issue is the handling of an unregularized bias. In Section 4.6, we suggest taking the bias associated with $\tilde{w}$ to be the same as that associated with w. However, one could alternatively optimize a version of Problem 4.1 which learns a new bias $\tilde{b}$ along with $\tilde{w}$. The resulting subgradient descent algorithm (see Appendix D) is slightly more complicated, but its use may result in a small boost in performance.

6.1. Aggressive variant

A more substantial deviation from our basic algorithm is to try to be more aggressive about maintaining sparsity by re-using existing support vectors when optimizing Problem 4.1. This can be done in the the following way: at each iteration, check if there is a support vector (i.e. a training point already added to the support set) for which $h_i - \langle \tilde{w}, \Phi(x_i) \rangle \leq \epsilon$ (where ϵ is the termination threshold, $1/2$ in the analysis of Section 4). If there is such a support vector, increase its coefficient α_i—only take a step on index i which is not currently in the support set if all current support vectors satisfy the constraint. This yields a potentially sparser solution at the cost of more iterations.

Table 1. Datasets used in our experiments. Except for TIMIT, these are a subset of the datasets, with the same parameters, as were compared in Nguyen et al. (2010). We use a Gaussian kernel $K(x, x') = \exp(-\gamma \|x - x'\|)$ with parameter γ, and regularization tradeoff parameter C.

	Training	Testing	γ	C
Adult	22696	9865	0.1	1
IJCNN	35000	91701	1	10
Web	49749	14951	0.1	10
TIMIT	63881	22257	0.025	1
Forest	522910	58102	0.0001	10000

7. Experiments

Basing our experiments on recent comparisons between sparse SVM optimizers (Keerthi et al., 2006; Nguyen et al., 2010), we compare our implementation[3] to the following methods[4]:

1. SpSVM (Keerthi et al., 2006), using Olivier Chapelle's implementation[5].
2. CPSP (Joachims & Yu, 2009), using SVM-Perf.
3. Osuna & Girosi's algorithm (Osuna & Girosi, 1998), using LIBSVM (Chang & Lin, 2001) to optimize the resulting SVR problems.
4. RSVM (Lee & Mangasarian, 2001), using the LIBSVM Tools implementation (Lin & Lin, 2003).
5. CSVM (Nguyen et al., 2010). We did not perform these experiments ourselves, and instead present the results reported in the CSVM paper.

Our comparison was performed on the datasets listed in Table 1. Adult and IJCNN are the "a8a" and "ijcnn1" datasets from LIBSVM Tools. Web and Forest are from the LibCVM Toolkit[6]. We also use a multiclass dataset[7] derived from the TIMIT speech corpus, on which we perform one-versus-rest classification, with class number 3 (the phoneme `/k/`) providing the "positive" instances. Both Adult and TIMIT have relatively high error rates, making them more challenging for sparse SVM solvers. Both our algorithm and that of Osuna & Girosi require a reference classifier w, found using GTSVM (Cotter et al., 2011).

We experimented with two versions of our algorithm, both incorporating the modifications of Section 6, dif-

[3] http://ttic.uchicago.edu/~cotter/projects/SBP

[4] We were unable to find a reduced set implementation on which we could successfully perform our experiments

[5] http://olivier.chapelle.cc/primal

[6] http://c2inet.sce.ntu.edu.sg/ivor/cvm.html

[7] http://ttic.uchicago.edu/~cotter/projects/gtsvm

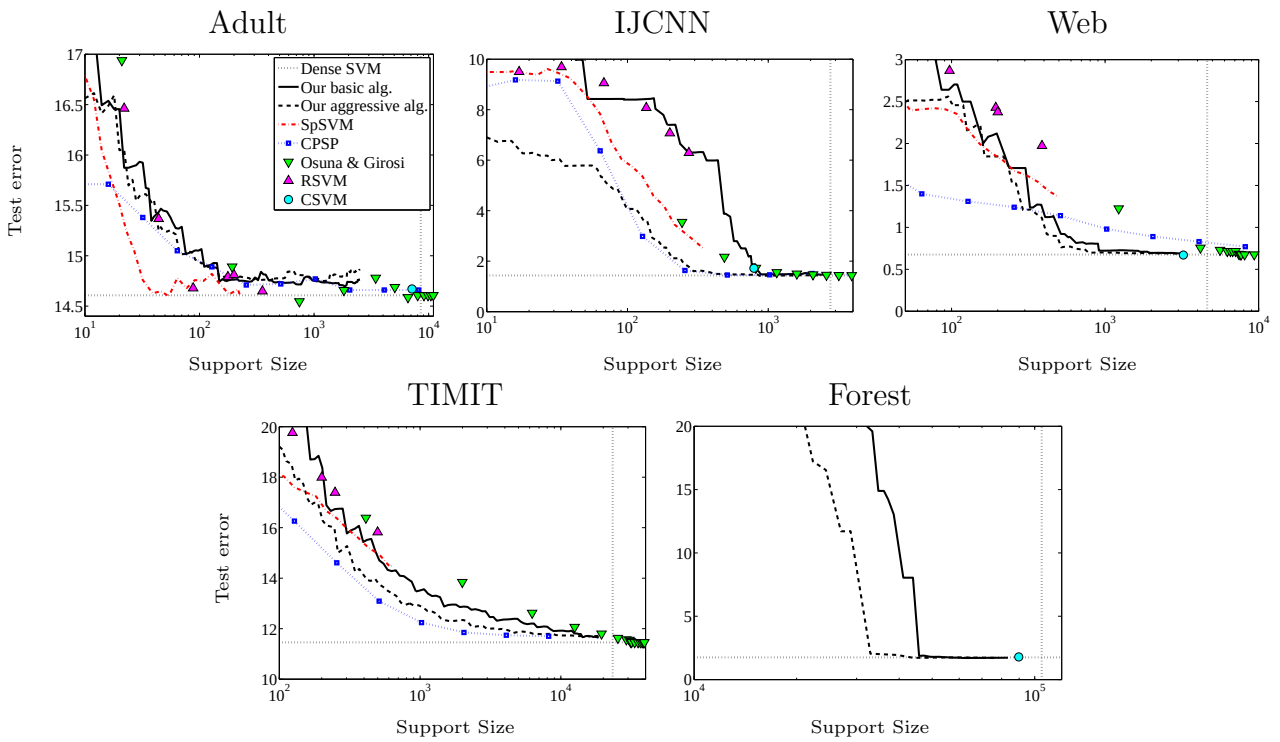

Figure 2. Plots of test error (linear scale) versus support size (log scale). The horizontal and vertical dotted lines are the test error rate and support size of the classifier found by GTSVM. TIMIT was not included in the experiments of Nguyen et al. (2010). On Forest, SpSVM ran out of memory, CPSP failed to terminate in one week for 4096 or more basis functions, LIBSVM failed to optimize the SVR problem (Problem 5.1) in 4 days for $\epsilon < 1$, and RSVM's solutions were limited to 200 support vectors, far too few to perform well on this dataset.

fering only in whether they include the aggressive variation. For the "basic" version, we tried $\eta = \{4^{-4}, 4^{-3}, \ldots, 4^2\}$, keeping track of the progress of the algorithm throughout the course of optimization. For each support size, we chose the best η based on a validation set (half of the original test set) reporting errors on an independent test set (the other half). This was then averaged over 100 random test/validation splits.

For the aggressive variant (Section 6.1), we experimented not only with multiple choices of η, but also termination thresholds $\epsilon \in \{2^{-4}, 2^{-3}, \ldots, 1\}$, running until this threshold was satisfied. Optimization over the parameters was then performed using the same validation approach as for the "basic" algorithm.

In our CPSP experiments, the target numbers of basis functions were taken to be powers of two. For Osuna & Girosi's algorithm, we took the SVR regularization parameter $\tilde{C}$ to be that of Table 1 (i.e. $\tilde{C} = C$), and experimented with $\epsilon \in \{2^{-16}, 2^{-15}, \ldots, 2^4\}$. For RSVM, we tried subset ratios $\nu \in \{2^{-16}, 2^{-15}, \ldots, 1\}$—however, the implementation we used was unable to find a support set of size larger than 200, so many of the larger values of ν returned duplicate results.

The results are summarized in Figure 7. Our aggressive variant achieved a test error / support size tradeoff comparable to or better than the best competing algorithms, except on the Adult and TIMIT datasets, on the latter of which performance was fairly close to that of CPSP. On the Adult data set, the test errors (reported) are significantly higher then the validation errors, indicating our methods are suffering from parameter overfitting due to too small a validation set (this is also true, to a lesser degree, on TIMIT). Note that SpSVM and CPSP, both of which perform very well, failed to find good solutions on the forest dataset within a reasonable timeframe, illustrating the benefits of the simplicity of our approach.

To summarize, not only does our proposed method achieve optimal theoretical guarantees (the best possible sparseness guarantee with the best known sample complexity and runtime for kernelized SVM learning), it is also computationally inexpensive, simple to implement, and performs well in practice.

Acknowledgments: S. Shalev-Shwartz is supported by the Israeli Science Foundation grant number 598-10.

References

Burges, C. and Schölkopf, B. Improving the accuracy and speed of support vector machines. In *NIPS'97*, pp. 375–381. MIT Press, 1997.

Cavallanti, G., Cesa-Bianchi, N., and Gentile, C. Tracking the best hyperplane with a simple budget perceptron. *Machine Learning*, 69(2-3), December 2007.

Chang, C-C. and Lin, C-J. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Collobert, Ronan, Sinz, Fabian, Weston, Jason, and Bottou, Léon. Trading convexity for scalability. In *ICML'06*, pp. 201–208, 2006.

Cotter, A., Srebro, N., and Keshet, J. A GPU-tailored approach for training kernelized SVMs. In *KDD'11*, 2011.

Cotter, A., Shalev-Schwartz, S., and Srebro, N. The kernelized stochastic batch perceptron. In *ICML'12*, 2012a.

Cotter, A., Shalev-Schwartz, S., and Srebro, N. The kernelized stochastic batch perceptron. http://arxiv.org/abs/1204.0566, 2012b.

Dekel, O., Shalev-Shwartz, S., and Singer, Y. The forgetron: A kernel-based perceptron on a fixed budget. In *NIPS'05*, pp. 259–266, 2005.

Freund, Y. and Schapire, R. E. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

Joachims, T. and Yu, Chun-Nam. Sparse kernel svms via cutting-plane training. *Machine Learning*, 76(2–3):179–193, 2009. European Conference on Machine Learning (ECML) Special Issue.

Keerthi, S. Sathiya, Chapelle, Olivier, and DeCoste, Dennis. Building support vector machines with reduced classifier complexity. *Journal of Machine Learning Research*, 7:1493–1515, 2006.

Lee, Y-J. and Mangasarian, O. RSVM: Reduced support vector machines. In *Data Mining Institute, Computer Sciences Department, University of Wisconsin*, pp. 00–07, 2001.

Lin, K-M. and Lin, C-J. A study on reduced support vector machines. *IEEE Transactions on Neural Networks*, 2003.

Nesterov, Y. Primal-dual subgradient methods for convex problems. *Math. Program.*, 120(1):221–259, Apr 2009.

Nguyen, D D., Matsumoto, K., Takishima, Y., and Hashimoto, K. Condensed vector machines: learning fast machine for large data. *Trans. Neur. Netw.*, 21(12):1903–1914, Dec 2010.

Osuna, E. and Girosi, F. Reducing the run-time complexity of support vector machines, 1998.

Shalev-Shwartz, S. Introduction to machine learning, lecture notes. Technical report, The Hebrew University, 2010. http://www.cs.huji.ac.il/~shais/Handouts2010.pdf.

Srebro, N., Sridharan, K., and Tewari, A. Smoothness, low-noise and fast rates. In *NIPS'10*, 2010.

Wu, M., Schölkopf, B., and Bakir, G. Building sparse large margin classifiers. In *ICML'05*, pp. 996–1003, New York, NY, USA, 8 2005. Max-Planck-Gesellschaft, ACM.

Zhan, Y. and Shen, D. Design efficient support vector machine for fast classification. *Pattern Recognition*, 38(1):157–161, 2005.