Kernel SVMs

Setting
- Training examples $x_1, ..., x_n$ with labels $y_1, ..., y_n \in \{\pm 1\}$
- Kernel function $K(x_i, x_j) = \phi(x_i), \phi(x_j)$
- Weight vector $w = \sum_i \alpha_i y_i \phi(x_i)$, so $\langle w, \phi(x) \rangle = \sum_i \alpha_i y_i K(x_i, x)$

The SVM objective is:
$$\text{minimize} \quad \frac{1}{2} ||w||^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i \langle w, \phi(x_i) \rangle)$$

We are interested in predictors with $L$ support vectors:
- An example $x_i$ is classified using the sign of $\langle w, \phi(x_i) \rangle$,
- The classifier runtime is proportional to the number of support vectors: $|\alpha|_0$.
- The classifier’s memory footprint is, likewise, $|\alpha|_0$, training vectors.

For traditional SVM optimizers, every classification error—indeed, every margin violation—will be a support vector. We can do better!

Contributions
- We show that it’s possible to have a support size of $O(\sqrt{\log n})$ without losing accuracy.
- This is tight (up to a constant factor)—no algorithm has a smaller support size without losing accuracy.
- Our approach increases neither the asymptotic runtime nor the sample complexity over the best known kernel SVM training guarantees.

Intuition
To see why we seek a solution with $f(w) \leq \frac{1}{2}$, consider how such a solution behaves on the training data:

Experiments

### Adult

#### Error

<table>
<thead>
<tr>
<th>Support size</th>
<th>Testing 01 error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^0$</td>
<td>1.70</td>
</tr>
<tr>
<td>$10^1$</td>
<td>1.50</td>
</tr>
<tr>
<td>$10^2$</td>
<td>1.30</td>
</tr>
</tbody>
</table>

#### IJCNN

<table>
<thead>
<tr>
<th>Support size</th>
<th>Testing 01 error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^0$</td>
<td>1.70</td>
</tr>
<tr>
<td>$10^1$</td>
<td>1.50</td>
</tr>
<tr>
<td>$10^2$</td>
<td>1.30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Support size</th>
<th>Testing 01 error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^0$</td>
<td>1.70</td>
</tr>
<tr>
<td>$10^1$</td>
<td>1.50</td>
</tr>
<tr>
<td>$10^2$</td>
<td>1.30</td>
</tr>
</tbody>
</table>

Please see our paper for citations of the compared algorithms. Our aggressive algorithm is a variant of our basic subgradient descent approach which makes a greater effort to ensure sparsity during optimization, at the cost of increased runtime.

Method

We propose a two-step optimization procedure: first, solve the SVM using a standard optimizer, and next find a sparse approximation to the SVM solution:

1. Optimize the SVM problem, yielding a solution $w$.
2. Find a sparse approximation $w'$ to $w$ by optimizing the following to within $f(w') \leq \frac{1}{2}$ using subgradient descent:
   $$\minimize : f(w') = \max_i (h_i - y_i \langle w', \phi(x_i) \rangle)$$
   $$\text{where: } h_i = \min_{i' > 0} (1, y_i \langle w, \phi(x_i) \rangle)$$
3. Predict using a randomized classification rule: the sign of $\langle w', \phi(x) \rangle + Z$, where $Z$ is a uniform random variable on $[-\frac{1}{2}, \frac{1}{2}]$.

If we optimize using $O(||w||^2)$ iterations of subgradient descent, then:
- $f(w') \leq \frac{1}{2}$ with high probability.
- $w'$ generalizes just as well as $w$, again with high probability.
- $w'$ will be supported on $O(||w||^2)$ training examples.
- The computational cost of finding $w'$ is negligible compared to the (best known) cost of finding $w$.

Combined with the initial SVM optimization step, for our overall procedure:

If the number of training samples satisfies:
$$n = O\left( \frac{\mathcal{L}_{\text{hinge}}(w') + \epsilon}{\epsilon} \frac{||w||^2}{\epsilon} \right)$$

Then our overall procedure yields a $w'$ for which, with high probability:
$$\mathcal{L}_{0:1}(w') \leq \mathcal{L}_{\text{hinge}}(w') + \epsilon$$

Furthermore:
- The support size $|\alpha|^0$ of $w'$ satisfies $|\alpha|^0 = O(\|w\|^2)$.
- The computational cost of the entire procedure is dominated by that of the initial SVM optimization.

Observe that the support size of $w'$ has no $\epsilon$ dependence.

Matching lower bound:
- A data distribution exists for which any solution which generalizes as well as $w'$ must have at least $\|w\|^2/2$ support vectors.
- Hence, in terms of a worst-case analysis, our algorithm is optimal up to a constant factor.