Incremental Training of Support Vector Machines

A. Shilton*, M. Palaniswami*, D. Ralph***, A. C. Tsoi**

*Department of Electrical and Electronic Engineering
The University of Melbourne, Victoria-3010, Australia
apsh@ee.mu.oz.au
swami@ee.mu.oz.au

**University of Wollongong
Wollongong, NSW, Australia
act@uow.edu.au

***The Judge Institute of Management Studies
University of Cambridge
Trumpington St, Cambridge CB2 1AG, UK.
danny.ralph@jims.cam.ac.uk

Abstract

We consider the situation in which an optimal SVM for a particular set of training data has been found and there is a need to add a small number of additional training vectors. We investigate active set training methods for incremental learning in Support Vector Machine. Using a simple quadratic hot-start method, we demonstrate the computational superiority of incremental methods over the usual batch methods with a sample application.

1 Introduction

Support Vector Machine (SVM) learning methods were originally developed by Vapnik et al [1], to tackle the problem of pattern recognition. The problem is thus – given a “training set” of vectors, each belonging to some known category, the machine must learn, based on the information implicitly contained in this set, how to classify vectors of unknown type into one of the specified categories.

Binary SVMs implicitly map the training data into a (usually) higher-dimensional feature space. A hyperplane (decision surface) is then constructed in this feature space that bisects the two categories and maximises the margin of separation between itself and those points lying nearest it (called the support vectors). This decision surface can then be used as a basis for classifying vectors of uncertain type.

The main advantages of the SVM approach are:

- SVMs are known to implement a form of structural risk minimisation [1] – they minimise a bound on generalisation error rather than empirical risk.
- The problem (in its dual form) is a convex quadratic programming problem. Hence it is readily solvable, and has no local minima.
- The resulting classifier can be specified completely in terms of its support vectors and kernel function type.

While many papers have been published on SVM training, relatively few have looked at the problem of incremental training. Most concentrate instead on batch training, in which the SVM is always trained from scratch, even if we are making only small changes to the training set. But if we are adding a small amount of data to a large training set, then it will likely have only a minimal effect on the decision surface. Re-solving the problem from scratch seems computationally wasteful.

An alternative is to “hot-start” our solution process by using our old solution as a starting point to find our new solution. This approach is called incremental learning. In this paper we investigate the computational cost savings that can be had by applying incremental methods to a practical problem of fish species identification.

2 Support Vector Machines

We now look briefly at how the SVM pattern recognition problem is formulated [3]. For simplicity, we will first consider the case of binary classification, and then extend this to the multiclass version.
We define our training set as:
\[
\Theta = \{(x_1, d_1), (x_2, d_2), \ldots, (x_N, d_N)\}
\]
\[x_i \in \mathbb{R}^{d_i}, \quad d_i \in \{+1, -1\}\]

We also define a mapping from our input space to a (usually) higher dimensional feature space as:
\[
\varphi(x) = \begin{bmatrix} \varphi_1(x) \\ \varphi_2(x) \\ \vdots \\ \varphi_{d_N}(x) \end{bmatrix} \in \mathbb{R}^{d_N}
\]

Assume that our two training classes are linearly separable when mapped into feature space. Then we can define a discriminant function \(g(x)\) thus:
\[
g(x) = w^T \varphi(x) + b
\]
\[g(x_i) > 0 \quad \forall \ d_i = +1
\]
\[g(x_i) < 0 \quad \forall \ d_i = -1
\]

The discriminant function may be used to classify points of unknown classification as follows:
\[d_i = \text{sgn}[g(y)]\]

Any discriminant function (2.3) defines a linear (flat) decision surface in feature space that bisects our two classes. This plane is characterised by:
\[w^T \varphi(x) + b = 0\]

However, there will be infinitely many such planes. To select the surface best suited to the task, the SVM maximises the distance between the decision surface and those training points lying closest to it (the support vectors). It is easy to show (see [2] for example) that maximising the distance is equivalent to solving
\[
\min_{w, b} \frac{1}{2} w^T w
\]
subject to
\[d_i (w^T \varphi(x_i) + b) \geq 1\]

If our training classes are not linearly separable in feature space, we must relax the inequalities in (2.6) using slack variables and modify our cost function to penalise any failure to meet the original (strict) inequalities. From [2]:
\[
\min_{w, \xi} \frac{1}{2} w^T w + C \sum \xi
\]
subject to
\[d_i (w^T \varphi(x_i) + b) \geq 1 - \xi_i \]
\[\xi \geq 0\]

This is computationally difficult, so we instead use the dual [2]:
\[
\min_{\alpha} \frac{1}{2} \alpha^T G \alpha - \alpha^T \mathbf{1}
\]
such that
\[d^T \alpha = 0 \quad \alpha \geq 0 \quad \alpha \leq C\]

Where:
\[G \in \mathbb{R}^{N \times N} \quad G_{ii} = d_i d_j K(x_i, x_j)
\]

\[K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)\]

We can express our discriminant as:
\[g(y) = \sum_{i=1}^{N} \alpha_i d_i K(x_i, y) + b\]
\[b = d_j \left(1 - \sum_{i=1}^{N} \alpha_i G_{ij}\right), \quad 0 < \alpha_i < C\]

We will be using the dual form in this paper. Note that:
- There is exactly one \(\alpha\) for every training vector.
- Only those \(\alpha\)'s corresponding to support vectors will be non-zero. Hence the SVM can be fully specified using only the support vectors.
- The matrix \(G\) is positive semi-definite and the constraints are linear. Hence the program is convex.
- The dimension of the feature space is hidden by the kernel function \(K(x, y)\), circumventing the “curse of dimensionality” which affects the primal form.
- So long as our kernel function satisfies Mercer’s condition [5], we need never explicitly know what our feature mapping actually is.

### 2.1 Multiclass Classification – Voting Algorithms
For our example, we are dealing with more than two classes of data (5 in fact). We have achieved this using a simple voting algorithm constructed as follows:
- Suppose we have \(K\) categories of data.
- We train a 1-versus-2 binary classifier (using training points from classes 1 for “-1” points and 2 for “+1” points), a 1-versus-3 binary classifier (using training points from classes 1 and 3), a 2-versus-3 and so on until we dealt with all non-trivial combinations.
Once the machines have been trained, we can classify an unknown vector $\mathbf{y}$ as follows:

- Each binary classifier will vote for one or other of its two given classes.
- The vector $\mathbf{y}$ is taken to belong to the class that receives the most votes.

Assuming that the vector $\mathbf{y}$ does not directly on any decision surface, there can be no ambiguity in the vote (i.e. no draws will occur).

3 Method of Solving Dual Problem

We implemented a simple hot-start iterative active set solver for this problem [4]. While reasonably efficient, the algorithm presented here is not optimal. However, it is good enough to demonstrate the potential of the incremental approach, and the computational cost savings which may be achieved in comparison to the usual batch method.

The dual form of the SVM problem is:

$$
\min_{\alpha} Q(\alpha) = \frac{1}{2} \alpha^T G \alpha - \alpha^T \mathbf{1}
$$

such that

- $\mathbf{d}^T \alpha = 0$
- $\alpha \geq 0$
- $\alpha \leq C1$

Suppose we start at point $\alpha(0)$. This will not necessarily satisfy the equality constraint in (3.1), so we define (at each iteration):

$$
j^{(k)} = \mathbf{d}^T \alpha^{(k)}
$$

The gradient of $Q(\alpha)$ is:

$$
e^{(k)} = G \alpha^{(k)} - \mathbf{1}
$$

We can partition the problem thus:

$$
G = \begin{bmatrix}
\hat{G} & \mathbf{g} \\
\mathbf{g}^T & g_E^2
\end{bmatrix}, \quad \alpha = \begin{bmatrix}
\alpha \\
\alpha_E
\end{bmatrix}, \quad e = \begin{bmatrix}
e_U \\
ice_E
\end{bmatrix}
$$

Such that $0 < \alpha_E < C$.

Using this, we can eliminate the equality constraint to get the reduced problem:

$$
\min_{\alpha} \tilde{Q}(\alpha) = \frac{1}{2} \alpha^T \hat{H} \alpha + \alpha^T \mathbf{h}
$$

such that

- $\alpha \geq 0$
- $-\mathbf{d}^T \alpha \geq 0$
- $\alpha \leq C1$

Where:

$$
\hat{H} = \hat{G} - \mathbf{g}d_E \mathbf{d}^T - \tilde{\mathbf{d}} d_E \mathbf{g}^T + \tilde{\mathbf{d}} g_E \mathbf{d}^T
$$

$$
\mathbf{h} = -\mathbf{1} + d_E \mathbf{d}
$$

$$
\alpha_E = -d_E \mathbf{d}^T \alpha
$$

We also define our reduced gradient:

$$
e^{(k)} = H \alpha^{(k)} - \mathbf{h}
$$

$$
= e^{(k)} - \tilde{\mathbf{d}}_E e_E - (\mathbf{g} - \tilde{\mathbf{d}} d_E g_E) d_E j
$$

Now, by the definition of our partition, constraints (*) in (3.5) are never active. Hence so long as we keep the problem appropriately partitioned during the solution process, we need only worry about the first two bounds.

Therefore, we need only consider the simplified reduced problem:

$$
\min_{\alpha} Q(\alpha) = \frac{1}{2} \alpha^T \hat{H} \alpha + \alpha^T \mathbf{h}
$$

such that

- $\alpha \geq 0$
- $\alpha \leq C1$

Following the active set method [4], we can further partition the problem thus:

$$
\begin{bmatrix}
H_{IA} & H_{Ib} & H_{Ic} \\
H_{a} & H_{b} & H_{c}
\end{bmatrix}
$$

$$
\alpha = \begin{bmatrix}
\mathbf{0} \\
\mathbf{h}_{a1} \\
\mathbf{h}_{a2} \\
\alpha_E
\end{bmatrix}, \quad \mathbf{h} = \begin{bmatrix}
\mathbf{e}_{a1} \\
\mathbf{e}_{a2} \\
\mathbf{e}_E
\end{bmatrix}
$$

Where $\alpha_B$ represents those variables not affected by the active set of inequalities. Neglecting for the moment our inactive constraints, we seek to solve the following problem:

$$
\min_{\alpha_B} \tilde{Q}(\alpha_B^{(k+1)}) = \frac{1}{2} \alpha_B^{(k+1)} H_B \alpha_B^{(k+1)} + \mathbf{r}^T \alpha_B^{(k+1)}
$$

where

$$
\mathbf{r} = CH \mathbf{1} + \mathbf{h}_B
$$

In order to find our step $\delta_B^{(k)}$:

$$
\alpha_B^{(k+1)} = \alpha_B^{(k)} + \delta_B^{(k)}
$$

Assuming that $H_B$ is non-singular, the step is:
\[ \delta_k^{(i+1)} = \delta_k^{(i)} = \mathbf{0} \]
\[ \delta_b^{(i)} = -H_b^{T} e_b^{(i)} \]
\[ \delta_t^{(i)} = -d_x^T \left( j^{(i)} + d_x^T \delta_b^{(i)} \right) \]
\[ \Delta e_A^{(i)} = H_A^{T} \delta_b^{(i)} \]
\[ \Delta e_{K}^{(i)} = H_K^{T} \delta_b^{(i)} \]
\[ \Delta e_{b}^{(i)} = -e_b^{(i)} \]
\[ \Delta e_{e}^{(i)} = g_b^T \delta_b^{(i)} + g_e \delta_{e}^{(i)} \]
\[ \Delta j^{(i)} = -j^{(i)} \]

(3.12)

Note that by storing \( H_2 \) in cholesky factorised form no direct matrix inversion is needed. Results in this paper are based on this method (for details on the necessary cholesky update algorithms see for example [7]). Note also that for our algorithm, we store and update all of the variables in (3.12), not just \( \alpha \).

For the singular case, assume that we can obtain a partial cholesky factorisation for \( H_2 \) thus:
\[ H_2 = \begin{bmatrix} V_U & 0 & I & 0 \\ c^T & 0 & 0 & 0 \\ 0^T & -d_x & 0 & 1 \end{bmatrix} H_L \]
\[ V_U \text{ is lower triangular} \]
\[ d_x \geq 0 \]

(3.13)

We used a heuristic described in [6] to obtain the following step for the singular case:
\[ \delta_b^{(i)} = q^{(i)} \begin{bmatrix} s^{(i)} \\ -1 \\ 0 \end{bmatrix} \]
\[ \Delta e_A^{(i)} = \begin{bmatrix} 0 \\ q^{(i)} d_x \\ \Delta e_{b}^{(i)} \end{bmatrix} , \Delta e_{e}^{(i)} = q^{(i)} H_L \begin{bmatrix} s^{(i)} \\ -1 \end{bmatrix} \]

(3.14)

Where:
\[ s^{(i)} = V_U^T c \]
\[ q^{(i)} = \begin{cases} C & \text{if } s^T e_{b}^{(i)} < e_{b}^{(i)} \\ -C & \text{if } s^T e_{b}^{(i)} > e_{b}^{(i)} \end{cases} \]
\[ 0 < V < 1 \]
\[ e_b^{(i)} = \begin{bmatrix} e_{b}^{(i)} \\ e_{e}^{(i)} \\ e_{b}^{(i)} \end{bmatrix} \]

This step is guaranteed to not be in a direction of ascent.

Now we must scale our step to ensure the feasibility of the next iterate. Defining:
\[ \beta^{(i)} = \min \left[ 1, \min_{j^{(i)} \in \mathcal{V}} \left( \frac{\alpha^{(i)}_j}{\delta^{(i)}_j} \right), \min_{j^{(i)} \in \mathcal{E}} \left( \frac{C - \alpha^{(i)}_j}{\delta^{(i)}_j} \right) \right] \]
\[ p = \arg \min_{j^{(i)} \in \mathcal{V}} \left( \frac{\alpha^{(i)}_j}{\delta^{(i)}_j} \right), \min_{j^{(i)} \in \mathcal{E}} \left( \frac{C - \alpha^{(i)}_j}{\delta^{(i)}_j} \right) \]

(3.16)

Our scaled step is:
\[ \alpha_{a1}^{(i+1)} = \alpha_{a1}^{(i)} \]
\[ \alpha_{a2}^{(i+1)} = \alpha_{a2}^{(i)} \]
\[ \alpha_{b}^{(i+1)} = \alpha_{b}^{(i)} + \beta^{(i)} \delta_{b}^{(i)} \]
\[ \alpha_{e}^{(i+1)} = \alpha_{e}^{(i)} + \beta^{(i)} \delta_{e}^{(i)} \]
\[ e_{a1}^{(i+1)} = e_{a1}^{(i)} + \beta^{(i)} \Delta e_{a1}^{(i)} \]
\[ e_{a2}^{(i+1)} = e_{a2}^{(i)} + \beta^{(i)} \Delta e_{a2}^{(i)} \]
\[ e_{b}^{(i+1)} = e_{b}^{(i)} + \beta^{(i)} \Delta e_{b}^{(i)} \]
\[ e_{e}^{(i+1)} = e_{e}^{(i)} + \beta^{(i)} \Delta e_{e}^{(i)} \]
\[ j^{(i+1)} = (1 - \beta^{(i)}) j^{(i)} \]

(3.17)

Now, if \( \beta < 1 \) then we must reformulate our active set by adding the relevant constraint (lower or upper bound) on \( \alpha_{p} \) to it (note that we cannot constrain \( \alpha_{s} \) to \( \alpha_{p} \), so we may need to re-partition our problem at this point). Once this is done, return to (3.12) to calculate our next step. Otherwise, we have found an optimal solution if our current active set is optimal.

Assuming we have found an optimal solution for our current active set, we must check the optimality of our active set. It can be shown [4] that our active set is optimal if the following constraints are satisfied:
\[ e_{a1}^{(i+1)} \geq 0 \]
\[ e_{a2}^{(i+1)} \leq 0 \]

(3.18)

If (3.18) is satisfied then we have found an optimal solution to our SVM training problem, and we can terminate the algorithm. Otherwise, we must alter our active set by removing one of our active constraints from it. To decide which, calculate:
\[ q = \arg \min_{j^{(i)} \in \mathcal{V}} \left( \frac{\alpha^{(i)}_j}{\delta^{(i)}_j} \right), \min_{j^{(i)} \in \mathcal{E}} \left( \frac{C - \alpha^{(i)}_j}{\delta^{(i)}_j} \right) \]

(3.19)

We then relax the constraint on \( \alpha_{q} \) and return to equation (3.12).
Note that for this algorithm:

- We will always find the optimal solution in a finite number of steps.
- The equality constraint in (3.1) will always be satisfied by the optimal solution.

4 Incremental Learning

We have chosen to use a fairly simple heuristic to investigate incremental learning. Suppose we have a trained SVM, and wish to add some additional training vectors:

\[
\Theta = \left\{ (\tilde{x}_1, \tilde{d}_1), (\tilde{x}_2, \tilde{d}_2), \ldots, (\tilde{x}_N, \tilde{d}_N) \right\} \quad (4.1)
\]

Define \( \hat{\alpha} \) such that:

\[
\hat{\alpha}_i = \begin{cases} 
0 & \text{if } \tilde{d}_i \cdot \tilde{x}_i \geq 1 \\
C & \text{if } \tilde{d}_i \cdot \tilde{x}_i \leq 1 
\end{cases} \quad (4.2)
\]

Now, we simply extend our previous SVM as follows:

\[
\begin{align*}
\alpha &\mapsto \begin{bmatrix} \hat{\alpha} \\ \alpha' \end{bmatrix}, \\
G &\mapsto \begin{bmatrix} G & G' \\ G' & g \\ g' & g' \\ g' & g' \\
g &\end{bmatrix} \\
G_{\alpha} &\mapsto G_x + G_{\alpha} + G_{\alpha} + g_e \\
G_{\alpha} &\mapsto g + g' + g' + g' + g' + g'
\end{align*} \quad (4.3)
\]

Consequently:

\[
\begin{align*}
\hat{e} &\mapsto \begin{bmatrix} \hat{e} + H_{\alpha} \hat{\alpha} \\ H_{\alpha} \hat{\alpha} + H_{\alpha} \hat{\alpha} + \hat{h} \end{bmatrix} \\
e_e &\mapsto \hat{e} + \hat{g}' \hat{\alpha} \\
j &\mapsto j + \hat{d}' \hat{\alpha}
\end{align*} \quad (4.4)
\]

We must re-partition for consistency at this point. We then re-enter the solver routine at equation (3.12).

5 Experimental Results

5.1 Methodology

Training data was obtained from images of fish under controlled conditions (fish tank). There were five species of fish in the tank, with four representatives of each species, making a total of twenty fish. Seventy images were taken of each fish. From each image we extracted 110 features, from which we selected the 10 most relevant (i.e. those that gave the best results).

- From each species, three fish (210 vectors) were used for training purposes (1050 vectors total).

- Images of the remaining fish were used for testing, giving 70 test images per species (350 vectors total).

The classifier was a 5-class voting SVM as described in section 2.1. C was chosen to be 10 experimentally. The following quadratic kernel was also chosen experimentally:

\[
K(x, y) = (1 + x^T y)^2 \quad (5.1)
\]

Training was carried out in a block-wise incremental fashion. M vectors (drawn equally from each species) were added to the training set per increment until all 1050 training vectors had been incorporated. Block sizes M used were: 5, 10, 15, 25, 30, 35, 50, 70, 75, 105, 150, 175, 210, 350, 525 and 1050. Note that the first block solve under this method is essentially a batch solve for M training vectors.

Computational cost was measured in flops (1 flop = 1 addition or multiplication or square root). Using this measure, we were able to compare the cost of, say, incrementally adding 5 additional training vectors to a set of 520, and batch solving from scratch.

All coding was done in C++. All testing was performed in dos (extended) on a Pentium 120.

5.2 Results Obtained

Figure 1 shows the computational cost for the batch for given training set size (all flop counts include all 10 binary SVMs which make up the 5-category SVM voting SVM). As can be seen, the computational cost quickly becomes excessive, as it increases in an almost quadratic fashion.

![Figure 1: Computational cost versus training set size (batch training).](image)
A sample of the incremental computational costs obtained for a range of increment block sizes are shown in figure 2. It is clear from the graphs that even in the most extreme case (where the “increment” is actually half of the total training set), it is invariably computationally cheaper to add the new training data incrementally rather than in the usual batch manner.

For example, suppose we have 1000 vectors in our current training set and wish to add some additional vectors. If we were to rely on batch training, this would cost approximately 650 million flops (about 45 minutes on the test computer) regardless of how many additional points we wish to add. However, as shown in table 1, the incremental method takes approximately 2% (if we wish to add 5 additional vectors) to 12% (if we add 50) of the time of the time it would take to do a batch solve – or about 1 to 5 minutes. This is a significant improvement.

For completeness, I have also included (figure 3) the accuracy results for the problem. As can be seen from the graph, results are generally good, with the exception of species 1.

### Table 1: Incremental cost when adding to a training set of 1000 vectors.

<table>
<thead>
<tr>
<th>Additional training points added</th>
<th>Cost (flops)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12×10^6</td>
</tr>
<tr>
<td>10</td>
<td>21×10^6</td>
</tr>
<tr>
<td>15</td>
<td>28×10^6</td>
</tr>
<tr>
<td>25</td>
<td>43×10^6</td>
</tr>
<tr>
<td>30</td>
<td>46×10^6</td>
</tr>
<tr>
<td>35</td>
<td>68×10^6</td>
</tr>
<tr>
<td>50</td>
<td>77×10^6</td>
</tr>
</tbody>
</table>

### 6 Summary and Conclusion

We have investigated the use of incremental active set methods in the training of support vector machines. We have presented a simple hot-start active set algorithm applied to the SVM problem, and demonstrated the computational advantages which incremental methods have over batch methods when used in appropriate circumstances. This has been demonstrated on a practical, non-trivial problem, and the improvement shown to be significant.

### 7 References


---

![Figure 2: Incremental update flop counts for different sizes of incremental block – (a) 5, (c) 10, (d) 50, (b) 525. Flop counts shown are for incremental part of solve only.](image1)

![Figure 3: SVM accuracy for different training set sizes.](image2)
Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:
Shilton, A.; Palaniswami, M.; Ralph, D.; Tsoi, A. C.

Title:
Incremental training of support vector machines

Date:
2001

Citation:

Publication Status:
Published

Persistent Link:
http://hdl.handle.net/11343/34060

File Description:
Incremental Training in Support Vector Machines

Terms and Conditions:
Terms and Conditions: Copyright in works deposited in Minerva Access is retained by the copyright owner. The work may not be altered without permission from the copyright owner. Readers may only download, print and save electronic copies of whole works for their own personal non-commercial use. Any use that exceeds these limits requires permission from the copyright owner. Attribution is essential when quoting or paraphrasing from these works.