

# Geometric SVM : A Fast and Intuitive SVM Algorithm

S.V.N. Vishwanathan, M. Narasimha Murty\*  
 Dept. of Comp. Sci. and Automation,  
 Indian Institute of Science,  
 Bangalore 560 012,  
 INDIA

## Abstract

We present a geometrically motivated algorithm for finding the Support Vectors of a given set of points. This algorithm is reminiscent of the DirectSVM algorithm, in the way it picks data points for inclusion in the Support Vector set, but it uses an optimization based approach to add them to the Support Vector set. This ensures that the algorithm scales to  $O(n^3)$  in the worst case and  $O(n|S|^2)$  in the average case where  $n$  is the total number of points in the data set and  $|S|$  is the number of Support Vectors. Further the memory requirements also scale as  $O(n^2)$  in the worst case and  $O(|S|^2)$  in the average case. The advantage of this algorithm is that it is more intuitive and performs extremely well when the number of Support Vectors is only a small fraction of the entire data set. It can also be used to calculate leave one out error based on the order in which data points were added to the Support Vector set. We also present results on real life data sets to validate our claims.

## 1. Introduction

Support Vector Machines (SVM) have recently gained prominence in the field of machine learning and pattern classification [8]. Classification is achieved by realizing a linear or non linear separation surface in the input space.

In Support Vector classification, the separating function can be expressed as a linear combination of the kernels associated with the Support Vectors as

$$f(x) = \sum_{x_j \in S} \alpha_j y_j K(x_j, x) + b$$

where  $x_i$  denotes the training patterns,  $y_j \in \{+1, -1\}$  denotes the corresponding class labels and  $S$  denotes the set of Support Vectors [8].

\*Corresponding Author

The Lagrangian ( $W$ ) of the primal problem can be expressed as

$$\min_{0 \leq \alpha_i \leq C} W = \frac{1}{2} \sum_{i,j} \alpha_i Q_{ij} \alpha_j - \sum_i \alpha_i + b \sum_i y_i \alpha_i \quad (1)$$

where  $\alpha_i$  are the corresponding coefficients.  $b$  is the offset.  $Q_{ij} = y_i y_j K(x_i, x_j)$  is a symmetric positive definite kernel matrix and  $C$  is the parameter used to penalize error points in the inseparable case [2]. The KKT conditions for the dual can be expressed as

$$g_i = \frac{\partial W}{\partial \alpha_i} = \sum_j Q_{ij} \alpha_j + y_i b - 1 = y_i f(x_i) - 1 \quad (2)$$

$$\frac{\partial W}{\partial b} = \sum_i y_i \alpha_i = 0 \quad (3)$$

This partitions the training set into  $S$  the Support Vector set ( $0 < \alpha_i < C, g_i = 0$ ), the error set ( $\alpha_i = C, g_i < 0$ ) and the well classified set ( $\alpha_i = 0, g_i > 0$ ) [2].

If the points in error are penalized quadratically with a penalty factor  $C$ , then, it has been shown that the problem reduces to that of a separable case with  $C = \infty$  [3]. The kernel function is modified as

$$K'(x_i, x_j) = K(x_i, x_j) + \frac{1}{C} \delta_{ij}$$

where  $\delta_{ij} = 1$  if  $i = j$  and  $\delta_{ij} = 0$  otherwise.

It can be seen that training the SVM involves solving a quadratic optimization problem which requires the use of optimization routines from numerical libraries. This step is computationally intensive, can be subject to stability problems and is non-trivial to implement [6]. Attractive iterative algorithms have been proposed to overcome this problem [6, 5].

The DirectSVM is an intuitively appealing, iterative algorithm, which builds the Support Vector set incrementally [7]. It starts off with the closest points of opposite classes

in the Support Vector set. During each iteration it finds the maximum violator and makes it a Support Vector. In case the dimension of the space is exceeded or all the data points are used up, without convergence. it reinitializes with the next closest pair of points from opposite classes [7].

The advantage of the DirectSVM algorithm is that it is geometrically motivated and intuitively appealing. But, the major problem with the algorithm is that it **remembers** each one of the previous updates of the support plane in order to calculate the next update. As a result the algorithm scales badly as the number of Support Vectors increases.

Recently some work has **also** been done on incremental SVM algorithms which can converge to exact solutions and also efficiently calculate leave one out errors [2]. The incremental algorithm is not easy to implement and requires some amount of book keeping.

In this paper we present a geometrically motivated algorithm which combines the intuitive nature of the DirectSVM algorithm with the efficiency of the Incremental algorithm. Section 2 talks about our algorithm and presents proofs. We talk about the results obtained on real life data sets in Section 3. Finally we conclude in Section 4 with comments. We also point out areas where applying our algorithm could be advantageous

## 2. Geometric SVM

We present a new algorithm that essentially uses the same idea as the DirectSVM for incrementally building the Support Vector set. But our algorithm uses a different technique to add new points to the Support Vector set. We use the quadratic penalty formulation to ensure that the data points are always linearly separable in the kernel space.

The outline of our algorithm is presented in Algorithm 1.

---

### Algorithm 1 Geometric SVM

---

Initialize with the next closest pair of points from opposite class

**if** all points are well classified **then**

**stop**

**end if**

Find max violator and make it a Support Vector

**if** number of Support Vectors > dimension of space **then**

**reinitialize**

**end if**

---

We now pose the subproblem of adding data points to the Support Vector set  $S$  as follows:

**Given a set  $S$  which contains only Support Vectors, add another Support Vector  $c$  to  $S$ .**

From Equations 2 and 3 we get the change in  $g_i$  as

$$\Delta g_i = Q_{ic} \Delta \alpha_c + \sum_{j \in S} Q_{ij} \Delta \alpha_j + y_i \Delta b \quad (4)$$

$$0 = y_c \Delta \alpha_c + \sum_{j \in S} y_j \Delta \alpha_j$$

where  $\Delta a$ , is the change in the value of  $a$ , and  $\Delta b$  is the change in the value of  $b$ . We start off with  $a_c = 0$  and so we can conclude that after the update  $\Delta \alpha_c = a_c$ .

Because all the vectors in  $S$  are Support Vectors we know from Equation 2 that  $g_i = 0 \quad \forall i$ . Since all the vectors in  $S$  continue to remain Support Vectors in  $S \cup \{c\}$  we require that  $\Delta g_i = 0$  for all vectors in  $S$ .

If we define [2]

$$P = \begin{bmatrix} 0 & y_1 & \dots & y_s \\ y_1 & Q_{11} & \dots & Q_{1s} \\ \vdots & \vdots & \ddots & \vdots \\ y_s & Q_{s1} & \dots & Q_{ss} \end{bmatrix}$$

then from Equations 4 and 5

$$P \begin{bmatrix} \Delta b \\ \Delta \alpha_1 \\ \vdots \\ \Delta \alpha_s \end{bmatrix} = - \begin{bmatrix} y_c \\ Q_{1c} \\ \vdots \\ Q_{sc} \end{bmatrix} \Delta \alpha_c$$

Thus we have

$$Ab = \beta \Delta \alpha_c \quad (6)$$

and

$$\Delta \alpha_j = \beta_j \Delta \alpha_c \quad (7)$$

If  $R = P^{-1}$  then we can write

$$\begin{bmatrix} \beta \\ \beta_1 \\ \vdots \\ \beta_s \end{bmatrix} = -R \begin{bmatrix} y_c \\ Q_{1c} \\ \vdots \\ Q_{sc} \end{bmatrix}$$

It is also clear that we want  $g_c = 0$  so that  $c$  can become a Support Vector Therefore

$$g_c = Q_{cc} \Delta \alpha_c + \sum_{j \in S} Q_{cj} (\alpha_j + \Delta \alpha_j) + y_c (b + \Delta b) - 1 = 0$$

From Equation 6 and 7 we get

$$(Q_{cc} + \sum_{j \in S} Q_{cj} \beta_j + y_c \beta) \Delta \alpha_c + \sum_{j \in S} Q_{cj} \alpha_j + y_c b - 1 = 0$$

and hence

$$\Delta \alpha_c = - \frac{\sum_{j \in S} Q_{cj} \alpha_j + y_c b - 1}{Q_{cc} + \sum_{j \in S} Q_{cj} \beta_j + y_c \beta} \quad (8)$$

If we define

$$\gamma_c = Q_{cc} + \sum_{j \in S} Q_{cj} \beta_j + y_c \beta$$

From [2] we know that we can expand R as

$$R = \begin{bmatrix} R & 0 \\ 0 & 0 \end{bmatrix} + \frac{1}{\gamma_c} \begin{bmatrix} \beta \\ \beta_1 \\ \vdots \\ \beta_s \\ 1 \end{bmatrix} \begin{bmatrix} \beta & \beta_1 & \dots & \beta_s & 1 & 1 \end{bmatrix} \quad (9)$$

Thus our new  $\mathbf{a}$  is

$$\alpha = \begin{bmatrix} \alpha + \Delta\alpha \\ \alpha_c \end{bmatrix}$$

and our new  $\mathbf{b}$  is  $\mathbf{b} + \mathbf{A}\mathbf{b}$ . Thus using this technique we can update the Support Vector set per iteration in an efficient way.

To calculate the maximum violator per iteration we need to do  $O(n|S|)$  work. To calculate  $\mathbf{A}\mathbf{a}$ , we require to do  $O(|S|)$  work per iteration. It is clear from Equation 9 that we require to do  $(|S| + 1) \times (|S| + 1)$  work to update R. Thus to add a Support Vector to  $S$  we need to do  $O(n|S|)$  work. In the average case we need not reinitialize. So the algorithm requires just  $|S|$  iterations to converge and hence the total work done is  $O(n|S|^2)$ . In the worst case the algorithm may need to reinitialize and that means that it has to consider every point as a Support Vector. So the worst case behavior is  $O(n^3)$ . We would like to point out that the algorithm did not require re-initialization for any of the real life data sets that we used in our experiments.

During an iteration the algorithm requires a kernel matrix corresponding to the current set of Support Vectors. That means that the memory requirements of the algorithm scale as  $O(|S|^2)$  in the average case and  $O(n^2)$  in the worst case. Typically the number of Support Vectors is a small fraction of the entire data set. As a result the memory requirements of the algorithm are also bounded. On the average, the algorithm never computes more than  $|S| \times |S|$  distinct kernels. The algorithm can be speeded up considerably by caching these kernel calculations.

### 3. Results

First of all we compare the performance of the DirectSVM and the Geometric SVM algorithm on the UCI Sonar data set [4]. This data set consists of 208 data points each of 60 dimensions. The training set contains 104 data points and the test set contains 104 data points. We use the polynomial kernel

$$K(x, y) = (x \cdot y)^d$$

and vary the degree of  $d$  [7]. We found that the accuracy of the Geometric SVM and DirectSVM on the sonar data set are exactly identical and are reproduced in table 1. It can also be seen that the Support Vectors produced by both the algorithms are identical.

$d$	1	2	3	4	5	6	7	8
Errors	33	15	12	17	17	12	13	13
Support Vectors	59	65	61	71	75	72	71	68

**Table 1. Performance of the Geometric SVM and DirectSVM on Sonar dataset**

Next we used the Spiral dataset proposed by Alexis Wieland of the MITRE Corporation and available from the CMU Artificial Intelligence repository. We use the Gaussian Kernel

$$K(x, y) = \exp(-0.5\|x - y\|^2/\sigma^2)$$

with  $\sigma^2 = 0.5$  [5]. We vary the value of  $C'$  and reproduce our results in table 2.

$C'$	SV	Kernel Eval.	Cache Hits( $10^6$ )
1.0	194	18915	1.21667
10.0	184	18860	1.20639
100.0	181	18824	1.19984
1000.0	177	18762	1.18878

**Table 2. Performance of the Geometric SVM on Spirals dataset**

We used the WPBC data set from the UCI Machine Learning repository [1]. This data set consists of 683 data points. each having a dimension of 9. Again we used the Gaussian kernel with  $\sigma^2 = 4.0$  [5]. We vary the value of  $C'$  and reproduce our results in table 3.

To compare the performance of the DirectSVM algorithm on this data set we selected a value of  $C' = 500$  and ran both Geometric SVM and DirectSVM on a unloaded single processor PIII 800 MHz machine with 128 MB RAM running Mandrake Linux 8.0. The DirectSVM algorithm accessed the cache  $9.011 \times 10^7$  times as opposed to  $2.191 \times 10^7$  by the Geometric SVM algorithm. Besides, the running time of DirectSVM was around 158 Seconds while that of the Geometric SVM was around 15 Seconds. The memory requirements of the DirectSVM algorithm were also higher than that of the Geometric SVM. This is because the DirectSVM has to store all the previous updates to compute the next update.

We used the Adult-1 data set again from the UCI Machine Learning repository [1]. This dataset consists of 1605

$C'$	SV	Kernel Eval.	Cache Hits( $10^7$ )
0.1	505	217655	4.411
1.0	352	178971	2.783
10.0	312	164580	2.306
100.0	307	162710	2.248
500.0	302	160815	2.191

**Table 3. Performance of the Geometric SVM on WPBC dataset**

data points, each having a dimension of 123. We used the Gaussian kernel with  $\sigma^2 = 10.0$ [6]. We vary the value of  $C'$  and reproduce our results in table 4.

$C'$	SV	Kernel Eval.	Cache Hits( $10^8$ )
0.1	1378	1262940	6.515
1.0	1086	1154910	5.204
10.0	912	1048340	4.143
100.0	779	947264	3.291
1000.0	720	896760	2.913

**Table 4. Performance of the Geometric SVM on Adult-1 dataset**

#### 4. Conclusion

We have presented a new algorithm that is efficient, intuitive and fast. While Keerthi et. al. [5] state that the total number of kernel evaluations is a measure of the efficiency of an algorithm we argue that the number of unique kernel calculations that the algorithm performs is an effective measure of its efficiency. If this number is small then we can utilize a caching scheme to achieve better performance. For example on a machine with 64MB of main memory, if we assume that storing a double requires 4 bytes, we can cache the kernel matrix corresponding to as many as 5600 Support Vectors. In this sense our algorithm is very efficient. Another advantage is that our algorithm does not suffer from numerical instabilities and round off errors that plague other numerical algorithms for the SVM problem.

It can be observed that the addition of a vector to the Support Vector set is entirely reversible. Using this property Poggio et. al. [2] have calculated the leave one out error. We propose to use similar techniques to calculate the leave one out error based on the order in which the data points were added to Support Vector set. We also propose to look at approximating a set of Support Vectors by a single vector to decrease the storage requirements.

#### References

- [1] C. Blake and C. Merz. UCI repository of machine learning databases, 1998.
- [2] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *NIPS Proceedings*. NIPS, MIT Press, November 2000.
- [3] T. Friess, N. Cristianini, and C. Campbell. The kernel adatron algorithm: a fast and simple learning procedure for support vector machine. In *Proceedings of 15th International Conference on Machine Learning*. Morgan Kaufman, 1998.
- [4] R. P. Gorman and T.J. Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75-89, 1988.
- [5] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Munhy. A fast iterative nearest point algorithm for support vector machine classifier design. *IEEE Transactions on Neural Networks*, 11(1):124-136, 2000.
- [6] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, December 1998.
- [7] D. Roobaen. DirectSVM: A fast and simple support vector machine perceptron. In *Proceedings of IEEE International Workshop on Neural Networks for Signal Processing*, Sydney, Australia, December 2000.
- [8] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 2<sup>nd</sup> edition, 2000.