SSVM : A Simple SVM Algorithm

S.V.N. Vishwanathan, M. Narasimha Murty
{vishy, mnm}@csa.iisc.ernet.in
Dept. of Comp. Sci. and Automation,
Indian Institute of Science,
Bangalore 560 012,
INDIA

Abstract - We present a fast iterative algorithm for identifying the Support Vectors of a given set of points. Our algorithm works by maintaining a candidate Support Vector set. It uses a greedy approach to pick points for inclusion in the candidate set. When the addition of a point to the candidate set is blocked because of other points already present in the set we use a backtracking approach to prune away such points. To speed up convergence we initialize our algorithm with the nearest pair of points from opposite classes. We then use an optimization based approach to increment or prune the candidate Support Vector set. The algorithm makes repeated passes over the data to satisfy the KKT constraints. The memory requirements of our algorithm scale as $O(|S|^2)$ in the average case, where $|S|$ is the size of the Support Vector set. We show that the algorithm is extremely competitive as compared to other conventional iterative algorithms like SMO and the NPA. We present results on a variety of real life datasets to validate our claims.

I. 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 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, $y_i \in \{+1, -1\}$ denotes the corresponding class labels and S denotes the set of Support Vectors [8].

The dual formulation yields

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

where α_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 [8]. The Karush-Kuhn-Tucker (KKT) conditions for the dual can be expressed as

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

and

$$\frac{\partial W}{\partial b} = \sum_j y_j \alpha_j = 0 \quad (3)$$

This partitions the training set into S the Support Vector set ($0 < \alpha_i < C$, $g_i = 0$), E the error set ($\alpha_i = C$, $g_i < 0$) and R 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. The advantage of this formulation is that the SVM problem reduces to that of a linearly separable case [4].

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 [5]. Attractive iterative algorithms like the Sequential Minimal Optimization (SMO), Nearest Point Algorithm (NPA) etc. have been proposed to overcome this problem [5], [4]. This paper makes another contribution in this direction.

A. DirectSVM and Geometric SVM

The DirectSVM is an intuitively appealing algorithm, which builds the Support Vector set incrementally [6]. Recently it has been proved that the closest pair of points of the opposite class are always Support Vectors [7]. DirectSVM starts off with this pair of points in the candidate Support Vector set. It has been conjectured that the maximum violator during each

Corresponding author - M. Narasimha Murty

iteration is a Support Vector [7]. The algorithm finds the maximum violator during each iteration and rotates the candidate Support Plane to make the maximum violator a Support Vector. In case the dimension of the space is exceeded or all the data points are used up, without convergence, the algorithm reinitializes with the next closest pair of points from opposite classes [6]. The advantage of the DirectSVM algorithm is that it is geometrically motivated and simple to understand.

The Geometric SVM proposed by us improves the scaling behavior of the DirectSVM by using an optimization based approach to add points to the candidate Support Vector set [9].

We observed that neither of the algorithms has a provision to backtrack, i.e. once they decide to include a point in the candidate Support Vector set they cannot discard it. During each step, both the algorithms spend their maximum effort in finding the maximum violator. If S is the current candidate Support Vector set, and n the size of the dataset, the algorithms spend $(n-s)|S|$ kernel evaluations to locate the maximum violator. Caching schemes have been proposed to increase the efficiency of this step [9].

Our observations led us to a greedy algorithm which picks the next immediately available violating point for inclusion in the candidate Support Vector set. Some stray non Support Vectors may be picked up for inclusion in the candidate set because of this greedy approach. Our algorithm backtracks later by pruning away such stray points from the candidate Support Vector Set. In order to satisfy the KKT conditions for all points it has to make repeated passes though the dataset.

B. Incremental Algorithm

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 algorithm works by adding a point at a time to the current dataset. It is shown that the addition of a point converges in a finite number of steps. The algorithm calculates the change in the KKT conditions of each one of the current data points in order to do book keeping. In case the size of the current data set is n and the number of Support Vectors is $|S|$, then, this algorithm has to perform $n|S|$ kernel operations for addition of the next point. The authors suggest a practical online variant where they introduce an δ margin and concentrate only those points which are within the δ margin of the boundary [2]. But, it is clear that the results may vary by varying the value of δ.

Instead of concentrating on the entire current dataset, which may be costly, our algorithm focuses its attention only on the current Support Vector set. It then picks data points greedily for inclusion in the current Support Vector set. While augmenting the current Support Vector set, some already existing Support Vectors may become well classified because of the addition of a new Support Vector. We prune away those points using the Decremental technique described in [2]. It may also happen that other points in the dataset may become violators or Support Vectors. The Incremental algorithm handles such points by doing costly book keeping in the current iteration itself and hence does not need to make repeated passes over the dataset [2]. On the other hand our algorithm does not handle such points in the current iteration, it makes repeated passes over the dataset to identify and satisfy such points.

C. Outline of Our Paper

In this paper we present a fast iterative algorithm which combines the the theoretically motivated foundation of the Incremental algorithm with the speed typical of iterative algorithms like NPA or the SMO. We borrow the initialization technique from the DirectSVM algorithm to speed up convergence of our algorithm. Section II talks about our algorithm and provides proofs. We also talk about the memory requirements of our algorithm. We present and discuss results obtained on real life datasets in Section III. We conclude in Section IV by pointing out advantages of our algorithm and indicate situations under which it can perform extremely well. We also provide pointers to our ongoing and future work.

II. Simple SVM

Our algorithm maintains a candidate Support Vector set. It initializes the set with the closest pair of points from opposite classes like the DirectSVM algorithm. As soon as the algorithm finds a violating point in the dataset it greedily adds it to the candidate set. It may so happen that addition of the violating point as a Support Vector may be prevented by other candidate Support Vectors already present in the set. We simply prune away all such points from the candidate set. To ensure that the KKT conditions are satisfied we make repeated passes through the dataset until no violators can be found. We use the quadratic penalty formulation to ensure linear separability of the data points in the kernel space.

A. Finding the Closest Pair of Points

First of all, we observe that, finding the closest pair of points in kernel space requires n^2 kernel computations where n represents the total number of data points. But, in case we use a *distance preserving* kernel like the exponential kernel the nearest neighbors in the feature space are the same as the nearest neighbors in the kernel space. Hence we need not perform any costly kernel evaluations for the initialization step.

B. Adding a Point to the Support Vector Set

Given a set S which contains only Support Vectors, we wish to add another Support Vector c to S. From Equations 2 and

3 we get the change in g_i due to addition of a new point c as

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

and

$$0 = y_c\Delta\alpha_c + \sum_{j\in S} y_j\Delta\alpha_j \tag{5}$$

where $\Delta\alpha_i$ is the change in the value of α_i and Δb is the change in the value of b. We start off with $\alpha_c = 0$ and update α_c as we go along.

Because all the vectors in S are Support Vectors we know from Equation 2 that $g_i = 0 \quad \forall i$. If none of the current Support Vectors blocks the addition of c to S then all the vectors in S continue to remain Support Vectors in $S \cup \{c\}$ and hence we require that $\Delta g_i = 0$ for all vectors in S. It is shown that [2]

$$\Delta b = \beta\Delta\alpha_c \tag{6}$$

and

$$\Delta\alpha_j = \beta_j\Delta\alpha_c \tag{7}$$

If we define

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

then

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

We want $g_c = 0$ so that c can become a Support Vector. Some algebra yields

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

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

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

Thus our new α is

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

and our new b is $b + \Delta b$.

C. Pruning

In the discussion above we tacitly assumed that $(\alpha + \Delta\alpha) > 0 \quad \forall p \in S$. But, this condition may be violated if any point in S blocks c. When we say that a point $p \in S$ is blocking the addition of c to S what we mean is that α_p of that point may become negative due to the addition of c to S. What it physically implies is that p is making a transition from S to the well classified set R. Because of the presence of such points we may not be able to update α_c by the amount specified by Equation 8. In such a case we can prune away p from S by using [2]

$$R_{ij} = R_{ij} - R_{ss}^{-1}R_{is}R_{sj} \quad \forall R_{ij} \in R$$

We remove the alpha entry corresponding to p from S so that all the other points in S continue to remain Support Vectors. We now try to add c to this reduced S. We keep pruning points from S till c can actually become a Support Vector.

D. Our Algorithm

Using the ideas we discussed above an iterative algorithm can be designed which scans through the dataset looking for violators. Using ideas presented in Section II-B the violator is made a Support Vector. Blocking points are identified and pruned away by using the ideas presented in Section II-C. The algorithm stops when all points are classified within an error bound i.e. $y_i f(x_i) > 1 - \epsilon \quad \forall i$. The outline of our algorithm is presented in Algorithm 1.

Algorithm 1 Simple SVM

```
candidateSV = { closest pair from opposite classes }
while there are violating points do
  Find a violator
  candidateSV = candidateSV ⋃ violator
  if any α_p < 0 due to addition of c to S then
    candidateSV = candidateSV \ p
    repeat till all such points are pruned
  end if
end while
```

E. Scaling

As is clear from the discussion in Section II-B and II-C, the algorithm needs to store the values of αs corresponding to the candidate Support Vector set (S). This occupies at most $O(|S|)$ space. Storing the value of b requires $O(1)$ space. It also needs to store the R matrix corresponding to the candidate Support Vector set. Since R is a $|S| \times |S|$ matrix its size scales up as $O(|S|^2)$. Thus we can conclude that the memory requirements of the algorithm scale up as $O(|S|^2)$ in the average case.

III. Results

Recently Keerthi et. al. have suggested using the number of kernel evaluations performed by a Support Vector algorithm as an effective measure of its speed [4]. Comparisons of the NPA algorithm with various other conventional iterative algorithms shows that NPA performs significantly faster [4]. We compared our algorithm with the NPA on 5 publicly available datasets. All experiments were run on a 800 MHz, Intel Pentium III machine with 128MB RAM running Linux Mandrake 8.0 unless mentioned otherwise. The code was written in C++ as well as in MATLAB[1]. We uniformly used a value of 0.001 for the error bound i.e. we stop the algorithm when $y_i f(x_i) > 0.999 \quad \forall i$. The NPA results are those reported in [4].

First 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 $\sigma^2 = 0.5$ [4]. We vary the value of C' and reproduce our results in table I.

	Simple SVM			NPA	
C'	Itr	SV	Kernel Eval	SV	Kernel Eval
0.03	192	194	0.038	195	0.104
0.1	192	194	0.038	195	0.121
0.2	192	194	0.038	195	0.134
0.3	192	194	0.038	195	0.143
0.6	192	194	0.038	195	0.162
1.0	192	194	0.038	195	0.188
2.0	199	194	0.038	195	0.238
3.0	201	194	0.038	194	0.276
5.0	204	188	0.040	188	0.340
10	204	184	0.039	185	0.452
50	240	182	0.044	181	1.094
100	240	180	0.045	181	1.300
500	250	178	0.054	179	1.540
1000	250	176	0.055	178	2.210

TABLE I
NPA VS SIMPLE SVM ON SPIRALS DATASET. KERNEL EVALUATIONS ARE $\times 10^6$

We used the WPBC dataset from the UCI Machine Learning repository [1]. This dataset consists of 683 data points, each having a dimension of 9. We used the Gaussian kernel with $\sigma^2 = 4.0$ [4]. We vary the value of C' and reproduce our results in table II.

[1]Working code and datasets used for the experiments are available at `http://www2.csa.iisc.ernet.in/~vishy`

	Simple SVM			NPA	
C'	Itr	SV	Kernel Eval	SV	Kernel Eval
0.03	665	652	0.490	652	1.21
0.1	599	505	0.518	505	1.04
0.2	587	473	0.493	473	0.99
0.3	549	434	0.461	427	1.31
0.6	476	360	0.411	360	0.74
1.0	454	352	0.414	352	0.85
2.0	420	330	0.384	330	0.59
3.0	399	327	0.372	327	0.71
5.0	400	320	0.434	319	0.73
10	383	311	0.406	311	0.85
50	372	304	0.456	303	0.78
100	364	302	0.451	303	0.78
500	356	296	0.443	298	0.69

TABLE II
NPA VS SIMPLE SVM ON WPBC DATASET. KERNEL EVALUATIONS ARE $\times 10^6$

We used the Adult-1 dataset from the UCI Machine Learning repository [1]. This is a sparse dataset which consists of 1605 data points, each having a dimension of 123. We used the Gaussian kernel with $\sigma^2 = 10.0$ [5]. We vary the value of C' and reproduce our results in table III.

The Adult-4 dataset is also from the UCI Machine Learning repository [1]. This is a sparse dataset and consists of 4781 data points, each having a dimension of 123. We used the Gaussian kernel with $\sigma^2 = 10.0$ [5]. We vary the value of C' and reproduce our results in table IV.

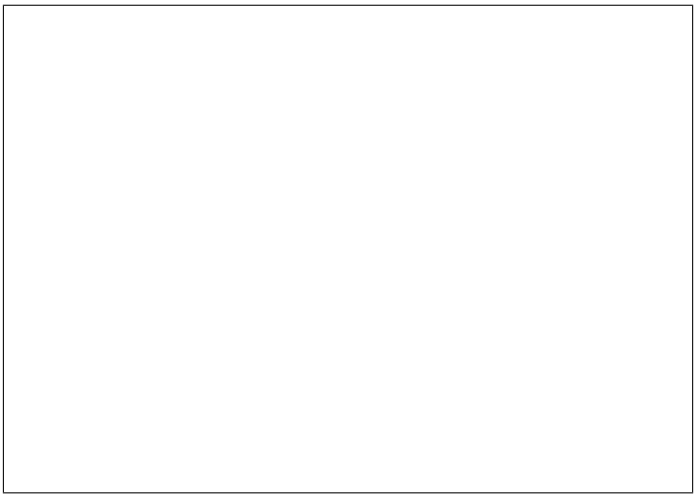

Fig. 1. Comparison of the Kernel Evaluations performed by Simple SVM and the NPA on the Adult-4 Dataset.

The Adult-7 dataset from the UCI Machine Learning repos-

	Simple SVM			NPA	
C'	Itr	SV	Kernel Eval	SV	Kernel Eval
0.03	1603	1602	2.58	1602	7.4
0.1	1535	1377	3.07	1377	8.1
0.2	1459	1246	3.03	1246	8.5
0.3	1394	1192	2.99	1191	7.9
0.6	1315	1119	2.83	1118	8.9
1.0	1077	1272	2.73	1077	9.6
2.0	1257	1032	2.77	1032	10.6
3.0	1252	996	2.90	996	11.5
5.0	1217	951	2.23	950	11.9
10	1191	898	2.88	898	14.1
50	1176	776	2.82	776	26.2
100	1188	735	3.18	734	33.7
500	1200	687	3.08	687	56.9
1000	1224	677	3.20	677	66.0

TABLE III
NPA VS SIMPLE SVM ON ADULT-1 DATASET. KERNEL EVALUATIONS ARE $\times 10^6$

	Simple SVM			NPA	
C'	Itr	SV	Kernel Eval	SV	Kernel Eval
0.03	4554	4112	2.65	4111	5.88
0.1	4051	3571	2.57	3569	6.22
0.2	3799	3384	2.45	3383	6.84
0.3	3700	3305	2.39	3305	7.04
0.6	3605	3201	2.40	3199	8.26
1.0	3552	3117	2.36	3116	8.96
2.0	3517	2999	2.44	2998	9.88
3.0	3466	2911	2.66	2915	9.87
5.0	3448	2882	2.63	2824	11.51
10	3486	2679	2.64	2680	15.06
50	3336	2313	2.64	2371	35.55
100	3420	2194	3.06	2242	46.08
500	3590	2032	2.74	2034	102.33
1000	3662	1967	3.02	1968	146.89

TABLE IV
NPA VS SIMPLE SVM ON ADULT-4 DATASET. KERNEL EVALUATIONS ARE $\times 10^7$

itory is a sparse data set of $16,100$ points, each having a dimension of 123 [1]. We used the Gaussian kernel with $\sigma^2 = 10.0$ [5]. The experiments were conducted on a 800 MHz PIII machine with 256 MB RAM, running Windows 2000. We vary the value of C' and reproduce our results in table V.

	Simple SVM			NPA	
C'	Itr	SV	Kernel Eval	SV	Kernel Eval
1.0	11284	10035	3.07	10181	11.31
10	10958	8833	3.64	9343	20.53
50	10540	7654	3.92	8393	51.05

TABLE V
NPA VS SIMPLE SVM ON ADULT-7 DATASET. KERNEL EVALUATIONS ARE $\times 10^8$

A. Discussion of the Results

As can be seen the Simple SVM algorithm outperforms the NPA on each one of the datasets tested. For example on the Spiral dataset the Simple SVM is an order of magnitude faster than the NPA. On the Adult-4 dataset with $C' = 1000$ the Simple SVM algorithm is nearly 50 times faster than the NPA. On the average the Simple SVM is seen to be around 3 to 5 times faster than the NPA.

The kernel computations performed by NPA, SMO and other iterative algorithms tend to vary with the value of C' and the total number of Support Vectors. On the other hand the kernel evaluations of the Simple SVM seem to be a function of the dataset only. They are fairly independent of the value of C' or the total number of Support Vectors. This is illustrated by the graph in Fig 1.

The difference between the number of iterations and the number of final Support Vectors indicates the number of times the algorithm had to backtrack. It can be seen that the Simple SVM algorithm needs to backtrack only around 20% of the times and hence the penalty incurred due to a greedy approach is not very significant. This validates the observation that a greedy approach taken by SMO like algorithms produces faster convergence.

The number of Support Vectors found by our algorithm and the NPA is different in the case of Adult-7 dataset. One possible explanation for this observation is that the stopping criterion employed by the two algorithms are different. Round off errors due to differences in the precisions used for the computations may further aggravate the problem.

IV. Conclusion and Future Work

We have presented a new algorithm that is efficient, intuitive and fast. We show that the algorithm significantly outper-

forms other iterative algorithms like the NPA in terms of the number of kernel computations. Because of the approach used to build the Support Vector, set our algorithm does not suffer from numerical instabilities and round off errors that plague other numerical algorithms for the SVM problem. Our algorithm currently does not use any kind of kernel cache to reuse kernel computations. We are currently investigating methods to speed up the algorithm using some efficient caching scheme.

We also observe that the memory utilization of the algorithm is governed by the R matrix which scales as $O(|S|^2)$. Hence, our algorithm does not scale well for those problems where the R matrix cannot be held in main memory. But, this is not a serious limitation, for example on a machine with 256 MB of main memory we can store the R matrix corresponding to as many as $10,000$ Support Vectors. We are investigating methods to store a compact representation of R in order to reduce this memory overhead.

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.

References

[1] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[2] Gert Cauwenberghs and Tomaso Poggio. Incremental and decremental support vector machine learning. In *Advances in Neural Information Processing Systems, (NIPS*2000)*, volume 13. NIPS, Cambridge MA: MIT Press, 2001.

[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] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. A fast iterative nearest point algorithm for support vector machine classifier design. *IEEE Transactions on Neural Networks*, 11(1):124–136, 2000.

[5] 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*. Cambridge MA: MIT Press, December 1998.

[6] Danny Roobaert. 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.

[7] Danny Roobaert. DirectSVM: A simple support vector machine perceptron. *Journal of VLSI Signal Processing Systems*, 2001. To appear.

[8] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 2^{nd} edition, 2000.

[9] S. V. N. Vishwanathan and M. Narasimha Murty. Geometric SVM: A fast and intuitive SVM algorithm. Technical Report IISC-CSA-2001-14, Dept. of CSA, Indian Institute of Science, Bangalore, India, November 2001. Submitted to ICPR 2002.