EXTENDING THE DECOMPOSITION ALGORITHM FOR SUPPORT VECTOR MACHINES TRAINING

N. M. Zaki*, S. Deris*, K. K. Chin**

**Faculty of Computer Science & Information System,*
University Technology Malaysia, Johor , Malaysia
nazar@siswa.utm.my

***Engineering Department, Cambridge University,*
Trumpington Street, Cambridge, UK.

ABSTRACT

The Support Vector Machine (SVM) is found to be a capable learning machine. It has the ability to handle difficult pattern recognition tasks such as speech recognition, and has demonstrated reasonable performance. The formulation in a SVM is elegant in that it is simplified to a convex Quadratic Programming (QP) problem. Theoretically the training is guaranteed to converge to a global optimal. The training of SVM is not as straightforward as it seems. Numerical problems will cause the training to give non-optimal decision boundaries. Using a conventional optimizer to train SVM is not the ideal solution. One can design a dedicated optimizer that will take full advantage of the specific nature of the QP problem in SVM training. The decomposition algorithm developed by Osuna et al. (1997a) reduces the training cost to an acceptable level. In this paper we have analyzed and developed an extension to Osuna's method in order to achieve better performance. The modified method can be used to solve the training of practical SVMs, in which the training might not otherwise converge.

Key words: Support vector machines, Decomposition, Pattern recognition, and Learning

1.0 INTRODUCTION

Solving a Quadratic Programming (QP) problem with a size of more than a few thousand LOC is very challenging in terms of memory requirements and computation costs. In practical pattern classification problems, the number of training data points can exceed 50,000. The training duration for large training data sets must be reduced before Support Vectors Machines (SVM) can be used to tackle practical problems. This problem is solved by taking advantage of the fact that the expected number of support vectors is a small fraction of the number of total training data points (Boser et al., 1992; Osuna et al., 1997b). However, this approach fails when the number of support vectors exceeds 3000. This can happen in the following cases:

- The amount of training data is very large, so the number of support vectors is a very small fraction of the total training data size; **it** is still too large for the **above algorithm** to handle.
- When data is highly un-separable, the generalization error will be high. Since the ratio between the number of support vectors and the total number of training data points is the upper bound of expected error (Vapnik, 1995), we can expect that the number of support vectors will increase for highly un-separable problems.

2.0 OVERVIEW OF OSUNA'S DECOMPOSITION ALGORITHM

In Osuna's decomposition algorithm, it is proposed that the original QP is replaced by a sequence of smaller sub-problems that does not require a small number of support vectors (Osuna et al., 1997a). A set of variables in Λ is partitioned into two sets Λ_B and Λ_N. Set B is called the *working set*, since variables in this set are allowed to change while the variables in the set *N* are fixed. These results are shown in the following sub-problems:

$$\min_{\Lambda_B} W(\Lambda_B) = -\Lambda_B^T 1 + \frac{1}{2}[\Lambda_B^T D_{BB}\Lambda_B + \Lambda_B^T D_{BN}\Lambda_N \\ + \Lambda_N^T D_{NB}\Lambda_B + \Lambda_N^T D_{NN}\Lambda_N] - \Lambda_N^T \qquad \textbf{(2.1)}$$

Subject to,

$$\Lambda_B^T y_B + \Lambda_N^T y_N = 0$$
$$\Lambda_B - C1 \leq 0$$
$$-\Lambda_B \leq 0 \qquad \textbf{(2.2)}$$

Since $k(x,y)$ is symmetric, $\Lambda_B^T D_{BN} \Lambda_N + \Lambda_N^T D_{NB} \Lambda_B$ can be replaced by $2\Lambda_B^T q_{BN}$,
where:

$$(q_{BN})_i = y_i \sum_{j \in N} \lambda_j y_j K(x_i, y_i) \quad i \in B \qquad \textbf{(2.3)}$$

With this replacement, and the fact that $\frac{1}{2}\Lambda_N^T D_{NN} \Lambda_N - \Lambda_N^T$ is constant, the size of the sub-problems is independent of the number of variables in set N regardless of whether the training data points corresponding to those variables are support vectors or not. Hence, the number of support vectors will not affect the performance of this algorithm.

The optimal solution for (2.1) can be achieved by introducing Lagrangian (L) and it must satisfy the following Karush-Kuhn-Tucker (KKT) condition:

$$f(x_j)y_j = \begin{cases} \geq L & \text{if} \quad \lambda_j = 0 \\ < L & \text{if} \quad \lambda_j = C \\ = L & \text{if} \quad 0 \leq \lambda_j \leq C \end{cases} \qquad \textbf{(2.4)}$$

where $f(x_i)$ is the decision function defined as

$$f(x_i) = \sum_{j=1}^{l} \lambda_j y_j K(x_i, x_j) + b, \quad i, j = 1,..,l \qquad \textbf{(2.5)}$$

The decomposition algorithm is then defined as follows:

1. Arbitrarily choose |B| points from the training data set.
2. Solve the sub-problems defined by the variables in set B.
3. Replace any $\lambda_i, i \in B$, with any $\lambda_j, j \in N$ that violate the condition in (2.4). It is also possible to sort λ_j according to the degree of violation, and select the highest degree of violation first.
4. Repeat step 2 until the values of λ satisfy the KKT condition in (2.4).

In this decomposition algorithm, it is assumed that the QP sub-problem at each iteration will converge. The consequence of this limitation is that the decomposition algorithm does not always converge to the global optimal solution in a finite number of iterations. The non-convergence of the QP algorithm manifests itself in two ways:

- The solver fails to change the values of the variables in set *B*. This happens when the solver finds that the cost function increases in all search directions before it reaches the saddle point. The solver will then assume wrongly that this is the saddle point on the cost surface. It then terminates without changing the values of variables in set *B*. Once the solver is stuck, it is very likely that all subsequent iterations will be stuck too. The training will now lock into an infinite loop. The decomposition algorithm will call the QP solver repeatedly until the classifier is converged.

- Due to the finite precision of computations, the optimal solution cannot be reached as the solution is hopping around the theoretical optimal solution at every iteration.

In the original implementation by Joachims (1997), the selection of λ_j from set *N* is randomized after every fixed number of iterations. This helps to reduce the possibility of non-converging SVM training. Despite the presence of this random element, non-convergence of SVM training still occurs, especially when the number of training data points exceeds 300. To improve the robustness of this training algorithm, more sophisticated mechanisms have to be implemented to avoid and/or detect these non-converging situations and stops the training.

3.0 EXPERIMENT

It is easy to detect when the training gets trapped, i.e. when the variables remain unchanged. The total change in all λ values can be computed as shown in equation (3.1) below. The training can then be stopped when it remains below an arbitrarily small threshold value for more than a chosen number of iterations. But this might stop the training prematurely, especially with a large number of training data and a large |B|. To avoid this, the following steps are taken whenever $d(\Lambda^t)$ is smaller than an arbitrarily small threshold:

1. Form a set of variables M that includes all λ in N that violate the condition in (2.4).
2. Select a set of λ from set M to form a set A, where $|A| << |B|$. Solve the sub-problems where A is the working set and all other variables are fixed.
3. Re-compute $d(\Lambda^t)$ and remove all λ in set A from set M.
4. If $d(\Lambda^t)$ is still below an arbitrarily small threshold and the set M is not empty, start from step 2 again.

$$d(\Lambda^t) = \sum_i^l | \lambda_i^t - \lambda_i^{t-1} | \qquad \textbf{(3.1)}$$

Since there is no simple solution to avoid the second non-converging situation, the next step is to define a robust mechanism to decide when to stop SVM training. The training must not be stopped prematurely and must stop as soon as possible when convergence to the global optimal is not possible. There is no way one can know whether a particular training will converge. Setting a maximum number of iterations is the simplest solution, but the following checks work well in the experiments of this project:

- $d(\Lambda^t)$ is less than an a priori threshold, and an a priori percent of the members in set B for time t is also in set B for time t-1 (i.e. if the value for λ does not change, and a very similar set of λ is selected for the next sub-problem).
- The current situation (i.e. number of mis-classified data, number of support vectors and largest classification error) of the training does not change more than an a priori percentage.

The training will be stopped if the above situation occurs, more than a pre-selected number of times. This is an ad hoc solution and there are many user-defined parameters, but it gives better control over SVM training and it works well in experiments performed.

3.1 Incremental Training of the Support Vector Machine

The computation cost for the training of SVMs can be further reduced by applying an incremental training scheme. The algorithm is defined as follows:

1. Divide the training data into segments. The segment size is decided heuristically. The SVM is initialized with an empty support vector set.
2. Select a segment and retrain the SVM using these data.

3. Discard all the data in the selected segment, except the support vectors.
4. Repeat step 2 and 3 until no segments are left.

This scheme is similar to the chucking method described by Boser et al. (1992) except that in the chucking method, any data point dropped from the chunk can be reselected. In this scheme, once a segment is selected it will not be selected again. Hence any discarded data will not be reselected, and there is a possibility that the final classifier might not classify these data correctly.

3.2 Complexity and Scalability of the Training Algorithm

The complexity of a QP solver is very much dependent on the training data. There is no known method to define the complexity analytically. All three solvers used in this project are iterative and the complexity of each iteration is $O(N^3)$ on average.

1. LOQO: At each iteration, the computation cost is dominated by solving a Symmetric Quasi-definite System of size 3N + 1 (Vanderbei, 1994a). In LOQO, a modified Cholesky factorisation (Vanderbei, 1996b) (an algorithm of $(O(N^3))$) is used to solve this system.
2. MATLAB QP: Computing the search direction at each iteration is the dominating cost factor. This operation involves multiplication and inversion of matrices of size N, which has complexity of $O(N^3)$.
3. DONLP2: At every iteration, an equality constraint QP sub-problem is solved. The size of this sub-problem is $N + E + A$ (Spellucci, 1996), where A is the number of inequality constraints in the working set, and E is the number of equality constraints for the QP problem. The worst case value for A is the number of inequality constraints for the QP problem (all the inequality constraints are selected), which is $2N$ in this case (the QP problem in SVM training has $2N$ bound constraints). Hence the worst case size of the QP sub-problems is 3N + 1 (the QP problem in SVM training has only one equality constraint). The equality constraint QP sub-problem is solved by solving a linear system of the same size, which is an operation with a complexity of $O(N^3)$.

4.0 RESULT AND DISCUSSION

The decomposition algorithm described in this paper is used to train the classifiers in the Gordon Peterson and Harold Barney vowels database (Peterson and Barney 1952). Peterson and Barney describe a detailed investigation of sustained American English vowels. The results are shown in Table 1. As expected, the number of iterations to achieve convergence increases as the sub-problem size decreases. But if the sub-problem size is small, the time taken for each iteration is very short. Hence, a small sub-problem size will speed up the training considerably. There is a lower bound on the sub-problem size to ensure fast convergence of the SVM training. When the sub-problem size is more than half of the total training data, the training duration will be longer than when no decomposition is applied (the training will take several iterations to converge and the duration for each iteration is close to the duration of one single training iteration without the decomposition algorithm).

It is not surprising that the classifiers trained by different sub-problem sizes have significantly different decision boundaries. Fig. 1 - 3 show the decision boundaries for different sub-problem sizes for class 2-to-rest. In general, as the sub-problem size approaches the total amount of training data, the boundaries also tend to be more similar compared to those of the classifiers trained without the decomposition algorithm.

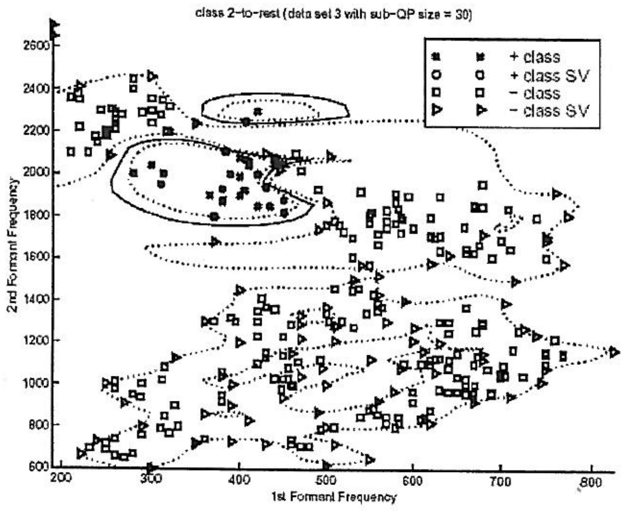

Fig.1: Decision boundary of data set 3, class 2 (sub-QPsize = 30)

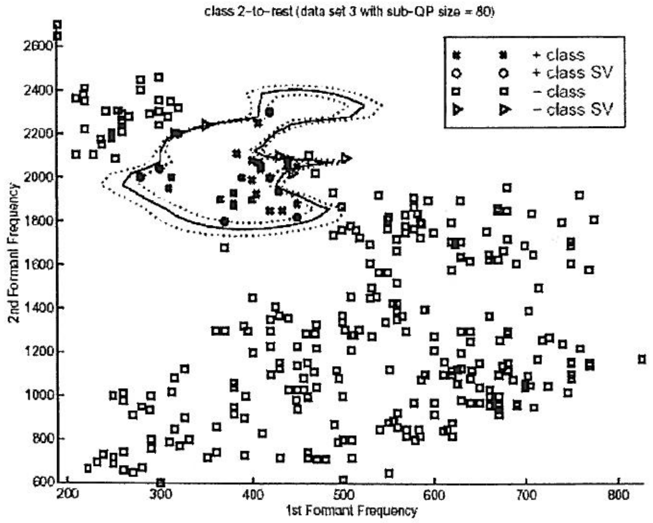

Fig. 2: Decision boundary of data set 3, class 2 (sub-QP size = 80)

Table 1: The results of SVM training with different sub-problem size

QP sub-problem size	Average training duration (per Sec)	Average iteration	Accuracy	Average number of SVs
30	62.16	62.8	58.75%	59
50	0097.24	43	51.88%	52.7
80	129.80	18.2	59.56%	42.89
160	831.96	12.4	56.56%	44.79
240	1959.50	7	56.88%	46.39
320	897.12	1	60.94%	48.6

The results in Table 1 show that the accuracy of the classifiers trained by the decomposition algorithm is lower compared to those trained without the decomposition algorithm. All the solutions that correspond to results in Table 1 are sub-optimal solutions; none of them can be proved to be the optimal solution. Table 2 shows the accuracy of classifiers trained on 5 sets of random data (each with 160 data points) with different sub-problem sizes. From these results, it is clear that none of the sub-problem sizes consistently give better results than the rest.

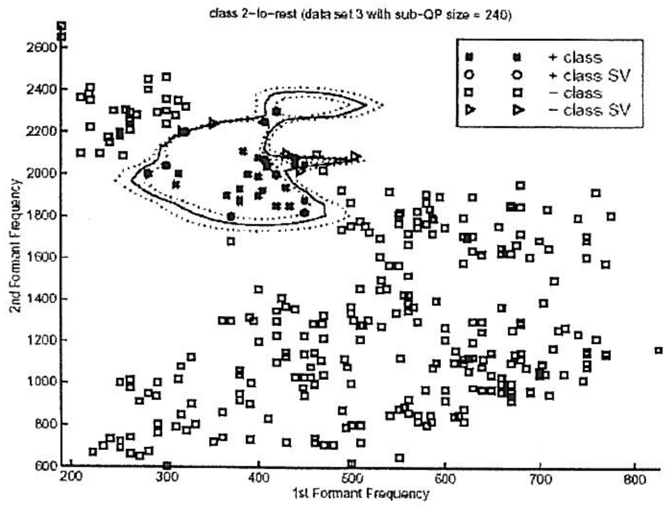

Fig. 3: Decision boundary of data set 3, class 2 (sub-QP size = 240)

Table 2: The results of SVM training with different sub-problem size on 5 random sets of 160 points training data

QP sub-problem size	Set 1	Set 2	Set 3	Set 4	Set 5
30	69.38%	62.50%	65.00%	74.37%	55.00%
50	69.38%	63.13%	65.00%	74.37%	55.00%
80	69.38%	63.13%	65.62%	74.37%	55.00%
160	70.63%	61.87%	65.62%	74.37%	55.00%

The number of iterations required to solve a QP problem is not known for any of the solvers above, it very much depends on the data (i.e. the surface of the cost function and the constraints). From experimental observations, the number of iterations is of O(N)(LOQO definitely converges with much less iterations compared to DONLP2 and MATLAB QP). Hence, the complexity of solving the QP problem in the SVM training can be approximated as O(N^4). When the number of training data points exceeds a few hundreds, the computation cost for the SVM training will be unacceptable. The training duration for larger training data sets can be estimated as in Table 3.

Table 3: The estimated training duration of SVM with large training data set

QP sub-problem	Estimated Training Duration
640	4 hour
1280	64 hour
2560	1024 hour
5120	16384 hour
10240	262144 hour

It is clear that without a decomposition algorithm, a SVM can never be used as a practical classifier (the training cost is too high to be practical). The complexity of the decomposition algorithm in Section 3.2 is defined as follows:

- N is the total number of training data points
- D is the number of dimensions for each data point
- Q is the size of the working set (i.e. |B|)
- S is the number of support vectors
- qp (Q) is the complexity of the QP sub-problem (qp(Q)) is estimated to be Q^4.
- From the above definition, the complexity of each iteration of the decomposition algorithm is $O(QND + N^2D + Q^4)$.

The number of iterations is also assumed to be linearly related to *N* and inversely to Q. As Q increases, the number of iterations required to solve the QP problem will decrease (see Table 1). Hence, the number of iterations is of the order of O(N/Q). The approximate complexity of the SVM training with the decomposition algorithm is $O(N^2D + N^3D/Q + NQ^3)$. From these analyses, it is clear that the complexity of the algorithm grows linearly with respect to the dimension of the training data. The computation cost can be significant if N is very large.

The choice of Q is very important in order to obtain fast convergence. Results in Table 1 suggest that a smaller Q will converge faster. From the complexity analysis, it is clear that when N and D are large, the cost for each iteration will be high, and a smaller Q requires more iteration, and therefore, converges more slowly. Fig. 4 and 5 show that, for every N and D value, there is an optimal Q setting.

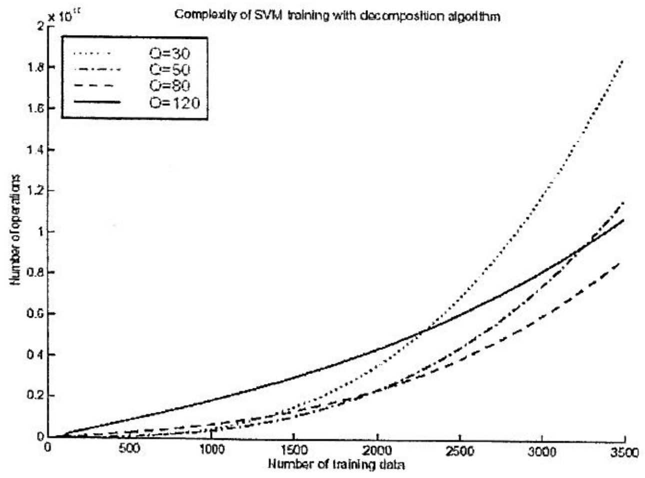

Fig. 4: The complexity of the SVM algorithm with different Q settings

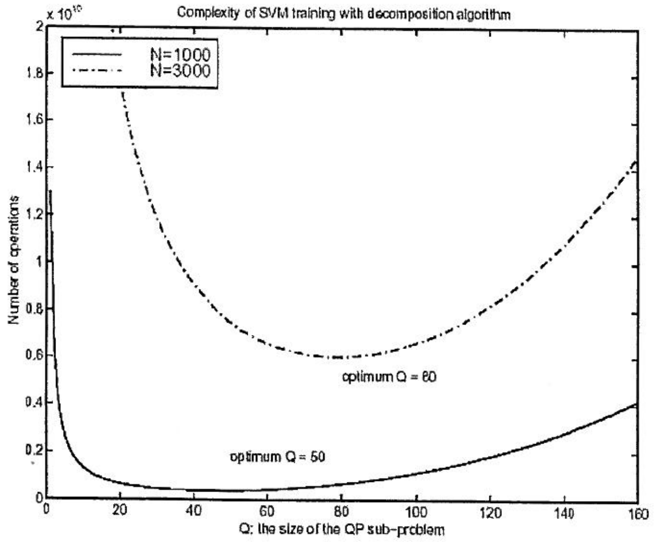

Fig. 5: The minimum Q value training with the decomposition for different training data sizes

Fig. 6 shows the comparison between the complexity of the standard SVM training, and that of the SVM training with the decomposition algorithm. The decomposition reduces the complexity significantly, and hence, allows SVM to be trained with a reasonable computation cost.

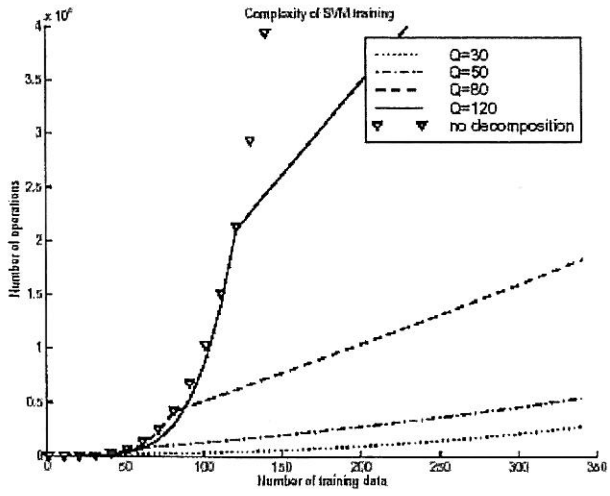

Fig. 6: Complexity of SVM training algorithms with and without decomposition algorithm

5.0 CONCLUSION AND FURTHER RESEARCH WORK

Support Vector Machines has the ability to handle difficult speech recognition tasks, and they give reasonable performance. The formulation of a SVM is elegant in that it is simplified to a convex QP problem. Theoretically, the training is guaranteed to converge to a global optimal. This ability to select the training data that defines the classification boundaries could have many applications other than in pattern classification. The training of a SVM is not as straightforward as it seems. Numerical problems will cause the training to give non-optimal decision boundaries. Using a conventional optimizer to train a SVM is not the ideal solution. One can design a dedicated optimizer that will take full advantage of the specific nature of the QP problem in SVM training. The decomposition algorithm developed by Osuna et al. (1997a) reduces the training cost to an acceptable level. In this paper we have analyzed and developed an extension to Osuna's method in order to achieve better performance. The method can be used to solve the training of practical SVMs, in which the training might not otherwise converge. For further improvement in the performance of the decomposition algorithm, one can consider incorporating information from the optimizer, i.e., use the knowledge about the current situation of the global QP problem to select the member in the working set. This could help to speed up convergence.

REFERENCES

Boser, B. E., Guyon, I. M., and Vapnik, V. (1992). A training algorithm for optimum margin classifiers. In *Fifth Annual Workshop on Computational Learning Theory*. Pittsburgh, ACM.

Peterson, G. and Barney, H. (1952). Control methods used in a study of vowels. *Journal of the Acoustical Society of America*, vol. 24, pp. 175-184.

Joachims, T. (1997). *Svm light: Implementation of the decomposition training algorithm*. Bell Lab. Lucent Technologies.

Osuna, E., Freund, R., and Girosi, F. (1997a). An improved training algorithm for support vector machines. *In Proc. of IEEE NNSP'97*. Amelia Island: FL, 24-26.

Osuna, E., Freund, R., and Girosi, F. (1997b). Support vector machines, Training and applications. Technical report memo 1602, MIT AI Lab. CBCL.

Spellucci, P. (1996). A sqp method for general nonlinear programs using only equality constrained sub-problems. Technical report, Dept. of Mathematics, Technical University at Darmstadt.

Vanderbei, R. J. (1994a). Loqo: An interior point code for quadratic programming, Technical report, Program in Statistics & Operations Research. Princeton University.

Vanderbei, R. J. (1994b). Symmetric quasi-definite matrices. *SIAM J. Optimization,* 5 (1)100-113.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. New York: Springer Verlag