INCAS: An Incremental Active Set Method for SVM

Shai Fine
IBM Research Labs in Haifa, Israel
shai@il.ibm.com

Katya Scheinberg
IBM T. J. Watson Research Center
katyas@us.ibm.com

November 15, 2002

Abstract

We propose an active set algorithm to solve the convex quadratic programming (QP) problem which is the core of the support vector machine (SVM) training. The underlying method is not new and have been used by the optimization community to solve moderately sized convex QPs. However, its application to large scale SVM problems is new. The special structure of SVM problems makes it possible to adapt this algorithm to work efficiently for large problems. We also incorporate several "tricks" (such as shrinking) used by other advanced SVM implementations. We present computational results comparing our method with Joachims' SVM^{light} [11]. The results show that our method has overall similar or better behavior on many SVM problems. It has particularly strong advantage in problems that either have bad separability in the feature space or have a lot of noise; i.e., problems where the solution contains a lot of vectors violating the margin condition (outliers). Another advantage of the method is that it does not require the user to "guess" the chunk size and that it typically produces more accurate solution in similar time. The algorithm also naturally extends to the incremental mode.

1 Introduction

In this paper we introduce an active set method to solve the following convex quadratic programming (QP) optimization problem which is defined by *1-Norm Soft Margin* SVM problem.

$$
\begin{array}{lll}
 & \max & -\frac{1}{2}\alpha^{T} Q\alpha - c\sum_{i=1}^{n}\xi_i \\
(P) & \text{s.t.} & -Q\alpha + by + s - \xi = -e, \\
 & & 0 \le \alpha \le c,\ s \ge 0,\ \xi \ge 0,
\end{array}
$$

where $\alpha \in \mathbf{R}^n$ is the vector of dual variables, b is the bias (scalar) and s and ξ are the n-dimensional vectors of slack and surplus variables, respectively. y is a vector of labels, ± 1. Q is the label encoded kernel matrix, i.e. $Q_{ij} = y_i y_j K(x_i, x_j)$, e is the vector of all 1's of length n and c is the penalty associated with errors. The dual of this problem is

$$
\begin{array}{lll}
 & \min & \frac{1}{2}\alpha^{T} Q\alpha - e^{T}\alpha \\
(D) & \text{s.t.} & y^{T}\alpha = 0, \\
 & & 0 \le \alpha \le c.
\end{array}
$$

General convex QPs are typically solved by one of the two approaches: *interior point method* approach or *active set method* approach. If the Hessian of an objective function (matrix Q in case of SVM) and/or the constraint matrix of the QP problem is large and sparse than an interior point method is usually the method of choice. If the problem is of moderate size but the matrices are dense, then active set method is preferable. In SVM problems the Q matrix is typically dense. Thus, large SVM problems present a challenge for both approaches. Recently it was shown [6], [3] that for some classes of SVMs, for which Q is dense but low-rank, one can adapt an interior point method to work very efficiently. However, if the rank of Q is high, an active set approach seems to remain the only main alternative.

One of the most "traditional" active set methods in the optimization literature is the *Simplex* method for linear programming (LP) problems. The Simplex method is known to have very good practical performance. The QP analogues, though not as extensively tested in practice, are also considered to

be very efficient. There are a few methods based on the Simplex method idea for solving QP problems, see e.g., [7], [8], [9]. Many of them are theoretically equivalent, meaning that they produce the same sequence of iterations, but they have different numerical properties (such as per-iteration complexity and numerical stability). In this paper we derive an implementation targeted to SVM problems based on the framework described in [7], [8] and [12].

The main idea of this method in the context of SVM is to fix, at each iteration, all variables in the current dual active set[1] at their current values(0 or c), and then to solve the reduced dual problem. After obtaining a solution - decide whether it is optimal for the overall dual problem (same as being feasible for the overall primal problem), or if any of the dual variables should be released from the active set.

When applied to SVM, this approach poses the following problem: if the complement of the dual active set (the set of "free" variables) has large cardinality, then solving the restricted subproblems may be too expensive, since Q is completely dense. Also determining the next variable to leave the active set may be expensive for the same reason. Therefore, updating all "free" variables at once was considered impractical.

The most common approach to large SVM problems is to use a restricted active set method, such as Chunking [2] or Decomposition [13], [11] where at each iteration only a small number of variables are allowed to be varied. The size of such "chunk" is determined heuristically. There are a few skillfully implemented SVM solvers based on this type of restricted active set methods, see e.g., [11], [14]. The main disadvantage of these methods is that their performance is sensitive to the changes in chunk size and there is no good way of predicting a good choice for the size of the chunks for a particular problem[2] . Moreover, these methods tend to have slow convergence when getting closer to the optimal solution.

A full active set method, such as the one presented in this paper, avoids these disadvantages. The method itself is not new, see e.g. [12]. Our contribution is to adapt it to the SVM framework and provide an efficient implementation. We will refer to our implementation of the method as INCAS (INCremental Active Set method). We use the term "incremental" to point

[1]The dual active set is the set of dual variables α whose values are at their bound.

[2]See section 12.1.1 in [14] for a similar discussion which motivated Platt's SMO. Essentially SMO is an active set method in which the chunk size is fixed to be the smallest possible, i.e. 2.

out that this method can be naturally used in an incremental setting, and that at each iteration the active set is updated by increments of only one point.

First we notice that the support vectors that violate the margin constraint (i.e., the corresponding ξ surplus variable is positive) correspond to variables α which are at their upper bound and therefore are in the dual active set. The complement of the dual active set contains variables α that are strictly between the upper and lower bounds. Such variables correspond only to the support vectors that are exactly on the margin (i.e., both corresponding slack and surplus variables are zero). The current number of such support vectors, n_s, is the size of the reduced QP. Solving such QP requires at least $O(n_s^3)$ operations, which might be prohibitively expensive if repeated over and over again and if n_s is relatively large. If, however, the active set is incremented only by one variable at a time (either one variable leaving, or one entering the active set) then one can store and update a factorization of the reduced matrix Q. Each update takes $O(n_s^2)$ operations and so does solving a system of equations with the reduced matrix Q.

To ensure "one-variable-at-a-time" increment strategy we cannot require solving the subproblems (reduced QPs) for optimality at each iteration. Instead we take a few steps towards optimality. At each such step we either find an optimal solution or encounter a bound on one of the "free" variables. In the latter case this variable is included into the active set and the process repeats. This process does not always produce an optimal solution to the subproblem, but usually produces a good approximation of it. Typically this does not affect the overall number of iterations significantly, whereas the reduction of the per-iteration cost is significant.

If the search for the optimum of the subproblem is terminated then our method determines whether the primal feasibility was achieved and if not, which dual variable should leave the active set. To do that we need to compute a product between a part of Q that corresponds to the variables at their upper bounds and the unit vector of an appropriate length. This can be very expensive to compute at each iteration, instead one should rather store and update the result of this multiplication. Another advantage of using "one-variable-at-a-time" increments is in potentially reducing the cost of such updates.

The multiple updates to the active set, which are used in "chunking" and "decomposition" methods could still have an advantage if the overall number

of iterations were significantly smaller than in the case of single updates. But as our computational results indicate this is not the case. We offer some intuition to support this claim. Assume that your data contains 10 identical data points which at the current iteration are the most violated examples and we would like to introduce them into the next "chunk". Introducing all the 10 at once implies 10 times more work than introducing just one. Yet since they are identical, then introducing just one produces the same result as introducing all ten. Since the training data is often somewhat repetitive (there may not be *identical* points, but rather very similar points, e.g. in clustered data sets) this example is not too far fetched.

As the computational results show, our method has the main advantage over SVM^{light} on problems where the number of variables at the upper bound is large (but not necessarily excessive, e.g., $\sim$ 1000 out of 20000 vectors). This is very common in problems where the data either has a lot of noise, or cannot be separated well in the feature space. Our algorithm is currently slower on problems where the number of outliers is not too large but the number of support vectors is large. This often means that the chosen kernel suffers from overfitting the data, so the problem is badly posed in some sense. On the other hand the performance on such problems may still be address by improving the implementation.

Finally we would like to point out that many of the ideas presented here are independently developed but are similar to the ones described in Cauwenberghs and Poggio at [4]. However, our method is somewhat different, it is applied not only in the incremental mode and it is expected to be more efficient[3] (the real comparison is hard to produce, since their code is in Matlab and only works in incremental mode which is inherently slower).

In the next section we will briefly introduce the method and discuss the key implementation points. In Section 3 we will present the computational results. We'll wrap it up in Section 4 with some concluding remarks.

[3]it is analogous to dual Simplex method, whereas their method is analogous to the inferior primal-dual Simplex method

2 Active set method for SVM

Any optimal solution to problems (P) or (D) must satisfy the Karush-Kuhn-Tucker (KKT) necessary and sufficient optimality conditions:

$$\begin{array}{ll} \mathbf{1} & \alpha_i s_i = 0, \quad i = 1, \ldots, n \\ \mathbf{2} & (c - \alpha_i)\xi_i = 0, \quad i = 1, \ldots, n \\ \mathbf{3} & y^T \alpha = 0, \\ \mathbf{4} & -Q\alpha + by + s - \xi = -e, \\ \mathbf{5} & 0 \leq \alpha \leq c, \\ \mathbf{6} & s \geq 0,\ \xi \geq 0. \end{array} \tag{1}$$

Let us introduce some notation. A primal-dual solution (α, b, s, ξ) is called *dual basic feasible* if it satisfies condition (**1**)-(**5**) of the KKT system, but may violate condition (**6**). For a given dual basic feasible solution, (α, b, s, ξ), we partition the index set $I = \{1, \ldots, n\}$ into three sets I_0, I_c and I_s in the following way: $\forall i \in I_0\ s_i \geq 0$ and $\alpha_i = 0$, $\forall i \in I_c\ \xi_i \geq 0$ and $\alpha_i = c$ and $\forall i \in I_s\ s_i = \xi_i = 0$ and $0 < \alpha_i < c$. It is easy to see that $I_0 \cup I_c \cup I_s = I$ and $I_0 \cap I_c = I_c \cap I_s = I_0 \cap I_s = \emptyset$. We will refer to I_s as the *primal active set* and to $I_0 \cup I_c$ as the *dual active set*.

Based on the partition (I_0, I_c, I_s) we define Q_{ss} (Q_{cs} Q_{sc} Q_{cc}, Q_{0s}, Q_{00}) as the submatrix of Q whose columns are the columns of Q indexed by the set I_s (I_c, I_s, I_c, I_0, I_0) and whose rows are the rows of Q indexed by I_s (I_s, I_c, I_c, I_s, I_0). We also define y_s (y_c, y_0) and α_s (α_c, α_0) and the subvectors of y and α whose entries are indexed by I_s (I_c, I_0). By e_s (e_c) we denote a vector of all ones of the appropriate size.

To initiate the algorithm we assume that we have a dual basic feasible solution α^0, b, s^0, ξ^0 and the corresponding partition $({I_0}^0, {I_c}^0, {I_s}^0)$. For example setting $\alpha^0 = 0$ and $I_0 = \{1, \ldots, n\}$ produces a starting point for the algorithm.

We know that $\forall i \in I_0\ \alpha_i = 0$ and $\forall i \in I_c\ \alpha_i = c$. Then if we fix the variables in the dual active set then our dual problem reduces to

$$\begin{array}{ll} \min_{\alpha_s} & \frac{1}{2}{\alpha_s}^T Q_{ss}\alpha_s + c{e}^T Q_{cs}\alpha_s - {e}^T\alpha_s \\ \text{s.t.} & {y_s}^T\alpha_s = -{y_c}^T\alpha_c, \\ & 0 \leq \alpha_s \leq c. \end{array}$$

The outline of the algorithm is the following:

Step 1

(i) Solve

$$\begin{aligned} \min_{\alpha_s} \quad & \frac{1}{2}\alpha_s^T Q_{ss}\alpha_s + ce^T Q_{cs}\alpha_s - e^T\alpha_s \\ \text{s.t.} \quad & y_s^T\alpha_s = -y_c^T\alpha_c \end{aligned}$$

(ii) From the current iterate make a step towards the solution until for some $i \in I_s$ $(\alpha_s)_i = 0$ or $(\alpha_s)_j = c$ or until solution is reached.

(iii) If for some $i \in I_s$, $(\alpha_s)_i = 0$
Then update $I_s = I_s \backslash \{i\}$, $I_0 = I_0 \cup \{i\}$, and go to step (i).

(iv) If for some $i \in I_s$, $(\alpha_s)_i = c$
then update $I_s = I_s \backslash \{i\}$, $I_c = I_c \cup \{i\}$, and go to step (i).

(v) If the optimum is reached in step (ii), proceed to **Step 2**.

Step 2

(i) Compute s_0

$$s_0 = -Q_{0s}\alpha_s - y_0\beta + 1 - cQ_{0c}e$$

and ξ_c

$$\xi_c = Q_{cs}\alpha_s + y_c\beta - 1 + cQ_{cc}e$$

(ii) Find $i_0 = \text{argmin}_i\{s_i : i \in I_0\}$.
Find $i_c = \text{argmin}_i\{\xi_i : i \in I_c\}$.

(iii) If $s_{i_0} \geq 0$ and $\xi_{i_c} \geq 0$ then the current solution is optimal, **Exit**.
If $s_{i_0} \leq \xi_{i_c}$, then $I_s = I_s \cup \{i_0\}$ and $I_0 = I_0 \backslash \{i_0\}$.
Else, $I_s = I_s \cup \{i_c\}$ and $I_c = I_c \backslash \{i_c\}$.
Go to **Step 1**.

The most time consuming steps of the algorithm are **Step 1**(i) and **Step 2**(i). To solve the reduced QP in **Step 1**(i) one needs to solve one system of

linear equations (since the problem does not have inequality constraints, the problem has a closed form solution). The matrix of this system is

$$\begin{bmatrix} Q_{ss} & y \\ y^T & 0 \end{bmatrix}.$$

To invert this matrix one needs to have an inverse or factorization of Q_{ss}. Since Q_{ss} is positive semidefinite, using Cholesky factorization is more efficient and more numerically stable than using the inverse. If n_s is the cardinality of I_s then computing Cholesky factorization of Q_{ss} from scratch requires $O(n_s^3)$ operations. Since at each step I_s can be either reduced or increased by only one element, then the matrix Q_{ss} can be updated by adding or dropping one row and one column. The Cholesky factors of the new Q_{ss} can be obtained from the old Cholesky factors by $O(n_s^2)$ operations. Details of this update will be given in the full version of the paper and also can be found in [10].

If n_s is very large and is comparable to n then even storing and updating the Cholesky factors of Q_{ss} become too expensive. Our method is not practical on such problems. However, it is questionable whether such problems should ever be solved, since the resulting classifiers is most likely overfitting the data and its generalization properties are expected to be very poor [4] .

If k is the number of operations required to compute one kernel value, then **Step 2**(i) requires $O(n_s(n-n_s)k)$ operations to compute $Q_{0s}\alpha_s$ and $Q_{cs}\alpha_s$. To reduce the cost we store as many computed kernel values in Q_{0s} and Q_{cs} as possible. Currently INCAS is implemented in Fortran and proper caching is not done. We roughly mimic caching by providing a few levels of storage. In our computational examples, however, the memory was always sufficient to store all values of current Q_{0s} and Q_{cs}. Hence the work is reduced to $O(n_s(n-n_s))$. Next, we use "shrinking", or selective pricing, as it is called in the optimization literature, to reduce the workload further. The idea behind shrinking is that if some elements of s_0 or ξ_c remain positive for many consecutive iterations, then it is likely that they will remain positive through the rest of the iterations. Hence we can skip computing their values at **Step 2**(i). We do however, compute their value to verify the optimality when the algorithm terminates and if not all of these elements are positive we

[4] See for example [5], Theorem 4.25, for the generalization power of compression schemes, and the discussion right after and in chapter 6.

proceed with more iterations. Shrinking typically has very significant effect on overall computational time.

One can also argue that computing all values in s_0 and ξ_c is not necessary. As soon as a negative value is encountered, one can drop the corresponding element from the active set and proceed with the algorithm. However, in our experiments we discovered that the overall reduction in the number of iterations achieved by choosing the most violated slack or surplus variable is well worth the extra per-iteration effort.

Finally, instead of computing vectors $Q_{0c}e$ and $Q_{cc}e$ at each iteration we store and update these vectors each time we update I_c. The storage and update cost is $O(n)$ and, hence is negligible.

It is easy to see that this algorithm naturally adapts to the incremental setting. As more data becomes available one initially assigns it to I_0 and then applies **Step 2** to the new data. If as a result the active set gets updated, then the algorithm is applied again from **Step 1**. We will discuss this in more detail in the full version of the paper, space permitting.

3 Computational results

In this section we present the computational results[5]. We compare INCAS to SVM^{light} [11]. We used a high-end IBM RS/6000 workstation in our experiments. We made the same amount of memory available to both methods (which exceeded 40 megabytes only on 2 examples). Just as in SVM^{light} the sparsity of the examples is exploited by INCAS during the kernel evaluations. Unlike SMO [14] there is no special handling for the case of linear kernel).

We used the following data sets in our experiments:

- Letter-G: The Letter Image Recognition dataset from the UCI Repository [1] - A large number of black-and-white character images were randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15. In the table we examined performances on an arbitrary binary classification problem which was set to separate the letter "G" from all the other letters.

[5] we expect to have more extensive results in the full version of the paper

- OCR: USPS (United States Postal Service) data set of hand written digits. This data set comprises 7291 training and 2007 test patterns, represented as 257 dimensional vectors with entries between 0 and 255. T0(T9) stand for the binary classification problem in which the target is the digit 0(9) vis. the all the other digits.

- Web and Adult[6] : We used the tasks that was compiled by Platt and available from the SMO home page[7]

  - Adult - The goal is to predict whether a household has an income greater than $50000. After discretization of the continuous attributes, there are 123 binary features, with ≈ 14 non-zeros per example.
  - Web - A text classification problem with binary representation based on 300 keyword features. This representation is extremely sparse. On the average there are only ≈ 12 non-zero features per example.

- Abalone: The Abalone dataset from the UCI Repository [1]. Since, we were not interested in evaluating generalization performances, we fed the training algorithm with increasing subsets up to the whole set (of size 4177). The gender encoding (male/female/infant) was mapped into {(1,0,0),(0,1,0),(0,0,1)}. Then data was scaled to lie in the [-1,1] interval.

For each data set we used a selection of kernels and parameters to demonstrate how the performance of the methods is affected by $|I_s|$ - the number of support vectors at the margin, and $|I_c|$ - the number of support vectors at the upper bound. For the same reason we use various values of C. We use RBF kernel with parameter σ. We also use the linear kernel, and we indicate it in the table of results by $\sigma = 0$. We also use polynomial kernel of degree 5 for the Abalone dataset and we indicate it by $\sigma = -5$.

We provide two columns of CPU times for SVM^{light}. The first one, SVM^{light}, contains the time of the runs with default accuracy. The second column, SVM$^{light}_{\epsilon}$ contains the CPU time of the runs with the accuracy set to 10^{-6} which is the accuracy of INCAS.

[6]Original data set is from the UCI Repository [1]

[7]http://www.research.microsoft.com/~jplatt/smo.html

We chose CPU time as the most reasonable performance measure in our setting. A "*" in the table indicates that SVM^{light} was terminated before it reached the solution because it was taking "unreasonably" long time (at least 20 minutes and more than 5 times longer than INCAS). On Adult-7 with $\sigma = 40$ INCAS ran into memory problems due to the fact that $|I_s|$ became greater than 2000, which was the maximum allowed value in the current implementation (recall that INCAS is a FORTRAN 77 code, and hence does not handle dynamic memory allocation). The "-" in the table indicate the failure of INCAS on that problem.

Below is the table of results.

Problem	dim	n	C	σ	$\|I_s\|$	$\|I_c\|$	SVM^{light}	SVM$_\epsilon^{light}$	INCAS
Letter-G	16	20000	100	0	48	1051	*	*	128
Letter-G	16	20000	10	100	193	146	40	78	21
Letter-G	16	20000	10	40	320	57	30	45	22
Letter-G	16	20000	100	40	346	8	56	87	20
OCR-T0	256	7291	100	256	309	0	18	20	12
OCR-T9	256	7291	100	256	378	0	30	38	17
Web-7a	300	24692	1	0	200	724	103	965	238
Web-7a	300	24692	5	200	258	1163	384	410	144
Adult-3a	123	3185	100	100	383	832	53	109	26
Adult-3a	123	3185	100	40	720	503	92	185	183
Adult-3a	123	3185	100	0	151	1025	*	*	71
Adult-7a	123	16101	100	100	871	4824	*	*	1359
Adult-7a	123	16101	100	40	>2000	-	*	*	-
Adult-7a	123	16101	100	0	376	5573	*	*	2108
Abalone	10	4177	100	4	64	1863	673	*	22
Abalone	10	4177	100	-5	163	1700	*	*	73

4 Concluding Remarks

- The results show that our method has overall similar or better performance than SVM^{light} on many SVM problems.
- The suggested approach has an advantage in problems where the solution contains a lot of outliers, such as noisy or inseparable problems.

- On the other hand, it performances drops when there are the solution contain many support vectors on the margin, a setting which usually indicates over-fitting.
- Another advantage of the method is that it does not require the user to "guess" the chunk size and that it typically produces more accurate solution in similar time.
- on the other hand - it is difficult (though possible in some cases) to relax the high accuracy and trade it with running time.
- The "one-variable-at-a-time" increment strategy has many computational advantages over other active set updates schemes. Moreover, it provides the means for a natural extension of the algorithm to be used in an incremental mode of operation.

References

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

[2] B. Boser, I. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

[3] Ferris M. C. and T. S. Munson. Interior point methods for massive support vector machines. Technical Report 00-05, Computer Sciences Department, University of Wisconsin, Madison, WI, 2000.

[4] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 409–415. MIT Press, 2001.

[5] N. Cristianini and J. Shawe-Taylor. *An Introductin to Support Vector Macines and Other Kernel-Based Learning Methods.* Cambridge University Press, 2000.

[6] S. Fine and K. Scheinberg. Efficient svm training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2001.

[7] R. Fletcher. A general quadratic programming algorithm. *Journal of the Institute of Mathematics and Its Applications*, pages 76–91, 1971.

[8] D. Goldfarb. Extension of newton's method and simplex method for solving quadratic problems. In F. A. Lootsma, editor, *Numerical Methods for Nonlinear Optimization*, pages 239–254. Academic Press, London, 1972.

[9] D. Goldfarb and A. Idnani. A numerically stable dual method for solving strictly convex quadratic programms. *Mathematical Programming*, 27:1–33, 1983.

[10] G. H. Golub and Ch. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore and London, 3 edition, 1996.

[11] T. Joachims. Making large-scale support vector machine nlearning practicle. In B. Schölkopf, C. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods*, chapter 12, pages 169–184. MIT Press, 1999.

[12] J Nocedal and S. Wright. *Numerical Optimization*. Springer-Verlag, 1999.

[13] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In *Proceedings of the IEEE Neural Networks for signal Processing VII Workshop*, pages 276–285. IEEE, 1997.

[14] J. C. Platt. Fast trining support vector machines using sequential mininal optimization. In B. Schölkopf, C. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods*, chapter 12, pages 185–208. MIT Press, 1999.