Optimisation on support vector machines

João Pedro Pedroso
Faculdade de Ciências da Universidade de Lisboa
D.E.I.O. and C.I.O., 1749-016 Lisboa, Portugal
e-mail: jpp@fc.ul.pt

Noboru Murata
Riken Brain Science Institute
Hirosawa 2-1, Wako-shi, Saitama 351-0198, Japan
e-mail: mura@brain.riken.go.jp

December 1999

Abstract

In this paper we deal with the optimisation problem involved in determining the maximal margin separation hyperplane in support vector machines. We consider three different formulations, based on L_2 norm distance (the standard case), L_1 norm, and L_∞ norm. We consider separation in the original space of the data (i.e., there are no kernel transformations).

For any of these cases, we focus on the following problem: having the optimal solution for a given training data set, one is given a new training example. The purpose is to use the information about the solution of the problem without the additional example in order to speed up the new optimisation problem. We also consider the case of reoptimisation after removing an example from the data set.

We report results obtained for some standard benchmark problems.

1 Introduction

The support vector machine standard formulation of the problem of optimal separation of two classes of points can be found, for example, in [9]. It consists on finding the hyperplane that separates the two sets in such a way that the (Euclidean, L_2 norm) distance between the hyperplane and nearest point of each of the data sets is maximum. In this paper, we consider additionally the cases of distance measured by the L_1 norm and the L_∞ norm. These last distances allow us to write a version of the problem which shares the theoretical advantages of the original formulation, but that are much simpler to tackle, using linear programming (LP).

The focus of this paper in on optimisation issues for linear separation (i.e., separation not based on kernel transformations), especially for the case of, knowing the optimal separation hyperplane for a given data set, recalculating it when a new training point is added, or an existing training point is removed.

1.1 Support vector machine formulations

Let us label the training data $\boldsymbol{x}_i \in \mathbb{R}^d$ with a label $y_i \in \{-1, +1\}$, for all the training examples $i = 1, \ldots, l$, where l is the number of examples, d is the number of discriminating attributes (i.e., the dimension of the problem). Let us call positive (negative) examples those for which $y_i = +1$ ($y_i = -1$). The set of positive examples is denoted by $\mathcal{P}$, and the set of negative examples is denoted by $\mathcal{N}$.

1.1.1 Separable case

Admitting that there exists a hyperplane $H_{\boldsymbol{w},b} : \boldsymbol{w} \cdot \boldsymbol{x} + b = 0$ separating positive from negative examples, the separation problem is to determine the hyperplane such that $\boldsymbol{w} \cdot \boldsymbol{x}_i + b \geq +1$ for positive examples, $\boldsymbol{w} \cdot \boldsymbol{x}_i + b \leq -1$ for negative examples, with maximal distance to the closest point of each of the classes.

		L_2 norm	L_1 norm	L_∞ norm
distance(H_1, H_2)		$\frac{\lvert b_1 - b_2 \rvert}{\lVert \boldsymbol{w} \rVert_2}$	$\frac{\lvert b_1 - b_2 \rvert}{\lVert \boldsymbol{w} \rVert_\infty}$	$\frac{\lvert b_1 - b_2 \rvert}{\lVert \boldsymbol{w} \rVert_1}$
margin (to maximise)		$\frac{2}{\lVert \boldsymbol{w} \rVert_2} = \frac{2}{\sqrt{\sum_j w_j^2}}$	$\frac{2}{\lVert \boldsymbol{w} \rVert_\infty} = \frac{2}{\max_j \lvert w_j \rvert}$	$\frac{2}{\lVert \boldsymbol{w} \rVert_1} = \frac{2}{\sum_j \lvert w_j \rvert}$
objective (separable case)	$\underset{\boldsymbol{w},b}{\text{minimise}}$	$\sum_j w_j^2$	$\max_j \lvert w_j \rvert$	$\sum_j \lvert w_j \rvert$
objective (non separable case)	$\underset{\boldsymbol{w},b,\boldsymbol{\xi}}{\text{minimise}}$	$\sum_j w_j^2 + C \sum_i \xi_i$	$\max_j \lvert w_j \rvert + C \sum_i \xi_i$	$\sum_j \lvert w_j \rvert + C \sum_i \xi_i$

Table 1: Distance between two parallel hyperplanes $H_1 : \boldsymbol{w} \cdot \boldsymbol{x} + b_1 = 0$ and $H_2 : \boldsymbol{w} \cdot \boldsymbol{x} + b_2 = 0$ for the three norms considered. Margin between the "support" hyperplanes of each class. Objective function for the separable and non-separable cases.

We impose that $\boldsymbol{w} \cdot \boldsymbol{x}_i + b = \pm 1$ for support vectors (this corresponds to scaling $(\boldsymbol{w}, b)$). The distance between the two "support hyperplanes" that go through the support vectors of each class are, then, calculated with the formulae listed in the first row of table 1, which lead to the margins between "support" hyperplanes of the two classes listed on the second row. Using these values for the margins, it is trivial to derive the formulae for the objectives, also on that table, which are to be optimised subject to the classification constraints $\boldsymbol{w} \cdot \boldsymbol{x}_i + b \geq +1 \; \forall i \in \mathcal{P}$ and $\boldsymbol{w} \cdot \boldsymbol{x}_i + b \leq -1 \; \forall i \in \mathcal{N}$. For a complete derivation of these formulae, together with the linear programming formulations for L_1 norm and L_∞ norm, the reader is referred to [7].

One can expect that L_2 norm gives, in the separable case, a result that is "between" L_1 norm and L_∞ norm. Intuitively, we can see that for a given classification problem, the L_∞ norm support vector machine will be closer to a parallel of one of the axes than the L_2 norm machine; and the L_1 norm machine will be closer to a 45 degree hyperplane. This can be visualised in figure 1.

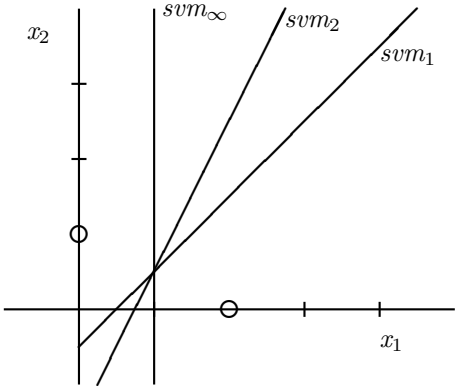

Figure 1: Support vector machines obtained for distances measured with L_1, L_2, and L_∞ norms, when there are two data points to classify: (0,1) for one class and (2,0) for the other.

Another observation that we can make for the support vector machines obtained for each of the norms considered is that as the number of training examples gets larger, and the version space (i.e., the feasible space for separating hyperplanes) gets narrower, the solutions given using L_1, L_2, and L_∞ norms will be closer and closer to each other. On the limit when the version space collapses to a single point, the three solutions are necessarily identical.

The support vectors are the training data points for which $\boldsymbol{w} \cdot \boldsymbol{x}_i + b = \pm 1$, i.e., the points for which the corresponding inequalities are binding. These points can be easily determined, as standard mathematical

programming software outputs values of the slack and dual variables for each constraint.

1.1.2 Non separable case

If the problem stated above has no feasible solution, we can relax the constraints, and add a penalty to the objective function (using the so-called *soft margins*). In this case, the constraints become $\boldsymbol{w} \cdot \boldsymbol{x}_i + b \geq +1 - \xi_i$ for positive examples, $\boldsymbol{w} \cdot \boldsymbol{x}_i + b \leq -1 + \xi_i$ for negative examples, with $\xi_i \geq 0$. One can then add a term to the objective function, proportional to the violation of the original constraints: $C \sum_i \xi_i$, that we want as small as possible. This leads to the formulae of the non-separable objectives in table 1. The parameter C is a user defined value that controls the importance of the classification errors in the training process.

2 Runtime optimisation

In many practical situations, one is confronted with the problem of recomputing the solution after a new training example is added. We are hence dealing with the situation of, given an existing set of examples and the corresponding optimal separation hyperplane, and being supplied a new example, having to resolve for the new optimal separating hyperplane.

Normally, this re-optimisation process should be an easier problem than that of full optimisation, as the solution that we have in hands is likely to give useful information for the calculation of the new one.

We will call *runtime optimisation* to the calculation of the optimal solution when the last example is added, once the solution for all the examples except that last one is known. In this context, by *batch optimisation* we mean the optimisation for the complete data set, without knowledge of a starting solution.

Another situation that may arise in runtime optimisation, that we also handle in this section, is that of removal of a existing training example and subsequent reoptimisation.

2.1 Experimental results

We have tested the performance of the models using benchmark problems commonly referred in the literature. The data for these tests, and their description, are available in [3].

The breast cancer databases (below referred to as "*breast*") was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg [1]. Another test is the heart disease diagnosis test ("*heart*"), which was collected in the Cleveland Clinic Foundation [4]. The "*sonar*" test corresponds to the data set used by Gorman and Sejnowski in their study of the classification of sonar signals using a neural network[5]. The task is to train a network to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. The last test ("*votes*") concerns data of votes for each of the U.S. House of Representatives Congressmen, originally described in [8].

2.1.1 Algorithm's performance

The performance of the algorithm in terms of separation accuracy is the same for batch and runtime optimisation, as the same optimal solution is obtained in both cases. The strategy is to first try to linearly separate the data, and use soft margins only if the problem is infeasible. (In practice, hard margins were used only for the *sonar*test, as all the other tests where not linearly separable). We report in table 2 the results that were obtained in [7], using 10 fold cross validation: each data set was randomly divided into 10 disjoint subsets with identical size; 9 of these subsets randomly chosen were then combined for training, and the remaining one for testing. This process was then repeated, and averaged over 10 trials.

We also report the CPU times involved in the optimisation for the three cases of L_2, L_1 and L_∞ norm distances. The two latter are solvable by linear optimisation, and are numerically much easier to solve than the quadratic optimisation program corresponding to the L_2 norm. The CPU times reported are the percentage of the L_2 norm optimisation time, using the MINOS [6] software as optimiser. This is a solver for constrained, nonlinear optimisation problems. Notice, though, that for the L_1 and L_∞ case, a specialised solver for linear programming is expected to lead to very significant improvements.

		breast	*heart*	*sonar*	*votes*
L_2 norm	train	97.218	84.665	100	97.419
	test	97.567	81.419	92.857	96.337
L_1 norm	train	97.266	84.555	100	97.675
	test	97.567	81.742	93.333	97.262
L_∞ norm	train	97.218	84.665	100	97.444
	test	97.567	81.742	92.381	97.262

Table 2: Average separation performance of support vector machines using different norms, and the parameter $c = 1.0$. Results in bold face correspond to the machines with best empirical performance. (The *sonar*problem is linearly separable, and hence the machine performance does not depend on the parameter C.)

	breast	*heart*	*sonar*	*votes*
L_2 norm	100.0%	100.0%	100.0%	100.0%
L_1 norm	76.2%	51.2%	9.4%	73.2%
L_∞ norm	80.2%	51.2%	17.2%	65.4%

Table 3: Comparison of CPU times for training the support vector machines using the MINOS optimiser, as a percentage of the L_2 norm machine CPU time.

2.2 Runtime optimisation for L_2 norm support vector machines

In nonlinear programming, the knowledge of a good starting solution is very important, as most of the algorithms tend to perform much better when they start close to the optimal solution. In the case of the optimisation problem of L_2 norm support vector machines, we have a quadratic programming, and dual information too can be be exploited by the solver, both in case of addition and removal of training examples, as the existing solution is probably rather close to the optimum. Additionally, in the particular case of removing an example, the solution that had obtained before removal is still feasible. The details on how to exploit this information are algorithm-dependent, and are out of the scope of this paper. Our aim is to assess the order of magnitude of the improvements on the CPU time required for optimisation that can be achieved, using a given numerical solver (in our case MINOS, [6]). See 2.4 and 2.5 for the results obtained.

2.3 Runtime optimisation for L_1 and L_∞ norm

If the formulation of the svm as an LP, we can exploit some properties of linear programming during the runtime optimisation task. Let us start by recalling that adding a new example corresponds to the addition of a new constraint to the LP. This means that the previous solution remains a basic feasible solution for the dual of the new LP. Hence, optimisation of this problem starting from that previous basis is likely to be computationally inexpensive, as compared to solving the batch problem. The most natural thing to do would be to switch to an LP solver like the dual-simplex, that would take full benefit of the known solution.

Nevertheless, for the purposes of comparing results with the L_2 norm case, we have again used MINOS[1], provided it with the known basis (obtained using all but the last training data), and then solved the runtime optimisation problem.

2.4 Results for runtime optimisation: addition of examples

For the numerical experiments that we present in this section, we have used MINOS [6] as the numerical solver for determining the optimal separating hyperplane, for any of the different norms.

We have firstly set up a problem in which we have excluded the last training point for each of the benchmark problems, and solved this problem. We saved the solution (the *basis*) obtained.

[1]MINOS does not include an implementation of the dual simplex method.

Then, we added the constraint corresponding to the last example, and resolved the problem, supplying the solver with the former basis. For each of the L_1, L_2, and L_∞ norm cases, we have compared the CPU time of runtime optimisation with that of batch optimisation. These are the results shown in table 4. As expected, runtime optimisation requires substantially less CPU time that batch optimisation; for the benchmark problems used in this paper, the reduction is to less that 10%, generally speaking.

For each of the benchmarks, we have also compared the CPU times for runtime optimisation for the L_1 and L_∞ norms with the CPU time for runtime optimisation on the L_2 norm case. As for the batch optimisation cases, calculation of the L_2 norm machines requires considerably more time than for the other two norms, and the differences are generally as dramatic as in the batch case. Results are shown in table 4.

EXAMPLE	% of batch optimisation CPU time				% of L_2 norm runtime optim. CPU time			
ADDITION	*breast*	*heart*	*sonar*	*votes*	*breast*	*heart*	*sonar*	*votes*
L_2 norm	6.1	13.9	2.4	7.2	100.0	100.0	100.0	100.0
L_1 norm	3.3	7.5	11.2	4.6	43.8	29.4	50.4	50.4
L_∞ norm	3.1	7.3	5.7	5.2	43.0	27.6	48.7	51.1

Table 4: Results for runtime optimisation, where the last training example of each data set is added, and the optimal separating hyperplane before that addition is known. CPU time for runtime optimisation, as a percentage of the batch optimisation time (left part). CPU time for runtime optimisation for the L_1 and L_2 norm-based machines, as a percentage of that time for the L_2 norm machine (right part).

In a true runtime system, the reoptimisation step would not be always required. Before performing the optimisation, one could check what the margin for the new data point is, with respect to the previous optimal separating hyperplane. If it is greater than the margin for support vectors, it means that the new example is redundant, and therefore the previous hyperplane remains optimal.

2.5 Results for runtime optimisation: removal of examples

For these results, we have firstly saved the solution obtained for the batch problem. We have then removed the last training example, and reoptimised given that solution as the starting point. In table 5, we compared, as before, each of the L_1, L_2, and L_∞ norm cases, in terms of percent of the CPU time of batch optimisation required and in terms of the percent of the CPU time required for runtime optimisation on the L_2 norm case. As in the previous section, runtime optimisation requires much less CPU time that batch optimisation; also, L_2 norm machines again require considerably more time than for the other two norms.

EXAMPLE	% of batch optimisation CPU time				% of L_2 norm runtime optim. CPU time			
REMOVAL	*breast*	*heart*	*sonar*	*votes*	*breast*	*heart*	*sonar*	*votes*
L_2 norm	4.9	12.2	4.8	7.7	100.0	100.0	100.0	100.0
L_1 norm	1.2	5.7	7.4	3.4	32.6	25.7	17.0	34.9
L_∞ norm	2.0	5.9	3.9	5.0	34.3	25.2	16.7	45.8

Table 5: Results for runtime optimisation, where the optimal separating hyperplane for the complete data set is known, and last example is removed. CPU time for runtime optimisation, as a percentage of the batch optimisation time (left part). CPU time for runtime optimisation for the L_1 and L_2 norm-based machines, as a percentage of that time for the L_2 norm machine (right part).

As for the case of adding examples, the removal of an example in a true runtime system does not mean that a reoptimisation step would be required. Whenever the example being removed is not a support vector, its remotion does not affect the optimal separating hyperplane, and reoptimisation in that case is unnecessary.

3 Conclusion

In this paper we numerically exploited the optimisation problems involved in training support vector machines. We have considered three different cases, concerning formulations based on L_1 and L_∞ norms, in addition to the currently used L_2 norm. In addition to the standard, batch optimisation, we considered reoptimisation when a solution is known, and training examples are added or removed.

When the optimal separation problem is solvable by linear programming software (L_1 and L_∞ norms), it becomes much easier to tackle, as shown by the improvements on the training time in the results presented. This is true for both batch optimisation and runtime optimisation.

Runtime optimisation after a new training examples is added or removed is substantially less CPU intensive than batch optimisation. It hopefully opens the application of support vector machines to a wider range of applications, in situations where the training set is supplied dynamically and fast optimisation is crucial.

References

[1] Kristin P. Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1:23–34, 1992.

[2] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2, 1998.

[3] E. Keogh C. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[4] R. Detrano, A. Janosi, W. Steinbrunn, M. Pfisterer, J. Schmid, S. Sandhu, K. Guppy, S. Lee, and V. Froelicher. International application of a new probability algorithm for the diagnosis of coronary artery disease. *American Journal of Cardiology*, 64:304–310, 1989.

[5] 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.

[6] B. A. Murtagh and M. A. Saunders. MINOS 5.5 user's guide. Technical Report SOL 83–20, SOL, Stanford University, Palo Alto, CA, 1978. Revised: July 1998.

[7] João P. Pedroso and Noboru Murata. Support vector machines for linear programming: motivation and formulation. BSIS Technical Report 99-2, Riken Brain Science Institute, Wako-shi, Saitama, Japan, 1999.

[8] J. C. Schlimmer. *Concept acquisition through representational adjustment.* PhD thesis, Department of Information and Computer Science, University of California, Irvine, 1987.

[9] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory.* Springer, 1995.