LIBSVM: Introduction and Benchmarks

Chih-Chung Chang and Chih-Jen Lin*

Abstract

Most available support vector machines (SVM) software are either quite complicated or are not suitable for large problems. Instead of seeking a very fast software for difficult problems, we provide a simple, easy-to-use, and moderately efficient package for SVM classification: LIBSVM. It is a simplification of both SMO by Platt and SVM^{light} by Joachims. LIBSVM is also a simplification of the modification 2 of SMO by Keerthi et al. We hope it helps users from other fields to easily use SVM as a tool. This document includes the introduction of the software and benchmarks.

1 Introduction

Given training vectors $x_i \in R^n, i = 1, \ldots, l$, in two classes, and a vector $y \in R^l$ such that $y_i \in \{1, -1\}$, a major form for solving the support vector machines (SVM) is through the following optimization problem:

$$
\begin{aligned}
\min \quad & \frac{1}{2}\alpha^T Q \alpha - e^T \alpha \\
& 0 \le \alpha_i \le C, \qquad i = 1, \ldots, l \\
& y^T \alpha = 0,
\end{aligned} \tag{1.1}
$$

where e is the vector of all ones, C is the upper bound, Q is an l by l positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$, and $K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j)$ is the kernel. Note that training vectors x_i are mapped into a higher dimensional space by the function ϕ. After solving (1.1), the decision function is

$$
f(x) = \sum_{i=1}^{l} y_i \alpha_i K(x_i, x) + b,
$$

where b is a constant.

The difficulty of solving (1.1) is the density of Q because Q_{ij} is in general not zero. Traditional optimization algorithms such as Newton, Quasi Newton, etc., cannot be directly applied. Several authors (for example, Osuna et al. (1997), Joachims (1998), Platt (1998), and Saunders et al. (1998)) have proposed decomposition methods to conquer this difficulty. Basically they separate the index $\{1, \ldots, l\}$ of the training set to two sets B and N, where B is the working set and $N = \{1, \ldots, l\} \backslash B$. If we denote α_B and α_N as vectors containing corresponding elements, the objective value is equal to $\frac{1}{2}\alpha_B^T Q_{BB} \alpha_B - (e_B - Q_{BN}\alpha_N)^T \alpha_B +$

*Department of Computer Science and Information Engineering, National Taiwan University, Taipei 106, Taiwan (`cjlin@csie.ntu.edu.tw`).

$\frac{1}{2}\alpha_N^T Q_{NN}\alpha_N - e_N^T\alpha_N$. At each iteration, α_N is fixed and the following sub-problem with the variable α_B is solved:

$$\begin{aligned} \min \quad & \frac{1}{2}\alpha_B^T Q_{BB}\alpha_B - (e_B - Q_{BN}\alpha_N)^T\alpha_B \\ & 0 \le (\alpha_B)_i \le C, i = 1, \dots, q, \\ & y_B^T\alpha_B = -y_N^T\alpha_N, \end{aligned} \tag{1.2}$$

where $\begin{bmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{bmatrix}$ is a permutation of the matrix Q and q is the size of B. The strict decrease of the objective function holds and the theoretical convergence was studied in Chang et al. (2000) and Keerthi (2000).

Therefore, Q_{ij} is only calculated when it is used. Usually a special storage using the idea of a cache is used to store recently used Q_{ij}. Hence the computational cost of later iterations can be reduced.

There are different methods for selecting the working set B. For example, in Joachims (1998), the following problem is solved:

$$\begin{aligned} \min \quad & \nabla f(\alpha_k)^T d \\ & y^T d = 0, \ -1 \le d \le 1, && (1.3) \\ & d_i \ge 0, \text{ if } (\alpha_k)_i = 0, \ d_i \le 0, \text{ if } (\alpha_k)_i = C, \\ & |\{d_i \mid d_i \ne 0\}| = q, && (1.4) \end{aligned}$$

where we represent $f(\alpha) \equiv \frac{1}{2}\alpha^T Q\alpha - e^T\alpha$, α_k is the solution at the kth iteration, $\nabla f(\alpha_k)$ is the gradient of $f(\alpha)$ at α_k. Note that $|\{d_i \mid d_i \ne 0\}|$ means the number of components of d which are not zero. The constraint (1.4) implies that a descent direction involving only q variables is obtained. Then components of α_k with non-zero d_i are included in the working set B which is used to construct the sub-problem (1.2). Note that d is only used for identifying B but not as a search direction.

Among available software for large problems, SVM^{light} and BSVM allow users to specify q, the size of the working set. Typically q is from 2 to 100. Therefore, they have to use an optimization software in order to solve (1.2) in each iteration. Since using a numerical optimization procedure is not a trivial task, SVM software become quite complicated.

On the other hand, Sequential Minimal Optimization (SMO) by Platt (1998) puts the decomposition method to the extreme by restricting q to be 2. The main advantage is that an analytic solution of (1.2) can be obtained so there is no need to use an optimization software. Platt proposed a two-loop heuristic to choose the working set B. Available implementation of SMO are, for example, (Ge, 1999) and (Rifkin, 1999). However, they do not implement caches so they might not be suitable for very large problems. In addition, the original SMO may not converge in some special situations so in (Keerthi et al., 1999), two modifications are proposed.

In the last section of (Hsu & Lin, 1999) we have discussed that though from an optimization point of view the decomposition method has the disadvantage of slow convergence, it is enough for typical SVM problems. We suggest that for very difficult models, software like SVM^{light} or BSVM are still necessary as they involve more optimization knowledge and provide more flexibility. On the other hand, for general problems we think that a simple library is useful for users from other fields.

Based on this principle, we implement LIBSVM which is essentially a simplification of both SMO and SVM^{light}. First we restrict $q = 2$ which is the same as SMO. Second, for the working set selection, we think the two-loop heuristic of SMO is a little too complicated. Hence we simply use the strategy of SVM^{light}. We also notice that if without considering the two-loop heuristic of SMO, the modification 2 of (Keerthi et al., 1999) performs the same algorithm of SVM^{light} with $q = 2$. Hence we can also say that LIBSVM is a simplified implementation of the modification 2 of (Keerthi et al., 1999). In the next section we discuss some implementation issues of LIBSVM.

2 Working Set Selection, Stopping Criteria, and Solving the Sub-Problem

When $q = 2$, B has only two elements. If they are i_1 and i_2, the solution of (1.3) has a very simple form:

$$i_1 = \text{argmin}\{\nabla f(\alpha_k)_i d_i \mid y_i d_i = 1; d_i \geq 0, \text{ if } \alpha_i = 0; d_i \leq 0, \text{ if } \alpha_i = C.\}$$
$$i_2 = \text{argmin}\{\nabla f(\alpha_k)_i d_i \mid y_i d_i = -1; d_i \geq 0, \text{ if } \alpha_i = 0; d_i \leq 0, \text{ if } \alpha_i = C.\}$$

That is,

$$i_1 = \text{argmax}(\{-\nabla f(\alpha_k)_i \mid y_i = 1, \alpha_i < C\}, \{\nabla f(\alpha_k)_i \mid y_i = -1, \alpha_i > 0\}), \quad (2.1)$$
$$i_2 = \text{argmax}(\{-\nabla f(\alpha_k)_i \mid y_i = -1, \alpha_i < C\}, \{\nabla f(\alpha_k)_i \mid y_i = 1, \alpha_i > 0\}). \quad (2.2)$$

This becomes a simple and implementable procedure. Here we change argmin to argmax for easy analysis. Note that at the final solution, from the KKT condition, we have

$$\begin{array}{llll} y = 1, \alpha_i < C & \Rightarrow & (Q\alpha)_i + b \geq 1 & \Rightarrow \; b \geq 1 - (Q\alpha)_i = -\nabla f(\alpha_k)_i, \\ y = -1, \alpha_i > 0 & \Rightarrow & (Q\alpha)_i - b \leq 1 & \Rightarrow \; b \geq (Q\alpha)_i - 1 = \nabla f(\alpha_k)_i, \\ y = -1, \alpha_i < C & \Rightarrow & (Q\alpha)_i - b \geq 1 & \Rightarrow \; b \leq (Q\alpha)_i - 1 = \nabla f(\alpha_k)_i, \\ y = 1, \alpha_i > 0 & \Rightarrow & (Q\alpha)_i + b \leq 1 & \Rightarrow \; b \leq 1 - (Q\alpha)_i = -\nabla f(\alpha_k)_i. \end{array} \quad (2.3)$$

We then define

$$O_{i_1} \equiv \begin{cases} -\nabla f(\alpha_k)_{i_1} & \text{if } y_{i_1} = 1, \alpha_{i_1} < C, \\ \nabla f(\alpha_k)_{i_1} & \text{if } y_{i_1} = -1, \alpha_{i_1} > 0, \end{cases} \tag{2.4}$$

and

$$O_{i_2} \equiv \begin{cases} -\nabla f(\alpha_k)_{i_2} & \text{if } y_{i_2} = -1, \alpha_{i_2} < C, \\ \nabla f(\alpha_k)_{i_2} & \text{if } y_{i_2} = 1, \alpha_{i_2} > 0. \end{cases} \tag{2.5}$$

From (2.3), we know

$$O_{i_1} \le -O_{i_2}$$

implies that α is a final solution. Practically the stopping criteria can be written and implemented as:

$$O_{i_1} \le -O_{i_2} + \epsilon, \tag{2.6}$$

where ϵ is a small number.

With (2.1), (2.2), and (2.6), the working set selection and stopping criteria are in a short subroutine:

```
// return 1 if already optimal, return 0 otherwise
int Solver::select_working_set(int &out_i, int &out_j)
{
        double Omax1 = -INF;      // max { -grad(f)_i * d | y_i*d = +1 }
        int Omax1_idx = -1;

        double Omax2 = -INF;      // max { -grad(f)_i * d | y_i*d = -1 }
        int Omax2_idx = -1;

        for(int i=0;i<l;i++)
        {
                if(y[i]==+1)    // y = +1
                {
                        if(!is_upper_bound(i))  // d = +1
                        {
                                if(-O[i] > Omax1)
                                {
                                        Omax1 = -O[i];
                                        Omax1_idx = i;
                                }
                        }
                        if(!is_lower_bound(i))  // d = -1
                        {
                                if(O[i] > Omax2)
                                {
                                        Omax2 = O[i];
```

```
                                        Omax2_idx = i;
                                }
                        }
                }
                else            // y = -1
                {
                        if(!is_upper_bound(i))  // d = +1
                        {
                                if(-O[i] > Omax2)
                                {
                                        Omax2 = -O[i];
                                        Omax2_idx = i;
                                }
                        }
                        if(!is_lower_bound(i))  // d = -1
                        {
                                if(O[i] > Omax1)
                                {
                                        Omax1 = O[i];
                                        Omax1_idx = i;
                                }
                        }
                }
        }

        if(Omax1+Omax2 < eps)
                return 1;

        out_i = Omax1_idx;
        out_j = Omax2_idx;
        return 0;
}
```

This is simpler than the derivation of (Keerthi et al., 1999). We start from the problem (1.3) for finding the working set B. In (Keerthi et al., 1999) it is derived from considering the violation of the KKT condition where several index sets have to be maintained.

Beside the working set selection, another important issue is to solve the sub-problem (1.2). Now (1.2) is a simple problem with only two variables. The detail for finding an analytic solution is in (Platt, 1998). Here we only discuss some additional numerical issues.

Basically Platt substituted $\alpha_{i_1} = y_{i_1}(-y_N^T\alpha_N - y_{i_2}\alpha_{i_2})$ into the objective function of (1.2) and solved an unconstrained minimization on α_{i_2}. The following solution is obtained:

$$\alpha_{i_2}^{new} = \begin{cases} \alpha_{i_2} + \frac{-O_{i_1}-O_{i_2}}{Q_{i_1i_1}+Q_{i_2i_2}+2Q_{i_1i_2}} & \text{if } y_{i_1} \neq y_{i_2}, \\ \alpha_{i_2} + \frac{O_{i_1}-O_{i_2}}{Q_{i_1i_1}+Q_{i_2i_2}-2Q_{i_1i_2}} & \text{if } y_{i_1} = y_{i_2}. \end{cases} \tag{2.7}$$

If this value is outside the possible region of α_{i_2} (that is, exceeds the feasible region of (1.2)), the value of (2.7) is clipped into the feasible region and is assigned as the new α_{i_2}.

Note that here we do not follow the formulation in (Platt, 1998) as we intend to continue the derivation of (2.3)-(2.6). By some very simple calculation, we can show that (2.7) is the same as the one used by Platt.

We may worry that the denominator in (2.7) is zero sometime. That is, Q_{BB} is a 2 by 2 singular matrix so Q is only a positive semidefinite matrix. This happens, for example, in the following situations:

1. The function ϕ does not map data to independent vectors in a higher dimensional space so Q is only positive semidefinite. For example, using the linear or low degree polynomial kernels. Then it is possible that a singular Q_{BB} is picked.

2. Some kernels have a nice property that $\phi(x_i), i = 1, \ldots, l$ are independent if $x_i \neq x_j$. Thus Q as well as all possible Q_{BB} are positive definite. An example is the RBF kernel (Micchelli, 1986). However, for many practical data we have encountered, some of $x_i, i = 1, \ldots, l$ are the same. Therefore, several rows (columns) of Q are exactly the same so Q_{BB} may be singular.

In general optimization software face more difficulties for solving a semidefinite problem than a positive definite one. Therefore, for software like BSVM or SVM^{light}, choosing an appropriate optimization software is crucial.

For the algorithm of LIBSVM, we claim that during the iterative process the second situation, that is, duplicated rows (columns) of Q, does not happen. In other words, the algorithm does not select a working set $B = \{i_1, i_2\}$ where $i_1 \neq i_2$ but $[x_{i_1}; y_{i_1}] = [x_{i_2}; y_{i_2}]$. Otherwise, from (2.4) and (2.5), $O_{i_1} = -O_{i_2}$. That means we are already at a solution and causes a contradiction. On the other hand, if $q > 2$, by using (1.2), it is possible that some i_1 and i_2 with $[x_{i_1}; y_{i_1}] = [x_{i_2}; y_{i_2}]$ are included in the working set B.

The above advantage does not rule out all possibilities that the denominator of (2.7) is zero. Next we explain that even if it is zero, there are no numerical problems: From (2.6), we note that

$$O_{i_1} + O_{i_2} \geq \epsilon$$

during the iterative process. Hence the situation of 0/0 which is defined as NaN by IEEE standard does not appear. Therefore, (2.7) returns $\pm\infty$ if the denominator is zero. Note that $\pm\infty$ and NaN here are special quantities of IEEE standard. Then we can continue the operation on $\pm\infty$ to clip them to regular floating-point numbers. We have to make sure first that 0/0 does not happen because the operation on clipping NaN does not return meaningful values.

If without considering the cache implementation and input/output subroutines, the main program of the iterative process is very short. Hence in this report we do not supply any pseudo code. All details can be directly found in the file svm.cpp.

3 Benchmarks

Table 3.1: RBF kernel and $C = 1$

Problem	l	LIBSVM	BSVM	SVM^{light}
australian	690	0.33	0.40	0.75
diabetes	768	0.38	0.48	0.38
german	1000	1.25	1.30	1.17
heart	270	0.07	0.10	0.09
vehicle	846	0.70	0.85	0.82
satimage	4435	8.09	8.33	5.35
letter	15000	21.71	22.85	20.36
shuttle	43500	393.72	479.20	478.28
dna	2000	8.09	8.56	6.15
segment	2310	2.09	2.42	1.56
fourclass	862	0.28	0.73	0.41
adult1	1605	2.01	2.17	2.11
adult4	4781	20.01	21.10	19.25
web1	2477	1.00	2.27	3.05
web4	7366	6.73	13.94	15.22
web7	24692	68.72	162.16	154.10

Table 3.2: RBF kernel and $C = 1000$

Problem	l	LIBSVM	BSVM	SVM^{light}
australian	690	10.45	7.10	176.44
diabetes	768	17.36	11.62	115.26
german	1000	10.79	13.13	93.13
heart	270	0.35	0.37	7.01
vehicle	846	18.77	6.80	268.53
satimage	4435	27.42	17.05	71.68
letter	15000	134.18	116.38	135.39
shuttle	43500	1146.26	323.74	1243.39
dna	2000	5.96	7.17	5.93
segment	2310	1.25	1.06	1.87
fourclass	862	8.19	0.57	19.41
adult1	1605	21.70	11.37	49.85
adult4	4781	236.98	296.63	1109.83
web1	2477	4.79	3.40	7.87
web4	7366	65.90	28.32	56.35
web7	24692	804.74	671.64	971.40

In this section, we compare LIBSVM (Version 1.0), BSVM[‡] (Version 1.0), and SVM^{light}[§] (Version 3.2). The computational experiments for this section were done on a Pentium III-

500 using the gcc compiler. Details of test problems and test environment can be found in (Hsu & Lin, 1999, Section 4).

For LIBSVM, we use (2.6) with $\epsilon = 0.001$ as the stopping criteria. All three software implement a cache for avoiding the recomputation of columns of Q. For experiments here we use 160MB as the cache size.

Results using RBF kernel are in Tables 3.1 to 3.2. We present the computational time (in *seconds*).

We use all default options of BSVM and SVM^{light}. Note that here we do not use the default optimizer of SVM^{light} for solving (1.1). Following the suggestion by Joachims we link SVM^{light} with LOQO (Vanderbei, 1994) to achieve better stability.

From two tables we can see that LIBSVM is able to solve all problems stably and is competitive with the other two software.

4 Further Information

LIBSVM is available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

Acknowledgments

This work was supported in part by the National Science Council of Taiwan via the grant NSC 89-2213-E-002-013. The authors thank Chih-Wei Hsu and Jen-Hao Lee for many helpful discussions and comments.

References

Chang, C.-C., Hsu, C.-W., & Lin, C.-J. (2000). The analysis of decomposition methods for support vector machines. *IEEE Trans. Neural Networks.* (To appear)

Ge, X. (1999). *Sequential minimal optimization* (Tech. Rep.). (Software available at `http://www.ics.uci.edu/~xge/svm/`)

Hsu, C.-W., & Lin, C.-J. (1999). *A simple decomposition method for support vector machines* (Tech. Rep.). Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan.

‡ BSVM is available at `http://www.csie.ntu.edu.tw/~cjlin/bsvm`

§ SVM^{light} is available at `http://www-ai.cs.uni-dortmund.de/FORSCHUNG/VERFAHREN/SVM_LIGHT/svm_light.eng.html`

Joachims, T. (1998). Making large-scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, & A. J. Smola (Eds.), *Advances in kernel methods - support vector learning.* Cambridge, MA: MIT Press.

Keerthi, S., Shevade, S., Bhattacharyya, C., & Murthy, K. (1999). *Improvements to Platt's SMO algorithm for SVM classifier design* (Tech. Rep.). Department of Mechanical and Production Engineering, National University of Singapore.

Keerthi, S. S. (2000). *Convergence of a generalized SMO algorithm for SVM classifier design* (Tech. Rep. No. CD-00-01). Singapore: Department of Mechanical and Production Engineering, National University of Singapore.

Micchelli, C. A. (1986). Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constructive Approximation*, *2*, 11-22.

Osuna, E., Freund, R., & Girosi, F. (1997). Training support vector machines: An application to face detection. In *Proceedings of CVPR'97.*

Platt, J. C. (1998). Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, & A. J. Smola (Eds.), *Advances in kernel methods - support vector learning.* Cambridge, MA: MIT Press.

Rifkin, R. (1999). *SvmFu* (Tech. Rep.). (Software available at `http://five-percent-nation.mit.edu/PersonalPages/rif/SvmFu/index.html`)

Saunders, C., Stitson, M. O., Weston, J., Bottou, L., Schölkopf, B., & Smola, A. (1998). *Support vector machine reference manual* (Tech. Rep. No. CSD-TR-98-03). Egham, UK: Royal Holloway, University of London.

Vanderbei, R. (1994). *LOQO: An interior point code for quadratic programming* (Tech. Rep. No. SOR 94-15). Statistics and Operations Research, Princeton University. (revised November, 1998)