An Improved Caching Strategy for Training SVMs

Liang Zhou[1] **Fen Xia**[1] **Yanwu Yang**[1]

[1]Institute of Automation,Chinese Academy of Sciences, Beijing 100080, P. R. China

Abstract

Computational complexity is one of the most important issues while dealing with the training of Support Vector Machines(SVMs), which is done by solving corresponding linear constrained convex quadratic programming problems. The state-of-the-art training of SVMs takes iterative decomposition strategies that focus on working-set selection to solve quadratic programming problems. Shrinking and caching are two indispensable strategies to reduce the complexity of the decomposition process. Yet, most existing caching strategies mainly consider usage records of samples, while ignoring probabilities of samples being selected into working sets. These probabilities might determine the efficiency of caching. This paper proposes an improved caching strategy by taking into account these probabilities of samples being selected into working sets, to reduce computational costs of kernel evaluations in the training of SVMs. Experiments on several benchmark data sets show that our caching strategy is more efficient than those existing ones.

Keywords: Support vector machine, Working set selection, Shrinking, Caching, Kernel evaluation

1. Introduction

Support Vector Machines(SVMs) are a family of powerful statistical learning techniques for pattern recognition, regression and density estimation problems. SVMs have proven to be effective in many practical applications. SVMs learning methods are based on the structural risk minimization (SRM) induction principle, which is derived from the statistical learning theory [1]-[3]. The problem of training SVMs is converted to a linear constrained convex quadratic programming problem.

A lot of commercial packages are used to solve quadratic programming problems. In these packages, a whole kernel matrix Q is held in the memory, thus the quadratic programming problem has the space complexity $O(l^2)$ and the time complexity $O(l^3)$, where l is the size of training set. When l is too large, it's impossible to afford the space and the time. Decomposition algorithms are mainstream approaches to tackle this problem. They decompose a large-scale optimization problem into several small-scale sub-problems. By solving a series of small-scale sub-problems sequently, they obtain a solution to the large-scale optimization problem. At each step a fixed-size subset(called working-set B) of the whole training set is chosen to form a small-scale optimization sub-problem. When the small-scale optimization sub-problem is solved, some samples of the working-set are substituted by those outside (of working set) with some working-set selection strategies. Then the problem of solving the small-scale sub-problem is formulated in the same fashion through an iterative process. Finally, decomposition algorithms terminate when some criteria (e.g. KKT conditions for all samples) are satisfied.

Kernel evaluations are the main computational complexity. So the number of kernel evaluations is a good criteria to evaluate the performance of training SVMs. Some algorithms are proposed to reduce the total number of iteration [4] or to minimize the number of kernel evaluations per iteration [5].

The state-of-the-art training of SVMs takes iterative decomposition strategies that focus on working-set selection to solve quadratic programming problems. A series of kernel evaluations are repeated in the training of SVMs, therefore, caching strategies can be used to avoid the recomputation of those evaluations. This implies searching for an elegant trade-off between memory consumption and the training time. In many tasks the number of Support Vectors(SVs)[1] is much smaller than the number of training samples. This implies that, kernel evaluations can be dealt by only considering evaluations relevant to those SVs, which approximates to some shrinking strategies.

[1]In (2), support vectors refers to samples whose corresponding α_i is not zero.

Shrinking and caching are two indispensable strategies to reduce the complexity of the decomposition process. Joachims [6] first took shrinking and caching strategies to train SVMs, and previous experiments demonstrated that shrinking and caching had very good effects. To the best of our knowledge, there are two popular packages of training SVMs using caching strategies: SVM^{light} and $LIBSVM$. The former records the times of (samples) being selected into the working-set, then replaces samples with least times in caching by those being selected. The latter views the caching as a queue. and deletes the head of the queue while adding samples being selected into the tail of the queue. Most existing caching strategies mainly consider usage records of samples, while ignoring probabilities of samples being selected into working sets. These probabilities might determine the efficiency of caching. In this paper, we propose an improved caching strategy with consideration of these probabilities to save computational costs of kernel evaluation in the training of SVMs. Experiments on several benchmark data sets show that our caching strategy is more efficient than those existing ones.

The remaining of the paper is organized as follows. Section 2 introduces the standard SVMs. Section 3 discusses the general decomposition algorithm and some existing working-set selection algorithms. Section 4 describes most existing shrinking and caching strategies. An improved caching strategy is proposed in Section 5. Experiments and discussions are presented in Section 6. Section 7 concludes the paper.

2. Standard support vector machines

To facilitate the reading, here are some important notations in this paper:

- l: size of training samples
- n: dimension of the sample's feature
- $\phi(x)$: the mapping function
- $\mathbf{w}$: weights vector
- α: Lagrange multipliers vector
- b: bias term
- K: sample inner product=$< \phi(x_i), \phi(x_j) >$
- ϵ: stop condition
- Q: kernel matrix= $y_i y_j < \phi(x_i), \phi(x_j) >$
- B: working set
- $\mathbf{e}$: the vector of all ones
- C: regularization parameter (≥ 0)
- $M(\alpha)$: $\min\limits_{i \in I_{low}(\alpha)} -y_i \bigtriangledown f(\alpha)_i$
- $m(\alpha)$: $\max\limits_{i \in I_{up}(\alpha)} -y_i \bigtriangledown f(\alpha)_i$
- I_{up} (α): $\{t \mid \alpha_t \leq C, y_t = 1 \parallel \alpha_t \geq 0, y_t = -1\}$,
- $I_{low}(\alpha)$: $\{t \mid \alpha_t \leq C, y_t = -1 \parallel \alpha_t \geq 0, y_t = 1\}$,

In this paper, we just consider standard SVMs(1-norm, proposed by V.N.Vapnik[1][2] for classification). Given training vectors $\mathbf{x}_i \in R^n, i = 1, ..., l$, in two classes, and a vector $\mathbf{y} \in R^l$ such that $y_i \in \{1, -1\}$, the standard SVMs solve the following primal problem:

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b \geq 1 - \xi_i, \\ & \xi_i \geq 0, i = 1, ..., l. \end{aligned} \tag{1}$$

Its dual problem is

$$\begin{aligned} \min_{\alpha} \quad & \tfrac{1}{2}\alpha^T Q\alpha - \mathbf{e}^T\alpha \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, i = 1, ..., l, \\ & \sum_{i=1}^{l} y_i\alpha_i = 0. \end{aligned} \tag{2}$$

Usually, we solve the dual problem (2), as it is easier to solve (2) than (1).

3. Decomposition algorithm and working set selection

In this section, we introduce the history and details of decomposition algorithm. First, the simplest heuristic is known as *chunking* by V.N.Vapnik[1]. And then, Osuna [7] proved that if B contains a variable, which violates the optimality condition, the object function will have a strict improvement when the sub-optimal is re-optimized. Osuna also gave an improved algorithm for training SVMs.

The Sequential Minimal Optimization (SMO) [8] algorithm is derived by taking the idea of the decomposition method to its extreme and optimizing a minimal subset of just two points[2] at each iteration. The power of this technique resides in the fact that the optimization problem for two data points admits an analytical solution, eliminating the need to use an iterative quadratic programme optimizer as part of the algorithm. In general a decomposition algorithm can be formulated as table 1.

[2]The requirement that the condition $\sum_{i=1}^{l} y_i\alpha_i = 0$ is enforced throughout the iterations implies that the smallest number of B is 2.

1 Given training set $\{x_i, i = 1, ..., l.\}$
2 $\alpha^{\mathbf{0}} \leftarrow$ feasible starting point, $t = 1$
3 **Do**
4 select working set B^t
5 solve $f_{B^{(t)}}$ and update α^t
6 $t = t + 1$
7 **While** stopping criterion is not met

Table 1: Decomposition Algorithm In General

1 Select
2 $i \in arg\max_t\{-y_t \bigtriangledown f(\alpha^k)_t \mid t \in I_{up}(\alpha^k)\}$,
3 $j \in arg\min_t\{-y_t \bigtriangledown f(\alpha^k)_t \mid t \in I_{low}(\alpha^k)\}$,
4 Return $B = \{i, j\}$.

Table 2: Maximal Violating Pair

The fourth step in table 1 decides the process of the algorithm iterative, and the convergence of the decomposition algorithm depends strongly on the working set selection algorithm. There are two main categories of working set selection algorithm, one uses first-order information and the other uses second-order information. The first-order information algorithm selects pairs of variables that mostly violate(MVP) [9][3], which is formulated as table 2, the Karush-Kuhn-Tucker(KKT) condition for optimality.

So far, there are two algorithms taking second-order information into account(LIBSVM-2.8 [4] [10] and HMG [5]). Although both of them use second-order information, ultimately they could not avoid all C_l^2 pairs to get the optimal B. In fact, they all use the greedy strategy. LIBSVM-2.8 selects the same i as MVP, and then checks only $O(l)$ pairs to decide j, while HMG takes one variable of the previous B^t into the current B. They both have disadvantages in per iteration: LIBSVM-2.8 can not guarantee that the corresponding kernels[4] of i are available in the matrix cache, while HMG can only take full advantage of a sample point of information in each iteration as it takes one point of the previous B^t. The LIBSVM-2.8 and HMG working set selection algorithm are presented in table 3 and table 4 respectively. Next section will give the details of shrinking and caching strategies used in popular software($LIBSVM$ and SVM^{light}) of training SVMs.

[3]Detailed derivation is specified in [4] [9].

[4]Evaluating the sub-object function should use the kernel $Q_{ij}, j = 1, ..., l$.

$f_{sub}(i, j)$ is the sub-optimal problem defined at [3].
1 $\alpha^{\mathbf{0}} \leftarrow$ feasible starting point, $t = 1$
2 select $i \in arg\max_t\{-y_t \bigtriangledown f(\alpha^k)_t \mid t \in I_{up}(\alpha^k)\}$,
3 select $j \in arg\min_t\{f_{sub}(i, t) \mid t \in I_{low}(\alpha^k)$,
4 $-y_t \bigtriangledown f(\alpha^k)_t < -y_i \bigtriangledown f(\alpha^k)_i\}$.
5 Return $B^{(t)} = \{i, j\}$.

Table 3: LIBSVM-2.8

4. Shrinking and caching

Shrinking and Caching are two very effective strategies for training SVMs. They are first proposed by Joachims [6], and implemented in software SVM^{light}. Shrinking is based on the idea that those samples, whose α_i are in the value of the border, may not be changed in the future training. So those samples can be shrunken in the training procedure, then the whole training problem becomes smaller and easier to be solved. P.-H. Chen [11] proves the following theorem:

Theorem 1 Assume Q is positive semi-definite and the decomposition algorithm uses LIBSVM-2.8 working set selection algorithm. Let $I \equiv \{i \mid -y_i \bigtriangledown f(\alpha)_i > M(\alpha) \, or \, -y_i \bigtriangledown f(\alpha)_i < m(\alpha)\}$. There is $\overline{k}$ such that after $k > \overline{k}$, every $\alpha_i^k, i \in I$ has reached the unique and bounded optimal solution. It remains the same in all subsequent iterations and $i \in I$ is not in the following set:

$$\{t \mid M(\alpha^k) \leq -y_t \bigtriangledown f(\alpha^k)_t \leq m(\alpha^k)\}. \quad (3)$$

The software $LIBSVM$ [5] shrinks the α_i considering set (3), while SVM^{light} is based on the idea that any α_i which has stayed at the same bound for a certain number of iterations can be removed.

Caching strategy can effectively improve the efficiency of procedures and save training time. However, in large-scale data processing, as lack of memory space, caching strategy has to be used. There are two different relatively simple caching strategies in existing software. The software $LIBSVM$ implements a simple least-recent strategy for caching. It dynamically caches only recently used kernel of Q_{B^t}. Its details are shown in table 5.

And the idea of SVM^{light}, which is based on the least-used strategy, is similar with $LIBSVM$'s least-recent strategy. Therefore, its strategy is to preferentially delete the α_i, which is the least num-

[5]As the implement of HMG is updated from software $LIBSVM$, it uses the same shrinking and caching techniques as $LIBSVM$.

$f_{B^{(t)}}$ is the sub-optimal problem defined at [4].
1 **If** $t = 1$
2 Select arbitrary $B^1 = \{b_1, b_2\}, y_{b_1} \neq y_{b_2}$
3 **Else**
4 **If** $\forall i \in B^{(t-1)} : \alpha_i \leq \eta \cdot C \parallel \alpha_i \geq (1-\eta) \cdot C$
5 $i \in arg\max_t\{-y_t \bigtriangledown f(\alpha^k)_t \mid t \in I_{up}(\alpha^k)\}$,
6 $j \in arg\min_t\{-y_t \bigtriangledown f(\alpha^k)_t \mid t \in I_{low}(\alpha^k)\}$,
7 **Else**
8 select pair $B^{(t)} = \{b_1, b_2 \mid arg\max_{b_1,b_2} f_{B^{(t)}}$,
9 where $b_1 \in B^{(t-1)}, b_2 \in \{1, ..., l\}$,
10 **Return** $B^{(t)} = \{i, j\}$.

Table 4: Hybrid Maximum-Gain $0 < \eta \ll 1$

1 Using a circular list to record α
2 If α_i's kernel is called
3 Add α_i to the tail of the circular list
4 If not sufficient memory space for α_i
5 release α_j from the circular list's head
6 until sufficient memory space for α.
7 End
8 End

Table 5: Caching in $LIBSVM$

ber of kernel used, in the current cache. Also, details are shown in table 6.

5. An improved caching strategy

Currently, the focus of decomposition algorithm is how to select the working set. The most popular algorithm is using the second-order information of the sub-optimal problem to select working

1 Using a circular list to record α
2 Using a counter cnt_i to record each α_i
3 If α_i's kernel is called and
4 the $cnt_i = cnt_i + 1$ Then
4 Add α_i to the tail of the circular list
5 If not sufficient memory space for α_i
6 release α_j which is the least cnt_j
7 until sufficient memory space for α_i.
8 End
9 End

Table 6: Caching in SVM^{light}

1 Using a circular list to record α
2 update $\alpha_i^{old} \rightarrow \alpha_i^{new}$
3 If $\alpha_i^{new} < \epsilon \parallel \alpha_i^{new} > C - \epsilon$
4 set α_i as preferentially to delete
5 Else
6 set α_i as not preferentially to delete
7 End
8 If α_i's kernel is called
9 If not sufficient memory space for α_i And
10 the α_j is preferentially to delete Then
11 release α_j
12 Until sufficient memory space for α_i.
13 End
14 End

Table 7: A New Caching Strategy

set. Just as mentioned in section 3, there are two main working set selection algorithms using second-order information: HMG and LIBSVM-2.8. The difference between LIBSVM-2.8 and HMG working set selection algorithm, which are specified in table 3 and 4 respectively, is that they use different way to solve the similar second-order sub-optimal problem. Although the working set selection algorithm decides the training process, different strategies of shrinking and caching will have different effect on saving time. As far as shrinking and caching are concerned, the implement of HMG is the same as LIBSVM-2.8. Compared to $LIBSVM$, SVM^{light} uses first-order information of the sub-optimal problem to select working set, as well as different shrinking and caching strategies. This paper focuses on proposing a generic caching strategy that is based on the combination of working set selection algorithm and shrinking strategy. The specific strategy can be referred to table 7.

This strategy fully takes into account the interdependency of the working set selection, shrinking and caching, which are three different main aspects in the special optimization problem of SVMs. As in the HMG working set selection algorithm, if current $\alpha_i \in B$ is in the border, this α_i will have a great probability of not being re-selected. Meanwhile in the $LIBSVM$, the nature of shrinking algorithm is also removing the border α_i from the training set. As we already know the fact that some samples will no longer be selected into set B, we can base on this real-time information to improve caching strategy. As in the table 7, if $\alpha_i^{new} < \epsilon \parallel \alpha_i^{new} > C - \epsilon$, the α_i^{new} will have a great probability of not being re-selected to set B. Based on the above considerations, we present this generic caching strategy,

Data set	# data	# feature	# class
$a9a$	32,561	123	2
$ijcnn1$	49,990	22	2
$w8a$	49,749	300	2
$protein$	14,895	357	3
$usps$	7,291	256	10

Table 8: Data Statistics

which is preferentially to delete the α_i on the border. This strategy is not ever considered in early implement. Now, we propose the kernel evaluation rate for evaluation criteria :

$$Rate_{ke} = \frac{\#\quad total\quad kernel\quad evaluations}{\#\quad total\quad kernel\quad calls} \tag{4}$$

Total kernel evaluations refers to the actual amount of computation kernel, while total kernel calls refers to the actual demand of kernel. Clearly, caching will not affect the iteration of training process, the entire iterative process is decided by the working set selection algorithms. But as a caching strategy, a lower kernel evaluation rate means that more kernel evaluations are avoided. This can be calculated from the total amount of kernel evaluation, a common evaluation criteria, to be tested.

6. Experiments

In this section we use $LIBSVM$ software as our benchmark. We will adapt the caching strategy of the primitive $LIBSVM$ to *Bound* (we proposed) and $Used$(caching strategy in software SVM^{light}). we name primitive $LIBSVM$'s caching strategy as *Recent*. With such three caching strategies implement, comparison will be performed between these three versions.

Five data sets($a9a$, $ijcnn1$, $w8a$, $protein$ and $usps$)[6] are considered and we only take classification for comparison. We compare the performance of our caching strategy against the existing ones. Data statistics are specified in table 8.

6.1. Experimental settings

We know that the SVMs can be used for classification, one class, regression and density estimation. We use classification, the basic usage of SVMs, to evaluate different caching strategies. In classification, different SVMs parameters such as C in (1) and kernel parameters affect the training procedure. It is difficult to evaluate the three caching strategies under every parameter setting. Here we use the similar experiments setting as [4]. To have a fair comparison, we simulate how one uses SVMs in practice. We consider two following training procedures:

1. "Parameters selection": using five-fold cross validation to find the best parameters.
2. "Final training": train the whole set.

In "Parameters selection" step, we use a fixed small caching space to evaluate the three caching strategies, while in "final training" step, we use various caching space. Each step will be with or without shrinking. For simpleness, we only use the RBF kernel($\gamma = 1$ for this experiments):

$$K(x_i, x_j) = e^{-\gamma\|x_i - x_j\|^2} \tag{5}$$

For completeness we give the details of experiment platform here. The experiments are reported to have been carried out on a 2.4 GHz Pentium 4 processor with 512M of RAM running cygwin, using g++ 3.4.4 compiler.

6.2. Comparison and analysis

We train SVMs on all data sets as shown in Table 8 to compare these three caching strategies. The kernel evaluation rate(4) is used as the evaluation criteria of performance. Results are shown in table 9 and table 10. Table 9 shows results of "Parameters selection" procedure, which uses fixed caching space(10Mb) and different C varying among 1, 10 and 100. Table 10 is for the "Final training" procedure. It uses the same C(100), but different caching space(from 10 to 200). In the table 9 and table 10, the lowest values are boldfaced, which mean more kernel evaluations are saved.

As the number of kernel evaluation is independent of the caching space, the training time is proportional to kernel evaluation rate. In the "Final training" procedure, the processes of training with the three caching strategies are same, which means the total numbers of kernel evaluation are also same, while the caching spaces are different. Table 10 shows that the *Bound* caching strategy performs best and the $Used$ is the worst among the three caching strategies. In our version of $Used$, if the sample is deleted from the circular list, the cnt_i will be set 0. This might be not appropriate. We leave it for the further work.

[6] All data sets presented in this paper are available at http://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/.

strategy	a9a,No shrinking		
	1	10	100
Recent	0.374	0.412	0.534
Used	0.412	0.547	0.647
Bound	**0.368**	**0.402**	**0.532**
	a9a, shrinking		
	1	10	100
Recent	**0.418**	0.429	0.412
Used	0.425	0.459	0.584
Bound	0.420	**0.427**	**0.410**
	ijcnn1,No shrinking		
	1	10	100
Recent	0.384	0.339	0.311
Used	0.404	0.429	0.586
Bound	**0.379**	**0.327**	**0.303**
	ijcnn1, shrinking		
	1	10	100
Recent	**0.408**	**0.411**	0.424
Used	0.412	0.464	0.490
Bound	**0.408**	**0.411**	**0.421**
	w8a,No shrinking		
	1	10	100
Recent	0.410	0.457	0.493
Used	0.453	0.559	0.632
Bound	**0.395**	**0.443**	**0.491**
	w8a, shrinking		
	1	10	100
Recent	0.435	0.433	0.341
Used	0.457	0.540	0.570
Bound	**0.433**	**0.422**	**0.336**
	protein,No shrinking		
	1	10	100
Recent	**0.387**	0.242	0.303
Used	0.402	0.606	0.468
Bound	**0.387**	**0.230**	**0.290**
	protein, shrinking		
	1	10	100
Recent	**0.412**	0.380	0.397
Used	0.428	0.446	0.594
Bound	0.416	**0.378**	**0.390**
	usps,No shrinking		
	1	10	100
Recent	0.212	0.069	0.053
Used	0.212	0.069	0.053
Bound	0.212	0.069	0.052
	usps, shrinking		
	1	10	100
Recent	0.212	0.069	0.054
Used	0.212	0.069	0.054
Bound	0.212	0.069	0.054

Table 9: caching space=10Mb,cv = 5 fold,C=?

strategy	a9a,No shrinking				
	10	20	40	100	200
Recent	0.569	0.501	0.375	0.105	0.047
Used	0.645	0.642	0.610	0.590	0.505
Bound	**0.568**	**0.499**	**0.370**	**0.094**	**0.038**
	a9a, shrinking				
	10	20	40	100	200
Recent	0.433	0.393	0.349	0.282	0.218
Used	0.593	0.577	0.556	0.524	0.485
Bound	**0.431**	**0.391**	**0.345**	**0.276**	**0.214**
	ijcnn1,No shrinking				
	10	20	40	100	200
Recent	0.373	0.244	0.150	0.115	0.105
Used	0.573	0.572	0.563	0.326	0.287
Bound	**0.365**	**0.230**	**0.125**	**0.099**	**0.099**
	ijcnn1, shrinking				
	10	20	40	100	200
Recent	0.436	0.410	0.375	0.337	0.318
Used	0.493	0.488	0.478	0.463	0.457
Bound	**0.435**	**0.407**	**0.370**	**0.335**	**0.317**
	w8a,No shrinking				
	10	20	40	100	200
Recent	0.523	0.433	0.273	0.053	0.032
Used	0.636	0.621	0.534	0.479	0.349
Bound	**0.521**	**0.429**	**0.265**	**0.042**	**0.028**
	w8a, shrinking				
	10	20	40	100	200
Recent	0.400	0.267	0.194	0.136	0.100
Used	0.570	0.544	0.517	0.483	0.298
Bound	**0.396**	**0.260**	**0.187**	**0.126**	**0.088**
	protein,No shrinking				
	10	20	40	100	200
Recent	0.412	0.191	0.110	0.078	0.060
Used	0.611	0.604	0.590	0.507	0.274
Bound	**0.399**	**0.168**	**0.087**	**0.061**	**0.052**
	protein, shrinking				
	10	20	40	100	200
Recent	0.432	0.369	0.299	0.224	0.172
Used	0.594	0.587	0.565	0.440	0.244
Bound	**0.425**	**0.358**	**0.280**	**0.187**	**0.153**
	usps,No shrinking				
	10	20	40	100	200
Recent	0.052	0.052	0.052	0.052	0.052
Used	0.052	0.052	0.052	0.052	0.052
Bound	0.052	0.052	0.052	0.052	0.052
	usps, shrinking				
	10	20	40	100	200
Recent	0.054	0.054	0.054	0.054	0.054
Used	0.054	0.054	0.054	0.054	0.054
Bound	0.053	0.053	0.053	0.053	0.053

Table 10: C=100,caching space=?(Mb)

Experiments with large caching space show that the caching strategy is very important to save time. If the caching space is sufficient enough, the $Rate_{ke}$ will be very low, leading to great reduction of kernel evaluation. For example, *Bound* achieves 0.028 of $Rate_{ke}$ in $w8a$ data set of table 10, when no shrinking is used and caching space is set 200.

Usps data set is a multi-class problem. All three strategies on this data set show good similar performance(the $Rate_{ke}$ is very low, around 0.05). There are two main explanations leading to such an outcome, one is that the software $LIBSVM$ decomposes a multi-class problem into pairwise binary classification problems. The size of training set becomes small. The other is that the size of each class is small. Therefore, even if the caching space is small, it is enough for those one-against-one binary classifications.

In Table 10, all results show that *Bound* perform best in all settings. In Table 9, *Bound* perform best in most settings, but we also note several results of *Recent* are better than *Bound*. A reasonable explanation is that the data sets are different because of the randomness produced in the process cross validation.

Overall, all results in Table 9 and Table 10 are further less than 1, which demonstrate that the caching strategies are effective to reduce the computational complexity. Probabilities of examples being selected in working set mainly contribute to the efficiency of caching. By considering probabilities of examples being selected in working set, our caching strategy achieve best performance on most of data sets.

7. Conclusions

Working-set selection, caching and shrinking are three components which are related to the computational complexity of the training SVMs. Previous works mainly focused on the working-set selection, while only a little works focused on the caching and shrinking strategies. In this paper, we propose an improved caching strategy, taking into account probabilities of samples being selected into working set in the training of SVMs. Experimental results on several benchmark data sets show the performance of our strategy is better than existing ones used in $LIBSVM$ and SVM^{light}.

In the current stage we mainly deal with caching strategy. Good combination of decomposition algorithms, shrinking and caching strategies may further improve the efficiency of training SVMs. Our further work will focus on this combination.

References

[1] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, 1995.

[2] I. B.E.Boser 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,ACM Press*, pages 144–152, 1992.

[3] Nello Cristianini and John Shawe-Taylor *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge University Press ,2000.

[4] R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training SVM. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

[5] C. I. Tobias Glasmachers. Maximum-gain working set selection for svms. *Journal of Machine Learning Research*, 7:1437–1466, July 2006.

[6] T. Joachims. *11 in: Making Large-Scale SVM Learning Practical.Advances in Kernel Methods - Support Vector Learning*. B. Schölkopf and C. Burges and A. Smola (ed.), MIT Press, 1999.

[7] R. F. E. Osuna and F. Girosi. Improved training algorithm for support vector machines. *In J. Principe, L. Giles, N. Morgan, and E. Wilson, editors, Neural Networks for Signal Processing,IEEE Press,*, VII:276–285, 1997.

[8] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C.Burges, and A. J. Smola,editors, *Advances in Kernel Methods - Support Vector Learning,* chapter 12, pages 185–208. MIT Press, 1999.

[9] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13:637–649, 2001.

[10] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[11] P.-H. Chen, R.-E. Fan, and C.-J. Lin. A study on SMO-type decomposition methods for support vector machines. *IEEE Transactions on Neural Networks*, 17:893–908, July 2006.