A parallel implementations of the Sequential Minimal Optimization (SMO) algorithm to train Support Vector Machines

Namit Shetty (namits)
Namit Katariya (nkatariy)

April 18, 2013

15-853 Final Project
The relevant code can be found at
`https://github.com/namitk/algorithms-in-real-world/tree/master/project`

Contents

Chapter 1

Introduction to SVMs

In its simplest linear form, a support vector machine is a hyperplane that separates a set of positive examples from a set of negative examples with the maximum margin. In the linear case, the margin is defines by the distance of the hyperplane to the nearest of the positive and negative examples. Throughout our implementations, we have assumed the following model for the output of a linear SVM

$$u = \vec{w}.\vec{x} - b$$

where $\vec{x}$ is the input vector. The separating hyperplane is the plane $u = 0$. The nearest points lie on the planes $u = \pm 1$ The margin m is thus $m = \frac{1}{\|w\|_2}$

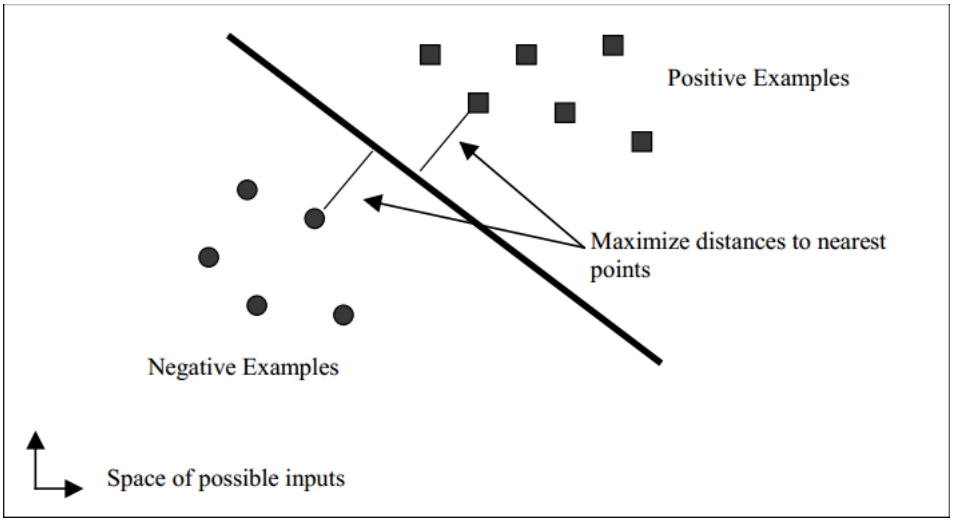


Figure 1.1: Linear support vector machines

1.1 The optimization problem

The objective above can be framed as a convex, quadratic programming optimization problem as illustrated below.

1.1.1 Linearly separable data

$$\begin{aligned} \max_{\vec{w},b} \quad & \frac{1}{2} \cdot \frac{1}{\|w\|_2} \\ \text{subject to} \quad & \vec{w}.\vec{x_i} - b \geq 1, \forall i \text{ s.t.} y_i = 1 \\ & \vec{w}.\vec{x_i} - b \leq -1, \forall i \text{ s.t.} y_i = -1 \end{aligned}$$

where x_i is the i-th training example and y_i is its true label (+1 or -1)

.

This can be equivalently and succinctly framed as

$$\begin{aligned} \min_{\vec{w},b} \quad & \frac{1}{2} \cdot \|w\|_2 \\ \text{subject to} \quad & y_i \cdot (\vec{w}.\vec{x_i} - b) \geq 1, \forall i \end{aligned}$$

1.1.2 Linearly unseparable data

In this case, we allow examples to violate the margin condition but penalize the violation. This can be framed as

$$\begin{aligned} \min_{\vec{w},b,\xi_i} \quad & \frac{1}{2} \cdot \|w\|_2 + C \sum_{i=1}^{N} \xi_i \\ \text{subject to} \quad & y_i \cdot (\vec{w}.\vec{x_i} - b) \geq 1 - \xi_i, \forall i \\ & \xi_i \geq 0 \ \forall i \end{aligned} \tag{1.1}$$

where ξ_i are slack variables that permit margin failure and C is a parameter which trades off wide margin with a small number of margin failures.

1.1.3 Duality

The sequential minimal optimization algorithm attempts to solve the dual of the optimization problem in equation 1.1. Using the Lagrangian, the problem can be converted into its dual form

$$\begin{aligned} \min_{\vec{\alpha}} \quad & \frac{1}{2} \cdot \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j (x_i \cdot x_j) \alpha_i \alpha_j - \sum_{i=1}^{N} \alpha_i \\ \text{subject to} \quad & \sum_{i=1}^{N} y_i \alpha_i = 0 \\ & 0 \leq \alpha_i \leq C \ \forall i \end{aligned} \tag{1.2}$$

Here α_i are the lagrange multipliers. There is a one-to-one relationship between lagrange multipliers and the training examples. In the general i.e where we might look for non-linear separating surfaces, the optimization essential remains the same except $x_i \cdot x_j$ being replaced by $K(x_i, x_j)$ where K is a kernel function that measures the similarity or distance between the vectors x_i and x_j The vector $\vec{w}$ and the threshold b can be derived from the lagrange multipliers using

$$\vec{w} = \sum_{i=1}^{N} y_i \alpha_i \vec{x_i}, \; b = \vec{w}\vec{x_k} - y_k \text{ for some } \alpha_k > 0$$

Note that the slack variables ξ_i do not appear in the dual formulation problem.

The KKT conditions for the QP problem in 1.2 imply that the problem is solved when, for all i

$$\alpha_i = 0 \Leftrightarrow y_i u_i \geq 1 \tag{1.3}$$
$$0 < \alpha_i < C \Leftrightarrow y_i u_i = 1 \tag{1.4}$$
$$\alpha_i = C \Leftrightarrow y_i u_i \leq 1 \tag{1.5}$$

where u_i is the output of the SVM for the i-th training example.

Chapter 2

Sequential Minimal Optimization (SMO)

SMO solves the SVM QP problem by decomposing it into small QP subproblems that can be solved analytically rather than numerically. For the SVM problem, the smalles optimization problem involves two lagrange multipliers because of the linear equality that the multipliers must satisfy. At every step, SMO chooses two lagrange multipliers to jointly optimize, finds the optimal values for these multipliers and updates the SVM to reflect the new optimal values.

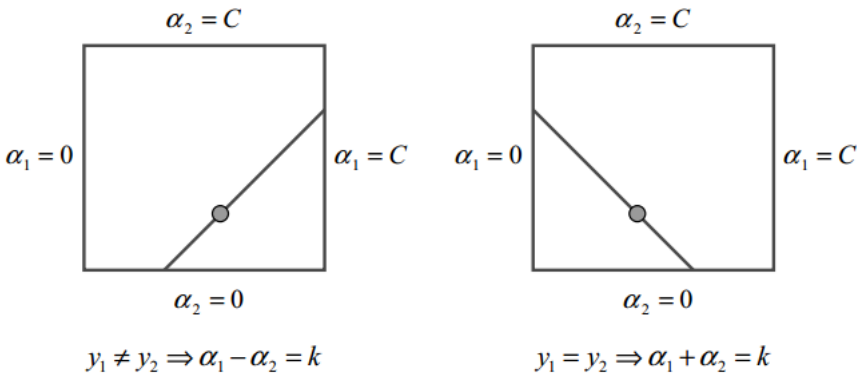


Figure 2.1: The idea of optimizing two multipliers at a time

The advantage of SMO lies in the fact that solving for two lagrange mutipliers can be done analytically. Also, the inner loop of the algorithm is quite short and even though many optimization sub-problems need to be solved in SMO, each sub-problem can be solved very quickly thus leading to the overall QP problem being solved quickly.

There are two components to SMO: an analytic method for solving for the two lagrange multipliers and a heuristic for choosing which multipliers to optimize. Both

these are described in detail in [1] and in the sequential version of the algorithm, we implemented them according to the above.

2.1 Brief overview of the SMO algorithm

We setup the notation first

$$I_0 = \{i : 0 < \alpha_i < C\}$$
$$I_1 = \{i : y_i = 1, \alpha_i = 0\}$$
$$I_2 = \{i : y_i = -1, \alpha_i = C\}$$
$$I_3 = \{i : y_i = 1, \alpha_i = C\}$$
$$I_4 = \{i : y_i = -1, \alpha_i = 0\}$$
$$f_i = \sum_{j=1}^{l} \alpha_j y_j K(x_i, x_j) - y_i$$
$$b_{up} = min\{f_i : i \in I_0 \cup I_1 \cup I_2\},\ b_{low} = max\{f_i : i \in I_0 \cup I_3 \cup I_4\}$$
$$I_{low} = \text{argmax}_i\ f_i$$

The idea is to optimize the two α_i, $\alpha_1 \& \alpha_2$ associated with b_{up} and b_{low} as follows:

$$\alpha_2^{new} = \alpha_2^{old} - \frac{y_2(f_1^{old} - f_2^{old})}{\eta} \tag{2.1}$$
$$\alpha_1^{new} = \alpha_1^{old} + y_1 y_2(\alpha_2^{old} - \alpha_2^{new}) \tag{2.2}$$

where $\eta = 2K(x_1, x_2) - K(x_1, x_1) - K(x_2, x_2)$. α_1^{new} and α_2^{new} need to be clipped so that they lie in $[0, C]$

After optimizing α_1 and α_2, f_i denoting the error on the i-th training instance is updated as follows

$$f_i^{new} = f_i^{old} + (\alpha_1^{new} - \alpha_1^{old}) y_1 K(x_1, x_i) + (\alpha_2^{new} - \alpha_2^{old}) y_2 K(x_2, x_i) \tag{2.3}$$

Based on the updated values of f_i, b_{up} and b_{low} and the associated index I_{up} and I_{low} are updated again according to their definitions. The updated values are then used to choose another two new α_i to optimize at the next step.

In addition, the value of the objective in 1.2 is updated at each step

$$\text{Dual}^{new} = \text{Dual}^{old} - \frac{(\alpha_1^{new} - \alpha_1^{old})}{y_1}(f_1^{old} - f_2^{old}) + \frac{1}{2}\eta(\frac{\alpha_1^{new} - \alpha_1^{old}}{y_1})^2 \tag{2.4}$$

And *DualityGap* representing the difference between the primal and the dual objective function in SVM is calculated as

$$\text{DualityGap} = \sum_{i=0}^{l} \alpha_i y_i f_i + \sum_{i=0}^{l} \epsilon_i \tag{2.5}$$

where

$$\epsilon_i = \begin{cases} C\max(0, b - f_i) & \text{if } y_i = 1 \\ C\max(0, -b + f_i) & \text{if } y_i = -1 \end{cases}$$

SMO Pseudocode

Initialize $\alpha_i = 0,\ f_i = -y_i$, Dual$= 0,\ i = 1, \cdots, l$
Calculate $b_{up},\ I_{up},\ b_{low},\ I_{low}$, DualityGap
repeat
    Optimize $\alpha_{I_{up}},\ \alpha_{I_{low}}$
    Update $f_i,\ i = 1, \cdots, l$
    Calculate $b_{up},\ I_{up},\ b_{low},\ I_{low}$, DualityGap and update Dual
until DualityGap $\leq$ tolerance.|Dual|

Chapter 3

Parallelizing SMO algorithm

In the sequential SMO algorithm, most of the computation time is dominated by the updating of the f_i array as it includes kernel evaluations and is also required for every training example. So the first idea is to parallelize the updating of f_i array. Since the update is performed one training example at a time, what we can do is to partition the entire training data equally into as many partitions as the number of processors. Thus, each processor will update a different subset of the f_i array based on the assigned training data to that processor.

Besides updating f_i array, calculating Calculate b_{up}, b_{low}, I_{up} and I_{low} can also be performed in parallel as the calculation involves examining all the training examples. Each processor could obtain its own b_{up} and b_{low} as well as the associated I_{up} and I_{low} based on training data assigned to it. Note that these are local and not the globally minimum and maximum values. The global b_{up} is the minimum of all the local b_{up} and similarly, the global b_{low} is the maximum of all the local b_{low}. In this manner, the two α_i to optimize can be gotten and then optimized by any one of the processor.

DualityGap is also independently evaluated one training example at a time and hence the same concept as used in the calculation of b_{up} and b_{low} can be used. Each processor calculates a different subset of DualityGap based on the assigned training data subset and the true, global value of the DualityGap is the sum of DualityGap calculated by each processor.

3.1 Implementation

We started off by implementing the serial version of the SMO algorithm as proposed in [4] However, this turned out to be really slow on even small datasets. Hence we implemented an improved heuristic for choosing the α_i to be optimized proposed in [3] The parallel algorithm too used the same optimization so that it would be a fair comparison.

We implemented the parallel SMO using Message Passing Interface (MPI) framework. The "Single Program Multiple Data (SPMD)" mode where different processors execute the same program but use different data is usually used in MPI programs and clearly fits the mentioned ideas for parallelizing.

Chapter 4

Experiments

4.1 Datasets

We assumed a *libsvm* formatted dataset as input where each line is given in a sparse vector format i.e.

```
label feature_id:feature_value feature_id:feature_value ...
```

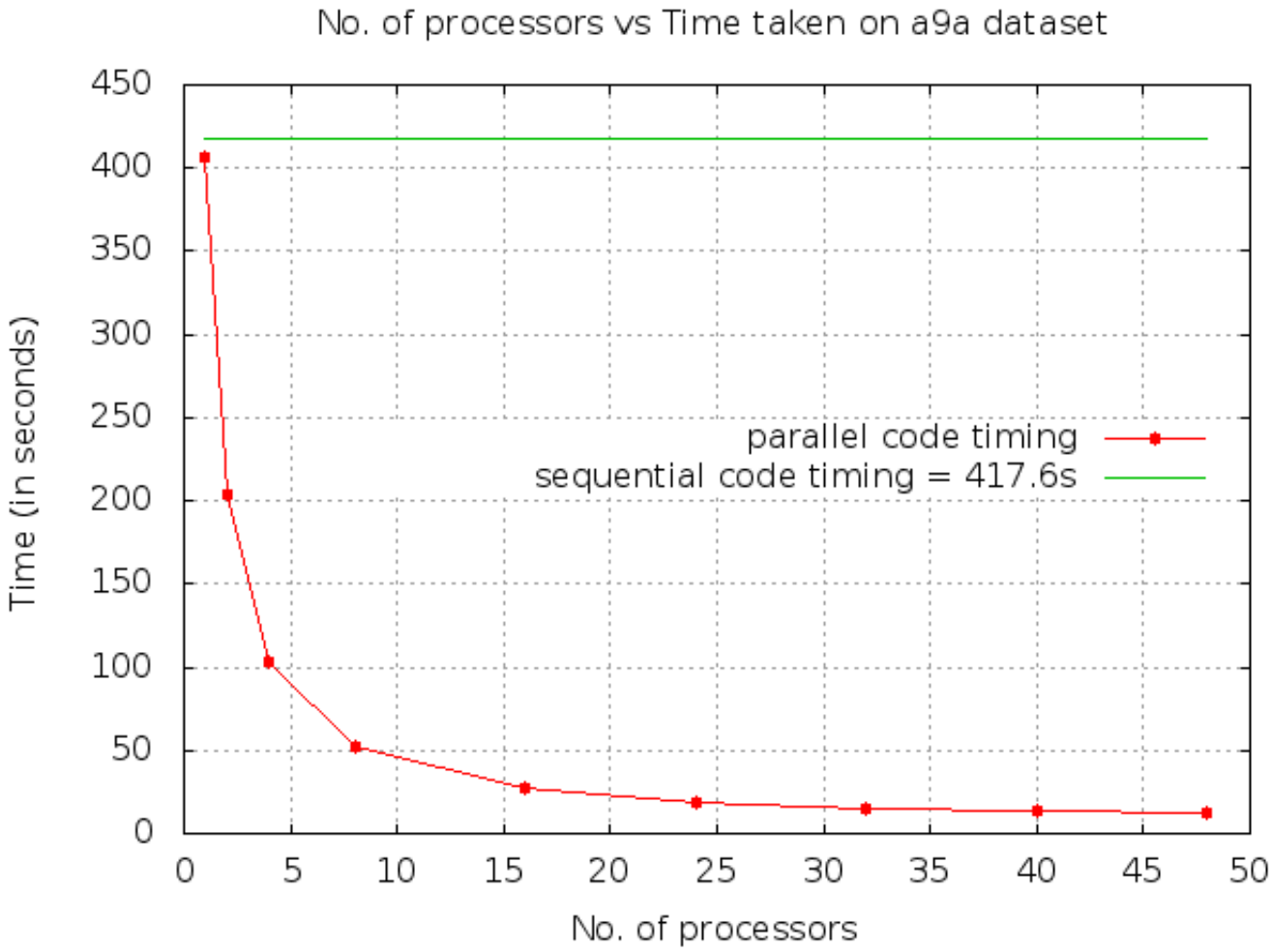
We used the following datasets that are provided on the *libsvm* website [2].

Dataset	# training-examples	# features
a9a	32,561	123
gisette	6000	5000
w8a	49,749	14,951
real-sim	72,309	20,958

4.2 Timing comparison

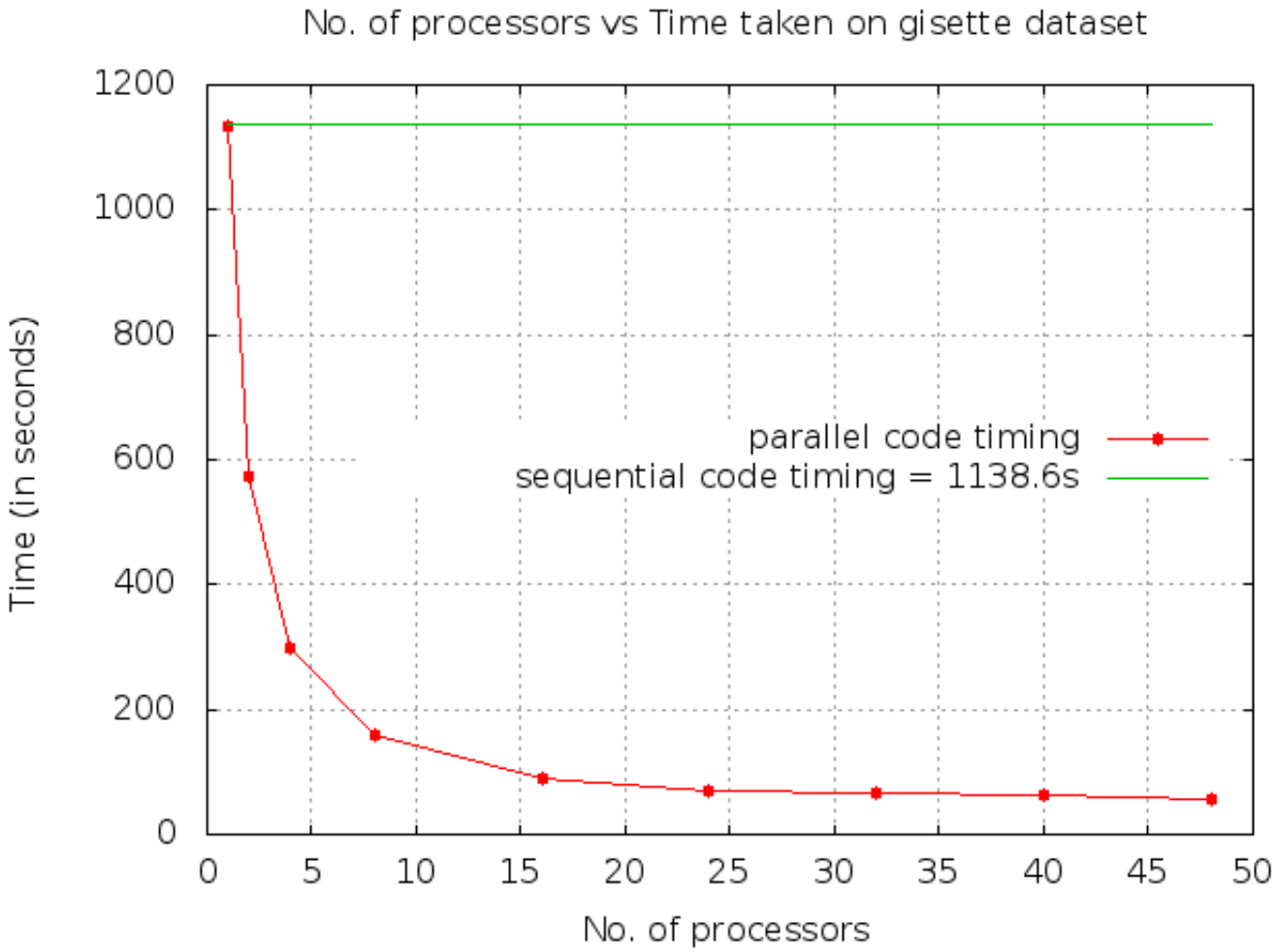
a9a

# processors	1	2	4	8	16	24	32	40	48
Training time (seconds)	406.6	203.9	103.0	51.9	27.1	18.8	15.4	13.7	12.0
Speed-up w.r.t serial code	1.03	2.05	4.05	8.05	15.41	22.21	27.12	30.48	34.80



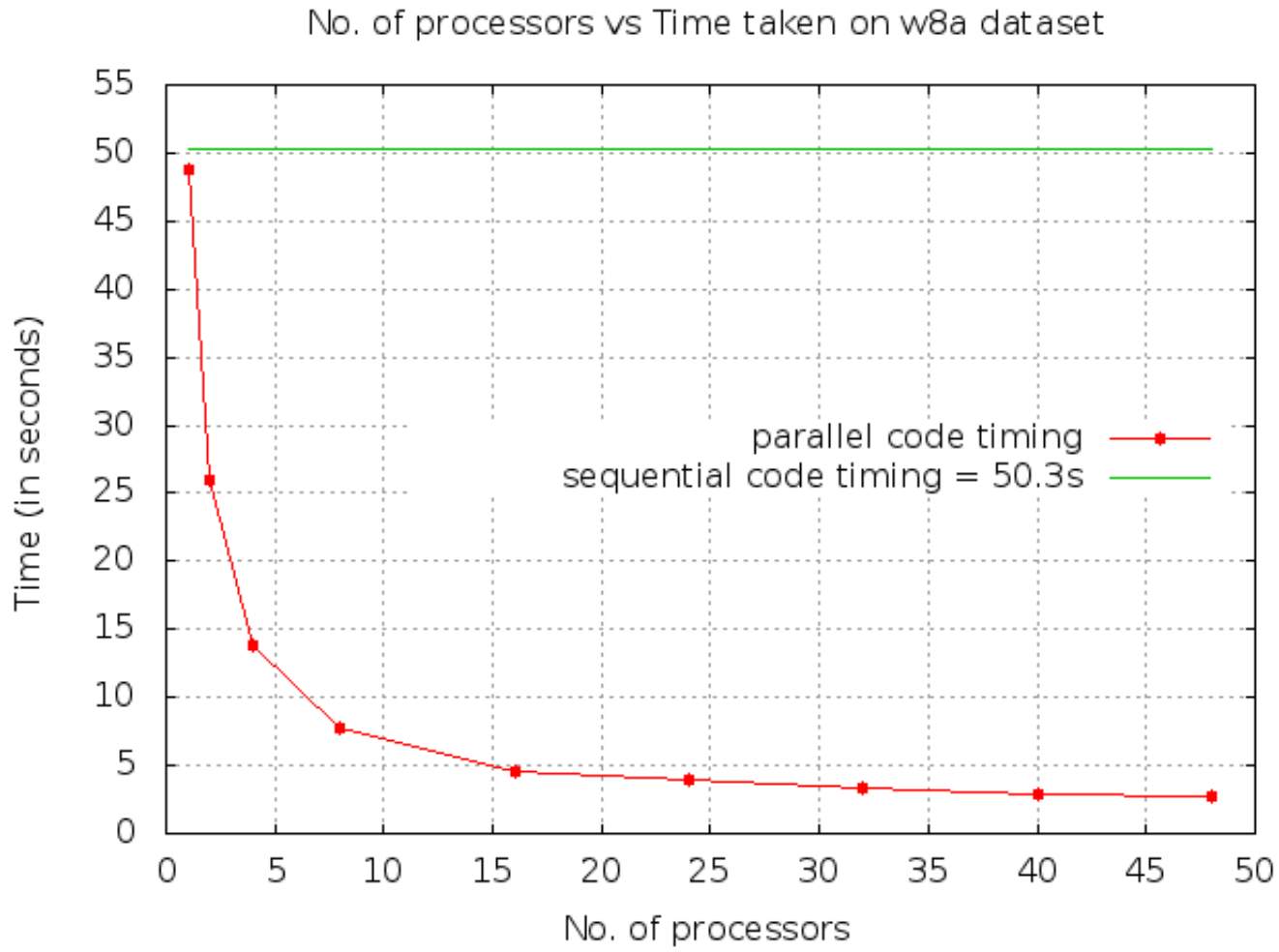
gisette

# processors	1	2	4	8	16	24	32	40	48
Training time (seconds)	1133.1	575.1	297	158.3	89.6	69.2	67.4	62.2	57.5
Speed-up w.r.t serial code	1.00	1.98	3.83	7.19	12.71	16.45	16.89	18.31	19.80



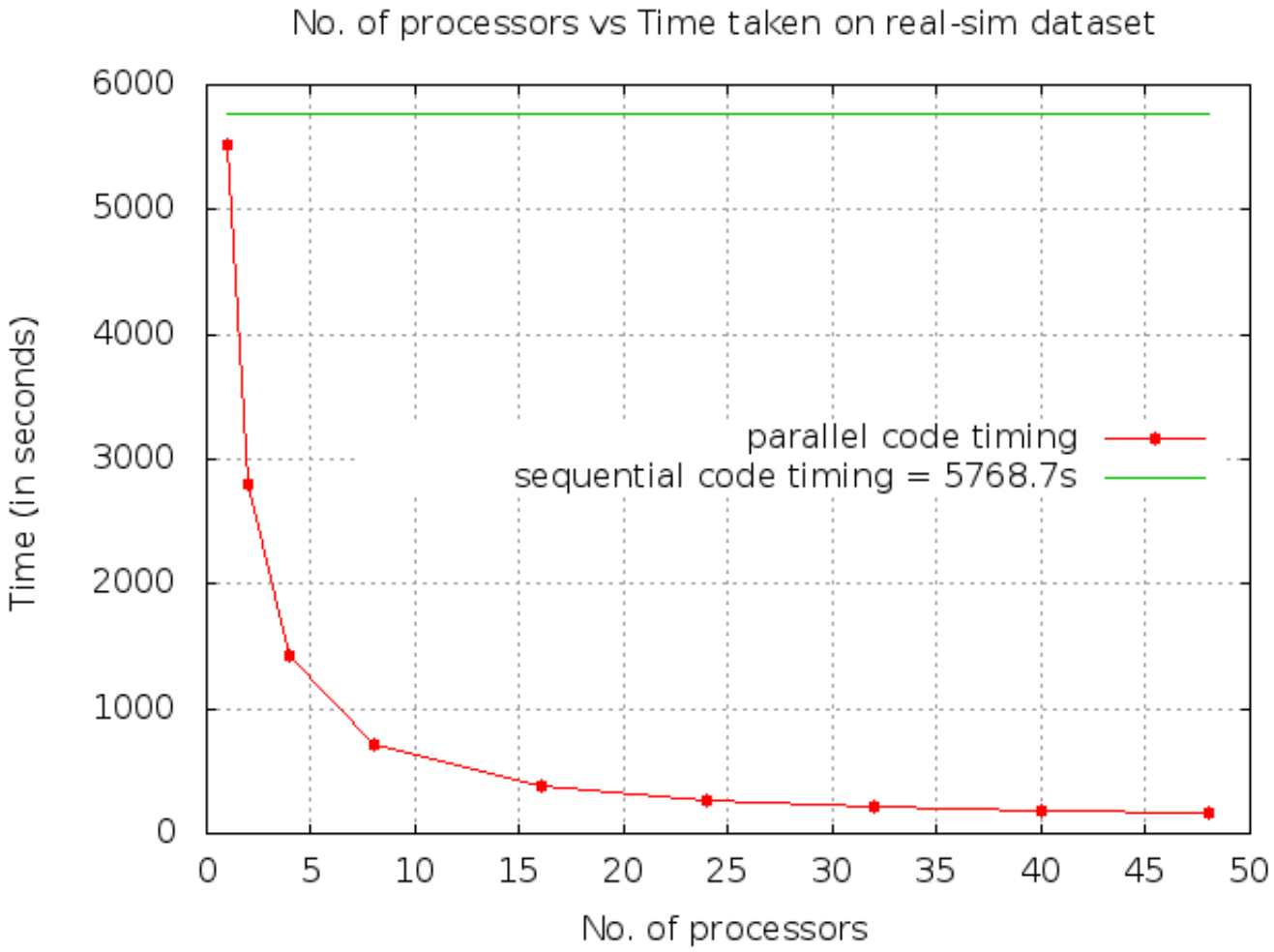
w8a

# processors	1	2	4	8	16	24	32	40	48
Training time (seconds)	48.8	26.0	13.9	7.7	4.5	4.0	3.4	2.9	2.8
Speed-up w.r.t serial code	1.03	1.93	3.62	6.53	11.18	12.57	14.79	17.34	17.96



real-sim

# processors	1	2	4	8	16	24	32	40	48
Training time (seconds)	5514.2	2798.0	1417.9	716.3	382.9	257.3	216.2	183.0	158.4
Speed-up w.r.t serial code	1.05	2.06	4.07	8.05	15.07	22.42	26.68	31.52	36.42



As is evident, we get significant speed-ups using the parallel version of the SMO algorithm.

Bibliography

[1] Li Juan Cao, SS Keerthi, Chong-Jin Ong, JQ Zhang, Uvaraj Periyathamby, Xiu Ju Fu, and HP Lee. Parallel sequential minimal optimization for the training of support vector machines. *Neural Networks, IEEE Transactions on*, 17(4):1039–1049, 2006.

[2] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[3] S. Sathiya Keerthi, Shirish Krishnaj Shevade, Chiranjib Bhattacharyya, and Karuturi Radha Krishna Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Computation*, 13(3):637–649, 2001.

[4] John Platt et al. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.