Decision Support

An incremental least squares algorithm for large scale linear classification

A. Cassioli a,c,*, A. Chiavaiolia, C. Manesb, M. Sciandronec

aUniversità degli Studi di Firenze, Dipartimento di Sistemi e Informatica, Via di S. Marta 3, 50139 Firenze, Italy
bUniversità degli Studi dell’Aquila, Dip. di Ingegneria e Scienze dell’Informazione e Matematica, Via Vetoio, 67100 Coppito (AQ), Italy
cRadiation Oncology - Massachusetts General Hospital and Harvard Medical School, 30 Fruit Street, Boston, MA 02114, USA

A R T I C L E   I N F O

Article history:
Received 5 December 2011
Accepted 2 September 2012
Available online 19 September 2012

Keywords:
Large scale optimization
Machine learning
Linear classification
Incremental algorithms

A B S T R A C T

In this work we consider the problem of training a linear classifier by assuming that the number of data is huge (in particular, data may be larger than the memory capacity). We propose to adopt a linear least-squares formulation of the problem and an incremental recursive algorithm which requires to store a square matrix (whose dimension is equal to the number of features of the data). The algorithm (very simple to implement) converges to the solution using each training data once, so that it effectively handles possible memory issues and is a viable method for linear large scale classification and for real time applications, provided that the number of features of the data is not too large (say of the order of thousands). The extensive computational experiments show that the proposed algorithm is at least competitive with the state-of-the-art algorithms for large scale linear classification.

© 2012 Elsevier B.V. All rights reserved.

1. Introduction

Machine learning models are useful tools for many real life problems and their construction, based on a set of data called the training set, requires very often the solution of optimization problems (as deeply examined in [4,2,18], this latter issue represents the strong interaction of machine learning and mathematical programming).

In classification the training data are labeled, that is, each instance is associated to one of two or more classes. Starting with the training set, the machine learning task for classification is to construct a predictive model able to correctly predict as possible the class of new instances (generalization capability).

Support Vector Machines (SVMs) are widely used as a simple and efficient tool for linear and nonlinear classification (as well as for regression problems). The basic training principle of SVM, motivated by the statistical learning theory [23], is that the expected classification error for unseen test samples is minimized, so that SVM define good predictive models. The design of efficient and reliable optimization methods for SVM training is an active and dynamic research area as pointed by many papers devoted to the topic and published in the last decade (see, e.g., [1,14,13,22,16,17,19,12]).

In this work we focus on linear classification, which is a useful tool in many real applications and represents a very active topic (see, e.g., [9,24,20]). The aim is to develop fast optimization algorithms for training linear classifiers on large scale data (in particular, data may be larger than the memory capacity).

Given a finite set (the training set) of data (not necessarily linearly separable)\[ TS = \{(u^i, d^i) : u^i \in \mathbb{R}^n, \quad d^i \in \{-1, 1\}, \quad i = 1, \ldots, m\}, \]
where the label $d^i$ denotes the class of the vector $u^i$, we consider the problem of training a linear machine defined by the decision function:
\[ y(u) = \text{sgn}(w^T u + \theta), \]
where $u \in \mathbb{R}^n$ is the input vector, $w \in \mathbb{R}^n$ is the vector of weights, $\theta \in \mathbb{R}$ is the threshold, $\text{sgn} : \mathbb{R} \rightarrow \{-1, 1\}$ is the sign function such that $\text{sgn}(t) = \begin{cases} 1 & t \geq 0 \\ -1 & t < 0 \end{cases}$

We assume that the number $m$ of training data is huge.

The standard SVM linear classifier is obtained by computing the solution of the problem:
\[
\min_{w \in \mathbb{R}^n, \theta \in \mathbb{R}, C \in \mathbb{R}^+} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m \xi^i \quad \text{subject to} \quad d^i (w^T u^i + \theta) \geq 1 - \xi^i \quad i = 1, \ldots, m \quad \xi^i \geq 0 \quad i = 1, \ldots, m,
\]
where $\| \cdot \|$ is the Euclidean norm. The term $C > 0$ trades off margin size (related to the generalization capability) and training error, and is usually determined by standard cross-validation tuning procedures.

* Corresponding author. Tel.: +39 055 4796 464.
E-mail addresses: cassioli@dsi.unifi.it, cassioliandre@gmail.com (A. Cassioli), alessandrochiavaiolio@gmail.com (A. Chiavaiolio), costanzo.manes@univaq.it (C. Manes), sciandrono@dsi.unifi.it (M. Sciandrone).

0377-2217/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ejor.2012.09.004
Robustness with respect to the parameter $C$ is a very desirable property that, as reported in Section 4, the classifier proposed in this work seems to possess.

Problem (2), usually referred to as primal problem, is equivalent to a non-smooth unconstrained optimization problem of the form

$$\min_{\textbf{w}, \theta} \frac{1}{2} ||\textbf{w}||^2 + C \sum_{i=1}^{m} \max\{0, 1 - d(\textbf{w}' \textbf{u}^{(i)} + \theta)\}$$

(3)

Using Lagrangian duality, we have that the dual problem of (2) is a convex quadratic problem (whose dimension is equal to the number $m$ of training data) with one linear equality constraint and box constraints. There exist both (primal) algorithms for solving Problem (3) and (dual) algorithms for solving its dual.

A more general primal form is

$$\min_{\textbf{w}, \theta} \rho(\textbf{w}, \theta) + C \sum_{i=1}^{m} \mathcal{Z}(\textbf{w}, \theta, \textbf{u}^{(i)}, \theta),$$

(4)

where $\rho(\textbf{w}, \theta)$ is the regularization term and $\mathcal{Z}(\textbf{w}, \theta, \textbf{u}^{(i)}, \theta)$ is the loss function associated with the observation $(\textbf{u}^{(i)}, \theta)$. There exist algorithms (see, e.g., [6,15,20]) which refer to problem (4) and others (see, e.g., [10,25]) for solving the corresponding dual. For what concerns the loss function $\mathcal{Z}(\textbf{w}, \theta, \textbf{u}^{(i)}, \theta)$, two common forms are

$$\mathcal{Z}(\textbf{w}, \theta, \textbf{u}^{(i)}, \theta) = \max\{0, 1 - d(\textbf{w}' \textbf{u}^{(i)} + \theta)\},$$

$$\mathcal{Z}(\textbf{w}, \theta, \textbf{u}^{(i)}, \theta) = \max\{0, 1 - d(\textbf{w}' \textbf{u}^{(i)} + \theta)\}^2.$$

For the regularization term $\rho(\cdot)$ standard choices are the following

$$\rho(\textbf{w}, \theta) = ||\textbf{w}||^2,$$

$$\rho(\textbf{w}, \theta) = ||\textbf{w}||.$$  

We are assuming that the number $m$ of training data is so huge that either the whole training set cannot be stored in the computer memory or, in the case that memory is enough, loading data to memory can be too expensive. Therefore, we focus on incremental methods which use some observations at a time rather than using the whole training set. We address the reader to [25] for the state of art concerning linear large-scale classification, we only cite two efficient optimization methods for large scale linear classification proposed in [11,5], respectively. The former is a batch method using the whole training set, the latter is based on an incremental strategy. Both methods, as the one proposed in this work are able to solve problems with a huge number of observations having a relatively small number of features.

In this work we introduce a linear classifier obtained by solving a linear least squares problem corresponding to a primal formulation of the form (4), and we adopt an incremental recursive least-squares algorithm as training algorithm. The results of the extensive computational experiments reported in the paper show that the presented methodology (very simple to implement) may be an effective tool for large scale linear classification whenever the number $n$ of features is not huge (say $n$ of the order of thousands).

The paper is organized as follows. In Section 2 we recall the structure of a recursive linear least-squares algorithm. In Section 3 we present the linear classifier and the training algorithm. In Section 4 we report the results of extensive computational experiments performed on large dimensional classification problems and we show the comparison with many other linear classifiers and training algorithms.

2. Recursive linear least-squares algorithm

Let us consider the linear least-squares problem

$$\min_{\textbf{x} \in \mathbb{R}^n} ||A_k \textbf{x} - \textbf{b}_k||^2,$$

(5)

where $A_k \in \mathbb{R}^{k \times n}$ with $\text{rank}(A_k) = n$, $\textbf{b}_k \in \mathbb{R}^k$. The solution $\textbf{x}_k$ of (5) satisfies the normal equations

$$A_k^T A_k \textbf{x} = A_k^T \textbf{b}_k,$$

Assume that a new equation

$$\textbf{a}_{k+1} \textbf{x} = b_{k+1}$$

is added, so that the new problem to be solved is

$$\min_{\textbf{x} \in \mathbb{R}^n} ||A_{k+1} \textbf{x} - b_{k+1}||^2,$$

(6)

where

$$A_{k+1} = \begin{bmatrix} A_k \\ \textbf{a}_{k+1} \end{bmatrix}, \quad \textbf{b}_{k+1} = \begin{bmatrix} \textbf{b}_k \\ b_{k+1} \end{bmatrix}.$$

Letting $H_k^{-1} = (A_k^T A_k)^{-1}$ and denoting by $x_{k+1}$ the solution of (6), we have the recursive updating formula (see, e.g., [3])

$$x_{k+1} = x_k + v_{k+1} (b_{k+1} - a_{k+1}' x_k),$$

(7)

where

$$v_{k+1} = H_{k+1}^{-1},$$

(8)

and

$$H_{k+1}^{-1} = H_k^{-1} - \frac{\textbf{s}_k \textbf{s}_k'}{1 + \textbf{a}_{k+1}' H_k^{-1} \textbf{a}_{k+1}}, \quad \textbf{s}_k = H_k^{-1} \textbf{a}_{k+1}.$$  

(9)

Eqs. (7)-(9) allow us to define an incremental algorithm for a linear least-squares problem

$$\min_{\textbf{x} \in \mathbb{R}^n} ||A \textbf{x} - \textbf{b}||^2,$$

where $A \in \mathbb{R}^{m \times n}$ and $\textbf{b} \in \mathbb{R}^m$ with $m > n$ and $\text{rank}(A) = n$. The algorithm requires to store a positive definite $n \times n$ matrix, and at each iteration updates the estimate of the solution using a single observation. Once initialized $H_1^{-1}$, the algorithm requires exactly $m-n$ iterations for determining the solution of the problem. In Section 3 we will present the training problem of a linear classifier formulated as a linear least-squares problem and the incremental training algorithm, which can be conveniently adopted for solving large-scale problems.

3. The linear classifier and the training algorithm

Consider the problem of designing a linear classifier using a given training set

$$\mathcal{T}(\mathbf{u}, \mathbf{d}) = \{ (\mathbf{u}^i, d^i) : \mathbf{u}^i \in \mathbb{R}^d, d^i \in \{-1, 1\}, i = 1, \ldots, m \}.$$

As introduced in Section 1, a standard linear SVM can be trained by solving the unconstrained non-smooth optimization problem (3). Note that the classification constraints are imposed by means of linear inequalities, and transferred by a penalty term in the objective function.

We consider here the regularized least-squares formulation (see, e.g., [21] as introduction to least-squares Support Vector Machines). In particular, the classification constraints, whose violation must be penalized, are defined by linear equalities, that is,

$$\textbf{w}' \textbf{u}^i + \theta = d^i \quad i = 1, \ldots, m,$$

so that, the linear least-squares formulation corresponding to (4) becomes

$$\min_{\textbf{x} \in \mathbb{R}^n} \frac{1}{2} ||\textbf{x}||^2 + C \sum_{i=1}^{m} (\textbf{w}' \textbf{u}^i + \theta - d^i)^2.$$

(10)
Setting 
\[
\begin{bmatrix}
(u^1)^T \\
\vdots \\
(u^m)^T
\end{bmatrix} \in \mathbb{R}^{m \times n}, \quad d = \begin{bmatrix}
d^1 \\
\vdots \\
d^m
\end{bmatrix} \in \mathbb{R}^m, \quad 1 = \begin{bmatrix}
1 \\
\vdots \\
1
\end{bmatrix} \in \mathbb{R}^m,
\]
we rewrite problem (10) in the following (slightly more general) form
\[
\min_{w \in \mathbb{R}^p; 0 \leq \theta} \| \bar{A}w + \theta 1 - d \|^2 + c_w \| w \|^2 + c_d \theta^2,
\]  
with \(c_w > 0, c_d \geq 0\). The value of the parameter \(c_d\) allows us either to include or not the bias \(\theta\) in the regularization term.

Finally, setting
\[
A = \begin{bmatrix}
\bar{A} & 1_m \\
\sqrt{c_w} I_{n \times n} & 0_n \\
0_n^T & \sqrt{c_d}
\end{bmatrix} \in \mathbb{R}^{(m+1) \times (n+1)}, \quad b = \begin{bmatrix}
d \\
0_{n+1}
\end{bmatrix} \in \mathbb{R}^{m+1},
\]
we obtain the linear least-squares formulation
\[
\min_{x \in \mathbb{R}^{n+1}} \| Ax - b \|^2,
\]  
where
\[
x = \begin{bmatrix}
w \\
\theta
\end{bmatrix}.
\]

We are assuming that the number \(m\) of training data is so huge that either the whole training set cannot be stored in the computer memory or, in the case that memory is enough, loading data to memory can be too expensive. Then, we adopt an incremental approach using one training instance at a time. More specifically, we define the recursive least-squares algorithm based on formulae (7)–(9) for solving problem (13) exploiting the structure (12).

To this aim, we distinguish two cases:

- \(c_w = c_d > 0\): according to the notation of Section 2, we set
  \[
  A_0 = \begin{bmatrix}
  \sqrt{c_w} I_{n \times n} & 0_n \\
  0_n^T & \sqrt{c_d}
  \end{bmatrix}, \quad b_0 = \begin{bmatrix}
  0_{n+1}
  \end{bmatrix}.
  \]  
  and we have
  \[
  H_0 = A_0^T A_0 = c_w I_{(n+1) \times (n+1)},
  \]
  so that we can initialize
  \[
  H_0^{-1} = \frac{1}{c_w} I_{(n+1) \times (n+1)},
  \]  
  \(c_w = 0\): the last row of matrix \(A\) can be dropped, one of the first \(m\) rows (say the first) is considered to define a nonsingular matrix \(A_0\), so that we set
  \[
  A_0 = \begin{bmatrix}
  (u^1)^T \\
  \vdots \\
  (u^m)^T
  \end{bmatrix} \in \mathbb{R}^{m \times n}, \quad b_0 = \begin{bmatrix}
d^1 \\
\vdots \\
d^m
\end{bmatrix} \in \mathbb{R}^m,
  \]
  and we have
  \[
  H_0 = A_0^T A_0 = \begin{bmatrix}
  (u^1)^T & + c_w I_{n \times n} & u^1 \\
  (u^1)^T & + c_d u^1 & 1
  \end{bmatrix},
  \]
  so that we can initialize
  \[
  H_0^{-1} = \frac{1}{c_w} I_{n \times n} - \frac{u^T}{c_d (u^1)^T c_w + (u^1)^T u^1}.
  \]

Once defined the initial matrix \(H_0^{-1}\), according to the incremental strategy underlying the recursive-least squares algorithm of the preceding section, the observations \((u^j d^j)\), i.e. the first \(m\) rows of the matrix \(A\) coupled with the first \(m\) elements of \(b\), are sequentially considered.

We formally define the incremental training procedure, denoted as ILS, in Algorithm 1.

**Algorithm 1.** The incremental least-square training algorithm for binary classification (ILS).

**Input:** The training set \(TS\) and weights \(c_w > 0, c_d \geq 0\)

**Output:** The classifier parameters \(x^T = (w, \theta)^T\)

1. initialize \(H_0^{-1}\) and \(b_0\) (see Eq. (15), (14) or (17) and (16));
2. \(x_0 \leftarrow H_0^{-1} b_0\);
3. \(k \leftarrow 1\);
4. while \(k \leq m\) do
   5. \(a_k^T \leftarrow (u^k, 1)^T\);
   6. \(s_{k-1} \leftarrow H_{k-1}^{-1} a_k^T\);
   7. \(H_k \leftarrow H_{k-1}^{-1} - \frac{1}{s_k} s_k a_k^T\);
   8. \(v_k \leftarrow H_k^{-1} a_k\);
   9. \(x_k \leftarrow x_{k-1} + v_k (d^k - a_k x_{k-1})\);
   10. \(k \leftarrow k + 1\);
11. end

We observe that.

- The regularized least-squares formulation allows us to easily initialize \(H_0^{-1}\) (which depends only on the regularization terms); thus, in order to find the solution of (13), formulae (7)–(9) must be applied one and only one time for each training instance.
- The algorithm determines the solution of (13) in exactly \(m\) iterations, being \(m\) the number of training data, so that it is a direct method and therefore does not need a stopping criterion.
- The computational complexity of the algorithm is \(O(n^2)\) per iteration; more specifically, Steps 6–8 require \(O(n^2)\) operations, and Step 9 requires \(O(n)\) operations.
- Data are used only once, so that the algorithm can effectively handle possible memory issues and eventually reduce disk storage access.
- The algorithm needs to store a \((n+1) \times (n+1)\) matrix, being \(n\) the number of features of the input vectors, and hence \(n\) must be not too high (say of the order of the thousands).
- The algorithm is suitable even for online learning (and hence for real time applications), where the training set is not a priori fixed, but new training instances are continuously generated.
- As evident, the algorithm is very simple to implement; furthermore, the operations required by the algorithm are suitable for an efficient implementation in a parallel computation setting.

Finally, we remark that a nonlinear version of the recursive least-squares algorithm was presented in [8] and designed for nonlinear regression problems. Since the order of the kernel-based regression model would be, in principle, equal to the number of training data, to keep the complexity of the algorithm bounded, the authors proposed a sequential sparsification process to generate the so-called dictionary vectors. In this way, the complexity of the algorithm (in terms of space and time) depends on the number of dictionary vectors. However, it cannot be a priori ensured that the number of dictionary vectors remains necessarily low, and this may be a limit for the application of the algorithm on very large data sets.
4. Computational results

Algorithm 1 has been implemented in C/C++ as an extension of the LibLinear package, one of the most used SVM package at the present time. This choice gives us the possibility to obtain more accurate benchmarks and comparisons because all the tested algorithms share the same code for input/output operations, sparse data representation and linear algebra. We remark that LibLinear is a C/C++ code optimized for sparse data: data storage and linear algebra operations are implemented efficiently for this kind of setting. In implementing Algorithm 1, we have used data sparsity whenever possible.

All tests have been performed on an Intel i5 standard desktop machine with 2 giga bytes RAM, running Linux and compiled with gcc. In each test we consider \(\mathbf{c}_w = \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}\}\) and \(\mathbf{c}_p = \{0, \mathbf{c}_w\}\). Concerning LibLinear, in each test we consider the corresponding value \(C\) given by

\[
C = \frac{1}{\mathbf{c}_w}.
\]

Tests are divided in two parts: in Section 4.1 we report the results obtained on test problems whose dimensions are such that the data can fit in memory, and standard training algorithms can be applied; in Section 4.2 we show the results obtained on problems whose data cannot fit in memory, so that block minimization algorithms must be used.

4.1. Experiments on data sets that can fit in memory

A first test has been performed on standard data sets for binary classification available at http://www.csie.ntu.edu.tw/cjlin/libsvmtools/datasets/ and listed in Table 1. All data sets have been normalized with the scale utility to ease numerical problems. These data sets have a limited number of features but a considerable number of observations, still small enough to be handled by standard implementations.

We compare the proposed classifier (ILS) against all the classification algorithms available in LibLinear:

- L2-regularized L2-loss dual (L2L2D).
- L2-regularized L2-loss primal (L2L2P).
- L1-regularized L1-loss dual (L1L1D).
- L1-regularized L2-loss primal (L1L2P).

Results reported in Table 2–5 show the CPU time in seconds and accuracy on the training set. All tests have been performed with default parameters.

We may observe that:

- The two versions of ILS (with and without the bias in the objective function) are equivalent both in terms of CPU time and of accuracy, so the presence or not of the bias \(\theta\) in the regularization term does not seem significant.
- As expected, the CPU time required by ILS does not depend on the value of the regularization parameter \(c_p\).
- For each of the four test problems, the accuracy obtained by ILS is the same for almost all the tried values of the parameter \(c_p\).
- All the algorithms can be considered substantially equivalent in terms of accuracy attained.
- The best performance in terms of CPU time are obtained by L2L2P and ILS, both primal algorithms; the differences with respect to the performance of the other algorithms are quite evident for the larger test problems, i.e., covtype and cod-rna.

Then, on the basis of the results of the experiments, we may conclude that on the whole the proposed algorithm shows a good robustness with respect to its parameters (and this is an important...

<table>
<thead>
<tr>
<th>Table 1 Data sets from the LibSVM repository.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data set name</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>a9a</td>
</tr>
<tr>
<td>cod-rna</td>
</tr>
<tr>
<td>ijcnn1</td>
</tr>
<tr>
<td>covtype</td>
</tr>
</tbody>
</table>

| Table 2 Results on the a9a data set (execution time in seconds/accuracy). |
|-----------------------------|-------------------|
| C L2L2D L2L2P L1L1D L1L2P ILS (\(c_0 = 0\)) ILS (\(c_0 = c_w\)) |
|-----------------------------|-------------------|
| \(10^{-9}\) 0.11/ 0.11/ 0.10/ 0.10/ 0.10/ 0.10/ 0.10/ 0.10/ 0.6/ 0.6/ |
| \(10^{-3}\) 0.16/ 0.16/ 0.14/ 0.14/ 0.14/ 0.14/ 0.14/ 0.14/ 0.6/ 0.6/ |
| \(10^{-1}\) 0.24/ 0.24/ 0.19/ 0.19/ 0.19/ 0.19/ 0.19/ 0.19/ 0.6/ 0.6/ |
| \(10^{0}\) 0.98/ 0.98/ 0.88/ 0.88/ 0.88/ 0.88/ 0.88/ 0.88/ 0.5/ 0.5/ |
| \(10^{3}\) 0.80/ 0.80/ 0.61/ 0.61/ 0.61/ 0.61/ 0.61/ 0.61/ 0.5/ 0.5/ |
| \(10^{4}\) 0.69/ 0.69/ 0.52/ 0.52/ 0.52/ 0.52/ 0.52/ 0.52/ 0.4/ 0.4/ |
| \(10^{5}\) 0.59/ 0.59/ 0.43/ 0.43/ 0.43/ 0.43/ 0.43/ 0.43/ 0.3/ 0.3/ |

| Table 3 Results on the cod-rna data set (execution time in seconds/accuracy). |
|-----------------------------|-------------------|
| C L2L2D L2L2P L1L1D L1L2P ILS (\(c_0 = 0\)) ILS (\(c_0 = c_w\)) |
|-----------------------------|-------------------|
| \(10^{-9}\) 0.20/ 0.20/ 0.19/ 0.19/ 0.19/ 0.19/ 0.19/ 0.19/ 0.17/ 0.17/ |
| \(10^{-3}\) 6.67/ 6.67/ 6.67/ 6.67/ 6.67/ 6.67/ 6.67/ 6.67/ 6.67/ 6.67/ |
| \(10^{-1}\) 0.27/ 0.27/ 0.22/ 0.22/ 0.22/ 0.22/ 0.22/ 0.22/ 0.18/ 0.18/ |
| \(10^{0}\) 0.93/ 0.93/ 0.89/ 0.89/ 0.89/ 0.89/ 0.89/ 0.89/ 0.89/ 0.89/ |
| \(10^{3}\) 0.85/ 0.85/ 0.80/ 0.80/ 0.80/ 0.80/ 0.80/ 0.80/ 0.80/ 0.80/ |
| \(10^{4}\) 0.64/ 0.64/ 0.59/ 0.59/ 0.59/ 0.59/ 0.59/ 0.59/ 0.59/ 0.59/ |

| Table 4 Results on the ijcnn1 data set (execution time in seconds/accuracy). |
|-----------------------------|-------------------|
| C L2L2D L2L2P L1L1D L1L2P ILS (\(c_0 = 0\)) ILS (\(c_0 = c_w\)) |
|-----------------------------|-------------------|
| \(10^{-9}\) 0.25/ 0.25/ 0.26/ 0.26/ 0.26/ 0.26/ 0.26/ 0.26/ 0.25/ 0.25/ |
| \(10^{-3}\) 0.32/ 0.32/ 0.27/ 0.27/ 0.27/ 0.27/ 0.27/ 0.27/ 0.26/ 0.26/ |
| \(10^{-1}\) 0.92/ 0.92/ 0.92/ 0.92/ 0.92/ 0.92/ 0.92/ 0.92/ 0.91/ 0.91/ |
| \(10^{0}\) 0.29/ 0.29/ 0.28/ 0.28/ 0.28/ 0.28/ 0.28/ 0.28/ 0.28/ 0.28/ |
| \(10^{3}\) 0.92/ 0.92/ 0.92/ 0.92/ 0.92/ 0.92/ 0.92/ 0.92/ 0.91/ 0.91/ |
| \(10^{4}\) 0.14/ 0.14/ 0.14/ 0.14/ 0.14/ 0.14/ 0.14/ 0.14/ 0.14/ 0.14/ |
| \(10^{5}\) 0.88/ 0.88/ 0.88/ 0.88/ 0.88/ 0.88/ 0.88/ 0.88/ 0.88/ 0.88/ |

1 Freely available at http://www.csie.ntu.edu.tw/cjlin/liblinear/.
We report in Tables 7–11 the results in terms of elapsed real time provided by used for comparison both in terms of CPU time and of obtained accuracy whenever the number of features is of the order of hundreds, as in alpha, delta, gamma, epsilon and zeta data sets. The former value has been reported because the I/O operations are a crucial element in block training and may represent a considerable part of the running time.

From the reported results we get that the performance of ILS algorithm are clearly better than those of the other algorithms used for comparison both in terms of CPU time and of obtained accuracy whenever the number of features is of the order of hundreds, as in alpha, delta, gamma, epsilon and zeta data sets (see Tables 7, 9, 10).

Concerning the results on the other two test problems, where the number of features is of the order of thousands, we may observe that.

Table 6

<table>
<thead>
<tr>
<th>Name</th>
<th>#Training</th>
<th>#Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>500000</td>
<td>500</td>
</tr>
<tr>
<td>delta</td>
<td>500000</td>
<td>500</td>
</tr>
<tr>
<td>gamma</td>
<td>500000</td>
<td>500</td>
</tr>
<tr>
<td>epsilon</td>
<td>500000</td>
<td>2000</td>
</tr>
<tr>
<td>zeta</td>
<td>500000</td>
<td>2000</td>
</tr>
</tbody>
</table>

Table 7

<table>
<thead>
<tr>
<th>C</th>
<th>L2D</th>
<th>L1D</th>
</tr>
</thead>
<tbody>
<tr>
<td>10^0</td>
<td>227.01/94.38</td>
<td>431.45/420.96</td>
</tr>
<tr>
<td>10^1</td>
<td>1765.59</td>
<td>428.17/418.20</td>
</tr>
<tr>
<td>10^2</td>
<td>1849.50</td>
<td>435.57/419.32</td>
</tr>
</tbody>
</table>

Table 8

<table>
<thead>
<tr>
<th>C</th>
<th>L2D</th>
<th>L1D</th>
</tr>
</thead>
<tbody>
<tr>
<td>10^0</td>
<td>329.92/83.88</td>
<td>12221.72/12305.94</td>
</tr>
<tr>
<td>10^1</td>
<td>1779.36</td>
<td>12504.54/12403.86</td>
</tr>
<tr>
<td>10^2</td>
<td>1873.79</td>
<td>12210.10/12404.27</td>
</tr>
</tbody>
</table>

Table 9

<table>
<thead>
<tr>
<th>C</th>
<th>L2D</th>
<th>L1D</th>
</tr>
</thead>
<tbody>
<tr>
<td>10^0</td>
<td>177.76/72.80</td>
<td>425.01/425.41</td>
</tr>
<tr>
<td>10^1</td>
<td>1881.09</td>
<td>422.18/421.27</td>
</tr>
<tr>
<td>10^2</td>
<td>1820.23</td>
<td>425.29/423.64</td>
</tr>
</tbody>
</table>

4.2. Experiments with large data sets that cannot fit in memory

We perform a second batch of tests focusing on the situation in which the data set at hand cannot be stored in the working memory; in this case data must be split in blocks of smaller size (denoted as chunks) stored into files to be accessed. Then, we deal with the so-called block training for which, as explained in [25], “a viable method must satisfy the following conditions:

1. Each optimization step reads a contiguous chunk of training data.
2. The optimization procedure converges toward the optimum even though each step uses only a subset of training data.
3. The number of optimization steps (iterations) should not be too large. Otherwise, the same data point may be accessed from the disk too many times.”

As already explained, the proposed classifier has all these properties, so that it directly extends to the block training. In particular, we use the same code as in Section 4.1, with minor modifications, in LibLinear specialized version2.

Data sets have been taken from the ones proposed for the Pascal Large Scale Learning Challenge 20083, whose details are summarized in Table 6. Each data set has been split, using the blocksplit tool provided by LibLinear, in 32 blocks.

In this case we compare the proposed classifier against the two available in the block training version of LibLinear:

- L2-loss dual formulation (L2D).
- L1-loss dual formulation (L1D).

Both algorithms have been executed with default parameters. We report in Tables 7–11 the results in terms of elapsed real time (i.e. including the I/O operations), CPU time and accuracy on the training set. The former value has been reported because the I/O operations are a crucial element in block training and may represent a considerable part of the running time.

From the reported results we get that the performance of ILS algorithm are clearly better than those of the other algorithms used for comparison both in terms of CPU time and of obtained accuracy whenever the number of features is of the order of hundreds, as in alpha, delta, gamma, epsilon and zeta data sets.

Concerning the results on the other two test problems, where the number of features is of the order of thousands, we may observe that.

---

3 Freely available at http://largescale.ml.tu-berlin.de/instructions/.

References:

- [http://largescale.ml.tu-berlin.de/instructions/](http://largescale.ml.tu-berlin.de/instructions/)
its parameters in each test problem, with the exception of at most of the order of thousands, and this represents its main limitation. The extensive numerical experiments performed have shown that the proposed method is quite robust with respect to its parameters and is at least competitive with the state-of-the-art algorithms for large scale linear classification. Future work will be devoted to extend the approach to train Radial Basis Function neural networks for tackling large-scale nonlinear classification/regression problems.

References


