Large Linear Classification When Data Cannot Fit In Memory

Hsiang-Fu Yu
Department of Computer Science
National Taiwan University

Joint work with C.-J. Hsieh, K.-W. Chang, and C.-J. Lin
July 27, 2010
Outline

- Introduction
- A Block Minimization Framework for Linear SVMs
- Implementations for SVM
- Techniques to Reduce the Training Time
- Other Functionalities
- Experiments
- Conclusions
Outline

- Introduction
  - A Block Minimization Framework for Linear SVMs
  - Implementations for SVM
  - Techniques to Reduce the Training Time
  - Other Functionalities
  - Experiments
  - Conclusions
Recently linear classification is a popular research topic.

- By linear we mean that kernel is not used.
- Though linear may not be as good as nonlinear.
- For some problems: accuracy by linear is as good as nonlinear, and training and testing are much faster.
- This talk addresses on large linear classification.
Motivation

Existing approaches for large linear classification:

- Data smaller than memory:
  Efficient methods are well-developed
- Data beyond disk size:
  Usually handled in a distributed way

Can we have something in the between?

- A simple setting
  memory < data < disk

- Ferris and Munson (2003) proposed a method,
  but only for data with # features ≪ # instances
When Data Cannot Fit In Memory

LIBLINEAR on a machine with 1 GB memory:

Disk swap causes lengthy training time
The Goal

Goal: construct large linear classifiers for ordinary users on a single machine

Assumptions
- memory < data < disk
- Sub-sampling causes lower accuracy

Requirement: must be simple so that it supports
- Multi-class classification
- Parameter selection,
- Other functionalities
Modeling the Training Time

\[
\text{train time} = \text{time to train in-memory data} + \text{time to access data from disk}
\]

- Now need to pay attention to the second part
- Loading time may dominate the training time even data can fit in memory
- 

```bash
> ./liblinear-1.51/train rcv1_test.binary
rcv1_test.binary: > half millions of documents
Loading time: > 1 minute
Computing time: < 5 seconds
```
Conditions for a Viable Method

1. Each optimization step reads a *continuous* chunk of training data.
2. The optimization procedure *converges* toward the optimum.
3. The number of optimization steps should not be too large.
Linear SVM as the Linear Classifier

We consider SVM as the linear classifier

- Training data \( \{(y_i, x_i)\}_{i=1}^l, x_i \in \mathbb{R}^n, y_i = \pm 1 \)
- \( n \): # of features, \( l \): # of data
- Primal SVM:
  \[
  \min_w \frac{1}{2} w^T w + C \sum_{i=1}^l \max(0, 1 - y_i w^T x_i)
  \]
- Dual SVM:
  \[
  \min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - \mathbf{e}^T \alpha
  \]
  subject to \( 0 \leq \alpha_i \leq C, \forall i \),

- \( \mathbf{e} = [1, \ldots, 1]^T, Q_{ij} = y_i y_j x_i^T x_j \)
- \( \alpha \in \mathbb{R}^l \), each \( \alpha_i \) corresponds to \( x_i \)
Outline

- Introduction
- A Block Minimization Framework for Linear SVMs
- Implementations for SVM
- Techniques to Reduce the Training Time
- Other Functionalities
- Experiments
- Conclusions
Algorithm 1

1. Split \( \{1, \ldots, l\} \) to \( B_1, \ldots, B_m \) such that \( B_i \) fits in memory, and store data into \( m \) files accordingly.
2. Set initial \( \alpha \) or \( w \)
3. For \( k = 1, 2, \ldots \) (outer iteration)
   For \( j = 1, \ldots, m \) (inner iteration)
   (a) Read \( x_r, \forall r \in B_j \) from disk
   (b) Conduct operations on \( \{x_r \mid r \in B_j\} \)
   (c) Update \( \alpha \) or \( w \)

Here we do not specify operations on each block
Block Minimization

A classical optimization method

- Block of variables
- Widely used in nonlinear SVM
- Here need a connection between a **block of data** and a **block of variables**

In the situation, data $\succ$ memory

- to avoid random access on the disk
- **cannot** use holistic methods/flexible ways to select block variables

$B_1, \ldots, B_m$: fixed partition of $\{1, \ldots, l\}$
Number of Blocks and Block Size

How to decide $m$ (# of blocks)
- Assume all blocks have similar size $|B|$
- # blocks: $m = \frac{l}{|B|}$

Block size
- Cannot be too large: each $B_j$ must fit in memory
- Cannot be too small: should be as large as possible

Total time for an outer iteration:
$$(T_m(|B|) + T_d(|B|)) \times \frac{l}{|B|} \quad m = \frac{l}{|B|}$$

- $T_m(|B|)$: time cost of one inner iteration in memory
- $T_d(|B|)$: time cost of reading $B$ from disk
- Both $T_m(|B|)$ and $T_d(|B|)$ are functions of $|B|$
Block Size Should Be Large

Total time for an outer iteration:

\[(T_m(|B|) + T_d(|B|)) \times \frac{l}{|B|}\]

Past, \(T_m(|B|)\) only: \(T_m(|B|)\) more than linear to \(|B|\)

- Total time = \(T_m(|B|) \times \frac{l}{|B|}\)

- previous SVM works: smaller \(|B|\) is better

Now, \(T_d(|B|)\) added: \(T_d(|B|)\): initial cost + \(O(|B|)\)

- Total reading time = initial cost \(\times \frac{l}{|B|}\) + \(O(1)\)

- Larger \(|B|\) is better (but can’t exceed memory)
Outline

- Introduction
- A Block Minimization Framework for Linear SVMs
- Implementations for SVM
- Techniques to Reduce the Training Time
- Other Functionalities
- Experiments
- Conclusions
Sub-problem for Dual SVM

Let $f(\alpha)$ be the dual function:

$$f(\alpha) \equiv \frac{1}{2} \alpha^T Q \alpha - e^T \alpha$$

Each block of variables corresponds to a block of data

$$\min_{d_{B_j}} f(\alpha + d)$$

s.t. $d_{\bar{B}_j} = 0$ and $0 \leq \alpha_i + d_i \leq C$, $\forall i \in B_j$

- $\bar{B}_j = \{1, \ldots, l\} \backslash B_j$; only $\alpha_{B_j}$ is changed
- (1) involves all data; handled by some techniques (details omitted)
Algorithm 2 A special case of Algorithm 1

1. Split \( \{1, \ldots, l\} \) to \( B_1, \ldots, B_m \) and store data to \( m \) files
2. Set initial \( \alpha \) and \( w \)
3. For \( k = 1, 2, \ldots \) (outer iteration)
   For \( j = 1, \ldots, m \) (inner iteration)
   (a) Read \( x_r, \forall r \in B_j \) from disk
   (b) Approximately solve the sub-problem to obtain \( d_{B_j}^* \).
   (c) Update \( \alpha_{B_j} \leftarrow \alpha_{B_j} + d_{B_j}^* \) and \( w \)
Any bound-constrained method can be used
- We consider LIBLINEAR: a coordinate descent method

**Two-level** block minimization
- Used in some algorithms (e.g., Memisevic, 2006; Pérez-Cruz et al., 2004; Rüping, 2000)

But here \textbf{inner} ⇒ memory, \textbf{outer} ⇒ disk

- An approximate solution for the sub-problem in practice

Sub-problem \textbf{stopping criterion} and \textbf{convergence} are issues
Sub-problem Stopping Condition and Overall Convergence

Two approaches

1. A fixed number of passes to all variables in $B_j$
   Need to decide the number of passes

2. Gradient-based stopping condition
   Easy to know how accurate the sub-problem’s solution is; we use the one in LIBLINEAR

Convergence holds for both conditions (details omitted)
Let $f^P$ be the primal function

$$f^P(w) = \frac{1}{2} w^T w + C \sum_{i=1}^{l} \max(0, 1 - y_i w^T x_i)$$

A block of primal variable $w$

- corresponds to a subset of features
- no connection to a block of data

Stochastic gradient descent (SGD) approach

- For each update only a block of data is needed
- We use Pegasos (Shalev-Shwartz et al., 2007)
Algorithm 3 A special cases of Algorithm 1

1. Split \{1, \ldots, \ell\} to \(B_1, \ldots, B_m\) and store data into \(m\) files accordingly.
2. \(t = 0\) and initial \(w = 0\)
3. For \(k = 1, 2, \ldots\)
   For \(j = 1, \ldots, m\)
   (a) Find a partition of \(B_j\): \(B_j^1, \ldots, B_j^\bar{r}\)
   (b) For \(r = 1, \ldots, \bar{r}\)
       (b.1) Apply Pegasos update on \(B_j^r\)
       (b.2) \(t \leftarrow t + 1\)

- \(\bar{r} = 1\): only one update on the whole block
- \(\bar{r} = |B|\): \(|B|\) updates, one for each data instance
Outline

- Introduction
- A Block Minimization Framework for Linear SVMs
- Implementations for SVM
- Techniques to Reduce the Training Time
- Other Functionalities
- Experiments
- Conclusions
Techniques To Reduce the Training Time

Data compression for disk reading time $T_d(|B|)$

- Except initial time, $T_d(|B|) \propto \text{data size } |B|$
- Data compression effectively reduces the disk reading time (Details not shown)

Initial Split of Data

- If original data ordered by labels
  a whole block with same label
  \( \Rightarrow \) slow convergence
- A random split is needed
Techniques To Reduce the Training Time

Two tasks in the beginning:
- random split
- data compression

Data > memory:
- avoid re-reading data from disk
- A carefully design ensures
  Random split + data compression by going data only once
Outline

• Introduction
• A Block Minimization Framework for Linear SVMs
• Implementations for SVM
• Techniques to Reduce the Training Time
• Other Functionalities
• Experiments
• Conclusions
Due to the simplicity and block design, we can support
  - Cross validation
  - Multi-class classification
  - Incremental/Decremental setting

Details omitted here.
Outline

- Introduction
- A Block Minimization Framework for Linear SVMs
- Implementations for SVM
- Techniques to Reduce the Training Time
- Other Functionalities
- Experiments
- Conclusions
### Data and Environment

<table>
<thead>
<tr>
<th>Data set</th>
<th>$l$ (data)</th>
<th>$n$ (features)</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>yahoo-korea</td>
<td>460,554</td>
<td>3,052,939</td>
<td>2.5GB</td>
</tr>
<tr>
<td>webspam</td>
<td>350,000</td>
<td>16,609,143</td>
<td>20.8GB</td>
</tr>
<tr>
<td>epsilon</td>
<td>500,000</td>
<td>2,000</td>
<td>16.0GB</td>
</tr>
</tbody>
</table>

- 64-bit machine with **1 GB RAM**
- Data 20 times larger
Compared Methods

**BLOCK-⋆-⋆**: Block minimization methods.

1. **BLOCK-L-N**: Solving dual. LIBLINEAR goes through each block $N$ rounds; $N = 1, 10, 20$.
2. **BLOCK-L-D**: Solving dual. LIBLINEAR default stopping condition for each block.
3. **BLOCK-P-B**: Solving primal. Pegasos on each whole block (one update).
4. **BLOCK-P-I**: Solving primal. Pegasos on each data instance of the block ($|B|$ updates).
5. **LIBLINEAR**: The standard LIBLINEAR without any modification.
Function Value Reduction

webspam

yahoo-korea

Time for initial block split
Function Value Reduction

Proposed methods are faster than LIBLINEAR
Function Value Reduction

webspam

Magnified view

BLOCK-P-⋆ are worse than BLOCK-L-⋆
BLOCK-P-* are worse than BLOCK-L-*

BLOCK-P-B: applies only one update on each block

Information of a block underutilized
Due to long reading time: put more effort on each block.
**Other Experimental Results**

**Random split vs. Raw**

- **raw**: Data are ordered according to labels
- **random split**: Initial random split

---

**Diagram Description**

- The graph compares the relative function value difference over time for raw and random split datasets.
- The x-axis represents time in seconds, ranging from 2000 to 3000.
- The y-axis shows the relative function value difference, ranging from 0.0 to 1.0.
- The graph includes two lines, one for the raw data and another for the random-split data.

---

**Additional Notes**

- Hsiang-Fu Yu (National Taiwan Univ.)
- June 02, 2010
Random split vs. Raw

Random split is useful
Other Experimental Results

Random split vs. Raw

Different block size

\[ m \text{: # of blocks } \Rightarrow \text{smaller } m; \] should use larger \(|B|\)
Outline

- Introduction
- A Block Minimization Framework for Linear SVMs
- Implementations for SVM
- Techniques to Reduce the Training Time
- Other Functionalities
- Experiments
- Conclusions
Conclusions

- We have proposed methods to effectively handle data 20 times larger than memory.
- Our implementation is available at:
  [http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/#large_linear_classification_when_data_cannot_fit_in_memory](http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/#large_linear_classification_when_data_cannot_fit_in_memory)
- You can now train pretty large data on your laptop.