Distributed Box-Constrained Quadratic Optimization for Dual Linear SVM

Lee, Ching-pei
University of Illinois at Urbana-Champaign

Joint work with Dan Roth
ICML 2015

Outline

Introduction

Algorithm

Experiments

Discussions and Conclusions

Distributed Linear Classification

- Distributed linear classification is the essential technique for dealing with data larger than the capacity of a single machine.
- In distributed training, computation and data I/O time are usually short.
- But communication and syncronization costs become the bottleneck.
- We thus need to consider methods requiring fewer rounds of communication. That is, methods that converge faster in terms of iterations.

Linear SVM

- We solve dual problems.
- We use the dual problem of linear support vector machines (SVM) by Boser et al. (1992); Vapnik (1995) as an example for illustration.
- The paper also covers squared-hinge loss (L2-loss) SVM (skipped).
- Can also be easily extended to other problems like linear regression, multi-class classification, structured SVM (Lee et al., 2015).

Why Dual?

Why Dual?

- Because primal methods don't have convergence guarantee

"Current batch primal methods, such as L-BFGS, come with no convergence guarantees even in the serial case......" — Martin Jaggi, Virginia Smith, Martin Takáč, Jonathan Terhorst, Thomas Hofmann, and Michael I. Jordan. In NIPS 2014 author rebuttal of Jaggi et al. (2014). `http://media.nips.cc/nipsbooks/nipspapers/paper_files/nips27/reviews/1588.html`

Why Dual?

- Single machine experience: dual methods are sometimes faster, especially when #features > #instances.
- Some problems are hard to directly solve the primal.

Dual Problem

- Given the binary-labeled training instances $\{(\mathbf{x}_i, y_i) \in \mathbf{R}^n \times \{-1, 1\}\}_{i=1}^{\ell}$, linear SVM dual problem solves

$$\begin{aligned} \min_{\alpha} \quad & f(\boldsymbol{\alpha}) \equiv \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \mathbf{e}^T \boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \le \boldsymbol{\alpha} \le C\mathbf{e}, \end{aligned}$$

  where $\mathbf{e} \in \mathbf{R}^n$ is the vector of ones, $Q_{i,j} = y_i y_j \mathbf{x}_i^T \mathbf{x}_j$, and $C > 0$ is a parameter specified by user.
- Convert the dual solution to the primal model $\mathbf{w}$ by $\mathbf{w} = \sum_{i=1}^{\ell} y_i \mathbf{x}_i \alpha_i$.

Distributed Environment

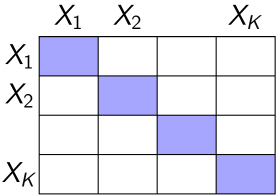
- We solve SVM dual in a distributed environment, where the training instances are disjointly stored among K machines.

partition 1 $\longrightarrow$	(X_1, Y_1)
partition 2 $\longrightarrow$	(X_2, Y_2)
	$\cdots$
partition $K \longrightarrow$	(X_K, Y_K)

- X_i are data matrices: each row is an instance. Y_i are diagonal matrices representing the labels.
- Dual variables: $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_K)$.
- Then $\mathbf{w} = \sum_{i=1}^{\ell} y_i \mathbf{x}_i \alpha_i = \sum_{k=1}^{K} Y_k X_k \boldsymbol{\alpha}_k$

Challenges of Distributed Dual SVM

- Each node only gets partial information of the data.
- To avoid frequent communication, can only use block-diagonal approximation of Q because data are not available.



- Existing dual methods provide only sub-linear convergence rates.

Our Contributions

- We first establish global linear convergence, i.e., $O(\log(1/\epsilon))$ iteration complexity, for distributedly solving non-strongly-convex dual problems like the SVM dual.
- The key factor is a communication-efficient line search method to better reduce function values under the same computation and communication cost.
- As a result, practical training speed is significantly faster than existing distributed dual methods.

Outline

Algorithm Overview

- Let the iterate at iteration t be $\boldsymbol{\alpha}^t$, all iterative algorithms update it by

$$\boldsymbol{\alpha}^{t+1} = \boldsymbol{\alpha}^t + \eta_t \Delta\boldsymbol{\alpha}^t,$$

  $\Delta\boldsymbol{\alpha}^t$ is the direction and η_t is the step size.
- Line search for η_t may be expensive, so existing works all use fixed step sizes.
- We propose efficient methods for conducting line search.

Algorithm Overview (Cont'd)

- First, we obtain $\Delta\boldsymbol{\alpha}^t$ by constructing independent sub-problems that can be solved distributedly without communication.
- Machine k computes corresponding $\Delta\boldsymbol{\alpha}_k^t$ only.
- Then conduct an $O(n)$ communication to make

$$\Delta\mathbf{w}^t = \sum_{k=1}^{K} Y_k X_k \Delta\boldsymbol{\alpha}_k^t$$

  available to all machines.
- An $O(1)$ communication is then conducted to let each machine obtain the same step size η_t.

1. Given $\boldsymbol{\alpha}^t$ and the corresponding $\mathbf{w}^t$. Each machine has $\boldsymbol{\alpha}_k^t$ and the whole $\mathbf{w}^t$.
2. Each machine solves local sub-problem in parallel to obtain $\Delta\boldsymbol{\alpha}^t$.
3. Gather $\Delta\mathbf{w}^t = \sum_{k=1}^{K} Y_k X_k \Delta\boldsymbol{\alpha}_k^t$ from local results and broadcast. ($O(n)$ communication)
4. Compute the step size η_t using $\boldsymbol{\alpha}^t$, $\Delta\boldsymbol{\alpha}^t$, $\mathbf{w}^t$, and $\Delta\mathbf{w}^t$ (Key factor, $O(1)$ communication)
   4.1 Option 1: Armijo line search (omitted)
   4.2 Option 2: exact line search
5. $\mathbf{w}^{t+1} = \mathbf{w}^t + \eta_t \Delta\mathbf{w}^t, \boldsymbol{\alpha}_k^{t+1} = \boldsymbol{\alpha}_k^t + \eta_t \Delta\boldsymbol{\alpha}_k^t$.

1. Given $\boldsymbol{\alpha}^t$ and the corresponding $\mathbf{w}^t$. Each machine has $\boldsymbol{\alpha}_k^t$ and the whole $\mathbf{w}^t$.
2. Each machine solves local sub-problem in parallel to obtain $\Delta\boldsymbol{\alpha}^t$.
3. Gather $\Delta\mathbf{w}^t = \sum_{k=1}^{K} Y_k X_k \Delta\boldsymbol{\alpha}_k^t$ from local results and broadcast. ($O(n)$ communication)
4. Compute the step size η_t using $\boldsymbol{\alpha}^t$, $\Delta\boldsymbol{\alpha}^t$, $\mathbf{w}^t$, and $\Delta\mathbf{w}^t$ (Key factor, $O(1)$ communication)
   - 4.1 Option 1: Armijo line search (omitted)
   - 4.2 Option 2: exact line search
5. $\mathbf{w}^{t+1} = \mathbf{w}^t + \eta_t \Delta\mathbf{w}^t, \boldsymbol{\alpha}_k^{t+1} = \boldsymbol{\alpha}_k^t + \eta_t \Delta\boldsymbol{\alpha}_k^t$.

Update Direction

- Obtain $\Delta\boldsymbol{\alpha}^t$ by solving a quadratic problem.

$$\Delta\boldsymbol{\alpha}^t = \arg\min_{\mathbf{d}:\mathbf{0}\leq\boldsymbol{\alpha}^t+\mathbf{d}\leq C\mathbf{e}} \nabla f(\boldsymbol{\alpha}^t)^T\mathbf{d} + \frac{1}{2}\mathbf{d}^T H\mathbf{d},$$

  where H is some approximation of Q.
- Block-diagonal $\Rightarrow$ can be solved locally.
- A simple choice: $H = \tilde{Q} + \epsilon I$ with any $\epsilon > 0$,

$$\tilde{Q}_{i,j} = \begin{cases} Q_{i,j} & \text{if } i,j \text{ are in the same partition.} \\ 0 & \text{otherwise.} \end{cases}$$

- But can use any positive definite symmetric matrix and still have linear convergence.

1. Given $\boldsymbol{\alpha}^t$ and the corresponding $\mathbf{w}^t$. Each machine has $\boldsymbol{\alpha}_k^t$ and the whole $\mathbf{w}^t$.
2. Each machine solves local sub-problem in parallel to obtain $\Delta\boldsymbol{\alpha}^t$.
3. Gather $\Delta\mathbf{w}^t = \sum_{k=1}^{K} Y_k X_k \Delta\boldsymbol{\alpha}_k^t$ from local results and broadcast. ($O(n)$ communication)
4. Compute the step size η_t using $\boldsymbol{\alpha}^t$, $\Delta\boldsymbol{\alpha}^t$, $\mathbf{w}^t$, and $\Delta\mathbf{w}^t$ (Key factor, $O(1)$ communication)
   4.1 Option 1: Armijo line search (omitted)
   4.2 Option 2: exact line search
5. $\mathbf{w}^{t+1} = \mathbf{w}^t + \eta_t \Delta\mathbf{w}^t, \boldsymbol{\alpha}_k^{t+1} = \boldsymbol{\alpha}_k^t + \eta_t \Delta\boldsymbol{\alpha}_k^t$.

1. Given $\boldsymbol{\alpha}^t$ and the corresponding $\mathbf{w}^t$. Each machine has $\boldsymbol{\alpha}_k^t$ and the whole $\mathbf{w}^t$.
2. Each machine solves local sub-problem in parallel to obtain $\Delta\boldsymbol{\alpha}^t$.
3. Gather $\Delta\mathbf{w}^t = \sum_{k=1}^{K} Y_k X_k \Delta\boldsymbol{\alpha}_k^t$ from local results and broadcast. ($O(n)$ communication)
4. Compute the step size η_t using $\boldsymbol{\alpha}^t$, $\Delta\boldsymbol{\alpha}^t$, $\mathbf{w}^t$, and $\Delta\mathbf{w}^t$ (Key factor, $O(1)$ communication)
   4.1 Option 1: Armijo line search (omitted)
   4.2 Option 2: exact line search
5. $\mathbf{w}^{t+1} = \mathbf{w}^t + \eta_t \Delta\mathbf{w}^t, \boldsymbol{\alpha}_k^{t+1} = \boldsymbol{\alpha}_k^t + \eta_t \Delta\boldsymbol{\alpha}_k^t$.

1. Given $\boldsymbol{\alpha}^t$ and the corresponding $\mathbf{w}^t$. Each machine has $\boldsymbol{\alpha}_k^t$ and the whole $\mathbf{w}^t$.
2. Each machine solves local sub-problem in parallel to obtain $\Delta\boldsymbol{\alpha}^t$.
3. Gather $\Delta\mathbf{w}^t = \sum_{k=1}^{K} Y_k X_k \Delta\boldsymbol{\alpha}_k^t$ from local results and broadcast. ($O(n)$ communication)
4. Compute the step size η_t using $\boldsymbol{\alpha}^t$, $\Delta\boldsymbol{\alpha}^t$, $\mathbf{w}^t$, and $\Delta\mathbf{w}^t$ (Key factor, $O(1)$ communication)
   4.1 Option 1: Armijo line search (omitted)
   4.2 Option 2: exact line search
5. $\mathbf{w}^{t+1} = \mathbf{w}^t + \eta_t \Delta\mathbf{w}^t, \boldsymbol{\alpha}_k^{t+1} = \boldsymbol{\alpha}_k^t + \eta_t \Delta\boldsymbol{\alpha}_k^t$.

Exact Line Search

- Observe $f(\boldsymbol{\alpha}^t + \eta_t \Delta \boldsymbol{\alpha}^t)$ is a convex quadratic function of η_t, so we can simply solve

$$\frac{\partial f(\boldsymbol{\alpha} + \eta_t \Delta \boldsymbol{\alpha})}{\partial \eta_t} = 0$$

Exact Line Search

- Observe $f(\boldsymbol{\alpha}^t + \eta_t \Delta\boldsymbol{\alpha}^t)$ is a convex quadratic function of η_t, so we can simply solve

$$\frac{\partial f(\boldsymbol{\alpha} + \eta_t \Delta\boldsymbol{\alpha})}{\partial \eta_t} = 0 \Rightarrow \eta_t^* = \frac{-\nabla f(\boldsymbol{\alpha}^t)^T \Delta\boldsymbol{\alpha}^t}{(\Delta\boldsymbol{\alpha}^t)^T Q \Delta\boldsymbol{\alpha}^t}.$$

Exact Line Search

- Observe $f(\boldsymbol{\alpha}^t + \eta_t \Delta\boldsymbol{\alpha}^t)$ is a convex quadratic function of η_t, so we can simply solve

$$\frac{\partial f(\boldsymbol{\alpha} + \eta_t \Delta\boldsymbol{\alpha})}{\partial \eta_t} = 0 \Rightarrow \eta_t^* = \frac{-\nabla f(\boldsymbol{\alpha}^t)^T \Delta\boldsymbol{\alpha}^t}{(\Delta\boldsymbol{\alpha}^t)^T Q \Delta\boldsymbol{\alpha}^t}.$$

- $(\Delta\boldsymbol{\alpha}^t)^T Q \Delta\boldsymbol{\alpha}^t = (\Delta\mathbf{w}^t)^T \Delta\mathbf{w}^t$ can be obtained because $\Delta\mathbf{w}^t$ is available.
- To ensure $\mathbf{0} \leq \boldsymbol{\alpha}^t + \eta_t \Delta\boldsymbol{\alpha}^t \leq C\mathbf{e}$, take

$$\eta_t = \min\{\eta^*, \max\{\eta \mid \mathbf{0} \leq \boldsymbol{\alpha}^t + \eta\Delta\boldsymbol{\alpha}^t \leq C\mathbf{e}\}\}.$$

Convergence

Theorem

Our algorithm with either line search approach has global linear convergence. Therefore this approach requires $O(\log(1/\epsilon))$ iterations to obtain an ϵ-accurate solution.

Pocket Approach

- Dual solvers do not guarantee decreasing primal objective
- But we want $\mathbf{w}$ that minimizes the primal problem
- Maintain $\mathbf{w}^k$ that has the smallest primal objective as the current output model

Outline

Introduction

Algorithm

Experiments

Discussions and Conclusions

Solvers

Method	H	η_t
DSVM-AVE (Pechyony et al., 2011)	$\tilde{Q}$	$1/K$
Jaggi et al. (2014)	$\tilde{Q}$	$1/K$
DisDCA (Yang, 2013), practical variant	$K\tilde{Q}$	1
Ma et al. (2015)	$K\tilde{Q}^*$	1

*:Other choices require solving an eigenvalue problem

- $O(1/\epsilon)$ convergence.
- Distributed Newton method (Zhuang et al., 2015): solves the primal, requires differentiability. Compare in L2-loss SVM.
- **BQO-E**: ours: exact line search ($H = \tilde{Q} + \epsilon I$)
- **BQO-A**: Armijo line search ($H = \tilde{Q} + \epsilon I$)

Data Sets and Experiment Setting

Data set	#instances	#features	#nonzeros
webspam	280,000	16,609,143	1,044,393,506
url	1,916,904	3,231,961	221,663,296
epsilon	400,000	2,000	800,000,000

- 16 machines in a local cluster.
- $C = 1$.
- All methods implemented in MPI/C++.
- Every dual solver: one pass through data (sample without replacement) by dual coordinate descent (Hsieh et al., 2008) and then communication.
- Pocket approach for all dual solvers.

Wait

- Your framework requires obtaining min of the problem

$$\Delta\boldsymbol{\alpha}^t = \arg\min_{\mathbf{d}:\mathbf{0}\le\boldsymbol{\alpha}^t+\mathbf{d}\le C\mathbf{e}} \nabla f(\boldsymbol{\alpha}^t)^T\mathbf{d} + \frac{1}{2}\mathbf{d}^T H\mathbf{d},$$

  but one pass apparently is not the min.
- One pass will generate a descent direction with enough decrease (Wang and Lin, 2014), and we can find an H such that this direction is its minimizer.
- Still global linear convergence.

Results

Training time of L1-loss SVM for reaching

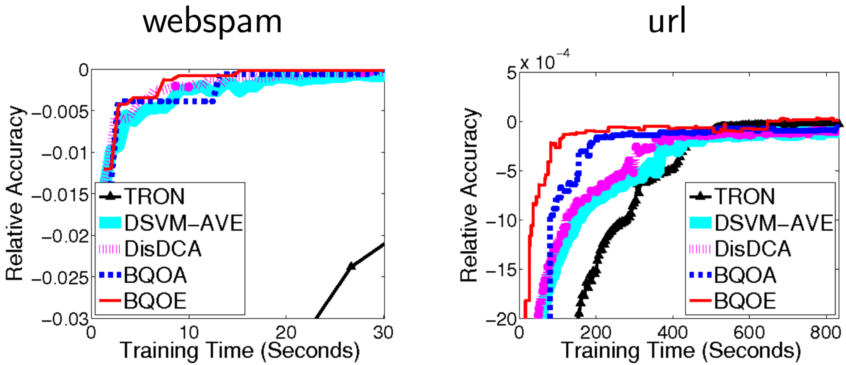
$$\left|\frac{f(\alpha) - f(\alpha^*)}{f(\alpha^*)}\right| \leq 0.01.$$

Data set	**BQO-E**	**BQO-A**	DisDCA	DSVM-AVE
webspam	21.7	30.6	54.7	79.5
url	1,092.9	1,801.2	3,524.2	4,343.4
epsilon	2.8	3.9	8.0	13.2

- **BQO-E** is 2.56 - 4.76 times faster than existing methods

Results (Cont'd)

L2-loss SVM: test accuracy



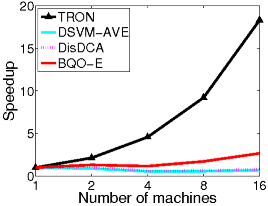
BQO-E achieves stable test accuracies the fastest

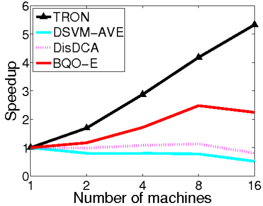
Speedup

L2-loss SVM: speedup of different #machines

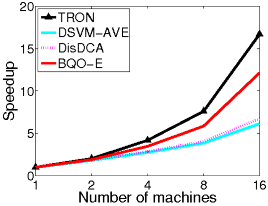
epsilon | webspam

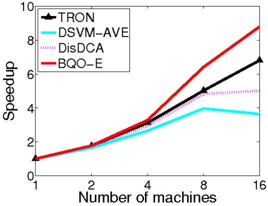
Training





Training + IO





Outline

Speedup

- All dual methods have bad speedup in comparison to Newton because of the block-diagonal approximation.
- But running time might still be good enough for moderate number of machines if we also consider I/O time.
- Still important in problems in which directly solving the primal is not feasible, like multi-class SVM and structured SVM.

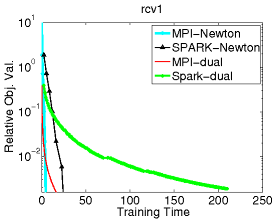
Why not Spark?

- Apache Spark is emerging as a new distributed platform, and we had successful experiences in developing packages on it (Lin et al., 2014).
- But dual methods do not work well on Spark because of higher overheads.

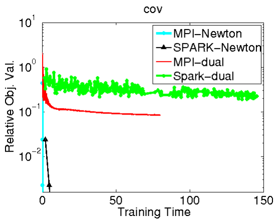
Why not Spark?

- Apache Spark is emerging as a new distributed platform, and we had successful experiences in developing packages on it (Lin et al., 2014).
- But dual methods do not work well on Spark because of higher overheads.
- L2-loss SVM, $C = 1$, 16 machines.

rcv1-test



covtype



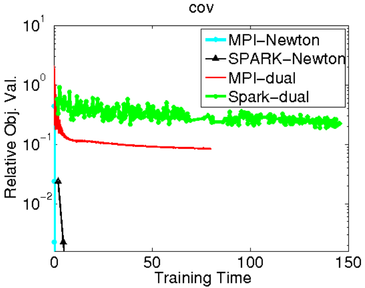
*Spark-dual: L2-loss modified from `http://github.com/gingsmith/cocoa/`.

When is Dual Method not Suitable?

- When $\ell \gg n$, usually dual methods are worse.

When is Dual Method not Suitable?

- When $\ell \gg n$, usually dual methods are worse.
- Example (from the previous page): covtype: $\ell = 581,012$, $n = 54$.



Conclusions

- An efficient method for distributedly training dual problems.
- The algorithm is superior to state of the art distributed linear SVM solvers both theoretically and practically.
- Implementation available in `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/distributed-liblinear/`.