Large-scale Linear Classification: Status and Challenges

Chih-Jen Lin
Department of Computer Science
National Taiwan University

Talk at Criteo Machine Learning Workshop, November 8, 2017

Outline

Outline

Linear Classification

- Although many new and advanced techniques are available (e.g., deep learning), linear classifiers remain to be useful because of their simplicity
- We have fast training/prediction for large-scale data
- A large-scale optimization problem is solved
- The focus of this talk is on how to solve this optimization problem

The Software LIBLINEAR

- My talk will be very related to research done in developing the software LIBLINEAR for linear classification

  `www.csie.ntu.edu.tw/~cjlin/liblinear`

- It is now one of the most used linear classification tools

Linear and Kernel Classification

Methods such as SVM and logistic regression are often used in two ways

- Kernel methods: data mapped to another space

$$\boldsymbol{x} \Rightarrow \phi(\boldsymbol{x})$$

  $\phi(\boldsymbol{x})^T\phi(\mathbf{y})$ easily calculated; no good control on $\phi(\cdot)$

- Feature engineering + linear classification:

  Directly use $\boldsymbol{x}$ without mapping. But $\boldsymbol{x}$ may have been carefully generated. Full control on $\boldsymbol{x}$

Comparison Between Linear and Kernel

- For certain problems, accuracy by linear is as good as kernel

  But training and testing are much faster
- Especially document classification

  Number of features (bag-of-words model) very large

  Large and sparse data
- Training millions of data in just a few seconds

Comparison Between Linear and Nonlinear (Training Time & Testing Accuracy)

	Linear		RBF Kernel	
Data set	Time	Accuracy	Time	Accuracy
MNIST38	0.1	96.82	38.1	99.70
ijcnn1	1.6	91.81	26.8	98.69
covtype_multiclass	1.4	76.37	46,695.8	96.11
news20	1.1	96.95	383.2	96.90
real-sim	0.3	97.44	938.3	97.82
yahoo-japan	3.1	92.63	20,955.2	93.31
webspam	25.7	93.35	15,681.8	99.26

Size reasonably large: e.g., yahoo-japan: 140k instances and 830k features

Comparison Between Linear and Nonlinear (Training Time & Testing Accuracy)

	Linear		RBF Kernel	
Data set	Time	Accuracy	Time	Accuracy
MNIST38	0.1	96.82	38.1	99.70
ijcnn1	1.6	91.81	26.8	98.69
covtype_multiclass	1.4	76.37	46,695.8	96.11
news20	1.1	96.95	383.2	96.90
real-sim	0.3	97.44	938.3	97.82
yahoo-japan	3.1	92.63	20,955.2	93.31
webspam	25.7	93.35	15,681.8	99.26

Size reasonably large: e.g., yahoo-japan: 140k instances and 830k features

Comparison Between Linear and Nonlinear (Training Time & Testing Accuracy)

	Linear		RBF Kernel	
Data set	Time	Accuracy	Time	Accuracy
MNIST38	0.1	96.82	38.1	99.70
ijcnn1	1.6	91.81	26.8	98.69
covtype_multiclass	1.4	76.37	46,695.8	96.11
news20	1.1	96.95	383.2	96.90
real-sim	0.3	97.44	938.3	97.82
yahoo-japan	3.1	92.63	20,955.2	93.31
webspam	25.7	93.35	15,681.8	99.26

Size reasonably large: e.g., yahoo-japan: 140k instances and 830k features

Binary Linear Classification

- Training data $\{y_i, \boldsymbol{x}_i\}, \boldsymbol{x}_i \in R^n, i = 1, \ldots, l, y_i = \pm 1$
- l: # of data, n: # of features

$$\min_{\boldsymbol{w}} \quad f(\boldsymbol{w}), \text{ where } f(\boldsymbol{w}) \equiv$$

$$C \sum_{i=1}^{l} \xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i) + \begin{cases} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} & \text{L2 regularization} \\ \|\boldsymbol{w}\|_1 & \text{L1 regularization} \end{cases}$$

- $\xi(\boldsymbol{w}; \boldsymbol{x}, y)$: loss function: we hope $y\boldsymbol{w}^T\boldsymbol{x} > 0$
- C: regularization parameter

Loss Functions

- Some commonly used loss functions.

$$\xi_{\text{L1}}(\boldsymbol{w}; \boldsymbol{x}, y) \equiv \max(0, 1 - y\boldsymbol{w}^T\boldsymbol{x}), \quad (1)$$

$$\xi_{\text{L2}}(\boldsymbol{w}; \boldsymbol{x}, y) \equiv \max(0, 1 - y\boldsymbol{w}^T\boldsymbol{x})^2, \quad (2)$$

$$\xi_{\text{LR}}(\boldsymbol{w}; \boldsymbol{x}, y) \equiv \log(1 + e^{-y\boldsymbol{w}^T\boldsymbol{x}}). \quad (3)$$

- SVM (Boser et al., 1992; Cortes and Vapnik, 1995): (1)-(2)
- Logistic regression (LR): (3)

Outline

Optimization Methods

- A difference between linear and kernel is that for kernel, optimization must be over a variable $\boldsymbol{\alpha}$ (usually through the dual problem) where

$$\boldsymbol{w} = \sum_{i=1}^{l} \alpha_i \phi(\boldsymbol{x}_i)$$

  We cannot minimize over $\boldsymbol{w}$, which may be infinite dimensional
- However, for linear, minimizing over $\boldsymbol{w}$ or $\boldsymbol{\alpha}$ is ok

Optimization Methods (Cont'd)

Unconstrained optimization methods can be categorized to

- Low-order methods: quickly get a model, but slow final convergence
- High-order methods: more robust and useful for ill-conditioned situations

We will show both types of optimization methods are useful for linear classification

Further, to handle large problems, the algorithms must take problem structure into account

Let's discuss a low-order method (coordinate descent) in detail

Coordinate Descent

- We consider L1-loss and the dual SVM problem

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & f(\boldsymbol{\alpha}) \\ \text{subject to} \quad & 0 \le \alpha_i \le C, \forall i, \end{aligned}$$

where

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

and

$$Q_{ij} = y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j, \quad \boldsymbol{e} = [1, \dots, 1]^T$$

- We will apply coordinate descent (CD) methods
- The situation for L2 or LR loss is very similar

Coordinate Descent (Cont'd)

- For current $\boldsymbol{\alpha}$, change α_i by fixing others
- Let

$$\boldsymbol{e}_i = [0, \ldots, 0, 1, 0, \ldots, 0]^T$$

- The sub-problem is

$$\min_d \ f(\boldsymbol{\alpha} + d\boldsymbol{e}_i) = \frac{1}{2} Q_{ii} d^2 + \nabla_i f(\boldsymbol{\alpha}) d + \text{constant}$$

$$\text{subject to } 0 \leq \alpha_i + d \leq C$$

- Without constraints

$$\text{optimal } d = -\frac{\nabla_i f(\boldsymbol{\alpha})}{Q_{ii}}$$

Coordinate Descent (Cont'd)

- Now $0 \le \alpha_i + d \le C$

$$\alpha_i \leftarrow \min\left(\max\left(\alpha_i - \frac{\nabla_i f(\boldsymbol{\alpha})}{Q_{ii}}, 0\right), C\right)$$

- Note that

$$\nabla_i f(\boldsymbol{\alpha}) = (Q\boldsymbol{\alpha})_i - 1 = \sum_{j=1}^{l} Q_{ij}\alpha_j - 1$$
$$= \sum_{j=1}^{l} y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \alpha_j - 1$$

- Expensive: $O(ln)$, l: # instances, n: features

Coordinate Descent (Cont'd)

- A trick in Hsieh et al. (2008) is to define and maintain

$$\mathbf{u} \equiv \sum_{j=1}^{l} y_j \alpha_j \boldsymbol{x}_j,$$

- Easy gradient calculation: the cost is $O(n)$

$$\nabla_i f(\boldsymbol{\alpha}) = y_i \mathbf{u}^T \boldsymbol{x}_i - 1$$

- Note that this cannot be done for kernel as $\boldsymbol{x}_i$ is high dimensional

Coordinate Descent (Cont'd)

The procedure

- While $\boldsymbol{\alpha}$ is not optimal (Outer iteration)
  - For $i = 1, \ldots, l$ (Inner iteration)

    (a) $\bar{\alpha}_i \leftarrow \alpha_i$

    (b) $G = y_i \mathbf{u}^T \boldsymbol{x}_i - 1$

    (c) $\alpha_i \leftarrow \min(\max(\alpha_i - G/Q_{ii}, 0), C)$

    (d) If α_i needs to be changed

    $$\mathbf{u} \leftarrow \mathbf{u} + (\alpha_i - \bar{\alpha}_i) y_i \boldsymbol{x}_i$$

Maintaining $\mathbf{u}$ also costs

$$O(n)$$

Coordinate Descent (Cont'd)

- Having

$$\mathbf{u} \equiv \sum_{j=1}^{l} y_j \alpha_j \boldsymbol{x}_j,$$

$$\nabla_i f(\boldsymbol{\alpha}) = y_i \mathbf{u}^T \boldsymbol{x}_i - 1$$

and

$$\bar{\alpha}_i : \text{ old} ; \qquad \alpha_i : \text{ new}$$

$$\mathbf{u} \leftarrow \mathbf{u} + (\alpha_i - \bar{\alpha}_i) y_i \boldsymbol{x}_i.$$

is very essential

- This isn't the vanilla CD dated back to Hildreth (1957)
- We take the problem structure into account

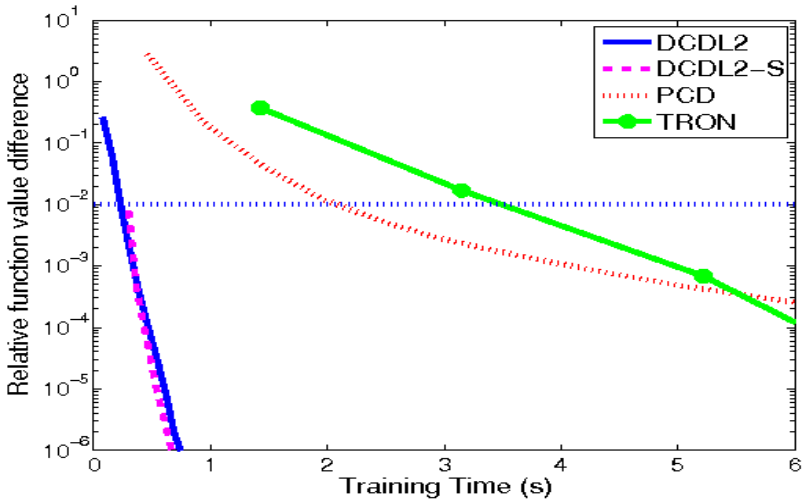
Comparisons

L2-loss SVM is used

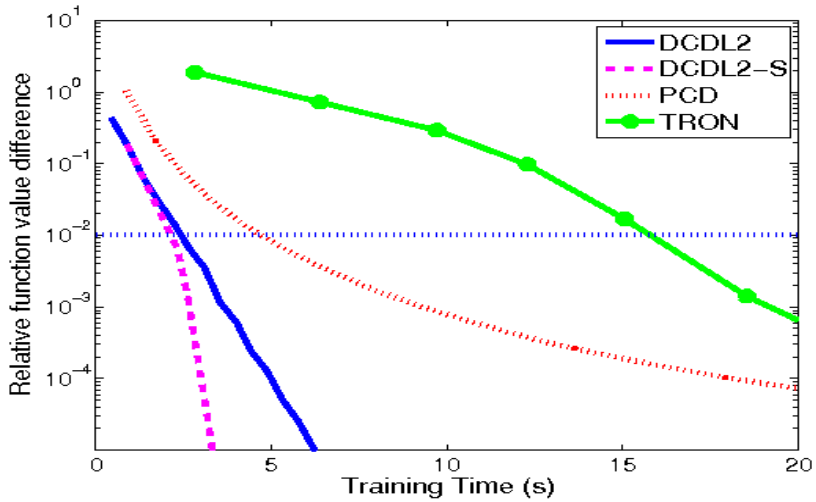
- DCDL2: Dual coordinate descent
- DCDL2-S: DCDL2 with shrinking
- PCD: Primal coordinate descent
- TRON: Trust region Newton method

This result is from Hsieh et al. (2008) with $C = 1$

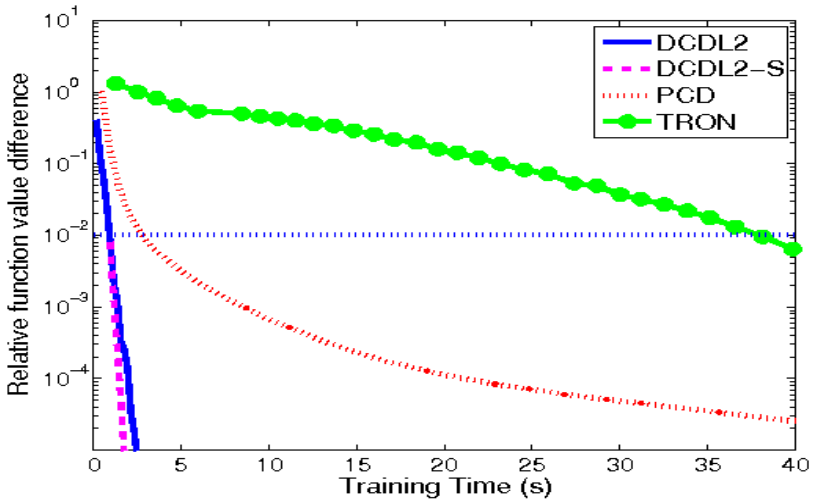
Objective values (Time in Seconds)



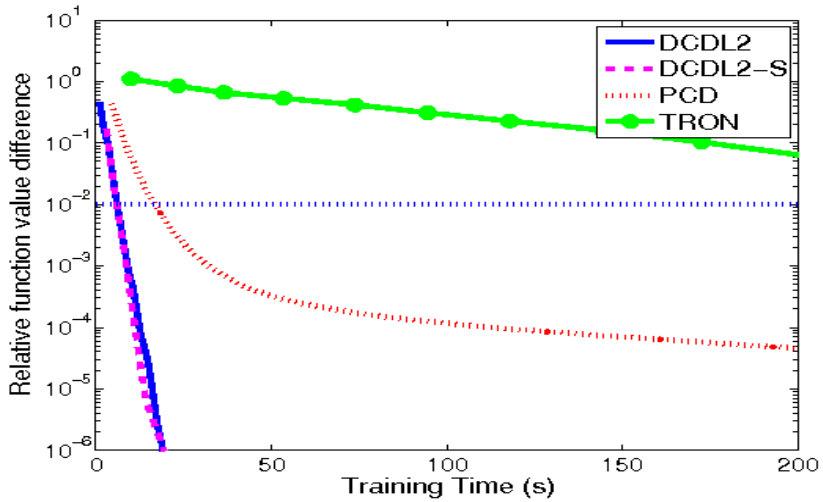
news20



rcv1



yahoo-japan



yahoo-korea

Low- versus High-order Methods

- We see low-order methods are efficient, but high-order methods are useful for difficult situations
- CD for dual

```
$ time ./train -c 1 news20.scale
2.528s
$ time ./train -c 100 news20.scale
28.589s
```

- Newton for primal

```
$ time ./train -c 1 -s 2 news20.scale
8.596s
$ time ./train -c 100 -s 2 news20.scale
11.088s
```

Training Median-sized Data: Status

- Basically a solved problem
- However, as data and memory continue to grow, new techniques are needed for large-scale sets.
- Two possible strategies are
  1. Multi-core linear classification
  2. Distributed linear classification

Outline

Multi-core Linear Classification

- Nowadays each CPU has several cores
- However, parallelizing algorithms to use multiple cores may not be that easy
- In fact, algorithms may need to be redesigned
- Since two years ago we have been working on multi-core LIBLINEAR

Multi-core Linear Classification (Cont'd)

- Three multi-core solvers have been released
  1. Newton method for primal L2-regularized problem (Lee et al., 2015)
  2. Coordinate descent method for dual L2-regularized problem (Chiang et al., 2016)
  3. Coordinate descent method for primal L1-regularized problem (Zhuang et al., 2017)
- They are practically useful. For example, one user from USC thanked us because "a job (taking >30 hours using one core) now can finish within 5 hours"
- We will briefly discuss the 2nd and the 3rd

Multi-core CD for Dual

Recall the CD algorithm for dual is

- While $\boldsymbol{\alpha}$ is not optimal (Outer iteration)

  For $i = 1, \ldots, l$ (Inner iteration)

  (a) $\bar{\alpha}_i \leftarrow \alpha_i$

  (b) $G = y_i \mathbf{u}^T \boldsymbol{x}_i - 1$

  (c) $\alpha_i \leftarrow \min(\max(\alpha_i - G/Q_{ii}, 0), C)$

  (d) If α_i needs to be changed

  $$\mathbf{u} \leftarrow \mathbf{u} + (\alpha_i - \bar{\alpha}_i) y_i \boldsymbol{x}_i$$

Multi-core CD for Dual (Cont'd)

- The algorithm is inherently sequential
- Suppose

$$\alpha_{j'} \text{ is updated after } \alpha_j$$

  Then $\alpha_{j'}$ must wait until the latest $\mathbf{u}$ is obtained
- The parallelization is difficult

Multi-core CD for Dual (Cont'd)

- Asynchronous CD is possible (Hsieh et al., 2015), but may diverge
- We note that for a given set $\bar{B}$

$$\nabla_i f(\boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}_i, \forall i \in \bar{B}$$

  can be calculated in parallel
- We then propose a framework

Multi-core CD for Dual (Cont'd)

- While $\boldsymbol{\alpha}$ is not optimal
  - (a) Select a set $\bar{B}$
  - (b) Calculate $\nabla_{\bar{B}} f(\boldsymbol{\alpha})$ in parallel
  - (c) Select $B \subset \bar{B}$ with $|B| \ll |\bar{B}|$
  - (d) Sequentially update $\alpha_i, i \in B$

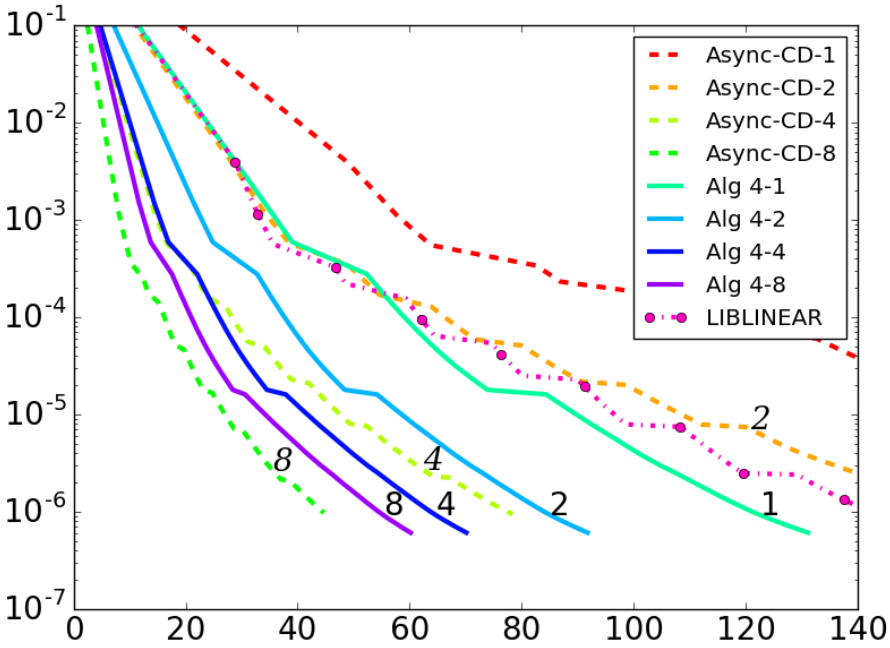
Multi-core CD for Dual (Cont'd)

- The selection of

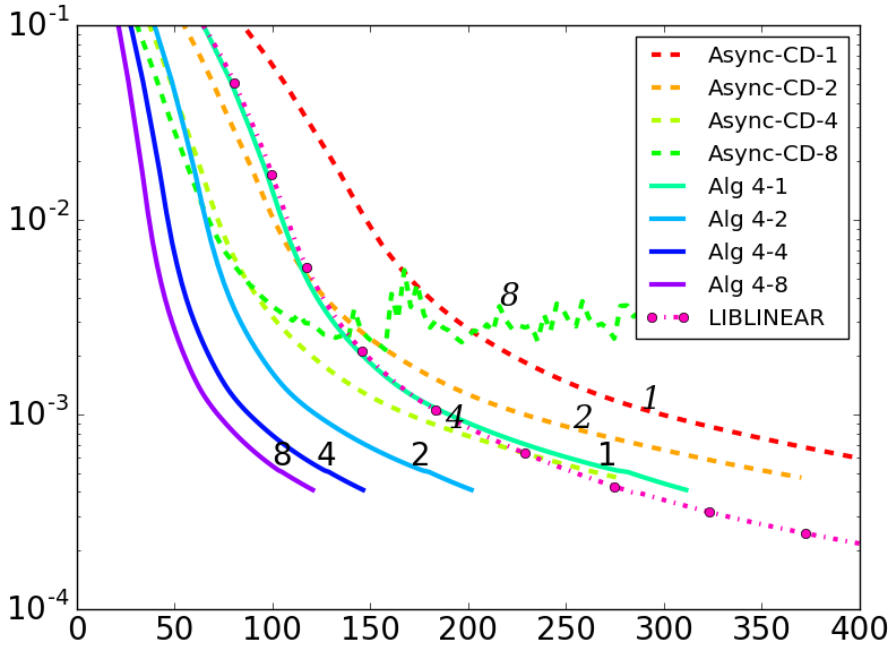
$$B \subset \bar{B} \text{ with } |B| \ll |\bar{B}|$$

  is by $\nabla_{\bar{B}} f(\boldsymbol{w})$
- The idea is simple, but needs efforts to have a practical setting (details omitted)

Multi-core CD for Dual (Cont'd)



webspam



url_combined

- Alg-4: the method in Chiang et al. (2016)
- Asynchronous CD (Hsieh et al., 2015)

Multi-core CD for L1 Regularization

- Currently, primal CD (Yuan et al., 2010) or its variants (Yuan et al., 2012) is the state-of-the-art for L1
- Each CD step involves one feature
- Some attempts of parallel CD for L1 include
  - Asynchronous CD (Bradley et al., 2011)
  - Block CD (Bian et al., 2013)
- These methods are not satisfactory for either
  - divergence issue, or
  - poor speedup

Multi-core CD for L1 Regularization (Cont'd)

- We struggled for years for find a solution
- Recently, in a work (Zhuang et al., 2017) we have an effective setting
- It's partially supported by Criteo Faculty Research Award
- Our idea is simple: direct parallelization of CD
- But wait.. This shouldn't work because each CD iteration is cheap

Direct Parallelization of CD

- Let's consider a simple setting to decide if one CD step should be parallelized or not
- **if** #non-zeros in an instance/feature $\geq$ a threshold **then**

  multi-core

  else

  single-core
- Idea: a CD step is parallelized if there are enough operations

Direct Parallelization of CD (Cont'd)

- Speedup of CD for dual, L2 regularization

Data set		#threads: 2	4	8
sparse sets	avazu-app	0.4	0.3	0.2
	criteo	0.5	0.3	0.2
dense sets	epsilon_normalized	1.3	1.3	1.1
	splice_site.t.10%	1.8	2.8	4.1

- CD for dual: one instance at a time
- Threshold: 0 (sparse), 500 (dense)
- If 500 for sparse, no instance parallelized
- The speedup is poor

% of instances/features containing 50% and 80% #non-zeros

Data set	Instance		Feature	
avazu-app	50%	80%	0.2%	1%
criteo	50%	80%	0.01%	0.2%
kdd2010-a	40%	73%	0.03%	2%
kdd2012	50%	80%	0.003%	0.5%
rcv1_test	24%	54%	1%	5%
splice_site.t.10%	50%	80%	9%	57%
url_combined	44%	76%	0.002%	0.006%
webspam	29%	55%	0.6%	2%
yahoo-korea	20%	48%	0.07%	0.5%

Features' non-zero distribution is extremely skewed

Non-zeros are in few dense (and parallelizable) features

Speedup of CD for L1 Regularization

LR loss used	Naive			Block CD			Async. CD		
Data set	2	4	8	2	4	8	2	4	8
avazu-app	1.9	3.4	5.6	0.4	0.7	1.0	1.4	2.7	3.4
criteo	1.8	3.3	5.5	0.7	1.2	1.9	1.5	2.9	4.8
epsilon_normalized	2.0	4.0	7.9	x	x	x	1.3	2.1	x
HIGGS	2.0	3.9	7.5	0.7	0.8	0.9	1.0	1.3	x
kdd2010-a	1.7	2.4	3.1	0.8	1.4	2.4	1.5	2.7	4.8
kdd2012	1.9	2.8	3.9	0.2	0.4	0.6	2.1	4.7	7.0
rcv1_test	1.9	3.4	5.9	x	x	x	1.3	2.5	4.5
splice_site.t.10%	1.9	3.6	6.2	x	x	x	1.6	2.7	4.3
url_combined	2.0	3.5	6.2	0.5	0.9	1.3	1.0	1.7	1.7
webspam	1.8	3.2	4.8	0.1	0.3	0.5	1.4	2.5	4.1
yahoo-korea	1.9	3.5	5.9	0.2	0.3	0.5	1.3	2.4	4.4

Outline

Distributed Linear Classification

- It's even more complicated than multi-core
- I don't have time to discuss this topic in detail, but let me share some lessons
- A big mistake was that we worked on distributed before multi-core

Distributed Linear Classification (Cont'd)

- A few years ago, big data was hot. So we extended a Newton solver in LIBLINEAR to MPI (Zhuang et al., 2015) and Spark (Lin et al., 2014)
- We were a bit ahead of time; Spark MLlib wasn't even available then
- Unfortunately, very few people use our code, especially the Spark one
- We moved to multi-core. Immediately, multi-core LIBLINEAR has many users

Distributed Linear Classification (Cont'd)

- Why we failed? Several possible reasons
- Not many people have big data??
- System issues are more important than we thought.

  At that time Spark wasn't easy to use and was being actively changed
- System configuration and application scenarios may significantly vary

  An algorithm useful for systems with fast network speed may be useless for systems with slow communication

Distributed Linear Classification (Cont'd)

- Application dependency is stronger.

  L2 and L1 regularization often give similar accuracy. On a single machine, we may not want to use L1 because training is more difficult and the smaller model size isn't that important

  However, for distributed applications many have told me that they need L1
- A lesson is that for people from academia, it's better to collaborate with industry for research on distributed machine learning

Outline

Conclusions

- Linear classification is an old topic, but it remains to be useful for many applications
- Efficient training relies on designing optimization algorithms by incorporating the problem structure
- Many issues about multi-core and distributed linear classification still need to be studied