Large-scale Linear Classification

Chih-Jen Lin
Department of Computer Science
National Taiwan University

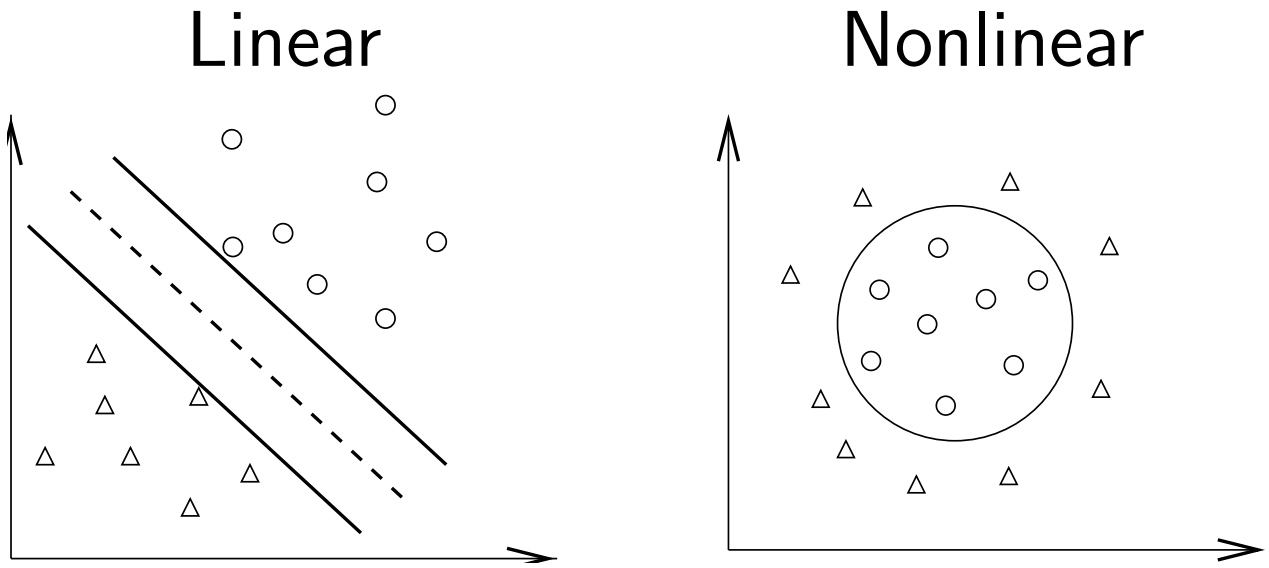
Talk at Criteo, August 1, 2014

Outline

- Introduction
- Optimization methods
- Extension of linear classification
- Big-data linear classification
- Conclusions and future directions

Outline

Linear and Nonlinear Classification



Linear: a linear function to separate data in the original input space; nonlinear: data mapped to other spaces

Original: [height, weight]

Nonlinear: [height, weight, weight/height^2]

Kernel is one of the nonlinear methods

Linear and Nonlinear Classification (Cont'd)

Methods such as SVM and logistic regression can be used in two ways

- Kernel methods: data mapped to another space

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

  $\phi(\boldsymbol{x})^T\phi(\boldsymbol{y})$ easily calculated; no good control on $\phi(\cdot)$
- Linear classification + feature engineering: Directly use $\boldsymbol{x}$ without mapping. But $\boldsymbol{x}$ may have been carefully generated using some nonlinear information. Full control on $\boldsymbol{x}$

We will focus on the 2nd type of approaches in this talk

Why Linear Classification?

- If $\phi(\boldsymbol{x})$ is high dimensional, decision function

$$\text{sgn}(\boldsymbol{w}^T\phi(\boldsymbol{x}))$$

  is expensive
- Kernel methods:

$$\boldsymbol{w} \equiv \sum_{i=1}^{l} \alpha_i \phi(\boldsymbol{x}_i) \text{ for some } \boldsymbol{\alpha}, K(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$$

  New decision function: $\text{sgn}\left(\sum_{i=1}^{l} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x})\right)$
- Special $\phi(\boldsymbol{x})$ so calculating $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is easy. Example:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv (\boldsymbol{x}_i^T \boldsymbol{x}_j + 1)^2 = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j), \phi(\boldsymbol{x}) \in R^{O(n^2)}$$

Why Linear Classification? (Cont'd)

- Prediction

$$\boldsymbol{w}^T\boldsymbol{x} \quad \text{versus} \quad \sum_{i=1}^{l} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x})$$

- If $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ takes $O(n)$, then

$$O(n) \quad \text{versus} \quad O(nl)$$

- Kernel: cost related to size of training data

  Linear: cheaper and simpler

Linear is Useful in Some Places

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

  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	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	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	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

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}), \quad f(\boldsymbol{w}) \equiv \frac{\boldsymbol{w}^T \boldsymbol{w}}{2} + C \sum_{i=1}^{l} \xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i)$$

- $\boldsymbol{w}^T \boldsymbol{w}/2$: regularization term (we have no time to talk about L1 regularization here)
- $\xi(\boldsymbol{w}; \boldsymbol{x}, y)$: loss function: we hope $y \boldsymbol{w}^T \boldsymbol{x} > 0$
- C: regularization parameter

Loss Functions

- Some commonly used ones:

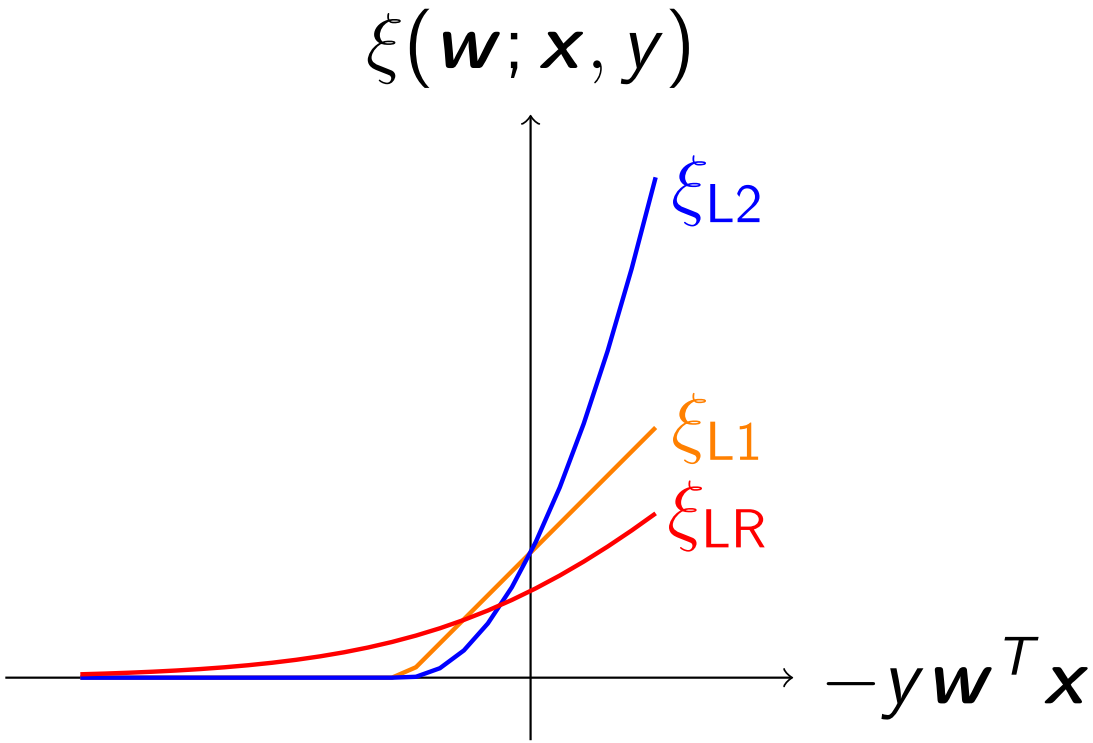
$$\xi_{\text{L1}}(\boldsymbol{w}; \boldsymbol{x}, y) \equiv \max(0, 1 - y\boldsymbol{w}^T\boldsymbol{x}), \quad (1)$$

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

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

- SVM (Boser et al., 1992; Cortes and Vapnik, 1995): (1)-(2)
- Logistic regression (LR): (3); no reference because it can be traced back to 19th century

Loss Functions (Cont'd)



Their performance is usually similar

Loss Functions (Cont'd)

However,

ξ_{L1}: not differentiable
ξ_{L2}: differentiable but not twice differentiable
ξ_{LR}: twice differentiable

The same optimization method may not be applicable to all these losses

Outline

Optimization Methods

- Many unconstrained optimization methods can be applied
- For kernel, optimization is over a variable α where

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

We cannot minimize over $\boldsymbol{w}$ because it may be infinite dimensional

- However, for linear, minimizing over $\boldsymbol{w}$ or α is ok

Optimization Methods (Cont'd)

Among unconstrained optimization methods,

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

We will quickly discuss some examples and show both types of optimization methods are useful for linear classification

Optimization: 2nd Order Methods

- Newton direction

$$\min_{\boldsymbol{s}} \quad \nabla f(\boldsymbol{w}^k)^T \boldsymbol{s} + \frac{1}{2} \boldsymbol{s}^T \nabla^2 f(\boldsymbol{w}^k) \boldsymbol{s}$$

- This is the same as solving Newton linear system

$$\nabla^2 f(\boldsymbol{w}^k) \boldsymbol{s} = -\nabla f(\boldsymbol{w}^k)$$

- Hessian matrix $\nabla^2 f(\boldsymbol{w}^k)$ too large to be stored

$$\nabla^2 f(\boldsymbol{w}^k) : n \times n, \quad n : \text{ number of features}$$

- But Hessian has a special form

$$\nabla^2 f(\boldsymbol{w}) = \mathcal{I} + CX^T DX,$$

Optimization: 2nd Order Methods (Cont'd)

- X: data matrix. D diagonal. For logistic regression,

$$D_{ii} = \frac{e^{-y_i \boldsymbol{w}^T \boldsymbol{x}_i}}{1 + e^{-y_i \boldsymbol{w}^T \boldsymbol{x}_i}}$$

- Using CG to solve the linear system. Only Hessian-vector products are needed

$$\nabla^2 f(\boldsymbol{w})\boldsymbol{s} = \boldsymbol{s} + C \cdot X^T(D(X\boldsymbol{s}))$$

- Therefore, we have a Hessian-free approach

Optimization: 1st Order Methods

- 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

1st Order Methods (Cont'd)

- Coordinate descent: a simple and classic technique
  Change one variable at a time
- Given current $\boldsymbol{\alpha}$. Let $\boldsymbol{e}_i = [0, \ldots, 0, 1, 0, \ldots, 0]^T$.

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

- Without constraints

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

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

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

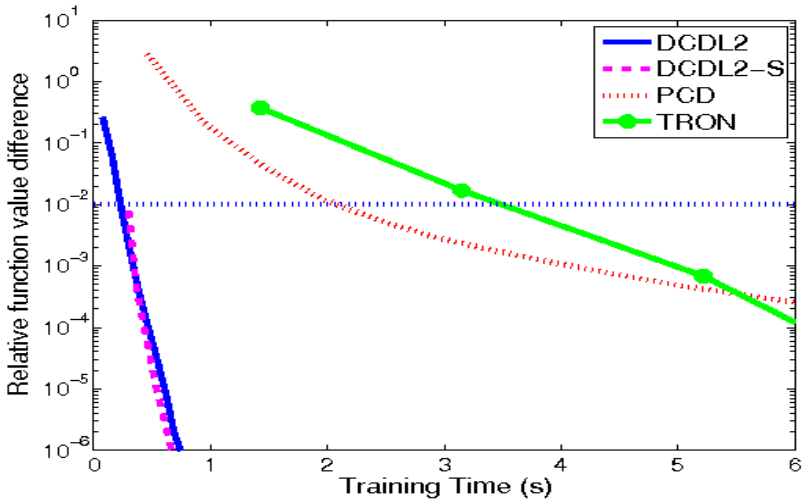
Comparisons

L2-loss SVM is used

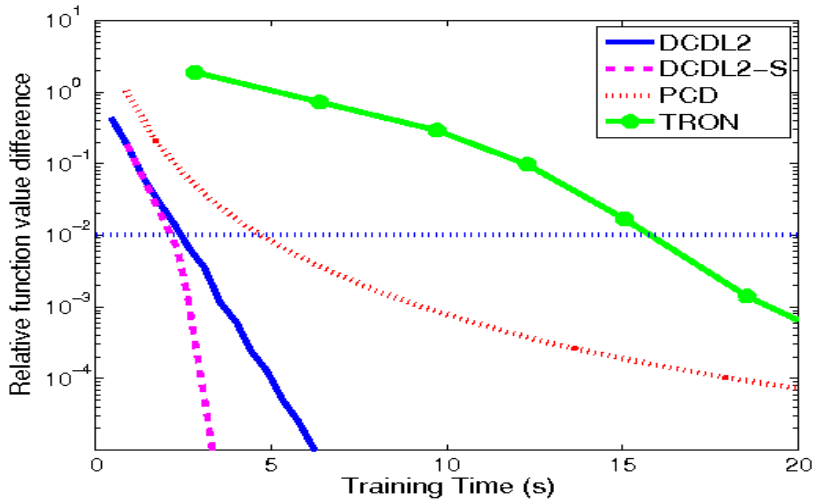
- DCDL2: Dual coordinate descent
- DCDL2-S: DCDL2 with shrinking
- PCD: Primal coordinate descent
- TRON: Trust region Newton method

This result is from Hsieh et al. (2008)

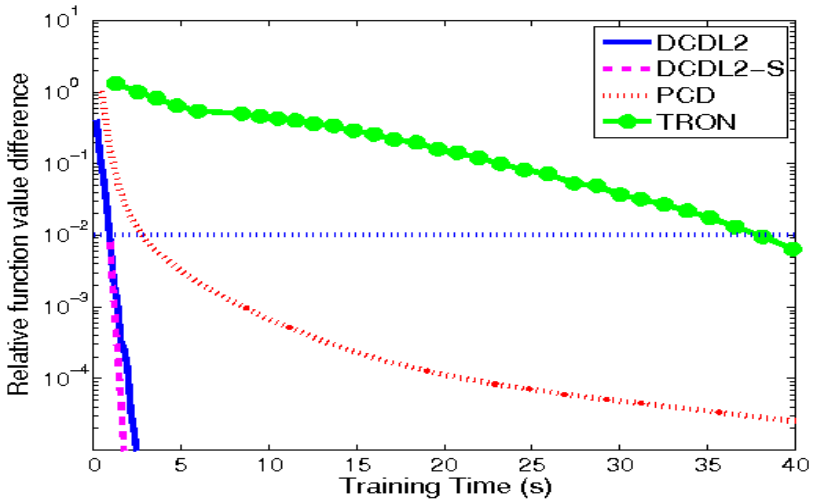
Objective values (Time in Seconds)



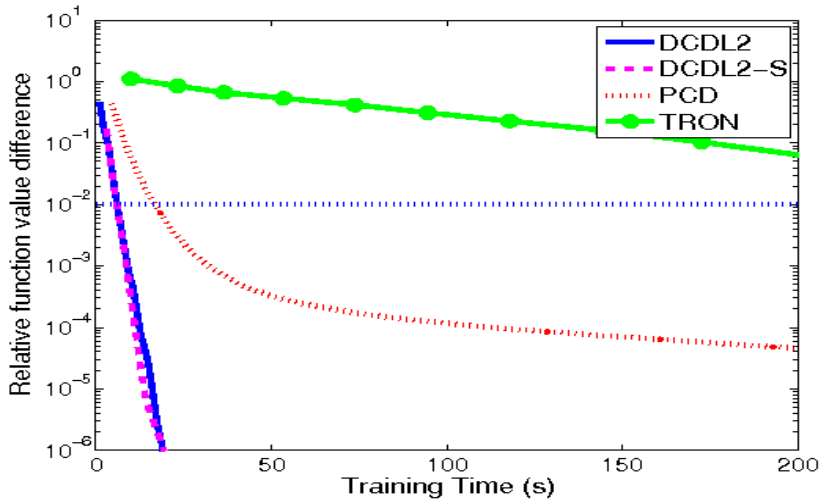
news20



rcv1



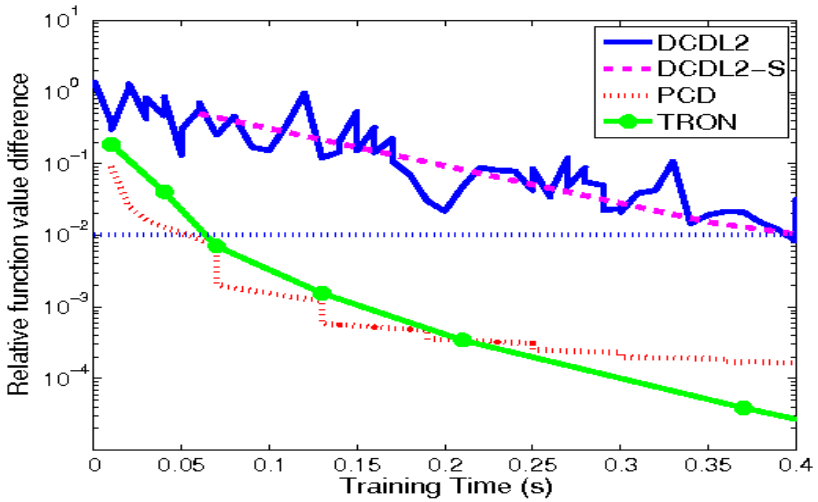
yahoo-japan



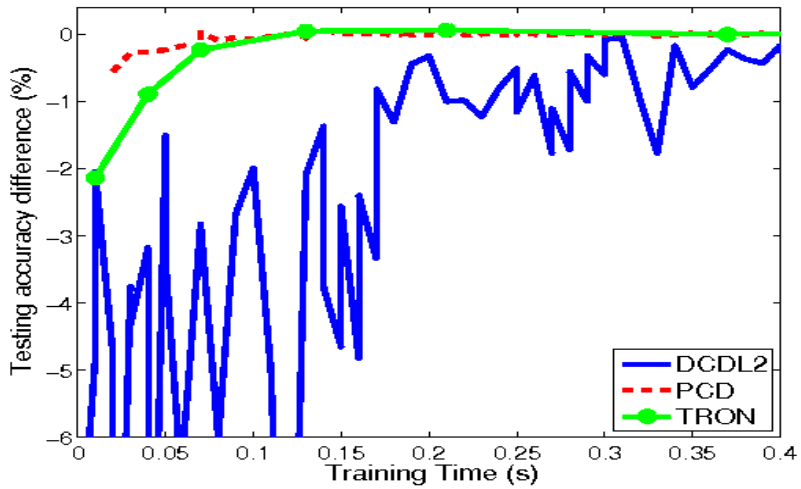
yahoo-korea

Low- versus High-order Methods

- We saw that low-order methods are efficient to give a model. However, high-order methods may be useful for difficult situationa
- An example: # instance: 32,561, # features: 123



Objective value



Accuracy

features is small $\Rightarrow$ solving primal is more suitable

Outline

Extension of Linear Classification

- Linear classification can be extended in different ways
- An important one is to approximate nonlinear classifiers
- Goal: better accuracy of nonlinear but faster training/testing
- Examples
  1. Explicit data mappings + linear classification
  2. Kernel approximation + linear classification
- I will focus on the first

Linear Methods to Explicitly Train $\phi(\boldsymbol{x}_i)$

- Example: low-degree polynomial mapping:

$$\phi(\boldsymbol{x}) = [1, x_1, \ldots, x_n, x_1^2, \ldots, x_n^2, x_1x_2, \ldots, x_{n-1}x_n]^T$$

- For this mapping, # features $= O(n^2)$
- When is it useful?

  Recall $O(n)$ for linear versus $O(nl)$ for kernel
- Now $O(n^2)$ versus $O(nl)$
- Sparse data

  $n \Rightarrow \bar{n}$, average # non-zeros for sparse data

  $\bar{n} \ll n \Rightarrow O(\bar{n}^2)$ may be much smaller than $O(l\bar{n})$

Example: Dependency Parsing

A multi-class problem with sparse data

n	Dim. of $\phi(\boldsymbol{x})$	l	$\bar{n}$	$\boldsymbol{w}$'s # nonzeros
46,155	1,065,165,090	204,582	13.3	1,438,456

- $\bar{n}$: average # nonzeros per instance
- Degree-2 polynomial is used
- Dimensionality of $\boldsymbol{w}$ is very high, but $\boldsymbol{w}$ is sparse
  Some training feature columns of $x_i x_j$ are entirely zero
- Hashing techniques are used to handle sparse $\boldsymbol{w}$

Example: Dependency Parsing (Cont'd)

	LIBSVM		LIBLINEAR	
	RBF	Poly	Linear	Poly
Training time	3h34m53s	3h21m51s	3m36s	3m43s
Parsing speed	0.7x	1x	1652x	103x
UAS	89.92	91.67	89.11	91.71
LAS	88.55	90.60	88.07	90.71

- We get faster training/testing, but maintain good accuracy
- See detailed discussion in Chang et al. (2010)

Discussion

- In the above example, we use all pairs
- This is fine for some applications, but # features may become too large
- People have proposed projection or hashing techniques to use fewer features as approximations

  Examples: Kar and Karnick (2012); Pham and Pagh (2013)
- This has been used in computational adversitements (Chapelle et al., 2014)

Outline

- Introduction
- Optimization methods
- Extension of linear classification
- Big-data linear classification
- Conclusions and future directions

Big-data Linear Classification

- Nowadays data can be easily larger than memory capacity
- Disk-level linear classification: Yu et al. (2012) and subsequent developments
- Distributed linear classification: recently an active research topic
- Example: we can parallelize the 2nd-order method discussed earlier. Recall the Hessian-vector product

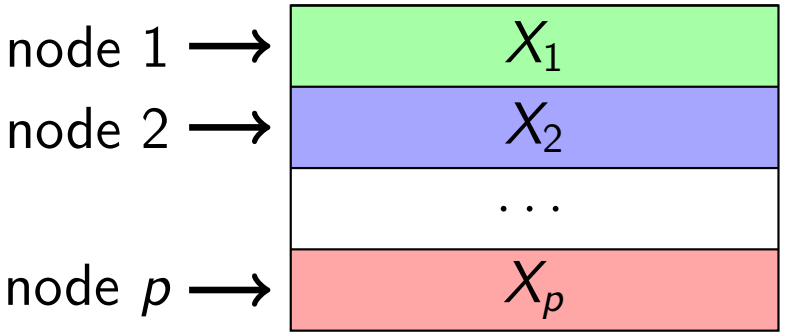
$$\nabla^2 f(\boldsymbol{w})\boldsymbol{s} = \boldsymbol{s} + C \cdot X^T(D(X\boldsymbol{s}))$$

Parallel Hessian-vector Product

- Hessian-vector products are the computational bottleneck

$$X^T D X \boldsymbol{s}$$

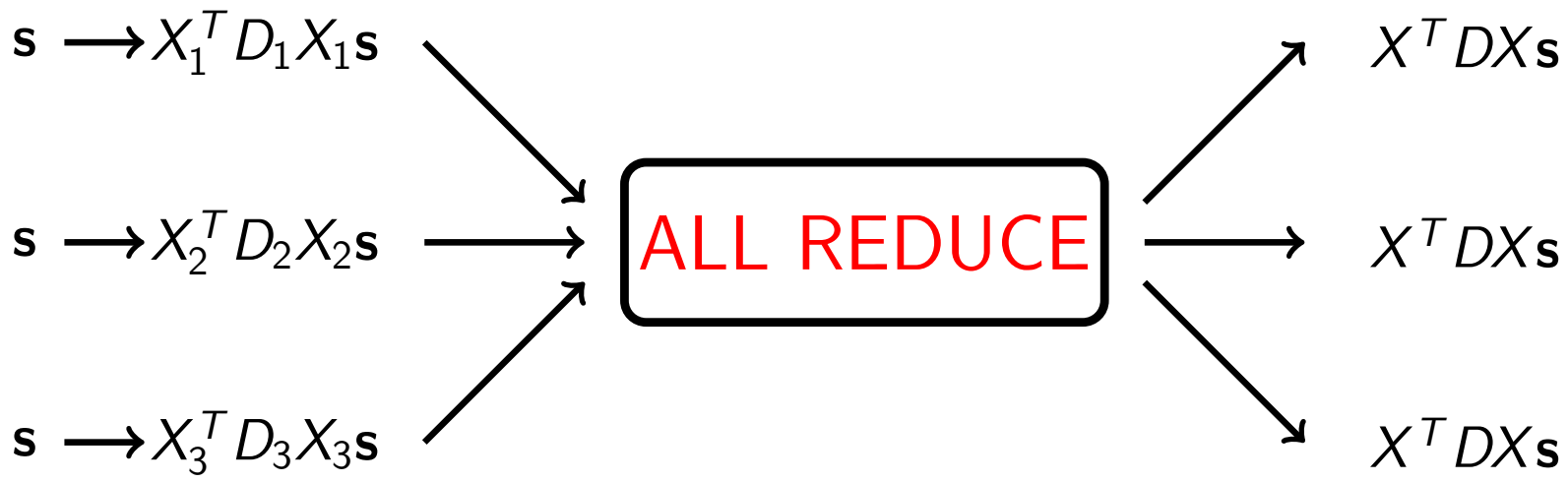
- Data matrix X is now distributedly stored



$$X^T D X \boldsymbol{s} = X_1^T D_1 X_1 \boldsymbol{s} + \cdots + X_p^T D_p X_p \boldsymbol{s}$$

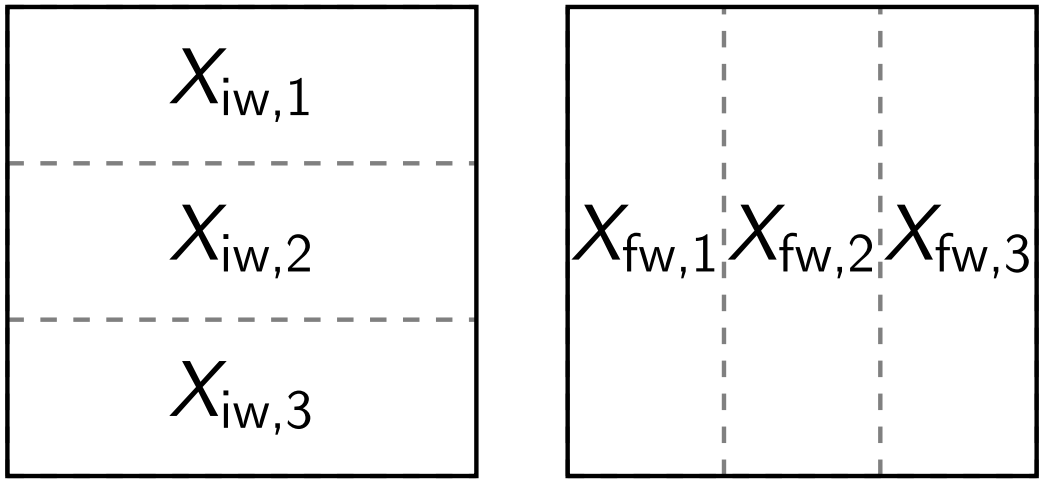
Parallel Hessian-vector Product (Cont'd)

We use allreduce to let every node get $X^T DX\mathbf{s}$



Allreduce: reducing all vectors ($X_i^T D_i X_i \boldsymbol{x}, \forall i$) to a single vector ($X^T DX\boldsymbol{s} \in R^n$) and then sending the result to every node

Instance-wise and Feature-wise Data Splits



- Feature-wise: each machine calculates part of the Hessian-vector product

$$(\nabla^2 f(\boldsymbol{w})\mathbf{v})_{\text{fw},1} = \boldsymbol{v}_1 + CX_{\text{fw},1}^T D(X_{\text{fw},1}\boldsymbol{v}_1 + \cdots + X_{\text{fw},p}\boldsymbol{v}_p)$$

Instance-wise and Feature-wise Data Splits (Cont'd)

- $X_{\text{fw},1}\boldsymbol{v}_1 + \cdots + X_{\text{fw},p}\boldsymbol{v}_p \in R^l$ must be available on all nodes (by allreduce)
- Data moved per Hessian-vector product

  Instance-wise: $O(n)$, Feature-wise: $O(l)$

Experiments

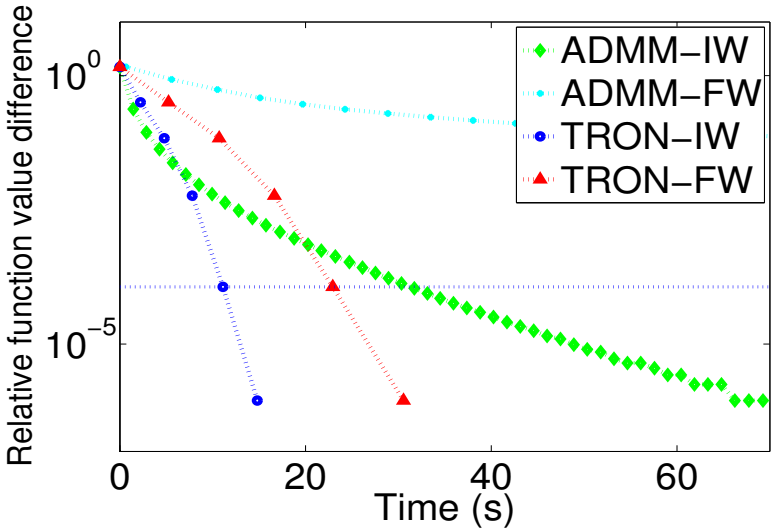
- Two sets:

Data set	l	n	#nonzeros
epsilon	400,000	2,000	800,000,000
webspam	350,000	16,609,143	1,304,697,446

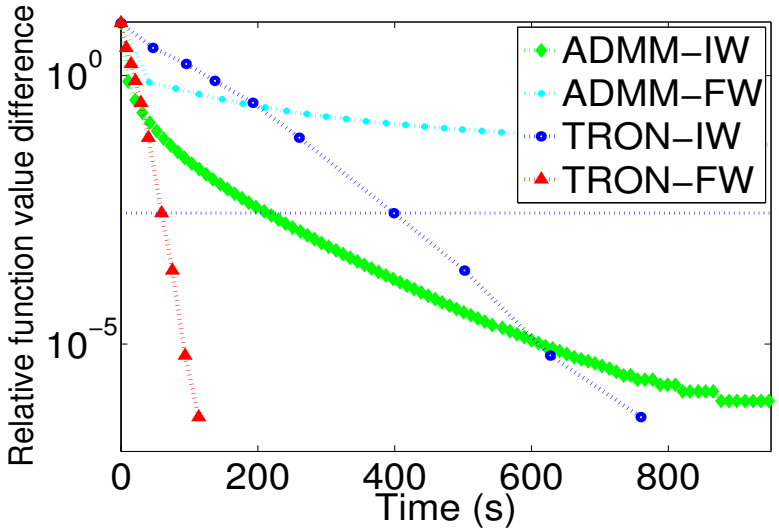
- For results of more sets, see Zhuang et al. (2014)
- We use Amazon AWS
- We compare

  1. TRON: Trust-region Newton method
  2. ADMM: alternating direction method of multipliers (Boyd et al., 2011; Zhang et al., 2012)

Experiments (Cont'd)



epsilon



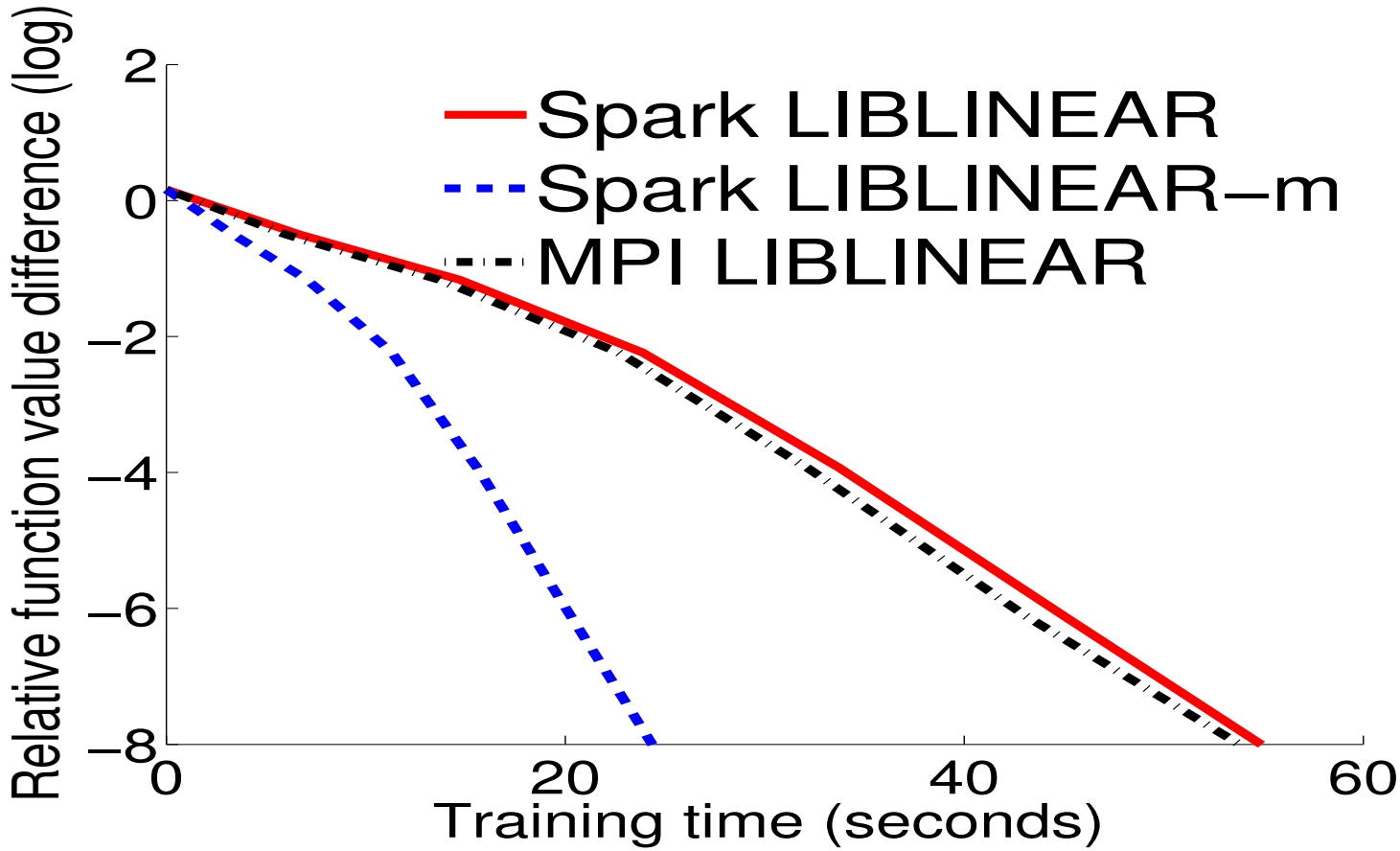
webspam

- 16 machines are used
- Horizontal line: test accuracy has stabilized
- TRON has faster convergence than ADMM
- Instance-wise and feature-wise splits useful for $l \gg n$ and $l \ll n$, respectively

Programming Frameworks

- We use MPI for the above experiments
- How about others like MapReduce?
- MPI is more efficient, but has no fault tolerance
- In contrast, MapReduce is slow for iterative algorithms due to heavy disk I/O
- Many new frameworks are being actively developed
  1. Spark (Zaharia et al., 2010)
  2. REEF (Chun et al., 2013)
- Selecting suitable frameworks for distributed classification isn't that easy!

A Comparison Between MPI and Spark



We use the data set epsilon (8 nodes). Spark is slower, but in general competitive

Outline

- Introduction
- Optimization methods
- Extension of linear classification
- Big-data linear classification
- Conclusions and future directions

Resources on Linear Classification

- Since 2007, we have been actively developing the software LIBLINEAR for linear classification

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

  It's now widely used in Internet companies
- An earlier survey on linear classification is Yuan et al. (2012)

  Recent Advances of Large-scale Linear Classification. *Proceedings of IEEE*, 2012

  It contains many references on this subject

Distributed LIBLINEAR

- We recently released an extension of LIBLINEAR for distributed classification
- See `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/distributed-liblinear`
- We support both MPI and Spark
- The development is still in an early stage. Your comments are very welcome.

Conclusions

- Linear classification is an old topic; but recently there are new and interesting applications
- Kernel methods are still useful for many applications, but linear classification + feature engineering are suitable for some others
- Advantages of linear: easier feature engineering
- We expect that linear classification can be widely used in situations ranging from small-model to big-data classification

Acknowledgments

- Many students have contributed to our research on large-scale linear classification
- We also thank the partial support from National Science Council of Taiwan