Optimization Methods for Large-scale Linear Classification

Chih-Jen Lin
Department of Computer Science
National Taiwan University

Talk at University of Rome "La Sapienza," June 25, 2013

- Part of this talk is based on our recent survey paper in *Proceedings of IEEE*, 2012

  G.-X. Yuan, C.-H. Ho, and C.-J. Lin. Recent Advances of Large-scale Linear Classification.
- It's also related to our development of the software LIBLINEAR

  `www.csie.ntu.edu.tw/~cjlin/liblinear`
- Due to time constraints, we will give overviews instead of deep technical details.

Outline

- Introduction
- Binary linear classification
- Optimization Methods: Second-order Methods
- Optimization Methods: First-order Methods
- Experiments
- Big-data Machine Learning
- Conclusions

Outline

Linear and Nonlinear Classification

Some popular methods such as SVM and logistic regression can be used in two ways

- Kernel methods: data mapped to another space

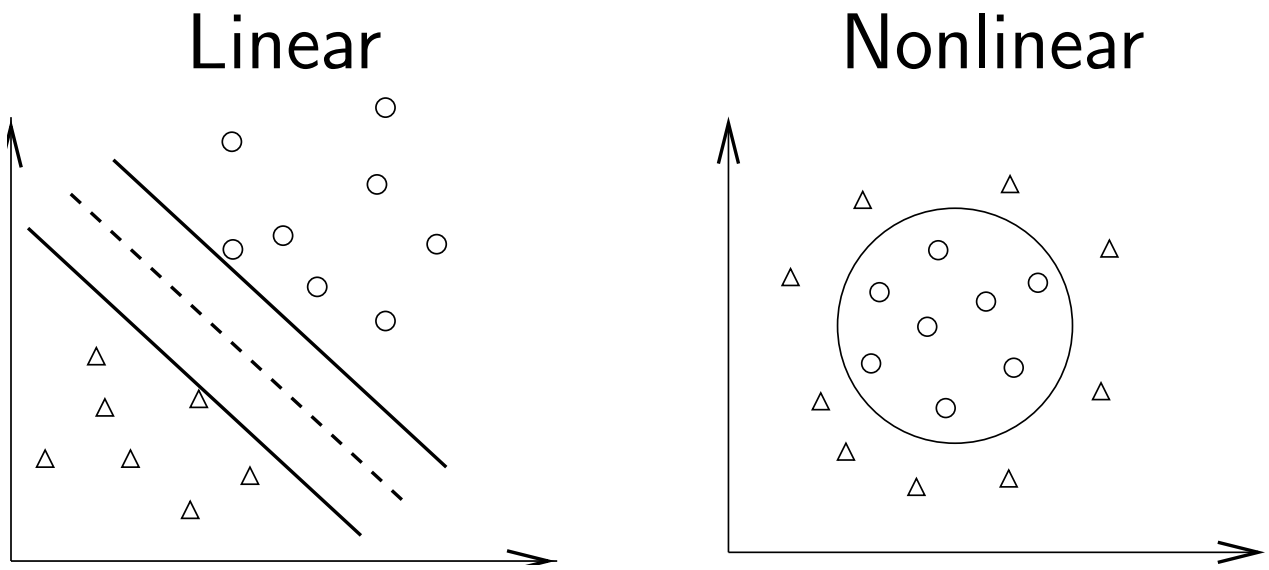
$$\mathbf{x} \Rightarrow \phi(\mathbf{x})$$

  $\phi(\mathbf{x})^T\phi(\mathbf{y})$ easily calculated; no good control on $\phi(\cdot)$
- Linear classification + feature engineering:

  We have $\mathbf{x}$ without mapping. Alternatively, we can say that $\phi(\mathbf{x})$ is our $\mathbf{x}$; full control on $\mathbf{x}$ or $\phi(\mathbf{x})$

We refer to them as nonlinear and linear classifiers; we will focus on linear here

Linear and Nonlinear Classification



By linear we mean data not mapped to a higher dimensional space

Original: [height, weight]

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

Linear and Nonlinear Classification (Cont'd)

- Given training data $\{y_i, \mathbf{x}_i\}, \mathbf{x}_i \in R^n, i = 1, \ldots, l$, $y_i = \pm 1$
  l: # of data, n: # of features
- Linear: find $(\mathbf{w}, b)$ such that the decision function is

$$\operatorname{sgn}\left(\mathbf{w}^T\mathbf{x} + b\right)$$

- Nonlinear: map data to $\phi(\mathbf{x}_i)$. The decision function becomes

$$\operatorname{sgn}\left(\mathbf{w}^T\phi(\mathbf{x}) + b\right)$$

- Later b is omitted

Why Linear Classification?

- If $\phi(\mathbf{x})$ is high dimensional, $\mathbf{w}^T\phi(\mathbf{x})$ is expensive
- Kernel methods:

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

$$\text{New decision function: } \operatorname{sgn}\left(\sum_{i=1}^{l} \alpha_i K(\mathbf{x}_i, \mathbf{x})\right)$$

- Special $\phi(\mathbf{x})$ so that calculating $K(\mathbf{x}_i, \mathbf{x}_j)$ is easy
- Example:

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

Why Linear Classification? (Cont'd)

- Prediction

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

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

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

- Nonlinear: more powerful to separate 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
- Recently linear classification is a popular research topic

Outline

Binary Linear Classification

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

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

- $\mathbf{w}^T\mathbf{w}/2$: regularization term
- $\xi(\mathbf{w}; \mathbf{x}, y)$: loss function: we hope $y\mathbf{w}^T\mathbf{x} > 0$
- C: regularization parameter

Loss Functions

- Some commonly used ones:

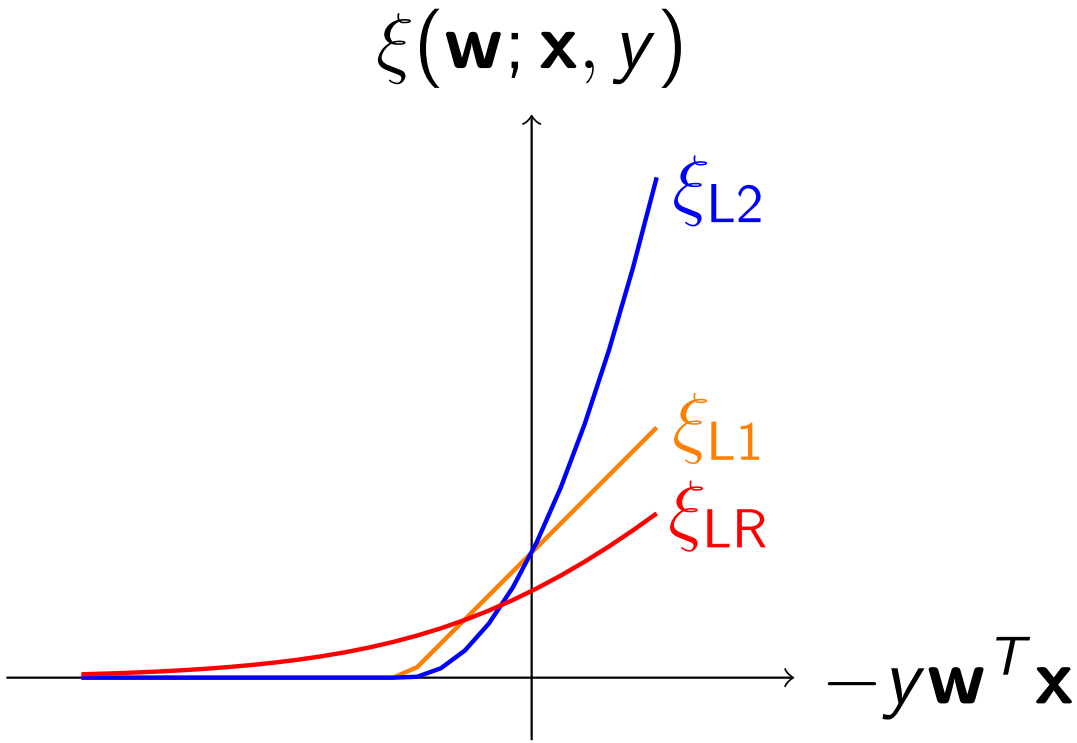
$$\xi_{\text{L1}}(\mathbf{w}; \mathbf{x}, y) \equiv \max(0, 1 - y\mathbf{w}^T\mathbf{x}), \quad (1)$$

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

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

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

Loss Functions (Cont'd)



They are similar in terms of performance

Loss Functions (Cont'd)

However,

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

Many optimization methods can be used

Outline

Truncated Newton Method

- Newton direction

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

- This is the same as solving Newton linear system

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

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

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

- For document data, n can be millions or more

Using Special Properties of Data Classification

- But Hessian has a special form

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

- D diagonal. For logistic regression,

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

- X: data, # instances $\times$ # features

$$X = [\mathbf{x}_1, \ldots, \mathbf{x}_l]^T$$

Using Special Properties of Data Classification (Cont'd)

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

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

- Therefore, we have a Hessian-free approach
- In Lin et al. (2008), we use the trust-region + CG approach by Steihaug (1983)
- Quadratic convergence is achieved

Training L2-loss SVM

- The loss function is differentiable but not twice differentiable

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

- We can use generalized Hessian (Mangasarian, 2002)
- Works well in practice, but no theoretical quadratic convergence

Outline

- First-order methods are popular in data classification
- Reason: no need to accurately solve the optimization problem
- We consider L1-loss SVM as an example here, though same methods may be extended to L2 and logistic loss

SVM Dual

- From primal dual relationship

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

where

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

and

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

Dual Coordinate Descent

- Very simple: minimizing one variable at a time
- While α not optimal

  For $i = 1, \ldots, l$

$$\min_{\alpha_i} f(\ldots, \alpha_i, \ldots)$$

- A classic optimization technique
- Traced back to Hildreth (1957) if constraints are not considered

The Procedure

- Given current $\boldsymbol{\alpha}$. Let $\mathbf{e}_i = [0, \ldots, 0, 1, 0, \ldots, 0]^T$.

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

- Without constraints

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

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

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

The Procedure (Cont'd)

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

$$= \sum_{j=1}^{l} y_i y_j \mathbf{x}_i^T \mathbf{x}_j \alpha_j - 1$$

- Directly calculating gradients costs $O(ln)$
  l:# data, n: # features
- For linear SVM, define

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

- Easy gradient calculation: costs $O(n)$

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

The Procedure (Cont'd)

- All we need is to maintain $\mathbf{u}$

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

- If

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

then

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

Also costs $O(n)$

Algorithm

- Given initial $\boldsymbol{\alpha}$ and find

$$\mathbf{u} = \sum_i y_i \alpha_i \mathbf{x}_i.$$

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

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

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

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

  (c) If α_i can be changed

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

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

Analysis

- Convergence; from Luo and Tseng (1992)

$$f(\boldsymbol{\alpha}^{k+1}) - f(\boldsymbol{\alpha}^*) \leq \mu(f(\boldsymbol{\alpha}^k) - f(\boldsymbol{\alpha}^*)), \forall k \geq k_0.$$

  $\boldsymbol{\alpha}^*$: optimal solution

- Recently we prove the result with $k_0 = 1$ (Wang and Lin, 2013)
- Difficulty: the objective function is convex only rather than strictly convex

Careful Implementation

Some techniques can improve the running speed

- Shrinking: remove α_i if it is likely to be bounded at the end

  Easier to conduct shrinking than the kernel case (details not shown)
- Order of sub-problems being minimized

$$\alpha_1 \to \alpha_2 \to \cdots \to \alpha_l$$

Can use any random order at each outer iteration

$$\alpha_{\pi(1)} \to \alpha_{\pi(2)} \to \cdots \to \alpha_{\pi(l)}$$

Very effective in practice

Difference from the Kernel Case

- What if coordinate descent methods are applied to kernel classifiers?
- Recall the gradient is

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

but we cannot do this for kernel because

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

is not separated

- If using kernel, the cost of calculating $\nabla_i f(\boldsymbol{\alpha})$ must be $O(ln)$

Difference from the Kernel Case (Cont'd)

- This difference is similar to our earlier discussion on the prediction cost

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

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

- However, if $O(ln)$ cost is spent, the whole $\nabla f(\boldsymbol{\alpha})$ can be maintained (details not shown here)
- In contrast, the setting of using $\mathbf{u}$ knows $\nabla_i f(\boldsymbol{\alpha})$ rather than the whole $\nabla f(\boldsymbol{\alpha})$

Difference from the Kernel Case (Cont'd)

- In existing coordinate descent methods for kernel classifiers, people also use $\nabla f(\alpha)$ information to select variable for update
- Recall there are two types of coordinate descent methods

  Gauss-Seidel: sequential selection of variables

  Gauss-Southwell: greedy selection of variables
- To do greedy selection, usually the whole gradient must be available

Difference from the Kernel Case (Cont'd)

- Existing coordinate descent methods for linear $\Rightarrow$ related to Gauss-Seidel

  Existing coordinate descent methods for kernel $\Rightarrow$ related to Gauss-Southwell

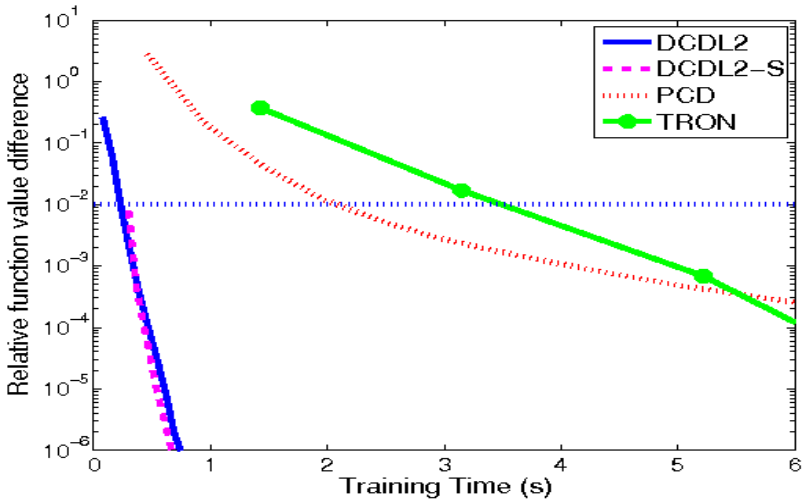
Outline

Comparisons

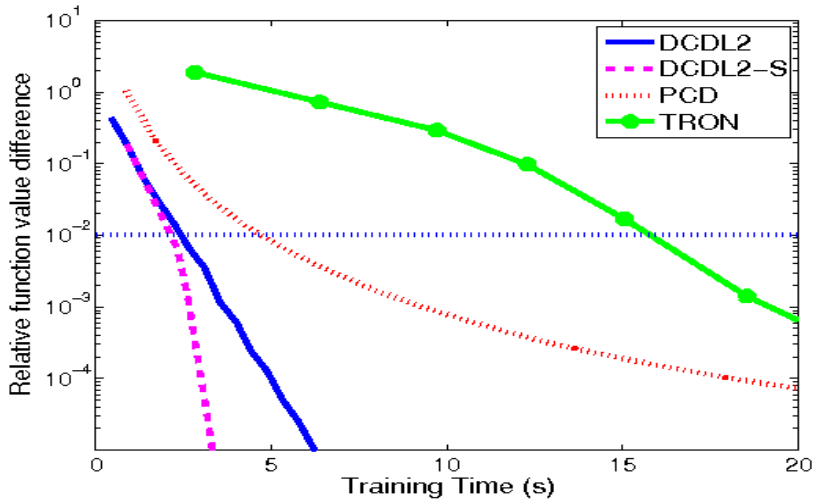
L2-SVM is used

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

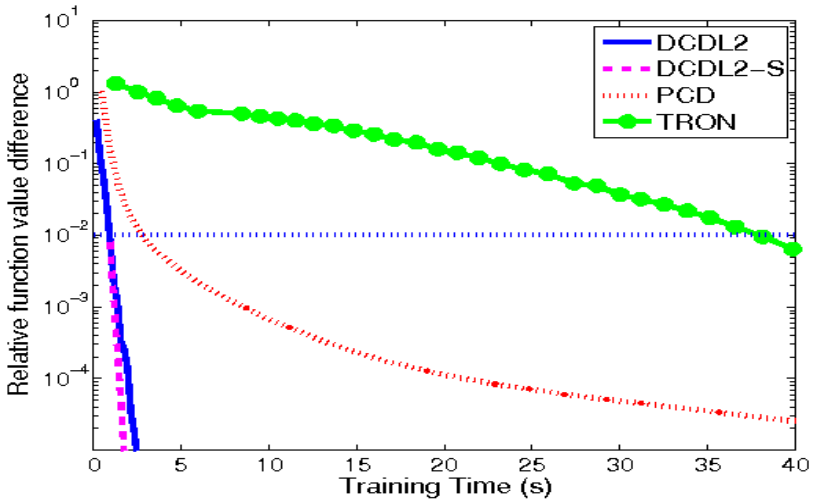
Objective values (Time in Seconds)



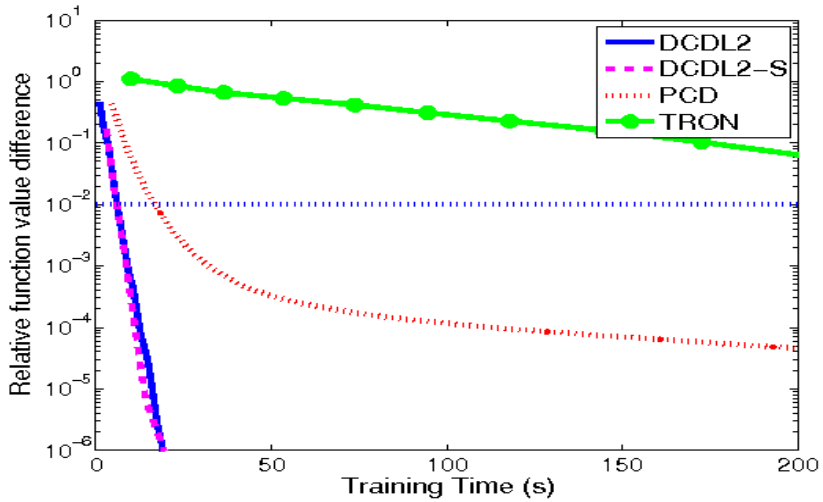
news20



rcv1



yahoo-japan



yahoo-korea

Analysis

- Dual coordinate descents are very effective if # data, # features are large

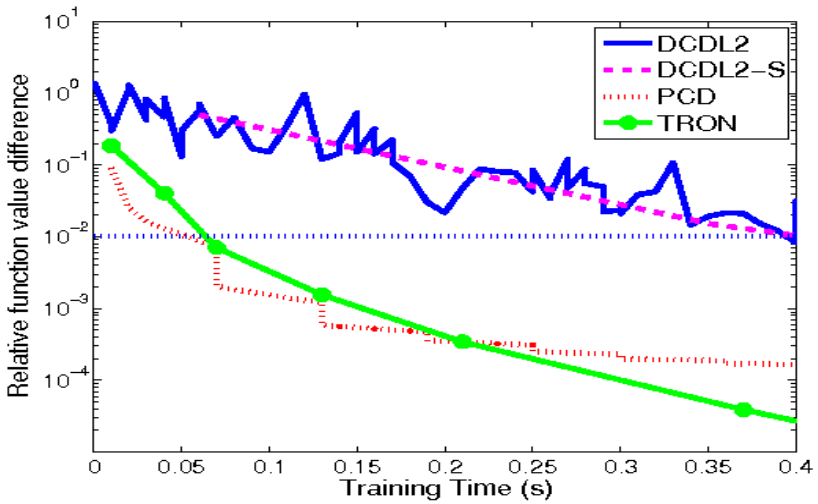
  Useful for document classification
- Half million data in a few seconds
- However, it is less effective if

  # features small: should solve primal; or

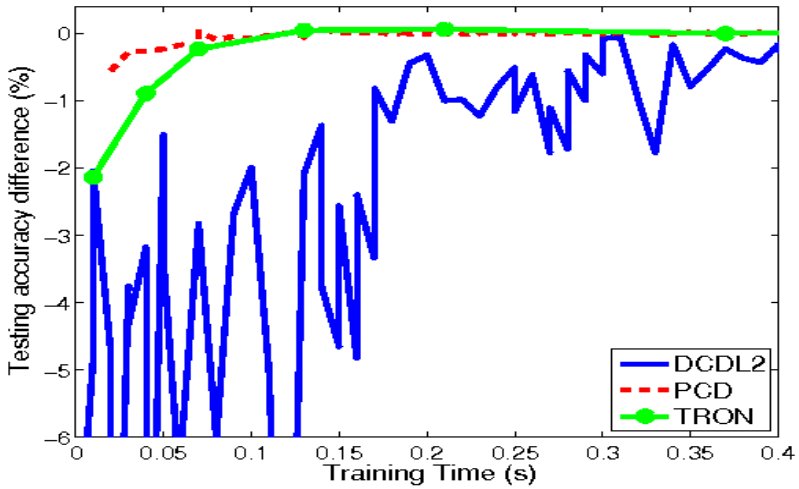
  large penalty parameter C; problems are more ill-conditioned

An Example When # Features Small

- # instance: 32,561, # features: 123



Objective value



Accuracy

Outline

Big-data Machine Learning

- Data distributedly stored
- This is a new topic and many research works are still going on
- You may ask what the difference is from distributed optimization
- They are related, but now the algorithm must avoid expensive data accesses

Big-data Machine Learning (Cont'd)

- Issues for parallelization

  - Many methods (e.g., stochastic gradient descent or coordinate descent) are inherently sequential

  - Communication cost is a concern

Simple Distributed Linear Classification I

- Bagging: train several subsets and ensemble results

  - Useful in distributed environments; each node $\Rightarrow$ a subset

  - Example: Zinkevich et al. (2010)

- Some results by averaging models

	yahoo-korea	kddcup10	webspam	epsilson
Using all	87.29	89.89	99.51	89.78
Avg. models	86.08	89.64	98.40	88.83

- Using all: solves a single linear SVM

Simple Distributed Linear Classification II

- Avg. models: each node solves a linear SVM on a subset
- Slightly worse but in general OK

ADMM by Boyd et al. (2011) I

- Recall the SVM problem (bias term b omitted)

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)$$

- An equivalent optimization problem

$$\min_{\mathbf{w}_1,\dots,\mathbf{w}_m,\mathbf{z}} \quad \frac{1}{2}\mathbf{z}^T\mathbf{z} + C\sum_{j=1}^{m}\sum_{i\in B_j}\max(0, 1 - y_i\mathbf{w}_j^T\mathbf{x}_i) + \frac{\rho}{2}\sum_{j=1}^{m}\|\mathbf{w}_j - \mathbf{z}\|^2$$

$$\text{subject to} \quad \mathbf{w}_j - \mathbf{z} = \mathbf{0}, \forall j$$

ADMM by Boyd et al. (2011) II

- The key is that

$$\mathbf{z} = \mathbf{w}_1 = \cdots = \mathbf{w}_m$$

  are all optimal $\mathbf{w}$
- This optimization problem was proposed in 1970s, but is now applied to distributed machine learning
- Each node has a subset B_j and updates $\mathbf{w}_j$
- Only $\mathbf{w}_1, \ldots, \mathbf{w}_m$ must be collected

  Data are not moved; less communication cost
- Still, we cannot afford too many iterations because of communication cost

Vowpal_Wabbit (Langford et al., 2007) I

- It started as a linear classification package on a single computer
- After version 6.0, Hadoop support has been provided
- A hybrid approach: parallel SGD initially and switch to LBFGS (quasi Newton)
- They argue that AllReduce is a more suitable operation than MapReduce
- What is AllReduce?

  Every node starts with a value and ends up with the sum at all nodes

Vowpal_Wabbit (Langford et al., 2007) II

- In Agarwal et al. (2012), the authors argue that many machine learning algorithms can be implemented using AllReduce

  LBFGS is an example
- They train 17B samples with 16M features on 1K nodes $\Rightarrow$ 70 minutes

Outline

- Introduction
- Binary linear classification
- Optimization Methods: Second-order Methods
- Optimization Methods: First-order Methods
- Experiments
- Big-data Machine Learning
- Conclusions

Conclusions

- Linear classification is an old topic; but recently there are new applications and large-scale challenges
- The optimization problem can be solved by many existing techniques
- However, some machine-learning aspects must be considered
- In particular, data access may become a bottleneck in large-scale scenarios
- Overall, linear classification is still an on-going and exciting research area