Optimization and Machine Learning

Chih-Jen Lin
Department of Computer Science
National Taiwan University

Talk at Summer School on Optimization, Big Data and Applications, July, 2017

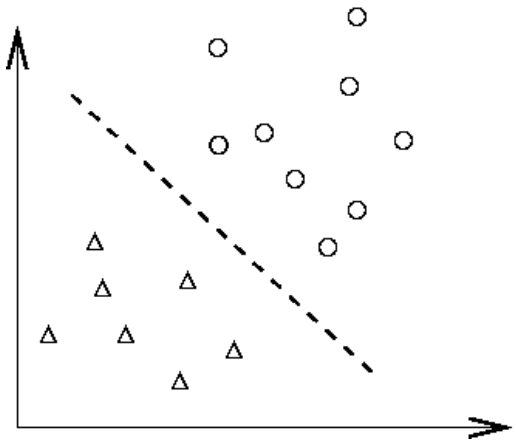
Outline

Talk slides are available at `http://www.csie.ntu.edu.tw/~cjlin/talks/italy_optml.pdf`

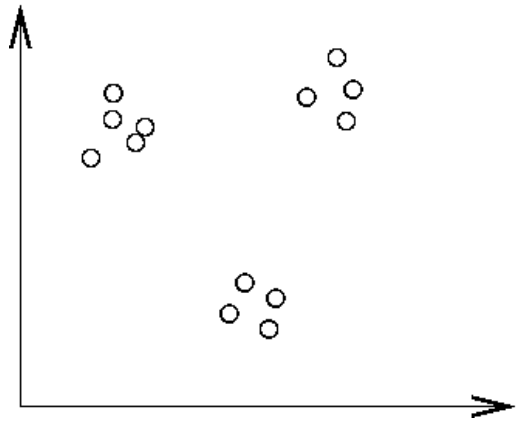
Outline

What is Machine Learning?

- Extract knowledge from data and make predictions
- Representative tasks: classification, clustering, and others



Classification



Clustering

- We will focus on classification

Data Classification

- Given training data in different classes (labels known)

  Predict test data (labels unknown)
- Classic example
  1. Find a patient's blood pressure, weight, etc.
  2. After several years, know if he/she recovers
  3. Build a machine learning model
  4. New patient: find blood pressure, weight, etc
  5. Prediction
- Two main stages: training and testing

Why Is Optimization Used?

- Usually the goal of classification is to

  minimize the test error

- Therefore, many classification methods solve optimization problems

Optimization and Machine Learning

- Standard optimization packages may be directly applied to machine learning applications
- However, efficiency and scalability are issues
- Very often machine learning knowledge must be considered in designing suitable optimization methods

Optimization and Machine Learning (Cont'd)

- Sometimes optimization researchers fail to make real impact because of not knowing the differences between the two areas
- I like to talk about the connection between these two areas because I was trained as an optimization researcher but now work on machine learning

Optimization and Machine Learning (Cont'd)

- In this talk, I will discuss some lessons learned in developing two popular packages
  - LIBSVM: 2000–now
    A library for support vector machines
  - LIBLINEAR: 2007–now
    A library for large linear classification
- Let me shamelessly say a bit about how they have been widely used

Optimization and Machine Learning (Cont'd)

- LIBSVM is probably the most widely used SVM package. Its implementation paper has been cited more than $32,000$ times (Google Scholar, 5/2017)
- LIBLINEAR is popularly used in Internet companies for large-scale linear classification

Minimizing Training Errors

- Basically a classification method starts with minimizing the training errors

$$\min_{\text{model}} \quad (\text{training errors})$$

- That is, all or most training data with labels should be correctly classified by our model
- A model can be a decision tree, a support vector machine, a neural network, or other types

Minimizing Training Errors (Cont'd)

- We consider the model to be a vector $\boldsymbol{w}$
- That is, the decision function is

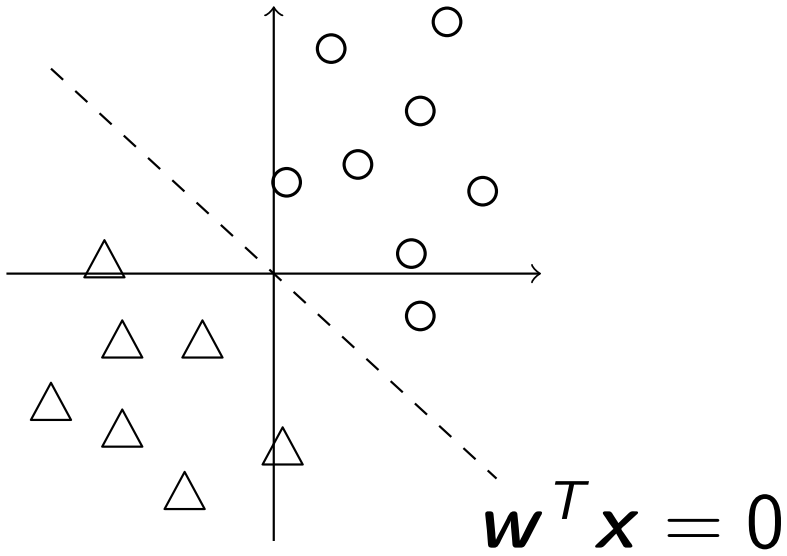
$$\text{sgn}(\boldsymbol{w}^T\boldsymbol{x})$$

- For any data, $\boldsymbol{x}$, the predicted label is

$$\begin{cases} 1 & \text{if } \boldsymbol{w}^T\boldsymbol{x} \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Minimizing Training Errors (Cont'd)

- The two-dimensional situation



- This seems to be quite restricted, but practically $\boldsymbol{x}$ is in a much higher dimensional space

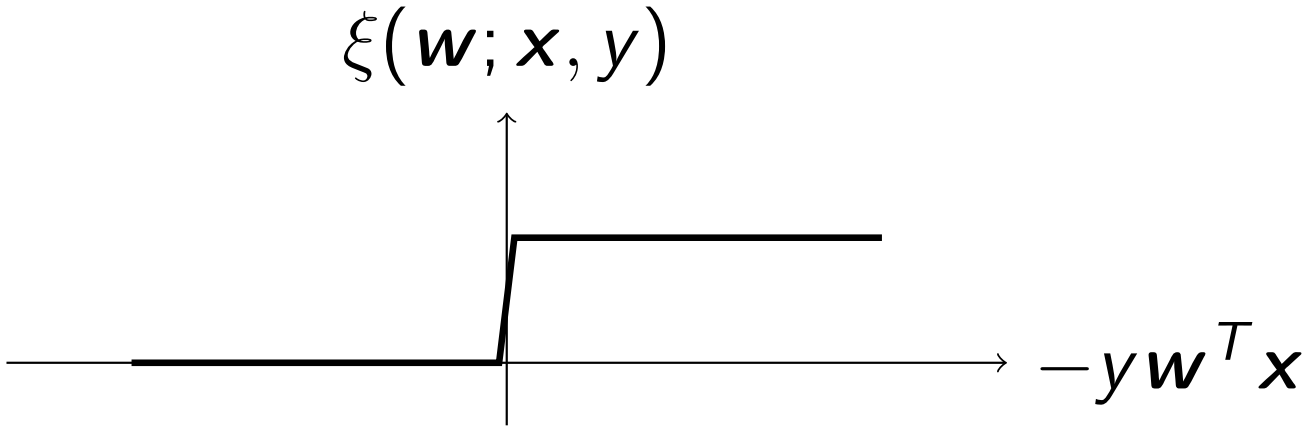
Minimizing Training Errors (Cont'd)

- To characterize the training error, we need a loss function $\xi(\boldsymbol{w}; \boldsymbol{x}, y)$ for each instance $(\boldsymbol{x}, y)$
- Ideally we should use 0–1 training loss:

$$\xi(\boldsymbol{w}; \boldsymbol{x}, y) = \begin{cases} 1 & \text{if } y\boldsymbol{w}^T\boldsymbol{x} < 0, \\ 0 & \text{otherwise} \end{cases}$$

Minimizing Training Errors (Cont'd)

- However, this function is discontinuous. The optimization problem becomes difficult



- We need continuous approximations

Loss Functions

- Some commonly used ones:

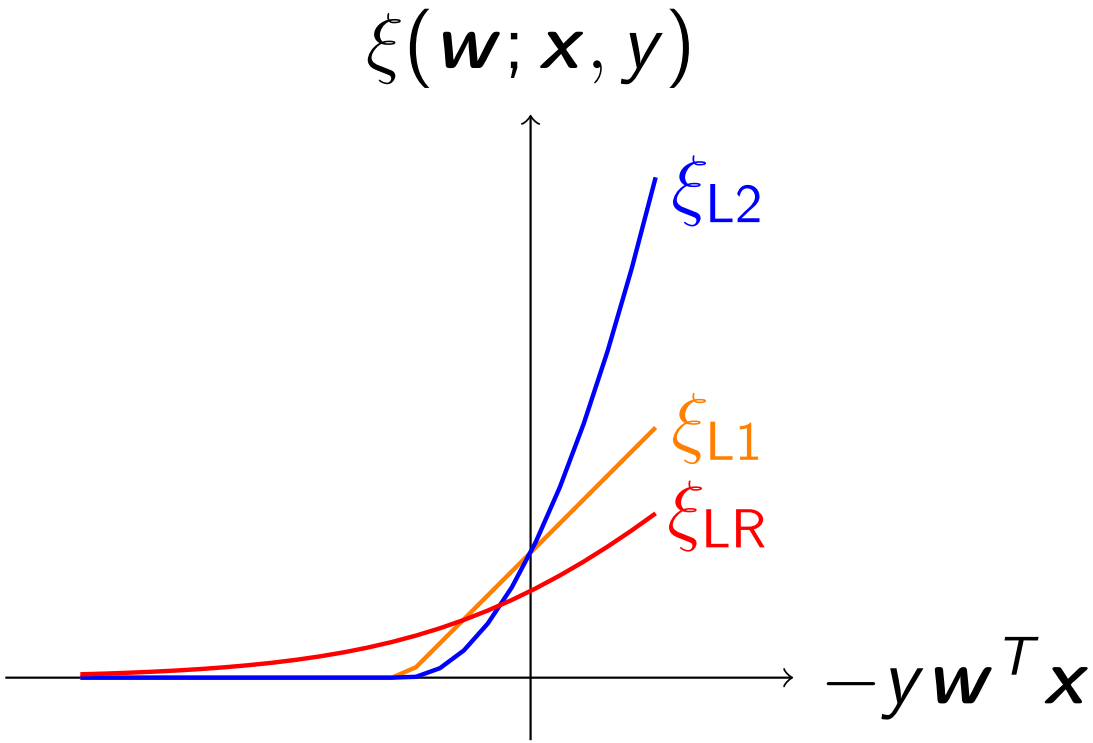
$$\xi_{\text{L1}}(\boldsymbol{w}; \boldsymbol{x}, y) \equiv \max(0, 1 - y\boldsymbol{w}^T\boldsymbol{x}), \quad (1)$$

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

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

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

Loss Functions (Cont'd)



Their performance is usually similar

Loss Functions (Cont'd)

These loss functions have different differentiability

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

Loss Functions (Cont'd)

- However, minimizing training losses may not give a good model for future prediction
- Overfitting occurs

Overfitting

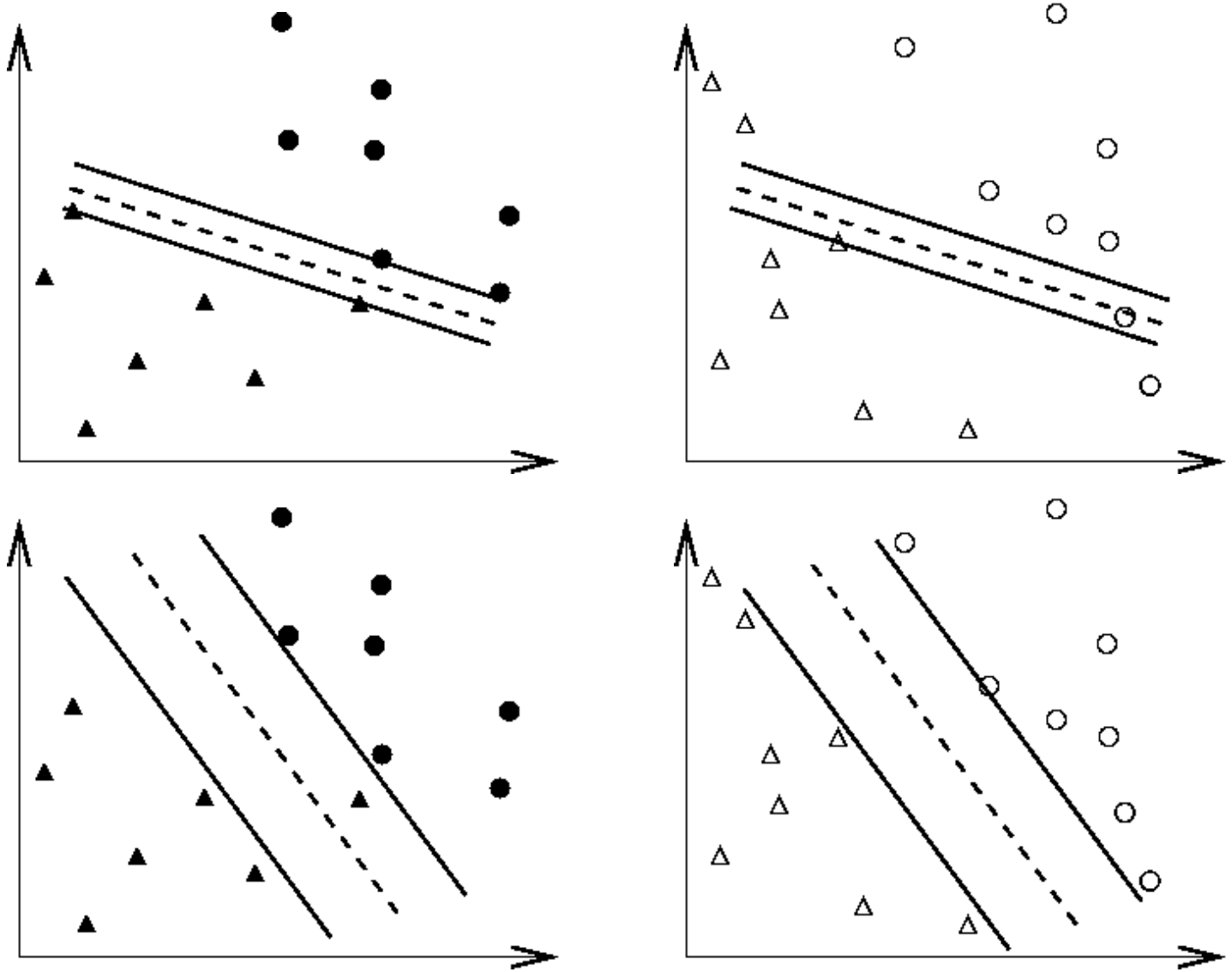
- See the illustration in the next slide
- For classification,

  You can easily achieve 100% training accuracy
- This is useless
- When training a data set, we should

  Avoid underfitting: small training error

  Avoid overfitting: small testing error

● and ▲: training; ○ and △: testing



Regularization

- To minimize the training error we manipulate the $\boldsymbol{w}$ vector so that it fits the data
- To avoid overfitting we need a way to make $\boldsymbol{w}$'s values less extreme.
- One idea is to make $\boldsymbol{w}$ values closer to zero
- We can add, for example,

$$\frac{\boldsymbol{w}^T\boldsymbol{w}}{2} \quad \text{or} \quad \|\boldsymbol{w}\|_1$$

to the function that is minimized

Regularized 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

Outline

Kernel Methods

- Kernel methods are a class of classification techniques where major operations are conducted by kernel evaluations
- A representative example is support vector machine (Boser et al., 1992; Cortes and Vapnik, 1995)

Support Vector Classification

- Training data $(\boldsymbol{x}_i, y_i), i = 1, \ldots, l, \boldsymbol{x}_i \in R^n, y_i = \pm 1$
- Minimizing training losses with regularization

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

- Note that here we add a bias term b so the decision function is

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

Support Vector Classification (Cont'd)

- Then the hyperplane does not pass through 0
- If n (# of features) is small, b may be important. Otherwise, it's not
- There are also historical reasons
- In our discussion we sometimes include b but sometimes do not

Mapping Data to a Higher Dimensional Space

- To better separate the data, we map data to a higher dimensional space

$$\phi(\boldsymbol{x}) = [\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), \ldots]^T.$$

- For example,

$$\frac{\text{weight}}{\text{height}^2}$$

is a useful new feature to check if a person overweights or not

Difficulties After Mapping Data to a High-dimensional Space

- # variables in $\boldsymbol{w}$ = dimensions of $\phi(\boldsymbol{x})$
- Infinite variables if $\phi(\boldsymbol{x})$ is infinite dimensional
- Cannot do an infinite-dimensional inner product for predicting a test instance

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

- Use kernel trick to go back to a finite number of variables

Support Vector Classification (Cont'd)

- The dual problem (finite # variables)

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \boldsymbol{e}^T \boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \le \alpha_i \le C, i = 1, \ldots, l \\ & \mathbf{y}^T \boldsymbol{\alpha} = 0, \end{aligned}$$

  where $Q_{ij} = y_i y_j \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$ and $\boldsymbol{e} = [1, \ldots, 1]^T$

- At optimum

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

- Kernel: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$; closed form

  Example: Gaussian (RBF) kernel: $e^{-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}$

Kernel Tricks

- It can be shown that at optimum, $\boldsymbol{w}$ is a linear combination of training data

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

  Proofs not provided here.

- Special $\phi(\boldsymbol{x})$ such that the decision function becomes

$$\begin{aligned}\operatorname{sgn}(\boldsymbol{w}^T \phi(\boldsymbol{x}) + b) &= \operatorname{sgn}\left(\sum_{i=1}^{l} y_i \alpha_i \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}) + b\right) \\ &= \operatorname{sgn}\left(\sum_{i=1}^{l} y_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b\right)\end{aligned}$$

Kernel Tricks (Cont'd)

- $\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$ needs a closed form
- Example: $\boldsymbol{x}_i \in R^3, \phi(\boldsymbol{x}_i) \in R^{10}$

$$\phi(\boldsymbol{x}_i) = [1, \sqrt{2}(x_i)_1, \sqrt{2}(x_i)_2, \sqrt{2}(x_i)_3, (x_i)_1^2,$$
$$(x_i)_2^2, (x_i)_3^2, \sqrt{2}(x_i)_1(x_i)_2, \sqrt{2}(x_i)_1(x_i)_3, \sqrt{2}(x_i)_2(x_i)_3]^T$$

Then $\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j) = (1 + \boldsymbol{x}_i^T\boldsymbol{x}_j)^2$.

- Kernel: $K(\boldsymbol{x}, \mathbf{y}) = \phi(\boldsymbol{x})^T\phi(\mathbf{y})$; common kernels:

$$e^{-\gamma\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}, \text{ (Radial Basis Function)}$$
$$(\boldsymbol{x}_i^T\boldsymbol{x}_j/a + b)^d \text{ (Polynomial kernel)}$$

$K(\boldsymbol{x}, \mathbf{y})$ can be inner product in infinite dimensional space. Assume $x \in R^1$ and $\gamma > 0$.

$$e^{-\gamma\|x_i - x_j\|^2} = e^{-\gamma(x_i - x_j)^2} = e^{-\gamma x_i^2 + 2\gamma x_i x_j - \gamma x_j^2}$$

$$= e^{-\gamma x_i^2 - \gamma x_j^2}\left(1 + \frac{2\gamma x_i x_j}{1!} + \frac{(2\gamma x_i x_j)^2}{2!} + \frac{(2\gamma x_i x_j)^3}{3!} + \cdots\right)$$

$$= e^{-\gamma x_i^2 - \gamma x_j^2}\left(1 \cdot 1 + \sqrt{\frac{2\gamma}{1!}} x_i \cdot \sqrt{\frac{2\gamma}{1!}} x_j + \sqrt{\frac{(2\gamma)^2}{2!}} x_i^2 \cdot \sqrt{\frac{(2\gamma)^2}{2!}} x_j^2\right.$$

$$\left. + \sqrt{\frac{(2\gamma)^3}{3!}} x_i^3 \cdot \sqrt{\frac{(2\gamma)^3}{3!}} x_j^3 + \cdots\right) = \phi(x_i)^T \phi(x_j),$$

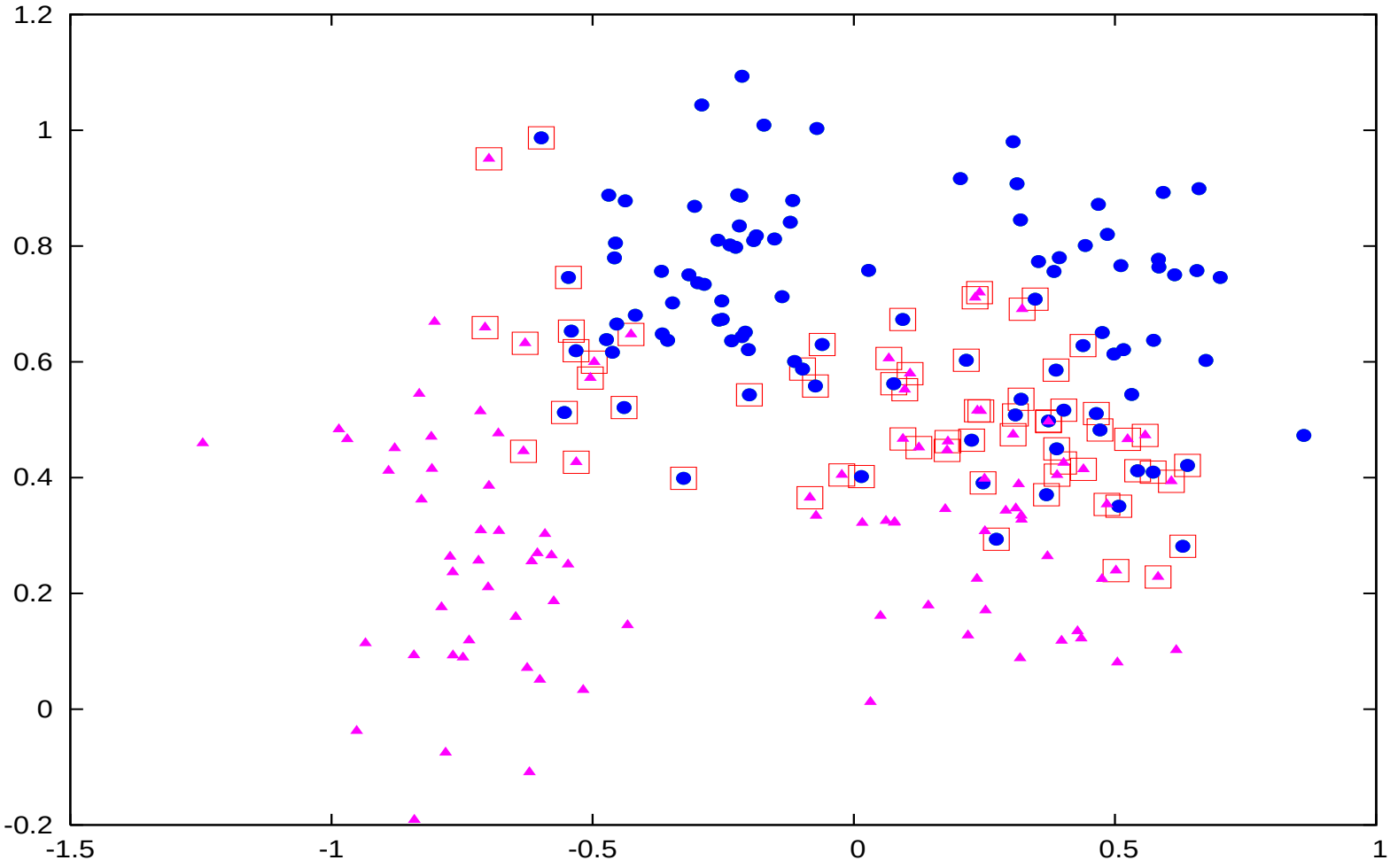
where

$$\phi(x) = e^{-\gamma x^2}\left[1, \sqrt{\frac{2\gamma}{1!}} x, \sqrt{\frac{(2\gamma)^2}{2!}} x^2, \sqrt{\frac{(2\gamma)^3}{3!}} x^3, \cdots\right]^T.$$

- We don't discuss the primal-dual relationship here
- In this lecture we focus more on optimization algorithms rather than optimization theory

Support Vector Classification (Cont'd)

Only $\boldsymbol{x}_i$ of $\alpha_i > 0$ used $\Rightarrow$ support vectors



Large Dense Quadratic Programming

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \boldsymbol{e}^T \boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \le \alpha_i \le C, i = 1, \dots, l \\ & \mathbf{y}^T \boldsymbol{\alpha} = 0 \end{aligned}$$

- $Q_{ij} \neq 0$, Q : an l by l fully dense matrix
- 50,000 training points: 50,000 variables:
  $(50,000^2 \times 8/2)$ bytes $=$ 10GB RAM to store Q

Large Dense Quadratic Programming (Cont'd)

- Tradition optimization methods cannot be directly applied here because Q cannot even be stored
- Currently, decomposition methods (a type of coordinate descent methods) are commonly used in practice

Outline

Decomposition Methods

- Working on some variables each time (e.g., Osuna et al., 1997; Joachims, 1998; Platt, 1998)
- Similar to coordinate-wise minimization
- Working set B, $N = \{1, \ldots, l\} \backslash B$ fixed
- Let the objective function be

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

Decomposition Methods (Cont'd)

- Sub-problem on the variable $\boldsymbol{d}_B$

$$\begin{aligned} \min_{\boldsymbol{d}_B} \quad & f(\begin{bmatrix} \boldsymbol{\alpha}_B \\ \boldsymbol{\alpha}_N \end{bmatrix} + \begin{bmatrix} \boldsymbol{d}_B \\ \mathbf{0} \end{bmatrix}) \\ \text{subject to} \quad & -\alpha_i \le d_i \le C - \alpha_i, \forall i \in B \\ & d_i = 0, \forall i \notin B, \\ & \mathbf{y}_B^T \boldsymbol{d}_B = 0 \end{aligned}$$

- The objective function of the sub-problem

$$\begin{aligned} & f(\begin{bmatrix} \boldsymbol{\alpha}_B \\ \boldsymbol{\alpha}_N \end{bmatrix} + \begin{bmatrix} \boldsymbol{d}_B \\ \mathbf{0} \end{bmatrix}) \\ =& \frac{1}{2} \boldsymbol{d}_B^T Q_{BB} \boldsymbol{d}_B + \nabla_B f(\boldsymbol{\alpha})^T \boldsymbol{d}_B + \text{constant}. \end{aligned}$$

Avoid Memory Problems

- Q_{BB} is a sub-matrix of Q

$$\begin{bmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{bmatrix}$$

- Note that

$$\nabla f(\boldsymbol{\alpha}) = Q\boldsymbol{\alpha} - \boldsymbol{e}, \qquad \nabla_B f(\boldsymbol{\alpha}) = Q_{B,:}\boldsymbol{\alpha} - \boldsymbol{e}_B$$

Avoid Memory Problems (Cont'd)

- Only B columns of Q are needed
- In general $|B| \leq 10$ is used. We need $|B| \geq 2$ because of the linear constraint

$$\mathbf{y}_B^T \boldsymbol{d}_B = 0$$

- Calculated when used: trade time for space
- But is such an approach practical?

Algorithm of Decomposition Methods

While $\boldsymbol{\alpha}$ is not optimal

- Select a working set B
- Solve the sub-problem of $\boldsymbol{d}_B$
- $\boldsymbol{\alpha}_B \leftarrow \boldsymbol{\alpha}_B + \boldsymbol{d}_B$

We will talk about the selection of B later

How Decomposition Methods Perform?

- Convergence not very fast. This is known because of using only first-order information
- But, no need to have very accurate α

  decision function:

$$\text{sgn}(\boldsymbol{w}^T\phi(\boldsymbol{x}) + b) = \text{sgn}\left(\sum_{i=1}^{l} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b\right)$$

  Prediction may still be correct with a rough α
- Further, in some situations,

$$\#\text{ support vectors} \ll \#\text{ training points}$$

  Initial $\alpha^1 = 0$, some instances never used

How Decomposition Methods Perform? (Cont'd)

- An example of training 50,000 instances using the software LIBSVM

```
$svm-train -c 16 -g 4 -m 400 22features
Total nSV = 3370
Time 79.524s
```

- This was done on a typical desktop
- Calculating the whole Q takes more time
- #SVs $= 3{,}370 \ll 50{,}000$

  A good case where some remain at zero all the time

How Decomposition Methods Perform? (Cont'd)

- Because many $\alpha_i = 0$ in the end, we can develop a shrinking techniques

  Variables are removed during the optimization procedure. Smaller problems are solved
- For example, if $\alpha_i = 0$ in more than 100 iterations, probably we can remove it
- More advanced techniques are possible
- Of course in the end we must check all variables again for the stopping condition

Machine Learning Properties are Useful in Designing Optimization Algorithms

We have seen that special properties of SVM contribute to the viability of decomposition methods

- For machine learning applications, no need to accurately solve the optimization problem
- Because some optimal $\alpha_i = 0$, decomposition methods may not need to update all the variables
- Also, we can use shrinking techniques to reduce the problem size during decomposition methods

Issues Related to Optimization

- Working set selection
- Asymptotic convergence
- Finite termination & stopping conditions
- Convergence rate
- Numerical issues

Working Set Selection

- In general we don't choose a large $|B|$ because of the following reasons
  1. The sub-problem becomes expensive: $O(|B|^3)$
  2. # iterations may not be significantly decreased
- Currently a popular setting is to choose $|B| = 2$

$$B = \{i, j\}$$

- This is called SMO (Sequential Minimal Optimization)

Working Set Selection (Cont'd)

- One idea is to use gradient information
- If

$$\alpha_i < C, y_i = 1, \text{ and } -\nabla_i f(\boldsymbol{\alpha}) > 0,$$

  then we can enlarge α_i
- Therefore, one possibility is

$$i \in \arg\max\{-y_t \nabla f(\boldsymbol{\alpha})_t \mid \alpha_t < C, y_t = 1 \text{ or } \alpha_t > 0, y_t = -1\}$$

$$j \in \arg\min\{-y_t \nabla f(\boldsymbol{\alpha})_t \mid \alpha_t < C, y_t = -1 \text{ or } \alpha_t > 0, y_t = 1\}$$

- They somewhat correspond to the maximal violation of the optimality condition

Working Set Selection (Cont'd)

- This setting was first used in Joachims (1998)
- Can we use a more sophisticated one?
- It's difficult because cost is a concern
- For example, if we check the second-order information (e.g., objective-value reduction) of all

$$\{i, j\} \text{ pairs},$$

then

$$O(l^2)$$

cost is needed

Working Set Selection (Cont'd)

- Currently in LIBSVM, we use

$$i \in \arg\max\{-y_t \nabla f(\boldsymbol{\alpha})_t \mid \alpha_t < C, y_t = 1 \text{ or } \alpha_t > 0, y_t = -1\}$$

  and select j by second-order information (Fan et al., 2005)
- The cost is still

$$O(l)$$

Complexity of Decomposition Methods

Let's describe the algorithm again

While $\boldsymbol{\alpha}$ is not optimal

- Select a working set B
- Solve the sub-problem of $\boldsymbol{d}_B$
- $\boldsymbol{\alpha}_B \leftarrow \boldsymbol{\alpha}_B + \boldsymbol{d}_B$

Complexity of Decomposition Methods (Cont'd)

- To construct the sub-problem,

$$\nabla_B f(\boldsymbol{\alpha}) = Q_{B,:}\boldsymbol{\alpha} - \boldsymbol{e}_B$$

needs

$$O(|B| \times l \times n)$$

if each kernel evaluation costs $O(n)$

- But for the working set selection, we need the whole $\nabla f(\boldsymbol{\alpha})$
- The cost can be as large as

$$O(l \times l \times n)$$

Complexity of Decomposition Methods (Cont'd)

- Fortunately, we can use

$$\nabla f(\boldsymbol{\alpha} + \begin{bmatrix} \boldsymbol{d}_B \\ \mathbf{0} \end{bmatrix}) = Q\boldsymbol{\alpha} - \boldsymbol{e} + Q \begin{bmatrix} \boldsymbol{d}_B \\ \mathbf{0} \end{bmatrix}$$
$$= \nabla f(\boldsymbol{\alpha}) + Q_{:,B}\boldsymbol{d}_B$$

- Note that $Q_{:,B}$ is available earlier
- Therefore, the cost of decomposition methods for kernel SVM is

$$O(|B| \times l \times n) \times \#\text{ iterations}$$

Differences between Optimization and Machine Learning

- The two topics may have different focuses. We give the following example
- The decomposition method we just discussed converges more slowly when C is large
- Using $C = 1$ on a data set

  # iterations: 508
- Using $C = 5,000$

  # iterations: 35,241

- Optimization researchers may rush to solve difficult cases of large C

  That's what I did before
- It turns out that large C is less used than small C
- Recall that SVM solves

$$\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C(\text{sum of training losses})$$

- A large C means to overfit training data
- This does not give good test accuracy

Outline

Linear and Kernel Classification

- We have

  Kernel $\Rightarrow$ map data to a higher space

  Linear $\Rightarrow$ use the original data; $\phi(\boldsymbol{x}) = \boldsymbol{x}$

- Intuitively, kernel should give superior accuracy than linear
- There are even theoretical proofs. Roughly speaking, from the Taylor expansion of the Gaussian (RBF) kernel

$$e^{-\gamma\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}$$

  linear SVM is a special case of RBF-kernel SVM (Keerthi and Lin, 2003)

Linear and Kernel Classification

- Optimization people may think there is no need to specially consider linear SVM
- That is, same optimization algorithms are enough for both linear and kernel cases
- However, this is wrong if we consider their practical use

Linear and Kernel Classification (Cont'd)

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

- Kernel methods: data mapped to a higher dimensional space

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

  $\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$ easily calculated; little control on $\phi(\cdot)$
- Feature engineering + linear classification :

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

Linear and Kernel Classification (Cont'd)

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

  But training and testing are much faster
- This particularly happens for document classification

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

  Data very sparse (i.e., few non-zeros)

Comparison Between Linear and Kernel (Training Time & Test Accuracy)

			Linear		RBF Kernel	
Data set	#data	#features	Time	Acc.	Time	Acc.
MNIST38	11,982	752	0.1	96.82	38.1	99.70
ijcnn1	49,990	22	1.6	91.81	26.8	98.69
covtype_multiclass	464,810	54	1.4	76.37	46,695.8	96.11
news20	15,997	1,355,191	1.1	96.95	383.2	96.90
real-sim	57,848	20,958	0.3	97.44	938.3	97.82
yahoo-japan	140,963	832,026	3.1	92.63	20,955.2	93.31
webspam	280,000	254	25.7	93.35	15,681.8	99.26

Comparison Between Linear and Kernel (Training Time & Test Accuracy)

			Linear		RBF Kernel	
Data set	#data	#features	Time	Acc.	Time	Acc.
MNIST38	11,982	752	0.1	96.82	38.1	99.70
ijcnn1	49,990	22	1.6	91.81	26.8	98.69
covtype_multiclass	464,810	54	1.4	76.37	46,695.8	96.11
news20	15,997	1,355,191	1.1	96.95	383.2	96.90
real-sim	57,848	20,958	0.3	97.44	938.3	97.82
yahoo-japan	140,963	832,026	3.1	92.63	20,955.2	93.31
webspam	280,000	254	25.7	93.35	15,681.8	99.26

Comparison Between Linear and Kernel (Training Time & Test Accuracy)

Data set	#data	#features	Linear Time	Linear Acc.	RBF Kernel Time	RBF Kernel Acc.
MNIST38	11,982	752	0.1	96.82	38.1	99.70
ijcnn1	49,990	22	1.6	91.81	26.8	98.69
covtype_multiclass	464,810	54	1.4	76.37	46,695.8	96.11
news20	15,997	1,355,191	1.1	96.95	383.2	96.90
real-sim	57,848	20,958	0.3	97.44	938.3	97.82
yahoo-japan	140,963	832,026	3.1	92.63	20,955.2	93.31
webspam	280,000	254	25.7	93.35	15,681.8	99.26

Therefore, there is a need to develop optimization methods for large linear classification

Why Linear is Faster in Training and Prediction?

- The reason is that

  for linear, $\boldsymbol{x}_i$ is available

  but

  for kernel, $\phi(\boldsymbol{x}_i)$ is not

- To illustrate this point, let's modify kernel decomposition methods discussed earlier for linear

Outline

Decomposition Method for Linear Classification

- Recall for kernels, we solve the sub-problem:

$$\min_{\boldsymbol{d}_B} \quad \frac{1}{2}\begin{bmatrix}(\boldsymbol{\alpha}_B+\boldsymbol{d}_B)^T & (\boldsymbol{\alpha}_N)^T\end{bmatrix}\begin{bmatrix}Q_{BB} & Q_{BN}\\ Q_{NB} & Q_{NN}\end{bmatrix}\begin{bmatrix}\boldsymbol{\alpha}_B+\boldsymbol{d}_B\\ \boldsymbol{\alpha}_N\end{bmatrix}$$
$$-\begin{bmatrix}\boldsymbol{e}_B^T & \boldsymbol{e}_N^T\end{bmatrix}\begin{bmatrix}\boldsymbol{\alpha}_B+\boldsymbol{d}_B\\ \boldsymbol{\alpha}_N\end{bmatrix}$$

subject to $0 \le \alpha_t \le C, t \in B,\ \mathbf{y}_B^T(\boldsymbol{\alpha}_B+\boldsymbol{d}_B) = -\mathbf{y}_N^T\boldsymbol{\alpha}_N$

- The objective function over $\boldsymbol{d}_B$ is

$$\frac{1}{2}\boldsymbol{d}_B^T Q_{BB}\boldsymbol{d}_B + (-\boldsymbol{e}_B + Q_{B,:}\boldsymbol{\alpha})^T\boldsymbol{d}_B$$

Decomposition Method for Linear Classification (Cont'd)

- We need

$$Q_{B,:}\boldsymbol{\alpha} = \begin{bmatrix} Q_{BB} & Q_{BN} \end{bmatrix} \boldsymbol{\alpha}$$

- The cost is

$$O(|B| \times l \times n)$$

because for $i \in B$,

$$Q_{i,:}\boldsymbol{\alpha} = \sum_{j=1}^{l} y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)\alpha_j$$

- n: # features, l: # data

Decomposition Method for Linear Classification (Cont'd)

- In the linear case,

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

$$\Rightarrow \quad \sum_{j=1}^{l} y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \alpha_j = y_i \boldsymbol{x}_i^T \left(\sum_{j=1}^{l} y_j \boldsymbol{x}_j \alpha_j \right)$$

- Assume

$$\mathbf{u} \equiv \sum_{j=1}^{l} y_j \alpha_j \boldsymbol{x}_j \tag{4}$$

is available

Decomposition Method for Linear Classification (Cont'd)

- The cost is significantly reduced

$$O(|B| \times l \times n) \quad \Rightarrow \quad O(|B| \times n)$$

- The main difference is that in kernel

$$\sum_{j=1}^{l} y_j \alpha_j \phi(\boldsymbol{x}_j)$$

cannot be written down. But for linear we can!

Decomposition Method for Linear Classification (Cont'd)

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

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

Then the following update rule can be applied

$$\mathbf{u} \leftarrow \mathbf{u} + \sum_{i:i \in B} (d_i) y_i \boldsymbol{x}_i.$$

- The cost is also

$$O(|B| \times n)$$

Decomposition Method for Linear Classification (Cont'd)

- Note that eventually

$$\mathbf{u} \rightarrow \text{ primal optimal } \boldsymbol{w}$$

One-variable Procedure

- This is what people use now in practice
- Let's consider the optimization problem without the bias term

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

- The dual problem now doesn't have a linear constraint

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \le \alpha_i \le C, i = 1, \ldots, l, \end{aligned}$$

One-variable Procedure (Cont'd)

- Here

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

and

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

- Without the linear constraint

$$\mathbf{y}^T \boldsymbol{\alpha} = 0$$

we can choose one variable at a time

One-variable Procedure (Cont'd)

- Given current $\boldsymbol{\alpha}$. Let the working set be $B = \{i\}$
- 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

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

One-variable Procedure (Cont'd)

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

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

- Note that

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

One-variable Procedure (Cont'd)

- As before, define

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

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

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

Algorithm: Dual Coordinate Descent

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

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

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

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

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

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

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

  (d) If α_i needs to be changed

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

One-variable Procedure (Cont'd)

- Maintaining $\mathbf{u}$ also costs

$$O(n)$$

- Thus the total cost is

$$O(n) \times \#\text{ iterations}$$

- Recall that the cost for kernel is

$$O(l \times n) \times \#\text{ iterations}$$

  if we don't have b and select $|B| = 1$

- We will explain some interesting differences between the two

Recap: Dual Coordinate Descent

- It's 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
- So what's new?

Recap: Dual 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 is another example where we take the problem structure into account

Recap: Dual Coordinate Descent (Cont'd)

- Such a setting was first developed at Hsieh et al. (2008)

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)
- Cyclic order to update elements

$$\alpha_1 \rightarrow \alpha_2 \rightarrow \cdots \rightarrow \alpha_l$$

A random order gives faster convergence

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

We will go back to this issue later

Difference from the Kernel Case

- We have seen that coordinate-descent type of methods are used for both linear and kernel classifiers
- Recall the i-th element of gradient costs $O(n)$ by

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

but we cannot do this for kernel because

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

cannot be separated

Difference from the Kernel Case (Cont'd)

- If using kernel, the cost of calculating $\nabla_i f(\boldsymbol{\alpha})$ must be $O(ln)$
- However, if $O(ln)$ cost is spent, the whole $\nabla f(\boldsymbol{\alpha})$ can be maintained

$$\nabla f(\boldsymbol{\alpha}) \leftarrow \nabla f(\boldsymbol{\alpha}) + Q_{:,i} d_i$$

- In contrast, the linear setting of using $\mathbf{u}$ knows only $\nabla_i f(\boldsymbol{\alpha})$ rather than the whole $\nabla f(\boldsymbol{\alpha})$
- This situation affects the working set selection

Difference from the Kernel Case (Cont'd)

- We have mentioned that for existing decomposition methods (or say coordinate descent methods) for kernel classifiers, $\nabla f(\boldsymbol{\alpha})$ information is used to select variables for update
- Therefore, we have the following situations for two working set selections
  1. Greedy: Using $\nabla f(\boldsymbol{\alpha})$
  2. Cyclic

	Kernel		Linear	
	Greedy	Cyclic	Greedy	Cyclic
Update α_i	$O(1)$	$O(ln)$	$O(1)$	$O(n)$
Maintain $\nabla f(\boldsymbol{\alpha})$	$O(ln)$	NA	$O(ln)$	NA

Difference from the Kernel Case (Cont'd)

- In optimization there are two types of coordinate descent methods
  1. Gauss-Seidel: sequential selection of variables
  2. Gauss-Southwell: greedy selection of variables
- To do greedy selection, usually the whole gradient must be available
- Existing coordinate descent methods for linear $\Rightarrow$ related to Gauss-Seidel

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

Difference from the Kernel Case (Cont'd)

- In general greedy leads to fewer iterations than cyclic
- So is the cyclic setting for linear practically viable?

Working Set Selection

- Without gradient information, looks like we can only do a cyclic update

$$1, 2, 3, \ldots$$

- However, Hsieh et al. (2008, Section 3.1) showed that with random permutation, the convergence is much faster
- Let's try an example by using LIBLINEAR
- real-sim is a data set with

72,309 instances and 20,958 features

Working Set Selection (Cont'd)

- With random permutation

```
$ ./train ~/Desktop/real-sim
optimization finished, #iter = 13
Objective value = -4396.370629
```

  Here an iteration means to go through all instances once (though shrinking is applied)

- Without random permutation (cyclic)

```
$ ./train ~/Desktop/real-sim
optimization finished, #iter = 326
Objective value = -4391.706239
```

Working Set Selection (Cont'd)

- Here l1 loss is used
- Both approaches are under the same stopping condition
- Let's see the algorithm with random permutation

Working Set Selection (Cont'd)

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

  Randomly permute $1, \ldots, l$

  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

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

- Note that it's not useful if we only randomly permute data once

Working Set Selection (Cont'd)

- Randomly permute $1, \ldots, l$

  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

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

- On the other hand, random CD works:

Working Set Selection (Cont'd)

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

  Randomly select i

  (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

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

- Turns out it is easier to analyze the complexity of the random CD; see Shalev-Shwartz and Zhang (2013) and many subsequent works

Working Set Selection (Cont'd)

- It's difficult to analyze the setting of using random permutations. Some recent attempts (for simplified situations) include Lee and Wright (2016)
- This is still an ongoing research issue
- Another line of research is to cheaply find some important indices.
- See Glasmachers and Dogan (2013) and other developments
- In other words, we think some distributions are better than uniform

Working Set Selection (Cont'd)

- However, it is difficult to beat the uniform setting if shrinking has been applied

Working Set Selection and Bias Term

- Recall if we use

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

as the decision function, then the dual problem has a linear constraint

$$\mathbf{y}^T\boldsymbol{\alpha} = 0$$

- Then at each iteration, two indices

$$\{i, j\}$$

must be selected

Working Set Selection and Bias Term (Cont'd)

- If we use a cyclic setting, this means

$$(1,2),(1,3),(1,4),\ldots,(2,3),\ldots$$

- The training may be terribly slow because for many pairs we cannot change α at all
- Therefore, an interesting situation is as follows

Working set selection	Without bias	With bias	Used for
cyclic	OK	not OK	linear
greedy (with $\nabla f(\alpha)$)	OK	OK	kernel

Working Set Selection and Bias Term (Cont'd)

- Fortunately, linear classification is often used to handle large data with many sparse features
- In such a high dimensional space, bias term is often not needed
- Even if it is, there is a trick of adding the bias term to the objective function

$$\min_{\boldsymbol{w},b} \ \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}+\frac{1}{2}b^2+C\sum_{i=1}^{l}\max(1-y_i(\begin{bmatrix}\boldsymbol{w}^T & b\end{bmatrix}\begin{bmatrix}\boldsymbol{x}\\ 1\end{bmatrix}),0)$$

Working Set Selection and Bias Term (Cont'd)

- The dual no longer has the linear constraint

$$\mathbf{y}^T \boldsymbol{\alpha} = 0$$

- However, for some problems (e.g., one-class SVM), a linear constraint must be there (details not shown)
- Then the training by coordinate descent for the linear case can be an issue

Other Losses

- Our discussion so far is for l1 loss
- All results can be applied to other losses such as l2 loss, logistic loss, etc.

Outline

More Optimization Methods can be Applied for Linear

- Recall that

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

- Kernel: can only solve an optimization problem of $\boldsymbol{\alpha}$ because $\boldsymbol{w}$ is too high dimensional
- Linear: can solve either $\boldsymbol{w}$ or $\boldsymbol{\alpha}$
- We will show an example to minimize over $\boldsymbol{w}$

Newton Method

- Let's minimize a twice-differentiable function

$$\min_{\boldsymbol{w}} f(\boldsymbol{w})$$

- For example, logistic regression has

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\log\left(1+e^{-y_i\boldsymbol{w}^T\boldsymbol{x}_i}\right).$$

- Newton direction at iterate $\boldsymbol{w}^k$

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

Truncated Newton Method

- The above sub-problem is equivalent to solving Newton linear system

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

- Approximately solving the linear system $\Rightarrow$ truncated Newton
- However, Hessian matrix $\nabla^2 f(\boldsymbol{w}^k)$ is too large to be stored

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

- For document data, n can be millions or more

Using Properties of Data Classification

- But Hessian has a special form

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

- D is diagonal. For logistic regression,

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

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

$$X = \begin{bmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_l^T \end{bmatrix} \in R^{l \times n}$$

Using Properties of Data Classification (Cont'd)

- Using Conjugate Gradient (CG) to solve the linear system.
- CG is an iterative procedure. Each CG step mainly needs one Hessian-vector product

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

- Therefore, we have a Hessian-free approach

Using Properties of Data Classification (Cont'd)

- Now the procedure has two layers of iterations

  Outer: Newton iterations

  Inner: CG iterations per Newton iteration
- Past machine learning works used Hessian-free approaches include, for example, (Keerthi and DeCoste, 2005; Lin et al., 2008)
- Second-order information used: faster convergence than first-order methods

Training L2-loss SVM

- The loss function is differentiable but not twice differentiable

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

- We can use generalized Hessian (Mangasarian, 2002); details not shown here

Preconditioning

- Each Hessian-vector product

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

costs

$$O(ln), \quad \text{where } X \in R^{l \times n}$$

- Each function/gradient evaluation also costs $O(ln)$; details omitted
- Therefore, the cost of each Newton iteration is roughly

$$O(ln) \times \#\text{ CG steps}$$

Preconditioning (Cont'd)

- Can we reduce the number of CG steps by preconditioning?
- This classic technique finds

$$EE^T \approx \nabla^2 f(\boldsymbol{w})$$

so the linear system

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

becomes

$$E^{-1}\nabla^2 f(\boldsymbol{w})E^{-T}\hat{\boldsymbol{s}} = -E^{-1}\nabla f(\boldsymbol{w})$$

Preconditioning (Cont'd)

- Note that

$$\boldsymbol{s} = E^{-T}\hat{\boldsymbol{s}}$$

- If

$$E^{-1}\nabla^2 f(\boldsymbol{w}^k)E^{-T} \approx \mathcal{I}$$

  then # CG steps can be reduced

- For example, we use the diagonal preconditioner so

$$EE^T = \text{diag}(\nabla^2 f(\boldsymbol{w}))$$

  and

$$E = \sqrt{\text{diag}(\nabla^2 f(\boldsymbol{w}))}$$

Preconditioning (Cont'd)

- Because

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

we have

$$\nabla^2_{j,j} f(\boldsymbol{w}) = 1 + C \sum_{i=1}^{l} D_{ii} X_{ij}^2$$

- The cost of constructing the preconditioner is

$$O(ln)$$

Preconditioning (Cont'd)

- In each CG step, doing

$$(E^{-1} \text{ or } E^{-T}) \times \text{ a vector}$$

costs

$$O(n)$$

- Therefore, extra cost of diagonal preconditioning at each Newton iteration is

$$O(n) \times \#\text{ CG steps} + O(ln)$$

- This is acceptable in compared with what we already need

$$O(ln) \times \#\text{ CG steps}$$

Preconditioning (Cont'd)

- However, the issue is that the number of CG steps may not be decreased
- In the area of numerical analysis, preconditioning is an art rather than a science
- Further, because of using a Hessian-free approach, many existing preconditioners are not directly applicable
- This is still a research issue

Sub-sampled Newton Methods

- The optimization problem can be rewritten as

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

- The second term is indeed the average loss
- A subset of data should give similar gradient or Hessian!!
- This kind of settings has been explored in Byrd et al. (2011); Martens (2010)
- Clearly this is an example of taking properties of machine learning problems

Lesson Learned from Kernel to Linear

- We must know the practical machine-learning use in order to decide if new optimization algorithms are needed for certain problems

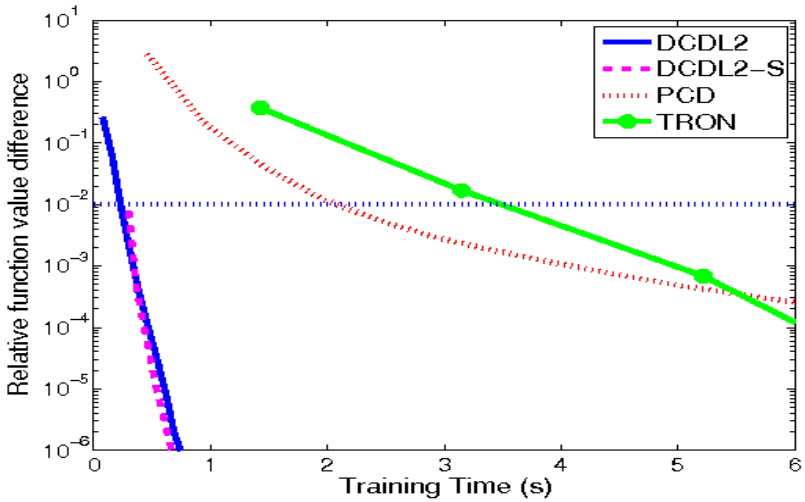
Outline

Comparisons

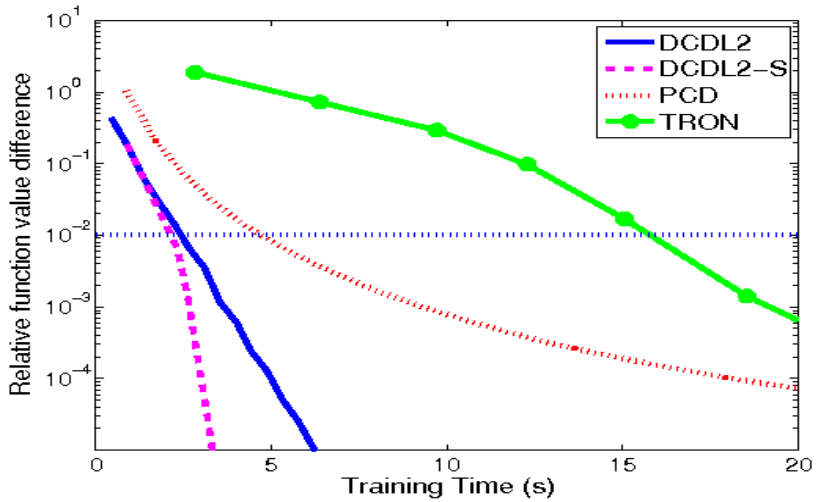
L2-loss SVM is used

- DCDL2: Dual coordinate descent (Hsieh et al., 2008)
- DCDL2-S: DCDL2 with shrinking (Hsieh et al., 2008)
- PCD: Primal coordinate descent (Chang et al., 2008)
- TRON: Trust region Newton method (Lin et al., 2008)

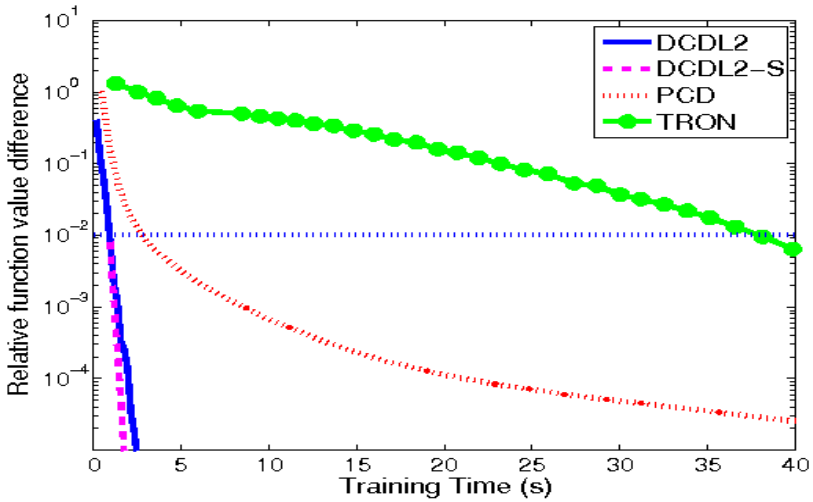
Objective values (Time in Seconds)



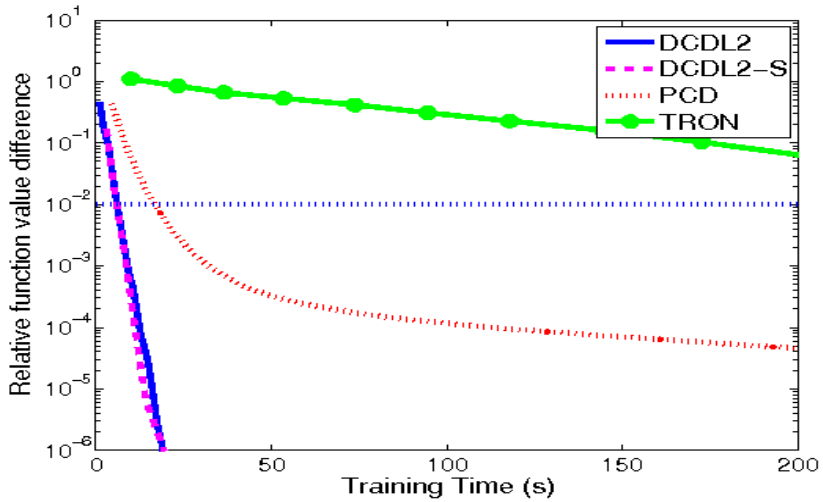
news20



rcv1



yahoo-japan



yahoo-korea

Analysis

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

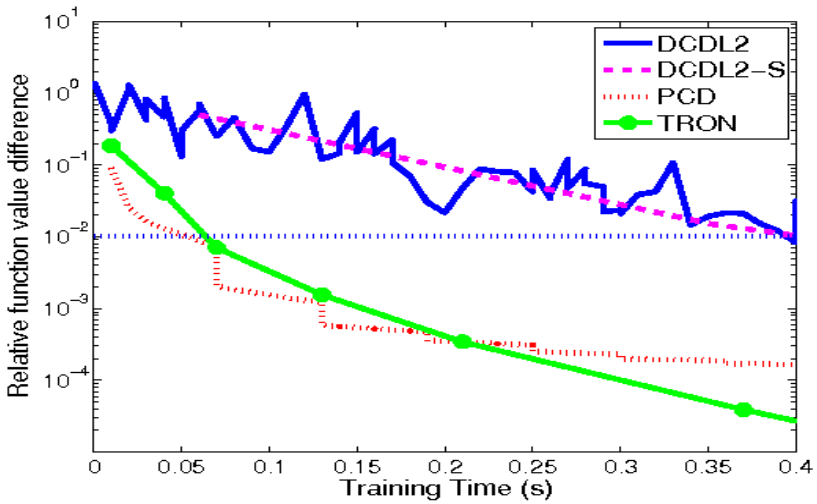
  Useful for document classification
- Half million data in a few seconds
- However, it is less effective if

  # features small: should solve primal; or

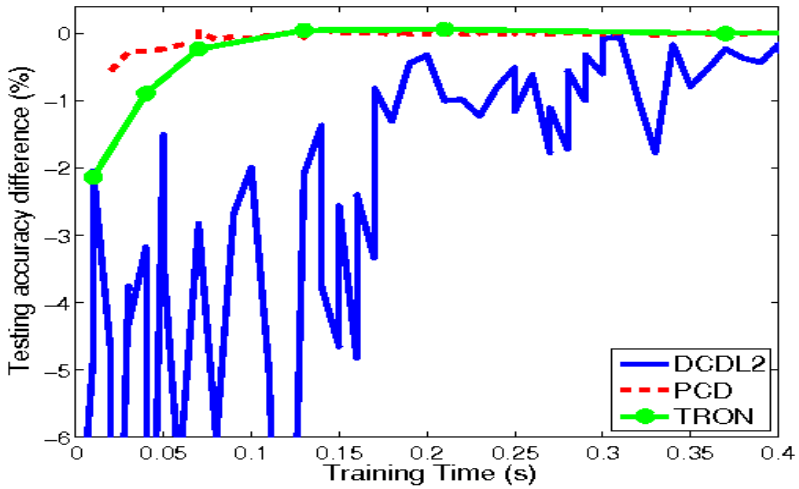
  large penalty parameter C; problems are more ill-conditioned

An Example When # Features Small

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



Objective value



Accuracy

Careful Evaluation

- This is very very important
- Let me give you one real personal experience

Careful Evaluation (Cont'd)

- In Lin et al. (2008), to check if diagonal perconditioner works in Newton methods we give the following table of total number of CG steps

Problem	CG	PCG
a9a	567	263
real-sim	104	160
news20	71	155
citeseer	113	115
yahoo-japan	278	326
rcv1	225	280
yahoo-korea	779	736

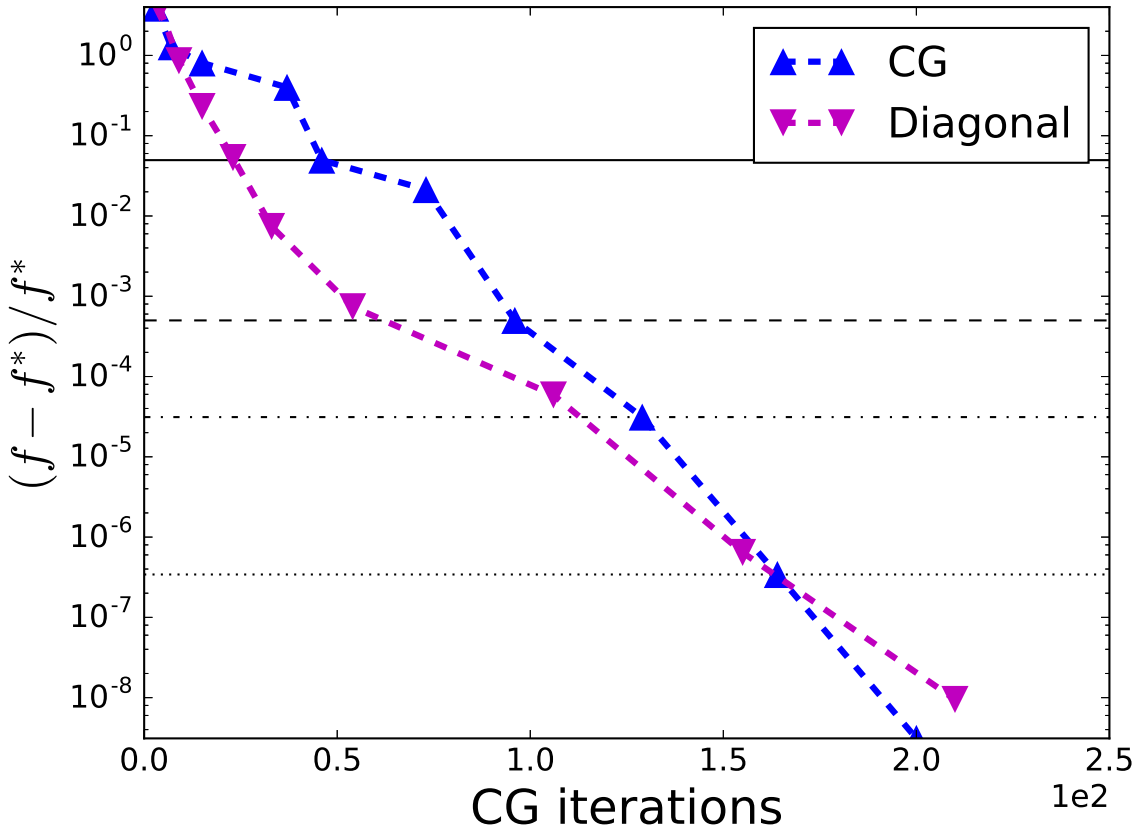
Careful Evaluation (Cont'd)

- Clearly, PCG may not be better
- But this conclusion is misleading
- In this experiment,

  a strict stopping condition is used

Careful Evaluation (Cont'd)

- Let's look at this figure for yahoo-japan



- You see four horizontal lines

Careful Evaluation (Cont'd)

- The second line corresponds to the default LIBLINEAR stopping condition
- The 4 lines are by using

$$10\epsilon, \quad \epsilon, \quad 0.1\epsilon, \quad 0.01\epsilon,$$

  where ϵ is the default LIBLINEAR stopping tolerance
- Things below/above these 4 lines are not useful
- They are too loose or too strict for the machine learning task

Careful Evaluation (Cont'd)

- Another issue is which regularization parameter C to be used in presenting results?
- In the table, we use $C = 1$, the simplest choice
- In the figure, we use

$$C_{\text{Best}} = 0.5,$$

where C_{Best} is the value leading to the best validation accuracy.

- A reasonable setting is to show figures of

$$C = C_{\text{Best}} \times \{0.01, 0.1, 1, 10, 100\},$$

Careful Evaluation (Cont'd)

- The reason is that in practice we start from a small C to search for C_{Best}.
- Things are larger than $100 \times C_{\text{Best}}$ are not important
- Lesson: even in comparing optimization algorithms for machine learning, we need to take machine learning properties into account

Outline

Multi-core Implementation

- Nowadays each computer has several cores
- They can do parallel computation
- However, most machine learning algorithms (including those we have discussed) do not take parallel computation into account

Multi-core Implementation (Cont'd)

- In fact, algorithms may need to be redesigned
- Recently we did two studies
  1. Newton method for solving the primal problem (Lee et al., 2015)
  2. Coordinate descent method for solving the dual problem (Chiang et al., 2016)
- We will discuss the Newton method in detail

Parallel Newton Implementation

- Recall the bottleneck is the Hessian-vector product

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

- This is conducted at each CG step

Matrix-vector Multiplications: More Than 90% of the Training Time

Data set	#instances	#features	ratio
kddb	19,264,097	29,890,095	82.11%
url_combined	2,396,130	3,231,961	94.83%
webspam	350,000	16,609,143	97.95%
rcv1_binary	677,399	47,236	97.88%
covtype_binary	581,012	54	89.20%
epsilon_normalized	400,000	2,000	99.88%
rcv1_multiclass	518,571	47,236	97.04%
covtype_multiclass	581,012	54	89.06%

Matrix-vector Multiplications: More Than 90% of the Training Time (Cont'd)

- This result is by Newton methods using one core
- We should parallelize matrix-vector multiplications
- For $\nabla^2 f(\boldsymbol{w})\boldsymbol{d}$ we must calculate

$$\mathbf{u} = X\boldsymbol{d} \tag{5}$$

$$\mathbf{u} \leftarrow D\mathbf{u} \tag{6}$$

$$\bar{\mathbf{u}} = X^T\mathbf{u}, \text{ where } \mathbf{u} = DX\boldsymbol{d} \tag{7}$$

- Because D is diagonal, (6) is easy

Matrix-vector Multiplications: More Than 90% of the Training Time (Cont'd)

- We will discuss the parallelization of (5) and (7)
- They are more complicated
- For our problems, X is large and sparse
- Efficient parallel sparse matrix-vector multiplications are still a research issue

Parallel Xd Operation

- Assume that X is in a row-oriented sparse format

$$X = \begin{bmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_l^T \end{bmatrix} \text{ and } \mathbf{u} = X\boldsymbol{d} = \begin{bmatrix} \boldsymbol{x}_1^T \boldsymbol{d} \\ \vdots \\ \boldsymbol{x}_l^T \boldsymbol{d} \end{bmatrix}$$

- we have the following simple loop

1: **for** $i = 1, \ldots, l$ **do**
2: $\quad u_i = \boldsymbol{x}_i^T \boldsymbol{d}$
3: **end for**

- Because the l inner products are independent, we can easily parallelize the loop by, for example, OpenMP

Parallel $X^T\mathbf{u}$ Operation

- For the other matrix-vector multiplication

$$\bar{\mathbf{u}} = X^T\mathbf{u}, \text{ where } \mathbf{u} = DX\boldsymbol{d},$$

  we have

$$\bar{\mathbf{u}} = u_1\boldsymbol{x}_1 + \cdots + u_l\boldsymbol{x}_l.$$

- Because matrix X is row-oriented, accessing columns in X^T is much easier than rows
- We can use the following loop

1: **for** $i = 1, \ldots, l$ **do**
2: $\quad \bar{\mathbf{u}} \leftarrow \bar{\mathbf{u}} + u_i\boldsymbol{x}_i$
3: **end for**

Parallel $X^T\mathbf{u}$ Operation (Cont'd)

- There is no need to store a separate X^T
- However, it is possible that threads on $u_{i_1}\boldsymbol{x}_{i_1}$ and $u_{i_2}\boldsymbol{x}_{i_2}$ want to update the same component $\bar{u}_s$ at the same time:

1: **for** $i = 1, \ldots, l$ **do** in parallel
2: **for** $(\boldsymbol{x}_i)_s \neq 0$ **do**
3: $\bar{u}_s \leftarrow \bar{u}_s + u_i(\boldsymbol{x}_i)_s$
4: **end for**
5: **end for**

Atomic Operations for Parallel $X^T\mathbf{u}$

- An atomic operation can avoid other threads to write $\bar{u}_s$ at the same time.

1: **for** $i = 1, \ldots, l$ **do** in parallel
2: **for** $(\boldsymbol{x}_i)_s \neq 0$ **do**
3: atomic: $\bar{u}_s \leftarrow \bar{u}_s + u_i(\boldsymbol{x}_i)_s$
4: **end for**
5: **end for**

- However, waiting time can be a serious problem

Reduce Operations for Parallel $X^T\mathbf{u}$

- Another method is using temporary dense arrays maintained by each thread, and summing up them in the end
- That is, store

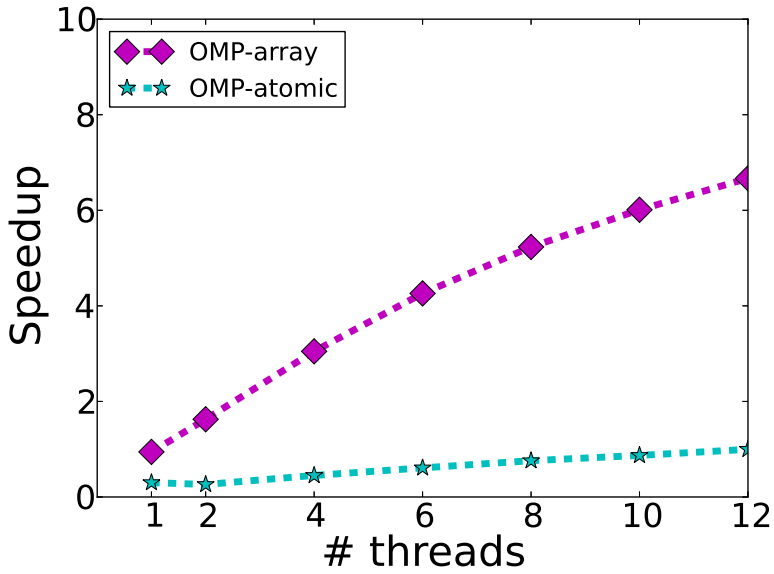
$$\hat{\mathbf{u}}^p = \sum_i \{u_i \boldsymbol{x}_i \mid i \text{ run by thread p}\}$$

and then

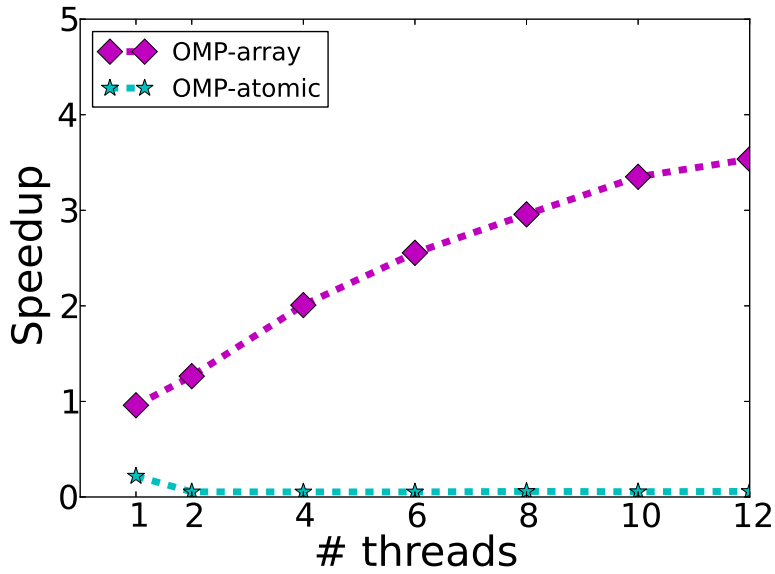
$$\bar{\mathbf{u}} = \sum_p \hat{\mathbf{u}}^p$$

Atomic Operation: Almost No Speedup

- Reduce operations are superior to atomic operations



rcv1_binary



covtype_binary

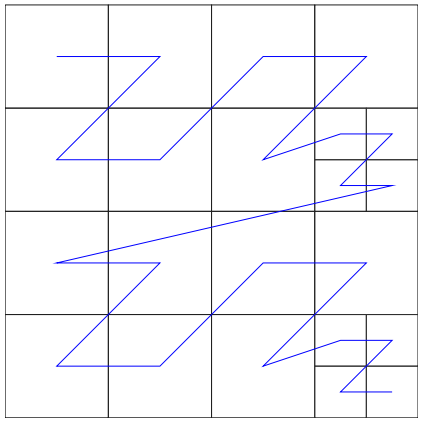
- Subsequently we use the reduce operations

Existing Algorithms for Sparse Matrix-vector Product

- Instead of our direct implementation to parallelize loops, in the next slides we will consider two existing methods

Recursive Sparse Blocks (Martone, 2014)

- RSB (Recursive Sparse Blocks) is an effective format for fast parallel sparse matrix-vector multiplications
- It recursively partitions a matrix to be like the figure



- Locality of memory references improved, but the construction time is not negligible

Recursive Sparse Blocks (Cont'd)

- Parallel, efficient sparse matrix-vector operations
- Improve locality of memory references
- But the initial construction time is about 20 multiplications, which is not negligible in some cases
- We will show the result in the experiments

Intel MKL

- Intel Math Kernel Library (MKL) is a commercial library including optimized routines for linear algebra (Intel)
- It supports fast matrix-vector multiplications for different sparse formats.
- We consider the row-oriented format to store X.

Experiments

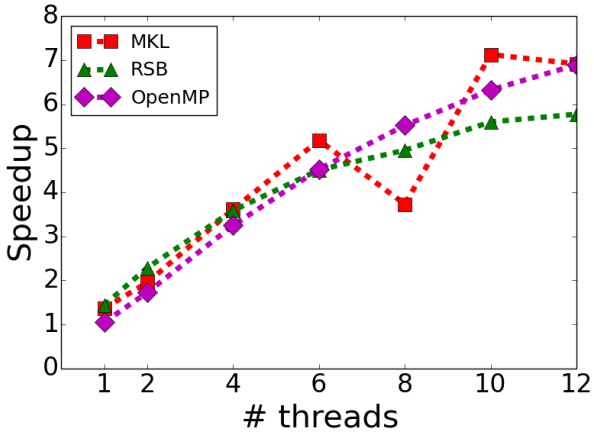
- Baseline: Single core version in LIBLINEAR 1.96. It sequentially run the following operations

$$\mathbf{u} = X\boldsymbol{d}$$
$$\mathbf{u} \leftarrow D\mathbf{u}$$
$$\bar{\mathbf{u}} = X^T\mathbf{u}, \text{ where } \mathbf{u} = DX\boldsymbol{d}$$

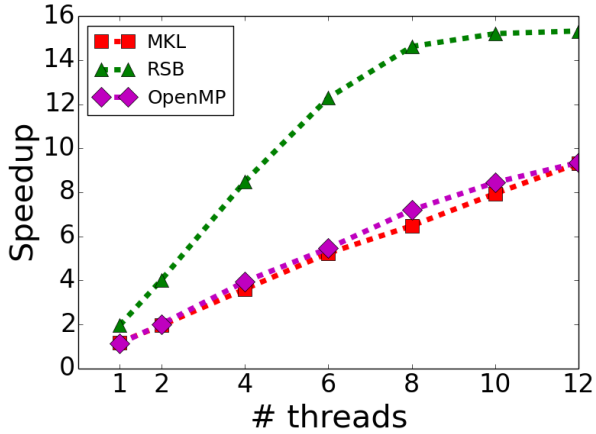
- OpenMP: Use OpenMP to parallelize loops
- MKL: Intel MKL version 11.2
- RSB: librsb version 1.2.0

Speedup of Xd: All are Excellent

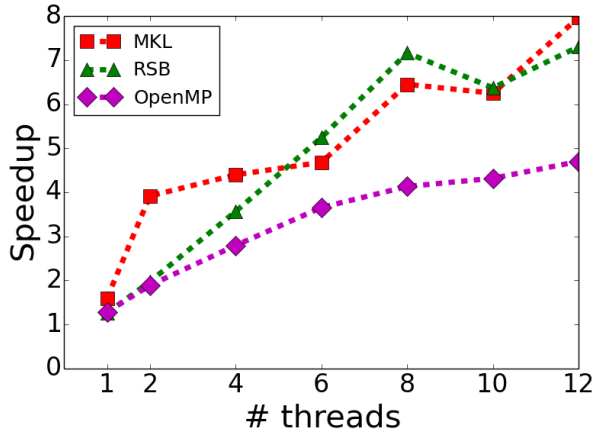
rcv1_binary



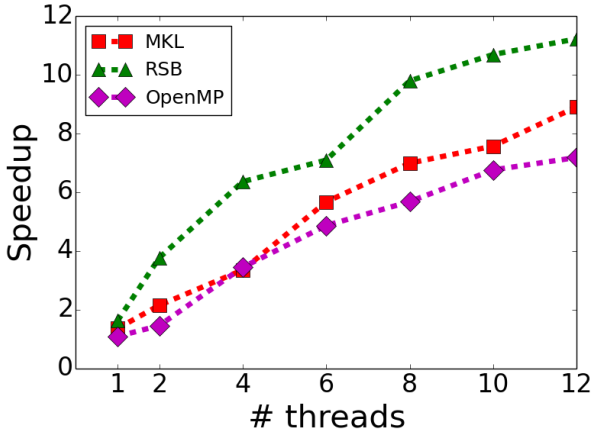
webspam



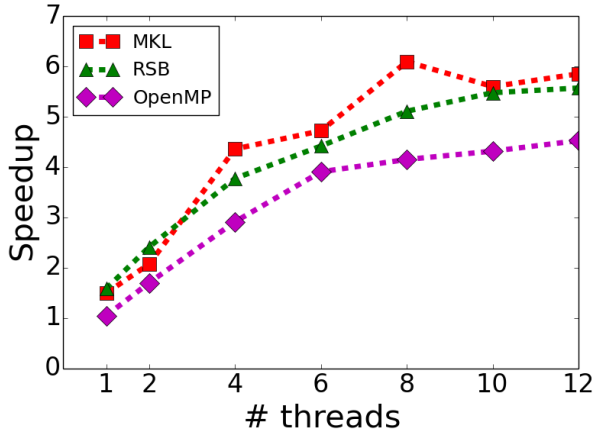
kddb



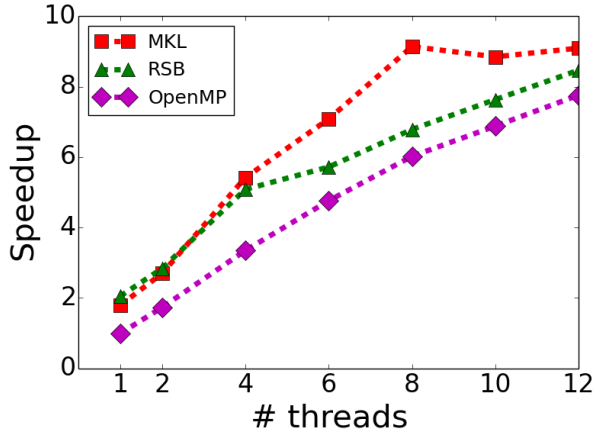
url_combined



covtype_binary

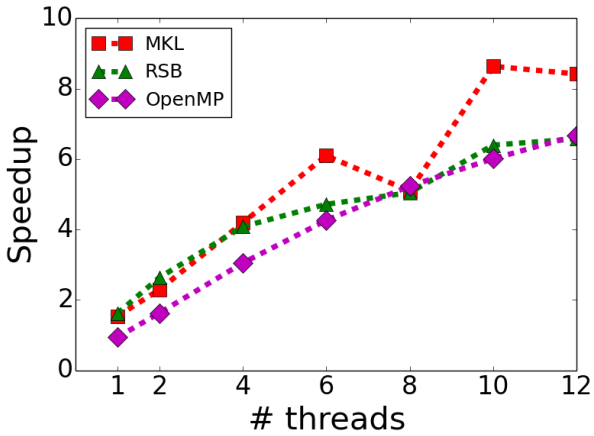


rcv1_multiclass

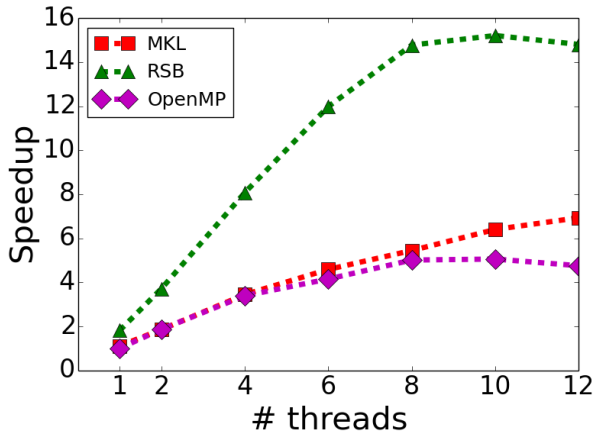


More Difficult to Speed up $X^T\mathbf{u}$

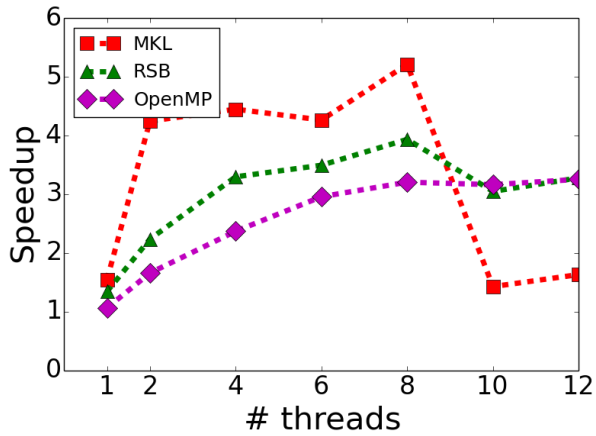
rcv1_binary



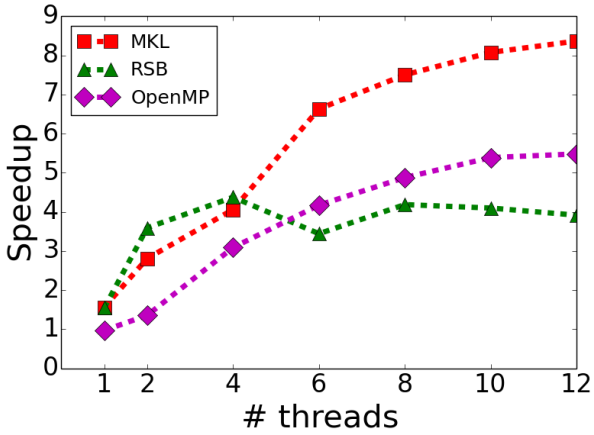
webspam



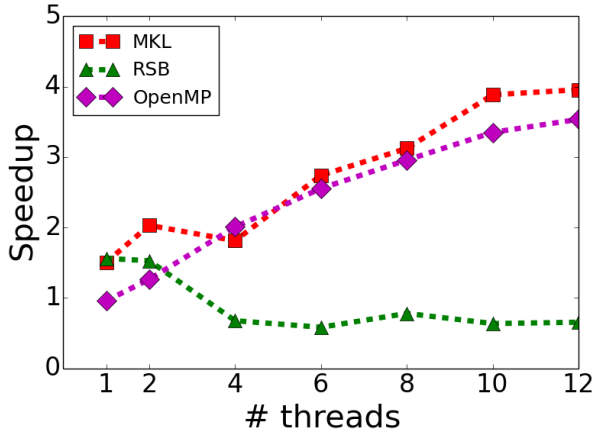
kddb



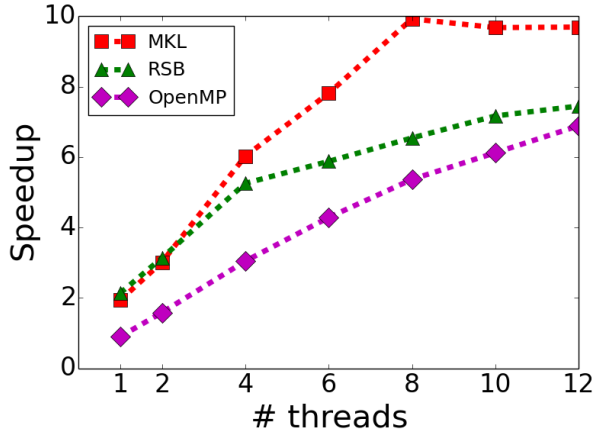
url_combined



covtype_binary



rcv1_multiclass



Indeed it's not easy to have a multi-core implementation that is

1. simple, and
2. reasonably efficient

Let me describe what we do in the end in multi-core LIBLINEAR

Reducing Memory Access to Improve Speedup

- In computing

$$X\boldsymbol{d} \text{ and } X^T(DX\boldsymbol{d})$$

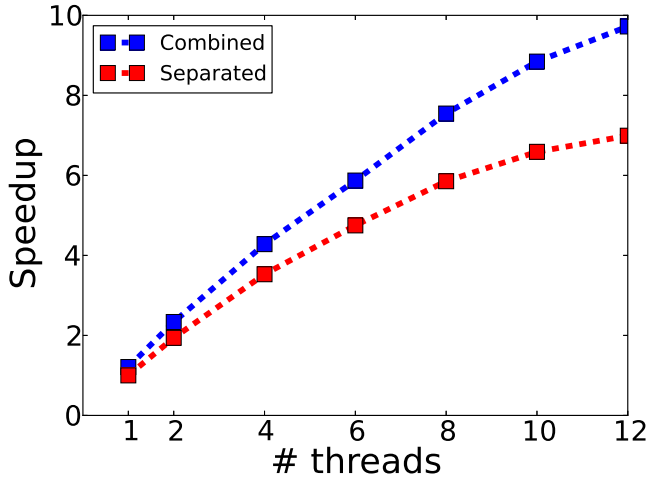
  the data matrix is accessed twice
- We notice that these two operations can be combined together

$$X^T DX\boldsymbol{d} = \sum_{i=1}^{l} \boldsymbol{x}_i D_{ii} \boldsymbol{x}_i^T \boldsymbol{d}$$

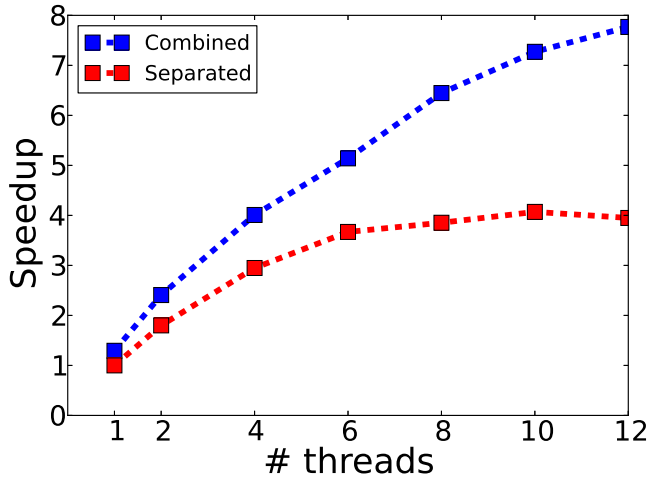
- We can parallelize one single loop by OpenMP

Reducing Memory Access to Improve Speedup (Cont'd)

- Better speedup as memory accesses are reduced



rcv1_binary



covtype_binary

- The number of operations is the same, but memory access dramatically affects the idle time of threads

Reducing Memory Access to Improve Speedup (Cont'd)

- Therefore, if we can efficiently do

$$\sum_{i=1}^{l} \boldsymbol{x}_i D_{ii} \boldsymbol{x}_i^T \boldsymbol{d} \tag{8}$$

  then probably we don't need sophisticated sparse matrix-vector packages
- However, for a simple operation like (8) careful implementations are still needed

OpenMP Scheduling

- An OpenMP loop assigns tasks to different threads.
- The default `schedule(static)` splits indices to P blocks, where each contains I/P elements.
- However, as tasks may be unbalanced, we can have a dynamic scheduling – available threads are assigned to the next tasks.
- For example, `schedule(dynamic,256)` implies that a thread works on 256 elements each time.
- Unfortunately, overheads occur for the dynamic task assignment.

OpenMP Scheduling (Cont'd)

- Deciding suitable scheduling is not trivial.
- Consider implementing $X^T\mathbf{u}$ as an example. This operation involves the following three loops.
  1. Initializing $\hat{\mathbf{u}}^p = \mathbf{0}, \forall p = 1, \ldots, P$.
  2. Calculating $\hat{\mathbf{u}}^p, \forall p$ by

$$\hat{\mathbf{u}}^p = \sum \{\mathbf{u}_i \boldsymbol{x}_i \mid i \text{ run by thread } p\}$$

  3. Calculating $\bar{\mathbf{u}} = \sum_{p=1}^{P} \hat{\mathbf{u}}^p$.

OpenMP Scheduling (Cont'd)

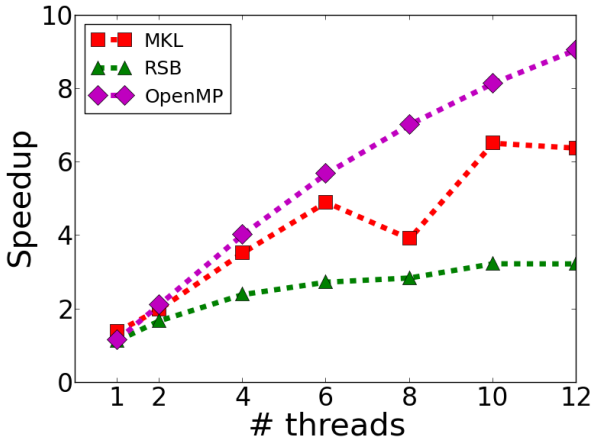
- Consider the second step

	covtype_binary	rcv1_binary
`schedule(static)`	0.2879	2.9387
`schedule(dynamic)`	1.2611	2.6084
`schedule(dynamic, 256)`	0.2558	1.6505

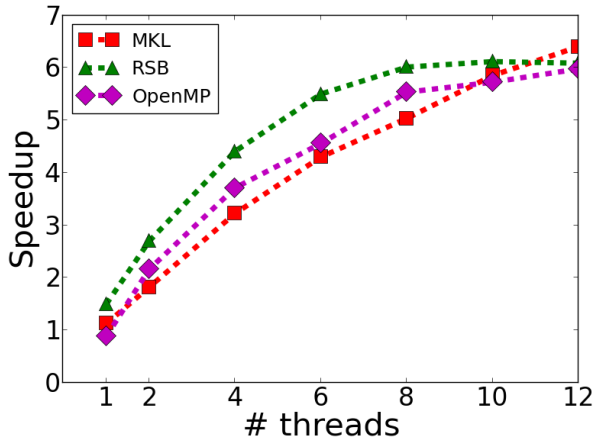
- Clearly, a suitable scheduling is essential
- The other two steps are more balanced, so `schedule(static)` is used (details omitted)

Speedup of Total Training Time

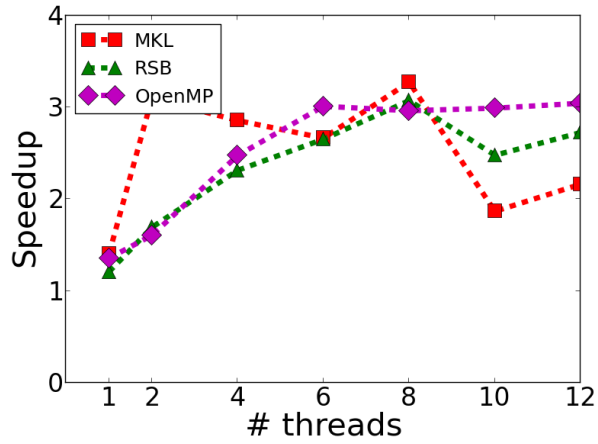
rcv1_binary



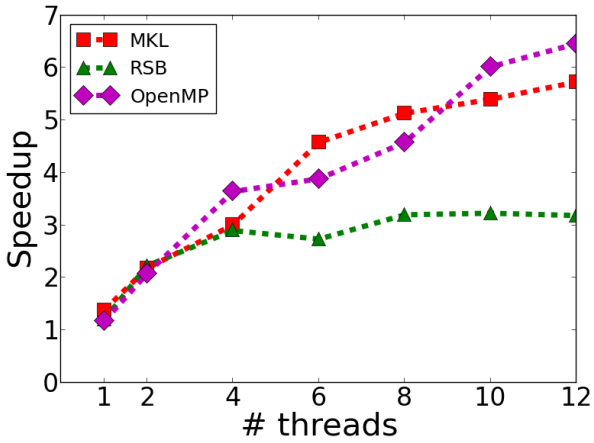
webspam



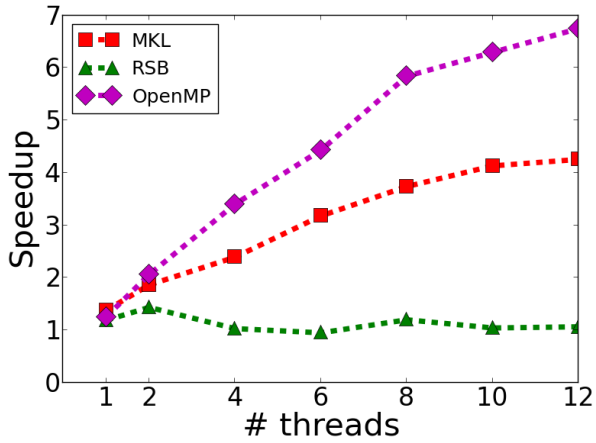
kddb



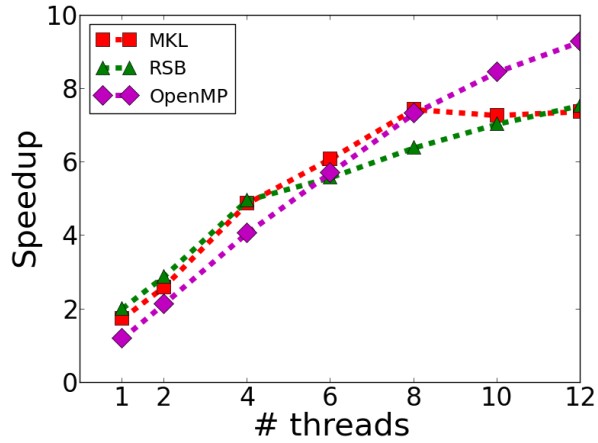
url_combined



covtype_binary



rcv1_multiclass



Analysis of Experimental Results

- For RSB, the speedup for $X\boldsymbol{d}$ is excellent, but is poor for $X^T\mathbf{u}$ on some $n \ll l$ data (e.g. covtype) Furthermore, construction time is expensive
- OpenMP is the best for almost all cases, mainly because of combing $X\boldsymbol{d}$ and $X^T\mathbf{u}$ together
- Therefore, with appropriate settings, simple implementations by OpenMP can achieve excellent speedup

Outline

Conclusions

- Optimization has been very useful for machine learning
- We must incorporate machine learning knowledge in designing suitable optimization algorithms and software
- The interaction between optimization and machine learning is very interesting and exciting.

References I

B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

R. H. Byrd, G. M. Chin, W. Neveitt, and J. Nocedal. On the use of stochastic Hessian information in optimization methods for machine learning. *SIAM Journal on Optimization*, 21(3):977–995, 2011.

K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. Coordinate descent method for large-scale L2-loss linear SVM. *Journal of Machine Learning Research*, 9:1369–1398, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/cdl2.pdf`.

W.-L. Chiang, M.-C. Lee, and C.-J. Lin. Parallel dual coordinate descent method for large-scale linear classification in multi-core environments. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/multicore_cddual.pdf`.

C. Cortes and V. Vapnik. Support-vector network. *Machine Learning*, 20:273–297, 1995.

R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training SVM. *Journal of Machine Learning Research*, 6:1889–1918, 2005. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/quadworkset.pdf`.

References II

T. Glasmachers and U. Dogan. Accelerated coordinate descent with adaptive coordinate frequencies. In *Proceedings of the 5th Asian Conference on Machine Learning*, volume 29 of *Proceedings of Machine Learning Research*, pages 72–86, 2013.

C. Hildreth. A quadratic programming procedure. *Naval Research Logistics Quarterly*, 4: 79–85, 1957.

C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the Twenty Fifth International Conference on Machine Learning (ICML)*, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/cddual.pdf`.

Intel. *Intel Math Kernel Library Reference Manual*.

T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, pages 169–184, Cambridge, MA, 1998. MIT Press.

S. S. Keerthi and D. DeCoste. A modified finite Newton method for fast solution of large scale linear SVMs. *Journal of Machine Learning Research*, 6:341–361, 2005.

S. S. Keerthi and C.-J. Lin. Asymptotic behaviors of support vector machines with Gaussian kernel. *Neural Computation*, 15(7):1667–1689, 2003.

C.-P. Lee and S. J. Wright. Random permutations fix a worst case for cyclic coordinate descent, 2016. arXiv preprint arXiv:1607.08320.

References III

M.-C. Lee, W.-L. Chiang, and C.-J. Lin. Fast matrix-vector multiplications for large-scale logistic regression on shared-memory systems. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2015. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/multicore_liblinear_icdm.pdf`.

C.-J. Lin, R. C. Weng, and S. S. Keerthi. Trust region Newton method for large-scale logistic regression. *Journal of Machine Learning Research*, 9:627–650, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/logistic.pdf`.

O. L. Mangasarian. A finite Newton method for classification. *Optimization Methods and Software*, 17(5):913–929, 2002.

J. Martens. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010.

M. Martone. Efficient multithreaded untransposed, transposed or symmetric sparse matrix–vector multiplication with the recursive sparse blocks format. *Parallel Computing*, 40:251–270, 2014.

E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 130–136, 1997.

References IV

J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, Cambridge, MA, 1998. MIT Press.

S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14:567–599, 2013.