Large-scale Linear Classification

Chih-Jen Lin
Department of Computer Science
National Taiwan University

International Winter School on Big Data, 2016

Data Classification

- Given training data in different classes (labels known)

  Predict test data (labels unknown)
- Classic example: medical diagnosis

  Find a patient's blood pressure, weight, etc.

  After several years, know if he/she recovers

  Build a machine learning model

  New patient: find blood pressure, weight, etc

  Prediction
- Training and testing

Data Classification (Cont'd)

- Among many classification methods, linear and kernel are two popular ones
- They are very related
- We will detailedly discuss linear classification and its connection to kernel
- Talk slides:

  `http://www.csie.ntu.edu.tw/~cjlin/talks/course-bilbao.pdf`

Outline

Outline

Outline

Outline

Linear Classification

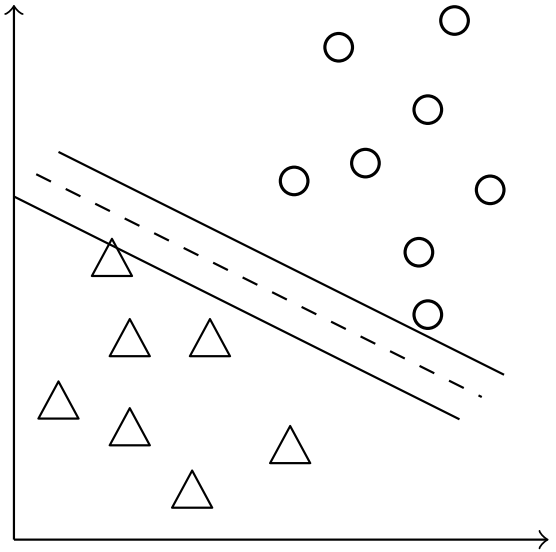
- Training vectors: $\boldsymbol{x}_i, i = 1, \ldots, l$
- Feature vectors. For example,

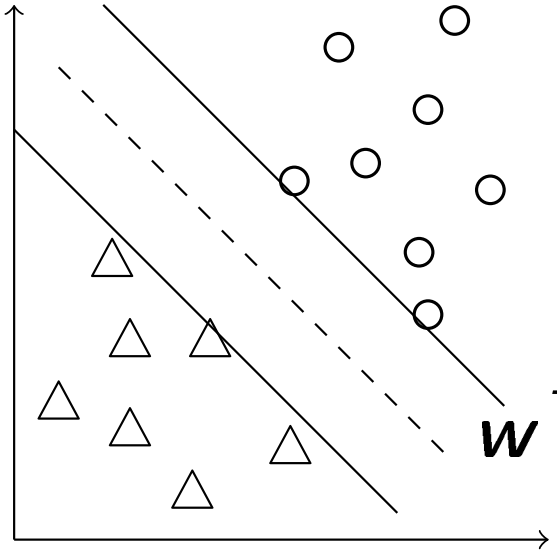
  A patient $=$ [height, weight, $\ldots$]^T
- Consider a simple case with two classes:

  Define an indicator vector $\mathbf{y} \in R^l$

$$y_i = \begin{cases} 1 & \text{if } \boldsymbol{x}_i \text{ in class 1} \\ -1 & \text{if } \boldsymbol{x}_i \text{ in class 2} \end{cases}$$

- A hyperplane to linearly separate all data





$$\boldsymbol{w}^T\boldsymbol{x} + b = \begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix}$$

- A separating hyperplane: $\boldsymbol{w}^T\boldsymbol{x} + b = 0$

$$\begin{aligned} (\boldsymbol{w}^T\boldsymbol{x}_i) + b \geq 1 \quad & \text{if } y_i = 1 \\ (\boldsymbol{w}^T\boldsymbol{x}_i) + b \leq -1 \quad & \text{if } y_i = -1 \end{aligned}$$

- Decision function $f(\boldsymbol{x}) = \text{sgn}(\boldsymbol{w}^T\boldsymbol{x} + b)$, $\boldsymbol{x}$: test data

Many possible choices of $\boldsymbol{w}$ and b

Maximal Margin

- Maximizing the distance between $\boldsymbol{w}^T\boldsymbol{x} + b = 1$ and -1:

$$2/\|\boldsymbol{w}\| = 2/\sqrt{\boldsymbol{w}^T\boldsymbol{w}}$$

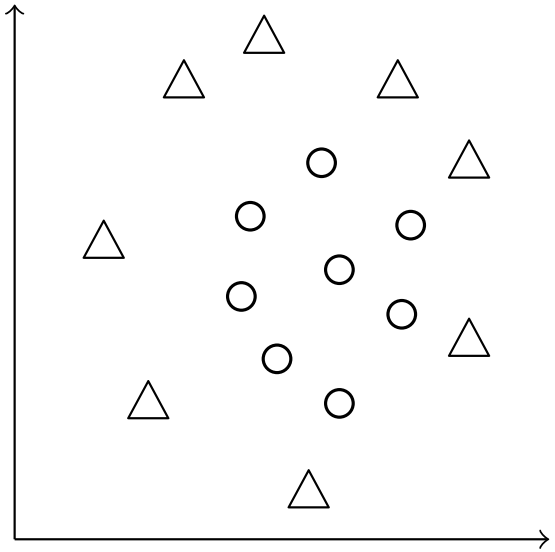
- A quadratic programming problem

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} \\ \text{subject to} \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1, \\ & i = 1, \ldots, l. \end{aligned}$$

- This is the basic formulation of support vector machines (Boser et al., 1992)

Data May Not Be Linearly Separable

- An example:



- We can never find a linear hyperplane to separate data
- Remedy: allow training errors

Data May Not Be Linearly Separable (Cont'd)

- Standard SVM (Boser et al., 1992; Cortes and Vapnik, 1995)

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0,\ i = 1, \ldots, l. \end{aligned}$$

- We explain later why this method is called support vector machine

The Bias Term b

- Recall the decision function is

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

- Sometimes the bias term b is omitted

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

  That is, the hyperplane always passes through the origin
- This is fine if the number of features is not too small
- In our discussion, b is used for kernel, but omitted for linear (due to some historical reasons)

Outline

Equivalent Optimization Problem

- Recall SVM optimization problem (without b) is

$$\begin{aligned} \min_{\boldsymbol{w},\boldsymbol{\xi}} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i\boldsymbol{w}^T\boldsymbol{x}_i \geq 1-\xi_i, \\ & \xi_i \geq 0,\ i = 1,\ldots,l. \end{aligned}$$

- It is equivalent to

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

- This reformulation is useful for subsequent discussion

Equivalent Optimization Problem (Cont'd)

- That is, at optimum,

$$\xi_i = \max(0, 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i)$$

- Reason: from constraint

$$\xi_i \geq 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i \text{ and } \xi_i \geq 0$$

but we also want to minimize ξ_i

Equivalent Optimization Problem (Cont'd)

- We now derive the same optimization problem (1) from a different viewpoint
- We now aim to minimize the training error

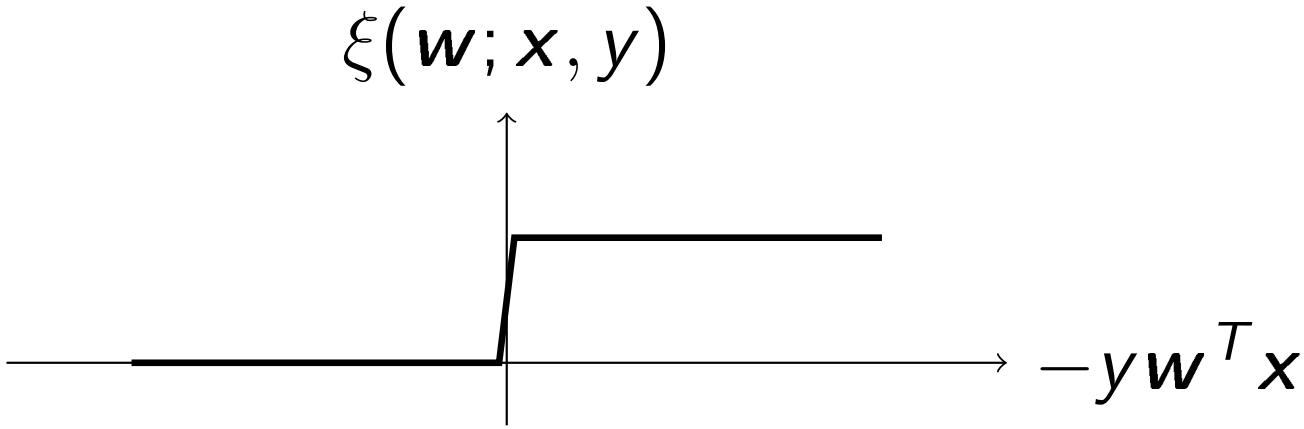
$$\min_{\boldsymbol{w}} \quad \text{(training errors)}$$

- 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:

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

Equivalent Optimization Problem (Cont'd)

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



- We need continuous approximations

Common Loss Functions

- Hinge loss (l1 loss)

$$\xi_{L1}(\boldsymbol{w}; \boldsymbol{x}, y) \equiv \max(0, 1 - y\boldsymbol{w}^T\boldsymbol{x}) \quad (2)$$

- Squared hinge loss (l2 loss)

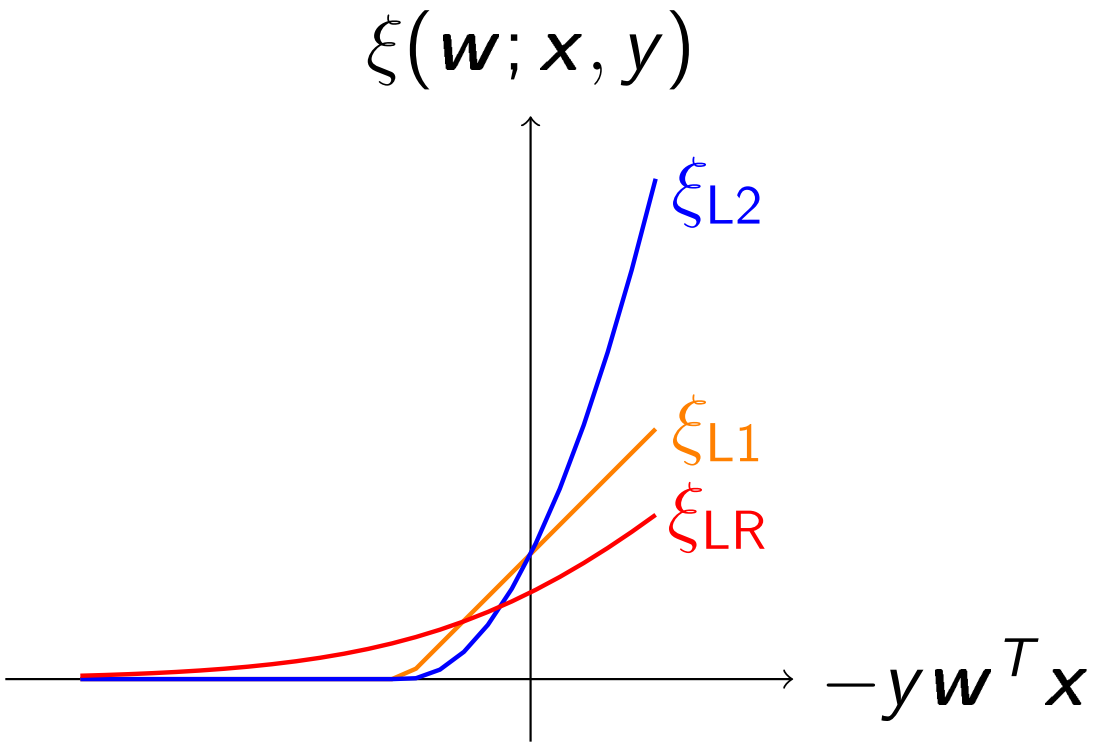
$$\xi_{L2}(\boldsymbol{w}; \boldsymbol{x}, y) \equiv \max(0, 1 - y\boldsymbol{w}^T\boldsymbol{x})^2 \quad (3)$$

- Logistic loss

$$\xi_{LR}(\boldsymbol{w}; \boldsymbol{x}, y) \equiv \log(1 + e^{-y\boldsymbol{w}^T\boldsymbol{x}}) \quad (4)$$

- SVM: (2)-(3). Logistic regression (LR): (4)

Common Loss Functions (Cont'd)



- Logistic regression is very related to SVM
- Their performance is usually similar

Common Loss Functions (Cont'd)

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

Overfitting

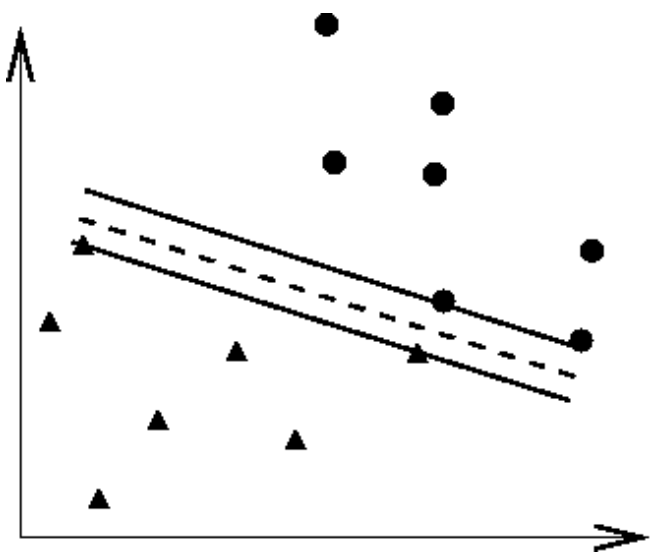
- See the illustration in the next slide
- For classification,

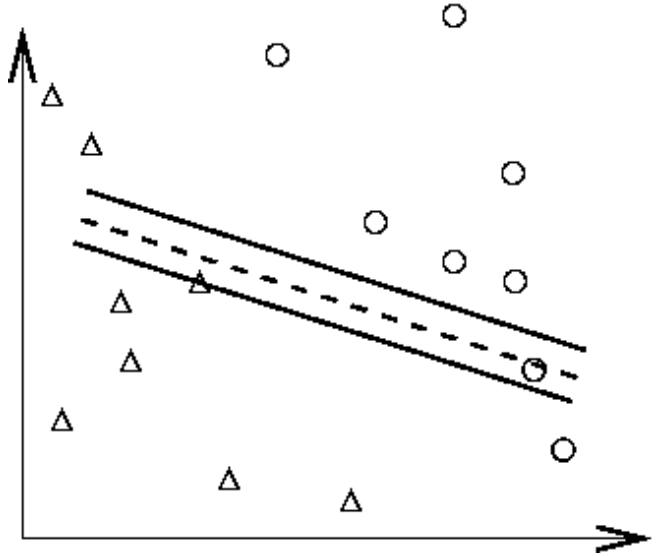
  You can easily achieve 100% training accuracy
- This is useless
- When training a data set, we should

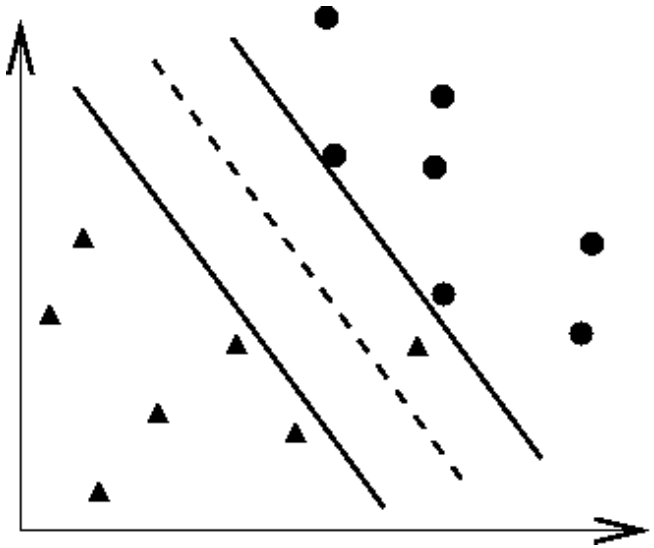
  Avoid underfitting: small training error

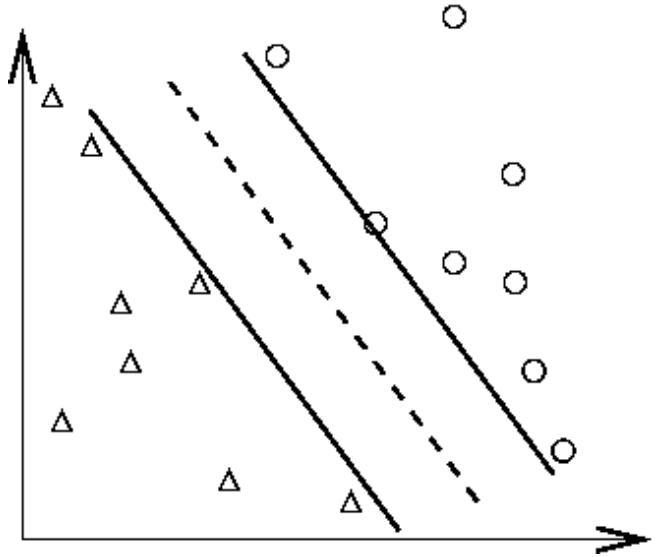
  Avoid overfitting: small testing error

● 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 the objective function smoother

General Form of 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

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}), \quad f(\boldsymbol{w}) \equiv \frac{\boldsymbol{w}^T \boldsymbol{w}}{2} + C \sum_{i=1}^{l} \xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i) \tag{5}$$

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

General Form of Linear Classification (Cont'd)

- If hinge loss

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

is used, then (5) goes back to the SVM problem described earlier (b omitted):

$$\begin{aligned} \min_{\boldsymbol{w},\boldsymbol{\xi}} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i\boldsymbol{w}^T\boldsymbol{x}_i \geq 1 - \xi_i, \\ & \xi_i \geq 0, \ i = 1, \ldots, l. \end{aligned}$$

Solving Optimization Problems

- We have an unconstrained problem, so many existing unconstrained optimization techniques can be used
- However,
  - ξ_{L1}: not differentiable
  - ξ_{L2}: differentiable but not twice differentiable
  - ξ_{LR}: twice differentiable
- We may need different types of optimization methods
- Details of solving optimization problems will be discussed later

Outline

Logistic Regression

- Logistic regression can be traced back to the 19th century
- It's mainly from statistics community, so many people wrongly think that this method is very different from SVM
- Indeed from what we have shown they are very related.
- Let's see how to derive it from a statistical viewpoint

Logistic Regression (Cont'd)

- For a label-feature pair $(y, \boldsymbol{x})$, assume the probability model

$$p(y|\boldsymbol{x}) = \frac{1}{1 + e^{-y\boldsymbol{w}^T\boldsymbol{x}}}.$$

- Note that

$$\begin{aligned} & p(1|\boldsymbol{x}) + p(-1|\boldsymbol{x}) \\ = & \frac{1}{1 + e^{-\boldsymbol{w}^T\boldsymbol{x}}} + \frac{1}{1 + e^{\boldsymbol{w}^T\boldsymbol{x}}} \\ = & \frac{e^{\boldsymbol{w}^T\boldsymbol{x}}}{1 + e^{\boldsymbol{w}^T\boldsymbol{x}}} + \frac{1}{1 + e^{\boldsymbol{w}^T\boldsymbol{x}}} \\ = & 1 \end{aligned}$$

Logistic Regression (Cont'd)

- Idea of this model

$$p(1|\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}}} \begin{cases} \to 1 & \text{if } \boldsymbol{w}^T\boldsymbol{x} \gg 0, \\ \to 0 & \text{if } \boldsymbol{w}^T\boldsymbol{x} \ll 0 \end{cases}$$

- Assume training instances are

$$(y_i, \boldsymbol{x}_i), i = 1, \ldots, l$$

Logistic Regression (Cont'd)

- Logistic regression finds $\boldsymbol{w}$ by maximizing the following likelihood

$$\max_{\boldsymbol{w}} \quad \prod_{i=1}^{l} p\left(y_i|\boldsymbol{x}_i\right). \tag{6}$$

- Negative log-likelihood

$$-\log \prod_{i=1}^{l} p\left(y_i|\boldsymbol{x}_i\right) = -\sum_{i=1}^{l} \log p\left(y_i|\boldsymbol{x}_i\right)$$

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

Logistic Regression (Cont'd)

- Logistic regression

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

- Regularized logistic regression

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l} \log\left(1 + e^{-y_i \boldsymbol{w}^T \boldsymbol{x}_i}\right). \qquad (7)$$

C: regularization parameter decided by users

Discussion

We see that the same method can be derived from different ways

SVM

- Maximal margin
- Regularization and training losses

LR

- Regularization and training losses
- Maximum likelihood

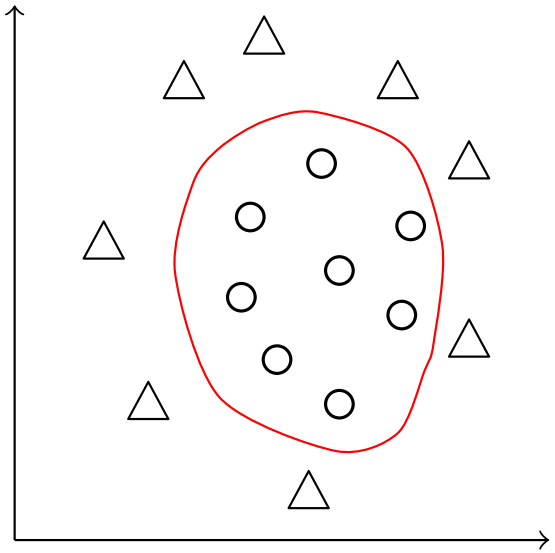
Outline

Outline

Outline

Data May Not Be Linearly Separable

- This is an earlier example:



- In addition to allowing training errors, what else can we do?
- For this data set, shouldn't we use a nonlinear classifier?

Mapping Data to a Higher Dimensional Space

- But modeling nonlinear curves is difficult. Instead, 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

Kernel Support Vector Machines

- Linear SVM:

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0,\ i = 1,\ldots,l. \end{aligned}$$

- Kernel SVM:

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{w}^T\phi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0,\ i = 1,\ldots,l. \end{aligned}$$

Kernel Logistic Regression

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

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

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

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

Outline

Kernel Tricks

- It can be shown 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. Later we will show that $\boldsymbol{\alpha}$ is the solution of a dual problem

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

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

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

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

Outline

Outline

Outline

Linear and Kernel Classification

Now we see that 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 $\phi(\cdot)$
- Linear classification + feature engineering:

  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

- The cost of using linear and kernel classification is different
- Let's check the prediction cost

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

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

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

- Linear is much cheaper
- A similar difference occurs for training

Linear and Kernel Classification (Cont'd)

- In fact, linear is a special case of kernel
- We can prove that accuracy of linear is the same as Gaussian (RBF) kernel under certain parameters (Keerthi and Lin, 2003)
- Therefore, roughly we have

$$\begin{aligned} \text{accuracy:} &\quad \text{kernel} \geq \text{linear} \\ \text{cost:} &\quad \text{kernel} \gg \text{linear} \end{aligned}$$

- Speed is the reason to use linear

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)

Outline

Comparison Between Linear and Kernel (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 Kernel (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 Kernel (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

Outline

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

- We may directly train $\phi(\boldsymbol{x}_i), \forall i$ without using kernel
- This is possible only if $\phi(\boldsymbol{x}_i)$ is not too high dimensional
- Next we show a real example of running a machine learning model is a small sensor hub

Example: Classifier in a Small Device

- In a sensor application (Yang, 2013), the classifier can use less than 16KB of RAM

Classifiers	Test accuracy	Model Size
Decision Tree	77.77	76.02KB
AdaBoost (10 trees)	78.84	1,500.54KB
SVM (RBF kernel)	85.33	1,287.15KB

- Number of features: 5
- We consider a degree-3 polynomial mapping

$$\text{dimensionality} = \binom{5+3}{3} + \text{ bias term} = 57.$$

Example: Classifier in a Small Device

- One-against-one strategy for 5-class classification

$$\binom{5}{2} \times 57 \times 4\text{bytes} = 2.28\text{KB}$$

  Assume single precision

- Results

SVM method	Test accuracy	Model Size
RBF kernel	85.33	1,287.15KB
Polynomial kernel	84.79	2.28KB
Linear kernel	78.51	0.24KB

Outline

Outline

Outline

Dual Problem

- Recall we said that the difficulty after mapping $\boldsymbol{x}$ to $\phi(\boldsymbol{x})$ is the huge number of variables
- We mentioned

$$\boldsymbol{w} = \sum_{i=1}^{l} \alpha_i y_i \phi(\boldsymbol{x}_i) \tag{8}$$

  and used kernels for prediction
- Besides prediction, we must do training via kernels
- The most common way to train SVM via kernels is through its dual problem

Dual Problem (Cont'd)

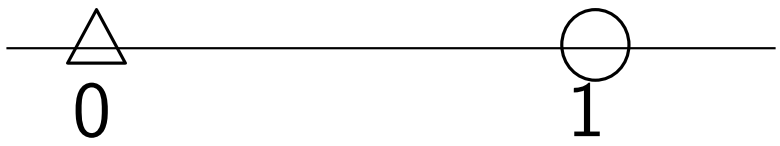
- The dual problem

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

  where $Q_{ij} = y_i y_j \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$ and $\boldsymbol{e} = [1, \dots, 1]^T$
- From primal-dual relationship, at optimum (8) holds
- Dual problem has a finite number of variables
- If no bias term b, then $\mathbf{y}^T \boldsymbol{\alpha} = 0$ disappears

Example: Primal-dual Relationship

- Consider the earlier example:



- Now two data are $\boldsymbol{x}_1 = 1, \boldsymbol{x}_2 = 0$ with

$$\mathbf{y} = [+1, -1]^T$$

- The solution is $(w, b) = (2, -1)$

Example: Primal-dual Relationship (Cont'd)

- The dual objective function

$$\begin{aligned} & \frac{1}{2}\begin{bmatrix}\alpha_1 & \alpha_2\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}\alpha_1\\ \alpha_2\end{bmatrix} - \begin{bmatrix}1 & 1\end{bmatrix}\begin{bmatrix}\alpha_1\\ \alpha_2\end{bmatrix} \\ = & \frac{1}{2}\alpha_1^2 - (\alpha_1 + \alpha_2) \end{aligned}$$

- In optimization, objective function means the function to be optimized
- Constraints are

$$\alpha_1 - \alpha_2 = 0, 0 \le \alpha_1, 0 \le \alpha_2.$$

Example: Primal-dual Relationship (Cont'd)

- Substituting $\alpha_2 = \alpha_1$ into the objective function,

$$\frac{1}{2}\alpha_1^2 - 2\alpha_1$$

  has the smallest value at $\alpha_1 = 2$.
- Because $[2, 2]^T$ satisfies constraints

$$0 \leq \alpha_1 \text{ and } 0 \leq \alpha_2,$$

  it is optimal

Example: Primal-dual Relationship (Cont'd)

- Using the primal-dual relation

$$\begin{aligned} w &= y_1\alpha_1 x_1 + y_2\alpha_2 x_2 \\ &= 1 \cdot 2 \cdot 1 + (-1) \cdot 2 \cdot 0 \\ &= 2 \end{aligned}$$

- This is the same as that by solving the primal problem.

Decision function

- At optimum

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

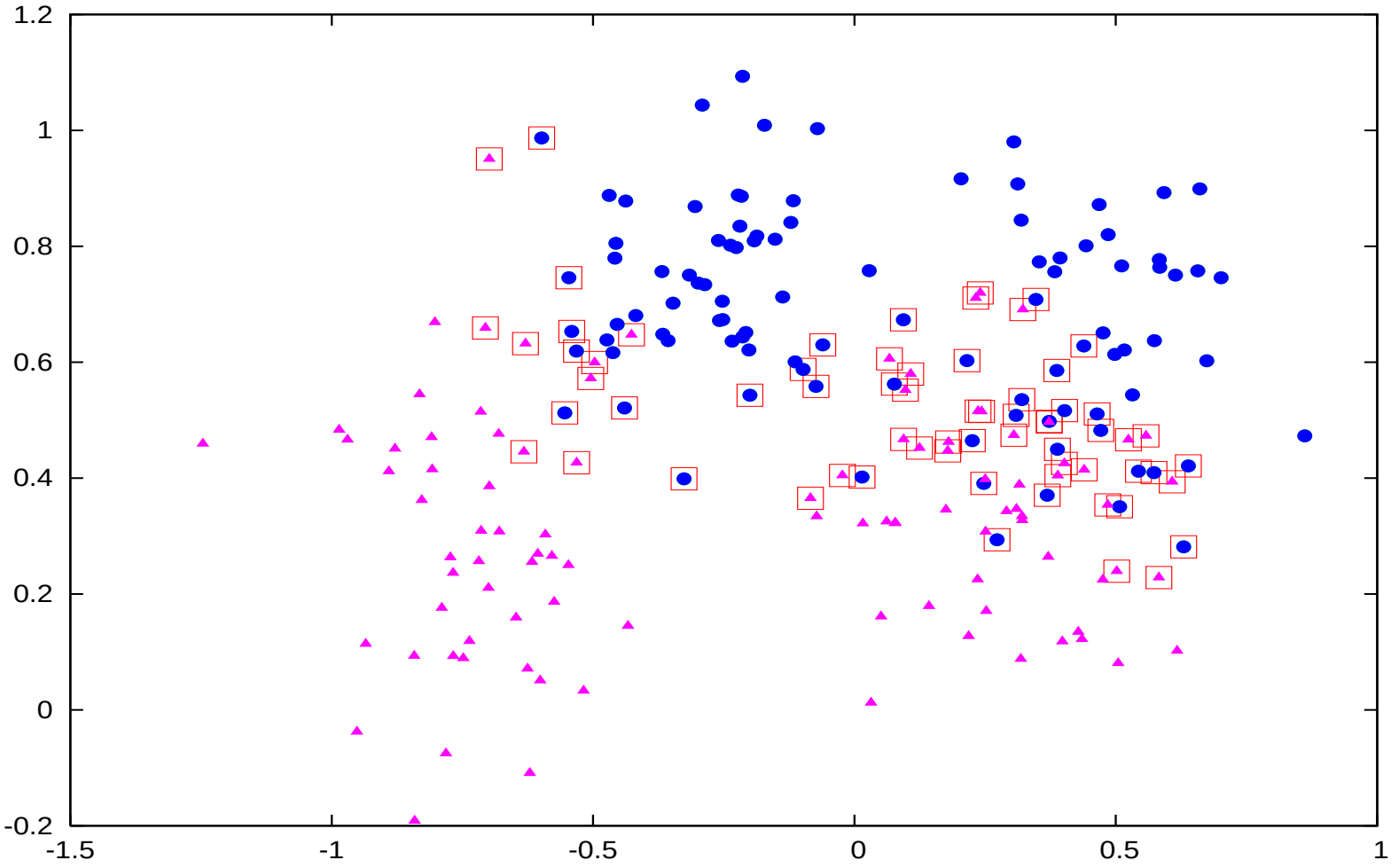
- Decision function

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

- Recall $0 \leq \alpha_i \leq C$ in the dual problem

Support Vectors

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)

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

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 $\boldsymbol{d}_B$

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

- The objective function of the sub-problem

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

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?

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 $\boldsymbol{\alpha}$

$$\text{decision function: } \sum_{i=1}^{l} y_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b$$

Prediction may still be correct with a rough $\boldsymbol{\alpha}$

- Further, in some situations,

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

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

How Decomposition Methods Perform? (Cont'd)

- An example of training 50,000 instances using the software LIBSVM ($|B| = 2$)

```
$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

Outline

Coordinate Descent Methods for Linear Classification

- We consider L1-loss SVM as an example here
- The same method can be extended to L2 and logistic loss
- More details in Hsieh et al. (2008); Yu et al. (2011)

SVM Dual (Linear without Kernel)

- 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

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

and

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

- No linear constraint $\mathbf{y}^T \boldsymbol{\alpha} = 0$ because of no bias term b

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

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

- This sub-problem is a special case of the earlier sub-problem of the decomposition method for kernel classifiers
- That is, the working set $B = \{i\}$
- Without constraints

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

The 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

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

The Procedure (Cont'd)

- Directly calculating gradients costs $O(ln)$

  l:# data, n: # features

  This is the case for kernel classifiers
- For linear SVM, define

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

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

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

The Procedure (Cont'd)

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

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

- If

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

then

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

Also costs $O(n)$

Algorithm: Dual Coordinate Descent

- 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

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

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

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 (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 variables (i.e., select the set B) for update
- In optimization there are two types of coordinate descent methods:

  sequential or random selection of variables

  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 sequential or random selection

  Existing coordinate descent methods for kernel $\Rightarrow$ related to greedy selection

Bias Term b and Linear Constraint in Dual

- In our discussion, b is used for kernel but not linear
- Mainly history reason
- For kernel SVM, we can also omit b to get rid of the linear constraint $\mathbf{y}^T \boldsymbol{\alpha} = 0$
- Then for kernel decomposition method, $|B| = 1$ can also be possible

Outline

Optimization for Linear and Kernel Cases

- Recall that

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

- Kernel: can only solve an optimization problem of $\boldsymbol{\alpha}$
- 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 Special Properties of Data Classification

- But Hessian has a special form

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

- D diagonal. For logistic regression,

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

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

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

Using Special 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 Special 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

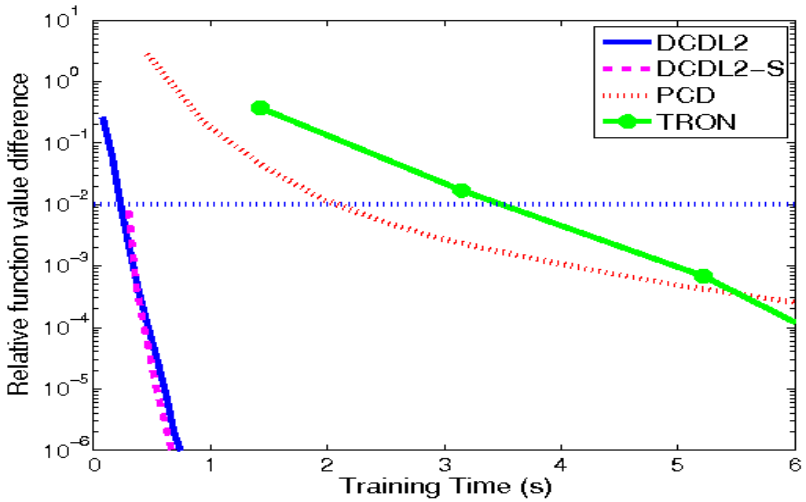
Outline

Comparisons

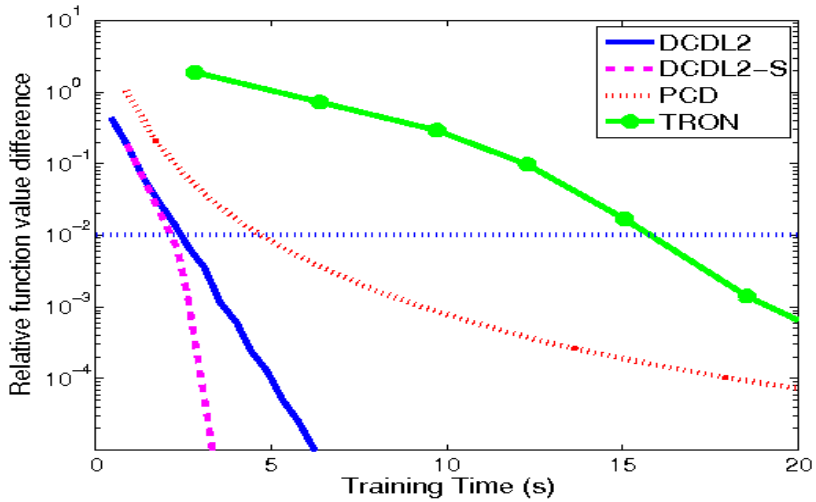
L2-loss SVM is used

- 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)

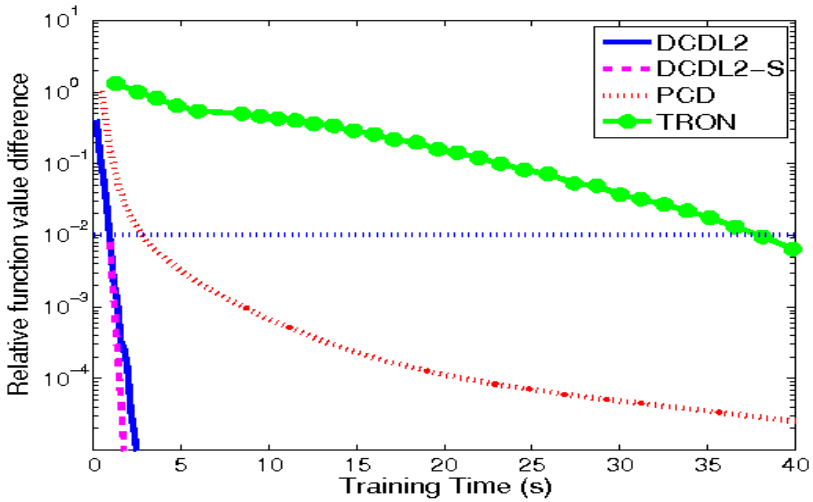
Objective values (Time in Seconds)



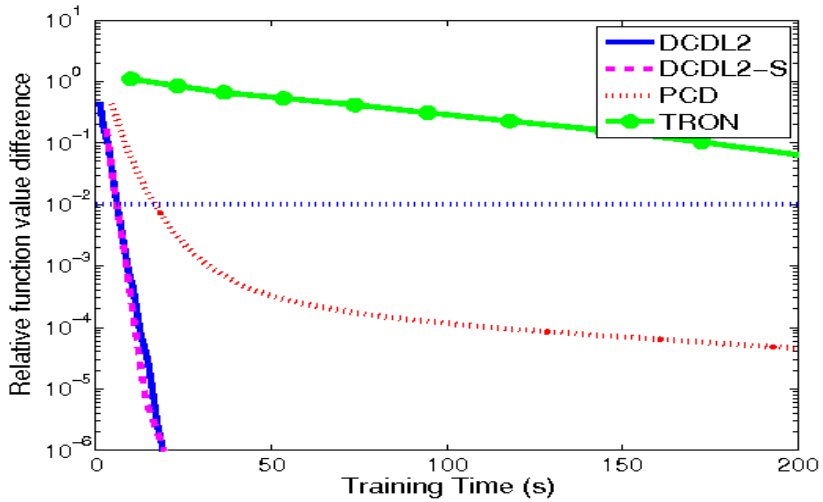
news20



rcv1



yahoo-japan



yahoo-korea

Analysis

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

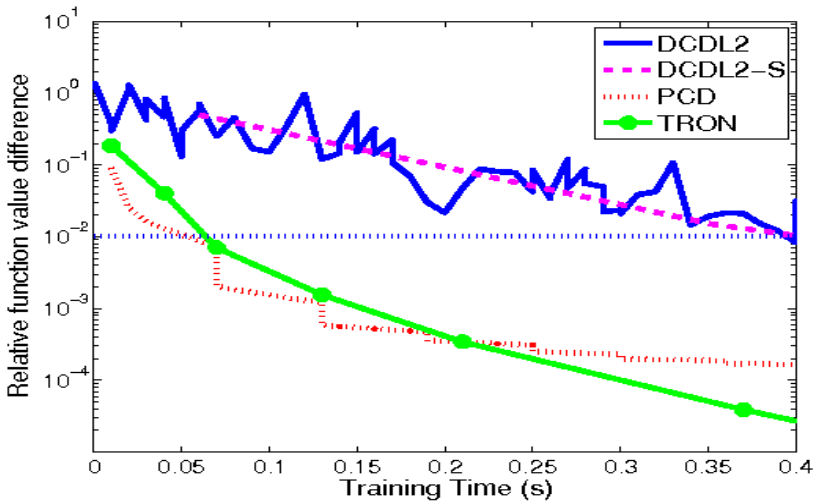
  Useful for document classification
- Half million data in a few seconds
- However, it is less effective if

  # features small: should solve primal; or

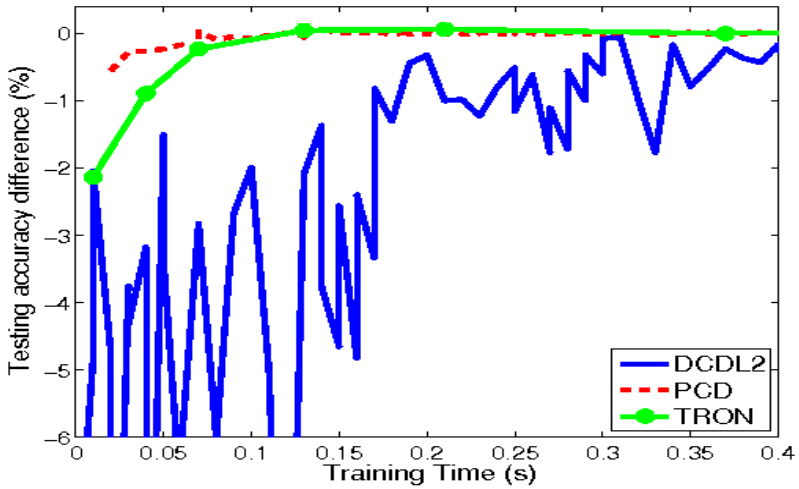
  large penalty parameter C; problems are more ill-conditioned

An Example When # Features Small

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



Objective value



Accuracy

Outline

Outline

Multi-core Linear Classification

- Parallelization in shared-memory system: use the power of multi-core CPU if data can fit in memory
- Example: we can parallelize the 2nd-order method (i.e., the Newton method) discussed earlier.
- We discuss the study in Lee et al. (2015)
- Recall the bottleneck is the Hessian-vector product

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

See the analysis in the next slide

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	518,571	47,236	97.04%
covtype	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

$$\mathbf{u} = X\boldsymbol{d} \tag{9}$$

$$\mathbf{u} \leftarrow D\mathbf{u} \tag{10}$$

$$\bar{\mathbf{u}} = X^T\mathbf{u}, \text{ where } \mathbf{u} = DX\boldsymbol{d} \tag{11}$$

- Because D is diagonal, (10) is easy
- We will discuss the parallelization of (9) and (11)

Outline

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, ..., l do in parallel
2:     for (x_i)_s ≠ 0 do
3:         atomic: ū_s ← ū_s + u_i (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 arrays maintained by each thread, and summing up them in the end
- That is, store

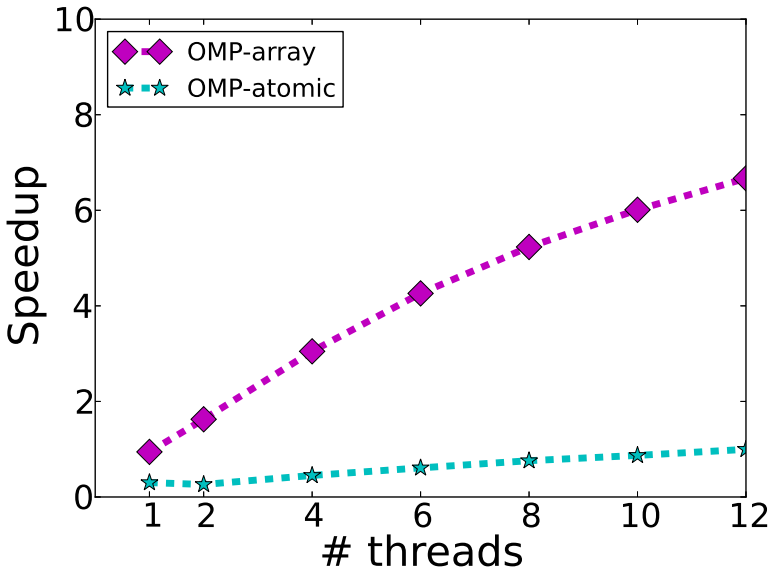
$$\hat{\mathbf{u}}^p = \sum_i \{u_i \mathbf{x}_i \mid i \text{ run by thread p}\}$$

and then

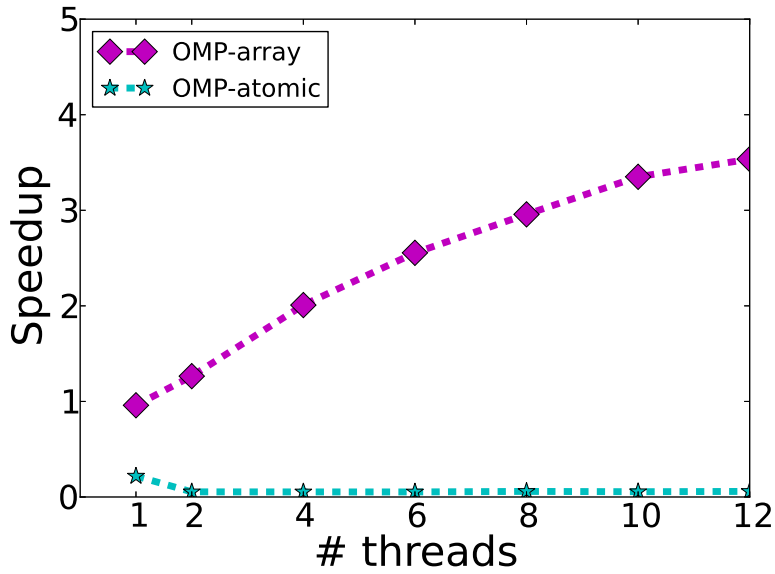
$$\bar{\mathbf{u}} = \sum_p \hat{\mathbf{u}}^p$$

Atomic Operation: Almost No Speedup

- Reduce operations are superior to atomic operations



rcv1_binary



covtype_binary

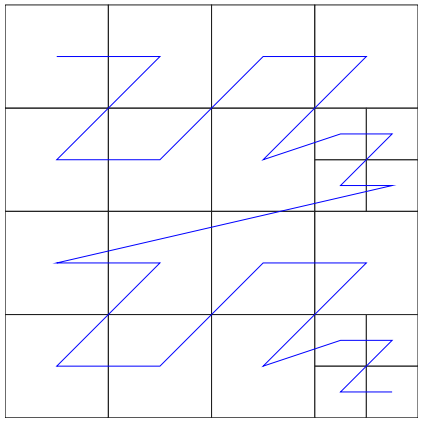
- Subsequently we use the reduce operations

Existing Algorithms for Sparse Matrix-vector Product

- This is always an important research issue in numerical analysis
- 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 experiment part

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.

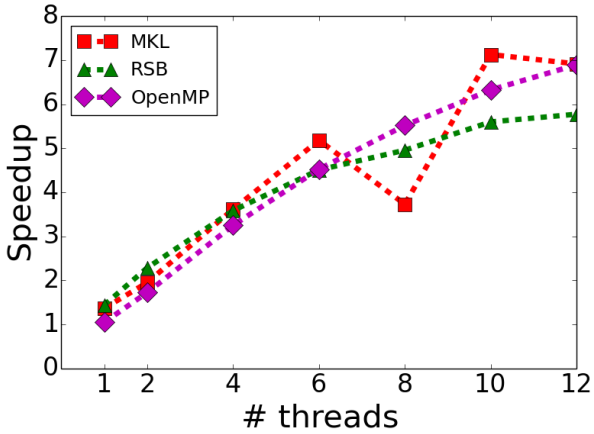
Outline

Experiments

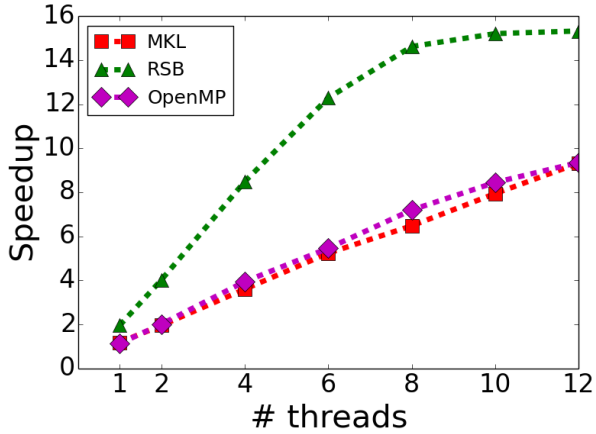
- Baseline: Single core version in LIBLINEAR 1.96
- OpenMP to parallelize loops
- MKL: Intel MKL version 11.2
- RSB: librsb version 1.2.0

Speedup of Xd: All are Excellent

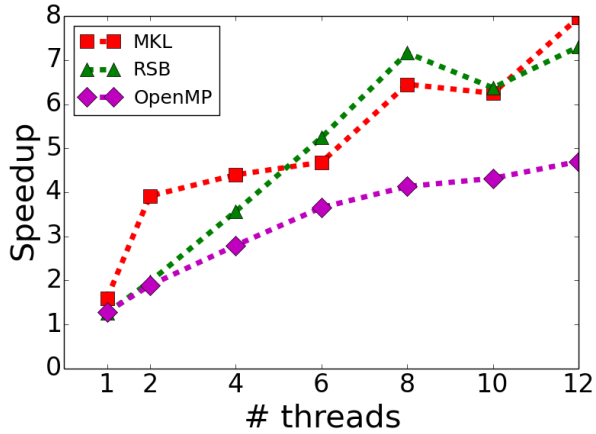
rcv1_binary



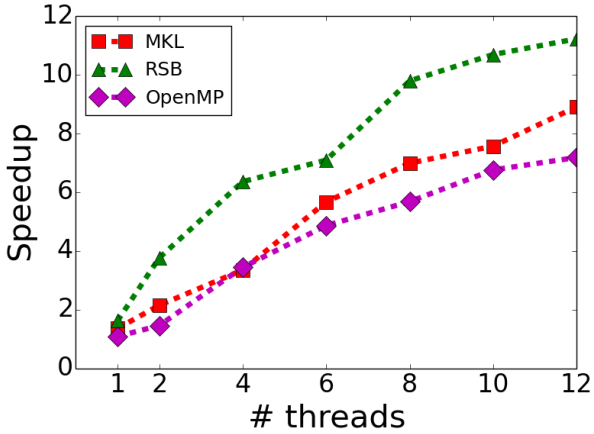
webspam



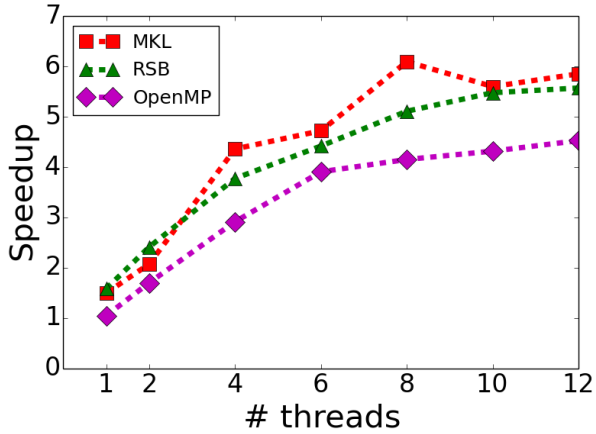
kddb



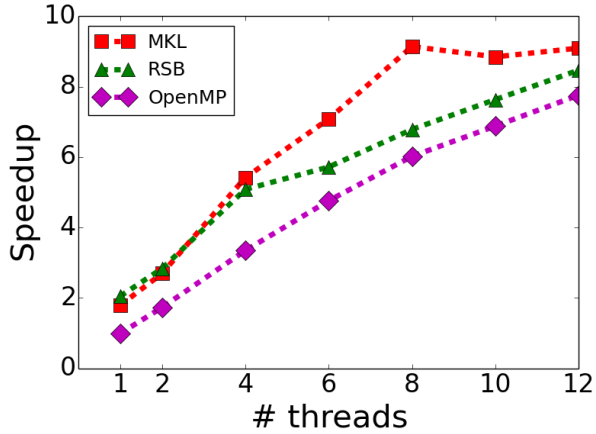
url_combined



covtype_binary

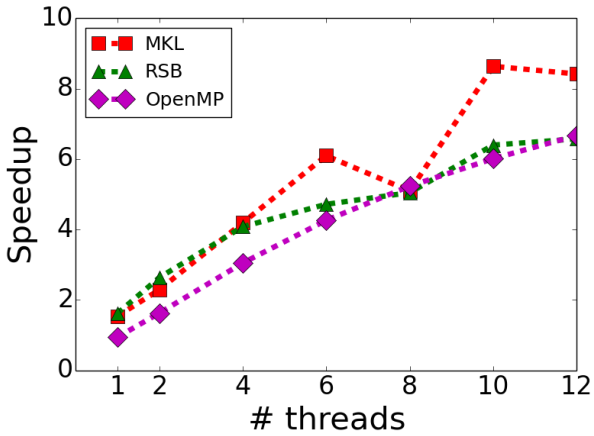


rcv1

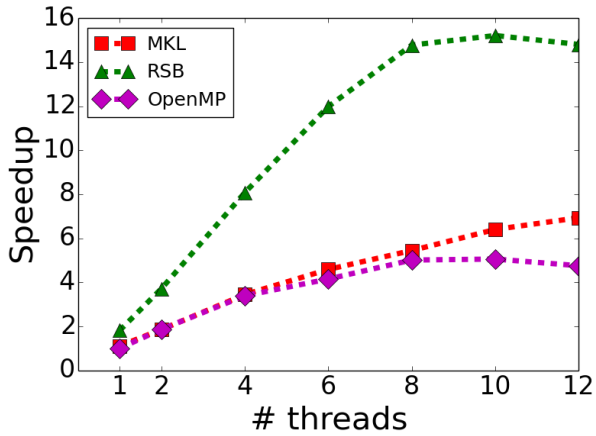


More Difficult to Speed up $X^T\mathbf{u}$

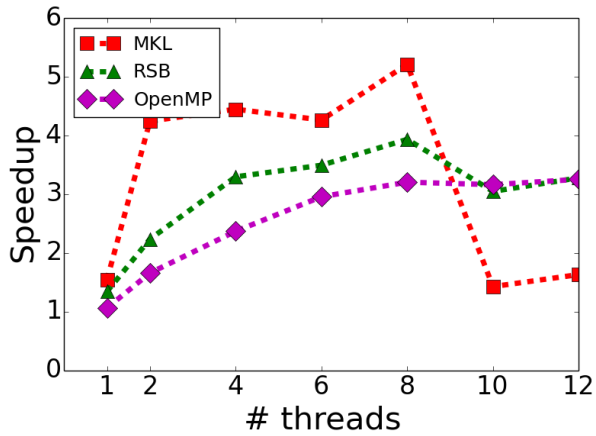
rcv1_binary



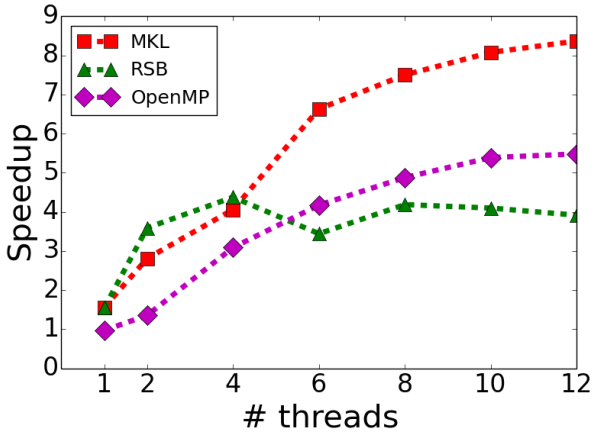
webspam



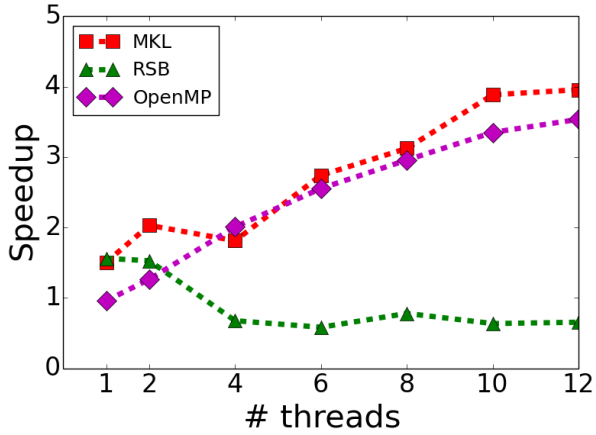
kddb



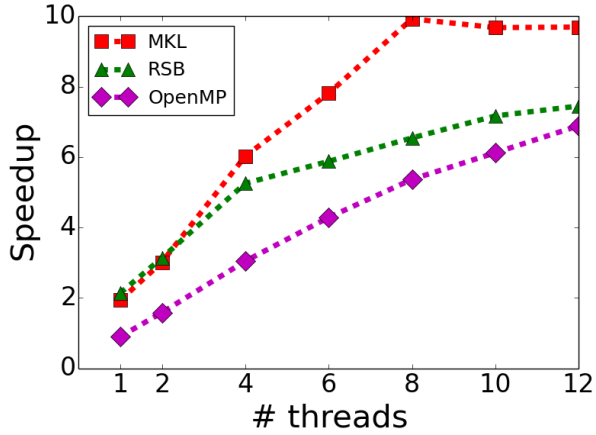
url_combined



covtype_binary



rcv1



Reducing Memory Access to Improve Speedup

- In computing

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

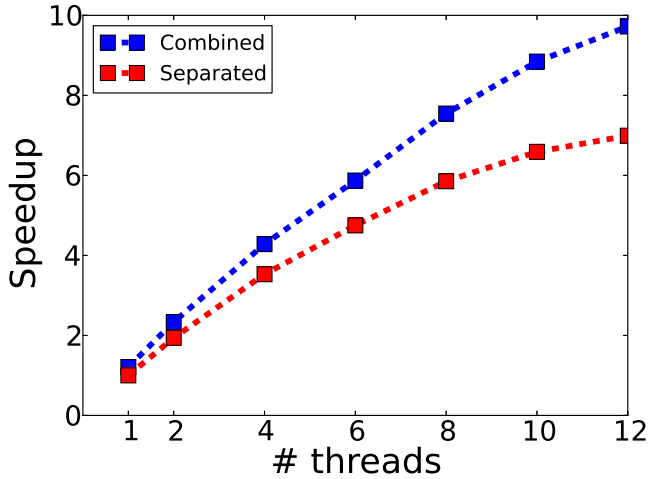
  the data matrix is accessed twice
- We notice that these two operations can be combined together

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

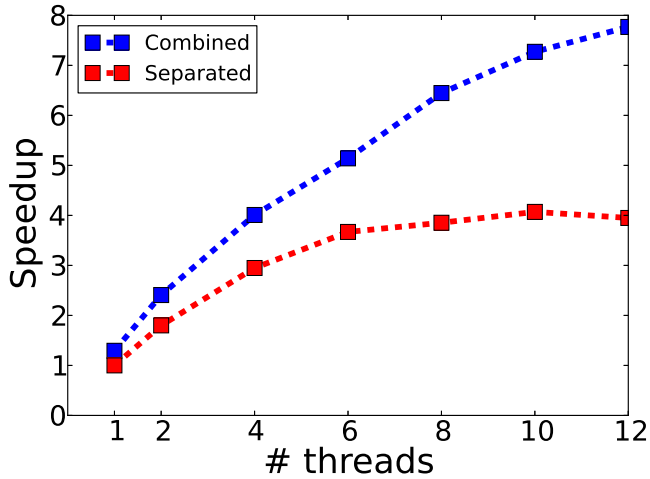
- We can parallelize one single loop by OpenMP

Reducing Memory Access to Improve Speedup (Cont'd)

- Better speedup as memory accesses reduced



rcv1_binary



covtype_binary

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

OpenMP Scheduling

- An OpenMP loop assigns tasks to different threads.
- The default `schedule(static)` splits indices to P blocks, where each contains l/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)

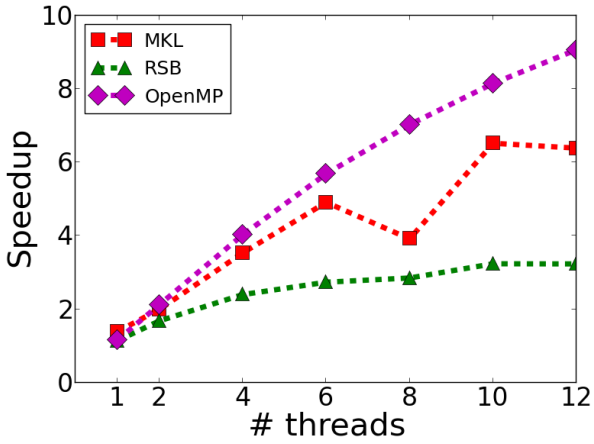
- Consider the second step

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

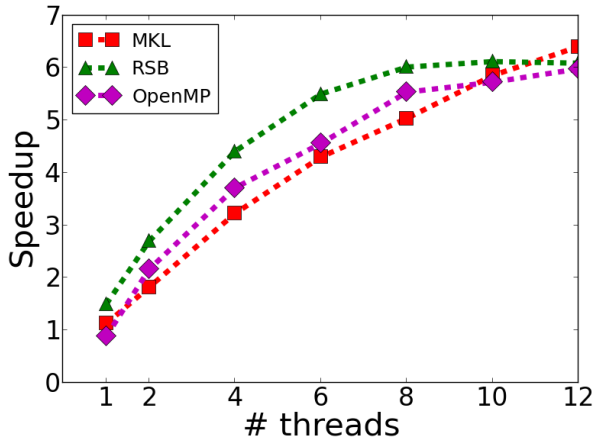
- Clearly, a suitable scheduling is essential
- The other two steps are more balanced, so `schedule(static)` is used (details omitted)

Speedup of Total Training Time

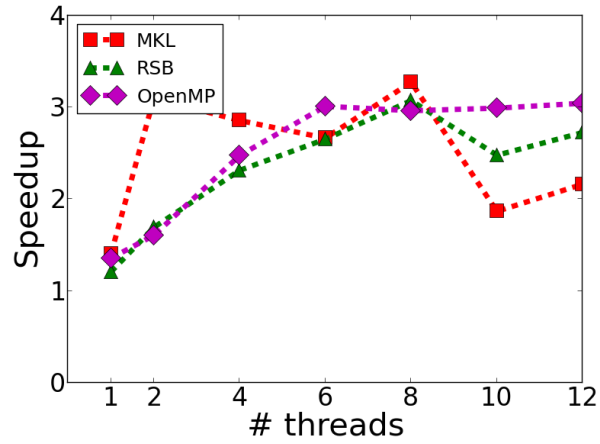
rcv1_binary



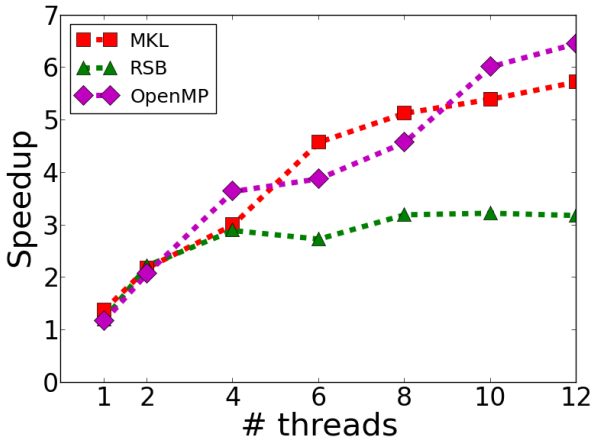
webspam



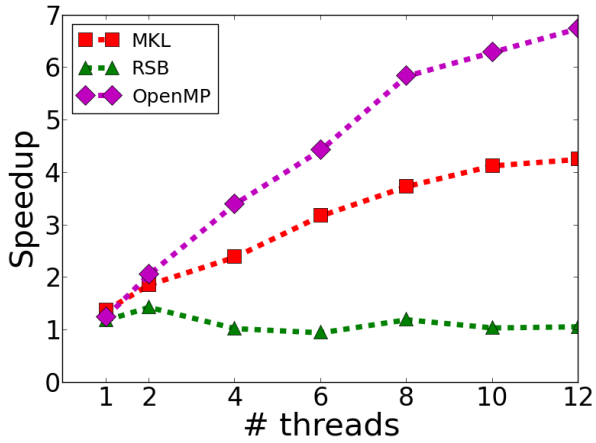
kddb



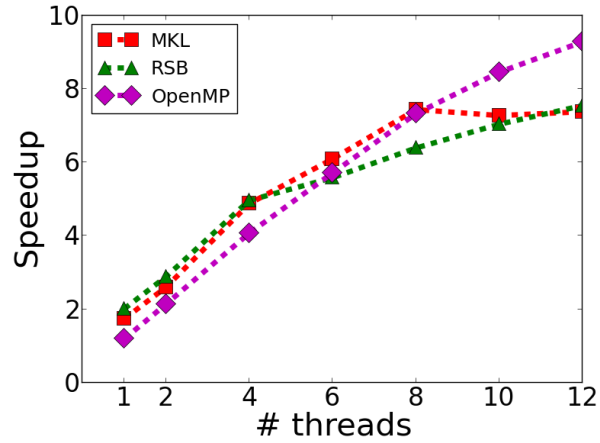
url_combined



covtype_binary



rcv1



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

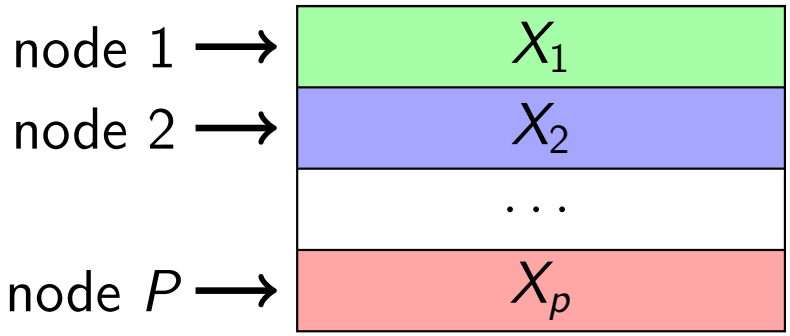
Outline

Outline

Outline

Data in a Distributed System

- When data are too large, we may for example let each node store some instances



- We would like to distributedly compute

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

Parallel Hessian-vector Product

- Like in the shared-memory situation, we have

$$X^T D X \boldsymbol{d} = X_1^T D_1 X_1 \boldsymbol{d} + \cdots + X_P^T D_P X_P \boldsymbol{d}$$

- We let each node calculate

$$X_p^T D_p X_p \boldsymbol{d}$$

  and sum resulting vectors together
- This is a reduce operation. We have used similar techniques for multi-core situations

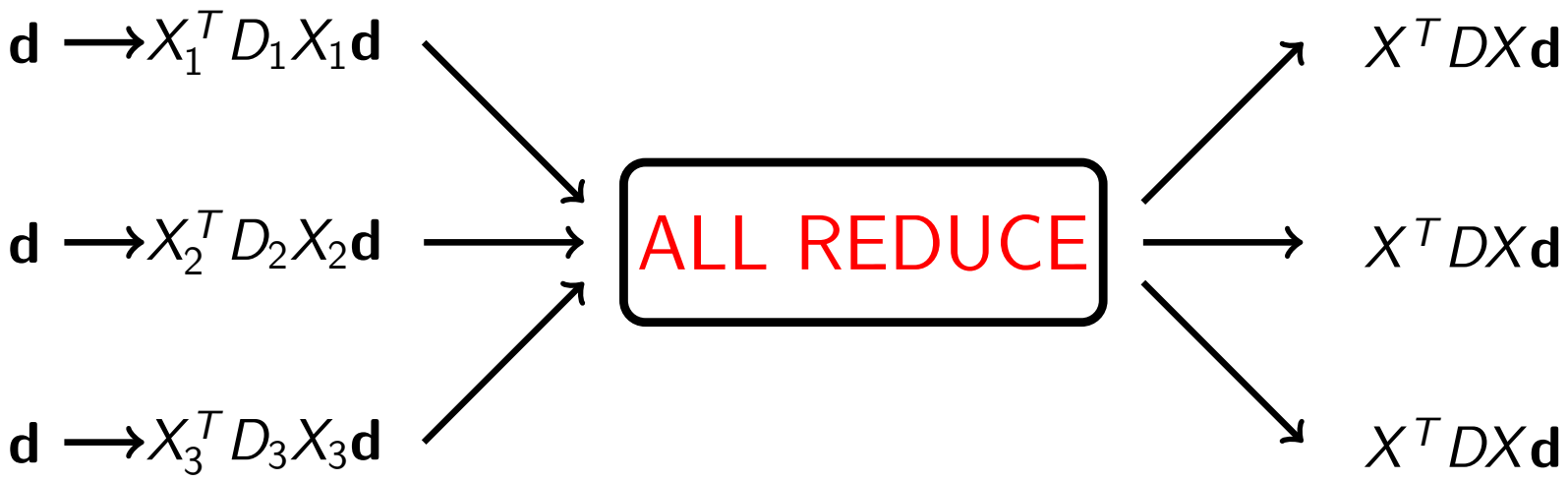
Master-slave or Master-master

- Master-slave: only master gets $X^T DX\boldsymbol{d}$ and runs the whole Newton method
- Master-master: every node gets $X^T DX\boldsymbol{d}$. Then each node has all the information to finish the Newton method
- Here we consider a master-master setting.

  One reason is that for master-slave, implementations on master and slaves are different
- This is different from multi-core situations, where only one master copy is run

Allreduce Operation

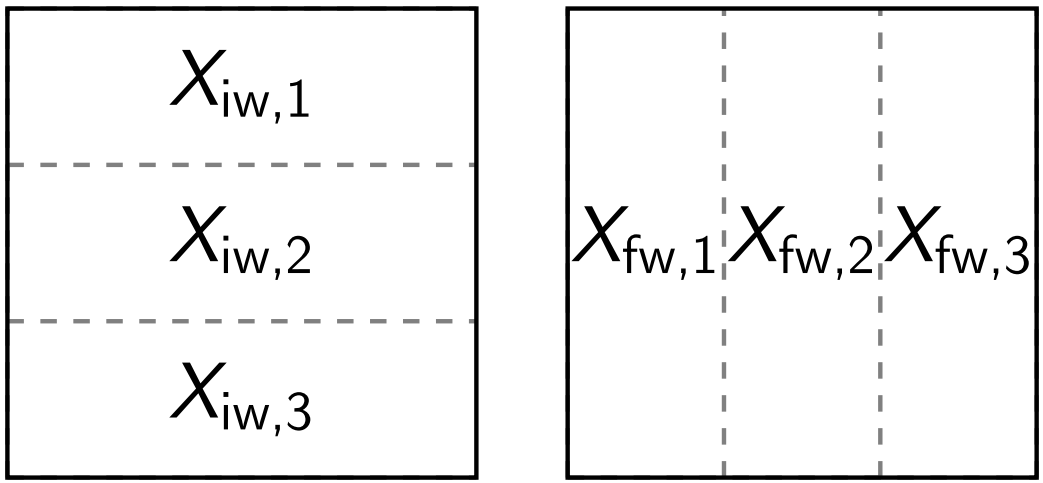
We let every node get $X^T DX\mathbf{d}$



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

Instance-wise and Feature-wise Data Splits

- Instead of storing a subset of data at each node, we can store a subset of features



Instance-wise Feature-wise

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

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

$$(\nabla^2 f(\boldsymbol{w})\boldsymbol{d})_{\text{fw},1} = \boldsymbol{d}_1 + CX_{\text{fw},1}^T D(X_{\text{fw},1}\boldsymbol{d}_1 + \cdots + X_{\text{fw},P}\boldsymbol{d}_P)$$

- $X_{\text{fw},1}\boldsymbol{d}_1 + \cdots + X_{\text{fw},P}\boldsymbol{d}_P \in R^l$ must be available on all nodes (by allreduce)
- Because

$$X_k^T D_k X_k \boldsymbol{d} : O(n), \qquad X_{\text{fw},p}\boldsymbol{d}_p : O(l),$$

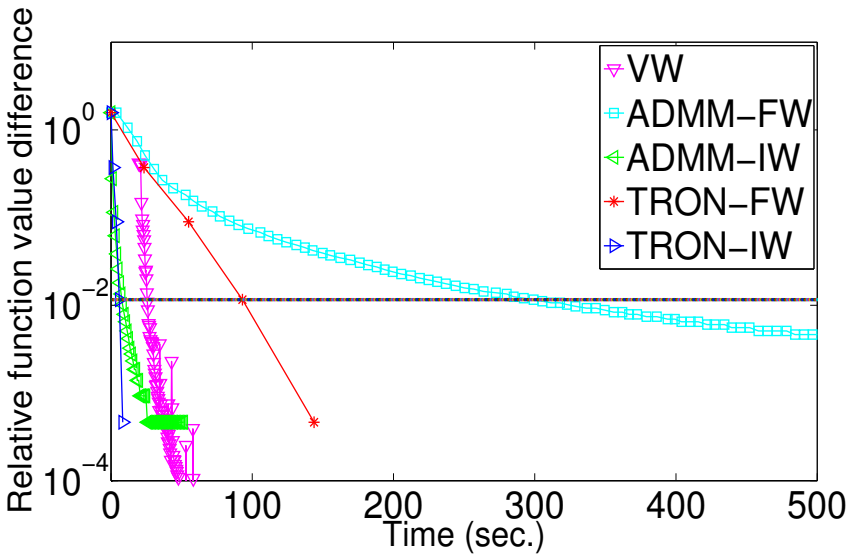
amount of data moved per Hessian-vector product:
Instance-wise: $O(n)$, Feature-wise: $O(l)$

Outline

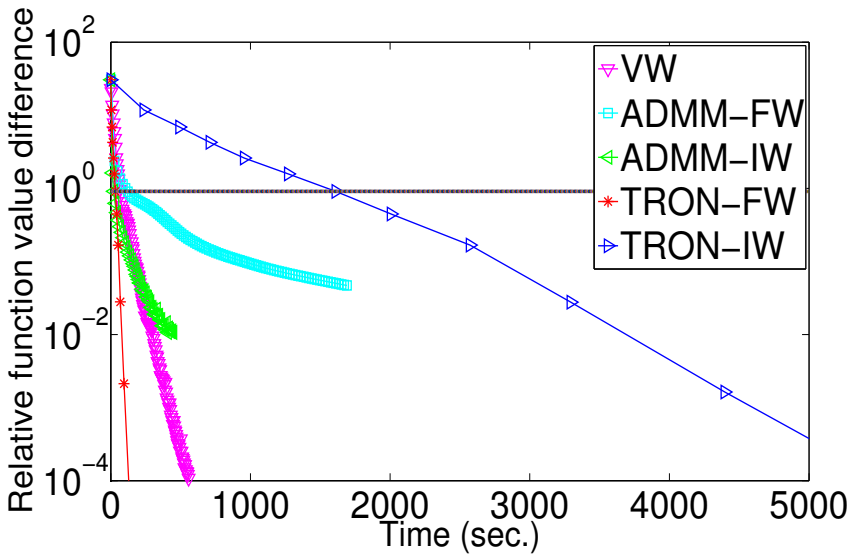
Experiments

- We compare
  - TRON: Newton method
  - ADMM: alternating direction method of multipliers (Boyd et al., 2011; Zhang et al., 2012)
  - Vowpal_Wabbit (Langford et al., 2007)
- TRON and ADMM are implemented by MPI
- Details in Zhuang et al. (2015)

Experiments (Cont'd)



epsilon



webspam

- 32 machines are used
- Horizontal line: test accuracy has stabilized
- Instance-wise and feature-wise splits useful for $l \gg n$ and $l \ll n$, respectively

Experiments (Cont'd)

- We have seen that communication cost is a big concern
- In terms of running time, multi-core implementation is often faster
- However, data preparation and loading are issues other than running time
- Overall we see that distributed training is a complicated issue

Outline

Outline

Outline

Software

- Most materials in this talks are based on our experiences in developing two popular software
- Kernel: LIBSVM (Chang and Lin, 2011)

  `http://www.csie.ntu.edu.tw/~cjlin/libsvm`
- Linear: LIBLINEAR (Fan et al., 2008).

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

  See also a survey on linear classification in Yuan et al. (2012)

Multi-core LIBLINEAR

- An extension of the software LIBLINEAR
- See `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/multicore-liblinear`
- This is based on the study in Lee et al. (2015)
- We already have many users. For example, one user from USC uses this tool to reduce his training time from over 30 hours to 5 hours

Distributed LIBLINEAR

- An extension of the software LIBLINEAR
- See `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/distributed-liblinear`
- We support both MPI (Zhuang et al., 2015) and Spark (Lin et al., 2014)

Outline

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
- Linear classification will continue to be used in situations ranging from small-model to big-data applications

Acknowledgments

- Many students have contributed to our research on large-scale linear classification
- We also thank the partial support from Ministry of Science and Technology in Taiwan

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.

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

C.-C. Chang and C.-J. Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

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`.

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

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: a library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf`.

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

References II

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.

J. Langford, L. Li, and A. Strehl. Vowpal Wabbit, 2007. `https://github.com/JohnLangford/vowpal_wabbit/wiki`.

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`.

References III

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`.

C.-Y. Lin, C.-H. Tsai, C.-P. Lee, and C.-J. Lin. Large-scale logistic regression and linear support vector machines using Spark. In *Proceedings of the IEEE International Conference on Big Data*, pages 519–528, 2014. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/spark-liblinear/spark-liblinear.pdf`.

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.

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.

T. Yang. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems 26*, pages 629–637. 2013.

References IV

H.-F. Yu, F.-L. Huang, and C.-J. Lin. Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning*, 85(1-2):41–75, October 2011. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/maxent_dual.pdf`.

G.-X. Yuan, C.-H. Ho, and C.-J. Lin. An improved GLMNET for l1-regularized logistic regression. *Journal of Machine Learning Research*, 13:1999–2030, 2012. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/l1_glmnet/long-glmnet.pdf`.

C. Zhang, H. Lee, and K. G. Shin. Efficient distributed linear classification algorithms via the alternating direction method of multipliers. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, 2012.

Y. Zhuang, W.-S. Chin, Y.-C. Juan, and C.-J. Lin. Distributed Newton method for regularized logistic regression. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2015.