# Large-scale Linear and Kernel Classification

Chih-Jen Lin
Department of Computer Science
National Taiwan University

MSR India Summer School 2015 on Machine Learning

## 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 discuss these two topics in detail in this lecture
- Talk slides:

  `http://www.csie.ntu.edu.tw/~cjlin/talks/msri.pdf`

## Outline

## Outline

## Outline

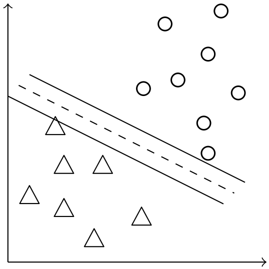
## Outline

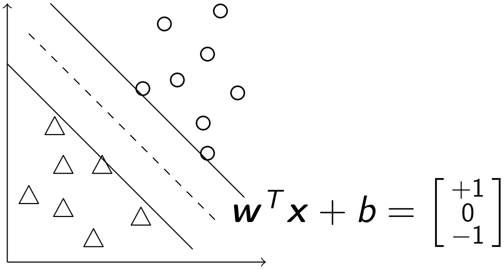
## Linear Classification

- 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





- 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 $-1$:

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

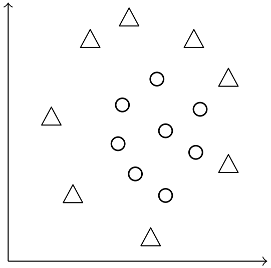
- A quadratic programming problem

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

- 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 $\xi_i$

## Equivalent Optimization Problem (Cont'd)

- We now derive the same optimization problem (1) from a different viewpoint

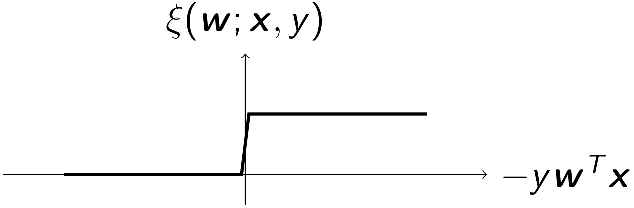
$$\min_{\boldsymbol{w}} \quad (\text{training errors})$$

- To characterize the training error, we need a loss function $\xi(\boldsymbol{w}; \boldsymbol{x}, y)$ for each instance $(\boldsymbol{x}_i, y_i)$
- 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)

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

- Squared hinge loss (l2 loss)

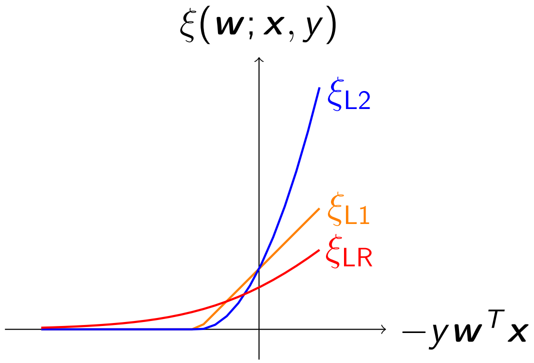
$$\xi_{\mathrm{L2}}(\boldsymbol{w};\boldsymbol{x},y) \equiv \max(0, 1 - y\boldsymbol{w}^T\boldsymbol{x})^2 \tag{3}$$

- Logistic loss

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

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

# Common Loss Functions (Cont'd)



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

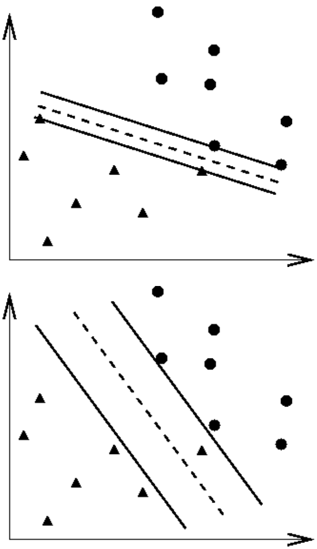
## Common Loss Functions (Cont'd)

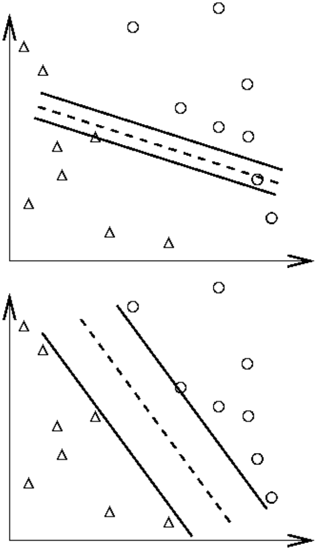
- However, minimizing training losses may not give a good model for future prediction
- Overfitting occurs

## 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

- In training we manipulate the $\boldsymbol{w}$ vector so that it fits the data
- So 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,

  $\xi_{\text{L1}}$: not differentiable

  $\xi_{\text{L2}}$: differentiable but not twice differentiable

  $\xi_{\text{LR}}$: 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

$$\begin{aligned} -\log \prod_{i=1}^{l} p\left(y_i|\boldsymbol{x}_i\right) &= -\sum_{i=1}^{l} \log p\left(y_i|\boldsymbol{x}_i\right) \\ &= \sum_{i=1}^{l} \log\left(1+e^{-y_i \boldsymbol{w}^T \boldsymbol{x}_i}\right) \end{aligned}$$

## 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

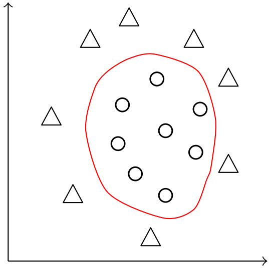
## Outline

2 Kernel classification
- Nonlinear mapping
- Kernel tricks

## 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:

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

- Kernel SVM:

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

## 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
$$\boldsymbol{w} = \sum_{i=1}^{l} y_i \alpha_i \phi(\boldsymbol{x}_i)$$
Details not provided here
- Special $\phi(\boldsymbol{x})$ such that the decision function becomes
$$\begin{aligned} \operatorname{sgn}(\boldsymbol{w}^T \phi(\boldsymbol{x})) &= \operatorname{sgn}\left(\sum\nolimits_{i=1}^{l} y_i \alpha_i \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x})\right) \\ &= \operatorname{sgn}\left(\sum\nolimits_{i=1}^{l} y_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x})\right) \end{aligned}$$

## 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:

$$e^{-\gamma\|\boldsymbol{x}_i-\boldsymbol{x}_j\|^2}, \text{ (Radial Basis Function)}$$

$$(\boldsymbol{x}_i^T\boldsymbol{x}_j/a + b)^d \text{ (Polynomial kernel)}$$

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

$$
\begin{aligned}
& e^{-\gamma\|x_i - x_j\|^2} = e^{-\gamma(x_i - x_j)^2} = e^{-\gamma x_i^2 + 2\gamma x_i x_j - \gamma x_j^2} \\
= & e^{-\gamma x_i^2 - \gamma x_j^2}\left(1 + \frac{2\gamma x_i x_j}{1!} + \frac{(2\gamma x_i x_j)^2}{2!} + \frac{(2\gamma x_i x_j)^3}{3!} + \cdots\right) \\
= & e^{-\gamma x_i^2 - \gamma x_j^2}\left(1 \cdot 1 + \sqrt{\frac{2\gamma}{1!}} x_i \cdot \sqrt{\frac{2\gamma}{1!}} x_j + \sqrt{\frac{(2\gamma)^2}{2!}} x_i^2 \cdot \sqrt{\frac{(2\gamma)^2}{2!}} x_j^2 \right. \\
& \left. + \sqrt{\frac{(2\gamma)^3}{3!}} x_i^3 \cdot \sqrt{\frac{(2\gamma)^3}{3!}} x_j^3 + \cdots\right) = \phi(x_i)^T \phi(x_j),
\end{aligned}
$$

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

3. Linear versus kernel classification
   - Comparison on the cost
   - Numerical comparisons

## Outline

## Linear and Kernel Classification

Now we see that methods such as SVM and logistic regression can 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

$$\begin{array}{ll} \text{accuracy:} & \text{kernel} \geq \text{linear} \\ \text{cost:} & \text{kernel} \gg \text{linear} \end{array}$$

- 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

3. Linear versus kernel classification
   - Comparison on the cost
   - Numerical comparisons

## 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

## Outline

## Outline

4. Solving optimization problems
   - Kernel: decomposition methods
   - Linear: coordinate descent method
   - Linear: second-order methods
   - Experiments

## 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)

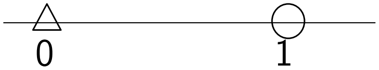
- The dual problem

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

  where $Q_{ij} = y_i y_j \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$ and $\boldsymbol{e} = [1, \ldots, 1]^T$
- From primal-dual relationship, at optimum (8) holds
- Dual problem has a finite number of variables

## Example: Primal-dual Relationship

- Consider the earlier example:



- Now two data are $x_1 = 1, x_2 = 0$ 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

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

## 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

$$\begin{aligned} w &= y_1\alpha_1x_1 + y_2\alpha_2x_2 \\ &= 1 \cdot 2 \cdot 1 + (-1) \cdot 2 \cdot 0 \\ &= 2 \end{aligned}$$

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

## Decision function

- At optimum

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

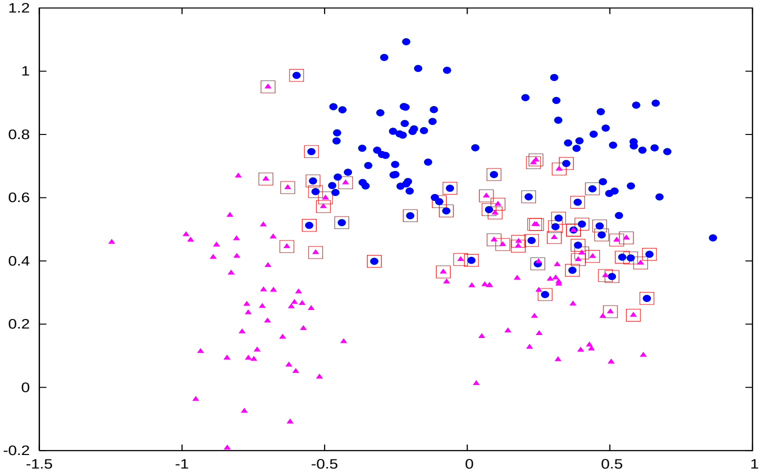
- Decision function

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

- 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

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

- $Q_{ij} \neq 0$, $Q$ : 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
- Sub-problem at the $k$th iteration:

$$\begin{aligned} \min_{\boldsymbol{\alpha}_B} \quad & \frac{1}{2} \begin{bmatrix} \boldsymbol{\alpha}_B^T & (\boldsymbol{\alpha}_N^k)^T \end{bmatrix} \begin{bmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_B \\ \boldsymbol{\alpha}_N^k \end{bmatrix} - \\ & \begin{bmatrix} \mathbf{e}_B^T & (\mathbf{e}_N^k)^T \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_B \\ \boldsymbol{\alpha}_N^k \end{bmatrix} \\ \text{subject to} \quad & 0 \le \alpha_t \le C, t \in B, \ \mathbf{y}_B^T \boldsymbol{\alpha}_B = -\mathbf{y}_N^T \boldsymbol{\alpha}_N^k \end{aligned}$$

## Avoid Memory Problems

- The new objective function

$$\frac{1}{2}\boldsymbol{\alpha}_B^T Q_{BB} \boldsymbol{\alpha}_B + (-\boldsymbol{e}_B + Q_{BN}\boldsymbol{\alpha}_N^k)^T \boldsymbol{\alpha}_B + \text{ constant}$$

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

$$\mathbf{y}_B^T \boldsymbol{\alpha}_B = -\mathbf{y}_N^T \boldsymbol{\alpha}_N^k$$

- 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 $\boldsymbol{\alpha}$

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

  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
- $\#\text{SVs} = 3{,}370 \ll 50{,}000$

  A good case where some remain at zero all the time

## Outline

4. Solving optimization problems
   - Kernel: decomposition methods
   - Linear: coordinate descent method
   - Linear: second-order methods
   - Experiments

## 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

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

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

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

- Without constraints

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

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

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

## The Procedure (Cont'd)

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

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

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

- 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 **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

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

- 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) If $\alpha_i$ can be changed
                $\alpha_i \leftarrow \min(\max(\alpha_i - G/Q_{ii}, 0), C)$
                $\mathbf{u} \leftarrow \mathbf{u} + (\alpha_i - \bar{\alpha}_i) y_i \boldsymbol{x}_i$
```

## Difference from the Kernel Case

- We have seen that coordinate descent is also the main method to train 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 $\nabla f(\boldsymbol{\alpha})$ 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

4 Solving optimization problems
- Kernel: decomposition methods
- Linear: coordinate descent method
- Linear: second-order methods
- Experiments

## 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

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

- 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

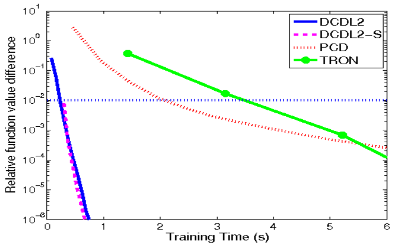
## Outline

## Comparisons

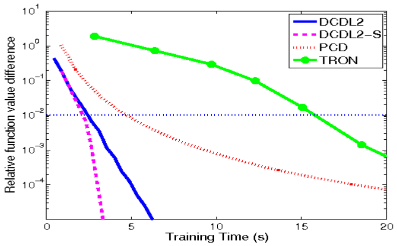
L2-loss SVM is used

- 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)

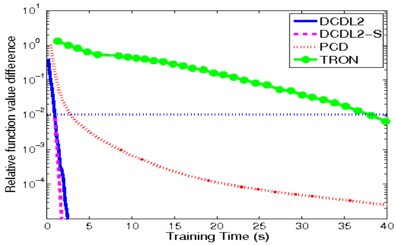
## Objective values (Time in Seconds)



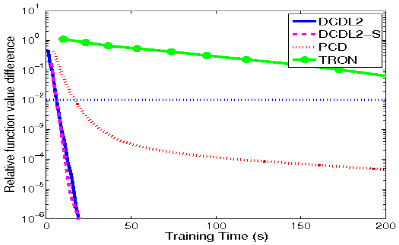
news20



rcv1



yahoo-japan



yahoo-korea

## Analysis

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

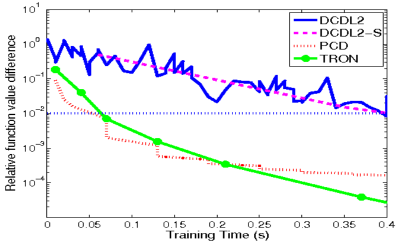
  Useful for document classification
- Half million data in a few seconds
- However, it is less effective if

  # features small: should solve primal; or

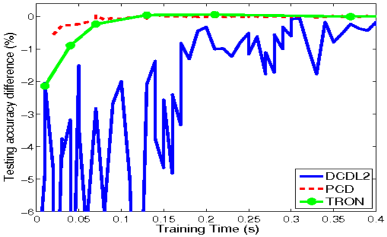
  large penalty parameter $C$; problems are more ill-conditioned

## An Example When # Features Small

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



Objective value



Accuracy

## Outline

## Outline

5. Big-data linear classification
   - Multi-core linear classification
   - Distributed linear classification

## Big-data Linear Classification

- Parallelization in shared-memory system: use the power of multi-core CPU if data can fit in memory
- Distributed linear classification: if data cannot be stored in one computer
- Example: we can parallelize the 2nd-order method (i.e., the Newton method) discussed earlier.
- Recall the bottleneck is the Hessian-vector product

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

See the analysis in the next slide

## Matrix-vector Multiplications

- Two sets:

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

- Matrix-vector multiplications occupy the majority of the running time

| Data set | matrix-vector ratio |
|---|---|
| epsilson | 99.88% |
| webspam | 97.95% |

- This is by Newton methods using one core
- We should parallelize matrix-vector multiplications

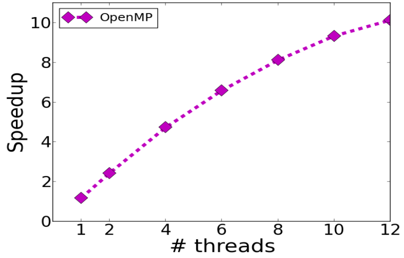
## Outline

## Parallelization by OpenMP

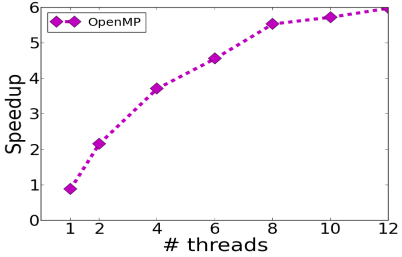
- The Hessian-vector product can be done by

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

- We can easily parallelize this loop by OpenMP
- Speedup; details in Lee et al. (2015)



epsilon

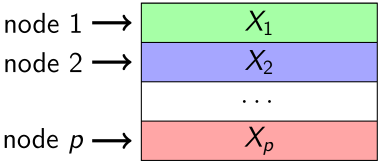


webspam

## Outline

## Parallel Hessian-vector Product

- Now data matrix $X$ is distributedly stored



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

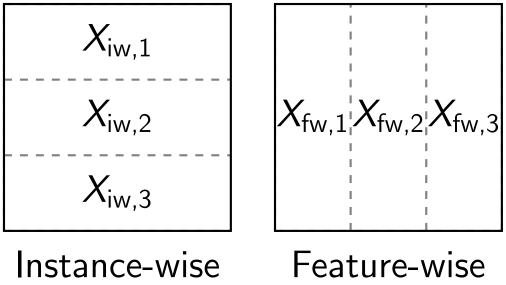
## Parallel Hessian-vector Product (Cont'd)

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

$$\begin{tikzcd}
\mathbf{s} \arrow[r] & X_1^T D_1 X_1 \mathbf{s} \arrow[dr] & & X^T DX\mathbf{s} \\
\mathbf{s} \arrow[r] & X_2^T D_2 X_2 \mathbf{s} \arrow[r] & \boxed{\text{ALL REDUCE}} \arrow[ur] \arrow[r] \arrow[dr] & X^T DX\mathbf{s} \\
\mathbf{s} \arrow[r] & X_3^T D_3 X_3 \mathbf{s} \arrow[ur] & & X^T DX\mathbf{s}
\end{tikzcd}$$

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

## Instance-wise and Feature-wise Data Splits



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

$$(\nabla^2 f(\boldsymbol{w})\boldsymbol{s})_{\mathrm{fw},1} = \boldsymbol{s}_1 + C X_{\mathrm{fw},1}^T D (X_{\mathrm{fw},1}\boldsymbol{s}_1 + \cdots + X_{\mathrm{fw},p}\boldsymbol{s}_p)$$

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

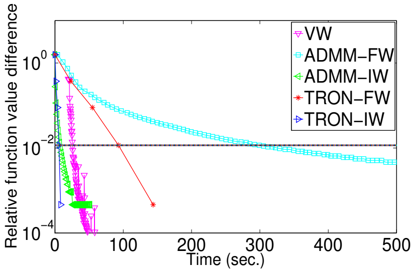
- $X_{\text{fw},1}\boldsymbol{s}_1 + \cdots + X_{\text{fw},p}\boldsymbol{s}_p \in R^l$ must be available on all nodes (by allreduce)
- Data moved per Hessian-vector product

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

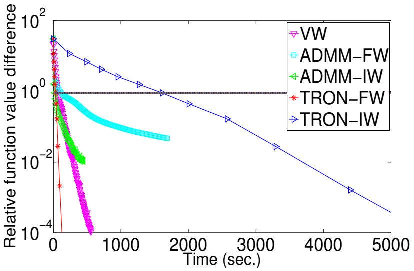
## Experiments

- 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

## 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)

## 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)
- The development is still in an early stage.

## Outline

## Conclusions

- Linear and kernel classification are old topics
- However, novel techniques are still being developed to handle large-scale data or new applications
- You are welcome to join to this interesting research area

## 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`.

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. Technical report, National Taiwan University, 2015.

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`.

## References III

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`.

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.

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.

## References IV

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.