Training Support Vector Machines: Status and Challenges

Chih-Jen Lin
Department of Computer Science
National Taiwan University

Talk at Microsoft Research Asia
October 13, 2009
Outline

- Training support vector machines
  - Training large-scale SVM
  - Linear SVM
  - SVM with Low-Degree Polynomial Mapping
  - Discussion and Conclusions
Support Vector Classification

- **Training** data \((x_i, y_i), i = 1, \ldots, l, x_i \in \mathbb{R}^n, y_i = \pm 1\)
- Maximizing the margin
  
  [Boser et al., 1992, Cortes and Vapnik, 1995]

  \[
  \min_{w, b} \frac{1}{2} w^T w + C \sum_{i=1}^{l} \max(1 - y_i (w^T \phi(x_i) + b), 0)
  \]

- **High dimensional (maybe infinite)** feature space
  \[
  \phi(x) = (\phi_1(x), \phi_2(x), \ldots).
  \]

- **w**: maybe **infinite** variables
Support Vector Classification (Cont’d)

- The **dual** problem (**finite # variables**)

\[
\min_{\alpha} \quad \frac{1}{2} \alpha^T Q \alpha - \mathbf{e}^T \alpha \\
\text{subject to} \quad 0 \leq \alpha_i \leq C, \ i = 1, \ldots, l \\
y^T \alpha = 0,
\]

where \( Q_{ij} = y_i y_j \phi(x_i)^T \phi(x_j) \) and \( \mathbf{e} = [1, \ldots, 1]^T \)

- At optimum

\[
\mathbf{w} = \sum_{i=1}^{l} \alpha_i y_i \phi(x_i)
\]

- Kernel: \( K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j) \); **closed form**

  E.g., RBF kernel: \( e^{-\gamma \|x_i - x_j\|^2} \)
Large Dense Quadratic Programming

- \( Q_{ij} \neq 0 \), \( Q \) : an \( l \) by \( l \) fully dense matrix

\[
\min_{\alpha} \quad \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \\
\text{subject to} \quad 0 \leq \alpha_i \leq C, \ i = 1, \ldots, l \\
y^T \alpha = 0
\]

- 50,000 training points: 50,000 variables:
  \((50,000^2 \times 8/2)\) bytes = 10GB RAM to store \( Q \)

- Traditional optimization methods cannot be directly applied

- Right now most use decomposition methods
Decomposition Methods

- Working on some variables each time (e.g., [Osuna et al., 1997, Joachims, 1998, Platt, 1998])
- Working set $B$, $N = \{1, \ldots, l\} \setminus B$ fixed
- Sub-problem at the $k$th iteration:

$$
\min_{\alpha_B} \frac{1}{2} \begin{bmatrix} \alpha_B^T & (\alpha_k^N)^T \end{bmatrix} \begin{bmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{bmatrix} \begin{bmatrix} \alpha_B \\ \alpha_k^N \end{bmatrix} - \\
\begin{bmatrix} e_B^T & (e_N^k)^T \end{bmatrix} \begin{bmatrix} \alpha_B \\ \alpha_k^N \end{bmatrix}
$$

subject to $0 \leq \alpha_i \leq C$, $i \in B$, $y_B^T \alpha_B = -y_N^T \alpha_k^N$
Avoid Memory Problems

- The new objective function

\[ \frac{1}{2} \alpha_B^T Q_{BB} \alpha_B + (-\mathbf{e}_B + Q_{BN} \alpha_N^k)^T \alpha_B + \text{constant} \]

- Only \( B \) columns of \( Q \) needed (\(|B| \geq 2\))
- Calculated when used
- Trade time for space
- Popular software such as \( SVM^{light} \) and LIBSVM are of this type
- Work well if data not too large (e.g., \( \leq 100k \))
Outline

- Training support vector machines
- **Training large-scale SVM**
- Linear SVM
- SVM with Low-Degree Polynomial Mapping
- Discussion and Conclusions
Is It Possible to Train Large SVM?

- Accurately solve quadratic programs with millions of variables or more?
- General approach: very unlikely
  - Cases with many support vectors: quadratic time bottleneck on $Q_{SV, SV}$
- Parallelization: possible but
  - Difficult in distributed environments due to high communication cost
For large problems, **approximation** almost unavoidable

That is, don’t accurately solve the quadratic program of the full training set
Approximately Training SVM

- Can be done in many aspects
- Data level: sub-sampling
- Optimization level:
  - Approximately solve the quadratic program
- Other non-intuitive but effective ways
  - I will show one today
- Many papers have addressed this issue
Subsampling

- Simple and often effective

Many more advanced techniques

- Incremental training: (e.g., [Syed et al., 1999])
  Data $\Rightarrow$ 10 parts
  train 1st part $\Rightarrow$ SVs, train SVs + 2nd part, ...
- Select and train good points: KNN or heuristics
  e.g., [Bakır et al., 2005]
Approximately Training SVM (Cont’d)

- Approximate the kernel; e.g.,
  [Fine and Scheinberg, 2001, Williams and Seeger, 2001]
- Use part of the kernel; e.g.,
  [Lee and Mangasarian, 2001, Keerthi et al., 2006]
- Early stopping of optimization algorithms
  [Tsang et al., 2005] and most parallel works
- And many others
  Some simple but some sophisticated
Approximately Training SVM (Cont’d)

- But sophisticated techniques may not be always useful
- Sometimes **slower than sub-sampling**
- covtype: 500k training and 80k testing
- rcv1: 550k training and 14k testing

<table>
<thead>
<tr>
<th>Training size</th>
<th>Accuracy</th>
<th>Training size</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>50k</td>
<td>92.5%</td>
<td>50k</td>
<td>97.2%</td>
</tr>
<tr>
<td>100k</td>
<td>95.3%</td>
<td>100k</td>
<td>97.4%</td>
</tr>
<tr>
<td>500k</td>
<td>98.2%</td>
<td>550k</td>
<td>97.8%</td>
</tr>
</tbody>
</table>
But sophisticated techniques may not be always useful

Sometimes **slower than sub-sampling**

covtype: 500k training and 80k testing
rcv1: 550k training and 14k testing

<table>
<thead>
<tr>
<th>Training size</th>
<th>Accuracy</th>
<th>Training size</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>50k</td>
<td>92.5%</td>
<td>50k</td>
<td>97.2%</td>
</tr>
<tr>
<td>100k</td>
<td>95.3%</td>
<td>100k</td>
<td>97.4%</td>
</tr>
<tr>
<td>500k</td>
<td>98.2%</td>
<td>550k</td>
<td>97.8%</td>
</tr>
</tbody>
</table>

Chih-Jen Lin (National Taiwan Univ.)
Personally I prefer specialized approach for large-scale scenarios.

Distribution of training data:

- Large
- Median and small

General Solvers (e.g., LIBSVM, SVM^{light})
We don’t have many large and well labeled sets
They appear in certain application domains
Specific properties of data should be considered
May significantly improve the training speed
We will illustrate this point using linear SVM
The design of software for large and median/small problems should be different
Outline

- Training support vector machines
- Training large-scale SVM
- Linear SVM
- SVM with Low-Degree Polynomial Mapping
- Discussion and Conclusions
Linear SVM

- Data not mapped to another space
- Primal without the bias term $b$

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

- Dual

$$\min_{\alpha} \quad f(\alpha) \equiv \frac{1}{2} \alpha^T Q \alpha - \mathbf{e}^T \alpha$$
subject to $0 \leq \alpha_i \leq C, \forall i$

- $Q_{ij} = y_i y_j \mathbf{x}_i^T \mathbf{x}_j$
Linear SVM (Cont’d)

- In theory, RBF kernel with certain parameters ⇒ as good as linear [Keerthi and Lin, 2003]
- RBF kernel:
  \[ K(x_i, x_j) = e^{-\gamma\|x_i - x_j\|^2} \]
- That is,
  Test accuracy of linear \( \leq \) Test accuracy of RBF
- Linear SVM not better than nonlinear; but
  An approximation to nonlinear SVM
Bag of words model (TF-IDF or others)
A large # of features
Accuracy similar with/without mapping vectors
What if training is much faster?
A very effective approximation to nonlinear SVM
A Comparison: LIBSVM and LIBLINEAR

- *rcv1*: # data: > 600k, # features: > 40k
- TF-IDF
- Using LIBSVM (linear kernel)
  - > 10 hours
- Using LIBLINEAR
  - Computation: < 5 seconds; I/O: 60 seconds
- Same stopping condition
- Accuracy similar to nonlinear; more than 100x speedup
Why Training Linear SVM Is Faster?

- In optimization, each iteration we often need

\[ \nabla_i f(\alpha) = (Q\alpha)_i - 1 \]

- Nonlinear SVM

\[ \nabla_i f(\alpha) = \sum_{j=1}^{l} y_i y_j K(x_i, x_j) \alpha_j - 1 \]

cost: \(O(nl);\ n: \# \text{features}, \ l: \# \text{data}\)

- Linear: use

\[ w \equiv \sum_{j=1}^{l} y_j \alpha_j x_j \text{ and } \nabla_i f(\alpha) = y_i w^T x_i - 1 \]

- Only \(O(n)\) cost if \(w\) is maintained
Faster if \( \not\) iterations not \( l \) times more

For details, see


Experiments

<table>
<thead>
<tr>
<th>Problem</th>
<th>( l ): # data</th>
<th>( n ): # features</th>
</tr>
</thead>
<tbody>
<tr>
<td>news20</td>
<td>19,996</td>
<td>1,355,191</td>
</tr>
<tr>
<td>yahoo-japan</td>
<td>176,203</td>
<td>832,026</td>
</tr>
<tr>
<td>rcv1</td>
<td>677,399</td>
<td>47,236</td>
</tr>
<tr>
<td>yahoo-korea</td>
<td>460,554</td>
<td>3,052,939</td>
</tr>
</tbody>
</table>
Testing Accuracy versus Training Time

news20

rcv1

yahoo-japan

yahoo-korea
Training Linear SVM Always Much Faster?

- No
- If \#data \gg \#features, the algorithm used above may not be very good
- Need some other ways
- But document data are not of this type
- Large-scale SVM training is domain specific
Outline

- Training support vector machines
- Training large-scale SVM
- Linear SVM
- SVM with Low-Degree Polynomial Mapping
- Discussion and Conclusions
Training Nonlinear SVM via Linear SVM

- Revisit nonlinear SVM
  \[
  \min_{\mathbf{w}} \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^{l} \max(1 - y_i \mathbf{w}^T \phi(\mathbf{x}_i), 0)
  \]

- Dimension of \(\phi(\mathbf{x})\): large
- If **not very large**, directly train SVM **without kernel**
- Calculate \(\nabla_i f(\alpha)\) at each step
  - Kernel: \(O(nl)\)
  - Linear SVM: dimension of \(\phi(\mathbf{x})\)
Degree-2 Polynomial Mapping

- Degree-2 polynomial kernel
  \[ K(x_i, x_j) = (1 + x_i^T x_j)^2 \]

- Instead we do
  \[ \phi(x) = [1, \sqrt{2}x_1, \ldots, \sqrt{2}x_n, x_1^2, \ldots, x_2^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{n-1}x_n]^T. \]

- Now we can just consider
  \[ \phi(x) = [1, x_1, \ldots, x_n, x_1^2, \ldots, x_n^2, x_1x_2, \ldots, x_{n-1}x_n]^T . \]

- \( O(n^2) \) dimensions can cause troubles; some considerations are needed
<table>
<thead>
<tr>
<th>Data set</th>
<th>Degree-2 Polynomial Time</th>
<th>Accuracy diff.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LIBLINEAR</td>
<td>LIBSVM</td>
</tr>
<tr>
<td>a9a</td>
<td>1.6</td>
<td>89.8</td>
</tr>
<tr>
<td>real-sim</td>
<td>59.8</td>
<td>1,220.5</td>
</tr>
<tr>
<td>ijcnn1</td>
<td>10.7</td>
<td>64.2</td>
</tr>
<tr>
<td>MNIST38</td>
<td>8.6</td>
<td>18.4</td>
</tr>
<tr>
<td>covtype</td>
<td>5,211.9</td>
<td>≥ 3 × 10^5</td>
</tr>
<tr>
<td>webspam</td>
<td>3,228.1</td>
<td>≥ 3 × 10^5</td>
</tr>
</tbody>
</table>

- Some problems: accuracy similar to RBF; but training much faster
- Less nonlinear SVM to approximate highly nonlinear SVM
In NLP (Natural Language Processing) degree-2 or degree-3 polynomial kernels very popular

- Competitive with RBF; better than linear
- No theory yet; but possible reasons
  - Bigram/trigram useful
- This is different from other areas (e.g., image), which mainly use RBF
- Currently people complain that training is slow
Dependency Parsing

John hit the ball with a bat.

NNP VBD DT NN IN DT NN .

nsubj ROOT det dobj prep det pobj p
Dependency Parsing

John hit the ball with a bat.

<table>
<thead>
<tr>
<th></th>
<th>LIBSVM</th>
<th></th>
<th>LIBLINEAR</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RBF</td>
<td>Poly</td>
<td>Linear</td>
<td>Poly</td>
</tr>
<tr>
<td>Training time</td>
<td>3h34m53s</td>
<td>3h21m51s</td>
<td>3m36s</td>
<td>3m43s</td>
</tr>
<tr>
<td>Parsing speed</td>
<td>0.7x</td>
<td>1x</td>
<td>1652x</td>
<td>103x</td>
</tr>
<tr>
<td>UAS</td>
<td>89.92</td>
<td>91.67</td>
<td>89.11</td>
<td>91.71</td>
</tr>
<tr>
<td>LAS</td>
<td>88.55</td>
<td>90.60</td>
<td>88.07</td>
<td>90.71</td>
</tr>
</tbody>
</table>
Details:

Outline

- Training support vector machines
- Training large-scale SVM
- Linear SVM
- SVM with Low-Degree Polynomial Mapping
- Discussion and Conclusions
What If Data Cannot Fit in Memory?

- We can manage to train data in disk
  Details not shown here
- However, what if data too large to store in one machine?
- So far not many such cases with well labeled data
  It’s expensive to label data
- We do see very large but low quality data
  Dealing with such data is different
L1-regularized Classifiers

- Replacing $\|w\|_2$ with $\|w\|_1$

\[
\min_w \|w\|_1 + C \times (\text{losses})
\]

- Sparsity: many $w$ elements are zeros
  Feature selection

- LIBLINEAR supports L2 loss and logistic regression

\[
\max (0, 1 - y_i w^T x_i)^2 \quad \text{and} \quad \log(1 + e^{-y_i w^T x_i})
\]

- If using least-square loss and $y \in R^l$, related to L1-regularized problems in signal processing
Conclusions

- Training large SVM is difficult
  The (at least) quadratic time bottleneck
- Approximation is often needed; but some are non-intuitive ways
  E.g., linear SVM good approximation to nonlinear SVM for some applications
- Difficult to have a general approach for all large scenarios
  Special techniques are needed
Conclusions (Cont’d)

- Software design for large and median/small problems should be different
  Median/small problems: general and simple software
- Sources for my past work are available on my page. In particular,
  LIBSVM: http://www.csie.ntu.edu.tw/~cjlin/libsvm
  LIBLINEAR: http://www.csie.ntu.edu.tw/~cjlin/liblinear
- I will be happy to talk to any machine learning users here