Training Support Vector Machines: Status and Challenges

Chih-Jen Lin
Department of Computer Science
National Taiwan University

ICML Workshop on Large Scale Learning Challenge
July 9, 2008
Outline

- SVM is popular
  - But its training isn’t easy
- We check existing techniques
- Large data sets
  - We show several approaches, and discuss various considerations
- Will try to partially answer why there are controversial comparisons
Outline

- Introduction to SVM
- Solving SVM Quadratic Programming Problem
- Training large-scale data
- Linear SVM
- 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]

\[
\begin{align*}
\min_{w,b,\xi} & \quad \frac{1}{2}w^T w + C \sum_{i=1}^{l} \xi_i \\
\text{subject to} & \quad y_i (w^T \phi(x_i) + b) \geq 1 - \xi_i, \\
& \quad \xi_i \geq 0, \quad i = 1, \ldots, l.
\end{align*}
\]

- **High dimensional** (maybe infinite) feature space

\[
\phi(x) = (\phi_1(x), \phi_2(x), \ldots).
\]
Support Vector Classification (Cont’d)

- **w**: maybe *infinite* variables
- The *dual* problem (finite ≠ variables)

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

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

- At optimum

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

- Kernel:

\[
K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j)
\]
Outline

- Introduction to SVM
- Solving SVM Quadratic Programming Problem
- Training large-scale data
- Linear SVM
- Discussion and Conclusions
Large Dense Quadratic Programming

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

- \( Q_{ij} \neq 0, Q : \) 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 \( Q \)
- Traditional methods:
  Newton, Quasi Newton cannot be directly applied
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 $B$, $N = \{1, \ldots, l\} \setminus B$ fixed
- Sub-problem at the $k$th iteration:

$$
\min_{\alpha_B} \frac{1}{2} \left[ \alpha_B^T (\alpha_N^k)^T \right] \begin{bmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{bmatrix} \begin{bmatrix} \alpha_B \\ \alpha_N^k \end{bmatrix} - \\
\begin{bmatrix} e_B^T & (e_N^k)^T \end{bmatrix} \begin{bmatrix} \alpha_B \\ \alpha_N^k \end{bmatrix}
$$

subject to $0 \leq \alpha_t \leq C$, $t \in B$, $y_B^T \alpha_B = -y_N^T \alpha_N^k$
Avoid Memory Problems

- The new objective function

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

- Only \( B \) columns of \( Q \) needed (\(|B| \geq 2\))

- Calculated when used

  Trade time for space

- Popular software such as \( SVM^{\text{light}} \), LIBSVM, SVM-Torch are of this type
How Decomposition Methods Perform?

- Convergence not very fast
- But, no need to have very accurate $\alpha$
  Prediction not affected much
- In some situations, $\#$ support vectors $\ll \#$ training points
  Initial $\alpha^1 = 0$, some instances never used
An example of training 50,000 instances using LIBSVM

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

On a Xeon 2.0G machine

Calculating $Q$ may have taken more time

$\#SVs = 3,370 \ll 50,000$
A good case where some remain at zero all the time
Issues of Decomposition Methods

Techniques for faster decomposition methods
- store recently used kernel elements
- working set size/selection
- theoretical issues: convergence
- and others (details not discussed here)

But training large data still difficult
- Kernel square to the number of data
  - Training millions of data time consuming
- Will discuss some possible approaches
Outline

- Introduction to SVM
- Solving SVM Quadratic Programming Problem
- Training large-scale data
- Linear SVM
- Discussion and Conclusions
Parallel: Multi-core/Shared Memory

- Most computation of decomposition methods: kernel evaluations
- Easily parallelized via openMP
- One line change of LIBSVM

Each core/CPU calculates part of a kernel column

<table>
<thead>
<tr>
<th>Multicore</th>
<th>Shared-memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>48</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>8</td>
<td>27</td>
</tr>
</tbody>
</table>

Same 50,000 data (kernel evaluations: 80% time)

- Using GPU [Catanzaro et al., 2008]
Parallel: Distributed Environments

What if data cannot fit into memory?

Use distributed environments

- PSVM: [Chang et al., 2007]
  http://code.google.com/p/psvm/
- $\pi$-SVM: http://pisvm.sourceforge.net,
- Parallel GPDT [Zanni et al., 2006]
- All use MPI
- They report good speed-up
- But on certain environments, communication cost a concern
Approximations

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]
Approximations (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]
- And many others
  Some simple but some sophisticated
Parallelization or Approximation

- Difficult to say
- Parallel: general
- Approximation: simpler in some cases
- We can do both
- For certain problems, approximation doesn’t easily work
Parallelization or Approximation (Cont’d)

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

<table>
<thead>
<tr>
<th>covtype</th>
<th>Training size</th>
<th>Accuracy</th>
<th>rcv1</th>
<th>Training size</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>50k</td>
<td>92.5%</td>
<td></td>
<td>50k</td>
<td>97.2%</td>
</tr>
<tr>
<td></td>
<td>100k</td>
<td>95.3%</td>
<td></td>
<td>100k</td>
<td>97.4%</td>
</tr>
<tr>
<td></td>
<td>500k</td>
<td>98.2%</td>
<td></td>
<td>550k</td>
<td>97.8%</td>
</tr>
</tbody>
</table>

- For large sets, selecting a right approach is essential
- We illustrate this point using linear SVM for document classification
Outline

- Introduction to SVM
- Solving SVM Quadratic Programming Problem
- Training large-scale data
- Linear SVM
- Discussion and Conclusions
Linear Support Vector Machines

- Data not mapped to another space
- In theory, RBF kernel with certain parameters
  ⇒ as good generalization performance as linear
  [Keerthi and Lin, 2003]
- But sometimes can easily solve much larger linear SVMs
- Training of linear/nonlinear SVMs should be separately considered
Linear SVM useful if accuracy similar to nonlinear
Will discuss an example of linear SVM for document classification
Linear SVM for Large Document Sets

Document classification
- Bag of words model (TF-IDF or others)
- A large \# of features
- Can solve larger problems than kernelized cases

Recently an active research topic
- $SVM^{perf}$ [Joachims, 2006]
- Pegasos [Shalev-Shwartz et al., 2007]
- LIBLINEAR [Lin et al., 2007, Hsieh et al., 2008]
- and others
Linear SVM

- Primal without the bias term $b$

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

- Dual

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

- No linear constraint $\mathbf{y}^T \alpha = 0$

- $Q_{ij} = y_i y_j \mathbf{x}_i^T \mathbf{x}_j$
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
Revisit Decomposition Methods

- The extreme: update one variable at a time
- Reduced to

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

where

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

- \( O(nl) \) to calculate \( i \)th row of \( Q \)
  
  \( n: \# \) features, \( l: \# \) data
For linear SVM, define

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

Much easier: $O(n)$

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

All we need is to maintain $\mathbf{w}$. If

$$\bar{\alpha}_i \leftarrow \alpha_i$$

then

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

Still $O(n)$
Testing Accuracy (Time in Seconds)

L1-SVM: news20

L2-SVM: news20

L1-SVM: rcv1

L2-SVM: rcv1
Analysis

- Other implementation details in [Hsieh et al., 2008]
- Decomposition method for linear/nonlinear kernels: $O(nl)$ per iteration
- New way for linear: $O(n)$ per iteration
  Faster if # iterations not $l$ times more
- A few seconds for million data; Any limitation?
- Less effective if
  - # features small: should solve primal
- Large penalty parameter $C$
Analysis (Cont’d)

- One must be careful on comparisons
- Now we have two decomposition methods (nonlinear and linear)
- Similar theoretical convergence rates
- Very different practical behaviors for certain problems
- This partially explains controversial comparisons in some recent work
A lesson: different SVMs
To handle large data \(\Rightarrow\) may need different training strategies

Even just for linear SVM

- \# data \(\gg\) \# features
- \# data \(\ll\) \# features
- \# data, \# features both large

Should use different methods

For example, \# data \(\gg\) \# features

primal based method; (but why not nonlinear?)
Discussion and Conclusions

Outline

- Introduction to SVM
- Solving SVM Quadratic Programming Problem
- Training large-scale data
- Linear SVM
- Discussion and Conclusions
Discussion and Conclusions

- Linear versus nonlinear
  In this competition, most use linear (wild track)
  Even accuracy may be worse
- Recall I mention “parallelization” & “approximation”
  Linear is essentially an approximation of nonlinear
- For large data, selecting a right approach seems to be essential
  But finding a suitable one is difficult
This (i.e., “too many approaches”) is indeed bad from the viewpoint of designing machine learning software

The success of LIBSVM and $SVM^{\text{light}}$

Simple and general

Developments in both directions (general and specific) will help to advance SVM training