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 $(\mathbf{x}_i, y_i), i = 1, \ldots, l, \mathbf{x}_i \in R^n, y_i = \pm 1$
- Maximizing the margin
  [Boser et al., 1992, Cortes and Vapnik, 1995]

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

- High dimensional (maybe infinite) feature space

$$\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots).$$

- $\mathbf{w}$: maybe infinite variables

Support Vector Classification (Cont'd)

- The dual problem (finite # variables)

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

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

- At optimum

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

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

  E.g., RBF kernel: $e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}$

Large Dense Quadratic Programming

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

$$
\begin{array}{ll}
\min\limits_{\boldsymbol{\alpha}} & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \mathbf{e}^T \boldsymbol{\alpha} \\
\text{subject to} & 0 \leq \alpha_i \leq C, i = 1, \ldots, l \\
& \mathbf{y}^T \boldsymbol{\alpha} = 0
\end{array}
$$

- 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, $N = \{1, \ldots, l\} \backslash B$ fixed
- Sub-problem at the kth iteration:

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

Avoid Memory Problems

- The new objective function

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

- 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

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

  $$Q_{\text{SV, SV}}$$

- Parallelization: possible but

  Difficult in distributed environments due to high communication cost

Is It Possible to Train Large SVM? (Cont'd)

- 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

Approximately Training SVM (Cont'd)

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

covtype		rcv1	
Training size	Accuracy	Training size	Accuracy
50k	92.5%	50k	97.2%
100k	95.3%	100k	97.4%
500k	98.2%	550k	97.8%

Approximately Training SVM (Cont'd)

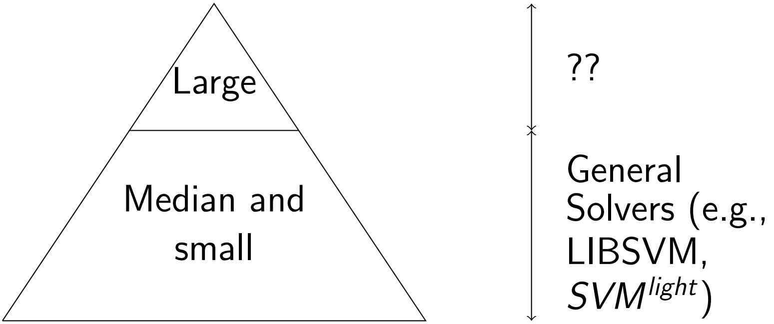
- But sophisticated techniques may not be always useful
- Sometimes slower than sub-sampling
- covtype: 500k training and 80k testing

  rcv1: 550k training and 14k testing

covtype		rcv1	
Training size	Accuracy	Training size	Accuracy
50k	92.5%	50k	97.2%
100k	95.3%	100k	97.4%
500k	98.2%	550k	97.8%

Approximately Training SVM (Cont'd)

- Personally I prefer specialized approach for large-scale scenarios
- Distribution of training data



Approximately Training SVM (Cont'd)

- 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

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

- Dual

$$\begin{array}{ll} \min\limits_{\boldsymbol{\alpha}} & f(\boldsymbol{\alpha}) \equiv \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \mathbf{e}^T\boldsymbol{\alpha} \\ \text{subject to} & 0 \le \alpha_i \le C, \forall i \end{array}$$

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

Linear SVM (Cont'd)

- In theory, RBF kernel with certain parameters
  $\Rightarrow$ as good as linear [Keerthi and Lin, 2003]
  RBF kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2}$$

- That is,

$$\begin{aligned} &\text{Test accuracy of linear} \\ \leq &\text{Test accuracy of RBF} \end{aligned}$$

- Linear SVM not better than nonlinear; but
  An approximation to nonlinear SVM

Linear SVM for Large Document Sets

- 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

$$\nabla_i f(\boldsymbol{\alpha}) = (Q\boldsymbol{\alpha})_i - 1$$

- Nonlinear SVM

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

  cost: $O(nl)$; n: # features, l: # data
- Linear: use

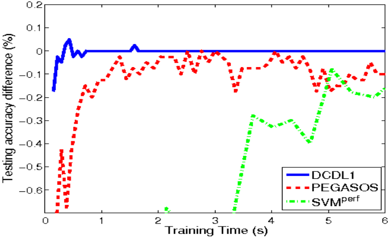
$$\mathbf{w} \equiv \sum_{j=1}^{l} y_j \alpha_j \mathbf{x}_j \text{ and } \nabla_i f(\boldsymbol{\alpha}) = y_i \mathbf{w}^T \mathbf{x}_i - 1$$

- Only $O(n)$ cost if $\mathbf{w}$ is maintained

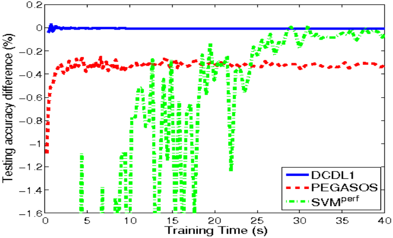
- Faster if # iterations not l times more
- For details, see
  - C.-J. Hsieh K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. *A dual coordinate descent method for large-scale linear SVM*. ICML 2008.
  - R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. *LIBLINEAR: A library for large linear classification*. Journal of Machine Learning Research 9(2008), 1871-1874.
- Experiments

Problem	l: # data	n: # features
news20	19,996	1,355,191
yahoo-japan	176,203	832,026
rcv1	677,399	47,236
yahoo-korea	460,554	3,052,939

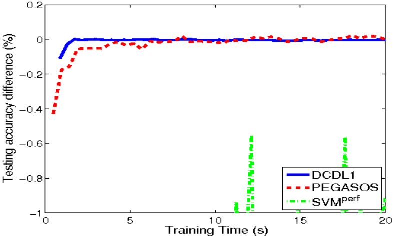
Testing Accuracy versus Training Time



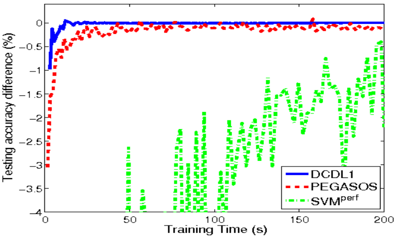
news20



yahoo-japan



rcv1



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

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

- Dimension of $\phi(\mathbf{x})$: large
- If not very large, directly train SVM without kernel
- Calculate $\nabla_i f(\boldsymbol{\alpha})$ at each step

  Kernel: $O(nl)$

  Linear SVM: dimension of $\phi(\mathbf{x})$

Degree-2 Polynomial Mapping

- Degree-2 polynomial kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$$

- Instead we do

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

- Now we can just consider

$$\phi(\mathbf{x}) = [1, x_1, \ldots, x_n, x_1^2, \ldots, x_n^2, x_1x_2, \ldots, x_{n-1}x_n]^T.$$

- $O(n^2)$ dimensions can cause troubles; some considerations are needed

Accuracy Difference with linear and RBF

Data set	Degree-2 Polynomial Time		Accuracy diff.	
	LIBLINEAR	LIBSVM	linear	RBF
a9a	1.6	89.8	0.07	0.02
real-sim	59.8	1,220.5	0.49	0.10
ijcnn1	10.7	64.2	5.63	-0.85
MNIST38	8.6	18.4	2.47	-0.40
covtype	5,211.9	$\geq 3 \times 10^5$	3.74	-15.98
webspam	3,228.1	$\geq 3 \times 10^5$	5.29	-0.76

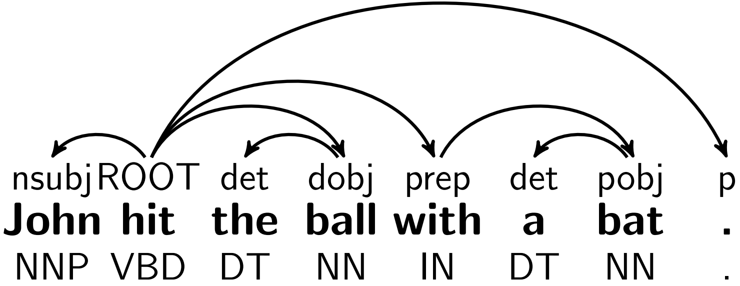
- Some problems: accuracy similar to RBF; but training much faster
- Less nonlinear SVM to approximate highly nonlinear SVM

NLP Applications

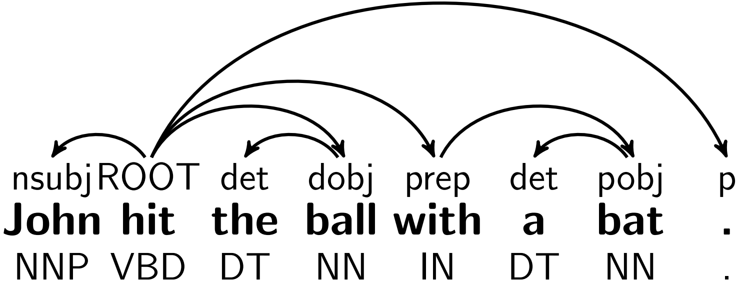
- In NLP (Natural Language Processing) degree-2 or degree-3 polynomial kernels very popular
- Competitive with RBF; better than linear
- No theory yet; but possible reasons

  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



Dependency Parsing



	LIBSVM		LIBLINEAR	
	RBF	Poly	Linear	Poly
Training time	3h34m53s	3h21m51s	3m36s	3m43s
Parsing speed	0.7x	1x	1652x	103x
UAS	89.92	91.67	89.11	91.71
LAS	88.55	90.60	88.07	90.71

Dependency Parsing (Cont'd)

Details:

- Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin. Low-degree polynomial mapping of data for SVM, 2009.

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 $\|\mathbf{w}\|_2$ with $\|\mathbf{w}\|_1$

$$\min_{\mathbf{w}} \|\mathbf{w}\|_1 + C \times \text{(losses)}$$

- Sparsity: many $\mathbf{w}$ elements are zeros
  Feature selection
- LIBLINEAR supports L2 loss and logistic regression

$$\max\left(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\right)^2 \text{ and } \log(1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i})$$

- If using least-square loss and $\mathbf{y} \in R^l$,
  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