Recent Advances in Large Linear Classification

Chih-Jen Lin
Department of Computer Science
National Taiwan University

Talk at NEC Labs, August 26, 2011
This talk is based on our recent survey paper invited by *Proceedings of IEEE*
G.-X. Yuan, C.-H. Ho, and C.-J. Lin. Recent Advances of Large-scale Linear Classification.

It’s also related to our development of the software LIBLINEAR

www.csie.ntu.edu.tw/~cjlin/liblinear

Due to time constraints, we will give overviews instead of deep technical details.
Outline

- Introduction
- Binary linear classification
- Multi-class linear classification
- Applications in non-standard scenarios
- Data beyond memory capacity
- Discussion and conclusions
Outline

- Introduction
- Binary linear classification
- Multi-class linear classification
- Applications in non-standard scenarios
- Data beyond memory capacity
- Discussion and conclusions
Linear and Nonlinear Classification

By linear we mean data not mapped to a higher dimensional space

Original: [height, weight]
Nonlinear: [height, weight, \(\text{weight/height}^2\)]
Linear and Nonlinear Classification (Cont’d)

- Given training data \( \{y_i, x_i\}, x_i \in \mathbb{R}^n, i = 1, \ldots, l, \) \( y_i = \pm 1 \)
  
  \( l: \# \) of data, \( n: \# \) of features

- Linear: find \((w, b)\) such that the decision function is
  
  \[
  \text{sgn} \left( w^T x + b \right)
  \]

- Nonlinear: map data to \( \phi(x_i) \). The decision function becomes
  
  \[
  \text{sgn} \left( w^T \phi(x) + b \right)
  \]

- Later \( b \) is omitted
Why Linear Classification?

- If $\phi(x)$ is high dimensional, $w^T \phi(x)$ is expensive
- Kernel methods:

\[ w \equiv \sum_{i=1}^{l} \alpha_i \phi(x_i) \text{ for some } \alpha, K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j) \]

New decision function: \( \text{sgn} \left( \sum_{i=1}^{l} \alpha_i K(x_i, x) \right) \)

- Special $\phi(x)$ so that calculating $K(x_i, x_j)$ is easy
- Example:

\[ K(x_i, x_j) \equiv (x_i^T x_j + 1)^2 = \phi(x_i)^T \phi(x_j), \phi(x) \in \mathbb{R}^{O(n^2)} \]
Why Linear Classification? (Cont’d)

- Prediction
  \[ w^T x \text{ versus } \sum_{i=1}^{l} \alpha_i K(x_i, x) \]

- If \( K(x_i, x_j) \) takes \( O(n) \), then
  \[ O(n) \text{ versus } O(nl) \]

- Nonlinear: more powerful to separate data
  Linear: cheaper and simpler
Linear is Useful in Some Places

- For certain problems, **accuracy** by linear is as good as nonlinear
  - But **training and testing are much faster**
- Especially document classification
  - Number of features (bag-of-words model) very large
- Recently linear classification is a popular research topic. Sample works in 2005-2008: Joachims (2006); Shalev-Shwartz et al. (2007); Hsieh et al. (2008)
  - They focus on large **sparse** data
- There are **many** other recent papers and software
## Comparison Between Linear and Nonlinear (Training Time & Testing Accuracy)

<table>
<thead>
<tr>
<th>Data set</th>
<th>Linear</th>
<th>RBF Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Accuracy</td>
</tr>
<tr>
<td>MNIST38</td>
<td>0.1</td>
<td>96.82</td>
</tr>
<tr>
<td>ijcnn1</td>
<td>1.6</td>
<td>91.81</td>
</tr>
<tr>
<td>covtype</td>
<td>1.4</td>
<td>76.37</td>
</tr>
<tr>
<td>news20</td>
<td>1.1</td>
<td>96.95</td>
</tr>
<tr>
<td>real-sim</td>
<td>0.3</td>
<td>97.44</td>
</tr>
<tr>
<td>yahoo-japan</td>
<td>3.1</td>
<td>92.63</td>
</tr>
<tr>
<td>webspam</td>
<td>25.7</td>
<td>93.35</td>
</tr>
</tbody>
</table>

Size reasonably large: e.g., yahoo-japan: 140k instances and 830k features
## Comparison Between Linear and Nonlinear (Training Time & Testing Accuracy)

<table>
<thead>
<tr>
<th>Data set</th>
<th>Linear</th>
<th>RBF Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Accuracy</td>
</tr>
<tr>
<td>MNIST38</td>
<td>0.1</td>
<td>96.82</td>
</tr>
<tr>
<td>ijcnn1</td>
<td>1.6</td>
<td>91.81</td>
</tr>
<tr>
<td>covtype</td>
<td>1.4</td>
<td>76.37</td>
</tr>
<tr>
<td>news20</td>
<td>1.1</td>
<td>96.95</td>
</tr>
<tr>
<td>real-sim</td>
<td>0.3</td>
<td>97.44</td>
</tr>
<tr>
<td>yahoo-japan</td>
<td>3.1</td>
<td>92.63</td>
</tr>
<tr>
<td>webspam</td>
<td>25.7</td>
<td>93.35</td>
</tr>
</tbody>
</table>

Size reasonably large: e.g., yahoo-japan: 140k instances and 830k features
Comparison Between Linear and Nonlinear (Training Time & Testing Accuracy)

<table>
<thead>
<tr>
<th>Data set</th>
<th>Linear</th>
<th>RBF Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Accuracy</td>
</tr>
<tr>
<td>MNIST38</td>
<td>0.1</td>
<td>96.82</td>
</tr>
<tr>
<td>ijcnn1</td>
<td>1.6</td>
<td>91.81</td>
</tr>
<tr>
<td>covtype</td>
<td>1.4</td>
<td>76.37</td>
</tr>
<tr>
<td>news20</td>
<td>1.1</td>
<td>96.95</td>
</tr>
<tr>
<td>real-sim</td>
<td>0.3</td>
<td>97.44</td>
</tr>
<tr>
<td>yahoo-japan</td>
<td>3.1</td>
<td>92.63</td>
</tr>
<tr>
<td>webspam</td>
<td>25.7</td>
<td>93.35</td>
</tr>
</tbody>
</table>

Size reasonably large: e.g., yahoo-japan: 140k instances and 830k features
Outline

- Introduction
- Binary linear classification
- Multi-class linear classification
- Applications in non-standard scenarios
- Data beyond memory capacity
- Discussion and conclusions
Binary Linear Classification

- Training data \( \{y_i, x_i\}, x_i \in \mathbb{R}^n, i = 1, \ldots, l, y_i = \pm 1 \)
- \( l \): # of data, \( n \): # of features

\[
\min_w r(w) + C \sum_{i=1}^{l} \xi(w; x_i, y_i)
\]

- \( r(w) \): regularization term
- \( \xi(w; x, y) \): loss function: we hope \( y \mathbf{w}^T \mathbf{x} > 0 \)
- \( C \): regularization parameter
Loss Functions

- Some commonly used ones:
  \[
  \xi_{L1}(w; x, y) \equiv \max(0, 1 - y w^T x), \tag{1}
  \]
  \[
  \xi_{L2}(w; x, y) \equiv \max(0, 1 - y w^T x)^2, \quad \text{and} \tag{2}
  \]
  \[
  \xi_{LR}(w; x, y) \equiv \log(1 + e^{-y w^T x}). \tag{3}
  \]

- SVM (Boser et al., 1992; Cortes and Vapnik, 1995): (1)-(2)
- Logistic regression (LR): (3)
They are similar
Regularization

- L1 versus L2

\[ \|w\|_1 \text{ and } w^T w / 2 \]

- \( w^T w / 2 \): smooth, easier to optimize
- \( \|w\|_1 \): non-differentiable
  - sparse solution; possibly many zero elements

Possible advantages of L1 regularization:

- Feature selection
- Less storage for \( w \)
Training Linear Classifiers

- Many recent developments; won’t show details here
- Why training linear is faster than nonlinear?
- Recall the $O(n)$ and $O(nl)$ difference in prediction:

  \[ \mathbf{w}^T \mathbf{x} \quad \text{and} \quad \sum_{i=1}^{l} \alpha_i K(\mathbf{x}_i, \mathbf{x}) \]

  $n$: # features, $l$: # data

- A similar situation happens here. During training:

  \[ \sum_{t=1}^{l} \alpha_t \mathbf{x}_i^T \mathbf{x}_t \quad \text{often needed} \Rightarrow O(nl) \quad (4) \]
Training Linear Classifiers (Cont’d)

- By maintaining

\[ u \equiv \sum_{t=1}^{l} y_t \alpha_t x_t \rightarrow u^T x_i \quad O(n) \text{ cost} \]

- \( u \): an intermediate variable during training; eventually approaches the final weight vector \( w \)

- Key: we are able to store \( x_t, \forall t \) and maintain \( u \)

Nonlinear: can’t store \( \phi(x_t) \)

- For linear, basically any optimization method can be applied
Choosing a Training Algorithm

- Data property
  - \# instances \ll \# features or the other way around
- Primal or dual
- First-order or higher-order
  - Now first-order is slightly preferred as seldom we need an accurate optimization solution
- Cost of operations
  - exp/log more expensive; avoid them in training LR
- Others
L1 Regularization

- **Non-differentiable**: need non-smooth optimization techniques
- Difficult to apply sophisticated methods
- Currently, coordinate descent or Newton with coordinate descent are among the most efficient (Yuan et al., 2010; Friedman et al., 2010; Yuan et al., 2011)
Outline

- Introduction
- Binary linear classification
- Multi-class linear classification
- Applications in non-standard scenarios
- Data beyond memory capacity
- Discussion and conclusions
Solving Several Binary Problems

- Same methods for linear and nonlinear classification
  - But there are some subtle differences
- One-vs-rest
  
  \( w_m \): class \( m \) positive; others negative
  
  \[ \text{class of } x \equiv \arg \max_{m=1,...,k} w_m^T x. \]

  Memory: \( O(kn) \); \( k \): \# classes
- One-vs-one: \( w_{1,2}, \ldots, w_{(k-1),k} \) constructed
  
  \( O(k^2n) \) memory cost
So one-vs-rest more suitable than one-vs-one
This \textit{isn’t} the case for kernelized SVM/LR
Considering All Data at Once

$$\min_{w_1, \ldots, w_k} \frac{1}{2} \sum_{m=1}^{k} \|w_m\|_2^2 + C \sum_{i=1}^{l} \xi(\{w_m\}_{m=1}^{k}; x_i, y_i),$$

Multi-class SVM by Crammer and Singer (2001)

loss function: $$\max_{m \neq y} \max_{m \neq y} (0, 1 - (w_y - w_m)^T x).$$

Maximum Entropy (ME)

loss function: $$P(y|x) \equiv \frac{\exp(w_y^T x)}{\sum_{m=1}^{k} \exp(w_m^T x)}.$$

Many don’t think that ME is close to SVM; but it is.

Note if \# classes = 2, ME ⇒ LR
Outline

- Introduction
- Binary linear classification
- Multi-class linear classification
- Applications in non-standard scenarios
- Data beyond memory capacity
- Discussion and conclusions
Linear classification can be applied to many other places.

An important one is to approximate nonlinear classifiers.

Goal: better accuracy of nonlinear but faster training/testing.

Two types of methods here:
- Linear-method for explicit data mappings
- Approximating kernels
Linear Methods to Explicitly Train $\phi(x_i)$

- Example: low-degree polynomial mapping:

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

- For this mapping, $\#$ features $= O(n^2)$

- When is it useful?
  Recall $O(n)$ for linear versus $O(nl)$ for kernel
  Now $O(n^2)$ versus $O(nl)$

- Sparse data
  $n \Rightarrow \bar{n}$, average $\#$ non-zeros for sparse data
  $\bar{n} \ll n \Rightarrow O(\bar{n}^2)$ may still be smaller than $O(l\bar{n})$

Chih-Jen Lin (National Taiwan Univ.)
Applications in non-standard scenarios

High Dimensionality of $\phi(x)$ and $w$

- Many **new** considerations in large scenarios
- For example, $w$ has $O(n^2)$ components if degree is 2
  
  Our application: $n = 46, 155, 20G$ for $w$

- See detailed discussion in Chang et al. (2010)

- A related development is the COFFIN framework by Sonnenburg and Franc (2010)
Applications in non-standard scenarios

An NLP Application: Dependency Parsing

Construct dependency graph: a multi-class problem

Very sparse: \( \bar{n} \), average \# nonzeros per instance

<table>
<thead>
<tr>
<th>( n )</th>
<th>Dim. of ( \phi(x) )</th>
<th>( l )</th>
<th>( \bar{n} )</th>
<th>w’s # nonzeros</th>
</tr>
</thead>
<tbody>
<tr>
<td>46,155</td>
<td>1,065,165,090</td>
<td>204,582</td>
<td>13.3</td>
<td>1,438,456</td>
</tr>
</tbody>
</table>
Applications in non-standard scenarios

An NLP Application (Cont’d)

<table>
<thead>
<tr>
<th></th>
<th>LIBSVM</th>
<th>LIBLINEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RBF</td>
<td>Poly</td>
</tr>
<tr>
<td>Training time</td>
<td>3h34m53s</td>
<td>3h21m51s</td>
</tr>
<tr>
<td>Parsing speed</td>
<td>0.7x</td>
<td>1x</td>
</tr>
<tr>
<td>UAS</td>
<td>89.92</td>
<td>91.67</td>
</tr>
<tr>
<td>LAS</td>
<td>88.55</td>
<td>90.60</td>
</tr>
</tbody>
</table>

- Explicitly using $\phi(x)$ instead of kernels
  $\Rightarrow$ faster training and testing
- Some interesting **Hashing** techniques used to handle sparse $w$
Approximating Kernels

Following Lee and Wright (2010), we consider two categories

Kernel matrix approximation:

- Original matrix $Q$ with
  \[ Q_{ij} = y_i y_j K(x_i, x_j) \]

- Consider
  \[ \bar{Q} = \bar{\Phi}^T \bar{\Phi} \approx Q. \]

- $\bar{\Phi} \equiv [\bar{x}_1, \ldots, \bar{x}_l]$ becomes new training data \Rightarrow trained by a linear classifier
Approximating Kernels (Cont’d)

- $\Phi \in \mathbb{R}^{d \times l}$, $d \ll l$. $\#$ features $\ll \#$ data
- Testing is an issue

Feature mapping approximation
- A mapping function $\phi: \mathbb{R}^n \rightarrow \mathbb{R}^d$ such that

  $$\phi(x)^T \phi(t) \approx K(x, t).$$

- Testing is straightforward because $\phi(\cdot)$ is available
- Many mappings have been proposed; in particular, Hashing
- $\phi(\cdot)$ may be dense or sparse
Outline

- Introduction
- Binary linear classification
- Multi-class linear classification
- Applications in non-standard scenarios
- Data beyond memory capacity
- Discussion and conclusions
Data Beyond Memory Capacity

- Most existing algorithms assume data in memory
- They are slow if data larger than memory
  Frequent disk access of data; CPU time no longer the main concern
- They cannot be run in distributed environments
- Many challenging research issues
When Data Cannot Fit In Memory

LIBLINEAR on machine with 1 GB memory:

Disk swap causes lengthy training time
Disk-level Data Classification

- Data larger than memory but smaller than disk
- Design algorithms so that disk access is less frequent
- An example (Yu et al., 2010): a decomposition method to load a block at a time but ensure overall convergence
- But loading time becomes a big concern
  Reading 1TB from a hard disk takes very long time
Distributed Linear Classification

- An important advantage: each node loads data in its disk
- Parallel data loading, but how about operations?

Issues
- Many methods (e.g., stochastic gradient descent or coordinate descent) are inherently sequential
- Communication cost is a concern
Distributed Linear Classification (Cont’d)

Simple approaches

- **Subsampling**: a subset to fit in memory
  - Simple and useful in some situations
  - In a sense, you do a “reduce” operation to collect data to one computer, and then conduct detailed analysis

- **Bagging**: train several subsets and ensemble results
  - Useful in distributed environments; each node ⇒ a subset
  - Example: Zinkevich et al. (2010)
Some results by averaging models

<table>
<thead>
<tr>
<th></th>
<th>yahoo-korea</th>
<th>kddcup10</th>
<th>webspam</th>
<th>epsilon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using all</td>
<td>87.29</td>
<td>89.89</td>
<td>99.51</td>
<td>89.78</td>
</tr>
<tr>
<td>Avg. models</td>
<td>86.08</td>
<td>89.64</td>
<td>98.40</td>
<td>88.83</td>
</tr>
</tbody>
</table>

- Using all: solves a single linear SVM
- Avg. models: each node solves a linear SVM on a subset
- Slightly worse but in general OK
Parallel optimization
Many possible approaches
If the method involves matrix-vector products, then such operations can be paralleled
Each iteration involves communication
Also MapReduce not very suitable for iterative algorithms (I/O for fault tolerance)
Should have as few iterations as possible
Distributed Linear Classification (Cont’d)

ADMM (Boyd et al., 2011)

\[
\begin{align*}
\min_{w_1, \ldots, w_m, z} & \quad \frac{1}{2} z^T z + C \sum_{j=1}^{m} \sum_{i \in B_j} \xi_{L1}(w; x_i, y_i) + \frac{\rho}{2} \sum_{j=1}^{m} \|w_j - z\|^2 \\
\text{subject to} & \quad w_j - z = 0, \forall j
\end{align*}
\]

- Each problem independently updated; but must collect \(w_j\)
- Some have tried MapReduce, but no public implementation yet
- Convergence may not be very fast (i.e., need some iterations)
Distributed Linear Classification (Cont’d)

Vowpal Wabbit (Langford et al., 2007)
- After version 6.0, Hadoop support has been provided
- LBFGS (quasi Newton) algorithms
- From John’s talk: 2.1T features, 17B samples, 1K nodes ⇒ 70 minutes
Outline

- Introduction
- Binary linear classification
- Multi-class linear classification
- Applications in non-standard scenarios
- Data beyond memory capacity
- Discussion and conclusions
Related Topics

Structured learning
- Instead of $y_i \in \{+1, -1\}$, $y_i$ becomes a **vector**
- Examples: condition random fields (CRF) and structured SVM
- They are **linear classifiers**

Regression
- Document classification has been widely used, but document regression (e.g., L2-regularized SVR) less frequently applied
- Example: $y_i$ is CTR and $x_i$ is a web page
- L1-regularized least-square regression is another story $\Rightarrow$ very popular for compressed sensing
Conclusions

- Linear classification is an old topic; **but new developments for large-scale applications are interesting**

- Linear classification works on $\mathbf{x}$ rather than $\phi(\mathbf{x})$
  - Easy and flexible for **feature engineering**
  - Linear classification + feature engineering useful for many real applications