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.

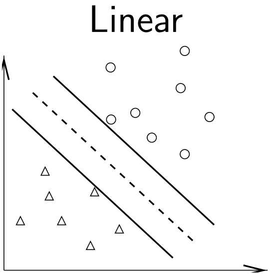
Outline

- 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, $\text{weight}/\text{height}^2$]

Linear and Nonlinear Classification (Cont'd)

- Given training data $\{y_i, \mathbf{x}_i\}, \mathbf{x}_i \in R^n, i = 1, \ldots, l,$
  $y_i = \pm 1$
  l: # of data, n: # of features
- Linear: find $(\mathbf{w}, b)$ such that the decision function is

$$\text{sgn}\left(\mathbf{w}^T \mathbf{x} + b\right)$$

- Nonlinear: map data to $\phi(\mathbf{x}_i)$. The decision function becomes

$$\text{sgn}\left(\mathbf{w}^T \phi(\mathbf{x}) + b\right)$$

- Later b is omitted

Why Linear Classification?

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

$$\mathbf{w} \equiv \sum_{i=1}^{l} \alpha_i \phi(\mathbf{x}_i) \text{ for some } \boldsymbol{\alpha}, K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

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

- Special $\phi(\mathbf{x})$ so that calculating $K(\mathbf{x}_i, \mathbf{x}_j)$ is easy
- Example:

$$K(\mathbf{x}_i, \mathbf{x}_j) \equiv (\mathbf{x}_i^T \mathbf{x}_j + 1)^2 = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j), \phi(\mathbf{x}) \in R^{O(n^2)}$$

Why Linear Classification? (Cont'd)

- Prediction

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

- If $K(\mathbf{x}_i, \mathbf{x}_j)$ takes $O(n)$, then

$$O(n) \quad \text{versus} \quad O(nl)$$

- 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)

	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 Nonlinear (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 Nonlinear (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

- 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 $\{y_i, \mathbf{x}_i\}, \mathbf{x}_i \in R^n, i = 1, \ldots, l, \ y_i = \pm 1$
- l: # of data, n: # of features

$$\min_{\mathbf{w}} \quad r(\mathbf{w}) + C \sum_{i=1}^{l} \xi(\mathbf{w}; \mathbf{x}_i, y_i)$$

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

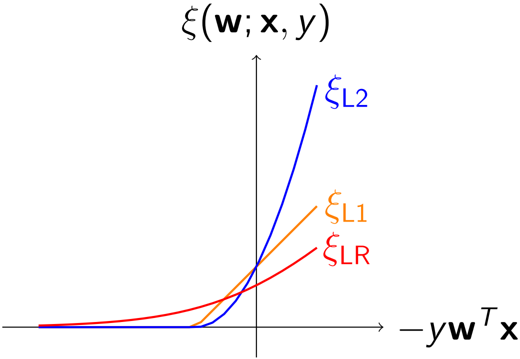
Loss Functions

- Some commonly used ones:

$$\xi_{\mathrm{L1}}(\mathbf{w};\mathbf{x},y) \equiv \max(0, 1 - y\mathbf{w}^T\mathbf{x}), \tag{1}$$
$$\xi_{\mathrm{L2}}(\mathbf{w};\mathbf{x},y) \equiv \max(0, 1 - y\mathbf{w}^T\mathbf{x})^2, \quad \text{and} \tag{2}$$
$$\xi_{\mathrm{LR}}(\mathbf{w};\mathbf{x},y) \equiv \log(1 + e^{-y\mathbf{w}^T\mathbf{x}}). \tag{3}$$

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

Loss Functions (Cont'd)



They are similar

Regularization

- L1 versus L2

$$\|\mathbf{w}\|_1 \text{ and } \mathbf{w}^T\mathbf{w}/2$$

- $\mathbf{w}^T\mathbf{w}/2$: smooth, easier to optimize
- $\|\mathbf{w}\|_1$: non-differentiable

  sparse solution; possibly many zero elements
- Possible advantages of L1 regularization:

  Feature selection

  Less storage for $\mathbf{w}$

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:

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

  n: # features, l: # data
- A similar situation happens here. During training:

$$\sum_{t=1}^{l} \alpha_t \mathbf{x}_i^T \mathbf{x}_t \text{ often needed } \Rightarrow O(nl) \tag{4}$$

Training Linear Classifiers (Cont'd)

- By maintaining

$$\mathbf{u} \equiv \sum_{t=1}^{l} y_t \alpha_t \mathbf{x}_t \quad \rightarrow \quad \mathbf{u}^T \mathbf{x}_i \qquad O(n) \text{ cost}$$

- $\mathbf{u}$: an intermediate variable during training; eventually approaches the final weight vector $\mathbf{w}$
- Key: we are able to store $\mathbf{x}_t, \forall t$ and maintain $\mathbf{u}$
  Nonlinear: can't store $\phi(\mathbf{x}_t)$
- 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

$$\mathbf{w}_m: \text{ class } m \text{ positive; others negative}$$
$$\text{class of } \mathbf{x} \equiv \arg \max_{m=1,\ldots,k} \mathbf{w}_m^T \mathbf{x}.$$

  Memory: $O(kn)$; k: # classes
- One-vs-one: $\mathbf{w}_{1,2}, \ldots, \mathbf{w}_{(k-1),k}$ constructed

$$O(k^2 n) \text{ memory cost}$$

Solving Several Binary Problems (Cont'd)

- So one-vs-rest more suitable than one-vs-one
- This isn't the case for kernelized SVM/LR

Considering All Data at Once

$$\min_{\mathbf{w}_1,\ldots,\mathbf{w}_k} \quad \frac{1}{2}\sum\nolimits_{m=1}^{k} \|\mathbf{w}_m\|_2^2 + C\sum\nolimits_{i=1}^{l} \xi(\{\mathbf{w}_m\}_{m=1}^{k}; \mathbf{x}_i, y_i),$$

Multi-class SVM by Crammer and Singer (2001)

$$\text{loss function}: \quad \max_{m\neq y} \max(0, 1-(\mathbf{w}_y-\mathbf{w}_m)^T\mathbf{x}).$$

Maximum Entropy (ME)

$$\text{loss function}: \quad P(y|\mathbf{x}) \equiv \frac{\exp(\mathbf{w}_y^T\mathbf{x})}{\sum_{m=1}^{k}\exp(\mathbf{w}_m^T\mathbf{x})},$$

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

Note if $\#$ classes $= 2$, ME $\Rightarrow$ LR

Outline

Applications in Non-standard Scenarios

- 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 $\phi(\mathbf{x}_i)$

- Example: low-degree polynomial mapping:

$$\phi(\mathbf{x}) = [1, x_1, \ldots, x_n, x_1^2, \ldots, x_n^2, x_1 x_2, \ldots, x_{n-1}x_n]^T$$

- 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

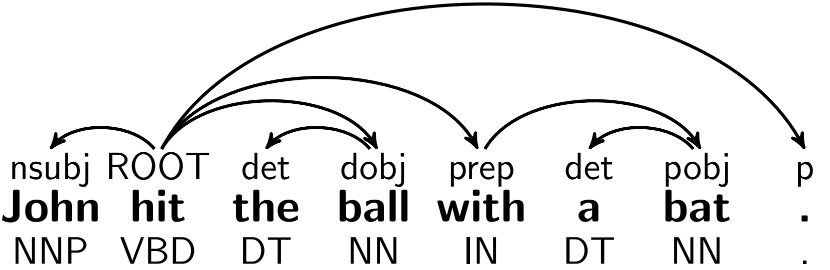
  $\bar{n} \ll n \Rightarrow O(\bar{n}^2)$ may still be smaller than $O(l\bar{n})$

High Dimensionality of $\phi(\mathbf{x})$ and $\mathbf{w}$

- Many new considerations in large scenarios
- For example, $\mathbf{w}$ has $O(n^2)$ components if degree is 2
  Our application: $n = 46,155$, 20G for $\mathbf{w}$
- See detailed discussion in Chang et al. (2010)
- A related development is the COFFIN framework by Sonnenburg and Franc (2010)

An NLP Application: Dependency Parsing

Construct dependency graph: a multi-class problem



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

n	Dim. of $\phi(\mathbf{x})$	l	$\bar{n}$	$\mathbf{w}$'s # nonzeros
46,155	1,065,165,090	204,582	13.3	1,438,456

An NLP Application (Cont'd)

	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

- Explicitly using $\phi(\mathbf{x})$ 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

$$Q_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

- Consider

$$\bar{Q} = \bar{\Phi}^T \bar{\Phi} \approx Q.$$

- $\bar{\Phi} \equiv [\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_l]$ becomes new training data $\Rightarrow$ trained by a linear classifier

Approximating Kernels (Cont'd)

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

Feature mapping approximation

- A mapping function $\bar{\phi} : \mathbf{R}^n \rightarrow \mathbf{R}^d$ such that

$$\bar{\phi}(\mathbf{x})^T \bar{\phi}(\mathbf{t}) \approx K(\mathbf{x}, \mathbf{t}).$$

- Testing is straightforward because $\bar{\phi}(\cdot)$ is available
- Many mappings have been proposed; in particular, Hashing
- $\bar{\phi}(\cdot)$ 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

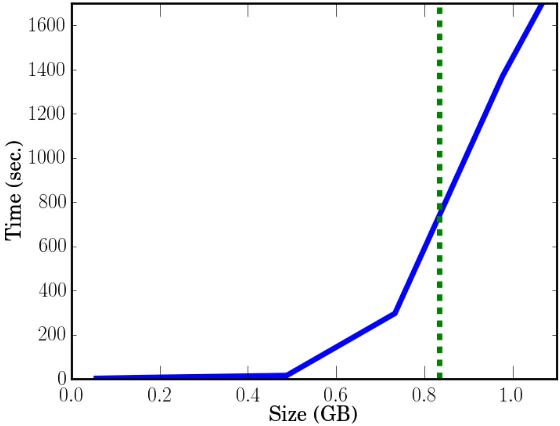
Data Beyond Memory Capacity

- 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 $\Rightarrow$ a subset

  - Example: Zinkevich et al. (2010)

Distributed Linear Classification (Cont'd)

Some results by averaging models

	yahoo-korea	kddcup10	webspam	epsilson
Using all	87.29	89.89	99.51	89.78
Avg. models	86.08	89.64	98.40	88.83

- Using all: solves a single linear SVM
- Avg. models: each node solves a linear SVM on a subset
- Slightly worse but in general OK

Distributed Linear Classification (Cont'd)

- 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)

$$\min_{\mathbf{w}_1,\ldots,\mathbf{w}_m,\mathbf{z}} \frac{1}{2}\mathbf{z}^T\mathbf{z} + C\sum_{j=1}^{m}\sum_{i\in B_j}\xi_{\text{L1}}(\mathbf{w};\mathbf{x}_i,y_i) + \frac{\rho}{2}\sum_{j=1}^{m}\|\mathbf{w}_j - \mathbf{z}\|^2$$

$$\text{subject to } \mathbf{w}_j - \mathbf{z} = \mathbf{0}, \forall j$$

- Each problem independently updated; but must collect $\mathbf{w}_j$
- 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 $\Rightarrow$ 70 minutes

Outline

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 $\mathbf{x}_i$ 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