Large-scale Non-linear Classification: Algorithms and Evaluations

Zhuang (John) Wang, Ph.D.
About the Tutorialist

- Work at Skytree, a California-based machine learning startup. Previously work at IBM Global Business Services and Siemens Research

- Research interests: Large-scale learning algorithms, machine learning applications, CRISP-DM

- 20+ papers on JMLR, ICML, KDD, AISTATS,…
Agenda

- Overview
- Large-scale *linear* classification basics
- Large-scale *non-linear* classification
- Parallelism
Most of the research in DM/ML has been directed to the problem of data classification in which algorithm learns linear/nonlinear models from data.

Non-linear data classification is particularly important as complex non-linear concepts often occur in the nature.

<table>
<thead>
<tr>
<th>Problem &amp; Data Understanding</th>
<th>Feature Engineering</th>
<th>Data Classification</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature1 … Feature k Label</td>
<td>Example 1 0.780 0.854 0.611 1.000</td>
<td>SVM classification plot</td>
<td></td>
</tr>
<tr>
<td>Example 2 0.486 0.928 0.519 0.000</td>
<td>…</td>
<td>…</td>
<td></td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td></td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td></td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td></td>
</tr>
<tr>
<td>Example n 0.885 0.255 0.113 0.000</td>
<td>…</td>
<td>…</td>
<td></td>
</tr>
</tbody>
</table>
New Requirements from Big Data

- Cheap, pervasive and networked computing devices are enhancing our ability to collect data to an even greater extent.

- What is big data?
  - A situation that \textit{exponentially grew complex} data makes us unable to easily make sense of it. So we need a wide variety of new technologies to tackle two key challenges: \textit{data management} and \textit{data analysis}. 

**Diagram:**
- Scalability
- Theory
- Accuracy
- Implementation
- Training time/ space
- Prediction time/ space

---

**Legend:**
- Ideal Algorithm
Large-scale Linear Classification Basics
Problem Setting

- Training examples:

\[ D = \{(x_i, y_i), \ i = 1, \ldots, N, \ x_i \in \mathbb{R}^M, \ y_i \in \{1, -1\}\} \]

- Goal: train a linear classifier to separate \( D \)

\[ \text{sgn}(f(x)) = \text{sgn}(w^T x), \text{ where } w \in \mathbb{R}^M \]

Note: we ignore bias term in \( f(x) \) for simplicity (bias term can be implicitly incorporated by adding a constant feature in the data)
Linear Support Vector Machine

- Train an optimal **linear** classifier by solving the optimization (Cortes et al., 1995)

Unconstrained form:

$$\min \frac{\lambda}{2} \| w \|^2 + \frac{1}{N} \sum_{i=1}^{N} \max(1 - y_i f(x_i), 0)$$

Constrained form:

$$\min_{w, \xi} \frac{1}{2} \| w \|^2 + C \sum \xi_i$$

s.t. \( y_i (w^T x_i) \geq 1 - \xi_i \)

- Note: in linear case, we can explicitly work on \( w \) (in primal form) rather than through SVs (in dual form), which makes our life much easier!!!

Fig. Source: http://www.ifp.illinois.edu/~yuhuang/sceneclassification.html
Stochastic Gradient Descent for Linear SVM

- SVM optimization
  $$\min_w \text{Obj}(w) = \frac{\lambda}{2} \| w \|^2 + \frac{1}{N} \sum_{i=1}^{N} \max(1 - y_i f(x_i), 0)$$

- Train SVM using gradient descent
  1. Initialize $w$
  2. Repeat until stopping criteria
     $$w \leftarrow w - \eta \frac{\partial \text{Obj}(w)}{\partial w}$$
     Require computation on all examples

- SGD: approximate the exact gradient by the gradient of the instantaneous objective
  $$w \leftarrow w - \eta \frac{\partial \text{InsObj}(w, i)}{\partial w}$$
  $$\text{InsObj}(w, i) = \frac{\lambda}{2} \| w \|^2 + \max(1 - y_i f(x_i), 0)$$

- Theory: when $i$ is i.i.d. sampled and #iterations is large, with high probability, $w$ converges to $w^*$ (Zhang, 2004; Shalev-Shwartz et al., 2008)
Stochastic Gradient Descent for Linear SVM (cont.)

- Algorithm (Zhang, 04; Shalev-Shwartz et al., 08)
  
  1. Initialize \( w \)
  2. Randomly select an example \( i \) in \( D \)
     - Do \( w \leftarrow (1 - \lambda \eta)w + \alpha_i x_i \) where \( \alpha_i = \begin{cases} \eta y_i, & \text{if } y_i f(x_i) < 1 \\ 0, & \text{otherwise} \end{cases} \)
  3. Repeat step 2 with enough iterations

- \( O(N) \) training time, \( O(M) \) training space*

*.: sequentially load data by chunk

Many research focusing on the learning rate design for speeding up training
Dual Coordinate Descent for Linear SVM

- SVM optimization in dual form
  \[ \max_{\alpha} \mathbf{1}^T \alpha - \frac{1}{2} \alpha^T Q \alpha, \quad \text{where} \quad Q_{ij} = y_i y_j k(x_i, x_j) \]
  s.t. \( \forall i, \ 0 \leq \alpha_i \leq C \)

- Idea: maximize the dual objective by iteratively optimizing one alpha at a time by keeping the rest variables fixed

- The new one-variable optimization leads the update rule:
  \[ \mathbf{w} \leftarrow \mathbf{w} + (\alpha_i^{\text{new}} - \alpha_i^{\text{old}}) \mathbf{x}_i, \quad \text{where} \quad \alpha_i^{\text{new}} = \min \left( \max \left( \alpha_i^{\text{old}} - \frac{y_i f(\mathbf{x}_i) - 1}{\| \mathbf{x}_i \|^2}, 0 \right), C \right) \]

- Algorithm (Hsieh et al., 08)
  - Initialize \( \mathbf{w} \) and \( \alpha_i^{\text{old}}, \ i = 1, \ldots, N \)
  - Iteratively access example \( i \) in \( D \) and update \( \mathbf{w} \) by the above rule (until stopping criteria)

  \( O(N) \) training time, \( O(N+M) \) training space
Other Popular Approaches

- Second-order stochastic gradient descent (Bordes et al., 2009)
- Bundle approach (Teo et al., 2010)
- Cutting plane approach (Joachims, 2006)
- Adaptive learning rate for SGD (Duchi et al., 2011)
- Methods for L1-regularized SVM and logistic regression

- Refer to the survey paper “Recent advance on large-scale linear classification” by Yuan et al.,
When Data Cannot Fit into Memory

- **Training time**
  
  \[ \text{Training time} = \text{in-memory computation time} + \text{I/O time} \]

- Prevent unnecessary I/O operation by fully operating on in-memory data rather than random access to disk

- **How?** Sequentially train data by chunk (Yu et al., 2010)
  
  - Not for every algorithm
  - Naturally fit for SGD and DCD

---

**Fig. Source:** [http://en.wikipedia.org/wiki/Virtual_memory](http://en.wikipedia.org/wiki/Virtual_memory) and Yu et al., 2011.

**When data becomes larger than RAM**

---

**How a linear complexity alg scales**
Toolboxes

- **Liblinear** (Fan et al., 2008)
  - Linear Classifiers (L1/L2 SVM, logistic regression)
  - Powered by dual coordinate descent, Newton method
  - Windows/Linux cmd-line tool with interfaces to many languages

- **Vowpal Wabbit** (http://hunch.net/~vw/)
  - Linear Classifiers (L1/L2 SVM, logistic regression)
  - Powered by gradient-based optimization
  - Linux cmd-line tool

- Both are
  - Well maintained projects
  - Suitable for single machine usage when data CANNOT fit into RAM
  - Train few GB data in a matter of secs/mins (on single machine)
  - Supporting distributed version
Why is Linear Classifier Popular?

- **Computationally cheap**
  - Train (or test) in linear time with $N$ (or $D$)

- **Sufficient accurate**
  - Carefully designed features already capture non-linear concepts, e.g. computer vision, computational advertising
  - In higher-dimensional feature spaces, data tends to be more linearly separable, e.g. document classification (bag-of-words representation)

- **Conceptually simple and interpretable**
  - Linear classifier with feature weighs is sort of “grey-box” model
Empirical Comparison between Linear and Non-linear Classification

<table>
<thead>
<tr>
<th>Data set</th>
<th>#instances</th>
<th>#features</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training</td>
<td>Testing</td>
</tr>
<tr>
<td>cod-RNA</td>
<td>59,535</td>
<td>271,617</td>
</tr>
<tr>
<td>ijcnn1</td>
<td>49,990</td>
<td>91,701</td>
</tr>
<tr>
<td>covtype</td>
<td>464,810</td>
<td>116,202</td>
</tr>
<tr>
<td>webspam</td>
<td>280,000</td>
<td>70,000</td>
</tr>
<tr>
<td>MNIST38</td>
<td>11,982</td>
<td>1,984</td>
</tr>
<tr>
<td>real-sim</td>
<td>57,848</td>
<td>14,461</td>
</tr>
<tr>
<td>rcv1</td>
<td>20,242</td>
<td>677,399</td>
</tr>
<tr>
<td>astro-physic</td>
<td>49,896</td>
<td>12,473</td>
</tr>
<tr>
<td>yahoo-japan</td>
<td>140,963</td>
<td>35,240</td>
</tr>
<tr>
<td>news20</td>
<td>15,997</td>
<td>3,999</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data set</th>
<th>Linear</th>
<th>Nonlinear (kernel)</th>
<th>Accuracy difference to nonlinear</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (s)</td>
<td>Testing accuracy</td>
<td>Time (s)</td>
</tr>
<tr>
<td></td>
<td>Training</td>
<td>Testing</td>
<td>Training</td>
</tr>
<tr>
<td>cod-RNA</td>
<td>3.1</td>
<td>70.71</td>
<td>80.2</td>
</tr>
<tr>
<td>ijcnn1</td>
<td>1.7</td>
<td>92.21</td>
<td>26.8</td>
</tr>
<tr>
<td>covtype</td>
<td>1.5</td>
<td>76.37</td>
<td>46,695.8</td>
</tr>
<tr>
<td>webspam</td>
<td>26.8</td>
<td>93.35</td>
<td>15,681.8</td>
</tr>
<tr>
<td>MNIST38</td>
<td>0.2</td>
<td>96.82</td>
<td>38.1</td>
</tr>
<tr>
<td>real-sim</td>
<td>0.3</td>
<td>97.44</td>
<td>938.3</td>
</tr>
<tr>
<td>rcv1</td>
<td>0.1</td>
<td>96.26</td>
<td>108.0</td>
</tr>
<tr>
<td>astro-physic</td>
<td>0.3</td>
<td>97.09</td>
<td>735.7</td>
</tr>
<tr>
<td>yahoo-japan</td>
<td>3.3</td>
<td>92.63</td>
<td>20,955.2</td>
</tr>
<tr>
<td>news20</td>
<td>1.2</td>
<td>96.95</td>
<td>383.2</td>
</tr>
</tbody>
</table>

Fig. Source: Yuan et al., 2012
Large-scale Non-linear Classification
When to Use Non-linear Classifier?

- When careful feature engineering still cannot cover all the complex non-linear concepts
- Accuracy critical use cases
  - E.g. even 1% accuracy improvement means a lot in your problem
Kernel Support Vector Machine

- Feature mapping
  
  \[ D = \{ (x_i, y_i), i = 1, ..., N \} \]
  
  \[ D' = \{ (\Phi(x_i), y_i), i = 1, ..., N \} \]

- SVM optimization on \( D' \):
  
  \[ \min_w = \frac{\lambda}{2} \| w \|^2 + \frac{1}{N} \sum_{i=1}^{N} \max(1 - y_i f(x_i), 0) \]

  where \( f(x) = w^T \Phi(x) \)

- Primal to dual transformation \( \Rightarrow f(x) = \sum_i \alpha_i y_i \Phi(x_i)^T \Phi(x) = \sum_i \alpha_i y_i k(x_i, x) \)

Note: \( w \) can be implicitly represented by SVs + their coefficients + kernel function

Fig. Source: www.imtech.res.in.
Decomposition Methods - SMO

- SVM dual form
  \[
  \max_{\alpha} \mathbf{1}^T \alpha - \frac{1}{2} \alpha^T Q \alpha, \quad \text{where } Q_{ij} = y_i y_j k(x_i, x_j)
  \]
  s.t. \( \forall i, \ 0 \leq \alpha_i \leq C \)

- Sequential Minimal Optimization (Platt, 98)
  The same idea as DCD for linear SVM but here we have to implicitly deal with \( w \)
  
  1. **Smartly select** a working example \( i \) and update \( \alpha_i \) by solving the one-variable optimization
    \[
    \max_{\alpha_i} \left( (1 - Q_{iU} \alpha_U) \alpha_i - \frac{1}{2} \alpha_i Q_{ii} \alpha_i \right), \quad \text{s.t. } 0 \leq \alpha_i \leq C, \\
    \text{Closed-form solution for } \alpha_i
    \]

  2. Repeat step 1 until stopping criteria
Decomposition Methods - Libsvm

- Libsvm (Chang and Lin, 01)
  - Highly optimized implementation of SMO (plus heuristic for fast convergence)
  - Actively-maintained open source project
  - Windows/Linux cmd-line tool and multiple language APIs
  - Exact SVM solver
  - Scalable with x00,000 low-dimensional examples*

*: we define “scalable” as training time less than 10hrs.
Decomposition Methods - Lasvm

- Lasvm (Bortes et al., 05): approximate SVM solver using online SMO approximation
  - Using less memory than Libsvm
  - Less accurate
  - Scalable with x,000,000 low-dimensional examples

- Lasvm algorithm
  - Online step
    • Access examples and add to $S$
    • Loosely run SMO on $S$
    • Delete some (currently) useless examples from $S$
  - Finishing step
    • Run full SMO on $S$
    • Tunable to switch between the 2 steps

Fig. Source: http://leon.bottou.org/projects/lasvm
Minimal Enclosing Ball Methods

- Minimal Enclosing Ball (MEB): the ball with the smallest radius that encloses all the points in a given set

\[ S_\varphi = \{ \varphi(x_1), \ldots, \varphi(x_n) \} \]

\[ (c^*, R^*) = \arg \min_{c,R} R^2 : \|c - \varphi(x_i)\|^2 \leq R^2 \ \forall i. \]

Dual form is a QP:

\[ \max \alpha' \text{diag}(K) - \alpha'K\alpha : 0 \leq \alpha, \ \alpha'1 = 1 \]

- Fast iterative approximate solver available for MEB optimization

Fig. Source: Tsang et al., 2005.
Minimal Enclosing Ball Methods (cont.)

- CVM (Tsang et al., 2005): square-loss SVM can be casted into a MEB problem

\[
\max \alpha' \text{diag}(K) - \alpha'K\alpha : \quad 0 \leq \alpha, \quad \alpha'1 = 1
\]

MEB dual:  

After dropping a constant here

Square-loss SVM dual:  

\[
\max -\alpha'\tilde{K}\alpha : \quad 0 \leq \alpha, \quad \alpha'1 = 1
\]

Kernel relationship:

\[
\tilde{k}(z_i, z_j) = y_i y_j k(x_i, x_j) + y_i y_j + \frac{\delta_{ij}}{C}
\]

- Thus SVM can be efficiently solved by an MEB solver

- BVM (Tsang et al., 2007): faster version of CVM by further approximation
Empirical Comparison: B/CVM vs Libsvm vs Lasvm

(a) testing accuracy (in %).

(b) CPU time.

(c) # support vectors.

Fig. Source: Tsang et al., 2007.
SGD with Kernel

- **Algorithm**

The same idea as SGD for linear SVM but here we have to implicitly deal with $w$

$$w = \text{Support Vectors (SVs) + their coefficients + kernel function}$$

1. Initialize $w$
2. Randomly select an example $i$ in $D$

   - Do $w \leftarrow (1 - \eta_i \lambda)w + \beta_i \Phi(x_i)$ where $\beta_i = \begin{cases} \eta_i y_i, & \text{if } y_if(x_i) < 1 \\ 0, & \text{otherwise} \end{cases}$

3. Repeat step 2 with enough iterations

- Scalable with $x0,000$ examples but not suitable for larger data
Budgeted SGD

- **BSGD algorithm** (Wang et al., 2012)
  1. Initialize \( \mathbf{w} \), set budget \( B \)
  2. Randomly select an example \( i \) in \( D \)
     - Do
       - \( \mathbf{w} \leftarrow (1-\eta_i \lambda) \mathbf{w} + \beta_i \Phi(x_i) \) where \( \beta_i = \begin{cases} \eta_i y_i, & \text{if } y_i f(x_i) < 1 \\ 0, & \text{otherwise} \end{cases} \)
     - if (#SVs > \( B \)) then \( \mathbf{w} \leftarrow \mathbf{w} - \Delta_i \) // reduce the size of SVs by one (while changing SVs and their coefficients to minimize the degradation)

   **Recall:** \( \mathbf{w} = \text{Support Vectors (SVs)} + \text{their coefficients} + \text{kernel function} \)
  3. Repeat step 2 with enough iterations

- **Design philosophy:**
  
  \[
  \text{Obj}(\mathbf{w}_N) - \text{Obj}(\mathbf{w}^*) \leq \frac{C_1 \ln(N)}{N} + C_2 \bar{E}
  \]
  
  where
  \[
  \bar{E} = \frac{1}{N} \sum_{t=1}^{N} \| \frac{\Delta_t}{\eta_t} \|
  \]
  
  \[
  \min \| \bar{E} \| \quad \min \| \Delta_t \| 
  \]
Budgeted Online Kernel Classifiers

- Algorithm framework
  - Iteratively access example $i$ in $D$
    - Do $w \leftarrow \alpha_i w + \beta_i \Phi(x_i)$
      where $\alpha_i$ and $\beta_i$ are calculated by old $w$ on $(x_i, y_i)$
    - if (#SVs > $B$) then $w \leftarrow w - \Delta_i$  \(//\) reduce the size of SVs by one (while changing SVs and their coefficients)

- Budget maintenance strategies
  - Removal-based (Cesa-Bianchi & Gentile, 06; Vucetic et al., 09; Dekel et al., 08; Cheng et al., 07; Crammer et al., 04; Weston et al., 05; Wang & Vucetic, 09)
  - Projection-based (Wang and Vucetic, 2010)
  - Merging-based (Wang et al., 2012)
Linearization Methods

- **Idea**
  - Explicitly represent data in a feature space
  - Train a linear SVM there

- **Exact methods:**
  - Poly2SVM (Chang et al., 2010), Coffin (Sonnenburg et al., 2010)

\[
K(x_i, x_j) = (\gamma x_i^T x_j + r)^d \quad \text{where } r = 1, \ d = 2
\]

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

- **Approximate methods:**
  - Random Features (Rahimi and Recht, 2007), Fastfood (Le et al., 2013), LLSVM (Zhang et al., 2012),

\[
k(x, y) = \langle \phi(x), \phi(y) \rangle \approx z(x)' z(y) \quad z(x) : \mathcal{R}^d \to \mathcal{R}^D
\]

Fig. Source: www.imtech.res.in.
Linearization Methods - LLSVM

- LLSVM (Zhang et al., 12): cast nonlinear SVM into an equivalent linear SVM through the decomposition of PSG kernel matrix

\[
K_{NN} = F_N B F_N^T, \quad \text{where } B \text{ is the rank of } K
\]

\[
K_{ij} = \Phi(x_i)^T \Phi(x_j) = F_i^T F_j
\]

\[
\min_{w,\xi} \frac{1}{2} \|w\|^2 + C \sum \xi_i
\]

s.t. \( y_i (w^T \Phi(x_i)) \geq 1 - \xi_i \)

A high-dim. example in feature space

\[
\min_{w,\xi} \frac{1}{2} \|w\|^2 + C \sum \xi_i
\]

s.t. \( y_i (w^T F_i) \geq 1 - \xi_i \)

\( B \)-dim. virtual example
Linearization Methods - LLSVM (cont.)

- Approximate the optimal decomposition by Nyström method

\[
K_{N \times N} \approx K_{N \times B} \Lambda^{-1/2} U \Lambda^{-1/2} (K_{N \times B} \Lambda^{-1/2})^T
\]

- LLSVM algorithm:
  1. Select \( B \) landmarks points using random sampling
  2. Compute eigen-decomposition of \( K_{BB} \): \( M = U \Lambda^{1/2} \)
  3. Train linear SVM on the virtual examples, where \( F_{N \times B} = K_{N \times B} U \Lambda^{1/2} \)

\( O(N) \) time complexity
New Formulation - Ramp Loss SVM

- SVM is less scalable on noisy data: the hinge loss forces all the noisy examples become SVs and a lot of SVs slows down convergence.

\[ w^* = -C \sum_i y_i H'(y_i, f(x_i))\Phi(x_i) \]

- Replacing hinge loss with ramp loss in the SVM optimization (Collobert et al., 06)

\[ \min_w \frac{1}{2} \|w\|^2 + C \sum_{t=1}^N R(y_t, f(x_t)) \]
New Formulation - Ramp Loss SVM (cont.)

- Ramp loss SVM algorithm (by ConCave Convex Procedure)
  1. Initialization: train $f^{(\text{old})}$ on a small random subset $V$ of $D$
  2. Repeat the following 3 steps until $V$ is static
     - Calculate $y_if^{(\text{old})}(x_i)$ for all $i$ in $D$
     - Train $f^{(\text{new})}$ on a subset $V$,
       where $V = \{(x_i,y_i), \text{any } i, y_if^{(\text{old})}(x_i) > -1\}$
     - $f^{(\text{old})} = f^{(\text{new})}$

Two Gaussians  SVM solution  Ramp loss SVM
New Formulation - AMM

- Idea: Similar to Crammer&Singer’s multi-class SVM formulation but assigning multiple linear hyperplanes to each class to increase representability (Aiolli & Sperduti, 05; Wang et al., 11)

\[ f(x) = \arg \max_{i \in Y} g(i, x) \]

where \( g(i, x) = \max_j w_{i,j}^T x \),

\[
W = \begin{bmatrix}
w_{1,1} \ldots w_{1,b_1} & w_{2,1} \ldots w_{2,b_2} & \ldots & w_{M,1} \ldots w_{M,b_M}
\end{bmatrix}
\]

\[
\min_W P(W) = \frac{\lambda}{2} \|W\|^2 + \frac{1}{N} \sum_{n=1}^{N} l(W; (x_n, y_n))
\]

where \( l(W; (x_n, y_n)) = \max \left( 0, 1 + \max_{i \in Y \setminus y_n} g(i, x_n) - g(y_n, x_n) \right) \)

Maximal prediction from the incorrect classes

The maximal prediction from the correct class

<table>
<thead>
<tr>
<th>Class 1</th>
<th>( w_{11} )</th>
<th>( w_{11}^T x = 1.2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( w_{12} )</td>
<td>( w_{12}^T x = 0.4 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class 2</th>
<th>( w_{21} )</th>
<th>( w_{21}^T x = 1.5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( w_{22} )</td>
<td>( w_{22}^T x = -0.1 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class 3</th>
<th>( w_{31} )</th>
<th>( w_{31}^T x = -0.7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( w_{32} )</td>
<td>( w_{32}^T x = 0.1 )</td>
</tr>
<tr>
<td></td>
<td>( w_{33} )</td>
<td>( w_{33}^T x = 0.6 )</td>
</tr>
</tbody>
</table>

The non-convex optimization is solved by a series of convex approximations
New Formulation – AMM (cont.)

- AMM fills the scalability and representability gap between Linear and RBF SVM.

RBF SVM didn’t finish in 10hrs.
**Toolbox**

- **BudgetedSVM**: a toolbox for scalable SVM approximations (Djuric, et al., 13)
  - Command-line (Windows/Linux/Mac), Matlab interfaces, C/C++ APIs
  - Include AMM, BSGD, LLSVM, SGD
  - Highly-optimized for large data when it cannot fit into memory
    (Online learning + constant-memory = scalable for arbitrarily large data)
  - Open source and commercial friendly (Modified BGD license)

Complexity Comparison

<table>
<thead>
<tr>
<th></th>
<th>Linear SVM (SGD)</th>
<th>AMM</th>
<th>LLSVM</th>
<th>BSGD</th>
<th>RBF-SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training time</td>
<td>(O(NCS))</td>
<td>(O(NSB))</td>
<td>(O(NSB^2 + NSB))</td>
<td>(O(N(C + S)B))</td>
<td>(O(INCS))</td>
</tr>
<tr>
<td>Prediction time</td>
<td>(O(CS))</td>
<td>(O(SB))</td>
<td>(O(SB^2 + SB))</td>
<td>(O((C + S)B))</td>
<td>(O(NCS))</td>
</tr>
<tr>
<td>Model size</td>
<td>(O(CD))</td>
<td>(O(DB))</td>
<td>(O(DB + B^2))</td>
<td>(O((C + D)B))</td>
<td>(O(NCS))</td>
</tr>
</tbody>
</table>

**AMM/Pegasos classifier:**

\[
f(x) = \arg\max_{i \in \mathcal{Y}} g(i, x), \quad \text{where} \quad g(i, x) = \max_{j} w_{ij}^T x
\]

**BSGD/RBF-SVM classifier:**

\[
f(x) = \arg\max_{i \in \mathcal{Y}} g(i, x), \quad \text{where} \quad g(i, x) = \sum_{j=1}^{B} \alpha_{ij} k(z_j, x)
\]

**LLSVM classifier:**

\[
f(x) = \text{sign}(w^T (M \cdot g(x))), \quad g(x) = [k(x, z_1), \ldots, k(x, z_B)]^T, \quad M \text{ is a } B \times B \text{ mapping matrix}
\]

- \(N\): #training examples
- \(D\): data dimensionality
- \(C\): #classes
- \(S\): average #non-zero features
- \(I\): #iteration for Libsvm, \(I = O(N) \sim O(N^2)\)
- \(B\): budget size for BSGD, #hyperplanes for AMM, #landmark points for LLSVM, \(B << N\)
Error Rate and Training Time Comparison

<table>
<thead>
<tr>
<th>Data set</th>
<th>Linear SVM (SGD)</th>
<th>AMM</th>
<th>LLSVM</th>
<th>BSGD</th>
<th>RBF-SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>e.r.</td>
<td>t.t.</td>
<td>e.r.</td>
<td>B</td>
<td>t.t.</td>
</tr>
<tr>
<td>wasn't</td>
<td>7.94</td>
<td>0.5s</td>
<td>4.74</td>
<td>9</td>
<td>3s</td>
</tr>
<tr>
<td>N = 280,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D = 254</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rcv1</td>
<td>2.73</td>
<td>1.5s</td>
<td>2.39</td>
<td>19</td>
<td>9s</td>
</tr>
<tr>
<td>N = 677,399</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D = 47,236</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mnist8m-bin</td>
<td>22.71</td>
<td>1.1m</td>
<td>3.16</td>
<td>18</td>
<td>4m</td>
</tr>
<tr>
<td>N = 8,000,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D = 784</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tradeoff between acc. and time by B (example by LLSVM)
Train with Smaller Size - Sampling Methods

- Train algorithms on a random subset of the data
- KDDCUP09 data (~5M examples, 129 dim.)
  - Best reported accuracy ~94%
  - Sampling method (using only 50 examples) + SVM, accuracy: ~92%, training time less than 1s
- Covetype data (500K examples, 57 dim.)
- Rcv1 data (550K examples, 47K dim.)

Accuracy can be further boosted by bagging \( F(x) = \text{ave}(f_i(x)) \)

<table>
<thead>
<tr>
<th>covtype</th>
<th>50k</th>
<th>92.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100k</td>
<td>95.3%</td>
</tr>
<tr>
<td></td>
<td>500k</td>
<td>98.2%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rcv1</th>
<th>50k</th>
<th>97.2%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100k</td>
<td>97.4%</td>
</tr>
<tr>
<td></td>
<td>550k</td>
<td>97.8%</td>
</tr>
</tbody>
</table>

Fig. Source: C. J. Lin, Talk at K. U. Leuven Optimization in Engineering Center, 2013.
Jochen Garcke, presentation at ICML'08 Workshop PASCAL Large Scale Learning Challenge, 2008
Train with Smaller Size - Data Summary Methods

- Summarize the data using meta-examples, then train model on meta-examples.

\[ D = \{(x_i, y_i), i = 1, \ldots, N\} \]

\[ D' = \{(q_i, y_i), i = 1, \ldots, B\}, \]

\[ B << N \]

data quantization or clustering
Train with Smaller Size – Data Summary Methods (cont.)

- **Simple approach**
  - pre-clustering on the data
  - train weighted SVM on cluster centers, where example weights are determined by the size/purity of the clusters

- **Support Cluster Machine** (Li et al., 07)
  - pre-clustering on the data
  - train weighted SVM on clusters, where clusters are treated as Gaussian distribution and the similarity is calculated by probability product kernel

- **Online Twin Vector Machine** (Wang and Vucetic, 10b)
  - Sequentially access example one at a time
  - Maintain a set of clusters by online clustering
  - Incremental and decremental learning with SVM on the clusters
Train with Smaller Size - Approximate Extreme Points

- **AESVM** (Nandan et al., 2014)
  - Train a non-linear SVM on $X$, the *representative set* of $D$

- **What is the representative set? Why does it make sense?**
  - Given a set of points, the *(approximate) extreme points* are those by which any point in the set can be (approximately) represented as a convex combination
  - **Representative set**: the union of all the approximate extreme point sets that each is separately obtained from a disjoint subset of $X$
  - Finding the representative set is in linear time

- Theoretical justification: the optimization gap between the optimal solution of SVM and AESVM is bounded by $O(C\sqrt{C\varepsilon})$ under certain assumption
Train with Smaller Size - Approximate Extreme Points Approach (cont.)

Fig. Source: Nandan et al., 2014.
Parallelism
When to Use Parallel Algorithm?

- Parallel computing environments have been so common. Why not take advantage of them?

- Learn extremely large data:
  
  \[
  \text{data loading I/O time} \gg \text{training time}
  \]
Parallel Kernel SVM – Cascade SVM

Cascade SVM (Graf et al., 05)

- Distribute data into nodes
- In each pass
  - Train local SVMs at the nodes of the current layer
  - Transfer the local SV sets to the next layer
- Converge after few passes in practice
Parallel Kernel SVM – Parallel Interior Point SVM

- PSVM (Chang et al., 07) - parallel Interior-Point method
  - IP method
    - Remove the linear constraint in SVM’s QP with barrier function
    - Then solve a sequence of the unconstraint problems with Newton method
    - $O(N^3)$ time and $O(N^2)$ space which is dominated by inverting kernel matrix

- Parallel IP method
  - Distribute both data loading and computation
  - Approximate expensive matrix manipulations using parallel computing
  - Intense communication between nodes
Parallel Kernel SVM – Parallel SGD

- P-pack SVM (Zhu et al., 09)
  - Distribute both the storage and computation of SVs across nodes

- Parallel SGD algorithm for kernel SVM
  1. Initialize $w$
  2. All nodes load a same example $i$ from $D$
     - Do $w \leftarrow (1 - \eta_i \lambda)w + \beta_i \Phi(x_i)$ where $\beta_i = \begin{cases} \eta_i y_i, & \text{if } y_i f(x_i) < 1 \\ 0, & \text{otherwise} \end{cases}$
     (If $x_i$ being determined as an SV, only one node stores it)
  1. Repeat step 2 with enough iterations
Bagging Method

- PSGD (Zinkevish et al., 10): Bagging + Linear SVM by SGD
  - Approximate solver
  - Little communication between nodes
  - Suitable for MapReduce

<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>

Fig. Source: C.-J. Lin, Talk at K. U. Leuven Optimization in Engineering Center, 2013.
ADMM Method

- ADMM for SVM (Boyd et al., 11; Zhang et al., 12b)

\[ \min_{w_1, \ldots, w_m, z} \frac{1}{2} \|z\|_2^2 + C \sum_{j=1}^m \sum_{i \in B_j} \max(1 - y_i w_j^T x_i, 0)^2 \]

\[ + \sum_{j=1}^m \frac{\rho}{2} \|w_j - z\|_2^2, \]

subject to \( w_j - z = 0, \quad j = 1, \ldots, m \)

\[ w_j^{k+1} = \arg \min_w C \sum_{i \in B_j} \max(1 - y_i w^T x_i, 0)^2 \]

\[ + \frac{\rho}{2} \|w - z^k + u_j^k\|_2^2, \quad j = 1, \ldots, m. \]

\[ z^{k+1} = \sum_{j=1}^m \frac{(w_j^{k+1} + u_j^k)}{m + 1/\rho} \]

\[ u_j^{k+1} = u_j^k + w_j^{k+1} - z^{k+1}, \quad j = 1, \ldots, m. \]

**Fig. Source:** Zhang et al., 2012b.

Optimized by a fast solver

---

Data \( x_{B_1} \rightarrow w_1, u_1 \)

Data \( x_{B_2} \rightarrow w_2, u_2 \)

\( \vdots \)

Data \( x_{B_m} \rightarrow w_m, u_m \)

\[ \bar{w} + \bar{u} \rightarrow z \]

MPI
MPI Allreduce Framework

- **MPI Allreduce**: for parallel applications which require accessing the global results across all processes rather than the root process

- **Distributed Vowpal Wabbit** (Agarwal et al., 2014) – distributed L-BFGS implemented in MPI Allreduce
  - Setup: distribute examples across nodes
  - Warm start
    - For each node, **local** \( w \) is trained by SGD-based algorithm
    - Initialize **global** \( w \) by weighted average of all local \( w \)’s
  - L-BFGS distributed training (iteratively run L-BFGS)
    - In each iteration, the global gradient is computed by summing up all local gradients and then is pushed back to all local nodes

- **Distributed Liblinear** (Lin, 2014)
  - Use a similar “**Allreduce**” strategy for distributed calculating Hessian-vector products in TRON (a Newton Method) for L2LR

Fig. Source: Agarwal et al., 2014.
Empirical Comparisons of Parallel Linear Classifiers

- Setting: Train *L2*-regularized *L2*-hingeLoss SVM with a data size (~9GB, N = ~20M, D = ~20M) on a cluster of 8 nodes with total RAM 96GB) by instance-wise distribution

Fig. Source: Zhang et al., 2012b.
Empirical Comparisons of Parallel Linear Classifiers (cont.)

- Setting: Train **L2-regularized Logistic Regression** on a cluster of 16 machines

<table>
<thead>
<tr>
<th>Data set</th>
<th>$l$</th>
<th>$n$</th>
<th>#nonzeros</th>
</tr>
</thead>
<tbody>
<tr>
<td>epsilon</td>
<td>400,000</td>
<td>2,000</td>
<td>800,000,000</td>
</tr>
<tr>
<td>webspam</td>
<td>350,000</td>
<td>16,609,143</td>
<td>1,304,697,446</td>
</tr>
</tbody>
</table>

IW: instance-wise distribution
FW: feature-wise distribution

Parallel (Deep) Neural Network

- DownpourSGD (Dean et al., 2012) for training (deep) neural network
  - Keep multiple model replicas where each is distributed on multiple machines
  - Distributed optimization within model replica through framework-managed communication, synchronization, data transfer with each model replica
  - Distributed optimization across multiple model replicas through asynchronously fetch global $w$ from and push local gradients to the centralized server
Parallel (Deep) Neural Network (cont.)

- Sandblaster-L-BFGS (Dean et al., 2012) for training (deep) neural network
  - In each L-BFGS iteration, the local gradient is computed at each model replica against a disjointed subset of data (which is assigned by the coordinator)
    - Faster replica will get more works to complement slower replica
    - After all the data is consumed, a global weight is computed at parameter server and then being fetched by model replicas
Training a 5-layer NN with 42 million of model parameters for speech recognition task.

Fig. Source: Dean et al. 2012.
# Comparisons of Parallel Computing Frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Pros</th>
<th>Cons</th>
<th>Comments</th>
<th>Suitable ML use cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapReduce</td>
<td>Fault tolerance</td>
<td>Communication overhead; Frequent re-loading data from sources; Good design of MR algorithm is often hard</td>
<td>Communication is most based on I/O</td>
<td>Extremely large data; Algorithms only require few iterations</td>
</tr>
<tr>
<td>Spark</td>
<td>Fault tolerance; Optimized for iterative process; Scala might be less efficient than C/C++ implementation</td>
<td>Data can be cached in memory between iterations</td>
<td>Algorithms require a lot of iterations</td>
<td></td>
</tr>
<tr>
<td>MPI</td>
<td>Mature and highly-optimized framework;</td>
<td>No/limited fault tolerance</td>
<td>Communication is via memory</td>
<td>Relatively large data (few hr training time; daily ML tasks)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Full size</th>
<th>10% sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapReduce</td>
<td>1690</td>
<td>1322</td>
</tr>
<tr>
<td>AllReduce (MPI)</td>
<td>670</td>
<td>59</td>
</tr>
</tbody>
</table>

Average training time per iteration of an internal logistic regression implementation using either MapReduce or AllReduce for gradients aggregation. The data set is the display advertising one and a subset of it.

Bottom Table Source: Agarwal et al., 2014.
Parallel Tree-based Methods

- Refer to the tutorial “Scale up decision tree ensembles” by M. Bilenko, R., Bekkerman, and J. Langford at KDD 2011
Conclusions

- Linear classification
  - Mature research area
  - Often accurate enough for many applications

- Non-linear classification
  - Suitable for accuracy-critical use cases
  - Approximate algorithms dominate

- Parallelism
  - Active research area
  - Good design = Maths + system
Thank you!

- My homepage at: [zhuang-john-wang.com](http://zhuang-john-wang.com)

References

References (cont.)

References (cont.)

References (cont.)


