Distributed linear SVM with the Alternating Direction Method of Multipliers

Raoul Lefmann

TU Dortmund

18. August 2016

Implementation of

C. Zhang, H. Lee, und K. G. Shin.
Efficient distributed linear classification algorithms via the alternating direction method of multipliers.
In *Proceedings of AISTATS 2012*, Seiten 1398–1406, 2012.

Outline

Consensus SVM

Dual coordinate descent

Implementation

Experiments

Consensus SVM: support vector machines

- Binary classification problem
- Dataset $\mathcal{D} = \{(x_i, y_i) \mid x_i \in \mathbb{R}^d, y_i \in \{-1, +1\}, i = 1, \ldots, m\}$
- Find hyperplane $H = \{x \mid w^T x + b = 0\}$ that separates classes with maximum margin
- Incorporate b into w:

$$x_i^T \leftarrow [1, x_i^T] \qquad w^T \leftarrow [b, w^T] \qquad d \leftarrow d + 1$$

- SVM can be formulated as an unconstrained optimization problem:

$$\min_w \frac{1}{2}\|w\|_2^2 + C \sum_{i=1}^{m} \ell(w, x_i, y_i)$$

- ℓ is a loss function. We use squared hinge loss (L2-SVM):

$$\min_{w} \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{m} \max\left\{0, 1 - y_i w^T x_i\right\}^2$$

- Equivalent constrained formulation:

$$\begin{aligned} \min_{w,\xi} \quad & \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{m} \xi_i^2 & \\ s.t. \quad & y_i w^T x_i \geq 1 - \xi_i & i = 1, \ldots, m \\ & \xi_i \geq 0 & i = 1, \ldots, m \end{aligned}$$

Consensus SVM: ADMM

- Framework for distributed optimization
- Optimization problems of type

$$\begin{aligned} \min_{w,z} \quad & f(w) + g(z) \\ s.t. \quad & Aw + Bz = c \end{aligned}$$

- Uses augmented Lagrangian:

$$\mathscr{L}_\rho(w, z, \lambda) = f(w) + g(z) + \lambda^T (Aw + Bz - c) + \frac{\rho}{2} \|Aw + Bz - c\|_2^2$$

- Update steps:

$$w \leftarrow \operatorname*{argmin}_w \mathscr{L}_\rho(w, z, \lambda) \tag{1}$$

$$z \leftarrow \operatorname*{argmin}_z \mathscr{L}_\rho(w, z, \lambda) \tag{2}$$

$$\lambda \leftarrow \lambda + \rho(Aw + Bz - b) \tag{3}$$

Consensus SVM

- Assume the dataset $\mathcal{D}$ is split across N nodes in a network. Let $B_i = \{j \mid (x_j, y_j) \in \mathcal{D} \text{ is stored in node } i\}$.
- Reformulate SVM as a consensus problem:

$$\begin{aligned} \min_{w,z} \quad & \frac{1}{2}\|z\|_2^2 + C\sum_{i=1}^{N}\sum_{j\in B_i} \max\{0, 1 - y_j\langle w_i, x_j\rangle\}^2 \\ s.t. \quad & w_i - z = 0 \quad i = 1, \ldots, N \end{aligned}$$

- Each node learns its own local w_i, consensus is reached via z
- Consensus problem can be solved in parallel using ADMM

Consensus SVM: ADMM updates

$$
\begin{aligned}
w_i &\leftarrow \operatorname*{argmin}_{w_i} \mathscr{L}_\rho(w, z, \lambda) \\
&= \operatorname*{argmin}_{w_i} C \sum_{j \in B_i} \max\{0, 1 - y_j \langle w_i, x_j \rangle\}^2 + \frac{\rho}{2} \|w_i - z\|_2^2 + \lambda_i (w_i - z) \\
z &\leftarrow \operatorname*{argmin}_{z} \mathscr{L}_\rho(w, z, \lambda) \\
\lambda_i &\leftarrow \lambda_i + \rho(w_i - z)
\end{aligned}
$$

- w-update and λ-update can be computed in parallel
- z-update has nice closed form solution

$$
z = \frac{\sum_{i=1}^N (w_i + \lambda_i)}{1 + \rho N}
$$

Consensus SVM: Reformulation

We can set $\mu_i = \frac{\lambda_i}{\rho}$ to obtain a simpler formulation:

$$w_i \leftarrow \operatorname*{argmin}_{w_i} C \sum_{j \in B_i} \max\{0, 1 - y_j \langle w_i, x_j \rangle\}^2 + \frac{\rho}{2} \|w_i - z - \mu_i\|_2^2$$

$$z \leftarrow \frac{\sum_{i=1}^{N} (w_i + \mu_i)}{N + 1/\rho}$$

$$\mu_i \leftarrow \mu_i + w_i - z$$

Dual coordinate descent

- We need to find a way to compute w-update efficiently

$$\underset{w_i}{\operatorname{argmin}} \frac{\rho}{2}\|w - v\|_2^2 + C\sum_{j=1}^{s} \max\{0, 1 - y_j w_i^T x_j\}^2$$

where $(x_1, y_1), \ldots, (x_s, y_s)$ are data on machine i and $v = z - \mu_i$

- Equivalent constrained problem:

$$\begin{aligned} \min_{w,\xi} \quad & \frac{\rho}{2}\|w - v\|_2^2 + C\sum_{i=1}^{s} \xi_i^2 & \\ s.t. \quad & y_i w^T x_i \geq 1 - \xi_i & i = 1, \ldots, s \\ & \xi_i \geq 0 & i = 1, \ldots, s \end{aligned}$$

Dual coordinate descent: duality

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2\rho}\alpha^T(Q+D)\alpha - b^T\alpha \\ s.t. \quad & \alpha_i \geq 0 \qquad\qquad\qquad\qquad i = 1, \ldots, s \end{aligned}$$

- $\alpha \in \mathbb{R}^s$
- $Q_{ij} = y_i y_j x_i^T x_j$
- D: diagonal matrix with $D_{ii} = \frac{\rho}{2C}$
- $b = [1 - y_1 v^T x_1, \ldots, 1 - y_s v^T x_s]^T$

Dual coordinate descent (DCD)

- Outer loop: update α in each iteration
- Inner loop: update each α_i separately
- Optimize one α_i at a time and then circularly move to the next variable
- The optimization for α_i has a closed form solution! 🙂

$$\alpha_i = \max\left\{0, \alpha_i - \frac{\rho}{(Q+D)_{ii}}\nabla_i\right\}$$

where ∇_i is the partial derivative w.r.t. α_i

DCD: compute partial derivative

- From the derivation of the dual we know

$$w = v + \frac{1}{\rho} \sum_{j=1}^{s} \alpha_j y_j x_j$$

- To get ∇_i we can first calculate w and then

$$\nabla_i = y_i w^T x_i + \alpha_i D_{ii} - 1$$

- We can update w easily once we have the new α_i:

$$w^{(t+1)} = w^{(t)} + (\alpha_i^{(t+1)} - \alpha_i^{(t)}) y_i x_i$$

DCD: projected partial derivative

- If $\nabla_i = 0$ and $\alpha_i > 0$, we do not need to update α_i
- Also if $\alpha_i = 0$ and $\nabla_i > 0$
- Projected partial derivative:

$$\tilde{\nabla}_i = \begin{cases} \min\{0, \nabla_i\} & \text{if } \alpha_i = 0, \\ \nabla_i & \text{otherwise} \end{cases}$$

$\Rightarrow$ Don't update α_i if $|\tilde{\nabla}_i| = 0$

Implementation: parallel computation

- Using Julias built-in parallelization framework
- processes are called workers

```
addprocs(N)
```

- Data are transformed into a `DistributedArray`

```
x = distribute(x, dist=[N,1])
y = distribute(y)
```

- Each worker has its local part on which it runs DCD

```
dcd(localpart(x), localpart(y), ...)
```

Implementation: program structure

main.jl Creates workers, preprocesses data and invokes ADMM.

admm.jl Implements the ADMM framework. Distributes code and data to the workers. Computes z and μ. Manages parallel computation of w_j and collects the results from the workers.

coord.jl Implements the dual coordinate descent. Must be available to all workers.

Implementation: stopping criteria for DCD

- Use the projected partial derivative as stopping criterion:

$$\max_i |\tilde{\nabla}_i| < \varepsilon$$

  Somehow this doesn't work, because $\tilde{\nabla}_i$ values get larger
- Alternative: use duality gap, but expensive to compute 😕
- Had to use maximum number of iterations as a stopping criterion

Experiments: spam detection

- Important application
- Typical linear classification task
- Users want better spam filters, but don't want to share their private emails

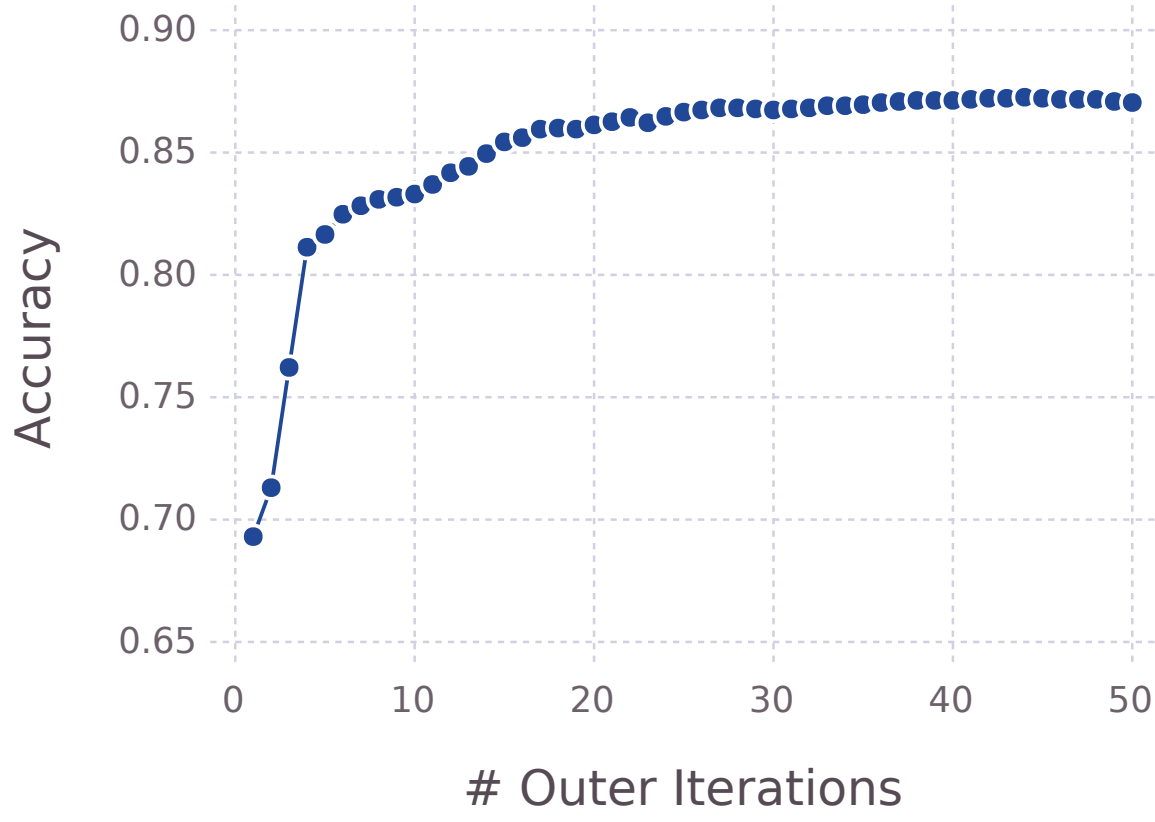
Experiments: dataset

UCI Spambase Data Set

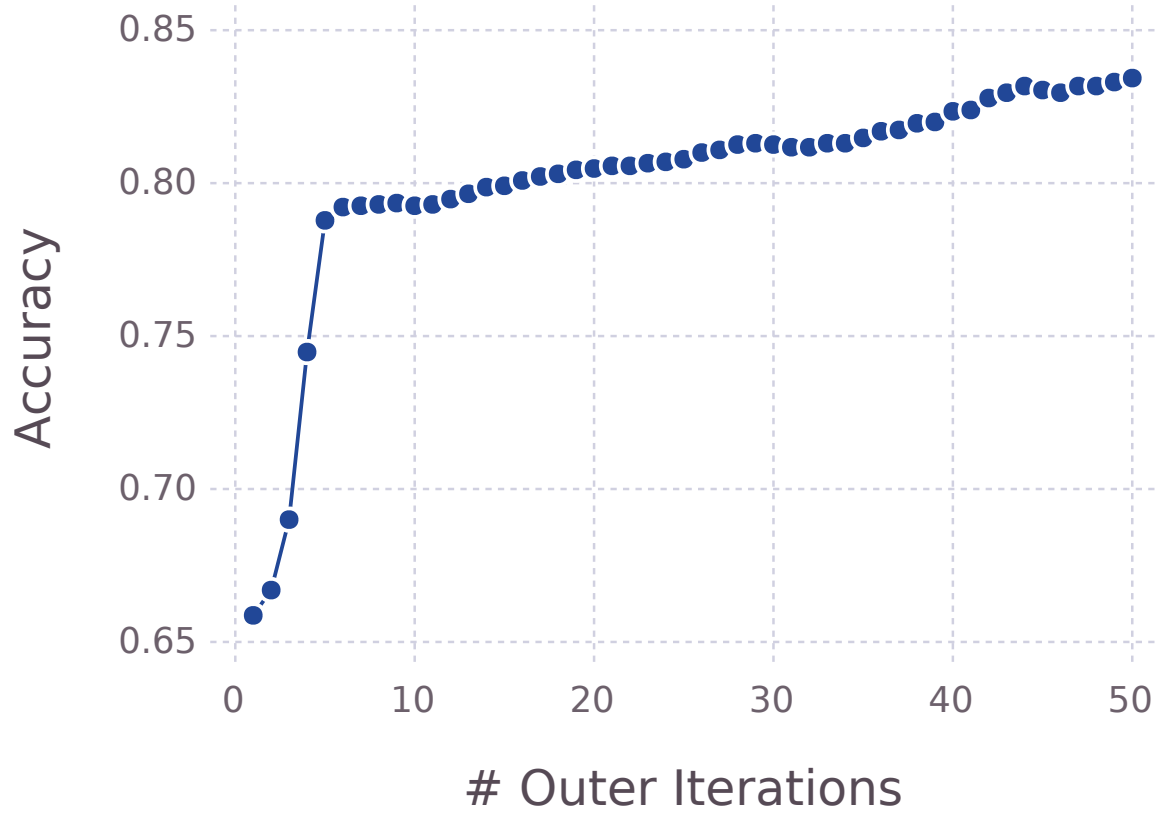
- 4600 data points (1813 Spam)
- 57 attributes + labels
- 48 attributes are frequencies of specific words
- Others include length of longest sequence of capital letters and total number of capital letter

Experiments: preprocessing

- Randomly permutate the rows of the data set
- Split into attributes and labels
- Convert labels to $\{-1, 1\}$ format
- Add a column of ones to the data
- Split the dataset equally into training and testing set



Accuracy when number of outer iterations is increased (number of inner iterations: 5)
Comparison: LIBSVM with same C and default values otherwise gives accuracy 84.22%



Accuracy when number of outer iterations is increased (number of inner iterations: 1)