Alternating Direction Method of Multipliers

CS 584: Big Data Analytics

Material adapted from
Stephen Boyd (https://web.stanford.edu/~boyd/papers/pdf/admm_slides.pdf) &
Ryan Tibshirani (http://stat.cmu.edu/~ryantibs/convexopt/lectures/21-dual-meth.pdf)

Goals of ADMM

- Arbitrary-scale optimization
  - Machine learning / statistics with huge data-sets
  - Dynamic optimization on large-scale network
- Decentralized optimization
  - Use many machines to solve large problem by passing relatively small messages

Dual Decomposition Review

- Convex equality constrained problem

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & Ax = b \end{aligned}$$

- Construct the Lagrangian

$$L(x, y) = f(x) + y^\top (Ax - b)$$

- Solve the dual problem for optimal y*

$$\max g(y) = \inf_x L(x, y)$$

- Recover the optimal point $x^* = \text{argmin}_x L(x, y^*)$

Dual ascent

- Gradient method for dual problem

$$y^{k+1} = y^k + \eta^k \nabla g(y^k)$$

- Dual ascent method:

$$x^{k+1} = \text{argmin}_x L(x, y^k)$$
$$y^{k+1} = y^k + \eta_k (Ax^{k+1} - b)$$

Dual Decomposition: Separability

- Suppose the objective function is separable (x can be divided into N blocks of variables)

$$f(x) = \sum_{i=1}^{N} f_i(x_i)$$

- Then the Lagrangian is separable in x

$$L_i(x_i, y) = f_i(x_i) + y^\top A_i x_i$$

- Then we can do dual ascent and have N separate minimizations carried out in parallel

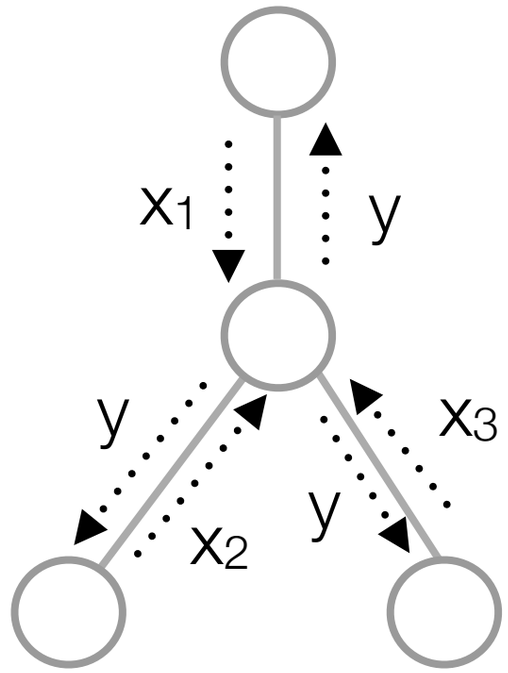
Dual Decomposition: Separability

- Dual decomposition algorithm:

$$x_i^{k+1} = \text{argmin}_{x_i} f_i(x_i) + (y^k)^\top A_i x_i$$

$$y^{k+1} = y^k + \eta_k(\sum_{i=1}^{N} A_i x_i^k - B)$$

- Can think of this as a two step process
  - Broadcast: send y to each of the N processors, each optimizes in parallel to find x_i
  - Gather: collect $A_i x_i$ from each processor, and update the global variable y



Dual Methods Properties

- (Pro) Decomposability and separability
- (Con) Require strong conditions to ensure primal iterates converge to solutions
- (Con) Can be slow to converge

Augmented Lagrangian

- A modification to the Lagrangian or a shifted quadratic penalty function (rho > 0)

$$L(x, y, \rho) = f(x) + y^\top (Ax - b) + \frac{\rho}{2} ||Ax - b||_2^2$$

- Adjust y and rho to encourage convergence (and feasibility of the iterates)

- Method of multipliers to solve:

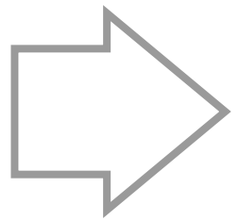
$$x^{k+1} = \text{argmin}_x L_\rho(x, y^k)$$

$$y^{k+1} = y^k + \rho(Ax^{k+1} - b)$$

Method of Multipliers Properties

- Converges under more relaxed conditions
- Unconstrained minimizer of the augmented Lagrangian coincides with constrained solution of original problem
- Bad news: quadratic penalty destroys splitting of the x-update so you can't do decomposition

$$\begin{aligned} \min \;\; & f(x) + g(y) \\ \text{s.t. } & Ax + By = c \end{aligned}$$



$$\begin{aligned} L(x, y, \lambda, \rho) = & f(x) + g(y) + \\ & \lambda^\top (Ax + By - c) + \\ & \frac{\rho}{2} ||Ax + By - c||_2^2 \end{aligned}$$

Alternating Direction Method of Multipliers (ADMM)

- Proposed by Gabay, Mercier, Glowinski, Marrocco in 1976
- Largely lost favor as an approach until recent revival
- Good robustness of method of multipliers and can support decomposition
- “Robust dual decomposition” or “decomposable method of multipliers”
- Best of both worlds (separability + better convergence)!

ADMM Formulation

- Set of variables that have separable objective

$$\begin{aligned} \min \;\; & f(x) + g(z) \\ \text{s.t.} \;\; & Ax + Bz = c \end{aligned}$$

- Construct the Augmented Lagrangian

$$L_\rho(x, z, y) = f(x) + g(z) + y^\top (Ax + Bz - c) + \frac{\rho}{2} ||Ax + Bz - c||_2^2$$

- Instead of doing the standard method of multipliers, we solve for each variable separately and sequentially

ADMM Algorithm

$$x^{k+1} = \text{argmin}_x L_\rho(x, z^k, y^k) \quad \text{// x-minimization}$$

$$z^{k+1} = \text{argmin}_z L_\rho(x^{k+1}, z, y^k) \quad \text{// z-minimization}$$

$$y^{k+1} = y^k + \rho(Ax^{k+1} + Bz^{k+1} - c) \quad \text{// dual update}$$

ADMM Properties

- Each iteration does a round of block-coordinate descent in (x,z)

- Minimizations over x and z only add a quadratic term to f and h, so doesn't alter cost much
    - Can be performed inexactly

- Embraces distributed computing for big data

- Convergence is often slow, but sufficient for many applications

Practicalities and Tricks

- ADMM usually obtains a relatively accurate solution in a handful of iterations, but requires a very large number of iterations for a highly accurate solution

- Choice of ρ can greatly influence convergence
  - Too large —> not enough emphasis on minimizing objective
  - Too small —> not enough emphasis on feasibility

- Boyd offers a strategy for varying ρ that can be useful in practice (but does not have convergence guarantees)

Example: LASSO

- Original problem

$$\min \frac{1}{2}||Ax - b||_2^2 + \lambda||x||_1$$

- ADMM form

$$\min \ \frac{1}{2}||Ax - b||_2^2 + \lambda||z||_1$$
$$\text{s.t. } x - z = 0$$

- ADMM updates

$$x^{k+1} = (A^\top A + \rho I)^{-1}(A^\top b + \rho(z^k - y^k))$$
$$z_j^{k+1} = S_{\lambda/\rho}(x^{k+1} + y^k)$$
$$y^{k+1} = y^k + x^{k+1} - z^{k+1}$$

Example: Lasso Results

- Dense A with 1500 measurements and 5000 regressors
- Computation times

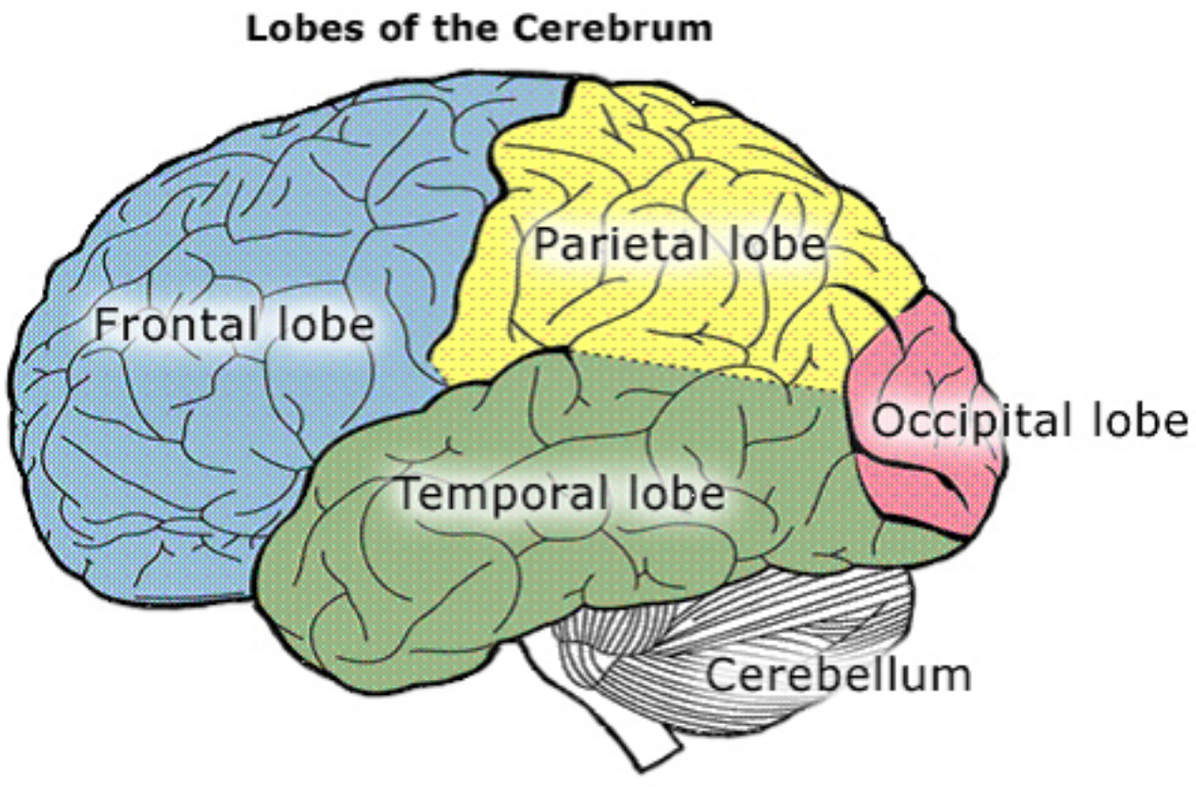
Factorization $(A^\top A + \rho I)^{-1}$	1.3s
Subsequent ADMM iterations	0.03 s
Lasso solve (~ 50 ADDM iterations)	2.9 s
Full regularization path (30 lambdas)	4.4s

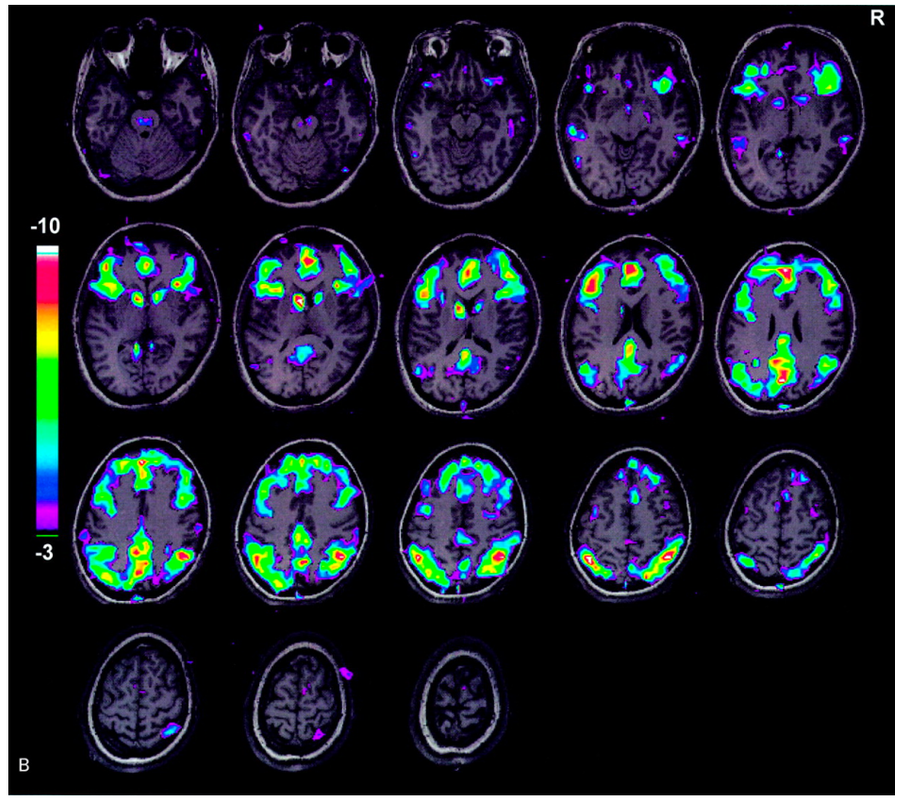
- Not bad for a very short Matlab script

Example: Group LASSO

- Extension of LASSO for variable selection on groups of variables in regression models

- Motivated by real-world examples where certain groups of variables are jointly activated





Example: Group LASSO (2)

- Optimization problem with J pre-defined groups

$$\min \frac{1}{2}||Ax - b||_2^2 + \lambda \sum_{j=1}^{J} c_j ||x_j||_2$$

- ADMM form:

$$\min \frac{1}{2}||Ax - b||_2^2 + \lambda \sum_{j=1}^{J} c_j ||z_j||_2$$

$$\text{s.t. } x - z = 0$$

- ADMM updates:

$$x^{k+1} = (A^\top A + \rho I)^{-1}(A^\top b + \rho(z^k - y^k))$$

$$z_j^{k+1} = R_{c_j \lambda/\rho}(x^{k+1} + y^k)$$

$$y^{k+1} = y^k + x^{k+1} - z^{k+1}$$

Example: Group LASSO (3)

- Main difference between Lasso and Group Lasso is the z update replaces the block soft thresholding with a vector soft thresholding operation

$$R_\kappa(a) = \left(1 - \frac{\kappa}{||a||_2}\right)_+ a$$

- ADMM algorithm can be developed for the case of overlapping groups (which is otherwise quite a hard problem to optimize!)

Consensus Optimization

- Solve a problem with N objective terms (e.g., the loss function for the ith block of training data)

$$\min \sum_{i=1}^{N} f_i(x)$$

- ADMM form:

$$\min \sum_{i=1}^{N} f_i(x)$$

$$\text{s.t. } x_i - z = 0$$

  - x_i are the local variables
  - z is the global variable
  - $x_i - z$ is the consistency or consensus constraints

Consensus Optimization Algorithm

- ADMM update:

$$x_i^{k+1} = \text{argmin}(f_i(x_i) + (y_i^k)^\top (x_i - z^k) + \rho/2||x_i - z^k||_2^2)$$

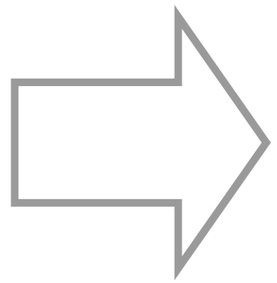
$$z^{k+1} = \frac{1}{N}\sum_{i=1}^{N}(x_i^{k+1} + (1/\rho)y_i^k)$$

$$y_i^{k+1} = y_i^k + \rho(x_i^{k+1} - z^{k+1})$$

- Note that z update is an average

$$z^{k+1} = \bar{x}^{k+1} + (1/\rho)\bar{y}^{k+1}$$

$$\bar{y}^{k+1} = \bar{y}^k + \rho(\bar{x}^{k+1} - z^{k+1})$$



$$\bar{y}^{k+1} = 0$$

Consensus Optimization Algorithm (2)

- Actual ADMM update:

$$\bar{x}^k = \frac{1}{N}\sum_{i=1}^{N} x_i^k$$

$$x_i^{k+1} = \text{argmin}(f_i(x_i) + (y_i^k)^\top (x_i - \bar{x}^k) + \rho/2||x_i - \bar{x}^k||_2^2)$$

$$y_i^{k+1} = y_i^k + \rho(x_i^{k+1} - \bar{x}^{k+1})$$

- For each iteration
  - Gather x_i and average the values
  - Scatter the average value to each processor
  - Update y_i locally
  - Update x_i locally

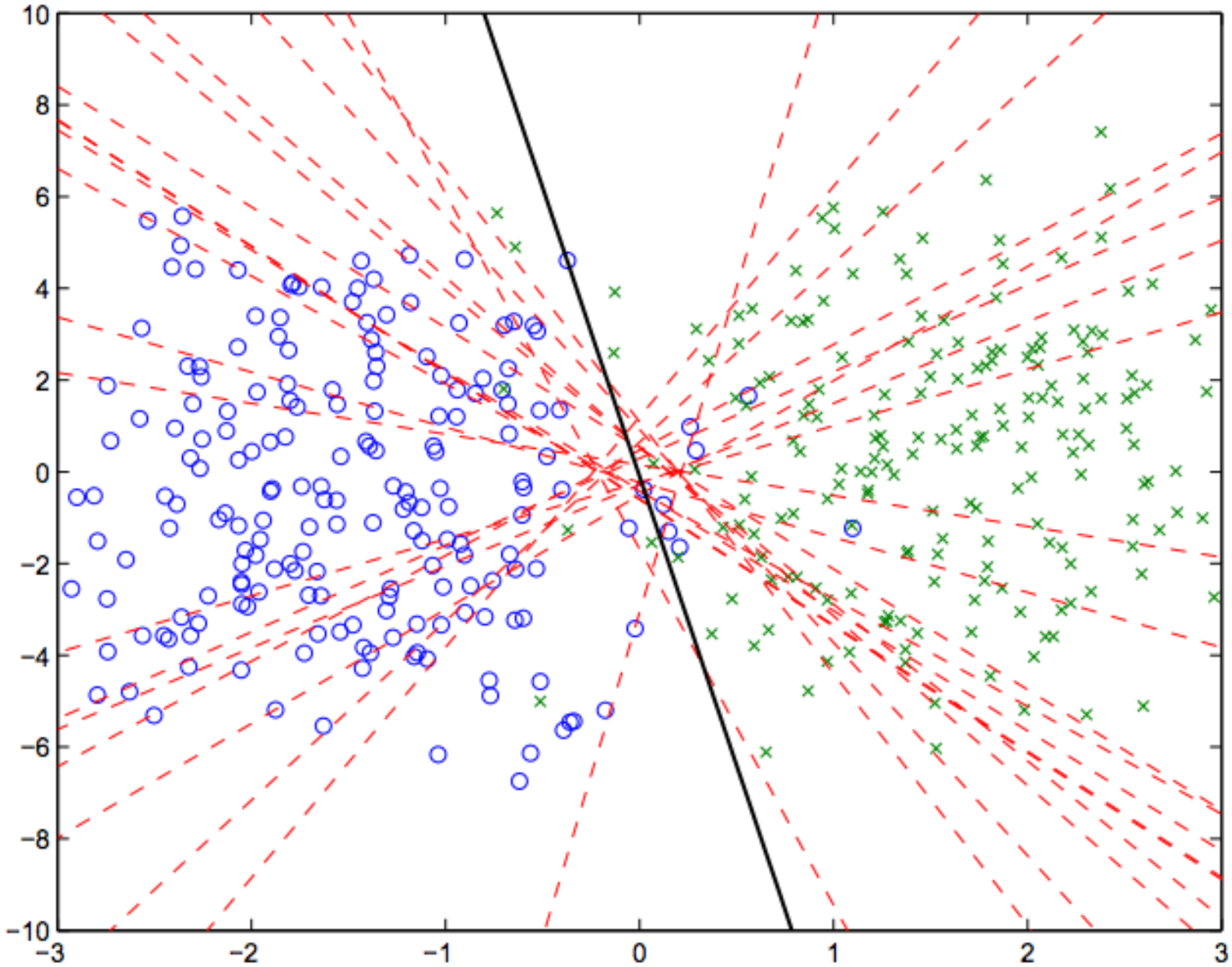
Example: Consensus SVM

- Data (x_i, y_i) where x_i are the features and y_i is the label
- Linear SVM with weight and offset: $\mathbf{sign}(a^\top w + v)$
- Hinge loss with ridge regression

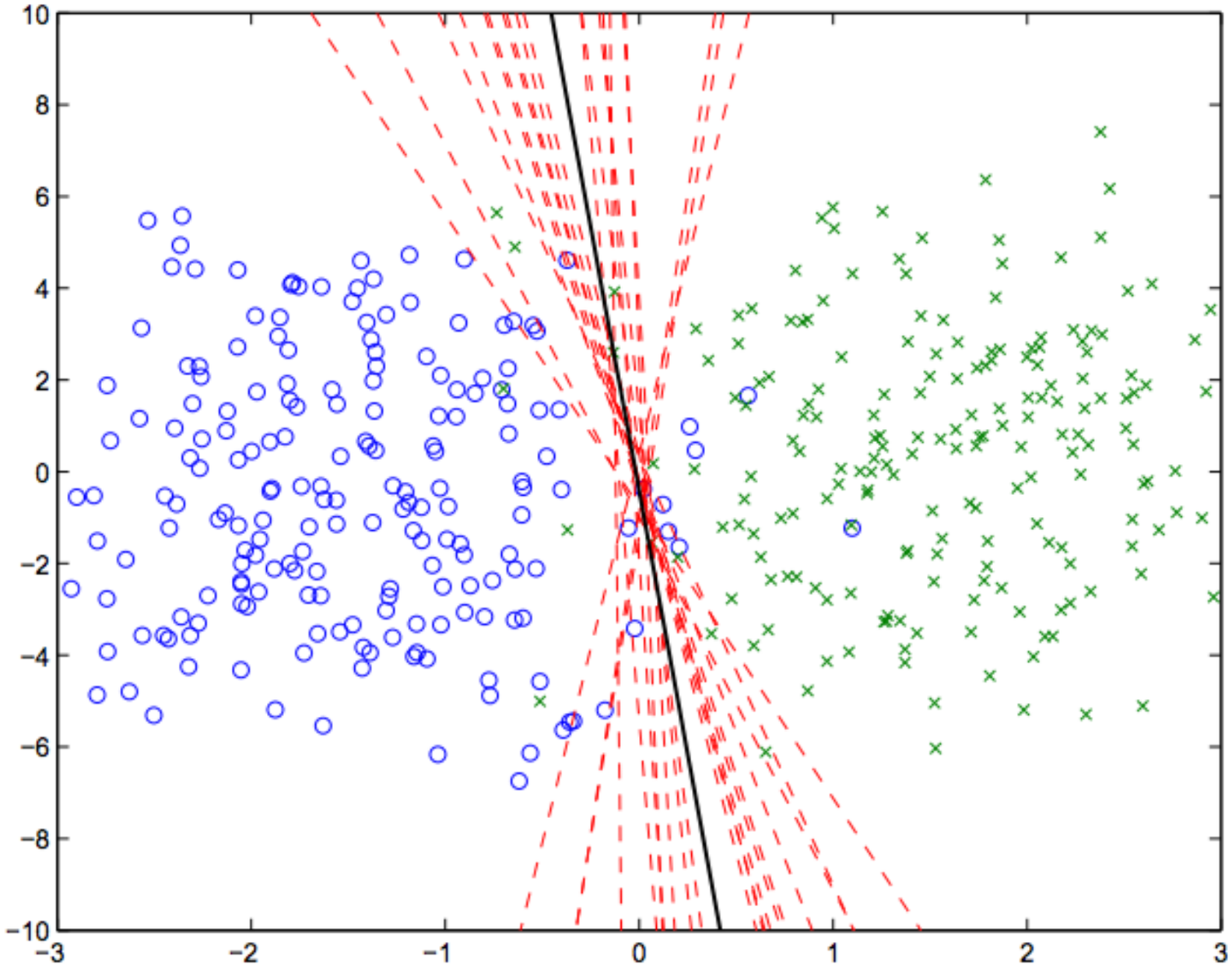
$$(1 - y_i(x_i^\top w + v))_+ + \beta||w||_2$$

- Baby problem with d = 2, and N = 400
- Examples split into 20 groups such that each group contains only positive or negative examples

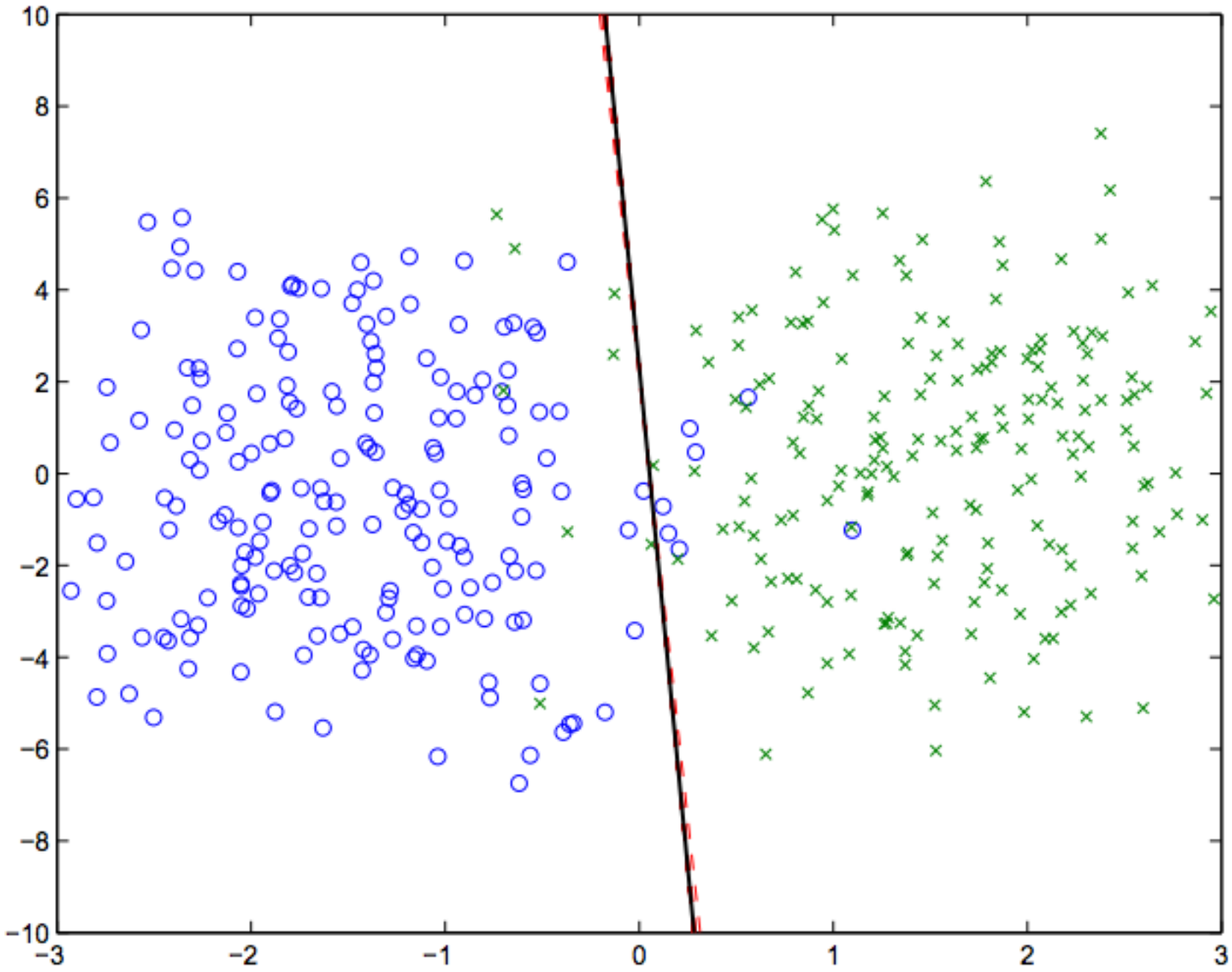
Example: Consensus SVM Iteration 1



Example: Consensus SVM Iteration 5



Example: Consensus SVM Iteration 40



Distributed Model Fitting

- General Fitting Problem with additive loss:

$$\min \sum_{i=1}^{m} l_i(a_i^\top x - b_i) + r(x)$$

- Two methods of distribution
  - Split across samples (consensus)
  - Split across features

Distributed Model Across Data

- Consider this when you have a modest number of features but a large number of training examples

- Examples: Social network data, wireless network sensors, and many cloud computing applications

- Partition your data based on the rows

$$A = \begin{bmatrix} A_1 \\ \vdots \\ A_N \end{bmatrix}, b = \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix}$$

Distributed Model Across Data Update

- Solve using the consensus ADMM algorithm

$$x_i^{k+1} := \arg\min_{x_i}(l_i(A_i x_i - b_i) + \frac{\rho}{2}\|x_i - z^k + u_i^k\|_2^2)$$

$$z^{k+1} := \arg\min_{z}(r(z) + \frac{N\rho}{2}\|z - \overline{x}^{k+1} - \overline{u}^k\|_2^2)$$

$$u_i^{k+1} := u_i^k + x_i^{k+1} - z^{k+1}$$

- For each iteration
  - Carry out regularization model fitting on each data block
  - Gathering variables to form the average
  - Update the dual variable to reflect the gap between local and global

Example: Distributed LASSO (by Samples)

- ADMM update:

$$x_i^{k+1} := (A_i^T A_i + \rho I)^{-1}(A_i^T b_i + \rho(z^k - u_i^k))$$
$$z^{k+1} := S_{\lambda/\rho N}(\overline{x}^{k+1} + \overline{u}^k)$$
$$u_i^{k+1} := u_i^k + x_i^{k+1} - z^{k+1}$$

- Very similar to non-distributed LASSO except the z-update collects and averages the computations for the different data blocks

Example: Distributed Lasso Results

- Dense A with 400000 measurements and 8000 regressors (~30 GB)
  - No optimization or tuned libraries (written in C)
  - Split using 80 subsystems across 10 (8-core) machines on Amazon EC2
  - Each subsystem has 5000 samples
- Computation times

Loading Data	30 s
Factorization	5m
Subsequent ADMM iterations	0.5 - 2s
Lasso solve (~ 15 ADDM iterations)	5-6 m
Total runtime	6 m

Distributed Model Across Features

- Consider this when you have a modest number of trying examples but a large number of features

- Examples: natural language processing, bioinformatics

- Partition the data based on features

$$x = \begin{pmatrix} x_1 & \dots & x_N \end{pmatrix}$$

$$A = \begin{bmatrix} A_1 & \dots & A_N \end{bmatrix}$$

$$r(x) = \sum_{i=1}^{N} r_i x_i$$

Distributed Model Across Features Update

- Solve using the sharing ADMM algorithm (dual to consensus)

$$x_i^{k+1} := \arg\min_{x_i}(r_i(x_i) + \frac{\rho}{2}\|A_i x_i - A_i x_i^k - \bar{z}^k + \overline{Ax}^k + u_k\|_2^2)$$

$$\bar{z}^{k+1} := \arg\min_{\bar{z}}(l(N\bar{z} - b) + \frac{N\rho}{2}\|\bar{z} - \overline{Ax}^{k+1} - u^k\|_2^2)$$

$$u^{k+1} := u^k + \overline{Ax}^{k+1} - \bar{z}^{k+1}$$

- For each iteration
  - Solve regularized least square problem for each feature block
  - Collect and sum the partial predictors to perform quadratically regularized loss minimization problem
  - Simple update of the dual variable

ADMM Iteration in Hadoop/MapReduce

- Easily represented using MapReduce Task
- Parallel local computations performed by maps
- Global aggregation performed by Reduce

Algorithm 2 An iteration of global consensus ADMM in Hadoop/ MapReduce.

function map(key i, dataset $\mathcal{D}_i$)

1. Read $(x_i, u_i, \hat{z})$ from HBase table.
2. Compute $z := \mathbf{prox}_{g,N\rho}((1/N)\hat{z})$.
3. Update $u_i := u_i + x_i - z$.
4. Update $x_i := \mathrm{argmin}_x \left(f_i(x) + (\rho/2)\|x - z + u_i\|_2^2\right)$.
5. *Emit* (key CENTRAL, record (x_i, u_i)).

function reduce(key CENTRAL, records $(x_1, u_1), \ldots, (x_N, u_N)$)

1. Update $\hat{z} := \sum_{i=1}^{N} x_i + u_i$.
2. *Emit* (key j, record $(x_j, u_j, \hat{z})$) to HBase for $j = 1, \ldots, N$.

ADDM Summary and Conclusions

- Has been around since the 190s and is the same or closely related to many methods with other names

- Gives simple single-processor algorithms that can be competitive with state-of-the-art algorithms

- Can be used to coordinate many processors, each solving a substantial problem, to solve a very large problem