A Tour of Proximal Algorithms

Neal Parikh

Cornell University

March 1, 2018

Motivation

- consider the generic convex optimization problem

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & x \in \mathcal{C} \end{array}$$

  with $x \in \mathbf{R}^n$

- possible 'classical' approaches, depending on f, $\mathcal{C}$, and n:
  - unconstrained: gradient method, Newton method, BFGS
  - constrained: projected gradient, primal-dual interior-point method

- potential issues:
  - applies only to unconstrained problems
  - assumes that objective is smooth
  - requires a problem transformation that obscures problem structure
  - does not scale beyond medium size problems
  - does not easily support distributed data or parallel computation

Example: ℓ_1 regularization

consider the (very common) problem

$$\text{minimize} \quad f(x) + \lambda \|x\|_1$$

options:

- generic subgradient method
- use a transformation like

$$\begin{array}{ll} \text{minimize} & s + \lambda 1^T t \\ \text{subject to} & f(x) \leq s \\ & x_i \leq t_i, \quad i = 1, \ldots, n \\ & -x_i \leq t_i, \quad i = 1, \ldots, n \end{array}$$

  and then attempt to use a symmetric cone solver

  – won't work if, *e.g.*, f involves exp or log
  – obscures structure, *e.g.*, solution is only approximately sparse

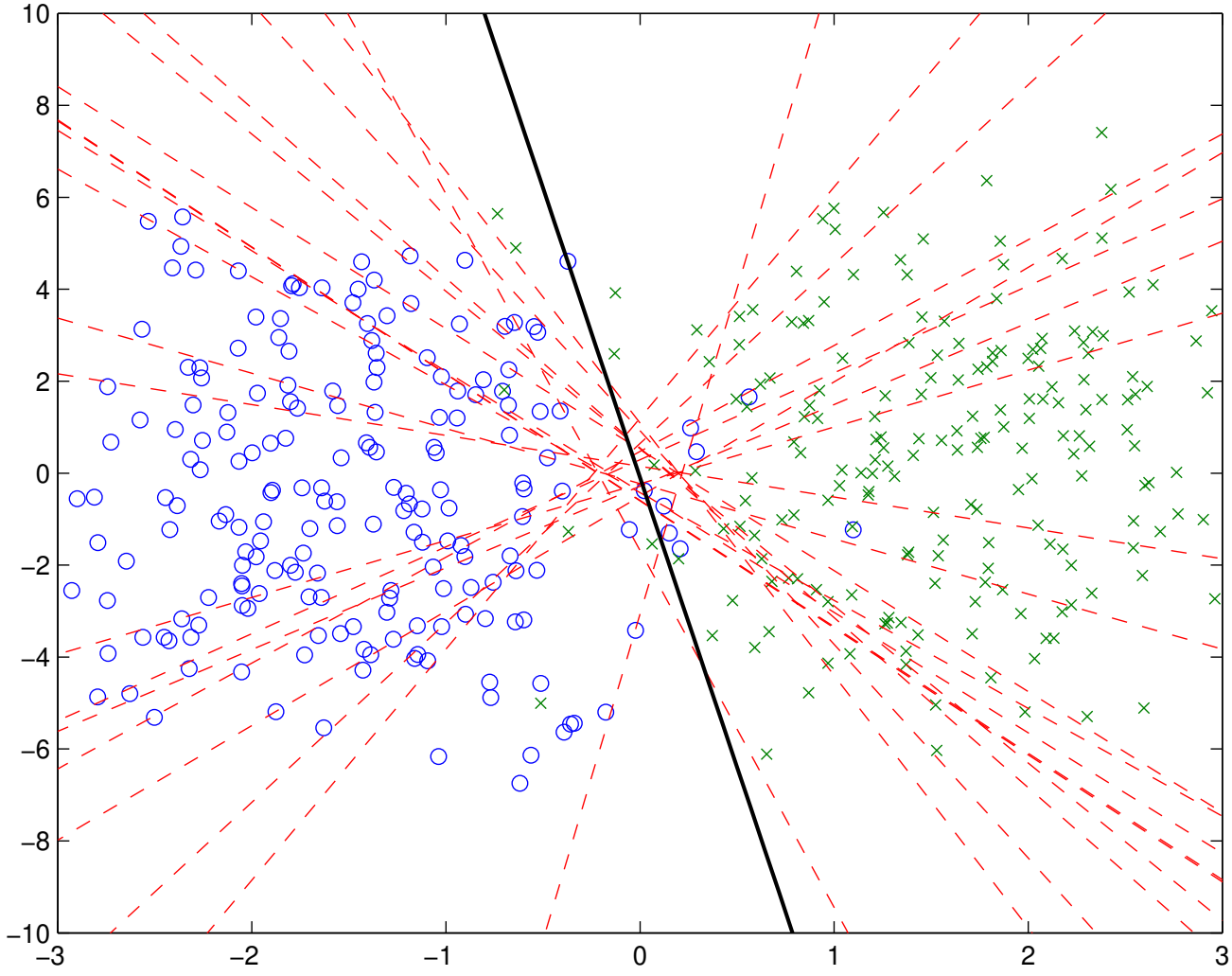
Goals

simple, general-purpose, non-heuristic methods for

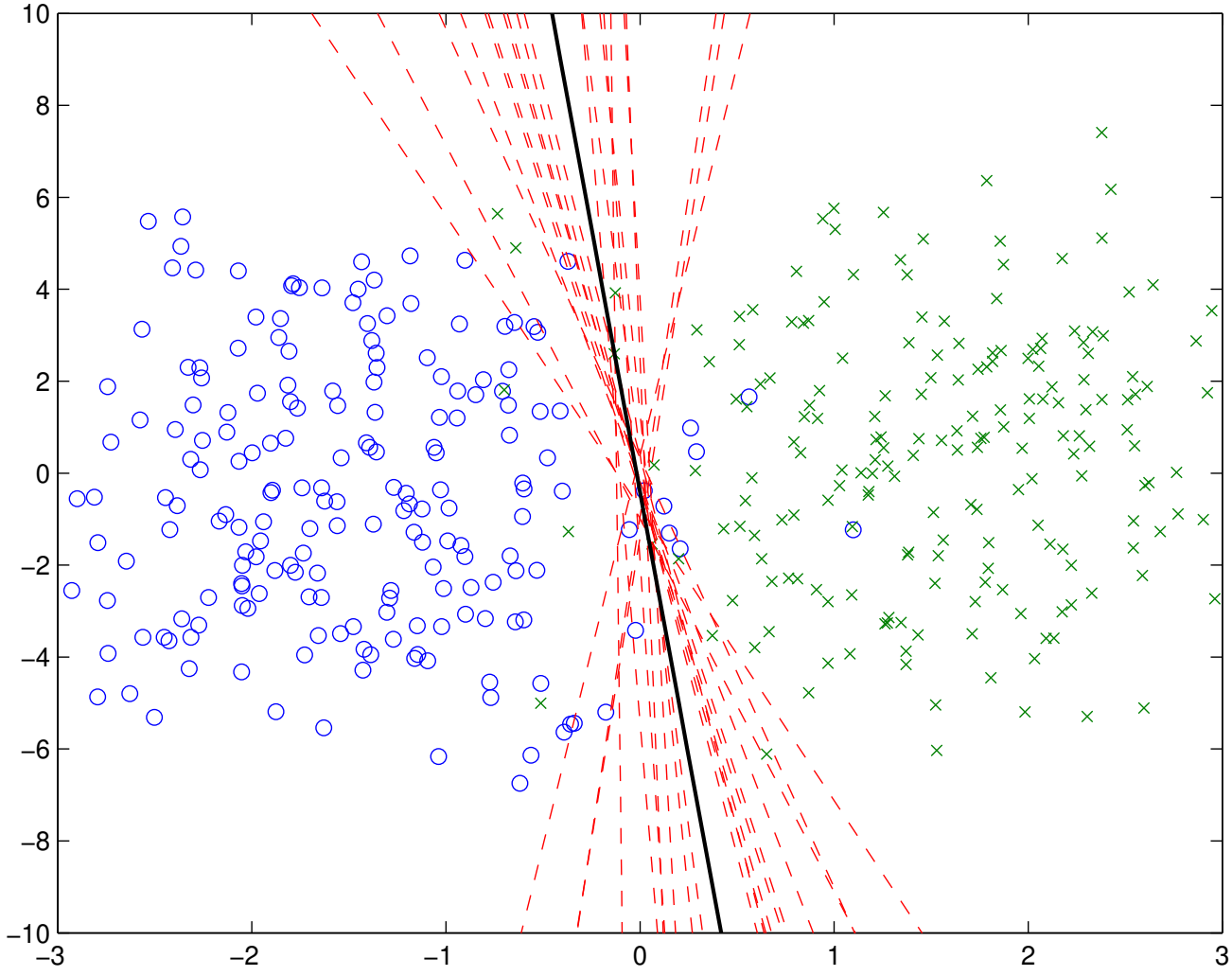
- arbitrary-scale optimization (like learning/statistics with huge datasets)
- use problem structure to decompose problems into smaller/simpler pieces

want to obtain an **exact, global solution** to original convex problem

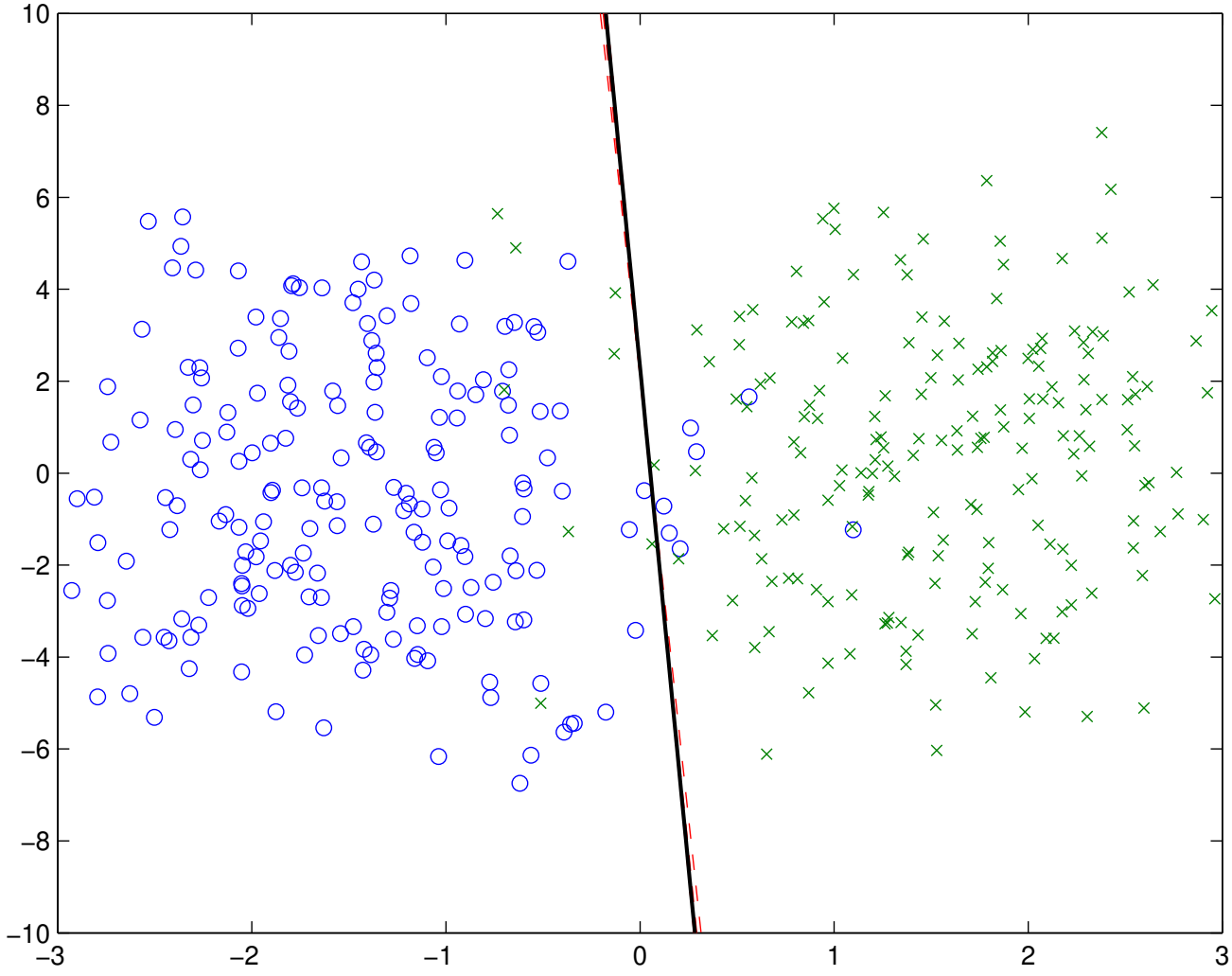
Distributed SVM: Iteration 1



Distributed SVM: Iteration 5



Distributed SVM: Iteration 40



This talk

interaction between three key ingredients:

1. proximal operator of a convex function
2. operator splitting algorithms
3. problem transformations

Structure and Regularization

Outline

ℓ_1 regularization

Examples and extensions

Proximal operators

Proximal algorithms

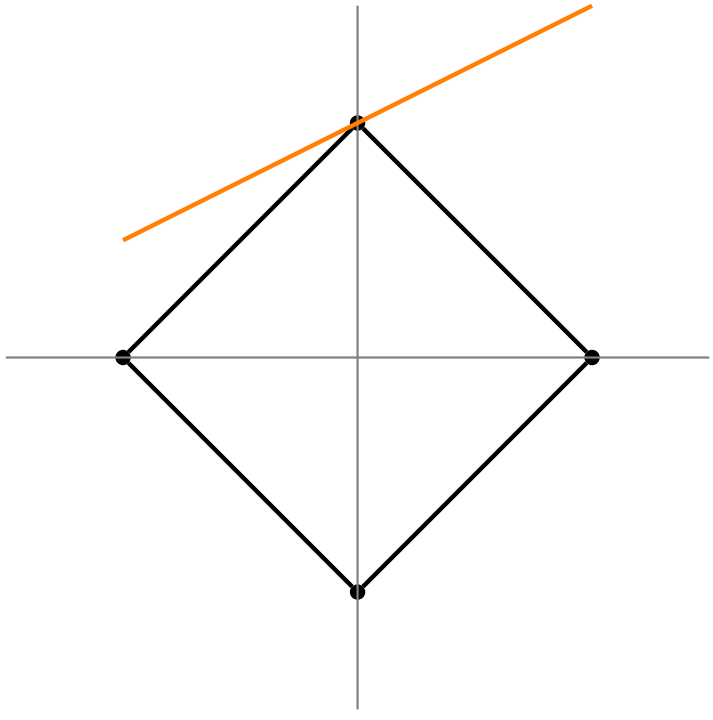
Structure in variables

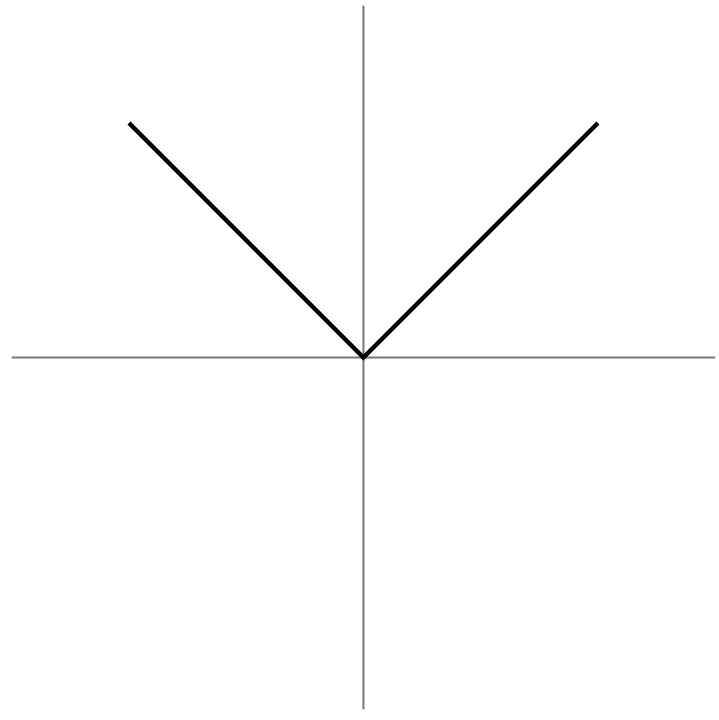
- often know or assume that solution to a problem is structured, *e.g.*,
  - convex-cardinality problems
  - high-dimensional statistics: assume low-dimensional structure
  - prior knowledge that variables have, *e.g.*, hierarchical or grouped structure
- handle by solving a problem with two conceptual components:
  - main objective of interest (model fit, satisfying constraints, . . .)
  - regularization term that encourages assumed form of structure
- possible structure of interest includes sparsity, low rank, . . .

this talk:

1. selecting regularization to promote assumed structure
2. many examples and applications (*i.e.*, sparsify everything in sight)
3. solving the resulting optimization problems

Geometric interpretation



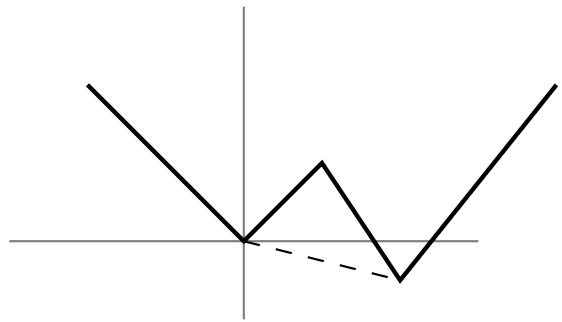


get sparsity/structure when corners/kinks appear at sparse/structured points

e.g., quadratic cone, linear functions on prob. simplex, nuclear norm, ...

Convex envelope interpretation

- convex envelope of (nonconvex) f is the largest convex underestimator g
- *i.e.*, the best convex lower bound to a function



- **example**: ℓ_1 is the envelope of $\mathbf{card}$ (on unit ℓ_∞ ball)
- **example**: $\|\cdot\|_*$ is the envelope of $\mathbf{rank}$ (on unit spectral norm ball)
- various characterizations: *e.g.*, f^{**} or convex hull of epigraph

Penalty function interpretation

- compared to ridge penalty $\|\cdot\|_2^2$, using ℓ_1 does two things:
  1. higher emphasis on small values to go to exactly zero
  2. lower emphasis on avoiding very large values
- thus useful for obtaining **sparse** or **robust** solutions to problems

Atomic norm interpretation

(Chandrasekaran, Recht, Parrilo, Willsky)

- convex surrogates for measures of 'simplicity'
- suppose underlying parameter vector or signal $x \in \mathbf{R}^n$ given by

$$x = \sum_{i=1}^{k} c_i a_i, \quad a_i \in \mathcal{A},\ c_i \geq 0,$$

  where $\mathcal{A}$ is set of 'atoms' and $k \ll n$ (d.f. $\ll$ ambient dimension)

- if $\mathcal{A}$ is usual basis vectors, model says that x is k-sparse, and

$$\mathbf{conv}(\mathcal{A}) = \text{unit } \ell_1 \text{ ball}$$

- then, *e.g.*, minimize $\|x\|_1$ subject to $y = Fx$

Outline

Sparse design

- find sparse design vector x satisfying specifications

$$\begin{array}{ll} \text{minimize} & \|x\|_1 \\ \text{subject to} & x \in \mathcal{C} \end{array}$$

- zero values of x simplify design or correspond to unneeded components
- when $\mathcal{C} = \{x \mid Ax = b\}$, called **basis pursuit** or **sparse coding**
- *e.g.*, find sparse representation of signal b in 'dictionary' or 'overcomplete basis' given by columns of A

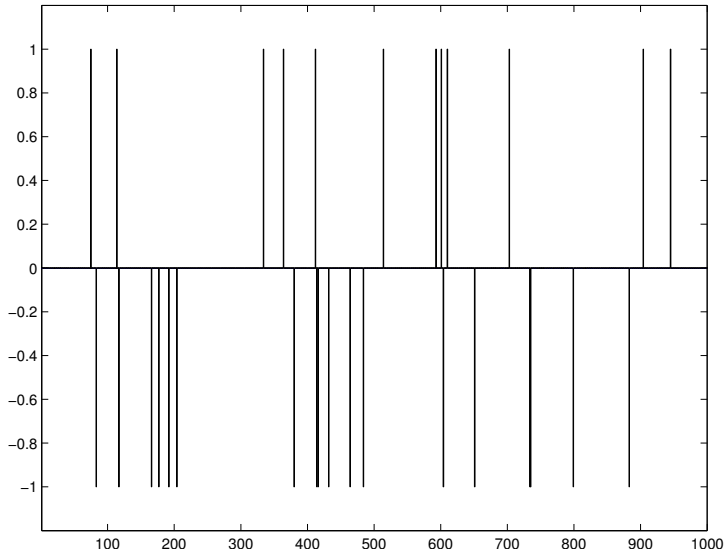
Sparse regression

- fit $b \in \mathbf{R}^m$ as linear combination of a subset of regressors

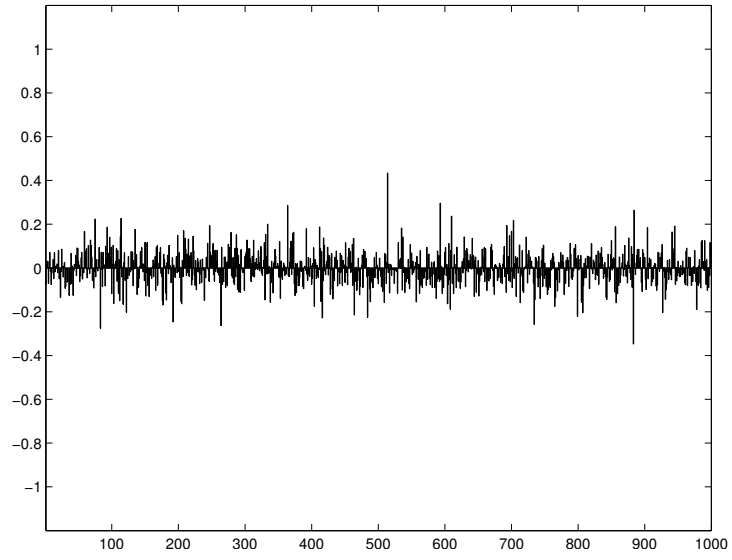
$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \lambda\|x\|_1$$

- zero values of x indicate features not predictive of the response
- also known as the lasso
- easily generalizes to other losses (*e.g.*, sparse logistic regression)

Sparse regression

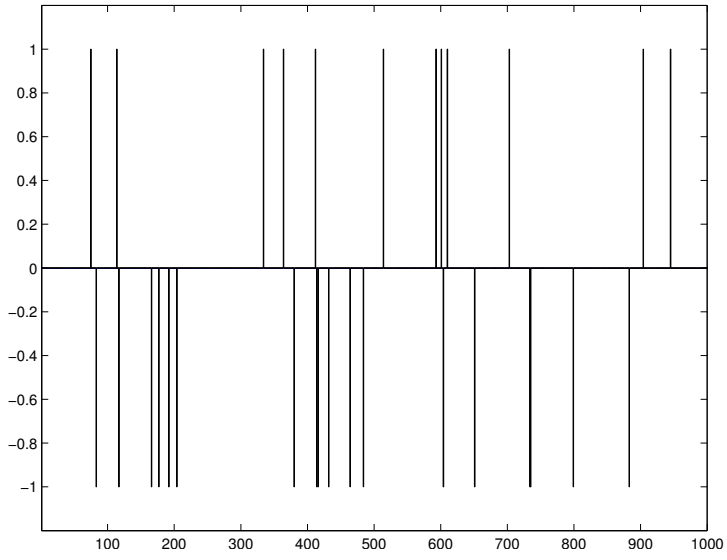


original

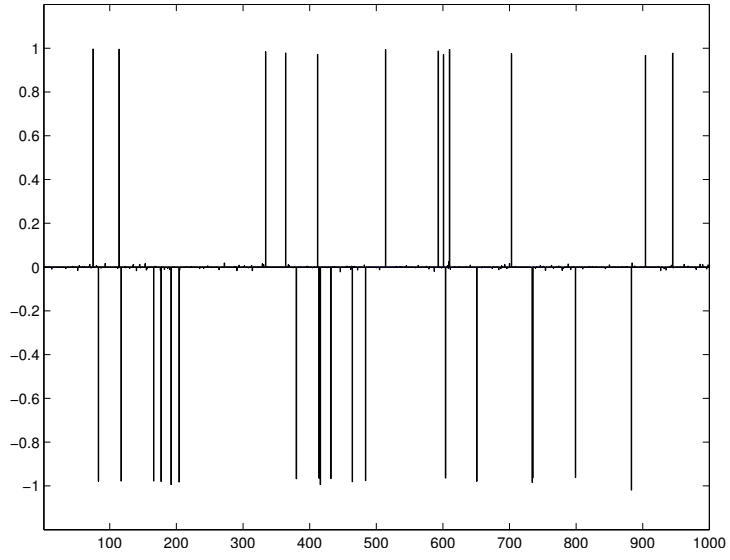


quadratic regularization

Sparse regression



original



ℓ_1 norm

Estimation with outliers

- measurements $y_i = a_i^T x + v_i + w_i$
- v_i is Gaussian noise (small), w is a sparse outlier vector (big)
- if $\mathcal{O} = \{i \mid w_i \neq 0\}$ is set of outliers, MLE given by

$$\begin{array}{ll} \text{minimize} & \sum_{i \notin \mathcal{O}} (y_i - a_i^T x)^2 \\ \text{subject to} & |\mathcal{O}| \leq k \end{array}$$

- convex approximation given by

$$\text{minimize} \quad (1/2)\|y - Ax - w\|_2^2 + \lambda \|w\|_1$$

- same idea used in support vector machine

Linear classifier with fewest errors

- want linear classifier $b \approx \mathbf{sign}(a^T x + s)$ from $(a_i, b_i) \in \mathbf{R}^n \times \{-1, 1\}$
- error corresponds to negative margin: $b_i(a_i^T x + s) \leq 0$
- find x, s that give fewest classification errors:

$$\begin{array}{ll} \text{minimize} & \|t\|_1 \\ \text{subject to} & b_i(a_i^T x + s) + t_i \geq 1, \quad i = 1, \ldots m \end{array}$$

  with variables x, s, t

- close to a support vector machine
- can generalize to other convex feasibility problems

Elastic net

(Zou & Hastie)

- problem:

$$\text{minimize} \quad f(x) + \lambda \|x\|_1 + (1-\lambda)\|x\|_2^2$$

  i.e., use both ridge and lasso penalties

- attempts to overcome the following potential drawbacks of the lasso:
  - lasso selects at most (# examples) variables
  - given group of very correlated features, lasso often picks one arbitrarily

- here, strongly correlated predictors are jointly included or not

- (in practice, need to do some rescaling above)

Fused lasso

(Tibshirani et al.; Rudin, Osher, Fatemi)

- problem:

$$\text{minimize} \quad f(x) + \lambda_1 \|x\|_1 + \lambda_2 \sum_{j=2}^{n} |x_j - x_{j-1}|$$

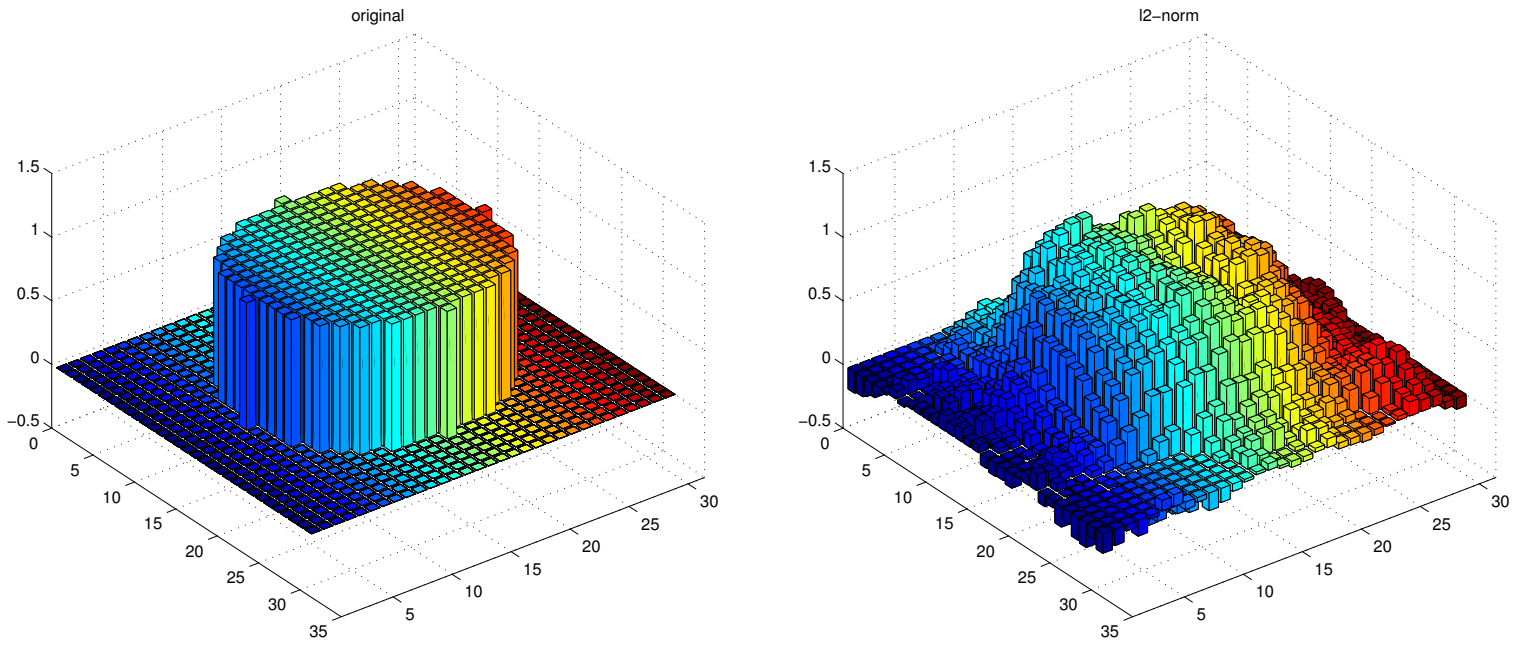
  i.e., encourage x to be both sparse and piecewise constant

- special case: **total variation denoising** (set $\lambda_1 = 0$)

- used in biology (*e.g.*, gene expression) and signal reconstruction

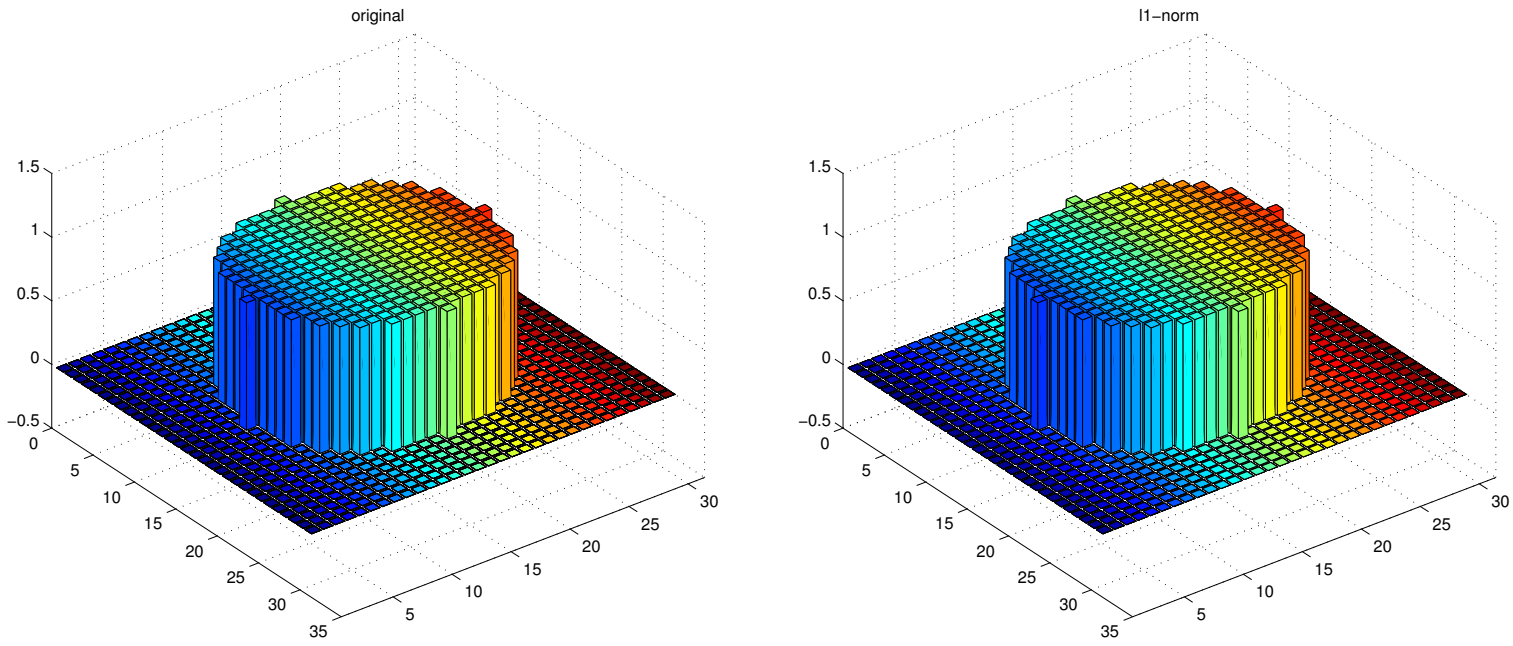
- can also write penalty as $\|Dx\|_1$; could consider other matrices

Total variation denoising



120 linear measurements and $31 \times 31 = 961$ variables ('8x undersampled')

Total variation denoising



120 linear measurements and $31 \times 31 = 961$ variables ('8x undersampled')

Group lasso

(*e.g.*, Yuan & Lin; Meier, van de Geer, Bühlmann; Jacob, Obozinski, Vert)

- problem:

$$\text{minimize} \quad f(x) + \lambda \sum_{i=1}^{N} \|x_i\|_2$$

  i.e., like lasso, but require groups of variables to be zero or not

- also called $\ell_{1,2}$ mixed norm regularization

- related to **multiple kernel learning** via duality (see Bach et al.)

Joint covariate selection for multi-task learning

(Obozinski, Taskar, Jordan)

- want to fit parameters $x^k \in \mathbf{R}^p$ for each of **multiple** datasets $\mathcal{D}^k$
- either use feature j in all tasks or none of them
- let $x_j = (x_j^1, \ldots, x_j^K)$ for $j = 1, \ldots, p$
- problem:

$$\text{minimize} \quad \sum_{k=1}^K f^k(x^k) + \lambda \sum_{j=1}^p \|x_j\|_2$$

  with variables $x^1, \ldots, x^K \in \mathbf{R}^p$

Structured group lasso

(Jacob, Obozinski, Vert; Bach et al.; Zhao, Rocha, Yu; . . .)

- problem:

$$\text{minimize} \quad f(x) + \sum_{i=1}^{N} \lambda_i \|x_{g_i}\|_2$$

  where $g_i \subseteq [n]$ and $\mathcal{G} = \{g_1, \ldots, g_N\}$

- like group lasso, but the groups can overlap arbitrarily

- particular choices of groups can impose 'structured' sparsity

- *e.g.*, topic models, selecting interaction terms for (graphical) models, tree structure of gene networks, fMRI data

- generalizes to the **composite absolute penalties family**:

$$r(x) = \|(\|x_{g_1}\|_{p_1}, \ldots, \|x_{g_N}\|_{p_N})\|_{p_0}$$

Structured group lasso

(Jacob, Obozinski, Vert; Bach et al.; Zhao, Rocha, Yu; . . .)

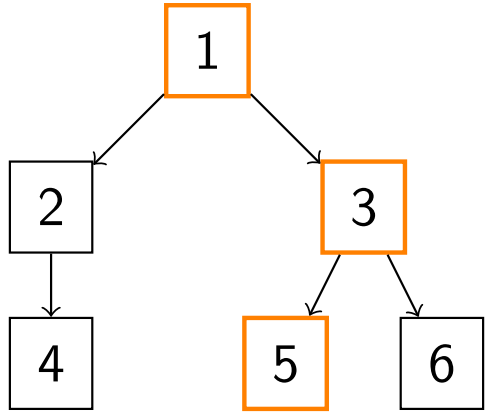
contiguous selection:

- $\mathcal{G} = \{\{1\}, \{5\}, \{1,2\}, \{4,5\}, \{1,2,3\}, \{3,4,5\}, \{1,2,3,4\}, \{2,3,4,5\}\}$
- nonzero variables are contiguous in x, *e.g.*, $x^\star = (0, *, *, 0, 0)$
- can extend the same idea to higher dimensions (*e.g.*, select rectangles)
- *e.g.*, time series, tumor diagnosis, . . .

Structured group lasso

(Jacob, Obozinski, Vert; Bach et al.; Zhao, Rocha, Yu; . . .)

hierarchical selection:



- $\mathcal{G} = \{\{4\}, \{5\}, \{6\}, \{2,4\}, \{3,5,6\}, \{1,2,3,4,5,6\}\}$
- nonzero variables form a rooted and connected subtree
  - if node is selected, so are its ancestors
  - if node is not selected, neither are its descendants

Matrix decomposition

- problem:

$$\begin{array}{ll} \text{minimize} & f_1(X_1) + \cdots + f_N(X_N) \\ \text{subject to} & X_1 + \cdots + X_N = A \end{array}$$

- many choices for the f_i:
  - squared Frobenius norm (least squares)
  - entrywise ℓ_1 norm (sparse matrix)
  - nuclear norm (low rank)
  - sum-{row,column}-norm (group lasso)
  - elementwise constraints (fixed sparsity pattern, nonnegative, . . .)
  - semidefinite cone constraint

Low rank matrix completion

(Candès & Recht; Recht, Fazel, Parrilo)

- problem:

$$\begin{array}{ll} \text{minimize} & \|X\|_* \\ \text{subject to} & X_{ij} = A_{ij}, \quad (i,j) \in \mathcal{D} \end{array}$$

  i.e., find low rank matrix that agrees with observed entries

- *e.g.*, Netflix problem

Robust PCA

(Candès et al.; Chandrasekaran et al.)

- regular PCA is the (nonconvex but solvable) problem

$$\begin{array}{ll} \text{minimize} & \|A - L\|_2 \\ \text{subject to} & \mathbf{rank}(L) \leq k \end{array}$$

  i.e., recover rank k matrix L_0 if $A = L_0 + N_0$, where N_0 is noise

- if matrix also has some sparse but large noise, instead solve

$$\begin{array}{ll} \text{minimize} & \|L\|_* + \lambda \|S\|_1 \\ \text{subject to} & L + S = A \end{array}$$

  i.e., recover low rank L and sparse corruption S if $A = L_0 + S_0 + N_0$

- sparse + low rank decomposition has other applications (*e.g.*, vision, video segmentation, background subtraction, biology, indexing)

Robust PCA

(Candès et al.; Chandrasekaran et al.)

Outline

Proximal operator

(Martinet; Moreau; Rockafellar)

- proximal operator of $f : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ is

$$\mathbf{prox}_{\lambda f}(v) = \operatorname*{argmin}_x \left(f(x) + (1/2\lambda)\|x - v\|_2^2 \right)$$

  with parameter $\lambda > 0$

- f may be nonsmooth, have embedded constraints, ...

- can evaluate with standard methods like BFGS, but often has an analytical solution or simple specialized linear-time algorithm

- *many* interpretations

- **example**: proximal operator of $I_{\mathcal{C}}$ is $\Pi_{\mathcal{C}}$ (generalized projection)

Polyhedra

- projection onto polyhedron $\mathcal{C} = \{x \mid Ax = b,\ Cx \leq d\}$ is a QP
- projection onto affine set $\mathcal{C} = \{x \mid Ax = b\}$ is a linear operator
- box or hyperrectangle $\mathcal{C} = \{x \mid l \preceq x \preceq u\}$:

$$(\Pi_{\mathcal{C}}(v))_k = \begin{cases} l_k & v_k \leq l_k \\ v_k & l_k \leq v_k \leq u_k \\ u_k & v_k \geq u_k, \end{cases}$$

- also simple methods for hyperplanes, halfspaces, simplexes, ...

Quadratic functions

- if $f(x) = (1/2)x^T Px + q^T x + r$, then

$$\mathbf{prox}_{\lambda f}(v) = (I + \lambda P)^{-1}(v - \lambda q)$$

- if evaluating repeatedly with different arguments v:
  - dense direct method: $O(n^3)$ flops first time and then $O(n^2)$
  - iterative method (CG, LSQR, ...): warm start beginning at v

Moreau envelope

- **Moreau envelope** or **Moreau-Yosida regularization** of f is

$$M_{\lambda f}(v) = \inf_x \left(f(x) + (1/2\lambda)\|x - v\|_2^2 \right)$$

- a smoothed or regularized form of f:
  - always has full domain
  - always continuously differentiable
  - has the same minimizers as f

- proximal operator is gradient step for Moreau envelope:

$$\mathbf{prox}_{\lambda f}(x) = x - \lambda \nabla M_{\lambda f}(x)$$

Moreau envelope

- motivation: in general, φ^* is smooth when φ is strongly convex
- can show that

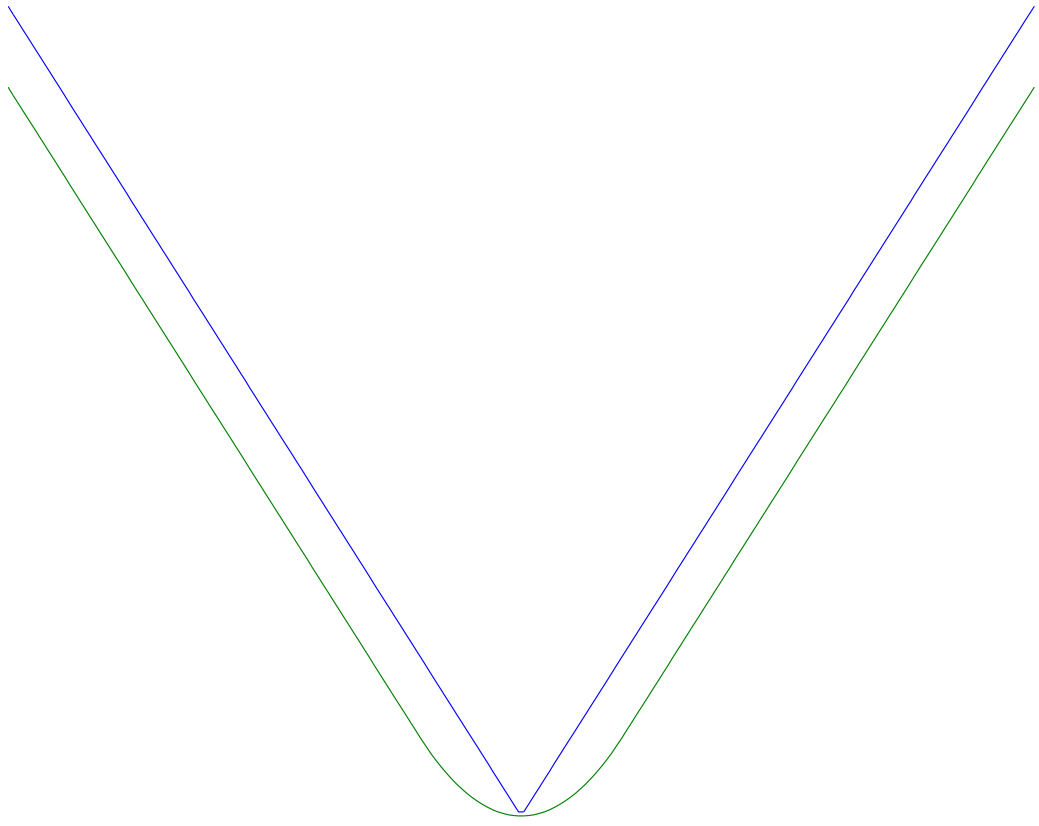
$$M_f = (f^* + (1/2)\|\cdot\|_2^2)^*$$

  so Moreau envelope obtains a smooth approximation via

  1. taking conjugate
  2. regularizing to get a strongly convex function
  3. taking conjugate again

- **example**: Moreau envelope of $|\cdot|$ is the Huber loss function

Moreau envelope: Huber loss



Moreau decomposition

- following relation always holds:

$$v = \mathbf{prox}_f(v) + \mathbf{prox}_{f^*}(v)$$

- main link between proximal operators and duality
- a generalization of orthogonal decomposition induced by subspace L:

$$v = \Pi_L(v) + \Pi_{L^\perp}(v)$$

follows from Moreau decomposition and $(I_L)^* = I_{L^\perp}$

Norms and norm balls

- **in general**: if $f = \|\cdot\|$ and $\mathcal{B}$ is unit ball of dual norm, then

$$\mathbf{prox}_{\lambda f}(v) = v - \lambda \Pi_{\mathcal{B}}(v/\lambda)$$

- if $f = \|\cdot\|_2$ and $\mathcal{B}$ is the unit ℓ_2 ball, then

$$\Pi_{\mathcal{B}}(v) = \begin{cases} v/\|v\|_2 & \|v\|_2 > 1 \\ v & \|v\|_2 \leq 1 \end{cases}$$

$$\mathbf{prox}_{\lambda f}(v) = \begin{cases} (1 - \lambda/\|v\|_2)v & \|v\|_2 \geq \lambda \\ 0 & \|v\|_2 < \lambda \end{cases}$$

sometimes called 'block soft thresholding' operator

Norms and norm balls

- if $f = \|\cdot\|_1$ and $\mathcal{B}$ is the unit ℓ_∞ ball, then

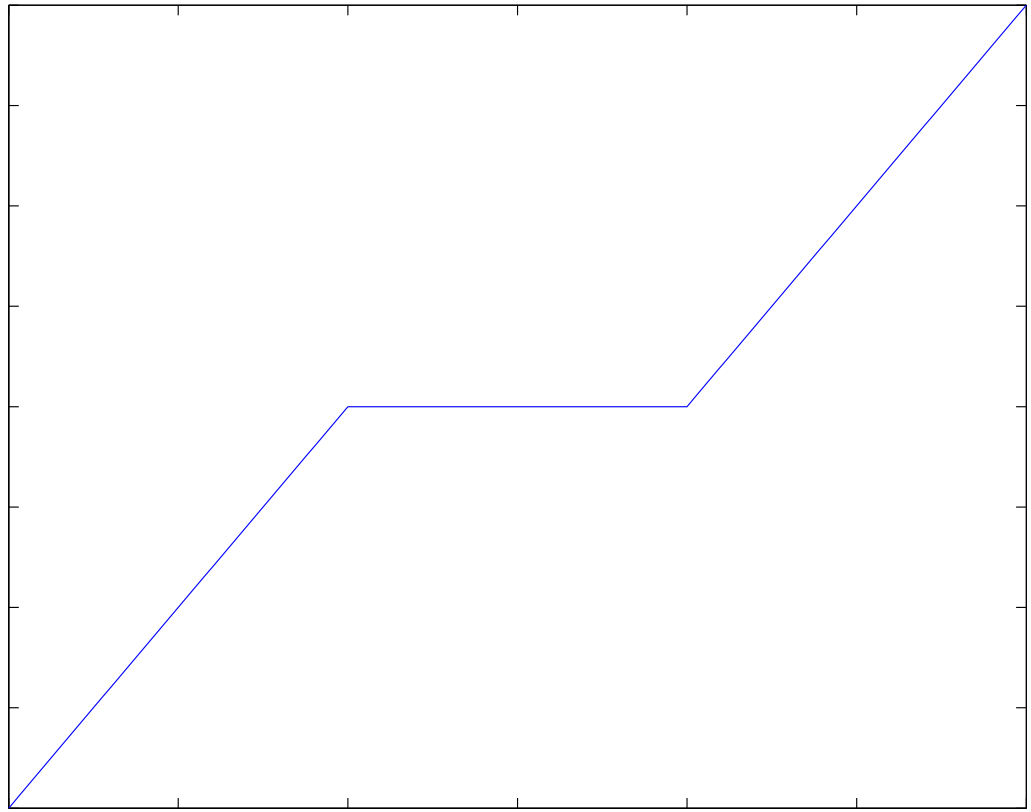
$$(\Pi_{\mathcal{B}}(v))_i = \begin{cases} 1 & v_i > 1 \\ v_i & |v_i| \leq 1 \\ -1 & v_i < -1 \end{cases}$$

  lets us derive (elementwise) **soft thresholding**

$$\mathbf{prox}_{\lambda f}(v) = (v-\lambda)_+ - (-v-\lambda)_+ = \begin{cases} v_i - \lambda & v_i \geq \lambda \\ 0 & |v_i| \leq \lambda \\ v_i + \lambda & v_i \leq -\lambda \end{cases}$$

- if $f = \|\cdot\|_\infty$ and $\mathcal{B}$ is unit ℓ_1 ball, simple algorithms available

Soft thresholding



Matrix functions

- suppose convex $F : \mathbf{R}^{m\times n} \to \mathbf{R}$ is orthogonally invariant:

$$F(QX\tilde{Q}) = F(X)$$

  for all orthogonal Q, $\tilde{Q}$

- then $F = f \circ \sigma$ and

$$\mathbf{prox}_{\lambda F}(A) = U\,\mathbf{diag}(\mathbf{prox}_{\lambda f}(d))V^T$$

  where $A = U\,\mathbf{diag}(d)V^T$ is the SVD of A and $\sigma(A) = d$

- *e.g.*, $F = \|\cdot\|_*$ has $f = \|\cdot\|_1$ so $\mathbf{prox}_{\lambda F}$ is 'singular value thresholding'

Outline

Proximal gradient method

(*e.g.*, Levitin & Polyak; Mercier; Chen & Rockafellar; Combettes; Tseng)

- problem form

$$\text{minimize} \quad f(x) + g(x)$$

  where f is smooth and $g : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ is closed proper convex

- method:

$$x^{k+1} := \mathbf{prox}_{\lambda^k g}(x^k - \lambda^k \nabla f(x^k))$$

- special case: projected gradient method (take $g = I_C$)

Accelerated proximal gradient method

(Nesterov; Beck & Teboulle; Tseng)

- problem form

$$\text{minimize} \quad f(x) + g(x)$$

  where f is smooth and $g : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ is closed proper convex

- method:

$$\begin{aligned} y^{k+1} &:= x^k + \omega^k \left(x^k - x^{k-1}\right) \\ x^{k+1} &:= \mathbf{prox}_{\lambda^k g}\left(y^{k+1} - \lambda^k \nabla f(y^{k+1})\right) \end{aligned}$$

  works for, *e.g.*, $\omega^k = k/(k+3)$ and particular λ^k

- faster in both theory and practice

ADMM

(*e.g.*, Gabay & Mercier; Glowinski & Marrocco; Boyd et al.)

- problem form

$$\text{minimize} \quad f(x) + g(x)$$

  where $f,\ g : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ are closed proper convex

- method:

$$\begin{aligned} x^{k+1} &:= \mathbf{prox}_{\lambda f}(z^k - u^k) \\ z^{k+1} &:= \mathbf{prox}_{\lambda g}(x^{k+1} + u^k) \\ u^{k+1} &:= u^k + x^{k+1} - z^{k+1} \end{aligned}$$

- basically, always works

Examples

- (accelerated) proximal gradient for elastic net:
  1. gradient step for smooth loss (*e.g.*, logistic, least squares, . . .)
  2. shrinkage and elementwise soft thresholding
- ADMM for multi-task learning with joint covariate selection:
  1. evaluate $\mathbf{prox}_{fk}$ (in parallel for each dataset)
  2. block soft thresholding (in parallel for each feature)
  3. dual update
- ADMM for robust PCA:
  1. singular value thresholding
  2. elementwise soft thresholding
  3. dual update

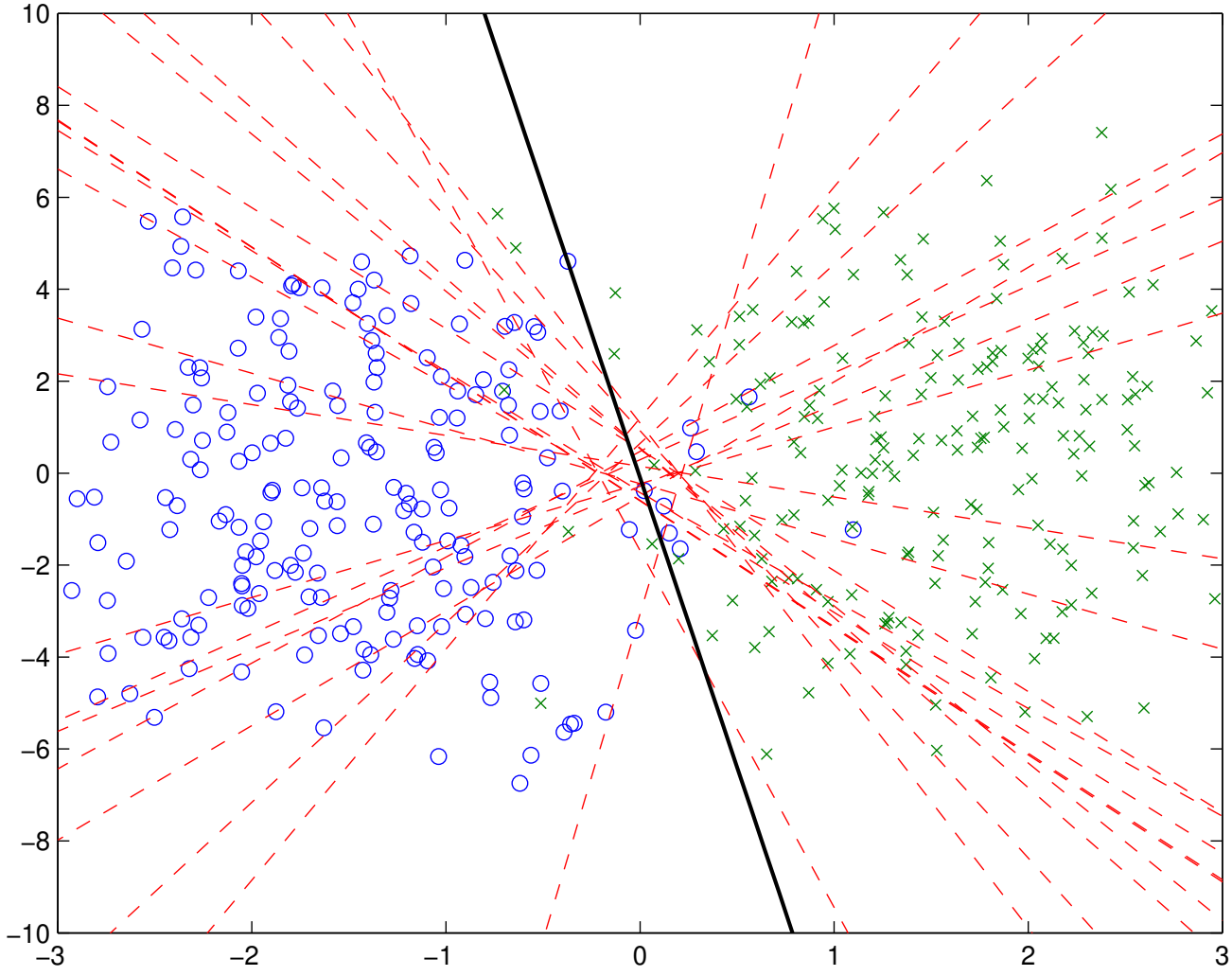
Distributed Optimization and Statistical Learning

Goals

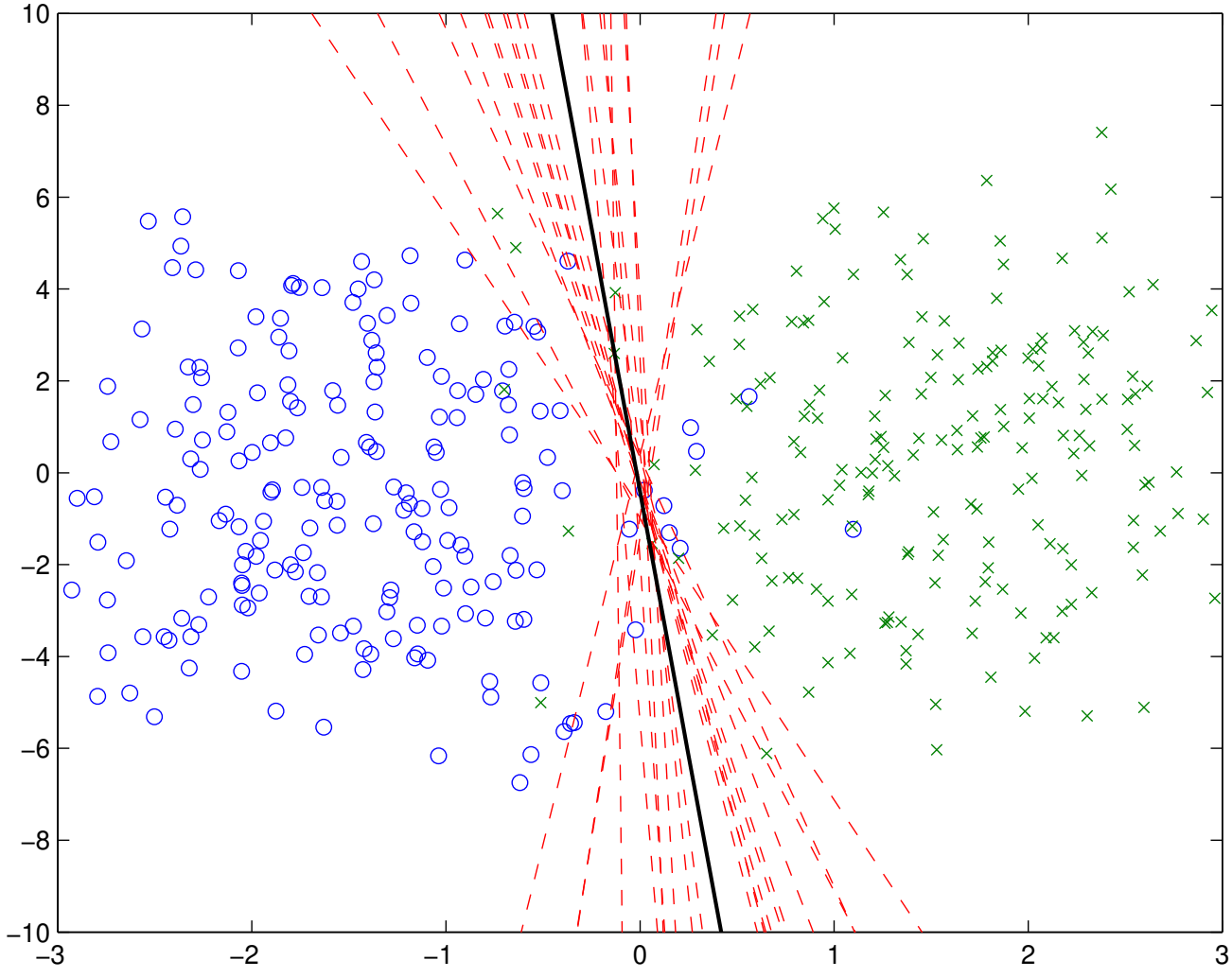
simple and robust methods for

- arbitrary-scale optimization
  - machine learning/statistics with huge datasets
- decentralized optimization
  - have devices/agents coordinate to solve problems by message passing

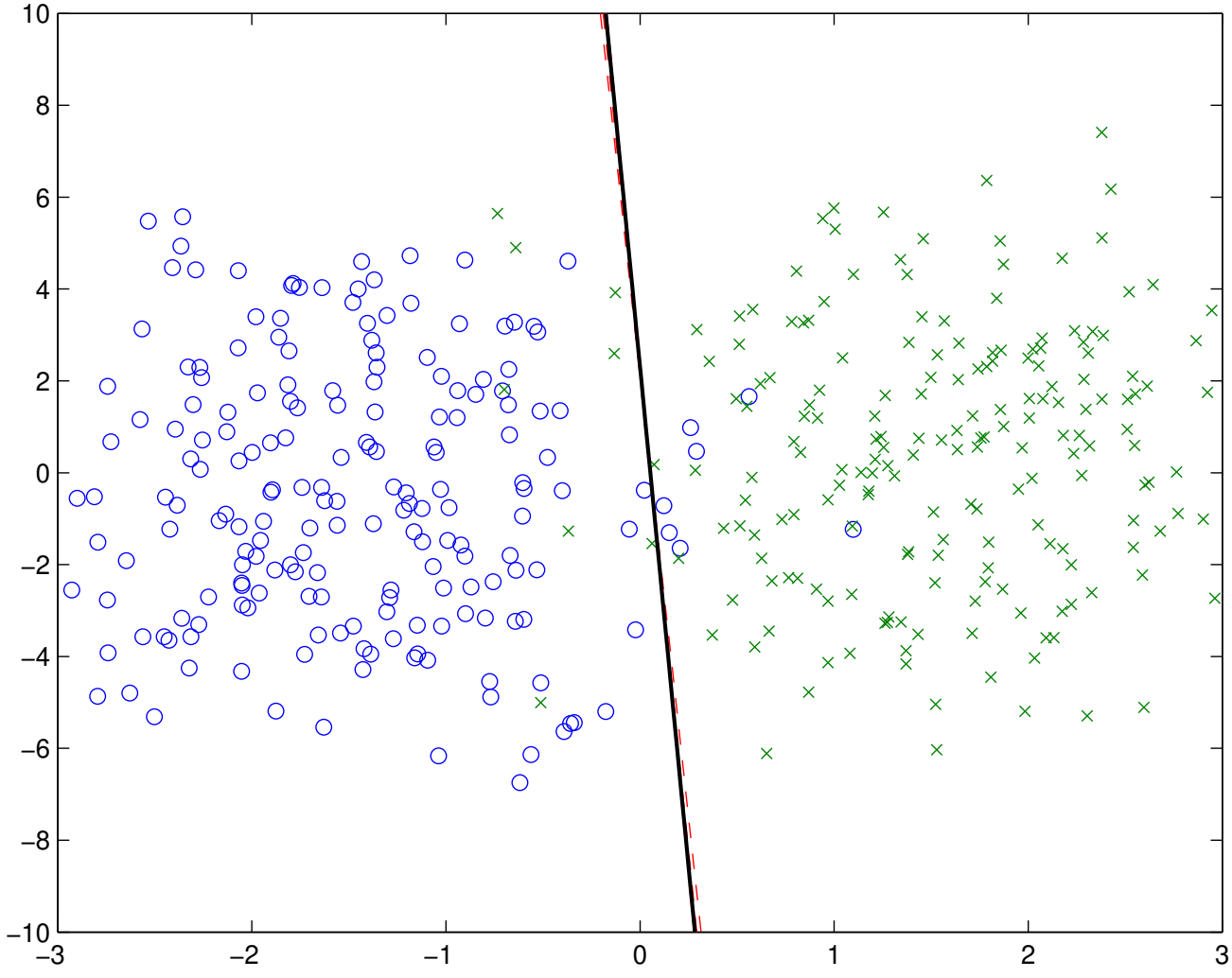
Distributed SVM: Iteration 1



Distributed SVM: Iteration 5



Distributed SVM: Iteration 40



Outline

Operator splitting

- the most useful proximal methods use the idea of **operator splitting**
- these algorithms minimize $f + g$ only using $\mathbf{prox}_f$ and/or $\mathbf{prox}_g$
- useful when f and g each have useful structure separately
- very common in statistical applications: loss + regularizer
- transform problem (if needed) so when an operator splitting method is applied, breaks apart into small pieces with simple proximal operators

Separable sum

- if f is block separable, so $f(x) = \sum_{i=1}^N f_i(x_i)$, then

$$(\mathbf{prox}_f(v))_i = \mathbf{prox}_{f_i}(v_i), \quad i = 1, \ldots, N$$

- key to parallel/distributed proximal algorithms

- for $f = \|\cdot\|_1$, get soft thresholding

$$\mathbf{prox}_{\lambda f}(v) = (v - \lambda)_+ - (-v - \lambda)_+ = \begin{cases} v_i - \lambda & v_i \geq \lambda \\ 0 & |v_i| \leq \lambda \\ v_i + \lambda & v_i \leq -\lambda \end{cases}$$

ADMM

(Douglas-Rachford 55, Gabay-Mercier 76, Glowinski-Marrocco 76)

$$\text{minimize} \quad f(x) + g(x)$$

$f,\ g : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ are closed proper convex

- method:

$$\begin{aligned} x^{k+1} &:= \mathbf{prox}_{\lambda f}(z^k - u^k) \\ z^{k+1} &:= \mathbf{prox}_{\lambda g}(x^{k+1} + u^k) \\ u^{k+1} &:= u^k + x^{k+1} - z^{k+1} \end{aligned}$$

- always converges (if problem is solvable)

Outline

Convex feasibility

- problem:

$$\text{find} \quad x \in \mathcal{C} \cap \mathcal{D}$$

- rewrite as

$$\text{minimize} \quad I_{\mathcal{C}}(x) + I_{\mathcal{D}}(x)$$

- ADMM:

$$\begin{aligned} x^{k+1} &:= \Pi_{\mathcal{C}}(z^k - u^k) \\ z^{k+1} &:= \Pi_{\mathcal{D}}(x^{k+1} + u^k) \\ u^{k+1} &:= u^k + x^{k+1} - z^{k+1} \end{aligned}$$

Positive semidefinite matrix completion

- given $A \in \mathbf{S}^n$ with $(i,j) \in \mathcal{K}$ known, fill in missing entries so in $\mathbf{S}^n_+$
- splitting:

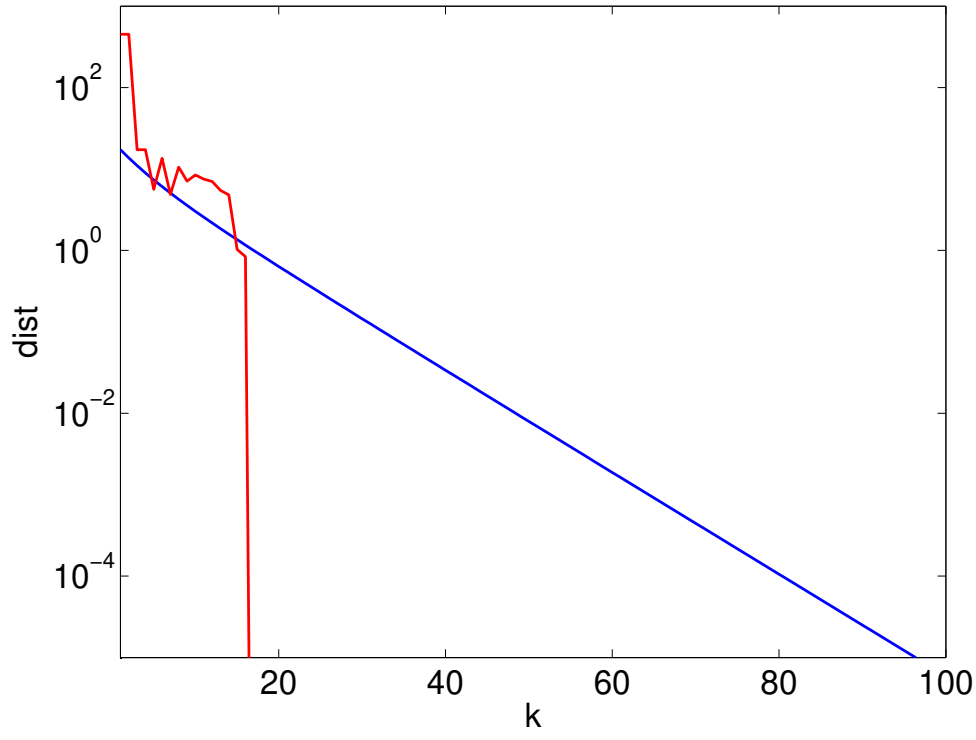
$$\mathcal{C} = \mathbf{S}^n_+, \quad \mathcal{D} = \{X \mid X_{ij} = A_{ij},\ (i,j) \in \mathcal{K}\}$$

- projection onto $\mathcal{C}$: find eigendecomposition $X = \sum_i \alpha_i v_i v_i^T$, then

$$\Pi_{\mathcal{C}} = \sum_{i=1}^{n} \max\{0, \alpha_i\} v_i v_i^T$$

- projection onto $\mathcal{D}$ sets entries to known values

Positive semidefinite matrix completion



- example with 50×50 matrix missing half its entries
- blue: alternating projections; red: ADMM
- $X^k \in \mathcal{C}$, $Z^k \in \mathcal{D}$

Lasso

- to minimize $(1/2)\|Ax - b\|_2^2 + \gamma\|x\|_1$:

$$\begin{aligned} x^{k+1} &:= (I + \lambda A^T A)^{-1}(z^k - u^k - \lambda A^T b) \\ z^{k+1} &:= \mathbf{prox}_{\lambda\gamma\|\cdot\|_1}(x^{k+1} + u^k) \\ u^{k+1} &:= u^k + x^{k+1} - z^{k+1} \end{aligned}$$

- faster implementations:
  - matrix inversion lemma
  - factorization caching
  - warm start
  - adjusting λ
- easily generalizes, *e.g.*, sparse inverse covariance selection

Global consensus optimization

- minimize $f(x) = \sum_{i=1}^N f_i(x)$; *e.g.*, f_i is loss for ith shard of data
- handle each shard separately via splitting

$$\text{minimize} \quad \sum_{i=1}^N f_i(x_i) + I_{\mathcal{C}}(x_1, \ldots, x_N)$$

  with **consensus set** $\mathcal{C} = \{(x_1, \ldots, x_N) \in \mathbf{R}^{nN} \mid x_1 = x_2 = \cdots = x_N\}$

- ADMM simplifies to

$$\begin{aligned} x_i^{k+1} &:= \mathbf{prox}_{f_i}(\overline{x}^k - u_i^k) \\ u_i^{k+1} &:= u_i^k + x_i^{k+1} - \overline{x}^{k+1} \end{aligned}$$

- intuition: u_i measures deviation from average, proximal penalty balances minimizing f_i while pulling towards average

Distributed lasso via consensus optimization

$$
\begin{aligned}
x_i^{k+1} &:= (I + \lambda A_i^T A_i)^{-1}(z^k - u_i^k - \lambda A_i^T b) \\
z^{k+1} &:= \mathbf{prox}_{(\lambda\gamma/N)\|\cdot\|_1}(\overline{x}^{k+1} + \overline{u}^k) \\
u_i^{k+1} &:= u_i^k + x_i^{k+1} - z^{k+1}
\end{aligned}
$$

Distributed global consensus optimization with MPI

initialize N processes, along with x_i, u_i, z.

repeat until converged

1. Update $u_i := u_i + x_i - z$.
2. Update $x_i := \mathbf{prox}_{\lambda f_i}(z - u_i)$.
3. Let $w := x_i + u_i$.
4. *Allreduce* w.
5. Update $z := \mathbf{prox}_{(\lambda/N)g}(w/N)$.

(SPMD: code runs on each separate machine, so i refers to 'local' version)

Distributed lasso example

- example with **dense** $A \in \mathbf{R}^{400000 \times 8000}$ (roughly 30 GB of data)
  - distributed solver written in C using MPI and GSL
  - no optimization or tuned libraries
  - split into 80 subsystems across 10 (8-core) machines on Amazon EC2

- computation times

loading data	30s
factorization	5m
subsequent ADMM iterations	0.5–2s
lasso solve (about 15 ADMM iterations)	5–6m

Matrix decomposition

- decompose matrix A into sum of 'simple' components:

$$\begin{array}{ll} \text{minimize} & f_1(X_1) + f_2(X_2) + \cdots + f_N(X_N) \\ \text{subject to} & A = X_1 + X_2 + \cdots + X_N \end{array}$$

- penalty functions can include
  - squared Frobenius norm
  - entrywise ℓ_1 norm
  - sum-{row,column} norm
  - indicator of elementwise constraints
  - indicator of semidefinite cone
  - nuclear norm

Matrix decomposition via ADMM

- splitting:

$$\text{minimize} \quad \sum_{i=1}^{N} f_i(X_i) + I_{\mathcal{C}}(X_1, \ldots, X_N)$$

with **equilibrium set** $\mathcal{C} = \{(X_1, \ldots, X_N) \mid A = X_1 + \cdots + X_N\}$

- ADMM simplifies to:

$$\begin{aligned} X_i^{k+1} &:= \mathbf{prox}_{\lambda f_i}(X_i^k - \overline{X}^k + (1/N)A - U^k) \\ U^{k+1} &:= U^k + \overline{X}^{k+1} - (1/N)A \end{aligned}$$

Matrix decomposition results

problem: decompose $A =$ rank 4 $+$ sparse $+$ small Gaussian noise

Method	m	n	Iterations	Time (s)
CVX	10	30	15	1.11
ADMM	10	30	45	0.02
CVX	20	50	17	2.54
ADMM	20	50	42	0.03
CVX	40	80	20	108.14
ADMM	40	80	36	0.07
ADMM	100	200	38	0.58
ADMM	500	1000	42	35.56

note: last instance has 1.5M variables and 500K constraints

Outline

Graph form problems

- graph form problem:

$$\begin{array}{ll} \text{minimize} & f(y) + g(x) \\ \text{subject to} & y = Ax \end{array}$$

  where $A \in \mathbf{R}^{m \times n}$, $f : \mathbf{R}^m \to \mathbf{R} \cup \{+\infty\}$, $g : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$

- x and y must lie in the graph $\{(x, y) \in \mathbf{R}^{m+n} \mid y = Ax\}$ of A

- refer to x as 'inputs' and y as 'outputs'

- f and g can encode constraints

Example: cone programming

- cone program in standard form:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \in \mathcal{K} \end{array}$$

  where $\mathcal{K}$ is a convex cone

- in graph form, let

$$f(y) = I_{\{b\}}(y), \qquad g(x) = c^T x + I_{\mathcal{K}}(x)$$

  where $I_{\mathcal{C}}$ is the indicator function of the convex set $\mathcal{C}$

- *e.g.*, *symmetric cone program* when $\mathcal{K}$ is a product of $\mathbf{R}^n_+$, $\mathbf{Q}^n$, $\mathbf{S}^n_+$

Example: loss minimization

- many statistics/ML problems take the form

$$\text{minimize} \quad l(Ax - b) + r(x)$$

- in graph form, let

$$f(y) = l(y - b), \qquad g(x) = r(x)$$

- *e.g.*, obtain the lasso with $l(u) = (1/2)\|u\|_2^2$ and $r(v) = \gamma\|v\|_1$
- can similarly express linear SVM, MLE/MAP in exponential families, ...

ADMM

- consider generic constrained convex program

$$\begin{array}{ll} \text{minimize} & f(z) \\ \text{subject to} & z \in \mathcal{C} \end{array}$$

- ADMM:

$$\begin{array}{lcll} z^{k+1/2} & := & \mathbf{prox}_f(z^k - \tilde{z}^k) & \text{// prox} \\ z^{k+1} & := & \Pi_{\mathcal{C}}(z^{k+1/2} + \tilde{z}^k) & \text{// projection} \\ \tilde{z}^{k+1} & := & \tilde{z}^k + z^{k+1/2} - z^{k+1} & \text{// dual update} \end{array}$$

- converges under very general conditions

Graph projection splitting

- applying this form of ADMM to graph form problem gives

$$\begin{aligned}
x^{k+1/2} &:= \mathbf{prox}_g(x^k - \tilde{x}^k) \\
y^{k+1/2} &:= \mathbf{prox}_f(y^k - \tilde{y}^k) \\
(x^{k+1}, y^{k+1}) &:= \Pi_A(x^{k+1/2} + \tilde{x}^k, y^{k+1/2} + \tilde{y}^k) \\
\tilde{x}^{k+1} &:= \tilde{x}^k + x^{k+1/2} - x^{k+1} \\
\tilde{y}^{k+1} &:= \tilde{y}^k + y^{k+1/2} - y^{k+1}
\end{aligned}$$

- Π_A is called *graph projection* and denotes projection onto graph of A
- **important**: f and g never interact directly with A, *i.e.*, Π_A is the only operation that touches the data

Graph projection

- evaluating $\Pi_A(c, d)$ involves solving

$$\begin{array}{ll} \text{minimize} & (1/2)\|x - c\|_2^2 + (1/2)\|y - d\|_2^2 \\ \text{subject to} & y = Ax \end{array}$$

- reduce to solving (quasidefinite) KKT system

$$\begin{bmatrix} I & A^T \\ A & -I \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} c + A^T d \\ 0 \end{bmatrix}$$

Implementing graph projections

- eliminate x, then solve for y

$$\begin{array}{lcl} y & := & (I + AA^T)^{-1}(Ac + AA^T d) \\ x & := & c + A^T(d - y) \end{array}$$

- eliminate y, then solve for x

$$\begin{array}{lcl} x & := & (I + A^T A)^{-1}(c + A^T d) \\ y & := & Ax \end{array}$$

- when A is dense, prefer first when A is fat and second when A is skinny
- factor the relevant coefficient matrix with, *e.g.*, Cholesky factorization
- see paper for more on other situations (*e.g.*, sparse A)
- **key point**: does not depend on λ, f, g

Example

- graph projection splitting algorithm for the lasso:

$$\begin{aligned} x^{k+1/2} &:= \mathbf{prox}_{\lambda\gamma\|\cdot\|_1}(x^k - \tilde{x}^k) \\ y^{k+1/2} &:= (1/(1+\lambda))(y^k - \tilde{y}^k) \\ (x^{k+1}, y^{k+1}) &:= \Pi_A(x^{k+1/2} + \tilde{x}^k, y^{k+1/2} + \tilde{y}^k) \\ \tilde{x}^{k+1} &:= \tilde{x}^k + x^{k+1/2} - x^{k+1} \\ \tilde{y}^{k+1} &:= \tilde{y}^k + y^{k+1/2} - y^{k+1} \end{aligned}$$

- here, all operations are trivial except for Π_A

- cache factorization needed to evaluate Π_A and reuse after first iteration
- regularization path: reuse across solves, varying γ
- model comparison: reuse across solves, varying f or g

Sample implementation: lasso

```
prox_f = @(v,lambda) (1/(1 + lambda))*(v - b) + b;
prox_g = @(v,lambda) (max(0, v - lambda) - max(0, -v - lambda));

AA = A*A';
L  = chol(eye(m) + AA);

for iter = 1:MAX_ITER
    xx = prox_g(xz - xt, lambda);
    yx = prox_f(yz - yt, lambda);

    yz = L \ (L' \ (A*(xx + xt) + AA*(yx + yt)));
    xz = xx + xt + A'*(yx + yt - yz);

    xt = xt + xx - xz;
    yt = yt + yx - yz;
end
```

Numerical example: regularization path for lasso

- dense $A \in \mathbf{R}^{5000 \times 8000}$
- 10 values of γ, log spaced from $0.01\gamma_{\max}$ to $\gamma_{\max}$
- solve all instances in 23 sec total, vs 72 sec if not sharing cache across problem instances
- (by comparison, solving one instance with CVX takes 2 minutes)

Block splitting

- now turn to distributed setting
- approach:
  1. express graph form problems using blocks
  2. use problem transformation
  3. apply version of ADMM given earlier
  4. simplify algorithm

Block partitioned form

- suppose f and g are *block separable*, *i.e.*

$$f(y) = \sum_{i=1}^{M} f_i(y_i), \qquad g(x) = \sum_{j=1}^{N} g_j(x_j),$$

  where $y_i \in \mathbf{R}^{m_i}$, $x_j \in \mathbf{R}^{n_j}$

- partition A conformably

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1N} \\ A_{21} & A_{22} & \cdots & A_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{M1} & A_{M2} & \cdots & A_{MN} \end{bmatrix}$$

Graph form problems

- graph form problem can then be expressed as

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{M} f_i(y_i) + \sum_{j=1}^{N} g_j(x_j) \\ \text{subject to} & y_i = \sum_{j=1}^{N} A_{ij} x_j, \quad i = 1, \ldots, M \end{array}$$

- $M = 1$ called column (feature) splitting
- $N = 1$ called row (data) splitting
- goal is to solve this in a way that allows each block A_{ij} to be handled by a separate process or machine

Example: loss minimization

- row splitting corresponds to splitting by data
- column splitting corresponds to splitting by features
- loss l is typically fully separable
- regularizer r is often separable and sometimes block separable (*e.g.*, group lasso or sum-of-norms regularization)

Problem transformation

- introduce additional variables

$$\begin{array}{lll} \text{minimize} & \sum_{i=1}^{M} f_i(y_i) + \sum_{j=1}^{N} g_j(x_j) & \\ \text{subject to} & x_{ij} = x_j, & i = 1, \ldots, M \\ & y_i = \sum_{j=1}^{N} y_{ij}, & i = 1, \ldots, M \\ & y_{ij} = A_{ij} x_{ij}, & i = 1, \ldots, M, \quad j = 1, \ldots, N \end{array}$$

- move some constraints into the objective

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{M} f_i(y_i) + \sum_{j=1}^{N} g_j(x_j) + \sum_{i=1}^{M} \sum_{j=1}^{N} I_{ij}(y_{ij}, x_{ij}) \\ \text{subject to} & x_{ij} = x_j \\ & y_i = \sum_{j=1}^{M} y_{ij} \end{array}$$

- now apply ADMM and simplify

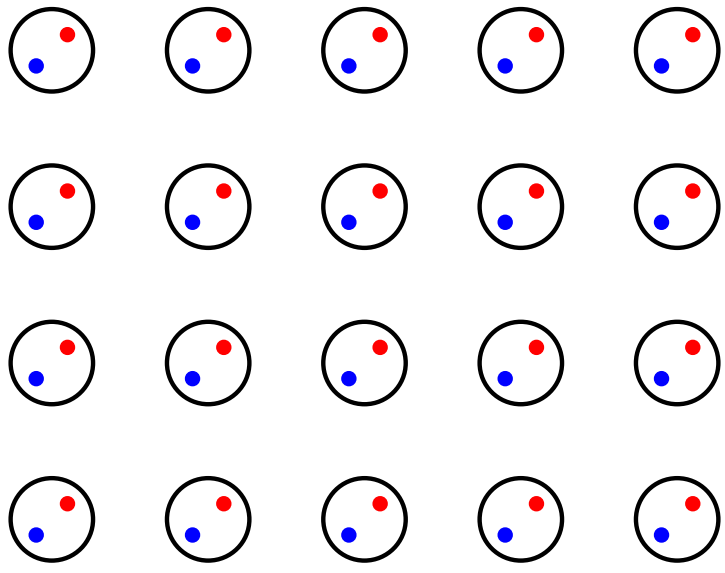
Block splitting algorithm

$$
\begin{aligned}
y_i^{k+1/2} &:= \mathbf{prox}_{l_i}(y_i^k - \tilde{y}_i^k) \\
x_j^{k+1/2} &:= \mathbf{prox}_{r_j}(x_j^k - \tilde{x}_j^k) \\
(y_{ij}^{k+1/2}, x_{ij}^{k+1/2}) &:= \Pi_{ij}(y_{ij}^k + \tilde{y}_i^k, x_j^k - \tilde{x}_{ij}^k) \\
x_j^{k+1} &:= \mathbf{avg}(x_j^{k+1/2}, \{x_{ij}^{k+1/2}\}_{i=1}^M) \\
(y_i^{k+1}, \{y_{ij}^{k+1}\}_{j=1}^N) &:= \mathbf{exch}(y_i^{k+1/2}, \{y_{ij}^{k+1/2}\}_{j=1}^N) \\
\tilde{x}_j^{k+1} &:= \tilde{x}_j^k + x_j^{k+1/2} - x_j^{k+1} \\
\tilde{y}_i^{k+1} &:= \tilde{y}_i^k + y_i^{k+1/2} - y_i^{k+1} \\
\tilde{x}_{ij}^{k+1} &:= \tilde{x}_{ij}^k + x_{ij}^{k+1/2} - x_j^{k+1}
\end{aligned}
$$

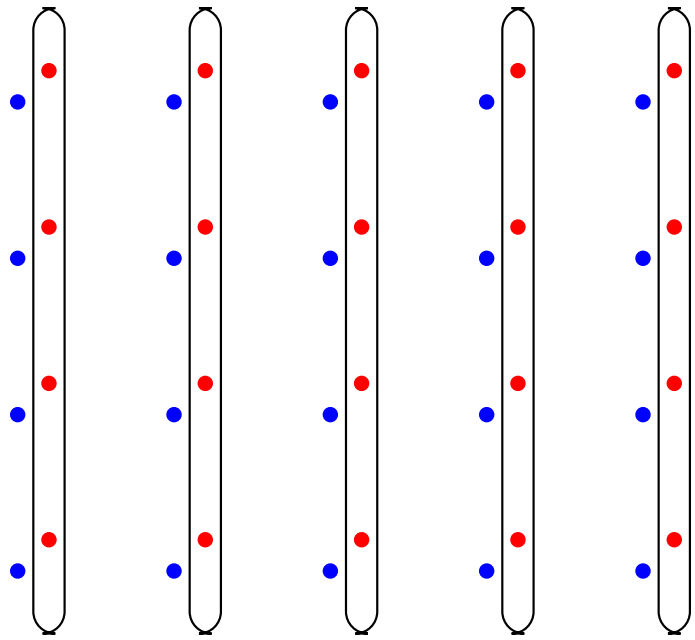
where $\mathbf{exch}(c, \{c_j\}_{j=1}^N)$ is given by

$$y_{ij}^{k+1} := c_j + (c - \sum_{j=1}^N c_j)/(N+1),\ y_i^{k+1} := c - (c - \sum_{j=1}^N c_j)/(N+1)$$

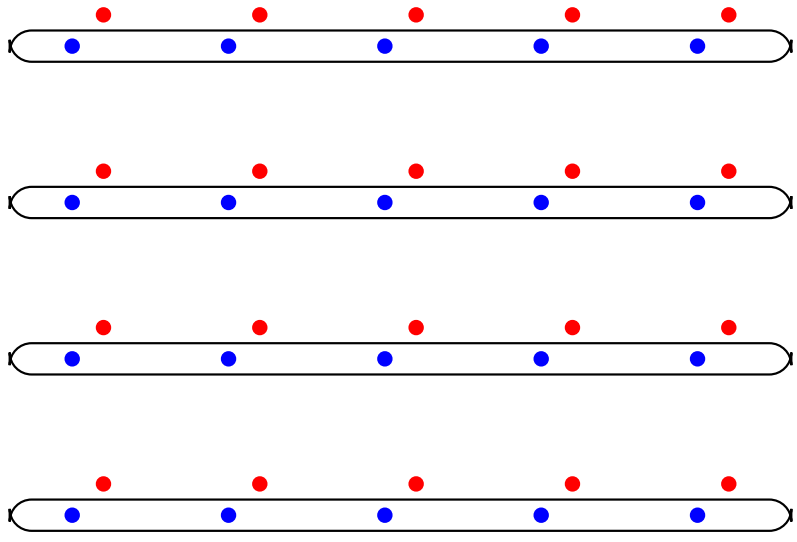
Computation: prox, Π_{ij}, dual updates



Computation: consensus (avg)



Computation: exchange (exch)



Distributed lasso

- examples with **dense** $A_{ij} \in \mathbf{R}^{3000 \times 5000}$
  - distributed solver written in C using MPI and GSL (ATLAS)
  - run on Amazon EC2 cluster compute nodes

- computation times (all times in seconds)

$M \times N$	4×2	8×5	8×10
nonzero entries	120MM	600MM	1.2B
# cores	8	40	80
factorization time	15	15	15
iteration time	0.05–0.15	0.05–0.15	0.05–0.15
# iterations	90	230	490
main loop time	10	27	60
total time	28	50	80

Outline

Summary and conclusions

- coordinate many processors, each solving a substantial problem, to solve a very large problem
- split by data, features, or both
- yields algorithms that are easy to implement (just supply prox operators)

Summary and conclusions

interaction between three key ingredients:

1. proximal operator of a convex function
2. operator splitting algorithms
3. problem transformations

References

- *Proximal algorithms* (Parikh, Boyd)
- *Distributed optimization and statistical learning via the alternating direction method of multipliers* (Boyd, Parikh, Chu, Peleato, Eckstein)
- *Block splitting for distributed optimization* (Parikh, Boyd)

available from `nparikh.org` or `stanford.edu/~boyd`