THE UNIVERSITY OF TEXAS AT AUSTIN
Department of Statistics and Data Sciences
College of Natural Sciences

SDS 385: Stat Models for Big Data

Lecture 5: Proximal methods

Purnamrita Sarkar
Department of Statistics and Data Science
The University of Texas at Austin

https://psarkar.github.io/teaching

Proximal methods

- You want to minimize functions of the form

$$f(x) = \underbrace{g(x)}_{convex, differentiable} + \underbrace{h(x)}_{convex, nonsmooth}$$

- If h was differentiable, we would use

$$x_{k+1} = x_k - \alpha \nabla f(x_k)$$

- Here we would use:

$$x_{k+1} = \arg\min_z \frac{1}{2\alpha} \underbrace{\|z - (x_t - \alpha \nabla g(x_t))\|^2}_{\text{Stay close to the gradient direction}} + \underbrace{h(z)}_{\text{minimize h}}$$

Proximal mapping

- Define:

$$\text{prox}_\alpha(x) = \arg\min_z \frac{1}{2\alpha}\|x - z\|^2 + h(z)$$

- Proximal GD:
  - Choose initial $x^{(0)}$
  - Repeat, for $k = 1, 2, 3$

$$x_{k+1} = \text{prox}_{\alpha_k}(x_k - \alpha_k \nabla g(x_k))$$

- But, we just turned one minimization into another. And both has h which is the troublesome part.

Example: Lasso

$$f(\beta) = \frac{1}{2}\|y - X\beta\|^2 + \lambda\|\beta\|_1$$

- The proximal map is:

$$\begin{aligned}
\mathrm{prox}_\alpha(\beta) &= \arg\min_z \left(\frac{1}{2\alpha}\|\beta - z\|^2 + \lambda\|z\|_1 \right) \\
&= S_{\lambda\alpha}(\beta) \\
[\mathrm{prox}_\alpha(\beta)]_i &= \begin{cases} \beta_i - \lambda\alpha & \text{If } \beta_i > \lambda\alpha \\ 0 & \text{If } |\beta_i| \le \lambda\alpha \\ \beta_i + \lambda\alpha & \text{If } \beta_i < -\lambda\alpha \end{cases}
\end{aligned}$$

Lasso

- In this case, the gradient is

$$\nabla g(\beta) = -X^T(y - X\beta)$$

- So the update step for Lasso becomes:

$$\beta_{k+1} = S_{\lambda\alpha}\left(\beta_k + \alpha X^T(y - X\beta)\right)$$

- This is the Iterative Soft Thresholding Algorithm (ISTA), due to Beck and Teboule, 2008. "A fast iterative shrinkage-thresholding algorithm for linear inverse problems"

Convergence

- Recall our setup. We are minimizing $g(\beta) + h(\beta)$, where
    - g is convex and differentiable (what you had assumed for gradient and stochastic gradient descent methods)
    - ∇g is $L-$Lipschitz
    - $h(x)$ is convex.
    - Now if you can compute the proximal operator, then:

Theorem
As long as $\alpha \leq 1/L$,

$$f(\beta^{(k)}) - f(\beta^*) \leq \frac{\|\beta^{(0)} - \beta^*\|_2^2}{2k\alpha}$$

- You can also add Nesterov's accelerated gradient to this.

FISTA-Fast Iterative Shrinkage-Thresholding Algorithm

- Start with $\beta^{(0)}$
- Compute $v = \beta^{(k-1)} + \frac{k-2}{k+1}(\beta^{(k-1)} - \beta^{(k-2)})$
- Compute $\beta^{(k)} = \text{prox}_{\alpha_k}(v - \alpha_k \nabla g(v))$
- Handwavy explanation
  - The $(k-2)/(k+1)$ is important.
  - Note this is $1/4$ in the beginning, but then increases to 1
  - As we keep getting closer to the optima, the gradient becomes smaller (its zero at the optima)
  - The acceleration basically pushes more and more in this direction if you are close to the optima (more so as $k \to \infty$, where momentum becomes 1.)
- Converges much faster.

Matrix completion

- Given the “observed” entries, we want to infer the unobserved entries.
- Helps in recommending new books/movies/music to users.
- Typically, we pose this as an optimization problem, with suitable constraints on the learned matrix.

Example: Recommender systems

Alice	5	---	5	---	---	1
Bob	---	3	---	5	4	---
Reba	4	---	4	5	---	4

- Here, each row represents a user or customer.
- Each column represents a product
    - This can be a movie for Netflix
    - This can be a book for Amazon or Goodreads
    - This can be a product on Amazon
- The $(i, j)^{th}$ entry represents the rating provided by user i for product j
- Not all elements are observed, since not every customer has rated every product

Matrix completion - formulation

- We will provide optimization objectives which deals directly with observed and unobserved entries.
- Notation – Let Ω denote the set of pairs (i, j) such that X_{ij} is observed.

$$\min_B \underbrace{\sum_{ij \in \Omega} (X_{ij} - B_{ij})^2}_{\text{loss over } \Omega} + \underbrace{\lambda \mathcal{R}(B)}_{\text{regularization}}$$

Matrix completion - formulation

$$\min_{B \in \mathbb{R}^{m \times n}} \underbrace{\sum_{ij \in \Omega} (X_{ij} - B_{ij})^2}_{\text{loss over } \Omega} + \underbrace{\lambda \mathcal{R}(B)}_{\text{regularization that involves all entries}}$$

- So what kind of regularization can we use?
- How about a rank constraint?

Matrix completion - formulation

$$\min_{B \in \mathbb{R}^{m \times n}} \underbrace{\sum_{ij \in \Omega} (X_{ij} - B_{ij})^2}_{\text{loss over } \Omega} + \underbrace{\lambda \mathcal{R}(B)}_{\text{regularization that involves all entries}}$$

- So what kind of regularization can we use?
- How about a rank constraint?

$$\min_{B} \sum_{ij \in \Omega} (X_{ij} - B_{ij})^2$$

$$\text{s.t.} \operatorname{rank}(B) = k$$

Matrix completion - formulation

- Rank constraints in the above setting can be combinatorially very hard.

Matrix completion - formulation

- Rank constraints in the above setting can be combinatorially very hard.
- Funny right? because some rank minimization problems are easy – if I ask you to return the best rank k approximation of a matrix. But the moment you add more structure, i.e. minimize frobenius norm over a set of pairs, things get hairy.

> There are several special cases of the RMP that have well known solutions. For example, approximating a given matrix with a low-rank matrix in spectral or Frobenius norm is an RMP that can be solved via singular value decomposition (SVD) [15]. However, in general, the RMP is known to be computationally intractable (NP-hard) [26]..

Rank Minimization and Applications in System Theory.Fazel, Hindi and Boyd, 2004

Matrix completion - formulation

- Instead, what is often used is the nuclear norm penalty.

$$\min_{B \in \mathbb{R}^{m \times n}} \sum_{ij \in \Omega} (X_{ij} - B_{ij})^2 + \lambda \|B\|_*$$

- Recall that the nuclear norm is basically the sum of the singular values of a matrix.
- The rank can be thought of as a ℓ_0 "norm" of the vector of singular values, constraints based on which are not convex
- The nuclear norm is like a ℓ_1 norm.

Matrix completion - formulation

- Instead, what is often used is the nuclear norm penalty.

$$\min_{B \in \mathbb{R}^{m \times n}} \sum_{ij \in \Omega} (X_{ij} - B_{ij})^2 + \lambda \|B\|_*$$

- Recall that the nuclear norm is basically the sum of the singular values of a matrix.
- The rank can be thought of as a ℓ_0 "norm" of the vector of singular values, constraints based on which are not convex
- The nuclear norm is like a ℓ_1 norm.
- As it turns out the nuclear norm is the tightest convex relaxation of of rank of a matrix. (See Fazel, Hindi and Boyd)

Matrix completion - formulation

- Instead, what is often used is the nuclear norm penalty.

$$\min_{B \in \mathbb{R}^{m \times n}} \sum_{ij \in \Omega} (X_{ij} - B_{ij})^2 + \lambda \|B\|_*$$

- Recall that the nuclear norm is basically the sum of the singular values of a matrix.
- The rank can be thought of as a ℓ_0 "norm" of the vector of singular values, constraints based on which are not convex
- The nuclear norm is like a ℓ_1 norm.
- As it turns out the nuclear norm is the tightest convex relaxation of of rank of a matrix. (See Fazel, Hindi and Boyd)
    - In plain words, over a bounded set, the nuclear norm function is the largest convex function smaller than the rank function, otherwise also known as the convex envelop.

Example: matrix completion

Given a matrix $X \in \mathbb{R}^{m \times n}$ and observed entries $(i,j) \in \Omega$, you want to fill missing entries by solving:

$$\min_{B \in \mathbb{R}^{m \times n}} \frac{1}{2} \sum_{ij \in \Omega} (X_{ij} - B_{ij})^2 + \lambda \|B\|_*$$

- $\|B\|_*$ is the nuclear norm of B, defined as:

$$\|B\|_* = \sum_{i=1}^{k} \sigma_i(B),$$

  where k is the rank of B and $\sigma_1(B) \geq \sigma_2(B) \dots$ are the singular values.
- Nuclear norm is a convex approximation of rank, think how you cannot easily minimize ℓ_0 norm, aka the number of nonzero entries, an instead minimize the ℓ_1 norm to induce sparsity in regression problems.

Proximal gradient

- $[P_\Omega(B)]_{ij} = B_{ij} 1((ij) \in \Omega)$
- So the optimization can also be written as:

$$\min \frac{1}{2} \|P_\Omega(X) - P_\Omega(B)\|_F^2 + \lambda \|B\|_*$$

- Gradient of smooth first part: $-(P_\Omega(X) - P_\Omega(B))$
- Prox function:

$$\text{prox}_\alpha(B) = \arg \min_{Z \in \mathbb{R}^{m \times n}} \frac{1}{2\alpha} \|B - Z\|_F^2 + \lambda \|Z\|_*$$

Proximal GD

- It can be shown that $\text{prox}_\alpha(B) = S_\alpha(B)$, where
- $S_\alpha(B)$ is $U\Sigma_\alpha V^T$, where $B = U\Sigma V^T$ and

$$\Sigma_\alpha(i,i) = \max(\Sigma_{ii} - \alpha, 0)$$

Proximal GD

- $B_{k+1} = S_{\lambda\alpha}(B + \alpha(P_\Omega(Y) - P_\Omega(B))$
- This is called the Soft Impute algorithm.
    - Cai et al, "A Singular Value Thresholding Algorithm for Matrix Completion", 2010.
    - Mazumdar et al 2011, "Spectral regularization algorithms for learning large incomplete matrices"

Algorithm

- Set $Z^{(0)} = 0$
- Set $B_{ij}^{(t+1)} = \begin{cases} Y_{ij} & (i,j) \in \Omega \\ Z_{ij}^{(t)} & (i,j) \notin \Omega \end{cases}$
- Compute $B^{(t)} = U\text{diag}[\sigma_1, \ldots, \sigma_r]V^T$
- Compute $Z^{(t+1)} = U\text{diag}[(\sigma_1 - \lambda)_+, \ldots, (\sigma_r - \lambda)_+]V^T$

Lets do a real example

- We will load an image and convert its grayscale version into a matrix.

```
from keras.preprocessing.image import load_img
# load the image
img = load_img('bondi_beach.jpeg',grayscale=True)
# report details about the image
print(type(img))
print(img.format)
print(img.mode)
print(img.size)
# show the image
img.show()
```

```
<class 'PIL.Image.Image'>
None
L
(640, 427)
```

Lets do a real example

- We will load an image and convert its grayscale version into a matrix.

```
from keras.preprocessing.image import img_to_array
img_array = np.squeeze(img_to_array(img))
print(img_array.dtype)
print(img_array.shape)
```

```
float32
(427, 640)
```

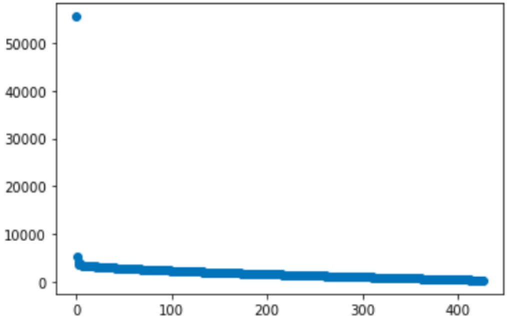
Lets do a real example

- Now we will sample 100,000 entries at random and withhold them.

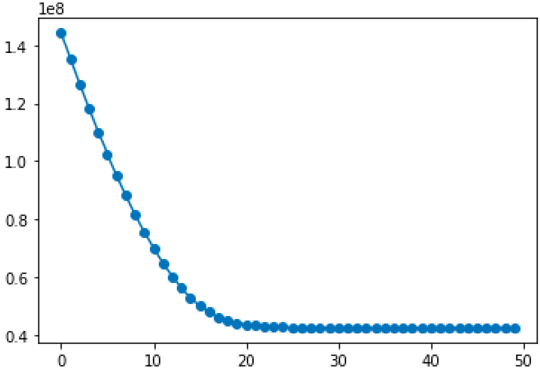
```
from keras.preprocessing.image import array_to_img

img2=copy.copy(img_array)
n=range(100000)
nrow=427
ncol=640
for i in n:
    row=random.choice(range(nrow))
    col=random.choice(range(ncol))
    row
    img2[row,col]=0;
plt.matshow(img2)
```

Applying the Soft Impute algorithm

- But what kind of a λ do we use?
- Here is a plot of the singular values of the matrix `img2`



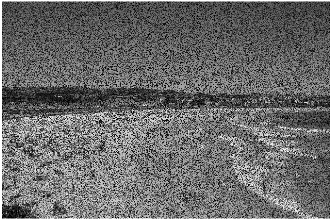
Applying the Soft Impute algorithm



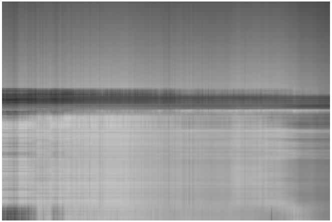
- Good sanity check to see if the loss is going down with the number of iterations.

Applying the Soft Impute algorithm with $\lambda = 2000$ and $\lambda = 50$

Original



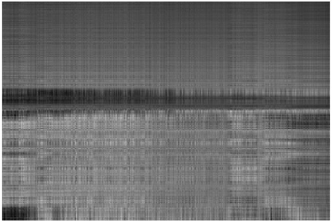
Noisy



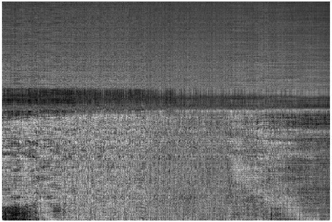
$\lambda = 2000$

$\lambda = 50$

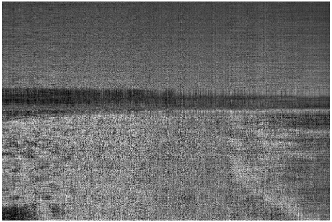
Applying SVD



$K = 5$



$K = 20$



$K = 30$

$K = 50$

Proximal GD

- We will show that $\text{prox}_\alpha(B) = S_\alpha(B)$, where
- $S_\alpha(B)$ is $U\Sigma_\alpha V^T$, where $B = U\Sigma V^T$ and

$$\Sigma_\alpha(i,i) = \max(\Sigma_{ii} - \alpha, 0)$$

- First, it is known that the subdifferential of the nuclear norm is given by: $\partial\|Z\|_* = \{UV^T + W : \|W\| \leq 1, U^T W = 0, WV = 0\}$, where $U \in \mathbb{R}^{m\times r}, V \in \mathbb{R}^{n\times r}$ where $Z = U\Sigma V^T$ where Σ contains the nonzero singular values of Z.
- Now we will show that $0 \in S_{\alpha\lambda}(B) - B + \lambda\alpha\partial\|S_{\alpha\lambda}(B)\|_*$

Proximal GD

- Take U_0, V_0 as the singular vectors corresponding to $\sigma_i(B) > \lambda\alpha =: t$.
- Take the remaining singular vectors as $U_\perp, V_\perp$ and the corresponding singular value matrix as $\Sigma_\perp$
- $S_t(B) - B = -tU_0V_0^T - U_\perp \Sigma_\perp V_\perp^T$
- $S_t(B) - B + t(U_0V_0^T + W) = tW - U_\perp \Sigma_\perp V_\perp^T$
- Taking $W = U_\perp \Sigma_\perp V_\perp^T / t$, we see that
  - $U^T W = 0$
  - $WV = 0$
  - $\|W\| \leq 1$

Acknowledgment

Cai et al, "A Singular Value Thresholding Algorithm for Matrix Completion", 2010.