STAT 432: Basics of Statistical Learning

Introduction to Convex Optimization

Shiwei Lan, Ph.D. <shiwei@illinois.edu>

http://shiwei.stat.illinois.edu/stat432.html

January 18, 2019

University of Illinois at Urbana-Champaign

Convex Optimization

- This lecture gives a very brief introduction to convex optimization
- The goal is to have sufficient knowledge to deal with specific problems such as Lasso, SVM, etc.
- Reference:
  Boyd, Stephen, and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
- Many of the figures in this lecture are taken from online sources. I want to thank all of them!

Convex Optimization

- The problem: minimizing a convex function in a convex set

$$
\begin{aligned}
\underset{\boldsymbol{\beta}}{\text{minimize}} \quad & f(\boldsymbol{\beta}) \\
\text{subject to} \quad & g_i(\boldsymbol{\beta}) \leq 0, \quad i = 1, \ldots, m \\
& \mathbf{A}\boldsymbol{\beta} = b
\end{aligned}
$$

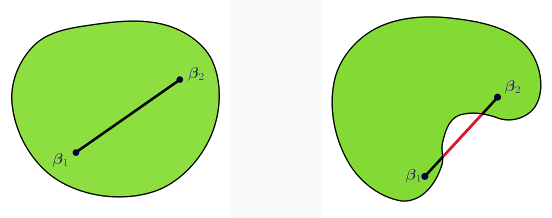
- Examples:
  - Linear regression: minimize $\frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2$, subject to none.
  - Ridge regression: minimize $\frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2$, subject to $\sum_{j=1}^{p} \beta_j^2 < s$
  - First principal component: maximize $\boldsymbol{\beta}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\boldsymbol{\beta}$, subject to $\boldsymbol{\beta}^\mathsf{T}\boldsymbol{\beta} = 1$

Convex Optimization

- What is a convex set $C \in \mathbb{R}^p$?

$$\boldsymbol{\beta}_1, \boldsymbol{\beta}_2 \in C \quad \Longrightarrow \quad \alpha\boldsymbol{\beta}_1 + (1-\alpha)\boldsymbol{\beta}_2 \in C, \quad \forall\ 0 \leq \alpha \leq 1.$$

- Visual:

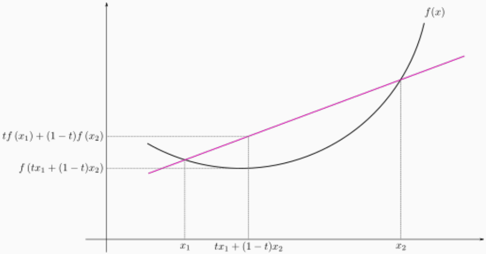


Convex Optimization

- What is a convex function $f : \mathbb{R}^p \to \mathbb{R}$?

$$f\left(\alpha \boldsymbol{\beta}_1 + (1-\alpha)\boldsymbol{\beta}_2\right) \leq \alpha f(\boldsymbol{\beta}_1) + (1-\alpha) f(\boldsymbol{\beta}_2) \quad \forall\ 0 \leq \alpha \leq 1.$$

- In Probability: Jensen's inequality
- Visual:



Convex functions

- To comply with notations in the literature, I will use $\mathbf{x}$ as the argument instead of using $\boldsymbol{\beta}$, and we are interested in the function $f(\mathbf{x})$.
- Examples of convex functions:
  - $\exp(x)$, $-\log(x)$, etc.
  - Affine: $a^{\mathsf{T}}\mathbf{x} + b$ is both convex and concave
  - Quadratic: $\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x} + b^{\mathsf{T}}\mathbf{x} + c$, if $\mathbf{A}$ is positive semidefinite.
  - All norms: ℓ_p
- A function is strictly convex if we can remove the equal sign:

$$f\big(\alpha\boldsymbol{\beta}_1 + (1-\alpha)\boldsymbol{\beta}_2\big) < \alpha f(\boldsymbol{\beta}_1) + (1-\alpha) f(\boldsymbol{\beta}_2) \quad \forall\, 0 \leq \alpha \leq 1.$$

- f is convex $\Longleftrightarrow -f$ is concave

Properties of Convex functions

- First-order property: If f is differentiable with convex domain, then f is convex iff

$$f(\mathbf{x}^*) \geq f(\mathbf{x}) + \triangledown f(\mathbf{x})^\mathsf{T}(\mathbf{x}^* - \mathbf{x})$$

- If we have a feasible point $\mathbf{x}$ with $\triangledown f(\mathbf{x}) = \mathbf{0}$, it means all alternative points $\mathbf{x}^*$ have larger function value $f(\mathbf{x}^*) \geq f(\mathbf{x})$.
- Hence, we call $\mathbf{x}$ a local minimizer. It may not be unique, but its as good as any other solution.
- Example: In a linear regression if we have linearly dependent columns in the design matrix. The solution of parameters is not unique.

Properties of Convex functions

- Second-order property: If f is twice differentiable with convex domain, then f is convex iff

$$\bigtriangledown^2 f(\mathbf{x}) \succeq 0 \quad \text{for any } \mathbf{x} \text{ in the domain,}$$

  where

$$\mathbf{H}(\mathbf{x}) = \bigtriangledown^2 f(\mathbf{x}) = \left(\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \right)$$

- $\mathbf{H}(\mathbf{x})$ is the Hessian matrix.
- If $\bigtriangledown^2 f(\mathbf{x}) \succ 0$ (positive definite), meaning f is strictly convex, then a local minimizer is also a global minimizer.
- Example: In linear regression when $\mathbf{X}^\mathsf{T}\mathbf{X} \succ 0$, i.e., invertible.

Solving convex problems

- In many situations, we just deal with an unconstrained, smooth convex function

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad f(\boldsymbol{\beta})$$

- OLS and ridge, etc. can all be formulated as this problem.
- Lasso is not smooth at some particular points of $\boldsymbol{\beta}$

Gradient descent

- One of the simplest algorithm is gradient descent.
- At any given point $\mathbf{x}$, we want to move it to the direction where f can decrease.
- Consider the Taylor expansion near $\mathbf{x}$:

$$
\begin{aligned}
& f(\mathbf{x}^*) \\
\approx\ & f(\mathbf{x}) + \triangledown f(\mathbf{x})^{\mathsf{T}}(\mathbf{x}^* - \mathbf{x}) + \frac{1}{2}(\mathbf{x}^* - \mathbf{x})^{\mathsf{T}}\mathbf{H}(\mathbf{x})(\mathbf{x}^* - \mathbf{x})
\end{aligned}
$$

- If we minimize this quadratic approximation, the new point that gives the smallest $f(\mathbf{x}^*)$ is

$$\mathbf{x}^* = \mathbf{x} - \mathbf{H}(\mathbf{x})^{-1} \triangledown f(\mathbf{x})$$

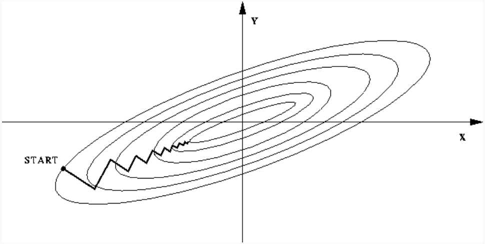
Gradient descent

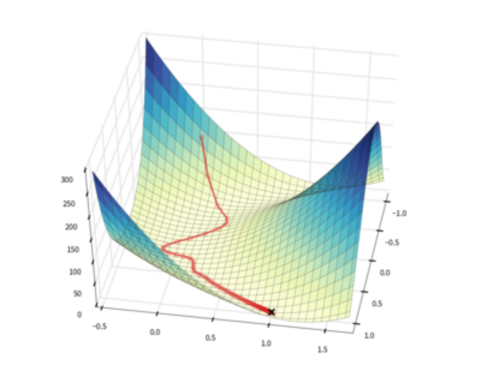
- Since calculating the second derivative (and inverse) $\mathbf{H}(\mathbf{x})$ can be difficult, lets just just use an identity matrix $\frac{1}{\delta}\mathbf{I}$.
- Then the new point is

$$\mathbf{x}^* = \mathbf{x} - \delta \bigtriangledown f(\mathbf{x})$$

- Gradient descent uses this updating scheme to iteratively archive smaller $f(\mathbf{x})$.
- However, we have to choose δ, which is know as the step size.
  - A step size too large may not even converge at all.
  - How about we just fix δ to be a small value, say 10^{-5}.
  - A step size too small will take many iterations to converge.
  - Line search is usually used. Sometimes, inexact line search.

Gradient descent





Newton-Raphson

- When we have explicit formula of the Hessian, we can simply compute them at each point $\mathbf{x}$ and follow the updating scheme

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{H}(\mathbf{x}^{(k)})^{-1} \bigtriangledown f(\mathbf{x}^{(k)})$$

  or, sometimes for numerical stability,

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \delta\, \mathbf{H}(\mathbf{x}^{(k)})^{-1} \bigtriangledown f(\mathbf{x}^{(k)})$$

- However, this is rare in practice since most objective functions are very complicated. Constrains may also creating difficulties.
- We can do this for ridge, but not Lasso.

Numerical approximation

- Sometimes even computing the gradient $\bigtriangledown f(\mathbf{x})$ is difficult.
- We can simply approximate them numerically (element-wise) by the definition

$$\bigtriangledown f(\mathbf{x})_j \approx \frac{f(\mathbf{x} + \delta \mathbf{e}_j) - f(\mathbf{x})}{\delta}$$

  where $\mathbf{e}_j$ is a vector with the jth element 1 and 0 everywhere else.
- Numerically approximating the gradient could be slow, depending on the size of the problem.

Numerical approximation

- In many cases, we have exact formula of the gradient, but want to speed things up with the second order information.
- You usually don't want to directly numerically approximate $\mathbf{H}(\mathbf{x})$ because its very very expensive.
- However, quasi-Newton methods, such as BFGS, progressively approximates $\mathbf{H}(\mathbf{x})^{-1}$ by utilizing the Sherman-Morrison formula.

  - Gradient descent can be viewed as using $\mathbf{H}(\mathbf{x})^{-1} = \mathbf{I}$
  - $\mathbf{x}^{(0)} \to \mathbf{x}^{(1)} \to \mathbf{x}^{(2)} \to \cdots \to \mathbf{x}^{(k)} \to \mathbf{x}^{(k+1)} \to$
  - Along this path, we computed $\to \bigtriangledown f(\mathbf{x}^{(k)}) \to \bigtriangledown f(\mathbf{x}^{(k+1)}) \to \cdots$
  - If we treat the function $f(\mathbf{x}^{(k)})$ locally as a quadratic function,

$$\bigtriangledown f(\mathbf{x}^{(k+1)}) - \bigtriangledown f(\mathbf{x}^{(k)}) \approx \mathbf{H}(\mathbf{x}^{(k)})(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)})$$

  - This leads to rank one/two updates of $\mathbf{H}$. Read DFP and BFGS references.

BFGS

Implementation

- If you have a smooth objective function, usually the log-likelihood $L(\mathbf{y}, \mathbf{X}, \boldsymbol{\beta})$, and we want to solve the parameters $\boldsymbol{\beta}$.
- You can utilize the `optim()` function in $\mathbb{R}$

```
> L <- function(b, X, Y) ...
> bhat = optim(rep(0, P), L, X = X, Y = Y, method = "BFGS")
```

- Sometimes, using a different initial value may be better.

None differentiable problems

- Non-differentiable problems become prevalent when we add penalties to the objective function
- Usually, these problems (at least the ones that we care about) are decomposable, meaning that

$$f(\mathbf{x}) = g(\mathbf{x}) + h(\mathbf{x})$$

  with differentiable g and non-differentiable h but still convex. For example:

$$\frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1$$

- There are other general approaches that can deal with this problem, but we will only look at the special case: Lasso.

Coordinate descent

- The Lasso problem has a special form, i.e., its separable:

$$f(\mathbf{x}) = g(\mathbf{x}) + \sum_{i=1}^{p} h(x_i)$$

- I will switch back to the "$\boldsymbol{\beta}$" notation for parameters:

$$f(\boldsymbol{\beta}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^\mathsf{T}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \sum_{i=1}^{p} |\beta_i|$$

- Instead of updating all parameters each step, we only update one parameter each step and loop over all parameters.

Coordinate descent

- The Gauss-Seidel style coordinate descent algorithm goes like, at the kth (grand) iteration:

$$
\begin{aligned}
\beta_1^{(k+1)} &= \underset{\beta_1}{\arg\min} \quad f(\beta_1, \beta_2^{(k)}, \ldots, \beta_p^{(k)}) \\
\beta_2^{(k+1)} &= \underset{\beta_2}{\arg\min} \quad f(\beta_1^{(k+1)}, \beta_2, \ldots, \beta_p^{(k)}) \\
&\ldots \\
\beta_p^{(k+1)} &= \underset{\beta_p}{\arg\min} \quad f(\beta_1^{(k+1)}, \beta_2^{(k+1)}, \ldots, \beta_p)
\end{aligned}
$$

- After we complete this loop, all β_j are updated to their new values, and we start over.

Coordinate descent

- The Jacobi style algorithm goes like, at the kth (grand) iteration:

$$\begin{aligned}
\beta_1^{(k+1)} &= \operatorname*{arg\,min}_{\beta_1} \quad f(\beta_1, \beta_2^{(k)}, \ldots, \beta_p^{(k)}) \\
\beta_2^{(k+1)} &= \operatorname*{arg\,min}_{\beta_2} \quad f(\beta_1^{(k)}, \beta_2, \ldots, \beta_p^{(k)}) \\
\ldots \\
\beta_p^{(k+1)} &= \operatorname*{arg\,min}_{\beta_p} \quad f(\beta_1^{(k)}, \beta_2^{(k)}, \ldots, \beta_p)
\end{aligned}$$

- After we complete this loop, update all β_j to their new values, and start over.
- Jacobi style algorithm can be computed in a parallel fashion, while Gauss-Seidel style can only be done sequentially.

Coordinate descent

- Two questions:
  1) is this going to be slower or faster than gradient descent?
  2) will it converge?
- Lets take a linear regression as an example (no penalty):

$$f(\boldsymbol{\beta}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^\mathsf{T}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

- At each iteration, for each β_i, lets fix all other parameters $\boldsymbol{\beta}_{(-j)}$.
- Suppose we do not know the Hessian matrix, the gradient descent goes like

$$\begin{aligned}\boldsymbol{\beta} &= \boldsymbol{\beta} - \delta \bigtriangledown f(\boldsymbol{\beta}) \\ &= \boldsymbol{\beta} - \delta\, \mathbf{X}^\mathsf{T}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})\end{aligned}$$

- This cost $O(np)$ flops.

Coordinate descent

- What about coordinate descent? It is a one-variable regression problem if we fix $\boldsymbol{\beta}_{(-j)}$:

$$f(\beta_j^{(k+1)}) = \frac{1}{2}\|\mathbf{y} - X_j\beta_j - \mathbf{X}_{(-j)}\boldsymbol{\beta}_{(-j)}^{(k)}\|_2^2$$

- Define $\mathbf{r} = \mathbf{y} - \mathbf{X}_{(-j)}\boldsymbol{\beta}_{(-j)}^{(k)} = \mathbf{y} - \mathbf{X}\boldsymbol{\beta}^{(k)} + X_j\beta_j^{(k)}$

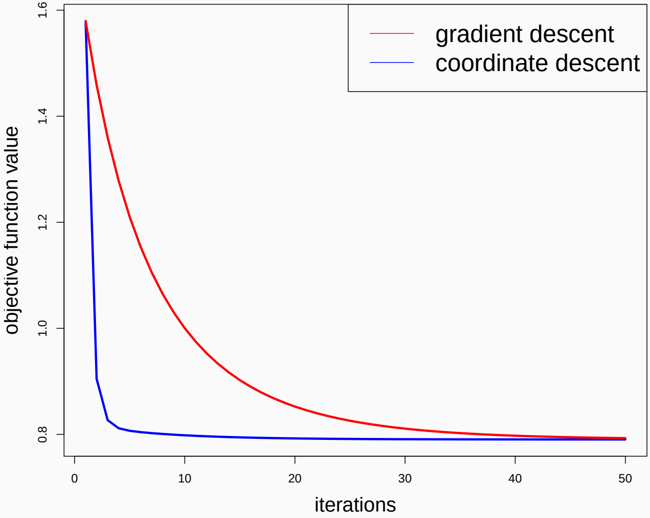
$$\beta_j^{(k+1)} = \frac{\mathbf{X}_j^\intercal \mathbf{r}}{\mathbf{X}_j^\intercal \mathbf{X}_j} = \frac{\mathbf{X}_j^\intercal(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}^{(k)})}{\mathbf{X}_j^\intercal \mathbf{X}_j} + \beta_j^{(k)}$$

- After updating β_j, put $X_j\beta_j^{(k+1)}$ back into $\mathbf{r}$, then subtract the effect from $(j+1)$ for the next update
- Usually this is proceeded with the Gauss-Seidel style.

Coordinate descent

- Intuition: we firstly take out all we currently explained ($\mathbf{X}\boldsymbol{\beta}$) from $\mathbf{y}$, save that as the residual, $\mathbf{r}$. Then we look in turn, how much extra each variable can explain.
- Updating each β_j cost $O(n)$ flops, then each iteration cost $O(np)$, same as gradient descent.
- What about their performances?

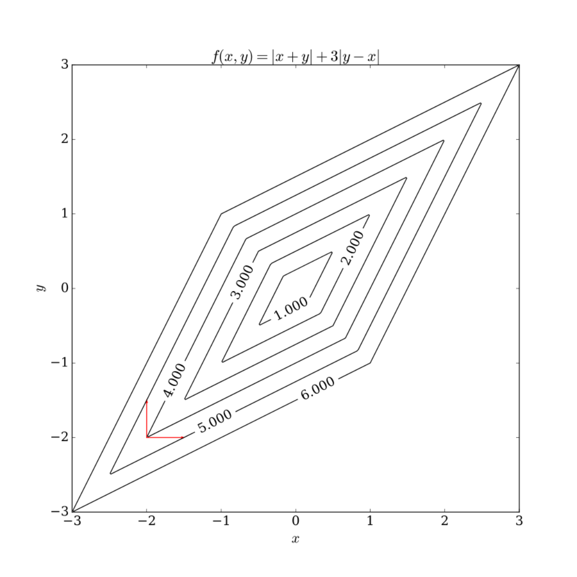
Coordinate descent



Coordinate descent

- What happened?
- Its not really a fair game: gradient descent only utilize the first order information, while coordinate descent updated each coordinate fully, at least for each iteration.
- When is coordinate descent useful/better?
  - If updating each coordinate is cheap, and maybe the solution is explicit (our lasso problem will be of this type).
  - More importantly, the problem has to be separable
- When will coordinate descent fail?

Coordinate descent



Coordinate descent for Lasso

- The Lasso solution can be obtained in the same way.
- Recall that the penalty is coordinate-wise, for each update, we are solving

$$f(\beta_j^{(k+1)}) = \frac{1}{2}\|\mathbf{y} - X_j\beta_j - \mathbf{X}_{(-j)}\boldsymbol{\beta}_{(-j)}^{(k)}\|_2^2 + \lambda|\beta_j|$$

- Again, this is a one-variable optimization problem, and we have the exact solution (see lecture note "Penalized"), which is a soft-threshold version of the OLS estimator.
- A path-wise algorithm starts with a large λ value, and run until no β can be moved anymore (converged), then reduce the λ by a factor and start over the update.
- This is how the glmnet package solves Lasso.