HW 7

Evan Ott
UT EID: eao466

October 31, 2016

Laplacian smoothing

(A)

FIXME: do this

(B)

The problem is

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \ \frac{1}{2}\|y - x\|_2^2 + \frac{\lambda}{2} x^\top L x = \frac{1}{2}\left(x^\top x - 2y^\top x + y^\top y + \lambda x^\top L x\right)$$

which we can find a solution for by taking the gradient w.r.t. x.

$$\begin{aligned} 0 &= \frac{1}{2}\left(2x - 2y + 0 + \lambda(L + L^\top)x\right) \\ &= \left(I + \frac{1}{2}\lambda(L + L^\top)\right)x - y \\ (I + \lambda L)\,\hat{x} &= y \end{aligned}$$

(C)

Gauss-Seidel

Solving $Ax = b$ in this framework amounts to splitting $A = L_* + U$ where L_* is the lower triangular matrix (including the diagonal) and U is the upper triangular matrix (excluding the diagonal).

The algorithm is then re-writing the problem iteratively as which can be solved with forward substitution.

$$L_* x^+ = b - Ux$$

See Barrett et al. (1994) §2.2.2 and Equation (2.6) (slightly different notation but same result).

Jacobi

Again solving $Ax = b$, we split into $A = D + R$ where D is just the diagonals and R is everything else (with 0 along the diagonal).

We then iterate

$$x^+ = D^{-1}(b - Rx)$$

What's nice here from an efficiency perspective is that D is easily invertible (so long as there are no zeros in the diagonal), and potentially linearizable depending on the underlying code implementation.

See Barrett et al. (1994) §2.2.1 and Equation (2.4) (slightly different notation but same result).

Conjugate Gradient

$Ax = b$ is equivalent to minimizing $\phi(x) = \frac{1}{2}x^\top Ax - b^\top x$ if A is symmetric.

$$\nabla\phi(x) = Ax - b \equiv r(x)$$

Then define $\{p_0, p_1, \ldots, p_n\}$ as being conjugate w.r.t. A, such that $p_i^\top Ap_j = 0$ when $i \neq j$.

Now the algorithm proceeds as

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \alpha_k p_k \\ \alpha_k &= -\frac{r_k^\top p_k}{p_k^\top Ap_k} \end{aligned}$$

So now we need to determine how to construct the p_k vectors. You could use eigenvectors, but those are in general pretty expensive to calculate. So now, let

$$\begin{aligned} p_0 &= -r_0 \\ p_k &= -r_k + \beta_k p_{k-1} \\ \beta_k &= \frac{r_{k+1}^\top r_{k+1}}{r_k^\top r_k} \end{aligned}$$

Notes from class

Next week: graph fused lasso.

Conjugate gradient is a lot more complicated, and is in fact solving a more general class of problems called Krylov subspace problems. And it is *fast* if the matrix falls into this Laplacian class of matrices (certain properties). In that case, as opposed to a standard matrix inverse, $O(n^3)$ or a sparse one, $O(n^2)$, it will actually be more like $O(n \ln n)$.

It turns out to be important to have a preconditioner. Solving $Ax = b$ is the same as solving $P^{-1}Ax = P^{-1}b$ where P is a "preconditioner." The closer A is to P (while P is still easy to invert), the closer $P^{-1}A$ is to the identity, making the problem trivial. The current state of the art is the *algebraic multigrid* which is that $O(n \ln n)$ type solution.

Notation for the rest of the notes:

$$\begin{aligned} C_\lambda \hat{x} &= y \\ \hat{x} &= C_\lambda^{-1} y \end{aligned}$$

so $\hat{x}$ is the smoothed/predicted y.

The question now is how to choose λ, potentially using C_p or AIC/BIC, cross-validation, etc.

The leave-one-out lemma (from Hastie et al. (2001)) allows us to calculate the LOOCV error. Assume $\hat{y} = Sy$ where $y, \hat{y} \in \mathbb{R}^n$ and S is a smoothing matrix (so the linear case we care about).

We need the degrees of freedom of an estimator/model (basically number of free parameters).

$$\begin{aligned} y &= X\beta + \epsilon \\ \hat{y} &= X\hat{\beta} \end{aligned}$$

has p degrees of freedom if $\beta \in \mathbb{R}^p$. We need to modify the definition to handle other cases.

Situations we need to address at a minimum:

1. Fit to p variables (as an example, OLS)

2. Choose p from $D > p$ candidate variables, then find the best fit for these p. This should have more degrees of freedom if our definition is supposed to make sense at all.

Suppose we have $\hat{y}$ such that $\hat{y}_i = g_i(y)$. Then

$$\begin{aligned} \mathrm{df}(\hat{y}) &= \frac{1}{\sigma^2}\sum_{i=1}^{n}\mathrm{cov}(\hat{y}_i, y_i) \\ \sigma^2 &= \mathrm{var}(y_i) \end{aligned}$$

Case: extreme overfitting: $\hat{y}_i = y_i$. Then

$$\mathrm{df}(\hat{y}) = \frac{1}{\sigma^2}\sum_{i=1}^{n}\mathrm{var}(y_i) = n$$

Case: extreme underfitting: $\hat{y}_i = \overline{y}$ Then

$$\begin{aligned} \mathrm{df}(\hat{y}) &= \frac{1}{\sigma^2}\sum_{i=1}^{n}\mathrm{cov}(\overline{y}, y_i) \\ \mathrm{cov}(\overline{y}, y_i) &= \mathrm{cov}\left(\frac{1}{N}y_i + \frac{1}{N}\sum_{j\neq i} y_j, y_i\right) \\ &= \mathrm{cov}\left(\frac{1}{N}y_i, y_i\right) = \frac{1}{N}\sigma^2 \\ \mathrm{df}(\hat{y}) &= \frac{1}{\sigma^2}\sum_{i=1}^{n}\frac{1}{N}\sigma^2 = 1 \end{aligned}$$

Case: linear smoothers: $\hat{y} = Sy$ then $\mathrm{df}(\hat{y}) = \mathrm{trace}(S)$.

Graph fused lasso

Switching to an ADMM framework, we have

$$\begin{aligned} &\underset{x\in\mathbb{R}^n}{\text{minimize}}\ \frac{1}{2}\|y - x\|_2^2 + \lambda\|r\|_1 \\ &\text{subject to } Dx - r = 0 \end{aligned}$$

References

Richard Barrett, Michael W Berry, Tony F Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst. *Templates for the solution of linear systems: building blocks for iterative methods*, volume 43. Siam, 1994. http://www.netlib.org/templates/templates.pdf.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The elements of statistical learning: data mining, inference and prediction. *New York: Springer-Verlag*, 1(8):371–406, 2001. http://statweb.stanford.edu/~tibs/ElemStatLearn/.