HW 7

Evan Ott
UT EID: eao466

November 9, 2016

Laplacian smoothing

(A)

FIXME: do this

(B)

The problem is

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

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

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

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

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

Notes from class 10/31

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:

$$C_\lambda \hat{x} = y$$
$$\hat{x} = C_\lambda^{-1} y$$

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).

$$y = X\beta + \epsilon$$
$$\hat{y} = X\hat{\beta}$$

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 $\mathrm{df}(\hat{y}) = \mathrm{trace}(S)$.

Picking up notes from last class

Laplacian smoothing should have degrees of freedom between these two extremes. For linear smoothers, $\hat{y} = Sy$ where $y, \hat{y} \in \mathbb{R}^n$. Then (where S_i is the i-th row of S):

$$\begin{aligned} \hat{y}_i &= \langle S_i, y \rangle \\ \mathrm{cov}(\hat{y}_i, y_i) &= \mathrm{cov}\left(\sum_{j=1}^{n} S_{ij} y_j, y_i \right) \\ &= \mathrm{cov}\left(S_{ii} y_i + \sum_{j \neq i} S_{ij} y_j, y_i \right) \\ &= \mathrm{cov}\left(S_{ii} y_i, y_i \right) \\ &= S_{ii} \mathrm{cov}(y_i, y_i) = S_{ii} \sigma^2 \\ \mathrm{df}(\hat{y}) &= \frac{1}{\sigma^2} \sum_{i=1}^{n} S_{ii} \sigma^2 = \mathrm{trace}(S) \end{aligned}$$

Case: linear least squares: $\hat{y} = X\beta + \epsilon$, where $\hat{\beta} = (X^\top X)^{-1} X^\top y$.

In linear regression, we would expect this to have p degrees of freedom. This is a linear smoother itself, as

$$\hat{y} = X\hat{\beta} = X(X^\top X)^{-1} X^\top y$$

So $\mathrm{df}(\hat{y}) = \mathrm{trace}(X(X^\top X)^{-1} X^\top)$. This is a projection matrix, so the trace is the rank, and the rank is the dimension of the underlying subspace, and this is p. Or, we can use the properties of trace.

$$\begin{aligned}\text{trace}(ABC) &= \text{trace}(CAB)\\ \text{trace}(X(X^\top X)^{-1}X^\top) &= \text{trace}(X^\top X(X^\top X)^{-1})\\ &= \text{trace}(I_p) = p\end{aligned}$$

It would be nice to be able to calculate the degrees of freedom of the lasso, but this result is hard to use in practice. Instead, we can use Stein's lemma. This shows that for a pretty general class of estimators, we can instead use an alternate formula to compute the degrees of freedom and it is equivalent to the formula above. See Zou et al. (2007) to see where this lemma is used to find the degrees of freedom of lasso.

The more philosophical question is how the degrees of freedom of linear regression and lasso are both p (where p for the lasso are the non-zero coefficients). More-or-less, this is because lasso also shrinks the variables it estimates. So the extra variables in lasso that can be selected cause an increase in degrees of freedom that is exactly canceled because of shrinkage.

Why do we even need the degrees of freedom? For example, it's for model selection using C_p or AIC/BIC, etc.

Graph-fused lasso

The degrees of freedom in this problem is the number of distinct constant regions in the fitted $\hat{x}$.

In terms of actually solving the problem, we have

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

Doing the normal ADMM will be fine, but you'll need to refactor a matrix every time. In the advanced version, we start from this same constrained optimization problem but move the constraint back into the objective.

$$\begin{aligned}\underset{x\in\mathbb{R}^n}{\text{minimize}}\ & \frac{1}{2}\|y-x\|_2^2 + \lambda\|r\|_1 + I_C(x,r)\\ I_C(x,r) &= \begin{cases}\infty & Dx \neq r\\ 0 & \text{o.w.}\end{cases}\end{aligned}$$

So now, we'll extend this to have have a slack variable for x and one for r.

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

However, it turns out that the x update is the most computationally expensive part. In the original ADMM, the x update is generically $A^{(t)}x = b^{(t)}$ but in this final form, it is $Ax = b^{(b)}$ so we can re-use the same matrix decomposition for the x update. At this point, all the updates will be a lot more cheap, including the x update, so everything will be considerably faster.

Graph fused lasso

Switching to an ADMM framework, we have

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

However, in this case, we can re-write the ADMM using additional slack variables as noted in the notes in the previous section. For additional information, see §5 in Tansey et al. (2014). So now, use the following:

$$\begin{aligned}\text{minimize } & \frac{1}{2}\|y-x\|_2^2 + \lambda\|r\|_1 + I_C(z,s) \\ \text{subject to } & x - z = 0 \wedge r - s = 0\end{aligned}$$

to which we introduce scaled slack variables u for $x - z = 0$ and t for $r - s = 0$:

$$\text{minimize } \frac{1}{2}\|y-x\|_2^2 + \lambda\|r\|_1 + I_C(z,s) + \frac{a}{2}\|x-z+u\|_2^2 + \frac{a}{2}\|r-s+t\|_2^2$$

x update

The augmented Lagrangian is separable in x_i so for each x_i we have:

$$\begin{aligned}\underset{x_i\in\mathbb{R}}{\text{minimize}} \ & \frac{1}{2}(y_i - x_i)^2 + \frac{a}{2}(x_i - z_i + u_i)^2 \\ 0 &= x_i - y_i + a(x_i - z_i + u_i) \\ \hat{x}_i &= \frac{1}{1+a}\left(y_i + az_i - au_i\right)\end{aligned}$$

Or as a vector,

$$\hat{x} = \frac{1}{1+a}\left(y + az - au\right)$$

r update

The augmented Lagrangian is also separable in r_i so we have:

$$\underset{r_i\in\mathbb{R}}{\text{minimize}} \ \lambda|r_i| + \frac{a}{2}\left(r_i - s_i + t_i\right)^2$$

As shown many times before, the solution is simply the soft-thresholding function:

$$\hat{r}_i = S_{\lambda/a}\left(s_i - t_i\right)$$

z, s updates

Because of the indicator function, z and s must be considered jointly (indeed, as vectors, not as elements because the constraint can't in general be checked without all values).

$$\underset{z\in\mathbb{R}^n,\ s\in\mathbb{R}^m}{\text{minimize}} \ I_C(z,s) + \frac{a}{2}\|x-z+u\|_2^2 + \frac{a}{2}\|r-s+t\|_2^2$$

We again play tricks and move the indicator part of the objective back to a constraint to remove s from the objective.

$$\begin{aligned}\underset{z\in\mathbb{R}^n}{\text{minimize}} \ & \frac{a}{2}\|x-z+u\|_2^2 + \frac{a}{2}\|r-Dz+t\|_2^2 \\ \text{subject to } & Dz - s = 0\end{aligned}$$

First, simplify notation such that $w = x + u$ and $v = r + t$. The objective can now be re-written as:

$$\frac{a}{2}\left[z^\top z - 2w^\top z + w^\top w + z^\top D^\top Dz - 2v^\top Dz + v^\top v\right]$$

which is simply a quadratic form in z, so

$$\begin{aligned}0 &= \frac{a}{2}\left[2z - 2w + 2D^\top Dz - 2D^\top v\right] \\ 0 &= (I + D^\top D)z - w - D^\top v \\ \hat{z} &= (I + D^\top D)^{-1}(w + D^\top v) = (I + D^\top D)^{-1}\left(x + u + D^\top[r+t]\right)\end{aligned}$$

Then set $\hat{s} = D\hat{z}$.

u, t updates

Finally, the scaled slack variables are updated as:

$$\hat{u} = x - z + u$$
$$\hat{t} = r - s + t$$

All updates together

Placing everything together, we have:

$$\begin{aligned}
x^{(k+1)} &= \frac{1}{1+a}\left(y + az^{(k)} - au^{(k)}\right) \\
r_i^{(k+1)} &= S_{\lambda/a}\left(s_i^{(k)} - t_i^{(k)}\right) \quad i \in \{1, \ldots, m\} \\
z^{(k+1)} &= (I + D^\top D)^{-1}\left(x^{(k+1)} + u^{(k)} + D^\top[r^{(k+1)} + t^{(k)}]\right) \\
s^{(k+1)} &= Dz^{(k+1)} \\
u^{(k+1)} &= x^{(k+1)} - z^{(k+1)} + u^{(k)} \\
t^{(k+1)} &= r^{(k+1)} - s^{(k+1)} + t^{(k)}
\end{aligned}$$

Convergence

I used the convergence criteria spelled out in Boyd et al. (2011) §3.3.1 with $\epsilon^{\text{abs}} = 10^{-8}$ and $\epsilon^{\text{rel}} = 10^{-5}$

Notes from class 11/7

Problem: James originally said that the original ADMM version would require re-factorizing the matrix for the x update, but that ends up only being the case if you have a weighted form $\frac{1}{2}\left\|W^{1/2}(y-x)\right\|_2^2$ where W may depend on the iteration number (such as being a function of x). In this case, that matrix is just the just the identity, so it's fine.

To do cross validation, basically, just throw out things in the L2 norm term, but keep them in the L1 term. Preserves smoothing.

Can re-write the objective as

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\ \frac{1}{2}\|y-x\|_2^2 + \lambda \sum_{(i,j)\in E} |x_i - x_j|$$

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\ \frac{1}{2}\|y-x\|_2^2 + \lambda \sum_{(i,j)\in R} |x_i - x_j| + \lambda \sum_{(i,j)\in C} |x_i - x_j|$$

$$R \cup C = E, \quad \text{R is edges along rows, C is edges along cols}$$

We then split along the edges in row 1, row 2, up to row D. Pretend we didn't have the column terms just for a moment. We could solve each row in parallel then. There's an algorithm for doing this chain graph-fused lasso that's similar to hidden Markov model Kalman filtering algorithms and it runs in linear time.

So, we just use a new variable, z for the column terms and add the constraint $z = x$. So, we just do ADMM. This is proximal stacking as in Barbero and Sra (2014). James has code on the class repo to show how to use it.

Notes from class 11/9

"Data-analysis applications of the fused lasso and related spatial smoothers" presentation.

Recurring problem: $\underset{\beta}{\text{minimize}}\ l(\beta) + \lambda\|D\beta\|_1$.

Toy example of chain graph fused lasso shows that coefficients get pulled toward the mean, may have a lot more levels than the underlying truth. In a way, it's shrinking the estimate of the *jumps* between segments.

Trend filtering

$$\underset{\beta \in \mathbb{R}^N}{\text{minimize}} \ \frac{1}{2}\|y-\beta\|_2^2 + \lambda \left\| D^{(k+1)}\beta \right\|_1$$

So $D^{(k+1)}$ is a discrete analog of the order-k derivative. Each one is like the "differences of the differences (of the differences ...)." You plug in the order of the derivative that you care about, so you're imposing a form on β. $k = 0$ is piecewise constants. $k = 1$ is piecewise linear. And so forth. There's a little subtlety because you lose a row each time you increase k.

This is different than splines. Splines are piecewise polynomial functions such that you get derivatives at all of the breakpoints up to the order of the polynomial minus 1. Why is this different than regression splines? In regression splines, you pick the breakpoints then fit by least squares. In the graph-fused lasso, you can change the regression at *any* data point, rather than selecting the breakpoints ahead of time. You also select the breakpoints jointly. You also need basis splines such as b-splines. In this problem, you get a falling-factorial basis which is related but not the same basis.

There's a fast way of solving the trend filtering, see Ramdas and Tibshirani (2015). See also `glmgen` on GitHub for implementation of this.

Can generalize to a general graph instead of a chain. See Wang et al. (2015) for how to generate the D matrix in the general case.

Spatially aware anomaly detection

This I think is partially Alex Reinhart's work with the radiation detector. In the presence of an anomaly, not only is the background rate going to change, but the energy spectrum is going to change. One simple thing to do to test is comparing the background (known) and the new empirical distribution with a Kolmogorov-Smirnoff difference of distribution test. ☢

Recursive dyadic partitions. Instead of taking just bins, split the space in half and count each side. Recursively split.

References

Álvaro Barbero and Suvrit Sra. Modular proximal optimization for multidimensional total-variation regularization. *arXiv preprint arXiv:1411.0589*, 2014. URL https://arxiv.org/pdf/1411.0589v2.pdf.

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.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. 3(1):1–122, 2011. http://stanford.edu/~boyd/papers/pdf/admm_distr_stats.pdf.

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

Aaditya Ramdas and Ryan J Tibshirani. Fast and flexible admm algorithms for trend filtering. *Journal of Computational and Graphical Statistics*, (just-accepted), 2015. URL http://www.tandfonline.com/doi/pdf/10.1080/10618600.2015.1054033.

Wesley Tansey, Oluwasanmi Koyejo, Russell A Poldrack, and James G Scott. False discovery rate smoothing. 2014. https://arxiv.org/pdf/1411.6144v1.pdf.

Yu-Xiang Wang, James Sharpnack, Alexander J Smola, and Ryan J Tibshirani. Trend filtering on graphs. In *AISTATS*, 2015. URL http://www.jmlr.org/proceedings/papers/v38/wang15d.pdf.

Hui Zou, Trevor Hastie, Robert Tibshirani, et al. On the "degrees of freedom" of the lasso. 35(5):2173–2192, 2007. http://projecteuclid.org/download/pdfview_1/euclid.aos/1194461726.