Case Study: Generalized Lasso Problems

Ryan Tibshirani
Convex Optimization 10-725/36-725
So far we’ve learned about:

- First-order methods
- Newton/quasi-Newton methods
- Interior point methods

These comprise a good part of the core tools in optimization, and are a big focus in this field

(Still, there’s a lot more out there. Before the course is over we’ll cover dual methods and ADMM, coordinate descent, proximal and projected Newton ...)

Given the number of available tools, it may seem overwhelming to choose a method in practice. A fair question: how to know what to use when?
It’s not possible to give a complete answer to this question. But the big algorithms table from last time gave guidelines. It covered:

- Assumptions on criterion function
- Assumptions on constraint functions/set
- Ease of implementation (how to choose parameters?)
- Cost of each iteration
- Number of iterations needed

Other important aspects, that it didn’t consider: parallelization, data storage issues, statistical interplay

Here, as any example, we walk through some of the high-level reasoning for related but distinct generalized lasso problem cases
Generalized lasso problems

Consider the problem

$$\min_\beta f(\beta) + \lambda \|D\beta\|_1$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth, convex function and $D \in \mathbb{R}^{m \times n}$ is a penalty matrix. This is called a generalized lasso problem.

The usual lasso, $D = I$, encodes sparsity in solution $\hat{\beta}$, while the generalized lasso encodes sparsity in

$$D\hat{\beta} = \begin{pmatrix} D_1\hat{\beta} \\ \vdots \\ D_m\hat{\beta} \end{pmatrix}$$

where $D_1, \ldots, D_m$ are the rows of $D$. This can result in interesting structure in $\hat{\beta}$, depending on choice of $D$. 

Outline

Today:

- Notable examples
- Algorithmic considerations
- Back to examples
- Implementation tips
Fused lasso or total variation denoising, 1d

Special case: fused lasso or total variation denoising in 1d, where

\[ D = \begin{bmatrix}
-1 & 1 & 0 & \ldots & 0 & 0 \\
0 & -1 & 1 & \ldots & 0 & 0 \\
\vdots & & & \ddots & & \\
0 & 0 & 0 & \ldots & -1 & 1 \\
\end{bmatrix} , \text{ so } \| D \beta \|_1 = \sum_{i=1}^{n-1} | \beta_i - \beta_{i+1} | \]

Now we obtain sparsity in adjacent differences \( \hat{\beta}_i - \hat{\beta}_{i+1} \), i.e., we obtain \( \hat{\beta}_i = \hat{\beta}_{i+1} \) at many locations \( i \)

Hence, plotted in order of the locations \( i = 1, \ldots n \), the solution \( \hat{\beta} \) appears piecewise constant
Examples:

**Gaussian loss**

\[
f(\beta) = \frac{1}{2} \sum_{i=1}^{n} (y_i - \beta_i)^2
\]

**Logistic loss**

\[
f(\beta) = \sum_{i=1}^{n} (-y_i \beta_i + \log(1 + e^{\beta_i}))
\]
Higher order polynomials fits are possible too. These are called trend filtering methods in 1d, i.e.,

\[
D = \begin{bmatrix}
1 & -2 & 1 & \ldots & 0 \\
0 & 1 & -2 & \ldots & 0 \\
\vdots \\
0 & 0 & 0 & \ldots & 1
\end{bmatrix} \quad \text{or} \quad D = \begin{bmatrix}
-1 & 3 & -3 & 1 & \ldots & 0 \\
0 & 1 & -3 & 3 & \ldots & 0 \\
\vdots \\
0 & 0 & 0 & 0 & \ldots & 1
\end{bmatrix}
\]

so \( \|D\beta\|_1 = \sum_{i=1}^{n-2} |\beta_i - 2\beta_{i+1} + \beta_{i+2}| \)

or \( \|D\beta\|_1 = \sum_{i=1}^{n-3} |\beta_i - 3\beta_{i+1} + 3\beta_{i+2} - \beta_{i+3}| \)

The first penalty gives piecewise linear solution \( \hat{\beta} \), and the second gives a piecewise quadratic
Examples:

Gaussian loss, linear trend
\[ f(\beta) = \frac{1}{2} \sum_{i=1}^{n} (y_i - \beta_i)^2 \]

Poisson loss, quadratic trend
\[ f(\beta) = \sum_{i=1}^{n} (-y_i \beta_i + e^{\beta_i}) \]
Fused lasso or total variation denoising, graphs

Special case: fused lasso or total variation denoising over a graph, \( G = (\{1, \ldots n\}, E) \). Here \( D \) is \(|E| \times n\), and if \( e_\ell = (i, j) \), then \( D \) has \( \ell \)th row

\[
D_\ell = (0, \ldots -1, \ldots 1, \ldots 0)
\]

so

\[
\|D\beta\|_1 = \sum_{(i, j) \in E} |\beta_i - \beta_j|
\]

Now at the solution, we get \( \hat{\beta}_i = \hat{\beta}_j \) across many edges \((i, j) \in E\), so \( \hat{\beta} \) is piecewise constant over the graph \( G \).
Example: Gaussian loss, $f(\beta) = \frac{1}{2} \sum_{i=1}^{n} (y_i - \beta_i)^2$, 2d grid graph

Data (noisy image)  Solution (denoised image)
Example: Gaussian loss, \( f(\beta) = \frac{1}{2} \sum_{i=1}^{n} (y_i - \beta_i)^2 \), Chicago graph

Data (observed crime rates)  Solution (estimated crime rates)
Problems with a big dense $D$

Special case: in some problems we encounter a big dense operator $D$, whose structure might as well be considered arbitrary.

E.g., we might have collected measurements that we know should lie mostly orthogonal to the desired estimate $\hat{\beta}$, and we stack these along the rows of $D$, known as analyzing operator in this setup.

E.g., equality-constrained lasso problems also fit into this case, as

$$\min_{\beta} f(\beta) + \lambda \|\beta\|_1 \quad \text{subject to} \quad A\beta = 0$$

can be reparametrized by letting $D \in \mathbb{R}^{n \times r}$ have columns to span $\text{null}(A)$. Then $A\beta = 0 \iff \beta = D\theta$ for some $\theta \in \mathbb{R}^r$, and the above is equivalent to

$$\min_{\theta} f(D\theta) + \lambda \|D\theta\|_1$$
Generalized lasso algorithms

Let's go through our toolset, to figure out how to solve

$$\min_{\beta} f(\beta) + \lambda \|D\beta\|_1$$

Subgradient method: subgradient of criterion is

$$g = \nabla f(\beta) + \lambda D^T \gamma$$

where $$\gamma \in \partial \|x\|_1$$ evaluated at $$x = D\beta$$, i.e.,

$$\gamma_i \in \begin{cases} \{\text{sign}((D\beta)_i)\} & \text{if } (D\beta)_i \neq 0 \\ [-1, 1] & \text{if } (D\beta)_i = 0 \end{cases}, \quad i = 1, \ldots, m$$

Downside (as usual) is that convergence is slow. Upside is that $$g$$ is easy to compute (provided $$\nabla f$$ is): if $$S = \text{supp}(D\beta)$$, then we let

$$g = \nabla f(\beta) + \lambda \sum_{i \in S} \text{sign}((D\beta)_i) \cdot D_i$$
Proximal gradient descent: prox operator is

$$\text{prox}_t(\beta) = \arg\min_z \frac{1}{2t} \|\beta - z\|_2^2 + \lambda \|Dz\|_1$$

This is not easy for a generic $D$ (compare soft-thresholding, when $D = I$). Actually, this is a highly nontrivial optimization problem, even when $D$ is structured (e.g., Gaussian trend filtering)

Could try reparametrizing the term $\|D\beta\|_1$ to make it linear, while introducing inequality constraints. We could then apply an interior point method

But we will have better luck going to the dual problem. (In fact, it is never a bad idea to look at the dual problem, even if you have a good approach for the primal problem!)
Generalized lasso dual

Our problems are

\[
\text{Primal} : \min_{\beta} f(\beta) + \lambda \|D\beta\|_1
\]

\[
\text{Dual} : \min_u f^*(-D^Tu) \text{ subject to } \|u\|_\infty \leq \lambda
\]

Here \( f^* \) is the conjugate of \( f \). Note that \( u \in \mathbb{R}^m \) (where \( m \) is the number of rows of \( D \)) while \( \beta \in \mathbb{R}^n \)

The primal and dual solutions \( \hat{\beta}, \hat{u} \) are linked by KKT conditions:

\[
\nabla f(\hat{\beta}) + D^T\hat{u} = 0, \quad \text{and}
\]

\[
\hat{u}_i \in \begin{cases} 
\{\lambda\} & \text{if } (D\hat{\beta})_i > 0 \\
\{-\lambda\} & \text{if } (D\hat{\beta})_i < 0, \quad i = 1, \ldots m \\
[-\lambda, \lambda] & \text{if } (D\hat{\beta})_i = 0 
\end{cases}
\]

Second property implies that: \( \hat{u}_i \in (-\lambda, \lambda) \implies (D\hat{\beta})_i = 0 \)
Let’s go through our toolset, to think about solving dual problem

$$\min_u f^*( -D^T u ) \text{ subject to } \|u\|_\infty \leq \lambda$$

Note the eventually we’ll need to solve $$\nabla f(\hat{\beta}) = -D^T \hat{u}$$ for primal solution, and tractability of this depends on $$f$$

**Proximal gradient descent:** looks much better now, because prox is

$$\text{prox}_t(u) = \arg\min_z \frac{1}{2t} \|u - z\|_2^2 \text{ subject to } \|z\|_\infty \leq \lambda$$

is easy. This is projection onto a box $$[-\lambda, \lambda]^m$$, i.e., prox returns $$\hat{z}$$ with

$$\hat{z}_i = \begin{cases} 
\lambda & \text{if } u_i > \lambda \\
-\lambda & \text{if } u_i < -\lambda \\
u_i & \text{if } u_i \in [-\lambda, \lambda]
\end{cases}, \quad i = 1, \ldots m$$
**Interior point method:** rewrite dual problem as

$$
\min_u f^*(-D^T u) \text{ subject to } -\lambda \leq u_i \leq \lambda, \ i = 1, \ldots m
$$

These are just linear constraints, so we can easily form log barrier\(^1\) as in

$$
\min_u t \cdot f^*(-D^T u) + \phi(u)
$$

where

$$
\phi(u) = -\sum_{i=1}^{m} \left( \log(\lambda - u_i) + \log(u_i + \lambda) \right)
$$

We either solve above problem with Newton’s method, or take one Newton step, and then increase \(t\)

How efficient are Newton updates?

---

\(^1\)There could be extra constraints from the domain of \(f^*\), e.g., this happens when \(f\) is the logistic loss, so these add extra log barrier terms
Define the barrier-smoothed dual criterion function

\[ F(u) = tf^*(-D^Tu) + \phi(u) \]

Newton updates follow direction \( H^{-1}g \), where

\[ g = \nabla F(u) = -t \cdot D(\nabla f^*(-D^Tu)) + \nabla \phi(u) \]

\[ H = \nabla^2 F(u) = t \cdot D(\nabla^2 f^*(-D^Tu))D^T + \nabla^2 \phi(u) \]

How difficult is it to solve a linear system in \( H \)?

- First term: if Hessian off the loss term \( \nabla^2 f^*(v) \) is structured, and \( D \) is structured, then often \( D\nabla^2 f^*(v)D^T \) is structured
- Second term: Hessian of log barrier term \( \nabla^2 \phi(u) \) is diagonal

So it really depends critically on first term, i.e., on conjugate loss \( f^* \) and penalty matrix \( D \)
Putting it all together:

- **Primal subgradient method:** iterations are cheap (we sum up rows of $D$ over active set $S$), but convergence is slow

- **Primal proximal gradient:** iterations involve evaluating

  $$\text{prox}_t(\beta) = \arg\min_z \frac{1}{2t} \|\beta - z\|_2^2 + \lambda \|Dz\|_1$$

  which can be very expensive, convergence is medium

- **Dual proximal gradient:** iterations involve projecting onto a box, so very cheap, convergence is medium

- **Dual interior point method:** iterations involve a solving linear system in

  $$Hx = g$$

  $$\begin{align*}
  H &= t \cdot D(\nabla^2 f^*(-D^T u))D^T + \nabla^2 \phi(u) \\
  \intertext{which may or may not be expensive, convergence is rapid} 
  \end{align*}$$
Suppose that we are studying the linear trend filtering problem, so

\[
D = \begin{bmatrix}
1 & -2 & 1 & \ldots & 0 & 0 \\
0 & 1 & -2 & \ldots & 0 & 0 \\
\vdots \\
0 & 0 & 0 & \ldots & -2 & 1
\end{bmatrix},
\]

and the loss is either Gaussian

\[
f(\beta) = \frac{1}{2} \sum_{i=1}^{n} (y_i - \beta_i)^2,
\]

or logistic

\[
f(\beta) = \sum_{i=1}^{n} (-y_i \beta_i + \log(1 + e^{\beta_i}))
\]

Suppose further that we desire solution at a high level of accuracy, otherwise, we notice “wiggles” when plotting \(\hat{\beta}\)

What algorithm should we use?
Primal subgradient and primal proximal gradient are out (slow and intractable, respectively)

As for dual algorithms, one can check that the conjugate $f^*$ has a closed-form for both the Gaussian and logistic cases:

$$f^*(v) = \frac{1}{2} \sum_{i=1}^{n} y_i^2 - \frac{1}{2} \sum_{i=1}^{n} (y_i + v_i)^2 \quad \text{and}$$

$$f^*(v) = \sum_{i=1}^{n} \left((v_i + y_i) \log(v_i + y_i) + (1 - v_i - y_i) \log(1 - v_i - y_i)\right)$$

respectively. We also have the expressions for primal solutions

$$\hat{\beta} = y - D^T \hat{u} \quad \text{and}$$

$$\hat{\beta}_i = -y_i \log \left(y_i (D^T \hat{u})_i\right) + y_i \log \left(1 - y_i (D^T \hat{u})_i\right), \quad i = 1, \ldots, n$$

respectively.
Dual proximal gradient descent admits very efficient iterations, as it just projects $u + tD\nabla f^*(-D^Tu)$ onto a box, repeatedly. But it takes far too long to converge to high accuracy: even more so than usual, because it suffers from poor conditioning of $D$

(Here $k = 0$: operator for fused lasso, $k = 1$: linear trend filtering, $k = 2$: quadratic trend filtering)
Importantly, $\nabla^2 f^*(v)$ is a \textit{diagonal} matrix in both the Gaussian and logistic cases:

$$\nabla^2 f^*(v) = I \quad \text{and}$$

$$\nabla^2 f^*(v) = \text{diag}\left(\frac{1}{v_i + y_i} + \frac{1}{1 - v_i - y_i}, \ i = 1, \ldots m\right)$$

respectively. Therefore the Newton steps in a \textit{dual interior point method} involve solving a linear system $Hx = g$ in

$$H = DA(u)D^T + B(u)$$

where $A(u), B(u)$ are both diagonal. This is a \textit{banded} matrix, and so these systems can be solved very efficiently, in $O(n)$ flops.

Hence, an interior point method on the dual problem is the way to go: \textit{cheap iterations}, and convergence to high accuracy is very fast.
Recall example from our first lecture:

Dual interior point method
20 iterations

Dual proximal gradient
10,000 iterations
Essentially the same story holds when $D$ is the fused lasso operator on an arbitrary graph:

- Primal subgradient is slow, primal prox is intractable
- Dual prox is cheap to iterate, but slow to converge
- Dual interior point method solves structured linear systems, so its iterations are efficient, and is preferred

**Dual interior point method** repeatedly solves $Hx = g$, where

$$H = DA(u)D^T + B(u)$$

and $A(u), B(u)$ are both diagonal. This no longer banded, but it is highly structured: $D$ is the edge incidence matrix of the graph, and $L = D^T D$ the graph Laplacian

But the story can suddenly change, with a tweak to the problem!
Consider the same $D$, and the same Gaussian and logistic losses, but with **regressors** or predictors $x_i \in \mathbb{R}^p$, $i = 1, \ldots, n$,

\[
f(\beta) = \frac{1}{2} \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 \quad \text{and} \quad f(\beta) = \sum_{i=1}^{n} ( - y_i x_i^T \beta + \log(1 + \exp(x_i^T \beta))
\]

respectively. E.g., if the predictors are connected over a graph, and $D$ is the graph fused lasso matrix, then the estimated coefficients will be constant over regions of the graph.

Assume that the predictors values are arbitrary. Everything in the dual is more complicated now.
Denote $X \in \mathbb{R}^{n \times p}$ as the predictor matrix (rows $x_i, i = 1, \ldots n$), and $f(\beta) = h(X\beta)$ as the loss. Our problems are

Primal: \[ \min_{\beta} h(X\beta) + \lambda \|D\beta\|_1 \]

Dual: \[ \min_{u,v} h^*(v) \]

subject to \[ X^T v + D^T u = 0, \|u\|_\infty \leq \lambda \]

Here $h^*$ is the conjugate of $h$. Note that we have $u \in \mathbb{R}^m$, $v \in \mathbb{R}^p$. Furthermore, the primal and dual solutions $\hat{\beta}$ and $\hat{u}, \hat{v}$ satisfy

\[ \nabla h(X\hat{\beta}) - \hat{v} = 0 \quad \text{or equivalently} \]

\[ X^T \nabla h(X\hat{\beta}) + D^T \hat{u} = 0 \]

Computing $\hat{\beta}$ from the dual requires solving a \textbf{linear system in $X$}, very expensive for generic $X$.
Dual proximal gradient descent has become intractable, because the prox operator is

$$\text{prox}_t(u, v) = \arg\min_{X^Tw + D^Tz = 0} \frac{1}{2t} \|u - z\|_2^2 + \frac{1}{2t} \|v - w\|_2^2 + \|u\|_\infty$$

This is finding the projection of \((u, v)\) onto the intersection of a plane and a (lower-dimensional) box.

Dual interior point methods also don’t look nearly as favorable as before, because the equality constraint

$$X^Tv + D^Tu = 0$$

must be maintained, so we augment the inner linear systems, and this ruins their structure, since \(X\) is assumed to be dense.

Primal subgradient method is still very slow. Must we use it?
In fact, for large and dense $X$, our best option is probably to use primal proximal gradient descent. The gradient

$$\nabla f(\beta) = X^T \nabla h(X \beta)$$

is easily computed via the chain rule, and the prox operator

$$\text{prox}_t(\beta) = \arg\min_{\beta} \frac{1}{2t} \| \beta - z \|_2^2 + \lambda \| D z \|_1$$

is not evaluable in closed-form, but it is precisely the same problem we considered solving before: graph fused lasso with Gaussian loss, and without regressors.

Hence to (approximately) evaluate the prox, we run a dual interior point method until convergence. We have freed ourselves entirely from solving linear systems in $X$.
Figure 4: Consistency of selected voxels in different trials of cross-validations. The results of 5 different folds of cross-validations are shown in (a)-(e) and the overlapping voxels in all 10 folds are shown in (f). The top row shows the results for GFL and the bottom row shows the results for \( L_1 \). The percentages of the overlapping voxels were: GFL(66%) vs. \( L_1 \)(22%).

Table 1: Comparison of the accuracy of AD classification.

<table>
<thead>
<tr>
<th>Task</th>
<th>LR SVM</th>
<th>LR+GFL</th>
<th>LR+ ( L_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD/NC</td>
<td>80.45%</td>
<td>82.71%</td>
<td>81.20%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>84.21%</td>
</tr>
<tr>
<td>MCI</td>
<td>63.83%</td>
<td>67.38%</td>
<td>68.79%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>70.92%</td>
</tr>
</tbody>
</table>

We compared GFL to logistic regression (LR), support vector machine (SVM), and logistic regression with an \( L_1 \) regularizer. The classification accuracies obtained based on a 10-fold cross validation (CV) are shown in Table 1, which shows that GFL yields the highest accuracy in both tasks. Furthermore, compared with other reported results, our performance are comparable with the state-of-the-art. In (Cheng, Zhang, and Shen 2012), the best performance with MCI tasks is 69.4% but our method reached 70.92%. In (Chu et al. 2012), a similar sample size is used as in our experiments, the performance of our method with AD/NC tasks is comparable to or better than their reported results (84.21% vs. 81-84%) whereas our performance with MCI tasks is much better (70.92% vs. 65%).

We applied GFL to all the samples where the optimal parameter settings were determined by cross-validation. Figure 5 compares the selected voxels with non-structured sparsity (i.e. \( L_1 \)), which shows that the voxels selected by GFL clustered into several spatially connected regions, whereas the voxels selected by \( L_1 \) were more scattered. We considered the voxels that corresponded to the top 50 negative \( i \)'s as the most atrophied voxels and projected them onto a slice. The results show that the voxels selected by GFL were concentrated in hippocampus, parahippocampal gyrus, which are believed to be the regions with early damage that are associated with AD. By contrast, \( L_1 \) selected either less critical voxels or noisy voxels, which were not in the regions with early damage (see Figure 5(b) and 5(c) for details). The voxels selected by GFL were also much more consistent than those selected by \( L_1 \), where the percentages of overlapping voxels according to the 10-fold cross-validation were: GFL=66% vs. \( L_1 \)=22%, as shown in Figure 4.

Conclusions

In this study, we proposed an efficient and scalable algorithm for GFL. We demonstrated that the proposed algorithm performs significantly better than existing algorithms. By exploiting the efficiency and scalability of the proposed algorithm, we formulated the diagnosis of AD as GFL. Our evaluations showed that GFL delivered state-of-the-art classification accuracy and the selected critical voxels were well structured.

(From Xin et al. (2014), “Efficient generalized fused lasso and its application to the diagnosis of Alzheimer’s disease”)

(From Xin et al. (2014), “Efficient generalized fused lasso and its application to the diagnosis of Alzheimer’s disease”)
Relative importance: dementia vs normal

Relative importance: death vs normal

(From Adhikari et al. (2015), “High-dimensional longitudinal classification with the multinomial fused lasso”)
Let’s turn to the special case of the fused lasso in 1d, recall

\[
D = \begin{bmatrix}
-1 & 1 & 0 & \ldots & 0 & 0 \\
0 & -1 & 1 & \ldots & 0 & 0 \\
\vdots & & & & & \\
0 & 0 & 0 & \ldots & -1 & 1
\end{bmatrix}
\]

The prox function in the primal is

\[
\text{prox}_t(\beta) = \arg\min_z \frac{1}{2t} \|\beta - z\|_2^2 + \lambda \sum_{i=1}^n |z_i - z_{i+1}|
\]

This can be directly computed using specialized approaches such as dynamic programming\(^2\) or taut-string methods\(^3\) in \(O(n)\) operations.

---

\(^2\)Johnson (2013), “A dynamic programming algorithm for the fused lasso and \(L_0\)-segmentation”

\(^3\)Davies and Kovac (2001), “Local extremes, runs, strings, multiresolution”
How fast is this prox operation, say with dynamic programming?

In short, really fast! Hence, primal proximal gradient descent looks very appealing, because the primal prox is so efficient. Note this is true for any loss function $f$. 

Dynamic programming versus Banded matrix solve
When $f$ is the Gaussian or logistic losses, without predictors, both primal proximal gradient and dual interior point method are strong choices. How do they compare? Logistic loss example, $n = 2000$:

![Graph showing comparison between proximal gradient and interior point method](image)

Primal prox gradient is better for large $\lambda$, dual interior point better for small $\lambda$. Why this direction?
Primal : \( \min_{\beta} f(\beta) + \lambda \|D\beta\|_1 \)

Dual : \( \min_u f^*(-D^T u) \) subject to \( \|u\|_\infty \leq \lambda \)

Observe:

- Large \( \lambda \): many components \((D\hat{\beta})_i = 0\) in primal, and many components \(\hat{u}_i \in (-\lambda, \lambda)\) in dual
- Small \( \lambda \): many components \((D\hat{\beta})_i \neq 0\) in primal, and many components \(|\hat{u}_i| = \lambda\) in dual

When many \((D\hat{\beta})_i = 0\), there are fewer “effective parameters” in the primal optimization; when many \(|\hat{u}_i| = \lambda\), the same is true in the dual

Hence, generally:

- Large \( \lambda \): easier for primal algorithms
- Small \( \lambda \): easier for dual algorithms
In the previous example, dual interior point method starts winning at about $\lambda = 0.1$, but statistically speaking, this is already crazily under-regularized.

Red: $\lambda = 0.01$, blue: $\lambda = 15$, dotted black: true underlying mean
Consider a problem with a dense, generic $D$, e.g., as an observed analyzing operator, or stemming from an equality-constrained lasso problem.

Primal prox is intractable, and a dual interior point method has extremely costly Newton steps.

But, provided that we can form $f^*$ (and relate the primal and dual solutions), dual proximal gradient still features efficient iterations: the gradient computation $D \nabla f^*(-D^T u)$ is more expensive than it would be if $D$ were sparse and structured, but still not anywhere as expensive as solving a linear system in $D$.

Its iterations simply repeat projecting $u + t D \nabla f^*(-D^T u)$ onto the box $[-\lambda, \lambda]^m$, hence, especially if we do not need a highly accurate solution, dual proximal gradient is the best method.
Finally, consider a twist on this problem in which $D$ is dense and so massive that even fitting it in memory is a burden.

Depending on $f$ and its gradient, **primal subgradient method** might be the only feasible algorithm; recall the subgradient calculation

$$g = \nabla f(\beta) + \lambda \sum_{i \in S} \text{sign}((D\beta)_i) \cdot D_i$$

where $S$ is the set of all $i$ such that $(D\beta)_i \neq 0$.

If $\lambda$ is large enough so that many $(D\beta)_i = 0$, then we only need to fit a **small part** of $D$ in memory (or, read a small part of $D$ from a file) to perform subgradient updates.

Combined with perhaps a stochastic trick in evaluating either part of $g$ above, this could be effective at large scale.
What did we learn from this?

From generalized lasso study (really, these are general principles):

- **There is no single best method:** performance depends greatly on the structure of penalty, conjugate of loss, desired accuracy level, and sought regularization level.
- **Duality is your friend:** dual approaches offer complementary strengths, move linear transformation from nonsmooth penalty into smooth loss, and strive in different regularization regime.
- **Regressors complicate duality:** presence of predictor variables in the loss complicate dual relationship, but proximal gradient will reduce this to a problem without predictors.
- **Recognizing easy subproblems:** if there is a subproblem that is specialized and efficiently solvable, then work around it.
- **Limited memory at scale:** for large problems, active set and/or stochastic methods may be only option.
Your toolbox will only get bigger

There are still many algorithms to be learned. E.g., for generalized lasso problems, depending on the setting, we may instead use:

- Alternating direction method of multipliers
- Proximal Newton’s method
- Projected Newton’s method
- Exact path-following methods

Remember, you don’t have to find/design the perfect optimization algorithm, just one that will work well for your problem!

For completeness, recall tools like cvx\(^4\) and tfocs\(^5\), if performance is not a concern, or you don’t want to expend programming effort.


**Implementation tips**

Implementation details are not typically the focus of optimization courses, because in a sense, implementation skills are under-valued.

Still an extremely important part of optimization. Considerations:

- Speed
- Robustness
- Simplicity
- Portability

First point doesn’t need to be explained. **Robustness** refers to the stability of implementation across various use cases. E.g., suppose our graph fused lasso solver supported edge weights. It performs well when weights are all close to uniform, but what happens under highly nonuniform weights? Huge and small weights, mixed?
Simplicity and portability are often ignored. An implementation with 20K lines of code may run fast, but what happens when a bug pops up? What happens when you pass it on to a friend? Tips:

- A constant-factor speedup is probably not worth a much more complicated implementation, especially if the latter is hard to maintain, hard to extend
- Speed of convergence to higher accuracy may be worth a loss of simplicity
- Write the code bulk in a low-level language (like C or C++), so that it can port to R, Matlab, Python, Julia, etc.
- Don’t re-implement standard routines, this is often not worth your time, and prone to bugs. Especially true for numerical linear algebra routines!
References

Some algorithms for generalized lasso problems:

- S.-J. Kim, K. Koh, S. Boyd, and D. Gorinevsky (2009), “$\ell_1$ trend filtering”
Some implementations of generalized lasso algorithms:

- T. Arnold and Ryan Tibshirani, genlasso, http://cran.r-project.org/package=genlasso