Lecture 5: Sparse GLMs

STATS305B: Applied Statistics II

Scott Linderman

January 22, 2025

Recap

Last time...

- Generalized Linear Models (GLMs)
- Canonical form
- Demo: Poisson GLM
- Non-canonical forms
- Model checking and comparison

Outline

Today...

- (Weighted) LASSO Regression
- Synthetic Demo
- The GLMNet Algorithm
- Why it Works: Proximal Methods

Motivation

We like linear and generalized linear models because the parameters are readily interpretable.

β_j relates changes in covariate x_j to changes in the natural parameter of the response distribution.

One common application of such models is for *variable selection*, finding a subset of covariates that are most predictive of the response.

For variable selection, we would like our estimates, $\hat{\boldsymbol{\beta}}$, to be *sparse*.

When we have a vast number covariates — as in genome-wide association studies (GWAS) where we aim to predict a trait given thousands of single nucleotide polymorphisms (SNPs) in the genome — sparse solutions help focus our attention on the most relevant covariates.

Lasso Regression

Consider a linear Gaussian model,

$$y_i \overset{\text{ind}}{\sim} \mathrm{N}(\beta_0 + \boldsymbol{x}_i^\top \boldsymbol{\beta}, 1) \qquad \text{for } i = 1, \ldots, n,$$

where β_0 is the intercept parameter, $\boldsymbol{x}_i \in \mathbb{R}^p$ are the covariates, and $\boldsymbol{\beta} \in \mathbb{R}^p$ are the weights. We factored out the intercept because we typically don't regularize that parameter.

Lasso Regression

The Lasso yields sparse solutions for linear models like this by minimizing the (average) negative log likelihood subject to ℓ_1 regularization.

$$\begin{aligned}\mathscr{L}(\beta_0, \boldsymbol{\beta}) &= -\frac{1}{n}\sum_{n=1}^{n} \log \mathrm{N}(y_i \mid \beta_0 + \boldsymbol{x}_i^\top \boldsymbol{\beta}, 1) + \lambda \|\boldsymbol{\beta}\|_1 \\ &= \frac{1}{2n}\sum_{i=1}^{n} (y_i - \beta_0 - \boldsymbol{x}_i^\top \boldsymbol{\beta})^2 + \lambda \|\boldsymbol{\beta}\|_1 \\ &= \frac{1}{2n}\sum_{i=1}^{n} (y_i - \beta_0 - \boldsymbol{x}_i^\top \boldsymbol{\beta})^2 + \lambda \sum_{j=1}^{p} |\beta_j|\end{aligned}$$

This is still a convex objective function!

It's tempting to just use vanilla gradient ascent to find the optimum,

$$\boldsymbol{\beta}^{(t+1)} \leftarrow \boldsymbol{\beta}^{(t)} - \alpha_t \nabla \mathscr{L}(\boldsymbol{\beta}^{(t)}),$$

Lasso Regression

Unfortunately, the Lasso objective it is not continuously differentiable: the **gradient at $\beta_j = 0$ is discontinuous** due to the absolute value in the ℓ_1 norm. What can we do instead?

- You only live once… run gradient descent and hope for the best!
- Use *subgradient* descent, taking a step in the direction of any subgradient of $\mathscr{L}$, but that approach can be much slower, with convergence rates of only $\mathcal{O}(1/\epsilon^2)$.
- Use *proximal* gradient descent, which amounts to *iterative soft thresholding* for the Lasso problem. This has much better convergence rates, as we'll discuss below.
- Use *coordinate descent*, which works very well in practice, even if it's harder to prove convergence rates.

Coordinate Descent

Fix all parameters except for β_j for some $j \in \{1, \ldots, p\}$. As a function of β_j, the average negative log likelihood is,

$$\mathscr{L}(\beta_j; \boldsymbol{\beta}_{\neg j}) = \frac{1}{2n} \sum_{i=1}^{n} (\widetilde{y}_{ij} - x_{ij}\beta_j)^2 + \lambda|\beta_j| + c,$$

where

$$\widetilde{y}_{ij} = y_i - \beta_0 - \sum_{k \neq j} x_{ik}\beta_k,$$

and where c is constant with respect to β_j.

Coordinate Descent

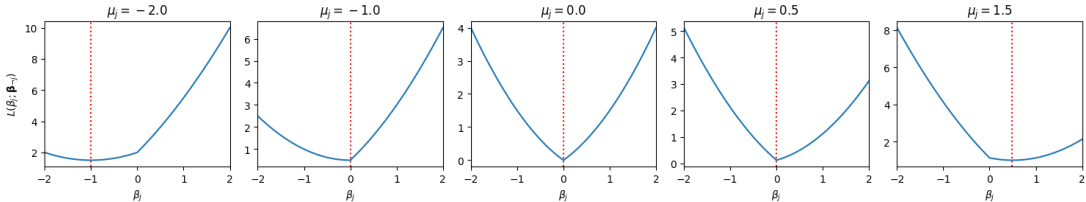
This is just a scalar minimization problem. Completing the square, we can rewrite the objective as,

$$\mathscr{L}(\beta_j; \boldsymbol{\beta}_{\neg j}) = \frac{1}{2}\frac{(\beta_j - \mu_j)^2}{\sigma_j^2} + \lambda|\beta_j| + c'$$

Exercise: Solve for μ_j and σ_j^2.

Coordinate Descent

Let's plot the objective for a few values of μ_j, fixing $\sigma_j^2 \lambda = 1$.



Note that many of the minimizers (denoted by the red lines) are obtained at $\beta_j = 0$!

The Soft-Thresholding Operator

With a bit of calculus, we can show that the minimizer is given by the **soft-thresholding operator**,

$$\begin{aligned}\beta_j^\star &= \begin{cases} \mu_j - \sigma_j^2\lambda & \text{if } \mu_j > \sigma_j^2\lambda \\ 0 & \text{if } |\mu_j| < \sigma_j^2\lambda \\ \mu_j + \sigma_j^2\lambda & \text{if } u_j < -\sigma_j^2\lambda \end{cases} \\ &\triangleq S_{\sigma_j^2\lambda}(\mu_j)\end{aligned}$$

We can write the soft-thresholding operator more compactly as,

$$S_\alpha(\mu) = \text{sign}(\mu)\max\{|\mu| - \alpha, 0\}.$$

Coordinate ascent step for the intercept

Exercise: Show that the coordinate update for the intercept is,

$$\beta_0^\star = \frac{1}{n}\sum_{i=1}^{n}\widetilde{y}_{i0},$$

where $\widetilde{y}_{i0} = y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta}$.

Weighted Lasso Regression

Finally, suppose we have heteroskedastic noise,

$$y_i \sim \mathrm{N}(\beta_0 + \boldsymbol{x}_i^\top \boldsymbol{\beta}, w_i^{-1})$$

where w_i is the inverse variance (precision) of the i-th observation.

Then the objective would become,

$$\begin{aligned}\mathscr{L}(\beta_0, \boldsymbol{\beta}) &= -\frac{1}{n}\sum_{n=1}^{n} \log \mathrm{N}(y_i \mid \beta_0 + \boldsymbol{x}_i^\top \boldsymbol{\beta}, w_i^{-1}) + \lambda \|\boldsymbol{\beta}\|_1 \\ &= \frac{1}{2n}\sum_{i=1}^{n} w_i (y_i - \beta_0 - \boldsymbol{x}_i^\top \boldsymbol{\beta})^2 + \lambda \|\boldsymbol{\beta}\|_1\end{aligned}$$

Note that the w_i's become the **weights** in the objective.

Weighted Lasso Regression

Following the same steps as above, we can write the objective as a function of β_j,

$$\mathscr{L}(\beta_j; \boldsymbol{\beta}_{\neg j}) = \frac{1}{2}\frac{(\beta_j - \mu_j)^2}{\sigma_j^2} + \lambda|\beta_j| + c'$$

where

$$\mu_j = \frac{h_j}{J_j}$$

$$\sigma_j^2 = \frac{1}{J_j}$$

$$J_j = \frac{1}{n}\sum_{i=1}^{n} w_i x_{ij}^2$$

$$h_j = \frac{1}{n}\sum_{i=1}^{n} w_i \tilde{y}_{ij} x_{ij}.$$

Weighted Lasso Regression

Again, the coordinate-wise minimum is obtained at $\beta_j^\star = S_{\sigma_j^2 \lambda}(\mu_j)$.

For the intercept, $\beta_0^\star = \frac{\sum_{i=1}^n w_i \widetilde{y}_{i0}}{\sum_{i=1}^n w_i}$.

Synthetic Demo

See Colab

Questions

1. Can you identify two was in which the Lasso estimate differs from the OLS estimate?
2. What estimate do you get if you set $\lambda = 0$?
3. What if you take $\lambda \to \infty$?

Fitting Sparse GLMs

Now let's generalize this approach to fit ℓ_1-regularized GLMs! This is exactly what the `glmnet` package (Friedmann et al., 2010) solves.

Suppose we have a GLM with the canonical mean function,

$$\mathbb{E}[Y_i] = f(\beta_0 + \boldsymbol{x}_i^\top \boldsymbol{\beta})$$

where we have again factored out the intercept.

The regularized objective is,

$$\mathscr{L}(\beta_0, \boldsymbol{\beta}) = -\frac{1}{n} \sum_{i=1}^{n} \log p(y_i \mid f(\beta_0 + \boldsymbol{x}_i^\top \boldsymbol{\beta})) + \lambda \|\boldsymbol{\beta}\|_1.$$

Review: Iteratively Reweighted Least Squares

Recall that Newton's method for canonical GLMs (without regularization) is equivalent to **iteratively reweighted least squares**. The $(t+1)$-th step of Newton's method is equivalent to solving a weighted least squares problem to find the minimum of an objective,

$$\widetilde{\mathscr{L}}(\beta_0, \boldsymbol{\beta}; \beta_0^{(t)}, \boldsymbol{\beta}^{(t)}) = \frac{1}{n}\sum_{i=1}^{n} w_i^{(t)} (z_i^{(t)} - \beta_0 - \boldsymbol{x}_i^\top \boldsymbol{\beta})^2$$

where the working responses are

$$z_i^{(t)} = \beta_0^{(t)} + \boldsymbol{x}_i^\top \boldsymbol{\beta}^{(t)} + \frac{y_i - \hat{y}_i^{(t)}}{w_i^{(t)}},$$

the predictions are,

$$\hat{y}_i^{(t)} = f(\beta_0^{(t)} + \boldsymbol{x}_i^\top \boldsymbol{\beta}^{(t)}),$$

Review: Iteratively Reweighted Least Squares

and the weights are equal to the conditional variances,

$$w_i^{(t)} = \text{Var}[Y_i \mid \boldsymbol{x}_i, \beta_0^{(t)}, \boldsymbol{\beta}^{(t)}].$$

For example, in a logistic regression,

$$\hat{y}_i^{(t)} = \sigma(\beta_0^{(t)} + \boldsymbol{x}_i^\top \boldsymbol{\beta}^{(t)})$$
$$w_i = \hat{y}_i^{(t)}(1 - \hat{y}_i^{(t)}).$$

The Algorithm

From here, we can sketch out a pretty straightforward algorithm for fitting sparse GLMs.

Within each Newton iteration, solve a weighted least squares problem, *subject to the ℓ_1-regularization penalty*, using coordinate descent.

Once the coordinate descent procedure converges, update the working responses and weights, then repeat.

This is essentially the algorithm in `glmnet` (Friedmann et al., 2010)!

Synthetic Demo

See Colab

Computational Tricks

There are several simple tricks to speed it up.

- Rather than recomputing the residual for each coordinate, we can update and downdate the residual after each coordinate update.
- You can show that the coordinate updates only depend on sufficient statistics $\sum_i w_i x_{ij} x_{ik}$ and $\sum_i w_i z_i x_{ij}$, and these statistics don't change within the each outer loop. We can save some time by precomputing these at the start of the `_glmnet_step` function. This trick is referred to as using *covariance updates*.
- When X is sparse, we can implement the sufficient statistics calculations even more efficiently.

Friedmann et al (2010) describe several other implementation-level details for making the code as fast as possible. You should also check out James Yang's thesis and his amazing `adelie` package for LASSO problems.

Proximal Methods

The `glmnet` algorithm is intuitive, but **why the heck does it work?!** To gain a deeper theoretical understanding, let's take a step back and talk about proximal methods.

Proximal Gradient Descent

Proximal gradient descent is an optimization algorithm for convex objectives that decompose into a differentiable part and a non-differentiable part,

$$\mathscr{L}(\boldsymbol{\beta}) = \mathscr{L}_{\text{d}}(\boldsymbol{\beta}) + \mathscr{L}_{\text{nd}}(\boldsymbol{\beta})$$

where $\mathscr{L}_{\text{d}}$ is convex and *differentiable*, whereas $\mathscr{L}_{\text{nd}}$ is convex but *not differentiable*. The idea is to stick as close to vanilla gradient descent as possible, while correcting for the non-differentiable part of the objective.

If we just had the differentiable part, $\mathscr{L}_{\text{d}}$, we could perform gradient descent. One way to think about the gradient descent update is as the solution to a quadratic minimization problem,

$$\boldsymbol{\beta}^{(t+1)} \leftarrow \arg\min_{\boldsymbol{z}} \mathscr{L}_{\text{d}}(\boldsymbol{\beta}^{(t)}) + \nabla \mathscr{L}_{\text{d}}(\boldsymbol{\beta}^{(t)})^{\top}(\boldsymbol{z} - \boldsymbol{\beta}^{(t)}) + \frac{1}{2\alpha_t}\|\boldsymbol{z} - \boldsymbol{\beta}^{(t)}\|_2^2$$

We can think of the surrogate problem as a second order approximation of the objective in which the Hessian is replaced with $\frac{1}{\alpha_t}\boldsymbol{I}$.

Proximal Gradient Descent

Proximal gradient descent follows the same logic, but it keeps the non-differentiable part,

$$\begin{aligned}\boldsymbol{\beta}^{(t+1)} &\leftarrow \arg\min_{\boldsymbol{z}} \mathscr{L}_{\text{d}}(\boldsymbol{\beta}^{(t)}) + \nabla \mathscr{L}_{\text{d}}(\boldsymbol{\beta}^{(t)})^{\top}(\boldsymbol{z} - \boldsymbol{\beta}^{(t)}) + \frac{1}{2\alpha_t}\|\boldsymbol{z} - \boldsymbol{\beta}^{(t)}\|_2^2 + \mathscr{L}_{\text{nd}}(\boldsymbol{\beta}^{(t)}) \\ &= \arg\min_{\boldsymbol{z}} \frac{1}{2\alpha_t}\|\boldsymbol{z} - (\boldsymbol{\beta}^{(t)} - \alpha_t \nabla \mathscr{L}_{\text{d}}(\boldsymbol{\beta}^{(t)}))\|_2^2 + \mathscr{L}_{\text{nd}}(\boldsymbol{z})\end{aligned}$$

The resulting update balances two parts:

1. Stay close to the vanilla gradient descent update, $\boldsymbol{\beta}^{(t)} - \alpha_t \nabla \mathscr{L}_{\text{d}}(\boldsymbol{\beta}^{(t)})$.
2. Also minimize the non-differentiable part of the objective, $\mathscr{L}_{\text{nd}}(\boldsymbol{\beta}^{(t)})$.

As a sanity check, note that we recover vanilla gradient descent with $\mathscr{L}_{\text{nd}}(\boldsymbol{\beta}^{(t)}) = 0$.

Proximal Mapping

We call the function,

$$\text{prox}(\boldsymbol{u}; \alpha_t) = \arg\min_{\boldsymbol{z}} \frac{1}{2\alpha_t} \|\boldsymbol{z} - \boldsymbol{u}\|_2^2 + \mathscr{L}_{\text{nd}}(\boldsymbol{z}) \tag{1}$$

the **proximal mapping**.

Notes

- The proximal mapping depends on the form of the non-differentiable part of the objective, even though we have suppressed that in the notation.
- However, it does *not* depend on the form of the continuous part of the objective.

Algorithm

With this definition, the proximal gradient descent algorithm is,

Proximal Gradient Descent

Input: Initial parameters $\boldsymbol{\beta}^{(0)}$, proximal mapping $\text{prox}(\cdot;\cdot)$.

- **For** $t = 1, \ldots, T$
  - Set $\boldsymbol{\beta}^{(t)} \leftarrow \text{prox}(\boldsymbol{\beta}^{(t-1)} - \alpha_t \nabla \mathscr{L}_{\text{d}}(\boldsymbol{\beta}^{(t-1)}); \alpha_t)$.

Return $\boldsymbol{\beta}^{(T)}$.

So far, it's not obvious that this framing is helpful. We still have a potentially challenging optimization problem to solve in computing the proximal mapping. However, for many problems of interest, the proximal mapping has simpled closed solutions.

Proximal Gradient Descent for Lasso Regression

Consider the Lasso problem. The objective decomposes into convex differentiable and non-differentiable parts,

$$\mathscr{L}_{\text{d}}(\boldsymbol{\beta}) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2$$
$$\mathscr{L}_{\text{nd}}(\boldsymbol{\beta}) = \lambda\|\boldsymbol{\beta}\|_1.$$

Proximal Mapping

The proximal mapping is,

$$\begin{aligned}\text{prox}(\boldsymbol{u}; \alpha_t) &= \arg\min_{\boldsymbol{z}} \frac{1}{2\alpha_t}\|\boldsymbol{z}-\boldsymbol{u}\|_2^2 + \lambda\|\boldsymbol{z}\|_1 \\ &= \arg\min_{\boldsymbol{z}} \sum_{j=1}^{p} \frac{1}{2\alpha_t}(z_j - u_j)^2 + \lambda|z_j|\end{aligned}$$

It separates into optimization problems for each coordinate, and each coordinate has a closed-form solution in terms of the soft-thresholding operator!

$$[\text{prox}(\boldsymbol{u}; \alpha_t)]_j = S_{\alpha_t\lambda}(u_j)$$

Iterative Soft-Thresholding Algorithm

Now let's plug in the gradient of the differentiable part,

$$\nabla \mathscr{L}_{\mathrm{d}}(\boldsymbol{\beta}) = \boldsymbol{X}^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}).$$

Substituting this into the proximal gradient descent algorithm yields what is sometimes called the **iterative soft-thresholding algorithm (ISTA)**,

Iterative Soft-Thresholding

Input: Initial parameters $\boldsymbol{\beta}^{(0)}$, covariates $\boldsymbol{X} \in \mathbb{R}^{n \times p}$, responses $\boldsymbol{y} \in \mathbb{R}^n$

- **For** $t = 1, \ldots, T$
  - Set $\boldsymbol{\beta}^{(t)} \leftarrow S_{\alpha_t \lambda}(\boldsymbol{\beta}^{(t-1)} - \alpha_t \boldsymbol{X}^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}^{(t-1)}))$.

Return $\boldsymbol{\beta}^{(T)}$.

Convergence

If $\nabla \mathscr{L}_{\text{d}}$ is L-smooth then proximal gradient descent with fixed step size $\alpha_t = 1/L$ then,

$$f(\boldsymbol{\beta}^{(t)}) - f(\boldsymbol{\beta}^\star) \leq \frac{L}{2t} \|\boldsymbol{\beta}^{(0)} - \boldsymbol{\beta}^\star\|_2^2,$$

so it matches the gradient descent convergence rate of $\mathcal{O}(1/\epsilon)$. (With Nesterov's accelerated gradient techniques, you can speed this up to $\mathcal{O}(1/\sqrt{\epsilon})$.

Proximal Newton Method

One great thing about proximal gradient descent is its generality. We could easily apply it to ℓ_1-regularized GLMs, substituting the gradient of the negative log likelihood, which also has a simple closed form expression. The proximal operator remains the same, and we obtain the same converge rates as gradient descent on standard GLMs.

However, we saw that Newton's method yielded significantly faster convergence rates of $\mathcal{O}(\log\log\frac{1}{\epsilon})$. Can we obtain similar performance for ℓ_1-regularized GLMs?

To obtain a **proximal Newton method**, we proceed in the same fashion as above, but rather than approximating the second order term with $\alpha_t^{-1}\boldsymbol{I}$, we will use the Hessian of $\mathscr{L}_{\mathrm{d}}$. That leads to a proximal mapping of the form,

$$\mathrm{prox}(\boldsymbol{u};\boldsymbol{H}_t) = \arg\min_{\boldsymbol{z}} \frac{1}{2}\|\boldsymbol{z}-\boldsymbol{u}\|_{\boldsymbol{H}_t}^2 + \mathscr{L}_{\mathrm{nd}}(\boldsymbol{z})$$

where $\|\boldsymbol{x}\|_{\boldsymbol{H}_t}^2 = \boldsymbol{x}^\top \boldsymbol{H}_t \boldsymbol{x}$ is a squared norm induced by the positive definite matrix $\boldsymbol{H}_t$.

Proximal Newton Method

Note: the proximal mapping for proximal gradient descent corresponds to the special case in which $\boldsymbol{H}_t = \frac{1}{\alpha_t}\boldsymbol{I}$.

Let $\boldsymbol{g}_t = \nabla \mathscr{L}_{\mathrm{d}}(\boldsymbol{\beta}^{(t)})$ and $\boldsymbol{H}_t = \nabla^2 \mathscr{L}_{\mathrm{d}}(\boldsymbol{\beta}^{(t)})$ denote the gradient and Hessian, respectively. The *undamped* proximal Newton update is,

$$\begin{aligned}
\hat{\boldsymbol{\beta}}^{(t+1)} &\leftarrow \arg\min_{\boldsymbol{z}} \mathscr{L}_{\mathrm{d}}(\boldsymbol{\beta}^{(t)}) + (\boldsymbol{z} - \boldsymbol{\beta}^{(t)})^\top \boldsymbol{g}_t + \frac{1}{2}(\boldsymbol{z} - \boldsymbol{\beta}^{(t)})^\top \boldsymbol{H}_t (\boldsymbol{z} - \boldsymbol{\beta}^{(t)}) + \mathscr{L}_{\mathrm{nd}}(z) \\
&= \arg\min_{\boldsymbol{z}} \frac{1}{2} \|\boldsymbol{z} - (\boldsymbol{\beta}^{(t)} - \boldsymbol{H}_t^{-1}\boldsymbol{g}_t)\|_{\boldsymbol{H}_t}^2 + \mathscr{L}_{\mathrm{nd}}(\boldsymbol{z}) \\
&= \mathrm{prox}(\boldsymbol{\beta}^{(t)} - \boldsymbol{H}_t^{-1}\boldsymbol{g}_t; \boldsymbol{H}_t)
\end{aligned}$$

As with Newton's method, however, we often need to use damped updates,

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} + \alpha_t(\hat{\boldsymbol{\beta}}^{(t+1)} - \boldsymbol{\beta}^{(t)}),$$

However, solving the proximal Newton mapping can be more challenging.

Proximal Newton for Sparse GLMs

Let's consider the proximal Newton mapping for ℓ_1-regularized GLMs, like logistic regression. Here, the non-differentiable part of the objective is $\mathscr{L}_{\text{nd}}(\boldsymbol{\beta}) = \lambda\|\boldsymbol{\beta}\|_1$. Unfortunately, the proximal Newton update no longer has a closed form solution because when we introduce the Hessian, the problem no longer separates across coordinates since the Hessian is generally not diagonal.

However, note that the proximal Newton step minimizes a second-order Taylor approximation of the log likelihood plus an ℓ_1-regularization penalty,

$$\text{prox}(\boldsymbol{u}; \boldsymbol{H}_t) = \arg\min_{\boldsymbol{z}} \mathscr{L}_{\text{d}}(\boldsymbol{\beta}^{(t)}) + (\boldsymbol{z} - \boldsymbol{\beta}^{(t)})^\top \boldsymbol{g}_t + \frac{1}{2}(\boldsymbol{z} - \boldsymbol{\beta}^{(t)})^\top \boldsymbol{H}_t (\boldsymbol{z} - \boldsymbol{\beta}^{(t)}) + \mathscr{L}_{\text{nd}}(z)$$

This is exactly the same problem in the inner loop of the `glmnet` algorithm! In particular, we can view the second-order Taylor approximation of the log likelihood as a weighted least squares objective with working responses and weights. We can solve that inner problem with coordinate ascent, just like above.

TL;DR: the intuitive algorithm we derived above is really a proximal Newton algorithm.

Caveats

As with regular Newton's method, proximal Newton exhibits local quadratic convergence to obtain error ϵ in $\mathcal{O}(\log\log 1/\epsilon)$ iterations. Though here, each iteration requires an inner coordinate descent loop to solve the proximal mapping.

Warning: In practice, you may need to also implement a backtracking line search to choose the step size α_t, since you may not start in the local quadratic regime. Logistic regression with decent initialization is reasonably well behaved, but Poisson regression with log link functions can be sensitive.

Conclusion

The proximal methods disussed today are what run behind the scenes of modern packages for sparse linear and logistic regression.

In particular, `sklearn.linear_model.Lasso` uses a fast coordinate descent algorithm like discussed above, and GLMNet (Friedmann et al., 2010) uses a proximal Newton algorithm with coordinate descent for the proximal step.