Graphical Lasso - and its application to Deep Learning

Yash Jakhmola

IISER Kolkata

What is lasso ?

Suppose we have N samples of data drawn from a linear model with p parameters:

$$y_i = \beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j$$

for $i = 1, 2, \ldots, N$. Our goal is to find the parameters. To do so, we minimize the squared error :

$$\min_{\beta_0, \beta_1, \cdots, \beta_p} \left(\frac{1}{2N} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 \right)$$

Now, we want to make this sparse. One of the reasons sparsity is good to have is the 'bet on sparsity' principle - "use a procedure that does well in sparse problems, since no procedure does well in dense problems."

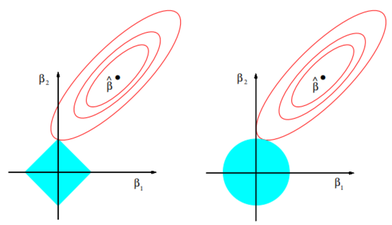
To do so, we constraint the parameters. First of all, let's normalize the data. This helps us get rid of β_0 and also make sure no value is too large or too small. Now, to constraint the parameters, we add the condition that $||\beta||_1 \leq t$, where $\beta = (\beta_1, \ldots, \beta_p)^T$, $||\cdot||_1$ is the l^1 norm and t controls how many parameters are to be set to zero. We can also write our problem as follows:

$$\min_{\beta_1, \cdots, \beta_p} \left(\frac{1}{2N} ||Y - X\beta||_2^2 + \lambda ||\beta||_1 \right)$$

where $Y = (y_1, \ldots, y_N)^T$, $X = (x_{ij})_{N \times p}$, $||\cdot||_2$ is the l^2 norm and $\lambda > 0$ is the penalty for not keeping $||\beta||_1$ 'small'.

But, if we have to just constraint the vector β, why not just put a constraint on the maximum value (l_∞ norm)? Or why not just put a constraint on any l^p norm?

Firstly, since $||x||_q \leq ||x||_p$ for all $q \geq p$, we want a norm l_p with p as small as possible, since restricting the l_p norm would mean restricting the l_q norm for all $q \geq p$. However, if $p < 1$, then our optimization problem is not convex anymore. In particular, the set $\{x \in \mathbb{R}^p : ||x||_p \leq t\}$ is not convex. It is very desirable for the optimization problem to be convex, as we shall see later.



Subdifferential of a function

Let $a, b \in \mathbb{R}$ and let $f : (a, b) \to \mathbb{R}$ be a convex function. Subderivative of f at some $x_0 \in (a, b)$ is defined to be a real number c such that

$$f(x) - f(x_0) \geq c(x - x_0)$$

for all $x \in (a, b)$.

The set of all subderivatives of f at x_0 is called its subdifferential, denoted by $\partial(f(x_0))$.

A convex function is differentiable at x_0 iff its subderivative at x_0 is the singleton set $\{f'(x_0)\}$.

A point is the global minimum of a convex function iff the subdifferential of the function at that point contains 0.

Solving the lasso

Consider there is just one parameter. We want to minimize

$$f(\beta) = \frac{1}{2N} \sum_{i=1}^{N} (y_i - z_i\beta)^2 + \lambda|\beta|$$

But, this function is not differentiable. Thus, we instead take the subdifferential and set it to zero to get the global minima. We can do this because the problem is convex.

$$\partial_\beta(f(\beta)) = \frac{1}{N}\langle z, y\rangle + \lambda \text{sgn}(\beta) = 0$$

$$\hat{\beta} = S_\lambda \left(\frac{1}{N}\langle z, y\rangle \right)$$

where $S_\lambda(x) = \text{sgn}(x)(|x| - \lambda)_+$. However, the solution is not as straightforward for multiple parameters. Let's minimize

$$f(\beta) = \frac{1}{2N} \sum_{i=1}^{N} (y_i - \sum_{k \neq j} x_{ik}\beta_k - x_{ij}\beta_j)^2 + \lambda \sum_{k \neq j} |\beta_k| + \lambda|\beta_j|$$

Then,

$$\hat{\beta}_j = S_\lambda \left(\frac{1}{N}\langle x, r^j\rangle \right)$$

where $r^j = (y_i - \sum_{k \neq j} x_{ik}\hat{\beta}_k)_{j=1}^n$. But, this solution clearly depends on other values of β_j. To reach the solution, we update each β_j in a cyclic fashion until we reach convergence. This is called cyclic coordinate descent. We start with an initial guess $\beta^{(0)}$. Then, for $k = 1, 2, \ldots$, we repeat the following:

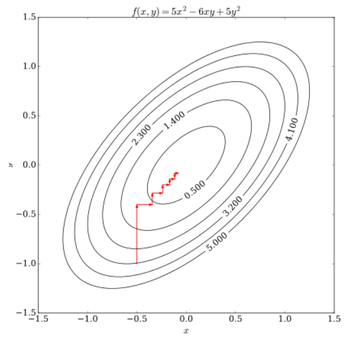
$$\beta_1^{(k)} \in \underset{\beta_1}{\text{argmin}}\, f(\beta_1, \beta_2^{(k-1)}, \ldots, \beta_n^{(k-1)})$$

$$\beta_2^{(k)} \in \underset{\beta_2}{\text{argmin}}\, f(\beta_1^{(k)}, \beta_2, \ldots, \beta_n^{(k-1)})$$

$$\cdots$$

$$\beta_n^{(k)} \in \underset{\beta_n}{\text{argmin}}\, f(\beta_1^{(k)}, \beta_2^{(k)}, \ldots, \beta_n)$$

The above process converges. This is because $\{f(\beta^{(k)})\}$ is a monotone decreasing bounded sequence, so it converges to its infimum. And, its infimum is precisely the solution of the lasso!



Graphical lasso

Suppose there are n iid random variables $X_1, \ldots, X_n$ drawn from $N(0, \Sigma)$. Then, the scaled log-likelihood of $\Theta = \Sigma^{-1}$ is

$$\mathcal{L}(\Theta) = \sum_{i=1}^{n} \log(f_\Theta(x_i)) = \log|\Theta| - \text{trace}(S\Theta)$$

where $S = \frac{1}{N}\sum_{i=1}^{n} x_i x_i^T$.

We would like to treat Θ as the adjacency matrix of a graph depicting the relationship between the random variables. In practical cases, usually this graph is sparse (each vertex is connected to only a few other vertices). Thus, we want to constraint the number of nonzero values of Θ. We can do so by a lasso constraint:

$$\hat{\Theta} = \text{argmax} \left(\log|\Theta| - \text{trace}(S\Theta) + \lambda \sum_{s \neq t} |\theta_{st}| \right)$$

This is called graphical lasso. To solve it, we need to differentiate $\mathcal{L}$ with respect to Θ and set it to zero: $\Theta^{-1} - S + \lambda\Phi = 0$, where $\Phi = (\text{sgn}(\theta_{st}))_{s,t}$. Now, we can solve this using block coordinate descent, which is nothing but the cyclic coordinate descent method but instead of numbers we use vectors.

What are (spatial) Graph Neural Networks ?

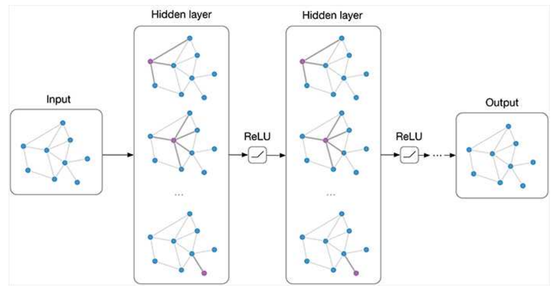
Consider the following problem: there is a time series of a vector of variables. The variables affect each others values. Given N samples of such data vectors, we want to predict the $N + 1^{\text{th}}$ vector.

An easy (and very effective) way to do so is by using deep learning. Let the input vector be X. Let the relationships between the dimensions be depicted by a graph with adjacency matrix A.

Let's multiply the input vector by an adjacency matrix A. This replaces each dimension by the sum of the values of each of its graph neighbours. Let's normalise this product just to ensure it does not blow up when the number of vertices is large : $D^{-1/2}AD^{-1/2}X$. Now, let's multiply it by a layer-specific learnable weights matrix W and pass through a nonlinear activation function (say sigmoid). Then, our output is

$$Y = \sigma(D^{-1/2}AD^{-1/2}XW)$$

We change the weights of the matrix by backpropagation, and pass the output of this layer to multiple such layers to get our final answer. But, what will the adjacency matrix be? There can be multiple answers to this question. One of the answers is to use graphical lasso.



References

[1] Robert Tibshirani Trevor Hastie and Martin Wainwright.
Statistical Learning with Sparsity : The Lasso and Generalizations.