l_1 regularization algorithms

Machine Learning: a probabilistic perspective: Kevin P Murphy (Chapter 13.3.2, 13.4)

In the case of linear regression, the ℓ_1 objective becomes

$$f(\mathbf{w}) = \sum_{i=1}^{N} -\frac{1}{2\sigma^2}(y_i - (w_0 + \mathbf{w}^T \mathbf{x}_i))^2 + \lambda ||\mathbf{w}||_1 \quad (13.35)$$

$$= \mathrm{RSS}(\mathbf{w}) + \lambda' ||\mathbf{w}||_1 \quad (13.36)$$

where $\lambda' = 2\lambda\sigma^2$. This method is known as **basis pursuit denoising** or **BPDN** (Chen et al. 1998).

$$\min_{\mathbf{w}} \mathrm{RSS}(\mathbf{w}) + \lambda ||\mathbf{w}||_1 \quad (13.37)$$

We can rewrite this as a constrained but smooth objective (a quadratic function with linear constraints):

$$\min_{\mathbf{w}} \mathrm{RSS}(\mathbf{w}) \quad \text{s.t.} \quad ||\mathbf{w}||_1 \leq B \quad (13.38)$$

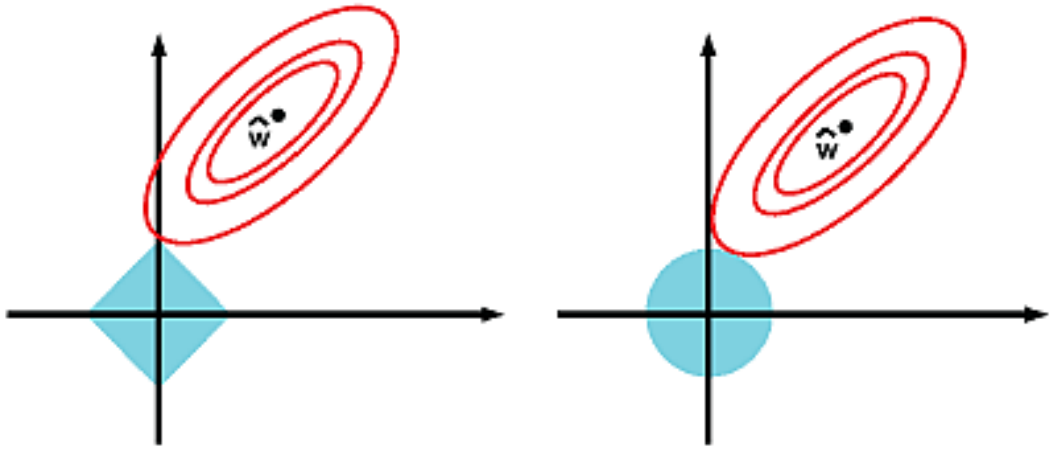
where B is an upper bound on the ℓ_1-norm of the weights: a small (tight) bound B corresponds to a large penalty λ, and vice versa.[2] Equation 13.38 is known as **lasso**, which stands for "least absolute shrinkage and selection operator" (Tibshirani 1996).

Similarly, we can write ridge regression

$$\min_{\mathbf{w}} \mathrm{RSS}(\mathbf{w}) + \lambda ||\mathbf{w}||_2^2$$

or as a bound constrained form:

$$\min_{\mathbf{w}} \mathrm{RSS}(\mathbf{w}) \quad \text{s.t.} \quad ||\mathbf{w}||_2^2 \leq B$$



In Figure 13.3, we plot the contours of the RSS objective function, as well as the contours of the ℓ_2 and ℓ_1 constraint surfaces. From the theory of constrained optimization, we know that the optimal solution occurs at the point where the lowest level set of the objective function intersects the constraint surface (assuming the constraint is active). It should be geometrically clear that as we relax the constraint B, we "grow" the ℓ_1 "ball" until it meets the objective; the corners of the ball are more likely to intersect the ellipse than one of the sides, especially in high dimensions, because the corners "stick out" more. The corners correspond to sparse solutions, which lie on the coordinate axes. By contrast, when we grow the ℓ_2 ball, it can intersect the objective at any point; there are no "corners", so there is no preference for sparsity.

$\mathbf{X}^T\mathbf{X} = \mathbf{I}$. In this case, the RSS is given by

$$\begin{aligned} \text{RSS}(\mathbf{w}) &= ||\mathbf{y} - \mathbf{X}\mathbf{w}||^2 = \mathbf{y}^T\mathbf{y} \div \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{y} && (13.59) \\ &= \text{const} + \sum_k w_k^2 - 2\sum_k \sum_i w_k x_{ik} y_i && (13.60) \end{aligned}$$

so we see this factorizes into a sum of terms, one per dimension. Hence we can write down the MAP and ML estimates analytically, as follows:

- **MLE** The OLS solution is given by

  $$\hat{w}_k^{OLS} = \mathbf{x}_{:k}^T\mathbf{y} \tag{13.61}$$

  where $\mathbf{x}_{:k}$ is the k'th column of $\mathbf{X}$. This follows trivially from Equation 13.60. We see that $\hat{w}_k^{OLS}$ is just the orthogonal projection of feature k onto the response vector (see Section 7.3.2).
- **Ridge** One can show that the ridge estimate is given by

  $$\hat{w}_k^{ridge} = \frac{\hat{w}_k^{OLS}}{1+\lambda} \tag{13.62}$$

- **Lasso** From Equation 13.55, and using the fact that $a_k = 2$ and $\hat{w}_k^{OLS} = c_k/2$, we have

  $$\hat{w}_k^{lasso} = \text{sign}(\hat{w}_k^{OLS})\left(|\hat{w}_k^{OLS}| - \frac{\lambda}{2}\right)_+ \tag{13.63}$$

  This corresponds to soft thresholding, shown in Figure 13.5(a).
- **Subset selection** If we pick the best K features using subset selection, the parameter estimate is as follows

  $$\hat{w}_k^{SS} = \begin{cases} \hat{w}_k^{OLS} & \text{if rank}(|w_k^{OLS}|) \leq K \\ 0 & \text{otherwise} \end{cases} \tag{13.64}$$

  where rank refers to the location in the sorted list of weight magnitudes. This corresponds to hard thresholding, shown in Figure 13.5(b).

Optimal solution to LASSO

- Let the lasso problem be defined as

$$f(\boldsymbol{w}) = RSS(\boldsymbol{w}) + \lambda\left|\left|\boldsymbol{w}\right|\right|_1$$

Where $RSS(\boldsymbol{w}) = \left|\left|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\right|\right|_2^2$

It can be shown that (Exercise 13.1)

$$\frac{\partial}{\partial w_j} RSS(\boldsymbol{w}) = a_j w_j - c_j, \textit{ where}$$

$$a_j = 2\sum_{i=1}^{N} x_{ij}^2 \,, = 2\left|\left|x_{:,j}\right|\right|^2$$

$$c_j = 2\sum_{i=1}^{N} x_{ij}\left(y_i - \boldsymbol{w}_{-j}^T x_{i,-j}\right) = 2\mathbf{x}_{:,j}^T \boldsymbol{r}_j$$

where $\boldsymbol{w}_{-j}$ is $\boldsymbol{w}$ without the component j and similarly for $x_{i,-j}$,

$$\boldsymbol{r}_j = y_i - \boldsymbol{w}_{-j}^T x_{i,-j}$$

- Adding in the penalty term, we find that the subderivative[1] is given by

$$\frac{\partial}{\partial \mathrm{w_j}} \mathrm{f}(\mathbf{w}) = \left(\mathrm{a_j w_j} - \mathrm{c_j}\right) + \lambda \frac{\partial}{\partial \mathrm{w_j}} \left|\left|\boldsymbol{w}\right|\right|_1$$

$$= \begin{Bmatrix} \{a_j w_j - c_j - \lambda\} & if\ w_j < 0 \\ \left[-c_j - \lambda, -c_j + \lambda\right] & if\ w_j = 0 \\ a_j w_j - c_j + \lambda & if\ w_j > 0 \end{Bmatrix}$$

- In the second case, when $w_j = 0$, $-\lambda \leq c_j \leq \lambda$.
- Thus, depending on the value of c_j, the solution to $\frac{\partial}{\partial \mathrm{w_j}} \mathrm{f}(\mathbf{w}) = 0$ can occur at 3 different values of w_j
- Subderivatives are computed because of the discontinuity at the corners imposed by constraints causing certain w_j's going to 0.

1. *Subderivative* defined in Murphy – Sec. 13.3.2

Soft thresholding

$$\widehat{w}_j(c_j) = \left\{ \begin{array}{cc} \frac{c_j + \lambda}{a_j} & if\ c_j < -\lambda \\ 0 & if\ c_j \in [-\lambda, \lambda] \\ \frac{c_j - \lambda}{a_j} & if\ c_j > \lambda \end{array} \right\}$$

This can be written as

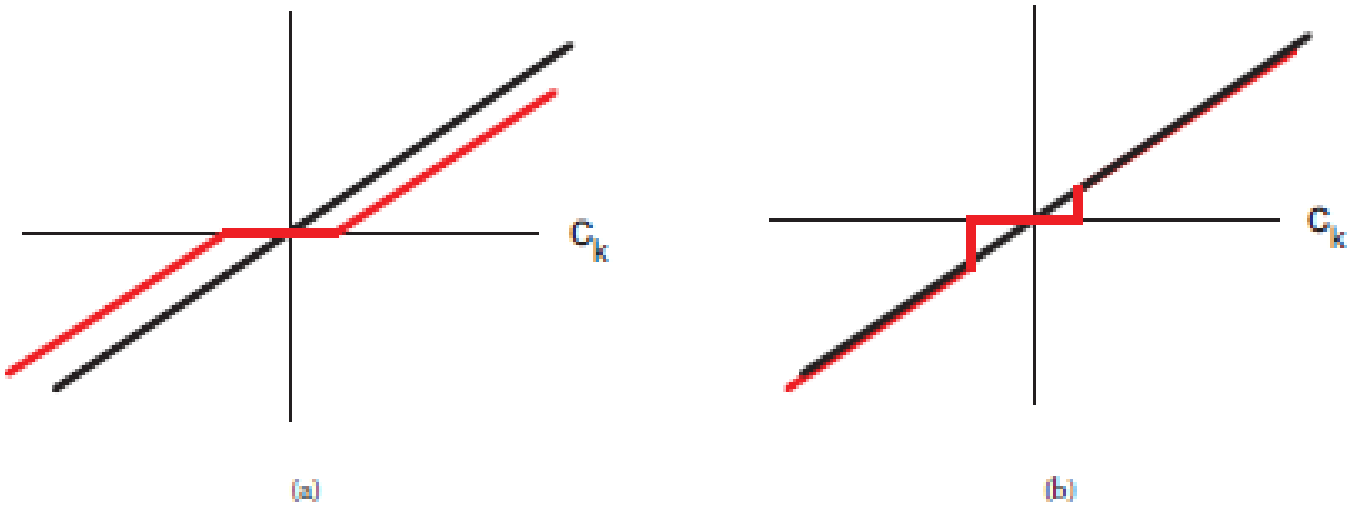
$$\widehat{w}_j = soft\left(\frac{c_j}{a_j}, \frac{\lambda}{a_j}\right)$$

where $soft(\frac{c_j}{a_j}, \frac{\lambda}{a_j}) \equiv sign\left(\frac{c_j}{a_j}\right)\left(\left|\frac{c_j}{a_j}\right| - \frac{\lambda}{a_j}\right)_+$

$\left(\left|\frac{c_j}{a_j}\right| - \frac{\lambda}{a_j}\right)_+ = \max\left(\left(\left|\frac{c_j}{a_j}\right| - \frac{\lambda}{a_j}\right), 0\right)$, the positive part. This is called **soft thresholding**

Soft thresholding vs hard thresholding

This is illustrated in Figure 13.5(a), plotting $\widehat{w}_j \; vs \; c_j$. The black line is the line $w_j = c_j/a_j$ corresponding to the least squares fit.
The red line, which represents the regularized estimate shifts the red line down (or up) by λ, except when $-\lambda \leq c \leq \lambda$ in which case it sets $w_j = 0$.



- **Figure 13.5 Left: soft thresholding. The flat region is the interval [-λ, +λ]. Right: hard thresholding.**

Hard thresholding

- By contrast, in Figure 13.5(b), **hard thresholding** is illustrated. This sets values of w_j to 0 if $-\lambda \leq c_j \leq \lambda$, but it does not shrink the values of w_j outside of this interval
- The slope of the soft thresholding line does not coincide with the diagonal, which means that even large coefficients are shrunk towards zero; consequently lasso is a biased estimator.
- This is undesirable, since if the likelihood indicates (via c_j) that the coefficient w_j should be large, we do not want to shrink it.

l_1 regularization algorithms

- The various algorithms solving l_1 regularization problems are
  - Coordinate descent
  - LARS
  - Proximal Gradient

Coordinate descent algorith,

- Lasso objective has the form

$$f(\boldsymbol{w}) = RSS(\boldsymbol{w}) + \lambda\left|\left|\boldsymbol{w}\right|\right|_1$$

Where $RSS(\boldsymbol{w}) = \left|\left|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\right|\right|_2^2$

- The coordinate descent aims at obtaining w_j one at a time
- In particular, we can solve for the j'th coefficient with all the others held fixed.

Coordinate descent algorithm

Algorithm 13.1: Coordinate descent for lasso (aka shooting algorithm)

Initialize $\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$;
repeat
  for $j = 1, \ldots, D$ **do**
    $a_j = 2\sum_{i=1}^{n} x_{ij}^2$;
    $c_j = 2\sum_{i=1}^{n} x_{ij}(y_i - \mathbf{w}^T\mathbf{x}_i + w_j x_{ij})$;
    $w_j = \text{soft}(\frac{c_j}{a_j}, \frac{\lambda}{a_j})$;
until *converged*;

Disadvantage : Since the variables are solved one at a time, the process is time consuming.
Thus, methods such as LARS and proximal gradient are used

Proximal gradient method

- Consider a convex objective function of the form

$$f(\boldsymbol{w}) = L(\boldsymbol{w}) + R(\boldsymbol{w})$$

where $L(\boldsymbol{w})$, representing loss, is convex and differentiable, whereas $R(\boldsymbol{w})$, representing regularization, is convex and maybe non-differentiable

- Solution is to split the functions individually to solve the loss function and the regularization function (using proximal gradient method).
- Named “proximal” because the non-smooth function is solved using its ‘proximity operator’

Proximal operator

- For example, consider $L(\boldsymbol{w}) = RSS(\boldsymbol{w})$, let $RSS(\boldsymbol{w}) = \left|\left|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\right|\right|_2^2$
- For the the design matrix $X = I$, the objective function becomes $f(\boldsymbol{w}) = R(\boldsymbol{w}) + \frac{1}{2}\left|\left|\boldsymbol{w} - y\right|\right|_2^2$
- The minimizer of the above function is given by the **proximal operator ($\boldsymbol{prox_R(w)}$)** for the convex set R.

$$prox_R(\boldsymbol{y}) = argmin_z \left(R(z) + \frac{1}{2}\left|\left|z - \boldsymbol{y}\right|\right|_2^2 \right)$$

- Intuitively, a point that minimizers R and also close (proximal) to $\boldsymbol{wy}$ is returned

Examples of Proximal operators

- If $R(\boldsymbol{w}) = \lambda\left|\left|\boldsymbol{w}\right|\right|_1$, the proximal operator is given by component wise soft thresholding

$$prox_R(\boldsymbol{w}) = soft(\boldsymbol{w}, \lambda)$$

- If $R(\boldsymbol{w}) = \lambda\left|\left|\boldsymbol{w}\right|\right|_0$, the proximal operator is given by component wise hard thresholding

$$prox_R(\boldsymbol{w}) = hard(\boldsymbol{w}, \sqrt{2\lambda})$$

- If $R(\boldsymbol{w}) = I_C(\boldsymbol{w})$, the proximal operator is given by the projection on to the set C:

$$prox_R(\boldsymbol{w}) = argmin_{z \in \mathrm{C}}\left|\left|z - \boldsymbol{w}\right|\right|_2^2 = proj_C(\boldsymbol{w})$$

Examples of Projection (proximal) operators

- For some convex sets, it is easy to compute the **projection operator**.
- For example, to project onto the rectangular set defined by the box constraints $C = \{\boldsymbol{w}: l_j \leq w_j \leq u_j\}$, we can use

$$proj_C(\boldsymbol{w})_j = \begin{cases} l_j & w_j \leq l_j \\ w_j & l_j \leq w_j \leq u_j \\ u_j & w_j \geq u_j \end{cases}$$

Examples of Projection (proximal) operators

- To project onto the Euclidean ball

$C = \{\boldsymbol{w}: \left|\left|\boldsymbol{w}\right|\right|_2 \leq 1\}$ we can use

$$proj_C(\boldsymbol{w}) = \begin{cases} \dfrac{\boldsymbol{w}}{\left|\left|\boldsymbol{w}\right|\right|_2} & \left|\left|\boldsymbol{w}\right|\right|_2 > 1 \\ \boldsymbol{w} & \left|\left|\boldsymbol{w}\right|\right|_2 \leq 1 \end{cases}$$

- To project onto the 1-norm ball

$C = \{\boldsymbol{w}: \left|\left|\boldsymbol{w}\right|\right|_1 \leq 1\}$ we can use

$proj_C(\boldsymbol{w}) = \text{soft}(\boldsymbol{w}, \lambda)$, where

$\lambda = 0$, if $\left|\left|\boldsymbol{w}\right|\right|_1 \leq 1$ *and*

λ is the soln of: $\sum_{j=1}^{D} \max\left(\left|w_j\right| - \lambda, 0\right) = 1$, otherwise

Proximal gradient method

- Proximal operator used in gradient descent routine
- Consider the function $f(\boldsymbol{w}) = L(\boldsymbol{w}) + R(\boldsymbol{w})$
- Basic idea: minimize a simple quadratic approximation to the loss function, centered on $\boldsymbol{w}^k$ for k^{th} iteration.
- Consider the Taylor series expansion of $\mathrm{L}(z)$

$$L(z) = L(\boldsymbol{w}^k) + g_k^T(z - \boldsymbol{w}^k) + \frac{1}{2t_k}\left|\left|z - \boldsymbol{w}^k\right|\right|_2^2$$

where $g_k = \nabla L(\boldsymbol{w}^k)$ and a simple approximation of Hessian of the loss $\nabla^2 L(\boldsymbol{w}^k) \approx \frac{1}{t_k} I$

Thus,

$$\boldsymbol{w}^{k+1} = argmin_z(R(z) + \underbrace{L(\boldsymbol{w}^k) + g_k^T(z - \boldsymbol{w}^k) + \frac{1}{2t_k}\left|\left|z - \boldsymbol{w}^k\right|\right|_2^2}_{L(z)})$$

Proximal gradient method

- Dropping the terms independent of z, multiplying by t_k and rewriting the above equation in terms of proximal operator for k^{th} iteration:

$$\boldsymbol{w}^{k+1} = argmin_z \left[t_k R(z) + \frac{1}{2} \left|\left| z - u_k \right|\right|_2^2 \right] = prox_{t_k R}(u_k)$$

$$u_k = \boldsymbol{w}^k - t_k g_k$$

$$g_k = \nabla L(\boldsymbol{w}^k)$$

(Verify : $g_k^T \left(z - \boldsymbol{w}^k\right) + \frac{1}{2t_k} \left|\left| z - \boldsymbol{w}^k \right|\right|_2^2 = \frac{1}{2t_k} \left|\left| z - u_k \right|\right|_2^2$)

- If $R(\boldsymbol{w}) = 0$, this is equivalent to **gradient descent**.
- If $R(\boldsymbol{w}) = I_C(\boldsymbol{w})$, this method is equivalent to **projected gradient descent**
- If $R(\boldsymbol{w}) = \lambda \left|\left| \boldsymbol{w} \right|\right|_1$, the method is known as **iterative soft thresholding**

References

- Machine Learning – A probabilistic perspective, Kevin P Murphy