CSE 546 HW #notes

Sam Kowash

October 16, 2018

(1) LASSO

Selects for sparse predictor w. We may want this either for efficiency or human-legibility. How do we do this? Well, greedy approach adds features one at a time based on improvement in test error, but that's pretty hacky. How do we know when to stop? How do we avoid just including a billion features?

Looking for sparse results is a type of regularization; we want to penalize feature overselection. This motivates the lasso objective:

$$\hat{w}_{lasso} = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda \|w\|_1. \tag{1.1}$$

This punishes big vectors w, which is what we want. Fact: for any $\lambda \geqslant 0$ for which $\hat{w}_r$ finds the minimum, there exists $\nu \geqslant 0$ such that

$$\hat{w}_r = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 \text{ subject to } r(w) \leqslant \nu. \tag{1.2}$$

That is, regularized regression problems can always be reframed as constrained optimization.

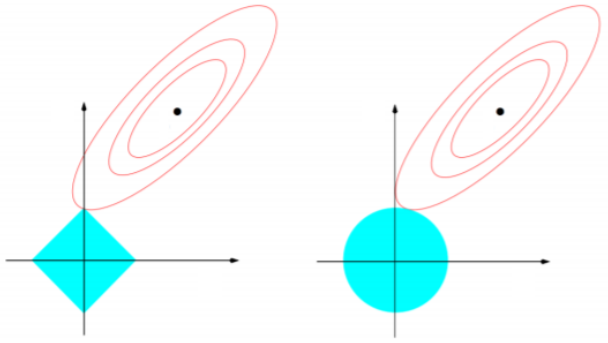

Figure 1.1: Lasso on left, ridge on right; lasso prefers solutions along coordinate axes (i.e. sparse)

If we incorporate an offset,

$$\hat{w}, \hat{b} = \arg\min_{w,b} \sum_{i=1}^{n} \left(y_i - (x_i^T w + b)\right)^2 + \lambda \|w\|_1, \tag{1.3}$$

but joint optimization is a pain, so let's prefer to de-mean our data. This is still actually kind of tricky minimization; the 1-norm isn't differentiable at the origin and this complicates things. Do by coordinate descent, minimize one direction at a time. This is guaranteed to approach the optimum for lasso (which is nice), but how do we pick our order of coordinate descent? Options,

- Random each time
- Round robin
- Try to pick "important" coordinates (biases us).

Let's see an example. Take $j \in \{1, \ldots, d\}$.

$$\sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda \|w\|_1 = \sum_{i=1}^{n} \left(y_i - \sum_{k=1}^{d} x_{i,k} w_k\right)^2 + \lambda \sum_{k=1}^{d} |w_k| \tag{1.4}$$

$$= \sum_{i=1}^{n} \left(\left(y_i - \sum_{k \neq j} x_{i,k} w_k\right) - x_{i,j} w_j\right)^2 + \lambda \sum_{k \neq j} |w_k| + \lambda |w_j|. \tag{1.5}$$

So set $\hat{w}_k = 0$ for $k \in \{1, \ldots, d\}$ and loop over points:

$$r_i^{(j)} = \sum_{k \neq j} x_{i,j} \hat{w}_k \tag{1.6}$$

$$\hat{w}_j = \arg\min_{w_j} \sum_{i=1}^{n} \left(r_i^{(j)} - x_{i,j} w_j\right)^2 + \lambda |w_j|. \tag{1.7}$$

Pulling out one 1-d problem at a time! Works b/c lasso objective is *separable*. Except... this isn't actually a lot better. Hard to optimize because of pointy bit. Need to extend some concept of derivative and convexity. Traditional definition for fn is that lines b/t points on $f(x)$ lie above $f(x)$ (epigraph is convex). We need a different one here:

$$f(y) \geqslant f(x) + \nabla f(x)^T (y - x) \quad \forall x, y \tag{1.8}$$

This amounts to saying that there's a "supporting hyperplane" touching the epigraph at x such that the epigraph lies entirely on one side of the plane. If the function is differentiable at x, this is going to be the tangent plane. If not, we may have *many* options, and these are called *subgradients* (denoted ∂_{w_j} for subgradient set at w_j). We call differentiable functions extremized at x where $\nabla f(x) = 0$, and similarly we will call other functions extremized when $f(x)$ admits 0 as a subgradient at x.

OK, so how do we actually take subgradients and set them to zero? Consider example,

$$\partial_{w_j} \left(\sum_{i=1} \left(r_i^{(j)} - x_{i,j} w_j\right)^2 + \lambda |w_j|\right) = \begin{cases} a_j w_j - c_j - \lambda & \text{if } w_j < 0 \\ [-c_j - \lambda, -c_j + \lambda] & \text{if } w_j = 0 \\ a_j w_j - c_j + \lambda & \text{if } w_j > 0 \end{cases}, \tag{1.9}$$

where

$$a_j = \left(\sum_{i=1}^{n} x_{i,j}^2\right), \qquad c_j = 2 \left(\sum_{i=1}^{n} r_i^{(j)} x_{i,j}\right). \tag{1.10}$$

This tells us how to do our minimization (look for regime that contains zero as subgrad):

$$\hat{w}_j = \begin{cases} \frac{c_j + \lambda}{a_j} & \text{if } c_j < -\lambda \\ 0 & \text{if } |c_j| \leqslant \lambda \\ \frac{c_j - \lambda}{a_j} & \text{if } c_j > \lambda \end{cases} \tag{1.11}$$

where, recall,

$$a_j = \sum_{i=1}^{n} x_{i,j}^2 \qquad\qquad c_j = 2\sum_{i=1}^{n}\left(y_i - \sum_{k\neq j} x_{i,k}w_k\right)x_{i,j}. \tag{1.12}$$

This central flattening behavior provides "soft thresholding"; can make predictor entries identically zero depending on strength of regularization.

(2) Classification problems

Different from regression! Same principles though. Need a loss function; what is? Let's start with binary classification, want to learn $f : X \to Y$ where X contains features, $Y \in \{0, 1\}$ is target class. Natural loss is 0/1 function: $\mathbf{1}\{f(X) \neq Y\}$. Expected loss is then

$$\mathbb{E}_{XY}\left[\mathbf{1}\{f(X) \neq Y\}\right] = \mathbb{E}_X\left[\mathbb{E}_{Y|X}\left[\mathbf{1}\{f(x) \neq Y\} \mid X = x\right]\right] \tag{2.1}$$

$$= \mathbb{E}_X\Big\{\mathbf{1}\{f(x) = 1\}\mathbb{P}(Y = 0 \mid X = x) + \mathbf{1}\{f(x) = 0\}\mathbb{P}(Y = 1 \mid X = x)\Big\}. \tag{2.2}$$

Supposing we know $P(Y \mid X)$, the Bayes optimal classifier is

$$f(x) = \arg\max_y \mathbb{P}(Y = y \mid X = x). \tag{2.3}$$

How do we actually estimate $\mathbb{P}(Y \mid X)$? Well, can't do linear, and we're out of tricks now. Need a new trick; some function that goes from $\mathbb{R}^d$ to $[0, 1]$ (called link function). A nice option is the sigmoid/logistic curve:

$$\mathbb{P}(Y = 0 \mid X, W) = \frac{1}{1 + \exp\left[w_0 + \sum_i w_i X_i\right]}. \tag{2.4}$$

Still a sort of linear model in terms of what we do to our data variables. Note that w_0 applies a horizontal shift, and the w_i "stretch" the curve. Note a nice property for binary classification, which is that

$$\frac{\mathbb{P}(Y = 1 \mid w, X)}{\mathbb{P}(Y = 0 \mid w, X)} = \exp\left[w_0 + w^T X\right]. \tag{2.5}$$

Reasonable to make our rule to classify as 1 if ratio is greater than 1, classify as 0 if ratio is less than 1. Equivalent result,

$$\ln \frac{\mathbb{P}(Y = 1 \mid w, X)}{\mathbb{P}(Y = 0 \mid w, X)} = w_0 + \sum_i w_i X_i \to \begin{cases} < 0 \implies \text{classify as } 0 \\ > 0 \implies \text{classify as } 1 \end{cases}. \tag{2.6}$$

Alternative formulation (conditional likelihood): say $y \in \{-1, 1\}$ instead of $\{0, 1\}$ so we can then write

$$\mathbb{P}(Y = y \mid x, w) = \frac{1}{1 + \exp\left(-yw^T x\right)} \tag{2.7}$$

and find the MLE:

$$\hat{w}_{\text{MLE}} = \arg\max_w \prod_{i=1}^{n} \mathbb{P}(y_i \mid x_i, w) \tag{2.8}$$

$$\hat{w}_{\text{MLE}} = \arg\min_w \sum_{i=1}^{n} \underbrace{\log\left(1 + \exp\left(-y_i x_i^T w\right)\right)}_{\sigma(y_i x_i^T w)} \tag{2.9}$$

Call the objective $J(w)$ (logistic loss): what the hell does it look like? Is it convex? (Yes.) How do we minimize it? Note that for an argument z, $\sigma(z) \sim |z|$ as $z \to -\infty$ and $\sigma(z) \to 0$ as $z \to +\infty$. In between, does some nonsense. So $\sigma(z)$ is easy to understand, but what about $J(w)$? Generally, no closed form in terms of w, can't just take derivative and set to zero. It turns out, moreover, that if we happen to have a w that correctly classifies every point (by dumb luck; linearly separable data set), then $y_i x_i^T w$ is strictly positive and so for $t > 0$, $y_i x_i^T (tw) > y_i x_i^T w$ and we will always prefer to infinitely extend our w along our "good" direction to continue reducing loss. This breaks the classifier. Sigmoid classifiers become step functions. We *need* to regularize this conditional log likelihood objective.