

Linear Regression

Gautam Kamath

Calculus Review: Derivatives and Gradients

- Derivative

- Let $f(x) : \mathbf{R} \rightarrow \mathbf{R}$ be a scalar-valued function of one variable
- Derivative $f'(x) = \frac{df}{dx} : \mathbf{R} \rightarrow \mathbf{R}$
- Example: if $f(x) = x^2$, then $f'(x) = 2x$

- Gradient

- Let $f(v) : \mathbf{R}^d \rightarrow \mathbf{R}$ be a scalar-valued function of d variables
- Gradient $\nabla f(v) = \left(\frac{\partial f}{\partial v_1}, \dots, \frac{\partial f}{\partial v_d} \right) : \mathbf{R}^d \rightarrow \mathbf{R}^d$
- Example: if $f(v) = v_1 v_2^2 + v_3^3$, then $\nabla f(v) = (v_2^2, 2v_1 v_2, 3v_3^2)$
- Most important mathematical object in this course (?)!

A bit more Calculus: Hessian

- Hessian
 - Let $f(v) : \mathbf{R}^d \rightarrow \mathbf{R}$ be a scalar-valued function of d variables
 - Hessian $\nabla^2 f(v) : \mathbf{R}^d \rightarrow \mathbf{R}^{d \times d}$

$$\begin{bmatrix} \frac{\partial^2 f}{\partial v_1^2} & \cdots & \frac{\partial^2 f}{\partial v_1 \partial v_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial v_1 \partial v_d} & \cdots & \frac{\partial^2 f}{\partial v_d^2} \end{bmatrix}$$

Statistical Learning (more general than before)

- Setup: Given $(x_1, y_1), \dots, (x_n, y_n) \sim_{i.i.d.} P$
 - This time: feature vector $x_i \in \mathbf{R}^d$, but label $y_i \in \mathbf{R}$ (as opposed to ± 1 before)
- Problem defined by a *loss function* $\ell_w(x, y)$
 - Sometimes written as $\ell(w, x, y)$. w is the *parameter vector*.
- Goal: output $\arg \min_w \mathbf{E}_{(x,y) \sim P} [\ell_w(x, y)]$
 - Parameter vector w which minimizes loss given new point from distribution
- Generalization of previous lecture's goal
 - $\ell_w(x, y) = 0$ if $\text{sign}(\langle w, x \rangle) = y$, $\ell_w(x, y) = 1$ if $\text{sign}(\langle w, x \rangle) \neq y$
 - Goal: output $\arg \min_w \mathbf{E}_{(x,y) \sim P} [\ell_w(x, y)] = \arg \min_w \Pr_{(x,y) \sim P} [\text{sign}(\langle w, x \rangle) \neq y]$

Empirical Risk Minimization (ERM)

- Goal: output $\arg \min_w \mathbf{E}_{(x,y) \sim P} [\ell_w(x, y)]$
 - But we don't know the distribution P – we only have (x_i, y_i) 's from P
 - What do we do?
- Minimize the expected loss over the *training dataset*
 - i.e., the *empirical* distribution
- Output

$$\arg \min_w \frac{1}{n} \sum_{i=1}^n \ell_w(x_i, y_i)$$

- Converges to desired quantity as $n \rightarrow \infty$
- Goal is to find w which minimizes some function

Convexity and Optimization

- How do we pick a good loss function?
 - Depends on structure we assume in data, consider e.g., perceptron
 - Also may depend on convenience, especially for optimization
- (Draw picture of convex function)
- Function f is convex iff for all $\lambda \in [0,1]$, x_1, x_2 ,
$$f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda)f(x_2)$$
- Alternatively: $f''(x) \geq 0$ (1D functions) or $\nabla^2 f(x) \succcurlyeq 0$
 - Matrix $M \in \mathbf{R}^{d \times d}$ is *positive semidefinite* (PSD) iff $v^T M v \geq 0$ for all vectors $v \in \mathbf{R}^d$
 - Also written $M \succcurlyeq 0$
- (Draw non-convex function, local, global min, saddle point)

Convexity

- Convexity is nice because it makes optimization easier
- Fermat's condition: If x is a local extremum of a function f , then $\nabla f(x) = \vec{0}$. Additionally, if f is convex, then the converse is true: $\nabla f(x) = \vec{0}$ implies that x is a local extremum.
- Tying back to ERM: goal is to find $\arg \min_w \frac{1}{n} \sum_{i=1}^n \ell_w(x_i, y_i)$
- If ℓ_w is convex (in w), then ERM is equivalent to finding w^* such that

$$\nabla_w \frac{1}{n} \sum_{i=1}^n \ell_{w^*}(x_i, y_i) = \frac{1}{n} \sum_{i=1}^n \nabla_w \ell_{w^*}(x_i, y_i) = \vec{0}$$

Linear Regression

- (Draw tipping example on board)
- Loss function $\ell_w(x, y) = (y - \langle w, x \rangle)^2$
 - Pays the square of the *residual* (draw on board)
- Resulting predictor is $y = \langle w, x \rangle$
- Use padding trick to allow line to not go through origin
 - Replace x by $[x, 1]$ and w by $[w, b]$
- Could imagine more complicated scenarios, e.g., polynomial regression (draw on board)
 - But not today

Looking closer at the loss function

- Loss function: $\sum (y_i - \langle w, x_i \rangle)^2$
- Let $A \in \mathbf{R}^{n \times d}$ and $z \in \mathbf{R}^n$ be the feature vectors and labels stacked
 - (draw on board)
- Then loss function is equivalently $\|Aw - z\|_2^2$
 - First entry of $Aw - z$ is $\langle x_1, w \rangle - y_1$, square and sum (draw on board)

The loss function is convex

- Loss fn $\|Aw - z\|_2^2 = (Aw - z)^T(Aw - z) = (w^T A^T - z^T)(Aw - z)$
 $= w^T A^T Aw - z^T Aw - w^T A^T z + z^T z$
 $= w^T A^T Aw - 2w^T A^T z + z^T z$
- Claim: if $f(x) = x^T Ax + x^T b + c$, then $\nabla f(x) = (A + A^T)x + b$
- Thus $\nabla_w \|Aw - z\|_2^2 = 2A^T Aw - 2z^T A$
- Checking the Hessian, $\nabla_w^2 \|Aw - z\|_2^2 = 2A^T A \succcurlyeq 0$
 - Why? Since $2v^T A^T Av = 2\|Av\|_2^2 \geq 0$ for any vector v
- Therefore the loss function is convex

Optimizing Least Squares

- So what if the loss function is convex?
- Setting the gradient to 0 minimizes the function
- Set $\nabla_w \|Aw - z\|_2^2 = 2A^T Aw - 2z^T A$ to be 0
- That is, find \hat{w} such that $A^T A \hat{w} = A^T z$
- Could solve for \hat{w} by computing $\hat{w} = (A^T A)^{-1} A^T z$
 - ...but requires $A^T A$ to be invertible
 - ...and could be slow, or imprecise if ill-conditioned
- Better to just solve the linear system $A^T A \hat{w} = A^T z$ for unknown \hat{w}

Where did squared loss come from?

An MLE perspective

- Gaussian distribution $N(\mu, \sigma^2)$ (draw picture)
 - $f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$
- Maximum Likelihood principle: find model parameters which maximize the probability of the observed data
- $\arg \max_{\text{model parameters}} \Pr[\text{observed data} | \text{model parameters}]$
- Needs some generative assumption on observed data wrt model parameters
 - i.e., $(x, y) \sim P_w$, where w are the model parameters
- A common assumption: $y = \langle w, x \rangle + z$, where $z \sim N(0, \sigma^2)$

Deriving the MLE

$$\begin{aligned}y &= \langle w, x \rangle + z, \text{ where } z \sim N(0, \sigma^2) \\ \hat{w} &= \arg \max_w \Pr[(x_1, y_1), \dots, (x_n, y_n) | w] \\ &= \arg \max_w \prod_i \Pr[(x_i, y_i) | w] \\ &= \arg \max_w \prod_i \Pr[y_i | x_i, w] \Pr[x_i | w] \\ &= \arg \max_w \prod_i \Pr[y_i | x_i, w] \\ &= \arg \max_w \prod_i \Pr[y_i | x_i, w] \\ &= \arg \max_w \log \left(\prod_i \Pr[y_i | x_i, w] \right)\end{aligned}$$

$$\begin{aligned}&= \arg \max_w \sum_i \log(\Pr[y_i | x_i, w]) \\ &\quad (\text{Note: } y_i | x_i, w \sim N(\langle w, x \rangle, \sigma^2)) \\ &= \arg \max_w \sum_i \log \left(\frac{1}{\sqrt{2\pi}\sigma} \exp \left(-\frac{(y - \langle w, x \rangle)^2}{2\sigma^2} \right) \right) \\ &= \arg \max_w \sum_i \log \left(\frac{1}{\sqrt{2\pi}\sigma} \right) + \log \left(\exp \left(-\frac{(y - \langle w, x \rangle)^2}{2\sigma^2} \right) \right) \\ &= \arg \max_w \sum_i -\frac{(y - \langle w, x \rangle)^2}{2\sigma^2} \\ &= \arg \min_w \sum_i (y - \langle w, x \rangle)^2 \\ &\text{Loss function is the squared error!}\end{aligned}$$

Regularization

- (Draw regression picture, with polynomial vs linear fit)
- Choosing the right model is important!
 - Sometimes simpler models are better
 - E.g., a more complex model which gets 0 training error may be worse than a simpler model which gets larger training model
- Tikhonov regularization or Ridge regression
 - $\arg \min_w \|Aw - z\|_2^2 + \lambda \|w\|_2^2$
- Lasso
 - $\arg \min_w \|Aw - z\|_2^2 + \lambda \|w\|_1$
 - Prefers *sparse* solutions

Hyperparameter selection

- Types of datasets
 - Training, validation, test
- Use validation to make sure you didn't overfit to training data
- Can try different hyperparameters using validation – but not too many/adaptively or you'll overfit to training + validation
 - E.g., commit to $\lambda = \{0.01, 0.1, 0.5, 1\}$, train all models on training data, choose the best one via the validation set
- What if we have no training set?

Cross Validation

- Split training data into k sets (draw on board), e.g. $k = 10$ is common

For each λ :

For $i = 1$ to k :

$w_{\lambda,i}$ = train on all data but split i with hyperparameter λ

$\text{perf}_{\lambda,i}$ = performance of $w_{\lambda,i}$ on the split i

$$\text{perf}_{\lambda} = \sum_i \text{perf}_{\lambda,i}$$

Return λ which has the biggest perf_{λ}

- Note: often turn regularization “off” for validation/test
 - $\|Aw - z\|_2^2 + \lambda\|w\|_2^2$ when training, but $\|Aw - z\|_2^2$ on validation