Lecture 8 - logistic regression & optimization

September 28, 2021 4:27 PM

Today: • logistic regression
• numerical optimization

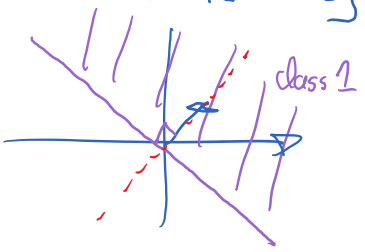
logistic regression model:

$p(y=1|x) = \sigma(w^T x)$ $\quad y=\{0,1\}$ decision rule: $\mathbb{1}\{w^T x > 0\}$

$p(y=0|x) = 1-\sigma(w^T x) = \sigma(-w^T x)$

$Y|X=x$ is Bernoulli $(\sigma(w^T x))$

$p(y|x) = \sigma(w^T x)^y \sigma(-w^T x)^{1-y}$



[if $y=\{\pm 1\}$ "Rademacher" R.V.

given $(x_i, y_i)_{i=1}^n$, encode $p(y|x) = \sigma(y w^T x)$

max. conditional log-likelihood to estimate $\hat{w}_{MCL}$

$$\ell(w) = \sum_{i=1}^n \log p(y_i|x_i,w) = \sum_{i=1}^n y_i \log \sigma(w^T x_i) + (1-y_i)\log \sigma(-w^T x_i)$$

$\nabla_w \sigma(w^T x) = x\left[\sigma(w^T x)\sigma(-w^T x)\right]$ let $v_i \triangleq w^T x_i$

$$\nabla \ell(w) = \sum_{i=1}^n x_i \left[y_i \sigma(v_i)\sigma(-v_i) + (1-y_i)(-1)\sigma(-v_i)\sigma(v_i)\right]$$

($\sigma(v_i)$ and $\sigma(-v_i)$ cancel)

$$= \sum_i x_i \left[y_i \left[\sigma(v_i)+\sigma(-v_i)\right] - \sigma(v_i)\right]$$

$$\boxed{\nabla \ell(w) = \sum_{i=1}^n x_i \left[y_i - \sigma(w^T x_i)\right]}$$

solve for $\nabla \ell(w) = 0 \Rightarrow$ need to solve a transcendental equation because $\frac{1}{1+\exp(-w^T x_i)}(\cdots) = 0$ (solve for this) → need to use numerical methods

contrast to linear regression $\nabla \ell(w) = \sum_{i=1}^n x_i [y_i - w^T x_i]$ (linear in w)

numerical optimization:

want to minimize $f(w)$ s.t. $w \in \mathbb{R}^d$ (unconstrained) [suppose f is diff.]

1) gradient descent (1st order method)

start w_0

iterate: $w_{t+1} = w_t - \gamma_t \nabla f(w_t)$

stopping criterion: if $\|\nabla f(w_t)\|^2 < \delta$ then stop

e.g. $\delta = 10^{-6}$

note: if f μ-strongly convex $\Rightarrow$ if $\|\nabla f(w_t)\|^2 \le s$

$\Leftrightarrow f(w) - \frac{\mu}{2}\|w\|^2$ is convex in w $\Rightarrow f(w_t) - \min_w f(w) \le \frac{1}{2\mu}\|\nabla f(w_t)\|^2$

Step-size rules:

a) constant step-size: $\gamma_t = \frac{1}{L}$ ← Lipschitz continuity constant for ∇f

ie. $\|\nabla f(w) - \nabla f(w')\| \le L\|w - w'\|$

b) decreasing step-size rule [this is more common for stochastic optimization]

$\gamma_t = \frac{c}{t}$ ← constant

usually want: $\sum_t \gamma_t = \infty$, $\sum_t \gamma_t^2 < \infty$

e.g. $f(w) \triangleq \mathbb{E}_\xi\, g(w, \xi)$

$w_{t+1} = w_t - \gamma_t \nabla_w g(w, \xi_t)$

e.g. $g(w, \xi) = \ell(y_i, h_w(x_i))$ for ERM, $\xi = (x_i, y_i)$

c) Choose γ_t by "line search": $\min_{\gamma \in \mathbb{R}} f(w_t + \gamma d_t)$

d_t: direction for update, e.g. $-\nabla f(w_t)$

Costly in general → instead do approximate search, e.g. Armijo line search (see Boyd's book)

17h20

Newton's method (2nd order method)

motivation: minimizing a quadratic approximation

Hessian $[H(w_t)]_{ij} = \frac{\partial^2 f(w_t)}{\partial w_i \partial w_j}$

Taylor expansion at w_t: $f(w) = f(w_t) + \nabla f(w_t)^T (w - w_t) + \frac{1}{2}(w - w_t)^T H(w_t)(w - w_t) + O(\|w - w_t\|^3)$

$\triangleq Q_t(w)$; $O(\|w - w_t\|^3)$: Taylor's remainder

$= Q_t(w) + O(\|w - w_t\|^3)$

$Q_t(w)$: quadratic model approx.

w_{t+1} ⇝ obtain by minimizing $Q_t(w)$

$\nabla_w Q_t(w) = 0$ $\quad \nabla f(w_t) + H(w_t)(w - w_t) \overset{\text{want}}{=} 0$

$\Rightarrow w - w_t = -H^{-1}(w_t)\nabla f(w_t)$

$$\boxed{w_{t+1} = w_t - H^{-1}(w_t)\nabla f(w_t)}$$ Newton's update

H^{-1}: inverse Hessian

$d_t = H^{-1}\nabla f$

$\sim O(d^3)$ time to compute in general and $O(d^2)$ space $\Leftarrow\!\Rightarrow$ $H d_t = \nabla f$

Note for hwk 2: solve for $H_t d_t = \nabla f(w_t)$

$$\min_d \|H_t d - \nabla f(w_t)\|^2$$

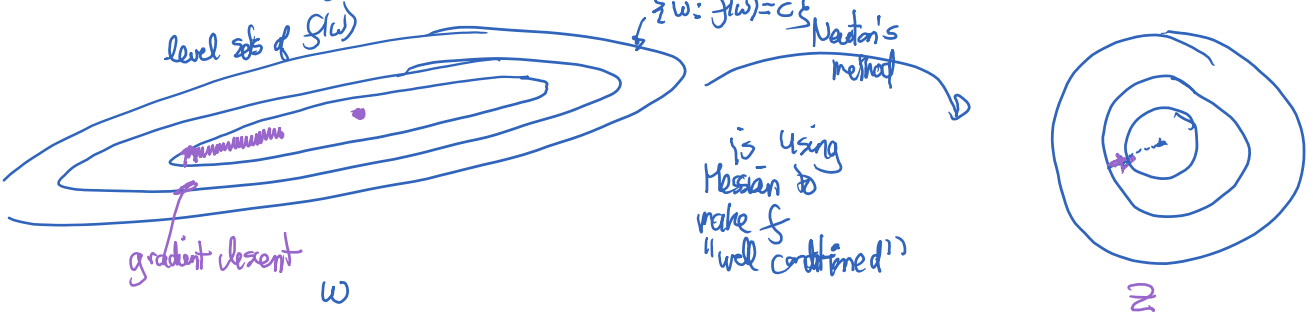
use `numpy.linalg.lstsq` to solve this → much more numerically stable

Damped Newton: you add a step-size to stabilize Newton's method

$$w_{t+1} = w_t - \underbrace{\alpha_t}_{\text{step-size}} H^{-1}(w_t) \nabla f(w_t)$$

Why Newton's method?

- much faster convergence in # of iterations vs. gradient descent
- affine covariant ⇒ method is invariant to rescaling of variables



$\frac{1}{2} w^T H w = c$ $\quad$ $\frac{1}{2} w^T P^T \Sigma P w = c$ $\quad\leftrightarrow\quad$ $\frac{1}{2} z^T z = c$

$H = P^T \Sigma P$ (H is symmetric ! PSD) $\quad$ diagonal $\quad z = H^{1/2} w = \Sigma^{1/2} P w$

exercise to reader:

$$z_{t+1} = z_t - \gamma \nabla f(z_t)$$

$$w_{t+1} = w_t - \gamma H^{-1} \nabla f(w_t)$$

Newton's method for logistic regression: IRLS

recall for $\ell(w)$: $\nabla \ell(w) = \sum_{i=1}^n x_i [y_i - \sigma(w^T x_i)]$

$$H(\ell(w)) = -\sum_{i=1}^n x_i x_i^T \sigma(w^T x_i)\sigma(-w^T x_i)$$

$$v^T H v = -\sum_i \underbrace{(v^T x_i)(x_i^T v)}_{(x_i^T v)^2 \geq 0} \underbrace{\sigma(-)\sigma(-)}_{\geq 0}$$

$v^T H v \leq 0 \quad \forall v \in \mathbb{R}^d$

ie. $H \preceq 0$

ie. concave fct

Newton's is maximizing instead of min.

notation: recall $X = \begin{pmatrix} -x_1^T- \\ \vdots \end{pmatrix}$

Notation: recall $X_{n \times d} = \begin{pmatrix} -x_1^T- \\ \vdots \\ -x_n^T- \end{pmatrix}$

Newton's is maximizing instead of min.

let $\mu_i \triangleq \sigma(w^T x_i) \in]0,1[$ $\quad \mu_w \triangleq \begin{pmatrix} \vdots \\ \sigma(w_t^T x_i) \\ \vdots \end{pmatrix}$

$\nabla \ell(w) = \sum_i x_i [y_i - \mu_i] = X^T(y - \mu)$

Hessian $= -\sum_i x_i x_i^T \mu_i (1-\mu_i) = -X^T D(w) X$ where $D_{ii} \triangleq \mu_i(1-\mu_i)$ (diagonal matrix)

Newton's update: $w_{t+1} = w_t - (-X^T D_t X)^{-1} X^T (y - \mu_t)$

$= (X^T D_t X)^{-1} \left[(X^T D_t X) w_t + X^T (y - \mu_t) \right]$

$$w_{t+1} = (X^T D_t X)^{-1} [X^T D_t z_t] \quad \text{where} \quad z_t \triangleq X w_t + D_t^{-1}(y - \mu_t)$$

this is a solution to a "weighted least square problem"

$$\min_w \| D^{1/2} (z_t - Xw) \|_2^2$$

weights, new target

$$\sum_i (z_i - w^T x_i)^2 \, d_{ii}$$

Compare with Gaussian noise model for least square

$$\sum_i \frac{(y_i - w^T x_i)^2}{2\sigma^2}$$

Newton's method for logistic regression = iterated reweighted least squares (IRLS)

Big data logistic regression:

- big d $\Rightarrow$ cannot do $O(d^2)$ or $O(d^3)$ operations $\Rightarrow$ first order methods
- if n is large, you cannot use batch methods $\nabla f(w) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(w)$ $O(nd)$

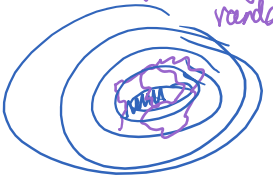
gradient of one datapoint

batch gradient

instead you use "incremental gradient methods"

e.g. stochastic gradient descent (SGD): $w_{t+1} = w_t - \alpha_t \nabla f_{i_t}(w_t)$ $O(d)$

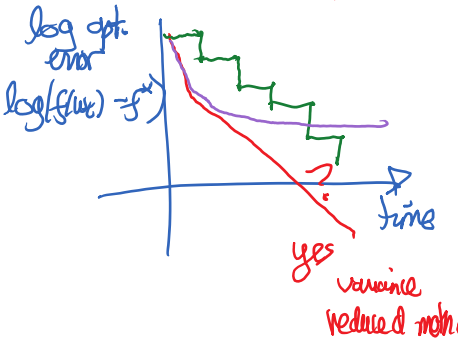
where i_t is picked unif. at random



SGD → cheap updates, but slower convergence per iteration

batch gradient → expensive, but faster convergence

batch gradient → expensive updates, but faster convergence



→ SAG: stochastic average gradient

GD: $w_{t+1} = w_t - \gamma_t \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w_t)$

SAG [2012]: $w_{t+1} = w_t - \gamma_t \frac{1}{n} \sum_{i=1}^{n} v_i$ v_i's memory

where $v_i = \nabla f_i(w_0)$

at each t, update $v_{i_t} = \nabla f_{i_t}(w_t)$

SAGA [2014]: $w_{t+1} = w_t - \gamma_t \left(\nabla f_{i_t}(w_t) + \frac{1}{n} \sum_{j=1}^{n} v_j - v_{i_t} \right)$

($\nabla f_{i_t}(w_t)$ − v_{i_t}: variance reduction correction)

(default method for logistic regression in Scikit-learn)

SVRG