EE5907/EE5027 Week 4: Logistic Regression

BT Thomas Yeo

ECE, CIRC, Sinapse, Duke-NUS, HMS

Last Week Recap

- Univariate Gaussian
  - ML, MAP, posterior predictive
- Naïve Bayes Classifier
  - Generative classifier: $p(x, y \mid \boldsymbol{\theta}) = p(y \mid \lambda)p(x \mid y, \eta)$
  - Features are independent given class labels
  - ML, MAP or posterior predictive strategies for estimating model parameters and classifying new test sample

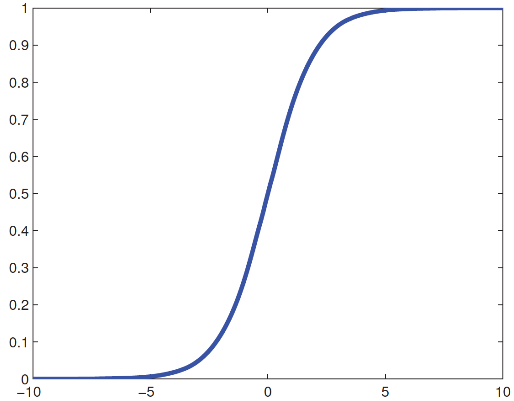
This Week

- Discriminative Classifier
  - Logistic regression
- Basic optimization techniques

Logistic Regression

Logistic Regression Model

- Features x and label y:
  - Generative classifier: build joint model $p(x, y) = p(y)p(x|y)$, estimate parameters of joint model from training data, then compute $p(y|x)$
  - Discriminative classifier: build model $p(y|x)$, estimate parameters of model from data, then compute $p(y|x)$
- Logistic regression
  - Despite name, it's a discriminative classifier
  - Binary case: $p(y|x, w) = \text{Ber}(y|\mu(x, w)) = \text{Ber}(y|\text{sigm}(w^T x))$



$$\text{sigm}(x) = \frac{1}{1 + e^{-x}}$$

Logistic Regression Advantages

- Easy to train
- Easy to interpret:

$$p(y=1|x) = \frac{1}{1+e^{-w^Tx}}$$

$$p(y=0|x) = 1 - \frac{1}{1+e^{-w^Tx}} = \frac{e^{-w^Tx}}{1+e^{-w^Tx}} = \frac{1}{1+e^{w^Tx}}$$

  - $\log \frac{p(y=1|x)}{p(y=0|x)} = w^Tx$ $\longleftarrow$ log odds
  - Suppose $x(1)$ is # cigarettes per day, $x(2)$ is minutes of exercise, $y =$ cancer, $w = (1.3, -1.1)$ $\Longrightarrow$ for every extra cigarette, cancer risk increases by factor of $e^{1.3}$

- Easy to extend to multi-class (covered in hidden slides)
- Easy to be nonlinear by using kernels (not covered)

Training Logistic Regression Model

Estimating Logistic Regression Parameters & Classifying New Samples

- Given training set $\{x_{1:N}, y_{1:N}\}$, where $x_{1:N}$ are feature vectors of N training samples and $y_{1:N}$ are corresponding class labels
- Already specify discriminative model $p(y \mid x, \mathrm{w})$
- First estimate $\hat{w} = \mathrm{argmax}_{\mathrm{w}}\, p(y_{1:N} \mid x_{1:N}, \mathrm{w})$
  - We did not specify joint distribution p(x, y) in our modeling, so we cannot do something like last week (naïve Bayes): $\boldsymbol{\theta}_{ML} = \mathrm{argmax}_{\boldsymbol{\theta}}\, p(x_{1:N}, y_{1:N} \mid \boldsymbol{\theta})$ or $\boldsymbol{\theta}_{MAP} = \mathrm{argmax}_{\boldsymbol{\theta}}\, p(\boldsymbol{\theta} \mid x_{1:N}, y_{1:N})$
- To predict label $\tilde{y}$ of test data $\tilde{x}$, plug $\hat{w}$ into posterior $p(\tilde{y} = c \mid \tilde{x}, \hat{w})$ and pick class c with highest posterior probability

Minimizing Negative Log Likelihood

$$
\begin{aligned}
\hat{w} &= \operatorname*{argmax}_w p(y_{1:N} | x_{1:N}, w) \\
&= \operatorname*{argmax}_w \log p(y_{1:N} | x_{1:N}, w) \\
&= \operatorname*{argmax}_w \log \prod_{i=1}^{N} p(y_i | x_i, w) \\
&= \operatorname*{argmax}_w \sum_{i=1}^{N} \log p(y_i | x_i, w) \\
&= \operatorname*{argmin}_w - \sum_{i=1}^{N} \log p(y_i | x_i, w) \stackrel{\triangle}{=} \operatorname*{argmin}_w NLL(w)
\end{aligned}
$$

Log is monotonic

Samples are independent

log ab = log a + log b

argmax_x f(x) = argmin_x -f(x)

- $NLL(w)$ stands for negative log likelihood.
- There is no real advantage to minimizing versus maximizing, but I am following book's convention

Expanding Negative Log Likelihood

- Negative log likelihood:

$$\log p(y_i = 1|x_i, w) = \log \frac{1}{1 + \exp(-w^T x_i)} = \log \mu_i$$

$$\log p(y_i = 0|x_i, w) = \log(1 - p(y_i = 1|x_i, w)) = \log(1 - \mu_i)$$

$$NLL(w) = -\sum_{i=1}^{N} \log p(y_i|x_i, w) = -\sum_{i=1}^{N} [y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i)]$$

  - When $y_i = 1$, first term $= \log \mu_i$, while $(1 - y_i) = 0$, so second term $= 0$
  - When $y_i = 0$, first term $= 0$, while $(1 - y_i) = 1$, so second term $= \log(1 - \mu_i)$

Gradient & Hessian

- From previous slide:

$$\log p(y_i = 1|x_i, w) = \log \frac{1}{1+\exp(-w^T x_i)} = \log \mu_i$$

$$\log p(y_i = 0|x_i, w) = \log(1 - p(y_i = 1|x_i, w)) = \log(1 - \mu_i)$$

$$NLL(w) = -\sum_{I=1}^{N} \log p(y_i|x_i, w) = -\sum_{I=1}^{N} [y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i)]$$

- Derivatives (see non-graded assignment)

$$g = \frac{d}{dw} NLL(w) = \sum_{i=1}^{N} (\mu_i - y_i) x_i = X^T (\mu - y)$$

D x 1 D x N N x 1 N x 1

  - $g = D \times 1$ vector, $X^T = [x_1, \cdots, x_N]$ ($D \times N$ matrix), μ, y are $N \times 1$ column vectors obtained by concatenating μ_i and y_i

Gradient & Hessian

- From previous slide:

$$\log p(y_i = 1|x_i, w) = \log \frac{1}{1+\exp(-w^T x_i)} = \log \mu_i$$

$$\log p(y_i = 0|x_i, w) = \log(1 - p(y_i = 1|x_i, w)) = \log(1 - \mu_i)$$

$$NLL(w) = -\sum_{I=1}^{N} \log p(y_i|x_i, w) = -\sum_{I=1}^{N} [y_i \log \mu_i + (1 - y_i)\log(1-\mu_i)]$$

- Derivatives (see non-graded assignment)

$$g = \frac{d}{dw} NLL(w) = \sum_{i=1}^{N} (\mu_i - y_i) x_i = X^T(\mu - y)$$

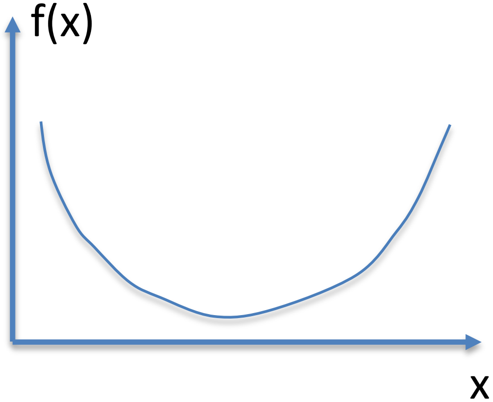
$$H = \frac{d}{dw} g(w)^T = \sum_{i=1}^{N} \mu_i (1-\mu_i) x_i x_i^T = X^T S X,$$

D x D

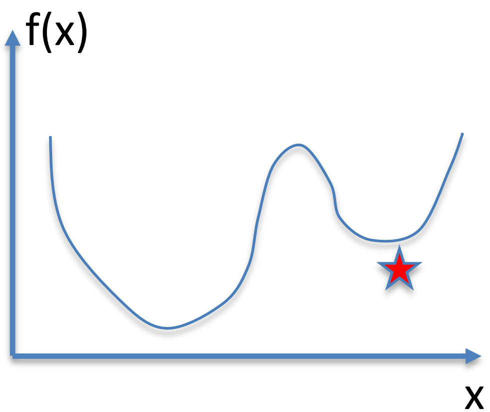
D x N N x N N x D

  - $g = D \times 1$ vector, $X^T = [x_1, \cdots, x_N]$ ($D \times N$ matrix), μ, y are $N \times 1$ column vectors obtained by concatenating μ_i and y_i
  - $H = D \times D$ matrix, $S = N \times N$ diagonal matrix (zeros except for diagonals), where i-th diagonal is $\mu_i(1-\mu_i)$

Interlude: Convexity & Global Minimum



2^{nd} derivative is positive everywhere => convex => any local minimum is a global minimum



2^{nd} derivative NOT positive everywhere => not convex => local minimum (e.g., red star) might not be global minimum

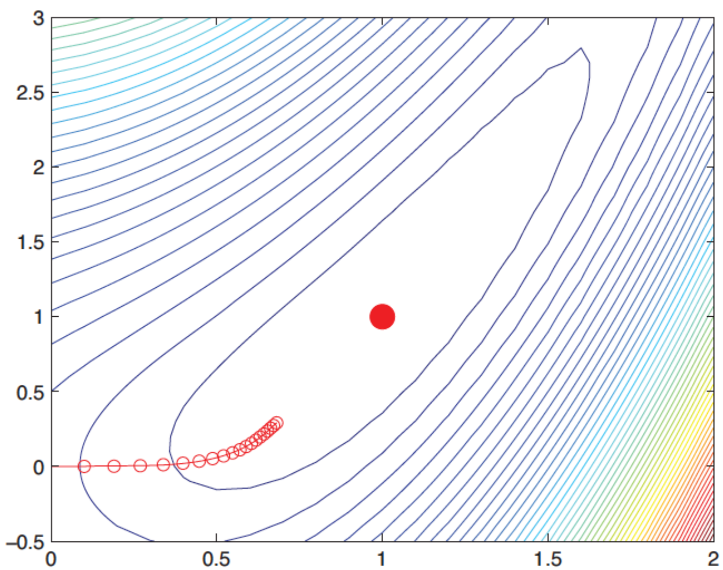
- In higher dimensions, "positive 2^{nd} derivative" condition becomes "positive-definite Hessian"
- Definition: D x D matrix H is positive definite if for all D x 1 vector z (which are not zero vectors), $z^T H z > 0$

- (See non-graded assignment): In logistic regression, H(w) is positive definite for all w, hence NLL(w) is convex => unique global minimum
- There is no-closed form solution, so we need to do some numerical optimization
- Convexity is nice because numerical optimization can only give us local optimum, so with convexity, local optimum = global optimum

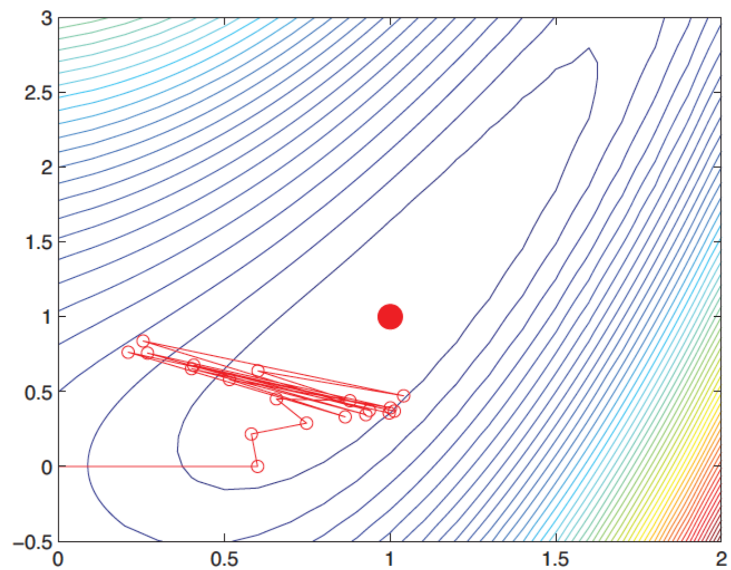
Interlude: Numerical Optimization

Gradient Descent

- Goal: $\text{argmin}_\theta f(\theta)$
- Gradient descent
  - Initialize $\theta = \theta_0$
  - $\theta_{k+1} = \theta_k + \eta_k d$ where $\eta_k > 0$ (called step size or learning rate) & $d =$ descent direction
- Steepest descent if $d = -\nabla f(\theta_k)$
  - $\nabla f(\theta_k) \neq 0 \implies$ there exist η_k such that $f(\theta_{k+1}) < f(\theta_k)$



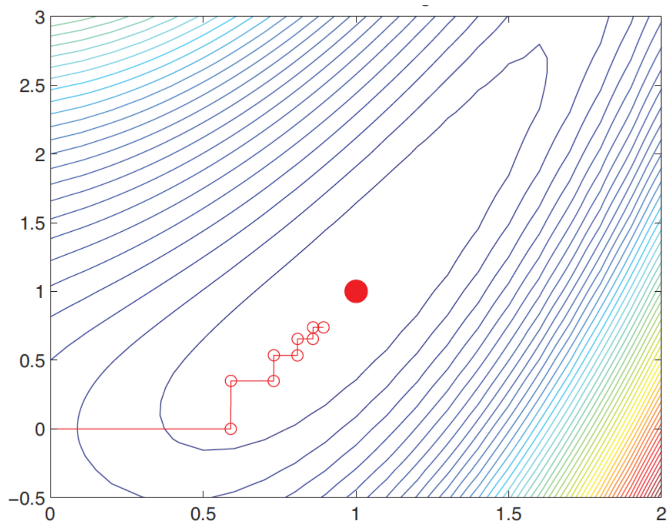
Small fixed η: convergence very slow
(gradients are perpendicular to level sets)



Big fixed η: might not converge
(gradients are perpendicular to level sets)

Gradient Descent

- Goal: $\text{argmin}_\theta f(\theta)$
- Gradient descent
  - Initialize $\theta = \theta_0$
  - $\theta_{k+1} = \theta_k + \eta_k d$ where $\eta_k > 0$ (called step size or learning rate) & $d =$ descent direction
- Steepest descent if $d = -\nabla f(\theta_k)$
  - $\nabla f(\theta_k) \neq 0 \implies$ there exist η_k such that $f(\theta_{k+1}) < f(\theta_k)$
  - Find best η by line search: $\hat{\eta} = \text{argmin}_\eta f(\theta_k + \eta d)$ (http://numerical.recipes/)
  - Line search leads to zig-zag through parameter space



Gradient Descent

- Goal: $\text{argmin}_\theta f(\theta)$
- Gradient descent
  - Initialize $\theta = \theta_0$
  - $\theta_{k+1} = \theta_k + \eta_k d$ where $\eta_k > 0$ (called step size or learning rate) & $d =$ descent direction
- Steepest descent if $d = -\nabla f(\theta_k)$
  - $\nabla f(\theta_k) \neq 0 \implies$ there exist η_k such that $f(\theta_{k+1}) < f(\theta_k)$
  - Find best η by line search: $\hat{\eta} = \text{argmin}_\eta f(\theta_k + \eta d)$ (http://numerical.recipes/)
  - Line search leads to zig-zag through parameter space
    - Let $\phi(\eta) = f(\theta_k + \eta d) \implies \phi'(n) = d^T \nabla f(\theta_k + \eta d)$
    - $\phi'(\hat{n}) = 0 \implies \nabla f = 0$ (local minimum) or $d \perp \nabla f(\theta_k + \hat{\eta} d)$

best η

descent direction in current iteration

negative of descent direction at next iteration

Gradient Descent

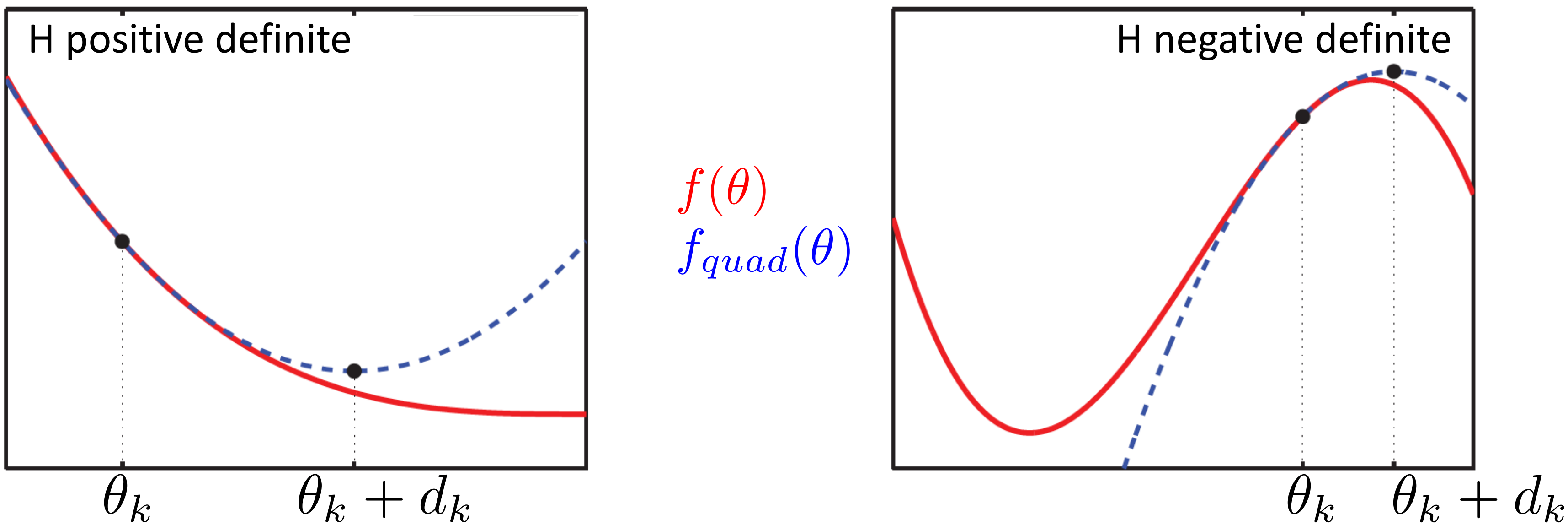
- Goal: $\operatorname{argmin}_\theta f(\theta)$
- Gradient descent
  - Initialize $\theta = \theta_0$
  - $\theta_{k+1} = \theta_k + \eta_k d$ where $\eta_k > 0$ (called step size or learning rate) & $d =$ descent direction
- Steepest descent if $d = -\nabla f(\theta_k)$
  - $\nabla f(\theta_k) \neq 0 \implies$ there exist η_k such that $f(\theta_{k+1}) < f(\theta_k)$
  - Find best η by line search: $\hat{\eta} = \operatorname{argmin}_\eta f(\theta_k + \eta d)$ (http://numerical.recipes/)
  - Line search leads to zig-zag through parameter space
    - Let $\phi(\eta) = f(\theta_k + \eta d) \implies \phi'(n) = d^T \nabla f(\theta_k + \eta d)$
    - $\phi'(\hat{n}) = 0 \implies \nabla f = 0$ (local minimum) or $d \perp \nabla f(\theta_k + \hat{\eta} d)$
- Momentum: $\theta_{k+1} = \theta_k - \eta_k \nabla f(\theta_k) + \mu_k(\theta_k - \theta_{k-1})$, where $0 \leq \mu_k \leq 1$
- Conjugate gradient (see http://numerical.recipes/)
  - Great for quadratic objectives $\theta^T A \theta$ or linear system $A\theta = b$: converge in D iterations (where D is dimension of θ)

Newton's Method

- Taylor expansion: $f(\theta_k + d_k) \approx f_{quad} = f(\theta_k) + d_k^T \nabla f + \frac{1}{2} d_k^T H d_k$
  - First (∇f) and second (H) derivatives evaluated at θ_k
- Differentiate f_{quad} with respect to d_k

$$\nabla f + H d_k = 0 \implies d_k = -H^{-1} \nabla f$$

  - f_{quad} minimum for $d_k = -H^{-1} \nabla f$ (assuming H positive definite)



Newton's Method

- Newton's update: $\theta_{k+1} = \theta_k + \eta_k d_k$, where $H_k d_k = -\nabla f_k$
- H has to be positive definite, else d_k may not decrease cost function
  - To see this: $\langle -\nabla f, -H^{-1}\nabla f \rangle > 0$ if H positive definite $\implies -H^{-1}\nabla f$ within 90 degrees direction of steepest descent direction, so will definitely decrease f for small η
  - If Hessian not positive definite, can use Levenberg-Marquardt in the case of nonlinear least squares (wikipedia has easy explanation)

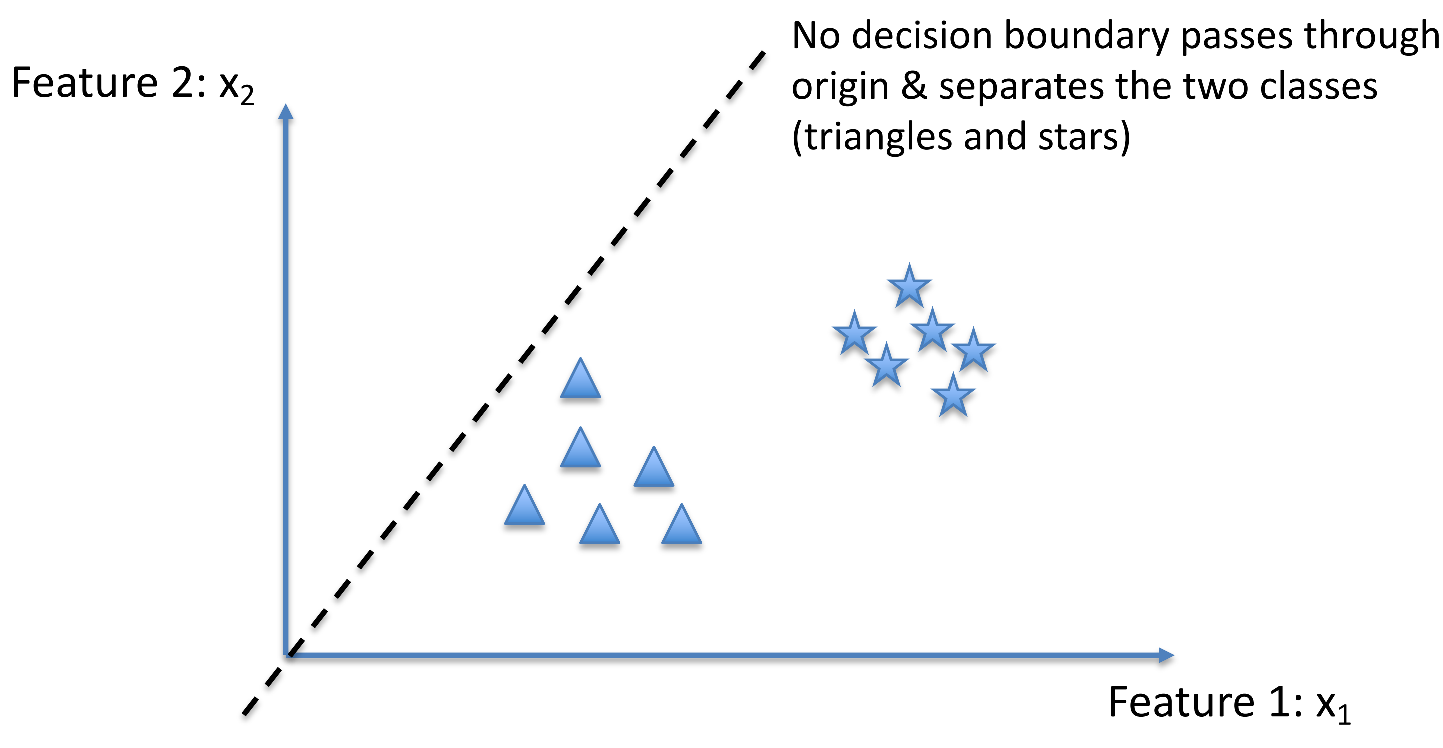
Algorithm 8.1: Newton's method for minimizing a strictly convex function

```
Initialize θ_0;
for k = 1, 2, ... until convergence do
    Evaluate g_k = ∇f(θ_k);
    Evaluate H_k = ∇²f(θ_k);
    Solve H_k d_k = −g_k for d_k;
    Use line search to find stepsize η_k along d_k;
    θ_{k+1} = θ_k + η_k d_k;
```

Optimizing Logistic Regression Parameters + Need For a Bias Term

Quick Interlude: Why we need bias term?

- In practice, p(y = 1 | x, w) = sigm($w_0 + w^T x$)
- Without bias term w_0, if all features (x) = 0, then p(y = 1 | x = $\vec{0}$, w) = sigm($w^T x$) = sigm($w^T \vec{0}$)= 0.5
- This means that decision boundary (locations where there are equal posterior probability of two classes) must past through origin



Newton's Method for Logistic Regression

- Previous slides:

$$g = \frac{d}{dw} NLL(w) = X^T(\mu - y)$$
$$H = \frac{d}{dw} g(w)^T = X^T S X,$$

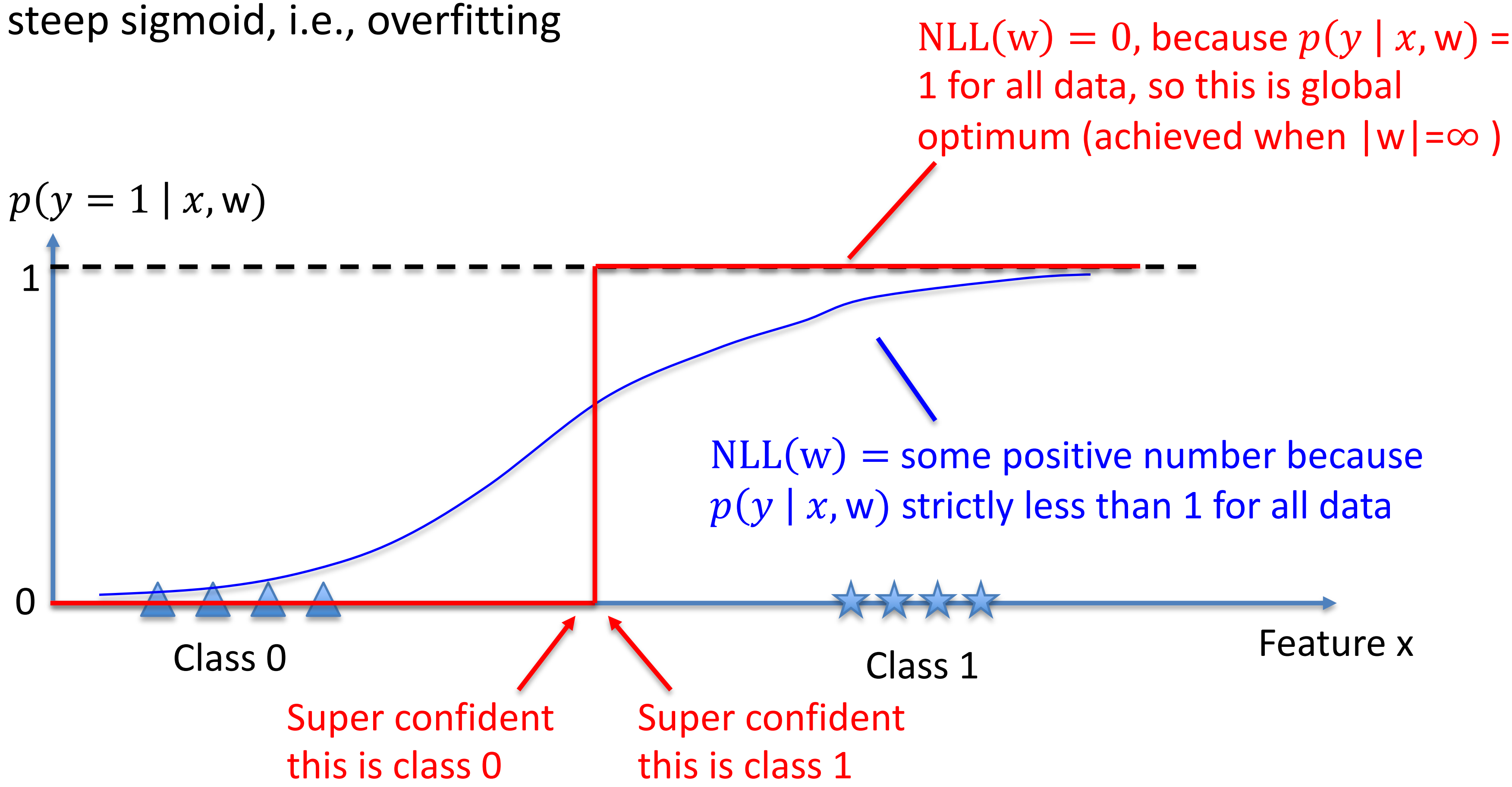
  - $g = D \times 1$ vector, $X^T = [x_1, \cdots, x_N]$ ($D \times N$ matrix), μ, y are $N \times 1$ column vectors obtained by concatenating μ_i and y_i
  - $H = D \times D$ matrix, $S = N \times N$ diagonal matrix (zeros except for diagonals), where i-th diagonal is $\mu_i(1 - \mu_i)$

- To introduce bias term, concatenate 1 to start of x_i, so length of feature vector is $D + 1$. Let's denote new feature vector $\mathbf{x}_i$
  - Still model $p(y_i = 1 | x_i, \mathbf{w}) = \text{sigm}(\mathbf{w}^T \mathbf{x}_i)$, so now $\mathbf{w}$ is $(D+1) \times 1$ vector, whose first element is now bias term
  - Above g and H can be computed, replacing x_i with $\mathbf{x}_i$, w with $\mathbf{w}$
  - Initialize by $w = \vec{0}_{D+1}$
  - Repeat until convergence: $\mathbf{w}_{k+1} = \mathbf{w}_k - H_k^{-1} g_k$ (no need for line search in assignment)

One More Modification: Regularization (Regularizations are additional constraints to reduce overfitting)

Why do we need regularization?

- In general, $\mathrm{NLL(w)} = -\sum_{i=1}^{N} \log p(y_i|x_i, w) \geq 0$ because $0 \leq p(y_i|x_i, w) \leq 1$
- When data is linearly separable, ||w|| becomes infinity, resulting in infinitely steep sigmoid, i.e., overfitting



l_2 Regularization

- l_2 regularization prevents w from exploding
  - $NLL_{reg}(\mathbf{w}) = NLL(\mathbf{w}) + \frac{1}{2}\lambda\mathbf{w}^T\mathbf{w}$
  - If $\mathbf{w}$ is big, then $\frac{1}{2}\lambda\mathbf{w}^T\mathbf{w}$ is big, so $NLL_{reg}(\mathbf{w})$ is big. Since we are minimizing $NLL_{reg}(\mathbf{w})$, this means big $\mathbf{w}$ is discouraged

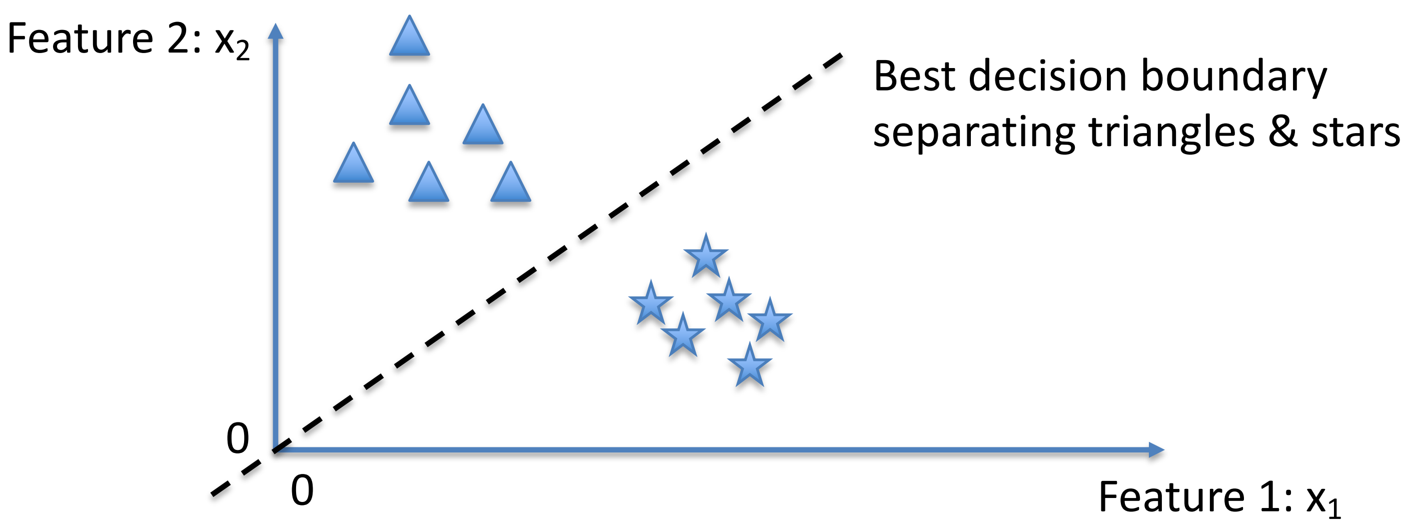
- New gradient and hessian:

$$g_{reg}(\mathbf{w}) = g(\mathbf{w}) + \lambda\mathbf{w}$$
$$H_{reg}(\mathbf{w}) = H(\mathbf{w}) + \lambda I,$$

where I is a $(D+1) \times (D+1)$ identity matrix

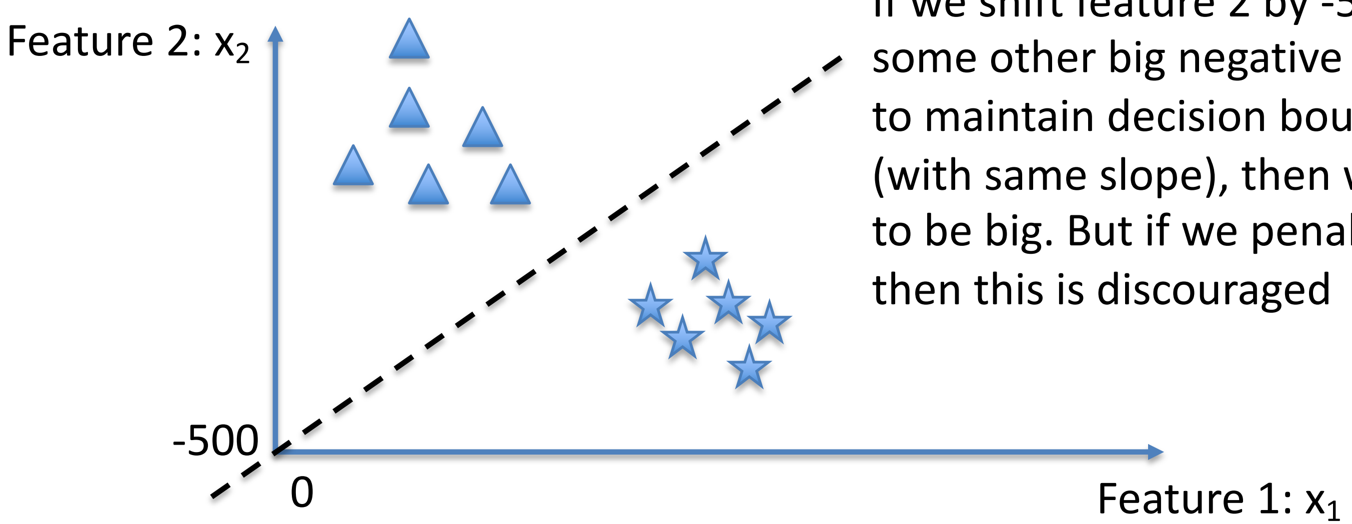
But we do not want to regularize bias term

- Another wrinkle: we don't want to regularize bias term w_0
  - Suppose features are 2-dimensional: x_1 and x_2
  - Decision boundary corresponds to $\text{sigm}(w_0 + w_1 x_1 + w_2 x_2) = 0.5 \implies w_0 + w_1 x_1 + w_2 x_2 = 0 \implies x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}$
  - If we put l_2 regularization on w_0 means we encourage decision boundary to pass close to origin



But we do not want to regularize bias term

- Another wrinkle: we don't want to regularize bias term w_0
  - Suppose features are 2-dimensional: x_1 and x_2
  - Decision boundary corresponds to $\text{sigm}(w_0 + w_1 x_1 + w_2 x_2) = 0.5 \implies w_0 + w_1 x_1 + w_2 x_2 = 0 \implies x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}$
  - If we put l_2 regularization on w_0 means we encourage decision boundary to pass close to origin



If we shift feature 2 by -500 (or some other big negative number), to maintain decision boundary (with same slope), then w_0 needs to be big. But if we penalize w_0, then this is discouraged

Exclude Bias from l_2 Regularization

- Exclude w_0 from regularization:
  - $NLL_{reg}(\mathbf{w}) = NLL(\mathbf{w}) + \frac{1}{2}\lambda w^T w$
  - $\mathbf{w}$ is $(D+1) \times 1$ vector, while w is $D \times 1$ vector ($\mathbf{w}$ without the first element)

$$g_{reg}(\mathbf{w}) = g(\mathbf{w}) + \lambda \begin{pmatrix} 0_{1\times 1} \\ w_{D\times 1} \end{pmatrix}$$

$$H_{reg}(\mathbf{w}) = H(\mathbf{w}) + \lambda \begin{pmatrix} 0_{1\times 1} & \cdots \\ \vdots & I_{D\times D} \end{pmatrix}$$

row is all zero

column is all zero

Summary

- Discriminative Classifier $p(y \mid x, w)$: Logistic Regression
- Numerical optimization
  - Gradient descent
  - Newton's method
  - Hessian positive definite everywhere => cost function convex => unique global minimum and every local minimum is global minimum
- Logistic regression
  - NLL is convex – optimize with Newton's method
  - Bias term
  - Regularization

Optional Reading

- Notes based on
  - KM Chapter 8.1, 8.3, 8.3 (beware of typos)