
Lecture Notes for Machine Learning in Python

Professor Eric Larson
Logistic Regression

Class Logistics and Agenda

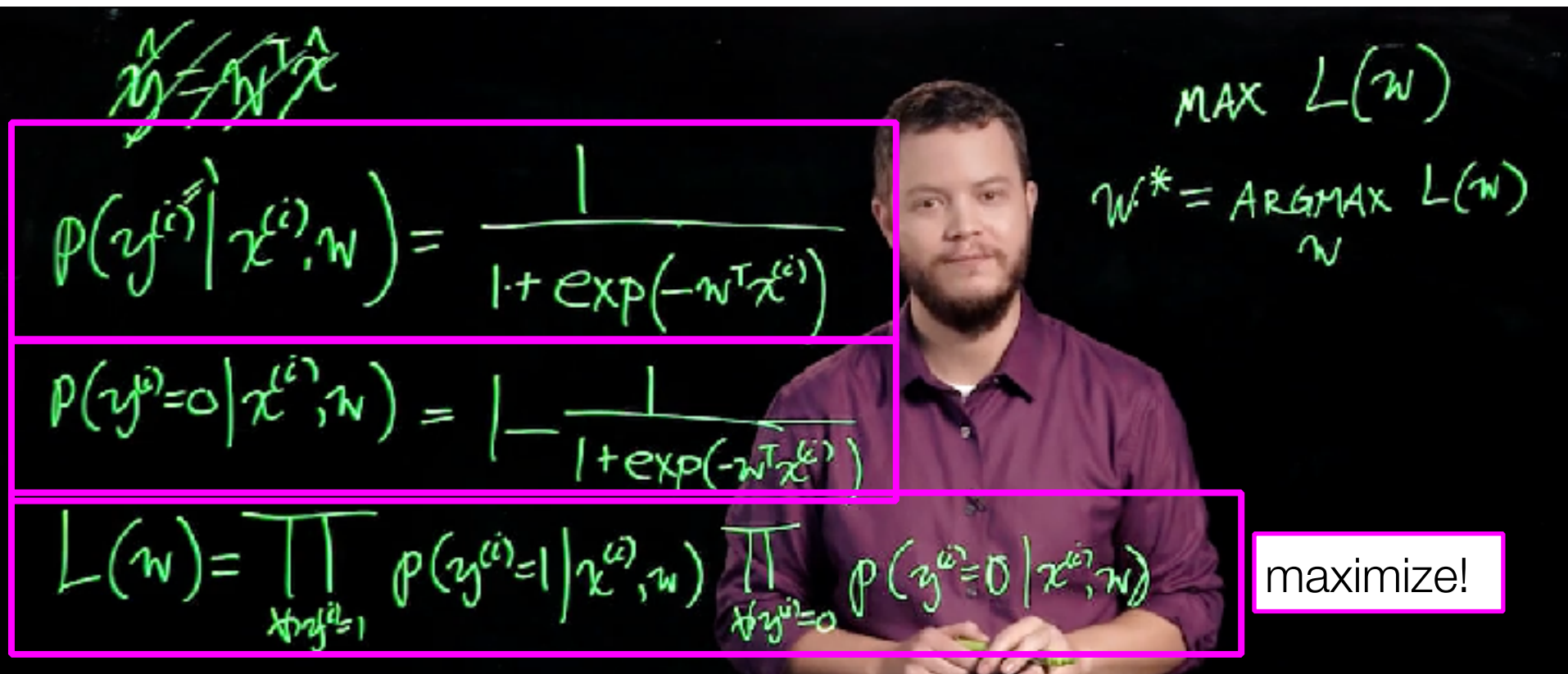
- Welcome back to lecture!
- Logistics
 - Nothing due this week
 - Next week: A3
- Agenda
 - Logistic Regression
 - Solving
 - Programming
 - Finally some real python!

Solving Logistic Regression



Setting Up Binary Logistic Regression

- From flipped lecture:



The image shows a man with a beard and short brown hair, wearing a purple button-down shirt, standing in front of a chalkboard. The chalkboard contains several handwritten equations in green chalk. The equations are as follows:

- Top left: $\hat{y} = w^T \hat{x}$
- Top right: $\text{MAX } L(w)$
- Middle right: $w^* = \underset{w}{\text{ARGMAX}} L(w)$
- Bottom left (highlighted with a pink box): $p(y^{(i)} | x^{(i)}, w) = \frac{1}{1 + \exp(-w^T x^{(i)})}$
- Bottom middle (highlighted with a pink box): $p(y^{(i)} = 0 | x^{(i)}, w) = 1 - \frac{1}{1 + \exp(-w^T x^{(i)})}$
- Bottom right (highlighted with a pink box): $L(w) = \prod_{y^{(i)}=1} p(y^{(i)}=1 | x^{(i)}, w) \prod_{y^{(i)}=0} p(y^{(i)}=0 | x^{(i)}, w)$

A pink box highlights the three equations on the left side of the chalkboard. A white box with a pink border is located at the bottom right of the chalkboard area, containing the text "maximize!".

Binary Solution for Update Equation

- Video Supplement (also on canvas):
 - <https://www.youtube.com/watch?v=FGnoHdjFrJ8>
- General Procedure:
 - Simplify $L(w)$ with logarithm, $l(w)$

$$l(w) = \sum_i y^{(i)} \ln(g(w^T x^{(i)})) + (1 - y^{(i)}) \ln(1 - g(w^T x^{(i)}))$$

- Take Gradient

$$= - \sum_i (y^{(i)} - g(w^T x^{(i)})) x_j^{(i)}$$

- Use gradient inside update equation for \mathbf{w}

Binary Solution for Update Equation

- Use gradient inside update equation for \mathbf{w}

$$\underbrace{w_j}_{\text{new value}} \leftarrow \underbrace{w_j}_{\text{old value}} + \underbrace{\eta \sum_{i=1}^M (y^{(i)} - g(x^{(i)})) x_j^{(i)}}_{\text{gradient}}$$

$$w \leftarrow w + \eta \sum_{i=1}^M (y^{(i)} - g(x^{(i)})) x^{(i)}$$

Reinvent sklearn **Logistic Regression**

Programming
Vectorization
Regularization
Multi-class extension



Other Tutorials:

<http://blog.yhat.com/posts/logistic-regression-python-rodeo.html>

http://scikit-learn.org/stable/auto_examples/linear_model/plot_iris_logistic.html

For Next Lecture

- **Next time:** Gradient based optimization for logistic regression
- **Next Next time:** SVMs in-class assignment

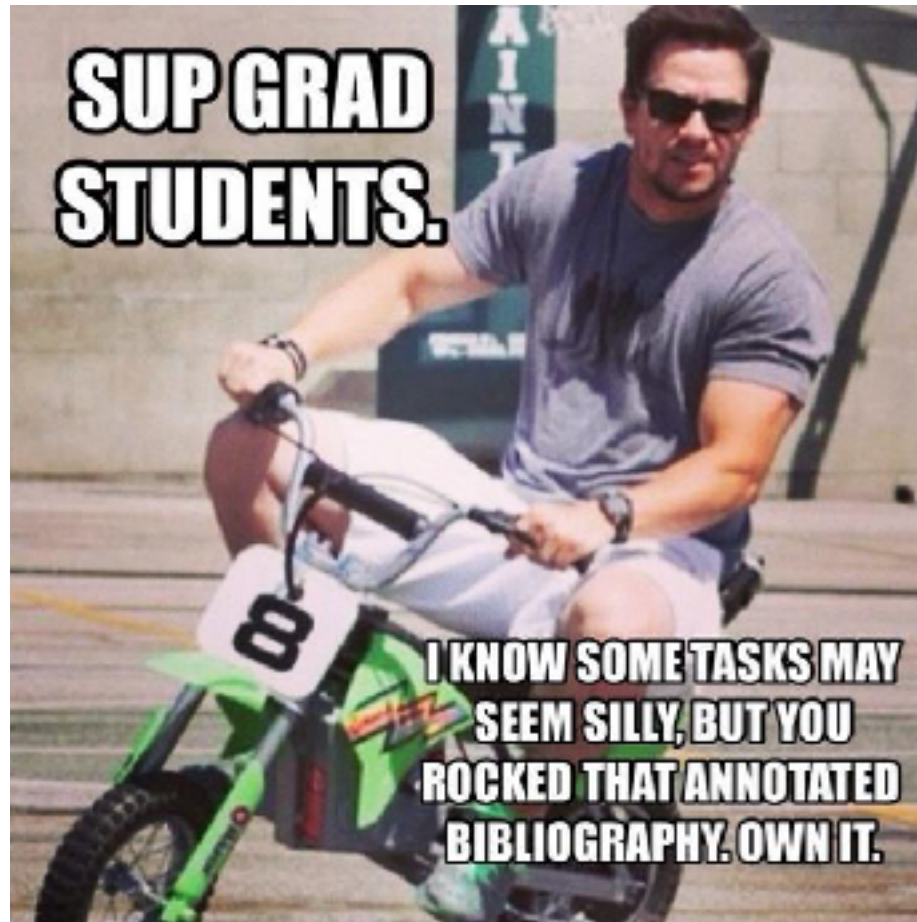
Lecture Notes for Machine Learning in Python

Professor Eric Larson
Optimization Techniques for Logistic Regression

Class Logistics and Agenda

- Agenda
 - Numerical Optimization Techniques
 - Types of Optimization
 - Programming the Optimization
- **Whirlwind Lecture Alert**
 - Entire classes cover these concepts
 - But you are smart enough to get them in one lecture!
 - Because science!

Gradient Descent Techniques

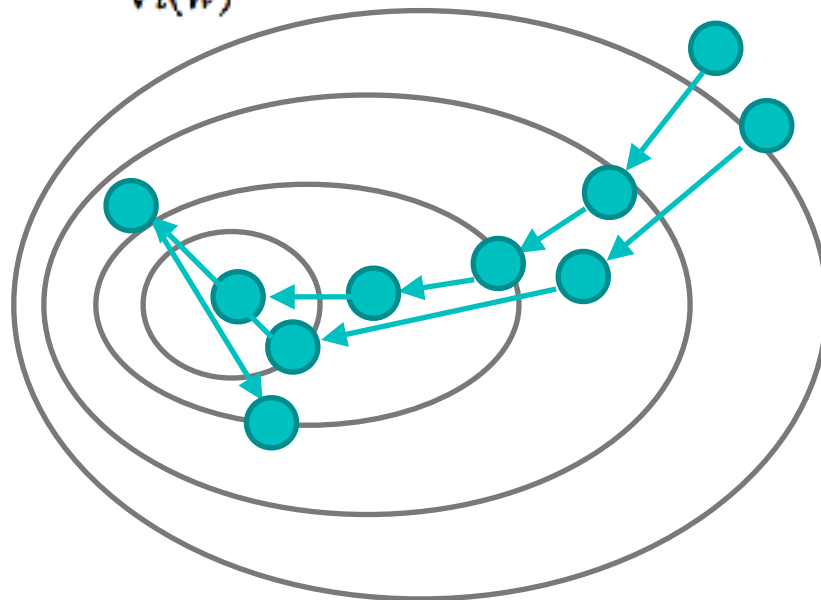


Optimization: gradient descent

- What we know thus far:

$$\underbrace{w_j}_{\text{new value}} \leftarrow \underbrace{w_j}_{\text{old value}} + \underbrace{\eta \left[\left(\sum_{i=1}^M (y^{(i)} - g(x^{(i)})) x_j^{(i)} \right) - C \cdot 2w_j \right]}_{\nabla l(w)}$$

$$w \leftarrow w + \eta \nabla l(w)$$



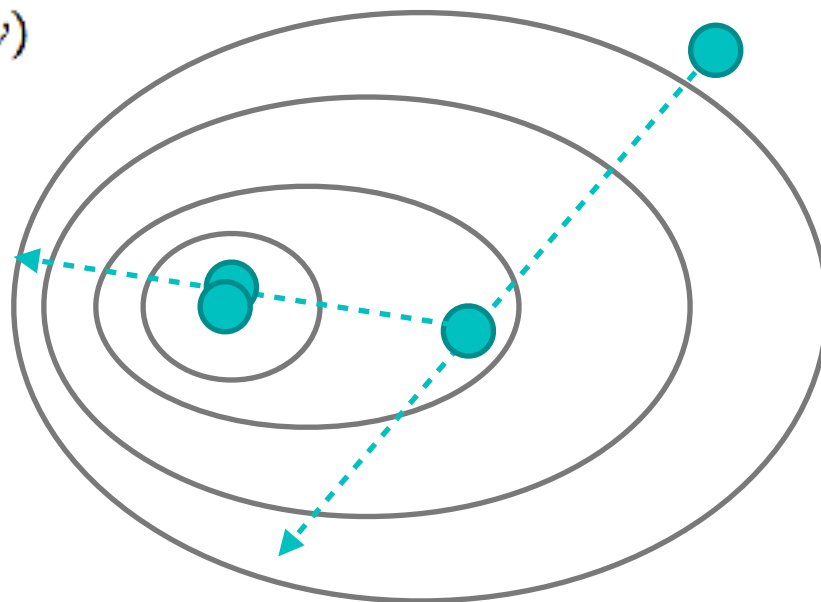
Line Search: a better method

- Line search in direction of gradient:

$$\eta \leftarrow \arg \max_{\eta} \underbrace{\sum_{i=1}^M (y^{(i)} - \hat{y}^{(i)})^2 - C \cdot \sum_j w_j^2}_{l(w)}$$

$$w \leftarrow w + \eta \nabla l(w)$$

$$w \leftarrow w + \underbrace{\eta}_{\text{best step?}} \nabla l(w)$$



Stochastic Methods

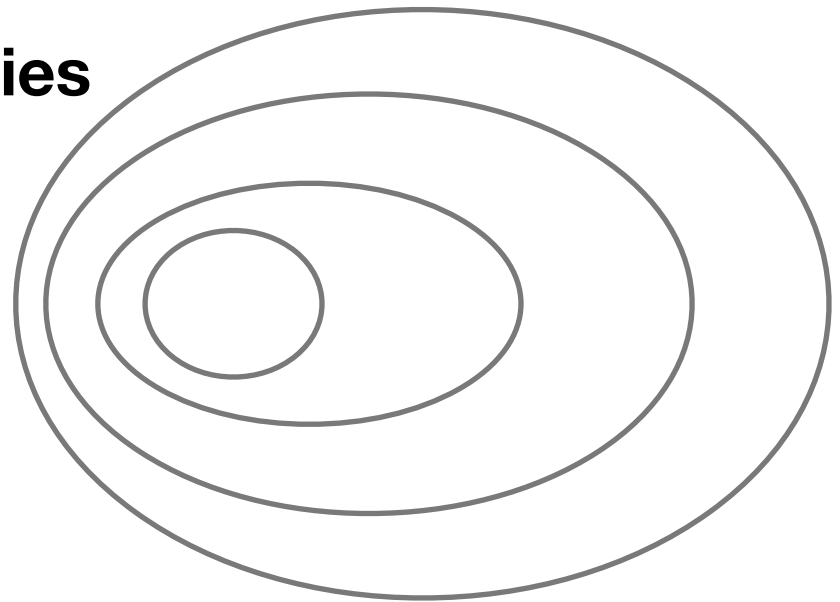
- How much computation is required (for gradient)?

$$\sum_{i=1}^M (y^{(i)} - \hat{y}^{(i)}) x^{(i)} - 2C \cdot w$$

M = number of instances
N = number of features

Self Test: How many multiplies per gradient calculation?

- A. M+N multiplications
- B. M*N multiplications
- C. 2N multiplications
- D. 2N-M multiplications



Stochastic Methods

- How much computation is required (for gradient)?

$$\sum_{i=1}^M (y^{(i)} - \hat{y}^{(i)})x^{(i)} - 2C \cdot w$$

Per iteration:

M*N multiplications

2M add/subtract

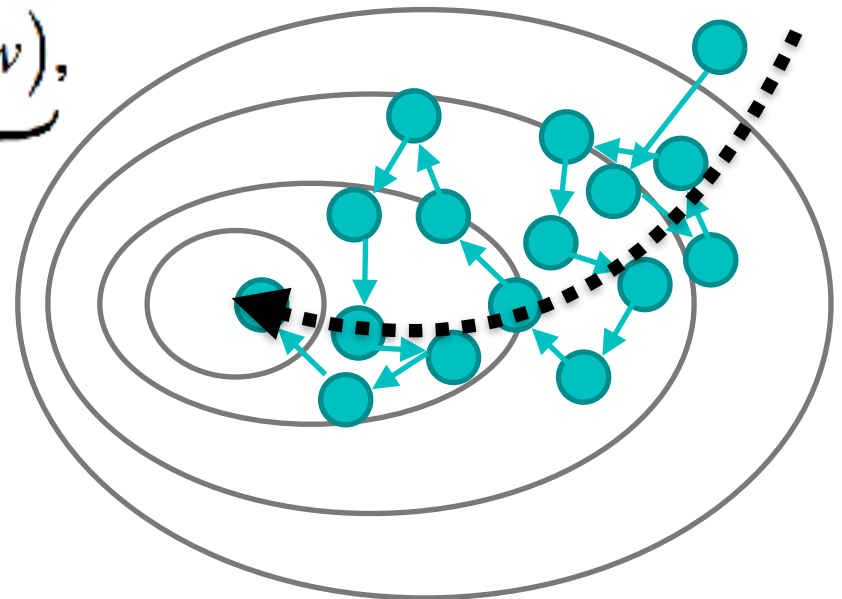
$$w \leftarrow w + \underbrace{\eta \left((y^{(i)} - \hat{y}^{(i)})x^{(i)} - 2C \cdot w \right)}_{\text{approx. gradient}},$$

i chosen at random

Per iteration:

N multiplications

1 add/subtract



Numerical Optimization

Gradient Descent (with line search)

Stochastic Gradient Descent

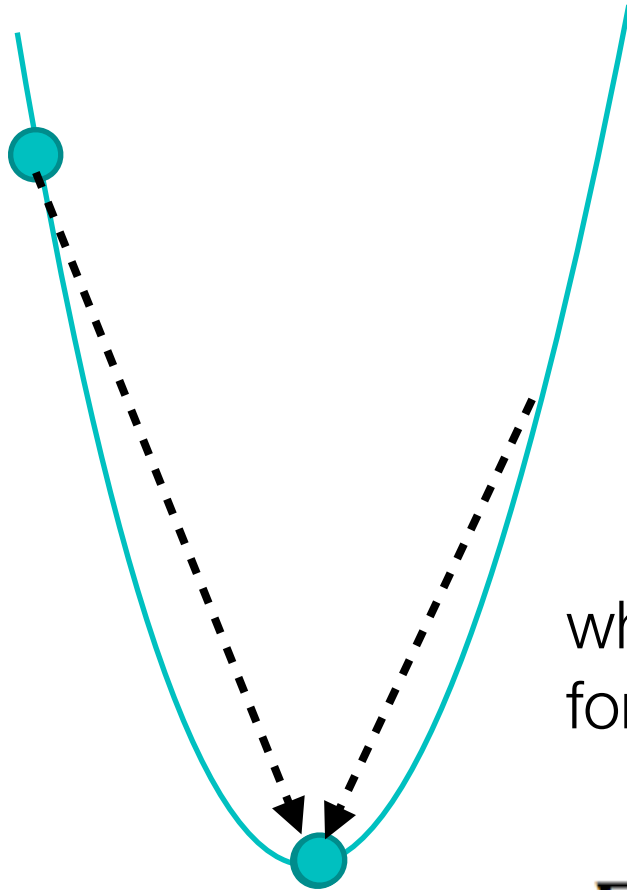


Optimization Techniques with the Hessian



The Hessian

- Assume function is quadratic:



function of one variable:

$$w \leftarrow w - \underbrace{\left[\frac{\partial^2}{\partial w^2} l(w) \right]^{-1}}_{\text{inverse 2nd deriv}} \underbrace{\frac{\partial}{\partial w} l(w)}_{\text{derivative}}$$

will solve in one step!

what is the second order derivative
for a multivariate function?

$$\nabla^2 l(w) = \mathbf{H}[l(w)]$$

The Hessian

- Assume function is quadratic:

function of one variable:

$$\mathbf{H}[l(w)] = \begin{bmatrix} \frac{\partial^2}{\partial w_1} l(w) & \frac{\partial}{\partial w_1} \frac{\partial}{\partial w_2} l(w) & \dots & \frac{\partial}{\partial w_1} \frac{\partial}{\partial w_N} l(w) \\ \frac{\partial}{\partial w_2} \frac{\partial}{\partial w_1} l(w) & \frac{\partial^2}{\partial w_2} l(w) & \dots & \frac{\partial}{\partial w_2} \frac{\partial}{\partial w_N} l(w) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial w_N} \frac{\partial}{\partial w_1} l(w) & \frac{\partial}{\partial w_N} \frac{\partial}{\partial w_2} l(w) & \dots & \frac{\partial^2}{\partial w_N} l(w) \end{bmatrix}$$



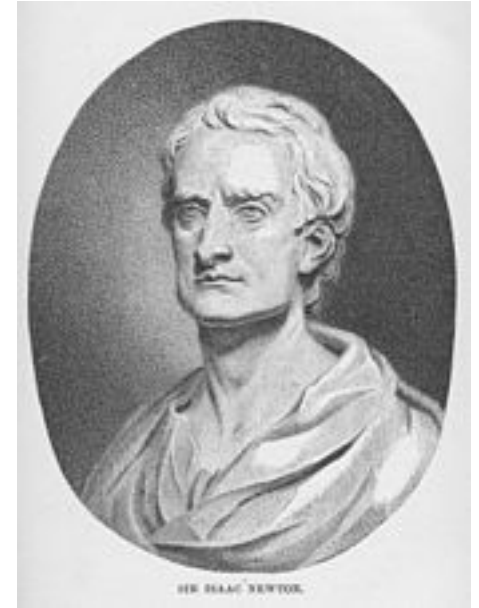
$$\nabla^2 l(w) = \mathbf{H}[l(w)]$$

The Newton Update Method

- Assume function is quadratic (in high dimensions):

$$w \leftarrow w - \underbrace{\left[\frac{\partial^2}{\partial w} l(w) \right]^{-1}}_{\text{inverse 2nd deriv}} \underbrace{\frac{\partial}{\partial w} l(w)}_{\text{derivative}}$$

$$w \leftarrow w + \eta \cdot \underbrace{\mathbf{H}[l(w)]^{-1}}_{\text{inverse Hessian}} \cdot \underbrace{\nabla l(w)}_{\text{gradient}}$$



Is. Newton

I do not know what I may appear to the world, but to myself I seem to have been only like a boy playing on the sea-shore, and diverting myself in now and then finding a smoother pebble or a prettier shell than ordinary, whilst the great ocean of truth lay all undiscovered before me.

$$\frac{\partial}{\partial w_j} l(w) = \sum_i (y^{(i)} - g(x^{(i)})) x_j^{(i)}$$

↓ PLUG IN

$$H[k,j] = \frac{\partial}{\partial w_k} \frac{\partial}{\partial w_j} l = \frac{\partial}{\partial w_k} \left(\sum_i (y^{(i)} - g(x^{(i)})) x_j^{(i)} \right)$$

$$= \sum_i \frac{\partial}{\partial w_k} y^{(i)} x_j^{(i)} - \frac{\partial}{\partial w_k} g(x^{(i)}) x_j^{(i)}$$

← LEFT OVER TERM

$$= - \sum_i \frac{\partial}{\partial w_k} g(x^{(i)}) x_j^{(i)}$$

Already know $\frac{\partial}{\partial w_k} g(w^T x^{(i)})$ side calculation

$$= g(x^{(i)}) (1 - g(x^{(i)})) \frac{\partial}{\partial w_k} (w^T x^{(i)})$$

$$= g(x^{(i)}) (1 - g(x^{(i)})) x_k^{(i)}$$

← PLUG IN

$$\therefore = - \sum_i g(x^{(i)}) (1 - g(x^{(i)})) x_k^{(i)} x_j^{(i)}$$

← THAT'S THE HESSIAN!

$H(k,j) =$ This is a valid equation for the Hessian, but we want to represent it using linear algebra

$$= - \sum_i \underset{\substack{\uparrow \\ \text{SIGMOID}}}{g(x^{(i)})} (1 - g(x^{(i)})) x_k^{(i)} x_j^{(i)}$$

SEE LINEAR ALG

The Hessian for Logistic Regression

- The hessian is easy to calculate from the gradient for logistic regression

$$w \leftarrow w + \eta \cdot \underbrace{\mathbf{H}[l(w)]^{-1}}_{\text{inverse Hessian}} \cdot \underbrace{\nabla l(w)}_{\text{gradient}}$$

$$\mathbf{H}_{j,k}[l(w)] = - \sum_{i=1}^M g(x^{(i)})(1 - g(x^{(i)})) x_k^{(i)} x_j^{(i)}$$

$$\mathbf{H}[l(w)] = X^T \cdot \text{diag}[g(x^{(i)})(1 - g(x^{(i)}))] \cdot X$$

$$w \leftarrow w + \eta [X^T \cdot \text{diag}[g(x^{(i)})(1 - g(x^{(i)}))] \cdot X]^{-1} \cdot X * y_{diff}$$

$$\sum_{i=1}^M (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$$

$$X * y_{diff}$$

Numerical Optimization

Newton's method



Problems with Newton's Method

- **Quadratic** isn't always a great assumption:
 - highly dependent on starting point
 - jumps can get **really random!**
 - near saddle points, inverse hessian **unstable**
 - hessian **not** always **invertible**...
 - or invertible with correct numerical precision

The solution: quasi Newton methods

- In general:
 - **approximate** the **Hessian** with something numerically sound and efficiently invertible
 - **back off to gradient** descent when the approximate hessian is **not stable**
 - use **momentum** to update approximate hessian
- **A popular approach:** use Broyden-Fletcher-Goldfarb-Shanno (BFGS)
 - which you can look up if you are interested ...

https://en.wikipedia.org/wiki/Broyden-Fletcher-Goldfarb-Shanno_algorithm

BFGS (if time)

$\mathbf{H}_0 = \mathbf{I}$	init
$p_k = -\mathbf{H}_k^{-1} \nabla l(w_k)$	get update direction
$w_{k+1} \leftarrow w_k + \eta \cdot p_k$	find next w
$s_k = \eta \cdot p_k$	get scaled direction
$v_k = \nabla l(w_{k+1}) - \nabla l(w_k)$	approx gradient change
$\mathbf{H}_{k+1} = \mathbf{H}_k + \frac{v_k v_k^T}{v_k^T s_k} - \frac{\mathbf{H}_k s_k s_k^T \mathbf{H}_k}{s_k^T \mathbf{H}_k s_k}$	update Hessian and inverse Hessian approx
$\mathbf{H}_{k+1}^{-1} = \mathbf{H}_k^{-1} + \frac{(s_k^T v_k + \mathbf{H}_k^{-1})(s_k s_k^T)}{(s_k^T v_k)^2} - \frac{\mathbf{H}_k^{-1} v_k s_k^T + s_k v_k^T \mathbf{H}_k^{-1}}{s_k^T v_k}$	
$k = k + 1$	increment k and repeat

invertibility of H well defined / only matrix operations

Numerical Optimization

BFGS (if time)
parallelization



For Next Lecture

- **Next time:** SVMs via in class assignment
- **Next Next time:** Neural Networks

Scratch Paper

Scratch Paper

Scratch Paper