Lecture Notes for

Machine Learning in Python

Professor Eric Larson

Logistic Regression

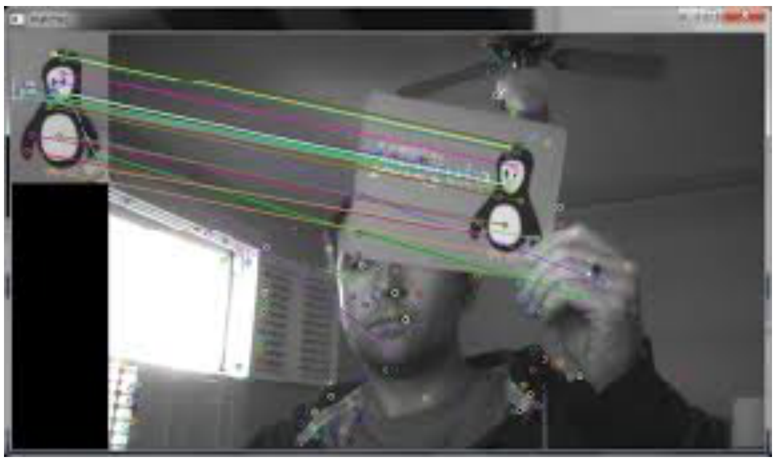
Class Logistics and Agenda

- Logistics
  - A2 Due soon!
- Agenda
  - Town Hall
  - Logistic Regression
    - Solving
    - Programming
    - Finally some real python!

Town Hall

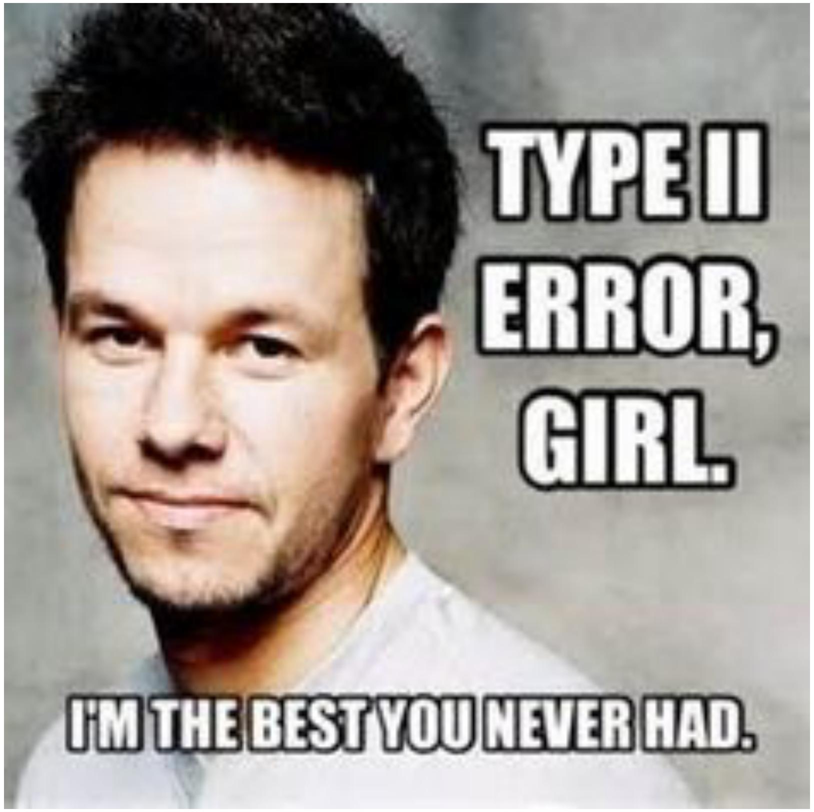
Matching versus Bag of Features

- Not a difference of vectors, but a percentage of matching points



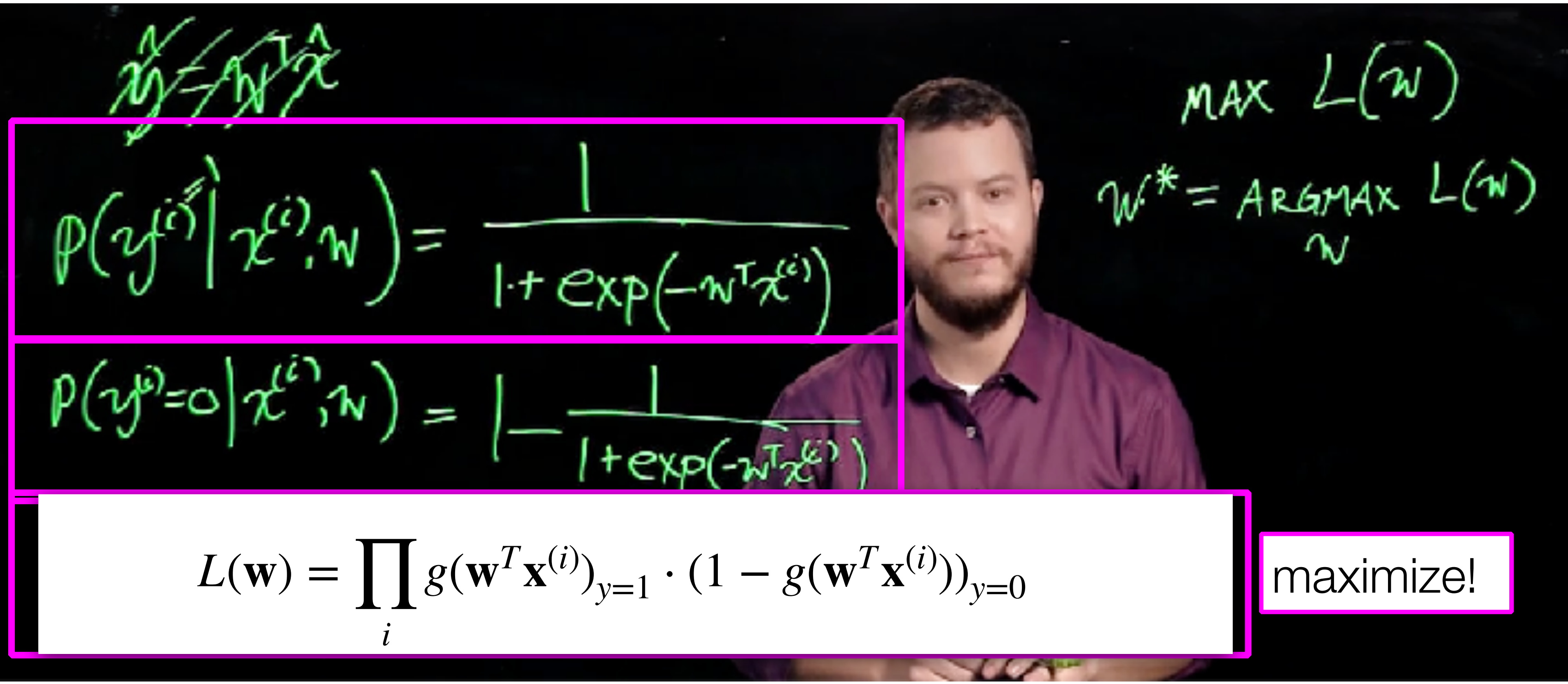
- SURF, ORB, SIFT, DAISY

Solving Logistic Regression



Setting Up Binary Logistic Regression

- From flipped lecture:



$$L(\mathbf{w}) = \prod_i g(\mathbf{w}^T \mathbf{x}^{(i)})_{y=1} \cdot (1 - g(\mathbf{w}^T \mathbf{x}^{(i)}))_{y=0}$$

maximize!

where $g(.)$ is a sigmoid

How do you optimize iteratively?

- **Objective Function**: the function we want to minimize or maximize
- **Update Formula**: what update “step”can we take to optimize the objective function
- **Parameters**: what are the parameters of the model that we can change to optimize the objective function

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)

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

    - Take Gradient

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

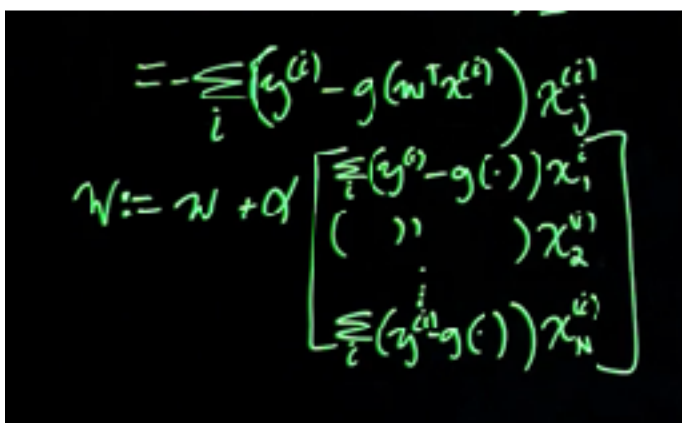
    - Use gradient inside update equation for **w**

Binary Solution for Update Equation

- Use gradient inside update equation for **w**

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

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



Demo

`05. Logistic Regression.ipynb`

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**: More gradient based optimization techniques for logistic regression

Lecture Notes for

Machine Learning in Python

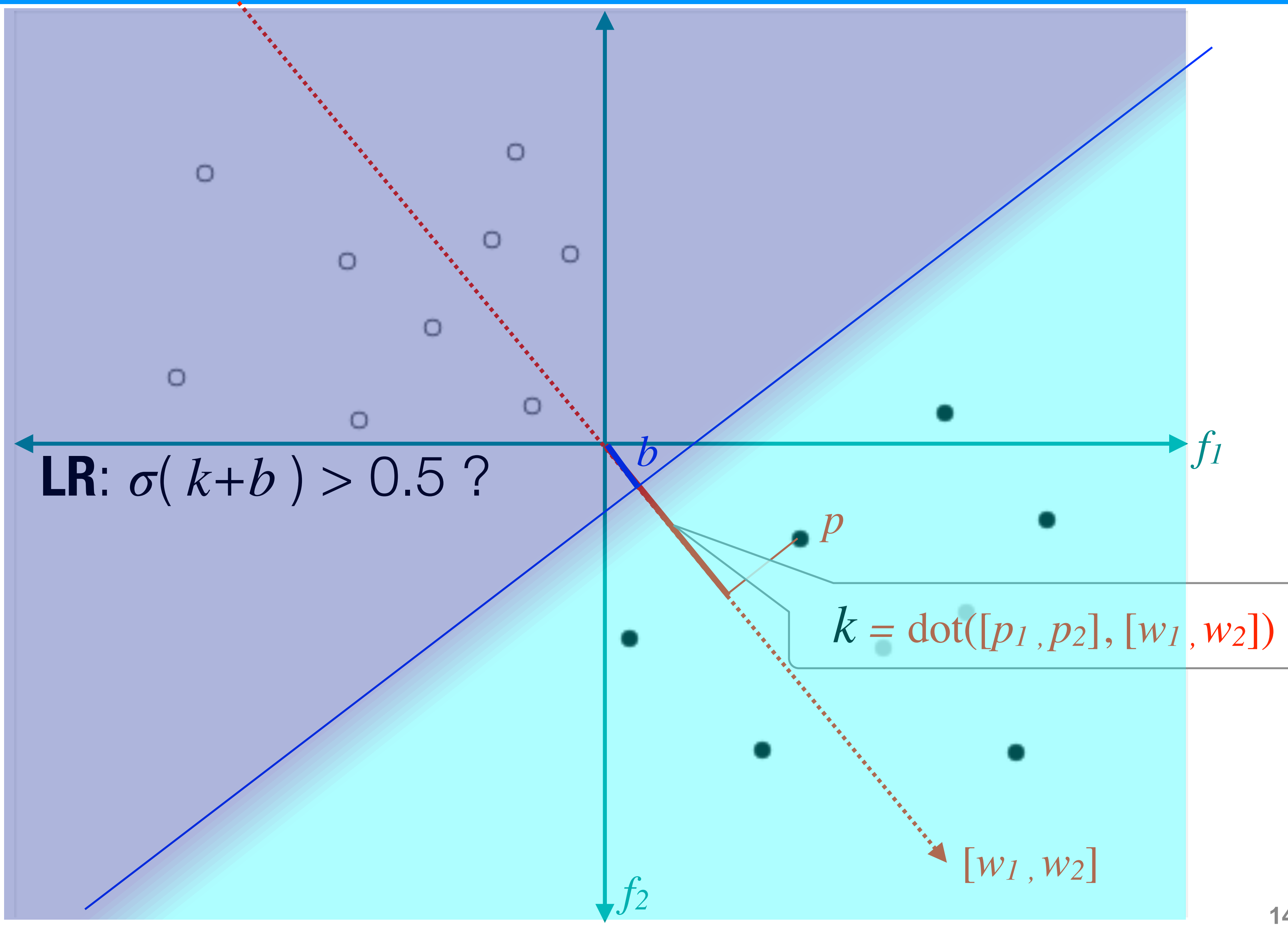
Professor Eric Larson

Optimization Techniques for Logistic Regression

Class Logistics and Agenda

- Agenda
  - Finish Logistic Regression
  - More Numerical Optimization Techniques
    - Types of Optimization
    - Programming the Optimization
- **Whirlwind Lecture Alert**
  - Get an intuition, program it, maybe you don't follow every mathematical concept in lecture
  - But you know how to approach it outside lecture

What do weights and intercept define?



Demo

`05. Logistic Regression.ipynb`

"Finish"
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

Demo Lecture

06. Optimization

Scratch Paper

Back Up Slides

Last time

$$p(y^{(i)} = 1 \mid x^{(i)}, w) = \frac{1}{1 + \exp(w^T x^{(i)})}$$

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

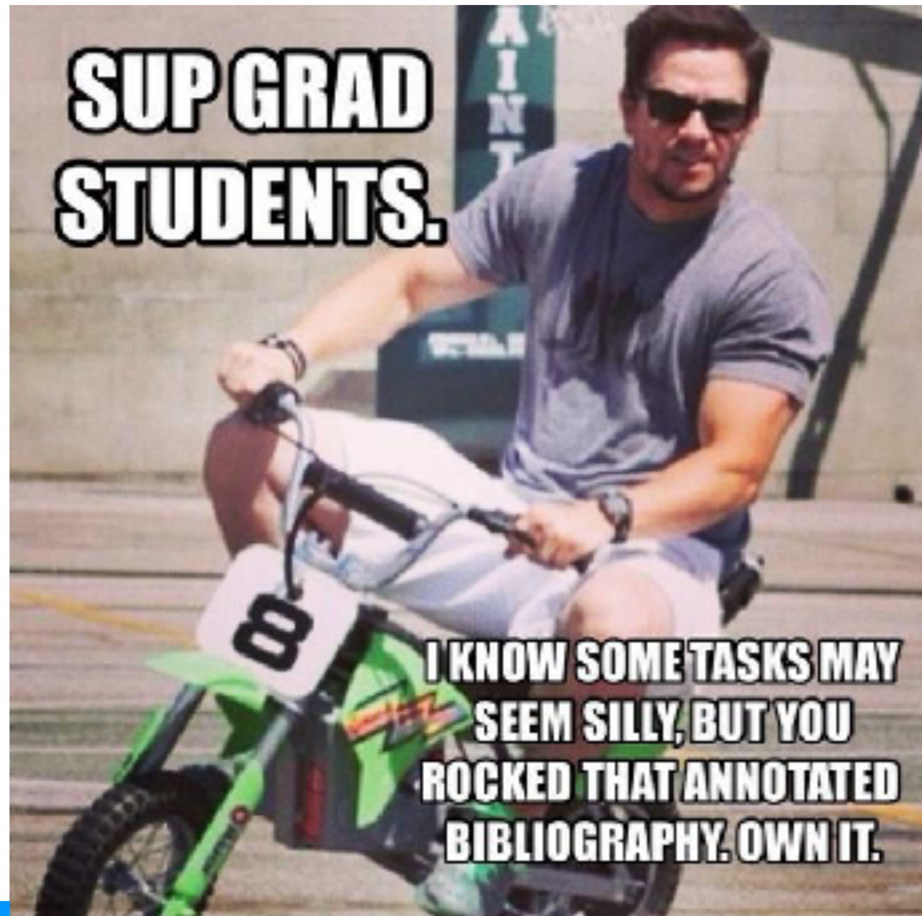
$$\underbrace{w_j}_{\text{new value}} \leftarrow \underbrace{w_j}_{\text{old value}} + \underbrace{\eta \sum_{i=1}^{M} (y^{(i)} - g(x^{(i)}))x_j^{(i)}}_{\text{gradient}}$$

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

$$w \leftarrow w + \eta \left[\underbrace{\nabla l(w)_{old}}_{\text{old gradient}} - C \cdot 2w \right]$$

```
def _get_gradient(self,X,y):
    # programming \sum_i (yi-g(xi))xi
    gradient = np.zeros(self.w_.shape) # set
    for (xi,yi) in zip(X,y):
        # the actual update inside of sum
        gradi = (yi - self.predict_proba(xi,
        # reshape to be column vector and ad
        gradient += gradi.reshape(self.w_.sh

    return gradient/float(len(y))
```

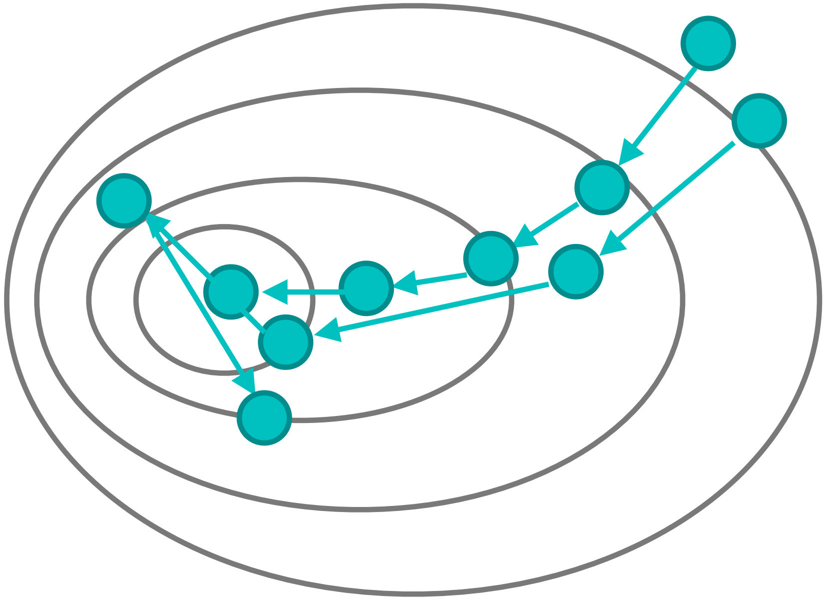

SUP GRAD STUDENTS.
I KNOW SOME TASKS MAY SEEM SILLY, BUT YOU ROCKED THAT ANNOTATED BIBLIOGRAPHY. OWN IT.

Optimization: gradient descent

- What we know thus far:

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

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



Line Search: a better method

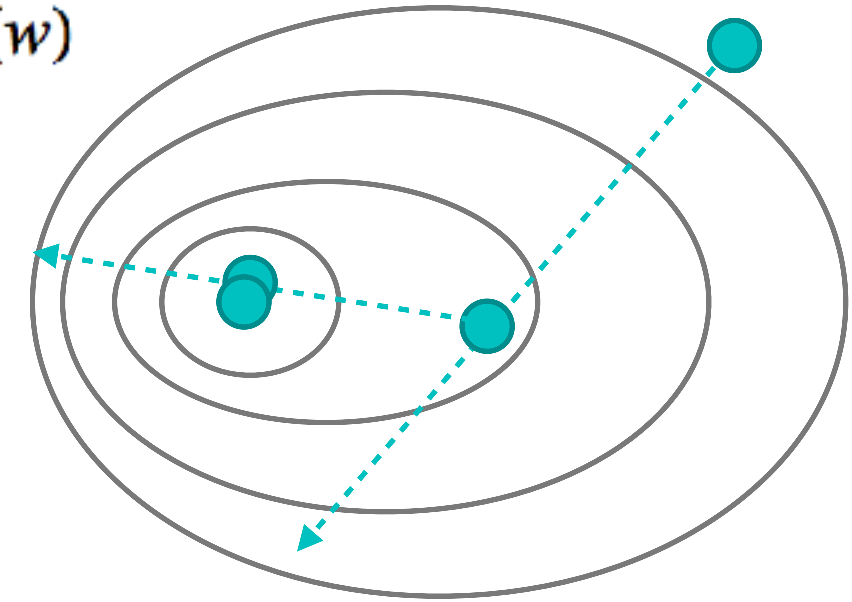
- Line search in direction of gradient:

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

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

fixed

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



Revisiting the Gradient

- 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+1 multiplications
B. (M+1)*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+1)*N multiplications

2M add/subtract

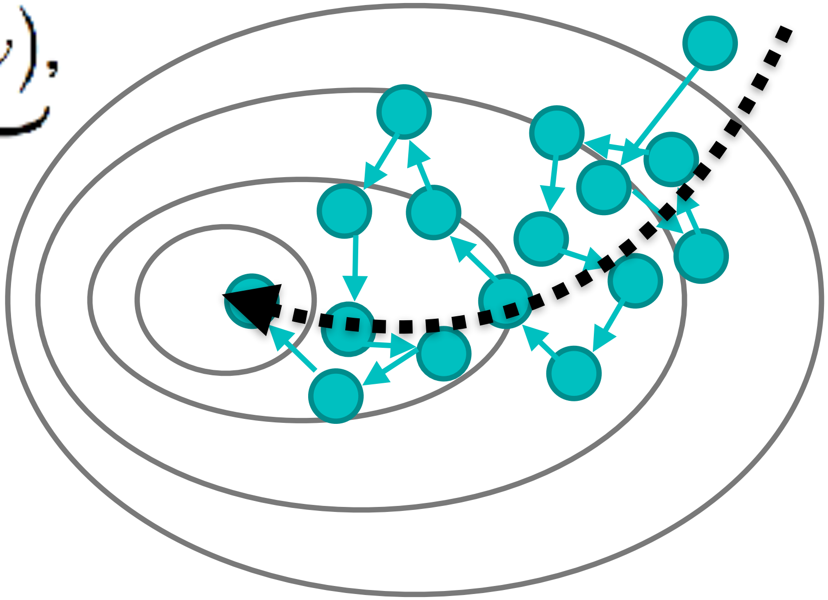
$$w \leftarrow w + \eta\underbrace{\left((y^{(i)} - \hat{y}^{(i)})x^{(i)} - 2C \cdot w\right)}_{\text{approx. gradient}},$$

i chosen at random

Per iteration:

N+1 multiplications

1 add/subtract



06. Optimization.ipynb **Demo**

Gradient Descent (with line search)
Stochastic Gradient Descent
Hessian
Quasi-Newton Methods
Multi-processing

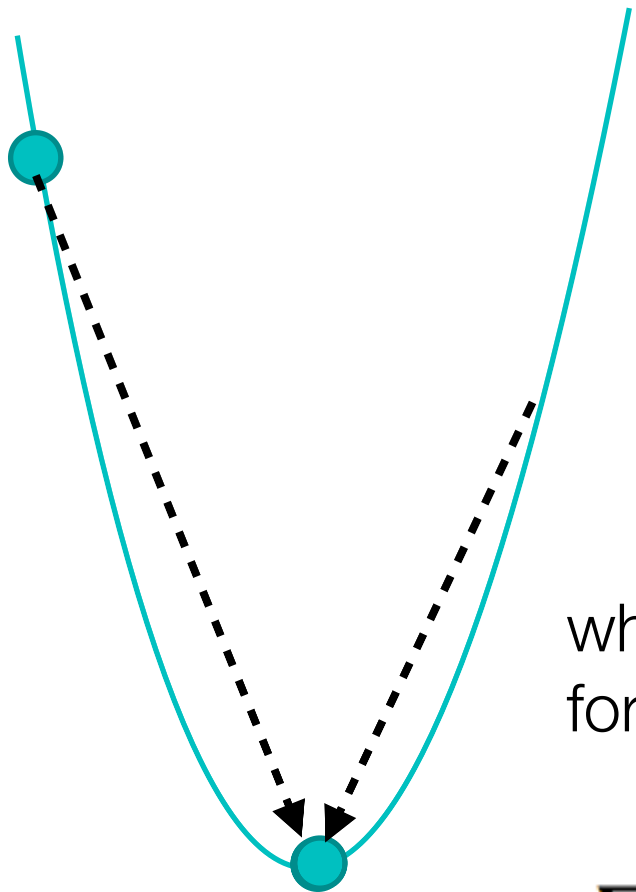
For Next Lecture

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


SUP. HAVE YOU HEARD THE LATEST STATS JOKE?
PROBABLY.

Can we do better than the gradient?

- Assume function is quadratic:



function of one variable:

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

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:

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

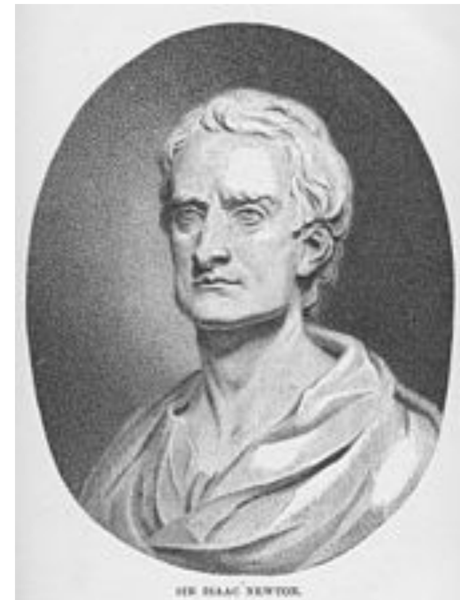
$$\nabla^2 l(w) = \mathbf{H}[l(w)]$$

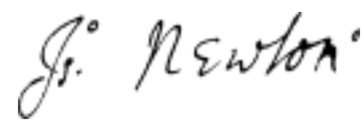
The Newton Update Method

- Assume function is quadratic (in high dimensions):

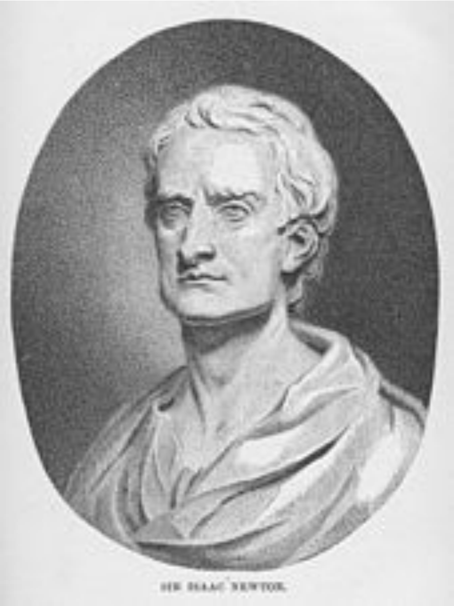
$$w \leftarrow w - \underbrace{[\frac{\partial^2}{\partial w} l(w)]^{-1}}_{\text{inverse 2nd deriv}} \underbrace{\frac{\partial}{\partial w} l(w)}_{\text{derivative}}$$

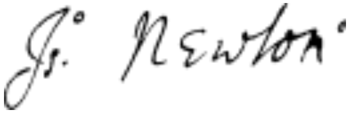
$$w \leftarrow w + \eta \cdot \underbrace{\mathbf{H}[l(w)]^{-1}}_{\text{inverse Hessian}} \cdot \underbrace{\nabla l(w)}_{\text{gradient}}$$





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.





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.

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

PLUG IN

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

$$= \sum_i \frac{\partial}{\partial w_k} \overset{\to 0}{y^{(i)}} x_j^{(i)} - \frac{\partial}{\partial w_k} g(x^{(i)}) x_j^{(i)}$$

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

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

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

PLUG IN

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

THAT'S THE HESSIAN!

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

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

... LINEAR ALG...

The Hessian for Logistic Regression

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

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

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

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

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

$$X * y_{diff}$$

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

Demo

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

$$\mathbf{H}_0 = \mathbf{I}$$ init

$$p_k = -\mathbf{H}_k^{-1} \nabla l(w_k)$$ get update direction

find next w $$w_{k+1} \leftarrow w_k + \eta \cdot p_k$$

get scaled direction $$s_k = \eta \cdot p_k$$

$$v_k = \nabla l(w_{k+1}) - \nabla l(w_k)$$ approx gradient change

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \underbrace{\frac{v_k v_k^T}{v_k^T s_k}}_{\text{approx. Hessian}} - \underbrace{\frac{\mathbf{H}_k s_k s_k^T \mathbf{H}_k}{s_k^T \mathbf{H}_k s_k}}_{\text{momentum}}$$

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

Demo

BFGS (if time)
parallelization