Do not distribute course material

You may not and may not allow others to reproduce or distribute lecture notes and course materials publicly whether or not a fee is charged.

Topic 1 continued

How would we modify our algorithm if we had more features? ($d \geq 1$)

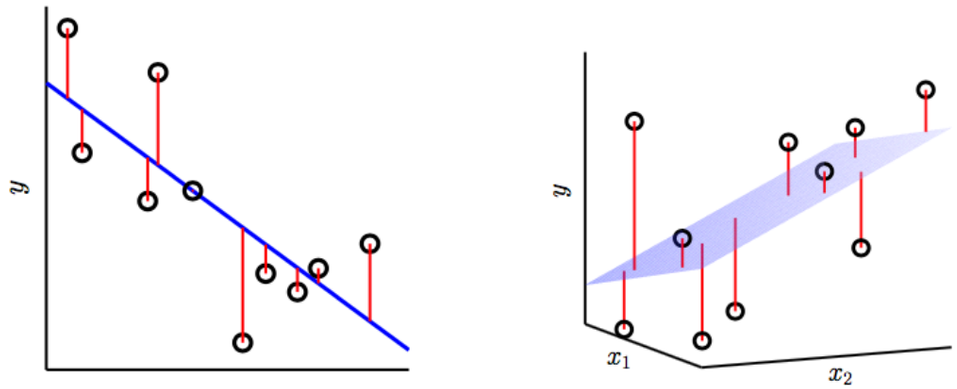
Outline

- Motivating Example: Predicting the mpg of a car
- Model: Linear Model
- Objective function: Least Squares Fit Problem
- Local Optimizer: “Batch” Gradient Descent
  - When $d = 1$ (simple linear regression)
  - What can go wrong?
  - General case, any $d > 0$ (multiple variable linear regression)
    - Example
    - Algorithm
    - Vectorized Algorithm
- Global Optimization: Normal Equations
- Feature transformation
- Assessing Goodness of Fit
- Objective function revisited: Probabilistic interpretation
- Extensions
- Removing features

“Batch” means we use all the training examples when computing the gradient

Least mean squares when $\mathbf{x}$ is d-dimensional

Finding the best line/hyperplane with the smallest squared residuals



$$y = f(\mathbf{x}) + \epsilon \qquad \longleftarrow \text{ noisy target } P(y|\mathbf{x})$$

$$y^{(i)} \approx \hat{y}^{(i)} = w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_d x_d^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)}$$

$$RSS(\mathbf{w}) = \sum_{i=1}^{N} (y^{(i)} - \hat{y}^{(i)})^2 = \sum_{i=1}^{N} (y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)})^2$$

We imagine that

$y = w_0 + w_1 x_1 + w_2 x_2 + \epsilon$

The pictures are from the Malik Magdon-Ismail and Hsuan-Tien Lin's lecture slides

Updated loss function

General ML problem

- Get data
- Pick a model with parameters
- Pick a loss function
  - Measures goodness of fit model to data
  - Function of the parameters
- Find parameters that minimizes loss

Linear regression

Data: $(\mathbf{x}^{(i)}, y^{(i)}), i = 1,2,\ldots,N$

Linear model:

$$\hat{y}^{(j)} = w_0 + w_1 x_1^{(j)} + w_2 x_2^{(j)} + \cdots + w_d x_d^{(j)}$$

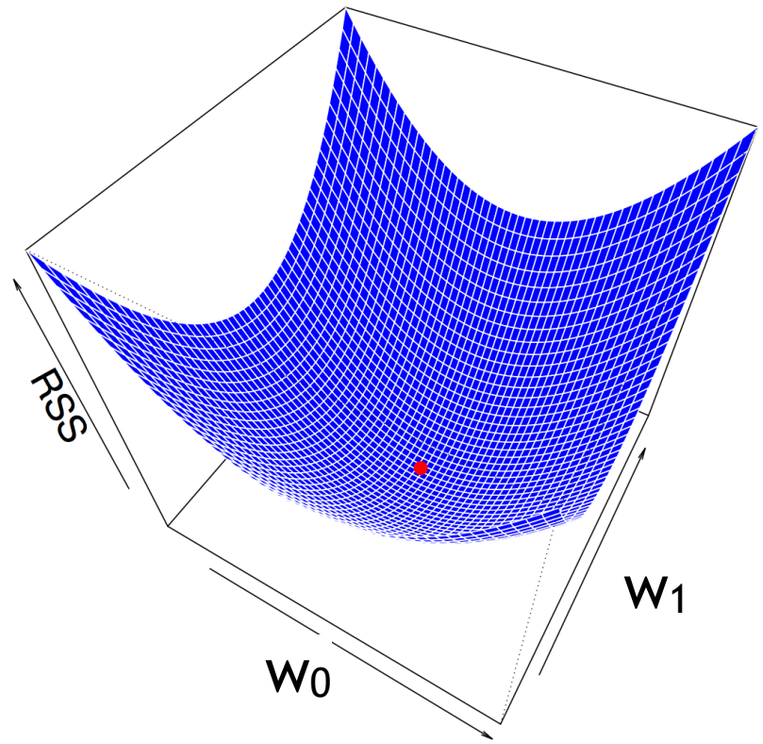
Loss function: $\frac{\text{RSS}(\mathbf{w})}{2N} = \frac{1}{2N}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2$

$$= \frac{1}{2N}\left[(y^{(1)} - \hat{y}^{(1)})^2 + (y^{(2)} - \hat{y}^{(2)})^2 + \cdots + (y^{(N)} - \hat{y}^{(N)})^2\right]$$

Select $\mathbf{w} = [w_0, w_1, w_2, \ldots, w_d]^T$ to minimize $\frac{\text{RSS}(\mathbf{w})}{2N}$ This $\mathbf{w}$ also minimizes $\text{RSS}(\mathbf{w})$

Gradient

$$J(\mathbf{w}) = \frac{1}{2N}\text{RSS}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})^2$$



From calculus we know that the direction for the maximum rate of change for a function J(w) is

$$\nabla J(\mathbf{w}) = \begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_0} \\ \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial w_d} \end{bmatrix}$$

Each coefficient (parameter/weight) should be updated as follows:

$$w_0 - \alpha\frac{\partial J(\mathbf{w})}{\partial w_0}$$

$$w_1 - \alpha\frac{\partial J(\mathbf{w})}{\partial w_1}$$

$$\vdots$$

$$w_d - \alpha\frac{\partial J(\mathbf{w})}{\partial w_d}$$

"Some of the figures in this presentation are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani "

The partial derivative of J(w)

$$\mathrm{J}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}((\mathbf{w}^T x^{(i)}) - y^{(i)})^2 \;= \frac{1}{2N}\sum_{i=1}^{N}(w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_j x_j^{(i)} + \cdots w_d x_d^{(i)} - y^{(i)})^2$$

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \frac{1}{N}\cdot\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})x_j^{(i)}$$

Verify at home 😊

$$= \frac{1}{N}\sum_{i=1}^{N}(w_0 x_0^{(i)} + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_j x_j^{(i)} + \cdots + w_d x_d^{(i)} - y^{(i)})x_j^{(i)}$$

Slide not presented in class

$$\mathrm{J}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}((\mathbf{w}^T x^{(i)}) - y^{(i)})^2 = \frac{1}{2N}\sum_{i=1}^{N}(w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_j x_j^{(i)} + \cdots w_d x_d^{(i)} - y^{(i)})^2$$

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \frac{1}{2N}\cdot\frac{\partial}{\partial w_j}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})^2 = \frac{1}{2N}\cdot\sum_{i=1}^{N}\frac{\partial}{\partial w_j}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})^2$$

$$= \frac{1}{2N}\cdot\sum_{i=1}^{N}2(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})\frac{\partial}{\partial w_j}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)}) \qquad = \frac{1}{N}\cdot\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})x_j^{(i)}$$

$$= \frac{1}{N}\sum_{i=1}^{N}(w_0 x_0^{(i)} + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_j x_j^{(i)} + \cdots + w_d x_d^{(i)} - y^{(i)})x_j^{(i)}$$

Cost function

- Minimizing $RSS(\mathbf{w}) = \sum_{i=1}^{N}((\mathbf{w}^T x^{(i)}) - y^{(i)})^2$ (our cost function for linear regression) is the same as minimizing:

$$\mathrm{J}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}((\mathbf{w}^T x^{(i)}) - y^{(i)})^2 = \frac{1}{2N}\mathrm{np.sum}(\mathrm{np.square}(X\mathbf{w} - y)) = \frac{1}{2N}RSS(\mathbf{w})$$

- Both RSS and J are convex function (as was x^2), so we can use the gradient to find the minimum by taking a sequence of steps. Here is the derivative for the J function wrt the parameters:

$$\frac{\partial J(\mathbf{w})}{\partial w_0} = \frac{1}{N}\sum_{i=1}^{N}(w_0 x_0^{(i)} + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_j x_j^{(i)} + \cdots w_d x_d^{(i)} - y^{(i)})x_0^{(i)} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}^T x^{(i)} - y^{(i)})x_0^{(i)}$$

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}(w_0 x_0^{(i)} + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_j x_j^{(i)} + \cdots + w_d x_d^{(i)} - y^{(i)})x_j^{(i)} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}^T x^{(i)} - y^{(i)})x_j^{(i)}$$

individual slope

average slope

updating by α·(averaging the slope over all training examples)

$x_0^{(i)} = 1$

By scanned by jd, photography by "frères Bisson" (nationale de France (BNF)) [Public domain], via Wikimedia Commons

Batch Gradient Descent

- Loss/cost function (objective) $\frac{1}{2N}\text{RSS}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})^2$
- The gradient is: $\frac{\partial J(\mathbf{w})}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})x_j^{(i)}$
- To decrease the cost, we update the parameters, $\mathbf{w} = [w_0, w_1, \ldots, w_d]^T$, using the update rule: $w_j - \alpha\frac{\partial J(\mathbf{w})}{\partial w_j}$

$x_0^{(i)} = 1$

for $i = 1$ to num_iter

$$temp0 = w_0 - \alpha\frac{\partial J(\mathbf{w})}{\partial w_0} = w_0 - \frac{\alpha}{N}\sum_{i=1}^{N}(w^T\mathbf{x}^{(i)} - y^{(i)})x_0^{(i)}$$

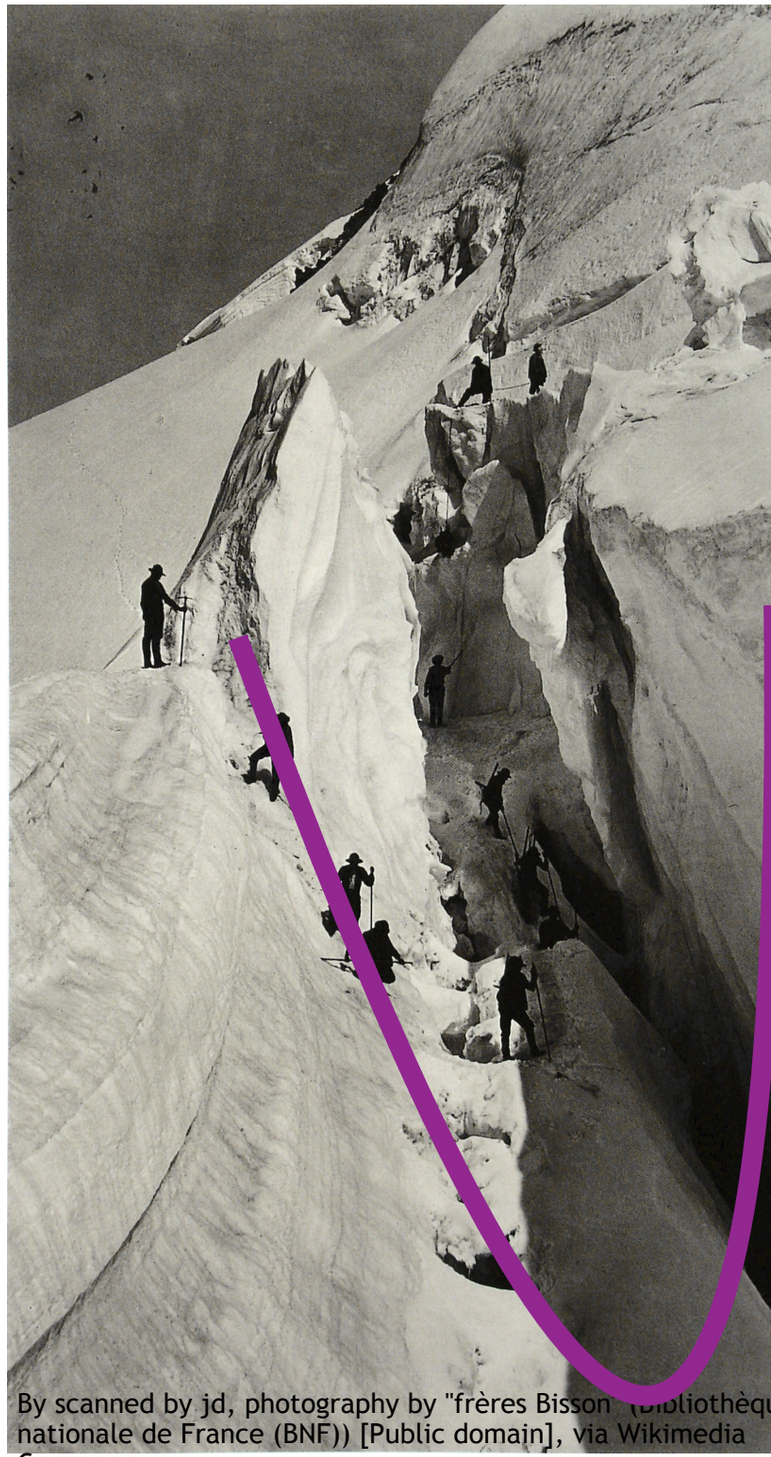
$$temp1 = w_1 - \alpha\frac{\partial J(\mathbf{w})}{\partial w_1} = w_1 - \frac{\alpha}{N}\sum_{i=1}^{N}(w^T\mathbf{x}^{(i)} - y^{(i)})x_1^{(i)}$$

$$\vdots$$

$$tempd = w_d - \alpha\frac{\partial J(\mathbf{w})}{\partial w_d} = w_d - \frac{\alpha}{N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})x_d^{(i)}$$

$$w_0 = temp0$$
$$w_1 = temp1$$
$$\vdots$$
$$w_d = tempd$$

Simultaneous update



By scanned by jd, photography by "frères Bisson" (Bibliothèque nationale de France (BNF)) [Public domain], via Wikimedia Commons

Outline

- Motivating Example: Predicting the mpg of a car
- Model: Linear Model
- Objective function: Least Squares Fit Problem
- Local Optimizer: “Batch” Gradient Descent
  - When $d = 1$ (simple linear regression)
  - What can go wrong?
  - General case, any $d > 0$ (multiple variable linear regression)
    - Example
    - Algorithm
    - Vectorized Algorithm
- Global Optimization: Normal Equations
- Feature transformation
- Assessing Goodness of Fit
- Objective function revisited: Probabilistic interpretation
- Extensions
- Removing features

“Batch” means we use all the training examples when computing the gradient

Vectorized Implementation

$$J(\mathbf{w}) = \frac{1}{2N}\text{RSS}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})^2$$

Small example to show $\nabla J(\mathbf{w}) = \frac{1}{N}X^T(X\mathbf{w} - \mathbf{y})$ if we have one feature: $\hat{y}^{(i)} = w_0 + w_1 x_1^{(i)} = \mathbf{w}^T\mathbf{x}^{(i)}$.

The partial derivative of our cost function is: $\frac{\partial J(\mathbf{w})}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})x_j^{(i)}$

If we have one feature and three examples ($N = 3$) the partial derivatives are:

$$\frac{\partial J(\mathbf{w})}{\partial w_0} = \frac{1}{3}\Big(\underbrace{(\mathbf{w}^T\mathbf{x}^{(1)})}_{\hat{y}^{(1)}} - y^{(1)}) + \underbrace{(\mathbf{w}^T\mathbf{x}^{(2)})}_{\hat{y}^{(2)}} - y^{(2)}) + \underbrace{(\mathbf{w}^T\mathbf{x}^{(3)})}_{\hat{y}^{(3)}} - y^{(3)}) \Big) = \frac{1}{3}\begin{bmatrix}1 & 1 & 1\end{bmatrix}\begin{bmatrix}\hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)}\end{bmatrix}$$

$$\frac{\partial J(\mathbf{w})}{\partial w_1} = \frac{1}{3}\Big(\underbrace{(\mathbf{w}^T\mathbf{x}^{(1)})}_{\hat{y}^{(1)}} - y^{(1)})x_1^{(1)} + \underbrace{(\mathbf{w}^T\mathbf{x}^{(2)})}_{\hat{y}^{(2)}} - y^{(2)})x_1^{(2)} + \underbrace{(\mathbf{w}^T\mathbf{x}^{(3)})}_{\hat{y}^{(3)}} - y^{(3)})x_1^{(3)} \Big) = \frac{1}{3}\begin{bmatrix}x_1^{(1)} & x_1^{(2)} & x_1^{(3)}\end{bmatrix}\begin{bmatrix}\hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)}\end{bmatrix}$$

Thus we can write the gradient for these three examples as:

$$\nabla J(\mathbf{w}) = \begin{bmatrix}\frac{\partial J(\mathbf{w})}{\partial w_0} \\ \frac{\partial J(\mathbf{w})}{\partial w_1}\end{bmatrix} = \frac{1}{3}\begin{bmatrix}1 & 1 & 1 \\ x_1^{(1)} & x_1^{(2)} & x_1^{(3)}\end{bmatrix}\begin{bmatrix}\hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)}\end{bmatrix}$$

Calculations continued on the next slide

$$J(\mathbf{w}) = \frac{1}{2N}\text{RSS}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})^2$$

Small example continued:

Using the fact that $X\mathbf{w} = \hat{\mathbf{y}}$, $d = 1$, and $N = 3$ we can show $\nabla J(\mathbf{w}) = \frac{1}{N}X^T(X\mathbf{w} - \mathbf{y})$

$$\nabla J(\mathbf{w}) = \begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_0} \\ \frac{\partial J(\mathbf{w})}{\partial w_1} \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ x_1^{(1)} & x_1^{(2)} & x_1^{(3)} \end{bmatrix}\begin{bmatrix} \hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)} \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ x_1^{(1)} & x_1^{(2)} & x_1^{(3)} \end{bmatrix}\left(\begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ 1 & x_1^{(3)} \end{bmatrix}\begin{bmatrix} w_0 \\ w_1 \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \end{bmatrix}\right) = \frac{1}{N}X^T(X\mathbf{w} - \mathbf{y})$$

$$J(\mathbf{w}) = \frac{1}{2N}\text{RSS}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})^2$$

2nd small example:

The partial derivative of our objective function $\frac{\partial J(\mathbf{w})}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})x_j^{(i)}$

If we have two features and three examples ($N = 3$) the partial derivatives are:

$$\frac{\partial J(\mathbf{w})}{\partial w_0} = \frac{1}{3}\Big((\mathbf{w}^T\mathbf{x}^{(1)} - y^{(1)}) + (\mathbf{w}^T\mathbf{x}^{(2)} - y^{(2)}) + (\mathbf{w}^T\mathbf{x}^{(3)} - y^{(3)}) \Big) = \frac{1}{3}[1 \quad 1 \quad 1]\begin{bmatrix}\hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)}\end{bmatrix}$$

$$\frac{\partial J(\mathbf{w})}{\partial w_1} = \frac{1}{3}\Big((\mathbf{w}^T\mathbf{x}^{(1)} - y^{(1)})x_1^{(1)} + (\mathbf{w}^T\mathbf{x}^{(2)} - y^{(2)})x_1^{(2)} + (\mathbf{w}^T\mathbf{x}^{(3)} - y^{(3)})x_1^{(3)} \Big) = \frac{1}{3}\begin{bmatrix}x_1^{(1)} & x_1^{(2)} & x_1^{(3)}\end{bmatrix}\begin{bmatrix}\hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)}\end{bmatrix}$$

$$\frac{\partial J(\mathbf{w})}{\partial w_2} = \frac{1}{3}\Big((\mathbf{w}^T\mathbf{x}^{(1)} - y^{(1)})x_2^{(1)} + (\mathbf{w}^T\mathbf{x}^{(2)} - y^{(2)})x_2^{(2)} + (\mathbf{w}^T\mathbf{x}^{(3)} - y^{(3)})x_2^{(3)} \Big) = \frac{1}{3}\begin{bmatrix}x_2^{(1)} & x_2^{(2)} & x_2^{(3)}\end{bmatrix}\begin{bmatrix}\hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)}\end{bmatrix}$$

Thus we can write the gradient for these three examples as:

$$\nabla J(\mathbf{w}) = \begin{bmatrix}\frac{\partial J(\mathbf{w})}{\partial w_0} \\ \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \frac{\partial J(\mathbf{w})}{\partial w_2}\end{bmatrix} = \frac{1}{3}\begin{bmatrix}1 & 1 & 1 \\ x_1^{(1)} & x_1^{(2)} & x_1^{(3)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)}\end{bmatrix}\begin{bmatrix}\hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)}\end{bmatrix} = \frac{1}{N}X^T(X\mathbf{w} - \mathbf{y})$$

Example

$$J(\mathbf{w}) = \frac{1}{2N}\text{RSS}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})^2$$

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})x_j^{(i)} = \frac{1}{N}\sum_{i=1}^{N}(-\epsilon^{(i)})x_j^{(i)}$$

$$\nabla J(\mathbf{w}) = \frac{1}{N}X^T(X\mathbf{w} - \mathbf{y}) = \frac{1}{N}X^T(\hat{\mathbf{y}} - \mathbf{y})$$

$$\nabla J(\mathbf{w}) = \begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_0} \\ \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial w_d} \end{bmatrix} = \frac{1}{N}$$

1	1	1	1	1	1	1
0.038	−0.002	0.085	−0.089	0.005	−0.093	−0.045
0.051	−0.045	0.051	−0.045	−0.045	−0.045	0.051
0.062	−0.051	0.044	−0.012	−0.036	−0.041	−0.047
0.022	−0.026	−0.006	−0.037	0.022	−0.019	−0.016
−0.044	−0.008	−0.046	0.012	0.004	−0.069	−0.04
−0.035	−0.019	−0.034	0.025	0.016	−0.079	−0.025
−0.043	0.074	−0.032	−0.036	0.008	0.041	0.001
−0.003	−0.039	−0.003	0.034	−0.003	−0.076	−0.039
0.02	−0.068	0.003	0.023	−0.032	−0.041	−0.063
−0.018	−0.092	−0.026	−0.009	−0.047	−0.096	−0.038

age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
1	0.038	0.051	0.062	$2.187e-02$	−0.044	$-3.482e-02$	0.043	−0.003	0.020	−0.018
1	−0.002	−0.045	−0.051	$-2.633e-02$	−0.008	$-1.916e-02$	−0.074	−0.039	−0.068	−0.092
1	0.085	0.051	0.044	$-5.670e-03$	−0.046	$-3.419e-02$	0.032	−0.003	0.003	−0.026
1	−0.089	−0.045	−0.012	$-3.666e-02$	0.012	$2.499e-02$	0.036	0.034	0.023	−0.009
1	0.005	−0.045	−0.036	$2.187e-02$	0.004	$1.560e-02$	−0.008	−0.003	−0.032	−0.047
1	−0.093	−0.045	−0.041	$-1.944e-02$	−0.069	$-7.929e-02$	−0.041	−0.076	−0.041	−0.096
1	−0.046	0.051	−0.047	$-1.600e-02$	−0.040	$-2.480e-02$	−0.001	−0.039	−0.063	−0.038
⋮										

$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \\ w_8 \\ w_9 \\ w_{10} \end{bmatrix} - \begin{bmatrix} 204.51116637 \\ 67.10485972 \\ 175.02956894 \\ 165.88615565 \\ 123.11207835 \\ 105.64709238 \\ 71.73293158 \end{bmatrix}$$

$$= \frac{1}{N}$$

1	1	1	1	1	1	1
0.038	−0.002	0.085	−0.089	0.005	−0.093	−0.045
0.051	−0.045	0.051	−0.045	−0.045	−0.045	0.051
0.062	−0.051	0.044	−0.012	−0.036	−0.041	−0.047
0.022	−0.026	−0.006	−0.037	0.022	−0.019	−0.016
−0.044	−0.008	−0.046	0.012	0.004	−0.069	−0.04
−0.035	−0.019	−0.034	0.025	0.016	−0.079	−0.025
−0.043	0.074	−0.032	−0.036	0.008	0.041	0.001
−0.003	−0.039	−0.003	0.034	−0.003	−0.076	−0.039
0.02	−0.068	0.003	0.023	−0.032	−0.041	−0.063
−0.018	−0.092	−0.026	−0.009	−0.047	−0.096	−0.038

$$\begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \hat{y}^{(3)} \\ \hat{y}^{(4)} \\ \hat{y}^{(5)} \\ \hat{y}^{(6)} \\ \hat{y}^{(7)} \end{bmatrix} - \begin{bmatrix} 204.51116637 \\ 67.10485972 \\ 175.02956894 \\ 165.88615565 \\ 123.11207835 \\ 105.64709238 \\ 71.73293158 \end{bmatrix}$$

$$= \frac{1}{N}$$

1	1	1	1	1	1	1
0.038	−0.002	0.085	−0.089	0.005	−0.093	−0.045
0.051	−0.045	0.051	−0.045	−0.045	−0.045	0.051
0.062	−0.051	0.044	−0.012	−0.036	−0.041	−0.047
0.022	−0.026	−0.006	−0.037	0.022	−0.019	−0.016
−0.044	−0.008	−0.046	0.012	0.004	−0.069	−0.04
−0.035	−0.019	−0.034	0.025	0.016	−0.079	−0.025
−0.043	0.074	−0.032	−0.036	0.008	0.041	0.001
−0.003	−0.039	−0.003	0.034	−0.003	−0.076	−0.039
0.02	−0.068	0.003	0.023	−0.032	−0.041	−0.063
−0.018	−0.092	−0.026	−0.009	−0.047	−0.096	−0.038

$$\begin{bmatrix} -\epsilon^{(1)} \\ -\epsilon^{(2)} \\ -\epsilon^{(3)} \\ -\epsilon^{(4)} \\ -\epsilon^{(5)} \\ -\epsilon^{(6)} \\ -\epsilon^{(7)} \end{bmatrix} = \nabla J(\mathbf{w}) = \begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_0} \\ \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial w_d} \end{bmatrix}$$

Batch Gradient Descent Using Vectorization

$$J(\mathbf{w}) = \frac{1}{2N}\text{RSS}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})^2 \qquad \nabla J(\mathbf{w}) = \frac{1}{N}X^T(X\mathbf{w} - \mathbf{y})$$

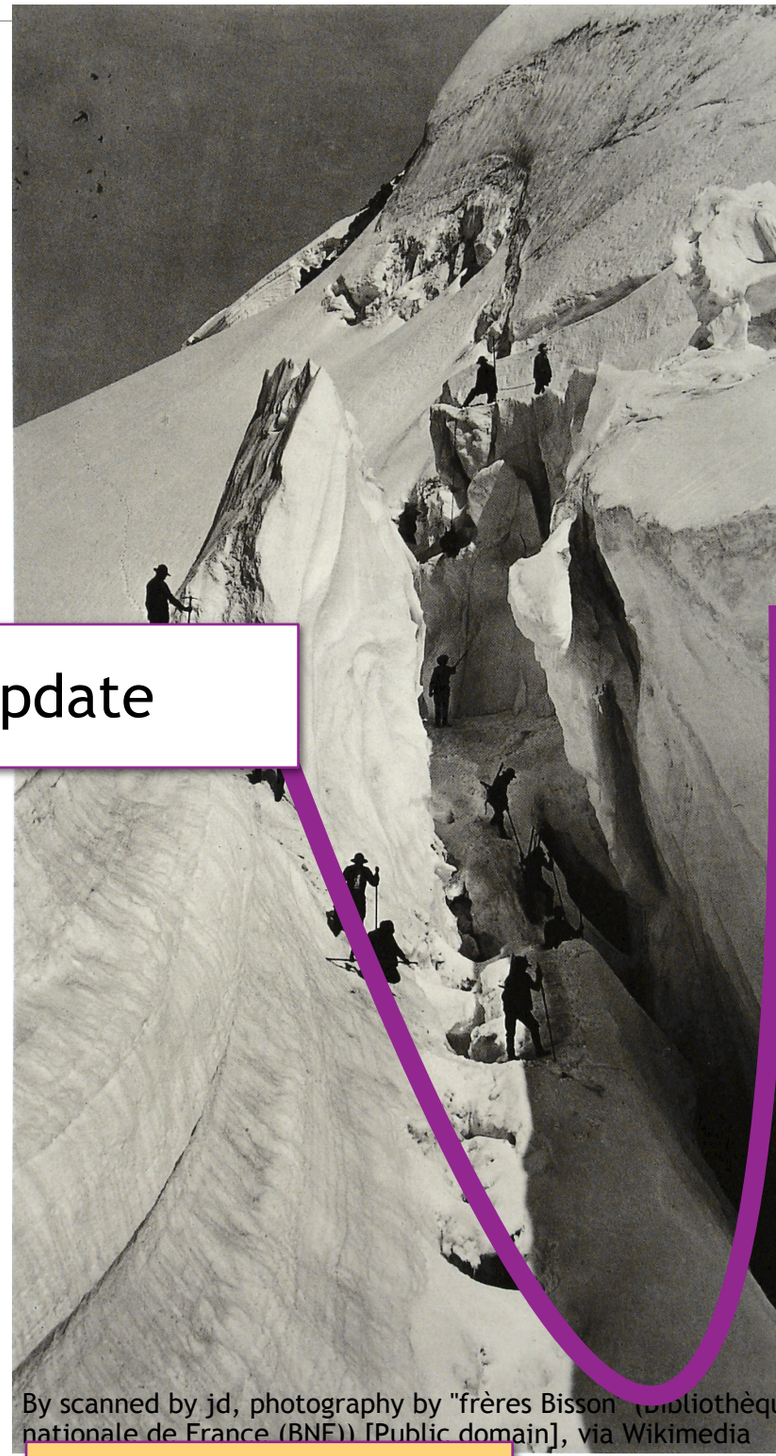
- We can perform gradient updates for all parameters $\mathbf{w} = [w_0, w_1, \ldots, w_d]$ in a single line of vectorized code

for $i = 1$ to num_iter

$$\mathbf{w} = \mathbf{w} - \alpha\nabla J(\mathbf{w}) = \mathbf{w} - \alpha\frac{1}{N}X^T(X\mathbf{w} - \mathbf{y})$$

Simultaneous update

- Note that:
  - $\mathbf{w}$ and $\nabla J(\mathbf{w})$ are $(d+1)\times 1$ column vectors
  - all $\mathbf{w}$ are getting updated simultaneously, and we do not need to store them in a temporary variable



By scanned by jd, photography by "frères Bisson (Bibliothèque nationale de France (BNF)) [Public domain], via Wikimedia

Running time: $O(Nd$ #iter$)$

Outline

- Motivating Example: Predicting the mpg of a car
- Model: Linear Model
- Objective function: Least Squares Fit Problem
- Local Optimizer: “Batch” Gradient Descent
  - When $d = 1$ (simple linear regression)
  - What can go wrong?
  - General case, any $d > 0$ (multiple variable linear regression)
    - Example
    - Algorithm
    - Vectorized Algorithm
- Global Optimization: Normal Equations
- Feature transformation
- Assessing Goodness of Fit
- Objective function revisited: Probabilistic interpretation.
- Extensions
- Removing features

Global Optimization

Our second method to finding the parameters $\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$

Updated loss function

General ML problem

- ❑ Get data
- ❑ Pick a model with parameters
- ❑ Pick a loss function
  - Measures goodness of fit model to data
  - Function of the parameters
- ❑ Find parameters that minimizes loss

Multiple linear regression

Data: $(\mathbf{x}^{(i)}, y^{(i)}), i = 1,2,\ldots,N$

Linear model:

$$\hat{y}^{(j)} = w_0 + w_1 x_1^{(j)} + w_2 x_2^{(j)} + \cdots + w_d x_d^{(j)}$$

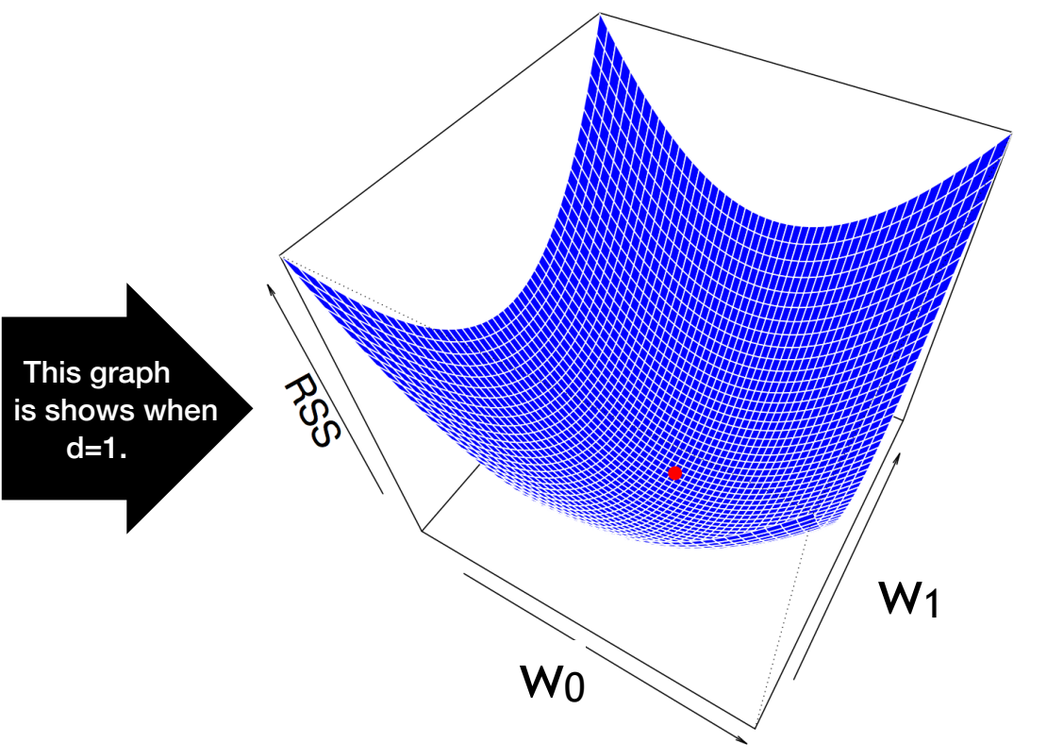
Loss function: $\frac{\text{RSS}(\mathbf{w})}{2N} = \frac{1}{2N}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2$

$$= \frac{1}{2N}\left[(y^{(1)} - \hat{y}^{(1)})^2 + (y^{(2)} - \hat{y}^{(2)})^2 + \cdots + (y^{(N)} - \hat{y}^{(N)})^2\right]$$

Select $\mathbf{w} = [w_0, w_1, w_2, \ldots, w_d]^T$ to minimize $\frac{\text{RSS}(\mathbf{w})}{2N}$ This $\mathbf{w}$ also minimizes $\text{RSS}(\mathbf{w})$

Global optimization for finding **w** to minimize

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$



$$\nabla J(\mathbf{w}) = \begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_0} \\ \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial w_d} \end{bmatrix} = \frac{1}{N}X^T(X\mathbf{w} - \mathbf{y}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Goal find $\mathbf{w}$ such that $\nabla J(\mathbf{w}) = \mathbf{0}$

Finding **w** to minimize J(w)

Goal find $\mathbf{w}$ such that $\nabla J(\mathbf{w}) = \mathbf{0}$

$$\nabla J(\mathbf{w}) = \frac{1}{N} X^T (X\mathbf{w} - \mathbf{y}) = \frac{1}{N}(X^T X\mathbf{w} - X^T\mathbf{y})$$

Setting $\nabla J(\mathbf{w}) = \frac{1}{N}(X^T X\mathbf{w} - X^T\mathbf{y}) = \mathbf{0}$

Results in: $X^T X\mathbf{w} = X^T\mathbf{y}$

Thus $\mathbf{w}_{\text{lin}} = \underbrace{\left(X^T X\right)^{-1} X^T}\mathbf{y}$

pseudoinverse
left inverse

We added 'lin' to $\mathbf{w}$ to specify it was linear regression

Running time:
$O(Nd^2)$ for $X^T X$
$O(d^3)$ for inverse $(X^T X)^{-1}$
$O(dN)$ for $X^T\mathbf{y}$
$O(d^2)$ for $(X^T X)^{-1}$ times $X^T\mathbf{y}$
————————————————————
——
Total $O(Nd^2) + O(d^3)$

*finding the inverse of a $d \times d$ matrix can be found in $O(d^{2.373})$ time

If the columns of the matrix X are linearly independent then $X^T X$ is invertible and

$$X^+ = (X^T X)^{-1} X^T$$

is the pseudo inverse, i.e. the left inverse of X

$$X^+ X = (X^T X)^{-1} X^T X = I$$

http://sci.utah.edu/~gerig/CS6640-F2012/Materials/pseudoinverse-cis61009sl10.pdf

Finding **w** to minimize E_{in} (and RSS)

Linear Regression Algorithm:

1. Construct the matrix X and the vector $\mathbf{y}$ from the data set $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$, where each $\mathbf{x}$ includes the $x_0 = 1$ coordinate,

$$\underbrace{\mathrm{X} = \begin{bmatrix} —\mathbf{x}_1— \\ —\mathbf{x}_2— \\ \vdots \\ —\mathbf{x}_N— \end{bmatrix}}_{\text{data matrix}}, \qquad \underbrace{\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\text{target vector}}.$$

2. Compute the pseudo inverse $\mathrm{X}^\dagger$ of the matrix X. If $\mathrm{X}^\mathrm{T}\mathrm{X}$ is invertible,

$$\mathrm{X}^\dagger = (\mathrm{X}^\mathrm{T}\mathrm{X})^{-1}\mathrm{X}^\mathrm{T}$$

3. Return $\mathbf{w}_{\text{lin}} = \mathrm{X}^\dagger\mathbf{y}$.

Algorithm from book Learning form Data

Example

$$\mathbf{w}_{\text{lin}} = \left(X^T X\right)^{-1} X^T \mathbf{y}$$

$$\mathbf{w}_{\text{lin}} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \\ w_8 \\ w_9 \\ w_{10} \end{bmatrix} = \left[\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots \\ 0.038 & -0.002 & 0.085 & -0.089 & 0.005 & -0.093 & -0.045 & \\ 0.051 & -0.045 & 0.051 & -0.045 & -0.045 & -0.045 & 0.051 & \\ 0.062 & -0.051 & 0.044 & -0.012 & -0.036 & -0.041 & -0.047 & \\ 0.022 & -0.026 & -0.006 & -0.037 & 0.022 & -0.019 & -0.016 & \\ -0.044 & -0.008 & -0.046 & 0.012 & 0.004 & -0.069 & -0.04 & \\ -0.035 & -0.019 & -0.034 & 0.025 & 0.016 & -0.079 & -0.025 & \\ -0.043 & 0.074 & -0.032 & -0.036 & 0.008 & 0.041 & 0.001 & \\ -0.003 & -0.039 & -0.003 & 0.034 & -0.003 & -0.076 & -0.039 & \\ 0.02 & -0.068 & 0.003 & 0.023 & -0.032 & -0.041 & -0.063 & \\ -0.018 & -0.092 & -0.026 & -0.009 & -0.047 & -0.096 & -0.038 & \end{bmatrix} \begin{bmatrix} & \texttt{age} & \texttt{sex} & \texttt{bmi} & \texttt{map} & \texttt{tc} & \texttt{ldl} & \texttt{hdl} & \texttt{tch} & \texttt{ltg} & \texttt{glu} \\ 1 & 0.038 & 0.051 & 0.062 & 2.187e-02 & -0.044 & -3.482e-02 & 0.043 & -0.003 & 0.020 & -0.018 \\ 1 & -0.002 & -0.045 & -0.051 & -2.633e-02 & -0.008 & -1.916e-02 & -0.074 & -0.039 & -0.068 & -0.092 \\ 1 & 0.085 & 0.051 & 0.044 & -5.670e-03 & -0.046 & -3.419e-02 & 0.032 & -0.003 & 0.003 & -0.026 \\ 1 & -0.089 & -0.045 & -0.012 & -3.666e-02 & 0.012 & 2.499e-02 & 0.036 & 0.034 & 0.023 & -0.009 \\ 1 & 0.005 & -0.045 & -0.036 & 2.187e-02 & 0.004 & 1.560e-02 & -0.008 & -0.003 & -0.032 & -0.047 \\ 1 & -0.093 & -0.045 & -0.041 & -1.944e-02 & -0.069 & -7.929e-02 & -0.041 & -0.076 & -0.041 & -0.096 \\ 1 & -0.046 & 0.051 & -0.047 & -1.600e-02 & -0.040 & -2.480e-02 & -0.001 & -0.039 & -0.063 & -0.038 \\ \vdots & & & & & & & & & & \end{bmatrix} \right]^{-1} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots \\ 0.038 & -0.002 & 0.085 & -0.089 & 0.005 & -0.093 & -0.045 & \\ 0.051 & -0.045 & 0.051 & -0.045 & -0.045 & -0.045 & 0.051 & \\ 0.062 & -0.051 & 0.044 & -0.012 & -0.036 & -0.041 & -0.047 & \\ 0.022 & -0.026 & -0.006 & -0.037 & 0.022 & -0.019 & -0.016 & \\ -0.044 & -0.008 & -0.046 & 0.012 & 0.004 & -0.069 & -0.04 & \\ -0.035 & -0.019 & -0.034 & 0.025 & 0.016 & -0.079 & -0.025 & \\ -0.043 & 0.074 & -0.032 & -0.036 & 0.008 & 0.041 & 0.001 & \\ -0.003 & -0.039 & -0.003 & 0.034 & -0.003 & -0.076 & -0.039 & \\ 0.02 & -0.068 & 0.003 & 0.023 & -0.032 & -0.041 & -0.063 & \\ -0.018 & -0.092 & -0.026 & -0.009 & -0.047 & -0.096 & -0.038 & \end{bmatrix} \begin{bmatrix} 204.51116637 \\ 67.10485972 \\ 175.02956894 \\ 165.88615565 \\ 123.11207835 \\ 105.64709238 \\ 71.73293158 \end{bmatrix}$$

$$\mathbf{w}_{\text{lin}} = \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \\ w_8 \\ w_9 \\ w_{10} \end{bmatrix} = \begin{bmatrix} 152.34786452 \\ -16.57607993 \\ -254.66532396 \\ 560.98630022 \\ 278.91811152 \\ -393.41357305 \\ 97.05460405 \\ -19.0023093 \\ 169.46450327 \\ 632.95050374 \\ 114.21638941 \end{bmatrix}$$

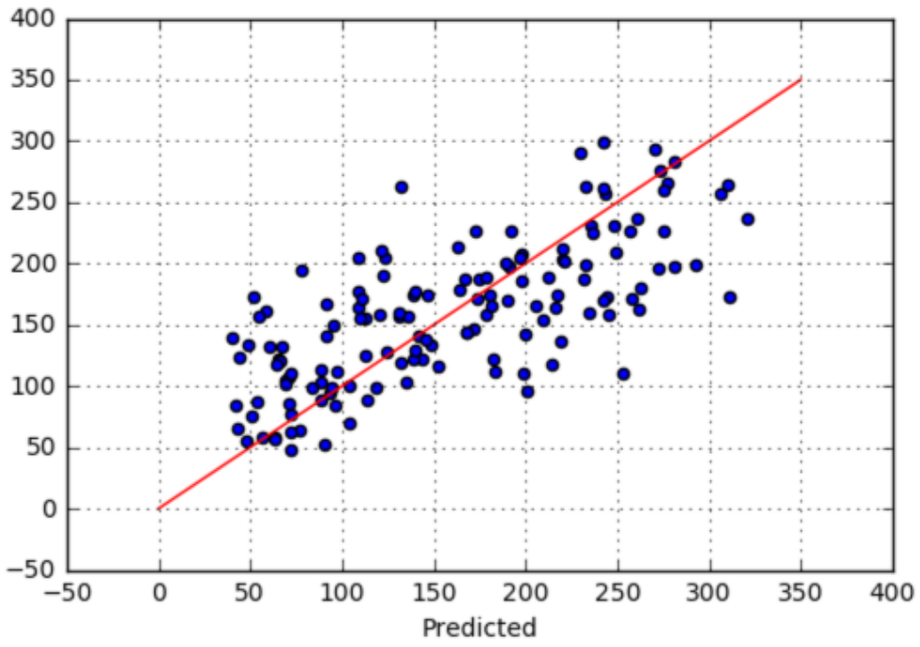
https://numpy.org/doc/stable/reference/generated/numpy.linalg.pinv.html

Python code: `w_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)`

Python code using the pseudoinverse: `w_best = np.linalg.pinv(X_b).dot(y)`

Best fitting coefficients/weights

- The difference between the predicted and the true values



$$w = \begin{bmatrix} 152.34786452 \\ -16.57607993 \\ -254.66532396 \\ 560.98630022 \\ 278.91811152 \\ -393.41357305 \\ 97.05460405 \\ -19.0023093 \\ 169.46450327 \\ 632.95050374 \\ 114.21638941 \end{bmatrix}$$

$R^2 = 0.514719$

measure of “goodness”

Outline

- Motivating Example: Predicting the mpg of a car
- Model: Linear Model
- Objective function: Least Squares Fit Problem
- Local Optimizer: “Batch” Gradient Descent
  - When $d = 1$ (simple linear regression)
  - What can go wrong?
  - General case, any $d > 0$ (multiple variable linear regression)
    - Example
    - Algorithm
    - Vectorized Algorithm
- Global Optimization: Normal Equations
- Feature transformation
- Assessing Goodness of Fit
- Objective function revisited: Probabilistic interpretation.
- Extensions
- Removing features

This is a very brief introduction to the topic. We will transform before training.

Why do we transform features

Warning - all transformations needs to also be performed on any test/validation/new data.

- Necessary transformations
  - Some algorithms require numeric data
  - Some algorithms expect the input to be of a specific size

1st

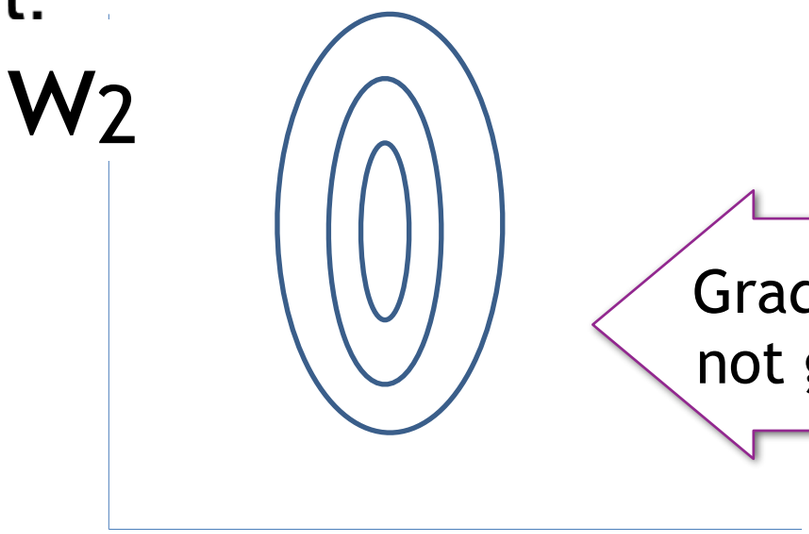
- Optional transformations
  - Normalizing numeric features
  - Creating non-linearities in the feature space - thus allowing us to still use linear models (next topic)

Normalization

Goal: transform features to have a similar scale

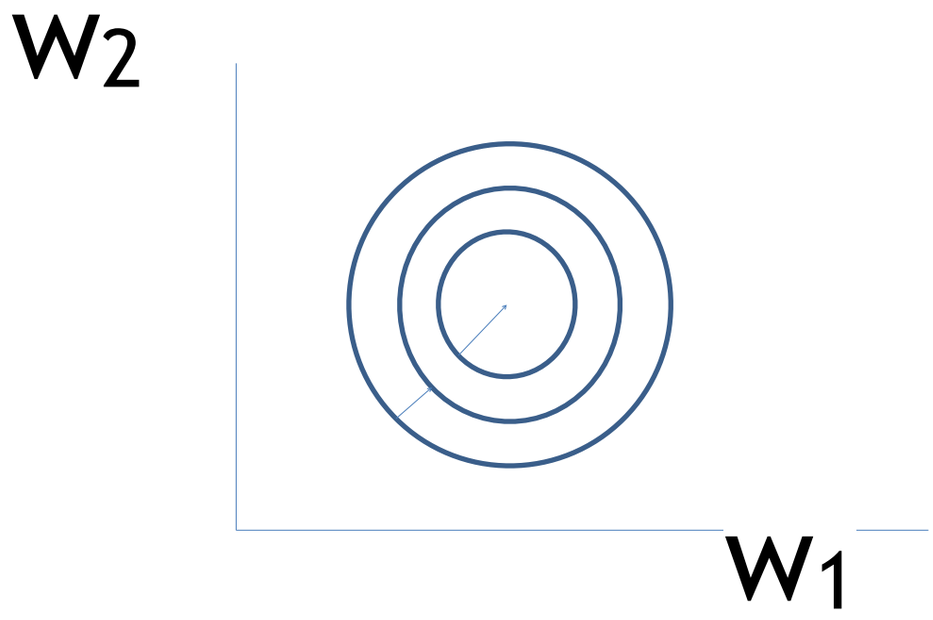
Cost Function

- Visualizing cost function using a contour plot.



Gradient points to steepest descent locally not globally

After Scaling



Example: Diabetes Design Matrix

age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
59	2	32.1	101	157	93.2	38	4	4.8598	87
48	1	21.6	87	183	103.2	70	3	3.8918	69
72	2	30.5	93	156	93.6	41	4	4.6728	85
24	1	25.3	84	198	131.4	40	5	4.8903	89
50	1	23	101	192	125.4	52	4	4.2905	80
23	1	22.6	89	139	64.8	61	2	4.1897	68
36	2	22	90	160	99.6	50	3	3.9512	82

age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
0.038	0.051	0.062	2.187e-02	-0.044	-3.482e-02	0.043	-0.003	0.020	-0.018
-0.002	-0.045	-0.051	-2.633e-02	-0.008	-1.916e-02	-0.074	-0.039	-0.068	-0.092
0.085	0.051	0.044	-5.670e-03	-0.046	-3.419e-02	0.032	-0.003	0.003	-0.026
-0.089	-0.045	-0.012	-3.666e-02	0.012	2.499e-02	0.036	0.034	0.023	-0.009
0.005	-0.045	-0.036	2.187e-02	0.004	1.560e-02	-0.008	-0.003	-0.032	-0.047
-0.093	-0.045	-0.041	-1.944e-02	-0.069	-7.929e-02	-0.041	-0.076	-0.041	-0.096
-0.046	0.051	-0.047	-1.600e-02	-0.040	-2.480e-02	-0.001	-0.039	-0.063	-0.038

Feature Scaling
AKA Data Normalization

Before applying *many* machine learning algorithms:

- Make features on a similar scale.
  Prevents one feature from overly influencing the algorithm.

Before performing gradient descent, apply feature scaling to improve the rate the algorithm converges

- For example, suppose you are using 2 features for predicting the price of a house
  - X_1 = size (0 – 4000 sq ft)
  - X_2 = No. of bedrooms (1 – 5)

Types of Feature Scaling
AKA Data Normalization

- Min/Max Normalization: scaling to a range
- Standardization (Z-score normalization)
- For other methods see https://en.wikipedia.org/wiki/Feature_scaling

Min/Max Normalization

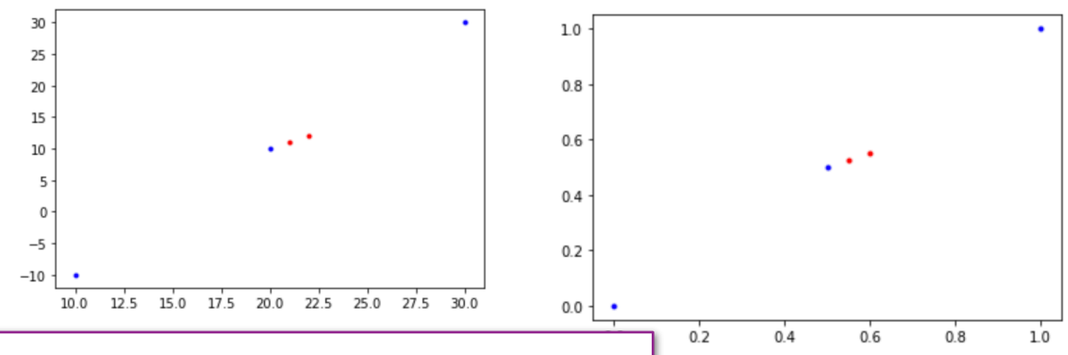
```
from sklearn.preprocessing import MinMaxScaler
data = np.array([[10, -10], [20, 10], [30, 30]])
test=np.array([[21,11],[22,12]])
scaler = MinMaxScaler()
scaler.fit(data)
scaled_data = scaler.transform(data)
scaled_test = scaler.transform(test)
```

- Scale the range of features to become [0,1] (or [a,b])
- Steps:

$$x_1^{(i)} = \frac{x_1^{(i)} - (10)}{30 - (10)} \qquad x_2^{(i)} = \frac{x_2^{(i)} - (-10)}{30 - (-10)}$$

For each feature, j, in the data (except x_0)

- Find the range of values $[\min(x_j), \max(x_j)]$ for the j[th] feature
- Update every example's jth feature:



$$x_j^{(i)} = \frac{x_j^{(i)} - \min(x_j)}{\max(x_j) - \min(x_j)}$$

Pair share: What happens if there is an outlier?
i.e. if you have a few large value and many small values?

Eg: If the range of feature 1 is [0, 4000]

update all training example such that $x^{(i)}_1 = (x^{(i)}_1 - 0)/4000$. Now the range of $x^{(i)}_1$ is [0,1]

If the range if the second feature is [1,5] update all training examples $x^{(i)}_2 = (x^{(i)}_2 - 1)/4$. Now the range of the second feature is [0, 1]

Min/Max Normalization

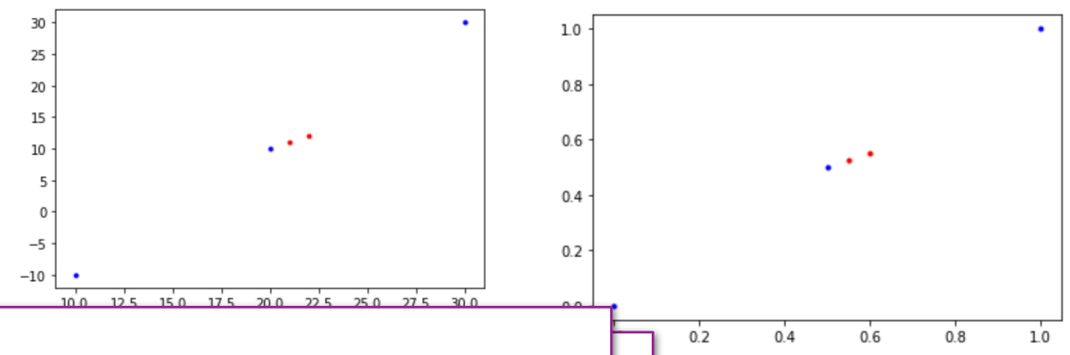
```
from sklearn.preprocessing import MinMaxScaler
data = np.array([[10, -10], [20, 10], [30, 30]])
test=np.array([[21,11],[22,12]])
scaler = MinMaxScaler()
scaler.fit(data)
scaled_data = scaler.transform(data)
scaled_test = scaler.transform(test)
```

- Scale the range of features to become [0,1] (or [a,b])
- Steps:

$$x_1^{(i)} = \frac{x_1^{(i)} - (10)}{30 - (10)} \qquad x_2^{(i)} = \frac{x_2^{(i)} - (-10)}{30 - (-10)}$$

For each feature, j, in the data (except x_0)

- Find the range of values $[\min(x_j), \max(x_j)]$ for the j[th] feature
- Update every example's jth feature:



$$x_j^{(i)} = \frac{x_j^{(i)} - \min(x_j)}{\max(x_j) - \min(x_j)}$$

Not covered in class: Feature clipping: caps all features above (or below) a certain value to a fixed value.

Eg: If the range of feature 1 is [0, 4000]

update all training example such that $x^{(i)}_1 = (x^{(i)}_1 - 0)/4000$. Now the range of $x^{(i)}_1$ is [0,1]

If the range if the second feature is [1,5] update all training examples $x^{(i)}_2 = (x^{(i)}_2 - 1)/4$. Now the range of the second feature is [0, 1]

Standardization (in social science this is call Z-score normalization)

```
from sklearn.preprocessing import StandardScaler
data = [[0, 2], [0, 0], [1, 4], [1, 1]]
scaler = StandardScaler()
print(scaler.fit(data))
print(scaler.mean_)
print(scaler.transform(data))

print(scaler.transform([[2, 2]]))
```

$$x_1^{(i)} = \frac{x_1^{(i)} - 0.5}{0.5}$$

- Scale the range of features to become zero centered and have unit variance
- Steps:

  For each feature, j, in the data

  - Find the average for the j[th] feature: $\text{ave}(x_j) = \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)}$
  - Find the standard deviation for the j[th] feature: $\text{STD}(x_j) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_j^{(i)} - \text{ave}(x_j)\right)^2}$
  - Update the j[th] feature

$$x_j^{(i)} = \frac{x_j^{(i)} - \text{ave}(x_j)}{\text{STD}(x_j)}$$

Eg: If the average value of feature j is 70, and the standard deviation is 12 update all training example such that $x_j^{(i)} = \frac{x_j^{(i)} - 70}{12}$ Now the average value of feature j is 0, and the standard deviation of feature j for the training examples is 1

It is much less affected by outliers.is not bounded to a certain range.

It translates the data to the mean vector of original data to the origin and squishes or expands.

Diabetes dataset

Mean Centering and Scaling to Unit Length example

"these data are first standardized to have zero mean and unit L2 norm before they are used in the examples."

The L2 norm is the standard deviations when the mean is zero

Original Data

age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
59	2	32.1	101	157	93.2	38	4	4.8598	87
48	1	21.6	87	183	103.2	70	3	3.8918	69
72	2	30.5	93	156	93.6	41	4	4.6728	85
24	1	25.3	84	198	131.4	40	5	4.8903	89
50	1	23	101	192	125.4	52	4	4.2905	80
23	1	22.6	89	139	64.8	61	2	4.1897	68
36	2	22	90	160	99.6	50	3	3.9512	82

$X =$

	age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
1	0.038	0.051	0.062	2.187e-02	-0.044	-3.482e-02	0.043	-0.003	0.020	-0.018
1	-0.002	-0.045	-0.051	-2.633e-02	-0.008	-1.916e-02	-0.074	-0.039	-0.068	-0.092
1	0.085	0.051	0.044	-5.670e-03	-0.046	-3.419e-02	0.032	-0.003	0.003	-0.026
1	-0.089	-0.045	-0.012	-3.666e-02	0.012	2.499e-02	0.036	0.034	0.023	-0.009
1	0.005	-0.045	-0.036	2.187e-02	0.004	1.560e-02	-0.008	-0.003	-0.032	-0.047
1	-0.093	-0.045	-0.041	-1.944e-02	-0.069	-7.929e-02	-0.041	-0.076	-0.041	-0.096
1	-0.046	0.051	-0.047	-1.600e-02	-0.040	-2.480e-02	-0.001	-0.039	-0.063	-0.038

work area

AGE		mean		
59		48.5		10.5
72		48.5		-0.5
24	-	48.5	=	23.5
50		48.5		-24.5
23		48.5		1.5
36		48.5		-25.5
				-12.5

The standard deviation of the age feature is 275.3

10.5/275.3	=	0.038
-0.5/275.3		-0.002
23.5/275.3		0.085
-24.5/275.3		-0.089
1.5/275.3		0.0054
-25.5/275.3		-0.093
-12.5/275.3		-0.045

Data from https://web.stanford.edu/~hastie/Papers/LARS/diabetes.data

I used the full dataset when I computed the mean and the l2 norm

Learn more at https://en.wikipedia.org/wiki/Feature_scaling

Necessary transformations

Many algorithms require numerical data

How would we handle Categorical data?

Categorical data contains labels: {BMW, VW, Ford, GM} or {low, medium, high}, etc

59	Female	'<1H OCEAN'	Cold
48	Male	‘ <1H OCEAN’	Warm
72	Female	'NEAR OCEAN'	Cold
24	Male	‘INLAND’	Hot
50	Male	'<1H OCEAN’	Warm
23	Male	'NEAR BAY’	Warm
36	Female	`NEAR OCEAN’	Cold

categorical data

Some categories contain a *natural ordering*: low, medium, high

Other categories *don't*: BMW, VW, Ford, GM

Many machine learning algorithms cannot work with categorical data.

How can we convert them to numerical data?

Two common approaches:

- Ordinal encoding
- One-hot encoding

ordinal encoding

If the $x_j \in$ { cold, warm, hot}

Cold	0
Warm	1
Cold	0
Hot	2
Warm	1

Suppose x_i is a categorical variable

- One of a finite number of choices
- Example: “place” (gold silver bronze), (cold, warm, hot), etc

“**Ordinal encoding** should be used for *ordinal variables* (where order matters, like `cold, warm, hot`)”

- Assigns an integer to encode each value

Warning! Sklearn OrdinalEncoder class assigns integers to categories based on alphabetic ordering
If you want the “right” encoding - assign the order “manually” by using the *categories* argument.

https://feature-engine.readthedocs.io/en/latest/encoding/OrdinalEncoder.html

https://datascience.stackexchange.com/questions/39317/difference-between-ordinalencoder-and-labelencoder

One Hot Coding

Suppose x_i is a categorical variable

- One of a finite number of choices
- Example: male or female, or model of a car, location of a house, etc

```
                                                    <1H OCEAN'  'INLAND'  'ISLAND'  'NEAR BAY'  'NEAR OCEAN'
[['<1H OCEAN']                                       [1., 0., 0., 0., 0.],
 ['<1H OCEAN']                                       [1., 0., 0., 0., 0.],
 ['NEAR OCEAN']                                      [0., 0., 0., 0., 1.],
 ...                                                 ...,
 ['INLAND']                                          [0., 1., 0., 0., 0.],
['<1H OCEAN']                                        [1., 0., 0., 0., 0.],
['NEAR BAY']]                                        [0., 0., 0., 1., 0.]
```

Dummy variable encoding

If the $x_i \in$ { Ford, BMW, GM, VW}

Model	u_1	u_2	u_3
Ford	0	0	0
BMW	1	0	0
GM	0	1	0
VW	0	0	1

Dummy variable encoding is the preferred method for linear regression.
This method avoids creating collinearity

One Hot Coding

Suppose x_i is a categorical variable

- One of a finite number of choices
- Example: male or female, or model of a car, location of a house, etc

```
['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'
...
['INLAND']
['<1H OCEAN']
['NEAR BAY']]
```

1 - if value is equal
0 - otherwise

```
<1H OCEAN'   'INLAND'   'ISLAND'   'NEAR BAY'   'NEAR OCEAN
[1., 0., 0., 0., 0.],
[1., 0., 0., 0., 0.],
[0., 0., 0., 0., 1.],
...,
[0., 1., 0., 0., 0.],
[1., 0., 0., 0., 0.],
[0., 0., 0., 1., 0.]
```

Dummy variable encoding

If the $x_i \in$ { Ford, BMW, GM, VW}

Dummy variable encoding is the preferred method for linear regression.
This method avoids creating collinearity

If the number of categories is 2, then create a new variable x_i that takes takes two values.
E.g. if we have a gender variable create

$$x_i^{(j)} = \begin{cases} 0 \text{ if jth person is female} \\ 1 \text{ if jth person is male} \end{cases}$$

Toy Example

Preprocessing step: feature transformation

59	Female	'<1H OCEAN'	Cold
48	Male	‘ <1H OCEAN’	Warm
72	Female	'NEAR OCEAN'	Cold
24	Male	‘INLAND’	Hot
50	Male	'<1H OCEAN’	Warm
23	Male	'NEAR BAY’	Warm
36	Female	NEAR OCEAN’	Cold

0.79	0	1, 0, 0, 0, 0.	0
0.19	1	1, 0, 0, 0, 0.	1
1.51	0	0, 0, 0, 0, 1	0
-1.14	1	0, 1, 0, 0, 0	2
0.30	1	1, 0, 0, 0, 0	1
-1.19	1	0, 0, 0, 1, 0	1
-0.47	0	0, 0, 0, 0, 1	0

Standardization for the first feature:

- Mean = 44.57
- Standard deviation = 18.09

(59 - 44.57)/18.09
(48 - 44.57)/18.09
(72 - 44.57)/18.09
(24 - 44.57)/18.09
(50 - 44.57)/18.09
(23 - 44.57)/18.09
(36 - 44.57)/18.09