Linear Regression

Adopted from slides by William Cohen, Geoffrey Hinton, and Alexander Ihler

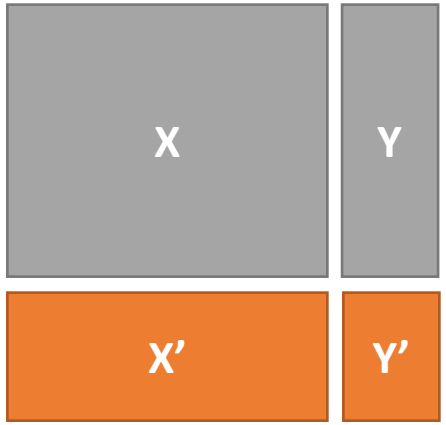
Supervised learning

- **Would like to do prediction:**

  estimate a function **f(x)** so that ***y = f(x)***

- **Where *y* can be:**
  - **Real number:** Regression
  - **Categorical:** Classification

- **Data is labeled:**
  - Have many pairs **{(x, y)}**
    - **x** … vector of binary, categorical, real valued features
    - **y** … class: {+1, -1}, or a real number

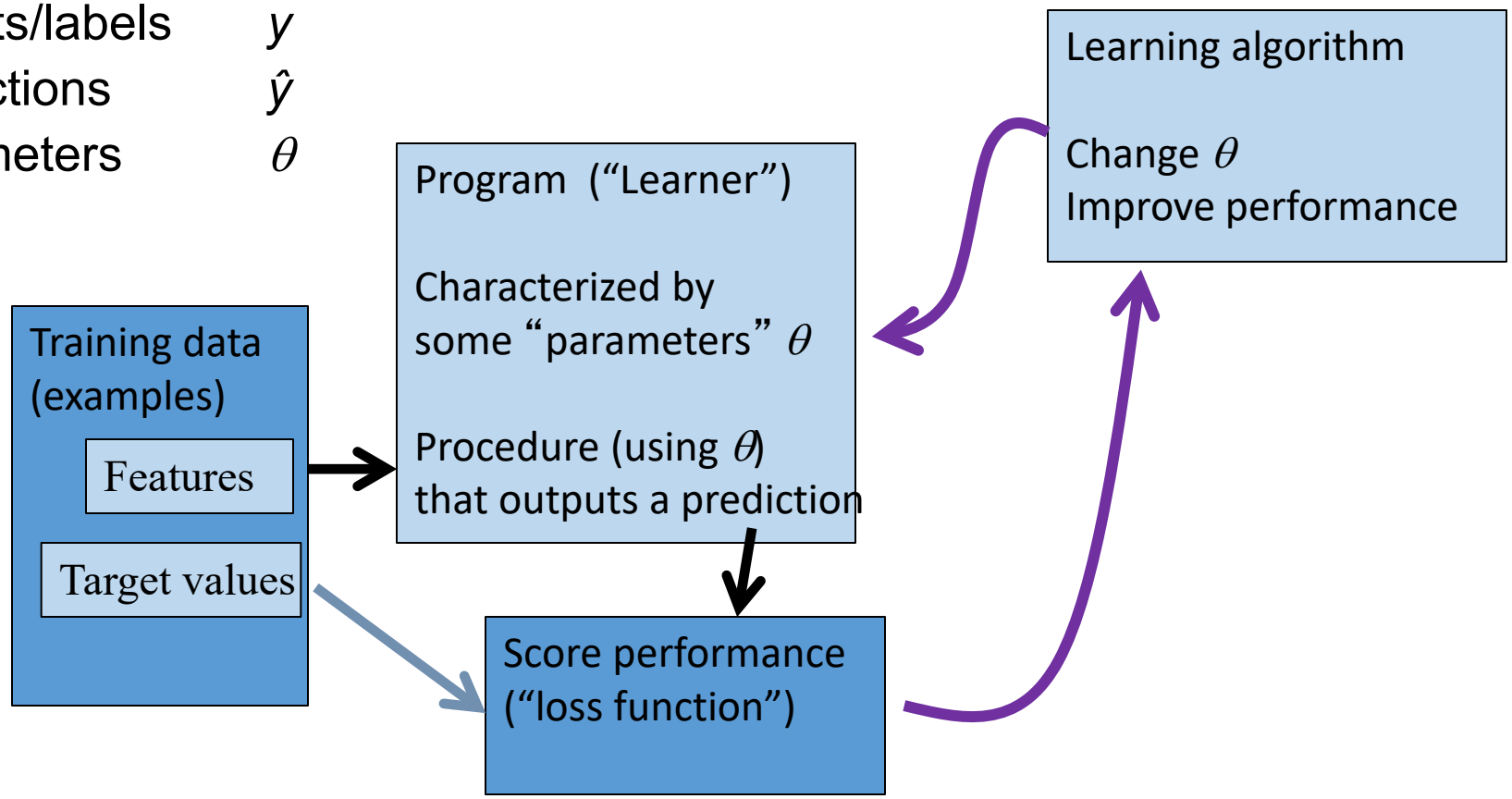


Training and **test** set

Estimate *y = f(x)* on X,Y.
Hope that the same *f(x)*
also works on unseen X', Y'

Supervised learning – Training of parametric models

- Notation
  - Features x
  - Targets/labels y
  - Predictions $\hat{y}$
  - Parameters θ



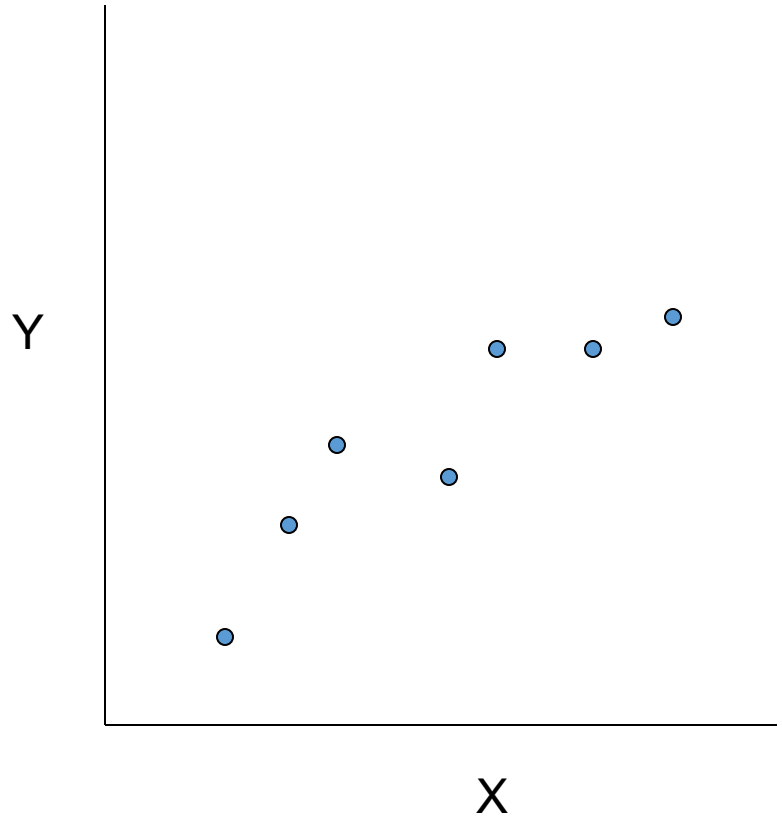
Linear regression

- Given an input x we would like to compute an output y
- For example:

  - Predict height from age

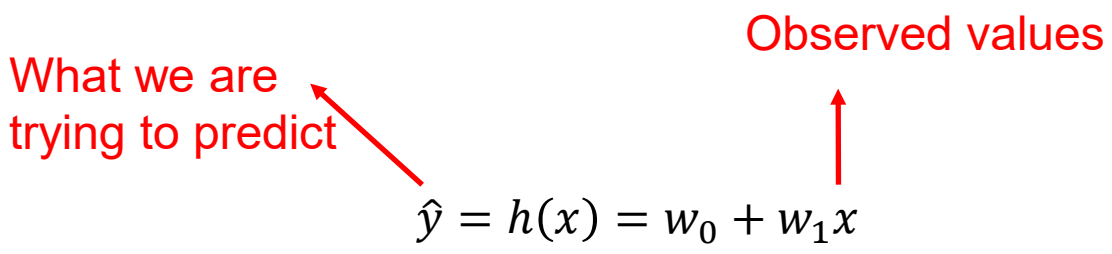
  - Predict Google's price from Yahoo's price

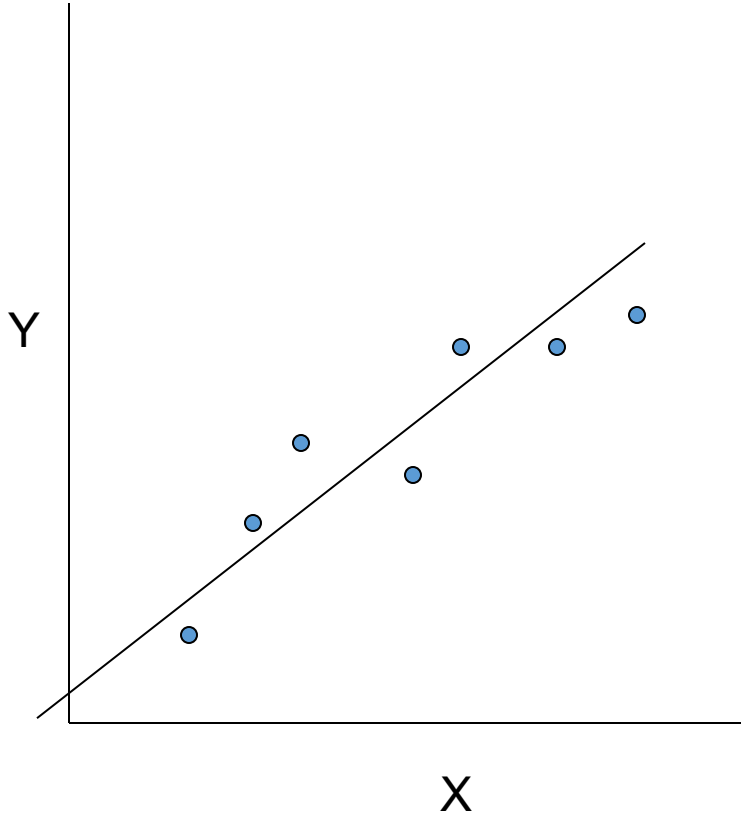
  - Predict distance from wall from sensors



Linear regression

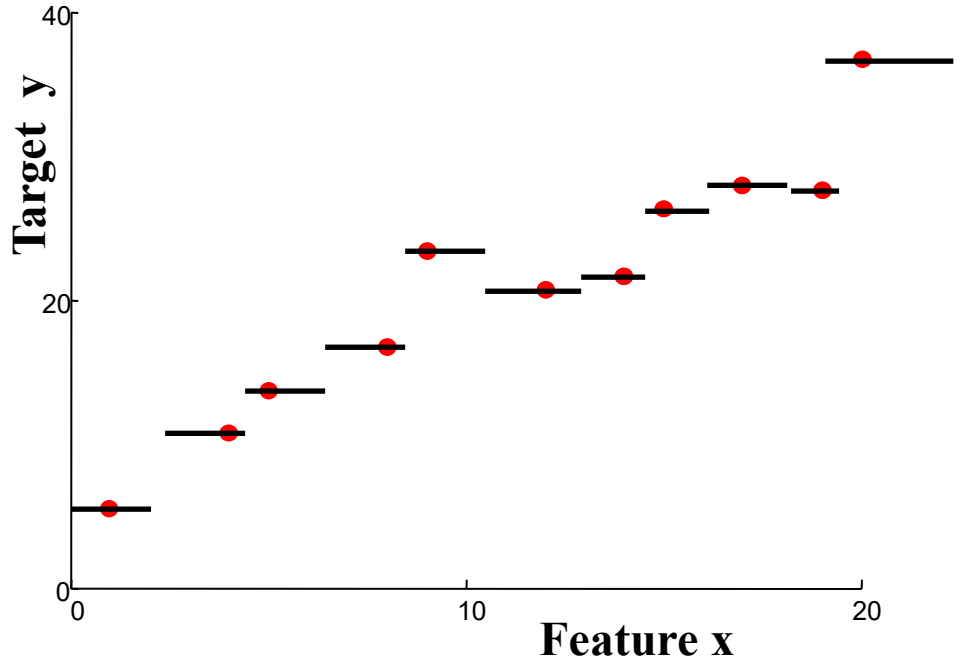
- Given an input x we would like to compute an output y
- In linear regression we assume that y and x are related with the following equation:





where w_0, w_1 are parameters

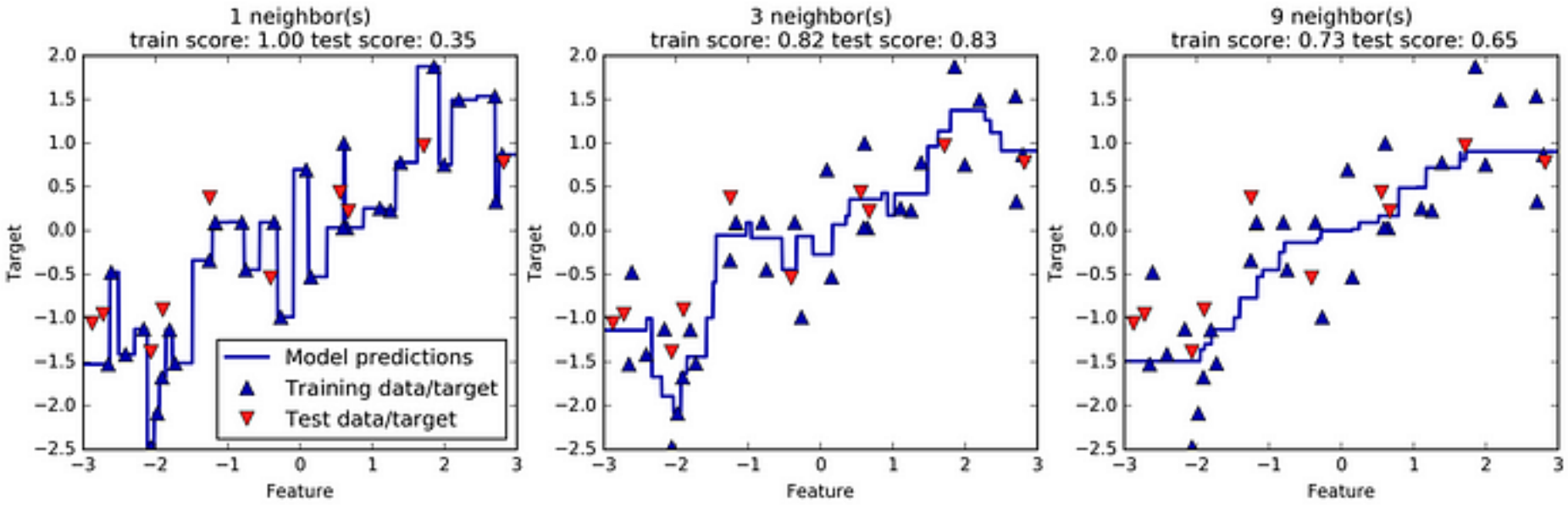
Nearest neighbor regression



"Predictor":
Given new features:
Find nearest example
Return its value

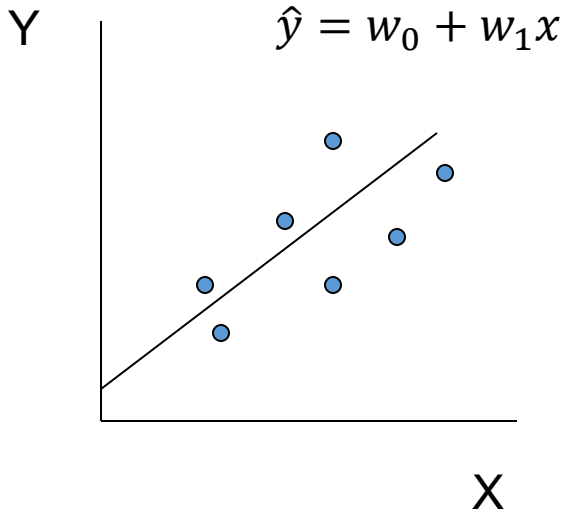
- Find training data $x^{(i)}$ closest to $x^{(new)}$; predict $y^{(i)}$
- Defines an (implict) function f(x)
- "Form" is piecewise constant

K-Nearest neighbor regression



ref : Andreas C.Muller and Sarah Guido. 2017. Introduction to machine learning with pyhton

Linear regression



• Our goal is to estimate w_0, w_1 from a training data of $< x^{(i)}, y^{(i)} >$ pairs

• Optimization goal: minimize squared error (least squares):

$$\text{argmin}_{w_0,w_1} \sum_i (y^{(i)} - w_0 - w_1 x^{(i)})^2$$

• Why least squares?

- minimizes squared distance between measurements and predicted line

- has a nice probabilistic interpretation

- the math is pretty

Solving linear regression

- To optimize – closed form:
- We just take the derivative w.r.t. to w and set to 0:

$$\frac{\partial}{\partial w}\sum_i (y_i - wx_i)^2 = 2\sum_i -x_i(y_i - wx_i) \Rightarrow$$

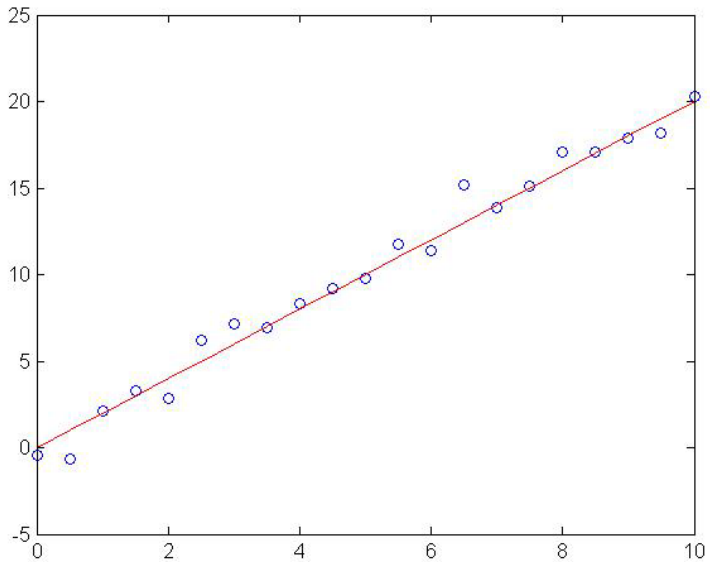
$$2\sum_i x_i(y_i - wx_i) = 0 \quad \Rightarrow 2\sum_i x_i y_i - 2\sum_i wx_i x_i = 0$$

$$\sum_i x_i y_i = \sum_i w x_i^2 \quad \Rightarrow$$

$$w = \frac{\sum_i x_i y_i}{\sum_i x_i^2}$$

Regression example

- Generated: w=2
- Noise: std=1
- Recovered: w=2.03



Multivariate regression

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
x_1	x_2	x_3	x_4	y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178

Multivariate regression

- What if we have several inputs?

  - Stock prices for Yahoo, Microsoft and Ebay for the Google prediction task

- This becomes a multivariate regression problem
- Again, its easy to model:

$$h(x) = w_0 + w_1x_1 + \ldots + w_nx_n$$

Google's stock price

Yahoo's stock price

Microsoft's stock price

Notation

- $h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

n features

- Define "feature" $x_0 = 1$, then
- $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x}$

$$\boldsymbol{\theta} = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \cdots \\ \theta_n \end{pmatrix} \qquad \boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ \cdots \\ x_n \end{pmatrix}$$

Notation

- m examples in the data: $(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \cdots, (\boldsymbol{x}^{(m)}, y^{(m)})$
- Loss function (squared error)

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right)\right)^2$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - \boldsymbol{\theta}^T\boldsymbol{x}^{(i)}\right)^2$$

Notation

Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
x_1	x_2	x_3	x_4	y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178

- Rewrite using matrix form

$$X = \begin{pmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{pmatrix} \qquad \boldsymbol{y} = \begin{pmatrix} 460 \\ 232 \\ 315 \\ 178 \end{pmatrix}$$

$$J(\boldsymbol{\theta}) = \frac{1}{m}\left(\boldsymbol{y^T} - \boldsymbol{\theta}^T X^T\right)\left(\boldsymbol{y^T} - \boldsymbol{\theta}^T X^T\right)^T$$

$$\boldsymbol{y} = \begin{pmatrix} y^{(1)} \\ \cdots \\ y^{(m)} \end{pmatrix} \qquad \boldsymbol{\theta} = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \cdots \\ \theta_n \end{pmatrix} \qquad X = \begin{pmatrix} x_0^{(1)} & \cdots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_0^{(m)} & \cdots & x_n^{(m)} \end{pmatrix}$$

Not all functions can be approximated by a line/hyperplane…

$$h(x) = \theta_0 + \theta_1 {x_1}^2 + \theta_2 {x_2}^2$$

In some cases we would like to use polynomial or other terms based on the input data, are these still linear regression problems?

Yes. As long as the *coefficients* are linear the equation is still a linear regression problem!

Learning/Optimizing Multivariate Least Squares

Approach 1: Normal equation

Solving linear regression

- To optimize – closed form:
- We just take the derivative w.r.t. to $\boldsymbol{\theta}$ and set to 0:

$$\text{argmin}_{\boldsymbol{\theta}} \sum_i (y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)})^2 \Rightarrow \frac{\partial}{\partial \theta_j} \sum_i (y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)})^2 = 0$$

Normal equation for multivariate regression

- To solve for $\boldsymbol{\theta}$ analytically

$$\boldsymbol{\theta} = (X^T X)^{-1} X^T \boldsymbol{y}$$

$$\boldsymbol{y} = \begin{pmatrix} y^{(1)} \\ \cdots \\ y^{(m)} \end{pmatrix} \qquad \boldsymbol{\theta} = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \cdots \\ \theta_n \end{pmatrix} \qquad X = \begin{pmatrix} 1 & \cdots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ 1 & \cdots & x_n^{(m)} \end{pmatrix}$$

Example

Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
x_1	x_2	x_3	x_4	y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178

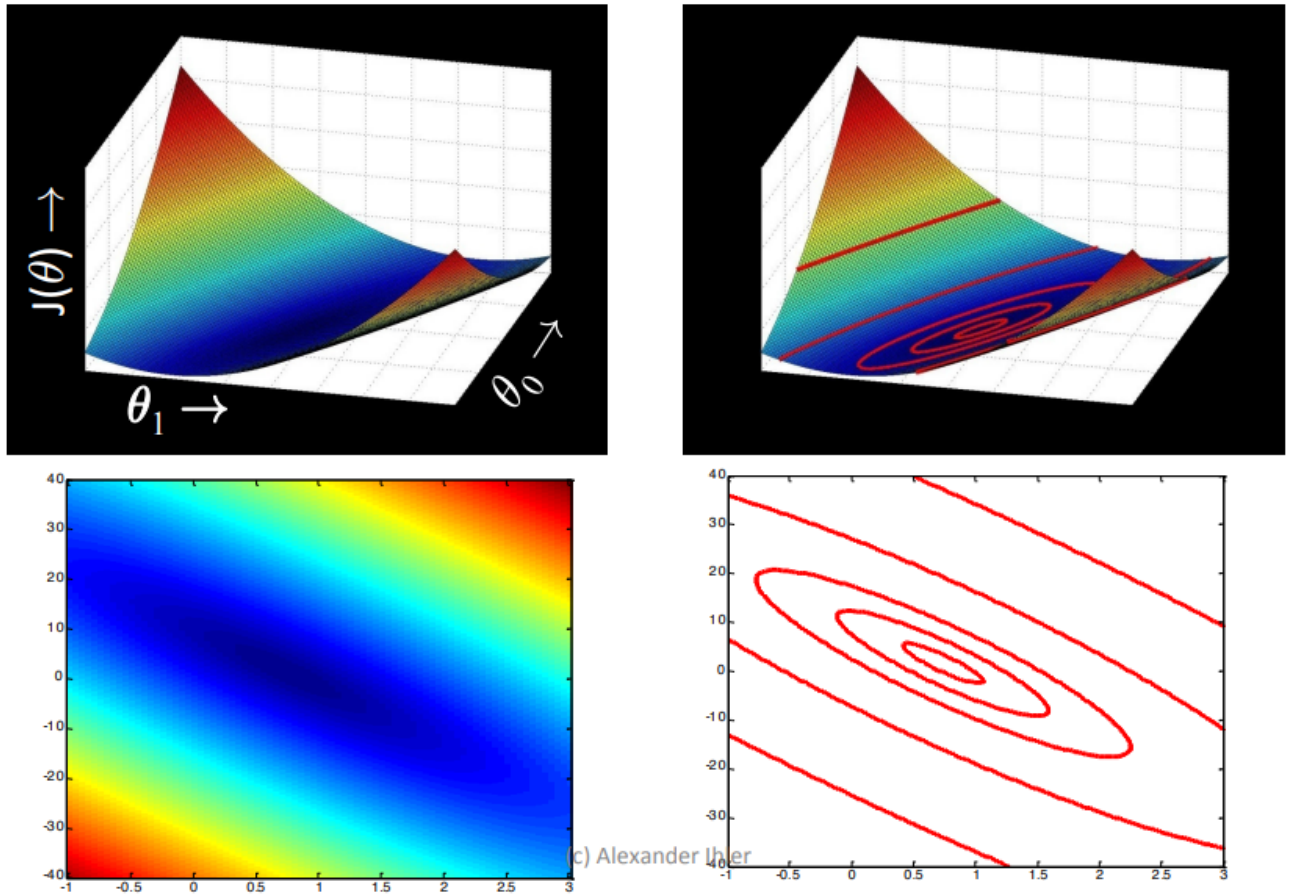
$$X = \begin{pmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{pmatrix} \qquad \boldsymbol{y} = \begin{pmatrix} 460 \\ 232 \\ 315 \\ 178 \end{pmatrix}$$

$$\boldsymbol{\theta} = (X^T X)^{-1} X^T \boldsymbol{y}$$

Learning/Optimizing Multivariate Least Squares

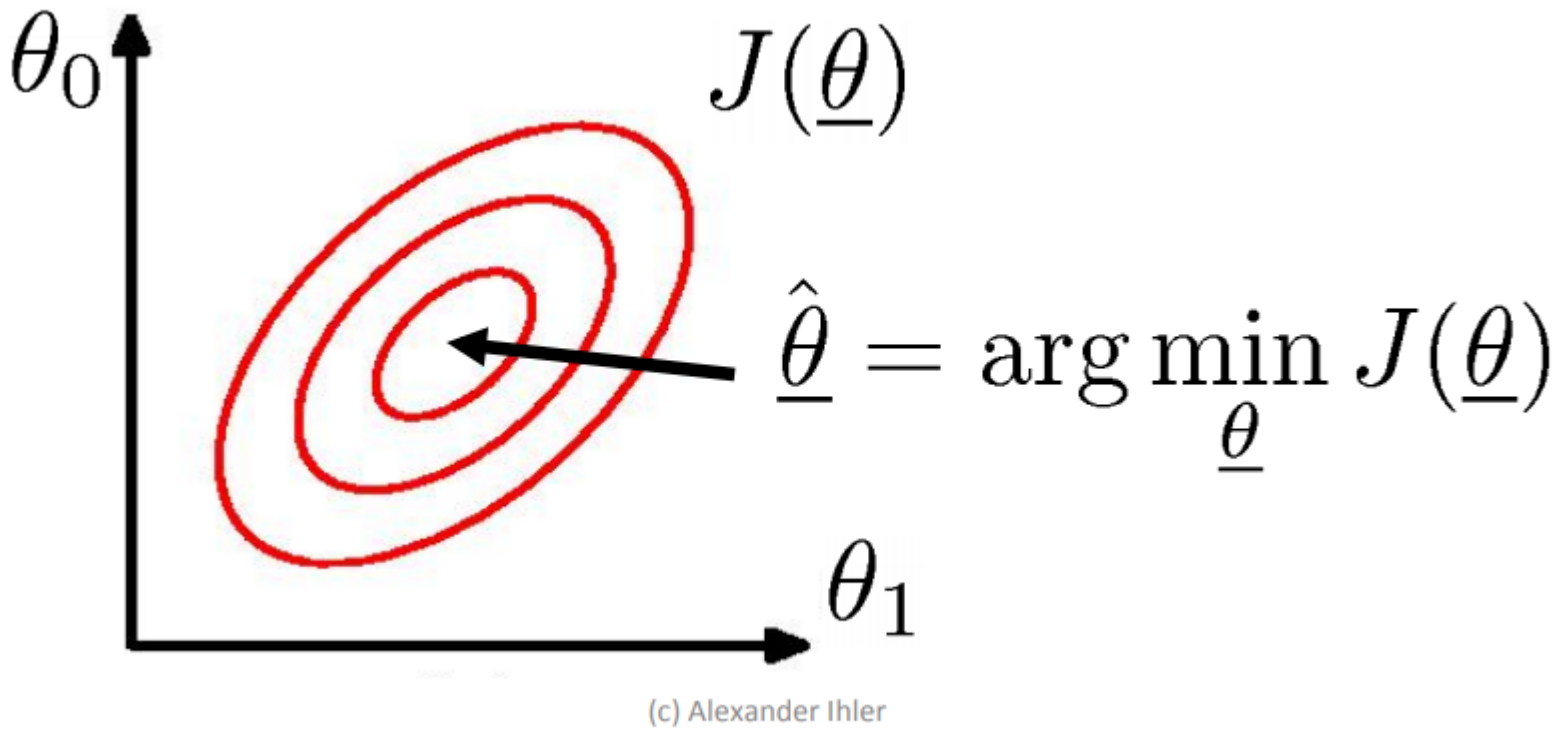
Approach 2: Gradient Descent

Visualizing the loss function

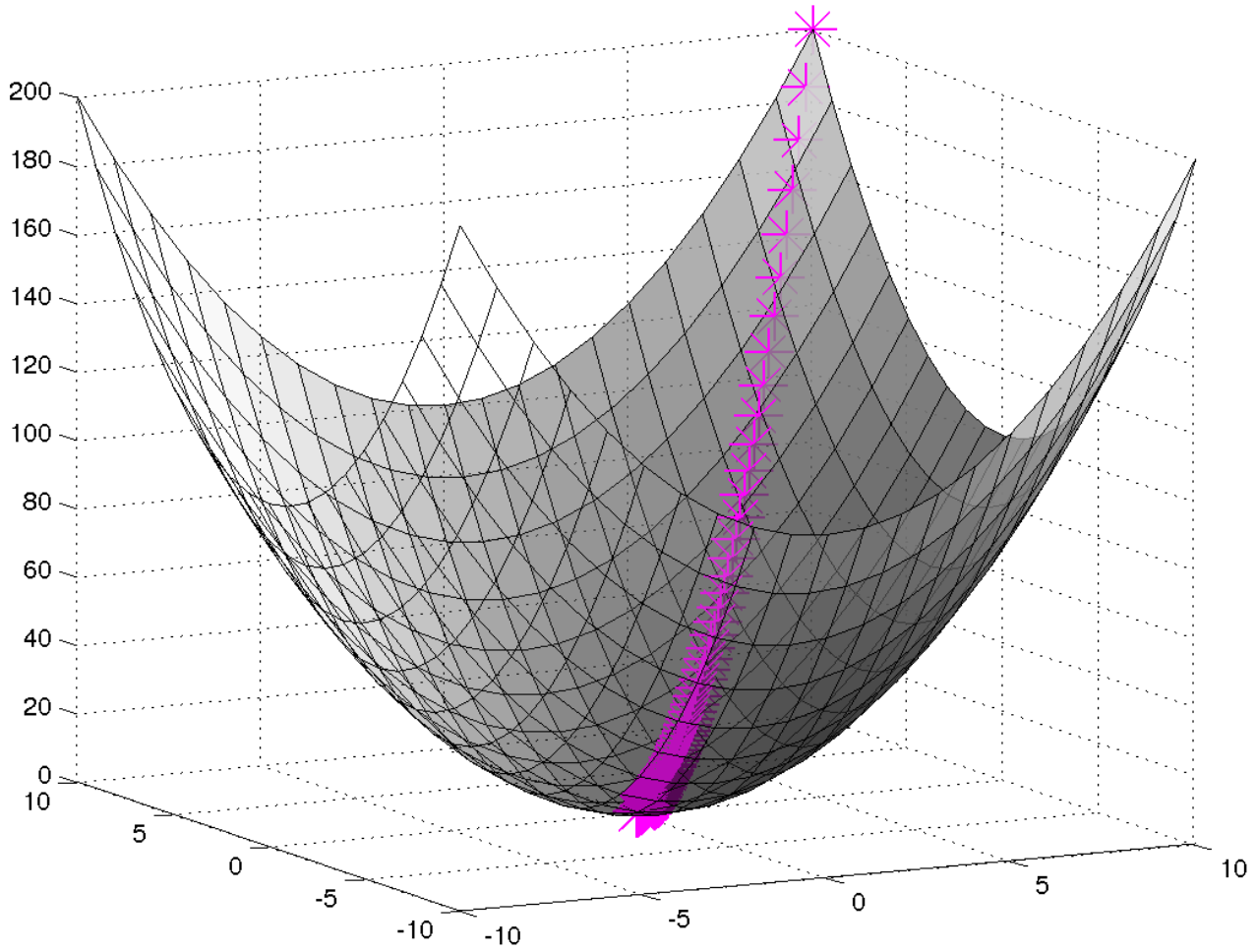


Gradient descent

- Want to find parameters which minimize the loss

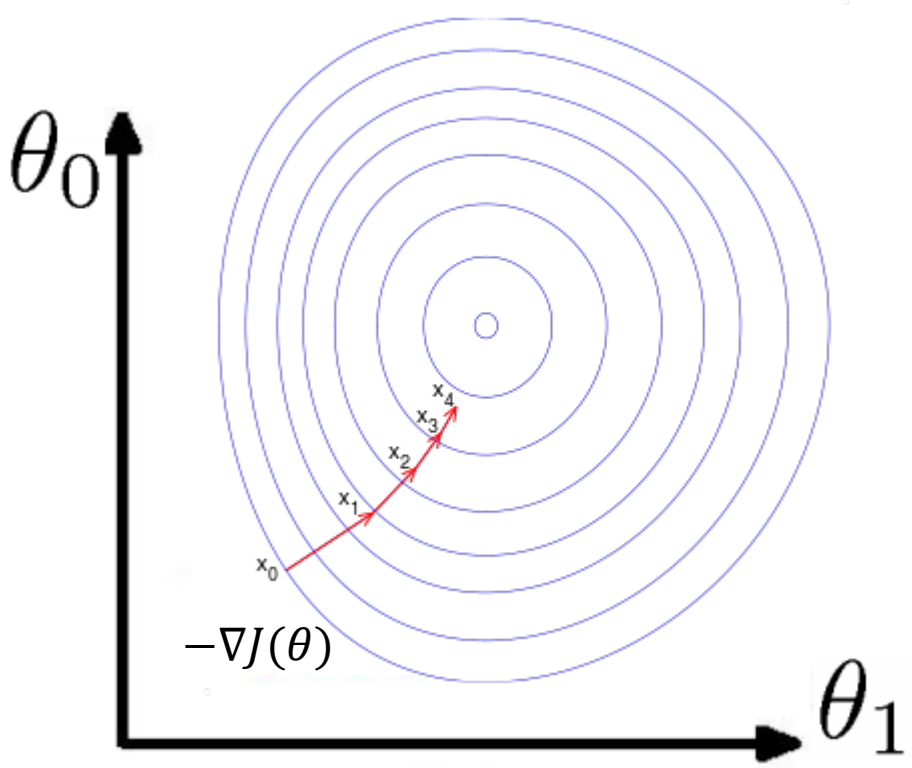


Gradient descent



Gradient descent in more dimensions

- Gradient vector
- $\nabla J(\theta) = \left(\frac{\partial J(\theta)}{\partial \theta_0}, \frac{\partial J(\theta)}{\partial \theta_1}, \dots\right)$
- Indicates direction of steepest ascent
- (negative = steepest descent)



Gradient descent

- Initialization
- Step size
  - Can change as a function of iteration
- Gradient direction
- Stop condition

Initialize $\boldsymbol{\theta}$
Do {
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla J(\boldsymbol{\theta})$
} **while** (stop condition)

Gradient descent for linear regression

- Goal: minimize loss function

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{m}\left(y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)}\right)^2 = \frac{1}{2}\sum_{i=1}^{m}\left(y^{(i)} - \sum_{j=0}^{n}\theta_j x_j^{(i)}\right)^2$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} = \frac{1}{2}\frac{\partial}{\partial \theta_j}\sum_{i=1}^{m}\left(y^{(i)} - \sum_{j=0}^{n}\theta_j x_j^{(i)}\right)^2$$

$$= \sum_{i=1}^{m}\left(y^{(i)} - \sum_{j=0}^{n}\theta_j x_j^{(i)}\right)\frac{\partial}{\partial \theta_j}\left(y^{(i)} - \sum_{j=0}^{n}\theta_j x_j^{(i)}\right)$$

$$= \sum_{i=1}^{m}\left(\sum_{j=0}^{n}\theta_j x_j^{(i)} - y^{(i)}\right)x_j^{(i)}$$

Gradient descent for linear regression

- **Initialize $\boldsymbol{\theta}$**
- **Do** {
- $\theta_j \leftarrow \theta_j - \alpha \sum_{i=1}^{m} \left(\sum_{j=0}^{n} \theta_j x_j^{(i)} - y^{(i)} \right) x_j^{(i)}$
- } **while** (stop condition)

(Simultaneously update θ_j for all j)

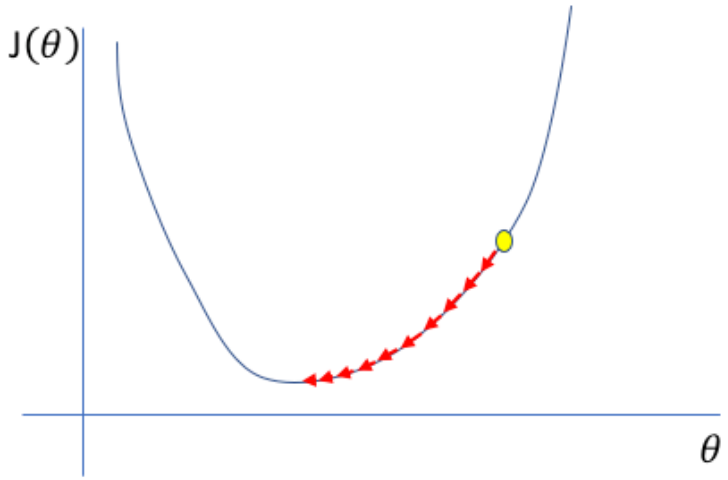
$$\theta'_0 \leftarrow \theta_0 - \alpha \sum_{i=1}^{m} \left(\sum_{j=0}^{n} \theta_j x_j^{(i)} - y^{(i)} \right)$$

$$\theta'_1 \leftarrow \theta_1 - \alpha \sum_{i=1}^{m} \left(\sum_{j=0}^{n} \theta_j x_j^{(i)} - y^{(i)} \right) x_1^{(i)}$$

$$\theta'_2 \leftarrow \theta_2 - \alpha \sum_{i=1}^{m} \left(\sum_{j=0}^{n} \theta_j x_j^{(i)} - y^{(i)} \right) x_2^{(i)}$$

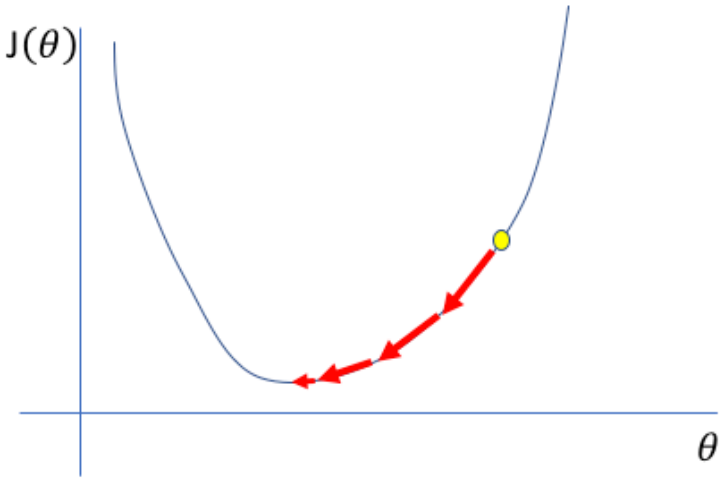
α - learning rate

Too low



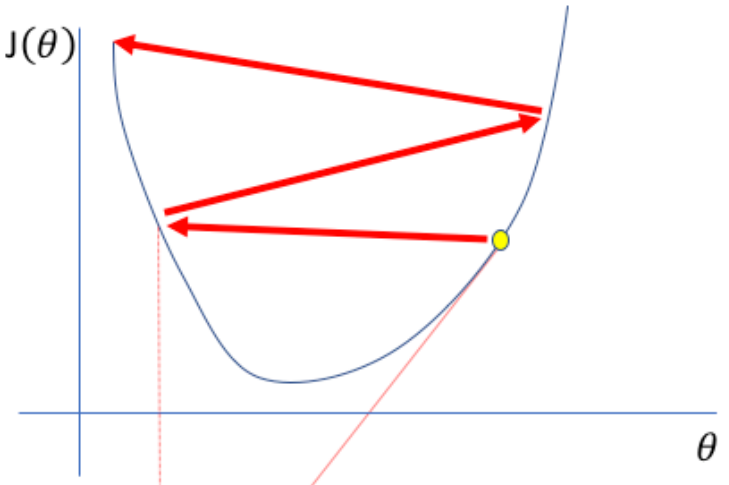
A small learning rate requires many updates before reaching the minimum point

Just right



The optimal learning rate swiftly reaches the minimum point

Too high



Too large of a learning rate causes drastic updates which lead to divergent behaviors

Gradient descent versus normal equation

Gradient descent

- Need to choose α
- Needs many iterations
- Works well even when n is large

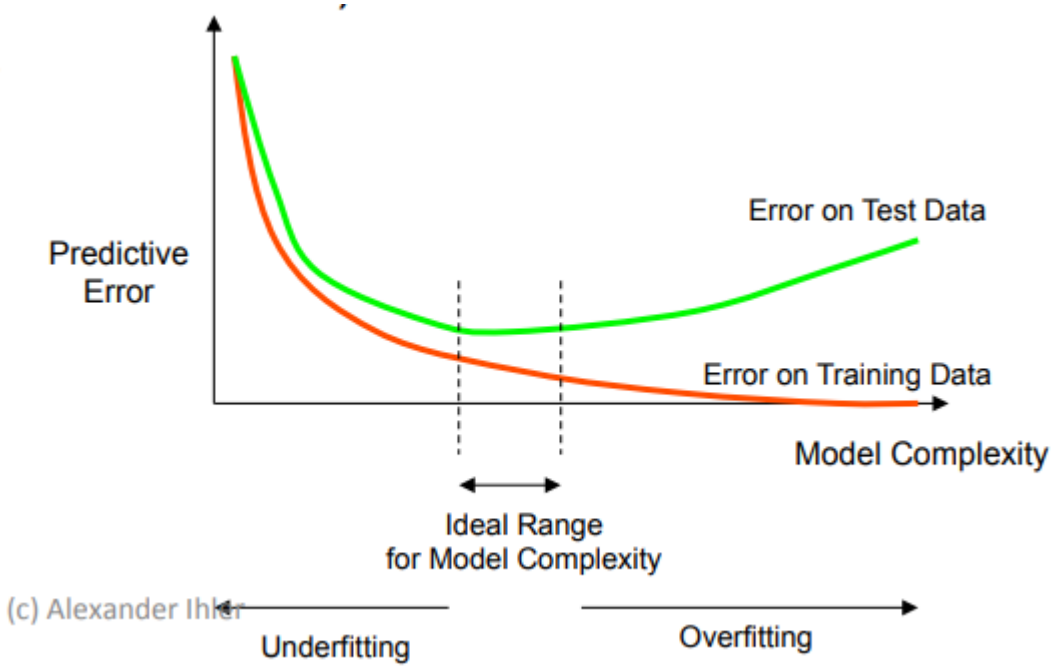
Normal equation

- No need to choose α
- Don’t need to iterate
- Need to compute $(X^T X)^{-1}$
- Slow if n is very large

Regularization

Underfitting and overfitting

- Ways to increase complexity
  - Add features
- Ways to decrease complexity
  - Remove features
  - Regularization



Regularization for linear regression

- Modify loss function to add “preference” for small parameter values

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{m} \left(y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)} \right)^2 + \frac{\lambda}{2} \|\boldsymbol{\theta}\|$$

- Normal equation

$$\boldsymbol{\theta} = (\lambda I + X^T X)^{-1} X^T \boldsymbol{y}$$

Regularization for linear regression

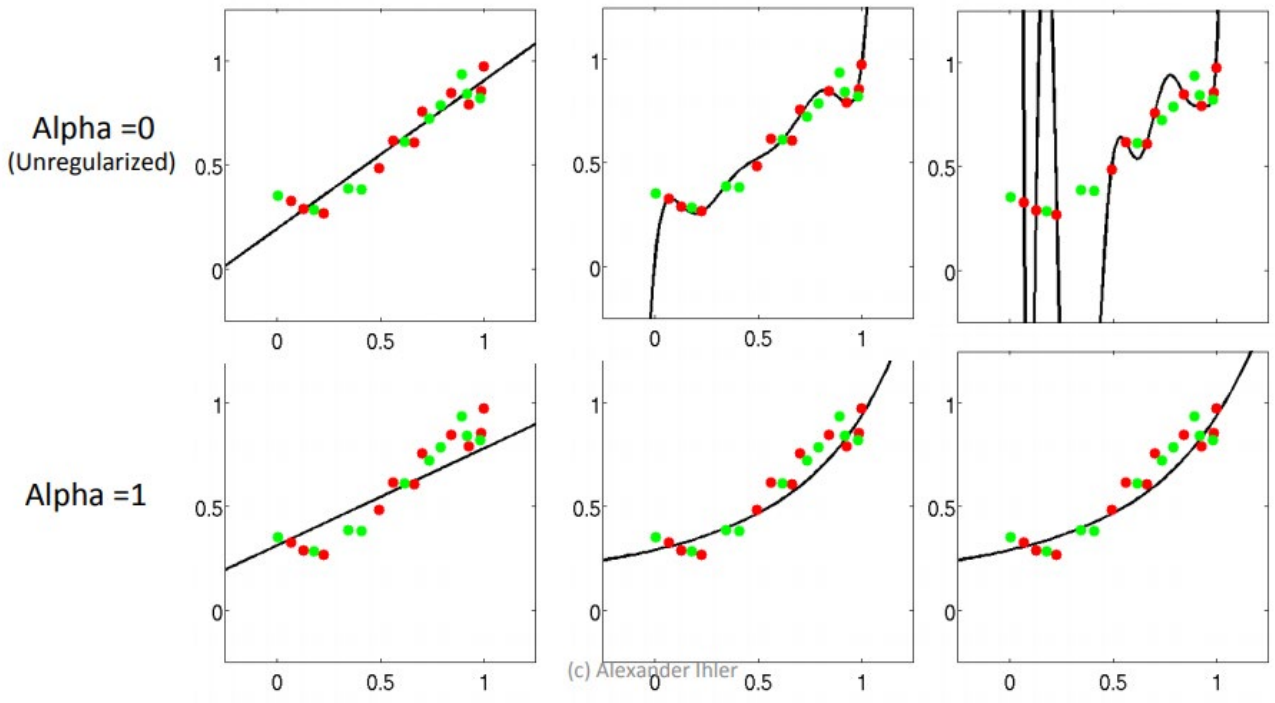
- Regularized gradient descent

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} = \sum_{i=1}^{m} \left(\sum_{j=0}^{n} \theta_j x_j^{(i)} - y^{(i)} \right) x_j^{(i)} + \lambda \theta_j$$

$$\theta_j \leftarrow \theta_j (1 - \alpha\lambda) - \alpha \sum_{i=1}^{m} \left(\sum_{j=0}^{n} \theta_j x_j^{(i)} - y^{(i)} \right) x_j^{(i)}$$

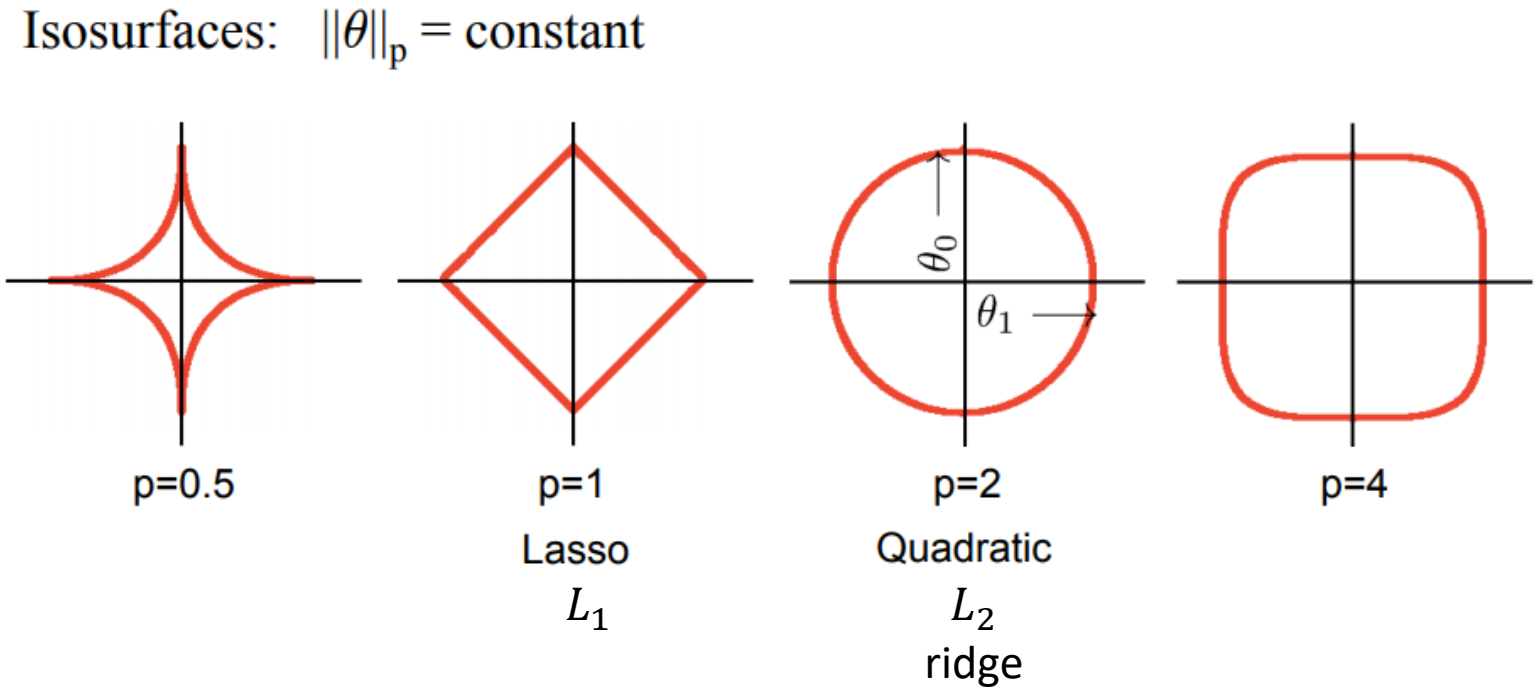
Regularization

- Compare between unreg. & reg. results



Different regularization functions

- More general, L_p regularizer: $\left(\sum_j |\theta_j|^p\right)^{\frac{1}{p}}$



A geometrical view of the lasso compared with a penalty on the squared weights

- Lasso tends to generate sparser solutions than a quadratic regularizer