CS 460200

Introduction to Machine Learning

Regression

Instructor: Po-Chih Kuo

Roadmap

- Introduction and Basic Concepts
- Regression
- Bayesian Classifiers
- Decision Trees
- KNN
- Linear Classifier
- Neural Networks
- Deep learning
- Convolutional Neural Networks
  - Autoencoder
  - Adversarial
  - Transfer learning
  - …
- RNN/Transformer
- Reinforcement Learning
- Model Selection and Evaluation
- Clustering
- Data Exploration & Dimensionality reduction

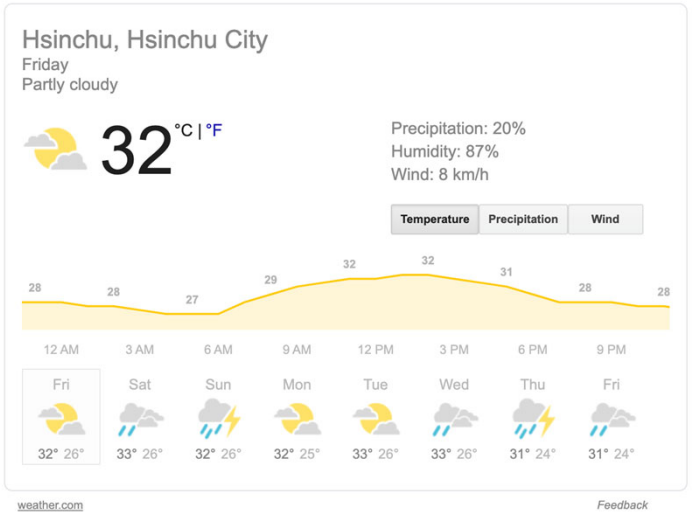
Supervised learning

This week

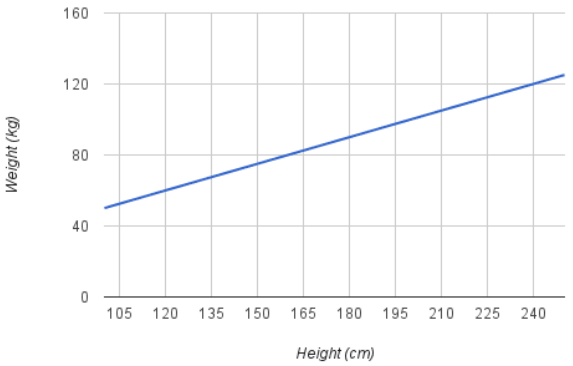
- Linear regression
  - As matrix inversion (Ordinary Least Squares)
  - As optimization: Gradient descent
    - Batch gradient decent
    - Stochastic gradient descent
- Overfitting and regularization
- Autoregression
- Logistic regression (more like classification)

Regression examples

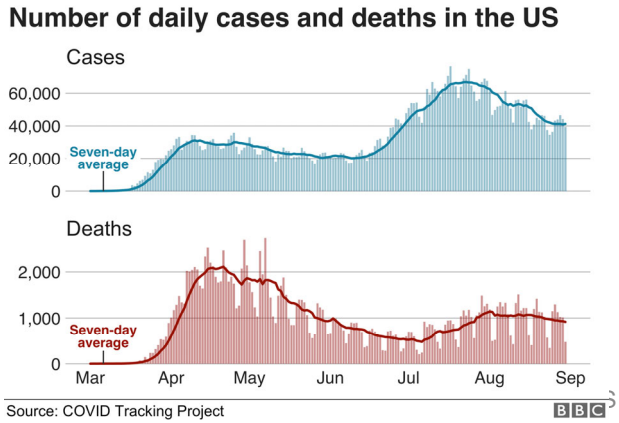
Weather



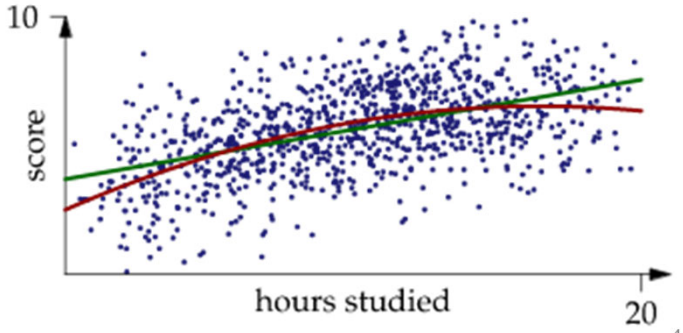
Weight vs. Height



Disease case



Score vs. Hours studied



Example: House price in the U.S

	x_i	y_i
i	sqft	price
1	5650	221900
2	7242	538000
3	10000	180000
4	5000	604000
5	8080	510000
6	101930	1230000

Training data

$\{(x_i, y_i)\}_{i=1}^{n}$

Machine learning algorithm

Prediction function

$f(x_i)$

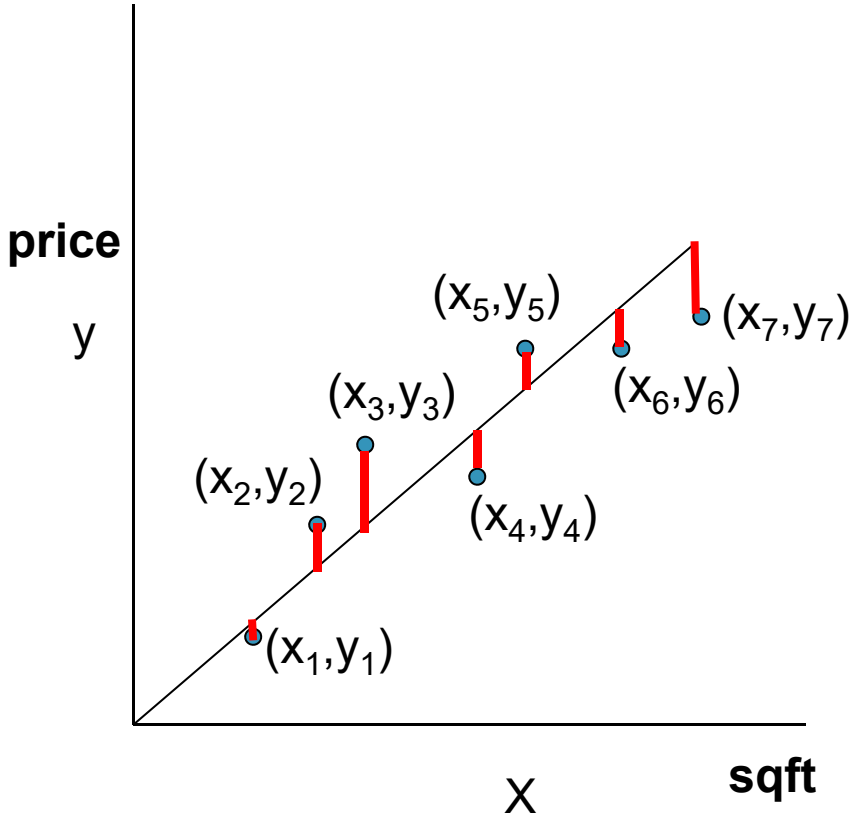
Linear regression

- 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:

What we are trying to predict

$$y = f(x) = wx + \varepsilon$$

where w is a parameter and ε represents measurement noise



Linear regression

- Our goal is to estimate *w* from a training data of $<x_i,y_i>$ pairs
- Optimization goal: minimize squared error (least squares):

$$\text{argmin}_w \sum_i (y_i - wx_i)^2$$

L(w)

- Why least squares?
  - Straightforward! minimizes squared distance between measurements and predicted line
  - Has probabilistic interpretation: equivalent to maximum likelihood estimation. (http://people.math.gatech.edu/~ecroot/3225/maximum_likelihood.pdf)
  - The math is pretty

Solving linear regression

- To optimize and get a closed-form solution
- Let's take the derivative w.r.t. to w and set to 0:

$$\frac{\partial}{\partial w}\sum_i (y_i - wx_i)^2 = 0$$

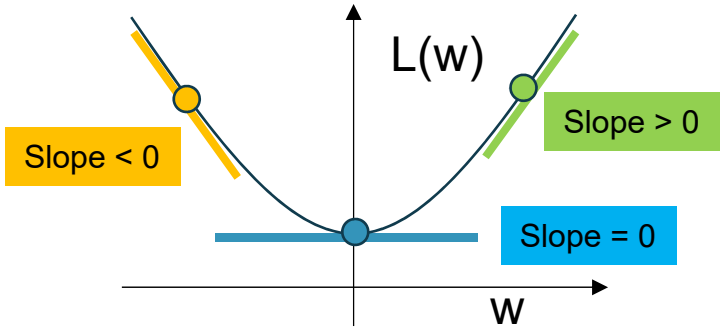
$$2\sum_i -x_i(y_i - wx_i) = 0$$

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

$$2\sum_i x_i y_i - 2\sum_i wx_i x_i = 0$$

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

$$L(w) = \sum_i (y_i - wx_i)^2$$



$$w = \frac{\sum_i x_i y_i}{\sum_i {x_i}^2} = \frac{COVAR(X,Y)}{VAR(X)}$$

if mean(X)=mean(Y)=0

Variance & Covariance

- Variance:
  - Measure of the deviation from the mean for points in one dimension
- Covariance:
  - Measure of how much each of the dimensions vary from the mean with respect to each other

$$\mathbf{VAR(X)} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - E(X))^2$$

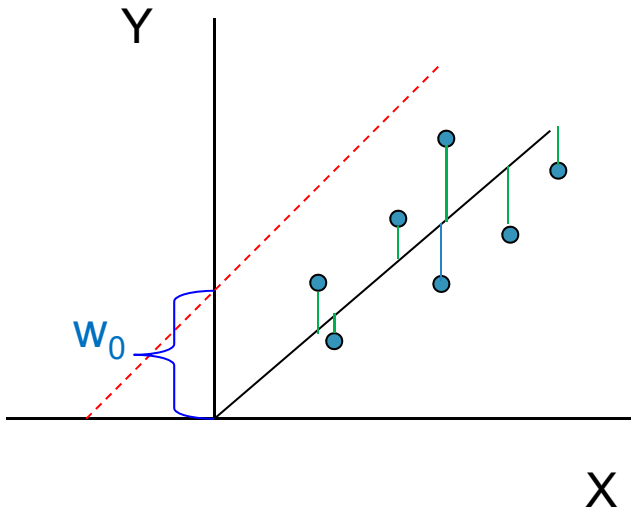
$$\mathbf{COV(X,\ Y)} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - E(X))(y_i - E(Y))$$

Intercept/Bias term

- So far, we assumed that the line passes through the origin. What if the line does not?
- Simply change the model to:

$$y = w_0 + w_1 x + \varepsilon$$

- We can use least squares to determine w_0 , w_1



$$w_0 = \frac{\sum y_i - w_1 \sum x_i}{n} \qquad w_1 = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{n \sum x_i^2 - (\sum x_i)^2}$$

Error vs Residual

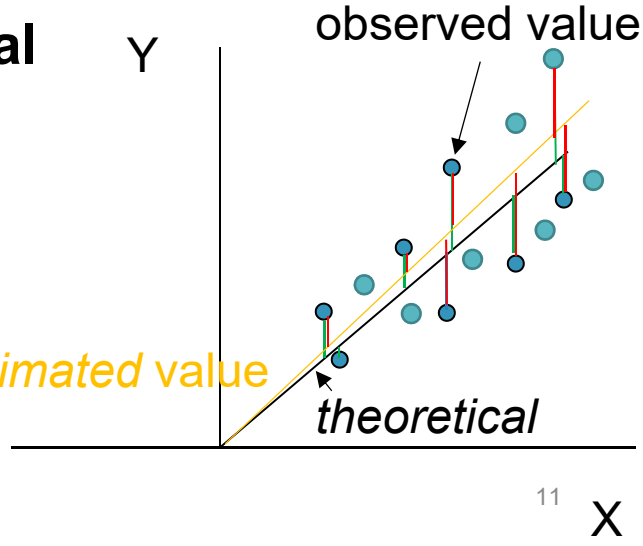
- The **error** of an observed value is the deviation of the observed value from the (unobservable) *true(theoretical)* value of a quantity of interest (for example, a population mean)
- The **residual** of an observed value is the difference between the observed value and the *estimated* value of the quantity of interest (for example, a sample mean).
- The error term accounts for the variation in the dependent variable(Y) that the independent variables(X) do not explain.
- Least square error -> Least square **residual**

$$y = wx + \varepsilon$$

$$\hat{y} = wx$$

$$\text{argmin}_w \sum_i (y_i - \hat{y}_i)^2$$

$$|y - \hat{y}|$$



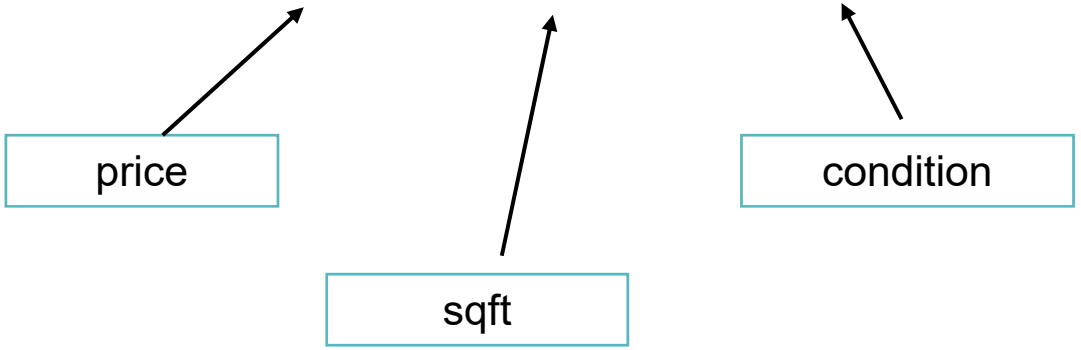
- What if we have several inputs?

	X_1	X_2	X_3	X_4	Y
I	bedrooms	bathrooms	condition	sqft	price
1	3	1	3	5650	221900
2	3	2.25	3	7242	538000
3	2	1	3	10000	180000
4	4	3	5	5000	604000
5	3	2	3	8080	510000
6	4	4.5	3	101930	1230000

Multiple regression

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

$$y = w_0 + w_1 x_1 + \cdots + w_k x_k + \varepsilon$$



Non-linear basis functions

- So far we only used the observed values x_1,x_2,…
- However, linear regression can be applied in the same way to **functions** of these values
  - Eg: to add a term w x_1x_2 add a new variable $z=x_1x_2$ so each example becomes: x_1, x_2, …. z
- As long as these functions can be directly computed from the observed values, the problem remains a multiple linear regression problem:

$$y = w_0 + w_1 x_1^{(2)} + \ldots + w_k x_k^{(2)} + \varepsilon$$

Non-Linear basis functions

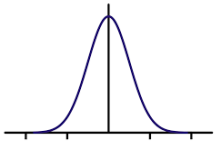
Similar idea in SVM

- What type of functions can we use?
- A few common examples:
  - Polynomial:

$$\phi_j(x) = x^j$$

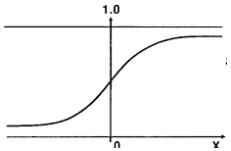
  - Gaussian:

$$\phi_j(x) = \exp(-\frac{(x-\mu_j)^2}{2s^2})$$



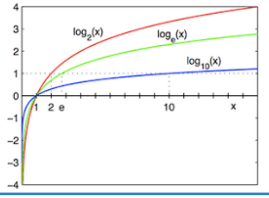
  - Sigmoid:

$$\phi_j(x) = \sigma(\frac{x-\mu_j}{s}) \qquad \sigma(a) = \frac{1}{1+\exp(-a)}$$



  - Logs:

$$\phi(x) = log(x+1)$$



Any function of the input values can be used. The solution for the parameters of the regression remains the same.

General linear regression problem

- Using our new notations for the basis function linear regression can be written as

$$y = \sum_{j=0}^{n} w_j \phi_j(x)$$

- Where $\phi_j(x)$ can be either x_j for multiple regression or one of the non-linear basis functions we defined
- $\phi_0(x)=1$ for the intercept term

Learning/Optimizing Multivariate Least Squares

Approach 1: Matrix Inversion

Approach 2: Gradient Descent

OLS (Ordinary Least Squares Solution)

OLS Assumption: The error term has a population mean of zero

$$\text{predict with}: \hat{y}^i = \sum_j^n w_j \phi_j(\mathbf{x}^i)$$

Goal: minimize the following loss function:

$$J_{\mathbf{X},\mathbf{y}}(\mathbf{w}) = \sum_i \left(y^i - \hat{y}^i\right)^2 = \sum_i \left(y^i - \sum_j w_j \phi_j(\mathbf{x}^i)\right)^2$$

sum over *n* examples

sum over *k+1* basis vectors

OLS (Ordinary Least Squares Solution)

Goal: minimize the following loss function:

$$J_{\mathbf{X},\mathbf{y}}(\mathbf{w}) = \sum_i \left(y^i - \hat{y}^i\right)^2 = \sum_i \left(y^i - \sum_j w_j \phi_j(\mathbf{x}^i)\right)^2$$

$$\frac{\partial}{\partial w_j} J(\mathbf{w}) = \frac{\partial}{\partial w_j} \sum_i \left(y^i - \hat{y}^i\right)^2$$

$$= -2 \sum_i \left(y^i - \hat{y}^i\right) \frac{\partial}{\partial w_j} \hat{y}^i$$

$$= -2 \sum_i \left(y^i - \hat{y}^i\right) \frac{\partial}{\partial w_j} \sum_j w_j \phi_j(\mathbf{x}^i)$$

$$= -2 \sum_i \left(y^i - \hat{y}^i\right) \phi_j(\mathbf{x}^i)$$

We see

$$J_{\mathbf{X},\mathbf{y}}(\mathbf{w}) = \sum_i \left(y^i - \hat{y}^i\right)^2 = \sum_i \left(y^i - \sum_j w_j \phi_j(\mathbf{x}^i)\right)^2$$

$$\frac{\partial}{\partial w_j} J(\mathbf{w}) = -2 \sum_i \left(y^i - \hat{y}^i\right) \phi_j(\mathbf{x}^i)$$

Now, let's consider ***i***

Notation:

$$\mathbf{y} = \begin{pmatrix} y^1 \\ \cdots \\ \cdots \\ y^n \end{pmatrix} \quad \Phi = \begin{pmatrix} \phi_0(\mathbf{x}^1) & \phi_1(\mathbf{x}^1) & \cdots & \phi_k(\mathbf{x}^1) \\ \phi_0(\mathbf{x}^2) & \phi_1(\mathbf{x}^2) & \cdots & \phi_k(\mathbf{x}^2) \\ \vdots & \vdots & \cdots & \vdots \\ \phi_0(\mathbf{x}^n) & \phi_1(\mathbf{x}^n) & \cdots & \phi_k(\mathbf{x}^n) \end{pmatrix} \quad \mathbf{w} = \begin{pmatrix} w_0 \\ .. \\ w_k \end{pmatrix}$$

k+1 basis vectors

n examples

k+1 basis vectors

$$\mathbf{y} = \begin{pmatrix} y^1 \\ \cdots \\ \cdots \\ y^n \end{pmatrix} \quad \Phi = \begin{pmatrix} \phi_0(\mathbf{x}^1) & \phi_1(\mathbf{x}^1) & \cdots & \phi_k(\mathbf{x}^1) \\ \phi_0(\mathbf{x}^2) & \phi_1(\mathbf{x}^2) & \cdots & \phi_k(\mathbf{x}^2) \\ \vdots & \vdots & \cdots & \vdots \\ \phi_0(\mathbf{x}^n) & \phi_1(\mathbf{x}^n) & \cdots & \phi_k(\mathbf{x}^n) \end{pmatrix} = \begin{pmatrix} \vec{\phi}^1 \\ \cdots \\ \cdots \\ \vec{\phi}^n \end{pmatrix} \quad \mathbf{w} = \begin{pmatrix} w_0 \\ .. \\ w_k \end{pmatrix}$$

n examples

$$\frac{\partial}{\partial w_j} J(\boldsymbol{w}) = \begin{pmatrix} \frac{\partial}{\partial w_0} J(\boldsymbol{w}) = 2 \sum_i (y^i - \widehat{y^i}) \emptyset_0(\boldsymbol{x}^i) \\ \cdots \\ \frac{\partial}{\partial w_k} J(\boldsymbol{w}) = 2 \sum_i (y^i - \widehat{y^i}) \emptyset_k(\boldsymbol{x}^i) \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial w_0} J(\boldsymbol{w}) = 2 \sum_i (y^i \emptyset_0^i - \hat{y}^i \emptyset_0^i) \\ \cdots \\ \frac{\partial}{\partial w_k} J(\boldsymbol{w}) = 2 \sum_i (y^i \emptyset_k^i - \hat{y}^i \emptyset_k^i) \end{pmatrix}$$

notation: $\phi_j^i \equiv \phi_j(\mathbf{x}^i)$

$$\text{recall } \hat{y}^i = \sum_j^n w_j \phi_j^i$$

$$= \vec{\phi}^i \mathbf{w}$$

$$\begin{pmatrix} \frac{\partial}{\partial w_0} J(\boldsymbol{w}) = 2 \sum_i (y^i \phi_0^i - \hat{y}^i \phi_0^i) \\ \cdots \\ \frac{\partial}{\partial w_k} J(\boldsymbol{w}) = 2 \sum_i (y^i \phi_k^i - \hat{y}^i \phi_k^i) \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial w_0} J(\boldsymbol{w}) = 2 \sum_i (y^i \phi_0^i - \phi^i \mathbf{w} \phi_0^i) \\ \cdots \\ \frac{\partial}{\partial w_k} J(\boldsymbol{w}) = 2 \sum_i (y^i \phi_k^i - \phi^i \mathbf{w} \phi_k^i) \end{pmatrix}$$

$$= \begin{pmatrix} \frac{\partial}{\partial w_0} J(\boldsymbol{w}) = 2 \sum_i (\phi_0^i y^i - \phi_0^i \phi^i \mathbf{w}) \\ \cdots \\ \frac{\partial}{\partial w_k} J(\boldsymbol{w}) = 2 \sum_i (\phi_k^i y^i - \phi_k^i \phi^i \mathbf{w}) \end{pmatrix}$$

n examples

$$\Phi^T = \begin{pmatrix} \phi_0(\mathbf{x}^1) & \phi_0(\mathbf{x}^2) & \cdots & \phi_0(\mathbf{x}^n) \\ \vdots & \vdots & \cdots & \vdots \\ \phi_k(\mathbf{x}^1) & \phi_k(\mathbf{x}^2) & \cdots & \phi_k(\mathbf{x}^n) \end{pmatrix} \quad k+1 \qquad \mathbf{y} = \begin{pmatrix} y^1 \\ \cdots \\ \cdots \\ \cdots \\ y^n \end{pmatrix}$$

$$\begin{pmatrix} \frac{\partial}{\partial w_0} J(\boldsymbol{w}) = 2\sum_i (\phi_0^i y^i - \phi_0^i \phi^i \mathbf{w}) \\ \cdots \\ \frac{\partial}{\partial w_k} J(\boldsymbol{w}) = 2\sum_i (\phi_k^i y^i - \phi_k^i \phi^i \mathbf{w}) \end{pmatrix} = 2\Phi^T \mathbf{y} - \ldots$$

n examples

k+1 basis vectors

$$\begin{pmatrix} \phi_0(\mathbf{x}^1) & \dots & \cdots & \phi_0(\mathbf{x}^n) \\ \phi_1(\mathbf{x}^1) & & & \phi_1(\mathbf{x}^n) \\ \vdots & \vdots & \cdots & \vdots \\ \phi_k(\mathbf{x}^1) & & \cdots & \phi_k(\mathbf{x}^n) \end{pmatrix} \quad \begin{pmatrix} \phi_0(\mathbf{x}^1) & \cdots & \phi_k(\mathbf{x}^1) \\ \phi_0(\mathbf{x}^2) & \cdots & \phi_k(\mathbf{x}^2) \\ \vdots & \vdots & \vdots \\ \phi_0(\mathbf{x}^n) & \cdots & \phi_k(\mathbf{x}^n) \end{pmatrix} \quad \begin{pmatrix} w_0 \\ .. \\ w_k \end{pmatrix}$$

$$\begin{pmatrix} \frac{\partial}{\partial w_0} J(\boldsymbol{w}) = 2\sum_i (\phi_0^i y^i - \phi_0^i \phi^i \mathbf{w}) \\ \cdots \\ \frac{\partial}{\partial w_k} J(\boldsymbol{w}) = 2\sum_i (\phi_k^i y^i - \phi_k^i \phi^i \mathbf{w}) \end{pmatrix} = \ldots - 2\Phi^T \Phi \mathbf{w}$$

k+1 basis vectors

$$\mathbf{w} = \begin{pmatrix} w_0 \\ .. \\ w_k \end{pmatrix} \qquad \mathbf{\Phi} = \begin{pmatrix} \phi_0(\mathbf{x}^1) & \phi_1(\mathbf{x}^1) & \cdots & \phi_k(\mathbf{x}^1) \\ \phi_0(\mathbf{x}^2) & \phi_1(\mathbf{x}^2) & \cdots & \phi_k(\mathbf{x}^2) \\ \vdots & \vdots & \cdots & \vdots \\ \phi_0(\mathbf{x}^n) & \phi_1(\mathbf{x}^n) & \cdots & \phi_k(\mathbf{x}^n) \end{pmatrix} \quad \text{n examples} \qquad \mathbf{y} = \begin{pmatrix} y^1 \\ \cdots \\ \cdots \\ y^n \end{pmatrix}$$

$$\begin{pmatrix} \frac{\partial}{\partial w_0} J(\boldsymbol{w}) = 2\sum_i (\phi_0^i y^i - \phi_0^i \phi^i \mathbf{w}) \\ \cdots \\ \frac{\partial}{\partial w_k} J(\boldsymbol{w}) = 2\sum_i (\phi_k^i y^i - \phi_k^i \phi^i \mathbf{w}) \end{pmatrix} = 2\Phi^T \mathbf{y} - 2\Phi^T \Phi \mathbf{w} \quad = 0$$

$$\mathbf{w} = \left(\Phi^T \Phi\right)^{-1} \Phi^T \mathbf{y}$$

OLS for general linear regression

$$\mathrm{w} = \frac{\sum_i x_i y_i}{\sum_i x_i^2} = \frac{COVAR(X,Y)}{VAR(X)}$$

$$J(\mathrm{w}) = \sum_i (y^i - \mathrm{w}^{\mathrm{T}} \phi(x^i))^2$$

Deriving w we get: $\mathrm{w} = (\Phi^T \Phi)^{-1} \Phi^T \mathrm{y}$

k+1 entries vector

n by k+1 matrix

n entries vector

This solution is also known as '**pseudo inverse**'

Learning/Optimizing Multivariate Least Squares

Approach 1: Matrix Inversion

Approach 2: Gradient Descent

Gradient descent for linear regression

$$\text{predict with}: \hat{y}^i = \sum_j^n w_j \phi_j(\mathbf{x}^i)$$

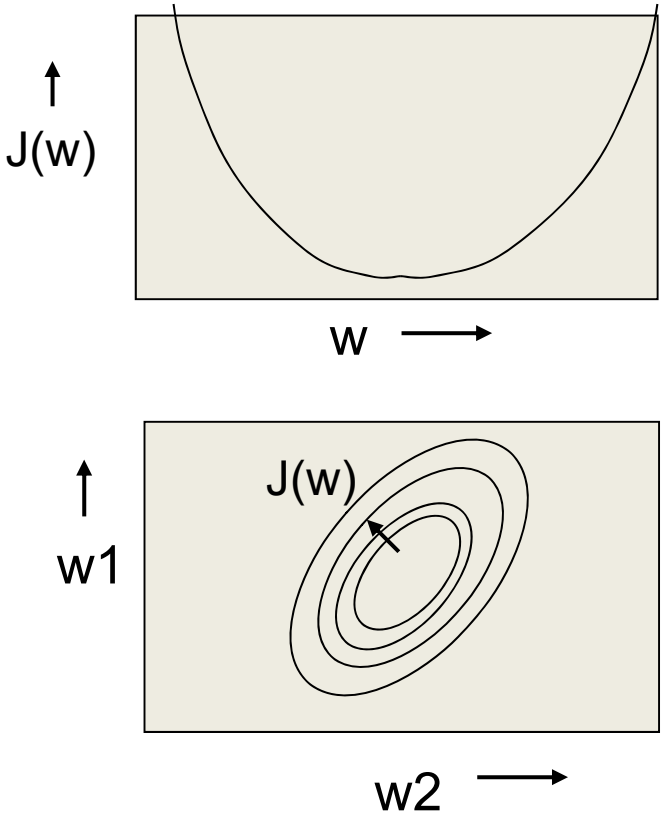
Goal: minimize the following loss function:

$$J_{\mathbf{X},\mathbf{y}}(\mathbf{w}) = \sum_i \left(y^i - \hat{y}^i\right)^2 = \sum_i \left(y^i - \sum_j w_j \phi_j(\mathbf{x}^i)\right)^2$$

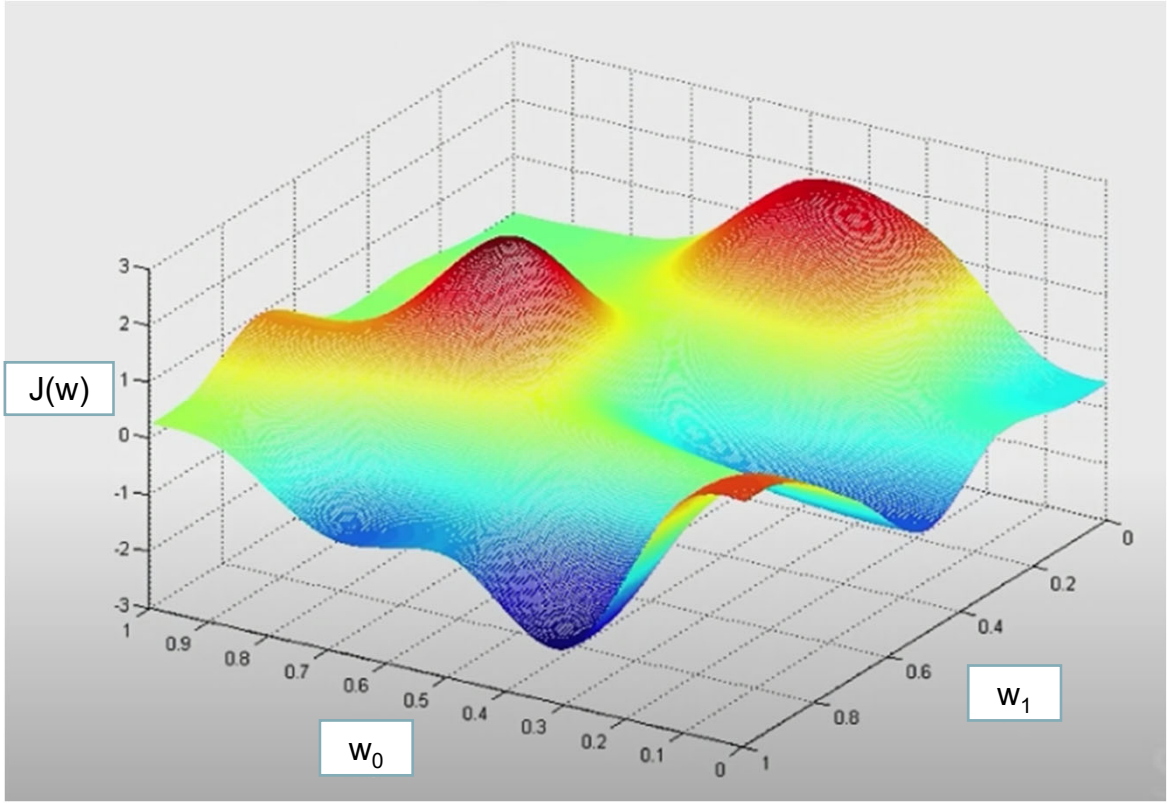
sum over *n* examples

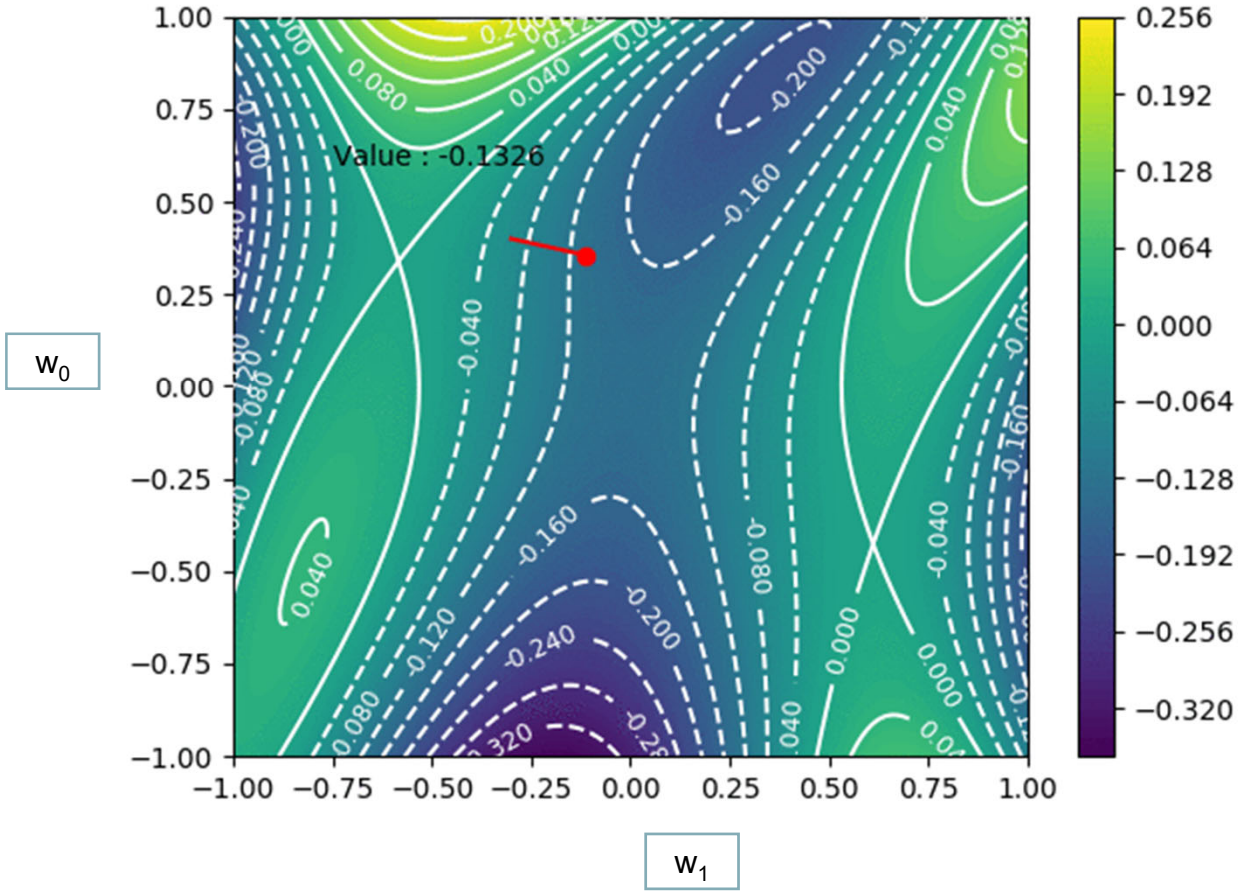
sum over *k+1* basis vectors

Gradient descent



Gradient descent





(https://github.com/Shathra/gradient-descent-demonstration)

Gradient descent for linear regression

Goal: minimize the following loss function:

$$J_{\mathbf{X},\mathbf{y}}(\mathbf{w}) = \sum_i \left(y^i - \hat{y}^i\right)^2 = \sum_i \left(y^i - \sum_j w_j \phi_j(\mathbf{x}^i)\right)^2$$

$$\frac{\partial}{\partial w_j} J(\mathbf{w}) = \frac{\partial}{\partial w_j} \sum_i \left(y^i - \hat{y}^i\right)^2$$

$$= -2 \sum_i \left(y^i - \hat{y}^i\right) \frac{\partial}{\partial w_j} \hat{y}^i$$

$$= -2 \sum_i \left(y^i - \hat{y}^i\right) \frac{\partial}{\partial w_j} \sum_j w_j \phi_j(\mathbf{x}^i)$$

$$= -2 \sum_i \left(y^i - \hat{y}^i\right) \phi_j(\mathbf{x}^i)$$

Gradient descent for linear regression

Learning algorithm:

(1) Initialize weights **w=0**
(2) For t=1,… until convergence:
(2-1) Predict for each example $\boldsymbol{x}^i$ using **w:** $\hat{y}^i = \sum_{j=0}^{k} w_j \phi_j(\mathbf{x}^i)$

(2-2) Compute gradient of loss: $\frac{\partial}{\partial w_j} J(\mathbf{w}) = -2 \sum_i \left(y^i - \hat{y}^i\right) \phi_j(\mathbf{x}^i)$
*This is a vector **g**

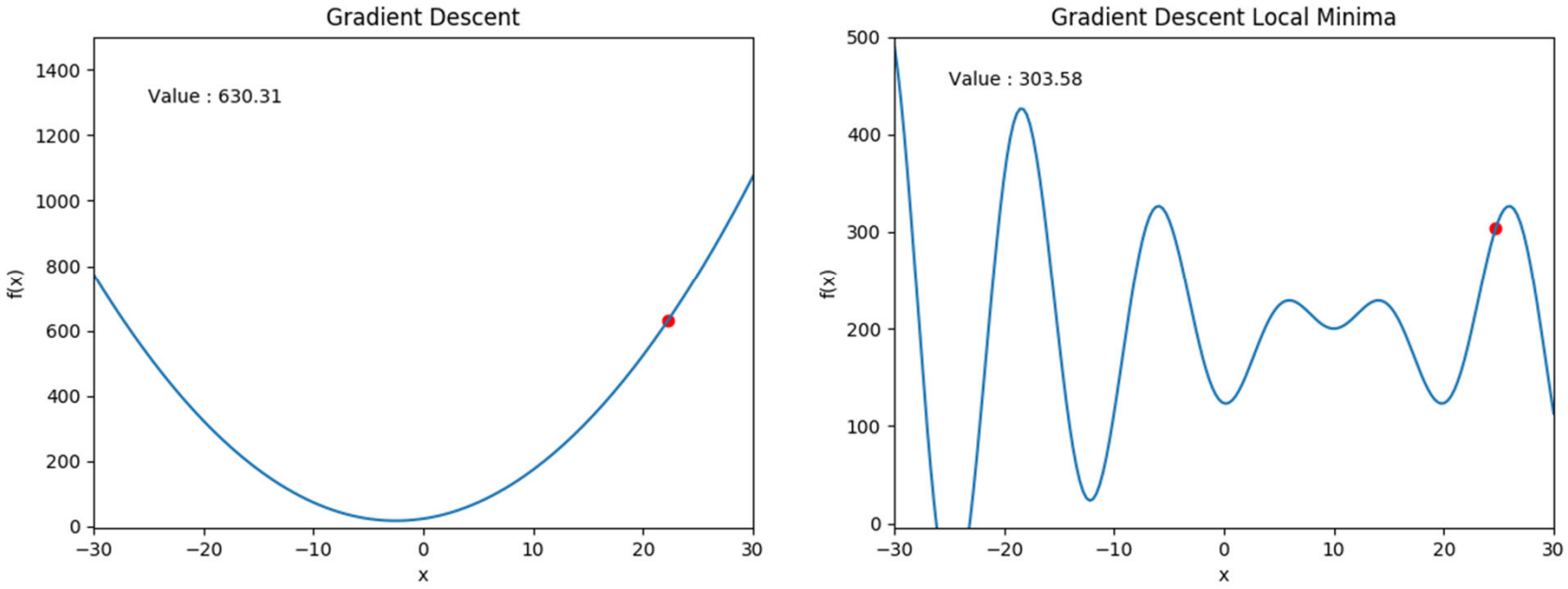
(2-3) Update: **w = w** – α**g,** α is the learning rate

$$\mathrm{w} = \mathrm{w} - \alpha \nabla J(w)$$

(bowl shaped)

Linear regression is a convex optimization problem

gradient descent can reach a *global* optimum

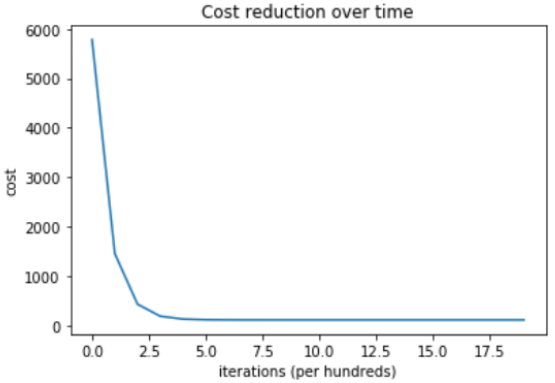


(https://github.com/Shathra/gradient-descent-demonstration)

Non-convex optimization

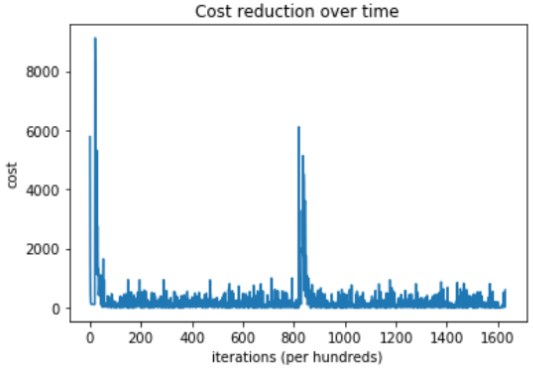
- NP-hard in general
- Deep learning

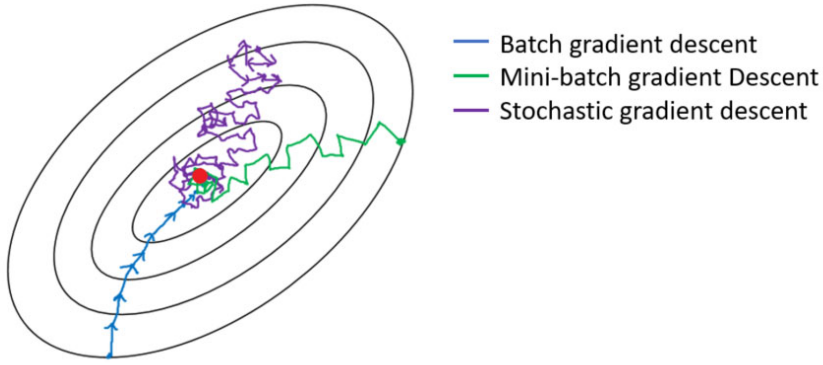
Stochastic gradient descent



- What we just see is “Batch” gradient decent
  - The gradient will be computed using the whole training data set
  - Since it uses the whole dataset at every iteration, hence, it makes the computation very, very slow especially when the data set is very large.

- Idea: rather than using the full gradient, just use one training example
  - Fast to compute $\mathrm{w}=\mathrm{w}-\alpha\nabla J(w; y_i)$





(Ref: https://suniljangirblog.wordpress.com/2018/12/13/variants-of-gradient-descent/)

OLS versus gradient descent

$$J(\mathrm{w}) = \sum_i (y^i - \mathrm{w}^{\mathrm{T}} \phi(x^i))^2 \qquad \mathrm{w} = (\Phi^T \Phi)^{-1} \Phi^T \mathrm{y}$$

Ordinary Least Squares (OLS) solution:

+ Very simple in programming

-Requires matrix inverse, which is expensive for a large matrix.

Gradient descent:

+ Fast for large matrices

+ Stochastic GD is very memory efficient

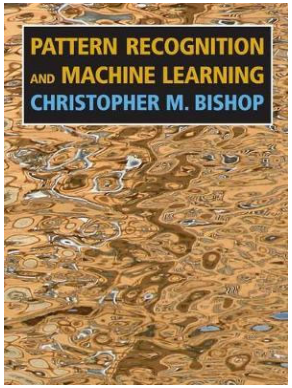
+ Easily extended to other cases

- Parameters to tune (how to decide convergence? what is the learning rate? ….)

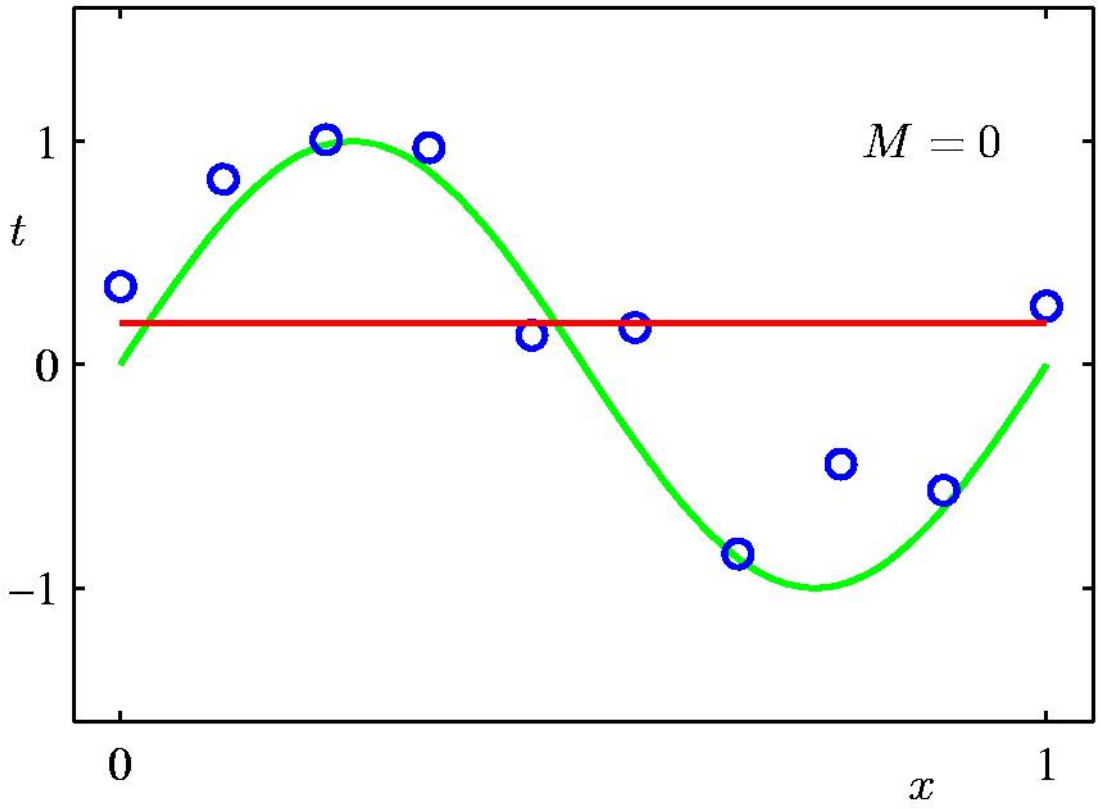
Overfitting and Regularization

An example: polynomial basis vectors on a small dataset

(From Bishop Ch 1)

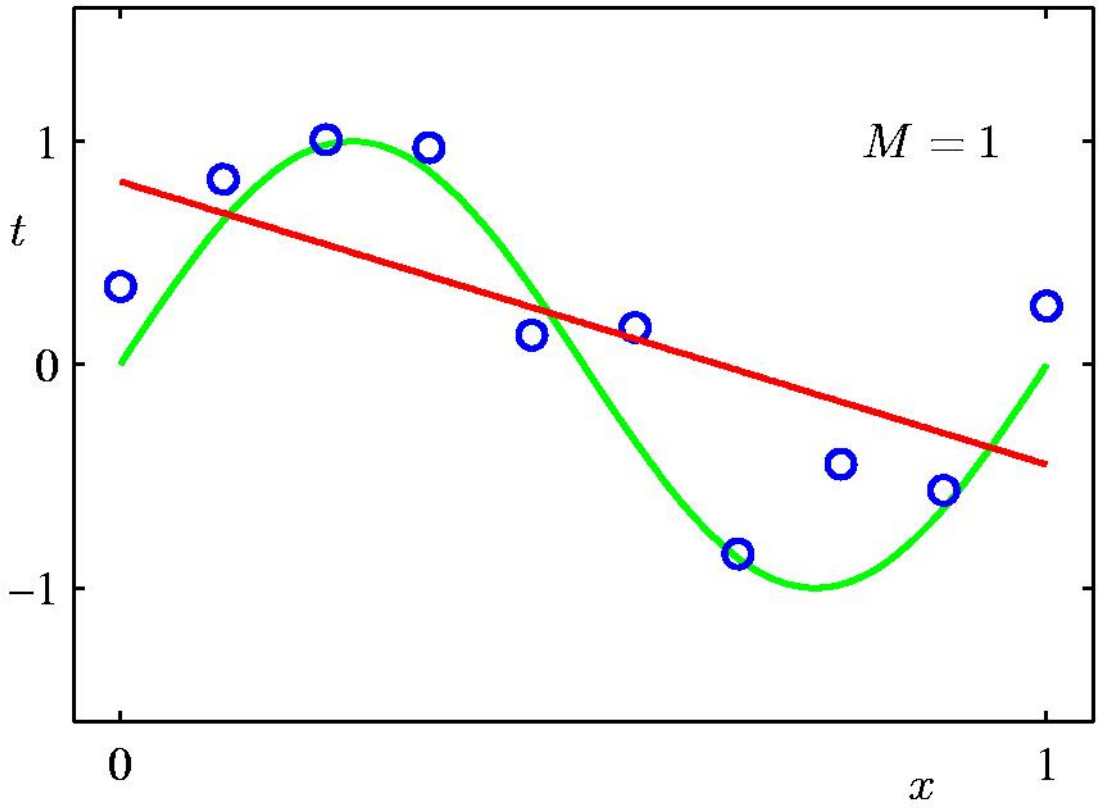


0^{th} Order Polynomial

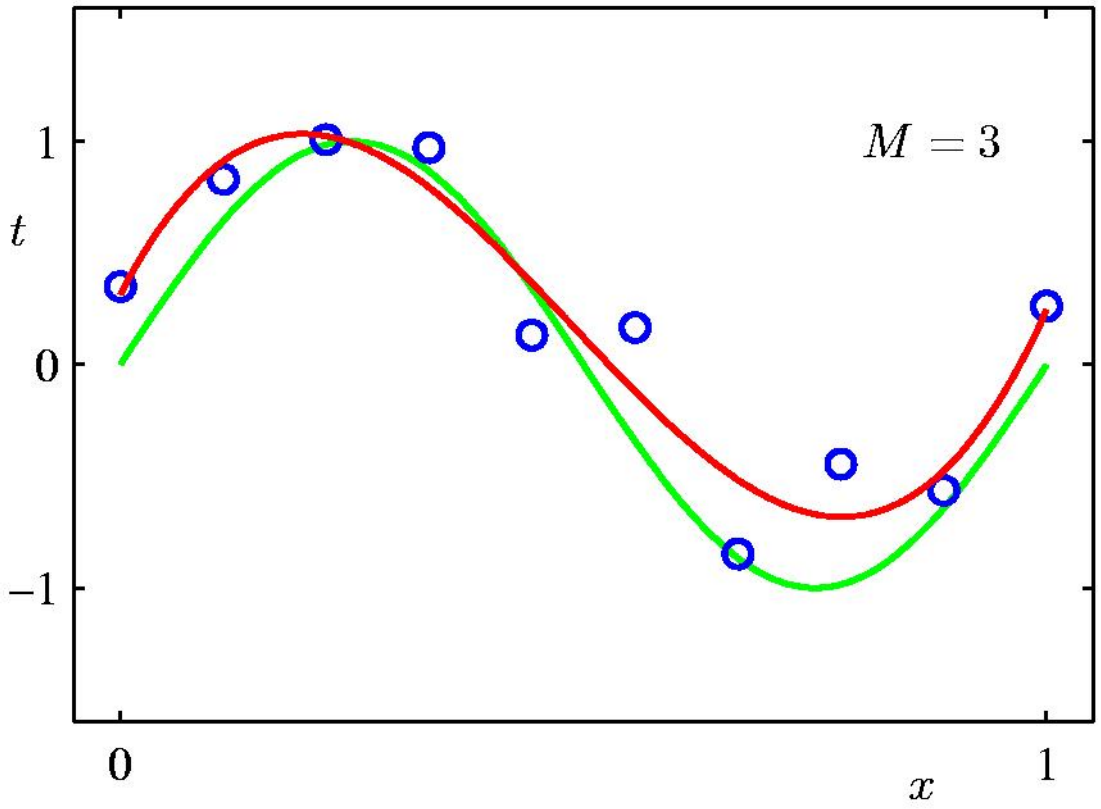


n=10

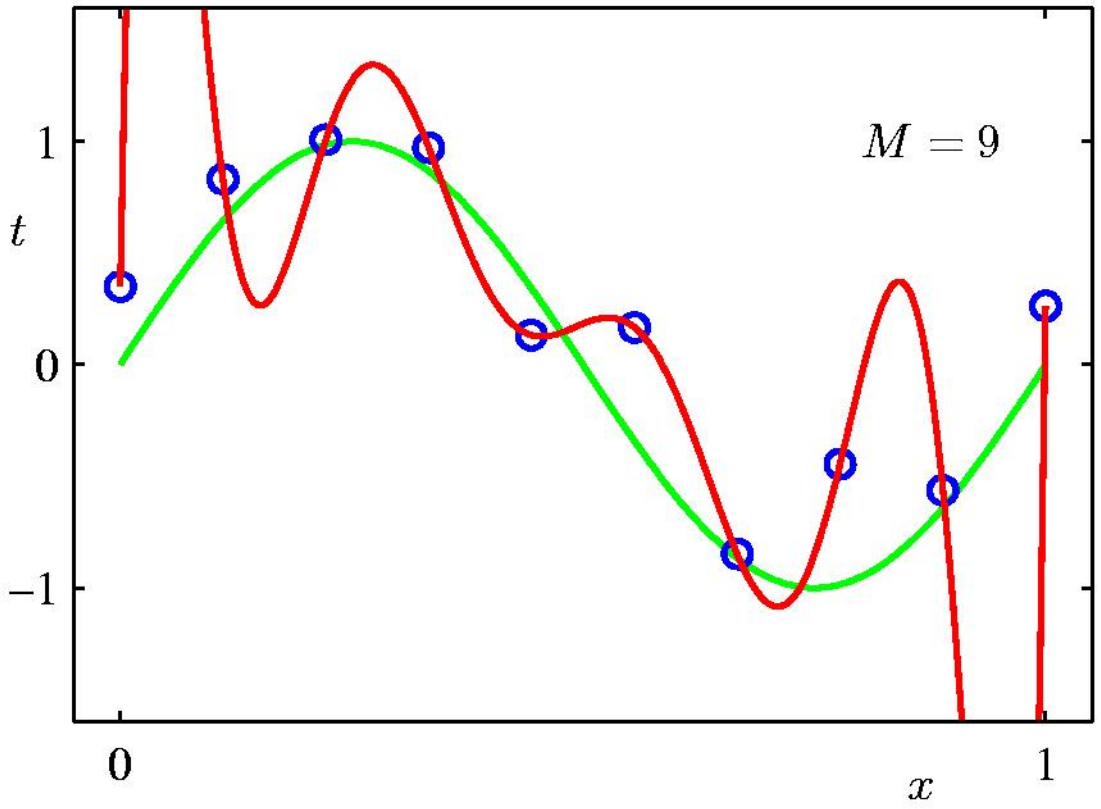
1st Order Polynomial



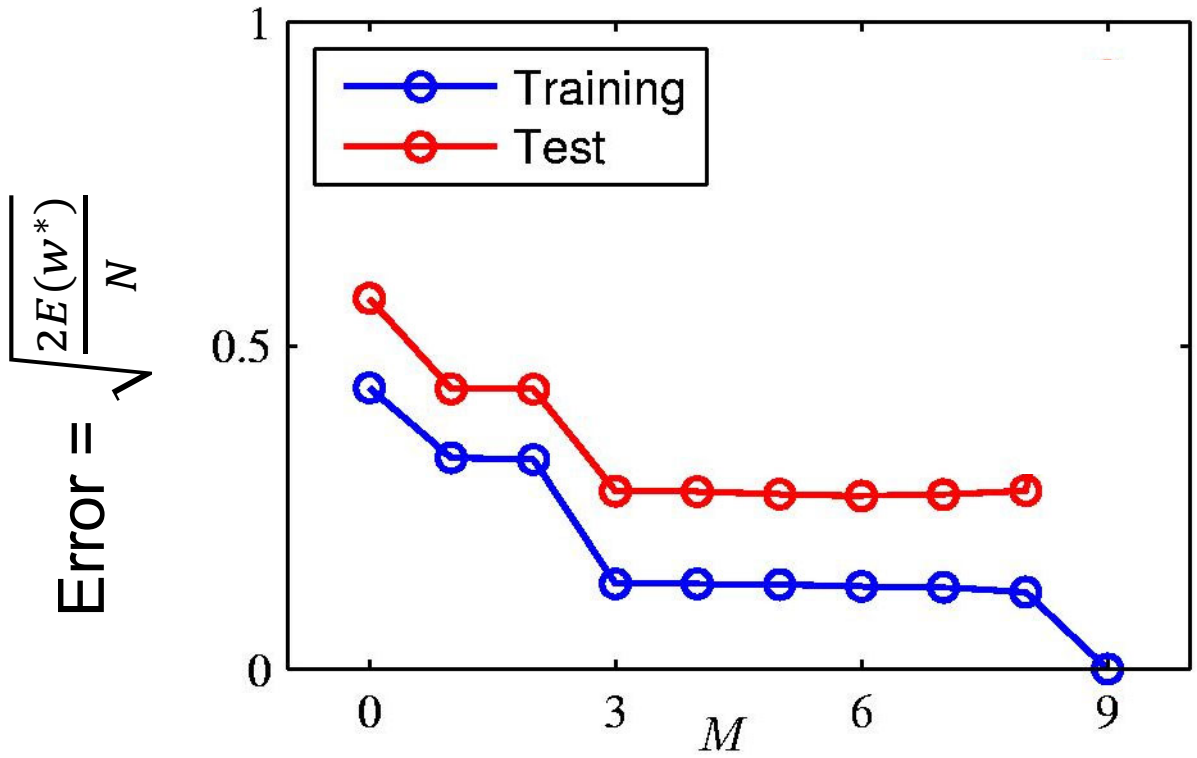
3rd Order Polynomial



9th Order Polynomial



Over-fitting

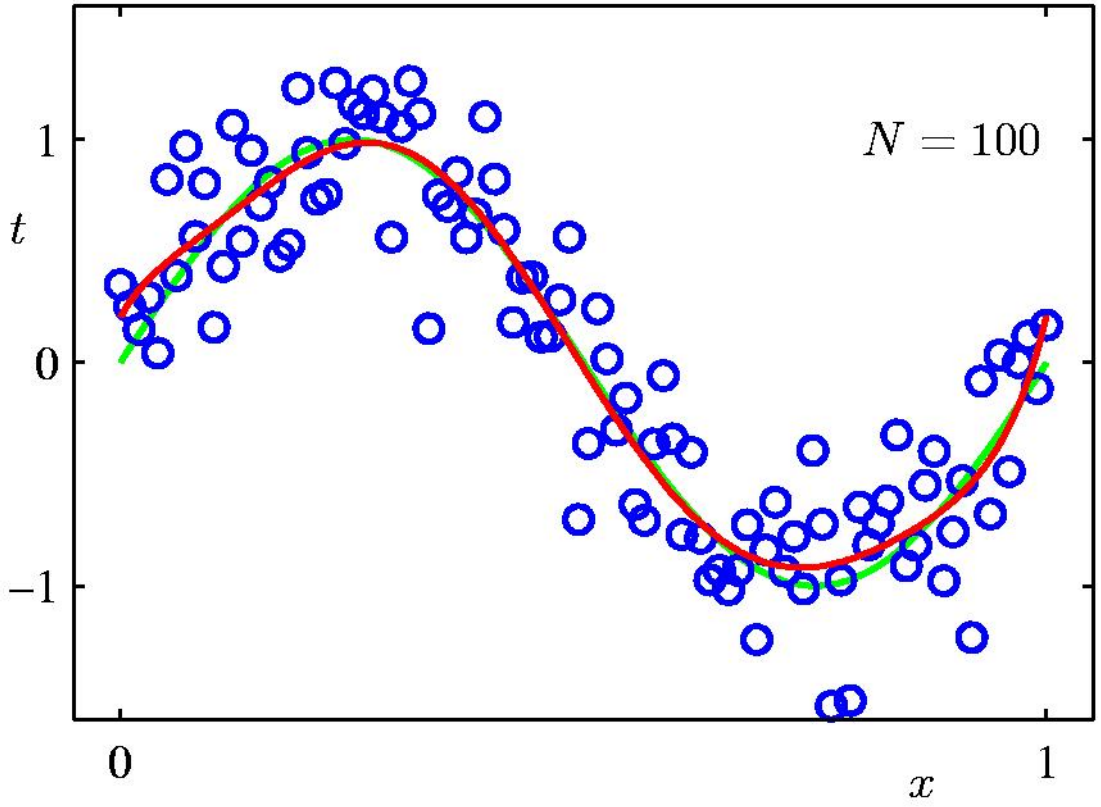


Polynomial Coefficients

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Data Set Size:

9th Order Polynomial $N = 100$



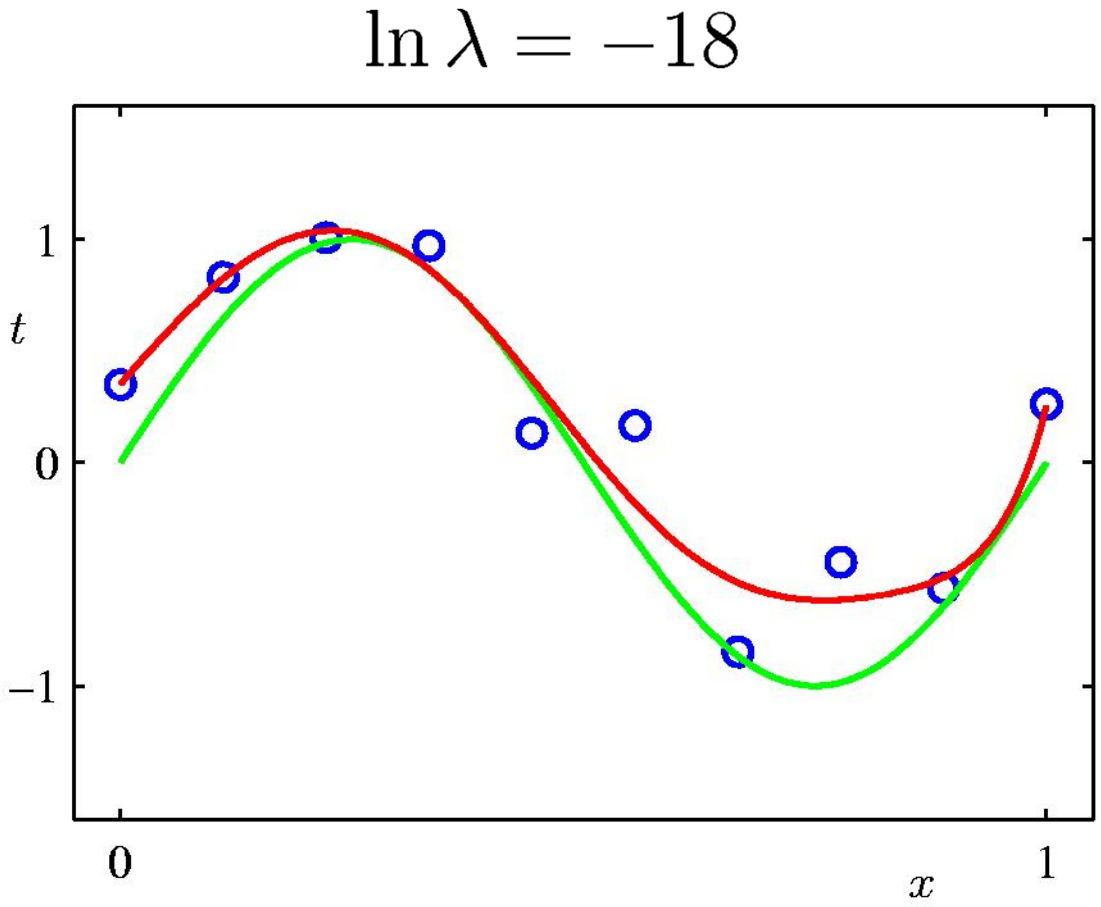
Regularization

Penalize large coefficient values

$$J_{\mathbf{X},\mathbf{y}}(\mathbf{w}) = \frac{1}{2}\sum_i \left(y^i - \sum_j w_j \phi_j(\mathbf{x}^i) \right)^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

Limit the flexibility of the model

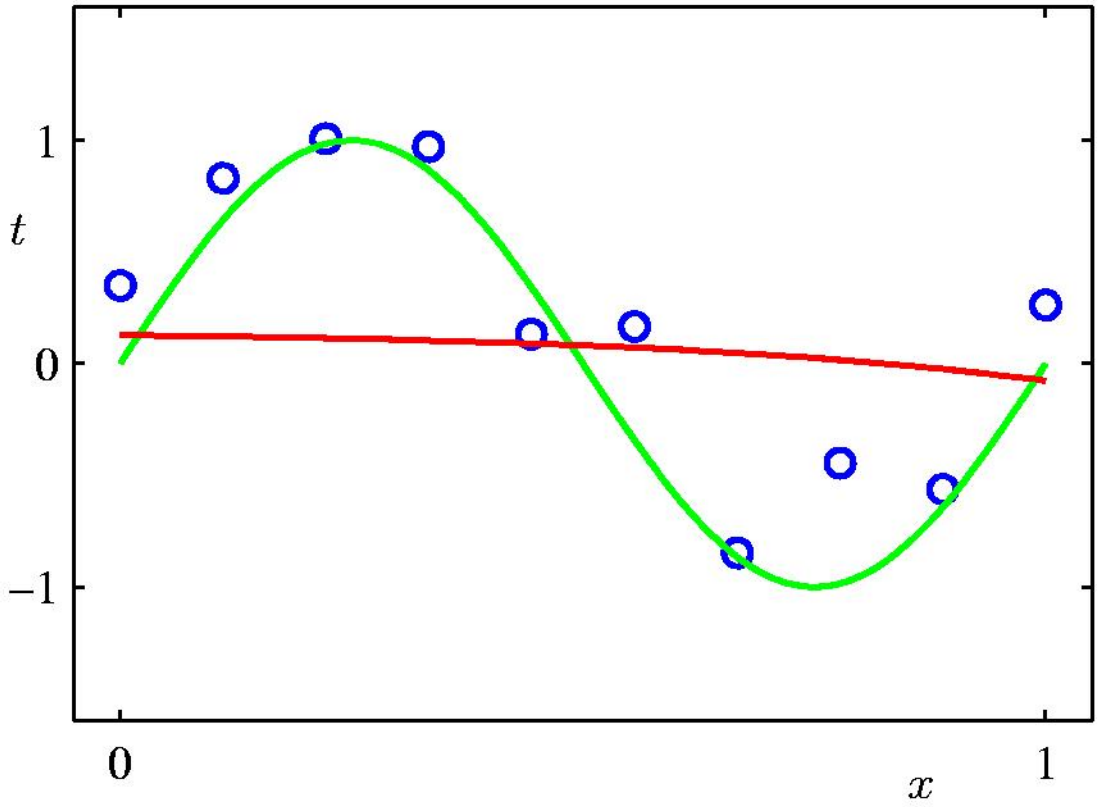
Regularization:



Polynomial Coefficients

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Over Regularization:



Example in Google colab notebook:

https://colab.research.google.com/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.06-Linear-Regression.ipynb#scrollTo=xW7yPbeMBvwy

Lp Regularization:

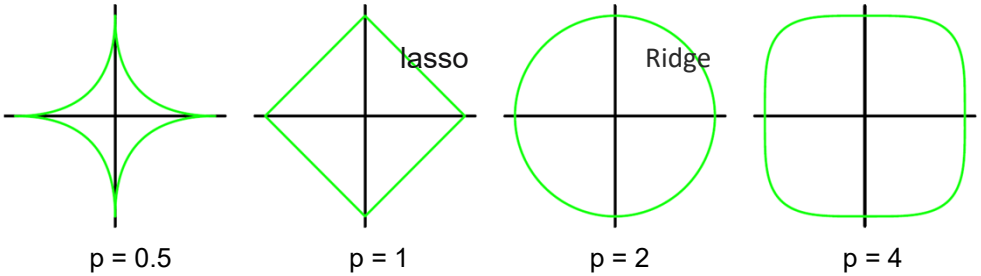
Computational simplicity

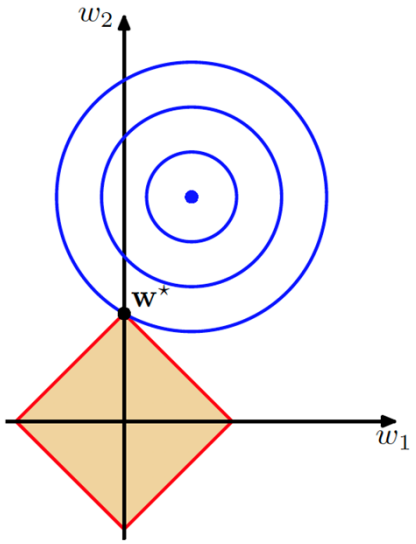
$$J(\boldsymbol{w}) = (\sum_i y^i - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}^i)^2 + \lambda \|\boldsymbol{w}\|_{\boldsymbol{p}}^{\boldsymbol{p}}$$

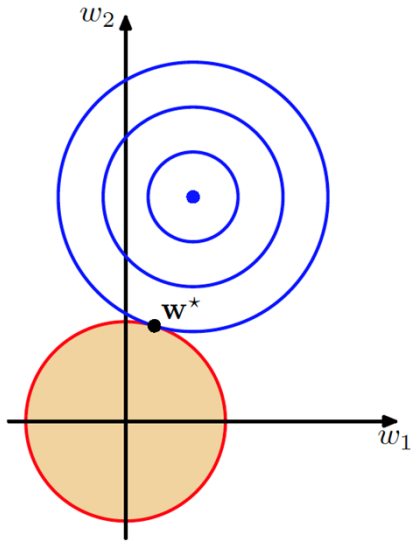
$$\|\boldsymbol{w}\|_p = (\sum_i |w_i|^p)^{\frac{1}{p}}$$

$$\|\boldsymbol{w}\|_2 = \sqrt{(\sum_i^I w_i{}^2)} = \sqrt{w_1{}^2 + w_2{}^2 + \cdots + w_I{}^2}$$

$$\|\boldsymbol{w}\|_1 = \sum_i^I |w_i| = |w_1| + |w_2| + \ldots + |w_I|$$







What if we are going to forecast the future?

Autoregressive (AR) Modeling

- Used for forecasting
- Takes Advantage of Autocorrelation
  - 1st order - correlation between consecutive values
  - 2nd order - correlation between values 2 periods apart
- Autoregressive Model for *p-th* order:

Random Error

$$Y_i = A_0 + A_1 Y_{i-1} + A_2 Y_{i-2} + \cdots + A_p Y_{i-p} + \delta_i$$

Classification by regression?

- Linear regression is not suitable for classification problem.

Example

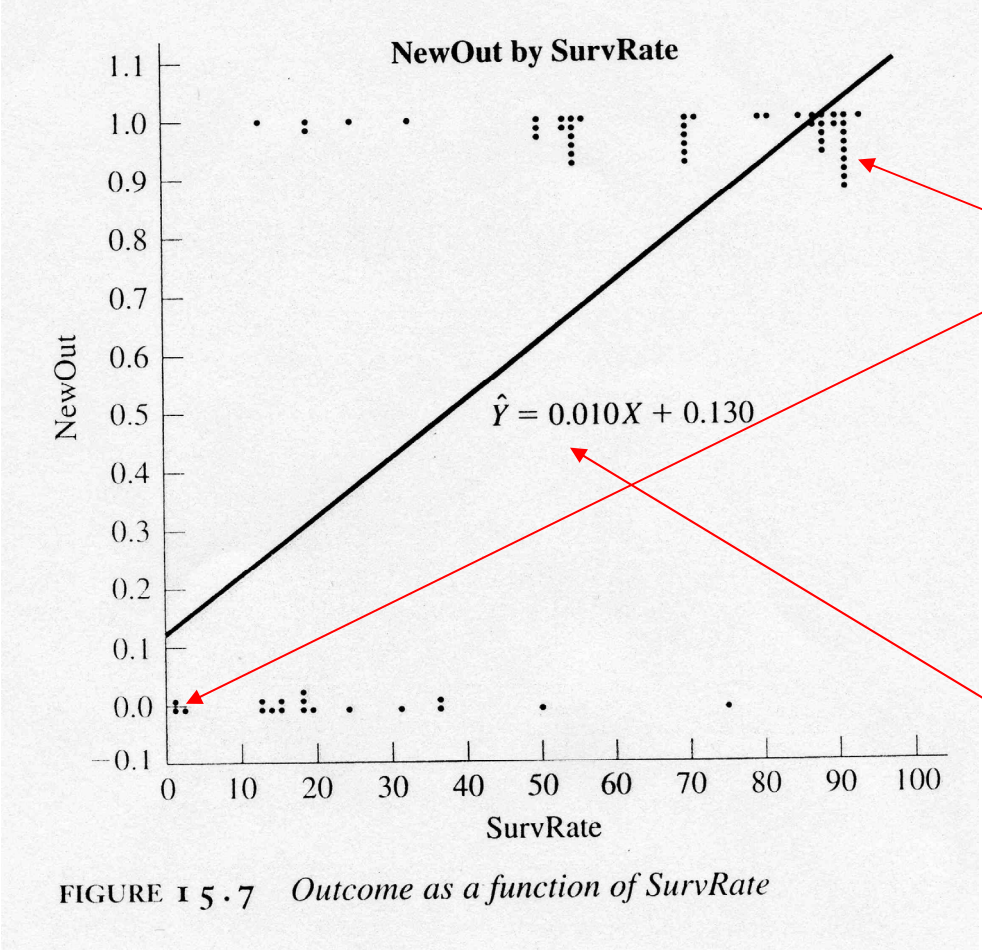


FIGURE 15.7 *Outcome as a function of SurvRate*

Observations:
For each value of SurvRate, the number of dots is the number of patients with that value of NewOut

Regression:
Standard linear regression

Problem: extending the regression line a few units left or right along the X axis produces predicted probabilities that fall outside of [0,1]

Logistic Regression: A better solution!

- To fit a curve to data in which the dependent variable is binary
- Typical application: Medicine
  - We might want to predict response to treatment, where we might code survivors as 1 and those who don't survive as 0

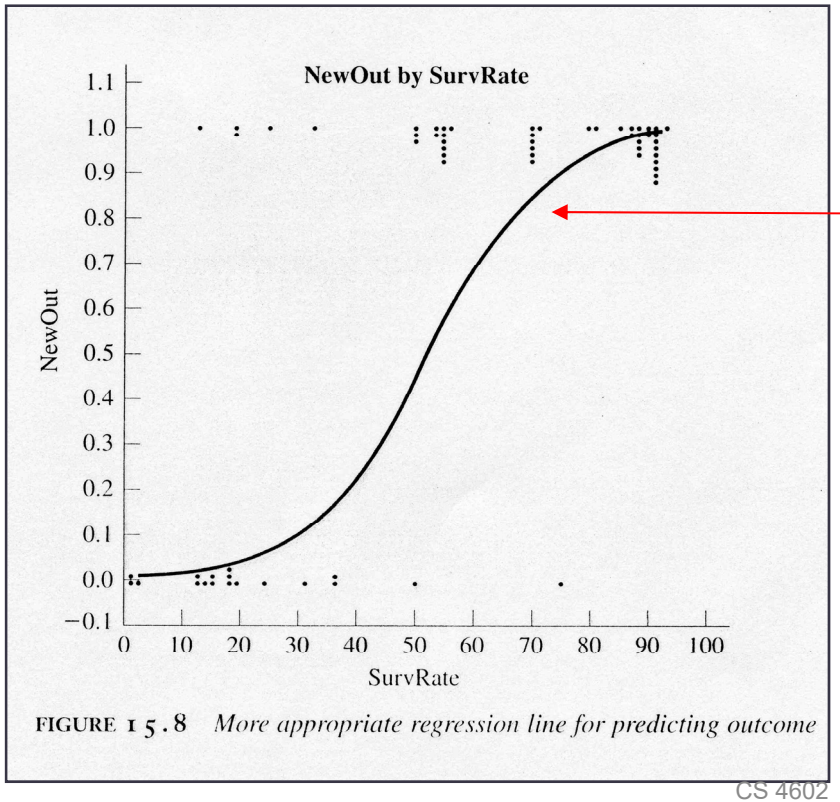


FIGURE 15.8 *More appropriate regression line for predicting outcome*

Regression Curve:
Sigmoid function!
(Logistic function)
(bounded by asymptotes y=0 and y=1)

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

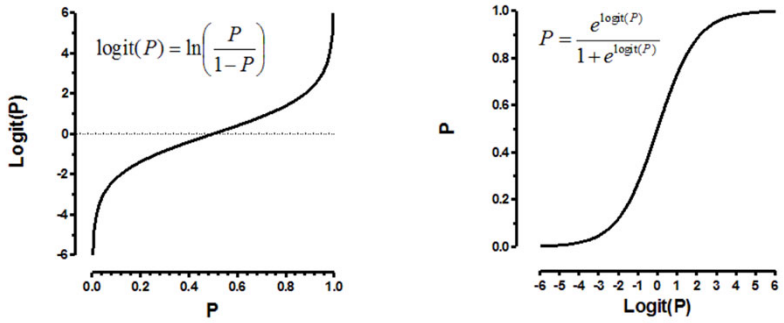
Logistic Regression

- In logistic regression, we seek a model:

$$\text{logit}(p) = \beta_0 + \beta_1 X$$

Y

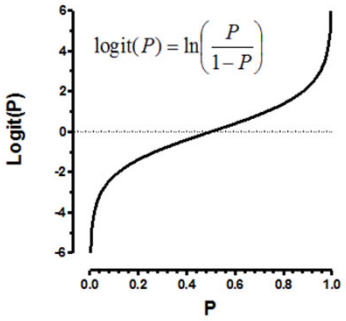
- That is, **the log odds (logit)** is assumed to be linearly related to the independent variable X
- So, now we can focus on solving an ordinary (linear) regression!

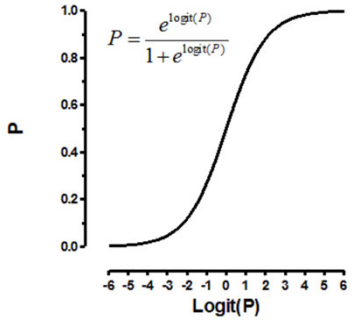


Logit Transform

- p is the probability that the event Y occurs, p(Y=1)
  - [range=0 to 1]
- p/(1-p) is the "odds"
  - [range=0 to ∞]
- ln[p/(1-p)]: log odds, or "logit“
  - [range=-∞ to +∞]

- logit(p) = ln(odds) = ln (p/(1-p))





Logistic Regression: $\text{logit}(p) = \beta_0 + \beta_1 X$

We've learned

- Regression *vs* Classification
- Linear regression – another discriminative learning method
  - As matrix inversion (Ordinary Least Squares)
  - As optimization: Gradient descent
    - Batch gradient decent
    - Stochastic gradient descent
- Overfitting and regularization
- Autoregression
- Logistic regression (more like classification)

Questions?