McGill **Artificial Intelligence** Society

Lecture 2: Regression

Slides based off of Machine Learning at Berkeley
https://github.com/mlberkeley/Machine-Learning-Decal-Fall-2018

Today’s Lesson Plan

Linear Regression

Optimization Via Gradient Descent

Logistic Regression

Multinomial Regression

Recall two main types of supervised algorithms

Regression:

- Input maps directly to a continuous output space
- Involves estimating or predicting a response
- Ex: predicting housing prices, computing trends in stock market

Classification:

- Input maps to a class label
- Classification is the act of identifying group membership
- Ex: image classification, semantic classification in text

Linear Regression

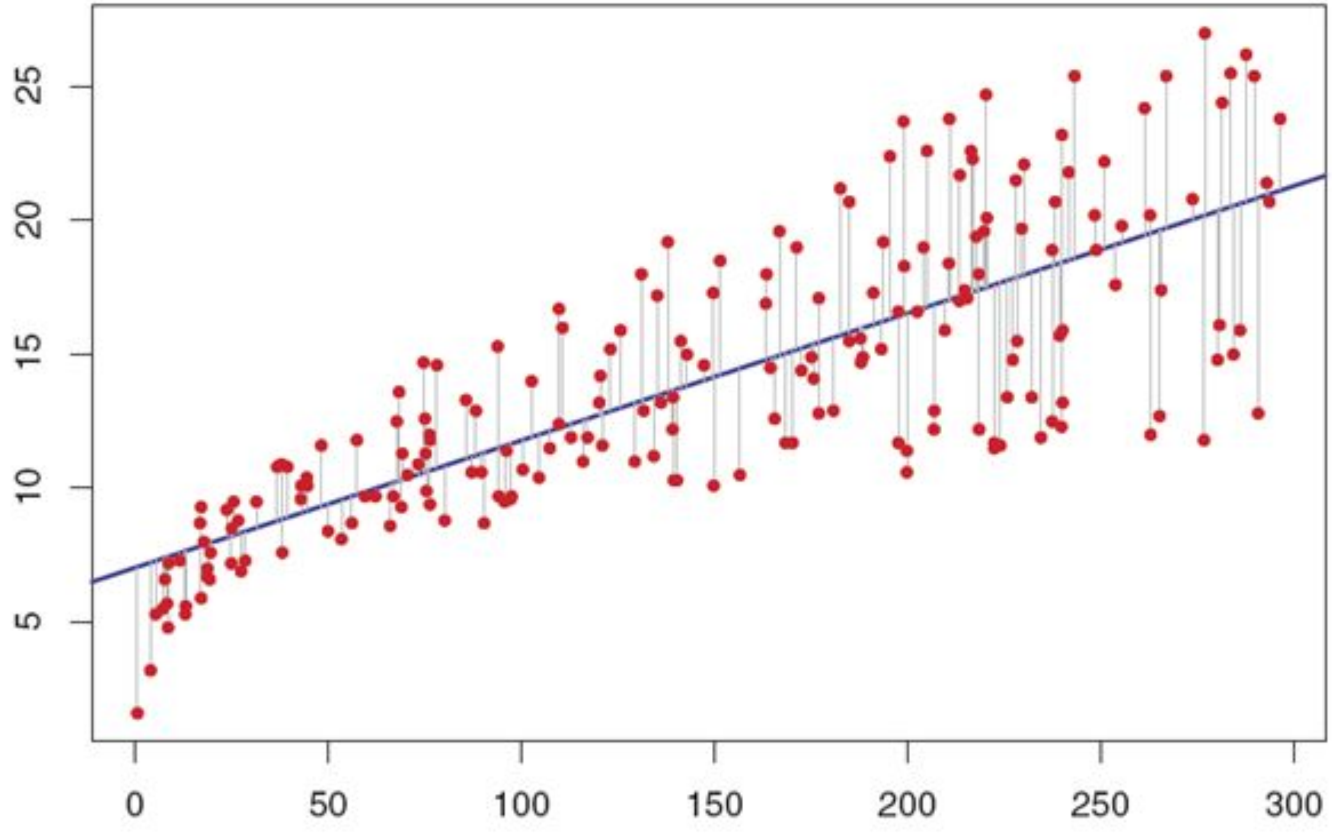


Image credits to Rachna Devasthali from towarddatascience.com

Linear Regression

We begin with the general format of a linear regression problem

$$y = \theta_1 x + \theta_0 + \epsilon$$

Where we model real-world practicalities with standard Gaussian noise

$$\epsilon \sim \mathcal{N}(0, \sigma^2)$$

In vector(compact) form, we write:

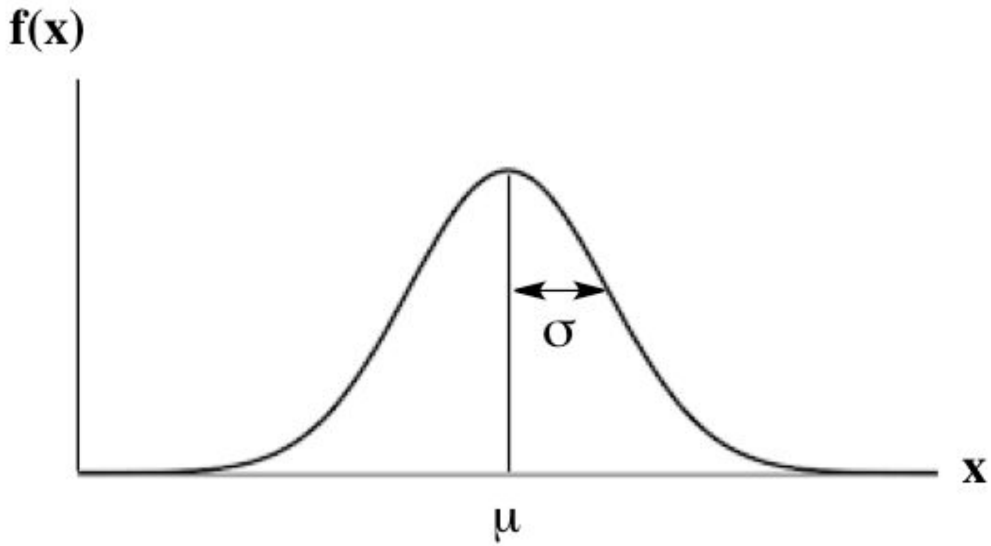
$$y = \theta^T X + \epsilon$$

Probabilistic interpretation:

$$p(y|x, \theta) = \mathcal{N}(y|w^T x, \sigma^2)$$

Quick Review Of Gaussian Distribution!

The Gaussian Distribution



$$f(x) = \frac{1}{\boxed{\sigma}\sqrt{2\pi}} e^{-(x-\boxed{\mu})^2/2\sigma^2}$$

$$e = 2.71828$$

Distribution defined by its mean, variance

What are we trying to solve?

In essence, we want to solve the optimization problem

$$\hat{\theta} = argmax_{\theta}\, log\; p(D|\theta)$$

Where D is a representation of the dataset. The log-likelihood is therefore written as follows:

$$l(\theta) = log\; p(D|theta) = \sum_{i=1}^{N} log\; p(y_i|x_i, \theta)$$

In practice, negative log likelihood is used

$$NLL(\theta) = -\sum_{i=1}^{N} log\; p(y_i|x_i, \theta)$$

Framing the Optimization Problem

We expand the likelihood equation to its full form

$$l(\theta) = \sum_{i=1}^{N} log[(\frac{1}{2\pi\sigma^2})^{1/2} exp(\frac{-1}{2\sigma^2}(y_i - w^T x_i)^2)]$$

$$f(x) = \frac{1}{\boxed{\sigma}\sqrt{2\pi}} e^{-(x-\boxed{\mu})^2/2\sigma^2}$$

$$= \frac{\boxed{-1}}{2\sigma^2} RSS(w) - \cancel{\frac{N}{2} log(2\pi\sigma^2)}$$

We do not consider this term in our optimization problem as it is constant with respect to the parameters of the model

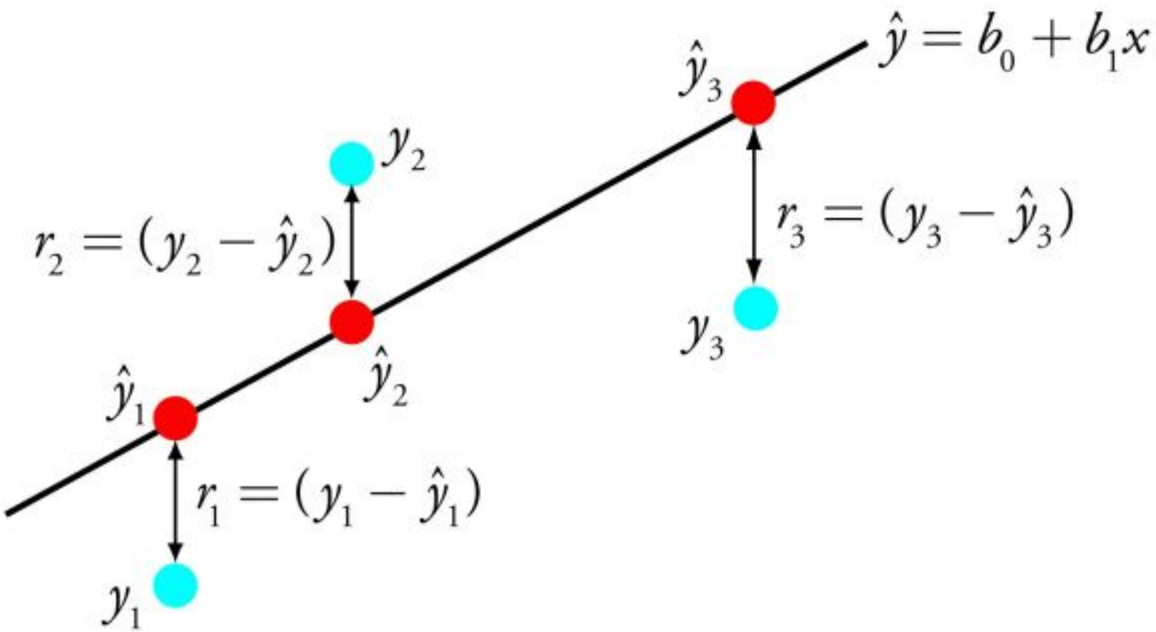
Where the residual sum of squares is:

$$\boxed{RSS(w) = \sum_{i=1}^{N} (y_i - w^T x_i)^2}$$

Linear Regression -Least Squares Method

Let $\hat{y}_i = h(x) = b_0 + b_1 x$

$$\min J(b_0, b_1)$$



Linear Regression -Least Squares Method

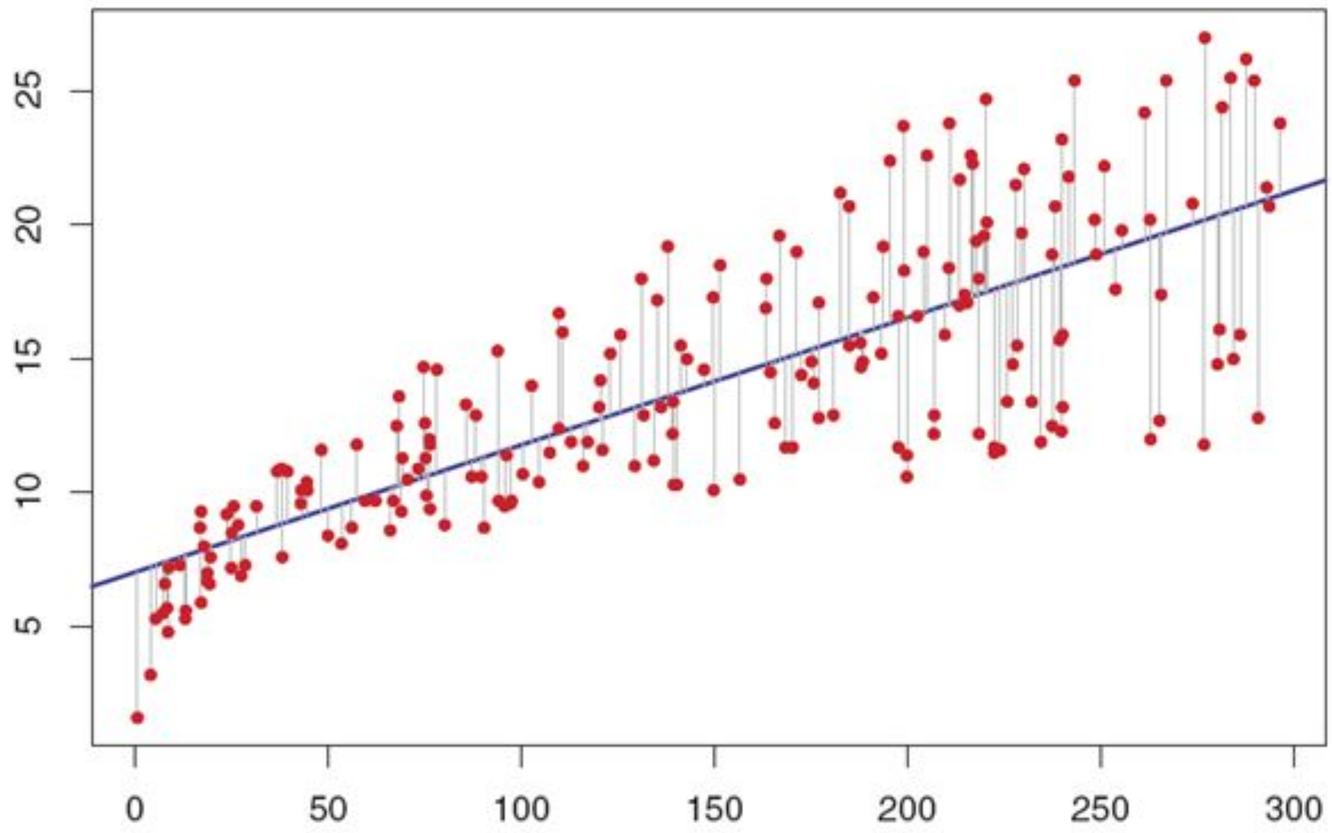
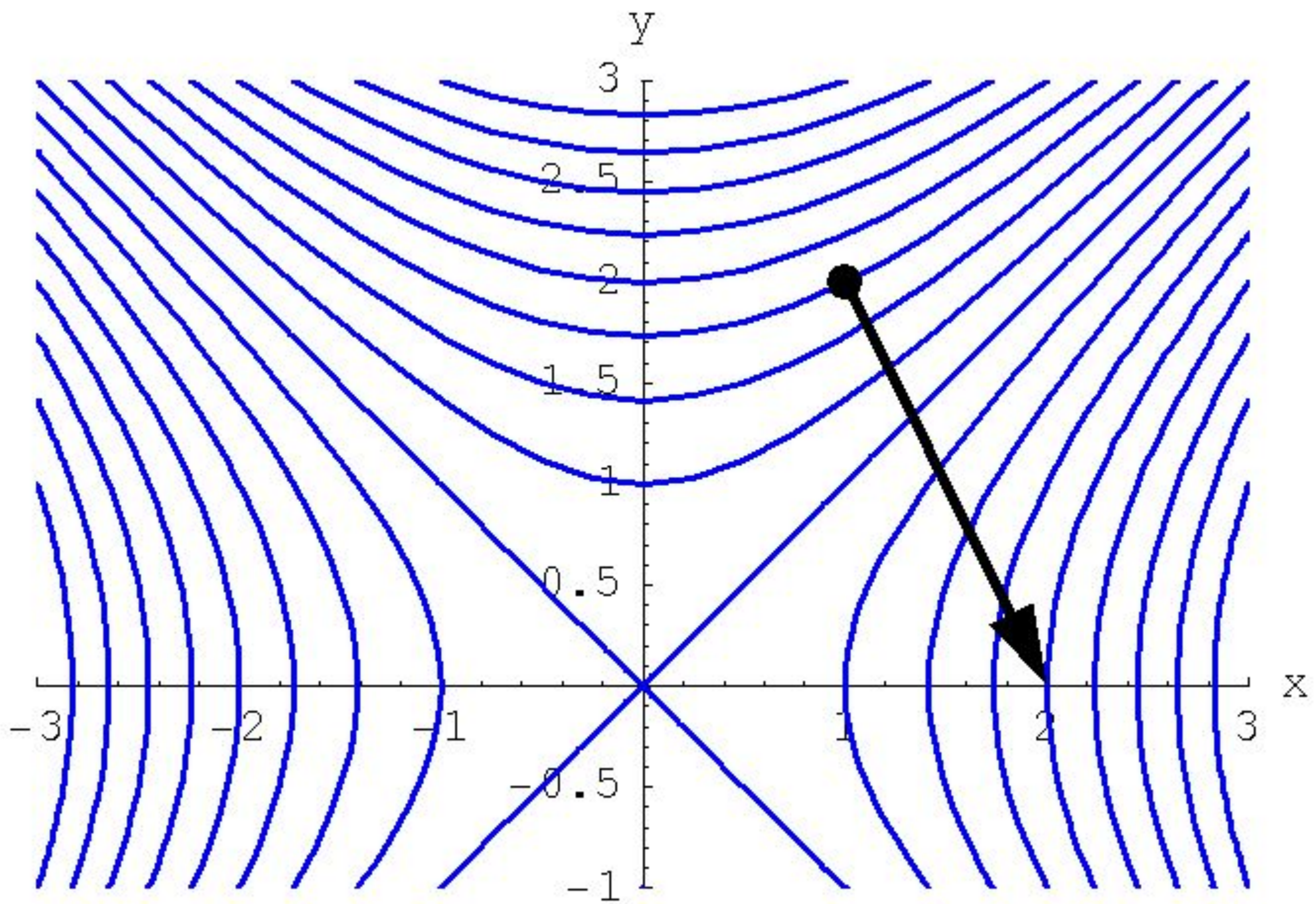


Image credits to Rachna Devasthali from towarddatascience.com

Optimization Via Gradient Descent



Derivation of the MLE

Rewriting the objective in a form that is amenable to differentiation

$$NLL(w) = \frac{1}{2}(y - Xw)^T(y - Xw) = \frac{1}{2}w^T(X^TX)w - w^T(X^Ty)$$

Where:

$$X^TX = \sum_{i=1}^{N} x_i x_i^T$$

And XTY follows, we can compute the gradient of the NLL, and we wish to set it to 0:

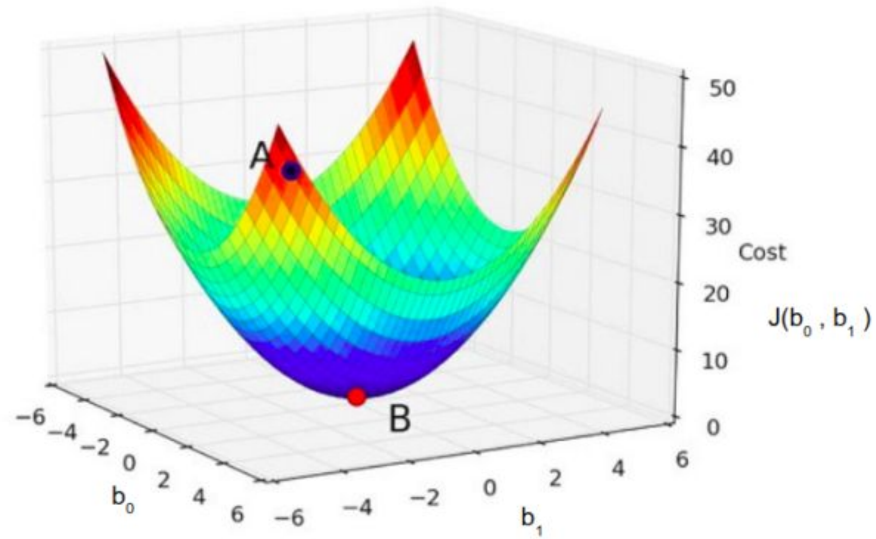
$$g(w) = [X^TXw - X^Ty] = \sum_{i=1}^{N} x_i(w^Tx_i - y_i)$$

How do we know this reaches a minima?

Derivation of the MLE

Given that this problem is convex (proof omitted), we know that it has a unique global minimizer, which we can find by setting the gradient to 0, where the solution to the problem is:

$$X^T X w = X^T y$$

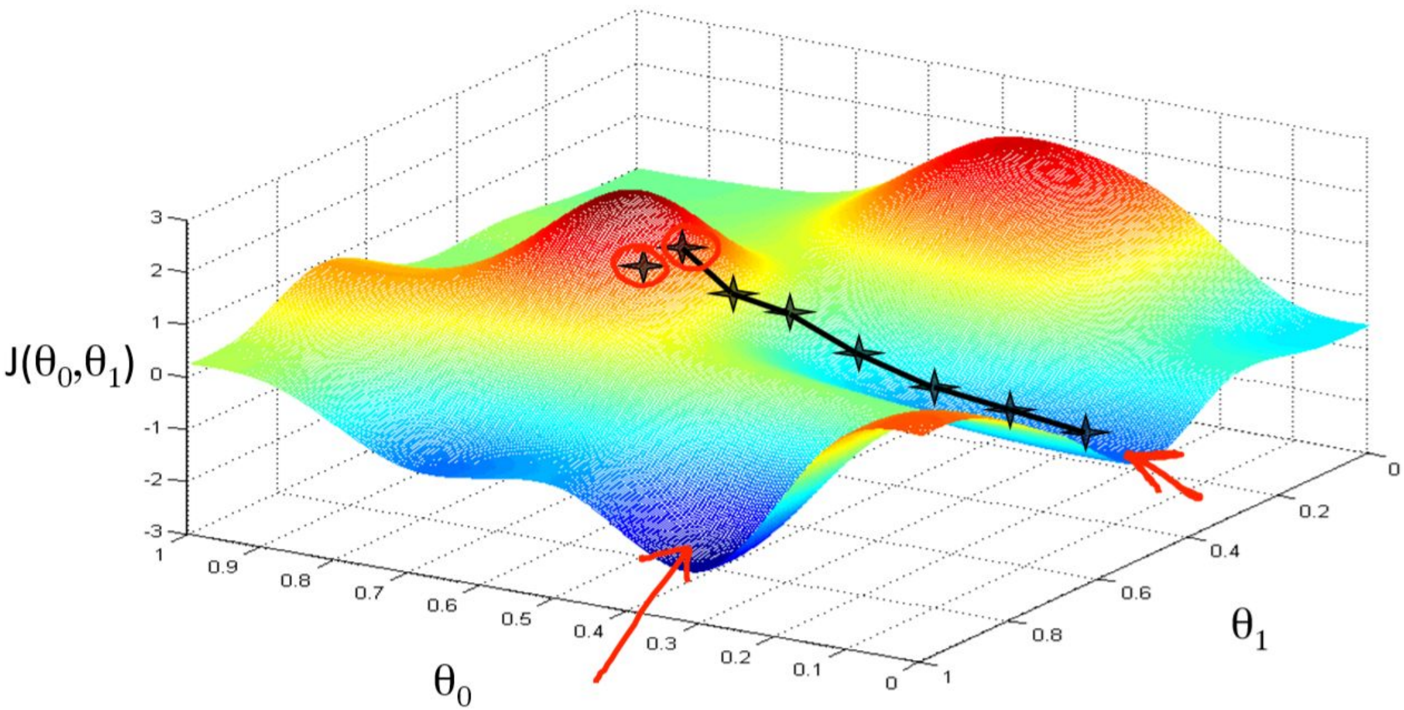


Rearranging the terms, we get the ordinary least squares solution in closed form:

$$\hat{w}_{OLS} = (X^T X)^{-1} X^T y$$

Observe complexity - 3 matrix multiplications, 1 matrix inverse: can compute in polynomial time, **bad for large datasets with many examples, many features**.

Gradient Descent



Gradient Descent

We essentially want to iteratively get closer to the minimum of our objective function (defined as the MSE with respect to our weights).

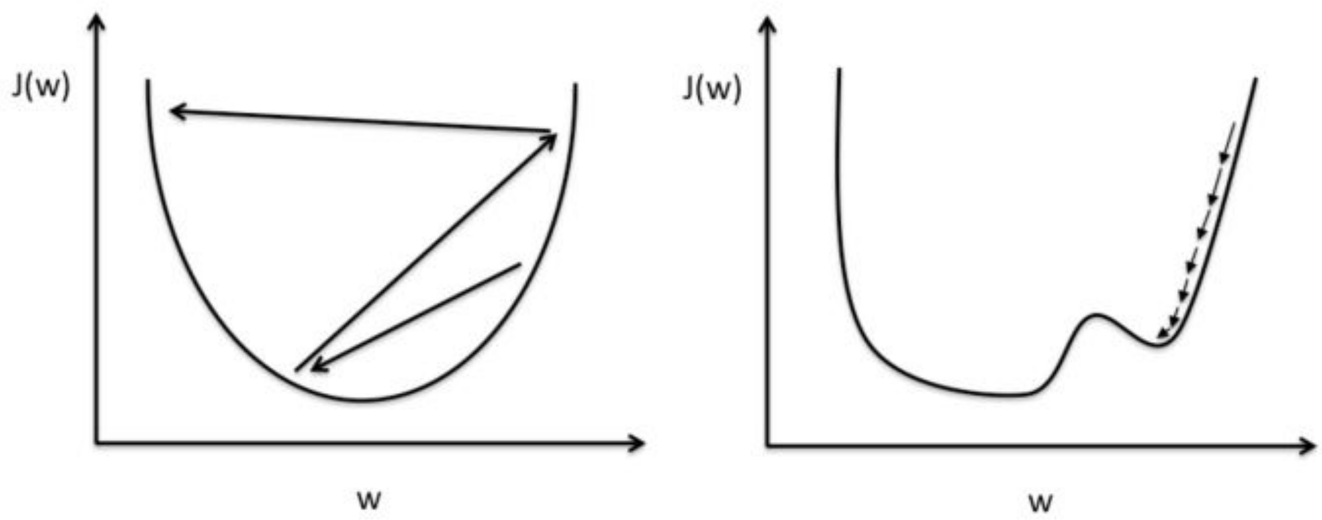
- w_o , w_1 , w_2 , ... such that MSE(w_o) > MSE(w_1) > MSE(w_2) > ...

1. Pick initial w_0
2. For k = 1, 2, ...,
   $w_{k+1} = w_k - \alpha\, g(w_k)$

   where α>0 is called the "learning rate"

End when $|w_{k+1} - w_k| < \varepsilon$

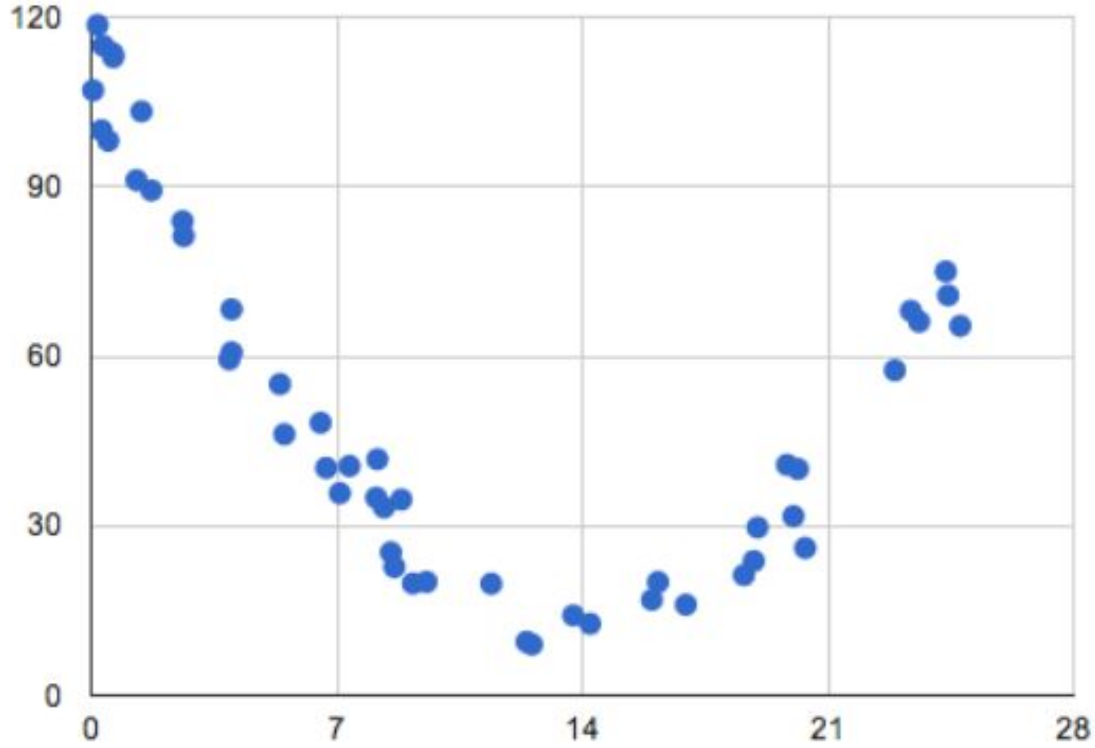
Picking α



Too large: we “overshoot” and don’t converge to the global minima

Too small: the weights might not move far enough to reach a local minima, slow convergence

What if Data is Non-Linear?



$$h(x) = b_0 + b_1 x + b_2 x^2$$

Introducing Polynomial Regression

We linearize our hypothesis h(x):

$$h(x) = b_0 + b_1 x + b_2 x^2$$

$$h(x_1, x_2) = b_0 + b_1 x_1 + b_2 x_2$$

Such that x_1 = x, and x_2 = x

This way, the model is still **linear** with respect to its **parameters**.

Introducing Polynomial Regression

Instead of just using X, we apply a basis function expansion by replacing x with some non-linear function of the inputs s.t.

$$p(y|x,\theta) = \mathcal{N}(y|w^T\phi(X), \sigma^2)$$

Where

$$\phi(X) = [1, x, x^2, ..., x^d]$$

:

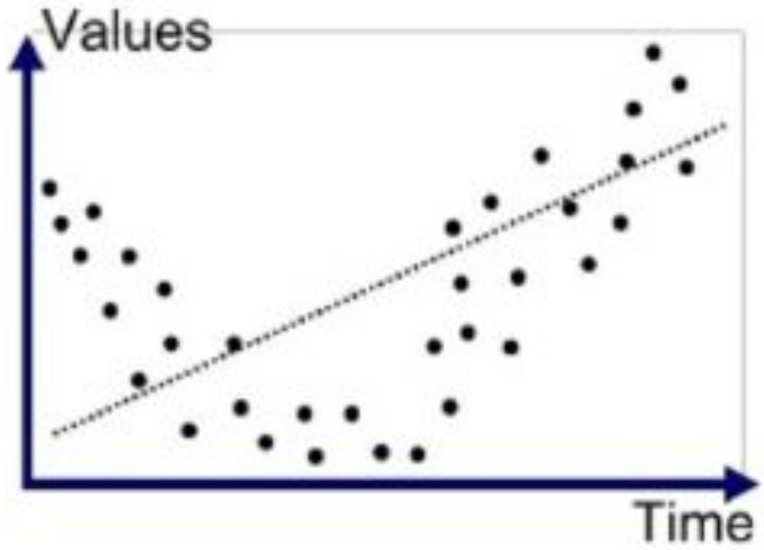
- model is still linear in parameters **w**

- allows to fit nonlinear data (ϕ(X) can be replaced with many other basis function expansions or kernels)

Overfitting vs. Underfitting

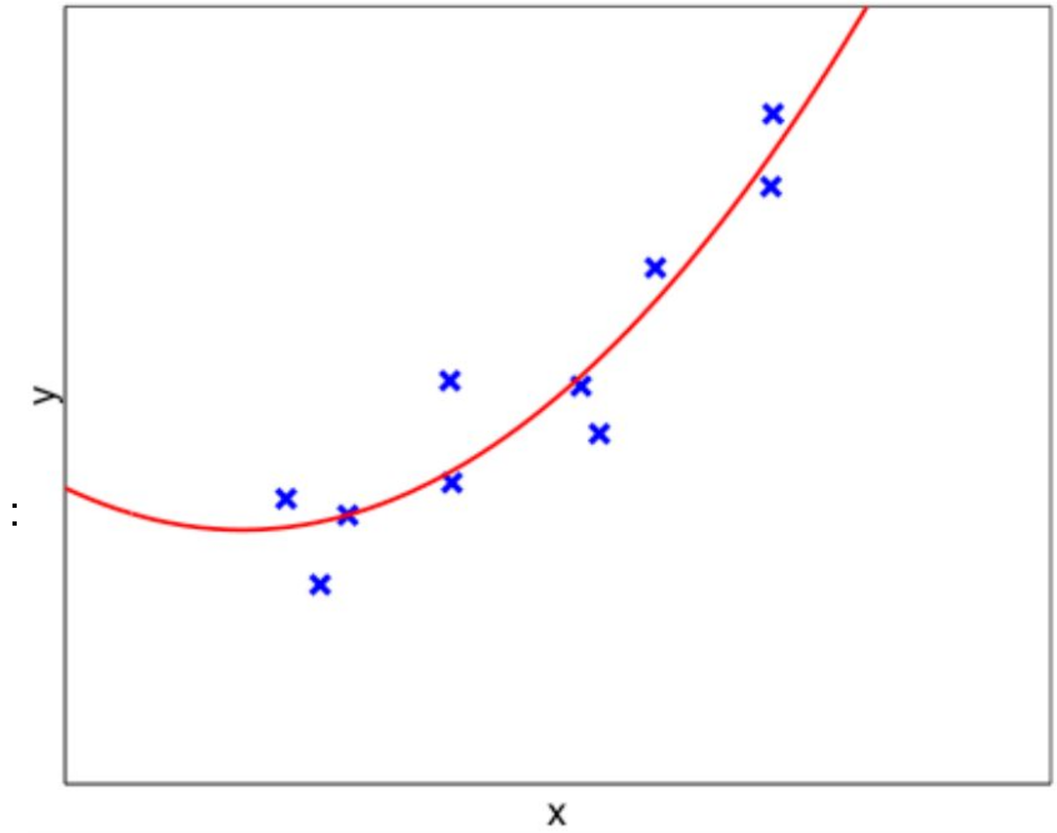
Intuitively, a linear (or a low dimensional polynomial) will not be powerful to fit more complex models → **underfitting**.

:



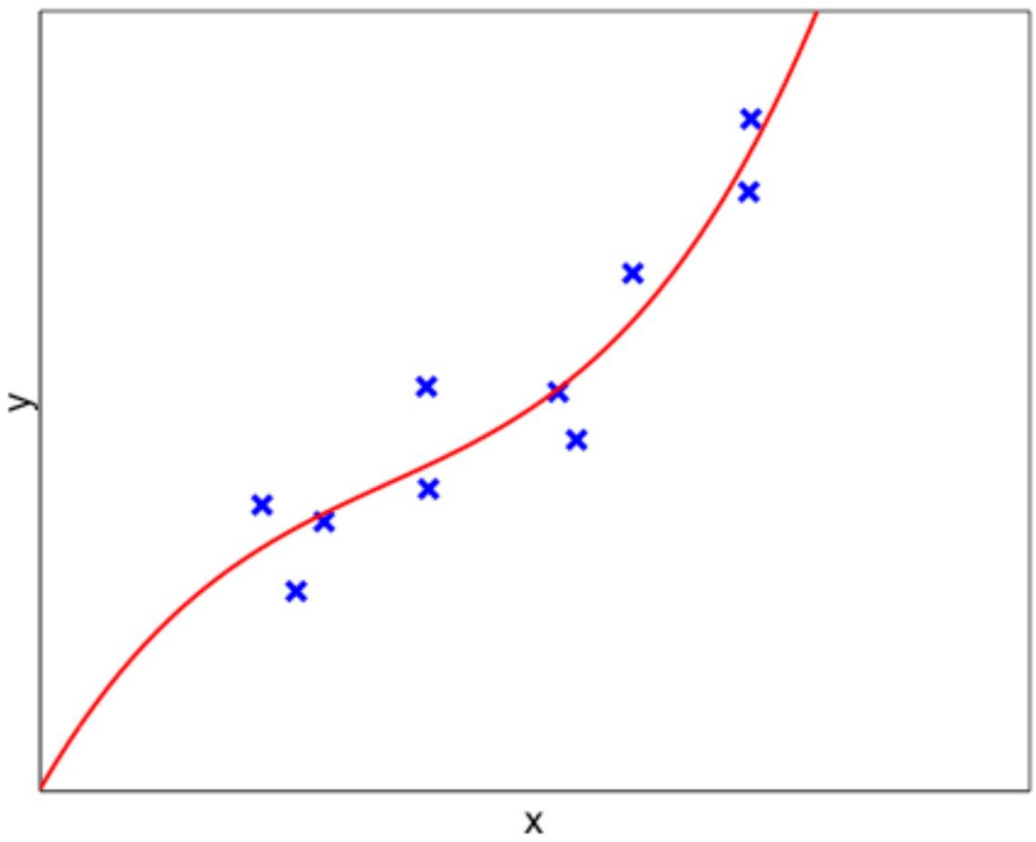
Overfitting -the phenomena by which the model is so adapted to the training set that it no longer generalizes well to the underlying model

Order 2 fit

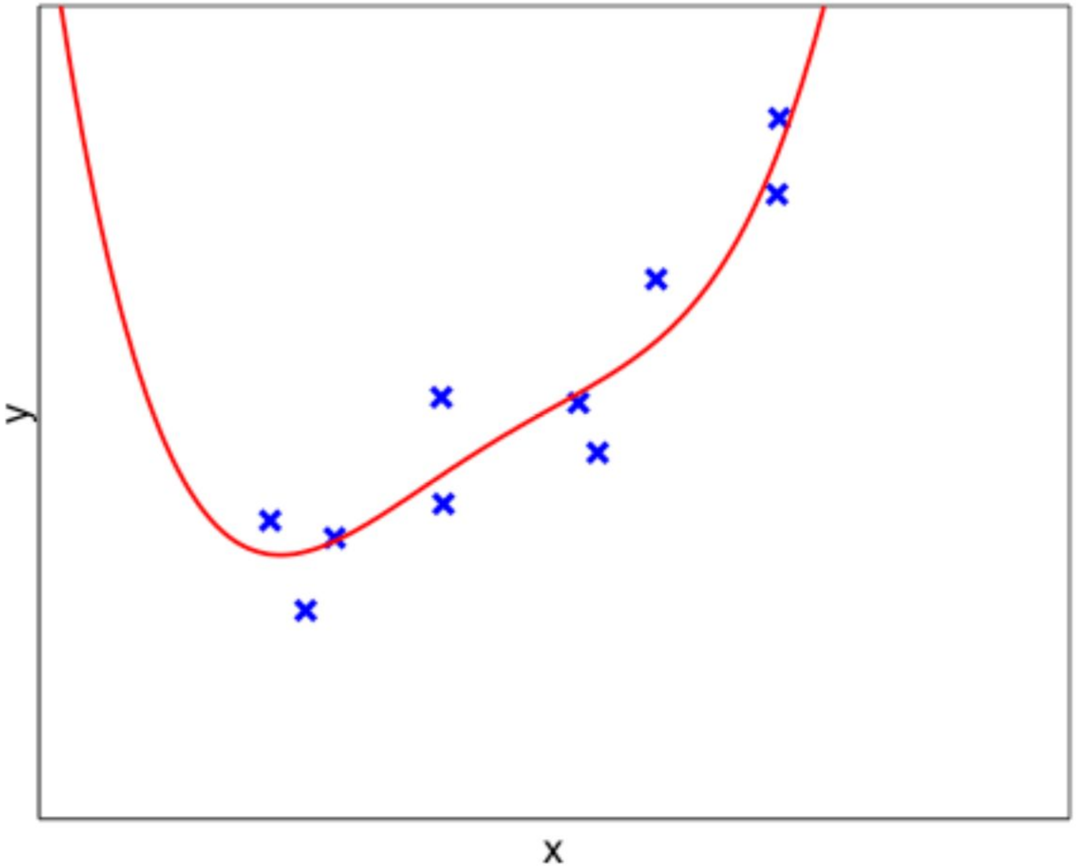


Order 3 fit

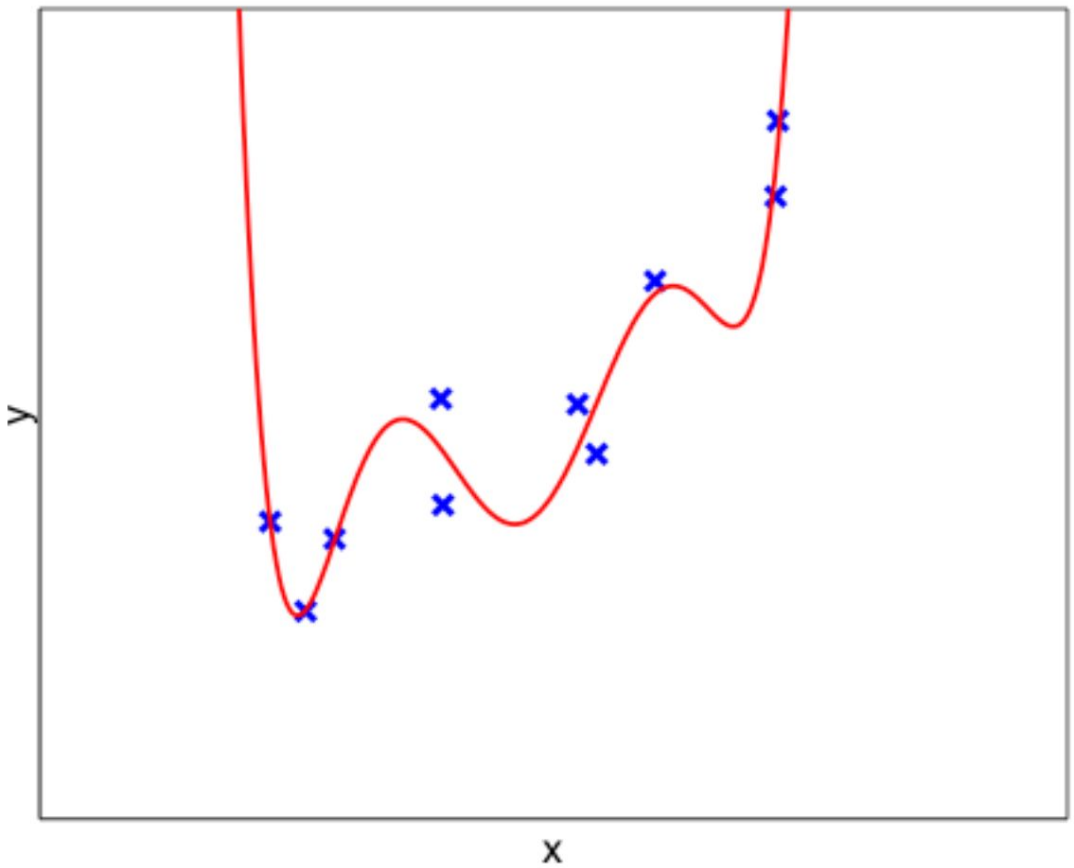
:



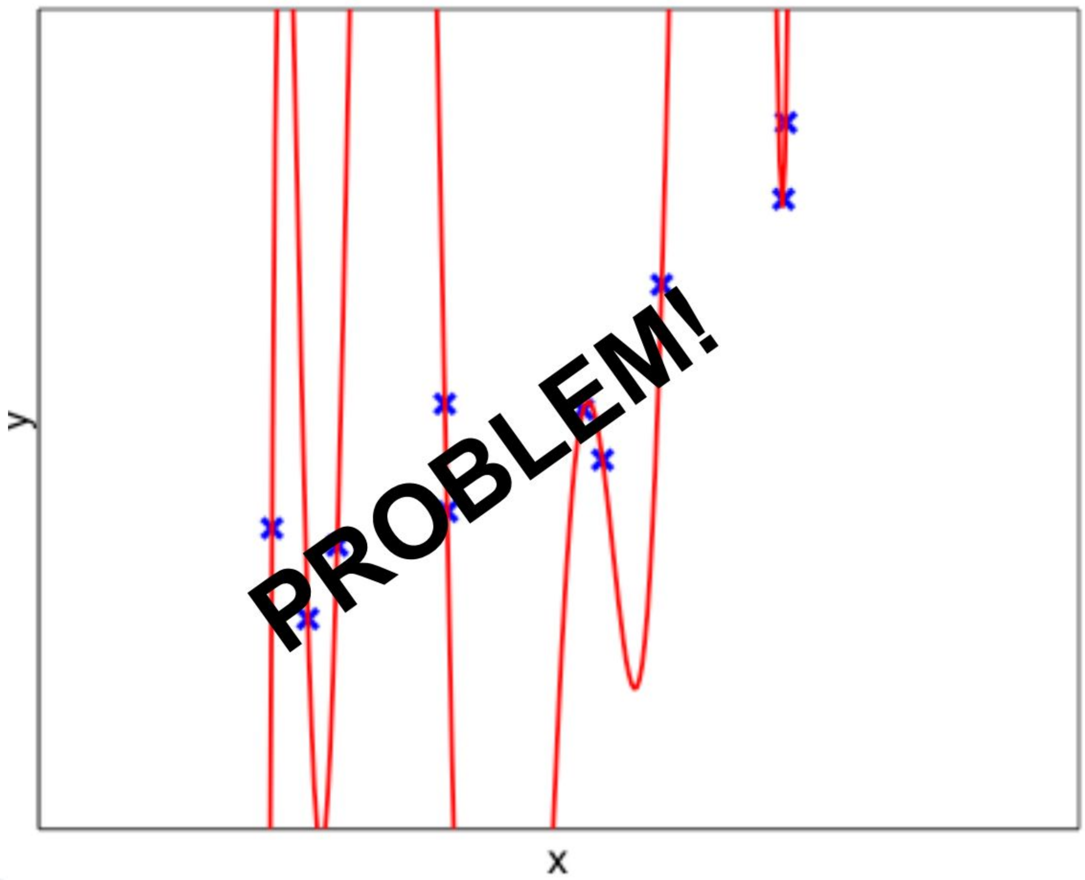
Order 4 fit



Order 6 fit



Order 9 fit



:

Addressing Overfitting

How to address overfitting -

1) **Hyperparameter tuning -**
   Simply modify hyperparameters that control the complexity of the model (in this case, the value of d) until you get the validation set accuracy optimized

2) **Adding more data-**
   Adding more data, so simply having more training set data can allow for a more complex model without overfitting

More sophisticated methods: Ridge regression (L2 regularization), dropout in neural networks, lasso regression (L1 regularization), cross-validation, etc.

:

5-MINUTE BREAK

Now Consider the Following Problem...

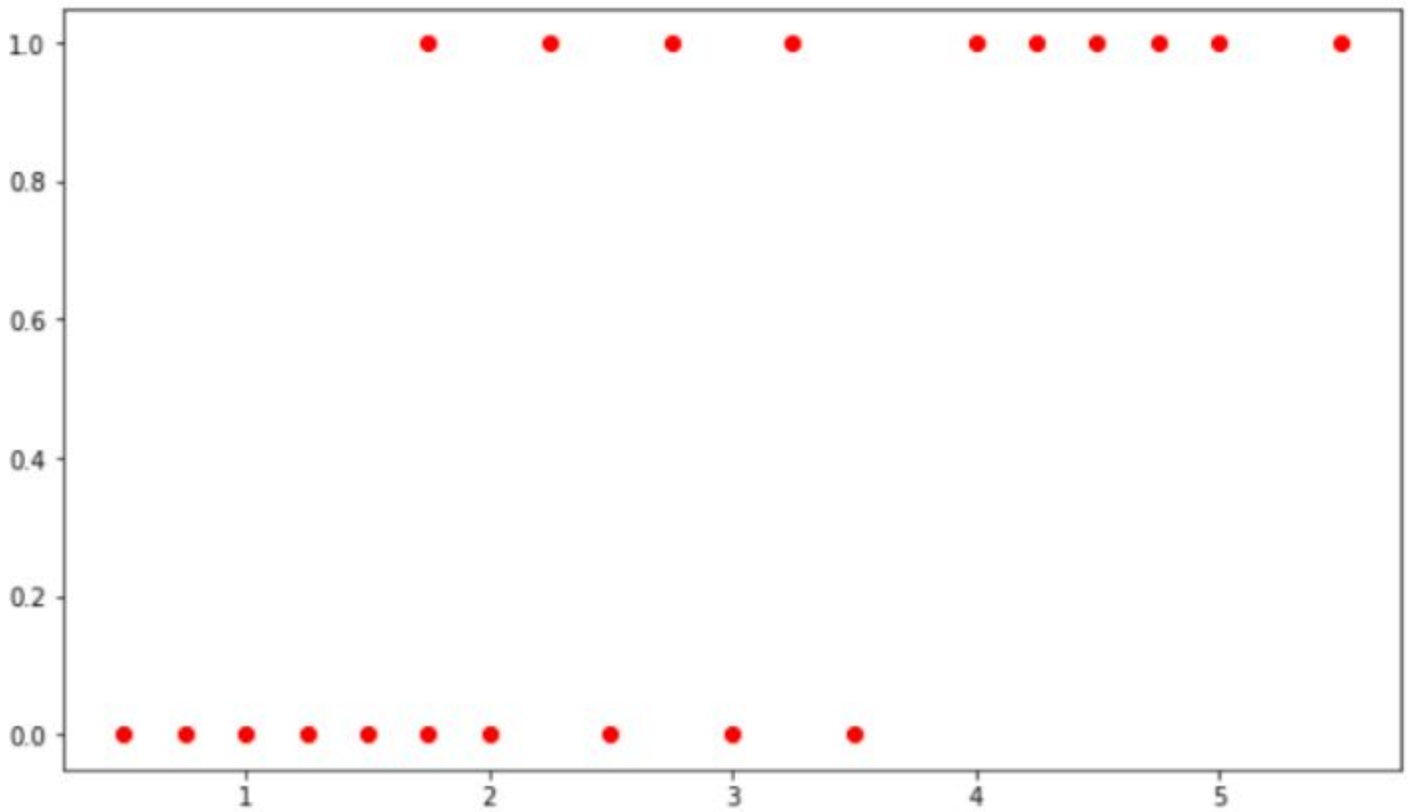


Image credits to Berkeley DeCal Course

Why Linear Regression Doesn't Work Here

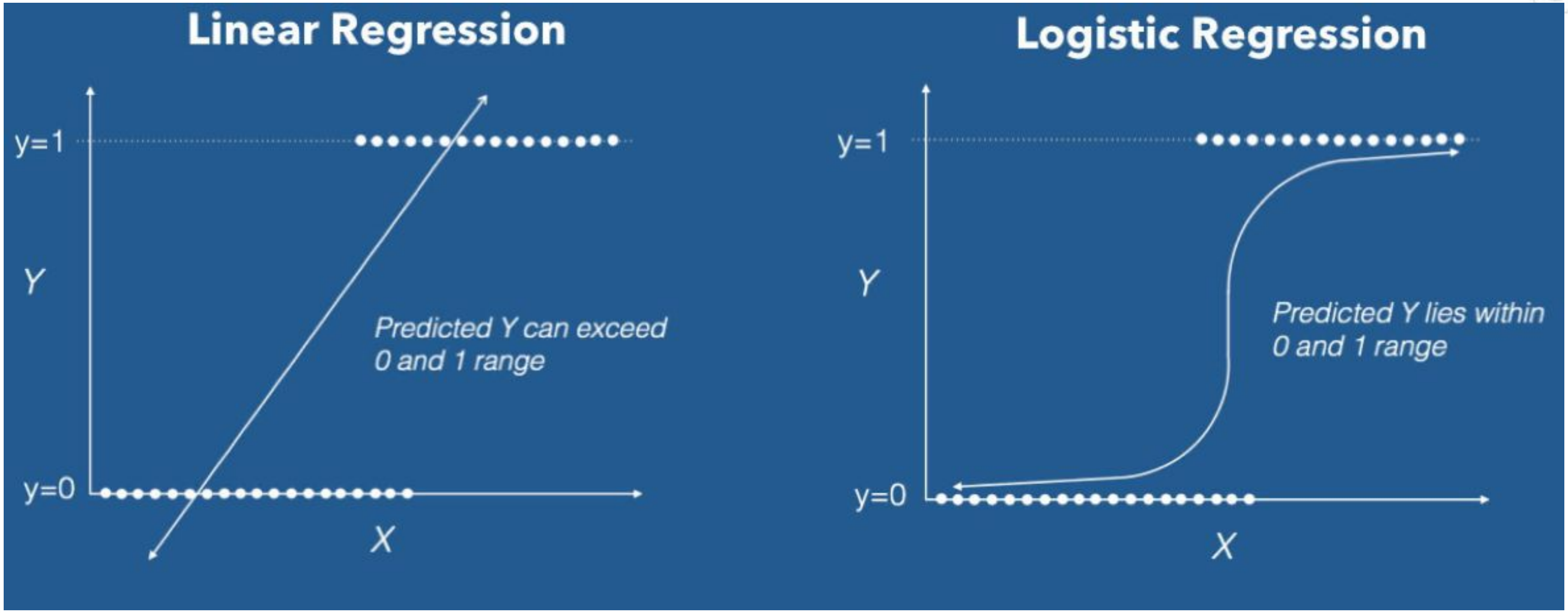


Image Credits

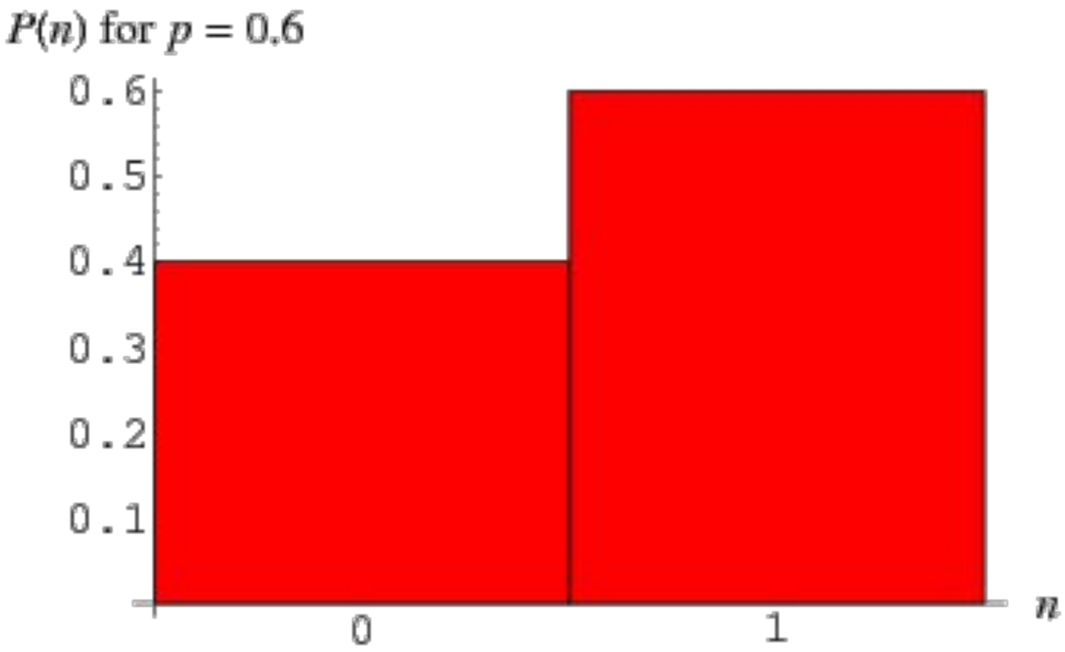
Logistic Regression

Instead of modeling our response directly, logistic regression models the probability that y belongs to a certain class:

$$p(y|x, w) = \boxed{Ber}(y|\boxed{sigm}(w^T x))$$

Bernoulli, sigmoid… what are those??

Bernoulli Random Variable



Sigmoid/ Logistic Function

We want a function f s.t. range(f) ϵ [0, 1] $\forall$ X

Logistic Function (Ct'd)

$$P(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

$$= \frac{1}{1 + e^{-(\beta_0+\beta_1)X}}$$

$$\frac{P(X)}{1-P(X)} = e^{\beta_0+\beta_1 X}$$

$$\boxed{\ln\left(\frac{P(X)}{1-P(X)}\right)} = \beta_0 + \beta_1 X$$

The "logit/ log-odds" function is linear in X

Computing Regression Coefficients

Although we can use the least-squares method and estimate using our training data, we prefer the maximum likelihood approach due to its better statistical properties (out of the scope of the course).

$$\ell(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'})).$$

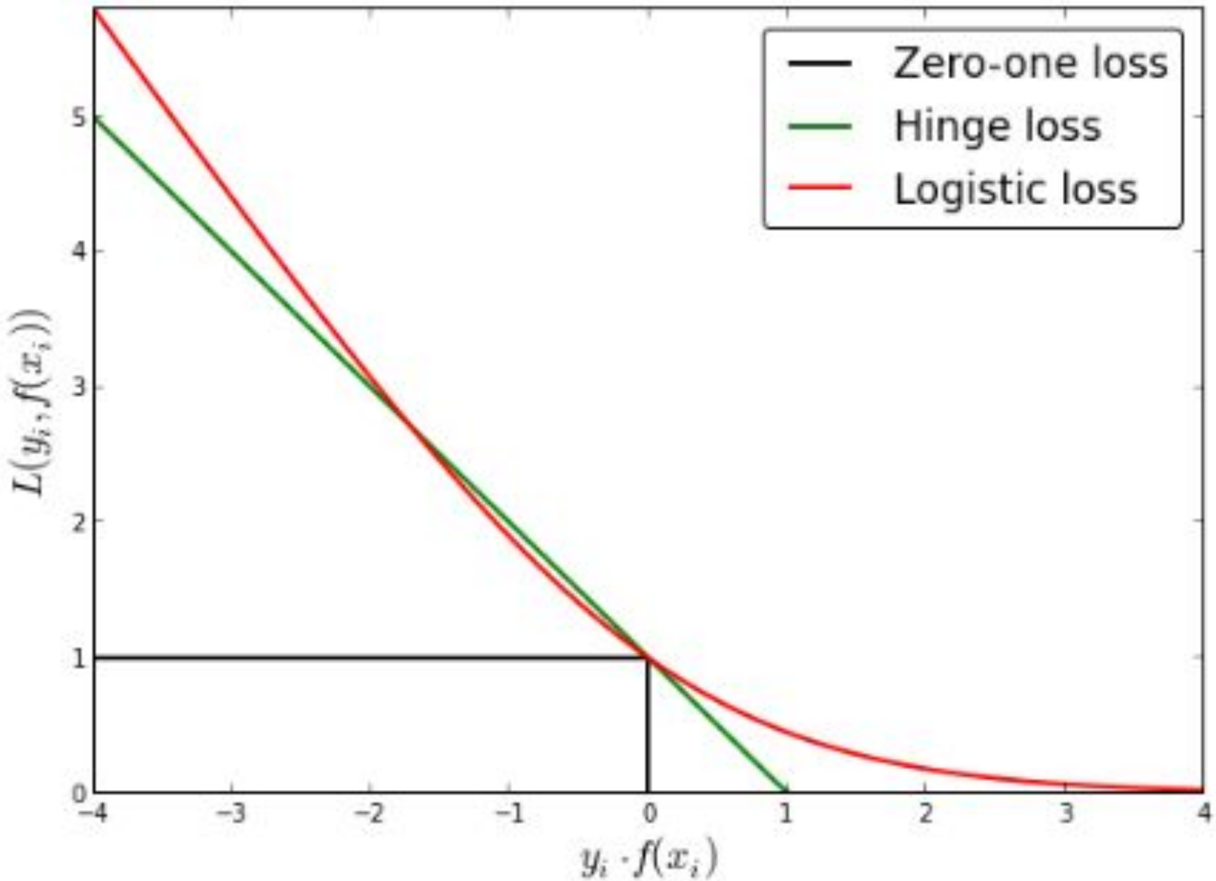
:

We seek to minimize logistic loss:

$$J(b) = -\sum_{i=1}^{m} \left(y^{(i)} \cdot \ln z^{(i)} + (1 - y^{(i)}) \cdot \ln (1 - z^{(i)}) \right)$$

$$z = h(x) = \frac{1}{1+e^{-\vec{b}^T \vec{x}}}$$

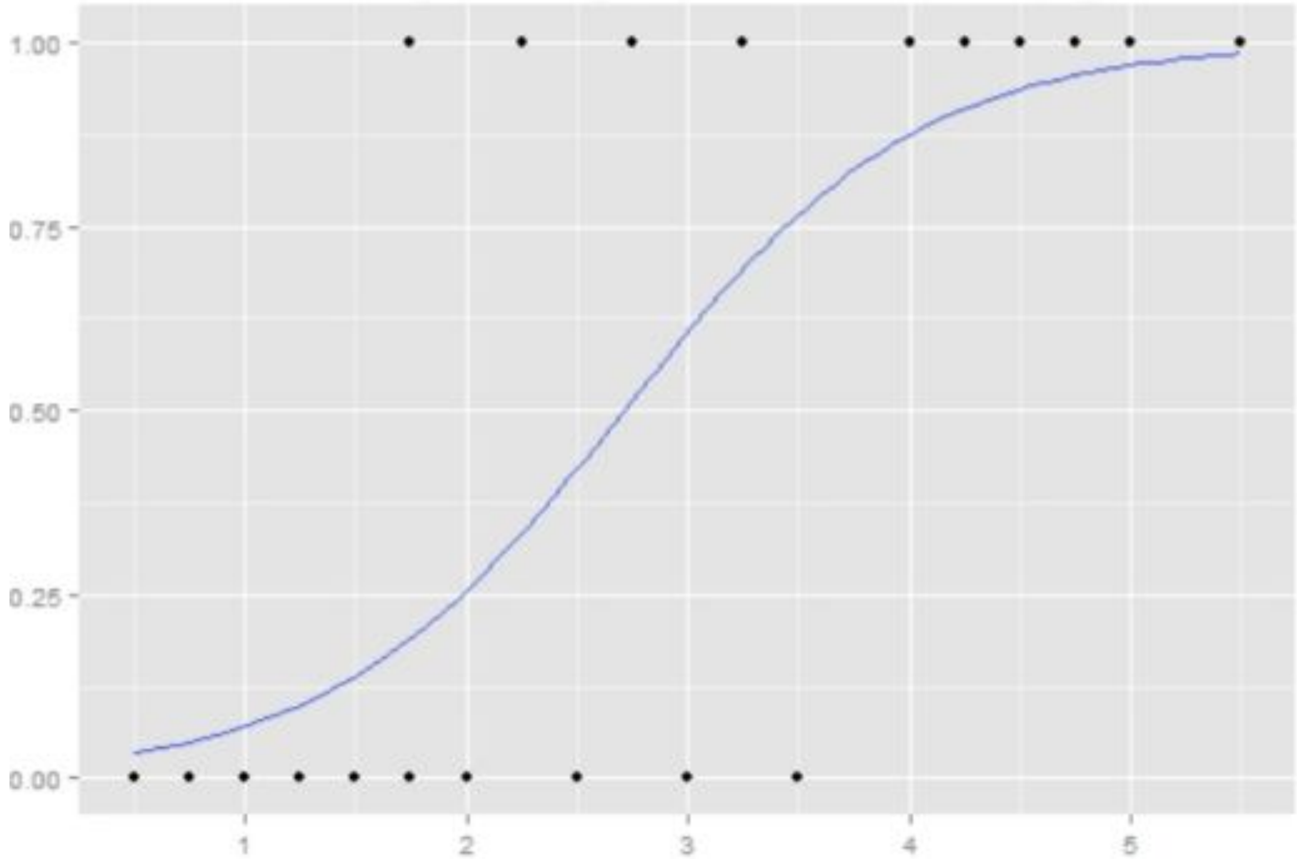
Logistic Loss



$$J(b) = -\sum_{i=1}^{m} \left(y^{(i)} \cdot \ln z^{(i)} + (1 - y^{(i)}) \cdot \ln (1 - z^{(i)}) \right)$$

Image from Stack Exchange

Coefficients should return something like this…



SUMMARY

	Linear	Logistic
Label Type	Continuous	Categorical
Problem Type	Actual Regression	Actually Classification
Hypothesis	$\theta^T x$	$s(\theta^T x)$
Loss	Mean Squared	Logistic
Analytical Solution	Yes	No

Thanks!

Any questions?

<u>Reminders</u>:

Homework 1 and deliverable 1 due before next lecture.