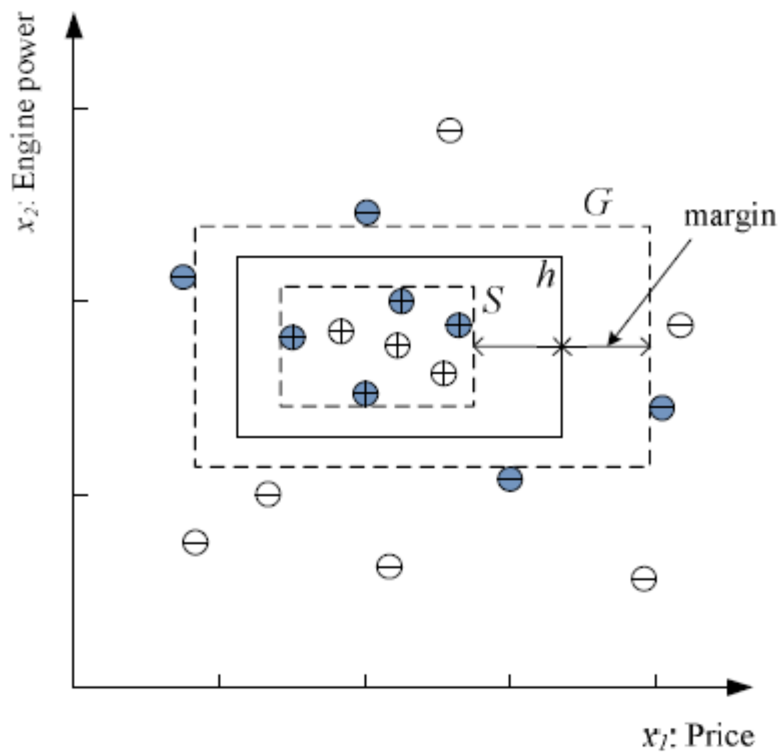
CS60020: Foundations of Algorithm Design and Machine Learning

Sourangshu Bhattacharya

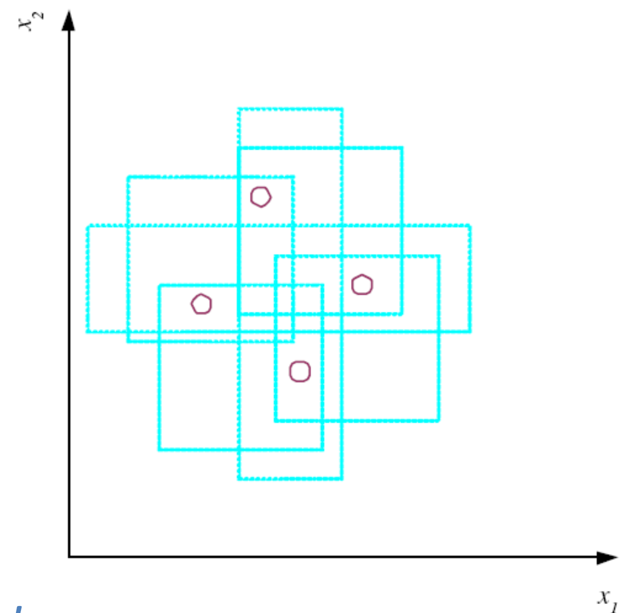
Margin

- Choose *h* with largest margin



VC Dimension

- N points can be labeled in 2^N ways as +/–
- $\mathcal{H}$ shatters N if there exists $h \in \mathcal{H}$ consistent for any of these: $VC(\mathcal{H}) = N$

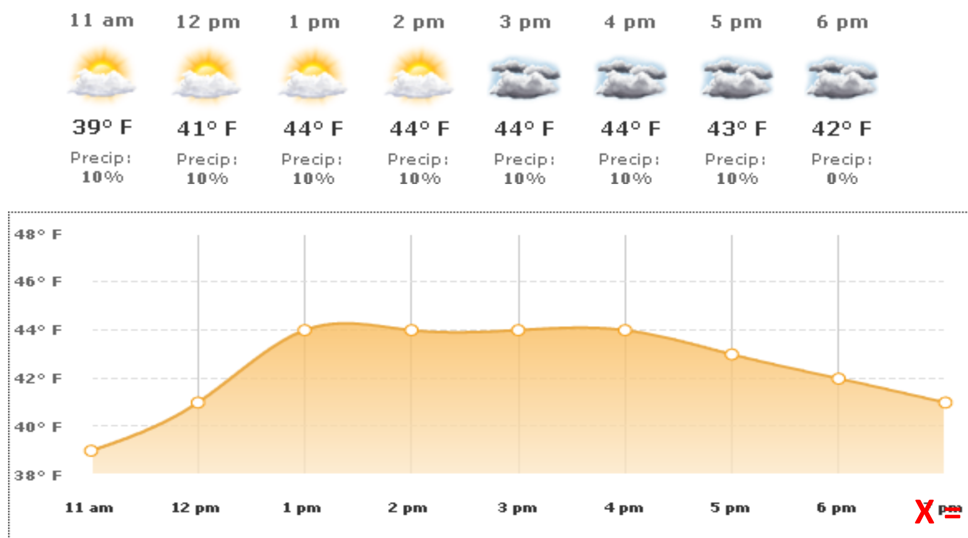


An axis-aligned rectangle shatters 4 points only !

STATISTICAL MACHINE LEARNING

Supervised Learning Tasks - Regression

Weather Prediction



Y = Temp

X = 7 pm

Estimating Contamination

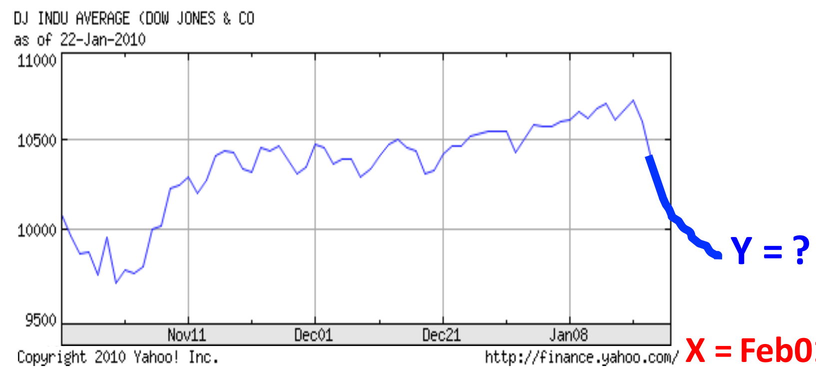
X = new location

Y = sensor reading

Supervised Learning

Goal: Construct a **predictor** $f : X \to Y$ to minimize loss function (performance measure)

Sports
Science
News



Classification:

$$P(f(X) \neq Y)$$

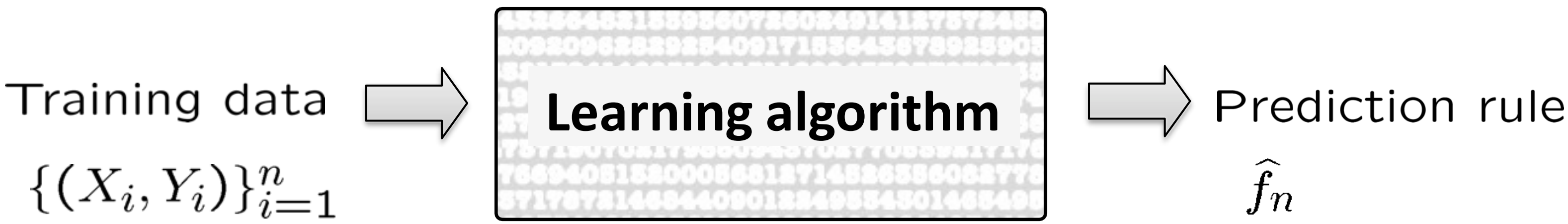
Probability of Error

Regression:

$$\mathbb{E}[(f(X) - Y)^2]$$

Mean Squared Error

Regression algorithms



Linear Regression

Replace Expectation with Empirical Mean

Optimal predictor: $$f^* = \arg\min_f \mathbb{E}[(f(X) - Y)^2]$$

Empirical Minimizer: $$\widehat{f}_n = \arg\min_{f \in \mathcal{F}} \frac{1}{n}\sum_{i=1}^{n}(f(X_i) - Y_i)^2$$

Empirical mean

Law of Large Numbers:

$$\frac{1}{n}\sum_{i=1}^{n}[\text{loss}(Y_i, f(X_i))] \xrightarrow{\boxed{n \longrightarrow \infty}} \mathbb{E}_{XY}[\text{loss}(Y, f(X))]$$

Restrict class of predictors

Optimal predictor: $$f^* = \arg\min_f \mathbb{E}[(f(X) - Y)^2]$$

Empirical Minimizer: $$\widehat{f}_n = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (f(X_i) - Y_i)^2$$

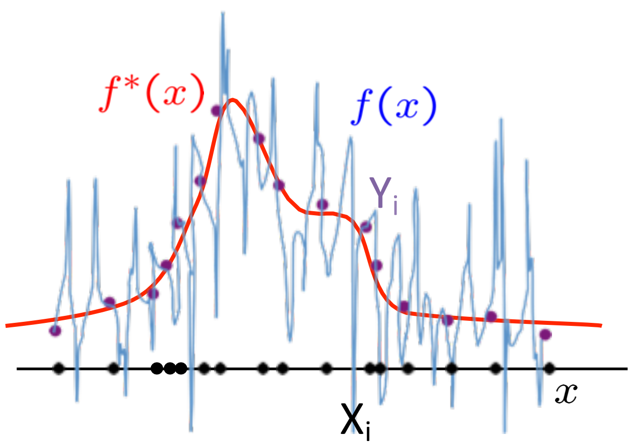
Class of predictors

Why?

Overfitting!

Empiricial loss minimized by any function of the form

$$f(x) = \begin{cases} Y_i, & x = X_i \text{ for } i = 1, \ldots, n \\ \text{any value}, & \text{otherwise} \end{cases}$$



Restrict class of predictors

Optimal predictor: $$f^* = \arg\min_f \mathbb{E}[(f(X) - Y)^2]$$

Empirical Minimizer: $$\widehat{f}_n = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (f(X_i) - Y_i)^2$$

Class of predictors

- Class of Linear functions
- Class of Polynomial functions
- Class of nonlinear functions

Linear Regression

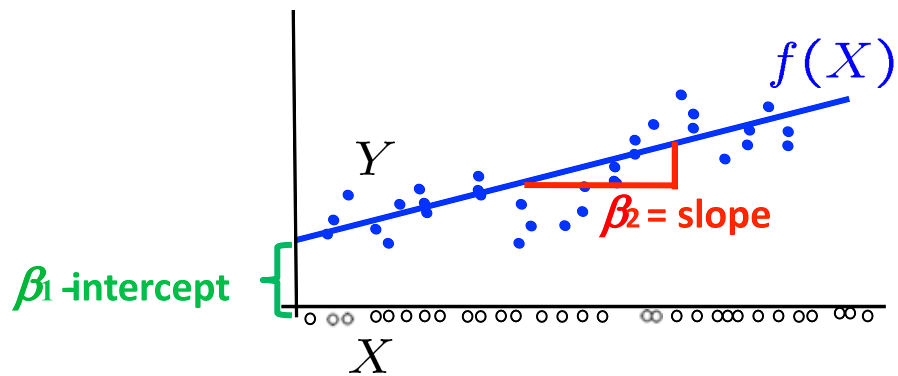
$$\widehat{f}_n^L = \arg\min_{f \in \mathcal{F}_L} \frac{1}{n} \sum_{i=1}^{n} (f(X_i) - Y_i)^2$$

Least Squares Estimator

$\mathcal{F}_L$ -Class of Linear functions

Uni--variate case:

$$f(X) = \beta_1 + \beta_2 X$$



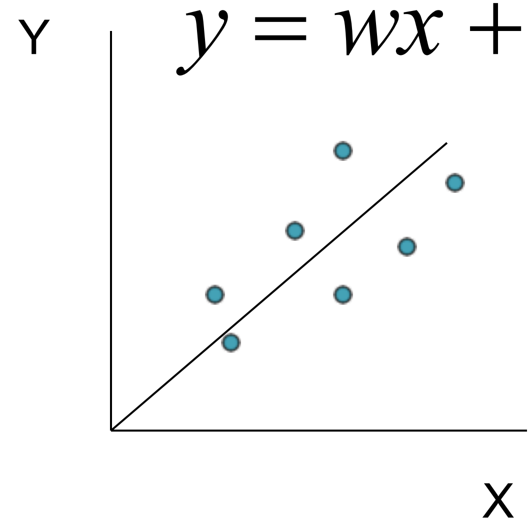
Multi--variate case:

$$f(X) = f(X^{(1)}, \ldots, X^{(p)}) = \beta_1 X^{(1)} + \beta_2 X^{(2)} + \cdots + \beta_p X^{(p)}$$

$$= X\beta \qquad \text{where} \quad X = [X^{(1)} \ldots X^{(p)}], \quad \beta = [\beta_1 \ldots \beta_p]^T$$

Linear regression in 1D

$$y = wx + \varepsilon$$



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

$$\arg\min_w \sum_i (y_i - wx_i)^2$$

- Why least squares?
  - minimizes squared distance between measurements and predicted line
  - the math is pretty

Solving Linear Regression in 1D

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

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

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

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

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

Least Squares Estimator

$$\widehat{f}_n^L = \arg\min_{f \in \mathcal{F}_L} \frac{1}{n} \sum_{i=1}^{n} (f(X_i) - Y_i)^2 \qquad f(X_i) = X_i \beta$$

$$\Downarrow$$

$$\widehat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} (X_i \beta - Y_i)^2 \qquad \widehat{f}_n^L(X) = X\widehat{\beta}$$

$$= \arg\min_{\beta} \frac{1}{n} (\mathbf{A}\beta - \mathbf{Y})^T (\mathbf{A}\beta - \mathbf{Y})$$

$$\mathbf{A} = \begin{bmatrix} X_1 \\ \vdots \\ X_n \end{bmatrix} = \begin{bmatrix} X_1^{(1)} & \dots & X_1^{(p)} \\ \vdots & \ddots & \vdots \\ X_n^{(1)} & \dots & X_n^{(p)} \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} \mathbf{Y}_1 \\ \vdots \\ \mathbf{Y}_n \end{bmatrix}$$

Least Squares Estimator

$$\widehat{\beta} = \arg\min_{\beta} \frac{1}{n}(\mathbf{A}\beta - \mathbf{Y})^T(\mathbf{A}\beta - \mathbf{Y}) = \arg\min_{\beta} J(\beta)$$

$$J(\beta) = (\mathbf{A}\beta - \mathbf{Y})^T(\mathbf{A}\beta - \mathbf{Y})$$

$$\left.\frac{\partial J(\beta)}{\partial \beta}\right|_{\widehat{\beta}} = 0$$

Normal Equations

$$(\mathbf{A}^T\mathbf{A})\widehat{\beta} = \mathbf{A}^T\mathbf{Y}$$

p xp p x1 p x1

If $(\mathbf{A}^T\mathbf{A})$ is invertible,

$$\boxed{\widehat{\beta} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{Y} \qquad \widehat{f}_n^L(X) = X\widehat{\beta}}$$

When is $(\mathbf{A}^T\mathbf{A})$ invertible ?
Recall: Full rank matrices are invertible.

What if $(\mathbf{A}^T\mathbf{A})$ is not invertible ?

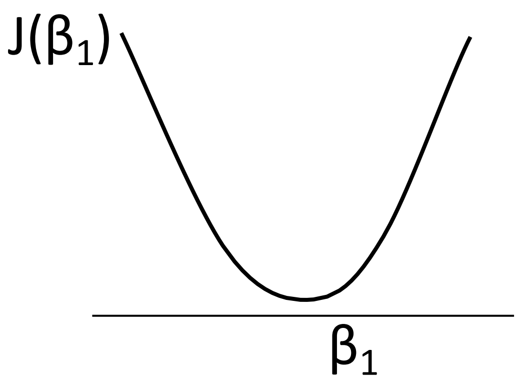
Gradient Descent

Even when $(\mathbf{A}^T\mathbf{A})$ is invertible, might be computationally expensive if **A** is huge.

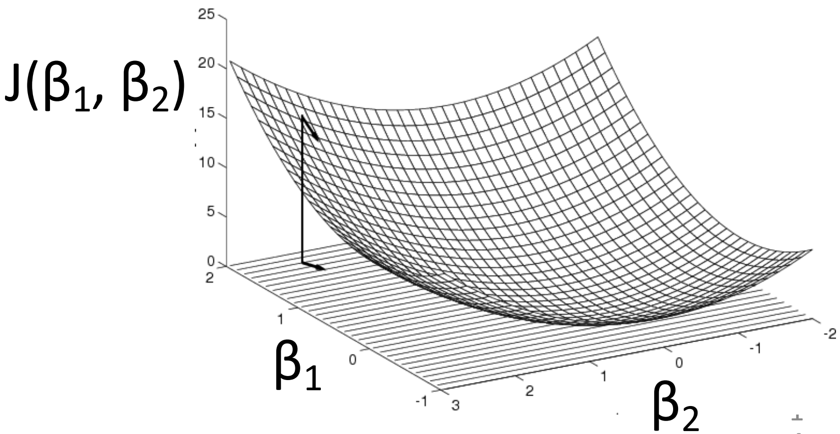
$$\widehat{\beta} = \arg\min_{\beta} \frac{1}{n}(\mathbf{A}\beta - \mathbf{Y})^T(\mathbf{A}\beta - \mathbf{Y}) = \arg\min_{\beta} J(\beta)$$

Treat as optimization problem

<u>Observation:</u> J(β) is convex in β.



How to find the minimizer?



Gradient Descent

Even when $(\mathbf{A}^T\mathbf{A})$ is invertible, might be computationally expensive if **A** is huge.

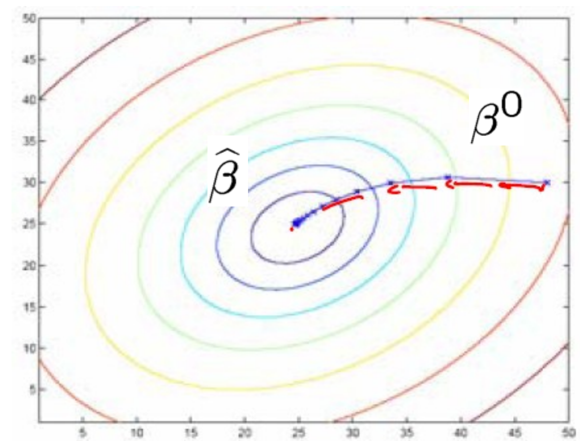
$$\widehat{\beta} = \arg\min_{\beta} \frac{1}{n}(\mathbf{A}\beta - \mathbf{Y})^T(\mathbf{A}\beta - \mathbf{Y}) = \arg\min_{\beta} J(\beta)$$

Since J(β) is convex, move along negative of gradient

Initialize: β^0

step size

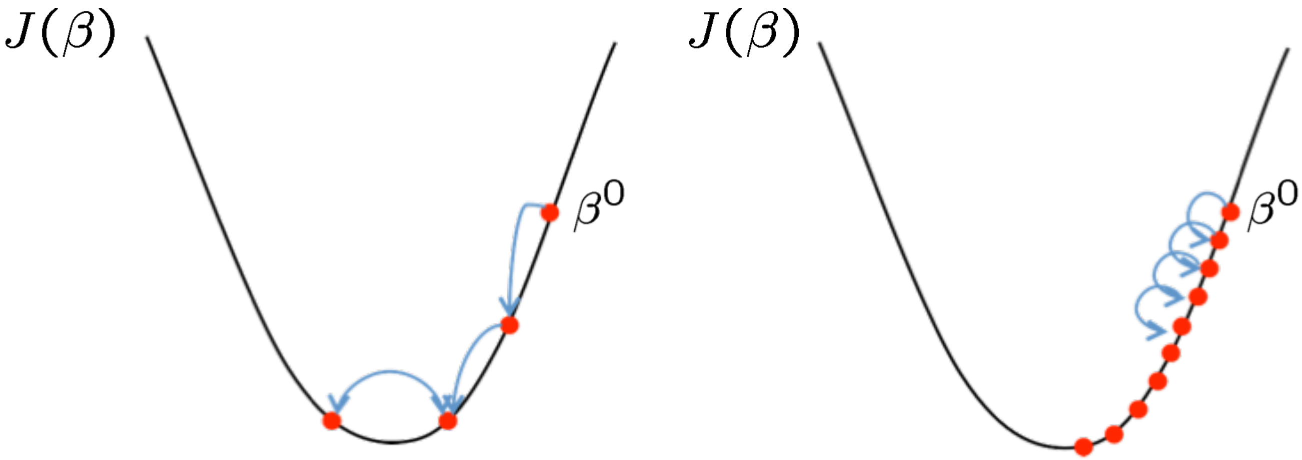
Update:
$$\begin{aligned}\beta^{t+1} &= \beta^t - \frac{\alpha}{2}\frac{\partial J(\beta)}{\partial \beta}\Big|_t \\ &= \beta^t - \alpha\ \mathbf{A}^T\underbrace{(\mathbf{A}\beta^t - Y)}_{0 \text{ if } \widehat{\beta} = \beta^t}\end{aligned}$$



Stop: when some criterion met e.g. fixed # iterations, or $\left.\frac{\partial J(\beta)}{\partial \beta}\right|_{\beta^t} < \varepsilon$.

Effect of step--size

α



Large α => Fast convergence but larger residual error
Also possible oscillations

Small α => Slow convergence but small residual error

Stochastic Gradient Descent

- Gradient descent (also known as Batch Gradient Descent) computes the gradient using the whole dataset
- Stochastic Gradient Descent computes the gradient using a single sample (or a mini-batch).

Non-Linear basis function

- So far we only used the observed values $x_1, x_2, \ldots$
- 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 parameters are still linear in the data and the problem remains a multi-variate linear regression problem

$$y = w_0 + w_1 x_1^2 + \ldots + w_k x_k^2 + \varepsilon$$

Non-linear basis functions

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

- Polynomial: $\phi_j(x) = x^j$ for j=0 … n

- Gaussian: $$\phi_j(x) = \frac{(x - \mu_j)}{2\sigma_j^2}$$

- Sigmoid: $$\phi_j(x) = \frac{1}{1 + \exp(-s_j x)}$$

- Logs: $$\phi_j(x) = \log(x+1)$$

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

Slide courtesy of William Cohen

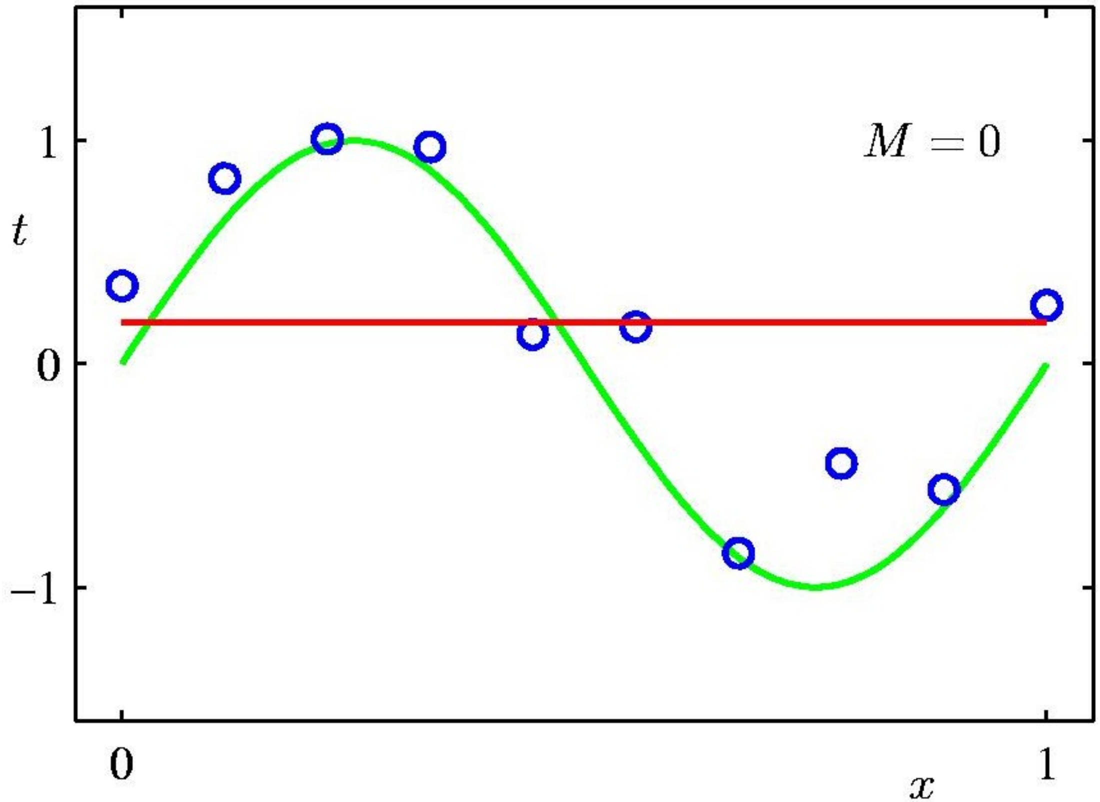
General linear regression problem

- 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 $\phi_j(\mathbf{x})$ can be either x_j for multivariate regression or one of the non-linear basis functions we defined
- … and $\phi_0(\mathbf{x})=1$ for the intercept term

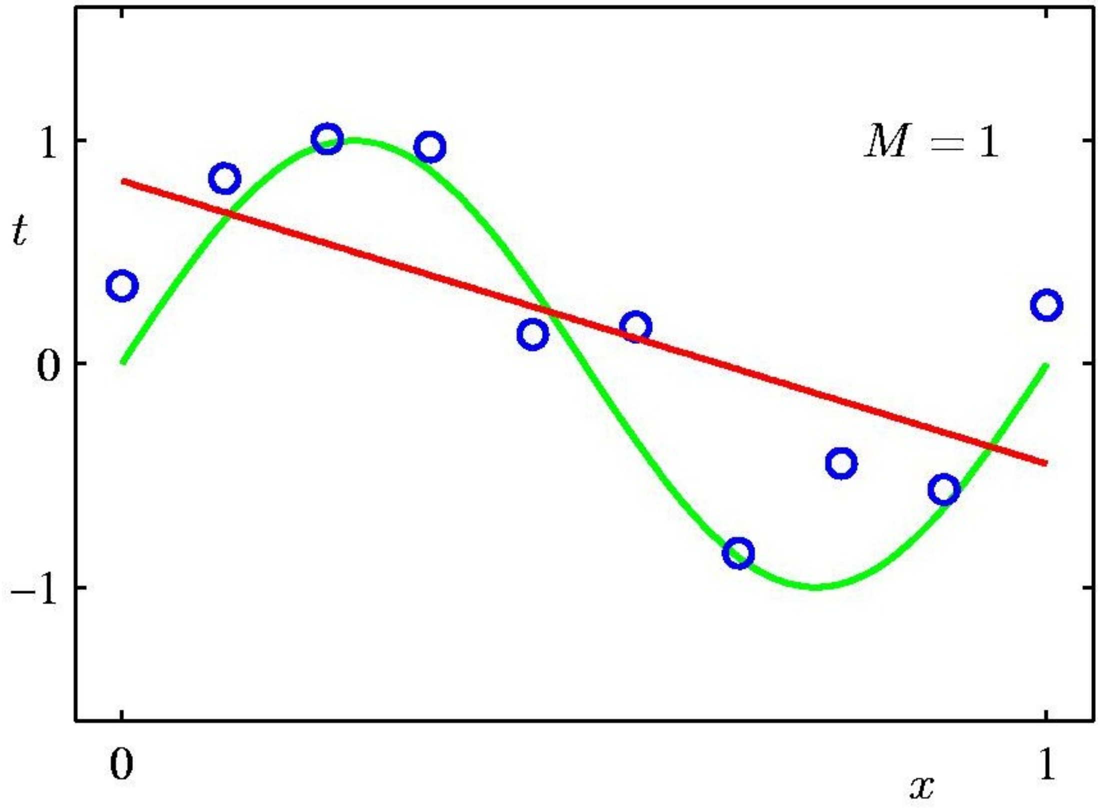
0^{th} Order Polynomial



n=10

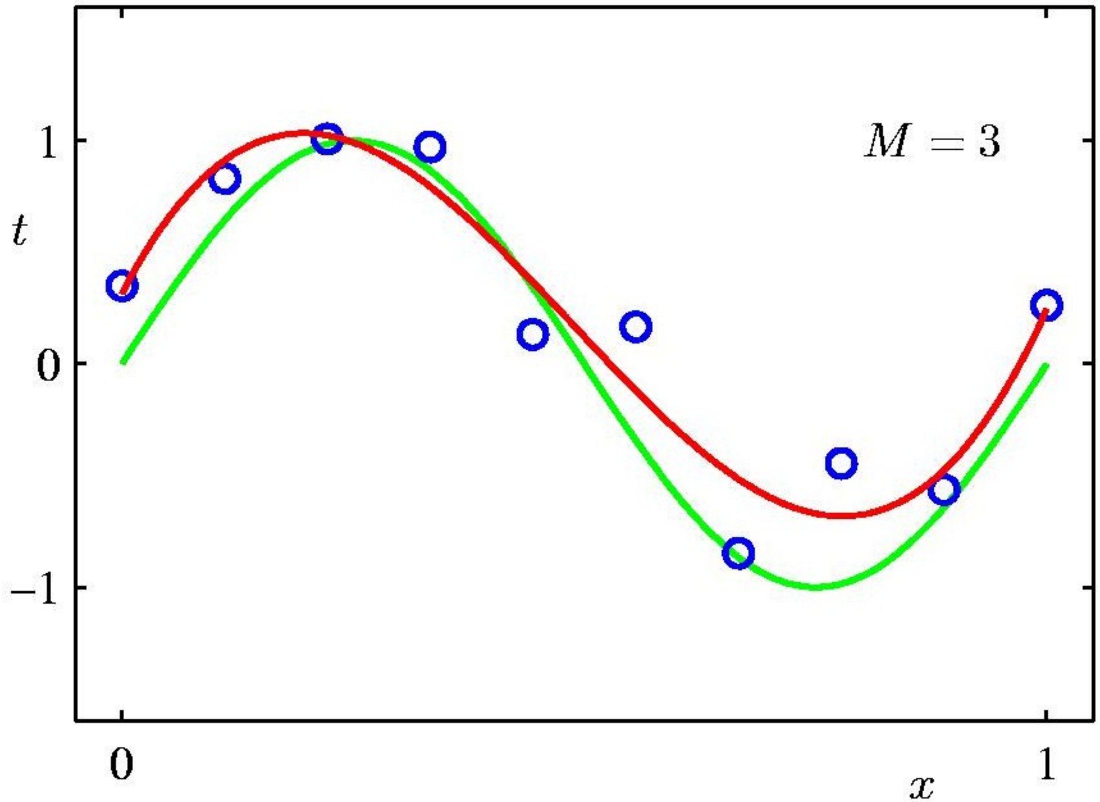
Slide courtesy of William Cohen

1st Order Polynomial

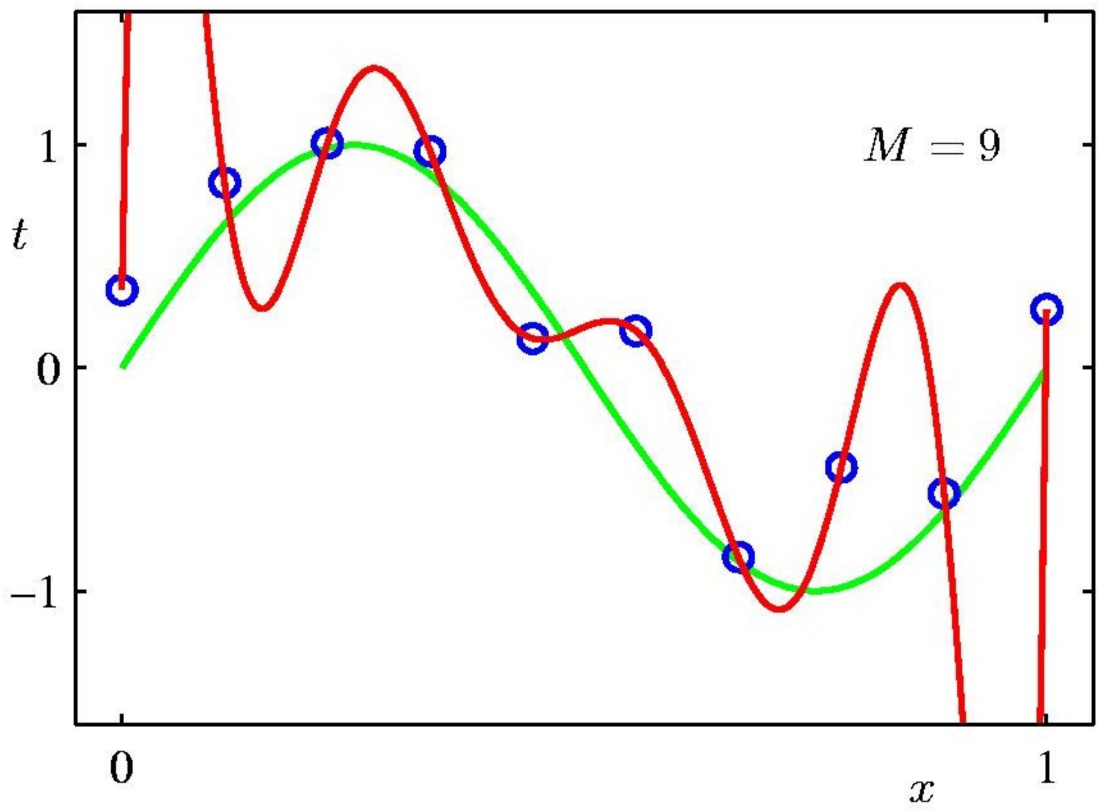


Slide courtesy of William Cohen

3rd Order Polynomial

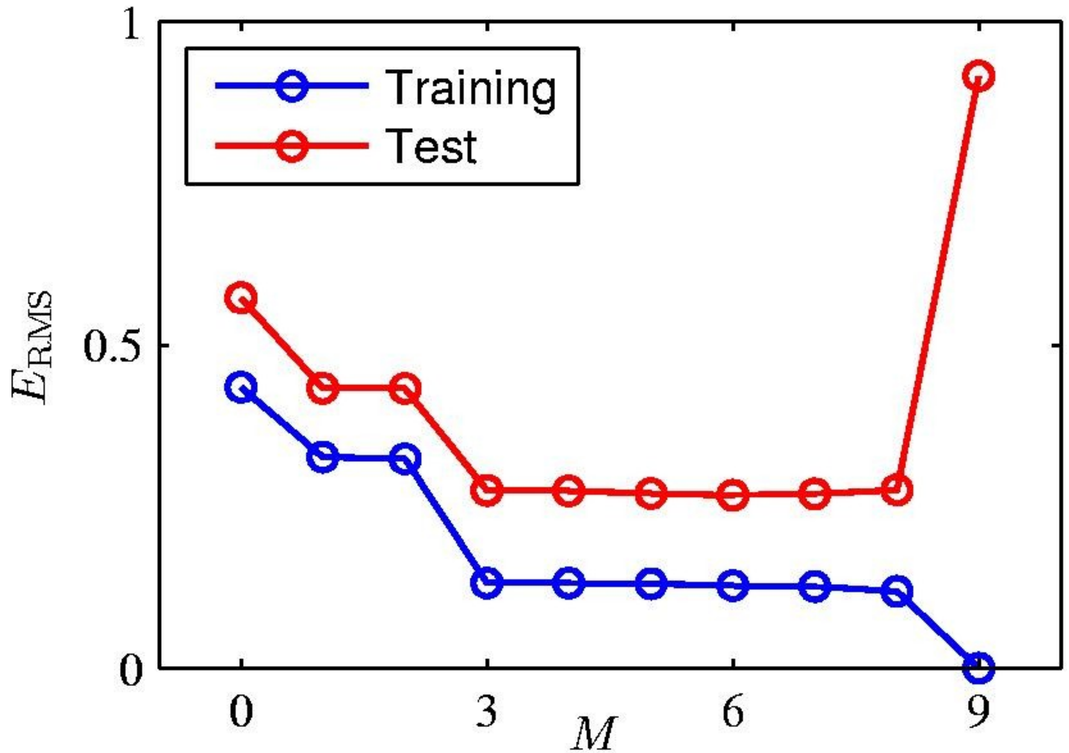


9^{th} Order Polynomial



Slide courtesy of William Cohen

Over-fitting



Root-Mean-Square (RMS) Error

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

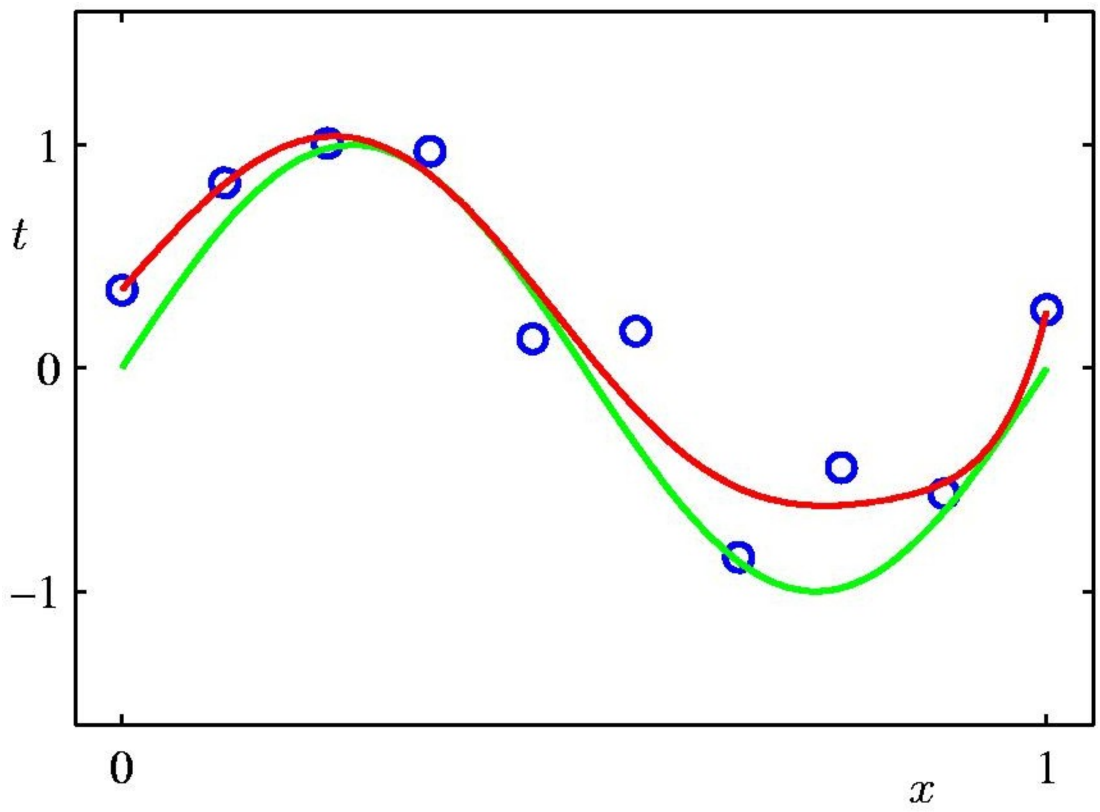
Regularization

Penalize large coefficient values

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

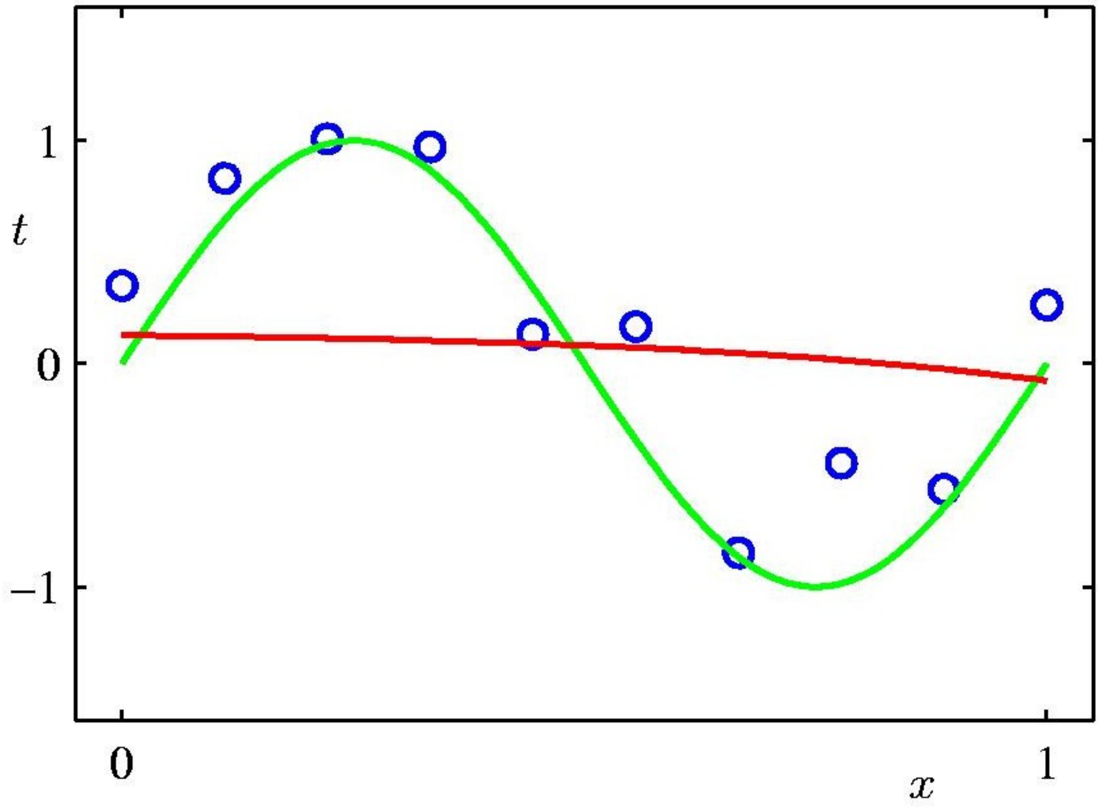
Regularization:

$$\ln \lambda = -18$$



Slide courtesy of William Cohen

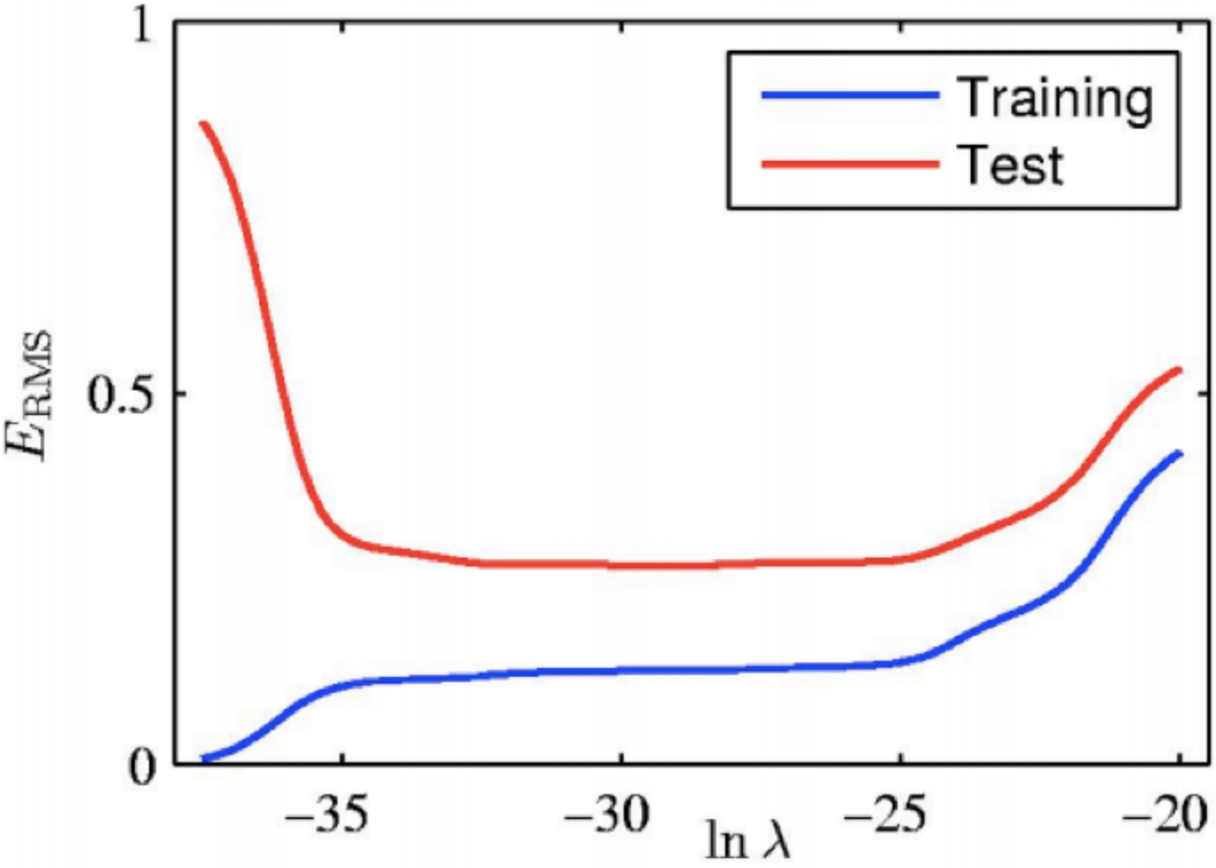
Over Regularization



Slide courtesy of William Cohen

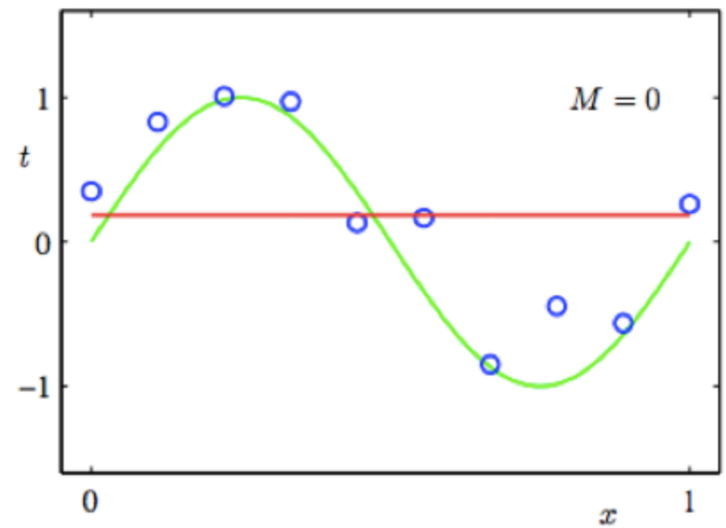
Regularization

9th Order Polynomial

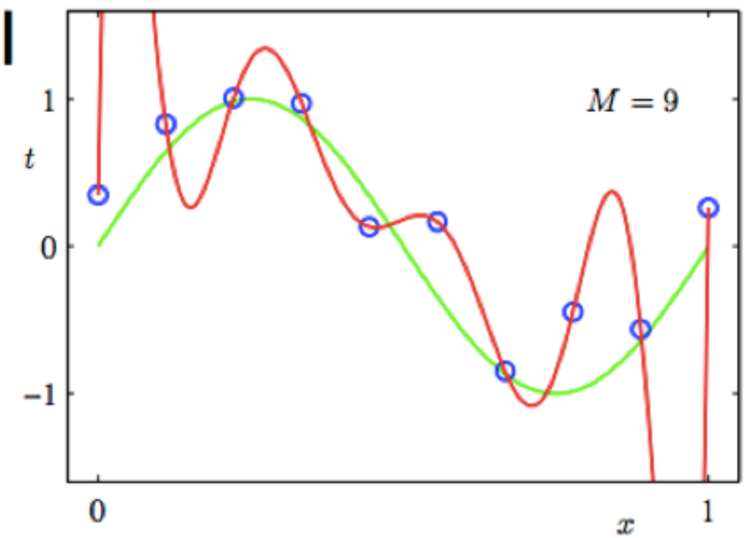


Bias-Variance Tradeoff

- Model too simple: does not fit the data well
  - A *biased* solution



- Model too complex: small changes to the data, solution changes a lot
  - A *high-variance* solution



Effect of Model Complexity

- If we allow very complicated predictors, we could overfit the training data.