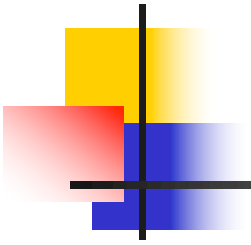
CSx090: Linear Regression

Spring 2018

Mehdi Allahyari
Georgia Southern University

(most of the slides borrowed from Emily Fox, Tom Mitchell and Ali Farhadi

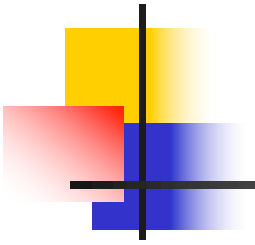


Regression

So far, we've been interested in learning P(Y|X) where Y has discrete values (called 'classification')

What if Y is continuous? (called 'regression')

- predict weight from gender, height, age, …
- predict Google stock price today from Google, Yahoo, MSFT prices yesterday
- predict each pixel intensity in robot's current camera image, from previous image and previous action



Regression

Wish to learn f:X→Y, where Y is real, given $\{<x^1,y^1>\ldots<x^n,y^n>\}$

Approach:

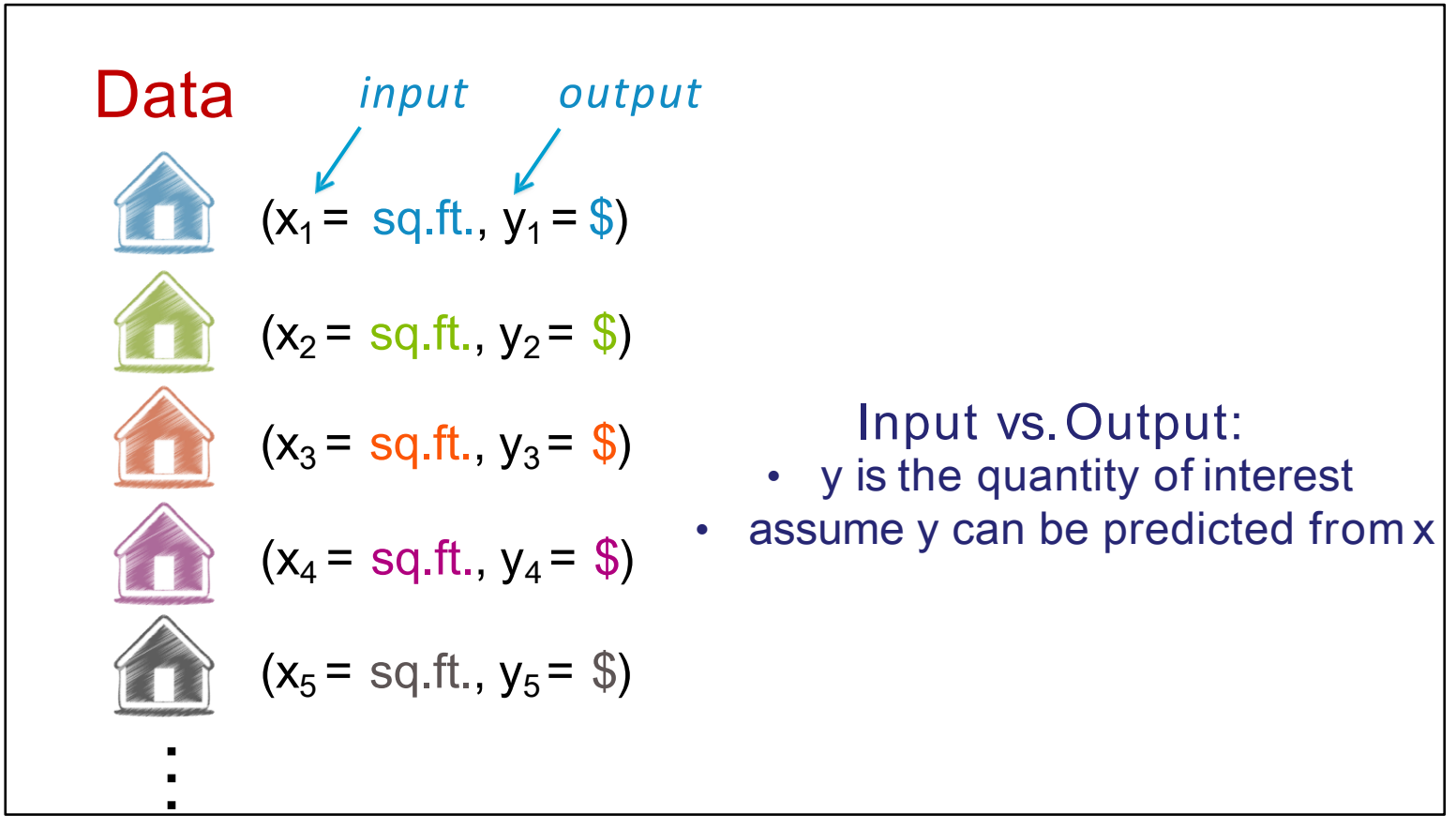
1. choose some parameterized form for P(Y|X; θ)
   (θ is the vector of parameters)

2. derive learning algorithm as MLE or MAP estimate for θ

We'll get back to this slide!

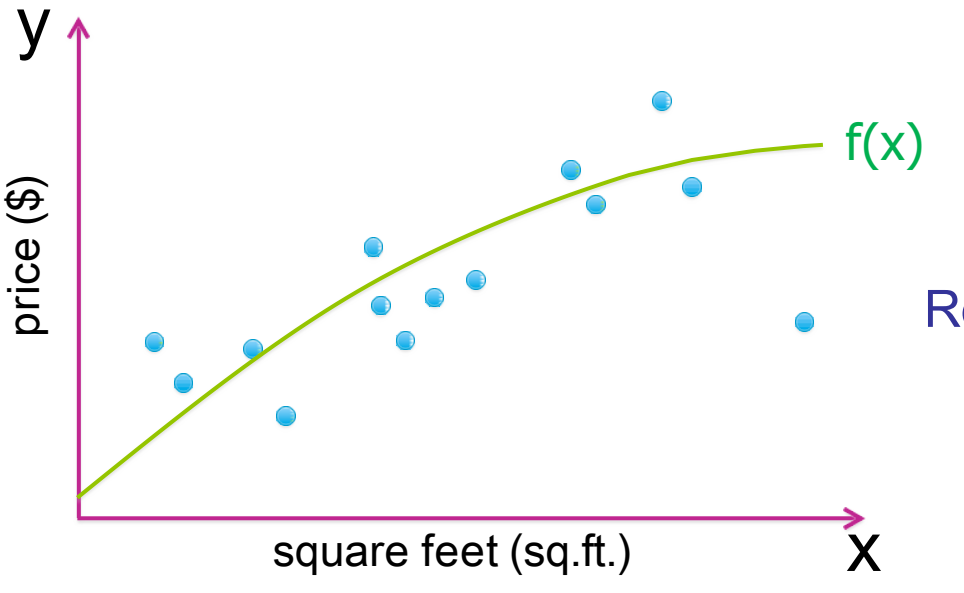
How much is my house worth?

How much is my house worth?


Data
input
output
$(x_1 =$ sq.ft., $y_1 =$ $)
$(x_2 =$ sq.ft., $y_2 =$ $)
$(x_3 =$ sq.ft., $y_3 =$ $)
$(x_4 =$ sq.ft., $y_4 =$ $)
$(x_5 =$ sq.ft., $y_5 =$ $)
Input vs. Output:
• y is the quantity of interest
• assume y can be predicted from x

Model

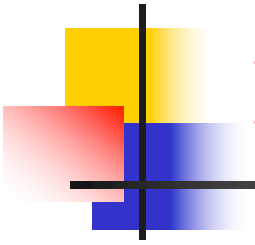
How we *assume* the world works



Regression model:

$y_i = f(x_i) + \epsilon_i$

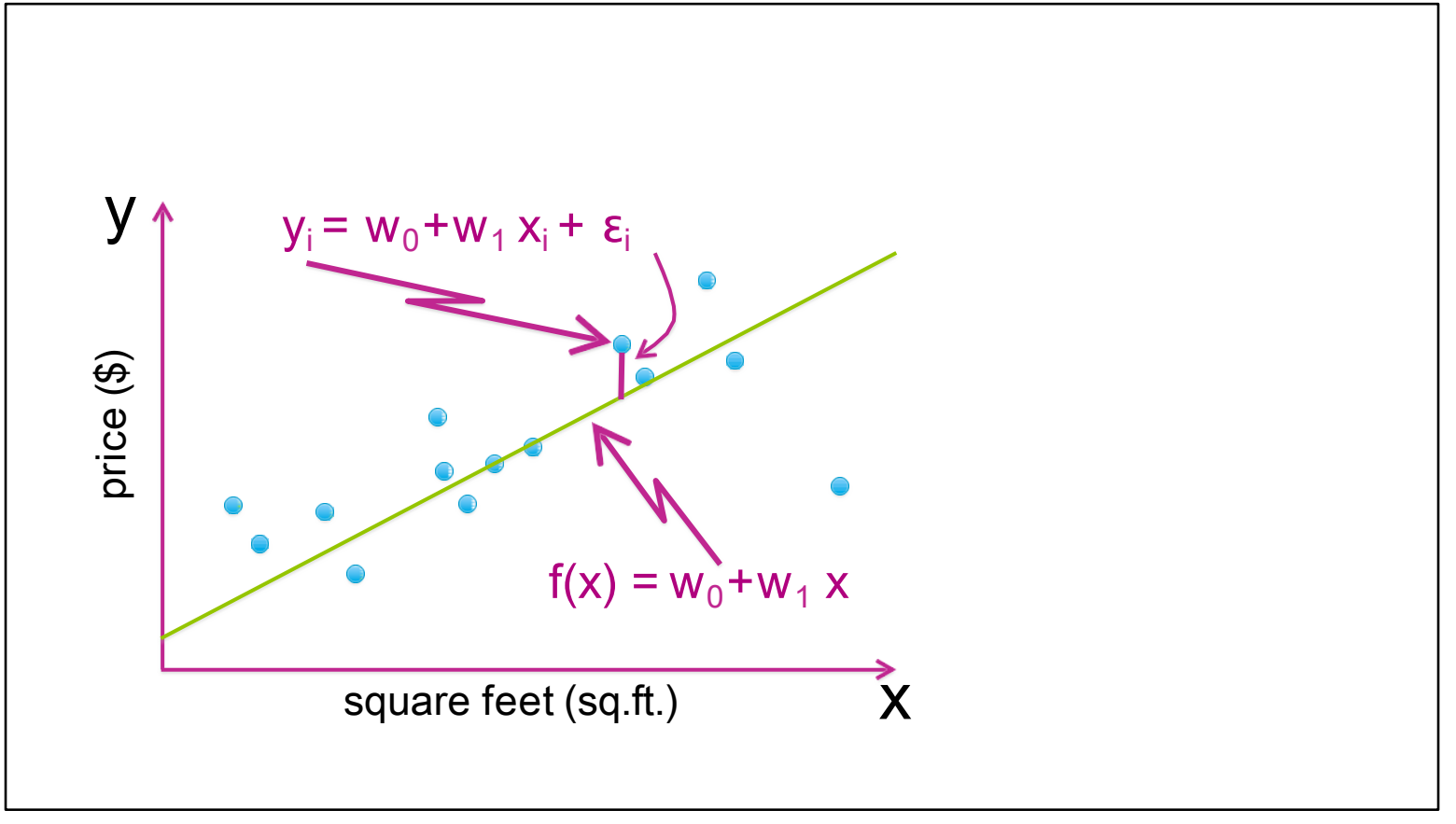
where $\epsilon \sim N(0, \sigma)$



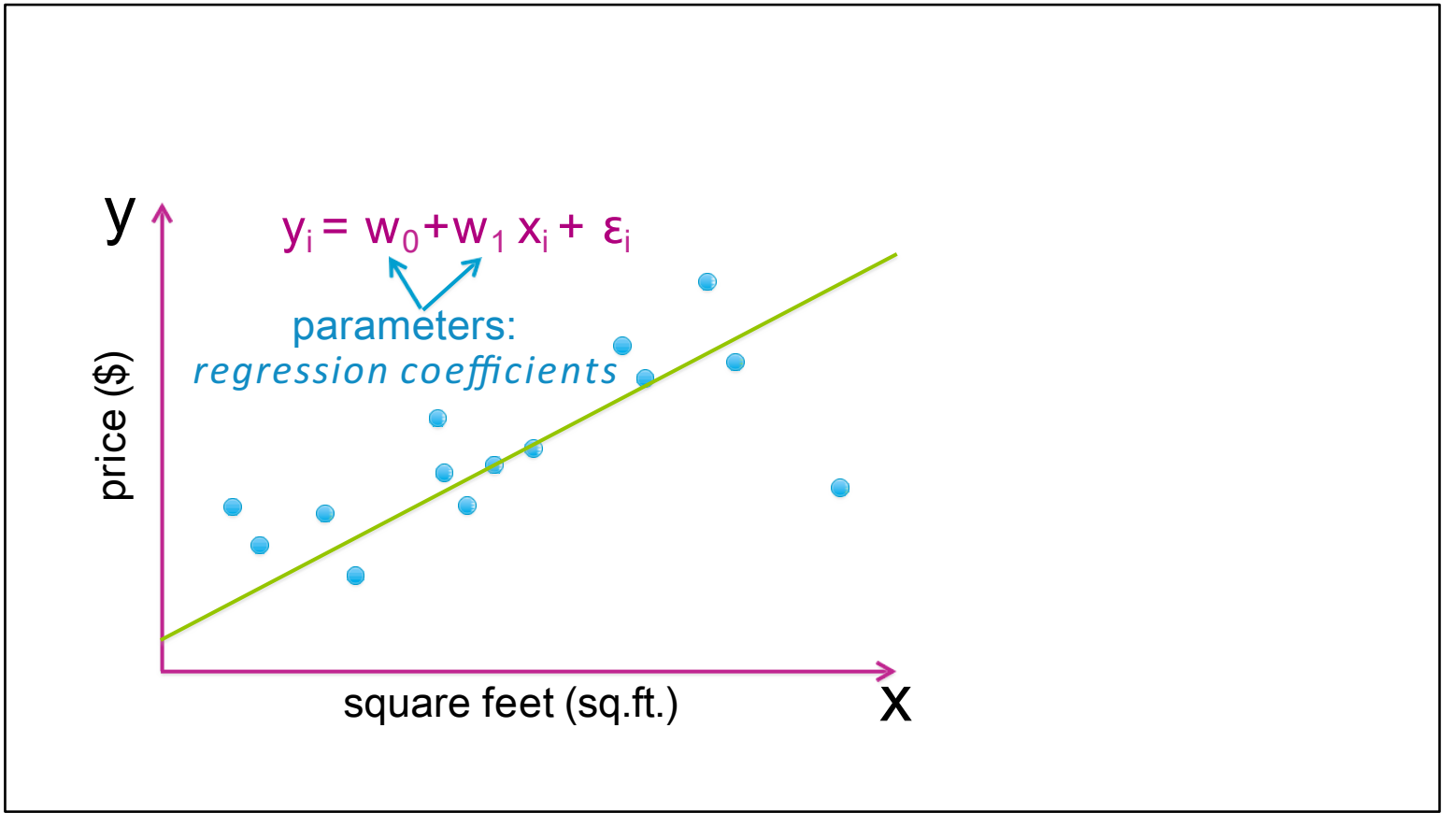
Noise

- A simple model typically does not exactly fit the data – lack of fit can be considered noise
- Sources of noise:
  - Imprecision in data attributes (input noise)
  - Errors in data targets (mislabeling)
  - Additional attributes not taken into account by data attributes, affect target values (latent variables)
  - Model may be too simple to account for data targets

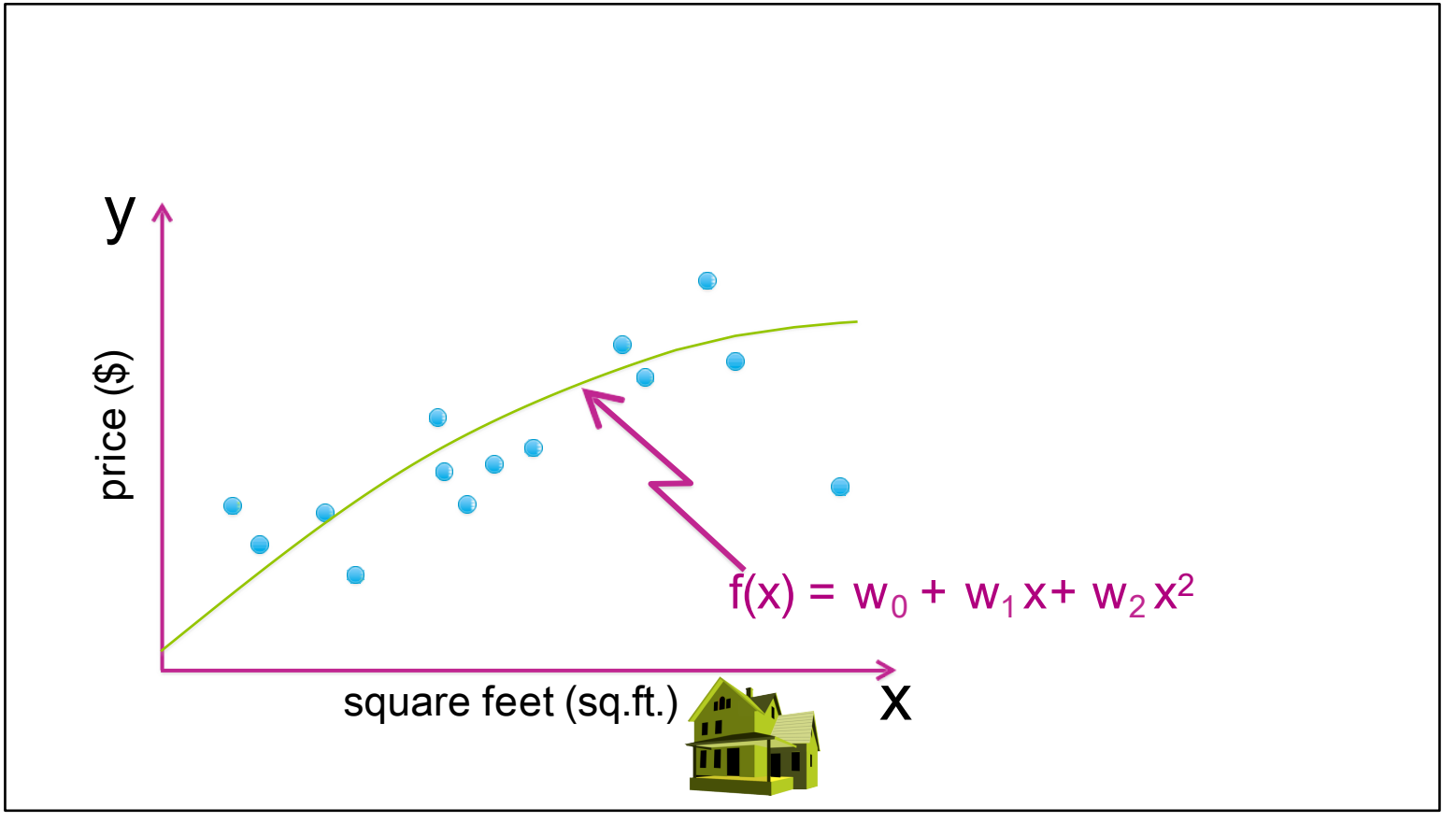
Simple linear regression model



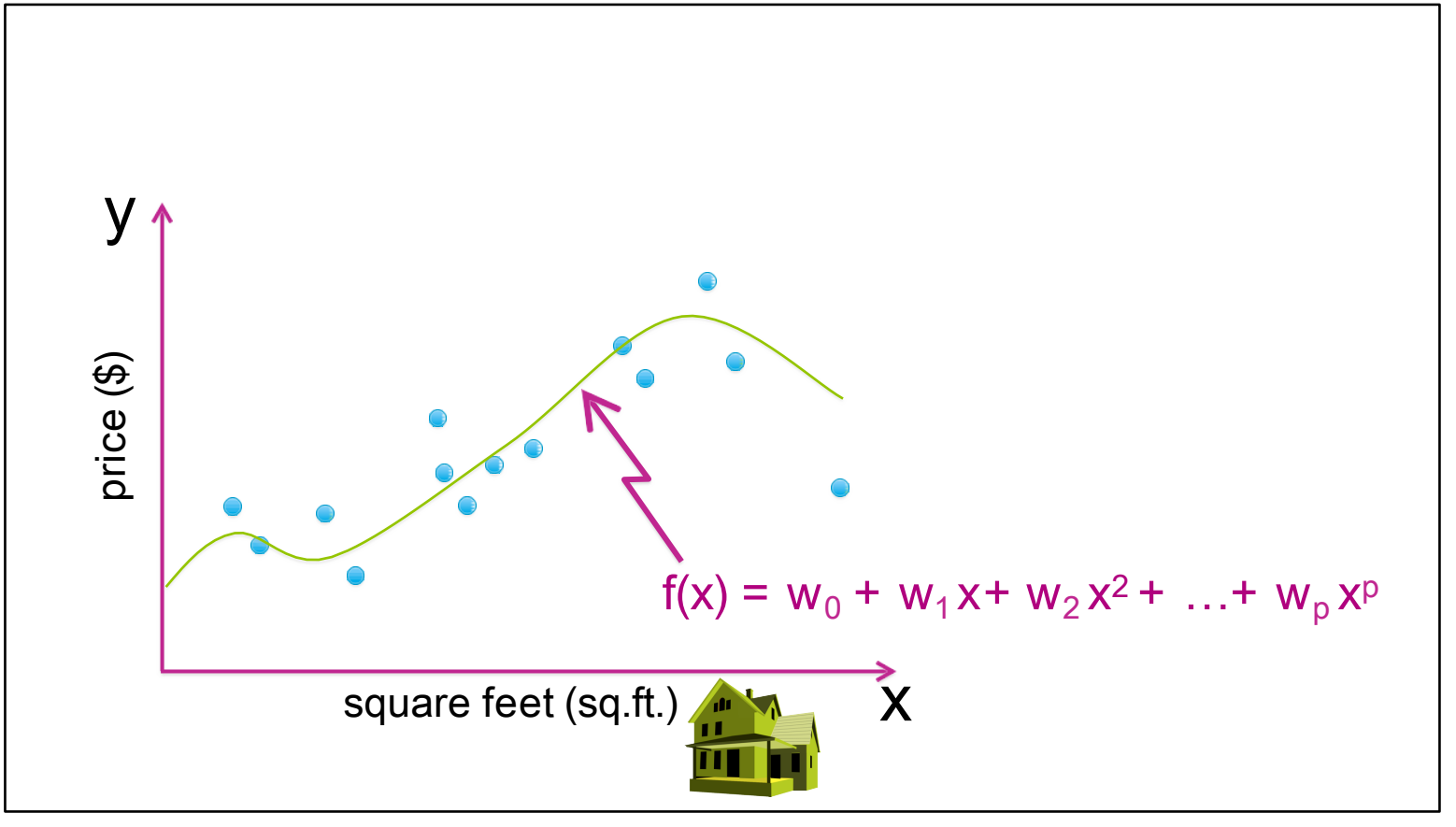
Simple linear regression model



What about a quadratic function?



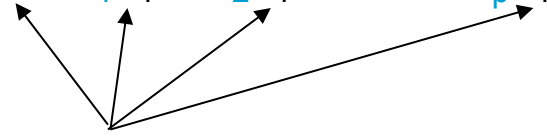
Even higher order polynomial



Polynomial regression

Model:

$$y_i = w_0 + w_1 x_i + w_2 x_i^2 + \ldots + w_p x_i^P + \varepsilon_i$$



treat as different features

feature 1 = 1 (constant) *parameter* 1 = w_0

feature 2 = x *parameter* 2 = w_1

feature 3 = x^2 *parameter* 3 = w_2

…

feature p+1 = x^p *parameter* p+1 = w_p

Generic basis expansion

Model:

$$y_i = w_0 h_0(x_i) + w_1 h_1(x_i) + \ldots + w_D h_D(x_i) + \varepsilon_i$$

$$= \sum_{j=0}^{D} w_j h_j(x_i) + \varepsilon_i$$

j^{th} *feature*

j^{th} *regression coefficient* or *weight*

Generic basis expansion

Model:

$y_i = w_0 h_0(x_i) + w_1 h_1(x_i) + \ldots + w_D h_D(x_i) + \varepsilon_i$

$= \sum_{j=0}^{D} w_j h_j(x_i) + \varepsilon_i$

feature 1 = $h_0(x)$…often 1 (constant)

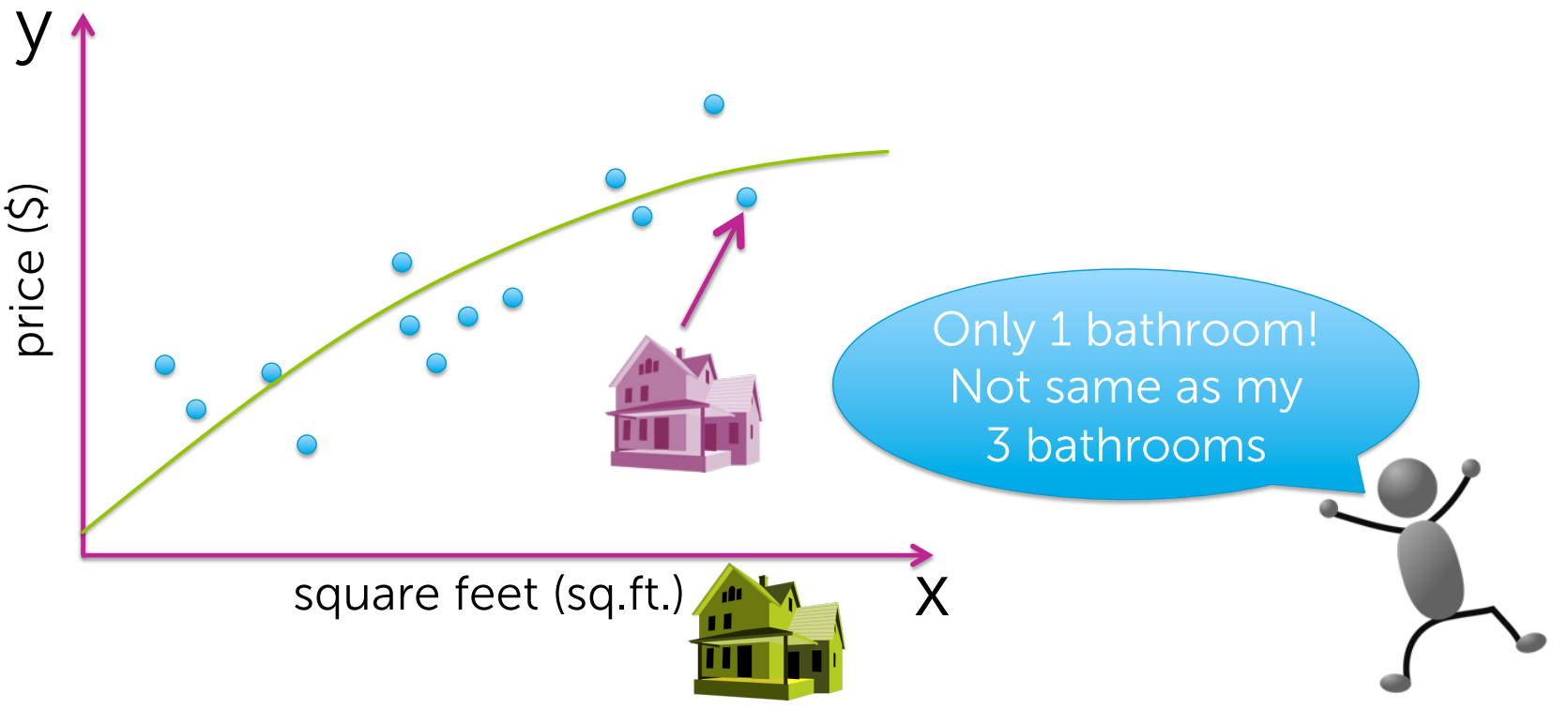
feature 2 = $h_1(x)$… e.g., x

feature 3 = $h_2(x)$… e.g., x^2 or $\sin(2\pi x/12)$

…

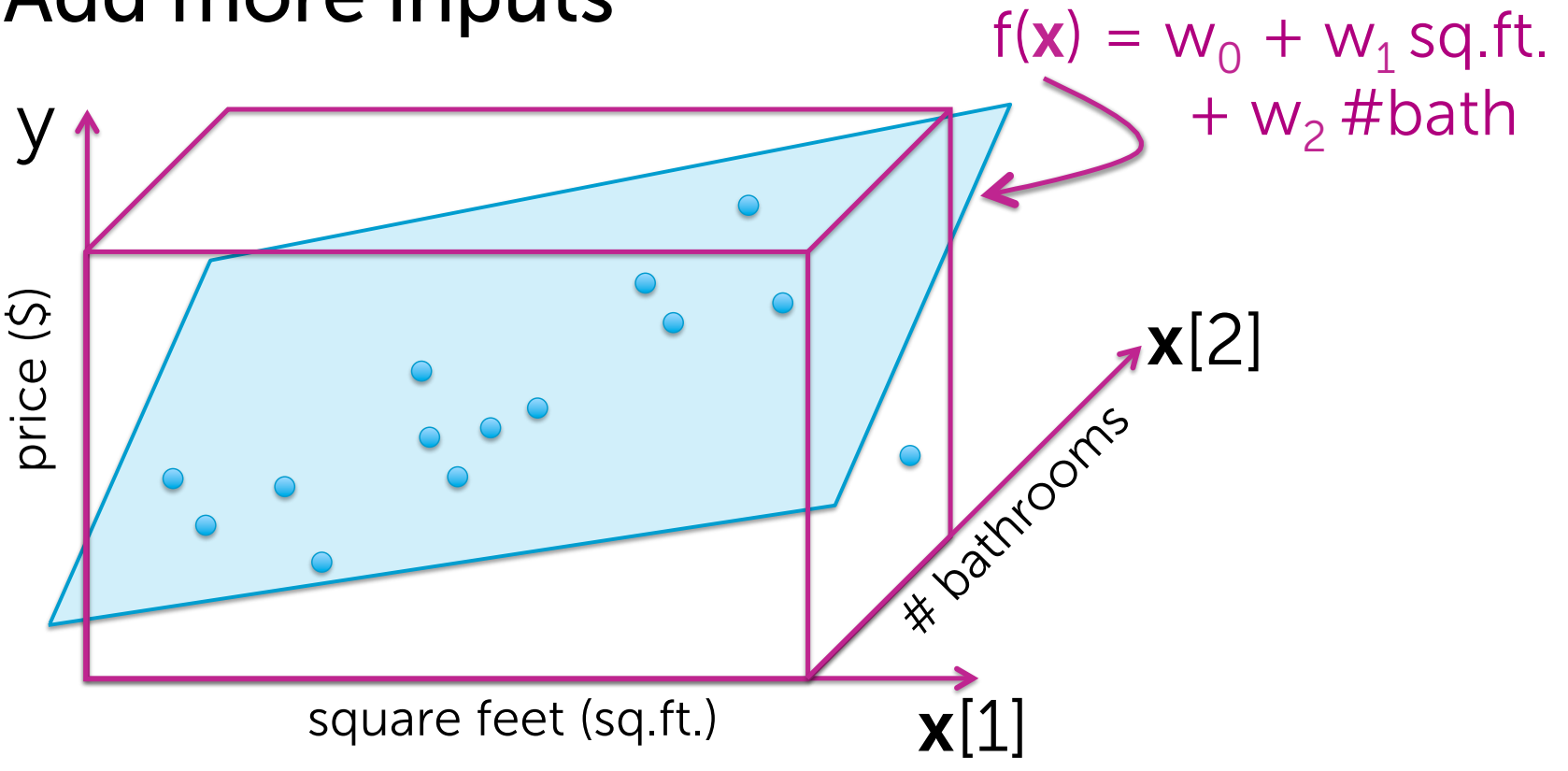
feature D+1 = $h_D(x)$… e.g., x^p

Predictions just based on house size



Add more inputs

Add more inputs



Many possible inputs

- Square feet
- # bathrooms
- # bedrooms
- Lot size
- Year built
- …

The Regression Problem

- Instances: <$\mathbf{x}_j$, t_j>
- Learn: Mapping from x to t($\mathbf{x}$)
- Hypothesis space:
  - Given, basis functions $\{h_1,\ldots,h_k\}$
  - $h_i(\mathbf{x}) \in \mathbb{R}$
  - Find coeffs $\mathbf{w}=\{w_1,\ldots,w_k\}$

$$H = \{h_1, \ldots, h_K\}$$

$$\underbrace{t(\mathbf{x})}_{\text{data}} \approx \hat{f}(\mathbf{x}) = \sum_i w_i h_i(\mathbf{x})$$

  - Why is this usually called *linear regression*?
    - model is linear in the parameters
    - Can we estimate functions that are not lines???

- Precisely, minimize the residual squared error:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

General Notation

Output: y ← scalar

Inputs: $\mathbf{x} = (\mathbf{x}[1], \mathbf{x}[2], \ldots, \mathbf{x}[d])$

↖ d-dim vector

Notational conventions:

- $\mathbf{x}[j]$ = j^{th} input (*scalar*)
- $h_j(\mathbf{x})$ = j^{th} feature (*scalar*)
- $\mathbf{x}_i$ = input of i^{th} data point (*vector*)
- $\mathbf{x}_i[j]$ = j^{th} input of i^{th} data point (*scalar*)

Generic Linear Regression Model

Model:

$$y_i = w_0\, h_0(\mathbf{x}_i) + w_1\, h_1(\mathbf{x}_i) + \ldots + w_D\, h_D(\mathbf{x}_i) + \varepsilon_i$$

$$= \sum_{j=0}^{D} w_j\, h_j(\mathbf{x}_i) + \varepsilon_i$$

feature 1 = $h_0(\mathbf{x})$ … e.g., 1
feature 2 = $h_1(\mathbf{x})$ … e.g., $\mathbf{x}$[1] = sq. ft.
feature 3 = $h_2(\mathbf{x})$ … e.g., $\mathbf{x}$[2] = #bath
or, log($\mathbf{x}$[7]) $\mathbf{x}$[2] = log(#bed) x #bath
…
feature D+1 = $h_D(\mathbf{x})$ … some other function of $\mathbf{x}$[1],…, $\mathbf{x}$[d]

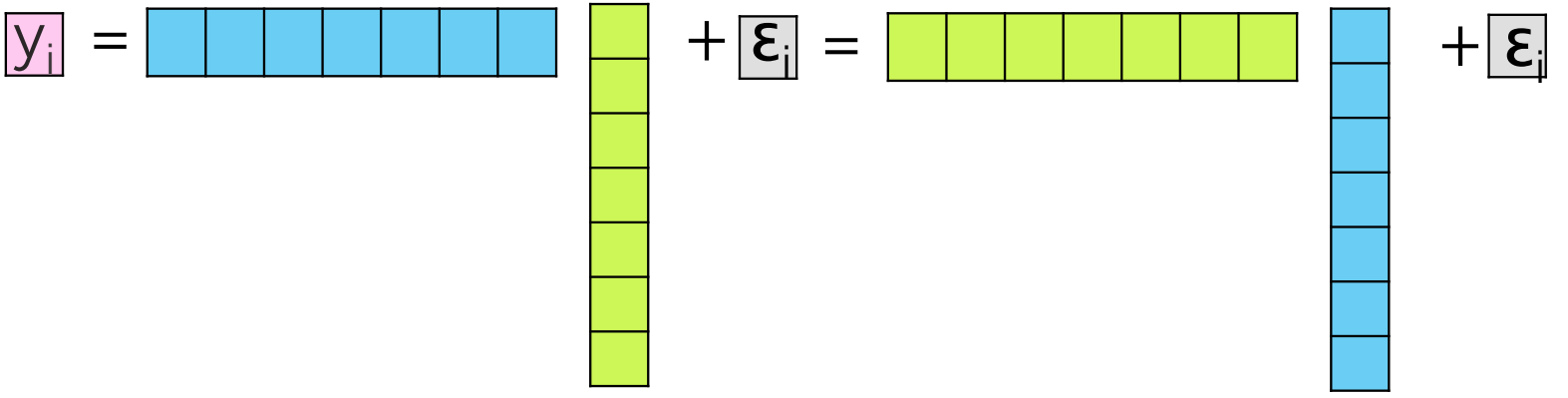
Fitting the Linear Regression Model

Step 1:
Rewrite the regression model

Rewrite in Matrix Notation

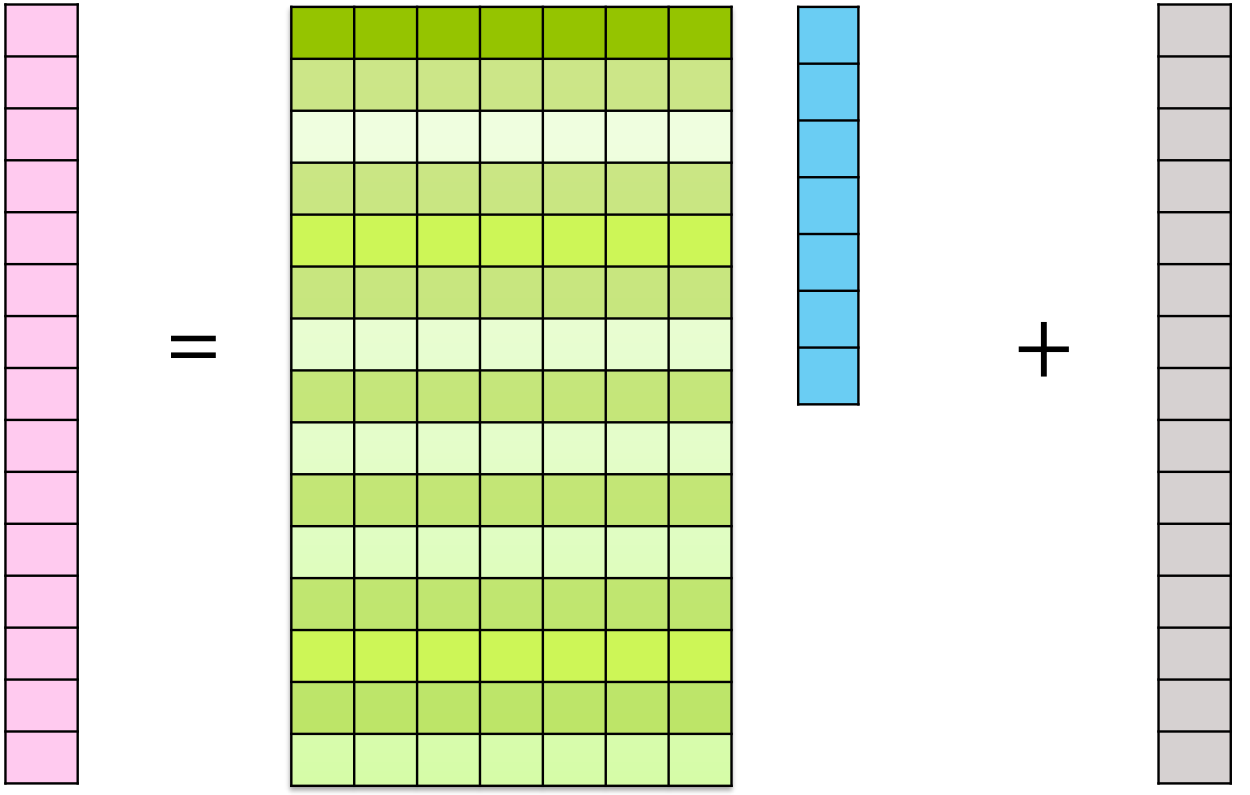
For observation i

$$y_i = \sum_{j=0}^{D} w_j \, h_j(\mathbf{x}_i) + \varepsilon_i$$



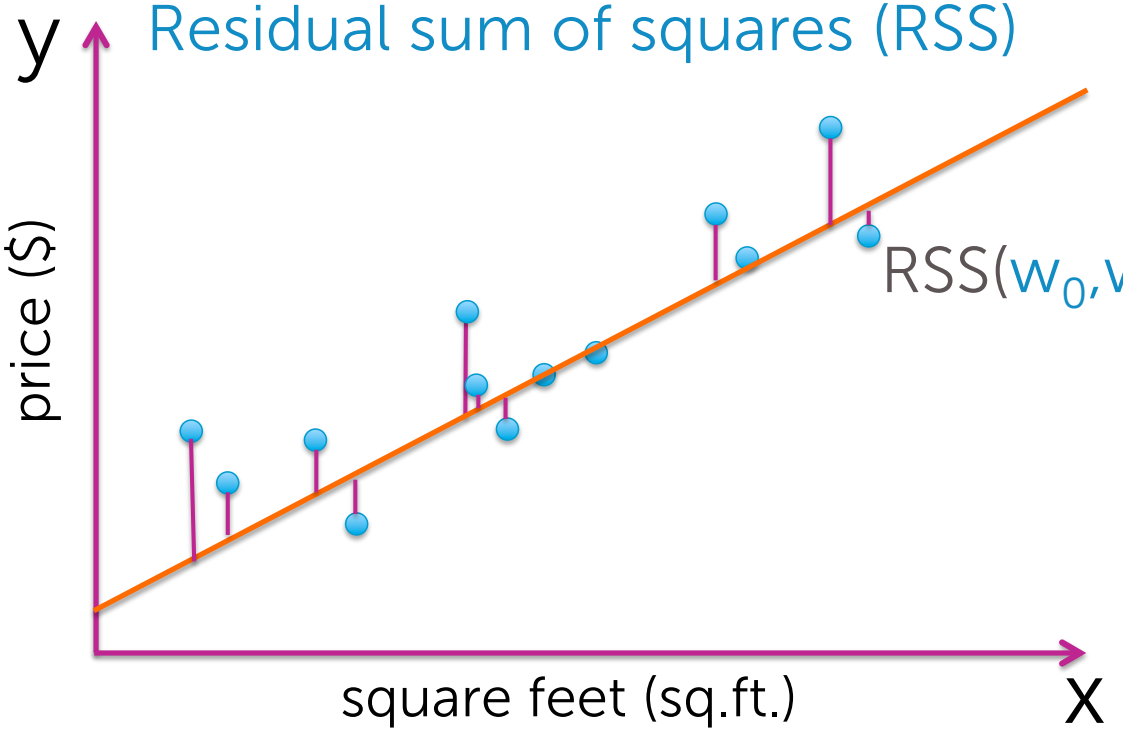
Rewrite in Matrix Notation

For all observations together



Step 2:
Compute the cost

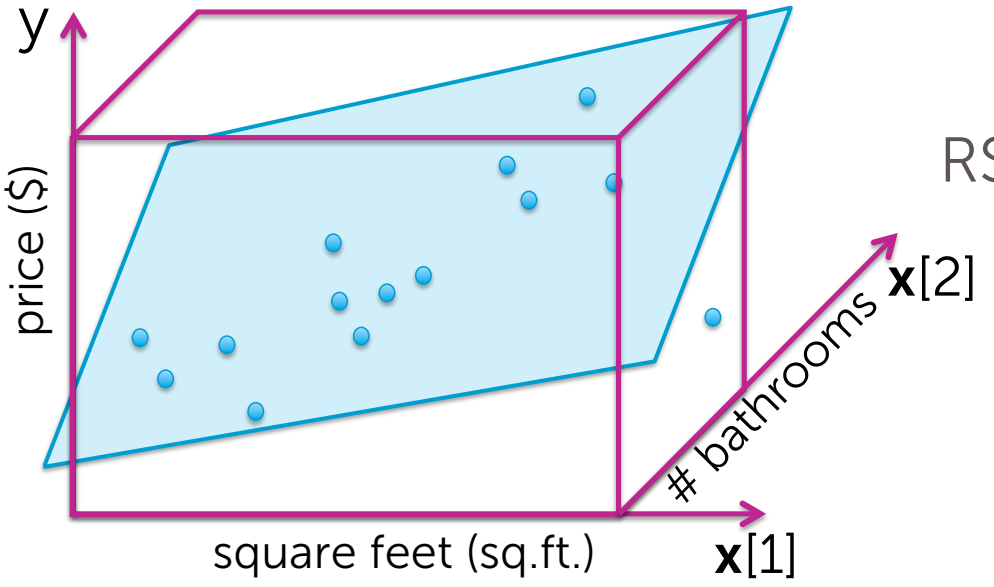
"Cost" of using a given line



Residual sum of squares (RSS)

$$\text{RSS}(w_0, w_1) = \sum_{i=1}^{N} (y_i - [w_0 + w_1 x_i])^2$$

RSS for multiple regression



$$\text{RSS}(\mathbf{w}) = \sum_{i=1}^{N} (y_i - \qquad)^2$$

$$= (\mathbf{y} - \mathbf{H}\mathbf{w})^T(\mathbf{y} - \mathbf{H}\mathbf{w})$$

Step 3:
Take the gradient

Gradient of RSS

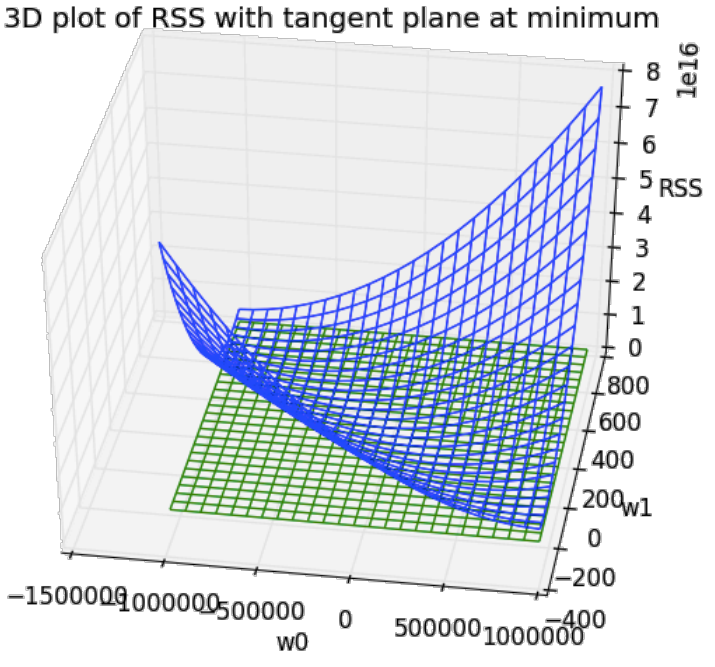
$$\nabla \mathrm{RSS}(\mathbf{w}) = \nabla[(\mathbf{y}-\mathbf{H}\mathbf{w})^T(\mathbf{y}-\mathbf{H}\mathbf{w})]$$

$$= -2\mathbf{H}^T(\mathbf{y}-\mathbf{H}\mathbf{w})$$

Why? By analogy to 1D case:

Step 4, Approach 1:
Set the gradient = 0

Closed-form solution

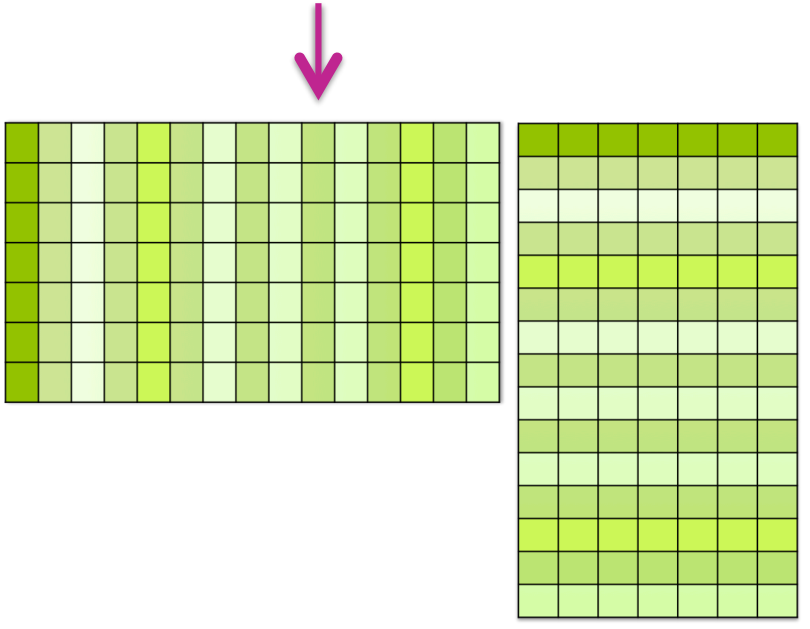


$$\nabla \mathrm{RSS}(\mathbf{w}) = -2\mathbf{H}^{\mathsf{T}}(\mathbf{y}-\mathbf{H}\mathbf{w}) = 0$$

Solve for **w**:

Closed-form solution

$$\hat{\mathbf{w}} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}$$



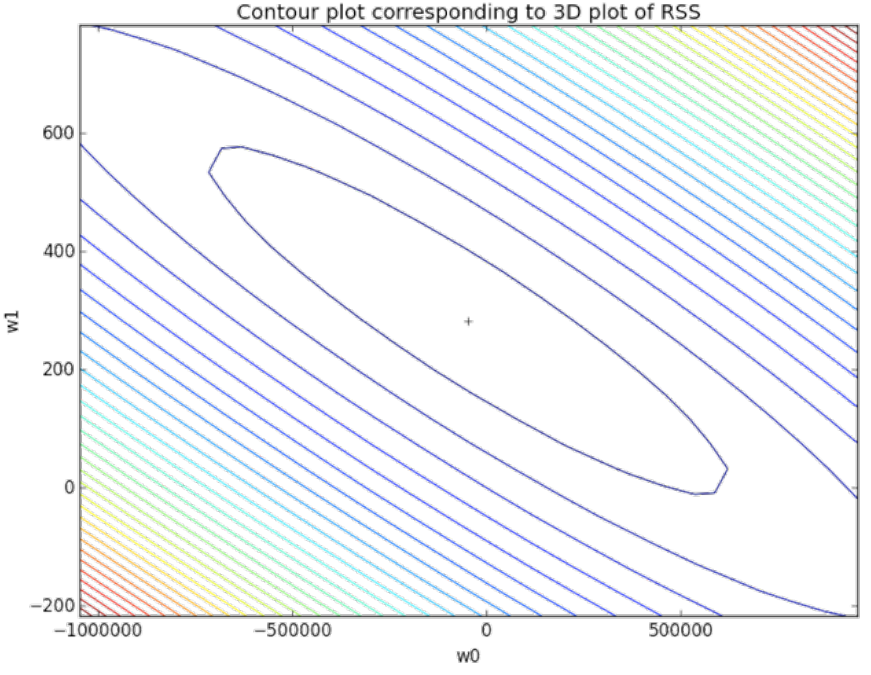
Invertible if:

Complexity of inverse:

$O(D^3)$

Step 4, Approach 2:
Gradient descent

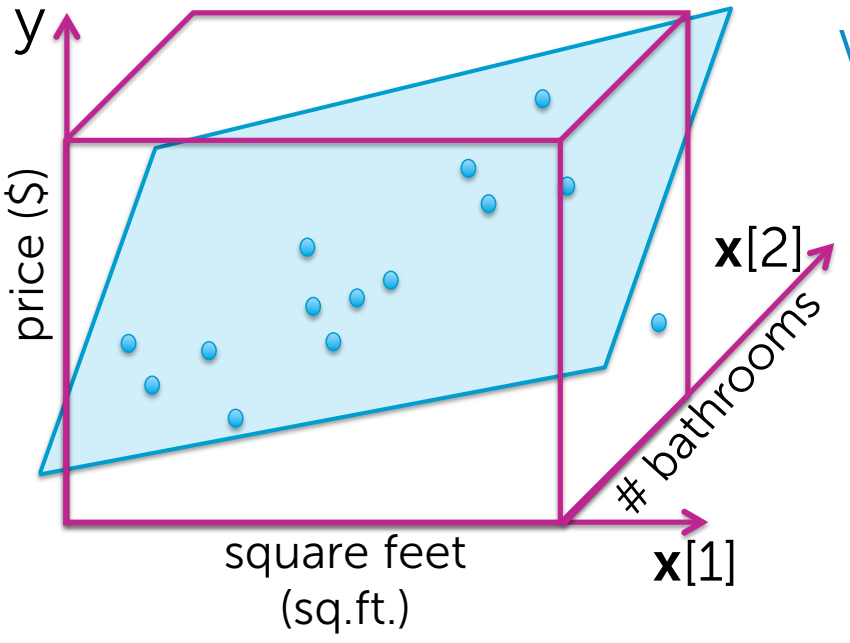
Gradient descent



while not converged

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \underbrace{\nabla \mathrm{RSS}(\mathbf{w}^{(t)})}_{-2\mathbf{H}^{\mathsf{T}}(\mathbf{y}-\mathbf{H}\mathbf{w})}$$

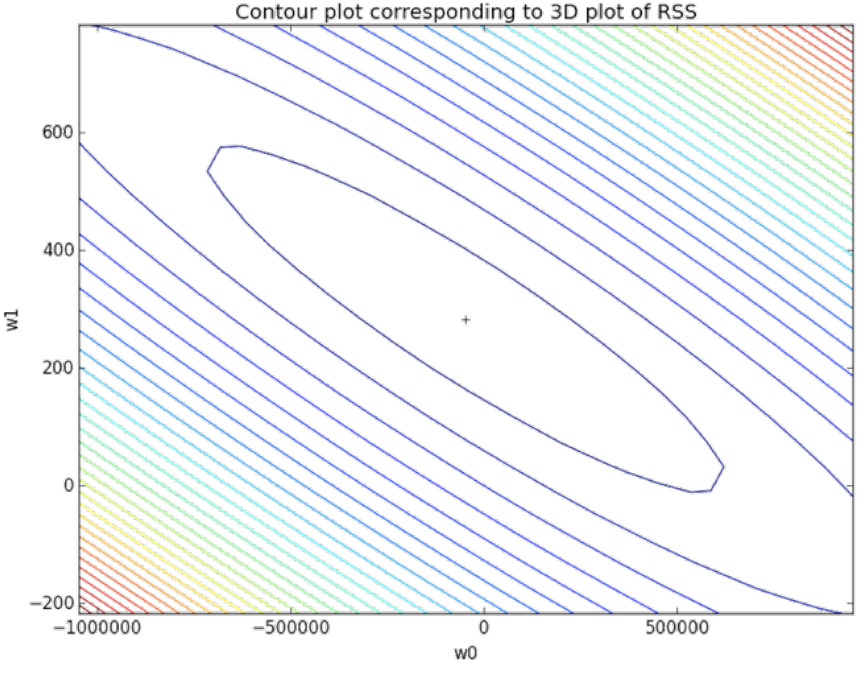
Interpreting elementwise



Update to j^{th} feature weight:

$$w_j^{(t+1)} \leftarrow w_j^{(t)} + 2\eta \sum_{i=1}^{N} h_j(\mathbf{x}_i)(y_i - \hat{y}_i(\mathbf{w}^{(t)}))$$

Summary of gradient descent for multiple regression



init $\mathbf{w}^{(1)}=0$ (or randomly, or smartly), t=1

while $||\nabla \text{RSS}(\mathbf{w}^{(t)})|| > \varepsilon$

  for j=0,...,D

    $\text{partial}[j] = -2\sum_{i=1}^{N} h_j(\mathbf{x}_i)(y_i - \hat{y}_i(\mathbf{w}^{(t)}))$

    $\mathbf{w}_j^{(t+1)} \leftarrow \mathbf{w}_j^{(t)} - \eta\ \text{partial}[j]$

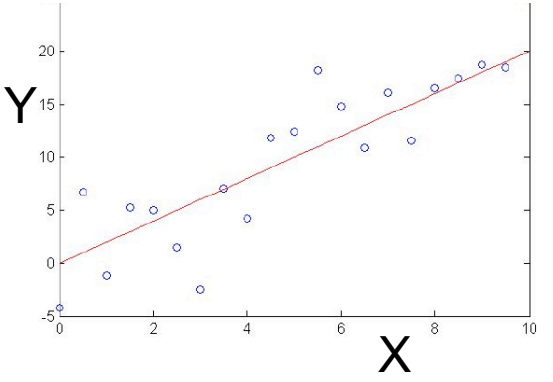
  $t \leftarrow t + 1$

Why min RSS?

Learn **W** using MLE

Gaussian method:

1. Choose parameterized form for P(Y|X; θ)



Assume Y is some deterministic f(X), plus random noise

$$y = f(x) + \epsilon \qquad \text{where} \quad \epsilon \sim N(0, \sigma)$$

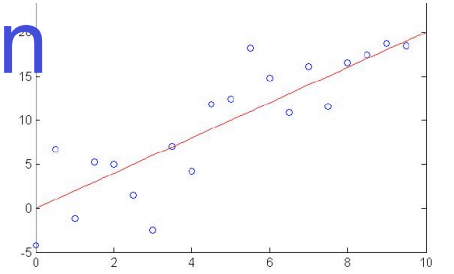
Therefore Y is a random variable that follows the distribution

$$p(y|x) = N(f(x), \sigma)$$

and the expected value of y for any given x is f(x)

Learn **W** using MLE

Consider Linear Regression



$$p(y|x) = N(f(x), \sigma)$$

E.g., assume f(x) is linear function of x

$$p(y|x) = N(w_0 + w_1 x, \sigma)$$

$$E[y|x] = w_0 + w_1 x$$

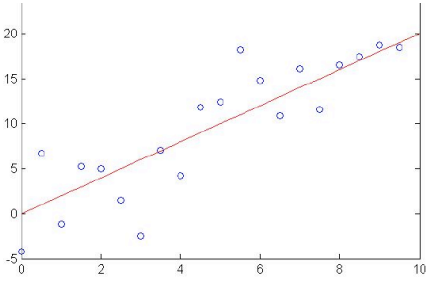
Notation: to make our parameters explicit, let's write

$$W = < w_0, w_1 >$$

$$p(y|x; W) = N(w_0 + w_1 x, \sigma)$$

Learn **W** using MLE

Training Linear Regression



$$p(y|x;W) = N(w_0 + w_1 x, \sigma)$$

How can we learn W from the training data?

Learn Maximum Conditional Likelihood Estimate!

$$W_{MCLE} = \arg\max_W \prod_l p(y^l|x^l, W)$$

$$W_{MCLE} = \arg\max_W \sum_l \ln p(y^l|x^l, W)$$

where

$$p(y|x;W) = \frac{1}{\sqrt{2\pi\sigma^2}} \; e^{-\frac{1}{2}(\frac{y-f(x;W)}{\sigma})^2}$$

Maximizing log-likelihood

Maximize wrt w:

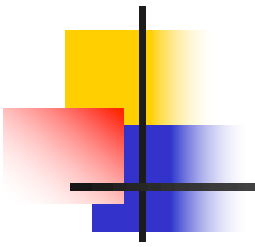
$$\ln P(\mathcal{D} \mid \mathbf{w}, \sigma) = \ln \left(\frac{1}{\sigma\sqrt{2\pi}} \right)^N \prod_{j=1}^{N} e^{\frac{-[t_j - \sum_i w_i h_i(\mathbf{x}_j)]^2}{2\sigma^2}}$$

$$\arg\max_w \ln \left(\frac{1}{\sigma\sqrt{2\pi}} \right)^N + \sum_{j=1}^{N} \frac{-[t_j - \sum_i w_i h_i(x_j)]^2}{2\sigma^2}$$

$$= \arg\max_w \sum_{j=1}^{N} \frac{-[t_j - \sum_i w_i h_i(x_j)]^2}{2\sigma^2}$$

$$= \arg\min_w \sum_{j=1}^{N} [t_j - \sum_i w_i h_i(x_j)]^2$$

For σ = 1 (or some constant) for each input, MLE for Gaussians is equivalent to the least-squares objective for linear regression



Regression

Wish to learn f:X→Y, where Y is real, given $\{<x^1,y^1>\ldots<x^n,y^n>\}$

Approach:

1. choose some parameterized form for $P(Y|X; \theta)$
   (θ is the vector of parameters)

2. derive learning algorithm as MLE or MAP estimate for θ

Regularization in Linear Regression

- **Potential problem:** *Overfitting*
  - especially when data is very high dimensional and training data is sparse.

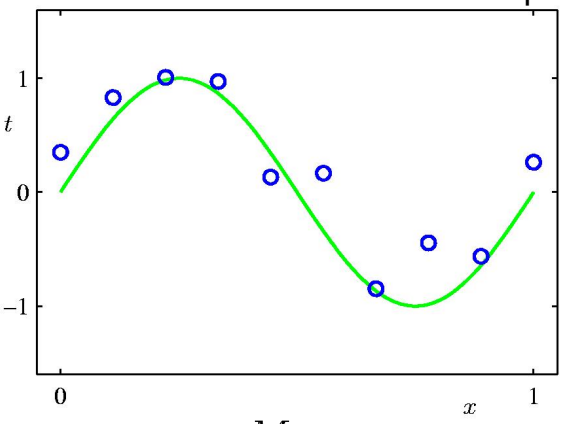
- One sign of overfitting: large parameter values!

- **Solution:** *Regularization*
  - penalized log likelihood function: penalizes large values of W

A simple example: Fitting a polynomial

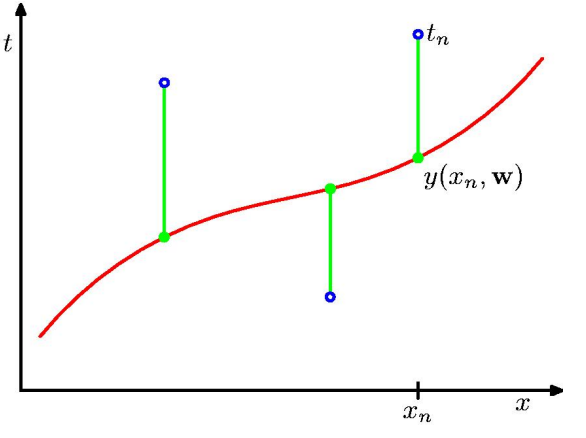
- The green curve is the true function (which is not a polynomial)
- The data points are uniform in x but have noise in y.

from Bishop

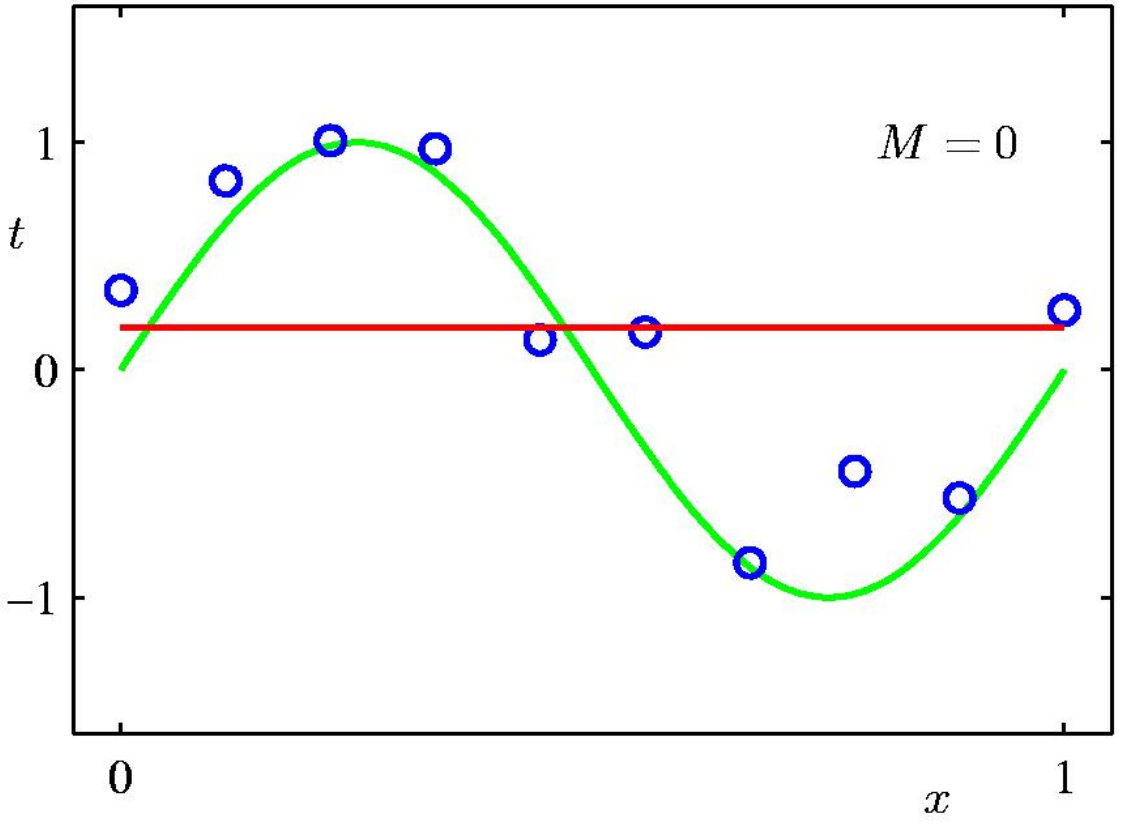


$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

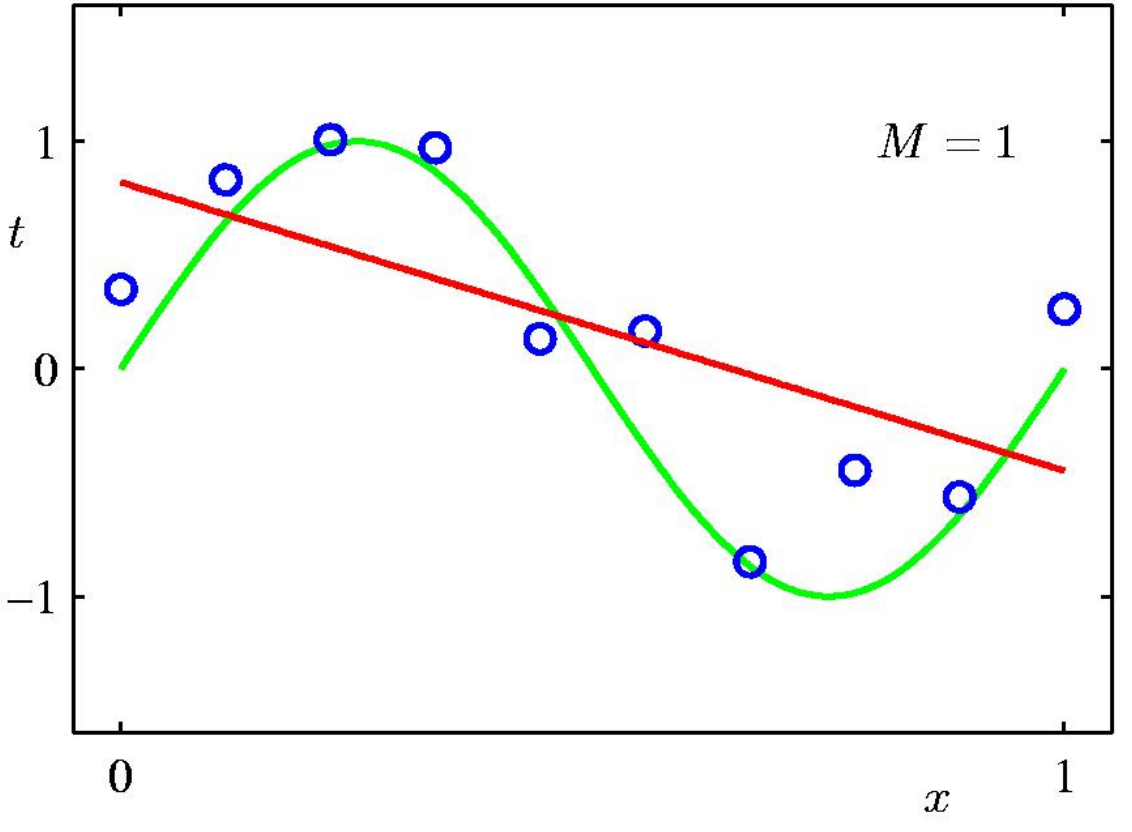
- We will use a loss function that measures the squared error in the prediction of y(x) from x. The loss for the red polynomial is the sum of the squared vertical errors.



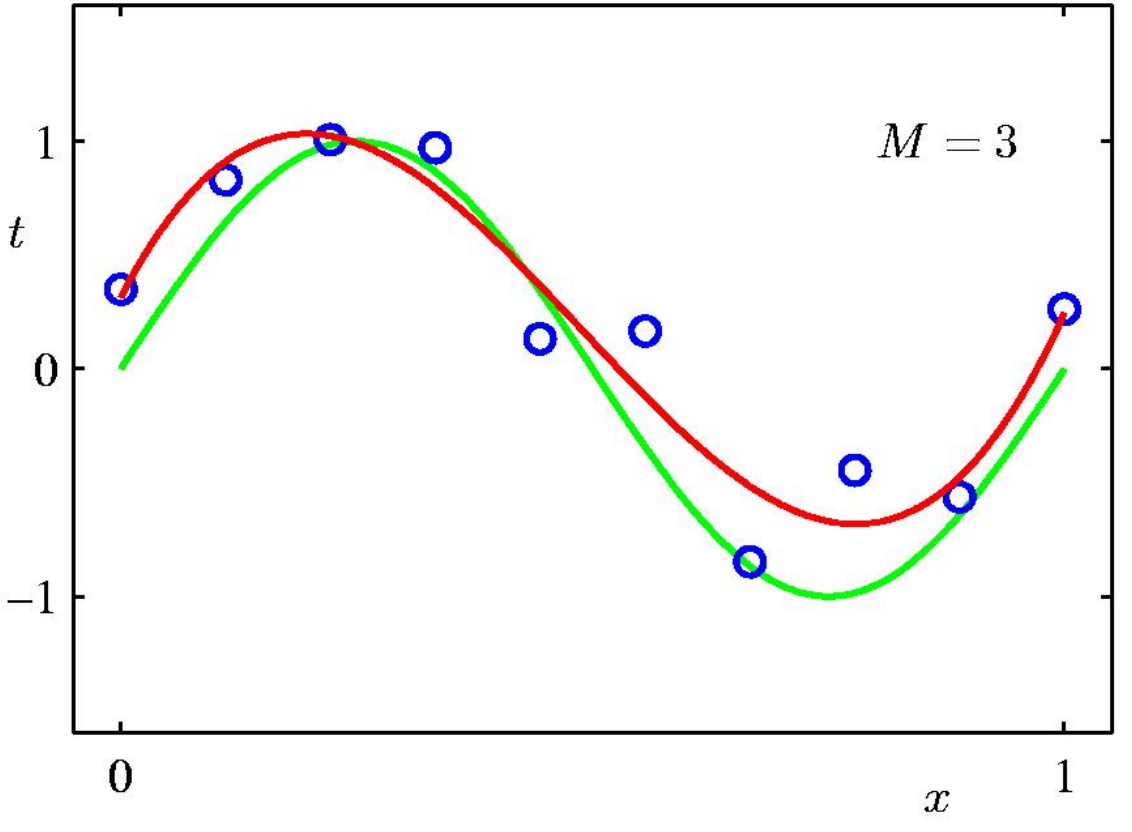
0^{th} Order Polynomial



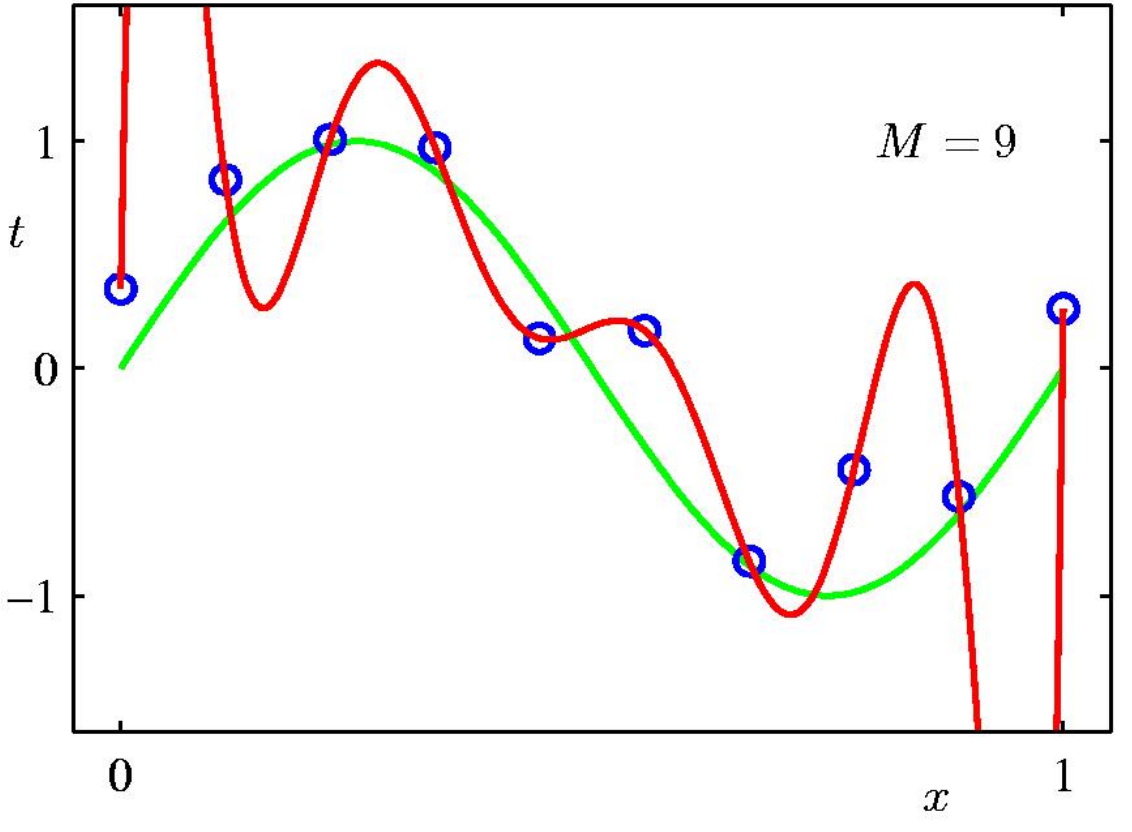
1st Order Polynomial



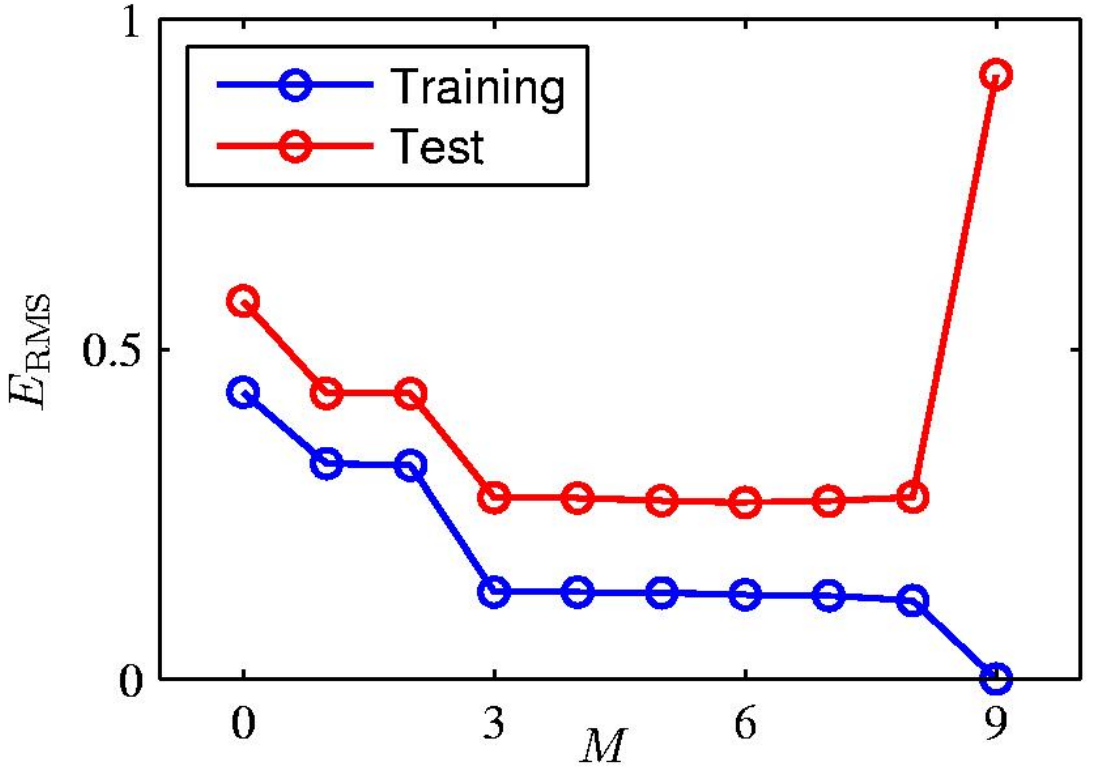
3rd Order Polynomial



9th Order Polynomial



Over-fitting



Root-Mean-Square (RMS) Error: $E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^{\star})/N}$

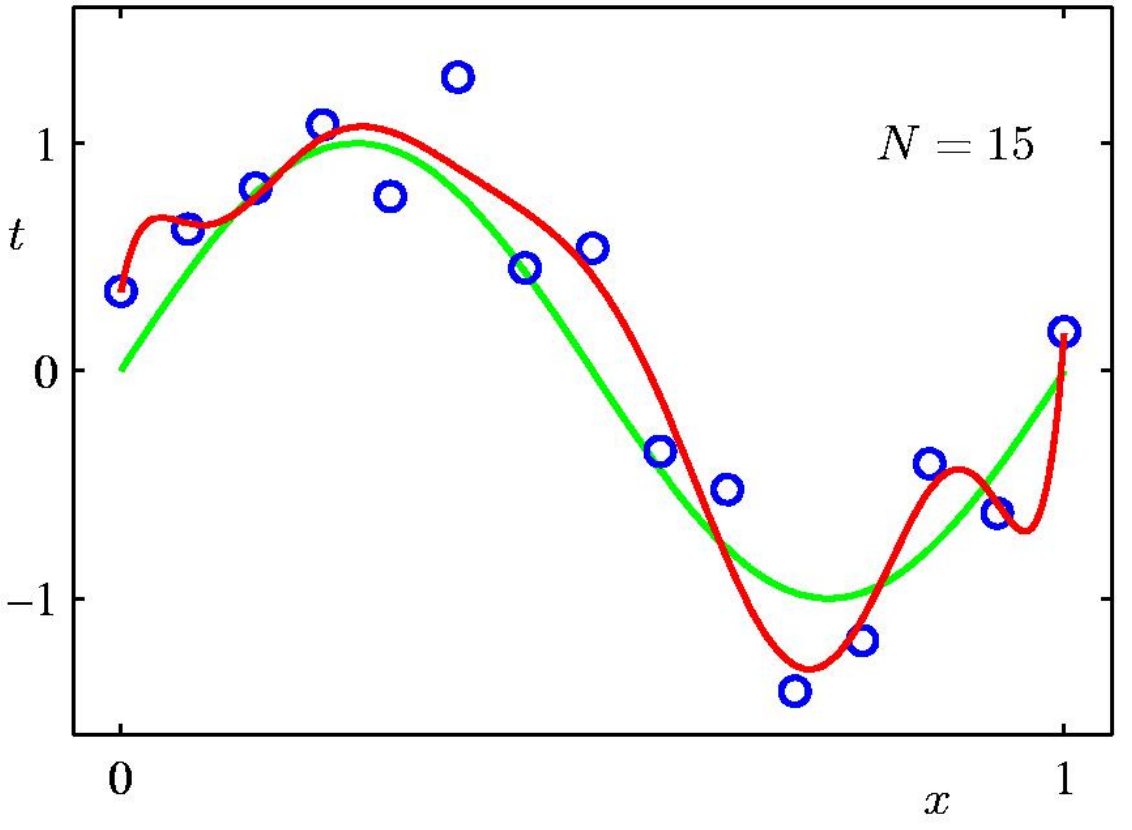
Polynomial Coefficients

	$M=0$	$M=1$	$M=3$	$M=9$
$w_0^\star$	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:

$N = 15$

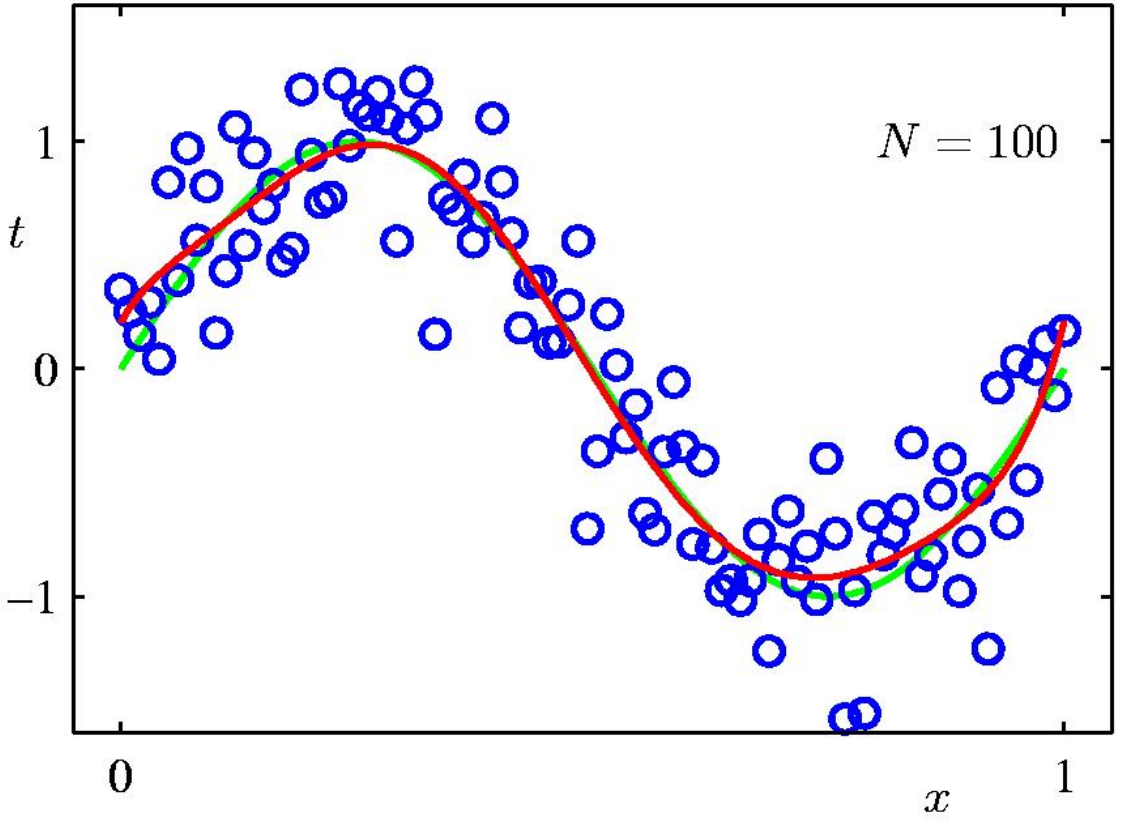
9th Order Polynomial



Data Set Size:

9th Order Polynomial

$$N = 100$$



Ridge Regularization

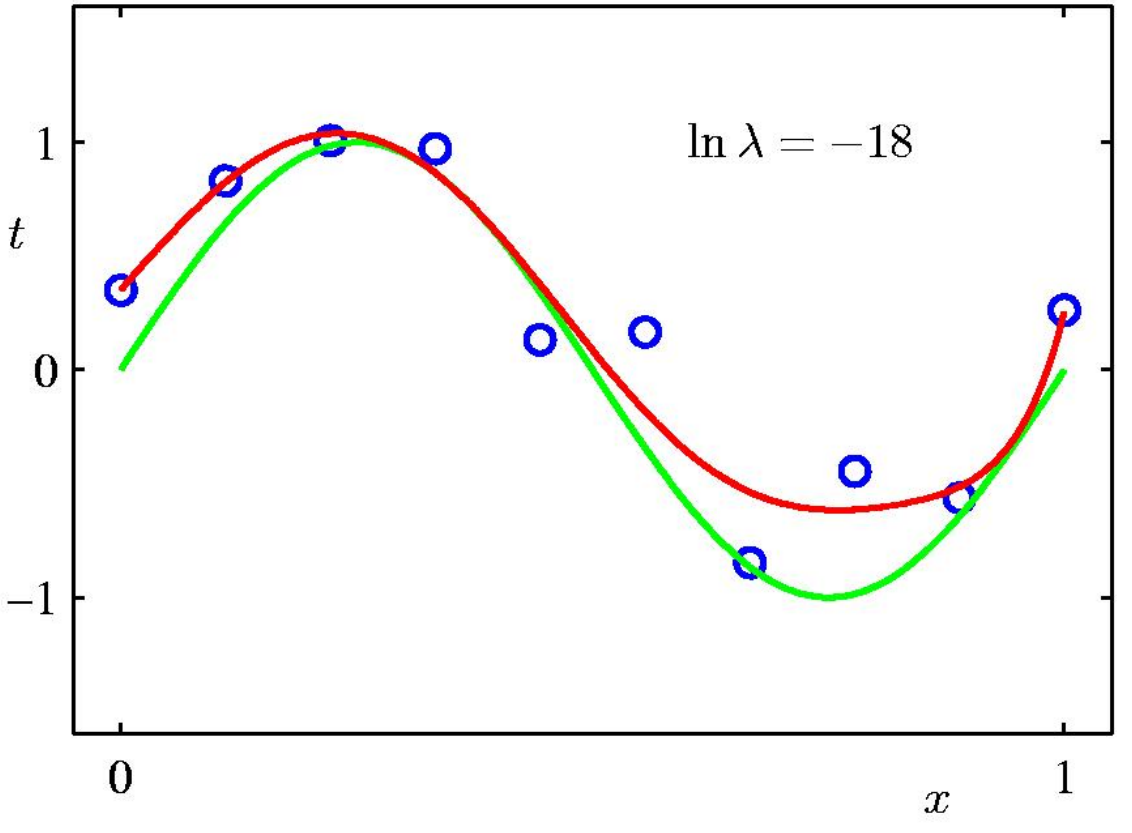
- Penalize large coefficient values

$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

  - Prefer low error but also add a square penalize for large weights
  - λ is hyperparameter that balances tradeoff

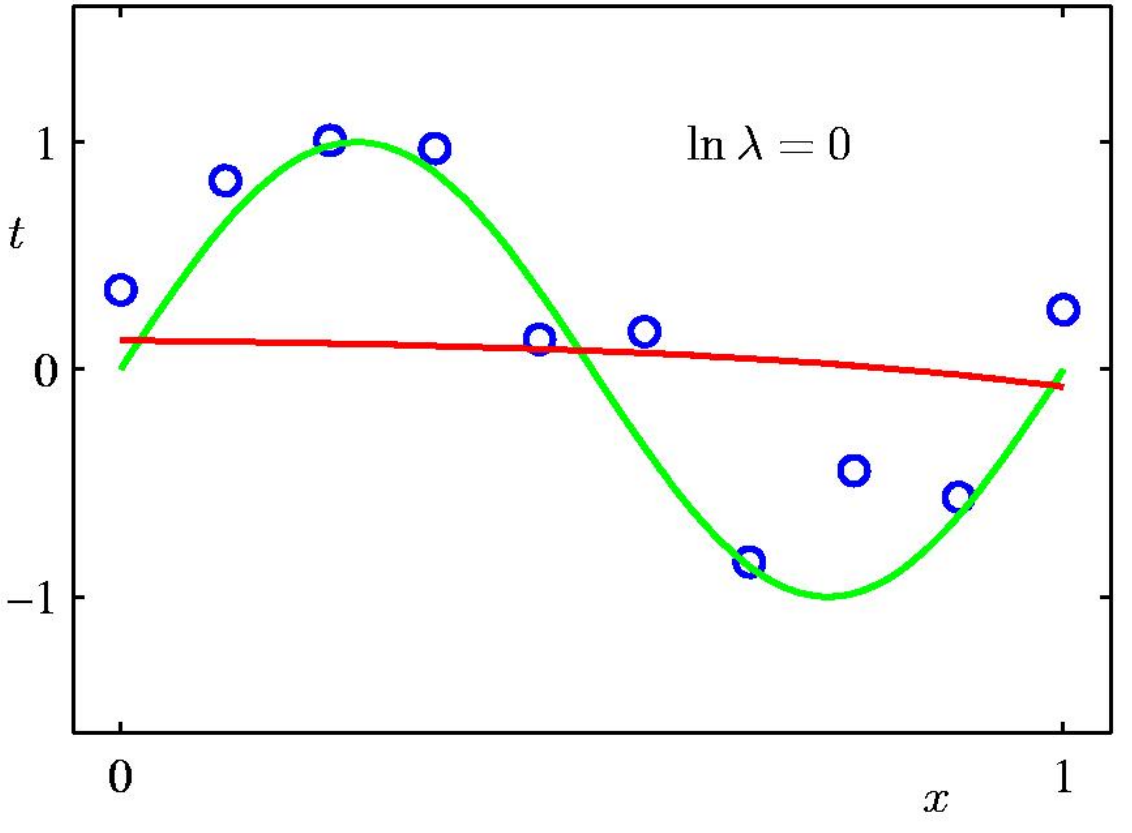
Regularization:

$$\ln \lambda = -18$$

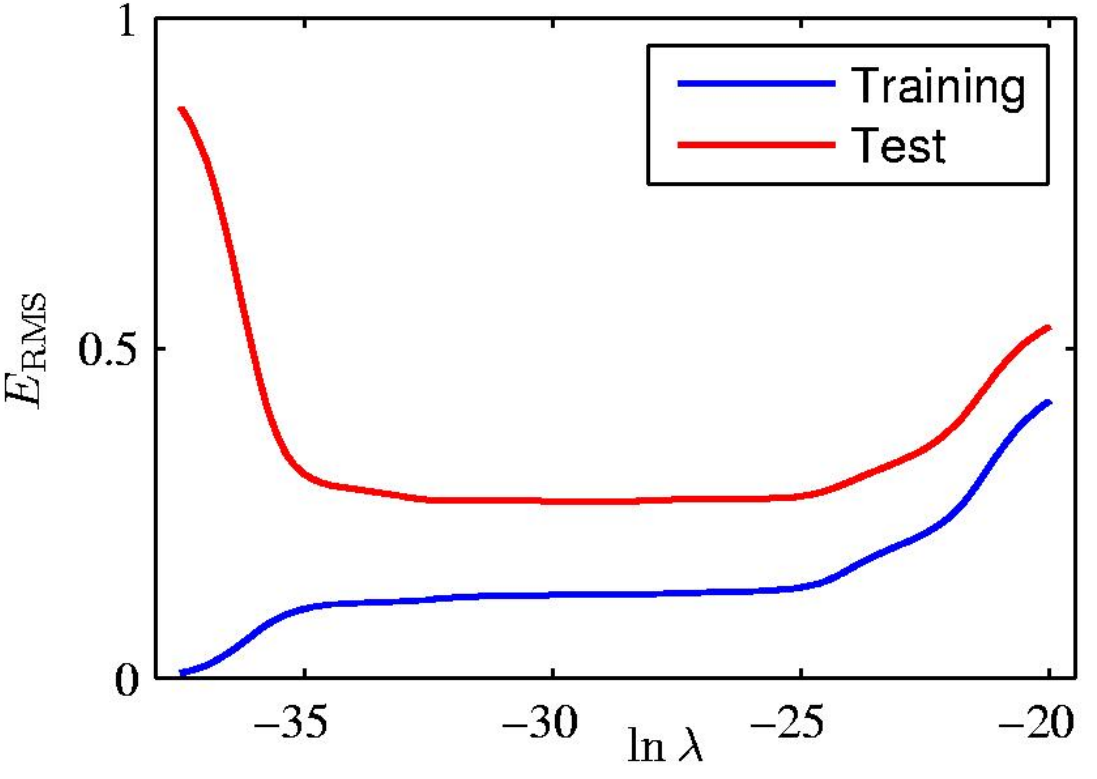


Regularization:

$$\ln \lambda = 0$$



Regularization: E_{RMS} vs. $\ln \lambda$

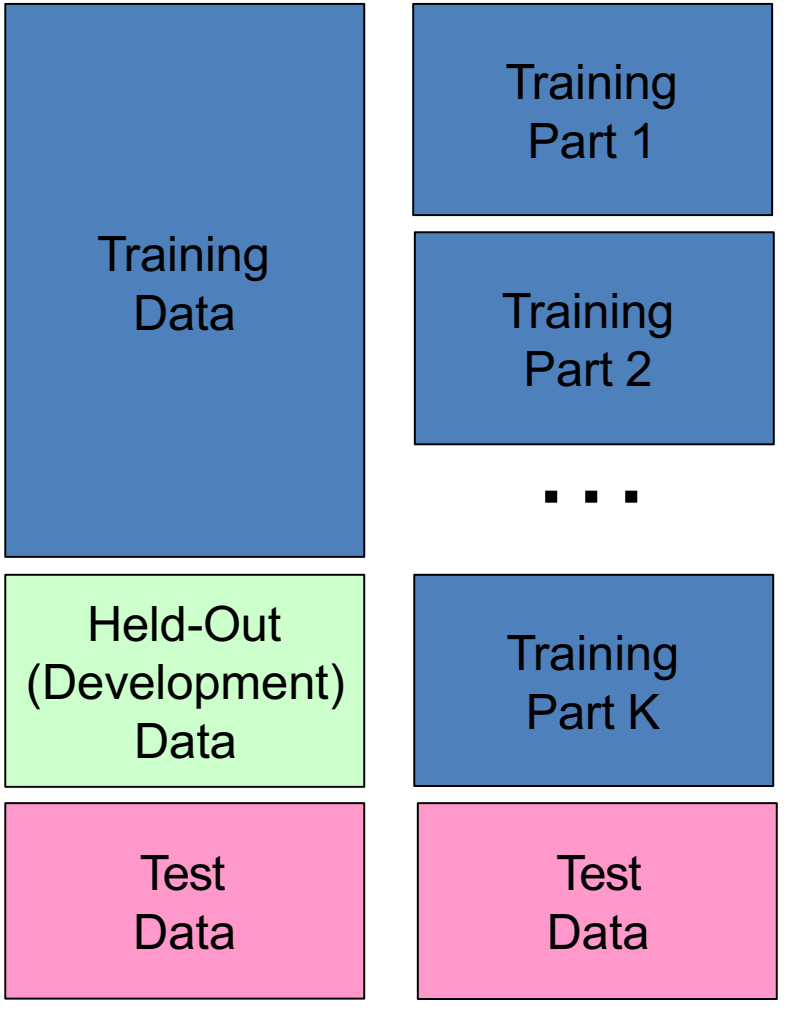


Polynomial Coefficients

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	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

How to pick lambda?

- Experimentation cycle
  - Select a hypothesis *f* to best match training set
  - Tune hyperparameters on held-out set
    - Try many different values of lambda, pick best one
- Or, can do k-fold cross validation
  - No held-out set
  - Divide training set into k subsets
  - Repeatedly train on k-1 and test on remaining one
  - Average the results



Linear Regression: MAP Solution

Assume a Gaussian prior distribution over the weight vector w=[w_1,…,w_D]

$$p(\boldsymbol{w}) = \mathcal{N}(0, \lambda^{-1}\mathbf{I}_D) = \left(\frac{\lambda}{2\pi}\right)^{D/2} \exp(-\frac{\lambda}{2}\boldsymbol{w}^\top \boldsymbol{w}) = \left(\frac{\lambda}{2\pi}\right)^{D/2} \exp(-\frac{\lambda}{2}||\boldsymbol{w}||^2)$$

Log posterior probability:

$$\log P(\mathbf{w} \mid \mathcal{D}) = \log \frac{P(\mathbf{w})P(\mathcal{D} \mid \mathbf{w})}{P(\mathcal{D})} = \log P(\mathbf{w}) + \log P(\mathcal{D} \mid \mathbf{w}) - \log P(\mathcal{D})$$

Maximum-a-Posteriori Solution: $\hat{\mathbf{w}}_{MAP} = \arg\max_{\mathbf{w}} \log P(\mathbf{w} \mid \mathcal{D})$

$$= \arg\max_{\mathbf{w}} \{\log P(\mathbf{w}) + \log P(\mathcal{D} \mid \mathbf{w}) - \log P(\mathcal{D})\}$$

$$= \arg\max_{\mathbf{w}} \{\log P(\mathbf{w}) + \log P(\mathcal{D} \mid \mathbf{w})\}$$

$$= \arg\max_{\mathbf{w}} \left\{ -\frac{D}{2}\log(2\pi) - \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w} + \sum_{n=1}^{N}\left\{ -\frac{1}{2}\log(2\pi\sigma^2) - \frac{(y_n - \mathbf{w}^\top\mathbf{x}_n)^2}{2\sigma^2} \right\} \right\}$$

$$= \arg\min_{\boldsymbol{w}} \underbrace{\frac{\beta}{2}\sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top\boldsymbol{x}_n)^2}_{\text{fit to the training data}} + \underbrace{\frac{\lambda}{2}\boldsymbol{w}^\top\boldsymbol{w}}_{\text{keep } \boldsymbol{w} \text{ "simple"}}$$

For $\sigma = 1$ (or some constant) for each input, it's equivalent to the regularized least-squares objective

Linear Regression: MLE vs. MAP

MLE solution:

$$\hat{\mathbf{w}}_{MLE} = \arg\min_{\mathbf{w}} \frac{1}{2\sigma^2} \sum_{n=1}^{N} (y_n - \mathbf{w}^\top \mathbf{x}_n)^2$$

MAP solution:

$$\hat{\mathbf{w}}_{MAP} = \arg\min_{\mathbf{w}} \frac{1}{2\sigma^2} \sum_{n=1}^{N} (y_n - \mathbf{w}^\top \mathbf{x}_n)^2 + \frac{\lambda}{2} \mathbf{w}^\top \mathbf{w}$$

Take-home messages:

- MLE estimation of a parameter leads to unregularized solutions
- MAP estimation of a parameter leads to regularized solutions
- The prior distribution acts as a regularizer in MAP estimation

Note: For MAP, different prior distributions lead to different regularizers

- Gaussian prior on $\mathbf{w}$ regularizes the ℓ_2 norm of $\mathbf{w}$
- Laplace prior $\exp(-C||\mathbf{w}||_1)$ on $\mathbf{w}$ regularizes the ℓ_1 norm of $\mathbf{w}$

Regression- What you should know

Under general assumption $p(y|x;W) = N(f(x;W), \sigma)$

1. MLE corresponds to minimizing sum of squared prediction errors
2. MAP estimate minimizes SSE plus sum of squared weights
3. Again, learning is an optimization problem once we choose our objective function
   - maximize data likelihood
   - maximize posterior prob of W

4. Again, we can use gradient descent as a general learning algorithm
   - as long as our objective function is differentiable wrt W
   - though we might learn local optima ins