Machine Learning
DSECL ZG565

Dr. Monali Mavani

BITS Pilani
Pilani Campus

Contents

- Learning Model Parameters linear regression– Closed Form Solution (using vectorization)
- Linear basis function models (3.1 Bishop)
- Over fitting, Under fitting, Regularization (1.1 Bishop, Andrew Ng notes)
- Bias-Variance Decomposition (Bishop)

Learning Model Parameters – Closed Form Solution (using vectorization)

Vectorization

- Benefits of vectorization
  - More compact equations
  - Faster code (using optimized matrix libraries)
- Consider our model:

$$h(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j x_j$$

- Let

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \qquad \boldsymbol{x}^\intercal = \begin{bmatrix} 1 & x_1 & \dots & x_d \end{bmatrix}$$

- Can write the model in vectorized form as $h(\boldsymbol{x}) = \boldsymbol{\theta}^\intercal \boldsymbol{x}$

Vectorization

- Consider our model for n instances:

$$h\left(\boldsymbol{x}^{(i)}\right) = \sum_{j=0}^{d} \theta_j x_j^{(i)}$$

- Let

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \qquad \boldsymbol{X} = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_d^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(i)} & \dots & x_d^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \dots & x_d^{(n)} \end{bmatrix}$$

$\mathbb{R}^{(d+1)\times 1}$ $\qquad$ $\mathbb{R}^{n\times(d+1)}$

- Can write the model in vectorized form as $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{X}\boldsymbol{\theta}$

Vectorization

- For the linear regression cost function:

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

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

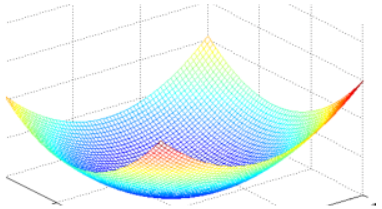
$$= \frac{1}{2n}\underbrace{(\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y})^{\intercal}}_{\mathbb{R}^{1\times n}}\,(\underbrace{\boldsymbol{X}}_{\mathbb{R}^{n\times(d+1)}}\underbrace{\boldsymbol{\theta}}_{\mathbb{R}^{(d+1)\times 1}} - \underbrace{\boldsymbol{y}}_{\mathbb{R}^{n\times 1}})$$

Let:

$$\boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

Closed Form Solution

- Instead of using GD, solve for optimal $\boldsymbol{\theta}$ analytically
  - Notice that the solution is when $\frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}) = 0$



- Derivation:

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{2n}(\boldsymbol{X\theta} - \boldsymbol{y})^\intercal(\boldsymbol{X\theta} - \boldsymbol{y})$$

1 x 1

$$\propto \boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{\theta} - \boxed{\boldsymbol{y}^\intercal \boldsymbol{X}\boldsymbol{\theta}} - \boxed{\boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{y}} + \boldsymbol{y}^\intercal \boldsymbol{y}$$

$$\propto \boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{\theta} - 2\boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{y} + \boldsymbol{y}^\intercal \boldsymbol{y}$$

Take derivative and set equal to 0, then solve for $\boldsymbol{\theta}$:

$$\frac{\partial}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{\theta} - 2\boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{y} + \cancel{\boldsymbol{y}^\intercal \boldsymbol{y}}) = 0$$

$$(\boldsymbol{X}^\intercal \boldsymbol{X})\boldsymbol{\theta} - \boldsymbol{X}^\intercal \boldsymbol{y} = 0$$

$$(\boldsymbol{X}^\intercal \boldsymbol{X})\boldsymbol{\theta} = \boldsymbol{X}^\intercal \boldsymbol{y}$$

Closed Form Solution: $\boldsymbol{\theta} = (\boldsymbol{X}^\intercal \boldsymbol{X})^{-1}\boldsymbol{X}^\intercal \boldsymbol{y}$

Closed Form Solution

- Can obtain $\boldsymbol{\theta}$ by simply plugging $\boldsymbol{X}$ and $\boldsymbol{y}$ into

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

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_d^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(i)} & \dots & x_d^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \dots & x_d^{(n)} \end{bmatrix} \qquad \boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

- If $\boldsymbol{X}^\intercal \boldsymbol{X}$ is not invertible (i.e., singular), may need to:
  - Use pseudo-inverse instead of the inverse
    - In python, `numpy.linalg.pinv(a)`
  - Remove redundant (not linearly independent) features
  - Remove extra features to ensure that $d \leq n$

Gradient Descent vs Closed Form

Gradient Descent	Closed Form Solution
• Requires multiple iterations • Need to choose α • Works well when n is large • Can support incremental learning	• Non-iterative • No need for α • Slow if n is large – Computing $(\boldsymbol{X}^{\top}\boldsymbol{X})^{-1}$ is roughly $O(n^3)$

Linear Basis Function Models

Extending Linear Regression to More Complex Models

- The inputs **X** for linear regression can be:
  - Original quantitative inputs
  - Transformation of quantitative inputs
    - e.g. log, exp, square root, square, etc.
  - Polynomial transformation
    - example: $y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot x^2 + \beta_3 \cdot x^3$
  - Basis expansions
  - Dummy coding of categorical inputs
  - Interactions between variables
    - example: $x_3 = x_1 \cdot x_2$

This allows use of linear regression techniques to fit non-linear datasets.

Linear Basis Function Models

- Basic Linear Model: $$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j x_j$$
- Generalized Linear Model: $$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j \phi_j(\boldsymbol{x})$$
- Once we have replaced the data by the outputs of the basis functions, fitting the generalized model is exactly the same problem as fitting the basic model
  - Unless we use the kernel trick – more on that when we cover support vector machines
  - Therefore, there is no point in cluttering the math with basis functions

Based on slide by Geoff Hinton

Linear Basis Function Models

- Generally,

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j \underbrace{\phi_j(\boldsymbol{x})}_{\text{basis function}}$$

- Typically, $\phi_0(\boldsymbol{x}) = 1$ so that θ_0 acts as a bias
- In the simplest case, we use linear basis functions :

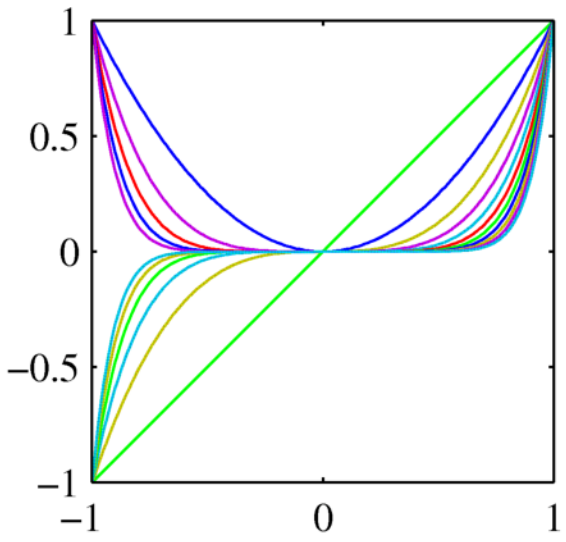
$$\phi_j(\boldsymbol{x}) = x_j$$

Linear Basis Function Models

- Polynomial basis functions:

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

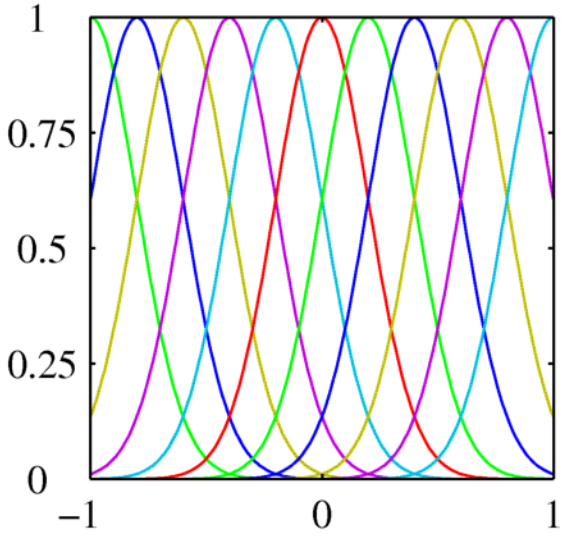
  – These are global; a small change in x affects all basis functions



- Gaussian basis functions:

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

  – These are local; a small change in x only affect nearby basis functions. μ_j and s control location and scale (width).



Based on slide by Christopher Bishop (PRML)

Linear Basis Function Models

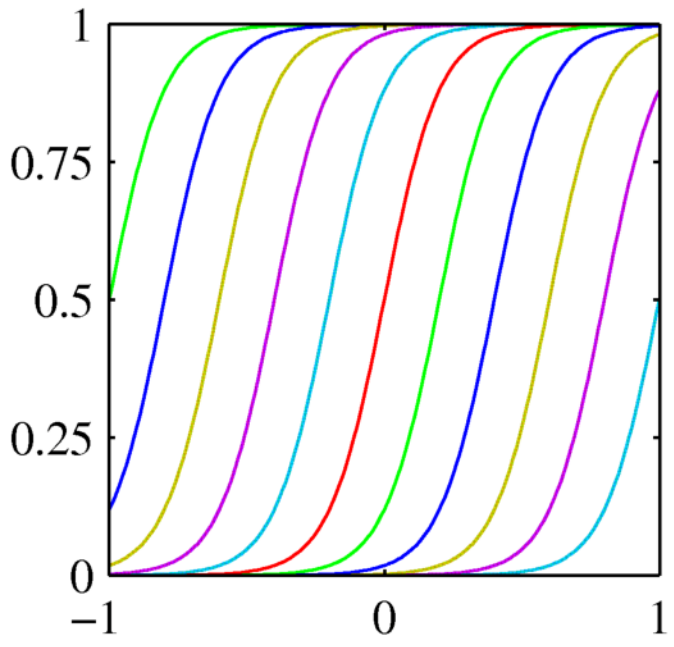
- Sigmoidal basis functions:

$$\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right)$$

where

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

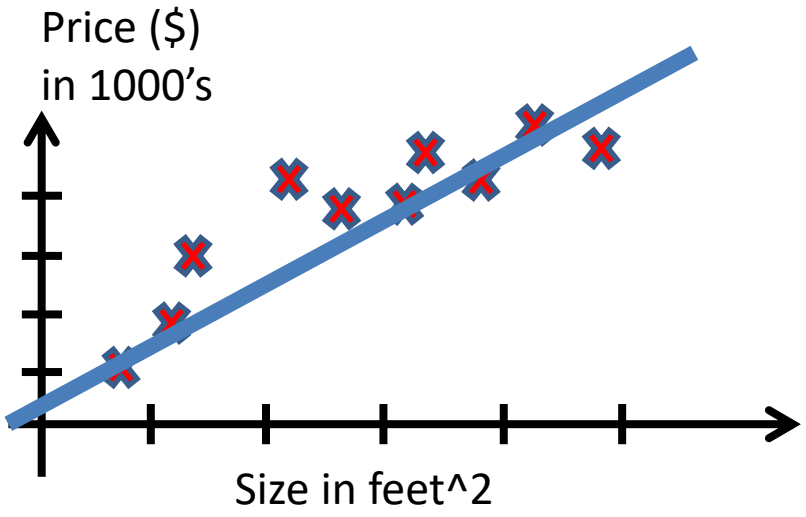
  – These are also local; a small change in x only affects nearby basis functions. μ_j and s control location and scale (slope).



By using nonlinear basis functions, we allow the function $y(\mathbf{x},\mathbf{w})$ to be a nonlinear function of the input vector **x**. They are called linear models because this function is linear in **w**. It is this

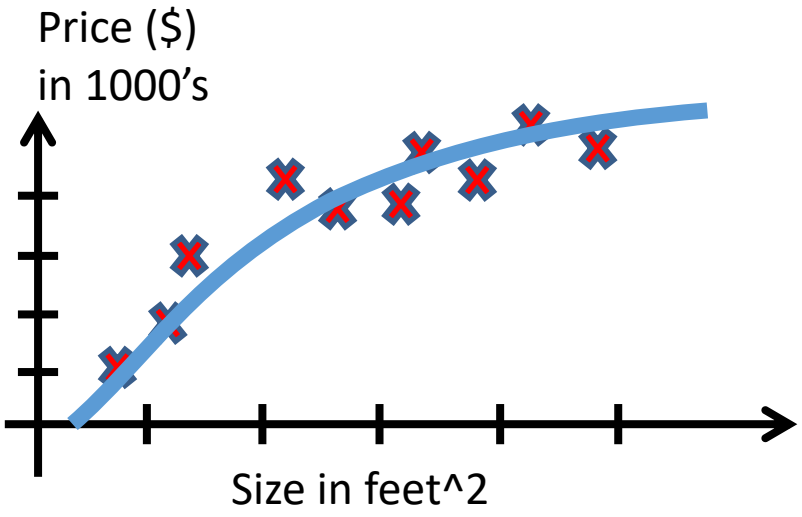
Over fitting, Under fitting, Regularization

Linear regression



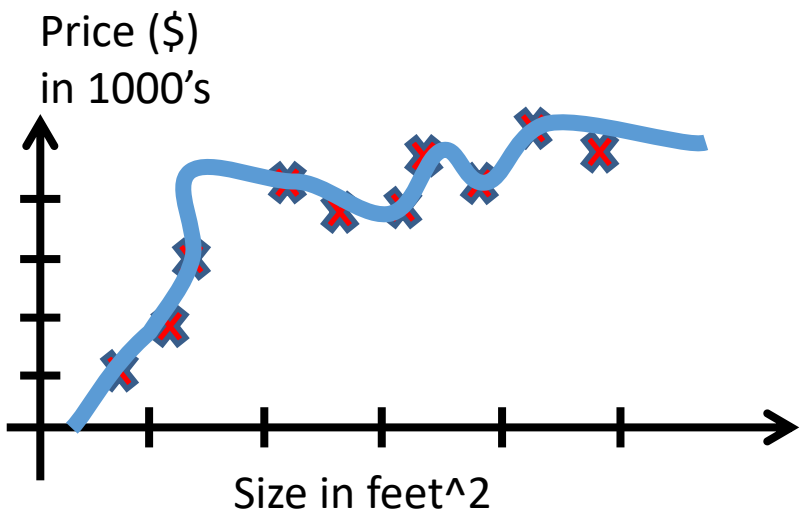
$$h_\theta(x) = \theta_0 + \theta_1 x$$

Underfitting



$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

Just right



$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \cdots$$

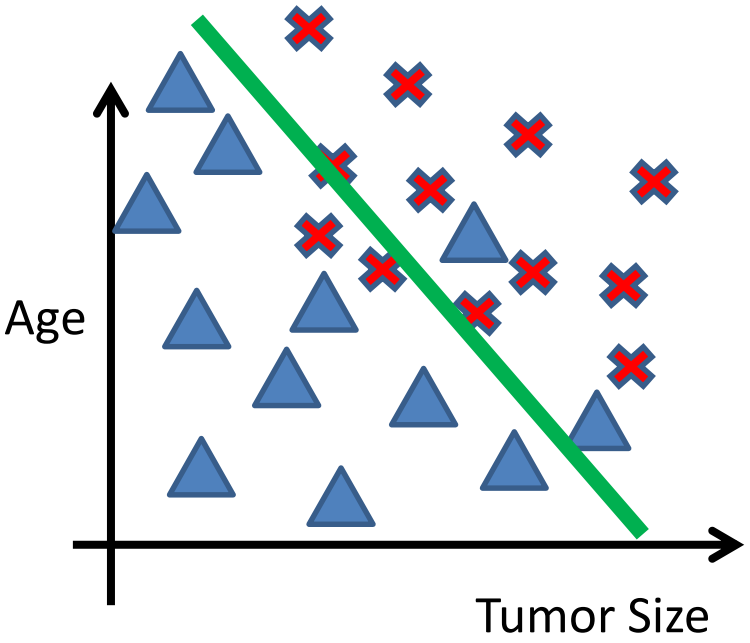
Overfitting

High bias

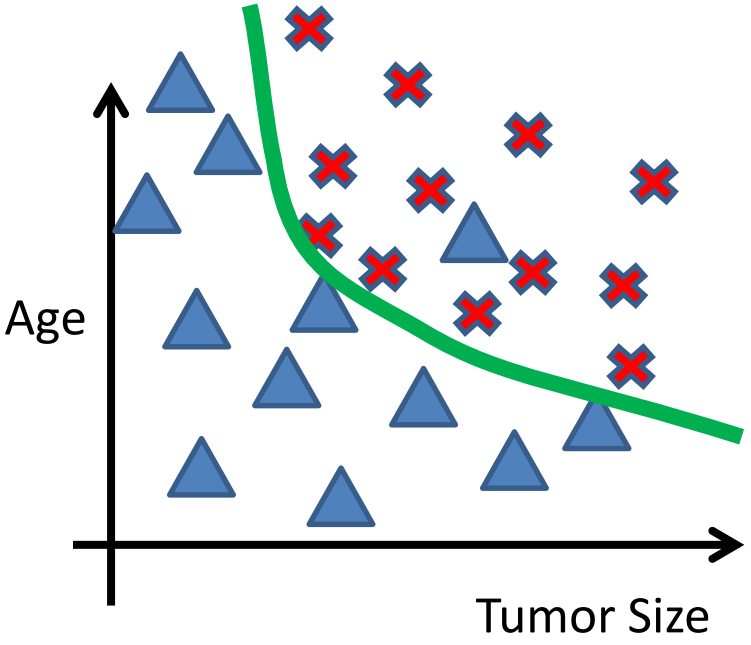
High variance

Slide credit: Andrew Ng

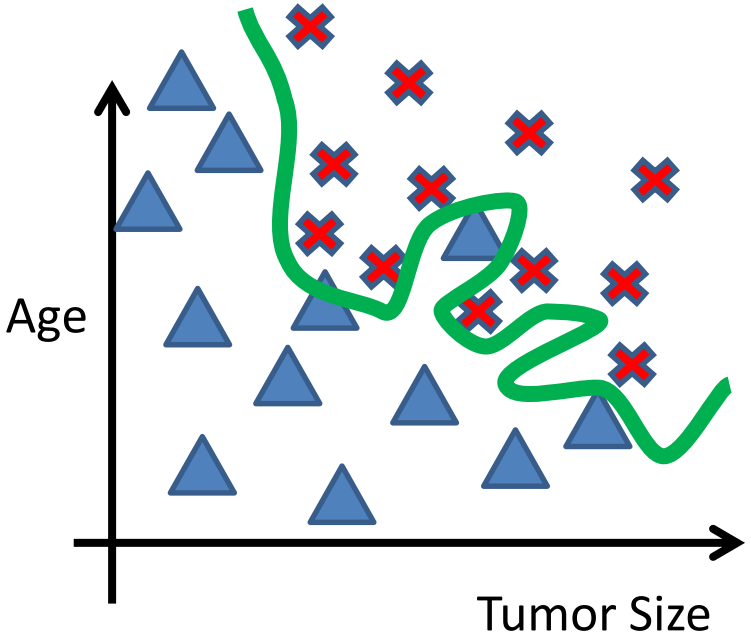
Logistic regression



$$h_\theta(x) = g(\theta_0 + \theta_1 x + \theta_2 x_2)$$



$$h_\theta(x) = g(\theta_0 + \theta_1 x + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$$



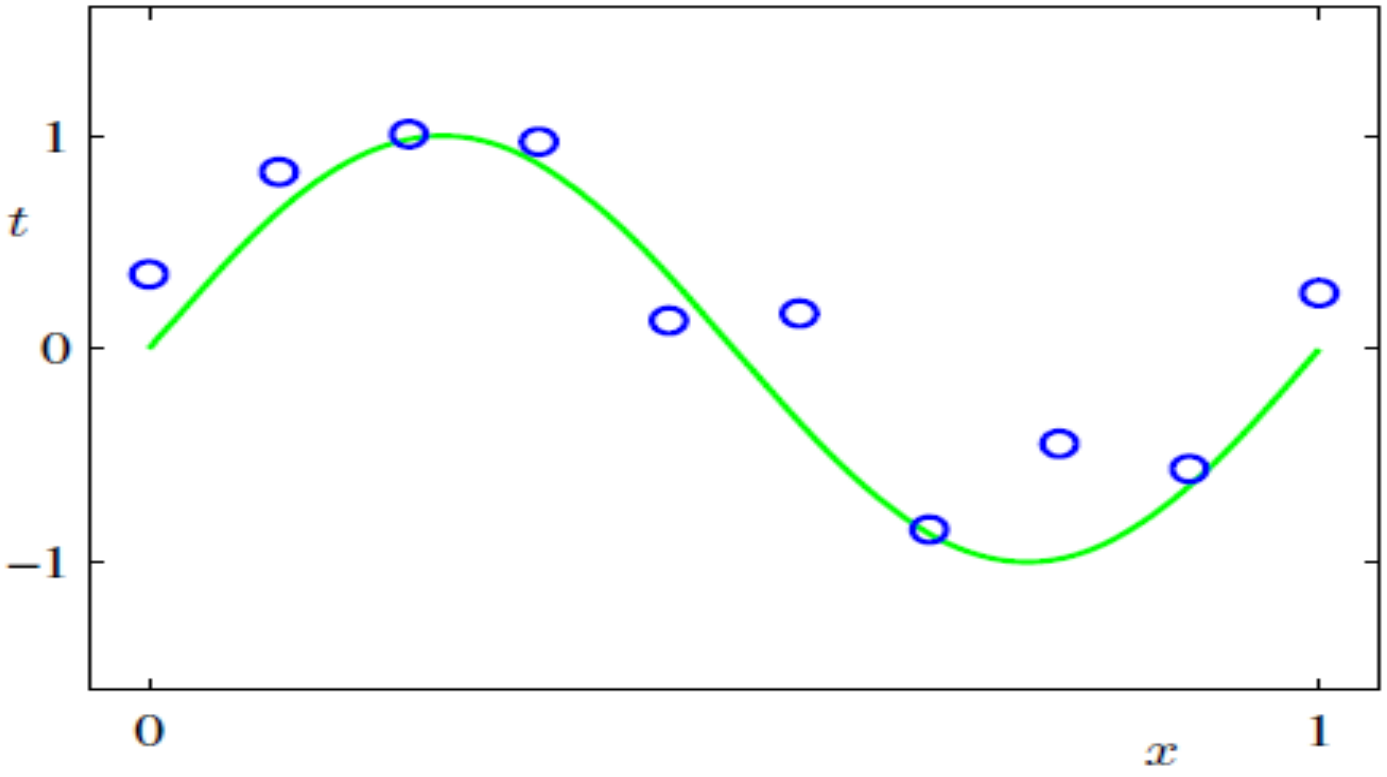
$$h_\theta(x) = g(\theta_0 + \theta_1 x + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2 + \theta_6 x_1^3 x_2 + \theta_7 x_1 x_2^3 + \cdots)$$

Underfitting

Overfitting

Slide credit: Andrew Ng

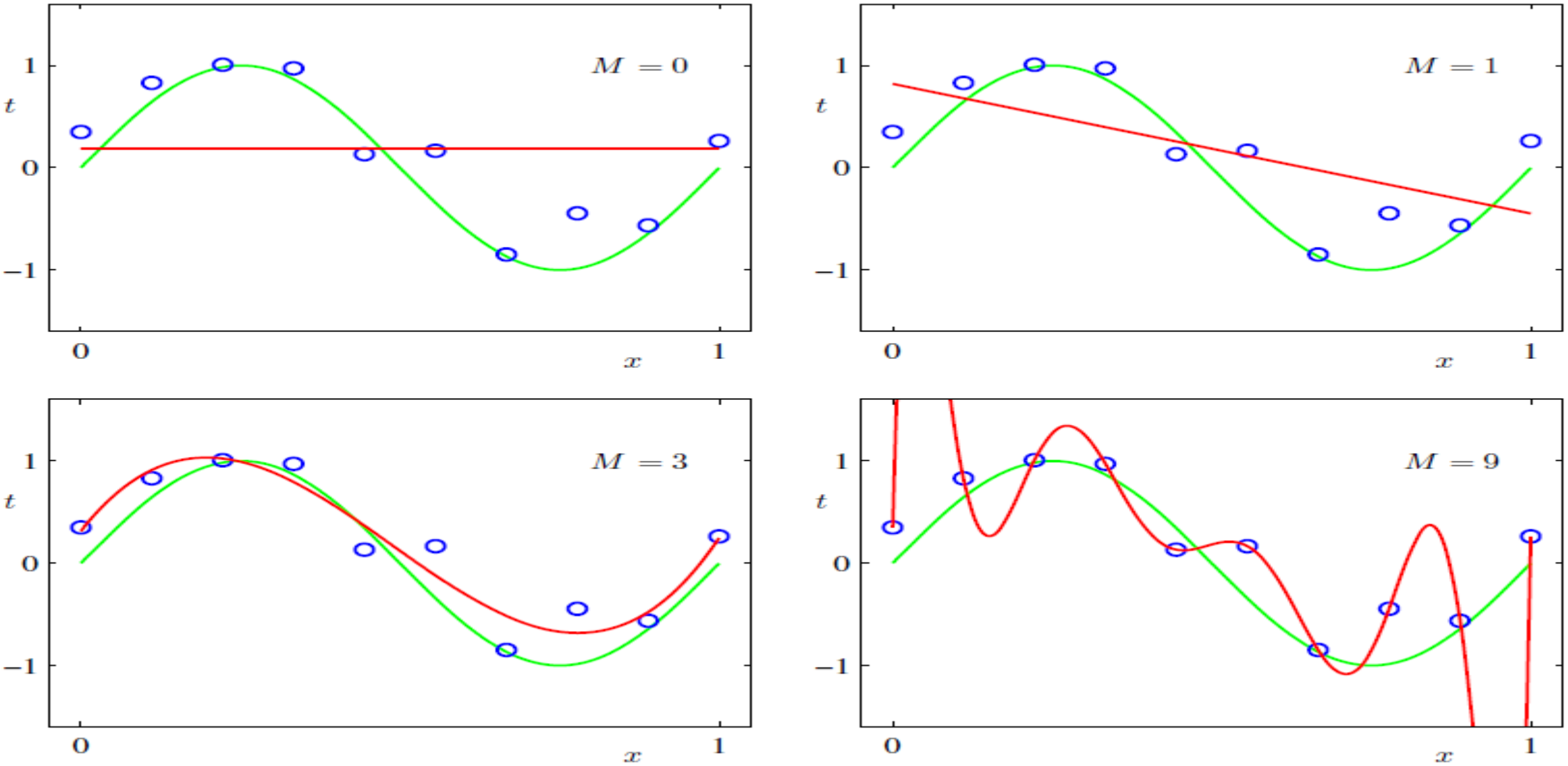
Example: Curve fitting problem



The green curve shows the function sin(2πx) used to generate the data.

F(x) = sin(2Πx) + noise (Gaussian)

Curve fitting problem: Model Selection



Overfitting

- If we have too many features (i.e. complex model), the learned hypothesis may fit the training set very well

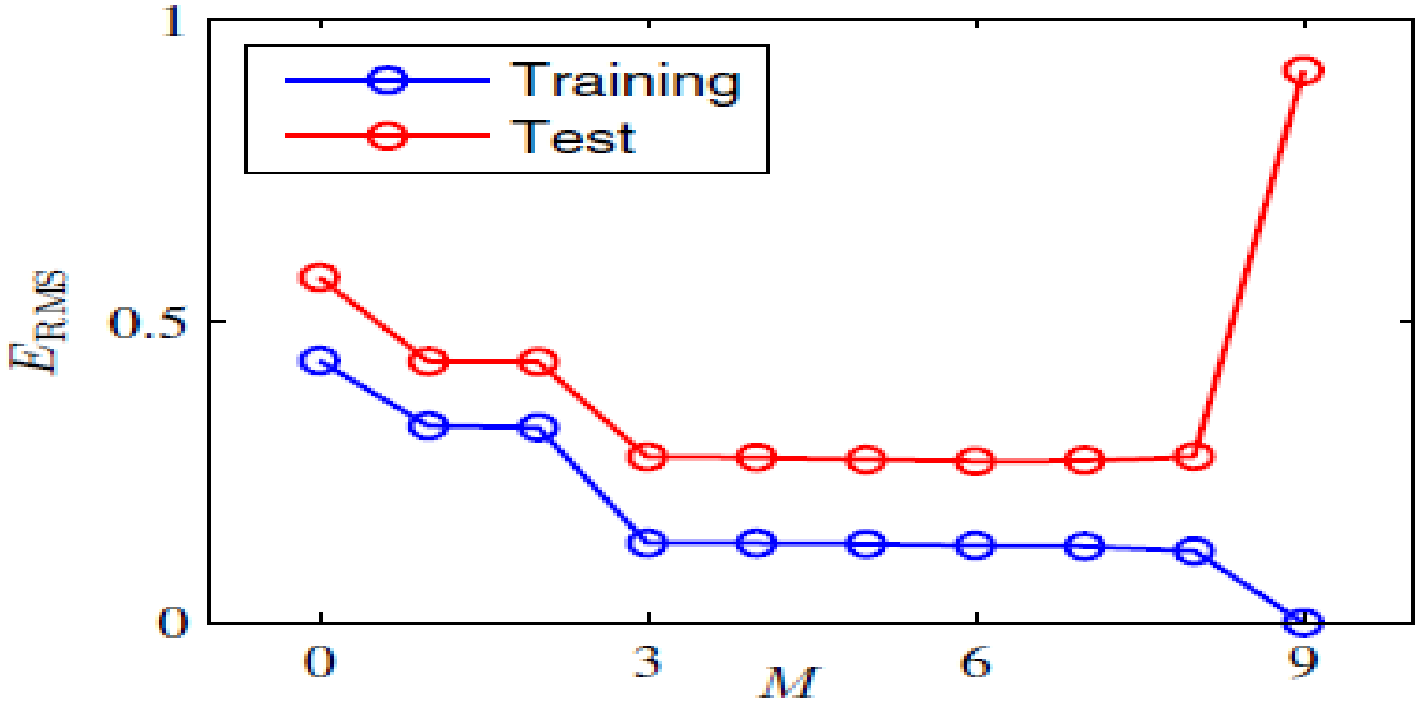
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \approx 0$$

but fail to generalize to new examples

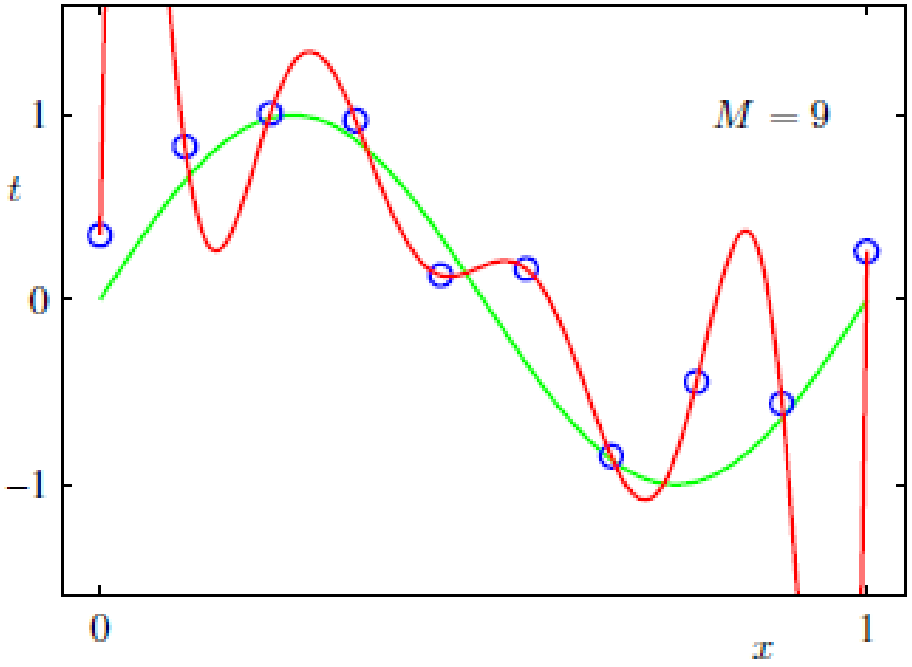
Slide credit: Andrew Ng

Model Selection

- Goal is to achieve good generalization.
- Generalization performance can be evaluated using **test data set**.
- Values of *M* in the range 3 *to* 8 give small values for the test set error, and these also give reasonable representations of the generating function sin(2*πx*)
- For *M* = 9, polynomial contains 10 degrees of freedom corresponding to the 10 coefficients *w*0, . . . , *w*9, can be tuned exactly to the 10 data points



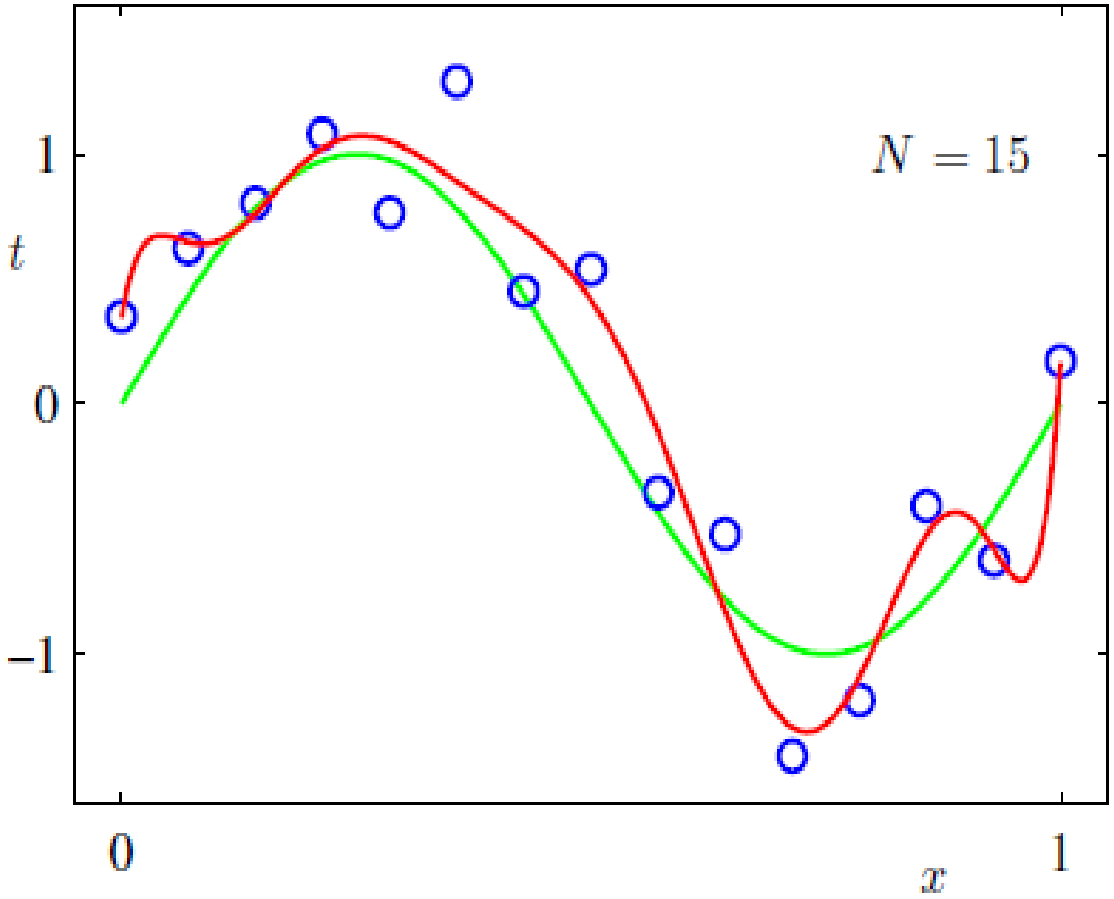
Coefficients for polynomials of various order

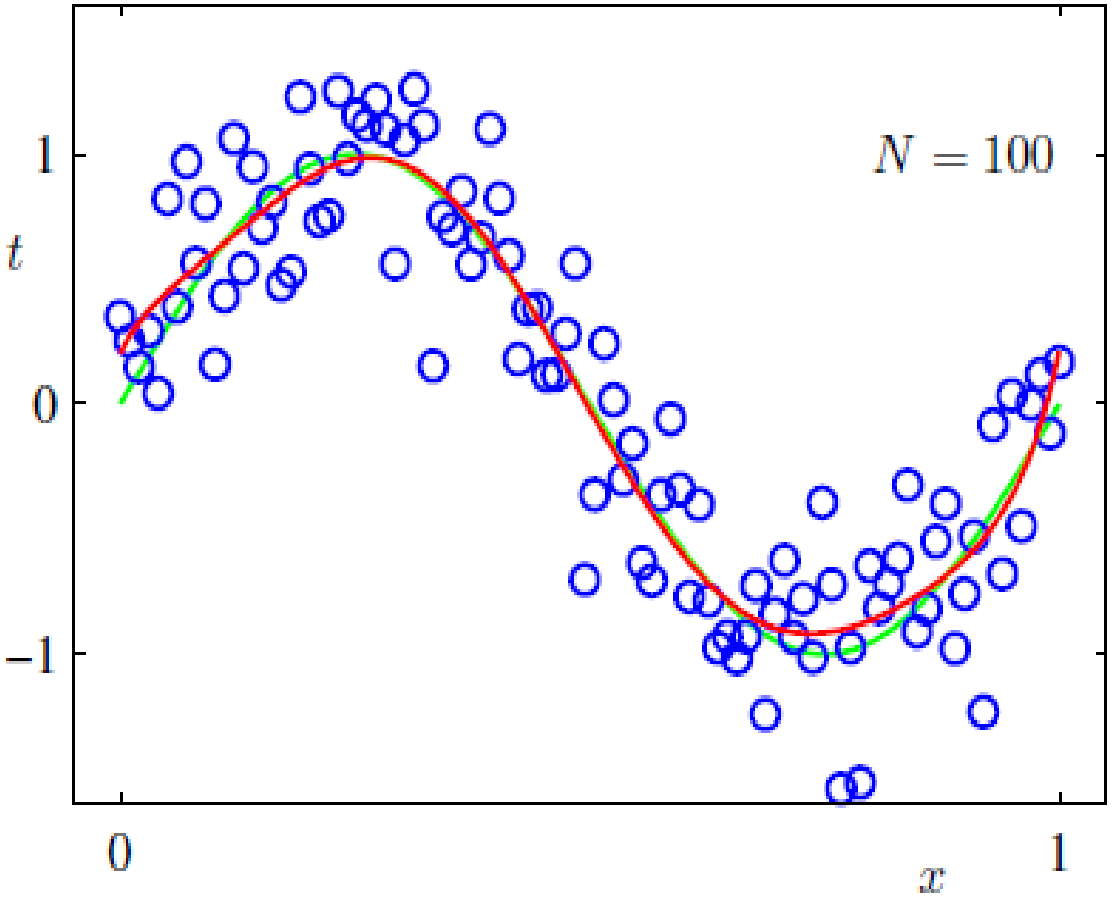


	$M = 0$	$M = 1$	$M = 6$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Coefficients have become finely tuned to the data by developing large positive and negative values, matches each of the data points exactly

Size of the data set reduces the over-fitting problem





Addressing overfitting

- **Reduce number of features**
  - Manually select which features to keep.
  - One rough heuristic says that data points should be multiple (5 to 10) parameters.
  - It seems more reasonable to take number of parameters according to the problem complexity rather that given data points.
- **Any Solution?**
  - **Regularization**
    - Adding a penalty term to the error function in order to discourage the coefficients from reaching large values. E.g a sum of squares of all of the coefficients,
    - Keep all the features, but reduce magnitude/values of parameters θ_j.
    - Works well when we have a lot of features, each of which contributes a bit to predicting y.
    - ***Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.***

Regularization

- Linear regression objective function

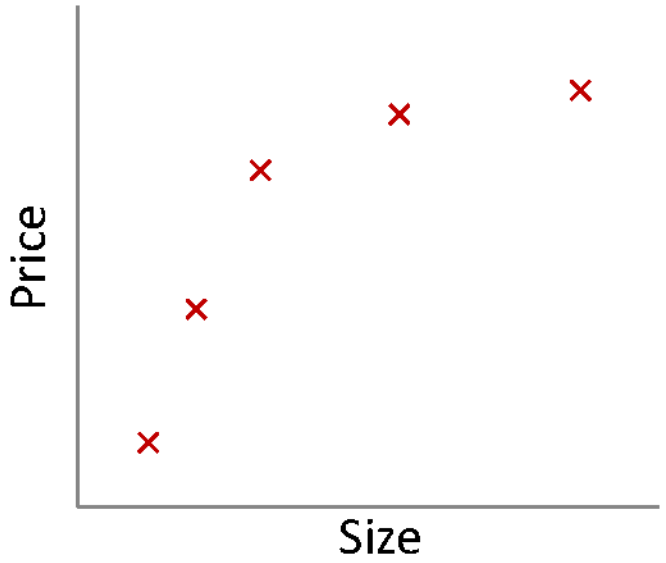
$$J(\boldsymbol{\theta}) = \underbrace{\frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2}_{\text{model fit to data}} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2}_{\text{regularization}}$$

  - λ is the regularization parameter ($\lambda \geq 0$)
  - No regularization on θ_0!

Understanding Regularization

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- What happens if we set λ to be huge (e.g., 10^{10})?



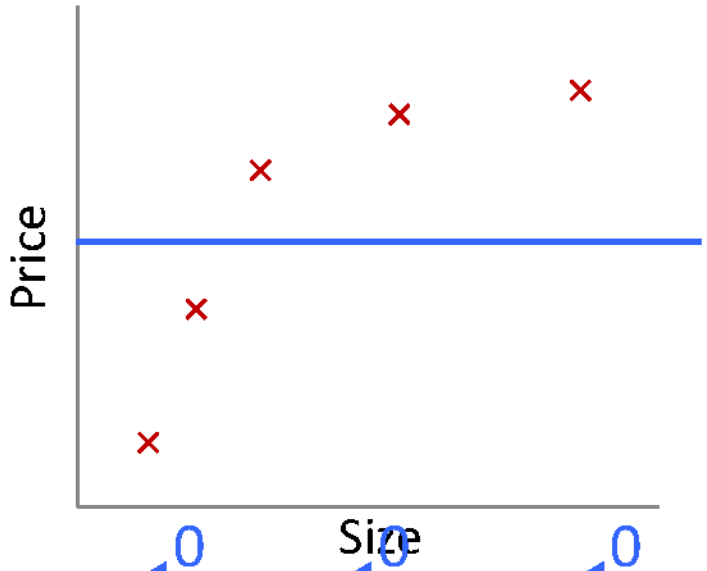
$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Based on example by Andrew Ng

Understanding Regularization

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- What happens if we set λ to be huge (e.g., 10^{10})?



$$\theta_0 + \cancelto{0}{\theta_1}x + \cancelto{0}{\theta_2}x^2 + \cancelto{0}{\theta_3}x^3 + \cancelto{0}{\theta_4}x^4$$

Based on example by Andrew Ng

Regularized linear regression

- Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- Fit by solving , Gradient update $\min_{\theta} J(\theta)$

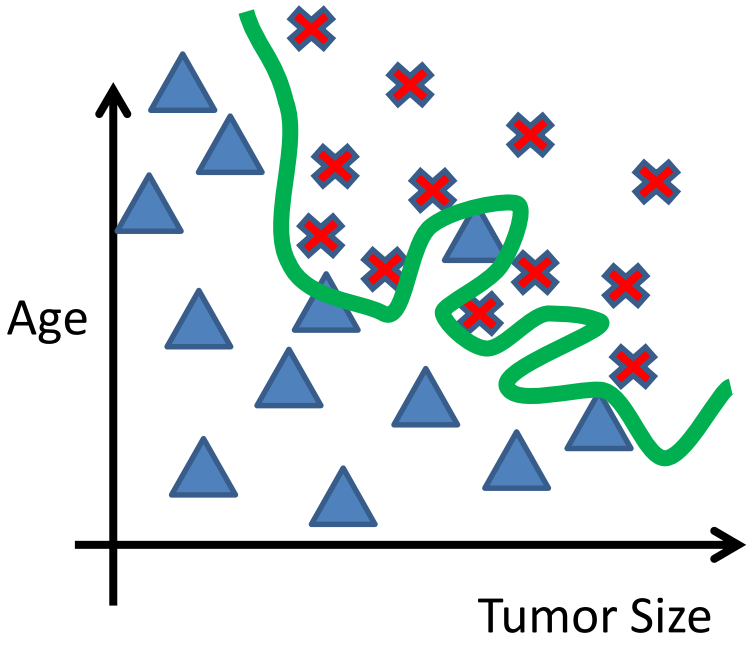
$$\frac{\partial}{\partial\theta_0}J(\theta) \qquad \theta_0 \leftarrow \theta_0 - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)$$

$$\frac{\partial}{\partial\theta_j}J(\theta) \qquad \theta_j \leftarrow \theta_j - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)} \underbrace{- \lambda\theta_j}_{\text{regularization}}$$

- Can be rearranged

$$\theta_j \leftarrow \theta_j\left(1 - \alpha\lambda\right) - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

Regularized logistic regression



$$h_\theta(x) = g(\theta_0 + \theta_1 x + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2 + \theta_6 x_1^3 x_2 + \theta_7 x_1 x_2^3 + \cdots)$$

- Cost function: $J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\ \log(h_\theta(x^{(i)})) + (1 - y^{(i)})\ \log(1 - h_\theta(x^{(i)}))]$

Slide credit: Andrew Ng

Regularized logistic regression

Regularized cost function

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\ \log(h_\theta(x^{(i)})) + (1-y^{(i)})\ \log(1-h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Gradient descent with regularized cost function

$$\begin{aligned}
&\text{Repeat } \{ \\
&\quad \theta_0 := \theta_0 - \alpha\ \frac{1}{m}\ \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)} \\
&\quad \theta_j := \theta_j - \alpha\ \left[\left(\frac{1}{m}\ \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j\right] \qquad j \in \{1, 2...n\} \\
&\}
\end{aligned}$$

Just like with linear regression, we will want to **separately** update θ_0 and the rest of the parameters because we do not want to regularize θ_0.

$|\theta|_1$: Lasso regularization

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda\boxed{\sum_{j=1}^{n}|\theta_j|}\right]$$

LASSO: Least Absolute Shrinkage and Selection Operator

Terminology

Regularization function	Name
$\|\theta\|_2^2 = \sum_{j=1}^{n} \theta_j^2$	Tikhonov regularization Ridge regression
$\left\|\|\theta\|\right\|_1 = \sum_{j=1}^{n} \|\theta_j\|$	LASSO regression
$\alpha\left\|\|\theta\|\right\|_1 + (1-\alpha)\ \|\theta\|_2^2$	Elastic net regularization

α is the mixing parameter between ridge (α = 0) and lasso (α = 1).

Bias-Variance Decomposition

Expectation of a random variable

- Let X be the random variable
- E[X] = **mean** of a large number of observations of the random variable
- The expectation of any constant c: E[c]=c
- E[E[X]]=E[X] because the expectation of a random variable is a constant.
- E[X+Y]=E[X]+E[Y]
- E[cX] = E[c] E[X]= cE[X]

Bias-Variance Tradeoff

- **Bias:** difference between
  what you expect to learn and **truth**
  - Measures how well you expect to represent true solution
  - Decreases with more complex model
- **Variance:** difference between
  what you expect to learn and **what you learn from a particular dataset**
  - Measures how sensitive learner is to specific dataset
  - Increases with more complex model

generalisation error = Bias^2 +Variance +Irreducible Error

Bias – Variance decomposition of error

$$E_{D,\varepsilon}\left\{\left(f(x)+\varepsilon-h_D(x)\right)^2\right\}$$

dataset and noise

true function

noise

learned from D

Fix test case *x*, then do this experiment:

1. Draw size *n* sample D=$(x_1,y_1),\ldots.(x_n,y_n)$
2. Train linear regressor h_D *using* D
3. Draw one test example $(x, f(x)+\varepsilon)$
4. Measure squared error of h_D on that example *x*
5. What's the expected error?

Bias – Variance decomposition of error

Notation - to simplify this

$$f \equiv f(x) + \varepsilon \qquad \hat{y} = \hat{y}_D \equiv h_D(x)$$

$$E_{D,\varepsilon}\left\{\left(f(x)+\varepsilon-h_D(x)\right)^2\right\}$$

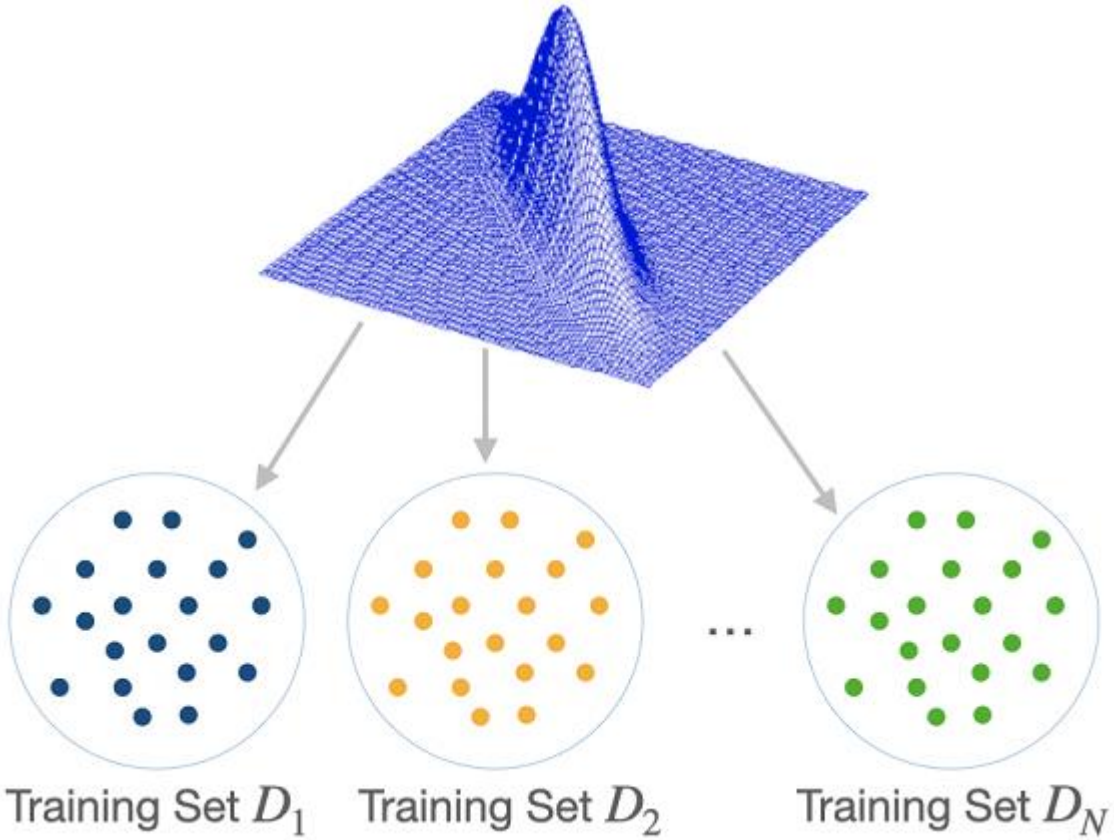
dataset and noise

true function

noise

learned from D

Bias – Variance decomposition of error



$$h \equiv E_D\{h_D(x)\}$$

long-term expectation of learner's prediction on this *x* averaged over many data sets *D*

Bias – Variance decomposition of error

$$h = E_D\{\hat{y}_D\}$$

$$\hat{y} = \hat{y}_D \equiv h_D(x)$$

$$f \equiv f(x) + \varepsilon$$

$$E_{D,\varepsilon}\{(f-y)^2\}$$

$$= E\left\{([f-h]+[h-\hat{y}])^2\right\}$$

$$= E\left\{[f-h]^2+[h-\hat{y}]^2+2[f-h][h-\hat{y}]\right\}$$

$$= E\left\{[f-h]^2+[h-\hat{y}]^2+2[fh-f\hat{y}-h^2+h\hat{y}]\right\}$$

$$= E[(f-h)^2]+E[(h-\hat{y})^2]+2\left(\cancel{E[fh]-E[f\hat{y}]-E[h^2]+E[h\hat{y}]}\right)$$

Bias – Variance decomposition of error

$$E_{D,\varepsilon}\{(f-\hat{y})^2\} = E[(f-h)^2] + E[(h-\hat{y})^2]$$

$$h \equiv E_D\{h_D(x)\}$$
$$\hat{y} = \hat{y}_D \equiv h_D(x)$$
$$f \equiv f(x) + \varepsilon$$

BIAS^2

Squared difference between <u>best possible</u> prediction for x, f(x), and our <u>"long-term" expectation for what the learner will do</u> if we averaged over many datasets D, $E_D[h_D(x)]$

VARIANCE

Squared difference btwn our long-term expectation for the learners performance, $E_D[h_D(x)]$, and what we <u>expect in a representative run on a dataset D</u> (hat y)

Bayesian linear regression

Bayesian analysis

- Bayesian analysis will show that
  - under certain assumptions any learning algorithm that minimizes the squared error between the output hypothesis predictions and the training data will output a maximum likelihood hypothesis

Training Linear Regression: MCLE, MLE

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} \ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

The first term in this expression is a constant independent of h, and can therefore be discarded, yielding

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} -\frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Maximizing this negative quantity is equivalent to minimizing the corresponding positive quantity.

$$h_{ML} = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{m} \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Finally, we can again discard constants that are independent of h.

$$h_{ML} = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{m} (d_i - h(x_i))^2 \tag{6.6}$$

References

- https://www.desmos.com/calculator/bgontvxotm
- https://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf

Thanks