EECS 545: Machine Learning

Lecture 3. Linear Regression

Honglak Lee

1/12/2011

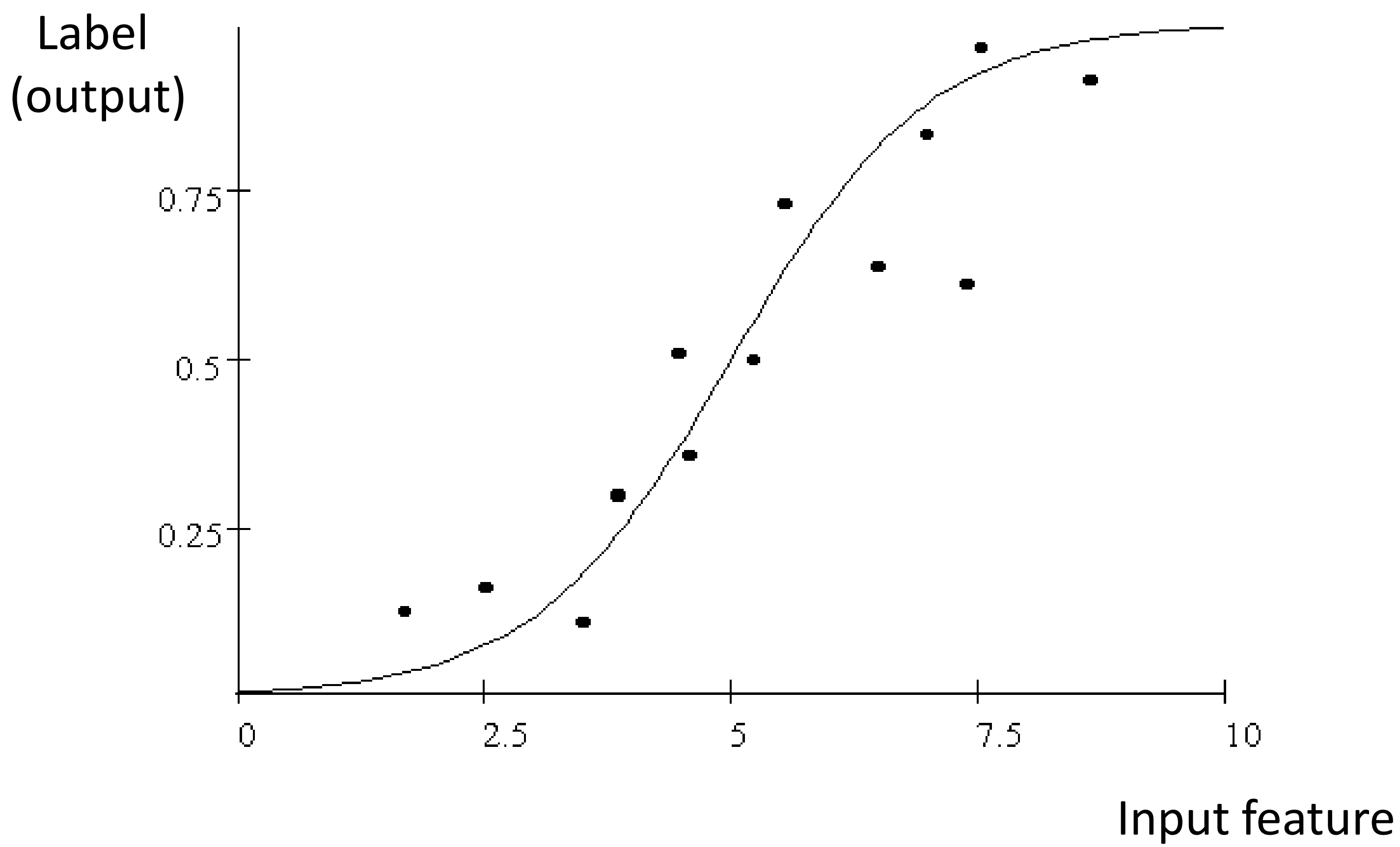
Outline

- Recap: Linear regression
- Regularized Linear Regression
- Locally-weighted Linear Regression
- Kernel Regression
- Classification: K Nearest Neighbor

Supervised Learning

- Goal:
  - Given data X in feature space and the labels Y
  - Learn to predict Y from X
- Labels could be discrete or continuous
  - Discrete labels: classification
  - Continuous labels: regression

Supervised Learning - Regression



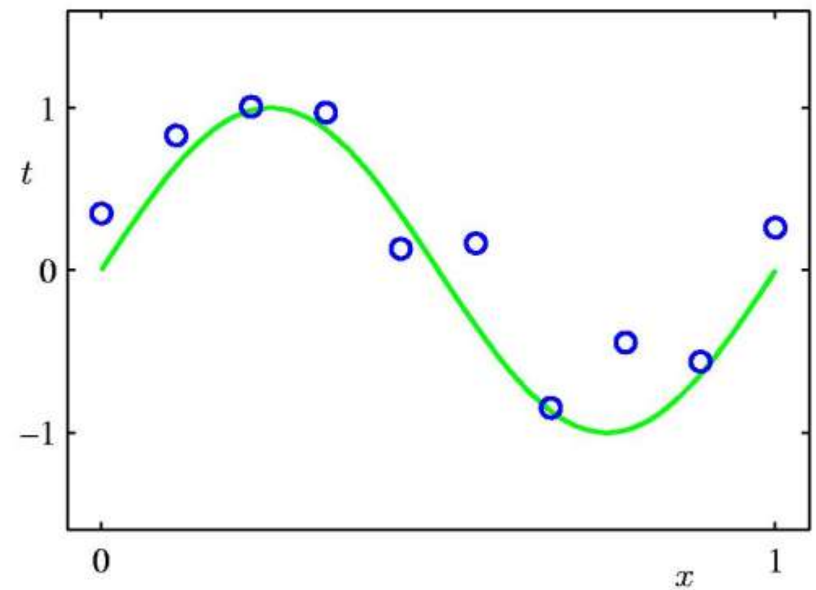
"Learning regression function f(X)"

Notation

- In this lecture, we will use
  - x: data (scalar or vector)
  - $\phi(x)$: features for x
  - t (or y): continuous-valued labels (target values)
- We will interchangeably use
  - $x^{(n)} \stackrel{\text{def}}{=} x_n$ to denote n-th training example.
  - $t^{(n)} \stackrel{\text{def}}{=} t_n$ to denote n-th target value.

Regression

- Given a set of observations
  - $\mathbf{x} = \{ x_1 \; . \; . \; . \; x_N \}$
- And corresponding target values:
  - $\mathbf{t} = \{ t_1 \; . \; . \; . \; t_N \}$



- We want to learn a function $y(x,w)=t$ to predict future values.

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

Slide credit: Ben Kuipers

Linear Regression

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

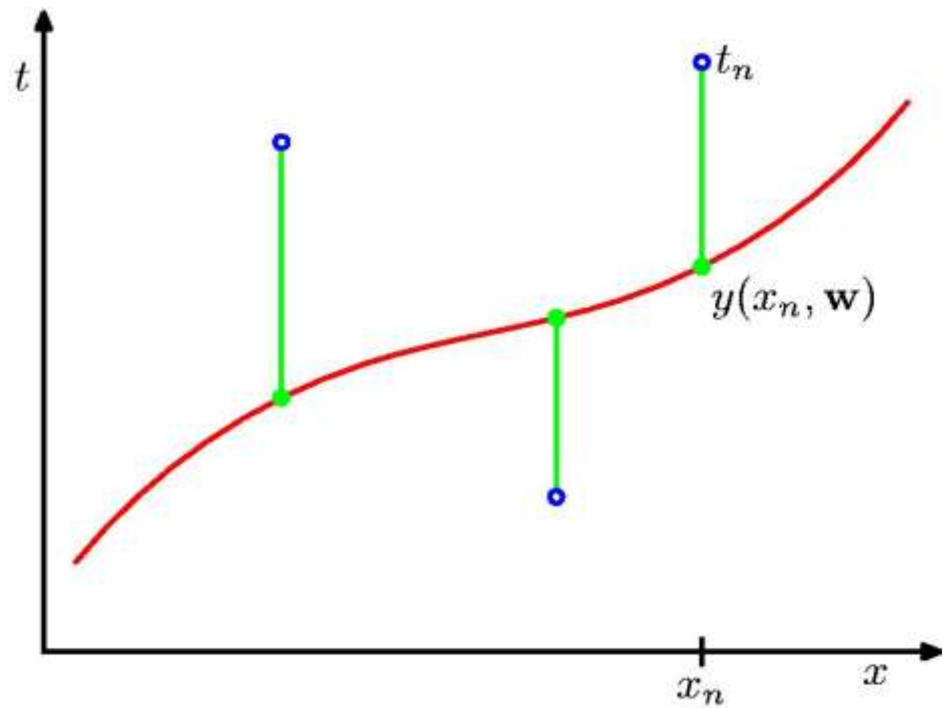
- The function *y*(x,w) is linear in parameters w.
  - Goal: find the best value for the weights, **w**.
- For simplicity, add a *bias function* $\phi_0(\mathbf{x}) = 1$

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

$$\mathbf{w} = (w_0, \ldots, w_{M-1})^T \qquad \phi = (\phi_0, \ldots, \phi_{M-1})^T$$

Slide credit: Ben Kuipers

Sum-of-Squares Error Function



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

Least squares problem

- Objective function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j\phi_j(x^{(n)}) - t^{(n)})^2$$

- Gradient

$$\begin{aligned}
\frac{\partial E(\mathbf{w})}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j\phi_j(x^{(n)}) - t^{(n)})^2 \\
&= \sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j\ \phi_j\ (x^{(n)}) - t^{(n)})\frac{\partial}{\partial w_j}(\sum_{j=0}^{M-1} w_j\ \phi_j\ (x^{(n)}) - t^{(n)}) \\
&= \sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j\ \phi_j\ (x^{(n)}) - t^{(n)})\phi_j(x^{(n)})
\end{aligned}$$

Least squares problem

- Gradient (compact, vectorized form)

$$\begin{aligned}\nabla_{\mathbf{w}} E(\mathbf{w}) &= \sum_{n=1}^{N} \left(\sum_{j'=0}^{M-1} w_{j'} \phi_{j'}(x^{(n)}) - t^{(n)}\right) \phi(x^{(n)}) \\ &= \sum_{n=1}^{N} (\mathbf{w}^T \phi(x^{(n)}) - t^{(n)}) \phi(x^{(n)})\end{aligned}$$

Batch Gradient Descent

- Given data (x, y), initial w
  - Repeat until convergence

$$\mathbf{w} := \mathbf{w} - \eta \nabla_{\mathbf{w}} E(\mathbf{w})$$

where

$$\begin{aligned} \nabla_{\mathbf{w}} E(\mathbf{w}) &= \sum_{n=1}^{N} \Big(\sum_{j'=0}^{M-1} w_{j'} \phi_{j'}(x^{(n)}) - t^{(n)} \Big) \phi(x^{(n)}) \\ &= \sum_{n=1}^{N} (\mathbf{w}^T \phi(x^{(n)}) - t^{(n)}) \phi(x^{(n)}) \end{aligned}$$

Stochastic Gradient Descent

- Main idea: instead of computing batch gradient (over entire training data), just compute gradient for individual example and update
- Repeat until convergence
  - for n=1,…,N

$$\mathbf{w} := \mathbf{w} - \eta \nabla_{\mathbf{w}} E(\mathbf{w}|x^{(n)})$$

where

$$\begin{aligned}\nabla_{\mathbf{w}} E(\mathbf{w}|x^{(n)}) &= \left(\sum_{j'=0}^{M-1} w_{j'}\phi_{j'}(x^{(n)}) - t^{(n)}\right)\phi(x^{(n)}) \\ &= (\mathbf{w}^T\phi(x^{(n)}) - t^{(n)})\phi(x^{(n)})\end{aligned}$$

Closed form solution

- Main idea:
  - Compute gradient and set gradient to 0. (condition for optimal solution)
  - Solve the equation in a closed form
- Objective function:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} (\sum_{j=0}^{M-1} w_j \phi_j(x^{(n)}) - t^{(n)})^2$$

- We will derive the gradient from matrix calculus

Closed form solution

- Objective function:

$$
\begin{aligned}
E(\mathbf{w}) &= \frac{1}{2}\sum_{n=1}^{N}\Big(\sum_{j=0}^{M-1} w_j\phi_j(x^{(n)}) - t^{(n)}\Big)^2 \\
&= \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^T\phi(x^{(n)}) - t^{(n)})^2 \\
&= \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^T\phi(x^{(n)}))^2 - \sum_{n=1}^{N} t^{(n)}\mathbf{w}^T\phi(x^{(n)}) + \frac{1}{2}\sum_{n=1}^{N} t^{(n)2} \\
&= \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}^T\Phi^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t}
\end{aligned}
$$

- Recap: matrix calculus (check previous review session)

The Data

- The design matrix is an NxM matrix, applying
  - the M basis functions (across)
  - to N data points (down)

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}$$

$$\boldsymbol{\Phi}\mathbf{w} \approx \mathbf{t}$$

Slide credit: Ben Kuipers

For polynomial fitting,

$$\Phi = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{M-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{M-1} \\ \vdots & & & & \\ 1 & x_N & x_N^2 & \cdots & x_N^{M-1} \end{bmatrix}$$

$$= \begin{bmatrix} - & \phi(x_1)^T & - \\ - & \phi(x_2)^T & - \\ & \vdots & \\ - & \phi(x_N)^T & - \end{bmatrix}$$

Gradients and Hessians of Quadratic and Linear Functions (Recap)

- $\nabla_x b^T x = b$
- $\nabla_x x^T A x = 2Ax$ (if A symmetric)
- $\nabla_x^2 x^T A x = 2A$ (if A symmetric)

Gradient via matrix calculus

- Compute gradient and set to zero

$$\begin{aligned}\nabla_{\mathbf{w}} E(\mathbf{w}) &= \nabla_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}^T\Phi^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} \\ &= \Phi^T\Phi\mathbf{w} - \Phi^T\mathbf{t}\end{aligned}$$

- Solve the resulting equation (normal equation)

$$\Phi^T\Phi\mathbf{w} = \Phi^T\mathbf{t}$$
$$\mathbf{w}_{ML} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\mathbf{t}$$

This is the *Moore-Penrose pseudo-inverse*: $\boldsymbol{\Phi}^\dagger = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T$

applied to: $\boldsymbol{\Phi}\mathbf{w} \approx \mathbf{t}$

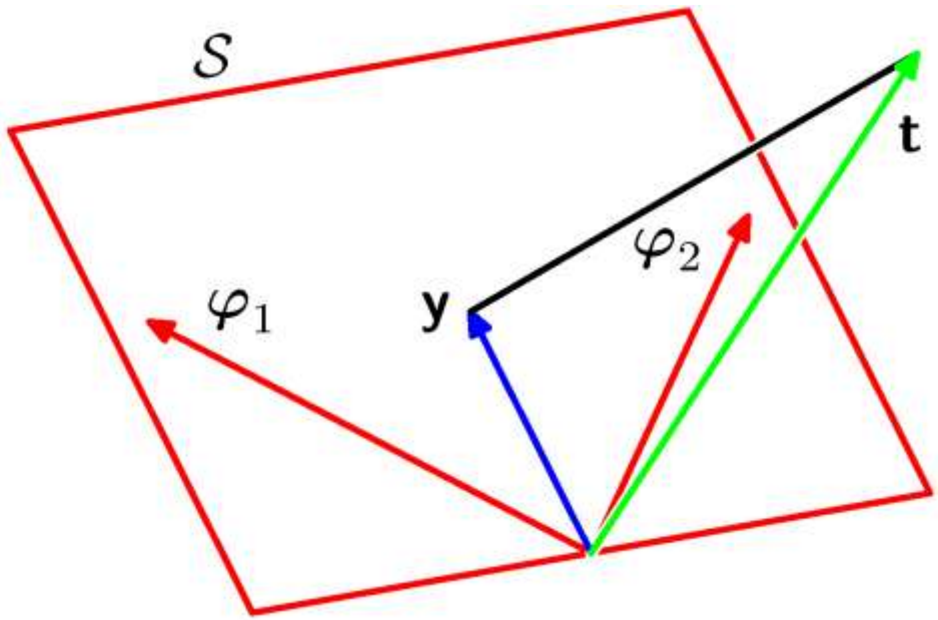
Geometric Interpretation

- Assuming many more observations (N) than the M basis functions $\phi_j(\mathbf{x})$
- View the observed target values t={t_1 … t_N} as a vector in an N-dimensional space.
- The M basis functions $\phi_j(\mathbf{x})$ span an M-dimensional subspace.
- $y(\mathrm{x},\mathrm{w}_{ML})$ is the point in the subspace with minimal squared error from t.
- It's the projection of t onto that subspace.

Slide credit: Ben Kuipers

Geometric Interpretation

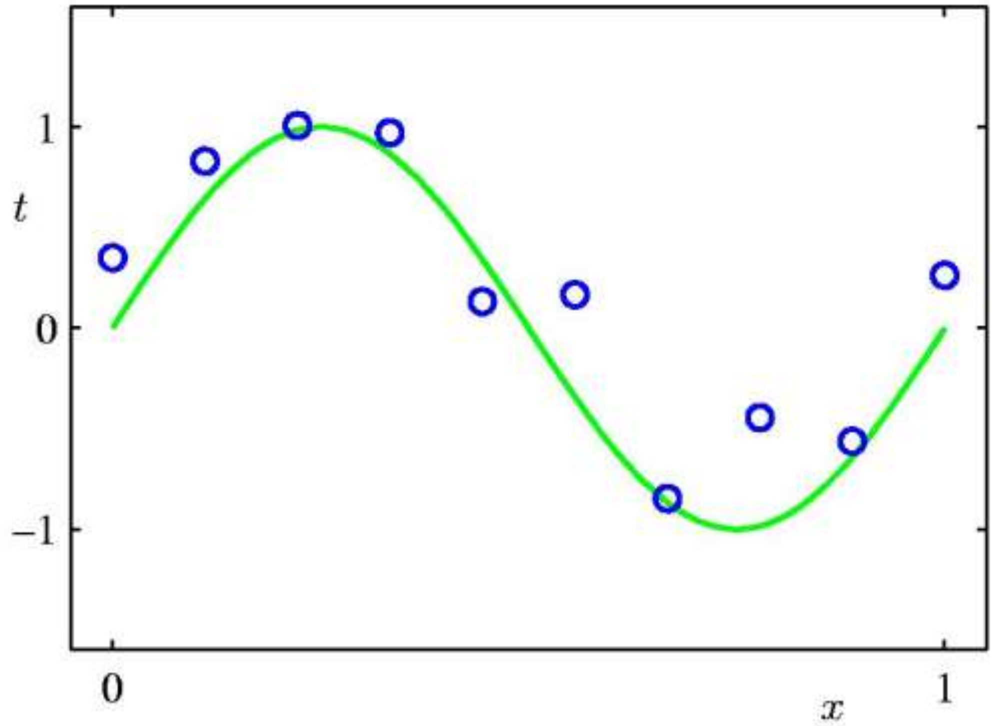
- $y(\mathrm{x},\mathrm{w}_{\mathrm{ML}})$ is the projection of t onto the subspace spanned by the M basis functions $\phi_j(\mathbf{x})$



Slide credit: Ben Kuipers

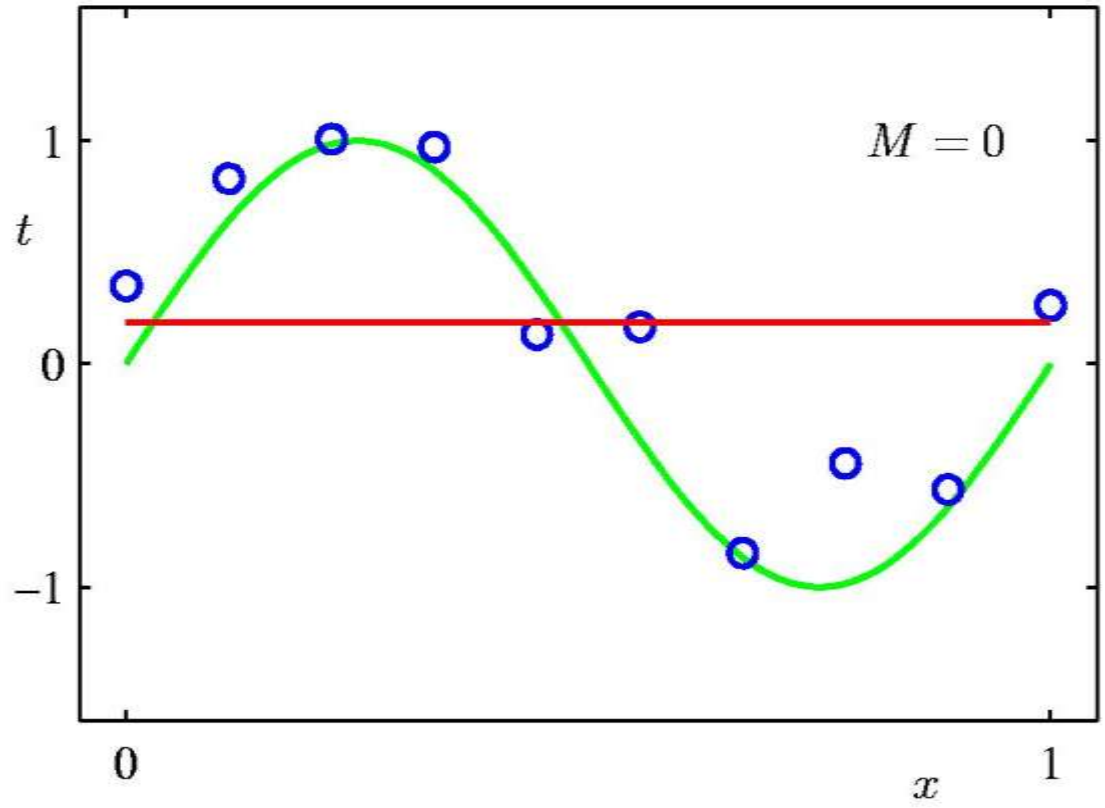
Back to curve-fitting examples

Polynomial Curve Fitting

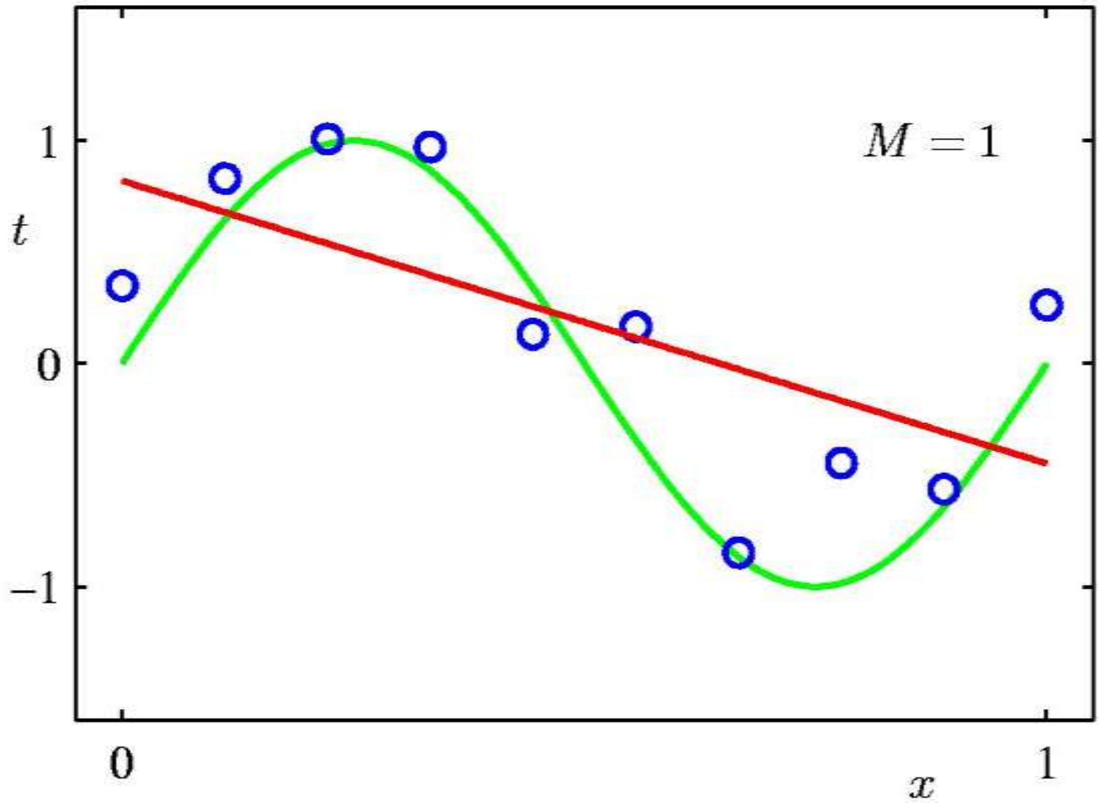


$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

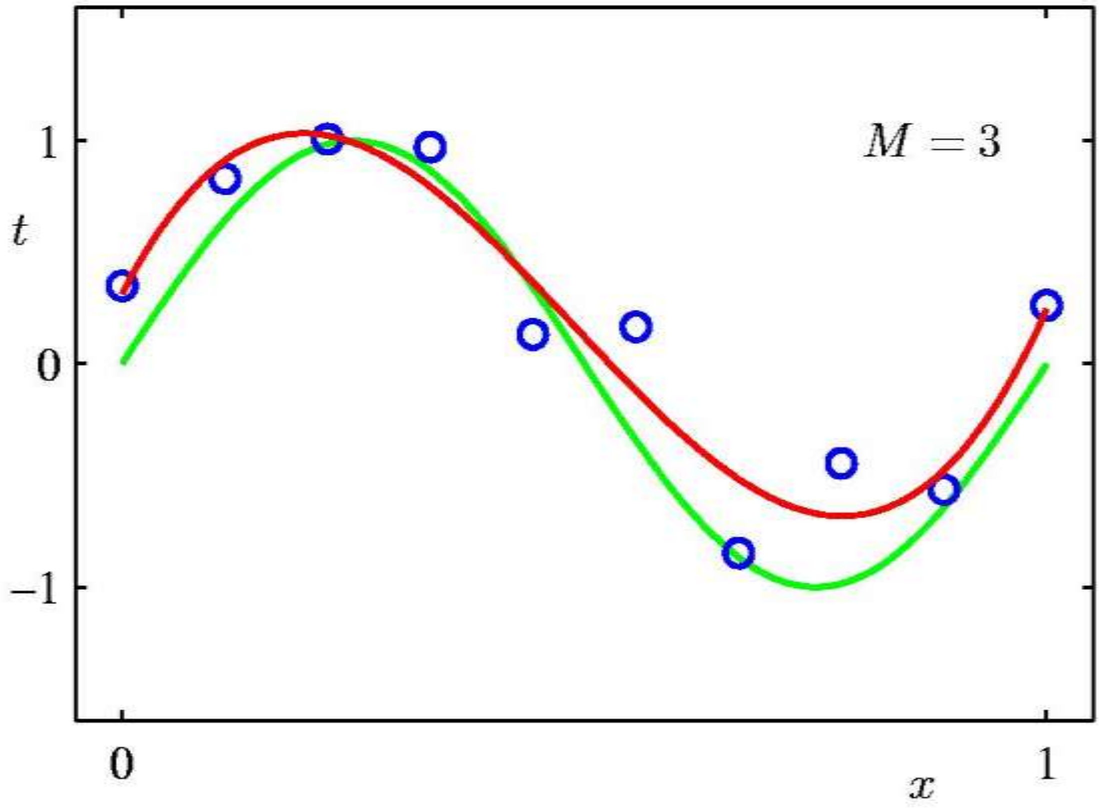
0^{th} Order Polynomial



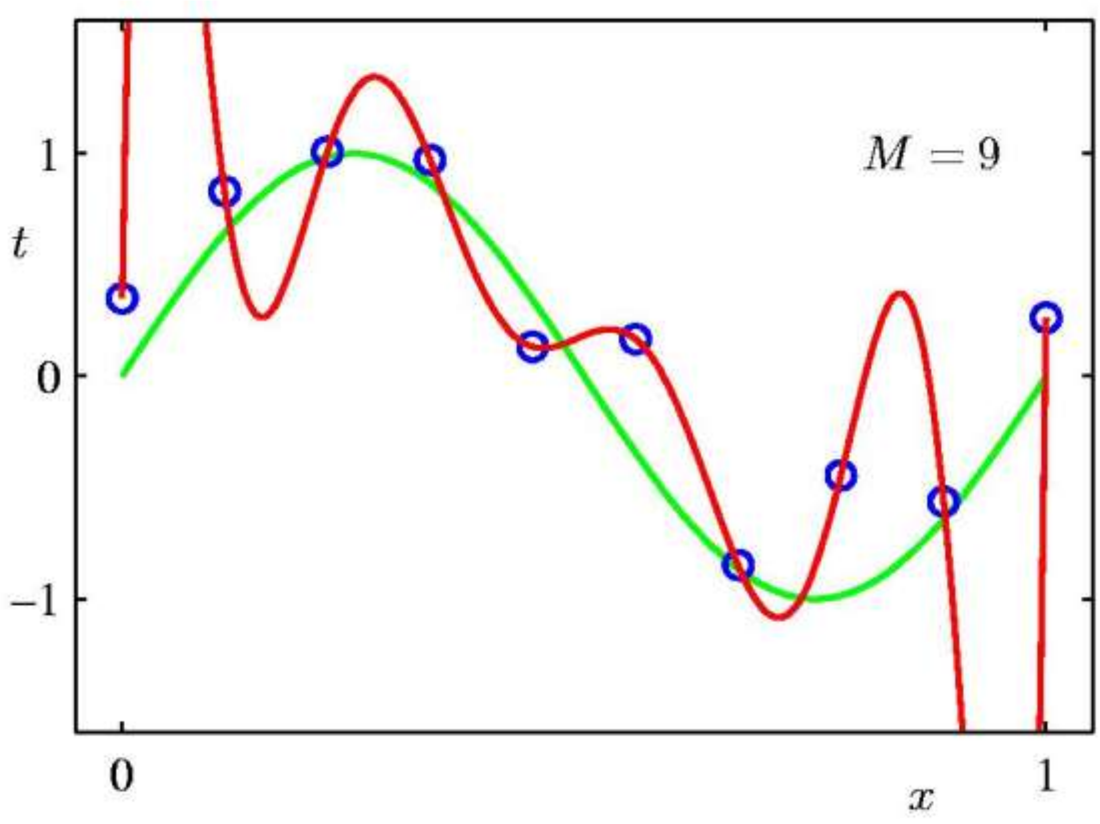
1st Order Polynomial



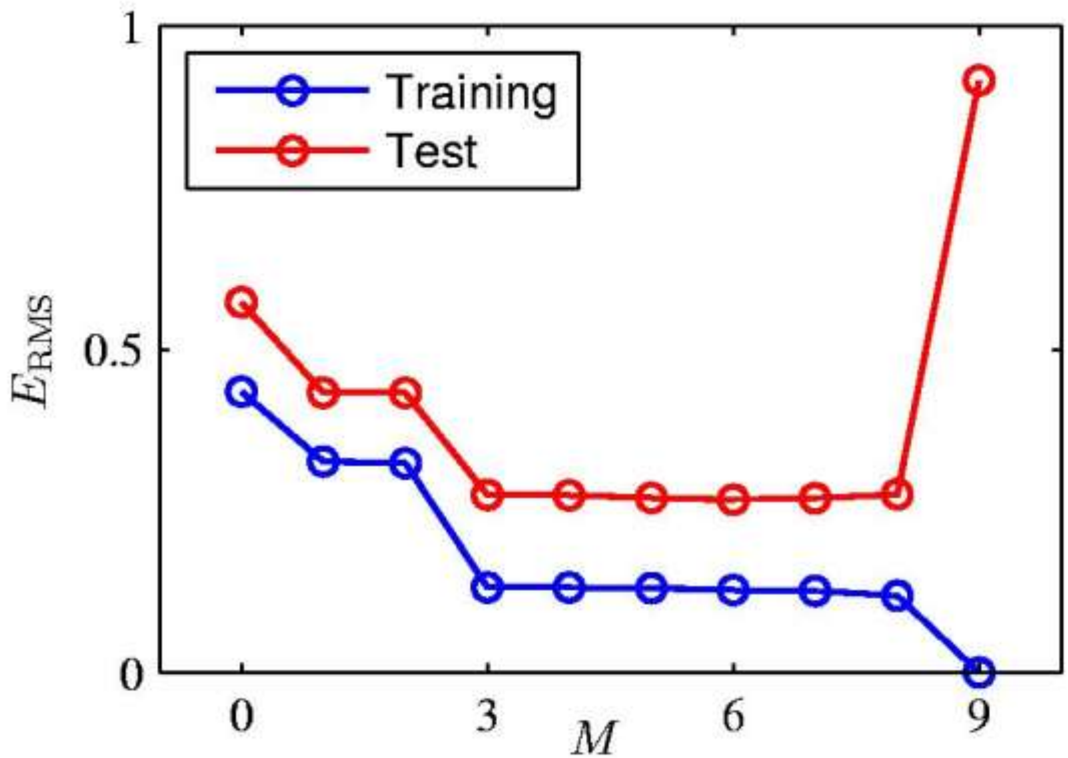
3^{rd} Order Polynomial



9th Order Polynomial



Over-fitting



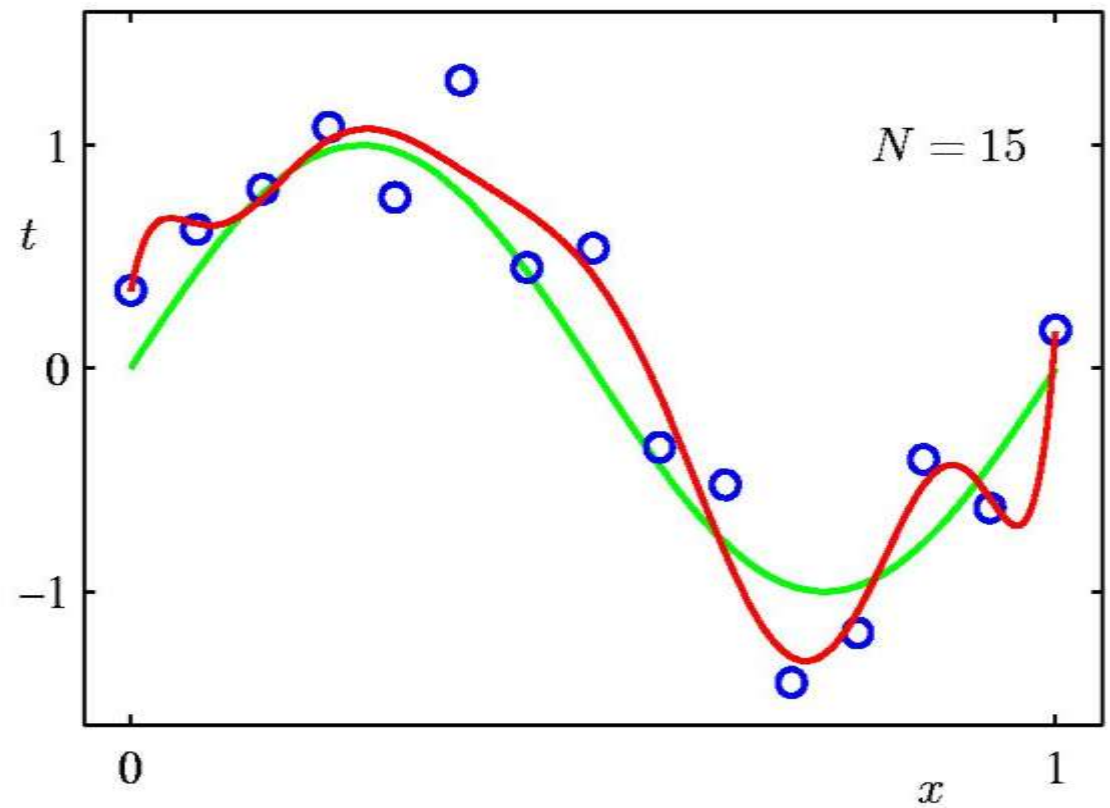
Root-Mean-Square (RMS) Error: $E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^\star)/N}$

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

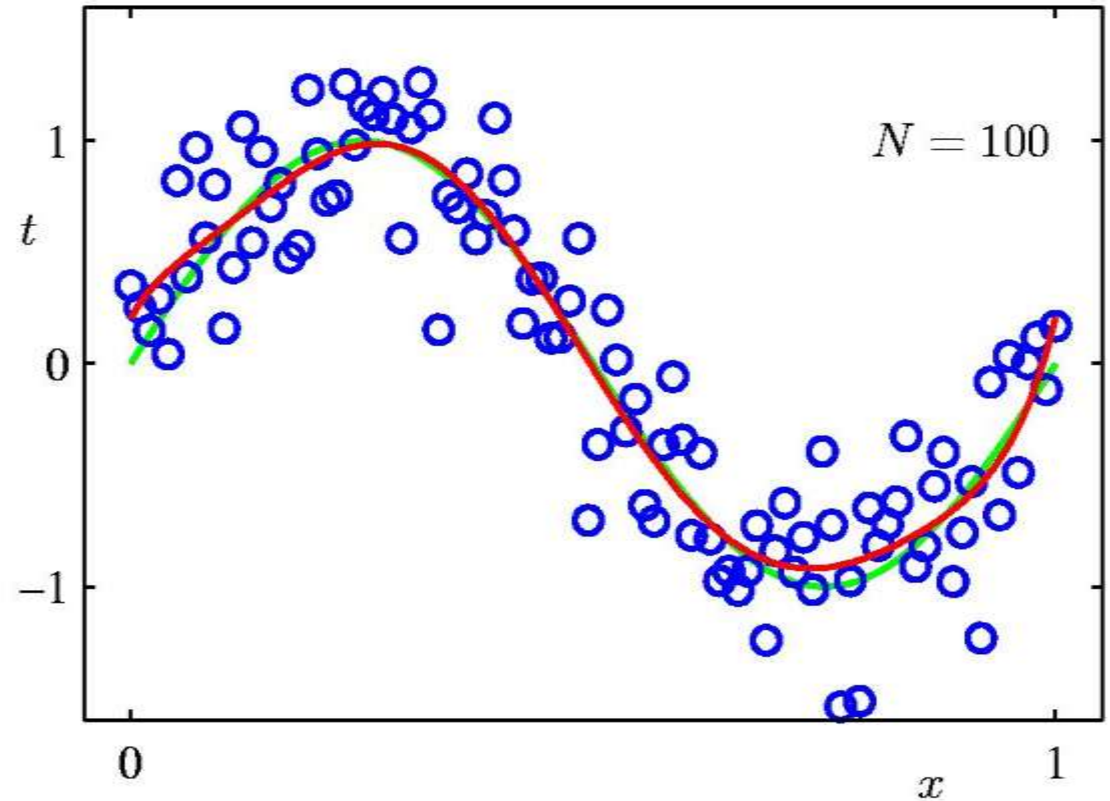
Data Set Size: $N = 15$

9^{th} Order Polynomial



Data Set Size: $N = 100$

9th Order Polynomial



Q. How do we choose the degree of polynomial?

Regularized Linear Regression

Regularized Least Squares (1)

- Consider the error function:

$$E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})$$

Data term + Regularization term

- With the sum-of-squares error function and a quadratic regularizer, we get

Penalize large coefficient values

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

λ is called the regularization coefficient.

- which is minimized by

$$\mathbf{w} = \left(\lambda \mathbf{I} + \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t}.$$

Derivation

$Ax=b$

① $\mathrm{inv}(A)*b$ (X)

② $A \backslash b$ (✓)

Objective function

$$
\begin{aligned}
E(\mathbf{w}) &= \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^T\phi(x^{(n)}) - t^{(n)})^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \\
&= \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}^T\Phi^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}
\end{aligned}
$$

Compute gradient and set it zero:

$$
\begin{aligned}
\nabla_{\mathbf{w}}E(\mathbf{w}) &= \nabla_{\mathbf{w}}[\frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}^T\Phi^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}] \\
&= \Phi^T\Phi\mathbf{w} - \Phi^T\mathbf{t} + \lambda\mathbf{w} \\
&= (\Phi^T\Phi + \lambda\mathbf{I})\mathbf{w} - \Phi^T\mathbf{t} \\
&= 0
\end{aligned}
$$

$$w^T w = \sum_{j=1}^{M} w_j^2$$

$$\frac{1}{2}\frac{\partial}{\partial w_j} w^T w = w_j$$

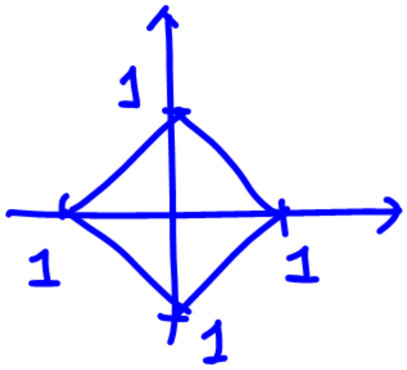
$$\frac{1}{2}\nabla_w \underset{w^T I w}{\underbrace{w^T w}} = Iw = w$$

Therefore, we get:

$$\mathbf{w}_{ML} = (\Phi^T\Phi + \lambda\mathbf{I})^{-1}\Phi^T\mathbf{t}$$

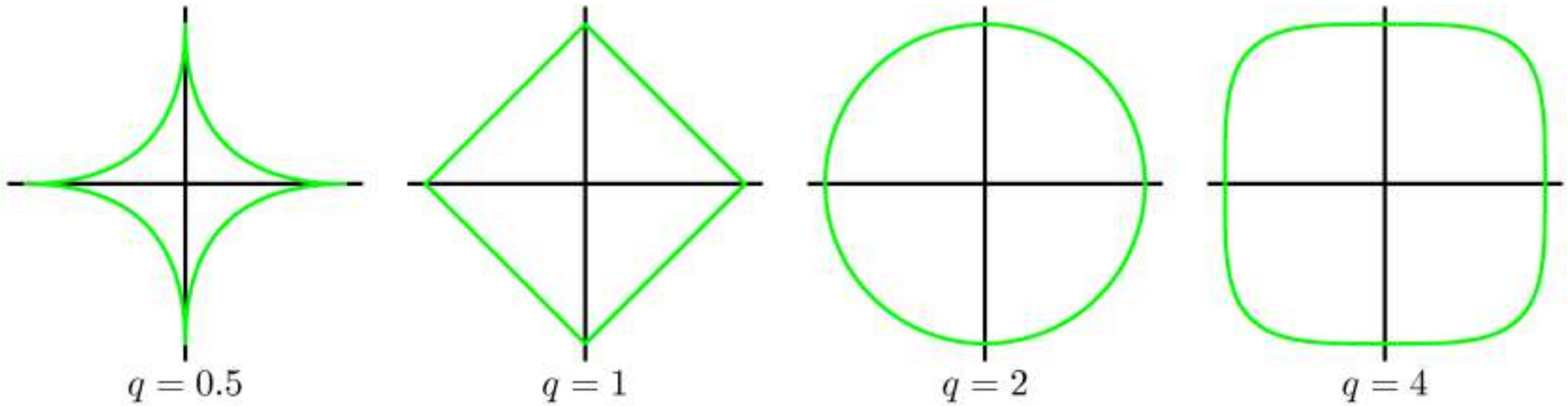
Regularized Least Squares (2)

$|w_1| + |w_2| = 1$



- With a more general regularizer, we have

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2 + \underline{\frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q}$$

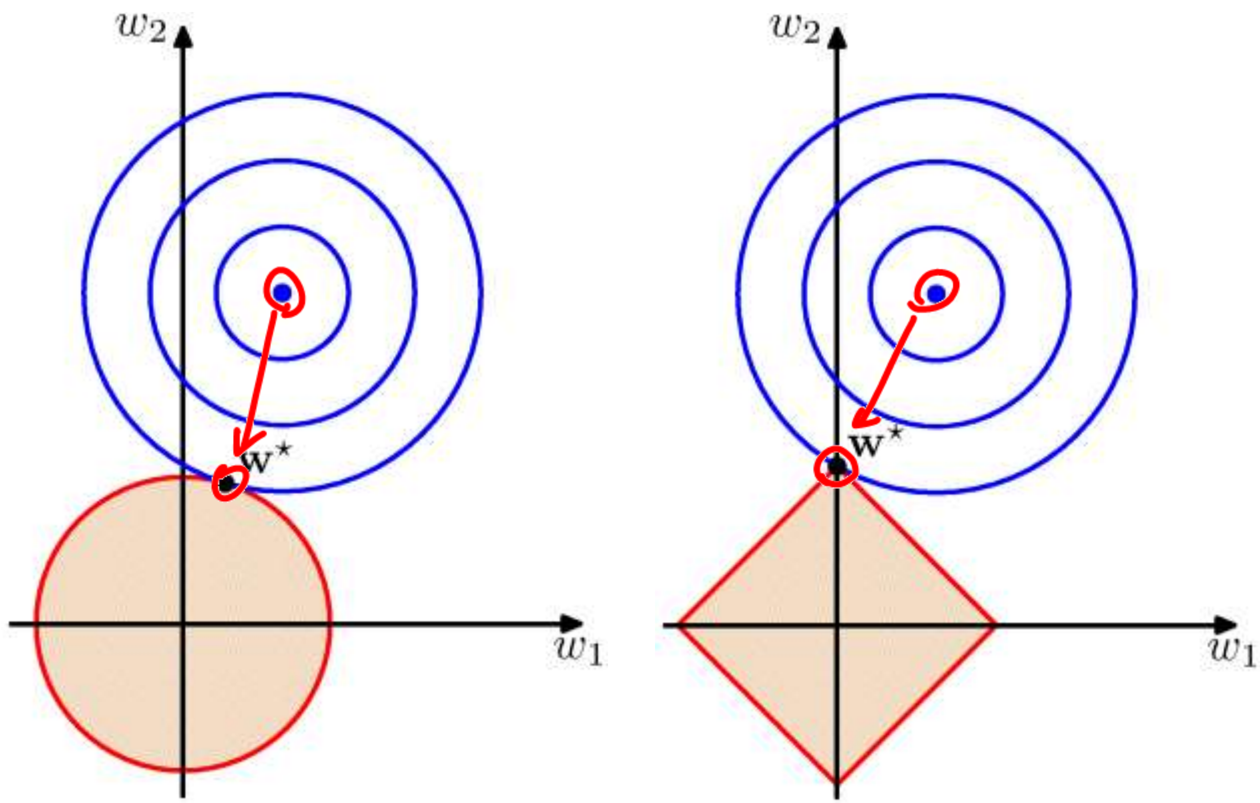


Lasso
"L1 regularization"

Quadratic
"L2 regularization"

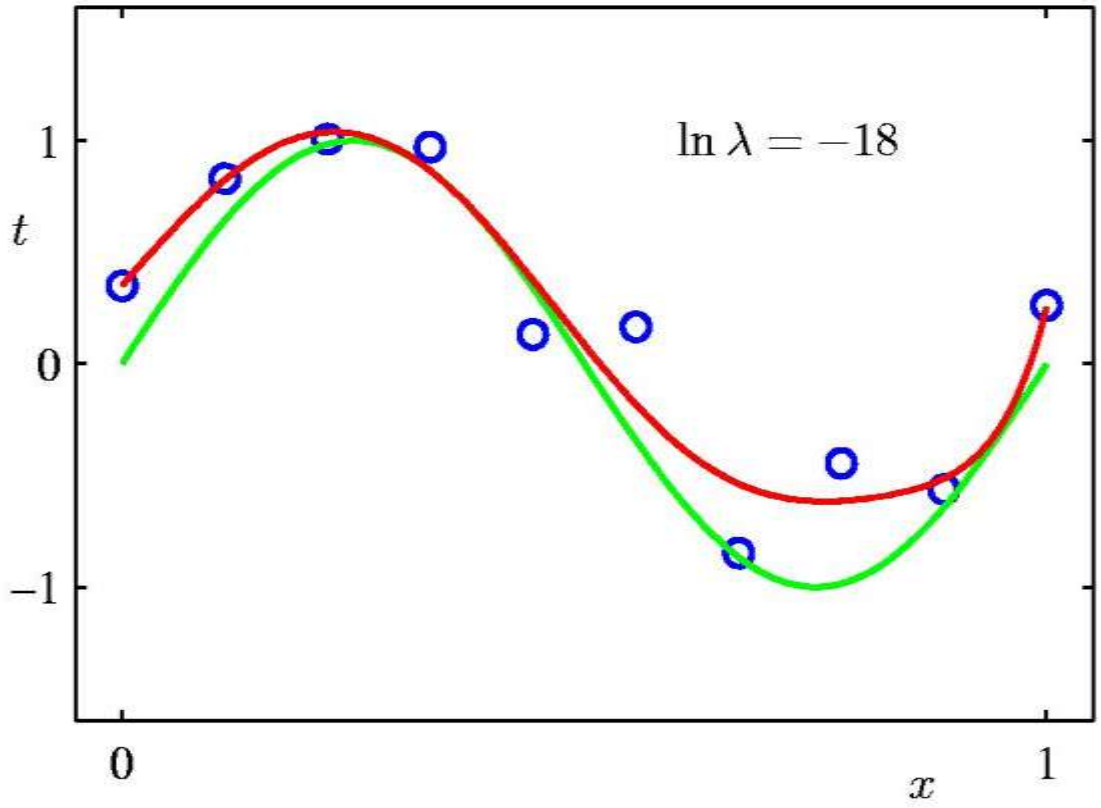
Regularized Least Squares (3)

- Lasso tends to generate sparser solutions than a quadratic regularizer.

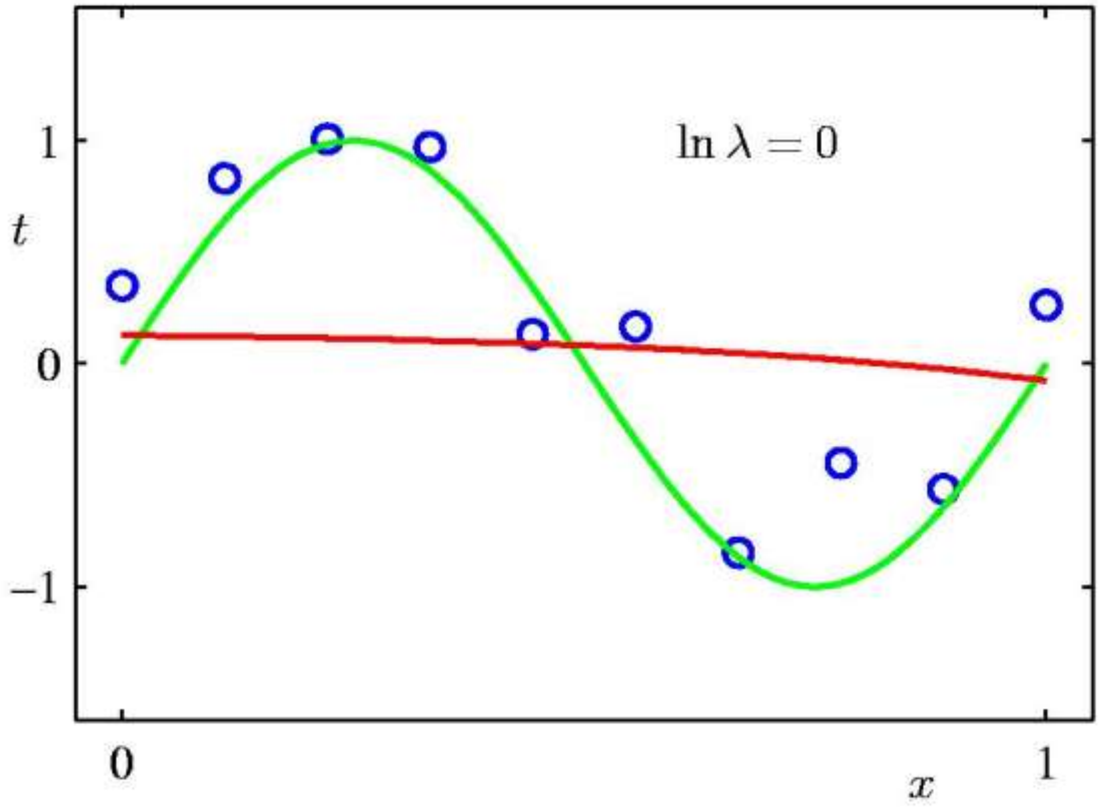


Regularization as constraints: $\sum_j w_j^2 \leq C$ $\sum_j |w_j| \leq C$

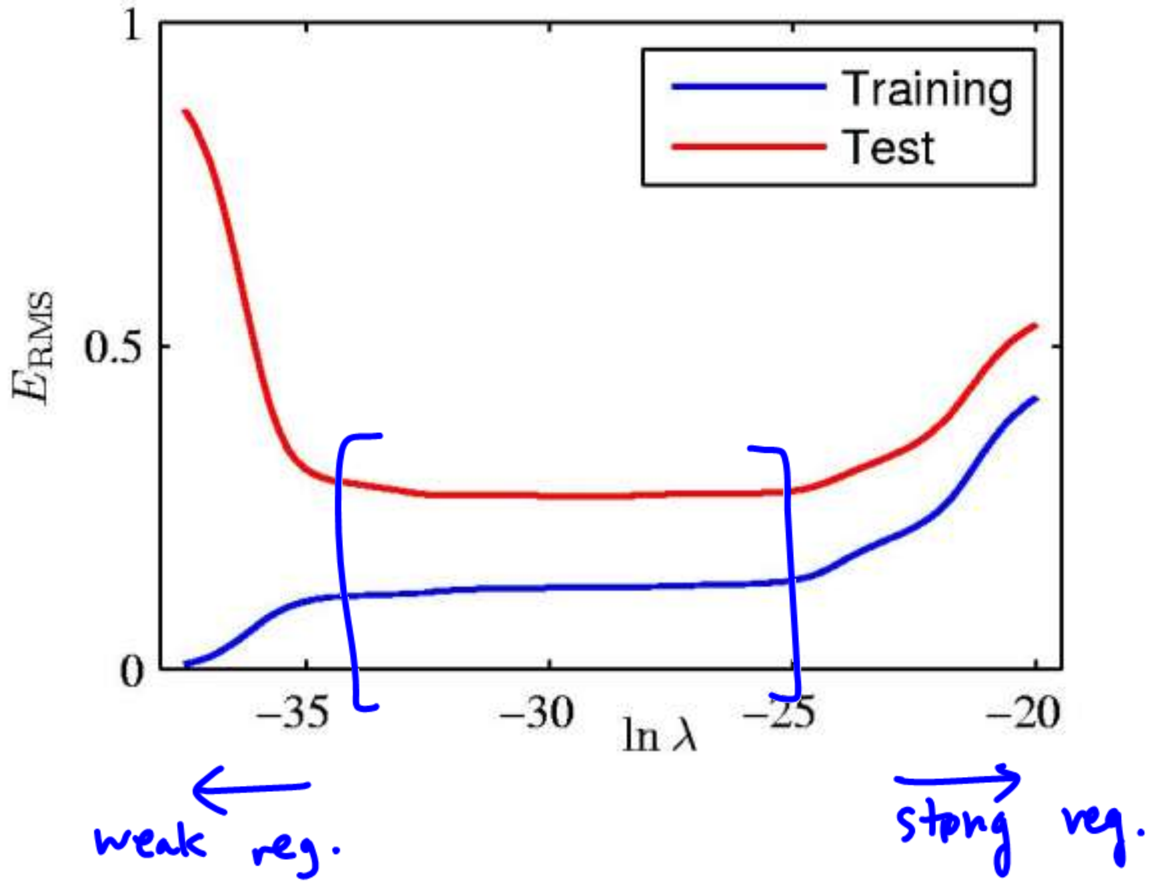
L2 Regularization: $\ln \lambda = -18$



L2 Regularization: $\ln \lambda = 0$

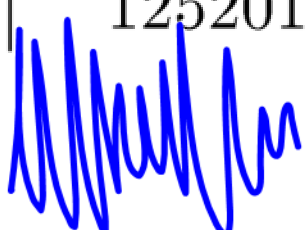


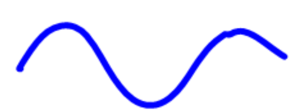
L2 Regularization: E_{RMS} vs. $\ln\lambda$



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





Locally-weighted Linear Regression (a.k.a. instance based regression)

Locally weighted linear regression

- Main idea: When predicting f(x), give high weights for “neighbors” of x.

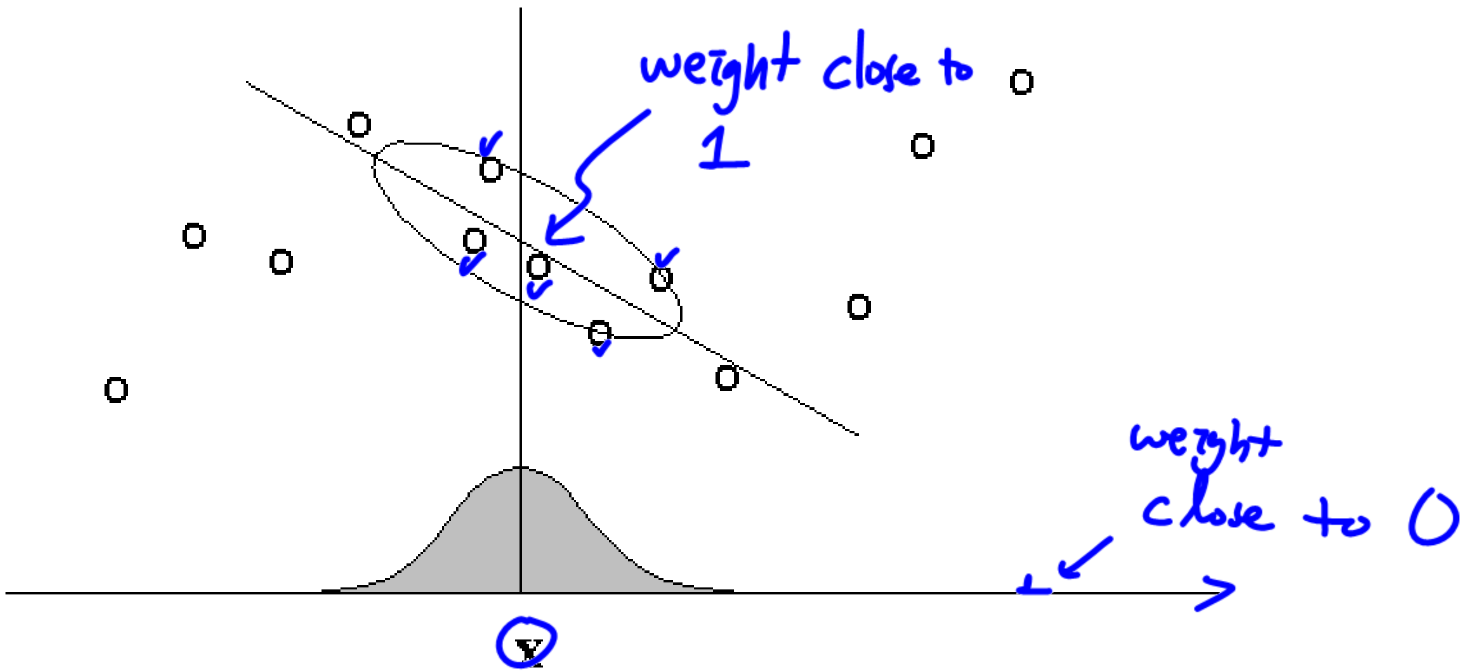


Figure 2: In locally weighted regression, points are weighted by proximity to the current **x** in question using a kernel. A regression is then computed using the weighted points.

Slide credit: William Cohen

Linear regression vs. Locally-weighted Linear Regression

- Linear regression

1. Fit θ to minimize $\sum_i (y^{(i)} - \theta^T x^{(i)})^2$.
2. Output $\theta^T x$.

- Locally-weighted linear regression

1. Fit θ to minimize $\sum_i w^{(i)} (y^{(i)} - \theta^T x^{(i)})^2$.
2. Output $\theta^T x$.

τ: "kernel width"

  - Standard choice: $w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$
  - The problem can be formulated as a modified version of least squares problem (Programming assignment in HW#1)

Locally weighted linear regression

- Choice of kernel width τ

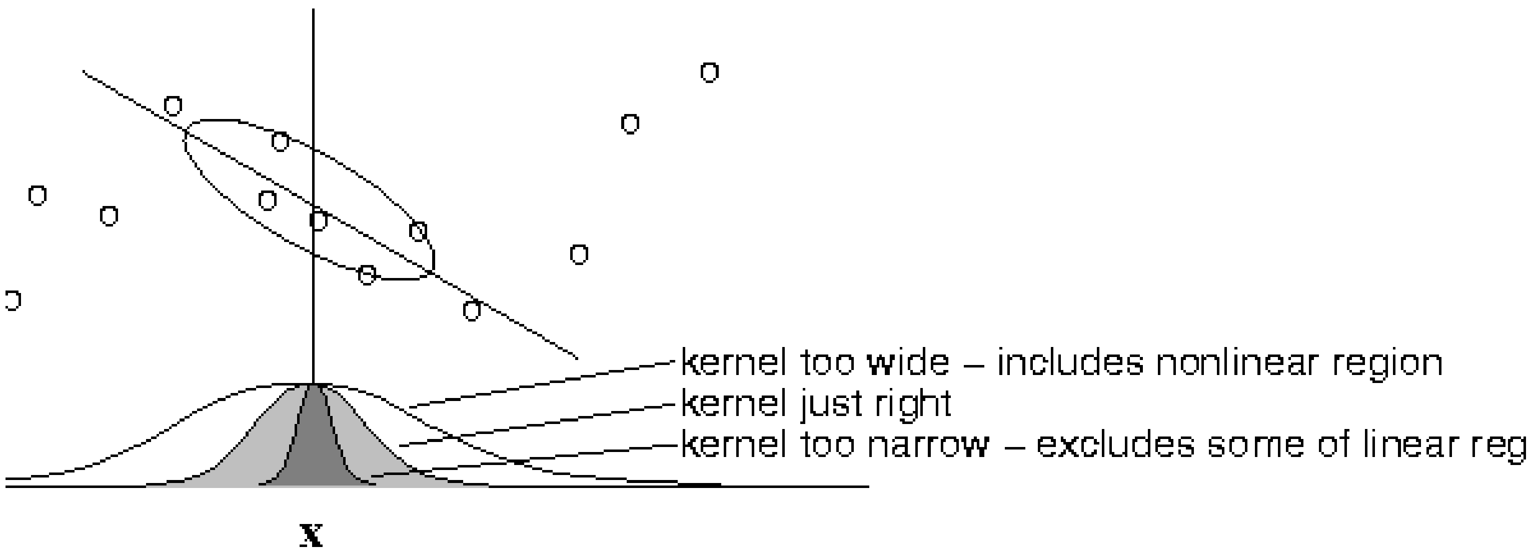


Figure 3: The estimator variance is minimized when the kernel includes as many training points as can be accommodated by the model. Here the linear LOESS model is shown. Too large a kernel includes points that degrade the fit; too small a kernel neglects points that increase confidence in the fit.

Slide credit: William Cohen

Classification: Kernel regression (a.k.a. instance based regression)

Locally-weighted Linear Regression vs. Kernel regression

- Locally-weighted linear regression

  1. Fit θ to minimize $\sum_i w^{(i)}(y^{(i)} - \theta^T x^{(i)})^2$.
  2. Output $\theta^T x$.

  τ: "kernel width"

  – Standard choice: $w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$

- Kernel regression (using Gaussian kernel)

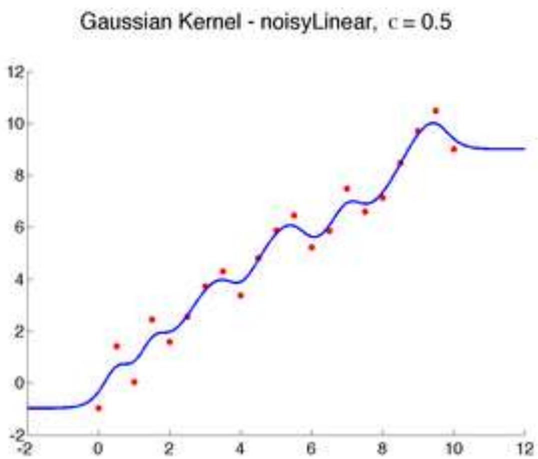
  – output: $\frac{\sum_i K(x,x^{(i)}) y^{(i)}}{\sum_i K(x,x^{(i)})}$

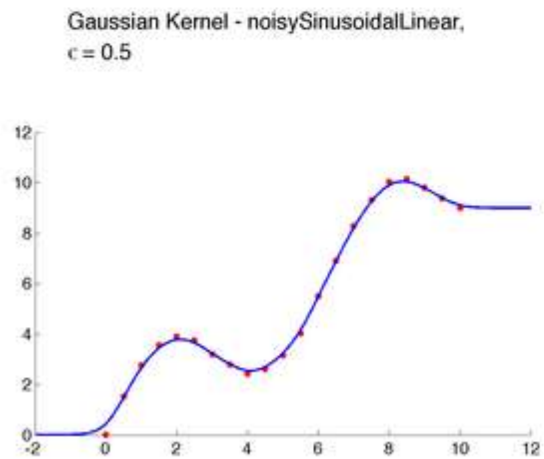
  where $K(x, x^{(i)})$ = $w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$

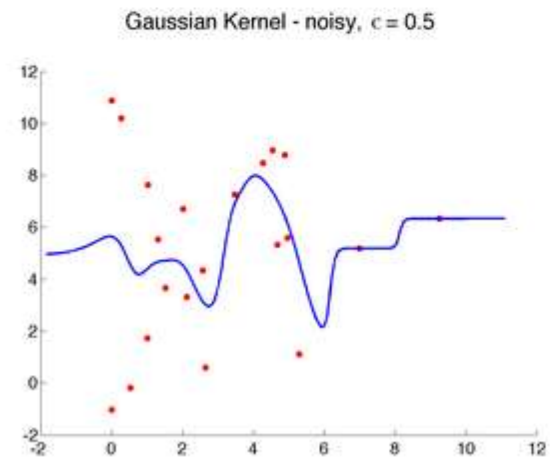
More generally, any distance metric (other than L2 or Eucleidian distance) can be used

Kernel regression

- Examples







Kernel regression: Classification vs Regression

- Note: it is very easy to formulate kernel regression into regression/classification

1. Given training data $D = \{\mathbf{x}_i, y_i\}$, Kernel function $K(\cdot,\cdot)$ and input $\mathbf{x}$
   - (regression) if $y \in \mathbf{R}$, return weighted average:

$$\frac{\sum_{i=1}^{n} K(\mathbf{x}, \mathbf{x}_i) y_i}{\sum_{i=1}^{n} K(\mathbf{x}, \mathbf{x}_i)}$$

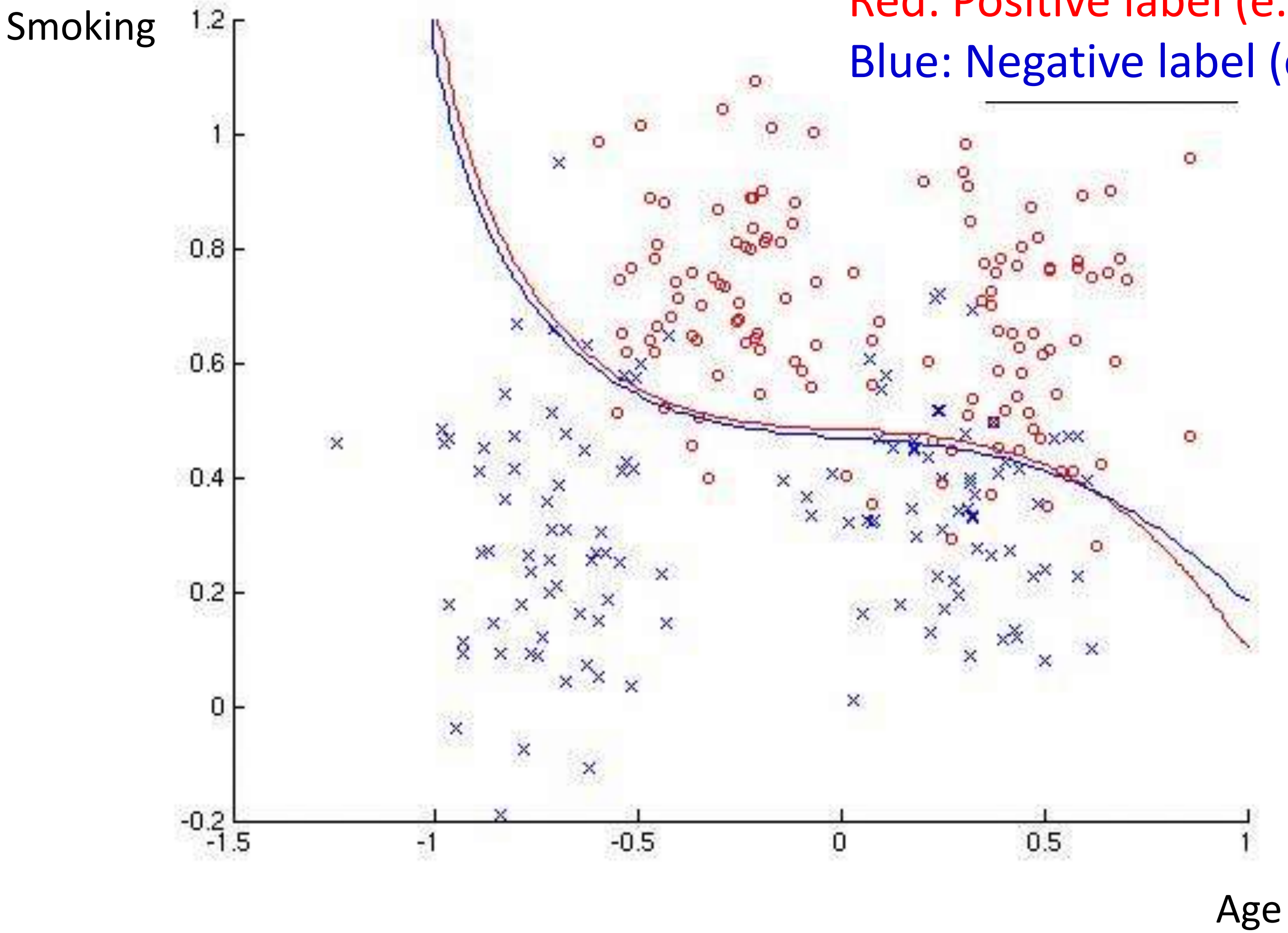
- (classification) if $y \in \pm 1$, return weighted majority:

$$sign(\sum_{i=1}^{n} K(\mathbf{x}, \mathbf{x}_i) y_i)$$

Supervised Learning: Classification

Supervised Learning - Classification

Red: Positive label (e.g., cancer)
Blue: Negative label (e.g., normal)



"Learning decision boundaries"

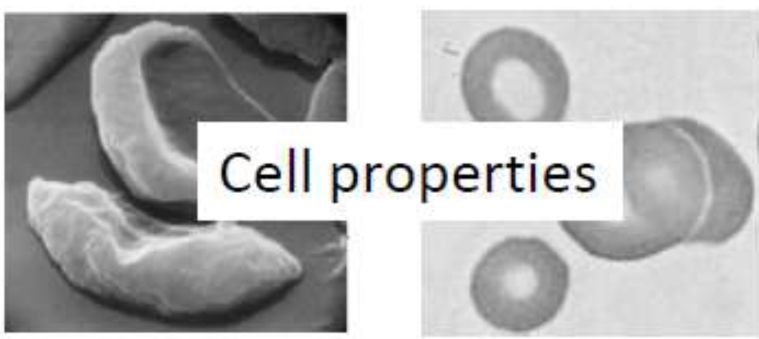
Supervised Learning - Classification

Feature Space $\mathcal{X}$

Label Space $\mathcal{Y}$

Words in a document

“Sports”
“News”
“Science”
…



Cell properties

“Anemic cell”
“Healthy cell”

Discrete Labels

Slide credit: Aarti Singh

Classification problem

- Training data $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$
  - For binary classification, $y_n \in \{0,1\}$ or $y_n \in \{-1,1\}$
  - Generally, $y_n \in \{1, \dots, C\}$, where C is the number of discrete labels
- Training: train a classifier h(x)
- Testing (evaluation):
  - The learning algorithm produces predictions $\mathrm{h}(\mathrm{x}_1^{\mathrm{test}}), \mathrm{h}(\mathrm{x}_2^{\mathrm{test}}), \dots, \mathrm{h}(\mathrm{x}_\mathrm{m}^{\mathrm{test}})$ for a set of testing data
  - 0-1 loss:

$$\text{error} = A = \frac{1}{m}\sum_{j=1}^{m} 1[h(\mathrm{x}_\mathrm{j}^{\mathrm{test}}) \neq \mathrm{y}_\mathrm{j}^{\mathrm{test}}]$$

Classification: K-nearest neighbor (a.k.a. instance based classification)

Basic k-nearest neighbor

- Training method:
  - Save the training examples
- At prediction time:
  - Find the k training examples $(x_1,y_1),\ldots(x_k,y_k)$ that are closest to the test example x
  - Predict the most frequent class among those y_i's.

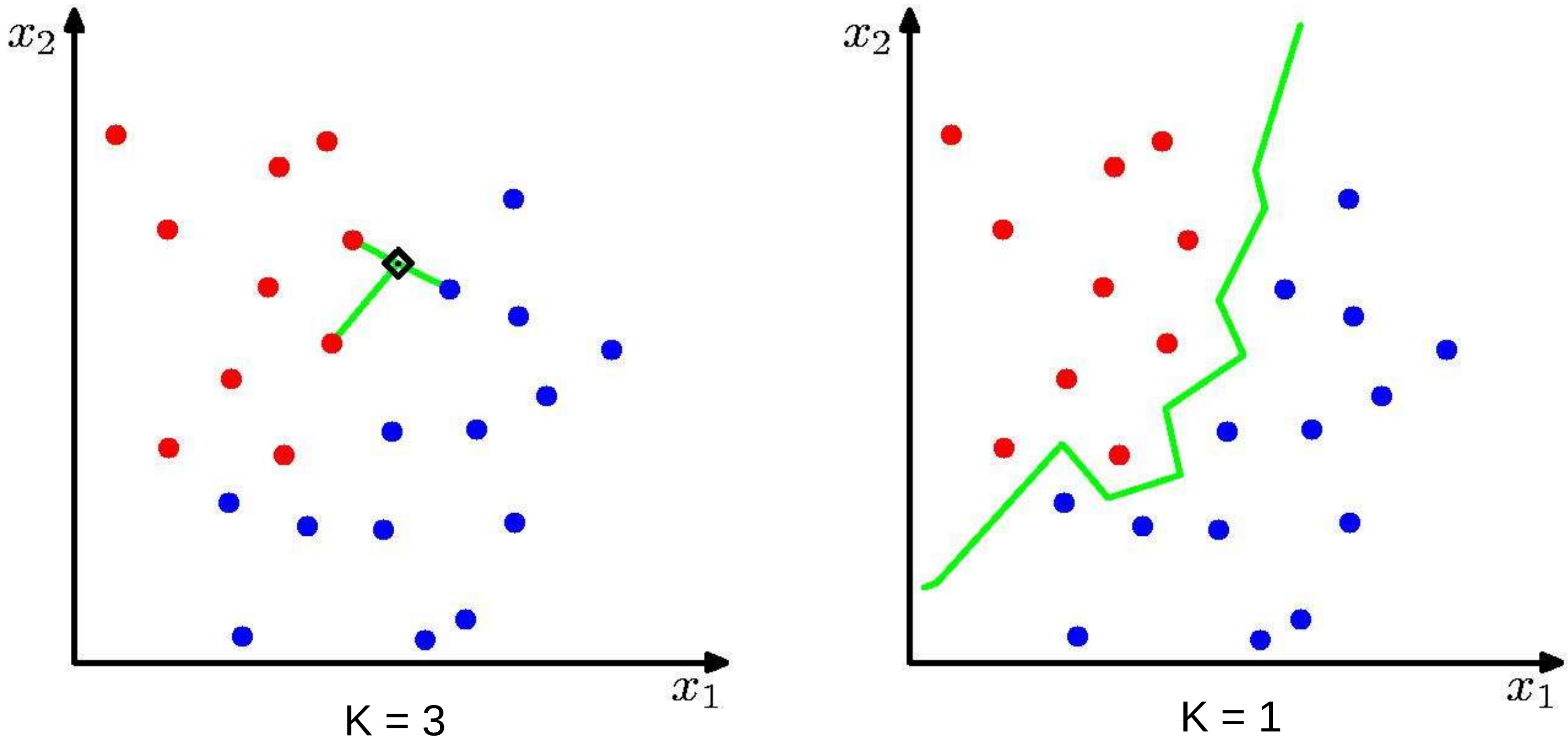
"majority vote"

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i, \qquad h(x) = sign(\hat{Y}(x))$$

- Note: this $\hat{Y}(x)$ function can be applied to regression!

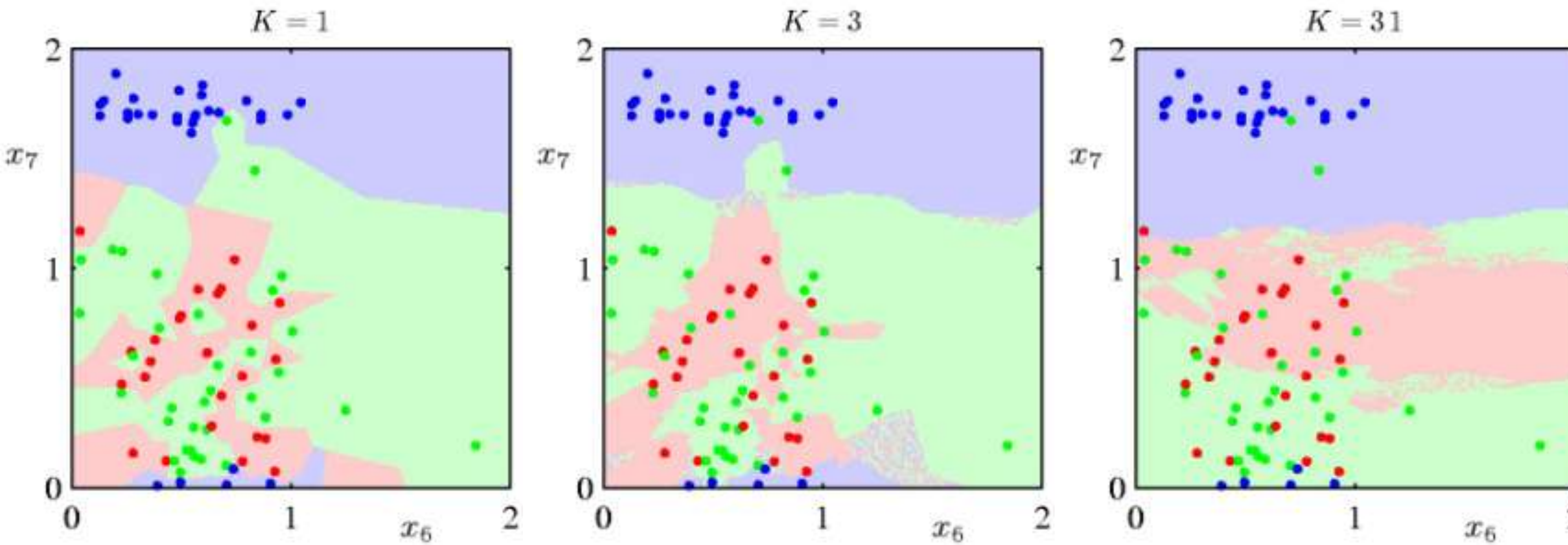
Slide credit: William Cohen

K-Nearest-Neighbours for Classification



Slide credit: Ben Kuipers

K-Nearest-Neighbours for Classification

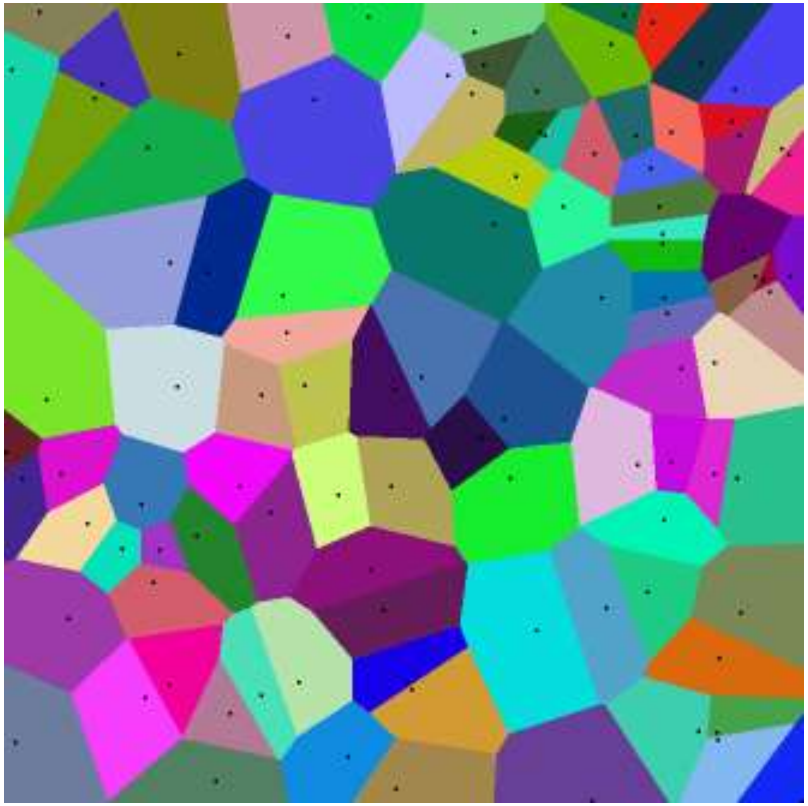


- K acts as a smother
- For $N \to \infty$, the error rate of the 1-nearest-neighbour classifier is never more than twice the optimal error (obtained from the true conditional class distributions).

Slide credit: Ben Kuipers

What is the decision boundary?

Voronoi diagram

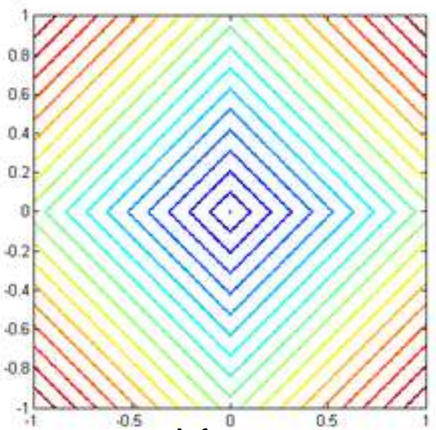


Slide credit: William Cohen

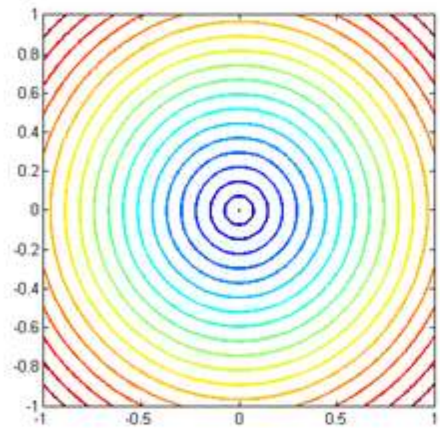
Dependence on distance metric (L^q norm)

Distance between i-th and j-th example: $\sqrt[q]{\sum_l \left(x_l^{(i)} - x_l^{(j)}\right)^q}$

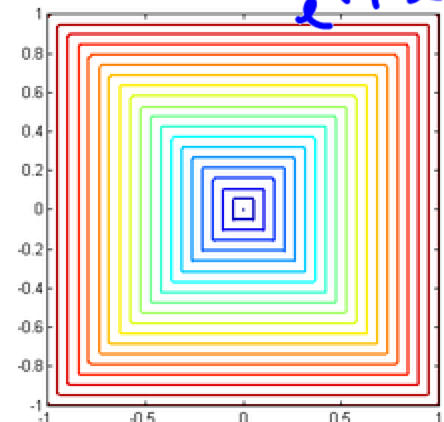
$\max_l(|x_l|)$



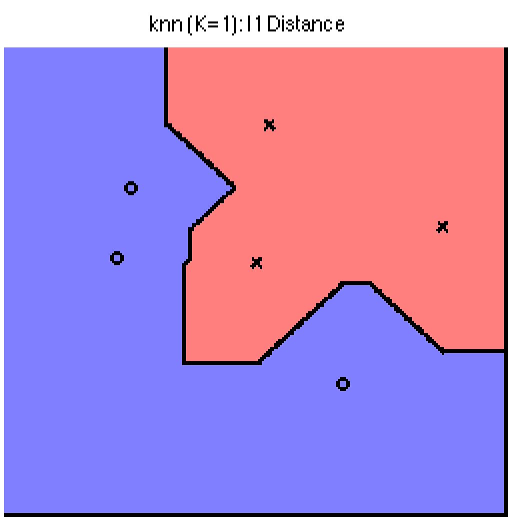
L1 norm

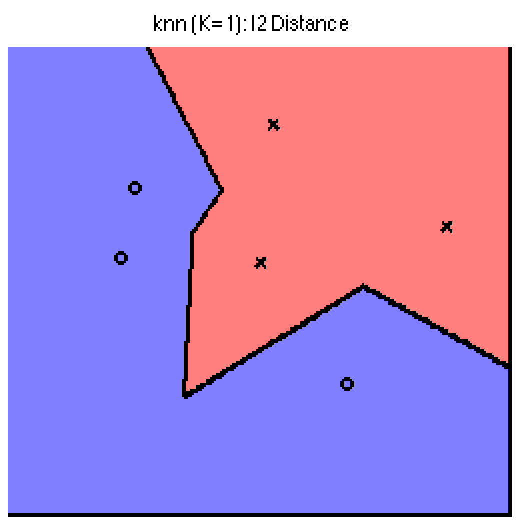


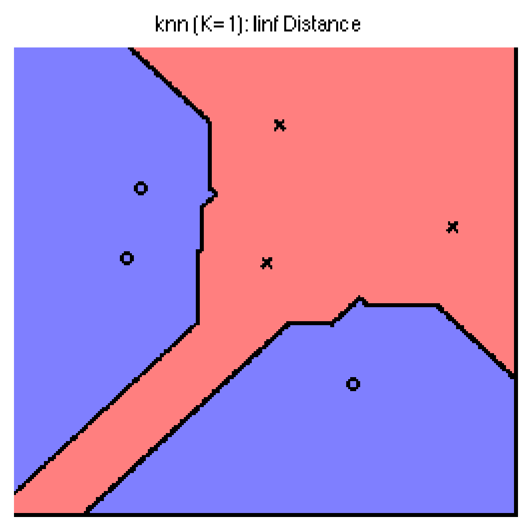
L2 norm



L_∞ norm







Slide credit: Ben Taskar

Advantage/disadvantages of instance-based (local) learning algorithms

- Advantage:
  – very flexible, simple, and effective
- Disadvantages:
  – Expensive: need to remember (store) and search through all the training data for every prediction
  – Curse-of-dimensionality: In high dimensions, all points are far
  – Irrelevant features: If x has irrelevant, noisy features, distance function becomes useless