Machine Learning
ITCS 5356

Polynomial Curve Fitting
Regularization

Razvan C. Bunescu

Department of Computer Science @ CCI

rbunescu@uncc.edu

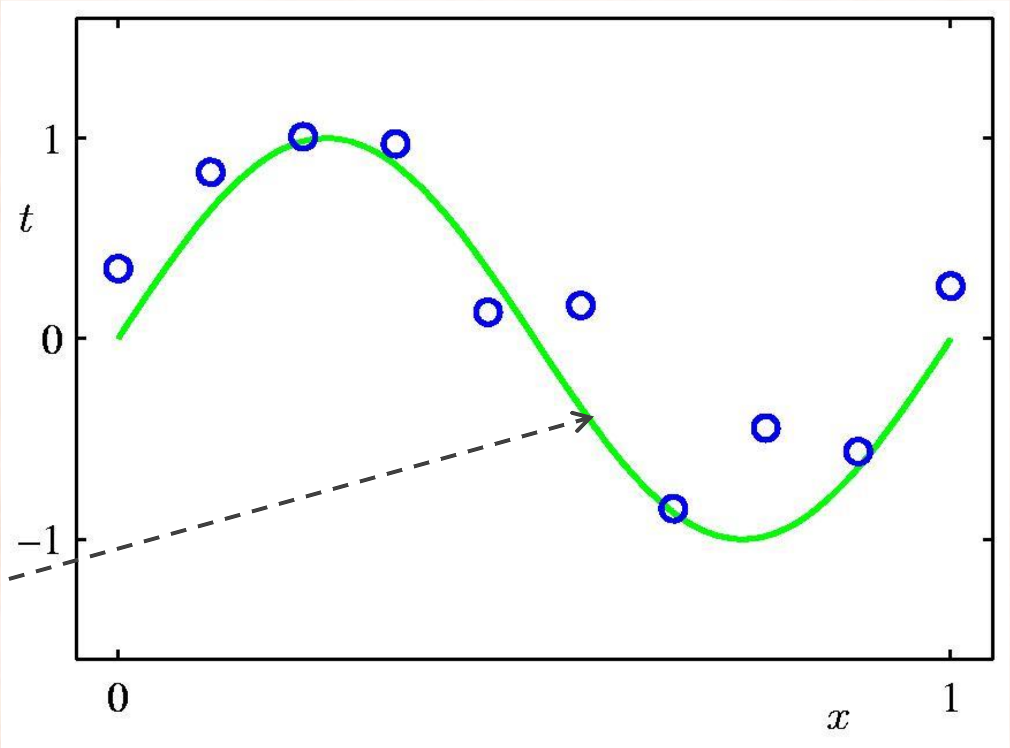
Simple Linear Regression

- Use a linear function approximation:
  - $\hat{y} = \mathbf{w}^{\mathrm{T}}\mathbf{x} = [w_0, w_1]^{\mathrm{T}}[1, x] = w_1 x + w_0$.
    - w_0 is the intercept (or the bias term).
    - w_1 controls the slope.
  - Learning = optimization:
    - Find $\mathbf{w}$ that obtains the best fit on the training data, i.e. find $\mathbf{w}$ that minimizes the **sum of square errors**:

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\left(\mathbf{w}^T\mathbf{x}^{(n)} - y_n\right)^2$$

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}}\, J(\mathbf{w})$$

Regression: Curve Fitting

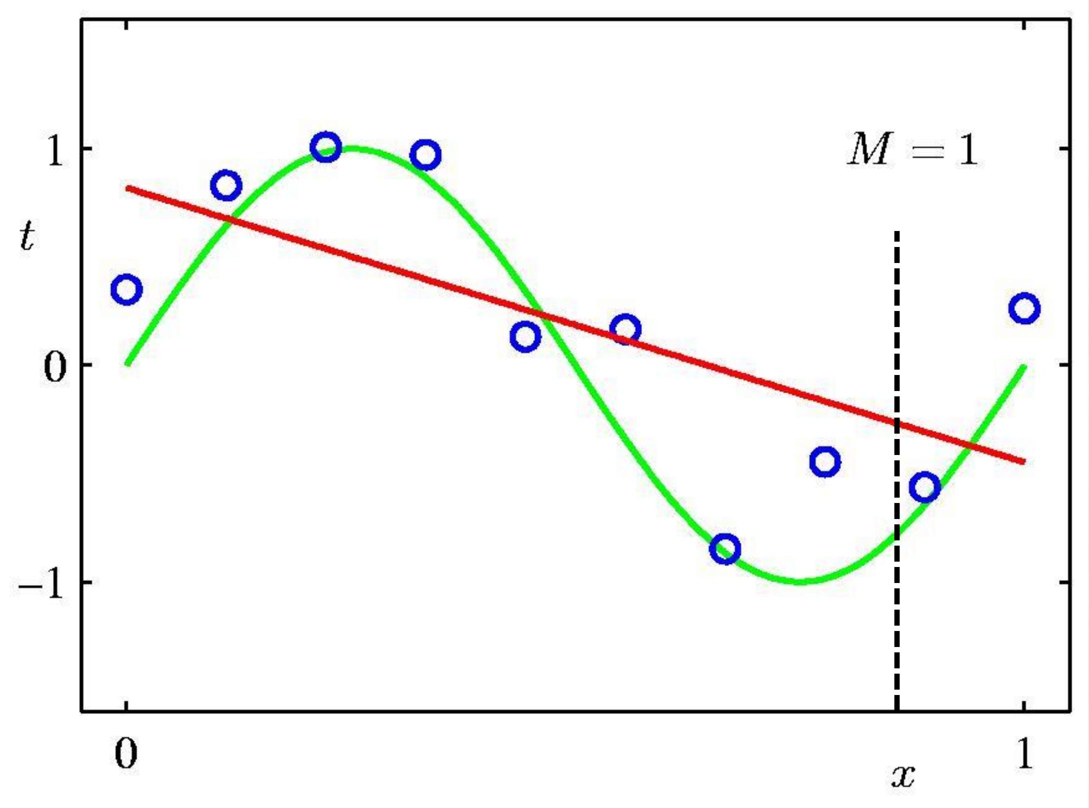


target t

- **Training**: Build a function $h(x)$, based on (noisy) training examples (x_1, y_1), (x_2, y_2), … (x_N, y_N)

What if the raw feature is insufficient?

- Simple linear regression = curve fitting with a 1-degree polynomial.

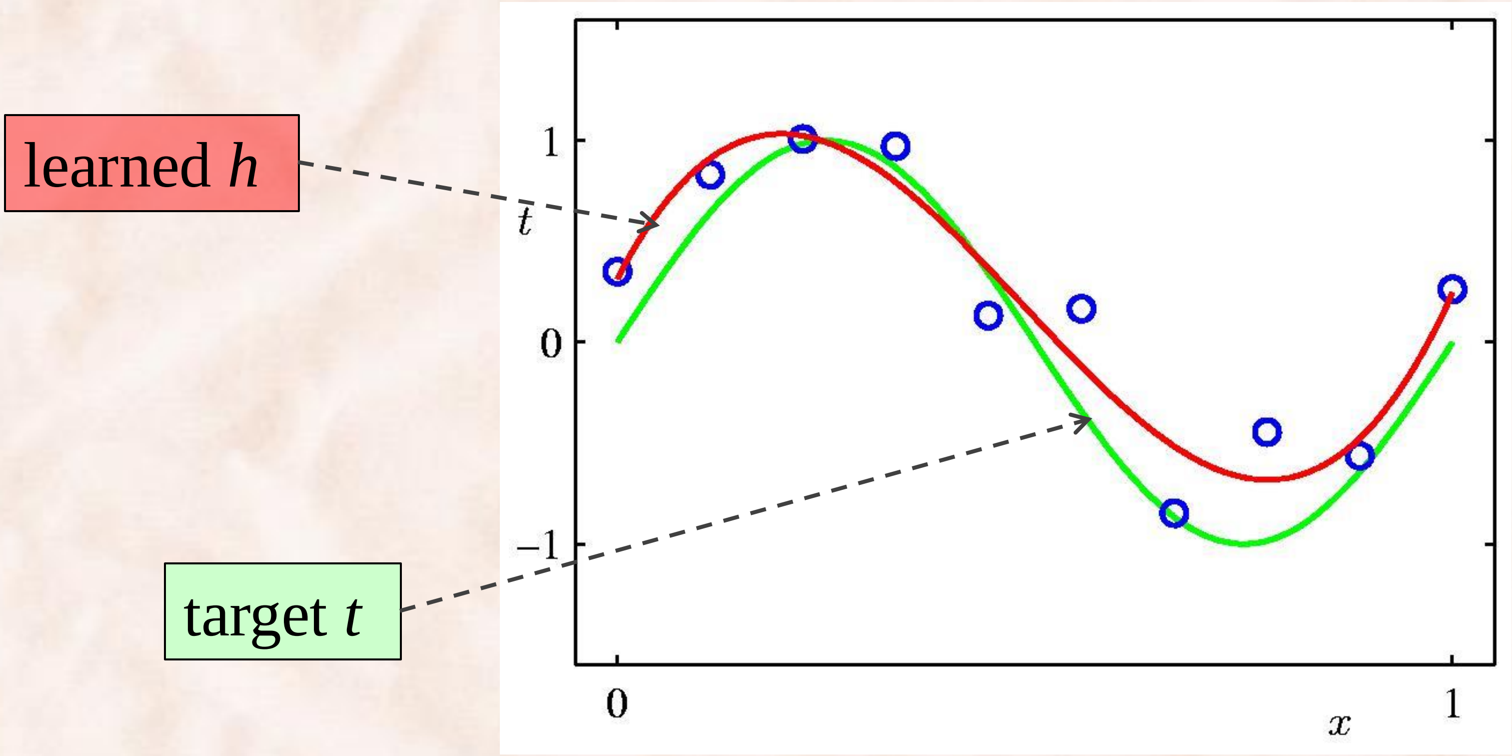


Polynomial Curve Fitting

- Generalize curve fitting, from a 1-degree to an M-degree polynomial.
  - Add new features, as polynomials of the original feature.

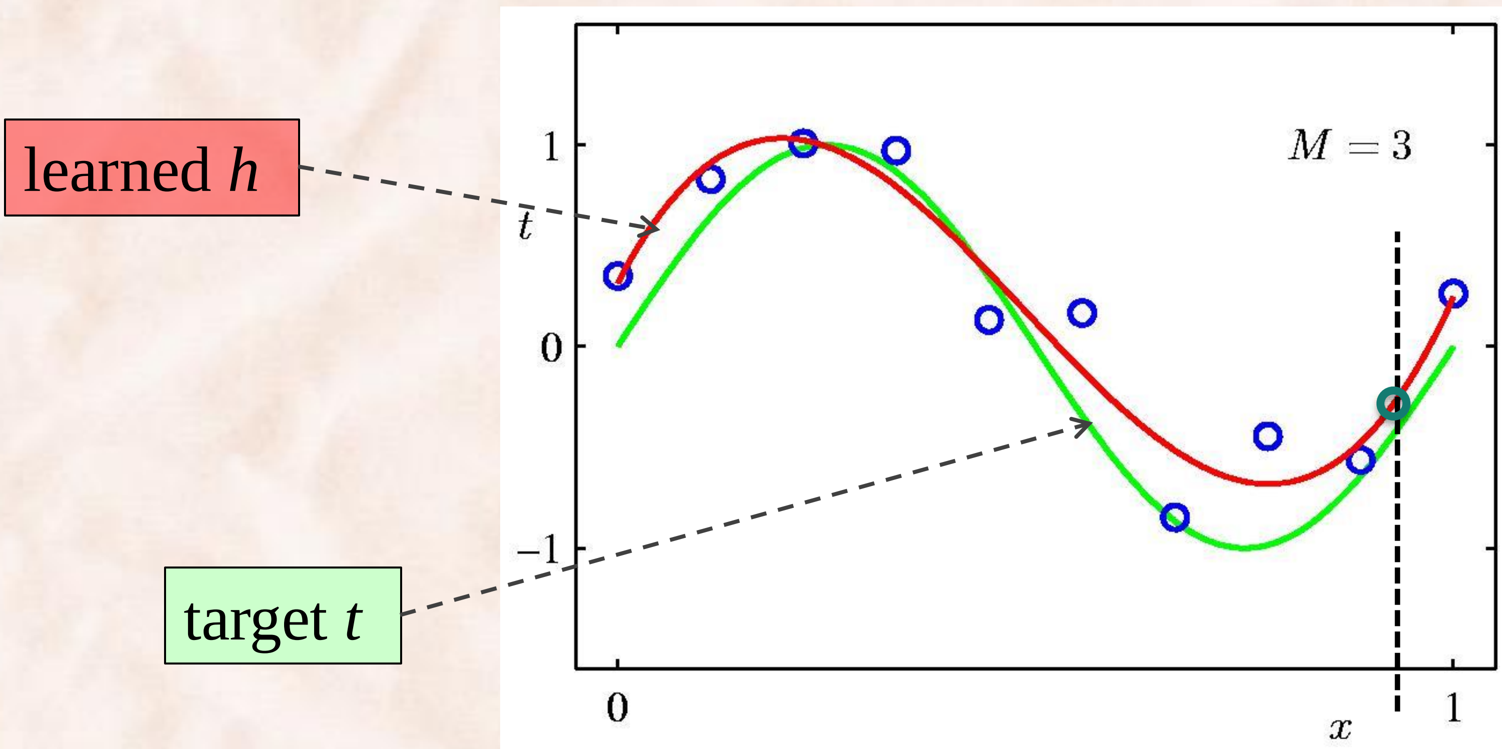
$$\hat{y} = h(x) = h(x, \underbrace{\mathbf{w}}_{\text{parameters}}) = w_0 + \underbrace{w_1 x + w_2 x^2 + \ldots + w_M x^M}_{\text{features}} = \sum_{j=0}^{M} w_j x^j$$

Regression: Curve Fitting



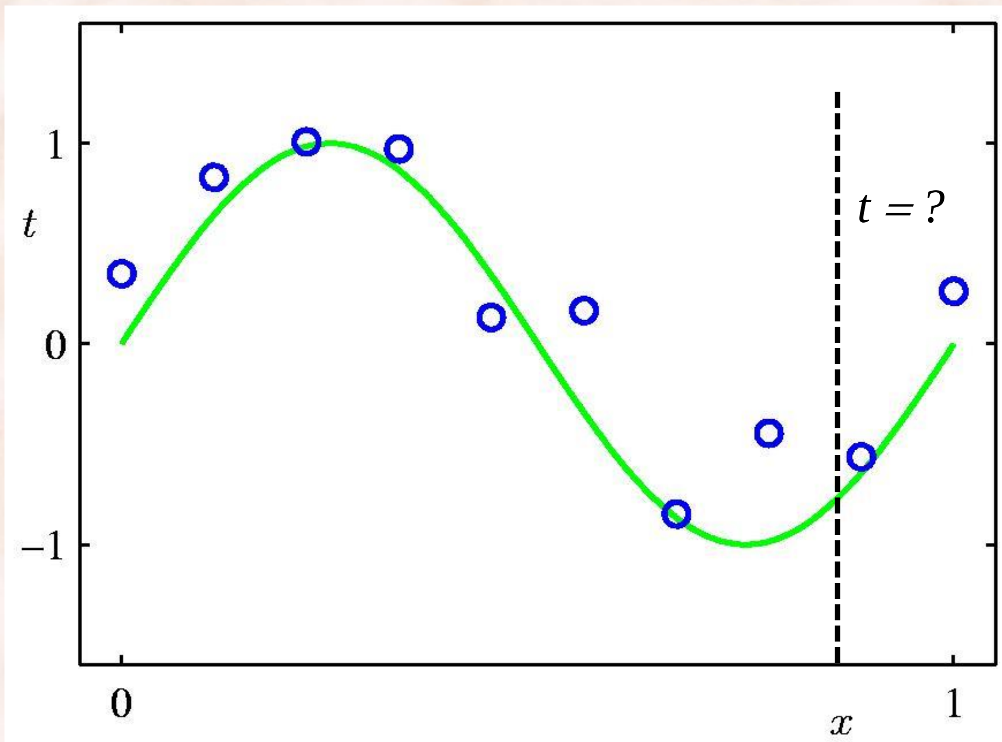
- **Training**: Build a function $h(x)$, based on (noisy) training examples (x_1,y_1), (x_2,y_2), … (x_N,y_N)

Regression: Curve Fitting



- **Testing**: for arbitrary (unseen) instance $x \in X$, compute target output $h(x)$; want it to be close to $y(x)$.

Regression: Polynomial Curve Fitting



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

parameters

features

Polynomial Curve Fitting

- Parametric model:

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

- Polynomial curve fitting is (Multiple) Linear Regression:

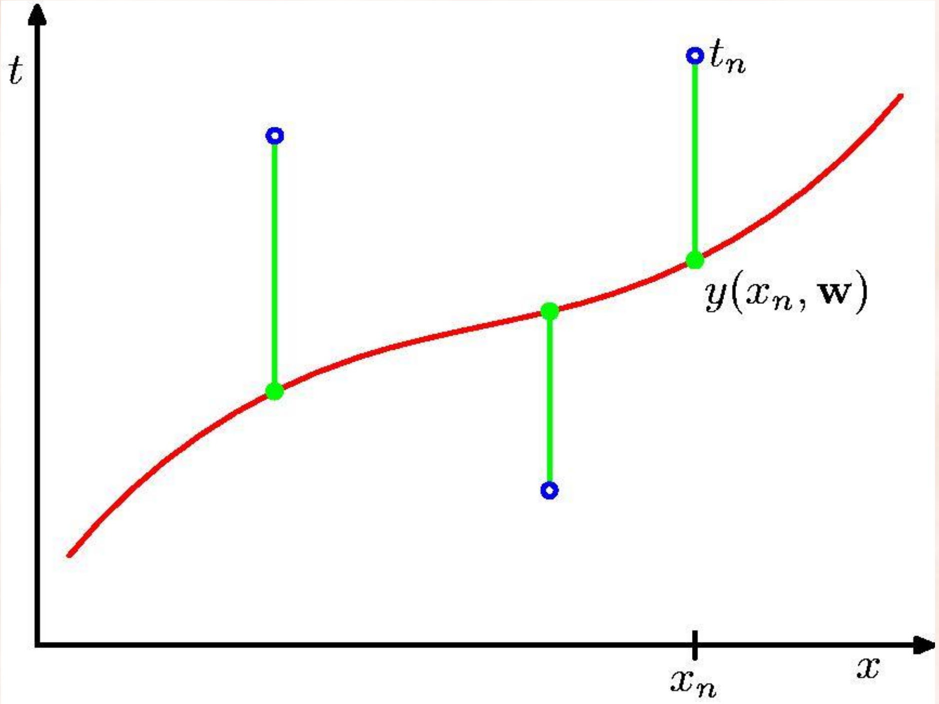
  $\mathbf{x} = [1, x, x^2, \ldots, x^M]^T$

  $\hat{y} = \mathbf{w}^T \mathbf{x}$

- **Learning** = minimize the Sum-of-Squares error function:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} J(\mathbf{w}) \qquad J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} \left(\mathbf{w}^T \mathbf{x}^{(n)} - y_n\right)^2$$

Sum-of-Squares Error Function



$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\left(\mathbf{w}^T\mathbf{x}^{(n)} - y_n\right)^2$$

- How to find $\hat{\mathbf{w}}$ that minimizes $J(\mathbf{w})$, i.e. $\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} J(\mathbf{w})$
- Solve $\nabla J(\mathbf{w}) = 0$.

Polynomial Curve Fitting

- *Least Square* solution is found by solving a set of M + 1 linear equations:

$$\mathrm{A}\mathbf{w} = \mathbf{b}$$

$$\sum_{j=0}^{M} a_{ij} w_j = b_i \quad \text{where} \quad a_{ij} = \sum_{n=1}^{N} x_n^{i+j} \qquad b_i = \sum_{n=1}^{N} y_n x_n^i$$

- Homework: Prove it.

Normal Equations

- Solution is $\mathbf{w} = \left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}}\mathbf{y}$

- X is the data matrix, or the **design matrix**:

$$X = \begin{pmatrix} \mathbf{x}^{(1)^{\mathrm{T}}} \\ \mathbf{x}^{(2)^{\mathrm{T}}} \\ \dots \\ \dots \\ \mathbf{x}^{(N)^{\mathrm{T}}} \end{pmatrix} = \begin{pmatrix} x_0^{(1)} & x_1^{(1)} & \dots & x_M^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \dots & x_M^{(2)} \\ & & \dots & \\ & & \dots & \\ x_0^{(N)} & x_1^{(N)} & \dots & x_M^{(N)} \end{pmatrix}$$

For poly fit:

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^M \\ 1 & x_2 & x_2^2 & \dots & x_2^M \\ & & \dots & & \\ & & \dots & & \\ 1 & x_N & x_N^2 & \dots & x_N^M \end{pmatrix}$$

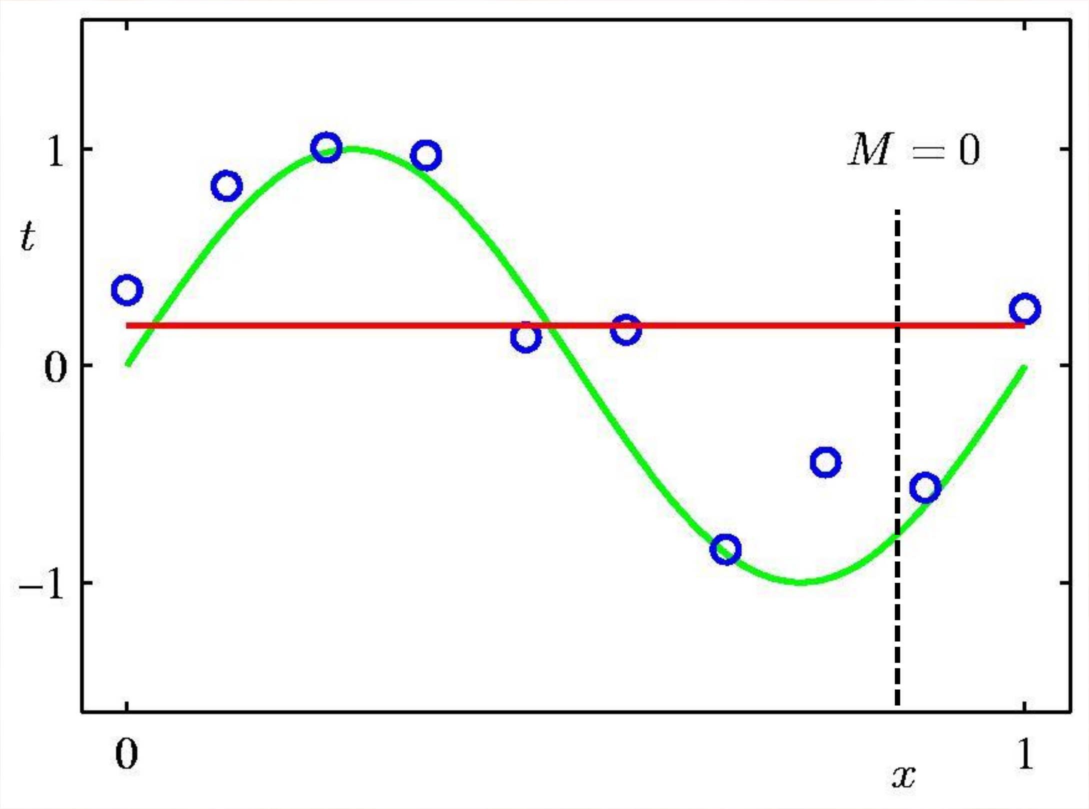
- $\mathbf{y} = [y_1, y_2, \dots, y_{\mathrm{N}}]^{\mathrm{T}}$ is the vector of labels.

Polynomial Curve Fitting

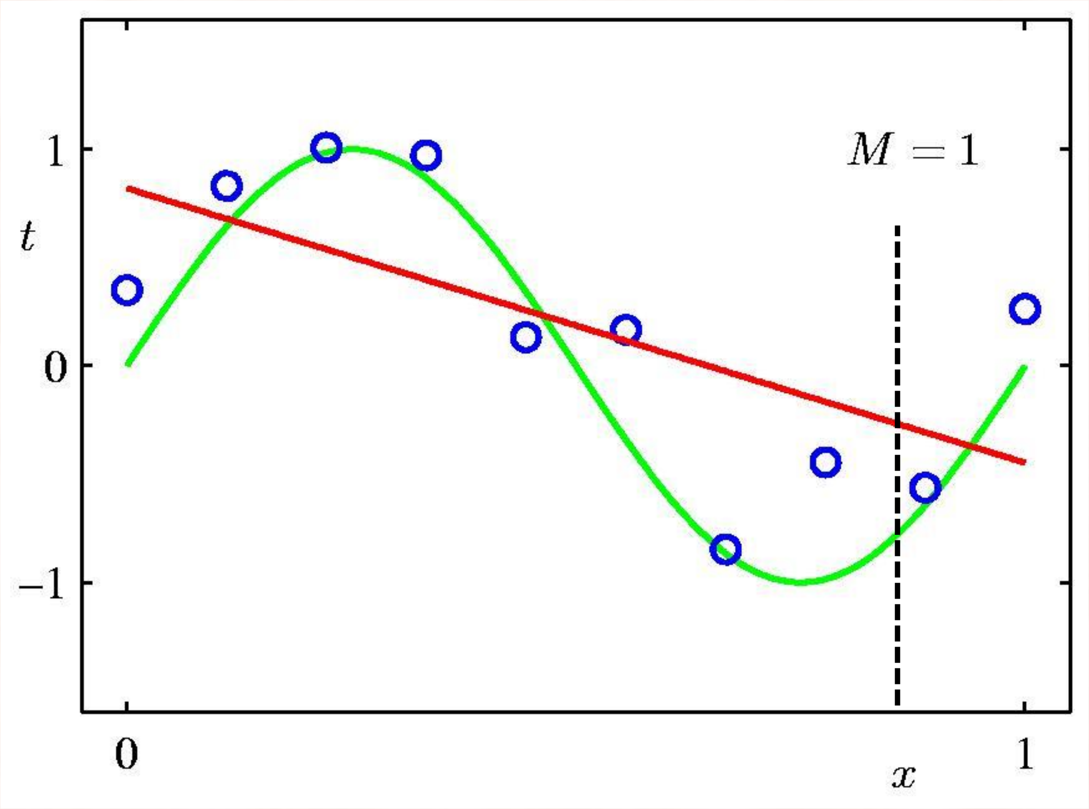
- Generalization = how well the parameterized $h(x,\mathbf{w})$ performs on arbitrary (unseen) test instances $x \in X$.

- Generalization performance depends on the value of M.

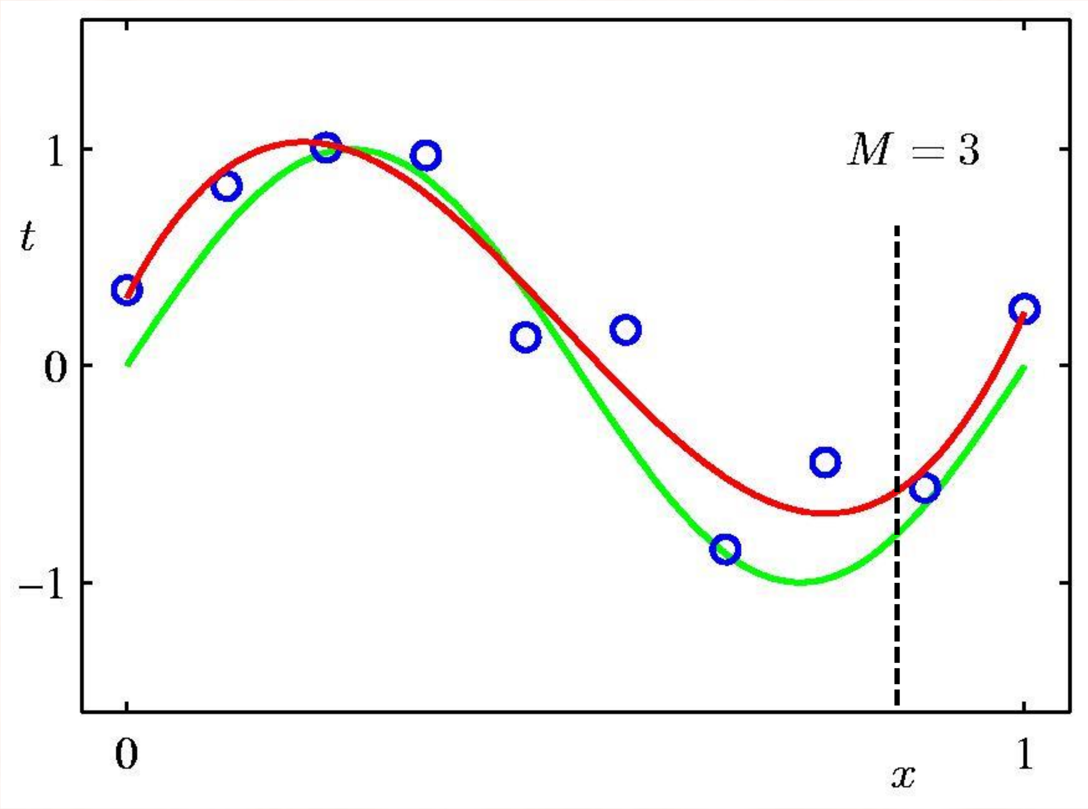
0^{th} Order Polynomial



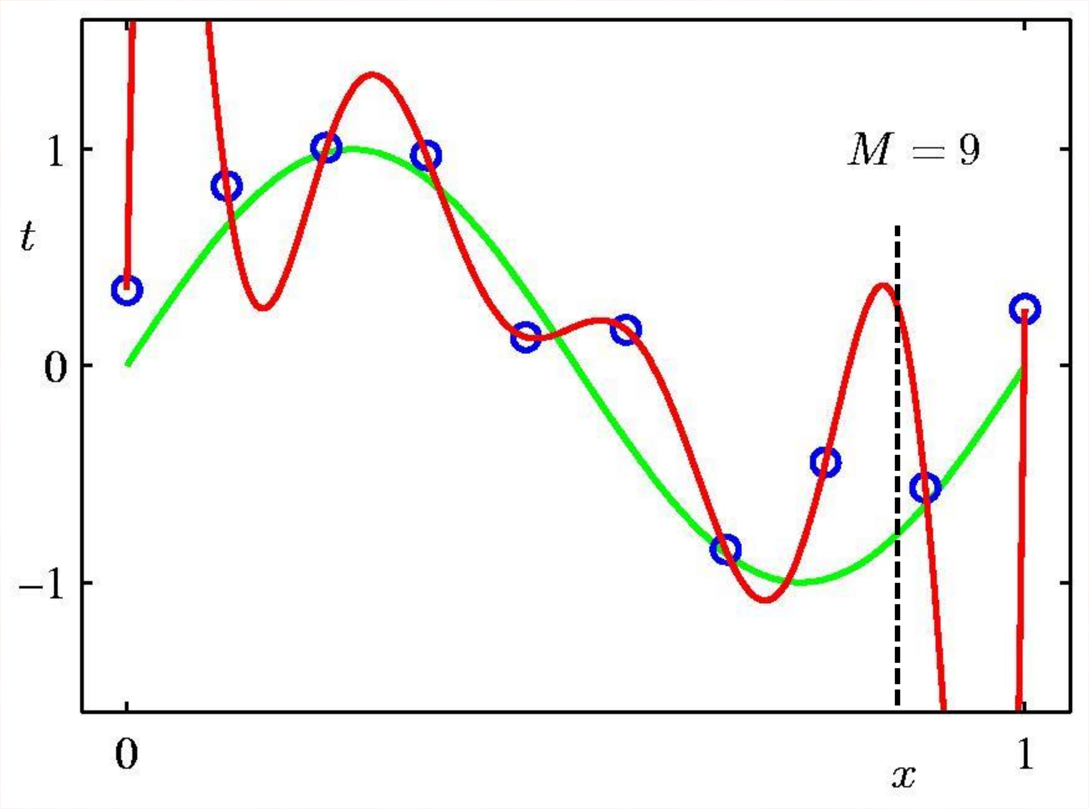
1^{st} Order Polynomial



3rd Order Polynomial



9th Order Polynomial



Polynomial Curve Fitting

- Model Selection: choosing the order M of the polynomial.
  - Best generalization obtained with $M = 3$.
  - $M = 9$ obtains poor generalization, even though it fits training examples perfectly:
    - But $M = 9$ polynomials subsume $M = 3$ polynomials!

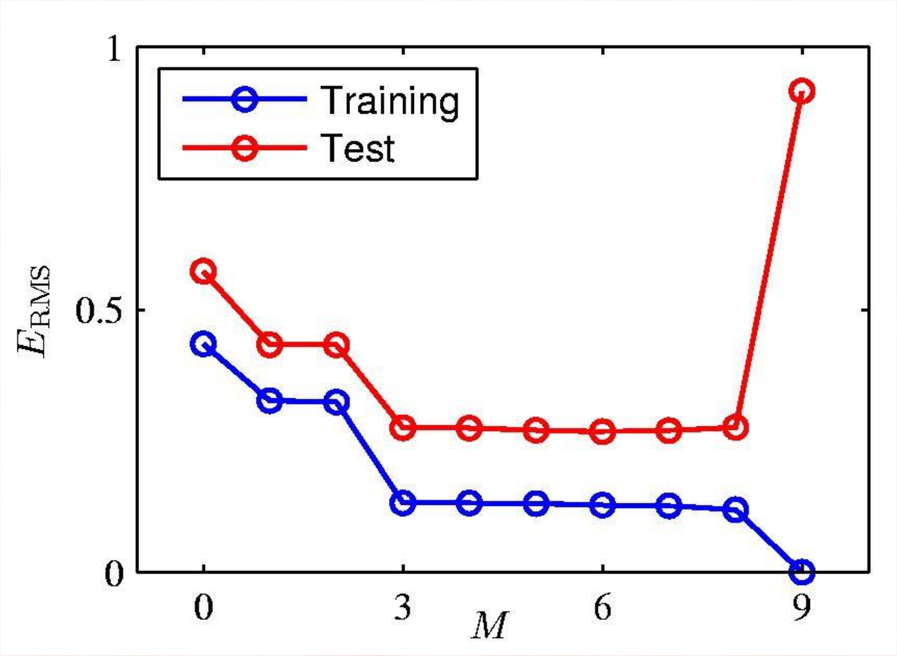
- Overfitting $\equiv$ good performance on training examples, poor performance on test examples.

Overfitting

- Measure fit using the Root-Mean-Square (RMS) error (RMSE):

$$E_{RMS}(\mathbf{w}) = \sqrt{\frac{\sum_n \left(\mathbf{w}^T \mathbf{x}_n - t_n\right)^2}{N}}$$

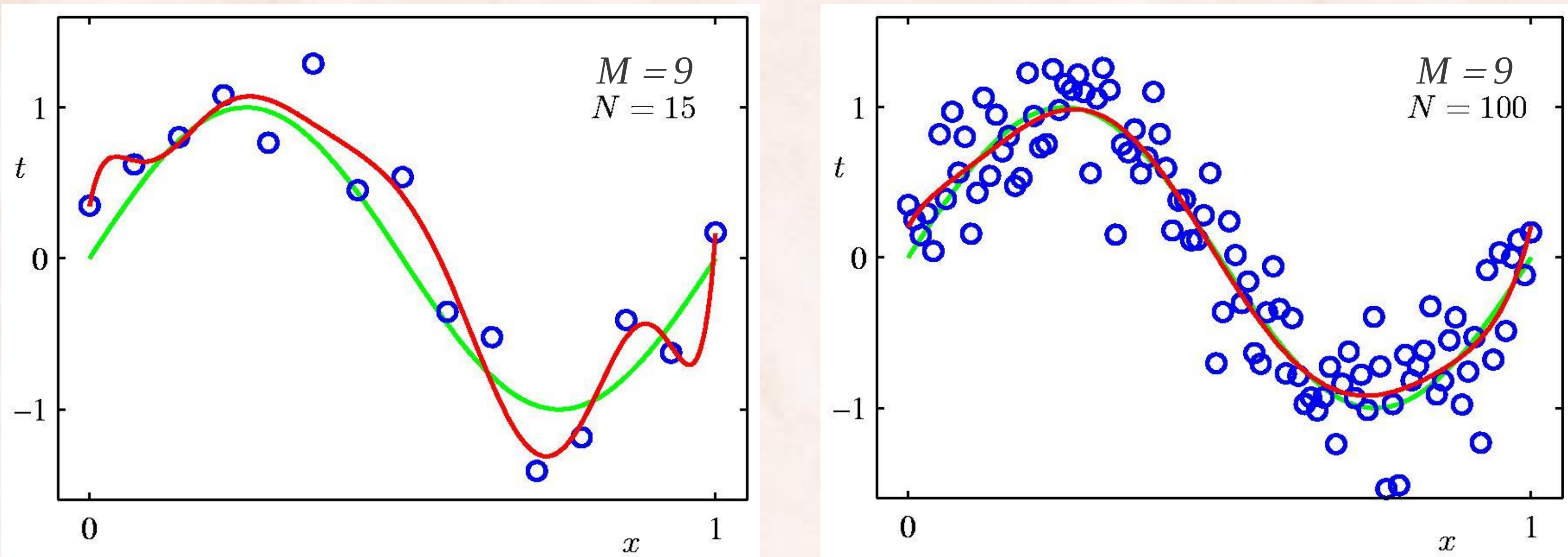
- Use 100 random test examples, generated in the same way:



Over-fitting and Parameter Values

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

Overfitting vs. Data Set Size



- More training data ⇒ less overfitting.
- What if we do not have more training data?
  - Use **regularization**.

Regularization

- **Parameter norm penalties** (term in the objective).
- Limit parameter norm (constraint).
- Dataset augmentation.
- Dropout.
- Ensembles.
- Semi-supervised learning.
- Early stopping.
- Noise robustness.
- Sparse representations.
- Adversarial training.

Regularization

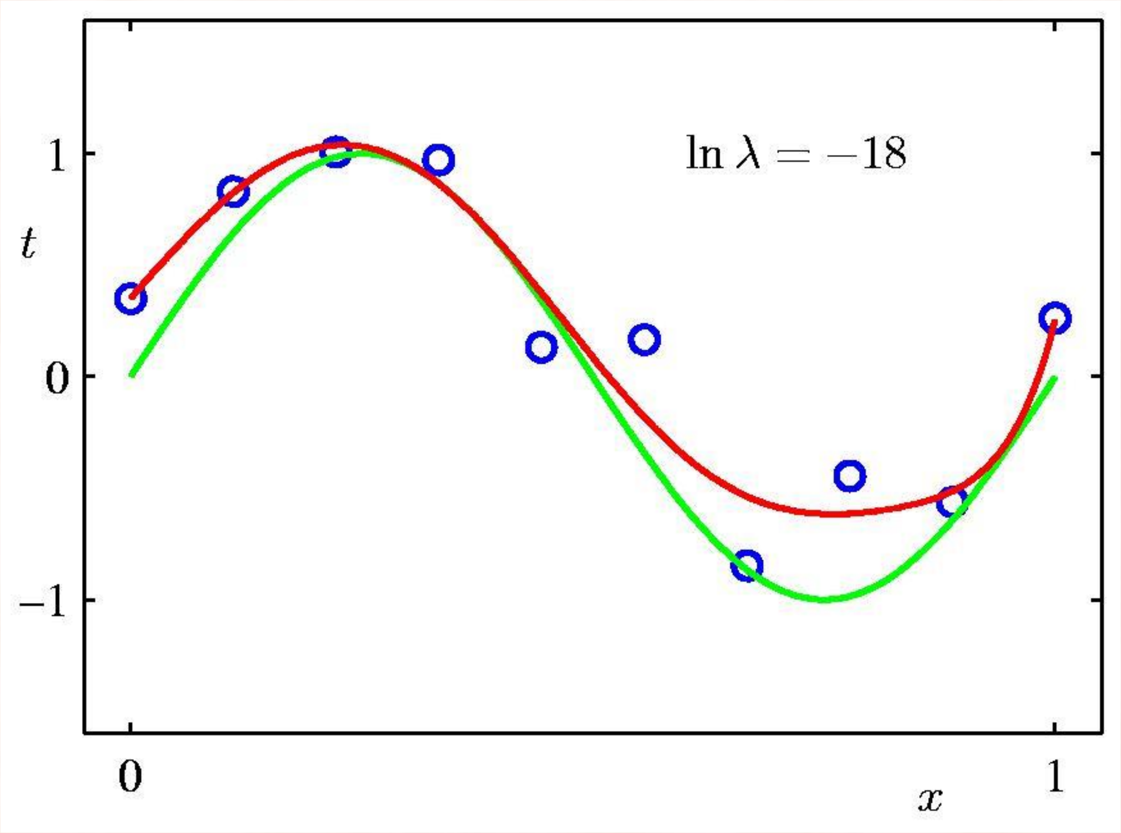
- Penalize large parameter values:

exclude w_0

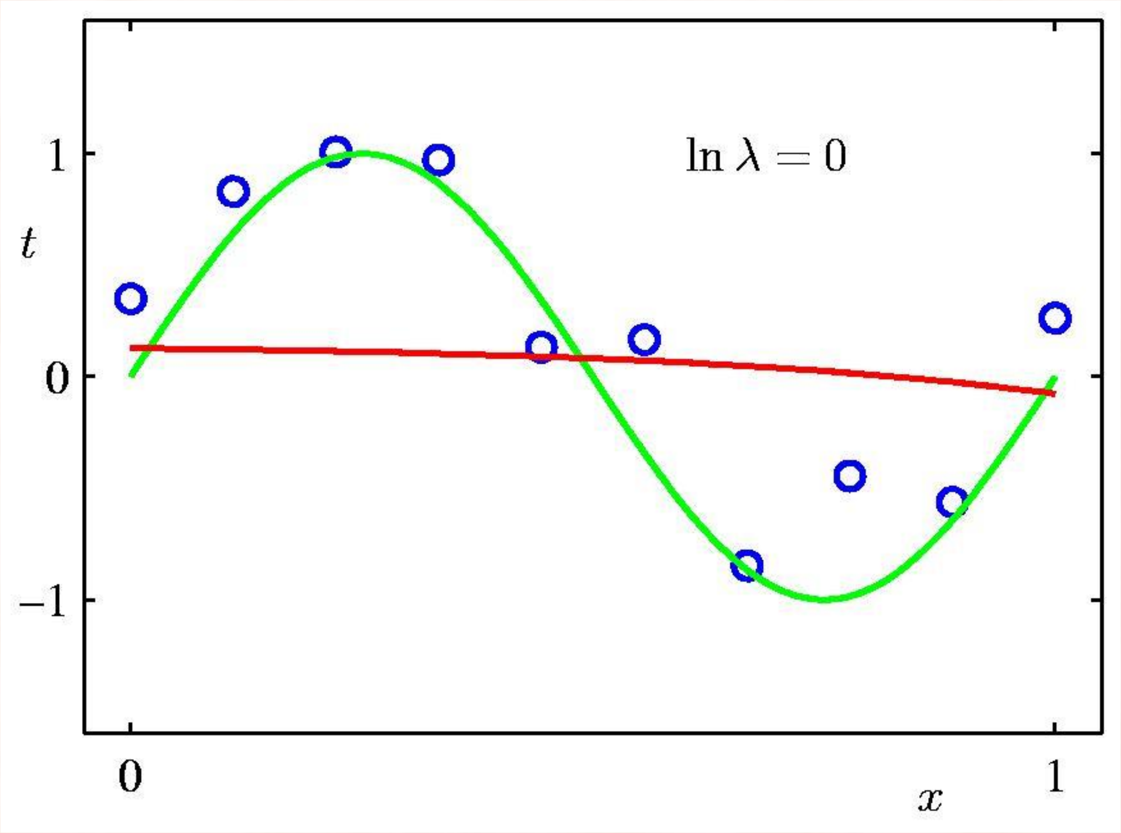
$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\left(\mathbf{w}^T\mathbf{x}^{(n)} - y_n\right)^2 + \underbrace{\frac{\lambda}{2}\|\mathbf{w}\|^2}_{L_2 \text{ norm regularizer}}$$

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} J(\mathbf{w})$$

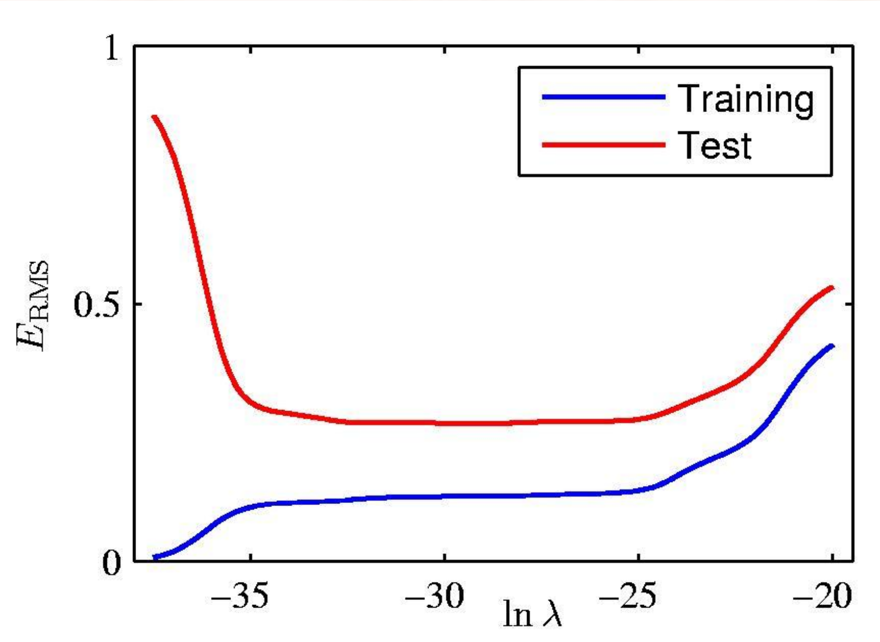
9^{th} Order Polynomial with Regularization



9^{th} Order Polynomial with Regularization



Training & Test error vs. $\ln \lambda$

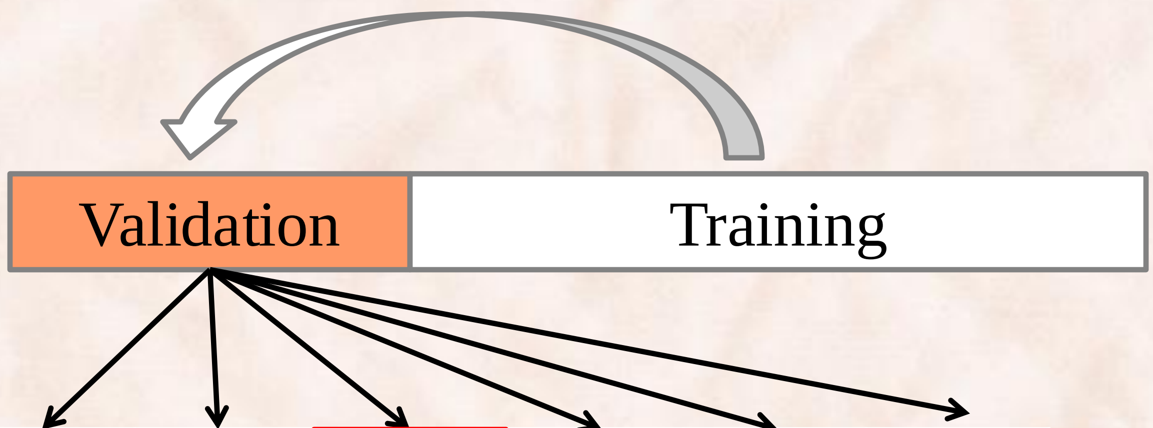


How do we find the optimal value of λ?

Model Selection

- Put aside an independent *validation set*.
- Select parameters giving best performance on validation set.

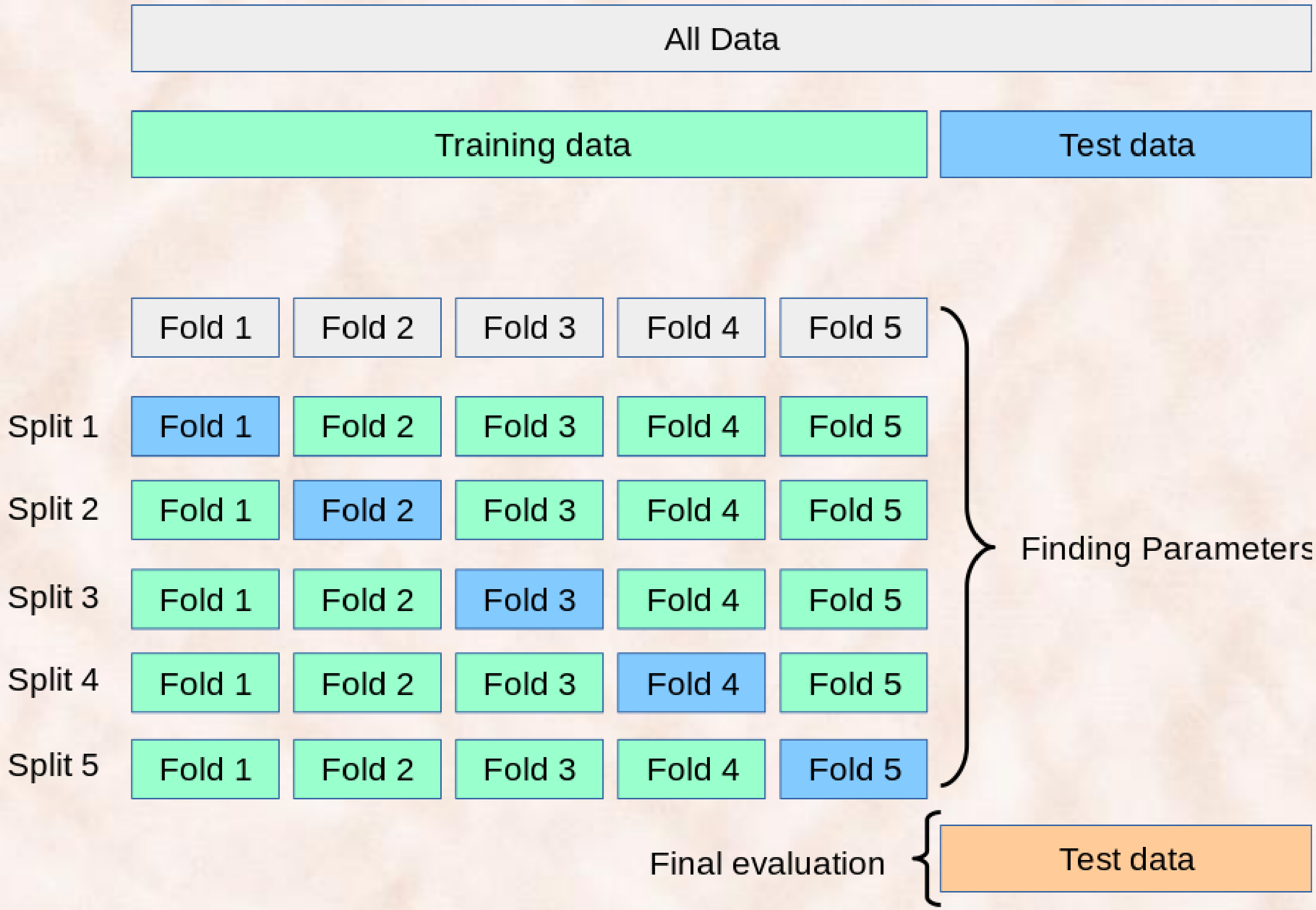
$$\ln \lambda \in \{-40, -35, -30, -25, -20, -15\}$$



$\ln \lambda$	-40	-35	-30	-25	-20	-15
E_{RMS}	1.05	0.30	0.25	0.27	0.52	0.55

K-fold Cross-Validation

https://scikit-learn.org/stable/modules/cross_validation.html



K-fold Cross-Validation

- Split the training data into K folds and try a wide range of tunning parameter values:
  - split the data into K folds of roughly equal size
  - iterate over a set of values for λ
    - iterate over k = 1, 2, ..., K
      - use all folds except k for training
      - validate (calculate test error) in the k-th fold
    - error[λ] = average error over the K folds
  - choose the value of λ that gives the smallest error.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoCV.html

Model Evaluation

- K-fold evaluation:
  - randomly partition dataset in K equally sized subsets P_1, P_2, … P_k
  - for each fold *i* in {1, 2, …, k}:
    - test on P_i, train on $P_1 \cup \ldots \cup P_{i-1} \cup P_{i+1} \cup \ldots \cup P_k$
  - compute average error/accuracy across K folds.

Normal Equations for Ridge Regression

- Multiple linear regression with L2 regularization:

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\left(\mathbf{w}^T\mathbf{x}^{(n)} - y_n\right)^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} J(\mathbf{w})$$

- Solution is $\mathbf{w} = \left(\lambda N\mathbf{I} + \mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}}\mathbf{t}$
  - Prove it.
    - This assumes w_0 is included in regularizer, rewrite so that it excludes w_0.

Batch Gradient Descent for Ridge Regression

- Sum-of-squares error + regularizer

$$\hat{y}_n = \mathbf{w}^T \mathbf{x}^{(n)}$$

$$J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} \left(\mathbf{w}^T \mathbf{x}^{(n)} - y_n\right)^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \, \nabla J(\mathbf{w}^{\tau})$$

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \left(\lambda \mathbf{w} + \frac{1}{N} \sum_{n=1}^{N} \left(\mathbf{w}^T \mathbf{x}^{(n)} - y_n\right) \mathbf{x}^{(n)} \right)$$

Implementation: Vectorization

- **Version 3**: Compute gradient, vectorized.

$$\nabla J(\mathbf{w}) = \lambda \mathbf{w} + \frac{1}{N}\sum_{n=1}^{N} \left(\mathbf{w}^T \mathbf{x}^{(n)} - y_n\right) \mathbf{x}^{(n)} \qquad \hat{y}_n = \mathbf{w}^T \mathbf{x}^{(n)}$$

grad = λ ∗ **w** + X.dot(**w**.dot(X) − **t**) / N

NumPy code above assumes examples stored in columns of X.

Homework: Rewrite to work with examples stored on rows.

Regularization: Ridge vs. Lasso

- Ridge regression:

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\left(\mathbf{w}^T\mathbf{x}^{(n)} - y_n\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{M} w_j^2$$

- Lasso:

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\left(\mathbf{w}^T\mathbf{x}^{(n)} - t_n\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{M} |w_j|$$

  – If λ is sufficiently large, some of the coefficients w_j are driven to 0 => *sparse* model.

Regularization: Ridge vs. Lasso

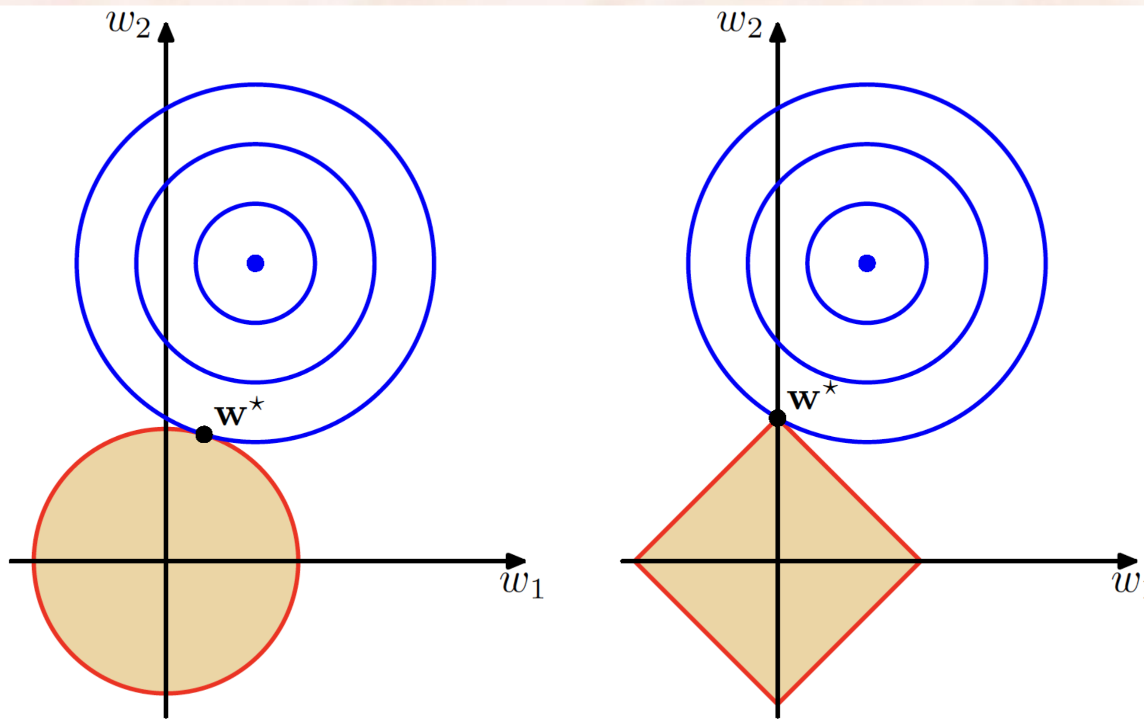


Figure 3.4 Plot of the contours of the unregularized error function (blue) along with the constraint region (3.30) for the quadratic regularizer $q = 2$ on the left and the lasso regularizer $q = 1$ on the right, in which the optimum value for the parameter vector $\mathbf{w}$ is denoted by $\mathbf{w}^\star$. The lasso gives a sparse solution in which $w_1^\star = 0$.

Regularization

- Regularization alleviates overfitting when using models with high capacity (e.g. high degree polynomials):
  - Want high capacity because we do not know how complicated the data is.

- *Q*: Can we achieve high capacity when doing curve fitting without using high degree polynomials?

- *A*: Use piecewise polynomial curves.
  - Example: **Cubic spline smoothing**.

Cubic Spline Smoothing

- **Cubic spline smoothing** is a regularized version of cubic spline interpolation.
  - **Cubic spline interpolation**: given *n* points $\{(x_i , y_i)\}$, connect adjacent points using cubic functions S_i , requiring that the spline and its first and second derivative remain continuous at all points:

$$S_i(x) = a_i(x-x_i)^3+b_i(x-x_i)^2+c_i(x-x_i)+d_i, \forall x \in [x_i, x_{i+1}]$$

  - **Cubic spline smoothing**: the spline S = $\{S_i\}$ is allowed to deviate from the data points and has low curvature => constrained optimization problem with objective:

$$L = \sum_{i=1}^{n} \frac{w_i}{Z}(S_i(x_i) - y_i)^2 + \frac{\lambda}{x_n - x_1} \int_{x_1}^{x_n} |S''(x)|^2 dx$$

$$w_i = \begin{cases} C, & \text{if } (x_i, y_i) \text{ is a significant local optima} \\ 1, & \text{otherwise} \end{cases}$$

Cubic Spline Smoothing

https://doi.org/10.1109/ICMLA.2011.39

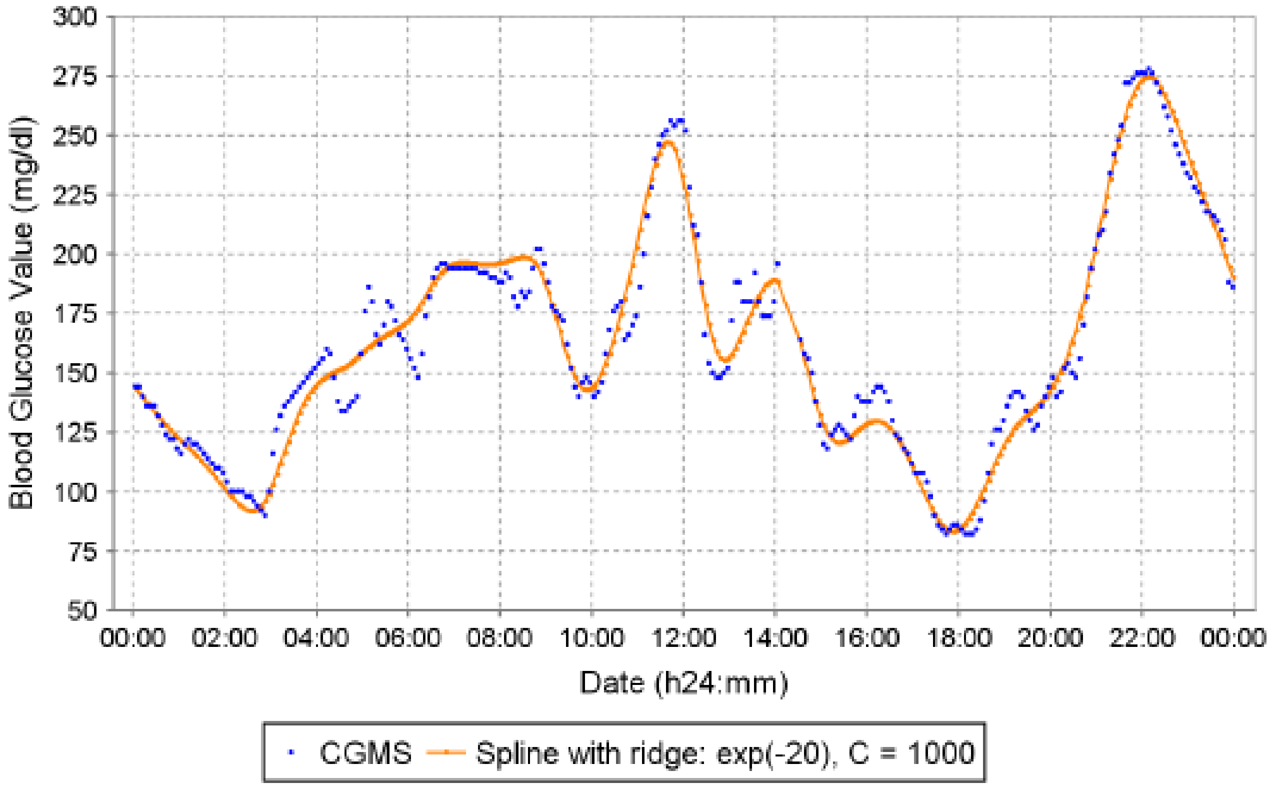
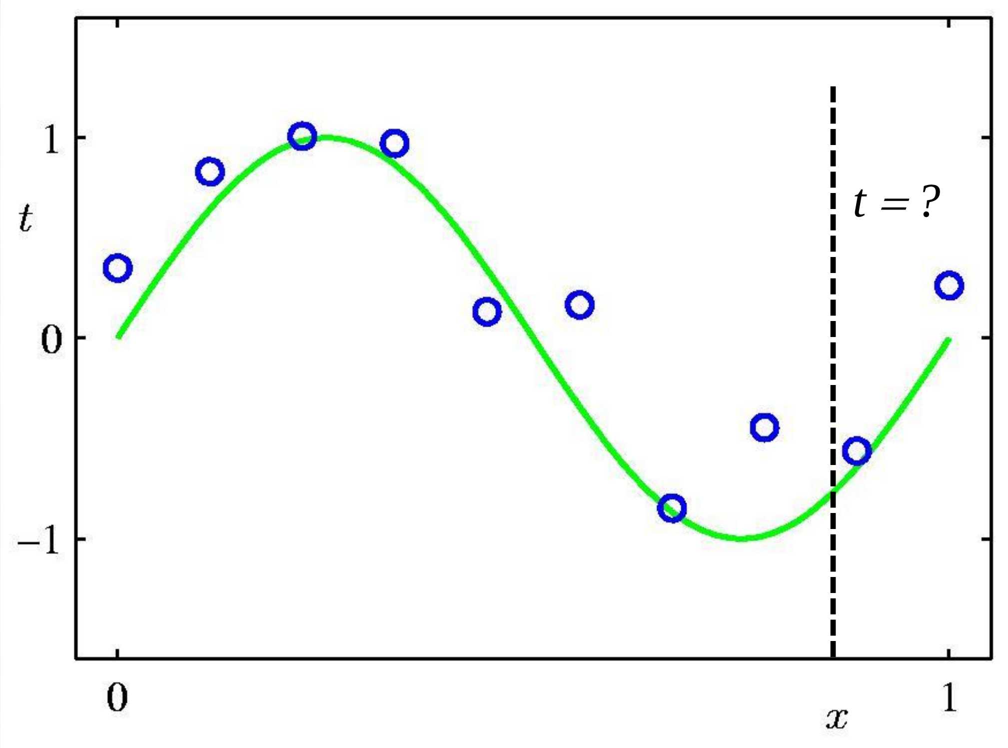


Fig. 3. Cubic spline smoothing with $\lambda = e^{-20}$ and $C = 1000$.

Polynomial Curve Fitting (Revisited)



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

parameters *features*

Generalization: Basis Functions as Features

- Generally

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

  where $\varphi_j(\mathbf{x})$ are known as *basis functions*.

- Typically $\varphi_0(\mathbf{x}) = 1$, so that w_0 acts as a bias.

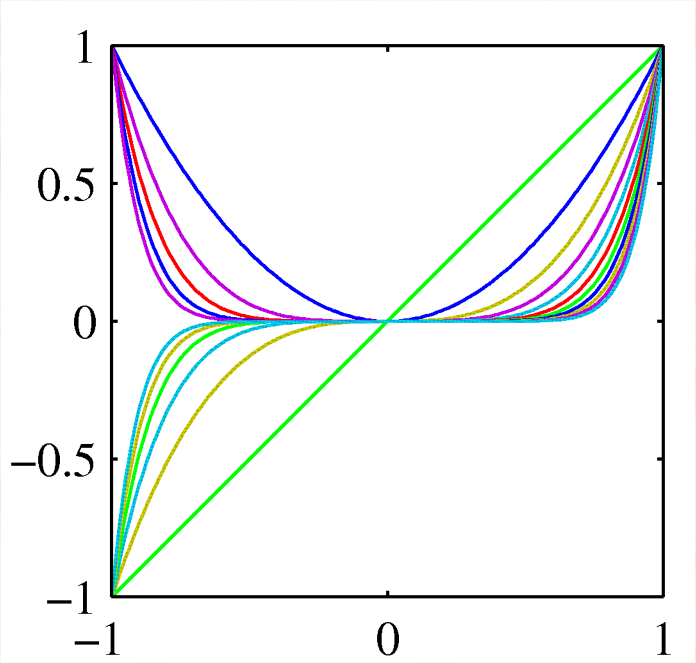
- In the simplest case, use linear basis functions : $\varphi_d(\mathbf{x}) = x_d$.

Linear Basis Function Models (1)

- Polynomial basis functions:

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

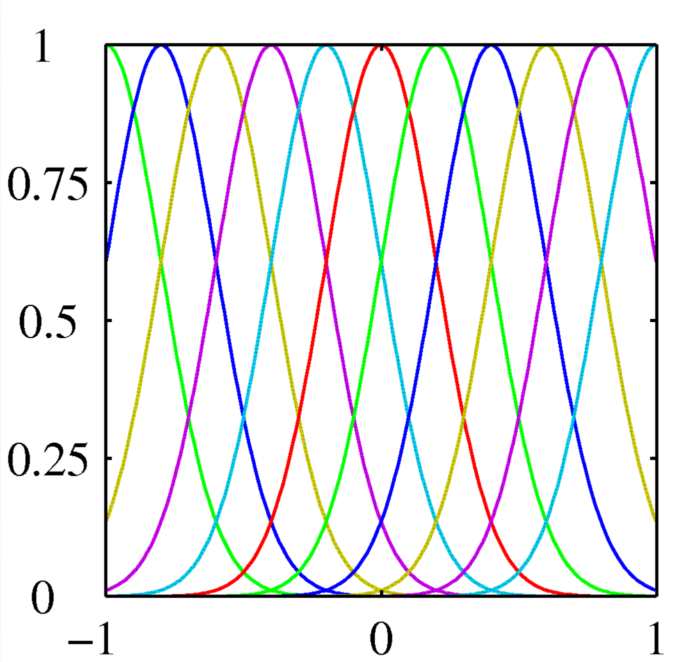
- Global behavior:
  - a small change in *x* affect all basis functions.



Linear Basis Function Models (2)

- Gaussian basis functions:

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



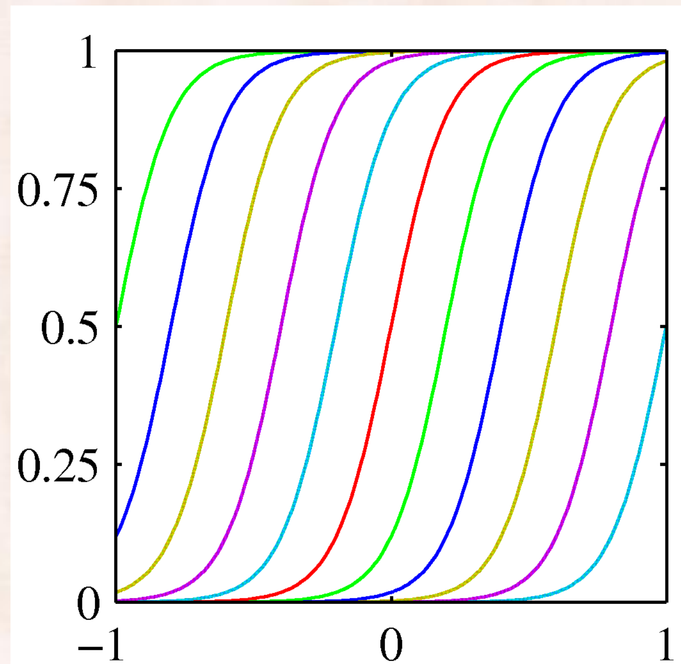
- Local behavior:
  - a small change in *x* only affects nearby basis functions.
  - μ_j and *s* control location and scale (width).

Linear Basis Function Models (3)

- Sigmoidal basis functions:

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

where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$.

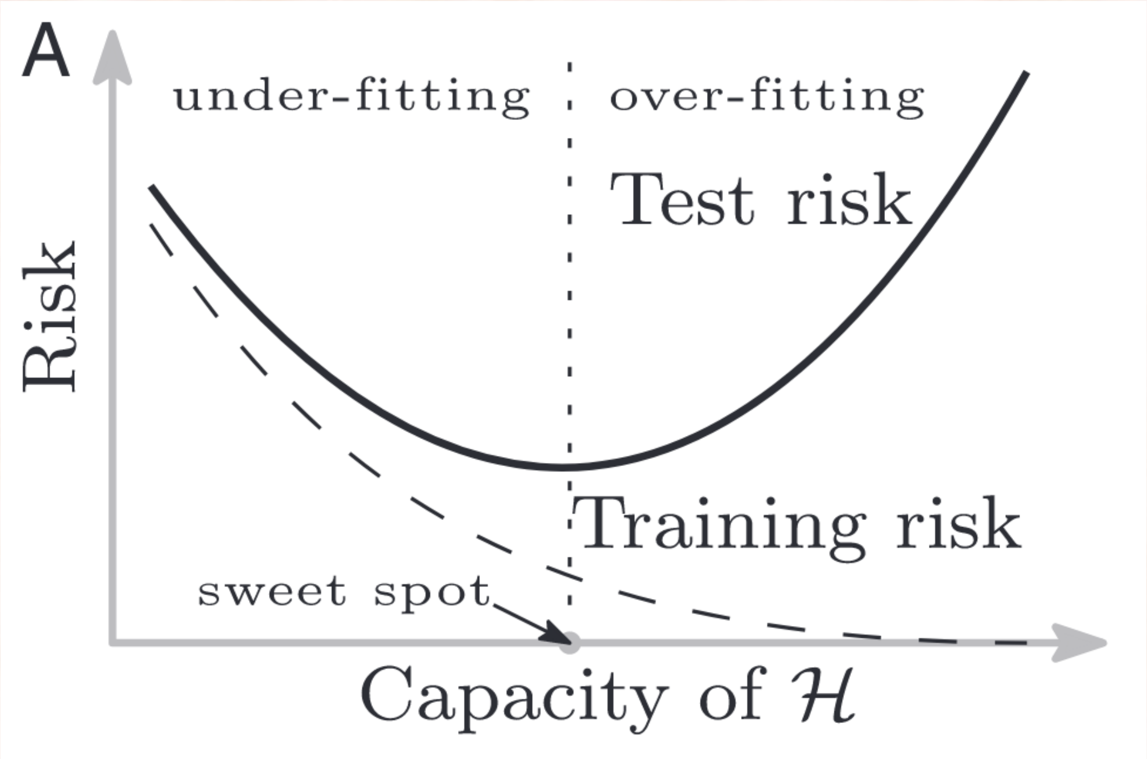


- Local behavior:
  - a small change in x only affect nearby basis functions.
  - μ_j and s control location and scale (slope).

Supplemental Topics

The Classical U-shaped risk curve

- The bias-variance trade-off:
  - Recommends balancing underfitting and overfitting.



The Modern Double-Descent risk curve

https://www.pnas.org/doi/10.1073/pnas.1903070116

- The modern interpolating regime:
  - Allow high capacity that can fit all training examples.
  - Of all models that fit, select model (fu with lowest norm.
    - e.g. from all degree M > N interpolating polynomials, select the one with lowest $||\mathbf{w}||^2$.