Machine Learning

BS/MS (Computer Science)

IQRA UNIVERSITY IU

Lecture-03
21-June-2014
Summer Semester

Course Layout

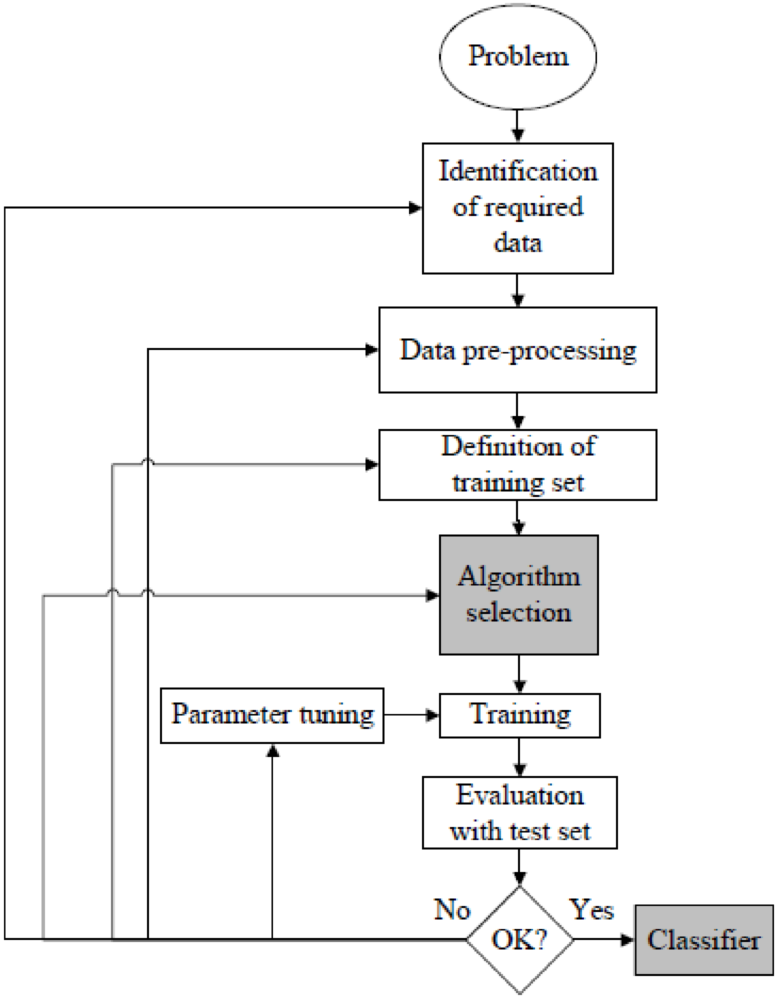
1. Introduction to Machine Learning
2. Overview of Machine Learning Algorithms
3. Data Analysis Methods
   - Scatter Plotting/Correlation Analysis
   - Principal Component Analysis
4. Supervised Machine Learning
   - Statistical Regression Methods
   - Artificial Neural Network
   - Decision Tree
   - Support Vector Machine
5. Unsupervised machine Learning
   - Clustering (k-means clustering, mixture models, hierarchical clustering)
   - Self-Organizing Map
   - Expectation Maximization Algorithm
6. Bayes Theorem and Bayesian Belief Network
7. Hidden Markov Model
8. Ensemble Learning Algorithms:
   - Bagging
   - Boosting
9. Pattern Mining
   - Association Rules
   - Apriori Algorithms
10. Information Search and Retrieval Methods
    - Vector Space Model
    - Latent Semantic Indexing
11. Application of Machine Leaning
    - Robotic/Image Processing/Fault Prediction

Lecture-03

Machine Learning: A Definition

Definition: A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

The Process Learning Model (Machine Learning)



Types of learning

- **Supervised learning**
  - Learning mapping between input **x** and desired output **y**
  - Teacher gives me y's for the learning purposes
- **Unsupervised learning**
  - Learning relations between data components
  - No specific outputs given by a teacher
- **Reinforcement learning**
  - Learning mapping between input **x** and desired output y
  - Critic does not give me y's but instead a signal (reinforcement) of how good my answer was
- **Other types of learning:**
  - **Concept learning, Active learning, Transfer learning, Deep learning**

Supervised learning

Data: $D = \{d_1, d_2, .., d_n\}$ **a set of *n* examples**

$d_i = < \mathbf{x}_i, y_i >$

$\mathbf{x}_i$ is input vector, and y is desired output (given by a teacher)

Objective: learn the mapping $f : X \rightarrow Y$

s.t. $y_i \approx f(x_i)$ for all $i = 1,.., n$

Two types of problems:

- **Regression:** X discrete or continuous $\rightarrow$ Y is **continuous**
- **Classification:** X discrete or continuous $\rightarrow$ Y is **discrete**

Supervised learning examples

- **Regression:** Y is **continuous**

Debt/equity
Earnings
Future product orders

→ company stock price

- **Classification:** Y is **discrete**

```
# ######
      ##
     ##
  ######
      ###
#      ##
# ##   ##
  #######
```

→ Label “3”

Handwritten digit (array of 0,1s)

Unsupervised learning

- **Data:** $D = \{d_1, d_2, .., d_n\}$
  $d_i = \mathbf{x}_i$ vector of values
  No target value (output) y

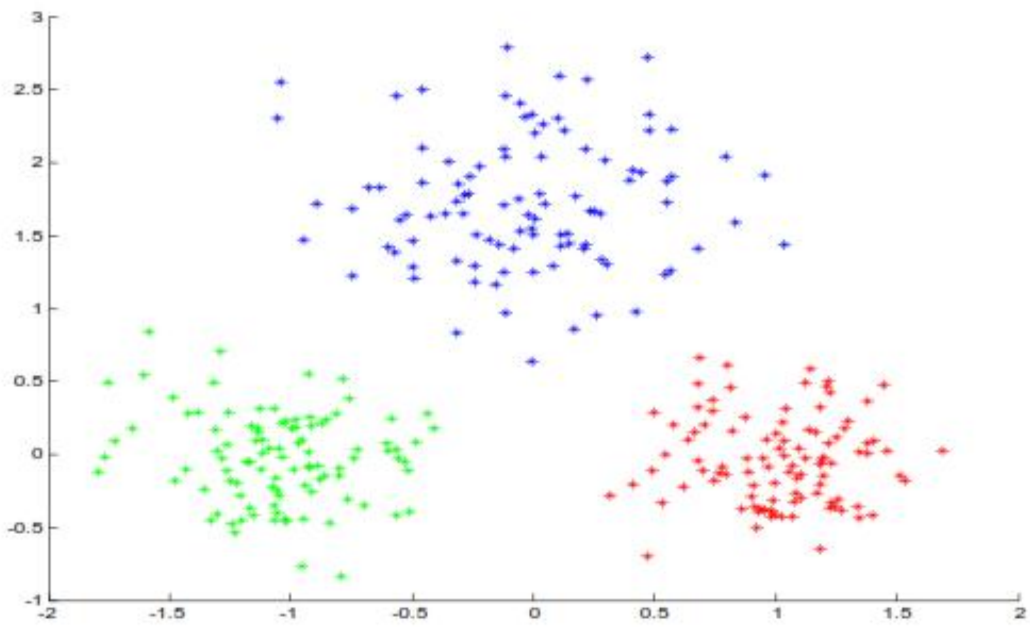
- **Objective:**
  – learn relations between samples, components of samples

Types of problems:

- **Clustering**
  Group together "similar" examples, e.g. patient cases
- **Density estimation**
  – Model probabilistically the population of samples

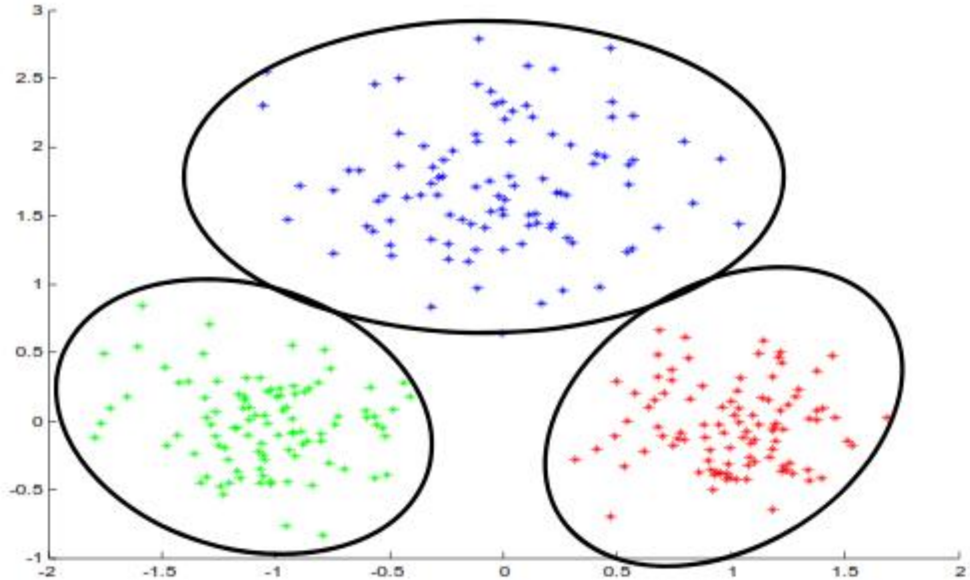
Unsupervised learning example

- **Clustering.** Group together similar examples $d_i = \mathbf{x}_i$



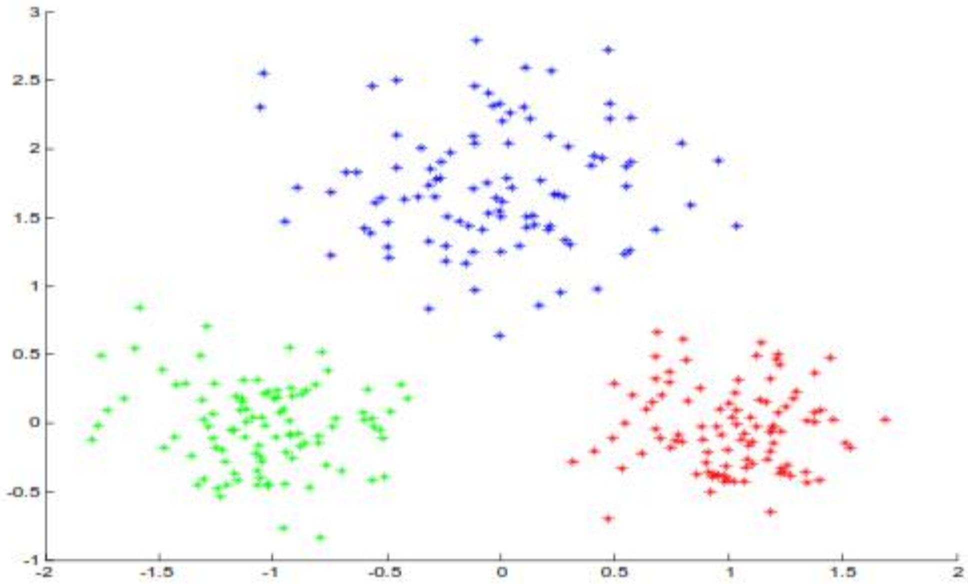
Unsupervised learning example

- **Clustering.** Group together similar examples $d_i = \mathbf{x}_i$



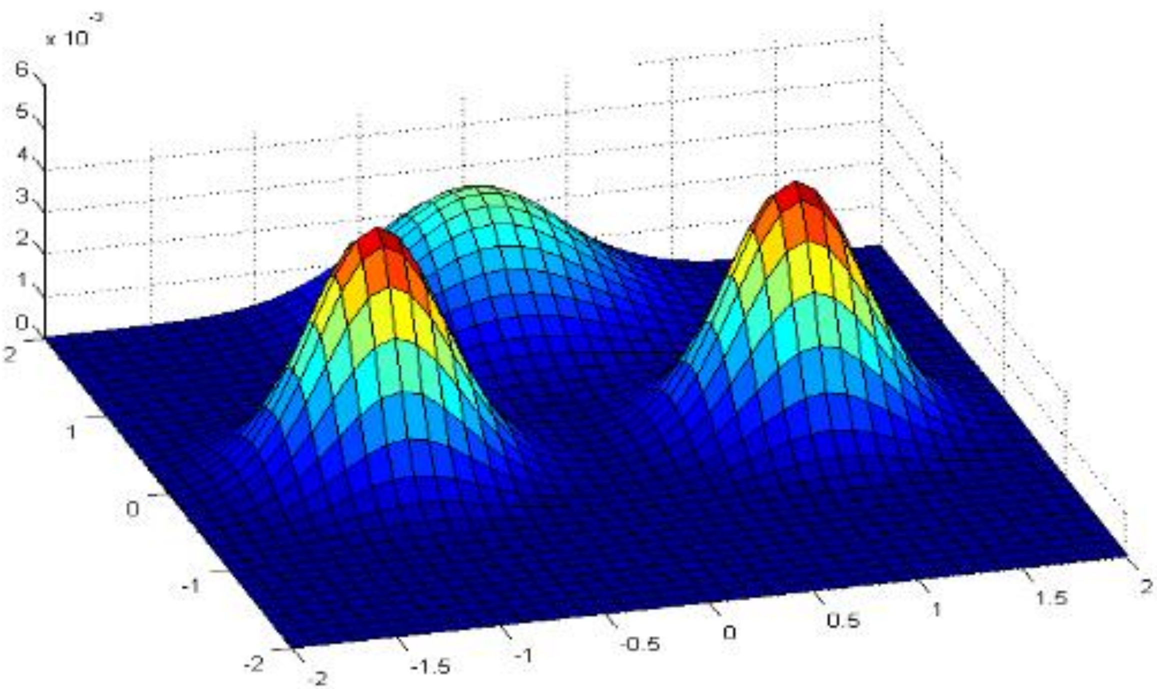
Unsupervised learning example

- **Density estimation.** We want to build the probability model P($\mathbf{x}$) of a population from which we draw examples $d_i = \mathbf{x}_i$



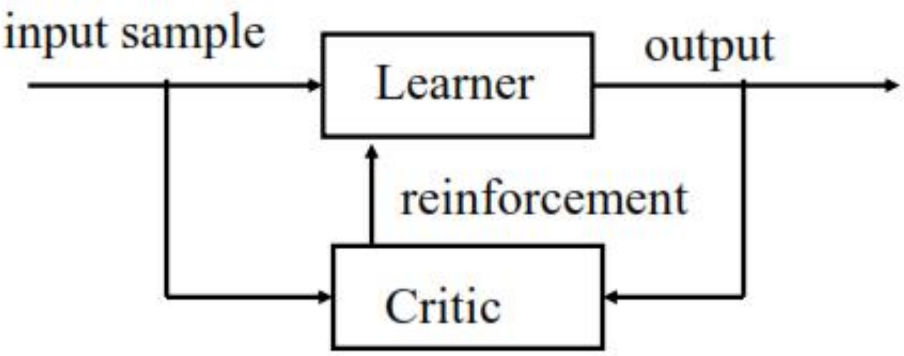
Unsupervised learning. Density estimation

- A probability density of a point in the two dimensional space
  - Model used here: **Mixture of Gaussians**



Reinforcement learning

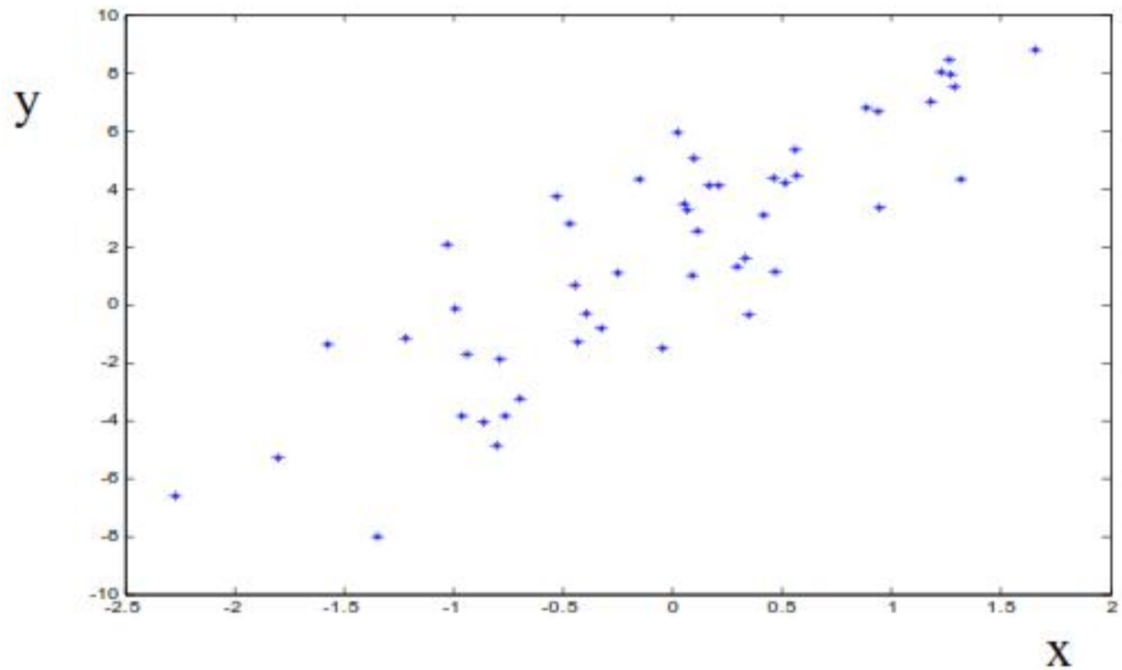
- We want to learn: $f : X \rightarrow Y$
- We see samples of $\mathbf{x}$ but not y
- Instead of y we get a feedback (reinforcement) from a **critic** about how good our output was



- The goal is to select outputs that lead to the best reinforcement

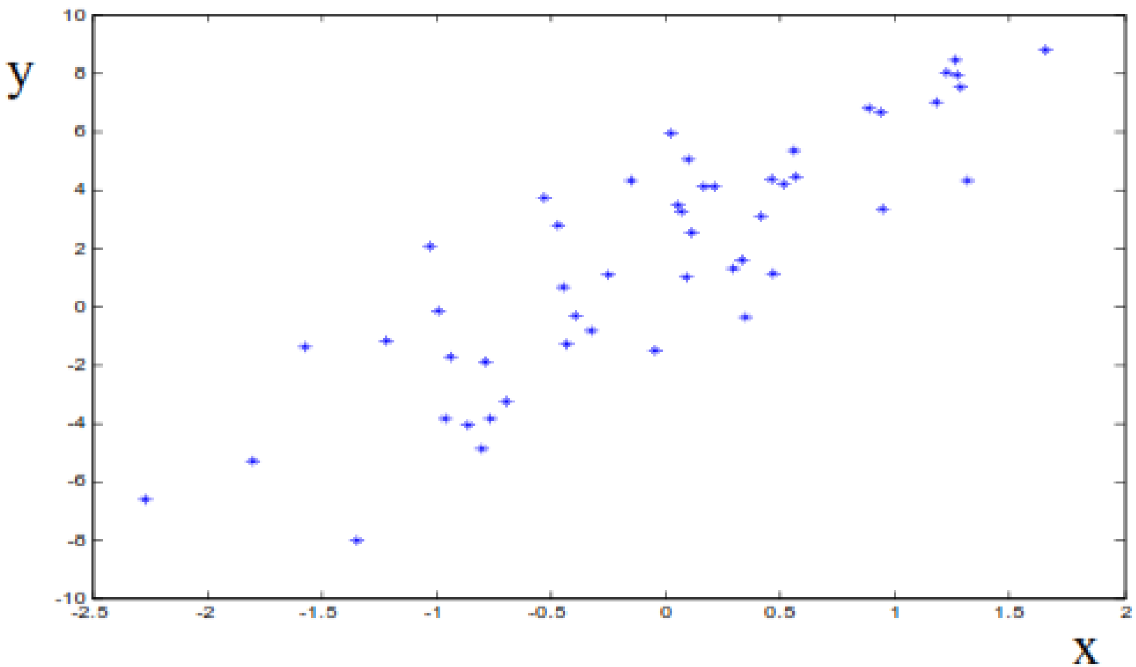
Learning: first look

- Assume we see examples of pairs ($\mathbf{x}$, y) in D and we want to learn the mapping $f : X \rightarrow Y$ to predict y for some future **x**
- We get the data D - what should we do?



Learning: first look

- **Problem:** many possible functions $f : X \rightarrow Y$ exists for representing the mapping between **x** and y
- Which one to choose? Many examples still unseen!

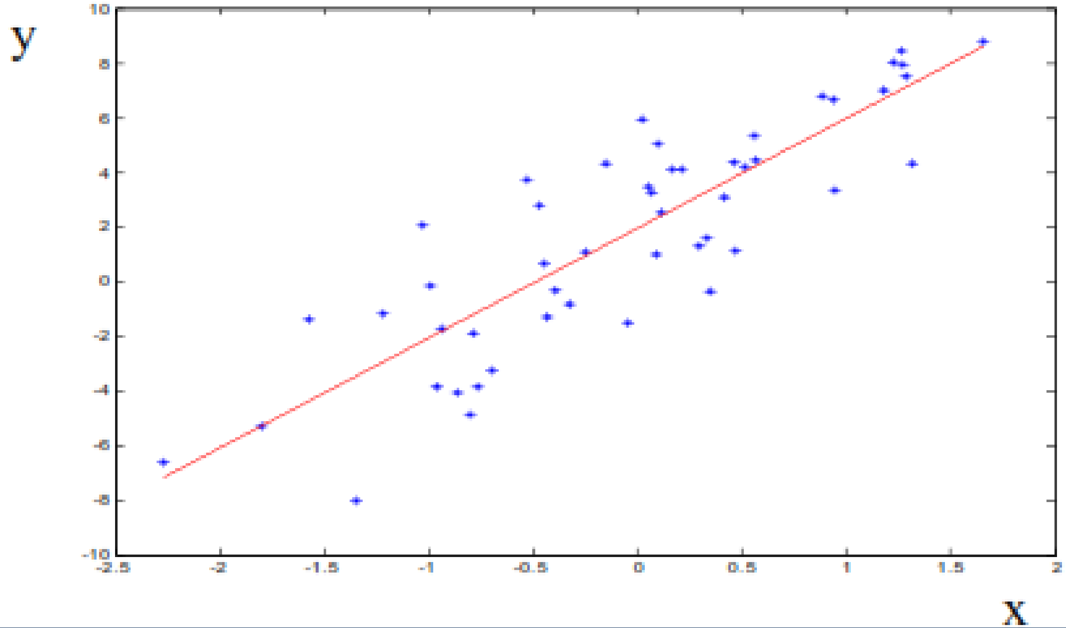


Learning: first look

- **Solution:** make an assumption about the model, say,

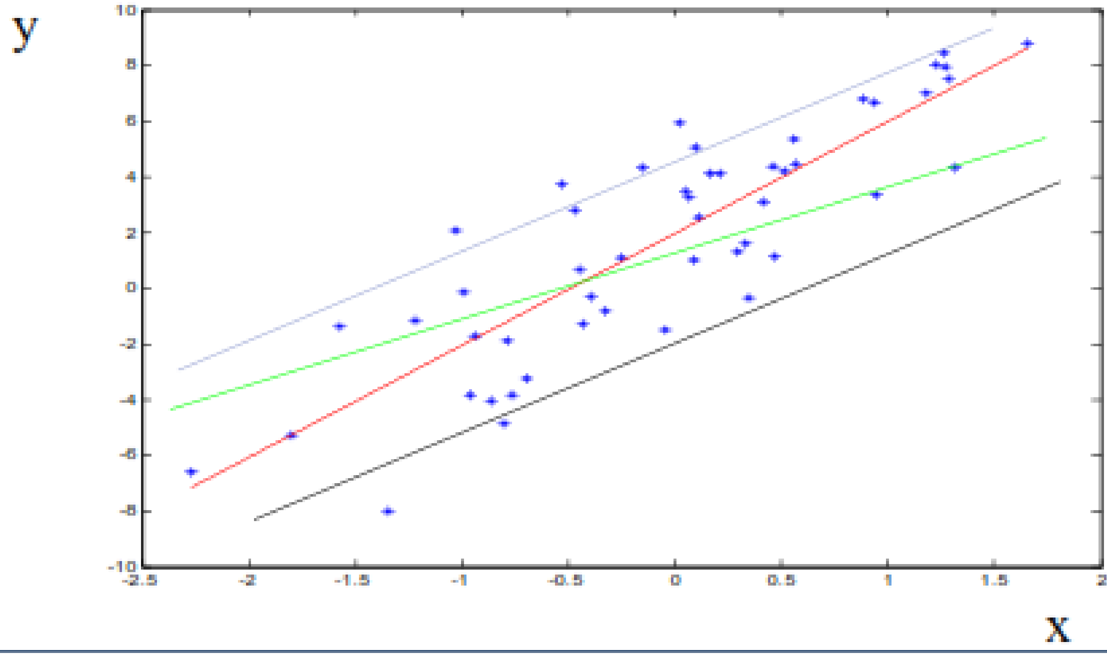
$$f(x) = ax + b + \varepsilon$$

$\varepsilon = N(0,\sigma)$ - random (normally distributed) noise



Learning: first look

- Choosing a parametric model or a set of models is not enough
  Still too many functions $f(x) = ax + b + \varepsilon \quad \varepsilon = N(0, \sigma)$
  - One for every pair of parameters a, b



Fitting the data to the model

- We want the **best set** of model parameters

Objective: Find parameters that:

- reduce the misfit between the model **M** and observed data ***D***
- Or, (in other words) explain the data the best

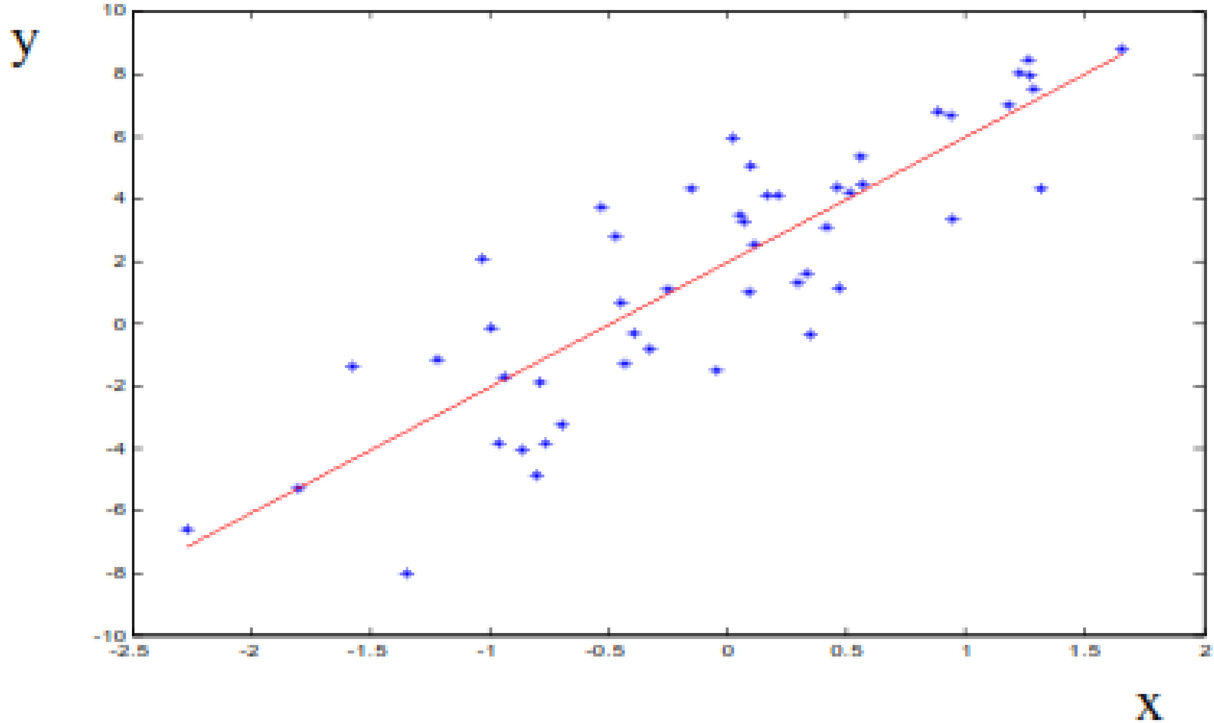
Objective function:

- **Error function: Measures the misfit between *D* and M**
- **Examples of error functions:**
  - Average Square Error $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$
  - Average misclassification error $\frac{1}{n}\sum_{i=1}^{n} 1_{y_i \neq f(x_i)}$

Average # of misclassified cases

Fitting the data to the model

- **Linear regression problem**
  - Minimizes the squared error function for the linear model
  - minimizes $$\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$



Learning: summary

Three basic steps:

- **Select a model** or a set of models (with parameters)

  E.g. $f(x) = ax + b$

- **Select the error function** to be optimized

  E.g. $$\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$

- **Find the set of parameters optimizing the error function**
  - The model and parameters with the smallest error represent the best fit of the model to the data

But there are problems one must be careful about …

Learning

Problem

- We fit the model based on past experience (past examples seen)
- But ultimately we are interested in learning the mapping that performs well on the whole population of examples

Training data: **Data used to fit the parameters of the model**

Training error: $$\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$

True (generalization) error (over the whole unknown population):

$$E_{(x,y)}[(y - f(x))^2] \quad \text{Mean squared error}$$

Training error tries to approximate the true error !!!!

Does a good training error imply a good generalization error ?

Learning

Problem

- We fit the model based on past examples observed in ***D***
- But ultimately we are interested in learning the mapping that performs well on the whole population of examples

Training data: Data used to fit the parameters of the model

Training error:

$$Error(D, f) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$

True (generalization) error (over the whole population):

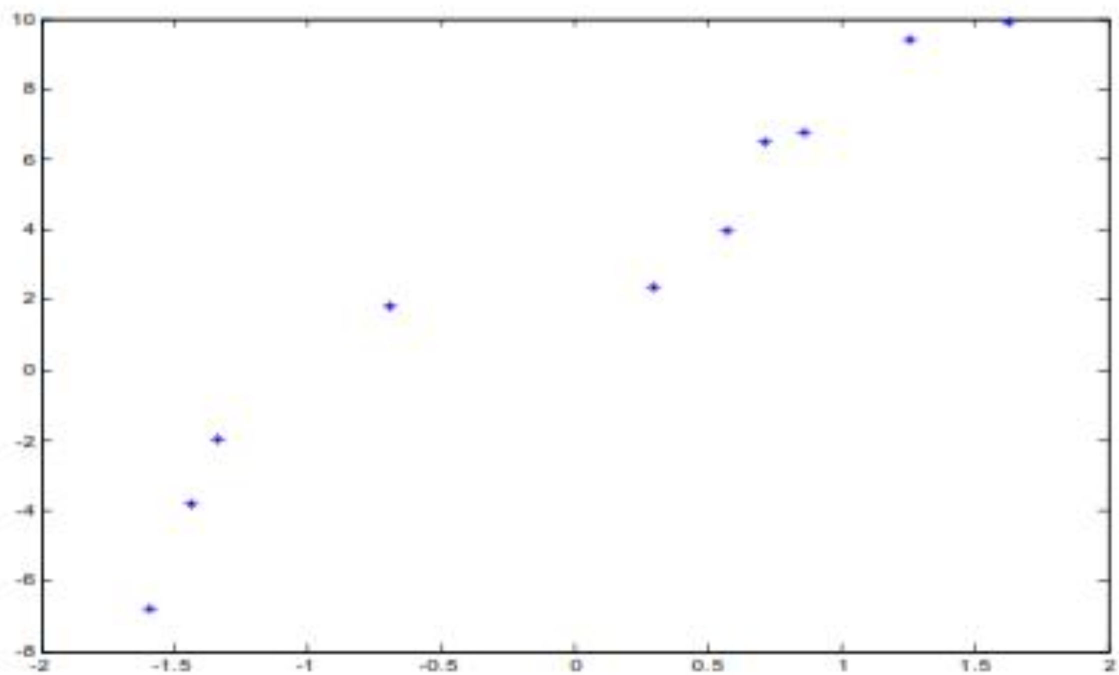
$$E_{(x,y)}[(y - f(x))^2] \qquad \text{Mean squared error}$$

Training error tries to approximate the true error !!!!

Does a good training error imply a good generalization error ?

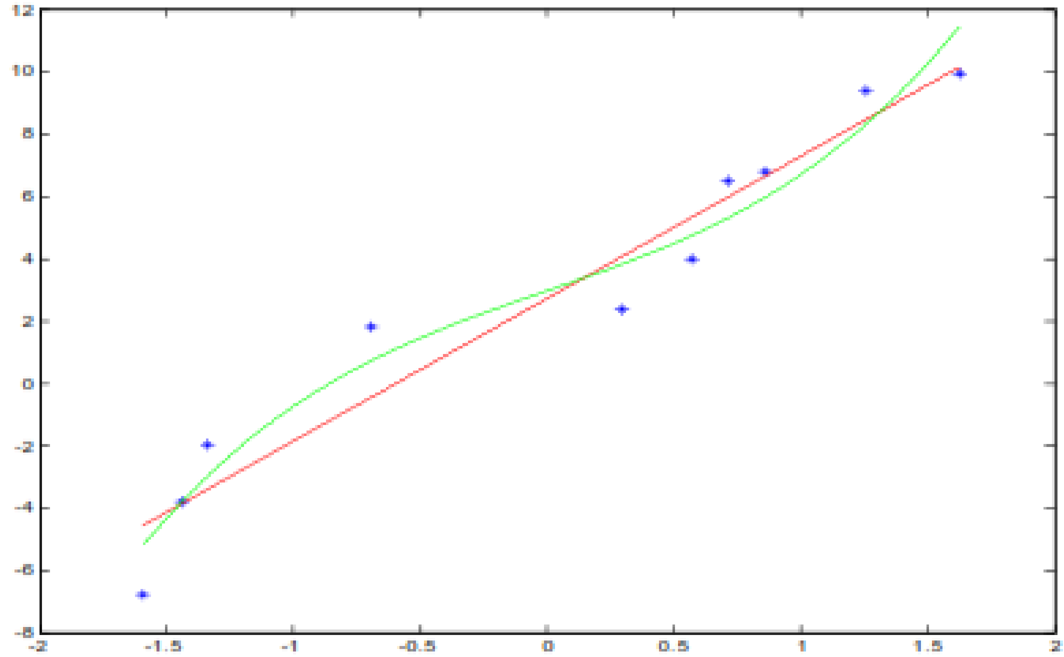
Overfitting

- Assume we have a set of 10 points and we consider polynomial functions as our possible models



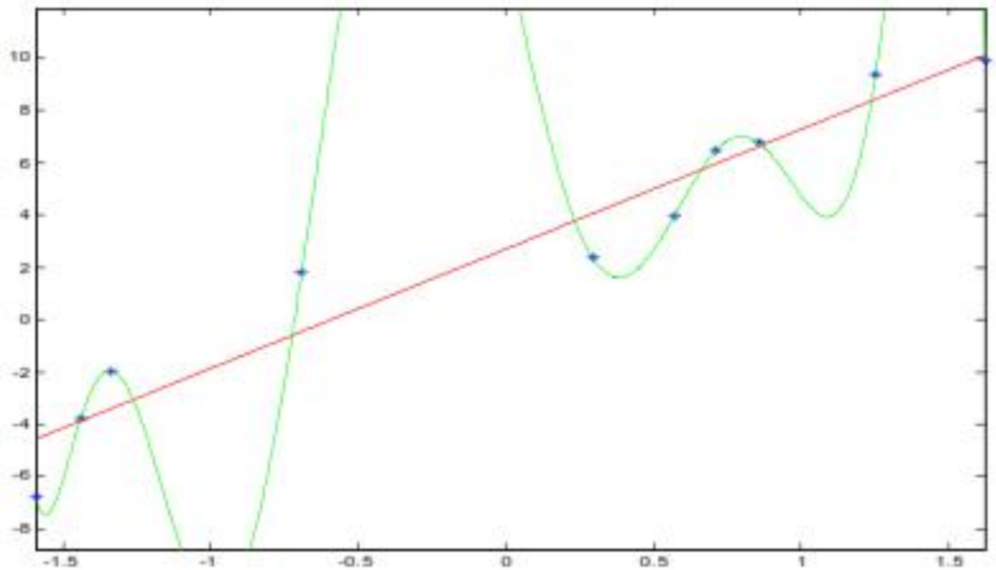
Overfitting

- Linear vs. cubic polynomial
- Higher order polynomial leads to a better fit, smaller error



Overfitting

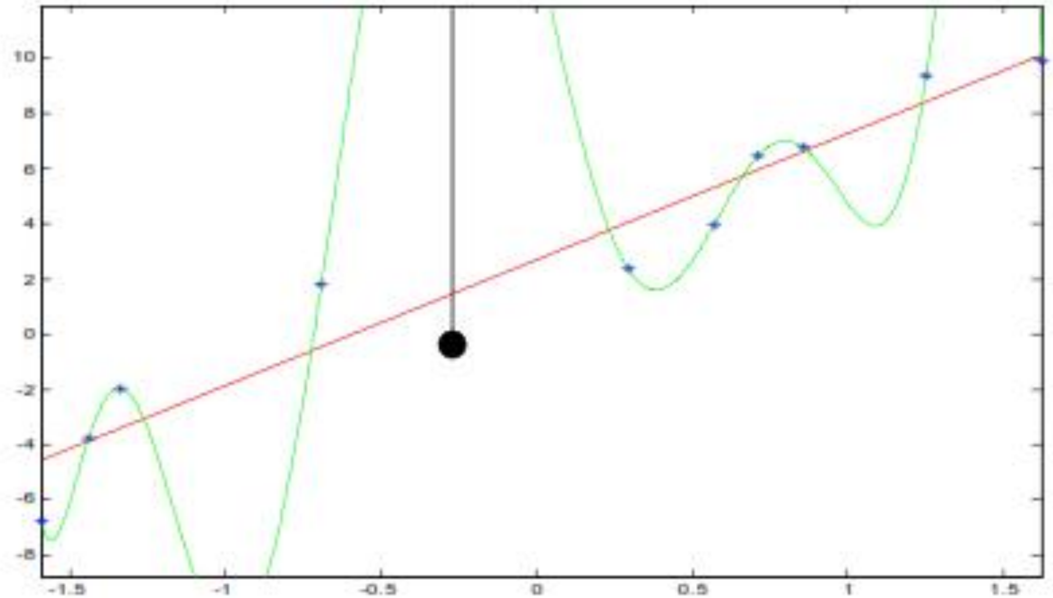
- For 10 data points, the degree 9 polynomial gives a perfect fit (Lagrange interpolation). Error is zero.
- Is it always good to minimize the training error?



Overfitting

Situation when the training error is low and the generalization error is high. Causes of the phenomenon:

- Model with a large number of parameters (degrees of freedom)
- Small data size (as compared to the complexity of the model)



Two types of linear model that are equivalent with respect to learning

$$y(\mathbf{x},\mathbf{w}) = \underset{\text{bias}}{w_0} + w_1 x_1 + w_2 x_2 + \ldots = \mathbf{w}^T \mathbf{x}$$

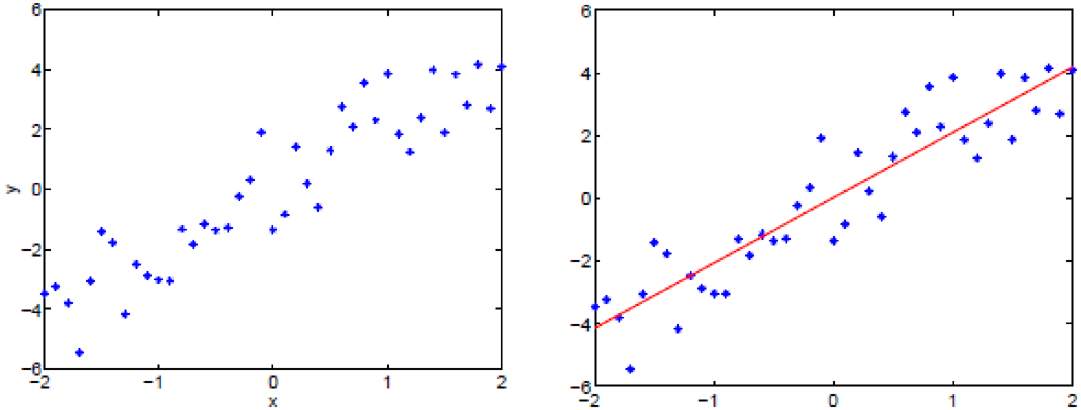
$$y(\mathbf{x},\mathbf{w}) = w_0 + w_1 \phi_1(\mathbf{x}) + w_2 \phi_2(\mathbf{x}) + \ldots = \mathbf{w}^T \Phi(\mathbf{x})$$

- The first model has the same number of adaptive coefficients as the dimensionality of the data +1.
- The second model has the same number of adaptive coefficients as the number of basis functions +1.
- Once we have replaced the data by the outputs of the basis functions, fitting the second model is exactly the same problem as fitting the first model.

The Loss Function

- Fitting a model to data is typically done by finding the parameter values that minimize some loss function.
- There are many possible loss functions. What criterion should we use for choosing one?
  - Choose one that makes the math easy (squared error)
  - Choose one that makes the fitting correspond to maximizing the likelihood of the training data given some noise model for the observed outputs.
  - Choose one that makes it easy to interpret the learned coefficients (easy if mostly zeros)
  - Choose one that corresponds to the real loss on a practical application (losses are often asymmetric)

Linear Regression



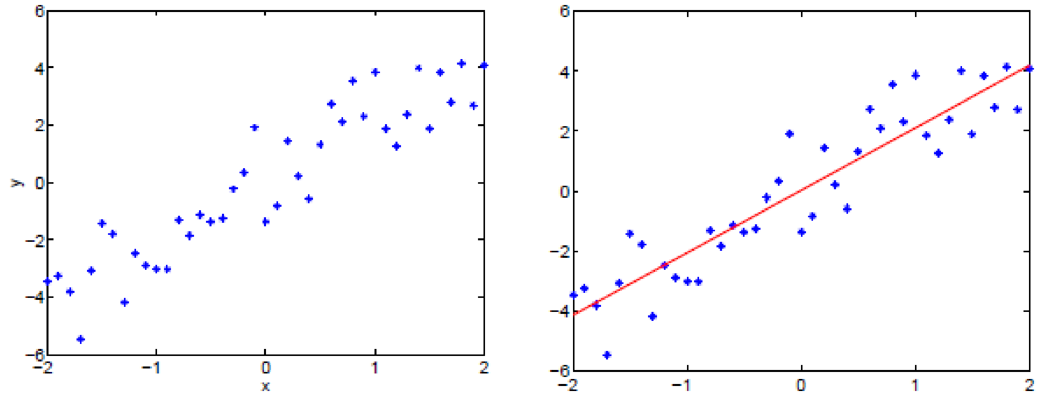
- We begin by considering linear regression (easy to extend to more complex predictions later on)

$$f : \mathcal{R} \rightarrow \mathcal{R} \qquad f(x; \mathbf{w}) = w_0 + w_1 x$$

$$f : \mathcal{R}^d \rightarrow \mathcal{R} \qquad f(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + \ldots w_d x_d$$

where $\mathbf{w} = [w_0, w_1, \ldots, w_d]^T$ are *parameters* we need to set.

Linear Regression: Squared Loss



$$f : \mathcal{R} \rightarrow \mathcal{R} \qquad f(x; \mathbf{w}) = w_0 + w_1 x$$

$$f : \mathcal{R}^d \rightarrow \mathcal{R} \qquad f(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + \ldots w_d x_d$$

- We can measure the prediction loss in terms of squared error, $\text{Loss}(y, \hat{y}) = (y - \hat{y})^2$, so that the empirical loss on n training samples becomes *mean squared error*

$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f(\mathbf{x}_i; \mathbf{w}) \right)^2$$

Linear Regression: Estimation

- We have to minimize the *empirical* squared loss

$$
\begin{aligned}
J_n(\mathbf{w}) &= \frac{1}{n}\sum_{i=1}^{n}\big(y_i - f(\mathbf{x}_i;\mathbf{w})\big)^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i)^2 \quad \text{(1-dim)}
\end{aligned}
$$

By setting the derivatives with respect to w_1 and w_0 to zero, we get necessary conditions for the "optimal" parameter values

$$
\begin{aligned}
\frac{\partial}{\partial w_1} J_n(\mathbf{w}) &= 0 \\
\frac{\partial}{\partial w_0} J_n(\mathbf{w}) &= 0
\end{aligned}
$$

Optimality Conditions: Derivation

$$
\begin{aligned}
\frac{\partial}{\partial w_1} J_n(\mathbf{w}) &= \frac{\partial}{\partial w_1} \frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)^2 \\
&= \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial w_1} (y_i - w_0 - w_1 x_i)^2 \\
&= \frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i) \frac{\partial}{\partial w_1} (y_i - w_0 - w_1 x_i) \\
&= \frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)(-x_i) = 0 \\
\frac{\partial}{\partial w_0} J_n(\mathbf{w}) &= \frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)(-1) = 0
\end{aligned}
$$

Interpretation

- If we denote the prediction error as $\epsilon_i = (y_i - w_0 - w_1 x_i)$ then the optimality conditions can be written as

$$\frac{1}{n}\sum_{i=1}^{n} \epsilon_i x_i = 0, \quad \frac{1}{n}\sum_{i=1}^{n} \epsilon_i = 0$$

Thus the prediction error is uncorrelated with any linear function of the inputs

but not with a quadratic function of the inputs

$$\frac{1}{n}\sum_{i=1}^{n} \epsilon_i x_i^2 \neq 0 \quad \text{(in general)}$$

Linear Regression: Matrix Notation

- We can express the solution a bit more generally by resorting to a matrix notation

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \cdots \\ y_n \end{bmatrix}, \; \mathbf{X} = \begin{bmatrix} 1 & x_1 \\ \cdots & \cdots \\ 1 & x_n \end{bmatrix}, \; \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

so that

$$\begin{aligned} \frac{1}{n}\sum_{t=1}^{n}(y_t - w_0 - w_1 x_t)^2 &= \frac{1}{n}\left\| \begin{bmatrix} y_1 \\ \cdots \\ y_n \end{bmatrix} - \begin{bmatrix} 1 & x_1 \\ \cdots & \cdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \right\|^2 \\ &= \frac{1}{n}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 \end{aligned}$$

Linear Regression: Solution

By setting the derivatives of $\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2/n$ to zero, we get the same optimality conditions as before, now expressed in a matrix form

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{w}} \frac{1}{n} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 &= \frac{\partial}{\partial \mathbf{w}} \frac{1}{n} (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) \\
&= \frac{2}{n} \mathbf{X}^T (\mathbf{y} - \mathbf{X}\mathbf{w}) \\
&= \frac{2}{n} (\mathbf{X}^T \mathbf{y} - \mathbf{X}^T \mathbf{X}\mathbf{w}) = \mathbf{0}
\end{aligned}$$

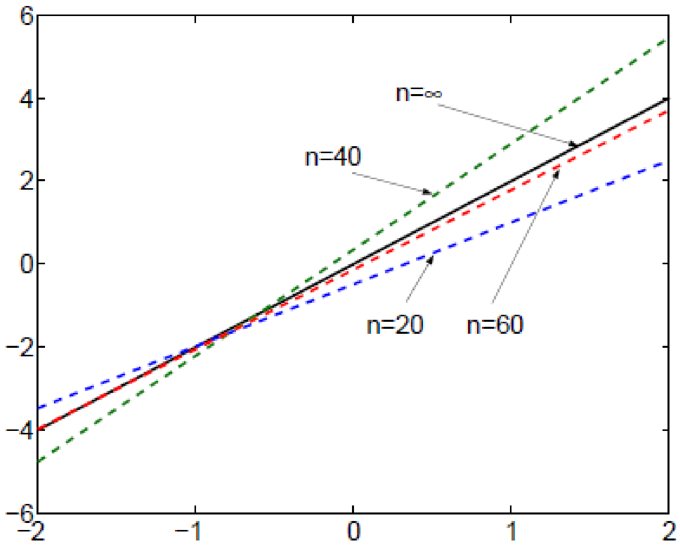
which gives

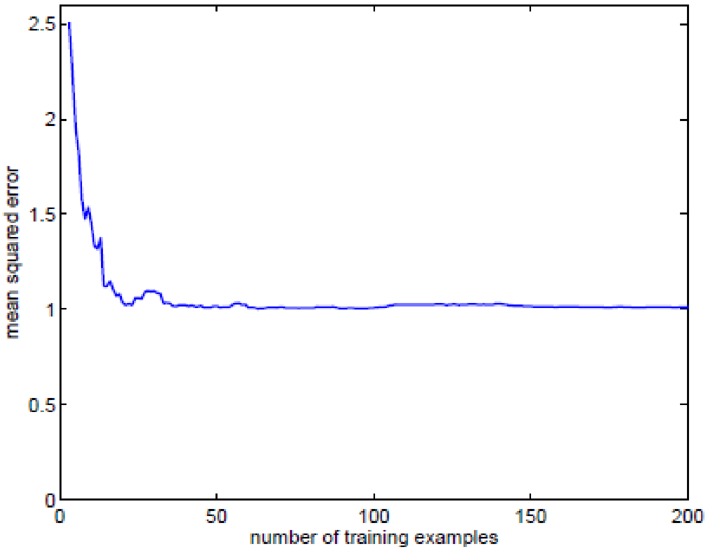
$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

- The solution is a linear function of the outputs y

Linear Regression: Generalization

- As the number of training examples increases our solution gets “better”





We'd like to understand the error a bit better

Minimizing Squared Error

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

$$error = \sum_n (t_n - \mathbf{w}^T \mathbf{x}_n)^2$$

$$\mathbf{w}^* = \boxed{(\mathbf{X}^T \mathbf{X})^{-1}} \mathbf{X}^T \boxed{\mathbf{t}}$$

vector of target values

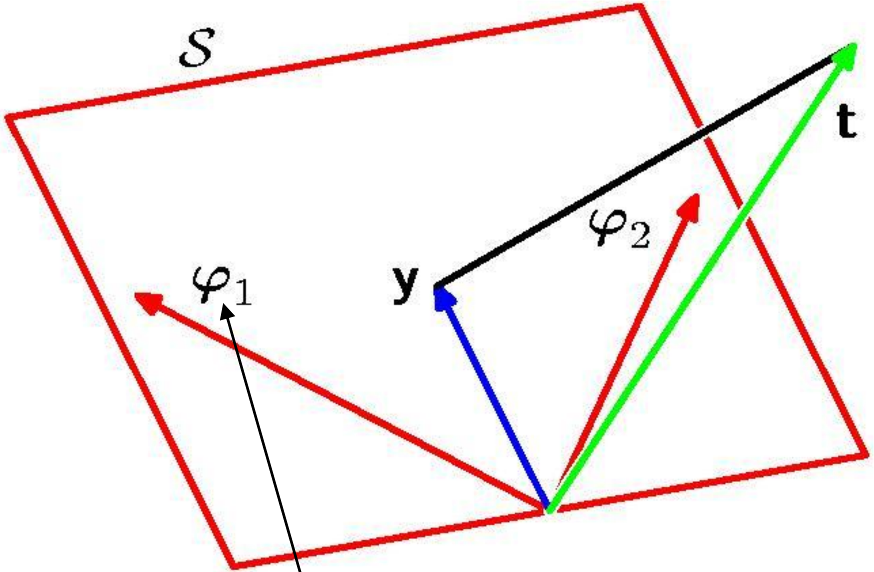
optimal weights

inverse of the covariance matrix of the input vectors

the transposed design matrix has one input vector per column

A Geometrical View of the Solution

- The space has one axis for each training case.
- So the vector of target values is a point in the space.
- Each vector of the values of one component of the input is also a point in this space.
- The input component vectors span a subspace, S.
  - A weighted sum of the input component vectors must lie in S.
- The optimal solution is the orthogonal projection of the vector of target values onto S.



3.1	4.2
1.5	2.7
0.6	1.8

input vector

component vector

Least Mean Squares: An alternative approach for really big datasets

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \boxed{\nabla E_{n(\tau)}}$$

weights after seeing training case tau+1

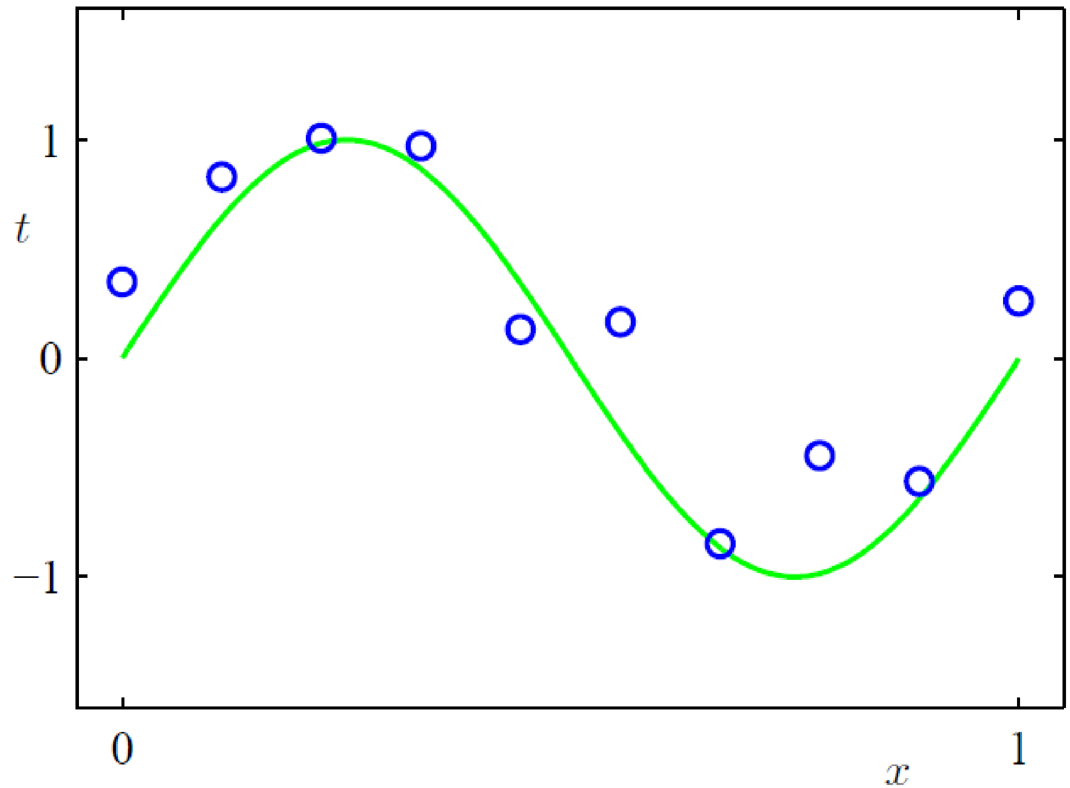
learning rate

vector of derivatives of the squared error w.r.t. the weights on the training case presented at time tau.

- This is called "online" learning. It can be more efficient if the dataset is very redundant and it is simple to implement in hardware.
  - It is also called stochastic gradient descent if the training cases are picked at random.
  - Care must be taken with the learning rate to prevent divergent oscillations, and the rate must decrease at the end to get a good fit.

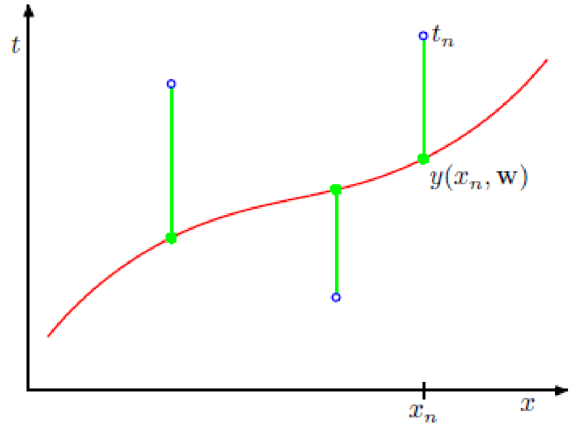
Improvement in Regression Model

A Regression Problem



Training set $X = \langle x_1, \ldots, x_N \rangle$ with targets $\mathbf{t} = (t_1, \ldots, t_N)^T$.
t_n is generated from x_n plus some Gaussian noise.
Goal: Predict value $\hat{t}$ for some new input $\hat{x}$.

The Model



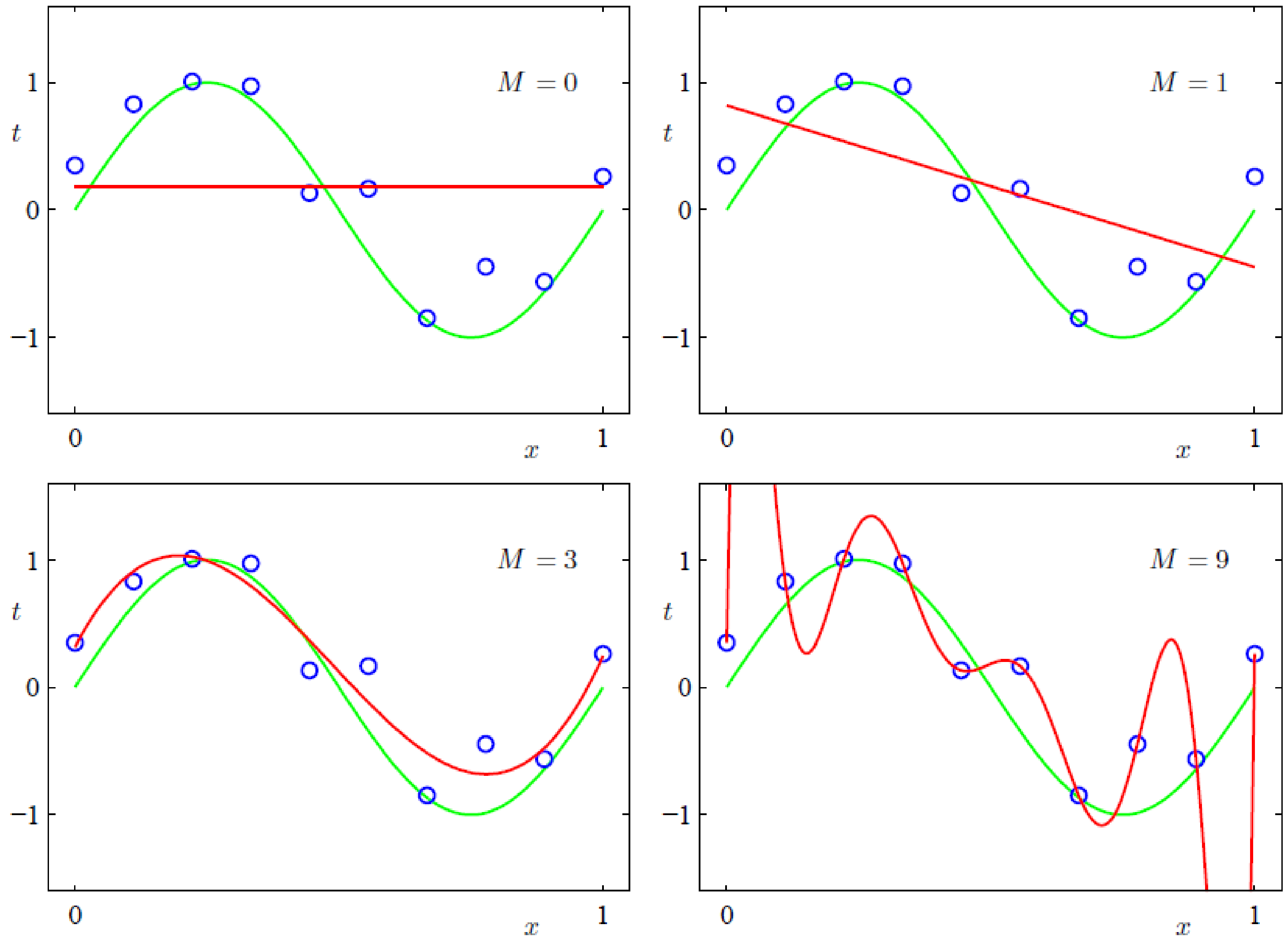
$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j.$$

The polynomial coefficients $w_0, \ldots, w_M$ are collectively denoted by the parameter vector $\mathbf{w}$.
We fit the model to the data by minimizing an **error function** that measures the misfit between $y(x, \mathbf{w})$ and the training data.

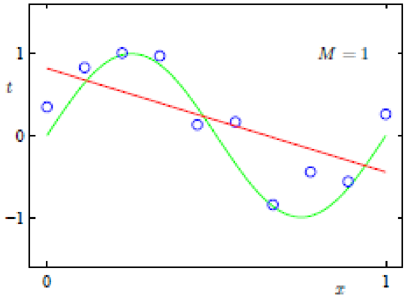
$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2.$$

We then choose parameter vector $\mathbf{w}^*$ such that $E(\mathbf{w}^*)$ is minimal.

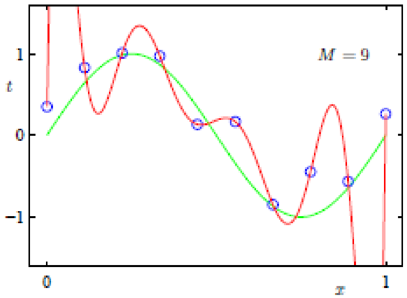
Problem: How to choose the order of the polynomial M?



Problem: How to choose the order of the polynomial M?



M is too small. The model is not expressive enough, **underfitting**.

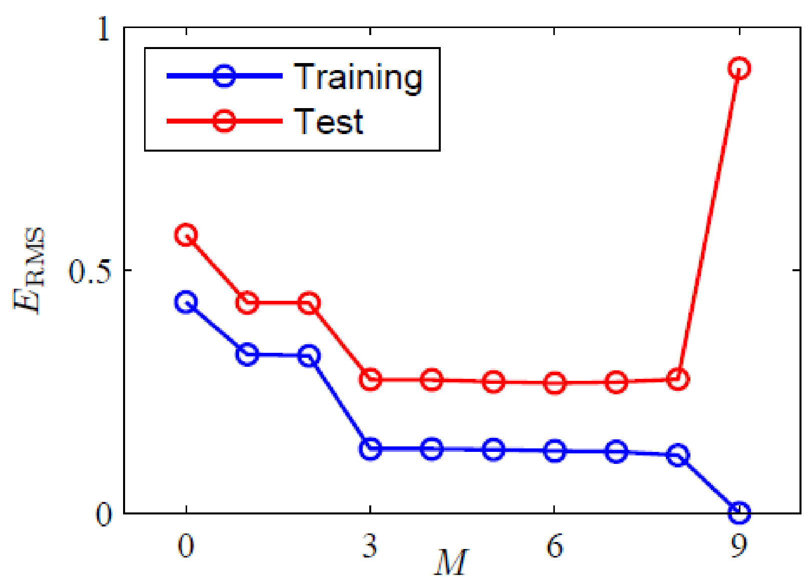


M is too large. The model captures the noise in the data, **overfitting**.

Test error vs. model complexity

One can obtain a quantitative estimate of the generalization with parameter vector $\mathbf{w}^*$ by considering a separate test set.
The root mean square (RMS) error for N examples is

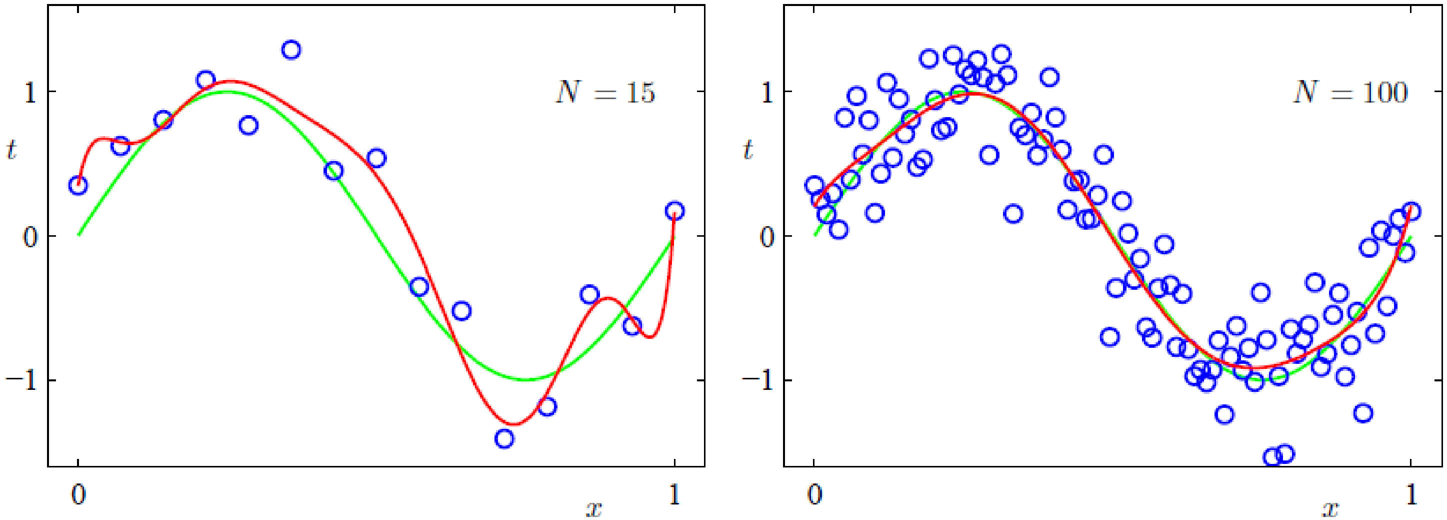
$$E_{RMS} = \sqrt{2E(w^*)/N} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2}$$



Complex models are finely tuned to the data

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
w_0^*	0.19	0.82	0.31	0.35
w_1^*		-1.27	7.99	232.37
w_2^*			-25.43	-5321.83
w_3^*			17.37	48568.32
w_4^*				-231639.30
w_5^*				640042.26
w_6^*				-1061800.52
w_7^*				1042400.18
w_8^*				-557682.99
w_9^*				125201.43

Size of the Training Set



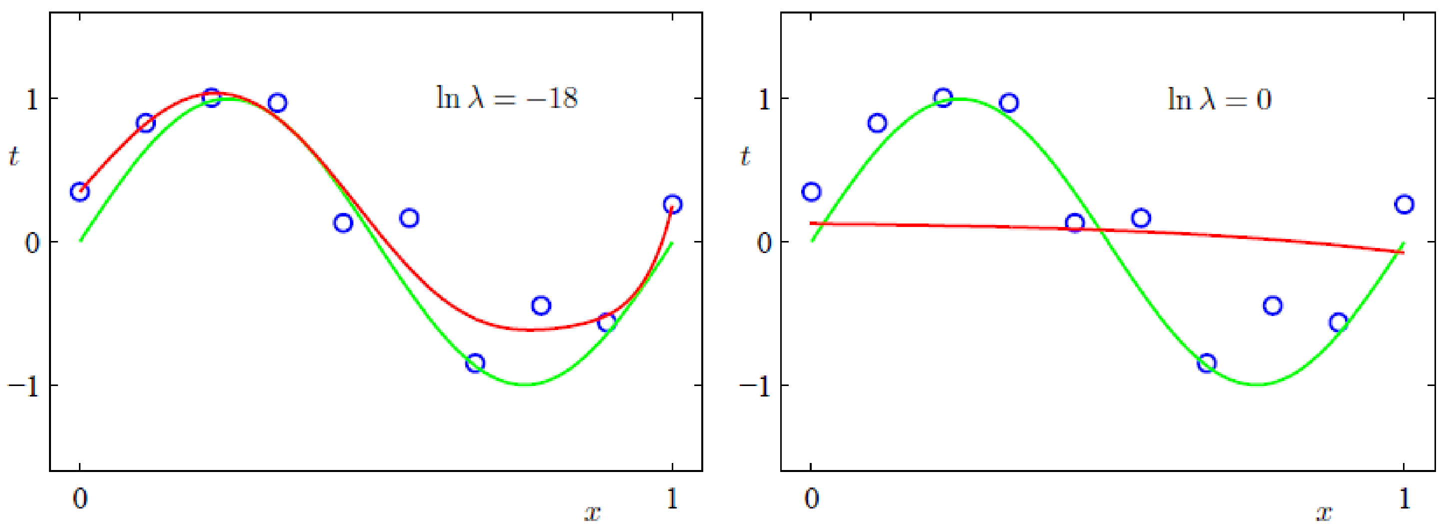
$M = 9$

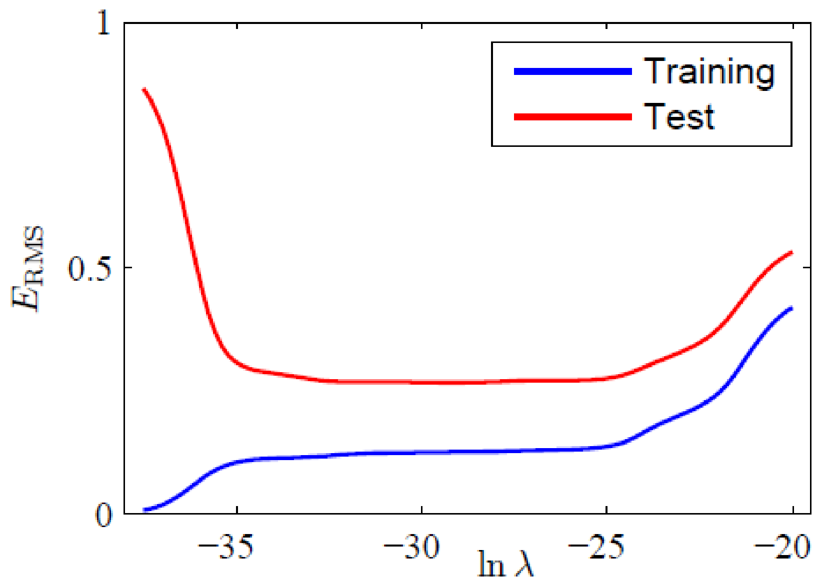
The larger the training set, the more complex models can be fitted.

Regularization

One would like to choose the complexity of the model according to the complexity of the problem being solved.
We introduce a penalty term for large parameters:

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





λ controls the effective complexity of the model.

Practical method: Hold back data, called **validation set**, to optimize model complexity (that is, M, or λ).

Training set: To optimize model parameters

Validation set: To optimize hyper parameters, find model complexity

Test set: To estimate the true error

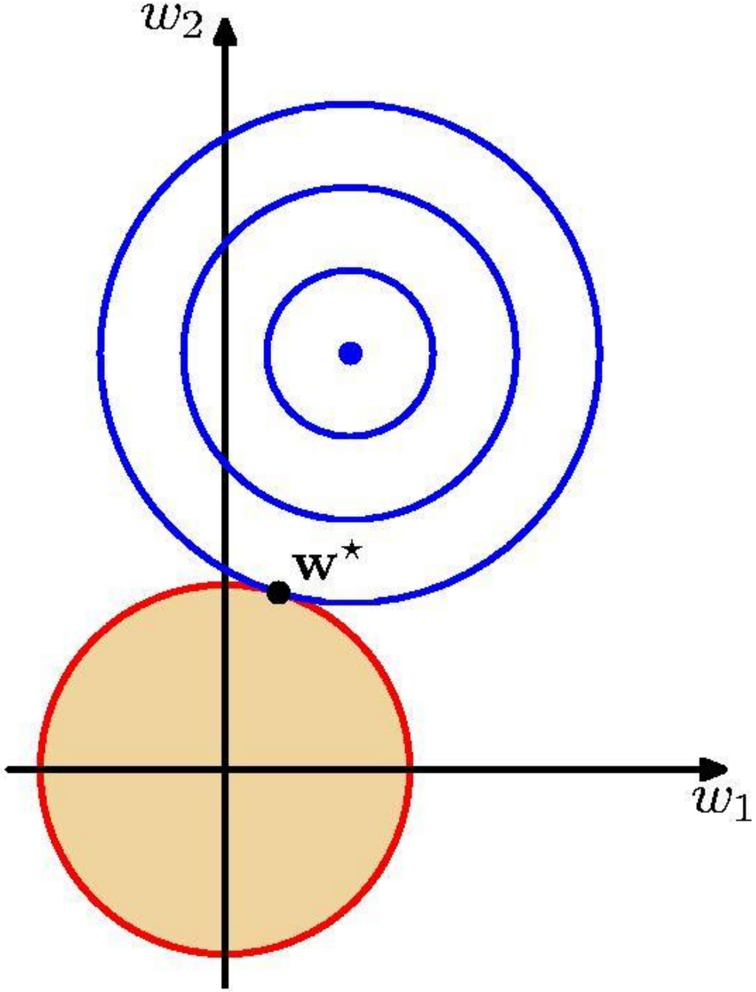
Regularized Least Squares

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

The penalty on the squared weights is mathematically compatible with the squared error function, so we get a nice closed form for the optimal weights with this regularizer:

$$\mathbf{w}^* = (\lambda \underbrace{\mathbf{I}}_{\text{identity matrix}} + \mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$$

A Picture of the Effect of the Regularizer



- The overall cost function is the sum of two parabolic bowls.
- The sum is also a parabolic bowl.
- The combined minimum lies on the line between the minimum of the squared error and the origin.
- The regularizer just shrinks the weights.