ML Algorithm Review

Jie Tang
Department of Computer Science & Technology
Tsinghua University

The State of Machine Learning

Classification models

Decision tree 
Bayesian classifier
Perceptron
Neural networks

Maximal Margin

SVM (Vapnik) 



Maximal margin sequential learning

M3N network (Ben Taskar) 

Deep learning

Deep belief networks




Topic models

PLSI, LDA, etc.




Sequential learning

HMM 

Sequential learning

MEMM, CRF, voted perceptron



Graphical models

Factor graph, Exponential model 



For more from http://aminer.org/

Basic ML Algorithm Review

- Basic algorithms review
  - Data representation
  - Supervised learning
    - Evaluation
    - Regression, Perceptron, Bayesian classificaiton, Decision tree, etc.
  - Unsupervised learning
    - K-means, K-medoid
  - Semi-supervised learning
    - Learning with labeled and unlabeled data
    - Active learning, Transfer learning

The first example—KNN

K-Nearest Neighbor Learning

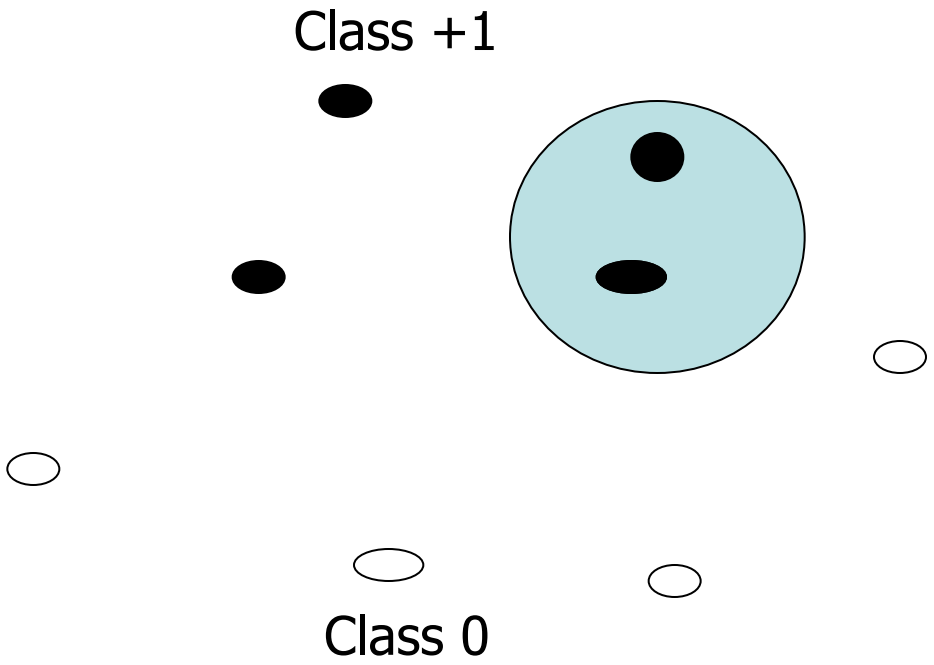
k-NN is a type of instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until classification

- Given a training set $S=\{(x_1, y_1),(x_2, y_2),\ldots,(x_N, y_N)\}$, and $x_i \in X=R^m$, $i=1,2,\ldots,N$, classify objects based on closest training examples in the feature space.

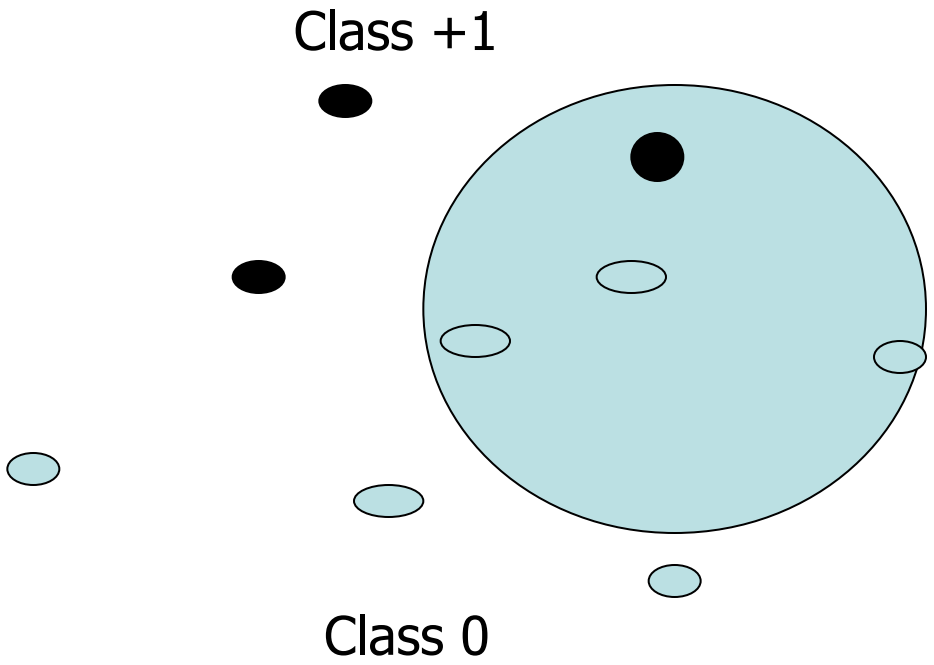
- Properties of kNN
  - No explicit hypothesis
  - No training stage
  - Distance function

$$d(x_i,x_j)=\sqrt{(|x_{i1}-x_{j1}|^2+|x_{i2}-x_{j2}|^2+\ldots+|x_{im}-x_{jm}|^2)}$$

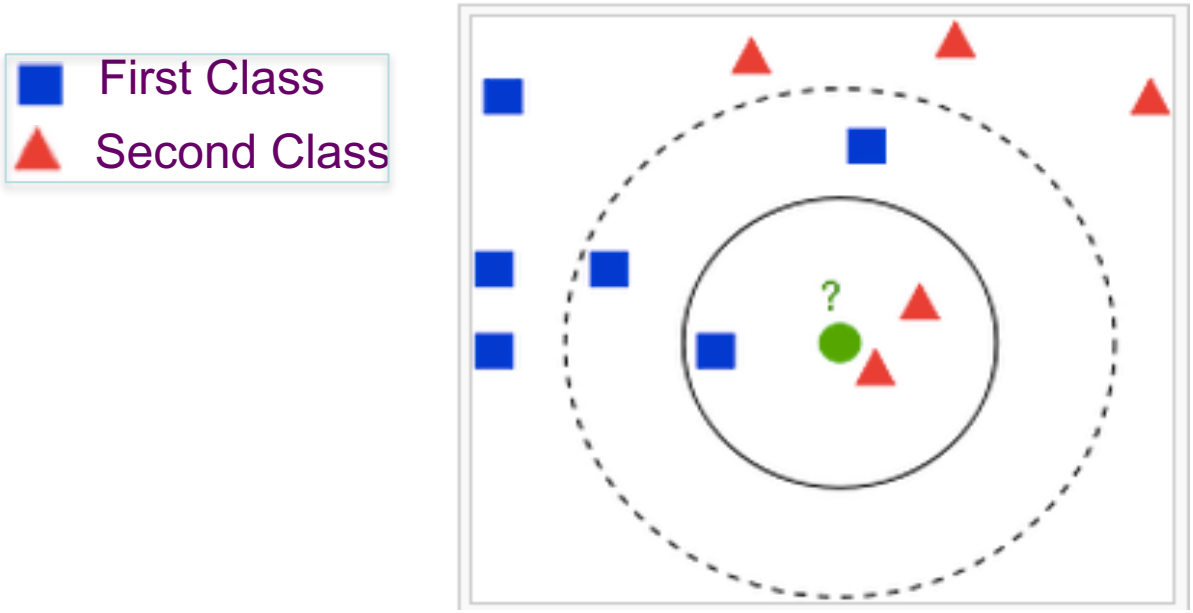
1-Nearest Neighbor



3-Nearest Neighbor



Properties



The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles.

- If k = 3 (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle.
- If k = 5 (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

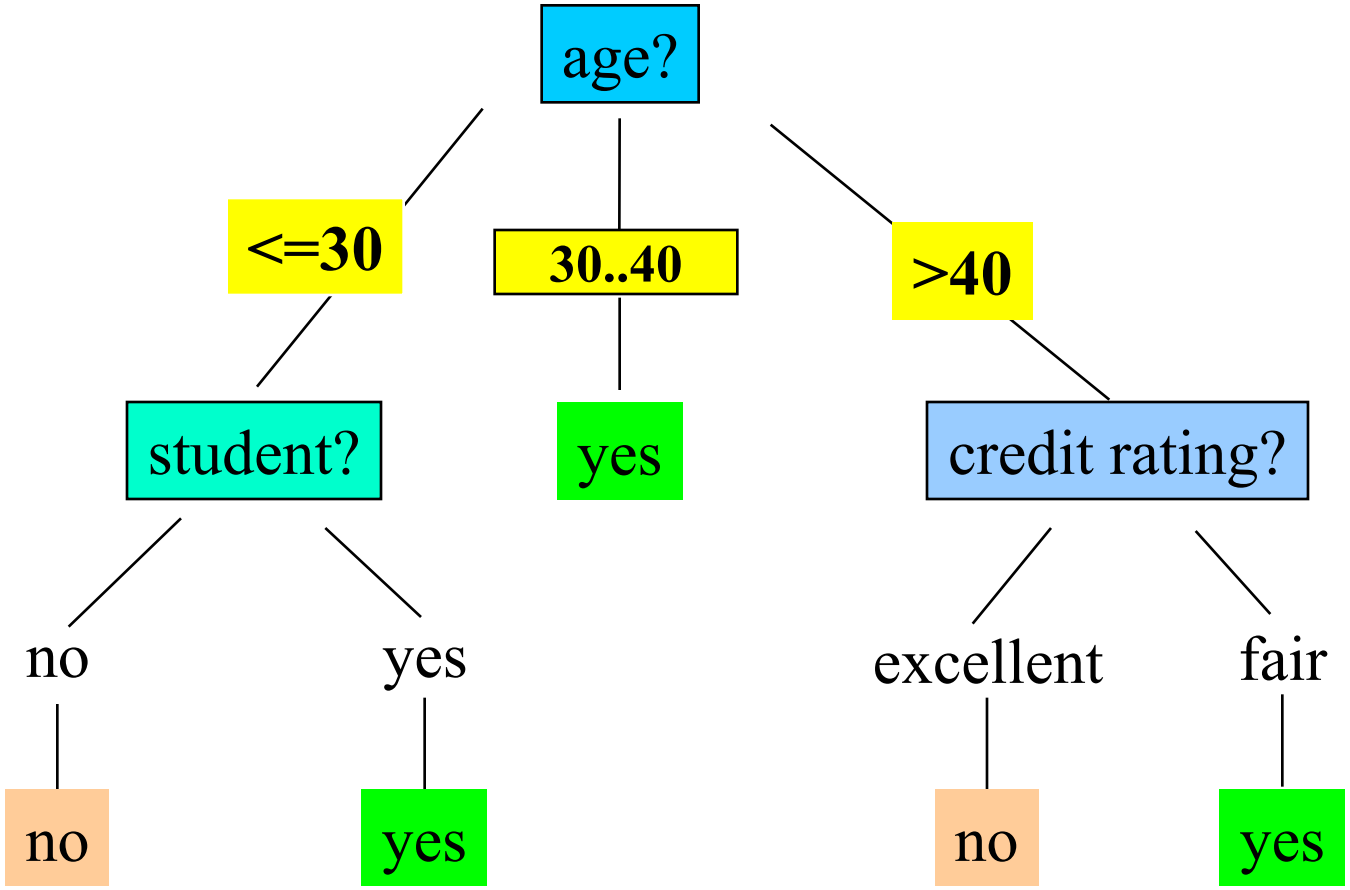
Another example—Decision Tree

Training Dataset

This follows an example from Quinlan's ID3

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Output: A Decision Tree for "*buys_computer*"



Algorithm for Decision Tree Induction

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Select the attribute with the highest information gain
- S contains s_i tuples of class y_i for i = {1, …, m}
- information measures info required to classify any arbitrary tuple

$$I(s_1,s_2,\ldots,s_m) = -\sum_{i=1}^{m} \frac{s_i}{s} log_2 \frac{s_i}{s}$$

- entropy of attribute A with values $\{a_1,a_2,\ldots,a_v\}$

$$E(A) = \sum_{j=1}^{v} \frac{s_{1j}+\ldots+s_{mj}}{s} I(s_{1j},\ldots,s_{mj})$$

- information gained by branching on attribute A

$$Gain(A) = I(s_1, s_2,\ldots,s_m) - E(A)$$

Attribute Selection by Information Gain Computation

- Class P: buys_computer = "yes"
- Class N: buys_computer = "no"
- I(p, n) = I(9, 5) =0.940
- Compute the entropy for *age*:

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
30…40	4	0	0
>40	3	2	0.971

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$$E(age) = \frac{5}{14} I(2,3) + \frac{4}{14} I(4,0) + \frac{5}{14} I(3,2) = 0.694$$

$\frac{5}{14} I(2,3)$ means "age <=30" has 5 out of 14 samples, with 2 yes'es and 3 no's. Hence

$$Gain(age) = I(p,n) - E(age) = 0.246$$

Similarly,

$$Gain(income) = 0.029$$

$$Gain(student) = 0.151$$

$$Gain(credit\_rating) = 0.048$$

Extracting Classification Rules from Trees

- Represent the knowledge in the form of IF-THEN rules
- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction
- The leaf node holds the class prediction
- Rules are easier for humans to understand
- Example

IF *age* = "<=30" AND *student* = "*no*" THEN *buys_computer* = "*no*"

IF *age* = "<=30" AND *student* = "*yes*" THEN *buys_computer* = "*yes*"

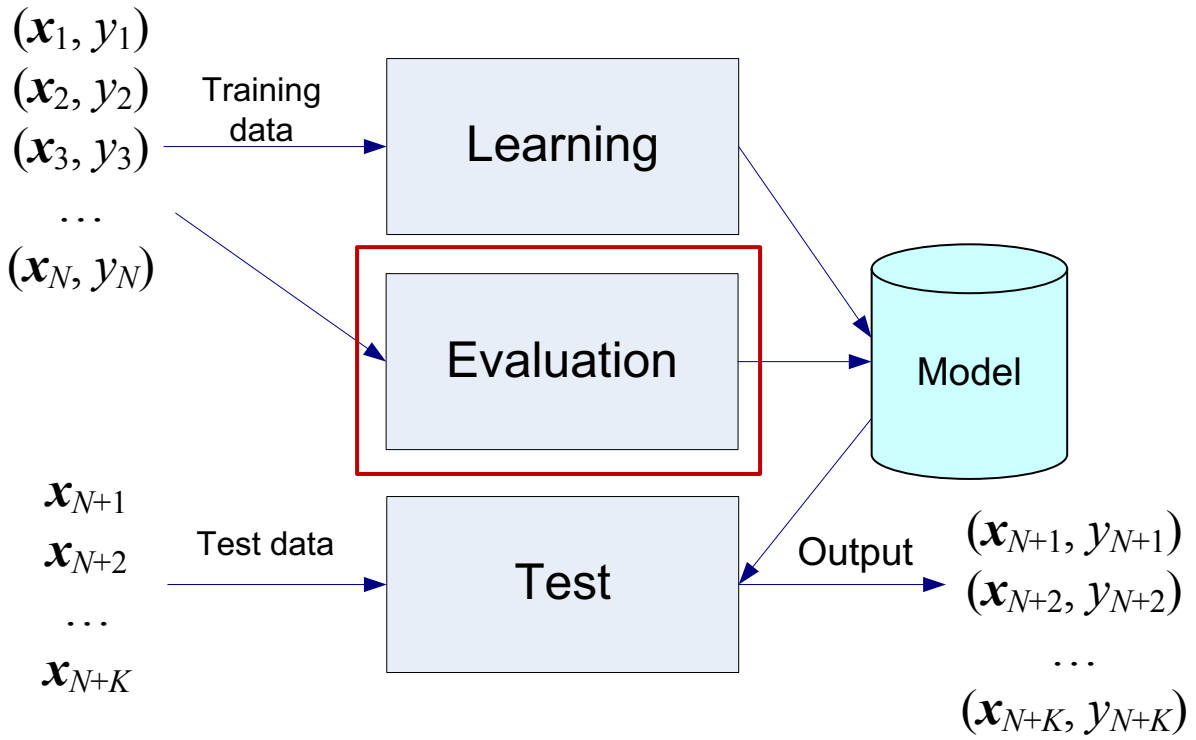
IF *age* = "31…40" THEN *buys_computer* = "*yes*"

IF *age* = ">40" AND *credit_rating* = "*excellent*" THEN *buys_computer* = "*yes*"

IF *age* = "<=30" AND *credit_rating* = "*fair*" THEN *buys_computer* = "*no*"

Evaluation—Supervised Learning

Supervised Learning



* x_i is a vector and x_{ik} represents the k-th feature of example x_i

Evaluation

- Accuracy on test set
  - The rate of correct classification on the testing set. E.g., if 90 are classified correctly out of the 100 testing cases, accuracy is 90%.
- Error Rate on test set
  - The percentage of wrong predictions on test set
- Confusion Matrix
  - For binary class values, "yes" and "no", a matrix showing true positive, true negative, false positive and false negative rates

		Predicted class	
		Yes	No
Actual class	Yes	True positive (TP)	False negative (FN)
	No	False positive (FP)	True negative (TN)

Type I error

Type II error

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

$$\text{F1} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

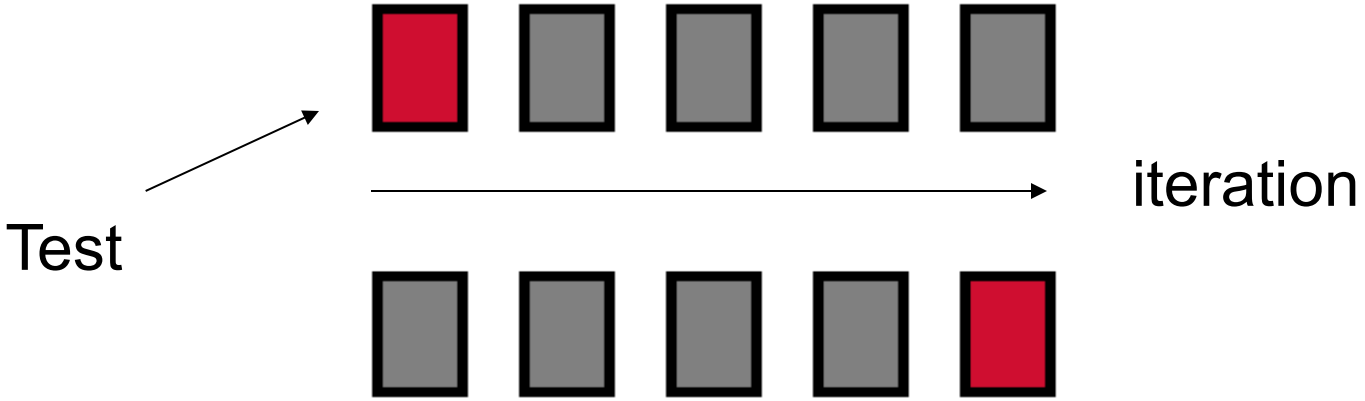
Evaluation Techniques

- *Holdout*: the training set/testing set.
  - Good for a large set of data.
- *k-fold Cross-validation (交叉验证)*:
  - Divide the data set into k sub-samples.
  - In each run, use one distinct sub-sample as testing set and the remaining k-1 sub-samples as training set.
  - Evaluate the method using the average of the k runs.
- This method reduces the randomness of training set/testing set.

Cross Validation: Holdout Method

Break up data into groups of the same size

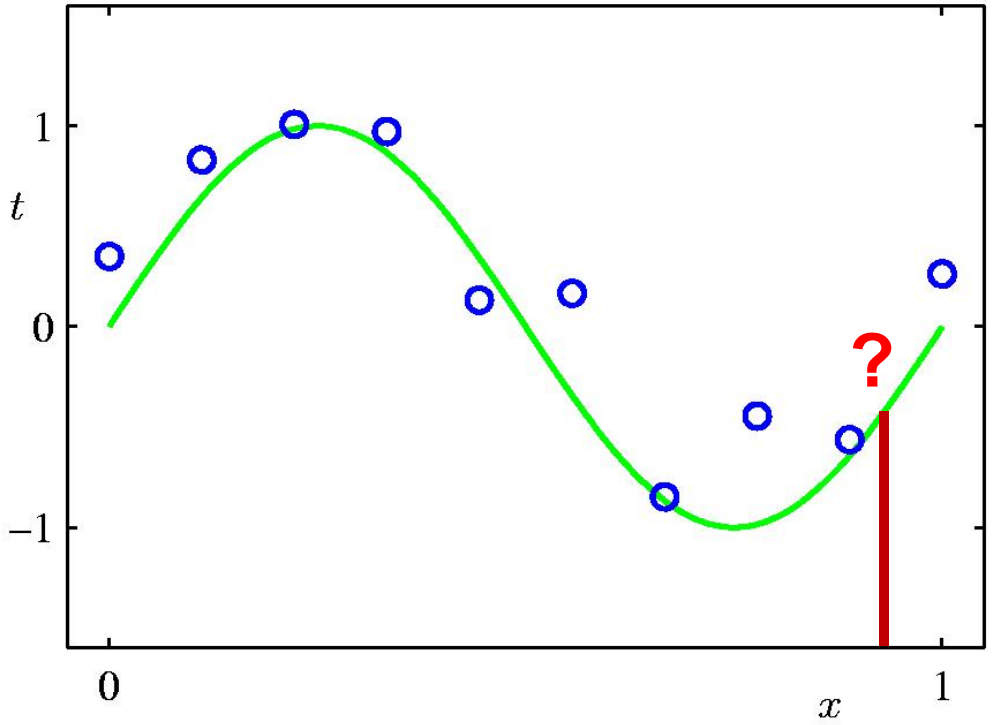
Hold aside one group for testing and use the rest to build model



Large *k* results in a small bias but large variance

A general case

Polynomial Curve Fitting



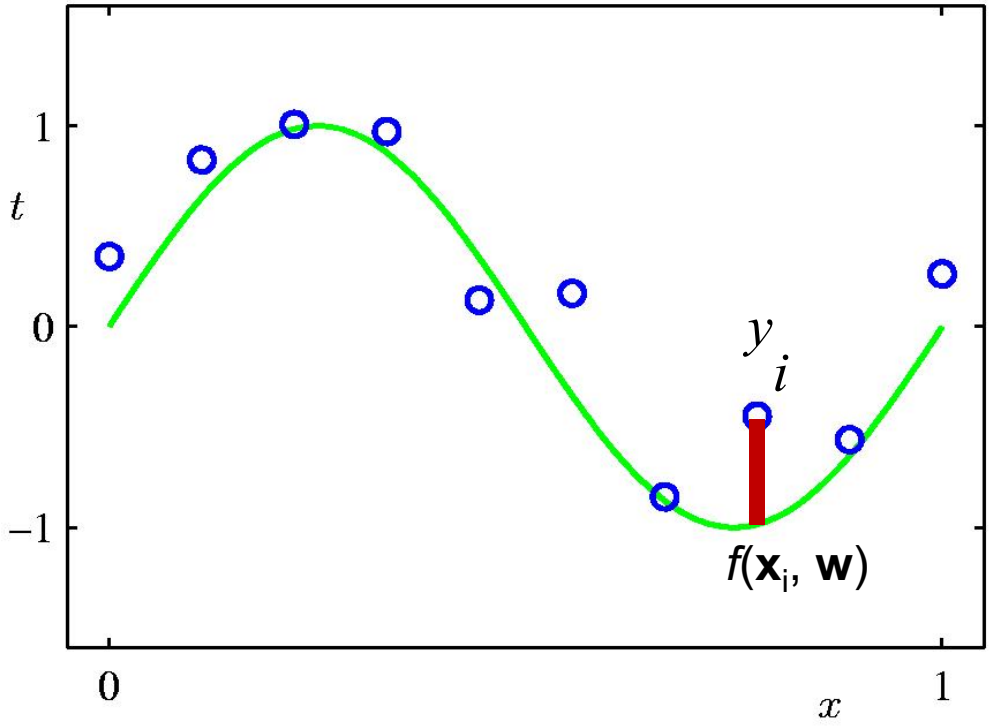
As the first step, we need to decide how we're going to represent the function *f*.
One example: polynomial curve fitting

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

Now, given a training set, how do we pick, or learn, the parameters **w**?

Notations:
$w_0 + w_1 x_1$
$w^T \boldsymbol{x}$

Least Squares Loss Function



$$L(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$$

Learning by Gradient Descent

How to choose **w** in order to minimize $L(\mathbf{w})$?

- General idea is to start with some "initial guess" for **w**, and then repeatedly changes **w** to make $L(\mathbf{w})$ smaller, until we converge to a value of **w** that minimize $L(\mathbf{w})$.
- Gradient descent is a natural search algorithm that update **w** in the direction of steepest decrease of L:

$$w_k = w_k - \Delta \frac{\partial L(\mathbf{w})}{\partial w_k}$$

where △ is the learning rate

Now let us calculate the partial derivative.
First consider one training instance (x, y), so the sum in L can be ignored:

$$\begin{aligned}\frac{\partial}{\partial w_k} L(\mathbf{w}) &= \frac{\partial}{\partial w_k} \frac{1}{2} (f(\mathbf{x},\mathbf{w}) - y)^2 \\ &= (f(\mathbf{x},\mathbf{w}) - y) \frac{\partial}{\partial w_k} \left(\sum_{k=0}^{m} w_k x_k - y \right) \\ &= (f(\mathbf{x},\mathbf{w}) - y) x_k\end{aligned}$$

$$w_k = w_k - \Delta (f(\mathbf{x},\mathbf{w}) - y) x_k$$

The rule is called LMS (least mean squares) update rule, aka Widrow-Hoff learning rule.

Intuition: The update is proportional to the error term ($f(x, \mathbf{w}) - y$). Thus for the training examples with prediction score close to the actual value y, there is little need to change the parameters; in contrast, a larger change to the parameters will be made.

Learning by Gradient Descent (cont.)

Then consider a training data set rather than only one example by two ways:

- Batch gradient descent

Scan through the entire training set before taking a single step of update

Repeat until convergence {

$$w_k = w_k - \alpha_t \sum_i (f(\mathbf{x}_i, \mathbf{w}) - y_i)\, x_{ik}$$

}

- Stochastic gradient descent

Start making progress right away, and continues to make progress with each example it looks at.

Much faster than batch gradient descent.

The parameters will keep oscillating around the minimum of *L*, but in practice most of the values near the minimum will be reasonably good approximations to the true minimum.

Repeat until convergence {

Random select a sample x_i {

$$w_k = w_k - \alpha_t (f(x_i, \mathbf{w}) - y_i)\, x_{ik}$$

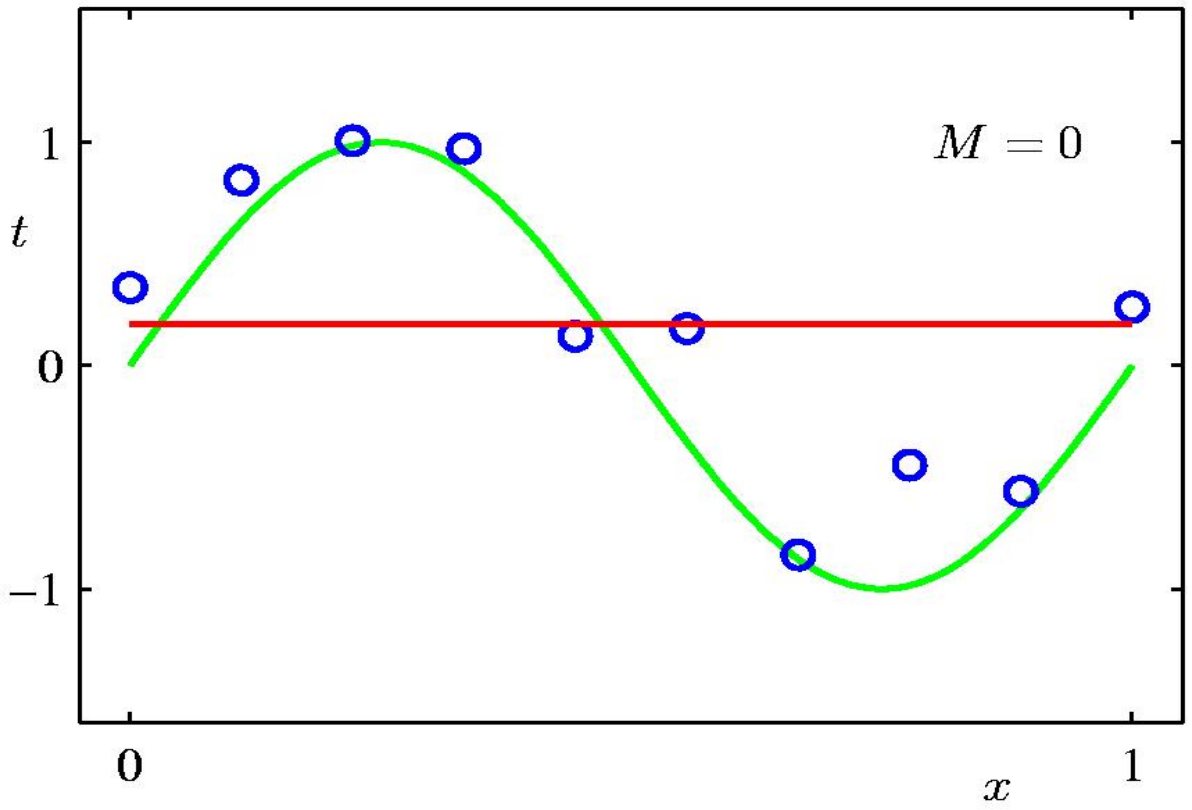
}

}

Particularly when the training set is large, stochastic gradient descent is often preferred.

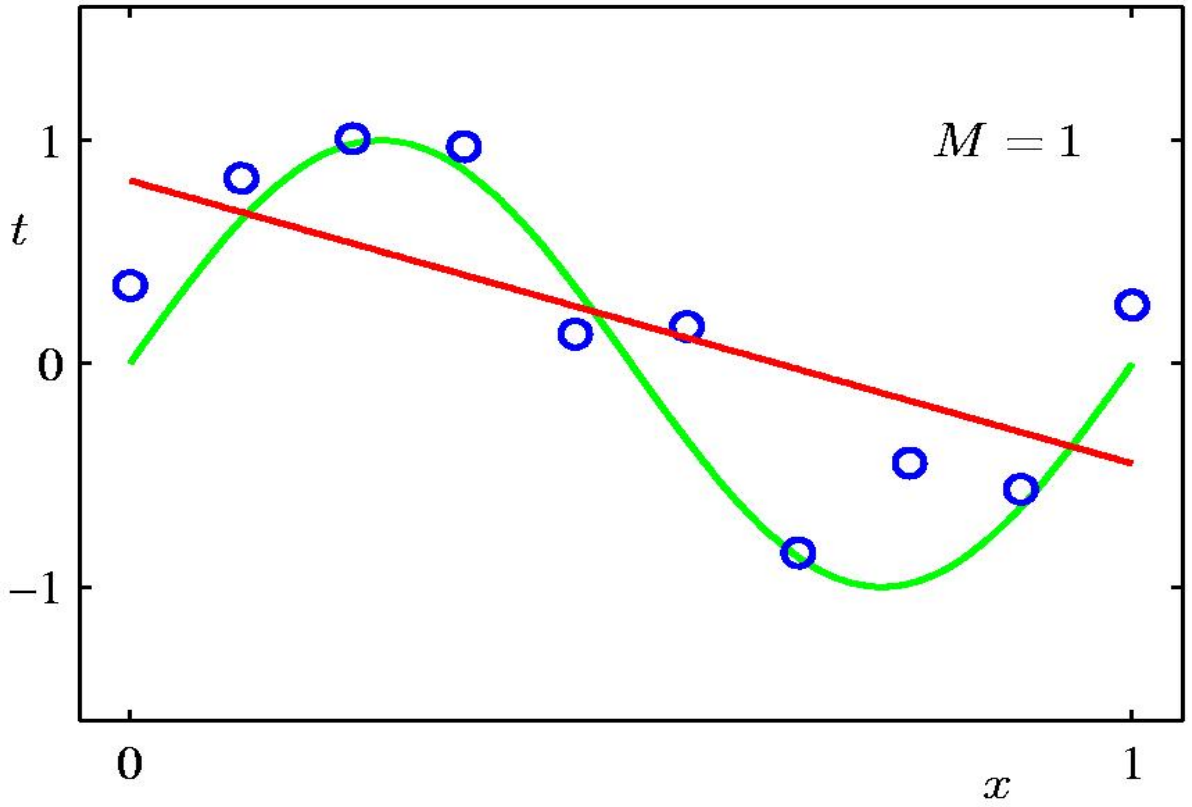
The learning rate α_t decreases with the iteration time t.

0^{th} Order Polynomial

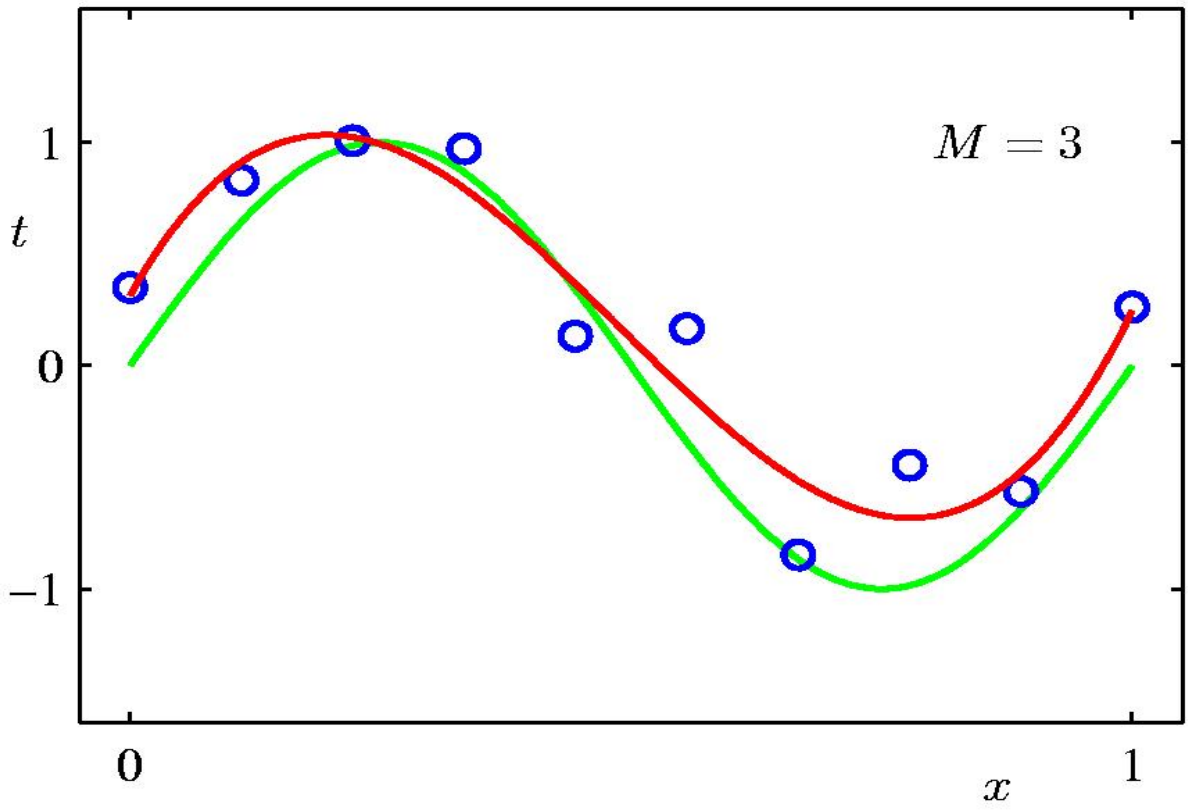


* A number of the following pages are from Bishop's slides

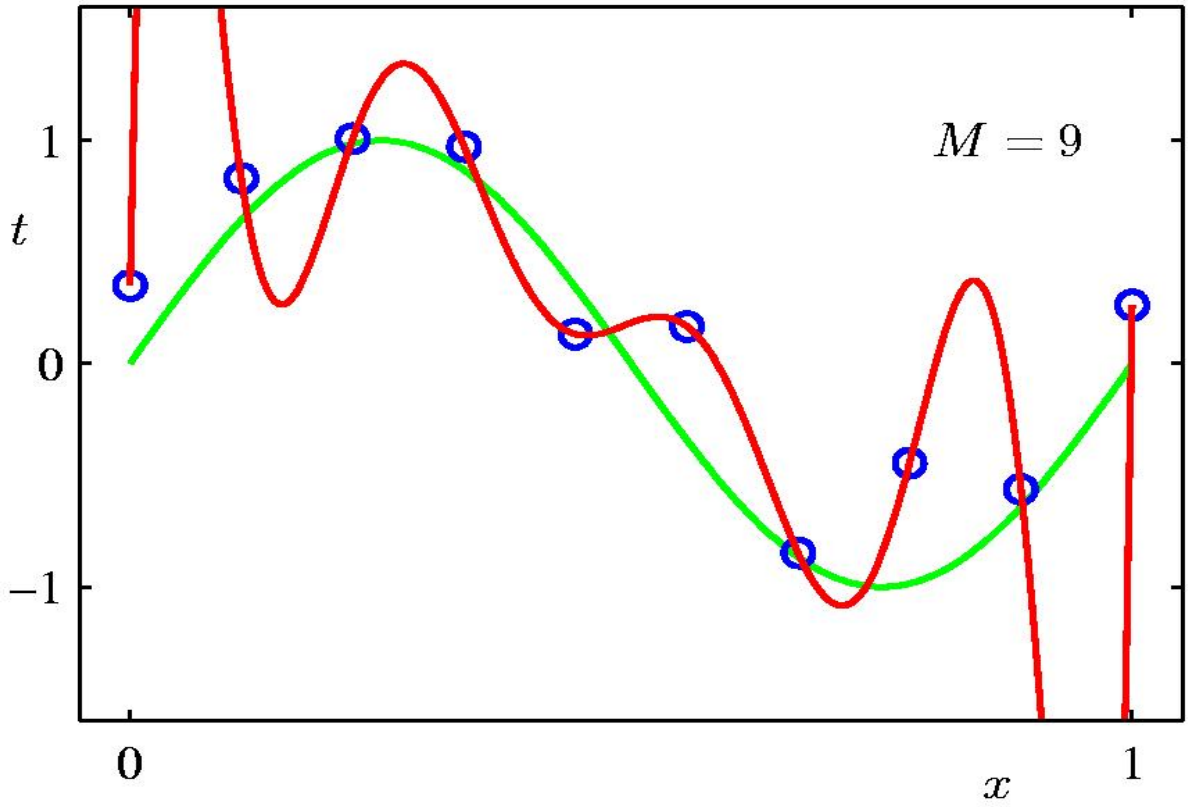
1st Order Polynomial



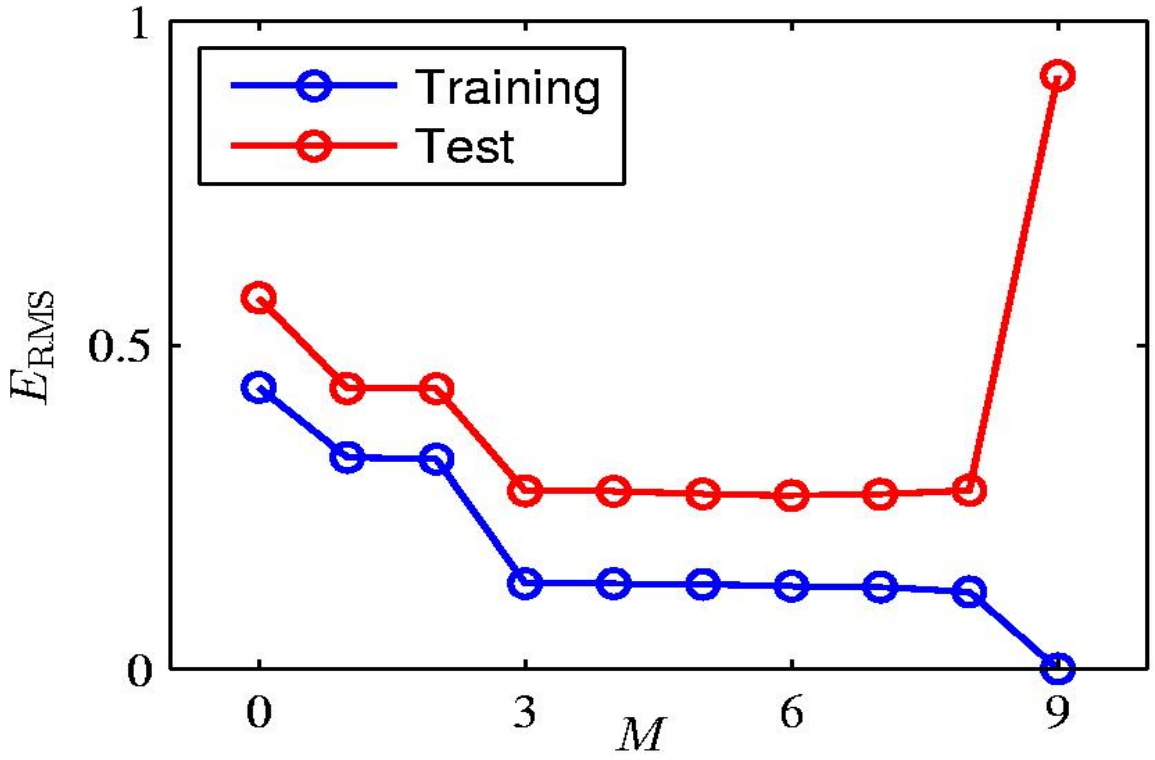
3rd Order Polynomial



9th Order Polynomial



Over-fitting



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

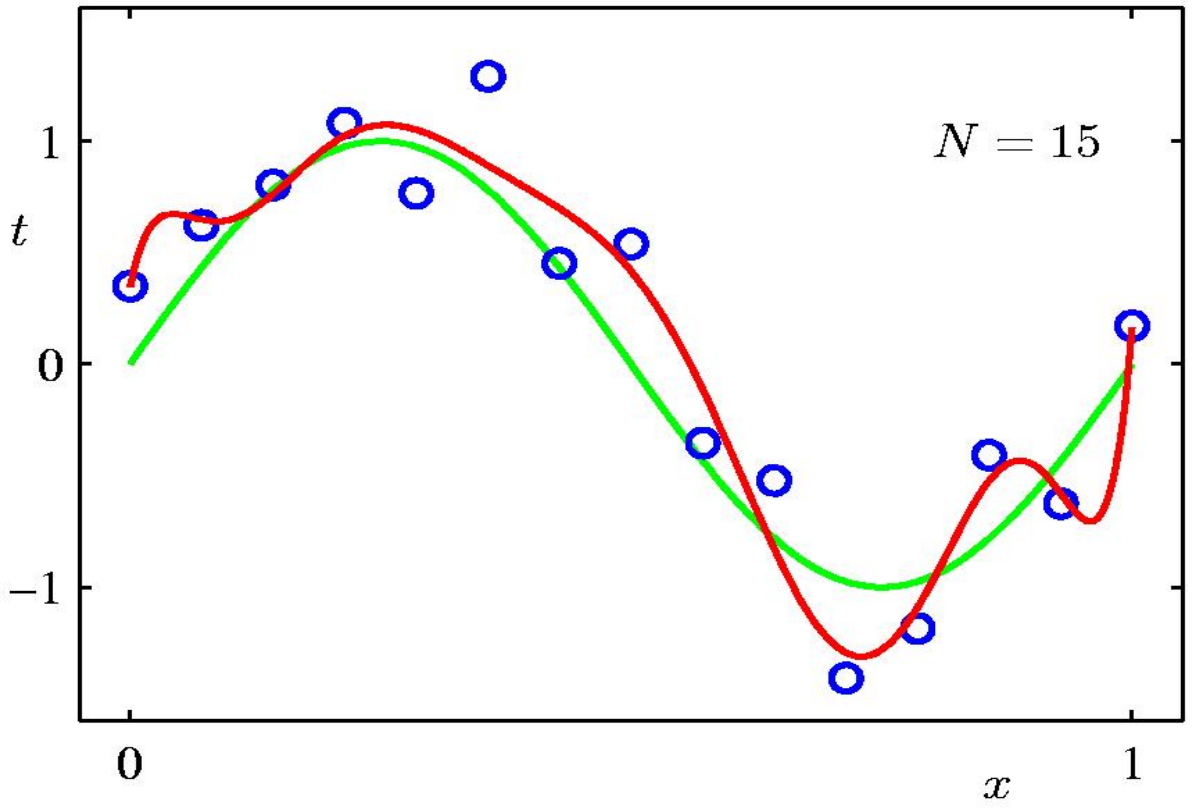
Polynomial Coefficients

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

Solution 1: Increasing Data Volume

$$N = 15$$

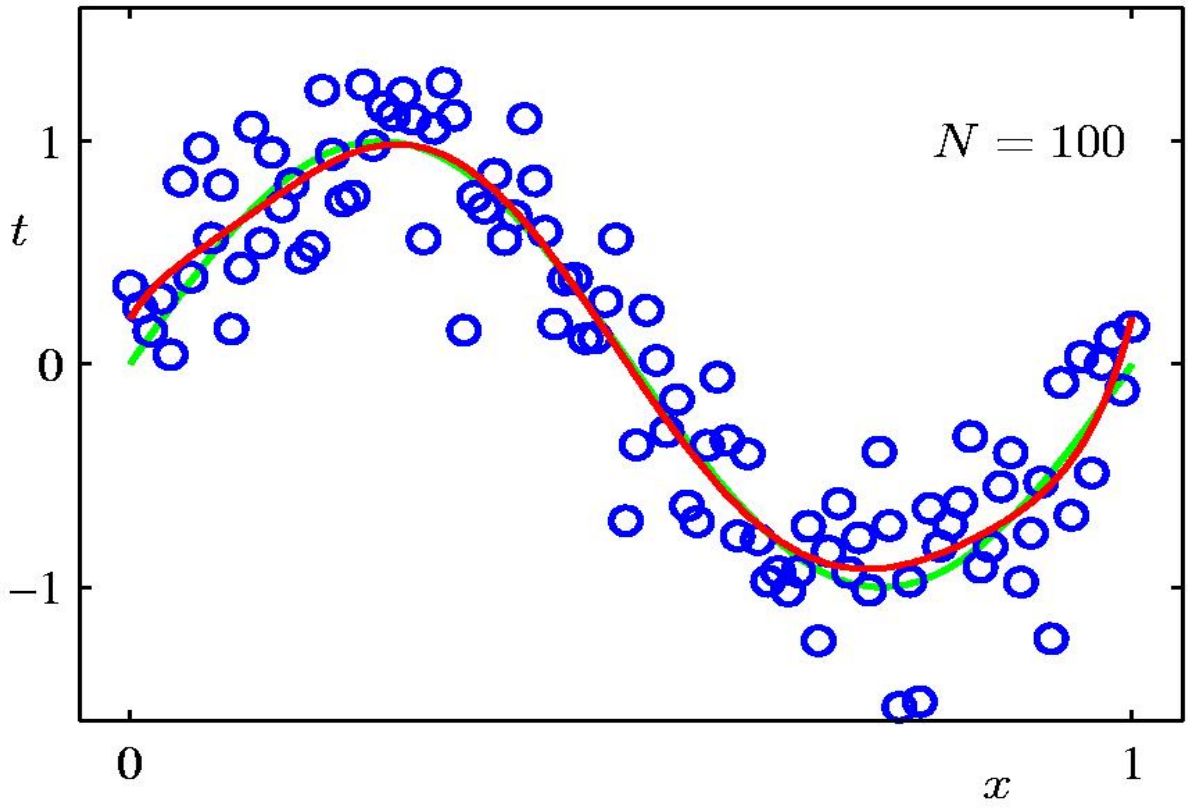
9th Order Polynomial



Data Set Size

$$N = 100$$

9th Order Polynomial



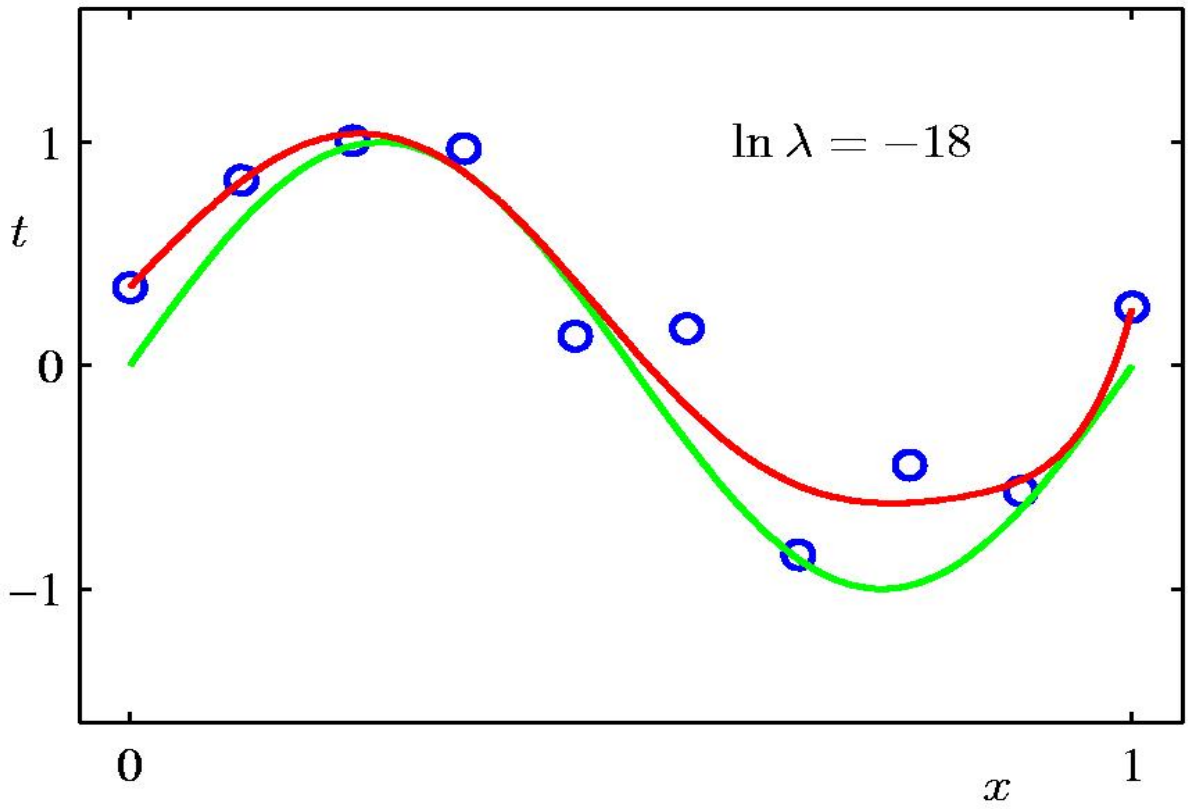
Solution 2: Regularization

Penalize large coefficient values

$$L(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(f(\mathbf{x}_i, \mathbf{w}) - y_i)^2 + \frac{\lambda}{2}||\mathbf{w}||^2$$

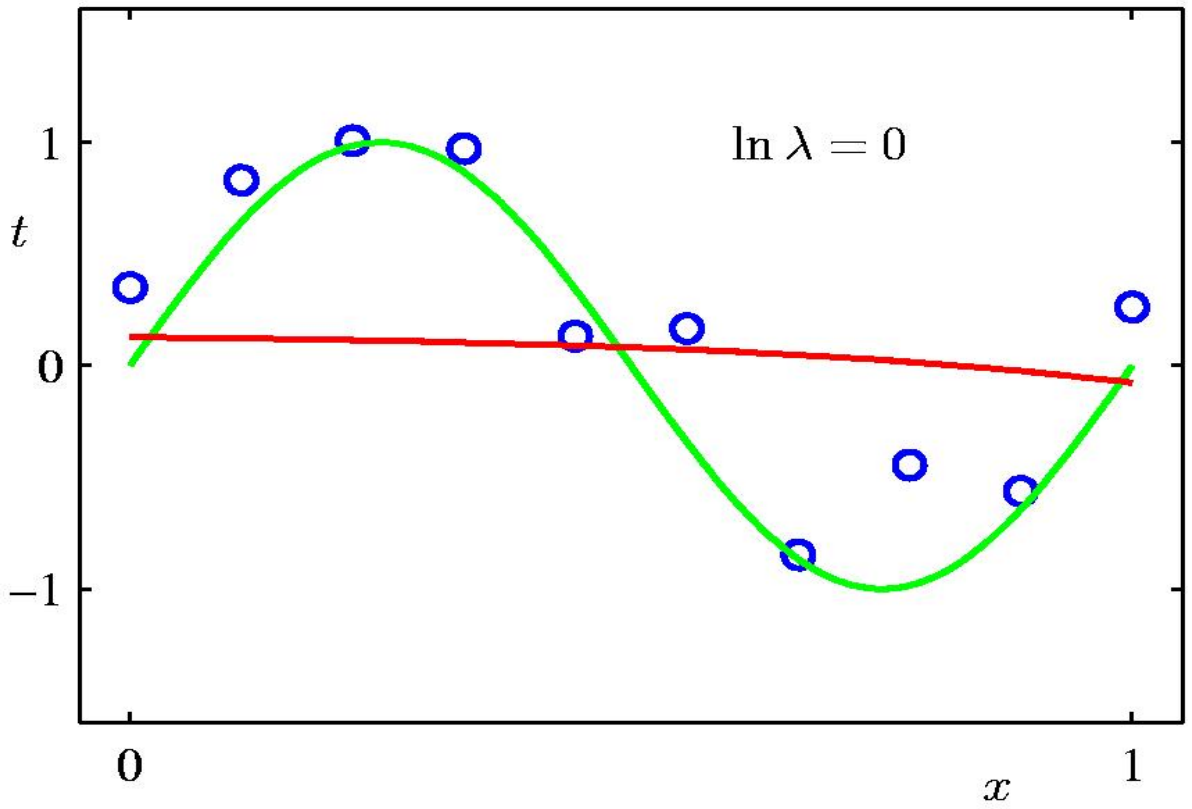
Regularization

$$\ln \lambda = -18$$

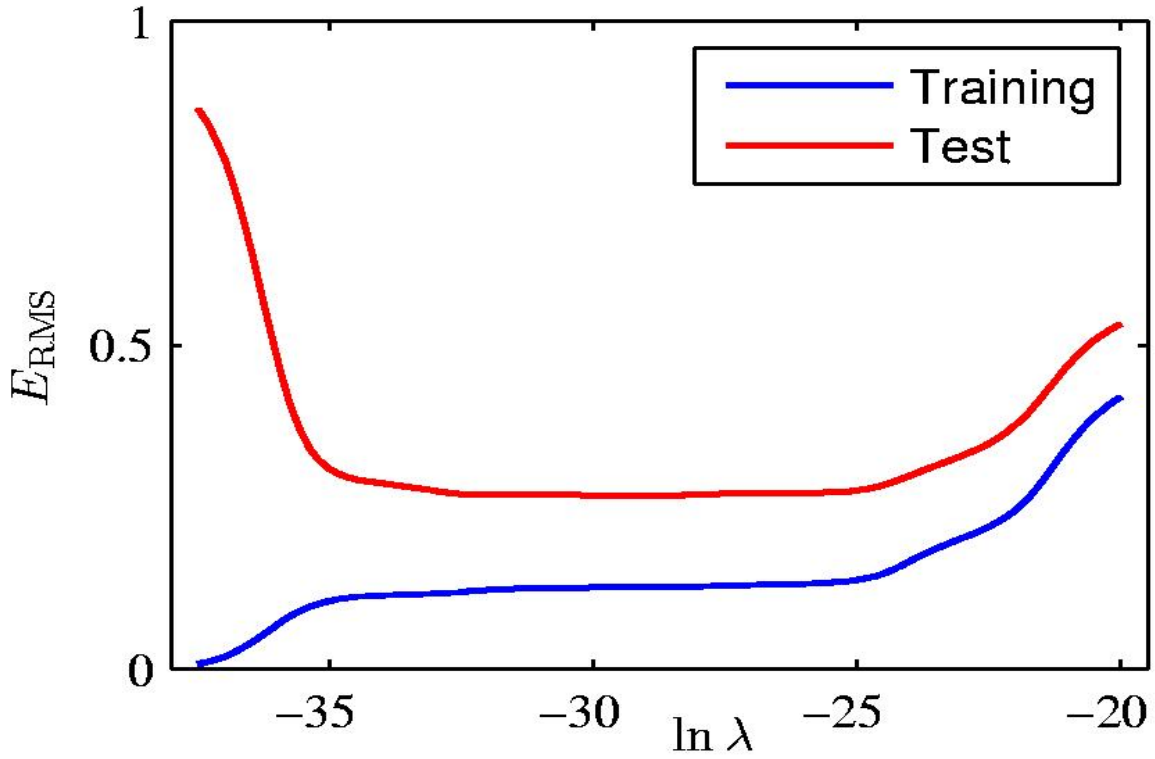


Regularization

$$\ln \lambda = 0$$



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

Classification—Perceptron

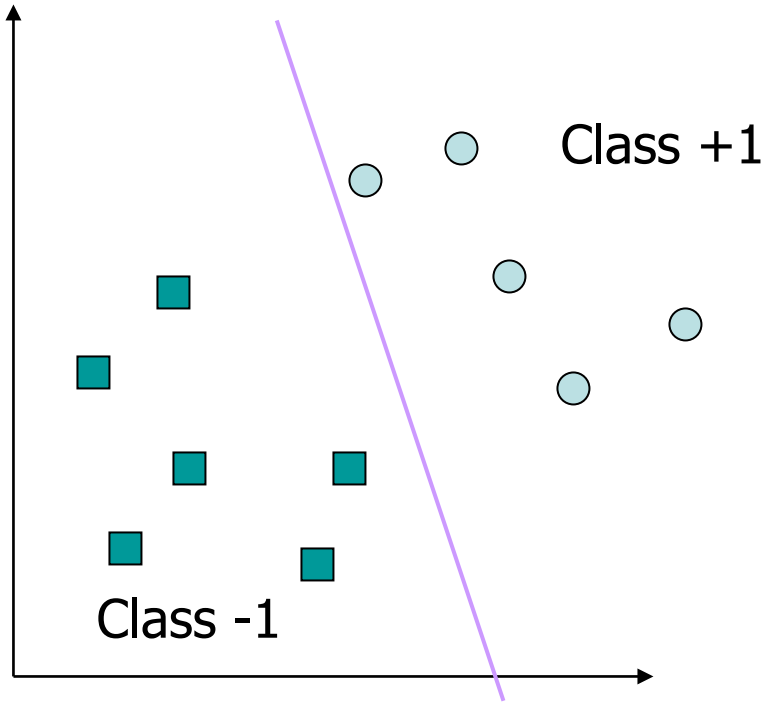
Perceptron—a linear model

- Consider a two-class, linearly separable classification problem
- We are looking for a linear function

$$y_i = \text{sign}(\mathbf{w} \cdot \mathbf{x}_i)$$

i.e.,

$$f(x, \mathbf{w}) = \begin{cases} 1, & \mathbf{w}^T \mathbf{x} \geq 0 \\ -1, & \mathbf{w}^T \mathbf{x} < 0 \end{cases}$$



Rosenblatt's Perceptron

- The Rosenblatt's perceptron algorithm considers each training point in turn, adjusting the parameters to correct any mistakes

Initialize: $\mathbf{w}=0$;
Repeat until convergence:
  for $i=1,\ldots,m$
    if $y_i\,(\mathbf{w}\cdot x_i) \le 0$ *then*
      $\mathbf{w} \leftarrow \mathbf{w} + y_i\,x_i$

$$f(x,\mathbf{w}) = \begin{cases} 1, & \mathbf{w}^T\mathbf{x} \ge 0 \\ -1, & \mathbf{w}^T\mathbf{x} < 0 \end{cases}$$

- The algorithm will converge (no mistakes) if the training points are *linearly separable through origin;* otherwise it won't converge

Perceptron: property on update

- If we make a mistake on x_i

$$y_i(\mathbf{x}_i \cdot \mathbf{w}) \leq 0$$

- After the update, we have

$$\mathbf{w}' = \mathbf{w} + y_i \mathbf{x}_i$$

$$y_i(\mathbf{w}' \cdot \mathbf{x}_i) = y_i([\mathbf{w} + y_i \mathbf{x}_i] \cdot \mathbf{x}_i)$$

$$= y_i \mathbf{w} \mathbf{x}_i + y_i^2 \left\| \mathbf{x}_i \right\|^2$$

$$= y_i \mathbf{w} \mathbf{x}_i + \left\| \mathbf{x}_i \right\|^2$$

- So that $y_i(\mathbf{x}_i \cdot \mathbf{w})$ increases based on update

Rosenblatt's Algorithm

Input: $S = \langle(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)\rangle \quad \vec{x}_i \in \Re^N \quad y_i \in \{1, -1\}$ (linear separable)

- $w_0 \leftarrow 0; b_0 \leftarrow 0; k \leftarrow 0$
- $R = max_i \left\|\vec{x}_i\right\|$
- repeat
  - for i=1 to n
    - if $y_i(\vec{w}_k \cdot \vec{x}_i + b_k) \leq 0$
      - $\vec{w}_{k+1} \leftarrow \vec{w}_k + \eta y_i \vec{x}_i$
      - $b_{k+1} \leftarrow b_k + \eta y_i R^2$
      - $k \leftarrow k = 1$
    - endif
  - endfor
- until no mistakes made in the for loop
- return $(\vec{w}_k, b_k)$

Update by minimizing LMS

- Still consider minimizing the least squares loss function

$$L(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(f(x_i, \mathbf{w}) - y_i)^2$$

- Then we obtain the update rule

$$w_j = w_j + \Delta(y - f(x, \mathbf{w}))x_j$$

Classification—Logistic Regression

Logistic Regression

Classification is like the regression problem, except that the values y take on (a small number of) discrete values. Let us first consider the binary case (binary logistic regression):

$$f(x,\mathbf{w}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$

$$g(z) = \frac{1}{1+e^{-z}}$$ Logistic or sigmoid function

How to fit **w** for logistic regression model?

$$P(y=1 \mid \mathbf{x};\mathbf{w}) = f(\mathbf{x},\mathbf{w})$$
$$P(y=0 \mid \mathbf{x};\mathbf{w}) = 1 - f(\mathbf{x},\mathbf{w})$$

i.e.,

$$p(y \mid \mathbf{x};\mathbf{w}) = f(\mathbf{x},\mathbf{w})^y (1 - f(\mathbf{x},\mathbf{w}))^{1-y}$$

Then we can obtain the log likelihood

$$\begin{aligned} L(\mathbf{w}) &= \log p(Y \mid X;\mathbf{w}) \\ &= \log \prod_{i=1}^{N} p(y_i \mid \mathbf{x}_i;\mathbf{w}) \\ &= \log \prod_{i=1}^{N} f(x_i,\mathbf{w})^{y_i} (1 - f(\mathbf{x}_i,\mathbf{w}))^{1-y_i} \\ &= \sum_{i=1}^{N} y_i \log f(\mathbf{x}_i,\mathbf{w}) + (1-y_i)\log(1 - f(\mathbf{x}_i,\mathbf{w})) \end{aligned}$$

Maximum Likelihood Estimation

We could learn **w** by maximizing the log likelihood using gradient ascent

$$\frac{\partial}{\partial w_k} L(\mathbf{w}) = \left(y \frac{1}{g(\mathbf{w}^T\mathbf{x})} - (1-y)\frac{1}{1-g(\mathbf{w}^T\mathbf{x})} \right) \frac{\partial}{\partial wk} g(\mathbf{w}^T\mathbf{x})$$

$$= \left(y \frac{1}{g(\mathbf{w}^T\mathbf{x})} - (1-y)\frac{1}{1-g(\mathbf{w}^T\mathbf{x})} \right) g(\mathbf{w}^T\mathbf{x})(1-g(\mathbf{w}^T\mathbf{x}))\frac{\partial}{\partial w_k}\mathbf{w}^T\mathbf{x}$$

$$= \left(y(1-g(\mathbf{w}^T\mathbf{x})) - (1-y)g(\mathbf{w}^T\mathbf{x}) \right)\mathbf{x}_k$$

$$= (y - f(\mathbf{x},\mathbf{w}))\mathbf{x}_k$$

This is because

$$g'(z) = \frac{d}{dz}\frac{1}{1+e^{-z}}$$

$$= \frac{1}{(1+e^{-z})^2}e^{-z}$$

$$= \frac{1}{(1+e^{-z})} \cdot \left(1 - \frac{1}{(1+e^{-z})}\right)$$

$$= g(z)(1-g(z))$$

Hence, the update rule is

$$w_k = w_k + \Delta(y - f(\mathbf{x},\mathbf{w}))\mathbf{x}_k$$

The update rule looks identical with LMS, but not the same, because $f(x, \mathbf{w})$ is now defined as a non-linear function.

Classification—Bayesian Classification

Bayesian Theorem

- Given training data X, *posteriori probability of a hypothesis* Y, P(Y|X) follows the Bayes theorem

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

- Informally, this can be written as

  posteriori = likelihood x prior / evidence

- MAP (maximum posteriori) hypothesis

$$h_{MAP} \equiv \arg\max_{y \in H} P(y|D) = \arg\max_{h \in H} P(X|y)P(y).$$

- Practical difficulty: require initial knowledge of many probabilities, significant computational cost

Naïve Bayes Classifier

- A simplified assumption: attributes are conditionally independent:

$$P(x_i \mid y) = \prod_{k=1}^{m} P(x_{ik} \mid y)$$

- No dependence relation between attributes
- Greatly reduces the computation cost, only count the class distribution.
- Once the probability $P(x_i|y)$ is known, assign x_i to the class *y* with maximum $P(x_i|y)$ * $P(y)$

Training dataset

Class:
y=1:buys_computer= ‘yes’
y=0:buys_computer= ‘no’

Data sample
x =(age<=30, Income=medium, Student=yes Credit_rating= Fair)

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
30…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Naïve Bayesian Classifier: An Example

- Compute $P(x_i|y=1)$, $P(x_i|y=0)$ for each class

P(age="<30" | buys_computer="yes") = 2/9=0.222
P(age="<30" | buys_computer="no") = 3/5 =0.6
P(income="medium" | buys_computer="yes")= 4/9 =0.444
P(income="medium" | buys_computer="no") = 2/5 = 0.4
P(student="yes" | buys_computer="yes)= 6/9 =0.667
P(student="yes" | buys_computer="no")= 1/5=0.2
P(credit_rating="fair" | buys_computer="yes")=6/9=0.667
P(credit_rating="fair" | buys_computer="no")=2/5=0.4

***x*=(age<=30 , income =medium, student=yes, credit_rating=fair)**

***P*(*x*|*y*)** : P(X|buys_computer="yes")= 0.222 x 0.444 x 0.667 x 0.0.667 =0.044
P(X|buys_computer="no")= 0.6 x 0.4 x 0.2 x 0.4 =0.019
***P*(*x*|*y*)*P(*y*)** : P(X|buys_computer="yes") * P(buys_computer="yes")=0.028
P(X|buys_computer="no") * P(buys_computer="no")=0.007

Therefore, *x* belongs to class "buys_computer=yes"

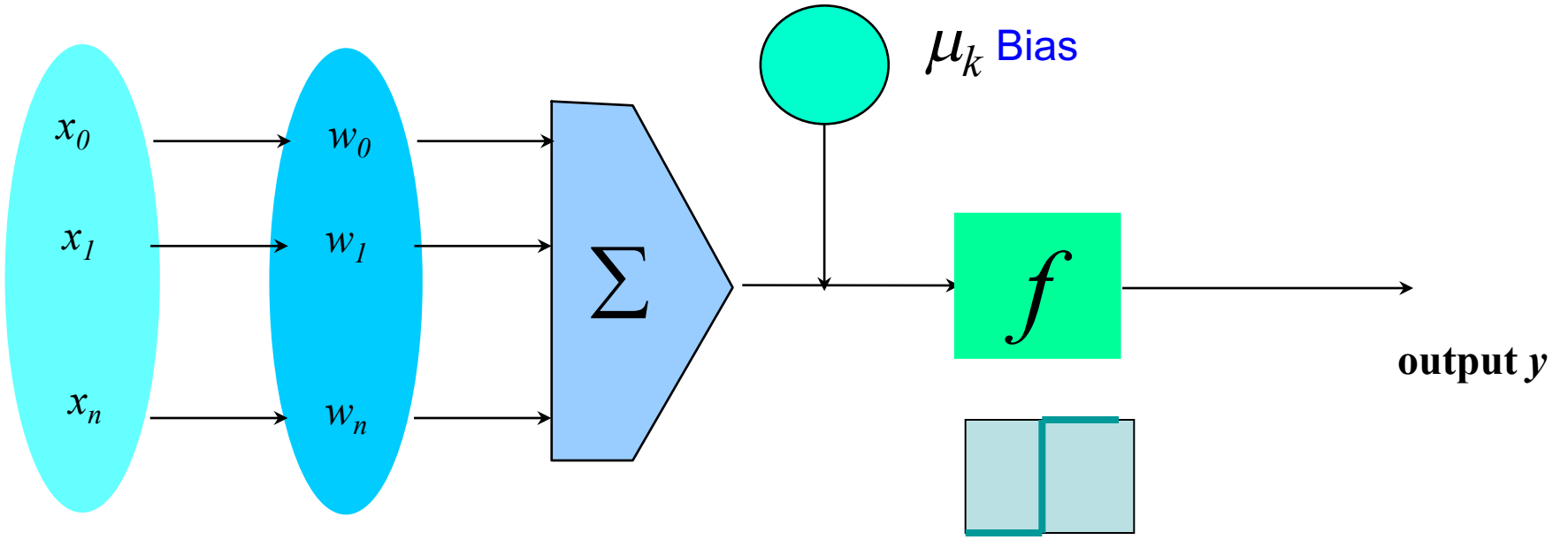
Naïve Bayesian Classifier: Comments

- Advantages
  - Easy to implement
  - Good results obtained in most of the cases
- Disadvantages
  - Assumption: class conditional independence, therefore loss of accuracy
  - Practically, dependencies exist among variables
  - E.g., hospitals: patients: Profile: age, family history etc

    Symptoms: fever, cough etc., Disease: lung cancer, diabetes etc
  - Dependencies among these cannot be modeled by Naïve Bayesian Classifier
- How to deal with these dependencies?
  - Bayesian Belief Networks

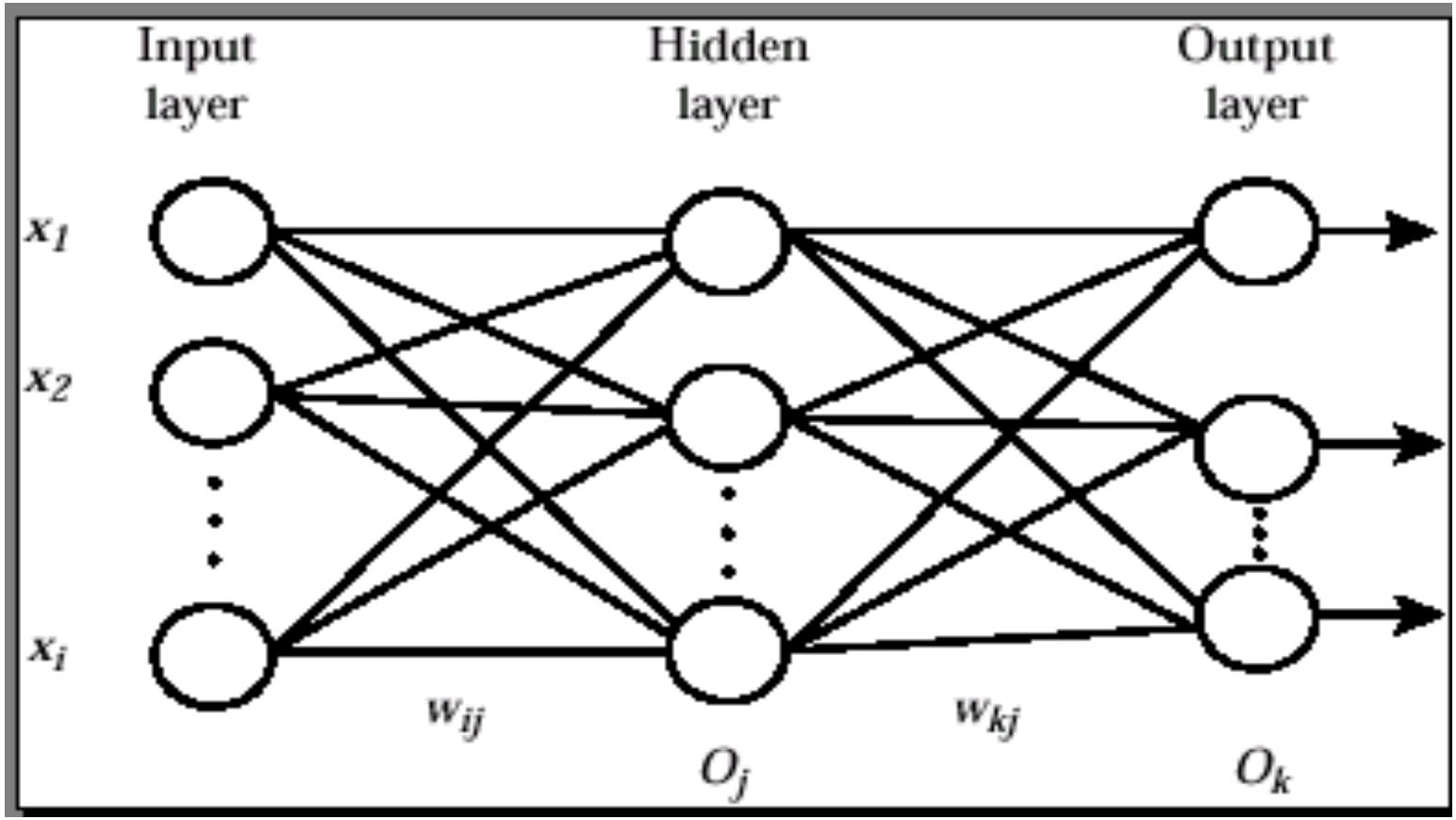
Classification—Neural Network

A Neuron (= a perceptron)



$$y = \text{sign}(\sum_{i=0}^{n} w_i \mathbf{x}_i + b)$$

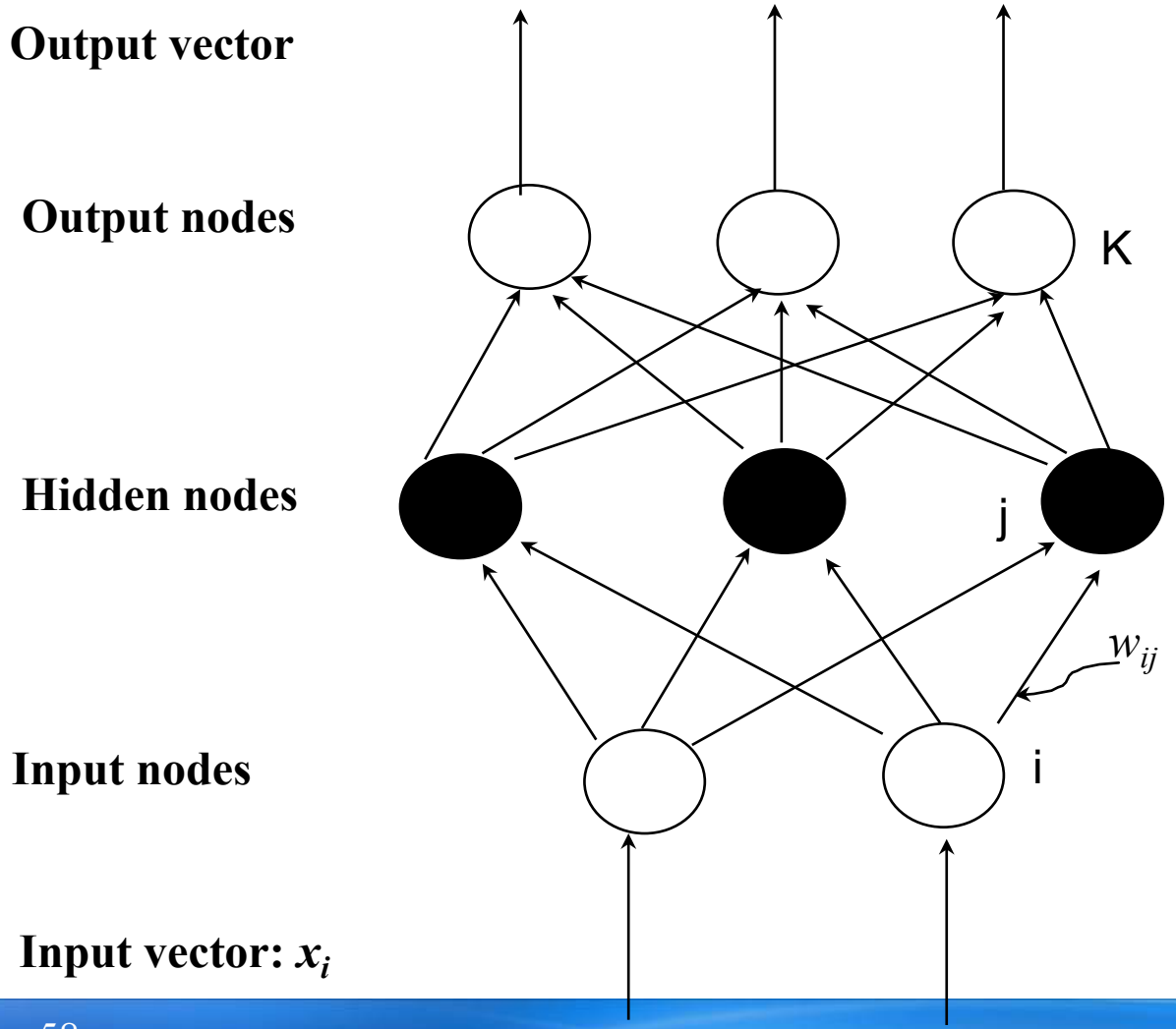
Multi layer Neural Network



Network Training

- The ultimate objective of training
  - obtain a set of weights that makes almost all the tuples in the training data classified correctly
- Steps
  - Initialize weights with random values
  - Feed the input tuples into the network one by one
  - For each unit
    - Compute the net input to the unit as a linear combination of all the inputs to the unit
    - Compute the output value using the activation function
    - Compute the error
    - Update the weights and the bias

Back Propagation



$$I_j = \sum_i w_{ij} O_i + \theta_j$$

$$O_j = \frac{1}{1 + e^{-I_j}}$$

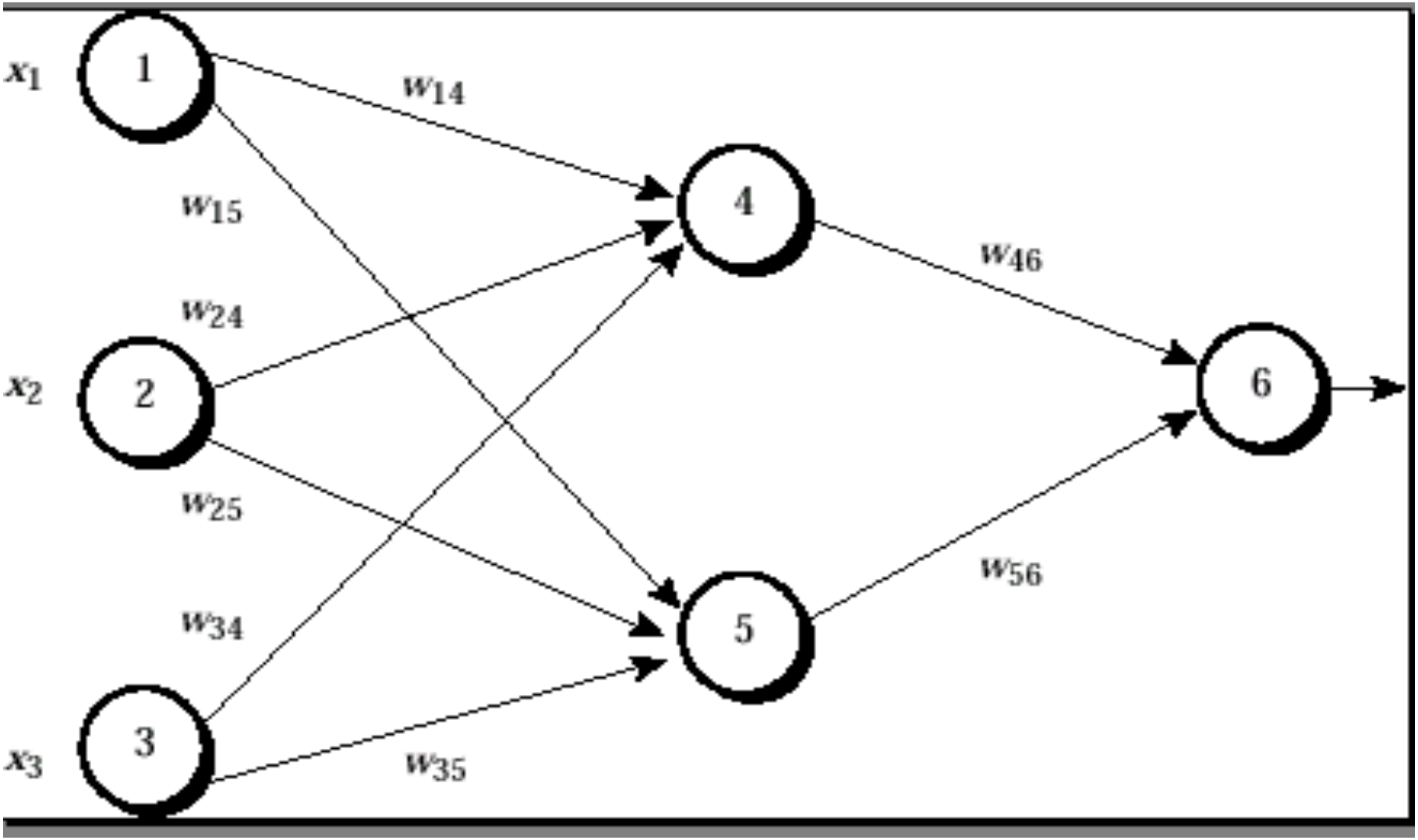
$$Err_j = O_j (1 - O_j)(T_j - O_j)$$

$$Err_j = O_j (1 - O_j) \sum_k Err_k w_{jk}$$

$$w_{ij} = w_{ij} + (l) Err_j O_i$$

$$\theta_j = \theta_j + (l) Err_j$$

Back Propagation



One example

- Input: X={1,0,1}, output: 1

- X1=1, x2=0, x3=1,
  w14=0.2, w15=-0.3,
  w24=0.4, w25=0.1,
  w34=-0.5, w35=0.2,
  w46=-0.3, w56=-0.2,

- Bias: node 4:-0.4, node 5:0.2, node 6:0.1

- Learning rate l=0.9

One example

- Node 4:

  input :w14*x1+w24*x2+w34*x3+bias of node 4=1*0.2+0.4*0-0.5*1-0.4=-0.7

  output:

  O_4=0.332

$$O_j = \frac{1}{1+e^{-I_j}}$$

- The same: node 5: input: 0.1, output:0.525

- Node 6:

  input :w46*o4+w56*o5+bias of node 6=-0.3*0.332-0.2*0.525+0.1=-0.105

  output:0.474

One example

- Node 6: $Err_j = O_j(1-O_j)(T_j - O_j)$

  0.474*(1-0.474)*(1-0.474)=0.1311

- Node 5: $Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$

  0.525*(1-0.525)*0.1311*(-0.2)=-0.0065

- Node 4:-0.0087

One example

- W46: $w_{ij} = w_{ij} + (l) Err_j O_i$

  -0.3+(0.9)(0.1311)(0.332)=-0.261

- Other Wij is the same with w46

- Bias of node 6: $\theta_j = \theta_j + (l) Err_j$

  0.1+(0.9)*(0.1311)=0.218

- Other bias is the same with node 6

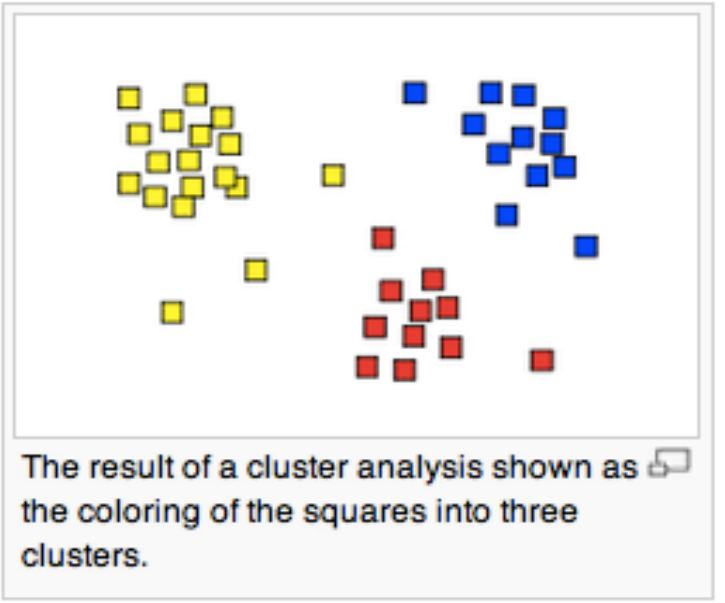
Other ML Scenarios

Clustering

Cluster analysis or clustering is the task of assigning a set of objects into groups (called clusters) so that the objects in the same cluster are more similar (in some sense or another) to each other than to those in other clusters.

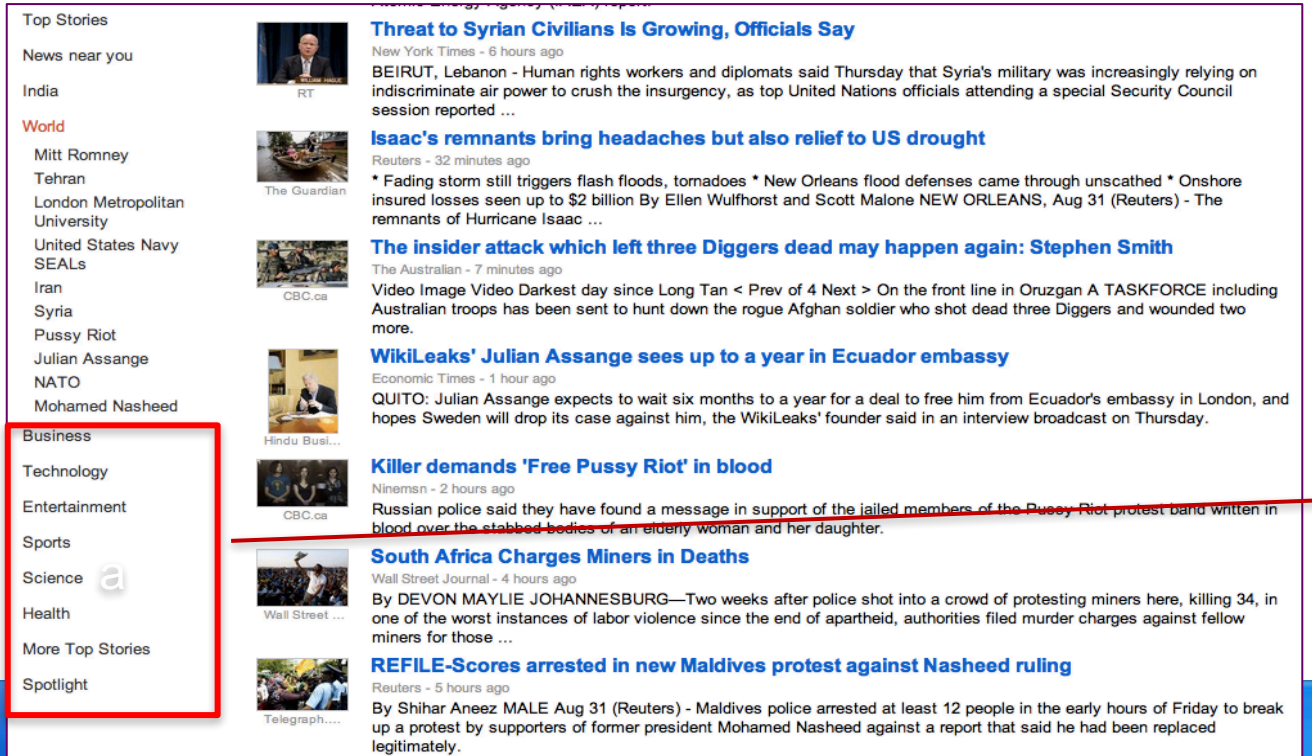
1. Social network analysis

In the study of social networks, clustering may be used to recognize communities within large groups of people.



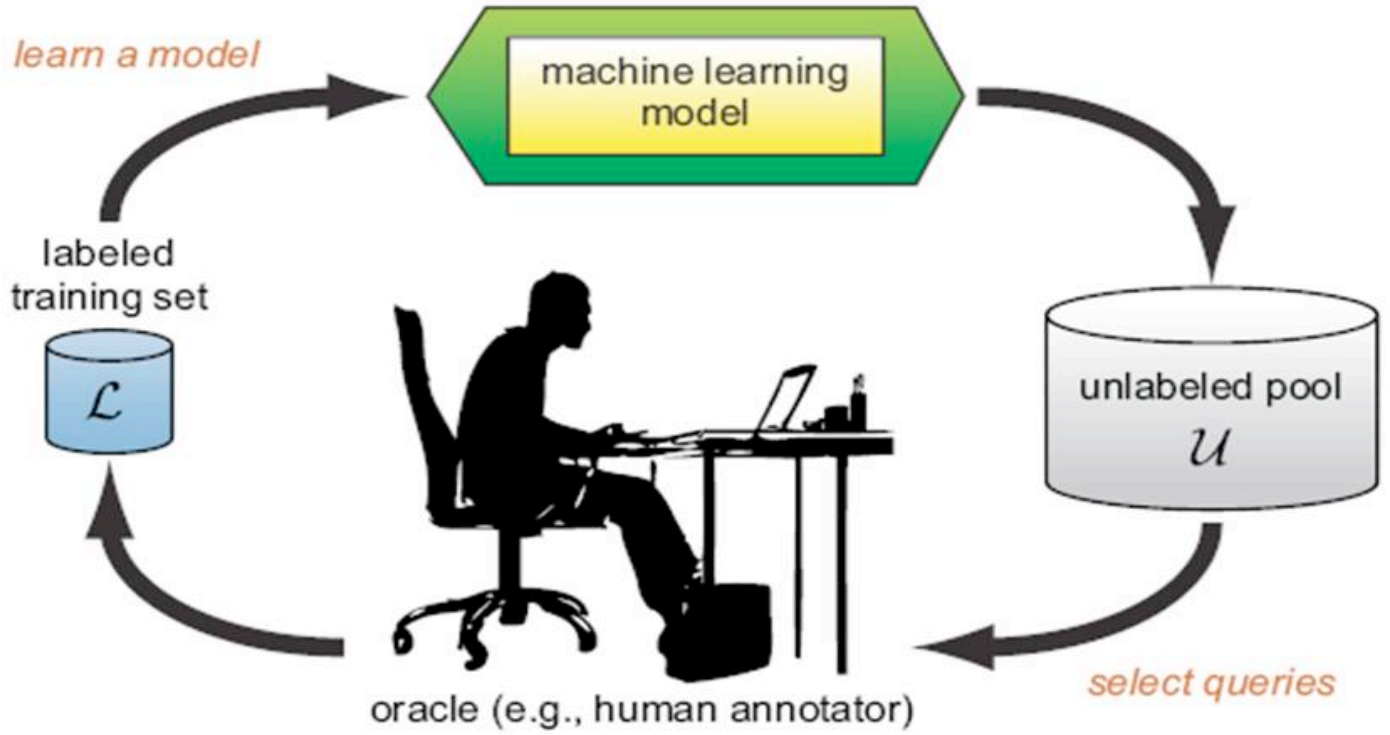
The result of a cluster analysis shown as the coloring of the squares into three clusters.

2. News Services like Google News(below figure)



Similar news are grouped together.

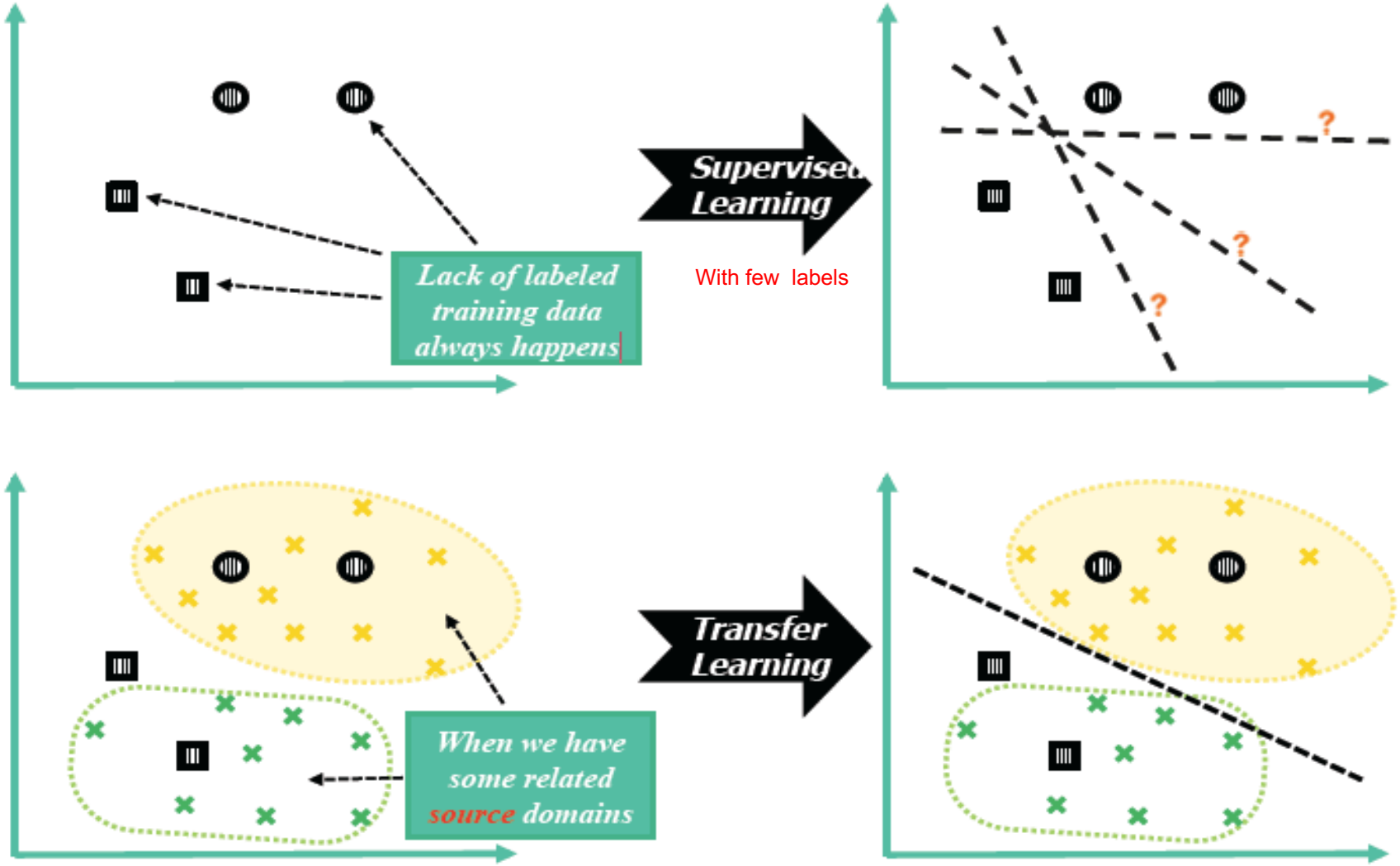
Active Learning



The key hypothesis is that if the learning algorithm is allowed to choose the data from which it learns, it will perform better with less training.

Active learning systems attempt to overcome the labeling bottleneck by asking queries from unlabeled instances to be labeled by an oracle(e.g., a human annotator).

Transfer Learning



Summary

- Supervised Learning
  - KNN
  - Decision Tree
  - Perceptron
  - Logistic Regression
  - Bayesian Classification
  - Neural Networks

The State of Machine Learning

Classification models

Decision tree 
Bayesian classifier
Perceptron
Neural networks

Maximal Margin

SVM (Vapnik) 



Maximal margin sequential learning

M3N network (Ben Taskar) 

Deep learning

Deep belief networks




Topic models

PLSI, LDA, etc.




Sequential learning

HMM 

Sequential learning

MEMM, CRF, voted perceptron



Graphical models

Factor graph, Exponential model 



For more from http://aminer.org/

Thanks!

Jie Tang, DCST

http://keg.cs.tsinghua.edu.cn/jietang/

http://aminer.org

Email: jietang@tsinghua.edu.cn