ML Algorithm Review

Jie Tang
Department of Computer Science & Technology
Tsinghua University
2012

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

Data Structures

- An example with *p* features

$$\boldsymbol{x}_i=[x_{i1}, x_{i2},\ldots, x_{ip}]$$

- Data (set) matrix

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{N1} & \cdots & x_{Nf} & \cdots & x_{Np} \end{bmatrix}$$

Type of data

- Interval-scaled variables
- Binary variables
- Nominal, ordinal, and ratio variables
- Variables of mixed types

- To make things easy, let us say $x_i \in X = R^n$

Similarity and Dissimilarity between Data Instances

- Distances are normally used to measure the similarity or dissimilarity between two data instances
- Some popular ones include: *Minkowski distance*:

$$d(i,j) = \sqrt[q]{(|x_{i1} - x_{j1}|^q + |x_{i2} - x_{j2}|^q + ... + |x_{ip} - x_{jp}|^q)}$$

where $i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ and $j = (x_{j1}, x_{j2}, \ldots, x_{jp})$ are two p-dimensional data objects, and q is a positive integer

- If $q = 1$, d is Manhattan distance

$$d(i,j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + ... + |x_{ip} - x_{jp}|$$

Similarity and Dissimilarity Between data Instances (Cont.)

- *If q = 2, d* is Euclidean distance:

$$d(i,j) = \sqrt{(|x_{i1} - x_{j1}|^2 + |x_{i2} - x_{j2}|^2 + ... + |x_{ip} - x_{jp}|^2)}$$

  - Properties
    - $d(i,j) \geq 0$
    - $d(i,i) = 0$
    - $d(i,j) = d(j,i)$
    - $d(i,j) \leq d(i,k) + d(k,j)$
- Or more general Mahalanobis distance $(x_i - x_j)^T A (x_i - x_j)$, where A is a covariance matrix.
- Also, one can use weighted distance, parametric Pearson product moment correlation, or other dissimilarity measures

Cosine Similarity

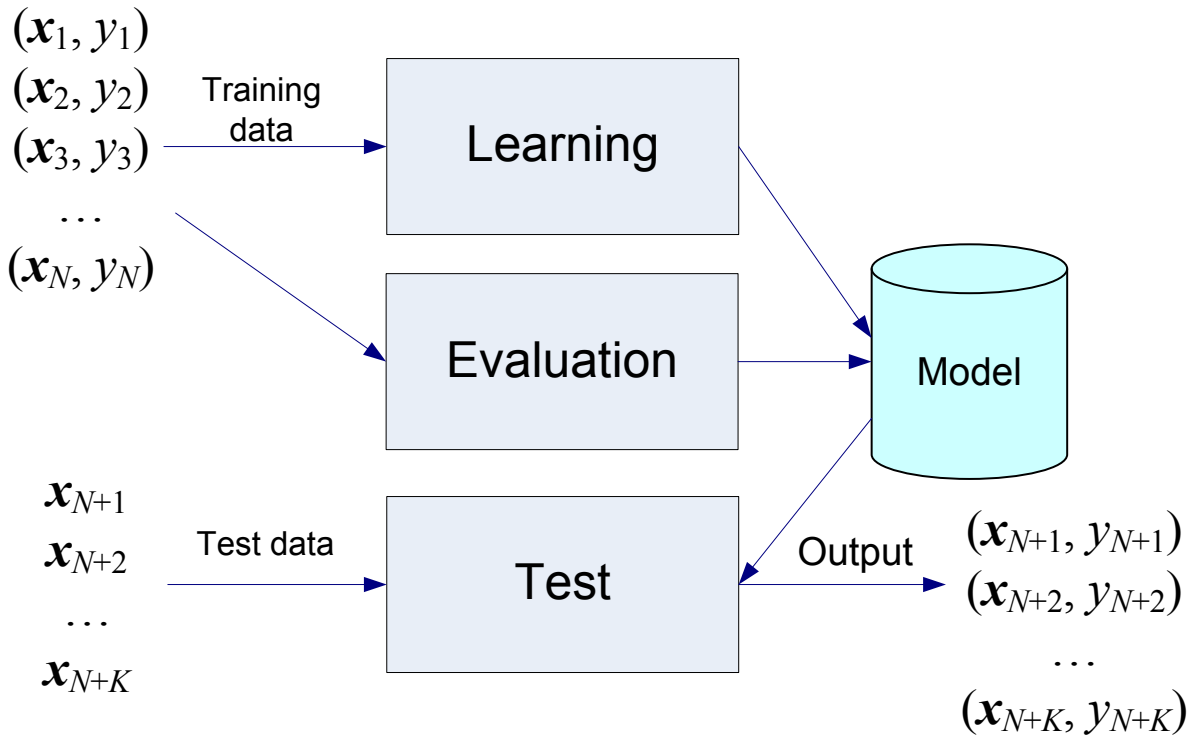
- Given two vectors (data instances), their similarity can be calculated by cosine similarity

$$sim(x_i, x_j) = \frac{\sum_{k=1,..,p} x_{ik} \times x_{jk}}{\sqrt{\sum_{k=1,..,p} x_{ik}^2} \times \sqrt{\sum_{k=1,..,p} x_{jk}^2}}$$

Basic ML Algorithm Review

- Basic algorithms review
  - Data representation
  - Supervised learning
    - Evaluation
    - 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

Supervised Learning

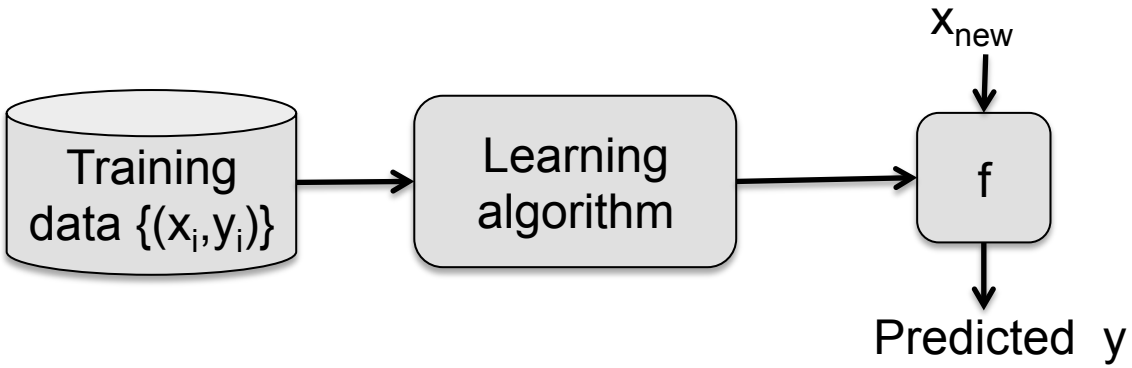


* x_i is a vector and x_{ij} represents the jth feature of example x_i

Supervised Learning

- Given a training set $S=\{(x_1, y_1),(x_2, y_2),\ldots,(x_N, y_N)\}$, and $x_i \in X=R^m$, $i=1,2,\ldots,N$
- To learn a function $f(x)$, which can best fit the data

$$y = f(x)$$



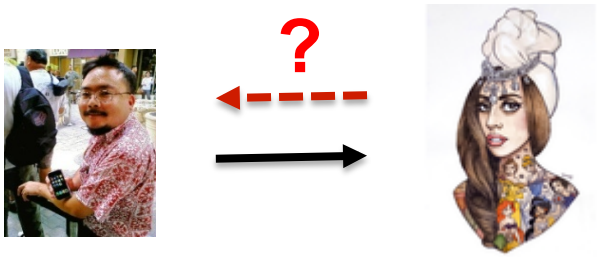
Regression: When y is continuous (i.e., $y_i \in Y=R$), we call the learning problem as regression.

Classification: When y can take on only a small number of discrete values (e.g., the binary case $y_i \in Y=\{1,-1\}$), we call the learning problem as classification.

Example

Living area (feet^2)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

VS



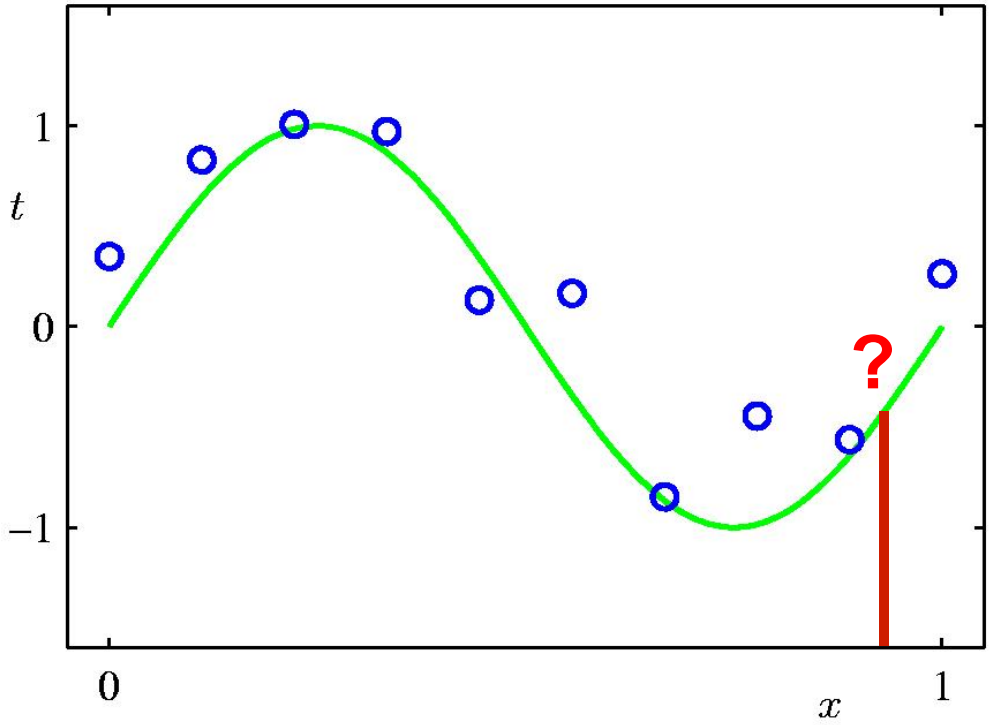
How much will it be, if we want to buy a house with a size of 8000 feet?

Regression

If you follow Lady Gaga on Twitter, will she follow back?

Classification

Linear Regression



As the first step, we need to decide how we're going to represent the function *f*.

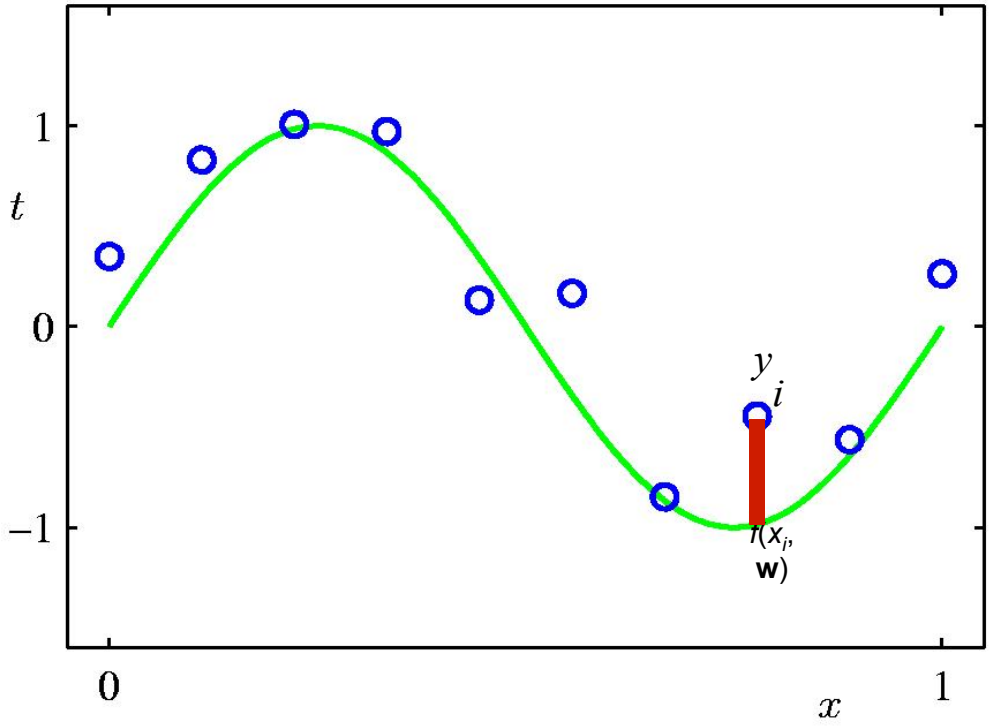
One example: polynomial curve fitting

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

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

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

Least Squares Loss Function



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

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 that repeatedly changes **w** to make $L(\mathbf{w})$ smaller, util we converge to a value of **w** that mimize $L(\mathbf{w})$.
- Gradient descent is a natural search algorithm that update **w** in the direction of steepest decrease of L:

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

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:

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

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

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 {

$$w_j = w_j + \triangle \sum_i (f(x_i, \mathbf{w}) - y_i)\, x_i^j$$

}

- 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 {

Foreach x_i {

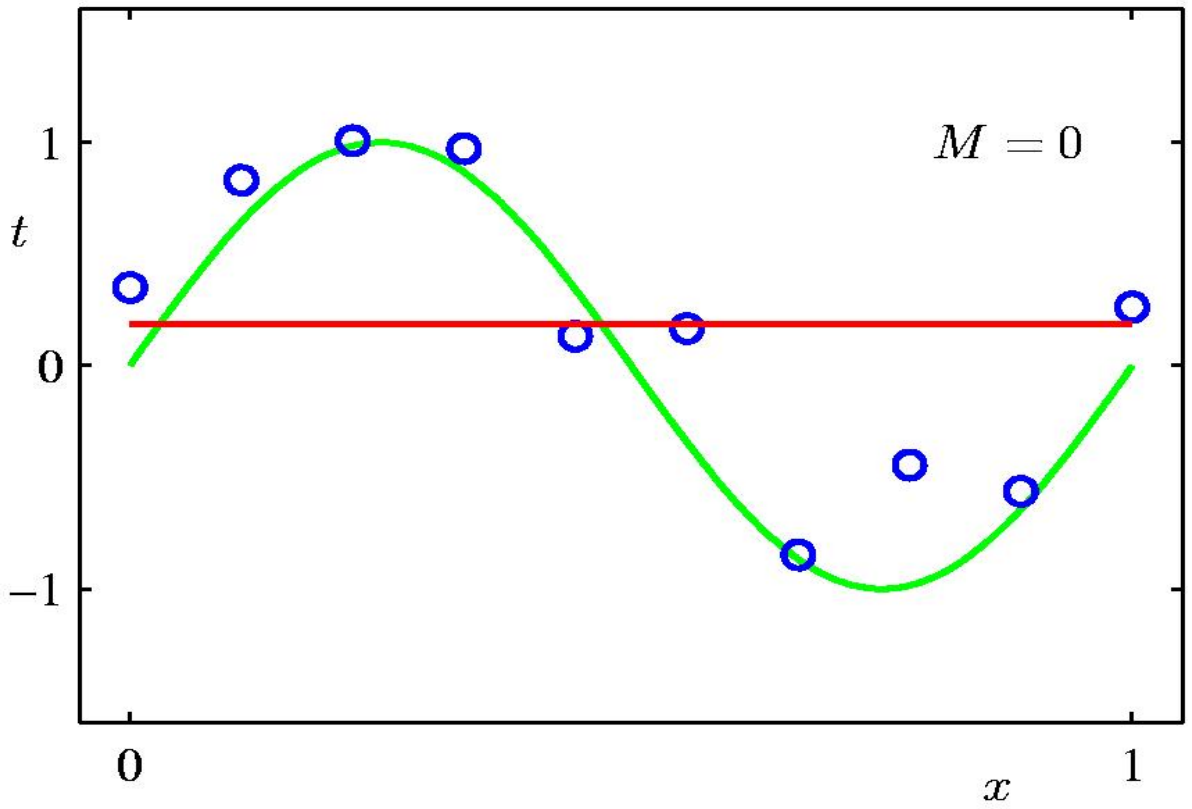
$$w_j = w_j + \triangle (f(x_i, \mathbf{w}) - y_i)\, x_i^j$$

}

}

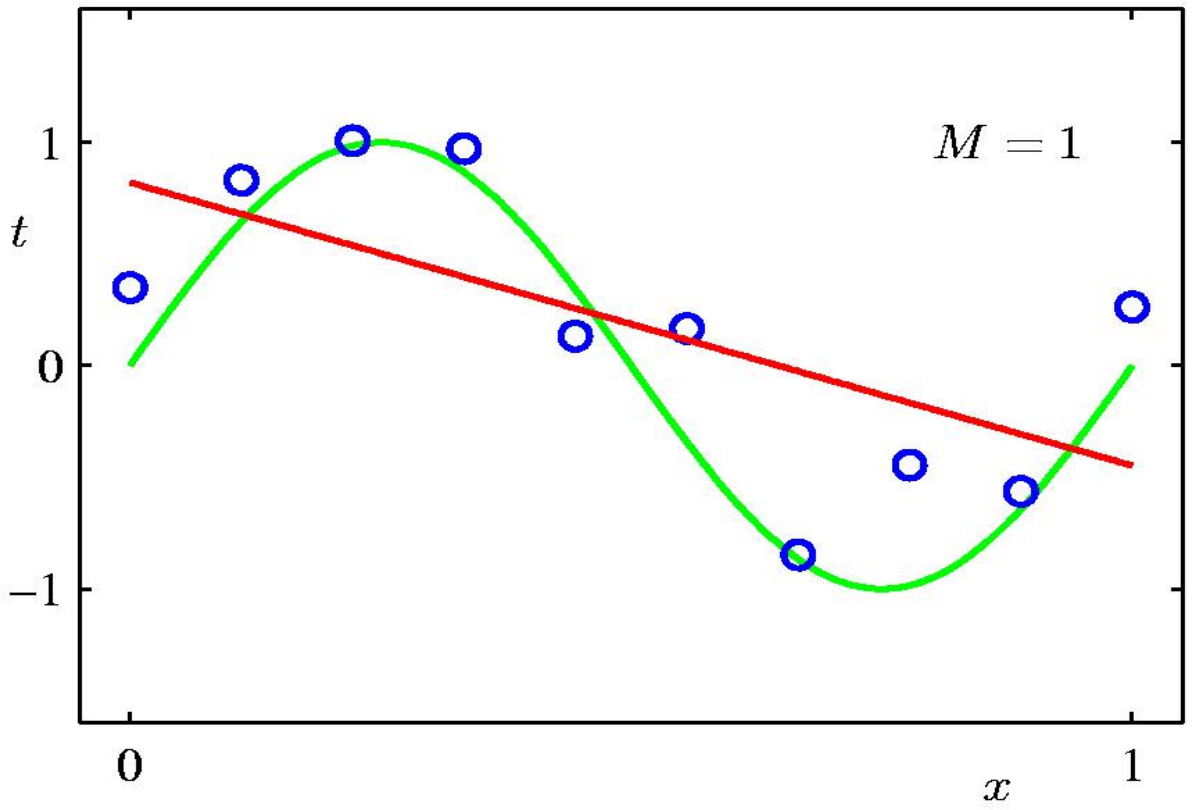
Particularly when the training set is large, stochastic gradient descent is often preferred.

0th Order Polynomial

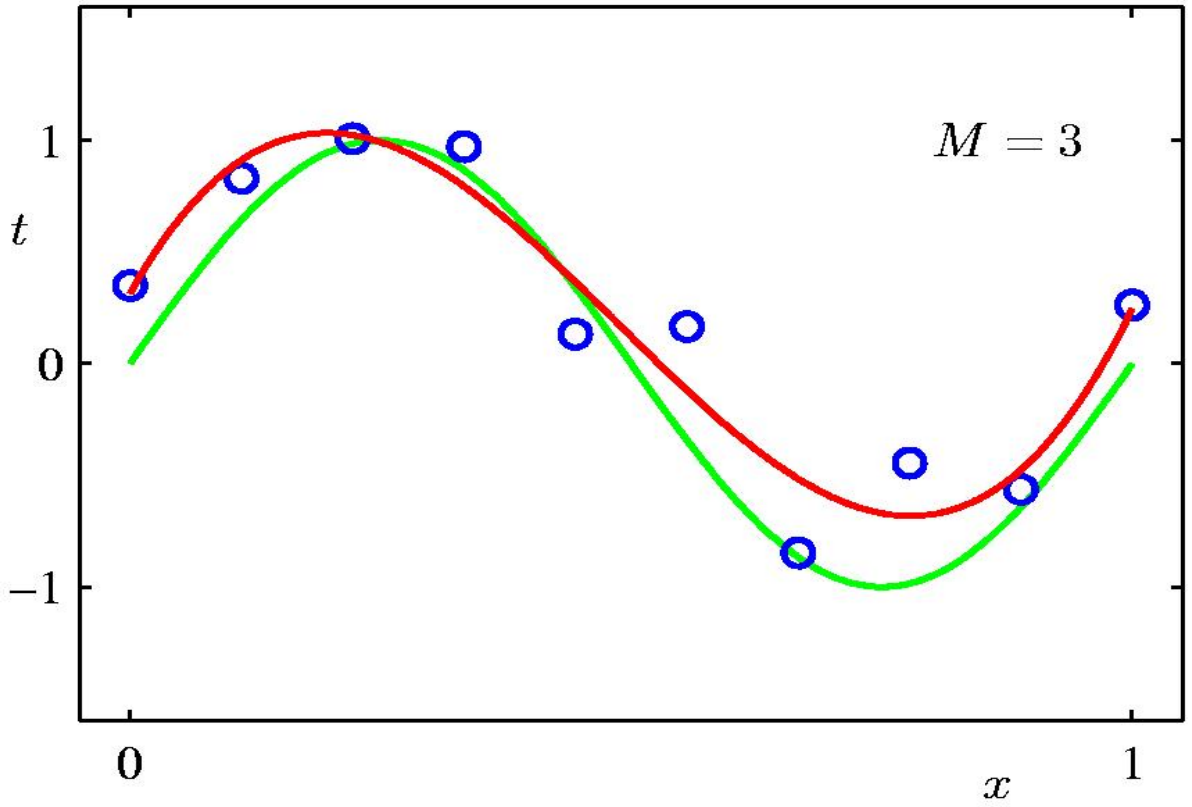


* A number of the following pages are from Bishop's slides

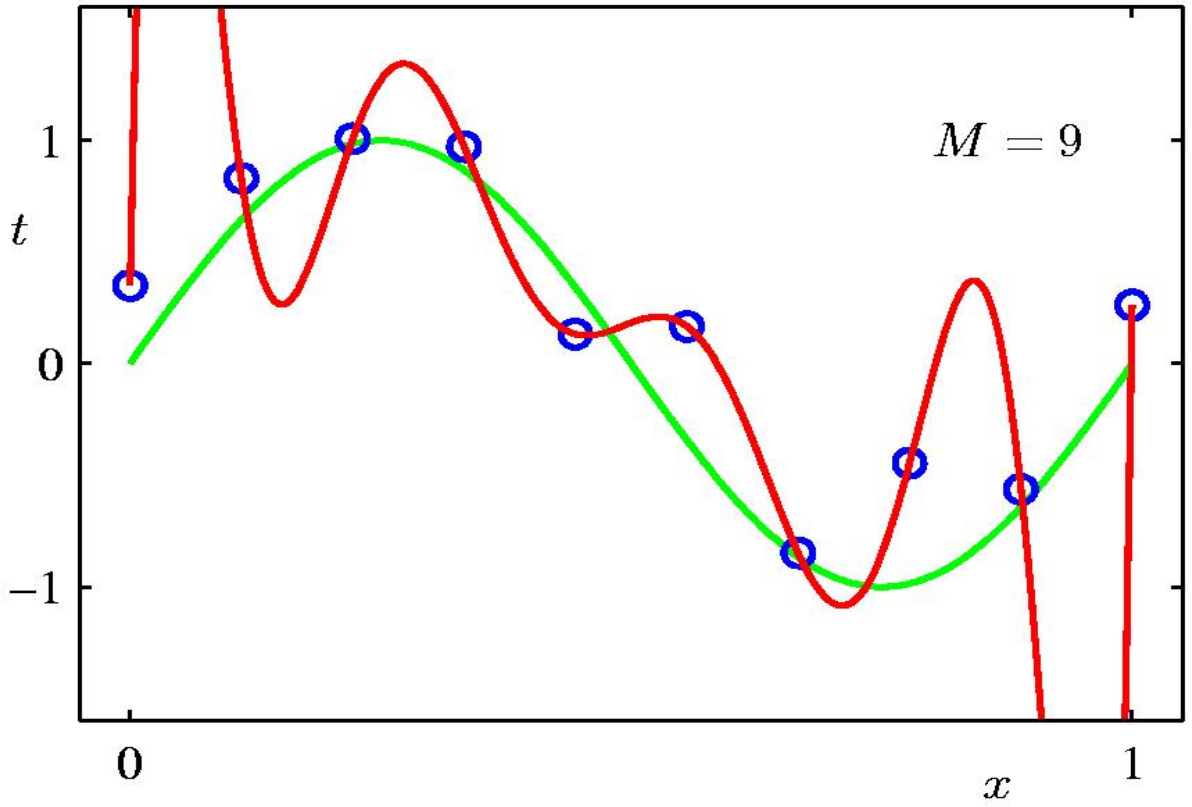
1st Order Polynomial



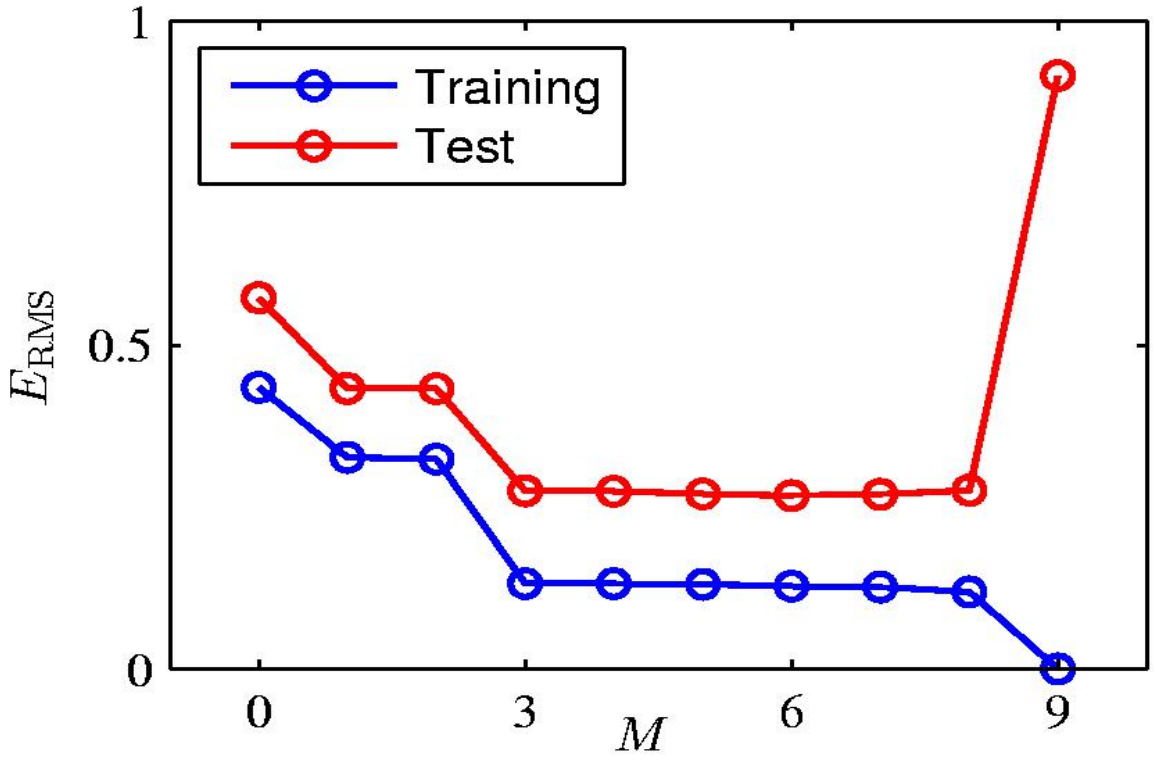
3rd Order Polynomial



9th Order Polynomial



Over-fitting



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

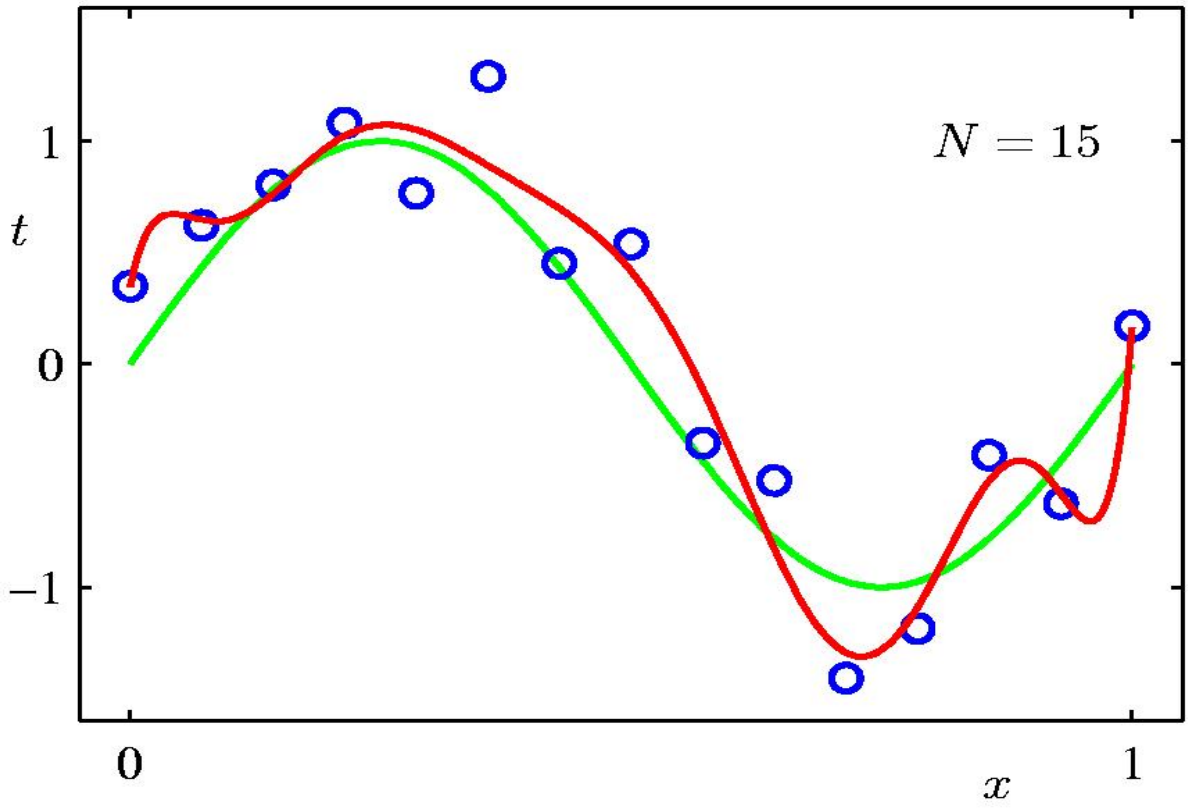
Polynomial Coefficients

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

Data Set Size

$$N = 15$$

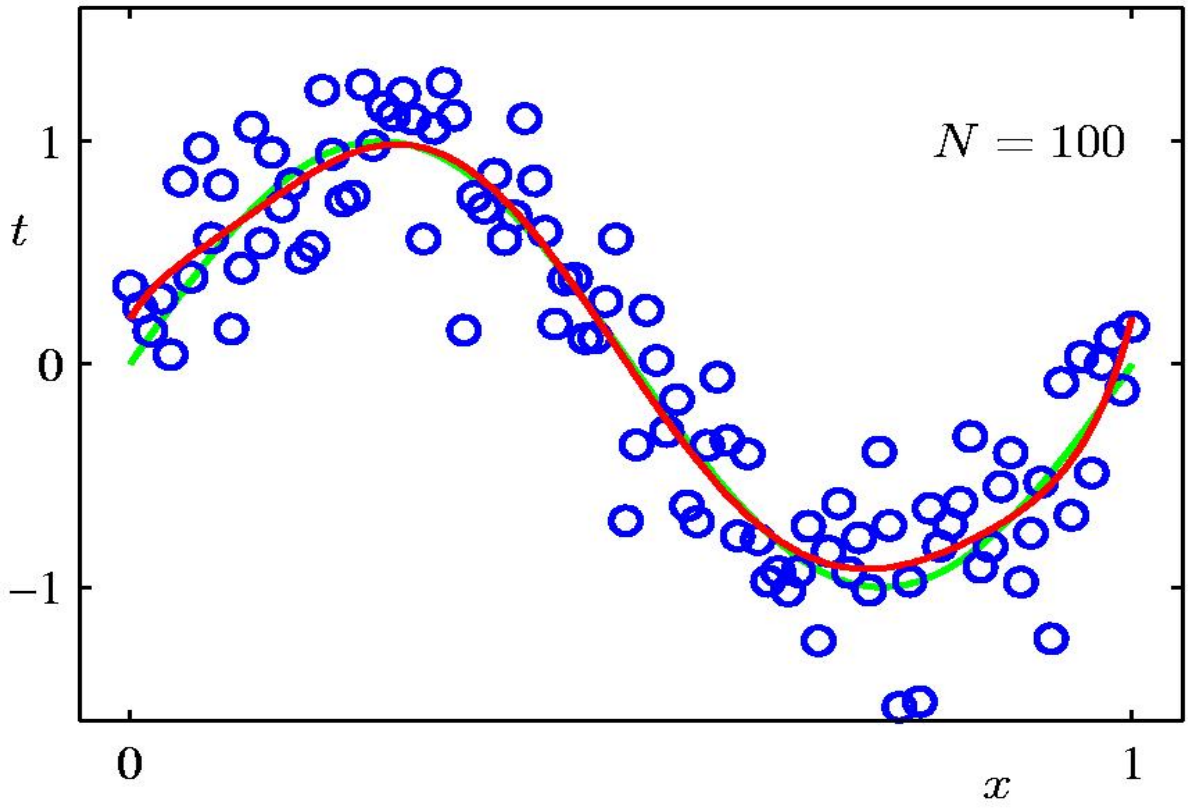
9th Order Polynomial



Data Set Size

$$N = 100$$

9th Order Polynomial



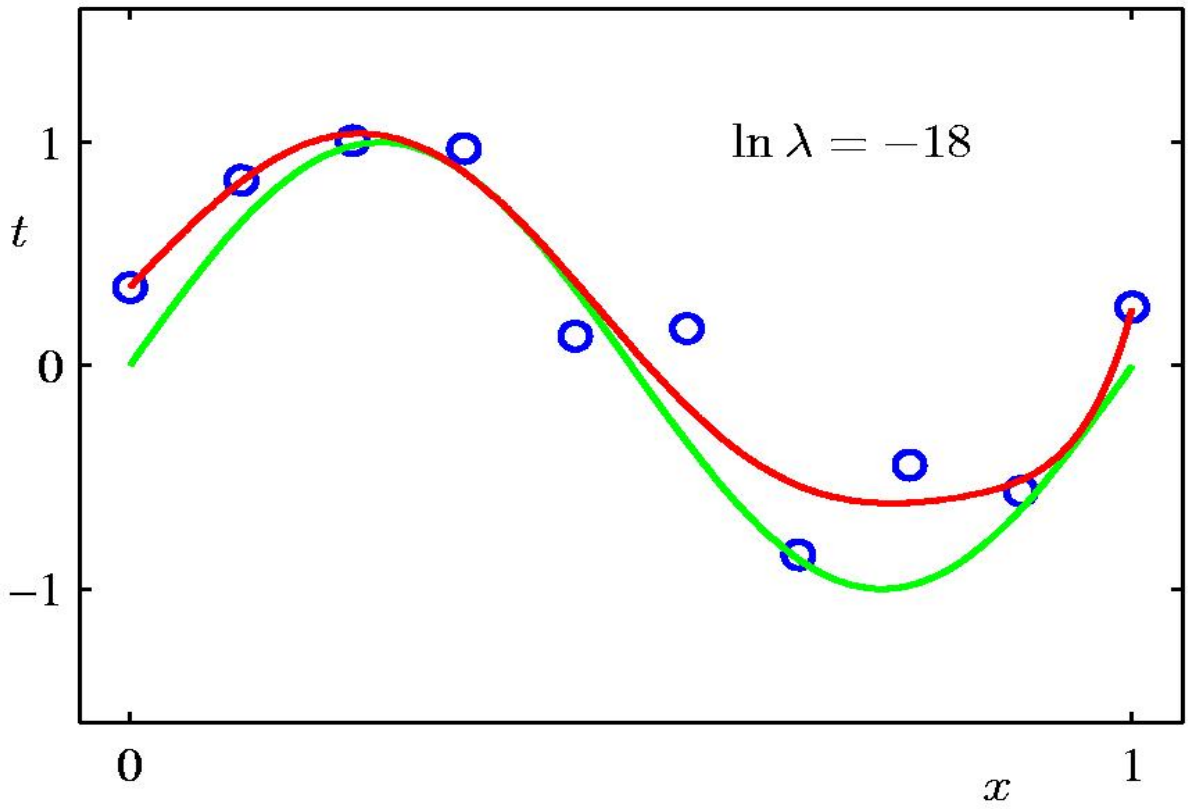
Regularization

Penalize large coefficient values

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

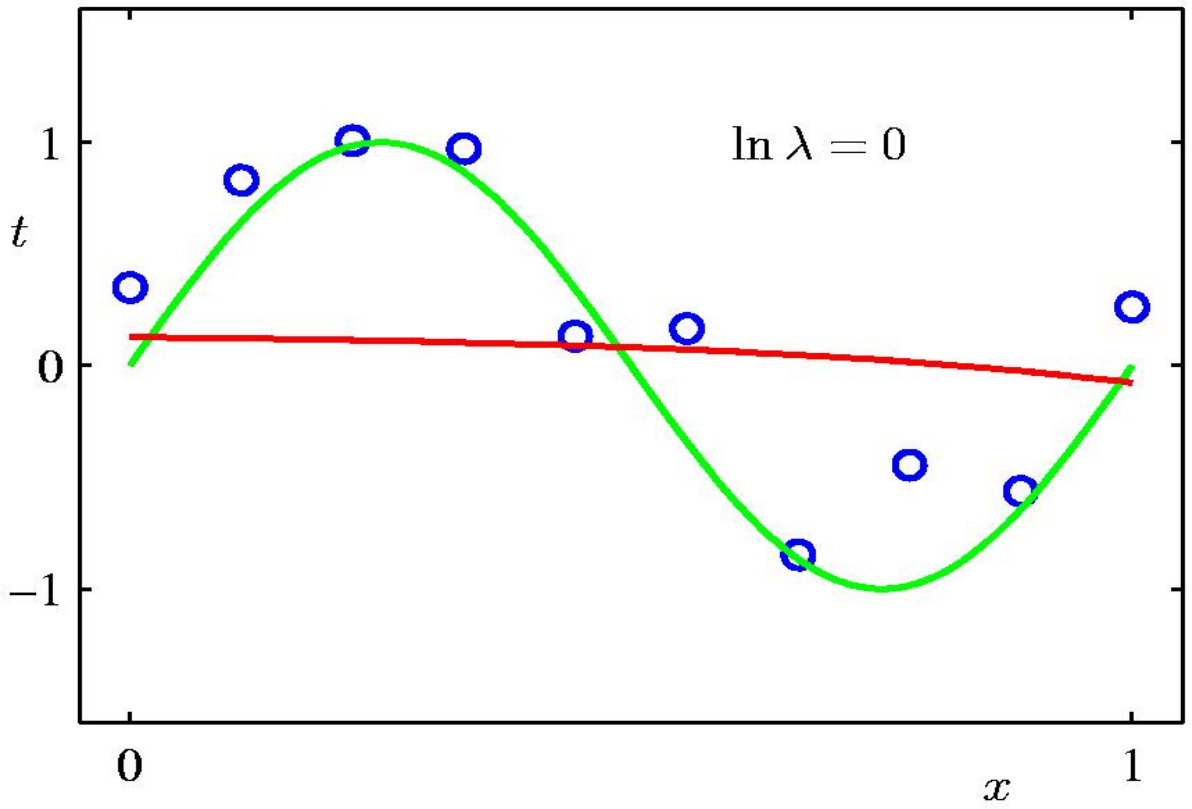
Regularization

$$\ln \lambda = -18$$

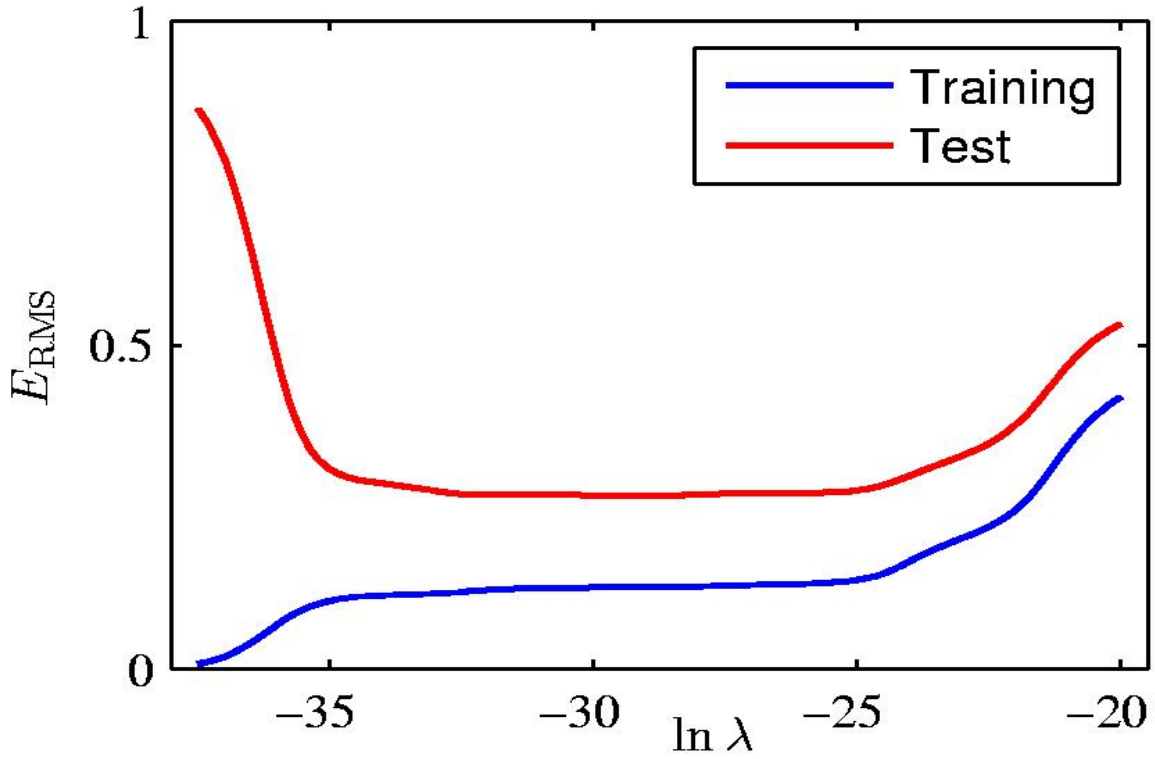


Regularization

$$\ln \lambda = 0$$



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

Let us back to the simple case…

- Linear function

$$f(x_i, \mathbf{w}) = w_0 + w_1 x_{i1} + \ldots + w_p x_{ip}$$

$$= \sum_{j=0}^{p} w_j x_{ij} = \mathbf{w}^T \mathbf{x}_i$$

- And the loss function

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

- Should all example be treated equally? How about if we assign different weights to different examples?

Locally Weighted Linear Regression

- Fiting **w** to minimize

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

where a_i is a non-negative valued weight for x_i

$$\alpha_i = \exp\left(-\frac{(x_i - x_q)^2}{2\tau^2}\right)$$

here x_q is the example we want to predict its y

- With this, we have the first example of a non-parametric algorithm
  - The number of parameters is not fixed and may grows with the increase of the size of the training set.

Probabilistic Interpretation

- Let us assume that each example has an error term to represent unmodeled effects.

$$y_i = \mathbf{w}^T \mathbf{x}_i + \varepsilon_i$$

- Also assume that ε satisfies IID, e.g.,

$$p(\varepsilon_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\varepsilon_i^2}{2\sigma^2}\right)$$

- Then we have

$$p(y_i \mid x_i; \mathbf{w}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \mathbf{w}^T \mathbf{x}_i)^2}{2\sigma^2}\right)$$

- Given data $X=\{x_i\}$, and their corresponding Y

$$L(\mathbf{w}) = p(Y \mid X; \mathbf{w}) = \prod_{i=1}^{N} p(y_i \mid x_i; \mathbf{w})$$

$$= \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \mathbf{w}^T \mathbf{x}_i)^2}{2\sigma^2}\right)$$

Maximum (log-)Likelihood

- Log likelihood

$$L(\mathbf{w}) = \log p(Y \mid X; \mathbf{w}) = \log \prod_{i=1}^{N} p(y_i \mid x_i; \mathbf{w})$$

$$= \sum_{i=1}^{N} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \mathbf{w}^T \mathbf{x}_i)^2}{2\sigma^2} \right)$$

$$= N \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

Which is the same as minimizing least squares

* Under certain probabilistic assumption, minimizing least squares regression corresponds to maximizing likelihood estimation of **w**

Classification

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

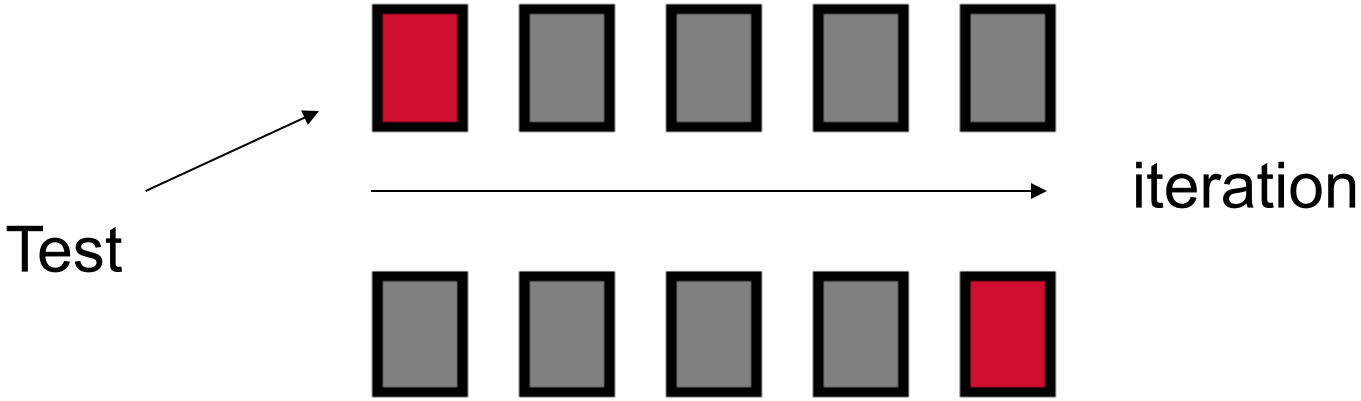
- *Holdout*: the training set/testing set.
  - Good for a large set of data.
- *k-fold Cross-validation (交叉验证)*:
  - divide the data set into k sub-samples.
  - In each run, use one distinct sub-sample as testing set and the remaining k-1 sub-samples as training set.
  - Evaluate the method using the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



A review of classification models

- Perceptron
- Logistic Regression
- Multinomial Logistic Regression
- k-Nearest Neighbor
- Decision Tree
- Bayesian Classification
- Neural Networks

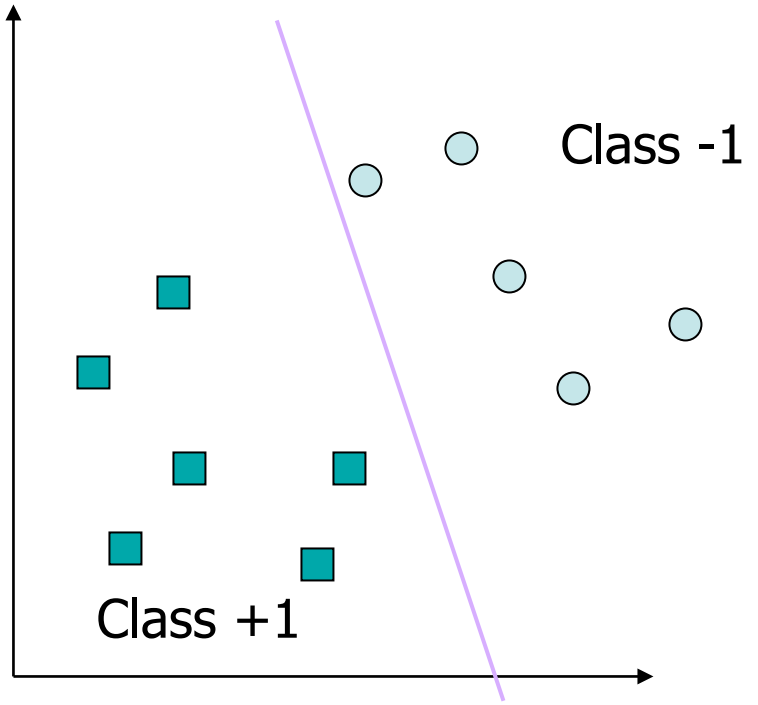
Perceptron—a linear model

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

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

i.e.,

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



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*
      $\mathbf{w} \leftarrow \mathbf{w} + y_i\,x_i$

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

- 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

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

- After the update, we have

$$\mathbf{w}' = \mathbf{w} + y_i x_i$$

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

$$= y_i \mathbf{w} x_i + y_i^2 \|x_i\|^2$$

$$= y_i \mathbf{w} x_i + \|x_i\|^2$$

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

Rosenblatt's Algorithm

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

- $w_0 \leftarrow 0; b_0 \leftarrow 0; k \leftarrow 0$
- $R = max_i \|\vec{x}_i\|$
- 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

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

A review of classification models

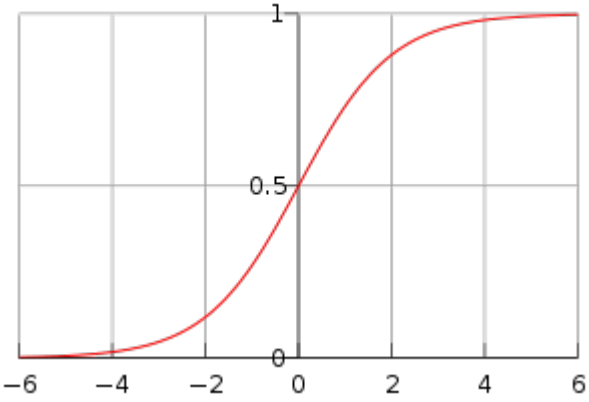
- Perceptron
- Logistic Regression
- Multinomial Logistic Regression
- k-Nearest Neighbor
- Decision Tree
- Bayesian Classification
- Neural Networks

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?

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

i.e.,

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

Then we can obtain the log likelihood

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

Maximum Likelihood Estimation

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

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

This is because

$$g'(z) = \frac{d}{dz} \frac{1}{1+e^{-z}}$$
$$= \frac{1}{(1+e^{-z})^2} e^{-z}$$
$$= \frac{1}{(1+e^{-z})} \cdot \left(1 - \frac{1}{(1+e^{-z})} \right)$$
$$= g(z)(1-g(z))$$

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

Hence, the update rule is

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

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

Learning with Newton's method

Gradient

$$w_j = w_j + \Delta f'(w_j)$$

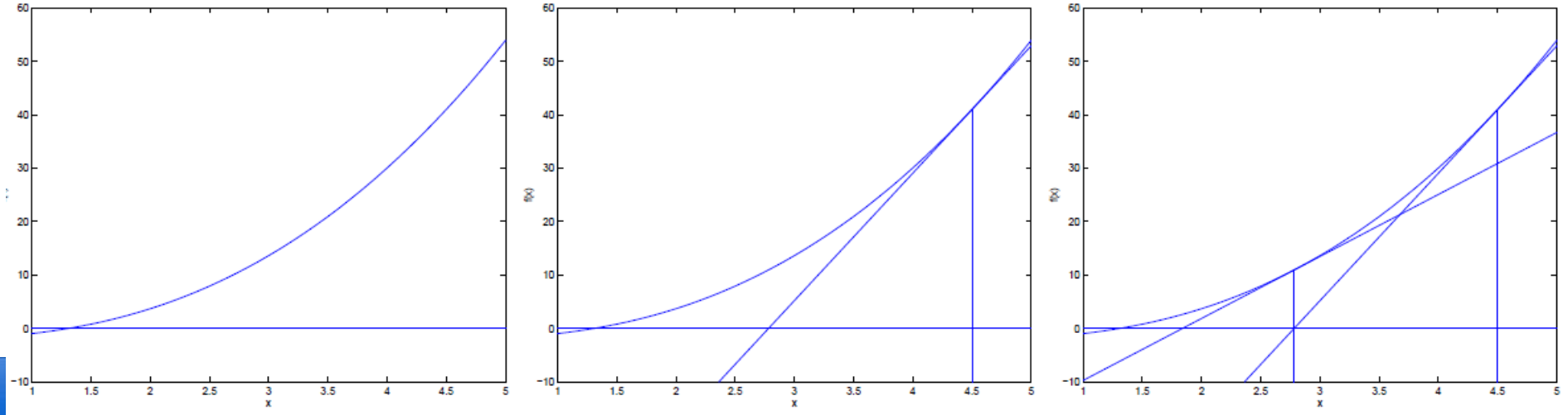
VS

Newton's method

$$w_j = w_j - \frac{L'(w_j)}{L''(w_j)}$$

- What is Newton's method?
- Given a function f(w), suppose that we want to find a value of w so that f(w)=0. Newton's method is to update w with the following rule:

$$w_j = w_j - \frac{f(w_j)}{f'(w_j)}$$



Learning with Newton's method

- Applying Newton's method to maximize the likelihood
- As the maxima of *L*(w) corresponds to points where its first derivative L'(w)=0. Therefore, by letting f(w)=L'(w) and then based on the Newton's method, we have

$$w_j = w_j - \frac{L'(w_j)}{L''(w_j)}$$

- Further considering **w** as a vector

$$\mathbf{w} = \mathbf{w} - H^{-1}\nabla_{\mathbf{w}} L(\mathbf{w})$$

where *H* is called Hessian matrix

$$H_{ij} = \frac{\partial^2 L(\mathbf{w})}{\partial w_i \partial w_j}$$

*The Newton's method is also called Fisher scoring when applying to maximize the logistic regression log likelihood function.

Multinomial Logistic Regression

Consider a multi-classification problem where y takes on value from {1,2,…,K}.

One fairly simple way to solve the problem is to image, for K possible outcomes, running K-1 independent binary logistic regression models, in which one outcome is chosen as a "pivot" and then the other K-1 outcomes are separately regressed against the pivot outcome.

If K is chosen as the pivot:

$$\log \frac{P(y_i = 1 \mid x_i; \mathbf{w})}{P(y_i = K \mid x_i; \mathbf{w})} = \mathbf{w}_1^T \cdot x_i$$

$$\log \frac{P(y_i = 2 \mid x_i; \mathbf{w})}{P(y_i = K \mid x_i; \mathbf{w})} = \mathbf{w}_2^T \cdot x_i$$

$$\cdots$$

$$\log \frac{P(y_i = K-1 \mid x_i; \mathbf{w})}{P(y_i = K \mid x_i; \mathbf{w})} = \mathbf{w}_{K-1}^T \cdot x_i$$

Then exponentiate both sides:

$$P(y_i = 1 \mid x_i; \mathbf{w}) = P(y_i = K \mid x_i; \mathbf{w}) \cdot e^{\mathbf{w}_1^T \cdot x_i}$$

$$P(y_i = 2 \mid x_i; \mathbf{w}) = P(y_i = K \mid x_i; \mathbf{w}) \cdot e^{\mathbf{w}_2^T \cdot x_i}$$

$$\cdots$$

$$P(y_i = K-1 \mid x_i; \mathbf{w}) = P(y_i = K \mid x_i; \mathbf{w}) \cdot e^{\mathbf{w}_{K-1}^T \cdot x_i}$$

Using the fact that all K of the probabilities must sum to one, we find:

$$P(y_i = K \mid x_i; \theta) = \frac{1}{1 + \sum_{l=1}^{K-1} e^{\theta_l \cdot x_i}}$$

Other probabilities can be found in the same way:

$$P(y_i = 1 \mid x_i; \mathbf{w}) = \frac{e^{\mathbf{w}_1^T \cdot x_i}}{1 + \sum_{l=1}^{K-1} e^{\mathbf{w}_l^T \cdot x_i}}$$

$$P(y_i = 2 \mid x_i; \mathbf{w}) = \frac{e^{\mathbf{w}_2^T \cdot x^{(i)}}}{1 + \sum_{l=1}^{K-1} e^{\mathbf{w}_l^T \cdot x_i}}$$

$$\cdots$$

$$P(y_i = K-1 \mid x_i; \mathbf{w}) = \frac{e^{\mathbf{w}_{K-1}^T \cdot x_i}}{1 + \sum_{l=1}^{K-1} e^{\mathbf{w}_l^T \cdot x_i}}$$

Then to learn the parameters θ, we can also use gradient descent to maximize the log-likelihood:

$$L(\mathbf{w}) = \log \prod_{i=1}^{N} p(y_i \mid x_i; \mathbf{w})$$

$$= \sum_{i=1}^{N} \log \prod_{l=1}^{k} \left(\frac{e^{\mathbf{w}_l^T \cdot x_i}}{\sum_{j=1}^{k} e^{\mathbf{w}_j^T \cdot x_i}} \right)^{1\{y_i = l\}}$$

indicator function takes on a value of 1 if its argument is true, and 0 otherwise.

A review of classification models

- Perceptron
- Logistic Regression
- Multinomial Logistic Regression
- k-Nearest Neighbor
- Decision Tree
- Bayesian Classification
- Neural Networks

K-Nearest Neighbor Learning

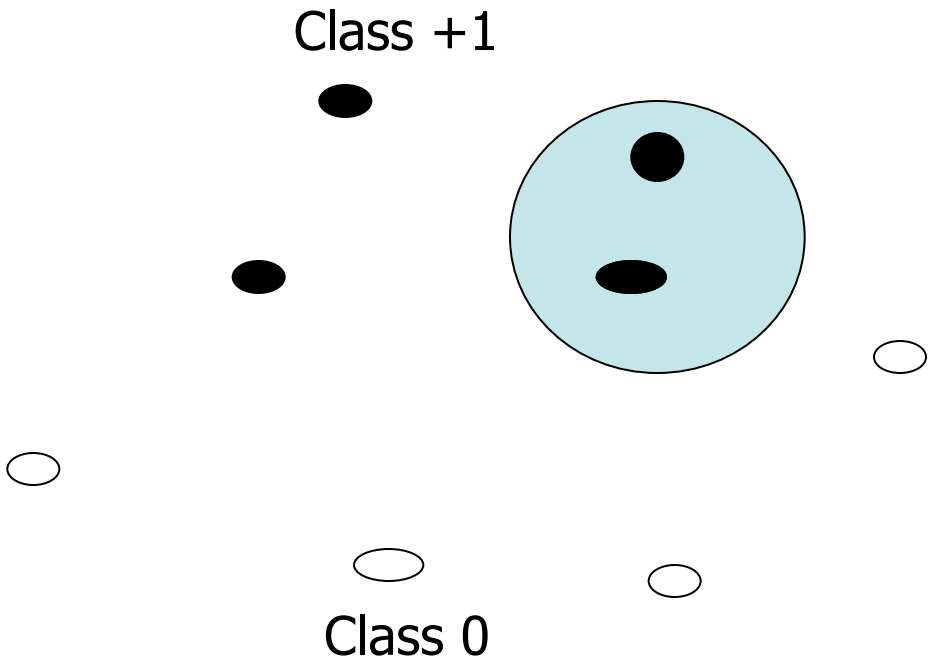
k-NN is a type of instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until classification

- 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.

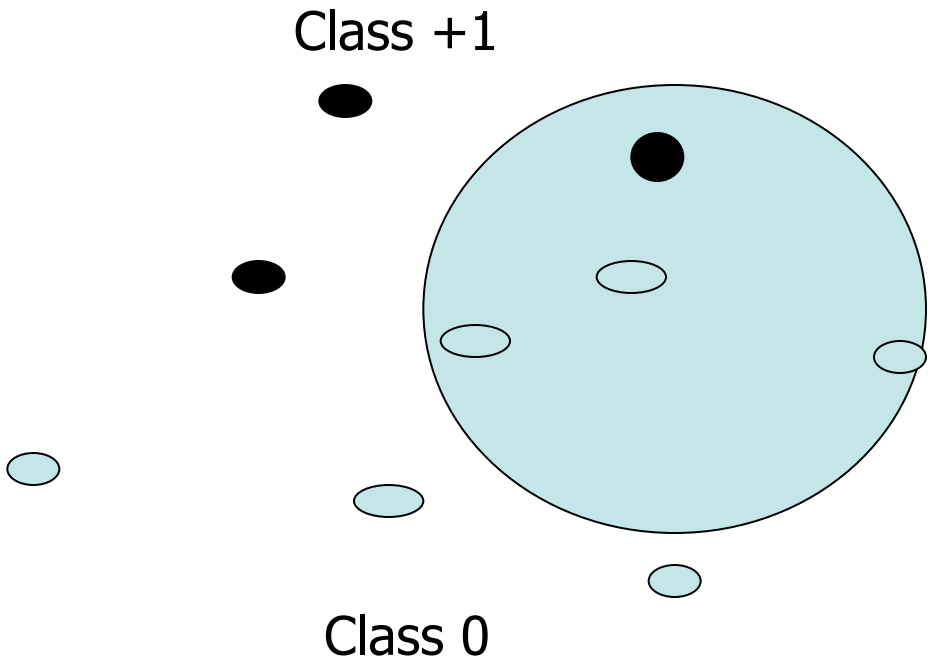
- Properties of kNN
  - No explicit hypothesis
  - No training stage
  - Distance function

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

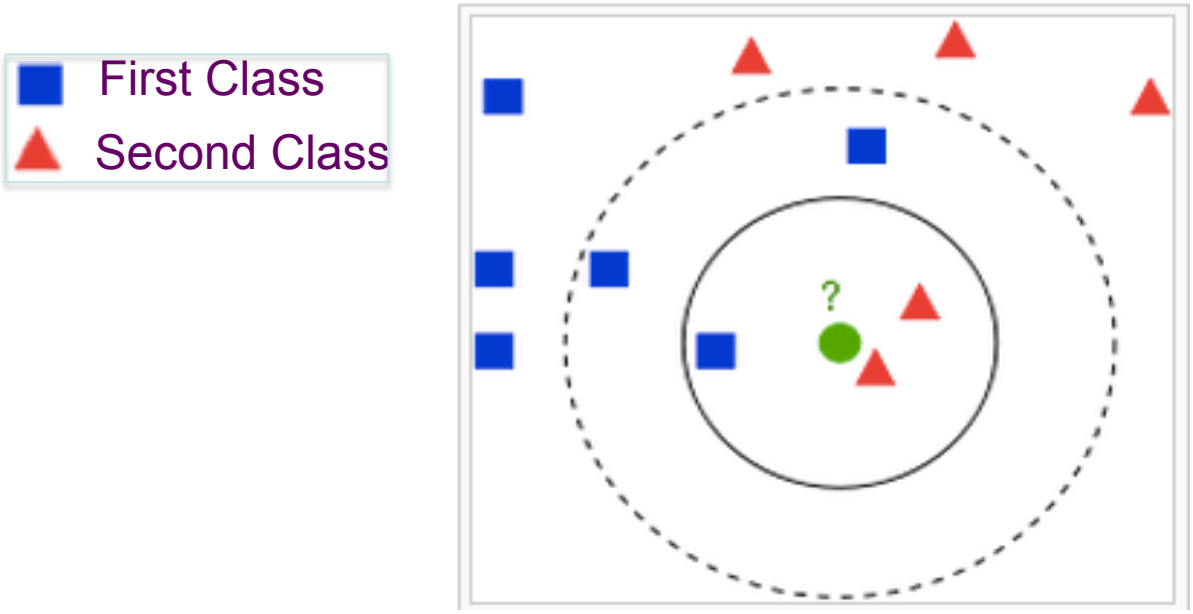
1-Nearest Neighbor



3-Nearest Neighbor



Properties

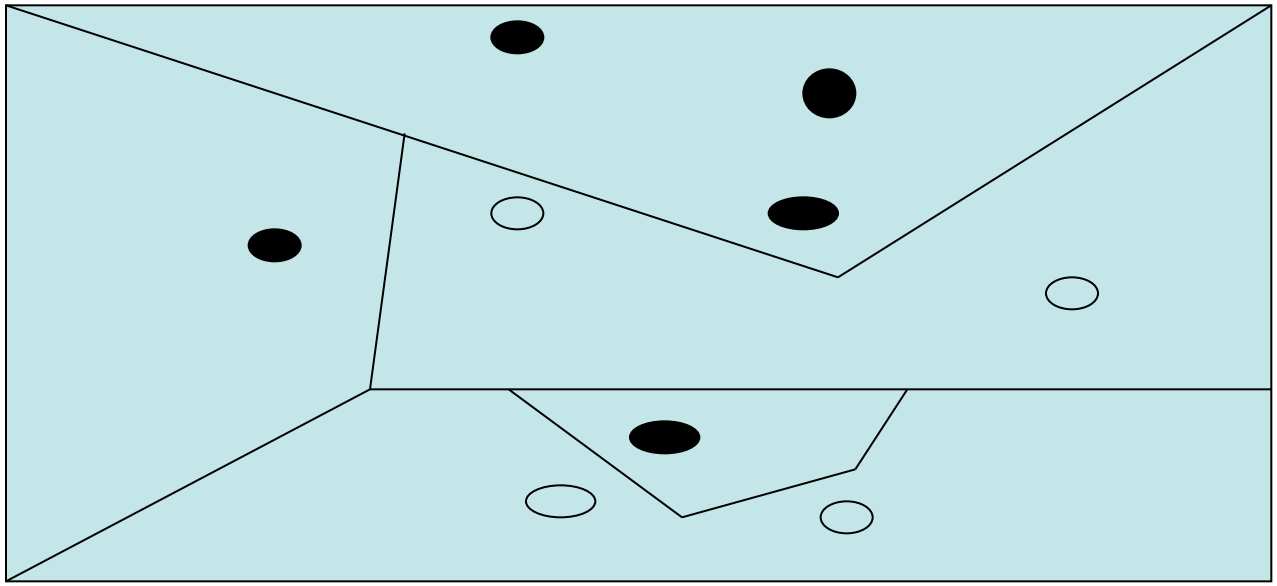


The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles.

- 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).

Voronoi Diagram

- Decision surface formed by the training examples



$$d(x_i, x_j) = \sqrt{(|x_{i1} - x_{j1}|^2 + |x_{i2} - x_{j2}|^2 + ... + |x_{ip} - x_{jp}|^2)}$$

A review of classification models

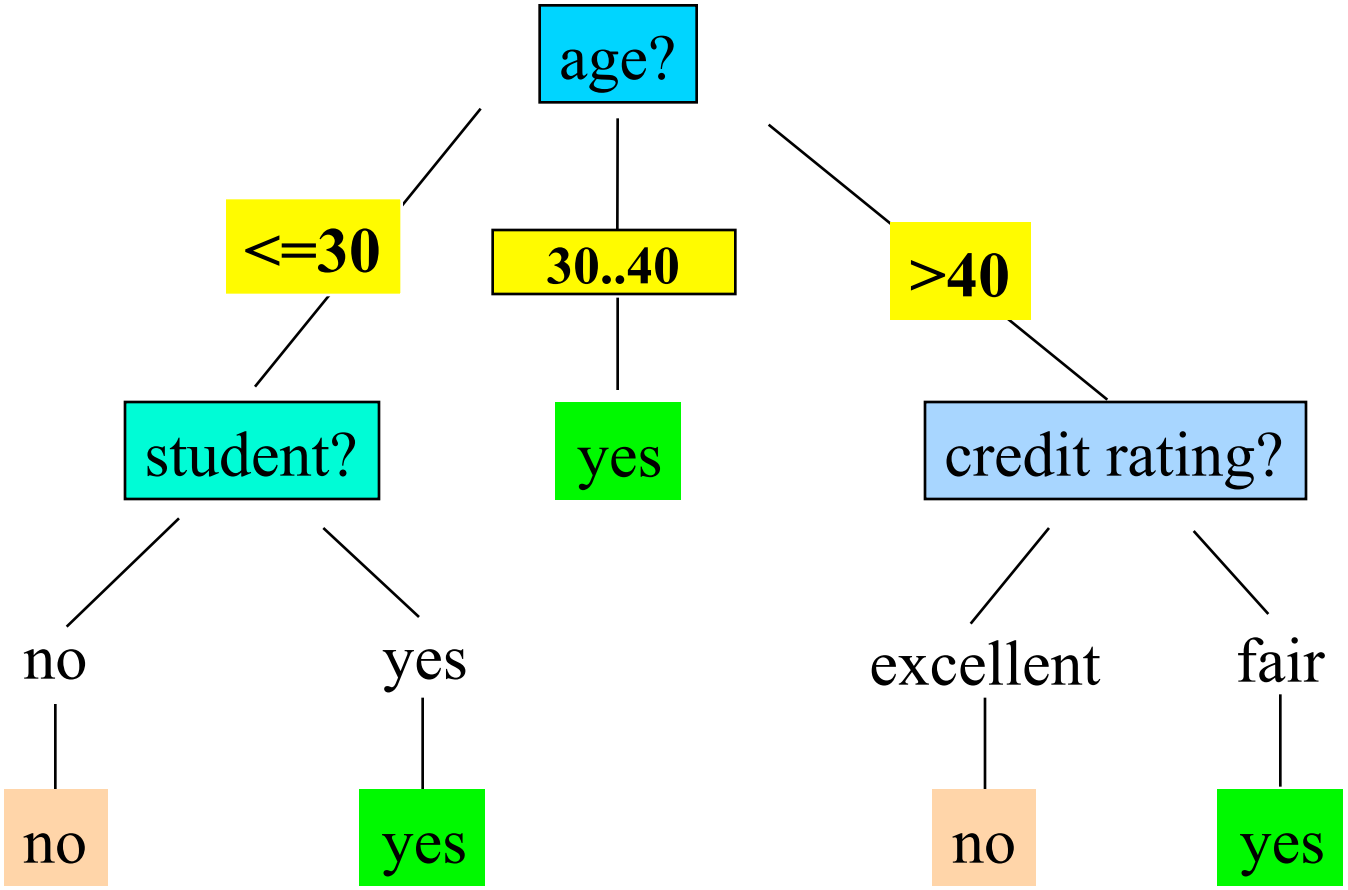
- Perceptron
- Logistic Regression
- Multinomial Logistic Regression
- k-Nearest Neighbor
- Decision Tree
- Bayesian Classification
- Neural Networks

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*"

A review of classification models

- Perceptron
- Logistic Regression
- Multinomial Logistic Regression
- k-Nearest Neighbor
- Decision Tree
- Bayesian Classification
- Neural Networks

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:

$$P(x_i \mid y) = \prod_{j=1}^{p} P(x_{ij} \mid y)$$

- 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"

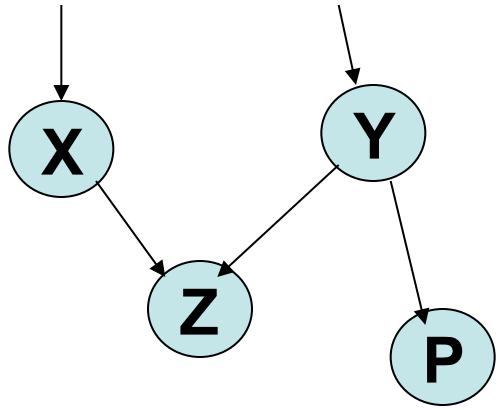
Naïve Bayesian Classifier: Comments

- 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

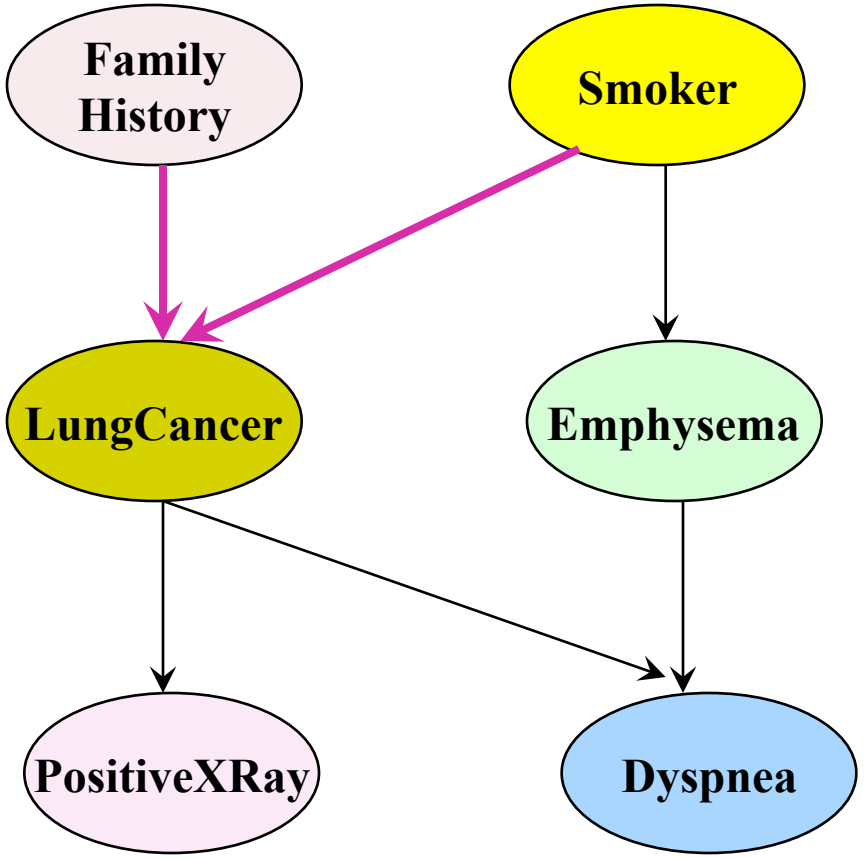
Bayesian Belief Networks

- Bayesian belief network allows a *subset* of the variables conditionally independent
- A graphical model of causal relationships
  - Represents dependency among the variables
  - Gives a specification of joint probability distribution



- ❑Nodes: random variables
- ❑Links: dependency
- ❑X,Y are the parents of Z, and Y is the parent of P
- ❑No dependency between Z and P
- ❑Has no loops or cycles

Bayesian Belief Network: An Example



Bayesian Belief Networks

	(FH, S)	(FH, ~S)	(~FH, S)	(~FH, ~S)
LC	0.8	0.5	0.7	0.1
~LC	0.2	0.5	0.3	0.9

The conditional probability table for the variable LungCancer:

Shows the conditional probability for each possible combination of its parents

$$P(z1,\ldots,zn) = \prod_{i=1}^{n} P(z_i \mid Parents(Z_i))$$

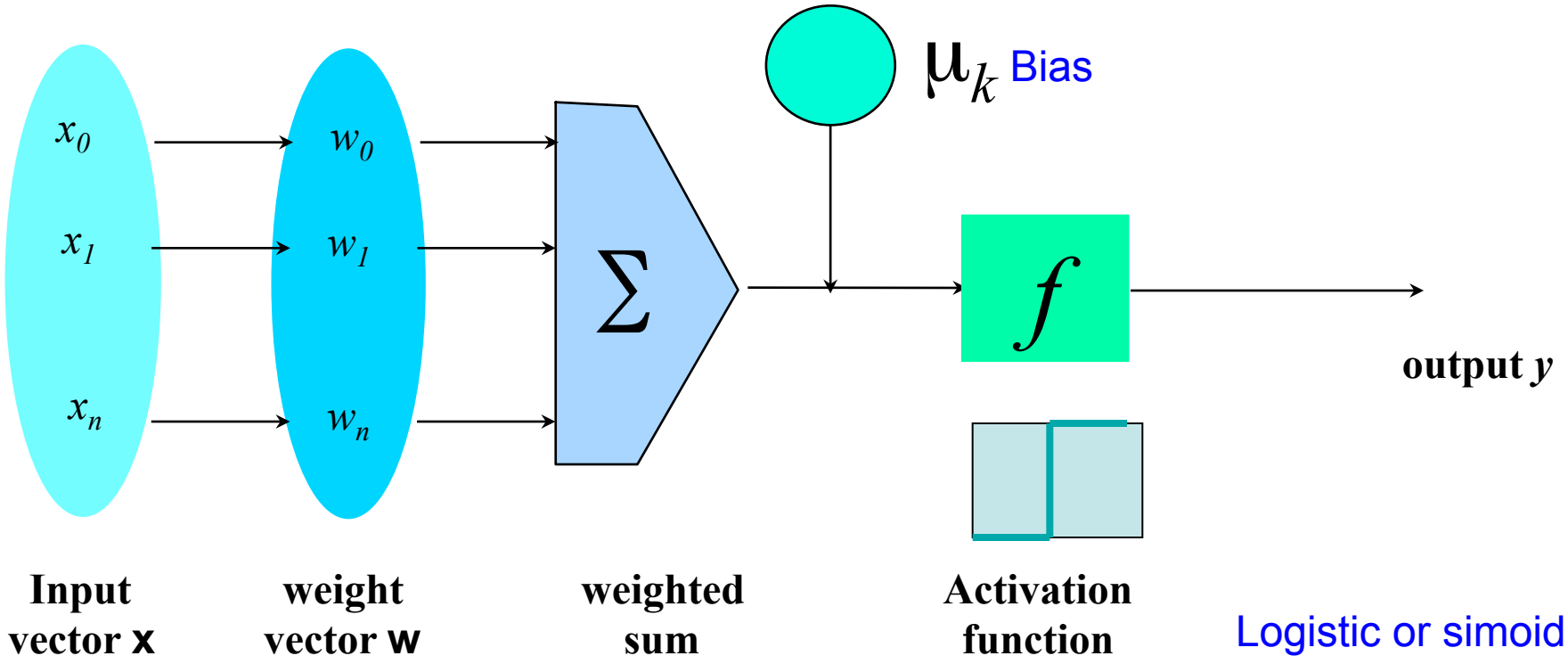
Learning Bayesian Networks

- Several cases
  - Given both the network structure and all variables observable: learn only the CPTs
  - Network structure known, some hidden variables: method of gradient descent, analogous to neural network learning
  - Network structure unknown, all variables observable: search through the model space to reconstruct graph topology
  - Unknown structure, all hidden variables: no good algorithms known for this purpose
- D. Heckerman, Bayesian networks for data mining

A review of classification models

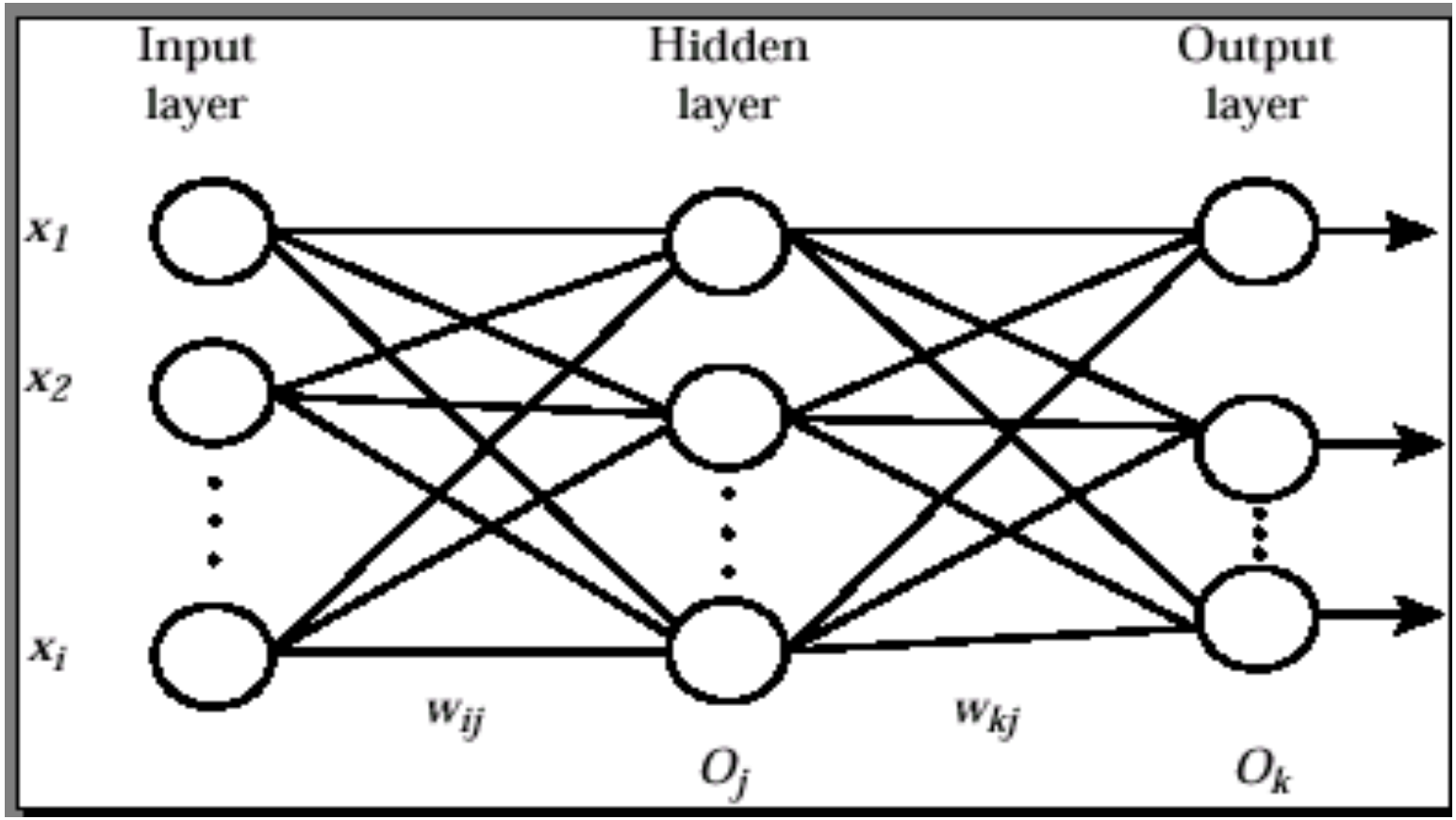
- Perceptron
- Logistic Regression
- Multinomial Logistic Regression
- k-Nearest Neighbor
- Decision Tree
- Bayesian Classification
- Neural Networks

A Neuron (= a perceptron)



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

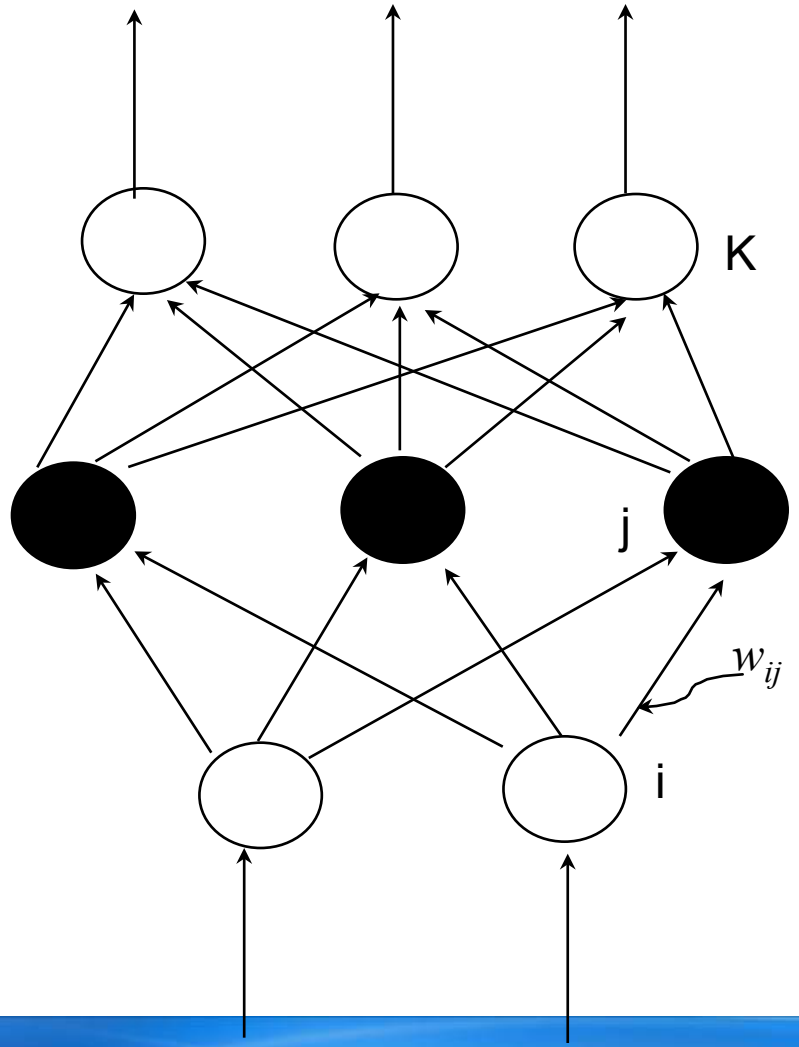
Multi layer Neural Network



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



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

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

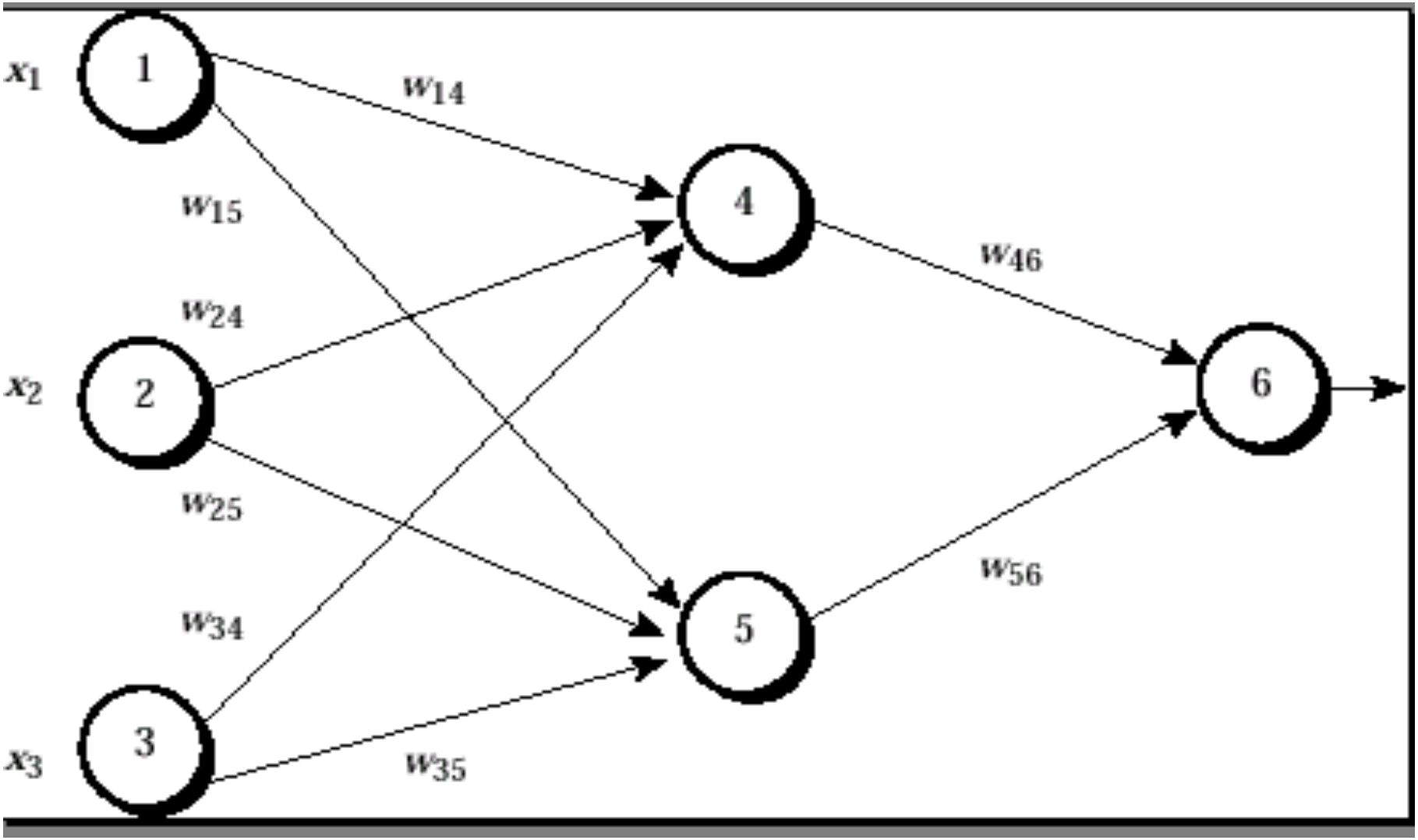
$$Err_j = O_j(1 - O_j)(T_j - O_j)$$

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

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

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

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

Basic ML Algorithm Review

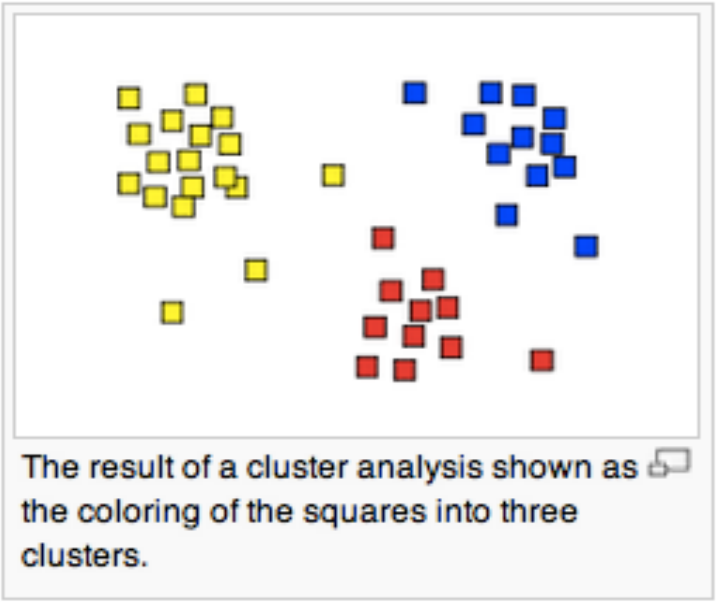
- Basic algorithms review
  - Data representation
  - Supervised learning
    - Evaluation
    - Regression, Perceptron, Bayesian classificaiton, Decision tree, etc.
  - Unsupervised learning
    - K-means, K-medoid
  - Semi-supervised learning
    - Learning with labeled and unlabeled data
    - Active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

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.

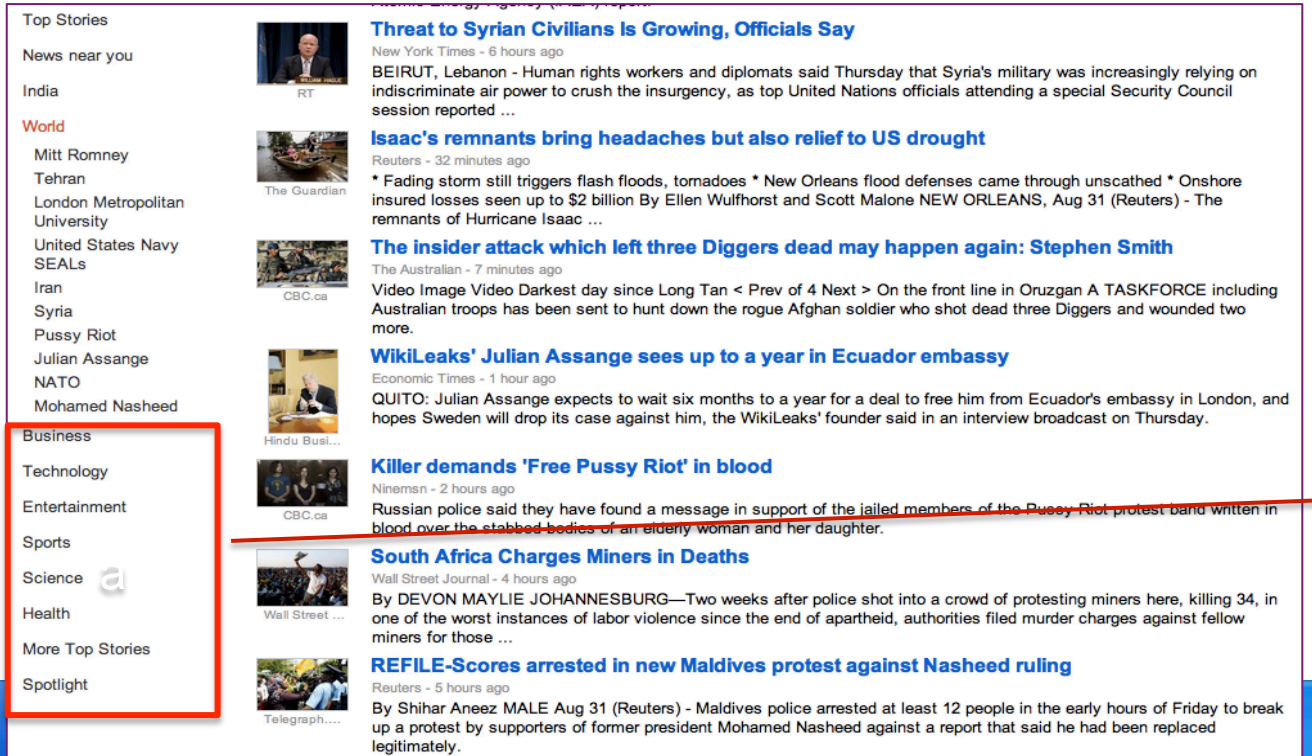
1. Social network analysis

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.

Partitioning Algorithms: Basic Concept

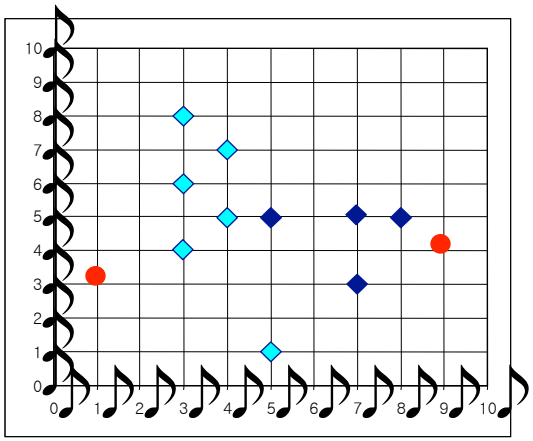
- Partitioning method: Construct a partition of a database ***D*** of ***N*** objects into a set of ***k*** clusters
- Given a *k*, find a partition of *k clusters* that optimizes the chosen partitioning criterion
  - Global optimal: exhaustively enumerate all partitions
  - Heuristic methods: *k-means* and *k-medoids* algorithms
  - *k-means* (MacQueen′67): Each cluster is represented by the center of the cluster
  - *k-medoids* or PAM (Partition around medoids) (Kaufman & Rousseeuw′87): Each cluster is represented by one of the objects in the cluster

The *K-Means* Clustering Method

- Given into *k* nonempty subsets
  - Compute seed *k*, the *k-means* algorithm is implemented in four steps:
  - Partition objects points as the centroids of the clusters of the current partition (the centroid is the center, i.e., *mean point*, of the cluster)
  - Assign each object to the cluster with the nearest seed point
  - Go back to Step 2, stop when no more new assignment

The *K-Means* Clustering Method

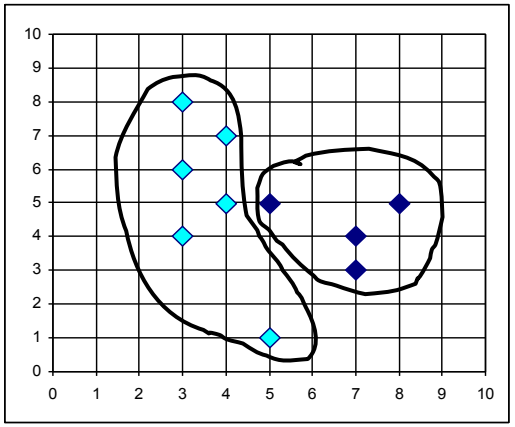
- Example



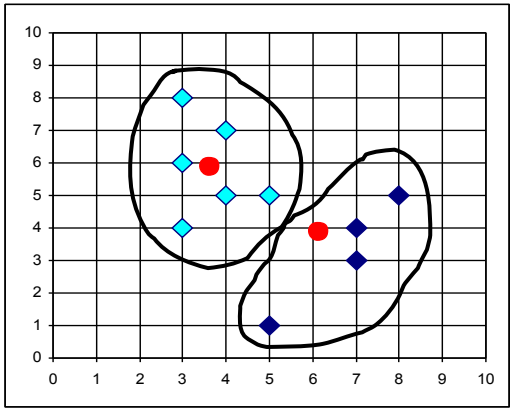
K=2

Arbitrarily choose K object as initial cluster center

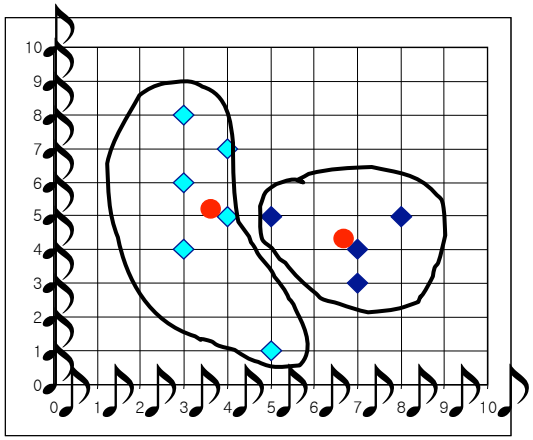
Assign each objects to most similar center



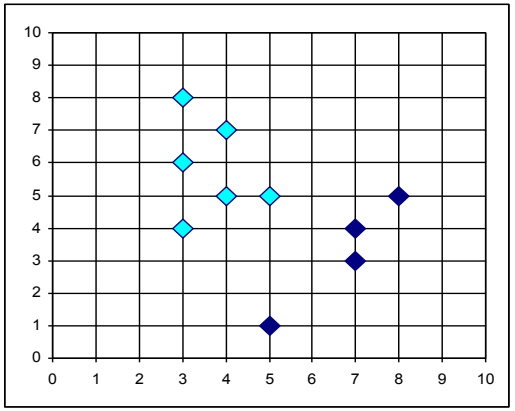
reassign



Update the cluster means



reassign



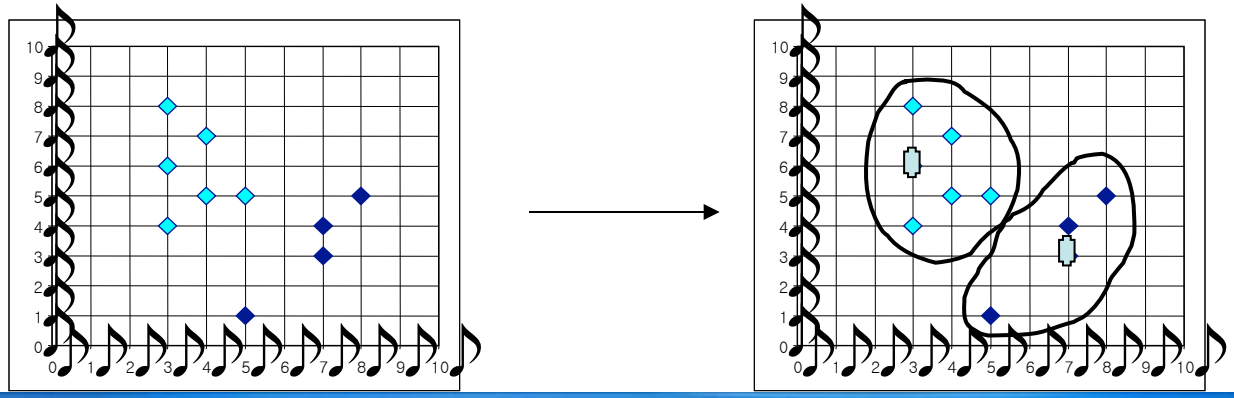
Update the cluster means

Comments on the *K-Means* Method

- Strength: *Relatively efficient*: *O*(*tkn*), where *n* is # objects, *k* is # clusters, and *t* is # iterations. Normally, *k*, *t* << *n*.
  - Comparing: PAM: $O(k(n-k)^2)$, CLARA: $O(ks^2 + k(n-k))$
- Comment: Often terminates at a *local optimum*. The *global optimum* may be found using techniques such as: *deterministic annealing* and *genetic algorithms*
- Weakness
  - Applicable only when *mean* is defined, then what about categorical data?
  - Need to specify *k,* the *number* of clusters, in advance
  - Unable to handle noisy data and *outliers*
  - Not suitable to discover clusters with *non-convex shapes*

The *K-Medoids* Clustering Method

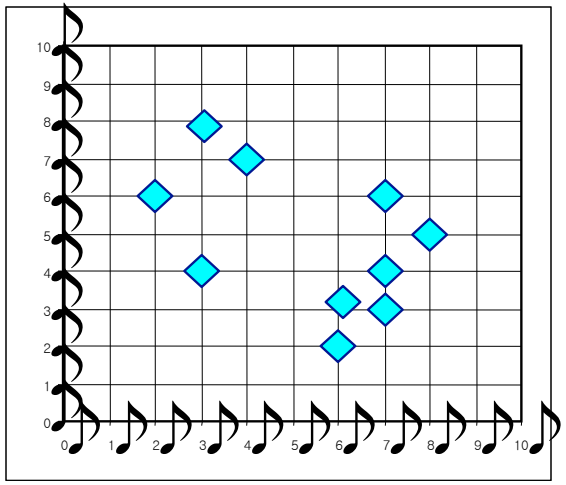
- The k-means algorithm is sensitive to outliers !
  - Since an object with an extremely large value may substantially distort the distribution of the data.
- K-Medoids: Instead of taking the **mean** value of the object in a cluster as a reference point, **medoids** can be used, which is the **most centrally located** object in a cluster.



The *K-Medoids* Clustering Method

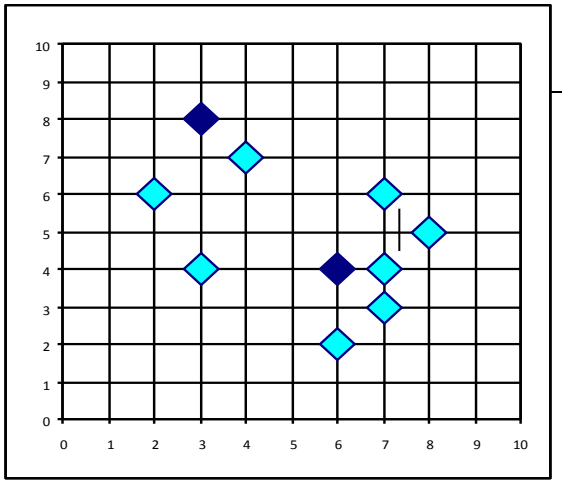
- Find *representative* objects, called medoids, in clusters
- *PAM* (Partitioning Around Medoids, 1987)
  - starts from an initial set of medoids and iteratively replaces one of the medoids by one of the non-medoids if it improves the total distance of the resulting clustering
  - *PAM* works effectively for small data sets, but does not scale well for large data sets
- *CLARA* (Kaufmann & Rousseeuw, 1990)
- *CLARANS* (Ng & Han, 1994): Randomized sampling
- Focusing + spatial data structure (Ester et al., 1995)

Typical k-medoids algorithm

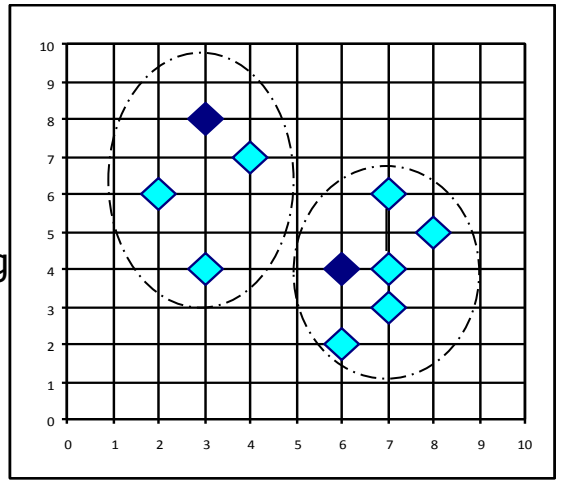


K=2

Arbitrary choose k object as initial medoids

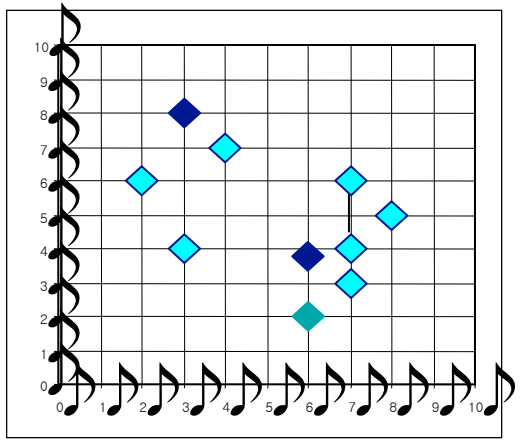


Assign each remaining object to nearest medoids

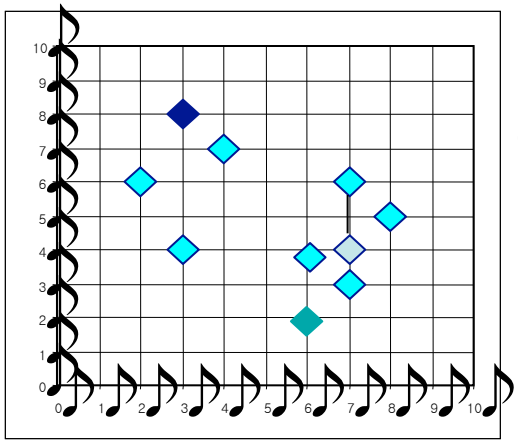


Total Cost = 20

Randomly select a nonmedoid object, O_{ramdom}



Compute total cost of swapping



Total Cost = 26

Swapping O and O_{ramdom}

If quality is improved.

Do loop

Until no change

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

Learning from labeled and unlabeled data

Two typical setting:

Semi-supervised classification

Training data *l* labeled instances $\{(\vec{x}_i, y_i)\}_{i=1}^{l}$ and *u* unlabeled instances $\{\vec{x}_j\}_{j=l+1}^{l+u}$; of $u \gg l$

Goal: Learn a better classifier *f* by using both labeled and unlabeled instances than using labeled data alone.

Constrained clustering

Unlabeled instance , and "constraints" between sets(pairs) of instances $\vec{x}_j{}_{j=1}^{l}$, often in the form of cannot-link , e.g. x_i and x_j cannot be in the same cluster and must-link constraints, e.g, x_i and x_j must be in the same cluster.

Goal: better clustering than from unlabeled data alone

Motivation for Semi-supervised Learning

- In speech recognition

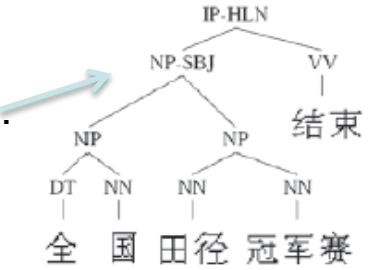
An instance is a speech utterance, the label is the corresponding transcript.

film ⇒ f ih_n uh_gl_n m
be all ⇒ bcl b iy iy_tr ao_tr ao l_dl

Accurate transcription by human expert annotators can be extremely time consuming: it took as long as 400 hours to transcribe 1 hour of speech at the phonetic level

- In natural language parsing

An instance is a sentence, the label is the corresponding parse tree.



V – Verb
N- Noun

Tree-banks are time consuming to construct: for a mere 4000 sentences in Penn Chinese Treebank, experts took two years to manually create the corresponding parse trees.

- Labeling tends to expensive, while, unlabeled data is available in large quantity and easy to collect, e.g.:

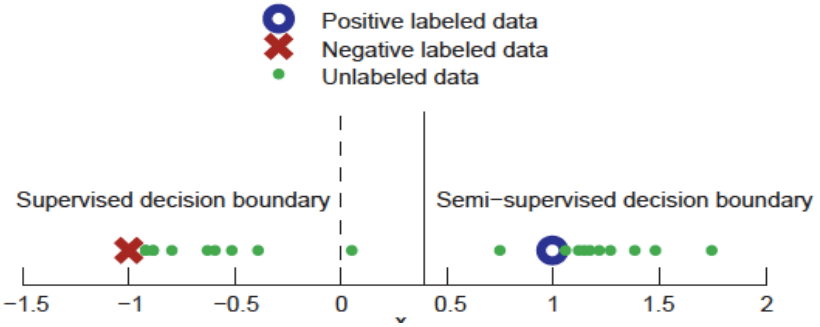
Speech utterances can be recorded from radio broadcasts.
Text sentences can be crawled from WWW
…

Solution:

Semi-Supervised learning can utilize both labeled and unlabeled data to achieve better performance than supervised learning.

How using “Unlabeled Data” Impact the decision boundary?

Give a simple example: each instance is a one-dimensional feature x. The goal is to classify instance into positive and negative.



Consider only labeled data:

Only two labeled training instances (x1, y1)=(-1,-) and (x2,y2)=(1,+).
The best estimate of the decision boundary is obviously x=0

Consider also the unlabeled training data:

The unlabeled training data gives more information because they form two groups.

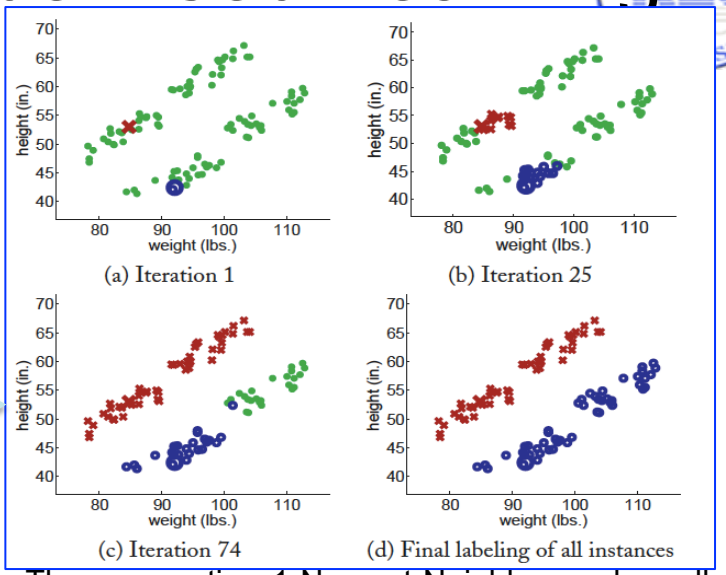
So the two labeled instances are not the most prototypical examples for the two classes.
In this way, the best estimate of the decision boundary should be $\mathbf{x} \approx 0.4$

Note: We don’t know the ground truth(for unlabelled data) from above figure

Simple Examples of Semi-supervised Learning

Self-training (Bootstrapping)

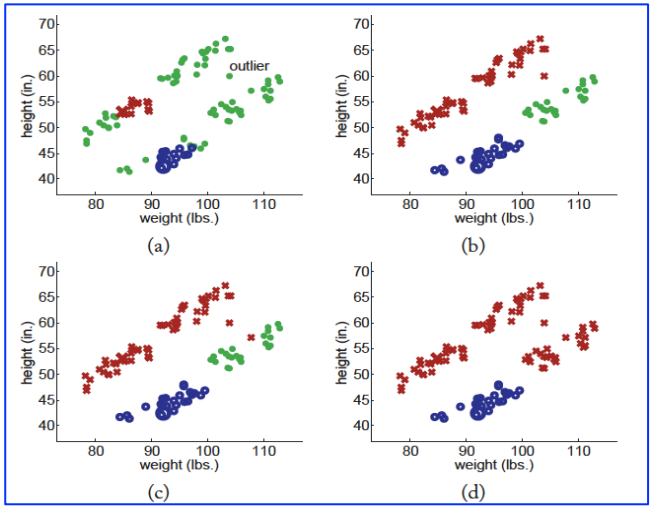
1. Initialise: $L = \{(\vec{x}_i, y_i)\}_{i=1}^{l}$, $U = \{\vec{x}_j\}_{j=l+1}^{l+u}$
2. Repeat:
3. Train f_i from L using supervised learning
4. Apply f_i to U using supervised learning
5. Identify a subset U' of U where $f_i(\vec{x}_j)$ is "confident"
6. $U \leftarrow U \setminus U'$, $L \leftarrow L \cup U''$ s.t. $U'' = \{(\vec{x}_j, f_i(\vec{x}_j)\}$
7. Until L is unchanged from one iteration to the next



Propagating 1-nearest neighbor:

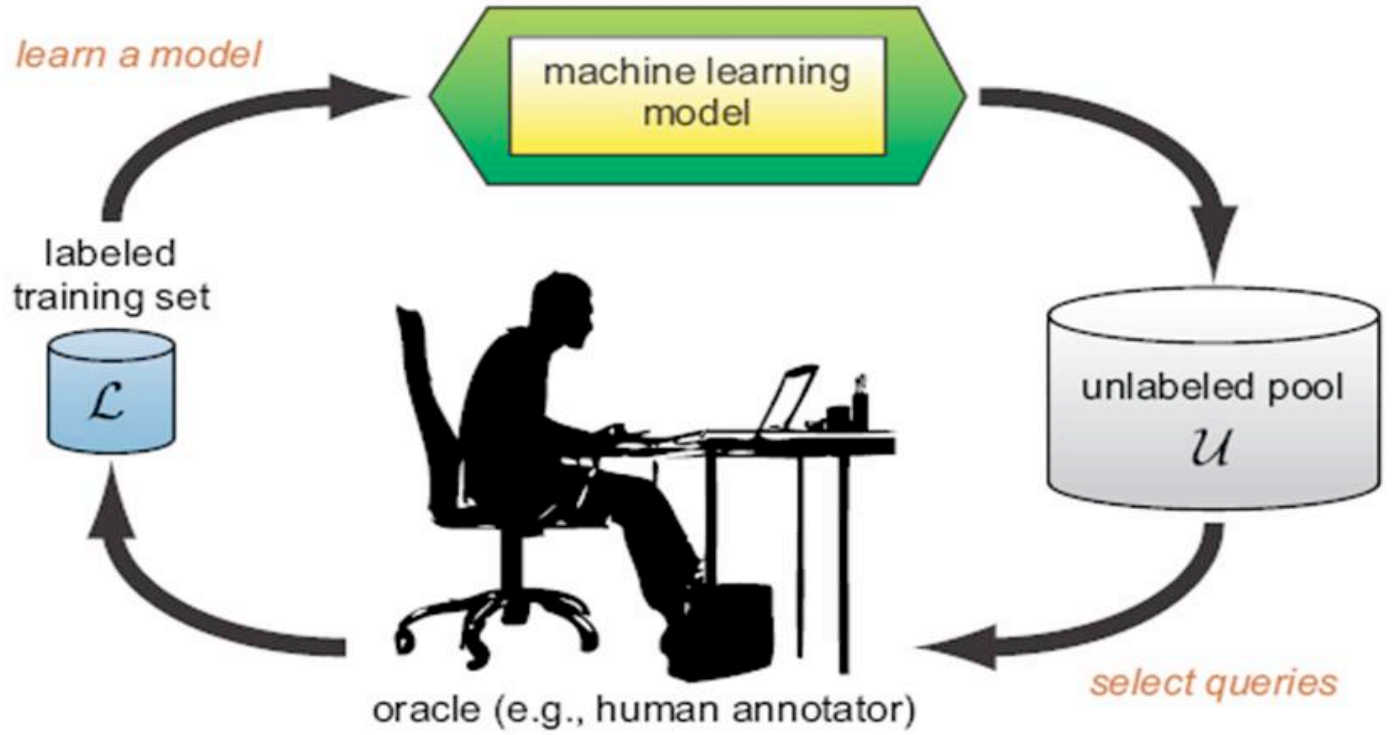
1. Initialise: $L = \{(\vec{x}_i, y_i)\}_{i=1}^{l}$, $U = \{\vec{x}_j\}_{j=l+1}^{l+u}$
2. Repeat:
3. Select $\vec{x}, \vec{x'} = \arg\min_{\vec{x} \in U, \vec{x'} \in L} \min d(\vec{x}, \vec{x'})$
4. $f_i(\vec{x}) = y'$
5. $U \leftarrow U \setminus \{\vec{x}\}$, $L \leftarrow L \cup \{(\vec{x}, f_i(\vec{x})\}$
6. Until $U = \phi$

The propagation 1-Nearest-Neighbor works well Because the classes form well-separated clusters.



The propagation 1-Nearest-Neighbor is highly sensitive to outliers that may lead to propagating incorrect information. So more than the single nearest neighbor should be considered in both selecting the next point or label as well as assigning it a label

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).

An Active Learning Motivation Example

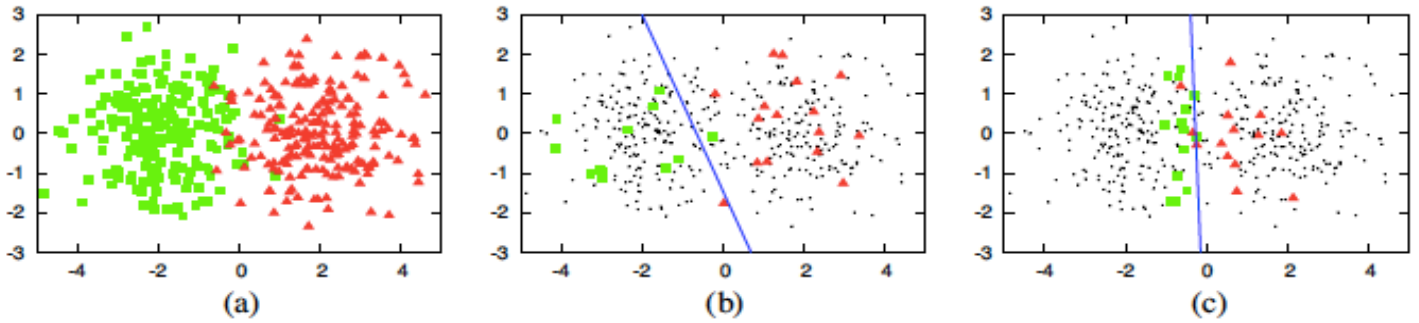
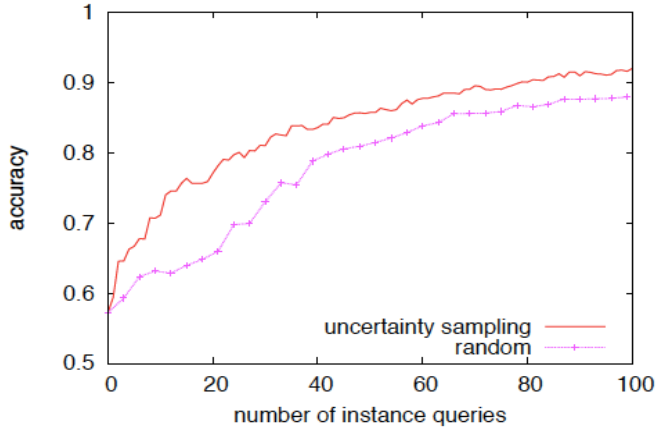
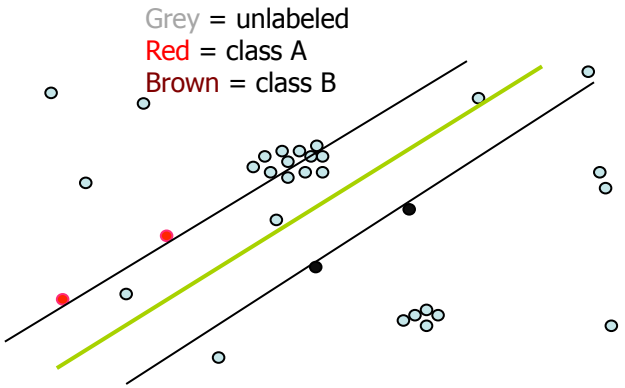


Figure 2: An illustrative example of pool-based active learning. (a) A toy data set of 400 instances, evenly sampled from two class Gaussians. The instances are represented as points in a 2D feature space. (b) A logistic regression model trained with 30 labeled instances randomly drawn from the problem domain. The line represents the decision boundary of the classifier (70% accuracy). (c) A logistic regression model trained with 30 actively queried instances using uncertainty sampling (90%).

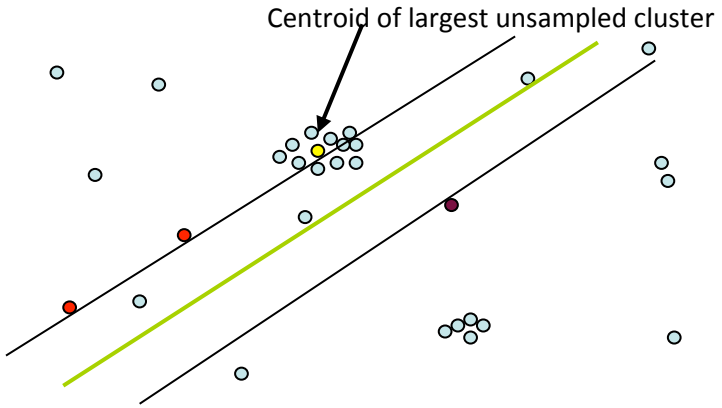


Query Strategy

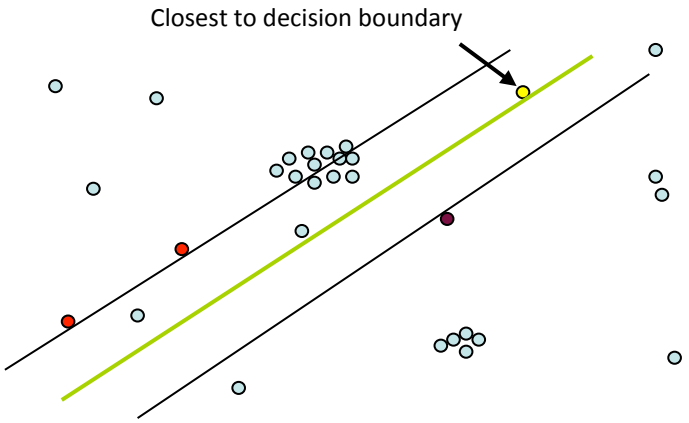
Which point to sample?



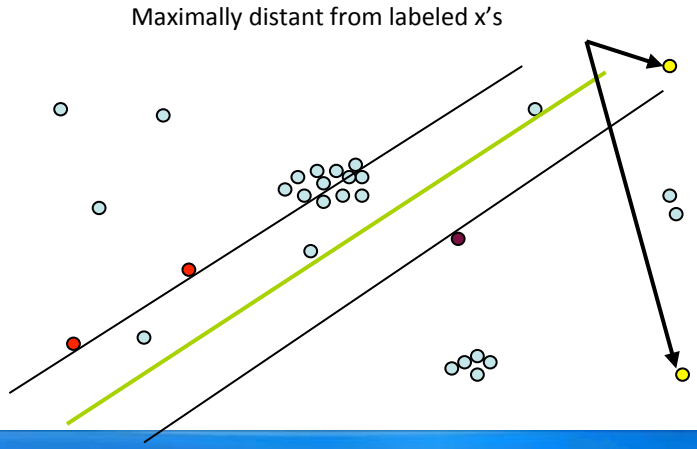
Density-based Sampling



Uncertainty Sampling



Maximal Diversity Sampling



Uncertainty Sampling

Key Idea: An active learner queries the instances about which it is least certain how to label.

Least Confident $x^*_{LC} = \underset{x}{\text{argmax}}\ 1 - P_\theta(\hat{y}|x),$

Where $\hat{y} = \text{argmax}_y P_\theta(y|x)$ or the class label with the highest posterior probability under the model θ

Margin Sampling $x^*_M = \underset{x}{\text{argmin}}\ P_\theta(\hat{y}_1|x) - P_\theta(\hat{y}_2|x)$

Where $\hat{y}_1$ anc $\hat{y}_2$ are the first and second most probable class labels under the model.
Least confident "throws away" information about the remaining label distribution. Margin sampling aims to solve this shortcoming.
The instances with small margins are more ambiguous, thus knowing the label would help the model discriminate more effectively between them.

Entropy $x^*_H = \underset{x}{\text{argmax}} -\sum_i P_\theta(y_i|x) \log P_\theta(y_i|x),$

Where y_i ranges over all possible labelings.
Entropy represents the amount of information needed to "encode" a distribution. Thus, it is often viewed as a measure of uncertainty or impurity.
Entropy can solve very large label sets.

Query-By-Committee Sampling

QBC maintains a committee C={θ1,θ2,…, θC} of models which are all trained on the current labeled set, but represent competing hypotheses.
Each committee member is then allowed to vote on the labelings of query candidates.
The most informative query is considered to be the instance about which they most disagree.

If we view machine learning as a search for the "best" model within the version space, then our goal in active learning is to constrain the size of this space as much as possible with a few labeled instances as possible.

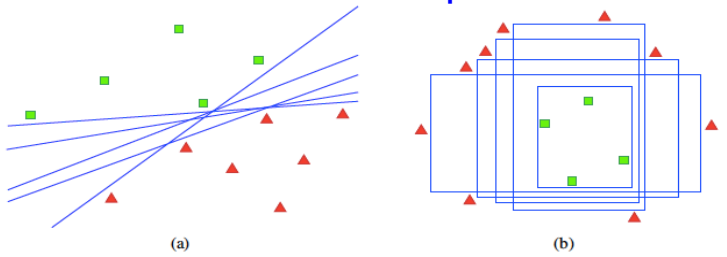


Figure 6: Version space examples for (a) linear and (b) axis-parallel box classifiers. All hypotheses are consistent with the labeled training data in $\mathcal{L}$ (as indicated by shaded polygons), but each represents a different model in the version space.

How to measure disagreement?

1. Vote entropy

$$x^*_{VE} = \underset{x}{\operatorname{argmax}} -\sum_i \frac{V(y_i)}{C} \log \frac{V(y_i)}{C}$$

Where y_i ranges over all possible labelings, and $V(y_i)$ is the number of "votes" that a label receives from among the committee members' predictions, and C is the committee size

2. KL divergence

$$x^*_{KL} = \underset{x}{\operatorname{argmax}} \frac{1}{C} \sum_{c=1}^{C} D(P_{\theta^{(c)}} \| P_{\mathcal{C}}),$$

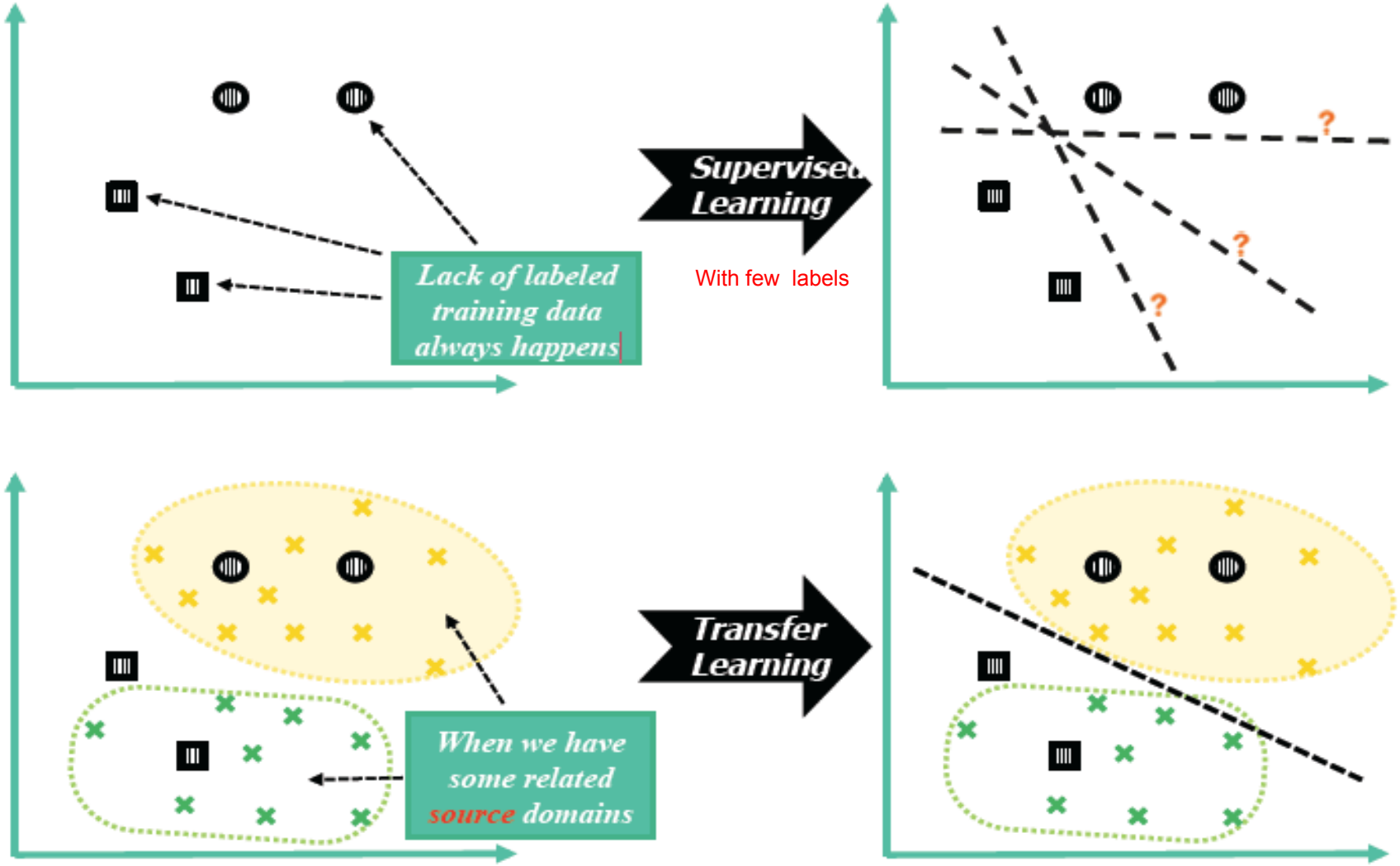
where:

$$D(P_{\theta^{(c)}} \| P_{\mathcal{C}}) = \sum_i P_{\theta^{(c)}}(y_i|x) \log \frac{P_{\theta^{(c)}}(y_i|x)}{P_{\mathcal{C}}(y_i|x)}.$$

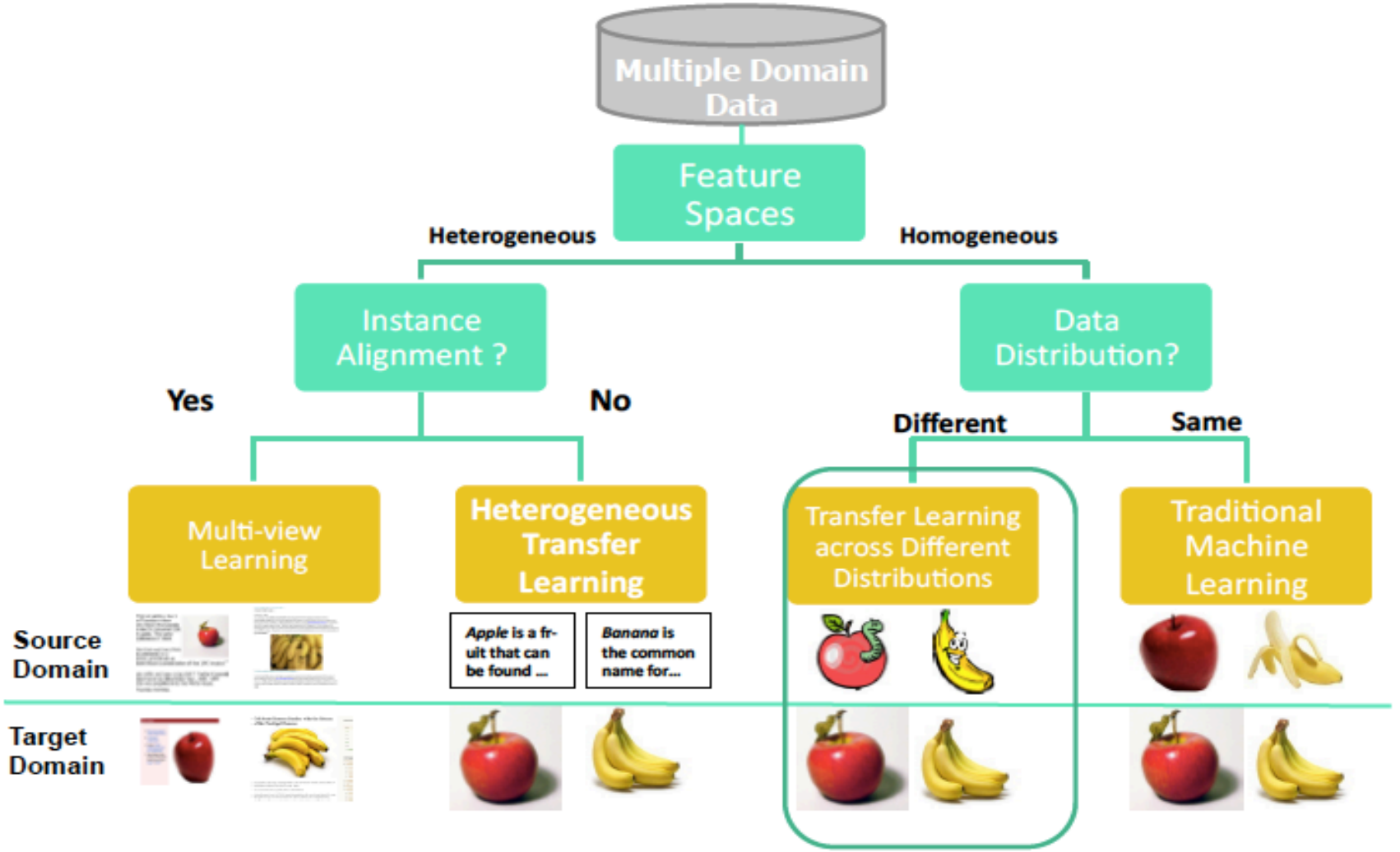
Where $\theta^{(c)}$ represents a particular model in the committee, and C represents the committee as a whole, thus $P_{\mathcal{C}}(y_i|x) = \frac{1}{C}\sum_{c=1}^{C} P_{\theta^{(c)}}(y_i|x)$ is the "consensus" probability that y_i is the correct label.

So the most informative query is the one with the largest average difference between the label distributions of any one committee member and the consensus.

Transfer Learning



The Types of Transfer Learning



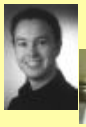
The State of Machine Learning

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: http://arnetminer.org/expertisesearch.do?keyword=Machine%20Learning

Thanks!

Jie Tang, DCST
http://keg.cs.tsinghua.edu.cn/jietang/
http://arnetminer.org
Email: jietang@tsinghua.edu.cn