

Machine Learning for Image Classification

----Part I: Traditional Approaches

Jianping Fan
Dept of Computer Science
UNC-Charlotte

Course Website:

<http://webpages.uncc.edu/jfan/itcs5152.html>

Machine Learning Problems

Supervised Learning

Unsupervised Learning

Discrete
Continuous

classification or
categorization

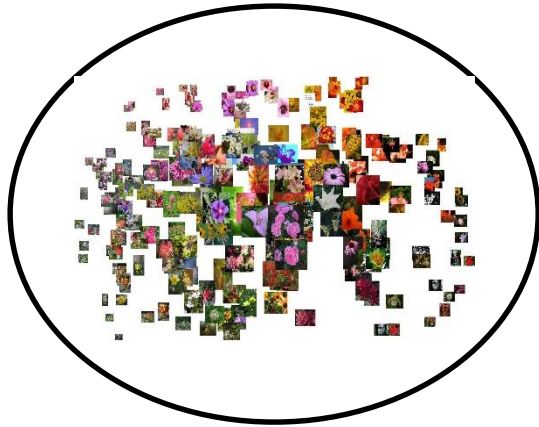
clustering

regression

dimensionality
reduction

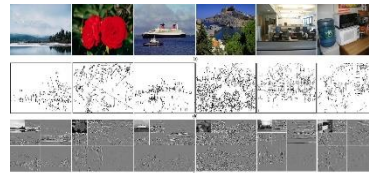
Pipeline for Traditional Image Classification System

Training Image Set

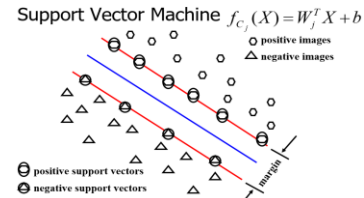


Offline Training
Online Testing

Feature Extraction



Classifier Training

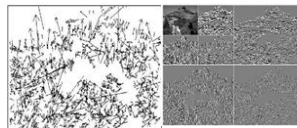


Classifier



Test Image

Feature Extraction



Classifier

Flower
Garden
Nature
.....

predictions

Machine Learning for Image Classification

- Apply a prediction function (classifier) to a feature representation of the image to get the desired output:

Prediction from Classifier
according to **Image Representation**

Classifier

$f(\text{apple image}) = \text{"apple"}$

$f(\text{tomato image}) = \text{"tomato"}$

$f(\text{cow image}) = \text{"cow"}$

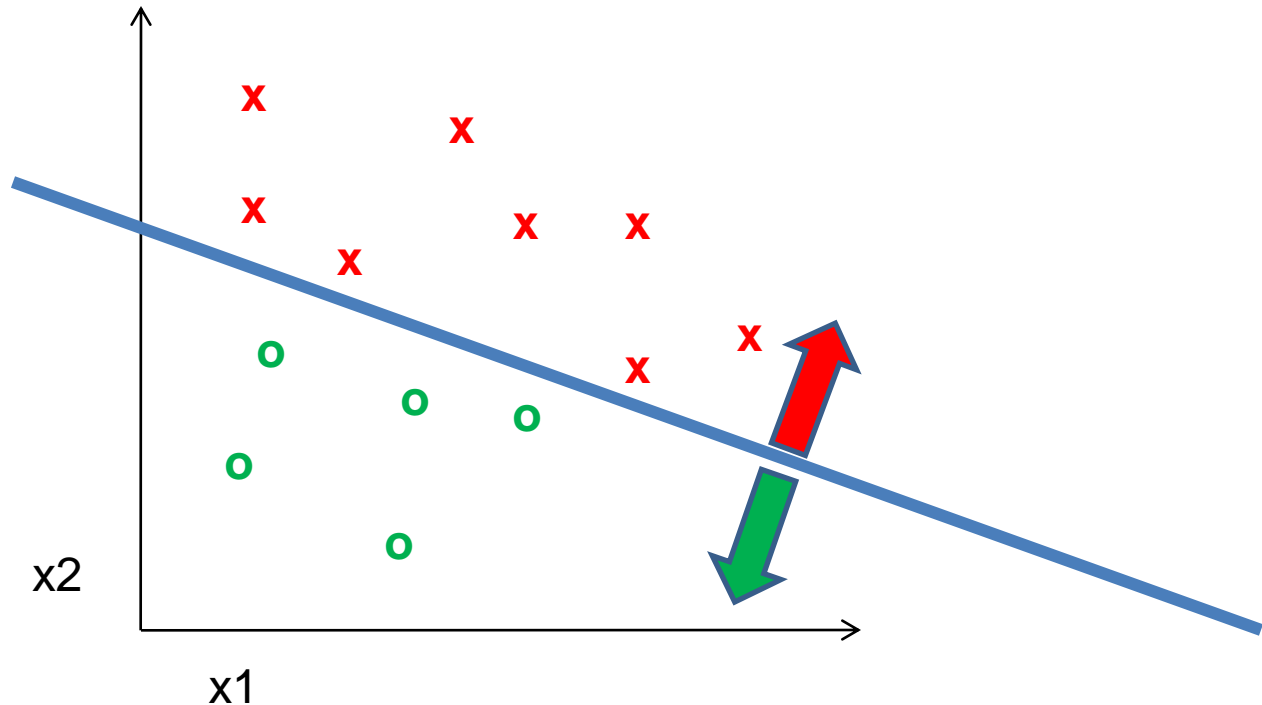
Feature Extraction for Image Representation

Various Types of Some Classifiers

- **K-nearest neighbor**
- **SVM**
- **Decision Trees**
- Neural networks
- **GMM & Naïve Bayes**
- Boosting
- Logistic regression
- Randomized Forests
- RBMs
- Etc.

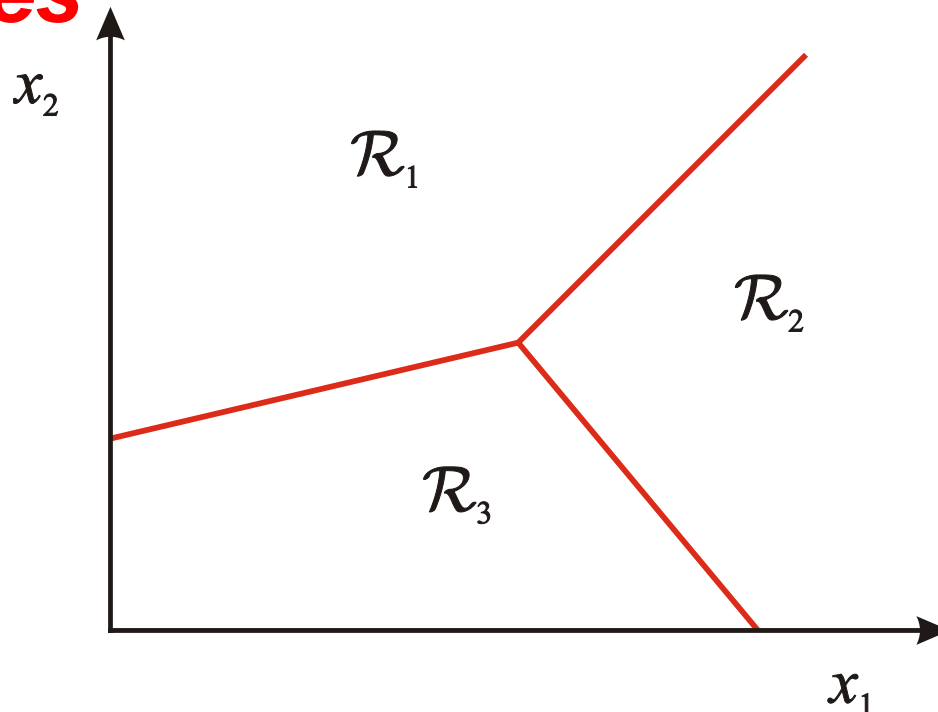
Learning a classifier

Given some set of features with corresponding labels, learn a function to predict the labels from the features



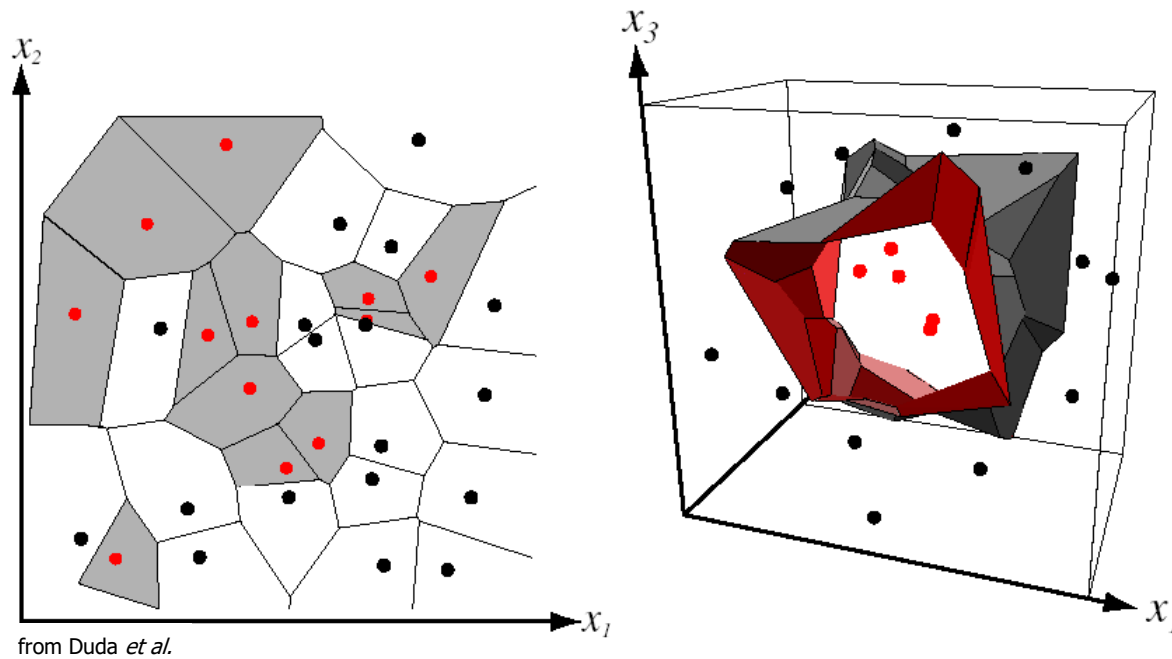
Classification

- Assign input vector to one of two or more classes
- Any decision rule divides input space into *decision regions* separated by **decision boundaries**



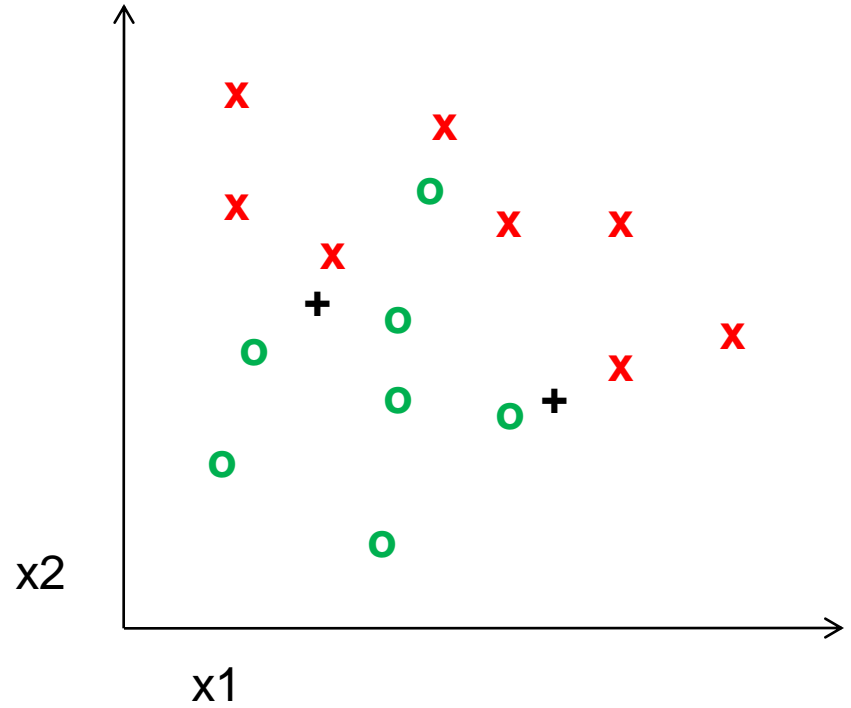
Nearest Neighbor Classifier

- Assign label of nearest training data point to each test data point

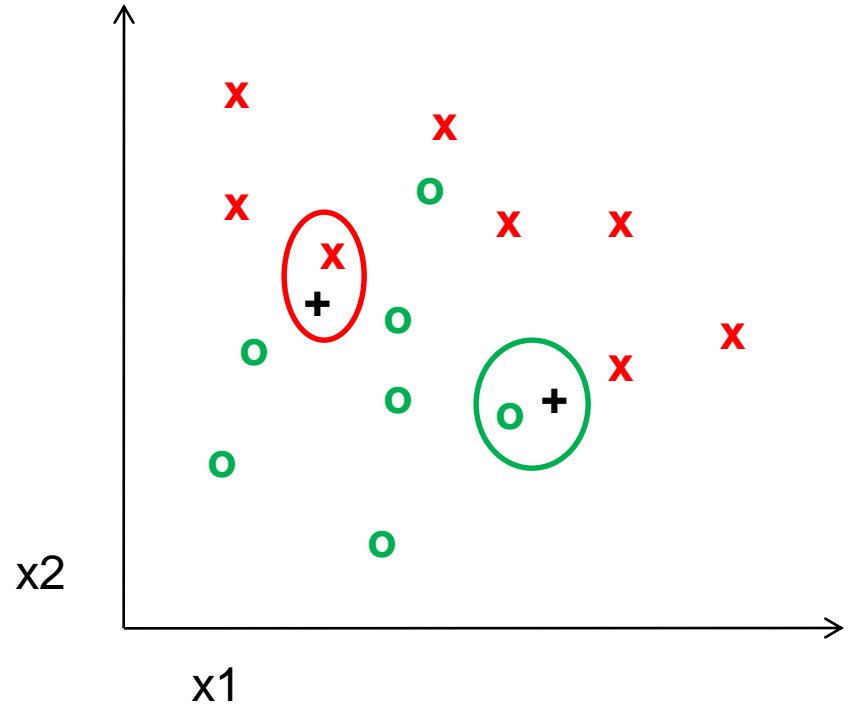


Voronoi partitioning of feature space
for two-category 2D and 3D data

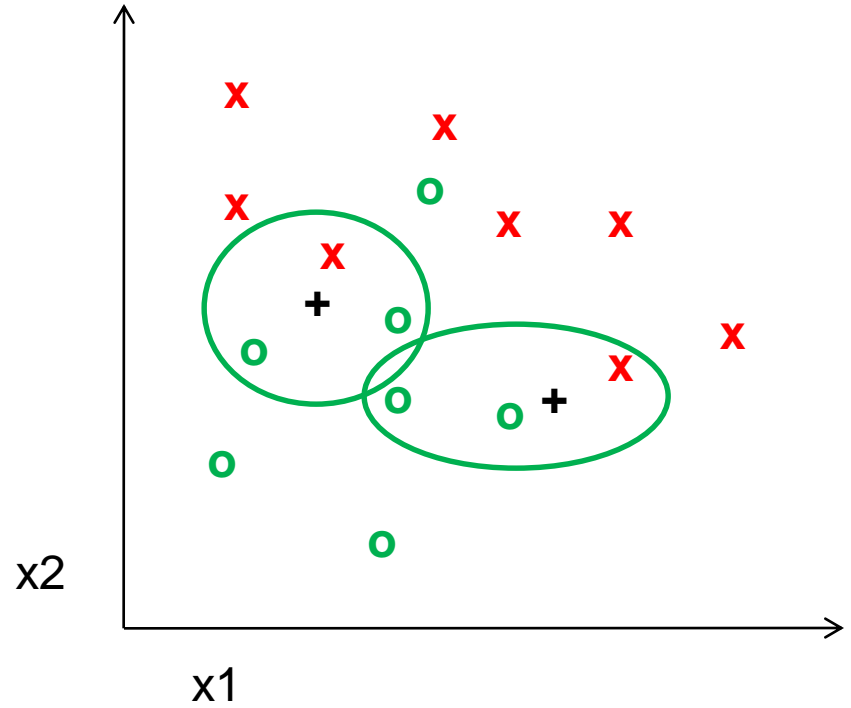
K-nearest neighbor



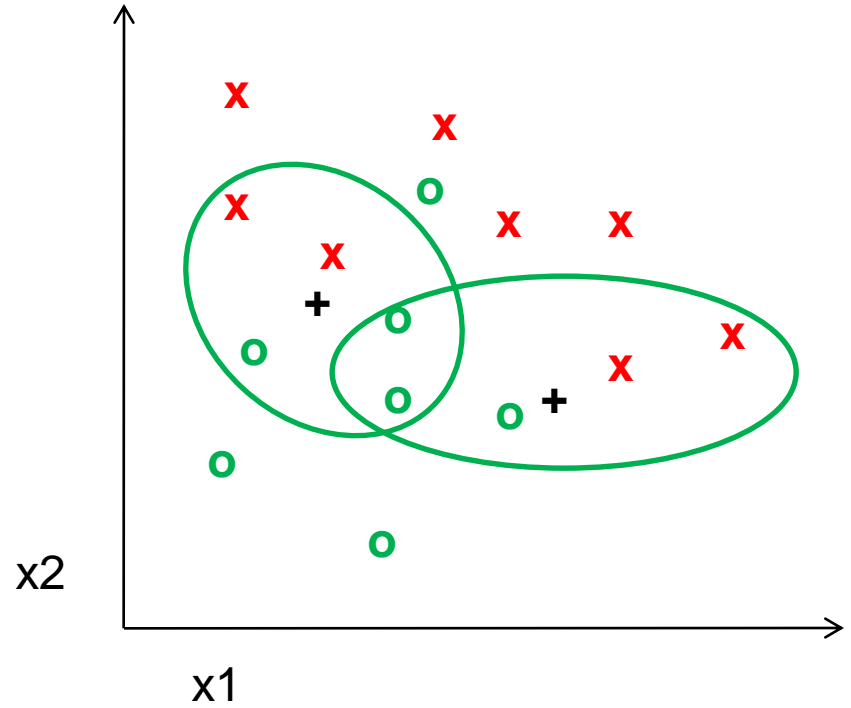
1-nearest neighbor



3-nearest neighbor



5-nearest neighbor



Ways of rescaling for KNN

Normalized L1 distance:

$$\Delta(X, Y) = \sum_{i=1}^n \delta(x_i, y_i)$$

where:

$$\delta(x_i, y_i) = \begin{cases} \text{abs}(\frac{x_i - y_i}{\text{max}_i - \text{min}_i}) & \text{if numeric, else} \\ 0 & \text{if } x_i = y_i \\ 1 & \text{if } x_i \neq y_i \end{cases}$$

Scale by IG:

$$\Delta(X, Y) = \sum_{i=1}^n w_i \delta(x_i, y_i)$$

$$w_i = H(C) - \sum_{v \in V_i} P(v) \times H(C|v)$$

Modified value distance metric:

$$\delta(v_1, v_2) = \sum_{i=1}^n |P(C_i|v_1) - P(C_i|v_2)|$$

Ways of rescaling for KNN

Dot product:

$$\Delta(X, Y) = dot_{max} - \sum_{i=1}^n w_i x_i y_i$$

Cosine distance:

$$\Delta(X, Y) = cos_{max} - \frac{\sum_{i=1}^n w_i x_i y_i}{\sqrt{\sum_{i=1}^n w_i x_i^2 \sum_{i=1}^n w_i y_i^2}}$$

TFIDF weights for text: for doc j, feature i: $x_i = tf_{i,j} * idf_i$:

#occur. of term i in doc j

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

#docs in corpus

$$idf_i = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|}$$

#docs in corpus that contain term i

Combining distances to neighbors

Standard KNN: $\hat{y} = \arg \max_y C(y, \text{Neighbors}(x))$

$$C(y, D') \equiv |\{(x', y') \in D' : y' = y\}|$$

Distance-weighted KNN:

$\hat{y} = \arg \max_y C(y, \text{Neighbors}(x))$

$$C(y, D') \equiv \sum_{\{(x', y') \in D' : y' = y\}} (\text{SIM}(x, x'))$$

$$C(y, D') \equiv 1 - \prod_{\{(x', y') \in D' : y' = y\}} (1 - \text{SIM}(x, x'))$$

$$\text{SIM}(x, x') \equiv 1 - \Delta(x, x')$$

Definition of Class Centroid

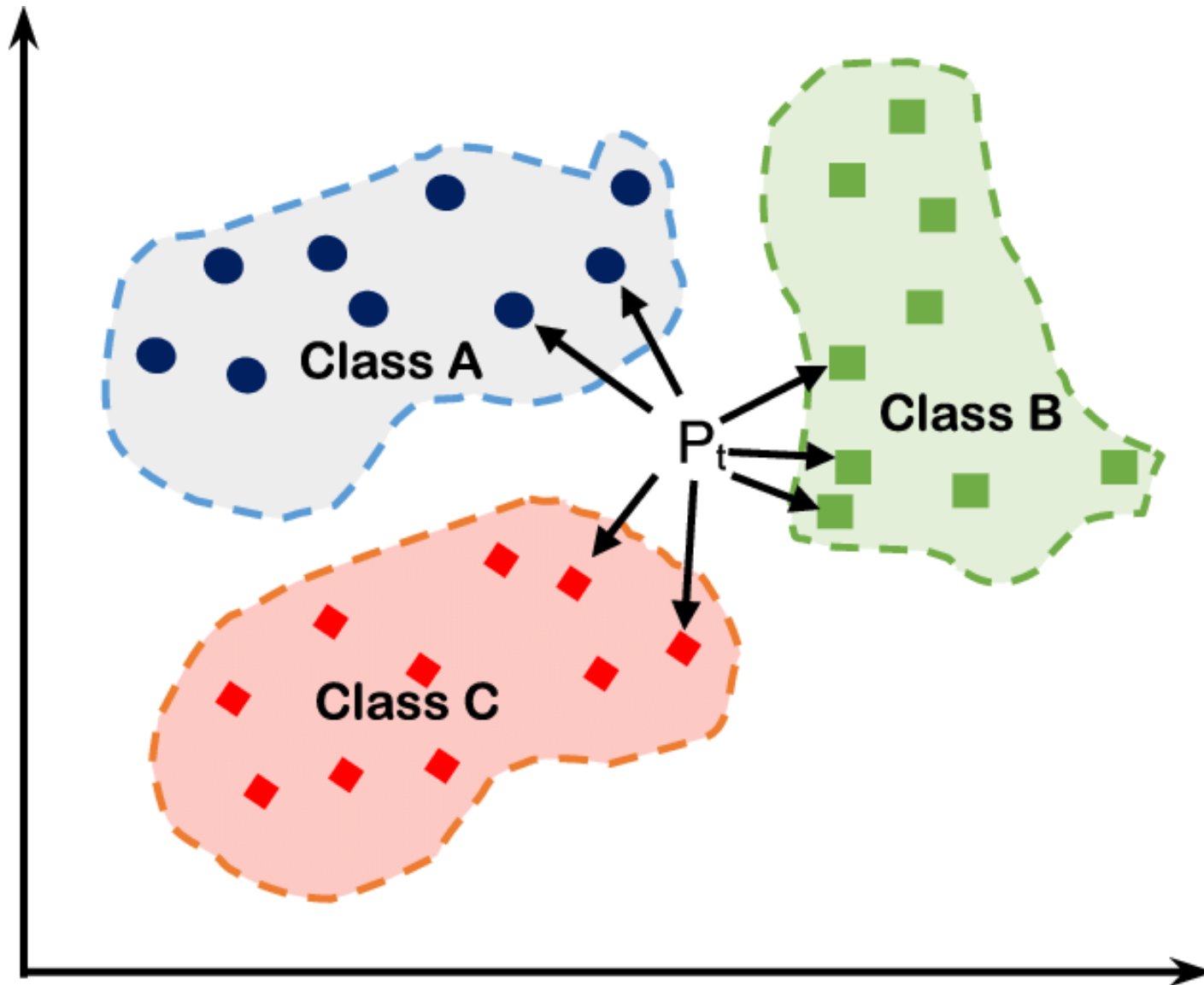
$$\vec{m}(c) = \frac{1}{|D_c|} \sum_{d \in D_c} \vec{v}(d)$$

- Where D_c is the set of all data points that belong to class c and $v(d)$ is the vector space representation of one specific data point d .
- *Note that centroid will in general not be a unit vector even when the inputs are unit vectors.*

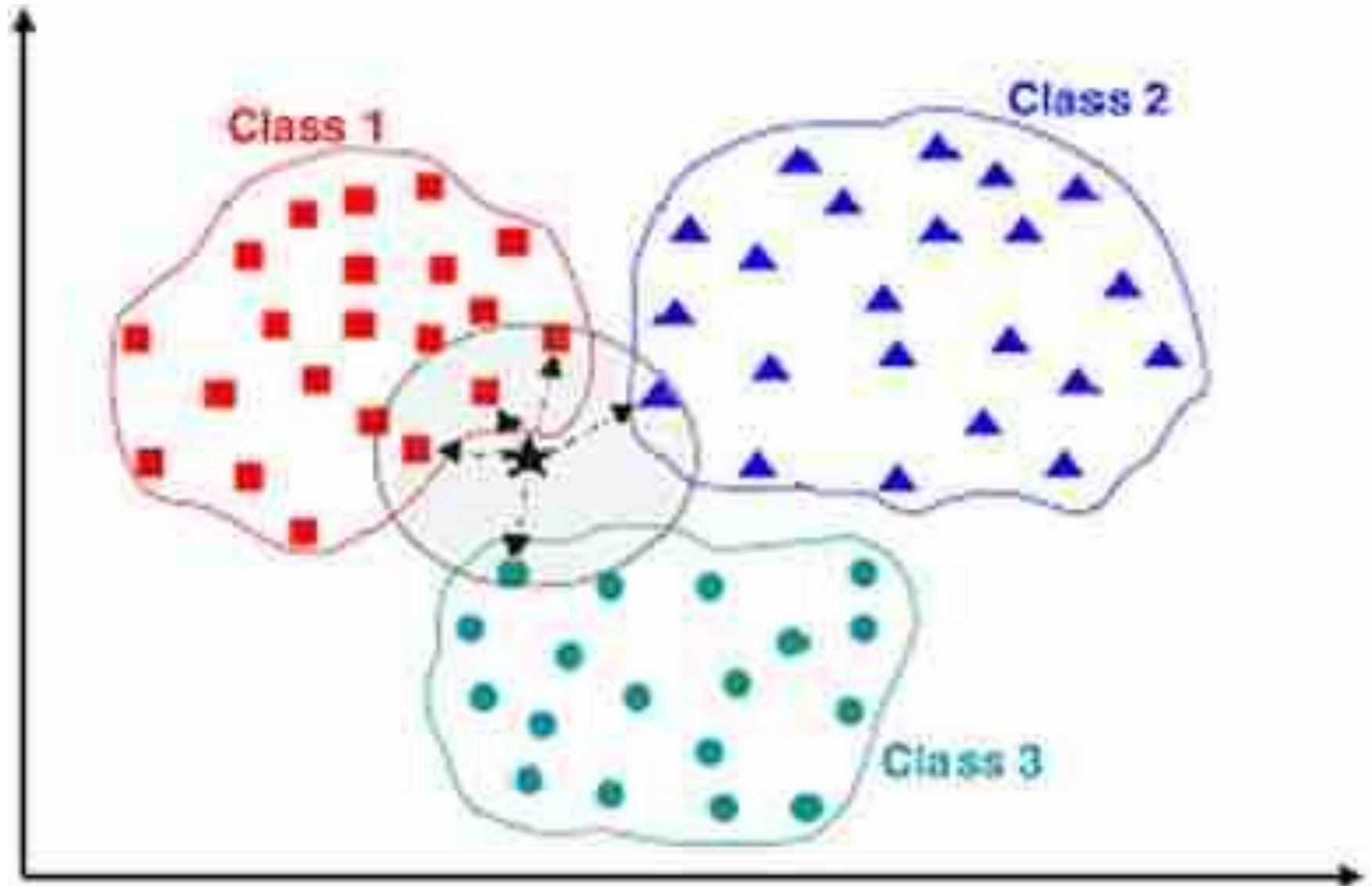
k Nearest Neighbor Classification

- k NN = k Nearest Neighbor
- To classify a document d :
- Define k -neighborhood as the k nearest neighbors of d
- Pick the majority class label in the k -neighborhood
- For larger k can roughly estimate $P(c|d)$ as $\#(c)/k$

Using K-NN



Using K-NN



Nearest-Neighbor Learning

- Learning: just store the labeled training examples D
- Testing instance x (*under 1NN*):
 - Compute similarity between x and all examples in D .
 - Assign x the category of the most similar example in D .
- Does not compute anything beyond storing the examples
- Also called:
 - Case-based learning
 - Memory-based learning
 - Lazy learning
- Rationale of kNN: contiguity hypothesis

k Nearest Neighbor

- Using only the closest example (1NN) subject to errors due to:
 - A single atypical example.
 - Noise (i.e., an error) in the category label of a single training example.
- More robust: find the k examples and return the majority category of these k
- k is typically odd to avoid ties; 3 and 5 are most common

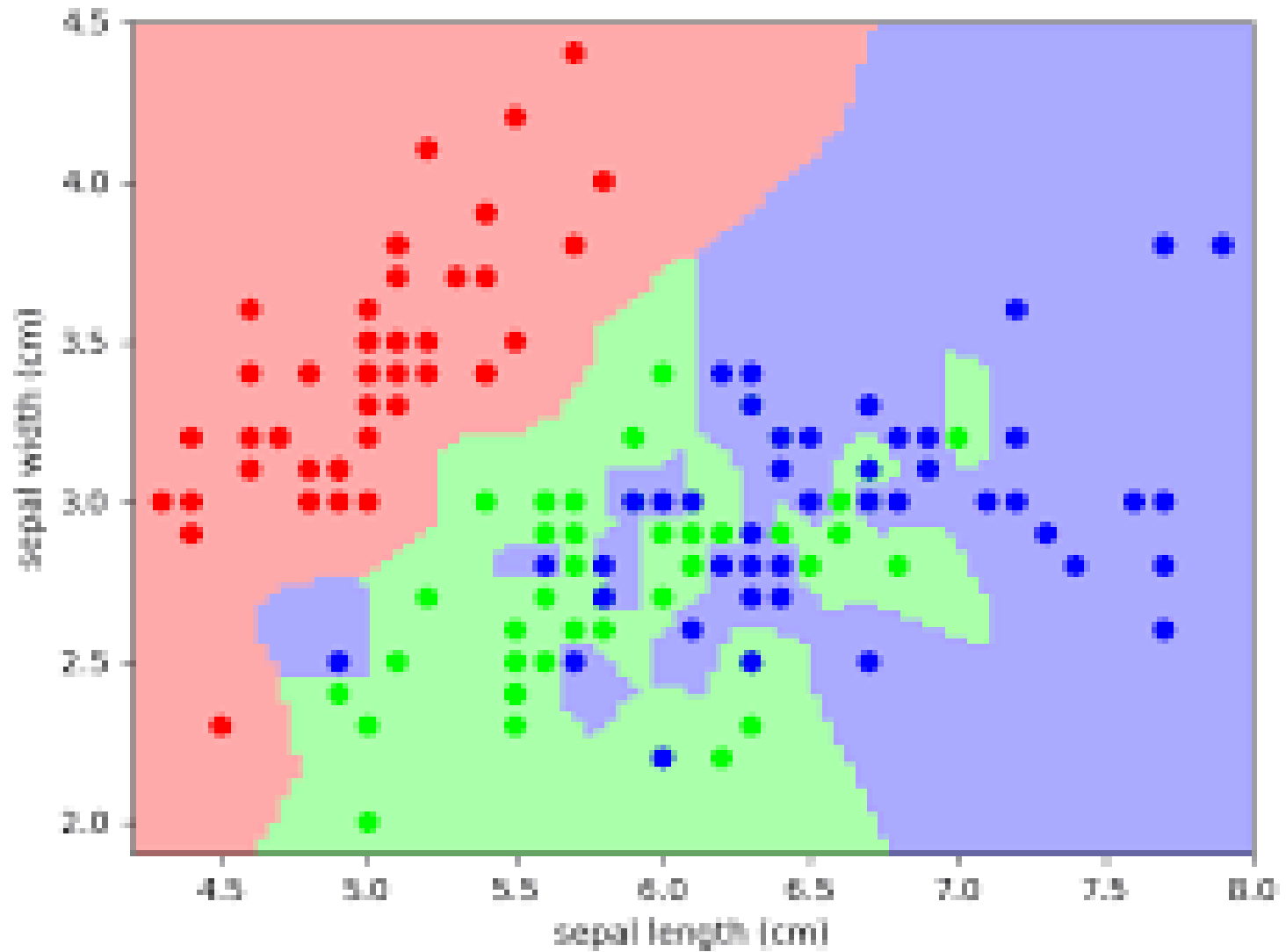
Nearest Neighbor with Inverted Index

- Naively finding nearest neighbors requires a linear search through $|D|$ documents in collection
- But determining k nearest neighbors is the same as determining the k best retrievals using the test document as a query to a database of training documents.
- Use standard vector space inverted index methods to find the k nearest neighbors.
- **Testing Time:** $O(B/V_t)$ where B is the average number of training documents in which a test-document word appears.
 - Typically $B \ll |D|$

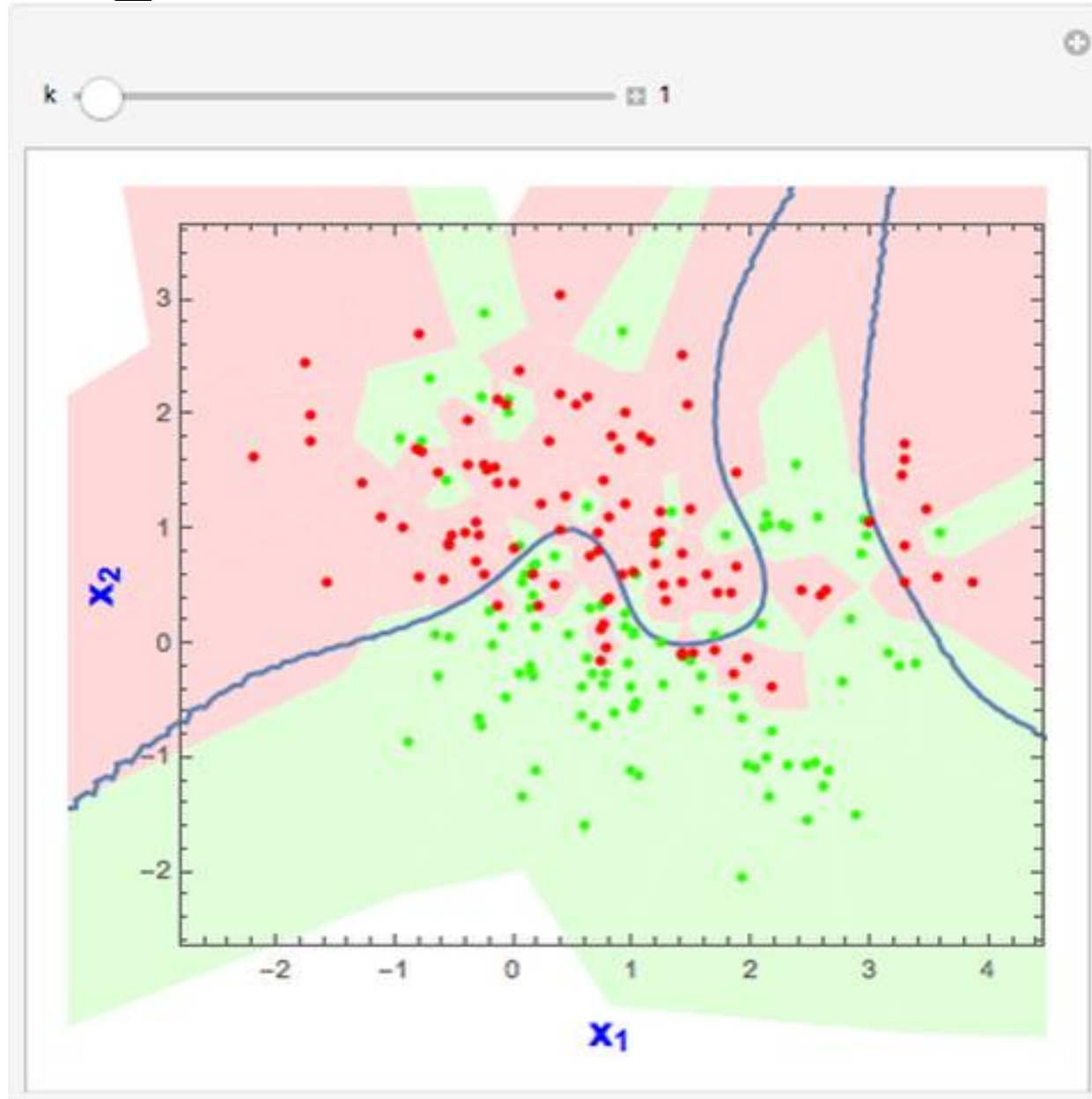
kNN: Discussion

- No feature selection necessary
- No training necessary
- Scales well with large number of classes
 - Don't need to train n classifiers for n classes
- Classes can influence each other
 - Small changes to one class can have ripple effect
- Done naively, very expensive at test time
- In most cases it's more accurate than NB or Rocchio

Using K-NN



Using K-NN



Basic Classification

Input

$x \in \mathcal{X}$

Output

$y \in \mathcal{Y}$

Spam
filtering

Spam vs. Not-Spam



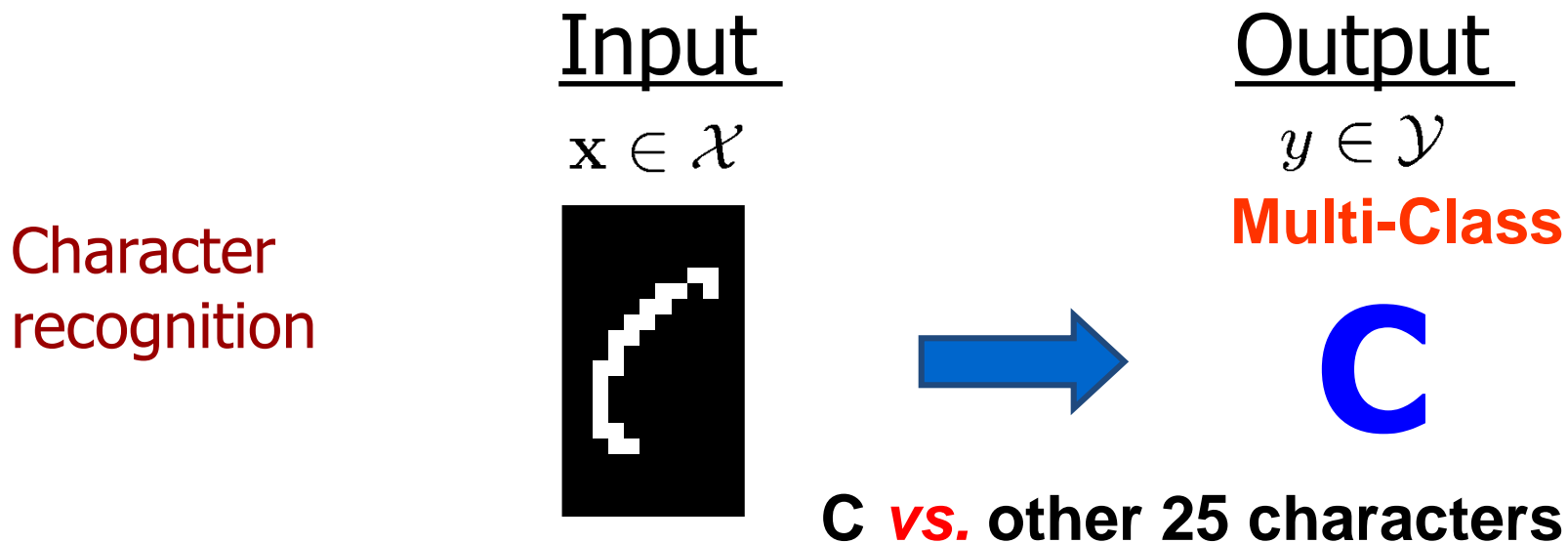
Binary



$$p(\text{spam} | x) = \frac{P(x | \text{spam})P(\text{spam})}{P(x | \text{spam})P(\text{spam}) + P(x | \text{not-spam})P(\text{not-spam})}$$

$$p(\text{not-spam} | x) = \frac{P(x | \text{not-spam})P(\text{not-spam})}{P(x | \text{spam})P(\text{spam}) + P(x | \text{not-spam})P(\text{not-spam})}$$

Basic Classification



$$P(C | x) = \frac{P(x | C)P(C)}{\sum_c P(x | c)P(c)}$$

$$c \in \Omega$$

Structured Classification

Input

$x \in \mathcal{X}$

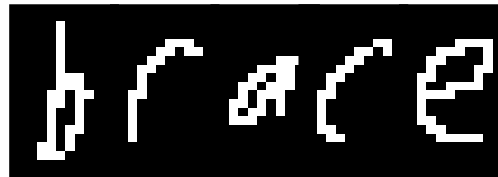
Output

$y \in \mathcal{Y}$

Handwriting
recognition

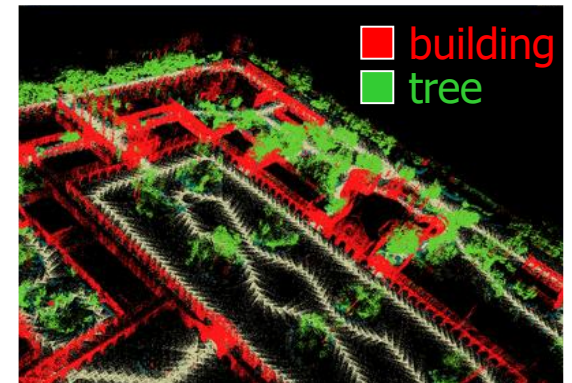
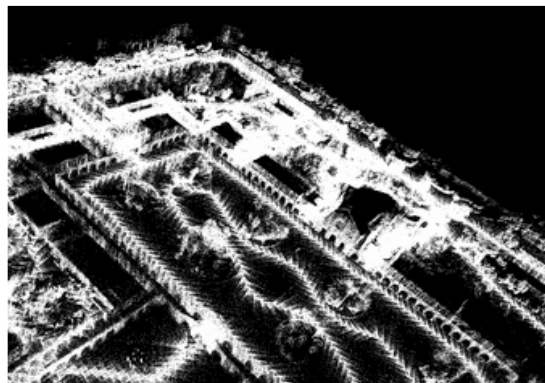
Graph Model

Structured output



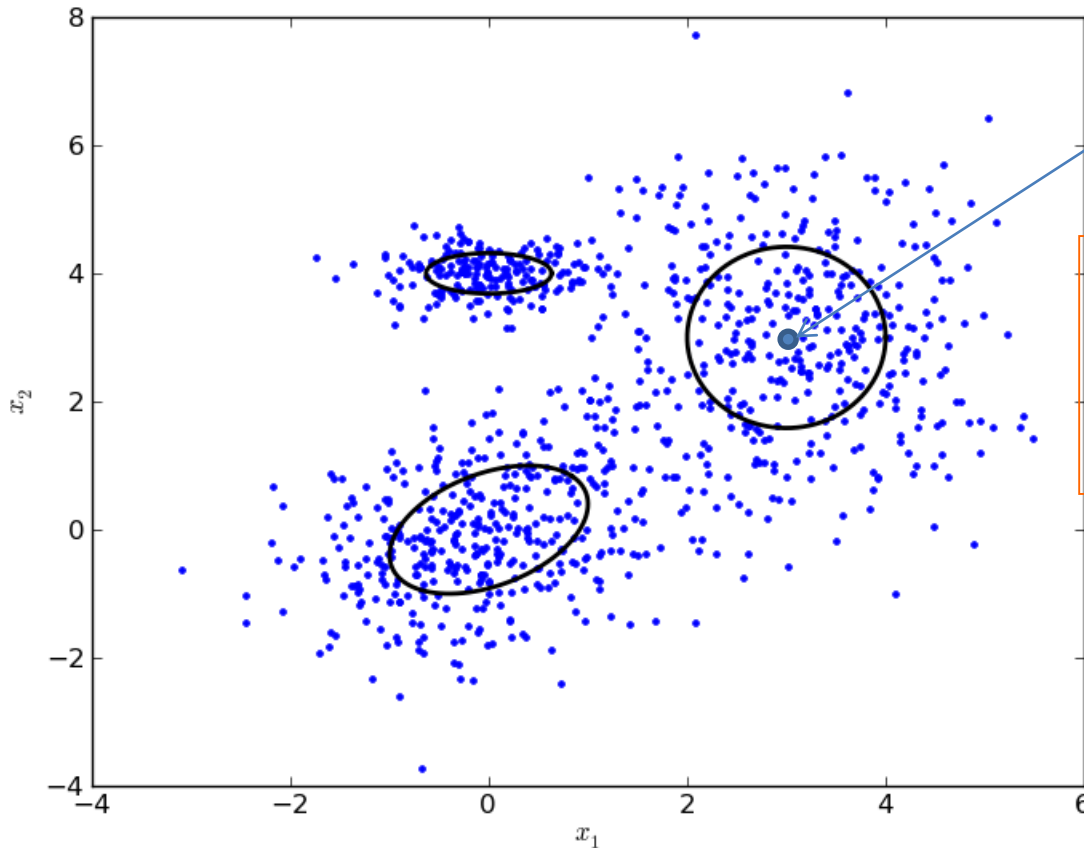
brace

3D object
recognition



Overview of Bayesian Decision

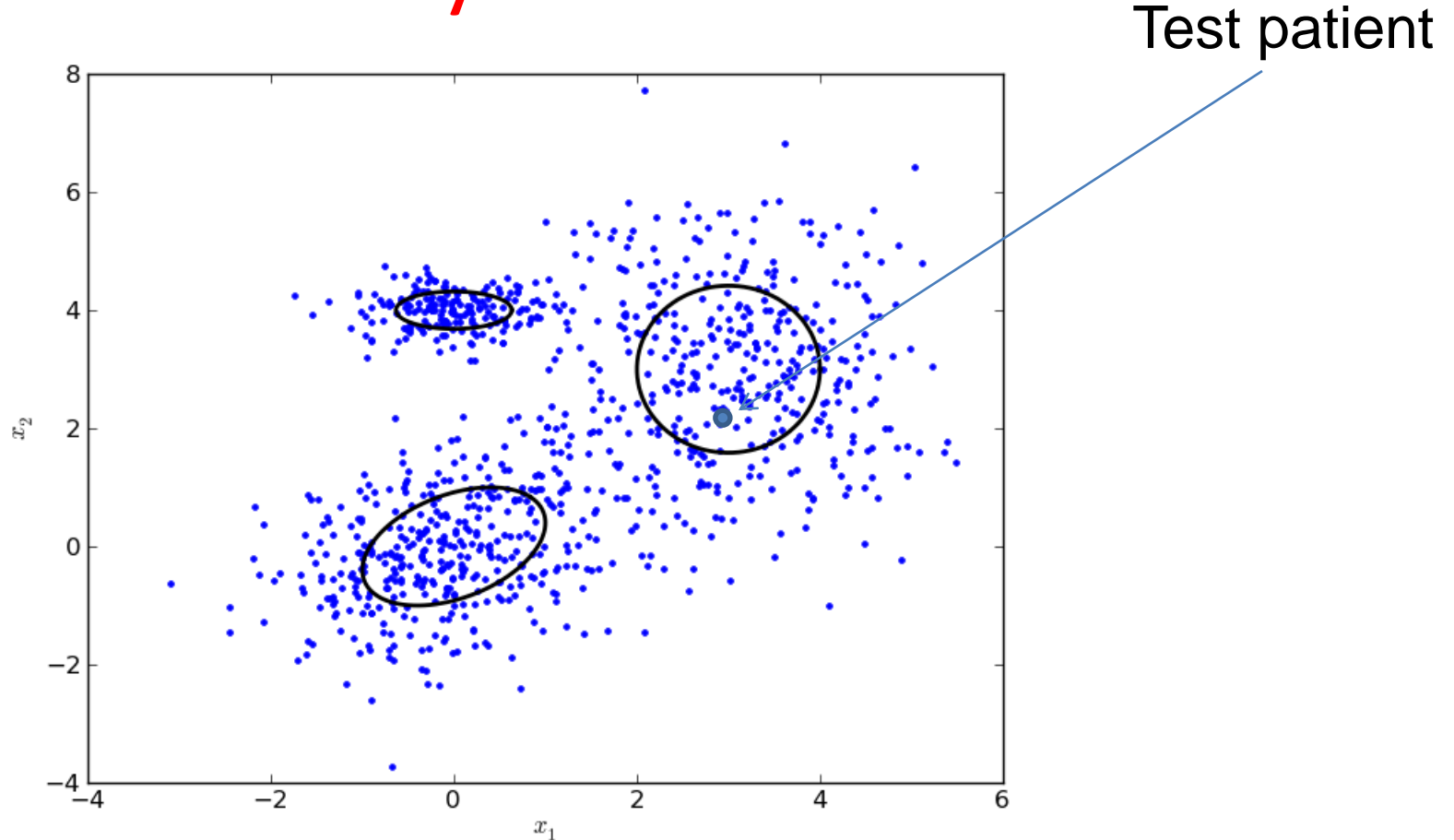
Test patient



$$P(C | x) = \frac{P(x | C)P(C)}{\sum_c P(x | c)P(c)}$$

- (a) Group assignment for test patient;
- (b) Prior knowledge about the assigned group
- (c) Properties of the assigned group (sick or healthy)

Overview of Bayesian Decision



Observations: Bayesian decision process is a data modeling process, e.g., **estimate the data distribution**

K-means clustering: any relationship & difference?

Bayes' Rule:

normalization

$$p(h | d) = \frac{P(d | h)P(h)}{P(d)}$$

$$= \frac{P(d | h)P(h)}{\sum_h P(d | h)P(h)}$$

Understanding Bayes' rule

d = data

h = hypothesis (model)

- rearranging

$$p(h | d)P(d) = P(d | h)P(h)$$

$$P(d, h) = P(d, h)$$

the same joint probability
on both sides

Who is who in Bayes' rule

$P(h)$: prior belief (probability of hypothesis h before seeing any data)

$P(d | h)$: likelihood (probability of the data if the hypothesis h is true)

$P(d) = \sum_h P(d | h)P(h)$: data evidence (marginal probability of the data)

$P(h | d)$: posterior (probability of hypothesis h after having seen the data d)

Bayes' Rule:

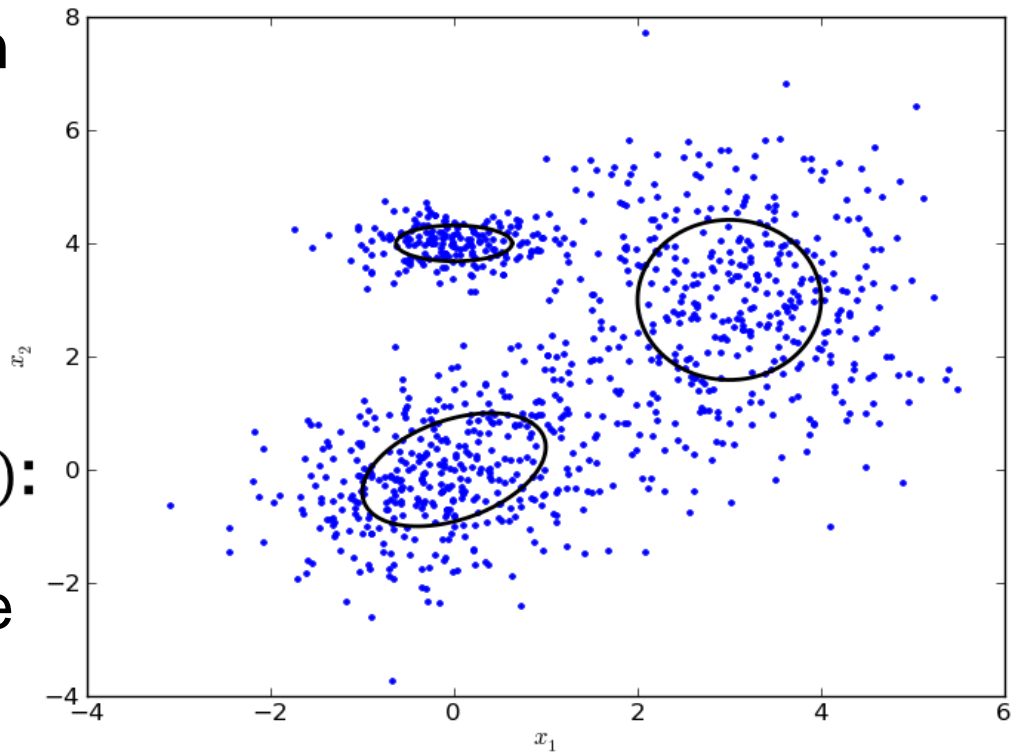
$$p(h | d) = \frac{P(d | h)P(h)}{P(d)} = \frac{P(d | h)P(h)}{\sum_h P(d | h)P(h)}$$

P(d|h): data distribution
of group h

P(h): importance
of group h

$$\mathbf{P(d)} = \sum_{h=1}^T \mathbf{P(d|h)P(h)}:$$

data distribution of whole
data set



Bayes' Rule:

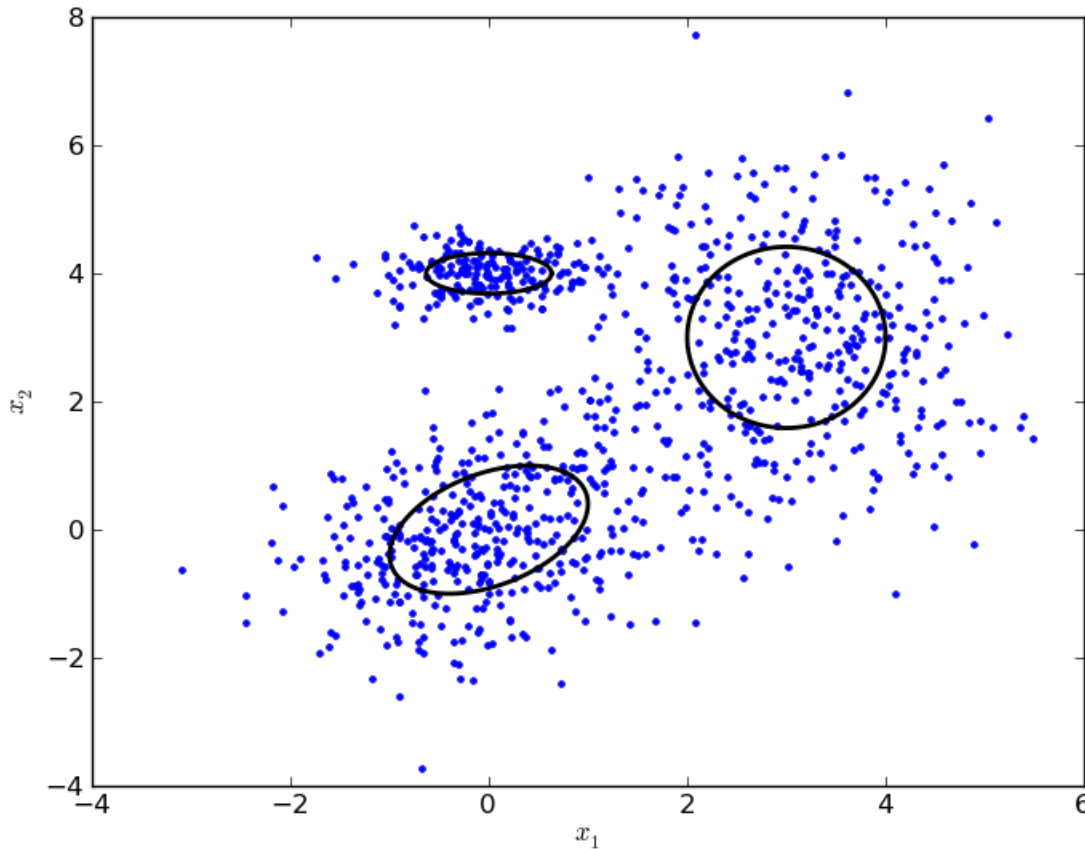
$$p(h | d) = \frac{P(d | h)P(h)}{P(d)} = \frac{P(d | h)P(h)}{\sum_h P(d | h)P(h)}$$

Works to support Bayesian decision:

- Estimating the data distribution for whole data set
- Estimating the data distribution for each specific group
- Prior knowledge about each specific group

Gaussian Mixture Model (GMM)

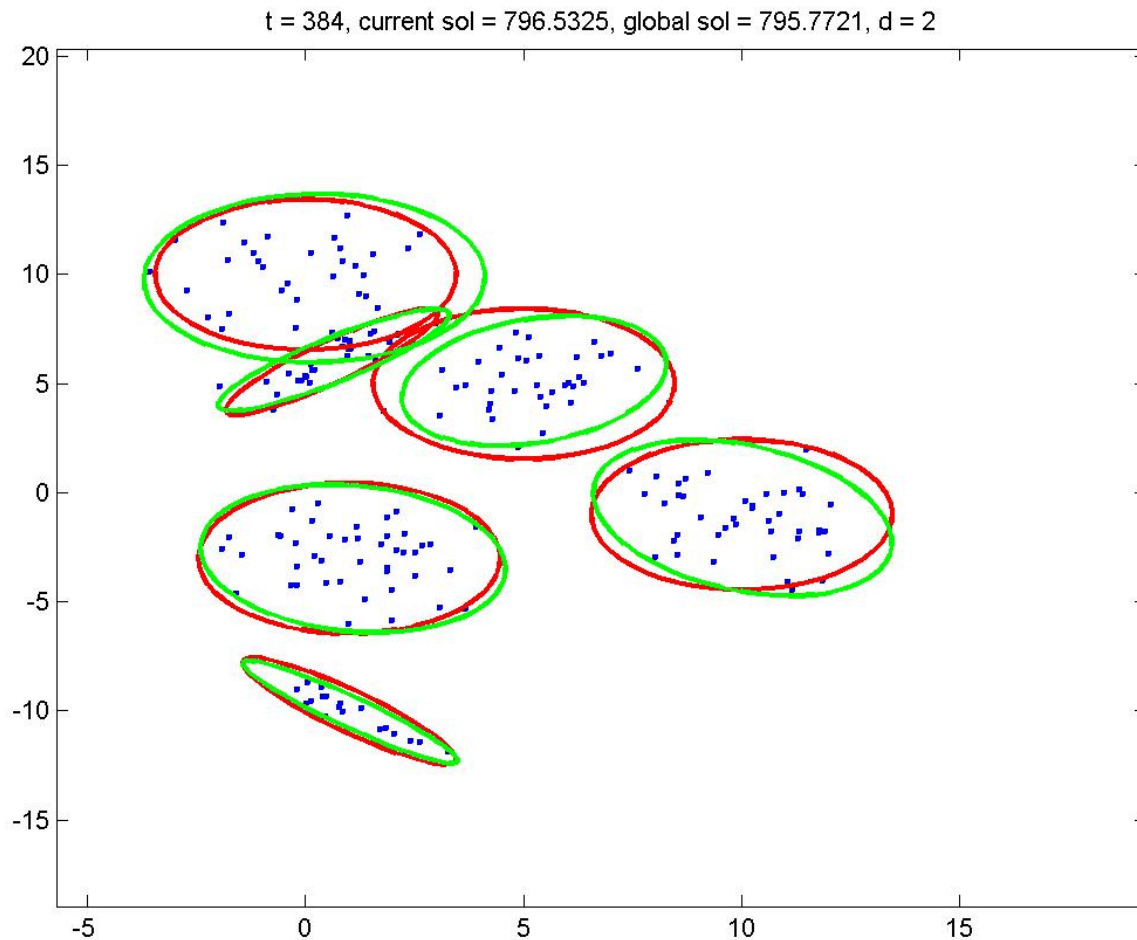
$$f(\mathbf{x}; \theta) = \sum_{k=1}^K p_k g(\mathbf{x}; \mathbf{m}_k, \sigma_k)$$



How to estimate the distribution for a given dataset?

Gaussian Mixture Model (GMM)

$$f(\mathbf{x}; \theta) = \sum_{k=1}^K p_k g(\mathbf{x}; \mathbf{m}_k, \sigma_k)$$



Gaussian Mixture Model (GMM)

$$f(\mathbf{x}; \theta) = \sum_{k=1}^K p_k g(\mathbf{x}; \mathbf{m}_k, \sigma_k)$$

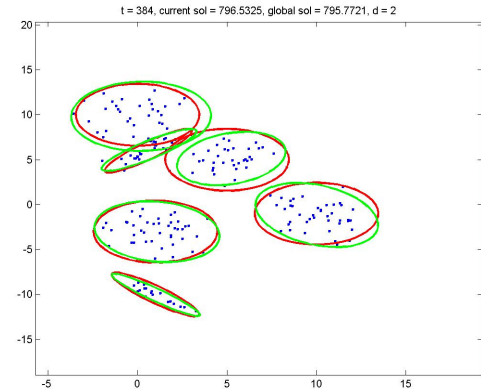
What does GMM mean?

$$100 = 5 \cdot 10 + 2 \cdot 20 + 2 \cdot 5$$

$$100 = 100 \cdot 1$$

$$100 = 10 \cdot 10$$

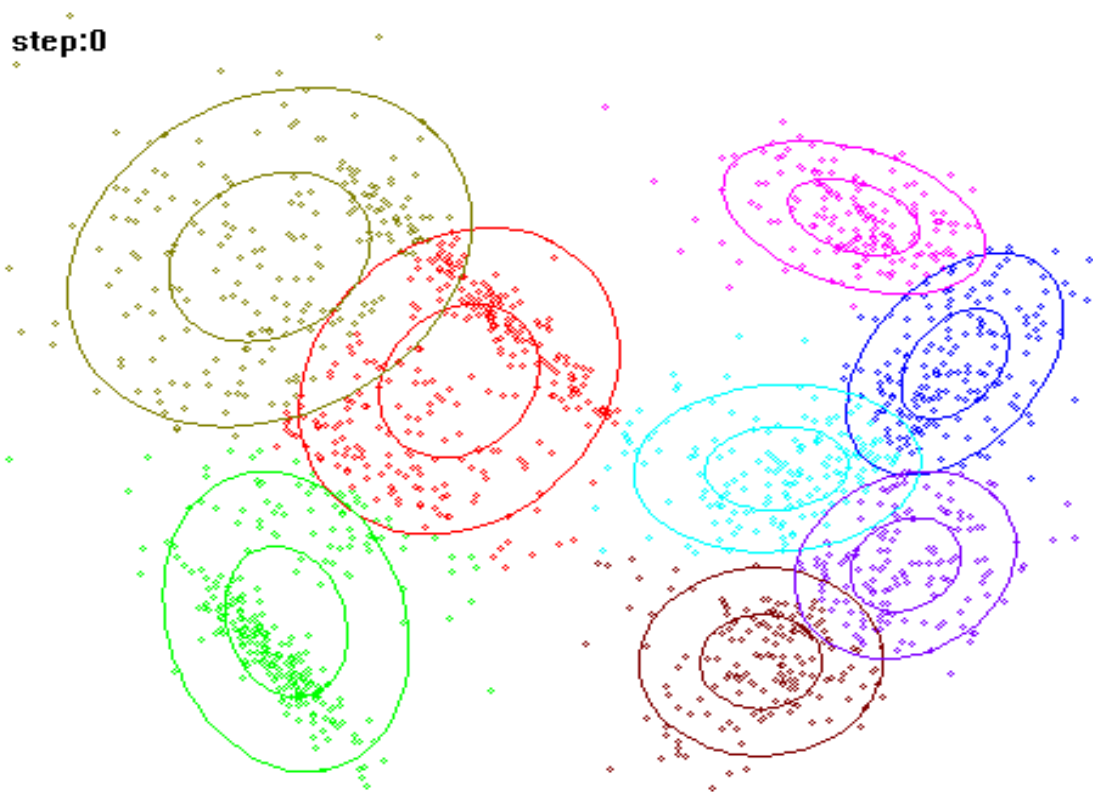
.....



GMM may prefer larger K with “smaller” Gaussians

Gaussian Mixture Model (GMM)

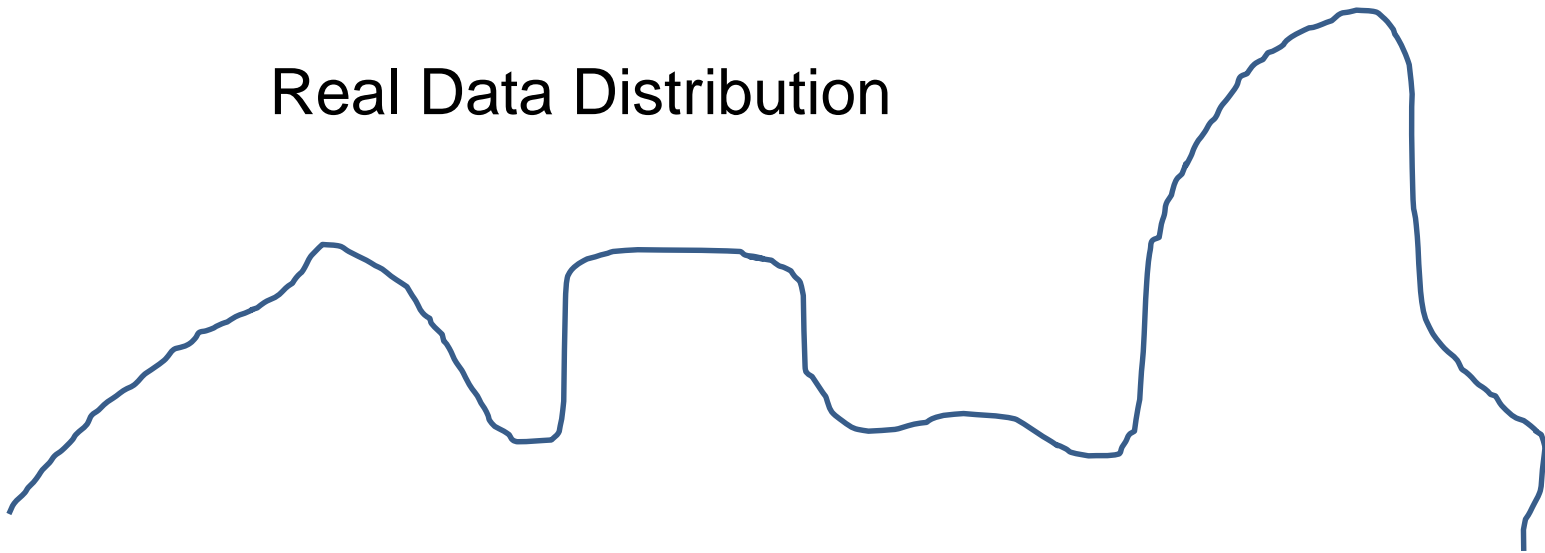
$$f(\mathbf{x}; \theta) = \sum_{k=1}^K p_k g(\mathbf{x}; \mathbf{m}_k, \sigma_k)$$



Gaussian Mixture Model (GMM)

$$f(\mathbf{x}; \theta) = \sum_{k=1}^K p_k g(\mathbf{x}; \mathbf{m}_k, \sigma_k)$$

Real Data Distribution

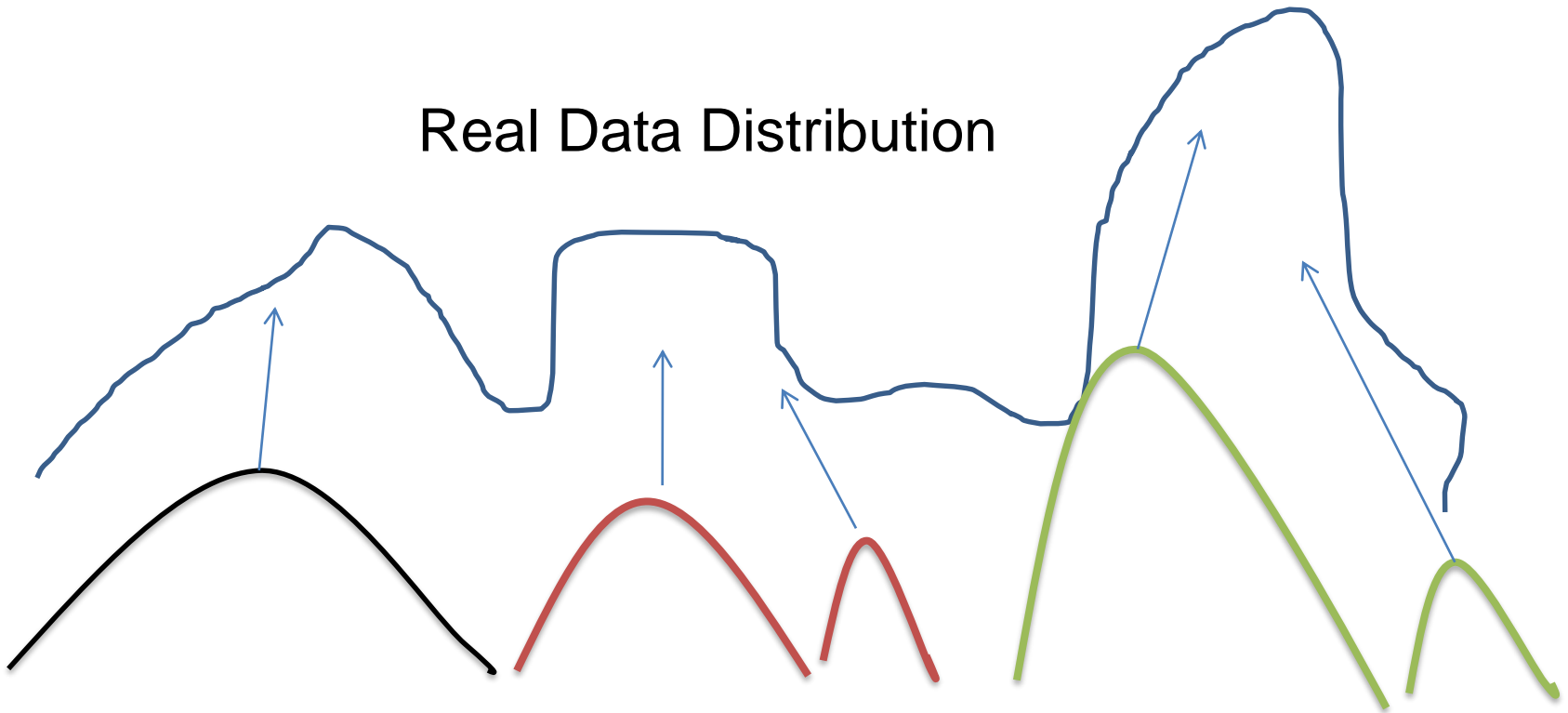


Any complex function (distribution) can be approximated by using a limited number of other functions (distributions) such as Gaussian functions.

Gaussian Mixture Model (GMM)

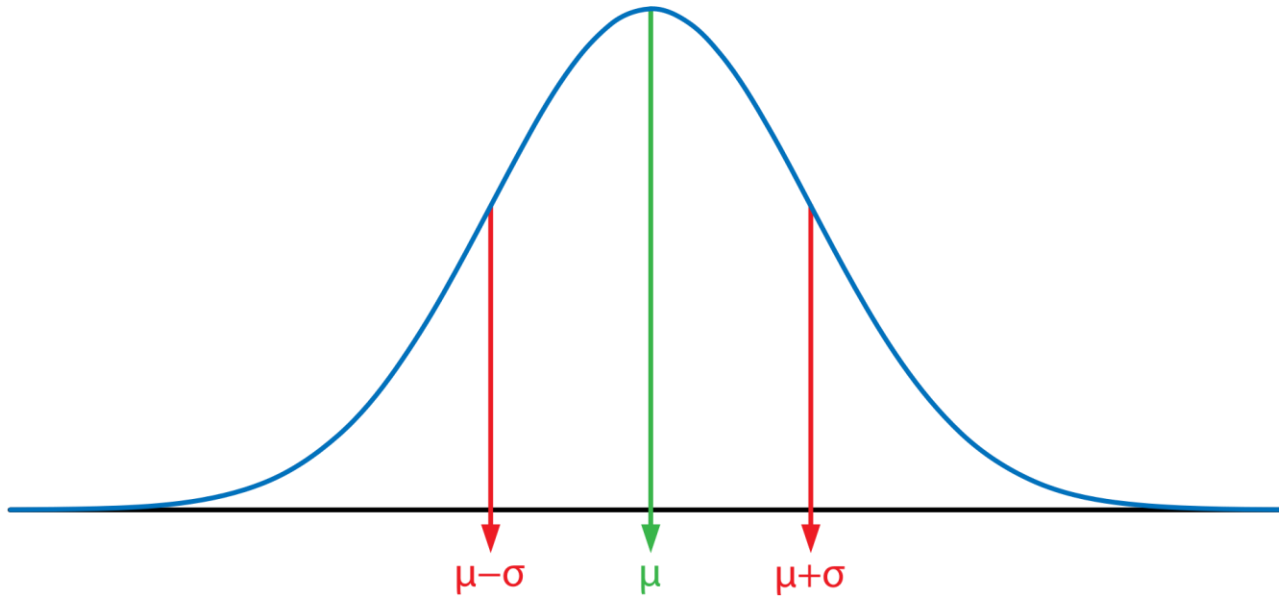
$$f(\mathbf{x}; \theta) = \sum_{k=1}^K p_k g(\mathbf{x}; \mathbf{m}_k, \sigma_k)$$

Real Data Distribution



Approximated Gaussian functions

Gaussian Function



$$f(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x - \mu)^2}{2\sigma^2}\right]$$

Why we select Gaussian function not others?

When one Gaussian Function is used to approximate data distribution

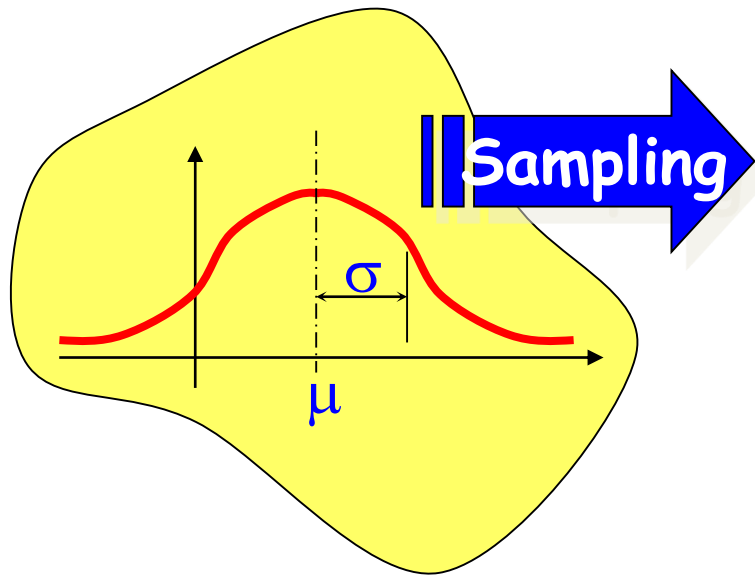


a **likelihood function** (often simply the **likelihood**) is a function of the parameters of a statistical model



Given a dataset, how to use likelihood function to determine Gaussian parameters?

Matching between Gaussian Function and Samples



$$X \sim N(\mu, \sigma^2)$$

$$\mathbf{X} = (x_1, x_2, \dots, x_n)^T$$

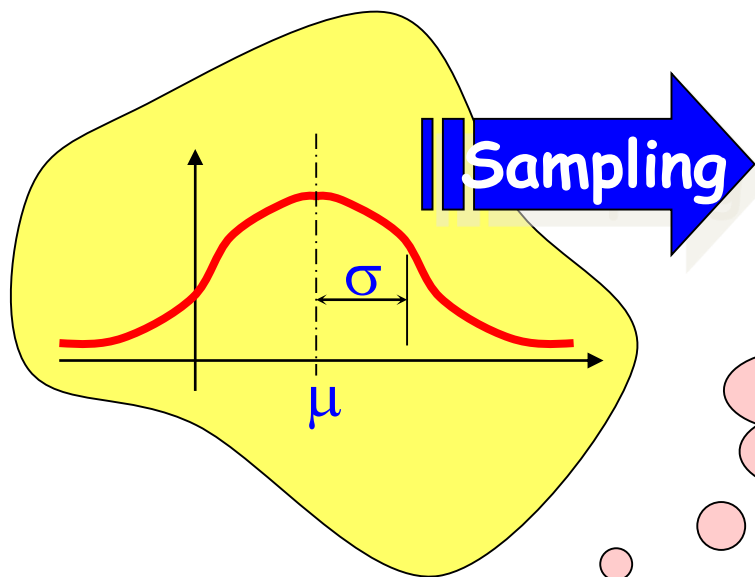
$$\hat{\mu} = ?$$

$$\hat{\sigma}^2 = ?$$

$$f(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left[-\frac{(x - \mu)^2}{2\sigma^2}\right]$$

Maximum Likelihood

$$f(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left[-\frac{(x - \mu)}{2\sigma^2}\right]$$



$$\mathbf{X} = (x_1, x_2, \dots, x_n)^T$$

We want to maximize it.

$$\mathcal{L}(\mu, \sigma^2 | \mathbf{X}) = f(\mathbf{x} | \mu, \sigma^2) = f(x_1 | \mu, \sigma^2) \cdots f(x_n | \mu, \sigma^2)$$

Given \mathbf{x} , it is a function of μ and σ^2

$$= \left(\frac{1}{2\pi\sigma^2}\right)^{n/2} \exp\left[-\sum_{i=1}^n \frac{(x_i - \mu)^2}{2\sigma^2}\right]$$


Log-Likelihood Function

$$\mathcal{L}(\boldsymbol{\mu}, \sigma^2 \mid \mathbf{X}) = \left(\frac{1}{2\pi\sigma^2} \right)^{n/2} \exp \left[-\sum_{i=1}^n \frac{(x_i - \boldsymbol{\mu})^2}{2\sigma^2} \right]$$

$$\ell(\boldsymbol{\mu}, \sigma^2 \mid \mathbf{X}) = \log \mathcal{L}(\boldsymbol{\mu}, \sigma^2 \mid \mathbf{X})$$

$$= \frac{n}{2} \log \frac{1}{2\pi\sigma^2} - \sum_{i=1}^n \frac{(x_i - \boldsymbol{\mu})^2}{2\sigma^2}$$

$$= -\frac{n}{2} \log \sigma^2 - \frac{n}{2} \log 2\pi - \frac{1}{2\sigma^2} \sum_{i=1}^n x_i^2 + \frac{\boldsymbol{\mu}}{\sigma^2} \sum_{i=1}^n x_i - \frac{n\boldsymbol{\mu}^2}{2\sigma^2}$$



Maximize
this instead

By setting

$$\frac{\partial}{\partial \boldsymbol{\mu}} \ell(\boldsymbol{\mu}, \sigma^2 \mid \mathbf{X}) = 0 \quad \text{and} \quad \frac{\partial}{\partial \sigma^2} \ell(\boldsymbol{\mu}, \sigma^2 \mid \mathbf{X}) = 0$$

Max. the Log-Likelihood Function

$$\ell(\mu, \sigma^2 \mid \mathbf{X}) = -\frac{n}{2} \log \sigma^2 - \frac{n}{2} \log 2\pi - \frac{1}{2\sigma^2} \sum_{i=1}^n x_i^2 + \frac{\mu}{\sigma^2} \sum_{i=1}^n x_i - \frac{n\mu^2}{2\sigma^2}$$

$$\frac{\partial}{\partial \mu} \ell(\mu, \sigma^2 \mid \mathbf{X}) = \frac{1}{\sigma^2} \sum_{i=1}^n x_i - \frac{n\mu}{\sigma^2} = 0$$



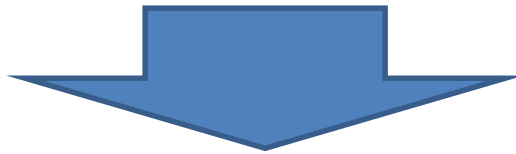
$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n x_i$$

Max. the Log-Likelihood Function

$$\ell(\boldsymbol{\mu}, \sigma^2 \mid \mathbf{X}) = -\frac{n}{2} \log \sigma^2 - \frac{n}{2} \log 2\pi - \frac{1}{2\sigma^2} \sum_{i=1}^n x_i^2 + \frac{\boldsymbol{\mu}}{\sigma^2} \sum_{i=1}^n x_i - \frac{n\boldsymbol{\mu}^2}{2\sigma^2}$$

$$\frac{\partial}{\partial \sigma^2} \ell(\boldsymbol{\mu}, \sigma^2 \mid \mathbf{X}) = -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4} \sum_{i=1}^n x_i^2 - \frac{\boldsymbol{\mu}}{\sigma^4} \sum_{i=1}^n x_i + \frac{n\boldsymbol{\mu}^2}{2\sigma^4} = 0$$

$$n\sigma^2 = \sum_{i=1}^n x_i^2 - 2\boldsymbol{\mu} \sum_{i=1}^n x_i + n\boldsymbol{\mu}^2 = \sum_{i=1}^n x_i^2 - \frac{2}{n} \left(\sum_{i=1}^n x_i \right)^2 + \frac{1}{n} \left(\sum_{i=1}^n x_i \right)^2$$



$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n x_i^2 - \hat{\boldsymbol{\mu}}^2 \quad \hat{\boldsymbol{\mu}} = \frac{1}{n} \sum_{i=1}^n x_i$$

When multiple Gaussian Functions are used to approximate data distribution



a **likelihood function** (often simply the **likelihood**) is a function of the parameters of a statistical model



Given a dataset, how to use likelihood function to determine parameters for multiple Gaussian functions?

Gaussian Mixture Model (GMM)

$$f(\mathbf{x}; \theta) = \sum_{k=1}^K p_k g(\mathbf{x}; \mathbf{m}_k, \sigma_k)$$

Parameters to be estimated:

- Number of Gaussian functions K
- Gaussian parameters: $\{m_k, \sigma_k \mid k=1, \dots, K\}$, $2K$
- Weights (importance) of each Gaussian, $\{p_k \mid k=1, \dots, K\}$, K

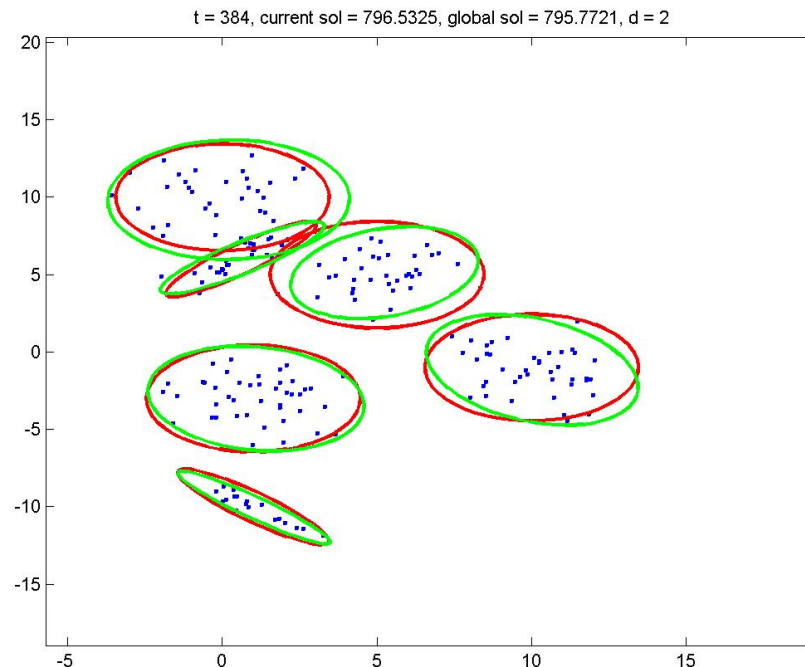
Training Set:

$$\{(x_i, y_i) \mid i=1, \dots, N\}$$

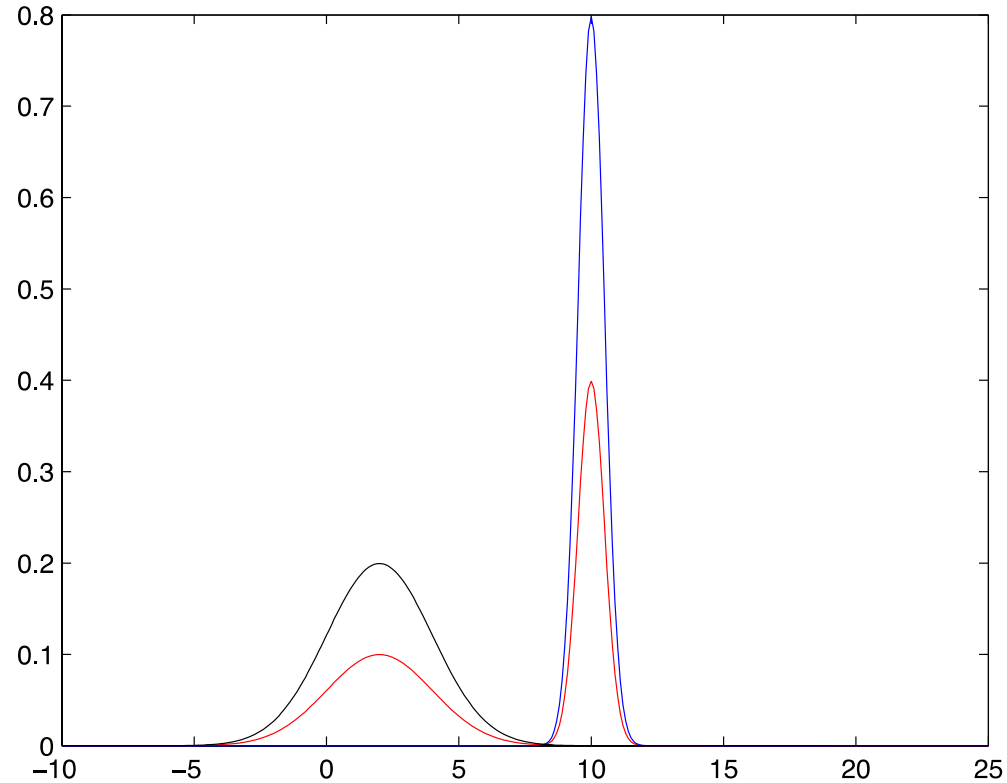
We assume K is available (or pre-defined)! Algorithms may always prefer larger K !

Gaussian Mixture Models

- Rather than identifying clusters by “nearest” centroids
- Fit a Set of k Gaussians to the data
- Maximum Likelihood over a mixture model



GMM example



$$f_0(x) = N(x; 2, 2)$$

$$f_1(x) = N(x; 10, .5)$$

$$\pi = [.5 \quad .5]^T$$

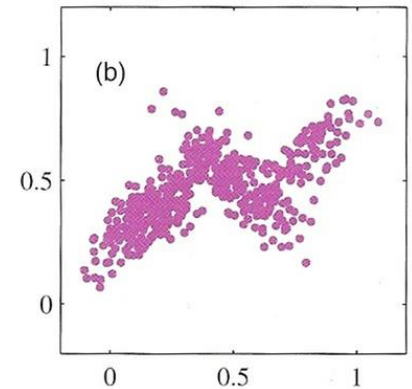
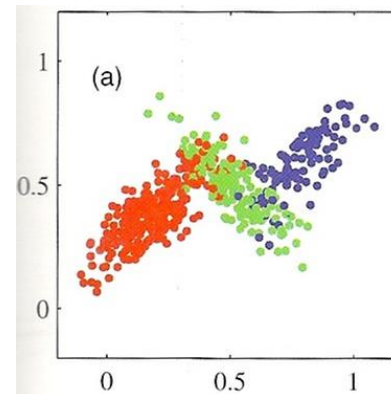
Mixture Models

- Formally a Mixture Model is the weighted sum of a number of pdfs where the weights are determined by a distribution,

$$p(x) = \pi_0 f_0(x) + \pi_1 f_1(x) + \pi_2 f_2(x) + \dots + \pi_k f_k(x)$$

where $\sum_{i=0}^k \pi_i = 1$

$$p(x) = \sum_{i=0}^k \pi_i f_i(x)$$



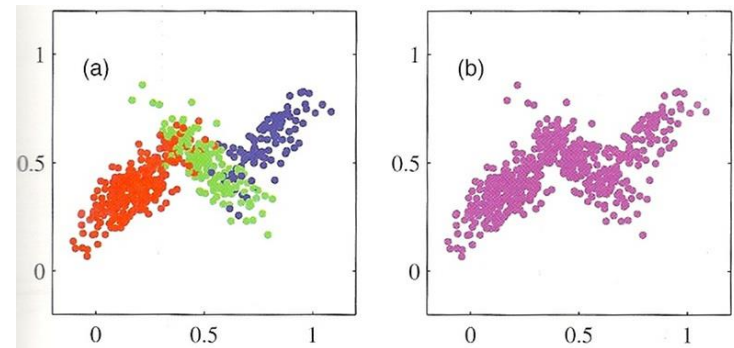
Gaussian Mixture Models

- GMM: the weighted sum of a number of Gaussians where the weights are determined by a distribution,

$$p(x) = \pi_0 N(x|\mu_0, \Sigma_0) + \pi_1 N(x|\mu_1, \Sigma_1) + \dots + \pi_k N(x|\mu_k, \Sigma_k)$$

$$\text{where } \sum_{i=0}^k \pi_i = 1$$

$$p(x) = \sum_{i=0}^k \pi_i N(x|\mu_k, \Sigma_k)$$



Maximum Likelihood over a GMM

- As usual: Identify a **likelihood function**

$$p(x|\pi, \mu, \Sigma) = \sum_{n=1}^N \left\{ \sum_{k=1}^K \pi_k N(x_n | \mu_k, \Sigma_k) \right\}$$

- Log-likelihood

$$\ln p(x|\pi, \mu, \Sigma) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k N(x_n | \mu_k, \Sigma_k) \right\}$$

Maximum Likelihood of a GMM

- Optimization of means.

$$\ln p(x|\pi, \mu, \Sigma) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k N(x_n | \mu_k, \Sigma_k) \right\}$$

$$\begin{aligned} \frac{\partial \ln p(x|\pi, \mu, \Sigma)}{\partial \mu_k} &= \sum_{n=1}^N \frac{\pi_k N(x_n | \mu_k, \Sigma_k)}{\sum_j \pi_j N(x_n | \mu_j, \Sigma_j)} \Sigma_k^{-1} (x_k - \mu_k) = 0 \\ &= \sum_{n=1}^N \tau(z_{nk}) \Sigma_k^{-1} (x_k - \mu_k) = 0 \end{aligned}$$



$$\mu_k = \frac{\sum_{n=1}^N \tau(z_{nk}) x_n}{\sum_{n=1}^N \tau(z_{nk})}$$

$k=1, \dots, K$

$$\tau(z_{nk}) = \frac{\pi_k N(x_n | \mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j N(x_n | \mu_j, \Sigma_j)}$$

Maximum Likelihood of a GMM

- Optimization of covariance

$$\ln p(x|\pi, \mu, \Sigma) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k N(x_n | \mu_k, \Sigma_k) \right\}$$

$$\frac{\partial}{\partial \sigma^2} \ln p(x|\pi, \mu, \Sigma) = \frac{\partial}{\partial \sigma^2} \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k N(x_n | \mu_k, \Sigma_k) \right\}$$



$$\Sigma_k = \frac{1}{\sum_{n=1}^N \tau(z_{nk})} \sum_{n=1}^N \tau(z_{nk}) (x_k - \mu_k)(x_k - \mu_k)^T$$

$k=1, \dots, K$

Maximum Likelihood of a GMM

- Optimization of mixing term

$$\ln p(x|\pi, \mu, \Sigma) + \lambda \left(\sum_{k=1}^K \pi_k - 1 \right)$$

$$0 = \sum_{n=1}^N \frac{\pi_k N(x_n|\mu_k, \Sigma_k)}{\sum_j \pi_j N(x_n|\mu_j, \Sigma_j)} + \lambda$$

$$\pi_k = \frac{\sum_{n=1}^N \tau(z_n k)}{N}$$

EM for GMMs

- Initialize the parameters
 - Evaluate the log likelihood
- **Expectation-step:** Evaluate the responsibilities
- **Maximization-step:** Re-estimate Parameters
 - Evaluate the log likelihood
 - Check for convergence

EM for GMMs

- **E-step**: Evaluate the Responsibilities

$$\tau(z_{nk}) = \frac{\pi_k N(x_n | \mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j N(x_n | \mu_j, \Sigma_j)}$$

EM for GMMs

- **M-Step:** Re-estimate Parameters

$$\mu_k^{new} = \frac{\sum_{n=1}^N \tau(z_{nk}) x_n}{N_k}$$

$$\Sigma_k^{new} = \frac{1}{N_k} \sum_{n=1}^N \tau(z_{nk}) (x_k - \mu_k^{new})(x_k - \mu_k^{new})^T$$

$$\pi_k^{new} = \frac{N_k}{N}$$

EM for GMMs

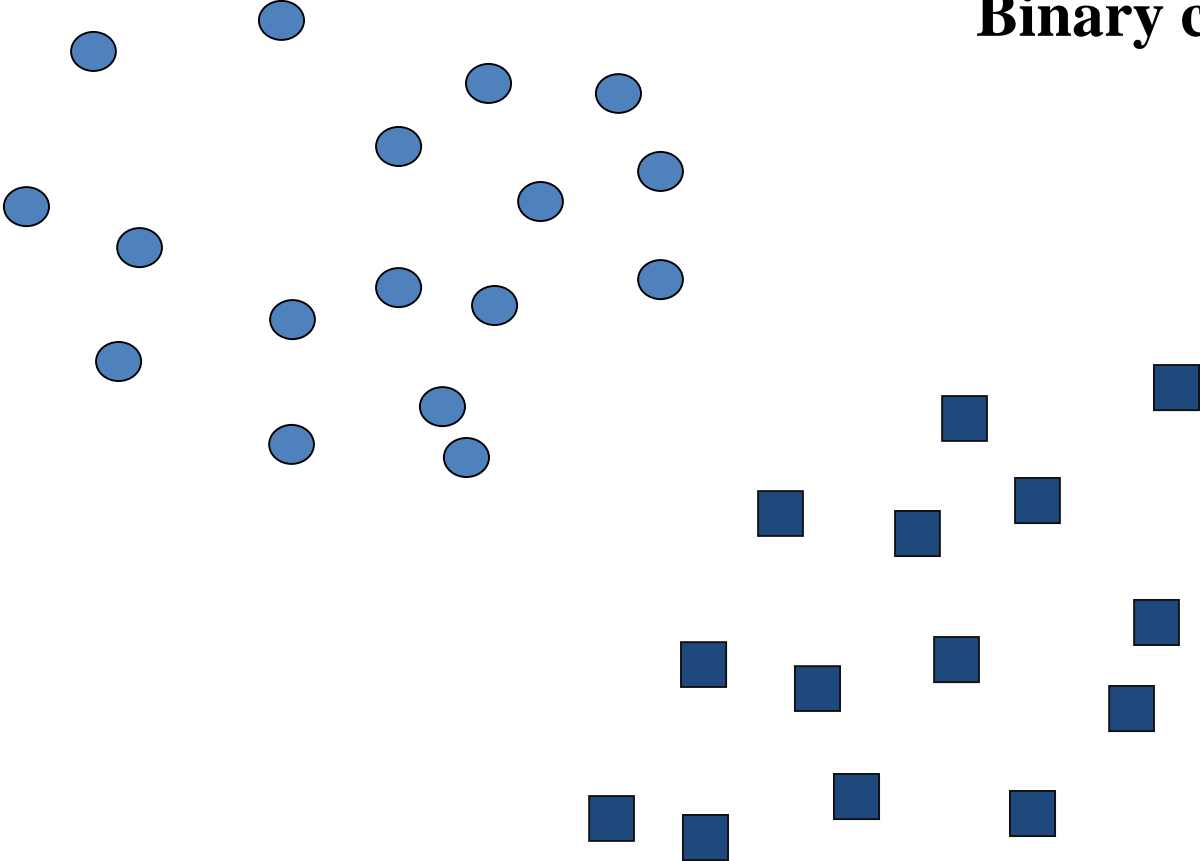
- Evaluate the log likelihood

$$\ln p(\mathbf{X}|\pi, \mu, \Sigma) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k) \right\}$$

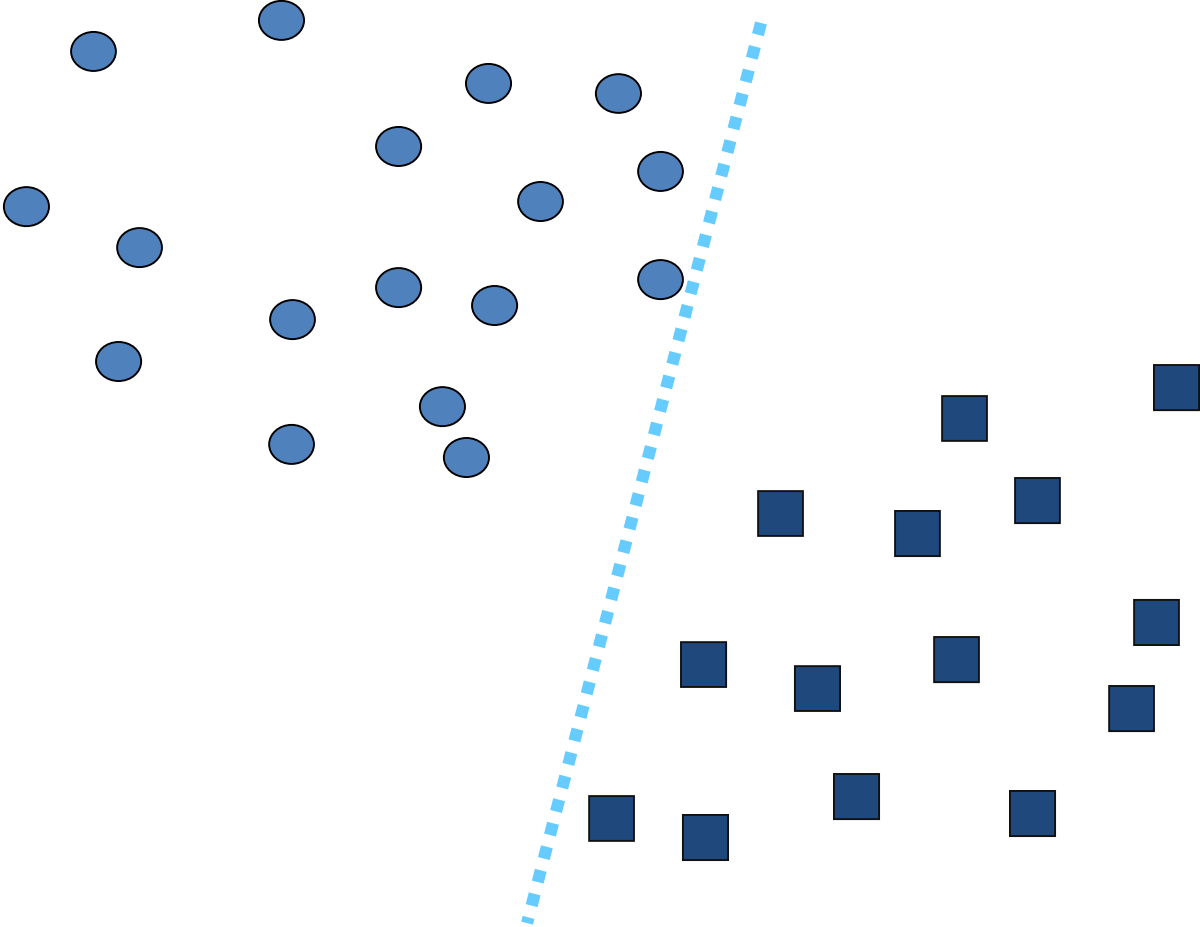
and check for convergence of either the parameters or the log likelihood. If the convergence criterion is not satisfied, **return to E-Step.**

Linear SVM

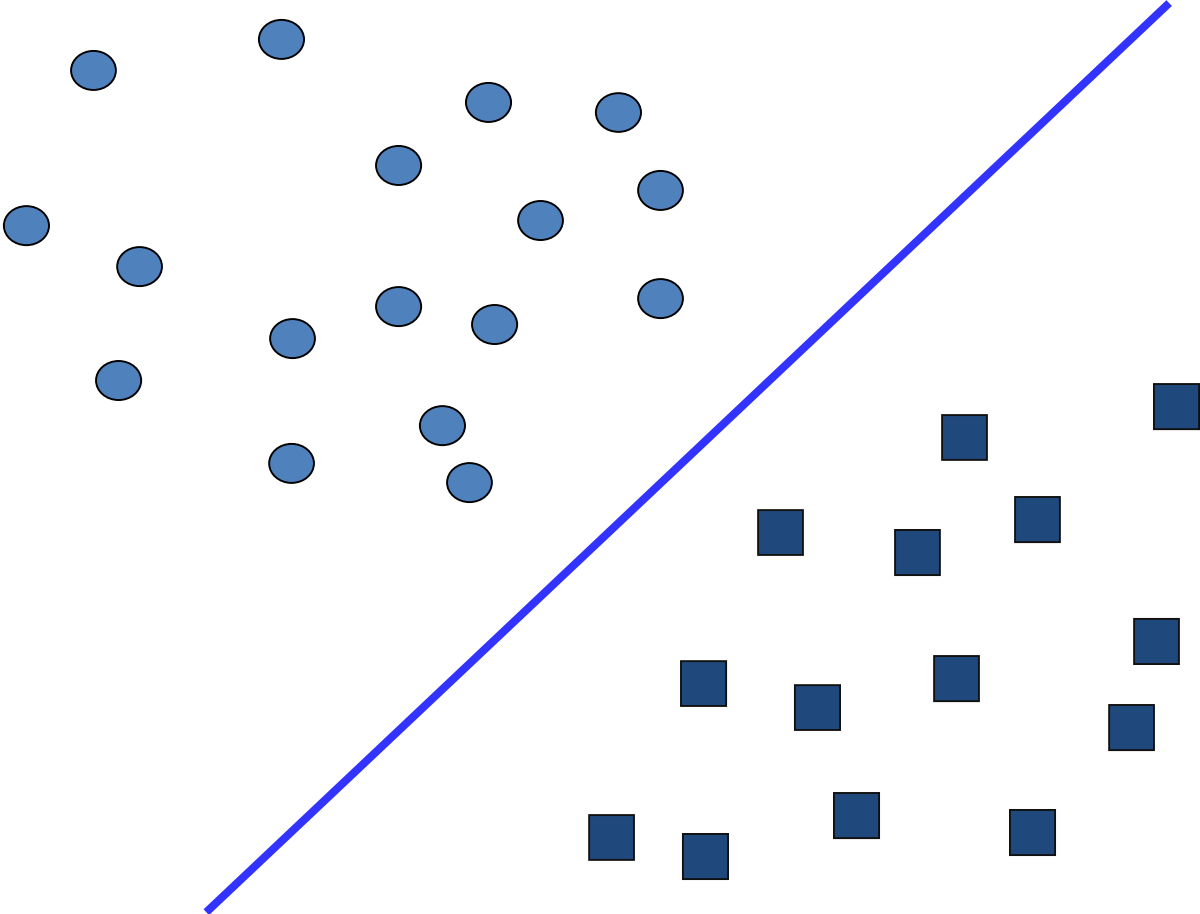
Binary case



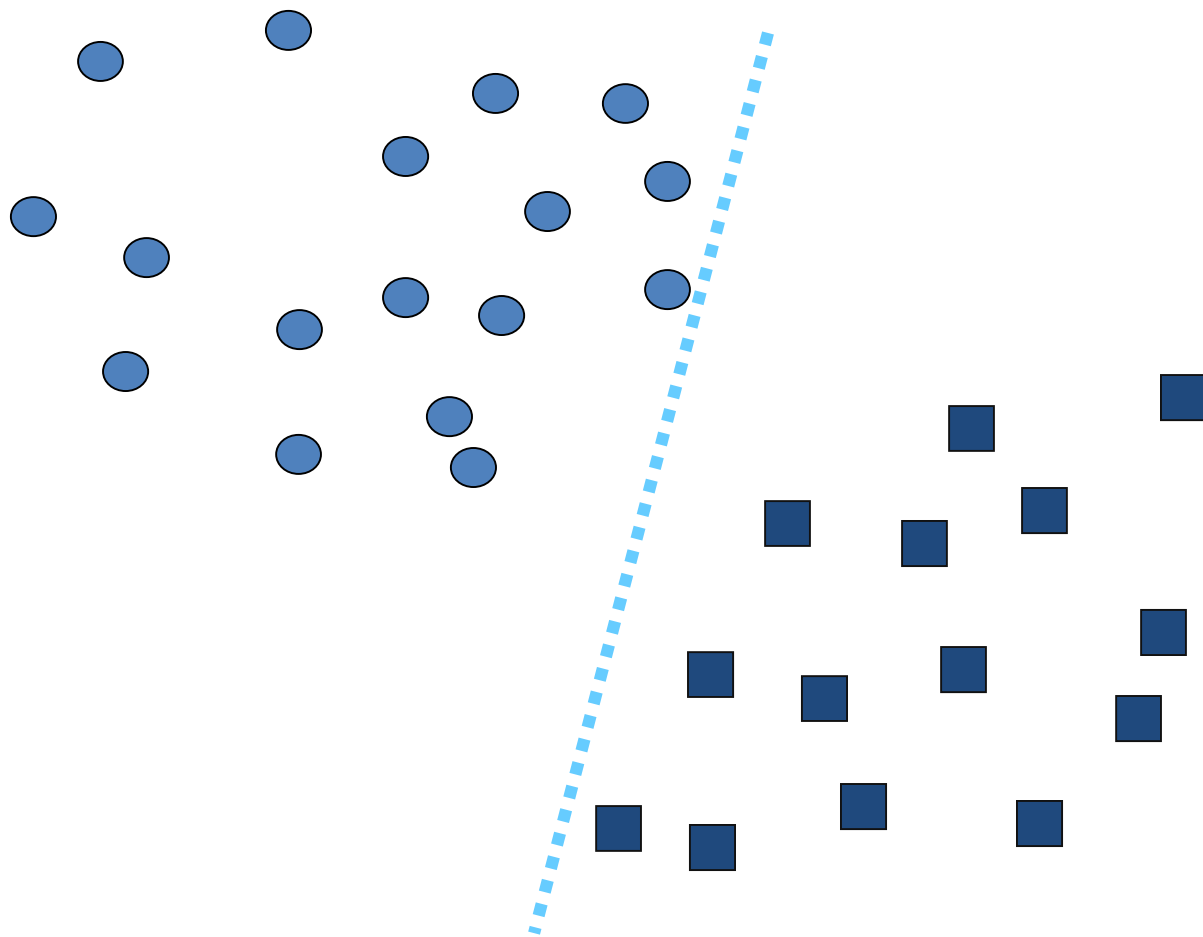
Linear SVM



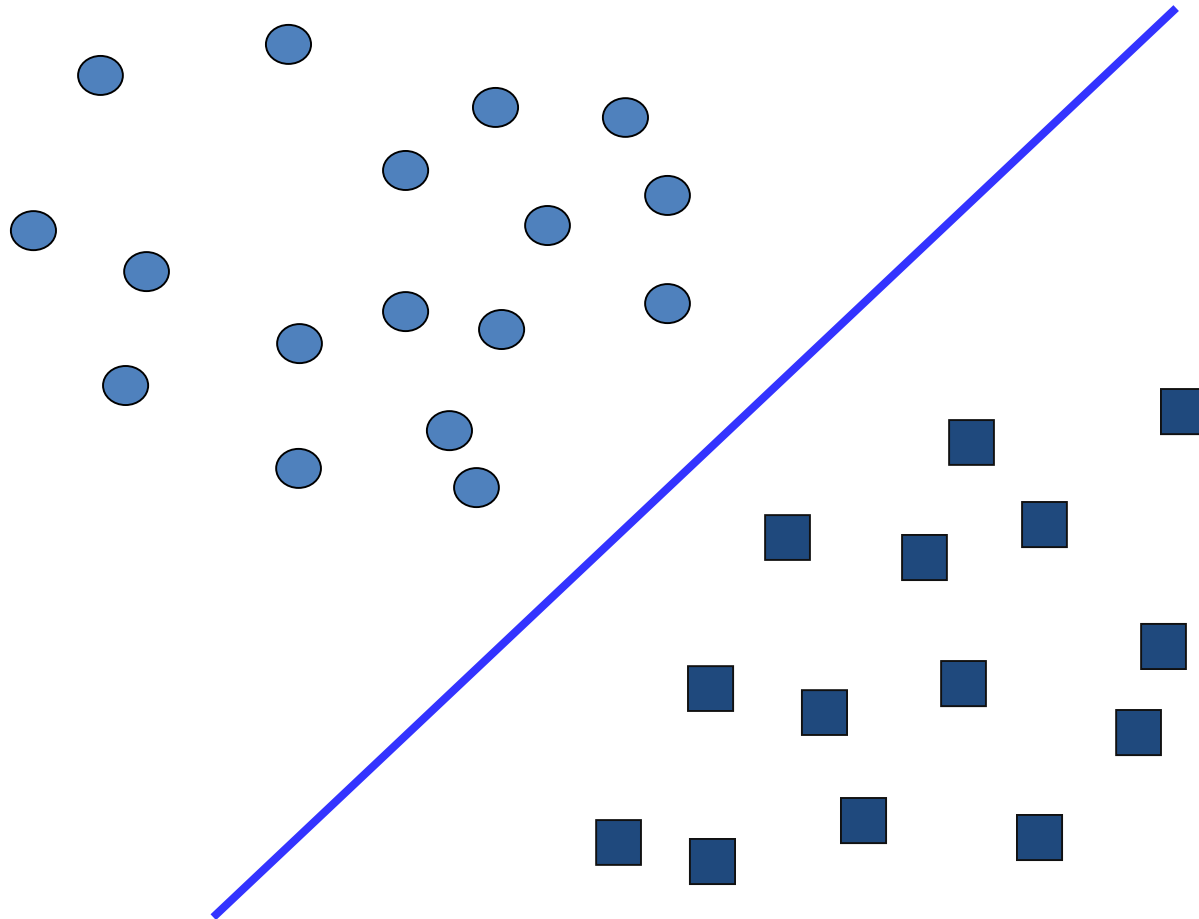
Linear SVM



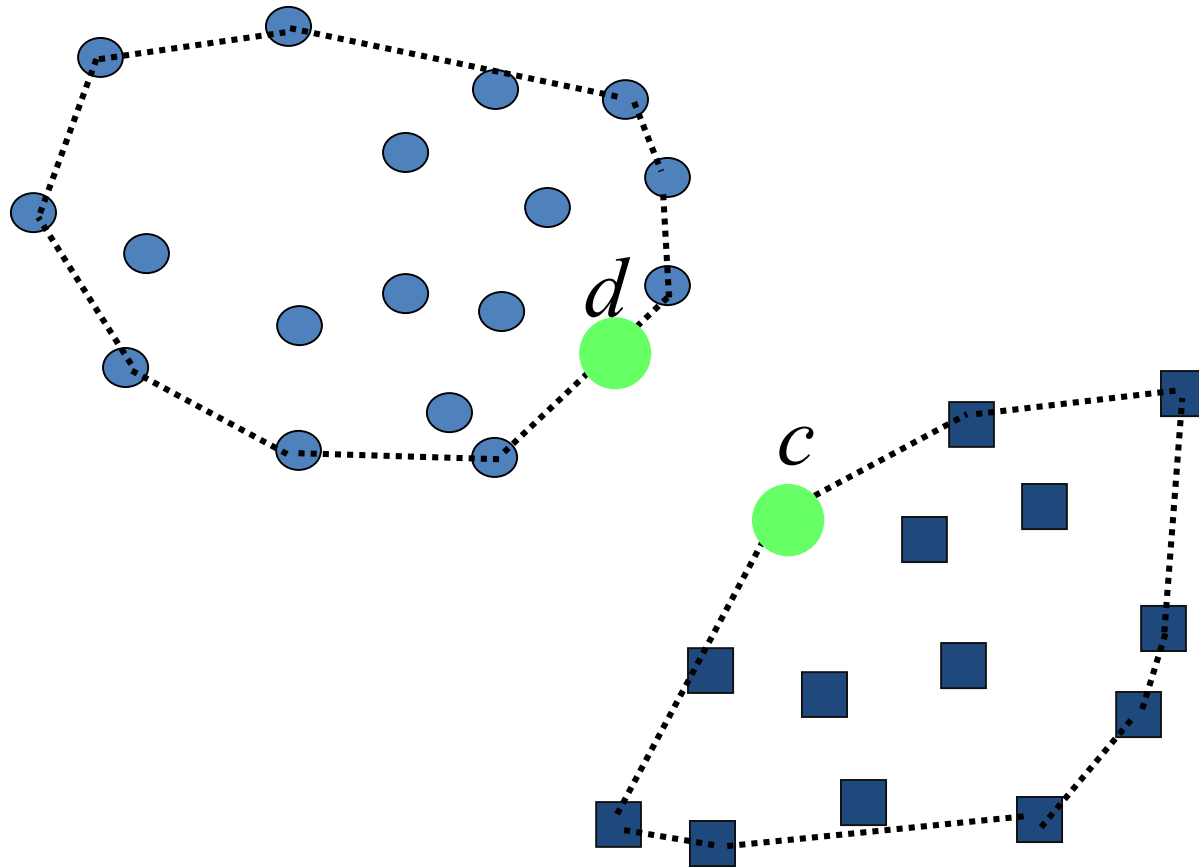
Linear SVM



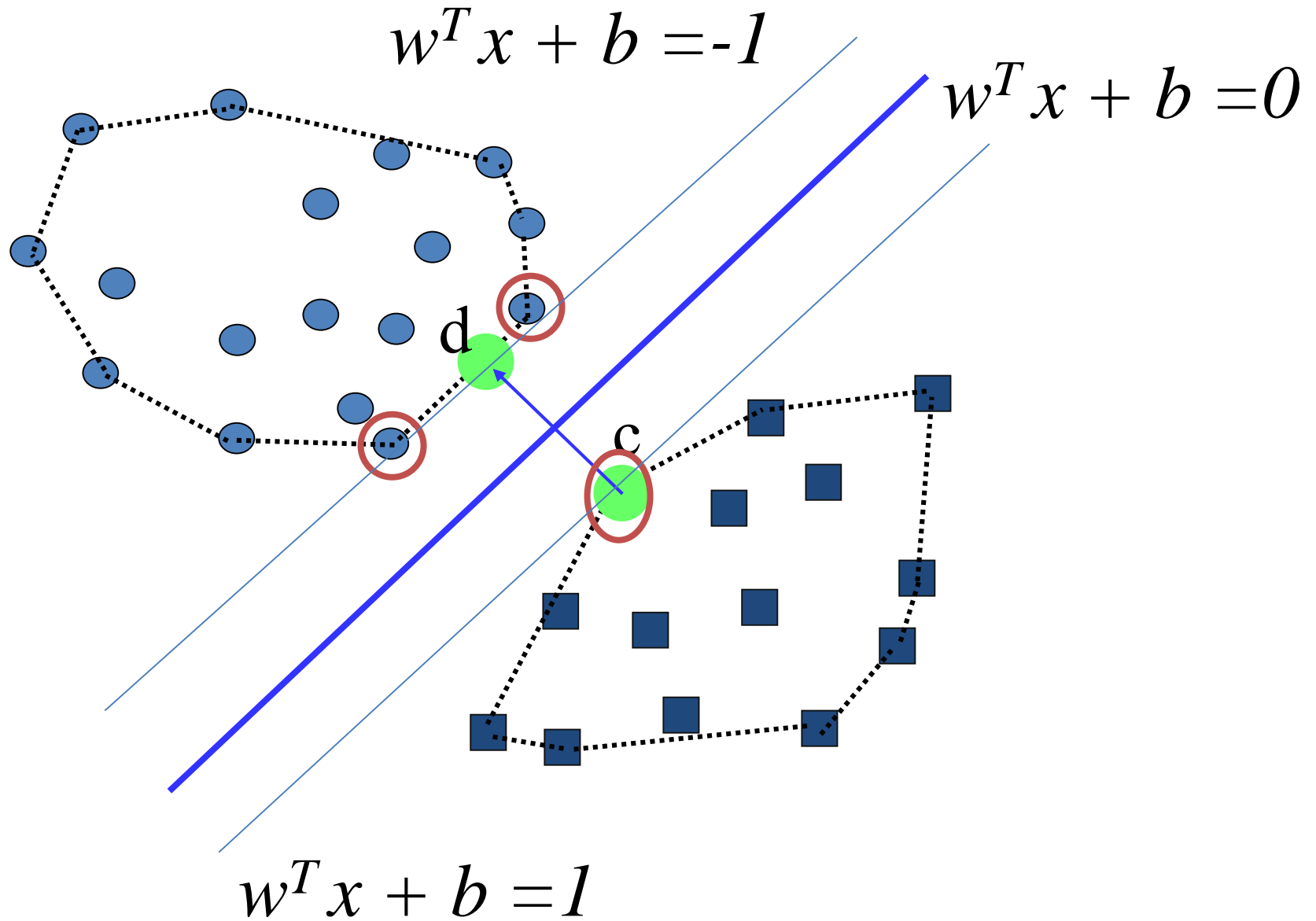
Linear SVM



Linear SVM: Find Closest Points in Convex Hulls

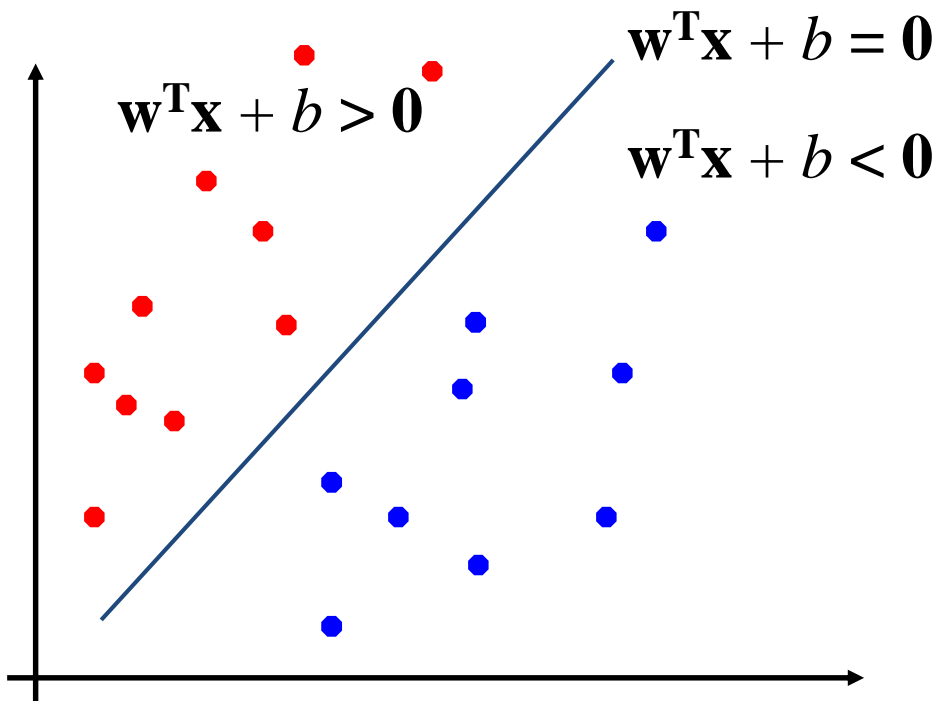


Plane Bisect Closest Points



Linear Discriminant Function

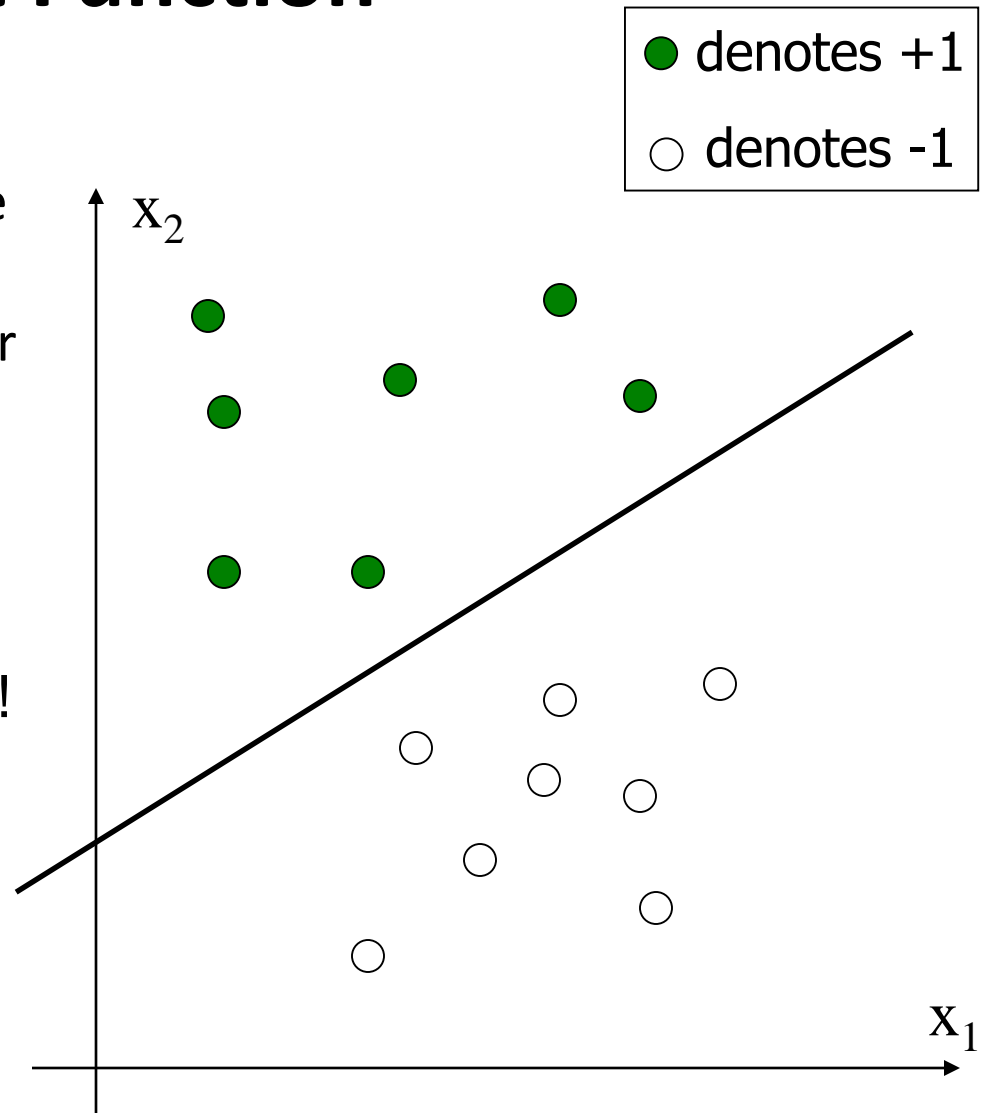
- Binary classification can be viewed as the task of separating classes in feature space:



$$f(\mathbf{x}) = \text{sign}(w^T \mathbf{x} + b)$$

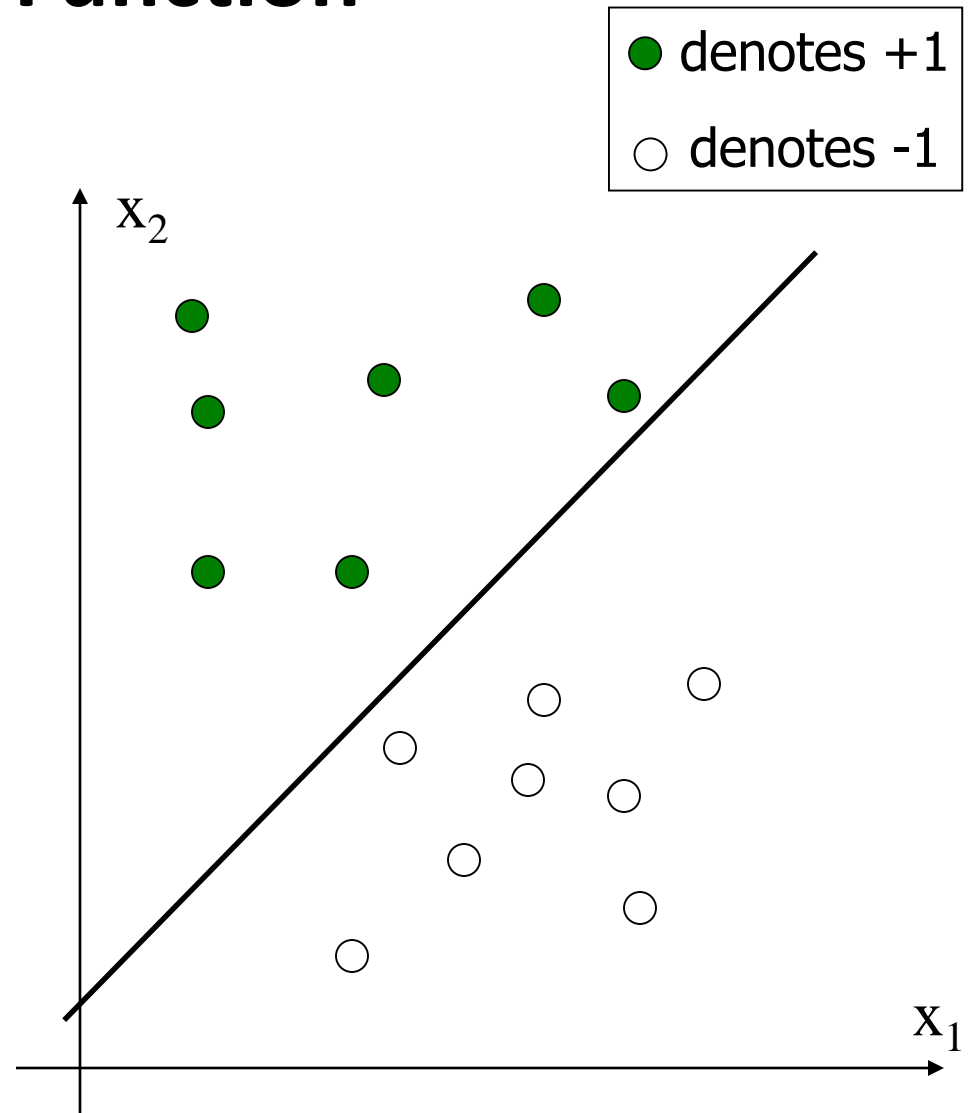
Linear Discriminant Function

- How would you classify these points using a linear discriminant function in order to minimize the error rate?
- Infinite number of answers!



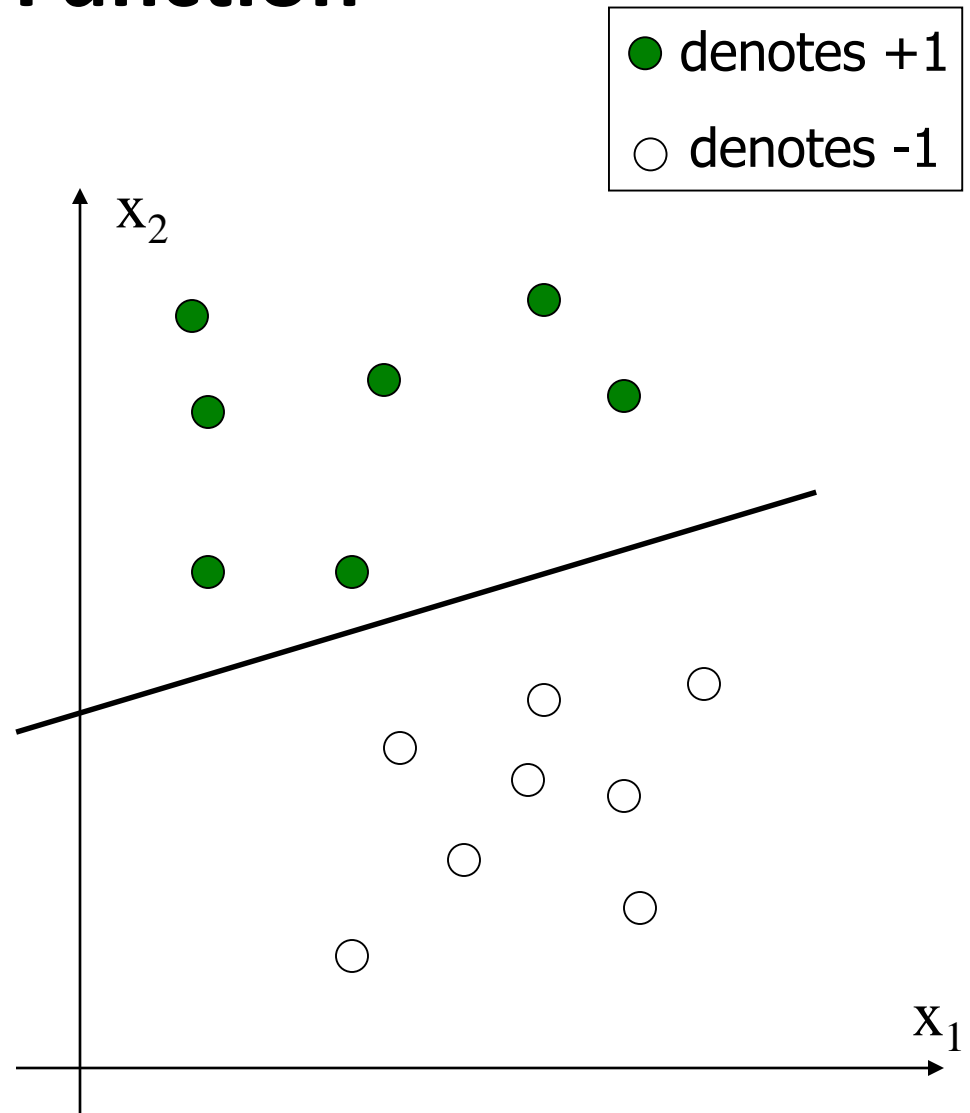
Linear Discriminant Function

- How would you classify these points using a linear discriminant function in order to minimize the error rate?
- Infinite number of answers!



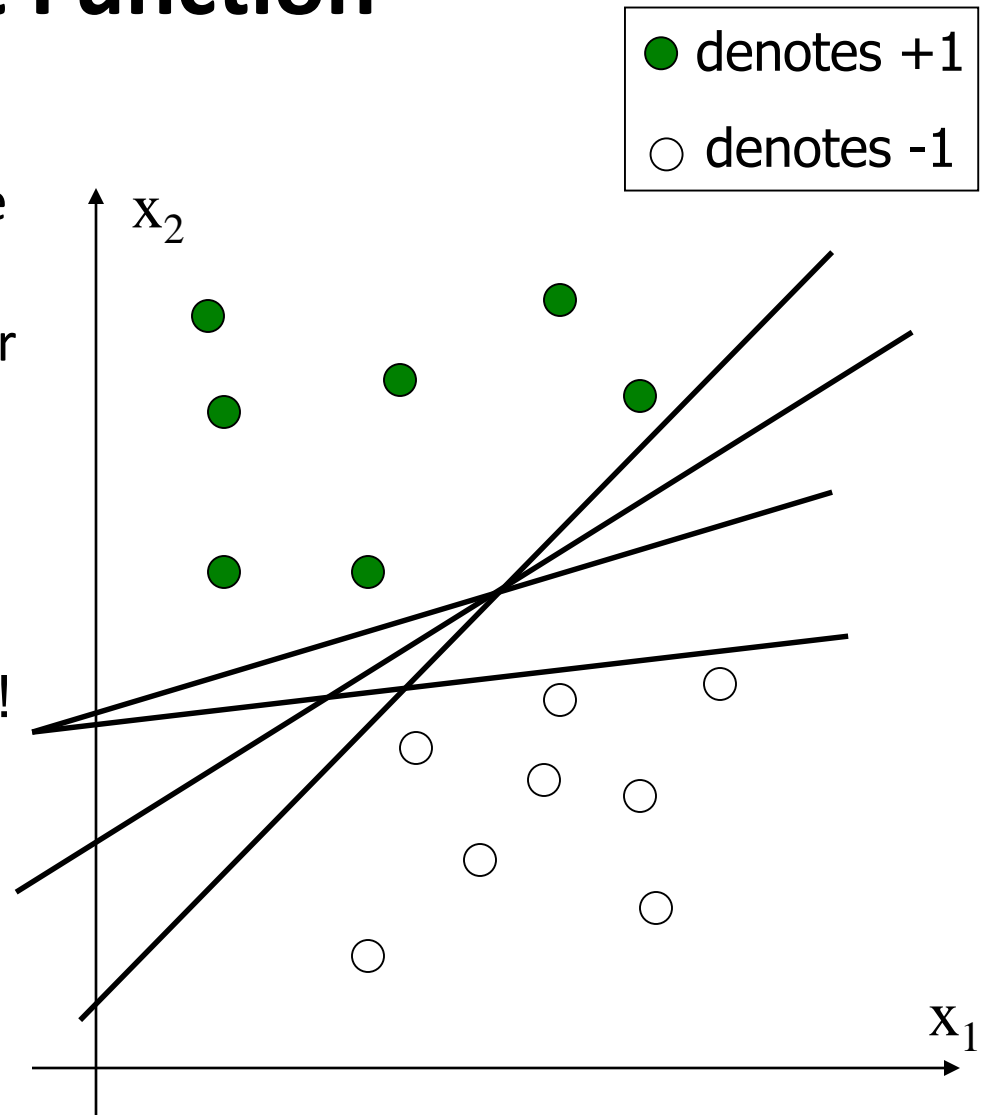
Linear Discriminant Function

- How would you classify these points using a linear discriminant function in order to minimize the error rate?
- Infinite number of answers!



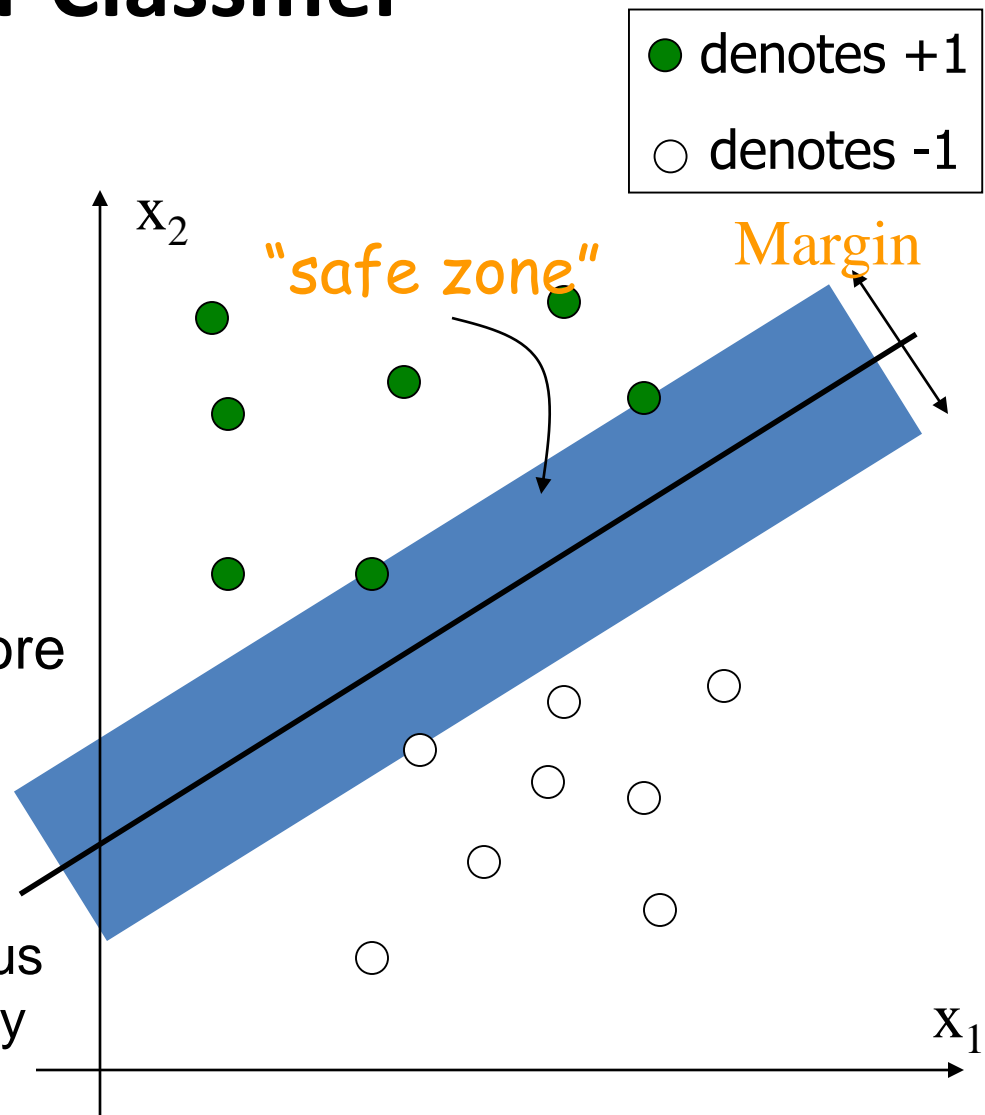
Linear Discriminant Function

- How would you classify these points using a linear discriminant function in order to minimize the error rate?
- Infinite number of answers!
- Which one is the best?



Large Margin Linear Classifier

- The linear discriminant function (classifier) with the maximum **margin** is the best
- Margin is defined as the width that the boundary could be increased by before hitting a data point
- Why it is the best?
 - Robust to outliers and thus strong generalization ability



Large Margin Linear Classifier

● denotes +1
○ denotes -1

- Given a set of data points:
 $\{(\mathbf{x}_i, y_i)\}, i = 1, 2, \dots, n$, where

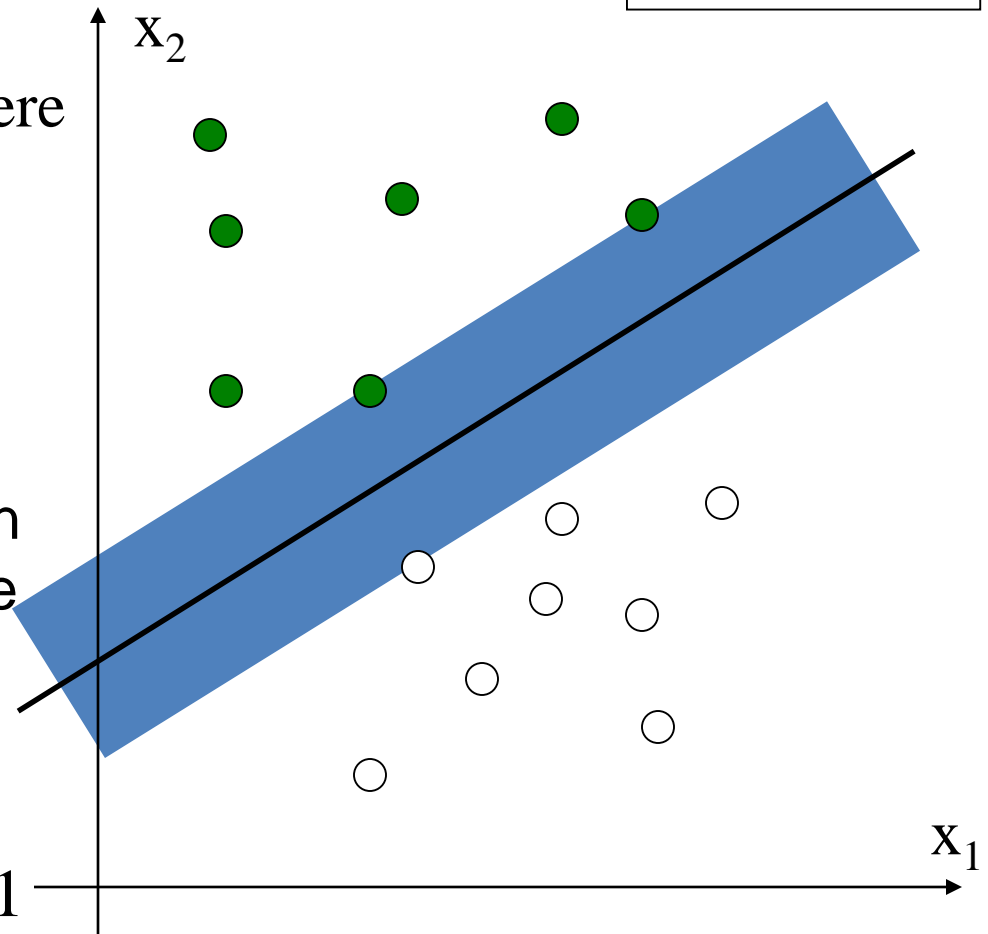
$$\text{For } y_i = +1, \quad \mathbf{w}^T \mathbf{x}_i + b > 0$$

$$\text{For } y_i = -1, \quad \mathbf{w}^T \mathbf{x}_i + b < 0$$

- With a scale transformation on both w and b , the above is equivalent to

$$\text{For } y_i = +1, \quad \mathbf{w}^T \mathbf{x}_i + b \geq 1$$

$$\text{For } y_i = -1, \quad \mathbf{w}^T \mathbf{x}_i + b \leq -1$$



Large Margin Linear Classifier

- We know that

$$\mathbf{w}^T \mathbf{x}^+ + b = 1$$

$$\mathbf{w}^T \mathbf{x}^- + b = -1$$

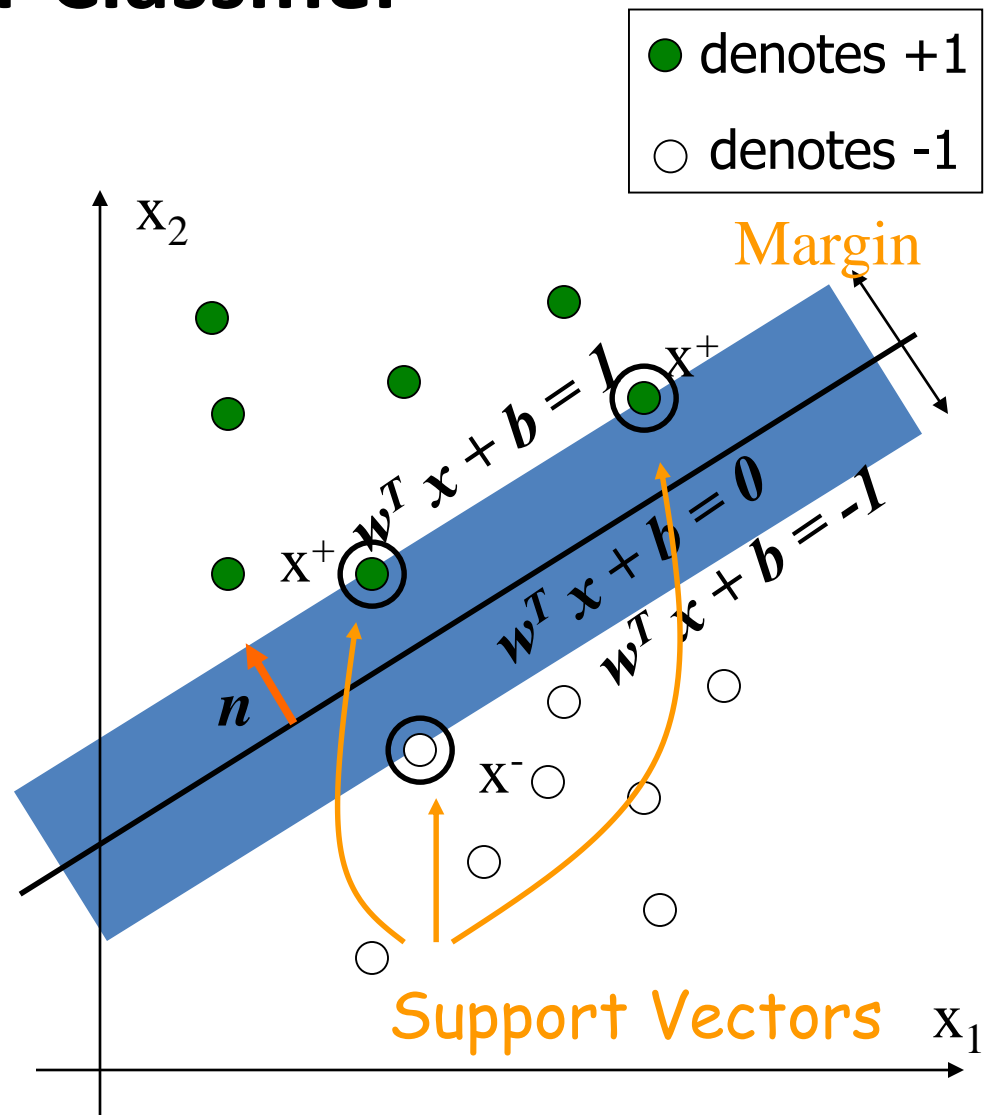


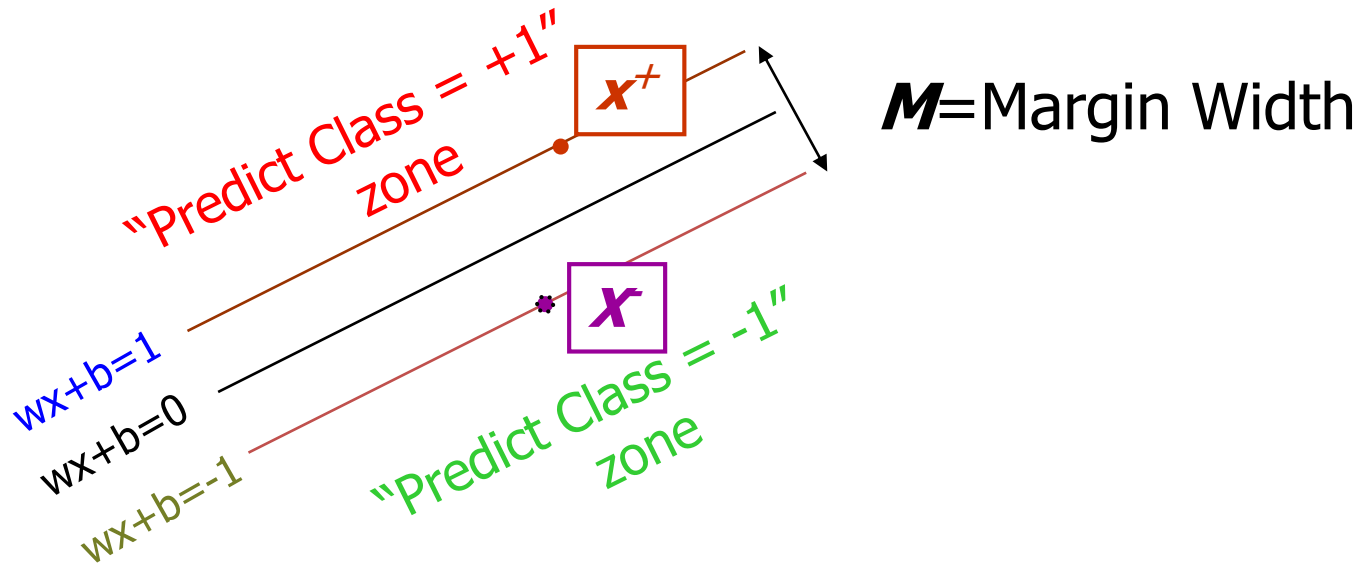
$$\mathbf{w} \cdot (\mathbf{x}^+ - \mathbf{x}^-) = 2$$

- The **margin** width is:

$$M = (\mathbf{x}^+ - \mathbf{x}^-) \cdot \mathbf{n}$$

$$= (\mathbf{x}^+ - \mathbf{x}^-) \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$$





What we know:

- $w \cdot x^+ + b = +1$
- $w \cdot x^- + b = -1$
- $w \cdot (x^+ - x^-) = 2$

$$M = \frac{(x^+ - x^-) \cdot w}{|w|} = \frac{2}{|w|}$$

Large Margin Linear Classifier

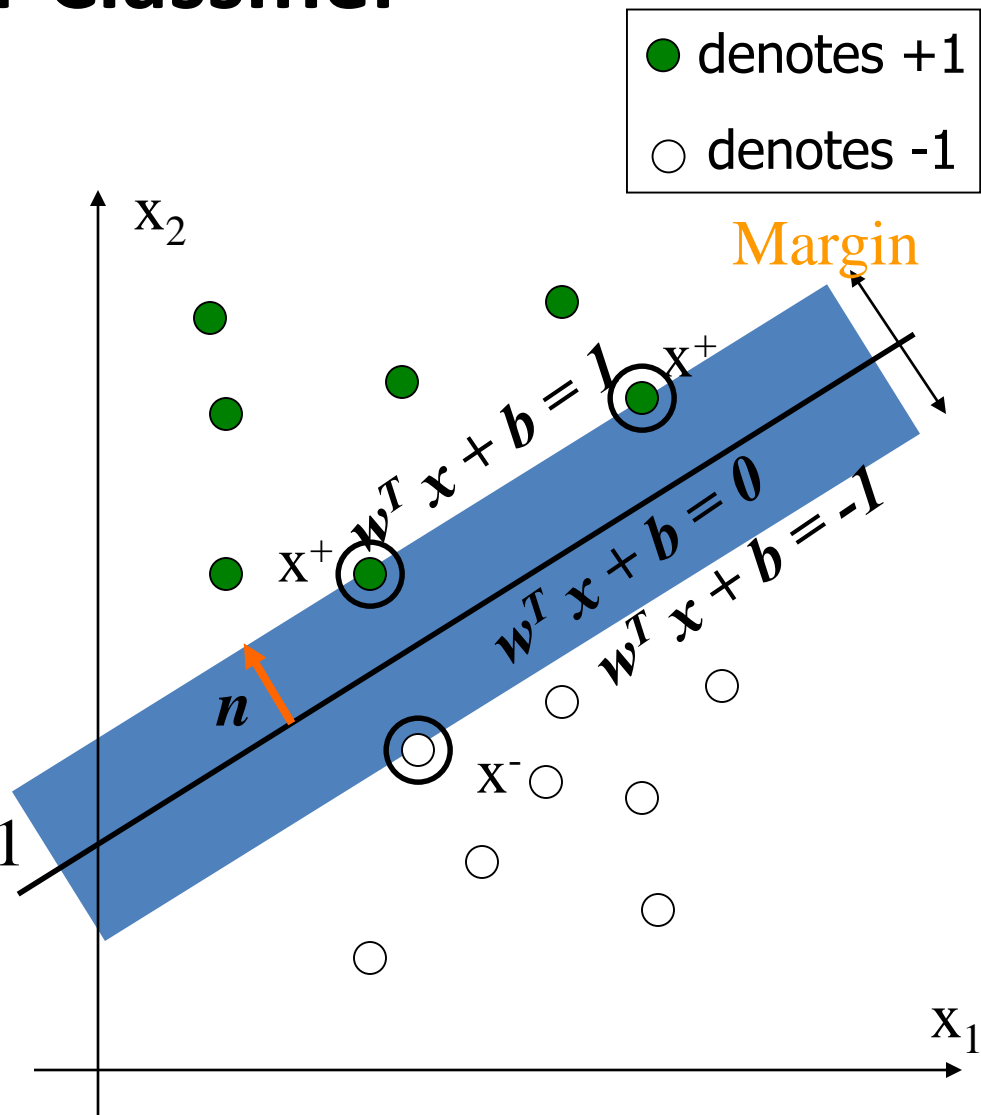
- Formulation:

$$\text{maximize } \frac{2}{\|\mathbf{w}\|}$$

such that

$$\text{For } y_i = +1, \quad \mathbf{w}^T \mathbf{x}_i + b \geq 1$$

$$\text{For } y_i = -1, \quad \mathbf{w}^T \mathbf{x}_i + b \leq -1$$



Large Margin Linear Classifier

● denotes +1
○ denotes -1

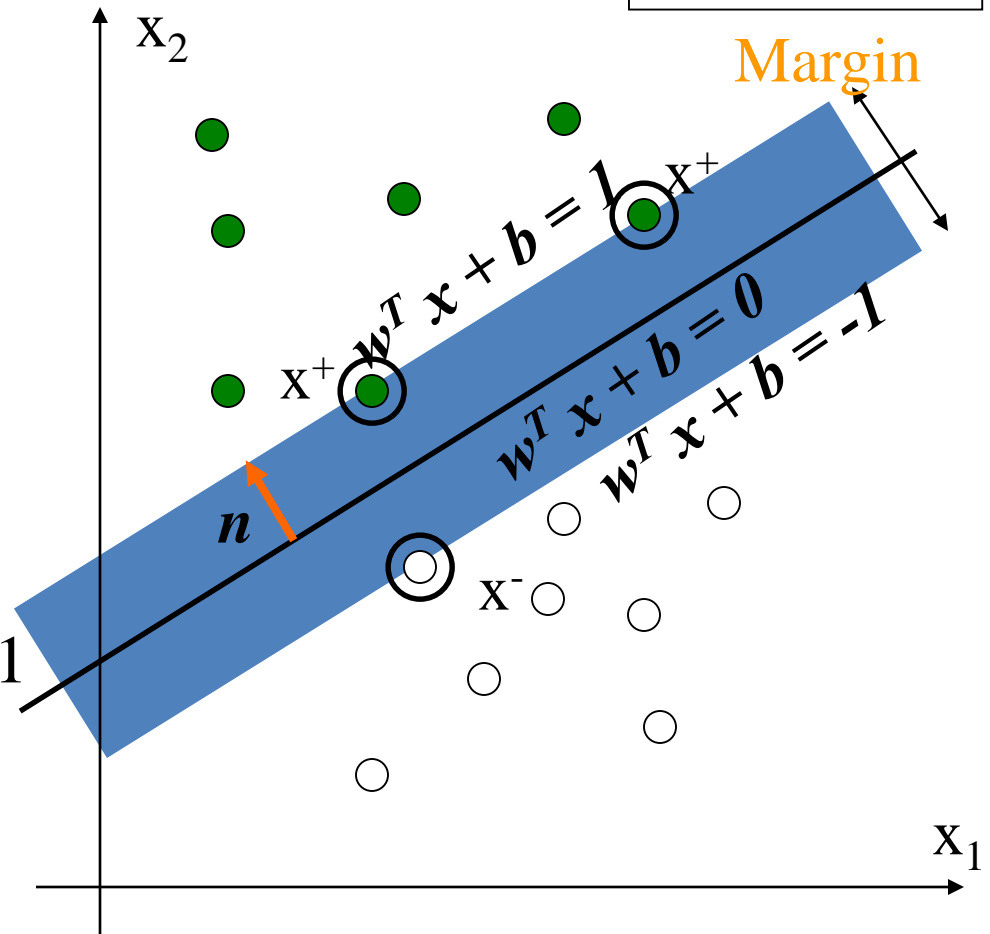
- Formulation:

$$\text{minimize } \frac{1}{2} \|\mathbf{w}\|^2$$

such that

$$\text{For } y_i = +1, \quad \mathbf{w}^T \mathbf{x}_i + b \geq 1$$

$$\text{For } y_i = -1, \quad \mathbf{w}^T \mathbf{x}_i + b \leq -1$$



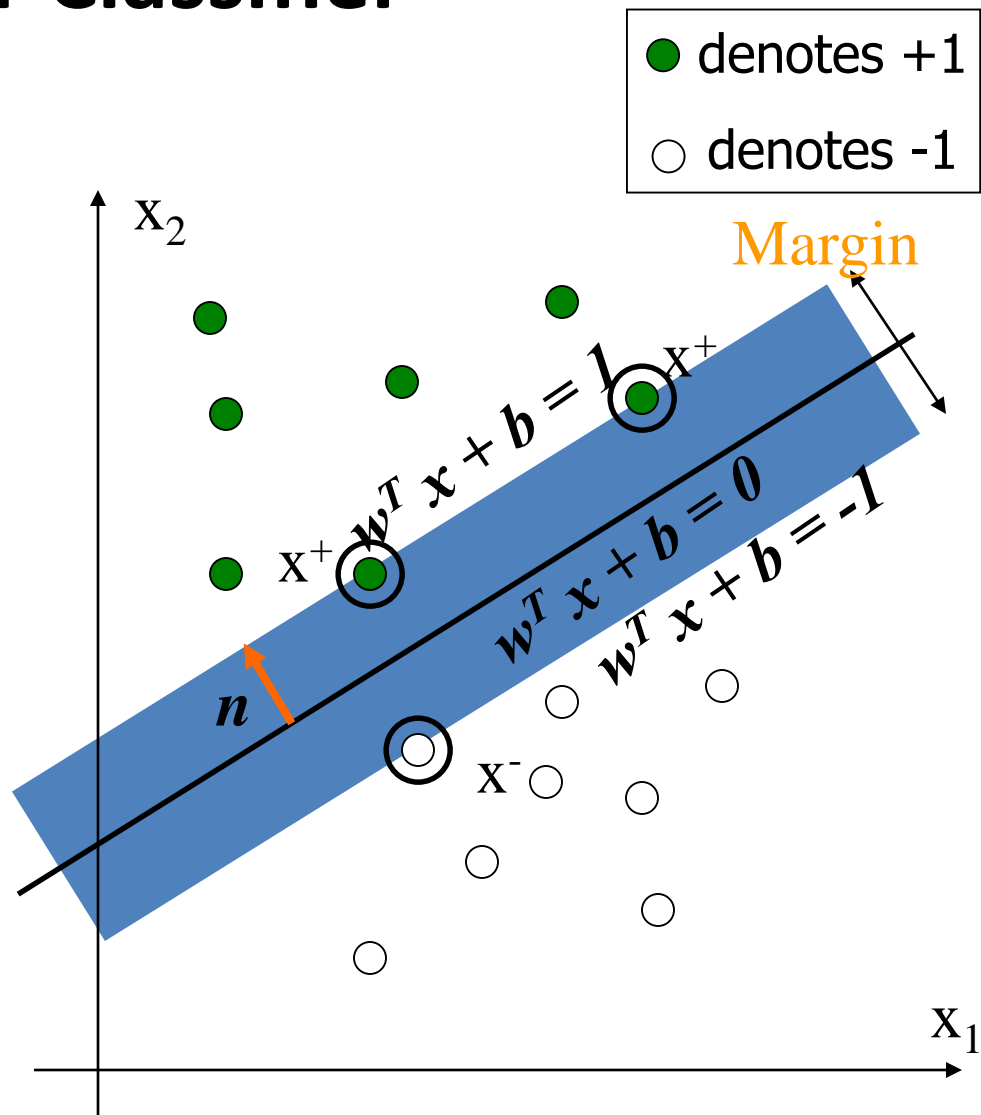
Large Margin Linear Classifier

- Formulation:

$$\text{minimize } \frac{1}{2} \|\mathbf{w}\|^2$$

such that

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$$

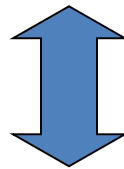


Solving the Optimization Problem

Quadratic
programming
with linear
constraints

$$\begin{aligned} & \text{minimize} && \frac{1}{2} \|\mathbf{w}\|^2 \\ & \text{s.t.} && y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \end{aligned}$$

Lagrangian
Function



$$\begin{aligned} & \text{minimize} && L_p(\mathbf{w}, b, \alpha_i) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1) \\ & \text{s.t.} && \alpha_i \geq 0 \end{aligned}$$

Solving the Optimization Problem

$$\begin{aligned} \text{minimize } L_p(\mathbf{w}, b, \alpha_i) &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1) \\ \text{s.t. } \alpha_i &\geq 0 \end{aligned}$$

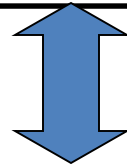
$$\frac{\partial L_p}{\partial \mathbf{w}} = 0 \quad \longrightarrow \quad \mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial L_p}{\partial b} = 0 \quad \longrightarrow \quad \sum_{i=1}^n \alpha_i y_i = 0$$

Solving the Optimization Problem

$$\begin{aligned} \text{minimize } L_p(\mathbf{w}, b, \alpha_i) &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1) \\ \text{s.t. } \alpha_i &\geq 0 \end{aligned}$$

Lagrangian Dual
Problem



$$\begin{aligned} \text{maximize } \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ \text{s.t. } \alpha_i \geq 0, \text{ and } \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$

Solving the Optimization Problem

- From KKT condition, we know:

$$\alpha_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1) = 0$$

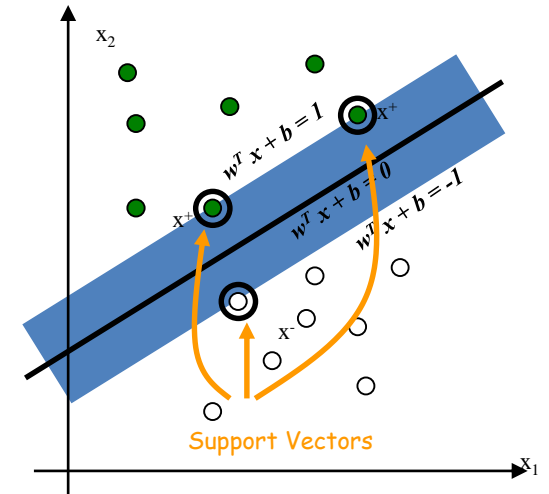
- Thus, only support vectors have $\alpha_i \neq 0$

- The solution has the form:

$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i = \sum_{i \in \text{SV}} \alpha_i y_i \mathbf{x}_i$$

get b from $y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 = 0$,

where \mathbf{x}_i is support vector



Solving the Optimization Problem

- The linear discriminant function is:

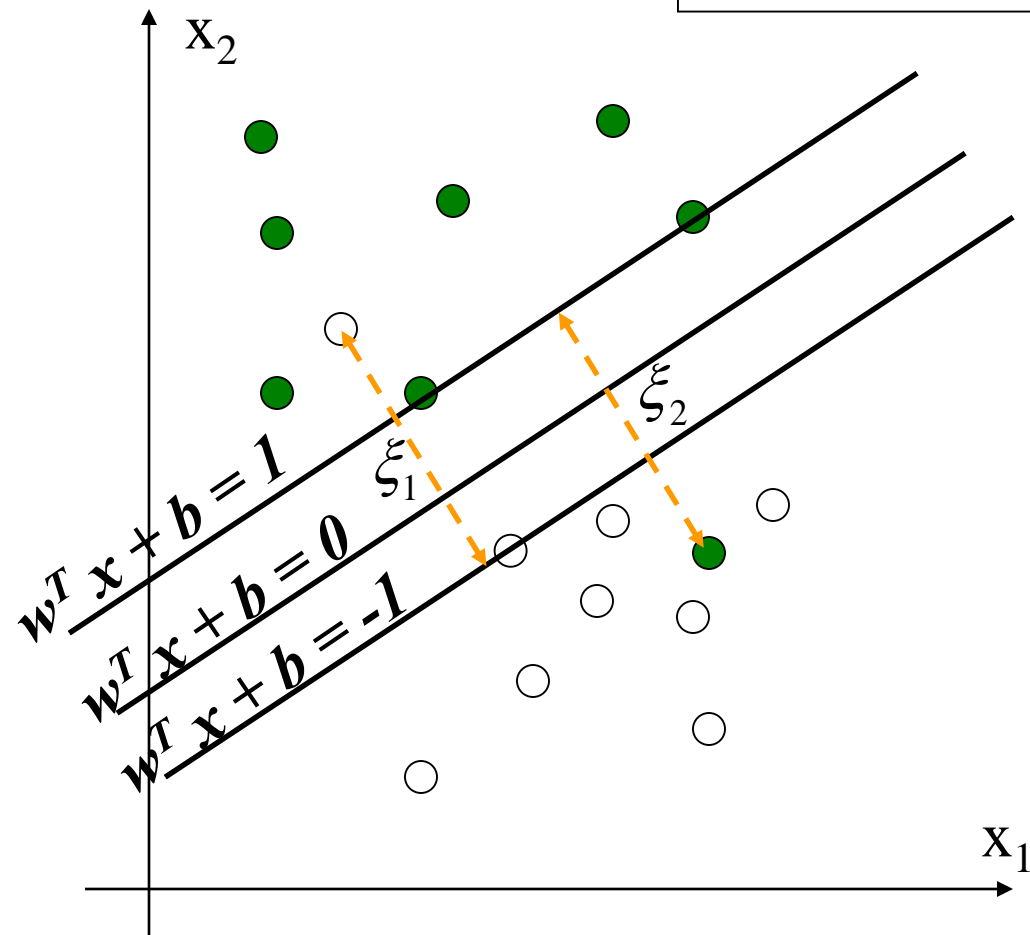
$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \sum_{i \in SV} \alpha_i \mathbf{x}_i^T \mathbf{x} + b$$

- Notice it relies on a *dot product* between the test point \mathbf{x} and the support vectors \mathbf{x}_i
- Also keep in mind that solving the optimization problem involved computing the *dot products* $\mathbf{x}_i^T \mathbf{x}_j$ between all pairs of training points

Non-ideal Situations

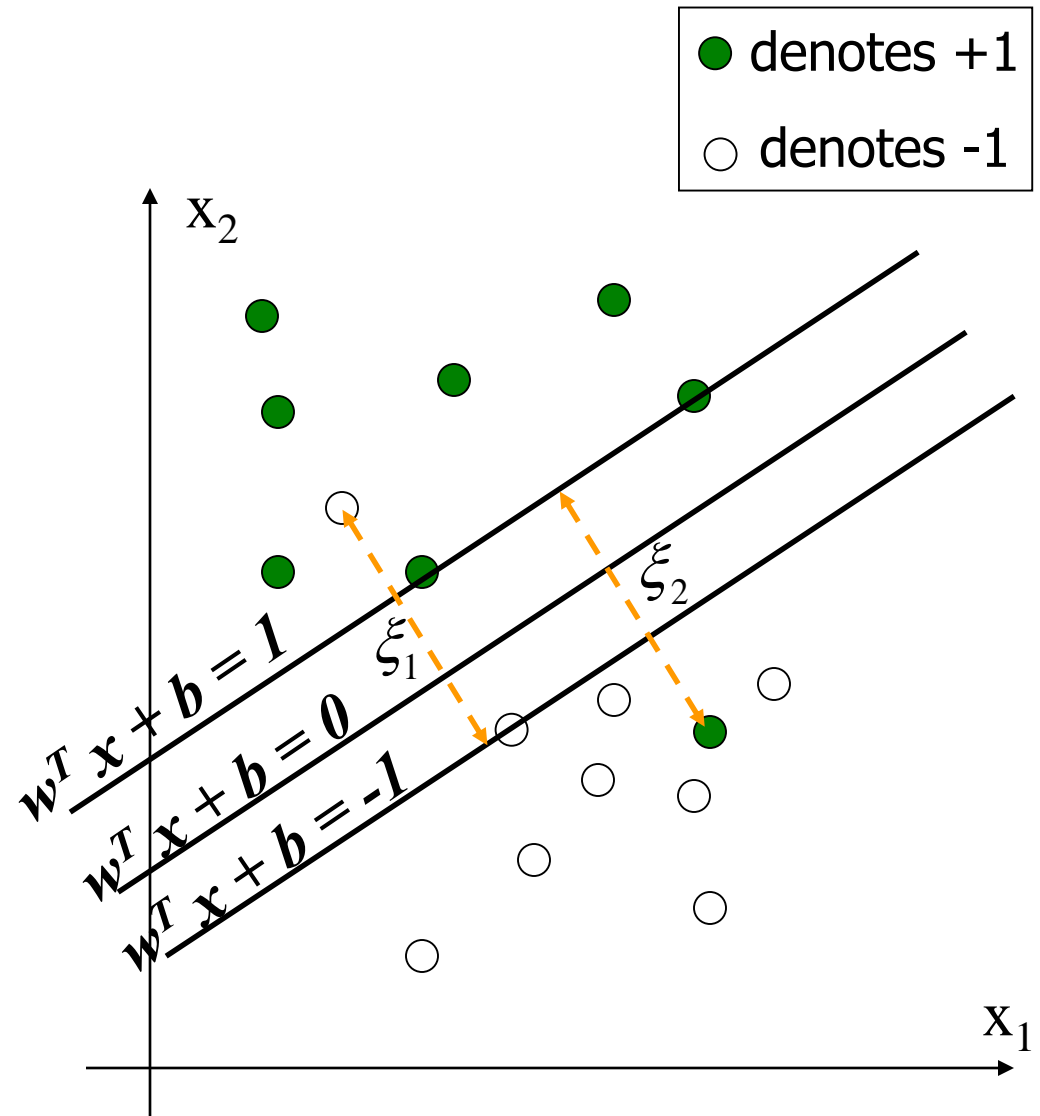
- Data partitioning has mistakes

● denotes +1
○ denotes -1



Large Margin Linear Classifier

- What if data is not linear separable? (noisy data, outliers, etc.)
- Slack variables ξ_i can be added to allow misclassification of difficult or noisy data points



Introducing slack variables

- Slack variables are constrained to be non-negative. When they are greater than zero they allow us to cheat by putting the plane closer to the datapoint than the margin. So we need to minimize the amount of cheating. This means we have to pick a value for lambda (this sounds familiar!)

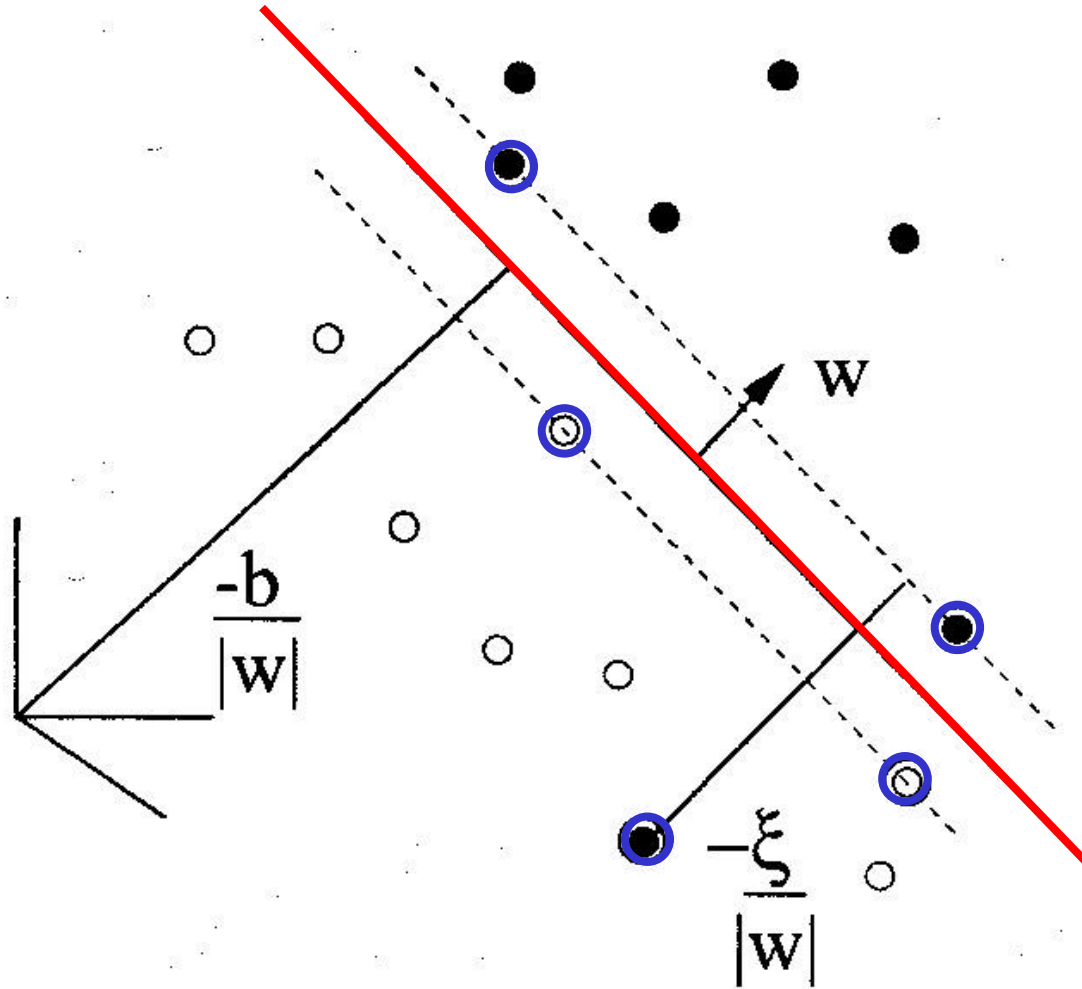
$$\mathbf{w} \cdot \mathbf{x}^c + b \geq +1 - \xi^c \quad \text{for positive cases}$$

$$\mathbf{w} \cdot \mathbf{x}^c + b \leq -1 + \xi^c \quad \text{for negative cases}$$

$$\text{with } \xi^c \geq 0 \quad \text{for all } c$$

$$\text{and } \frac{\|\mathbf{w}\|^2}{2} + \lambda \sum_c \xi^c \quad \text{as small as possible}$$

A picture of the best plane with a slack variable



Large Margin Linear Classifier

- **Formulation:**

$$\text{minimize } \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i$$

such that

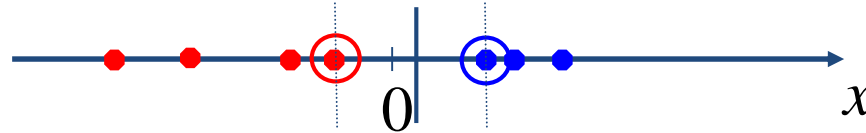
$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

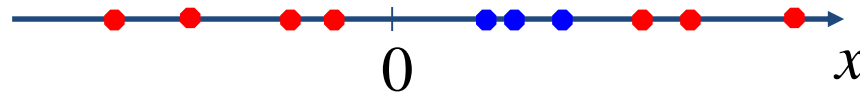
- Parameter C can be viewed as a way to control over-fitting.

Non-linear SVMs

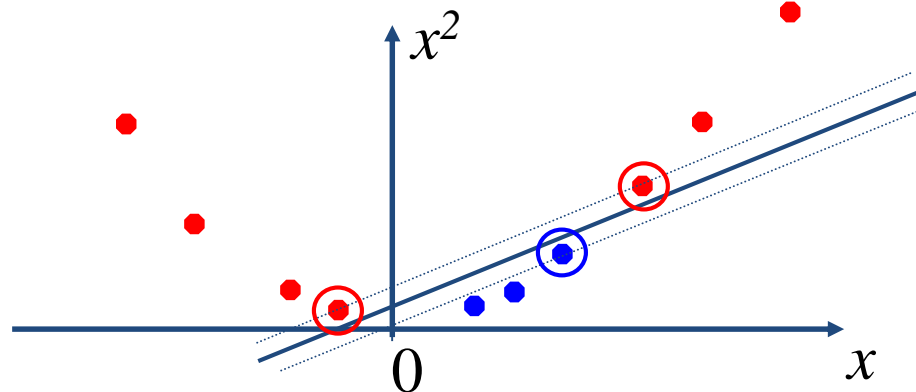
- Datasets that are **linearly separable** with noise work out great:



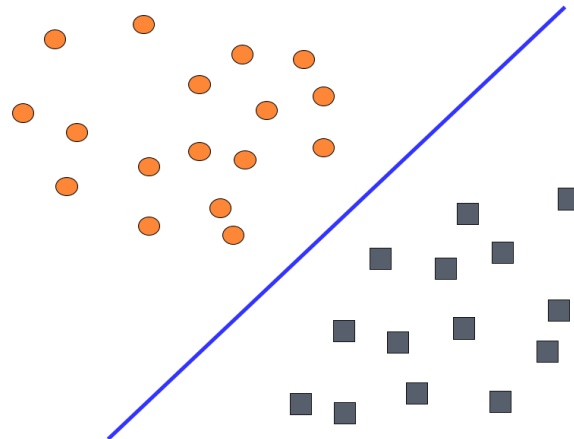
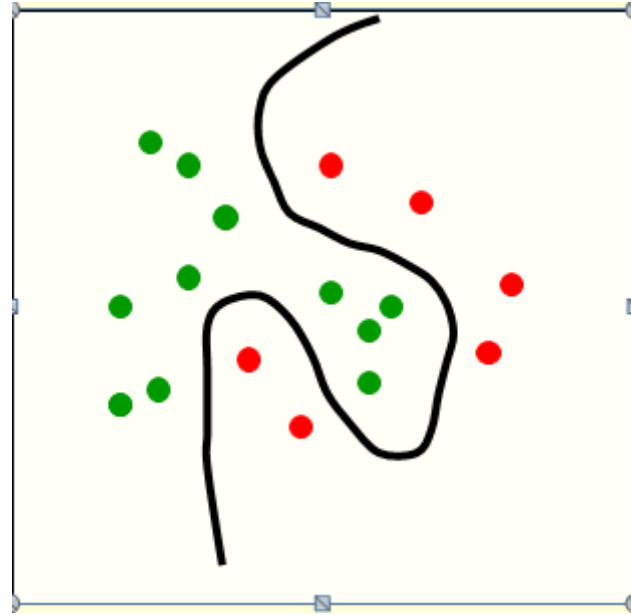
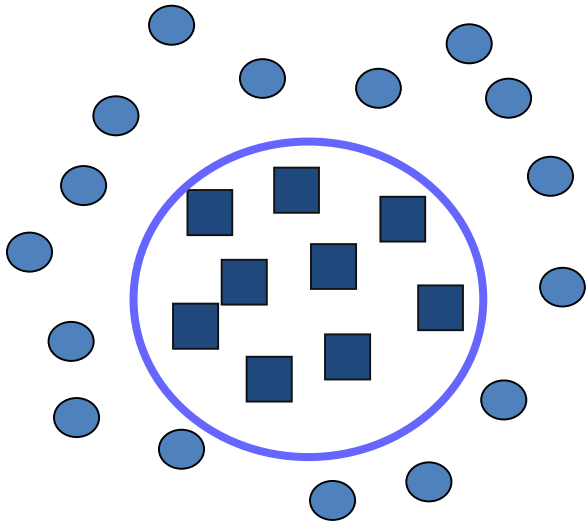
- But what are we going to do if the dataset is just too hard?



- How about... **mapping data to a higher-dimensional space:**



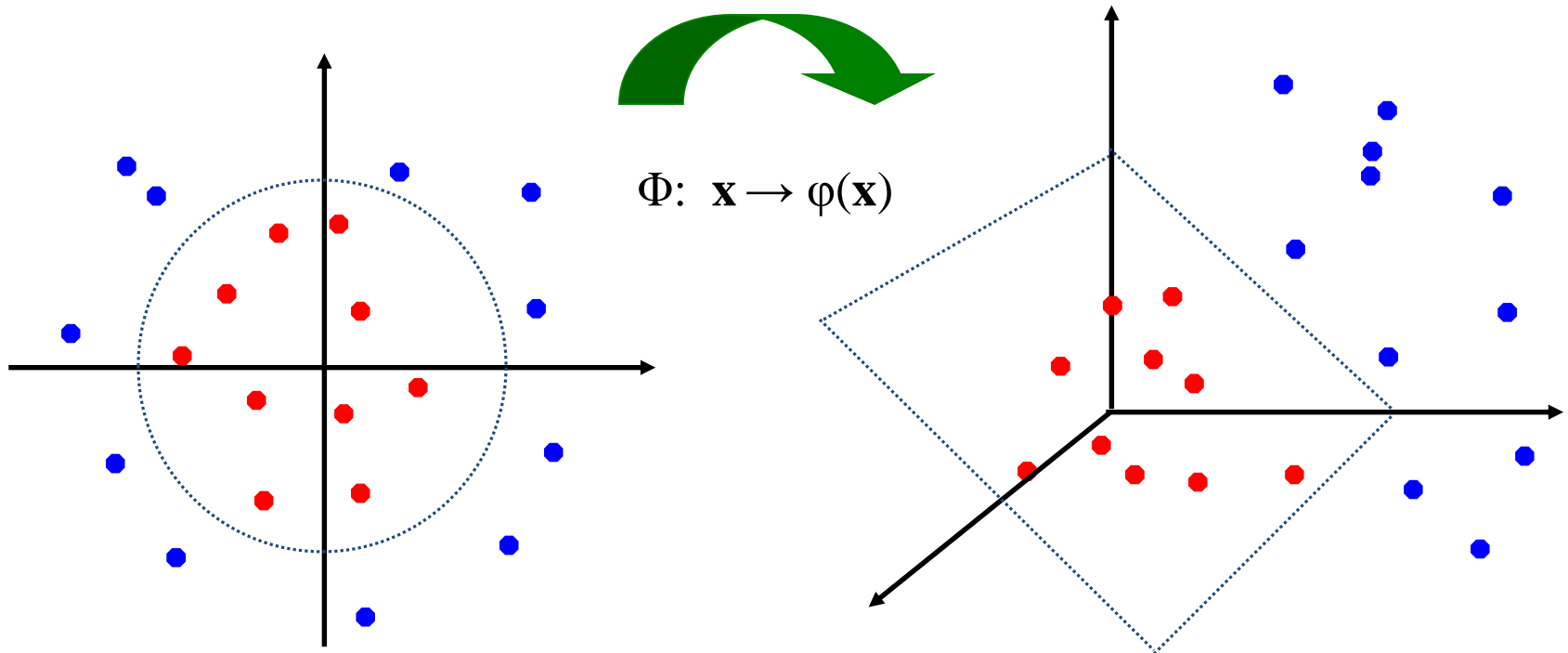
Nonlinear Classification



Linear Separable

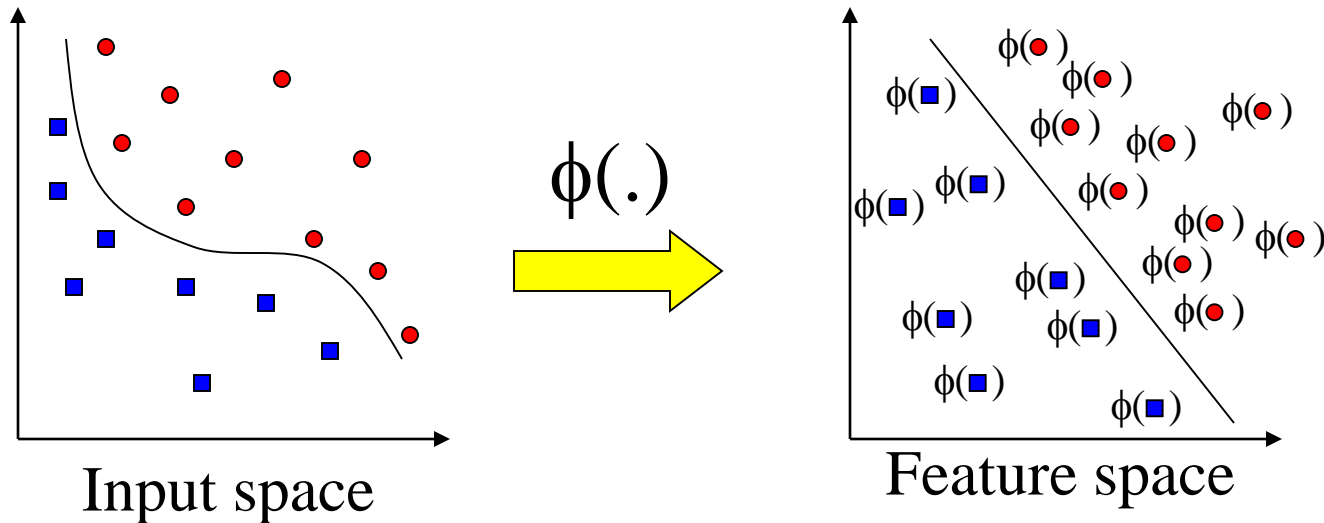
Non-linear SVMs: Feature Space

- General idea: the original input space can be **mapped to some higher-dimensional feature space** where the training set is separable:



Why high-dimensional space can have better separation?

Transforming the Data



- Computation in the feature space can be costly because it is high dimensional
 - The feature space is typically infinite-dimensional!
- The kernel trick comes to rescue

Nonlinear SVMs: The Kernel Trick

- With this mapping, our discriminant function is now:

$$g(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = \sum_{i \in SV} \alpha_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b$$

- No need to know this mapping explicitly, because we only use the **dot product** of feature vectors in both the training and test.
- A **kernel function** is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:

$$K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Nonlinear SVMs: The Kernel Trick

- An example:

2-dimensional vectors $\mathbf{x}=[x_1 \ x_2]$;

let $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$,

Need to show that $K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\varphi}(\mathbf{x}_i)^T \boldsymbol{\varphi}(\mathbf{x}_j)$:

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= (1 + \mathbf{x}_i^T \mathbf{x}_j)^2, \\ &= 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2} \\ &= [1 \ x_{i1}^2 \ \sqrt{2} x_{i1} x_{i2} \ x_{i2}^2 \ \sqrt{2} x_{i1} \ \sqrt{2} x_{i2}]^T [1 \ x_{j1}^2 \ \sqrt{2} x_{j1} x_{j2} \ x_{j2}^2 \ \sqrt{2} x_{j1} \ \sqrt{2} x_{j2}] \\ &= \boldsymbol{\varphi}(\mathbf{x}_i)^T \boldsymbol{\varphi}(\mathbf{x}_j), \quad \text{where } \boldsymbol{\varphi}(\mathbf{x}) = [1 \ x_1^2 \ \sqrt{2} x_1 x_2 \ x_2^2 \ \sqrt{2} x_1 \ \sqrt{2} x_2] \end{aligned}$$

Nonlinear SVMs: The Kernel Trick

- Examples of commonly-used kernel functions:

- Linear kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$

- Polynomial kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$

- Gaussian (Radial-Basis Function (RBF)) kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

- Sigmoid:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta_0 \mathbf{x}_i^T \mathbf{x}_j + \beta_1)$$

- In general, functions that satisfy *Mercer's condition* can be kernel functions.

Nonlinear SVM: Optimization

- Formulation: (Lagrangian Dual Problem)

$$\text{maximize } \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

such that

$$0 \leq \alpha_i \leq C$$

$$\sum_{i=1}^n \alpha_i y_i = 0$$

- The solution of the discriminant function is

$$g(\mathbf{x}) = \sum_{i \in \text{SV}} \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b$$

- The optimization technique is the same.

Support Vector Machine: Algorithm

- 1. Choose a kernel function
- 2. Choose a value for C
- 3. Solve the quadratic programming problem
(many software packages available)
- 4. Construct the discriminant function from the support vectors

Some Issues

- **Choice of kernel**
 - Gaussian or polynomial kernel is default
 - if ineffective, more elaborate kernels are needed
 - domain experts can give assistance in formulating appropriate similarity measures
- **Choice of kernel parameters**
 - e.g. σ in Gaussian kernel
 - σ is the distance between closest points with different classifications
 - In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.
- **Optimization criterion** – Hard margin v.s. Soft margin
 - a lengthy series of experiments in which various parameters are tested

Strengths and Weaknesses of SVM

- Strengths
 - Training is relatively easy
 - No local optimal, unlike in neural networks
 - It scales relatively well to high dimensional data
 - Tradeoff between classifier complexity and error can be controlled explicitly
 - Non-traditional data like strings and trees can be used as input to SVM, instead of feature vectors
 - By performing logistic regression (Sigmoid) on the SVM output of a set of data can map SVM output to probabilities.
- Weaknesses
 - Need to choose a “good” kernel function.

Summary: Support Vector Machine

- 1. Large Margin Classifier
 - Better generalization ability & less overfitting
- 2. The Kernel Trick
 - Map data points to higher dimensional space in order to make them linearly separable.
 - Since only dot product is used, we do not need to represent the mapping explicitly.

What about **multi-class SVMs**?

- Unfortunately, there is no “definitive” multi-class SVM formulation
- In practice, we have to obtain a multi-class SVM by combining multiple two-class SVMs
- One vs. others
 - Training: learn an SVM for each class vs. the others
 - Testing: apply each SVM to test example and assign to it the class of the SVM that returns the highest decision value
- One vs. one
 - Training: learn an SVM for each pair of classes
 - Testing: each learned SVM “votes” for a class to assign to the test example

Three Approaches to K-Class SVM

- **K one-versus-residue (OVR) binary SVM**
- $\frac{K(K-1)}{2}$ pairwise binary SVM
- **One K-Class SVM**

$$\max_W \left[\sum_i w_{y^i}^\top x^i - \max_j (1\{j \neq y^i\} + w_j^\top x^i) \right]$$



$$\min_{w_1, \dots, w_K} \frac{1}{2} \|(w_1, \dots, w_K)\|^2 + C \sum_{ik} \xi_{ik}$$

$$\text{s.t.} \quad \forall (i, k), \quad w_{y^i}^\top x^i - w_k^\top x^i \geq 1\{k \neq y^i\} - \xi_{ik}$$

Three Approaches to K-Class SVM

- **K one-versus-residue (OVR) binary SVM**
 - **Advantages**
 - **Disadvantages**

Three Approaches to K-Class SVM

- $\frac{K(K-1)}{2}$ pairwise binary SVM
 - Advantages
 - Disadvantages

Three Approaches to K-Class SVM

- One K-Class SVM

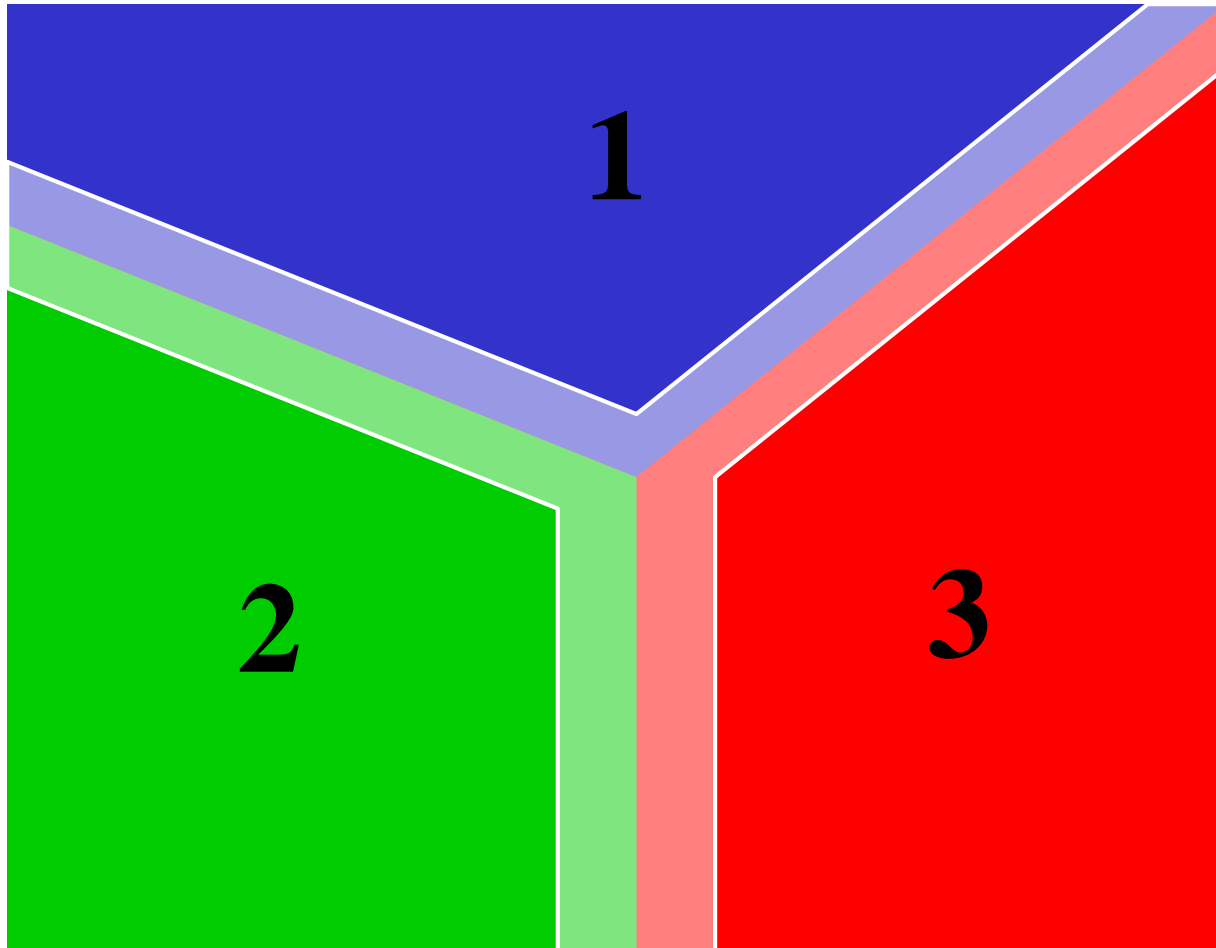
$$\max_W \left[\sum_i w_{y^i}^\top x^i - \max_j (1\{j \neq y^i\} + w_j^\top x^i) \right]$$



$$\min_{w_1, \dots, w_K} \frac{1}{2} \|(w_1, \dots, w_K)\|^2 + C \sum_{ik} \xi_{ik}$$

$$\text{s.t.} \quad \forall(i, k), \quad w_{y^i}^\top x^i - w_k^\top x^i \geq 1\{k \neq y^i\} - \xi_{ik}$$

K-class SVM



Multi-class SVM

Intuitive formulation: without regularization / for the separable case

$$\max_W \left[\sum_i w_{y^i}^\top x^i - \max_j (1\{j \neq y^i\} + w_j^\top x^i) \right]$$

Primal problem: QP

$$\begin{aligned} \min_{w_1, \dots, w_K} \quad & \frac{1}{2} \|(w_1, \dots, w_K)\|^2 + C \sum_{ik} \xi_{ik} \\ \text{s.t.} \quad & \forall (i, k), \quad w_{y^i}^\top x^i - w_k^\top x^i \geq 1\{k \neq y^i\} - \xi_{ik} \end{aligned}$$

Solved in the dual formulation, also Quadratic Program

Main advantage: Sparsity (but not systematic)

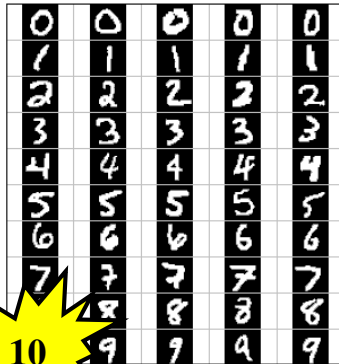
- Speed with SMO (heuristic use of sparsity)
- Sparse solutions

Drawbacks:

- Need to recalculate or store $x_i^\top x_j$
- Outputs not probabilities

Real world classification problems

Digit recognition



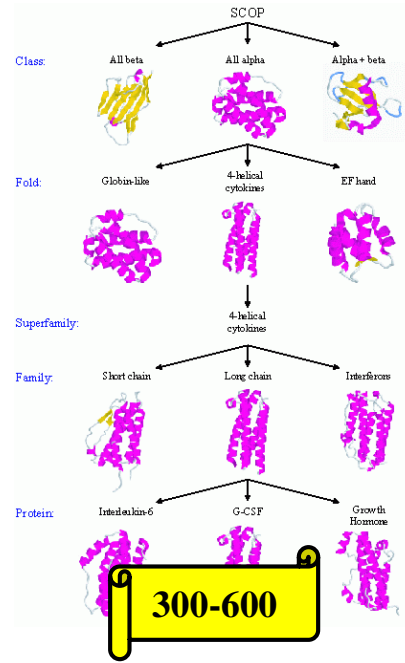
10

Phoneme recognition

ɪ READ	ɪ SIT	ʊ BOOK	u: TOO	ɪə HERE	eɪ DAY	
e MEN	ə AMERICA	ɜ: WORD	ɔ: SORT	ʊə TOUR	ɔɪ BOY	əʊ GO
æ CAT	ʌ BUT	ɑ: PART	ɒ NOT	eə WEAR	ɑɪ MY	ɑʊ HOW
p PIG	b BED	t TIME	d DO	tʃ CHURCH	dʒ JUDGE	k KILO
g GO	f FISH	θ THINK	ð THE	s SIX	z ZOO	ʃ SHORT
ʒ CASUAL	ŋ SING	h HELLO	l LIVE	r READ	w WINDOW	j YES

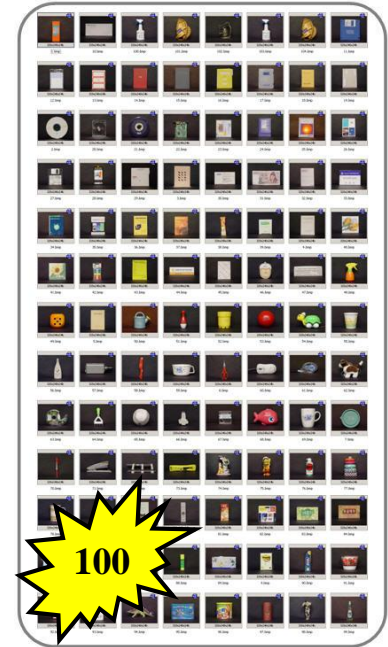
50

Automated protein classification



300-600

Object recognition



100

<http://www.gloe.umd.edu/~zhejin/recog.html>

- The number of classes is sometimes big
- The multi-class algorithm can be heavy

[Waibel, Hanzawa, Hinton, Shikano, Lang 1989]

Combining binary classifiers

One-vs-all For each class build a classifier for that class vs the rest

- Often very imbalanced classifiers (use asymmetric regularization)

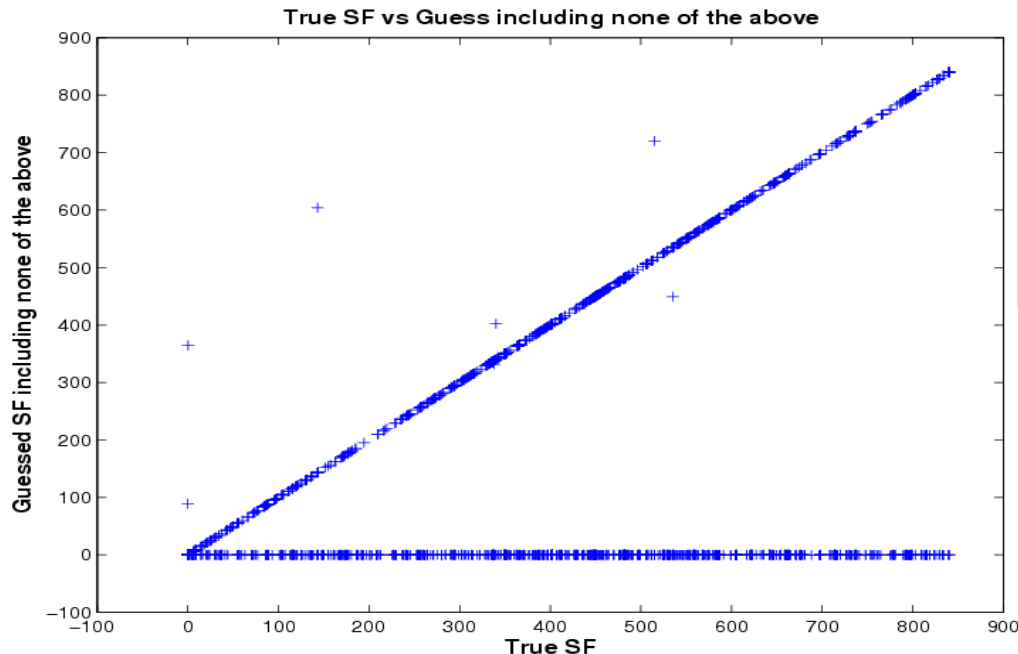
All-vs-all For each class build a classifier for that class vs the rest

- A priori a large number of classifiers $\binom{n}{2}$ to build *but...*
 - The pairwise classification are way much faster
 - The classifications are balanced (easier to find the best regularization)

... so that in many cases it is clearly faster than one-vs-all

Confusion Matrix

- Visualize which classes are more difficult to learn
- Can also be used to compare two different classifiers
- Cluster classes and go hierarchical [Godbole, '02]



Classification of 20 news groups

Predicted classes

Actual classes

Classname	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
alt.atheism	1	251	6	1	3	32	1	1	2	1	2	0	0	0	0	0	0	0	0	0
soc.religion.christian	2	9	277	0	1	6	0	0	1	0	0	0	0	1	2	2	0	0	0	1
sci.space	3	3	1	273	1	0	1	2	0	1	1	9	0	0	1	2	3	0	0	1
talk.politics.misc	4	2	0	3	213	24	3	0	17	3	0	0	0	0	0	1	0	1	33	0
talk.religion.misc	5	88	36	2	23	132	0	1	0	0	0	0	0	0	0	2	0	1	15	0
rec.autos	6	0	0	0	3	1	272	0	0	0	7	1	2	1	6	4	1	0	0	2
comp.windows.x	7	1	1	2	1	0	1	246	0	2	2	30	5	3	1	1	2	1	1	0
talk.politics.mideast	8	0	3	1	18	0	0	0	275	0	1	0	0	0	0	0	0	1	1	0
sci.crypt	9	1	0	1	2	1	0	3	0	284	0	3	0	1	0	0	1	0	0	3
rec.motorcycles	10	0	0	0	1	0	4	1	0	0	286	1	2	0	1	2	1	0	0	1
comp.graphics	11	0	1	2	1	1	0	10	1	2	0	243	23	7	3	3	3	0	0	0
comp.sys.ibm.pc.hardware	12	0	0	0	0	0	2	7	0	1	0	5	243	23	12	3	1	3	0	0
comp.sys.mac.hardware	13	0	0	1	1	0	2	1	0	0	0	7	10	260	8	9	1	0	0	0
sci.electronics	14	1	0	1	0	1	5	2	0	2	0	7	13	13	245	6	3	0	1	0
misc.forsale	15	0	1	4	2	0	12	1	0	0	4	1	19	10	8	233	1	0	1	1
sci.med	16	0	1	5	0	1	1	0	0	1	2	0	2	7	2	275	0	1	1	1
comp.os.mswindows.misc	17	1	0	2	0	1	1	58	1	3	0	38	71	17	3	6	0	97	1	0
rec.sport.baseball	18	2	1	1	0	0	0	0	0	0	4	0	0	0	1	1	0	282	1	7
talk.politics.guns	19	0	0	0	9	5	1	0	0	1	0	0	0	1	0	0	1	1	281	0
rec.sport.hockey	20	0	1	0	0	0	1	0	0	0	2	0	0	1	1	0	0	0	3	0

[Godbole, '02]

BLAST classification of proteins in 850 superfamilies

Calibration

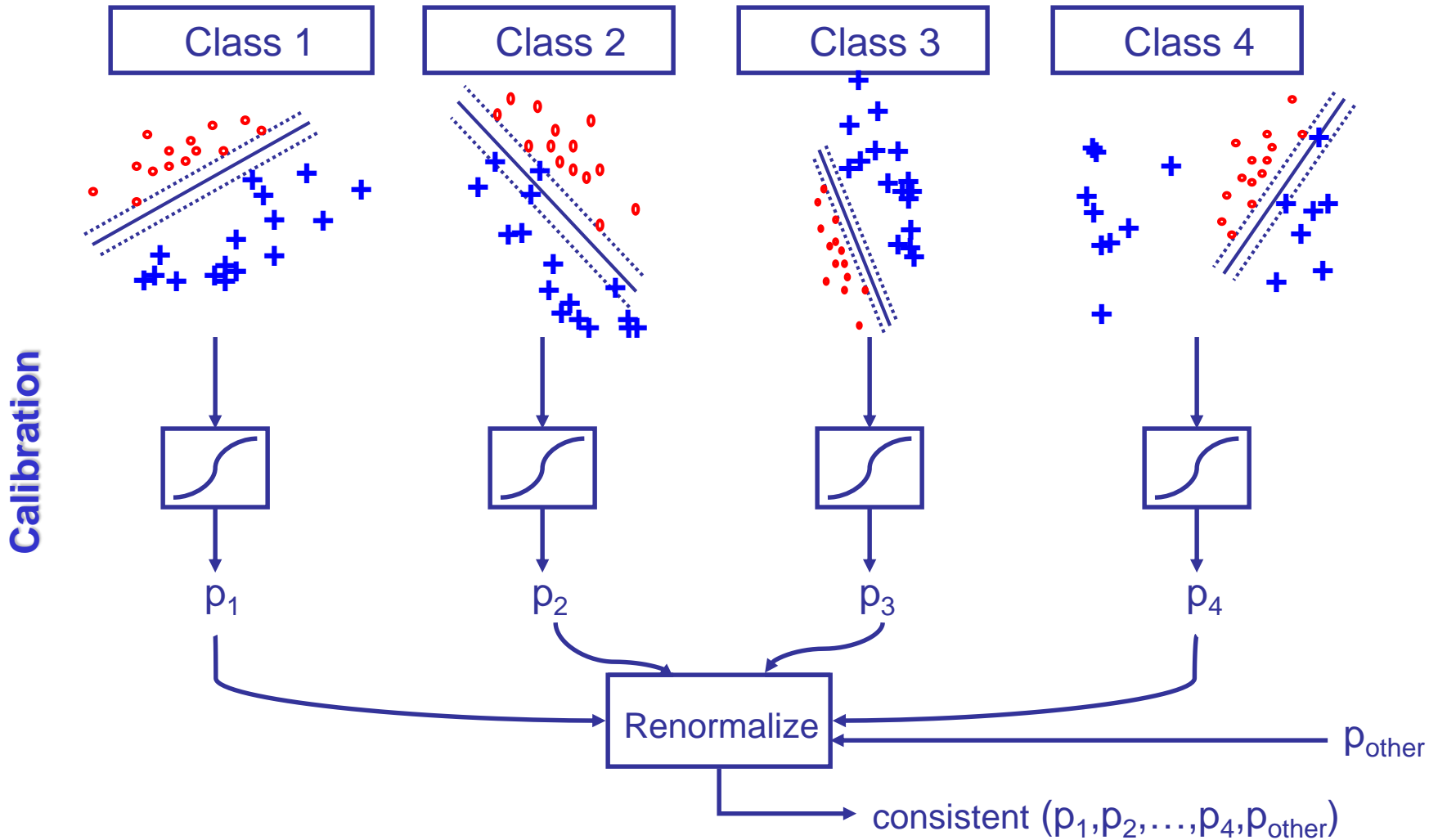
How to measure the confidence in a class prediction?

Crucial for:

1. Comparison between different classifiers
2. Ranking the prediction for ROC/Precision-Recall curve
3. In several application domains having a measure of confidence for each individual answer is very important (e.g. tumor detection)

Some methods have an implicit notion of confidence e.g. for SVM the distance to the class boundary relative to the size of the margin other like logistic regression have an explicit one.

Combining OVA calibrated classifiers



Exponential form

Once the graph is defined the model can be written in exponential form

$$p(x, y) = \frac{1}{Z} \exp \sum_{k, C} w_k^\top \phi_k(y_C, x_C)$$

$$p(x, y) = \frac{1}{Z} \exp \mathbf{w}^\top \Phi(y, x)$$

parameter vector

feature vector

Comparing two labellings with the likelihood ratio

$$\frac{p(x, \tilde{y})}{p(x, y)} = \frac{\exp \mathbf{w}^\top \Phi(\tilde{y}, x)}{\exp \mathbf{w}^\top \Phi(y, x)}$$

\tilde{y} wins over y when $\mathbf{w}^\top \Phi(\tilde{y}, x) > \mathbf{w}^\top \Phi(y, x)$

Discriminative Algorithms

Perceptron:
$$\max_{\mathbf{w}} \sum_i \left[\mathbf{w}^\top \Phi(x^i, y^i) - \max_y \mathbf{w}^\top \Phi(x^i, y) \right]$$

CRF:
(Conditional Random Field)
$$\max_{\mathbf{w}} \sum_i \left[\mathbf{w}^\top \Phi(x^i, y^i) - \operatorname{softmax}_y \mathbf{w}^\top \Phi(x^i, y) \right]$$

M³net:
$$\max_{\mathbf{w}} \sum_i \left[\mathbf{w}^\top \Phi(x^i, y^i) - \max_y (\ell(y, y^i) + \mathbf{w}^\top \Phi(x^i, y)) \right]$$

Example: multiclass setting

Predict: $\hat{y}_i = \arg \max_y w_y^\top x^i$

Update: if $\hat{y}_i \neq y^i$ then

$$w_{y^i, t+1} = w_{y^i, t} + \alpha x^i$$
$$w_{\hat{y}_i, t+1} = w_{\hat{y}_i, t} - \alpha x^i$$

Feature encoding:

$$\Phi(\mathbf{x}^i, y = 1)^\top = [\mathbf{x}^{i\top} \ 0 \ \dots \ 0]$$

$$\Phi(\mathbf{x}^i, y = 2)^\top = [0 \ \mathbf{x}^{i\top} \ \dots \ 0]$$

\vdots

$$\Phi(\mathbf{x}^i, y = K)^\top = [0 \ 0 \ \dots \ \mathbf{x}^{i\top}]$$

$$\mathbf{w}^\top = [w_1^\top \ w_2^\top \ \dots \ w_K^\top]$$

Predict: $\hat{y}_i = \arg \max_{y \in \mathcal{Y}} \mathbf{w}_t^\top \Phi(\mathbf{x}^i, y)$

Update: $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \underbrace{(\Phi(\mathbf{x}, y^i) - \Phi(\mathbf{x}^i, \hat{y}_i))}_{\text{update if } \hat{y}_i \neq y^i}$

Three Approaches to K-Class SVM

- **K one-versus-residue (OVR) binary SVM**
- $\frac{K(K-1)}{2}$ pairwise binary SVM
- **One K-Class SVM**

$$\max_W \left[\sum_i w_{y^i}^\top x^i - \max_j (1\{j \neq y^i\} + w_j^\top x^i) \right]$$

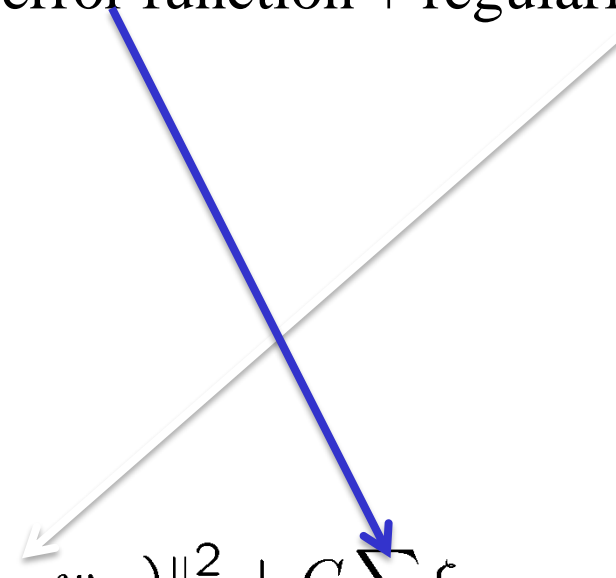


$$\min_{w_1, \dots, w_K} \frac{1}{2} \|(w_1, \dots, w_K)\|^2 + C \sum_{ik} \xi_{ik}$$

$$\text{s.t.} \quad \forall (i, k), \quad w_{y^i}^\top x^i - w_k^\top x^i \geq 1\{k \neq y^i\} - \xi_{ik}$$

General SVM

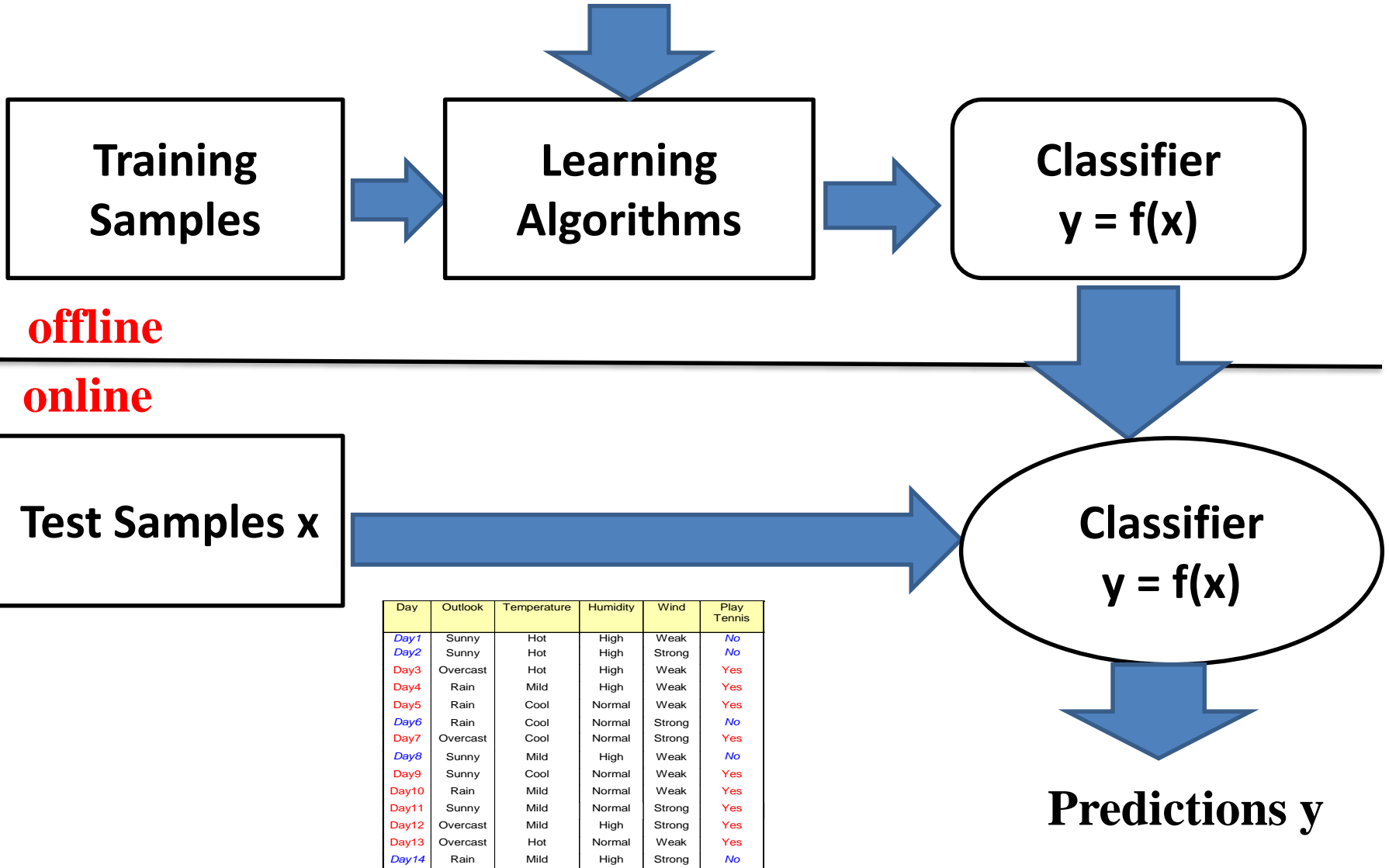
Objective function = error function + regularization term



The diagram shows two arrows originating from the text 'Objective function = error function + regularization term'. A blue arrow points from 'error function' to the term $C \sum_{ik} \xi_{ik}$ in the objective function. A white arrow points from 'regularization term' to the term $\frac{1}{2} \|(w_1, \dots, w_K)\|^2$ in the objective function.

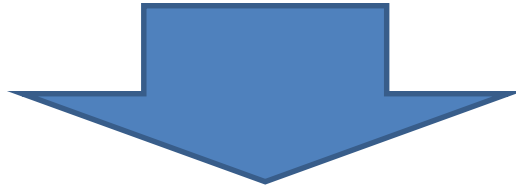
$$\begin{aligned} \min_{w_1, \dots, w_K} \quad & \frac{1}{2} \|(w_1, \dots, w_K)\|^2 + C \sum_{ik} \xi_{ik} \\ \text{s.t.} \quad & \forall (i, k), \quad w_{y^i}^\top x^i - w_k^\top x^i \geq \mathbf{1}\{k \neq y^i\} - \xi_{ik} \end{aligned}$$

SVM & GMM (Bayes Rule) cannot handle this case!



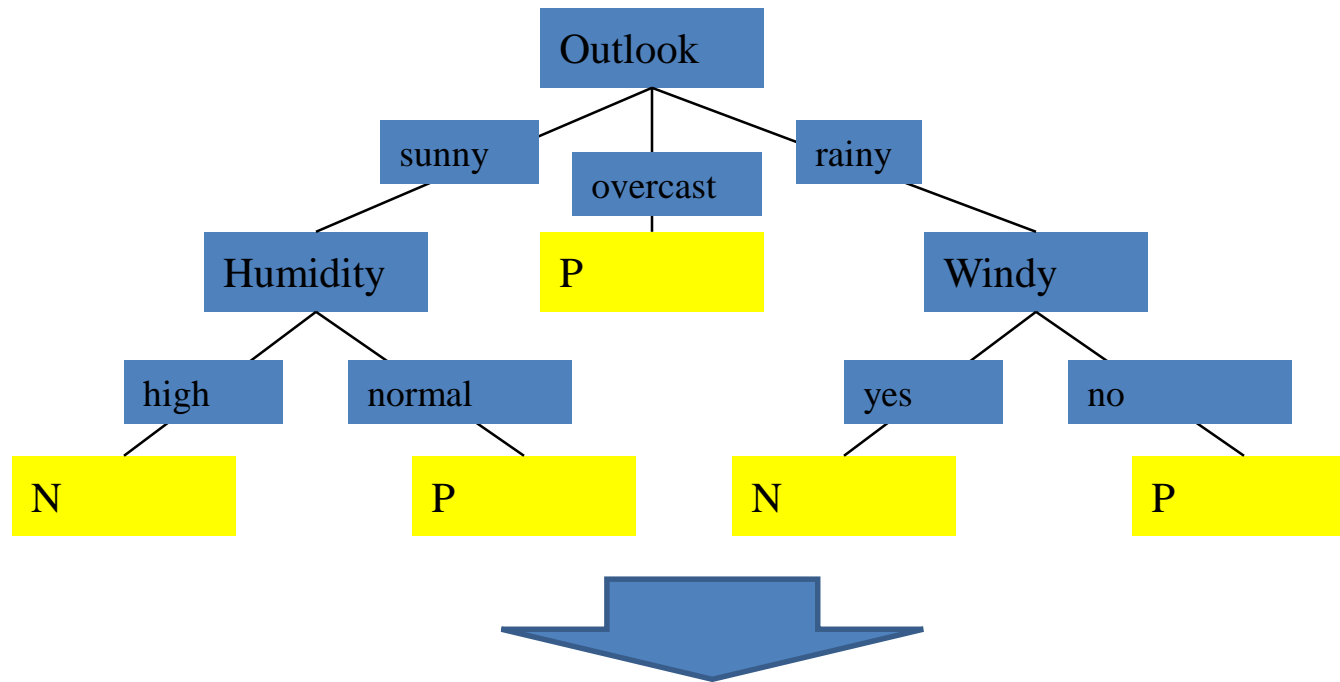
Wish List:

- Even these features or attributes x are not comparable directly, the decisions y they make could be comparable!
- For different decisions y , we can use different features or attributes x !



Classifier Training with Feature Selection!

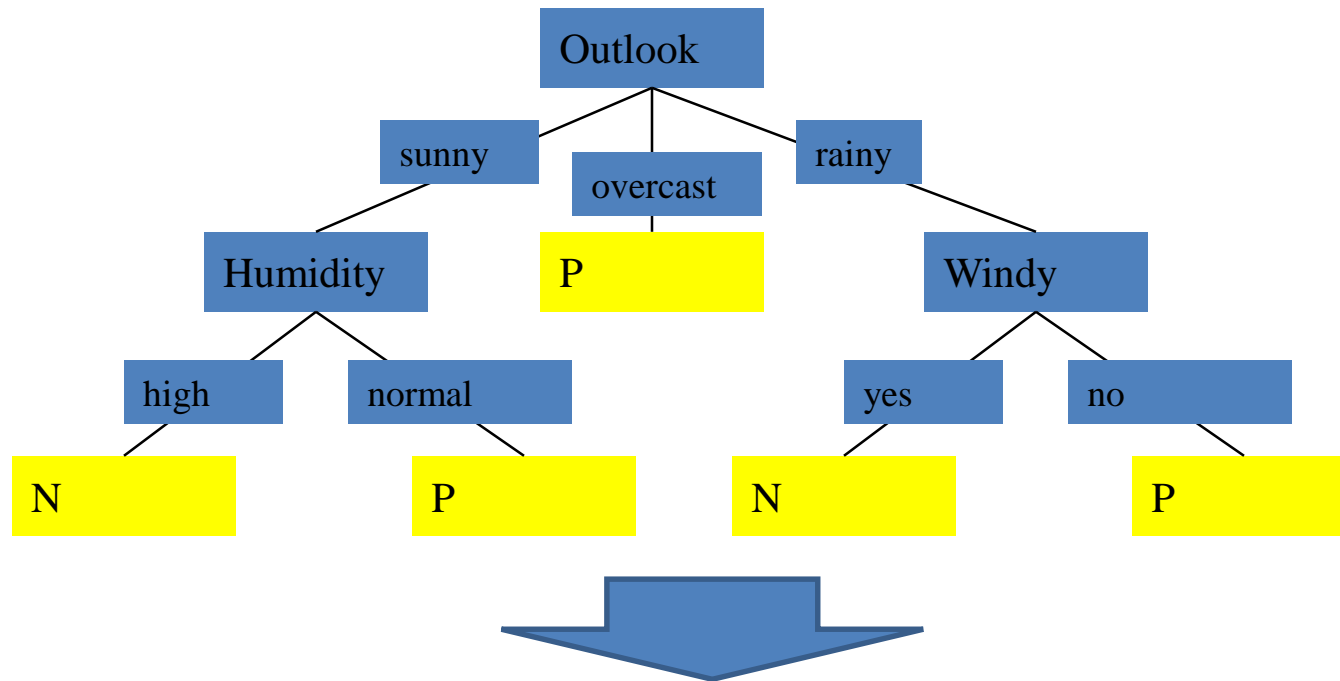
Our Expectations



How to make this happen?

Compare it with our wish list

Our Expectations



- Different nodes can select and use one single feature or attribute x
- Use various paths to combine multiple features or attributes x

How to select feature for each node? When we stop such path?

Decision Tree

- What is a Decision Tree
- Sample Decision Trees
- How to Construct a Decision Tree
- Problems with Decision Trees
- Summary

Definition

- Decision tree is a classifier in the form of a tree structure
 - **Decision node**: specifies a test on a **single attribute** or **feature x**
 - **Leaf node**: indicates the value of the **target attribute (label) y**
 - **Arc/edge**: split of one attribute (**could be multiple partitions y or binary ones y**)
 - **Path**: a disjunction of test to make the final decision (**all attributes x could be used**)
- Decision trees classify instances or examples by starting at the root of the tree and moving through it until a leaf node.

Why decision tree?

- Decision trees are powerful and popular tools for classification and prediction.
- Decision trees represent *rules*, which can be understood by humans and used in knowledge system such as database.

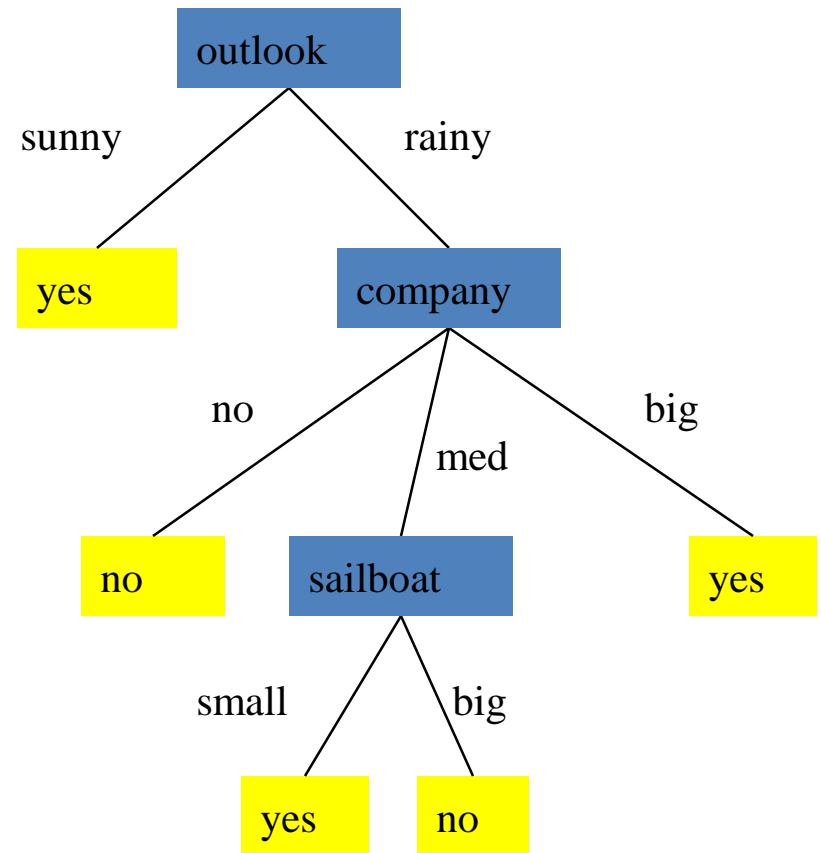
Compare these with our wish list

key requirements

- **Attribute-value description:** object or case must be expressible in terms of a fixed collection of properties or attributes x (e.g., hot, mild, cold).
- **Predefined classes (target values y):** the target function has **discrete output values y** (binary or multiclass)
- **Sufficient data:** enough training cases should be provided to learn the model.

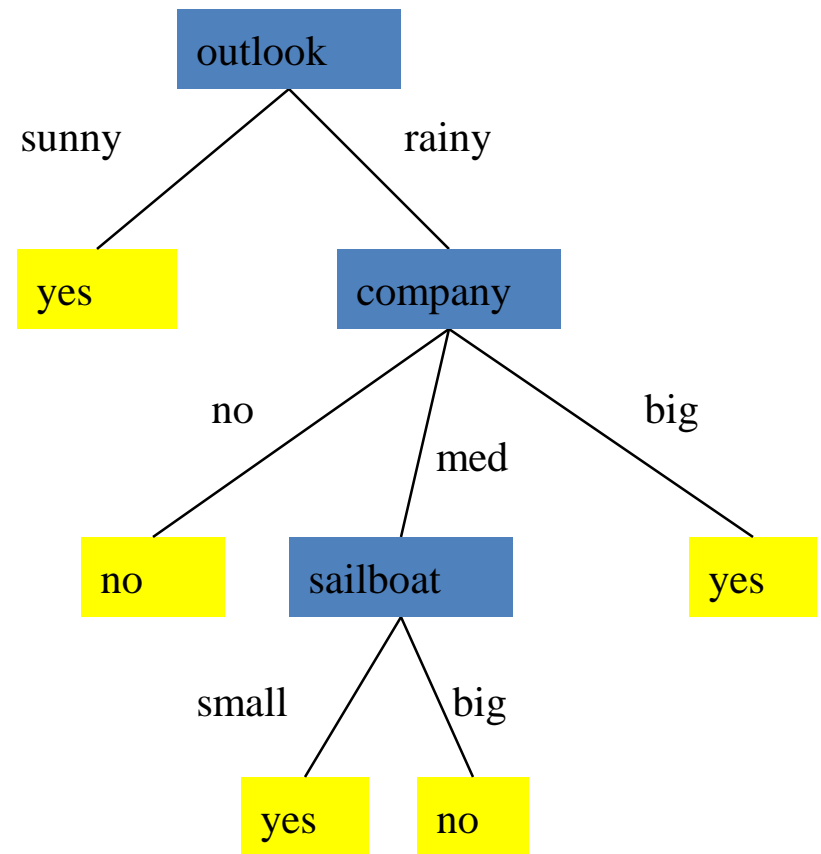
An Example Data Set and Decision Tree

#	Attribute			Class Sail?
	Outlook	Company	Sailboat	
1	sunny	big	small	yes
2	sunny	med	small	yes
3	sunny	med	big	yes
4	sunny	no	small	yes
5	sunny	big	big	yes
6	rainy	no	small	no
7	rainy	med	small	yes
8	rainy	big	big	yes
9	rainy	no	big	no
10	rainy	med	big	no



Classification

#	Attribute			Class
	Outlook	Company	Sailboat	Sail?
1	sunny	no	big	?
2	rainy	big	small	?

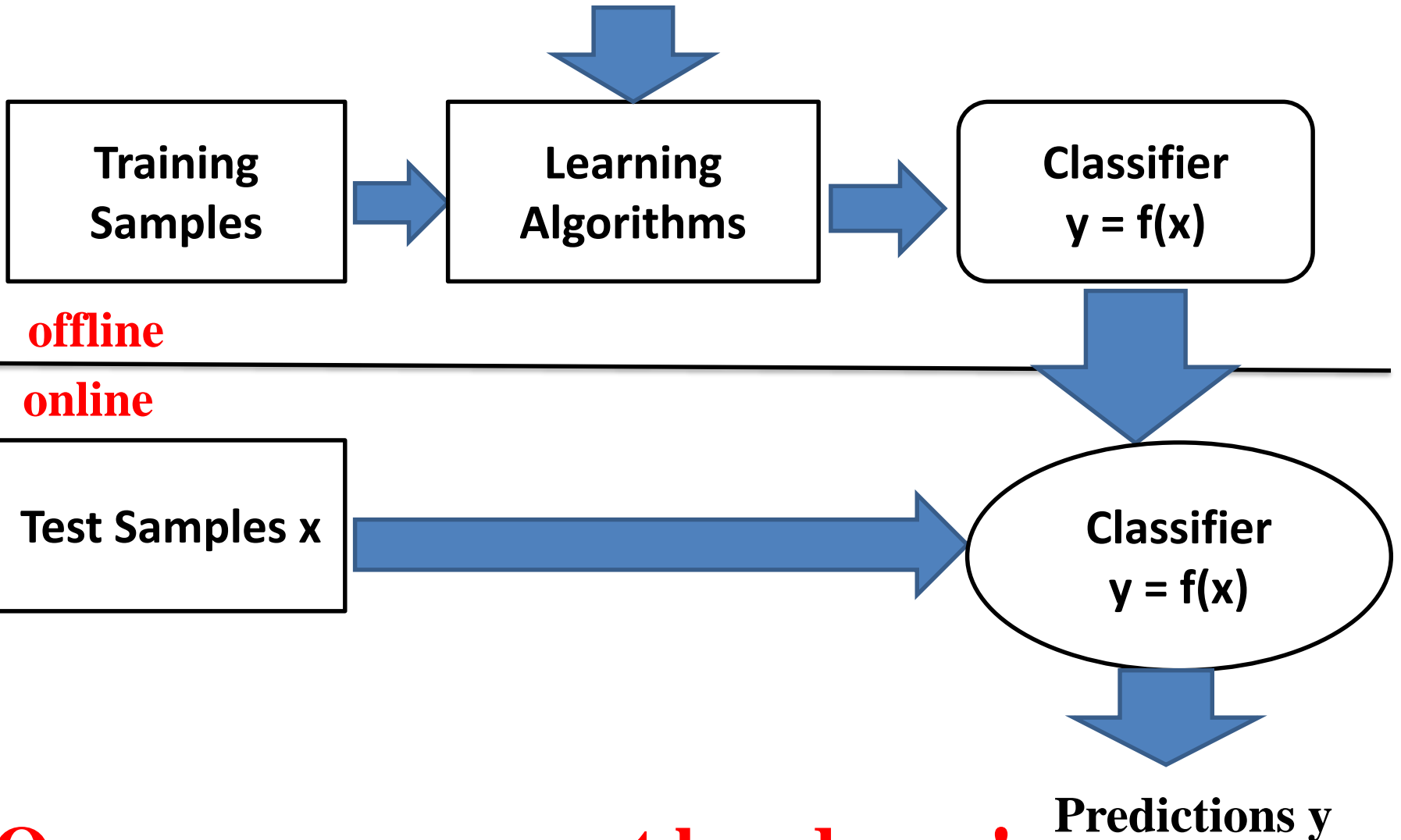


DECISION TREE

- **An internal node is a test on an attribute.**
- A **branch** represents an outcome of the test, e.g., Color=red.
- A **leaf node** represents a class label or class label distribution.
- **At each node**, one attribute is chosen to split training examples into distinct classes as much as possible
- A **new case** is classified by following a matching path to a leaf node.

Each node uses one single feature to train one classifier!

SVM, GMM (Bayes Rule) & Decision Tree



One more weapon at hand now!

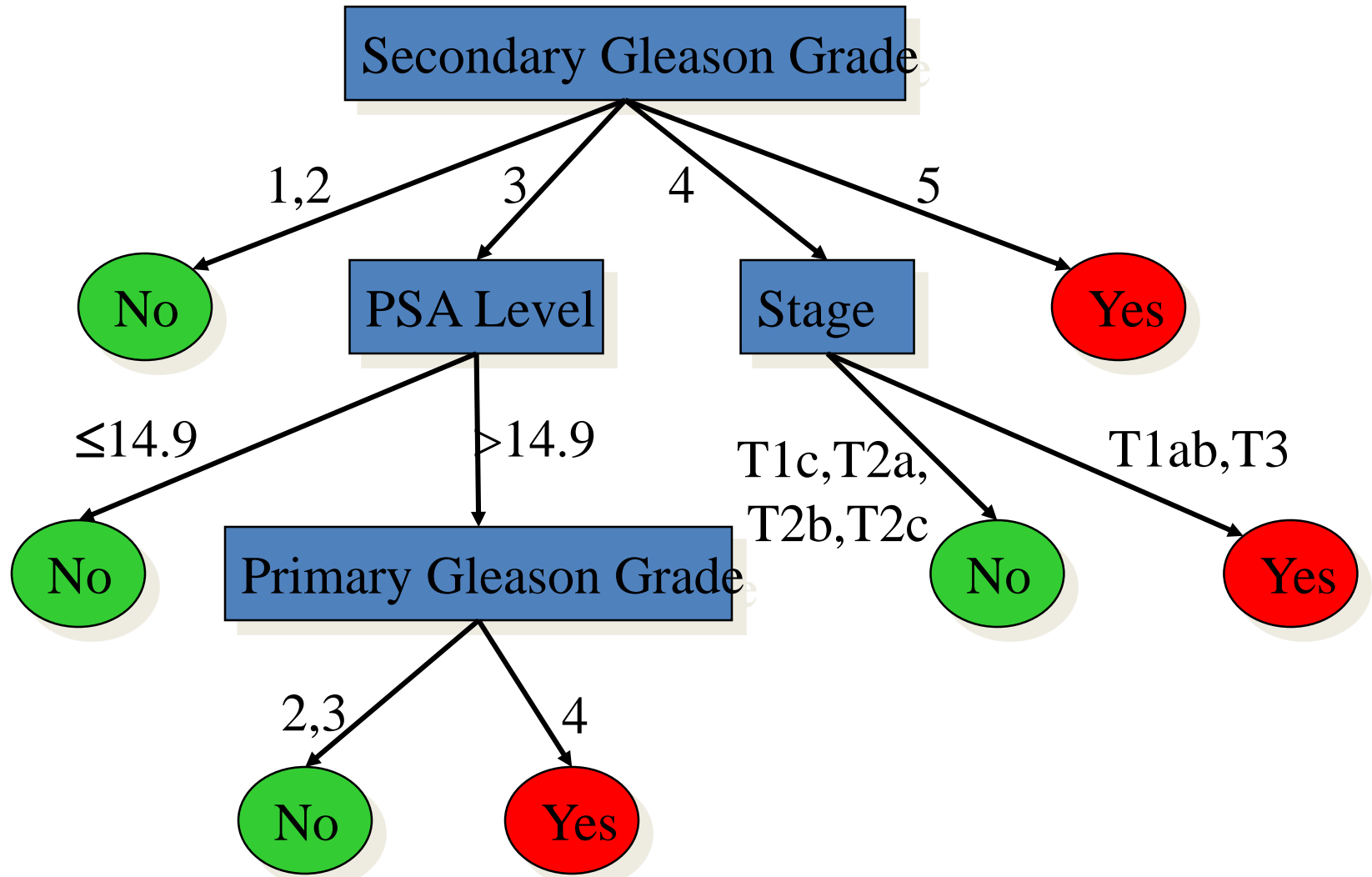
Decision Tree Construction

- Top-Down Decision Tree Construction
- Choosing the Splitting Attribute: **Feature Selection**
- Information Gain and Gain Ratio: **Classifier Training**

Decision Tree Construction

- Selecting the best-matching **feature** or **attribute x** for each node
 - what kind of criteria can be used?
- Training the node **classifier** $y = f(x)$ under the selected feature x
 - what kind of classifiers can be used?

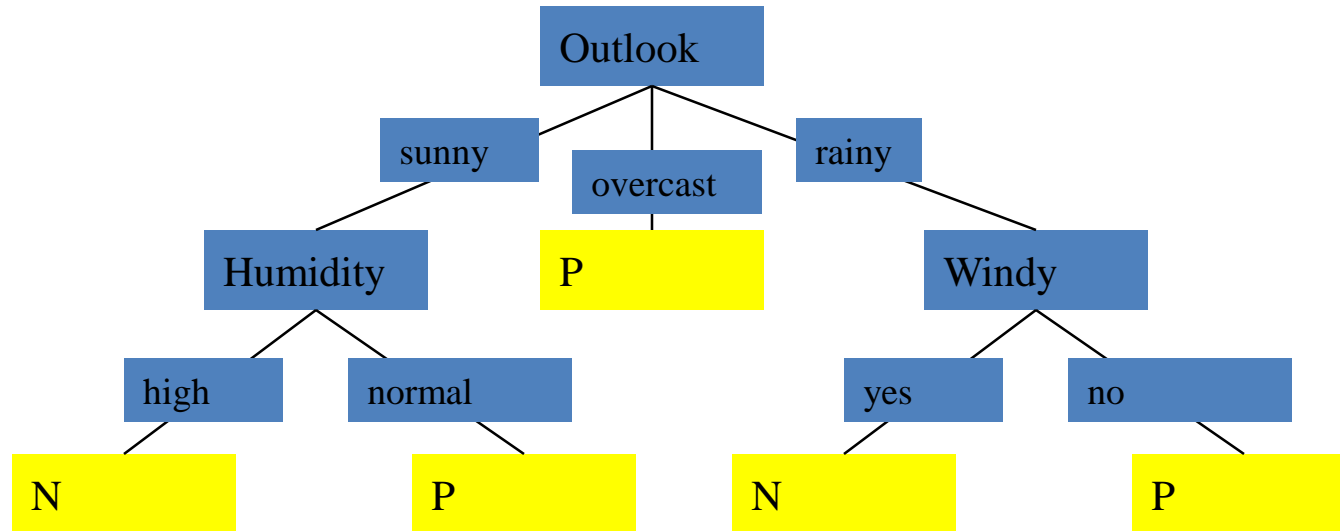
Prostate cancer recurrence



Another Example

#	Attribute				Class
	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	no	N
2	sunny	hot	high	yes	N
3	overcast	hot	high	no	P
4	rainy	moderate	high	no	P
5	rainy	cold	normal	no	P
6	rainy	cold	normal	yes	N
7	overcast	cold	normal	yes	P
8	sunny	moderate	high	no	N
9	sunny	cold	normal	no	P
10	rainy	moderate	normal	no	P
11	sunny	moderate	normal	yes	P
12	overcast	moderate	high	yes	P
13	overcast	hot	normal	no	P
14	rainy	moderate	high	yes	N

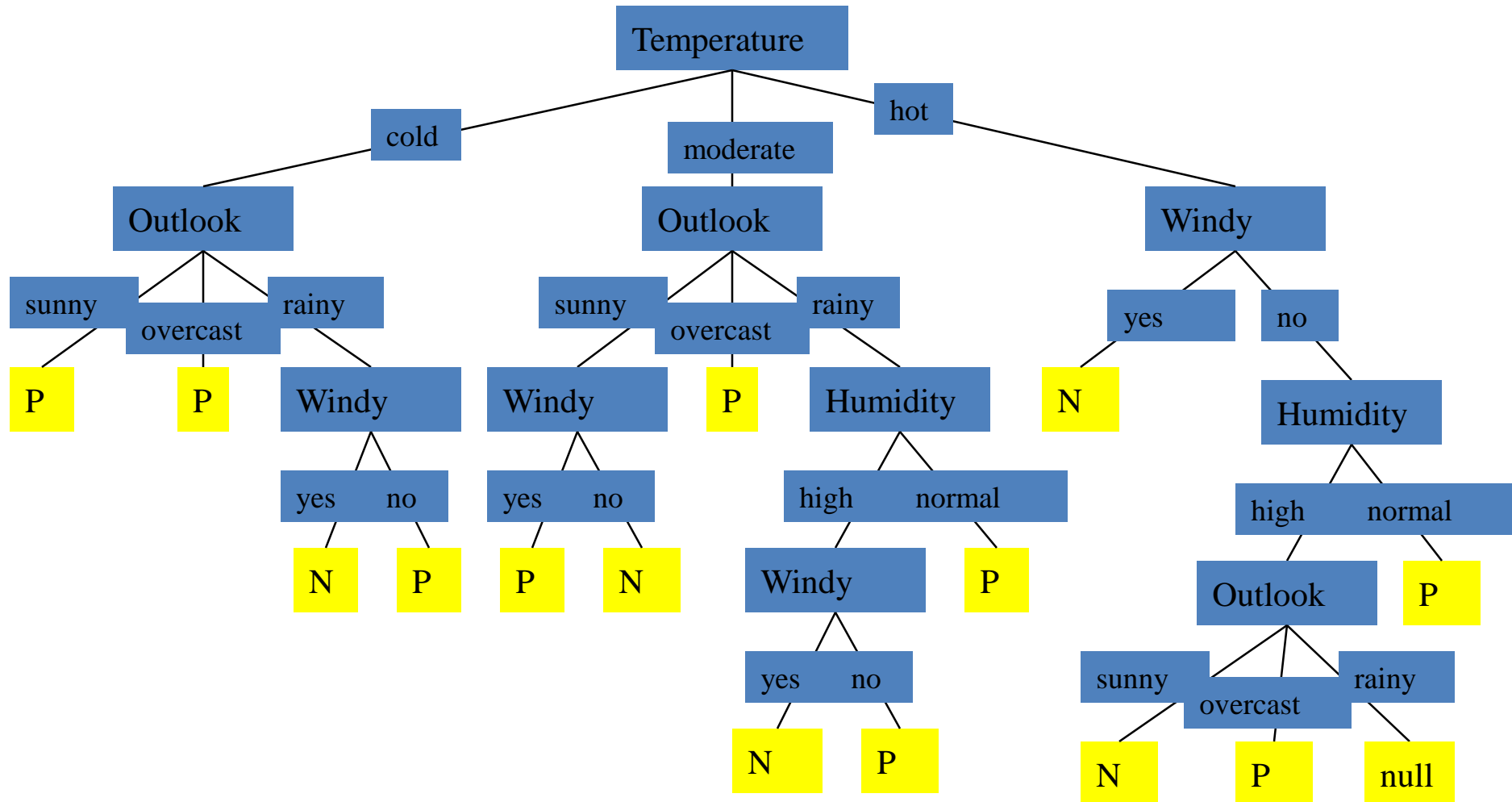
Simple Tree



Decision Tree can select different attributes for different decisions!

Complicated Tree

Given a data set, we could have multiple solutions!



Attribute Selection Criteria

- Main principle
 - Select attribute which partitions the learning set into subsets **as “pure” as possible**
- Various measures of **purity**
 - Information-theoretic
 - Gini index
 - χ^2
 - ReliefF
 - ...
- Various improvements
 - probability estimates
 - normalization
 - binarization, subsetting

Information-Theoretic Approach

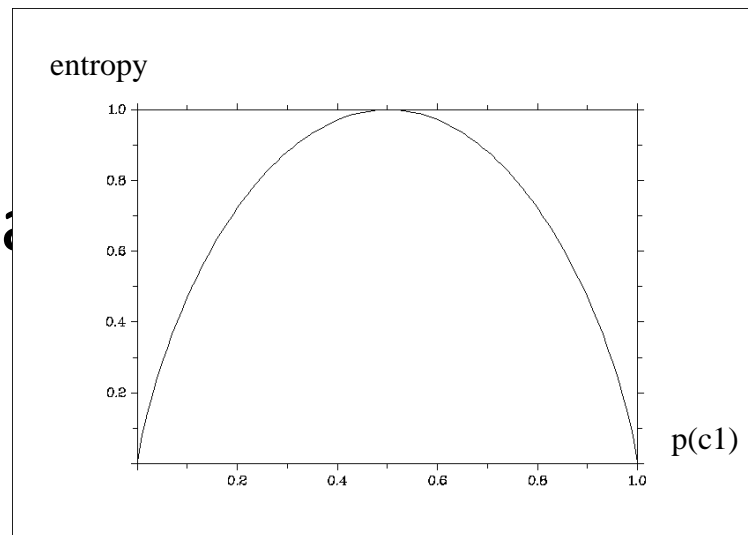
- To classify an object, a certain information is needed
 - I , information
- After we have learned the value of attribute A , we only need some remaining amount of information to classify the object
 - I_{res} , residual information
- Gain
 - $\text{Gain}(A) = I - I_{res}(A)$
- The most 'informative' attribute is the one that minimizes I_{res} , *i.e.*, maximizes Gain

Entropy

- The average amount of information / needed to classify an object is given by

the entropy
$$I = - \sum_c p(c) \log_2 p(c)$$

- For a two-class



Residual Information

- After applying attribute A , S is partitioned into subsets according to values v of A
- I_{res} is equal to weighted sum of the amounts of information for the subsets

$$I_{res} = - \sum_v p(v) \sum_c p(c|v) \log_2 p(c|v)$$

Triangles and Squares

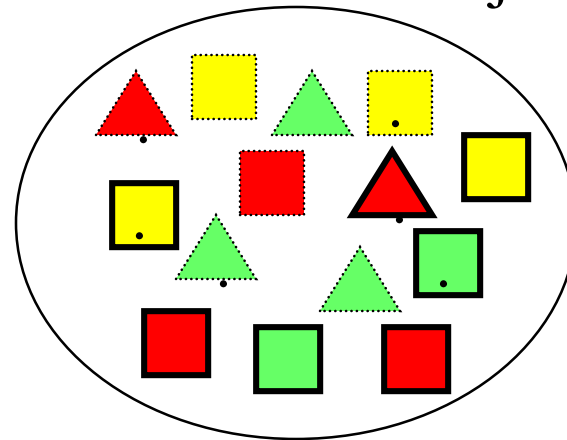
#	Attribute			Shape
	Color	Outline	Dot	
1	green	dashed	no	triange
2	green	dashed	yes	triange
3	yellow	dashed	no	square
4	red	dashed	no	square
5	red	solid	no	square
6	red	solid	yes	triange
7	green	solid	no	square
8	green	dashed	no	triange
9	yellow	solid	yes	square
10	red	solid	no	square
11	green	solid	yes	square
12	yellow	dashed	yes	square
13	yellow	solid	no	square
14	red	dashed	yes	triange

Triangles and Squares

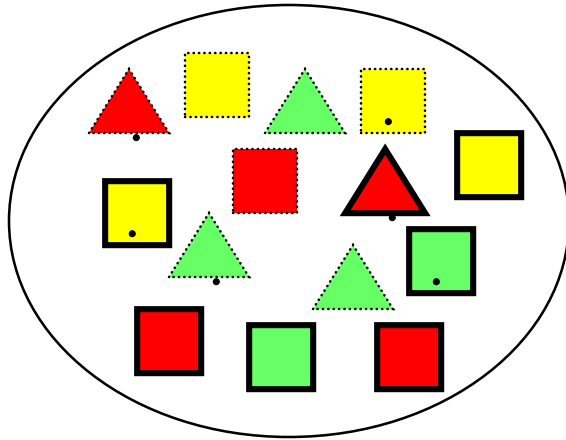
#	Attribute			Shape
	Color	Outline	Dot	
1	green	dashed	no	triange
2	green	dashed	yes	triange
3	yellow	dashed	no	square
4	red	dashed	no	square
5	red	solid	no	square
6	red	solid	yes	triange
7	green	solid	no	square
8	green	dashed	no	triange
9	yellow	solid	yes	square
10	red	solid	no	square
11	green	solid	yes	square
12	yellow	dashed	yes	square
13	yellow	solid	no	square
14	red	dashed	yes	triange

Data Set:

A set of classified objects



Entropy



- 5 triangles
- 9 squares
- class probabilities

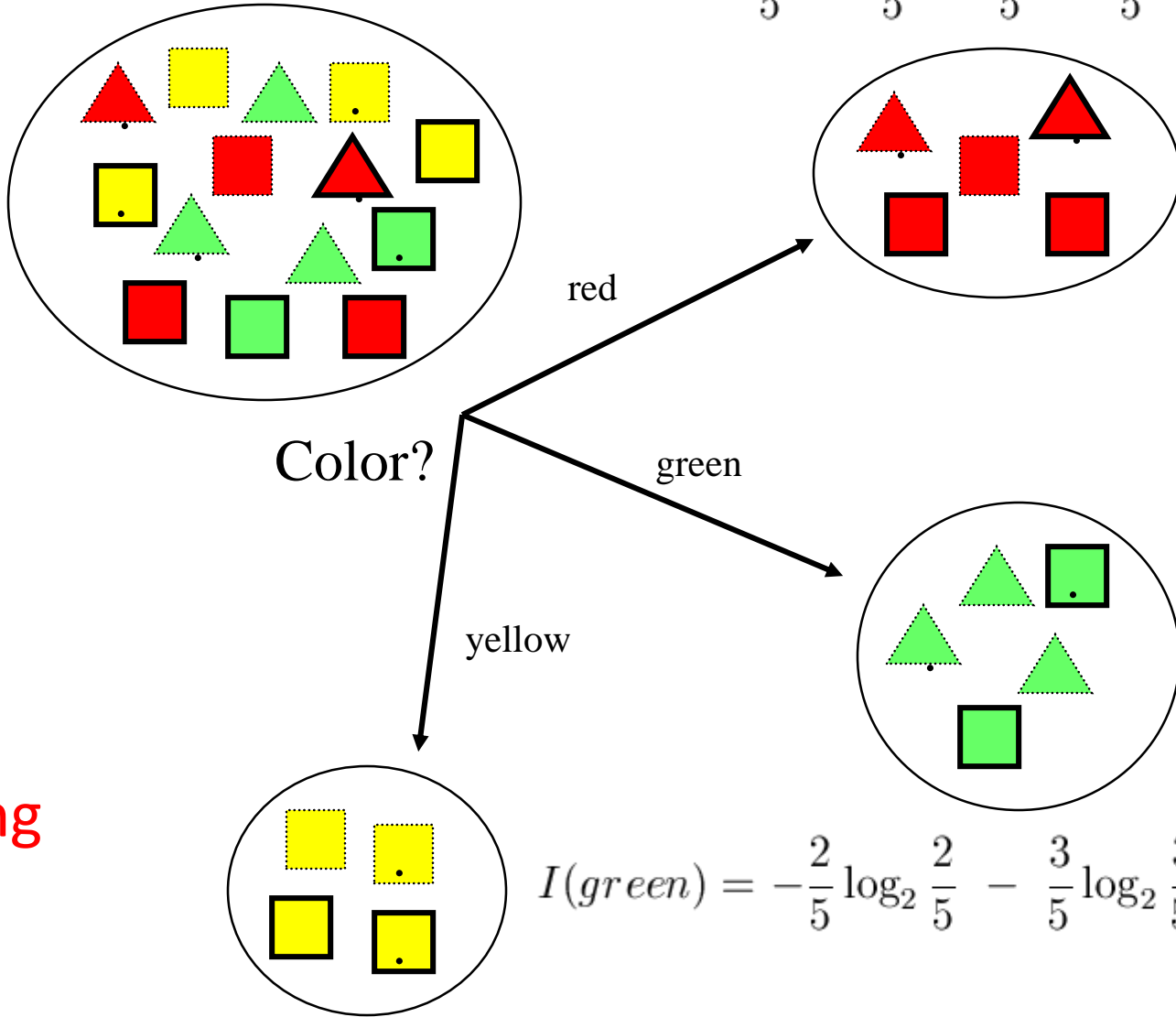
$$p(\square) = \frac{9}{14}$$

$$p(\triangle) = \frac{5}{14}$$

- entropy

$$I = -\frac{9}{14} \log_2 \frac{9}{14} - \frac{5}{14} \log_2 \frac{5}{14} = 0.940 \text{ bits}$$

$$I(\text{red}) = -\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5} = 0.971 \text{ bits}$$

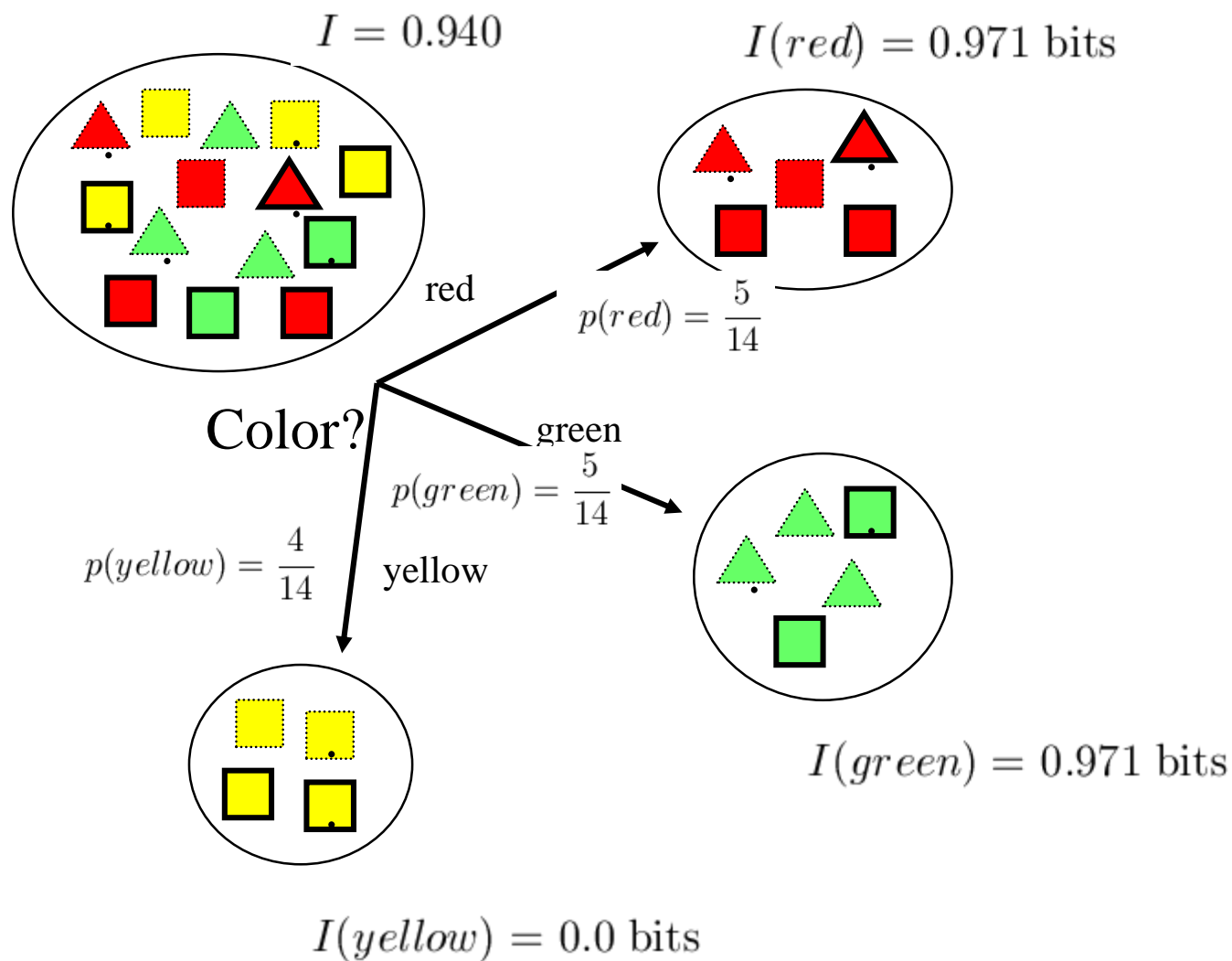


$$I(\text{green}) = -\frac{2}{5} \log_2 \frac{2}{5} - \frac{3}{5} \log_2 \frac{3}{5} = 0.971 \text{ bits}$$

$$I(\text{yellow}) = -\frac{4}{4} \log_2 \frac{4}{4} - \frac{0}{4} \log_2 \frac{0}{4} = 0.0 \text{ bits}$$

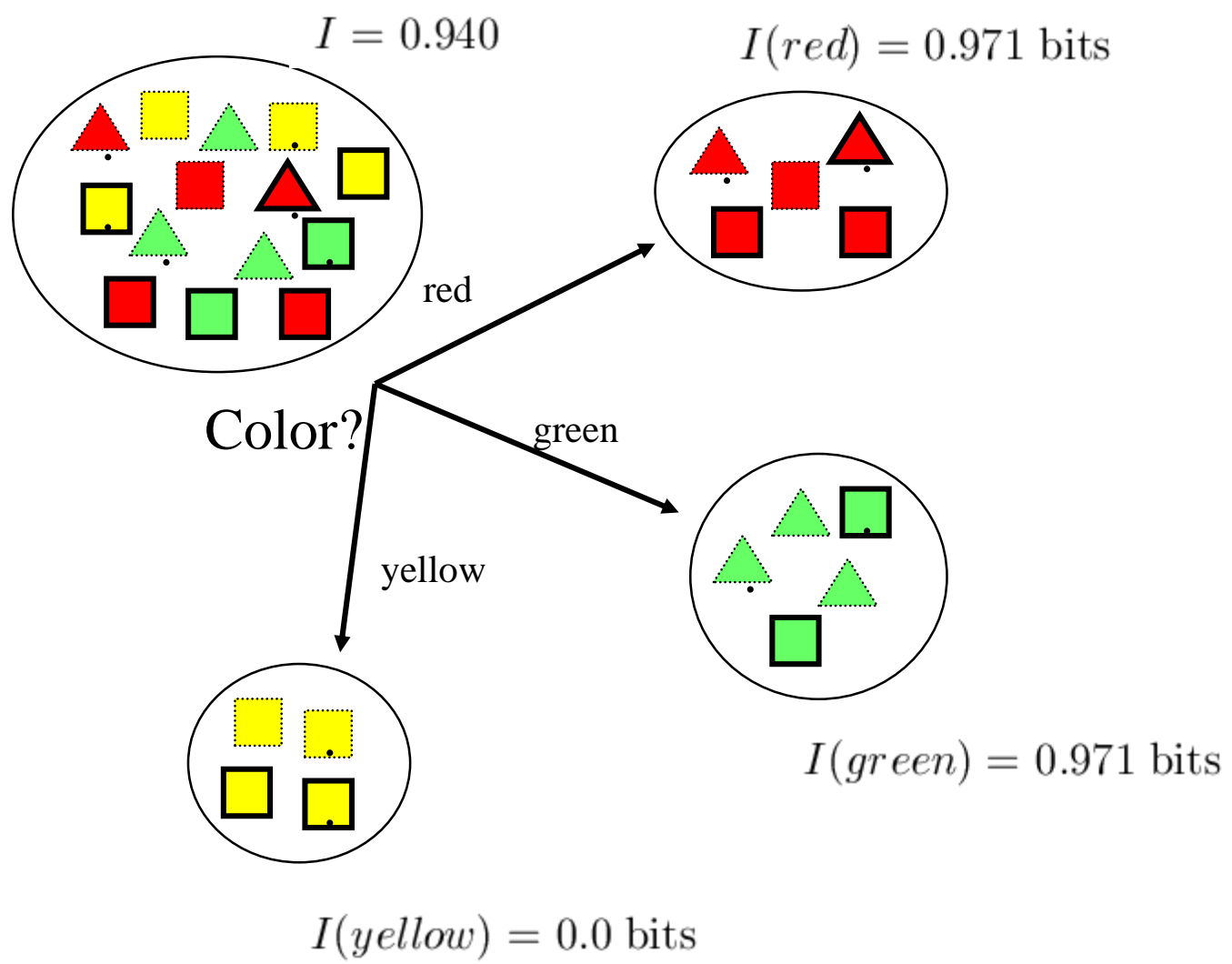
Entropy
reduction
by
data set
partitioning

Entropija vrednosti atributa



$$I_{res}(\text{Color}) = \sum p(v)I(v) = \frac{5}{14}0.971 + \frac{5}{14}0.971 + \frac{4}{14}0.0 = 0.694 \text{ bits}$$

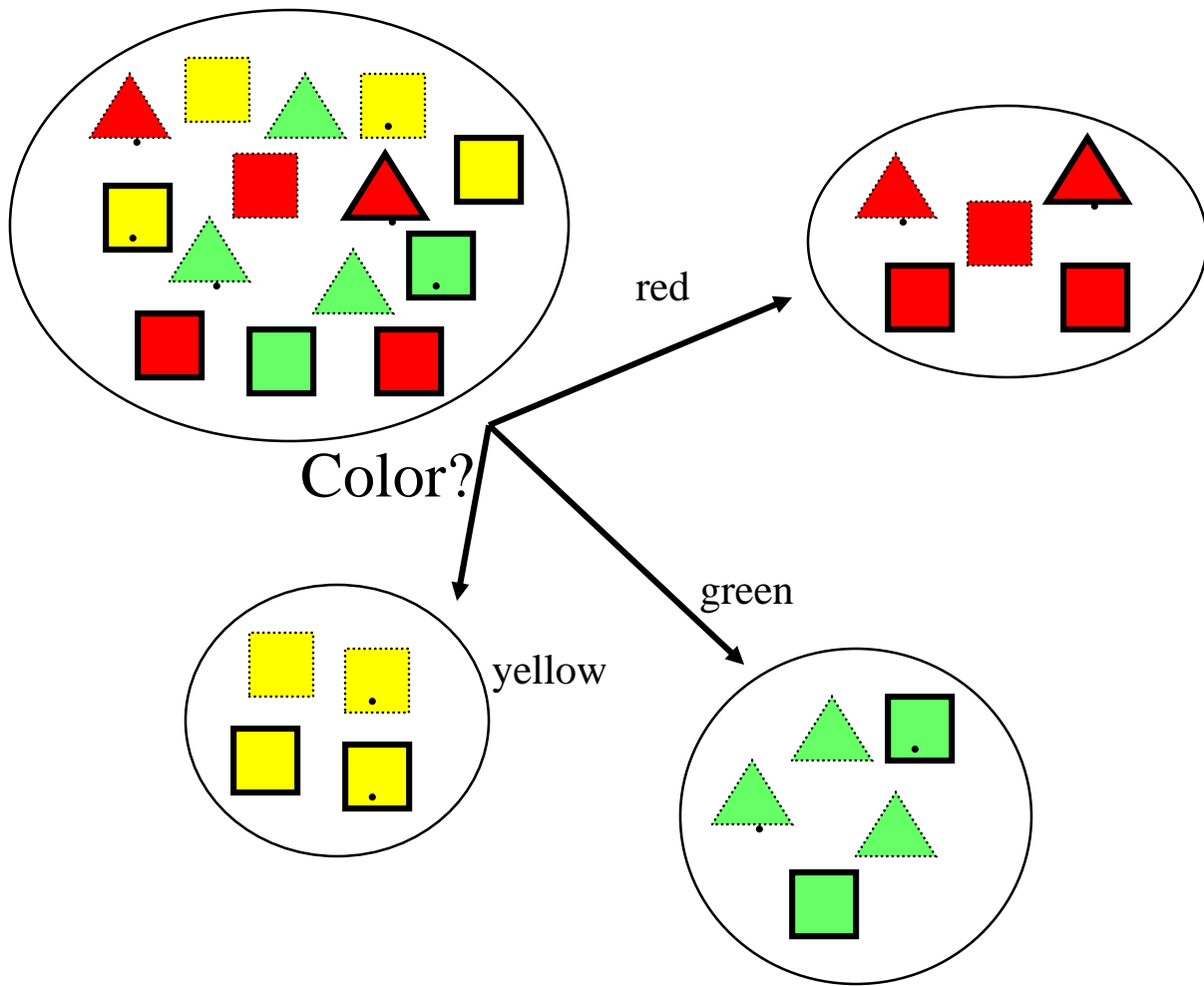
Information Gain



$Gain(\text{Color}) = I - I_{res}(\text{Color}) = 0.940 - 0.694 = 0.246 \text{ bits}$

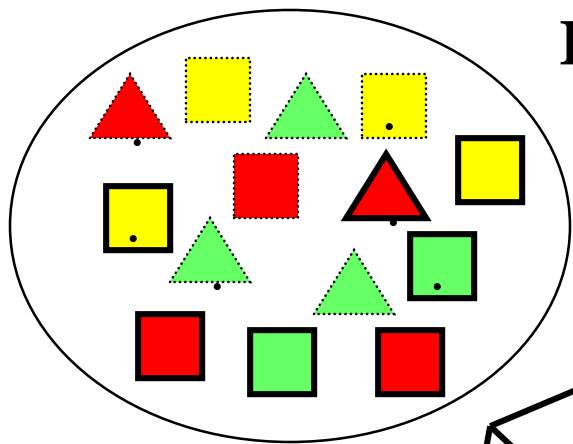
Information Gain of The Attribute

- **Attributes**
 - $\text{Gain}(\text{Color}) = 0.246$
 - $\text{Gain}(\text{Outline}) = 0.151$
 - $\text{Gain}(\text{Dot}) = 0.048$
- Heuristics: **attribute with the highest gain is chosen**
- This heuristics is local (local minimization of impurity)



$$\text{Gain(Outline)} = 0.971 - 0 = 0.971 \text{ bits}$$

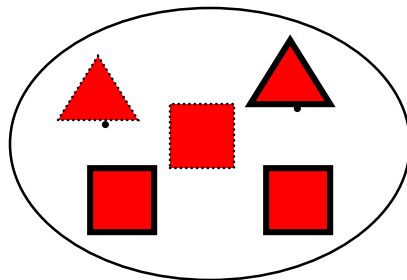
$$\text{Gain(Dot)} = 0.971 - 0.951 = 0.020 \text{ bits}$$



Initial decision

Color?

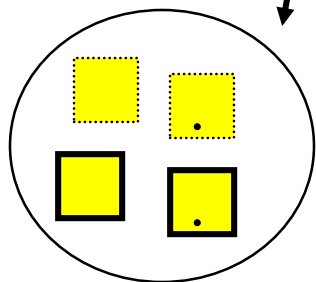
red



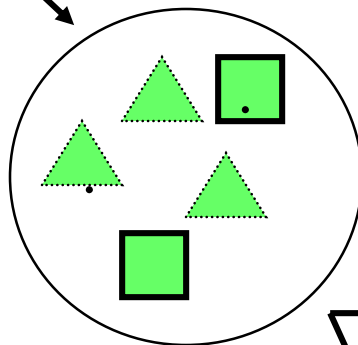
$$\text{Gain(Outline)} = 0.971 - 0.951 = 0.020 \text{ bits}$$

$$\text{Gain(Dot)} = 0.971 - 0 = 0.971 \text{ bits}$$

green

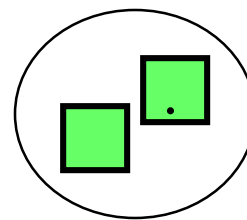


yellow

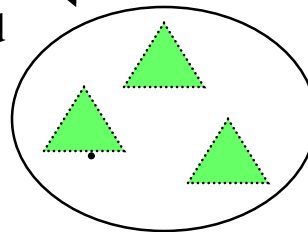


Outline?

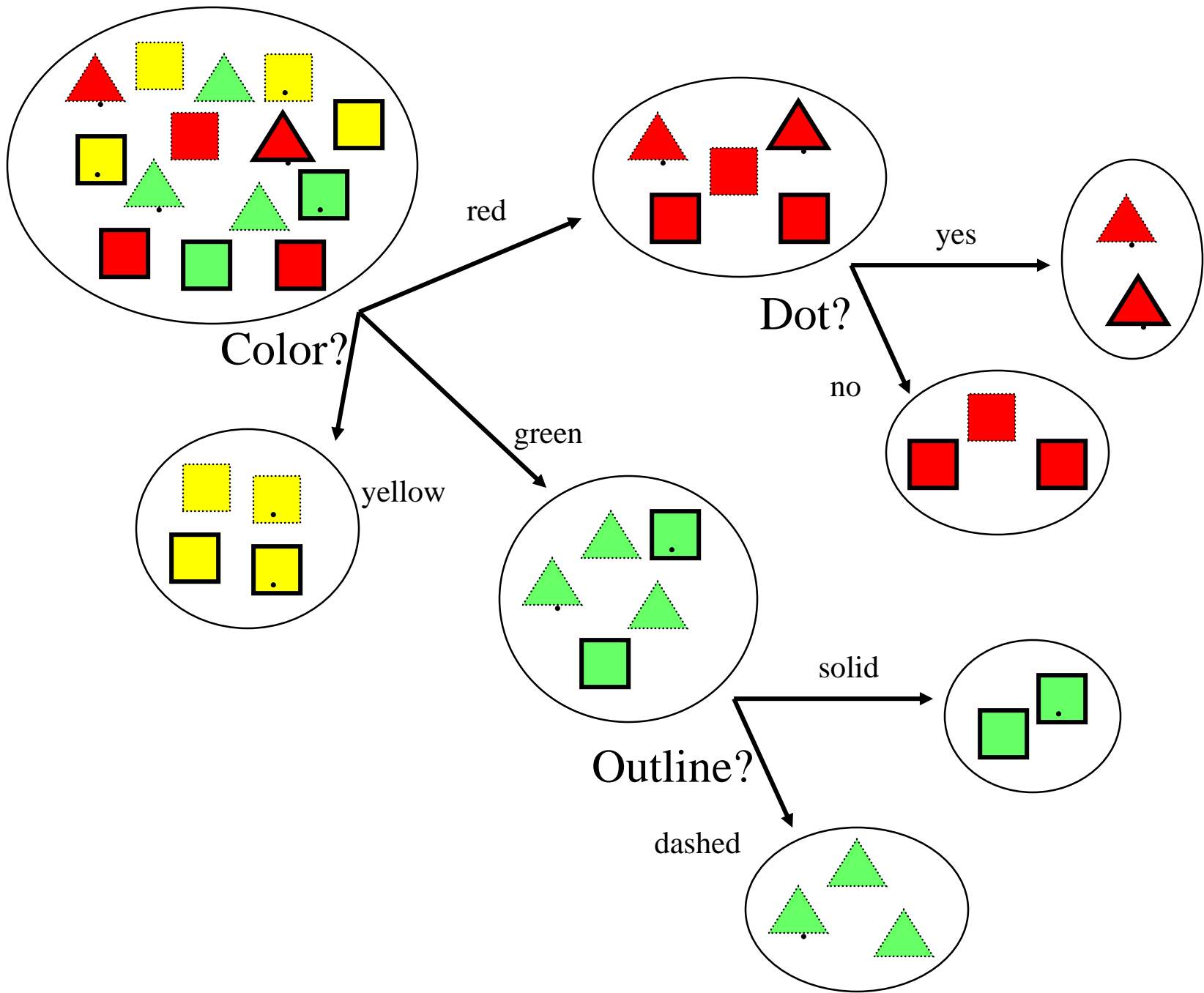
solid



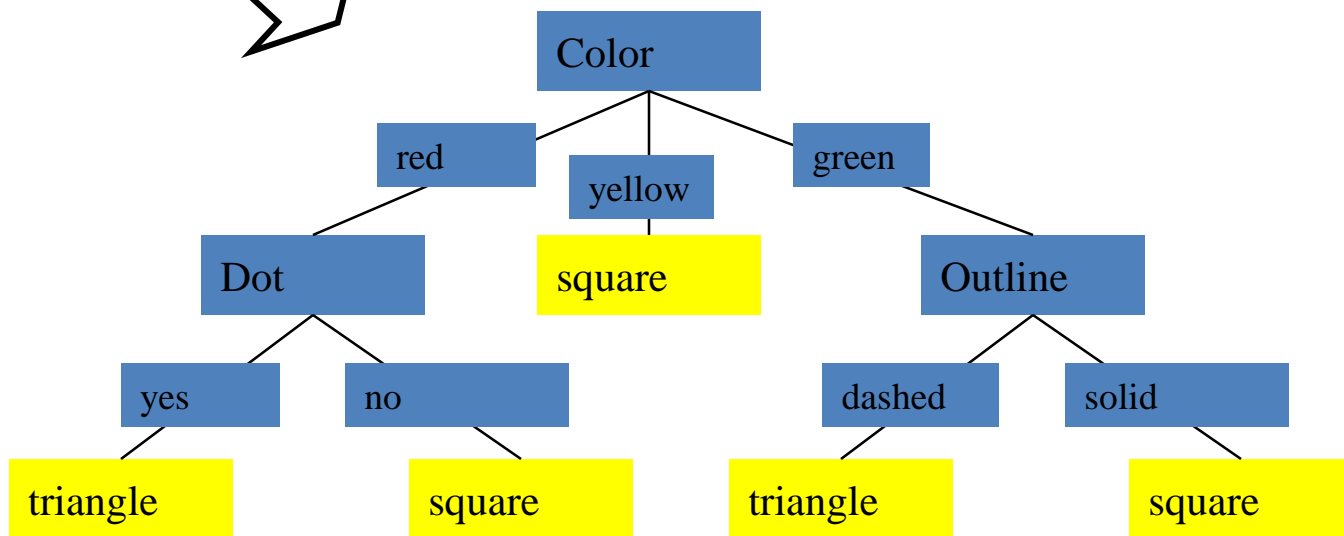
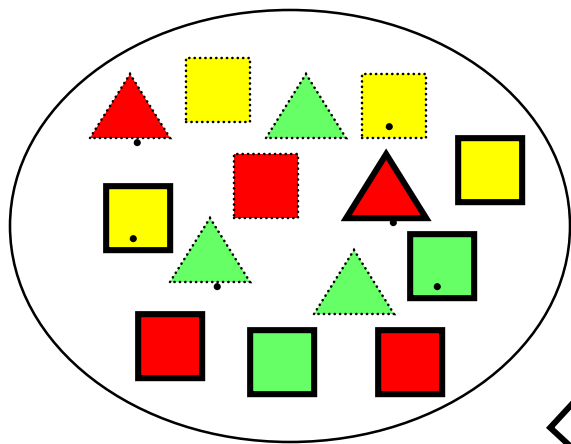
dashed



Conditional decision



Decision Tree



A Defect of *Ires*

- *Ires* favors attributes with many values
- Such attribute splits S to many subsets, and if these are small, they will tend to be pure anyway
- One way to rectify this is through a corrected measure of **information gain ratio**.

Information Gain Ratio

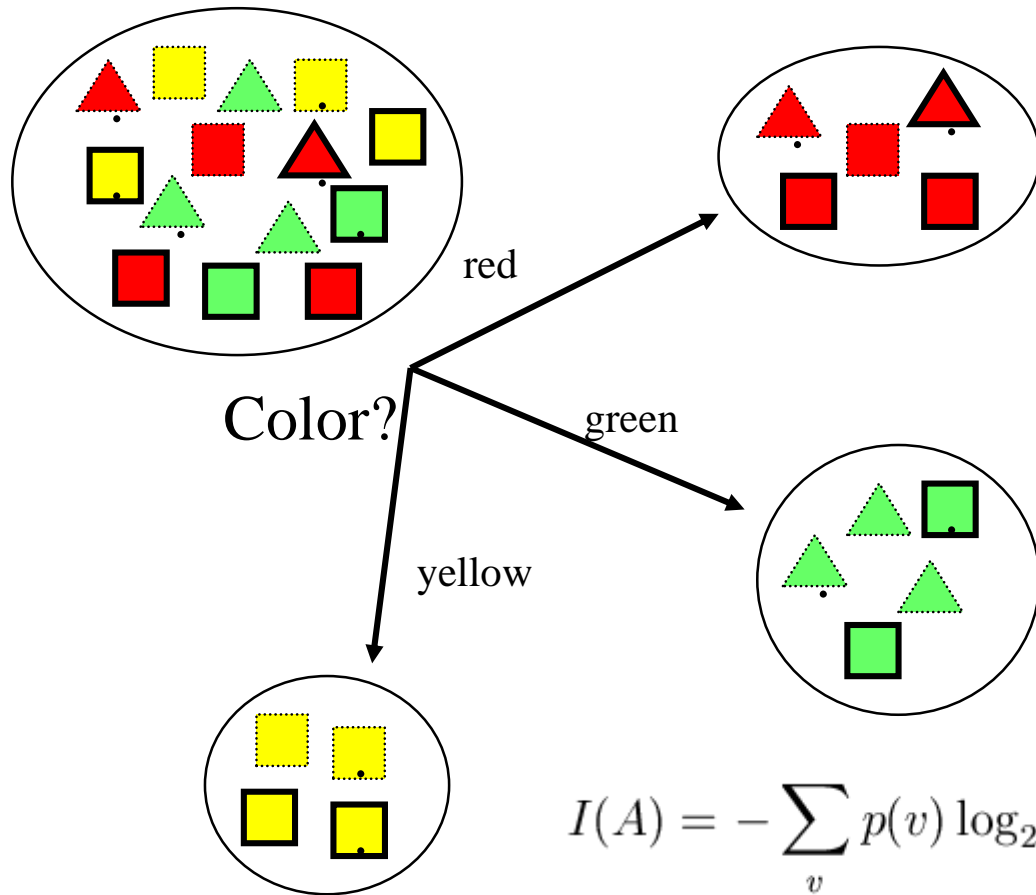
- $I(A)$ is amount of information needed to **determine the value of an attribute A**

$$I(A) = - \sum_v p(v) \log_2(p(v))$$

- Information gain ratio

$$\text{GainRatio}(A) = \frac{\text{Gain}(A)}{I(A)} = \frac{I - I_{res}(A)}{I(A)}$$

Information Gain Ratio



$$I(A) = - \sum_v p(v) \log_2(p(v))$$

$$I(\text{Color}) = -\frac{5}{14} \log_2 \frac{5}{14} - \frac{5}{14} \log_2 \frac{5}{14} - \frac{4}{14} \log_2 \frac{4}{14} = 1.58 \text{ bits}$$

$$\text{GainRatio}(\text{Color}) = \frac{\text{Gain}(\text{Color})}{I(\text{Color})} = \frac{0.940 - 0.694}{1.58} = 0.156$$

Information Gain and Information Gain Ratio

<i>A</i>	$ v(A) $	<i>Gain(A)</i>	<i>GainRatio(A)</i>
Color	3	0.247	0.156
Outline	2	0.152	0.152
Dot	2	0.048	0.049

Gini Index

- Another sensible measure of impurity (i and j are classes)

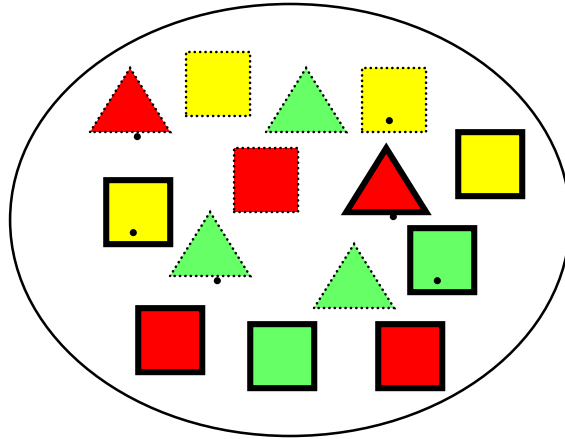
$$Gini = \sum_{i \neq j} p(i)p(j)$$

- After applying attribute A, the resulting Gini index is

$$Gini(A) = \sum_v p(v) \sum_{i \neq j} p(i|v)p(j|v)$$

- Gini can be interpreted as expected error rate

Gini Index



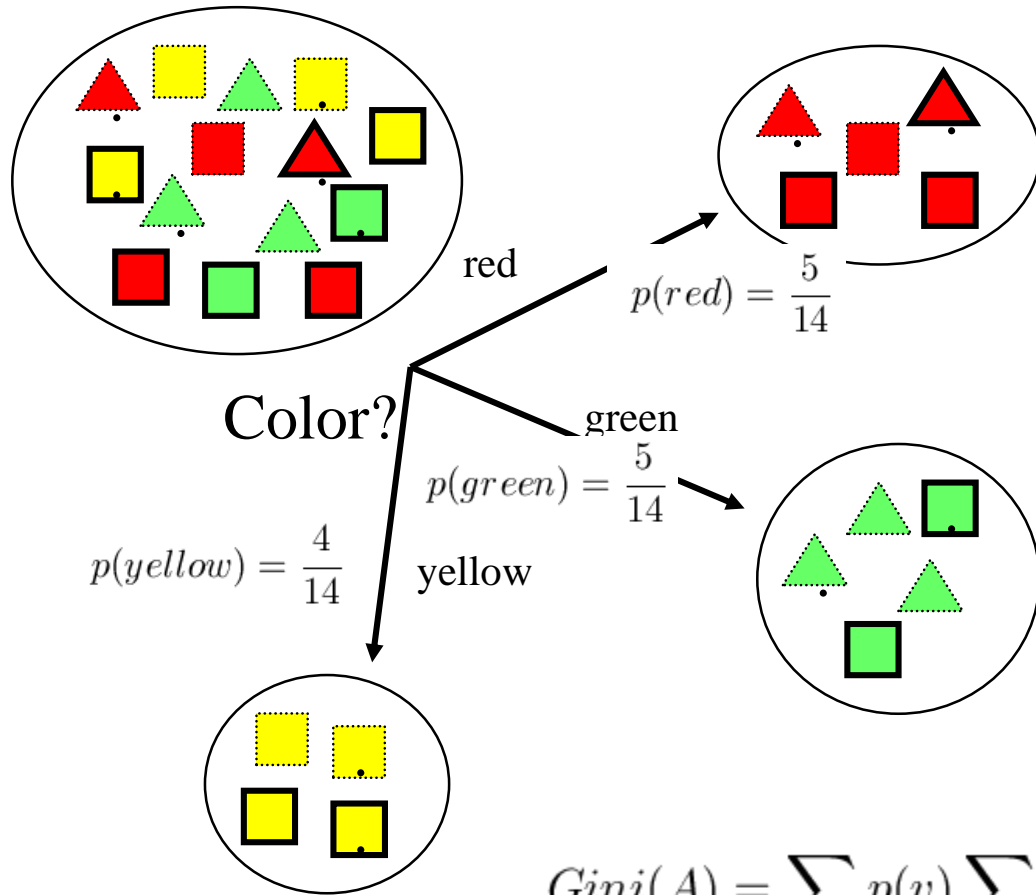
$$p(\square) = \frac{9}{14}$$

$$p(\Delta) = \frac{5}{14}$$

$$Gini = \sum_{i \neq j} p(i)p(j)$$

$$Gini = \frac{9}{14} \times \frac{5}{14} = 0.230$$

Gini Index for Color



$$Gini(A) = \sum_v p(v) \sum_{i \neq j} p(i|v)p(j|v)$$

$$Gini(\text{Color}) = \frac{5}{14} \times \left(\frac{3}{5} \times \frac{2}{5} \right) + \frac{5}{14} \times \left(\frac{2}{5} \times \frac{3}{5} \right) + \frac{4}{14} \times \left(\frac{4}{4} \times \frac{0}{4} \right) = 0.171$$

Gain of Gini Index

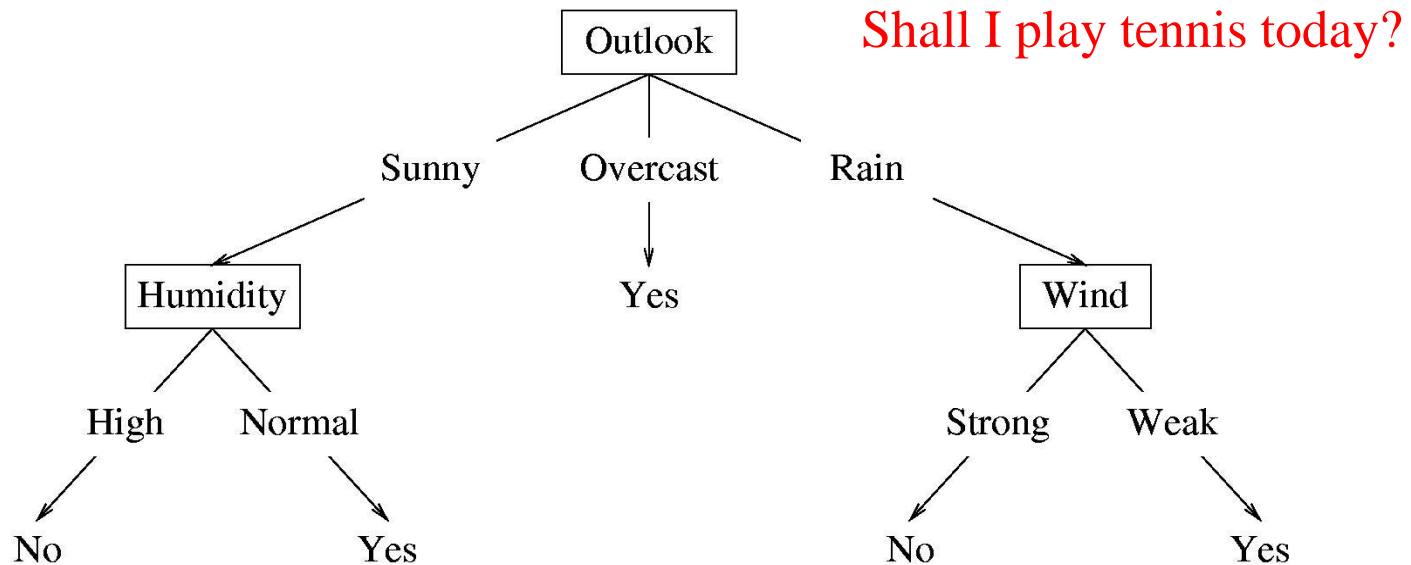
$$Gini = \frac{9}{14} \times \frac{5}{14} = 0.230$$

$$Gini(\text{Color}) = \frac{5}{14} \times \left(\frac{3}{5} \times \frac{2}{5}\right) + \frac{5}{14} \times \left(\frac{2}{5} \times \frac{3}{5}\right) + \frac{4}{14} \times \left(\frac{4}{4} \times \frac{0}{4}\right) = 0.171$$

$$GiniGain(\text{Color}) = 0.230 - 0.171 = 0.058$$

Decision Tree Hypothesis Space

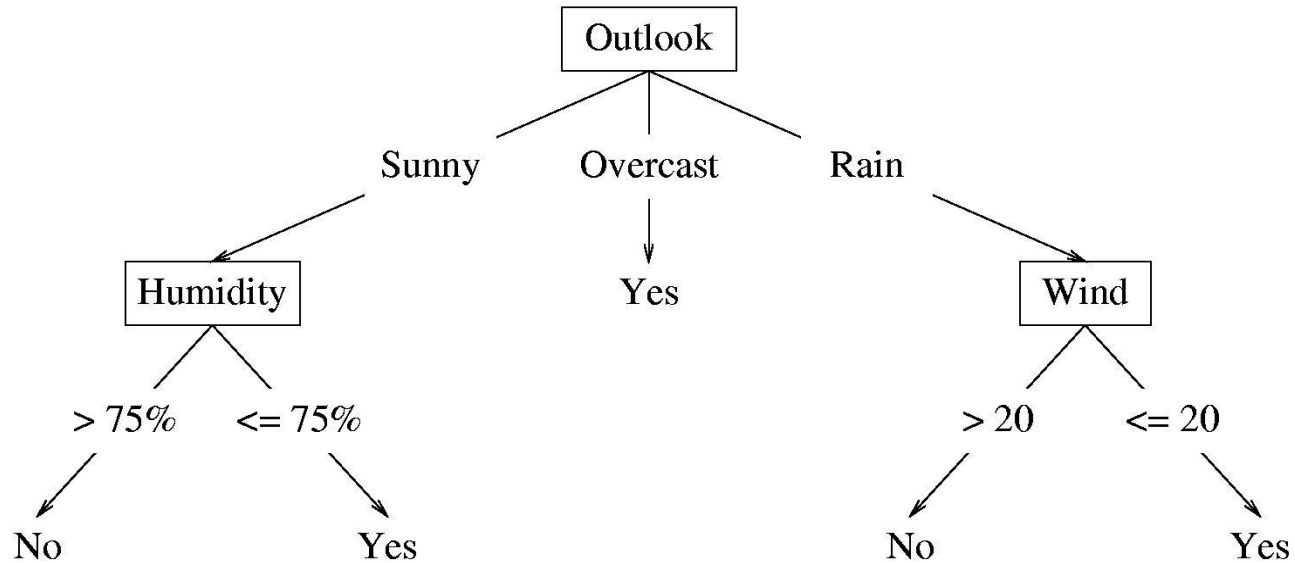
- **Internal nodes** test the value of particular features x_j and branch according to the results of the test.
- **Leaf nodes** specify the class $h(\mathbf{x})$.



Suppose the features are **Outlook** (x_1), **Temperature** (x_2), **Humidity** (x_3), and **Wind** (x_4). Then the feature vector $\mathbf{x} = (\text{Sunny}, \text{Hot}, \text{High}, \text{Strong})$ will be classified as **No**. The **Temperature** feature is irrelevant.

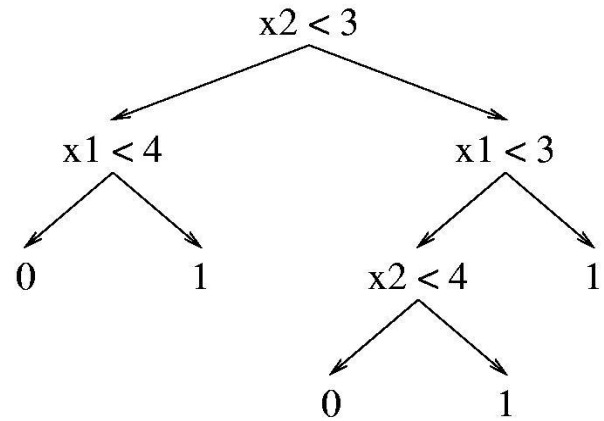
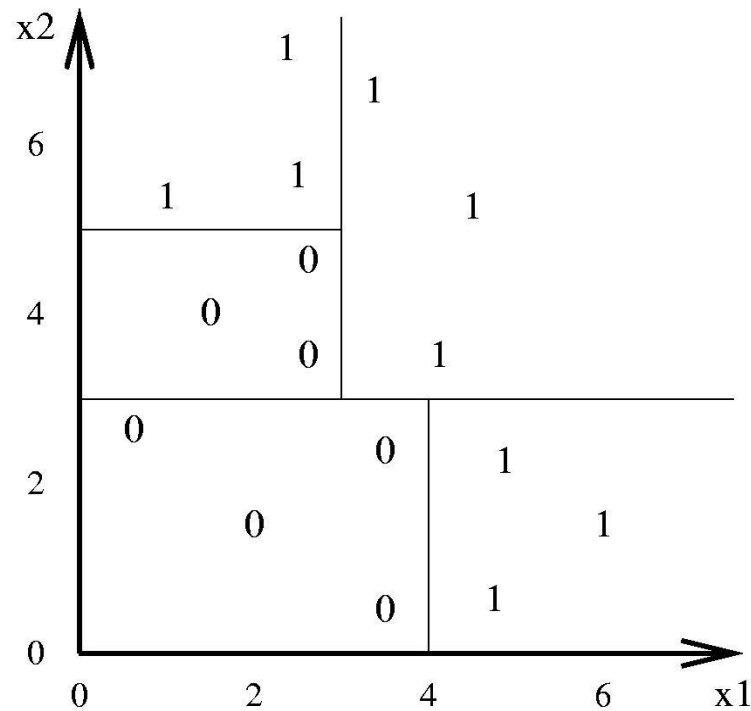
Decision Tree Hypothesis Space

If the features are continuous, internal nodes may test the value of a feature against a threshold.



Decision Tree Decision Boundaries

Decision trees divide the feature space into axis-parallel rectangles, and label each rectangle with one of the K classes.



Learning Decision Trees

Decision trees provide a very popular and efficient hypothesis space.

- **Variable Size.** Any boolean function can be represented.
- **Deterministic.**
- **Discrete and Continuous Parameters.**

Learning Algorithm for Decision Trees

The same basic learning algorithm has been discovered by many people independently:

GROWTREE(S)

if ($y = 0$ for all $\langle \mathbf{x}, y \rangle \in S$) **return** new leaf(0)

else if ($y = 1$ for all $\langle \mathbf{x}, y \rangle \in S$) **return** new leaf(1)

else

 choose best attribute x_j

$S_0 =$ all $\langle \mathbf{x}, y \rangle \in S$ with $x_j = 0$;

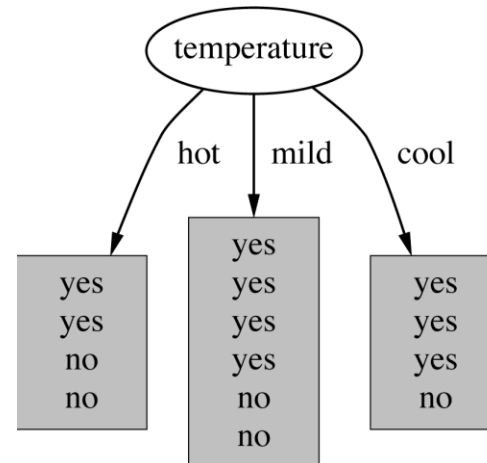
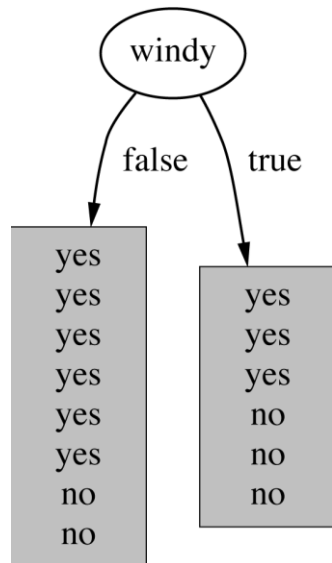
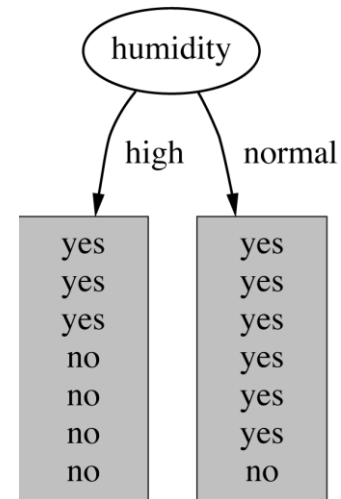
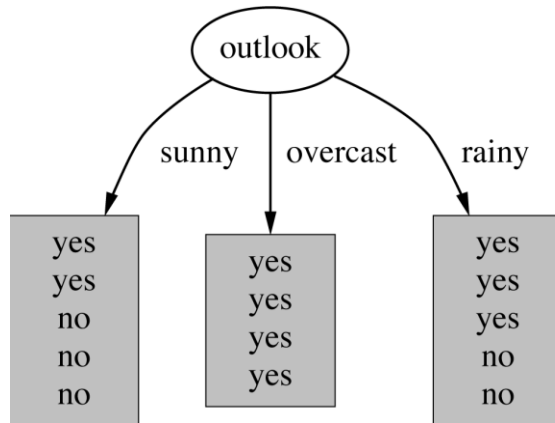
$S_1 =$ all $\langle \mathbf{x}, y \rangle \in S$ with $x_j = 1$;

return new node(x_j , **GROWTREE**(S_0), **GROWTREE**(S_1))

How do we choose the best attribute?

What should that attribute do for us?

Which attribute to select?



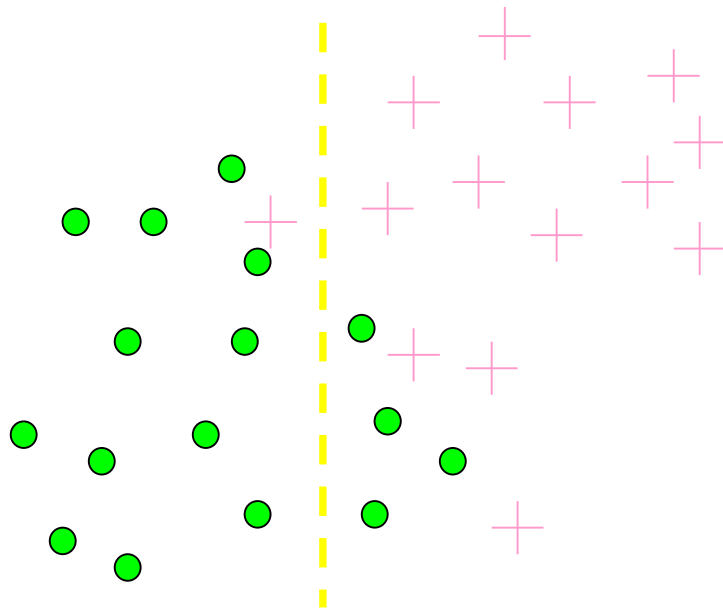
Criterion for attribute selection

- Which is the best attribute?
 - The one which will result in the smallest tree
 - Heuristic: choose the attribute that produces the “purest” nodes
- Need a good measure of purity!
 - Maximal when?
 - Minimal when?

Information Gain

Which test is more informative?

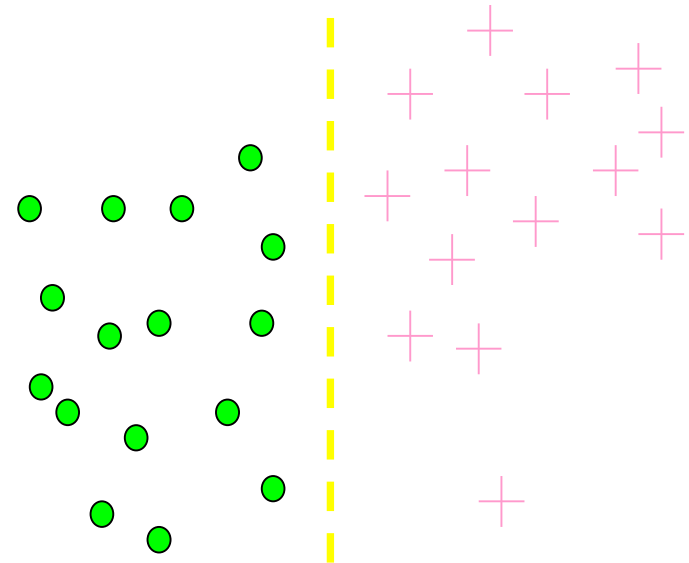
**Split over whether
Balance exceeds 50K**



Less or equal 50K

Over 50K

**Split over whether
applicant is employed**

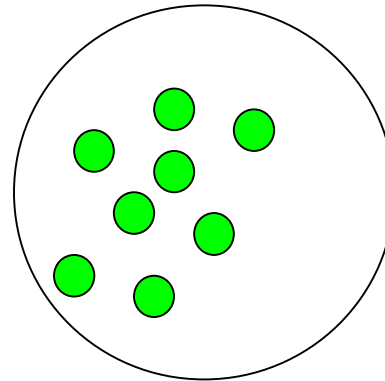
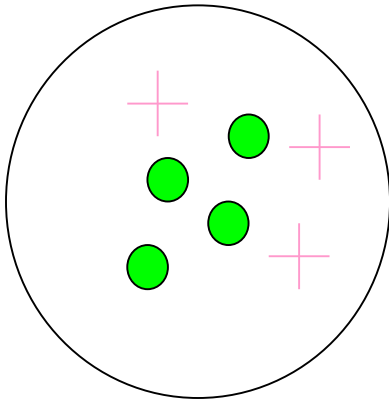


Unemployed

Employed

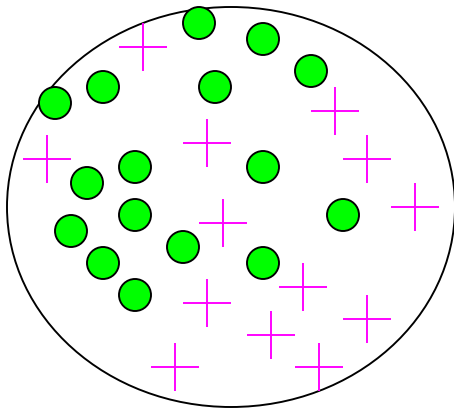
Impurity/Entropy (informal)

- Measures the level of **impurity** in a group of examples

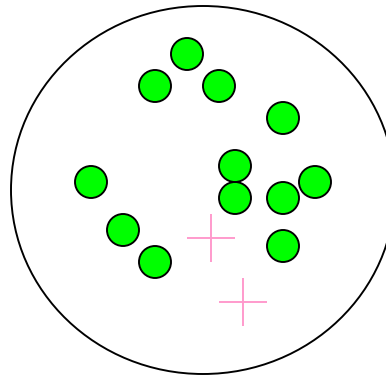


Impurity

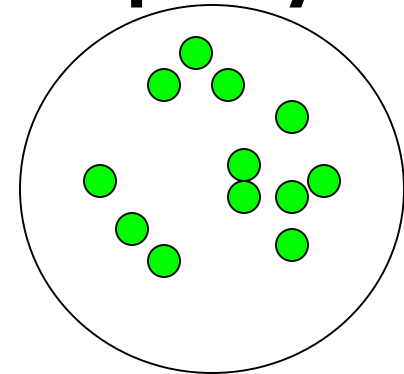
Very impure group



Less impure



Minimum impurity

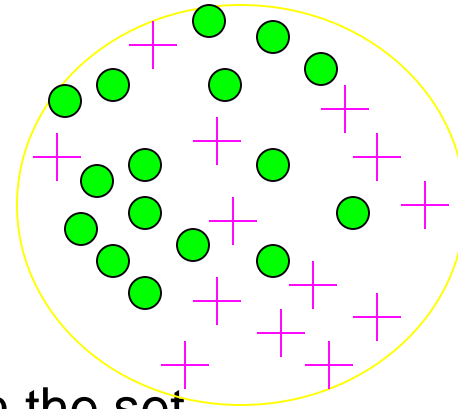


Entropy: a common way to measure impurity

- Entropy =
$$\sum_i -p_i \log_2 p_i$$

p_i is the probability of class i

Compute it as the proportion of class i in the set.



- Entropy comes from information theory. The higher the entropy the more the information content.

What does that mean for learning from examples?

2-Class Cases:

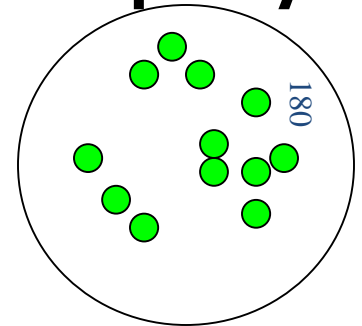
- What is the entropy of a group in which all examples belong to the same class?
 - entropy = $-1 \log_2 1 = 0$

not a good training set for learning

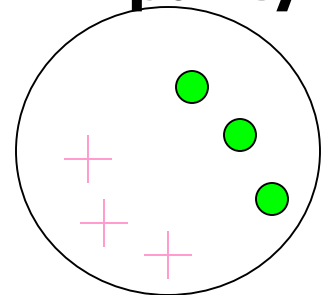
- What is the entropy of a group with 50% in either class?
 - entropy = $-0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$

good training set for learning

Minimum impurity



Maximum impurity



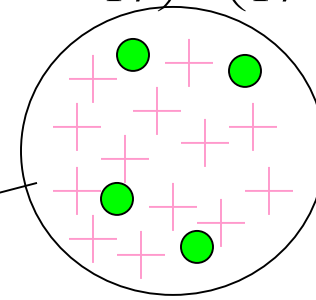
Information Gain

- We want to determine **which attribute** in a given set of training feature vectors is **most useful** for discriminating between the classes to be learned.
- **Information gain** tells us how important a given attribute of the feature vectors is.
- We will use it to decide the ordering of attributes in the nodes of a decision tree.

Calculating Information Gain

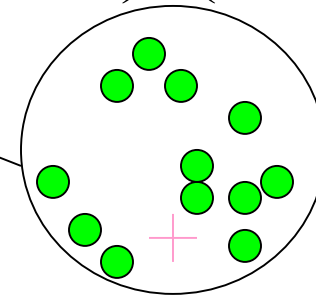
Information Gain = entropy(parent) – [average entropy(children)]

child entropy $-\left(\frac{13}{17} \cdot \log_2 \frac{13}{17}\right) - \left(\frac{4}{17} \cdot \log_2 \frac{4}{17}\right) = 0.787$



17 instances
182

child entropy $-\left(\frac{1}{13} \cdot \log_2 \frac{1}{13}\right) - \left(\frac{12}{13} \cdot \log_2 \frac{12}{13}\right) = 0.391$



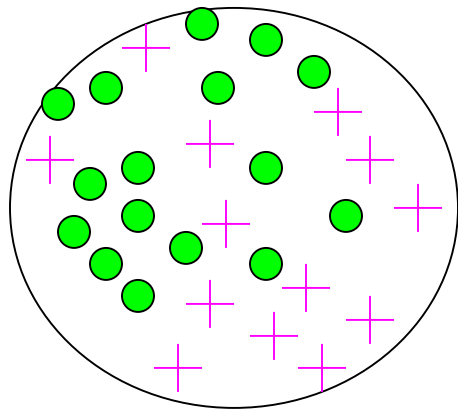
13 instances

parent entropy $-\left(\frac{14}{30} \cdot \log_2 \frac{14}{30}\right) - \left(\frac{16}{30} \cdot \log_2 \frac{16}{30}\right) = 0.996$

(Weighted) Average Entropy of Children = $\left(\frac{17}{30} \cdot 0.787\right) + \left(\frac{13}{30} \cdot 0.391\right) = 0.615$

Information Gain = 0.996 - 0.615 = 0.38

Entire population (30 instances)



Entropy-Based Automatic Decision Tree Construction

Training Set S

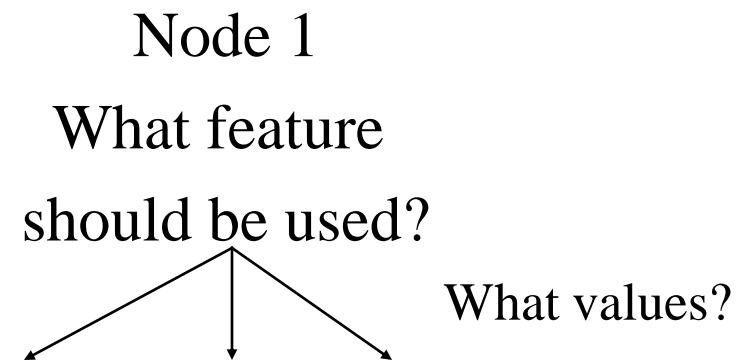
$x_1=(f_{11},f_{12},\dots,f_{1m})$

$x_2=(f_{21},f_{22}, \dots, f_{2m})$

.

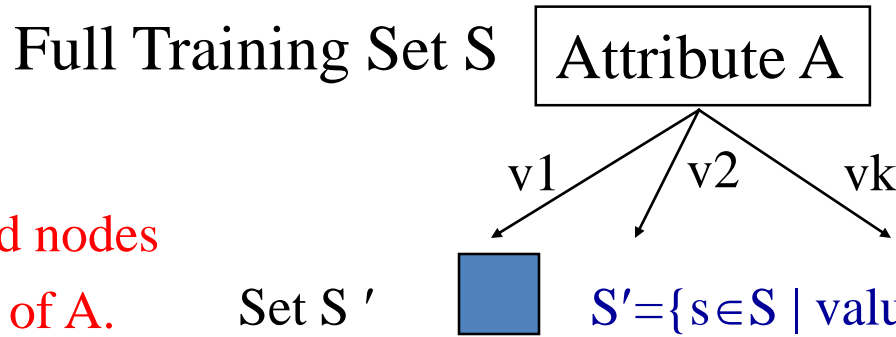
.

$x_n=(f_{n1},f_{n2}, \dots, f_{nm})$



Quinlan suggested **information gain** in his ID3 system and later the **gain ratio**, both based on **entropy**.

Using Information Gain to Construct a Decision Tree



Choose the attribute A with highest information gain for the full training set at the root of the tree

181

Construct child nodes for each value of A.

Each has an associated subset of vectors in which A has a particular value.

repeat recursively till when?

Information gain has the disadvantage that it prefers attributes with large number of values that split the data into small, pure subsets. Quinlan's gain ratio did some normalization to improve this.

Information Content

The information content $I(C;F)$ of the class variable C with possible values $\{c_1, c_2, \dots, c_m\}$ with respect to the feature variable F with possible values $\{f_1, f_2, \dots, f_d\}$ is defined by:

$$I(C; F) = \sum_{i=1}^m \sum_{j=1}^d P(C = c_i, F = f_j) \log_2 \frac{P(C = c_i, F = f_j)}{P(C = c_i)P(F = f_j)}$$

- $P(C = c_i)$ is the probability of class C having value c_i .
- $P(F=f_j)$ is the probability of feature F having value f_j .
- $P(C=c_i, F=f_j)$ is the joint probability of class $C = c_i$ and variable $F = f_j$.

These are estimated from frequencies in the training data.

Simple Example

- Sample Example

X	Y	Z	C
1	1	1	I
1	1	0	I
0	0	1	II
1	0	0	II

How would you distinguish class I from class II?

Example (cont)

X	Y	Z	C
1	1	1	I
1	1	0	I
0	0	1	II
1	0	0	II

$$\begin{aligned}I(C, X) &= P(C = I, X = 1) \log_2 \frac{P(C = I, X = 1)}{P(C = I)P(X = 1)} \\ &+ P(C = I, X = 0) \log_2 \frac{P(C = I, X = 0)}{P(C = I)P(X = 0)} \\ &+ P(C = II, X = 1) \log_2 \frac{P(C = II, X = 1)}{P(C = II)P(X = 1)} \\ &+ P(C = II, X = 0) \log_2 \frac{P(C = II, X = 0)}{P(C = II)P(X = 0)} \\ &= .5 \log_2 \frac{.5}{.5 \times .75} + 0 + .25 \log_2 \frac{.25}{.5 \times .25} + .25 \log_2 \frac{.25}{.5 \times .75} \\ &= 0.311\end{aligned}$$

$$\begin{aligned}I(C, Y) &= .5 \log_2 \frac{.5}{.5 \times .5} + 0 + .5 \log_2 \frac{.5}{.5 \times .5} + 0 \\ &= 1.0\end{aligned}$$

$$\begin{aligned}I(C, Z) &= .25 \log_2 \frac{.25}{.5 \times .5} + .25 \log_2 \frac{.25}{.5 \times .5} + .25 \log_2 \frac{.25}{.5 \times .5} + .25 \log_2 \frac{.25}{.5 \times .5} \\ &= 0.0\end{aligned}$$

Which attribute is best? Which is worst? Does it make sense?

Using Information Content

- Start with the root of the decision tree and the whole training set.
- Compute $I(C,F)$ for each feature F .
- Choose the feature F with highest information content for the root node.
- Create branches for each value f of F .
- On each branch, create a new node with reduced training set and repeat recursively.

Non-Boolean Features

- **Features with multiple discrete values**

Construct a multiway split?

Test for one value versus all of the others?

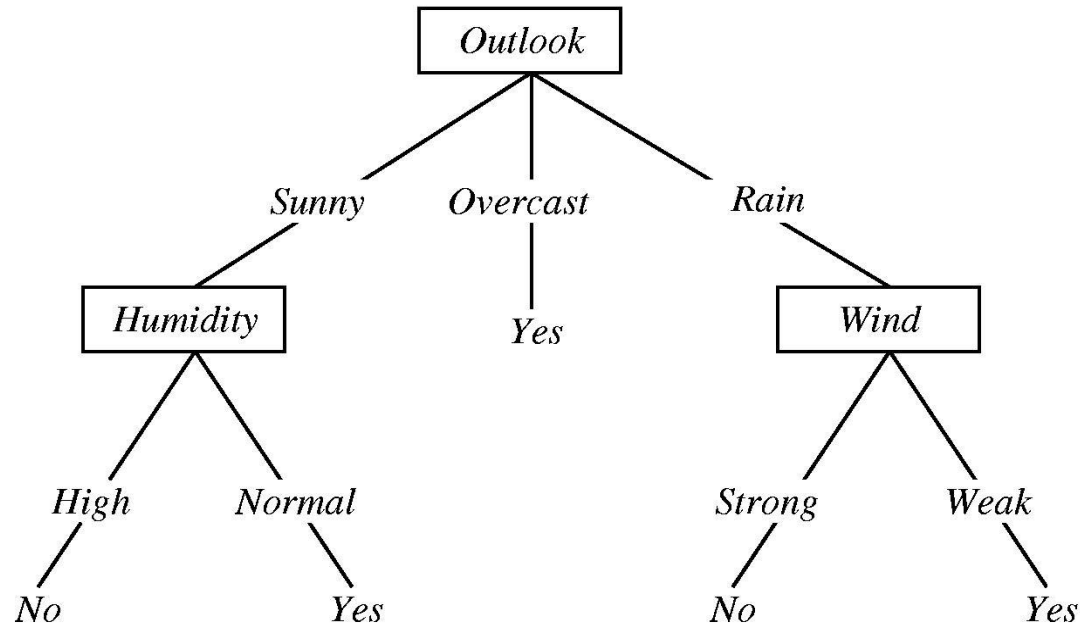
Group the values into two disjoint subsets?

- **Real-valued features**

Consider a threshold split using each observed value of the feature.

Whichever method is used, the mutual information can be computed to choose the best split.

Overfitting in Decision Trees












Consider adding a noisy training example:

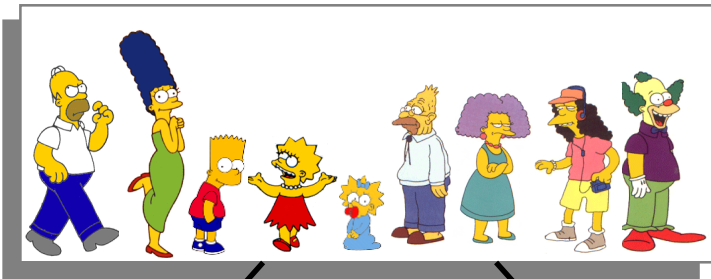
Sunny, Hot, Normal, Strong, PlayTennis=No

What effect on tree?

Example: The Simpsons

Person	Hair Length	Weight	Age	Class
 Homer	0"	250	36	M
 Marge	10"	150	34	F
 Bart	2"	90	10	M
 Lisa	6"	78	8	F
 Maggie	4"	20	1	F
 Abe	1"	170	70	M
 Selma	8"	160	41	F
 Otto	10"	180	38	M
 Krusty	6"	200	45	M

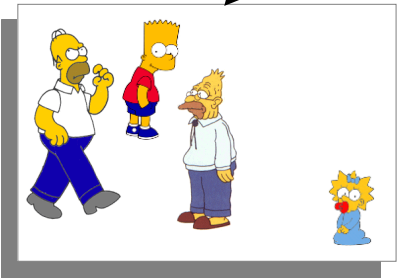
	Comic	8"	290	38	?
---	-------	----	-----	----	----------



$$Entropy(S) = -\frac{p}{p+n} \log_2\left(\frac{p}{p+n}\right) - \frac{n}{p+n} \log_2\left(\frac{n}{p+n}\right)$$

$$Entropy(4F,5M) = -(4/9)\log_2(4/9) - (5/9)\log_2(5/9) = 0.9911$$

yes
no
Hair Length <= 5?



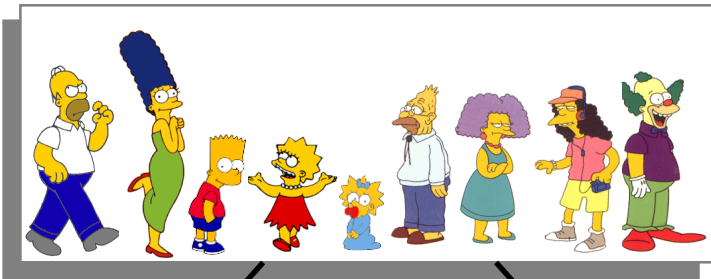
Let us try splitting
on *Hair length*

$$Entropy(1F,3M) = -(1/4)\log_2(1/4) - (3/4)\log_2(3/4) = 0.8113$$

$$Entropy(3F,2M) = -(3/5)\log_2(3/5) - (2/5)\log_2(2/5) = 0.9710$$

$$Gain(A) = E(\text{Current set}) - \sum E(\text{all child sets})$$

$$Gain(\text{Hair Length} \leq 5) = 0.9911 - (4/9 * 0.8113 + 5/9 * 0.9710) = 0.0911$$



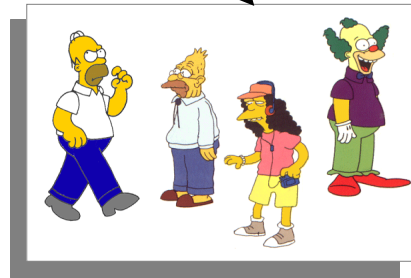
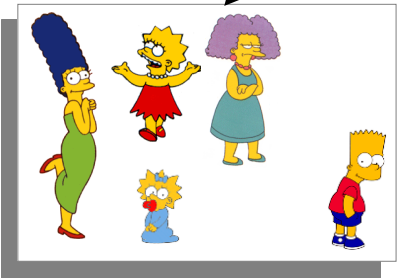
$$Entropy(S) = -\frac{p}{p+n} \log_2\left(\frac{p}{p+n}\right) - \frac{n}{p+n} \log_2\left(\frac{n}{p+n}\right)$$

$$Entropy(4F,5M) = -(4/9)\log_2(4/9) - (5/9)\log_2(5/9) = 0.9911$$

yes

Weight <= 160?

no



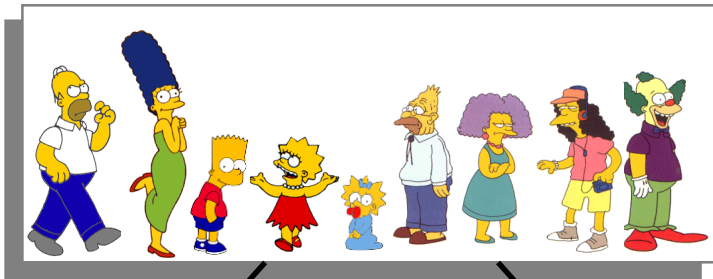
Let us try splitting on *Weight*

$$Entropy(4F,1M) = -(4/5)\log_2(4/5) - (1/5)\log_2(1/5) = 0.7219$$

$$Entropy(0F,4M) = -(0/4)\log_2(0/4) - (4/4)\log_2(4/4) = 0$$

$$Gain(A) = E(Current\ set) - \sum E(all\ child\ sets)$$

$$Gain(Weight \leq 160) = 0.9911 - (5/9 * 0.7219 + 4/9 * 0) = 0.5900$$



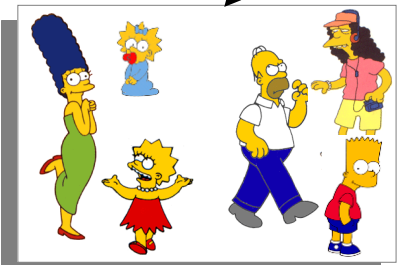
$$Entropy(S) = -\frac{p}{p+n} \log_2\left(\frac{p}{p+n}\right) - \frac{n}{p+n} \log_2\left(\frac{n}{p+n}\right)$$

$$Entropy(4\mathbf{F},5\mathbf{M}) = -(4/9)\log_2(4/9) - (5/9)\log_2(5/9) = \mathbf{0.9911}$$

yes

age <= 40?

no



Let us try splitting on Age

$$Entropy(3\mathbf{F},3\mathbf{M}) = -(3/6)\log_2(3/6) - (3/6)\log_2(3/6) = \mathbf{1}$$

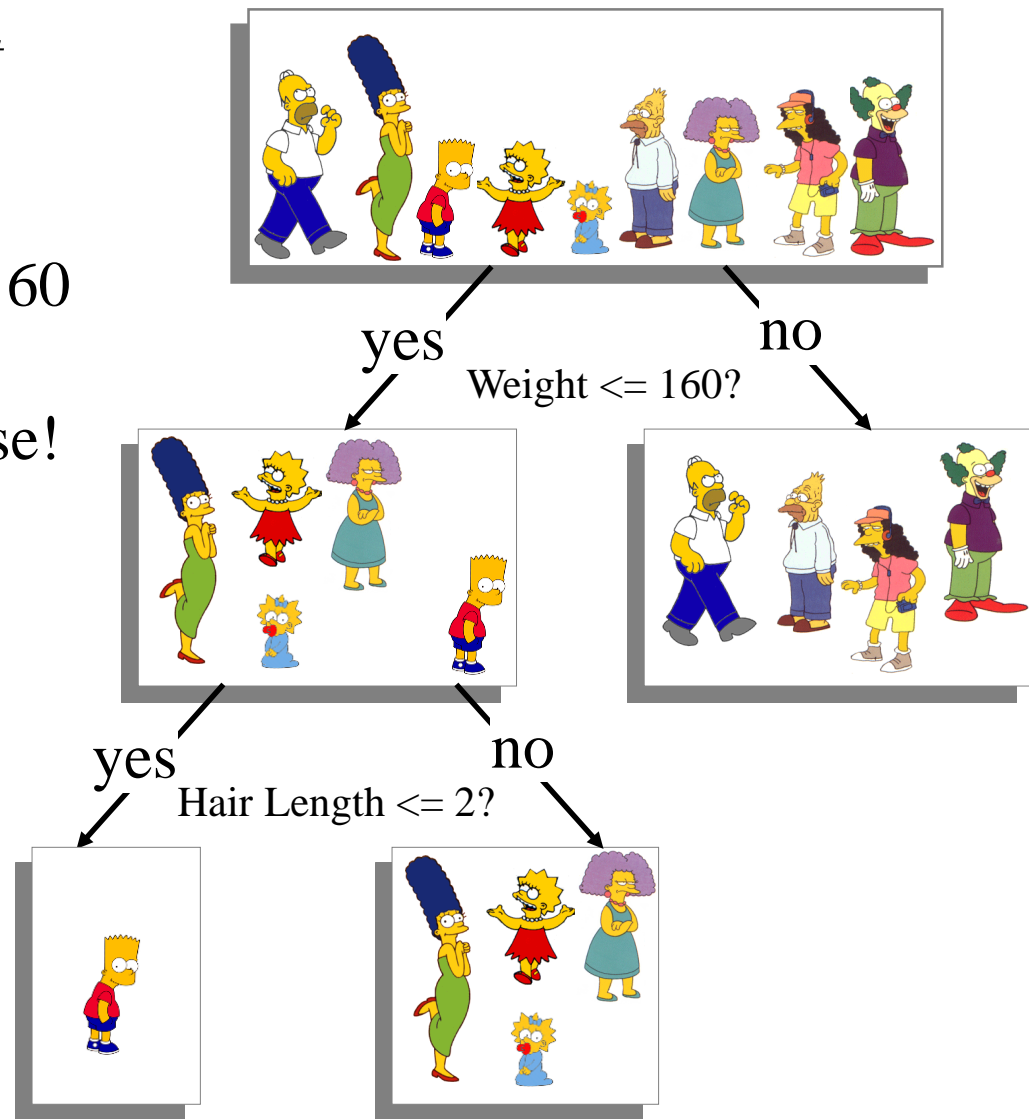
$$Entropy(1\mathbf{F},2\mathbf{M}) = -(1/3)\log_2(1/3) - (2/3)\log_2(2/3) = \mathbf{0.9183}$$

$$Gain(A) = E(\text{Current set}) - \sum E(\text{all child sets})$$

$$Gain(\text{Age} \leq 40) = \mathbf{0.9911} - (6/9 * \mathbf{1} + 3/9 * \mathbf{0.9183}) = \mathbf{0.0183}$$

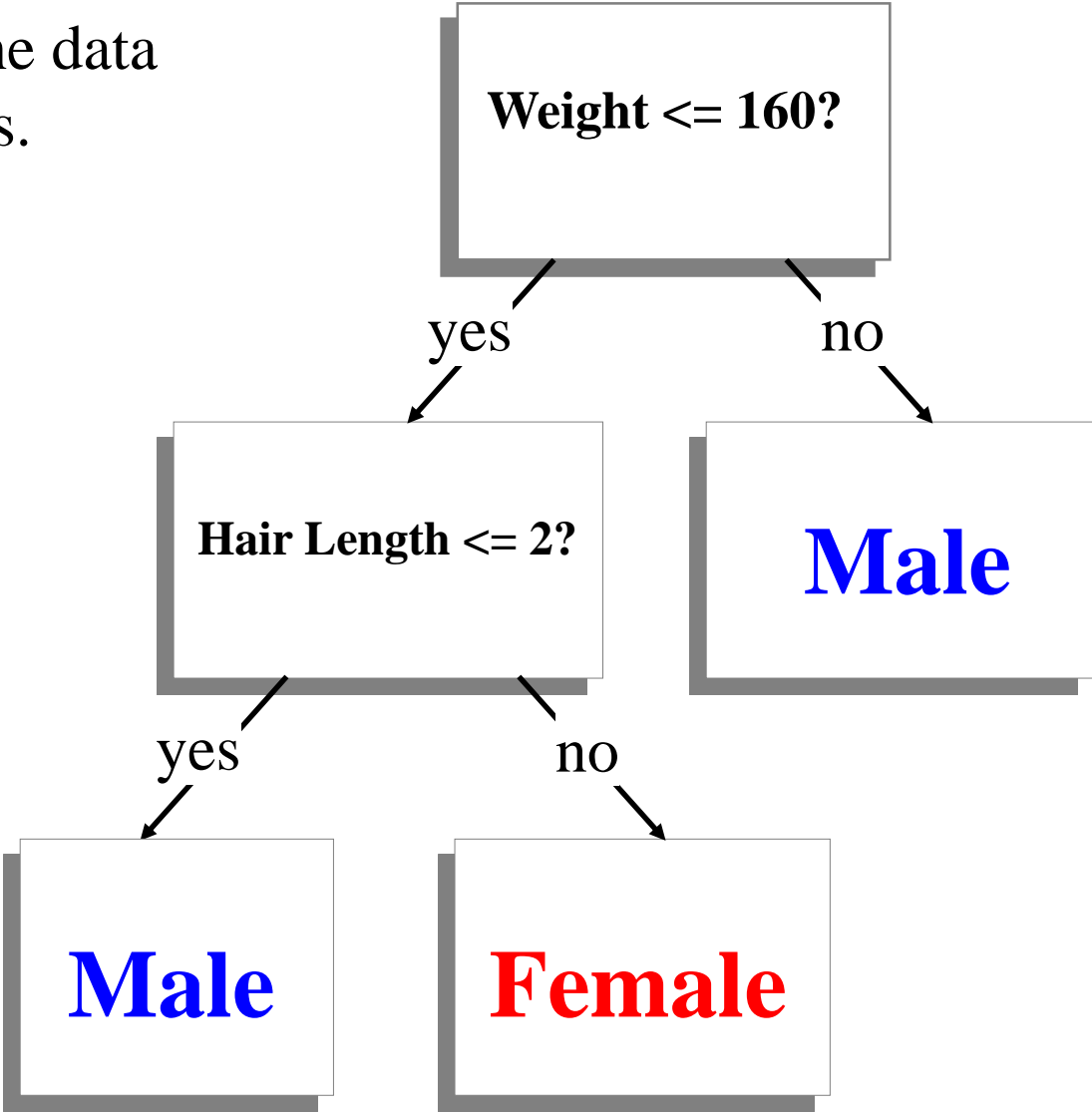
Of the 3 features we had, *Weight* was best. But while people who weigh over 160 are perfectly classified (as males), the under 160 people are not perfectly classified... So we simply recurse!

This time we find that we can split on *Hair length*, and we are done!

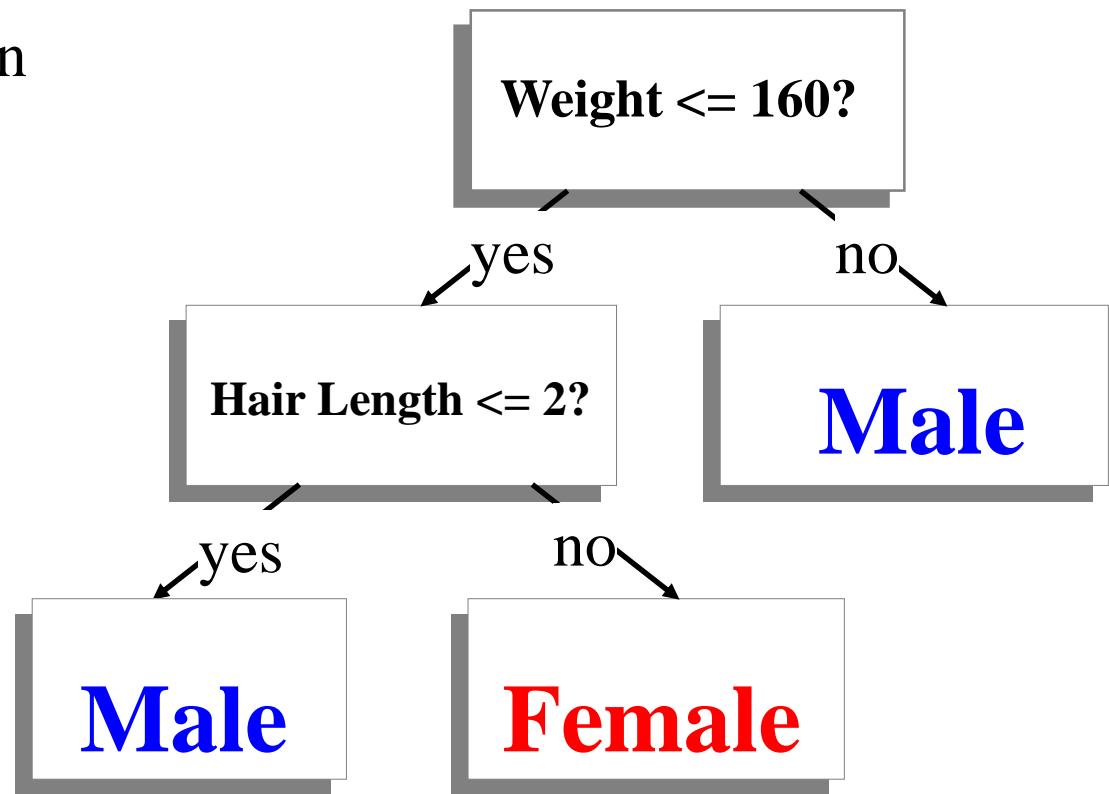


We need don't need to keep the data around, just the test conditions.

How would these people be classified?



It is trivial to convert Decision Trees to rules...



Rules to Classify Males/Females

If *Weight* **greater than** 160, classify as **Male**

Elseif *Hair Length* **less than or equal** to 2, classify as **Male**

Else classify as **Female**

Building Decision Tree [Q93]

- Top-down tree construction
 - At start, all training examples are at the root.
 - Partition the examples recursively by choosing one attribute each time.
- Bottom-up tree pruning
 - Remove subtrees or branches, in a bottom-up manner, to improve the estimated accuracy on new cases.

Top-Down Approach

- Top-Down Decision Tree Construction
- Choosing the Splitting Attribute
- Information Gain biased towards attributes with a large number of values
- Gain Ratio takes number and size of branches into account when choosing an attribute

Choosing the Splitting Attribute

- At each node, available attributes are evaluated on the basis of separating the classes of the training examples. A Goodness function is used for this purpose.
- Typical goodness functions:
 - information gain (ID3/C4.5)
 - information gain ratio
 - gini index

A criterion for attribute selection

- Which is the best attribute?
 - The one which will result in the smallest tree
 - Heuristic: choose the attribute that produces the “purest” nodes
- Popular *impurity criterion: information gain*
 - Information gain increases with the average purity of the subsets that an attribute produces
- Strategy: choose attribute that results in greatest information gain

Computing information

- Information is measured in *bits*
 - Given a probability distribution, the info required to predict an event is the distribution's *entropy*
 - Entropy gives the information required in bits (this can involve fractions of bits!)
- For a probability distribution with probabilities p_1, p_2, \dots, p_n , the entropy is given by:
$$\text{entropy}(p_1, p_2, \dots, p_n) = -p_1 \log p_1 - p_2 \log p_2 \dots - p_n \log p_n$$

Evaluation

- Training accuracy

- How many training instances can be correctly classify based on the available data?
- Is high when the tree is deep/large, or when there is less confliction in the training instances.
- however, higher training accuracy does not mean good generalization

- Testing accuracy

- Given a number of new instances, how many of them can we correctly classify?
- Cross validation

Strengths

- can generate understandable rules
- perform classification without much computation
- can handle continuous and categorical variables
- provide a clear indication of which fields are most important for prediction or classification

Weakness

- Not suitable for prediction of continuous attribute.
- Perform poorly with many class and small data.
- Computationally expensive to train.
 - At each node, each candidate splitting field must be sorted before its best split can be found.
 - In some algorithms, combinations of fields are used and a search must be made for optimal combining weights.
 - Pruning algorithms can also be expensive since many candidate sub-trees must be formed and compared.
- Do not treat well non-rectangular regions.

Summary

- Decision trees can be used to help predict the future
- The trees are easy to understand
- Decision trees work more efficiently with **discrete attributes**
- The trees may suffer from **error propagation**

What to remember about classifiers

- **No free lunch:** machine learning algorithms are tools, not dogmas
- Try simple classifiers first
- Better to have **smart features** and **simple classifiers** than simple features and smart classifiers
- Use increasingly powerful classifiers with **more training data** (bias-variance tradeoff)

Generalization



Training set (labels known)



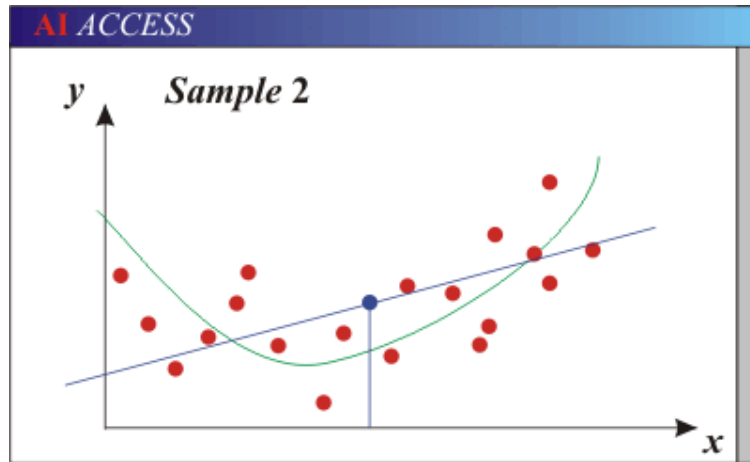
Test set (labels unknown)

- How well does a learned model generalize from the data it was trained on to a new test set?

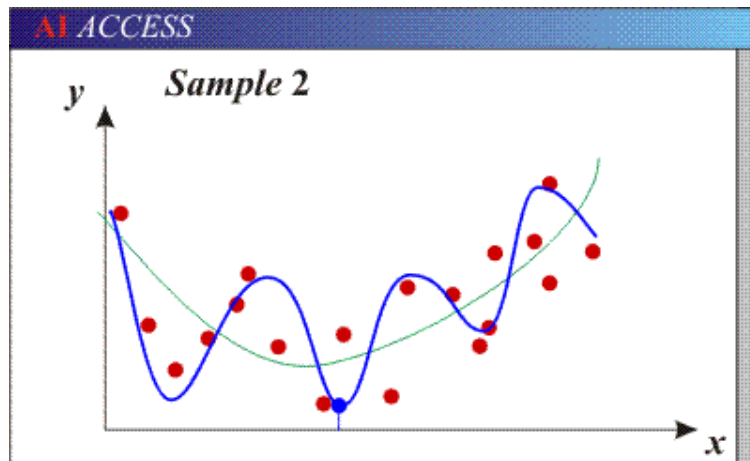
Generalization

- Components of generalization error
 - **Bias:** how much the average model over all training sets differ from the true model?
 - Error due to inaccurate assumptions/simplifications made by the model.
 - **Variance:** how much models estimated from different training sets differ from each other.
- **Underfitting:** model is too “simple” to represent all the relevant class characteristics
 - High bias (few degrees of freedom) and low variance
 - High training error and high test error
- **Overfitting:** model is too “complex” and fits irrelevant characteristics (noise) in the data
 - Low bias (many degrees of freedom) and high variance
 - Low training error and high test error

Bias-Variance Trade-off

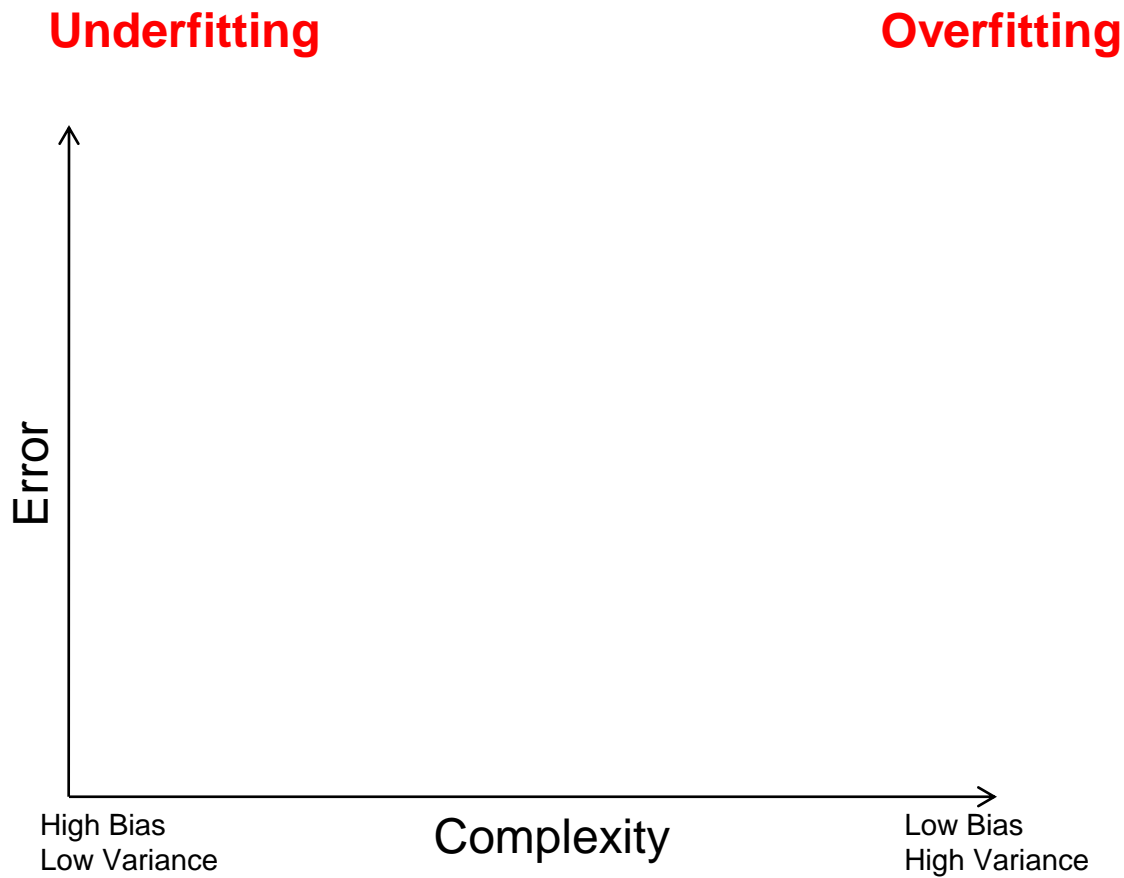


- Models with **too few parameters** are inaccurate because of a large bias (**not enough flexibility**).

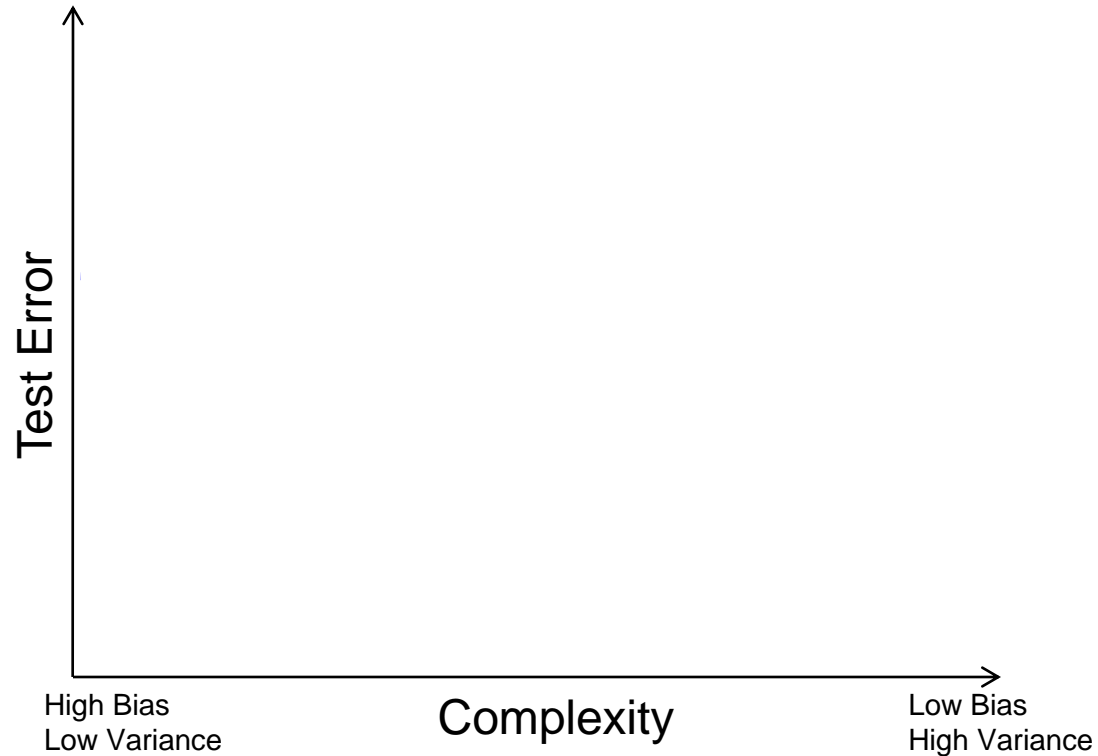


- Models with **too many parameters** are inaccurate because of a large variance (**too much sensitivity to the sample**).

Bias-variance tradeoff

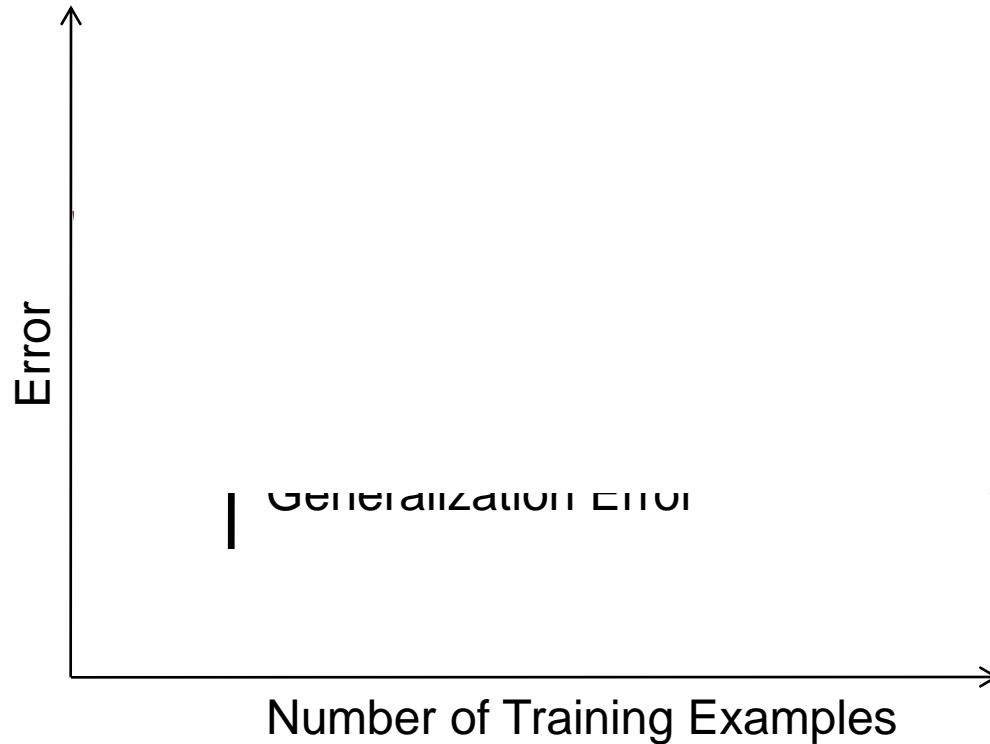


Bias-variance tradeoff



Effect of Training Size

Fixed prediction model



Remember...

- No classifier is inherently better than any other: you need to make assumptions to generalize
- Three kinds of error
 - **Inherent**: unavoidable
 - **Bias**: due to over-simplifications
 - **Variance**: due to inability to perfectly estimate parameters from limited data



How to reduce variance?

- Choose a simpler classifier
- Regularize the parameters
- Get more training data

Generative vs. Discriminative Classifiers

Generative Models

- Represent both the data and the labels
- Often, makes use of conditional independence and priors
- Examples
 - Naïve Bayes classifier
 - Bayesian network
- Models of data may apply to future prediction problems

Discriminative Models

- Learn to directly predict the labels from the data
- Often, assume a simple boundary (e.g., linear)
- Examples
 - Logistic regression
 - SVM
 - Boosted decision trees
- Often easier to predict a label from the data than to model the data

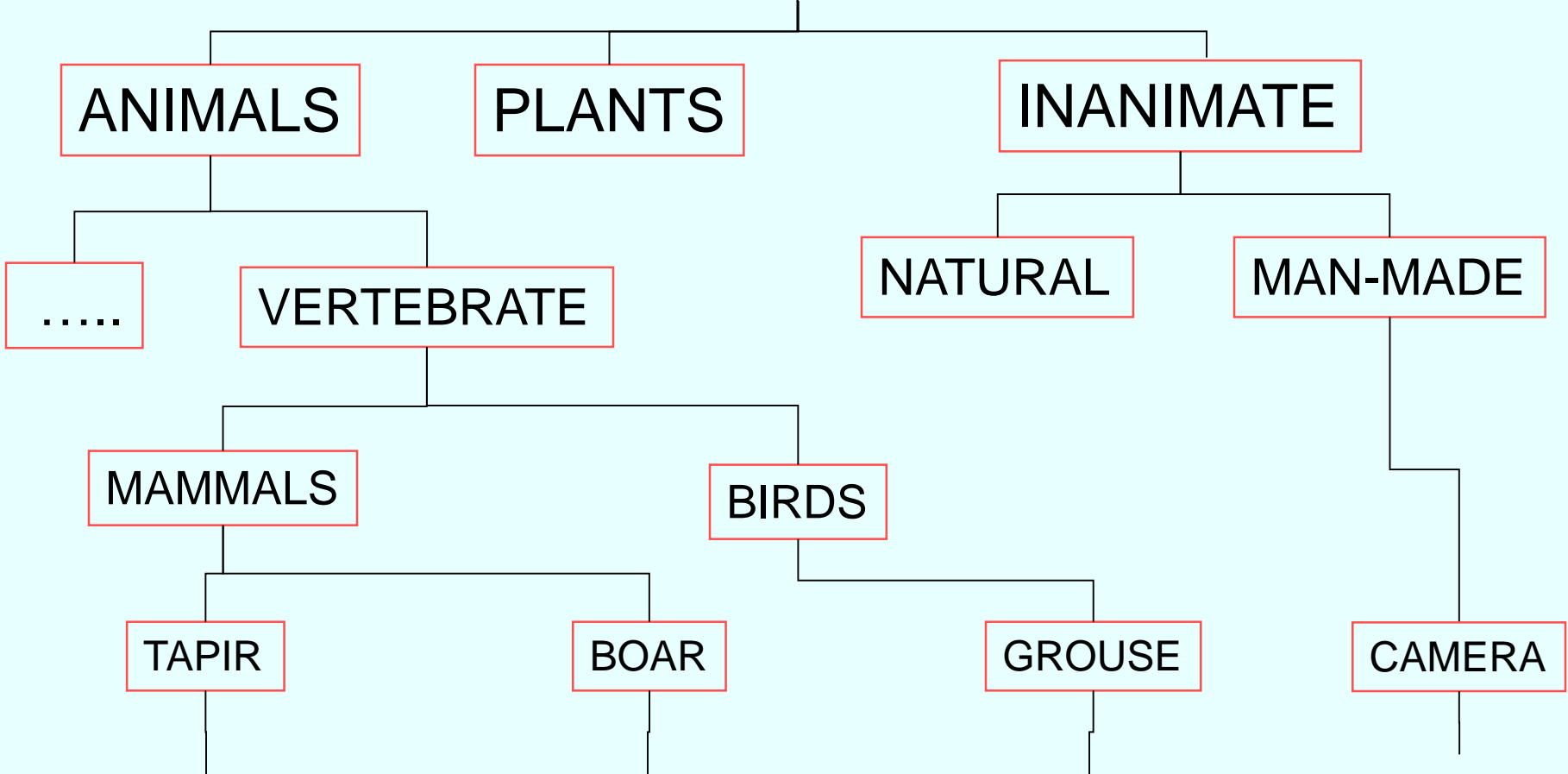
Other Issues for Image Classification



~10,000 to 30,000



OBJECTS



Specific recognition tasks



Scene categorization or classification



- outdoor/indoor
- city/forest/factory/etc.

Image annotation / tagging / attributes



- street
- people
- building
- mountain
- tourism
- cloudy
- brick
- ...

Object detection

- find pedestrians

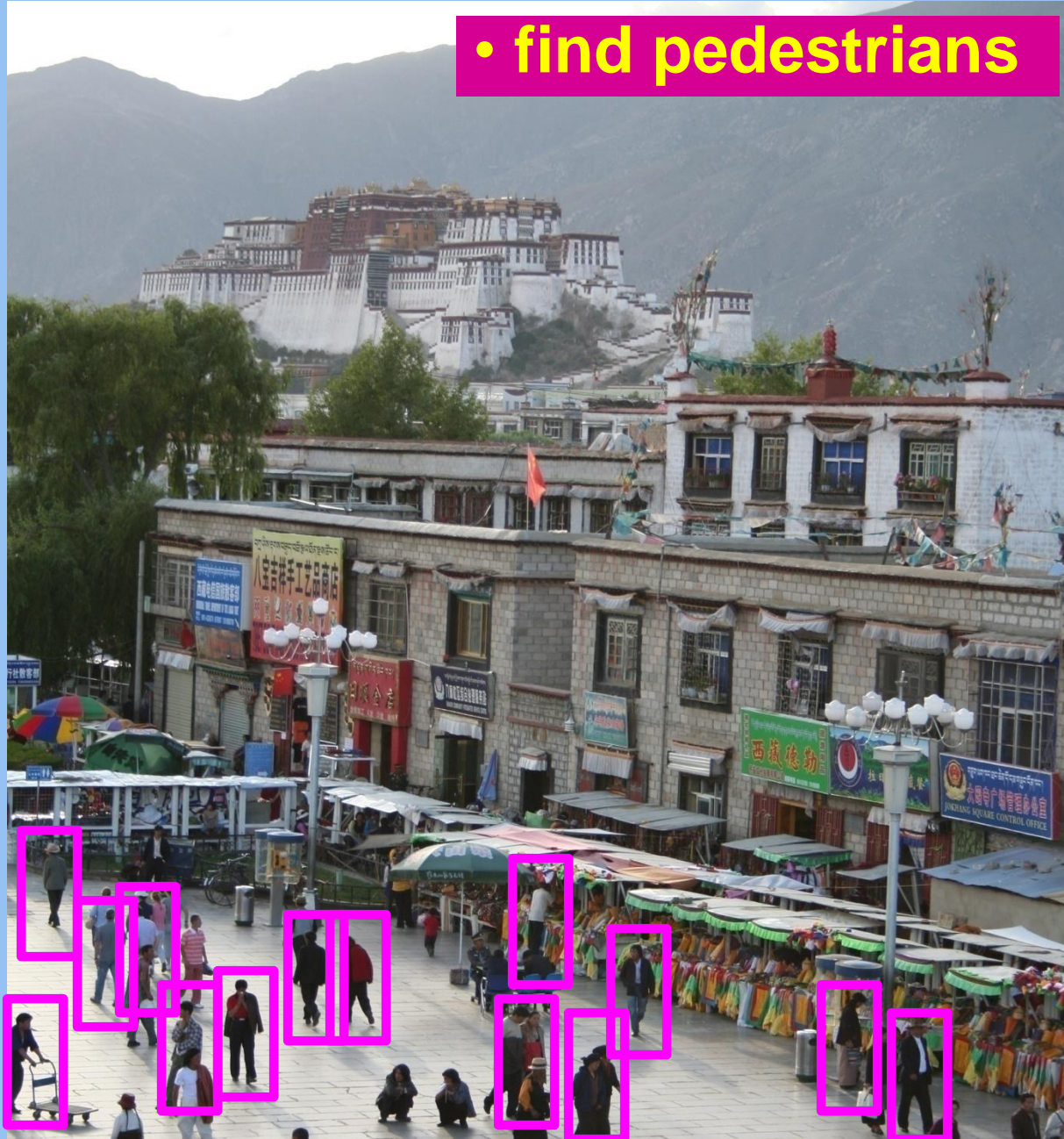


Image parsing / semantic segmentation



Scene understanding?

