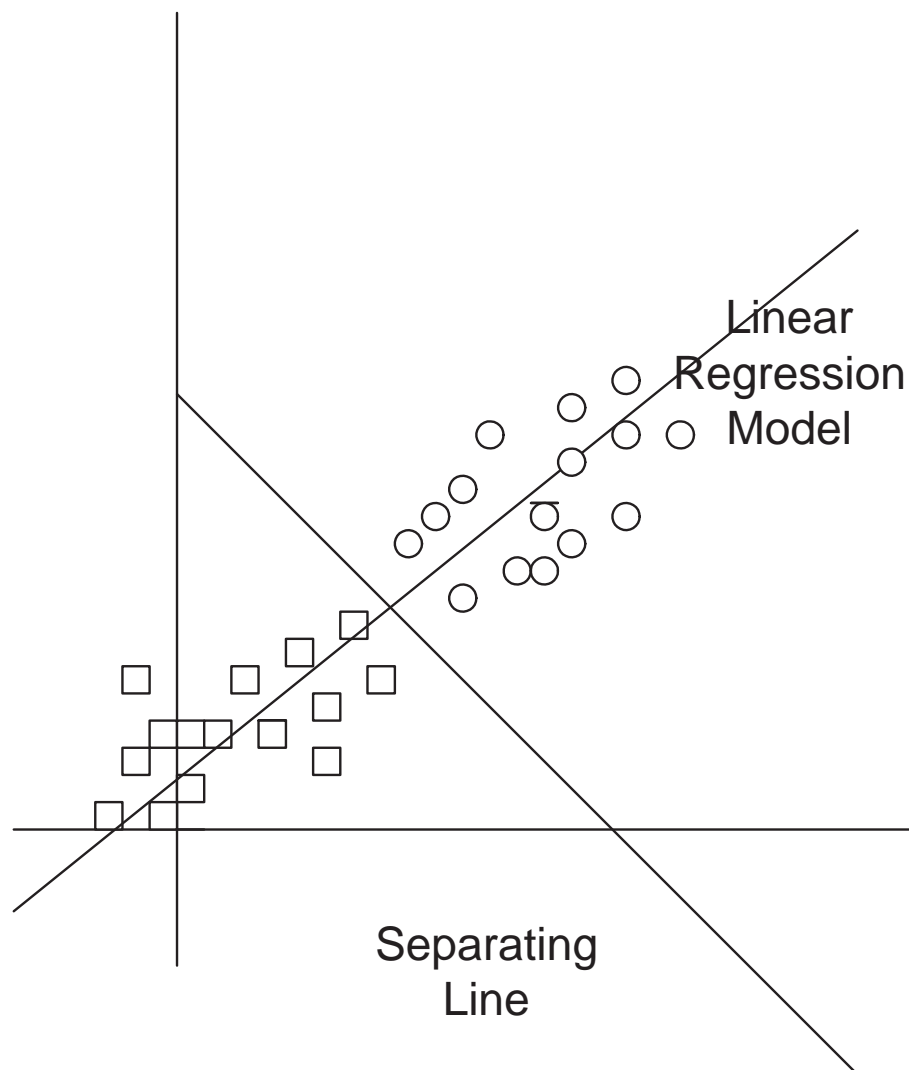


# Advanced Numeric Classifiers

---

Linear regression can help us to find some linear classifiers to separate point sets. But sometimes this approach does not work.



What to do?

# Geometric margin:I

---

Consider linear binary classification. Let

$$f(x) = w^T x + b = \sum_{i=1}^n w_i x_i + b, \quad \forall x \in \mathbb{R}^n.$$

Two classes in  $\mathbb{R}^n$ , denoted by '+' and '-'.

- $f(x) \geq 0$  if  $x$  is in the '+' class;
- $f(x) < 0$  if  $x$  is in the '-' class;

Then  $(w, b)$  is the hyperplane separating two classes. Let  $Y = \{1, -1\}$ , and define

$$S = ((x_1, y_1), \dots, (x_l, y_l)) \subset (X \times Y)^l.$$

**Definition:** The margin of an example  $(x_i, y_i)$  with respect to a hyperplane  $(w, b)$  is the quantity

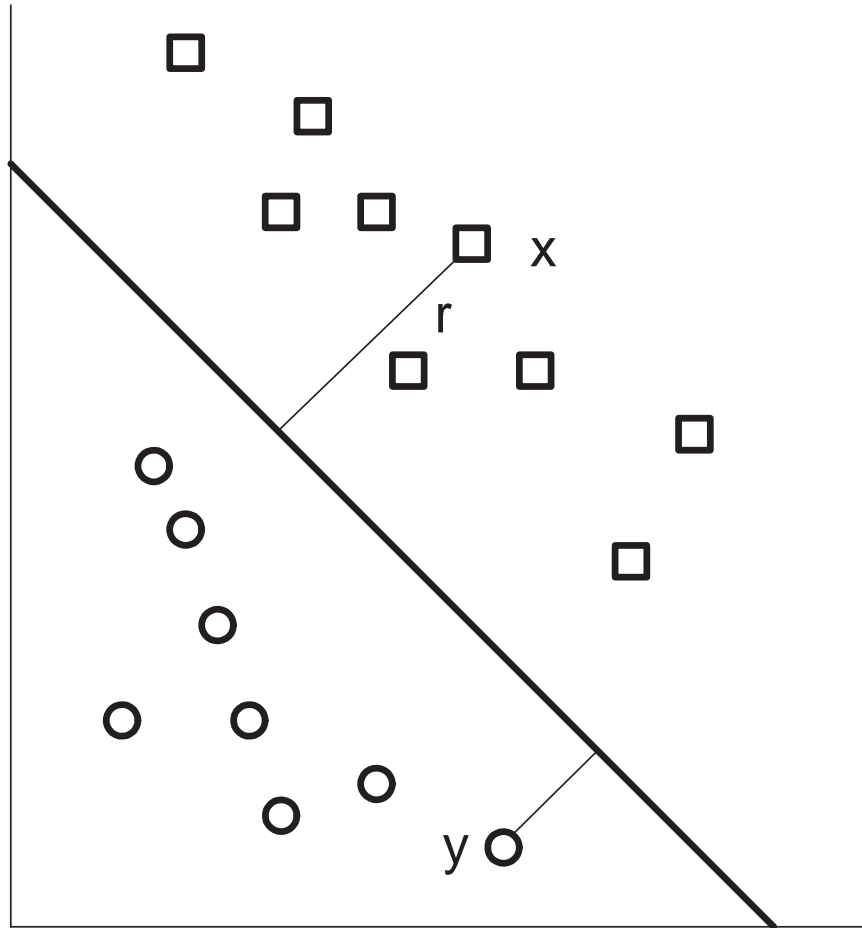
$$\gamma_i = y_i(w^T x_i + b).$$

**Definition** Fix a value  $\gamma > 0$ , the margin slack variable of an example  $(x_i, y_i)$  with respect to the hyperplane  $(w, b)$  and the target margin  $\gamma$  is

$$\zeta((x_i, y_i), (w, b), \gamma) = \zeta_i = \max(0, \gamma - y_i(w^T x_i + b)).$$

# Geometric margin:II

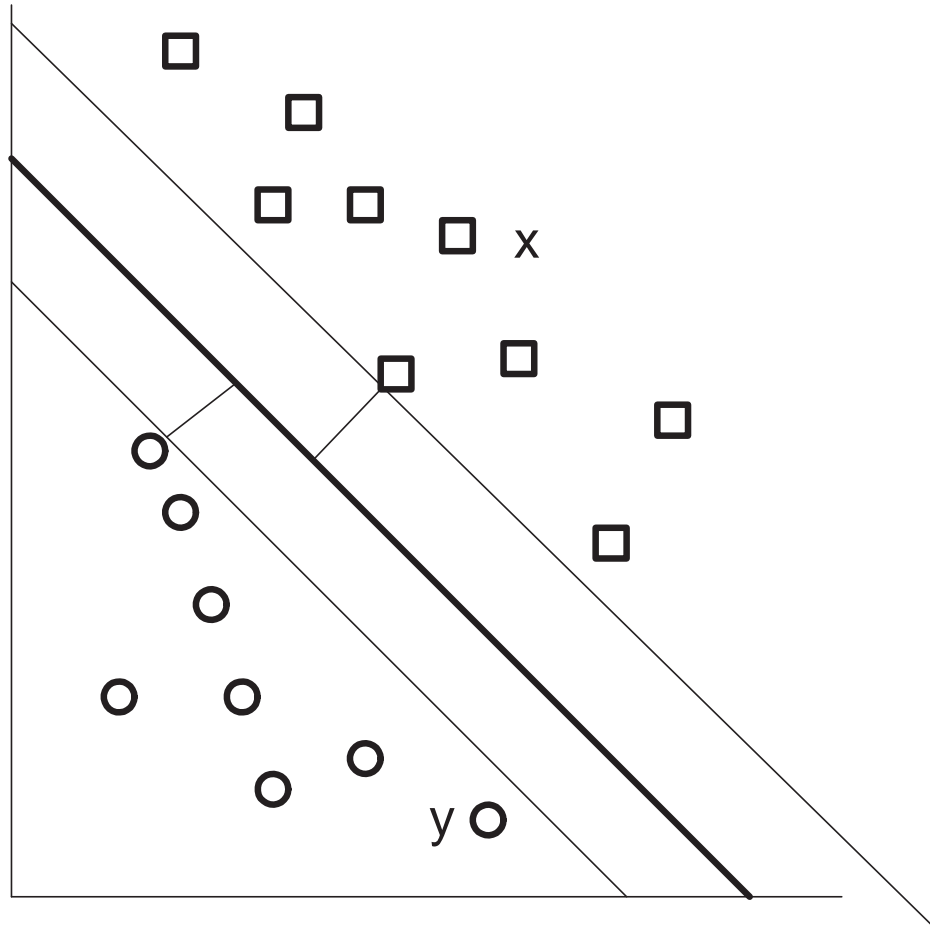
---



The geometric margin of two points

# Margin for point sets

---



Margin for a training set

# Rosenblatt's algorithm

---

**Input:** Training set  $S$  and learning rate  $\eta > 0$

$$w_0 = 0, b_0 = 0, k = 0; R = \max_{1 \leq i \leq l} \|x_i\|$$

Repeat for  $i = 1$  to  $l$

if  $y_i(w_k^T x_i + b_k) \leq 0$  then

$$w_{k+1} = w_k + \eta y_i x_i;$$

$$b_{k+1} = b_k + \eta y_i R^2, k := k + 1$$

end if

end for

until no mistakes made within the for loop,  
and return  $(w_k, b_k)$  and  $k$ .

**Theorem:** Let  $S$  be the training set and  $R = \max_{1 \leq i \leq l} \|x_i\|$ . Suppose that there exists a vector  $W_{opt}$  such that  $\|W_{opt}\| = 1$  and

$$y_i(w_{opt}^T x_i + b_{opt}) \geq \gamma, \forall 1 \leq i \leq l.$$

Then the mistakes made by the perceptron algorithm on  $S$  is at most  $4R^2/\gamma^2$ .

This theorem gives the finite convergence of the algorithm for separable problems.

# Examples

---

Consider the problem of classifying two sets

$$S_1 = \{(1, 2), (2, 1)\}; \quad S_2 = \{(1, -1), (0, 3)\}$$

For the given data set,  $R = 3$ . We choose  $\eta = 1$ . Following the order

$$(1, 2) \rightarrow (2, 1) \rightarrow (1, -1) \rightarrow (0, 3),$$

Rosenblatt's method takes 22 iterations to find a linear classifier

$$7x_1 + 2x_2 - 9.$$

Change the order to

$$(0, 3) \rightarrow (1, -1) \rightarrow (2, 1) \rightarrow (1, 2),$$

Rosenblatt's algorithm find another classifier

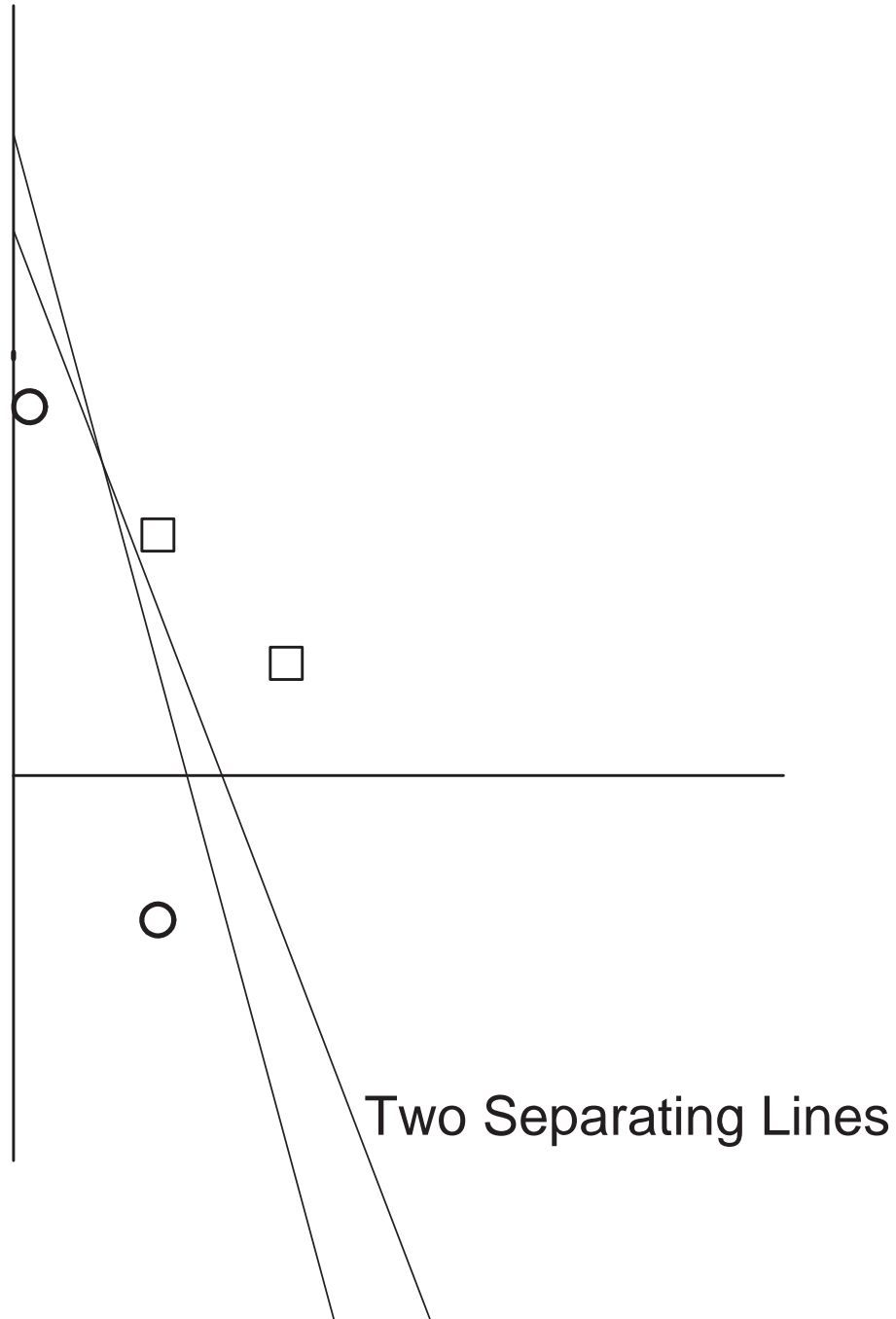
$$15x_1 + 3x_2 - 18,$$

with 37 iterations.

**Note:** The process is unstable and sensitive to the order of the instances in the data set

# Two separating lines

---



# Maximal margin classifier

---

**Observation:** Perceptron algorithm is not robust (killed by Minsky and papert in 1980s);

- We have the freedom of scaling  $(w, b)$  if the linear classifier can classify the classes.

Denote the functional margin by  $m_s(f)$  with  $f(x) = w^T x + b$ , and we want to find the maximal margin classifier, that is to maximize the parameter  $\gamma$  satisfying

$$\gamma_i = y_i(w_{opt}^T x_i + b_{opt}) \geq \gamma, \forall 1 \leq i \leq l.$$

The larger the parameter  $\gamma$  is, more robust the classifier!

# Geometric Margin

---

To find such a robust classifier, we refer to the so-called

**Geometric margin:** the functional margin of a normalized weight vector.

Instead of maximizing  $\gamma$ , we can scale  $(w, b)$  such that the resulting functional margin is 1, i.e., for instances  $x_+$  and  $x_-$ , we have

$$w^T x_+ + b = 1, \quad w^T x_- + b = -1.$$

The geometric margin is then computed by

$$\begin{aligned} \gamma &= \frac{1}{2} \left( \frac{w^T x_+}{\|w\|} - \frac{w^T x_-}{\|w\|} \right) \\ &= \frac{1}{2 \|w\|} w^T (x_+ - x_-) = \frac{1}{\|w\|}. \end{aligned}$$

**Conclusion:** Maximizing  $\gamma$  equals to minimize the norm of  $w$ !

# Line Support Vector Machine

---

**Theorem:** *Given a linearly separable training example  $S = (x_1, y_1), \dots, (x_l, y_l)$ . The hyperplane  $(w, b)$  that solves the following optimization problem*

$$\begin{aligned} \min_{w, b} \quad & w^T w \\ \text{S.T.} \quad & y_i(w^T x_i + b) \geq 1, \forall 1 \leq i \leq l \end{aligned}$$

*gives the maximal margin hyperplane with geometric margin  $\gamma = 1 / \|w\|$ .*

This model was introduced by Mangasarian, a computer scientist from Wisconsin, in the 1960s, and rediscovered and extensively studied by Vapnik in the 1990s, and many other researchers.

# The dual formulation

---

By using the duality theory in optimization, we can get the dual of the above problem.

**Theorem:** *Given a linearly separable training example  $S = \{(x_1, y_1), \dots, (x_l, y_l)\}$  and suppose that the parameter  $\alpha^*$  solves the following quadratic optimization problem*

$$\begin{aligned} \max \quad & W(\alpha) = \sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i,j=1}^l y_i y_j \alpha_i \alpha_j x_i^T x_j \\ \text{S.T.} \quad & \sum_{i=1}^l y_i \alpha_i = 0, \\ & \alpha_i \geq 0, i = 1, \dots, l. \end{aligned}$$

*Then the weight vector  $w^* = \sum_{i=1}^l y_i \alpha_i^* x_i$  yields the maximal margin hyperplane with geometric  $\gamma = 1 / \|w^*\|$ .*

The dual model use clearly the information about the kernel space, which allows us to find a suitable kernel space for a given data set.

# Linear SVMs:I

---

The instances closest to the maximum margin hyperplane are called **support vectors**.

**Important observation:** the support vectors define the maximum margin hyperplane!

- All other instances can be deleted without changing the position and orientation of the hyperplane!

We need only to know the set of active constraints at the optimal solution, i.e.,  $(\alpha_i^* > 0)$ , and ignore the inactive constraints  $(\alpha_i^* = 0)$ .

Consider the same problem of classifying

$$S_1 = \{(1, 2), (2, 1)\}; \quad S_2 = \{(1, -1), (0, 3)\}$$

We can ignore the point  $(2, 1)$ . (Why?)

The unique classifier found by SVM is

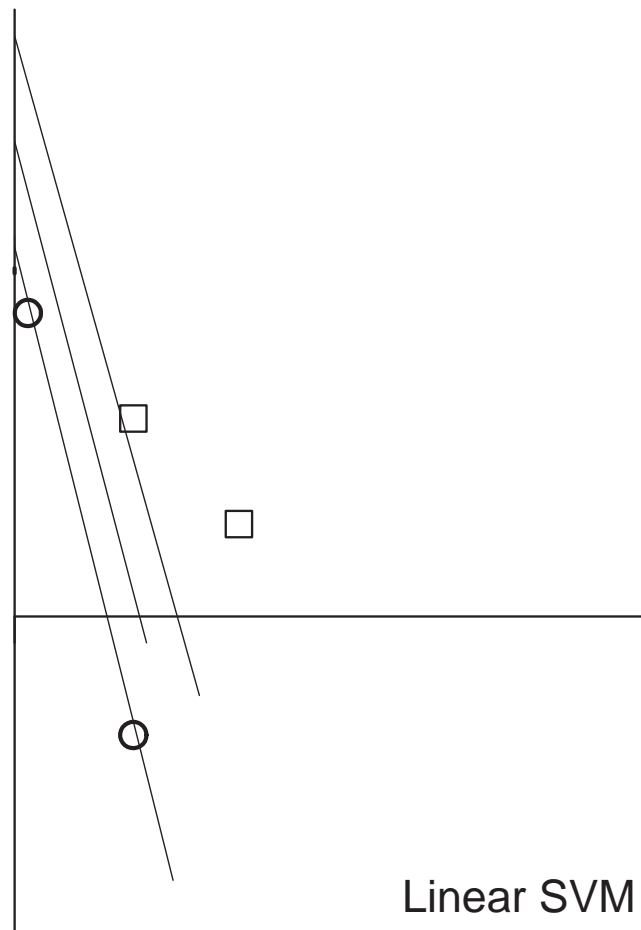
$$4x_1 + x_2 - \frac{9}{2}.$$

# Linear SVMs:II

---

Standard optimization tools play a very important role in linear SVMs and most involved problems can be solved efficiently.

Linear Classifiers can be used for multi-classes classification by using pair-wise classifiers and voting.



# Nonlinear numeric models

---

Linear classifiers can not model nonlinear class boundaries

Simple tricks to find nonlinear boundaries:

- Map attributes into new space consisting of combinations of attribute values
- E.g., all products in order  $n$  that constructed from the attributes, i.e.,

$$y = w_1 a_1^3 + w_2 a_1^2 a_2 + w_3 a_1 a_2^2 + w_4 a_2^3.$$

**Problems** with this approach:

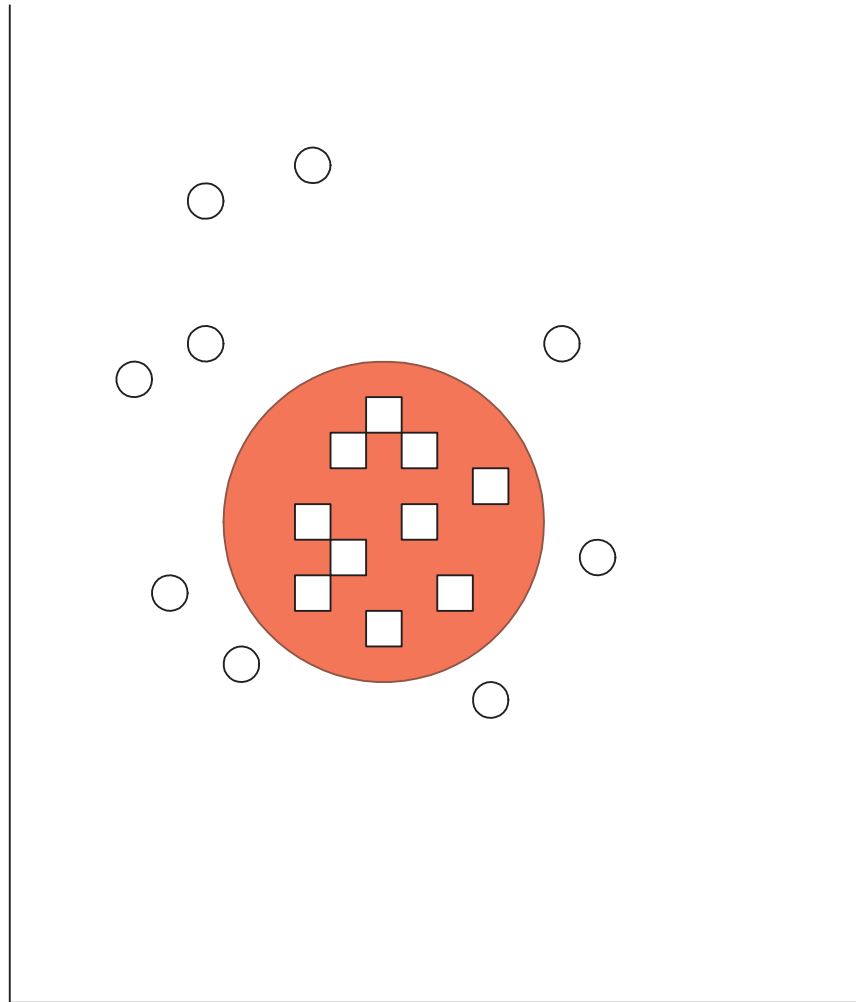
- High computational cost: with 10 attributes and  $n = 5$ , more than 2000 coefficients are needed
- Linear regression (with attribute selection) running time is cubic in the number of attributes
- Overfitting: Number of coefficients is too large compared with the number of training instances
- Curse of dimensionality kicks in

Linear SVMs don't suffer from those issues!

---

# Figure: nonlinear separation

---



Nonlinear SVM

# Nonlinear SVMs

---

Similar ideas as in linear case can be applied.

- Employ a clever mathematical trick to avoid the creation of "pseudo-attributes";
- Create the nonlinear space implicitly.

Recall the dual problem for SVM:

$$\begin{aligned} \max \quad & W(\alpha) = e^T \alpha - \frac{1}{2} \alpha^T Y X^T X Y \alpha \\ \text{S.T.} \quad & y^T \alpha = 0, \\ & \alpha \geq 0, \alpha = (\alpha_1, \dots, \alpha_l)^T. \end{aligned}$$

Here  $e = (1, \dots, 1)^T$ ,  $Y = \text{diag}(y)$ ,  $y = (y_1, \dots, y_l)^T$  and  $X = (x_1, \dots, x_l) \in \mathbb{R}^{n \times l}$ . The matrix  $X^T X$  is called **the kernel matrix**.

**A mathematical trick** is to compute the inner product before the nonlinear mapping is performed.

**Example:**

$$y = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i (x_i^T x)^n$$

Mapping the instance into the space spanned by all products of  $n$  attributes.

# Other kernel functions

---

The mapping is performed by the kernel function. We can use other kernel functions

$$y = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i K(x_i, x),$$

## Examples:

$$K(x_i, x_j) = (x_i^T x_j + 1)^d, K(x_i, x_j) = \exp^{-\frac{(x_i - x_j)^2}{2\sigma^2}}.$$

This will yield a new kernel matrix  $K$  whose element is defined by  $K_{i,j} = K(x_i, x_j)$ !

- Particular case  $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$

To find a nonlinear model, we first compute the kernel matrix  $K$  and then substitute it into the following problem

$$\begin{aligned} \max \quad & W(\alpha) = e^T \alpha - \frac{1}{2} \alpha^T Y K Y \alpha \\ \text{S.T.} \quad & y^T \alpha = 0, \\ & \alpha \geq 0, \alpha = (\alpha_1, \dots, \alpha_l)^T. \end{aligned}$$

We end up with a similar quadratic optimization problem as in the linear case!

# SVMs for noisy data

---

SVMs can be applied to noisy data by introducing a penalty parameter  $\rho$  to bound the influence of any training instance on the decision boundary.

$$\begin{aligned} \min_{w,b} \quad & w^T w + \rho \sum_{i=1}^l \epsilon_i \\ \text{S.T.} \quad & y_i(w^T x_i + b) - \epsilon_i \geq 1, \quad 1 \leq i \leq l \\ & \epsilon_i \geq 0. \end{aligned}$$

The parameter  $\rho$  has an important role in the model. It has been shown that for data set with a certain distribution,  $\rho$  and the number of support vectors can provide a bound of total misclassification errors. However, a very large  $\rho$  might lead to overfitting, while a small  $\rho$  can not guarantee a high accuracy even for the training data. A suitable value for  $\rho$  needs to be identified by experimentation.

# Instance-based learning

---

Practical problems of the lazy 1-NN scheme:

- Too slow to test a new instance for large training set.

- Remedy: removing irrelevant data.

- Poor performance for noisy data

- Remedy: removing noisy instances or using k-NN

- All attributes deemed equally important

- Remedy: attribute weighting. How?

- Does not perform explicit generalization.

No meaningful pattern is extracted.

- Remedy: rule-based NN approach

**Edited NN classifiers** discard some of the training instances before making predictions to save memory and speeds-up classification.

**Note:** Important instances might be discarded too early.

# Dealing with noise:I

---

**IB2:** incremental NN learner that incorporates misclassified instances into the classifier, however this implies noisy data gets incorporated.

**An expensive way:** cross-validation-based k-NN classifier, and assign the majority class to the unknown instance.

**Different approach:** discard instances that don't perform well and keep only successful ones (IB3)

- Use thresholds for instance's success rate.
- When an exemplar's performance is below the lower bound, remove it.
- When its performance exceeds the upper bound, add it to the exemplar set.
- For exemplar whose performance is between two thresholds, it is used only for the new instance that is closest to it and its success ratio should be updated after that.

# Dealing with noise:II

---

- Set confidence levels;
- Estimate the success bounds of an exemplar at a certain confidence level ( $s/n$ ):  $s$  successes in  $n$  trials;
- Estimate the bounds for default success rate:  $c$  occurrence of exemplar's class out of  $N$  instances, with a certain confidence level;
- If the lower bound of an exemplar's success rate exceeds the upper bound of the default success rate, the exemplar is accepted;
- An exemplar is rejected if the upper confidence bound of its success rate lies below the lower bound on the default success rate.

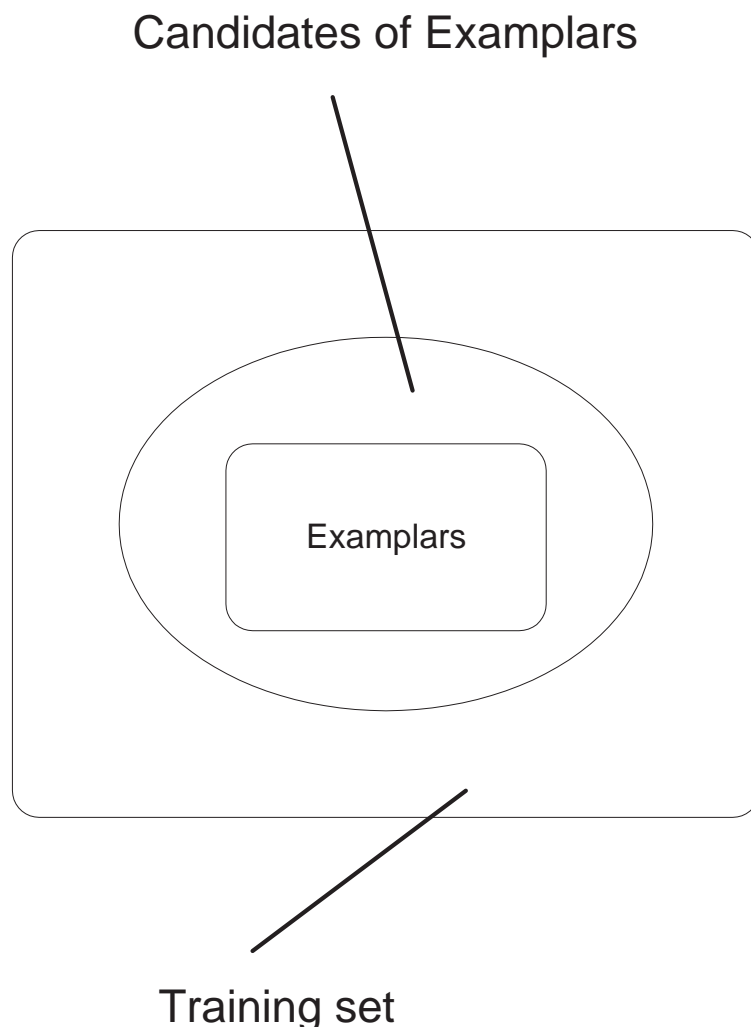
In IB3, a confidence level 5% is used for acceptance, and 12.5% for rejection.

Quite stringent in acceptance, but a bit loose for dropping exemplars, because a dropped exemplar might be replaced by a similar instance later on.

# Exemplars

---

**Example:** The default rate 75%, the upper bound for acceptance 85%. If the success rate of an exemplar is 85%, it will be placed in the candidate set. If its success rate is 95% and its lower bound based on the confidence level exceeds 85%, then accept it.



# Weighting attributes

---

**Problems:** some attributes are more important than others

**Remedy:** attribute selection and weighting

- Class-specific weights: use dif. weights for different classes, might lead to unclassified cases and multiple classifications.

Based 1-NN, we can update the weighted distance as follows

$$\left( \sum_{i=1}^n w_i (x_i - y_i)^2 \right)^{\frac{1}{2}}$$

- Class predicted correctly/ incorrectly  $\Rightarrow$  Associated weights increase/decreased
- $|x_i - y_i|$ : a measure of the contribution to the decision; The smaller the value is, more important its contribution is;
- Updating Schemes:  
amount ( $|x_i - y_i|$ ) large/small  $\iff$  weight ( $w_i$ ) change small/large.

# Weighted distance: Examples

---

**Example:** A training instance (1,2,2,'yes'),  
a new instance (2,2,2,'no')

Default distance function (all  $w_i = 1$ ),

**Constraint:**  $w_1 + w_2 + w_3 = 3$ ;

- 1-NN predicts incorrectly

Update: associated weights  $w_2, w_3$  are reduced to  $\frac{1}{2}$ , while  $w_1$  is increased to 2.

- another instance (1,2,1,'yes')

1-NN predicts correctly

Update: associated weights  $w_1, w_2$  are increased from 1 to 1.25, while  $w_3$  is reduced to  $\frac{1}{2}$ .

- There are many other ways for updating weights.

# Generalized exemplars

---

To use rectangles as exemplars, we define

**The distance** between an instance and an exemplar hyperrectangle is given by the distance from the instance to the nearest part of the hyperrectangle.

## Online version

- Using the existing rectangle to predict the new instance;
- If it classifies the instance correctly, then merge the new instance into the rectangle;
- If misclassified, then shrink it.

**Off-line version** tries to find small set of rectangles covering given set of instances

Important design decisions:

- Overlapping rectangles allowed?
  - An instance can be classified via several different rules and conflicting solutions might appear.
- Nested rectangles allowed for exceptions.

# Numeric prediction

---

All schemes discussed so far can be adapted for numeric prediction!

- Decision trees, rule learners, SVMs, etc.

**Regression tree:** each leaf represents the average value of the instances in that leaf;

**Model tree:** each leaf contains a model to predict the value of the instance.

## Regression VS Decision Trees

Splitting minimizing intra-subset variation

Pruning based on numeric error measure

Output Leaf node predicts 'average' class values of training instances reaching that node

Can approximate piecewise constant functions

Easy to interpret

# Smoothing Model Trees

---

Each leaf gives a linear model for prediction.

**Issue:** Sharp discontinuity among models at various nodes, in particular when training set is limited.

**Smoothing:** Naive method to reduce this gap and improve the accuracy

- Predicted value is weighted average of LR models along path from root to leaf
- Bottom-up Smoothing Formula:

$$p' = \frac{np + kq}{n + k},$$

$k$ : the smoothing constant,

$q$ : the predicted value at this level,

$p$ : the value passed to this node from below,

$n$ : the size of the subset in that branch,

$p'$ : the value passed to the next higher node.

# Building the tree

---

**Splitting criterion:** standard deviation reduction

$$SDR = sd(T) - \sum_i \frac{T_i}{T} sdr(T_i), \quad T = \sum_i T_i.$$

The attribute with maximal expected error reduction is selected.

**Termination criteria:** Standard deviation becomes smaller than certain fraction of sd for full training set (e.g. 5%)

- Prerequisite in minimal coverage to avoid overfitting

**Pruning** is based on estimated absolute error of LR models

**Heuristic estimate:** let  $v$  be the number of parameters in LR and  $n$  the number of instances reached that branch

$$\frac{n + v}{n - v} \text{average}_{error}$$

LR models are pruned by greedily removing terms to minimize the estimated error

Pruning proceeds bottom-up: error for LR model at internal node is compared to error for subtree

# Nominal attributes

---

Converting nominal attributes into binary

- Sort nominal values by its average class value
- For  $k$  values,  $k - 1$  binary attributes are generated
- The  $i$ -th binary attribute is 0 if an instance's value is one of the first  $i$ 's in the ordering, 1 otherwise

**Observation:** The best split on one of the new attributes is the best binary split on original

**Missing values** Modify splitting criterion:

$$SDR = \frac{m}{T} [sd(T) - \sum_j \frac{T_j}{T} sdr(T_j)],$$

where  $m$  is the number of instances without missing value.

# Surrogate Splitting

---

**Surrogate splitting:** Choose an attribute for splitting that is most highly correlated with original attribute

- Use only complete instances to select split point
- Separate all the instances into two subsets L and R based on the splitting point
  - Assume L has smaller average class value than R
  - Let  $m$  be the average of the two averages
  - If an instance with a missing value has class value smaller than  $m$  it goes into L, otherwise into R
- After full tree has been built, missing values are replaced by average values from corresponding leaf nodes

**Problem:** complex and time-consuming

**Simple Alternative:** always use the class

**Testing:** replace missing value with average

# M5'

---

The resulting algorithm for model tree is called M5', which consists of four parts:

- Main method: `MakeModelTree()`
- Splitting: `split()`
- Pruning: `prune()`
- Computing error: `subtreeError()`

Linear regression is the core.

# M5'

---

## **MakeModelTree (instances)**

```
{  $SD = sd(\text{instances})$   
for each  $k$ -valued nominal attribute  
convert into  $k - 1$  synthetic binary attributes  
 $root = \text{newNode}$   
 $root.\text{instances} = \text{instances}$   $\text{split}(root)$   
 $\text{prune}(root)$   
 $\text{printTree}(root)$  }
```

## **Split(node)**

```
{ if  $\text{sizeof}(\text{node}.\text{instances}) < 4$  or  
   $sd(\text{node}.\text{instances}) < 0.05SD$   
   $\text{node}.\text{type} = \text{LEAF}$   
else  
   $\text{node}.\text{type} = \text{INTERIOR}$   
for each attribute  
  for all possible split positions of the at-  
  tribute  
    calculate the attributes SDR  
   $\text{node}.\text{attribute} = \text{attribute with maximum}$   
SDR  
   $\text{split}(\text{node}.\text{left})$   
   $\text{split}(\text{node}.\text{right})$  }
```

# M5'

---

## **Prune(node)**

```
{ if node = INTERIOR then
  prune(node.leftChild)
  prune(node.rightChild)
  node.model = linearRegression(node)
if subtreeError(node) > error(node) then
  node.type = LEAF }
```

## **SubtreeError(node)**

```
{ l = node.left; r = node.right
if node = INTERIOR then
  return (sizeof(l.instances)*subtreeError(l)
  + sizeof(r.instances)*subtreeError(r))
  /sizeof(node.instances)
else return error(node) }
```

# Locally weighted regression

---

Numeric prediction using **instance-based learning** combined with **linear regression**

Lazy learning scheme: linear regression function is computed at prediction time

Training instances are weighted according to distance to test instance

- Requires a weighted version of linear regression

Advantage: nonlinear approximation, incremental

Disadvantage: too slow, biased to testing set

**Types** of weighting function:

- Inverse of Euclidean distance
- Gaussian kernel applied to Euclidean distance
  - Distance is multiplied by inverse of a smoothing parameter
  - Possible choice: distance of kth nearest training instance (makes it data dependent)

# Clustering

---

**Unsupervised Learning:** no target value to be predicted

Differences between models/ algorithms:

- Exclusive vs. Overlapping
- Deterministic vs. Probabilistic
- Hierarchical vs. Flat
- Incremental vs. Batch learning

The clustering tools will decide what we can learn.

**Evaluation:** very hard, usually done by inspection

- If clustering is treated as a density estimation problem, then it can be evaluated on test data!

We discuss

- Distance-based clustering;
- Incremental Clustering;
- Statistical Clustering.

# The k-means clustering

---

The K-means algorithm clusters the data into  $k$  groups where  $k$  is predefined

**Step 1:** Choose cluster centers at random;

**Step 2:** Assign instances to clusters based on their distance to the cluster centers

**Step 3:** centers of clusters are adjusted

**Step 4:** go to Step 1 until convergence

**Notes:** Initial choice of seeds might produce an important role in the final result.

Algorithm can get trapped in a local minimum

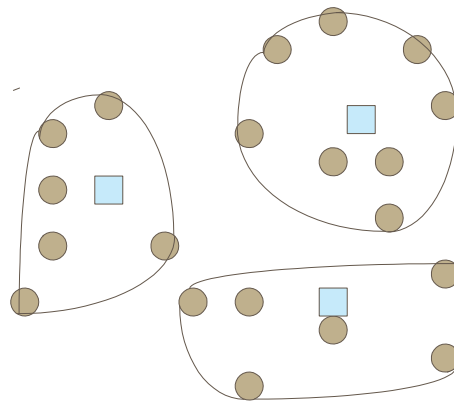
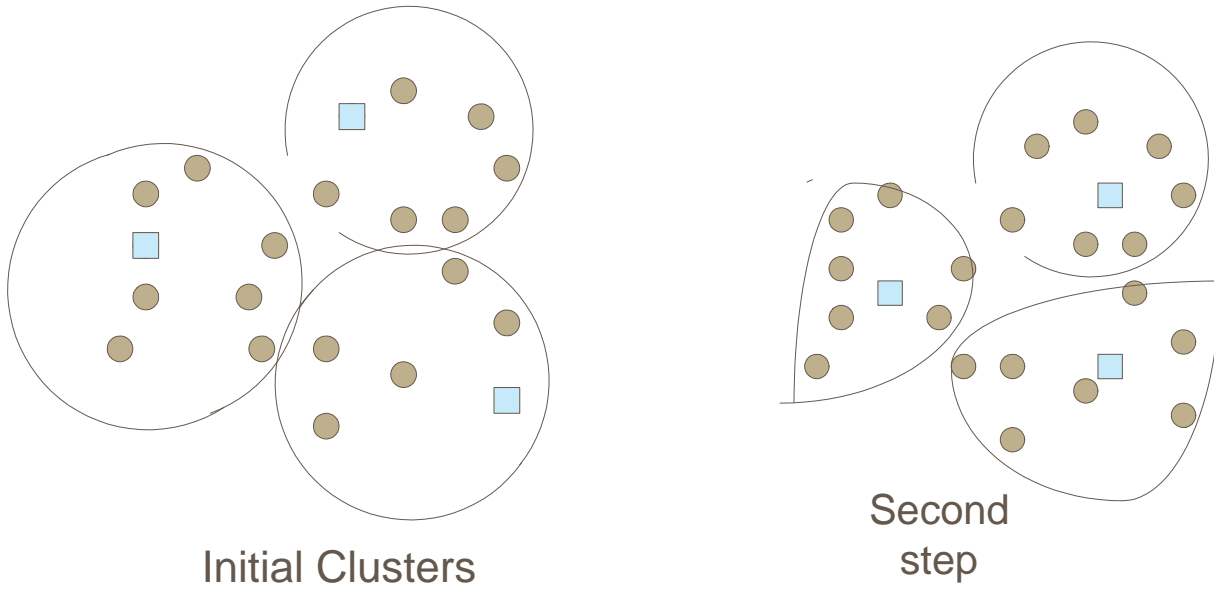
- Example: four instances at the vertices of a two-dimensional rectangle

- Local minimum: two cluster centers at the midpoints of the rectangles long sides

A simple way to increase chance of finding a global optimum: restart with different random seeds

# K-means Clustering

---



The final clusters

# K-medoids method

---

**The medoid** is the most centrally located instance in a cluster.

## **Algorithm:**

**Step 1:** Choose  $k$  medoids at random;

**Step 2:** Repeat

- Assign instances to clusters based on their distance to the cluster medoids
- Randomly select a nonmedoid object  $O_{random}$ ,
- Compute the cost  $S$  of, swapping  $O_j$  and  $O_{random}$ ;
- Update the set of Medoids if  $S < 0$ ;

**Step 3:** go to Step 2 until convergence

Once the random medoid is selected, we need to reassign the instances based on the new set of medoids, and then see whether this new medoid can improve the the quality of clusters (based on the sum of dissimilarities between an instance and its reference medoid).

More robust but much more time-consuming than K-means method.

# Bottom-up clustering

---

**Idea:** Merge small clusters into large ones

**Measurements:** Average distance

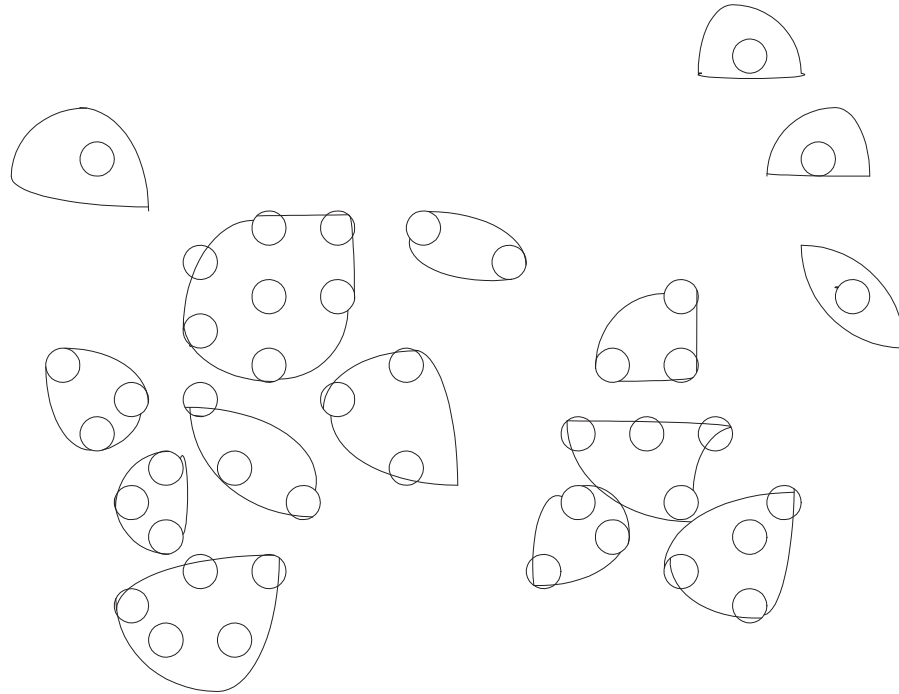
$$d_{avg}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{p \in C_i, q \in C_j} \|p - q\|,$$

where  $|C_i|$  is the number of instances in cluster  $C_i$ . Other measurements can be used.

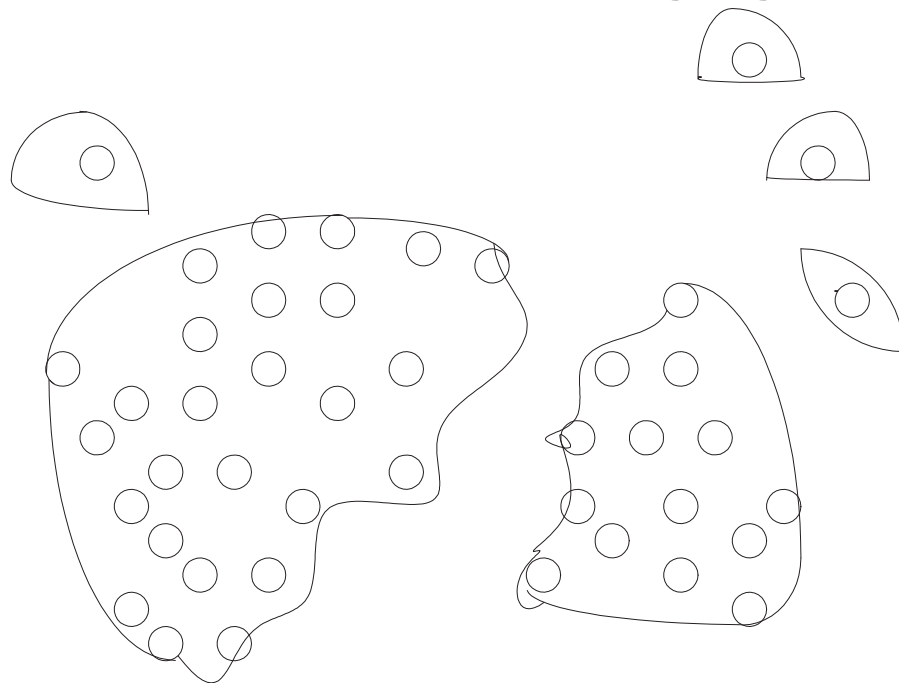
First partition all the instances into small groups, and merge them based on some measurements. Clusters with few instances, but no improvement in the process can be removed (treated as outliers).

# Bottom-up clustering

---



**Clusters after merging**



# Incremental clustering: COBWEB

---

Forming a hierarchy of clusters step by step

- Starts with an empty-root tree;
- Instances are added one by one, and the tree is updated correspondingly
  - Need to find the right host-leaf for an instance or restructuring the tree;

Updating Criteria: **Category utility**, a loss function based on conditional probabilities:

$$CU(C_1, \dots, C_k) = \frac{\sum_l Pr[C_l] \sum_{i,j} (Pr[a_i = v_{ij}|C_l]^2 - Pr[a_i = v_{ij}]^2)}{k},$$

where  $C_l$  are clusters,  $v_{ij}$  are class values, and  $a_i$  is the attribute.

- A cluster is useful if it predicts the distribution of the attribute's values better;
- $Pr[a_i = v_{ij}|C_l]$  is a better probability estimation than  $Pr[a_i = v_{ij}]$ .

If every instance gets put into a different category the numerator becomes

$$n - \sum_i \sum_j Pr[a_i = v_{ij}]^2,$$

where  $n$  is the total number of values that the set of attributes can assume.

# COBWEB

---

For numerical attributes, we use:

$$f(a) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(a-\mu)^2}{2\sigma^2}}$$
$$\sum_j Pr[a_i = v_{ij}]^2 = \int f(a_i)^2 da_i = \frac{1}{\sqrt{2\pi}\sigma}.$$

Therefore, the category utility becomes

$$CU = \frac{\sum_l Pr[C_l] \frac{1}{\sqrt{2\pi}} \sum_i (\frac{1}{\sigma_{il}} - \frac{1}{\sigma_i})}{k}.$$

Zero standard deviation must be avoided(impose minimum variance)!

## Procedure of COBWEB

---

**Step 0:** First establish k-clusters (hosts);

**Repeat** for all new instances;

- Select the best host among the existing hosts and the runner-up;
- Merging the best host and the runner-up into a new candidate cluster;
- Comparing the CUs obtained by adding the new instance to best host, or to the candidate cluster, and select the cluster with a larger CU;
- If merging does not help, consider splitting the best host(if it allows).

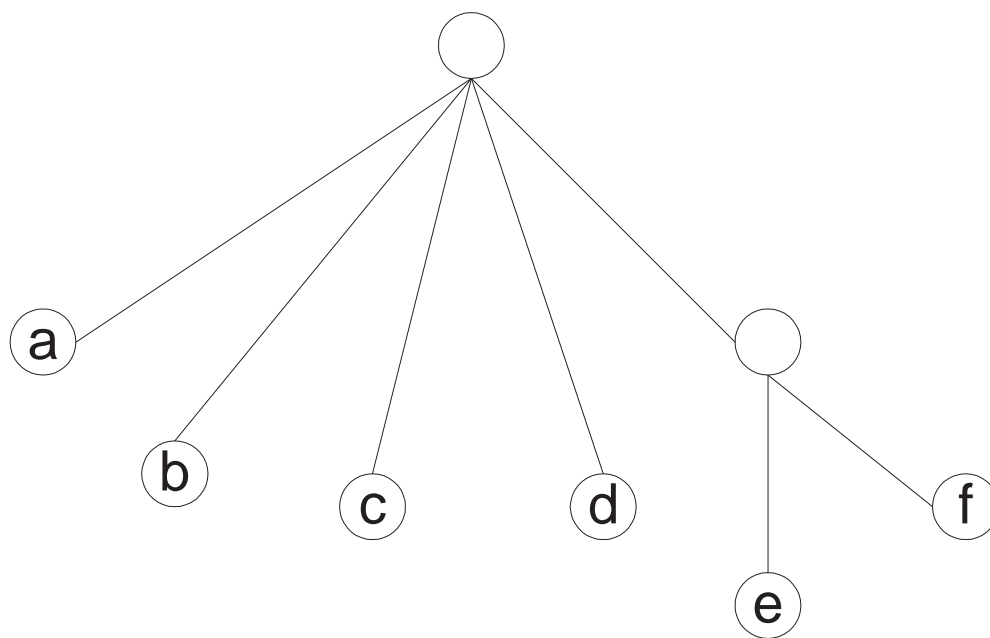
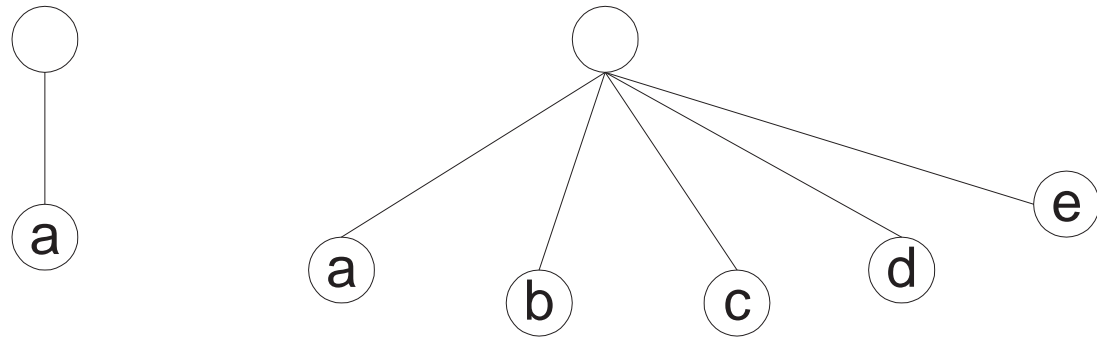
# Sample Question

---

Outlook	Temper.	humidity	windy	ID
Sunny	Hot	High	False	a
Sunny	Hot	High	True	b
Overcast	Hot	High	False	c
Rainy	Mild	High	False	d
Rainy	Cool	Normal	False	e
Rainy	Cool	Normal	True	f
Overcast	Cool	Normal	True	g
Sunny	Mild	High	False	h
Sunny	Cool	Normal	False	i
Rainy	Mild	Normal	False	j
Sunny	Mild	Normal	True	k
Overcast	Mild	High	True	l
Overcast	Hot	Normal	False	m
Rainy	Mild	High	True	n

# Clustering the weather problem

---

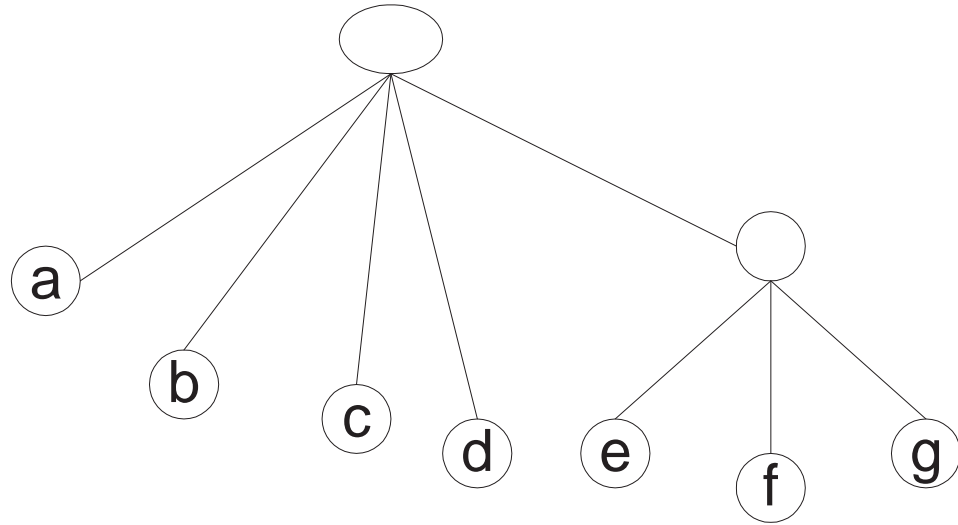


Step 1--3

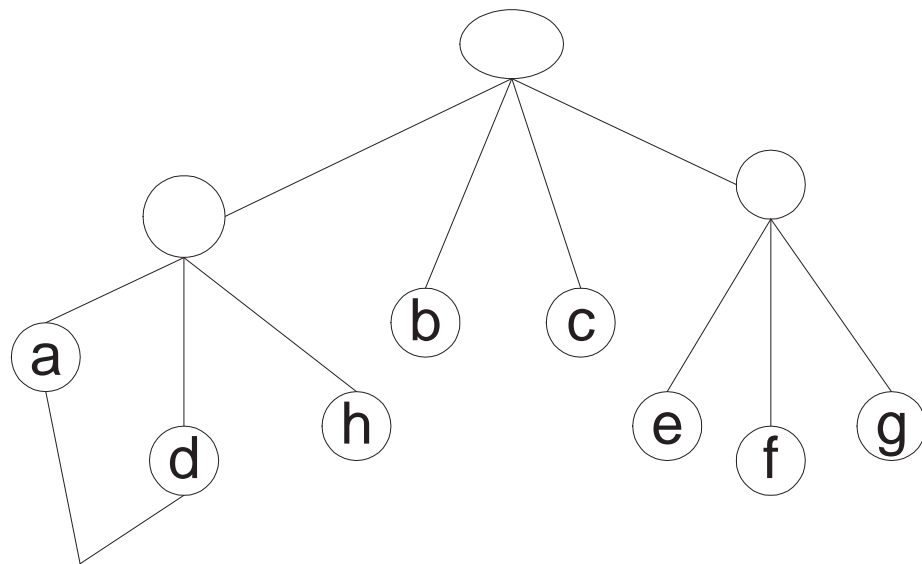
**Cobweb for the weather problem**

# Clustering the weather problem

---



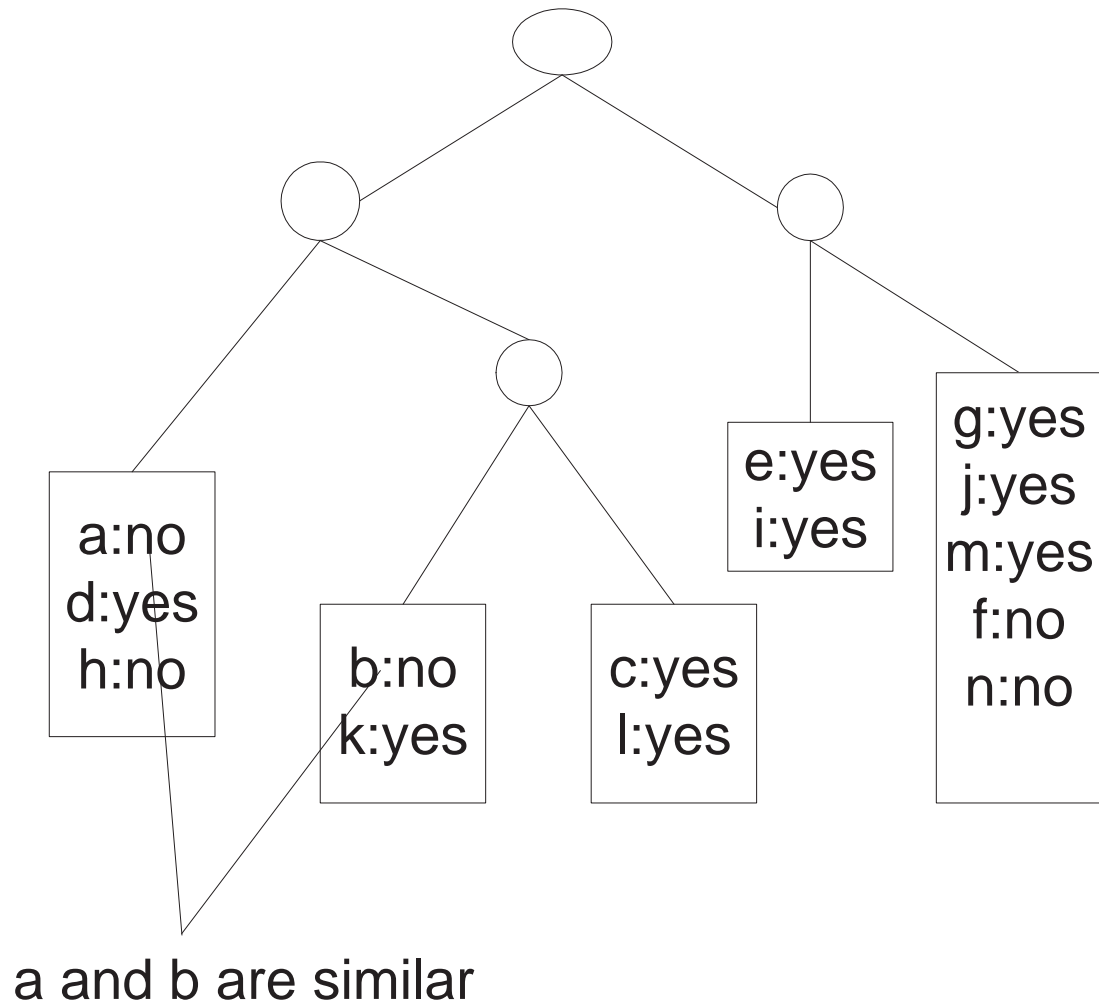
Step 4--5



The best host and runner-up  
are merged

# Clustering weather problem

---



# Probability-based clustering

---

**Problems** in heuristic approach:

- Division by  $k$ ?
- Order of instances?
- Is result at least locally optimal?

From a probabilistic perspective, we want to find the most likely clusters given the data

- Instance only has certain probability of belonging to a particular cluster

Probabilistic clustering algorithms model the data using a mixture of distributions:

- Each cluster is represented by one distribution
  - The distribution governs the probabilities of attributes values in the corresponding cluster

They are called **finite mixtures** because there is only a finite number of clusters being represented

- Individual distributions are normal distributions;
- Distributions are combined using cluster weights

# Probabilistic Clustering

---

The probability of an instance  $x$  belonging to cluster  $A$  is:

$$Pr[A|x] = \frac{Pr[x|A]Pr[A]}{Pr[x]}$$

For numeric attribute  $x$ , we have

$$Pr[x|A] = f(x, \mu_A, \sigma_A) = \frac{1}{\sqrt{2\pi}\sigma_A} e^{-\frac{(x-\mu_A)^2}{2\sigma_A^2}}.$$

The total likelihood of an instance for given the clusters is:

$$Pr[x|\text{the distributions}] = \sum_i Pr[x|C_i]Pr[C_i].$$

A simple case with one numerical attribute and two unknown clusters  $A$  and  $B$ :

- Involved parameters: mean  $\mu_A, \mu_B$ , deviation  $\sigma_A, \sigma_B$ , and the probabilities  $P_A, P_B$  with  $P_A + P_B = 1$ .

Learning the clusters  $\Leftrightarrow$  Finding the means and deviations of clusters

# Learning the clusters

---

**Expectation-Maximization Algorithm:** generalizing k-means to probabilistic setting

**Performance criterion:** the likelihood of the training data given the clusters

- Finding a maximum of the likelihood

**Iterative procedure:**

1. Guess the initial model parameters:

$$\mu_A^0, \sigma_A^0, P_A^0, \mu_B^0, \sigma_B^0, \text{ and } P_B^0, \epsilon;$$

2. At each iteration  $j$ : calculate the probabilities of  $x$  in  $A$  and  $B$

$$Pr[A|x] = \frac{Pr^j[x|A]P_A^j}{Pr^j[x]}, Pr^j[B|x] = \frac{Pr^j[x|B]P_B^j}{Pr^j[x]}$$

3. Update the mixture parameters on the basis of the new estimate:

$$\begin{aligned} P_A^{j+1} &= \frac{1}{n} \sum_x Pr[A|x], & P_B^{j+1} &= \frac{1}{n} \sum_x Pr[B|x]; \\ \mu_A^{j+1} &= \frac{\sum_x x Pr[A|x]}{\sum_x Pr[A|x]}, & \mu_B^{j+1} &= \frac{\sum_x x Pr[B|x]}{\sum_x Pr[B|x]}; \\ \sigma_A^{j+1} &= \frac{\sum_x P[A|x](x - \mu_A^{j+1})^2}{\sum_x P[A|x]}, & \sigma_B^{j+1} &= \frac{\sum_x P[B|x](x - \mu_B^{j+1})^2}{\sum_x P[B|x]}; \end{aligned}$$

4. Compute the log estimate  $E_j = \sum_x \log(Pr^j(x))$ . Stop if  $|E_j - E_{j+1}| \leq \epsilon$ ; Otherwise set  $j = j + 1$  and go to Step 2.

# Minimizing the likelihood

---

**Warning:** The simple EM method is not spatial sensitive!

Note that the total likelihood of an instance  $x_i$  for given clusters  $A$  and  $B$  is

$$p_A Pr[x_i|A] + p_B Pr[x_i|B].$$

The overall Log-likelihood for this distribution is given by

$$\sum_i \log(p_A Pr[x_i|A] + p_B Pr[x_i|B]).$$

Our target is to minimize this objective!

**Extensions:** use more than two distributions

- Easy for independent attributes
- Difficult for correlated attributes:
  - Modelled jointly using a bivariate normal distribution with a covariance matrix
  - $n$  attributes requires estimating  $\frac{n(n+1)}{2}$  parameters

**Nominal attributes:** easy if independent

# Discussion

---

Extension to various cases is possible, although more complex.

Clusters can be interpreted by using supervised learning in a post-processing step

Can be used to fill in missing values

May be advantageous to make attributes more independent in pre-processing step

- I.e. using principal component analysis

Big advantage of probabilistic clustering schemes:

- Likelihood of data can be estimated and used to compare different clustering models