Learning from Observations –Instance Based Learning

Instance-Based Learning

- Distance function defines what's learned
- Most instance-based schemes use *Euclidean distance*:

$$\sqrt{(a_1^{(1)}-a_1^{(2)})^2+(a_2^{(1)}-a_2^{(2)})^2+\ldots(a_k^{(1)}-a_k^{(2)})^2}$$

$\mathbf{a}^{(1)}$ and $\mathbf{a}^{(2)}$: two instances with k attributes

- Taking the square root is not required when comparing distances
- Other popular metric: *city-block metric*
  - Adds differences without squaring them

Normalization and Other Issues

- Different attributes are measured on different scales ⇒ need to be *normalized*:

$$a_i = \frac{v_i - minv_i}{maxv_i - minv_i}$$

  v_i : the actual value of attribute *i*

- Nominal attributes: distance either 0 or 1
- Common policy for missing values: assumed to be maximally distant (given normalized attributes)

Finding Nearest Neighbors Efficiently

- Simplest way of finding nearest neighbor: linear scan of the data
  - Classification takes time proportional to the product of the number of instances in training and test sets
- Nearest-neighbor search can be done more efficiently using appropriate data structures

Discussion of Nearest-Neighbor Learning

- Often very accurate
- Assumes all attributes are equally important
  - Remedy: attribute selection or weights
- Possible remedies against noisy instances:
  - Take a majority vote over the *k* nearest neighbors
  - Removing noisy instances from dataset (difficult!)
- Statisticians have used *k*-NN since early 1950s
- If $n \to \infty$ and $k/n \to 0$, error approaches minimum

More Discussion

- Instead of storing all training instances, compress them into regions
- Simple technique (Voting Feature Intervals):
  - Construct intervals for each attribute
- Discretize numeric attributes
- Treat each value of a nominal attribute as an “interval”
  - Count number of times class occurs in interval
  - Prediction is generated by letting intervals vote (those that contain the test instance)

EXAMPLE

Temperature		Humidity		Wind		Play
45		10		50		Yes
-20		0		30		Yes
65		50		0		No

1. Normalize the data:
   new value = (original value – minimum value)/(max – min)

EXAMPLE

Temperature		Humidity		Wind		Play
45	0.765	10	0.2	50	1	Yes
-20	0	0	0	30	0.6	Yes
65	1	50	1	0	0	No

1. Normalize the data:
   new value = (original value – minimum value)/(max – min)

   So for Temperature:
   new = (45 - -20)/(65 - -20) = 0.765
   new = (-20 - -20)/(65 - -20) = 0
   new = (65 - -20)/(65 - -20) = 1

EXAMPLE

Temperature		Humidity		Wind		Play	Distance
45	0.765	10	0.2	50	1	Yes	
-20	0	0	0	30	0.6	Yes	
65	1	50	1	0	0	No	

Temperature		Humidity		Wind		Play
35	0.647	40	0.8	10	0.2	???

1. Normalize the data in the new case (so it's on the same scale as the instance data):
   new value = (original value – minimum value)/(max – min)

2. Calculate the distance of the new case from each of the old cases (we're assuming linear storage rather than some sort of tree storage here).

EXAMPLE

Temperature		Humidity		Wind		Play	Distance
45	0.765	10	0.2	50	1	Yes	1.007
-20	0	0	0	30	0.6	Yes	1.104
65	1	50	1	0	0	No	0.452

Temperature		Humidity		Wind		Play
35	0.647	40	0.8	10	0.2	???

2. Calculate the distance of the new case from each of the old.

$$d(1) = \sqrt{(0.647 - 0.765)^2 + (0.8 - 0.2)^2 + (0.2 - 1)^2} = 1.007$$

$$d(2) = \sqrt{(0.647 - 0)^2 + (0.8 - 0)^2 + (0.2 - 0.6)^2} = 1.104$$

$$d(3) = \sqrt{(0.647 - 1)^2 + (0.8 - 1)^2 + (0.2 - 0)^2} = 0.452$$

EXAMPLE

Temperature		Humidity		Wind		Play	Distance
45	0.765	10	0.2	50	1	Yes	1.007
-20	0	0	0	30	0.6	Yes	1.104
65	1	50	1	0	0	No	0.452

Temperature		Humidity		Wind		Play
35	0.647	40	0.8	10	0.2	???

3. Determine the nearest neighbor (the smallest distance).

   We can see that our current case is closest to the third example so we would use that prediction for play – that is, we would predict Play = No.

Instance-Based Learning

- Practical problems of 1-NN scheme:
  - Slow (but: fast tree-based approaches exist)
    - Remedy: remove irrelevant data
  - Noise (but: *k* -NN copes quite well with noise)
    - Remedy: remove noisy instances
  - All attributes deemed equally important
    - Remedy: weight attributes (or simply select)
  - Doesn’t perform explicit generalization
    - Remedy: rule-based NN approach

Learning Prototypes

- Only those instances involved in a decision need to be stored
- Noisy instances should be filtered out
- Idea: only use *prototypical* examples

Speed Up,
Combat Noise

- IB2: save memory, speed up classification
  - Work incrementally
  - Only incorporate misclassified instances
  - Problem: noisy data gets incorporated
- IB3: deal with noise
  - Discard instances that don’t perform well

Weight Attributes

- IB4: weight each attribute (weights can be class-specific)
- Weighted Euclidean distance:

$$\sqrt{w_1^2(x_1 - y_1)^2 + \cdots + w_n^2(x_n - y_n)^2}$$

- Update weights based on nearest neighbor
  - Class correct: increase weight
  - Class incorrect: decrease weight
  - Amount of change for *i* th attribute depends on $|x_i - y_i|$

Generalized Exemplars

- Generalize instances into *hyperrectangles*
  - Online: incrementally modify rectangles
  - Offline version: seek small set of rectangles that cover the instances
- Important design decisions:
  - Allow overlapping rectangles?
    - Requires conflict resolution
  - Allow nested rectangles?
  - Dealing with uncovered instances?

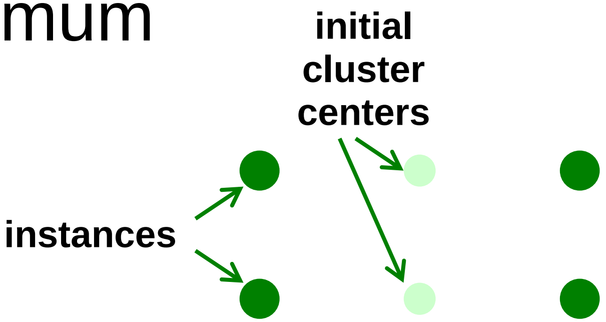
Learning from Observations – Clustering

Clustering

- Clustering techniques apply when there is no class to be predicted
- Aim: divide instances into “natural” groups
- Clusters can be:
    - Disjoint vs. overlapping
    - Deterministic vs. probabilistic
    - Flat vs. hierarchical
- We'll look at a classic clustering algorithm called *k-means*
    - *k-means* clusters are disjoint and deterministic

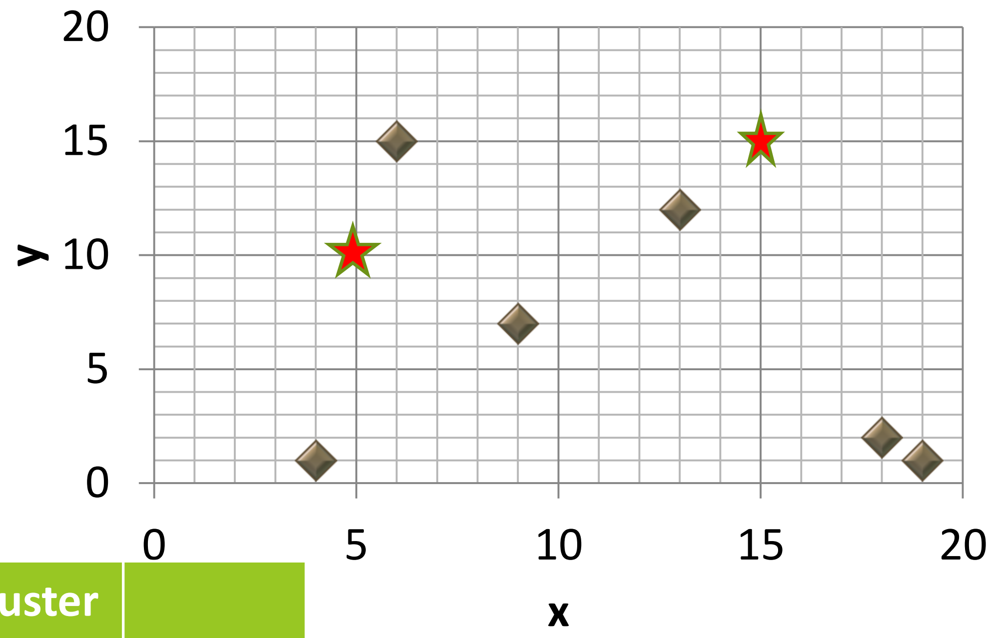
Discussion

- Algorithm minimizes distance to cluster centers
- Result can vary significantly
  - based on initial choice of seeds
- Can get trapped in local minimum
  - Example:



- To increase chance of finding global optimum: restart with different random seeds
- Can be applied recursively with $k = 2$

EXAMPLE



$$\sqrt{(a_1^{(1)}-a_1^{(2)})^2+(a_2^{(1)}-a_2^{(2)})^2+\dots(a_k^{(1)}-a_k^{(2)})^2}$$

Data		Cluster 1		Cluster 2	
X	Y	X=5	Y=10	X=15	Y=15
19	1				
13	12				
9	7				
6	15				
18	2				
4	1				

EXAMPLE

Data		Cluster 1		Cluster 2	
X	Y	X=5	Y=10	X=15	Y=15
19	1	16.64		14.56	
13	12	8.25		3.61	
9	7	5.00		10.00	
6	15	5.10		9.00	
18	2	15.26		13.34	
4	1	9.06		17.80	

$$d(1) = \sqrt{(19-5)^2+(1-10)^2} = 16.64$$

$$d(1) = \sqrt{(19-15)^2+(1-15)^2} = 14.56$$

EXAMPLE

Data		Cluster 1		Cluster 2	
X	Y	X=5	Y=10	X=15	Y=15
19	1	16.64		14.56	
13	12	8.25		3.61	
9	7	5.00		10.00	
6	15	5.10		9.00	
18	2	15.26		13.34	
4	1	9.06		17.80	

Now we assign each instance to the cluster which it's closest to (highlighted In the table.)

EXAMPLE

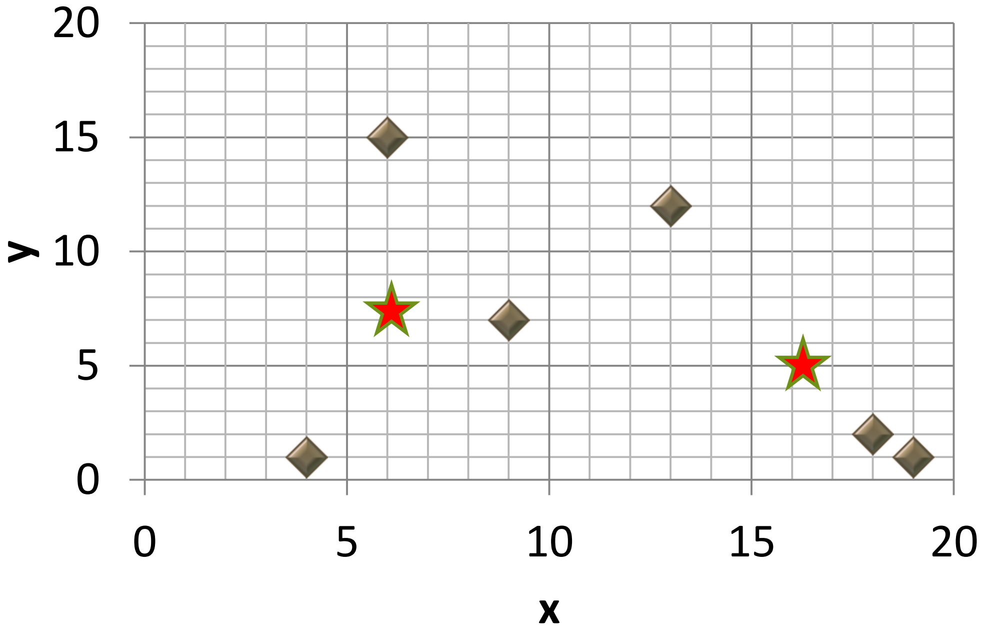
Data		Cluster 1		Cluster 2	
X	Y	X=5	Y=10	X=15	Y=15
19	1	16.64		14.56	
13	12	8.25		3.61	
9	7	5.00		10.00	
6	15	5.10		9.00	
18	2	15.26		13.34	
4	1	9.06		17.80	

Then we adjust the cluster centers to be the average of all of the instances assigned to them. (This is called the centroid.)

Cluster Center 1, X = (9+6+4)/3 = 6.33; Y = (7+15+1)/3 = 7.67

Cluster Center 2, X = (19+13+18)/3 = 16.67; Y = (1+12+2)/3 = 5

EXAMPLE



We place the new cluster centers and do the entire process again. We repeat this until no changes happen on an iteration.

Clustering: How Many Clusters?

- How to choose *k* in *k*-means? Possibilities:
  - Choose *k* that minimizes cross-validated squared distance to cluster centers
  - Use penalized squared distance on the training data (eg. using an MDL criterion)
  - Apply *k*-means recursively with *k* = 2 and use stopping criterion (eg. based on MDL)
    - Seeds for subclusters can be chosen by seeding along direction of greatest variance in cluster (one standard deviation away in each direction from cluster center of parent cluster)

Hierarchical Clustering

- Recursively splitting clusters produces a hierarchy that can be represented as a *dendogram*
  - Could also be represented as a Venn diagram of sets and subsets (without intersections)
  - Height of each node in the dendogram can be made proportional to the dissimilarity between its children

Agglomerative Clustering

- Bottom-up approach
- Simple algorithm
  - Requires a distance/similarity measure
  - Start by considering each instance to be a cluster
  - Find the two closest clusters and merge them
  - Continue merging until only one cluster is left
  - The record of mergings forms a hierarchical clustering structure – a *binary dendogram*

- *Single-linkage*
  - Minimum distance between the two clusters
  - Distance between the clusters closest two members
  - Can be sensitive to outliers
- *Complete-linkage*
  - Maximum distance between the two clusters
  - Two clusters are considered close only if all instances in their union are relatively similar
  - Also sensitive to outliers
  - Seeks compact clusters

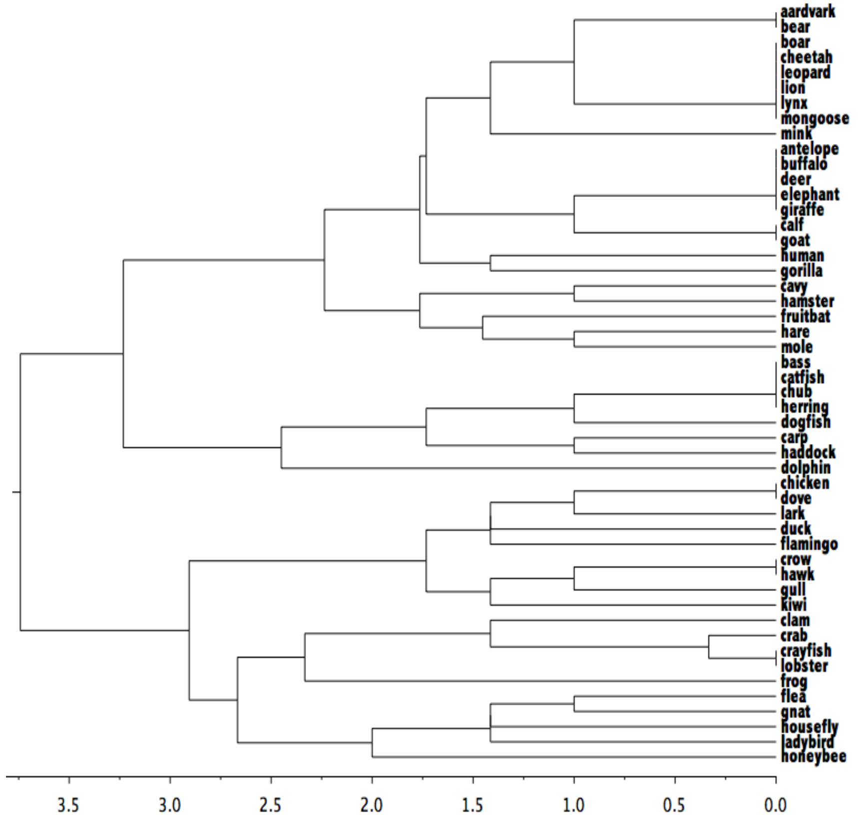
Distance Measures

Distance Measures (cont.)

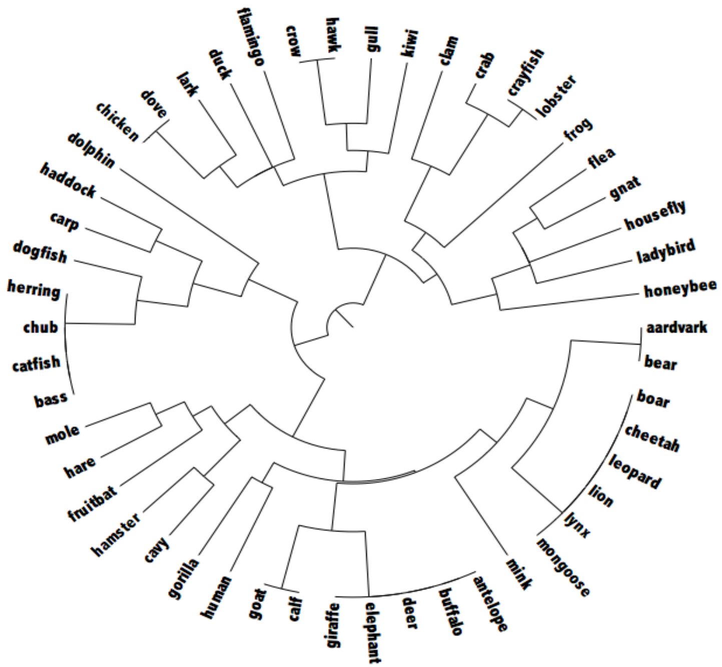
- Compromise between the extremes of minimum and maximum distance
- Represent clusters by their centroid, and use distance between centroids – *centroid linkage*
- Calculate average distance between each pair of members of the two clusters – *average-linkage*

Example Hierarchical Clustering

- 50 examples of different creatures from zoo data



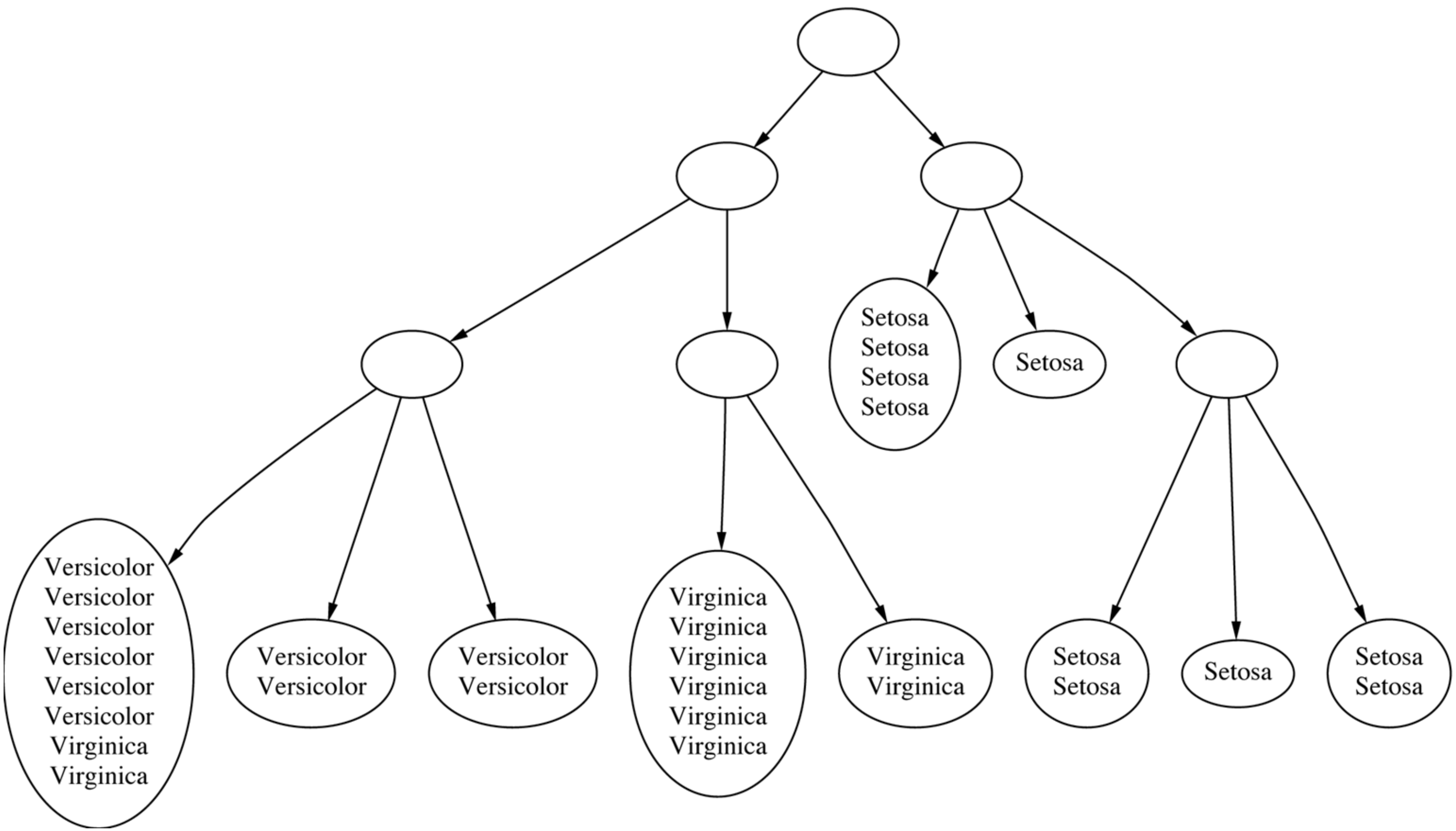
Dendogram



Polar Plot

Incremental Clustering

- Heuristic approach (COBWEB/CLASSIT)
- Form a hierarchy of clusters incrementally
- Start:
  - Tree consists of empty root node
- Then:
  - Add instances one by one
  - Update tree appropriately at each stage
  - To update, find the right leaf for an instance
  - May involve restructuring the tree
- Base update decisions on *category utility*


Setosa
Setosa
Setosa
Setosa
Setosa
Versicolor
Versicolor
Versicolor
Versicolor
Versicolor
Versicolor
Virginica
Virginica
Versicolor
Versicolor
Versicolor
Versicolor
Virginica
Virginica
Virginica
Virginica
Virginica
Virginica
Virginica
Virginica
Setosa
Setosa
Setosa
Setosa
Setosa

Probability-Based Clustering

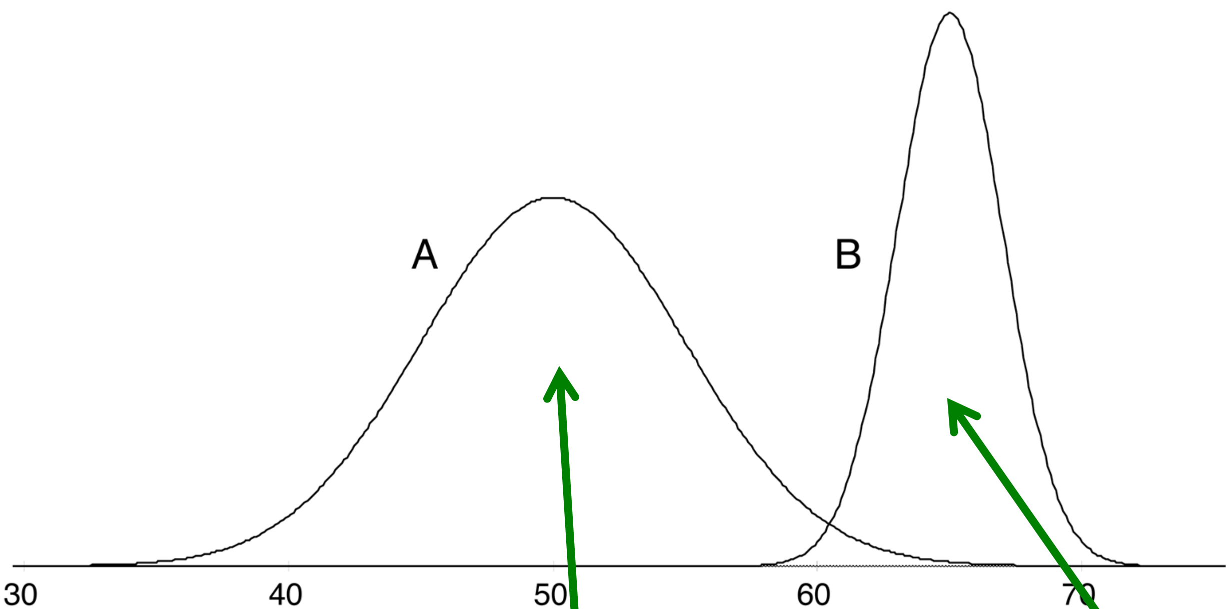
- Probabilistic perspective $\Rightarrow$
seek the *most likely* clusters given the data
- Also: instance belongs to a particular cluster *with a certain probability*

Two-Class Mixture Model

Data

A	**51**	**B**	**62**	**B**	**64**	**A**	**48**	**A**	**39**	**A**	**51**
A	**43**	**A**	**47**	**A**	**51**	**B**	**64**	**B**	**62**	**A**	**48**
B	**62**	**A**	**52**	**A**	**52**	**A**	**51**	**B**	**64**	**B**	**64**
B	**64**	**B**	**64**	**B**	**62**	**B**	**63**	**A**	**52**	**A**	**42**
A	**45**	**A**	**51**	**A**	**49**	**A**	**43**	**B**	**63**	**A**	**48**
A	**42**	**B**	**65**	**A**	**48**	**B**	**65**	**B**	**64**	**A**	**41**
A	**46**	**A**	**48**	**B**	**62**	**B**	**66**	**A**	**48**		
A	**45**	**A**	**49**	**A**	**43**	**B**	**65**	**B**	**64**		
A	**45**	**A**	**46**	**A**	**40**	**A**	**46**	**A**	**48**		

Model



$\mu_A=50,\ \sigma_A=5,\ p_A=0.6$

$\mu_B=65,\ \sigma_B=2,\ p_B=0.4$

Learning the Clusters

- Assume:
  - We know there are *k* clusters
- Learn the clusters ⇒
  - Determine their parameters
  - i.e. means and standard deviations
- Performance criterion:
  - *Probability of training data given the clusters*
- EM algorithm
  - Finds a local maximum of the likelihood

Extending the Mixture Model

- More then two distributions: easy
- Several attributes: easy—assuming independence
- Correlated attributes: difficult
  - Joint model: bivariate normal distribution with a (symmetric) covariance matrix
  - n attributes: need to estimate $n + n(n+1)/2$ parameters

Multi-Instance Learning

- Simplicity-first methodology can be applied to multi-instance learning with surprisingly good results
- Two simple approaches, both using standard single-instance learners:
  - Manipulate the input to learning
  - Manipulate the output of learning

Aggregating the Input

- Convert multi-instance problem into single-instance one
  - Summarize the instances in a bag by computing mean, mode, minimum and maximum as new attributes
  - To classify a new bag the same process is used

Aggregating the Output

- Learn a single-instance classifier directly from the original instances in each bag
- To classify a new bag:
  - Decide on cluster for each instance in the bag
  - Aggregate the cluster predictions to produce a prediction for the bag as a whole
  - One approach: treat predictions as votes for the various clusters
  - A problem: bags can contain differing numbers of instances $\rightarrow$ give each instance a weight inversely proportional to the bag's size

Discussion

- Can interpret clusters by using supervised learning
  - Post-processing step
- Decrease dependence between attributes?
  - Pre-processing step
  - E.g. use *principal component analysis*