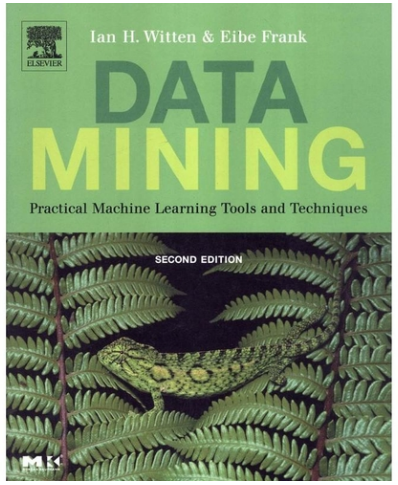
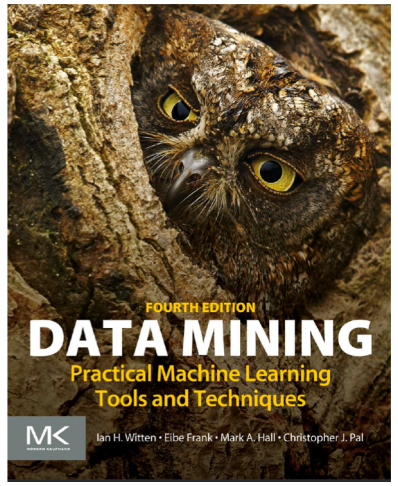
Machine Learning for Cyber-Security & Artificial Intelligence

Part 1 – Clustering

Hermes Senger





Adapted from:

Chapters 3.6, 4.8, 9.3, Bayesian networks
Data Mining: Practical Machine Learning Tools and Techniques,
4th Edition. By Ian H. Witten, Eibe Frank, Mark A. Hall, Christopher J. Pal.
Morgan Kauffman, 2017.

Agenda

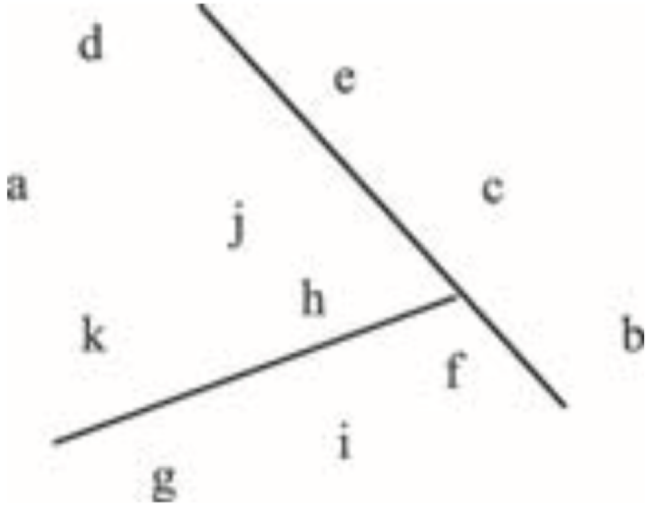
- Clusters – Chapter 3.6
- Clustering – Chapter 4.8
- Clustering and probability density estimation – Chapter 9.2

Agenda

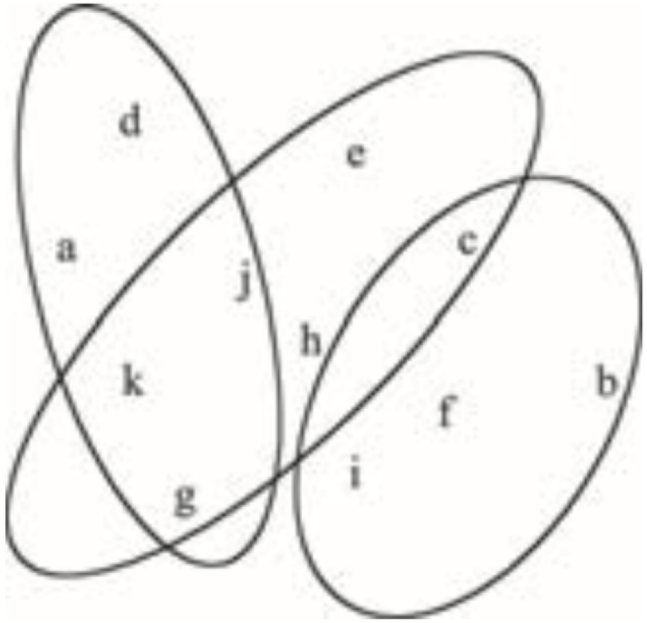
- Clusters – Chapter 3.6
- Clustering – Chapter 4.8
- Clustering and probability density estimation – Chapter 9.2

Representing clusters I

Simple 2-D representation



Venn diagram

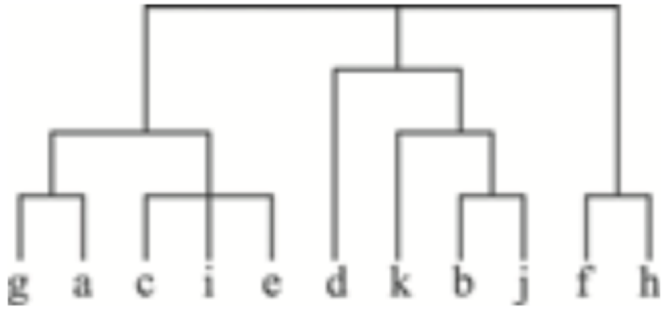


Representing clusters II

Probabilistic assignment

	1	2	3
a	0.4	0.1	0.5
b	0.1	0.8	0.1
c	0.3	0.3	0.4
d	0.1	0.1	0.8
e	0.4	0.2	0.4
f	0.1	0.4	0.5
g	0.7	0.2	0.1
h	0.5	0.4	0.1
…			

Dendrogram



Agenda

- Clusters – Chapter 3.6
- Clustering – Chapter 4.8
- Clustering and probability density estimation – Chapter 9.2

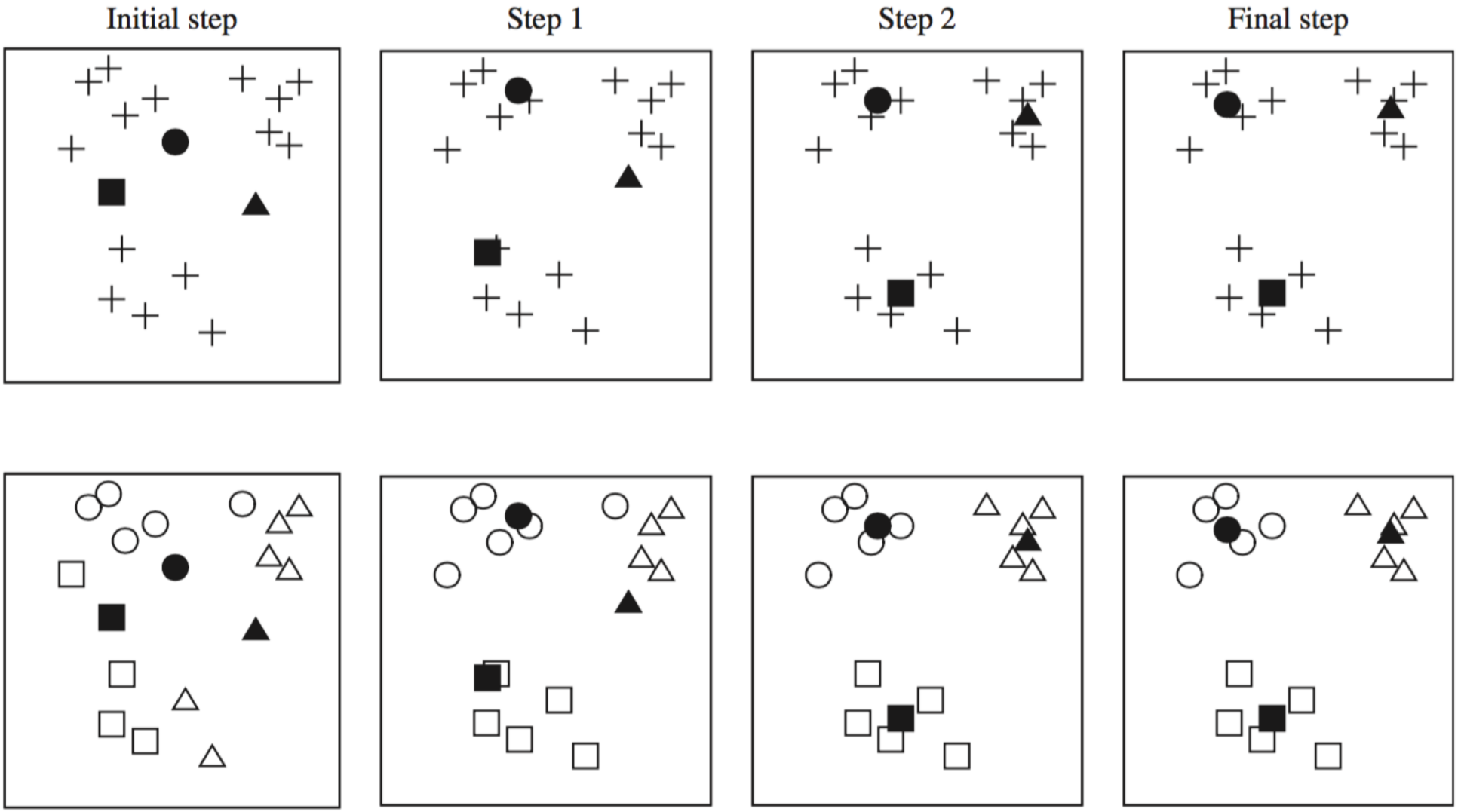
Clustering

- Clustering techniques apply when there is no class to be predicted: they perform unsupervised learning
- Aim: divide instances into “natural” groups
- As we have seen, clusters can be:
  - disjoint vs. overlapping
  - deterministic vs. probabilistic
  - flat vs. hierarchical
- We will look at a classic clustering algorithm called *k-means*
- *k-means* clusters are disjoint, deterministic, and flat

The *k*-means algorithm

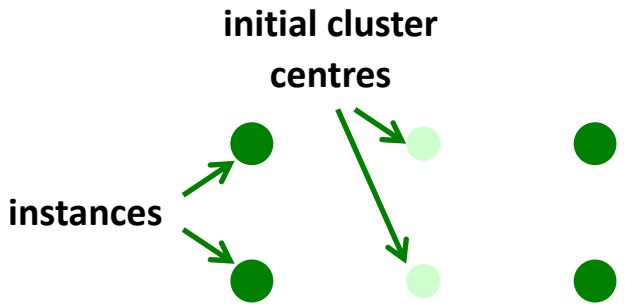
- Step 1: Choose *k* random cluster centers
- Step 2: Assign each instance to its closest cluster center based on Euclidean distance
- Step 3: Recompute cluster centers by computing the average (aka *centroid*) of the instances pertaining to each cluster
- Step 4: If cluster centers have moved, go back to Step 2
- This algorithm minimizes the squared Euclidean distance of the instances from their corresponding cluster centers
  - Determines a solution that achieves a *local* minimum of the squared Euclidean distance
- Equivalent termination criterion: stop when assignment of instances to cluster centers has not changed

The *k*-means algorithm: example



Discussion

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

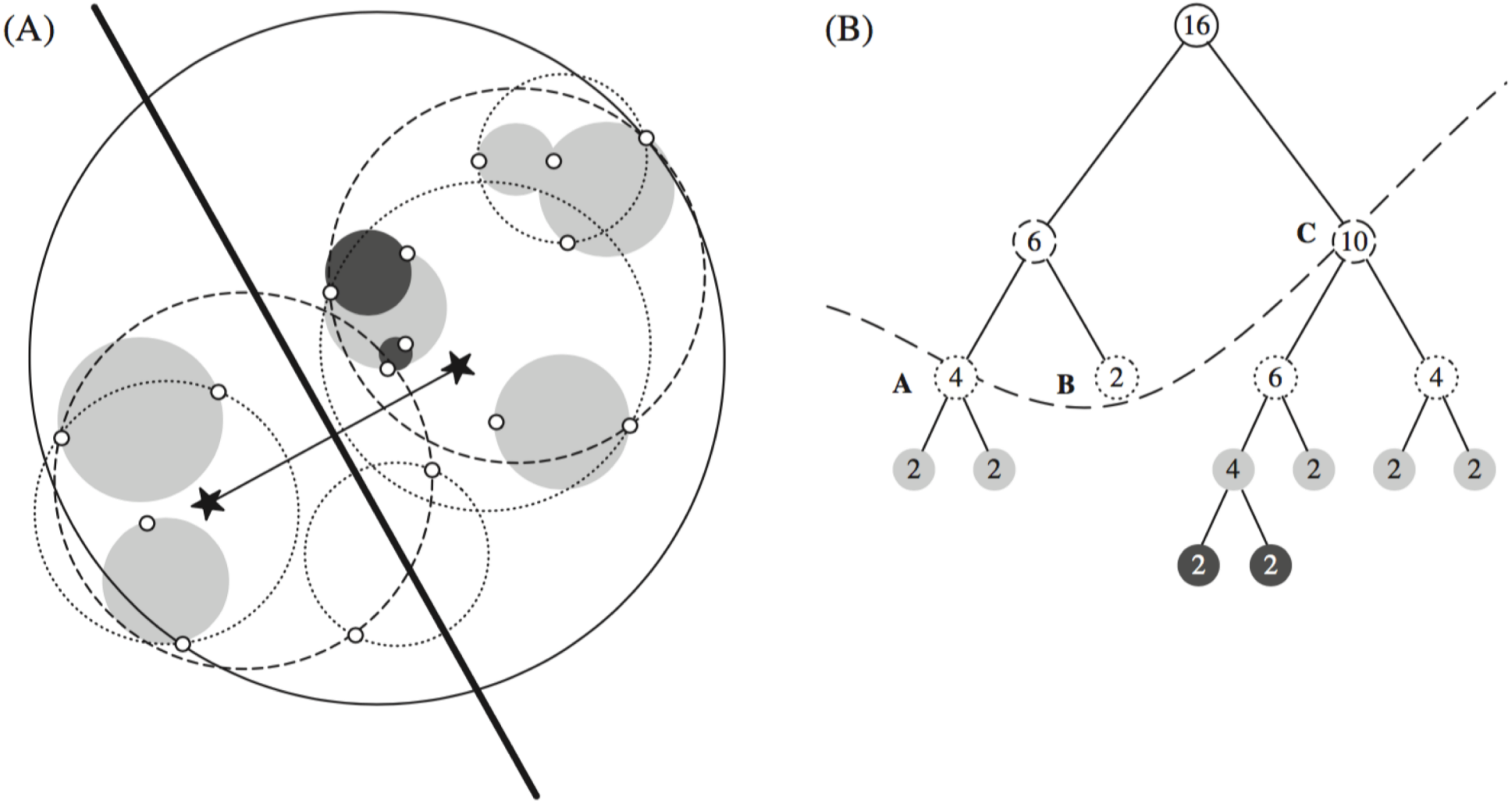


- To increase chance of finding global optimum: restart with different random seeds
- Can we applied recursively with $k = 2$

Faster distance calculations

- Can we use *k*D-trees or ball trees to speed up the process? Yes, we can:
  - First, build the tree data structure, which remains static, for all the data points
  - At each node, store the number of instances and the sum of all instances (summary statistics)
  - In each iteration of *k*-means, descend the tree and find out which cluster each node belongs to
  - Can stop descending as soon as we find out that a node belongs entirely to a particular cluster
  - Use summary statistics stored previously at the nodes to compute new cluster centers

Example scenario (using a ball tree)



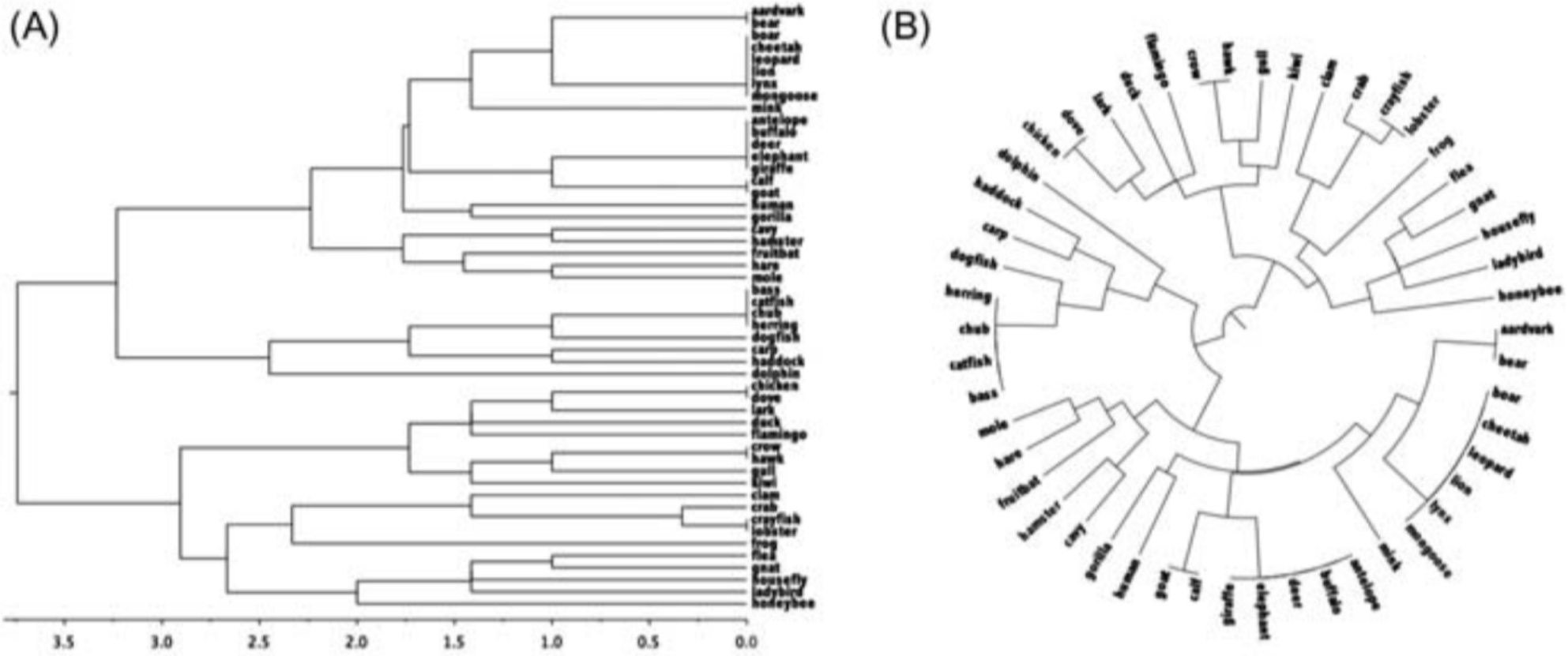
Choosing the number of clusters

- Big question in practice: what is the right number of clusters, i.e., what is the right value for *k*?
- Cannot simply optimize squared distance on training data to choose k
  - Squared distance decreases monotonically with increasing values of *k*
- Need some measure that balances distance with complexity of the model, e.g., based on the MDL principle (covered later)
- Finding the right-size model using MDL becomes easier when applying a recursive version of *k*-means (*bisecting k-means*):
  - Compute A: information required to store data centroid, and the location of each instance with respect to this centroid
  - Split data into two clusters using 2-means
  - Compute B: information required to store the two new cluster centroids, and the location of each instance with respect to these two
  - If A > B, split the data and recurse (just like in other tree learners)

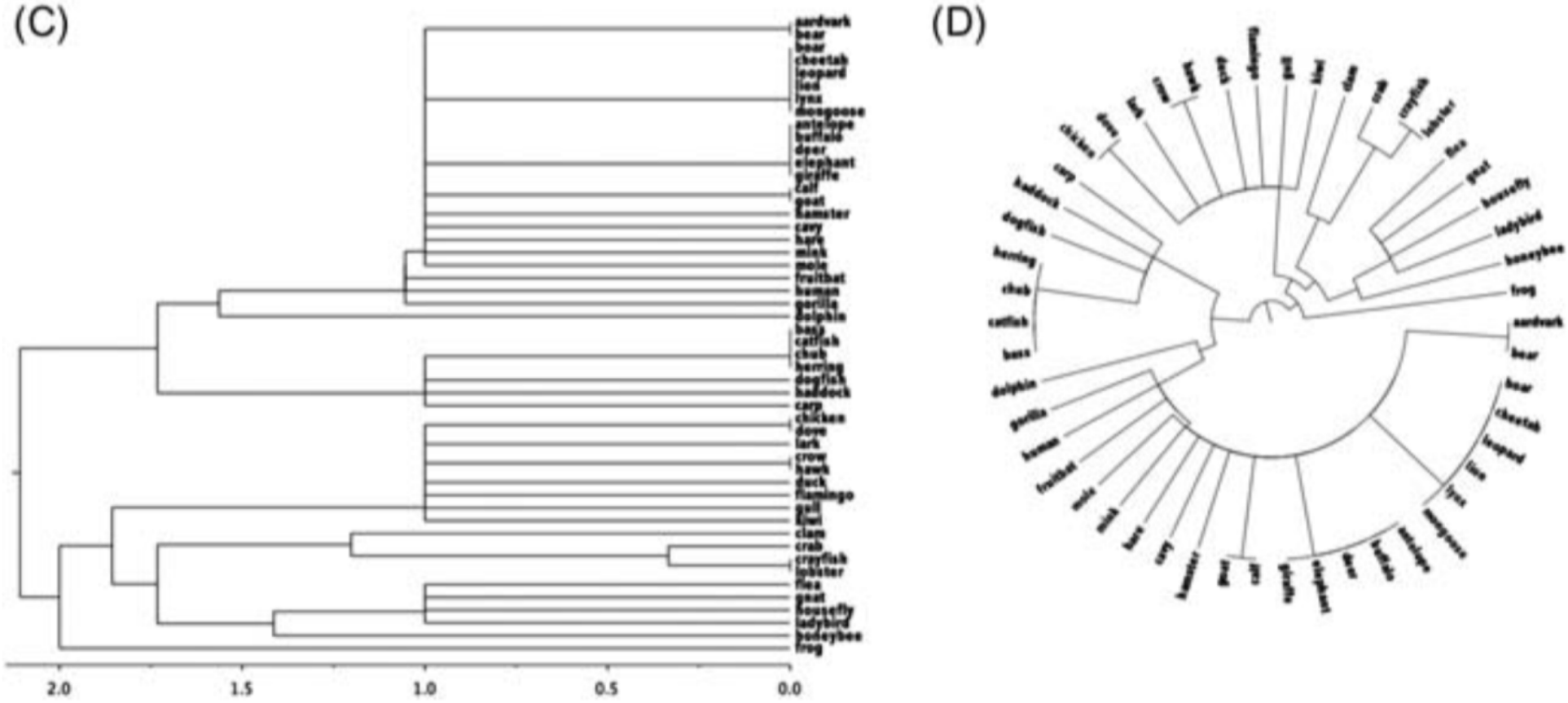
Hierarchical clustering

- Bisecting *k*-means performs hierarchical clustering in a top-down manner
- Standard hierarchical clustering performs clustering in a bottom-up manner; it performs *agglomerative* clustering:
  - First, make each instance in the dataset into a trivial mini-cluster
  - Then, find the two closest clusters and merge them; repeat
  - Clustering stops when all clusters have been merged into a single cluster
- Outcome is determined by the distance function that is used:
  - *Single-linkage* clustering: distance of two clusters is measured by finding the two closest instances, one from each cluster, and taking their distance
  - *Complete-linkage* clustering*:* use the two most distant instances instead
  - *Average-linkage* clustering: take average distance between all instances
  - *Centroid-linkage* clustering: take distance of cluster centroids
  - *Group-average* clustering: take average distance in merged clusters
  - *Ward's method*: optimize *k-means* criterion (i.e., squared distance)

Example: complete linkage



Example: single linkage



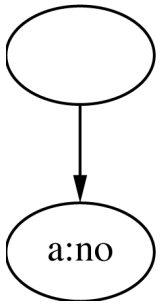
Incremental clustering

- Heuristic approach (COBWEB/CLASSIT)
- Forms 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 using *merging* or *splitting* of nodes
- Update decisions are based on a goodness measure called *category utility*

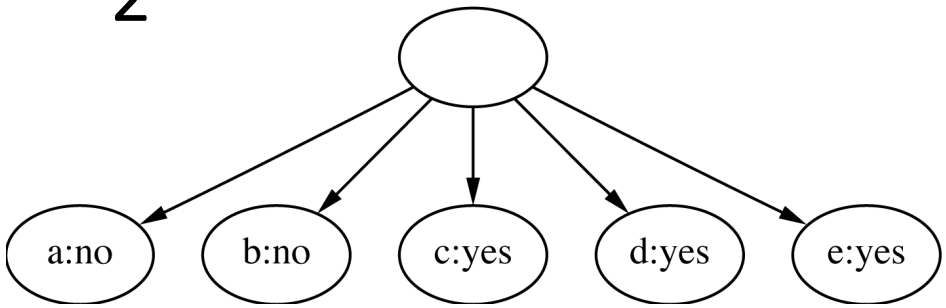
Clustering the weather data I

ID	Outlook	Temp.	Humidity	Windy
A	Sunny	Hot	High	False
B	Sunny	Hot	High	True
C	Overcast	Hot	High	False
D	Rainy	Mild	High	False
E	Rainy	Cool	Normal	False
F	Rainy	Cool	Normal	True
G	Overcast	Cool	Normal	True
H	Sunny	Mild	High	False
I	Sunny	Cool	Normal	False
J	Rainy	Mild	Normal	False
K	Sunny	Mild	Normal	True
L	Overcast	Mild	High	True
M	Overcast	Hot	Normal	False
N	Rainy	Mild	High	True

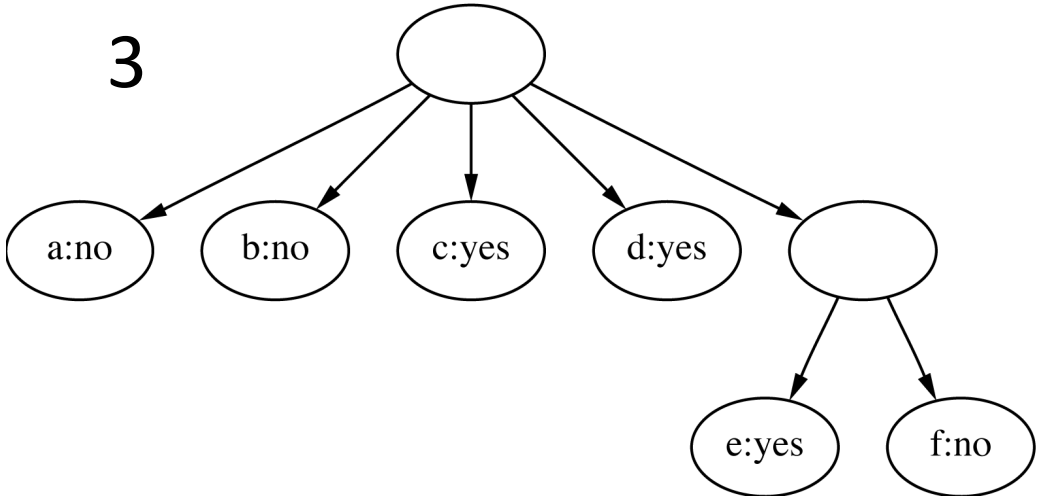
1



2



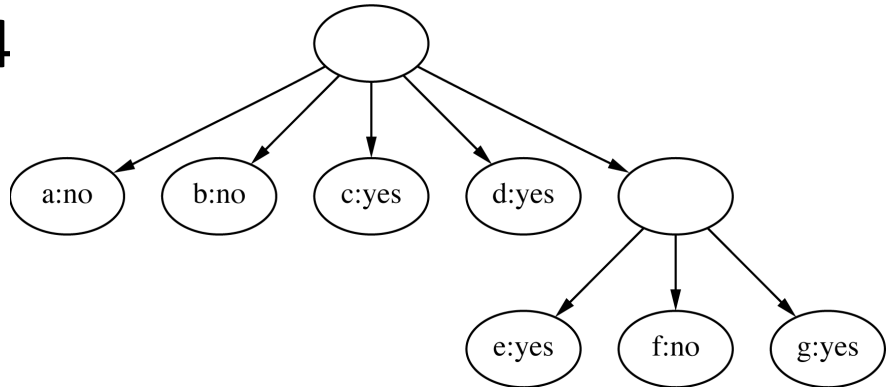
3



Clustering the weather data II

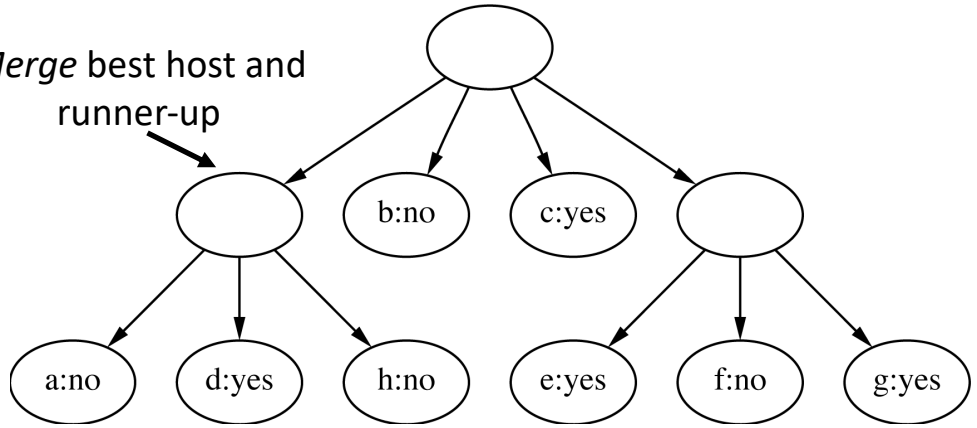
ID	Outlook	Temp.	Humidity	Windy
A	Sunny	Hot	High	False
B	Sunny	Hot	High	True
C	Overcast	Hot	High	False
D	Rainy	Mild	High	False
E	Rainy	Cool	Normal	False
F	Rainy	Cool	Normal	True
G	Overcast	Cool	Normal	True
H	Sunny	Mild	High	False
I	Sunny	Cool	Normal	False
J	Rainy	Mild	Normal	False
K	Sunny	Mild	Normal	True
L	Overcast	Mild	High	True
M	Overcast	Hot	Normal	False
N	Rainy	Mild	High	True

4



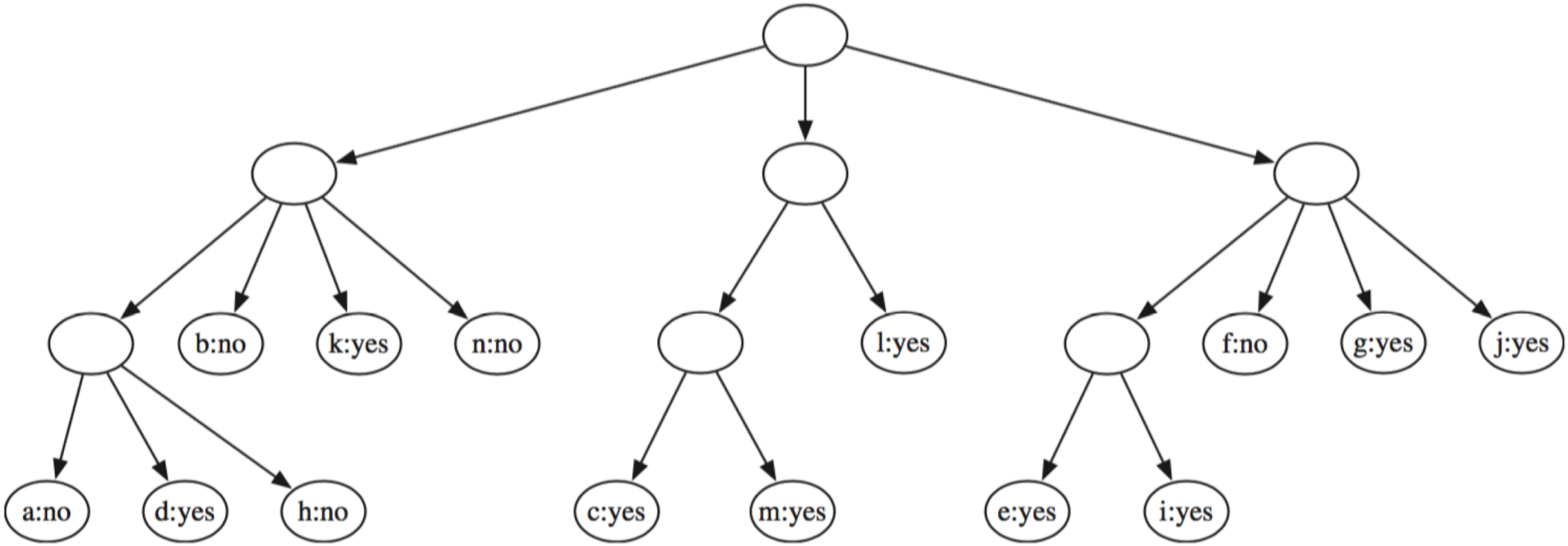
5

Merge best host and runner-up

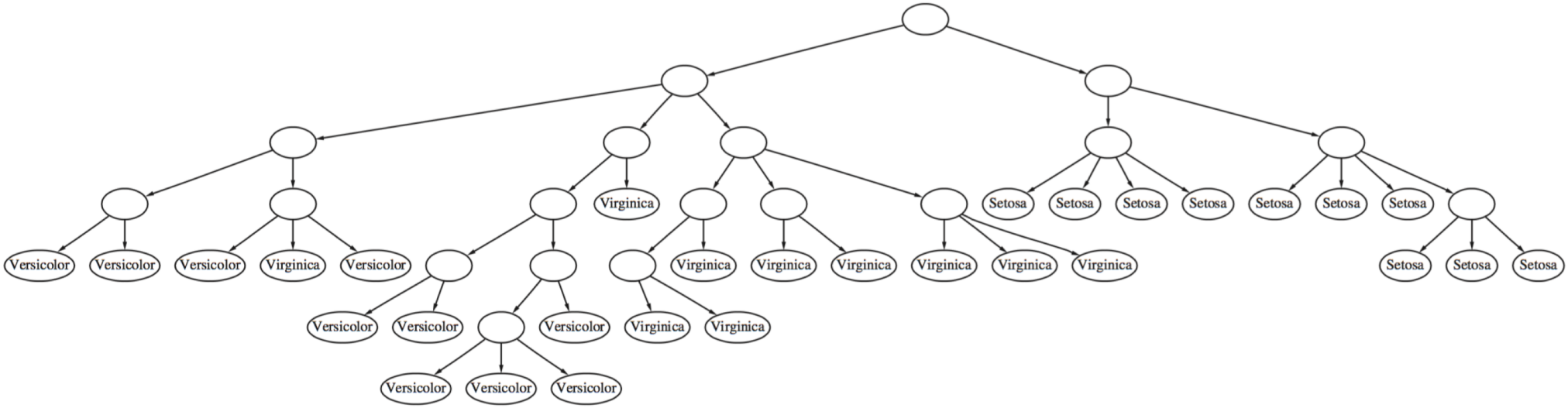


Consider *splitting* the best host if merging does not help

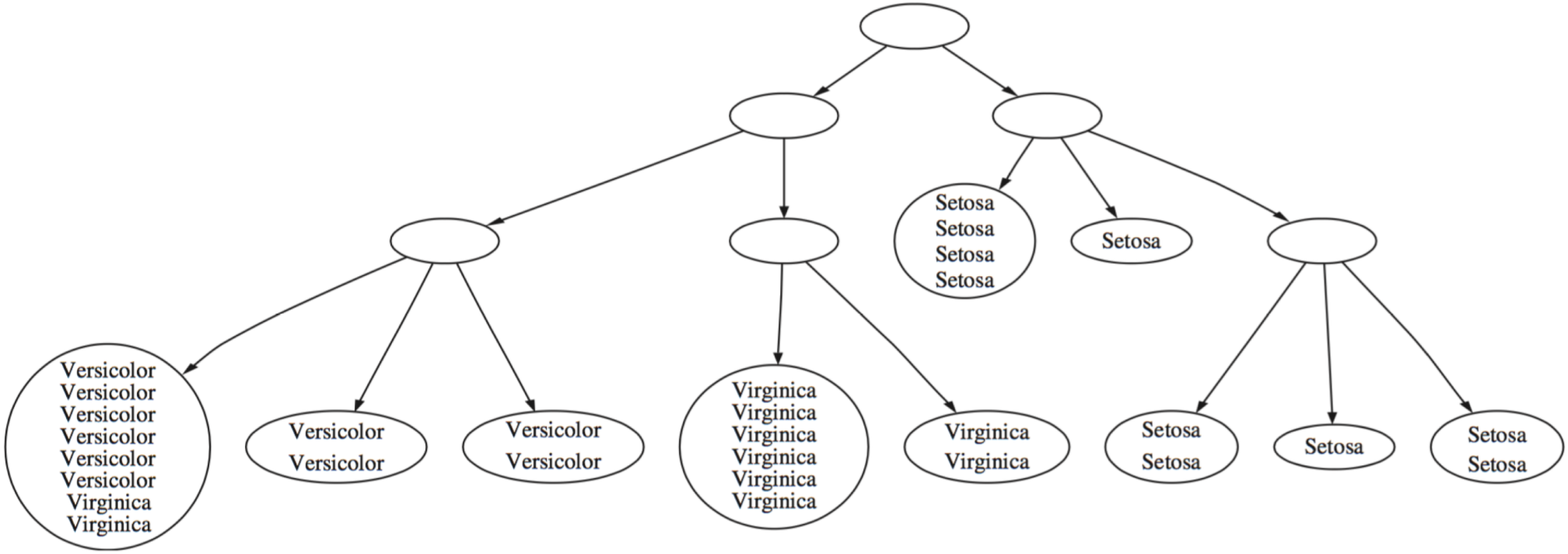
Final clustering



Example: clustering a subset of the iris data



Example: iris data with cutoff



The category utility measure

- Category utility: quadratic loss function defined on conditional probabilities:

$$CU(C_1, C_2, \ldots, C_k) = \frac{\sum_l \mathrm{P}(C_l) \sum_i \sum_j (\mathrm{P}(a_i = v_{ij} \mid C_l)^2 - \mathrm{P}(a_i = v_{ij})^2)}{k}$$

- Every instance in a different category ⇒ numerator becomes

$$m - \mathrm{P}(a_i = v_{ij})^2 \longleftarrow \text{maximum}$$

m: number of attributes

Numeric attributes?

- Assume normal distribution: $f(a) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(a-\mu)^2}{2\sigma^2}}$
- Then: $\sum_j \mathrm{P}(a_i = v_{ij})^2 \Leftrightarrow \int f(a_i)^2 \, da_i = \frac{1}{2\sqrt{\pi}\sigma_i}$
- Thus

$$CU = \frac{\sum_l \mathrm{P}(C_l) \sum_i \sum_j (\mathrm{P}(a_i = v_{ij} \mid C_l)^2 - \mathrm{P}(a_i = v_{ij})^2)}{k}$$

becomes

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

- Prespecified minimum variance can be enforced to combat overfitting (called *acuity* parameter)