Laboratory Module 8

Hierarchical Clustering

Purpose:

- Understand theoretical aspects and the most important algorithms used for Hierarchical Clustering;
- See examples of the domains where hierarchical clustering are used in practice;
- Solve practical problems with hierarchical clustering.

1. Theoretical aspects: Assignments

1.1 What is Hierarchical clustering?

Hierarchical clustering is a method of cluster analysis which follows to build a hierarchy of clusters.
Hierarchical cluster analysis (or hierarchical clustering) is a general approach to cluster analysis, in which the object is to group together objects or records that are "close" to one another.

A key component of the analysis is repeated calculation of distance measures between objects, and between clusters once objects begin to be grouped into clusters. The outcome is represented graphically as a dendrogram (the dendrogram is a graphical representation of the results of hierarchical cluster analysis).

The initial data for the hierarchical cluster analysis of N objects is a set of **N x (N – 1)/ 2** object-to-object distances and a **linkage function** for computation of the cluster-to-cluster distances. **A linkage function** is an essential feature for hierarchical cluster analysis. Its value is a measure of the "distance" between two groups of objects (i.e. between two clusters).

The two main **categories of methods for hierarchical cluster analysis** are ***divisive methods*** and ***agglomerative methods***. In practice, the agglomerative methods are of wider use. On each step, the pair of clusters with smallest cluster-to-cluster distance is fused into a single cluster.

1.2 Where Hierarchical Clustering is useful?

First example where hierarchical clustering would be useful is a study *to predict the cost impact of deregulation*. To do the requisite analysis, economists would need to build a detailed cost model of the various utilities. It would save a considerable amount of time and effort if we could cluster similar types of utilities, build detailed cost models for just one typical utility in each cluster, then scale up from these models to estimate results for all utilities.

Second example where hierarchical clustering would be useful is for *automatic control of urban road traffic with both adaptive traffic lights and variable message signs*. Using hierarchical cluster analysis we can specify the needed number of stationary road traffic sensors and their preferable locations within a given road network.

Third example of using a hierarchical clustering is *to take a file that contains nutritional information for a set of breakfast cereals*. We have the following information: the cereal name, cereal manufacturer, type (hot or cold), number of calories per serving, grams of protein, grams of fat, milligrams of sodium, grams of fiber, grams of carbohydrates, grams of sugars, milligrams of potassium, typical percentage of the FDA's RDA of vitamins, the weight of one serving, the number of cups in one serving. *Hierarchical Clustering help to find which cereals are the best and worst in a particular category.*

1.3 Algorithms for hierarchical clustering:

The most common algorithms for hierarchical clustering are:

Agglomerative methods

An agglomerative hierarchical clustering procedure produces a series of partitions of the data, P_n, P_{n-1}, … , P_1. The first P_n consists of n single object 'clusters', the last P_1, consists of single group containing all n cases.

At each particular stage the method joins together the two clusters which are closest together (most similar). (At the first stage, of course, this amounts to joining together the two objects that are closest together, since at the initial stage each cluster has one object.)

Differences between methods arise because of the different ways of defining distance (or similarity) between clusters. Several agglomerative techniques will now be described in detail.

Single linkage clustering

One of the simplest agglomerative hierarchical clustering method is single linkage, also known as the nearest neighbor technique. The defining feature of the method is that distance between groups is defined as the distance between the closest pair of objects, where only pairs consisting of one object from each group are considered.

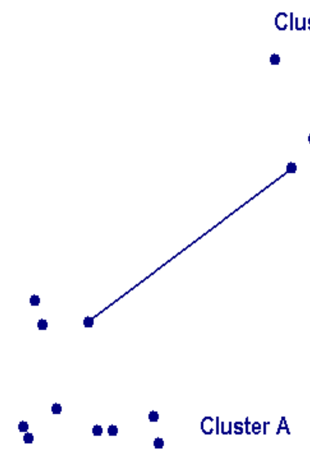
In the single linkage method, $\boldsymbol{D(r,s)}$ is computed as

$$\boldsymbol{D(r,s)} = \text{Min} \{ \ d(i,j) : \text{Where object } i \text{ is in cluster } \boldsymbol{r} \text{ and object } j \text{ is cluster } \boldsymbol{s} \ \}$$

Here the distance between every possible object pair (i,j) is computed, where object i is in cluster ***r*** and object *j* is in cluster ***s***. The minimum value of these distances is said to be the distance between clusters ***r*** and ***s.*** In other words, the distance between two clusters is given by the value of the shortest link between the clusters.

At each stage of hierarchical clustering, the clusters ***r*** and ***s*** , for which ***D(r,s)*** is minimum, are merged.

This measure of inter-group distance is illustrated in the figure below:



Complete linkage clustering

The complete linkage, also called farthest neighbor, clustering method is the opposite of single linkage. Distance between groups is now defined as the distance between the most distant pair of objects, one from each group.

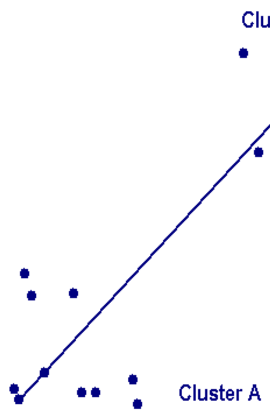
In the complete linkage method, ***D(r,s)*** is computed as

D(r,s) = Max { *d(i,j)* : Where object *i* is in cluster ***r*** and object *j* is cluster ***s*** }

Here the distance between every possible object pair *(i,j)* is computed, where object *i* is in cluster ***r*** and object *j* is in cluster ***s*** and the maximum value of these distances is said to be the distance between clusters ***r*** and ***s.*** In other words, the distance between two clusters is given by the value of the longest link between the clusters.

At each stage of hierarchical clustering, the clusters ***r*** and ***s*** , for which ***D(r,s)*** is minimum, are merged.

The measure is illustrated in the figure below:



Average linkage clustering

The distance between two clusters is defined as the average of distances between all pairs of objects, where each pair is made up of one object from each group.

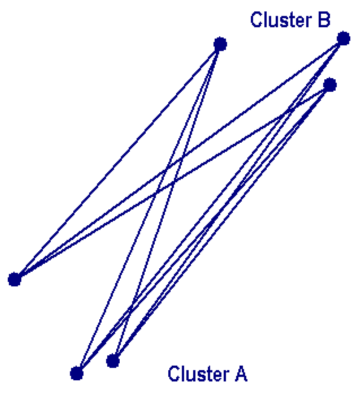
In the average linkage method, ***D(r,s)*** is computed as

$$D(r,s) = T_{rs} / (N_r * N_s)$$

Where T_{rs} is the sum of all pairwise distances between cluster ***r*** and cluster ***s***. N_r and N_s are the sizes of the clusters ***r*** and ***s*** respectively.

At each stage of hierarchical clustering, the clusters ***r*** and ***s*** , for which ***D(r,s)*** is the minimum, are merged.

The figure below illustrates average linkage clustering:



Average group linkage

With this method, groups once formed are represented by their mean values for each variable, that is, their mean vector, and inter-group distance is now defined in terms of distance between two such mean vectors.

In the average group linkage method, the two clusters **r** and **s** are merged such that, after merger, the average pairwise distance within the newly formed cluster, is minimum. Suppose we label the new cluster formed by merging clusters **r** and **s**, as **t**. Then **D(r,s)** , the distance between clusters **r** and **s** is computed as

D(r,s) = Average { d(i,j) : Where observations i and j are in cluster **t**, the cluster formed by merging clusters **r** and **s** }

At each stage of hierarchical clustering, the clusters **r** and **s** , for which **D(r,s)** is minimum, are merged. In this case, those two clusters are merged such that the newly formed cluster, on average, will have minimum pairwise distances between the points in it.

Cobweb

Cobweb generates hierarchical clustering, where clusters are described probabilistically. Below is an example clustering of the weather data (weather.arff). The class attribute (play) is ignored (using the ignore attributes panel) in order to allow later classes to clusters evaluation. Doing this automatically through the "Classes to clusters" option does not make much sense for hierarchical clustering, because of the large number of clusters. Sometimes we need to evaluate particular clusters or levels in the clustering hierarchy.

How Weka represents the Cobweb clusters?

Below is a copy of the output window, showing the run time information and the structure of the clustering tree.

```
Scheme:       weka.clusterers.Cobweb -A 1.0 -C 0.234
Relation:     weather
Instances:    14
Attributes:   5
              outlook
              temperature
              humidity
              windy
Ignored:
              play
Test mode:    evaluate on training data
```

```
Clustering model (full training set)

Number of merges: 2
Number of splits: 1
Number of clusters: 6

node 0 [14]
|   node 1 [8]
|   |   leaf 2 [2]
|   node 1 [8]
|   |   leaf 3 [3]
|   node 1 [8]
|   |   leaf 4 [3]
node 0 [14]
|   leaf 5 [6]


Evaluation on training set

Number of merges: 2
Number of splits: 1
Number of clusters: 6

node 0 [14]
|   node 1 [8]
|   |   leaf 2 [2]
|   node 1 [8]
|   |   leaf 3 [3]
|   node 1 [8]
|   |   leaf 4 [3]
node 0 [14]
|   leaf 5 [6]

Clustered Instances

2       2 ( 14%)
3       3 ( 21%)
4       3 ( 21%)
5       6 ( 43%)
```

Comments on the output above:

- **node N** or **leaf N** represents a **subcluster, whose parent cluster is N.**
- The **clustering tree** structure is shown as a horizontal tree, where subclusters are aligned at the same column. For example, cluster 1 (referred to in node 1) has three subclusters 2 (leaf 2), 3 (leaf 3) and 4 (leaf 4).
- The **root** cluster is 0. Each line with **node 0** defines a subcluster of the root.
- The number in square brackets after **node N** represents the number of instances in the parent cluster **N**.
- Clusters with [1] at the end of the line are **instances**.
- For example, in the above structure cluster 1 has 8 instances and its subclusters 2, 3

and 4 have 2, 3 and 3 instances correspondingly.

- To view the clustering tree **right click** on the last line in the **result list** window and then select **Visualize tree**.

To **evaluate** the Cobweb clustering using the **classes to clusters** approach we need to know the class values of the instances, belonging to the clusters. We can get this information from Weka in the following way: After Weka finishes (with the class attribute ignored), **right click** on the last line in the **result list** window. Then choose **Visualize cluster assignments** - you get the **Weka cluster visualize** window. Here you can view the clusters, for example by putting **Instance_number** on X and **Cluster** on Y. Then click on **Save** and choose a file name (*.arff). Weka saves the **cluster assignments** in an ARFF file. Below is shown the file corresponding to the above Cobweb clustering.

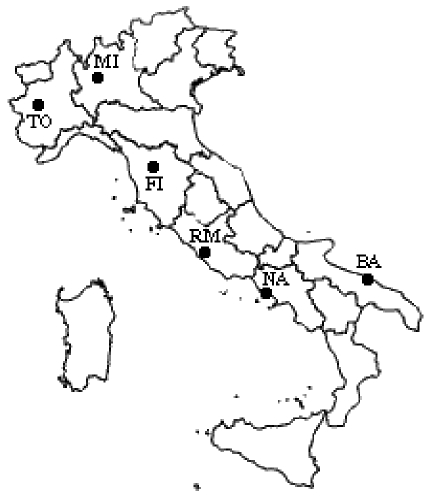
2. Examples

First example:

A hierarchical clustering of distances in kilometers between some Italian cities. The method used is *single-linkage*.

Input distance matrix (L = 0 for all the clusters):

	BA	FI	MI	NA	RM	TO
BA	0	662	877	255	412	996
FI	662	0	295	468	268	400
MI	877	295	0	754	564	138
NA	255	468	754	0	219	869
RM	412	268	564	219	0	669
TO	996	400	138	869	669	0

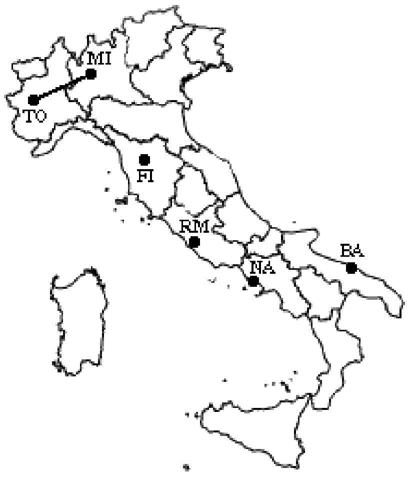


The nearest pair of cities is MI and TO, at distance 138. These are merged into a single cluster called "MI/TO". The level of the new cluster is L(MI/TO) = 138 and the new sequence number is m = 1.

Then we compute the distance from this new compound object to all other objects. In single link clustering the rule is that the distance from the compound object to another object is equal to the shortest distance from any member of the cluster to the outside object. So the distance from "MI/TO" to RM is chosen to be 564, which is the distance from MI to RM, and so on.

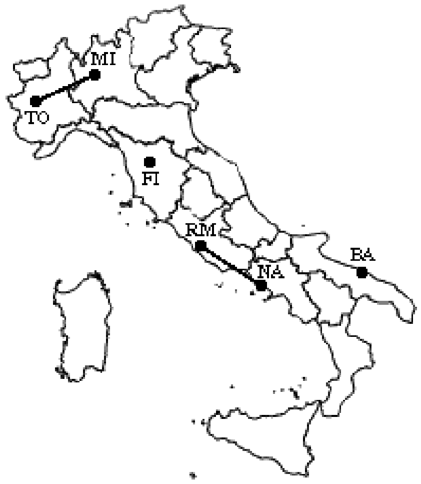
After merging MI with TO we obtain the following matrix:

	BA	**FI**	**MI/TO**	**NA**	**RM**
BA	0	662	877	255	412
FI	662	0	295	468	268
MI/TO	877	295	0	754	564
NA	255	468	754	0	219
RM	412	268	564	219	0



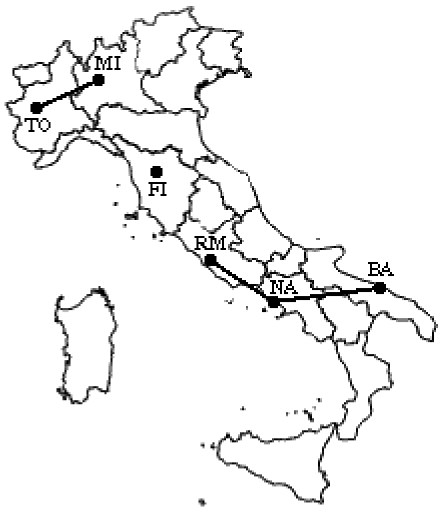
min d(i,j) = d(NA,RM) = 219 => merge NA and RM into a new cluster called NA/RM
L(NA/RM) = 219
m = 2

	BA	**FI**	**MI/TO**	**NA/RM**
BA	0	662	877	255
FI	662	0	295	268
MI/TO	877	295	0	564
NA/RM	255	268	564	0



min d(i,j) = d(BA,NA/RM) = 255 => merge BA and NA/RM into a new cluster called BA/NA/RM
L(BA/NA/RM) = 255
m = 3

	BA/NA/RM	**FI**	**MI/TO**
BA/NA/RM	0	268	564
FI	268	0	295
MI/TO	564	295	0



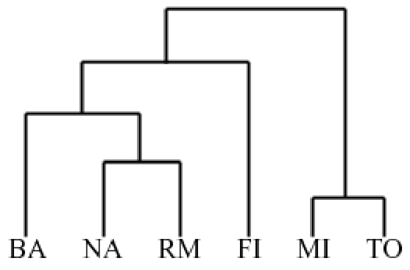
min d(i,j) = d(BA/NA/RM,FI) = 268 => merge BA/NA/RM and FI into a new cluster called BA/FI/NA/RM
L(BA/FI/NA/RM) = 268
m = 4

	BA/FI/NA/RM	MI/TO
BA/FI/NA/RM	0	295
MI/TO	295	0

MI
TO
FI
RM
NA
BA

Finally, we merge the last two clusters at level 29

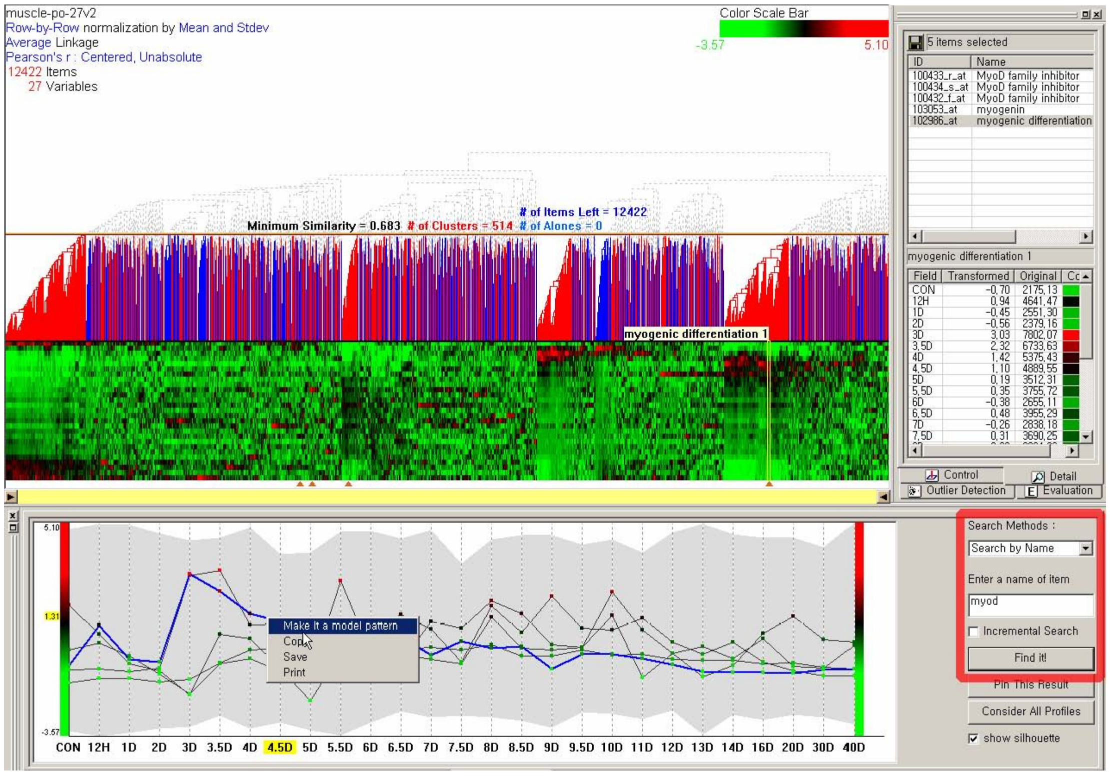
The process is summarized by the following hierarchical tree:



<u>Second example</u>:

Coordination Example:

Researchers performed a microarray experiment to generate a gene expression profile data set that indicates relative levels of expression for each of these genes (> 12000) in murine muscle samples. They measured expression levels at 27 time points to find genes that are biologically relevant to the muscle regeneration process. They already know that *MyoD* is a gene that is the most relevant to muscle regeneration. They run the hierarchical clustering with the data set, and identify a relevant cluster that peaks at day 3. In the parallel coordinates view, they search *MyoD* using search-by-name query, then make it a model pattern to perform a model-based query. They modify the model pattern to emphasize the peak at day 3 and then adjust the similarity thresholds to get the search result that mostly overlaps with the relevant day 3 cluster (Fig. 1 & Fig. 2). Finally, they confirm through other biological experiments that 2 genes (*Cdh15* and *Stam*) in the overlapped result set are novel downstream targets of *MyoD*.



muscle-po-27v2
Row-by-Row normalization by Mean and Stdev
Average Linkage
Pearson's r : Centered, Unabsolute
12422 Items
27 Variables

Minimum Similarity = 0.683 # of Clusters = 514 # of Items Left = 12422 # of Alones = 0

5 items selected

ID	Name
100433_r_at	MyoD family inhibitor
100434_s_at	MyoD family inhibitor
100432_f_at	MyoD family inhibitor
103053_at	myogenin
102986_at	myogenic differentiation

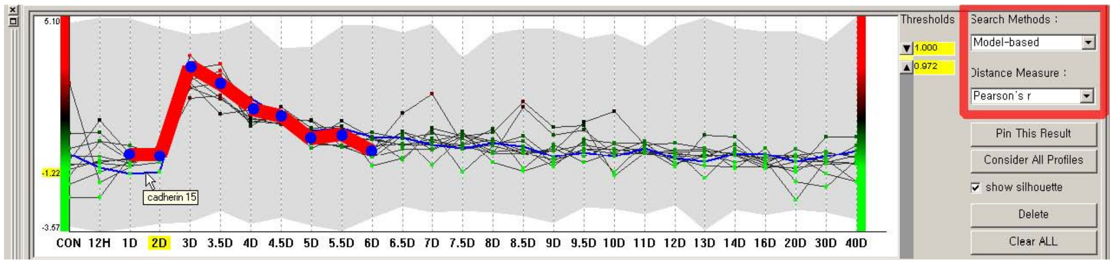
myogenic differentiation 1

Field	Transformed	Original	Cc
CON	-0,70	2175,13	🟩
12H	0,94	4641,47	■
1D	-0,45	2551,30	🟩
2D	-0,56	2379,16	🟩
3D	3,03	7802,07	🟥
3,5D	2,32	6733,63	🟥
4D	1,42	5375,43	■
4,5D	1,10	4889,55	■
5D	0,19	3512,31	🟩
5,5D	0,35	3755,72	🟩
6D	-0,38	2655,11	🟩
6,5D	0,48	3955,29	🟩
7D	-0,26	2838,18	🟩
7,5D	0,31	3690,25	🟩

Search Methods : Search by Name
Enter a name of item: myod
Incremental Search

Fig. 1

Run a search-by-name query with *MyoD* to find 5 genes whose names contain *MyoD*, and the 5 genes are projected onto the current clustering result visualization shown by triangles under the color mosaic. Select a gene (*myogenic differentiation 1*) and make it a model pattern for next query.



Thresholds: 1.000, 0.972
Search Methods : Model-based
Distance Measure : Pearson's r

5.10
-1.22
-3.57
cadherin 15
Pin This Result
Consider All Profiles
show silhouette
Delete
Clear ALL
CON 12H 1D 2D 3D 3.5D 4D 4.5D 5D 5.5D 6D 6.5D 7D 7.5D 8D 8.5D 9D 9.5D 10D 11D 12D 13D 14D 16D 20D 30D 40D

Fig. 2

Modify the model pattern to emphasize the peak at day 3 (notice the bold red line), and run a model-based query to find a small set of candidate genes. The updated search result will be highlighted in the dendrogram view and other views.

3. Assignments

Problem 1:

We have the following data files:

cereal.txt (without 'vitamin' and 'rating' columns) : 77 x 9
here: http://www.cs.umd.edu/hcil/hce/examples/cereal/cereal.txt

cereal-updated.txt (with 'vitamin' and 'rating' columns) : 77 x 11
here: http://www.cs.umd.edu/hcil/hce/examples/cereal/cereal-updated.txt

The meaning of each column :

1. 1st column : Name of cereal
2. **calories**: calories per serving
3. **protein**: grams of protein
4. **fat**: grams of fat
5. **sodium**: milligrams of sodium
6. **fiber**: grams of dietary fiber
7. **carbo**: grams of complex carbohydrates
8. **sugars**: grams of sugars
9. **potass**: milligrams of potassium
10. **vitamins**: vitamins and minerals - 0, 25, or 100, indicating the typical percentage of FDA recommended
11. **shelf**: display shelf (1, 2, or 3, counting from the floor)
12. **rating**: a rating of the cereals (calculated by Consumer Reports)

Requirements:

Use the given data files to find the following using WEKA:
1. Is a strong correlation between dietary fiber and potassium?
2. Are groups of cereals from which we can choose according to our preferences?
3. See other correlation between the data given in the files.

Problem 2:

We have the following data files:

netscan-08-2003.txt (activity log of newsgroups where name contains "*windowsxp*" for August 2003) : 91x10

here: http://www.cs.umd.edu/hcil/hce/examples/netscan/netscan-08-2003.txt

netscan-1year.txt (activity log of newsgroups where name contains "*windowsxp*" for a year) : 104 x 10

here: http://www.cs.umd.edu/hcil/hce/examples/netscan/netscan-1year.txt

The meaning of each column :

1. 1st column : name of newsgroup
2. **Posts** : # of messages that were contributed to the newsgroup
3. **Posters**: : # of people who contributed at least on message to the newsgroup
4. **PPRatio**: the ratio of posters to posts
5. **Returnees**: # of people who contributed to the newsgroup in the current time period and also contributed a message in the previous time period
6. **Replies**: # of people who contributed at least one message that was a reply to another message
7. **UnRMSGS**: # of messages in the newsgroup that did not receive any reply in the newsgroup
8. **Avg.LineCT**: average # of lines in each message
9. **XPosts**:# of messages that were shared with at least one other newsgroup
10. **XPTgs**:# of newsgroups that shared messages with the selected newsgroups

Requirements:

Use the given data files to find the following using WEKA:
1. What are the most active groups in terms of the number of people involved cluster together?
2. What are the most active communitie?

The COBWEB Conceptual Clustering Algorithm

The COBWEB algorithm was developed by machine learning researchers in the 1980s for clustering objects in a object-attribute data set. The COBWEB algorithm yields a clustering dendrogram called classification tree that characterizes each cluster with a probabilistic description.

Operation of the COBWEB algorithm

The COBWEB algorithm constructs a classification tree incrementally by inserting the objects into the classification tree one by one.
When inserting an object into the classification tree, the COBWEB algorithm traverses the tree top-down starting from the root node.

At each node, the COBWEB algorithm considers 4 possible operations and select the one that yields the highest CU function value:

- insert.
- create.
- merge.
- split.

The COBWEB algorithm operates based on the so-called **category utility function** (**CU**) that measures clustering quality.
If we partition a set of objects into *m* clusters, then the CU of this particular partition is

$$CU(C_1, C_2, \ldots, C_k) = \frac{\sum_l \Pr[C_l] \sum_i \sum_j \overbrace{\left(\Pr[a_i = v_{ij} \mid C_l]^2 - \Pr[a_i = v_{ij}]^2\right)}^{\text{Improvement in probability estimate because of instance cluster assigment}}}{k}$$

If each instance in its own cluster:

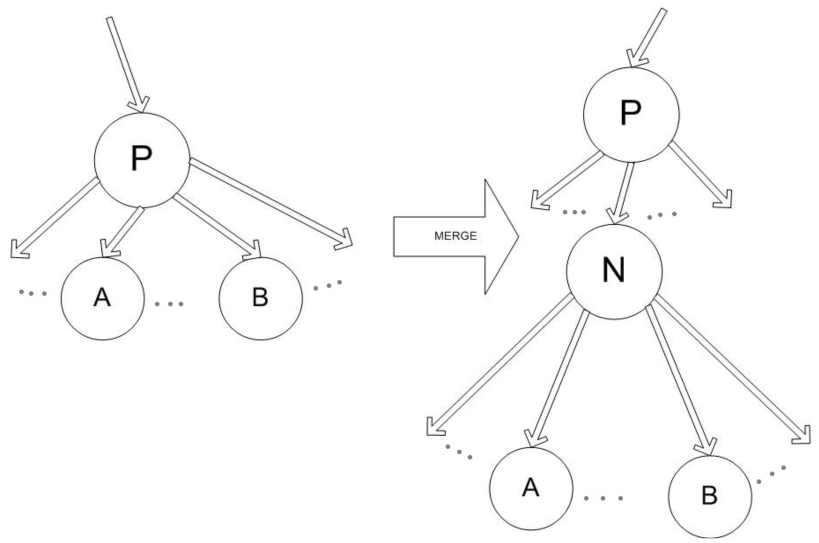
$$\Pr[a_i = v_{ij} \mid C_l] = \begin{cases} 1 & v_{ij} = \text{actual value of instance} \\ 0 & \text{otherwise} \end{cases}$$

Category utility function becomes:

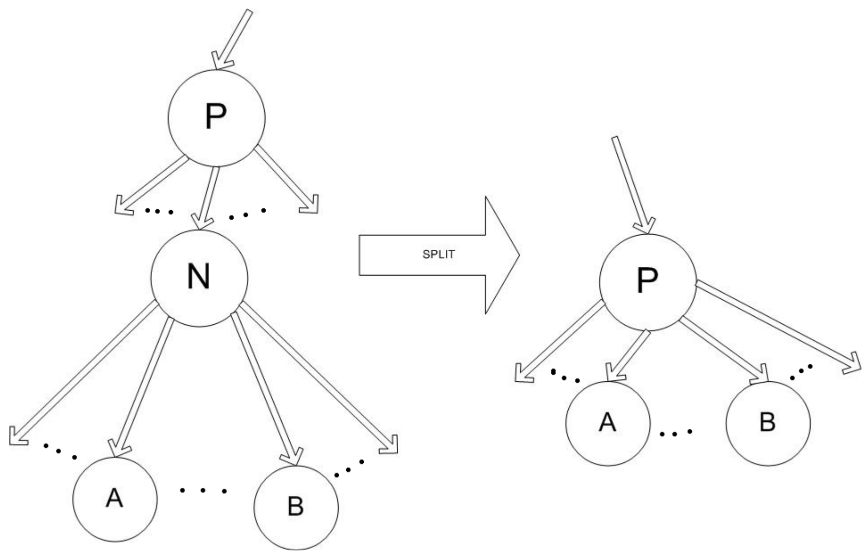
$$CU(C_1, C_2, \ldots, C_k) = \frac{n - \sum_i \sum_j \Pr[a_i = v_{ij}]^2}{k}$$

Without k it would always be best for each instance to have its own cluster, **overfitting**!

Insertion means that the new object is inserted into one of the existing child nodes. The COBWEB algorithm evaluates the respective CU function value of inserting the new object into each of the existing child nodes and selects the one with the highest score.
The COBWEB algorithm also considers creating a new child node specifically for the new object.
The COBWEB algorithm considers merging the two existing child nodes with the highest and second highest scores.



The COBWEB algorithm considers spliting the existing child node with the highest score.



The COBWEB Algorithm

Input:	The current node N in the concept hierarchy.
	An unclassified (attribute-value) instance I.
Results:	A concept hierarchy that classifies the instance.
Top-level call:	Cobweb(Top-node, I).
Variables:	C, P, Q, and R are nodes in the hierarchy.
	U, V, W, and X are clustering (partition) scores.

```
Cobweb(N, I)
    If N is a terminal node,
        Then Create-new-terminals(N, I)
```

```
                    Incorporate(N,I).
    Else Incorporate(N, I).
        For each child C of node N,
            Compute the score for placing I in C.
        Let P be the node with the highest score W.
        Let Q be the node with the second highest score.
        Let X be the score for placing I in a new node R.
        Let Y be the score for merging P and Q into one node.
        Let Z be the score for splitting P into its children.
        If W is the best score,
            Then Cobweb(P, I) (place I in category P).
        Else if X is the best score,
            Then initialize R's probabilities using I's values
                (place I by itself in the new category R).
        Else if Y is the best score,
            Then let O be Merge(P, R, N).
                Cobweb(O, I).
        Else if Z is the best score
            Then Split(P, N).
                Cobweb(N, I).
```

Auxiliary COBWEB Operations

```
Variables:  N, O, P, and R are nodes in the hierarchy.
            I is an unclassified instance.
            A is a nominal attribute.
            V is a value of an attribute.
Incorporate(N, I)
    update the probability of category N.
    For each attribute A in instance I,
        For each value V of A,
            Update the probability of V given category N.
Create-new-terminals(N, I)
    Create a new child M of node N.
    Initialize M's probabilities to those for N.
    Create a new child O of node N.
    Initialize O's probabilities using I's value.

Merge(P, R, N)
    Make O a new child of N.
    Set O's probabilities to be P and R's average.
    Remove P and R as children of node N.
    Add P and R as children of node O.
    Return O.
Split(P, N)
    Remove the child P of node N.
```

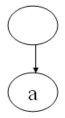
Promote the children of P to be children of N.

An example of using COBWEB Algorithm:

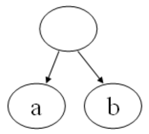
We have the following data:

Outlook	Temp.	Humidity	Windy	Play
Sunny	Hot	High	FALSE	No
Sunny	Hot	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Rainy	Mild	High	FALSE	Yes
Rainy	Cool	Normal	FALSE	Yes
Rainy	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Sunny	Mild	High	FALSE	No
Sunny	Cool	Normal	FALSE	Yes
Rainy	Mild	Normal	FALSE	Yes
Sunny	Mild	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Rainy	Mild	High	TRUE	No

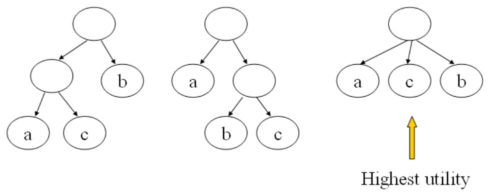
Weather Data (without Play)
Label instances: a,b,….,n
Start by putting the first instance in its own cluster:



Add another instance in its own cluster:

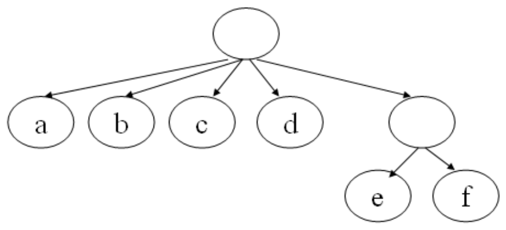


Adding the Third Instance
Evaluate the category utility of adding the instance to one of the two clusters versus adding it as its own cluster:



Adding Instance f

First instance not to get its own cluster:



Look at the instances:

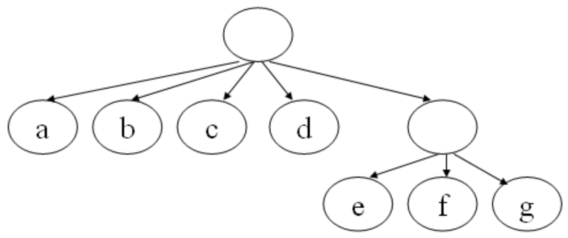
```
Rainy Cool Normal FALSE
Rainy Cool Normal TRUE
```

Quite similar!

Add Instance g

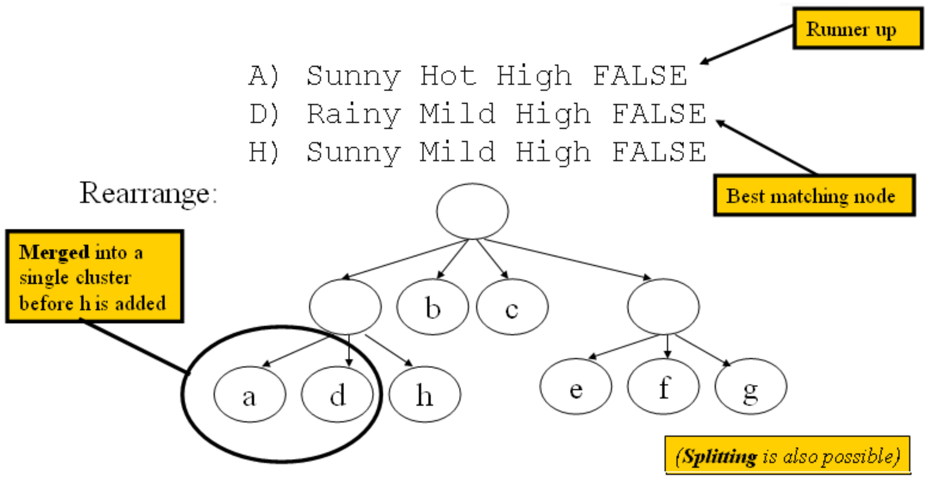
Look at the instances:

```
E) Rainy Cool Normal FALSE
F) Rainy Cool Normal TRUE
G) Overcast Cool Normal TRUE
```



Add Instance h

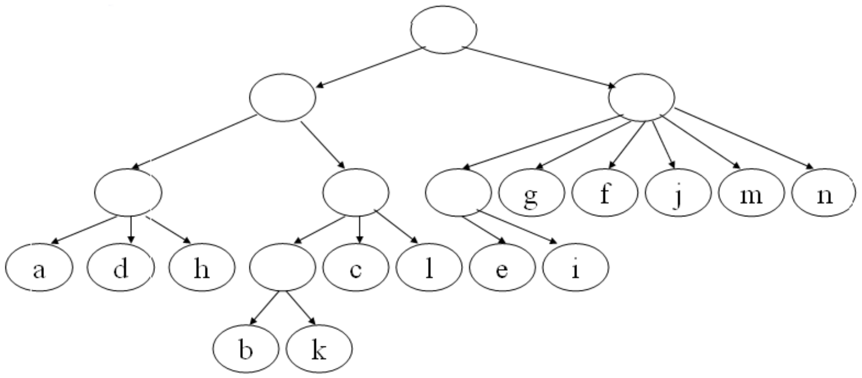
Look at the instances:



```
A) Sunny Hot High FALSE
D) Rainy Mild High FALSE
H) Sunny Mild High FALSE
```

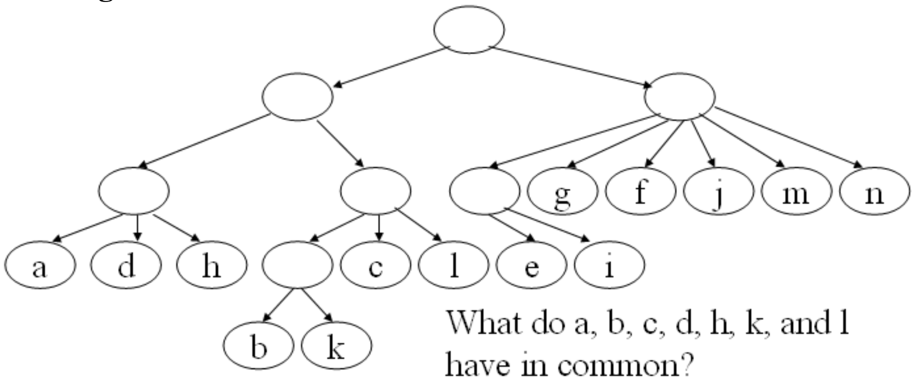
Runner up
Best matching node
Rearrange:
Merged into a single cluster before h is added
a
d
h
b
c
e
f
g
(Splitting is also possible)

Final Hierarchy



What next?

Dendrogram → Clusters



What do a, b, c, d, h, k, and l have in common?