

Session 10

Hierarchical and density-based clustering

Hierarchical clustering produces a tree of merges that does not need k in advance; DBSCAN finds clusters of arbitrary shape and marks noise. Both are run on the datasets you built in Session 1.

Objectives

✗ Do not copy. Read for understanding and the viva

- Complete questions 22 to 23 of the manual: hierarchical and density-based clustering
- Prepare the deliverable before the lab and finish it during the session
- Be ready to explain every step in the viva

Questions Covered

✗ Do not copy. Read for understanding and the viva

Question	Requirement	Status
Q22	Implement Hierarchical Clustering Algorithm to demonstrate the clustering rule process...	Complete
Q23	Implement Density based Clustering Algorithm to demonstrate the clustering rule...	Complete

Preparation

✗ Do not copy. Read for understanding and the viva

- `HierarchicalClusterer` with `linkType` SINGLE, COMPLETE, AVERAGE; compare the dendrograms (Visualize tree) and note how single linkage chains.

- `DBSCAN` (install via the package manager if missing) needs `epsilon` and `minPoints`; start with `epsilon` 0.9 on normalised data and `minPoints` 3, then vary.
- For each run, record cluster sizes, unclustered (noise) instances, and one sentence on what the clusters mean for employees or students.

Question 22

Problem Statement

■ Write in lab record

Implement Hierarchical Clustering Algorithm to demonstrate the clustering rule process in the following datasets:

1. `employee.arff`
2. `student.arff`

Solution

■ Write in lab record

Steps

1. **Open file...**, `employee.arff` (Session 7). **Cluster** tab, **Ignore attributes**, select `department` and `promoted` so the three numeric attributes `age`, `experience` and `salary` are used (WEKA normalises them to 0 to 1 inside `EuclideanDistance`).
2. **Choose**, `HierarchicalClusterer`. Click the name: `numClusters` 2, `linkType` SINGLE, `distanceFunction` `EuclideanDistance`, `printNewick` true. **Start**.
3. Read the output: the Newick string describing the tree with merge heights, then Clustered Instances. Right-click the result, **Visualize tree** for the dendrogram.
4. Change `linkType` to COMPLETE, **Start**; then AVERAGE, **Start**. Then repeat all three with `numClusters` 3.
5. Repeat steps 1 to 4 for `student.arff` with `stream` and `result` ignored.
6. `hierarchical.py` does the same agglomeration and prints every merge with its distance (which is the dendrogram written as a list) and the clusters at the cut. Run `python3 hierarchical.py employee.arff average 3` and the other linkages.

Program

hierarchical.py

```

1  #!/usr/bin/env python3
2  """Agglomerative hierarchical clustering (single, complete, average linkage)
3  attributes of an ARFF file, normalised to [0,1] like WEKA's HierarchicalClus
4  Prints every merge (the dendrogram as text) and the clusters when the tree i
5  Run: python3 hierarchical.py employee.arff average 2
6  """
7  import math
8  import sys
9
10
11 def load_numeric(path):
12     names, nominal, rows, data = [], [], [], False
13     for line in open(path):
14         line = line.strip()
15         if not line or line.startswith('%') or line.lower().startswith('@rel
16             continue
17         if line.lower().startswith('@attribute'):
18             _, name, typ = line.split(None, 2)
19             names.append(name)
20             nominal.append(typ.strip().startswith('{'))
21         elif line.lower() == '@data':
22             data = True
23         elif data:
24             rows.append([v.strip() for v in line.split(',')])
25     num = [i for i, n in enumerate(nominal) if not n]
26     X = [[float(r[i]) for i in num] for r in rows]
27     lo, hi = [min(c) for c in zip(*X)], [max(c) for c in zip(*X)]
28     Xn = [[(v - l) / ((h - l) or 1) for v, l, h in zip(r, lo, hi)] for r in
29     return [names[i] for i in num], X, Xn
30
31
32 def dist(a, b):
33     return math.sqrt(sum((x - y) ** 2 for x, y in zip(a, b)))
34
35
36 def linkage_distance(kind, A, B, D):
37     ds = [D[i][j] for i in A for j in B]
38     return {'single': min, 'complete': max, 'average': lambda v: sum(v) / le
39
40
41 def cluster(Xn, kind, k):
42     n = len(Xn)
43     D = [[dist(a, b) for b in Xn] for a in Xn]

```

```

44     clusters = [[i] for i in range(n)]
45     merges = []
46     while len(clusters) > 1:
47         best = min(((linkage_distance(kind, A, B, D), i, j) for i, A in enumerate(X), j, B in enumerate(Xn)))
48         d, i, j = best
49         merges.append((d, clusters[i], clusters[j]))
50         clusters = [c for t, c in enumerate(clusters) if t not in (i, j)] +
51         if len(clusters) == k:
52             cut = [sorted(c) for c in clusters]
53     return merges, cut
54
55
56 def main(path, kind, k):
57     names, X, Xn = load_numeric(path)
58     merges, cut = cluster(Xn, kind, k)
59     print(f"{path}: {len(X)} instances on {names}, {kind} linkage\n")
60     print("merge order (instance numbers start at 1; distance is on normalis")
61     for step, (d, A, B) in enumerate(merges, 1):
62         show = lambda c: '{' + ','.join(str(i + 1) for i in sorted(c)) + '}'
63         print(f"step {step:2d} d = {d:.4f} {show(A)} + {show(B)}")
64     print(f"\nclusters when cut at k = {k}:")
65     for j, c in enumerate(cut):
66         mean = [sum(X[i][a] for i in c) / len(c) for a in range(len(names))]
67         print(f"cluster {j}: {len(c)} instances {[i + 1 for i in c]}")
68         print('          mean ' + ', '.join(f"{n} = {m:.1f}" for n, m in zip(names, mean)))
69     # self-check: merge distances never decrease for single/complete/average
70     assert all(a[0] ≤ b[0] + 1e-12 for a, b in zip(merges, merges[1:]))
71
72
73 if __name__ == '__main__':
74     a = sys.argv[1:] + ['employee.arff', 'average', '2'][len(sys.argv) - 1:]
75     main(a[0], a[1], int(a[2]))

```

Output

Employee, average linkage, cut at k = 3 (run for real; only the last merges and the clusters are shown, the script prints all 29 steps):

```
1 employee.arff: 30 instances on ['age', 'experience', 'salary'], average link
2
3 merge order (instance numbers start at 1; distance is on normalised attribut
4 step 1 d = 0.0281 {1} + {22}
5 step 2 d = 0.0291 {5} + {24}
6 ...
7 step 26 d = 0.2442 {1,2,3,4,21,22,23,30} + {5,6,7,8,9,10,11,24,25,26,27}
8 step 27 d = 0.2975 {12,13,14,15,16,17,28} + {18,19,20}
9 step 28 d = 0.4740 {1,2,3,4,5,6,7,8,9,10,11,21,22,23,24,25,26,27,30} + {12
10 step 29 d = 1.3367 {29} + {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
11
12 clusters when cut at k = 3:
13 cluster 0: 1 instances [29]
14         mean age = 58.0, experience = 35.0, salary = 250.0
15 cluster 1: 19 instances [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 21, 22, 23, 24,
16         mean age = 28.4, experience = 4.5, salary = 41.0
17 cluster 2: 10 instances [12, 13, 14, 15, 16, 17, 18, 19, 20, 28]
18         mean age = 39.3, experience = 15.3, salary = 85.3
```

Summary of all six employee runs and all six student runs (the height is the distance of the last merge that the cut undoes):

Data set	Linkage	k = 2 clusters (sizes)	k = 3 clusters (sizes)	Height of top merge
employee	single	outlier 29 alone; all 29 others	29; 20; the other 28	0.745
employee	complete	29 alone; others	29; junior 19; senior 10	1.732
employee	average	29 alone; others	29; junior 19; senior 10	1.337
student	single	outlier 29 alone; others	29; five top students (2, 5, 10, 14, 18); the other 24	0.344
student	complete	9 weak (rows 20 to 27, 30); 21 others	same 9; 5 top students (2, 5, 10, 14, 18); 16 middle	1.732
student	average	9 strong (rows 1, 2, 5, 7, 10, 12, 14, 16, 18); 21 others	6 weak; 9 strong; 15 middle	0.836

Dendrogram, employee with average linkage, drawn from the merge list (heights not to scale):

```

1 height
2 1.34 |-----+
3      |
4 0.47 |-----+
5      |           |
6 0.24 |-----+           | 0.30 |-----+
7      |           |           |           |
8      juniors     juniors     seniors     seniors  29 (58 yrs, 35 yr
9      1-4,21-23,30 5-11,24-27 12-17,28    18-20    salary 250)

```

Newick form as WEKA prints it for the same tree, abbreviated: `((((1:0.03,22:0.03):0.2,...):0.47, (...):0.47):1.34,29:1.34)`. Every pair of numbers after a colon is a branch length; equal lengths on both children of a node mean the node is a merge at that height.

Explanation

- **Agglomeration.** Every row starts as its own cluster; at each step the two closest clusters merge; after 29 merges one cluster remains. The sequence of merge distances is the dendrogram, and "cut at k" means undoing the last k minus 1 merges. Nothing is recomputed to change k, which is the advantage over k-means.
- **Linkage decides what "closest" means.** Single linkage uses the nearest pair of rows, so a chain of rows each close to the next is merged early: on both files the whole population chains together and the only thing left at the top is the isolated outlier (row 29 on employee, the noisy row 29 on student). That is the chaining effect. Complete linkage uses the farthest pair, so it refuses to merge two spread-out groups and produces compact, balanced clusters: on student it isolates the nine weak students first. Average linkage lies between: it still separates the outlier on employee but gives the sensible junior/senior split at $k = 3$, and on student the nine strongest students form the first group.
- **Reading the heights.** The top merge on employee is at 1.34 for average linkage against 0.47 for the merge below it; a gap that large says "two natural groups plus an outlier". On student with complete linkage the top merge (1.73, the maximum possible in three normalised dimensions) joins the weak nine to everyone else, so the strongest structure in the data is weak versus the rest.
- **Meaning for the business or the college.** Employee: a junior cluster (mean 28 years, 4.5 years of experience, salary 41) and a senior cluster (39 years, 15 years, salary 85), with one executive who is unlike anyone else. Student: a group of nine consistent high performers, a group of six to nine at risk, and a middle group where attendance decides the outcome.

Question 23

Problem Statement

■ Write in lab record

Implement Density based Clustering Algorithm to demonstrate the clustering rule process on dataset employee.arff.

Solution

■ Write in lab record

Steps

1. Install the `optics_dbScan` package (GUI Chooser, **Tools, Package manager**), restart WEKA, load `employee.arff`, **Cluster** tab, **Ignore attributes** `department` and `promoted`.
2. **Choose**, `DBSCAN`. Click the name: `epsilon` 0.15, `minPoints` 3, `database_distanceType` EuclideanDataObject (attributes are normalised to 0 to 1). Cluster mode **Use training set, Start**.
3. Read "Number of generated clusters", the Clustered Instances block and "Unclustered instances" (the noise count). Right-click the result, **Visualize cluster assignments**, X `experience`, Y `salary`; noise rows have no cluster colour.
4. Vary `epsilon` over 0.05, 0.10, 0.15, 0.20 and 0.30 and `minPoints` over 2, 3 and 5, recording clusters and noise for each pair. `dbscan.py` prints the whole grid in one run: `python3 dbscan.py employee.arff 0.15 3`.

Program

dbscan.py

```

1 #!/usr/bin/env python3
2 """DBSCAN on the numeric attributes of an ARFF file, normalised to [0,1] lik
3 DBSCAN package. Prints a grid of epsilon and minPoints (clusters found, nois
4 then the membership for one chosen setting.
5 Run: python3 dbscan.py employee.arff 0.15 3
6 """
7 import math
8 import sys
9
10
11 def load_numeric(path):
12     nominal, rows, data = [], [], False
13     for line in open(path):
14         line = line.strip()
15         if not line or line.startswith('%') or line.lower().startswith('@rel
16             continue
17         if line.lower().startswith('@attribute'):
18             nominal.append(line.split(None, 2)[2].strip().startswith('{'))
19         elif line.lower() == '@data':
20             data = True
21         elif data:
22             rows.append([v.strip() for v in line.split(',')])
23     X = [[float(r[i]) for i, n in enumerate(nominal) if not n] for r in rows
24         lo, hi = [min(c) for c in zip(*X)], [max(c) for c in zip(*X)]
25     return [[(v - l) / ((h - l) or 1) for v, l, h in zip(r, lo, hi)] for r i
26
27
28 def dbscan(X, eps, min_pts):
29     n = len(X)
30     near = [[j for j in range(n) if math.dist(X[i], X[j]) ≤ eps] for i in r
31     label = [None] * n # None = unvisited, -1 = noise, else cluster id
32     c = 0
33     for i in range(n):
34         if label[i] is not None:
35             continue
36         if len(near[i]) < min_pts:
37             label[i] = -1
38             continue
39         label[i] = c
40         queue = list(near[i])
41         while queue:
42             j = queue.pop()
43             if label[j] == -1:

```

```

44         label[j] = c # border point
45         if label[j] is not None:
46             continue
47         label[j] = c
48         if len(near[j]) ≥ min_pts: # core point: expand
49             queue.extend(near[j])
50         c += 1
51     return label
52
53
54 def main(path, eps, min_pts):
55     X = load_numeric(path)
56     print(f"{path}: {len(X)} instances\n")
57     print(f"{'epsilon':>8} {'minPoints':>9} {'clusters':>8} {'noise':>5} cl
58     for e in (0.05, 0.10, 0.15, 0.20, 0.30):
59         for m in (2, 3, 5):
60             lab = dbscan(X, e, m)
61             k = max(lab) + 1
62             print(f"{e:8.2f} {m:9d} {k:8d} {lab.count(-1):5d} {[lab.count(j
63     lab = dbscan(X, eps, min_pts)
64     print(f"\n≡ epsilon = {eps}, minPoints = {min_pts} ≡")
65     for j in range(max(lab) + 1):
66         print(f"cluster {j}: instances {[i + 1 for i, l in enumerate(lab) if
67     print(f"noise (unclustered): instances {[i + 1 for i, l in enumerate(lab
68     # self-check: a point inside a cluster is never left as noise
69     assert all(l is not None for l in lab)
70
71
72 if __name__ == '__main__':
73     a = sys.argv[1:] + ['employee.arff', '0.15', '3'][len(sys.argv) - 1:]
74     main(a[0], float(a[1]), int(a[2]))

```

Output

```
python3 dbscan.py employee.arff 0.15 3 (run for real):
```

```

1  employee.arff: 30 instances
2
3  epsilon minPoints clusters noise  cluster sizes
4    0.05      2      5      8  [13, 2, 3, 2, 2]
5    0.05      3      2     14  [13, 3]
6    0.05      5      1     25  [5]
7    0.10      2      1      4  [26]
8    0.10      3      1      4  [26]
9    0.10      5      1      4  [26]
10   0.15      2      1      2  [28]
11   0.15      3      1      2  [28]
12   0.15      5      1      3  [27]
13   0.20      2      1      1  [29]
14   0.20      3      1      1  [29]
15   0.20      5      1      1  [29]
16   0.30      2      1      1  [29]
17   0.30      3      1      1  [29]
18   0.30      5      1      1  [29]
19
20  ≡≡≡ epsilon = 0.15, minPoints = 3 ≡≡≡
21  cluster 0: instances [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
22  noise (unclustered): instances [20, 29]

```

WEKA's block for the chosen setting:

```

1  DBSCAN clustering results
2  =====
3
4  Clustered DataObjects: 30
5  Number of attributes: 3
6  Epsilon: 0.15; minPoints: 3
7  Index: weka.clusterers.forOPTICSAndDBScan.Databases.SequentialDatabase
8  Distance-type: weka.clusterers.forOPTICSAndDBScan.DataObjects.EuclideanData0
9  Number of generated clusters: 1
10 Elapsed time: .01
11
12 Clustered Instances
13
14 0      28 (100%)
15
16 Unclustered instances : 2

```

Noise rows by setting: at epsilon 0.10 the four noise rows are 18, 19, 20 and 29 (the three most senior staff and the executive); at 0.15 only 20 and 29; at 0.20 only 29; row 29 (age 58, 35 years, salary 250) is noise until epsilon reaches about 0.75 on the normalised scale, because that is its distance to the nearest other row.

Explanation

- **Reading the grid.** A small epsilon (0.05) fragments the data: with minPoints 2 there are five tiny clusters and eight noise rows, and with minPoints 5 only five closely packed employees (rows 5, 6, 7, 24, 25) qualify as a cluster and 25 rows are noise. From epsilon 0.10 upwards the employees are one connected dense region and only the far end of the seniority scale is left out; from 0.20 upwards everyone but the executive is inside. So the clusters are stable across a wide band of epsilon and the noise set shrinks monotonically, which is the behaviour to look for when choosing the parameters.
- **Choosing epsilon and minPoints.** The rule of thumb is minPoints at least the number of attributes plus one (here 4, and 3 or 5 give the same answer), and epsilon at the knee of the sorted k-distance plot: for each row compute the distance to its minPoints-th neighbour, sort, and pick the value where the curve turns upward. On employee that knee is near 0.15, the setting used above.
- **Why DBSCAN and not k-means here.** k-means with $k = 2$ places the executive in a cluster of its own or shifts the senior centroid towards salary 250; hierarchical single linkage isolates the same row but only after chaining everyone else. DBSCAN reports it as noise directly, with no k , and would have found a second dense region of any shape had one existed.
- **What the clusters mean.** One cluster: the ordinary career ladder from junior to senior is a continuous band in age, experience and salary; the two noise rows are the people outside that band, who deserve a separate look rather than a place in a cluster average.

Formula Sheet

✗ Do not copy. Read for understanding and the viva

Association rules

For a rule $X \Rightarrow Y$ over N transactions:

$$\text{support}(X \Rightarrow Y) = \frac{|X \cup Y|}{N}, \quad \text{confidence}(X \Rightarrow Y) = \frac{|X \cup Y|}{|X|}, \quad \text{lift}(X \Rightarrow Y) = \frac{\text{confidence}(X \Rightarrow Y)}{\text{support}(Y)}$$

WEKA's Apriori starts at the upper bound of minimum support and lowers it by the delta each pass until the requested number of rules is found or the lower bound is reached.

Entropy, information gain and Gini

For a set S with class proportions $p_1, \dots, p_c$:

$$H(S) = - \sum_{i=1}^c p_i \log_2 p_i, \quad \text{Gain}(S, A) = H(S) - \sum_{v \in \text{values}(A)} \frac{|S_v|}{|S|} H(S_v), \quad \text{Gini}(S) = 1 - \sum_{i=1}^c p_i^2$$

ID3 splits on the attribute with the highest gain; J48 (C4.5) uses the gain ratio

$\text{Gain}(S, A) / \text{SplitInfo}(S, A)$ where $\text{SplitInfo}(S, A) = - \sum_v \frac{|S_v|}{|S|} \log_2 \frac{|S_v|}{|S|}$.

Classifier evaluation

From the confusion matrix with true positives TP , false positives FP , false negatives FN , true negatives TN :

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \quad \text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}, \quad F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Kappa compares observed agreement p_o (accuracy) with the agreement expected by chance p_e :

$$\kappa = \frac{p_o - p_e}{1 - p_e}, \quad p_e = \sum_i \frac{(\text{row}_i \text{ total}) (\text{column}_i \text{ total})}{N^2}$$

The ROC curve plots true positive rate $TP / (TP + FN)$ against false positive rate $FP / (FP + TN)$; the area under it (AUC) is 0.5 for guessing and 1.0 for a perfect classifier.

Naive Bayes and k-nearest neighbour

$$P(C | x_1, \dots, x_n) \propto P(C) \prod_{i=1}^n P(x_i | C)$$

k-NN assigns the majority class among the k nearest training records under Euclidean distance

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2}$$

after normalising each attribute to $[0, 1]$ with $x' = (x - x_{\min}) / (x_{\max} - x_{\min})$.

Linear regression

$$\hat{y} = \beta_0 + \beta_1 x, \quad \beta_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}, \quad \beta_0 = \bar{y} - \beta_1 \bar{x}$$

WEKA reports the correlation coefficient, mean absolute error and root mean squared error $\sqrt{\frac{1}{N} \sum (y_i - \hat{y}_i)^2}$.

Clustering

k-means minimises the within-cluster sum of squared errors over clusters $C_1, \dots, C_k$ with centroids μ_j :

$$SSE = \sum_{j=1}^k \sum_{\mathbf{x} \in C_j} \|\mathbf{x} - \mu_j\|^2, \quad \mu_j = \frac{1}{|C_j|} \sum_{\mathbf{x} \in C_j} \mathbf{x}$$

Hierarchical (agglomerative) clustering merges the two closest clusters each step; linkage defines closeness: single $\min d(a, b)$, complete $\max d(a, b)$, average $\frac{1}{|A||B|} \sum d(a, b)$.

DBSCAN calls a point a core point when at least minPts points lie within radius ϵ ; clusters grow from core points, and points reachable from none are noise.

Viva Questions

✗ Do not copy. Read for understanding and the viva

Q: What is the difference between agglomerative and divisive hierarchical clustering? **A:** Agglomerative starts from one cluster per row and merges; divisive starts from one cluster and splits. WEKA's HierarchicalClusterer is agglomerative.

Q: Define single, complete and average linkage. **A:** Distance between clusters is the minimum, maximum or mean of the pairwise row distances respectively.

Q: What is the chaining effect? **A:** With single linkage a sequence of rows each close to the next merges into one long cluster even if its ends are far apart.

Q: What is a dendrogram? **A:** The tree of merges with the merge distance as height; cutting it horizontally at a height gives a clustering.

Q: How do you get k clusters from a dendrogram? **A:** Undo the last k minus 1 merges, or set numClusters in WEKA.

Q: Define core, border and noise points in DBSCAN. **A:** A core point has at least minPoints rows within epsilon; a border point is within epsilon of a core point but is not one; noise is neither.

Q: Why does DBSCAN not need k? **A:** Clusters are connected components of core points, and their number is whatever the density structure gives.

Q: What happens to DBSCAN when clusters have different densities? **A:** One epsilon cannot suit both; the sparse cluster becomes noise or the dense ones merge. OPTICS addresses this.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Running HierarchicalClusterer with the nominal attributes included and getting merges driven by department mismatches.
- Reporting the $k = 2$ cut of single linkage as “two groups” when one group is a single outlier row.
- Setting epsilon in the original units (say, 5 years) while the distance is computed on normalised 0 to 1 attributes.
- Choosing epsilon so large that everything is one cluster and concluding the data has no structure.
- Treating noise rows as errors of the algorithm rather than as a result to report.
- Forgetting to install the `optics_dbScan` package and looking for DBSCAN under the built-in clusterers.

Session Summary

■ Write in lab record

- Question 22: HierarchicalClusterer with single, complete and average linkage on employee and student at $k = 2$ and 3; merge lists from `hierarchical.py`, dendrogram sketched, chaining of single linkage and the junior/senior and weak/strong splits recorded.
- Question 23: DBSCAN on employee with epsilon 0.05 to 0.30 and minPoints 2, 3, 5; grid from `dbscan.py`, chosen setting epsilon 0.15 and minPoints 3 giving one cluster of 28 with rows 20 and 29 as noise.

< Previous
Session 9