

Session 9

k-means clustering

Clustering finds structure without a class attribute. k-means is the standard starting point; the session runs it with several values of k and reads the sum of squared errors and centroids.

Objectives

✕ Do not copy. Read for understanding and the viva

- Complete questions 19 to 21 of the manual: k-means 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
Q19	Demonstrate Clustering features in Large Databases with noise	Complete
Q20	Implement simple K-Means Algorithm to demonstrate the clustering rule on the following...	Complete
Q21	Perform the following	Complete

Preparation

✕ Do not copy. Read for understanding and the viva

- `SimpleKMeans` parameters: `numClusters` (k), `seed`, `distanceFunction`. Run k = 2, 3, 4, 5 and tabulate SSE against k; the elbow suggests the natural k.

- For iris, use Classes to clusters evaluation to compare the three clusters with the three species.
- Question 19 asks about large noisy databases: relate it to DBSCAN and to BIRCH or CLARANS conceptually, and demonstrate `DBSCAN` in WEKA on a dataset with a few outliers.

Question 19

Problem Statement

■ Write in lab record

Demonstrate Clustering features in Large Databases with noise.

Solution

■ Write in lab record

Steps

1. **Concept first.** A “large database with noise” means more rows than fit in memory, clusters of arbitrary shape, and a fraction of rows that belong to no cluster (errors, outliers). k-means fails on all three: it needs several passes, finds only round clusters, and drags every outlier into some centroid. Three families of algorithms were designed for this and two are demonstrated below.
2. **DBSCAN** (density based). Two parameters: `epsilon`, the neighbourhood radius, and `minPoints`. A row with at least `minPoints` rows inside its radius is a **core point**; a row inside a core point’s radius but with fewer neighbours is a **border point**; every other row is **noise**. Clusters are the connected sets of core points plus their borders. One scan per point, any shape, noise labelled explicitly, no k.
3. **BIRCH** (Balanced Iterative Reducing and Clustering using Hierarchies). One scan of the data builds a height-balanced **CF tree** whose leaves hold clustering features $CF = (N, LS, SS)$: the count, the linear sum and the square sum of the points absorbed. Centroid, radius and diameter of any subcluster come from the CF alone, so the raw rows never need to be kept. Leaves with a radius above threshold split; outlier leaves with very few points are written to disk and revisited later. A conventional algorithm (k-means or hierarchical) then clusters the leaf CFs in memory.

4. **CLARANS** (Clustering Large Applications based on Randomised Search). A medoid-based method (like PAM) that treats each set of k medoids as a node of a graph whose neighbours differ in one medoid; it starts from a random node, moves to a random better neighbour, gives up after `maxneighbor` failed tries, and repeats `numlocal` times. Medoids are actual rows, so noise cannot pull a centre away the way it moves a mean; sampling makes it scale to large tables.
5. **Demonstration in WEKA.** DBSCAN is a package: GUI Chooser, **Tools, Package manager**, install `optics_dbScan`, restart. Then **Open file...**, `employee.arff` (Session 7), **Cluster** tab, **Ignore attributes** and select `department` and `promoted` so only the three numeric attributes are used, **Choose**, `DBSCAN`, click the name and set `epsilon` 0.15 and `minPoints` 3, **Start**. Then right-click the result, **Visualize cluster assignments**, X `experience`, Y `salary`, to see the noise rows drawn separately.
6. The same run is reproduced by `dbscan.py` (Session 10), which normalises the attributes to 0 to 1 as the WEKA package does.

Output

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

```
1 employee.arff: 30 instances
2
3 == epsilon = 0.15, minPoints = 3 ==
4 cluster 0: instances [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
5 noise (unclustered): instances [20, 29]
```

The corresponding WEKA block:

```

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

```

The two unclustered rows are the deliberate outliers of the file: row 29 (age 58, experience 35, salary 250) and row 20 (age 46, experience 22, salary 120). k-means with $k = 2$ on the same data puts row 29 in a cluster of its own or drags the senior cluster's centroid towards it; DBSCAN simply reports it as noise.

Explanation

Property	k-means	DBSCAN	BIRCH	CLARANS
Needs k in advance	yes	no	for the final phase	yes
Passes over the data	many	one per point (with an index)	one, plus optional refinement	samples, many
Cluster shape	spherical	arbitrary	spherical (CF radius)	spherical around medoids
Handles noise	no, absorbs it	labels it	outlier leaves set aside	resistant (medoids)
Memory	all rows	rows plus neighbour index	CF tree only	sample only

- DBSCAN's weakness is one global `epsilon` : clusters of different density need different radii, which OPTICS (in the same WEKA package) fixes by ordering points by reachability distance instead of fixing a radius.
- BIRCH's weakness is order dependence and its preference for round clusters, because the CF radius is a single number.
- CLARANS is a compromise between PAM (exact but quadratic) and CLARA (fast but fixed samples); its quality depends on `numLocal` and `maxneighbor` .

Question 20

Problem Statement

■ Write in lab record

Implement simple K-Means Algorithm to demonstrate the clustering rule on the following datasets:

1. `iris.arff`
2. `student.arff`

Solution

■ Write in lab record

Steps

1. **Open file...**, `data/iris.arff` (150 instances, four numeric attributes, class with three species of 50 each).
2. **Cluster** tab, **Choose**, `SimpleKMeans` . Click the name: `numClusters` 3, `seed` 10, `distanceFunction` EuclideanDistance (which normalises every attribute to 0 to 1 before computing distances, so the printed "Within cluster sum of squared errors" is on the normalised scale). Under Cluster mode pick **Use training set, Start**.
3. Read the output: number of iterations, the SSE line, the centroid table (Full Data column, then one column per cluster, with the count in brackets) and the Clustered Instances block.
4. Switch Cluster mode to **Classes to clusters evaluation**, class `class` , **Start**. The output adds a matrix of species against cluster, the majority species assigned to each cluster, and the "Incorrectly clustered instances" line.

5. Repeat for `student.arff` with **Ignore attributes** set to `stream` and `result`, `numClusters` 2, and Classes to clusters evaluation on `result`.
6. `kmeans.py` reimplements SimpleKMeans (normalised Euclidean distance, k random starting rows, best of 10 seeds) and prints the same tables. Because the full 150-row iris file is long, the script ships with `iris_sample.arff`: rows 1 to 10, 51 to 60 and 101 to 110 of the standard data, ten per species; the full WEKA numbers are quoted below it.

Program

- Lab record: every tab is one file of the answer. Write all of them.

`iris_sample.arff`

iris_sample.arff

```
1 % iris_sample.arff : rows 1-10, 51-60 and 101-110 of the standard iris data
2 @relation iris_sample
3
4 @attribute sepallength numeric
5 @attribute sepalwidth numeric
6 @attribute petallength numeric
7 @attribute petalwidth numeric
8 @attribute class {Iris-setosa,Iris-versicolor,Iris-virginica}
9
10 @data
11 5.1,3.5,1.4,0.2,Iris-setosa
12 4.9,3.0,1.4,0.2,Iris-setosa
13 4.7,3.2,1.3,0.2,Iris-setosa
14 4.6,3.1,1.5,0.2,Iris-setosa
15 5.0,3.6,1.4,0.2,Iris-setosa
16 5.4,3.9,1.7,0.4,Iris-setosa
17 4.6,3.4,1.4,0.3,Iris-setosa
18 5.0,3.4,1.5,0.2,Iris-setosa
19 4.4,2.9,1.4,0.2,Iris-setosa
20 4.9,3.1,1.5,0.1,Iris-setosa
21 7.0,3.2,4.7,1.4,Iris-versicolor
22 6.4,3.2,4.5,1.5,Iris-versicolor
23 6.9,3.1,4.9,1.5,Iris-versicolor
24 5.5,2.3,4.0,1.3,Iris-versicolor
25 6.5,2.8,4.6,1.5,Iris-versicolor
26 5.7,2.8,4.5,1.3,Iris-versicolor
27 6.3,3.3,4.7,1.6,Iris-versicolor
28 4.9,2.4,3.3,1.0,Iris-versicolor
29 6.6,2.9,4.6,1.3,Iris-versicolor
30 5.2,2.7,3.9,1.4,Iris-versicolor
31 6.3,3.3,6.0,2.5,Iris-virginica
32 5.8,2.7,5.1,1.9,Iris-virginica
33 7.1,3.0,5.9,2.1,Iris-virginica
34 6.3,2.9,5.6,1.8,Iris-virginica
35 6.5,3.0,5.8,2.2,Iris-virginica
36 7.6,3.0,6.6,2.1,Iris-virginica
37 4.9,2.5,4.5,1.7,Iris-virginica
38 7.3,2.9,6.3,1.8,Iris-virginica
39 6.7,2.5,5.8,1.8,Iris-virginica
40 7.2,3.6,6.1,2.5,Iris-virginica
```

Output

```
python3 kmeans.py iris_sample.arff 3 (run for real):
```

```

1 iris_sample.arff: 30 instances, numeric attributes ['sepalength', 'sepalwid
2
3 k   SSE (normalised) iterations  cluster sizes
4 2       3.4477           6 [19, 11]
5 3       1.9136          10 [14, 10, 6]
6 4       1.3931           3 [6, 9, 5, 10]
7 5       1.1310           7 [8, 5, 6, 6, 5]
8
9 ≡≡≡ k = 3: centroids in original units (WEKA layout) ≡≡≡
10 Attribute      Full Data  Cluster 0  Cluster 1  Cluster 2
11              (30)      (14)      (10)      (6)
12 sepalength      5.8433      6.7643      4.8600      5.3333
13 sepalwidth      3.0400      3.0500      3.3100      2.5667
14 petallength      3.8633      5.4357      1.4500      4.2167
15 petalwidth      1.2133      1.8286      0.2200      1.4333
16
17 Within cluster sum of squared errors: 1.9136
18 Clustered Instances 0  14 (47%)
19 Clustered Instances 1  10 (33%)
20 Clustered Instances 2   6 (20%)
21
22 ≡≡≡ Classes to Clusters ≡≡≡
23   0   1   2 ← assigned to cluster
24   0  10   0 | Iris-setosa
25   6   0   4 | Iris-versicolor
26   8   0   2 | Iris-virginica
27 Cluster 0 ← Iris-virginica
28 Cluster 1 ← Iris-setosa
29 Cluster 2 ← Iris-versicolor
30 Incorrectly clustered instances : 8    26.6667 %

```

On the full `iris.arff` WEKA prints, for $k = 3$ and seed 10: 6 iterations, within cluster sum of squared errors 6.998, clusters of 61, 50 and 39 instances with centroids (5.8885, 2.7377, 4.3967, 1.418), (5.006, 3.418, 1.464, 0.244) and (6.8462, 3.0821, 5.7026, 2.0718), and under Classes to clusters evaluation 17 incorrectly clustered instances (11.33 percent): all 50 setosa in one cluster, the errors all between versicolor and virginica.

python3 kmeans.py student.arff 2 (numeric attributes only):

```

1  k  SSE (normalised) iterations  cluster sizes
2  2          2.2219          2  [15, 15]
3  3          1.0154          8  [12, 9, 9]
4  4          0.6540          5  [6, 8, 6, 10]
5  5          0.5215          3  [7, 2, 9, 6, 6]
6
7  === k = 2: centroids in original units (WEKA layout) ===
8  Attribute          Full Data  Cluster 0  Cluster 1
9                      (30)      (15)      (15)
10 hours              9.4333     5.8667     13.0000
11 attendance         70.7000     58.7333     82.6667
12 internal           16.6333     10.7333     22.5333
13
14 Within cluster sum of squared errors: 2.2219
15 Clustered Instances  0  15 (50%)
16 Clustered Instances  1  15 (50%)
17
18 === Classes to Clusters ===
19      0    1  ← assigned to cluster
20      5   14 | pass
21     10    1 | fail
22 Cluster 0 ← fail
23 Cluster 1 ← pass
24 Incorrectly clustered instances : 6    20.0000 %

```

Explanation

- k-means alternates two steps until nothing moves: assign each row to the nearest centroid, then move each centroid to the mean of its rows. Each step can only lower the SSE of the formula sheet, so it converges, but to a local minimum that depends on the starting rows; hence WEKA's `seed` and the script's best-of-ten restarts.
- The centroid table is read per column: cluster 1 of the iris sample has petal length 1.45 and petal width 0.22, the small-petal setosa profile, and contains all ten setosa rows. Clusters 0 and 2 split versicolor and virginica by size rather than by species, which is why 8 of 20 non-setosa rows count as incorrectly clustered: those two species overlap in all four measurements.
- "Incorrectly clustered" is computed after mapping each cluster to its majority class; it is not an accuracy, because the algorithm never saw the class. It measures how well the natural groups

line up with the labels.

- On `student`, two clusters split the rows at roughly attendance 70 and internal 16; cluster 0 (low hours, low attendance, low marks) is 10 fail and 5 pass, so the unsupervised split recovers the pass/fail label for 24 of 30 rows.

Question 21

Problem Statement

■ Write in lab record

Perform the following:

- Load each dataset into WEKA and run simple k-means clustering algorithm with different values of k (number of desired clusters).
- Study the clusters formed.
- Observe the sum of squared errors and centroids, and derive insights.
- Explore other clustering techniques available in WEKA.
- Explore visualization features of WEKA to visualize the clusters.
- Derive interesting insights and explain.

Solution

■ Write in lab record

Steps

1. With `iris.arff` loaded, run `SimpleKMeans` four times with `numClusters` 2, 3, 4 and 5 (seed 10 each time) and copy the SSE, the iteration count and the cluster sizes into a table. Do the same for `student.arff` (nominal attributes ignored). The script prints this table in one go.
2. For each run, read the centroid table and write one sentence per cluster describing it in the units of the data.
3. **Other techniques.** Run `EM` (choose `numClusters` -1 to let it pick k by cross-validation, then read the log likelihood), `HierarchicalClusterer` (Session 10), `FarthestFirst` (a fast k-means-like seeding), `Cobweb` (incremental, builds a tree of concepts), `Canopy` (a cheap pre-clustering used to seed k-means on big data) and `MakeDensityBasedClusterer` wrapped around `SimpleKMeans` (gives a log likelihood so runs with different k can be compared).

4. Visualisation. Right-click a result, **Visualize cluster assignments**. Set X to `petalength`, Y to `petalwidth`, Colour to `cluster`; move the Jitter slider to separate overlapping points. Click a point to see its row. Then change Colour to `class` to compare clusters with species side by side. On the **Visualize** tab of the Explorer, the scatter-plot matrix shows every attribute pair at once; the petal pair separates the species best, the sepal pair worst.

Output

SSE against k (normalised attributes, best of ten seeds, from `kmeans.py`):

k	iris sample SSE	drop	student SSE	drop
2	3.4477		2.2219	
3	1.9136	1.534	1.0154	1.207
4	1.3931	0.521	0.6540	0.361
5	1.1310	0.262	0.5215	0.133

Full `iris.arff` in WEKA, seed 10: k = 2 gives SSE 12.14 with clusters of 100 and 50; k = 3 gives 6.998 (61, 50, 39); record k = 4 and 5 from your run and expect the drop to keep shrinking.

Cluster descriptions for iris, k = 3 (from the centroid table above): cluster 1, small flowers with petals about 1.5 by 0.2 cm, all setosa; cluster 0, the largest flowers, petals 5.4 by 1.8 cm, mostly virginica; cluster 2, medium flowers with narrow sepals, petals 4.2 by 1.4 cm, mostly versicolor.

Cluster descriptions for student, k = 3: 12 students studying about 13 hours with 83 percent attendance and internal marks near 23 (all pass); 9 studying about 4 hours with 50 percent attendance and marks near 8 (all fail); 9 in between (hours 8, attendance 68, marks 14), where the passes and fails mix.

Explanation

- **The elbow.** SSE always falls as k rises (a cluster can only get tighter when it is split), so the smallest SSE is never the answer; the useful k is where the drop stops being large. For the iris sample the drop from k = 2 to 3 is 1.53 and from 3 to 4 only 0.52, so k = 3 is the elbow, matching the three species. For student the elbow is also at 3, but two clusters already line up with pass and fail; the third cluster is the borderline group.

- **Centroids as summaries.** A centroid is the mean row of its cluster, so it reads like a typical member; comparing the cluster column with the Full Data column shows which attributes define the cluster (petal size for iris, all three for student, in the same direction).
- **Insights.** Setosa is separable by petal size alone; versicolor and virginica overlap, so any clustering of iris will misplace about a sixth of them. On student, unsupervised clustering recovers the pass/fail split without ever seeing the label, which says the label is largely determined by the numeric attributes.
- **Which other technique to prefer.** EM gives soft memberships and a likelihood to compare k values; hierarchical clustering gives the whole tree so k is chosen afterwards; DBSCAN is the one to use when noise must be labelled. Cobweb and Canopy are for streaming or very large data.

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 $\frac{1}{|A \cup B|} \sum d(a, b)$.

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: Why does k-means need normalised attributes? **A:** Euclidean distance would otherwise be dominated by the attribute with the largest range, `attendance` on student or `salary` on employee.

Q: What does "Within cluster sum of squared errors" measure? **A:** The total squared distance from every row to its own centroid on the normalised scale; lower means tighter clusters.

Q: Why does SSE always fall when k increases? **A:** Any cluster can be split into two whose members are closer to their new centroids, so the minimum can only decrease.

Q: How is "incorrectly clustered instances" computed? **A:** Each cluster is mapped to its majority class and every member of another class is counted as incorrect.

Q: What are epsilon and minPoints in DBSCAN? **A:** The neighbourhood radius and the minimum neighbours for a point to be a core point; rows reachable from no core point are noise.

Q: What is a clustering feature in BIRCH? **A:** The triple (N, LS, SS) for a subcluster, from which its centroid and radius follow without the rows.

Q: Why does the seed change the k-means result? **A:** The starting centroids are random rows and the algorithm converges to the nearest local minimum of SSE.

Q: Name a WEKA clusterer that does not need k. **A:** DBSCAN, EM with `numClusters -1`, Cobweb, or HierarchicalClusterer read at any cut.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Leaving the class attribute in as an input; k-means will happily cluster on it and the evaluation becomes circular.
- Comparing SSE values between different k and picking the smallest; the elbow is the answer, not the minimum.
- Comparing SSE between WEKA and a script that does not normalise; WEKA's figure is on the 0 to 1 scale.
- Reporting "incorrectly clustered" as classification accuracy.
- Running DBSCAN on unnormalised data with epsilon 0.15 and getting every row as noise.
- Confusing cluster numbers between runs; cluster 0 in one run may be cluster 2 in the next.

Session Summary

■ Write in lab record

- Question 19: DBSCAN, BIRCH and CLARANS explained for large noisy databases, comparison table written, DBSCAN run on employee.arff (epsilon 0.15, minPoints 3: one cluster of 28 and two noise rows).
- Question 20: SimpleKMeans on iris (k = 3: SSE 1.9136 on the 30-row sample, 6.998 on the full file, setosa perfectly separated) and on student (k = 2 recovers pass/fail for 24 of 30); `kmeans.py` and `iris_sample.arff` recorded.
- Question 21: SSE against k for k = 2 to 5 tabulated with the elbow at 3, centroids interpreted, EM, FarthestFirst, Cobweb, Canopy and HierarchicalClusterer tried, clusters visualised with Visualize cluster assignments.

[✎ Edit this page](#)

< Previous
Session 8

Next >
Session 10