

**Batch: H-ADS (H2\_3) Roll No.: 16010122221**

**Experiment No 08**

**Title: To implement clustering using K-means algorithm**

**AIM:** To understand the Clustering algorithm.

**Expected Outcome of Experiment:**

**CO4:** Understand the basic concept and techniques of Machine Learning clustering.

**Books/ Journals/ Websites referred:**

1. <https://uc-r.github.io/kmeans_clustering>
2. <https://en.wikipedia.org/wiki/K-means_clustering>

**Pre Lab/ Prior Concepts:**

**K-means Algorithm**

K-Means Clustering is an [Unsupervised Learning algorithm](https://www.javatpoint.com/unsupervised-machine-learning), which groups the unlabelled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, andfor K=3, there will be three clusters, and so on.

It allows us to cluster the data into different groups and a convenient way to discoverthe categories of groups in the unlabelled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data pointand their corresponding clusters.

The algorithm takes the unlabelled dataset as input, divides the dataset into k-number ofclusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means [clustering](https://www.javatpoint.com/clustering-in-machine-learning) algorithm mainly performs two tasks:

Determines the best value for K centre points or centroids by an iterative process.

Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.



Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

**Description of the dataset used in implementation:**

**Iris dataset**

**Code ( R code):**

# K-Means Clustering # Importing the dataset

dataset = read.csv('Iris.csv')

dataset = dataset[2:5]

# Splitting the dataset into the Training set and Test set # install.packages('caTools')

# library(caTools) # set.seed(123)

# split = sample.split(dataset$DependentVariable, SplitRatio = 0.8) # training\_set = subset(dataset, split == TRUE)

# test\_set = subset(dataset, split == FALSE)

# Feature Scaling

# training\_set = scale(training\_set) # test\_set = scale(test\_set)

# Using the elbow method to find the optimal number of clusters set.seed(6)

wcss = vector()

for (i in 1:10) wcss[i] = sum(kmeans(dataset, i)$withinss) plot(1:10,

wcss, type = 'b',

main = paste('The Elbow Method'), xlab = 'Number of clusters',

ylab = 'WCSS')

# Fitting K-Means to the dataset set.seed(29) kmeans = kmeans(x = dataset, centers = 3) y\_kmeans = kmeans$cluster

# Visualising the clusters library(cluster) clusplot(dataset, y\_kmeans,

lines = 0,

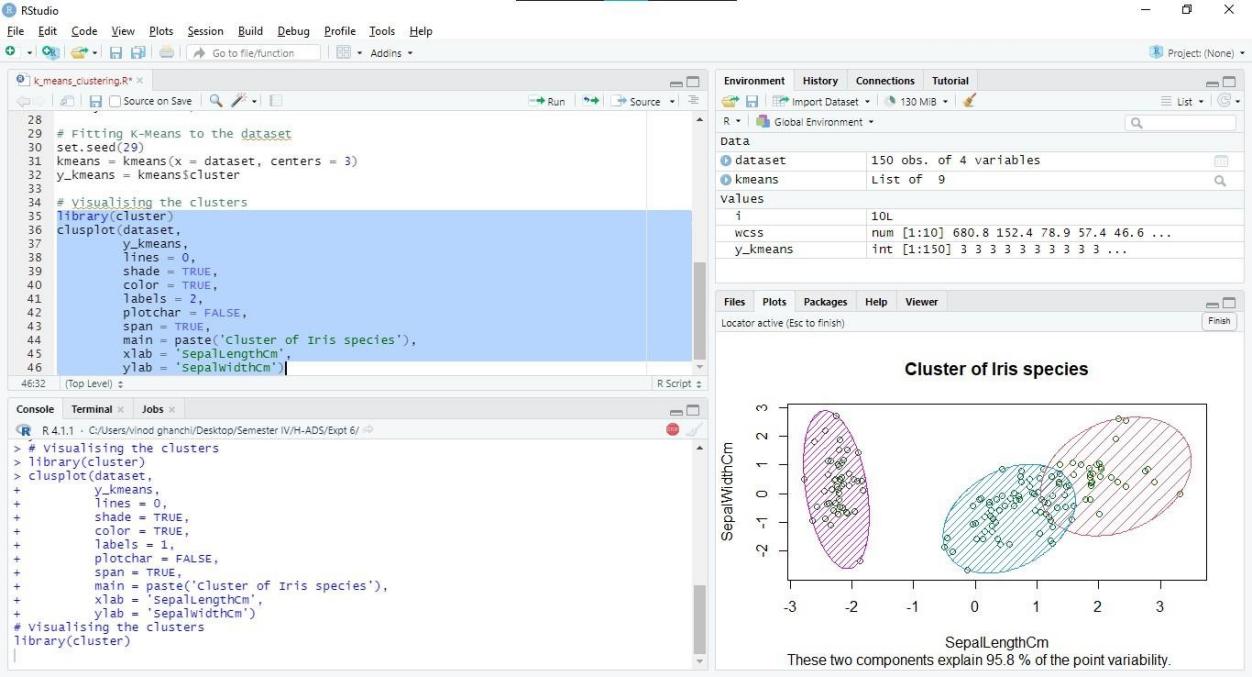
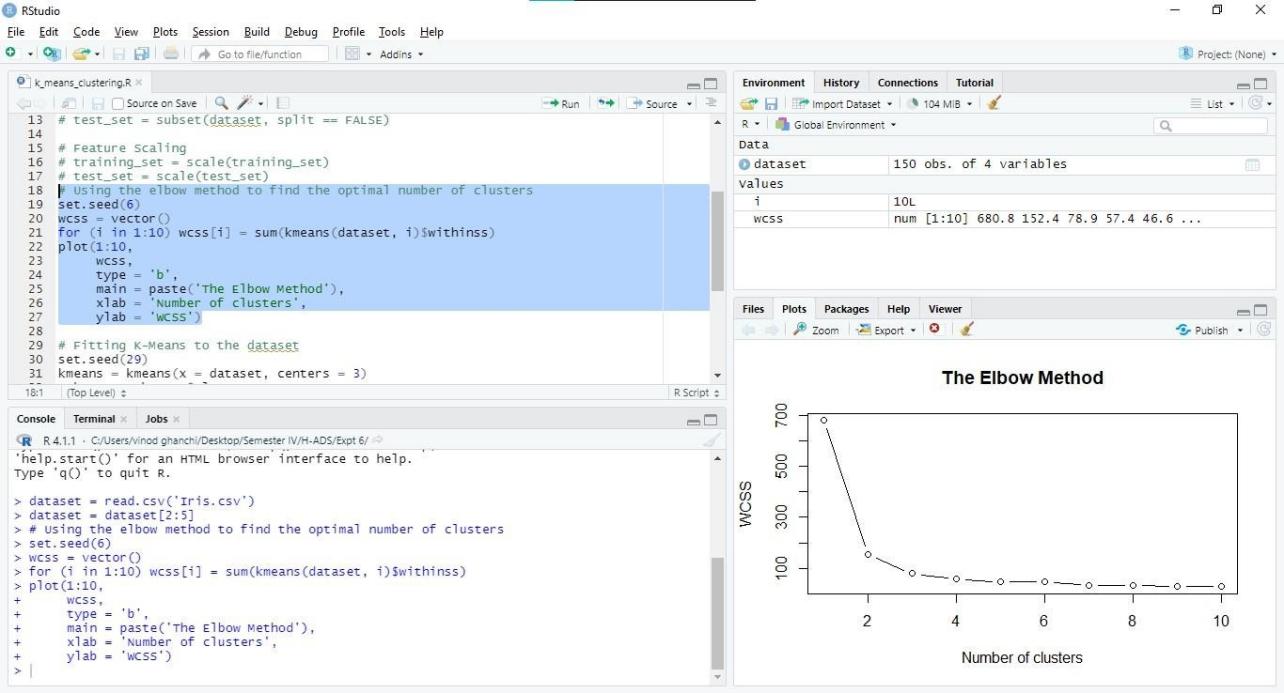


shade = TRUE, color = TRUE, labels = 2, plotchar = FALSE, span = TRUE,

main = paste('Cluster of Iris species'), xlab = 'SepalLengthCm',

ylab = 'SepalWidthCm')

**Output:**





**Conclusion:**

The experiment aimed to implement clustering using the K-means algorithm, a widely used unsupervised machine learning technique. Through iterative assignment and reassignment of data points to clusters based on centroid proximity, K-means effectively partitions the dataset into distinct groups. Despite its simplicity, K-means offers versatility in various domains, enabling efficient data segmentation for tasks such as customer segmentation, image compression, and anomaly detection. Overall, the experiment demonstrated the practical applicability and effectiveness of the K-means algorithm in clustering analysis.

**Post Lab Questions**

1. **Explain Apriori algorithm with suitable Numerical based example.**

**Answer:**

The Apriori algorithm is a classic algorithm used for association rule mining in data mining and machine learning. It's particularly useful for discovering frequent item sets in transactional databases. Here's an explanation of the Apriori algorithm with a numerical-based example:

Let's consider a hypothetical transactional dataset representing purchases in a grocery store:

|  |  |
| --- | --- |
| Transaction ID | Items Purchased |
| 1 | Bread, Milk, Eggs |
| 2 | Bread, Butter |
| 3 | Milk, Eggs |
| 4 | Bread, Milk, Butter |
| 5 | Bread, Milk |

In this dataset, each row represents a transaction, and the items purchased are listed.

The Apriori algorithm works in two main steps:

* + Finding frequent item sets: The algorithm starts by finding all frequent item sets, i.e., sets of items that occur together frequently in transactions. It does this by iteratively generating candidate item sets of increasing size and pruning those that do not meet a minimum support threshold.
  + Generating association rules: Once frequent item sets are found, the algorithm generates association rules based on these sets. An association



rule typically has the form "If {itemset1} then {itemset2}". The confidence of a rule is calculated as the ratio of the support of the combined item set to the support of the antecedent item set.

Let's illustrate this with an example:

* + Step 1: Finding frequent item sets

Iteration 1: Find all frequent item sets of size 1 (single items) with minimum support, say 40%.

Bread: 4

Milk: 4

Eggs: 2

Butter: 2

(Counts represent the number of transactions containing each item) Iteration 2: Generate candidate item sets of size 2 based on frequent item sets from the previous iteration.

{Bread, Milk}: 3

{Bread, Eggs}: 1

{Bread, Butter}: 2

{Milk, Eggs}: 2

(Counts represent the number of transactions containing each item set) Pruning: Remove item sets that do not meet the minimum support threshold (e.g., 40%).

* + Step 2: Generating association rules

Generate association rules from the frequent item sets found in Step 1. Calculate the confidence of each rule.

For example, if we have {Bread, Milk} as the antecedent and {Eggs} as the consequent, the confidence would be calculated as the ratio of the support of {Bread, Milk, Eggs} to the support of {Bread, Milk}.

Using these steps, the Apriori algorithm efficiently identifies frequent item sets and generates meaningful association rules from transactional data, enabling insights into customer behavior and purchasing patterns.



1. **Write a program to implement the Apriori algorithm.**

**Answer:**

import java.util.ArrayList; import java.util.HashSet; import java.util.List; import java.util.Set;

public class Apriori {

public static List<List<String>> generateCandidates(List<List<String>> prevCandidates, int k) {

List<List<String>> candidates = new ArrayList<>(); int n = prevCandidates.size();

for (int i = 0; i < n; i++) {

for (int j = i + 1; j < n; j++) {

if (prevCandidates.get(i).subList(0, k - 1).equals(prevCandidates.get(j).subList(0, k - 1))) {

List<String> candidate = new ArrayList<>(prevCandidates.get(i)); candidate.add(prevCandidates.get(j).get(k - 1)); candidates.add(candidate);

}

}

}

return candidates;

}

public static List<List<String>> pruneCandidates(List<List<String>> candidates, List<List<String>> prevFrequentSets) {

List<List<String>> prunedCandidates = new ArrayList<>(); for (List<String> candidate : candidates) {

boolean isSubsetInPrevFrequentSets = true; for (int i = 0; i < candidate.size(); i++) {

List<String> subset = new ArrayList<>(candidate); subset.remove(i);

if (!prevFrequentSets.contains(subset)) { isSubsetInPrevFrequentSets = false; break;

}

}

if (isSubsetInPrevFrequentSets) { prunedCandidates.add(candidate);

}

}

return prunedCandidates;

}



public static int calculateSupport(List<List<String>> transactions, List<String> candidate) {

int count = 0;

for (List<String> transaction : transactions) { if (transaction.containsAll(candidate)) {

count++;

}

}

return count;

}

public static List<List<String>> apriori(List<List<String>> transactions, double minSupport) {

List<List<String>> frequentSets = new ArrayList<>(); List<List<String>> candidates = new ArrayList<>(); Set<String> items = new HashSet<>();

for (List<String> transaction : transactions) { items.addAll(transaction);

}

for (String item : items) {

List<String> candidate = new ArrayList<>(); candidate.add(item); candidates.add(candidate);

}

int k = 1;

while (!candidates.isEmpty()) {

candidates = generateCandidates(candidates, k); candidates = pruneCandidates(candidates, frequentSets);

List<List<String>> frequentCandidates = new ArrayList<>(); for (List<String> candidate : candidates) {

int support = calculateSupport(transactions, candidate);

double supportPercentage = (double) support / transactions.size(); if (supportPercentage >= minSupport) {

frequentCandidates.add(candidate);

}

}

frequentSets.addAll(frequentCandidates); k++;

}

return frequentSets;

}

public static void main(String[] args) { List<List<String>> transactions = new ArrayList<>(); transactions.add(List.of("Bread", "Milk", "Eggs")); transactions.add(List.of("Bread", "Butter"));



transactions.add(List.of("Milk", "Eggs")); transactions.add(List.of("Bread", "Milk", "Butter")); transactions.add(List.of("Bread", "Milk"));

double minSupport = 0.4;

List<List<String>> frequentItemsets = apriori(transactions, minSupport); System.out.println("Frequent Item Sets:");

for (List<String> itemset : frequentItemsets) { System.out.println(itemset);

}

}

}