Department of
Computer Science and Engineering

Title: K-Means Clustering Algorithm Implementation

Artificial Intelligence Lab

CSE 404

Green University of Bangladesh

1 Objective(s)

- To understand K-Means algorithm to solve clustering of points
- To understand how to assign clusters and steps to finalize the cluster using Euclidean distance

2 Problem analysis

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled 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, and for 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 discover the categories of groups in the unlabeled 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 point and their corresponding clusters.

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

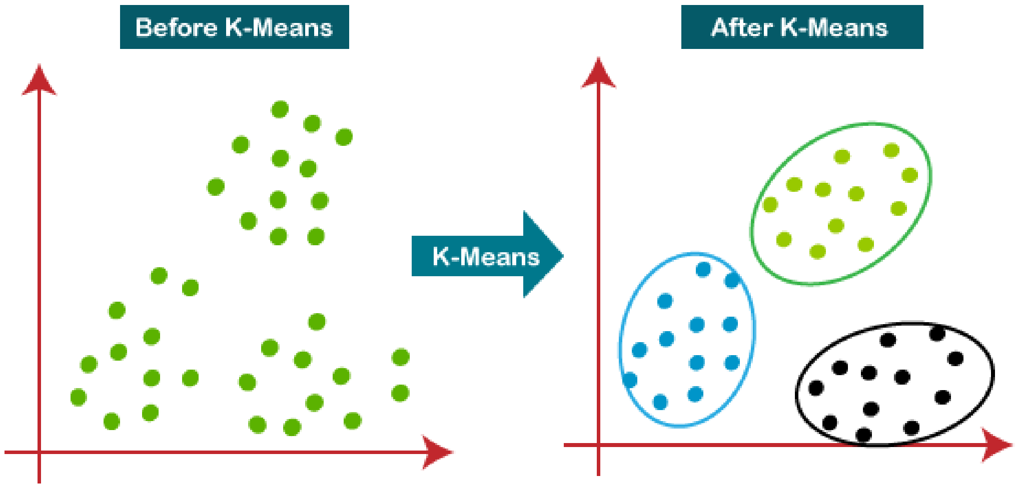


Figure 1: Clustering using K-Means

3 Algorithm

Step 1: Select the number K to decide the number of clusters.

Step 2: Select random K points or centroids. (It can be other from the input dataset).

Step 3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step 4: Calculate the variance and place a new centroid of each cluster.

Step 5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step 6: If any reassignment occurs, then go to step-4 else go to FINISH.

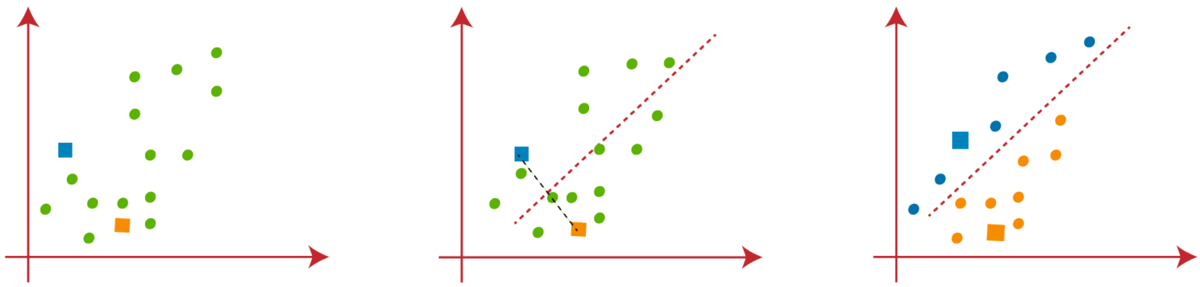
3.1 Working example of K-Means clustering

Let's take number k of clusters, i.e., K=2, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.

We need to choose some random k points or centroid to form the cluster like in figure 2a. These points can be either the points from the dataset or any other point. So, here we are selecting the below two points as k points, which are not the part of our dataset.

Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids (figure 2b).

From the figure 2b, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization which is given in figure 2c



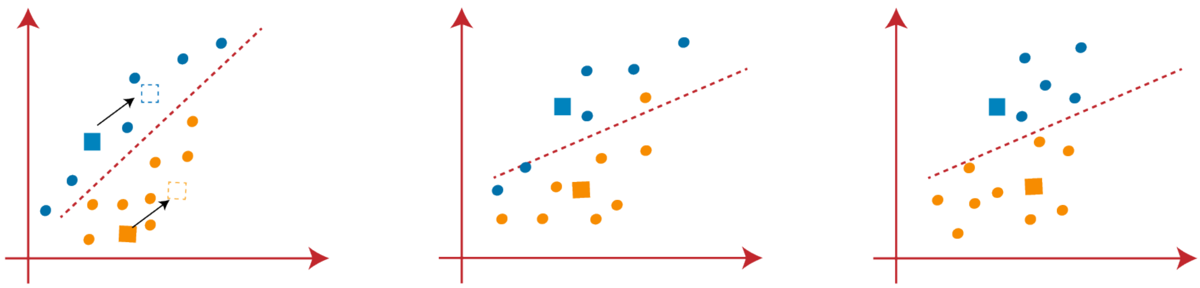
(a) (b) (c)

Figure 2

As we need to find the closest cluster, so we will repeat the process by choosing a new centroid. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as in figure 3a

Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like as in figure 3b.

From the figure 3b, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So in figure 3c, these three points will be assigned to new centroids.



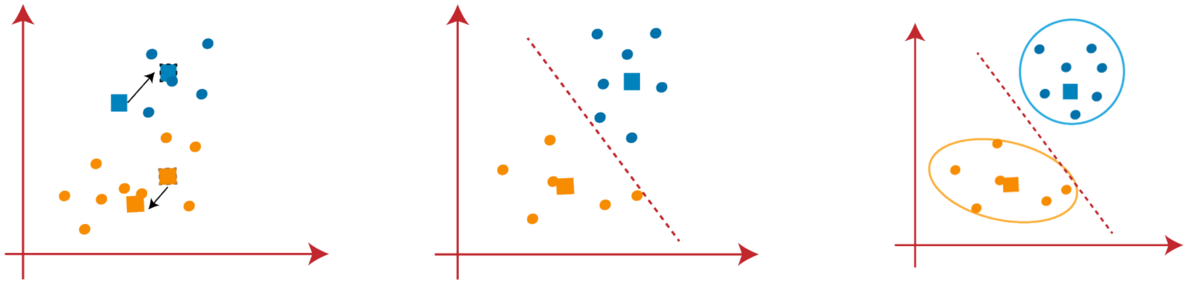
(a) (b) (c)

Figure 3

As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points (figure 4a)

We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the figure 4b:

As we got the new centroids so again will draw the median line and reassign the data points. So, form figure 4c, we can see there are no dissimilar data points on either side of the line, which means our model is formed.



(a) (b) (c)

Figure 4

4 Implementation in Java

```java
package k_means;

import java.util.ArrayList;
import java.util.Random;
import java.util.Scanner;

/**
 *
 * @author Jargis Ahmed
 */
class points {

    int x, y, cl;

    points(int x, int y) {  // points on the grid (x,y)
        this.x = x;
        this.y = y;
    }

    void setClass(int c) {  // cluster number in which this (x,y) point reside
        cl = c;
    }
}

class St {

    int pt, ks;     // pt is the number of points and ks is the number of clusters
    points[] p, k;
    ArrayList<points[]> fkc;
    Random rand = new Random();

    St(int points, int clusters) {
        pt = points;
        ks = clusters;
        fkc = new ArrayList<points[]>();
        Start();
    }

    void Start() {
```

```java
        p = new points[pt]; // create random points coordinate
        for (int i = 0; i < pt; i++) {
            int x = rand.nextInt(pt);
            int y = rand.nextInt(pt);
            p[i] = new points(x, y);
        }
        k = new points[ks]; // create random cluster points coordinate
        for (int i = 0; i < ks; i++) {
            int x = rand.nextInt(pt);
            int y = rand.nextInt(pt);
            k[i] = new points(x, y);
        }
        int i, j;
        double min;
        int count = 0;
        while (true) {
            // starting cluster allocation of point p(x,y) based on minimum distance from each cluster points
            for (i = 0; i < pt; i++) {
                for (j = 0, min = 10000; j < ks; j++) {
                    double d1 = Math.sqrt(Math.pow((double) (k[j].x - p[i].x), 2) + Math.pow((double) (k[j].y - p[i].y), 2));
                    if (d1 < min) {
                        p[i].setClass(j);
                        min = d1;
                    }
                }
            }
            points[] kdup = new points[ks];     // creating duplicate set of cluster points to store starting cluster coordinates
            for (i = 0; i < ks; i++) {
                kdup[i] = new points(k[i].x, k[i].y);
            }
            // calculating mean for each points in different clusters
            for (j = 0; j < ks; j++) {
                int x = 0, y = 0, ci = 0;
                for (i = 0; i < pt; i++) {
                    if (p[i].cl == j) {
                        x += p[i].x;
                        y += p[i].y;
                        ci++;
                    }
                }
                if (ci != 0) { // allocating the mean as new cluster point coordinate
                    k[j].x = x / ci;
                    k[j].y = y / ci;
                }
            }
            int err = 0;
            // claculating error between previous and present cluster points coordinates
            for (i = 0; i < ks; i++) {
                err += (k[i].x - kdup[i].x) + (k[i].y - kdup[i].y);
            }
            count++;
            // 0 error between previous and present cluster points coordinates indicates clustring is finalized
```

```java
            if (err == 0) {
                break;
            }
        }

        double IntraDis;
        for (i = 0; i < ks; i++) {
            for (j = 0, IntraDis = 0; j < pt; j++) {
                if (p[j].cl == i) {
                    IntraDis += Math.sqrt(Math.pow((double) (k[i].x - p[j].x),
                        2) + Math.pow((double) (k[i].y - p[j].y), 2));
                }
            }
            System.out.println("Cluster " + (i + 1) + " Intra-distance = " +
                IntraDis);
        }

        for(i=0;i<ks;i++)
        {
            for(j=0;j<p.length;j++)
            {
                if(p[j].cl==i)
                {
                    System.out.println("Point ("+p[j].x+", "+p[j].y+") "+ "
                        Cluster - "+(+p[j].cl+1));
                }
            }

        }
        //System.out.println("count " + count);
        differClass();
        // print a 2D graph alike matrix showing the points and its cluster
        //printGrapgh();
    }

    void differClass() {
        int kval[] = new int[ks];
        int i, j, k;
        for (i = 0; i < pt; i++) {
            kval[p[i].cl]++;
        }
        for (i = 0; i < ks; i++) {
            points[] c1 = new points[kval[i]];
            for (j = 0, k = 0; j < pt; j++) {
                if (p[j].cl == i) {
                    c1[k++] = p[j];
                }
            }
            fkc.add(c1);
            //System.out.println(kval[i]);
        }
        //System.out.println(fkc.size());
        System.out.println();
    }
}

public class K_Means {

```

```
    public static void main(String[] args) {
        // TODO code application logic here
        Scanner sc = new Scanner(System.in);
        System.out.println("Insert number of points");
        int points = sc.nextInt();
        System.out.println("Insert number of clusters");
        int clusters = sc.nextInt();
        St s = new St(points, clusters);
    }
}
```

5 Sample Input/Output (Compilation, Debugging & Testing)

Input:

```
Insert number of points
20
Insert number of clusters
4
```

Output:

```
Cluster 1 Intra-distance = 30.904902140081614
Cluster 2 Intra-distance = 13.93010659580075
Cluster 3 Intra-distance = 11.404929644971588
Cluster 4 Intra-distance = 14.246537600251443
Point (19, 8) Cluster - 1
Point (12, 0) Cluster - 1
Point (17, 7) Cluster - 1
Point (19, 7) Cluster - 1
Point (10, 0) Cluster - 1
Point (19, 0) Cluster - 1
Point (6, 8) Cluster - 2
Point (4, 3) Cluster - 2
Point (5, 8) Cluster - 2
Point (6, 5) Cluster - 2
Point (1, 4) Cluster - 2
Point (19, 11) Cluster - 3
Point (18, 13) Cluster - 3
Point (17, 13) Cluster - 3
Point (15, 19) Cluster - 3
Point (8, 15) Cluster - 4
Point (11, 17) Cluster - 4
Point (6, 17) Cluster - 4
Point (7, 12) Cluster - 4
Point (9, 11) Cluster - 4

A 16 3
B 4 5
C 17 14
D 8 14
```

6 Discussion & Conclusion

Based on the focused objective(s) to understand the K-Means clustering problem and its steps to assign different points under some cluster points or centroids, the additional lab exercise will increase confidence towards the fulfilment of the objectives(s).

7 Lab Exercise (Submit as a report)

- Rewrite K-Means clustering algorithm where -
  - Construct a data file containing 100 cartesian points P(x, y) and 10 cluster points C(x, y)
  - Calculate distance using "Manhattan distance" method
  - Make a 2D visualization of points and its cluster points by only using print functionality. (Hint: Use a 2D Matrix which will hold the points information and clusters. Print the matrix content accordingly)

8 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected for lab exercise.