Υπολογιστική Νοημοσύνη

Βάντσης Δημήτρης ΑΜ:2643

K-means

Η υλοποίηση του αλγόριθμου K-means έγινε στη γλώσσα προγραμματισμού Java.

Για να γίνει το compile του προγράμματος, θα πρέπει να δοθούν στο τερματικό οι εντολές:

javac SDO.java

javac Kmeans.java

και για να τρέξει το πρόγραμμα, θα πρέπει να δοθούν στο τερματικό οι εντολές:

java SDO

java Kmeans

Με βάση την εκφώνηση ο αριθμός των ομάδων είναι M = 3, 6, 9, 12 και θα πρέπει να γίνουν 15 τρεξίματα του αλγόριθμου για κάθε M.

Το πρόγραμμα είναι χωρισμένο σε 2 files, οπότε στα print screen που ακολουθούν,

- η σταθερά numberOfPoints αποτελεί το πλήθος των σημείων και από την εκφώνηση είναι ίσο με 1200.
- η σταθερά runs είναι τα τρεξίματα του αλγόριθμου και είναι ίση με 15.
- ο αριθμός των ομάδων M έχει οριστεί ως ένας πίανακας με τιμές 3, 6, 9 και 12. Ο λόγος είναι γιατί θέλουμε να γίνουν 15 τρεξίματα για κάθε M, οπότε αφού ολοκληρωθεί το πρώτο for loop με τα runs, θα πάρει την επόμενη τιμή του M και θα συνεχίσει. Έτσι, αν θέλουμε να αλλάξουμε τον αριθμό εκτελέσεων γίνεται με την σταθερά runs και τον αριθμό των ομάδων από τις τιμές του πίνακα M.
- για το σύνολο δεδομένων, γνωρίζοντας από την εκφώνηση ότι υπάρχουν συγκεκριμένα σημεία, φτιάξαμε ένα πίνακα totalPoints που περιέχει το πλήθος τους και τον πίνακα coordinates με τις συντεταγμένες στο τετράγωνο. Έτσι, μπορεί να γίνει η αντιστοίχηση των συντεταγμένων με το πλήθος των σημείων.

- για να είναι δυναμικό το πρόγραμμα και να μπορούν εύκολα να γίνουν αλλαγές ή επεκτάσεις, τα σημεία και τα κεντροειδή έγιναν σε arraylists ώστε το πλήθος τους να μην επηρεάζει την μνήμη της δομής δεδομένων.

```
public class Kmeans {
    static final int numberOfPoints = 1200, runs = 15;
    private static List<Point> points = new ArrayList<>();
    private static final List<Centroid> centroids = new ArrayList<>();
    private static final ArrayList <Double> listOfErrors = new ArrayList<>(runs);
    static final int[] M = {3, 6, 9, 12};
```

```
public class SDO {
    static final int[] totalPoints = {150, 150, 150, 150, 150, 75, 75, 75, 75, 150};
    static final double[][] coordinates = {
            {0.8, 1.2, 0.8, 1.2},
            {0.0, 0.5, 0.0, 0.5},
            {1.5, 2.0, 0.0, 0.5},
            {0.0, 0.5, 1.5, 2.0},
            {1.5, 2.0, 1.5, 2.0},
            {0.0, 0.4, 0.8, 1.2},
            {1.6, 2.0, 0.8, 1.2},
            {0.8, 1.2, 0.3, 0.7},
            {0.8, 1.2, 1.3, 1.7},
            {0.0, 2.0, 0.0, 2.0}
    };
```

Για να είναι λειτουργικό το πρόγραμμα θα πρέπει πρώτα να τρέξει η SDO κλάση ώστε να υπάρχει το σύνολο δεδομένων, γιατί η Kmeans κλάση έχει ως δεδομένο ότι υπάρχει το αρχείο αυτό και το διαβάζει. Έτσι, η SDO περιέχει μια main μέθοδο που καλεί την generateDataset με όρισμα το όνομα του αρχείου. Η μέθοδος generateDataset κατασκευάζει το σύνολο δεδομένων με τα σημεία και τις συντεταγμένες, όπου η επιλογή των τυχαίων σημείων γίνεται με την μέθοδο Rand και από την generateRandomPoints που δεσμεύει το πλήθος των σημείων με τις συντεταγμένες.

```
public static void generateDataset(String filename) {
    try (PrintWriter writer = new PrintWriter(new FileWriter(filename))) {
        for (int i = 0; i < totalPoints.length; i++) {
            generateRandomPoints(writer, totalPoints[i], coordinates[i]);
        }
    }
    catch (IOException e) {
        e.printStackTrace(System.err);
    }
}

private static void generateRandomPoints(PrintWriter writer, int totalPoints, double[] coordinates) {
    double minX = coordinates[0], maxX = coordinates[1], minY = coordinates[2], maxY = coordinates[3];
    Random rand = new Random();

    for (int i = 0; i < totalPoints; i++) {
        double x = minX + (maxX - minX) * rand.nextDouble();
        double y = minY + (maxY - minY) * rand.nextDouble();
        writer.println(x + " " + y);
    }
}

public static void main (String [] args) {
    generateDataset( filename: "SDO.txt");
}
```

Ακολουθεί το print screen της main μεθόδου, όπου στην αρχή διαβάζει το σύνολο δεδομένων με την μέθοδο readDataset και για κάθε τιμή του M εφαρμόζει τον K-means στο

ίδιο σύνολο δεδομένων και προσθέτει τα αποτελέσματα σε διαφορετικά αρχεία ώστε να αναπαραστηθούν από την Python.

```
public static void main(String[] args) {
    readDataset();

    for (int currentM : M) {
        runKMeans(currentM);

        try (PrintWriter writer = new PrintWriter(new FileWriter( fileName: "errors.txt",  append: true))) {
            writer.println(currentM + " " + Collections.min(listOfErrors) );
        } catch (IOException e) { e.printStackTrace(System.err); }


        try (PrintWriter writer = new PrintWriter(new FileWriter( fileName: currentM + "centroids.txt"))) {
            for (Centroid centroid : centroids)
                writer.println(centroid.x + " " + centroid.y);
        } catch (IOException e) { e.printStackTrace(System.err); }

        try {
            Thread.sleep( millis: 1000); // delay
        } catch (InterruptedException e) { e.printStackTrace(System.err); }
    }
}
```

Σημαντική παρατήρηση είναι ότι το σύνολο δεδομένων κατασκευάζεται στην αρχή του προγράμματος για μια φορά, οπότε όλα τα M θα επεξεργαστούν τα ίδια σημεία και έτσι τα αποτελέσματα θα είναι έγκυρα.

Για την εγγραφή των αποτελεσμάτων,

- για το αρχείο errors.txt, στον FileWriter δίνεται η τιμή true, ώστε να μπορεί να κάνει append το αρχείο και μην το φτιάχνει κάθε φορά από την αρχή. Έτσι, διατηρεί το κάθε ελάχιστο σφάλμα για κάθε τιμή του M.
- για τα αρχεία Mcentroids.txt θέλουμε να γράφει το αρχείο από την αρχή καθώς αποθηκεύουμε όλα τα κεντροειδή σε διαφορετικά αρχεία για την αναπαράσταση τους.

Προσθέσαμε και το Thread.sleep για να φαίνονται πιο καθαρά τα αποτελέσματα που τρέχουν στην μνήμη του προγράμματος.

```
    try (PrintWriter writer = new PrintWriter(new FileWriter( fileName: "errors.txt",  append: true))) {
        writer.println(currentM + " " + Collections.min(listOfErrors) );
    } catch (IOException e) { e.printStackTrace(System.err); }



    try (PrintWriter writer = new PrintWriter(new FileWriter( fileName: currentM + "centroids.txt"))) {
        for (Centroid centroid : centroids)
            writer.println(centroid.x + " " + centroid.y);
    } catch (IOException e) { e.printStackTrace(System.err); }

    try {
        Thread.sleep( millis: 1000); // delay
    } catch (InterruptedException e) { e.printStackTrace(System.err); }
}
```

Στο τέλος της μεθόδου runKmeans, πριν την εκτύπωση των αποτελεσμάτων στην μνήμη, επειδή θέλουμε να κρατήσουμε τα κεντροειδή με το μικρότερο σφάλμα, κάνουμε centroids.clear() και στην συνέχεια προσθέτουμε τα κατάλληλα κεντροειδή για την απεικόνηση τους με βάση την θέση του ελάχιστου σφάλματος, γιατί το mapCentroids είναι ορισμένο ως προς το run και τα M. Έτσι, έχοντας το run με το μικρότερο σφάλμα και τα M παίρνουμε τα σωστά κεντροειδή.

Η εφαμοργή του αλγόριθμου γίνεται στην μέθοδο runKmeans που δέχεται ως όρισμα την τιμή του M (για M = 3, 6, 9, 12). Επειδή όμως θέλουμε κάθε φορά που καλείται η μέθοδος να έχει διαφορετικό M, οι λίστες που περιέχουν το σφάλμα και τα κεντροειδή θα ξανά αρχικοποιούνται.

Για κάθε τρέξιμο επιλέγονται τυχαία κεντροειδή M και στην συνέχεια γίνεται ο υπολογισμός του αλγόριθμου. Αφού ολοκληρώσει το τρέξιμο θέλουμε να κρατήσουμε τα κεντροειδή που επιλέχθηκαν στην λίστα mapCentroids για κάθε τρέξιμο του M και το σφάλμα που πέτυχε στην λίστα listOfErrors.

Έτσι, έχοντας όλα αυτά για κάθε M, στο τέλος εκτυπώνονται στην μνήμη τα αποτελέσματα

Παράδειγμα εκτέλεσης του προγράμματος:

```
|M = 3|. K-means algorithm finished after 15 runs!  >best error: 372065.7959562082<

Best centroids by error:
(1)  0.3936571587642688 0.520904523764867
(2)  1.6642119659907335 0.5427881786903452
(3)  1.0209580631394892 1.5442661742518176
==========================================

|M = 6|. K-means algorithm finished after 15 runs!  >best error: 160357.09545523935<

Best centroids by error:
(1)  1.6930036400348085 1.608842600302464
(2)  1.7420140305693341 0.3781673944198817
(3)  1.0032141150339524 0.9485782385117386
(4)  0.35041233534358157 1.7207951214721497
(5)  0.2864405886362521 0.25846832688090987
(6)  0.22205826468992618 1.0239613953024784
==========================================
```

```
|M = 9|. K-means algorithm finished after 15 runs!  >best error: 136067.18264002012<

Best centroids by error:
(1)  1.0091225217001745 0.8896960440829725
(2)  0.2705906108416552 1.7346908539740655
(3)  1.7508498772169485 1.7400432762925682
(4)  1.0052535618533376 1.5307665720700299
(5)  0.5314523177837802 0.2951172497531649
(6)  1.725683250812754 0.24980896385330476
(7)  1.7985927678179943 1.0186150326282273
(8)  0.14975793905056525 0.2504533274516725
(9)  0.22205826468992618 1.0239613953024784
==========================================
```

```
|M = 12|. K-means algorithm finished after 15 runs!  >best error: 113339.79203663368<

Best centroids by error:
(1)  0.2630037357429691 1.6173289167646383
(2)  1.1124463387525512 1.0176883798333944
(3)  0.8897953658991923 1.0056085607980272
(4)  1.7980962904610152 1.0258656211091006
(5)  0.21304527247963964 1.0257016402301862
(6)  0.96834941934035 0.49004012515553275
(7)  1.6159166551968047 0.23648707616990552
(8)  0.25839032790680877 0.2583662739135892
(9)  1.874407959276795 0.2585400566265756
(10)  1.0059697067845585 1.5396268594809963
(11)  0.2783671578178083 1.8549868396137257
(12)  1.7505668952351021 1.7444622111633663
==========================================
```

Μπορούμε να παρατηρήσουμε ότι στα συγκεκριμένα τυχαία σημεία, δεν βρέθηκαν κεντροειδή που είναι κενά και αυτό γιατί καλύψαμε την περίπτωση του να βρει κενό σημείο ή την κανονική εφαρμογή του ή την περίπτωση που θα βρει κενό κεντροειδή. Σε κάθε περίπτωση γίνεται η καταλληλη επιλογή δεδομένων ώστε το πρόγραμμα να κάνει σωστούς υπολογισμούς.

```
private static void updateCentroids(int M) {
    for (int j = 0; j < M; j++) {
        double totalX = 0, totalY = 0;
        int count = 0;

        for (Point point : points) {
            if (point == null) continue;  // Skip null points

            if (point.clusterIndex == j) {
                totalX = totalX + point.x;
                totalY = totalY + point.y;
                count++;
            }
        }

        if (count > 0) {
            centroids.get(j).lastStateOfX = centroids.get(j).x;  // Update before changing x
            centroids.get(j).x = totalX / count;
            centroids.get(j).lastStateOfY = centroids.get(j).y;  // Update before changing y
            centroids.get(j).y = totalY / count;
        } else {
            centroids.get(j).lastStateOfX = centroids.get(j).x;  // Update before changing x
            centroids.get(j).lastStateOfY = centroids.get(j).y;  // Update before changing y
        }
    }
}
```

Η Ευκλείδια απόσταση υπολογίζεται στη μέθοδο euclideanDistance που δέχεται ως όρισμα με βάση τον τύπο το σημείο από το κέντρο του.

Το σφάλμα υπολογίζεται με την μέθοδο SSE (Sum Squared Error) όπου φτιάχνει ένα αντικείμενο τύπου ErrorCalculator και συνεχίζει με τους υπολογισμούς του σφάλματος.

```
private static double SSE() {
    double tempSSE = 0;
    ErrorCalculator calculator = new ErrorCalculator( initialError: 0);

    for (Point point : points) {
        if (point == null) continue;  // Skip null points

        double dx = point.x - centroids.get(point.clusterIndex).x;
        double dy = point.y - centroids.get(point.clusterIndex).y;

        // Square the differences before summing
        double maxDistance = Math.sqrt(dx * dx + dy * dy);

        calculator.addMaxDistance(maxDistance);
        tempSSE = tempSSE + calculator.getError();
    }
    return tempSSE;
}
```

Αρχεία

Τα αρχεία που προκύπτουν μετά την εφαρμογή του αλγόριθμου για 15 τρεξίματα για κάθε αριθμό ομάδων M = 3, 6, 9, 12 με βάση το σύνολο δεδομένων SDO (ΣΔΟ) είναι τα εξής:


3centroids.txt


6centroids.txt


9centroids.txt


12centroids.txt

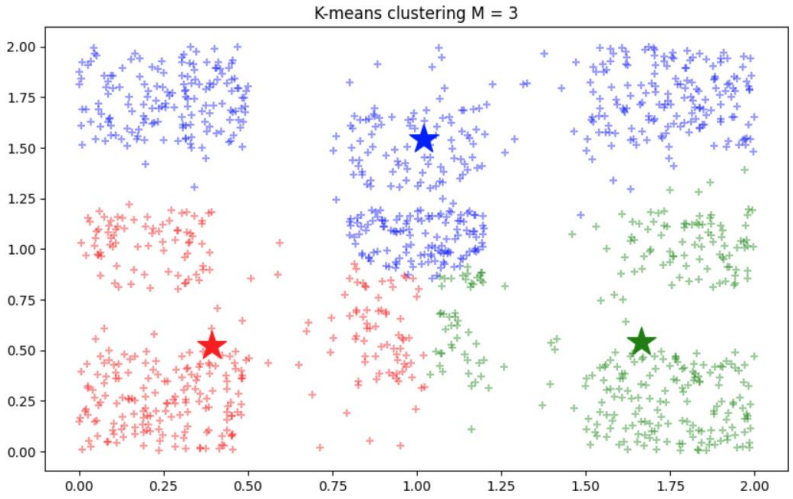

errors.txt

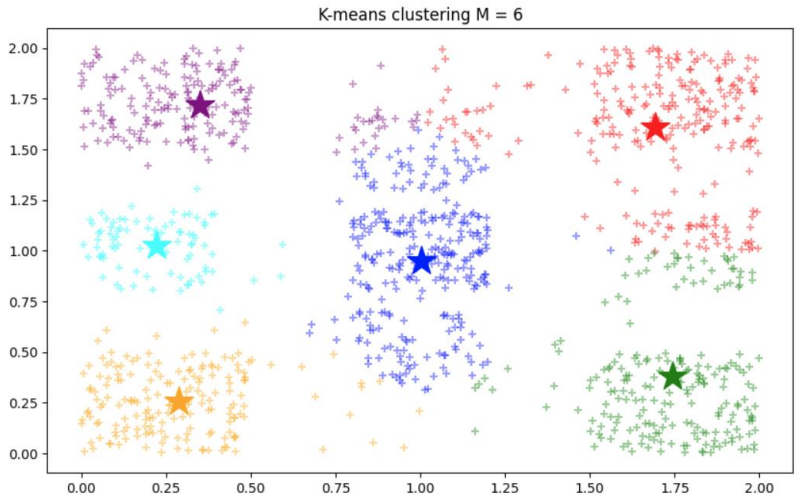

SDO.txt

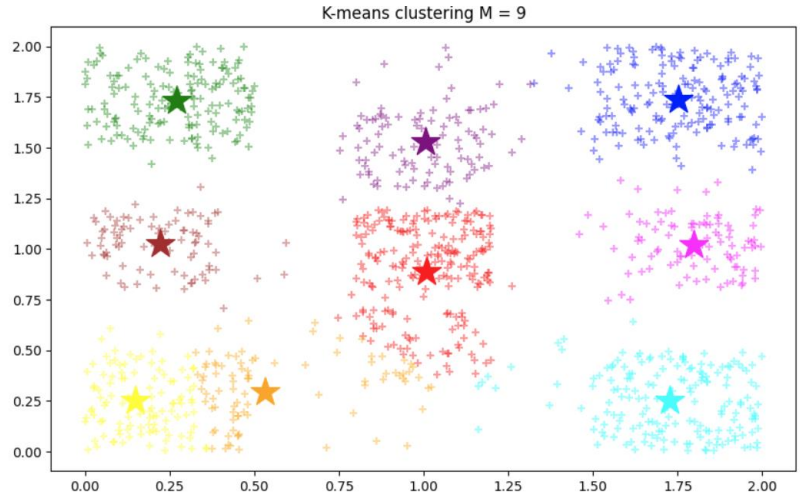
Αναπαράσταση δεδομένων

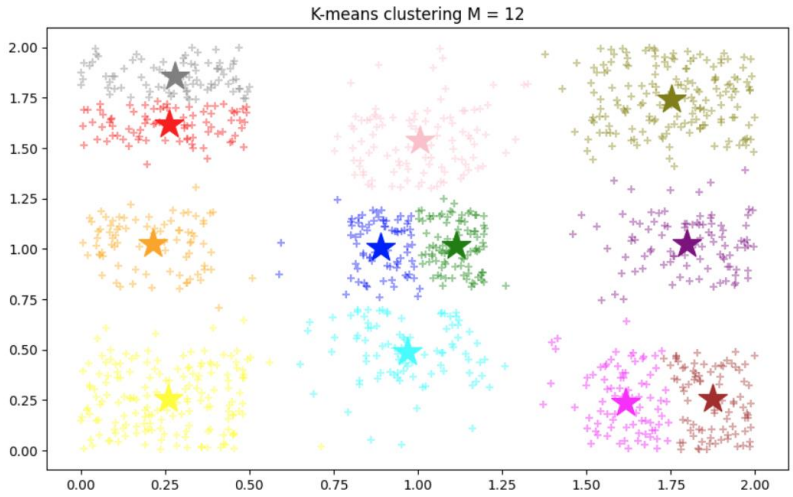
Με την βιβλιοθήκη matplotlib και την Python έγινε δυνατή η αναπαράσταση των δεδομένων για την καλύτερη κατανόηση των αποτελεσμάτων του αλγόριθμου K-means.

Με βάση τα αρχεία που πήραμε από την εφαρμογή του προγράμματος στη Java, χρησιμοποιήσαμε το Python script στον ιστότοπο Kaggle.
Για να είναι λειτουργικός ο κώδικας, θα πρέπει να προστεθούν τα δεδομένα (txt αρχεία) στον ιστότοπο σε ένα σύνολο δεδομένων με ονομασία anlog2kmeans.
Οπότε, με το path των αρχείων που δίνεται από το Kaggle, μπορούν να φορτωθούν τα δεδομένα σε λίστες.

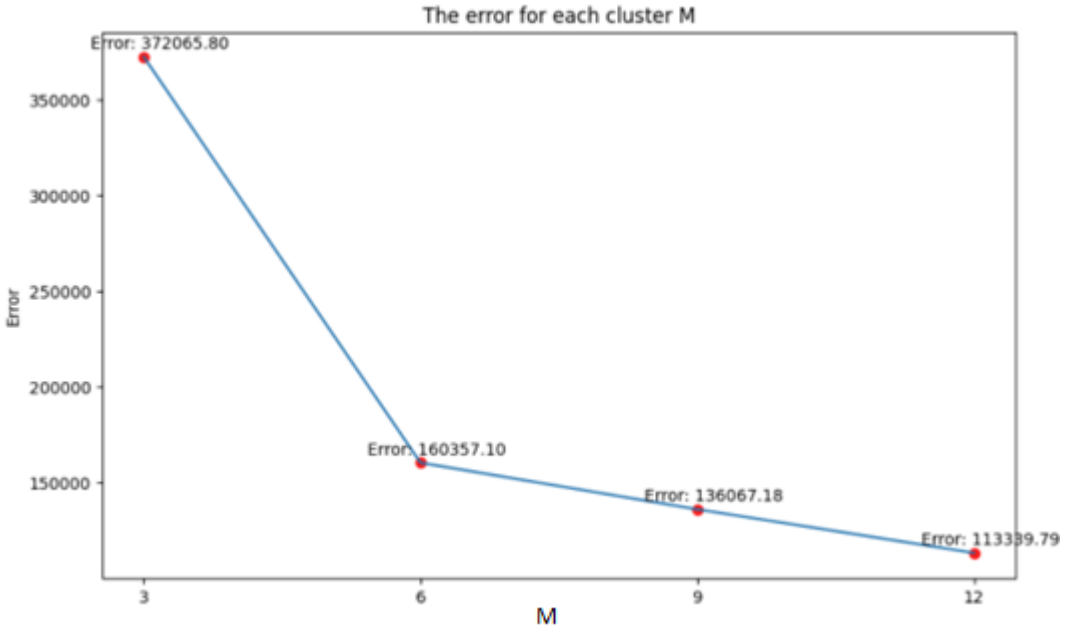








Για το error, παρατηρούμε στο γράφημα ότι και σύμφωνα με την εκφώνηση το σφάλμα αρχίζει και γίνεται πιο σταθερό από τις 9 ομάδες και μετά, διότι πλέον δεν μειώνεται απότομα όπως σε προηγούμενες τιμές των clusters. Αυτό μας οδηγεί στο συμπέρασμα να συμφωνήσουμε με την εκφώνηση ότι ο πραγματικός αριθμός των ομάδων είναι 9.



Python script

```
import matplotlib.pyplot as plt
import numpy as np


# load data - centroids - errors
dataset = np.loadtxt("/kaggle/input/anlog2kmeans/SDO.txt")
centroids3 = np.loadtxt("/kaggle/input/anlog2kmeans/3centroids.txt")
centroids6 = np.loadtxt("/kaggle/input/anlog2kmeans/6centroids.txt")
centroids9 = np.loadtxt("/kaggle/input/anlog2kmeans/9centroids.txt")
centroids12 = np.loadtxt("/kaggle/input/anlog2kmeans/12centroids.txt")
errors_data = np.loadtxt("/kaggle/input/anlog2kmeans/errors.txt")
```

```
colors = ["red", "green", "blue", "purple", "orange", "cyan", "magenta", "yellow", "brown",
"pink", "gray", "olive"]



# clustering M = 3

cluster_assign = np.argmin(np.linalg.norm(dataset[:, None] - centroids3, axis = 2), axis = 1)


plt.figure(figsize = (10, 6))

plt.title("K-means clustering M = 3")


for i in range(len(centroids3)):

   cluster_points = dataset[cluster_assign == i]


   plt.scatter(cluster_points[:, 0], cluster_points[:, 1], marker = "+", color = colors[i], alpha =
0.4)

   plt.scatter(centroids3[i, 0], centroids3[i, 1], color = colors[i], marker = "*", s = 500)

plt.show()


##########################################################

# clustering M = 6

cluster_assign = np.argmin(np.linalg.norm(dataset[:, None] - centroids6, axis = 2), axis = 1)


plt.figure(figsize = (10, 6))

plt.title("K-means clustering M = 6")


for i in range(len(centroids6)):

   cluster_points = dataset[cluster_assign == i]


   plt.scatter(cluster_points[:, 0], cluster_points[:, 1], marker = "+", color = colors[i], alpha =
0.4)

   plt.scatter(centroids6[i, 0], centroids6[i, 1], color = colors[i], marker = "*", s = 500)

plt.show()
```

```
#########################################################
# clustering M = 9
cluster_assign = np.argmin(np.linalg.norm(dataset[:, None] - centroids9, axis = 2), axis = 1)

plt.figure(figsize = (10, 6))
plt.title("K-means clustering M = 9")

for i in range(len(centroids9)):
   cluster_points = dataset[cluster_assign == i]

   plt.scatter(cluster_points[:, 0], cluster_points[:, 1], marker = "+", color = colors[i], alpha = 0.4)
   plt.scatter(centroids9[i, 0], centroids9[i, 1], color = colors[i], marker = "*", s = 500)
plt.show()

#########################################################
# clustering M = 12
cluster_assign = np.argmin(np.linalg.norm(dataset[:, None] - centroids12, axis = 2), axis = 1)

plt.figure(figsize = (10, 6))
plt.title("K-means clustering M = 12")

for i in range(len(centroids12)):
   cluster_points = dataset[cluster_assign == i]

   plt.scatter(cluster_points[:, 0], cluster_points[:, 1], marker = "+", color = colors[i], alpha = 0.4)
   plt.scatter(centroids12[i, 0], centroids12[i, 1], color = colors[i], marker = "*", s = 500)
plt.show()

#########################################################
```

```
# plot errors

m_values = errors_data[:, 0]

errors = errors_data[:, 1]


plt.figure(figsize = (10, 6))

plt.title("The error for each cluster M")


plt.plot(m_values, errors)

plt.scatter(m_values, errors, marker="o", color="red")


for i, txt in enumerate(errors):

    plt.annotate(f"Error: {errors[i]:.2f}", (m_values[i], errors[i]), textcoords="offset points", xytext=(10, 5), ha="center")


plt.xlabel("M")

plt.xticks(m_values)

plt.ylabel("Error")

plt.show()
```