ΑΝΑΓΝΩΡΙΣΗ ΠΡΟΤΥΠΩΝ
2η Σειρά Ασκήσεων
Ομαδοποίηση Δεδομένων

Τσάλεσης Ευάγγελος, ΑΜ: 1779

1. ΕΙΣΑΓΩΓΗ

Στόχος της άσκησης ήταν η υλοποίηση 2 μεθόδων ομαδοποίησης δεδομένων. Οι μέθοδοι είναι:

- ***k-Means με ευκλείδεια απόσταση***. Δοκιμάστηκαν οι τιμές του k = [2,3,4].
- ***Συνθετική Ιεραρχική Ομαδοποίηση (Agglomerative Hierarchical Clustering)***. Ως μέτρο για την συγχώνευση ομάδων επιλέχτηκε η ελάχιστη απόσταση των μέσων 2 ομάδων. Η μέθοδος εξετάστηκε, επίσης, όπως η προηγούμενη μέθοδος, για πλήθος ομάδων 2,3 και 4.

Η υλοποίηση έγινε σε γλώσσα προγραμματισμού java (ο πλήρης κώδικας παρατίθεται στο παράρτημα Α).

Μετά την υλοποίηση οι μέθοδοι δοκιμάστηκαν σε δύο πειραματικά σύνολα δεδομένων 2 διαστάσεων και 2 κατηγοριών (*cross*: 500 σημείων, *moonandsun*: 500 σημείων).

Η αξιολόγηση των μεθόδων, για κάθε πλήθος ομάδων και κάθε σύνολο δεδομένων, έγινε υπολογίζοντας τις τιμές των παρακάτω μέτρων:

- ***Purity***: Το μέσο των σωστά ταξινομημένων σημείων. Ως σωστά ταξινομημένο σημείο θεωρήθηκε αυτό του οποίου η πραγματική κατηγορία συμπίπτει με την πλειοψηφούσα πραγματική κατηγορία των σημείων της ομάδας που τοποθετήθηκε από την μέθοδο.
- ***Total F-measure:*** Το άθροισμα των f-measure τιμών κάθε ομάδας (για α=1)

2. ΥΛΟΠΟΙΗΣΗ

2.1 K-means με ευκλείδεια απόσταση

Αρχικά επιλέγονται K τυχαία σημεία από το σύνολο των δεδομένων ως μέσα των K ομάδων.

Στην συνέχεια κάθε σημείο τοποθετείται στην ομάδα με τον κοντινότερο από αυτό μέσο και υπολογίζεται μια τιμή ***E***, η οποία ορίζεται ως το άθροισμα των αποστάσεων κάθε σημείου από τον τρέχοντα μέσο της τρέχουσας ομάδας του.

Υπολογίζονται τα νέα μέσα των ομάδων και η παραπάνω διαδικασία επαναλαμβάνεται, μέχρι η διαφορά μεταξύ 2 διαδοχικών τιμών Ε να γίνει μικρότερη από μια τιμή ακρίβειας ***ε*** ή όταν τα μέσα των ομάδων δεν μεταβάλλονται σημαντικά σε 2 διαδοχικά βήματα.

Η μέθοδος επαναλαμβάνεται 10 φορές, με κάθε φορά διαφορετικά αρχικά μέσα και τελικά επιλέγεται η επανάληψη που οδήγησε σε ελάχιστη τιμή του Ε.

2.2 Συνθετική Ιεραρχική Ομαδοποίηση

Η μέθοδος ξεκινά θεωρώντας ότι κάθε σημείο αποτελεί και διαφορετική ομάδα.

Βρίσκει το ζεύγος των σημείων με την μικρότερη απόσταση και τα συγχωνεύει σε μια ομάδα, με μέσο το μέσο των σημείων. Έτσι το πλήθος των συνολικών ομάδων μειώνεται κατά ένα.

Ο αλγόριθμος επαναλαμβάνεται συγχωνεύοντας σε κάθε βήμα τις ομάδες με την μικρότερη απόσταση μέσων και υπολογίζει τα νέα μέσα.

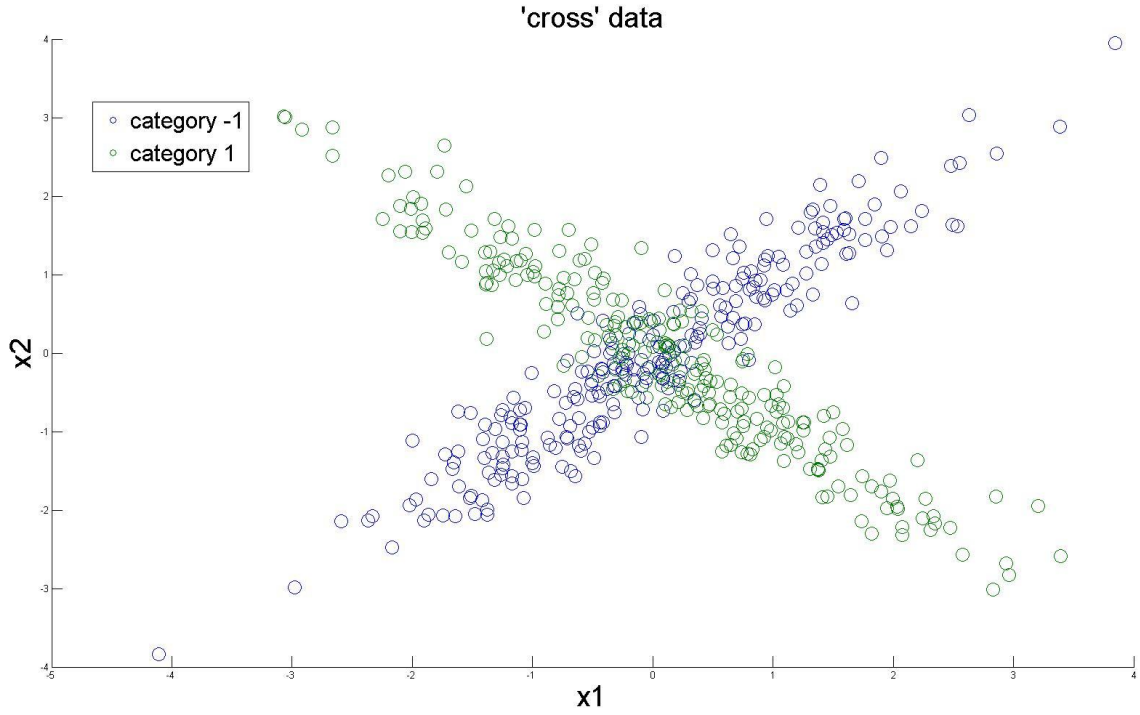
Ολοκληρώνεται όταν επιτευχθεί το πλήθος των ομάδων που επιθυμούμε. Στην παρούσα άσκηση το τελικό πλήθος είναι 2.

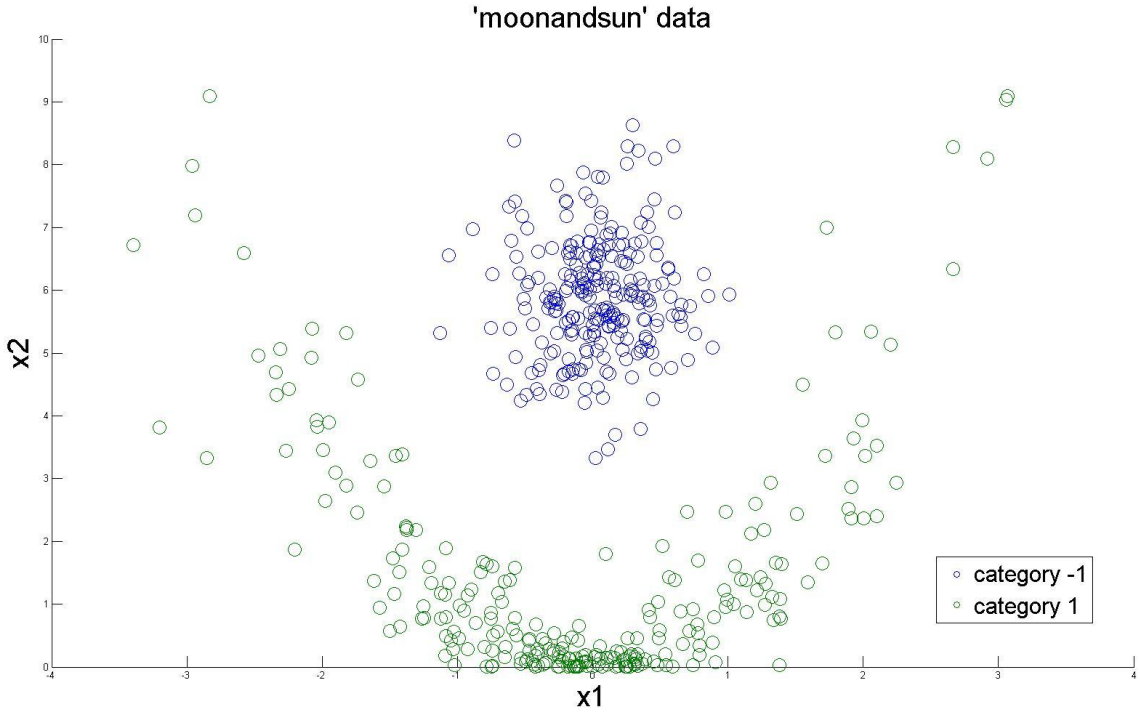
Κατά το τελευταίο βήμα της μεθόδου (για πλήθος ομάδων 2), καθώς και κατά τα προηγούμενα δύο (για πλήθος ομάδων 3 και 4) γίνεται αξιολόγηση της μεθόδου και αποθηκεύονται τα αποτελέσματα.

3. ΑΠΟΤΕΛΕΣΜΑΤΑ

3.1 Εισαγωγή

Παρακάτω φαίνονται τα σημεία των 2 πειραματικών συνόλων ***"cross"*** και ***"moonandsun"*** και οι κατηγορίες στις οποίες ανήκουν.





3.2 K-means με ευκλείδεια απόσταση

Οι τιμές των ***purity***, ***F-measure*** (για κάθε ομάδα) και ***total F-measure*** (άθροισμα) της μεθόδου για διάφορες τιμές του πλήθους των ομάδων ***K*** για τα δύο σύνολα δεδομένων φαίνονται στους παρακάτω πίνακες.

cross

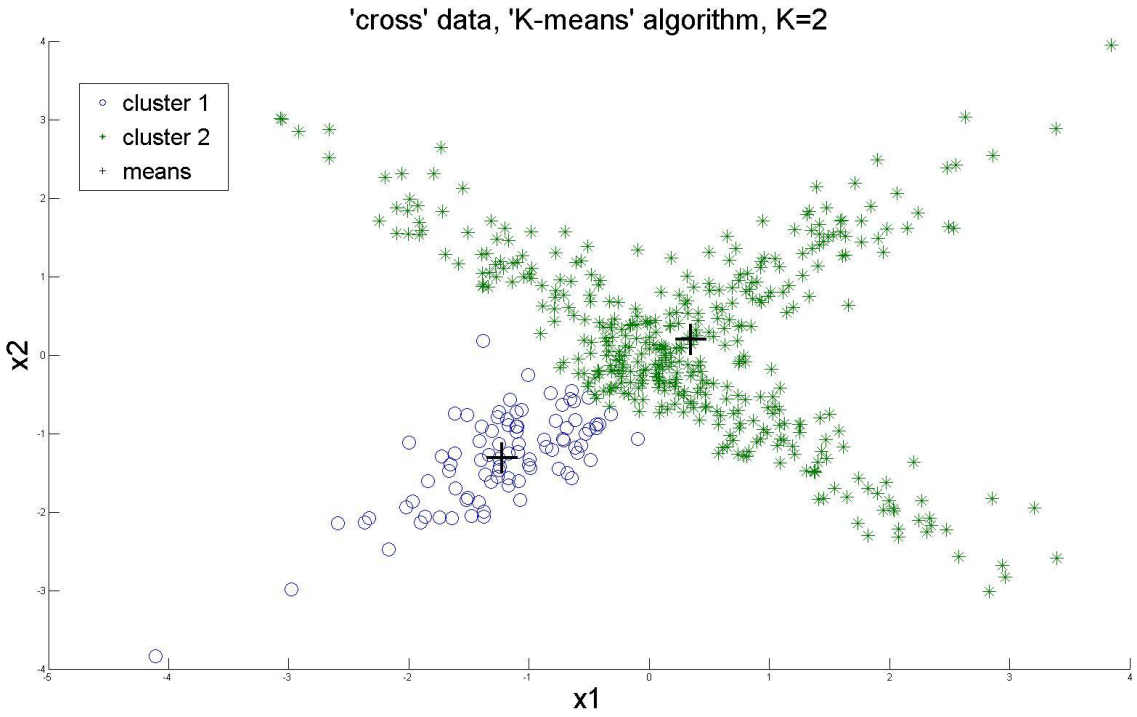
K	purity	F- measure	
2	0.674	k = 1	0.519
		k = 2	0.753
		total	**1.272**
3	0.694	k = 1	0.554
		k = 2	0.548
		k = 3	0.564
		total	**1.666**
4	0.808	k = 1	0.438
		k = 2	0.521
		k = 3	0.503
		k = 4	0.638
		total	**2.100**

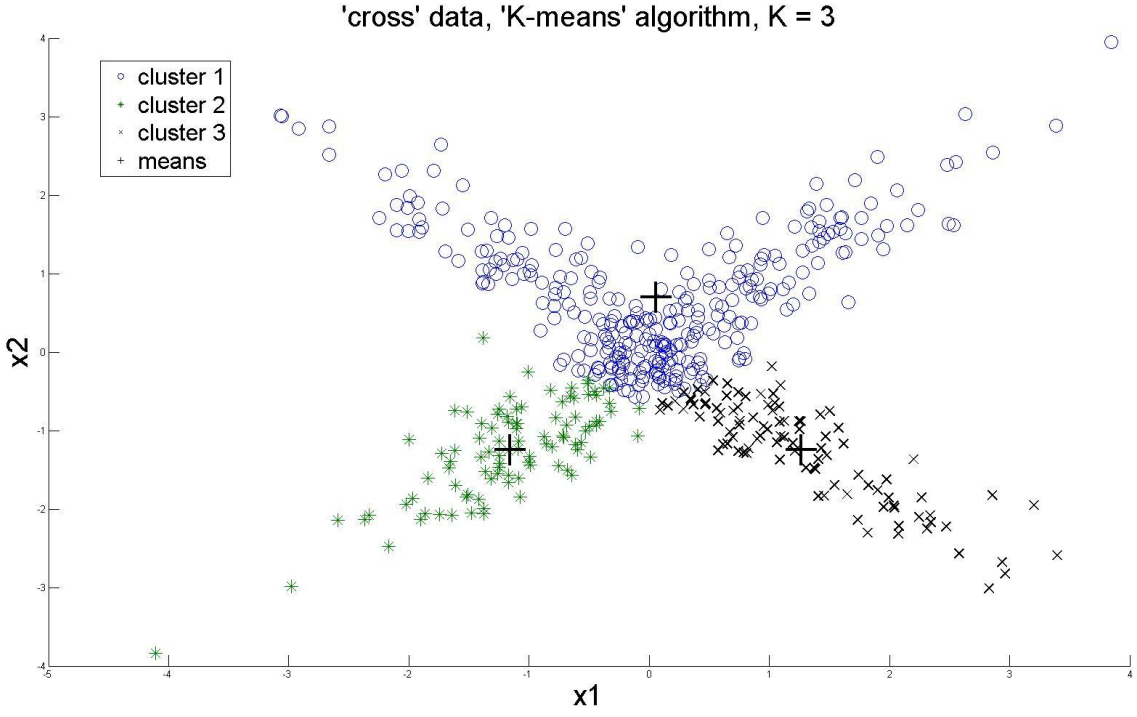
moonandsun

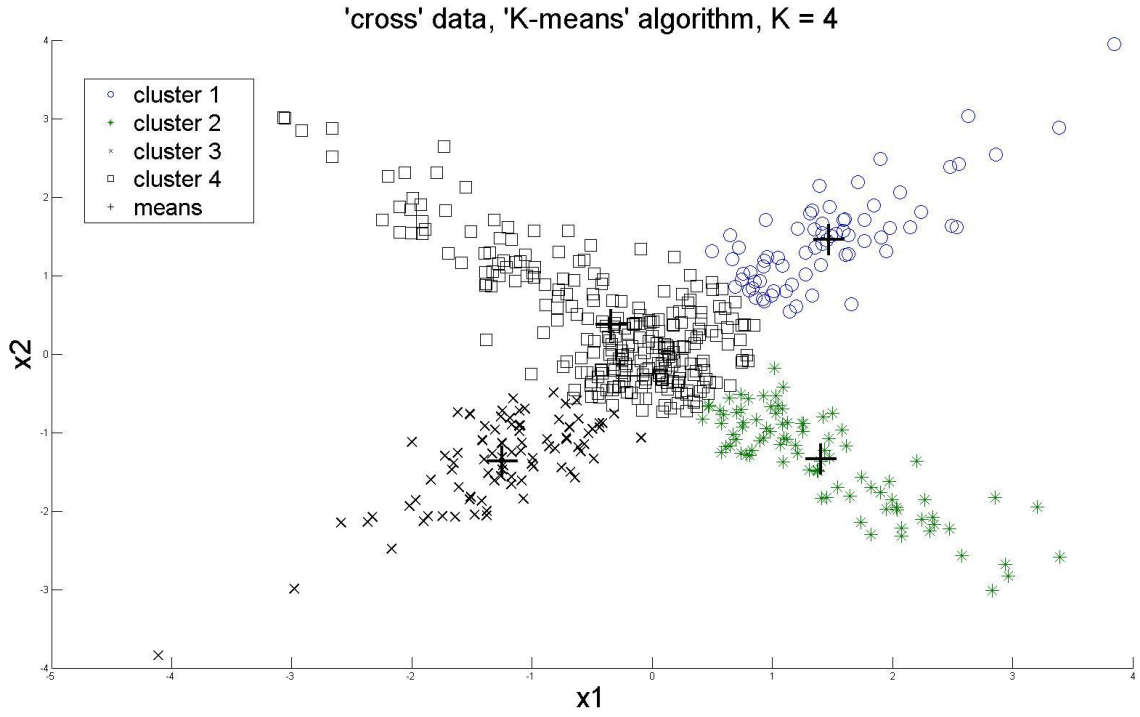
K	purity	F- measure	
2	0.924	k = 1	0.918
		k = 2	0.929
		total	**1.847**
3	0.910	k = 1	0.555
		k = 2	0.901
		k = 3	0.701
		total	**2.157**
4	0.924	k = 1	0.701
		k = 2	0.565
		k = 3	0.355
		k = 4	0.775
		total	**2.396**

Στα παρακάτω διαγράμματα φαίνονται εποπτικά για κάθε ***K*** και κάθε σύνολο δεδομένων οι ομάδες που προέκυψαν από την μέθοδο και τα αντίστοιχα μέσα τους.

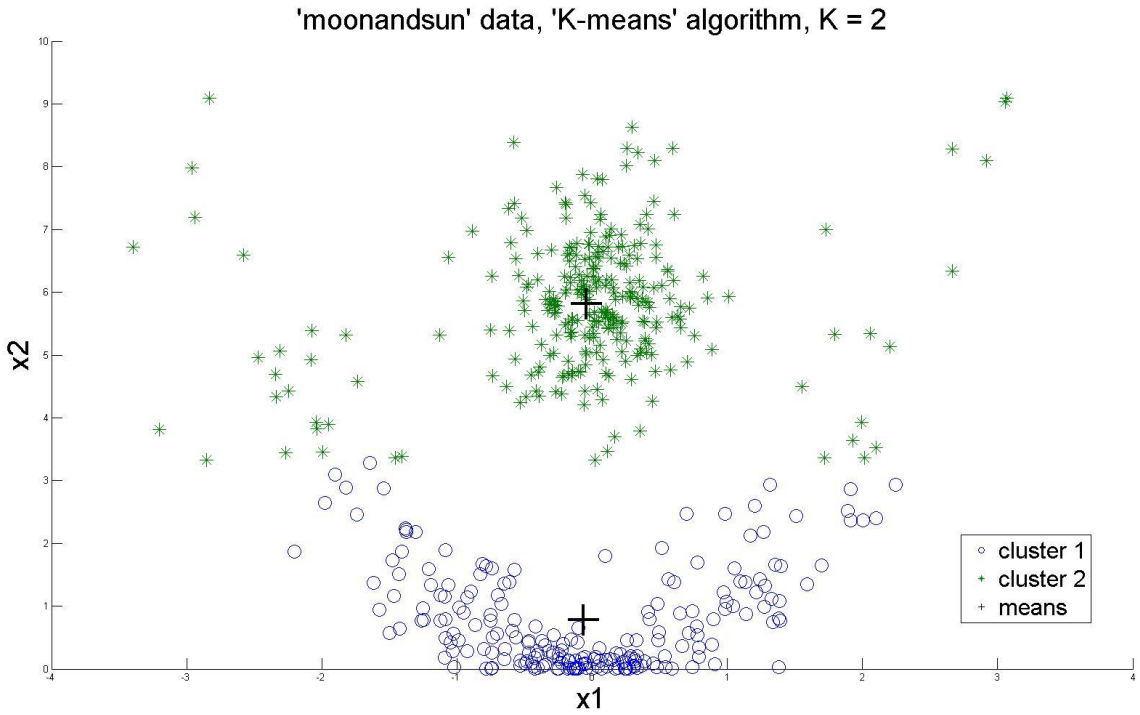
cross

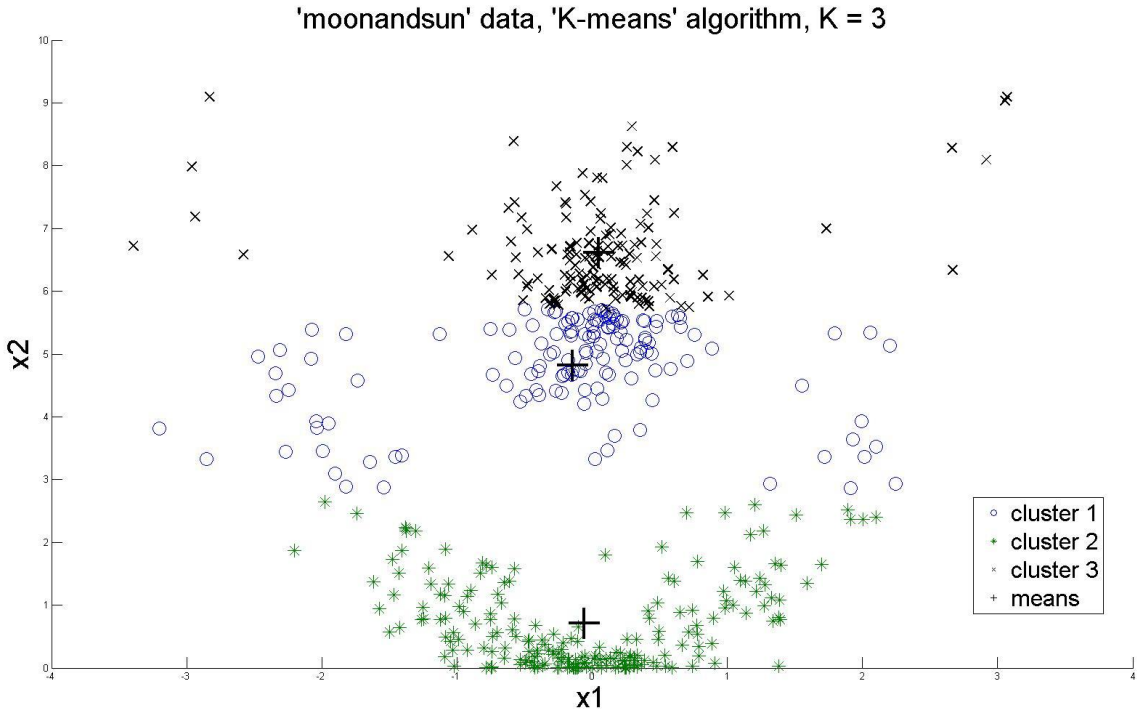

'cross' data, 'K-means' algorithm, K=2
cluster 1
cluster 2
means
x2
x1

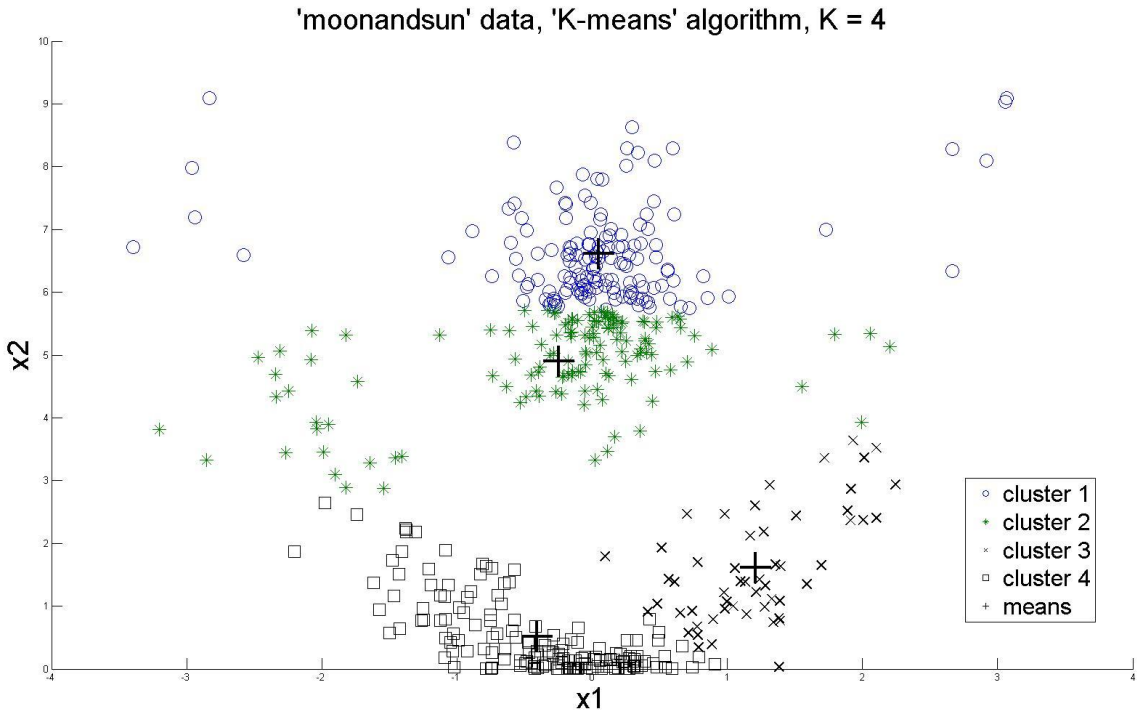

'cross' data, 'K-means' algorithm, K = 3
cluster 1
cluster 2
cluster 3
means
x2
x1


'cross' data, 'K-means' algorithm, K = 4
cluster 1
cluster 2
cluster 3
cluster 4
means
x1
x2

moonandsun


'moonandsun' data, 'K-means' algorithm, K = 2
cluster 1
cluster 2
means
x1
x2


'moonandsun' data, 'K-means' algorithm, K = 3
x2
x1
cluster 1
cluster 2
cluster 3
means


'moonandsun' data, 'K-means' algorithm, K = 4
x2
x1
cluster 1
cluster 2
cluster 3
cluster 4
means

3.3 Συνθετική Ιεραρχική Ομαδοποίηση

Οι τιμές των ***purity***, ***F-measure*** (για κάθε ομάδα) και ***total F-measure*** (άθροισμα) της μεθόδου για διάφορες τιμές του πλήθους των ομάδων ***K*** για τα δύο σύνολα δεδομένων φαίνονται στους παρακάτω πίνακες.

cross

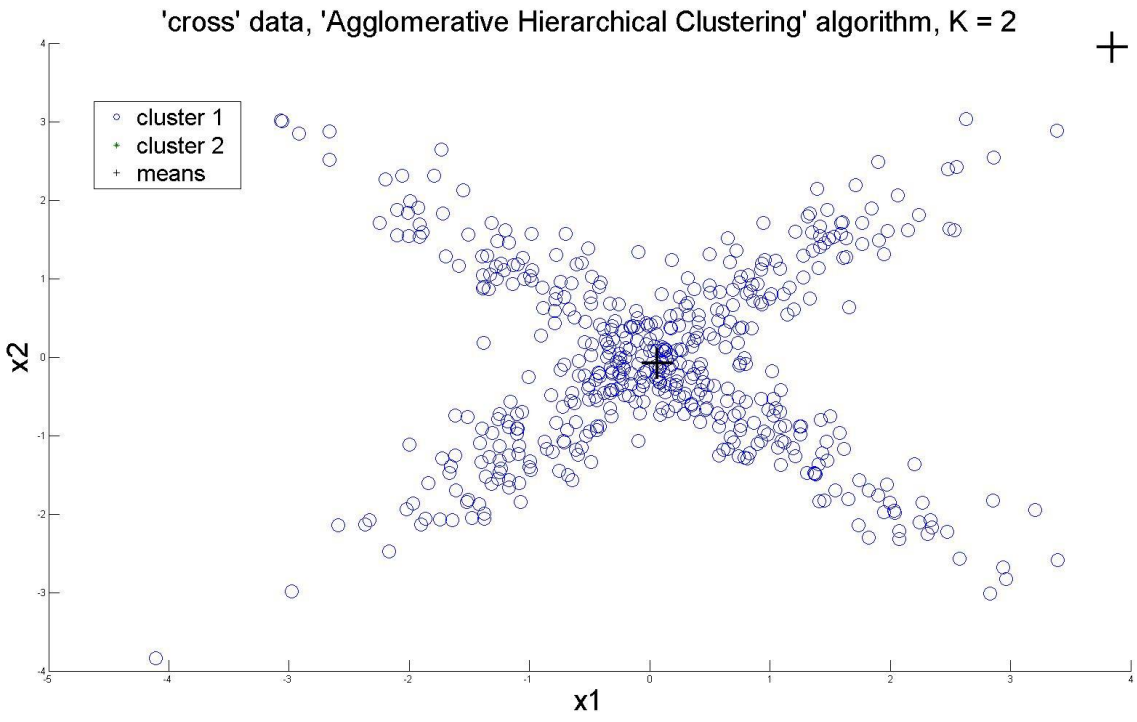
K	purity	F- measure	
2	0.502	k = 1	0.668
		k = 2	0.008
		total	**0.676**
3	0.506	k = 1	0.669
		k = 2	0.016
		k = 3	0.008
		total	**0.693**
4	0.560	k = 1	0.689
		k = 2	0.214
		k = 3	0.016
		k = 4	0.008
		total	**0.927**

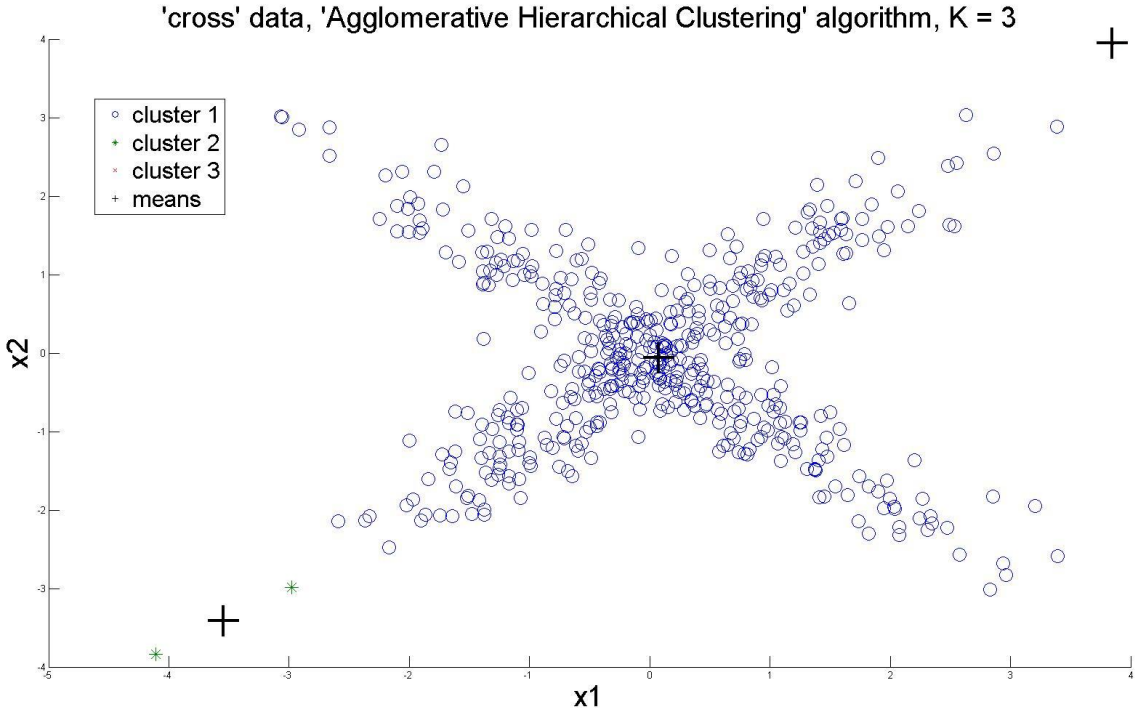
moonandsun

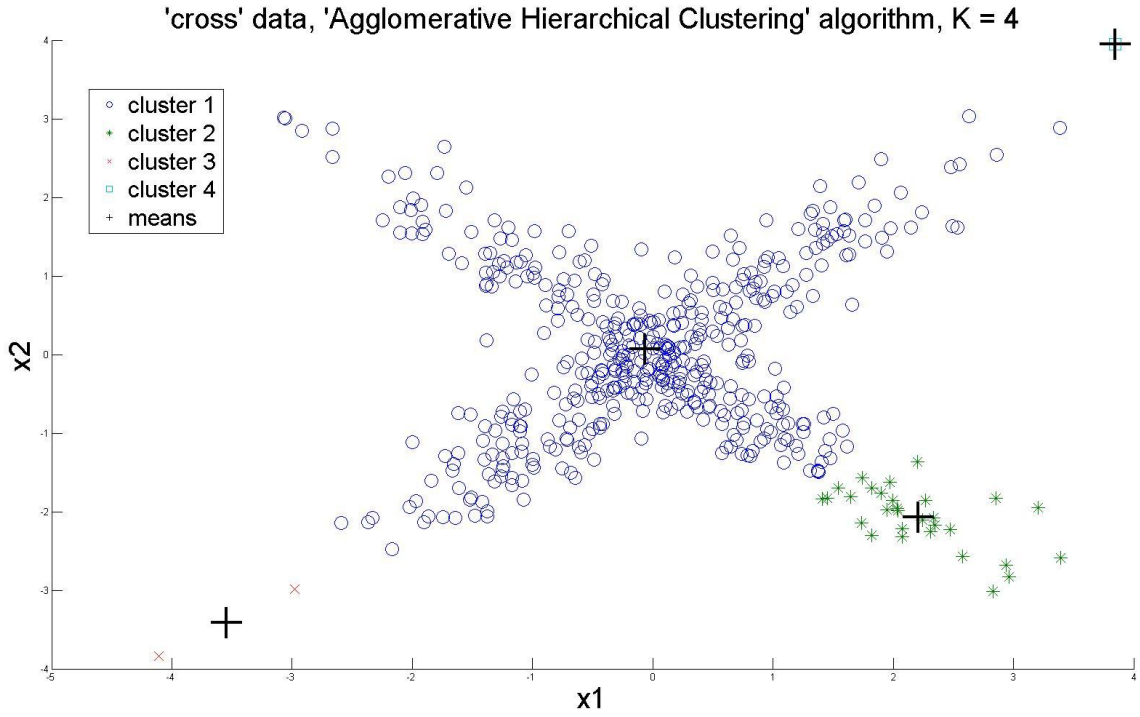
K	purity	F- measure	
2	0.904	k = 1	0.894
		k = 2	0.912
		total	**1.806**
3	0.912	k = 1	0.894
		k = 2	0.919
		k = 3	0.031
		total	**1.844**
4	0.922	k = 1	0.894
		k = 2	0.928
		k = 3	0.039
		k = 4	0.031
		total	**1.892**

Στα παρακάτω διαγράμματα φαίνονται εποπτικά για κάθε ***K*** και κάθε σύνολο δεδομένων οι ομάδες που προέκυψαν από την μέθοδο και τα αντίστοιχα μέσα τους.

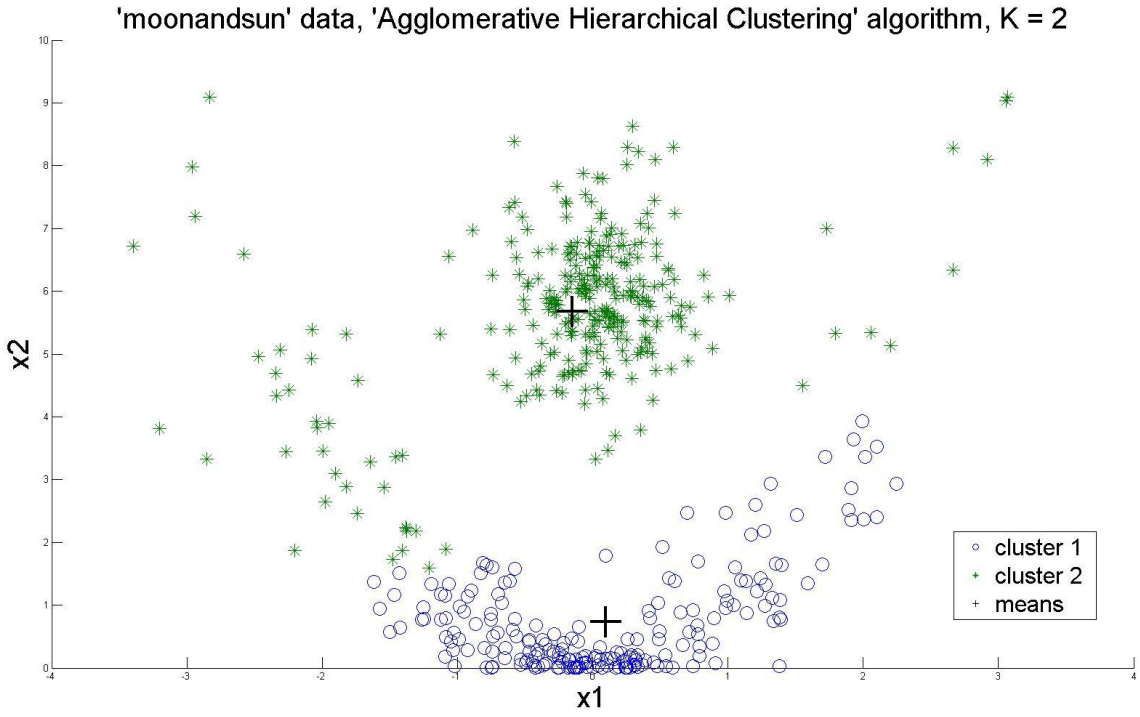
cross

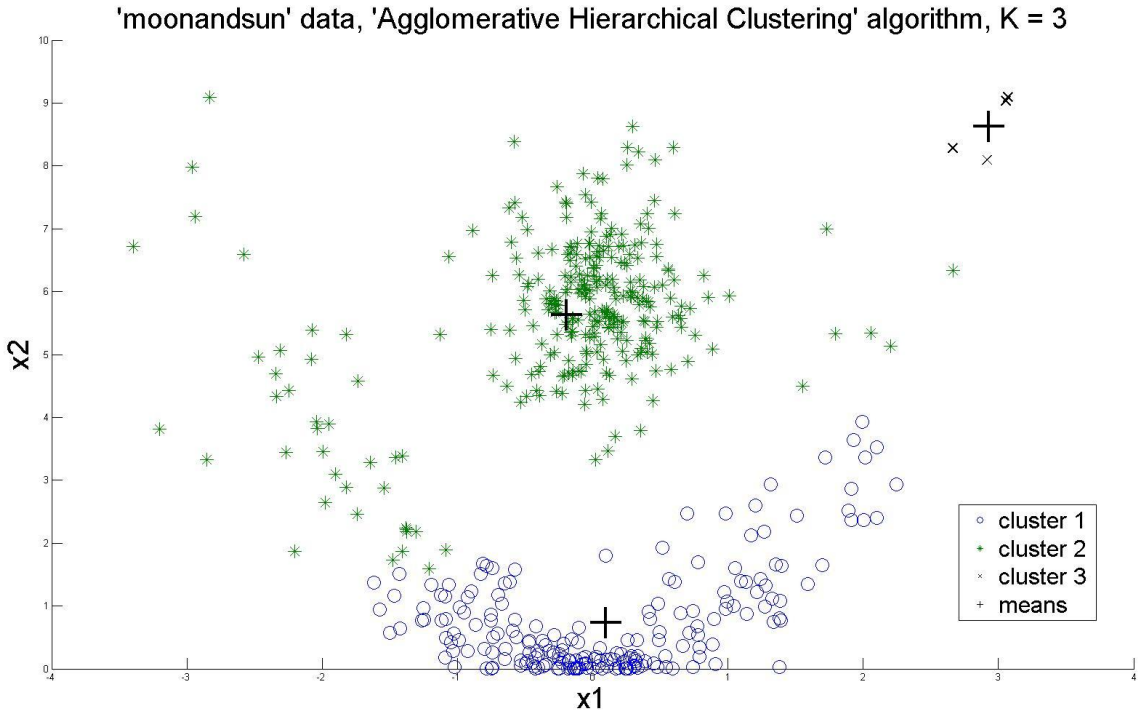

'cross' data, 'Agglomerative Hierarchical Clustering' algorithm, K = 2
cluster 1
cluster 2
means
x1
x2

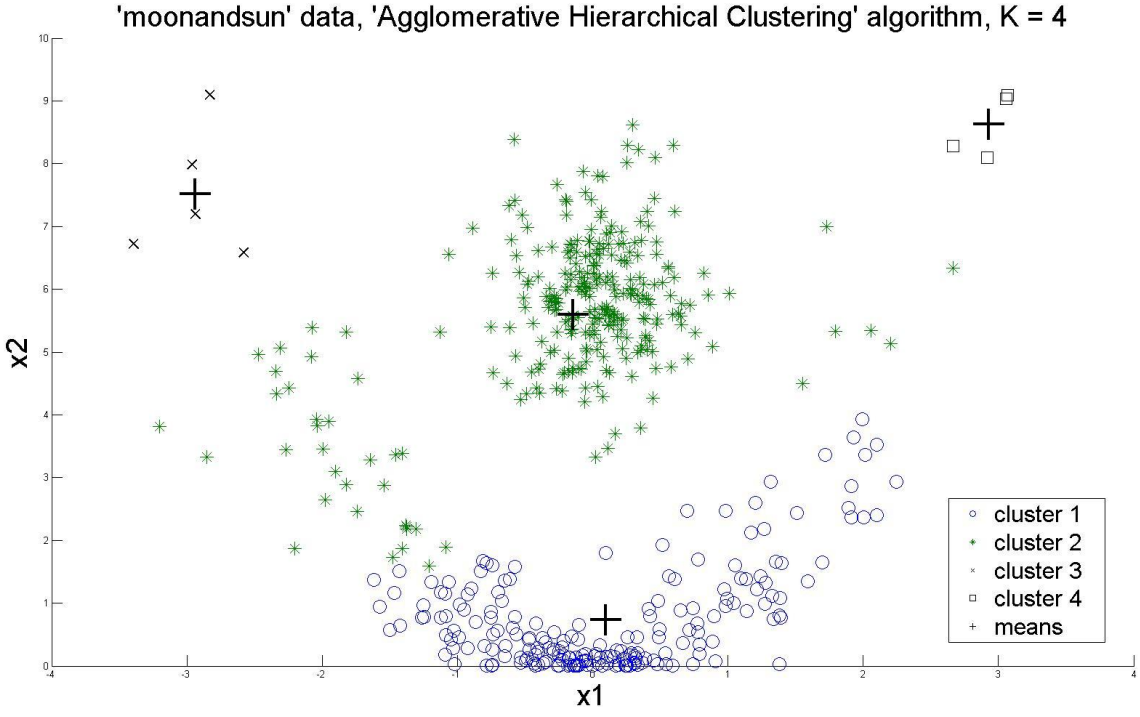

'cross' data, 'Agglomerative Hierarchical Clustering' algorithm, K = 3
cluster 1
cluster 2
cluster 3
means
x1
x2


'cross' data, 'Agglomerative Hierarchical Clustering' algorithm, K = 4
cluster 1
cluster 2
cluster 3
cluster 4
means
x1
x2

moonandsun


'moonandsun' data, 'Agglomerative Hierarchical Clustering' algorithm, K = 2
cluster 1
cluster 2
means
x1
x2


'moonandsun' data, 'Agglomerative Hierarchical Clustering' algorithm, K = 3
x2
x1
cluster 1
cluster 2
cluster 3
means


'moonandsun' data, 'Agglomerative Hierarchical Clustering' algorithm, K = 4
x2
x1
cluster 1
cluster 2
cluster 3
cluster 4
means

ΠΑΡΑΡΤΗΜΑ Α (κώδικας)

```
// TSALESIS EVANGELOS
// AM: 1779

// CLUSTERING DATA (2 METHODS)

import java.util.Scanner;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Random;


public class PatternRecognition2 {

    // MAIN
    public static void main(String[] args) {
       // imports data into a string array (every element a line of data file)
        Scanner inputReader = null;
        ArrayList<String> lines = new ArrayList<>();
        try {
            inputReader = new Scanner(new
FileInputStream("C:/users/vagtsal/desktop/assignment2/data/cross.dat"));
        }
        catch (FileNotFoundException e){
            System.out.println("File not found");
            System.exit(0);
        }
        while (inputReader.hasNextLine()){
            lines.add(inputReader.nextLine());
        }

        int dataSize = lines.size();
        int[][] category = new int[dataSize][2];      // category[0] = true, category[1] =
predicted
        double[][] x = new double[dataSize][2];

        // putting txt data in arrays
        for (int i=0; i<dataSize;i++){
            String samplePoint[] = lines.get(i).trim().split("\\s+");
            category[i][0] = Integer.parseInt(samplePoint[0]);
            category[i][1] = 0;
            x[i][0] = Double.parseDouble(samplePoint[1]);
            x[i][1] = Double.parseDouble(samplePoint[2]);
        }

        System.out.println("K-means Clustering");
        Kmeans(category, x, dataSize);
        System.out.println("-------------------------------------------------");
        System.out.println("Agglomerative Hierarchical Clustering");
        AHC(category, x, dataSize);
    }


    // "K-means" METHOD
    public static void Kmeans(int[][] category, double[][] x, int dataSize){
        double epsilon = 0.00001; // accuracy
        int[] categoryPr = new int[dataSize];     // predicted category

        Random random = new Random();
        // For K = 2,3,4
        for (int k=2; k<=4; k++){
            //10 tries for every number of K
            double minE = 0;                          // minimum E of 10 tries...
            double[][] minMeans = new double[k][2]; // ...and corresponding means
            for (int r=0; r<10; r++){
                double E;
                // initial means (random sample points)
```

```
                double[][] means = new double[k][2];
                for (int j=0; j<k; j++){
                    int rn = random.nextInt(dataSize);
                    means[j][0] = x[rn][0];
                    means[j][1] = x[rn][1];
                }

                // calculate initial E, predict initial cluster of every sample point
                E = 0;
                double previousE;
                for (int i=0; i<dataSize; i++){
                    double minDistance = Math.sqrt((x[i][0]-means[0][0])*(x[i][0]-
means[0][0]) + (x[i][1]-means[0][1])*(x[i][1]-means[0][1]));
                    categoryPr[i] = 1;
                    for (int j=1; j<k; j++){
                        double distance = Math.sqrt((x[i][0]-means[j][0])*(x[i][0]-
means[j][0]) + (x[i][1]-means[j][1])*(x[i][1]-means[j][1]));
                        if (minDistance > distance) {
                            minDistance = distance;
                            categoryPr[i] = j+1;
                        }
                    }
                    E = E + minDistance;
                }

                // predict cluster of every sample point
                do {
                    // calculate new means
                    double[][] previousMeans = new double[k][2];
                    for (int j=0; j<k; j++){
                        previousMeans[j][0] = means[j][0];
                        previousMeans[j][1] = means[j][1];
                        means[j][0] = 0;
                        means[j][1] = 0;
                        int counter = 0;
                        for (int i=0; i<dataSize; i++){
                            if (categoryPr[i] == j+1){
                                means[j][0] = means[j][0] + x[i][0];
                                means[j][1] = means[j][1] + x[i][1];
                                counter++;
                            }
                        }
                        means[j][0] = means[j][0] / counter;
                        means[j][1] = means[j][1] / counter;
                    }


                    // check if new means are equal to old means
                    boolean flagSameMeans = false;
                    for (int j=0; j<k; j++){
                        if (Math.abs(previousMeans[j][0] - means[j][0]) > epsilon ||
Math.abs(previousMeans[j][1] - means[j][1]) > epsilon){
                            continue;
                        }
                        flagSameMeans = true;
                    }
                    if (flagSameMeans == true){
                        break;
                    }

                    // calculate new E, predict new cluster of every sample point
                    previousE = E;
                    E = 0;
                    for (int i=0; i<dataSize; i++){
                        double minDistance = Math.sqrt((x[i][0]-means[0][0])*(x[i][0]-
means[0][0]) + (x[i][1]-means[0][1])*(x[i][1]-means[0][1]));
                        categoryPr[i] = 1;
                        for (byte j=1; j<k; j++){
```

```
                            double distance = Math.sqrt((x[i][0]-means[j][0])*(x[i][0]-
means[j][0]) + (x[i][1]-means[j][1])*(x[i][1]-means[j][1]));
                            if (minDistance > distance) {
                                minDistance = distance;
                                categoryPr[i] = j+1;
                            }
                        }
                        E = E + minDistance;
                    }
                } while (Math.abs(E - previousE) > epsilon );

                // find the minimum E of the 10 tries (and keep the corresponding means
and sample points' categories)
                if (minE > E || minE == 0){
                    minE = E;
                    for (int j=0; j<k; j++){
                        minMeans[j][0] = means[j][0];
                        minMeans[j][1] = means[j][1];
                        for (int i=0; i<dataSize; i++){
                            category[i][1] = categoryPr[i];
                        }
                    }
                }
            }

            // evaluate method (purity, f-measure)
            evaluate(category, dataSize, k);

            // print means and clusters (debugging)
            /*for (int j=0; j<k; j++){
                System.out.println(minMeans[j][0] + "," + minMeans[j][1]);
            }
            for(int i=0; i<dataSize;i++){
                System.out.println(x[i][0] + " " + x[i][1] + " " + category[i][0] + " " +
category[i][1]);
            }*/
        }
    }


    // "Agglomerative Hierarchical Clustering" METHOD
    public static void AHC(int[][] category, double[][] x, int dataSize){
        ArrayList<double[]> means = new ArrayList<>();                  // stores means of
every cluster
        ArrayList<ArrayList<Integer>> clusters = new ArrayList<>();     // stores a list
of sample points' indices of every cluster

        // initiate clusters (every sample point is a cluster)
        for (int i=0; i<dataSize; i++){
            // means
            means.add(new double[2]);
            means.get(means.size()-1)[0] = x[i][0];
            means.get(means.size()-1)[1] = x[i][1];

            // elements
            clusters.add(new ArrayList<>());
            clusters.get(clusters.size()-1).add(i);
        }

        for (int k=dataSize-1; k>=2; k--){
            // find the pair of clusters with the minimum distance
            double minDistance = 0.0;
            int minI = -1;
            int minJ = -1;
            for (int i=0; i<clusters.size(); i++){
                for (int j=0; j<clusters.size(); j++){
                    if (i != j){
```

```
                        double distance = (means.get(i)[0] -
means.get(j)[0])*(means.get(i)[0] - means.get(j)[0]) + (means.get(i)[1] -
means.get(j)[1])*(means.get(i)[1] - means.get(j)[1]);
                        if (distance < minDistance || minDistance == 0){
                            minDistance = distance;
                            minI = i;
                            minJ = j;
                        }
                    }
                }
            }

            //  calculate new mean and merge clusters
            means.get(minI)[0] = (means.get(minI)[0]*clusters.get(minI).size() +
means.get(minJ)[0]*clusters.get(minJ).size())/(clusters.get(minI).size() +
clusters.get(minJ).size());
            means.get(minI)[1] = (means.get(minI)[1]*clusters.get(minI).size() +
means.get(minJ)[1]*clusters.get(minJ).size())/(clusters.get(minI).size() +
clusters.get(minJ).size());
            clusters.get(minI).addAll(clusters.get(minJ));

            means.remove(minJ);
            clusters.remove(minJ);

            // update sample points' categories, calculate purity, f-measure
            if (k == 2 || k == 3 || k == 4){
                for (int j=0; j<k; j++){
                    for (int i=0; i<clusters.get(j).size(); i++){
                       category[clusters.get(j).get(i)][1] = j+1;
                    }
                }

                // evaluate method (purity, f-measure)
                evaluate(category, dataSize, k);


                // print means and clusters (debugging)
                /*for (int j=0; j<k; j++){
                    System.out.println(means.get(j)[0] + "," + means.get(j)[1]);
                }
                for(int i=0; i<dataSize;i++){
                    System.out.println(x[i][0] + " " + x[i][1] + " " + category[i][0] + "
" + category[i][1]);
                }*/
            }
        }
    }


    // EVALUATE METHODS
    public static void evaluate (int[][] category, int dataSize, int k){
        // calculate purity
        int purityCounter = 0;
        int[][] majorCategory = new int[k][2];
        for (int j=0; j<k; j++){
            majorCategory[j][0] = 0;     //[k][0]: number of sample points in cluster k
with true category -1
            majorCategory[j][1] = 0;     //[k][1]: number of sample points in cluster k
with true category 1
            for(int i=0; i<dataSize; i++){
                if (category[i][1] == j+1){
                    if (category[i][0] == 1){
                        majorCategory[j][1]++;
                    }
                    else if (category[i][0] == -1){
                        majorCategory[j][0]++;
                    }
                }
            }
```

```
            for(int i=0; i<dataSize; i++){
                 if (category[i][1] == j+1){
                    if (majorCategory[j][1] >= majorCategory[j][0]){
                        if (category[i][0] == 1){
                            purityCounter++;
                        }
                    }
                    else if (majorCategory[j][1] < majorCategory[j][0]){
                        if (category[i][0] == -1){
                            purityCounter++;
                        }
                    }
                 }
            }
        }
        System.out.println("Purity (K = " + k + ") : " +
(double)purityCounter/(double)dataSize);


        // calculate F-measure (a=1)
        double totalFmeasure = 0;
        for (int j=0; j<k; j++){
            int truePos = 0;
            int trueNeg = 0;
            int falsePos = 0;
            int falseNeg = 0;
            for(int i=0; i<dataSize; i++){
                if (category[i][1] == j+1){
                    if ((category[i][0] == -1 && majorCategory[j][1] <
majorCategory[j][0]) || (category[i][0] == 1 && majorCategory[j][1] >=
majorCategory[j][0])){
                        truePos++;
                    }
                    else {
                        falsePos++;
                    }
                }
                else {
                    if ((category[i][0] == 1 && majorCategory[j][1] < majorCategory[j][0])
|| (category[i][0] == -1 && majorCategory[j][1] >= majorCategory[j][0])){
                        trueNeg++;
                    }
                    else {
                        falseNeg++;
                    }
                }
            }
            double precision = (double) truePos/(truePos + falsePos);
            double recall = (double) truePos/(truePos + falseNeg);

            double Fmeasure = 2 * ((precision*recall)/(precision+recall));
            totalFmeasure += Fmeasure;

            System.out.println("F-measure (k = " + (j+1) + "): " + Fmeasure);
        }
        System.out.println("Total F-measure: " + totalFmeasure);
        System.out.println("---------------------------------");
    }
}
```