

Φοιτητές:

Νικολαΐδου Βαΐα, AM: 4448

Γεώργιος Λυπόπουλος, AM: 4411

Δημήτρης Βουρδουγιάννης, AM: 4326

Άσκηση 1: Υλοποίηση MLP με Gradient Descent

Εισαγωγή

Αυτή η αναφορά περιγράφει την υλοποίηση ενός νευρωνικού δικτύου MLP στη γλώσσα προγραμματισμού C. Το δίκτυο εκπαιδεύεται χρησιμοποιώντας τον αλγόριθμο βελτιστοποίησης gradient descent και αξιολογείται για τις δυνατότητές του να γενικεύει χρησιμοποιώντας ένα σύνολο δεδομένων ελέγχου. Ο κώδικας έχει οργανωθεί σε διάφορες συναρτήσεις και δομές δεδομένων για να ενισχύσει την κατανόηση.

Δομές Δεδομένων

- **Network_t**: Αναπαριστά τη δομή του νευρωνικού δικτύου, περιλαμβάνοντας τα επίπεδα και τους νευρώνες.
- **Input_t**: Αποθηκεύει τα χαρακτηριστικά εισόδου (x1, x2) μαζί με τις αντίστοιχες ετικέτες κατηγορίας.
- **Neuron_t**: Αναπαριστά έναν μεμονωμένο νευρώνα, συμπεριλαμβανομένων των βαρών, των παραγώγων σφάλματος και των τιμών εξόδου.

Αρχικοποίηση (Συνάρτηση init)

- Φορτώνει τα σύνολα εκπαίδευσης και ελέγχου από αρχεία CSV (train_dataset.csv και test_dataset.csv).
- Αρχικοποιεί το νευρωνικό δίκτυο με τυχαία βάρη για όλους τους νευρώνες.
- Ορίζει παραμέτρους όπως ο αριθμός των νευρώνων ανά επίπεδο, οι συναρτήσεις ενεργοποίησης και άλλες ρυθμίσεις.

Κωδικοποίηση Κατηγοριών Εισόδου (Συνάρτηση encode_input)

- Κατηγοριοποιεί τα σημεία δεδομένων εισόδου σε τέσσερις κατηγορίες (C1, C2, C3 και C4) βάσει συγκεκριμένων συνθηκών.
- Χρησιμοποιεί μέθοδο κωδικοποίησης 1-από-p για να αντιστοιχίσει ετικέτες κατηγορίας σε τα δεδομένα εισόδου.

Συνάρτηση forward_pass

- Εκτελεί την προώθηση δεδομένων μέσα από το νευρωνικό δίκτυο.
- Υπολογίζει την έξοδο κάθε νευρώνα σε κάθε επίπεδο, συνδυάζοντας το βαρυτικό άθροισμα των εισόδων με μια συνάρτηση ενεργοποίησης.

Συνάρτηση reverse_pass

- Υλοποιεί την αντίστροφη διάβαση ή backpropagation για τον υπολογισμό των σημάτων σφάλματος (deltas) για κάθε νευρώνα.
- Χρησιμοποιεί τον κανόνα της αλυσίδας για να μεταδώσει τα σφάλματα προς τα πίσω μέσω των επιπέδων.

Εκπαίδευση με Gradient Descent (Συνάρτηση train_using_gradient_descent)

- Εκπαιδεύει το νευρωνικό δίκτυο χρησιμοποιώντας τη μέθοδο gradient descent.
- Επαναλαμβάνει τις εποχές και τα mini-batches για να ενημερώσει τα βάρη του δικτύου.
- Υπολογίζει και μερικές παραγώγους για τις ενημερώσεις των βαρών.
- Καταγράφει τα σφάλματα εκπαίδευσης, αποθηκεύοντας τα σε ένα αρχείο CSV για ανάλυση.
- Υπολογίζει το σφάλμα εκπαίδευσης στο τέλος κάθε εποχής.
- Το σφάλμα ορίζεται ως ο μέσος τετραγωνικός όρος μεταξύ των προβλεπόμενων και των πραγματικών ετικετών κατηγορίας.
- Ενημερώνει τα βάρη για κάθε νευρώνα σε κάθε επίπεδο χρησιμοποιώντας τις συσσωρευμένες μερικές παραγώγους και τον ρυθμό μάθησης (η).

Αξιολόγηση Γενίκευσης (Συνάρτηση calculate_generalization_capability)

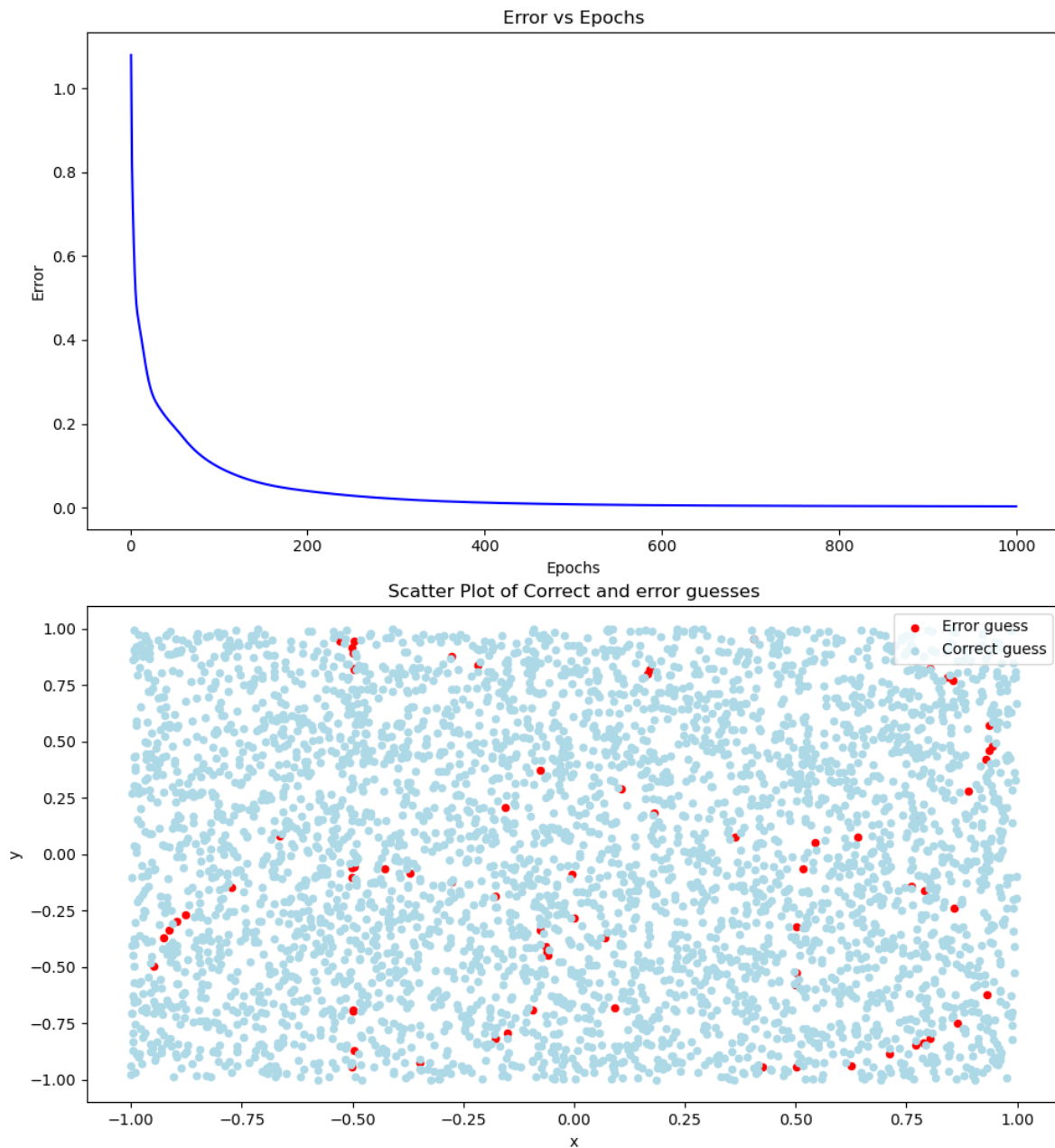
- Αξιολογεί τις δυνατότητες γενίκευσης του εκπαιδευμένου δικτύου χρησιμοποιώντας το σύνολο δεδομένων ελέγχου.
- Καταγράφει σωστά και εσφαλμένα ταξινομημένα σημεία δεδομένων.
- Υπολογίζει το ποσοστό σφάλματος και την ακρίβεια των προβλέψεων του δικτύου.

Αποτελέσματα:

H1	H2	H3	Learning rate	Batch size	Activation function	Accuracy
10	10	10	0.1	40	Logistic	96,75%
10	10	10	0.1	40	Tanh	18.7%
10	10	10	0.1	40	Relu	30.97%
10	30	10	0.1	40	Logistic	88.48%
10	30	10	0.1	40	Tanh	30.97%
30	30	30	0.1	40	Logistic	95.8%
30	30	30	0.1	40	Tanh	31.42%
30	30	30	0.1	40	Relu	30.97%
30	30	30	0,001	40	Logistic	97.52%
30	30	30	0,0001	40	Logistic	97.25%
30	30	30	0,01	400	Logistic	96.60%
50	50	50	0,001	40	Logistic	97.67%
10	50	50	0,001	40	Logistic	95.95%

H1	H2	H3	Learning rate	Batch size	Activation function	Accuracy
30	50	50	0,001	40	Logistic	97.28%
30	50	30	0,001	40	Logistic	97.55%
30	70	30	0,001	40	Logistic	97.35%
30	30	30	0,001	40	Logistic	97.12%
30	30	30	0,0001	40	Logistic	97.87%
30	50	50	0,0001	40	Logistic	98,30%

Παρατηρούμε ότι το δίκτυο είναι πιο ακριβές χρησιμοποιώντας H1: 30, H2: 50, H3: 50 με learning rate 0,0001 και batch size: 40.



Οπτικοποίηση των αποτελεσμάτων της καλύτερης περίπτωσης

Αρχεία και Περιγραφές

1. mlp.c

- Περιγραφή: Υλοποίηση του πολυεπίπεδου νευρωνικού δικτύου MLP.

2. mlp.h

- Περιγραφή: Περιέχει κώδικα για τη δημιουργία των συνόλων εκπαίδευσης και δοκιμής.

3. Makefile

- Περιγραφή: Makefile για τη μεταγλώττιση του προγράμματος.

4. generate_dataset.py

- Περιγραφή: Σενάριο Python για τη δημιουργία των συνόλων δοκιμής και εκπαίδευσης.

5. visualize.py

- Περιγραφή: Κώδικας για την οπτικοποίηση των αποτελεσμάτων.

Μεταγλώττιση και Εκτέλεση

Για να μεταγλωττίσετε και να εκτελέσετε το πρόγραμμα, ακολουθήστε αυτά τα βήματα:

- Ανοίξτε ένα τερματικό.
- Πλοηγηθείτε στον φάκελο του έργου που περιέχει τα αναφερόμενα αρχεία.
- Εκτελέστε τις παρακάτω εντολές:
`make && ./mlp`

Πρόσθετες Σημειώσεις

- Προσαρμόστε τις παραμέτρους και τις ρυθμίσεις εντός των αρχείων κώδικα (mlp.c, generate_dataset.py, κλπ.) όπως απαιτείται για τα file paths.
- Χρησιμοποιήστε τον κώδικα visualize.py για να οπτικοποιήσετε τα αποτελέσματα που προκύπτουν από την εκτέλεση του πολυεπίπεδου νευρωνικού δικτύου.

Άσκηση 2: Υλοποίηση K-Means Αλγορίθμου

Αυτή η αναφορά περιγράφει την υλοποίηση του K-Means αλγορίθμου για την ομαδοποίηση M ομάδων στη γλώσσα προγραμματισμού Java.

Περιγραφή Κώδικα

1) Δημιουργία τυχαίων σημείων

Για την αναπαράσταση των σημείων υλοποιήθηκε η κλάση με όνομα `DoublePoint` η οποία αναπαριστά τα τυχαία σημεία μας στο χώρο με την χρήση 2 `double` μεταβλητών `x`, `y`.

```
78 usages
public class DoublePoint {

    4 usages
    private double x;
    4 usages
    private double y;

    42 usages
    public DoublePoint(double x, double y){
        this.x = x;
        this.y = y;
    }

    4 usages
    public double getX() { return x; }

    no usages
    public void setX(double x) { this.x = x; }

    4 usages
    public double getY() { return y; }

    no usages
    public void setY(double y) { this.y = y; }

    @Override
    public String toString() {
        return "DoublePoint{" +
            "x=" + x +
            ", y=" + y +
            '}';
    }
}
```

Για την δημιουργία των τυχαίων σημείων υλοποιήθηκε η κλάση με όνομα `CreatePoints`. Η συγκεκριμένη κλάση είναι υπέθυνη για την δημιουργία αντικειμένων τύπου `DoublePoint` με τρόπο που περιγράφεται στην εκφώνηση της άσκησης. Αυτό επιτυγχάνεται με την χρήση της μεθόδου `generateRandomPoints`. Τα τυχαία

σημεία που δημιουργούνται με την ολοκλήρωση του προγράμματος αποθηκεύονται σε ένα txt file με όνομα points.txt.

```
1 usage
private static void generateRandomPoints() {
    List<DoublePoint> points = new ArrayList<>();
    Random random = new Random();

    for (int i = 0; i < 150; i++) {
        points.add(new DoublePoint((0.8 + 0.4 * random.nextDouble()), y: 0.8 + 0.4 * random.nextDouble()));
        points.add(new DoublePoint(x: 0.5 * random.nextDouble(), y: 0.5 * random.nextDouble()));
        points.add(new DoublePoint(x: 1.5 + 0.5 * random.nextDouble(), y: 0.5 * random.nextDouble()));
        points.add(new DoublePoint(x: 0.5 * random.nextDouble(), y: 1.5 + 0.5 * random.nextDouble()));
        points.add(new DoublePoint(x: 1.5 + 0.5 * random.nextDouble(), y: 1.5 + 0.5 * random.nextDouble()));
        points.add(new DoublePoint(x: 2 * random.nextDouble(), y: 2 * random.nextDouble()));
    }

    for (int j = 0; j < 75; j++) {
        points.add(new DoublePoint(x: 0.4 * random.nextDouble(), y: 0.8 + 0.4 * random.nextDouble()));
        points.add(new DoublePoint(x: 1.6 + 0.4 * random.nextDouble(), y: 0.8 + 0.4 * random.nextDouble()));
        points.add(new DoublePoint(x: 0.8 + 0.4 * random.nextDouble(), y: 0.3 + 0.4 * random.nextDouble()));
        points.add(new DoublePoint(x: 0.8 + 0.4 * random.nextDouble(), y: 1.3 + 0.4 * random.nextDouble()));
    }

    writePointsToFile(points, filename: "points.txt");
}
```

```
1 0.9031521277109835 1.0854061998778065
2 0.4478458577304104 0.24887200306477375
3 1.6367631554643345 0.29293352551949425
4 0.11227477500670024 1.939162697201853
5 1.7567536034052351 1.5330628359845822
6 1.144894535144474 0.07975205219629977
7 1.022367374101813 0.9543575251840528
8 0.1486844577249058 0.10171207742173816
9 1.5331602385083019 0.18843386270345192
10 0.123314532257297 1.5833126732145058
11 1.9404767149327555 1.9626581920273196
12 1.8916352353429684 0.35426430168275
13 1.089820147105813 1.052792500523886
14 0.12256165734751279 0.37426919934057507
15 1.944597903456969 0.3288038742923005
16 0.041530685214191676 1.9731360254956993
17 1.6633262882616622 1.7308292839958617
18 0.6898049798230415 1.5779509279195716
19 1.0055227476477557 1.1602191675971696
20 0.4221665654342517 0.03016241721461388
21 1.569678662513922 0.4708873487251741
22 0.3380899199329063 1.5034529024540584
23 1.564800605280511 1.535811265894765
24 1.846765075051866 0.21935084274944816
25 1.1128588349540007 0.9306522189142769
26 0.35900167960310675 0.1798525957407347
27 1.904304735646643 0.38141816272622375
28 0.4769542595945538 1.6216569067609679
29 1.9504093008069745 1.5459079251866918
30 1.055726771177554 1.8947245318138963
31 0.8119214794083297 0.8635413327356196
32 0.4230655074049341 0.06619905479298793
33 1.5313928565878938 0.4580489708421121
34 0.23763376771977712 1.9947498595599147
35 1.720320805594921 1.7761803870474604
36 0.47731606292260764 0.3921709340364594
37 1.0387115812931929 0.9924970477128159
38 0.38221949510980346 0.4083083285122979
39 1.759042952332806 0.48028609351843327
40 0.2379022100462458 1.5780187285778202
```

Στιγμιότυπο από το αρχείο points.txt

2) Υλοποίηση K-Means Αλγορίθμου

Πρώτο βήμα αποτελεί η φορτώσει των τυχαίων σημείων που έχουμε αποθήκευση μέσα στο αρχείο points.txt. Σε αυτό βοηθάει η δομή δεδομένων `List<DoublePoint> points` με το την χρήση της μεθόδου `readPoints` που βρίσκεται μέσα στην κλάση `Main`. Η υλοποίηση του αλγορίθμου γίνεται στην κλάση με όνομα `Kmeans`.

Αρχικό βήμα για τον αλγόριθμο αποτελεί η αρχικοποίηση του μετρητή επαναλήψεων και των κέντρων `M`. Στην συγκεκριμένη υλοποίηση ορίσαμε τον αριθμό επαναλήψεων ίσο με 1000. Για την επιλογή τυχαίων κέντρων βοήθησε η συνάρτηση που έχει υλοποιηθεί μέσα στην κλάση `Kmeans` με όνομα `initializeCenters`.

```
1 usage
50 @ private List<DoublePoint> initializeCenters(long seed) {
51     List<DoublePoint> centers = new ArrayList<>();
52     Random random = new Random(seed);
53     for (int i = 0; i < numOfCenters; i++) {
54         int randomIndex = random.nextInt(points.size());
55         centers.add(points.get(randomIndex));
56     }
57
58     return centers;
59 }
```

Μια βασική λεπτομέρεια που πρέπει να αναλύσουμε είναι η επιλογή χρήσης `seed` για την μεταβλητή `random`. Συγκεκριμένα αυτό που θέλουμε να πετύχουμε με την συγκεκριμένη προσθήκη είναι η δυνατότητα να παράγουμε κάθε φορά τυχαία κέντρα τα οποία όμως να έχουμε την δυνατότητα σε επόμενη προσομοίωση να τα ξανά παράξουμε. Επομένως εφόσον μας ζητείται στην συγκεκριμένη άσκηση να τρέξουμε 15 διαφορετικές προσομοιώσεις για κάθε διαφορετικό αριθμό `M` πλέον έχουμε την δυνατότητα να ορίσουμε για κάθε μια από αυτές τις 15 προσομοιώσεις και διαφορετικό αριθμό `seed` με αποτέλεσμα και την παραγωγή κάθε φορά διαφορετικών κέντρων με την διαφορά όμως ότι πλέον έχουμε την δυνατότητα να ξανά παράξουμε τα ίδια κέντρα μεμονομένα σε μελλοντικές προσομοιώσεις.

Επόμενο βήμα για την υλοποίηση του αλγορίθμου αποτελεί ο υπολογισμός της Ευκλείδειας απόστασης για κάθε σημείο, που έχει αποθηκευτεί μέσα στην `points`, από τα κέντρα που έχουν ήδη οριστεί. Με σκοπό την τοποθέτηση καθενός από αυτά τα σημεία σε ομάδα της οποίας το κέντρο βρίσκεται σε μικρότερη απόσταση

από αυτά των άλλων ομάδων. Αυτό επιτυγχάνετε με την χρήση των μεθόδων `assignToClusters` και `findClosestCentroid`.

2 usages

```
private List<List<DoublePoint>> assignToClusters(List<DoublePoint> centroids) {  
    List<List<DoublePoint>> clusters = new ArrayList<>();  
  
    for (int i = 0; i < numOfCenters; i++) {  
        clusters.add(new ArrayList<>());  
    }  
  
    for (DoublePoint point : points) {  
        int clusterIndex = findClosestCentroid(point, centroids);  
        clusters.get(clusterIndex).add(point);  
    }  
  
    return clusters;  
}
```

1 usage

```
private int findClosestCentroid(DoublePoint point, List<DoublePoint> centers) {  
    int closestIndex = 0;  
    double closestDist = findDist(point, centers.get(0));  
  
    for (int i = 1; i < centers.size(); i++) {  
        double currDist = findDist(point, centers.get(i));  
        if (currDist < closestDist) {  
            closestIndex = i;  
            closestDist = currDist;  
        }  
    }  
  
    return closestIndex;  
}
```

Επόμενο και αρκετά σημαντικό βήμα αποτελεί η ανανέωση των κέντρων βάσης του μέσου όρου των στοιχείων που ανήκουν στην ομάδα του συγκεκριμένου κέντρου. Αυτό επιτυγχάνετε με την χρήση της μεθόδου `updateCenters`.

```
1 usage
private List<DoublePoint> updateCenters(List<List<DoublePoint>> clusters) {
    List<DoublePoint> newCenters = new ArrayList<>();

    for (List<DoublePoint> cluster : clusters) {
        if (!cluster.isEmpty()) {
            double sumX = 0;
            double sumY = 0;

            for (DoublePoint point : cluster) {
                sumX += point.getX();
                sumY += point.getY();
            }

            double centerX = sumX / cluster.size();
            double centerY = sumY / cluster.size();

            newCenters.add(new DoublePoint(centerX, centerY));
        }
    }

    return newCenters;
}
```

Επόμενο βήμα αποτελεί ο έλεγχος τερματισμού του αλγορίθμου. Η βασική προϋπόθεση για την έναρξη νέας επανάληψης αποτελεί η σύγκριση των παλιών με των νέων κέντρων. Αν τα κέντρα παρουσιάζουν μεταβολή σε σχέση με τα προηγούμενα τότε ο αλγόριθμος συνεχίζει κανονικά. Σε άλλη περίπτωση θεωρούμε ότι ο αλγόριθμος έχει συγκλίνει και γίνεται τερματισμός.

Στην συγκεκριμένη υλοποίηση έχουμε ορίσει ένα threshold το οποίο ορίζει την ελάχιστη μεταβολή που θα πρέπει να έχει συμβεί στα κέντρα για να θεωρήσουμε ότι ο αλγόριθμος έχει συγκλίνει. Η μεταβλητή αυτή είναι η EPSILON. Ο έλεγχος αυτός γίνεται με την χρήση της συνάντησης hasConverged πριν την έναρξη νέας επανάληψης.

```
1 usage
private boolean hasConverged(List<DoublePoint> oldCentroids, List<DoublePoint> newCentroids) {
    for (int i = 0; i < oldCentroids.size(); i++) {
        if (findDist(oldCentroids.get(i), newCentroids.get(i)) > EPSILON) {
            return false;
        }
    }

    return true;
}
```

Τελικό βήμα αποτελεί ο υπολογισμός του σφάλματος ομαδοποίησης της συγκεκριμένης προσομοίωσης. Αυτό επιτυγχάνεται με τον άθροισμα των αποστάσεων κάθε σημείου από το κέντρο της ομάδας στην οποία ανήκει. Στην δικιά μας προσομοίωση υλοποιείται με την χρήση της μεθόδου

```
1 usage
127 public double calculateTotalDistance() {
128     totalDistance = 0;
129
130     for (int i = 0; i < numOfCenters; i++) {
131         DoublePoint center = centers.get(i);
132
133         for (DoublePoint point : clusters.get(i)) {
134             double dist = findDist(point, center);
135             totalDistance += dist;
136         }
137     }
138
139     System.out.println("clustering error: " + totalDistance);
140     return totalDistance;
141 }
142
```

3) Αποτελέσματα

Τα αποτελέσματα των προσομοιώσεων αποθηκεύονται σε ένα txt file με όνομα results_simulation. Πιο συγκεκριμένα σε αυτό το αρχείο διατηρούνται για κάθε διαφορετική προσομοίωση τα τελικά κέντρα ο αριθμός των clusters που έχει οριστεί καθώς και το Total Error(σφάλμα ομαδοποίησης). Παρακάτω φαίνεται

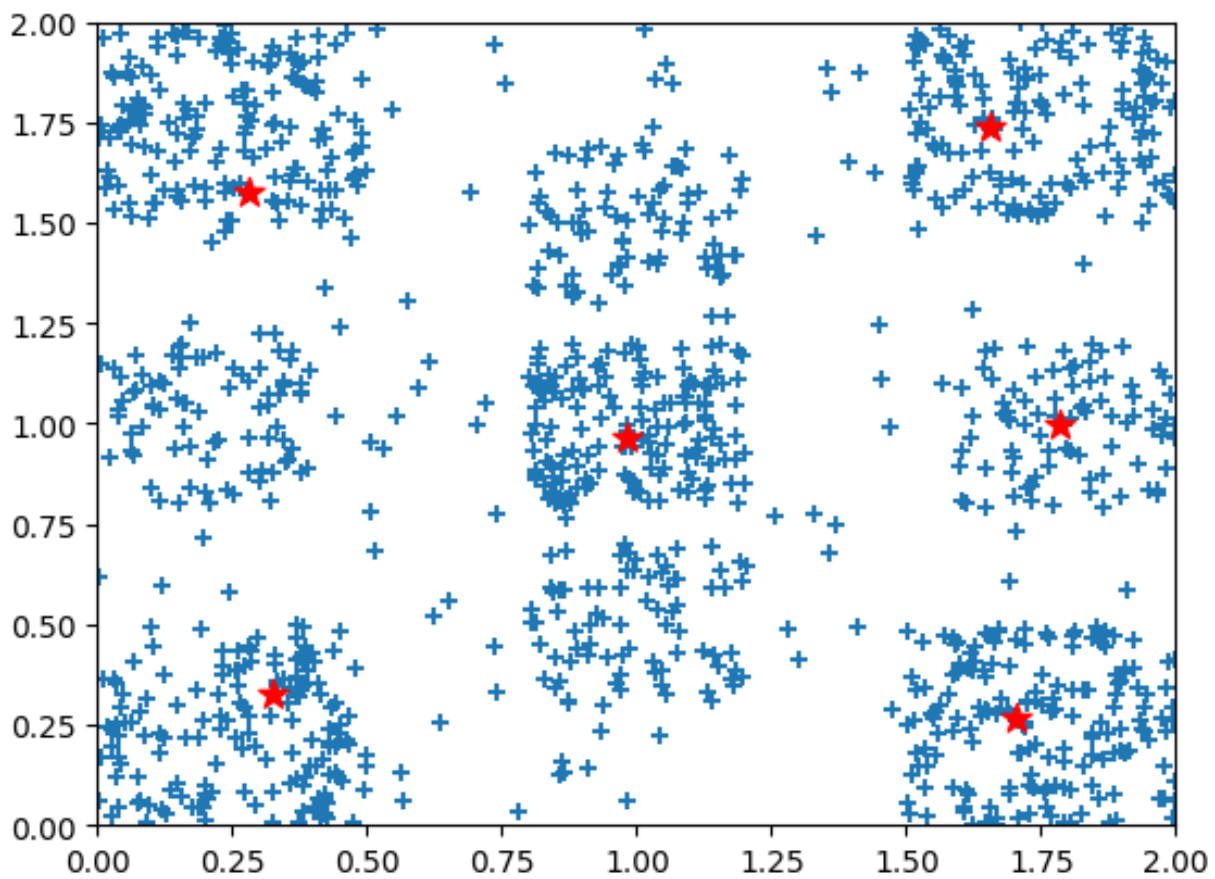
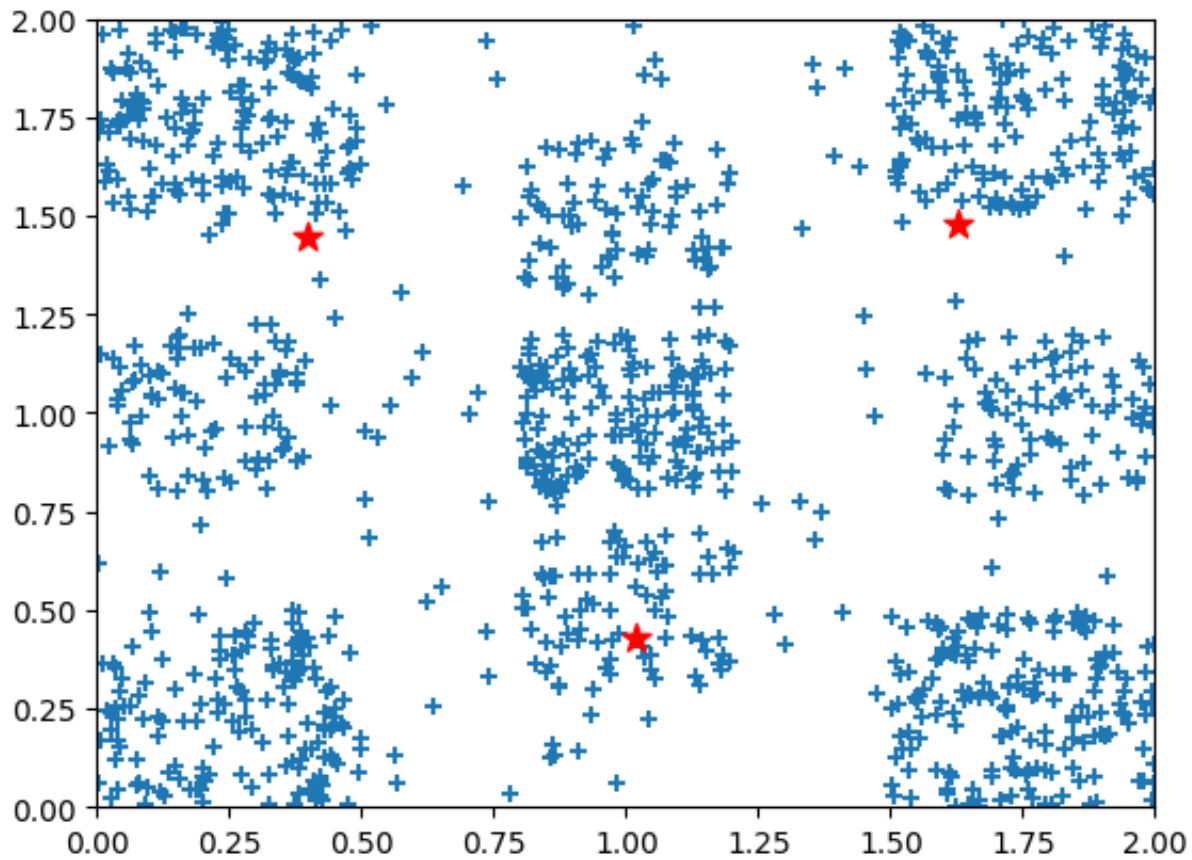
```
2 Centers: [DoublePoint{x=1.018635647710859, y=0.42749112474977125}, DoublePoint{x=1.6283498268788184, y=1.4779169812414308}, DoublePoint{x=0.3966620575814449, y=1.4460304394188646}]
3 Number of Clusters: 3
4 Total Error: 618.8373874839423
5 #####
```

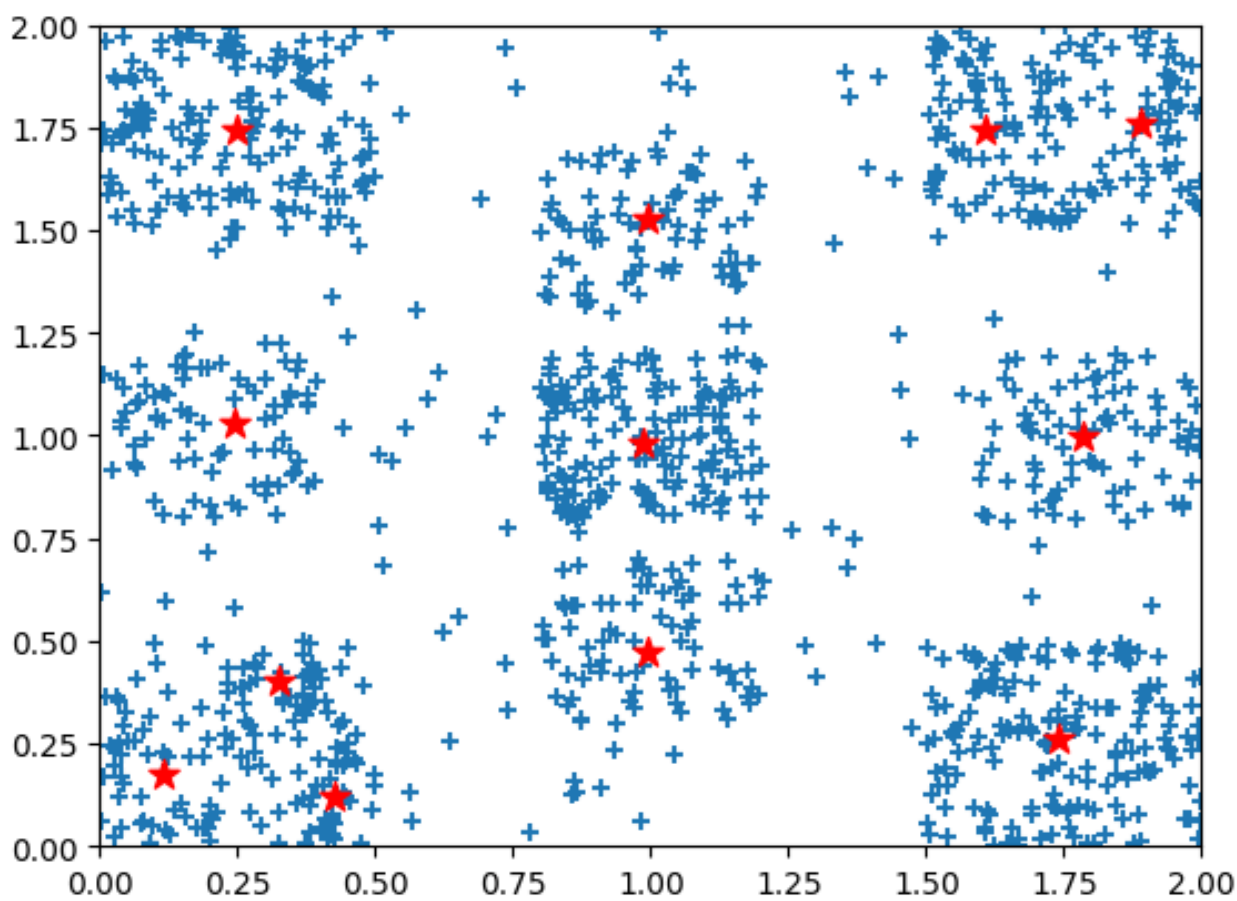
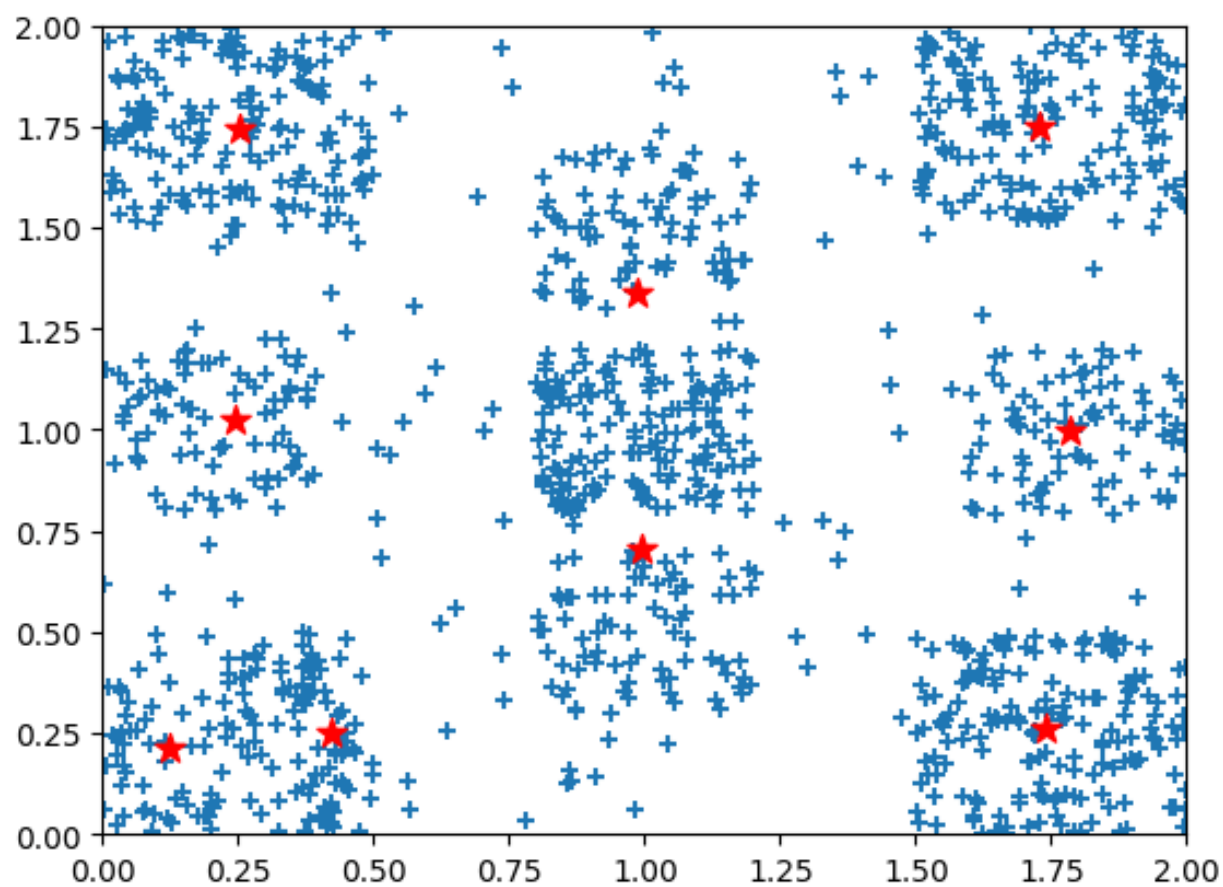
ένα στιγμιότυπο.

Για τα ζητούμενα της άσκησης με την ολοκλήρωση όλων των προσομοιώσεων για κάθε διαφορετικό αριθμό του M επιλέχθηκαν μονάχα οι προσομοιώσεις που φέρουν το μικρότερο σφάλμα ομαδοποίησης.

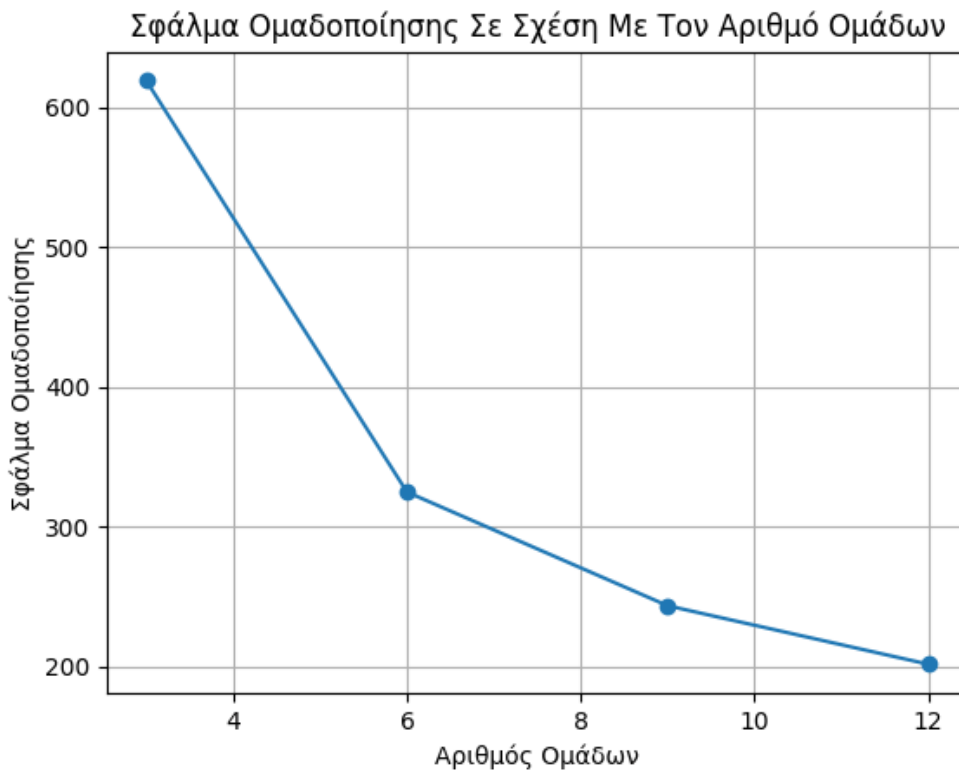
```
1 M = 3
2 Centers: [DoublePoint{x=1.018635647710859, y=0.42749112474977125},
3 Number of Clusters: 3
4 Total Error: 618.8373874839423
5 #####
6
7 M = 6
8 Centers: [DoublePoint{x=0.32558948352972744, y=0.32581414512453793},
9 Number of Clusters: 6
10 Total Error: 324.7315580433579
11 #####
12
13 M = 9
14 Centers: [DoublePoint{x=0.9874601772877413, y=1.338560079841218}, D
15 Number of Clusters: 9
16 Total Error: 243.4344195961283
17 #####
18
19 M = 12
20 Centers: [DoublePoint{x=1.7842014507664128, y=0.9991510605699042},
21 Number of Clusters: 12
22 Total Error: 201.60075665242567
23 #####
24
```

Στην συνέχεια για κάθε μια από αυτές τις 4 περιπτώσεις δημιουργήθηκαν τα κατάλληλα plots.







Στην συνέχεια εμφανίζεται το plot που δείχνει πώς μεταβάλλεται το σφάλμα ομαδοποίησης με τον αριθμό των ομάδων.



Παρατηρούμε ότι μπορούμε να εκτιμήσουμε τον πραγματικό αριθμό ομάδων με τη χρήση του σφάλματος ομαδοποίησης και συγκεκριμένα στο παράδειγμά μας, ορίζοντας το $m=9$, παρατηρούμε ότι το σφάλμα ομαδοποίησης έχει ελαχιστοποιηθεί αρκετά σε σχέση με τις προηγούμενες τιμές. Για $m>9$ παρατηρούμε μικρές βελτιώσεις στο σφάλμα ομαδοποίησης, κάτι το οποίο μας δείχνει ότι η επιλογή για $m=9$ αποτελεί τη βέλτιστη λύση.

Εκτέλεση Προγράμματος

Για την εύκολη και γρήγορη εκτέλεση των προσομοιώσεων δημιουργήθηκε ένα bash script με όνομα

```
1   #!/bin/bash
2  for i in {1..15}; do
3      java -cp out/production/NNPROJECT Main 3 $i
4  done
5
6  for i in {1..15}; do
7      java -cp out/production/NNPROJECT Main 6 $i
8  done
9
10 for i in {1..15}; do
11     java -cp out/production/NNPROJECT Main 9 $i
12 done
13
14 for i in {1..15}; do
15      java -cp out/production/NNPROJECT Main 12 $i
16 done
```

bash.sh.

Μπορούμε να καταλάβουμε ότι για να τρέξουμε μεμονομένα μια προσομοίωση αρκεί να δώσουμε την εντολή:

— **java Main numberOfM seed**

- 1) **numberOfM** να είναι ένας int (3,6,9,12).
- 2) **Seed** ένας τυχαίος int αριθμός.

Σε περίπτωση που θέλουμε να δημιουργήσουμε νέα points αρκεί να τρέξουμε την εντολή

— **java CreatePoints**