ΥΠΟΛΟΓΙΣΤΙΚΗ ΝΟΗΜΟΣΥΝΗ

ΕΡΓΑΣΤΗΡΙΑΚΕΣ ΑΣΚΗΣΕΙΣ

2021-2022

ΕΥΤΥΧΙΑ ΜΠΟΥΡΛΗ ΑΜ: 4441

ΔΗΜΗΤΡΗΣ ΒΑΓΕΝΑΣ ΑΜ: 2941

Οδηγίες εγκατάστασης του DirectX11 σε Ubuntu για την εμφάνιση των εικόνων

Για να εμφανιστούν οι εικόνες με τα σημεία όπως φαίνεται στο report πρέπει ο υπολογιστής να έχει εγκατεστημένο το DirectX11.

Στα Ubuntu μπορεί να εγκατασταθεί αν δεν υπάρχει με τις παρακάτω εντολές:

sudo apt install libvulkan1 libvulkan-dev vulkan-utils

sudo add-apt-repository ppa:oibaf/graphics-drivers

sudo apt update

sudo apt upgrade

sudo apt install mesa-vulkan-drivers

sudo add-apt-repository ppa:graphics-drivers/ppa

sudo apt update

sudo apt upgrade

Έχει και την παρακάτω εντολή η οποία δεν έτρεξε. Παρόλα αυτά οι εικόνες τυπώνονται κανονικά.

sudo apt install vulkan

Πηγή: https://linuxconfig.org/improve-your-wine-gaming-on-linux-with-dxvk

Οδηγίες εκτέλεσης

Σε κάθε φάκελο για το αντίστοιχο πρόγραμμα (P1, P2, P3) υπάρχει ένας

φάκελος terminal και ένας eclipse.

Αν θέλετε να τρέξετε το πρώτο πρόγραμμα από το τερματικό απλά ανοίξτε τερματικό

στον φάκελο P1terminal και εκτελέστε: java PerceptronTwoLevels.

Αν θέλετε να τρέξετε το πρώτο πρόγραμμα με το eclipse κάντε

import τον φάκελο P1eclipse και εκτελέστε το αρχείο PerceptronTwoLevels σαν εφαρμογή.

Αντίστοιχα για το δεύτερο πρόγραμμα:

Αν θέλετε να τρέξετε το δεύτερο πρόγραμμα από το τερματικό απλά ανοίξτε τερματικό

στον φάκελο P2terminal και εκτελέστε: java Perceptron3Layers

Αν θέλετε να τρέξετε το δεύτερο πρόγραμμα με το eclipse κάντε

import τον φάκελο P2eclipse και εκτελέστε το αρχείο java Perceptron3Layers σαν εφαρμογή.

Αντίστοιχα για το τρίτο πρόγραμμα:

Αν θέλετε να τρέξετε το τρίτο πρόγραμμα από το τερματικό απλά ανοίξτε τερματικό

στον φάκελο P3terminal και εκτελέστε: java Kmean

Αν θέλετε να τρέξετε το τρίτο πρόγραμμα με το eclipse κάντε

import τον φάκελο P3eclipse και εκτελέστε το αρχείο java Kmean σαν εφαρμογή.

Άσκηση 1 : προγράμματα ταξινόμησης βασισμένα στο πολυεπίπεδο perceptron (MLP). Το πρόγραμμα Π1 υλοποιεί MLP με δύο κρυμμένα επίπεδα και το πρόγραμμα Π2 υλοποιεί MLP με τρία κρυμμένα επίπεδα.

Σύνολο Σ1:

Δημιουργήσαμε ένα αρχείο με τις συντεταγμένες των τυχαίων σημείων του συνόλου Σ1 και το διατηρήσαμε ώστε όταν τρέχουμε το πρόγραμμα με διαφορετικές κάθε φορά αρχικοποιήσεις να λαμβάνουμε αντικειμενικά αποτελέσματα.

Πρόγραμμα Π1 και Π2:

Τα δύο αυτά προγράμματα είναι παρόμοια με την μόνη διαφορά ότι στο Π2 υλοποιούμε ένα παραπάνω κρυμμένο επίπεδο. Για την υλοποίησή τους σχεδιάσαμε 3 classes και στα δύο. Δεν προλάβαμε να υλοποιήσουμε το Β όπου ανάλογα θα είχαμε σειριακή ή ομαδική ενημέρωση. Τώρα κάνουμε ομαδική.

PerceptronTwoLevels:

Η κλάση αυτή περιέχει την main μέθοδο του προγράμματός μας.

Στην αρχή γίνονται αρχικοποιήσεις τιμών και πινάκων όπου δίνεται ο ρυθμός μάθησης, ο αριθμός των νευρώνων ανά επίπεδο, το κατώφλι σφάλματος και η επιλογή για τον αν θέλουμε να χρησιμοποιήσουμε για συνάρτηση ενεργοποίησης την ReLU ή την υπερβολική εφαπτομένη.

Μέσα στην main διαβάζουμε το αρχείο learning.txt μέσα στο οποίο έχουμε τα μισά παραδείγματα με κάποια από αυτά να βρίσκονται σε λάθος ομάδες και καλούμε την μέθοδο learn().

Η μέθοδος learn() αρχικά δίνει random τιμές στα βάρη και τιμή 0 στις πολώσεις.

Στην συνέχεια η διαδικασία μέσα στην while είναι

-καθαρισμός πινάκων μερικού σφάλματος

-υπολογισμός εξόδων σε κάθε επίπεδο με forwardPass()

-υπολογισμός σφάλματος με backprop()

-ανανέωση των βαρών

-Υπολογισμός σφάλματος εποχής και σύγκρισή του με το σφάλμα της προηγούμενης.

Αφού τρέξει 700 εποχές, αν η διαφορά των σφαλμάτων μεταξύ δύο εποχών είναι μικρότερη από το κατώφλι που έχουμε ορίσει, τότε τερματίζουμε και στην γραμμή 131 ξεκινάμε τον έλεγχο. Στον έλεγχο, διαβάζουμε το αρχείο checking.txt που περιλαμβάνει τα άλλα μισά παραδείγματα χωρίς λάθη εκτελώντας forwardPass() χρησιμοποιώντας τα βάρη που υπολογίσαμε .

Η μέθοδος updateWeight() ανανεώνει τα βάρη

$$w_i(t+1)=w_i(t)-\eta\frac{\partial E}{\partial w_i},\ i=1,..,L$$

Η μέθοδος cleaner() καθαρίζει τους πίνακες errorOl1,2,3.

Η μέθοδος backprop() υπολογίζει τα σφάλματα «δ» των εξόδων κάθε επιπέδου και στη συνέχεια κάνουμε μία τριπλή for για να υπολογίσουμε το άθροισμα των παραγώγων για κάθε βάρος (ομαδική ενημέρωση) και άλλη μία διπλή for για τις πολώσεις για τα weight3 και το κάνουμε αυτό άλλες δύο φορές για τους πίνακες weight2 και weight1

Μερική παράγωγος βάρους σύνδεσης:

σφάλμα προορισμού x έξοδος πηγής

$$\frac{\partial E^n}{\partial w_{ij}^{(h)}}=\delta_i^{(h)}y_j^{(h-1)}$$

Μερική παράγωγος πόλωσης = σφάλμα του νευρώνα

$$\frac{\partial E^n}{\partial w_{i0}^{(h)}}=\delta_i^{(h)}$$

$$\frac{\partial E}{\partial w_i}:=\frac{\partial E}{\partial w_i}+\frac{\partial E^n}{\partial w_i}$$

Η μέθοδος calculateExodusError() υπολογίζει τα δέλτα των εξόδων του επιπέδου της εξόδου.

Νευρώνες εξόδου (επίπεδο H+1) (συν. ενεργοποίησης g_{H+1})

$$\delta_i^{(H+1)}=g'_{H+1}(u_i^{(H+1)})(o_i-t_{ni}),\ i=1,...,p$$

Η μέθοδος calculateLevelError() υπολογίζει τα δέλτα των εξόδων ενός κρυφού επιπέδου.

Νευρώνες κρυμμένων επιπέδων: για επίπεδο h=H,...,1 (συν. ενεργοποίησης g_h)

$$\delta_i^{(h)}=g'_h(u_i^{(h)})\sum_{j=1}^{d_{h+1}}w_{ji}^{(h+1)}\delta_j^{(h+1)},\ \ i=1,...,d_h$$

Η μέθοδος forwardPass() υπολογίζει τις τιμές των εξόδων του επιπέδου.

(Και για τα ενδιάμεσα επίπεδα και για να δώσουμε την τελική έξοδο χρησιμοποιούμε την ίδια συνάρτηση ενεργοποίησης)

Η μέθοδος createWeights() δίνει random τιμές στα βάρη και 0 τιμή στις πολώσεις.

(Οι πολώσεις είναι αποθηκευμένες στην γραμμή 0 του πίνακα βαρών.)

Η μέθοδος relu() και η tanh() υλοποιούν τις δύο συναρτήσεις ενεργοποίησης.

Η μέθοδος readData() διαβάζει συντεταγμένες σημείων από αρχείο txt.

Η μέθοδος showGraph() τυπώνει τη γραφική παράσταση των σημείων με διαφορετικά χρώματα ανά ομάδα.

Η μέθοδος generateNewData() δημιουργεί αρχείο.

(Αρχικά τα δύο υποσύνολα –ελέγχου και μάθησης- ήταν σε ένα αρχείο στο Data.txt το οποίο το σπάσαμε σε δύο διαφορετικά αρχεία το learning.txt και checking.txt

Η μέθοδος distance2() υπολογίζει απόσταση μεταξύ δύο σημείων.

Η μέθοδος dataGenerator() χρησιμοποιήθηκε για να δημιουργήσουμε 4000 τυχαία σημεία.

Η μέθοδος randomGenerator() δίνει τυχαίες τιμές εντός κάποιου διαστήματος.

Η μέθοδος pointGenerator() παράγει ένα τυχαίο σημείο και του αναθέτουμε σε ποια ομάδα ανήκει.

(Αυτή η μέθοδος χρησιμοποιήθηκε με error1 =11 για τα 4000 σημεία που χρησιμοποιούνται για τον έλεγχο και με error = r.nextInt(100) για τα υπόλοιπα 4000 σημεία του συνόλου εκπαίδευσης που έχουν πιθανότητα 0.1 να βρίσκονται σε λάθος ομάδα).

<u>Point:</u>

Η κλάση αυτή ορίζει το αντικείμενο «point» με πεδία x,y που είναι οι συντεταγμένες του, το clusterId που κρατάει το όνομα της ομάδας στην οποία ανήκει.

Graph:

Η κλάση αυτή περιέχει τον κώδικα για τον σχηματισμό των plot που τυπώνουμε.

Προκύπτει κάποιο όφελος από τη χρήση του τρίτου κρυμμένου επιπέδου;

Στο δικό μας πρόβλημα παρατηρήσαμε ότι το 3ο κρυμμένο επίπεδο δεν είχε μικρότερο σφάλμα αλλά μεγαλύτερο. Ωστόσο, το αν κάποιο επιπλέον επίπεδο θα βοηθήσει ή όχι εξαρτάται από την πολυπλοκότητα του προβλήματός μας.

Αν βάλουμε είτε λιγότερα είτε παραπάνω επίπεδα από όσα χρειάζονται τότε το σφάλμα μεγαλώνει.(έχουμε υποεκπαίδευση και υπερεκπαίδευση)

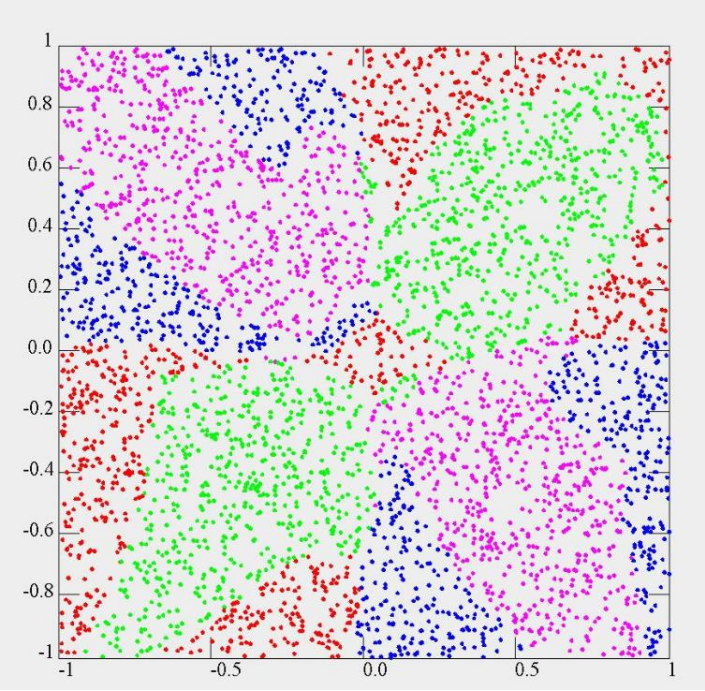
Παράδειγμα:

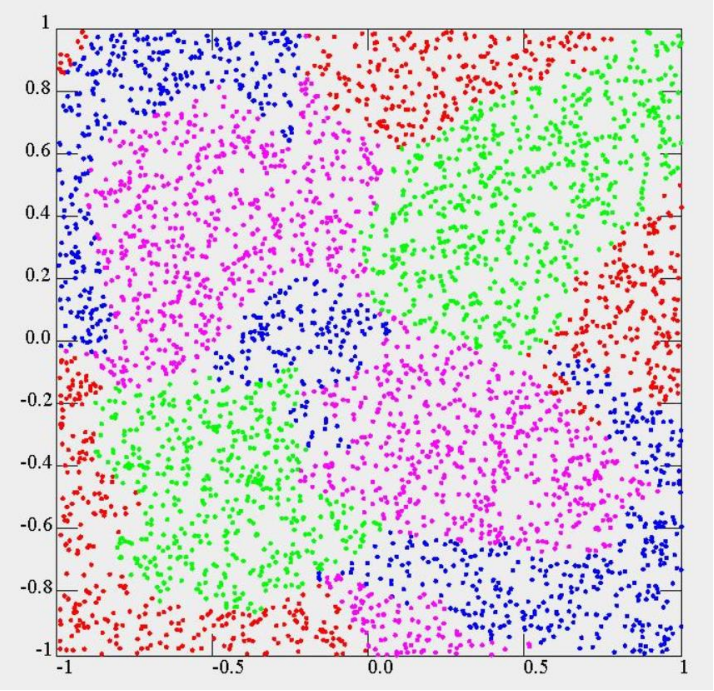
```
private static double n = 0.000002;
private static int B = 1;
private static int h1 = 30;
private static int h2 = 30;
private static double katwfli = 0.1;
private static boolean pickRelu = true;
```

```
private static double n = 0.000002;
private static int B = 1;
private static int h1 = 30;
private static int h2 = 30;
private static int h3 = 30;
private static double katwfli = 0.1;
private static boolean pickRelu = true;
```

Π1: Σωστά 2663/4000

Π2: Σωστά 2482/4000





Παρατηρήσεις:

Με τα συγκεκριμένα ορίσματα παρατηρούμε επίσης ότι έχουμε και στα δύο υπερεκπαίδευση. Κάτι που προσπαθήσαμε να βελτιώσουμε δοκιμάζοντας διαφορικές τιμές κατωφλίου.

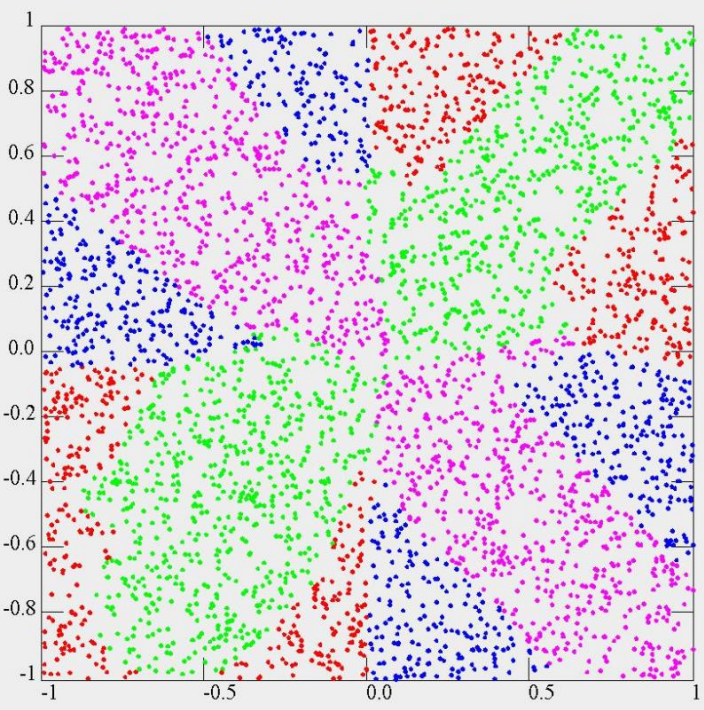
Επίσης παρατηρήσαμε ότι όσους περισσότερους νευρώνες χρησιμοποιούσαμε τόσο πιο μεγάλη ακρίβεια πετυχαίναμε.

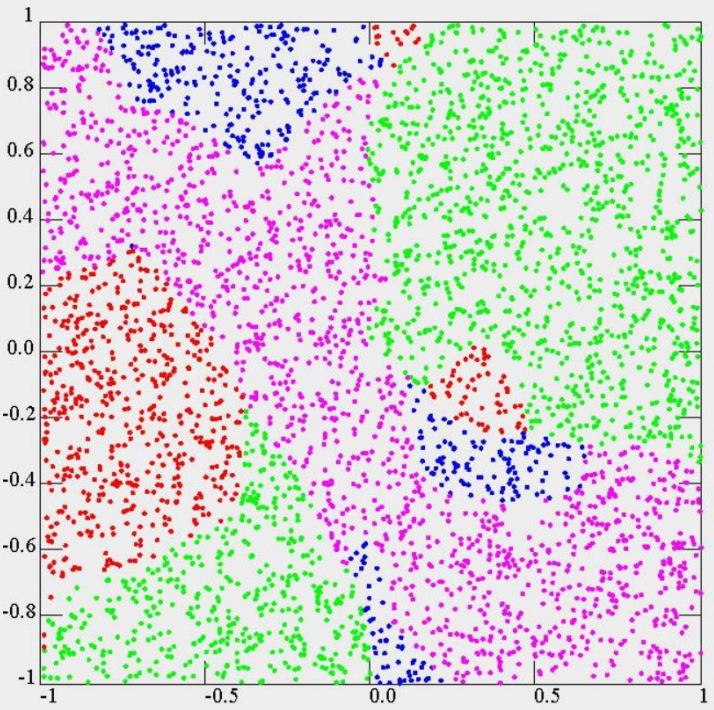
Επίσης παρατηρήσαμε ότι για μεγάλο ρυθμό μάθησης το σφάλμα ανά εποχή αυξομειώνονταν συνέχεια χωρίς να μπορεί να μειωθεί πολύ.

Επιπλέον παραδείγματα με διαφορετικά ορίσματα:

```
private static String data = "";
private static int tes = 4000;
private static double katwfli = 0.1;  // 0.6  0.8
private static String[] arrOfStr= new String[tes];
private static Point[] points = new Point[tes];
private static int h1 = 30; //30
private static int h2 = 40; //30
```

```
private static double n = 0.000001;
private static int B = 1;
private static int h1 = 10;
private static int h2 = 10;
private static int h3 = 10;
private static double katwfli = 0.1;
private static boolean pickRelu = true;
```





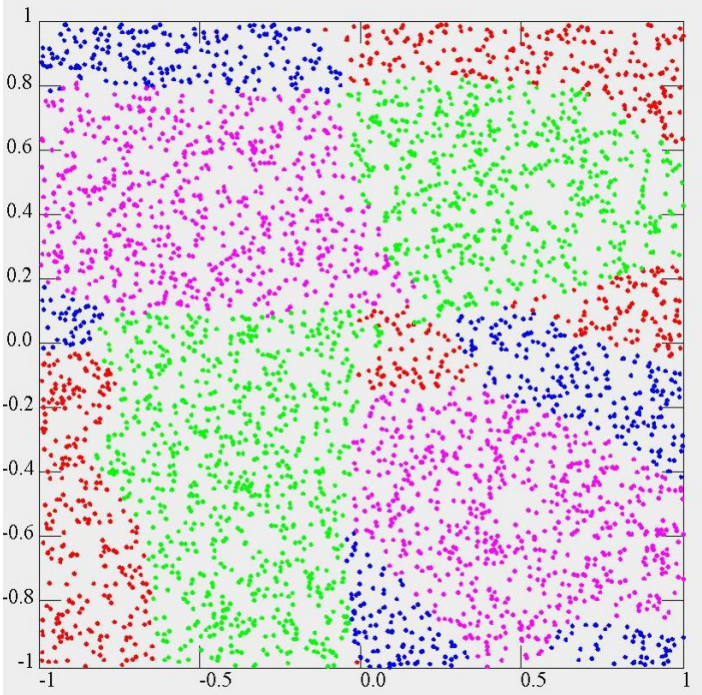
Π1: Σωστά 2552/4000

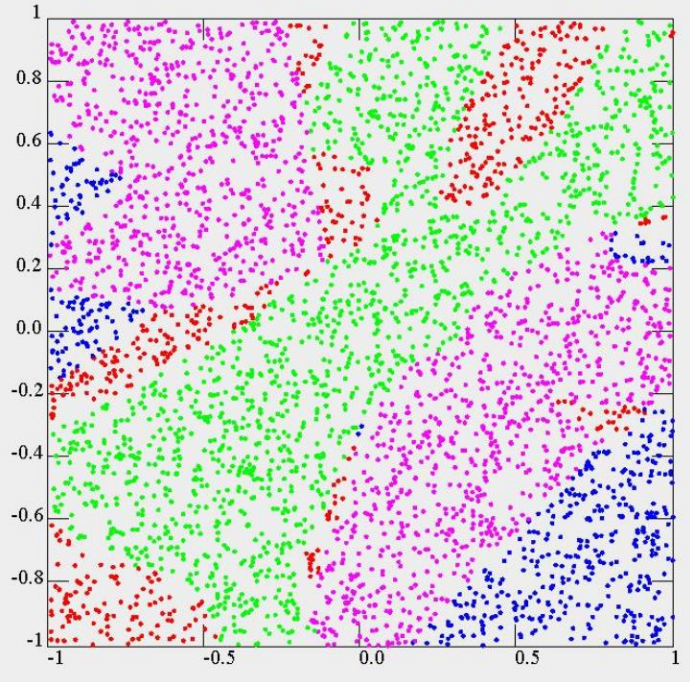
Π2: Σωστά 1795/4000

```
private static String data = "";
private static int tes = 4000;
private static double katwfli = 0.1;  // 0.6  0.8
private static String[] arrOfStr= new String[tes];
private static Point[] points = new Point[tes];
private static int h1 = 20; //30
private static int h2 = 20; //30
```

```
private static double n = 0.000002; //00001
```

```
private static double n = 0.000002;
//private static int B = 1;
private static int h1 = 20;
private static int h2 = 20;
private static int h3 = 20;
private static double katwfli = 0.1;
private static boolean pickRelu = true;
```

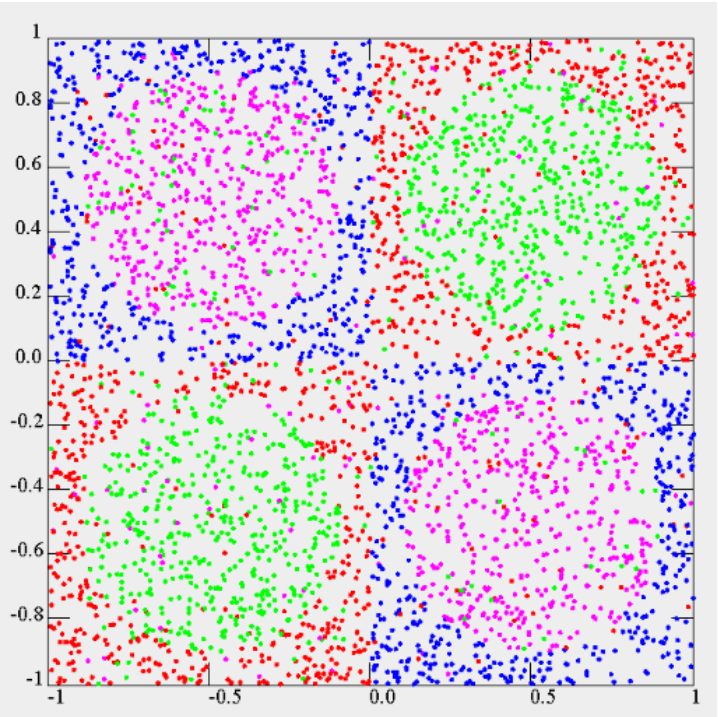


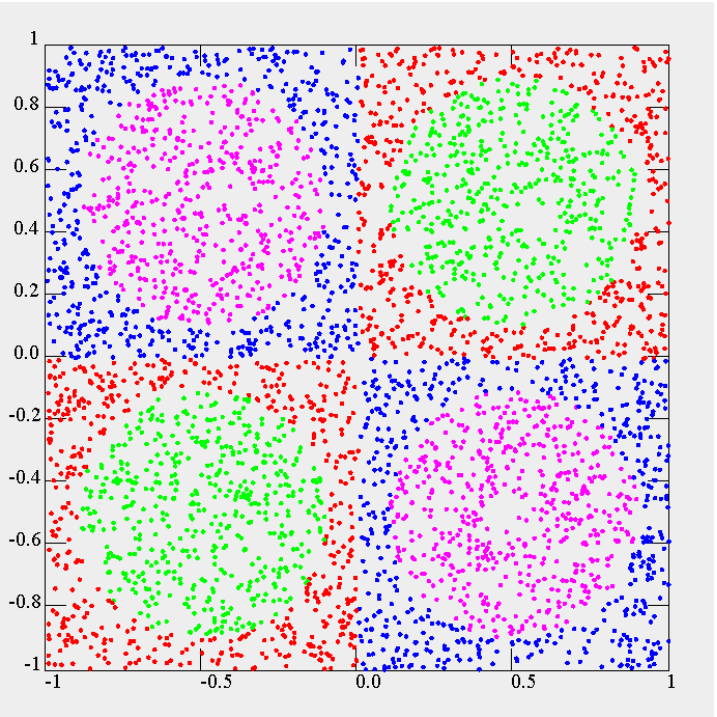


Π1: Σωστά 2594/4000

Π2: Σωστά 2015/4000

Εικόνες από τα σύνολα learning και checking αντίστοιχα:





Άσκηση 2 : Πρόγραμμα ομαδοποίησης (Π3) με Μ ομάδες βασισμένο στον αλγόριθμο k-means.

Σύνολο Σ2:

Δημιουργήσαμε ένα αρχείο με τις συντεταγμένες των τυχαίων σημείων του συνόλου Σ2 και το διατηρήσαμε ώστε όταν τρέχουμε το πρόγραμμα με διαφορετικές κάθε φορά αρχικοποιήσεις να λαμβάνουμε αντικειμενικά αποτελέσματα.

Πρόγραμμα Π3:

Για την υλοποίησή του σχεδιάσαμε 4 classes.

Kmean:

Η κλάση αυτή περιέχει την main μέθοδο του προγράμματός μας και υλοποιεί το αλγόριθμο Kmean.

Αναλυτικότερα, ο κώδικας αρχίζει με την αρχικοποίηση του πλήθους των κέντρων (γραμμή 16). Στη συνέχεια, έχουμε την main μέσα στην οποία έχουμε σε σχόλια τον κώδικα για την δημιουργία του αρχείου data.txt το οποίο περιλαμβάνει τις συντεταγμένες x,y των σημείων ανά γραμμή (γραμμή 24-35). Μετά διαβάζουμε αυτό το αρχείο και δημιουργούμε τα σημεία δίνοντάς τους τιμές στα πεδία x και y (γραμμή 38-52). Στη συνέχεια για λόγους ευκολίας καλούμε την runALL3_13() έξι φορές, δίνοντάς της κάθε φορά διαφορετική αρχική τιμή κέντρων (γραμμή 54-57).

Η μέθοδος runALL3_13() με την σειρά της καλεί την callKmean() είκοσι φορές με τον ίδιο αριθμό κέντρων και στο τέλος τυπώνει το σχεδιάγραμμα της εκτέλεσης που είχε το μικρότερο σφάλμα ομαδοποίησης.(Κάθε plot Point γράφει στον τίτλο του και το πλήθος των κέντρων)

Η μέθοδος callKmean() επιλέγει τυχαία κάποια από τα σημεία που δημιουργήσαμε και τα ορίζει ως αρχικά κέντρα των ομάδων. Στη συνέχεια, καλεί την μέθοδο Kmean() και υπολογίζει το σφάλμα ομαδοποίησης της λύσης της, το οποίο και επιστρέφει στην runALL3_13().

Η μέθοδος kMean() ορίζει τις ομάδες με βάση τα κέντρα και στη συνέχεια υπολογίζει τα νέα κέντρα. Όσο τα νέα κέντρα διαφέρουν από τα παλιά επαναλαμβάνουμε την διαδικασία αυτή.

Τέλος, για την υλοποίηση όλων αυτών έχουμε τη μέθοδο distance(), η οποία υπολογίζει την απόσταση μεταξύ δύο σημείων, τη μέθοδο clustering(), η οποία ορίζει για το κάθε σημείο ότι θα ανήκει στην ομάδα που απέχει λιγότερο από το κέντρο της, τη μέθοδο newClusters(), η οποία ορίζει νέο κέντρο σε κάθε ομάδα και τέλος τη μέθοδο claculateNewCentroid(), η οποία υπολογίζει με βάση τα σημεία μιας ομάδας το κέντρο της.

Cluster:

Η κλάση αυτή ορίζει το αντικείμενο «ομάδα» με πεδία το id που είναι το όνομα της ομάδας, το centroid που κρατάει το κέντρο της και μια λίστα από τα σημεία που περιέχονται στην ομάδα.

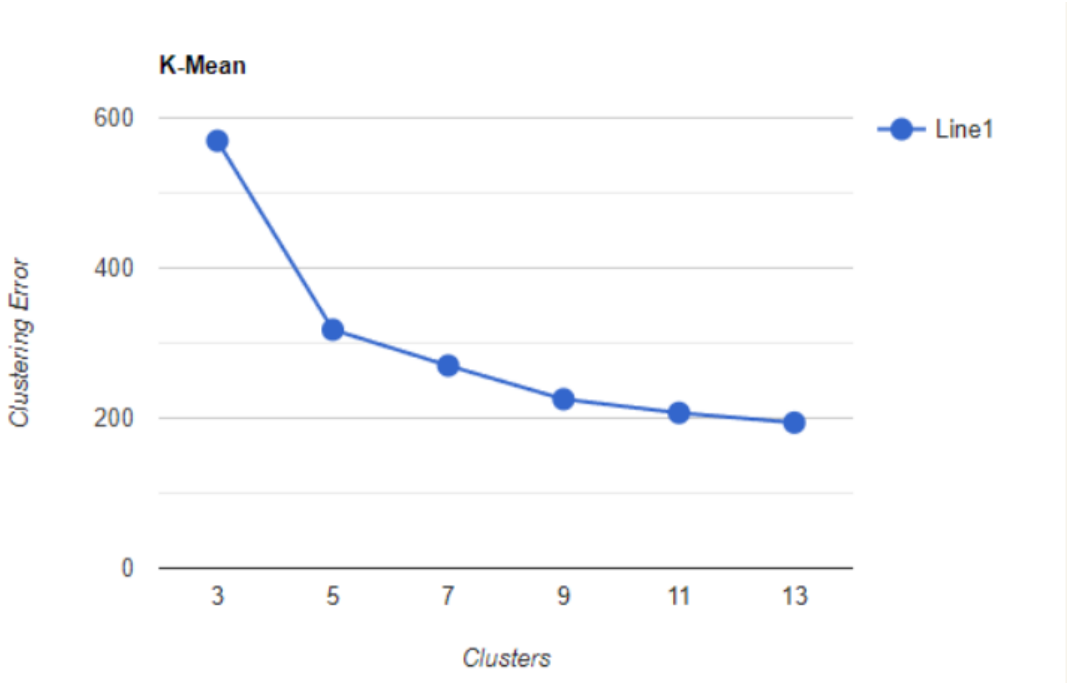
Point:

Η κλάση αυτή ορίζει το αντικείμενο «point» με πεδία x,y που είναι οι συντεταγμένες του, το clusterId που κρατάει το όνομα της ομάδας στην οποία ανήκει και το distanceFromCentroid που κρατάει την απόσταση του σημείου από το κέντρο της ομάδας του.

Graph:

Η κλάση αυτή περιέχει τον κώδικα για τον σχηματισμό των plot που τυπώνουμε.

Διάγραμμα μεταβολής σφάλματος σε σχέση με των αριθμό των ομάδων



<u>Μπορεί να χρησιμοποιηθεί το σφάλμα ομαδοποίησης για να εκτιμήσουμε τον πραγματικό αριθμό ομάδων;</u>

Ναι, μπορεί να χρησιμοποιηθεί το σφάλμα ομαδοποίησης για να εκτιμήσουμε τον πραγματικό αριθμό ομάδων. Όπως βλέπουμε και από το σχήμα, το σφάλμα μειώνεται απότομα καθώς πλησιάζουμε τον πραγματικό αριθμό ομάδων, ενώ όταν τον ξεπεράσουμε ο ρυθμός μείωσης σφάλματος αρχίζει να γίνεται πολύ μικρός.

Αυτό φαίνεται και από το σχήμα μας καθώς η καμπύλη έχει απότομη κλήση μέχρι τις 9 ομάδες, ενώ μετά τις 9 τείνει να οριζοντιωθεί.

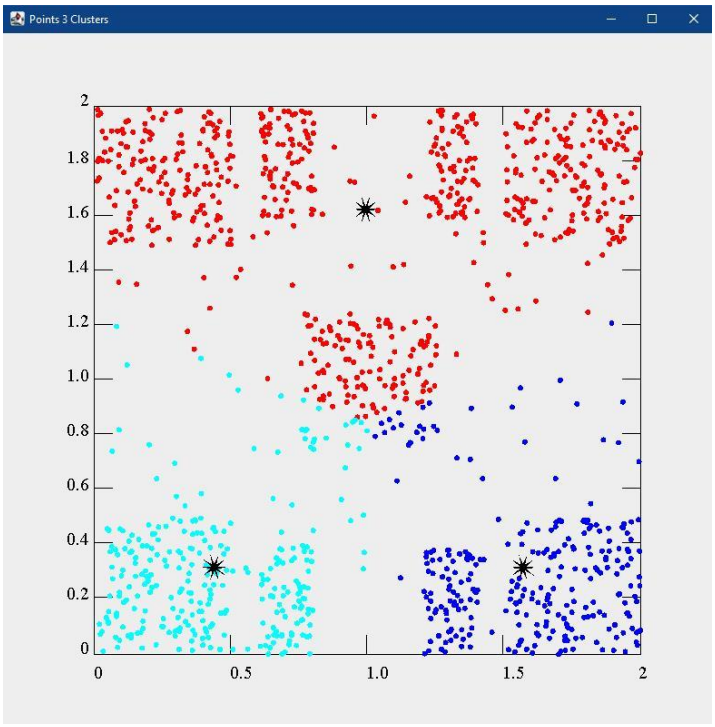
Άρα ο σωστός αριθμός ομάδων είναι 9.

<u>Παράδειγμα μιας εκτέλεσης:</u>

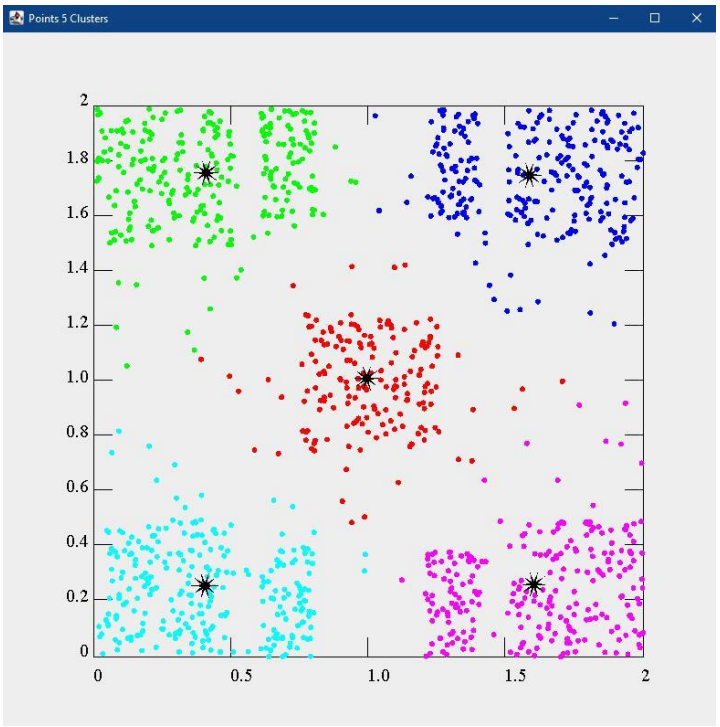
Μία εκτέλεση θα επιστρέψει 6 γραφικές παραστάσεις.

Για κάθε αριθμό ομάδων τυπώνει την καλύτερη από τις 20 φορές.

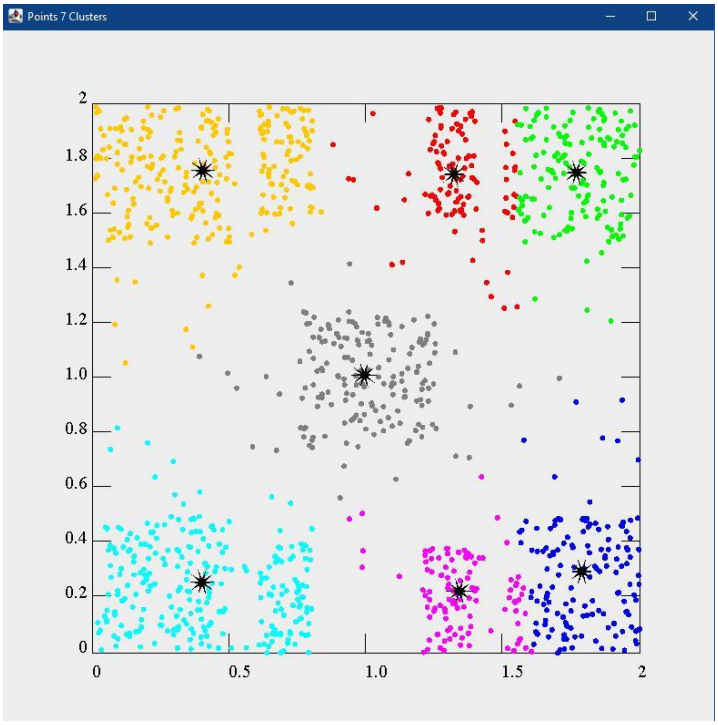
M=3


Points 3 Clusters
2
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
0
0.5
1.0
1.5
2

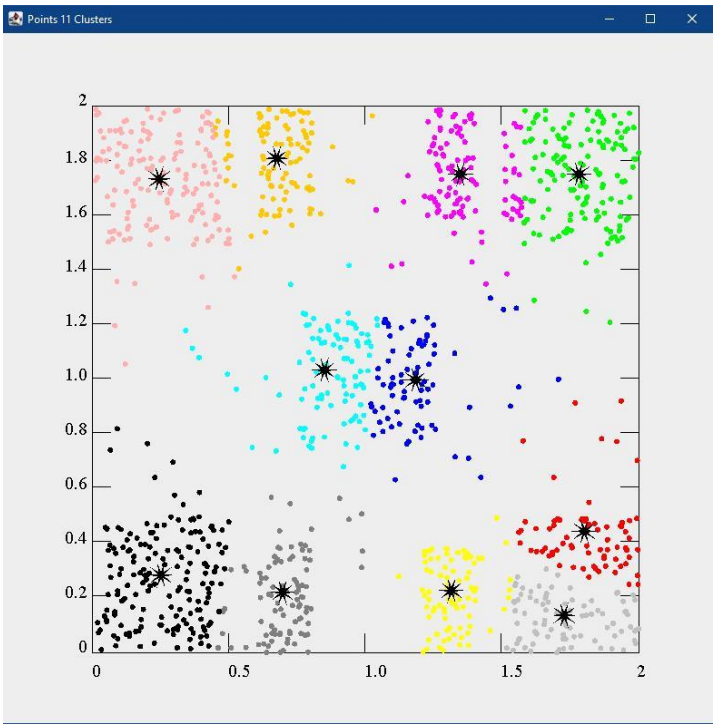
M=5


Points 5 Clusters
2
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
0
0.5
1.0
1.5
2

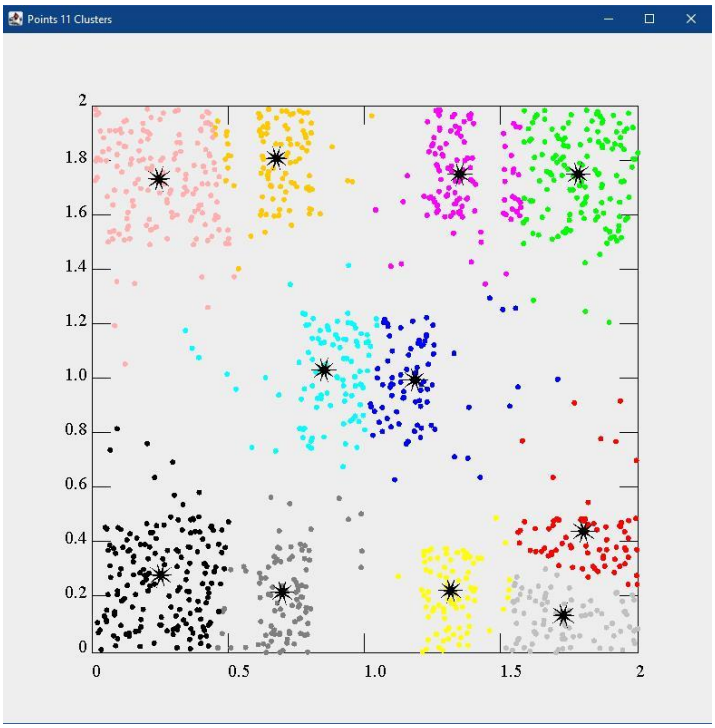
M=7


Points 7 Clusters
2
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
0
0.5
1.0
1.5
2

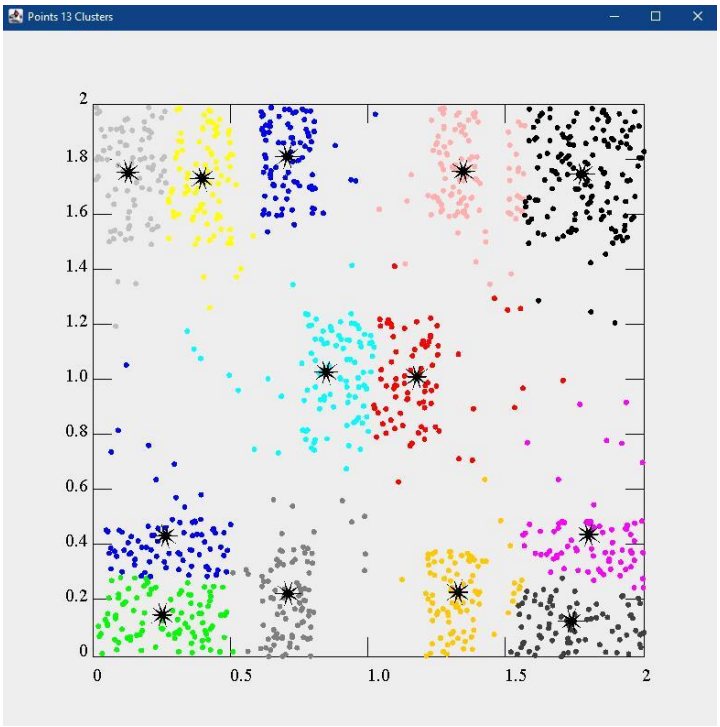
M=9


Points 11 Clusters
2
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
0
0.5
1.0
1.5
2

M=11

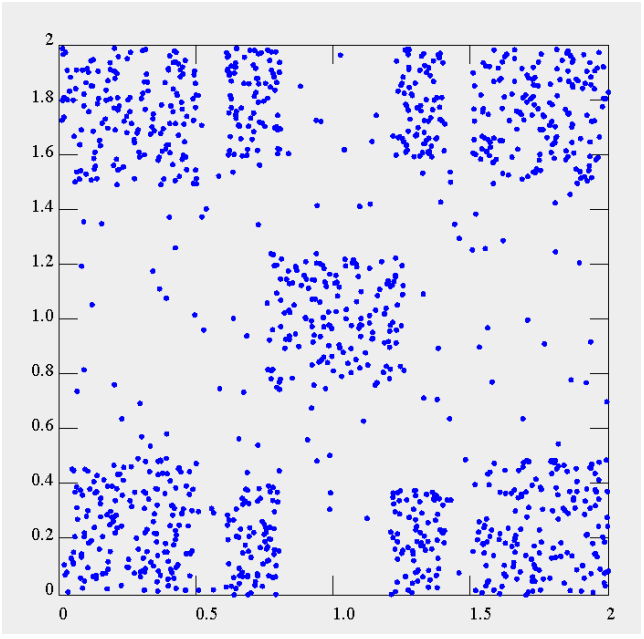


M=13




```
3 Clusters with error: 569.60858636502
5 Clusters with error: 317.81124322748377
7 Clusters with error: 269.9735852638376
9 Clusters with error: 225.1871125242846
11 Clusters with error: 206.90681777621472
13 Clusters with error: 191.11244014308843
```

Εικόνα συνόλου Σ2



ΤΕΛΟΣ