Αναφορά
Εργαστηριακών Ασκήσεων ακ.έτους(2022-23)
Υπολογιστικής Νοημοσύνης

Μέλη Ομάδας
Γκάνος Στέφανος, Α.Μ. 4043

Μπαλής Χρήστος, Α.Μ. 4429

Μπαλής Κωνσταντίνος, Α.Μ. 4117

Άσκηση 1

Για την πρώτη άσκηση που αφορά την κατασκευή ενός πολυεπίπεδου perceptron (MLP) με τρία κρυμμένα επίπεδα δημιουργήσαμε 2 αρχεία. Το ένα αφορά την δημιουργία των τυχαίων παραδειγμάτων (CreateRandomData.c) και το δεύτερο (NeuralNetwork.c) αφορά την υλοποίηση του MLP σύμφωνα με τις οδηγίες της εκφώνησης.

Για την μεταγλώττιση του προγράμματος ο χρήστης πρέπει να μεταβεί από το τερματικό στην τοποθεσία που βρίσκονται τα αρχεία και στην συνέχεια να εκτελέσει τις ακόλουθες εντολές :

gcc CreateRandomData.c -lm

./a.out

gcc NeuralNetwork.c -lm

./a.out

Μέσα στον φάκελο προσθέσαμε και το αρχείο που χρησιμοποιήσαμε για να κάνουμε plot τα δεδομένα με την καλύτερη γενικευτική ικανότητα "PlotNeuralNetworkData ".

Εκτελώντας τις παραπάνω εντολές θα δημιουργηθεί στην τοποθεσία που βρίσκονται τα αρχεία ένα αρχείο "data.txt" με τα τυχαία παραδείγματα και ένα ακόμη αρχείο "results.txt" που περιέχει στην πρώτη γραμμή των αριθμό των σωστών προβλέψεων που κάνει το MLP, στην συνέχεια όλα τα σωστά παραδείγματα που προβλέπει το MLP με την κατηγορία τους και τέλος τα παραδείγματα που προβλέπει λανθασμένα το MLP που θα βοηθήσουν στο να τα τυπώσουμε στην συνέχεια.

Ο χρήστης μπορεί να καθορίσει τιμές όπως ο αριθμός των νευρώνων σε καθένα από τα 3 κρυμμένα επίπεδα (H1, H2, H3), το είδος συνάρτησης ενεργοποίησης (λογιστική, tanh, relu), τον ρυθμό μάθησης, τον αριθμό των mini-batches και το κατώφλι τερματισμού και γι΄αυτό στην αρχή του NeuralNetwork.c έχουμε τις μεταβλητές που μπορεί να αλλάξει ο χρήστης με το μήνυμα
</* User can change those below */> όπως φαίνεται παρακάτω :

```
/*  User can change those below */
#define H1 12                           /*  Number of neuron in first hidden layer  */
#define H2 8                            /*  Number of neuron in second hidden layer */
#define H3 10                           /*  Number of neuron in third hidden layer  */
#define ACTIVATION_FUNC 0               /*  Use 0 for logistic, 1 for tanh and 2 for relu   */
#define n 0.005                         /*  Learning rate   */
#define B 40                            /*  Number of min-batches   */
#define TERMINATION_THRESHOLD 0.0001
```

α). Μεταβολή γενικευτικής ικανότητας του δικτύου για διάφορους συνδυασμούς τιμών για τα H1, H2, H3.

(H1, H2,H3) (Γενικευτική Ικανότητα)	(2, 2, 2)	(6, 6, 6)	(10, 10, 10)
Γενικευτική Ικανότητα	0.662250	0.841750	0.958000

(9, 4, 5)	(3, 8, 2)	(4, 3, 11)	(7, 10, 2)	(9, 2, 8)	(3, 8, 12)
0.792250	0.693750	0.780250	0.895750	0.840000	0.771250

β). Μεταβολή γενικευτικής ικανότητας του δικτύου ανάλογα με την συνάρτηση ενεργοποίησης. (Τα πειράματα έγιναν για τιμές ρυθμού μάθησης (0.0005), κατώφλι (0.0001) και mini-batches (n/100 = 40)).

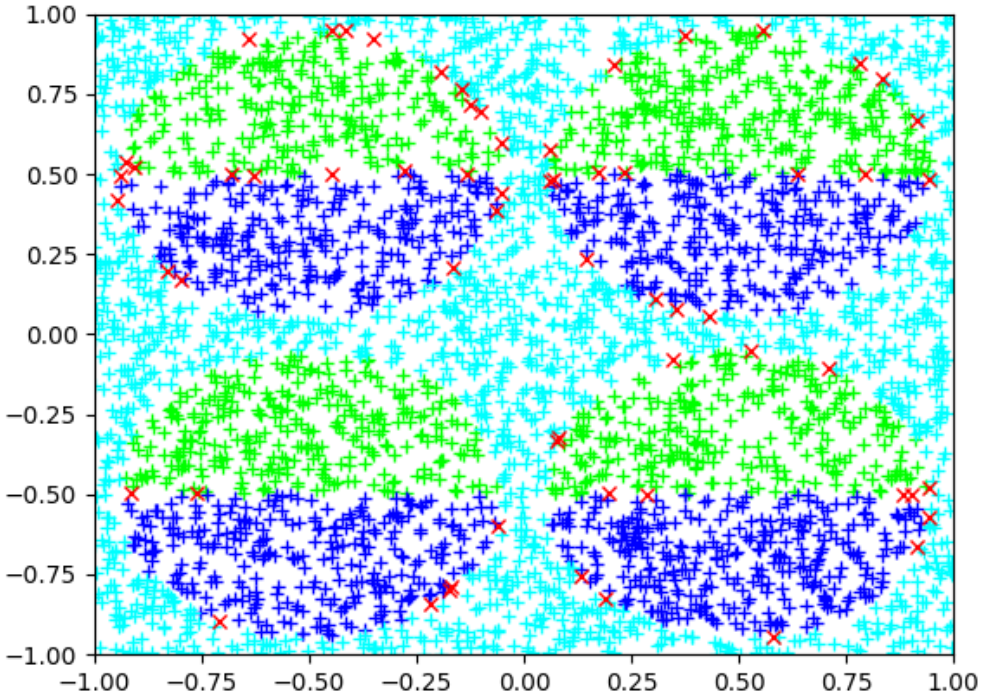
Συνάρτηση ενεργοποίησης (Γενικευτική Ικανότητα)	Λογιστική	Υπερβολική εφαπτομένη	Relu
Γενικευτική Ικανότητα	0.984250	0.975250	0.924250

(Να σημειώσουμε ότι για τις παραπάνω τιμές ρυθμού μάθησης, κατωφλίου και αριθμού mini-batches η λογιστική χρειάστηκε υπερβολικά μεγάλο αριθμό εποχών (πάνω από 20000) και για αυτό το λόγο εκτελέσαμε ακόμη ένα πείραμα για ρυθμό μάθησης 0.005 που πέτυχε εξίσου ικανοποιητική γενικευτική ικανότητα 0.970250 με πολύ λιγότερες εποχές. Για αντίστοιχο αριθμό ρυθμού μάθησης η υπερβολική εφαπτομένη και η relu τερμάτιζαν πολύ γρήγορα και παρατηρήσαμε αυξομειώσεις στο σφάλμα γι' αυτό και αναγκαστήκαμε να μειώσουμε τον ρυθμό μάθησης).

γ). Μεταβολή γενικευτικής ικανότητας του δικτύου σε σχέση με τον αριθμό mini-batches. (Για ρυθμό μάθησης 0.005 (για λογιστική) και 0.0005 (για υπερβολική εφαπτομένη και relu) (εξηγήσαμε στην παραπάνω σημείωση το λόγο) και κατώφλι 0.0001)

Συνάρτηση ενεργοποίησης Mini-Batches	Λογιστική	Υπερβολική εφαπτομένη	Relu
B = n/10 (400)	0.760750	0.926250	0.961250
B = n/100 (40)	0.947000	0.930500	0.981750

Τέλος παρακάτω μπορείτε να δείτε τυπωμένο το παράδειγμα με την καλύτερη γενικευτική ικανότητα που βρήκαμε και είναι για λογιστική συνάρτηση ενεργοποίησης, ρυθμό μάθησης (0.0005), κατώφλι (0.0001) και αριθμό mini-batches (n/100 = 40) και έχει γενικευτική ικανότητα 0.984250.



Άσκηση 2

Για την δεύτερη άσκηση που μας ζητείται να φτιάξουμε ένα πρόγραμμα ομαδοποίησης βασισμένο στον K-Means δημιουργήσαμε δύο αρχεία τα Main.java όπου δημιουργούμε τα τυχαία σημεία όπως αυτά ζητήθηκαν από την εκφώνηση, εκτελούμε για καθένα από τα M κέντρα (M = 3,6,9,12) 15 φορές τον αλγόριθμο k-means και δημιουργούμε τα αρχεία στα οποία θα καταγράψουμε τα αποτελέσματα του αλγορίθμου για την λύση με το μικρότερο σφάλμα για κάθε τιμή του M ώστε στη συνέχεια να τυπώσουμε. Ακόμη δημιουργήσαμε ένα αρχείο KMeans.java στο οποίο υλοποιούμε τον αλγόριθμο k-means όπου αρχικοποιούμε τα κέντρα σε τυχαίες θέσεις και στη συνέχεια κατατάσσουμε καθένα από τα σημεία στο πλησιέστερο από τα κέντρα.

Για την μεταγλώττιση του προγράμματος ο χρήστης πρέπει να μεταβεί από το τερματικό στην τοποθεσία που βρίσκονται τα αρχεία και στην συνέχεια να εκτελέσει τις ακόλουθες εντολές :

javac KMeans.java

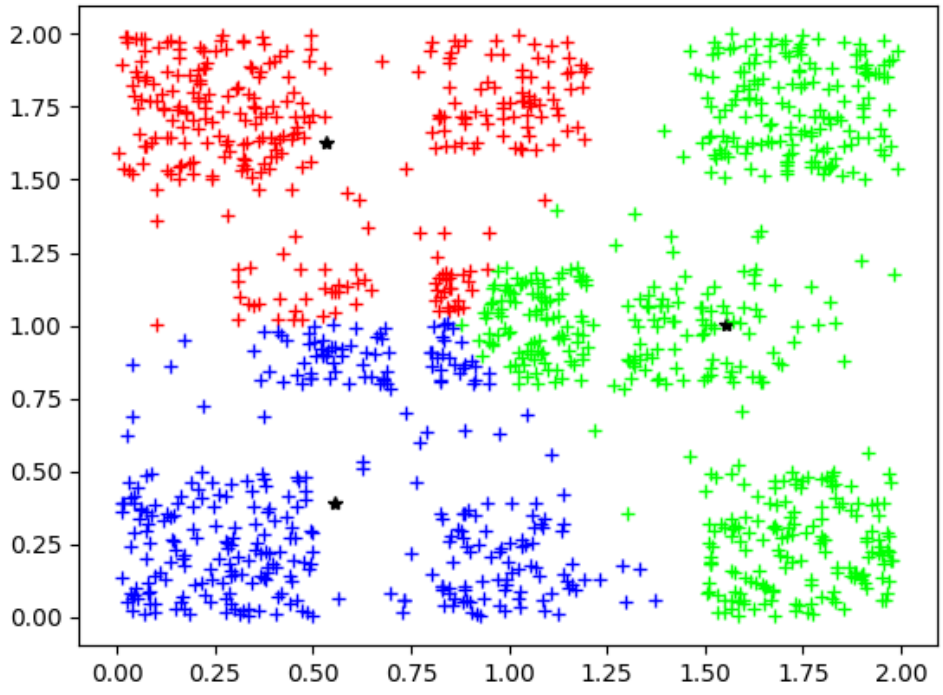
javac Main.java

java Main

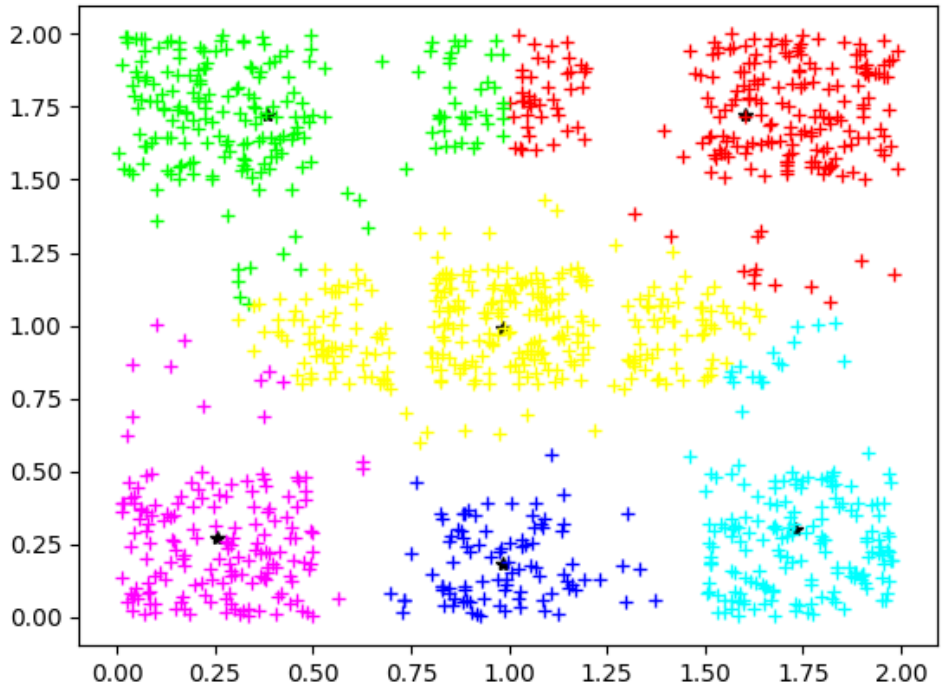
Εκτελώντας τις παραπάνω εντολές θα δημιουργηθεί στην τοποθεσία που εκτελέστηκαν οι παραπάνω εντολές τέσσερα αρχεία “3_clusters.txt”, “6_clusters.txt”, “9_clusters.txt”, “12_clusters.txt”. Στην πρώτη γραμμή κάθε αρχείου υπάρχει το σφάλμα ομαδοποίησης, στη συνέχεια οι θέσεις των σημείων με τον cluster στον οποίο ανήκουν και τέλος τις συντεταγμένες των κέντρων.

Για την δημιουργία των plots δημιουργήσαμε ένα αρχείο “PlotKMeansData.py” όπου διαβάζει τα αρχεία που δημιουργήσαμε και τυπώνει τα αποτελέσματα το οποίο έχουμε προσθέσει στον φάκελο για την δεύτερη άσκηση. Για να είναι ευδιάκριτες οι ομάδες (clusters) αναπαριστούμε κάθε μια από αυτές με διαφορετικό χρώμα και για τα κέντρα επιλέξαμε ένα διαφορετικό σύμβολο «*» αντί για «+» που χρησιμοποιήσαμε για τα μέλη των ομάδων.

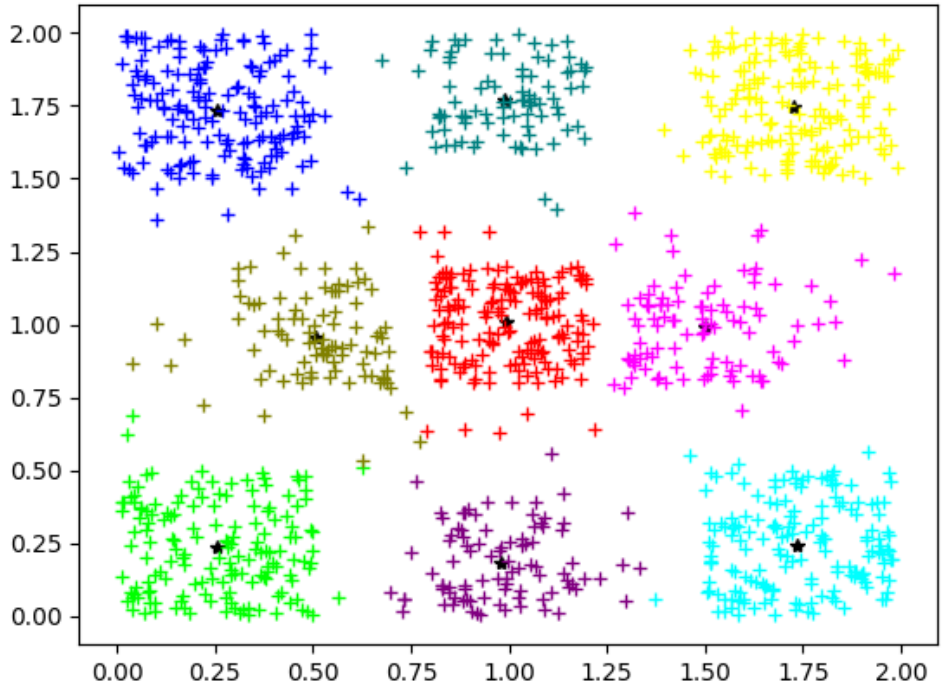
Για M = 3 ομάδες (clusters) :



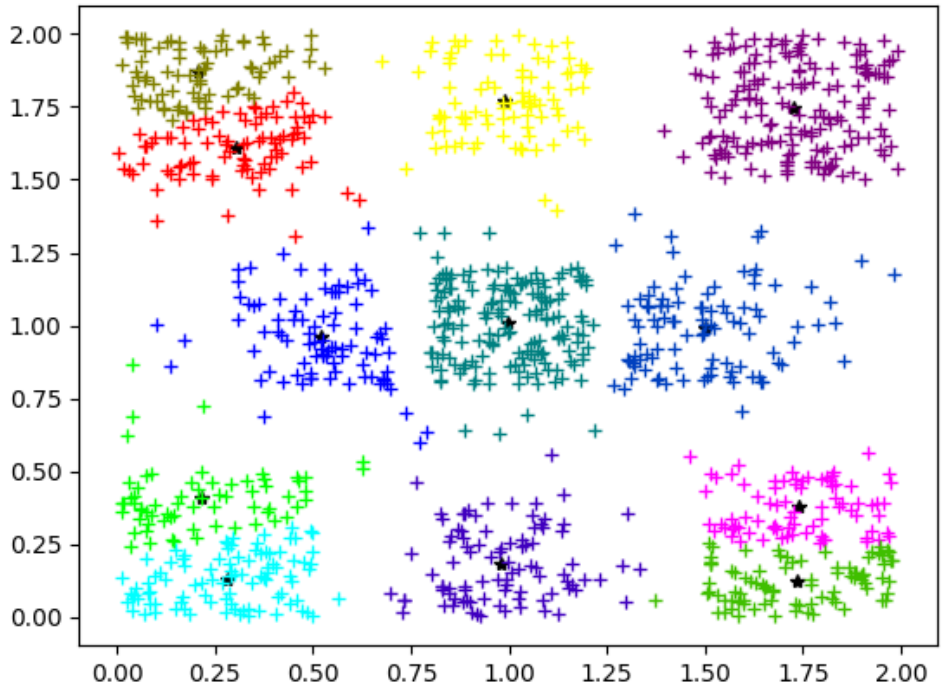
Για M = 6 ομάδες (clusters) :



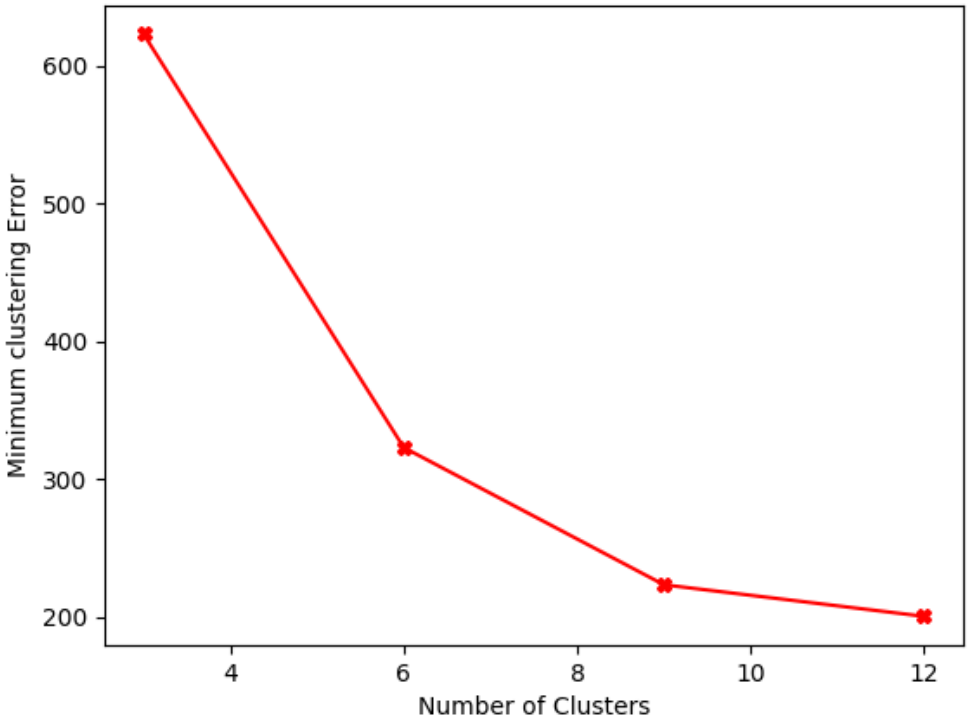
Για M = 9 ομάδες (clusters) :



Για M = 12 ομάδες (clusters) :



Τέλος δημιουργήσαμε ένα διάγραμμα που δείχνει πως μεταβάλλεται το σφάλμα ομαδοποίησης με τον αριθμό των ομάδων.



Για να εκτιμήσουμε τον πραγματικό αριθμό ομάδων παρατηρούμε το trade-off μεταξύ του αριθμού των cluster και του σφάλματος καθώς επιθυμούμε να έχουμε όσο το δυνατόν μικρότερο σφάλμα έχοντας όσο το δυνατόν λιγότερους clusters. Στο γράφημα παρατηρούμε ότι οι μεταβολές των σφαλμάτων από 3 clusters σε 6 και από 6 σε 9 είναι αρκετά απότομες με το σφάλμα να μειώνεται σημαντικά όμως από 9 σε 12 παρατηρούμε μικρή μεταβολή οπότε μπορούμε να εκτιμήσουμε ότι ο πραγματικός αριθμός των ομάδων είναι 9.

(Σε αυτό το σημείο να σημειώσουμε πως κατά την εκτέλεση της Main για την δεύτερη άσκηση παρατηρήσαμε ότι υπάρχουν περιπτώσεις που **δεν** δημιουργούνται όλα τα αρχεία. Δεν μπορέσαμε να καταλάβουμε το λόγο για τον οποίο συμβαίνει αυτό (αν και συνήθως μετά από λίγες εκτελέσεις θα δημιουργήσει και τα τέσσερα επιθυμητά αρχεία ενώ έχουμε βάλει και να τυπώνει και αντίστοιχο μήνυμα στον χρήστη για το ποια αρχεία έχει δημιουργήσει) και γι’ αυτό μέσα στον φάκελο δημιουργήσαμε ένα υποφάκελο όπου έχουμε κρατήσει τα αρχεία από μια επιτυχημένη εκτέλεση του προγράμματος).