ΥΠΟΛΟΓΙΣΤΙΚΗ ΝΟΗΜΟΣΥΝΗ ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

Ιωάννης Κούσιας 4611

Βασίλης Βαλεράς 4031

Αλίκο Μούσκα 4427

**Άσκηση 1η**

**2.1 Το πρόβλημα:**

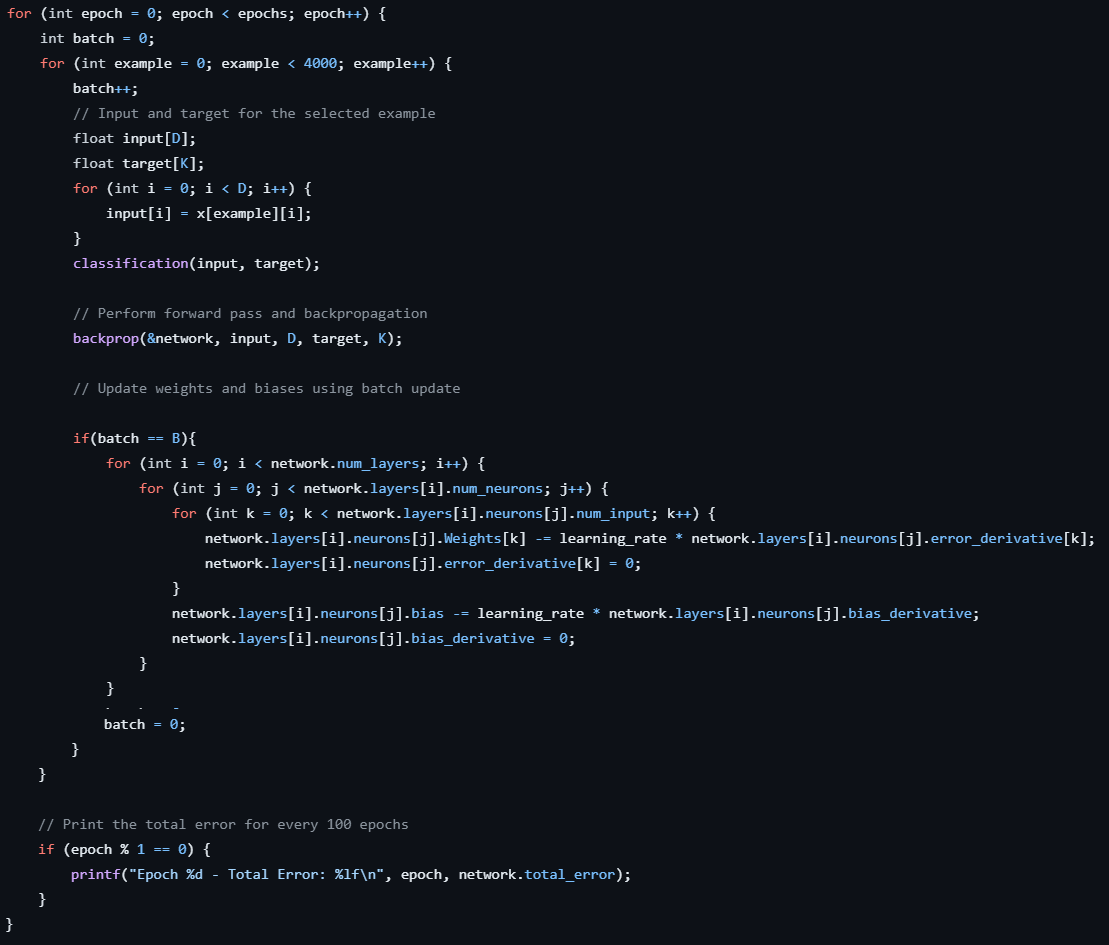
Στη συγκεκριμένη άσκηση ζητείται να υλοποιηθεί ένα πρόγραμμα ταξινόμησης με την χρήση πολυεπίπεδου perceptron(MLP) νευρωνικου δικτύου με τρία κρυμμένα επίπεδα. Στο νευρωνικο δίκτυο θα δίνεται ενα σημειο στο καρτεσιανο επιπεδο και αυτο θα πρεπει να προβλέπει σε πια ομαδα βρίσκεται με βαση τους κανονες που μας έχουν δωθεί.

**2.2 Υλοποίηση (Κώδικας):**

**Compile Αρχικα για να κανουμε compile το προγραμμα:**

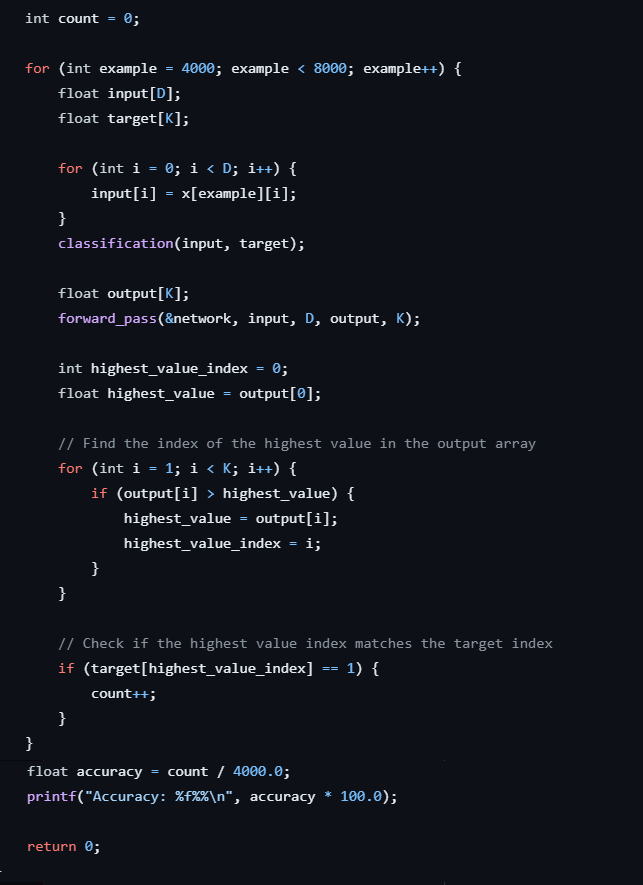
**gcc main.c -lm και gcc -o dataset\_maker dataset\_maker.c**

Ξεκινώντας απο την main αφου φορτώσουμε τις πληροφοριες του file βλέπουμε την παρακάτω for.

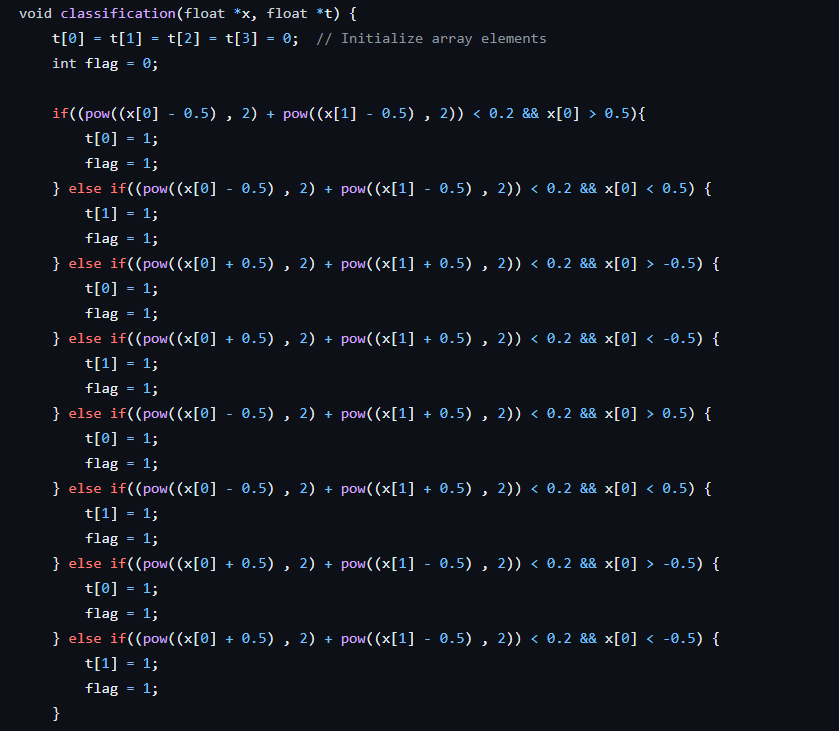


Σε κάθε εποχη παίρνουμε τα πρωτα 4000 examples και χρησιμοποιώντας την classification (χρησιμοποιείται για να ορίσει τον στόχο) και την backprop (υπολογίζει τις παραγώγους του σφάλματος ως προς οποιαδήποτε παράμετρο) οπου θα δείξουμε παρακάτω για να κανουμε το update στα βάρη.Στο τέλος καθε εποχης εκτυπωνεται το total error.

Μετα την εκπαίδευση χρησιμοποιουμε τα υπολοιπα 4000 example για να υπολογισουμε την γενικευτικη ικανοτητα του δικτυου.

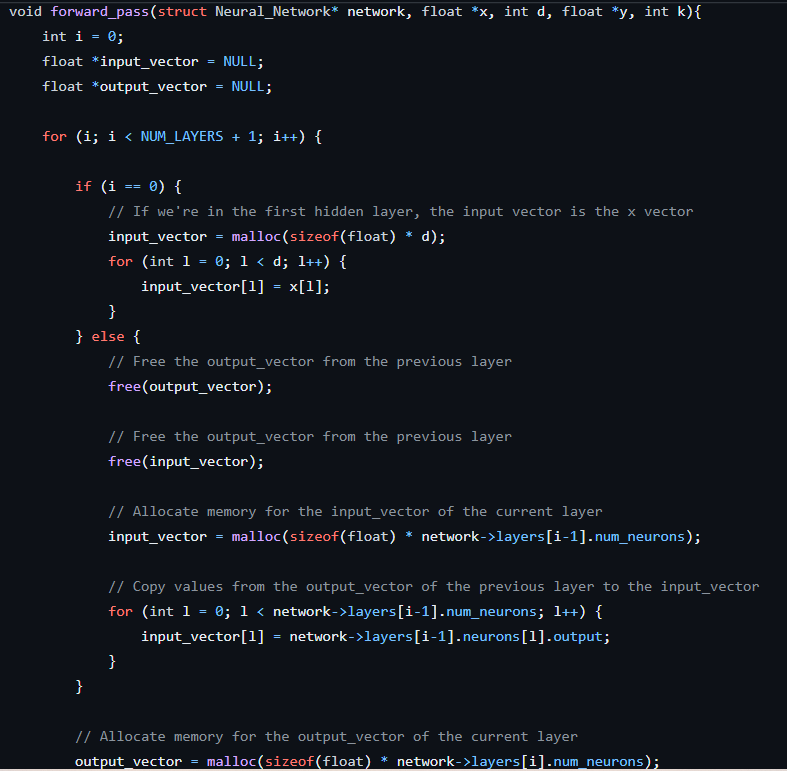


**Classification**

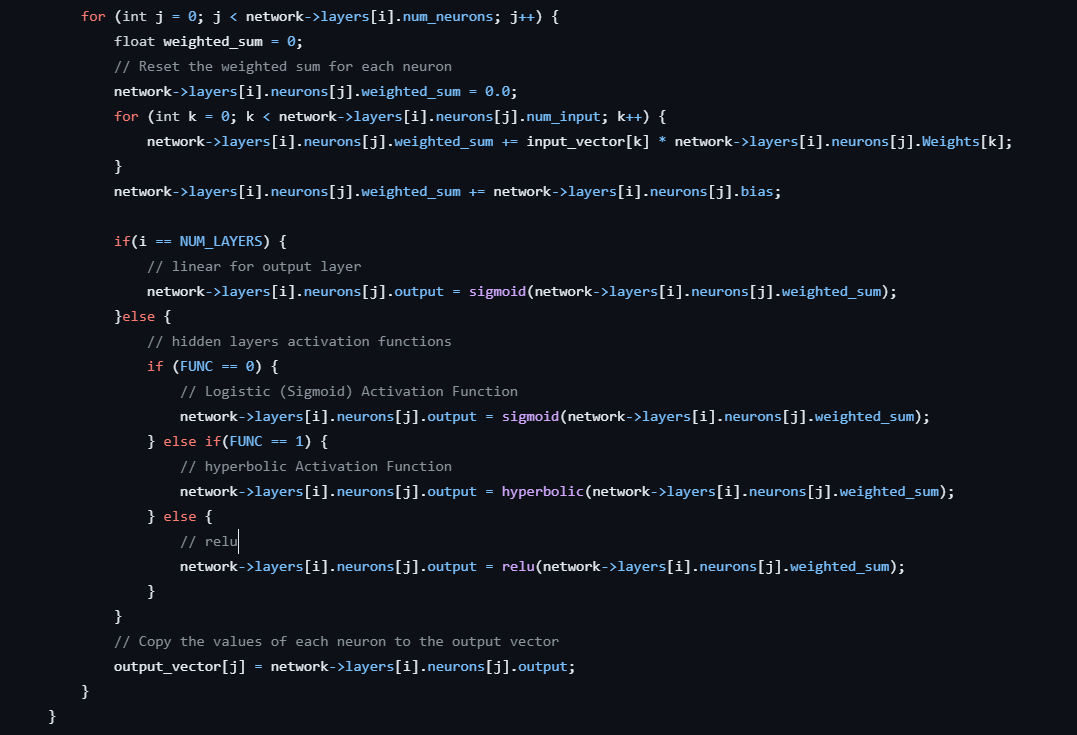


Η συνάρτηση classification εκχωρεί στον πίνακα t τους στόχους (targets) για ένα δεδομένο παράδειγμα βάσει των εισόδων x. Μετα απο τον έλενγχο το t θα εχει την τιμη 1 στο index της ομαδας οπου ανηκει το x.

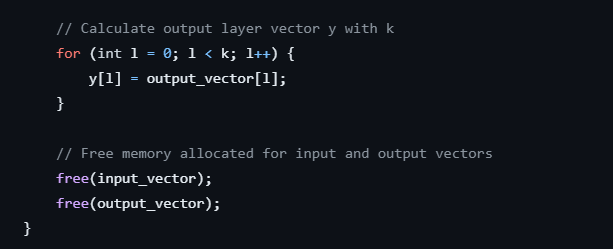
**forward\_pass**

****

Αρχικα χρησιμοποιηουμε τον πινακα input\_vector για layer ως εισοδο. Αν βρισκομαστε στο πρωτο hidden layer παιρνουμε τιμες για το input\_vector απο το x. Αλλιως παιρνουμε τις τιμες απο το output του προηγουμενου layer.

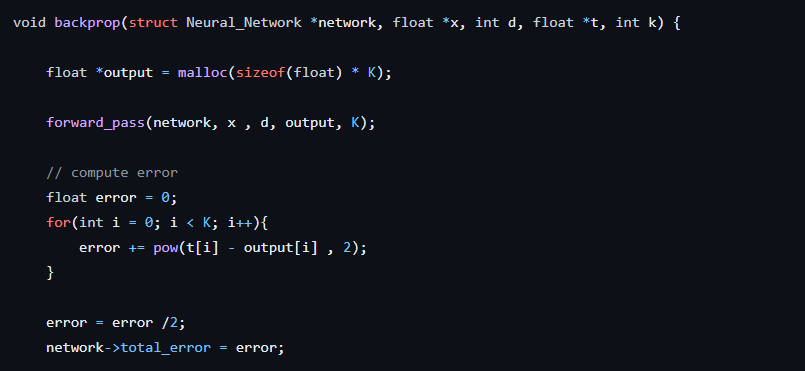


Η πρώτη for σκοπεύει να διατρέξει ολους του νευρωνες στο συγκεκριμένο layer. Η δευτερη υπολογιζει το αθροισμα wi\*xi και του bias. Στην συνέχεια με την if else επιλέγεται η συναρτηση ενεργοποιησης συμφωνα με την παραμετρο που έχουμε δώσει.



Τελος το output περνιέται στο πινακα y.

**Backprop**



Καλεί τη συνάρτηση forward\_pass για να υπολογίσει την έξοδο του δικτύου για την εισαγόμενη είσοδο x και αποθηκεύει τα αποτελέσματα στον πίνακα output. Υπολογίζει το σφάλμα χρησιμοποιώντας το τετραγωνικό σφάλμα μεταξύ του αποτελέσματος output και του πραγματικού αποτελέσματος t.



Ξεκινώντας από το τελευταίο επίπεδο και κινούμενοι προς τα πίσω, υπολογίζει τις παραγώγους του σφάλματος ως προς ολες τις παράμετρους. Για τα κρυφά επίπεδα, υπολογίζει το σφάλμα χρησιμοποιώντας το σφάλμα του επόμενου επιπέδου.

Ετσι λοιπον με αυτη την συναρτηση μπορουμε στην main οπως δειξαμε παραπανω να υλοποιησουμε τον αλγοριθμο gradient descent.

**2.3 Αποτελέσματα**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **#** | **H1** | **H2** | **H3** | **Activation function** | **Β** | **γενικευτική ικανότητα** |
| 1 | 2 | 2 | 2 | Logistic | 1 | 31% |
| 2 | 2 | 2 | 2 | hyberbolic | 1 | 66.7% |
| 3 | 2 | 2 | 2 | relu | 1 | 46.3% |
| 4 | 2 | 2 | 2 | relu | 400 | 52.3% |
| 5 | 2 | 2 | 2 | relu | 40 | 47.2% |
| 6 | 5 | 5 | 5 | logistic | 40 | 32.15% |
| 7 | 5 | 5 | 5 | hyberbolic | 40 | 75.25% |
| 8 | 5 | 5 | 5 | relu | 40 | 80.92% |
| 9 | 5 | 5 | 5 | relu | 400 | 71.87% |
| 10 | 5 | 5 | 5 | relu | 1 | 80.2% |
| 11 | 7 | 10 | 7 | relu | 1 | 82.65% |
| 12 | 7 | 10 | 7 | hyberbolic | 1 | 90.62% |
| 13 | 7 | 10 | 7 | logistic | 1 | 38.77% |
| 14 | 7 | 10 | 7 | hyberbolic | 40 | 81.87% |
| 15 | 7 | 10 | 7 | hyberbolic | 400 | 83.52% |
| 16 | 7 | 10 | 7 | hyberbolic | 400 | 83.57% |
| 17 | 12 | 10 | 12 | hyberbolic | 1 | 92.97% |
| 18 | 12 | 10 | 12 | relu | 1 | 86.47% |
| 19 | 12 | 10 | 12 | logistic | 1 | 39.64% |
| 20 | 12 | 10 | 12 | hyberbolic | 40 | 94.59% |
| 21 | 12 | 10 | 12 | hyberbolic | 400 | 97.72% |
| 22 | 12 | 15 | 13 | hyberbolic | 1 | 97.32% |
| 23 | 12 | 15 | 13 | relu | 1 | 94.07% |
| 24 | 12 | 15 | 13 | hyberbolic | 40 | 97.25% |
| 25 | 12 | 15 | 13 | relu | 40 | 94.67% |
| 26 | 12 | 15 | 13 | hyberbolic | 400 | 94.27% |
| 27 | 12 | 15 | 13 | relu | 400 | 92.32% |
| 28 | 15 | 15 | 15 | hyberbolic | 1 | 96.77% |
| 29 | 15 | 15 | 15 | relu | 1 | 94.57% |
| 30 | 15 | 15 | 15 | logistic | 1 | 38.07% |
| 31 | 15 | 15 | 15 | hyberbolic | 40 | 97.72% |
| 32 | 15 | 15 | 15 | relu | 40 | 95.12% |
| 33 | 15 | 15 | 15 | hyberbolic | 400 | 95.89% |
| 34 | 15 | 15 | 15 | relu | 400 | 94.16% |
| 35 | 20 | 20 | 20 | hyberbolic | 1 | 97.67% |
| 36 | 20 | 20 | 20 | relu | 1 | 95.89% |
| 37 | 30 | 30 | 30 | hyberbolic | 1 | 97.72% |
| 38 | 30 | 30 | 30 | relu | 1 | 96.70% |

Για κάθε συνδιασμο τρέξαμε το πρόγραμμα 3 φορες και κρατήσαμε το καλύτερο αποτέλεσμα.

**2.4 Συμπεράσματα:**

Βλέπουμε η γενικευτικη ικανοτητα του δικτυου αυξανεται σημαντικα απο τα δικτυα h1=2, h2=2, h3=2 στα h1=5, h2=5, h3=5 με την #8 και #10 να φτανουν πανω απο 80% γενικευτικη ικανοτητα.

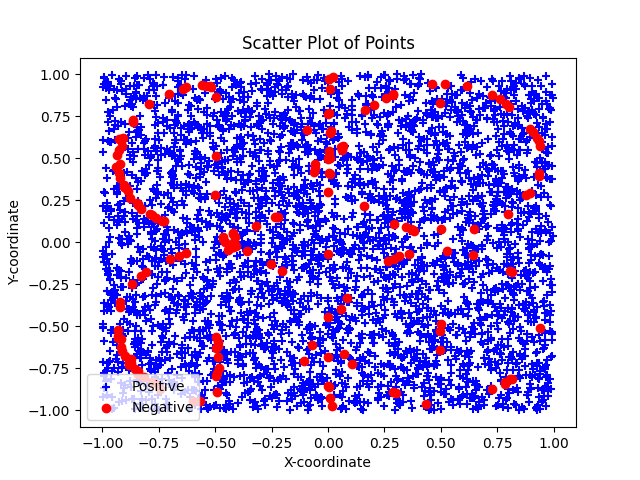
Στα h1=7, h2=10, h3=7 βλέπουμε πως ολοι οι συνδιασμοι φτανουν ικανοποιητικη συμπεριφερα εκτος απο το δικτυο με συναρτηση ενεργοποιησης logistic (#13). Σε αυτα τα δικτυα φτασαμε τo 90% γενικευτικη ικανοτητα (#12).

Στο επομενο βημα δοκιμασαμε h1=12, h2=10, h3=12. Εδω βλέπουμε σε ολες τις περιπτωσεις πολλη καλη αποδοση εκτος απο την logistic (#19) για αυτο και στις επομενες δοκιμασες επικεντρωνόμαστε στην hyberbolic και relu. Εδω να πουμε πως η #21 ( h1=12, h2=10, h3=12, B=400) εφτασε μια απο τις καλυτερες αποδοσεις με λιγουτερους νευρωνες απο τις επομενες δοκιμες.

Οι δοκιμες με h1=12, h2=15, h3=13 και h1=15, h2=15, h3=15, h1=20, h2=20, h3=20 και h1=30, h2=30, h3=30 (πλην της logistic) εδειξαν ολες εξαιρετική συμπεριφορα φτάνοντας ολες πάνω απο 90% γενικευτικη ικανοτητα.

Καθως τυχαίνει μερικες αρχιτεκτονικες να εχουν την ίδια γενικευτικη ικανοτητα θα επιλέξουμε αυτη με τους λιγοτερους νευρωνες καθως ειναι πιο αποδοτική και δεν ειναι τοσο ευαλωτη σε φενομενα overtraining. Ετσι λοιπον ως καλυτερη περιπτωση κραταμε την #21 με h1=12, h2=10, h3=12, B=400 και συναρτηση ενεργοποιησης την hyberbolic.

Γραφημα ταξινομημενων σημειων απο το δικτυο #21



γενικευτική ικανότητα

Βλέπουμε πως το δίκτυο λειτουργεί σωστα στο μεγαλυτερο μερος. Παρολα αυτά φαινεται η δυσκολια του δικτου να ταξινομήσει σημεια που ειναι κοντα στα “συνορα” ενος ή παραπάνω ομαδων.

**Άσκηση 2η (K-Means)**

**2.1 Το πρόβλημα:**

Στη συγκεκριμένη άσκηση ζητείται να υλοποιηθεί ένα πρόγραμμα ομαδοποίησης . Πιο συγκεκριμένα απαιτείται το πρόγραμμα να βασίζεται στον αλγόριθμο k-means με Μ ομάδες . Το M θα ορίζεται με την εντολή #define έτσι ώστε να υπάρχει η δυνατότητα εκτέλεσης και μελέτης των αποτελεσμάτων για διαφορετικό αριθμό ομάδων. Η αρχική θέση των κέντρων γίνεται τυχαία επιλέγοντας κάποιο από τα παραδείγματα του dataset που δημιουργήσαμε. Στη συνέχεια θέλουμε να εκτελείται ο αλγόριθμος και να υπολογίζονται τα τελικά κέντρα καθώς και το σφάλμα ομαδοποίησης .

**2.2 Υλοποίηση (Κώδικας):**

**Compile: Αρχικα για να κανουμε compile το προγραμμα: gcc -ο dataset-maker dataset-maker.c και gcc kMeans.c -lm**

**dataset-maker.c**A computer screen shot of a program code

Description automatically generated

Αρχικά έχουμε τον κώδικα για την παραγωγή του dataset. Ο παραπάνω κώδικας είναι απλός , ανοίγει ένα αρχείο για γράψιμο και καλεί την συνάρτηση create\_values() για την παραγωγή τυχαίων τιμών. Η συνάρτηση παίρνει σαν όρισμα τα μέγιστα και τα ελάχιστα x,y καθώς και τον file pointer . Σκοπός της συνάρτησης είναι να δίνει τυχαίες τιμές στο εύρος τιμών που δόθηκε και να τις γράφει στο αρχείο . Ο κώδικας αυτός δοκιμάστηκε κάποιες φορές και στη συνέχεια κρατήσαμε ένα σταθερό dataset για να τρέξουμε τον αλγόριθμο , έτσι ώστε να μπορέσουμε να μελετήσουμε τα αποτελέσματα.

**kMeans.h**

A screen shot of a computer code

Description automatically generated

Ο παραπάνω κώδικας είναι βοηθητικός . Ουσιαστικά θέλαμε να κρατήσουμε σε ένα αρχείο .h τις εντολές #define και #include σε ένα ξεχωριστό αρχείο. Επίσης περιέχει την δομή center\_s η οποία μας παρέχει πληροφορίες για το κέντρο, την ομάδα του και το σύνολο των στοιχείων που έχουν τοποθετηθεί σε αυτή . Τέλος περιέχει τις νέες συντεταγμένες του κέντρου της ομάδας για να γίνει σύγκριση με το αποτέλεσμα της προηγούμενης εποχής . Στο παραπάνω αρχείο τοποθετήθηκαν επίσης τα πρωτότυπα των συναρτήσεων που απαιτεί η γλώσσα προγραμματισμού C για να γίνει η κλήση τους από οποιοδήποτε σημείο του κώδικα.

**kMeans.c**

A screen shot of a computer program

Description automatically generated

Η συνάρτηση load\_data() δεσμεύει χώρο στη μνήμη και φορτώνει τα στοιχεία από το dataset, στη συνέχεια τα επιστρέφει στην συνάρτηση από την οποία κλήθηκε.

A computer code with text

Description automatically generated

Στην συνάρτηση allocate\_numOf\_teams() δεσμεύουμε χώρο για να κρατήσουμε τα σφάλματα κάθε ομάδας και επιστρέφουμε τον πίνακα. Η παραπάνω ενέργεια χρησιμεύει στον υπολογισμό του σφάλματος ομαδοποίησης όπως θα δούμε παρακάτω που θα αναλύσουμε τον κώδικα του κυρίως αλγορίθμου.

A computer screen shot of text

Description automatically generated

Η συνάρτηση create\_random\_values() επιστρέφει μία τυχαία τιμή μέσα στο εύρος τιμών που δίνουμε σαν είσοδο . Η centers\_init() χρησιμοποιεί την παραπάνω συνάρτηση για να διαλέξει την σειρά του dataset στην οποία θα αρχικοποιήσει το κάθε κέντρο . Για αυτόν τον λόγο δίνουμε ως είσοδο το εύρος τιμών από 0 έως 1199 (1200 στοιχεία). Οπότε αφού δεσμεύσει χώρο για τα κέντρα με βάση το Μ γίνεται αρχικοποίηση των κέντρων και των πεδίων του και τέλος επιστρέφει δείκτη στα αρχικοποιημένα κέντρα.

A computer screen shot of code

Description automatically generated

Η find\_min() με βάση τις αποστάσεις από όλα τα κέντρα επιλέγει την μικρότερη και την επιστρέφει μαζί με τον αριθμό της ομάδας σε έναν πίνακα που θα περιέχει αυτές τις δύο τιμές.

Η συνάρτηση calculate\_distance() υπολογίζει την ευκλείδεια απόσταση και την επιστρέφει στην συνάρτηση που την κάλεσε.

A screen shot of a computer program

Description automatically generated

Στην main() αρχικά ορίζουμε τις απαραίτητες μεταβλητές που θα χρειαστούν για την υλοποίηση του αλγορίθμου. Στη συνέχεια ανοίγουμε το αρχείο dataset για δίαβασμα και κάνουμε load τα στοιχεία στη μνήμη με χρήση της συνάρτησης load\_data(). Επίσης Δεσμεύουμε χώρο για τα κέντρα και τα αρχικοποιούμε με την συνάρτηση centers\_init(). Τέλος δεσμεύουμε μνήμη για τα error κάθε ομάδας .

A computer screen shot of a program code

Description automatically generated

Έχοντας κάνεις τις απαραίτητες δεσμεύσεις μνήμης και αρχικοποιήσεις για τα κέντρα μέσα σε έναν βρόγχο while υλοποιούμε τον αλγόριθμο. Αρχικά αρχικοποιούμε κάποιες επιπλέον τοπικές μεταβλητές που μας χρησιμεύουν σε υπολογισμούς . Στη συνέχεια, για κάθε στοιχείο του dataset υπολογίζουμε τις αποστάσεις του από κάθε κέντρο με χρήση της συνάρτησης calculate\_distance(). Επόμενο βήμα είναι η επιλογή της ομάδας στην οποία θα ενταχθεί το στοιχείο για την συγκεκριμένη εποχή. Η παραπάνω ενέργεια επιτυγχάνεται με την χρήση της συνάρτησης find\_min() . Ακολουθεί η αρχικοποίηση των τιμών για το νέο κέντρο και ο υπολογισμός σφάλματος ομαδοποίησης. Τέλος ελέγχουμε αν για κάθε κέντρο έχουν τροποποιηθεί οι συντεταγμένες του και αυξάνουμε έναν counter. Επίσης , μηδενίζουμε κάποιες μεταβλητές για να τις χρησιμοποιήσουμε στην επόμενη επανάληψη αν αυτή υπάρχει.

A screen shot of a computer program

Description automatically generated

Πρακτικά σε αυτό το σημείο εχει ολοκληρωθεί μία εποχή , έτσι τυπώνουμε τον αριθμό της και το σφάλμα ομαδοποίησης . Στη συνέχεια τυπώνουμε τις συντεταγμένες των κέντρων για την συγκεκριμένη εποχή. Ο κώδικας της main ολοκληρώνεται με τον έλεγχο του counter που προαναφέραμε . Συγκεκριμένα , αν ο μετρητής ισούται με τον αριθμό των κέντρων σημαίνει ότι κανένα κέντρο δεν άλλαξε ανάμεσα στις δύο τελευταίες επαναλήψεις οπότε σταματάει η εκτέλεση του βρόγχου. Αν δεν ισχύει η παραπάνω συνθήκη έχουμε τοποθετήσει έναν ενδεικτικό έλεγχο για να μην ξεφεύγουμε σε επαναλήψεις χωρίς λόγο , διότι για τα δεδομένα της εκφώνησης δεν απαιτούνται τόσες επαναλήψεις και μας βοήθησε να καταλήξουμε στην τελική λύση. Τέλος αυξάνουμε τον αριθμό των εποχών .

**2.3 Αποτελέσματα – Γραφικές παραστάσεις**

A screen shot of a graph

Description automatically generated

A screen shot of a graph

Description automatically generatedA screen shot of a graph

Description automatically generated

A screen shot of a graph

Description automatically generated

A graph with a blue line

Description automatically generated

**2.4 Συμπεράσματα:**

Με βάση τις παραπάνω γραφικές παραστάσεις ερχόμαστε στο συμπέρασμα ότι με τη μεταβολή του σφάλματος ομαδοποίησης μπορούμε να εκτιμήσουμε τον πραγματικό αριθμό των ομάδων. Αν παρατηρήσουμε προσεκτικά την μεταβολή του σφάλματος είναι εύκολα ορατό πως μετά τις 9 ομάδες η τιμή του δεν μειώνετε σημαντικά όπως στις προηγούμενες επαναλήψεις. Αν προσθέταμε και άλλες ομάδες η μείωση θα ήταν ακόμα μικρότερη. Συνεπώς μπορούμε να πούμε πως στο σημείο του γραφήματος όπου η μείωση της συνολικής διασποράς επιβραδύνεται αισθητά έχουμε μια προσέγγιση του πραγματικού αριθμού των ομάδων .