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

(dataset.txt πρέπει να ειναι στον ιδιο φακελο με το εκτελεσιμο)

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

```
for (int epoch = 0; epoch < epochs; epoch++) {
    int batch = 0;
    for (int example = 0; example < 4000; example++) {
        batch++;
        // Input and target for the selected example
        float input[D];
        float target[K];
        for (int i = 0; i < D; i++) {
            input[i] = x[example][i];
        }
        classification(input, target);

        // Perform forward pass and backpropagation
        backprop(&network, input, D, target, K);

        // Update weights and biases using batch update

        if(batch == B){
            for (int i = 0; i < network.num_layers; i++) {
                for (int j = 0; j < network.layers[i].num_neurons; j++) {
                    for (int k = 0; k < network.layers[i].neurons[j].num_input; k++) {
                        network.layers[i].neurons[j].Weights[k] -= learning_rate * network.layers[i].neurons[j].error_derivative[k];
                        network.layers[i].neurons[j].error_derivative[k] = 0;
                    }
                    network.layers[i].neurons[j].bias -= learning_rate * network.layers[i].neurons[j].bias_derivative;
                    network.layers[i].neurons[j].bias_derivative = 0;
                }
            }
            batch = 0;
        }
    }

    // Print the total error for every 100 epochs
    if (epoch % 1 == 0) {
        printf("Epoch %d - Total Error: %lf\n", epoch, network.total_error);
    }
}
```

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.

```
int count = 0;

for (int example = 4000; example < 8000; example++) {
    float input[D];
    float target[K];

    for (int i = 0; i < D; i++) {
        input[i] = x[example][i];
    }
    classification(input, target);

    float output[K];
    forward_pass(&network, input, D, output, K);

    int highest_value_index = 0;
    float highest_value = output[0];

    // Find the index of the highest value in the output array
    for (int i = 1; i < K; i++) {
        if (output[i] > highest_value) {
            highest_value = output[i];
            highest_value_index = i;
        }
    }

    // Check if the highest value index matches the target index
    if (target[highest_value_index] == 1) {
        count++;
    }
}
float accuracy = count / 4000.0;
printf("Accuracy: %f%%\n", accuracy * 100.0);

return 0;
```

Classification

```
void classification(float *x, float *t) {
    t[0] = t[1] = t[2] = t[3] = 0;  // Initialize array elements
    int flag = 0;

    if((pow((x[0] - 0.5) , 2) + pow((x[1] - 0.5) , 2)) < 0.2 && x[0] > 0.5){
        t[0] = 1;
        flag = 1;
    } else if((pow((x[0] - 0.5) , 2) + pow((x[1] - 0.5) , 2)) < 0.2 && x[0] < 0.5) {
        t[1] = 1;
        flag = 1;
    } else if((pow((x[0] + 0.5) , 2) + pow((x[1] + 0.5) , 2)) < 0.2 && x[0] > -0.5) {
        t[0] = 1;
        flag = 1;
    } else if((pow((x[0] + 0.5) , 2) + pow((x[1] + 0.5) , 2)) < 0.2 && x[0] < -0.5) {
        t[1] = 1;
        flag = 1;
    } else if((pow((x[0] - 0.5) , 2) + pow((x[1] + 0.5) , 2)) < 0.2 && x[0] > 0.5) {
        t[0] = 1;
        flag = 1;
    } else if((pow((x[0] - 0.5) , 2) + pow((x[1] + 0.5) , 2)) < 0.2 && x[0] < 0.5) {
        t[1] = 1;
        flag = 1;
    } else if((pow((x[0] + 0.5) , 2) + pow((x[1] - 0.5) , 2)) < 0.2 && x[0] > -0.5) {
        t[0] = 1;
        flag = 1;
    } else if((pow((x[0] + 0.5) , 2) + pow((x[1] - 0.5) , 2)) < 0.2 && x[0] < -0.5) {
        t[1] = 1;
        flag = 1;
    }
```

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

```
void forward_pass(struct Neural_Network* network, float *x, int d, float *y, int k){
    int i = 0;
    float *input_vector = NULL;
    float *output_vector = NULL;

    for (i; i < NUM_LAYERS + 1; i++) {

        if (i == 0) {
            // If we're in the first hidden layer, the input vector is the x vector
            input_vector = malloc(sizeof(float) * d);
            for (int l = 0; l < d; l++) {
                input_vector[l] = x[l];
            }
        } else {
            // Free the output_vector from the previous layer
            free(output_vector);

            // Free the output_vector from the previous layer
            free(input_vector);

            // Allocate memory for the input_vector of the current layer
            input_vector = malloc(sizeof(float) * network->layers[i-1].num_neurons);

            // Copy values from the output_vector of the previous layer to the input_vector
            for (int l = 0; l < network->layers[i-1].num_neurons; l++) {
                input_vector[l] = network->layers[i-1].neurons[l].output;
            }
        }

        // Allocate memory for the output_vector of the current layer
        output_vector = malloc(sizeof(float) * network->layers[i].num_neurons);
```

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.

```
        for (int j = 0; j < network->layers[i].num_neurons; j++) {
            float weighted_sum = 0;
            // Reset the weighted sum for each neuron
            network->layers[i].neurons[j].weighted_sum = 0.0;
            for (int k = 0; k < network->layers[i].neurons[j].num_input; k++) {
                network->layers[i].neurons[j].weighted_sum += input_vector[k] * network->layers[i].neurons[j].Weights[k];
            }
            network->layers[i].neurons[j].weighted_sum += network->layers[i].neurons[j].bias;

            if(i == NUM_LAYERS) {
                // linear for output layer
                network->layers[i].neurons[j].output = sigmoid(network->layers[i].neurons[j].weighted_sum);
            }else {
                // hidden layers activation functions
                if (FUNC == 0) {
                    // Logistic (Sigmoid) Activation Function
                    network->layers[i].neurons[j].output = sigmoid(network->layers[i].neurons[j].weighted_sum);
                } else if(FUNC == 1) {
                    // hyperbolic Activation Function
                    network->layers[i].neurons[j].output = hyperbolic(network->layers[i].neurons[j].weighted_sum);
                } else {
                    // relu
                    network->layers[i].neurons[j].output = relu(network->layers[i].neurons[j].weighted_sum);
                }
            }
            // Copy the values of each neuron to the output vector
            output_vector[j] = network->layers[i].neurons[j].output;
        }
    }
```

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.

```
    // Calculate output layer vector y with k
    for (int l = 0; l < k; l++) {
        y[l] = output_vector[l];
    }

    // Free memory allocated for input and output vectors
    free(input_vector);
    free(output_vector);
}
```

Τελος το output περνιέται στο πινακα y.

Backprop

```
void backprop(struct Neural_Network *network, float *x, int d, float *t, int k) {

    float *output = malloc(sizeof(float) * K);

    forward_pass(network, x , d, output, K);

    // compute error
    float error = 0;
    for(int i = 0; i < K; i++){
        error += pow(t[i] - output[i] , 2);
    }

    error = error /2;
    network->total_error = error;
```

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.

```
    for(int i = NUM_LAYERS; i >= 0; i--){
        for(int j = 0; j < network->layers[i].num_neurons; j++){
            if(i == NUM_LAYERS) {
                network->layers[i].neurons[j].error = network->layers[i].neurons[j].output * (1 - network->layers[i].neurons[j].output) *
                ( network->layers[i].neurons[j].output- t[j]);
            }else {
                float error_accumulator = 0.0;
                for(int k = 0; k < network->layers[i + 1].num_neurons; k++){
                        error_accumulator += network->layers[i + 1].neurons[k].Weights[j] *
                                    network->layers[i + 1].neurons[k].error;
                }
                if(FUNC == 0) {
                    network->layers[i].neurons[j].error = network->layers[i].neurons[j].output *
                                                (1 - network->layers[i].neurons[j].output) *
                                                error_accumulator;
                } else if(FUNC == 1){
                    network->layers[i].neurons[j].error = (1 - pow(network->layers[i].neurons[j].output, 2)) * error_accumulator;
                } else {
                    float relu_der = 0;
                    if(network->layers[i].neurons[j].output > 0) {
                        relu_der = 1;
                    } else {
                        relu_der = 0;
                    }
                    network->layers[i].neurons[j].error = relu_der * error_accumulator;

                }

            }

            if(i == 0){
                // compute error derivatives
                for(int l = 0; l < network->layers[i].neurons[j].num_input; l++){
                    network->layers[i].neurons[j].error_derivative[l] += network->layers[i].neurons[j].error * x[l];
                }
            }else{
                for(int l = 0; l < network->layers[i].neurons[j].num_input; l++){
                    network->layers[i].neurons[j].error_derivative[l] += network->layers[i].neurons[j].error * network->layers[i - 1].neurons[l].output;
                }
            }
            network->layers[i].neurons[j].bias_derivative += network->layers[i].neurons[j].error;

        }
    }

}
```

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	B	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%
39	5	5	5	hyberbolic	4000	71.02%
40	5	5	5	relu	4000	63.02%
41	15	15	15	hyberbolic	4000	83.55%
42	15	15	15	relu	4000	83.85%

43	20	20	20	hyberbolic	4000	87.69%

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.

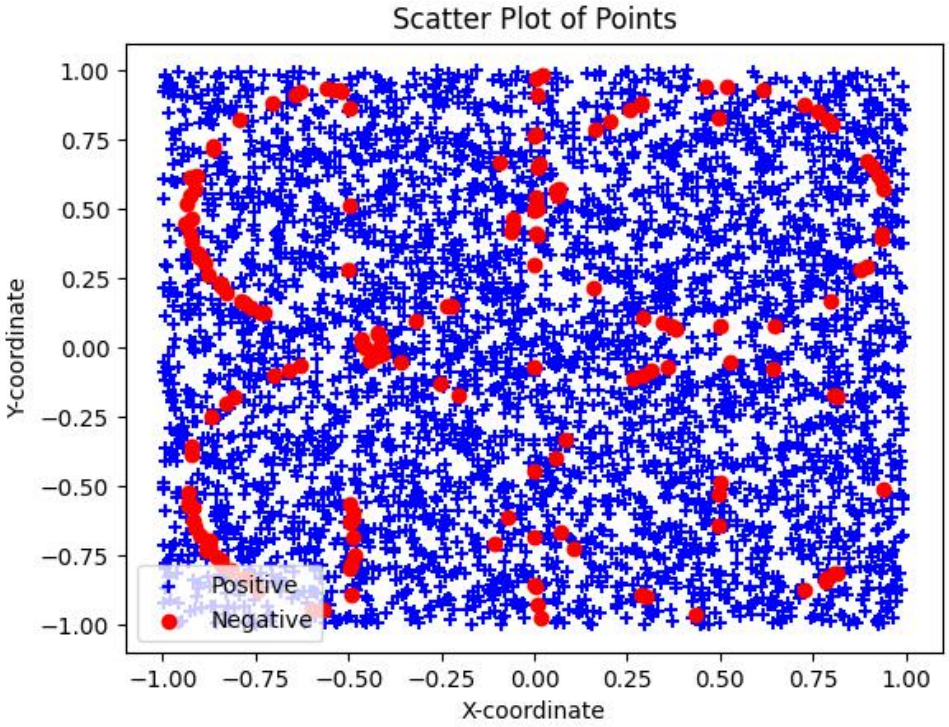
Στα h1=7, h2=10, h3=7 βλέπουμε πως ολοι οι συνδιασμοι φτανουν ικανοποιητικη συμπεριφερα εκτος απο το δικτυο με συναρτηση ενεργοποιησης logistic (#13). Σε αυτα τα δικτυα φτασαμε το 90% γενικευτικη ικανοτητα (#12).

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 Μ 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 -o dataset-maker dataset-maker.c και gcc kMeans.c -lm

(dataset2.txt πρέπει να ειναι στον ιδιο φακελο με το εκτελεσιμο)

dataset-maker.c

```c
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void create_values(double min_x, double max_x, double min_y, double max_y, int reps, FILE *fp);

int main(){

    srand(time(NULL));
    FILE *file = fopen("dataset2.txt", "w");

    if(file == NULL){
    printf("Something went wrong with the file\n");
        return -1;
    }

    create_values(0.8,1.2,0.8,1.2,150,file);                        //1)
    create_values(0,0.5,0,0.5,150,file);                            //2)
    create_values(1.5,2,1.5,2,150,file);                            //3)
    create_values(0,0.5,1.5,2,150,file);                            //4)
    create_values(1.5,2,1.5,2,150,file);                            //5)
    create_values(0,0.4,0.8,1.2,75,file);                           //6)
    create_values(1.6,2,0.8,1.2,75,file);                           //7)
    create_values(0.8,1.2,0.3,0.7,75,file);                         //8)
    create_values(0.8,1.2,1.3,1.7,75,file);                         //9)
    create_values(0,2,0,2,150,file);                                //10)

    return 0;
}

    void create_values(double min_x, double max_x, double min_y, double max_y,int reps, FILE *fp){

        double x,y,range_x, range_y,div_x, div_y;
        int i;

        range_x = (max_x - min_x);
        range_y = (max_y - min_y);

        div_x = RAND_MAX /range_x;
        div_y = RAND_MAX /range_y;

        for(i=0; i<reps; i++){

            x = min_x + (rand()/div_x);
            y = min_y + (rand()/div_y);

            fprintf(fp,"(x1,x2) = (%f,%f)\n",x,y);
        }
    }
```

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

#define M 12
#define N 1200

struct center_s{
    double x;
    double y;
    int team;
    int size;
    double new_x;
    double new_y;
};

typedef struct center_s center_t;

double** load_data(FILE *fp);
center_t *centers_init();
int create_random_values(int min_y, int max_y);
double** allocate_numOf_teams();
double* resize_teams(double *teams, center_t center);
double calculate_distance(double x, double y, double cen_x, double cen_y);
void find_min(double *values, double *array);
```

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

```
177  double** load_data(FILE *fp){
178    double **array;
179    double x;
180    double y;
181    int i;
182    int count = 0;
183    int cols = 2;
184    int rows = N;
185
186    array = (double **)malloc(cols*sizeof(double *));
187    if(array==NULL){
188      perror("Error allocating memory");
189      return NULL;
190    }
191
192    for(i=0; i<cols; i++){
193      array[i] = (double *) malloc(rows*sizeof(double));
194      if(array[i] == NULL){
195        perror("Error allocating memory");
196        return NULL;
197      }
198    }
199
200    while(fscanf(fp,"(x1,x2)=(%lf,%lf)\n",&x,&y)!= EOF){
201      array[0][count] = x;
202      array[1][count] = y;
203      count++;
204    }
205
206    return array;
207  }
208
```

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.

```
156  double** allocate_numOf_teams(){
157    double **oj;
158    int i;
159
160    oj = (double **)malloc(M*sizeof(double *));
161    if(oj == NULL){
162      perror("Error allocating memory for the teams\n");
163      return NULL;
164    }
165
166    for(i=0; i<M; i++){
167      oj[i] = (double *)malloc(N*sizeof(double));
168      if(oj[i] == NULL){
169        perror("Error allocating memory for the teams\n");
170        return NULL;
171      }
172    }
173
174    return oj;
175  }
```

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.

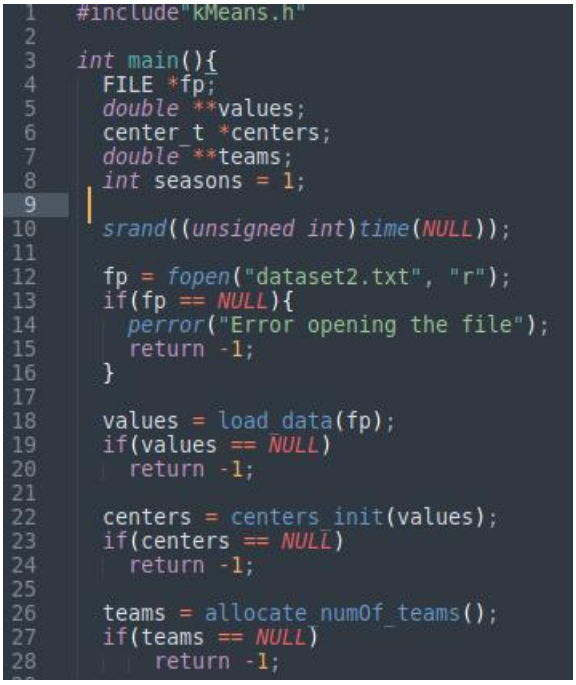
```c
center_t *centers_init(double **values){
  center_t *centers;
  int i;

  centers = (center_t *)malloc(M*sizeof(center_t));
  if(centers ==NULL){
    perror("There was an error allocating memory for centers\n");
    return NULL;
  }

  for(i=0; i<M;i++){
    centers[i].y = create_random_values(0,1199);

    centers[i].x = (values[0][(int)(centers[i].y)]);
    centers[i].y = (values[1][(int)(centers[i].y)]);
    centers[i].team = i;
    centers[i].size = 0;
    centers[i].new_x = 0;
    centers[i].new_y = 0;
  }

  return centers;
}

int create_random_values(int min_y, int max_y){
  int value, y, range_y;

  range_y = (max_y - min_y);

  value = (double)(min_y + rand()%(range_y+1));

  return value;
}
```

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 M γίνεται αρχικοποίηση των κέντρων και των πεδίων του και τέλος επιστρέφει δείκτη στα αρχικοποιημένα κέντρα.

```c
void find_min(double *values, double *array){
  int i;
  array[0] = values[0];
  array[1] = 0;
  for(i=1; i<M; i++){
    if(array[0]>values[i]){
      array[0]= values[i];
      array[1] = i;

    }
  }

}

double calculate_distance(double x, double y, double cen_x, double cen_y){

  return sqrt(pow((cen_x-x), 2)+pow((cen_y-y), 2));
}
```

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.



```c
#include"kMeans.h"

int main(){
  FILE *fp;
  double **values;
  center_t *centers;
  double **teams;
  int seasons = 1;

  srand((unsigned int)time(NULL));

  fp = fopen("dataset2.txt", "r");
  if(fp == NULL){
    perror("Error opening the file");
    return -1;
  }

  values = load_data(fp);
  if(values == NULL)
    return -1;

  centers = centers_init(values);
  if(centers == NULL)
    return -1;

  teams = allocate_numOf_teams();
  if(teams == NULL)
      return -1;
```

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 .

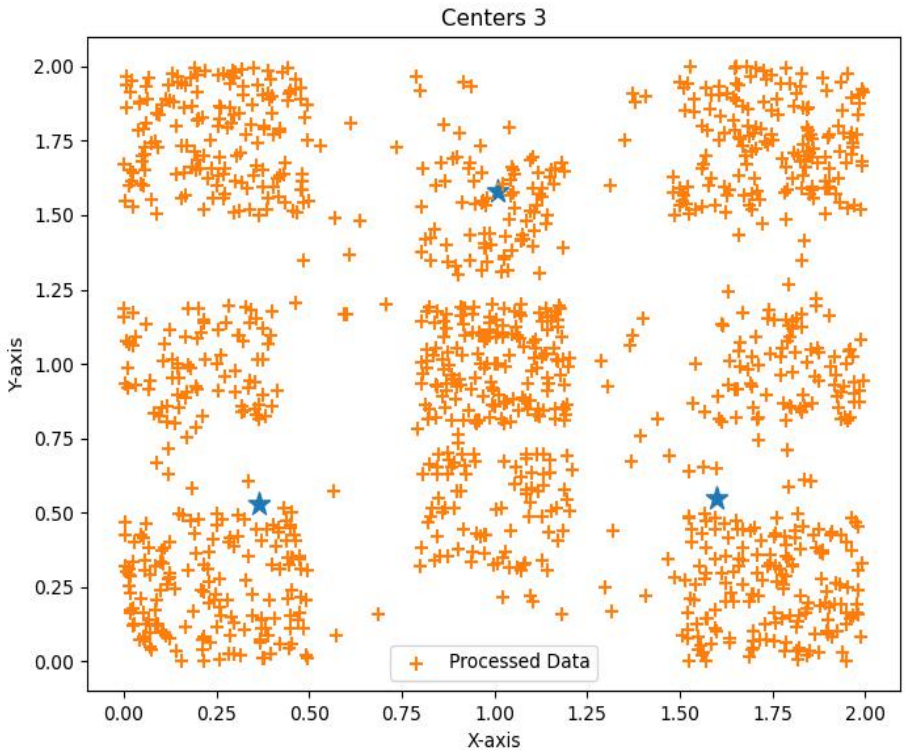
```
30    while(1){
31      double error = 0;
32      int count = 0;
33      int i,j;
34      double minTeam [2] ;
35
36      for(i=0; i<N; i++){
37        double min_dist[M];
38        for(j=0; j<M; j++){
39          min_dist[j] = calculate_distance(values[0][i],values[1][i],centers[j].x,centers[j].y);
40
41        }
42
43        find_min(min_dist,minTeam);
44        centers[(int)minTeam[1]].size++;
45        centers[(int)minTeam[1]].new_x += values[0][i];
46        centers[(int)minTeam[1]].new_y += values[1][i];
47        teams[(int)minTeam[1]][centers[(int)minTeam[1]].size] = minTeam[0];
48      }
49
50      for(i=0; i<M; i++){
51        for(j=0; j<centers[i].size; j++){
52          error += teams[i][j];
53
54        }
55        if(centers[i].size>0){
56          centers[i].new_x = centers[i].new_x / centers[i].size;
57          centers[i].new_y = centers[i].new_y / centers[i].size;
58        }
59      }
60
61      for(i=0; i<M; i++){
62        if(centers[i].new_x == centers[i].x && centers[i].new_y == centers[i].y)
63          count++ ;
64
65        centers[i].x = centers[i].new_x;
66        centers[i].y = centers[i].new_y;
67        centers[i].new_x = 0;
68        centers[i].new_y = 0;
69        centers[i].size = 0;
70      }
```

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.

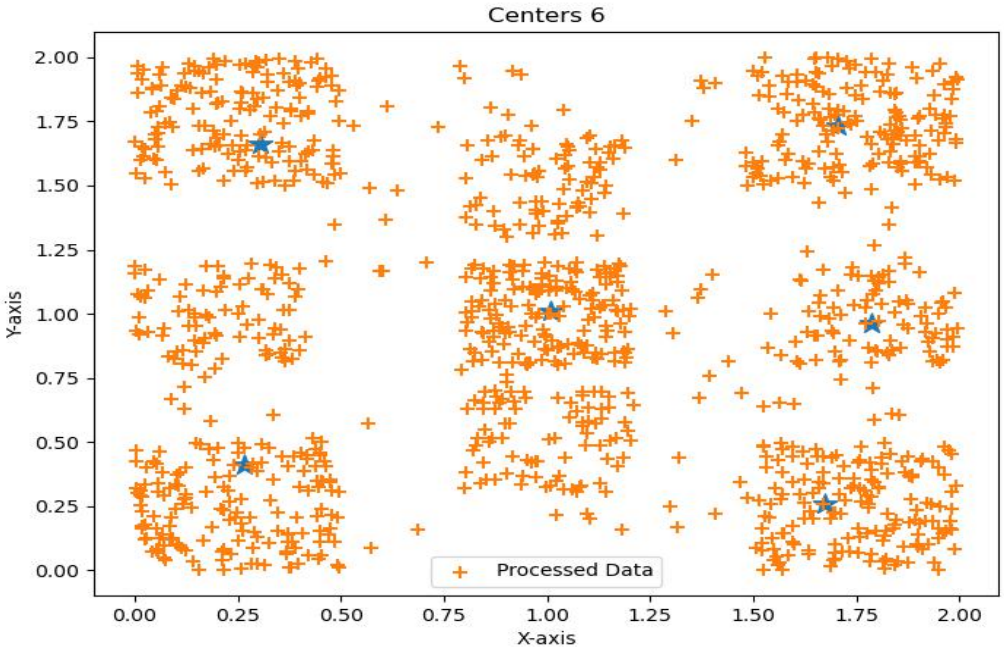
```
72      printf("Season %d error : %lf\n",seasons,error);
73
74      for(i=0;i<M;i++){
75        printf("center[%d]-team : %d , (x = %lf ,y = %lf)\n",i+1,centers[i].team+1,centers[i].x,centers[i].y);
76      }
77
78      printf("\n");
79
80      if(count == M)
81        break;
82
83      if(seasons == 1000){
84        perror("Too many Seasons");
85        break;
86      }
87      seasons++;
88    }
89
90    return 0;
91
92  }
```

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 .

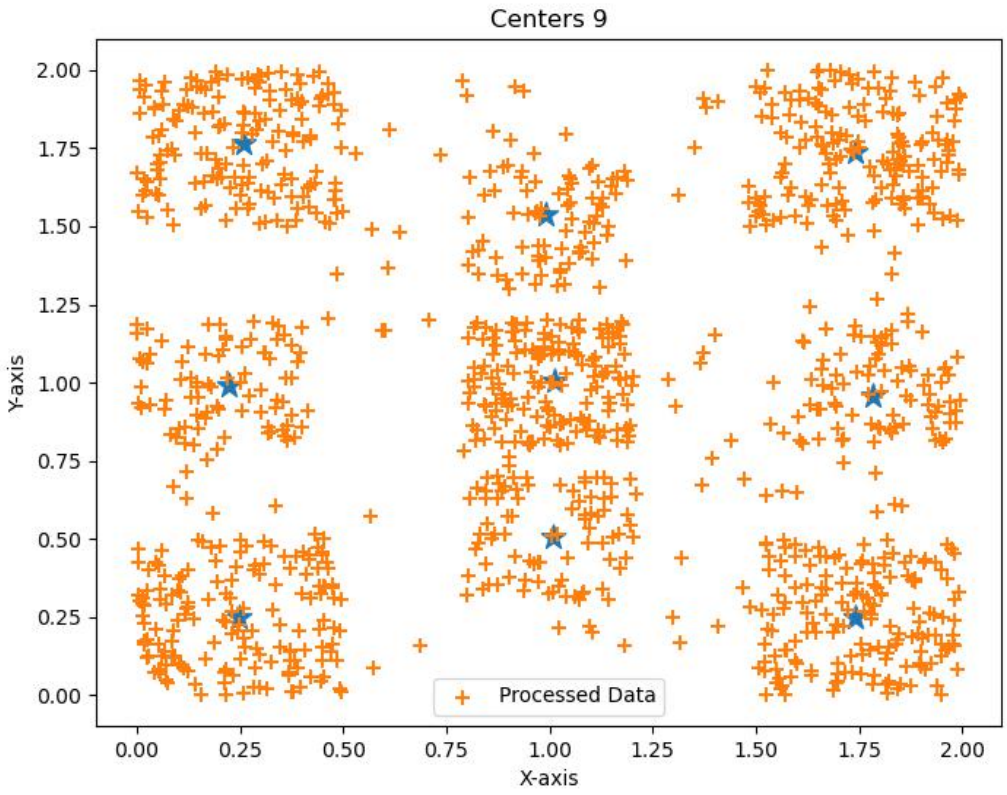
2.3 Αποτελέσματα – Γραφικές παραστάσεις



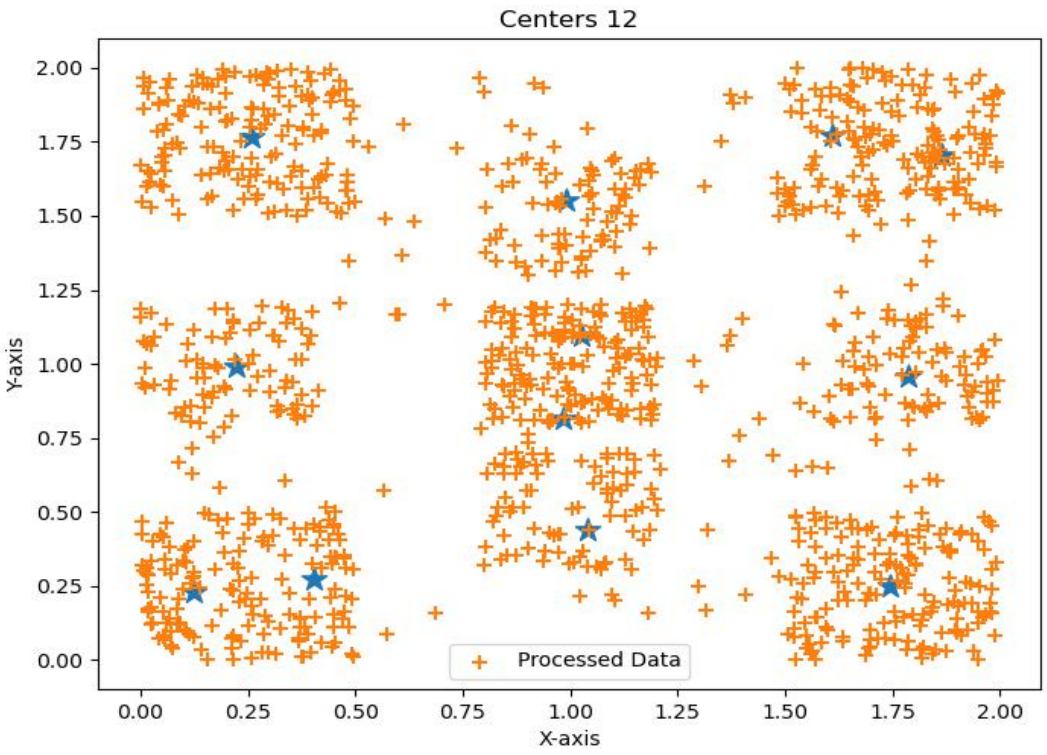
3 ομαδες


Centers 6
Y-axis
X-axis
Processed Data
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

δομαδες


Centers 9
Y-axis
X-axis
Processed Data
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

9 ομαδες


Centers 12
Processed Data
X-axis
Y-axis
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

12 ομαδες


Error Plot
Data
M
Y-axis
200
250
300
350
400
450
500
4
6
8
10
12

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 .