Homework7

cat1997

April 9, 2020

1 Pledge

I have neither given nor received unauthorized assistance on this assignment

2 Kmeans.c File

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

//struct to hold cluster info separate from data
typedef struct{
  float** k_means; //array of cluster means
  float** members;//array of members in cluster
}cluster;

//float array constructors
float** arr_construct(int num_rows, int num_cols){
  float** float_arr = malloc(num_rows*sizeof(float*));
  for(int i=0;i<num_rows;++i){
    float_arr[i] = calloc(num_cols, sizeof(float));
  }
  return float_arr;
}
//initializes cluster struct with 7 randomly chosen points as
//initail cluster means
cluster initClust(dataArray data){
  cluster c;
  c.members = arr_construct(data.clusters, data.rows);//array of members in cluster
  c.k_means = arr_construct(data.clusters, 2);
  time_t t;
```

```
  srand((unsigned) time(&t));
  for(int i = 0;i<data.clusters;i++){
    int rand_num = rand() % data.rows;
    c.k_means[i][0] = data.x[rand_num][0];
    c.k_means[i][1] = data.x[rand_num][1];
    c.members[i][0] = 1; //count for how many vals
    c.members[i][1] = rand_num;
  }
  return c;
}
//calculates diatance between 2 points
float distance(float* d, float* c){ //d = data point, c = cluster pt
  float num_1 = pow((d[0] - c[0]), 2);
  float num_2 = pow((d[1] - c[1]), 2);
  return sqrt(num_1 + num_2);
}

int find_cluster(float* d, float** means, int num_c){
  float min_dist = distance(d, means[0]);
  int clust_idx = 0;
  for(int i=1;i<num_c;i++){
    float dist = distance(d, means[i]);
    if(dist<min_dist){
      min_dist = dist;
      clust_idx = i;
    }
  }
  return clust_idx;
}
//frees the float arrays in the struct
void float_destruct(float** f,int n_rows){
  for(int i=0; i < n_rows;++i){
       free(f[i]);
  }
  free(f);
}
//claucluates the means of each cluster given the points in them
float calc_means(int clust_num, cluster c, dataArray data, int x_y){
  int idx = 1;
  float total = 0;
  while(idx<=c.members[clust_num][0]){
    int index = (int)c.members[clust_num][idx];
    total = total + data.x[index][x_y];
    idx++;
  }
  return total/(idx-1);
```

```
}
//returns the cluster that a given point is a member of given its index
int idx_to_cluster(int index, dataArray data, cluster c){
  for(int j=0; j<data.clusters; j++){//go thru all clusters
    for(int k=1; k<=c.members[j][0];k++){//each member in cluster
      if(index==c.members[j][k]){
        return j;
      }
    }
  }
  return -1;
}
//writes data to file in format ('x'  'y'  'clutster #')
void write_to_file(FILE* fptr, cluster c, dataArray d){
  fprintf(fptr, "x            y           cluster\n");
  for(int i=0;i<d.rows; i++){
    float* point = d.x[i];
    int cluster = idx_to_cluster(i, d, c);
    fprintf(fptr,"%.5f     %.5f      %d\n", point[0], point[1], cluster+1);
  }

}


int main(int carg, char** varg){
  dataArray data = setClustArray(varg[1],varg[2]);
  cluster c = initClust(data);
  //first round of putting points into clusters
  for(int i=0; i<data.rows; i++){ //check if index has already been sorted
    int check = idx_to_cluster(i, data, c);
    if(check == -1){
      int cluster = find_cluster(data.x[i], c.k_means,data.clusters);
      c.members[cluster][0] =  c.members[cluster][0]+1;
      int next_spot =(int)c.members[cluster][0];
      c.members[cluster][next_spot] = i;
      c.k_means[cluster][0] = calc_means(cluster, c, data, 0);
      c.k_means[cluster][1] = calc_means(cluster, c, data, 1);
    }
  }
  //go through points and recalc clusters until they no linger change
  int changed = 1;
  while(changed){
    int num_changes = 0;
    for(int i=0; i<data.rows; i++){
      int og_clust = idx_to_cluster(i, data, c);
      if(og_clust < 0){
```

```
      ;
    }
    else{
      float low_dist = distance(data.x[i], c.k_means[og_clust]);
      int new_clust = -1;
      //calc distances for each cluster and see if there's a better fit
      for(int k=0; k<data.clusters;k++){
        float new_dist = distance(data.x[i], c.k_means[k]);
        if(new_dist < low_dist){
          low_dist = new_dist;
          new_clust = k;
        }
      }
      //recalculate cluster means  if a point was moved
      if(new_clust>=0){
        num_changes++;
        c.members[new_clust][0] =  c.members[new_clust][0] + 1;
        int next_spot = (int)c.members[new_clust][0];
        c.members[new_clust][next_spot] = i;
        c.members[og_clust][0] =  c.members[og_clust][0] -1;
        c.k_means[new_clust][0] = calc_means(new_clust, c, data, 0);
        c.k_means[new_clust][1] = calc_means(new_clust, c, data, 1);
        c.k_means[og_clust][0] = calc_means(og_clust, c, data, 0);
        c.k_means[og_clust][1] = calc_means(og_clust, c, data, 1);
      }
    }
  }
  changed = num_changes;//num_changes will be 0 if nothing changed. Signifies
                        //all clusters are found
}
FILE *f = fopen("scatterdata.txt", "w");
write_to_file(f, c, data);
float_destruct(c.k_means, data.clusters);//free allocated arrays
float_destruct(c.members, data.clusters);
float_destruct(data.x, data.rows);
return 0;
}
```

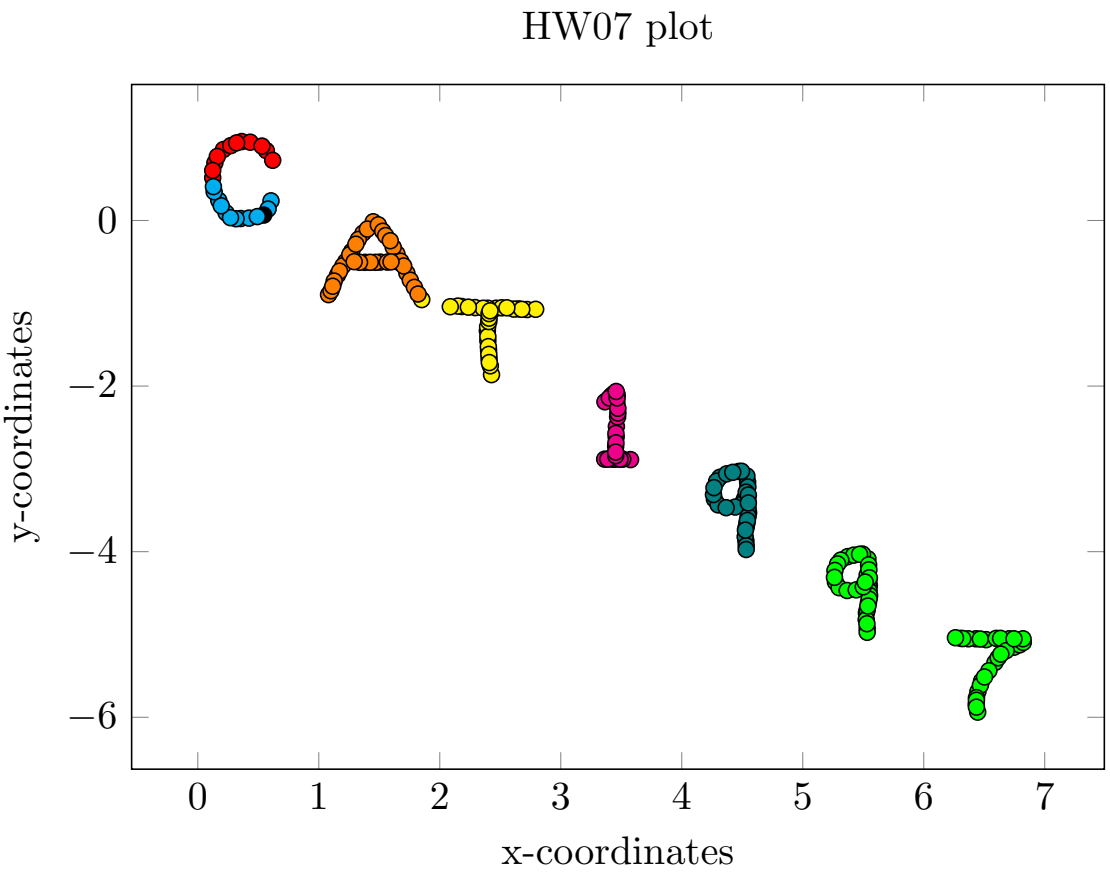
3 Modified Header File

```
/*
array.h by Jonathan Baker

This file defines a useful array struct, similar to a matrix,
and functions for working with arrays.
```

```c
You may use and modify this code for HW 7.
*/
//I have neither given nor received unauthorized assistance on this assignment

#include<stdlib.h>
#include<stdio.h>

typedef struct{
    int m,n;
    float** x;
} array;

typedef struct{
  int rows;
  float**x;
  int clusters;
}dataArray;

array arrayConstructor(int m,int n){
    /* Construct a blank array */
    array a;
    a.m = m;
    a.n = n;
    a.x = malloc(m*sizeof(float*));
    for(int i=0;i<m;++i){
        a.x[i] = malloc(n*(sizeof(float)));
    }
    return a;
}

void arraySet(array a,int i,int j,float value){
    /* Set the value of a single entry in an array */
    if(i<0 || j<0){
        fprintf(stderr,"Indices must be non-negative\n");
        exit(-1);
    }
    if(i>=a.m || j>=a.n){
        fprintf(stderr,"Getting index (%d,%d) out of bounds on array of size %d x %d\n",i,j,
        exit(-1);
    }
    a.x[i][j] = value;
}

float arrayGet(array a,int i,int j){
    /* Get a single entry in an array */
```

```
    if(i<0 || j<0){
        fprintf(stderr,"Indices must be non-negative\n");
        exit(-1);
    }
    if(i>=a.m || j>=a.n){
        fprintf(stderr,"Getting index (%d,%d) out of bounds on array of size %d x %d\n",i,j,
        exit(-1);
    }
    return a.x[i][j];
}

array arrayMean(array data){
    /* Returns the column-wise mean of array "data" */
    array means = arrayConstructor(1,data.n);
    int i,j;
    float tot;
    for(j=0;j<data.n;++j){
        tot = 0;
        for(i=0;i<data.m;++i){
            tot += arrayGet(data,i,j);
        }
        tot /= data.m;
        arraySet(means,0,j,tot);
    }
    return means;
}

void arrayDestructor(array a){
    for(int i=0; i < a.m ;++i){
        free(a.x[i]);
    }
    free(a.x);
}

array readData2D(char* fname){
    /* Read 2D data from file with the given name and store it in "data" */
    FILE* fptr = fopen(fname,"r");
    if(fptr == NULL){
        fprintf(stderr,"Failed to open file '%s'\n",fname);
        exit(-1);
    }
    int BUFFSIZE = 1024;
    char line[BUFFSIZE];

    // Skip header line
    fgets(line,BUFFSIZE,fptr);
```

```
    // Get number of data lines
    fgets(line,BUFFSIZE,fptr);
    int n = atoi(line);
    array data = arrayConstructor(n,2);

    // Skip header line
    fgets(line,BUFFSIZE,fptr);

    // Get data
    char* space;
    for(int i=0;i<n;++i){
        // Get the next line
        fgets(line,BUFFSIZE,fptr);
        // Locate the space that divides x from y
        space = strstr(line," ");
        // Overwrite the space to split the string in two
        space[0] = '\0';
        // Record x data
        arraySet(data,i,0,atof(line));
        // Record y data
        arraySet(data,i,1,atof(space+1));
    }
    fclose(fptr);
    return data;
}

//initializes and sets values in cluster array data struct
dataArray setClustArray(char *fname, char *clusters){
  dataArray c;
  array initData = readData2D(fname);
  c.clusters = atoi(clusters);//set num clusters from arg
  c.rows = initData.m; //set num of lines
  c.x = malloc(c.rows*sizeof(float*));
  for(int i=0;i<c.rows;++i){
    c.x[i] = malloc(initData.n*(sizeof(float)));
  }
  for(int i=0;i<c.rows;++i){
    //set x vals
    c.x[i][0] = initData.x[i][0];
    //set y vals
    c.x[i][1] = initData.x[i][1];
  }
  arrayDestructor(initData);
  return c;
}
```



4 Valgrind Output

```
cat@cat-VirtualBox:~/cmda3634/HW07$ gcc -ggdb -o means kmeans.c -lm
cat@cat-VirtualBox:~/cmda3634/HW07$ valgrind --leak-check=full --show-leak-kinds=all ./means
==11093== Memcheck, a memory error detector
==11093== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==11093== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==11093== Command: ./means test.dat 7
==11093==
==11093==
==11093== HEAP SUMMARY:
==11093==     in use at exit: 0 bytes in 0 blocks
==11093==   total heap usage: 444 allocs, 444 frees, 22,124 bytes allocated
==11093==
==11093== All heap blocks were freed -- no leaks are possible
==11093==
```

```
==11093== For counts of detected and suppressed errors, rerun with: -v
==11093== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

5 Discussion

My program enters an infinite loop about half the time you run it. I realize that each letter is supposed to be a cluster so something must be a little off with my logic. I'm pretty sure there is a problem with my function where you input a data point's index and get back the cluster that that it is a member of. It works most times but once it gets up to the higher indexes sometimes it returns that it hasn't been added to a cluster yet even after they all should have been added multiple times. This interferes when it's iterating through and looking to find if there is a different cluster with a smaller distance after they have all been added once. I think it is also the reason for the infinite loop because it never reaches an equilibrium point where the clusters stop changing. However this doesn't always happen. I didn't have time to figure out how to fix that so I just put in an if statement to prevent it from reaching this point so it'd at least return something, and it only seemed to mess up the results a small amount.