//usr/bin/cc -Ofast -lm "${0}" -o "${0%.c}" && ./"${0%.c}" "$@"; s=$?; rm ./"${0%.c}"; exit $s

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

typedef struct matrix{

int rows, cols;

double \*\*vals;

} matrix;

matrix csv\_to\_matrix(char \*filename, int header);

matrix make\_matrix(int rows, int cols);

void zero\_matrix(matrix m);

void copy(double \*x, double \*y, int n);

double dist(double \*x, double \*y, int n);

int \*sample(int n);

int find\_int\_arg(int argc, char \*\*argv, char \*arg, int def);

int find\_arg(int argc, char\* argv[], char \*arg);

int closest\_center(double \*datum, matrix centers)

{

int j;

int best = 0;

double best\_dist = dist(datum, centers.vals[best], centers.cols);

for(j = 0; j < centers.rows; ++j){

double new\_dist = dist(datum, centers.vals[j], centers.cols);

if(new\_dist < best\_dist){

best\_dist = new\_dist;

best = j;

}

}

return best;

}

double dist\_to\_closest\_center(double \*datum, matrix centers)

{

int ci = closest\_center(datum, centers);

return dist(datum, centers.vals[ci], centers.cols);

}

int kmeans\_expectation(matrix data, int \*assignments, matrix centers)

{

int i;

int converged = 1;

for(i = 0; i < data.rows; ++i){

int closest = closest\_center(data.vals[i], centers);

if(closest != assignments[i]) converged = 0;

assignments[i] = closest;

}

return converged;

}

void kmeans\_maximization(matrix data, int \*assignments, matrix centers)

{

int i,j;

int \*counts = calloc(centers.rows, sizeof(int));

zero\_matrix(centers);

for(i = 0; i < data.rows; ++i){

++counts[assignments[i]];

for(j = 0; j < data.cols; ++j){

centers.vals[assignments[i]][j] += data.vals[i][j];

}

}

for(i = 0; i < centers.rows; ++i){

if(counts[i]){

for(j = 0; j < centers.cols; ++j){

centers.vals[i][j] /= counts[i];

}

}

}

}

double WCSS(matrix data, int \*assignments, matrix centers)

{

int i, j;

double sum = 0;

for(i = 0; i < data.rows; ++i){

int ci = assignments[i];

sum += (1 - dist(data.vals[i], centers.vals[ci], data.cols));

}

return sum / data.rows;

}

typedef struct{

int \*assignments;

matrix centers;

} model;

void smart\_centers(matrix data, matrix centers) {

int i,j;

copy(data.vals[rand()%data.rows], centers.vals[0], data.cols);

double \*weights = calloc(data.rows, sizeof(double));

int clusters = centers.rows;

for (i = 1; i < clusters; ++i) {

double sum = 0;

centers.rows = i;

for (j = 0; j < data.rows; ++j) {

weights[j] = dist\_to\_closest\_center(data.vals[j], centers);

sum += weights[j];

}

double r = sum\*((double)rand()/RAND\_MAX);

for (j = 0; j < data.rows; ++j) {

r -= weights[j];

if(r <= 0){

copy(data.vals[j], centers.vals[i], data.cols);

break;

}

}

}

free(weights);

}

void random\_centers(matrix data, matrix centers){

int i;

int \*s = sample(data.rows);

for(i = 0; i < centers.rows; ++i){

copy(data.vals[s[i]], centers.vals[i], data.cols);

}

free(s);

}

model do\_kmeans(matrix data, int k)

{

matrix centers = make\_matrix(k, data.cols);

int \*assignments = calloc(data.rows, sizeof(int));

smart\_centers(data, centers);

//random\_centers(data, centers);

if(k == 1) kmeans\_maximization(data, assignments, centers);

while(!kmeans\_expectation(data, assignments, centers)){

kmeans\_maximization(data, assignments, centers);

}

model m;

m.assignments = assignments;

m.centers = centers;

return m;

}

int main(int argc, char \*argv[])

{

if(argc < 3){

fprintf(stderr, "usage: %s <csv-file> [points/centers/stats]\n", argv[0]);

return 0;

}

int i,j;

srand(time(0));

matrix data = csv\_to\_matrix(argv[1], 0);

int k = find\_int\_arg(argc, argv, "-k", 2);

int header = find\_arg(argc, argv, "-h");

int count = find\_arg(argc, argv, "-c");

if(strcmp(argv[2], "assignments")==0){

model m = do\_kmeans(data, k);

int \*assignments = m.assignments;

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

if(i != 0) printf("-\n");

for(j = 0; j < data.rows; ++j){

if(!(assignments[j] == i)) continue;

printf("%f, %f\n", data.vals[j][0], data.vals[j][1]);

}

}

}else if(strcmp(argv[2], "centers")==0){

model m = do\_kmeans(data, k);

printf("WCSS: %f\n", WCSS(data, m.assignments, m.centers));

int \*counts = 0;

if(count){

counts = calloc(k, sizeof(int));

for(j = 0; j < data.rows; ++j){

++counts[m.assignments[j]];

}

}

for(j = 0; j < m.centers.rows; ++j){

if(count) printf("%d, ", counts[j]);

printf("%f, %f\n", m.centers.vals[j][0], m.centers.vals[j][1]);

}

}else if(strcmp(argv[2], "scan")==0){

for(i = 1; i <= k; ++i){

model m = do\_kmeans(data, i);

printf("%f\n", WCSS(data, m.assignments, m.centers));

}

}

return 0;

}

// Utility functions

int \*sample(int n)

{

int i;

int \*s = calloc(n, sizeof(int));

for(i = 0; i < n; ++i) s[i] = i;

for(i = n-1; i >= 0; --i){

int swap = s[i];

int index = rand()%(i+1);

s[i] = s[index];

s[index] = swap;

}

return s;

}

double dist(double \*x, double \*y, int n)

{

int i;

double mw = (x[0] < y[0]) ? x[0] : y[0];

double mh = (x[1] < y[1]) ? x[1] : y[1];

double inter = mw\*mh;

double sum = x[0]\*x[1] + y[0]\*y[1];

double un = sum - inter;

double iou = inter/un;

return 1-iou;

}

void copy(double \*x, double \*y, int n)

{

int i;

for(i = 0; i < n; ++i) y[i] = x[i];

}

void error(char \*s){

fprintf(stderr, "Error: %s\n", s);

exit(-1);

}

char \*fgetl(FILE \*fp)

{

if(feof(fp)) return 0;

int size = 512;

char \*line = malloc(size\*sizeof(char));

if(!fgets(line, size, fp)){

free(line);

return 0;

}

int curr = strlen(line);

while(line[curr-1]!='\n'){

size \*= 2;

line = realloc(line, size\*sizeof(char));

if(!line) error("Malloc");

fgets(&line[curr], size-curr, fp);

curr = strlen(line);

}

line[curr-1] = '\0';

return line;

}

// Matrix stuff

int count\_fields(char \*line)

{

int count = 0;

int done = 0;

char \*c;

for(c = line; !done; ++c){

done = (\*c == '\0');

if(\*c == ',' || done) ++count;

}

return count;

}

double \*parse\_fields(char \*l, int n)

{

int i;

double \*field = calloc(n, sizeof(double));

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

field[i] = atof(l);

l = strchr(l, ',')+1;

}

return field;

}

matrix make\_matrix(int rows, int cols)

{

matrix m;

m.rows = rows;

m.cols = cols;

m.vals = calloc(m.rows, sizeof(double \*));

int i;

for(i = 0; i < m.rows; ++i) m.vals[i] = calloc(m.cols, sizeof(double));

return m;

}

void zero\_matrix(matrix m)

{

int i, j;

for(i = 0; i < m.rows; ++i){

for(j = 0; j < m.cols; ++j) m.vals[i][j] = 0;

}

}

matrix csv\_to\_matrix(char \*filename, int header)

{

FILE \*fp = fopen(filename, "r");

if(!fp) error(filename);

matrix m;

m.cols = -1;

char \*line;

int n = 0;

int size = 1024;

m.vals = calloc(size, sizeof(double\*));

if(header) fgetl(fp);

while((line = fgetl(fp))){

if(m.cols == -1) m.cols = count\_fields(line);

if(n == size){

size \*= 2;

m.vals = realloc(m.vals, size\*sizeof(double\*));

}

m.vals[n] = parse\_fields(line, m.cols);

free(line);

++n;

}

m.vals = realloc(m.vals, n\*sizeof(double\*));

m.rows = n;

return m;

}

// Arguement parsing

void del\_arg(int argc, char \*\*argv, int index)

{

int i;

for(i = index; i < argc-1; ++i) argv[i] = argv[i+1];

argv[i] = 0;

}

int find\_arg(int argc, char\* argv[], char \*arg)

{

int i;

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

if(!argv[i]) continue;

if(0==strcmp(argv[i], arg)) {

del\_arg(argc, argv, i);

return 1;

}

}

return 0;

}

int find\_int\_arg(int argc, char \*\*argv, char \*arg, int def)

{

int i;

for(i = 0; i < argc-1; ++i){

if(!argv[i]) continue;

if(0==strcmp(argv[i], arg)){

def = atoi(argv[i+1]);

del\_arg(argc, argv, i);

del\_arg(argc, argv, i);

break;

}

}

return def;

}

float find\_float\_arg(int argc, char \*\*argv, char \*arg, float def)

{

int i;

for(i = 0; i < argc-1; ++i){

if(!argv[i]) continue;

if(0==strcmp(argv[i], arg)){

def = atof(argv[i+1]);

del\_arg(argc, argv, i);

del\_arg(argc, argv, i);

break;

}

}

return def;

}

char \*find\_char\_arg(int argc, char \*\*argv, char \*arg, char \*def)

{

int i;

for(i = 0; i < argc-1; ++i){

if(!argv[i]) continue;

if(0==strcmp(argv[i], arg)){

def = argv[i+1];

del\_arg(argc, argv, i);

del\_arg(argc, argv, i);

break;

}

}

return def;

}