#include "matrix.h"

#include "utils.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <assert.h>

#include <math.h>

void free\_matrix(matrix m)

{

int i;

for(i = 0; i < m.rows; ++i) free(m.vals[i]);

free(m.vals);

}

float matrix\_topk\_accuracy(matrix truth, matrix guess, int k)

{

int\* indexes = (int\*)xcalloc(k, sizeof(int));

int n = truth.cols;

int i,j;

int correct = 0;

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

top\_k(guess.vals[i], n, k, indexes);

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

int class\_id = indexes[j];

if(truth.vals[i][class\_id]){

++correct;

break;

}

}

}

free(indexes);

return (float)correct/truth.rows;

}

void scale\_matrix(matrix m, float scale)

{

int i,j;

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

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

m.vals[i][j] \*= scale;

}

}

}

matrix resize\_matrix(matrix m, int size)

{

int i;

if (m.rows == size) return m;

if (m.rows < size) {

m.vals = (float\*\*)xrealloc(m.vals, size \* sizeof(float\*));

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

m.vals[i] = (float\*)xcalloc(m.cols, sizeof(float));

}

} else if (m.rows > size) {

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

free(m.vals[i]);

}

m.vals = (float\*\*)xrealloc(m.vals, size \* sizeof(float\*));

}

m.rows = size;

return m;

}

void matrix\_add\_matrix(matrix from, matrix to)

{

assert(from.rows == to.rows && from.cols == to.cols);

int i,j;

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

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

to.vals[i][j] += from.vals[i][j];

}

}

}

matrix make\_matrix(int rows, int cols)

{

int i;

matrix m;

m.rows = rows;

m.cols = cols;

m.vals = (float\*\*)xcalloc(m.rows, sizeof(float\*));

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

m.vals[i] = (float\*)xcalloc(m.cols, sizeof(float));

}

return m;

}

matrix hold\_out\_matrix(matrix \*m, int n)

{

int i;

matrix h;

h.rows = n;

h.cols = m->cols;

h.vals = (float\*\*)xcalloc(h.rows, sizeof(float\*));

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

int index = rand()%m->rows;

h.vals[i] = m->vals[index];

m->vals[index] = m->vals[--(m->rows)];

}

return h;

}

float \*pop\_column(matrix \*m, int c)

{

float\* col = (float\*)xcalloc(m->rows, sizeof(float));

int i, j;

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

col[i] = m->vals[i][c];

for(j = c; j < m->cols-1; ++j){

m->vals[i][j] = m->vals[i][j+1];

}

}

--m->cols;

return col;

}

matrix csv\_to\_matrix(char \*filename)

{

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

if(!fp) file\_error(filename);

matrix m;

m.cols = -1;

char \*line;

int n = 0;

int size = 1024;

m.vals = (float\*\*)xcalloc(size, sizeof(float\*));

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

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

if(n == size){

size \*= 2;

m.vals = (float\*\*)xrealloc(m.vals, size \* sizeof(float\*));

}

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

free(line);

++n;

}

m.vals = (float\*\*)xrealloc(m.vals, n \* sizeof(float\*));

m.rows = n;

return m;

}

void matrix\_to\_csv(matrix m)

{

int i, j;

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

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

if(j > 0) printf(",");

printf("%.17g", m.vals[i][j]);

}

printf("\n");

}

}

void print\_matrix(matrix m)

{

int i, j;

printf("%d X %d Matrix:\n",m.rows, m.cols);

printf(" \_\_");

for(j = 0; j < 16\*m.cols-1; ++j) printf(" ");

printf("\_\_ \n");

printf("| ");

for(j = 0; j < 16\*m.cols-1; ++j) printf(" ");

printf(" |\n");

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

printf("| ");

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

printf("%15.7f ", m.vals[i][j]);

}

printf(" |\n");

}

printf("|\_\_");

for(j = 0; j < 16\*m.cols-1; ++j) printf(" ");

printf("\_\_|\n");

}

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

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

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

int \*sample(int n);

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

{

int j;

int best = 0;

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

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

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

if (new\_dist < best\_dist) {

best\_dist = new\_dist;

best = j;

}

}

return best;

}

float dist\_to\_closest\_center(float \*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)

{

matrix old\_centers = make\_matrix(centers.rows, centers.cols);

int i, j;

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

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

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

old\_centers.vals[i][j] = centers.vals[i][j];

centers.vals[i][j] = 0;

}

}

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];

}

}

}

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

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

if(centers.vals[i][j] == 0) centers.vals[i][j] = old\_centers.vals[i][j];

}

}

free(counts);

free\_matrix(old\_centers);

}

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);

}

int \*sample(int n)

{

int i;

int\* s = (int\*)xcalloc(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;

}

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

{

//printf(" x0 = %f, x1 = %f, y0 = %f, y1 = %f \n", x[0], x[1], y[0], y[1]);

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

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

float inter = mw\*mh;

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

float un = sum - inter;

float iou = inter / un;

return 1 - iou;

}

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

{

int i;

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

}

model do\_kmeans(matrix data, int k)

{

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

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

//smart\_centers(data, centers);

random\_centers(data, centers); // IoU = 67.31% after kmeans

/\*

// IoU = 63.29%, anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326

centers.vals[0][0] = 10; centers.vals[0][1] = 13;

centers.vals[1][0] = 16; centers.vals[1][1] = 30;

centers.vals[2][0] = 33; centers.vals[2][1] = 23;

centers.vals[3][0] = 30; centers.vals[3][1] = 61;

centers.vals[4][0] = 62; centers.vals[4][1] = 45;

centers.vals[5][0] = 59; centers.vals[5][1] = 119;

centers.vals[6][0] = 116; centers.vals[6][1] = 90;

centers.vals[7][0] = 156; centers.vals[7][1] = 198;

centers.vals[8][0] = 373; centers.vals[8][1] = 326;

\*/

// range centers [min - max] using exp graph or Pyth example

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

int i;

for(i = 0; i < 1000 && !kmeans\_expectation(data, assignments, centers); ++i) {

kmeans\_maximization(data, assignments, centers);

}

printf("\n iterations = %d \n", i);

model m;

m.assignments = assignments;

m.centers = centers;

return m;

}