**Feature extraction and classification.cpp**

#include<iostream>

#include<fstream>

#include<vector>

#include<string.h>

#include <iomanip>

#include<stdlib.h>

#include<math.h>

#include"inner.c"

#define MAXSZ 90 //max size of training set

#define K 7 // no of features

using namespace std;

int row, col;

float \*\*a;

float sum;

float mean;

float SD;

float sum1;

float sum2;

float variance;

floatmini,maxi;

int y;

floatx,z;

double M[10][10];

double g;

doublewmean;

float b[MAXSZ][K];

float X[K];

float total;

int flag=0;

voidreadff()

{

inti,j;

FILE \*ftp;

ftp = fopen("sample.txt","r");

for(i = 0; i < 7; i++)

for(j = 0; j < 7; j++)

fscanf(ftp, "%lf", &M[i][j]);

fscanf(ftp,"%lf",&wmean);

}

voidwriteff()

{

inti,j,row,col;

row=7;

col=7;

ofstreamfout("sampleinput.txt");

for(i = 0; i < row; i++)

{

for(j = 0; j < col; j++)

{

g=M[i][j]\*10;

fout<<g;

if(j != col - 1)

fout<<" ";

}

if(i != row - 1)

fout<<endl;

}

fout.close();

}

voiddisplayMatrix(float \*\*mat, int row, int col)

{

/\* Function to display a row x col matrix

Input:

mat: matrix to be displayed

row: number of rows

col: number of columns \*/

inti,j;

for(i = 0; i < row; i++)

{

for(j = 0; j < col; j++)

{

printf(" %f ",mat[i][j]);

}

printf("\n");

}

}

voiddisplayVector(float\* vec, int size)

{

/\* Function to display a one dimensional array

Input:

vec: vector to be displayed

size: number of elements in the vector \*/

int i;

for(i = 0; i < size; i++)

printf("%f ",vec[i]);

printf("\n");

}

voidreadFromFile()

{

/\* Function to read a matrix from a file

Input:

filename: path to the file which contains the matrix \*/

size\_tlen;

char line[200];

stringstr();

vector<string> numbers;

int r = 0,c = 0;

FILE \*fp = fopen("sampleinput.txt", "r");

if(!fp)

{

cout<<"Error\n";

return;

}

while(!feof(fp))

{

fgets(line,200,fp);

r++;

c = 0;

intstartpos = 0;

intendpos = 0;

for(int i = 0; line[i] != '\0'; i++)

{

if(line[i] == ' ')

{

c++;

stringtempstr("");

for(int j = startpos; j <= endpos; j++)

{

tempstr += line[j];

}

numbers.push\_back(tempstr);

startpos = endpos + 1;

endpos = startpos;

}

else

endpos++;

}

c++;

stringtempstr("");

for(int j = startpos; j <= endpos; j++)

{

tempstr += line[j];

}

numbers.push\_back(tempstr);

}

a = (float \*\*) malloc(r \* sizeof(float\*));

for(int i = 0; i < r; i++)

{

\*(a+i) = (float\*) malloc(c \* sizeof(float));

}

for(int i = 0; i < r; i++)

{

for(int j = 0; j < c; j++)

{

a[i][j] = atof(numbers[i\*c+j].c\_str());

}

}

row = r;

col = c;

}

voidwriteToFile(float \*\*a, int row, int col)

{

/\* Function to write a matrix to an output file

Input:

a: matrix to be written to the file

row: number of rows of the matrix

col: number of columns of the matrix

filename: path to the output file \*/

inti,j;

ofstreamfout("test.txt");

for (i = 0; i < row; i++)

{

for (j=0;j<col;j++)

{

sum = sum + a[i][j];

}

}

mean = (sum / (row\*col));

for (i = 0; i < row; i++)

{

for(j=0;j<col;j++)

{

sum1 = sum1 + pow((a[i][j] - mean), 2);

}

}

variance = sum1 / (float)(row\*col);

SD = sqrt(variance);

maxi=mini=0.0;

for( i = 0 ; i < row ; i++ )

{

for( j = 0 ; j < col ; j++ )

{

if ( a[i][i] > maxi )

maxi = a[i][j];

if ( a[i][i] < mini )

mini = a[i][j];

}

}

for(i=0;i<row;i++)

{

for(j=0;j<col-1;j++)

{

if(a[i][j+1] > 0)

{

x=a[i][j+1] - a[i][j];

sum2=sum2+fabsf(x);

}

}

}

z=sum2/(row\*col);

fout<<fixed <<setprecision(1)<< mean<<" "<<SD<<" "<<variance<<" "<<maxi<<" "<<mini<<" "<<z<<" "<<wmean<<";";

fout.close();

}

void PCA(float \*w, int col)

{

/\* Function to perform PCA

Input:

w: The vector of sigma values

col: number of elements in the vector \*/

int i;

float sum = 0, sum99Percent, currentSum = 0;

for(i = 0; i < col; i++)

{

sum += w[i];

}

sum99Percent = sum \* 99.0/100.0;

for(i = 0;i < col; i++)

{

if(currentSum>= sum99Percent)

w[i]=0;

currentSum += w[i];

}

}

voidswapColumns(float \*\*u, int i, int j, int row, int col)

{

/\* Function to swap two columns of a matrix

Input:

u: matrix whose columns are to be swapped

i: index of first column

j: index of second column

row: number of rows in the matrix

col: number of columns in the matrix \*/

int index;

for(index=0;index<row;index++)

{

float temp = u[index][i];

u[index][i] = u[index][j];

u[index][j] = temp;

}

}

voidsortSigmaValues(float \*\*u, float \*w, float \*\*v, int row, int col)

{

/\* Function to sort the sigma values (and correspondingly sort other matrices) in decreasing order

Input:

u, w, v : the resultant matrices from SVD

row: number of rows of u

col: number of columns of u

Uses:

- swapColumns() \*/

inti,j;

for(i = 0; i < col; i++)

{

for(j = i + 1; j < col; j++)

{

if(w[i] < w[j])

{

float temp = w[i];

w[i] = w[j];

w[j] = temp;

swapColumns(u, i, j, row, col);

swapColumns(v, i, j, col, col);

}

}

}

}

voidreadffk()

{

inti,j;

FILE \*ftp;

ftp = fopen("training.txt","r");

for(i = 0; i < MAXSZ; i++)

for(j = 0; j < K; j++)

fscanf(ftp, "%f", &b[i][j]);

fclose(ftp);

FILE \*ft;

ft = fopen("test.txt","r");

for(j = 0; j < K; j++)

fscanf(ftp, "%f", &X[j]);

fclose(ft);

}

int Classify(float y)

{

doubledd = 0;

intmaxn = 0;

intfreq[K];

intmfreq;

int i, j;

if(flag==1)

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

{

dd = Distance();

if(dd< MAXSZ) {

maxn = mfreq;

}

}

for(i = 0;i <K;i++)

freq[i] = 1;

for(i = 0;i <K;i++)

for( j = 0;j <K;j++)

if((i!=j))

freq[i]+=1;

for(i = 0;i <K;i++)

if(freq[i] >mfreq) {

mfreq = freq[i];

}

returnmaxn;

}

else

{

if(y<5000.00)

{

cout<<"Authentication Successful!!! USer name = Ajith"<<endl;

}

else

{

cout<<"Authentication Failed!!!"<<endl;

}

}

}

int main()

{

inti,j,k;

readff();

writeff();

// Initialize

readFromFile();

// Setting up RESULT

float \*\*res = (float\*\*) malloc(row \* sizeof(float\*));

for(i = 0; i < row; i++)

{

\*(res+i) = (float\*) malloc(col \* sizeof(float));

}

// Setting up V

float \*\*v = (float\*\*) malloc(col \* sizeof(float\*));

for(i=0;i<col;i++)

{

\*(v+i) = (float\*) malloc(col \* sizeof(float));

}

// Setting up W

float \*w = (float\*) malloc(col \* sizeof(float));

// Calling SVD

dsvd(a, row, col, w, v);

// Sort the sigma values in decreasing order

sortSigmaValues(a, w, v, row, col);

// Perform principal component analysis

PCA(w, col);

// Perform first multiplication u.v

for(i=0;i<row;i++)

{

for(j=0;j<col;j++)

{

float sum = 0;

for(k=0;k<col;k++)

{

if(k==j)

sum += a[i][k] \* w[k];

}

a[i][j]=sum;

}

}

// Perform second multiplication (uv).w

for(i=0;i<row;i++)

{

for(j=0;j<col;j++)

{

float sum = 0;

for(k=0;k<col;k++)

{

sum += a[i][k] \* v[j][k];

}

res[i][j]= sum;

}

}

writeToFile(res, row, col);

// Cleanup

for(i=0;i<row;i++)

{

free(a[i]);

free(res[i]);

}

for(i=0;i<col;i++)

{

free(v[i]);

}

free(a);

free(v);

free(w);

readffk();

}