东 莞 理 工 学 院

课程设计

课程　　运筹学与最优化方法

题目　　K-means For Image Clustering
图像聚类

院系名称 计算机与网络安全学院

班　级 2014信息与计算科学1、2班

学生姓名 郑楚锐、黄华锋、杨浩鑫

学　号 201441410104（郑楚锐）
201441410144（黄华锋）
201441410217（杨浩鑫）

小　组 第4小组

指导老师 徐欢乐

时　间 2017.07.09

目录

The table of contents is empty because you aren’t using the paragraph styles set to appear in it.

题目

K-means for image clustering. Dataset is available on THE MNIST DATABASE. In this project, you need to:

Step 1 : Combine both the training set and test set into one set and randomly pick 50000 images.

Step 2 : Use K-means to cluster the images you have chosen in Step 1 into 10 different classes and compute the error rate using the ground truth.

Step 3 : Apply Fuzz C-means method in C-means to cluster the images you have chosen in Step 1 into 10 different classes and compute the error rate using the ground truth.Compare the result with that of Step 2.

K-means用于图像聚类。数据集可在“MNIST数据库”上获得。在这个项目中，您需要：

1) 将训练集和测试集合组合成一组，随机选取50000张图像。
2) 使用K-means将您在步骤1中选择的图像聚类成10个不同的类，并计算错误率。
3) 在C-means中应用Fuzz C-means方法，将您在步骤1中选择的图像聚类成10个不同的类，并计算错误率。将结果与步骤2的结果进行比较。

1.数据处理

从http://yann.lecun.com/exdb/mnist/中THE MNIST DATABASE of handwritten digits数据库上下载相关数据集：

train-images-idx3-ubyte.gz: training set images
train-labels-idx1-ubyte.gz: training set labels
t10k-images-idx3-ubyte.gz: test set images
t10k-labels-idx1-ubyte.gz: test set labels

利用函数loadMNISTLabels()和loadMNISTImages()可读取相关图片及其特征标签，并存入相关定义的矩阵中，再分别对图片的矩阵以及标签的矩阵进行拼接，然后根据题目从中提取50000个相关元素并作相关处理（保存原先的标签值便于后面步骤的比较，然后重置标签值）。数据处理相关代码如下：

```
function [trainImage, truth] = data_processing(n)
%n代表随机选取样本的数量
%predict为预测结果
%truth为实际结果
%trainImage为处理完成后的数据矩阵

%读取图片数据
image1 = loadMNISTImages('t10k-images.idx3-ubyte');
image2 = loadMNISTImages('train-images.idx3-ubyte');
%图片数据矩阵的拼接
image=[image1;image2];

%读取标签数据
label1 = loadMNISTLabels('t10k-labels.idx1-ubyte');
label2 = loadMNISTLabels('train-labels.idx1-ubyte');
%标签数据矩阵的拼接
label = [label1;label2];

%拼接图片数据矩阵以及标签数据矩阵,即将训练集和测试集数据合并
data = [image,label];

%选取n个随机样本
trainImage = data(randperm(70000,n),:);
%提取相应的标签值
truth = trainImage(:,785);
```

```
%处理矩阵
trainImage(:,785) = [];
```

函数**loadMNISTLabels()**代码如下：

```
function labels = loadMNISTLabels(filename)
%loadMNISTLabels returns a [number of MNIST images]x1 matrix
containing
%the labels for the MNIST images

fp = fopen(filename, 'rb');
assert(fp ~= -1, ['Could not open ', filename, '']);

magic = fread(fp, 1, 'int32', 0, 'ieee-be');
assert(magic == 2049, ['Bad magic number in ', filename, '']);

numLabels = fread(fp, 1, 'int32', 0, 'ieee-be');

labels = fread(fp, inf, 'unsigned char');

assert(size(labels,1) == numLabels, 'Mismatch in label count');

fclose(fp);

end
```

函数**loadMNISTImages()**的代码如下：

```
function images = loadMNISTImages(filename)
%loadMNISTImages returns a 28x28x[number of MNIST images] matrix
containing
%the raw MNIST images

fp = fopen(filename, 'rb');
assert(fp ~= -1, ['Could not open ', filename, '']);

magic = fread(fp, 1, 'int32', 0, 'ieee-be');
assert(magic == 2051, ['Bad magic number in ', filename, '']);

numImages = fread(fp, 1, 'int32', 0, 'ieee-be');
numRows = fread(fp, 1, 'int32', 0, 'ieee-be');
numCols = fread(fp, 1, 'int32', 0, 'ieee-be');

images = fread(fp, inf, 'unsigned char');
images = reshape(images, numCols, numRows, numImages);
images = permute(images,[2 1 3]);

fclose(fp);

% Reshape to #pixels x #examples
images = reshape(images, size(images, 1) * size(images, 2),
size(images, 3));
% Convert to double and rescale to [0,1]
images = double(images) / 255;
images = images';

end
```

2.基本模型

K-means算法是最为经典的基于距离的聚类方法，是十大经典数据挖掘算法之一。K-means算法的基本思想是：以空间中k个点为中心进行聚类，对最靠近他们的对象归类。通过迭代的方法，逐次更新各聚类中心的值，直至得到最好的聚类结果。

K-means 算法接受参数 k ；然后将事先输入的n个数据对象划分为 k个聚类以便使得所获得的聚类满足：同一聚类中的对象相似度较高；而不同聚类中的对象相似度较小。聚类相似度是利用各聚类中对象的均值所获得一个“中心对象”（引力中心）来进行计算的。

假设要把样本集分为c个类别，算法描述如下：

（1）适当选择c个类的初始中心；

（2）在第k次迭代中，对任意一个样本，求其到c个中心的距离，将该样本归到距离最短的中心所在的类；

（3）利用均值等方法更新该类的中心值；

（4）对于所有的c个聚类中心，如果利用（2）（3）的迭代法更新后，值保持不变，则迭代结束，否则继续迭代。

该算法的最大优势在于简洁和快速。算法的关键在于初始中心的选择和距离公式。

算法流程

首先从n个数据对象任意选择 k 个对象作为初始聚类中心；而对于所剩下其它对象，则根据它们与这些聚类中心的相似度（距离），分别将它们分配给与其最相似的（聚类中心所代表的）聚类；

然后再计算每个所获新聚类的聚类中心（该聚类中所有对象的均值）；

不断重复这一过程直到标准测度函数开始收敛为止。

一般都采用均方差作为标准测度函数. k个聚类具有以下特点：各聚类本身尽可能的紧凑，而各聚类之间尽可能的分开。

算法停止条件

一般是目标函数达到最优或者达到最大的迭代次数即可终止。对于不同的距离度量，目标函数往往不同。当采用欧式距离时，目标函数一般为最小化对象到其簇质心的距离的平方和，如下：

$$\min \sum_{i=1}^{K} \sum_{x \in C_i} dist\left(c_i, x\right)^2$$

当采用余弦相似度时，目标函数一般为最大化对象到其簇质心的余弦相似度和，如下：

$$\max \sum_{i=1}^{K} \sum_{x \in C_i} \cos ine\left(c_i, x\right)$$

K-means算法的工作原理及流程

输入：聚类个数k，以及包含 n个数据对象的数据库。

输出：满足方差最小标准的k个聚类。

具体流程为：

输入：k, data[n];

（1） 选择k个初始中心点，例如c[0]=data[0],…c[k-1]=data[k-1];

（2） 对于data[0]….data[n], 分别与c[0]…c[k-1]比较，假定与c[i]差值最少，就标记为i;

（3） 对于所有标记为i点，重新计算c[i]={ 所有标记为i的data[j]之和}/标记为i的个数；

（4） 重复(2)(3),直到所有c[i]值的变化小于给定阈值。

K-means算法是将样本聚类成k个簇（cluster），具体算法描述如下：

1、 随机选取k个聚类质心点（cluster centroids）为$\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n$。

2、 重复下面过程直到收敛 {

对于每一个样例i，计算其应该属于的类

$$c^{(i)} := \arg\min_j ||x^{(i)} - \mu_j||^2.$$

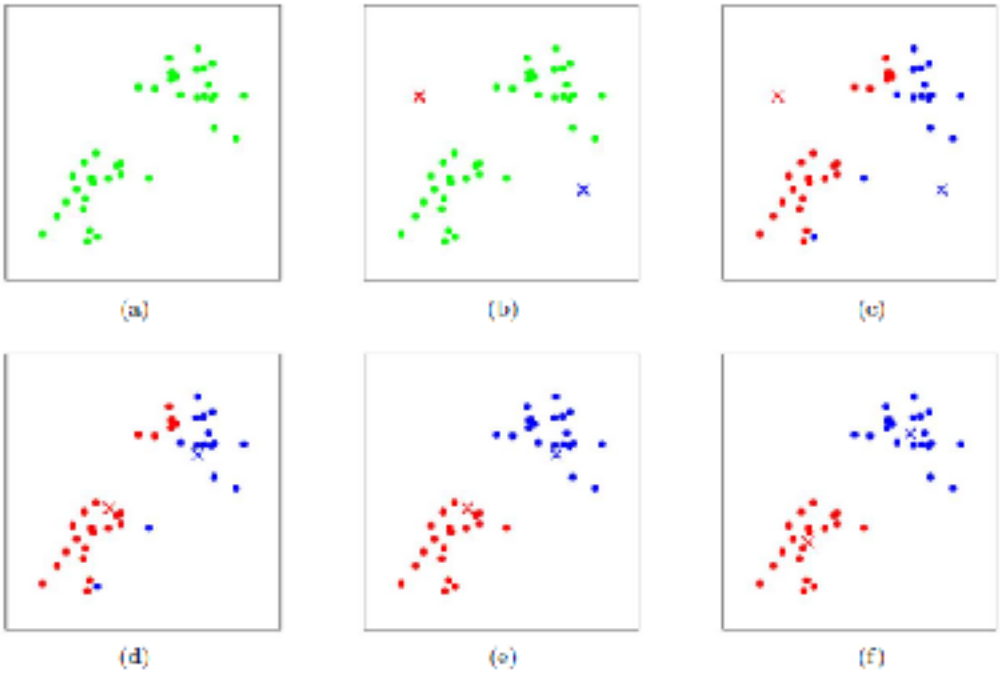
对于每一个类j，重新计算该类的质心

$$\mu_j := \frac{\sum_{i=1}^{m} 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)} = j\}}.$$

}

K是我们事先给定的聚类数，$c^{(i)}$代表样例i与k个类中距离最近的那个类，$c^{(i)}$的值是1到k中的一个。质心μ_j代表我们对属于同一个类的样本中心点的猜测，拿星团模型来解释就是要将所有的星星聚成k个星团，首先随机选取k个宇宙中的点（或者k个星星）作为k个星团的质心，然后第一步对于每一个星星计算其到k个质心中每一个的距离，然后选取距离最近的那个星团作为$c^{(i)}$，这样经过第一步每一个星星都有了所属的星团；第二步对于每一个星团，重新计算它的质心μ_j（对里面所有的星星坐标求平均）。重复迭代第一步和第二步直到质心不变或者变化很小。

下图展示了对n个样本点进行K-means聚类的效果，这里k取2。



3.加强版模型

Fuzz C-means算法是基于对目标函数的优化基础上的一种数据聚类方法。聚类结果是每一个数据点对聚类中心的隶属程度，该隶属程度用一个数值来表示。FCM算法是一种无监督的模糊聚类方法，在算法实现过程中不需要人为的干预。

FCM算法是一种摹于划分的聚类算法，它的思想就是使得被划分到同一簇的对象之间相似度最大，而不同簇之间的相似度最小。模糊 c 均值算法是普通 c 均值算法的改进，普通 C 均值算法对于数据的划分是硬性的，而FCM则是一种柔性的模糊划分。FCM算法需要两个参数一个是聚类数目 C，另一个是参数 m。一般来讲 c 要远远小于聚类样本的总个数，同时要保证 C ＞ 1 。对于 m，它是一个控制算法的柔性的参数，如果 m 过大，则聚类效果会很次，而如果 m 过小则算法会接近HCM聚类算法。算法的输出是 C 个聚类中心点向量和 C * N 的一个模糊划分矩阵，这个矩阵表示的是每个样本点属于每个类的隶属度。根据这个划分矩阵按照模糊集合中的最大隶属原则就能够确定每个样本点归为哪个类。聚类中心表示的是每个类的平均特征，可以认为是这个类的代表点，FCM算法对于满足正态分布的数据聚类效果会很好。

FCM是基于以下目标函数的最小化。

目标函数J如下：

$$J = \sum_{i=1}^{K} J_i = \sum_{i=1}^{K} \left(\sum_{j=1}^{N} w_{ji}^{m} \left\| X_j - C_i \right\|^2 \right)$$

其中：

—Xj为数据点

—Ci为聚类中心点

—N为数据个数

—K为聚类中心点个数

—m为权重指数

C-means实现步骤

1.设定分类个数k，设定初始权重矩阵，随机给定0~1之值，并满足权重总和为1如下式：

$$\sum_{i=1}^{K} w_{ji} = 1, \forall w_{ji} \in [0,1], j = 1,.....N, 0 < \sum_{j=1}^{N} w_{ji} < N$$ （2）

2.如下式计算聚类中心点：

$$C_i = \frac{\sum_{j=1}^{N} w_{ji}^{m} X_j}{\sum_{j=1}^{N} w_{ji}^{m}}$$ （3）

3. 由（1）式计算目标函数值，当目标函数值小于设定的容忍误差可结束迭代过程，否则执行步骤4。

$$E(t) = \left\| J^{(t)} - J^{(t-1)} \right\| < \varepsilon$$

4.重新计算权重W如下式，并回到步骤2进行运算

$$w_{ji} = \frac{1}{\sum_{s=1}^{K} \left(\frac{\left\| X_j - C_i \right\|}{\left\| X_j - C_s \right\|} \right)^{\frac{2}{m-1}}}$$

Fuzzy C-Means实现步骤

1.设定聚类数目K，最大执行步骤tmax，一个很小的容忍误差ε>0

2.决定聚类中心起始位置Cj(0)，0＜j≤K

3.for t=1,……,tmax

（A）for j=1,……,N，计算隶属度矩阵wji(t)= $\frac{1}{\sum_{z=1}^{K} \left(\frac{\left\| X_j - C_i^{(t)} \right\|}{\left\| X_j - C_z^{(t)} \right\|} \right)^{\frac{2}{m-1}}}$

（B）for i=1,……,K，更新聚类中心点

$$C_i^{(t)} = \frac{\sum_{j=1}^{N} w_{ji}^{m(t)} X_j}{\sum_{j=1}^{N} w_{ji}^{m(t)}}$$

（C）计算收敛准则，若 $E(t) = \left\| J^{(t)} - J^{(t-1)} \right\| < \varepsilon$ 成立，则停止运算，否则进行下一轮迭代 $E(t) = \left\| C^{(t)} - C^{(t-1)} \right\| < \varepsilon$ 。

4.模型的比较和分析

K均值聚类算法即是HCM（普通硬-C均值聚类算法），它是一种硬性划分的方法，结果要么是1要么是0，没有其他情况，具有"非此即彼"的性质。里面的隶属度矩阵是U。FCM是把HCM算法推广到模糊情形，用在模糊性的分类问题上，给了隶属度一个权重。隶属度矩阵用U的m次方表示。

可以简单理解为：K的算法先初始化聚类中心，后确定隶属矩阵，而FCM下好相反，利用随机数初始化隶属矩阵，后再求中心。

K-means实现思路：
1. 随机从数据集中选择k个中心点
2. 对于数据集中的每一条数据，比较与k个中心点的距离，然后把当前数据分配给最近的中心点
3. 重新计算数据集中k个中心点
4. 重复第二步和第三步，只到中心点的位置变化不变或者变化在指定的范围内或者迭代次数满足设定的目标。
比较与中心点的距离一般采用的是欧式距离，k-means可能会选择到最差的结果，需要多跑几次，而且每次跑的结果不一定相同。

FCM实现思路：
1. 指定中心点的个数k并且随机的为每个数据点分配中心点的概率
2. 重新计算中心点
3. 更新每个数据点属于中心点的根据
4. 重复2.3直到样本点属于中心点的相关系数不变或者变化在一个可控的范围内或者迭代次数到达了设定的值。

K-means算法与**Fuzz C-means**算法的优缺点分析：

K-Means聚类算法的优点主要集中在:
1.算法快速、简单;
2.对大数据集有较高的效率并且是可伸缩性的;
3.时间复杂度近于线性，而且适合挖掘大规模数据集。K-Means聚类算法的时间复杂度是O(nkt) ,其中n代表数据集中对象的数量，t代表着算法迭代的次数，k代表着簇的数目。

K-Means聚类算法的缺点主要体现在:
在 K-means 算法中 K 是事先给定的，这个 K 值的选定是非常难以估计的。很多时候，事先并不知道给定的数据集应该分成多少个类别才最合适。这也是 K-means 算法的一个不足。有的算法是通过类的自动合并和分裂，得到较为合理的类型数目 K，例如 ISODATA 算法。关于 K-means 算法中聚类数目K 值的确定在文献中，是根据方差分析理论，应用混合 F 统计量来确定最佳分类数，并应用了模糊划分熵来验证最佳分类数的正确性。在文献中，使用了一种结合全协方差矩阵的 RPCL 算法，并逐步删除那些只包含少量训练数据的类。而文献中使用的是一种称为次胜者受罚的 竞争学习规则，来自动决定类的适当数目。它的思想是：对每个输入而言，不仅竞争获胜单元的权值被修正以适应输入值，而且对次胜单元采用惩罚的方法使之远离输入值。

在 K-means 算法中，首先需要根据初始聚类中心来确定一个初始划分，然后对初始划分进行优化。这个初始聚类中心的选择对聚类结果有较大的影响，一旦初始值选择的不好，可能无法得到有效的聚类结果，这也成为 K-means算法的一个主要问题。对于该问题的解决，许多算法采用 遗传算法（GA），例如文献 中采用遗传算法（GA）进行初始化，以内部聚类准则作为评价指标。

从 K-means 算法框架可以看出，该算法需要不断地进行样本分类调整，不断地计算调整后的新的聚类中心，因此当数据量非常大时，算法的时间开销是非常大的。所以需要对算法的时间复杂度进行分析、改进，提高算法应用范围。在文献中从该算法的时间复杂度进行分析考虑，通过一定的相似性准则来去掉聚类中心的侯选集。而在文献中，使用的 K-means 算法是对样本数据进行聚类，无论是初始点的选择还是一次迭代完成时对数据的调整，都是建立在随机选取的样本数据的基础之上，这样可以提高算法的收敛速度。

FCM对初始聚类中心敏感，需要人为确定聚类数，容易陷入局部最优解；

Fuzz C-means算法的不足之处:首先，算法中需要设定一些参数，若参数的初始化选取的不合适，可能影响聚类结果的正确性;其次，当数据样本集合较大并且特征数目较多时，算法的实时性不太好。

相关代码：

（注：由于函数运行中存在随机数，结果有一定随机性，总结部分有一定说明）

计算错误率的函数：

```
function [errorScore, errorCount] =
compute_error_rate(truth,predict,n)
%该函数用于计算错误率，然后通过1-错误率得到准确率
%predict为预测结果
%truth为实际结果
%n代表分类类数
data = [truth,predict];
errorCount = 0;
for i=0:(n-1)
    a = data(data(:,1)==i,:);
    modeNum = mode(a(:,2));
    modeCount = sum(a(:,2)==modeNum);
    count = length(a);
    errorCount = errorCount + (count-modeCount);
end
errorScore = errorCount/length(data);
```

K-means函数：

```
function [labels] = kmeans_clustering( data, k )
%k代表分类类数
%predict是该算法运行出来的分类结果
[num,~]=size(data);
ind = randperm(num);
ind = ind(1:k);
centers = data(ind,:);
d=inf;
labels = nan(num,1);
while d>0
    labels0 = labels;
    dist = pdist2(data, centers);
    [~,labels] = min(dist,[],2);
    d= sum(labels0 ~= labels);
    for i=1:k
        centers(i,:)=mean(data(labels == i,:),1);
    end
end
labels = labels-1;
end
```

对应的运行代码：

```
clear;
n = 50000;
[trainImage, truth] = data_processing(n);
[predict] = kmeans_clustering(trainImage, 10);
[errorScore, errorCount] = compute_error_rate(truth,predict,10);
```

运行结果：

错误率为0.3874，即正确率为0.6126.

多次检验得到最好的结果如下：

错误率为0.3763，即正确率为0.6237.

C-means函数：

MATLAB系统Fuzzy C-means Clustering

【功能描述】

模糊C均值聚类算法，可将输入的数据集data聚为指定的cluster_n类

【函数描述】

语法格式

[center, U, obj_fcn] = FCM(data, cluster_n, options)

用法：

1. [center,U,obj_fcn] = FCM(Data,N_cluster,options);
2. [center,U,obj_fcn] = FCM(Data,N_cluster);

输入变量

data ---- n*m矩阵,表示n个样本,每个样本具有m维特征值

cluster_n ---- 标量,表示聚合中心数目,即类别数

options ---- 4*1列向量，其中

options(1): 隶属度矩阵U的指数，>1(缺省值: 2.0)

options(2): 最大迭代次数(缺省值: 100)

options(3): 隶属度最小变化量,迭代终止条件(缺省值: 1e-5)

options(4): 每次迭代是否输出信息标志(缺省值: 0)

输出变量

center ---- 聚类中心

U ---- 隶属度矩阵

obj_fcn ---- 目标函数值

```
function [center, U, obj_fcn] = FCM(data, cluster_n, options)
% 采用FCM(模糊C均值)算法将数据集data聚为cluster_n类
```

```
% MATLAB自带的FCM算法整合整理+注释详解整理
% by faruto @ faruto's Studio http://blog.sina.com.cn/faruto
% Email:patrick.lee@foxmail.com QQ:516667408
% http://www.matlabsky.com http://www.mfun.la
% last modified 2010.08.21
% 用法：
% 1. [center,U,obj_fcn] = FCM(Data,N_cluster,options);
% 2. [center,U,obj_fcn] = FCM(Data,N_cluster);
%
% 输入：
% data ---- n*m矩阵,表示n个样本,每个样本具有m维特征值
% cluster_n ---- 标量,表示聚合中心数目,即类别数
% options ---- 4*1列向量，其中
% options(1): 隶属度矩阵U的指数，>1(缺省值: 2.0)
% options(2): 最大迭代次数(缺省值: 100)
% options(3): 隶属度最小变化量,迭代终止条件(缺省值: 1e-5)
% options(4): 每次迭代是否输出信息标志(缺省值: 0)
% 输出：
% center ---- 聚类中心
% U ---- 隶属度矩阵
% obj_fcn ---- 目标函数值
% Example:
% data = rand(100,2);
% options = [2;100;1e-5;1];
% [center,U,obj_fcn] = FCM(data,2,options);
% figure;
% plot(data(:,1), data(:,2),'o');
% title('DemoTest of FCM Cluster');
% xlabel('1st Dimension');
% ylabel('2nd Dimension');
% grid on;
% hold on;
% maxU = max(U);
% index1 = find(U(1,:) == maxU);
% index2 = find(U(2,:) == maxU);
% line(data(index1,1),data(index1,2),'marker','*','color','g');
% line(data(index2,1),data(index2,2),'marker','*','color','r');
% plot([center([1 2],1)],[center([1 2],2)],'*','color','k')
% hold off;
%% 初始化initialization

% 输入参数数量检测
if nargin ~= 2 && nargin ~= 3 %判断输入参数个数只能是2个或3个
    error('Too many or too few input arguments!');
end

data_n = size(data, 1); % 求出data的第一维(rows)数,即样本个数
data_m = size(data, 2); % 求出data的第二维(columns)数，即特征属性个数
% 设置默认操作参数
default_options = ...
[2; % 隶属度矩阵U的指数
100; % 最大迭代次数
1e-5; % 隶属度最小变化量,迭代终止条件
0]; % 每次迭代是否输出信息标志
if nargin == 2
% 如果输入参数个数是二那么就调用默认的option
    options = default_options;
else
% 如果用户给的opition数少于4个那么其他用默认值
    if length(options) < 4
        tmp = default_options;
```

```
        tmp(1:length(options)) = options;
        options = tmp;
    end
% 检测options中是否有nan值
    nan_index = find(isnan(options)==1);
% 将denfault_options中对应位置的参数赋值给options中不是数的位置.
    options(nan_index) = default_options(nan_index);
% 如果模糊矩阵的指数小于等于1，给出报错
    if options(1) <= 1,
        error('The exponent should be greater than 1!');
    end
end
% 将options中的分量分别赋值给四个变量
expo = options(1); % 隶属度矩阵U的指数
max_iter = options(2); % 最大迭代次数
min_impro = options(3); % 隶属度最小变化量,迭代终止条件
display = options(4); % 每次迭代是否输出信息标志

obj_fcn = zeros(max_iter, 1); % 初始化输出参数obj_fcn
U = initfcm(cluster_n, data_n); % 初始化模糊分配矩阵,使U满足列上相加为1
%% Main loop 主要循环
for i = 1:max_iter
% 在第k步循环中改变聚类中心ceneter,和分配函数U的隶属度值;
    [U, center, obj_fcn(i)] = stepfcm(data, U, cluster_n, expo);
    if display,
        fprintf('FCM:Iteration count = %d, obj.fcn = %f\n', i, obj_fcn(i));
    end
% 终止条件判别
    if i > 1 && abs(obj_fcn(i) - obj_fcn(i-1)) <= min_impro
        break;
    end
end
iter_n = i; % 实际迭代次数
obj_fcn(iter_n+1:max_iter) = [];

%% initfcm子函数
function U = initfcm(cluster_n, data_n)
% 初始化fcm的隶属度函数矩阵
% 输入:
% cluster_n ---- 聚类中心个数
% data_n ---- 样本点数
% 输出：
% U ---- 初始化的隶属度矩阵
U = rand(cluster_n, data_n);
col_sum = sum(U);
U = U./col_sum(ones(cluster_n, 1), :);
%% stepfcm子函数
function [U_new, center, obj_fcn] = stepfcm(data, U, cluster_n, expo)
% 模糊C均值聚类时迭代的一步
% 输入：
% data ---- n*m矩阵,表示n个样本,每个样本具有m维特征值
% U ---- 隶属度矩阵
% cluster_n ---- 标量,表示聚合中心数目,即类别数
% expo ---- 隶属度矩阵U的指数
% 输出：
% U_new ---- 迭代计算出的新的隶属度矩阵
% center ---- 迭代计算出的新的聚类中心
% obj_fcn ---- 目标函数值
mf = U.^expo; % 隶属度矩阵进行指数运算结果
center = mf*data./((ones(size(data, 2), 1)*sum(mf'))'); % 新聚类中心
```

```
dist = distfcm(center, data); % 计算距离矩阵
obj_fcn = sum(sum((dist.^2).*mf)); % 计算目标函数值
tmp = dist.^(-2/(expo-1));
U_new = tmp./(ones(cluster_n, 1)*sum(tmp)); % 计算新的隶属度矩阵
%% distfcm子函数
function out = distfcm(center, data)
% 计算样本点距离聚类中心的距离
% 输入：
% center ---- 聚类中心
% data ---- 样本点
% 输出：
% out ---- 距离
out = zeros(size(center, 1), size(data, 1));
for k = 1:size(center, 1) % 对每一个聚类中心
% 每一次循环求得所有样本点到一个聚类中心的距离
      out(k, :) = sqrt(sum(((data-ones(size(data,1),1)*center(k,:)).^2)',1));
end
```

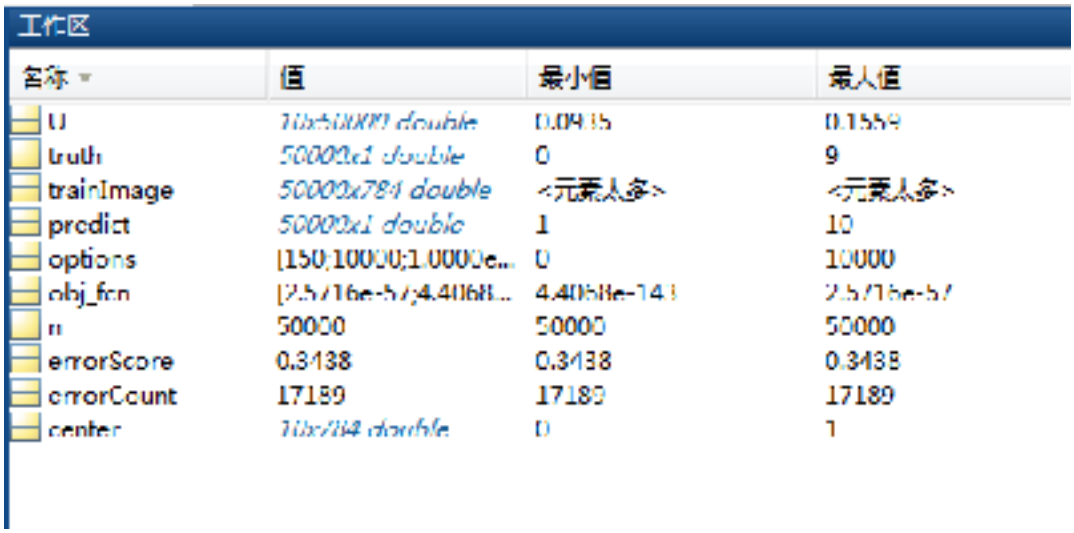
对应的运行代码：

```
clear;
n = 50000;
[trainImage, truth] = data_processing(n);
options = [150;10000;1e-5;0];
[center,U,obj_fcn] = fcm(trainImage, 10,options);
%找到所属的类并转秩,得出分类结果
[~,predict] = max(U);
predict = predict';
[errorScore, errorCount] = compute_error_rate(truth,predict,10);
```

运行结果：

错误率为0.4483，即正确率为0.5517.

运行结果：



错误率为0.3438，即正确率为0.6562.

多次检验得到最好的结果如下：

错误率为0.1188，即正确率为0.8812.

参考编写**C-means**函数：

```
function [center, U, obj_fcn] = FCM(data, cluster_n, options)
% 采用FCM(模糊C均值)算法将数据集data聚为cluster_n类
% MATLAB自带的FCM算法整合整理+注释详解整理
% by faruto @ faruto's Studio http://blog.sina.com.cn/faruto
% Email:patrick.lee@foxmail.com QQ:516667408
% http://www.matlabsky.com http://www.mfun.la
% last modified 2010.08.21
% 用法：
% 1. [center,U,obj_fcn] = FCM(Data,N_cluster,options);
% 2. [center,U,obj_fcn] = FCM(Data,N_cluster);
%
% 输入：
% data ---- n*m矩阵,表示n个样本,每个样本具有m维特征值
% cluster_n ---- 标量,表示聚合中心数目,即类别数
% options ---- 4*1列向量，其中
% options(1)：隶属度矩阵U的指数，>1(缺省值：2.0)
% options(2)：最大迭代次数(缺省值：100)
% options(3)：隶属度最小变化量,迭代终止条件(缺省值：1e-5)
% options(4)：每次迭代是否输出信息标志(缺省值：0)
% 输出：
```

```
% center ---- 聚类中心
% U ---- 隶属度矩阵
% obj_fcn ---- 目标函数值
% Example:
% data = rand(100,2);
% options = [2;100;1e-5;1];
% [center,U,obj_fcn] = FCM(data,2,options);
% figure;
% plot(data(:,1), data(:,2),'o');
% title('DemoTest of FCM Cluster');
% xlabel('1st Dimension');
% ylabel('2nd Dimension');
% grid on;
% hold on;
% maxU = max(U);
% index1 = find(U(1,:) == maxU);
% index2 = find(U(2,:) == maxU);
% line(data(index1,1),data(index1,2),'marker','*','color','g');
% line(data(index2,1),data(index2,2),'marker','*','color','r');
% plot([center([1 2],1)],[center([1 2],2)],'*','color','k')
% hold off;
%% 初始化initialization

% 输入参数数量检测
if nargin ~= 2 && nargin ~= 3 %判断输入参数个数只能是2个或3个
       error('Too many or too few input arguments!');
end

data_n = size(data, 1); % 求出data的第一维(rows)数,即样本个数
data_m = size(data, 2); % 求出data的第二维(columns)数，即特征属性个数
% 设置默认操作参数
default_options = ...
[2; % 隶属度矩阵U的指数
100; % 最大迭代次数
1e-5; % 隶属度最小变化量,迭代终止条件
0]; % 每次迭代是否输出信息标志
if nargin == 2
% 如果输入参数个数是二那么就调用默认的option
    options = default_options;
else
% 如果用户给的opition数少于4个那么其他用默认值
if length(options) < 4
    tmp = default_options;
    tmp(1:length(options)) = options;
    options = tmp;
end
% 检测options中是否有nan值
nan_index = find(isnan(options)==1);
% 将denfault_options中对应位置的参数赋值给options中不是数的位置.
options(nan_index) = default_options(nan_index);
% 如果模糊矩阵的指数小于等于1，给出报错
if options(1) <= 1,
    error('The exponent should be greater than 1!');
end
end
% 将options中的分量分别赋值给四个变量
expo = options(1); % 隶属度矩阵U的指数
max_iter = options(2); % 最大迭代次数
min_impro = options(3); % 隶属度最小变化量,迭代终止条件
display = options(4); % 每次迭代是否输出信息标志
```

```
obj_fcn = zeros(max_iter, 1); % 初始化输出参数obj_fcn
U = initfcm(cluster_n, data_n); % 初始化模糊分配矩阵,使U满足列上相加为1
%% Main loop 主要循环
for i = 1:max_iter
% 在第k步循环中改变聚类中心ceneter,和分配函数U的隶属度值;
    [U, center, obj_fcn(i)] = stepfcm(data, U, cluster_n, expo);
if display,
    fprintf('FCM:Iteration count = %d, obj.fcn = %f\n', i,
obj_fcn(i));
end
% 终止条件判别
if i > 1 && abs(obj_fcn(i) - obj_fcn(i-1)) <= min_impro
    break;
end
end
iter_n = i; % 实际迭代次数
obj_fcn(iter_n+1:max_iter) = [];
```

对应的运行代码：

```
clear;
n = 50000;
[trainImage, truth] = data_processing(n);
options = [150;10000;1e-5;0];
[center, U, obj_fcn] = FCM(trainImage,10,options);
%找到所属的类并转秩,得出分类结果
[~,predict] = max(U);
predict = predict';
[errorScore, errorCount] = compute_error_rate(truth,predict,10);
```

运行结果：

错误率为0.3181，即正确率为0.6819.

名称	值	最小值	最大值
U	10x50000 double	0.0938	0.1543
truth	50000x1 double	0	9
trainImage	50000x784 double	<元素太多>	<元素太多>
predict	50000x1 double	1	10
options	[150;10000;1.0000e...	0	10000
obj fcn	[1.6692e-57;4.4720...	4.4720e-143	1.6692e-57
n	50000	50000	50000
errorScore	0.2498	0.2498	0.2498
errorCount	12492	12492	12492
center	10x784 double	0	1

错误率为0.2498，即正确率为0.7502.

名称	值	最小值	最大值
U	10x50000 double	0.0911	0.1485
truth	50000x1 double	0	9
trainImage	50000x784 double	<元素太多>	<元素太多>
predict	50000x1 double	1	10
options	[150;10000;1.0000e...	0	10000
obj fcn	[6.9457e-55;4.6153...	4.6153e-143	6.9457e-55
n	50000	50000	50000
errorScore	0.3432	0.3432	0.3432
errorCount	17159	17159	17159
center	10x784 double	0	1.0000

错误率为0.3432，即正确率为0.6568.

关于聚类算法**K-means**和**FCM**的小结

K-means与FCM都是经典的聚类算法，K-means是排他性聚类算法，即一个数据点只能属于一个类别，而FCM只计算数据点与各个类别的相似度。可理解为：对任一个数据点，使用K-means算法，其属于某个类别的相似度要么100%要么0%（非是即否）；而对于FCM算法，其属于某个类别的相似度只是一个百分比。

两个算法的思想都可归结为求一个相似度对象函数的最小值。设X是一个有n个元素的d维数据集合：$X=\{x_i \mid x_i=(x_{i1},x_{i2},\cdots,x_{id}),1\le i\le n,i\in N^+,d\in N^+\}$，将其聚为k类（或C类）。

K-means对象函数的数学模型为：$J=\sum_{j=1}^{k}\sum_{i=1}^{n}\left\|x_i^{(j)}-c_j\right\|^2$。其几何含义是设定k个子集$C_j$：$C_j\in X,1\le j\le k$，每个子集$C_j$都有一个质心$c_j$，求得集合$C_j$中所有点$x_i^{(j)}$到质心$c_j$的距离之和，所有这些距离之和就是J。每一次迭代就是计算子集C_j的质心c_j，以及哪些数据点$x_i^{(j)}$归属到C_j中，迭代结束的依据就是

$x_i^{(j)}$ 的归属不再发生变化。其中 $\|x_i^{(j)}-c_j\|^2$ 是选定的一种距离描述算法，用于描述某数据点 $x_i^{(j)}$ 到质心 c_j 的距离，一般选用闵式距离。

FCM对象函数的数学模型为：$J_m=\sum_{i=1}^{n}\sum_{j=1}^{c}u_{ij}^m\|x_i-c_j\|^2, 1\le m<\infty$。其数学含义是设定C个子集 C_j：$C_j\in X, 1\le j\le C$，c_j 是 C_j 的质心，集合X中每一个数据点 x_i 归属到子集 C_j 的概率是 u_{ij}，m是任意大于等于1的实数的权重指数（m取值的研究这里不涉及），$\|x_i-c_j\|^2$ 可以是任意表示数据点 x_i 与子集 C_j 的相似度算法，算法迭代的过程就是求得归属概率 u_{ij} 以及质心 c_j，迭代终止的依据是需要设定一个终止值 ε，使得 $\max_{ij}\{|u_{ij}^{k+1}-u_{ij}^{k}|\}<\varepsilon, 1\le i\le n, 1\le j\le C$，即第k轮与第k+1轮迭代求得的 u_{ij} 差距的最大值要小于 ε，其几何含义将使得 J_m 最小或位于马鞍形的位置。

质心 c_j 的算法公式：$c_j=\frac{\sum_{i=1}^{n}u_{ij}^m\times x_i}{\sum_{i=1}^{n}u_{ij}^m}$，即对于某一类别j，计算所有数据节点 x_i 与其归属概率 u_{ij}（加权）的乘积之和再和归属概率（加权）之和的比值。

归属概率 u_{ij} 的算法公式：$u_{ij}=\frac{1}{\sum_{k=1}^{c}\left(\frac{\|x_i-c_j\|}{\|x_i-c_k\|}\right)^{\frac{2}{m-1}}}$，即数据点 x_i 与质心 c_j 的相似度（加权）和与 x_i 其他类别质心的相似度（加权）的比值之和。

u_{ij} 的性质有：1）$u_{ij}\in[0,1], \forall i,j$，概率可能是0~1之间的任意实数；2）

$\sum_{j=1}^{c} u_{ij} = 1, \forall i$，对于任意数据点，其归属到所有类别的概率之和为1；3）$0 < \sum_{i=1}^{n} u_{ij} < n, \forall n, j$，对于任意类别，任意数据数目，所有数据点归属于该类别的概率之和小于数据点的总数，但总是大于0。

将K-means与FCM算法使用矩阵表示，性质如下：

$$
\begin{array}{c}
X \\
x_1 \\
x_2 \\
x_i \\
\cdots \\
x_n \\
c
\end{array}
\begin{array}{c}
\begin{array}{cccccc} 1 & 2 & j & \cdots & C \end{array} \\
\begin{bmatrix}
u_{11} & u_{12} & u_{1j} & \cdots & u_{1C} \\
u_{21} & u_{22} & u_{2j} & \cdots & u_{2C} \\
u_{i1} & u_{i2} & u_{ij} & \cdots & u_{iC} \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
u_{n1} & u_{n2} & u_{nj} & \cdots & u_{nC}
\end{bmatrix} \\
\begin{array}{cccccc} c_1 & c_2 & c_j & \cdots & c_C \end{array}
\end{array}
$$

对于K-means算法，每一行只会有一个元素为100%，其他均为0，对于FCM算法，每一行的归属概率之和为100%。

课程设计总结

本文简要介绍了聚类算法K-means以及Fuzz C-means算法相关的基本思想以及算法实现的流程，然后对于这次的实验也给出了具体的方法。最终是实验代

码与结果，并对于结果进行了分析。

通过这次课程设计，我们小组对聚类算法K-means以及Fuzz C-means算法有了相对课堂知识的进一步的理解与学习。由于本小组能力有限，尝试了其他办法想进一步减少算法的时间复杂度，从而去提高结果的准确率，但是没有得到很好的效果。在完成本次课程设计的过程中，我们小组针对每个部分共同讨论学习，对相关的编程知识有了新的认识，通过在网上查找相关资料以及寻求同学的帮助，算是比较完整地完成了这次课程设计。但是试验的结果通过多次调试未能达到老师提出的标准，仍需继续努力研究。

在这次代码检验结果中，由于每一次选取随机数存在一定的随机性，因此每次得出的结果也有一定的随机性。通过多次反复地测试检验，可发现相关算法的结果大致上都在一定范围内波动，本次报告中有选取部分结果以及摘取了多次检验得出的最优结果。通过比较可以发现，Fuzz C-means算法相对于K-means算法得到的结果一般要更精确一些，大致符合实际的理论。

通过这次课程设计，我们充分认识到了平时学习上的不足，对课程设计相关知识的探索有利于提高我们的思维能力，是一种很好的学习机会。虽然结果不太理想，但还有机会继续来学习更好的优化方法，新的聚类方法，来提高聚类的正确率。

附录：

在做这次课设的过程中找到了相关的资料，不过是关于KNN（K-近邻算法）的，觉得有一定的参考价值。

K-近邻算法的基本概念

k-近邻算法，即是给定一个训练数据集，对新的输入实例，在训练数据集中找到与该实例最邻近的k个数据， 这k个数据的多数属于某个类，就把该输入实例分类到这个类中。k-近邻算法中，所选择的邻居都是已经正确分类的对象。该方法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。

k-近邻方法虽然从原理上也依赖于极限定理，但在类别决策时，只与极少量的相邻样本有关。由于k-近邻方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属类别的，因此对于类域的交叉或重叠较多的待分样本集来说，k-近邻方法较其他方法更为适合。

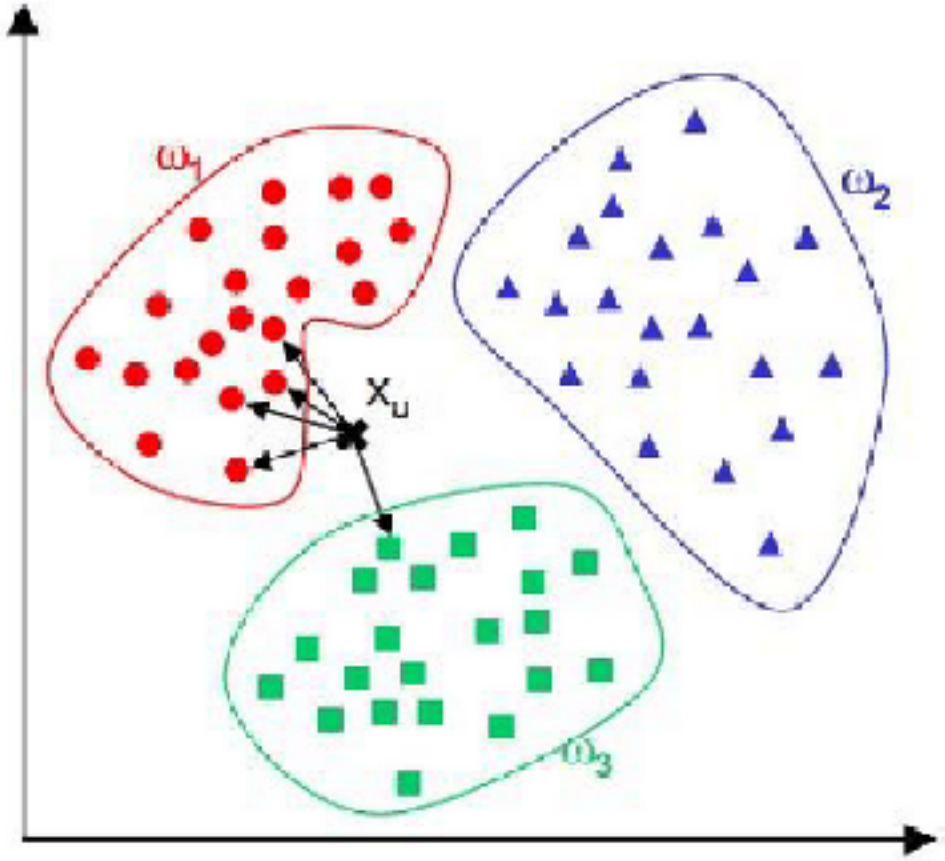
k-近邻算法不仅可以用于分类，还可以用于回归。通过找出一个样本的k个最近邻居，将这些邻居的属性的平均值赋给该样本，就可以得到该样本的属性。更有用的方法是将不同距离的邻居对该样本产生的影响给予不同的权值(weight)，如权值与距离成正比（组合函数）。

该算法在分类时有个主要的不足是，当样本不平衡时，如一个类的样本容量很大，而其他类样本容量很小时，有可能导致当输入一个新样本时，该样本的K个邻居中大容量类的样本占多数。 该算法只计算“最近的”邻居样本，某一类的样本数量很大，那么或者这类样本并不接近目标样本，或者这类样本很靠近目标样本。无论怎样，数量并不能影响运行结果。可以采用权值的方法（和该样本距离小的邻居权值大）来改进。该方法的另一个不足之处是计算量较大，因为对每一个待分类的文本都要计算它到全体已知样本的距离，才能求得它的K个最近邻点。目前常用的解决方法是事先对已知样本点进行剪辑，事先去除对分类作用不大的样本。该算法比较适用于样本容量比较大的类域的自动分类，而那些样本容量较小的类域采用这种算法比较容易产生误分。

K近邻算法的工作原理

存在一个样本数据集合（即训练样本集），并且样本集中每个数据都存在标签（即每个数据与所属分类的对应关系）。输入没有标签的新数据之后，将新数据的每个特征与样本集中数据对应的特征进行比较，算法将提取出样本集中特征最相似数据（最近邻）的分类标签。

一般选择样本数据集中前K个最相似的数据，k一般不大于20的整数。



K近邻算法的算法步骤

Step.1 计算已知类别数据集中的点与当前点之间的距离；

$$d=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$$

Step.2 按照距离递增次序排序；
Step.3 选取与当前点距离最小的k个点；
Step.4 确定前k个点所在类别的出现频率；
Step.5 返回前k个点出现频率最高的类别作为当前点的预测分类。

K-近邻算法的优缺点

k-近邻算法拥有以下优点:

精度高、对异常值不敏感、无输入数据假定

k-近邻算法拥有以下缺点：

时间复杂度和空间复杂度都很高

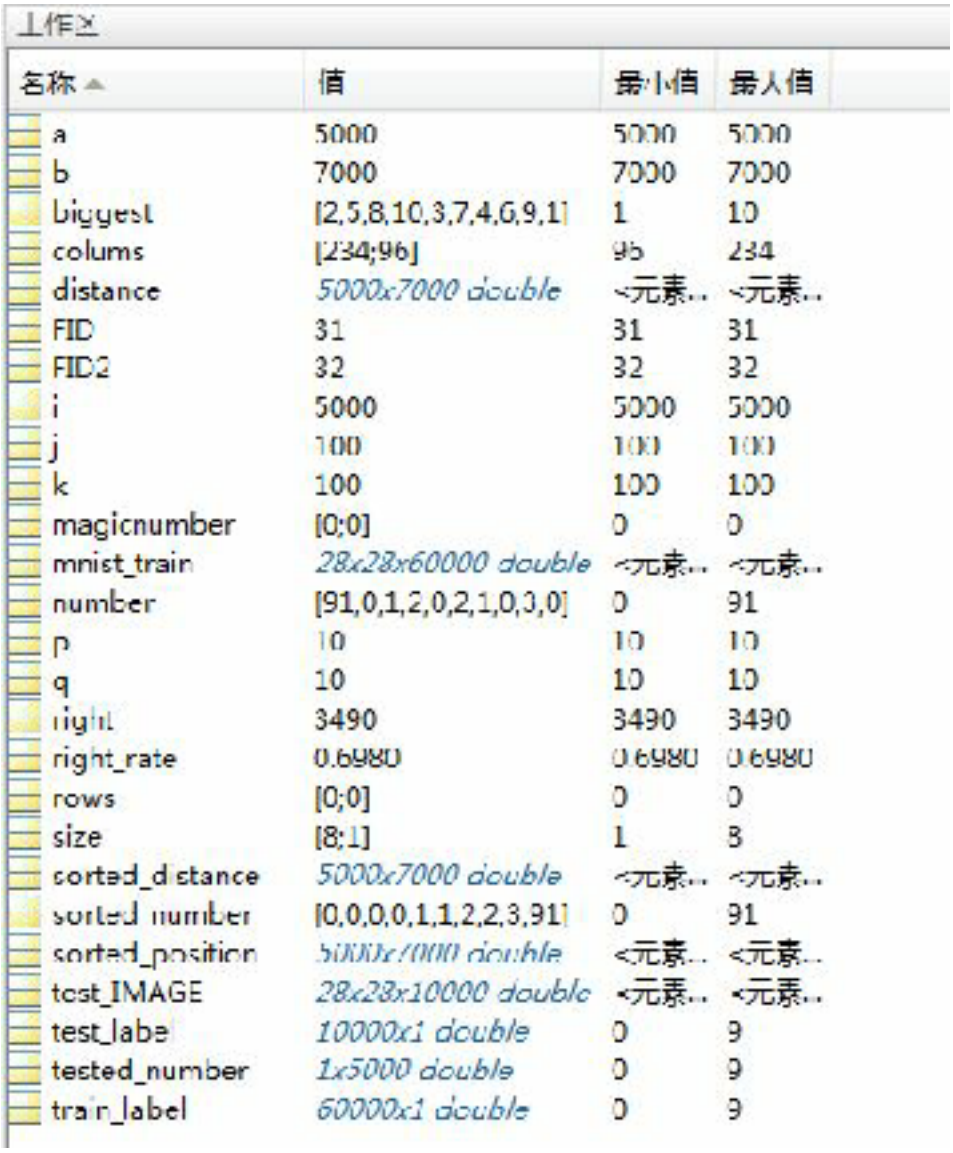
相关代码及运行结果：

```
clc
clear
k=100;%k距离
a=950;%待测数据个数
b=7500;%训练数据个数

test_IMAGE=zeros(28,28,10000);
FID = fopen('t10k-images.idx3-ubyte','r');
for i=1:10000
        train_image=fread(FID,[28,28]);
        %imshow(train_image');
        for p=1:28
            for q=1:28
```

```
                test_IMAGE(p,q,i)=train_image(p,q);%赋值给三维矩阵
            end
        end
end


mnist_train=zeros(28,28,60000);
FID = fopen('train-images.idx3-ubyte','r');
for i=1:60000
        train_image=fread(FID,[28,28]);
        %imshow(train_image');
        for p=1:28
            for q=1:28
                mnist_train(p,q,i)=train_image(p,q);%赋值给三维矩阵
            end
        end
end


C=cat(3,test_IMAGE,mnist_train);


distance=zeros(a,b);
number=zeros(1,10);
tested_number=zeros(1,a);

FID = fopen('t10k-labels.idx1-ubyte','r');%载入待测图像标签
test_label=fread(FID);

FID2 = fopen('train-labels.idx1-ubyte','r');%载入训练图像标签
train_label=fread(FID2);

label=[test_label(9:10000,:);train_label(9:60000,:)];

randjz=round((a+b)*rand(1,a));

for i=1:a
    for p=1:28
        for q=1:28
            E(p,q,i)=C(p,q,randjz(i));
            F(i)=label(randjz(i));
        end
    end
end



for i=1:a
        for j=1:b
                for p=1:28
                    for q=1:28
                        if (E(p,q,i) - C(p,q,j)) > 45
                             distance(i,j)=distance(i,j)+1;
                        end
                    end
                end
        end
end
%计算距离矩阵
[sorted_distance,sorted_position] = sort(distance,2);%距离矩阵排序
for i=1:a
        for q=1:10
```

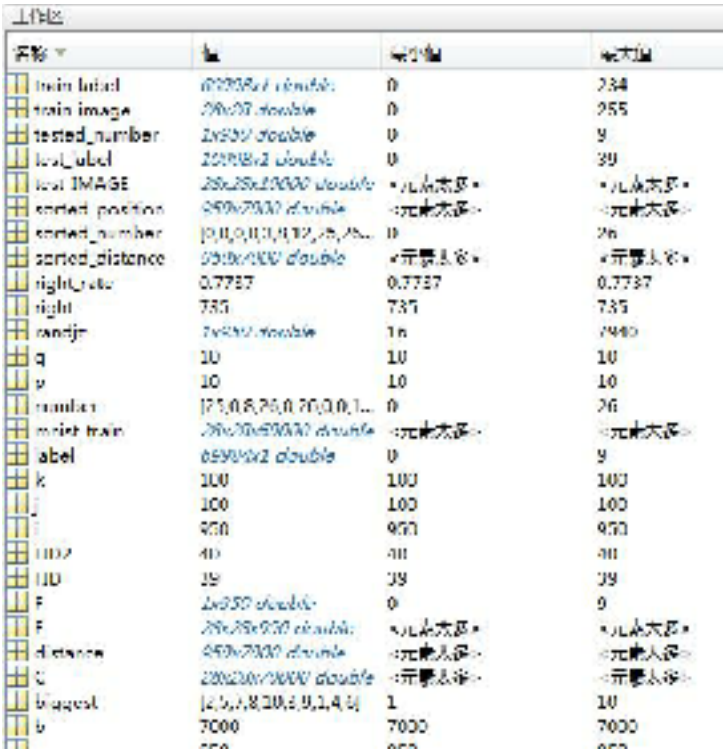
```
            number(q)=0;
        end
        for j=1:k
            for p=1:10
                if label(sorted_position(i,j))==(p-1)
                    number(p)=number(p)+1;
                end
            end
        end
%计算k距离内数据个类数目
        [sorted_number,biggest]=sort(number,2);
        tested_number(1,i)=biggest(10)-1;%给待测图像分类
end
right=0;
for i=1:a
        if tested_number(i)==F(i)
            right=right+1;
        end
end
right_rate=right/a;%计算正确率
```

结果：
值得注意的是，该代码不能测试过多的图片，会提示内存不足。
训练集7000张图片，测试集5000张：



训练集6000张图片，测试集950张：

训练集7000张图片，测试集950张：



训练集7500张图片，测试集950张：

当训练集7000张图片，测试集950张时结果最好，正确率为0.7737.

K-Means聚类算法与KNN的区别

K-Means	KNN
目的是为了将一系列点集分成k类	目的是为了确定一个点的分类
K-Means是聚类算法	KNN是分类算法
非监督学习，将相似数据归到一起从而得到分类，没有外部分类	监督学习，分类目标事先已知
训练数据集无label，是杂乱无章的，经过聚类后才变得有点顺序，先无序，后有序	训练数据集有label，已经是完全正确的数据
有明显的前期训练过程	没有明显的前期训练过程，属于memory-based learning
K的含义：“k”是类的数目。K是人工固定好的数字，假设数据集合可以分为K个簇，由于是依靠人工定好，需要一点先验知识	K的含义：“k”是用来计算的相邻数据数。来了一个样本x，要给它分类，即求出它的y，就从数据集中，在x附近找离它最近的K个数据点，这K个数据点，类别c占的个数最多，就把x的label设为c
K值确定后每次结果可能不同，从 n个数据对象任意选择 k 个对象作为初始聚类中心，随机性对结果影响较大	K值确定后每次结果固定
时间复杂度：O(n*k*t)，t为迭代次数	时间复杂度：O（n）
相似点：都包含这样的过程，给定一个点，在数据集中找离它最近的点。即二者都用到了NN(Nears Neighbor)算法，一般用KD树来实现NN。	