聚类——GAKFCM的matlab程序

作者：凯鲁嘎吉 - 博客园 http://www.cnblogs.com/kailugaji/

在聚类——GAKFCM文章中已介绍了GAKFCM算法的理论知识，现在用matlab进行实现,下面这个例子是用GA初始化聚类中心。

1.matlab程序

GAKFCM_main.m

```
function [ave_acc_GAKFCM,max_acc_GAKFCM,min_acc_GAKFCM,ave_iter_GA,ave_iter_KFCM,ave_run_time]=GAKFCM_main(X,real_label,K)
%输入K:聚的类，max_iter是最大迭代次数,T:遗传算法最大迭代次数，n:种群个数， X：没有进行归一化
%输出ave_acc_KFCM：迭代max_iter次之后的平均准确度,iter:实际KFCM迭代次数
t0=cputime;
max_iter=20;
s=0;
s_1=0;
s_2=0;
iter_GA=zeros(max_iter,1);
iter_KFCM=zeros(max_iter,1);
accuracy=zeros(max_iter,1);
for i=1:max_iter
    [label, iter_KFCM(i), ~,iter_GA(i)]=My_GAKFCM(X,K);
    accuracy(i)=succeed(real_label,K,label);
    s=s+accuracy(i);
    s_1=s_1+iter_GA(i);
    s_2=s_2+iter_KFCM(i);
    fprintf('第 %2d 次，GA的迭代次数为：%2d，KFCM的迭代次数为：%2d，准确度为：%.8f\t\n', i, iter_GA(i), iter_KFCM(i), accuracy(i));
end
ave_acc_GAKFCM=s/max_iter;
max_acc_GAKFCM=max(accuracy);
min_acc_GAKFCM=min(accuracy);
ave_iter_GA=s_1/max_iter;
ave_iter_KFCM=s_2/max_iter;
run_time=cputime-t0;
ave_run_time=run_time/max_iter;
```

My_GAKFCM.m

```
function  [label, iter_KFCM, para_miu,iter_GA]=My_GAKFCM(X,K)
%用GA初始聚类中心
%输入K：聚类数，X：数据集
%输出：label:聚的类, para_miu:模糊聚类中心μ，iter_KFCM：KFCM迭代次数
format long
eps=1e-4;  %定义迭代终止条件的eps
alpha=2;  %模糊加权指数，[1,+无穷)
T=100;  %最大迭代次数
%sigma_2=2^(-4);  %高斯核函数的参数2*sigma^2
sigma_2=150;  %高斯核函数的参数sigma^2
[X_num,X_dim]=size(X);
fitness=zeros(X_num,1);  %目标函数
responsivity=zeros(X_num,K);  %隶属函数
R_up=zeros(X_num,K);  %隶属函数的分子部分
count=zeros(X_num,1);  %统计distant中每一行为0的个数
%随机初始化K个聚类中心
% [X_num,~]=size(X);
% rand_array=randperm(X_num);  %产生1~X_num之间整数的随机排列
% para_miu=X(rand_array(1:K),:);  %随机排列取前K个数，在X矩阵中取这K行作为初始聚类中心
%用GA初始聚类中心
[para_miu,iter_GA]=my_genetic(X,K);
% KFCM算法
for t=1:T
    %欧氏距离，计算（X-para_miu）^2=X^2+para_miu^2-2*para_miu*X'，矩阵大小为X_num*K
    distant=(sum(X.*X,2))*ones(1,K)+ones(X_num,1)*(sum(para_miu.*para_miu,2))'-2*X*para_miu';
    %高斯核函数，X_num*K的矩阵
    kernel_fun=exp((-distant)./(sigma_2));
    %更新隶属度矩阵X_num*K
    for i=1:X_num
        count(i)=sum(kernel_fun(i,:)==1);
        if count(i)>0
            for k=1:K
                if kernel_fun(i,k)==1
                    responsivity(i,k)=1./count(i);
                else
                    responsivity(i,k)=0;
                end
            end
        else
            R_up(i,:)=(1-kernel_fun(i,:)).^(-1/(alpha-1));  %隶属度矩阵的分子部分
            responsivity(i,:)= R_up(i,:)./sum( R_up(i,:),2);
        end
    end
    %目标函数值
    fitness(t)=2*sum(sum((ones(X_num,K)-kernel_fun).*(responsivity.^(alpha))));
     %更新聚类中心K*X_dim
    miu_up=(kernel_fun.*(responsivity.^(alpha)))'*X;  %μ的分子部分
    para_miu=miu_up./(sum(kernel_fun.*(responsivity.^(alpha)))'*ones(1,X_dim));
    if t>1
        if abs(fitness(t)-fitness(t-1))<eps
            break;
        end
    end
end
iter_KFCM=t;  %实际迭代次数
[~,label]=max(responsivity,[],2);
```

succeed.m

```
function accuracy=succeed(real_label,K,id)
%输入K：聚的类，id：训练后的聚类结果，N*1的矩阵
N=size(id,1);   %样本个数
p=perms(1:K);   %全排列矩阵
p_col=size(p,1);   %全排列的行数
new_label=zeros(N,p_col);   %聚类结果的所有可能取值，N*p_col
num=zeros(1,p_col);  %与真实聚类结果一样的个数
%将训练结果全排列为N*p_col的矩阵，每一列为一种可能性
for i=1:N
    for j=1:p_col
        for k=1:K
            if id(i)==k
                new_label(i,j)=p(j,k);  %iris数据库，1 2 3
            end
        end
    end
end
%与真实结果比对，计算精确度
for j=1:p_col
    for i=1:N
        if new_label(i,j)==real_label(i)
                num(j)=num(j)+1;
        end
    end
end
accuracy=max(num)/N;
```

my_genetic.m

```
function [para_miu_new,iter]=my_genetic(data,K)
%data:数据集，K:聚类数
pc_0=0.6;  %初始交叉概率
pm_0=0.1;  %初始变异概率
```

```
eps=1e-4;  %定义迭代终止条件的eps
n=50;  %n:n个初始个体，每个个体为K*X_dim
T=100;  %T:最大迭代次数
pc=zeros(T,1);  %交叉概率
pm=zeros(T,1);  %变异概率
fitness=zeros(n,1);
ave_fitness=zeros(T,1);
%实数编码
%对data做最大-最小归一化处理
[data_num,~]=size(data);
X=(data-ones(data_num,1)*min(data))./(ones(data_num,1)*(max(data)-min(data)));
%产生初始种群
population=init_population(X,K,n);
for t=1:T
    %更新适应度
    fitness=fit_vector(X,K,population,n);
    %非线性排序选择
    population=sort_select(population,fitness);
    %计算交叉概率，进行交叉操作
    pc(t)=pc_0*(1-(t-1)/T);
    population=crossover(population,pc(t));
    %计算变异概率，进行变异操作
    pm(t)=pm_0*(1-(t-1)/T);
    population=mutation(population,pm(t));
    ave_fitness(t)=sum(fitness)/n;
    if t>1
        if abs( ave_fitness(t)- ave_fitness(t-1))<eps
            break;
        end
    end
end
iter=t;  %实际迭代次数
%输出适应度最大的个体
[~,index_final]=max(fitness);
para_miu=population(:,:,index_final);
%解码para_miu
para_miu_new=para_miu.*(ones(K,1)*(max(data)-min(data)))+ones(K,1)*min(data);
```

init_population.m

```
function population=init_population(X,K,n)
%data:数据集，K:聚类数，n:n个初始个体，每个个体为K*X_dim，new_index为排序后的个体序号
rand_num=3;  %rand_num:随机取rand_num个样本作为一类
[X_num,X_dim]=size(X);
individual=zeros(K,X_dim);  %individual为聚类中心矩阵，K*X_dim的矩阵
population=zeros(K,X_dim,n);
```

```
for i=1:n
    %随机初始化K个聚类中心
    for k=1:K
        rand_array=randperm(X_num);  %产生1~X_num之间整数的随机排列
        temp=X(rand_array(1:rand_num),:);
        individual(k,:)=sum(temp)./rand_num;  %individual(k)为1*X_dim的矩阵，为一类的聚类中心，对rand_num取平均
    end
    population(:,:,i)=individual;
end
```

fit_vector.m

```
function fitness=fit_vector(X,K,population,n)
fitness=zeros(n,1);
for i=1:n
    %计算个体适应度
    fitness(i)=fitness_value(X,K,population(:,:,i));  %fitness为GAKFCM适应度函数  n*1的矩阵
end
```

fitness_value.m

```
function fitness=fitness_value(X,K,para_miu)
%X是数据，para_miu为每一个individual矩阵，K*X_dim，fitness为GAKFCM适应度函数
%sigma_2=2^(-4);  %高斯核函数的参数2*sigma^2
sigma_2=150;  %高斯核函数的参数sigma^2
alpha=2;  %模糊加权指数，[1,+无穷)
[X_num,~]=size(X);
responsivity=zeros(X_num,K);  %隶属函数
R_up=zeros(X_num,K);
count=zeros(X_num,1);  %统计distant中每一行为0的个数
%欧氏距离，计算（X-para_miu）^2=X^2+para_miu^2-2*para_miu*X'，矩阵大小为X_num*K
distant=(sum(X.*X,2))*ones(1,K)+ones(X_num,1)*(sum(para_miu.*para_miu,2))'-2*X*para_miu';
%高斯核函数，X_num*K的矩阵
kernel_fun=exp((-distant)./(sigma_2));
%更新隶属度矩阵X_num*K
for i=1:X_num
    count(i)=sum(kernel_fun(i,:)==1);
    if count(i)>0
        for k=1:K
            if kernel_fun(i,k)==1
                responsivity(i,k)=1./count(i);
            else
                responsivity(i,k)=0;
            end
        end
    else
        R_up(i,:)=(1-kernel_fun(i,:)).^(-1/(alpha-1));  %隶属度矩阵的分子部分
        responsivity(i,:)= R_up(i,:)./sum( R_up(i,:),2);
    end
end
%目标函数值
fitness_KFCM=2*sum(sum((ones(X_num,K)-kernel_fun).*(responsivity.^(alpha))));   %KFCM的目标函数
fitness=1/(1+fitness_KFCM);  %fitness为GAKFCM适应度函数
```

sort_select.m

```
function population=sort_select(population,fitness)
%q属于(0,1)为参数, i表示排序序号，本文取q=0.1
q=0.1;
[n,~]=size(fitness);
new_index=zeros(n,1);    %选择之后最优个体的序号
fun=zeros(n,1);  %非线性排序选择概率分布函数
add_pro=zeros(n,1);  %累积概率
[~,index_fit]=sort(fitness,'descend');  %将fitness按降序排序
%计算每个个体选择的概率
for i=1:n
    fun(i)=q*(1-q)^(i-1);
end
new_fun=fun/sum(fun);
%求累积概率
for i=1:n
    add_pro(i)=sum(new_fun(1:i));
end
%选择最优个体，求其在X中的顺序
for t=1:n
    rand_pro=rand();   %[0,1]之间的随机数
    if rand_pro<=add_pro(1)
        new_index(t)=index_fit(1);
    end
    for i=2:n
        if (rand_pro>add_pro(i-1))&&(rand_pro<=add_pro(i))
            new_index(t)=index_fit(i);
        end
    end
    population(:,:,t)=population(:,:,new_index(t));
end
```

crossover.m

```
function population=crossover(population,pc)
%个体之间进行交叉操作，交换两行
[K,~,n]=size(population);
num=floor(n/2);   %对n/2向下取整
for i=1:num
    rand_c=rand();   %[0,1]之间的随机数
    rand_pro=unidrnd(K);   %[1,K]之间的随机整数
    %交换两个矩阵中的第rand_pro行
    if pc>rand_c
        t=population(rand_pro,:,2*i-1);
        population(rand_pro,:,2*i-1)=population(rand_pro,:,2*i);
        population(rand_pro,:,2*i)=t;
    end
end
```

mutation.m

```
function population=mutation(population,pm)
%个体进行变异操作
[K,X_dim,n]=size(population);
for i=1:n
    rand_m=rand();   %[0,1]之间的随机数
    rand_pro=unidrnd(K);   %[1,K]之间的随机整数
    if pm>rand_m
        %对第rand_pro行进行变异操作
        population(rand_pro,:,i)=rand(1,X_dim);
    end
end
```

2.在UCI数据库的iris上的运行结果

```
>> data_load=dlmread('E:\My matlab\database\iris.data');data=data_load(:,1:4);real_label=data_load(:,5);
>> [ave_acc_GAKFCM,max_acc_GAKFCM,min_acc_GAKFCM,ave_iter_GA,ave_iter_KFCM,ave_run_time]=GAKFCM_main(data,real_label,3)
第  1 次，GA的迭代次数为：10，KFCM的迭代次数为： 6，准确度为：0.89333333
第  2 次，GA的迭代次数为： 3，KFCM的迭代次数为：13，准确度为：0.89333333
第  3 次，GA的迭代次数为：39，KFCM的迭代次数为： 8，准确度为：0.89333333
第  4 次，GA的迭代次数为：66，KFCM的迭代次数为：10，准确度为：0.89333333
第  5 次，GA的迭代次数为：18，KFCM的迭代次数为： 8，准确度为：0.89333333
第  6 次，GA的迭代次数为：26，KFCM的迭代次数为： 6，准确度为：0.89333333
第  7 次，GA的迭代次数为：93，KFCM的迭代次数为： 7，准确度为：0.89333333
第  8 次，GA的迭代次数为：70，KFCM的迭代次数为： 5，准确度为：0.89333333
第  9 次，GA的迭代次数为：11，KFCM的迭代次数为： 8，准确度为：0.89333333
第 10 次，GA的迭代次数为： 9，KFCM的迭代次数为： 9，准确度为：0.89333333
第 11 次，GA的迭代次数为：80，KFCM的迭代次数为： 7，准确度为：0.89333333
第 12 次，GA的迭代次数为：39，KFCM的迭代次数为： 7，准确度为：0.89333333
```

```
第 13 次，GA的迭代次数为：12，KFCM的迭代次数为： 6，准确度为：0.89333333
第 14 次，GA的迭代次数为：22，KFCM的迭代次数为： 6，准确度为：0.89333333
第 15 次，GA的迭代次数为： 7，KFCM的迭代次数为： 8，准确度为：0.89333333
第 16 次，GA的迭代次数为：13，KFCM的迭代次数为： 8，准确度为：0.89333333
第 17 次，GA的迭代次数为：19，KFCM的迭代次数为：15，准确度为：0.89333333
第 18 次，GA的迭代次数为：22，KFCM的迭代次数为：14，准确度为：0.89333333
第 19 次，GA的迭代次数为：30，KFCM的迭代次数为： 9，准确度为：0.89333333
第 20 次，GA的迭代次数为：13，KFCM的迭代次数为： 7，准确度为：0.89333333

ave_acc_GAKFCM =
   0.893333333333333

max_acc_GAKFCM =
   0.893333333333333

min_acc_GAKFCM =
   0.893333333333333

ave_iter_GA =
  30.100000000000001

ave_iter_KFCM =
   8.350000000000000

ave_run_time =
   2.457812500000000
```

遗传算法中的具体实现细节有可能有误，望指正。