哈尔滨工业大学计算机科学与技术学院

# 实验报告

课程名称： 机器学习

课程类型： 选修

实验题目： k-means 聚类和混合高斯模型

学号：陈宇豪
姓名： 1190201115

# 一、实验目的

- 理解 k-means 聚类过程
- 理解混合高斯模型(GMM)用 EM 估计参数的实现过程
- 掌握 k-means 和混合高斯模型的联系
- 学会 EM 估计参数的方法和代码实现
- 学会用 GMM 解决实际问题

# 二、实验要求及实验环境

## 实验要求：

测试：
用高斯分布产生 k 个高斯分布的数据（不同均值和方差）。
（1）用 k-means 聚类，测试效果；
（2）用混合高斯模型和你实现的 EM 算法估计参数，看看每次迭代后似然值变化情况，考察 EM 算法是否可以获得正确的结果。

应用：可以 UCI 上找一个简单问题数据，用你实现的 GMM 进行聚类。

## 实验环境：

Windows 10，matlab2016a

# 三、设计思想（本程序中的用到的主要算法及数据结构）

本次实验分为 k-means 和 GMM 算法两部分，它们同属于 EM 算法。EM 算法是在概率模型中寻找参数最大似然估计或者最大后验估计的算法，其中概率模型依赖于无法观测的隐性变量。

最大期望算法经过两个步骤交替进行计算：

第一步是计算期望（E），利用对隐藏变量的现有估计值，计算其最大似然估计值；
第二步是最大化（M），最大化在 E 步上求得的最大似然值来计算参数的值。M 步上找到的参数估计值被用于下一个 E 步计算中，这个过程不断交替进行。

## 3.1 kmeans 算法

kmeans 算法的实现步骤如下：

1. 随机在数据中寻找 k 个不同的点（k 为计划分成的类数）作为初始聚类中心

```
%产生随机且不同的k个中心点
center=zeros(k,col);
i=1;
while i<k+1
    temp=X(randi([1,num],1),:);
    if ismember(temp,center)
        continue;
    end
    center(i,:)=temp;
    i=i+1;
end
```

2. 对数据集中的每一个数据，按照距离各中心点的距离，将其与最近的中心点关联起来，所有与同一个中心点关联的数据聚成一类。

```
for i=1:num
    minlen=9999;
    tag=1;
    for j= 1:k
        len=cal_distance(X(i,:),center(j,:),col);
        if minlen>len
            minlen=len;
            tag=j;
        end
    end
    class(i)=tag;
    tagnum(tag)=tagnum(tag)+1;
end
```

3. 计算同一组数据点的坐标平均值，将该组关联的中心点移动到该处。

```
for i=1:num
    newcenter(class(i),:)=newcenter(class(i),:)+X(i,:);
end
for i=1:k
    newcenter(i,:)=newcenter(i,:)./tagnum(i);
end
```

4. 重复 2、3 步，直到聚类中心点不再变化。

## 3.2 GMM 算法

多维高斯分布生成的多维随机变量 x 的密度函数为：

$$p(x|\mu,\Sigma)=\frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}}\exp(-\frac{1}{2}(x-\mu)\Sigma^{-1}(x-\mu)^{T})$$

其中，$\mu$ 为平均值，Σ 为方差矩阵，d 为数据的维数。

给定样本集 $X=\begin{bmatrix}x1\\x2\\...\\xn\end{bmatrix}$，X 是一个 n*d 的矩阵，可以将其视作由多个多元高斯分布叠加产生，假设数据由 k 个高斯分布混合而成，产生数据 xi 的概率为在 k 个高斯分布中产生概率的叠加。

$$p(x_i)=\sum_{j=1}^{k}\pi_j p(x_i|\mu_j,\Sigma_j)$$

$\pi_j$ 为混合系数，$\mu_j$ 为对应高斯分布的平均值，$\Sigma_j$ 为对应高斯分布的协方差矩阵。

j 个高斯分布的混合系数之和为 1.

因此，可以认为该数据的生成相当于从 k 个高斯分布中挑选出一个生成，我们设 k 维二值变量 z，只有其中一维数据为 1，其他都为 0。z 的先验分布为：

$$p(z)=\prod_{j=1}^{k}\pi_j^{zj}$$

后验概率为

$$\gamma(z_j)\equiv p(z_j=1|x_i)=\frac{p(z_j=1)p(x_i|z_j=1)}{p(x_i)}=\frac{\pi_j p(x_i|\mu_j,\Sigma_j)}{\sum_{l=1}^{k}\pi_l p(x_i|\mu_l,\Sigma_l)}$$

当后验概率已知时，混合高斯模型将训练样本划分成 k 个簇，对于每一个样本 x i，选择后验概率最大的类别作为标签类别。可以采用极大似然估计来求解样本的类别分布：

$$\ln p(X|\pi,\mu,\Sigma)=\ln\prod_{i=1}^{n}p(x_i)=\sum_{i=1}^{n}\ln\sum_{j=1}^{k}\pi_j p(x_i|u_j,\Sigma_j)$$

使上式最大化，对 $\mu_j$ 求导令导数为 0：

$$n_j = \sum_i \gamma(z_{ij})$$

$$\mu_j = \frac{1}{n_j} \sum_{i=1}^{n} \gamma(z_{ij}) x_i$$

同理，对$\Sigma_j$求导令导数为 0：

$$\Sigma_j = \frac{\sum_{i=1}^{n} \gamma(z_{ij})(x_i - \mu_j)(x_i - \mu_j)^T}{n_j}$$

对于混合系数$\pi_j$，有

$$\pi_j = \frac{n_j}{n}$$

gmm 算法过程概述如下：
1. 初始化各参数

$$\gamma(z_j) = \frac{\pi_j p(x_i|\mu_j, \Sigma_j)}{\sum_{l=1}^{k} \pi_l p(x_i|\mu_l, \Sigma_j)}$$

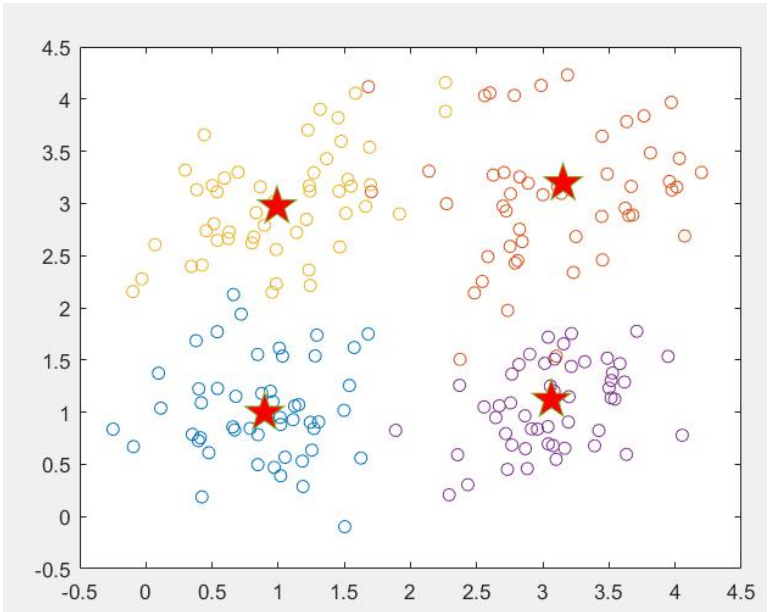
2. E 步根据公式 计算每个样本由各个混合高斯成分生成的后验概率
3. M 步：用上述公式更新参数 π i ， μ i ， Σ i
4. 重复以上过程，直到参数不再变化。

# 四、实验结果与分析

## 4.1kmeans

测试中产生数据的每个高斯分布的均值和协方差均不同

1. 数据点数量为 200，分 4 类

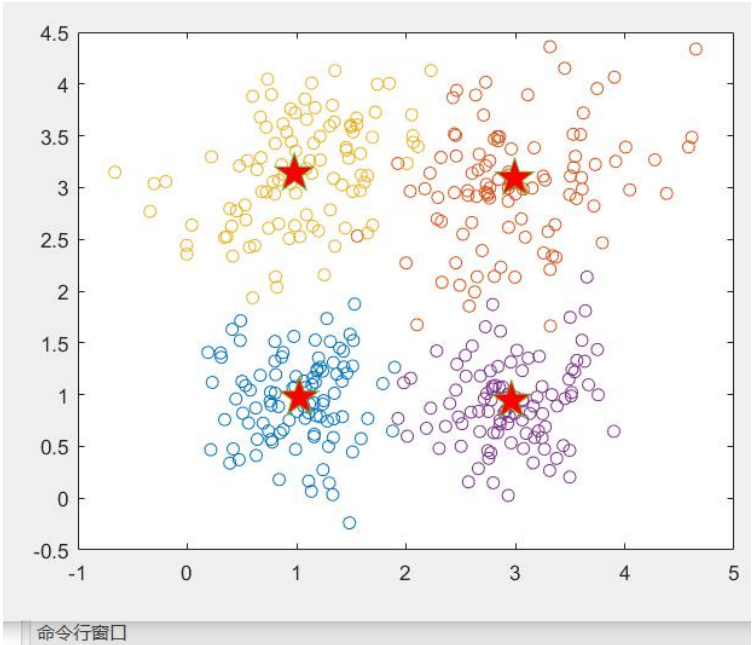



命令行窗口

```
iteration =

     7


accuracy =

    0.9500
```

迭代次数为 7，聚类准确度为 0.95.

2. 数据集数量为 400，分四类

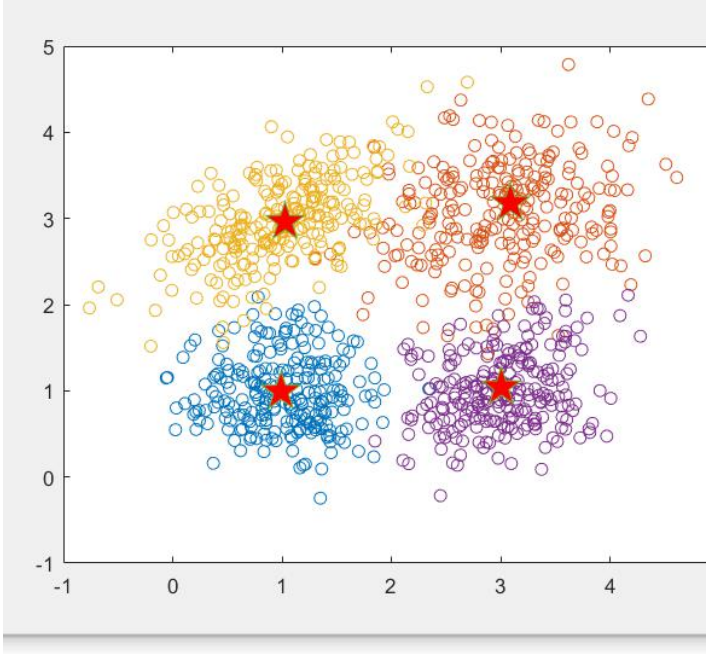



命令行窗口

```
iteration =

     6


accuracy =

    0.9575
```

迭代次数为 6，准确度为 0.9575

3. 数据集数量为 1000，分 4 类

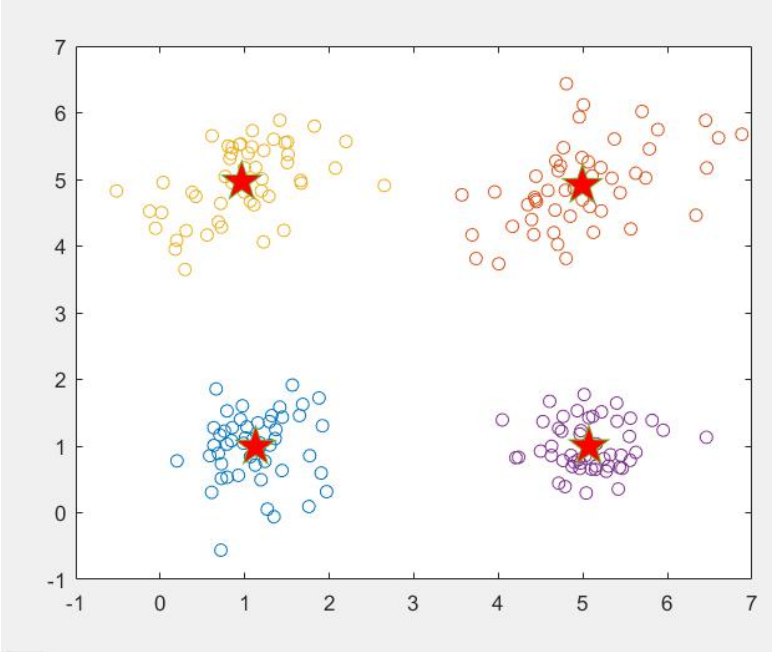



```
iteration =

     7


accuracy =

    0.9500
```

迭代次数为 7，准确度为 0.95

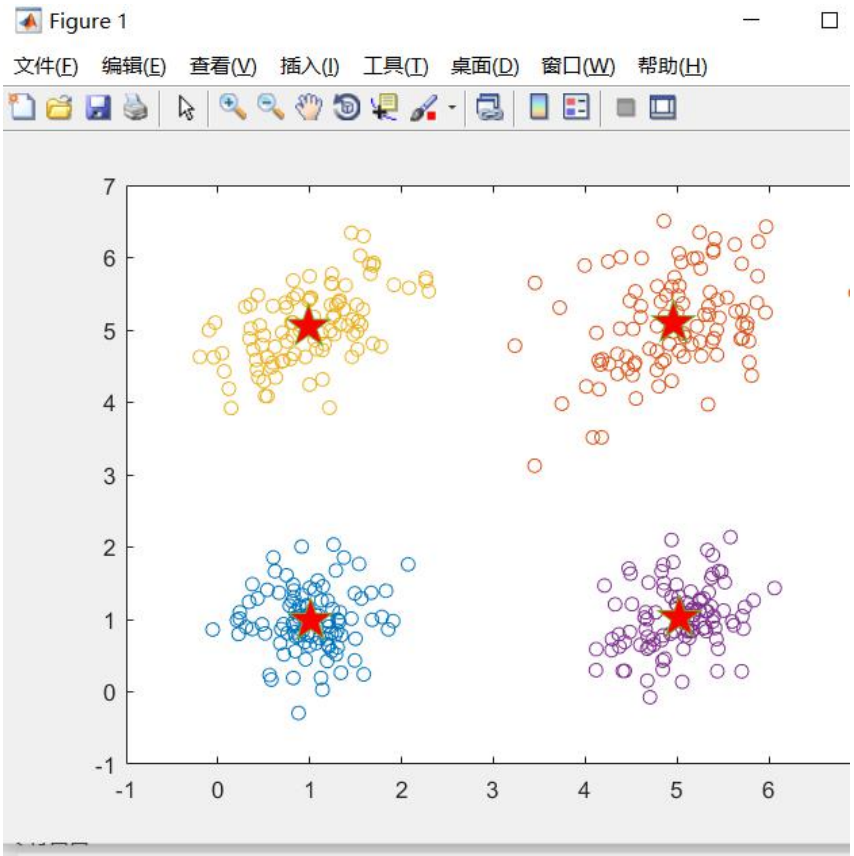
由以上测试可见，当类数保持不变时，数据集数量的增加对迭代次数和准确度影响不大。

4. 改变平均值，使得不同数据集之间距离更远，样本数量为 200




```
iteration =

     4


accuracy =

     1
```

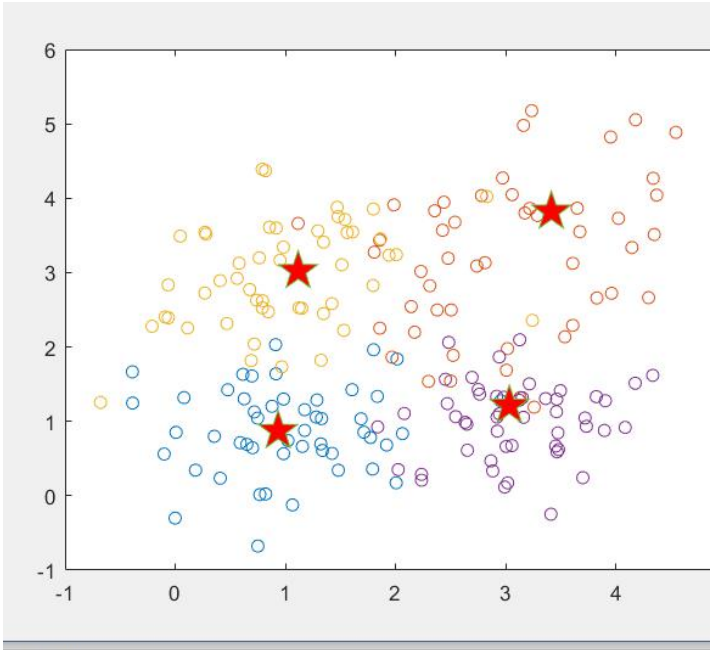
5. 数据样本数改为 400




```
iteration =

     3


accuracy =

     1
```

由此可以发现，当数据分布较分离时，迭代次数会变得很小，同时准确率变得很高。

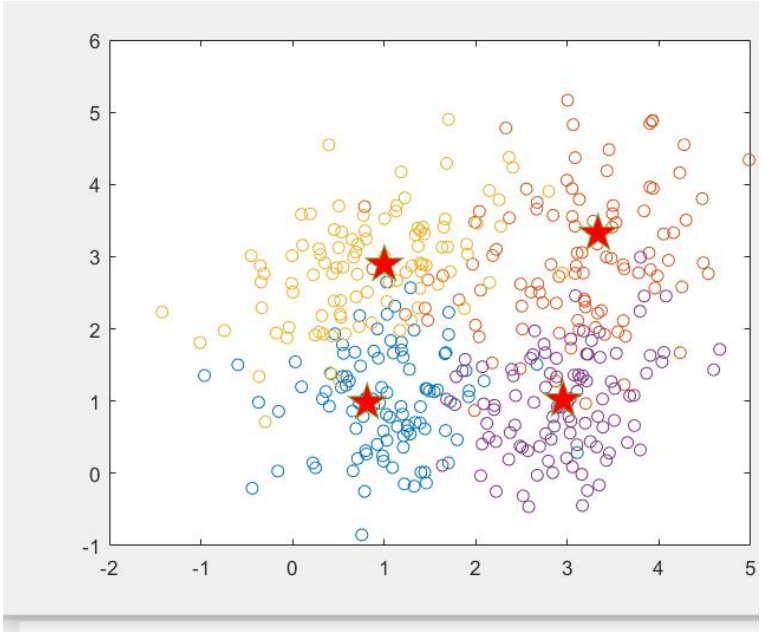
6. 将平均值改回初始状态，将方差调整得大一些，测试样本为 200




令行窗口

```
iteration =

     7


accuracy =

    0.8300
```

7. 测试样本为 400 时

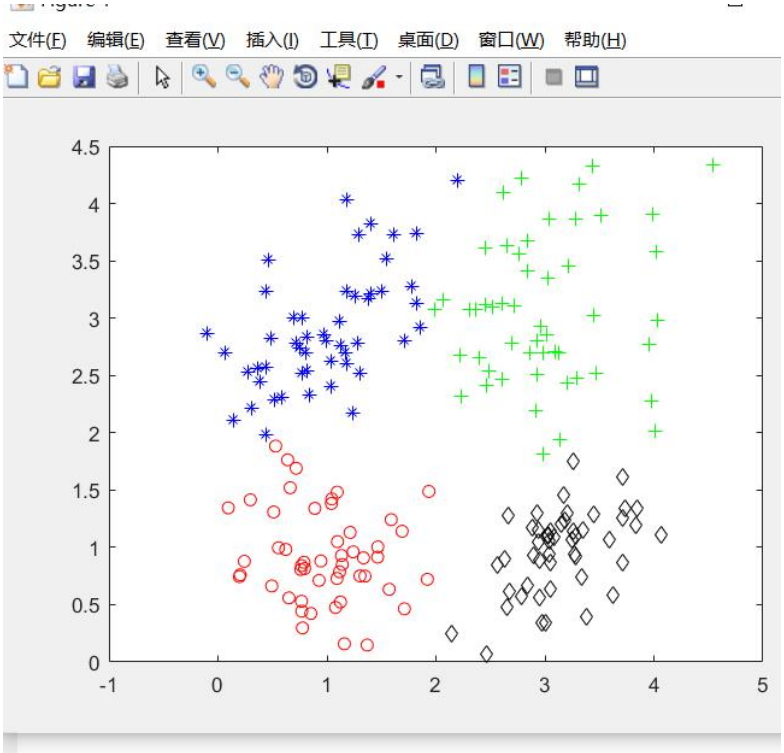



```
iteration =

    22


accuracy =

    0.8075
```

由此可见，当方差较大时会使聚类效果变差，迭代次数也会有一定程度的提高。

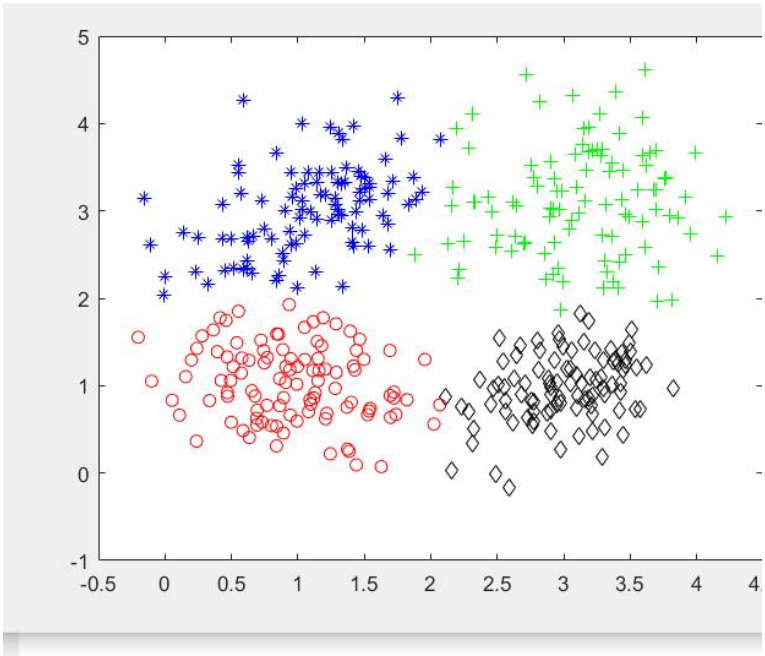
## 4.2 gmm

1. 数据点数量为 200，分 4 类



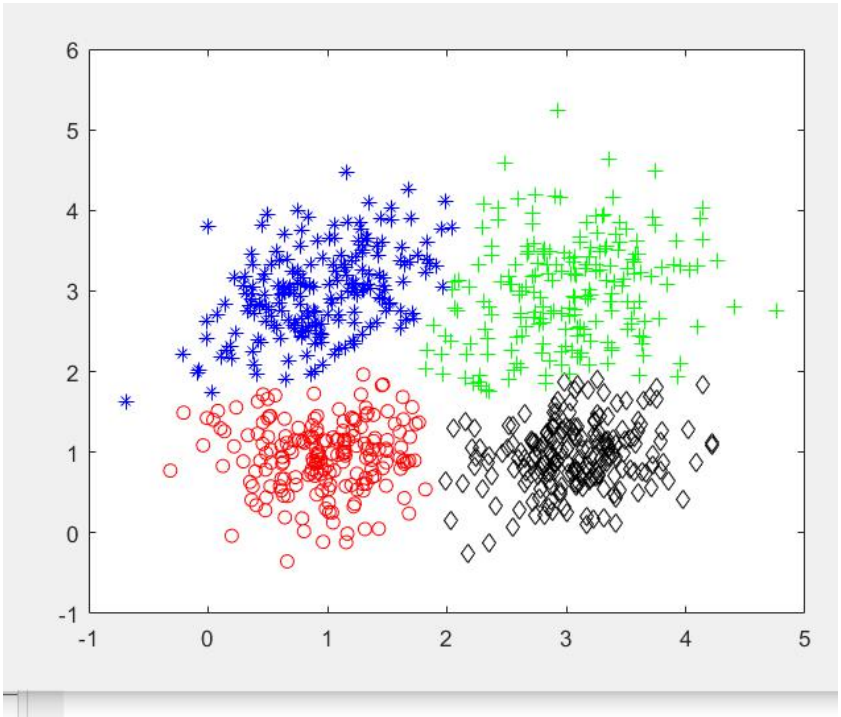

```
accuracy =

    0.9850
```

2. 数据点数量为 400，分 4 类




```
accuracy =

    0.9500
```

3. 数据点数量为 800，分 4 类

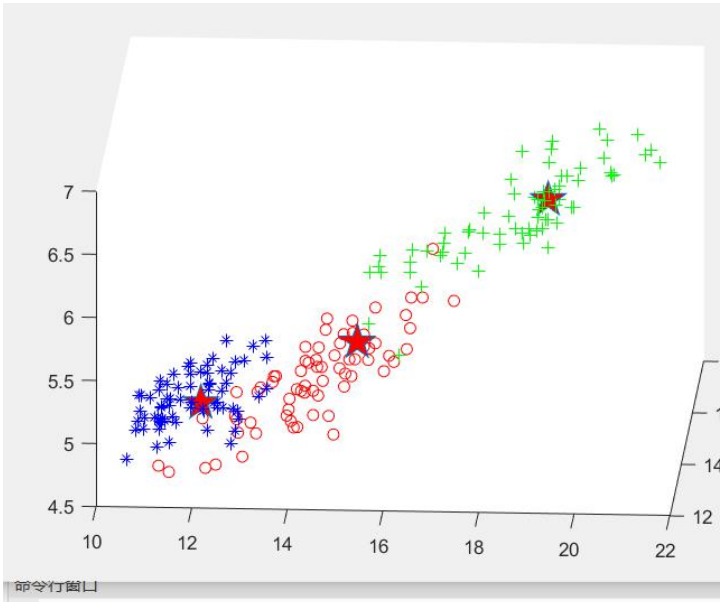



```
>> main

accuracy =

    0.9575
```

## 4.3UCI 数据

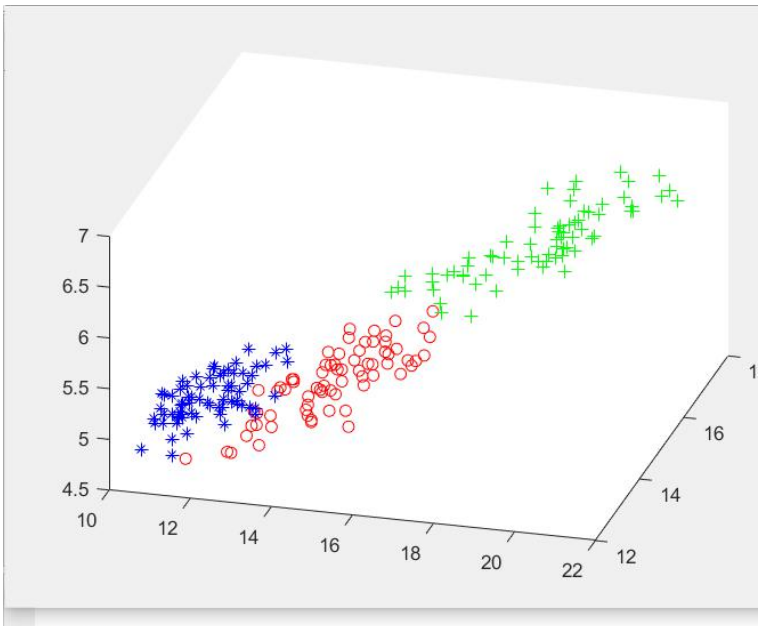
在 UCI 上下载数据，样本集大小为 210，共三类，参数有七个维度。已经做好标记。为了方便使结果可视化，我取了其中三维，去除标记后进行训练。

kmeans 算法




```
accuracy =

    0.8476
```

GMM 算法




```
accuracy =

    0.9524
```

## 五、结论

1. K-Means 的分类结果受初始中心点的影响较大，如果初始随机点取得特别接

近，可能最后的聚合结果会非常差；GMM-EM 算法对初值的敏感程度不高。
2. kmeans 和 gmm 算法在数据集数量增大或减小时，准确率变化不大。
3. 一般来说，kmeans 聚类的效果不如 gmm，但是 kmeans 的计算速度要快得多。

## 六、参考文献

https://veal98.gitee.io/cs-wiki/#/%E4%BA%BA%E5%B7%A5%E6%99%BA%E8%83%BD/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0/%E5%90%B4%E6%81%A9%E8%BE%BE/7-%E8%81%9A%E7%B1%BB+%E9%99%8D%E7%BB%B4

## 七、附录：源代码（带注释）

### github 仓库：

https://github.com/1190201115/HIT-machineLearningLab/tree/main/lab3

### main 函数：

```
%实验三,主函数
%%数据生成
num=200;

%标准数据
%{
k=4;
mean_all=[[1,1],[3,3],[1,3],[3,1]];
cov_all=[[0.2,0;0,0.2],[0.4,0.1;0.1,0.4],[0.3,0.15;0.15,0.3],[0.2,0.05;0.05,0.2]];
[data1,data2,data3,data4,class]=create_data(num,mean_all,cov_all);
X=[data1;data2;data3;data4];
%}

%%修改平均值，使分离
%{
k=4;
mean_all=[[1,1],[5,5],[1,5],[5,1]];
cov_all=[[0.2,0;0,0.2],[0.4,0.1;0.1,0.4],[0.3,0.15;0.15,0.3],[0.2,0.05;0.05,0.2]];
[data1,data2,data3,data4,class]=create_data(num,mean_all,cov_all);
X=[data1;data2;data3;data4];
%}
```

```
%%修改协方差矩阵，使样本集内部分散程度更高
%{
k=4;
mean_all=[[1,1],[3,3],[1,3],[3,1]];
cov_all=[[0.4,0;0,0.4],[0.8,0.2;0.2,0.8],[0.6,0.3;0.3,0.6],[0.4,0.1;0.
1,0.4]];
[data1,data2,data3,data4,class]=create_data(num,mean_all,cov_all);
X=[data1;data2;data3;data4];
%}

%%gmm 算法
%{
step=50;
mark=gmm(cov_all,mean_all,step,num*k,X,k);
draw(num*k,mark,X);
cal_accuracy(class,mark,num*k,1);
%}

%%kmeans 算法
%{
[center,mark]=kmeans(X,k,num*k);
plot(data1(:,1),data1(:,2),'o');
hold on;
plot(data2(:,1),data2(:,2),'o');
hold on;
plot(data3(:,1),data3(:,2),'o');
hold on;
plot(data4(:,1),data4(:,2),'o');
hold on;
plot(center(:,1),center(:,2),'p','MarkerFaceColor','r','MarkerSize',2
0);
cal_accuracy(class,mark,num*k,2);
%}
```

## 数据产生函数

```
function [ data1,data2,data3,data4,tag] =
create_data( num,mean_all,cov_all)
%产生混合高斯模型
mean1=mean_all(1,[1,2]);
mean2=mean_all(1,[3,4]);
mean3=mean_all(1,[5,6]);
mean4=mean_all(1,[7,8]);
```

```
cov1=cov_all([1,2],[1,2]);
cov2=cov_all([1,2],[3,4]);
cov3=cov_all([1,2],[5,6]);
cov4=cov_all([1,2],[7,8]);
data1=mvnrnd(mean1, cov1, num);
data2=mvnrnd(mean2, cov2, num);
data3=mvnrnd(mean3, cov3, num);
data4=mvnrnd(mean4, cov4, num);
tag1=ones(num,1);
tag2=tag1.*2;
tag3=tag1.*3;
tag4=tag1.*4;
tag=[tag1;tag2;tag3;tag4];
end
```

## kmeans 算法

```
function [center,class] = kmeans( X,k,num )
%kmeans 分类算法
%
center=cal_center(X,k,num);
[~,col]=size(center);
class=zeros(num,1);%存放每个点距离最近的中心点序号
get=0;
iteration=0
while get==0
      newcenter=zeros(k,col);
      tagnum=zeros(k,1);%记录每个区域有多少数据点
for i=1:num
    minlen=9999;
    tag=1;
    for j= 1:k
        len=cal_distance(X(i,:),center(j,:),col);
        if minlen>len
            minlen=len;
            tag=j;
        end
    end
    class(i)=tag;
    tagnum(tag)=tagnum(tag)+1;
end
for i=1:num
    newcenter(class(i),:)=newcenter(class(i),:)+X(i,:);
```

```
end
for i=1:k
    newcenter(i,:)=newcenter(i,:)./tagnum(i);
end
    if newcenter==center
        get=1;
        iteration
    end
    iteration=iteration+1;
        center=newcenter;
end
```

## 产生随机初始中心点

```
function [center ] = cal_center( X,k,num)
%产生初始中心点
% k 个中心，不能相同
[~,col]=size(X);
%产生随机且不同的 k 个中心点
center=zeros(k,col);
i=1;
while i<k+1
    temp=X(randi([1,num],1),:);
    if ismember(temp,center)
        continue;
    end
    center(i,:)=temp;
    i=i+1;
end
end
```

## 计算两点间距离

```
function [distance ] = cal_distance( x,y,col )
%计算两点间距离，为了方便只写了二维和三维的，如需要可以改写为 k 维的
if col==2
distance=sqrt((x(1)-y(1))^2+(x(2)-y(2))^2);
else
    distance=sqrt((x(1)-y(1))^2+(x(2)-y(2))^2+(x(3)-y(3))^2);
end
```

## 绘图函数

```
function [ output_args ] = draw( num,class,X )
%UNTITLED3 此处显示有关此函数的摘要
%   此处显示详细说明
for i=1:num
switch class(i)
        case 1
            plot(X(i,1),X(i,2),'or');
            hold on;
        case 2
            plot(X(i,1),X(i,2),'+g');
             hold on;

        case 3
            plot(X(i,1),X(i,2),'*b');
             hold on;
        case 4
            plot(X(i,1),X(i,2),'dk');
             hold on;
end
end

end
```

## gmm 算法函数

```
function [final_mark] = gmm( cov_all,mean_all,step,num,X,kind )
%gmm 算法

[~,col]=size(X);
total_prob=zeros(num,kind);
single_prob=zeros(1,kind);
possible=zeros(1,kind);
possible(1,1:kind)=0.25;
while step>0
    step=step-1;
    %每个样本对每个类别的响应度
    for i=1:num
        total_weight=0;
        for j=1:kind
            single_prob(j)=cal_probability( cov_all,mean_all,X,i,j);
            total_weight=total_weight+(1/kind)*single_prob(j);
        end
```

```
        for j=1:kind
            total_prob(i,j)=single_prob(j)/total_weight;
        end
    end
    for k=1:kind
        mulsum=zeros(1,col);
        sum=0;
        for j=1:num
            sum=sum+total_prob(j,k);
            mulsum=mulsum+total_prob(j,k)*X(j,:);
        end
        diff=X-repmat(mean_all(1,col*(k-1)+1:col*k),num,1);
        eye=zeros(num);
        for j=1:num
            eye(j,j)=total_prob(j,k);
        end
        cov_all(1:col,col*(k-1)+1:col*k)=diff'*eye*diff/sum;
        possible(k)=sum/num;
    end
    final_mark=zeros(num,1);
    for i=1:num
        max=0;
        tag=1;
        for k=1:kind
            weight=total_prob(i,k);
            if weight>max
                max=weight;
                tag=k;
            end
        end
        final_mark(i)=tag;
    end
end
```

计算概率函数

```
function [ probability ] =cal_probability( cov_all,mean_all,X,j,k )
%   计算第 j 个数据属于第 k 类的概率
[~,col]=size(X);
denominator=((2*pi)^(col/2))*sqrt(det(cov_all(1:col,col*(k-1)+1:col*k)
));
numerator=exp( -0.5* ( (X(j,:)-mean_all(1,col*(k-1)+1:col*k)) *
```

```
inv(cov_all(1:col,col*(k-1)+1:col*k)) *
(X(j,:)-mean_all(1,col*(k-1)+1:col*k))') );
probability=numerator/denominator;
end
```

计算聚类准确度：

```
function [ accuracy ] = cal_accuracy(class,mark,num,needchange)
%计算聚类的准确度。由于聚类后的分类标签顺序未必与分类前一致，故写了一个简单的算法使其对应，传入参数 needchange 为 2 时表示需要对应。
right=0;
kind=class(num);
if needchange==2
num_now=class(1);
n=1;
exchange=zeros(kind,1);
count=zeros(kind,1);
    max=1;
    maxtag=1;
while n<=num
    if class(n)==num_now
        count(mark(n))=count(mark(n))+1;
        n=n+1;
    else
    for k=1:kind
        if count(k)>max
            max=count(k);
            maxtag=k;
        end
    end
    exchange(num_now)=maxtag;
    num_now=class(n);
    count=zeros(kind,1);
    max=1;
    maxtag=1;
    continue;
    end
end
for k=1:kind
        if count(k)>max
            max=count(k);
            maxtag=k;
```

```
        end
    end
    exchange(num_now)=maxtag;
for i=1:num
   class(i)=exchange(class(i));
end

end
a=class-mark;
for i=1:num
    if a(i)==0
        right=right+1;
    end
end
accuracy=right/num
end
```