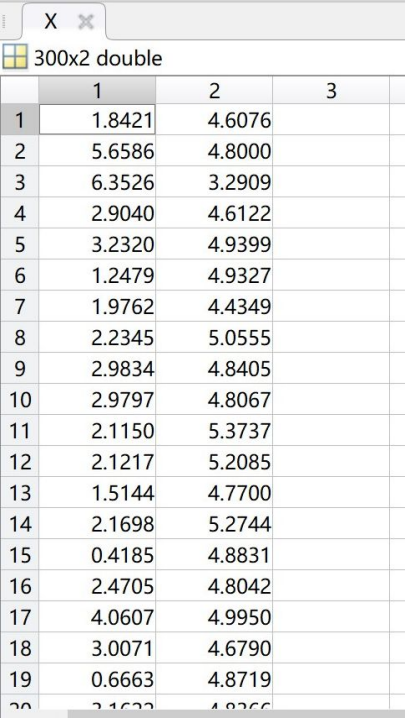


TP K-Means

1. 输入

X的输入：X代表所有要分类的点，X(:,1)代表横坐标，X(:,2)代表纵坐标。

> load('ex7data2.mat') :



	1	2	3
1	1.8421	4.6076	
2	5.6586	4.8000	
3	6.3526	3.2909	
4	2.9040	4.6122	
5	3.2320	4.9399	
6	1.2479	4.9327	
7	1.9762	4.4349	
8	2.2345	5.0555	
9	2.9834	4.8405	
10	2.9797	4.8067	
11	2.1150	5.3737	
12	2.1217	5.2085	
13	1.5144	4.7700	
14	2.1698	5.2744	
15	0.4185	4.8831	
16	2.4705	4.8042	
17	4.0607	4.9950	
18	3.0071	4.6790	
19	0.6663	4.8719	
20	2.1632	4.8266	

K的输入：一般会预先给定要分类的个数K。

> K=3 ;

随机3个类的重心(centroide)的输入：手动输入。

> initialisation_centroides = [3 3; 6 2; 8 5] ;

2. 给每一个数据点分类

affection_classes(X, centroides)

输入的参数：一个包含所有点数据的X矩阵(300*2)

一个包含了当前重心的矩阵(3*2)

返回值：一个idx向量(300*1)代表每一个数据点所被分配到的cluster的编号(1,2,3)

%%%%%%%%%%函数开始%%%%%%%%%

>function idx = affection_classes(X, centroides)

% 通过重心矩阵的行数可以得知K的个数且size(A)=[行数, 列数]。

>K = size(centroides, 1);

% 新建idx向量, 大小为 (X向量的行数(即输入所有点的个数=300), 1)

>idx = zeros(size(X, 1), 1);

% 遍历X中的每一个点

>for i = 1:size(X, 1)

 % $s_i = \operatorname{argmin} D^2(x_i, u_k) = (x_i(1) - u_k(1))^2 + (x_i(2) - u_k(2))^2$

 % 返回向量中非零元素的位置 find(A) 或 返回满足某条件的值find(A**)

 % sum(A,2) 代表矩阵每一行的总和 %sum(A) 代表矩阵每一列的总和

 % min(A) 返回一个数组中的最小值

 % 故此处find返回的是 X中每一个点到三个重心中最近的重心的重心编号。

 q = find(sum((ones(K, 1)*X(i,:) - centroides).^2, 2) ==
 min(sum((ones(K, 1)*X(i,:) - centroides).^2, 2)));

 idx(i) = q;

>end

>end

%%%%%%%%%%函数结束%%%%%%%%%

% ones(K,1)*X(i,:) 用于取出X中每一行的数据, 并且复制K=3遍

>ones(K, 1)*X(i,:)

ans =

1.8421 4.6076

1.8421 4.6076

1.8421 4.6076

% 将复制了三遍的X中的某一行(某一个点)分别与K个重心做欧式距离

>ones(3, 1)*X(1,:) - initialisation_centroides

ans =

-1.1579 1.6076

-4.1579 2.6076

-6.1579 -0.3924

3.重新定义重心

actualisation_centroides(X, idx, K)

输入的参数：一个包含所有点数据的X矩阵(300*2)；
一个包含了所有点所在的类的编号(300*1)；
K个类。

输出值：根据已有的类(已包含与其对应的数据点)，重新计算后得到的每个类的重心。

```
%%%%%%%%%%%%函数开始%%%%%%%%%
```

```
>function centroides = actualisation_centroides(X, idx, K)
```

```
% 获取X的shape=(300, 2)。
```

```
>[m n] = size(X);
```

```
% 初始化输出，K行，n=2列。
```

```
>centroides = zeros(K,n);
```

```
% 遍历K=3个类
```

```
>for k = 1:K
```

```
    % 获取idx向量中等于k{1, 2, 3}的index，存入p中，所以p也是一个向量。
```

```
    p = find(idx==k)
```

```
    % 计算新的  $u_k(x,y) = (1/N_k) * \sum(x_i)$ 
```

```
    centroides(k,1:n) = 1/length(p)*sum(X(p,:),1)
```

```
>end
```

```
>end
```

```
%%%%%%%%%%%%函数结束%%%%%%%%%
```

```
% 实例化 X(p,:,1)
```

```
>X([3,4,5],:,1)
```

```
ans =
```

```
    6.3526    3.2909
```

```
    2.9040    4.6122
```

```
    3.2320    4.9399
```

```
% 实例化 sum(X(p,:,1)) 按列相加
```

```
>sum(X([3,4,5],:,1))
```

```
ans =
```

```
    12.4886    12.8430
```

4.K-Means算法

4.1 K-means主函数：

lancement_Kmeans(X, initialisation_centroides, max_iters, plot_progress)

输入的参数：一个包含所有点数据的X矩阵(300*2)；

一个包含了手动输入的初始重心坐标的矩阵(3*2)

最大循环次数

输出的值：[centroides, idx] 重心坐标 与 每个点的分类组成的数组(300*1)

```
%%%%%%%%%%函数开始%%%%%%%%%
```

```
% 为plot progress设置默认值。
```

```
% exist函数主要有两种形式，作用都是用于确定某值是否存在：
```

```
% 一个参数时，exist(A)=B, 若A存在则B=1。
```

```
% 两个参数时，exist('name', 'kind') 且 kind 表示 name 的类型。k=builtin,class,dir(文件夹),file,var。
```

```
%若不存在plot_progress作为var变量或者plot_progress为空
```

```
> if ~exist('plot_progress', 'var') || isempty(plot_progress)
    plot_progress = false;
> end
```

```
% 如果plot_progress为true时，作图
```

```
> if plot_progress
```

```
    % 新建窗口。
```

```
    figure;
```

```
    %使得之后要做的图 依然在同一个窗口上。
```

```
    hold on;
```

```
> end
```

```
% 初始化变量
```

```
> [m n] = size(X);
```

```
% K只取初始的重心的行数=3
```

```
> K = size(initialisation_centroides, 1);
```

```
> centroides = initialisation_centroides;
```

```
> previous_centroids = centroides;
```

```
% 初始化每个点对应的类的编号为0, idx=(300*1)
```

```
> idx = zeros(m, 1);
```

```
% 运行K-Means算法
```

```
%循环次数不超过10次
```

```
> for i=1:max_iters
```

```
    % Output输出到终端的代码
```

```
    fprintf('K-Means iteration %d/%d...\n', i, max_iters);
```

```
% 如果存在“OCTAVE_VERSION”作为变量。OCTAVE是一个类MATLAB软件。
```

```
> if exist('OCTAVE_VERSION')
```

```
    %在printf()后使用fflush(stdout)的作用是立刻清空输出缓冲区，并把缓冲区内容输出。
```

```
    fflush(stdout);
```

```

➤ end
% 对于每一个X中的数据点，计算初它所属的类
➤ idx = affectation_classes(X, centroides);

➤ if plot_progress
    % 将分好类的点用不同颜色标出，画出当前的中心坐标，与之前的重心坐标的位置移动。
    plotProgresskMeans(X, centroides, previous_centroids, idx, K, i);
    previous_centroids = centroides;
    fprintf('Press enter to continue.\n');
    pause;
➤ end

% 基于当前每个类中的点，计算出新的重心坐标
➤ centroides = actualisation_centroides(X, idx, K);
➤ end

% 当上面的循环结束且plot_progress的值为true，则结束作图。
➤ if plot_progress
    hold off;
➤ end
➤ end

%%%%%%%%%%函数结束%%%%%%%%%

```

4.2 K-Means主函数用到的PLOT函数

plotProgresskMeans(X, centroides, previous, idx, K, i)

```

%%%%%%%%%%函数开始%%%%%%%%%

➤ function plotProgresskMeans(X, centroides, previous, idx, K, i)
% 本函数会将属于不同类的数据点用不同颜色打印。
% 当前循环 i 次之后的类的重心 与 循环 i-1次的类的重心之间，会做一根线，表示重心位置的转移。

% 用不同颜色 打印 属于不同类的 数据点。(即同一个类的点的颜色相同)。
➤ plotDataPoints(X, idx, K);

% Plot the centroids as black x's
➤ plot(centroides(:,1), centroides(:,2), 'x', 'MarkerEdgeColor','k', 'MarkerSize', 10, 'LineWidth', 3);
% Plot the history of the centroids with lines
➤ for j=1:size(centroides,1)
    drawLine(centroides(j, :), previous(j, :));
➤ end

% 标题
➤ title(sprintf('Iteration number %d', i))

```

```
> end
```

```
%%%%%%%%%%函数结束%%%%%%%%%
```

4.3 PLOT函数用到的drawLine(p1, p2, varargin)函数

```
> function drawLine(p1, p2, varargin)
```

```
% 本函数在点p1与p2之间画一条线
```

```
> plot([p1(1) p2(1)], [p1(2) p2(2)], varargin{:});
```

```
> end
```

4.4 PLOT函数用到的plotDataPoints(X, idx, K)函数

```
> function plotDataPoints(X, idx, K)
```

```
% 本函数会将属于不同类的数据点用不同颜色打印，相同类的点用相同的颜色打印。
```

```
% 创建画板。
```

```
> palette = hsv(K + 1);
```

```
> colors = palette(idx, :);
```

```
% 打印散点。
```

```
> scatter(X(:,1), X(:,2), 15, colors);
```

```
> end
```

5.K-Means的main()

```
%%%%%%%%%%Program开始%%%%%%%%%%
```

```
% 导入数据
```

```
>load('ex7data2.mat');
```

```
% 参数 : 类的个数K以及最大循环次数10
```

```
>K = 3;
```

```
>max_iters = 10;
```

```
%此处手动定义各个类的重心坐标, 但是我们可以用例如: tirage aleatoire来自动定义。
```

```
>initialisation_centroides = [3 3; 6 2; 8 5];
```

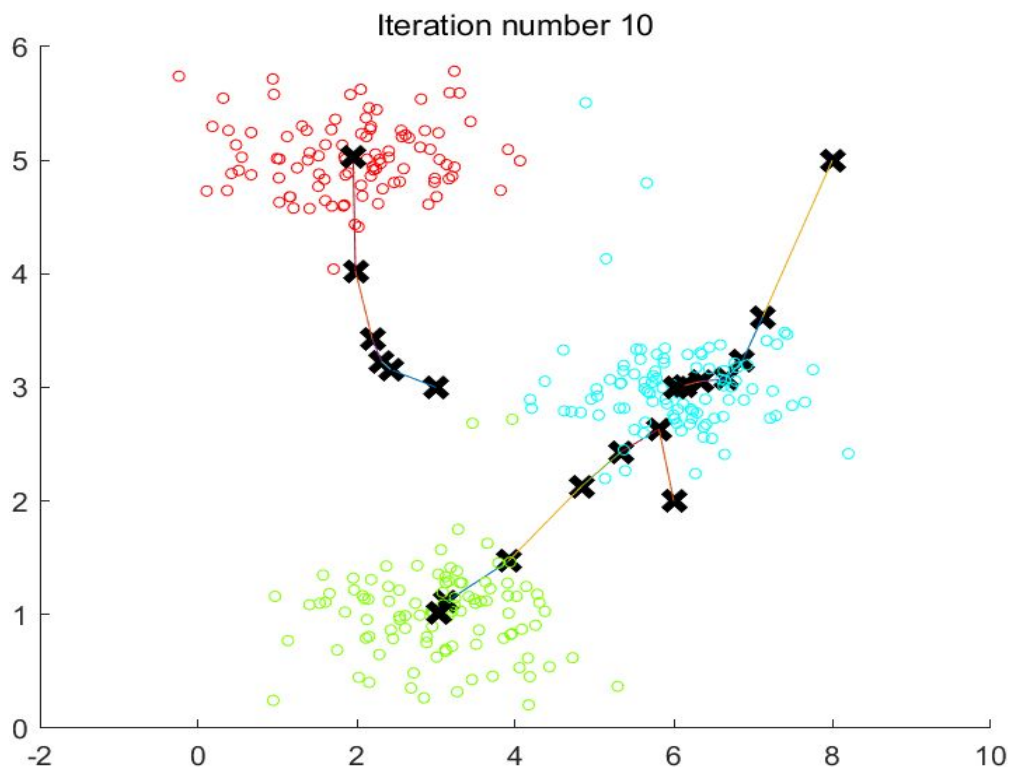
```
% 运行K-Means算法。参数“true”使得我们可以看到运行时的过程(类的重心的演变, 类的演变等等)
```

```
>[centroides, idx] = lancement_Kmeans(X, initialisation_centroides, max_iters, true);
```

```
>fprintf('Program paused. Press enter to continue.\n');
```

```
>pause;
```

```
%%%%%%%%%%Program结束%%%%%%%%%%
```



6.Mesure de silhouette(侧影判断分类好坏)

6.1 侧影主程序

%%%%%%%%%%Progam开始%%%%%%%%%

% 初始化侧影距离，每一个数据点对应一个距离，(300*1)

➤ `distance_measure=zeros(size(X,1));`

% 循环每一个数据点 i

➤ `for i=1:size(X,1)`

 % 数据点 i 到 数据点(1..i-1) 之间都进行距离计算

➤ `for j=1:i-1`

 % $\sqrt{\sum((x_i(1)-x_j(1))^2 + (x_i(1)+x_j(1))^2)}$ 欧式距离

`distance_measure(i,j)=sqrt(sum((X(i,:)-X(j,:)).^2));`

➤ `end`

➤ `end`

% 循环每一个数据点 i

➤ `for i=1:size(X,1)`

 % 数据点 i 到 数据点(j=i+1..j=300) 之间都进行距离计算

➤ `for j=i+1:size(X,1)`

 % 由于已经计算过 i=(1..300) 到 j=(1 .. i-1)之间的距离，此处要计算的 i=(1..300) 到 j=(i+1 .. 300)之间的距离 = 之前计算过的 `distance_measure(j,i)`

`distance_measure(i,j)=distance_measure(j,i);`

➤ `end`

➤ `end`

%%%%%%%%%%Exemple举例分析%%%%%%%%%

假设一共三个数据点 x_1, x_2, x_3 。

做第一个循环，求 i=(1..300) 到 j=(1 .. i-1)之间的距离：

i=1, j=(1,0) 故 j 取不到值。

i=2, j=(1,1) 故j=1, 求得 `distance_measure(2,1)`。

i=3, j=(1,2) 故j=1或2, 求得 `distance_measure(3,1)` 与 `distance_measure(3,2)`。

做第二个循环，求 i=(1..300) 到 j=(i+1 .. 300)之间的距离：

i=1, j=(2,3) 故j=2或3, 求得 `distance_measure(1,2)` 与 `distance_measure(1,3)`。

i=2, j=(3,3) 故j=3, 求得 `distance_measure(2,3)`。

i=3, j=(4,3) 故 j 取不到值。

所以两个循环最后的到的距离答案是一致的，只是颠倒了起点与终点。

%%%%%%%%%%Exemple结束%%%%%%%%%

% 初始化两个大小为(300*1)的数组。

➤ `tab_a=zeros(size(X,1),1);` % 关于点p(l)到类内其他点的平均距离。

➤ `tab_b=zeros(size(X,1),1);` % 从点 p(l) 到其他的类的最小距离。

% 遍历每一个类。

➤ for i=1:K

% ismember(a,b) 用于 矩阵a中的数是不是矩阵b中的成员，是的话结果返回1，不是返回0。

% 将 ismember(idx,i) 的结果求和，即对1的求和，最终得到关于第 i 类的cardinal值。

number_elements_same_class=sum(ismember(idx,i));

% 将 p 赋值为一个 由数据点索引(属于第 i 类) 组成的向量。

p=find(idx==i);

% 对于p中的每一个数据点(每一个属于第 i 类的数据点)进行操作。

➤ for l=1:length(p)

% 给tab_a中的第p(l)项赋值为 将第p(l)个点到p=(1..length(p))个点之间的距离一一相加后，

除以 该类的 (cardinal-1), 求得 关于点p(l)到类内其他点的平均距离。

tab_a(p(l))=sum(distance_measure(p(l),p))/(number_elements_same_class-1);

➤ end

➤ end

% 遍历每一个类。

➤ for i=1:K

% 将 p 赋值为一个 由数据点索引(属于第 i 类) 组成的向量。

p=find(idx==i);

% 初始化一个 val向量，大小为(length(p), K-1)。

val=zeros(length(p),K-1);

inc=1;

% 再遍历每一个类。

➤ for j=1:K

% 若 j 和 i 不是同一个类

➤ if j~=i

% 求出当前第 j 类的点的个数。

number_elements_current_class_j=sum(ismember(idx,j));

% 将 q 赋值为一个 由数据点索引(属于第 j 类) 组成的向量。

q=find(idx==j);

% 遍历第 i 类中的每一个数据点。

➤ for l=1:length(p)

% 计算 (第 i 类中的p(l)点 到 q 中所有点的距离之和) / 第j类的cardinal = 类间平均距离

val(l,inc)=sum(distance_measure(p(l),q))/(number_elements_current_class_j);

➤ end

inc=inc+1;

➤ end

➤ end

% 定义tab_b为从点 p(l) 到其他的类的最小距离。

tab_b(p)=min(val,[],2);

➤ end

% mesure silhouette的公式： $s(i) = (b(i) - a(i)) / \max(a(i), b(i))$ 。

➤ *s=(tab_b-tab_a)./(max(tab_a,tab_b)+0.000001);*

%%%%%%%%%%Progam结束%%%%%%%%%

6.2 可视化侧影

%%%%%%%%%%Program开始%%%%%%%%%%

% 用于重新定义s矩阵为一个数组(降维)。

> reshape_s=[];

> for i=1:K

 % 将 p 赋值为一个 由数据点索引(属于第 i 类) 组成的向量。

 p=find(idx==i);

 % 在每一个类中加入两个值 0,0用于区分类。再将每一个类的s值按降序重新排序，获得新的reshape_s。

 reshape_s=[reshape_s;0;0;sort(s(p),'descend')];

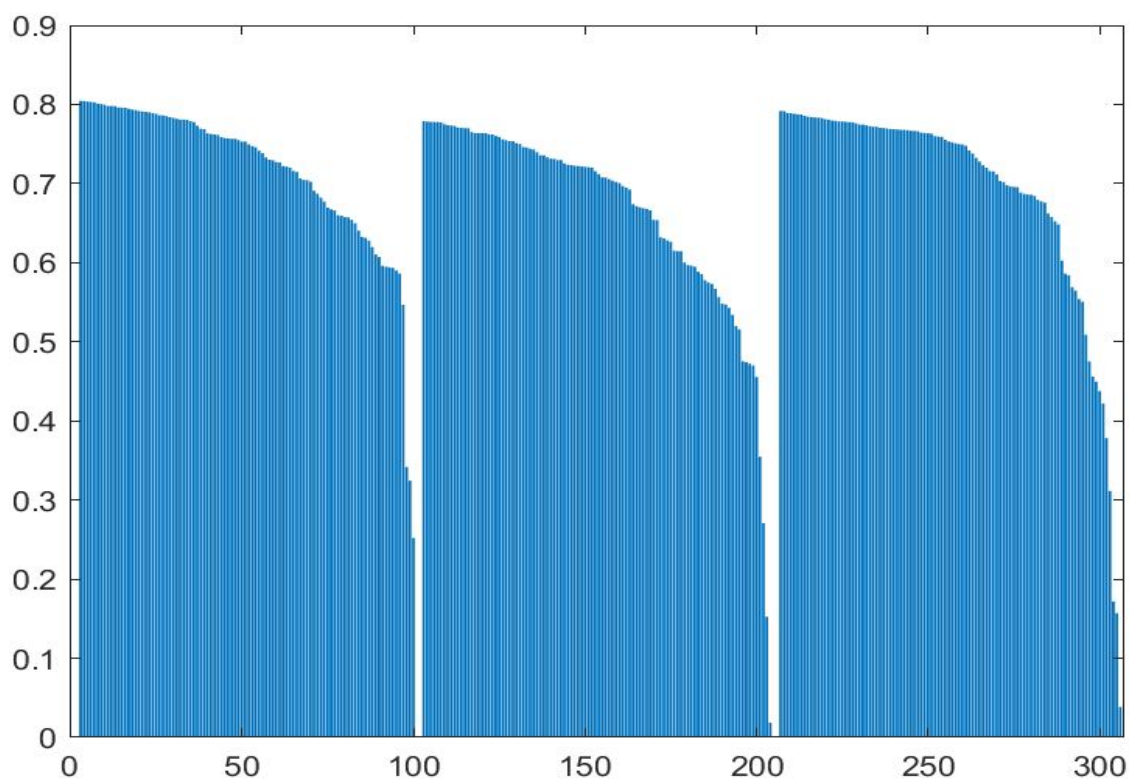
> end

> figure;

% 用柱状图作出reshape_s数组。

> bar(reshape_s);

%%%%%%%%%%Program结束%%%%%%%%%%



7.K-Means应用于压缩图片

%%%%%%%%%%Program开始%%%%%%%%%

% 以double的形式导入图片。A是三维的，图片的长(pixels)，宽(pixels)和高(pixels)。

➤ *A = double(imread('desert.jpg'));*

% 把得到的每一个double值除以255，使其归一化，即(0, 1)之间。

➤ *A = A / 255;*

% 图片的大小就是 三维矩阵A 的大小: (321,481,3)。

➤ *img_size = size(A);*

% 把A矩阵reshape成 X矩阵。X的大小是，(N*3)，期中N代表pixels数，3代表R,G,B。(此处将A降维到二维)

➤ *X = reshape(A, img_size(1) * img_size(2), 3);*

% 初始化类的个数；

% 初始化最大循环次数；

% 随机初始化两个类的重心坐标。

➤ *K = 2;*

➤ *max_iters = 10;*

➤ *initialisation_centroides = initialisation_aleatoire_centroides(X, K);*

% 运行KMeans算法

➤ *[centroides, idx] = lancement_Kmeans(X, initialisation_centroides, max_iters);*

%%%%%%%%%%Program结束%%%%%%%%%

%%%%%%%%%%Program开始%%%%%%%%%

➤ *fprintf("\n Application a la compression \n\n");*

% 给图片中的每个数据点关于 类的重心坐标 进行分类。

➤ *idx = affectation_classes(X, centroides);*

%定义 X_recovered 为每一个数据点对应的重心坐标，大小为(154401,3)，共321*481=154401个数据点。

➤ *X_recovered = centroides(idx,:);*

% 再把 X_recovered 从 (154401,3) 导回和原图尺寸一致的大小：(321,481,3)。

➤ *X_recovered = reshape(X_recovered, img_size(1), img_size(2), 3);*

% 画出初始图。

➤ *figure;*

% 一行两列的左边第一个。

➤ *subplot(1, 2, 1);*

% imagesc会对图像进行适当的缩放（显示出来的尺寸大小）。

➤ *imagesc(A);*

➤ *title('Image originale');*

```
% 画出压缩后的图片。
```

```
% 一行两列的左边第二个。
```

```
➤ subplot(1, 2, 2);
```

```
➤ imagesc(X_recovered)
```

```
➤ title(sprintf('Version compressee avec %d couleurs.', K));
```

```
%%%%%%%%%%%%Progam结束%%%%%%%%
```

