# EM算法

# 一、[EM算法](https://www.cnblogs.com/pinard/p/6912636.html)定义

EM算法也称期望最大化（Expectation-Maximum,简称EM）算法，它是一个基础算法，是很多机器学习领域算法的基础，比如隐式马尔科夫算法（HMM）， LDA主题模型的变分推断等等。

## 1、极大似然估计

### （1）举例说明：经典问题——学生身高问题

　　我们需要调查我们学校的男生和女生的身高分布。 假设你在校园里随便找了100个男生和100个女生。他们共200个人。将他们按照性别划分为两组，然后先统计抽样得到的100个男生的身高。假设他们的身高是服从高斯分布的。但是这个分布的均值u和方差∂2我们不知道，这两个参数就是我们要估计的。记作θ=[u, ∂]T。

　　问题：我们知道样本所服从的概率分布的模型和一些样本，而不知道该模型中的参数。



　　我们已知的有两个：（1）样本服从的分布模型（2）随机抽取的样本  需要通过极大似然估计求出的包括：模型的参数

　　总的来说：极大似然估计就是用来估计模型参数的统计学方法。

### （2）如何估计

问题数学化： （1）样本集X={x1,x2,…,xN} N=100 （2）概率密度：p(xi|θ)抽到男生i（的身高）的概率 100个样本之间独立同分布，所以我同时抽到这100个男生的概率就是他们各自概率的乘积。就是从分布是p(x|θ)的总体样本中抽取到这100个样本的概率，也就是样本集X中各个样本的联合概率，用下式表示：https://images2015.cnblogs.com/blog/797505/201604/797505-20160401125104144-1825220157.png

这个概率反映了，在概率密度函数的参数是θ时，得到X这组样本的概率。 需要找到一个参数θ，其对应的似然函数L(θ)最大，也就是说抽到这100个男生（的身高）概率最大。这个叫做θ的最大似然估计量，记为

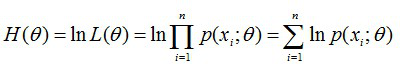
https://images2015.cnblogs.com/blog/797505/201604/797505-20160401125217441-826849783.png

### （3）求最大似然函数估计值的一般步骤

　　首先，写出似然函数：

https://images2015.cnblogs.com/blog/797505/201604/797505-20160401125318285-1823853438.png

　　其次，对似然函数取对数，并整理：



　　然后，求导数，令导数为0，得到似然方程；

　　最后，解似然方程，得到的参数即为所求。

### （4）总结

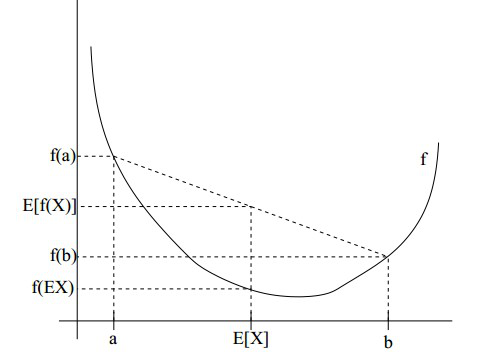
 　　多数情况下我们是根据已知条件来推算结果，而极大似然估计是已经知道了结果，然后寻求使该结果出现的可能性最大的条件，以此作为估计值。

## 2、Jensen不等式

### （1）定义

设f是定义域为实数的函数，如果对于所有的实数x。如果对于所有的实数x，f(x)的二次导数大于等于0，那么f是凸函数。  Jensen不等式表述如下：      如果f是凸函数，X是随机变量，那么：E[f(X)]>=f(E[X])  。当且仅当X是常量时，上式取等号。

### （2）举例



图中，实线f是凸函数，X是随机变量，有0.5的概率是a，有0.5的概率是b。X的期望值就是a和b的中值了，图中可以看到E[f(X)]>=f(E[X])成立。Jensen不等式应用于凹函数时，不等号方向反向。

# 二、EM算法要解决的问题

　　我们经常会从样本观察数据中，找出样本的模型参数。 最常用的方法就是极大化模型分布的对数似然函数。

　　但是在一些情况下，我们得到的观察数据有未观察到的隐含数据，此时我们未知的有隐含数据和模型参数，因而无法直接用极大化对数似然函数得到模型分布的参数。怎么办呢？这就是EM算法可以派上用场的地方了。

　　EM算法解决这个的思路是使用启发式的迭代方法，既然我们无法直接求出模型分布参数，那么我们可以先猜想隐含数据（EM算法的E步），接着基于观察数据和猜测的隐含数据一起来极大化对数似然，求解我们的模型参数（EM算法的M步)。由于我们之前的隐藏数据是猜测的，所以此时得到的模型参数一般还不是我们想要的结果。不过没关系，我们基于当前得到的模型参数，继续猜测隐含数据（EM算法的E步），然后继续极大化对数似然，求解我们的模型参数（EM算法的M步)。以此类推，不断的迭代下去，直到模型分布参数基本无变化，算法收敛，找到合适的模型参数。

从上面的描述可以看出，EM算法是迭代求解最大值的算法，同时算法在每一次迭代时分为两步，E步和M步。一轮轮迭代更新隐含数据和模型分布参数，直到收敛，即得到我们需要的模型参数。

# 三、与k-means算法比较

　　一个最直观了解EM算法思路的是K-Means算法，见之前写的[K-Means聚类算法](http://www.cnblogs.com/pinard/p/6164214.html)。在K-Means聚类时，每个聚类簇的质心是隐含数据。我们会假设KK个初始化质心，即EM算法的E步；然后计算得到每个样本最近的质心，并把样本聚类到最近的这个质心，即EM算法的M步。重复这个E步和M步，直到质心不再变化为止，这样就完成了K-Means聚类。

　　当然，K-Means算法是比较简单的，实际中的问题往往没有这么简单。上面对EM算法的描述还很粗糙，我们需要用数学的语言精准描述。

# 四、EM算法流程

　　现在我们总结下EM算法的流程。

　　输入：观察数据x=(x(1),x(2),...x(m))x=(x(1),x(2),...x(m))，联合分布p(x,z|θ)p(x,z|θ), 条件分布p(z|x,θ)p(z|x,θ), 最大迭代次数JJ。

　　1) 随机初始化模型参数θθ的初值θ0θ0。

　　2） for j  from 1 to J开始EM算法迭代：

　　a) E步：计算联合分布的条件概率期望：

Qi(z(i))=P(z(i)|x(i)，θj))Qi(z(i))=P(z(i)|x(i)，θj))

L(θ,θj)=∑i=1m∑z(i)Qi(z(i))logP(x(i)，z(i)|θ)L(θ,θj)=∑i=1m∑z(i)Qi(z(i))logP(x(i)，z(i)|θ)

　　b) M步：极大化L(θ,θj)L(θ,θj),得到θj+1θj+1:

θj+1=argmaxθL(θ,θj)θj+1=argmaxθL(θ,θj)

　　c) 如果θj+1θj+1已收敛，则算法结束。否则继续回到步骤a)进行E步迭代。

　　输出：模型参数θθ。

# 五. EM算法的收敛性思考

　　EM算法的流程并不复杂，但是还有两个问题需要我们思考：

　　1） EM算法能保证收敛吗？

　　2） EM算法如果收敛，那么能保证收敛到全局最大值吗？

　　首先我们来看第一个问题, EM算法的收敛性。要证明EM算法收敛，则我们需要证明我们的对数似然函数的值在迭代的过程中一直在增大。即：

∑i=1mlogP(x(i)|θj+1)≥∑i=1mlogP(x(i)|θj)∑i=1mlogP(x(i)|θj+1)≥∑i=1mlogP(x(i)|θj)

　　由于

L(θ,θj)=∑i=1m∑z(i)P(z(i)|x(i)，θj))logP(x(i)，z(i)|θ)L(θ,θj)=∑i=1m∑z(i)P(z(i)|x(i)，θj))logP(x(i)，z(i)|θ)

　　令：

H(θ,θj)=∑i=1m∑z(i)P(z(i)|x(i)，θj))logP(z(i)|x(i)，θ)H(θ,θj)=∑i=1m∑z(i)P(z(i)|x(i)，θj))logP(z(i)|x(i)，θ)

　　上两式相减得到：

∑i=1mlogP(x(i)|θ)=L(θ,θj)−H(θ,θj)∑i=1mlogP(x(i)|θ)=L(θ,θj)−H(θ,θj)

　　在上式中分别取θθ为θjθj和θj+1θj+1，并相减得到：

∑i=1mlogP(x(i)|θj+1)−∑i=1mlogP(x(i)|θj)=[L(θj+1,θj)−L(θj,θj)]−[H(θj+1,θj)−H(θj,θj)]∑i=1mlogP(x(i)|θj+1)−∑i=1mlogP(x(i)|θj)=[L(θj+1,θj)−L(θj,θj)]−[H(θj+1,θj)−H(θj,θj)]

　　要证明EM算法的收敛性，我们只需要证明上式的右边是非负的即可。

　　由于θj+1θj+1使得L(θ,θj)L(θ,θj)极大，因此有:

L(θj+1,θj)−L(θj,θj)≥0L(θj+1,θj)−L(θj,θj)≥0

　 而对于第二部分，我们有：

H(θj+1,θj)−H(θj,θj)=∑i=1m∑z(i)P(z(i)|x(i)，θj)logP(z(i)|x(i)，θj+1)P(z(i)|x(i)，θj)≤∑i=1mlog(∑z(i)P(z(i)|x(i)，θj)P(z(i)|x(i)，θj+1)P(z(i)|x(i)，θj))=∑i=1mlog(∑z(i)P(z(i)|x(i)，θj+1))=0(3)(4)(5)(3)H(θj+1,θj)−H(θj,θj)=∑i=1m∑z(i)P(z(i)|x(i)，θj)logP(z(i)|x(i)，θj+1)P(z(i)|x(i)，θj)(4)≤∑i=1mlog(∑z(i)P(z(i)|x(i)，θj)P(z(i)|x(i)，θj+1)P(z(i)|x(i)，θj))(5)=∑i=1mlog(∑z(i)P(z(i)|x(i)，θj+1))=0

　　其中第（4）式用到了Jensen不等式，只不过和第二节的使用相反而已，第（5）式用到了概率分布累积为1的性质。

　　　　至此，我们得到了：∑i=1mlogP(x(i)|θj+1)−∑i=1mlogP(x(i)|θj)≥0∑i=1mlogP(x(i)|θj+1)−∑i=1mlogP(x(i)|θj)≥0, 证明了EM算法的收敛性。

　　从上面的推导可以看出，EM算法可以保证收敛到一个稳定点，但是却不能保证收敛到全局的极大值点，因此它是局部最优的算法，当然，如果我们的优化目标L(θ,θj)L(θ,θj)是凸的，则EM算法可以保证收敛到全局最大值，这点和梯度下降法这样的迭代算法相同。至此我们也回答了上面提到的第二个问题。

# 六、EM主要缺点

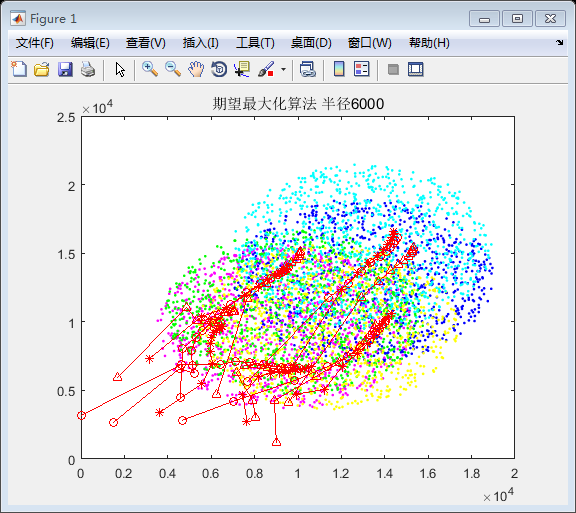
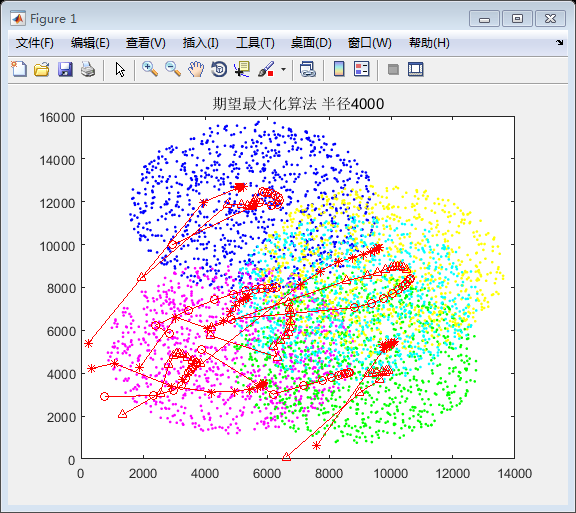
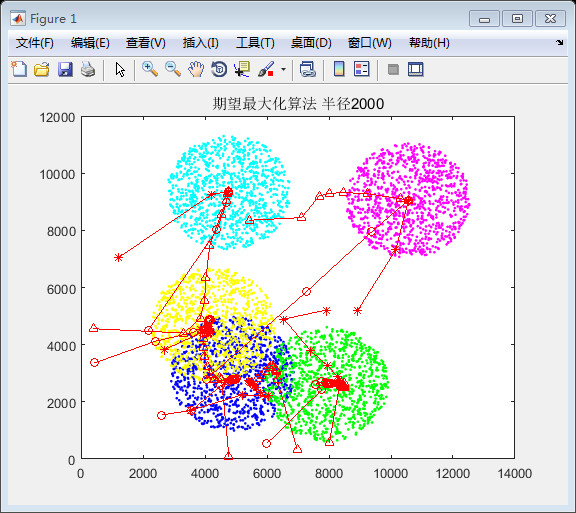
收敛速度慢;

算法高度依赖初始值的选择.

# 七、数据集描述及实验结果

1. 数据初始化  
   1.1 随机生成5个中心点  
   1.2 以这5个中心点为圆心，RADIUS为半径内，随机生成1000个点
2. EM算法主体  
   2.1 在图中，另外再取5个随机的初始点  
   2.2 对于所有5000个点，计算它相对5个初始点的距离，保存到 *5000 \* 5* 的矩阵中 (Expectation)  
   2.3 把所有5000个点按照 *相对5个初始点的距离* 归成5类 (Maximization)  
   2.4 计算5类中，所有点的坐标的平均值，得到5个新的点，作为下次迭代的5个初始点  
   2.5 从2.2开始迭代，直到收敛  
   2.6 把n次迭代得到的点连成线

## 迭代2，计算不同随机初始点的收敛结果



# 八、总结

1、分类的效果还是比较显著的

2、聚类结果总是能收敛的

3、圆形点阵比较分散的时候，无论随机起始点在哪里，总是能收敛到相同的位置

4、当圆形点阵比较接近的时候，可能会出现若干个局部极点，随机的起始点的位置一定程度上可以影响最后收敛的位置

5、一般迭代规模若干次就能够收敛

代码

% global parameter which can be altered easily to run a different result

WIDTH = 10000;

HEIGHT = 10000;

RADIUS = 6000;

ITERATE\_COUNT = 10;

% generate the initial points

center\_points = [];

all\_points = [];

color = ['b', 'm', 'g', 'y', 'c'];

path = ['\*', 'o', '^'];

for i = 1:5

center\_point = [randi(WIDTH) + RADIUS, randi(HEIGHT) + RADIUS];

center\_points(i, :) = center\_point;

for j = 1:1000

while (true)

point = [randi(WIDTH + RADIUS \* 2), randi(HEIGHT + RADIUS \* 2)];

if (sqrt((point(1) - center\_point(1))^2 + (point(2) - center\_point(2))^2) <= RADIUS)

if (length(all\_points) == 0)

all\_points(1, :) = point;

break;

end

all\_points(length(all\_points(:,1)) + 1, :) = point;

break;

end

end

plot(point(1), point(2), [color(i), '.']);

hold on;

end

end

% calculate three paths with three independent initial start points

for m = 1:3

% init the start points

res = [];

for i = 1:5

point = [randi(WIDTH), randi(HEIGHT)];

res(1, i, :) = point;

end

% iterate

distance = [];

for i = 1:ITERATE\_COUNT

s = size(res);

% cur\_center is a 3-dimension matrix

% the length of first dimension is 1

cur\_center = res(s(1), :, :);

for j = 1:length(all\_points);

for k = 1: s(2);

point\_1 = all\_points(j, :);

point\_2 = [cur\_center(1, k, 1), cur\_center(1, k, 2)];

distance(j, k) = sqrt((point\_1(1) - point\_2(1))^2 + (point\_1(2) - point\_2(2))^2);

end

end

% assign each point to the new defined category

assign = zeros(length(distance), 1);

for j = 1:s(2)

temp = distance(:, j);

smallest = sort(temp);

smallest = smallest(1:10);

for l=1:length(smallest)

for k = 1:length(temp)

if (smallest(l) == temp(k))

assign(k) = j;

end

end

end

end

for j = 1:length(distance)

if (assign(j) ~= 0)

continue;

end

[value, index] = min(distance(j, :));

assign(j) = index;

end

% calculate the total value of x, y for each category

temp2 = zeros(s(2), 3);

for j = 1:length(assign)

temp2(assign(j), 3) = temp2(assign(j), 3) + 1;

temp2(assign(j), 1) = temp2(assign(j), 1) + all\_points(j, 1);

temp2(assign(j), 2) = temp2(assign(j), 2) + all\_points(j, 2);

end

% get the new local center

for j = 1:s(2)

res(s(1) + 1, j, :) = [temp2(j, 1) / temp2(j, 3), temp2(j, 2) / temp2(j, 3)];

end

end

% plot the local center changing path

s = size(res);

for i = 1:s(2)

x = res(1:s(1), i, 1);

y = res(1:s(1), i, 2);

plot(x, y, ['r', path(m), '-']);

hold on;

end

end

% save the picture

title(['ÆÚÍû×î´ó»¯Ëã·¨ °ë¾¶', num2str(RADIUS)]);

saveas(gcf, ['em\_', num2str(RADIUS), '.jpg'])

hold off;