高斯混合模型（GMM）参数优化及实现

龚 勋 （2010-11-13）

1 高斯混合模型概述

高斯密度函数估计是一种参数化模型。有单高斯模型（Single Gaussian Model, SGM）和高斯混合模型（Gaussian mixture model，GMM）两类。类似于聚类，根据高斯概率密度函数（PDF，见公式 1）参数的不同，每一个高斯模型可以看作一种类别，输入一个样本 $\boldsymbol{x}$，即可通过 PDF 计算其值，然后通过一个阈值来判断该样本是否属于高斯模型。很明显，SGM 适合于仅有两类别问题的划分，而 GMM 由于具有多个模型，划分更为精细，适用于多类别的划分，可以应用于复杂对象建模。

下面以视频前景分割应用场景为例，说明 SGM 与 GMM 在应用上的优劣比较：

- SGM 需要进行初始化，如在进行视频背景分割时，这意味着如果人体在前几帧就出现在摄像头前，人体将会被初始化为背景，而使模型无法使用；
- SGM 只能进行微小性渐变，而不可突变。如户外亮度随时间的渐变是可以适应的，如果在明亮的室内突然关灯，单高斯模型就会将整个室内全部判断为前景。又如，若在监控范围内开了一辆车，并在摄像头下开始停留。由于与模型无法匹配，车会一直被视为前景。当车过很长时间离去时，由于车停留点的亮度发生了很大的变化，因此已经无法与先前的背景模型相匹配；
- SGM 无法适应背景有多个状态，如窗帘，风吹的树叶。单高斯模型无法表示这种情况，而使得前背景检测混乱，而 GMM 能够很好地描述不同状态；
- 相对于单高斯模型的自适应变化，混合高斯模型的自适应变化要健壮的多。它能解决单高斯模型很多不能解决的问题。如无法解决同一样本点的多种状态，无法进行模型状态转化等。

1.1 单高斯模型

多维高斯（正态）分布概率密度函数 PDF 定义如下：

$$N(\boldsymbol{x};\boldsymbol{\mu},\Sigma)=\frac{1}{\sqrt{(2p)|\Sigma|}}\exp[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\sum^{-1}(\boldsymbol{x}-\boldsymbol{\mu})] \qquad (1)$$

注意与一维高斯分布不同，其中 $\boldsymbol{x}$ 是维数为 d 的样本向量（列向量），$\boldsymbol{\mu}$ 是模型期望，$\boldsymbol{\Sigma}$ 是模型方差。

对于单高斯模型，由于可以明确训练样本是否属于该高斯模型（如训练人脸肤色模型时，将人脸图像肤色部分分割出来，形成训练集），故 $\boldsymbol{\mu}$ 通常由训练样本均值代替，$\boldsymbol{\Sigma}$ 由样本方差代替。为了将高斯分布用于模式分类，假设训练样本属于类别 C，那么，式(1)可以改为如下形式：

$$N(\boldsymbol{x}/C)=\frac{1}{\sqrt{(2p)|\Sigma|}}\exp[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\sum^{-1}(\boldsymbol{x}-\boldsymbol{\mu})] \qquad (2)$$

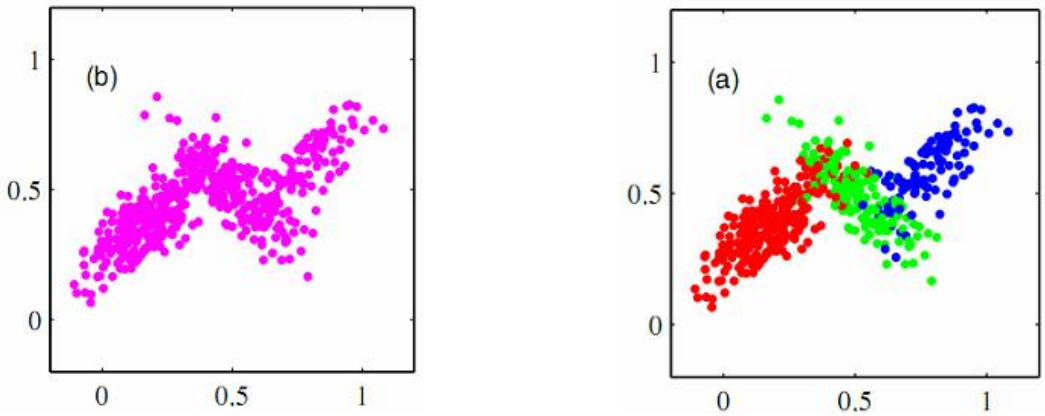
式(2)表明样本属于类别 C 的概率大小。从而将任意测试样本 $\boldsymbol{x}_i$ 输入式(2)，均可以得到一个标量 $N(\boldsymbol{x}_i;\boldsymbol{\mu},\Sigma)$，然后根据阈值 t 来确定该样本是否属于该类别。

- **阈值 t 的确定**：可以为经验值，也可以通过实验确定。另外也有一些策略可以参考：如文献[1]中采用搜索策略：首先令 t =0.7，以 0.05 为步长一直减到 0.1 左右，选择使样本变化最小的那个阈值做为最终 t 值，也就是意味着所选 t 值所构造的分类模型最稳定。
- **几何意义理解**：根据单高斯分布 PDF 的含义我们可以知道，符合 SGM 分布的二维点在平面上应该近似椭圆形；相应地，三维点在空间中则近似于椭球状

1.2 高斯混合模型

高斯混合模型是单一高斯机率密度函数的延伸，由于 GMM 能够平滑地近似任意形状的密度分布，因此近年来常被用在语音、图像识别等方面，得到不错的效果。

例：有一批观察数据 $X=\{\boldsymbol{x}_1,...,\boldsymbol{x}_n\}$，数据个数为 n，在 d 维空间中的分布不是椭球状（如图 1(a)），那么就不适合以一个单一的高斯密度函数来描述这些数据点的机率密度函数。此时我们采用一个变通方案，假设每个点均由一个单高斯分布生成（如图 1(b)，具体参数 $\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j$ 未知），而这一批数据共由 M （明确）个单高斯模型生成，具体某个数据 $\boldsymbol{x}_i$ 属于哪个单高斯模型未知，且每个单高斯模型在混合模型中占的比例 a_j 未知，将所有来自不同分布的数据点混在一起，该分布称为高斯混合分布。



(a) (b)

图 1 高斯混合模型图示，(a)表示所有样本数据；(b)表示已经明确了样本的分类

从数学上讲，我们认为这些数据的概率分布密度函数可以通过加权函数表示：

$$p(\boldsymbol{x}_i)=\sum_{j=1}^{M}a_j N_j(\boldsymbol{x}_i;\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j) \tag{3}$$

上式即称为 GMM，$\sum_{j=1}^{M}a_j=1$，其中

$$N_j\left(\boldsymbol{x};\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j\right)=\frac{1}{\sqrt{(2p)^m\left|\boldsymbol{\Sigma}_j\right|}}\exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_j)^T\boldsymbol{\Sigma}_j^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j)\right] \tag{4}$$

表示第 j 个 SGM 的 PDF。

令 $q_j=(a_j,\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j)$，GMM 共有 M 个 SGM 模型，现在，我们就需要通过样本集 X 来估计 GMM 的所有参数：$\Theta=(q_1,\cdots,q_M)^T$。样本 X 的概率公式为：

$$\begin{aligned} l(X|\Theta)&=\log\prod_{i=1}^{N}\sum_{j=1}^{M}a_j N_j\left(\boldsymbol{x};\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j\right)\\ &=\sum_{i=1}^{N}\log\sum_{j=1}^{M}a_j N_j\left(\boldsymbol{x};\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j\right)\end{aligned} \tag{5}$$

2 采用 EM 估计 GMM 的参数

通常采用 EM 算法（期望值最大，Expectation Maximum）根据(5)式对 GMM 参数进行估计。具体推导过程在很多文献可以参阅，这里不详述，我们主要介绍采用 EM 估计 GMM 参数的方法。

描述估计高斯混合分布的 EM 算法（参考“!!EM 算法.ppt”）：

- **初始值**：

方案 1：协方差矩阵 $\boldsymbol{\Sigma}_{j0}$ 设为单位矩阵；每个模型比例的先验概率： $a_{j0}=1/M$ ；均值 $\boldsymbol{\mu}_{j0}$ 设为随机数；

方案 2：由 k 均值（k-mean）聚类算法对样本进行聚类，利用各类的均值作为 $\boldsymbol{\mu}_{j0}$ ，并计算 $\boldsymbol{\Sigma}_{j0}$ ， a_{j0} 取各类样本占样本总数的比例。

- **算法流程**：

（1）估计步骤（E-step）：

令 a_j 的后验概率为：

$$b_j = E(a_j|\boldsymbol{x}_i;\Theta) = \frac{a_j N_j(\boldsymbol{x}_i;\Theta)}{\sum_{l}^{M} a_l N_l(\boldsymbol{x}_i;\Theta)}, \quad 1\le i\le n, \ 1\le j\le M \tag{6}$$

注意：写代码实现公式(6)时，对于每个 SGM 分别用公式(4)计算每个样本点 $\boldsymbol{x}_i$ 在该模型下的概率密度值 $N_j(\boldsymbol{x}_i;\Theta)$，对于所有样本，得到一个 $n*1$ 的向量，计算 M 次，得到 $n*M$ 的矩阵，每一列为所有点在该模型下的概率密度值（PDF）；实现 $\sum_{l}^{M} a_l N_l(\boldsymbol{x}_i;\Theta)$ 时，需要针对每个点计算在各个 SGM 的概率值总和。公式(6)可以通过如下 matlab 代码描述：

```
function prob = GaussPDF(Data, Mu, Sigma)
[dim,N] = size(Data);
Data = Data' - repmat(Mu',N,1);
prob = sum((Data*inv(Sigma)).*Data, 2);
prob = exp(-0.5*prob) / sqrt((2*pi)^dim * (abs(det(Sigma))+realmin));
```

（2）最大化步骤（M-step）：

- 更新权值： $a_j' = \frac{\sum_{i=1}^{N} b_{ij}}{N}$ (7)
- 更新均值： $\boldsymbol{\mu}_j' = \frac{\sum_{i=1}^{n} b_{ij}\boldsymbol{x}_i}{\sum_{i=1}^{n} b_{ij}}$ (8)
- 更新方差矩阵： $\boldsymbol{\Sigma}_j' = \frac{\sum_{i=1}^{n} b_{ij}(\boldsymbol{x}_i - m_j')(\boldsymbol{x}_i - m_j')^T}{\sum_{i=1}^{n} b_{ij}}$ (9)

注意，有些文献把方差矩阵当作对角阵处理，即认为 $\boldsymbol{\Sigma}_j = diag(s^2)$，故采用如下公式估计 s^2：

$$s^2 = \frac{\sum_{i=1}^{n} b_{ij}(\boldsymbol{x}_i - m_j')^T(\boldsymbol{x}_i - m_j')}{\sum_{i=1}^{n} b_{ij}} \quad (10)$$

经本人实验验证，公式(10)会给最终结果带来误差，故在运算速度能保证的情况下建议采用公式(9)估计方差矩阵。

（3）收敛条件

方案 1：不断地迭代 E 和 M 步骤，重复更新上面的三个值，直到$\left|l(X|\Theta) - l(X|\Theta)'\right| < e$，通常$e = 10^{-5}$，$l(X|\Theta)$通过公式(5)计算，$l(X|\Theta)'$表示更新参数后计算的值；

方案 2：不断地迭代 E 和 M 步骤，重复更新上面的三个值，直到参数的变化不显著，即$\left|\Theta - \Theta'\right| < e$，$\Theta'$为更新后的参数，通常$e = 10^{-5}$。

注：实验表明方案 1 和方案 2 效果接近，但方案 2 明显运算量要小，故推荐使用。

3 源码

3.1 单高斯模型

下面代码实现了 SGM，并实现了人脸肤色检测。其中图像处理、矩阵运算采用了 openCV 库函数

```
/*********************************************************************************
    Single Gaussian Model for skin color extraction
    Param:
        img -- input image to extract the face region
        skinImg -- result
*********************************************************************************/
void CSkinColor::RunSGM(IplImage *img, IplImage **skinImg)
{
    if (img == NULL) return -1;
    ////////////////////////////////////////////////////////////////////////
    // 以下参数一组(117.4361,156.5599)来自源码 light2，与文章《王航宇:基于 YCbCr 高斯肤色模型的
    // 人脸检测技术研究》相同，另一组来自源码“肤色检测正式版”（103.0056, 140.1309）
    double M[]={103.0056, 140.1309}/*{117.4361,156.5599}*/;//M 为肤色在 YCbCr 颜色空间的样本均值(Cb, Cr)，经验值
    double C[2][2]={{160.1301,12.1430},//C 为肤色相似度模型的协方差矩阵，同上为经验值
        {12.1430,299.4574}};// 注：因为运算仅需要该矩阵的逆矩阵值，故该值没有使用，仅作参考
    double invC[2][2]={0.0077 ,-0.0041,-0.0041 ,0.0047
    };//Ct 为 C 的逆矩阵值，由 matlab 计算而得
    ////////////////////////////////////////////////////////////////////////
    IplImage* pImg = img;
    double CrMean=0,CbMean=0,YMean=0;
    // 1 颜色转换：BGR->YCrCb
    IplImage*imgYCrCb=cvCreateImage(cvGetSize(pImg),IPL_DEPTH_8U,3);// YCrCb 图像
```

```
cvCvtColor(pImg, imgYCrCb, CV_BGR2YCrCb);//  第 0,1,2 层分别为 Y,Cr,Cb

IplImage *imgY = cvCreateImage(cvGetSize(pImg),IPL_DEPTH_8U,1);// YCrCb 图像
IplImage *imgCr = cvCreateImage(cvGetSize(pImg),IPL_DEPTH_8U,1);// YCrCb 图像
IplImage *imgCb = cvCreateImage(cvGetSize(pImg),IPL_DEPTH_8U,1);// YCrCb 图像
IplImage *imgY32 = cvCreateImage(cvGetSize(pImg),IPL_DEPTH_32F,1);// YCrCb 图像
IplImage *imgCr32 = cvCreateImage(cvGetSize(pImg),IPL_DEPTH_32F,1);// YCrCb 图像
IplImage *imgCb32 = cvCreateImage(cvGetSize(pImg),IPL_DEPTH_32F,1);// YCrCb 图像
cvSplit(imgYCrCb, imgY, imgCr, imgCb, NULL);
cvConvert(imgY, imgY32);
cvConvert(imgCr, imgCr32);
cvConvert(imgCb, imgCb32);
//////////////////////////////////////////////////////////////////////////
// 2  根据 Sigle Gaussian Model 计算颜色模型
IplImage *PCbCr=cvCreateImage(cvGetSize(pImg), IPL_DEPTH_32F, 1);//YCrCb 颜色模型
IplImage *tempA=cvCreateImage(cvGetSize(pImg), IPL_DEPTH_32F, 1);//YCrCb 颜色模型
IplImage *tempB=cvCreateImage(cvGetSize(pImg), IPL_DEPTH_32F, 1);//YCrCb 颜色模型
cvSubS(imgCb32, cvScalar(M[0]), imgCb32);// x-m
cvSubS(imgCr32, cvScalar(M[1]), imgCr32);// x-m
cvAddWeighted(imgCb32, invC[0][0], imgCr32, invC[1][0], 0, tempA);
cvAddWeighted(imgCb32, invC[0][1], imgCr32, invC[1][1], 0, tempB);
cvMul(imgCb32, tempA, tempA, -0.5);
cvMul(imgCr32, tempB, tempB, -0.5);
cvAdd(tempA, tempB, PCbCr);
cvExp(PCbCr, PCbCr);
double max_val=0,min_val=0;
cvMinMaxLoc(PCbCr,&min_val,&max_val);
IplImage *proImg=cvCreateImage(cvGetSize(pImg),IPL_DEPTH_8U, 1);//YCrCb 颜色模型
double a=255/(max_val);
cvConvertScaleAbs(PCbCr,proImg,a,0);
m_proimg = cvCloneImage(proImg);

if ((*skinImg)!=NULL) cvReleaseImage(skinImg);
*skinImg = cvCreateImage(cvGetSize(pImg),IPL_DEPTH_8U, 1);//肤色结果
//  释放内存
cvReleaseImage(&proImg);
cvReleaseImage(&imgYCrCb);
cvReleaseImage(&imgY);
cvReleaseImage(&imgCr);
cvReleaseImage(&imgCb);
cvReleaseImage(&imgY32);
cvReleaseImage(&imgCr32);
cvReleaseImage(&imgCb32);
cvReleaseImage(&PCbCr);
```

```
    cvReleaseImage(&tempA);
    cvReleaseImage(&tempB);
}
```

3.1 高斯混合模型

（1）以下 matlab 代码实现了高斯混合模型：

```
function [Alpha, Mu, Sigma] = GMM_EM(Data, Alpha0, Mu0, Sigma0)
%% EM 迭代停止条件
loglik_threshold = 1e-10;
%% 初始化参数
[dim, N] = size(Data);
M = size(Mu0,2);
loglik_old = -realmax;
nbStep = 0;

Mu = Mu0;
Sigma = Sigma0;
Alpha = Alpha0;
Epsilon = 0.0001;
while (nbStep < 1200)
  nbStep = nbStep+1;
  %% E-步骤 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  for i=1:M
    % PDF of each point
    Pxi(:,i) = GaussPDF(Data, Mu(:,i), Sigma(:,:,i));
  end

  % 计算后验概率 beta(i|x)
  Pix_tmp = repmat(Alpha,[N 1]).*Pxi;
  Pix = Pix_tmp ./ (repmat(sum(Pix_tmp,2),[1 M])+realmin);
  Beta = sum(Pix);
  %% M-步骤 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  for i=1:M
    % 更新权值
    Alpha(i) = Beta(i) / N;
    % 更新均值
    Mu(:,i) = Data*Pix(:,i) / Beta(i);
    % 更新方差
    Data_tmp1 = Data - repmat(Mu(:,i),1,N);
    Sigma(:,:,i) = (repmat(Pix(:,i)',dim, 1) .* Data_tmp1*Data_tmp1') / Beta(i);
    %% Add a tiny variance to avoid numerical instability
    Sigma(:,:,i) = Sigma(:,:,i) + 1E-5.*diag(ones(dim,1));
  end
```

```
%    %% Stopping criterion 1 %%%%%%%%%%%%%%%%%%%%
%    for i=1:M
       %Compute the new probability p(x|i)
%        Pxi(:,i) = GaussPDF(Data, Mu(:,i), Sigma(i));
%    end
   %Compute the log likelihood
%    F = Pxi*Alpha';
%    F(find(F<realmin)) = realmin;
%    loglik = mean(log(F));
   %Stop the process depending on the increase of the log likelihood
%    if abs((loglik/loglik_old)-1) < loglik_threshold
%        break;
%    end
%    loglik_old = loglik;

   %% Stopping criterion 2 %%%%%%%%%%%%%%%%%%%%
   v = [sum(abs(Mu - Mu0)), abs(Alpha - Alpha0)];
   s = abs(Sigma-Sigma0);
   v2 = 0;
   for i=1:M
      v2 = v2 + det(s(:,:,i));
   end

   if ((sum(v) + v2) < Epsilon)
      break;
   end
   Mu0 = Mu;
   Sigma0 = Sigma;
   Alpha0 = Alpha;
end
nbStep
```

（2）以下代码根据高斯分布函数计算每组数据的概率密度，被 GMM_EM 函数所调用

```
function prob = GaussPDF(Data, Mu, Sigma)
%
% 根据高斯分布函数计算每组数据的概率密度 Probability Density Function (PDF)
% 输入 ----------------------------------------------------------------
%     o Data:    D x N ， N 个 D 维数据
%     o Mu:       D x 1 ， M 个 Gauss 模型的中心初始值
%     o Sigma: M x M ， 每个 Gauss 模型的方差（假设每个方差矩阵都是对角阵，
%                              即一个数和单位矩阵的乘积）
% Outputs ----------------------------------------------------------------
%     o prob:    1 x N array representing the probabilities for the
```

```
%                   N datapoints.
[dim,N] = size(Data);
Data = Data' - repmat(Mu',N,1);
prob = sum((Data*inv(Sigma)).*Data, 2);
prob = exp(-0.5*prob) / sqrt((2*pi)^dim * (abs(det(Sigma))+realmin));
```

（3）以下是演示代码 demo1.m

```
% 高斯混合模型参数估计示例 （基于 EM 算法）
% 2010 年 11 月 9 日
[data, mu, var, weight] = CreateSample(M, dim, N);  // 生成测试数据
[Alpha, Mu, Sigma] = GMM_EM(Data, Priors, Mu, Sigma)
```

（4）以下是测试数据生成函数，为 demo1.m 所调用：

```
function [data, mu, var, weight] = CreateSample(M, dim, N)
% 生成实验样本集，由 M 组正态分布的数据构成
% % GMM 模型的原理就是仅根据数据估计参数：每组正态分布的均值、方差，
% 以及每个正态分布函数在 GMM 的权重 alpha。
% 在本函数中，这些参数均为随机生成，
%
% 输入
%    M     ：高斯函数个数
%    dim   ：数据维数
%    N     ：数据总个数
% 返回值
%    data : dim-by-N, 每列为一个数据
%    miu   : dim-by-M, 每组样本的均值，由本函数随机生成
%    var   : 1-by-M, 均方差，由本函数随机生成
%    weight: 1-by-M, 每组的权值，由本函数随机生成
% ------------------------------------------------
%
% 随机生成不同组的方差、均值及权值
weight = rand(1,M);
weight = weight / norm(weight, 1);        % 归一化，保证总合为 1
var = double(mod(int16(rand(1,M)*100),10) + 1);  % 均方差，取 1~10 之间，采用对角矩阵
mu = double(round(randn(dim,M)*100));            % 均值，可以有负数

for(i = 1: M)
   if (i ~= M)
      n(i) = floor(N*weight(i));
   else
      n(i) = N - sum(n);
   end
end
```

```
% 以标准高斯分布生成样本值，并平移到各组相应均值和方差
start = 0;
for (i=1:M)
   X = randn(dim, n(i));
   X = X.* var(i) + repmat(mu(:,i),1,n(i));
   data(:,(start+1):start+n(i)) = X;
   start = start + n(i);
end
save('d:\data.mat', 'data');
```