# 漫谈 Clustering (3): Gaussian Mixture Model

by pluskid, on 2009-02-02, in Machine Learning　194 comments

*本文是"漫谈 Clustering 系列"中的第 4 篇，参见本系列的其他文章。*

上一次我们谈到了用 k-means 进行聚类的方法，这次我们来说一下另一个很流行的算法：Gaussian Mixture Model (GMM)。事实上，GMM 和 k-means 很像，不过 GMM 是学习出一些概率密度函数来（所以 GMM 除了用在 clustering 上之外，还经常被用于 density estimation ），简单地说，k-means 的结果是每个数据点被 assign 到其中某一个 cluster 了，而 GMM 则给出这些数据点*被 assign 到每个 cluster 的概率*，又称作 *soft assignment* 。

得出一个概率有很多好处，因为它的信息量比简单的一个结果要多，比如，我可以把这个概率转换为一个 score ，表示算法对自己得出的这个结果的把握。也许我可以对同一个任务，用多个方法得到结果，最后选取"把握"最大的那个结果；另一个很常见的方法是在诸如疾病诊断之类的场所，机器对于那些很容易分辨的情况（患病或者不患病的概率很高）可以自动区分，而对于那种很难分辨的情况，比如，49% 的概率患病，51% 的概率正常，如果仅仅简单地使用 50% 的阈值将患者诊断为"正常"的话，风险是非常大的，因此，在机器对自己的结果把握很小的情况下，会"拒绝发表评论"，而把这个任务留给有经验的医生去解决。

废话说了一堆，不过，在回到 GMM 之前，我们再稍微扯几句。我们知道，不管是机器还是人，学习的过程都可以看作是一种"归纳"的过程，在归纳的时候你需要有一些假设的前提条件，例如，当你被告知水里游的那个家伙是鱼之后，你使用"在同样的地方生活的是同一种东西"这类似的假设，归纳出"在水里游的都是鱼"这样一个结论。当然这个过程是完全"本能"的，如果不仔细去想，你也不会了解自己是怎样"认识鱼"的。另一个值得注意的地方是这样的假设并不总是完全正确的，甚至可以说总是会有这样那样的缺陷的，因此你有可能会把虾、龟、甚至是潜水员当做鱼。也许你觉得可以通过修改前提假设来解决这个问题，例如，基于"生活在同样的地方并且穿着同样衣服的是同一种东西"这个假设，你得出结论：在水里有并且身上长有鳞片的是鱼。可是这样还是有问题，因为有些没有长鳞片的鱼现在又被你排除在外了。

在这个问题上，机器学习面临着和人一样的问题，在机器学习中，一个学习算法也会有一个前提假设，这里被称作"归纳偏执 (bias)"（bias 这个英文词在机器学习和统计里还有其他许多的意思）。例如线性回归，目的是要找一个函数尽可能好地拟合给定的数据点，它的归纳偏执就是"满足要求的函数必须是线性函数"。一个没有归纳偏执的学习算法从某种意义上来说毫无用处，就像一个完全没有归纳能力的人一样，在第一次看到鱼的时候有人告诉他那是鱼，下次看到另一条鱼了，他并不知道那也是鱼，因为两条鱼总有一些地方不一样的，或者就算是同一条鱼，在河里不同的地方看到，或者只是看到的时间不一样，也会被他认为是不同的，因为他无法归纳，无法提取主要矛盾、忽略次要因素，只好要求所有的条件都完全一样——然而哲学家已经告诉过我们了：世界上不会有任何样东西是完全一样的，所以这个人即使是有无比强悍的记忆力，也绝学不到任何一点知识。

这个问题在机器学习中称作"过拟合 (Overfitting)"，例如前面的回归的问题，如果去掉"线性函数"这个归纳偏执，因为对于 N 个点，我们总是可以构造一个 N-1 次多项式函数，让它完美地穿过所有的这 N 个点，或者如果我用任何大于 N-1 次的多项式函数的话，我甚至可以构造出无穷多个满足条件的函数出来。如果假定特定领域里的问题所给定的数据个数总是有个上限的话，我可以取一个足够大的 N ，从而得到一个（或者无穷多个）"超级函数"，能够 fit 这个领域内所有的问题。然而这个（或者这无穷多个）"超级函数"有用吗？只要我们注意到*学习*的目的（通常）不是解释现有的事物，而是从中*归纳*出*知识*，并能应用到*新的*事物上，结果就显而易见了。

没有归纳偏执或者归纳偏执太宽泛会导致 Overfitting ，然而另一个极端——限制过大的归纳偏执也是有问题的：如果数据本身并不是线性的，强行用线性函数去做回归通常并不能得到好结果。难点正在于在这之间寻找一个平衡点。不过人在这里相对于（现在的）机器来说有一个很大的优势：人通常不会孤立地用某一个独立的系统和模型去处理问题，一个人每天都会从各个来源获取大量的信息，并且通过各种手段进行整合处理，归纳所得的所有知识最终得以*统*一地存储起来，并能*有机*地组合起来去解决特定的问题。这里的"有机"这个词很有意思，搞理论的人总能提出各种各样的模型，并且这些模型都有严格的理论基础保证能达到期望的目的，然而绝大多数模型都会有那么一些"参数"（例如 K-means 中的 k ），通常没有理论来说明参数取哪个值更好，而模型实际的效果却通常和参数是否取到最优值有很大的关系，我觉得，在这里"有机"不妨看作是所有模型的参数已经自动地取到了最优值。另外，虽然进展不大，但是人们也一直都期望在计算机领域也建立起一个统一的知识系统（例如语意网就是这样一个尝试）。

废话终于说完了，回到 GMM 。按照我们前面的讨论，作为一个流行的算法，GMM 肯定有它自己的一个相当体面的归纳偏执了。其实它的假设非常简单，顾名思义，Gaussian Mixture Model ，就是假设数据服从 Mixture Gaussian Distribution ，换句话说，数据可以看作是从数个 Gaussian Distribution 中生成出来的。实际上，我们在 K-means 和 K-medoids 两篇文章中用到的那个例子就是由三个 Gaussian 分布从随机选取出来的。实际上，从中心极限定理可以看出，Gaussian 分布（也叫做正态 (Normal) 分布）这个假设其实是比较合理的，除此之外，Gaussian 分布在计算上也有一些很好的性质，所以，虽然我们可以用不同的分布来随意地构造 XX Mixture Model ，但是还是 GMM 最为流行。另外，Mixture Model 本身其实也是可以变得任意复杂的，通过增加 Model 的个数，我们可以任意地逼近任何连续的概率密分布。

每个 GMM 由 $K$ 个 Gaussian 分布组成，每个 Gaussian 称为一个"Component"，这些 Component 线性加成在一起就组成了 GMM 的概率密度函数：

$$\begin{aligned} p(x) &= \sum_{k=1}^{K} p(k)p(x|k) \\ &= \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \end{aligned}$$

根据上面的式子，如果我们要从 GMM 的分布中随机地取一个点的话，实际上可以分为两步：首先随机地在这 $K$ 个 Component 之中选一个，每个 Component 被选中的概率实际上就是它的系数 $\pi_k$ ，选中了 Component 之后，再单独地考虑从这个 Component 的分布中选取一个点就可以了——这里已经回到了普通的 Gaussian 分布，转化为了已知的问题。

那么如何用 GMM 来做 clustering 呢？其实很简单，现在我们有了数据，假定它们是由 GMM 生成出来的，那么我们只要根据数据推出 GMM 的概率分布来就可以了，然后 GMM 的 $K$ 个 Component 实际上就对应了 $K$ 个 cluster 了。根据数据来推算概率密度通常被称作 density estimation ，特别地，当我们在已知（或假定）了概率密度函数的形式，而要估计其中的参数的过程被称作"参数估计"。

现在假设我们有 $N$ 个数据点，并假设它们服从某个分布（记作 $p(x)$ ），现在要确定里面的一些参数的值，例如，在 GMM 中，我们就需要确定 $\pi_k$、$\mu_k$ 和 $\Sigma_k$ 这些参数。 我们的想法是，找到这样一组参数，它所确定的概率分布生成这些给定的数据点的概率最大，而这个概率实际上就等于 $\prod_{i=1}^{N} p(x_i)$ ，我们把这个乘积称作似然函数 (Likelihood Function)。通常单个点的概率都很小，许多很小的数字相乘起来在计算机里很容易造成浮点数下溢，因此我们通常会对其取对数，把乘积变成加和 $\sum_{i=1}^{N} \log p(x_i)$ ，得到 log-likelihood function 。接下来我们只要将这个函数最大化（通常的做法是求导并令导数等于零，然后解方程），亦即找到这样一组参数值，它让似然函数取得最大值，我们就认为这是最合适的参数，这样就完成了参数估计的过程。

下面让我们来看一看 GMM 的 log-likelihood function ：

$$\sum_{i=1}^{N}\log\left\{\sum_{k=1}^{K}\pi_k\mathcal{N}(x_i|\mu_k,\Sigma_k)\right\}$$

由于在对数函数里面又有加和，我们没法直接用求导解方程的办法直接求得最大值。为了解决这个问题，我们采取之前从 GMM 中随机选点的办法：分成两步，实际上也就类似于 K-means 的两步。

1. 估计数据由每个 Component 生成的概率（并不是每个 Component 被选中的概率）：对于每个数据 $x_i$ 来说，它由第 $k$ 个 Component 生成的概率为

$$\gamma(i,k)=\frac{\pi_k\mathcal{N}(x_i|\mu_k,\Sigma_k)}{\sum_{j=1}^{K}\pi_j\mathcal{N}(x_i|\mu_j,\Sigma_j)}$$

   由于式子里的 $\mu_k$ 和 $\Sigma_k$ 也是需要我们估计的值，我们采用迭代法，在计算 $\gamma(i,k)$ 的时候我们假定 $\mu_k$ 和 $\Sigma_k$ 均已知，我们将取上一次迭代所得的值（或者初始值）。

2. 估计每个 Component 的参数：现在我们假设上一步中得到的 $\gamma(i,k)$ 就是正确的“数据 $x_i$ 由 Component $k$ 生成的概率”，亦可以当做该 Component 在生成这个数据上所做的贡献，或者说，我们可以看作 $x_i$ 这个值其中有 $\gamma(i,k)x_i$ 这部分是由 Component $k$ 所生成的。集中考虑所有的数据点，现在实际上可以看作 Component 生成了 $\gamma(1,k)x_1,\ldots,\gamma(N,k)x_N$ 这些点。由于每个 Component 都是一个标准的 Gaussian 分布，可以很容易分布求出最大似然所对应的参数值：

$$\mu_k=\frac{1}{N_k}\sum_{i=1}^{N}\gamma(i,k)x_i$$

$$\Sigma_k=\frac{1}{N_k}\sum_{i=1}^{N}\gamma(i,k)(x_i-\mu_k)(x_i-\mu_k)^T$$

   其中 $N_k=\sum_{i=1}^{N}\gamma(i,k)$，并且 $\pi_k$ 也顺理成章地可以估计为 $N_k/N$ 。

3. 重复迭代前面两步，直到似然函数的值收敛为止。

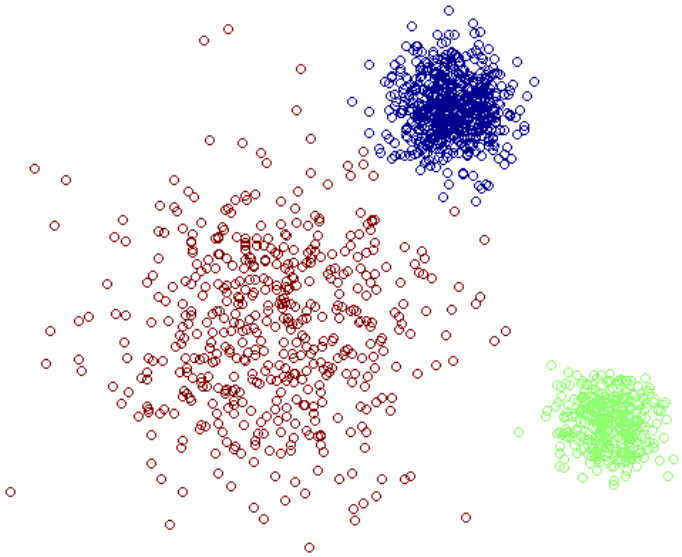
当然，上面给出的只是比较“直观”的解释，想看严格的推到过程的话，可以参考 *Pattern Recognition and Machine Learning* 这本书的第九章。有了实际的步骤，再实现起来就很简单了。Matlab 代码如下：

（Update 2012.07.03：如果你直接把下面的代码拿去运行了，碰到 covariance 矩阵 singular 的情况，可以参见这篇文章。）

```
function varargout = gmm(X, K_or_centroids)
% ========================================================
% Expectation-Maximization iteration implementation of
% Gaussian Mixture Model.
%
% PX = GMM(X, K_OR_CENTROIDS)
% [PX MODEL] = GMM(X, K_OR_CENTROIDS)
%
%  - X: N-by-D data matrix.
%  - K_OR_CENTROIDS: either K indicating the number of
%       components or a K-by-D matrix indicating the
%       choosing of the initial K centroids.
%
%  - PX: N-by-K matrix indicating the probability of each
%       component generating each point.
%  - MODEL: a structure containing the parameters for a GMM:
%       MODEL.Miu: a K-by-D matrix.
%       MODEL.Sigma: a D-by-D-by-K matrix.
%       MODEL.Pi: a 1-by-K vector.
% ========================================================

    threshold = 1e-15;
    [N, D] = size(X);

    if isscalar(K_or_centroids)
        K = K_or_centroids;
        % randomly pick centroids
        rndp = randperm(N);
        centroids = X(rndp(1:K), :);
    else
        K = size(K_or_centroids, 1);
        centroids = K_or_centroids;
    end

    % initial values
    [pMiu pPi pSigma] = init_params();

    Lprev = -inf;
    while true
        Px = calc_prob();

        % new value for pGamma
        pGamma = Px .* repmat(pPi, N, 1);
        pGamma = pGamma ./ repmat(sum(pGamma, 2), 1, K);

        % new value for parameters of each Component
```

```
        Nk = sum(pGamma, 1);
        pMiu = diag(1./Nk) * pGamma' * X;
        pPi = Nk/N;
        for kk = 1:K
            Xshift = X-repmat(pMiu(kk, :), N, 1);
            pSigma(:, :, kk) = (Xshift' * ...
                (diag(pGamma(:, kk)) * Xshift)) / Nk(kk);
        end

        % check for convergence
        L = sum(log(Px*pPi'));
        if L-Lprev < threshold
            break;
        end
        Lprev = L;
    end

    if nargout == 1
        varargout = {Px};
    else
        model = [];
        model.Miu = pMiu;
        model.Sigma = pSigma;
        model.Pi = pPi;
        varargout = {Px, model};
    end

    function [pMiu pPi pSigma] = init_params()
        pMiu = centroids;
        pPi = zeros(1, K);
        pSigma = zeros(D, D, K);

        % hard assign x to each centroids
        distmat = repmat(sum(X.*X, 2), 1, K) + ...
            repmat(sum(pMiu.*pMiu, 2)', N, 1) - ...
            2*X*pMiu';
        [dummy labels] = min(distmat, [], 2);

        for k=1:K
            Xk = X(labels == k, :);
            pPi(k) = size(Xk, 1)/N;
            pSigma(:, :, k) = cov(Xk);
        end
    end

    function Px = calc_prob()
        Px = zeros(N, K);
        for k = 1:K
            Xshift = X-repmat(pMiu(k, :), N, 1);
            inv_pSigma = inv(pSigma(:, :, k));
            tmp = sum((Xshift*inv_pSigma) .* Xshift, 2);
            coef = (2*pi)^(-D/2) * sqrt(det(inv_pSigma));
            Px(:, k) = coef * exp(-0.5*tmp);
        end
    end
end
```

函数返回的 `Px` 是一个 $N \times K$ 的矩阵，对于每一个 $x_i$，我们只要取该矩阵第 $i$ 行中最大的那个概率值所对应的那个 Component 为 $x_i$ 所属的 cluster 就可以实现一个完整的聚类方法了。对于最开始的那个例子，GMM 给出的结果如下：



相对于之前 K-means 给出的结果，这里的结果更好一些，左下角的比较稀疏的那个 cluster 有一些点跑得比较远了。当然，因为这个问题原本就是完全有 Mixture Gaussian Distribution 生成的数据，GMM （如果能求得全局最优解的话）显然是可以对这个问题做到的最好的建模。

另外，从上面的分析中我们可以看到 GMM 和 K-means 的迭代求解法其实非常相似（都可以追溯到 EM 算法，下一次会详细介绍），因此也有和 K-means 同样的问题——并不能保证总是能取到全局最优，如果运气比较差，取到不好的初始值，就有可能得到很差的结果。对于 K-means 的情况，我们通常是重复一定次数然后取最好的结果，不过 GMM 每一次迭代的计算量比 K-means 要大许多，一个更流行的做法是先用 K-means （已经重复并取最优值了）得到一个粗略的结果，然后将其作为初值（只要将 K-means 所得的 centroids 传入 `gmm` 函数即可），再用 GMM 进行细致迭代。

如我们最开始所讨论的，GMM 所得的结果（`Px`）不仅仅是数据点的 label ，而包含了数据点标记为每个 label 的概率，很多时候这实际上是非常有用的信息。最后，需要指出的是，GMM 本身只是一个模型，我们这里给出的迭代的办法并不是唯一的求解方法。感兴趣的同学可以自行查找相关资料。

Tags: Clustering, Unsupervised Learning

### 194 comments to 漫谈 Clustering (3): Gaussian Mixture Model

« Older Comments 1 2 3

weixue
November 10th, 2013 at 5:10 pm · Reply

你好，我在测试数据时候，遇到问题是矩阵本身不是奇异的，但是对角矩阵的对角元比较小例如0.003，d较大（超过100）的时候matlab处理精度不够运行出来det（covMatrix）=0，我加了缩放因子之后得到的likelyhood就特别大，是处理方式上有问题吗？谢谢指教

菜鸟一枚
November 27th, 2013 at 9:02 am · Reply

楼主，看了你的帖子我明白了GMM 的实现过程受益匪浅，请问楼主如果我想检测图像中的异常行为，首先我提取了正常行为的特征信息，然后用GMM为正常行为的特征空间建立模型。但是，检测时我有遇到问题，按照我的想法是将被测特征向量输入到训练好的GMM中，希望能得到一个概率，如果这个概率小于某个阈值它就不是正常行为，可是，我这里属于被测特征后，得到值非常大，请问楼主我该怎么应用GMM才可达到我预期的目的的（就是希望GMM能给出关于被测点的一个概率，然后，通过这个概率进行判断），当然我的数据属于离散的，因为针对的是图像，希望楼主给予答复十分感谢。

pluskid
November 30th, 2013 at 7:53 am · Reply

你好，GMM 是连续型的概率分布，所以你不会得到一个 0 到 1 之间的概率值。我觉得你需要同时搜集正常数据和异常数据然后根据这些数据决定出一个合理的阈值出来，或者你也可以正常和异常各训练一个模型然后比较概率（密度）大小。

luo mingqi
December 12th, 2013 at 11:13 am · Reply

你好，我也是用这个做图像的，能不能交流一下的？

luo mingqi
December 12th, 2013 at 11:12 am · Reply

这个程序我怎么的实现不了的，在的matlab上不能运行的？

```
im=imread('img1.jpg');
im1=rgb2gray(im);
imshow(im1)
varargout = gmm(im1, 2);
```

不能调用的，谢谢了，我是菜鸟的，不要见笑

wenxingche
March 19th, 2014 at 10:31 pm · Reply

博主，你好。你写的这个关于EM算法的博文很好很强大。我用你的MATLAB代码跑了下数据。发现，最后算出来，每个数据的最后的概率都很低啊，10^(-5)数量级的概率值都算大的了。请问这个正常么？我感觉从道理上讲不通啊，我即便讲训练出来的均值代入训练好的模型，计算出来的概率也很小啊。

pluskid
March 20th, 2014 at 3:24 am · Reply

连续型随机变量得到的是概率密度并不是概率值。

wenxingche
March 20th, 2014 at 8:24 am · Reply

我的意思是，概率密度函数是最后训练出来的模型啊，就是混合高斯模型中，每component的协方差矩阵和期望，以及每个component对应的概率pi.有了这个模型后，将一个数据X代入这个模型，对于每个component都会算出一个对应的概率，然后这个概率乘以相应的pi，最求就和，就是这个数据X，在这个混合模型下，算出的概率啊。我的理解不对么？？代码跑完，最后结果的PX，不就是所有训练数据X在各个component下的概率么？看起来都非常小啊。