Regularized Gaussian Covariance Estimation

http://freemind.pluskid.org/machine-learning/regularized-gaussian-covariance-estimation

posted on Free Mind on July 1, 2012
generated with pandoc on December 3, 2015
category: Machine Learning

tags: Overfitting, Regularization, Maximum Likelihood Estimation

我之前写过一篇介绍 Gaussian Mixture Model (GMM) 的文章，并在文章里贴了一段 GMM 实现的 Matlab 示例代码，然后就不断地有人来问我关于那段代码的问题，问得最多的就是大家经常发现在跑那段代码的时候估计出来的 Covariance Matrix 是 singular 的，所以在第 96 行求逆的时候会挂掉。这是今天要介绍的主要话题，我会讲得罗嗦一点，把关于那篇文章我被问到的一些其他问题也都提到一下，不过，在步入正题之前，不妨先来小小地闲扯一下。

我自己以前就非常在痛恨书里看到伪代码，又不能编译运行，还搞得像模像样的像代码一般，所以我自己在写 blog 的时候就尝试给出直接可运行的代码，但是现在我渐渐理解为什么书的作者喜欢用伪代码而不是可运行的代码了（除非是专门讲某编程语言的书）。原因很简单：示例用的代码和真实项目中的代码往往是差别很大的，直接给出可运行的示例代码，读者就会直接拿去运行（因为包括我在内的大部分人都是很懒的，不是说 “懒” 是程序员的天性么?），而往往示例代码为了结构清晰并突出算法的主要脉络，对很多琐碎情况都没有处理，都使用最直接的操作，没有做任何优化（例如我的那个 GMM 示例代码里直接用 matlab 的 `inv` 函数来求 Covariance 矩阵的逆），等等，因此直接运行这样的代码——特别是在实际项目中使用，是非常不明智的。

可是那又为什么不直接给出实际项目级别的代码呢？第一个原因当然是工作量，我想程序员都知道从一个算法到一个健壮、高效的系统之间有多长的路要走，而且很多时候都需要根据不同的项目环境和需要做不同的修改。所以当一个人实际上是在介绍算法的时候让他去弄一套工业级别的代码出来作为示例，实在是太费力了。当然更重要的原因是：工业级别的代码通常经过大量的优化并充斥着大量错误处理、以及为了处理其他地方的 bug 或者整个系统混乱的 API 等而做的各种 workaround 等等……也许根本就看不懂，基本完全失去了示例的作用。所以，伪代码就成为了唯一的选择。

罗嗦了半天，现在我们回到正题，那篇文章讲的是 GMM 的学习，所以关于 GMM 的问题可以直接参考那篇文章，出问题的地方仅在于 GMM 的每个 component (Gaussian Distribution) 的参数估计上，特别地，在于 Covariance Matrix 的估计上。所以，我们今天主要来探讨 Gaussian 分布的协方差矩阵的估计问题。首先来看一下多维 Gaussian 分布的概率密度函数：

$$\mathcal{N}(x|\mu,\Sigma) = \frac{1}{(2\pi)^{d/2}\det(\Sigma)^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right) \quad (1)$$

参数估计的过程是给定一堆数据点 $x_1, \ldots, x_n$，并假设他们是由某个真实的 Gaussian 分布独立地 (IID) 采样出来的，然后根据这堆点来估计该 Gaussian 分布的参数。对于一个多维 Gaussian 分布来说，需要估计的就是均值 μ 和协方差矩阵 Σ 两 "个" 参数——用引号是因为 μ 和 Σ 都不是标量，假设我们的数据是 d 维的，那么 μ 实际上是 d 个参数，而作为一个 $d \times d$ 的矩阵，Σ 则是由 d^2 个参数组成。根据协方差矩阵的对称性，我们可以进一步把 Σ 所包含的参数个数降低为 $d(d+1)/2$ 个。

值得一提的是，我们从参数估计的角度并不能一下子断定 Σ 就是对称的，需要稍加说明（这实际上是《Pattern Recognition and Machine Learning》书里的习题 2.17）。记 $\Lambda = \Sigma^{-1}$，假设我们的 Λ 不是对称的，那么我们把他写成一个对称矩阵和一个反对称阵之和（任何一个矩阵都可以这样表示）：

$$\Lambda = \frac{\Lambda + \Lambda^T}{2} + \frac{\Lambda - \Lambda^T}{2} \triangleq \Lambda^S + \Lambda^A$$

将起代入式 (1) 的指数部分中，注意到对于反对称部分（书写方便起见我们暂时另 $z = x - \mu$）：

$$\begin{aligned} z^T \Lambda^A z &= \frac{1}{2} z^T (\Lambda - \Lambda^T) z \\ &= \frac{1}{2} \left(\sum_{i,j} z_i \Lambda_{ij} z_j - \sum_{i,j} z_i \Lambda_{ji} z_j \right) \\ &= \frac{1}{2} \left(\sum_{i,j} z_i \Lambda_{ij} z_j - \sum_{i,j} z_j \Lambda_{ji} z_i \right) \\ &= \frac{1}{2} \left(\sum_{i,j} z_i \Lambda_{ij} z_j - \sum_{i,j} z_i \Lambda_{ij} z_j \right) \\ &= 0 \end{aligned}$$

所以指数部分只剩下对称阵 Λ^S 了。也就是说，如果我们找出一个符合条件的非对称阵 Λ，用它的对称部分 Λ^S 代替也是可以的，所以我们就可以 "不失一般性" 地假设 Σ 是对称的（因为对称阵的逆矩阵也是对称的）。注意我们不用考虑式 (1) 指数前面的系数（里的那个 Σ 的行列式），因为前面的系数是起 normalization 作用的，使得全概率积分等于 1，指数部分变动之后只要重新 normalize 一下即可（可以想像一下，如果 Σ 真的是非对称的，那么 normalization 系数那里应该也不会是 $\det(\Sigma)$ 那么简单的形式吧？有兴趣的同学可以自己研究一下）。

好，现在回到参数估计上。对于给定的数据 x_i，将其代入式 (1) 中可以得到其对应的 likelihood $N(x_i|\mu, \Sigma)$，注意这个并**不是** x_i 的概率。和离散概率分布不同的是，这里谈论单个数据点的概率是没有多大意义的，因为单点集的测度为零，它的概率也总是为零。所以，这里的 likelihood 出现大于 1 的值也是很正常的——这也是我经常被问到的一个问题。虽

然不是概率值，但是我想从 likelihood 或者概率密度的角度去理解，应该也是可以得到直观的认识的。

参数估计常用的方法是**最大似然估计**(Maximum Likelihood Estimation, MLE)。就是将所有给定数据点的似然全部乘起来，得到一个关于参数(这里是 μ 和 Σ) 的函数，然后最大化该函数，所对应的参数值就是最大似然估计的值。通常为了避免数值下溢，会使用单调函数 log 将相乘变成相加在计算。

对于最大似然估计，我的理解是寻找最可能生成给定的数据的模型参数。这对于离散型的分布来说是比较好解释的，因为此时一点的似然实际上就是该点的概率，不过对于像 Gaussian 这样的针对连续数据的分布来说，就不是那么自然了。不过我们其实有另一种解释，这是我在 Coursera 的 Probabilistic Graphical Model 课上看到的：

对于一个真实的分布 P^*，假设我们得到了它的一个估计 $\tilde{P}$，那么如何来衡量这个估计的好坏呢？比较两个概率分布的一个常用的办法是用 KL-Divergence，又叫 Relative Entropy（以及其他若干外号），定义为：

$$D(P^* \| \tilde{P}) = \mathbb{E}_{\xi \sim P^*} \left[\log \left(\frac{P^*(\xi)}{\tilde{P}(\xi)} \right) \right]$$

不过由于我们通常是不知道真实的 P^* 的，所以这个值也没法直接算，于是我们把它简单地变形一下：

$$\begin{aligned} D(P^* \| \tilde{P}) &= \mathbb{E}_{\xi \sim P^*} \left[\log P^*(\xi) - \log \tilde{P}(\xi) \right] \\ &= \mathbb{E}_{\xi \sim P^*} \left[\log P^*(\xi) \right] - \mathbb{E}_{\xi \sim P^*} \left[\log \tilde{P}(\xi) \right] \\ &= -\mathbb{H}(P^*) - \mathbb{E}_{\xi \sim P^*} \left[\log \tilde{P}(\xi) \right] \end{aligned}$$

其中蓝色部分是分布 P^* 自己的 Entropy，由于我们比较不同的模型估计好坏的时候这个量是不变的，所以我们可以忽略它，只考虑红色部分。为了使得 KL-Divergence 最小，我们需要将红色部分（注意不包括前面的负号）最大化。不过由于仍然依赖于真实分布 P^*，所以红色部分还是不能直接计算，但是这是一个我们熟悉的形式：某个函数关于真实分布的期望值，而我们手里面又有从真实分布里 IID 采样出来的 n 个数据点，所以可以直接使用标准的近似方法：用数据的经验分布期望来代替原始的期望：

$$\mathbb{E}_{\xi \sim P^*} \left[\log \tilde{P}(\xi) \right] \approx \sum_{i=1}^{n} \frac{1}{n} \log \tilde{P}(x_i)$$

于是我们很自然地就得到了最大似然估计的目标函数。这也从另一个角度解释了最大似然估计的合理性：同时对离散型和连续型的数据都适用。

接下来我们再回到 Gaussian 分布，将 Gaussian 的概率密度函数 (1) 代入最大似然估计的目标函数，可以得到：

$$\begin{aligned}\mathcal{L} &= \sum_{i=1}^{n} \log\left(\frac{1}{(2\pi)^{d/2}\det(\Sigma)^{1/2}} \exp\left(-\frac{1}{2}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)\right)\right) \\ &= -\frac{nd}{2}\log(2\pi) - \frac{n}{2}\log\det(\Sigma) - \frac{1}{2}\sum_{i=1}^{n}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)\end{aligned} \tag{2}$$

由于本文主要讨论协方差矩阵的估计，我们这里就不管均值矩阵 μ 的估计了，方法上是类似的，下面式子中的 μ 将表示已知的或者同样通过最大似然估计计算出来的均值。为了找到使得上式最大化的协方差矩阵，我们将它对于 Σ 求导：

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\Sigma} &= \frac{n}{2}\Sigma^{-1} - \frac{1}{2}\sum_{i=1}^{n}\Sigma^{-1}(x_i-\mu)(x_i-\mu)^T\Sigma^{-1} \\ &= \frac{n}{2}\Sigma^{-1} - \frac{1}{2}\Sigma^{-1}\left(\sum_{i=1}^{n}(x_i-\mu)(x_i-\mu)^T\right)\Sigma^{-1}\end{aligned}$$

令导数等于零，即可得到最优解：

$$\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)(x_i-\mu)^T \tag{3}$$

于是 $\hat{\Sigma}$ 就是协方差矩阵的最大似然估计。到这里问题就出来了，再回到 Gaussian 分布的概率密度函数 (1)，如果把 $\hat{\Sigma}$ 带进去的话，可以看到是需要对它进行求逆的，这也是我被问得最多的问题：由于 $\hat{\Sigma}$ singular 了，导致求逆失败，于是代码也就运行失败了。

从 (3) 可以大致看到什么时候 $\hat{\Sigma}$ 会 singular：由于该式的形式，可以知道 $\hat{\Sigma}$ 的 rank 最大不会超过 n，所以如果 $d > n$，也就是数据的维度大于数据点的个数的话，得到的估计值 $\hat{\Sigma}$ 肯定是不满秩的，于是就 singular 了。当然，即使不是这么极端的情况，如果 n 比较小或者 d 比较大的时候，仍然会有很大的几率出现不满秩的。

这个问题还可以从更抽象一点的角度来看。我们之前计算过估计 Covariance 矩阵实际上是要估计 $d(d+1)/2$ 这么多个参数，如果 d 比较大的话，这个数目也会变得非常多，也就意味着模型的复杂度很大，对于很复杂的模型，如果训练数据的量不够多，则无法确定一个唯一的最优模型出来，这是机器学习中非常常见的一个问题，通常称为 overfitting。解决 overfitting 问题的方法有不少，最直接的就是用更多的训练数据（也就是增大 n），当然，有时候训练数据只有那么多，于是就只好从反面入手，降低模型复杂度。

在降低模型复杂度方面最常用的方法大概是正则化了，而正则化最简单的例子也许是 Ridge Regression 了，通过在目标函数后面添加一项关于参数的 ℓ_2-norm 的项来对模型参数进行限制；此外也有诸如 LASSO 之类的使用 ℓ_1-norm 来实现稀疏性的正则化，这个我们在之前也曾经简单介绍过。

在介绍估计 Covariance 的正则化方法之前，我们首先来看一种更直接的方法，直接限制模型的复杂度：特别地，对于我们这里 Gaussian 的例子，比如，我们可以要求协方差矩阵是一个对角阵，这样一来，对于协方差矩阵，我们需要估计的参数个数就变成了 d 个对角元素，而不是原来的 $d(d+1)/2$ 个。大大减少参数个数的情况下，overfitting 的可能性也极大地降低了。

注意 Covariance Matrix 的非对角线上的元素是对应了某两个维度之间的相关性 (Correlation) 的，为零就表示不相关。而对于 Gaussian 分布来说，不相关和独立 (Independence) 是等价的，所以说在我们的假设下，向量 x 的各个维度之间是相互独立的，换句话说，他们的联合概率密度可以分解为每个维度各自的概率密度（仍然是 Gaussian 分布）的乘积，这个特性使得我们可以很方便地对协方差矩阵进行估计：只有每个维度上当做一维的 Gaussian 分布各自独立地进行参数估计就可以了。注意这个时候除非某个维度上的坐标全部是相等的，否则 Covariance 矩阵对角线上的元素都能保证大于零，也就不会出现 singular 的情况了。

那么直观上将协方差矩阵限制为对角阵会产生什么样的模型呢？我们不妨看一个二维的简单例子。下面的代码利用了 scikit-learn 里的 GMM 模块来估计一个单个 component 的 Gaussian，它在选项里已经提供了 Covariance 的限制，这里我们除了原始的 `full` 和对角阵的 `diag` 之外，还给出一个 `spherical`，它假定协方差矩阵是 αI 这样的形式，也就是说，不仅是对角阵，而且所有对角元素都相等。下面的代码把三种情况下 fit 出来的模型画出来，在图 1 中可以看到。

```
import numpy as np
import pylab as pl
from sklearn import mixture

n_samples = 300
c_types = ['full', 'diag', 'spherical']

np.random.seed(0)
C = np.array([[0., -0.7], [3.5, 1.7]])
X_train = np.dot(np.random.randn(n_samples, 2), C)

pl.figure(dpi=100,figsize=(3,3))
pl.scatter(X_train[:, 0], X_train[:, 1], .8)
pl.axis('tight')
```

```
pl.savefig('GaussianFit-data.svg')
pl.close()

for c_type in c_types:
    clf = mixture.GMM(n_components=1, covariance_type=c_type)
    clf.fit(X_train)

    x = np.linspace(-15.0, 20.0, num=200)
    y = np.linspace(-10.0, 10.0, num=200)
    X, Y = np.meshgrid(x, y)
    XX = np.c_[X.ravel(), Y.ravel()]
    Z = np.log(-clf.eval(XX)[0])
    Z = Z.reshape(X.shape)

    pl.figure(dpi=100,figsize=(3,3))
    CS = pl.contour(X, Y, Z)
    pl.scatter(X_train[:, 0], X_train[:, 1], .8)

    pl.axis('tight')
    pl.savefig('GaussianFit-%s.svg' % c_type)
    pl.close()
```

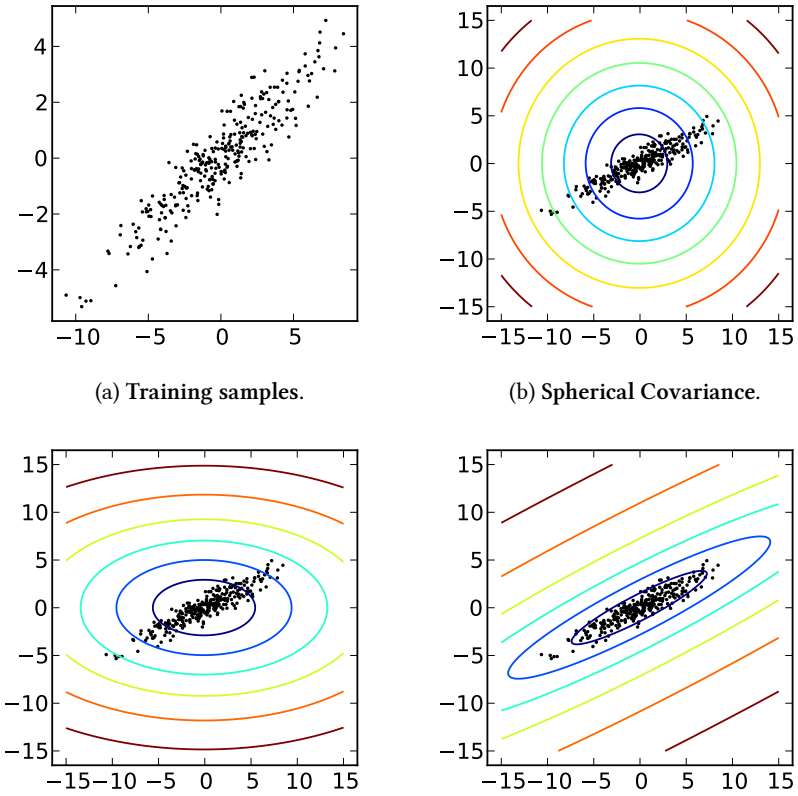
可以看到，从直观上来讲：spherical Covariance 的时候就是各个维度上的方差都是一样的，所以形成的等高线在二维上是正圆。而 diagonal Covariance 则可以 capture 更多的特征，因为它允许形成椭圆的等高线，但是它的限制是这个椭圆是不能旋转的，永远是正放的。Full Covariance 是 fit 最好的，但是前提是数据量足够来估计这么多的参数，虽然对于 2 维这样的低维度通常并不是问题，但是维度高了之后就经常会变得很困难。

一般来说，使用对角线的协方差矩阵是一种不错的降低模型复杂度的方式，在 scikit-learn 的 GMM 模块中这甚至是默认的。不过如果有时候你觉得这样的限制实在是和你的真实数据想去甚远的话，也有另外的处理方式。最简单的处理办法是在求出了 Covariance 矩阵的估计值之后直接加上一个 λI，也就是在对角线上统一加上一个很小的正数 λ，通常称作正则化系数。例如，Matlab 自带的统计工具箱里的 `gmdistribution.fit` 函数就有一个可选参数叫做 `Regularize`，就是我们这里说的 λ。为什么加上 λI 之后就不会 singular 了呢？首先 Σ 本身至少肯定是半正定的，所以 $\forall v \neq 0$:

$$v^T(\Sigma + \lambda I)v = v^T\Sigma v + \lambda v^T I v \geq 0 + \lambda\|v\|^2 > 0$$

所以这样之后肯定就是正定的了。此外在 scikit-learn 的 GMM 学习的函数中也有一个类似的参数 `min_covar`，它的文档里说的是 “Floor on the

Figure 1: .....



(a) **Training samples.** (b) **Spherical Covariance.**

(c) **Diagonal Covariance.** (d) **Full Covariance.**

diagonal of the covariance matrix to prevent overfitting. Defaults to 1e-3." 感觉好像是如果原来的 Covariance 矩阵对角线元素小于该参数的时候就将它设置为该参数，不过这样的做法是不是一定产生正定的协方差矩阵似乎就不像上面那种那样直观可以得出结论了。

不过这样的做法虽然能够解决 singular 的问题，但是总让人有些难以信服：你莫名其妙地在我的协方差矩阵上加上了一些东西，这到底是什么意思呢？最简单的解释可以从式 (1) 那里协方差矩阵你的地方来看，对于矩阵求逆或者解方程的时候出现 singular 的情况，加上一个 λI 也算是数值上的一种标准处理方式，叫做 Tikhonov Regularization，Ridge Regression 本身也可以通过这个角度来解释。从数值计算的角度来说，这样处理能够增加数值稳定性，直观地来讲，稳定性是指 Σ 元素的微小数值变化，不会使得求逆（或者解方程之类的）之后的解产生巨大的数值变化，如果 Σ 是 singular 的话，通常就不具有这样的稳定性了。此外，随着 regularization coefficient λ 逐渐趋向于零，对应的解也会逐渐趋向于真实的解。

如果这个解释还不能令出信服的话，我们还可以从 Bayes 推断的角度来看这个问题。不过这本身是一个很大的话题，我在这里只能简略地讲一个思路，想了解更多的同学可以参考《Pattern Recognition and Machine

Learning》第二章或者其他相关的书籍。

总的来说，我们这里要通过 Maximum a posteriori (MAP) Estimation 来对协方差矩阵进行估计。做法是首先为 Σ 定义一个先验分布 (prior) $P(\Sigma)$，然后对于数据 $x_1, \ldots, x_n$，我们根据 Bayes 公式，有

$$P(\Sigma|x_1, \ldots, x_n) = \frac{P(\Sigma)P(x_1, \ldots, x_n|\Sigma)}{P(x_1, \ldots, x_n)} \tag{4}$$

其中等式左边称作后验 (posterior)，右边红色部分是似然 (likelihood)，蓝色部分是用于将概率分布 normalize 到全概率为 1 的，通常并不需要直接通过 x 去计算，而只要对求出的概率分布进行归一化就可以了。MAP 的思路是将 posterior 最大的那个 Σ 作为最佳估计值，直观上很好理解，就是给定了数据点之后 “最可能的” 那个 Σ。而计算上我们是通过 (4) 来实现的，因为分母是与 Σ 无关的，所以在计算的时候（因为只要比较相对大小嘛）忽略掉，这样一来，就可以看到，其实 MAP 方法和刚才我们用的最大似然 (MLE) 方法唯一的区别就在于前面乘了一个先验分布。

虽然从最终的计算公式上来看差别很细微，但是 MAP 却有很多好处。和本文最相关的当然是先验的引入，这在很大程度上和我们平时用正则化的目的是一样的：加入我们自己对模型的先验知识或者假设或者要求，这样可以避免在数据不够的时候产生 overfitting 。此外还有另一个值得一提的好玩特性就是 MAP 可以在线计算：注意到后验分布本身也是关于 Σ 的一个合法分布，所以在我们计算出后验之后，如果又得到了新的训练数据，那么我们可以将之前算出来的后验又作为先验，代入新的一轮的计算，这样可以一直不断地重复下去。`:D`

不过，虽然 MAP 框架看上去很美，但是在 general 的情况下计算却可能会变得很复杂，因此，虽然理论上来说，我们可以把任何先验知识通过先验分布的方式加入到模型中来，但是在实际中先验的选择往往是根据 “哪个更好计算” 而不是根据 “哪个更合理” 的准则来做出的。基于这个准则，有了 Conjugate Prior 的概念，简单地来说，如果一个 prior 和 likelihood 相乘再归一化之后得到的 posterior 形式上是和 prior 一样的话，那么它就是该 likelihood 的一个 conjugate prior。

选择 conjugate prior 的好处是显而易见的，由于先验和后验的形式是一样的，都是某一类型的分布，只是参数的值发生了变化，所以说可以看成是在观察到数据之后对模型参数进行修正（注意这里的模型是关于 Σ 的，而 Σ 本身又是关于数据的模型——一个 Gaussian 分布——的参数，不要搞混淆了），并且这种修正有时候可以得到非常直观的解释，不过我在这里就不细说了。此外，由于形式没有发生变化，所以在使用在线计算的时候，每一次观察到新的数据所使用的计算公式都还是一样的。

回到我们的问题，关于 Gaussian 分布的协方差矩阵的 Conjugate Prior 是一个叫做 Inverse Wishart Distribution 的家伙，它的概率密度函数是这个样子的：

$$\mathcal{W}^{-1}(\Sigma|\Phi,\nu) = \frac{\det(\Phi)^{\nu/2}}{2^{\nu d/2}\Gamma_d\left(\frac{\nu}{2}\right)}\det(\Sigma)^{-\frac{\nu+d+1}{2}}\exp\left(-\frac{1}{2}\mathrm{tr}\left(\Phi\Sigma^{-1}\right)\right)$$

其中 Σ 和参数 Φ 都是 $d\times d$ 的正定矩阵，而 $\Gamma_d(\cdot)$ 则是 Multivariate Gamma Function。是不是看上去有点恐怖？`^_^bbbb` 我也觉得有点恐怖……接下来让我们来看 MAP，首先将这个 prior 乘以我们的 likelihood，和 MLE 一样的，我们在 log 空间中计算，所以对整个式子取对数，然后丢掉和 Σ 无关的常数项（因为是关于 Σ 最大化嘛），得到下面的式子：

$$-\frac{\nu+d+1+n}{2}\log\det(\Sigma)-\frac{1}{2}\mathrm{tr}\left(\Phi\Sigma^{-1}\right)+\sum_{i=1}^{n}-\frac{1}{2}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)$$

可以看到现在已经变得不是那么复杂了，形式上和我们之前的 (2) 是一样的，只是多了红色的一项，接下来对 Σ 进行求导并令导数等于零，类似地得到如下方程：

$$0=\frac{\nu+d+1+n}{2}\Sigma^{-1}-\frac{1}{2}\Sigma^{-1}\Phi\Sigma^{-1}-\frac{1}{2}\Sigma^{-1}\left(\sum_{i=1}^{n}(x_i-\mu)(x_i-\mu)^T\right)\Sigma^{-1}$$

于是得到最优解为

$$\tilde{\Sigma}=\frac{\Phi+\sum_{i=1}^{n}(x_i-\mu)(x_i-\mu)^T}{\nu+d+1+n}=\frac{\Phi+n\hat{\Sigma}}{\nu+d+1+n}$$

这里 $\hat{\Sigma}$ 是之前的最大似然估计。接下来如果我们将我们 prior 中的参数 Φ 选为 $n\alpha I$，就可以得到 MAP 估计为：

$$\begin{aligned}\tilde{\Sigma}&=\frac{n}{\nu+d+1+n}\hat{\Sigma}+\frac{n\alpha}{\nu+d+1+n}I\\&=\frac{n}{\nu+d+1+n}\left(\hat{\Sigma}+\alpha I\right)\end{aligned}$$

就得到了我们刚才的正则化的形式，虽然有一点细微的区别就是这里前面多了一个缩放系数。这样一来，我们就得到了这种正则化方法的一个看起来比较靠谱的解释：实际上就是我们先验地假设协方差矩阵满足一个均值为 spherical 形状（所有对角元素都相等的一个对角阵）的 Inverse Wishart 分布（关于 Inverse Wishart 分布的均值的公式可以参见 wikipedia），然后通过数据来对这个先验进行修正得到后验概率，并从后验中取了概率（密度）最大的那个 Σ 作为最终的估计。

这种方式比暴力地直接强制要求协方差矩阵具有 diagonal 或者 spherical 形式要温柔得多，并且随着训练数据的增多，先验的影响也会逐渐

减小，从而趋向于得到真实的参数，而强制结构限制的结果则是不论你最后通过什么手段搞到了多少数据，如果真实模型本身不是 diagonal 或者 spherical 的话，那么你的参数估计的 bias 将永远都无法被消除，这个我们已经在图 1 中直观地看到了。当然，在训练数据本身就不足的情况下，两种选择谁好谁好就没有定论了，不要忘了，强制结构限制的计算复杂度要小很多。:)

最后，对于不想看长篇大论的人，我做一下简单的总结：如果碰到估计 Gaussian 的协方差矩阵时结果 singular 的情况，解决方法一般有两种：

1. 直接限制协方差矩阵的结构，比如要求它是一个对角阵，这种情况下数据的各个维度将会是独立的，所以可以把每个维度看成一个一维的 Gaussian 分布来单独进行估计。
2. 在估计出来的协方差矩阵的对角线上统一加上一个很小的正数，使得它变成正定的。这种方法的合理性可以通过正则化或者 MAP 估计来进行解释。

然后再补充几点不知道该放在哪里的注意事项：

- prior、posterior 和 likelihood 比较容易搞混淆，记住 prior 和 posterior 都是关于参数（比如我们这里就是 Σ）的分布，prior 是我们所假设的参数本来的分布，而 posterior 则是在观察到训练数据之后得到的条件分布。而 likelihood 则完全不一样，一方面它是关于数据 x 的，另一方面它没有被归一化，所以也并不是一个合法的概率分布。
- 标量值函数关于矩阵变量的求导原理上其实就是和多元标量值函数求导一样的，不过求起来非常繁琐，一般不会自己算，网上可以找到一个叫做《Matrix Cookbook》的小册子，里面有搜集了各种常用的矩阵求导的公式，基本上直接从那里查询就可以解决大部分问题了。