VII-3-4. 混合ガウスモデルによる非階層的クラスター分析

VII-3-4-1. 混合ガウスモデルの考え方

VII-3-3 の K-means 法のところで、K-means は EM アルゴリズムを使っているのだと説明をしました。同じような考え方で、あらかじめ中心点を与えて EM アルゴリズムで中心点とクラスの分け方を確率的に最適化するのが混合ガウスモデルです。K-means 法と似ているのですが、確率的な考え方がはっきりと見えているのが混合ガウスモデルです。K-means 法は混合ガウスモデルの特殊例だと考えた方が適切かもしれません。混合ガウスモデルという名称ですが、実際に使われている確率密度関数は、ガウス分布の一種である多変量正規分布ですから、混合正規分布モデルと言った方がわかりやすいかもしれません。混合ガウスモデルでは、一つひとつの点は与えられたクラスの数(K)のすべてのクラスに属する可能性を持っているという確率論的な考え方をします。そのような確率の重なり合いが、混合ガウスモデルです。そして、一つのクラスの中でその点のデータが得られる確率は、多次元正規分布にしたがうと考えます。

$$P(\boldsymbol{x_i}|z_{ik}=1)=\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu}_k,\Sigma_k)$$

$\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu}_k,\Sigma_k)$は$\boldsymbol{x_i}$が分布中心が$\boldsymbol{\mu}_k$、分散（分散共分散行列）が$\Sigma_k$で多次元正規分布するクラス$k$に属するときの、そのクラス内での確率密度の意味です。実際に式で書くと、

$$\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu}_k,\Sigma_k)=\frac{1}{\sqrt{(2\pi)^P|\Sigma_k|}}e^{\left\{-\frac{1}{2}(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T{\Sigma_k}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu_k})\right\}} \qquad \text{i}$$

です。項目間の相関を考えなければ、Σ_kは普通に分散と考えて、行列ではなくて数値（スカラー）を与えても良いのですが、ここでは$\boldsymbol{x}$の要素間に相関がある可能性を考えて分散共分散行列にしています。混合ガウスモデルでは、このような確率密度関数をK個重ね合わせて混合ガウス関数を作ります(GMF)。この時、各クラスの分布の大きさが違いますから、その大きさの比率を表す混合係数π_kを掛けてから足し合わせます。

$$GMF=\sum_{k=1}^{K}\pi_k\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu}_k,\Sigma_k) \qquad \text{ii}$$

混合係数は制約を付けて、$0\le\pi_k\le 1$ $\sum_{k=1}^{K}\pi_k=1$としておきます。図の 147 に 1 次元のデータで混合ガウス分布の作り方を示しました。

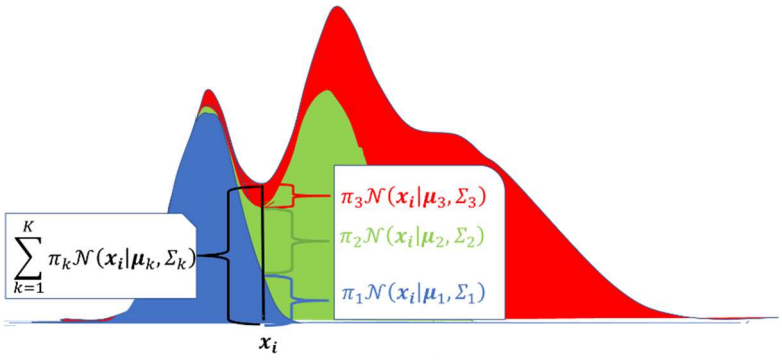

図 147. 正規分布を重ね合わせて混合ガウス分布を作る

これらが決まれば、点$\boldsymbol{x_i}$がそれぞれのクラスに属する確率$\boldsymbol{\gamma}_{ik}$が事後的に次のように決まります。

$$\boldsymbol{\gamma}_{ik} = \frac{\pi_k \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu}_k, \Sigma_k)}{GMF} = \frac{\pi_k \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu}_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu}_j, \Sigma_j)} \quad \text{iii}$$

$\boldsymbol{\gamma}_{ik}$は負担率(responsibility)といい、次のように表します。

$$\boldsymbol{\gamma_i} = (\gamma_{i1} \quad \cdots \quad \gamma_{iK})^{\mathbf{T}}$$

$$\sum_{k=1}^{K} \gamma_{ik} = 1$$

たとえば、

$$\boldsymbol{\gamma_1} = (0.5 \quad 0.3 \quad 0.2)^{\mathbf{T}}$$

$$\boldsymbol{\gamma_2} = (0.2 \quad 0.4 \quad 0.4)^{\mathbf{T}}$$

という感じです。

K-means 法では、クラスの所属を one hot エンコーディングを使って、$\boldsymbol{r_i} = (0 \quad 1 \quad 0 \quad \cdots \quad 0)$のように 0 か 1 で記述しました。混合ガウスモデルでも、最終的にはどれか一つのクラスに属するのだから one hot エンコーディングで表される潜在変数(latent variable)があるのです。一応、これを$\boldsymbol{r}$と区別して

$$\boldsymbol{z_i} = (0 \quad 1 \quad 0 \quad \cdots \quad 0)$$

のように表します。すると、γ_iは$\boldsymbol{x_i}$が与えられた時にそれが$z_{ik} = 1$である確率$P(z_{ik} = 1|\boldsymbol{x_i})$ということになります。$\boldsymbol{x_i}$は多次元正規分布していると考えているのだから、$\boldsymbol{x_i}$というデータが得られるのも確率的な事象で、$P(\boldsymbol{x_i})$と表せます。ここでベイズの定理を思い出します。念のために最初からベイズの定理を説明します。全体として、$\boldsymbol{x_i}$**かつ**$z_{ik} = 1$が起こる確率$P(\boldsymbol{x_i} \cap z_{ik} = 1)$は、$\boldsymbol{x_i}$が起こる確率$P(\boldsymbol{x_i})$と、$\boldsymbol{x_i}$が起きたことを前提に$z_{ik} = 1$が起きる確率$P(z_{ik} = 1|\boldsymbol{x_i})$の積です。確率$P(\boldsymbol{x_i} \cap z_{ik} = 1)$は別ルートからも式が作れて、$z_{ik} = 1$であった後に、それが$\boldsymbol{x_i}$である確率、つまり、$P(z_{ik} = 1)$と$P(\boldsymbol{x_i}|z_{ik} = 1)$の積も$P(\boldsymbol{x_i} \cap z_{ik} = 1)$です。

$$P(\boldsymbol{x_i} \cap z_{ik} = 1) = P(z_{ik} = 1|\boldsymbol{x_i})P(\boldsymbol{x_i}) = P(\boldsymbol{x_i}|z_{ik} = 1)P(z_{ik} = 1)$$

この 2 番目の式と 3 番目の式から

$$P(z_{ik} = 1|\boldsymbol{x_i})P(\boldsymbol{x_i}) = P(\boldsymbol{x_i}|z_{ik} = 1)P(z_{ik} = 1)$$

$$P(z_{ik} = 1|\boldsymbol{x_i}) = \frac{P(\boldsymbol{x_i}|z_{ik} = 1)P(z_{ik} = 1)}{P(\boldsymbol{x_i})} \quad \text{iv}$$

と変形したものが、ベイズの公式です。右辺の分子の$P(z_{ik} = 1)$はクラス k に属する確率でこれを混合係数(π_k)と言います。混合係数(π_k)の総和は 1 です$(\sum_{k=1}^{K} \pi_k = 1)$。右辺の分母$P(\boldsymbol{x_i})$は周辺確率で、各クラスごとに、$\boldsymbol{x_i}$の確率密度があってそれらに混合係数で重みをつけて足し合わせたものです。

$$P(\boldsymbol{x_i}) = \sum_{k=1}^{K} \pi_k P(\boldsymbol{x_i}|z_{ik} = 1)$$

この分母のところがガウス関数を重ね合わせた関数（混合ガウス関数）になっています。し

っかりとベイズの式を書き換えると、

$$P(z_{ik}=1|\boldsymbol{x_i}) = \frac{\pi_k P(\boldsymbol{x_i}|z_{ik}=1)}{\sum_{j=1}^{K}\pi_j P\left(\boldsymbol{x_i}|z_{ij}=1\right)} \quad \text{v}$$

ところで、$P(\boldsymbol{x_i}|z_{ik}=1) = N(\boldsymbol{x_i}|\boldsymbol{\mu_k},\Sigma_k)$ですから、

$$P(z_{ik}=1|\boldsymbol{x_i}) = \frac{\pi_k N(\boldsymbol{x_i}|\boldsymbol{\mu_k},\Sigma_k)}{\sum_{j=1}^{K}\pi_j \mathcal{N}\left(\boldsymbol{x_i}|\boldsymbol{\mu}_j,\Sigma_j\right)} \quad \text{vi}$$

この式は式 iii として示した、$\boldsymbol{x_i}$**というデータが、クラス**kに属する確率$\boldsymbol{\gamma_i}$そのものです。
混合ガウス分布全体で考えると、$\boldsymbol{x_i}$というデータが得られる確率密度が GMF ですから、

$\pi_k N(\boldsymbol{x_i}|\boldsymbol{\mu_k},\Sigma_k)$の$k=1$ からKの合計が GMF です。

$$GMF = \sum_{k=1}^{K}\pi_k \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu}_k,\Sigma_k)$$

これは$\boldsymbol{x_i}$が得られる尤度ですから、全体としての尤度はその総積となります。

$$L = \prod_{i=1}^{n}\sum_{k=1}^{K}\pi_k \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu}_k,\Sigma_k)$$

$\pi_k,\boldsymbol{\mu_k},\Sigma_k$を最適化するというのは、この尤度を最大化する$\pi_k,\boldsymbol{\mu_k},\Sigma_k$を見つけるということです。普通は$L$を微分して、

$$\frac{\partial L}{\partial \pi_k} = 0$$

$$\frac{\partial L}{\partial \boldsymbol{\mu_k}} = 0$$

$$\frac{\partial L}{\partial \Sigma_k} = 0$$

とおいて、$1,\cdots,K$の$\pi_k,\boldsymbol{\mu_k},\Sigma_k$についてこれを解けば良いのですが、変数分離できなければ通常のやり方では連立微分方程式を解けません。しかし、とりあえず計算の都合上対数尤度をとります。対数をとっても大小関係は変わりませんから、対数尤度の最大値を与える係数は尤度の最大値を与えます。対数尤度で計算する方が楽です。

$$LL = \sum_{i=1}^{n}\log\sum_{j=1}^{K}\pi_j \mathcal{N}\left(\boldsymbol{x_i}|\boldsymbol{\mu_j},\Sigma_j\right)$$

最初のシグマは総和ですから、それ以下が微分できれば問題ありませんが、log sum（$\log\sum_{j=1}^{K}\pi_j N\left(\boldsymbol{x_i}|\boldsymbol{\mu_j},\Sigma_j\right)$ができてしまいます。ここで何か工夫が必要です。$\pi_j$, $\boldsymbol{\mu_j}$,Σ_jで微分しますが、とりあえず何で微分するか決まっていないのでωで微分することにしておきます。

$$\frac{\partial LL}{\partial \omega} = \frac{\partial\sum_{i=1}^{n}\log\sum_{j=1}^{K}\pi_j \mathcal{N}\left(\boldsymbol{x_i}|\boldsymbol{\mu_j},\Sigma_j\right)}{\partial\omega} = \sum_{i=1}^{n}\frac{\partial\left(\log\sum_{j=1}^{K}\pi_j \mathcal{N}\left(\boldsymbol{x_i}|\boldsymbol{\mu_j},\Sigma_j\right)\right)}{\partial\omega}$$

とおいて、次に対数の微分の公式($(\log f(x))' = \frac{f(x)'}{f(x)}$)を思い出します。

$$\frac{\partial\, log_e f(\omega)}{\partial \omega} = \frac{1}{f(\omega)} \frac{\partial f(\omega)}{\partial \omega}$$

$$f(\omega) = \sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} | \boldsymbol{\mu_j}, \Sigma_j\right)$$

$$\frac{\partial f(\omega)}{\partial \omega} = \frac{\partial \left(\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} | \boldsymbol{\mu_j}, \Sigma_j\right)\right)}{\partial \omega}$$

$$\frac{\partial\, log_e f(\omega)}{\partial \omega} = \frac{\partial \log \sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} | \boldsymbol{\mu_j}, \Sigma_j\right)}{\partial \omega}$$

としてみると、

$$\frac{\partial \log \sum_{k=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} | \boldsymbol{\mu_j}, \Sigma_j\right)}{\partial \omega} = \frac{1}{\sum_{k=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} | \boldsymbol{\mu_j}, \Sigma_j\right)} \frac{\partial \left(\sum_{\mathrm{j}=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} | \boldsymbol{\mu_j}, \Sigma_j\right)\right)}{\partial \omega}$$

となって、対数を外すことが出来ました。これは知っていれば簡単にできる数学的なテクニックです。右辺の偏微分のところは

$$\sum_{k=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} | \boldsymbol{\mu_j}, \Sigma_j\right) = \pi_1 \mathcal{N}(\boldsymbol{x_i} | \boldsymbol{\mu_1}, \Sigma_1) + \cdots + \pi_K \mathcal{N}(\boldsymbol{x_i} | \boldsymbol{\mu_K}, \Sigma_K)$$

という形になっていて、一つひとつの項には添え字が同じものしか含まれません。

$$\frac{\partial \pi_j \mathcal{N}\left(\boldsymbol{x_i} | \boldsymbol{\mu_j}, \Sigma_j\right)}{\partial \omega_l} = 0 \quad when\ j \neq l$$

ですから

$$\frac{\partial \left(\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} | \boldsymbol{\mu_j}, \Sigma_j\right)\right)}{\partial \omega_k} = \frac{\partial \left(\pi_k \mathcal{N}(\boldsymbol{x_i} | \boldsymbol{\mu_k}, \Sigma_k)\right)}{\partial \omega_k}$$

となります。

$$\frac{\partial LL}{\partial \omega_k} = \sum_{i=1}^{n} \frac{1}{\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} | \boldsymbol{\mu_j}, \Sigma_j\right)} \frac{\partial \left(\pi_k \mathcal{N}(\boldsymbol{x_i} | \boldsymbol{\mu_k}, \Sigma_k)\right)}{\partial \omega_k} = \frac{1}{\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} | \boldsymbol{\mu_j}, \Sigma_j\right)} \sum_{i=1}^{n} \frac{\partial \left(\pi_k \mathcal{N}(\boldsymbol{x_i} | \boldsymbol{\mu_k}, \Sigma_k)\right)}{\partial \omega_k}$$

これで$\frac{\partial \left(\pi_k \mathcal{N}(\boldsymbol{x_i} | \boldsymbol{\mu_k}, \Sigma_k)\right)}{\partial \omega_k}$の微分だけを考えれば良いことになります。この先は、ω_kが何かで式が微分式が変わってきます。特にπ_kは制約条件付きで、しかも、多次元正規分布にかかわる変数ではありません。π_kの最適化は別途考えることにして、もう少し、微分式を変形します。π_kは確率の重なり合い方だから、個々のクラスの多変量正規分布密度にかかわらないので微分の外に出します。

$$\frac{\partial \left(\pi_k \mathcal{N}(\boldsymbol{x_i} | \boldsymbol{\mu_k}, \Sigma_k)\right)}{\partial \boldsymbol{\omega_k}} = \frac{\pi_k \partial \mathcal{N}(\boldsymbol{x_i} | \boldsymbol{\mu_k}, \Sigma_k)}{\partial \boldsymbol{\omega_k}}$$

ここで、Log sum を回避した時と同じテクニックを逆向きに使います。対数の微分の公式

$((\log f(x))' = \frac{f(x)'}{f(x)})$をすると、

$$f(x)' = f(x)(\log f(x))'$$

という公式が得られます。

$$f(x)' = \frac{\partial\left(\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k)\right)}{\partial \boldsymbol{\omega_k}}$$

とすると

$$f(x) = \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k)$$

$$(\log_e f(x))' = \frac{\partial \log_e \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k)}{\partial \boldsymbol{\omega_k}}$$

だから

$$\frac{\partial\left(\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k)\right)}{\partial \boldsymbol{\omega_k}} = \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k)\frac{\partial \log_e \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k)}{\partial \boldsymbol{\omega_k}}$$

$$\frac{\partial\left(\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k)\right)}{-\partial \boldsymbol{\omega_k}} = \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k)\frac{\partial \log_e \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k)}{\partial \boldsymbol{\omega_k}}$$

と一般化して書くことが出来ます。対数尤度の微分にもどして、全体を見ると、

$$\frac{\partial LL}{\partial \boldsymbol{\omega_k}} = \sum_{i=1}^{n} \frac{\partial\left(\pi_k \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k)\right)}{\partial \boldsymbol{\omega_k}} \Big/ \sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i}|\boldsymbol{\mu_j}, \Sigma_j\right) = \sum_{i=1}^{n} \frac{\pi_k}{\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i}|\boldsymbol{\mu_j}, \Sigma_j\right)} \frac{\partial\left(\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k)\right)}{\partial \boldsymbol{\omega_k}}$$

$$= \sum_{i=1}^{n} \frac{\pi_k}{\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i}|\boldsymbol{\mu_j}, \Sigma_j\right)} \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k) \frac{\partial(\log_e \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k))}{\partial \boldsymbol{\omega_k}}$$

$$= \sum_{i=1}^{n} \frac{\pi_k \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i}|\boldsymbol{\mu_j}, \Sigma_j\right)} \frac{\partial(\log_e \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k))}{\partial \boldsymbol{\omega_k}}$$

黄色の部分が対数の微分を変形した公式を使った変形です。この式の青い部分をよく見ると、これは式 iii の負担率γ_{ik}です。

結局、対数尤度の微分は

$$\frac{\partial LL}{\partial \omega} = \sum_{i=1}^{n} \gamma_{ik} \frac{\partial(\log_e \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k))}{\partial \boldsymbol{\omega_k}}$$

ときわめてシンプルな形になりました。ただ、微妙な議論を巧妙なロジックですり抜けているので気が付かないのですが、この結論は少し怪しげなところがあります。見てわかるとおり$\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k)$は$\boldsymbol{\mu_k}$の関数なのだから、それらを組み合わせて出来る$\gamma_{ik}$は$\boldsymbol{\omega_k}$の関数で、$\pi_k$, $\boldsymbol{\mu_k}$, Σ_kによって変わります。それを微分式の外に取り出して、π_k, $\boldsymbol{\mu_k}$, Σ_kを最適化しても意味がないのではないかということです。確かにγ_{ik}はπ_k, $\boldsymbol{\mu_k}$, Σ_kの関数で内生的に決まります。しかし、それだと微分方程式が解けません。この式の意味は、γ_{ik}を外生変数として与えれば、微分方程式を解くことが出来て、その条件下でπ_k, $\boldsymbol{\mu_k}$, Σ_kを最適化できるということ

です。何らかのπ_k，$\boldsymbol{\mu_k}$ ，Σ_kを与えて、内生的に期待値としてγ_{ik}を求め（E ステップ）、その期待値を外生的に使って尤度を最大化する方向で$\Sigma_j\pi_k$，$\boldsymbol{\mu_k}$ ，Σ_kを最適化して更新する（M ステップ）。これを繰り返して、漸近的に最適なπ_k，$\boldsymbol{\mu_k}$ ，Σ_kに近づいていくという方法が考えられます。こういう解法を EM アルゴリズムと言います。混合ガウスモデルは EM アルゴリズムで解きます。

いずれにしても、微分としては$\frac{\partial(\log_e \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k},\Sigma_k))}{\partial \boldsymbol{\omega_k}}$を解けば良いということです。

$$\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k},\Sigma_k)=\frac{1}{\sqrt{(2\pi)^P|\Sigma_k|}}e^{\left\{-\frac{1}{2}(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T{\Sigma_k}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu_k})\right\}}$$

対数をとって

$$\log_e \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k},\Sigma_k)=-\frac{P}{2}\log_e(2\pi)-\frac{1}{2}\log_e|\Sigma_k|+\log_e e^{\left\{-\frac{1}{2}(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T{\Sigma_k}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu_k})\right\}}=$$

$$=\log_e \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k},\Sigma_k)=-\frac{P}{2}\log_e(2\pi)-\frac{1}{2}\log_e|\Sigma_k|-\frac{1}{2}(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T{\Sigma_k}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu_k})$$

これを微分します。

$$\frac{\partial \log_e \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k},\Sigma_k)}{\partial \boldsymbol{\omega_k}}=-\frac{P}{2}\frac{\partial \log_e(2\pi)}{\partial \boldsymbol{\omega_k}}-\frac{1}{2}\frac{\partial \log_e|\Sigma_k|}{\partial \boldsymbol{\omega_k}}-\frac{1}{2}\frac{\partial\left\{(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T{\Sigma_k}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu_k})\right\}}{\partial \boldsymbol{\omega_k}}$$

まず、$\boldsymbol{\omega_k}=\boldsymbol{\mu_k}$の場合です。

$$\frac{\partial(\log_e \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k},\Sigma_k))}{\partial \boldsymbol{\mu_k}}=-\frac{P}{2}\frac{\partial \log_e(2\pi)}{\partial \boldsymbol{\mu_k}}-\frac{1}{2}\frac{\partial \log_e|\Sigma_k|}{\partial \boldsymbol{\mu_k}}-\frac{1}{2}\frac{\partial\left\{(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T{\Sigma_k}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu_k})\right\}}{\partial \boldsymbol{\mu_k}}$$

黄色の部分は、微分される関数が$\boldsymbol{\mu_k}$の関数ではないので定数の微分として扱われ微分値は 0 です。

$$\frac{\partial(\log_e \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k},\Sigma_k))}{\partial \boldsymbol{\mu_k}}=-\frac{1}{2}\frac{\partial\left\{(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T{\Sigma_k}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu_k})\right\}}{\partial \boldsymbol{\mu_k}}$$

$$\frac{\partial LL}{\partial \boldsymbol{\mu_k}}=\sum_{i=1}^{n}\gamma_{ik}\frac{-\frac{1}{2}\partial\left((\boldsymbol{x_i}-\boldsymbol{\mu_k})^T{\Sigma_k}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu_k})\right)}{\partial \boldsymbol{\mu_k}}$$

緑色の微分の解について考えます一見、複雑に見えますが、$(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T\Sigma_k{}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu_k})$が$\vec{\boldsymbol{x}}^{\boldsymbol{T}}\boldsymbol{A}\,\vec{\boldsymbol{x}}$の形の二次形式だということに気が付くでしょう。二次形式はスカラーになりますから、スカラーをベクトルで微分するということです。

スカラーS をベクトル$\vec{x}$で微分するとは、ベクトルの一つ一つの要素でスカラーを微分することですから、

$$\frac{dS}{d\vec{x}} = \begin{pmatrix} \frac{dS}{dx_1} \\ \frac{dS}{dx_2} \\ \vdots \\ \frac{dS}{dx_P} \end{pmatrix} \text{ただし} \vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_P \end{pmatrix}$$

内積もスカラーですから

$$\vec{\boldsymbol{a}} = \begin{pmatrix} a_1 \\ \vdots \\ a_P \end{pmatrix}, \qquad \vec{\boldsymbol{x}} = \begin{pmatrix} x_1 \\ \vdots \\ x_P \end{pmatrix}$$

$$\vec{\boldsymbol{a}}^{\boldsymbol{T}} \cdot \vec{\boldsymbol{x}} = \sum_{p=1}^{P} a_p \, x_p$$

$$\frac{\partial \vec{\boldsymbol{a}}^{\boldsymbol{T}} \cdot \vec{\boldsymbol{x}}}{\partial \vec{\boldsymbol{x}}} = \begin{pmatrix} \frac{\partial \sum_{p=1}^{P} a_p \, x_p}{\partial x_1} \\ \vdots \\ \frac{\partial \sum_{p=1}^{P} a_p \, x_p}{\partial x_P} \end{pmatrix}$$

$$\frac{\partial a_p x_p}{\partial x_j} = a_j (when \; p = j), \qquad 0 (when \; p \neq j)$$

$$\frac{\partial \vec{\boldsymbol{a}}^{\boldsymbol{T}} \cdot \vec{\boldsymbol{x}}}{\partial \vec{\boldsymbol{x}}} = \begin{pmatrix} a_1 \\ \vdots \\ a_P \end{pmatrix} = \vec{\boldsymbol{a}}$$

です。

二次形式もスカラーになりますから、同じように計算出来て、

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & \cdots & a_{1P} \\ \vdots & \ddots & \vdots \\ a_{P1} & \cdots & a_{PP} \end{pmatrix}$$

とすると、

$$\vec{\boldsymbol{x}}^{\boldsymbol{T}} \boldsymbol{A} \, \vec{\boldsymbol{x}} = \sum_{\boldsymbol{j=1}}^{P} \sum_{\boldsymbol{l=1}}^{P} a_{jl} \, x_j x_l$$

$$\frac{\partial \vec{\boldsymbol{x}}^{\boldsymbol{T}} \boldsymbol{A} \, \vec{\boldsymbol{x}}}{\partial \vec{\boldsymbol{x}}} = \begin{pmatrix} \frac{\partial \sum_{\boldsymbol{j=1}}^{P} \sum_{\boldsymbol{l=1}}^{P} a_{jl} x_j x_l}{\partial x_1} \\ \vdots \\ \frac{\partial \sum_{\boldsymbol{j=1}}^{P} \sum_{\boldsymbol{l=1}}^{P} a_{jl} \, x_j x_l}{\partial x_P} \end{pmatrix}$$

$$\frac{\partial a_{jl} x_j x_l}{\partial x_p} = a_{jp} x_j \; (when \; l = p), a_{jp} x_l \; (when \; l = p), 0 \; (when \; l \neq p)$$

だから

$$\frac{\partial \vec{x}^T A\,\vec{x}}{\partial \vec{x}} = \begin{pmatrix} \sum_{j=1}^{P} a_{j1}x_j + \sum_{l=1}^{P} a_{1l}x_l \\ \sum_{j=1}^{P} a_{j2}x_j + \sum_{l=1}^{P} a_{2l}x_l \\ \vdots \\ \sum_{j=1}^{P} a_{jP}x_j + \sum_{l=1}^{P} a_{Pl}x_l \end{pmatrix}$$

$j = l = p$と書き換えて

$$\frac{\partial \vec{x}^T A\,\vec{x}}{\partial \vec{x}} = \begin{pmatrix} \sum_{p=1}^{P} a_{p1}x_p + \sum_{p=1}^{P} a_{1p}x_p \\ \sum_{j=1}^{P} a_{j2}x_j + \sum_{l=1}^{P} a_{2l}x_p \\ \vdots \\ \sum_{j=1}^{P} a_{jP}x_j + \sum_{l=1}^{P} a_{Pl}x_p \end{pmatrix} = \begin{pmatrix} \sum_{p=1}^{P} (a_{p1}x_p + a_{1p}x_p) \\ \sum_{p=1}^{P} (a_{p2}x_p + a_{2p}x_p) \\ \vdots \\ \sum_{p=1}^{P} (a_{p2}x_p + a_{2p}x_p) \end{pmatrix} = \begin{pmatrix} \sum_{p=1}^{P} (a_{p1} + a_{1p})x_p \\ \sum_{p=1}^{P} (a_{p2} + a_{2p})x_p \\ \vdots \\ \sum_{p=1}^{P} (a_{p2} + a_{2p})x_p \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}+a_{11} & a_{21}+a_{12} & \cdots & a_{P1}+a_{1P} \\ a_{12}+a_{21} & a_{22}+a_{22} & \cdots & a_{P2}+a_{P2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1P}+a_{P1} & a_{2P}+a_{\mathrm{P1}} & \cdots & a_{PP}+a_{PP} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_P \end{pmatrix}$$

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & \cdots & a_{1P} \\ \vdots & \ddots & \vdots \\ a_{P1} & \cdots & a_{PP} \end{pmatrix}, \qquad \boldsymbol{A^T} = \begin{pmatrix} a_{11} & \cdots & a_{P1} \\ \vdots & \ddots & \vdots \\ a_{1P} & \cdots & a_{PP} \end{pmatrix}$$

$$\frac{\partial \vec{x}^T A\,\vec{x}}{\partial \vec{x}} = (\boldsymbol{A^T} + \boldsymbol{A})\,\vec{\boldsymbol{x}} = (\boldsymbol{A} + \boldsymbol{A^T})\,\vec{\boldsymbol{x}} \qquad \textit{式 86}$$

$\boldsymbol{A}$が対称行列であれば、$\boldsymbol{A^T} = \boldsymbol{A}$だから、

$$\frac{\partial \vec{x}^T A\,\vec{x}}{\partial \vec{x}} = \boldsymbol{2A\vec{x}} \qquad \textit{式 87}$$

要領がわかったところで、

$$\frac{\partial\left(-\frac{1}{2}(\boldsymbol{x_i} - \boldsymbol{\mu_k})^T {\Sigma_k}^{-1}(\boldsymbol{x_i} - \boldsymbol{\mu_k})\right)}{\partial \boldsymbol{\mu_k}} = \frac{-\frac{1}{2}\partial\left((\boldsymbol{x_i} - \boldsymbol{\mu_k})^T {\Sigma_k}^{-1}(\boldsymbol{x_i} - \boldsymbol{\mu_k})\right)}{\partial \boldsymbol{\mu_k}}$$

$\boldsymbol{x_i} - \boldsymbol{\mu_k} = \boldsymbol{X}$として

$$\frac{\partial \boldsymbol{X}}{\partial \boldsymbol{\mu_k}} = -1$$

$$\frac{\partial(\boldsymbol{X}^T {\Sigma_k}^{-1}\boldsymbol{X})}{\partial \boldsymbol{X}} = 2{\Sigma_k}^{-1}\boldsymbol{X} \quad \textit{（分散共分散行列は対称行列）}$$

$$\frac{-\frac{1}{2}\partial\left((\boldsymbol{x_i} - \boldsymbol{\mu_k})^T {\Sigma_k}^{-1}(\boldsymbol{x_i} - \boldsymbol{\mu_k})\right)}{\partial \boldsymbol{\mu_k}} = -\frac{1}{2}\frac{\partial(\boldsymbol{X}^T {\Sigma_k}^{-1}\boldsymbol{X})}{\partial \boldsymbol{X}}\frac{\partial \boldsymbol{X}}{\partial \boldsymbol{\mu_k}}$$

$$= -\frac{1}{2} \times 2{\Sigma_k}^{-1}\boldsymbol{X} \times (-\mathbf{1}) = {\Sigma_k}^{-1}\boldsymbol{X}$$

$$= {\Sigma_k}^{-1}(\boldsymbol{x_i} - \boldsymbol{\mu_k})$$

対数尤度の微分に戻ると

$$\frac{\partial LL}{\partial \boldsymbol{\mu_k}} = \sum_{i=1}^{n} \gamma_{ik}\, {\Sigma_k}^{-1}(\boldsymbol{x_i} - \boldsymbol{\mu_k})$$

となります。

$$\frac{\partial LL}{\partial \boldsymbol{\mu_k}} = 0$$

を解きます。

$$\frac{\partial LL}{\partial \boldsymbol{\mu_k}} = \sum_{i=1}^{n} \gamma_{ik}\, {\Sigma_k}^{-1}(\boldsymbol{x_i} - \boldsymbol{\mu_k}) = {\Sigma_k}^{-1}\sum_{i=1}^{n} \gamma_{ik}\,(\boldsymbol{x_i} - \boldsymbol{\mu_k}) = {\Sigma_k}^{-1}\left(\sum_{i=1}^{n} \gamma_{ik}\boldsymbol{x_i} - \sum_{i=1}^{n} \gamma_{ik}\boldsymbol{\mu_k}\right)$$

$$= \sum_{i=1}^{n} \gamma_{ik}\boldsymbol{x_i} - \boldsymbol{\mu_k}\sum_{i=1}^{n} \gamma_{ik} = 0$$

$$\boldsymbol{\mu_k}\sum_{i=1}^{n} \gamma_{ik} = \sum_{i=1}^{n} \gamma_{ik}\boldsymbol{x_i}$$

$$\boldsymbol{\mu_k} = \frac{\sum_{i=1}^{n} \gamma_{ik}\boldsymbol{x_i}}{\sum_{i=1}^{n} \gamma_{ik}}$$

分母はクラスkに属する確率の総和で、クラスkの属するデータの数に匹敵するので、これを

$$\sum_{i=1}^{n} \gamma_{ik} = N_k$$

とすると、

$$\boldsymbol{\mu_k} = \frac{\sum_{i=1}^{n} \gamma_{ik}\boldsymbol{x_i}}{N_k}$$

$\boldsymbol{\mu_k}$貢献度で重みを付けた平均（重心）となります。。

次に$\frac{\partial LL}{\partial \boldsymbol{\Sigma_k}}$を考えます。

$$\frac{\partial(\log_e \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k}, \Sigma_k))}{\partial \Sigma_k} = -\frac{P}{2}\frac{\partial \log_e(2\pi)}{\partial \Sigma_k} - \frac{1}{2}\frac{\partial \log_e|\Sigma_k|}{\partial \Sigma_k} - \frac{1}{2}\frac{\partial\{(\boldsymbol{x_i} - \boldsymbol{\mu_k})^T {\Sigma_k}^{-1}(\boldsymbol{x_i} - \boldsymbol{\mu_k})\}}{\partial \Sigma_k}$$

黄色の部分はΣ_kの関数ではないので、Σ_kで微分すれば 0 です。青の部分はΣ_kの関数ですから、微分を考えなくてはなりません。

$\frac{\partial \log_e|\Sigma_k|}{\partial \Sigma_k}$は対数行列式を行列で微分するという問題で、$\frac{\partial\{(\boldsymbol{x_i} - \boldsymbol{\mu_k})^T {\Sigma_k}^{-1}(\boldsymbol{x_i} - \boldsymbol{\mu_k})\}}{\partial \Sigma_k}$は逆行列でつながった二次形式を行列で微分するという問題です。どちらも難しくありませんが、やった経験がないと面倒です。

対数行列式の行列での微分から考えます。まず行列式を行列で微分してみます。行列式はスカラーですから、スカラーを行列で微分する形で、

$$\frac{d|\boldsymbol{A}|}{d\boldsymbol{A}}=\begin{pmatrix}\frac{d|\boldsymbol{A}|}{da_{11}} & \frac{d|\boldsymbol{A}|}{da_{12}} & \cdots & \frac{d|\boldsymbol{A}|}{da_{1P}}\\ \frac{d|\boldsymbol{A}|}{da_{21}} & \frac{d|\boldsymbol{A}|}{da_{22}} & \cdots & \frac{d|\boldsymbol{A}|}{da_{2P}}\\ \vdots & \vdots & \ddots & \vdots\\ \frac{d|\boldsymbol{A}|}{da_{P1}} & \frac{d|\boldsymbol{A}|}{da_{P1}} & \cdots & \frac{d|\boldsymbol{A}|}{da_{PP}}\end{pmatrix}$$

という形になります。一般化して書くと長くなって面倒なので、３X３の行列を例にして転置行列、余因子行列、行列式、逆行列を確認しておきます。ます。

$$\boldsymbol{A}=\begin{pmatrix}a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ a_{31} & a_{32} & a_{33}\end{pmatrix}$$

$$\boldsymbol{A^T}=\begin{pmatrix}a_{11} & a_{21} & a_{31}\\ a_{12} & a_{22} & a_{32}\\ a_{13} & a_{23} & a_{33}\end{pmatrix}$$

$$\widetilde{\boldsymbol{A}}=\begin{pmatrix}a_{22}a_{33}-a_{32}a_{23} & -a_{12}a_{33}+a_{32}a_{13} & a_{12}a_{23}-a_{22}a_{13}\\ -a_{21}a_{33}+a_{31}a_{23} & a_{11}a_{33}-a_{31}a_{13} & -a_{11}a_{23}+a_{21}a_{13}\\ a_{21}a_{32}-a_{31}a_{22} & -a_{11}a_{32}+a_{31}a_{12} & a_{11}a_{22}-a_{21}a_{22}\end{pmatrix}$$

$$|\boldsymbol{A}|=a_{11}a_{22}a_{33}+a_{12}a_{23}a_{31}+a_{13}a_{21}a_{32}-a_{11}a_{23}a_{32}-a_{12}a_{21}a_{33}-a_{13}a_{22}a_{31}$$

$$\boldsymbol{A^{-1}}=\frac{\widetilde{\boldsymbol{A}}}{|\boldsymbol{A}|}$$

この例で、たとえば、

$$\frac{d|\boldsymbol{A}|}{da_{11}}=\frac{d(a_{11}a_{22}a_{33}+a_{12}a_{23}a_{31}+a_{13}a_{21}a_{32}-a_{11}a_{23}a_{32}-a_{12}a_{21}a_{33}-a_{13}a_{22}a_{31})}{da_{11}}=a_{22}a_{33}-a_{32}a_{23}$$

$$\frac{d|\boldsymbol{A}|}{da_{12}}=\frac{d(a_{11}a_{22}a_{33}+a_{12}a_{23}a_{31}+a_{13}a_{21}a_{32}-a_{11}a_{23}a_{32}-a_{12}a_{21}a_{33}-a_{13}a_{22}a_{31})}{da_{12}}=-a_{21}a_{33}+a_{23}a_{31}$$

$$\frac{d|\boldsymbol{A}|}{da_{21}}=\frac{d(a_{11}a_{22}a_{33}+a_{12}a_{23}a_{31}+a_{13}a_{21}a_{32}-a_{11}a_{23}a_{32}-a_{12}a_{21}a_{33}-a_{13}a_{22}a_{31})}{da_{21}}=-a_{12}a_{33}+a_{32}a_{13}$$

となります。つまり、

$$\frac{d|\boldsymbol{A}|}{d\boldsymbol{A}}=\begin{pmatrix}a_{22}a_{33}-a_{32}a_{23} & -a_{21}a_{33}+a_{31}a_{23} & a_{21}a_{32}-a_{31}a_{22}\\ -a_{12}a_{33}+a_{32}a_{13} & a_{11}a_{33}-a_{31}a_{13} & -a_{11}a_{32}+a_{31}a_{12}\\ a_{12}a_{23}-a_{22}a_{13} & -a_{11}a_{23}+a_{21}a_{13} & a_{11}a_{22}-a_{21}a_{22}\end{pmatrix}=\widetilde{\boldsymbol{A}}^{\boldsymbol{T}}$$

余因子行列の転置行列になります。余因子行列を行列式で割れば逆行列ですから、逆行列で表すと、

$$\frac{d|\boldsymbol{A}|}{d\boldsymbol{A}}=|\boldsymbol{A}|\frac{\widetilde{\boldsymbol{A}}^{\boldsymbol{T}}}{|\boldsymbol{A}|}=|\boldsymbol{A}|(\boldsymbol{A^{-1}})^{\boldsymbol{T}} \qquad \textit{式 88}$$

となります。対数行列式の場合は、

$$\frac{d\,log_e|\boldsymbol{A}|}{d\boldsymbol{A}}=\frac{d\,log_e|\boldsymbol{A}|}{d|\boldsymbol{A}|}\frac{d|\boldsymbol{A}|}{d\boldsymbol{A}}=\frac{1}{|\boldsymbol{A}|}|\boldsymbol{A}|(\boldsymbol{A^{-1}})^{\boldsymbol{T}}=(\boldsymbol{A^{-1}})^{\boldsymbol{T}} \qquad \textit{式 89}$$

となります。*A*が対称行列ならば、

$$\frac{d\,log_e|\boldsymbol{A}|}{d\boldsymbol{A}} = \boldsymbol{A^{-1}} \qquad \textit{式 90}$$

結論として、対称行列ならば、対数行列式の行列による微分は逆行列が返ってきます。

$\frac{\partial\{(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T\boldsymbol{\Sigma_k}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu_k})\}}{\partial\boldsymbol{\Sigma_k}}$のように逆行列でつながった二次形式を行列で微分する公式を作ります。

$$A^{-1}A = \boldsymbol{I}$$

この両辺をスカラーで微分します。

$$\frac{d(A^{-1}A)}{dx} = \frac{d\boldsymbol{I}}{dx}$$

左辺は関数の積の微分で右辺の微分は$\boldsymbol{O}$

$$\frac{dA^{-1}}{dx}A + A^{-1}\frac{dA}{dx} = \boldsymbol{O}$$

$$\frac{dA^{-1}}{dx}A = -A^{-1}\frac{dA}{dx}$$

両辺に右からA^{-1}をかけます。

$$\frac{dA^{-1}}{dx}AA^{-1} = -A^{-1}\frac{dA}{dx}A^{-1} = \boldsymbol{O}$$

$$\frac{dA^{-1}}{dx} = -A^{-1}\frac{dA}{dx}A^{-1}$$

$x = a_{ij}$として逆行列を行列の要素で微分する式に書き換えると

$$\frac{dA^{-1}}{da_{ij}} = -A^{-1}\frac{dA}{da_{ij}}A^{-1}$$

これを二次形式にあてはめると、

$$\frac{d\vec{x}^TA^{-1}\vec{x}}{da_{ij}} = \vec{x}^T\left(-A^{-1}\frac{dA}{da_{ij}}A^{-1}\right)\vec{x} = -\vec{x}^TA^{-1}\frac{dA}{da_{ij}}A^{-1}\vec{x} = -((A^{-1})^T\vec{x})^T\frac{dA}{da_{ij}}A^{-1}\vec{x}$$

$$(A^{-1})^T\vec{x} = \vec{a}, \qquad A^{-1}\vec{x} = \vec{b}$$

とおく、行列で微分すると

$$\frac{d\vec{x}^TA^{-1}\vec{x}}{dA} = \frac{-d\vec{a}^TA^{-1}\vec{b}}{dA} = -\vec{a}^T\frac{dA}{dA}\vec{b} = -\vec{a}\vec{b}^T = -(A^{-1})^T\vec{x}\vec{x}^T(A^{-1})^T \qquad \textit{式 91}$$

Aが対称行列ならば

$$\frac{d\vec{x}^TA^{-1}\vec{x}}{dA} = -A^{-1}\vec{x}\vec{x}^TA^{-1} \qquad \textit{式 92}$$

という公式が得られます。これを使って

$$\frac{\partial\{(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T{\Sigma_k}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu_k})\}}{\partial\Sigma_k}=-{\Sigma_k}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu_k})(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T{\Sigma_k}^{-1}$$

さらに、式 90 を使って

$$\frac{\partial(\log_e \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k},\Sigma_k))}{\partial\Sigma_k}=-\frac{P}{2}\frac{\partial\log_e(2\pi)}{\partial\Sigma_k}-\frac{1}{2}\frac{\partial\log_e|\Sigma_k|}{\partial\Sigma_k}-\frac{1}{2}\frac{\partial\{(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T{\Sigma_k}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu_k})\}}{\partial\Sigma_k}$$

$$=-\frac{1}{2}{\Sigma_k}^{-1}+\frac{1}{2}{\Sigma_k}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu_k})(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T{\Sigma_k}^{-1}$$

$\frac{\partial LL}{\partial\Sigma_k}$にもどると

$$\frac{\partial LL}{\partial\Sigma_k}=\sum_{i=1}^{n}\gamma_{ik}\frac{\partial(\log_e \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k},\Sigma_k))}{\partial\Sigma_k}$$

$$=\sum_{i=1}^{n}\gamma_{ik}\left(-\frac{1}{2}{\Sigma_k}^{-1}+\frac{1}{2}{\Sigma_k}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu_k})(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T{\Sigma_k}^{-1}\right)$$

$$=\frac{1}{2}{\Sigma_k}^{-1}\sum_{i=1}^{n}\gamma_{ik}\left((\boldsymbol{x_i}-\boldsymbol{\mu_k})(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T{\Sigma_k}^{-1}-\boldsymbol{I}\right)$$

$$=\frac{1}{2}{\Sigma_k}^{-1}\left({\Sigma_k}^{-1}\sum_{i=1}^{n}\gamma_{ik}\left((\boldsymbol{x_i}-\boldsymbol{\mu_k})(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T\right)-\sum_{i=1}^{n}\gamma_{ik}\right)=0$$

$${\Sigma_k}^{-1}\sum_{i=1}^{n}\gamma_{ik}\left((\boldsymbol{x_i}-\boldsymbol{\mu_k})(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T\right)-\sum_{i=1}^{n}\gamma_{ik}=0$$

左からΣ_kをかけて、

$$\sum_{i=1}^{n}\gamma_{ik}\left((\boldsymbol{x_i}-\boldsymbol{\mu_k})(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T\right)-\Sigma_k\sum_{i=1}^{n}\gamma_{ik}=0$$

$$\Sigma_k\sum_{i=1}^{n}\gamma_{ik}=\sum_{i=1}^{n}\gamma_{ik}\left((\boldsymbol{x_i}-\boldsymbol{\mu_k})(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T\right)$$

$$\Sigma_k=\frac{\sum_{i=1}^{n}\gamma_{ik}\left((\boldsymbol{x_i}-\boldsymbol{\mu_k})(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T\right)}{\sum_{i=1}^{n}\gamma_{ik}}$$

$$\sum_{i=1}^{n}\gamma_{ik}=N_k$$

とすると、

$$\Sigma_k=\frac{\sum_{i=1}^{n}\gamma_{ik}\left((\boldsymbol{x_i}-\boldsymbol{\mu_k})(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T\right)}{N_k}$$

スッキリとまとまりましたが、これ何というのか知りません。重み付きの分散共分散行列とでもいうのでしょうか。

最後にπ_kの最適化を考えます。π_kは 2 つの点で$\boldsymbol{\mu_k}$やΣ_kと異なります。一つは、正規分布を決める係数ではないということです。もう一つは$\sum_{k=1}^{K}\pi_k$=1 という制約条件が付いているということです。制約条件付きの極値問題は、ラグランジェの未定乗数法で解きます。ラグラ

ンジェの未定乗数法は V-2-6.最大・最小のところで説明し、VI-1-4.因子分析のところでもう一度説明しましたので、ここではラグランジェの未定乗数法の使い方だけを説明しておきます。ラグランジェの未定乗数法は、$g(x,y)=0$という制約条件の下で、$f(x,y)$を最大化（あるいは最小化）する極値問題の解法です。以下のようにラグランジェ関数(LG)を作ってラグランジェ関数の極値問題として解きます。

$$LG = f(x,y) - \lambda g(x,y)$$

$$\frac{\partial LG}{\partial x} = 0$$

$$\frac{\partial LG}{\partial y} = 0$$

$$\frac{\partial LG}{\partial \lambda} = 0$$

この場合、

$$f(\pi_k) = LL = \sum_{i=1}^{n} \log \sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} \middle| \boldsymbol{\mu_j}, \Sigma_j\right)$$

$$g(x,y) = \sum_{k=1}^{K} \pi_k - 1$$

なので、

$$LG = \sum_{i=1}^{n} \log \sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} \middle| \boldsymbol{\mu_j}, \Sigma_j\right) - \lambda\left(\sum_{k=1}^{K} \pi_k - 1\right)$$

$$\frac{\partial LG}{\partial \pi_k} = \frac{\partial\left(\sum_{i=1}^{n} log \sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} \middle| \boldsymbol{\mu_j}, \Sigma_j\right)\right)}{\partial \pi_k} - \frac{\partial \lambda(\sum_{k=1}^{K} \pi_k - 1)}{\partial \pi_k}$$

となって、一項目と二項目に分けて微分できます。二項目の

$$\frac{\partial \lambda(\sum_{k=1}^{K} \pi_k - 1)}{\partial \pi_j} = \lambda$$

はすぐわかります。一項目については、いままでと同じように、対数の微分公式を使って

$$\frac{\partial\left(\sum_{i=1}^{n} \log \sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} \middle| \boldsymbol{\mu_j}, \Sigma_j\right)\right)}{\partial \pi_j} = \sum_{i=1}^{n} \frac{\partial\left(\log \sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} \middle| \boldsymbol{\mu_j}, \Sigma_j\right)\right)}{\partial \pi_j}$$

$$= \sum_{i=1}^{n} \frac{1}{\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} \middle| \boldsymbol{\mu_j}, \Sigma_j\right)} \frac{\partial\left(\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} \middle| \boldsymbol{\mu_j}, \Sigma_j\right)\right)}{\partial \pi_j}$$

ここまでは、$\boldsymbol{\mu_j}, \Sigma_j$とおなじなのですが、

$$\frac{\partial\left(\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x_i} \middle| \boldsymbol{\mu_j}, \Sigma_j\right)\right)}{\partial \pi_k} = \mathcal{N}(\boldsymbol{x_k} | \boldsymbol{\mu_k}, \Sigma_k)$$

ですから、

$$\frac{\partial\left(\sum_{i=1}^{n}\log\sum_{j=1}^{K}\pi_j\,\mathcal{N}\left(\boldsymbol{x_i}\middle|\boldsymbol{\mu_j},\Sigma_j\right)\right)}{\partial\pi_j}=\sum_{i=1}^{n}\frac{1}{\sum_{j=1}^{K}\pi_j\,\mathcal{N}\left(\boldsymbol{x_i}\middle|\boldsymbol{\mu_j},\Sigma_j\right)}\mathcal{N}(\boldsymbol{x_k}|\boldsymbol{\mu_k},\Sigma_k)$$

γ_{ik}を作りたいので、

$$=\frac{1}{\pi_k}\sum_{i=1}^{n}\frac{\pi_k\mathcal{N}(\boldsymbol{x_k}|\boldsymbol{\mu_k},\Sigma_k)}{\sum_{j=1}^{K}\pi_j\,\mathcal{N}\left(\boldsymbol{x_i}\middle|\boldsymbol{\mu_j},\Sigma_j\right)}$$

$$=\frac{1}{\pi_k}\sum_{i=1}^{n}\gamma_{ik}$$

二つの微分を合わせて、

$$\frac{\partial LG}{\partial\pi_k}=\frac{1}{\pi_k}\sum_{i=1}^{n}\gamma_{ik}-\lambda=0$$

$$\frac{1}{\pi_k}\sum_{i=1}^{n}\gamma_{ik}=\lambda$$

$\sum_{i=1}^{n}\gamma_{ik}=N_k$として、

$$\frac{1}{\pi_k}N_k=\lambda$$

$$N_k=\pi_k\lambda$$

ところで、

$$n=\sum_{k=1}^{K}N_k=\sum_{k=1}^{K}\pi_k\lambda=\lambda\sum_{k=1}^{K}\pi_k=\lambda$$

$$\because\sum_{k=1}^{K}\pi_k=1$$

$$\lambda=n$$

$$\pi_k=\frac{N_k}{n}$$

以上で、外生的に$\boldsymbol{\gamma_i}$が与えて、$\boldsymbol{x}$ によって$\boldsymbol{\pi,\mu,\Sigma}$を最適化し、$\boldsymbol{\pi,\mu,\Sigma}$から期待値として$\boldsymbol{\gamma_i}$を更新する式が出来ました。

これらを使って、EM アルゴリズムを作ります。

E ステップ（期待値として$\boldsymbol{\pi,\mu,\Sigma}$から$\boldsymbol{\gamma_i}$を求める。）

$$\gamma_{ik}=P(z_{ik}=1|\boldsymbol{x_i})=\frac{\pi_k N(\boldsymbol{x_i}|\boldsymbol{\mu_k},\Sigma_k)}{\sum_{j=1}^{K}\pi_j\mathcal{N}\left(\boldsymbol{x_i}\middle|\boldsymbol{\mu_j},\Sigma_j\right)}$$

M ステップ（外生的に与えられたγ_{ik}を使って、$\boldsymbol{\pi,\mu,\Sigma}$を最適化する。

$$\sum_{i=1}^{n}\gamma_{ik}=N_k$$

$$\sum_{i=1}^{n}N_k=N$$

$$\boldsymbol{\mu_k} = \frac{\sum_{i=1}^{n} \gamma_{ik} \boldsymbol{x_i}}{N_k}$$

$$\Sigma_k = \frac{\sum_{i=1}^{n} \gamma_{ik} \left((\boldsymbol{x_i} - \boldsymbol{\mu_k})(\boldsymbol{x_i} - \boldsymbol{\mu_k})^T \right)}{N_k}$$

$$\pi_k = \frac{N_k}{N}$$

これを繰り返すことになります。どのような条件で収束とするかはいろいろと考えようがありますが、とりあえず繰り返しごとに対数尤度を計算します。

$$LL = \sum_{i=1}^{n} \log \sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x_i} | \boldsymbol{\mu_j}, \Sigma_j)$$

更新ごとに、

$$LL_{after} - LL_{befre} \leq \varepsilon$$

として、εを決めておいて、一定以下の差になったら終了というやり方が考えられます。もちろん、これでいつでも妥当な解にたどり着くかどうかはわかりません。部分最適に陥って計算が止まる可能性はあります。それを防ぐには、あらかじめ、部分的に抽出したサンプルで階層的クラスター分析をしておくとか、初期値をいくつか変えて与えて安定した答えが得られることを確かめるなどというやり方があります。

EM アルゴリズムの説明、微分式の変形、行列による微分の公式化など、一つ一つ丁寧にやったので大変長い解説になりましたが、これで終わりではありません。最後のひと仕事があります。私たちが求めていたのは、あるデータが得られた時に、そのデータがクラス k に属することの確からしさ（尤度）でした。ベイズの式（式 iv）で書くと以下の確率です。

$$P(z_{ik} = 1 | \boldsymbol{x_i}) = \frac{P(\boldsymbol{x_i} | z_{ik} = 1) P(z_{ik} = 1)}{P(\boldsymbol{x_i})} \qquad \text{iv}$$

左片は$\boldsymbol{x_i}$**というデータがあるときにそれがクラス**kに属する確率の意味です。右辺の黄色の部分が負担率$\boldsymbol{\gamma_{ik}}$、青の部分は多変量確率密度分布（ガウス関数）

$$P(\boldsymbol{x_i} | z_{ik} = 1) = \mathcal{N}(\boldsymbol{x_i} | \boldsymbol{\mu}_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^P |\Sigma_k|}} e^{\left\{ -\frac{1}{2} (x_i - \mu_k)^T \Sigma_k^{-1} (x_i - \mu_k) \right\}}$$

です。$\boldsymbol{x_i}$を与えて

$$P(z_{ik} = 1 | \boldsymbol{x_i}) = \boldsymbol{\gamma_{ik}} \mathcal{N}(\boldsymbol{x_i} | \boldsymbol{\mu}_k, \Sigma_k)$$

を計算すれば、各クラスについて尤度を比較できます。また、$\boldsymbol{P}_{nk}$を N 行 K 列の行列、その要素を

$$P(z_{nk} = 1 | \boldsymbol{x_n}) = \boldsymbol{\gamma_{nk}} \mathcal{N}(\boldsymbol{x_n} | \boldsymbol{\mu}_k, \Sigma_k)$$

として、混合ガウスモデルを作った元データについて行列を完成させれば、クラスタリングの妥当性を検討できます。

VII-3-4-2.　混合ガウスモデルのプログラム（Python）

Python のライブラリーscikit learn (sklearn)には、mixture という混合ガウスモデルで非階層的クラスター分析をするプログラムが用意されています。これを使えば、簡単に混合ガウスモデルを実行することが可能です。VII-2-4-4 で mixture を使ったプログラムを実装しますが、その前に、混合ガウスモデルがどんな計算をしていて、どこの問題があるのかを理解するために、mixture を使わないプログラムを作って実行してみます（「やさしい水産学（自習室)」-「参考資料」-「python」-「VII-3-4-1. 混合ガウスモデル(scikit.learn.mixuture を使わない)」参照)。プログラムを作るにあたって、途中でいくつか動作確認をしましたが、それをそのまま残しておきました。動作確認することによって、プログラムのどの部分で何をしているのかがわかり、Python のプログラミングの技術も向上します。数学的な解説の内容と照らし合わせながら動作確認して機能を実感してください。

VII-3-4-1-i データの読み込み、主成分分析、（VII-3-2-i.と同じ）

```
#[A]必要なライブラリーの読み込み
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from scipy.cluster.hierarchy import linkage,dendrogram,fcluster
import decimal
decimal.getcontext().prec=6
#[B]データの読み込み
df =pd.read_csv("sample10.csv")
xn,D=df.shape
D1=D-1
#データフレームをつくる
dfX=pd.DataFrame(df)
X=df.values
X=np.delete(X,0,1)
#列名を付ける
for i in range (D):
    dfX=dfX.rename(columns={i:"X"+str(i+1)})
print (dfX)
D1=D-1
#[C]標準化後主成分分析を実行
import urllib.request
import matplotlib.pyplot as plt
import sklearn    #機械学習のライブラリ
from sklearn.decomposition import PCA   #主成分分析
from sklearn.preprocessing import StandardScaler #標準化
from IPython.display import display
#標準化
std_sc = StandardScaler()
std_sc.fit(X)
std_data = std_sc.transform(X)
std_data_df = pd.DataFrame(std_data)
display(std_data_df)
```

```
#主成分分析の実行
pca = PCA()
pca.fit(std_data_df)
# データを主成分空間に写像
pca_cor = pca.transform(std_data_df)
print(pca.get_covariance()) # 分散共分散行列
# 固有ベクトルのマトリックス表示
eig_vec = pd.DataFrame(pca.components_.T, ¥
                          columns = ["PC{}".format(x + 1) for x in range(len(std_data_df.columns))])
display(eig_vec)
# 固有値
eig = pd.DataFrame(pca.explained_variance_, index=["PC{}".format(x + 1) for x in
range(len(std_data_df.columns))], columns=['固有値']).T
display(eig)
# Rによるソースコードだと、固有値（分散）ではなく標準偏差を求めている。
# 主成分の標準偏差
dv = np.sqrt(eig)
dv = dv.rename(index = {'固有値':'主成分の標準偏差'})
display(dv)
# 寄与率
ev = pd.DataFrame(pca.explained_variance_ratio_, index=["PC{}".format(x + 1) for x in
range(len(std_data_df.columns))], columns=['寄与率']).T
display(ev)
# 累積寄与率
t_ev = pd.DataFrame(pca.explained_variance_ratio_.cumsum(), index=["PC{}".format(x + 1) for x in
range(len(std_data_df.columns))], columns=['累積寄与率']).T
display(t_ev)
# 主成分得点
print('主成分得点')
cor = pd.DataFrame(pca_cor, columns=["PC{}".format(x + 1) for x in range(len(std_data_df.columns))])
display(cor)
PCC=cor.values
dfS=pd.concat([dfX,cor],axis=1)
S=dfS.values
```

この部分の内容は、階層的クラスター分析で使ったデータの読み込みと同じですが、プログラムをつなげるために、一部の変数名を変えています（PC→PCC)。階層的クラスター分析とは関係ありませんが、主成分分析をしています。主成分分析をする目的は、データの圧縮で計算の負担を軽減するためです。データー項目間に強い相関がある場合、直交する主要な主成分に取りまとめることで、不必要な情報を切り捨てることが出来ます。

VII-3-4-1-ii. 主成分数の決定

```
#主成分の数を決定する
P=2 #主成分の数を入力
N,D=PCC.shape
PC=np.zeros((N,P))
for n in range(N):
    for p in range (P):
        PC[n,p]=PCC[n,p]
```

主成分分析の結果（累積寄与率）をもとに主成分の数を決定します（P=2)。主成分ではなくて、元のデータでクラスター分析をするのであれば、この部分に入力する必要はありません。

VII-3-4-1-iii.分析するデータを決定し、初期値を与える。

```
X=X
N,D=X.shape
X_range0=[-2,2]
X_range1=[-2,2]
#マーカーの色決定
x_col=np.array([[0,0,0.95],[0.95,0,0],[0,0.95,0],[0.95,0.95,0],[1,1,1],[0,0.95,0.95],[0,0,0],[0.95,0,0.
95]])
#初期条件
Pi0=np.array([0.2,0.2,0.2,0.2,0.2])
Pi=Pi0
Mu0=np.array([[1,1],[-1,1],[-1,-1],[1,-1],[0,0]])
Mu=Mu0
Sigma0=np.array([[[1,0],[0,1]],[[1,0],[0,1]],[[1,0],[0,1]],[[1,0],[0,1]],[[1,0],[0,1]]])
Sigma=Sigma0
N=X.shape[0]
K=len(Pi)
Gamma0=np.c_[np.ones((N,1)),np.zeros((N,K-1))]
Gamma=Gamma0
```

データをそのままクラスター分析するのであれば、X=X、主成分得点でクラスター分析するのであれば、X=PC を選択します。X_range0=[-3,3]、X_range1=[-3,3]は、散布図や等高線図を描く際の x 軸 y 軸の描写範囲の指定です。マーカーの色は、散布図や重心の所属クラスを明示すための色で、[0,0,1]：青、[1,0,0]:赤、[0,1,0]:緑、[1,1,0]:黄、[1,1,1]:白、[0,1,1]:シアン、[1,0,1]:マゼンタを指定しています。プログラウの途中で色を混ぜ合わせますがその際、色指定の値が 1 を超えると計算が止まるので、白以外の色は 1 ではなくて 0,95 と指定しています。もっと多くの色が必要な場合は、光の三原色の混合比を変えて新しい色を作ってください。以下は初期値の入力です。この例では 5 つのクラスが、ほぼ等しい割合で存在すると考えて混合係数は[0.2,0.2,0.2,0.2,0.2]としました。階層的クラスター分析や K-means 法の結果を参考に、各クラスの重心は[1,1],[-1,1],[-1,-1],[1,-1],[0,0]としました。分散共分散行列の指定の仕方ですが、また、初期を取っておきたいので、初期値は Pi0、Mu0、Sigma0、Gamma0 で指定して、Pi=Pi0 Mu=Mu0、Sigma=Sigma0、Gamma=Gamma0 として、実際の計算は Pi、Mu、Sigma、Gamma を更新していきます

Sigma0 は分散共分散行列の指定ですが、行列の指定は*行列*$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$を

[[*a,b,c*],[*d,e,f*],[*g,h,i*]]という形で指定します。ここでは、すべてのクラスについて、項目間の相関がなく、縦軸方向、横軸方向の分散がともに 1 となるという初期値を与えています。負担率(Gamma)は暫定的・形式的に、すべてのデータについて、クラス 1 だけが 1 で他がすべて 0 ということにしておきます。

VII-3-4-1-iv.関数の定義

```
#関数の定義
#ガウス関数の定義
def gauss(x,mu,sigma):
    N,D=x.shape
    c1=-(D/2)*np.log(2*np.pi)
```

```
    det_sigma=np.linalg.det(sigma)
    c2=-(1/2)*np.log(det_sigma)
    inv_sigma=np.linalg.inv(sigma)
    c3=x-mu
    c4=np.dot(c3,inv_sigma)
    c5=np.zeros(N)
    for d in range(D):
        c5=c5+c4[:,d]*c3[:,d]
    c5=-c5/2
    p=c1+c2+c5
    p=np.exp(p)
    return p
#混合ガウスモデル
def mixgauss(x,pi,mu,sigma):
    N,D=x.shape
    K=len(pi)
    p=np.zeros(N)
    for k in range(K):
        p=p+pi[k]*gauss(x,mu[k,:],sigma[k,:,:])
    return p
#e step (gammaの更新)
def e_step_mixgauss(x,pi,mu,sigma):
    N,D=x.shape
    K=len(pi)
    y=np.zeros((N,K))
    for k in range(K):
        y[:,k]=gauss(x,mu[k,:],sigma[k,:,:])#クラスｋの分布でXが得られる確率（分子）
    gamma=np.zeros((N,K))
    for n in range(N):
        wk=np.zeros(K)
        for k in range(K):
            wk[k]=pi[k]*y[n,k]
        gamma[n,:]=wk/np.sum(wk)
    return gamma
#m step (Pi,Mu,Sigmaの更新)
def m_step_mixgauss(x,gamma):
    N,D=x.shape
    N,K=gamma.shape
    #piを計算
    pi=np.sum(gamma,axis=0)/N
    #muを計算
    mu=np.zeros((K,D))
    for k in range(K):
        for d in range(D):
            mu[k,d]=np.dot(gamma[:,k],x[:,d])/np.sum(gamma[:,k])
    #sigmaを計算
    sigma=np.zeros((K,D,D))
    for k in range(K):
        for n in range(N):
            wk=x-mu[k,:]
            wk=wk[n,:,np.newaxis]
            sigma[k,:,:]=sigma[k,:,:]+gamma[n,k]*np.dot(wk,wk.T)
        sigma[k,:,:]=sigma[k,:,:]/np.sum(gamma[:,k])
    return pi,mu,sigma
```

```
#EMアルゴリズム
def em_alg(max_it,err,pi,mu,sigma):
    it=0
    for it in range(0,max_it):
        gamma=e_step_mixgauss(X,pi,mu,sigma)
        err[it]=nlh_mixgauss(X,pi,mu,sigma)
        pi,mu,sigma1=m_step_mixgauss(X,gamma)
    return err,pi,mu,sigma,gamma
#誤差関数の定義
def nlh_mixgauss(x,pi,mu,sigma):
    N,D=x.shape
    K=len(pi)
    y=np.zeros((N,K))
    for k in range(K):
        y[:,k]=gauss(x,mu[k,:],sigma[k,:,:])
    lh=0
    for n in range(N):
        wk=0
        for k in range(K):
            wk=wk+pi[k]*y[n,k]
        lh=lh+np.log(wk)
    return -lh
#尤度・確率計算
#個々入力データの尤度
def likelihood(xx,mu,pi,sigma):
    N,D=xx.shape
    ppk1=np.zeros((N,K))
    ppl1=np.zeros((N,K))
    g1=np.zeros((N,K))
    S1=np.zeros((N))
    for k in range(K):
        g1[:,k]=gauss(xx,mu[k,:],sigma[k,:,:] )
        ppk1[:,k]=pi[k]*g1[:,k]
    for n in range(N):
        for k in range(K):
            S1[n]=S1[n]+ppk1[n,k]
        for k in range(K):
            ppl1[n,k]=g1[n,k]*ppk1[n,k]/S1[n]
    ratio1=np.zeros((N,K))
    for n in range(N):
        SS1=0
        for k in range(K):
            SS1=SS1+ppl1[n,k]
        for k in range(K):
            ratio1[n,k]=ppl1[n,k]/SS1
    return g1,ppk1,ppl1,ratio1

#データの図示
#混合ガウス等高線表示
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
%matplotlib inline
def show_contour_mixgauss(pi,mu,sigma):
    xn=40 #解像度
```

```
    x0=np.linspace(X_range0[0],X_range0[1],xn)
    x1=np.linspace(X_range1[0],X_range1[1],xn)
    xx0,xx1=np.meshgrid(x0,x1)
    A=xx0.reshape(xn*xn,1)
    B=xx1.reshape(xn*xn,1)
    x=np.c_[B,A]
    f=mixgauss(x,pi,mu,sigma)
    f=f.reshape(xn,xn)
    f=f.T
    plt.contour(x0,x1,f,10,color="grey")
#混合ガウス 3D 表示
def show3d_mixgauss(ax,pi,mu,sigma):
    xn=40 #解像度
    x0=np.linspace(X_range0[0],X_range0[1],xn)
    x1=np.linspace(X_range1[0],X_range1[1],xn)
    xx0,xx1=np.meshgrid(x0,x1)
    A=xx0.reshape(xn*xn,1)
    B=xx1.reshape(xn*xn,1)
    x= np.c_[B,A]
    f=mixgauss(x,pi,mu,sigma)
    f=f.reshape(xn,xn)
    f=f.T
    ax.plot_surface(xx0,xx1,f,rstride=2,cstride=2,alpha=0.3,color='blue',edgecolor='black')
#データ色分け等高線付き
def show_mixgauss_prm(x,gamma,pi,mu,sigma):
    show_contour_mixgauss(pi,mu,sigma)
    for n in range(N):
        col=np.zeros(3)
        for k in range(K):
            col=col+gamma[n,k]*x_col[k]
        plt.plot(x[n,0],x[n,1],'o',color=tuple(col),markeredgecolor='black',markersize=6,alpha=1)
    for k in range(K):

plt.plot(mu[k,0],mu[k,1],marker='*',markerfacecolor=tuple(x_col[k]),markersize=15,markeredgecolor='k',m
arkeredgewidth=1)
    plt.grid(True)
#データ色分け等高なし、軸を選択
def show_mixgauss_prm2(x,gamma,pi,mu,sigma):
    for n in range(N):
        col=np.zeros(3)
        for k in range(K):
            col=col+gamma[n,k]*x_col[k,:]
        plt.plot(x[n,d0],x[n,d1],'o',color=tuple(col),markeredgecolor='black',markersize=6,alpha=1)
    for k in range(K):

plt.plot(mu[k,d0],mu[k,d1],marker='*',markerfacecolor=tuple(x_col[k]),markersize=15,markeredgecolor='k'
,markeredgewidth=1)
    plt.grid(True)
def show_mixgauss_prm3(x,pi,mu,sigma,ratio):
    show_contour_mixgauss(pi,mu,sigma)
    for n in range(N):
        col=np.zeros(3)
        for k in range(K):
            col=col+ratio[n,k]*x_col[k]
```

```
        plt.plot(x[n,0],x[n,1],'o',color=tuple(col),markeredgecolor='black',markersize=6,alpha=1)
    for k in range(K):

plt.plot(mu[k,0],mu[k,1],marker='*',markerfacecolor=tuple(x_col[k]),markersize=15,markeredgecolor='k',m
arkeredgewidth=1)
    plt.grid(True)
#データ色分け等高なし、軸を選択
def show_mixgauss_prm4(x,pi,mu,ratio):
    for n in range(N):
        col=np.zeros(3)
        for k in range(K):
            col=col+ratio[n,k]*x_col[k,:]
        plt.plot(x[n,d0],x[n,d1],'o',color=tuple(col),markeredgecolor='black',markersize=6,alpha=1)
    for k in range(K):

plt.plot(mu[k,d0],mu[k,d1],marker='*',markerfacecolor=tuple(x_col[k]),markersize=15,markeredgecolor='k'
,markeredgewidth=1)
    plt.grid(True)
```

混合ガウスモデルによる非階層的クラスター分析に使うすべての関数を定義しています。また、結果を図示するための関数もここで定義しています。

def gauss(x,mu,sigma)は多変量正規分布密度関数の定義で、

$$\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k},\Sigma_k)=\frac{1}{\sqrt{(2\pi)^P|\Sigma_k|}}e^{\left\{-\frac{1}{2}(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T{\Sigma_k}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu_k})\right\}}$$

という計算をします。

```
for d in range(D):
        c5=c5+c4[:,d]*c3[:,d]
```

のところは、$(\boldsymbol{x_i}-\boldsymbol{\mu_k})^T{\Sigma_k}^{-1}(\boldsymbol{x_i}-\boldsymbol{\mu_k})$という３つの行列の積の計算の後半部分です。c4 を c4=np.dot(c3,inv_sigma)と計算したので c5=np.dot(c4,c3)と書きたいのですが、変数の形が合わないなどという警告が出てうまくいかず、どうしたらよいのかわからないので、行列演算を分解して計算しています。

def mixgauss(x,pi,mu,sigma)は、混合ガウスモデルによる確率密度関数の定義です。

$$GMF=\sum_{k=1}^{K}\pi_k\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k},\Sigma_k)$$

という計算をします。

def e_step_mixgauss(x,pi,mu,sigma)　EM アルゴリズムの E ステップ（負担率の更新）です。

$$\boldsymbol{\gamma}_{ik}=\frac{\pi_k\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k},\Sigma_k)}{GMF}=\frac{\pi_k\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_k},\Sigma_k)}{\sum_{j=1}^{K}\pi_k\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu}_j,\Sigma_j)}$$

という計算をします。

def m_step_mixgauss(x,gamma)　EM アルゴリズムの M ステップ（$\boldsymbol{\pi,\mu,\Sigma}$***の最適化***）です。

$$\sum_{i=1}^{n}\gamma_{ik}=N_k$$

$$\pi_k = \frac{N_k}{N}$$

$$\boldsymbol{\Sigma}_k = \frac{\sum_{i=1}^{n} \gamma_{ik} \left((\boldsymbol{x_i} - \boldsymbol{\mu_k})(\boldsymbol{x_i} - \boldsymbol{\mu_k})^T \right)}{N_k}$$

$$\boldsymbol{\mu_k} = \frac{\sum_{i=1}^{n} \gamma_{ik} \boldsymbol{x_i}}{N_k}$$

という計算をします。

def em_alg(max_it,Err,Pi,Mu,Sigma)　EM アルゴリズムを実行する関数です。

を決めて、def e_step_mixgauss(x,pi,mu,sigma),def nlh_mixgauss(x,pi,mu,sigma),

def m_step_mixgauss(x,gamma)を凝り返し数(max_it)だけ繰り返すことによって、負担率$\boldsymbol{\gamma}$、**誤差**

Err*、*Pi*,　*Mu*,　*Sigma を更新していきます。

def nlh_mixgauss(x,pi,mu,sigma)　誤差関数の定義です。

目的関数として誤差関数を－対数尤度と定義して、繰り返しごとに、

$$LL = \sum_{i=1}^{n} \log \sum_{j=1}^{K} \pi_j \, \mathcal{N}\left(\boldsymbol{x_i} | \boldsymbol{\mu_j}, \Sigma_j\right)$$

を計算し、誤差記録(*ERR*)を次のように記録します。

$$ERR = -log(LL)$$

def　likelihood(xx,mu,pi,sigma):は与えられた座標各クラスに配分される尤度とその確率を計算します。

以下は結果を図示するためのグラフの書き方に関する関数です。無計画に作ったので、同じような作図のプログラムが６つもならんでしまいました。もう少し整理できると思います。

def show_contour_mixgauss(pi,mu,sigma)多次元空間の点が混合ガウスモデルで得られる確率分布を等高線表示する関数です。空間の一平面に確率を投影して等高線を描きます。プログラムでは平面を縦横 40 のメッシュに区切って、41×41 の点について、混合ガウス関数でそれらの点の存在確率を計算します。多次元の場合、投影する平面を指定しないと投影できません。現在、平面を指定するプログラムを作っていないので、今のところこの関数は２変数のデータでなければ機能しません。実用的には３次元以上のデータについても平面を指定して投影図が描けるようにしなければなりませんが、サンプルデータを用意しなかったので、平面を指定するプログラムを作りませんでした。将来、サンプルデータを作って、多次元データでも使えるように平面を指定する機能を加える必要があります。

def show3d_mixgauss(ax,pi,mu,sigma) ３次元グラフで平面上に多次元確率分布を描きます。等高線表示と同じように、この関数は２変数のデータでないと機能しません。将来、多次元データでも使えるように平面を指定する機能を加える必要があります。

def show_mixgauss_prm(x,gamma,pi,mu,sigma)クラスター分析前のデータで等高線図と散布図を重ね書きする関数。def show_contour_mixgauss(pi,mu,sigma)を使っているため、今所２変数の場合のみ機能します。

def show_mixgauss_prm2(x,gamma,pi,mu,sigma)、クラスター分析前のデータで散布図のみを描く関数。等高線を描かないので、３変数以上でも機能します。

def show_mixgauss_prm3(x,pi,mu,sigma,ratio):クラスター分析後のデータで等高線図と散布図を重ね書きする関数。def show_contour_mixgauss(pi,mu,sigma)を使っているため、今所２変数の場合のみ機能します。散布図の各点の各クラスに属する確率を色の違いで表します。

def show_mixgauss_prm4(x,pi,mu,ratio): クラスター分析後のデータで散布図のみを描く関数。等高線を描かないので、３変数以上でも機能します。散布図の各点の各クラスに属する確率を色の違いで表します。

以下、３つのコードリストは、定義した関数の動作確認ですが、同時に、入力した初期値を適正化するための操作になっています。与えた初期値が不適切だと計算が止まってしまいます。これをした方がより確実に妥当な結果が得られやすいと思います。

VII-3-4-1-v.等高線図と 3d グラフの動作確認

```
#混合ガウス関数（等高線と 3d)
#等高線図と 3d
Gamma2,Ppk1,Ppl1,Ratio1=likelihood(X,Mu,Pi,Sigma)
Fig=plt.figure(1,figsize=(8,3.5))
Fig.add_subplot(1,2,1)
show_mixgauss_prm3(X,Pi,Mu,Sigma,Ratio1)
plt.grid(True)

Ax=Fig.add_subplot(1,2,2,projection='3d')
show3d_mixgauss(Ax,Pi, Mu,Sigma)
Ax.set_xlabel('$x_0$',fontsize=14)
Ax.set_ylabel('$x_1$',fontsize=14)
Ax.view_init(40,-100)
plt.xlim(X_range0)
plt.ylim(X_range1)
plt.show()
```

等高線図（show_mixgauss_prm3(X,Pi,Mu,Sigma,Ratio1)）3d グラフ（def show3d_mixgauss(ax,pi,mu,sigma)）の動作を確認します。同時に、def mixgauss(x,pi,mu,sigma)、def gauss(x,mu,sigma)の動作も確認しています。サンプルデータ(sample11)を使って例示した初期条件を与えた場合、図 148 が得られます。

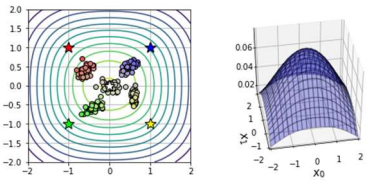

図 148.等高線図と 3 d グラフの動作確認の結果

VII-3-4-1-vi.E ステップの確認と Gamma の適正化

```
#e-step の動作試験
Gamma=e_step_mixgauss(X,Pi,Mu,Sigma)
plt.figure(1,figsize=(4,4))
show_mixgauss_prm3(X,Pi,Mu,Sigma,Ratio1)
plt.show()
```

E ステップ（def e_step_mixgauss(x,pi,mu,sigma)）の動作確認して、$\boldsymbol{\gamma}$を調整しています。サンプルデータと例示した初期条件だと図 148 が得られます。クラスの帰属確率が計算できていないのでγで色の混合率を計算していますので、色が濁っていますが、重心に近い点が同系統の色になっていること、重心の間の点が中間色になっています。重心（星印）の色は塗り分けられています。

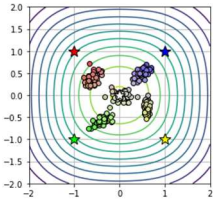

図 149.E ステップの動作確認。

VII-3-4-1-vii.動作試験（M ステップ）

```
#m-step の動作試験
Pi,Mu,Sigma=m_step_mixgauss(X,Gamma)
Gamma2,Ppk1,Ppl1,Ratio1=likelihood(X,Mu,Pi,Sigma)
plt.figure(1,figsize=(4,4))
show_mixgauss_prm3(X,Pi,Mu,Sigma,Ratio1)
plt.show()
print(Mu)
```

M ステップ（def m_step_mixgauss(x,gamma)）の動作確認です。外生変数的に$\boldsymbol{\gamma}$を与えて$\boldsymbol{X}$が得られる尤度を最大化するという方向で *Pi,Mu,Sigma* を最適化し、出来た混合ガウスモデルの形を等高線として示します。図 150 にその結果を示しました。図 149 とは、重心（星印）の位置と等高線の形が変化して M ステップによって混合ガウスモデルの形が変化したことがわかります。

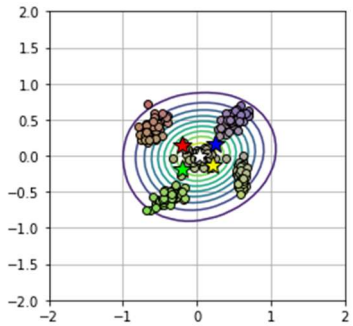

図 150.M ステップの動作確認

VII-3-4-1-viii. EM アルゴリズムの実施

```
#試行回数を決めて EM アルゴリズムを実施
max_it=400
```

```
Err=np.zeros(max_it)
Err1=Err
Pi1=Pi
Mu1=Mu
Sigma1=Sigma
Err2,Pi2,Mu2,Sigma2,Gamma2=em_alg(max_it,Err1,Pi1,Mu1,Sigma1)
plt.figure(2,figsize=(4,4))
plt.plot(np.arange(max_it)+1,Err2,color='k',linestyle='-',marker='o')
plt.grid(True)
plt.show()
print(Mu2)
```

E ステップ、M ステップを実施して調整された *Pi,Mu,Sigma,Gamma* を使って EM アルゴリズムを実行します。ここでは、繰り返し数(*max_it=200*)を 400 回としました。結果として *Err2,Pi2,Mu2,Sigma2,Gamma2* が返ってきて繰り返しごとの ***Err2*** をプロットした図（図 151）が示されます。

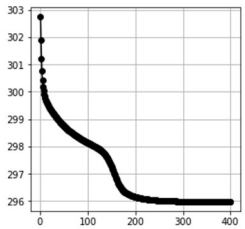

図 151.EM アルゴリズム実施中の誤差の時間変化

図 151 により、ほぼ 300 回の EM アルゴリズムの実施によって、混合ガウスモデルが収束したことがわかります。300 目以後誤差関数の値は全く変化しませんでした。したがって、誤差が最小値に達していると判断しても良いでしょう。収束時の重心の座標も返ってきます。

以下は結果の妥当性を検討するためのプログラムです。

VII-3-4-1-ix. 入力データの尤度

```
#個々入力データの尤度
xx=X
Gamma2,Ppk1,Ppl1,Ratio1=likelihood(xx,Mu2,Pi2,Sigma2)
print(Ppl1)
print(Ratio1)
```

入力したデータが各データが、それぞれのクラスに属する尤度を計算し返します (Ppl1)。尤度の総和を 1 として、全体に対する各クラスに属する尤度の比を各クラスに属する確率として返します (Ratio1)。例えば入力したデータの 250 番目の座標がクラス 1 に属する確率は、0.16。クラス 2 に属する確率が 0.00、クラス 3 に属する確率は 0.14 です。数クラス 4 に属する確率は 0.57、クラス 5 に属する確率は 0.12 です。数字ではわかりにくいので、この結果を色分けした散布図で示します。

VII-3-4-1-x. 結果の図示

```
#結果を色分け図で表示する。
#表示する軸を選択する
#変数の選択
dim1=1
```

```
dim2=2
x_range=[-2,2] #項目 1 の範囲
y_range=[-2,2] #項目 2 の範囲
d0=dim1-1
d1=dim2-1
plt.figure(2,figsize=(4,4))
show_mixgauss_prm4(X,Pi2,Mu2,Ratio1)
plt.show()
```

例として取り上げている sample11 の場合、図 152 のように結果が示されます。

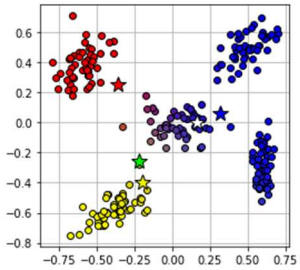

図 152 クラスター分析の結果

右上の集団と右下の集団が、同じ青のクラスに属することになっていて、中央部の集団も色が混ざり合っていて、一つの集団としてまとまっていません。この例の場合、我々はこのデータが 5 つのクラスに分かれるということを知っています。また、それを知らなくても、5 つの集団にクラスター分析するのが妥当だと視覚的にもわかります。ここ結果を妥当な解として受け入れられないことは明らかです。図 151 に戻ってみると、確かに、誤差は一定の値で変化しなくなって収束していますが、誤差の変化量は小さく、まだ誤差がかなり大きいことがわかります。図 152 から、EM アルゴリズムが部分的な極値に捉えられて、最小値にたどり着けなくなったのではないかと推測できます。

VII-3-4-3.　EM アルゴリズムでの混合ガウスモデルの動き

このような場合、部分的な極値から抜け出さなくてはなりませんが、その方法はいろいろありそうです。ここでは、重心の結果だけを受け入れて、Pi,Sigma は最初に与えた初期値に戻して EM アルゴリズムを再実行してみます。

VII-3-4-1-xi. 混合ガウスモデルの再計算

```
#初期条件
Pi2=Pi0
Mu2=Mu1
Sigma2=Sigma0
N=X.shape[0]
K=len(Pi)
Gamma2=Gamma
plt.figure(1,figsize=(10,6.5))
max_it=200
Err2=np.zeros((max_it))
i_subplot=1;
for it in range(0,max_it):
    Gamma2=e_step_mixgauss(X,Pi2,Mu2,Sigma2)
```

```
    Err2[it]=nlh_mixgauss(X,Pi2,Mu2,Sigma2)
    Pi2,Mu2,Sigma2=m_step_mixgauss(X,Gamma2)
    Gamma2,Ppk4,Ppl4,Ratio4=likelihood(xx,Mu2,Pi2,Sigma2)
    if it<=0:
        plt.subplot(2,3,i_subplot)
        show_mixgauss_prm3(X,Pi2,Mu2,Sigma2,Ratio4)
        plt.title("{0:d}".format(it+1))
        plt.xticks(range(X_range0[0],X_range0[1]),"")
        plt.yticks(range(X_range1[0],X_range1[1]),"")
        i_subplot=i_subplot+1
    if it>3 and it<5:
        plt.subplot(2,3,i_subplot)
        show_mixgauss_prm3(X,Pi2,Mu2,Sigma2,Ratio4)
        plt.title("{0:d}".format(it+1))
        plt.xticks(range(X_range0[0],X_range0[1]),"")
        plt.yticks(range(X_range1[0],X_range1[1]),"")
        i_subplot=i_subplot+1
    if it>8 and it<10:
        plt.subplot(2,3,i_subplot)
        show_mixgauss_prm3(X,Pi2,Mu2,Sigma2,Ratio4)
        plt.title("{0:d}".format(it+1))
        plt.xticks(range(X_range0[0],X_range0[1]),"")
        plt.yticks(range(X_range1[0],X_range1[1]),"")
        i_subplot=i_subplot+1
    if it>48 and it<50:
        plt.subplot(2,3,i_subplot)
        show_mixgauss_prm3(X,Pi2,Mu2,Sigma2,Ratio4)
        plt.title("{0:d}".format(it+1))
        plt.xticks(range(X_range0[0],X_range0[1]),"")
        plt.yticks(range(X_range1[0],X_range1[1]),"")
        i_subplot=i_subplot+1
    if it>98 and it<100:
        plt.subplot(2,3,i_subplot)
        show_mixgauss_prm3(X,Pi2,Mu2,Sigma2,Ratio4)
        plt.title("{0:d}".format(it+1))
        plt.xticks(range(X_range0[0],X_range0[1]),"")
        plt.yticks(range(X_range1[0],X_range1[1]),"")
        i_subplot=i_subplot+1
    if it>198:
        plt.subplot(2,3,i_subplot)
        show_mixgauss_prm3(X,Pi2,Mu2,Sigma2,Ratio4)
        plt.title("{0:d}".format(it+1))
        plt.xticks(range(X_range0[0],X_range0[1]),"")
        plt.yticks(range(X_range1[0],X_range1[1]),"")
        i_subplot=i_subplot+1
plt.show()
plt.figure(2,figsize=(4,4))
plt.plot(np.arange(max_it)+1,Err2,color='k',linestyle='-',marker='o')
plt.grid(True)
plt.show()
print(Mu2)
```

このプログラムでは VII-3-4-viii と同様に EM アルゴリズムを実行しているのですが、途中経過を見たいので、途中の散布図と等高線図を図示しました（図 153)。

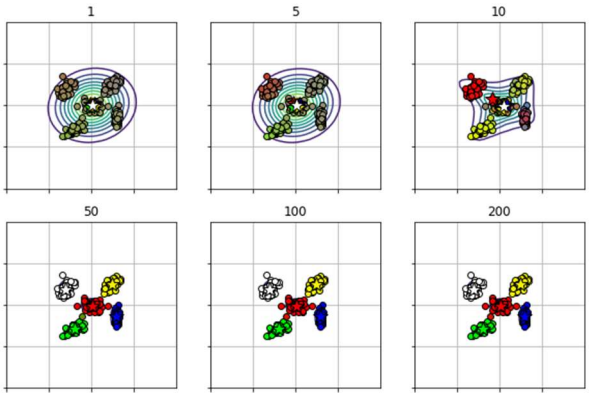

図 153.等高線、重心、散布図の時間変化

重心は一旦中央に集まりその後、周辺に広がっていき、それに伴ってそれぞれのクラスの色が鮮明に塗り分けられます。添れに伴って、誤差も小さくなり、負の値になって収束します（図 154）。誤差が最小化されている可能性は極めて高いでしょう。

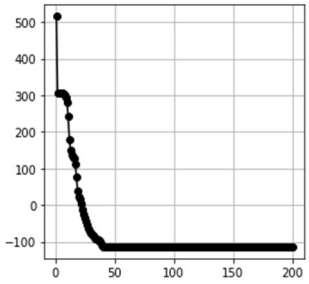

図 154.再計算時の誤差の時間変化

VII-3-4-1-xii. 再計算後の混合ガウスモデルの形

```
#混合ガウス関数（等高線と 3d）
#等高線図と 3d
Fig=plt.figure(1,figsize=(8,3.5))
Fig.add_subplot(1,2,1)
show_contour_mixgauss(Pi2,Mu2,Sigma2)
plt.grid(True)
Ax=Fig.add_subplot(1,2,2,projection='3d')
show3d_mixgauss(Ax,Pi2, Mu2,Sigma2)
Ax.set_xlabel('$x_0$',fontsize=14)
Ax.set_ylabel('$x_1$',fontsize=14)
Ax.view_init(40,-60)
plt.xlim(X_range0)
plt.ylim(X_range1)
plt.show()
```

図 153 は、等高線が見にくいので、等高線をとりだし、３ｄのグラフともに示したものが、図 155 です。５つのクラスに明瞭に分かれていることがわかります。

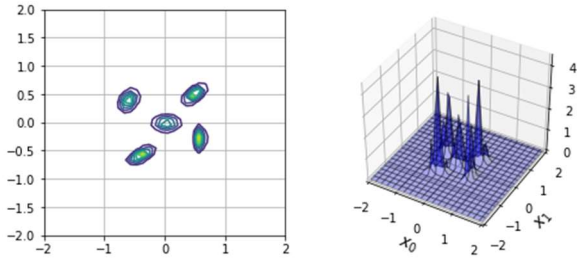

図 155.再計算収束時の混合ガウスモデル

VII-3-4-1-xiii. 入力データが各クラスに属する確率

```
#個々入力データの尤度
xx=X
Gamma2,Ppk2,Ppl2,Ratio2=likelihood(xx,Mu2,Pi2,Sigma2)
print(Ppl2)
print(Ratio2)
```

表 71.再計算後のクラス分け

		座標		再計算				
No	c	x1	x2	c1	c2	c3	c4	c5
1	1	0.41	0.42	0.00	0.00	0.00	1.00	0.00
2	1	0.32	0.39	0.00	0.00	0.00	1.00	0.00
51	2	-0.5	0.46	0.00	0.00	0.00	0.00	1.00
52	2	-0.7	0.26	0.00	0.00	0.00	0.00	1.00
101	3	-0.4	-0.6	0.00	0.00	1.00	0.00	0.00
102	3	-0.3	-0.5	0.00	0.00	1.00	0.00	0.00
151	4	0.58	-0.3	1.00	0.00	0.00	0.00	0.00
152	4	0.62	-0.1	1.00	0.00	0.00	0.00	0.00
201	5	-0.33	-0.03	0.00	1.00	0.00	0.00	0.00
202	5	0.07	-0.05	0.00	1.00	0.00	0.00	0.00

クラスター分析に使った個々のデータが各クラスに属する確率を計算します。結果の一部を取り出して表 71 に示しました。元のデータは筆者が 5 つのクラス別に乱数を与えて作ったものです。元々のクラスを表の 2 列目（c）に示しました。元のクラス番号が 1 だったデータは再計算では例外なくクラス番号 4（黄色）と識別され、元のクラス番号が 2 だったデータはクラス番号 5（白）に、元のクラス番号が 3 であったデータはクラス番号 3（緑）に、元のクラス番号が 4 であったデータはクラス番号 1（青）に、元のクラス番号が 5 であったデータはクラス番号 2 に、例外なくそ仕分けられています。再計算では識別される確率はほとんど 1 になりました。つまり、完璧にクラスを判別しているということです。この結果を図示すると、図 156 が得られます。

VII-3-4-1-xiv.再計算結果の図示（VII-3-4-x と同じ）

```
#結果を色分け図で表示する。
#表示する軸を選択する
#変数の選択
dim1=1
dim2=2
x_range=[-2,2] #項目 1 の範囲
y_range=[-2,2] #項目 2 の範囲
```

```
d0=dim1-1
d1=dim2-1
plt.figure(2,figsize=(4,4))
show_mixgauss_prm4(X,Pi2,Mu2,Ratio2)
plt.show()
```

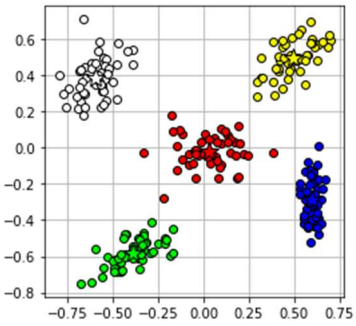

図 156．再計算の結果（色分け散布図）

識別が明瞭で色の重なり合いが少なく鮮やかな色の散布図になります。これは所属確率がほとんど 1 だからです。念のため、任意の座標について、所属確率を計算してみます。

VII-3-4-1-xv. 任意の座標の尤度と所属確率

```
#データの尤度の評価
#評価するデータの入力
xxx=np.array([0.2,0.25])
g=np.zeros((K))
ppk3=np.zeros((K))
ppl3=np.zeros((K))
c1=-(D/2)*np.log(2*np.pi)
for k in range(K):
    det_sigma=np.linalg.det(Sigma2[k])
    c2=-(1/2)*np.log(det_sigma)
    inv_sigma=np.linalg.inv(Sigma2[k])
    c3=xxx-Mu2[k]
    c4=np.dot(c3,inv_sigma)
    c5=np.dot(c4,c3)
    c5=-c5/2
    p=c1+c2+c5
    g[k]=np.exp(p)
    ppk3[k]=Pi2[k]*g[k]
S=0
for k in range(K):
    S=S+ppk3[k]
for k in range(K):
    ppl3[k]=g[k]*ppk3[k]/S
print(ppl3)
ratio3=np.zeros((K))
SS=0
for k in range(K):
    SS=SS+ppl3[k]
for k in range(K):
        ratio3[k]=ppl3[k]/SS
print(ratio3)
```

任意の座標を与えてその点のデータが各クラスに属する尤度と確率を計算します。(0.2　0.25)

の座標を与えると、[2.27813184e-72 1.27404080e-02 1.00655088e-47 1.46802039e-01 2.38162268e-39]　の尤度と[1.42791582e-71 7.98559149e-02 6.30898483e-47 9.20144085e-01 1.49278310e-38]の確率が返ってきます。クラス 4（黄色）に属する確率が 0.92、クラス 2（赤）に属する確率が 0.08 ということです。
これは人為的に合成されたデータで、私たちはクラス分けの正解を知っています。これらの結果は極めて納得がいく結果だと言えるでしょう。納得できれば、この結果を保存します。

VII-3-4-1-vi. 結果の保存

```
#結果の保存
df=pd.DataFrame(Err2)
df.to_csv('Error2_r.csv')
df=pd.DataFrame(Pi2)
df.to_csv('Pi2.csv')
df=pd.DataFrame(Mu2)
df.to_csv('Centers2.csv')
df=pd.DataFrame(Gamma2)
df.to_csv('Gamma2.csv')
df=pd.DataFrame(Ppl2)
df.to_csv('Likelihood2.csv')
df=pd.DataFrame(Ratio2)
df.to_csv('predict-P2.csv')
print(Sigma2)
```

得られた数値データを CSV で保存します。分散共分散行列は、3 次元の配列になっていて、各クラスごとに行列を保存しなければなりません。保存するファイルが大きくなってしまうので、ここでは保存しないことにして、二次元の配列を出力しました。
上記の結果は完璧に、元のクラス分けを再現しており、混合ガウスモデルによるクラスタリングが絶対に優れたクラスタリングのような気がしてきます。Sample11 は各クラスターがまとまって存在していて、重なり合いがないから、このような結果が得られるのです。ここで理解しておかなくてはならないのは、混合ガウスモデルでは、初期条件の与え方によって、結果が違ってくるということです。それは、部分的な極値に収束してしまううことがあるからです。それを防ぐためには、いくつか異なる初期条件を与えてどのような状況で収束したのかを見て、誤差の最小値と言えるかどうかを判断することが必要です。いつでも Sample11 のように完璧な答えにたどり着けるわっけではありません。

VII-3-4-4.　Scikit learn.mixuture を使った混合ガウスモデルのプログラム
大変長い複雑なプログラムを作っておきながら、今更なにをという感じですが、混合ガウスモデルの計算は、Scikit learn の mixuture というプログラムをインポートすれば簡単にできます。計算の仕方にはいくつかのオプションがありますが、それらの選択も容易にできます。
以下、Scikit learn.mixutre を使った混合ガウスモデルのプログラム・コード・リストを示します。実用的にはこのプログラムを使うことになるでしょう。「やさしい水産学（自習室）」-「参考資料」-「python」-「VII-3-4-2. 混合ガウスモデル(Scikit.learn.mixuture)」にプログラムのコードリストをあげておきました。

VII-3-4-2. 混合ガウスモデル(#scikit -learn、GMMによる非階層的クラスター分析)

VII-3-4-2-i. ライブラリーから必要なプログラムをインポートしてデータを入力

```
#Ssci-kit -learn、GMMによる非階層的クラスター分析
#必要なライブラリーのインポート
#[A]必要なライブラリーの読み込み
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from scipy.cluster.hierarchy import linkage,dendrogram,fcluster
from sklearn import cluster, preprocessing,mixture
import decimal
decimal.getcontext().prec=6

#[B]データの読み込み
df =pd.read_csv("sample11.csv")
#データフレームをつくる
dfX=pd.DataFrame(df)
X0=df.values
X=np.delete(X0,0,1)
```

後で、筆者がデータを作った時の元々のクラス分けと分析の結果を比較するために、データフレームとして読み込んだデータから、元々のクラス分けを含む配列（X0）とクラス分けを含まない配列（X）の二つのデータセットを作りました。

VII-3-4-2-ii.混合ガウスモデルによる非階層的クラスター分析の実行

```
X=X
x1=1 #作図する平面の選択x軸
x2=2 #作図する平面の選択y軸
x1=x1-1
x2=x2-1
x=X[:,x1]
y=X[:,x2]
nc=5 #クラスター数の決定
#実行
gmm=mixture.GaussianMixture(n_components=nc,covariance_type='full',max_iter=250,n_init=10,init_params='
kmeans')
z_gmm=gmm.fit(X)
predict1=z_gmm.predict(X)
print(predict1) #クラスの判別
print(gmm.means_) #各クラスの重心
print(gmm.covariances_) #各クラスの分散共分散行列
print(gmm.weights_) #混合係数
print(gmm.aic(X)) #赤池情報基準
print(gmm.bic(X)) #ベイズ情報基準
print(gmm.score(X))#平均対数尤度
#結果を図示
#平面を決定
plt.figure (1,figsize=(4,4))
plt.scatter(x,y,c=predict1)
plt.show()
```

分析に供するデータを指定し、作図する平面の x 軸 y 軸、クラスター数を指定して、混合ガウスモデルを作ります。(gmm=mixture.GaussianMixture(n_components=nc,covariance_type='full', max_iter=250,n_init=10,init_params='kmeans')。混合ガウスモデルの作り方にはいくつかオプションがあって、n_components はクラスの数です。 ここでは入力したクラスターの数 nc を指定します。covariance_type は分散・共分散行列（シグマ）の形の指定で full、tied、diag、spherical の 4 つのオプションがあります。デフォルトは full です。full は混合ガウスモデルで解説した通りに、分散・共分散行列を

$$\Sigma_k = \begin{pmatrix} \sigma_{k11} & \sigma_{k12} & \cdots & \sigma_{k1D} \\ \sigma_{k21} & \sigma_{k22} & \cdots & \sigma_{k2D} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{kD1} & \sigma_{kD2} & \cdots & \sigma_{kDD} \end{pmatrix}$$

という形で与えます。D=2 の二次元の場合、分布範囲を図持すると、図 155 の full のように、各クラスに属する確率密度の分布は各クラスごとに大きさ、扁平率、傾きの異なる楕円球の形になります。tied を選択すると、分散共分散行列は

$$\Sigma_1 = \Sigma_2 = \cdots = \Sigma_K = \Sigma = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1D} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{D1} & \sigma_{D2} & \cdots & \sigma_{DD} \end{pmatrix}$$

の形で指定されて、図 155 の tied に示したように、どのクラスも同じ大きさ、扁平率、傾きを持った楕円球の形の確率密度分布になります。diag を選択すると、分散・共分散行列は

$$\Sigma_k = \begin{pmatrix} \sigma_{k11} & 0 & \cdots & 0 \\ 0 & \sigma_{k22} & \cdots & 0 \\ & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{kDD} \end{pmatrix}$$

という形、つまり、対角成分の分散だけがあり、共分散（項目間の相関）がない形で指定します、確率密度の分布は図 157 の diag のような、傾きを持たない楕円球の形になります。Spherical を選択すると、

$$\Sigma_k = \sigma_k$$

という形で分散・共分散行列が指定され、どの次元に対しも同じ分散をもつ確率分布が指定されて、確率密度分布は図 157 の spherical のような球状になります。本来、混合ガウスモデルは様々な分布の仕方をしている確率密度を重ね合わせるので、分散共分散行列は full の形にすべきです。しかし、full の形だと推定するパラメータの数が多いので、部分最適的な極値に陥って、妥当なガウスモデルの形にたどり着けない可能性が高くなります。それを避けるためにパラメータの数を減らすためのオプションがあるのです。

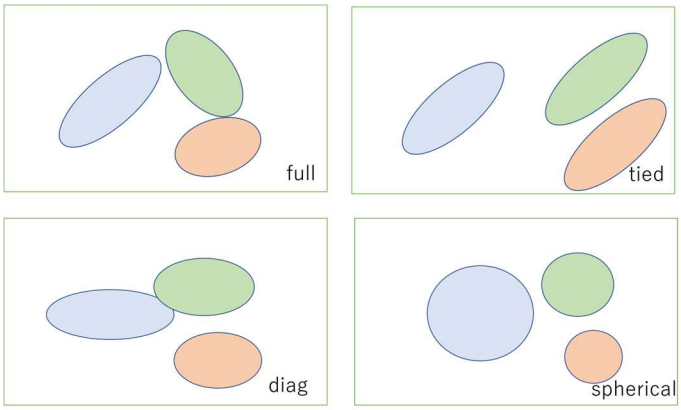

図 157.分散・共分散行列の指定と確率密度分布の違い（２変数の場合）

しかし、筆者は、それならば K-means 法を使えばよいような気がします。実際に、これらのオプションがどのように使われているのかよく知りません。max_iter は EM アルゴリズムが収束しない場合に、EM アルゴリズムを最大何回繰り返すのかというオプションです。デフォルトは 100 です。筆者が自作したプログラムでいくつか試したところ、100 回以内で収束する例は少ないようです。ここでは 250 回を指定しました。n_init は初期値を何回与えなおすかというオプションでデフォルトは 1 回です。init_params は初期値の与え方に関するオプションで、K-means 法の結果を使う、kmeans とランダムに与える random の二つのオプションがあります。デフォルトは kmeans です。random を使うとなかなか妥当解にたどり着きません。z_gmm=gmm.fit(X)でデータセット X を与えて、混合ガウスモデルによるクラスター分析を実行して、結果を z_gmm に保存します。次に、predict1=z_gmm.predict(X)で z_gmm を使って入力データ X のクラスの所属を判定し、predict1 に保存します。その結果を print(predict1)で出力します。print(gmm.means_)は各クラスの重心（$\boldsymbol{\mu}$）、print(gmm.covariances_)は各クラスの分散共分散行列（$\boldsymbol{\Sigma}$）、print(gmm.weights_) は各クラスの混合係数（$\boldsymbol{\pi}$）の出力です。Sample 11 を分析した場合、それぞれ次の値が出力されました。

重心

```
[[ 0.03437855 -0.01319453][ 0.59958001 -0.28628318][-0.59825888  0.3
8523758][ 0.49289727  0.49619117][-0.38890005 -0.57631267]]
```

分散共分散行列

```
[[[ 0.01775181 -0.00050421][-0.00050421  0.00668093]] [[ 0.00106675
 0.00012777][ 0.00012777  0.01425081]] [[ 0.00842054  0.00347018]
  [ 0.00347018  0.01204825]][[ 0.01061688  0.00584278][ 0.00584278
0.00879057]][[ 0.0136465   0.00639771][ 0.00639771  0.00708448]]]
```

混合係数

```
[0.19999045 0.19687827 0.20001219 0.20311919 0.19999989]
```

print(gmm.aic(X))、 print(gmm.bic(X))、print(gmm.score(X))は推定されたモデルの評価にかかわる数値で、それぞれ、赤池情報基準（*AIC*）、ベイズ情報基準（*BIC*）、平均対数尤度です。赤池

情報基準、ベイズ情報基準はモデルの評価に使われる指標です。モデルの当てはまりの良さは尤度あるいは対数尤度で評価しますが、当てはまりが良ければ良いモデルとは言えません。モデルのパラメータの数を増やしていけば最終的には、完全に当てはまるモデルが作れます。そのようなモデルは、たまたまサンプリングされたデータにオーバーフィッティングしているのです。同じ母集団からサンプリングされた他のデータセットには当てはまりが悪いでしょう。妥当性を論ずるには、尤度からパラメータの数の影響を差し引かなくてはならないのです。赤池情報基準とベイズ情報基準はパラメータの数を考慮したモデルの評価基準です。その式の導出から始めてその意味を解説すると大変長い解説になってしまいます。それについては教科書もありますので、解説を省略して式だけを示します。いずれも値が小さい方がモデルの評価が高くなります。

$$AIC = -2\log_e L + 2k$$
$$BIC = -2\log_e L + k\log_e n$$
L: *最大対数尤度*
k: *パラメータの数*
n: *サンプルサイズ*

AIC と *BIC* は異なる目的から導出されたのですが、結果的に極めてよく似た形になっています。違いは *BIC* ではサンプルサイズが考慮されているということです。ここで、理解しておかなければならないことは、サンプリングされたデータへの当てはまりの良さ（尤度）と、モデルを一般化した時そのモデルが妥当である可能性とは異なるということです。実際にクラスター数を 1 から 6 まで変化させて、*AIC*、*BIC*、平均対数尤度を計算した結果を図 158 に示しました。我々の期待通り、平均対数尤度はクラスの数の増加とともに増加しますが、*AIC*、*BIC* はクラスの数が 5 の時に最低値となり、クラス数を 5 とするモデルが妥当であることを示しています。入力データのクラスの判別結果は図 159 の混合ガウスモデルの散布図のようになりました。赤の矢印のデータの判定結果だけが元データと異なりますが、他はすべて元データと一致しました。赤い矢印のデータは中央のクラスと左下のクラスの中間に位置しどちらのクラスの分布中心から遠く、尤度が小さいためにその誤差が大きく、自作したプログラムによる結果と scikit learn.mixuture による判定結果が違ってくるのだと思います。

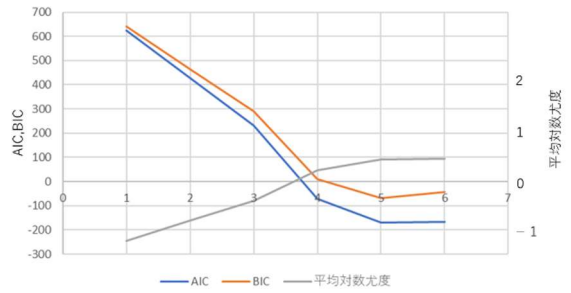

図 158. クラスの数に伴う *AIC*、*BIC*、平均対数尤度の変化(Sample11)

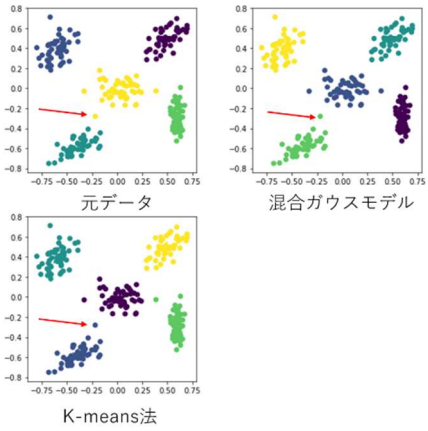

図 159.元データと混合ガウスモデル・K-means 法による判別結果の比較(Sample11)

これ以降は、VII-3-2-1 で作った自作のプログラムの一部を使って経緯と結果の確認しています。

VII-3-4-2-iii.元データのクラス分けを図示。

```
#元のクラス分け
X=X
x1=1
x2=2
x1=x1-1
x2=x2-1
x=X[:,x1]
y=X[:,x2]
z=X0[:,0]
plt.figure (1,figsize=(4,4))
plt.scatter(x,y,c=z)
plt.show()
```

VII-3-4-2-iv.K-means 法によるクラス分け図示

```
X=X
x1=1
x2=2
x1=x1-1
x2=x2-1
x=X[:,x1]
y=X[:,x2]
km=cluster.KMeans(n_clusters=5)
z_km=km.fit(X)
plt.figure(1,figsize=(4,4))
plt.scatter(x,y,c=z_km.labels_)
plt.scatter(z_km.cluster_centers_[:,x1],z_km.cluster_centers_[:,x2],s=250,marker='*',c=[0,1,2,3,4])
plt.show()
print(z_km.cluster_centers_)
```

リスト VII-3-4-2-v.作図のための関数定義

```
#関数の定義
#ガウス関数の定義
def gauss(x,mu,sigma):
    N,D=x.shape
    c1=-(D/2)*np.log(2*np.pi)
```

```
    det_sigma=np.linalg.det(sigma)
    c2=-(1/2)*np.log(det_sigma)
    inv_sigma=np.linalg.inv(sigma)
    c3=x-mu
    c4=np.dot(c3,inv_sigma)
    c5=np.zeros(N)
    for d in range(D):
        c5=c5+c4[:,d]*c3[:,d]
    c5=-c5/2
    p=c1+c2+c5
    p=np.exp(p)
    return p
#混合ガウスモデル
def mixgauss(x,pi,mu,sigma):
    N,D=x.shape
    K=len(pi)
    p=np.zeros(N)
    for k in range(K):
        p=p+pi[k]*gauss(x,mu[k,:],sigma[k,:,:])
    return p
#個々入力データの尤度
def likelihood(xx,mu,pi,sigma):
    N,D=xx.shape
    ppk1=np.zeros((N,K))
    ppl1=np.zeros((N,K))
    g1=np.zeros((N,K))
    S1=np.zeros((N))
    for k in range(K):
        g1[:,k]=gauss(xx,mu[k,:],sigma[k,:,:] )
        ppk1[:,k]=pi[k]*g1[:,k]
    for n in range(N):
        for k in range(K):
            S1[n]=S1[n]+ppk1[n,k]
        for k in range(K):
            ppl1[n,k]=g1[n,k]*ppk1[n,k]/S1[n]
    ratio1=np.zeros((N,K))
    for n in range(N):
        SS1=0
        for k in range(K):
            SS1=SS1+ppl1[n,k]
        for k in range(K):
            ratio1[n,k]=ppl1[n,k]/SS1
    return g1,ppk1,ppl1,ratio1

#データの図示
#混合ガウス等高線表示
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
%matplotlib inline
def show_contour_mixgauss(pi,mu,sigma):
    xn=40 #解像度
    x0=np.linspace(X_range0[0],X_range0[1],xn)
    x1=np.linspace(X_range1[0],X_range1[1],xn)
    xx0,xx1=np.meshgrid(x0,x1)
```

```
    A=xx0.reshape(xn*xn,1)
    B=xx1.reshape(xn*xn,1)
    x=np.c_[B,A]
    f=mixgauss(x,pi,mu,sigma)
    f=f.reshape(xn,xn)
    f=f.T
    plt.contour(x0,x1,f,10,color="grey")
#混合ガウス 3D 表示
def show3d_mixgauss(ax,pi,mu,sigma):
    xn=40 #解像度
    x0=np.linspace(X_range0[0],X_range0[1],xn)
    x1=np.linspace(X_range1[0],X_range1[1],xn)
    xx0,xx1=np.meshgrid(x0,x1)
    A=xx0.reshape(xn*xn,1)
    B=xx1.reshape(xn*xn,1)
    x= np.c_[B,A]
    f=mixgauss(x,pi,mu,sigma)
    f=f.reshape(xn,xn)
    f=f.T
    ax.plot_surface(xx0,xx1,f,rstride=2,cstride=2,alpha=0.3,color='blue',edgecolor='black')
```

VII-3-4-2-vi.等高線図と３D 確率分布

```
Pi2=gmm.weights_
Mu2=gmm.means_
Sigma2=gmm.covariances_
X_range0=[-2,2] #項目 1 の範囲
X_range1=[-2,2] #項目 2 の範囲
#混合ガウス関数 (等高線と 3d)
#等高線図と 3d
Fig=plt.figure(1,figsize=(8,3.5))
Fig.add_subplot(1,2,1)
show_contour_mixgauss(Pi2,Mu2,Sigma2)
plt.grid(True)
Ax=Fig.add_subplot(1,2,2,projection='3d')
show3d_mixgauss(Ax,Pi2, Mu2,Sigma2)
Ax.set_xlabel('$x_0$',fontsize=14)
Ax.set_ylabel('$x_1$',fontsize=14)
Ax.view_init(40,-60)
plt.xlim(X_range0)
plt.ylim(X_range1)
plt.show()
```

以下は、推定された係数うう使ったクラス判別と尤度・確率の計算です。

VII-3-4-2-vii.所属クラスの判別

```
Y=[[0.5,0.5],[-0.6,0.4],[-0.3,-0.6],[0.6,-0.3],[0,0]]
predict2=z_gmm.predict(Y)
print(predict2)
```

VII-3-4-2-viii. 任意の座標のクラスに属する尤度と確率

```
#
x=[[-0.21877,-0.28051],[-0.6,0.4],[-0.3,-0.6],[0.6,-0.3],[0.0]]
N=len(x)
D=len(x[0])
xx=np.zeros((N,D))
for n in range(N):
```

```
    xx[n]=x[n]
K=nc
Gamma2,Ppk2,Ppl2,Ratio2=likelihood(xx,Mu2,Pi2,Sigma2)
print(Ppl2)
print(Ratio2)
```

VII-3-4-5. 混合ガウスモデルの限界

混合ガウスモデルに限らず、クラスター分析には限界があります。もともとのデータの中に明瞭に分布の異なる小集団があれば、どんなクラスター分析もある程度の精度で、その小集団を識別します。中でも、混合ガウスモデルによるクラスター分析は、無理のある前提条件にたっていないので、リアリティーがあり、高い感度で小集団を識別できます。それでも、元々のデータで小集団の分布に重なりが多い場合には、それらを識別することは困難です。試しに、Sample11 と同じ分布中心を持ち、分散を１０倍に大きくした Sample10 を scikit.learn.mixuture でクラスター分析をしてみました。図 160 にクラスター数を変化させたときの *AIC*、*BIC*、平均対数尤度の変化を示しました。*BIC* はクラスター数 3 で最小値となり、*AIC* はクラスター数 5 で最小値となりました。元のデータを知っている私たちとしては、*AIC* の結果が正しいと判断したくなりますが、*AIC* の数値は極めて高く、平均対数尤度も低い値のままです。提案している混合ガウスモデルの信頼性はあまり高くないと判断すべきでしょう。クラス数を 5 にした時の混合ガウスモデルの判定結果を、元データ、K-means 法の結果と比べてみました（図 161)。当然のことですが、重なり合いが多いために、元データと分析結果は大きく異なっています。

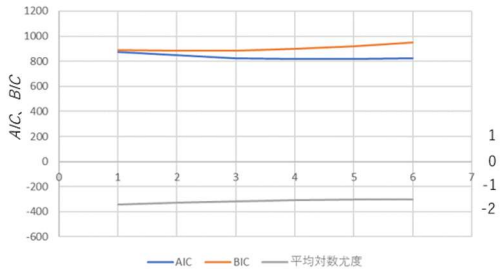

図 160. クラスの数に伴う *AIC*、*BIC*、平均対数尤度の変化(Sample10)

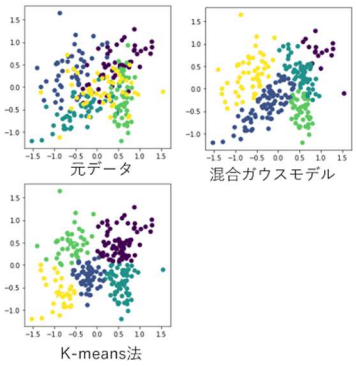

図 161.元データと混合ガウスモデル・K-means 法による判別結果の比較(Sample10)

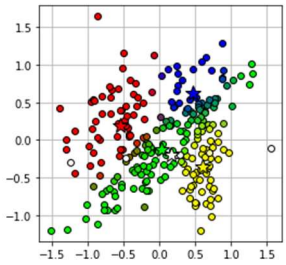

図 162．Sample10 再計算の結果（色分け散布図）

念のために自作のプログラムで同じデータを 5 つのクラスに分けると図 162 のようになります。同じ混合ガウスモデルで分析しても結果が若干異なります。これは、収束しないで、与えた初期条件の影響が残っているからだと思います。しかし、似ていることは似ています。両者の混合ガウスモデルの形を比較してみます（図 163、図 164）。軸のスケールが違うのでわかりにくいのですが、似たような形をしています。自作プログラムでは誤差の時間変化を視覚化できるので、視覚化してみると、図 165 のようになり、収束してもまだかなり誤差が大きいことがわかります。この解説では、クラス分けがあらかじめわかっているデータを分析しました。実際の分析は、そのような事前情報がない状態で分析します。誤差関数の時間変化や、平均対数尤度、AIC、BIC などの数値を参考に、分析結果を評価することになります。クラスター分析の目的は様々です。重なり合いが大きいデータの場合、クラスの存在を発見することが目的ならば Ward 法による階的クラスター―分析が有効でしょう。データの特性や、調べる目的によって、妥当なクラスタリングの方法は違ってきます。いくつかの方法を試してみるべきでしょう。さらに、分析者はいろいろな情報を持っています。それらの情報と照らし合わせてクラスタリングの意味を考えるということも重要です。

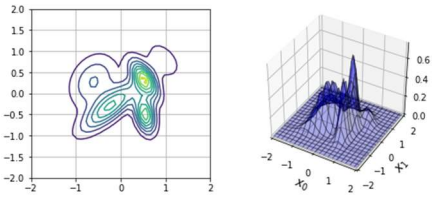

図 163．Scikit-klearn.mixture による Sample10 の混合ガウスモデル

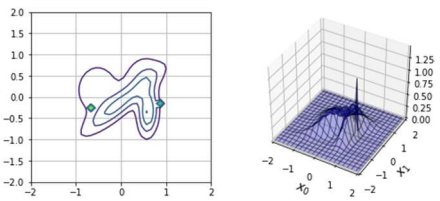

図 164．自作プログラムによる Sample10 の混合ガウスモデル）

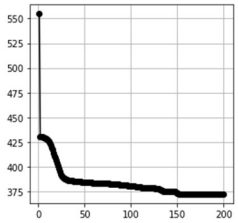

図 165．Sample10 再計算の結果(誤差の時間変化)