# EM アルゴリズム

EM アルゴリズムの導出と C++ での実装

2016 年 9 月 1 日

kivantium 活動日記 「C++ を使った EM アルゴリズムの実装（+ Python によるプロット)」[*1]の内容を実行してみる。

## 1 導出

### 1.1 準備

混合ガウス分布は、

$$\sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \tag{1}$$

で表される分布。$\sum_{k=1}^{K} \pi_k = 1,\ 0 \leq \pi_k \leq 1$ に注意。$\pi_k$ が $k$ 番目のガウス分布の割合を表している。

$D$ 次元ガウス分布は、

$$\mathcal{N}(x|\mu_k, \Sigma_k) = \frac{1}{(2\pi)^{\frac{D}{2}}} \frac{1}{|\Sigma|^{\frac{1}{2}}} \exp\left\{ -\frac{1}{2}(x-\mu)^\top \Sigma^{-1} (x-\mu) \right\} \tag{2}$$

という形で表される。

潜在変数は $z$。$K$ 次元ベクトル $z$ は、K-of-1 符号化がなされている。$z$ の確率分布は、$p(z_k = 1) = \pi_k$ となる。$z_k$ はどれか一つだけが 1 となるので、$p(\mathbf{z}) = p(z_1, \cdots, z_K) = \prod_{k=1}^{K} \pi_k^{z_k}$ が、1-of-K 表現の場合言えることに注意。

### 1.2 条件付き分布

$z$ が与えられた下での $x$ の条件付き分布は、

$$p(x|z_k = 1) = \mathcal{N}(x|\mu_k, \Sigma_k) \tag{3}$$

と表されるとする。このとき、$x$ の周辺分布は $z$ で周辺化して (以下の式の導出にはスライド[*2]も参考に)、

$$\begin{aligned}
p(\mathbf{x}) &= \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) && (4)\\
&= \sum_{\mathbf{z}} p(\mathbf{z}) p(\mathbf{x}|\mathbf{z}) && (5)\\
&= \sum_{\mathbf{z}} \prod_{k=1}^{K} \pi_k^{z_k} \prod_{k=1}^{K} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k} && (6)\\
&= \sum_{\mathbf{z}} \prod_{k=1}^{K} (\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k))^{z_k} && (7)
\end{aligned}$$

[*1] http://kivantium.hateblo.jp/entry/2015/08/17/235832

[*2] http://www.slideshare.net/takao-y/20131113-em

また、負担率 (データ $\mathbf{x}$ が与えられた下での $z_k = 1$ の確率)[*3]は以下のようになる (細かな導出はスライド参考のこと、PRML の 9.13 式)。

$$\gamma(z_k) \equiv p(z_k = 1|\mathbf{x}) = \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{8}$$

### 1.3　対数尤度関数と各パラメータ

これは、PRML や前述のスライドを確認

### 1.4　EM アルゴリズム

1. $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$, $\boldsymbol{\pi}_k$ を適当な値に初期化
2. E ステップ
   - 以下の式を現在のパラメータに基づいて計算する

$$\gamma(z_k) \equiv p(z_k = 1|\mathbf{x}) = \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

3. M ステップ
   - $N_k = \sum_{n=1}^{N} \gamma(z_{nk})$ として、新しい $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$, $\boldsymbol{\pi}_k$ を計算
4. 対数尤度が変化しなくなるか、一定の回数に達するまで E ステップと M ステップを交互に繰り返す

## 2　実装

### 2.1　正規分布

行列式が`.determinant()` で、$\det A$ や、$|A|$ と数式では表される。`M_PI` が $\pi$ のことなので注意。`.size()` が返すのは、”the number of coefficients, which is `rows()*cols()`”とのことだった。

```
double Norm(VectorXd x, VectorXd mu, MatrixXd sigma){ //Equation (2)
    return exp(((x-mu).transpose() * sigma.inverse() * (x-mu)/(-2.0)).value())
        / sqrt(sigma.determinant())
        / pow(2.0*M_PI, x.size()/2.0);
}
```

### 2.2　変数の宣言

C 言語において `const` 修飾子は、指定した変数が定数である（中身を変更 できない）ことを指定する。これによってバグの混入を防ぐことが出来る。

```
int main() {
    const int D = 2; // dimension
    const int K = 2; // number of distribution
    int N;           // number of data
```

[*3] $k$ 番目のガウス分布が $x$ を説明する度合いとしても考えられる

```
    vector<VectorXd> x;
```

### 2.3 データの読み込み

Stackoverflow[*4]の回答を参考にした。

```
#define MAXBUFSIZE  ((int) 1e6)
MatrixXd readMatrix(const char *filename)
    {
    int cols = 0, rows = 0;
    double buff[MAXBUFSIZE];

    // Read numbers from file into buffer.
    ifstream infile;
    infile.open(filename);
    while (! infile.eof())
        {
        string line;
        getline(infile, line);

        int temp_cols = 0;
        stringstream stream(line);
        while(! stream.eof())
            stream >> buff[cols*rows+temp_cols++];

        if (temp_cols == 0)
            continue;

        if (cols == 0)
            cols = temp_cols;

        rows++;
        }

    infile.close();

    rows--;

    // Populate matrix with numbers.
    MatrixXd result(rows,cols);
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            result(i,j) = buff[ cols*i+j ];

```

[*4] http://stackoverflow.com/questions/20786220/eigen-library-initialize-matrix-with-data-from-file-or-existing-stdvector

```
    return result;
};
```

## 2.4 推定

### 2.4.1 必要な変数の宣言

```
VectorXd mu[K]; // mean vector
MatrixXd sigma[K]; // covariance matrix
VectorXd pi[K]; // mixture
VectorXd N_k[K]; // effective cluster size
MatrixXd gamma(N,K); // responsibility
```