応用計量分析２（第6, 7, 8回）

混合分布モデルとEMアルゴリズム

担当教員: 梶野 洸（かじの ひろし）

概要

混合分布モデル

- 混合分布モデルの導入（混合ガウスモデル）
- EM アルゴリズムの導出

混合分布モデル

複数の分布を混合したモデル

$$p(X) = \sum_{z \in \mathcal{Z}} p(X \mid Z = z) p(Z = z)$$

- $p(X \mid Z)$: 複数の分布（$Z \in \mathcal{Z}$ごとに異なる分布）
- $p(Z)$: 混合割合
- $p(X)$: 最終的に得られる分布

例: 混合ガウスモデル

有限個のガウス分布を混合したモデル

$$p(X) = \sum_{k=1}^{K} p(X \mid Z = k) p(Z = k)$$

where

$$p(X \mid Z = k) = \mathcal{N}(X; \mu_k, \Sigma_k) \; (\mu_k \in \mathbb{R}^D, \Sigma_k \in \mathbb{S}_+^D)$$

$$p(Z = k) = \pi_k \; (k = 1, \ldots, K)$$

ここで $\sum_{k=1}^{K} \pi_k = 1$ が成り立つとする。

例: 混合ガウスモデル

有限個のガウス分布を混合したモデル

$$p(X) = \sum_{k=1}^{K} \pi_k \mathcal{N}(X; \mu_k, \Sigma_k)$$

と書くことが多い。

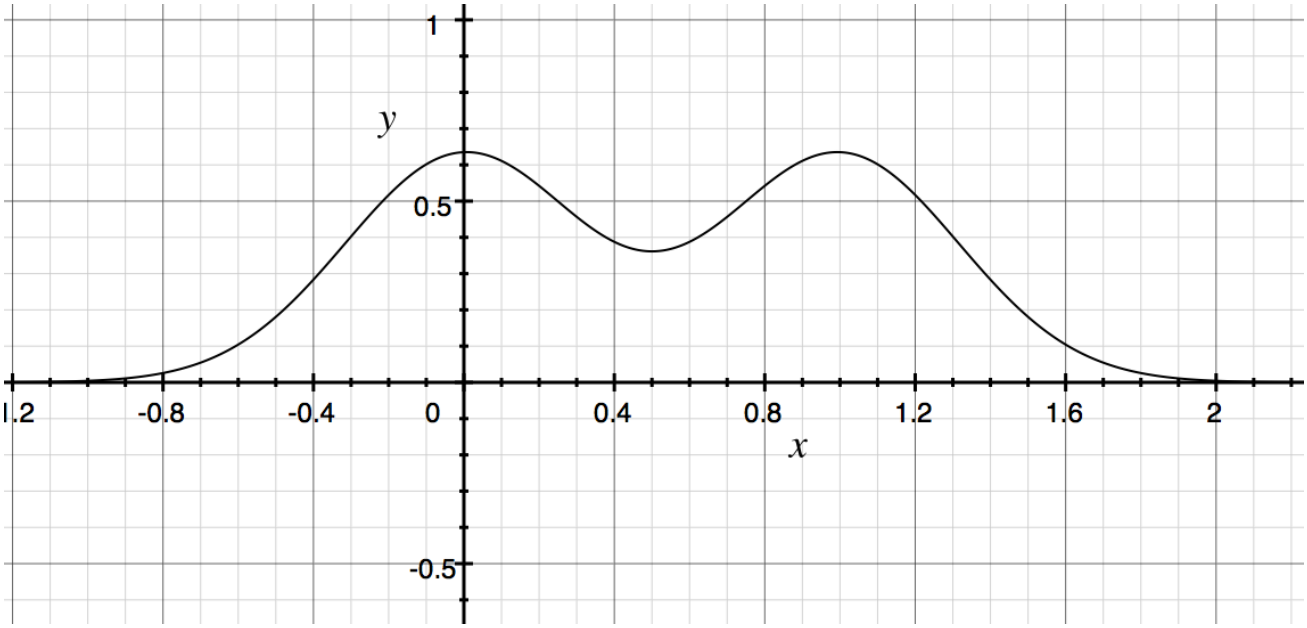
例: 混合ガウスモデル

混合ガウスモデルは $K+1$ 個の確率分布で構成される

- Categorical$(\pi), \pi \in \Delta^K$
  - カテゴリカル分布（K面サイコロ）
  - 各面 k が出る確率が π_k
- $\mathcal{N}(\mu_k, \Sigma_k)\,(k = 1, \ldots, K)$: K個の異なる正規分布
  - 平均 μ_k, 分散共分散行列 Σ_k

例: 混合ガウスモデル

2つの正規分布からなる GMM

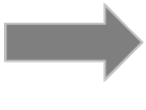


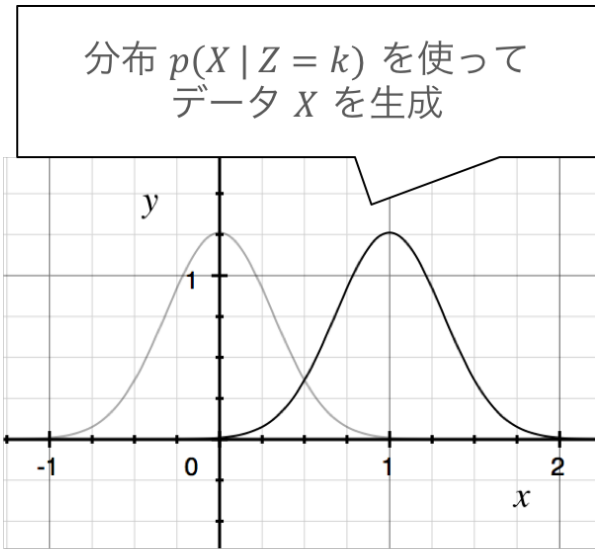
生成モデルとしての説明

確率変数 X を生成する過程としても解釈できる

- K 面サイコロで k の目が出る確率: $p(Z = k)$
- k の目が出た条件のもとで確率変数 X が従う確率分布: $p(X \mid Z = k)$
- (Z, X) の従う確率分布: $p(X, Z) = p(X \mid Z)p(Z)$
- X の従う確率分布だけ見ると $p(x) = \sum_{k=1}^{K} p(x \mid Z = k)p(Z = k)$

K面さいころを振って
$Z = k$ を得る





生成モデルとしての説明

数式では以下のように書く

$$
\begin{aligned}
Z &\sim \text{Categorical}(\pi) \\
X &\sim \mathcal{N}(\pi_Z, \Sigma_Z)
\end{aligned}
$$

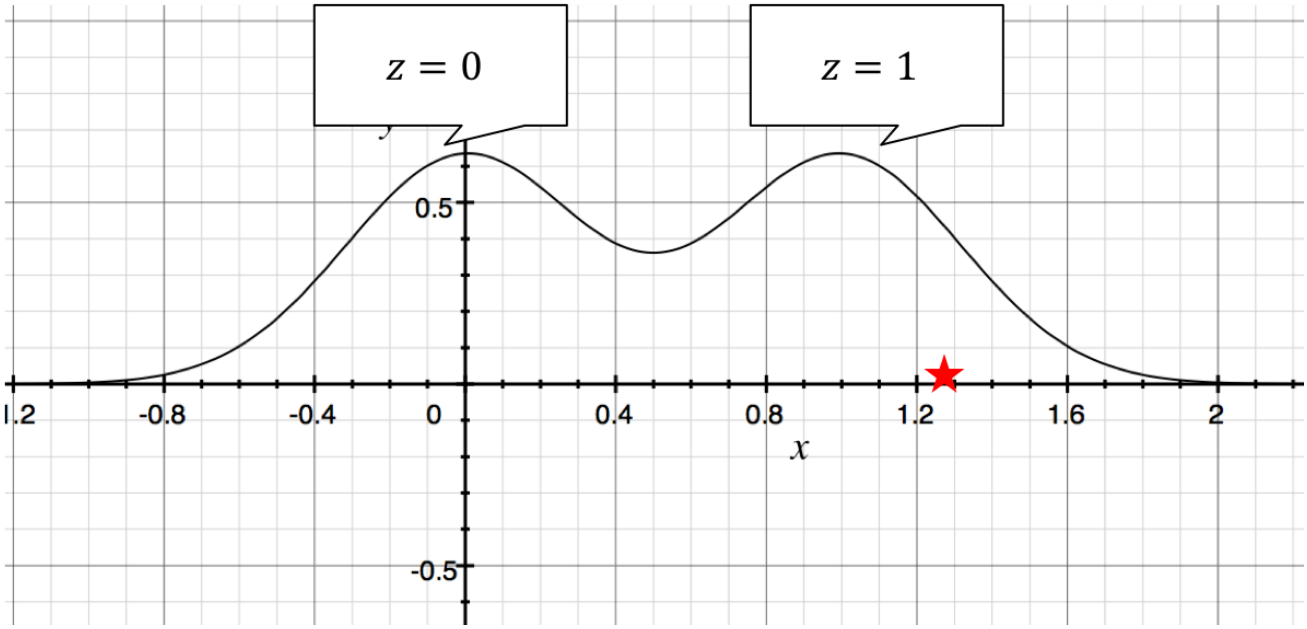
- 一行目でK面サイコロを振って Z の実現値（kとする）を得る
- 二行目で $Z = k$ に対応する正規分布から X の実現値を得る

混合モデルを考える理由

- 既知の確率分布を組み合わせて複雑な確率分布を表現できる
    - 正規分布単体では単峰の分布しか表現できない
    - 正規分布を重ね合わせると複数の山を表現できる

- 潜在変数Zからの知識発見
    - 事後分布 $p(Z \mid X = x)$ は、データ x がどの分布から出てきたのかを表す
    - クラスタリングのような効果

★のデータ X は $Z = 1$ から生成されたっぽい



ここまでのまとめ

- 混合分布モデルを使うと複雑な確率分布も表現できる
- 正規分布を有限個混合したモデルとして、混合ガウスモデルがある
- データが生成される過程として書くことができる
- クラスタリングのような使い方ができる

混合ガウスモデルの推定

- ここまでは確率モデルが与えられたときにそれをどう解釈するかを議論していた
- データセット $\mathcal{D} = \{x_1, \ldots, x_N\}$ が与えられた時、そのデータセットが従う GMM を推定するにはどうしたら良いか？

混合ガウスモデルの自由度

- K個の正規分布の平均ベクトル、共分散行列
- どの正規分布を選ぶか決める確率分布 $\pi_k = p(Z = k)\,(k = 1, \ldots, K)$
- 混合数K

最尤推定で求まるもの

混合数は求まらない

- K個の正規分布の平均ベクトル、共分散行列
- どの正規分布を選ぶか決める確率分布 $\pi_k = p(Z = k)\,(k = 1, \ldots, K)$

最尤推定

$$\theta^\star = \arg\max_{\theta \in \Theta} p(\mathcal{D}; \theta)$$

をパラメタの推定値とする

各データの独立同分布性を仮定すると

$$\log p(\mathcal{D};\theta) = \sum_{n=1}^{N} \log p(x_n;\theta)$$

モデルの仮定では、 $p(x;\theta) = \sum_{z=1}^{K} p(x \mid z;\theta)p(z;\theta)$ だったので、上式は、

$$\sum_{n=1}^{N} \log\left(\sum_{z_n=1}^{K} p(x_n \mid z_n;\theta)p(z_n;\theta) \right) \tag{1}$$

となる。

最尤推定量は式(1)を最大化することで求まる。

最大化の手法は二種類

1. 勾配法で式(1)を直接最大化する
2. EMアルゴリズムで最大化する（今回はこっち！）

Expectation-maximization (EM) アルゴリズム

隠れ変数があるモデルの最尤推定量を求めるアルゴリズム

- 現状は各データの隠れ変数 z_n もパラメタ θ もわからない

    1. z_n が1つに決まれば θ がわかる
    2. θ が1つに決まれば z_n がわかる

- とりあえずどちらかを初期化して、1, 2 を繰り返せばいい感じの z_n, θ が求まりそう
    - 局所最適解が求まることが示せる
    - （大域的最適解を求めるのはそもそも難しいので諦めている）

EM アルゴリズムの利点

- EステップとMステップが解析的に書ける
    - 毎ステップで尤度が下がらないことが保証される
    - 勾配法だと、ステップサイズによって尤度が下がることもある
- より難しいモデルの推定にも同じようなアイデアが適用できる

演習

1. X_n の実現値だけでなく Z_n の実現値も得られていた場合の最尤推定量 $\theta^\star$ を計算せよ
    - (x_n, z_n) が $p(x, z; \theta) = \pi_z \mathcal{N}(x; \mu_z, \Sigma_z)$ という同時分布に従っている
    - $-\sum_{n=1}^{N} \log p(x_n, z_n; \theta)$ を θ について最小化する
2. X_n の実現値だけでなく、パラメタ$\theta^\star$ の値もわかっている場合の隠れ変数の事後分布を計算せよ
    - $p(Z_n \mid X_n = x_n; \theta^\star)$ を計算する

答え

1. $\{(x_n, z_n)\}_{n=1}^N$ が得られていた場合、x_n, z_n の同時分布は、

$$p(x_n, z_n; \theta) = \pi_{z_n} \mathcal{N}(x_n; \mu_{z_n}, \Sigma_{z_n})$$

$$\begin{aligned}
&- \sum_{n=1}^{N} \log p(x_n, z_n; \theta) \\
&= - \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbb{1}[z_n = k] \left(\log \pi_k + \log \mathcal{N}(x_n; \mu_k, \Sigma_k) \right) \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbb{1}[z_n = k] \left(- \log \pi_k + \frac{1}{2} (x_n - \mu_k)^\top \Sigma_k^{-1} (x_n - \mu_k) \right. \\
&\qquad \left. + \frac{D}{2} \log 2\pi + \frac{1}{2} \log \det \Sigma_k \right) \qquad (2)
\end{aligned}$$

μ_k についての最大化

式(2)をμ_kで微分すると

$$\sum_{n=1}^{N} \mathbb{1}[z_n = k]\Sigma_k^{-1}(x_n - \mu_k)$$

なので、μ_kの最尤推定量は

$$\mu_k^{\star} = \frac{\sum_{n=1}^{N} \mathbb{1}[z_n = k]x_n}{\sum_{n=1}^{N} \mathbb{1}[z_n = k]}$$

となる。

Σ_k についての最大化

式(2)をΣ_kで微分すると

$$\frac{1}{2}\sum_{n=1}^{N}\mathbb{1}[z_n = k]\left(-\Sigma_k^{-1}(x_n-\mu_k)(x_n-\mu_k)^\top\Sigma_k^{-1}+\Sigma_k^{-1}\right)$$

なので、Σ_kの最尤推定量は

$$\Sigma_k^\star = \frac{\sum_{n=1}^{N}\mathbb{1}[z_n = k](x_n-\mu_k^\star)(x_n-\mu_k^\star)^\top}{\sum_{n=1}^{N}\mathbb{1}[z_n = k]}$$

となる

π_k についての最大化

π_k は $\sum_{k=1}^{K} \pi_k = 1$ を満たす必要があるため、式(2)に $\lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right)$ を足したものを π_k で微分して

$$\sum_{n=1}^{N} \mathbb{1}[z_n = k] \left(-\frac{1}{\pi_k^\star} \right) + \lambda = 0$$

を解けば良い。

$$\lambda \pi_k^\star = \sum_{n=1}^{N} \mathbb{1}[z_n = k]$$

を $k = 1, \ldots, K$ で足すと

$$\lambda = N.$$

よって、

$$\pi_k^\star = \frac{\sum_{n=1}^{N} \mathbb{1}[z_n = k]}{N}$$

(2) $p(Z \mid X; \theta) \propto p(X \mid Z; \theta)p(Z; \theta)$ なので、

$$p(z_n \mid X_n = x_n; \theta^\star) = \frac{\pi^\star_{z_n} \mathcal{N}(x_n; \mu^\star_{z_n}, \Sigma^\star_{z_n})}{\sum_{k=1}^{K} \pi^\star_k \mathcal{N}(x_n; \mu^\star_k, \Sigma^\star_k)}$$

演習まとめ

z_n が1つに決まっていると、

$$\sum_{n=1}^{N} \log\left(\sum_{z_n=1}^{K} p(x_n \mid z_n; \theta) p(z_n; \theta) \right)$$

の log のなかの足し算がなくなって、

$$\sum_{n=1}^{N} (\log p(x_n \mid z_n; \theta) + \log p(z_n; \theta))$$

の最大化となるため、解析的に解ける。

EM アルゴリズムの導出

任意の分布 $q(z)$ について

$$
\begin{aligned}
&\log\left(\sum_z p(x \mid z;\theta)p(z;\theta)\right)\\
=&\log\left(\sum_z p(x \mid z;\theta)\frac{p(z;\theta)}{q(z)}q(z)\right)\\
\geq&\sum_z q(z)\log\left(p(x \mid z;\theta)\frac{p(z;\theta)}{q(z)}\right)\\
=&\sum_z q(z)\left(\log p(x,z;\theta)-\log q(z)\right)
\end{aligned}
$$

が成り立つことを利用する。

証明

- $f: \mathbb{R}^D \to \mathbb{R}$: 任意の凸関数
- $x_1, \ldots, x_N \in \mathbb{R}^D$: f の定義域の任意のN点
- $w_1, \ldots, w_N$ s.t. $\sum_{n=1}^{N} w_n = 1$: 任意の重み $\in [0, 1]$

とする。

$$f\left(\sum_{n=1}^{N} w_n x_n\right) \leq \sum_{n=1}^{N} w_n f\left(x_n\right)$$

を示せ (Jensenの不等式)。

ヒント

$f : \mathbb{R}^D \to \mathbb{R}$ が凸関数 $\Longleftrightarrow$ 任意の$x_1, x_2 \in \mathbb{R}^D, w_1, w_2 \in [0, 1]$ s.t. $w_1 + w_2 = 1$ について

$$f(w_1 x_1 + w_2 x_2) \leq w_1 f(x_1) + w_2 f(x_2)$$

N の時に成り立っているとして、$N+1$の時にも成り立つことを示す。

$$
\begin{aligned}
& f\left(\sum_{n=1}^{N+1} w_n x_n\right) \\
=& f\left(w_{N+1}x_{N+1} + \sum_{n=1}^{N} w_n x_n\right) \\
=& f\left(w_{N+1}x_{N+1} + (1-w_{N+1})\sum_{n=1}^{N} w_n \frac{x_n}{1-w_{N+1}}\right) \\
\leq& w_{N+1} f(x_{N+1}) + (1-w_{N+1}) f\left(\sum_{n=1}^{N} \frac{w_n}{1-w_{N+1}} x_n\right)
\end{aligned}
$$

$$
\begin{aligned}
&\leq && w_{N+1} f(x_{N+1}) + (1 - w_{N+1}) \\
&\quad \sum_{n=1}^{N} \frac{w_n}{1 - w_{N+1}} f(x_n) && \left(\text{note that } \sum_{n=1}^{N} \frac{w_n}{1 - w_{N+1}} = 1 \right) \\
&= \sum_{n=1}^{N+1} w_n f(x_n)
\end{aligned}
$$

再掲

$\log x$ は凹関数なので、任意の分布 $q(z)$ について

$$
\begin{aligned}
&\log\left(\sum_z p(x \mid z;\theta)p(z;\theta)\right)\\
=&\log\left(\sum_z p(x \mid z;\theta)\frac{p(z;\theta)}{q(z)}q(z)\right)\\
\geq&\sum_z q(z)\log\left(p(x \mid z;\theta)\frac{p(z;\theta)}{q(z)}\right)\\
=&\sum_z q(z)\left(\log p(x,z;\theta)-\log q(z)\right) \qquad (3)
\end{aligned}
$$

が成り立つ。

EMアルゴリズムの導出

尤度（式(3)の左側）を最大化する代わりに尤度の下界（式(3)の右側）を最大化する:

$$\sum_{n=1}^{N} \sum_{z_n} q(z_n)(\log p(x_n, z_n; \theta) - \log q(z_n))$$

- q に関する最大化
- θ に関する最大化

q を固定して θ に関して最大化

$$
\begin{aligned}
& p(x_n, z_n; \theta) = \pi_{z_n} \mathcal{N}(x_n; \mu_{z_n}, \Sigma_{z_n}) \\
& - \sum_{n=1}^{N} \sum_{k=1}^{K} q(z_n = k) \log p(x_n, z_n = k; \theta) \\
= & - \sum_{n=1}^{N} \sum_{k=1}^{K} q(z_n = k) \left(\log \pi_k + \log \mathcal{N}(x_n; \mu_k, \Sigma_k) \right) \\
= & \sum_{n=1}^{N} \sum_{k=1}^{K} q(z_n = k) \left(- \log \pi_k + \frac{1}{2} (x_n - \mu_k)^\top \Sigma_k^{-1} (x_n - \mu_k) \right. \\
& \qquad \left. + \frac{D}{2} \log 2\pi + \frac{1}{2} \log \det \Sigma_k \right)
\end{aligned}
$$

式(2)で $\mathbb{1}[z_n = k]$ を $q(z_n = k)$ に変えたやつだ！

よって

$$\mu_k^\star = \frac{\sum_{n=1}^N q(z_n = k) x_n}{\sum_{n=1}^N q(z_n = k)}$$

$$\Sigma_k^\star = \frac{\sum_{n=1}^N q(z_n = k)(x_n - \mu_k^\star)(x_n - \mu_k^\star)^\top}{\sum_{n=1}^N q(z_n = k)}$$

$$\pi_k^\star = \frac{\sum_{n=1}^N q(z_n = k)}{N}$$

θ を固定して q に関して最大化

q は $\sum_{k=1}^{K} q(z_n = k) = 1$ $(n = 1, \dots, N)$ を満たす $N \times K$ 個の実数からなる。

$$
\begin{aligned}
\frac{\partial}{\partial q(z_n = k)} \Bigg(& \sum_{n=1}^{N} \sum_{k=1}^{K} q(z_n = k) \\
& \times (\log p(x_n, z_n = k; \theta) - \log q(z_n = k)) \\
& + \sum_{n=1}^{N} \lambda_n \left(\sum_{k=1}^{K} q(z_n = k) - 1 \right) \Bigg) \\
= & \log p(x_n, z_n = k; \theta) - \log q(z_n = k) - 1 + \lambda_n = 0
\end{aligned}
$$

を解くと

$$
\begin{aligned}
q(z_n = k) &= \frac{p(x_n, z_n = k; \theta)}{\sum_{k=1}^{K} p(x_n, z_n = k; \theta)} \\
&= \frac{\pi_k \mathcal{N}(x_n; \mu_k, \Sigma_k)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n; \mu_k, \Sigma_k)}
\end{aligned}
$$

EM アルゴリズム

- μ_k, Σ_k, π_k $(k = 1, \dots, K)$ を適当に初期化
- 以下を繰り返す:
  - E-step: for each $n = 1, \dots, N, k = 1, \dots, K$,

$$q(z_n = k) \leftarrow \frac{\pi_k \mathcal{N}(x_n; \mu_k, \Sigma_k)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n; \mu_k, \Sigma_k)}$$

  - M-step: for each $k = 1, \dots, K$,

$$\mu_k \leftarrow \frac{\sum_{n=1}^{N} q(z_n = k) x_n}{\sum_{n=1}^{N} q(z_n = k)}$$

$$\Sigma_k \leftarrow \frac{\sum_{n=1}^{N} q(z_n = k)(x_n - \mu_k)(x_n - \mu_k)^\top}{\sum_{n=1}^{N} q(z_n = k)}$$

$$\pi_k \leftarrow \frac{\sum_{n=1}^{N} q(z_n = k)}{N}$$

適当な終了条件で繰り返しをやめる

- 尤度の増分が一定以下になった
- パラメタの変化幅が一定以下になった

EMアルゴリズムまとめ

- 変分法を用いて尤度の下界を求めた
  - $\sum_{n=1}^{N} \sum_{z_n} q(z_n) \log p(x_n, z_n; \theta)$ みたいな形を出したかった
  - q と θ を交互に最適化するとそれぞれ解析的に解ける

EMアルゴリズムの実装

演習

混合ガウスモデルの場合、

- 内部状態
- 欲しい命令

は何か？

一つの答え

- 内部状態
  - 正規分布の平均、分散共分散行列$\times K$個
  - 重み$\pi = \begin{bmatrix} \pi_1 & \dots & \pi_K \end{bmatrix}$
- 命令
  - データを入力して、パラメタを最尤推定する
  - データを入力して、各データの潜在変数を推定する（現状の内部状態を使って）

他の答え

補助的なものを持っていてもよい

- 内部状態
  - 正規分布の平均、分散共分散行列$\times \boldsymbol{K}$個
  - 重み$\pi = \begin{bmatrix} \pi_1 & \dots & \pi_K \end{bmatrix}$
  - **混合数$\boldsymbol{K}$を陽に持っておくと何かと便利**
- 命令
  - データを入力して、パラメタを最尤推定する
    - **E ステップを実行する**
    - **M ステップを実行する**
    - **尤度を計算する**
  - データを入力して、各データの潜在変数を推定する（現状の内部状態を使って）

他の答え2

内部状態として他のクラスのオブジェクトを持っていてもよい

- 正規分布クラス
    - 内部状態
        - 平均
        - 分散共分散行列
    - 命令
        - データを入力して、パラメタを最尤推定する
        - データを入力して、尤度を計算する
- GMM クラス
    - 内部状態
        - 正規分布オブジェクト $\times \boldsymbol{K}$個
        - 重み $\pi = \begin{bmatrix} \pi_1 & \dots & \pi_K \end{bmatrix}$
        - **混合数 $\boldsymbol{K}$ を陽に持っておくと何かと便利**
    - 命令
        - データを入力して、パラメタを最尤推定する
            - **E ステップを実行する**
            - **M ステップを実行する**
            - **尤度を計算する**
        - データを入力して、各データの潜在変数を推定する（現状の内部状態を使って）

Python での実装

まずは正規分布クラスを実装してみよう

課題

正規分布クラスを実装せよ（以前の資料からコピペでOK）

In [3]:

```
import numpy as np

class Gaussian:
    def __init__(self, dim):
        '''コンストラクタ（みたいなもの）
        オブジェクトを作るときに初めに実行される。
        内部状態の初期化に使う
        '''
        self.dim = dim
        '''
        self = オブジェクトを指す。 self.dim は、オブジェクトの dim という変数を指す。
        上の命令は、 self.dim に dim の値を代入することを表す
        '''
        self.set_mean(np.random.randn(dim)) # オブジェクトの mean という変数をランダムに初期化
        self.set_cov(np.identity(dim))

    def log_pdf(self, X):
        ''' 確率密度関数の対数を返す

        Parameters
        ----------
        X : numpy.array, shape (sample_size, dim)

        Returns
        -------
        log_pdf : array, shape (sample_size,)
        '''
        if X.shape[1] != self.dim: # 入力の形をチェックしています
            raise ValueError('X.shape must be (sample_size, dim)')
        return -0.5 * np.sum((X - self.mean) * (np.linalg.solve(self.cov, (X - self.mean).T).T), axis=1) \
            -0.5 * self.dim * np.log(2.0 * np.pi) - 0.5 * np.linalg.slogdet(self.cov)[1]

    def fit(self, X):
        ''' X を使って最尤推定をする

        Parameters
        ----------
        X : numpy.array, shape (sample_size, dim)
        '''
        if X.shape[1] != self.dim: # 入力の形をチェックしています
            raise ValueError('X.shape must be (sample_size, dim)')
        self.set_mean(np.mean(X, axis=0))
        self.set_cov((X - self.mean).T @ (X - self.mean) / X.shape[0])

    def sample(self, sample_size):
        ''' 現状のパラメタを使って `sample_size` のサイズのサンプルを生成する

        Parameters
        ----------
        sample_size : int

        Returns
        -----------
        X : numpy.array, shape (sample_size, dim)
            各行は平均 `self.mean`, 分散 `self.cov` の正規分布に従う
        '''
        return np.random.multivariate_normal(self.mean, self.cov, size=sample_size)

    def set_mean(self, mean):
        if mean.shape != (self.dim,):
            raise ValueError('input shape inconsistency')
        self.mean = mean

    def set_cov(self, cov):
        if cov.shape != (self.dim, self.dim):
            raise ValueError('input shape inconsistency')
```

```
        if np.linalg.eigvalsh(cov)[0] <= 0:
            raise ValueError('covariance matrix must be positive semidefinite.')
        self.cov = cov
```

課題

- GMM クラスの `__init__` を書いてみよう

In [4]:

```
class GMM:
    def __init__(self, dim, num_components):
        self.dim = dim
        self.num_components = num_components

        self.weight = np.ones(self.num_components) / self.num_components
        self.gaussian_list = []
        for _ in range(self.num_components):
            self.gaussian_list.append(Gaussian(dim))
```

In [5]:
```
# gmm のオブジェクトができた
gmm = GMM(2, 10)
for each_gaussian in gmm.gaussian_list:
    print(each_gaussian.mean)
print(gmm.weight)
```

```
[ 0.11684931 -1.71658838]
[-1.10708776  0.25183766]
[-1.24486871  0.55898262]
[ 1.77706673 -1.59593316]
[ 1.5407428  -0.58973061]
[-1.81424689  0.16981966]
[-0.12951037  0.52775457]
[ 0.24461285 -0.3235077 ]
[1.2561859  0.75503971]
[ 0.15138818 -0.42148386]
[0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1]
```

以降、EMアルゴリズムでパラメタ推定をする `fit` を書きたい

- `log_pdf`: 現在のパラメタを用いてサンプルの対数尤度を計算するメソッド
- `e_step`: Eステップを実行するメソッド
- `m_step`: Mステップを実行するメソッド

の3つを実装し、これらを組み合わせて `fit` を書く

課題

- サンプルの対数尤度を計算するメソッド `log_pdf` を `GMM` に実装せよ
  - 引数: `x`: サンプルサイズ×次元の行列
  - 返り値: $\begin{bmatrix} \log p(x_1;\theta) & \dots & \log p(x_N;\theta) \end{bmatrix}$ (θは現在の内部状態を用いる)

$$p(x_n;\theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x_n;\mu_k,\Sigma_k)$$

In [6]:

```
class GMM:
    def __init__(self, dim, num_components):
        self. dim = dim
        self.num_components = num_components
        self.gaussian_list = []
        self.weight = np.ones(self.num_components) / self.num_components
        for _ in range(self.num_components):
            self.gaussian_list.append(Gaussian(dim))

    def log_pdf(self, X):
        sample_size = X.shape[0]
        likelihood = np.zeros(sample_size) # 各データの尤度を計算する
        '''
        ここで現在の内部状態での likelihood を計算する

        ** ヒント **
        self.gaussian_list の各要素は Gaussian クラスのオブジェクト
        各データの対数尤度を計算する log_pdf という命令が実装してあった
        '''
        return np.log(likelihood) # 対数尤度が欲しいので対数をとる
```

ヒント

- `self.gaussian_list[0].log_pdf(X)` はサンプル `X` のデータそれぞれに対する対数尤度を計算する
    - `X` が `sample_size x dim` とすると、返り値は長さ `sample_size` の配列

- `self.weight[0]` は $p(z = 0; \theta)$ に対応する
- `self.weight[0] * np.exp(self.gaussian_list[0].log_pdf(X))` は `np.exp(self.gaussian_list[0].log_pdf(X))` の各要素に `self.weight[0]` を掛けた値を返す
    - スカラー `*` 配列 と書くと、配列の各要素にスカラーを掛けてくれる（numpy の記法）

In [7]:

```
class GMM:
    def __init__(self, dim, num_components):
        self. dim = dim
        self.num_components = num_components
        self.gaussian_list = []
        self.weight = np.ones(self.num_components) / self.num_components
        for _ in range(self.num_components):
            self.gaussian_list.append(Gaussian(dim))

    def log_pdf(self, X):
        sample_size = X.shape[0]
        likelihood = np.zeros(sample_size) # 各データの尤度を計算する
        for each_component in range(self.num_components):
            likelihood = likelihood + self.weight[each_component] * np.exp(self.ga
ussian_list[each_component].log_pdf(X))
        return np.log(likelihood)
```

In [16]:
```
gmm = GMM(2, 10)
gmm.log_pdf(np.random.randn(100, 2)) # とりあえず何か計算はできている
```

Out[16]:
```
array([-3.12648963, -3.86705701, -2.57161702, -3.06551493, -2.65614086,
       -3.0071172 , -4.62457074, -2.88583395, -2.95423388, -2.93725819,
       -2.66778609, -2.91656884, -3.43291286, -2.61947626, -3.44059698,
       -3.25383126, -2.84392495, -2.79003912, -2.66291126, -2.99469256,
       -4.03689065, -2.83441528, -2.75630803, -2.67636738, -3.75517664,
       -2.64458188, -2.79363847, -2.87221519, -3.10244486, -2.69670577,
       -3.02748979, -2.64780433, -2.63585142, -3.09722914, -2.58475377,
       -2.73534958, -3.05715522, -2.72387022, -4.5341286 , -3.08127239,
       -3.21928694, -3.54947664, -3.14211745, -2.74879231, -3.44220236,
       -3.49531004, -4.53836991, -3.18854928, -3.53895284, -2.55987735,
       -3.11221332, -2.57703954, -3.19613213, -3.35386561, -2.7605182 ,
       -3.27197597, -2.80149132, -3.1221321 , -4.38693956, -3.09966228,
       -4.89665286, -2.68778129, -3.14177503, -2.7074782 , -3.16332312,
       -2.66797893, -2.89198018, -3.92306412, -3.82068393, -3.72077715,
       -3.35130862, -2.74938948, -3.1884004 , -3.23102166, -3.44943404,
       -2.59168975, -2.55253307, -3.3592686 , -4.24582831, -3.83563231,
       -3.23305835, -3.40850046, -3.27423081, -2.89178352, -3.01971239,
       -2.66453107, -2.55463812, -2.71222266, -3.1569365 , -2.73207968,
       -3.70887009, -3.55214124, -3.24868335, -2.5982095 , -3.72506253,
       -2.99258731, -3.05604243, -3.21803488, -3.46556688, -2.61541135])
```

課題

- Eステップを実行するメソッド `e_step` を実装せよ
    - 引数: データ `X`: `sample_size` × `dim` の配列
    - 出力: $q(z_n)$
        - どのような形式で出力すべきか？

- $q(z_n)$ は K 面サイコロだったので、K面サイコロの確率分布をデータの数だけ出力すれば良さそう
    - K = `n_components`
- `posterior`: `sample_size` × `n_components` の配列として出力
- `posterior[n, k]` = $q(z_n = k)$

E-step: for each $n = 1, \ldots, N, k = 1, \ldots, K$,

$$q(z_n = k) \leftarrow \frac{\pi_k \mathcal{N}(x_n; \mu_k, \Sigma_k)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n; \mu_k, \Sigma_k)}$$

In [9]:

```
class GMM:
    def __init__(self, dim, num_components):
        self.dim = dim
        self.num_components = num_components
        self.gaussian_list = []
        self.weight = np.ones(self.num_components) / self.num_components
        for _ in range(self.num_components):
            self.gaussian_list.append(Gaussian(dim))

    def log_pdf(self, X):
        sample_size = X.shape[0]
        likelihood = np.zeros(sample_size) # 各データの尤度を計算する
        for each_component in range(self.num_components):
            likelihood = likelihood + self.weight[each_component] * np.exp(self.ga
ussian_list[each_component].log_pdf(X))
        return np.log(likelihood)

    def e_step(self, X):
        sample_size = X.shape[0]
        posterior = np.zeros((sample_size, self.num_components))
        '''
        ここを埋める

        ** ヒント **
        各 n について、分母は共通
        1. 分子を先に計算して posterior に入れてしまう
        2. posterior[n, :].sum() は分母と等しくなる
        3. posterior[n, :] = posterior[n, :] / posterior[n, :].sum() とすれば良さそ
う
        '''
        return posterior
```

In [17]:

```
class GMM:
    def __init__(self, dim, num_components):
        self.dim = dim
        self.num_components = num_components
        self.gaussian_list = []
        self.weight = np.ones(self.num_components) / self.num_components
        for _ in range(self.num_components):
            self.gaussian_list.append(Gaussian(dim))

    def log_pdf(self, X):
        sample_size = X.shape[0]
        likelihood = np.zeros(sample_size) # 各データの尤度を計算する
        for each_component in range(self.num_components):
            likelihood = likelihood + self.weight[each_component] * np.exp(self.ga
ussian_list[each_component].log_pdf(X))
        return np.log(likelihood)

    def e_step(self, X):
        sample_size = X.shape[0]
        posterior = np.zeros((sample_size, self.num_components))
        # 各コンポーネントで対数尤度が計算できるので、それを利用
        for each_component in range(self.num_components):
            posterior[:, each_component] \
               = self.weight[each_component] * np.exp(self.gaussian_list[each_comp
onent].log_pdf(X))

        # まとめて正規化しているが、 for 文でやってもOK（遅くなるけど）
        posterior = posterior / posterior.sum(axis=1).reshape(-1, 1)
        return posterior
```

In [18]:
```
# q はデータがどちらのコンポーネントに近いかを表しているので、
# そんな感じの結果になって欲しい
gmm = GMM(2, 2)
X = gmm.gaussian_list[0].mean.reshape(1, 2)
print(gmm.e_step(X))
X = gmm.gaussian_list[1].mean.reshape(1, 2)
print(gmm.e_step(X))
```

```
[[0.86539486 0.13460514]]
[[0.13460514 0.86539486]]
```

課題

- Mステップを実行するメソッド `m_step` を実装せよ
    - 引数:
        - データ `X`: `sample_size` × `dim` の配列
        - $q(z_n = k)$: `sample_size` × `n_components`
    - 出力: なし

In [19]:

```
class GMM:
    def __init__(self, dim, num_components):
        self.dim = dim
        self.num_components = num_components
        self.gaussian_list = []
        self.weight = np.ones(self.num_components) / self.num_components
        for _ in range(self.num_components):
            self.gaussian_list.append(Gaussian(dim))

    def fit(self, X, eps=1e-8):
        sample_size = X.shape[0]
        converge = False
        old_ll = -np.inf
        new_ll = -np.inf
        while not converge:
            posterior = self.e_step(X)
            self.m_step(X, posterior)
            new_ll = self.log_pdf(X).sum()
            if new_ll < old_ll:
                raise ValueError('likelihood decreases!')
            if np.abs(old_ll - new_ll) / np.abs(new_ll) < eps:
                converge = True
            old_ll = new_ll
        return posterior

    def e_step(self, X):
        posterior = np.zeros((X.shape[0], self.num_components))
        for each_component in range(self.num_components):
            posterior[:, each_component] \
               = self.weight[each_component] * np.exp(self.gaussian_list[each_comp
onent].log_pdf(X))
        posterior = posterior / posterior.sum(axis=1).reshape(-1, 1)
        return posterior

    def m_step(self, X, posterior):
        self.weight = posterior.sum(axis=0)
        for each_component in range(self.num_components):
            self.gaussian_list[each_component].set_mean(posterior[:, each_componen
t] @ X / self.weight[each_component])
            self.gaussian_list[each_component].set_cov(
                (posterior[:, each_component] * (X - self.gaussian_list[each_compo
nent].mean).T) \
                @ (X - self.gaussian_list[each_component].mean) / self.weight[each
_component])
        self.weight = self.weight / np.sum(self.weight)

    def log_pdf(self, X):
        sample_size = X.shape[0]
        likelihood = np.zeros(sample_size) # 各データの尤度を計算する
        for each_component in range(self.num_components):
            likelihood = likelihood + self.weight[each_component] * np.exp(self.ga
ussian_list[each_component].log_pdf(X))
        return np.log(likelihood)
```

In [20]:
```
X = np.vstack((np.random.randn(100, 3), np.random.randn(100, 3) + 3))
gmm = GMM(3, 2)
posterior = gmm.fit(X)
```

In [14]:
```
print(gmm.gaussian_list[0].mean)
print(gmm.gaussian_list[1].mean)
```

```
[-0.15634628  0.0826993   0.04664785]
[2.99283013 3.03949123 2.96590465]
```

In [15]:
```
posterior
```

Out[15]:
```
array([[9.99999972e-01, 2.80218650e-08],
       [9.97371053e-01, 2.62894687e-03],
       [9.99999983e-01, 1.65791801e-08],
       [9.99999198e-01, 8.01597957e-07],
       [9.97733881e-01, 2.26611868e-03],
       [9.99476715e-01, 5.23284529e-04],
       [1.00000000e+00, 5.67816857e-11],
       [9.99999944e-01, 5.63259856e-08],
       [9.99999202e-01, 7.98373347e-07],
       [9.99999991e-01, 8.98339412e-09],
       [9.99999993e-01, 7.14412263e-09],
       [9.99999999e-01, 7.20960106e-10],
       [9.99999990e-01, 9.79199031e-09],
       [9.99999938e-01, 6.21605991e-08],
       [9.99999869e-01, 1.30754432e-07],
       [9.99999999e-01, 7.47173169e-10],
       [9.99985013e-01, 1.49867177e-05],
       [9.99999998e-01, 1.76905802e-09],
       [9.99999962e-01, 3.82747890e-08],
       [1.00000000e+00, 6.07185145e-13],
       [9.99999127e-01, 8.72579183e-07],
       [9.99913754e-01, 8.62462203e-05],
       [9.99999913e-01, 8.65319842e-08],
       [9.99999983e-01, 1.73070999e-08],
       [9.99999986e-01, 1.39271511e-08],
       [9.99999892e-01, 1.07590678e-07],
       [9.88421571e-01, 1.15784293e-02],
       [9.99999964e-01, 3.59867788e-08],
       [9.99999663e-01, 3.37265939e-07],
       [9.99999985e-01, 1.49309649e-08],
       [9.99999572e-01, 4.27887629e-07],
       [9.59783355e-01, 4.02166454e-02],
       [9.97050613e-01, 2.94938667e-03],
       [9.99999961e-01, 3.91009276e-08],
       [1.00000000e+00, 9.10492798e-13],
       [9.99999878e-01, 1.22356081e-07],
       [9.99965215e-01, 3.47851055e-05],
       [1.00000000e+00, 7.48456732e-11],
       [9.99993642e-01, 6.35809155e-06],
       [9.99999986e-01, 1.39305840e-08],
       [9.99994125e-01, 5.87501848e-06],
       [1.00000000e+00, 4.47041114e-12],
       [9.99987397e-01, 1.26034971e-05],
       [1.00000000e+00, 1.11335739e-12],
       [9.99998851e-01, 1.14868081e-06],
       [9.99999993e-01, 6.87674223e-09],
       [9.99999998e-01, 2.02409122e-09],
       [9.99998894e-01, 1.10599962e-06],
       [9.99996332e-01, 3.66762392e-06],
       [9.99999995e-01, 5.14285315e-09],
       [9.99999997e-01, 3.22461769e-09],
       [9.99999402e-01, 5.97516589e-07],
       [9.99989099e-01, 1.09005786e-05],
       [9.99971098e-01, 2.89020294e-05],
       [9.99896801e-01, 1.03198835e-04],
       [9.99999920e-01, 8.00432335e-08],
       [9.99999660e-01, 3.39700553e-07],
       [9.99619582e-01, 3.80418356e-04],
       [1.00000000e+00, 7.35290604e-15],
       [1.00000000e+00, 7.35528489e-12],
       [9.99979356e-01, 2.06439507e-05],
       [9.99998913e-01, 1.08707283e-06],
       [9.99999998e-01, 2.07980454e-09],
       [1.00000000e+00, 2.57652463e-10],
       [9.99997782e-01, 2.21792869e-06],
       [9.99999971e-01, 2.94073784e-08],
       [1.00000000e+00, 1.84888247e-13],
       [9.99999997e-01, 3.30709327e-09],
```

```
[9.99999989e-01, 1.07981319e-08],
[9.99999995e-01, 5.15835688e-09],
[9.99999998e-01, 2.09368217e-09],
[9.99675433e-01, 3.24567488e-04],
[9.99896928e-01, 1.03071596e-04],
[9.99996710e-01, 3.29020892e-06],
[9.99999998e-01, 1.59474924e-09],
[1.00000000e+00, 6.52173363e-11],
[9.99999934e-01, 6.63339570e-08],
[9.99999995e-01, 4.77295676e-09],
[9.99998473e-01, 1.52695122e-06],
[9.99999388e-01, 6.12160297e-07],
[1.00000000e+00, 1.86621547e-11],
[9.99999811e-01, 1.88552843e-07],
[9.99999978e-01, 2.21031636e-08],
[9.99999952e-01, 4.83580056e-08],
[1.00000000e+00, 2.18167492e-10],
[1.00000000e+00, 2.38769747e-10],
[9.99999747e-01, 2.53122619e-07],
[9.99999963e-01, 3.66072857e-08],
[9.99997640e-01, 2.36039863e-06],
[9.99999897e-01, 1.02749511e-07],
[9.99999963e-01, 3.71438255e-08],
[9.99999920e-01, 7.95790592e-08],
[9.98109261e-01, 1.89073896e-03],
[9.99999987e-01, 1.29753444e-08],
[9.99999994e-01, 6.49605132e-09],
[9.99979095e-01, 2.09052371e-05],
[9.99999619e-01, 3.80542622e-07],
[9.99962290e-01, 3.77098388e-05],
[9.99999929e-01, 7.09338253e-08],
[1.00000000e+00, 9.39713822e-12],
[2.57839929e-10, 1.00000000e+00],
[7.10030580e-10, 9.99999999e-01],
[7.77263991e-05, 9.99922274e-01],
[4.87672154e-07, 9.99999512e-01],
[1.06497653e-13, 1.00000000e+00],
[1.39125574e-09, 9.99999999e-01],
[1.45855962e-07, 9.99999854e-01],
[1.15844233e-10, 1.00000000e+00],
[1.30885221e-07, 9.99999869e-01],
[4.01342547e-03, 9.95986575e-01],
[2.23846844e-06, 9.99997762e-01],
[4.80096836e-12, 1.00000000e+00],
[2.53902784e-07, 9.99999746e-01],
[1.33699810e-06, 9.99998663e-01],
[9.26144795e-13, 1.00000000e+00],
[6.63946614e-09, 9.99999993e-01],
[4.74685878e-05, 9.99952531e-01],
[7.80325429e-12, 1.00000000e+00],
[4.09353206e-09, 9.99999996e-01],
[1.03403537e-07, 9.99999897e-01],
[7.36491730e-11, 1.00000000e+00],
[1.61149398e-08, 9.99999984e-01],
[3.57696778e-09, 9.99999996e-01],
[4.07324020e-10, 1.00000000e+00],
[8.32553012e-08, 9.99999917e-01],
[2.49562814e-04, 9.99750437e-01],
[1.06010371e-12, 1.00000000e+00],
[2.00498829e-08, 9.99999980e-01],
[8.40632056e-13, 1.00000000e+00],
[8.50063744e-10, 9.99999999e-01],
[2.44062509e-07, 9.99999756e-01],
[2.30363365e-08, 9.99999977e-01],
[1.54354318e-12, 1.00000000e+00],
[8.33215085e-06, 9.99991668e-01],
[1.17876940e-09, 9.99999999e-01],
[5.30530954e-09, 9.99999995e-01],
[5.99631715e-05, 9.99940037e-01],
[5.46127020e-10, 9.99999999e-01],
[1.51642669e-05, 9.99984836e-01],
```

```
[3.05980804e-07, 9.99999694e-01],
[1.02057016e-05, 9.99989794e-01],
[9.14538689e-06, 9.99990855e-01],
[1.00902475e-02, 9.89909752e-01],
[8.56886241e-08, 9.99999914e-01],
[8.12302937e-06, 9.99991877e-01],
[5.16071738e-06, 9.99994839e-01],
[3.19395709e-06, 9.99996806e-01],
[2.09877679e-08, 9.99999979e-01],
[3.56352670e-07, 9.99999644e-01],
[8.71899847e-10, 9.99999999e-01],
[1.41279230e-06, 9.99998587e-01],
[2.36027283e-05, 9.99976397e-01],
[6.41034443e-01, 3.58965557e-01],
[5.23253972e-09, 9.99999995e-01],
[1.32300243e-03, 9.98676998e-01],
[9.08053107e-11, 1.00000000e+00],
[6.73462148e-06, 9.99993265e-01],
[1.35862410e-05, 9.99986414e-01],
[1.55411117e-09, 9.99999998e-01],
[1.78898163e-08, 9.99999982e-01],
[5.56246657e-07, 9.99999444e-01],
[2.15857223e-04, 9.99784143e-01],
[3.36432353e-07, 9.99999664e-01],
[3.31609686e-07, 9.99999668e-01],
[6.36769143e-08, 9.99999936e-01],
[3.22594193e-10, 1.00000000e+00],
[6.34359329e-08, 9.99999937e-01],
[1.98389286e-07, 9.99999802e-01],
[2.74824322e-08, 9.99999973e-01],
[3.03502827e-09, 9.99999997e-01],
[3.03011961e-07, 9.99999697e-01],
[5.11191743e-09, 9.99999995e-01],
[3.98535732e-04, 9.99601464e-01],
[1.06161725e-06, 9.99998938e-01],
[3.34384016e-06, 9.99996656e-01],
[6.93024425e-10, 9.99999999e-01],
[3.38301508e-08, 9.99999966e-01],
[2.04188713e-01, 7.95811287e-01],
[4.71264963e-07, 9.99999529e-01],
[1.91537294e-10, 1.00000000e+00],
[6.60530787e-06, 9.99993395e-01],
[1.36686530e-12, 1.00000000e+00],
[1.24327968e-05, 9.99987567e-01],
[3.92591447e-06, 9.99996074e-01],
[2.39191117e-12, 1.00000000e+00],
[5.18414949e-10, 9.99999999e-01],
[6.89257801e-15, 1.00000000e+00],
[4.46495438e-06, 9.99995535e-01],
[2.60311464e-04, 9.99739689e-01],
[3.14866310e-09, 9.99999997e-01],
[5.24164837e-08, 9.99999948e-01],
[1.59225146e-08, 9.99999984e-01],
[3.06876781e-14, 1.00000000e+00],
[4.60126666e-11, 1.00000000e+00],
[1.57673959e-10, 1.00000000e+00],
[1.62251036e-08, 9.99999984e-01],
[1.77302902e-15, 1.00000000e+00],
[7.44778750e-12, 1.00000000e+00],
[6.25496083e-09, 9.99999994e-01],
[3.59014236e-08, 9.99999964e-01]])
```

In [ ]: