ＡＩ基礎セミナー

第１０回
機械学習（４回目：教師なし学習）

改訂履歴

日付	担当者	内容
2021/05/08	M.Takeda	Git 公開

目次

(1) はじめに

・第１０回では、機械学習の４回目として「教師なし学習」を取り上げます。

・入力変数（説明変数）に対する、出力変数（目的変数）が与えられていない状態、
つまり答えが判っていない状態で、入力ﾃﾞｰﾀ自体の背後にある性質や特徴や構造などを分析することを、
「教師なし学習（ｷｮｳｼﾅｼｶﾞｸｼｭｳ、Unsupervised Learning)」と言います。

・これもｾﾐﾅｰ第９回目「教師あり学習（回帰)」と同様に、
昨今の第3次AIﾌﾞｰﾑ以前からある統計手法です。

・昨今のｺﾛﾅｳｨﾙｽ禍でも、ｸﾗｽﾀｰという言葉を耳にする機会が増えましたが、
多種多様な教師無し学習のうち、本ｾﾐﾅｰでは「ｸﾗｽﾀﾘﾝｸﾞ」にﾃｰﾏを絞って解説します。

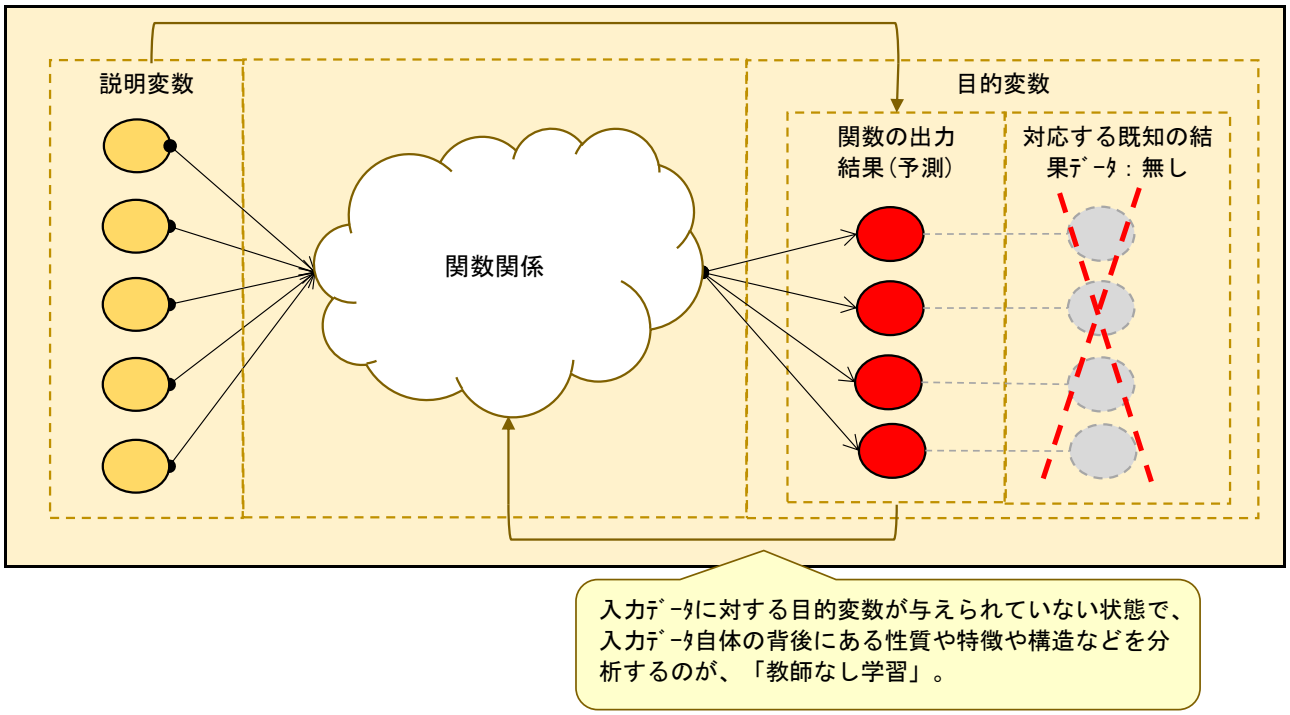
・ｾﾐﾅｰの目標を、教師なし学習の概要理解と、ｸﾗｽﾀｰ分析ﾓﾃﾞﾙの作成・評価方法の理解という点に置きます。

・確認問題は、本文を要約したような問題となっております。
解いてみて、理解度を確認してみてください。
あるいは、本文を読まずに確認問題から先に読んで概要を把握するという進め方でも良いでしょう。

(2) 教師なし学習とは

・入力変数（説明変数）に対する、出力変数（目的変数）が与えられていない状態で、
入力ﾃﾞｰﾀ自体の背後にある性質や特徴や構造などを分析することを、
「教師なし学習（ｷｮｳｼｶﾞｸｼｭｳ、Unsupervised Learning）」と言います。

・「教師なし学習」の例として、「ｸﾗｽﾀﾘﾝｸﾞ（Clustering）」による分類や、
「次元削減（ｼﾞｹﾞﾝｻｸｹﾞﾝ、Dimensionality Reduction）」による特徴量抽出などが挙げられます。



【出典・参考】
教師なし学習⇒「現場で使える！Python 機械学習入門」（2019年05月 翔泳社 大曾根圭輔 他著）

(3) ｸﾗｽﾀﾘﾝｸﾞ

・「ｸﾗｽﾀｰ (Cluster)」とは、色・形・大きさといった特徴が類似したﾃﾞｰﾀ分布の塊を言います。
「ｸﾗｽﾀﾘﾝｸﾞ (Clustering)」はﾃﾞｰﾀ分布の特徴からｸﾗｽﾀｰを見つけて、ﾃﾞｰﾀを分類することです。

・「教師あり学習」の「分類」は、分類済みの入力ﾃﾞｰﾀで分類方法を学習するのに対し、
「教師なし学習」の「ｸﾗｽﾀﾘﾝｸﾞ」は、未分類の似たﾃﾞｰﾀを集めてﾃﾞｰﾀ構造を発見します。
「ｸﾗｽﾀﾘﾝｸﾞ」のｱﾙｺﾞﾘｽﾞﾑとしては、「k平均法 (ｹｲﾍｲｷﾝﾎｳ、K-means clustering)」や
「混合ｶﾞｳｽ分布 (ｺﾝｺﾞｳｶﾞｳｽﾌﾞﾝﾌﾟ、Gaussian mixture models、GMM)」などがあります。

・この章では、「k平均法」と「混合ｶﾞｳｽ分布」について取り上げることにします。

【出典・参考】

k平均法⇒ https://ja.wikipedia.org/wiki/K平均法
自己組織化ﾏｯﾌﾟ⇒ 「あたらしい人工知能の教科書」(2017年08月 翔泳社 多田智史著)
ｽﾗｽﾀﾘﾝｸﾞ⇒ https://products.sint.co.jp/aisia/blog/vol1-9
ｸﾗｽﾀｰ分析⇒ https://bdm.change-jp.com/?p=2025

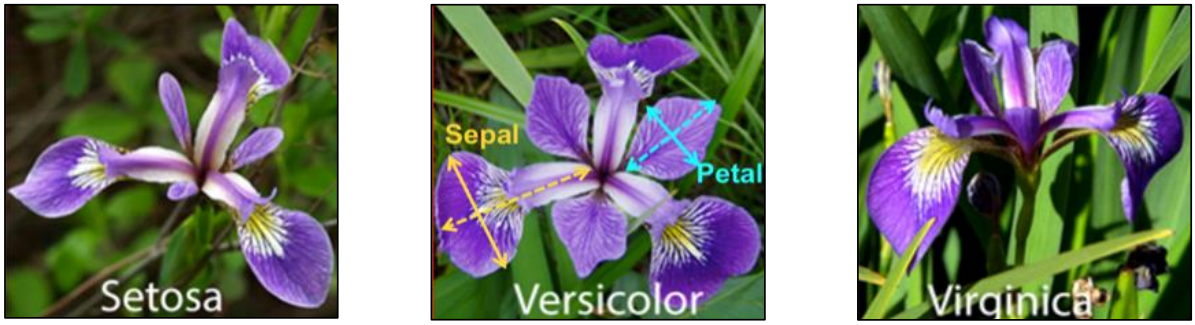
(3.1) 使用するﾃﾞｰﾀ

・この章では「Scikit-learn (ｻｲｷｯﾄﾗｰﾝ)」で提供している「ｱﾔﾒ (Iris)」のﾃﾞｰﾀを使用します。
これは機械学習の分類でよく参照されるﾃﾞｰﾀで、以下のようなものです。

- 3種類のｱﾔﾒ (Setosa、Versicolor、Virginica でﾗﾍﾞﾙ付けされている)
- 4つの特徴量 (がく片(sepals)の幅と長さ、花びら(petals)の幅と長さ、単位はcm)
- 150個のﾃﾞｰﾀ (各種類毎に50個ずつ)

ﾗﾍﾞﾙ付けされていることから判るように、これは教師あり学習で使用するﾃﾞｰﾀですが、
この章ではﾗﾍﾞﾙ付けの情報がない未分類のﾃﾞｰﾀとして参照し、
教師無し学習で、どのように分類されるのかを見てゆきます。

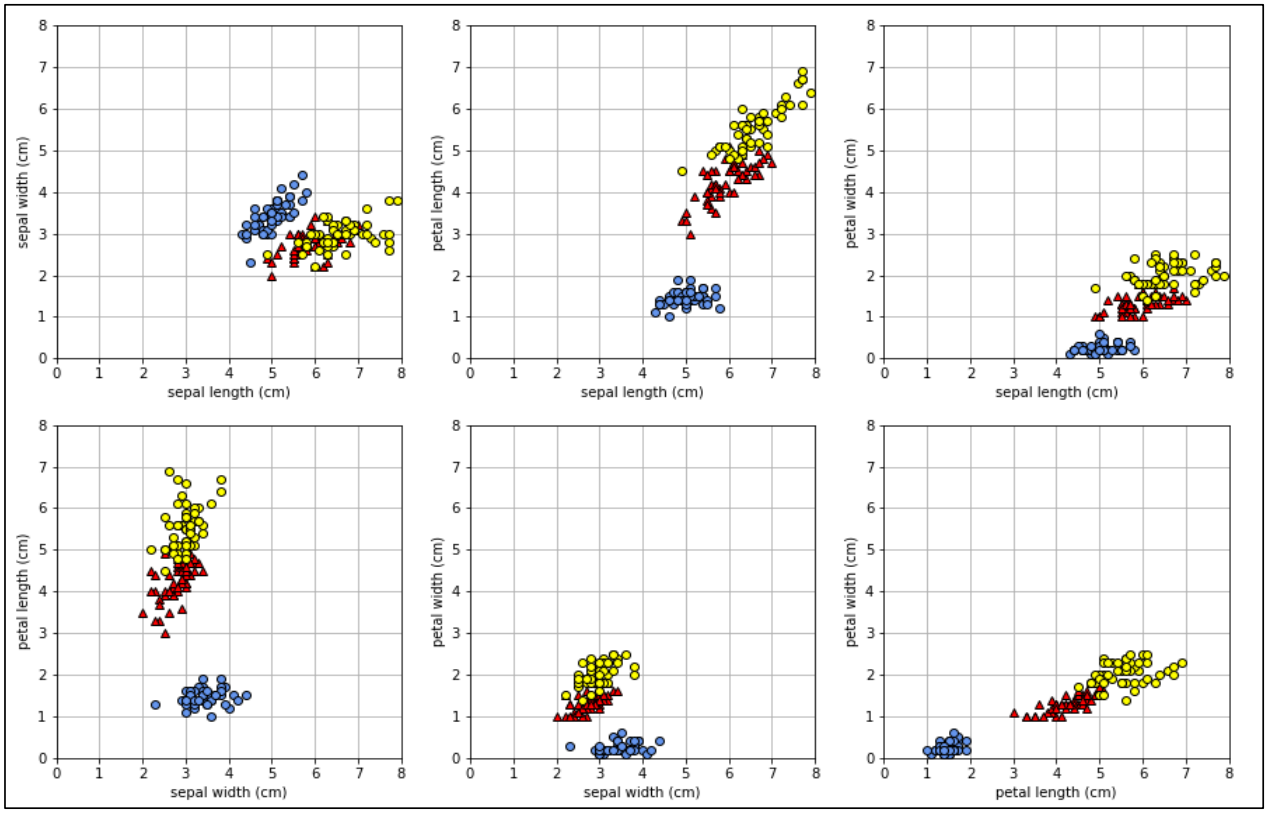
・3種類のｱﾔﾒの写真を掲載します。



引用元：Classification of Iris Varieties
http://suruchifialoke.com/2016-10-13-machine-learning-tutorial-iris-classification/

・取得したﾃﾞｰﾀの散布図を描いてみると、以下のようになりました。
ｸﾗｽﾀｰとしては大まかにみると2個のように見えますが、3つのﾗﾍﾞﾙが付いています。

(1) setosa　　：ｺｰﾝﾌﾗﾜｰﾌﾞﾙｰ○
(2) versicolor　：黄○
(3) virginica　　：赤△



参照(ﾘｽﾄ10-(03)-1_Scikit-learn Iris data)

・Irisﾃﾞｰﾀｾｯﾄの属性を以下に示します：

(1) ｷｰ名 (Iris keys)
(2) 各ｸﾗｽﾀｰのﾗﾍﾞﾙ名 (target_names)
(3) 各ﾃﾞｰﾀの所属ｸﾗｽﾀｰの番号(ﾗﾍﾞﾙ番号) (target)
(4) 4つの特徴量の名前 (Iris feature_names)
(5) 各ｸﾗｽﾀｰのﾃﾞｰﾀ個数 (Iris data number)
(6) 4つの特徴量の値の範囲 (Iris data range of 4 features)

```
--- Iris keys ---
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])

--- Iris target_names ---
target_names=['setosa' 'versicolor' 'virginica']

--- Iris target ---
target=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]

--- Iris feature_names ---
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

--- Iris data number ---
setosa : 50
versicolor : 50
virginica : 50

--- Iris data range of 4 features ---
sepal length (cm) : 4.3～7.9
sepal width (cm) : 2.0～4.4
petal length (cm) : 1.0～6.9
petal width (cm) : 0.1～2.5
( display range : 0～8 )
```

(ﾘｽﾄ10-(03)-1_Scikit-learn Iris data)

```
#**************************************************
# ﾘｽﾄ10-(03)-1_Scikit-learn Iris data
#**************************************************
from sklearn import datasets
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# load Iris data
iris_dt = datasets.load_iris()

# print Iris data
print("--- Iris keys ---")
print("{0}¥n".format(iris_dt.keys()))

print("--- Iris target_names ---")
print("target_names={0}¥n".format(iris_dt.target_names))

print("--- Iris target ---")
print("target={0}¥n".format(iris_dt.target))

print("--- Iris feature_names ---")
print("{0}¥n".format(iris_dt.feature_names))

print("--- Iris data number ---")
for i in range(0, len(iris_dt.target_names)):
    dataNi = sum(x==i for x in iris_dt.target)
```

```
    print("{0} : {1}".format(iris_dt.target_names[i], dataNi))
print("")

print("--- Iris data range of 4 features ---")
DISP_MIN = 0
DISP_MAX = 0
for i in range(0, len(iris_dt.feature_names)):
    print("{0} : {1}～{2}".format(iris_dt.feature_names[i], np.min(iris_dt.data[:, i]), np.max(iris_dt.data[:, i])))
    if i == 0:
        DISP_MIN = np.min(iris_dt.data[:, i])
        DISP_MAX = np.max(iris_dt.data[:, i])
    else:
        DISP_MIN = min(DISP_MIN, np.min(iris_dt.data[:, i]))
        DISP_MAX = max(DISP_MAX, np.max(iris_dt.data[:, i]))

# display range
DISP_MIN = int(DISP_MIN - 1)
DISP_MAX = int(DISP_MAX + 1)
print("( display range : {0}～{1} )".format(DISP_MIN, DISP_MAX))

# plot Iris data
axislist = [[0,1],[0,2],[0,3],[1,2],[1,3],[2,3]]
collist = ['cornflowerblue', 'red', 'yellow']
mrklist = ['o', '^', 'o']

plt.figure(figsize=(15, 10))

for xy in range(0,len(axislist)):
    plt.subplot(2,3, xy+1)
    axisX = axislist[xy][0]
    axisY = axislist[xy][1]

    for i in range(0, len(iris_dt.target_names)):
        iList = np.where(iris_dt.target==i)
        plt.plot(iris_dt.data[iList, axisX],
                 iris_dt.data[iList, axisY],
                 mrklist[i], linestyle='None',
                 markeredgecolor='black', color=collist[i])
    plt.xlim(DISP_MIN, DISP_MAX)
    plt.ylim(DISP_MIN, DISP_MAX)
    plt.xlabel(iris_dt.feature_names[axisX])
    plt.ylabel(iris_dt.feature_names[axisY])
    plt.grid(True)

plt.show()
```

（※参考）

・「Scikit-learn」について、筆者の Python 3.6 の環境ではｴﾗｰが発生してｲﾝｽﾄｰﾙできず、対応として Anaconda を最新のもので再ｲﾝｽﾄｰﾙして、Python 3.6 から 3.7 へ更新して、初めてｲﾝｽﾄｰﾙできました。

【出典・参考】

ｱﾔﾒ(iris)のﾃﾞｰﾀ⇒ Classification of Iris Varieties
http://suruchifialoke.com/2016-10-13-machine-learning-tutorial-iris-classification/

ｱﾔﾒ(iris)のﾃﾞｰﾀ⇒ 【python】iris(アヤメ)のデータセットをpandasとseabornを使って可視化する
https://kenyu-life.com/2019/05/14/iris/

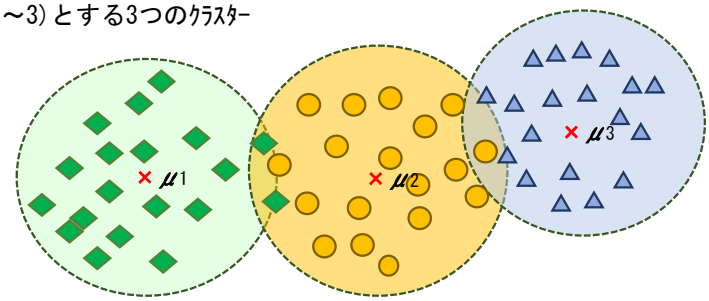
(3.2) K-means法

(3.2.1) K-means法の概要

・「ｸﾗｽﾀﾘﾝｸﾞ」のｱﾙｺﾞﾘｽﾞﾑとして代表的な「K-means法 (ｹｲﾐｰﾝｽﾞﾎｳ、K-平均法、K-means clustering)」について紹介します。

・「K-means」は「k個の平均値」という意味です。
これは、各ﾃﾞｰﾀをK個のｸﾗｽﾀｰの何れかに所属させるｱﾙｺﾞﾘｽﾞﾑで、
各ﾃﾞｰﾀをｸﾗｽﾀｰ中心(平均値)からの距離が最短のものに所属させます。

中心ﾍﾞｸﾄﾙを μ_k (k=1～3)とする3つのｸﾗｽﾀｰ



(3.2.2) K-means法のﾃﾞｰﾀ表現

・以下の説明では、次の表現を用いることにします：

K-means法でのﾃﾞｰﾀ表現

(凡例)
- $\boldsymbol{x}$：ﾃﾞｰﾀの座標値 $(x_1, x_2, \cdots, x_D)$
- n：ﾃﾞｰﾀ番号(n=1～N：ﾃﾞｰﾀ数)
- k：ｸﾗｽﾀｰ番号(k=1～K：ｸﾗｽﾀｰ数)
- d：説明変数番号(d=1～D：説明変数の次元)

・ﾃﾞｰﾀはD次元(D個の特徴量を持つ)で、ﾍﾞｸﾄﾙ $\boldsymbol{x}$ で表現します：

$$\boldsymbol{x} = (x_1, x_2, \cdots, x_D)$$

・ﾃﾞｰﾀは全部で N個あり、n番目のﾃﾞｰﾀ $\boldsymbol{x}_n$(n=1～N：ﾃﾞｰﾀ数)を以下のように表現します：

$$\boldsymbol{x}_n = (x_{n1}, x_{n2}, \cdots, x_{nD})$$

・ﾃﾞｰﾀが所属するｸﾗｽﾀｰは全部で K個あり、
k番目のｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙを $\boldsymbol{\mu}_k$ (k=1～K：ｸﾗｽﾀｰ数)で表現します：

$$\boldsymbol{\mu}_k = (\mu_{k1}, \mu_{k2}, \cdots, \mu_{kD})$$

・n番目のﾃﾞｰﾀ $\boldsymbol{x}_n$(n=1～N：ﾃﾞｰﾀ数)の所属ｸﾗｽﾀｰ k(k=1～K：ｸﾗｽﾀ数)を、
「1-of-K 符号化法 (1-of-K coding scheme)」でﾍﾞｸﾄﾙ $\boldsymbol{r}_n$ で表現します(※1)。
更に、これを全てのﾃﾞｰﾀについて表現し、ｸﾗｽ指示変数Rというﾏﾄﾘｸｽで表現します。

$$\boldsymbol{r}_n = (r_{n1}, r_{n2}, \cdots, r_{nK})$$

$$r_{nk} = \begin{cases} 1 & \text{：n番目のﾃﾞｰﾀ } \boldsymbol{x}_n \text{ がｸﾗｽﾀｰkに属す} \\ 0 & \text{：n番目のﾃﾞｰﾀ } \boldsymbol{x}_n \text{ がｸﾗｽﾀｰkに属さない} \end{cases}$$

ｸﾗｽ指示変数

$$\mathbf{R} = \begin{pmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \\ \vdots \\ \boldsymbol{r}_N \end{pmatrix} = \begin{pmatrix} r_{11}, & r_{12}, & \cdots, & r_{1K} \\ r_{21}, & r_{22}, & \cdots, & r_{2K} \\ \vdots & \vdots & & \vdots \\ r_{N1}, & r_{N2}, & \cdots, & r_{NK} \end{pmatrix}$$

(※1)「1-of-K 符号化法」はｾﾐﾅｰ第８回目を参照のこと

(3.2.3) K-means法のｱﾙｺﾞﾘｽﾞﾑ

・K-means法は、ﾃﾞｰﾀがｸﾗｽﾀｰ中心の周りに同心円状に分布しているものとして、
ﾃﾞｰﾀの分布に合わせてｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙ μ_k を逐次近似により最適化し、
ﾃﾞｰﾀのｸﾗｽﾀｰへの所属をｸﾗｽ指示変数Ｒというﾏﾄﾘｸｽとして決定するものです。

・K-means法のｱﾙｺﾞﾘｽﾞﾑは、
初めに k番目のｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙ μ_k (k=1～K：ｸﾗｽﾀｰ数)の初期値を与えて、
その中心ﾍﾞｸﾄﾙ μ_k を元に全ﾃﾞｰﾀについて所属ｸﾗｽﾀｰを「ｸﾗｽ指示変数Ｒ」として仮決めしておきます。
そのうえで、
(1st) ｸﾗｽﾀｰkに所属する全ﾃﾞｰﾀで、ｸﾗｽﾀｰ中心 μ_k を再計算する
(2nd) 全ﾃﾞｰﾀとｸﾗｽﾀｰ中心 μ_k 間の距離を再計算して、所属を再評価する(ｸﾗｽ指示変数Ｒを更新する)
という手順(1st)～(2nd)を、全ﾃﾞｰﾀの所属ｸﾗｽﾀｰが落着くまで繰り返す、というものです。

以下にｱﾙｺﾞﾘｽﾞﾑの詳細を記します。

K-means法のｱﾙｺﾞﾘｽﾞﾑ

(手順1：初期設定)

k番目のｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙ μ_k (k=1～K：ｸﾗｽﾀｰ数)の初期値を与えます。

$$\mu_k = (\mu_{k1},\ \mu_{k2},\ \cdots,\ \mu_{kD})$$

n番目のﾃﾞｰﾀ x_n(n=1～N：ﾃﾞｰﾀ数)について所属ｸﾗｽﾀｰ k(k=1～K：ｸﾗｽﾀｰ数)の初期値を与え、
ｸﾗｽ指示変数Ｒの初期値を設定します。

$$R = \begin{pmatrix} r_{11}, & r_{12}, & \cdots, & r_{1K} \\ r_{21}, & r_{22}, & \cdots, & r_{2K} \\ \vdots & \vdots & & \vdots \\ r_{N1}, & r_{N2}, & \cdots, & r_{NK} \end{pmatrix} \qquad r_{nk} = \begin{cases} 1：\text{n番目のﾃﾞｰﾀ}x_n\text{がｸﾗｽﾀｰkに属す} \\ 0：\text{n番目のﾃﾞｰﾀ}x_n\text{がｸﾗｽﾀｰkに属さない} \end{cases}$$

(手順2：ｸﾗｽ指示変数Ｒの更新)

n番目のﾃﾞｰﾀ x_n(n=1～N：ﾃﾞｰﾀ数)と k番目のｸﾗｽﾀｰの中心 μ_k との二乗距離 d_{nk}^2 を算出します。

$$\begin{aligned} d_{nk}^2 &= (x_{n1} - \mu_{k1})^2 + (x_{n2} - \mu_{k2})^2 + \cdots + (x_{nD} - \mu_{kD})^2 \\ &= \Sigma_{d=1}^{D}(x_{nd} - \mu_{kd})^2 \end{aligned} \qquad \cdots(\text{式}3.2\text{-}1)$$

そして、二乗距離が最小のｸﾗｽﾀｰを、n番目のﾃﾞｰﾀ x_n の所属ｸﾗｽﾀｰとして扱います。

$$r_{nk} = \begin{cases} 1：k = \mathrm{argmin}_j\ (d_{nj}^2)\text{の時} \\ 0：\text{上記以外の時} \end{cases} \qquad (\mathrm{argmin}_j(\alpha_j)\ \text{は集合}\{\alpha_j \mid 1 \leqq j \leqq K\}\ \text{の要素のうち最小値を与える添え字}j)$$

これにより、ｸﾗｽ指示変数Ｒを更新します。

$$R = \begin{pmatrix} r_{11}, & r_{12}, & \cdots, & r_{1K} \\ r_{21}, & r_{22}, & \cdots, & r_{2K} \\ \vdots & \vdots & & \vdots \\ r_{N1}, & r_{N2}, & \cdots, & r_{NK} \end{pmatrix} \qquad r_{nk} = \begin{cases} 1：\text{n番目のﾃﾞｰﾀ}x_n\text{がｸﾗｽﾀｰkに属す} \\ 0：\text{n番目のﾃﾞｰﾀ}x_n\text{がｸﾗｽﾀｰkに属さない} \end{cases}$$

この手順でｸﾗｽ指示変数Ｒに変更がない場合、
計算が収束したと見なし、本手続きを終了します。

(手順3：μの更新)

k番目のｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙμ_k (k=1～K：ｸﾗｽﾀ数)を、
(手順2)で決定したk番目のｸﾗｽﾀｰに所属する全てのﾃﾞｰﾀの平均値で更新します。
(この平均値はｸﾗｽﾀｰの中心座標に相当します)

$$\mu_k = (\mu_{k1}, \mu_{k2}, \cdots, \mu_{kD})$$

$$\mu_{kd} = 1/N_k \sum_{n' \in \{n' | \text{ｸﾗｽﾀｰkに属するﾃﾞｰﾀの番号}\}} x_{n'd} \quad \cdots(式3.2\text{-}2)$$

N_k：ｸﾗｽﾀｰkに属するﾃﾞｰﾀの総数

この手順で、全てのｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙμ_k (k=1～K：ｸﾗｽﾀｰ数)に変更がない場合、
計算が収束したと見なし、本手続きを終了します。

(手順4：損失関数「歪み尺度」計算)

全てのﾃﾞｰﾀ $\boldsymbol{x}_n$(n=1～N：ﾃﾞｰﾀ数)について、
ｸﾗｽ指示変数Ｒで示される所属ｸﾗｽﾀｰkの中心μ_kとの二乗距離の総和をＪとして計算します。
これを「歪み尺度（ﾕｶﾞﾐｼｬｸﾄﾞ、distortion measure)」と言います。
このＪが K-means法の損失関数で、最小になることを目標とします。

$$J = \sum_{n=1}^{N} d_{nk}^2 = \sum_{n=1}^{N} \sum_{d=1}^{D} (x_{nd} - \mu_{kd})^2 \quad \cdots(式3.2\text{-}3)$$

この手順で損失関数Ｊが十分小さくなった場合、
計算が収束したと見なし、本手続きを終了します。

(手順5)

(手順2)から(手順4)を繰り返します。
繰り返し回数の上限を超えた場合、処理を打ち切ります。

・上記手順で、繰り返し回数の上限を超えて処理を打ち切った場合は、損失関数の傾向をみて、
繰り返し回数を増やしたり、ﾓﾃﾞﾙを見直すといった作業が必要になります。

・計算が収束して処理を終了した場合、その時点での
各ｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙμ_k、およびｸﾗｽ指示変数Ｒがｸﾗｽﾀｰ分析の結果を与えます。

(3.2.4) K-means法の損失関数

・全てのﾃﾞｰﾀ $\boldsymbol{x}_n$(n=1～N：ﾃﾞｰﾀ数)について、
ｸﾗｽ指示変数Ｒで示される所属ｸﾗｽﾀｰkの中心μ_kとの二乗距離の総和をＪとして計算します。
これを「歪み尺度（ﾕｶﾞﾐｼｬｸﾄﾞ、distortion measure)」と言います。
このＪが K-means法の損失関数で、これを最小化するようにﾓﾃﾞﾙを作成します。

歪み尺度（ﾕｶﾞﾐｼｬｸﾄﾞ、distortion measure)

$$J = \sum_{n=1}^{N}\sum_{d=1}^{D}(x_{nd} - \mu_{kd})^2$$ …(式3.2-3)再掲

k ：n番目のﾃﾞｰﾀの所属ｸﾗｽﾀｰ
D ：説明変数の次元
$\boldsymbol{x}_n = (x_{n1}, x_{n2}, \cdots, x_{nD})$ ：n番目のﾃﾞｰﾀ（n=1～N：ﾃﾞｰﾀ数）
$\mu_k = (\mu_{k1}, \mu_{k2}, \cdots, \mu_{kD})$ ：k番目のｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙ（k=1～K：ｸﾗｽﾀｰ数）

なお、K-means法で、手順を繰り返すにつれてＪが最小になる方向に推移するかは、
初期値に依存します。
実例を、(3.2.5)節に記載します。

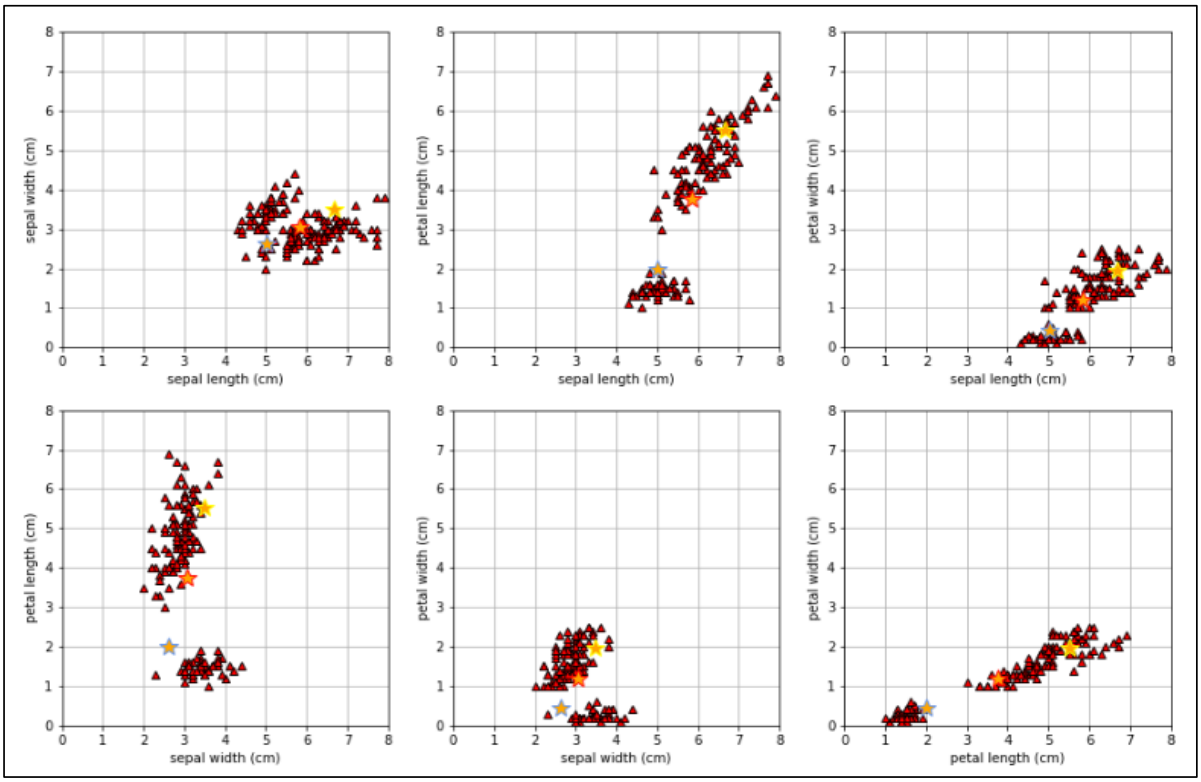
(3.2.5) K-means法の適用と実装例

・では K-means法を「(3.1) 使用するﾃﾞｰﾀ」で紹介したｱﾔﾒ(Iris)のﾃﾞｰﾀに適用してみます。
実は、K-means法で得られる解は、初期値によって変わることが分かっています。
それを踏まえて、2種類の初期化をしてその後の学習状況を見てみましょう。

(実装例 1)

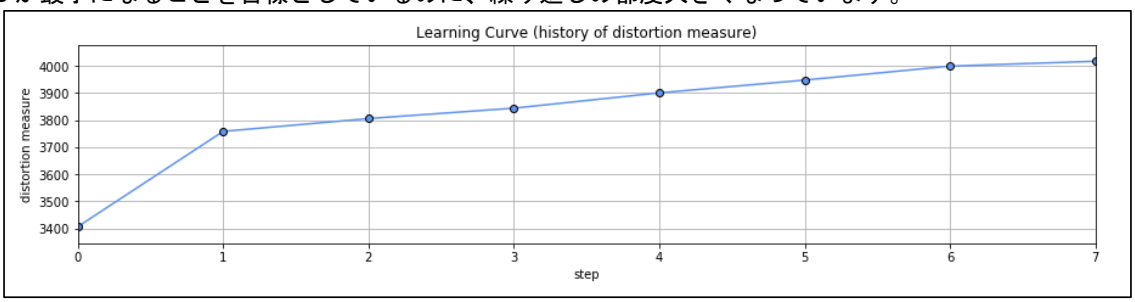
・最初の実装(ﾘｽﾄ10-(03)-2_Scikit-learn Iris data の K-means 法によるｸﾗｽﾀ分解その1)では
各ｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙ μ_k (k=1～K：ｸﾗｽﾀｰ数、図の☆印)を、
各特徴量の平均-σ, ±0, +σに仮決めし、
各ﾃﾞｰﾀを全て1番目のｸﾗｽﾀｰに仮所属させる。
という初期値を与えた例です：

(1) 初期状態で、各ｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙ μ_k (k=1～K：ｸﾗｽﾀｰ数、図の☆印)を
各特徴量の平均-σ, ±0, +σに仮決めし、各ﾃﾞｰﾀを全て1番目のｸﾗｽﾀｰに仮所属させます。



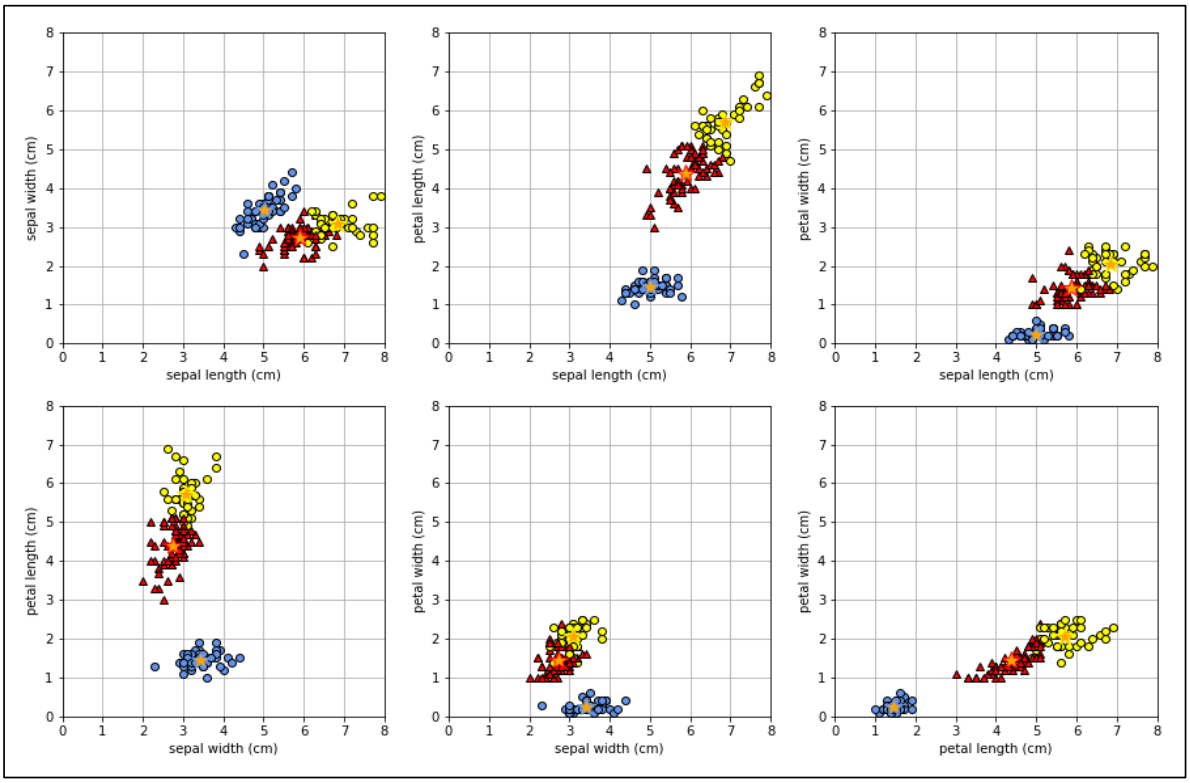
(2) ｸﾗｽﾀｰに所属するﾃﾞｰﾀで中心ﾍﾞｸﾄﾙ μ_k (k=1～K：ｸﾗｽﾀｰ数)を再計算し、
改めてﾃﾞｰﾀとｸﾗｽﾀｰ中心 μ_k 間の距離を再計算して所属を再評価しｸﾗｽ指示変数Rに反映する、
ということを繰り返します(ｽﾃｯﾌﾟ=1～7)。

・計算の繰り返しに伴う「歪み尺度」Ｊの変化の様子は以下のようになりました。
Ｊが最小になることを目標としているのに、繰り返しの都度大きくなっています。

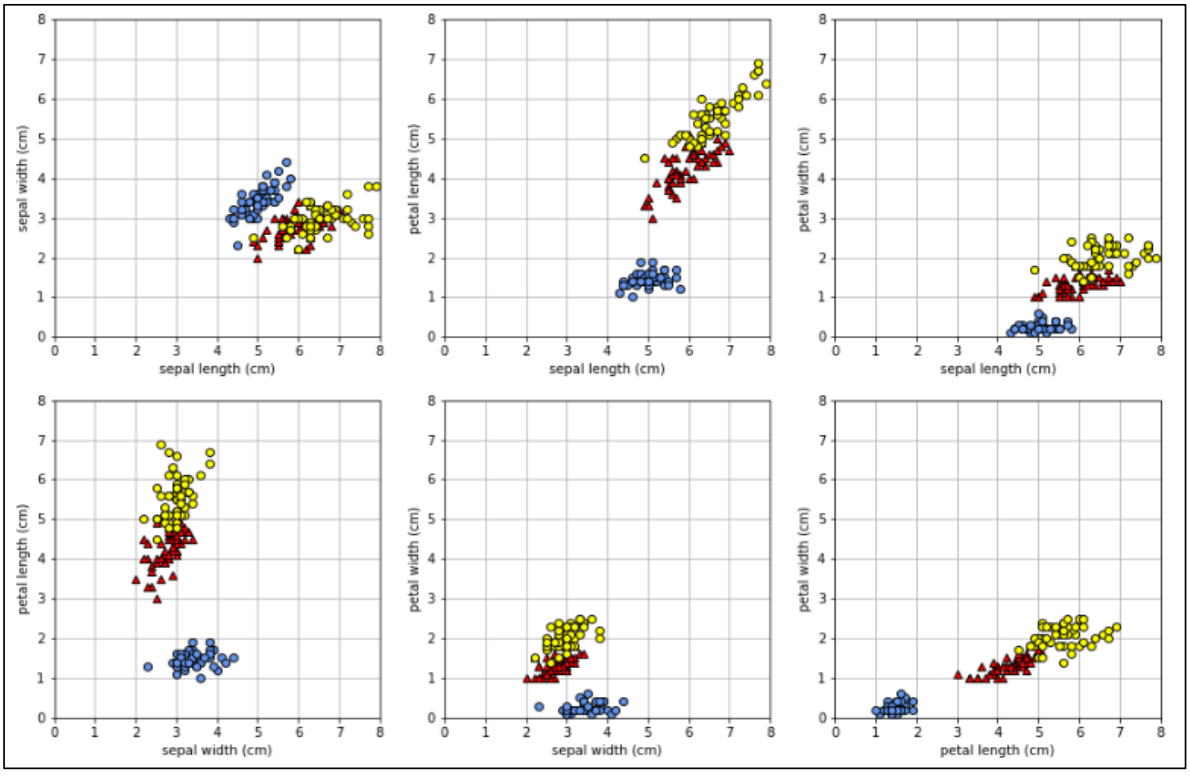


(3) ｽﾃｯﾌﾟ=7で全ﾃﾞｰﾀの所属ｸﾗｽﾀｰに変更が無かったので、
ｸﾗｽﾀｰ分割の処理終了となりました(右図)。
この時の中心(☆印)と分布は
下図のようになりました。

```
ｽﾃｯﾌﾟ=4 ------------------------
ｽﾃｯﾌﾟ=5 ------------------------
ｽﾃｯﾌﾟ=6 ------------------------
ｽﾃｯﾌﾟ=7 ------------------------
所属ｸﾗｽﾀｰに変更が無かったので処理終了。
```



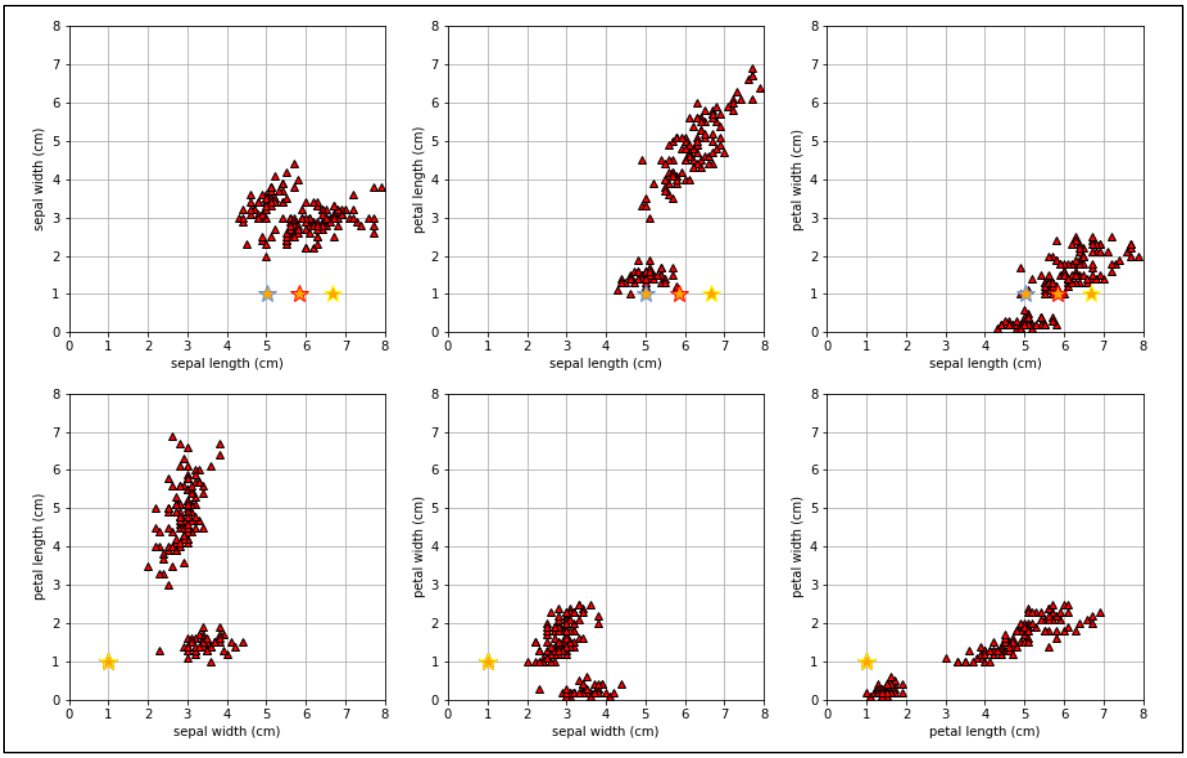
(参考)「(3.1) 使用するﾃﾞｰﾀ」で紹介した元々のﾃﾞｰﾀ分布(下図)と比較すると、
「setosa (ｺｰﾝﾌﾗﾜｰﾌﾞﾙｰ○)」は分離できているものの、
「versicolor (黄○)」と「virginica(赤△)」の境界付近の所属ｸﾗｽﾀｰに差異が見られます。



(実装例 2)

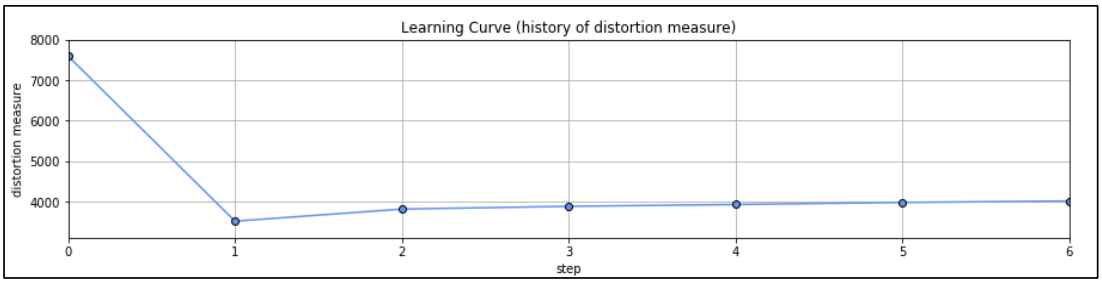
・次の実装(ﾘｽﾄ10-(03)-3_Scikit-learn Iris data の K-means 法によるｸﾗｽﾀ分解その2)では
各ｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙ μ_k (k=1～K：ｸﾗｽﾀｰ数、図の☆印)を、
一番目の特徴量のみ平均-σ, ±0, +σで初期化、他特徴量は1で仮決めし(※1)、
各ﾃﾞｰﾀを全て1番目のｸﾗｽﾀｰに仮所属させる。
という初期値を与えた例です：　　(※1) ここのみ(実装例 1)と異なる

(1) 初期状態で、各ｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙ μ_k (k=1～K：ｸﾗｽﾀｰ数、図の☆印)を
一番目の特徴量のみ平均-σ, ±0, +σで初期化、他特徴量は1で仮決めし、
各ﾃﾞｰﾀを全て1番目のｸﾗｽﾀｰに仮所属させます。



(2) ｸﾗｽﾀｰに所属するﾃﾞｰﾀで中心ﾍﾞｸﾄﾙ μ_k (k=1～K：ｸﾗｽﾀｰ数)を再計算し、
改めてﾃﾞｰﾀとｸﾗｽﾀｰ中心 μ_k 間の距離を再計算して所属を再評価しｸﾗｽ指示変数Rに反映する、
ということを繰り返します(ｽﾃｯﾌﾟ=1～6)。

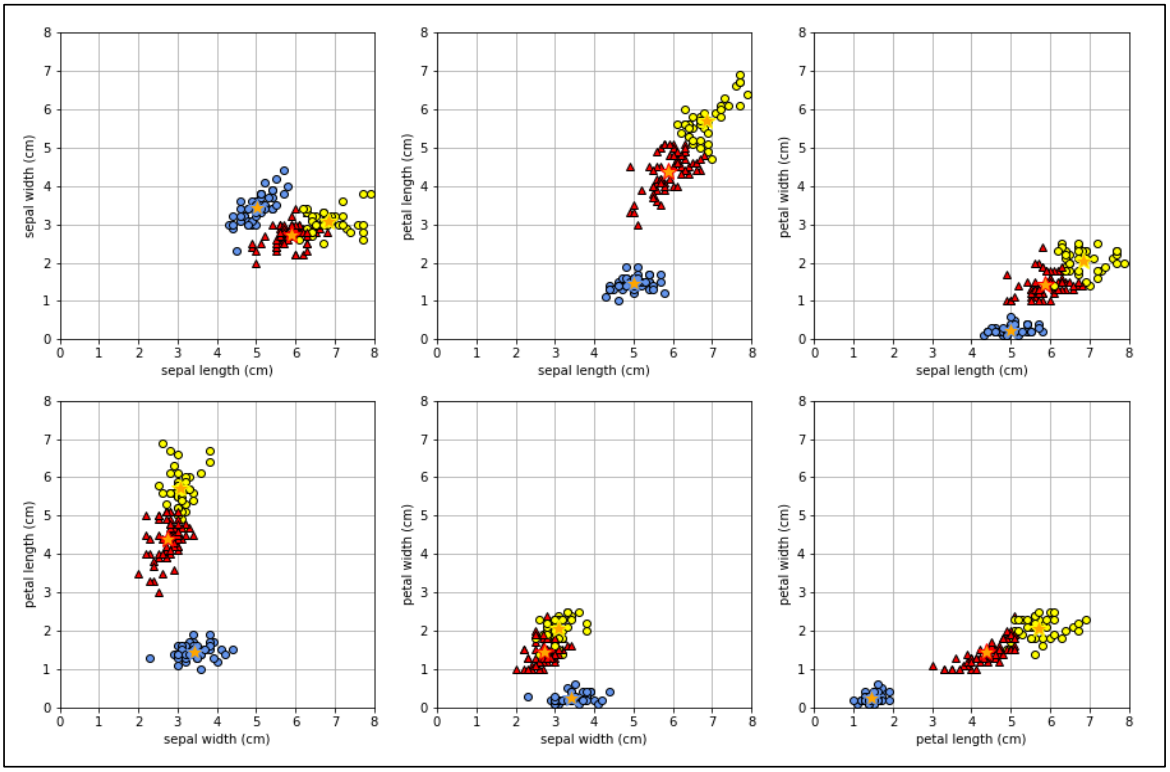
・計算の繰り返しに伴う「歪み尺度」Ｊの変化の様子は以下のようになりました。
Ｊは1回目の学習で下がっていますが、2回目以降の学習では改善はみられません。
(実装例 1)と同様で、Ｊ=4000 付近で収束してます。



(3) ｽﾃｯﾌﾟ=6で全ﾃﾞｰﾀの所属ｸﾗｽﾀｰに変更が無かったので、
ｸﾗｽﾀｰ分割の処理終了となりました(右図)。
この時の中心(☆印)と分布は
下図のようになりました。

```
ｽﾃｯﾌﾟ=3 ------------------------
ｽﾃｯﾌﾟ=4 ------------------------
ｽﾃｯﾌﾟ=5 ------------------------
ｽﾃｯﾌﾟ=6 ------------------------
所属ｸﾗｽﾀｰに変更が無かったので処理終了。
```

(実装例 1)と比べて、収束のｽﾃｯﾌﾟは一つ少なく、結果は同じものとなりました。



(ﾘｽﾄ10-(03)-2_Scikit-learn Iris data の K-means 法によるｸﾗｽﾀ分解その1)

```
#**************************************************************************
# ﾘｽﾄ10-(03)-2_Scikit-learn Iris data の K-means 法によるｸﾗｽﾀ分解その1
# -------------------------------------------------------------------------
# ｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙμの初期値を、各特徴量の平均-σ,±0,+σで初期化
#**************************************************************************
from sklearn import datasets
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# display range
DISP_MIN = 0
DISP_MAX = 8

#**************************************************************************/
# fPlotCluster                                                            */
# 分布を表示する                                                          */
#-------------------------------------------------------------------------*/
# 引数：                                                                  */
#   N       :ﾃﾞｰﾀ総数                                                     */
#   K       :ｸﾗｽﾀｰ総数                                                   */
#   Mu[,]   :ｸﾗｽﾀの中心座標 [k,d] (k=0～K-1:ｸﾗｽﾀ添字、d=0～D-1:特徴量添字)*/
#   Rmtrx[,]:ｸﾗｽ指示変数 [n,k] (n=0～N-1:ﾃﾞｰﾀ添字、k=0～K-1:ｸﾗｽﾀ添字)   */
#   iris_dt :Irisﾃﾞｰﾀ                                                      */
#-------------------------------------------------------------------------*/
# 戻り値：  なし                                                          */
#**************************************************************************/
def fPlotCluster(N, K, Mu, Rmtrx, iris_dt):
    # plot Iris data
    axislist = [[0,1],[0,2],[0,3],[1,2],[1,3],[2,3]]
    collist = ['cornflowerblue', 'red', 'yellow']
    mrklist = ['o', '^', 'o']

    plt.figure(figsize=(15, 10))

    # 座標の組み合わせ毎に表示
    for xy in range(0,len(axislist)):
        plt.subplot(2,3, xy+1)
        axisX = axislist[xy][0]
        axisY = axislist[xy][1]

        # 各ﾃﾞｰﾀをﾌﾟﾛｯﾄ
        for n in range(0, N):
            k = np.argmax(Rmtrx[n,:])
            plt.plot(iris_dt.data[n, axisX],
                     iris_dt.data[n, axisY],
                     mrklist[k], linestyle='None',
                     markeredgecolor='black', color=collist[k])

        # 各ｸﾗｽﾀｰ中心をﾌﾟﾛｯﾄ
        for k in range(0, K):
            plt.plot(Mu[k, axisX],
                     Mu[k, axisY],
                     '*', linestyle='None', markersize=15,
                     markeredgecolor=collist[k], color='orange')

        plt.xlim(DISP_MIN, DISP_MAX)
        plt.ylim(DISP_MIN, DISP_MAX)
        plt.xlabel(iris_dt.feature_names[axisX])
        plt.ylabel(iris_dt.feature_names[axisY])
        plt.grid(True)

    plt.show()
```

```
#*****************************************************************************/
# fPlotJhistory                                                               */
# 「歪み尺度」の時系列を表示                                                  */
#-----------------------------------------------------------------------------*/
# 引数：                                                                      */
#   Jhist  :「歪み尺度」の履歴  Jhist[j] (j=0～jmax:履歴添字)                 */
#   jmax   :「歪み尺度」の履歴の最終 添字                                     */
#-----------------------------------------------------------------------------*/
# 戻り値：  なし                                                              */
#*****************************************************************************/
def fPlotJhistory(Jhist, jmax):
    # 横軸
    jstep = np.zeros(jmax+1)
    for t in range(0, len(jstep)):
        jstep[t] = t

    # 縦軸表示範囲
    yrange = np.max(Jhist[:jmax+1]) - np.min(Jhist[:jmax+1])
    ymin = np.min(Jhist[:jmax+1]) - yrange * 0.1
    ymax = np.max(Jhist[:jmax+1]) + yrange * 0.1

    # 表示
    plt.figure(figsize=(15, 3))
    plt.title("Learning Curve (history of distortion measure)")
    plt.plot(jstep, Jhist[:jmax+1], marker='o', linestyle='-',
             markeredgecolor='black', color='cornflowerblue')
    plt.xlim(0, jmax)
    plt.ylim(ymin, ymax)
    plt.xlabel("step")
    plt.ylabel("distortion measure")
    plt.grid(True)
    plt.show()
    return

#*****************************************************************************/
# fStepR                                                                      */
# 各ﾃﾞｰﾀの所属ｸﾗｽﾀｰを決定し、ｸﾗｽ指示変数に格納する。                         */
# 同時に歪み尺度を計算する。                                                  */
#-----------------------------------------------------------------------------*/
# 引数：                                                                      */
#   N    :ﾃﾞｰﾀ総数                                                          */
#   K    :ｸﾗｽﾀｰ総数                                                        */
#   D    :特徴量ﾃﾞｰﾀの種類数(次元)                                          */
#   Mu   :ｸﾗｽﾀの中心座標 Mu[k,d] (k=0～K-1:ｸﾗｽﾀ添字、d=0～D-1:特徴量添字)*/
#   data :特徴量ﾃﾞｰﾀ data[n,d] (n=0～N-1:ﾃﾞｰﾀ添字、d=0～D-1:特徴量添字) */
#-----------------------------------------------------------------------------*/
# 戻り値:                                                                     */
#  (1) 計算しなおしたｸﾗｽ指示変数                                             */
#          Rnew[n,k] (n=0～N-1:ﾃﾞｰﾀ添字、k=0～K-1:ｸﾗｽﾀ添字)                 */
#  (2) 計算しなおした「歪み尺度」Jnew                                         */
#*****************************************************************************/
def fStepR(N, K, D, Mu, data):
    # ｸﾗｽ指示変数、歪み尺度
    Rnew = np.zeros((N,K))
    Jnew = 0

    # 各ﾃﾞｰﾀ毎に繰り返し
    for n in range(0, N):
        wk = np.zeros(K)

        # 各ｸﾗｽﾀｰ中心からの距離の二乗
        for k in range(0, K):
            for d in range(0, D):
                Jpart = (data[n, d] - Mu[k, d])**2
                wk[k] = wk[k] + Jpart
```

```
                Jnew = Jnew + Jpart

        # 最寄りのｸﾗｽﾀｰに所属させる
        Rnew[n, np.argmin(wk)] = 1

    return Rnew, Jnew

#*****************************************************************************/
# fStepMu                                                                    */
# 各ｸﾗｽﾀｰ中心を所属ﾃﾞｰﾀから更新する                                            */
#-----------------------------------------------------------------------------*/
# 引数：                                                                     */
#   N    :ﾃﾞｰﾀ総数                                                           */
#   K    :ｸﾗｽﾀｰ総数                                                         */
#   D    :特徴量ﾃﾞｰﾀの種類数(次元)                                           */
#   R    :ｸﾗｽ指示変数 R[n,k] (n=0～N-1:ﾃﾞｰﾀ添字、k=0～K-1:ｸﾗｽﾀ添字)        */
#   data :特徴量ﾃﾞｰﾀ data[n,d] (n=0～N-1:ﾃﾞｰﾀ添字、d=0～D-1:特徴量添字)  */
#-----------------------------------------------------------------------------*/
# 戻り値：   計算しなおしたｸﾗｽﾀの中心座標                                     */
#            Munew[k,d] (k=0～K-1:ｸﾗｽﾀ添字、d=0～D-1:特徴量添字)            */
#*****************************************************************************/
def fStepMu(N, K, D, R, data):
    Munew = np.zeros((K, D))
    for k in range(0, K):
        for d in range(0, D):
            Munew[k,d] = np.sum(R[:,k] * data[:,d]) / np.sum(R[:,k])

    return Munew

#*****************************************************************************/
# 本体処理                                                                   */
#*****************************************************************************/

#-----------------------------------------------------------------------------
# Iris ﾃﾞｰﾀをﾛｰﾄﾞし、想定するｸﾗｽﾀ数Kを与えます
#-----------------------------------------------------------------------------
iris_dt = datasets.load_iris()
data = iris_dt.data
N = len(iris_dt.target)

# 想定するｸﾗｽﾀｰ数
K = 3

# 特徴量の種類(次元数)
D = len(iris_dt.feature_names)

#-----------------------------------------------------------------------------
# (手順1：初期設定) 中心ﾍﾞｸﾄﾙμ、ｸﾗｽ指示変数R
#-----------------------------------------------------------------------------
#  ｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙμの初期化（各特徴量の平均-σ,±0,+σで初期化）
Mu = np.zeros([K,D])
Msigma = np.zeros(D)
Mmean = np.zeros(D)
for d in range(0,D):
    Mmean[d] = np.mean(iris_dt.data[:, d])
    Msigma[d] = np.std(iris_dt.data[:, d])
    for k in range(0,K):
        Mu[k][d] = Mmean[d] + Msigma[d] * (k+1-int((K+1)/2))

#  各ﾃﾞｰﾀの所属ｸﾗｽﾀを初期化(全て1番目のｸﾗｽﾀｰに所属、ｸﾗｽ指示変数Rで表現)
R = np.zeros((N,K), dtype=np.int)
for n in range(0,N):
    R[n][1] = 1

# 各ｸﾗｽﾀｰ中心と、各ﾃﾞｰﾀの分布を表示（初期状態）
```

朱書きの部分だけ
(ﾘｽﾄ10-(03)-3) と異なる

```
fPlotCluster(N, K, Mu, R, iris_dt)

#---------------------------------------------------------------------------
# 収束条件を満たすまで繰り返す
#---------------------------------------------------------------------------
# 最大繰り返し回数
REPT_MAX = 200

# 「損失関数（歪み尺度）」の履歴と減少率の境界値
LOSS_LIMIT = 0.00001  # 直前の損失関数値との差異の割合がこれ以下になったら収束とみなす
Jhist = np.zeros(REPT_MAX)
jmax = 0

# 最大で max_rep 回繰り返し
for j in range(0,REPT_MAX):

    # 各ｸﾗｽﾀｰ中心と、各ﾃﾞｰﾀの分布を表示
    print( "ｽﾃｯﾌﾟ={0} ------------------------".format(j))
    if( j == REPT_MAX-1 ):
        print( "最大繰り返し回数に達したので処理終了。")
        break

    # 各ﾃﾞｰﾀの所属ｸﾗｽﾀｰ(ｸﾗｽ指示変数R)を更新する
    Rnew, Jnew = fStepR(N, K, D, Mu, iris_dt.data)
    Jhist[j] = Jnew

    # 損失関数値の減少が殆どなくなったら処理終了
    if( Jhist[j] == 0):
        print( "損失関数値が０になったので処理終了。")
        break
    elif( j >= 1 ):
        lossDiff = abs((Jhist[j] - Jhist[j-1]) / Jhist[j-1])
        if(lossDiff < LOSS_LIMIT):
            print( "損失関数値の減少が殆どなくなったので処理終了。")
            break

    # 所属ｸﾗｽﾀｰに変更が無かったら処理終了
    if( (Rnew == R).all() ):
        print( "所属ｸﾗｽﾀｰに変更が無かったので処理終了。")
        fPlotCluster(N, K, Mu, R, iris_dt)
        break
    R = Rnew

    # 各ｸﾗｽﾀｰ中心(中心ﾍﾞｸﾄﾙμ)を所属ﾃﾞｰﾀから更新する
    Munew = fStepMu(N, K, D, R, iris_dt.data)

    # 各ｸﾗｽﾀｰ中心に変更が無かったら処理終了
    if( (Munew == Mu).all() ):
        print( "各ｸﾗｽﾀｰ中心に変更が無かったので処理終了。")
        fPlotCluster(N, K, Mu, R, iris_dt)
        break
    Mu = Munew

#「歪み尺度」の時系列を表示
fPlotJhistory(Jhist, j)
```

(ﾘｽﾄ10-(03)-3_Scikit-learn Iris data の K-means 法によるｸﾗｽﾀ分解その2)

```
#**************************************************************************
# ﾘｽﾄ10-(03)-3_Scikit-learn Iris data の K-means 法によるｸﾗｽﾀ分解その2
# -------------------------------------------------------------------------
# ｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙμの初期値を、
# 一番目の特徴量のみ平均-σ,±0,+σで初期化、他特徴量は1で初期化
#**************************************************************************
from sklearn import datasets
import numpy as np
import matplotlib.pyplot as plt
```

・・・(途中、(ﾘｽﾄ10-(03)-2) と同一につき、省略)・・・

```
#-------------------------------------------------------------------------
# (手順1：初期設定) 中心ﾍﾞｸﾄﾙμ、ｸﾗｽ指示変数R
#-------------------------------------------------------------------------
# ｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙμの初期化 (一番目の特徴量のみ平均-σ,±0,+σで初期化、他特徴量は1で初期化)
Mu = np.ones([K,D])
Msigma = np.zeros(D)
Mmean = np.zeros(D)
for d in range(0,1):
    Mmean[d] = np.mean(iris_dt.data[:, d])
    Msigma[d] = np.std(iris_dt.data[:, d])
    for k in range(0,K):
        Mu[k][d] = Mmean[d] + Msigma[d] * (k+1-int((K+1)/2))

#  各ﾃﾞｰﾀの所属ｸﾗｽﾀを初期化(全て1番目のｸﾗｽﾀｰに所属、ｸﾗｽ指示変数Rで表現)
R = np.zeros((N,K), dtype=np.int)
for n in range(0,N):
    R[n][1] = 1

# 各ｸﾗｽﾀｰ中心と、各ﾃﾞｰﾀの分布を表示 (初期状態)
fPlotCluster(N, K, Mu, R, iris_dt)

#-------------------------------------------------------------------------
# 収束条件を満たすまで繰り返す
#-------------------------------------------------------------------------
```

朱書きの部分だけ
(ﾘｽﾄ10-(03)-2) と異なる

・・・(途中、(ﾘｽﾄ10-(03)-2) と同一につき、省略)・・・

```
#「歪み尺度」の時系列を表示
fPlotJhistory(Jhist, j)
```

【参考】

K-means法

⇒「Pythonで動かして学ぶ！あたらしい機械学習の教科書」(2018年01月 翔泳社 伊藤真著)

⇒「ﾊﾟﾀｰﾝ認識と機械学習 下」(2019年03月 丸善 C.M.ﾋﾞｼｮｯﾌﾟ著)

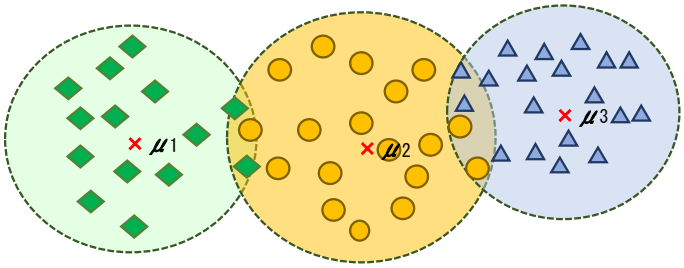
(3.3) 混合ｶﾞｳｽﾓﾃﾞﾙ

(3.3.1) 混合ｶﾞｳｽﾓﾃﾞﾙの概要

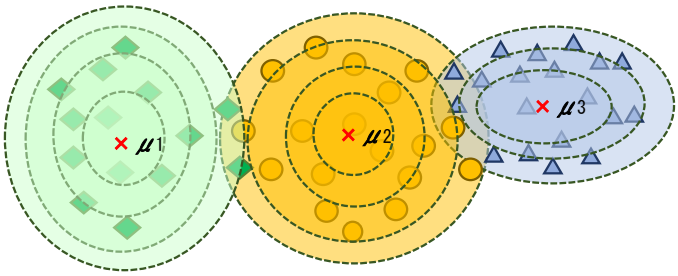
・前節で紹介した「K-means法」は、各ﾃﾞｰﾀをK個のｸﾗｽﾀｰのうち、ｸﾗｽﾀｰ中心(平均値)からの距離が最短のものに所属させるｱﾙｺﾞﾘｽﾞﾑで、分布の形が円(3次元の場合は球)であることを想定しています。

・しかしながら、ｸﾗｽﾀｰ境界付近に分布しているﾃﾞｰﾀが、どちらのｸﾗｽﾀｰに属しているのかは曖昧である為、ｸﾗｽﾀｰへの所属確率(これを「負担率 (ﾌﾀﾝﾘﾂ、responsibility)」といいます)で表現する方が適切です。
またｸﾗｽﾀｰの分布を円で表現するより、より実態に近い分布で表現したいところです。
この表現としてﾃﾞｰﾀ分布がK個のｶﾞｳｽ分布から構成されるﾓﾃﾞﾙを導入します。
これが「混合ｶﾞｳｽﾓﾃﾞﾙ (ｺﾝｺﾞｳｶﾞｳｽﾓﾃﾞﾙ、Gaussian mixture model、GMM)」です。

K-means法のﾓﾃﾞﾙは、K個の各ｸﾗｽﾀｰが円状に分布しているとし、ﾃﾞｰﾀのｸﾗｽﾀｰ所属の有無で表現します



混合ｶﾞｳｽﾓﾃﾞﾙは、K個の各ｸﾗｽﾀｰがｶﾞｳｽ分布しているとし、ﾃﾞｰﾀのｸﾗｽﾀへの所属確率で表現します



(3.3.2) 混合ｶﾞｳｽﾓﾃﾞﾙのﾃﾞｰﾀ表現

・「混合ｶﾞｳｽﾓﾃﾞﾙ」ではﾃﾞｰﾀの分布が、K個の(D次元)ｶﾞｳｽ分布(正規分布)の合成であるとします。
k番目(k=1～K：ｸﾗｽﾀｰ数)のｸﾗｽﾀｰは、その中心位置がｶﾞｳｽ分布の中心 $\boldsymbol{\mu}_k$(D次元)で、
分布の広がりは共分散行列 $\boldsymbol{\Sigma}_k$(D×D の正方行列)で表現されます。
また、各ｶﾞｳｽ分布の大きさの比率を π_k で表現します。

混合ｶﾞｳｽﾓﾃﾞﾙのﾃﾞｰﾀ表現

(凡例)
- $\boldsymbol{x}$：ﾃﾞｰﾀの座標値 $(x_1, x_2, \cdots, x_D)$
- n：ﾃﾞｰﾀ番号(n=1～N：ﾃﾞｰﾀ数)
- k：ｸﾗｽﾀｰ番号(k=1～K：ｸﾗｽﾀｰ数)
- d：説明変数番号(d=1～D：説明変数の次元)

・混合ｶﾞｳｽﾓﾃﾞﾙの分布関数は次式で与えられます：

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \, N_k(\boldsymbol{x}) \qquad \cdots(式3.3\text{-}1)$$

N_k：k番目のｶﾞｳｽ分布(D次元)

$$N_k = \{1/(2\pi)^{D/2}\} (\det \boldsymbol{\Sigma}_k)^{-1/2} \exp\{-(\boldsymbol{x}-\boldsymbol{\mu}_k)^T (\boldsymbol{\Sigma}_k)^{-1} (\boldsymbol{x}-\boldsymbol{\mu}_k)/2\} \qquad \cdots(式3.3\text{-}2)$$

π_k：k番目のｶﾞｳｽ分布 N_k の大きさの比率(混合係数)

$$\pi_k : \quad 0 \leqq \pi_k \leqq 1 \quad 、1 = \sum_{k=1}^{K} \pi_k \qquad \cdots(式3.3\text{-}3)$$

$\boldsymbol{\mu}_k$：k番目のｶﾞｳｽ分布 N_k の中心ﾍﾞｸﾄﾙ

$$\boldsymbol{\mu}_k = (\mu_{k1}, \mu_{k2}, \cdots, \mu_{kD}) \qquad \cdots(式3.3\text{-}4)$$

$\boldsymbol{\Sigma}_k$：k番目のｶﾞｳｽ分布 N_k の広がり方(共分散行列)

$$\boldsymbol{\Sigma}_k = \begin{pmatrix} \sigma_{k11}, & \sigma_{k12}, & \cdots, & \sigma_{k1D} \\ \sigma_{k21}, & \sigma_{k22}, & \cdots, & \sigma_{k2D} \\ \vdots & \vdots & & \vdots \\ \sigma_{kD1}, & \sigma_{kD2}, & \cdots, & \sigma_{kDD} \end{pmatrix} \qquad \sigma_{kij} = \sigma_{kji} \qquad \cdots(式3.3\text{-}5)$$

・混合ｶﾞｳｽﾓﾃﾞﾙの分布関数により、
座標 $\boldsymbol{x}$ にある点の、ｸﾗｽﾀｰk への所属確率 $p_k(\boldsymbol{x})$ は次式により与えられます：

$$p_k(\boldsymbol{x}) = \pi_k N_k(\boldsymbol{x}) \ / \ \sum_{k=1}^{K} \pi_k N_k(\boldsymbol{x}) \qquad \cdots(式3.3\text{-}6)$$

これにより、n番目のﾃﾞｰﾀ $\boldsymbol{x}_n$ のｸﾗｽﾀｰk への所属確率 γ_{nk} は次式で与えられます：

$$\gamma_{nk} = p_k(\boldsymbol{x}_n) = \pi_k N_k(\boldsymbol{x}_n) \ / \ \sum_{k=1}^{K} \pi_k N_k(\boldsymbol{x}_n) \qquad \cdots(式3.3\text{-}7)$$

所属確率 γ_{nk} を要素とする行列 $\boldsymbol{\gamma}$ は「負担率 (ﾌﾀﾝﾘﾂ、responsibility)」と言います：

$$\boldsymbol{\gamma} = \begin{pmatrix} \gamma_{11}, & \gamma_{12}, & \cdots, & \gamma_{1K} \\ \gamma_{21}, & \gamma_{22}, & \cdots, & \gamma_{2K} \\ \vdots & \vdots & & \vdots \\ \gamma_{N1}, & \gamma_{N2}, & \cdots, & \gamma_{NK} \end{pmatrix} \qquad \cdots(式3.3\text{-}8)$$

γ_{nk}：n番目のﾃﾞｰﾀ $\boldsymbol{x}_n$ がｸﾗｽﾀｰkに属している確率
(制約条件：$1 = \sum_{k=1}^{K} \gamma_{nk}$)

※参考までに、1次元(D=1)の場合はお馴染みの正規分布とります：

$\boldsymbol{\Sigma}_k$：k番目のｶﾞｳｽ分布 N_k の広がり方(共分散行列)

$$\boldsymbol{\Sigma}_k = \sigma_k \qquad (標準偏差)$$

N_k：k番目のｶﾞｳｽ分布(正規分布)

$$N_k = \{1/\sqrt{2\pi}\sigma_k\} \exp\{-(x-\mu_k)^2/2\sigma_k^2\}$$

(3.3.3) 混合ｶﾞｳｽﾓﾃﾞﾙのｱﾙｺﾞﾘｽﾞﾑ

・混合ｶﾞｳｽﾓﾃﾞﾙによるﾃﾞｰﾀのｸﾗｽﾀﾘﾝｸﾞは、
「EMｱﾙｺﾞﾘｽﾞﾑ（ｲｰｴﾑｱﾙｺﾞﾘｽﾞﾑ、Expectation-Maximization algorithm)」を用いて行います。
「EMｱﾙｺﾞﾘｽﾞﾑ」は、K-means法を拡張した方法で、ﾃﾞｰﾀの分布に合わせて構成するｶﾞｳｽ分布の
ﾊﾟﾗﾒｰﾀ(混合係数、中心、共分散行列)を逐次近似により最適化し、
ﾃﾞｰﾀのｸﾗｽﾀｰへの所属の程度を負担率という形で決定するｱﾙｺﾞﾘｽﾞﾑです。

概略ｱﾙｺﾞﾘｽﾞﾑは以下のようになります：

（初期化）　k番目のｸﾗｽﾀｰのｶﾞｳｽ分布N_kの広がり方(中心ﾍﾞｸﾄﾙ $\boldsymbol{\mu}_k$、共分散行列$\boldsymbol{\Sigma}_k$)、
および混合係数π_kの初期値を与えます。

（Eｽﾃｯﾌﾟ）　現時点でのπ_k、$\boldsymbol{\mu}_k$、$\boldsymbol{\Sigma}_k$を用いて、負担率$\boldsymbol{\gamma}$(各ﾃﾞｰﾀのｸﾗｽﾀｰへの所属確率)を計算します。

（Mｽﾃｯﾌﾟ）　現時点での負担率$\boldsymbol{\gamma}$を用いて、π_k、$\boldsymbol{\mu}_k$、$\boldsymbol{\Sigma}_k$を計算します。

上記（Eｽﾃｯﾌﾟ）と（Mｽﾃｯﾌﾟ）を計算が収束するまで交互に繰り返します。

・以下にｱﾙｺﾞﾘｽﾞﾑの詳細を記します。

EMｱﾙｺﾞﾘｽﾞﾑ

(手順1：初期設定)

ﾊﾟﾗﾒｰﾀを初期化します

(1) k番目(k=1～K：ｸﾗｽﾀｰ数)のｶﾞｳｽ分布N_kの混合係数π_k を初期化します

π_k ：　$0 \leqq \pi_k \leqq 1$　　(制約条件：$1 = \sum_{k=1}^{K} \pi_k$)

(2) k番目(k=1～K：ｸﾗｽﾀｰ数)のｶﾞｳｽ分布N_kの中心ﾍﾞｸﾄﾙ$\boldsymbol{\mu}_k$ を初期化します

$$\boldsymbol{\mu}_k = (\mu_{k1},\ \mu_{k2},\ \cdots,\ \mu_{kD})$$

(3) k番目(k=1～K：ｸﾗｽﾀｰ数)のｶﾞｳｽ分布N_kの共分散行列$\boldsymbol{\Sigma}_k$ を初期化します

$$\boldsymbol{\Sigma}_k = \begin{pmatrix} \sigma_{k11}, & \sigma_{k12}, & \cdots, & \sigma_{k1D} \\ \sigma_{k21}, & \sigma_{k22}, & \cdots, & \sigma_{k2D} \\ \vdots & \vdots & & \\ \sigma_{kD1}, & \sigma_{kD2}, & \cdots, & \sigma_{kDD} \end{pmatrix} \qquad \sigma_{kij} = \sigma_{kji},\ \sigma_{kii} > 0$$

(4) 負担率$\boldsymbol{\gamma}$ を初期化します

$$\boldsymbol{\gamma} = \begin{pmatrix} \gamma_{11}, & \gamma_{12}, & \cdots, & \gamma_{1K} \\ \gamma_{21}, & \gamma_{22}, & \cdots, & \gamma_{2K} \\ \vdots & \vdots & & \\ \gamma_{N1}, & \gamma_{N2}, & \cdots, & \gamma_{NK} \end{pmatrix}$$

γ_{nk}：n番目のﾃﾞｰﾀ$\boldsymbol{x}_n$がｸﾗｽﾀｰkに属している確率

(制約条件：$1 = \sum_{k=1}^{K} \gamma_{nk}$)

初期化例を以下に示します：

(1) 混合係数π_k　　：割合が等しいものとして初期化

$\pi_k = 1/K$　(for k=1～K-1)、　$\pi_K = 1 - \sum_{k=1}^{K-1} \pi_k$

(2) 中心ﾍﾞｸﾄﾙ$\boldsymbol{\mu}_k$　　：各成分d毎の平均m_dと標準偏差σ_dで初期化

$\mu_{kd} = m_d + \sigma_d \times (k - (\mathrm{int})((K+1)/2))$

(3) 共分散行列$\boldsymbol{\Sigma}_k$　　：$\boldsymbol{\Sigma}_k$ を単位行列 I で初期化

$\boldsymbol{\Sigma}_k = \mathrm{I}$

(4) 負担率$\boldsymbol{\gamma}$　　：全てｸﾗｽﾀｰ1に所属しているとして初期化

$\gamma_{n1} = 1$、　$\gamma_{nk} = 0$ (for $k \neq 1$)

(手順2：E Step) (Expectation Step)

現時点でのπ_k、$\boldsymbol{\mu}_k$、$\boldsymbol{\Sigma}_k$を用いて、負担率$\boldsymbol{\gamma}$を以下の関係式から計算します。

n番目のﾃﾞｰﾀ $\boldsymbol{x}_n$ の、ｸﾗｽﾀｰk への所属確率γ_{nk}は(式3.3-7)により与えられます：

$$\gamma_{nk} = p_k(\boldsymbol{x}_n) = \pi_k N_k(\boldsymbol{x}_n) \ / \ \sum_{k=1}^{K} \pi_k N_k(\boldsymbol{x}n) \qquad \cdots(式3.3\text{-}7)_{再掲}$$

この式で $N_k(\boldsymbol{x}_n)$ は(式3.3-2) により得られます：

$$N_k(\boldsymbol{x}_n) = \{1/(2\pi)^{D/2}\} (\det \boldsymbol{\Sigma}_k)^{-1/2} \exp\{-(\boldsymbol{x}_n-\boldsymbol{\mu}_k)^T(\boldsymbol{\Sigma}_k)^{-1}(\boldsymbol{x}_n-\boldsymbol{\mu}_k)/2\} \qquad \cdots(式3.3\text{-}9)$$

この手順で$\boldsymbol{\gamma}$に変更がない場合、
計算が収束したと見なし、本手続きを終了します。

(手順3：M Step) (Maximization Step)

現時点での負担率$\boldsymbol{\gamma}$を用いて、π_k、$\boldsymbol{\mu}_k$、$\boldsymbol{\Sigma}_k$を計算します。

(1) 負担率$\boldsymbol{\gamma}$から、k番目(k=1～K：ｸﾗｽﾀｰ数)のｸﾗｽﾀｰへの負担率の総和 R_k を計算します。
これは K-means法での k番目のｸﾗｽﾀｰへの所属ﾃﾞｰﾀ総数に相当します。

$$R_k = \sum_{n=1}^{N} \gamma_{nk} \qquad \cdots(式3.3\text{-}10)$$

(2) (式3.3-10)で求めた負担率の総和 R_k から混合係数π_k を更新します。
これはk番目のｸﾗｽﾀｰへの所属ﾃﾞｰﾀ数の全ﾃﾞｰﾀ数に占める割合です。

$$\pi_k = \sum_{n=1}^{N} \gamma_{nk} \ / \sum_{n=1}^{N} 1 = R_k/N \qquad \cdots(式3.3\text{-}11)$$

(3) 負担率$\boldsymbol{\gamma}$と(式3.3-10)で求めた負担率の総和 R_k から、中心ﾍﾞｸﾄﾙ$\boldsymbol{\mu}_k$ を更新します。
これは全ﾃﾞｰﾀ$\boldsymbol{x}_n$ (n=1～N)の、k番目のｸﾗｽﾀｰの負担率γ_{nk}を重みとする
加重平均(重心)に相当します。

$$\boldsymbol{\mu}_k = \sum_{n=1}^{N} \gamma_{nk} \boldsymbol{x}_n / \sum_{n=1}^{N} \gamma_{nk} = (1/R_k) \sum_{n=1}^{N} \gamma_{nk} \boldsymbol{x}_n \qquad \cdots(式3.3\text{-}12)$$

(4) k番目(k=1～K：ｸﾗｽﾀｰ数)のｶﾞｳｽ分布N_kの共分散行列$\boldsymbol{\Sigma}_k$ を次式で再計算します。
これはｸﾗｽﾀｰの負担率γ_{nk}という重みを付けたﾃﾞｰﾀの共分散行列を求めるものです。

$$\boldsymbol{\Sigma}_k = (1/R_k) \sum_{n=1}^{N} \gamma_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \qquad \cdots(式3.3\text{-}13)$$

この手順でπ_k、$\boldsymbol{\mu}_k$、$\boldsymbol{\Sigma}_k$ に変更がない場合、
計算が収束したと見なし、本手続きを終了します。

(手順4：損失関数計算)

現時点でのπ_k、$\boldsymbol{\mu}_k$、$\boldsymbol{\Sigma}_k$を用いて、損失関数E(π, $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$)を以下の関係式から計算します。

$$E(\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\sum_{n=1}^{N} \{ \log(\sum_{k=1}^{K} \pi_k N_k(\boldsymbol{x}_n)) \} \qquad \cdots(式3.3\text{-}14)$$

この式で $N_k(\boldsymbol{x}_n)$ は(式3.3-9) により得られます。

この手順で損失関数Eが十分小さくなった場合、
計算が収束したと見なし、本手続きを終了します。

(手順5)

(手順2)から(手順4)を繰り返します。
繰り返し回数の上限を超えた場合、処理を打ち切ります。

・上記手順で、繰り返し回数の上限を超えて処理を打ち切った場合は、損失関数の傾向をみて、
繰り返し回数を増やしたり、ﾓﾃﾞﾙを見直すといった作業が必要になります。

・計算が収束して処理を終了した場合、その時点での
各ｸﾗｽﾀｰのπ_k、$\boldsymbol{\mu}_k$、$\boldsymbol{\Sigma}_k$および$\boldsymbol{\gamma}$がｸﾗｽﾀｰ分析の結果を与えます。

(3.3.4) 混合ｶﾞｳｽﾓﾃﾞﾙの損失関数

・混合ｶﾞｳｽﾓﾃﾞﾙの損失関数Ｅ(π, μ, Σ)は、混合係数π_k、中心ﾍﾞｸﾄﾙ$\boldsymbol{\mu}_k$、共分散行列$\boldsymbol{\Sigma}_k$を用いて、
以下の関係式から計算します。

混合ｶﾞｳｽﾓﾃﾞﾙの損失関数

$$E(\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\sum_{n=1}^{N} \{ \log(\sum_{k=1}^{K} \pi_k N_k(\boldsymbol{x}_n)) \} \quad \cdots(式3.3\text{-}14)_{再掲}$$

$$N_k(\boldsymbol{x}_n) = \{1/(2\pi)^{D/2}\} (\det \boldsymbol{\Sigma}_k)^{-1/2} \exp\{-(\boldsymbol{x}_n-\boldsymbol{\mu}_k)^T (\boldsymbol{\Sigma}_k)^{-1} (\boldsymbol{x}_n-\boldsymbol{\mu}_k)/2\} \quad \cdots(式3.3\text{-}9)_{再掲}$$

$\boldsymbol{x}_n$ ：n番目のﾃﾞｰﾀ (n=1～N：ﾃﾞｰﾀ数)
N_k ：k番目(k=1～K：ｸﾗｽﾀｰ数)ｸﾗｽﾀｰのｶﾞｳｽ分布
π_k ：N_kの混合係数
$\boldsymbol{\mu}_k$ ：N_kの中心ﾍﾞｸﾄﾙ
$\boldsymbol{\Sigma}_k$ ：N_kの共分散行列

・まず、各ﾃﾞｰﾀ$\boldsymbol{x}_n$ のｸﾗｽﾀｰkへの所属確率は、(式3.3.9)の $N_k(\boldsymbol{x}_n)$で与えられます。
これに各ｸﾗｽﾀｰの混合係数π_kをかけて全ｸﾗｽﾀｰについて総和を取ったものが
(式3.3-14)の右辺の括弧 {} 内の式です。
これは各ﾃﾞｰﾀ$\boldsymbol{x}_n$ の全ｸﾗｽﾀｰへの所属確率の総和 $P(\boldsymbol{x}_n)$ になり、
ﾓﾃﾞﾙ全体での$\boldsymbol{x}_n$の所属確率になります：

$$P(\boldsymbol{x}_n) = \sum_{k=1}^{K} \pi_k N_k(\boldsymbol{x}_n) \quad \cdots(式3.3\text{-}15)$$

・(式3.3-15)を全ﾃﾞｰﾀについて総乗をとったものが、混合ｶﾞｳｽﾓﾃﾞﾙの
「尤度関数 (ﾕｳﾄﾞｶﾝｽｳ、likelihood function)」Ｌです。
尤度関数Ｌを最大化するようにﾓﾃﾞﾙを作成します：

$$L = \prod_{n=1}^{N} P(\boldsymbol{x}_n) = \prod_{n=1}^{N} \{ \sum_{k=1}^{K} \pi_k N_k(\boldsymbol{x}_n) \} \quad \cdots(式3.3\text{-}16)$$

・さらに尤度関数(式3.3-16) Ｌ の対数をとったものが
「対数尤度関数 (ﾀｲｽｳﾕｳﾄﾞｶﾝｽｳ、log likelihood function)」です。
対数関数の単調増加性により、
対数尤度関数を最大化することは、尤度を最大化することでもあります：

$$\begin{aligned} \log(L) &= \log\left(\prod_{n=1}^{N} P(\boldsymbol{x}_n)\right) \\ &= \sum_{n=1}^{N} \{ \log(P(\boldsymbol{x}_n)) \} \\ &= \sum_{n=1}^{N} \{ \log(\sum_{k=1}^{K} \pi_k N_k(\boldsymbol{x}_n)) \} \end{aligned} \quad \cdots(式3.3\text{-}17)$$

・尤度を最大化することは、ﾓﾃﾞﾙの誤差を最小化することであり、
対数尤度関数に、－１(ﾏｲﾅｽ1)を掛けたものを損失関数Ｅ(π, μ, Σ)とします。
これが、混合ｶﾞｳｽﾓﾃﾞﾙの損失関数であり、これを最小化するようにﾓﾃﾞﾙを作成します：

$$E(\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\sum_{n=1}^{N} \{ \log(\sum_{k=1}^{K} \pi_k N_k(\boldsymbol{x}_n)) \} \quad \cdots(式3.3\text{-}14)_{再掲}$$

(3.3.5) 混合ｶﾞｳｽﾓﾃﾞﾙの適用と実装例

・では混合ｶﾞｳｽﾓﾃﾞﾙを「(3.1) 使用するﾃﾞｰﾀ」で紹介したｱﾔﾒ(Iris)のﾃﾞｰﾀに適用してみます。
後述の実装では以下のような実行結果になりました。

(1) 初期状態は以下のとおりで仮決めし、
この初期化によるｸﾗｽﾀｰの分布状態は下図のとおりとなりました：

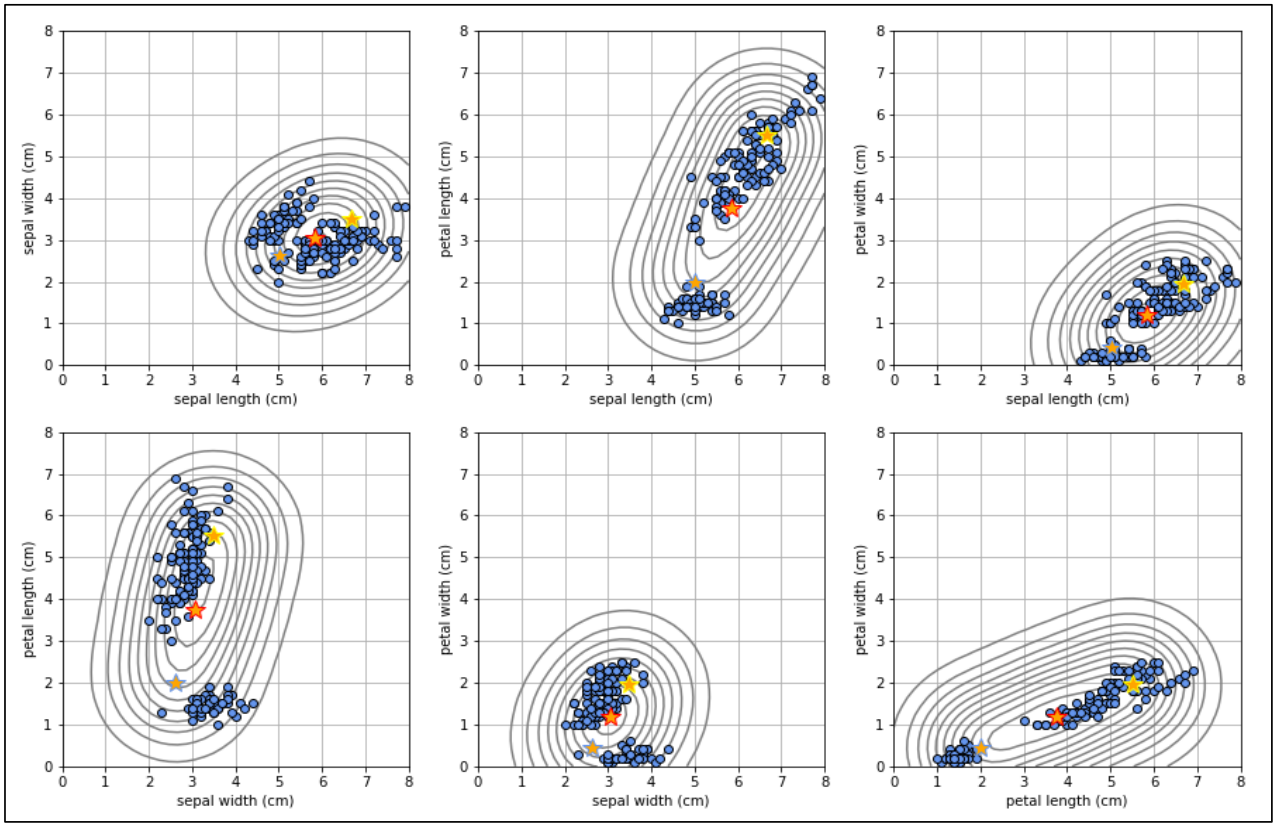
・混合係数 π_k　：割合が等しいものとして初期化
$\pi_k=1/K$ (for k=1～K-1)、　$\pi_K=1-\sum_{k=1}^{K-1}\pi_k$

・中心ﾍﾞｸﾄﾙ μ_k　：各成分d毎の平均m_dと標準偏差σ_dで初期化（図の☆印）
$\mu_{kd}= m_d + \sigma_d \times (k-(int)((K+1)/2))$

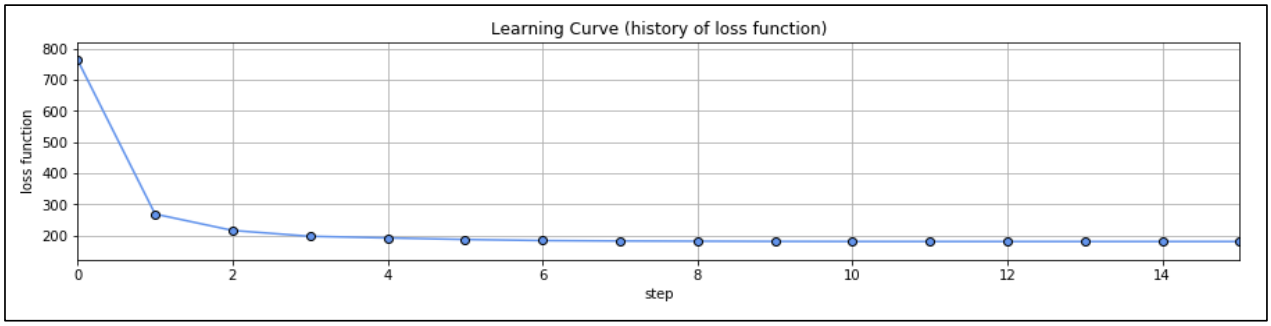
・共分散行列 Σ_k　：Σ_k を単位行列で初期化
$\Sigma_k = I$

・負担率 γ　：全てｸﾗｽﾀｰ1に所属しているとして初期化
$\gamma_{n1}=1$、
$\gamma_{nk}=0$ (for k≠1)

```
混合ガウス分布のﾓﾃﾞﾙ情報 ----------------
  ・ｸﾗｽﾀｰ数          =3
  ・説明変数の次元 =4
  ・混合係数 πk    =[0.33 0.33 0.34]
  ・中心座標 μk    =
[[5.01803204 2.62292237 1.99859593 0.43964071]
 [5.84333333 3.05733333 3.758      1.19933333]
 [6.66863463 3.4917443  5.51740407 1.95902596]]
  ・共分散行列 Σk =
[[[1 0 0 0]
  [0 1 0 0]
  [0 0 1 0]
  [0 0 0 1]]

 [[1 0 0 0]
  [0 1 0 0]
  [0 0 1 0]
  [0 0 0 1]]

 [[1 0 0 0]
  [0 1 0 0]
  [0 0 1 0]
  [0 0 0 1]]]
```



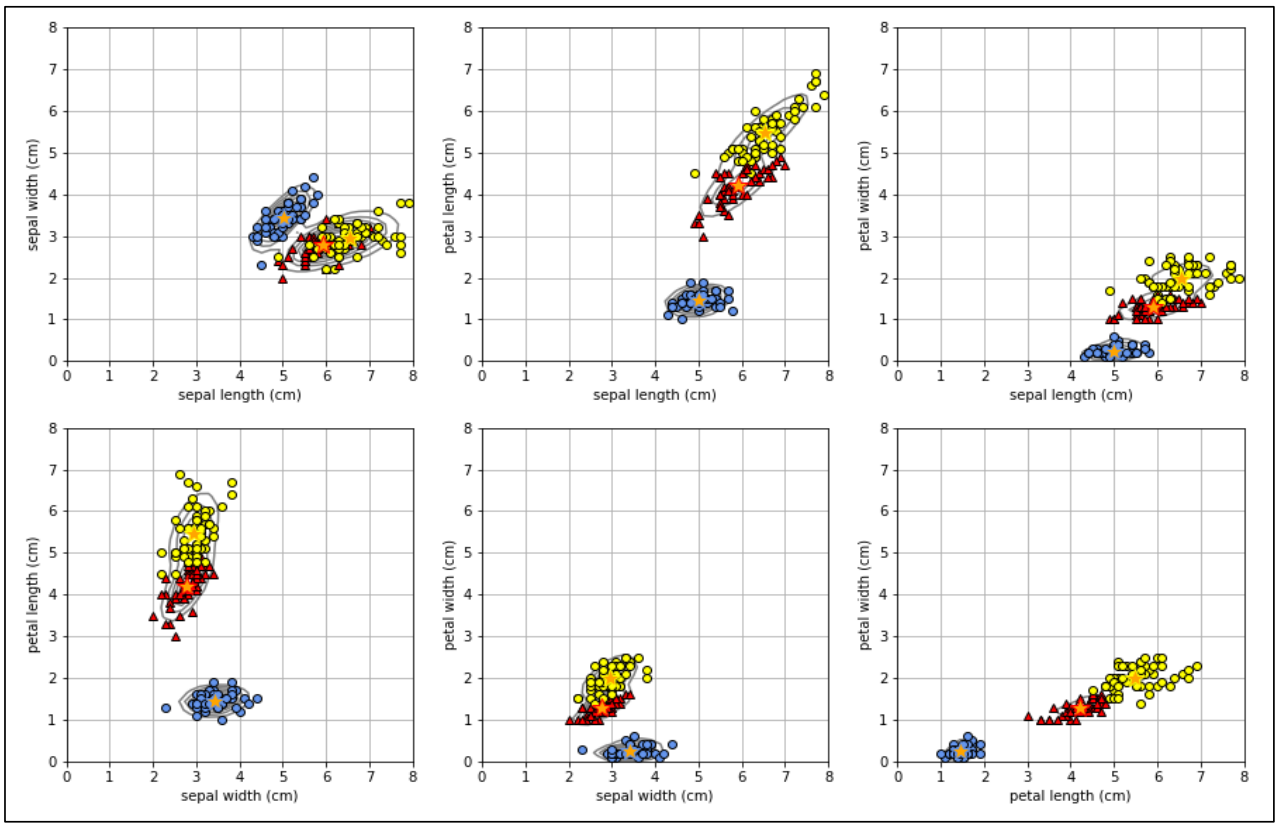
(2) EMｱﾙｺﾞﾘｽﾞﾑで、手順の繰り返しに伴う損失関数の変化の様子は、以下のようになりました。
最初の数ｽﾃｯﾌﾟで急速に誤差が減少し、10ｽﾃｯﾌﾟ前にほぼ収束していることが判ります。



(3) ｽﾃｯﾌﾟ=15で損失関数の減少が殆ど無いと判断し、
本ｱﾙｺﾞﾘｽﾞﾑの処理終了となりました(右図)。
この時の中心(☆印)と分布は
下図のようになりました。

共分散行列 Σ_k と下図の等高線を
初期状態のものと比べてわかるように、
分布が実ﾃﾞｰﾀに沿ったものに
絞り込まれていることが判ります。

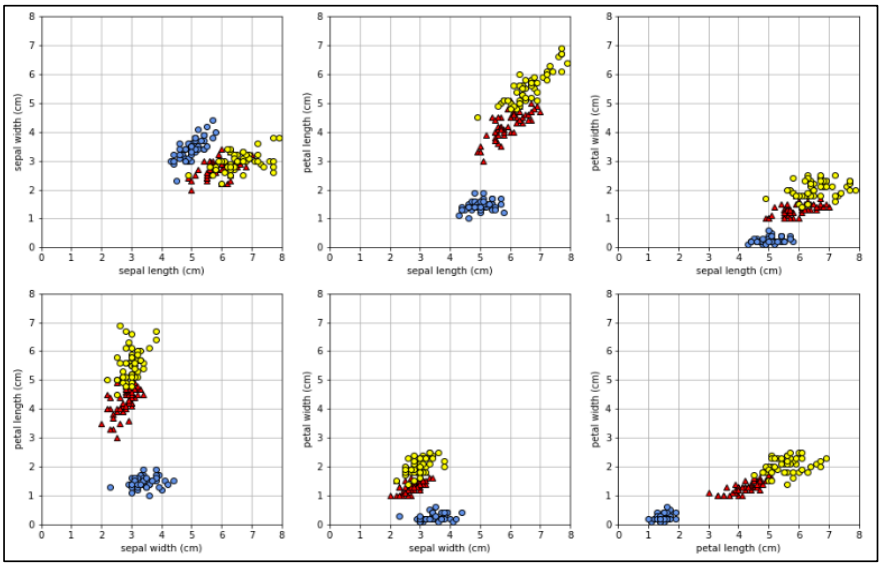
```
ｽﾃｯﾌﾟ=10 ------------------------
ｽﾃｯﾌﾟ=11 ------------------------
ｽﾃｯﾌﾟ=12 ------------------------
ｽﾃｯﾌﾟ=13 ------------------------
ｽﾃｯﾌﾟ=14 ------------------------
ｽﾃｯﾌﾟ=15 ------------------------
損失関数値の減少が殆どなくなったので処理終了。
混合ガウス分布のﾓﾃﾞﾙ情報 ----------------
   ・ｸﾗｽﾀｰ数          =3
   ・説明変数の次元 =4
   ・混合係数 πk    =[0.33333333 0.29960127 0.3670654 ]
   ・中心座標 μk    =
[[5.006      3.428      1.462      0.246     ]
 [5.91531914 2.77787664 4.20225319 1.29723971]
 [6.54496327 2.94882413 5.48040293 1.98514672]]
   ・共分散行列 Σk =
[[[0.121764   0.097232   0.016028   0.010124  ]
  [0.097232   0.140816   0.011464   0.009112  ]
  [0.016028   0.011464   0.029556   0.005948  ]
  [0.010124   0.009112   0.005948   0.010884  ]]

 [[0.27533679 0.09686941 0.18477615 0.05444927]
  [0.09686941 0.09262715 0.09111566 0.04299502]
  [0.18477615 0.09111566 0.2009098  0.06109855]
  [0.05444927 0.04299502 0.06109855 0.03205425]]

 [[0.38707212 0.09220746 0.30267949 0.06146615]
  [0.09220746 0.11034952 0.08419156 0.05595375]
  [0.30267949 0.08419156 0.32735406 0.07414696]
  [0.06146615 0.05595375 0.07414696 0.08559103]]]
```



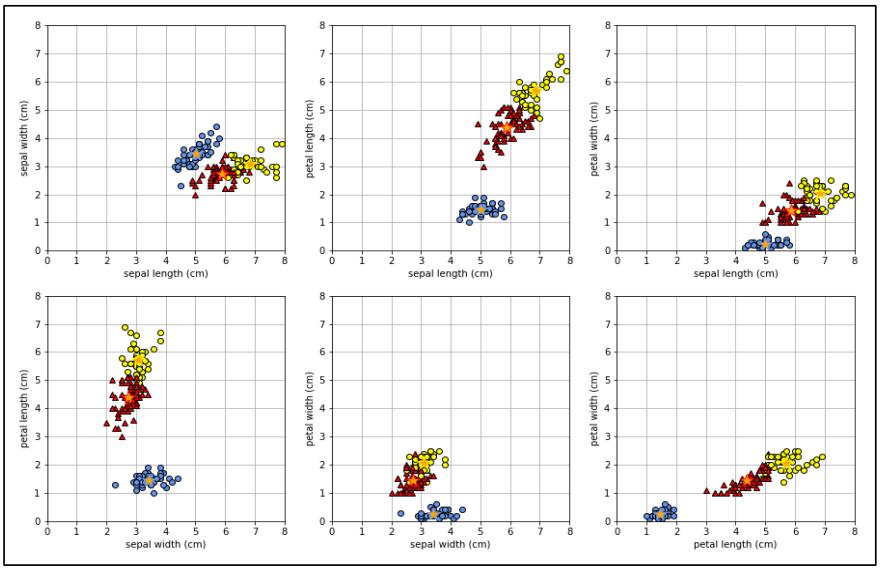
(結果の比較)

・「(3.1) 使用するﾃﾞｰﾀ」で紹介した元々のﾃﾞｰﾀ分布(下図)と比較すると、
「versicolor (黄○)」と「virginica(赤△)」の境界付近の所属ｸﾗｽﾀｰに差異が見られるものの、
K-means法より混合ｶﾞｳｽﾓﾃﾞﾙの方が、所属ｸﾗｽﾀｰの判別の精度が高いことが見てわかります。

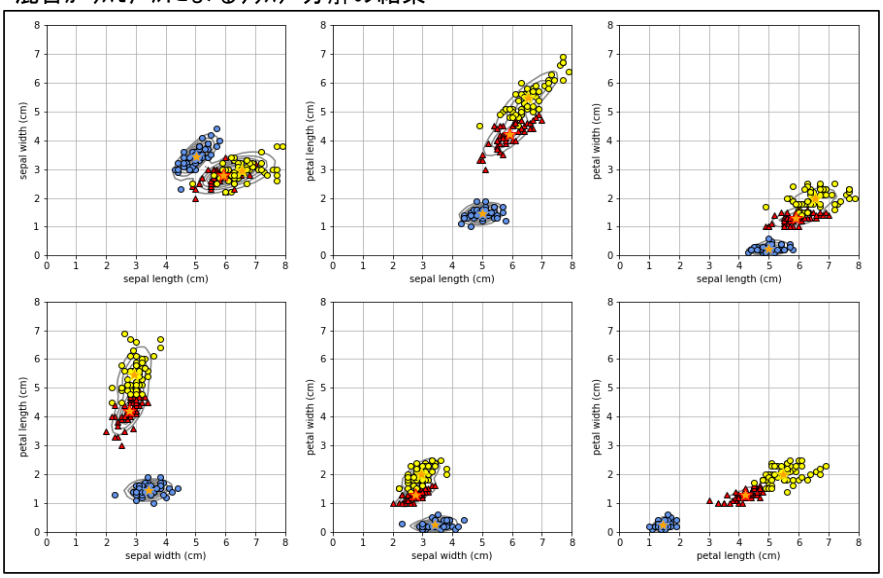
(3.1) 元々のﾃﾞｰﾀ分布



(3.2) K-means法によるｸﾗｽﾀｰ分解の結果



(3.3) 混合ｶﾞｳｽﾓﾃﾞﾙによるｸﾗｽﾀｰ分解の結果



(ﾘｽﾄ10-(03)-4_Scikit-learn Iris data の 混合ｶﾞｳｽﾓﾃﾞﾙによるｸﾗｽﾀ分解)

```
#**************************************************************************
# ﾘｽﾄ10-(03)-4_Scikit-learn Iris data の 混合ｶﾞｳｽﾓﾃﾞﾙによるｸﾗｽﾀ分解
#**************************************************************************
from sklearn import datasets
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# display range
DISP_MIN = 0
DISP_MAX = 8

# 分布表示の為の一辺あたりの格子点数
MESH_NO = 40

#-------------------------------------------------------------------------*/
# (添え字についての説明)                                                  */
#   k=0～K-1:ｸﾗｽﾀ添字 (K:ｸﾗｽﾀｰ総数)                                       */
#   d=0～D-1:特徴量添字  (D:ﾃﾞｰﾀの次元数)                                  */
#   n=0～N-1:ﾃﾞｰﾀ添字  (N:ﾃﾞｰﾀ総数)                                       */
#-------------------------------------------------------------------------*/

#*************************************************************************/
# fGauss                                                                 */
# 多変量ガウス関数（多変量正規分布）                                      */
#-------------------------------------------------------------------------*/
# 引数：                                                                 */
#   x[,]      :ﾃﾞｰﾀ：x[n,d]                                              */
#   muk[]     :着目ｸﾗｽﾀｰの中心座標μk：mu[d]                              */
#   sigmak[,] :着目ｸﾗｽﾀｰの共分散行列Σk：sigmak[d,d]                       */
#-------------------------------------------------------------------------*/
# 戻り値:                                                                */
#  (1) ガウス関数の値                                                    */
#*************************************************************************/
def fGauss(x, muk, sigmak):
    N, D = x.shape

    # マハラノビス距離 (Mahalanobis' Distance) の2乗計算
    invSigmak = np.linalg.inv(sigmak)
    xMinusMu = x - muk
    xMuDotInvS = np.dot(xMinusMu, invSigmak)
    xMDist2 = np.zeros(N)
    for d in range(D):
        xMDist2 = xMDist2 + xMuDotInvS[:,d] * xMinusMu[:,d]

    # 多変量ガウス関数の値
    fact1 = 1 / np.sqrt(((2*np.pi)**D) * np.linalg.det(sigmak))
    p = fact1 * np.exp(- xMDist2 / 2)
    return p

#*************************************************************************/
# fMixGauss                                                              */
# 多変量混合ガウス関数                                                    */
#-------------------------------------------------------------------------*/
# 引数：                                                                 */
#  x          :ﾃﾞｰﾀ：x[n,d]                                              */
#  piList     :全ｸﾗｽﾀの混合係数πkのﾘｽﾄ：piList[k]                         */
#  muList     :全ｸﾗｽﾀの中心座標μkのﾘｽﾄ：muList[k,d]                       */
#  sigmaList :全ｸﾗｽﾀの共分散行列Σkのﾘｽﾄ：sigmaList[k,d,d]                 */
#-------------------------------------------------------------------------*/
# 戻り値:                                                                */
#  (1) 混合ガウス関数の値  p[n,k]                                        */
#*************************************************************************/
def fMixGauss(x, piList, muList, sigmaList):
```

```
    N, D = x.shape
    K = len(piList)
    p = np.zeros((N,K))

    for k in range(K):
        p[:,k] = p[:,k] + piList[k] * fGauss(x, muList[k,:], sigmaList[k,:,:])
    return p

#*****************************************************************************/
# fLossFunction                                                              */
# 混合ガウスﾓﾃﾞﾙの損失関数（対数尤度に-1を掛けたもの）                          */
#-----------------------------------------------------------------------------*/
# 引数：                                                                     */
#  x          :ﾃﾞｰﾀ：x[n,d]                                                  */
#  piList     :全ｸﾗｽﾀの混合係数πkのﾘｽﾄ：piList[k]                            */
#  muList     :全ｸﾗｽﾀの中心座標μkのﾘｽﾄ：muList[k,d]                          */
#  sigmaList :全ｸﾗｽﾀの共分散行列Σkのﾘｽﾄ：sigmaList[k,d,d]                    */
#-----------------------------------------------------------------------------*/
# 戻り値:                                                                    */
#  (1) 混合ガウスﾓﾃﾞﾙの損失関数の値                                           */
#*****************************************************************************/
def fLossFunction(x, piList, muList, sigmaList):
    N, D = x.shape
    K = len(piList)

    # 混合ｶﾞｳｽ分布での各点での確率
    y = np.zeros((N,K))
    for k in range(K):
        y[:,k] = fGauss(x, muList[k,:], sigmaList[k,:,:])

    # 対数尤度
    logLikelihood = 0
    for n in range(N):
        wk = 0
        for k in range(K):
            wk = wk + piList[k] * y[n,k]
        logLikelihood = logLikelihood + np.log(wk)

    # 誤差関数
    loss = - logLikelihood
    return loss

#*****************************************************************************/
# fEStep                                                                     */
# 現時点でのπ、μ、Σを用いて、負担率γを計算する。                            */
#-----------------------------------------------------------------------------*/
# 引数：                                                                     */
#  x[,]           :ﾃﾞｰﾀ：x[n,d]                                              */
#  piList[]       :全ｸﾗｽﾀの混合係数πkのﾘｽﾄ：piList[k]                        */
#  muList[,]      :全ｸﾗｽﾀの中心座標μkのﾘｽﾄ：muList[k,d]                      */
#  sigmaList[,,] :全ｸﾗｽﾀの共分散行列Σkのﾘｽﾄ：sigmaList[k,d,d]               */
#-----------------------------------------------------------------------------*/
# 戻り値:                                                                    */
#  (1) 負担率γ[n,k]                                                         */
#*****************************************************************************/
def fEStep(x, piList, muList, sigmaList):
    # ﾃﾞｰﾀのｻｲｽﾞ情報
    N, D = x.shape
    K = len(piList)

    # 負担率γ
    Gnew = np.zeros((N,K))

    # 各ﾃﾞｰﾀの混合ｶﾞｳｽﾓﾃﾞﾙの分布関数値
    p = fMixGauss(x, piList, muList, sigmaList)
```

```
    # 各ﾃﾞｰﾀ毎に負担率γを算出
    for n in range(0, N):
        wk = np.zeros(K)
        for k in range(0, K):
            wk[k] = p[n,k]
        Gnew[n, :] = wk / np.sum(wk)

    return Gnew

#*****************************************************************************/
# fMStep                                                                     */
# 現時点での負担率γを用いて、πk、μk、Σkを計算する。                          */
#-----------------------------------------------------------------------------*/
# 引数：                                                                     */
#  x[,]          :ﾃﾞｰﾀ：x[n,d]                                               */
#  gammaList[,]  :負担率：γ[n,k]                                            */
#-----------------------------------------------------------------------------*/
# 戻り値:                                                                    */
#  (1) piList[]       :全ｸﾗｽﾀの混合係数πkのﾘｽﾄ：piList[k]                     */
#  (2) muList[,]      :全ｸﾗｽﾀの中心座標μkのﾘｽﾄ：muList[k,d]                   */
#  (3) sigmaList[,,] :全ｸﾗｽﾀの共分散行列Σkのﾘｽﾄ：sigmaList[k,d,d]           */
#*****************************************************************************/
def fMStep(x, gammaList):
    # ﾃﾞｰﾀのｻｲｽﾞ情報
    N, D = x.shape
    N, K = gammaList.shape

    # 負担率の総和 Rk を計算
    rkList = np.zeros(K)
    for k in range(0, K):
        for n in range(0, N):
            rkList[k] = rkList[k] + gammaList[n,k]

    # 混合係数πk を更新
    piList = np.zeros(K)
    for k in range(0, K):
        piList[k] = rkList[k] / N

    # 中心ﾍﾞｸﾄﾙμk を更新
    muList = np.zeros((K,D))
    for k in range(0, K):
        for n in range(0, N):
            muList[k,:] =  muList[k,:] + (gammaList[n,k] * x[n,:])
        muList[k,:] = muList[k,:] / rkList[k]

    # 共分散行列Σk を更新
    sigmaList = np.zeros((K,D,D))
    wk = np.zeros((1,D))
    for k in range(0, K):
        for n in range(0, N):
            for d in range(0,D):
                wk[0,:] = x[n,:] - muList[k,:]
            sigmaList[k,:,:] = sigmaList[k,:,:] + (gammaList[n,k] * np.dot(wk.T,wk))
        sigmaList[k,:,:] = sigmaList[k,:,:] / rkList[k]

    return piList, muList, sigmaList

#*****************************************************************************/
# fPlotLossHistory                                                           */
# 損失関数値（対数尤度に-1を掛けたもの)の時系列を表示                         */
#-----------------------------------------------------------------------------*/
# 引数：                                                                     */
#   lossHist :「損失関数値」の履歴  lossHist[j] (j=0～histMax:履歴添字) */
#   hmax     :「損失関数値」の履歴の最終 添字                                */
```

```
#----------------------------------------------------------------------------*/
# 戻り値：  なし                                                             */
#*****************************************************************************/
def fPlotLossHistory(lossHist, hmax):
    # 横軸
    jstep = np.zeros(hmax+1)
    for t in range(0, len(jstep)):
        jstep[t] = t

    # 縦軸表示範囲
    yrange = np.max(lossHist[:hmax+1]) - np.min(lossHist[:hmax+1])
    ymin = np.min(lossHist[:hmax+1]) - yrange * 0.1
    ymax = np.max(lossHist[:hmax+1]) + yrange * 0.1

    # 表示
    plt.figure(figsize=(15, 3))
    plt.title("Learning Curve (history of loss function)")
    plt.plot(jstep, lossHist[:hmax+1], marker='o', linestyle='-',
             markeredgecolor='black', color='cornflowerblue')
    plt.xlim(0, hmax)
    plt.ylim(ymin, ymax)
    plt.xlabel("step")
    plt.ylabel("loss function")
    plt.grid(True)
    plt.show()
    return

#*****************************************************************************/
# fGetMeshGauss                                                              */
# 混合ガウス分布の格子点上の値を得る                                         */
#----------------------------------------------------------------------------*/
# 引数：                                                                     */
#   K             :ｸﾗｽﾀｰ総数                                               */
#   D             :ﾃﾞｰﾀの次元数                                             */
#   piList[]      :全ｸﾗｽﾀの混合係数πkのﾘｽﾄ：piList[k]                       */
#   muList[,]     :全ｸﾗｽﾀの中心座標μkのﾘｽﾄ：muList[k,d]                     */
#   sigmaList[,,] :全ｸﾗｽﾀの共分散行列Σkのﾘｽﾄ：sigmaList[k,d,d]              */
#   xn            :輪郭表示時のﾌﾟﾛｯﾄ数（表示範囲を分割する個数）           */
#----------------------------------------------------------------------------*/
# 戻り値：                                                                   */
# (1) xcoord     :輪郭表示時の格子点座標値(1次元) xcoord(xn)                 */
# (2) xmeshGauss :格子点座標での混合ｶﾞｳｽ分布の値(4次元) xmeshGauss(xn,xn,xn,xn) */
#*****************************************************************************/
def fGetMeshGauss(K, D, piList, muList, sigmaList, xn):

    # 表示ﾒｯｼｭの格子点座標の値
    xcoord = np.linspace(DISP_MIN, DISP_MAX, xn)

    # 4次元の格子点座標
    xmeshNN = np.zeros((xn,xn,xn,xn,D))
    for x3 in range(xn):
        for x2 in range(xn):
            for x1 in range(xn):
                for x0 in range(xn):
                    xmeshNN[x3,x2,x1,x0, 3] = xcoord[x3]
                    xmeshNN[x3,x2,x1,x0, 2] = xcoord[x2]
                    xmeshNN[x3,x2,x1,x0, 1] = xcoord[x1]
                    xmeshNN[x3,x2,x1,x0, 0] = xcoord[x0]

    # 4次元の格子点座標での混合ｶﾞｳｽ分布の合成値
    xmeshNN = xmeshNN.reshape((xn*xn*xn*xn,D))
    xmeshGaussK = fMixGauss(xmeshNN, piList, muList, sigmaList)

    # ｸﾗｽﾀｰ分の混合ｶﾞｳｽ分布の合成
    xmeshGauss = np.zeros((xn*xn*xn*xn))
```

```
    for n in range(xn*xn*xn*xn):
        for k in range(K):
            xmeshGauss[n] = xmeshGauss[n] + xmeshGaussK[n,k]

    # 格子点での混合ｶﾞｳｽ分布の合成値
    xmeshGauss = xmeshGauss.reshape((xn,xn,xn,xn))

    return xcoord, xmeshGauss

#*************************************************************************/
# fPlotMixGaussContour                                                   */
# 混合ガウス分布の輪郭線を描画する                                       */
#-------------------------------------------------------------------------*/
# 引数：                                                                 */
#   plt         :ﾌﾟﾛｯﾄｵﾌﾞｼﾞｪｸﾄ                                         */
#   xn          :輪郭表示時のﾌﾟﾛｯﾄ数（表示範囲を分割する個数）           */
#   xcoord      :輪郭表示時の格子点座標値(1次元) xcoord(xn)              */
#   xmeshGauss :格子点座標での混合ｶﾞｳｽ分布の値(4次元) xmeshGauss(xn,xn,xn,xn) */
#   axisX       :輪郭表示時のX軸の次元番号 (0～D-1)                      */
#   axisY       :輪郭表示時のY軸の次元番号 (0～D-1)                      */
#-------------------------------------------------------------------------*/
# 戻り値：  なし                                                         */
#*************************************************************************/
def fPlotMixGaussContour(plt, xn, xcoord, xmeshGauss, axisX, axisY):

    # 着目平面への射影 (Z-buffering)
    xplaneGauss = np.zeros((xn,xn))
    for yi in range(xn):
        for xi in range(xn):
            for z1i in range(xn):
                for z2i in range(xn):
                    xadd = 0.0
                    if( (axisX==0) and (axisY==1) ):
                        xadd = xmeshGauss[z1i,z2i,yi,xi]
                    elif( (axisX==0) and (axisY==2) ):
                        xadd = xmeshGauss[z1i,yi,z2i,xi]
                    elif( (axisX==0) and (axisY==3) ):
                        xadd = xmeshGauss[yi,z1i,z2i,xi]
                    elif( (axisX==1) and (axisY==2) ):
                        xadd = xmeshGauss[z1i,yi,xi,z2i]
                    elif( (axisX==1) and (axisY==3) ):
                        xadd = xmeshGauss[yi,z1i,xi,z2i]
                    elif( (axisX==2) and (axisY==3) ):
                        xadd = xmeshGauss[yi,xi,z1i,z2i]

                    xplaneGauss[yi,xi] = xplaneGauss[yi,xi] + xadd

    # 表等高線表示
    plt.contour(xcoord, xcoord, xplaneGauss, 10, colors='gray')
    return

#*************************************************************************/
# fDPlotAdnDraw                                                          */
# 各点をプロットし、混合ガウス分布の輪郭線を描画する                     */
#-------------------------------------------------------------------------*/
# 引数：                                                                 */
#   N             :ﾃﾞｰﾀ総数                                            */
#   K             :ｸﾗｽﾀｰ総数                                           */
#   D             :ﾃﾞｰﾀの次元数                                        */
#   piList[]      :全ｸﾗｽﾀの混合係数πkのﾘｽﾄ：piList[k]                   */
#   muList[,]     :全ｸﾗｽﾀの中心座標μkのﾘｽﾄ：muList[k,d]                 */
#   sigmaList[,,] :全ｸﾗｽﾀの共分散行列Σkのﾘｽﾄ：sigmaList[k,d,d]          */
#   gamma[,]      :負担率：γ[n,k]                                      */
#-------------------------------------------------------------------------*/
# 戻り値：  なし                                                         */
```

```
#*****************************************************************************/
def fDPlotAdnDraw(N, K, D, piList, muList, sigmaList, gamma):
    # plot Iris data
    axislist = [[0,1],[0,2],[0,3],[1,2],[1,3],[2,3]]
    collist = ['cornflowerblue', 'red', 'yellow']
    mrklist = ['o', '^', 'o']

    plt.figure(figsize=(15, 10))

    # 格子点での混合ｶﾞｳｽ分布の値
    xcoord, xmeshGauss = fGetMeshGauss(K, D, piList, muList, sigmaList, MESH_NO)

    # 座標の組み合わせ毎に表示
    for xy in range(0,len(axislist)):
        plt.subplot(2,3, xy+1)
        axisX = axislist[xy][0]
        axisY = axislist[xy][1]

        # 混合ｶﾞｳｽ分布の輪郭を描画
        fPlotMixGaussContour(plt, MESH_NO, xcoord, xmeshGauss, axisX, axisY)

        # 各ﾃﾞｰﾀをﾌﾟﾛｯﾄ
        for n in range(0, N):
            k = np.argmax(gamma[n,:])
            plt.plot(iris_dt.data[n, axisX],
                     iris_dt.data[n, axisY],
                     mrklist[k], linestyle='None',
                     markeredgecolor='black', color=collist[k])

        # 各ｸﾗｽﾀｰ中心をﾌﾟﾛｯﾄ
        for k in range(0, K):
            plt.plot(muList[k, axisX],
                     muList[k, axisY],
                     '*', linestyle='None', markersize=15,
                     markeredgecolor=collist[k], color='orange')

        plt.xlim(DISP_MIN, DISP_MAX)
        plt.ylim(DISP_MIN, DISP_MAX)
        plt.xlabel(iris_dt.feature_names[axisX])
        plt.ylabel(iris_dt.feature_names[axisY])
        plt.grid(True)

    plt.show()

#*****************************************************************************/
# fPrintMixGaussModelInfo                                                    */
# 混合ガウス分布のﾓﾃﾞﾙ情報を印字する                                          */
#----------------------------------------------------------------------------*/
# 引数：                                                                     */
#   K             :ｸﾗｽﾀｰ数                                                   */
#   D             :説明変数の次元                                            */
#   piList[]      :全ｸﾗｽﾀの混合係数πkのﾘｽﾄ：piList[k]                        */
#   muList[,]     :全ｸﾗｽﾀの中心座標μkのﾘｽﾄ：muList[k,d]                       */
#   sigmaList[,,] :全ｸﾗｽﾀの共分散行列Σkのﾘｽﾄ：sigmaList[k,d,d]                */
#----------------------------------------------------------------------------*/
# 戻り値：  なし                                                             */
#*****************************************************************************/
def fPrintMixGaussModelInfo(K, D, piList, muList, sigmaList):
    print("混合ガウス分布のﾓﾃﾞﾙ情報 ----------------")
    print("  ・ｸﾗｽﾀｰ数        ＝{0}".format(K))
    print("  ・説明変数の次元 ＝{0}".format(D))
    print("  ・混合係数 πk    ＝{0}".format(piList))
    print("  ・中心座標 μk    ＝¥r¥n{0}".format(muList))
    print("  ・共分散行列 Σk ＝¥r¥n{0}".format(sigmaList))
```

```
#*************************************************************************/
# 本体処理                                                               */
#*************************************************************************/

#-------------------------------------------------------------------------
# Iris ﾃﾞｰﾀをﾛｰﾄﾞし、想定するｸﾗｽﾀ数Kを与えます
#-------------------------------------------------------------------------
# load Iris data
iris_dt = datasets.load_iris()
X = iris_dt.data
N, D = X.shape
K = 3   # 想定するｸﾗｽﾀｰ数

#-------------------------------------------------------------------------
# (手順1：初期設定) 混合係数π、中心ﾍﾞｸﾄﾙμ、共分散行列Σ、負担率γ
#-------------------------------------------------------------------------
# ｶﾞｳｽ分布の混合係数の初期化
PiList = np.array([0.33, 0.33, 0.34])

# k番目のｶﾞｳｽ分布Nkの中心ﾍﾞｸﾄﾙμの初期化 (特徴量平均-σ,±0,+σで初期化)
MuList = np.ones([K,D])
Msigma = np.zeros(D)
Mmean = np.zeros(D)
for d in range(0,D):
    Mmean[d] = np.mean(iris_dt.data[:, d])
    Msigma[d] = np.std(iris_dt.data[:, d])
    for k in range(0,K):
        MuList[k][d] = Mmean[d] + Msigma[d] * (k+1-int((K+1)/2))

# k番目のｶﾞｳｽ分布Nkの共分散行列Σkの初期化
#  (説明変数が4個なので、4×4の単位行列をｸﾗｽﾀｰ数分作成して初期化)
SigmaList = np.array([
    [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]],
    [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]],
    [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]])

# 負担率γを初期化 (ｸﾗｽﾀｰ数分)
GammaList = np.c_[np.ones((N,1)), np.zeros((N,2))]

# 各ｸﾗｽﾀｰ中心と、各ﾃﾞｰﾀの分布を表示 (初期状態)
fPrintMixGaussModelInfo(K, D, PiList, MuList, SigmaList)
fDPlotAdnDraw(N, K, D, PiList, MuList, SigmaList, GammaList)

#-------------------------------------------------------------------------
# E Step, M Step を収束するまで、または最大回数に達するまで繰り返す
#-------------------------------------------------------------------------
# 最大繰り返し回数
REPT_MAX = 200

# 「損失関数」の履歴と減少率の境界値
LOSS_LIMIT = 0.00001  # 直前の損失関数値との差異の割合がこれ以下になったら収束とみなす
lossHist = np.zeros(REPT_MAX)
hmax = 0

for j in range(0,REPT_MAX):
    print( "ｽﾃｯﾌﾟ={0} ------------------------".format(j))
    if( j == REPT_MAX-1 ):
        print( "最大繰り返し回数に達したので処理終了。")
        break

    #---------------------------------------------------------------------
    # 現時点でのπ、μ、Σから損失関数値を計算
    #---------------------------------------------------------------------
    hmax = j
```

```
    lossHist[j] = fLossFunction(X, PiList, MuList, SigmaList)
    if( lossHist[j] == 0):
        print( "損失関数値が0になったので処理終了。")
        break
    elif( j >= 1 ):
        lossDiff = abs((lossHist[j] - lossHist[j-1]) / lossHist[j-1])
        if(lossDiff < LOSS_LIMIT):
            print( "損失関数値の減少が殆どなくなったので処理終了。")
            break

    #---------------------------------------------------------------------
    # (手順2：E Step) 現時点でのπ、μ、Σを用いて、負担率γを計算
    #---------------------------------------------------------------------
    GammaNew = fEStep(X, PiList, MuList, SigmaList)
    if( (GammaList == GammaNew).all() ):
        print( "負担率γに変更が無かったので処理終了。")
        break
    else:
        GammaList = GammaNew

    #---------------------------------------------------------------------
    # (手順3：M Step) 現時点での負担率γを用いて、π、μ、Σを計算
    #---------------------------------------------------------------------
    PiListNew, MuListNew, SigmaListNew = fMStep(X, GammaNew)
    if( (PiList == PiListNew).all() and
        (MuList == MuListNew).all() and
        (SigmaList == SigmaListNew).all() ) :
        print( "混合係数π、中心ﾍﾞｸﾄﾙμ、共分散行列Σに変更が無かったので処理終了。")
        break
    else:
        PiList = PiListNew
        MuList = MuListNew
        SigmaList = SigmaListNew

#---------------------------------------------------------------------
# 終了時処理
#---------------------------------------------------------------------
# 各ｸﾗｽﾀｰ中心と、各ﾃﾞｰﾀの分布を表示（終了時）
fPrintMixGaussModelInfo(K, D, PiList, MuList, SigmaList)
fDPlotAdnDraw(N, K, D, PiList, MuList, SigmaList, GammaList)

# 「損失関数」の履歴を表示
fPlotLossHistory(lossHist, hmax)
```

【参考】

混合ｶﾞｳｽﾓﾃﾞﾙ

⇒「Pythonで動かして学ぶ！あたらしい機械学習の教科書」(2018年01月 翔泳社 伊藤真著)

⇒「ﾊﾟﾀｰﾝ認識と機械学習 下」(2019年03月 丸善 C.M.ﾋﾞｼｮｯﾌﾟ著)

多変量正規分布

⇒https://ja.wikipedia.org/wiki/多変量正規分布

(4) まとめ

・今回は「教師なし学習」について取り上げ、その例として、
「Scikit-learn (ｻｲｷｯﾄﾗｰﾝ)」で提供している「ｱﾔﾒ (Iris)」のﾃﾞｰﾀのｸﾗｽﾀｰ分析を
「K-means法」と「混合ｶﾞｳｽﾓﾃﾞﾙ」の2つの手法で試みました。

・実装はｱﾙｺﾞﾘｽﾞﾑが見えるように、出来合いのﾗｲﾌﾞﾗﾘをできるだけ使用しないで行いましたが、
実際は、Scikit-learn などの計算ﾗｲﾌﾞﾗﾘを使用するととｺｰﾃﾞｨﾝｸﾞ量も少なくて済むので、
現場ではそういう選択肢が良いでしょう。
何れにせよ、理論的背景を把握することにより、応用の幅が広がるのではないかと思います。

・教師無し学習には他にも多くの手法がありますので、これを踏み台にして、
自分も含めて、どんどん取り組んでいただきたいと思います。

以上です。

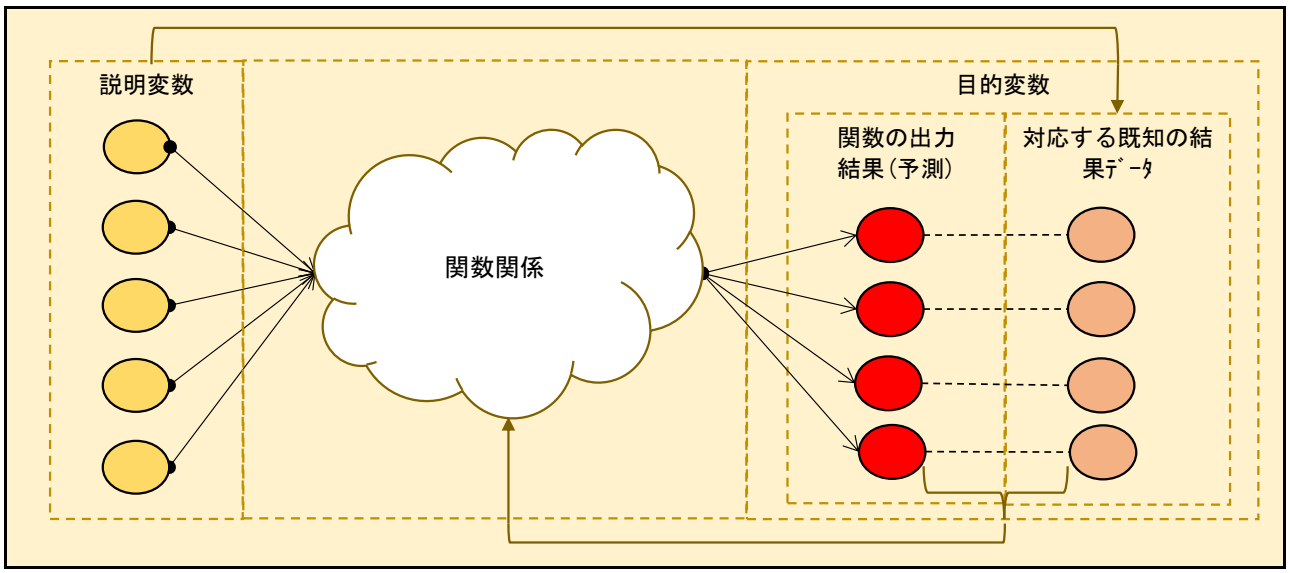
(5) 確認問題

(5.1) 教師あり学習と教師なし学習との差異

以下の空欄に最もあてはまる語句を選択肢から選び、その記号を回答欄に記入してください。

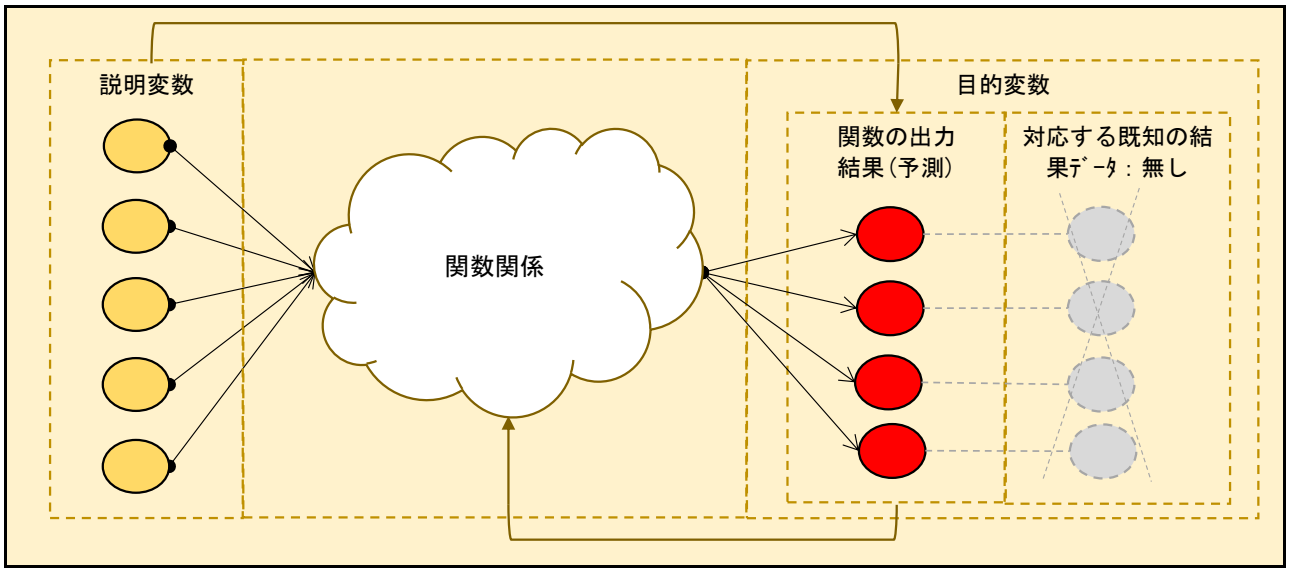
・何かの因果関係を探る場合、統計ﾓﾃﾞﾙでは、原因となる変数群を元に、
結果となる変数群を説明しようと試みます。
原因となる変数のことを「説明変数（ｾﾂﾒｲﾍﾝｽｳ、explanatory variable)」と言い、
結果となる変数のことを「目的変数（ﾓｸﾃｷﾍﾝｽｳ、response variable)」と言います。

・(1.1) _____ は　異なるﾃﾞｰﾀ間の関係性を、
一方を入力（説明変数）、他方を出力（目的変数）とする関数関係として記述し、
関数の出力を既知の結果ﾃﾞｰﾀ（目的変数の既知ﾃﾞｰﾀ）に近づけるように、関数関係を学習する方法です。
この場合、関数の出力結果(予測)に対して、既知の結果ﾃﾞｰﾀが教師ﾃﾞｰﾀの役割をします（下図参照）。



・(1.1) _____ において、目的変数に対応する確率変数が連続確率変数の場合、関数関係を推定する学習を
(1.2) _____ と呼びます。
・(1.1) _____ において、目的変数に対応する確率変数が離散確率変数の場合、関数関係を推定する学習を
(1.3) _____ と呼びます。

・入力変数（説明変数）に対する、出力変数（目的変数）が与えられていない状態で、
入力ﾃﾞｰﾀ自体の背後にある性質や特徴や構造などを学習することを、
(1.4) _____ と言います（下図参照）。



・「ｸﾗｽﾀｰ (Cluster)」とは、色・形・大きさといった特徴が類似したﾃﾞｰﾀ分布の塊を言います。
(1.5)＿＿＿＿＿＿＿＿ はﾃﾞｰﾀ分布の特徴からｸﾗｽﾀｰを見つけて、ﾃﾞｰﾀを分類することです。

・「教師あり学習」の (1.3)＿＿＿＿＿＿＿＿ は、分類済みの入力ﾃﾞｰﾀで分類方法を学習するのに対し、
「教師なし学習」の (1.5)＿＿＿＿＿＿＿＿ は、未分類の似たﾃﾞｰﾀを集めてﾃﾞｰﾀ構造を発見します。
「ｸﾗｽﾀﾘﾝｸﾞ」のｱﾙｺﾞﾘｽﾞﾑとしては、「k平均法 (ｹｲﾍｲｷﾝﾎｳ、K-means clustering)」や
「混合ｶﾞｳｽ分布 (ｺﾝｺﾞｳｶﾞｳｽﾌﾞﾝﾌﾟ、Gaussian mixture models、GMM)」などがあります。

(選択肢)

(a) 「強化学習 (ｷｮｳｶｶﾞｸｼｭｳ、Reinforcement learning)」
(b) 「教師あり学習 (ｷｮｳｼｱﾘｶﾞｸｼｭｳ, Supervised learning)」
(c) 「教師なし学習 (ｷｮｳｼﾅｼｶﾞｸｼｭｳ、Unsupervised Learning)」
(d) 「転移学習 (ﾃﾝｲｶﾞｸｼｭｳ、Transfer Learning)」
(e) 「因子分析 (ｲﾝｼﾌﾞﾝｾｷ、factor analysis、FA)」
(f) 「次元削減 (ｼﾞｹﾞﾝｻｸｹﾞﾝ、Dimensionality Reduction)」
(g) 「ｸﾗｽﾀﾘﾝｸﾞ (Clustering)」
(h) 「分類 (ﾌﾞﾝﾙｲ、Classification)」
(i) 「回帰 (ｶｲｷ、Regression)」

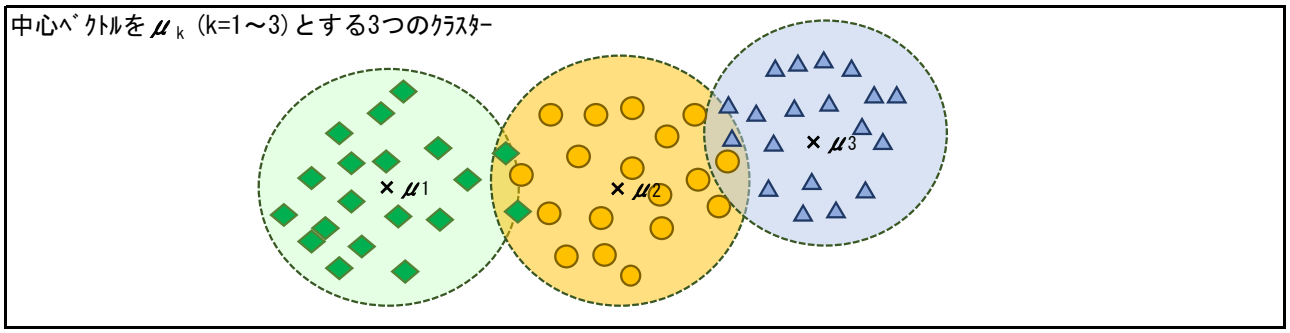
(回答)

(1.1)	
(1.2)	
(1.3)	
(1.4)	
(1.5)	

(5.2) K-means法

以下の空欄に最もあてはまる語句を選択肢から選び、その記号を回答欄に記入してください。

・「ｸﾗｽﾀﾘﾝｸﾞ」のｱﾙｺﾞﾘｽﾞﾑとして代表的な「K-means法 (ｹｲﾐｰﾝｽﾞﾎｳ、K-平均法、K-means clustering)」
の「K-means」は「k個の平均値」という意味です。
これは、各ﾃﾞｰﾀをK個のｸﾗｽﾀｰの何れかに所属させるｱﾙｺﾞﾘｽﾞﾑで、
各ﾃﾞｰﾀをｸﾗｽﾀｰ中心(平均値)からの距離が最短のものに所属させます。



・K-means法のｱﾙｺﾞﾘｽﾞﾑは、初めに k番目のｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙ μ_k (k=1～K：ｸﾗｽﾀｰ数)の初期値を与え、
全ﾃﾞｰﾀについて所属ｸﾗｽﾀｰを(ｸﾗｽ指示変数Rとして)仮決めしておいた上で、
(1st) ｸﾗｽﾀｰkに所属する全ﾃﾞｰﾀでｸﾗｽﾀｰ中心 μ_k を再計算する
(2nd) 全ﾃﾞｰﾀとｸﾗｽﾀｰ中心 μ_k 間の距離を再計算して所属を再評価する(ｸﾗｽ指示変数Rを更新する)
という手順(1st)、(2nd)を計算が収束するまで繰り返す、というものです。

ﾃﾞｰﾀのｸﾗｽﾀｰへの所属を表すｸﾗｽ指示変数Rは、以下のようなものです：

ｸﾗｽ指示変数R

・n番目のﾃﾞｰﾀ $\boldsymbol{x}_n$(n=1～N：ﾃﾞｰﾀ数)の所属ｸﾗｽﾀｰ k(k=1～K：ｸﾗｽﾀ数)を、
(2.1) でﾍﾞｸﾄﾙ $\boldsymbol{r}_n$ で表現します。
更に、これを全てのﾃﾞｰﾀについて行い、ｸﾗｽ指示変数Rというﾏﾄﾘｸｽで表現します。

$$\boldsymbol{r}_n = (r_{n1},\ r_{n2},\ \cdots,\ r_{nK}) \qquad r_{nk} = \begin{cases} 1: n番目のﾃﾞｰﾀ\ \boldsymbol{x}_n\ がｸﾗｽﾀｰkに属している \\ 0: n番目のﾃﾞｰﾀ\ \boldsymbol{x}_n\ がｸﾗｽﾀｰkに属していない \end{cases}$$

$$\mathbf{R} = \begin{pmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \\ \vdots \\ \boldsymbol{r}_N \end{pmatrix} = \begin{pmatrix} r_{11}, & r_{12}, & \cdots, & r_{1K} \\ r_{21}, & r_{22}, & \cdots, & r_{2K} \\ \vdots & \vdots & & \vdots \\ r_{N1}, & r_{N2}, & \cdots, & r_{NK} \end{pmatrix}$$

・全てのﾃﾞｰﾀ $\boldsymbol{x}_n$(n=1～N：ﾃﾞｰﾀ数)について、
ｸﾗｽ指示変数Rで示される所属ｸﾗｽﾀｰkの中心 μ_k との二乗距離の総和をJとして計算します。
これを(2.2)＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿ と言います。
このJが K-means法の損失関数で、これを最小化するようにﾓﾃﾞﾙを作成します。

K-means法の損失関数

$$J = \sum_{n=1}^{N} \sum_{d=1}^{D} (x_{nd} - \mu_{kd})^2$$

k ：n番目のﾃﾞｰﾀの所属ｸﾗｽﾀｰ
D ：説明変数の次元
$\boldsymbol{x}_n = (x_{n1},\ x_{n2},\ \cdots,\ x_{nD})$ ：n番目のﾃﾞｰﾀ (n=1～N：ﾃﾞｰﾀ数)
$\mu_k = (\mu_{k1},\ \mu_{k2},\ \cdots,\ \mu_{kD})$ ：k番目のｸﾗｽﾀｰの中心ﾍﾞｸﾄﾙ (k=1～K：ｸﾗｽﾀｰ数)

(選択肢)

(a) 「1-of-K 符号化法 (1-of-K coding scheme)」
(b) 「ﾊﾌﾏﾝﾌﾞﾛｯｸ符号化法 (Huffman block coding method)」
(c) 「尤度 (ﾕｳﾄﾞ、likelihood)」
(d) 「歪み尺度 (ﾕｶﾞﾐｼｬｸﾄﾞ、distortion measure)」

(回答)

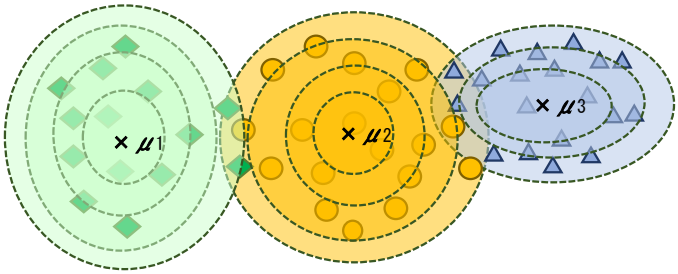
(2.1)	
(2.2)	

(5.3) 混合ｶﾞｳｽﾓﾃﾞﾙ

以下の空欄に最もあてはまる語句を選択肢から選び、その記号を回答欄に記入してください。

・ｸﾗｽﾀｰ分析で、ｸﾗｽﾀｰ境界付近に分布しているﾃﾞｰﾀが、どちらのｸﾗｽﾀｰに属しているのかは曖昧である為、
ｸﾗｽﾀｰへの所属確率(これを (3.1) といいます)で表現する方が適切です。
この表現としてﾃﾞｰﾀ分布がK個のｶﾞｳｽ分布から構成されるﾓﾃﾞﾙを導入します。
これが「混合ｶﾞｳｽﾓﾃﾞﾙ (ｺﾝｺﾞｳｶﾞｳｽﾓﾃﾞﾙ、Gaussian mixture model、GMM)」です。

混合ｶﾞｳｽﾓﾃﾞﾙは、K個の各ｸﾗｽﾀｰがｶﾞｳｽ分布しているとし、ﾃﾞｰﾀのｸﾗｽﾀへの所属確率で表現します



・「混合ｶﾞｳｽﾓﾃﾞﾙ」ではﾃﾞｰﾀの分布が、K個の(D次元)ｶﾞｳｽ分布(正規分布)の合成であるとします。
k番目(k=1～K：ｸﾗｽﾀｰ数)のｸﾗｽﾀｰはその中心位置がｶﾞｳｽ分布の中心 $\boldsymbol{\mu}_k$(D次元)で、
分布の広がりは共分散行列 $\boldsymbol{\Sigma}_k$(D×D の正方行列)で表現されます。
また、各ｶﾞｳｽ分布の大きさの比率を π_k(混合係数)で表現します。

混合ｶﾞｳｽﾓﾃﾞﾙのﾃﾞｰﾀ表現

(凡例)
- $\boldsymbol{x}$：ﾃﾞｰﾀの座標値 $(x_1, x_2, \cdots, x_D)$
- n：ﾃﾞｰﾀ番号(n=1～N：ﾃﾞｰﾀ数)
- k：ｸﾗｽﾀｰ番号(k=1～K：ｸﾗｽﾀｰ数)
- d：説明変数番号(d=1～D：説明変数の次元)

・混合ｶﾞｳｽﾓﾃﾞﾙの分布関数は次式で与えられます：

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \, N_k(\boldsymbol{x})$$

N_k：k番目のｶﾞｳｽ分布(D次元)

$$N_k = \{1/(2\pi)^{D/2}\} (\det \boldsymbol{\Sigma}_k)^{-1/2} \exp\{-(\boldsymbol{x}-\boldsymbol{\mu}_k)^T (\boldsymbol{\Sigma}_k)^{-1} (\boldsymbol{x}-\boldsymbol{\mu}_k)/2\}$$

π_k：k番目のｶﾞｳｽ分布 N_k の大きさの比率(混合係数)

$$\pi_k : \quad 0 \leqq \pi_k \leqq 1 \quad 、1 = \sum_{k=1}^{K} \pi_k$$

$\boldsymbol{\mu}_k$：k番目のｶﾞｳｽ分布 N_k の中心ﾍﾞｸﾄﾙ

$$\boldsymbol{\mu}_k = (\mu_{k1}, \ \mu_{k2}, \ \cdots, \ \mu_{kD})$$

$\boldsymbol{\Sigma}_k$：k番目のｶﾞｳｽ分布 N_k の広がり方(共分散行列)

$$\boldsymbol{\Sigma}_k = \begin{pmatrix} \sigma_{k11}, & \sigma_{k12}, & \cdots, & \sigma_{k1D} \\ \sigma_{k21}, & \sigma_{k22}, & \cdots, & \sigma_{k2D} \\ \vdots & \vdots & & \vdots \\ \sigma_{kD1}, & \sigma_{kD2}, & \cdots, & \sigma_{kDD} \end{pmatrix} \qquad \sigma_{kij} = \sigma_{kji}$$

・混合ｶﾞｳｽﾓﾃﾞﾙの分布関数により、n番目のﾃﾞｰﾀ $\boldsymbol{x}_n$ のｸﾗｽﾀｰk への所属確率 γ_{nk} を求めることができます。

混合ｶﾞｳｽﾓﾃﾞﾙでのn番目のﾃﾞｰﾀ $\boldsymbol{x}_n$ のｸﾗｽﾀｰk への所属確率 γ_{nk} の計算

・混合ｶﾞｳｽﾓﾃﾞﾙの分布関数により、
座標 $\boldsymbol{x}$ にある点の、ｸﾗｽﾀｰk への所属確率$p_k(\boldsymbol{x})$は次式により与えられます：

$$p_k(\boldsymbol{x}) = \pi_k N_k(\boldsymbol{x}) \ / \ \Sigma_{k=1}^{K} \pi_k N_k(\boldsymbol{x})$$

これにより、n番目のﾃﾞｰﾀ $\boldsymbol{x}_n$ のｸﾗｽﾀｰk への所属確率γ_{nk}は次式で与えられます：

$$\gamma_{nk} = p_k(\boldsymbol{x}_n) = \pi_k N_k(\boldsymbol{x}_n) \ / \ \Sigma_{k=1}^{K} \pi_k N_k(\boldsymbol{x}_n)$$

所属確率γ_{nk}を要素とする行列$\boldsymbol{\gamma}$を (3.1) と言います：

$$\boldsymbol{\gamma} = \begin{pmatrix} \gamma_{11}, & \gamma_{12}, & \cdots, & \gamma_{1K} \\ \gamma_{21}, & \gamma_{22}, & \cdots, & \gamma_{2K} \\ \vdots & \vdots & & \vdots \\ \gamma_{N1}, & \gamma_{N2}, & \cdots, & \gamma_{NK} \end{pmatrix}$$

γ_{nk}：n番目のﾃﾞｰﾀ$\boldsymbol{x}_n$がｸﾗｽﾀｰkに属している確率
(制約条件：$1 = \Sigma_{k=1}^{K} \gamma_{nk}$)

・混合ｶﾞｳｽﾓﾃﾞﾙによるﾃﾞｰﾀのｸﾗｽﾀﾘﾝｸﾞは、
(3.2) というｱﾙｺﾞﾘｽﾞﾑを用いて行います。
これは K-means法を拡張した方法で、以下のような手順です。

(初期化)　k番目のｸﾗｽﾀｰのｶﾞｳｽ分布N_kの広がり方(中心ﾍﾞｸﾄﾙ $\boldsymbol{\mu}_k$、共分散行列$\boldsymbol{\Sigma}_k$)、
および混合係数π_kの初期値を与えます。
(Eｽﾃｯﾌﾟ)　現時点でのπ_k、$\boldsymbol{\mu}_k$、$\boldsymbol{\Sigma}_k$を用いて、負担率$\boldsymbol{\gamma}$(各ﾃﾞｰﾀのｸﾗｽﾀｰへの所属確率)を計算します。
(Mｽﾃｯﾌﾟ)　現時点での負担率$\boldsymbol{\gamma}$を用いて、π_k、$\boldsymbol{\mu}_k$、$\boldsymbol{\Sigma}_k$を計算します。
上記 (Eｽﾃｯﾌﾟ)と(Mｽﾃｯﾌﾟ)を計算が収束するまで交互に繰り返します。

(3.2) は、ﾃﾞｰﾀの分布に合わせて、
構成するｶﾞｳｽ分布のﾊﾟﾗﾒｰﾀ(混合係数、中心、共分散行列)を逐次近似により最適化し、
ﾃﾞｰﾀのｸﾗｽﾀｰへの所属を(3.1) という形で決定するｱﾙｺﾞﾘｽﾞﾑです。

・混合ｶﾞｳｽﾓﾃﾞﾙでは、混合係数π_k、中心ﾍﾞｸﾄﾙ$\boldsymbol{\mu}_k$、共分散行列$\boldsymbol{\Sigma}_k$を用いて、
損失関数Ｅ(π, $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$)を以下の関係式から計算します。

混合ｶﾞｳｽﾓﾃﾞﾙの損失関数

$$E(\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\Sigma_{n=1}^{N} \left\{ \log\left(\Sigma_{k=1}^{K} \pi_k N_k(\boldsymbol{x}_n)\right) \right\}$$

$$N_k(\boldsymbol{x}_n) = \left\{1/(2\pi)^{D/2}\right\} (\det \boldsymbol{\Sigma}_k)^{-1/2} \exp\left\{-(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{T} (\boldsymbol{\Sigma}_k)^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)/2\right\}$$

$\boldsymbol{x}_n$　：n番目のﾃﾞｰﾀ (n=1～N：ﾃﾞｰﾀ数)
N_k　：k番目(k=1～K：ｸﾗｽﾀｰ数)ｸﾗｽﾀｰのｶﾞｳｽ分布
π_k　：N_kの混合係数
$\boldsymbol{\mu}_k$　：N_kの中心ﾍﾞｸﾄﾙ
$\boldsymbol{\Sigma}_k$　：N_kの共分散行列

・上記の混合ｶﾞｳｽﾓﾃﾞﾙの損失関数Ｅ(π, $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$)は、
(3.3) の対数をとった
(3.4) に、－１(ﾏｲﾅｽ1)を掛けたものです。
これが、混合ｶﾞｳｽﾓﾃﾞﾙの損失関数であり、これを最小化するようにﾓﾃﾞﾙを作成します。

（選択肢）

(a) 「EMｱﾙｺﾞﾘｽﾞﾑ（ｲｰｴﾑｱﾙｺﾞﾘｽﾞﾑ、Expectation-Maximization algorithm)」
(b) 「ｸﾗｽ指示変数Ｒ」
(c) 「対数尤度関数（ﾀｲｽｳﾕｳﾄﾞｶﾝｽｳ、log likelihood function)」
(d) 「負担率（ﾌﾀﾝﾘﾂ、responsibility)」
(e) 「尤度関数（ﾕｳﾄﾞｶﾝｽｳ、likelihood function)」
(f) 「歪み尺度（ﾕｶﾞﾐｼｬｸﾄﾞ、distortion measure)」

（回答）

(3.1)	
(3.2)	
(3.3)	
(3.4)	

以上。

(6) 確認問題回答用紙

提出者　：
提出日　：　年　月　日

回答　（全 11 問）

No.	回答
(1.1)	
(1.2)	
(1.3)	
(1.4)	
(1.5)	
(2.1)	
(2.2)	
(3.1)	
(3.2)	
(3.3)	
(3.4)	

（※黄色の枠　のみ記入をお願いします）

※ ご意見・ご要望など有りましたら、下欄に記してください。