# 機械学習基礎

赤穂 昭太郎

akaho@ism.ac.jp

https://github.com/toddler2009/ml-tutorial

2025 年 4 月 10 日

# 目次

# 第 3 章

# 教師なし学習

## 3.1　教師なし学習の導入

これまでの章では，正解ラベルが与えられたデータに基づいて学習を行う**教師あり学習**を扱ってきた．一方で，実世界のデータにはラベル情報が付与されていない場合も多く存在する．たとえば，ユーザの行動ログ，Web 上の文書群，画像や音声データなど，大量のデータは「分類されていない」状態で得られる．

このような**ラベルのないデータ**に対して，データの構造やパターンを自動的に発見する手法群を**教師なし学習（unsupervised learning）**と呼ぶ．教師なし学習の代表的な応用例には，以下のようなものがある：

- データの**可視化**や**次元削減**（例：主成分分析，t-SNE）
- **クラスタリング**によるグループ化（例：k-means，階層的クラスタリング）
- 異常検知，潜在構造の推定，推薦システムなど

### 3.1.1　高次元データの課題：次元の呪い

教師なし学習の多くの手法は，**高次元空間におけるデータの構造理解**を目的とする．しかし，高次元空間では次のような問題が発生することが知られている： (189)

**疎性（sparsity）**： 次元が増えると，データ点同士の距離が均一化し，近傍構造が意味をなさなくなる．

**距離の信頼性の低下**:　ユークリッド距離などの距離尺度が，高次元では識別力を失いやすい．

**視覚化の困難さ**:　人間は 2 次元または 3 次元でしか直感的にデータを理解できない．

このような現象は一般に**次元の呪い（curse of dimensionality）**と呼ばれ，高次元データ解析において重大な障害となる．

### 3.1.2　教師なし学習の目的

このような背景のもと，教師なし学習の目的は大きく分けて次の 2 つに分類できる：

1. **次元削減：** データの本質的な低次元構造を抽出し，可視化や効率的な分析に繋げる．
2. **クラスタリング：** データをラベルなしで自然なグループに分類し，潜在的な構造を発見する．

### 3.1.3　本章の構成

本章では，教師なし学習の代表例として**次元削減**と**クラスタリング**に焦点を当てる．まずは古典的な手法である**主成分分析（PCA）**や**階層的クラスタリング**，**k-means 法**の基礎を学んだ後，非線形な構造を捉えるための手法（例：t-SNE，UMAP，スペクトラルクラスタリングなど）へと進む．

### 3.1.4　補足：高次元における標準正規分布ベクトルの性質

次元の呪いの具体例として，高次元空間における標準正規分布ベクトルのノルムや相互距離，角度の挙動を考察する．以下では，$d$ 次元の標準正規分布 $N(0, I_d)$ に従うベクトル $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ を考える．

■(1) ノルムの集中（球面集中現象）　$\boldsymbol{x} \sim N(0, I_d)$ のとき，ノルムは

$$\|\boldsymbol{x}\| = \sqrt{\sum_{i=1}^{d} x_i^2}$$

と表される．このとき $\|\boldsymbol{x}\|^2$ は自由度 $d$ のカイ二乗分布 $\chi^2_d$ に従い，その期待値は $d$，分散は $2d$ である．中心極限定理により，

$$\|\boldsymbol{x}\| = \sqrt{d} + O_p(1)$$

が成り立ち，$\boldsymbol{x}$ のノルムは高次元になるほど $\sqrt{d}$ 付近に集中する．これを **球面集中現象（concentration on the sphere）** と呼ぶ.

■(2) ベクトル間の距離　$\boldsymbol{x}, \boldsymbol{y} \sim N(0, I_d)$ を独立に生成したとき，差 $\boldsymbol{x} - \boldsymbol{y}$ は $N(0, 2I_d)$ に従う．よってそのノルムは

$$\|\boldsymbol{x} - \boldsymbol{y}\| = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2} \approx \sqrt{2d} + O_p(1)$$

となる．すなわち，高次元では任意の 2 点間の距離がほぼ $\sqrt{2d}$ に集中する．

■(3) ベクトル間の角度　$\boldsymbol{x}, \boldsymbol{y}$ のなす角 $\theta$ は

$$\cos\theta = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|}$$

と定義される．高次元では $\boldsymbol{x} \cdot \boldsymbol{y}$ は平均 0，分散 $d$ の正規分布に従い，またノルムが $\sqrt{d}$ に集中するため，

$$\cos\theta \approx 0 \quad \Rightarrow \quad \theta \approx \frac{\pi}{2} + O_p(d^{-1/2})$$

が成り立つ．これは，高次元ではランダムな 2 ベクトルは互いに「ほぼ直交」することを意味する．

■**補足**　このような性質は，ユークリッド距離や角度に基づくアルゴリズム（例：k-NN 法，クラスタリング，カーネル法など）において，距離の有効性や解釈に大きな影響を与える．次元が高くなるほど「近い」「遠い」「似ている」といった直感的な関係が失われやすくなるため，次元圧縮などの前処理が重要となる．

## 3.2　主成分分析（PCA）

### 基本的な考え方と目的

(496)

主成分分析（Principal Component Analysis, PCA）は，教師なし学習における代表的な次元削減手法である．高次元のデータに対して，できるだけ情報（分散）を保ちながら，より低次元の空間に射影することを目的とする．

データを $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$ とし，平均 0 に標準化されたデータ行列を $X \in \mathbb{R}^{n \times d}$ とする．

### 分散最大化による導出

PCA は，データの分散が最大となる方向への射影を求める問題として定式化できる．

1 次元の主成分を求める場合，単位ベクトル $\boldsymbol{w} \in \mathbb{R}^d$ に対して，射影されたデータの分散を最大化する問題は：

$$\text{maximize} \quad \boldsymbol{w}^\top S \boldsymbol{w} \quad \text{subject to} \quad \|\boldsymbol{w}\| = 1$$

ここで $S = \frac{1}{n} X^\top X$ はデータの共分散行列である．この固有値問題の最大固有値に対応する固有ベクトル $\boldsymbol{w}_1$ が，第 1 主成分を与える．

同様に，第 2 主成分，第 3 主成分も直交制約のもとで逐次求められる．

**■補足：Lagrange の未定乗数法による導出**　PCA における第 1 主成分方向 $\boldsymbol{w}_1$ は，以下の最適化問題の解として求められる：

$$\text{maximize} \quad \boldsymbol{w}^\top S \boldsymbol{w} \quad \text{subject to} \quad \|\boldsymbol{w}\|^2 = 1$$

ここで $S = \frac{1}{n} X^\top X$ は共分散行列，$\boldsymbol{w}$ は単位長の方向ベクトルである．

この制約付き最適化問題を解くために，**ラグランジュの未定乗数法**を用いる．ラグランジュ関数を以下のように定義する：

$$\mathcal{L}(\boldsymbol{w}, \lambda) = \boldsymbol{w}^\top S\boldsymbol{w} - \lambda(\boldsymbol{w}^\top \boldsymbol{w} - 1)$$

ここで $\lambda$ は未定乗数である．最適化条件として，$\boldsymbol{w}$ に関する勾配がゼロである必要がある：

$$\nabla_{\boldsymbol{w}} \mathcal{L} = 2S\boldsymbol{w} - 2\lambda\boldsymbol{w} = 0 \quad \Rightarrow \quad S\boldsymbol{w} = \lambda\boldsymbol{w}$$

これは $\boldsymbol{w}$ が $S$ の固有ベクトルであり，$\lambda$ がその固有値であることを意味する．

したがって，$\boldsymbol{w}$ のとるべき方向は $S$ の固有ベクトルであり，最大分散を与える方向は**最大固有値に対応する固有ベクトル**である．

同様に，次の主成分は直交性制約のもとで次に大きい固有値に対応する固有ベクトルとして求められる．

### 特異値分解（SVD）との関係

PCA は, データ行列 $X$ の **特異値分解（SVD: Singular Value Decomposition）**としても表現できる．

$$X = U\Sigma V^\top$$

ここで,

- $U \in \mathbb{R}^{n\times n}$：左特異ベクトル（主成分スコア）
- $\Sigma \in \mathbb{R}^{n\times d}$：特異値（分散の平方根に対応）
- $V \in \mathbb{R}^{d\times d}$：右特異ベクトル（主成分方向）

このとき，PCA による $k$ 次元への次元削減は，$X \approx U_k\Sigma_k V_k^\top$ によるランク $k$ 近似とみなすことができる．したがって，PCA は行列の低ランク近似としても解釈できる．

### PCA の幾何学的性質と効果

- 主成分方向は，元の座標系に対する **回転** によって得られる新たな直交基底である．
- 分散の大きい方向（情報量の多い方向）を優先的に保持するため，次元削減後も重要な構造が保持されやすい．
- 元の空間におけるユークリッド距離が，新しい空間でも保存されやすい（線形写像）．

### 注意点と限界

- PCA は**線形手法**であり，非線形な構造（例：円環状，曲面構造など）は適切に表現できない．
- 特徴量のスケールに依存するため，単位系が違う特徴量をもつデータの場合などには**標準化（z スコア変換）**が必須である．
- 主成分軸の意味づけが難しいことがあり，**可視化には有効**だが**解釈には注意**が必要．

### 補足：2 次元の相関 PCA について

変数の標準化を行うのは単位系が異なるようなデータでは必須ではあるが，もともとの PCA の目的とは異なる解が得られる場合があることに注意する．2 次元の相関 PCA では必ず座標軸に対して 45° 方向に主軸が出ることを示そう．2 次元の射影軸に沿った単位ベクトルは角座標を使って $(\cos\theta, \sin\theta)$ と書くことができる．データをこの軸に射影した時の分散は（平均は 0 として）

$$E[(X_1\cos\theta + X_2\sin\theta)^2] = E[X_1^2\cos^2\theta + X_2^2\sin^2\theta + 2X_1X_2\cos\theta\sin\theta] = 1 + \rho\sin 2\theta$$

となり，これは相関係数 $\rho$ がどんなに小さくても厳密に 0 でない限り（実データではほぼ起こらない）$\theta = \pm\pi/4$ のどちらかで最大となる．2 次元ではこのように極端なことが起きるが，それより大きな次元でも多くの変数が相関するような軸が選ばれる

性質があることに注意する.

**■補足：主成分分析における次元数の選択**　主成分分析（PCA）は，高次元データをより少数の主成分に圧縮するための次元削減法であるが，圧縮後の次元数（主成分の数）をどのように選ぶかは実践上の重要な課題である.

**累積寄与率とその定義**　主成分分析では，共分散行列の固有値 $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$ を用いて，各主成分の寄与率（データの分散をどれだけ説明するか）を次のように定義する：

$$\text{寄与率（第 } k \text{ 主成分）} = \frac{\lambda_k}{\sum_{j=1}^{d} \lambda_j}$$

また，最初の $r$ 個の主成分で説明される**累積寄与率**は以下のように表される：

$$\text{累積寄与率} = \sum_{k=1}^{r} \frac{\lambda_k}{\sum_{j=1}^{d} \lambda_j}$$

この値が大きいほど，低次元の空間でも元のデータの分散情報をよく保持しているといえる.

**次元数決定の主な方法**

- **閾値による方法**：累積寄与率が 90% や 95% を超える最小の次元数 $r$ を選ぶ.
- **スクリー・プロット（scree plot）**：固有値の大小を折れ線グラフにし，「肩」や「ひじ（elbow)」のような曲がり点で次元数を決定する.
- **再構成誤差による評価**：元のデータと主成分空間での再構成との差（MSE など）を評価し，許容範囲内になる最小次元を選ぶ.

これらは主にヒューリスティックな手法であり，可視化やドメイン知識との組み合わせが有効である.

**ベイズモデリングによる次元数選択**　ベイズ的な PCA（Bayesian PCA）や確率的 PCA（Probabilistic PCA）では，次元数を固定せず，**事後分布として次元数に対する不確かさを表現**することができる.

さらに，モデル選択の枠組みでベイズ情報量基準（BIC）や周辺尤度（marginal likelihood）を用いて最も尤もらしい次元数を選択することもできる.

例えば，確率的 PCA では以下のようなモデルを仮定する：

$$\boldsymbol{x} = W\boldsymbol{z} + \boldsymbol{\epsilon}, \quad \boldsymbol{z} \sim \mathcal{N}(0, I), \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 I)$$

ここで潜在変数 $\boldsymbol{z}$ の次元を変化させてモデル選択を行うことで，**過学習を避けつつ最適な次元数を推定**できる．

■まとめ　主成分分析における次元数の選択は，情報の圧縮と損失のバランスを取る上で重要な要素である．累積寄与率に基づく閾値法やスクリー・プロットといった直感的手法に加えて，ベイズモデリングを活用することでより柔軟で理論的な次元選択が可能となる．

## まとめ

PCA は，データの分散構造を最大限に保持しつつ低次元に射影する代表的な線形次元削減手法であり，視覚化・前処理・特徴抽出など広範に用いられている．

## 補足：主成分分析の双対的視点と Biplot 表現

PCA における標準的な主成分軸は，共分散行列 $\boldsymbol{X}^\top \boldsymbol{X}$ の固有ベクトル（右特異ベクトル）に基づいて定義され，観測データ $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ に対して，低次元空間への射影：

$$\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{V}_r$$

を用いてデータ点（行方向）を可視化する．

これに対して，**biplot 表現**は，観測点（行）と変数（列）の両方を同時に可視化することを目的とした手法であり，以下のように行列 $\boldsymbol{X}\boldsymbol{X}^\top$ の固有ベクトル（左特異ベクトル）に基づいて主成分軸を構成する [10]．

### 双対的な固有空間の利用

$\boldsymbol{X}$ の特異値分解を

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$$

としたとき，$\boldsymbol{X}^\top\boldsymbol{X}$ の固有ベクトルは $\boldsymbol{V}$，$\boldsymbol{X}\boldsymbol{X}^\top$ の固有ベクトルは $\boldsymbol{U}$ であり，次の関係を持つ：

$$\boldsymbol{X}\boldsymbol{X}^\top\boldsymbol{U} = \boldsymbol{U}\boldsymbol{S}^2, \quad \boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{V} = \boldsymbol{V}\boldsymbol{S}^2$$

このとき，biplot 表現では，

- $\boldsymbol{Z} = \boldsymbol{U}_r\boldsymbol{S}_r$ により観測点をプロット
- $\boldsymbol{L} = \boldsymbol{V}_r$ により元の変数の方向ベクトル（負荷量）をプロット

することで，観測点と変数の両者を同じ図に表現する．

### 目的と使い方

biplot は次のような可視化を目的とする：

- 観測点（例：個体，サンプル）同士の類似性の可視化
- 各変数（特徴量）が主成分空間に与える影響（寄与）の可視化
- 主成分軸と変数の関係を矢印（方向ベクトル）として直感的に表示

### 注意点と限界

biplot 表現には以下のような注意が必要である：

- 主成分軸におけるスケーリングにより，矢印と点のバランスが変化する（対称 biplot, コントラスト強調型など）．
- 主成分の数を 2〜3 次元に制限して表示するため，情報の損失がある．
- 高次元のデータでは，主成分軸の解釈や方向の安定性に注意が必要．特に，$d$ 次元 t-分布のような裾の重い分布についての biplot 表現は座標軸に値が集中して意味のある情報が取り出せなくなることがあることが指摘されている（青嶋・矢田 [9]）

### MNIST データの PCA

Program 3.1　PCA

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import fetch_openml
from collections import defaultdict

# 1. MNIST データの読み込み
mnist = fetch_openml('mnist_784', version=1, as_frame=
    False)
X, y = mnist.data, mnist.target.astype(int)

### 標準化が必要な場合はコメントを外す
# # 2. 標準化
# scaler = StandardScaler()
# X_std = scaler.fit_transform(X)

# 3. の実行PCA
pca = PCA()
# X_pca = pca.fit_transform(X_std) # 標準化を行う場合
X_pca = pca.fit_transform(X) # 標準化を行わない場合

# 4. 累積寄与率のプロット
cum_var_ratio = np.cumsum(pca.explained_variance_ratio_)

plt.figure(figsize=(8, 4))
```

```
plt.plot(cum_var_ratio, marker='.')
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance")
plt.title("Cumulative Explained Variance Ratio")

# 閾値をマーク
for threshold in [0.80, 0.90, 0.95, 0.99]:
    d = np.argmax(cum_var_ratio >= threshold) + 1
    plt.axhline(y=threshold, color='gray', linestyle='--')
    plt.axvline(x=d, color='gray', linestyle='--')
    plt.text(d + 2, threshold - 0.05, f"{threshold*100:.0f}% → {d} dims", fontsize=9)

plt.grid(True)
plt.tight_layout()
plt.show()

# 5. PC1-PC2 のプロット（各クラスごとに n_samples 表示）
n_per_class = 200  # 各クラス表示数
class_data = defaultdict(list)
indices = []

for i, label in enumerate(y):
    if len(class_data[label]) < n_per_class:
        class_data[label].append(i)
        indices.append(i)
    if len(indices) >= 10 * n_per_class:
        break
```

```
53
54 X_proj = X_pca[indices, :2]
55 y_proj = y[indices]
56
57 plt.figure(figsize=(8, 6))
58 for digit in range(10):
59     mask = y_proj == digit
60     plt.scatter(X_proj[mask, 0], X_proj[mask, 1], label=
       f"{digit}", alpha=0.6, s=10)
61
62 plt.xlabel("PC1")
63 plt.ylabel("PC2")
64 plt.title(f"PCA Projection to 2D (Up to {n_per_class}
   per class)")
65 plt.legend()
66 plt.grid(True)
67 plt.tight_layout()
68 plt.show()
69
70 # 6. 主成分基底（固有ベクトル）を画像として表示（最初の 25 個）
71 n_components_to_show = 25
72 image_shape = (28, 28)
73
74 components = pca.components_[:n_components_to_show]
75
76 fig, axs = plt.subplots(5, 5, figsize=(7, 7))
77 for i, ax in enumerate(axs.ravel()):
78     comp_img = components[i].reshape(image_shape)
```

```
79      comp_img = (comp_img - comp_img.min()) / (comp_img.
            max() - comp_img.min())  # 0-1 に正規
            化
80      ax.imshow(comp_img, cmap='gray')
81      ax.set_title(f"PC{i+1}")
82      ax.axis('off')
83
84  plt.suptitle("Top 25 Principal Components as Images")
85  plt.tight_layout()
86  plt.show()
```

## 3.3　非線形次元削減法：t-SNE と UMAP の原理

PCA は線形射影によってデータの分散構造を保存する次元削減手法であるが，非線形な多様体構造を持つデータには対応できない．そのため，データ間の局所的な関係性（近傍構造）を保ちながら可視化を行うために，非線形次元削減法が提案されている．その代表例が **t-SNE** と **UMAP** である．

### 3.3.1　t-SNE の原理

t-SNE（t-distributed Stochastic Neighbor Embedding）は，データの**局所的な類似性（近さ）を確率分布としてモデル化**し，**高次元空間と低次元空間の類似性分布の差を最小化**することにより次元削減を行う．

#### 高次元空間における類似度分布

データ点 $\boldsymbol{x}_i \in \mathbb{R}^D$ に対し，ガウスカーネルによって点 $\boldsymbol{x}_j$ との類似度を以下のように定義する：

$$p_{j|i} = \frac{\exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2 / 2\sigma_i^2)}$$

ここで $\sigma_i$ は点 $\boldsymbol{x}_i$ ごとに設定される局所的な分布のスケールであり，**パープレキシ**

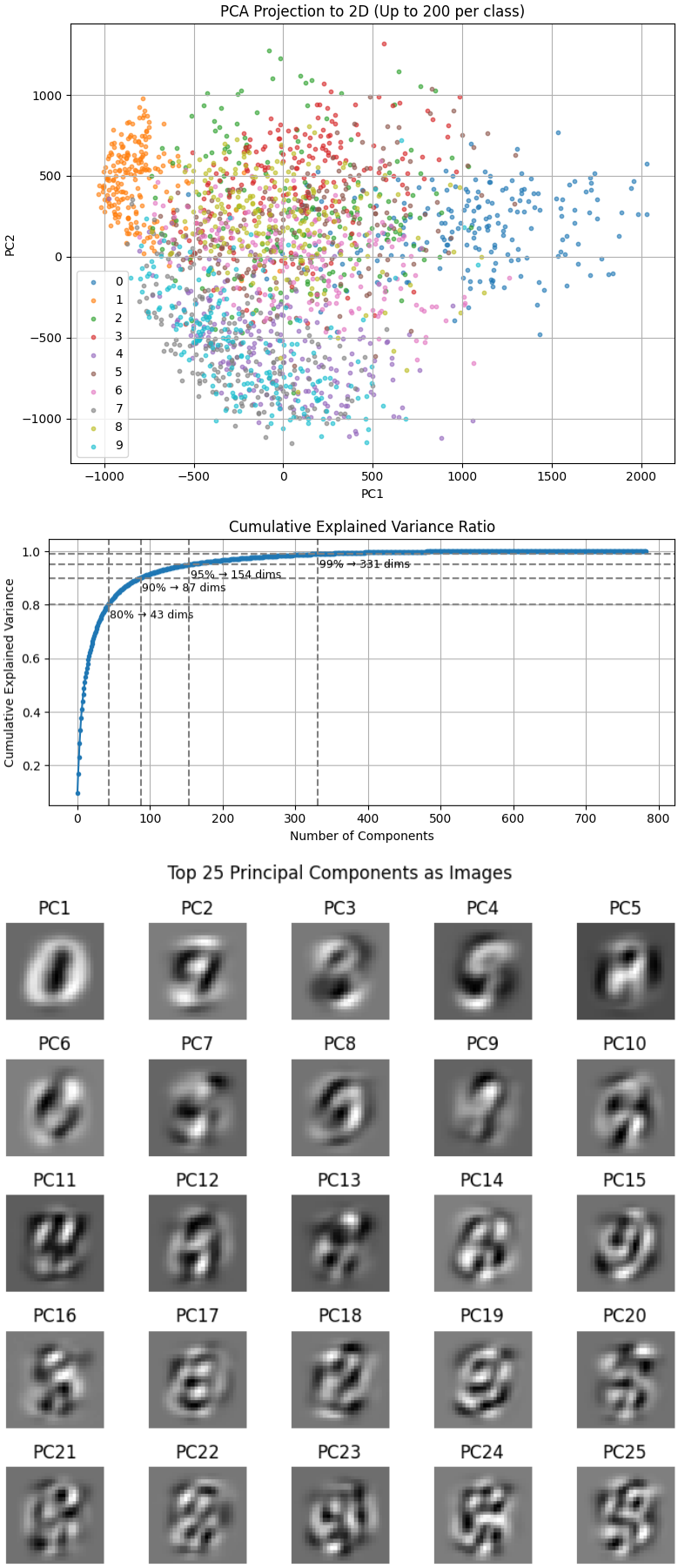

PCA Projection to 2D (Up to 200 per class)
PC1
PC2
0
1
2
3
4
5
6
7
8
9
Cumulative Explained Variance Ratio
Cumulative Explained Variance
Number of Components
80% → 43 dims
90% → 87 dims
95% → 154 dims
99% → 331 dims
Top 25 Principal Components as Images
PC1
PC2
PC3
PC4
PC5
PC6
PC7
PC8
PC9
PC10
PC11
PC12
PC13
PC14
PC15
PC16
PC17
PC18
PC19
PC20
PC21
PC22
PC23
PC24
PC25

図 3.1　PCA

**ティ（perplexity）**によって調整される．

対称化した類似度は以下で与えられる：

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

#### 低次元空間における類似度分布

対応する低次元の点 $\boldsymbol{y}_i \in \mathbb{R}^d$ に対して，Cauchy 分布（t 分布）に基づく類似度を定義する：

$$q_{ij} = \frac{(1 + \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|\boldsymbol{y}_k - \boldsymbol{y}_l\|^2)^{-1}}$$

t 分布を用いることで，**crowding problem（混雑問題）**を軽減し，低次元での適切な配置を実現している．

#### 目的関数と最適化

高次元と低次元の分布の乖離は，KL ダイバージェンスにより測定される：

$$\mathrm{KL}(P \parallel Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

この損失関数を最小化するように，確率的勾配降下法によって $\boldsymbol{y}_i$ を最適化する．

### 3.3.2　UMAP の原理

UMAP（Uniform Manifold Approximation and Projection）は，**多様体学習と位相空間論に基づく手法**であり，t-SNE と同様に局所的構造を保つが，より理論的に定式化された近傍グラフの構築と高速な学習が特徴である．

#### 高次元における近傍確率グラフの構築

各点 $\boldsymbol{x}_i$ に対して，距離に基づく局所的な類似度関数を用いて近傍確率を定義する．近傍点 $\boldsymbol{x}_j$ に対して：

$$p_{ij} = \exp\left(-\frac{\max(0, \|\boldsymbol{x}_i - \boldsymbol{x}_j\| - \rho_i)}{\sigma_i}\right)$$

ここで $\rho_i$ は点 $\boldsymbol{x}_i$ の最近傍との距離，$\sigma_i$ はスケールパラメータである．

この確率的近傍情報を用いて，**ファジー近傍グラフ（fuzzy simplicial set）**が構成される．

#### 低次元への埋め込みと損失関数

低次元空間での点 $\boldsymbol{y}_i$ に対して，対応するグラフ上の類似度 $q_{ij}$ を次のように定義する：

$$q_{ij} = \left(1 + a\|\boldsymbol{y}_i - \boldsymbol{y}_j\|^{2b}\right)^{-1}$$

ここで $a, b$ は事前に定められたパラメータである．

目的関数は以下のようなクロスエントロピー型の損失関数であり，$p_{ij}$ を近傍確率，$q_{ij}$ を低次元での対応確率とすると：

$$C = \sum_{i<j} \left[p_{ij} \log q_{ij} + (1 - p_{ij}) \log(1 - q_{ij})\right]$$

### t-SNE と UMAP の比較

- **局所構造の保持：** 両者とも近傍関係を重視しており，クラスタの視覚化に適している．
- **グローバル構造：** UMAP はある程度のグローバル構造も保持しやすい．
- **計算コスト：** UMAP はより高速でスケーラブルな実装が可能である．
- **距離スケールの扱い：** t-SNE は相対的な確率比，UMAP は距離変換関数に基づく．

これらの手法は，高次元データのクラスタ構造や多様体構造を視覚的に理解する上で極めて有効であり，教師なし学習における重要なツールである．

## 補足：t-SNE の perplexity と crowding 問題，ハイパーパラメータの選び方

**■t-SNE における Perplexity の意味** t-SNE では，高次元空間における点 $\boldsymbol{x}_i$ の近傍点との距離を正規分布に基づいて確率的に表現するが，その分布のスケール $\sigma_i$ は各点ごとに異なる．$\sigma_i$ の決定には，**パープレキシティ（perplexity）**というパラメータが用いられる．

パープレキシティは，近傍点の数に関する柔軟な尺度であり，点 $\boldsymbol{x}_i$ に対して次のように定義される：

$$\mathrm{Perp}(P_i) = 2^{H(P_i)} \quad \text{where} \quad H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}$$

ここで $P_i$ は条件付き確率分布 $p_{j|i}$，$H(P_i)$ はシャノンエントロピーである．Perplexity は「$i$ にとって意味のある近傍点の数」と解釈できる．

- 小さい perplexity（例：5〜30）：局所構造の保持に強く反応，クラスタ分離を強調
- 大きい perplexity（例：50〜100）：広い範囲の構造を捉えるが，クラスタの境界が曖昧になりやすい

一般的にはデータサイズに応じて 5 〜 50 程度で設定されることが多い．

**■Crowding problem（混雑問題）** 高次元空間では多くの点がある程度離れた距離に存在するのに対し，低次元空間ではその「中程度の距離」が表現しづらくなる．この現象を **crowding problem（混雑問題）** と呼ぶ．

この問題に対処するために，低次元空間では **t 分布（具体的には自由度 1 の Cauchy 分布）**を用いて，遠くの点同士の相互作用を抑制している．

**■t-SNE の主なハイパーパラメータ**

- **perplexity**：近傍数の柔軟な制御（上記参照）．小さすぎても大きすぎても性能が悪化．

- **learning rate**（学習率）：最適化の収束速度と安定性に影響．一般には 200〜1000 程度．
- **n_iter**：イテレーション数．デフォルトでは 1000 回程度で，250 回程度は初期配置の慣らし（early exaggeration）に用いられる．

■UMAP の主なハイパーパラメータと指針　UMAP でも，近傍の定義と空間構造に関するハイパーパラメータが性能に影響する：

- **n_neighbors**：各点の近傍数．小さい値（5〜15）は局所構造を重視，大きい値（30 以上）は大域構造を保持．
- **min_dist**：低次元空間における点の「最小間隔」．小さいほどクラスタが詰まり，クラスタの分離が明瞭になる．
- **metric**：距離尺度（例：‘euclidean’, ‘cosine’, ‘manhattan’）．データ構造に応じて選択．

■ハイパーパラメータの選び方の実践的な指針

- 可視化目的の場合は，いくつかの設定を試してクラスタ構造の安定性を比較する．
- 分類やクラスタリングの前処理として用いる場合には，クラスタ境界の保持とデータ分布の整合性に注目する．
- t-SNE では perplexity，UMAP では n_neighbors の値を変化させ，結果の安定性を見ることが重要．

Program 3.2　tSNE and UMAP

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.datasets import fetch_openml
4 from sklearn.preprocessing import StandardScaler
5 from sklearn.manifold import TSNE
6 import umap
```

```
from collections import defaultdict

# 1. MNIST データの読み込み
mnist = fetch_openml('mnist_784', version=1, as_frame=
    False)
X, y = mnist.data, mnist.target.astype(int)

# 2. 各クラスから n_per_class 個ずつサンプリング
n_per_class = 200
class_indices = defaultdict(list)
selected_indices = []

for i, label in enumerate(y):
    if len(class_indices[label]) < n_per_class:
        class_indices[label].append(i)
        selected_indices.append(i)
    if len(selected_indices) >= 10 * n_per_class:
        break

X_sub = X[selected_indices]
y_sub = y[selected_indices]

# 3. 標準化
scaler = StandardScaler()
X_sub_std = scaler.fit_transform(X_sub)

# 4. t-SNE の実行
tsne = TSNE(n_components=2, perplexity=30, random_state
    =42)
```

```
X_tsne = tsne.fit_transform(X_sub_std)

# 5. UMAP の実行
umap_model = umap.UMAP(n_components=2, random_state=42)
X_umap = umap_model.fit_transform(X_sub_std)

# 6. 可視化関数（共通化）
def plot_embedding(X_emb, title, y_labels):
    plt.figure(figsize=(8, 6))
    for digit in range(10):
        mask = y_labels == digit
        plt.scatter(X_emb[mask, 0], X_emb[mask, 1],
            label=str(digit), s=10, alpha=0.6)
    plt.xlabel("Component 1")
    plt.ylabel("Component 2")
    plt.title(title)
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

# 7. 結果の可視化
plot_embedding(X_tsne, "t-SNE Projection (MNIST)", y_sub
    )
plot_embedding(X_umap, "UMAP Projection (MNIST)", y_sub)
```

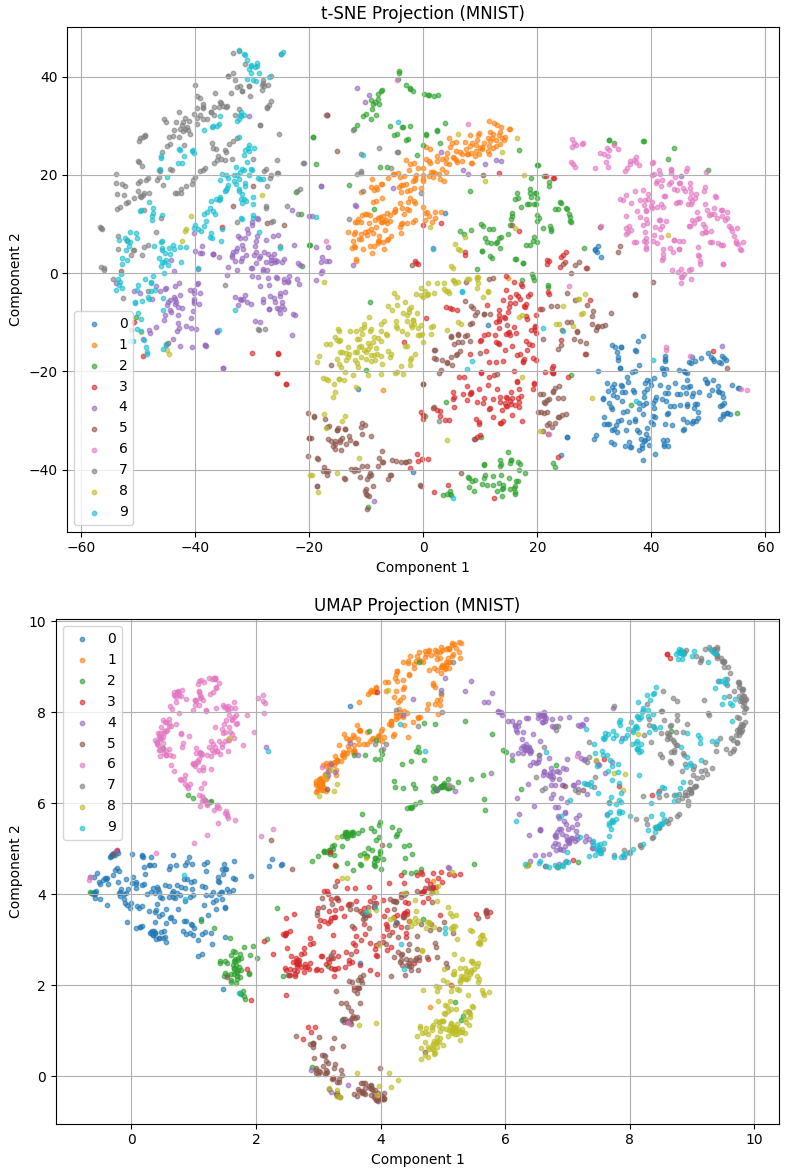



図 3.2　t-SNE と UMAP

## 3.4　その他の次元削減法：NMF とカーネル PCA

### 非負値行列因子分解（Non-negative Matrix Factorization, NMF）

NMF は，観測データ行列 $X \in \mathbb{R}_{\geq 0}^{n \times d}$ を 2 つの非負行列の積として近似する手法である：

$$X \approx WH, \quad W \in \mathbb{R}_{\geq 0}^{n \times r}, \quad H \in \mathbb{R}_{\geq 0}^{r \times d}$$

ここで $r$ は低次元のランクであり，$W$ は基底の重み（係数），$H$ は基底ベクトルと解釈できる．

目的関数は通常，次のような Frobenius ノルムの最小化として定式化される：

$$\min_{W,H \geq 0} \|X - WH\|_F^2$$

このような制約付き最小化問題に対して，交互最適化（multiplicative update rule など）が用いられる．

**■補足：確率的 NMF への変換**　非負行列 $X \in \mathbb{R}_{\geq 0}^{n \times d}$ に対し，$X \approx WH$ という NMF が成り立つとき，各行を正規化して得られる $X'$ に対しても次のような確率的 NMF が成り立つ：

$$X' = D_X^{-1} X \approx D_X^{-1} W D_H^{-1} H = W' H'$$

ここで $D_X$ は $X$ の各行の 1 ノルム（行和）を対角要素とする対角行列であり，$D_H$ も $H$ に対して同様に定義される．$X'$ と $H'$ の行和が 1 であることから $W' = D_X^{-1} W D_H^{-1}$ の行和も 1 となることが示せる．

**■NMF の特徴と解釈性**　NMF は，得られる分解要素がすべて非負であるため，データの加法的な構成要素として自然に解釈できる．これは，文書のトピック分解や画像のパーツ分解において有用である．

特に，NMF は確率的潜在意味解析（PLSA: Probabilistic Latent Semantic Analysis）と密接な関係があり，各行を確率分布（比率）として解釈できることから，**意味的に解釈しやすい表現を得る手法**として知られている．

- 各行：観測対象（例：文書）の潜在成分（例：トピック）に対する重み
- 各列：特徴次元（例：単語）の潜在成分における寄与度

Program 3.3　SHAP

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.decomposition import NMF
from sklearn.preprocessing import MinMaxScaler

# 1. データの読み込みMNIST
mnist = fetch_openml('mnist_784', version=1, as_frame=
    False)
X, _ = mnist.data, mnist.target

# 2. 入力データのスケーリング（は非負制約あり）NMF
# 元データは〜 0255 の画素値：そのままでもだが，〜にスケーリングすると安
    定OK01
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# 3. の実行（次元に圧縮）NMF25
n_components = 25
nmf = NMF(n_components=n_components, init='nndsvda',
    random_state=42)
W = nmf.fit_transform(X_scaled)
H = nmf.components_  # shape: (25, 784)

# 4. 基底ベクトル（の各行）を画像として表示H
fig, axs = plt.subplots(5, 5, figsize=(8, 8))
for i, ax in enumerate(axs.ravel()):
    component = H[i].reshape(28, 28)
    ax.imshow(component, cmap='gray')
```

```
    ax.set_title(f"Basis {i+1}", fontsize=8)
    ax.axis('off')

plt.suptitle("Top 25 NMF Components (Basis Images)",
    fontsize=14)
plt.tight_layout()
plt.show()

# 5. 特定の数字（例：「」）のインデックス取得8
target_digit = 8
indices = np.where(y == target_digit)[0][:1]   # 上位件だけ
    表示5

# 6. 表示処理
for idx in indices:
    original = X_scaled[idx].reshape(28, 28)
    coeffs = W[idx]
    reconstructed = np.dot(coeffs, H).reshape(28, 28)

    # プロット
    fig, axs = plt.subplots(1, 3, figsize=(12, 4))

    axs[0].imshow(original, cmap='gray')
    axs[0].set_title("Original")
    axs[0].axis('off')

    axs[1].imshow(reconstructed, cmap='gray')
    axs[1].set_title("Reconstructed (NMF)")
    axs[1].axis('off')
```

```
54
55      axs[2].bar(np.arange(len(coeffs)), coeffs)
56      axs[2].set_title("NMF Coefficients (W row)")
57      axs[2].set_xlabel("Component Index")
58      axs[2].set_ylabel("Weight")
59      axs[2].set_xticks(np.arange(len(coeffs)))
60
61      plt.suptitle(f"Example of Digit {target_digit}",
            fontsize=14)
62      plt.tight_layout()
63      plt.show()
```

### カーネル主成分分析（Kernel PCA）

カーネル PCA は，非線形変換 $\phi : \mathbb{R}^d \to \mathcal{F}$ を用いて，元のデータを高次元の特徴空間 $\mathcal{F}$ に写像した上で，その空間における主成分分析を行う手法である．

PCA と同様に，データの共分散行列に対する固有値問題に帰着するが，写像 $\phi$ を明示せず，カーネル関数 $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$ を用いて以下のように表現される：

$$K\boldsymbol{\alpha} = \lambda \boldsymbol{\alpha}$$

ここで $K$ はカーネル行列（Gram 行列），$\boldsymbol{\alpha}$ は固有ベクトル，$\lambda$ は固有値である．

#### ■カーネル PCA の特徴

- 非線形構造の次元削減が可能
- 選択したカーネル（例：RBF, 多項式）により表現力が異なる
- 変換後の空間は暗黙的であるため，元の特徴との対応解釈が難しい場合がある

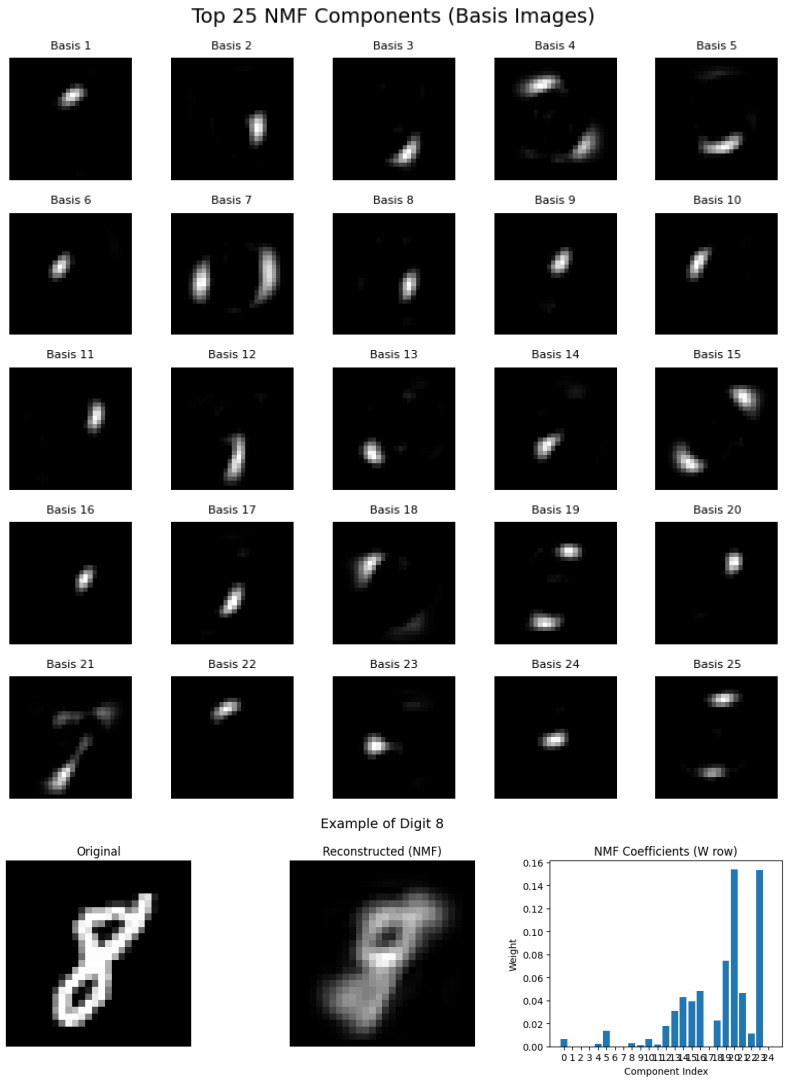



図 3.3　NMF

## MDS とカーネル PCA の関係：距離とカーネルの対応

カーネル PCA と多次元尺度構成法（MDS）は，一見異なる手法に見えるが，いずれも**内積情報や距離情報に基づいて埋め込みを構成する**という共通点を持つ．特に，両者は**中心化されたカーネル行列（または距離行列）**に基づく固有値問題として統一的に扱うことができる．

**■MDS の基本**　MDS（特にクラシカル MDS）は，観測された距離行列 $D = [d_{ij}]$ に対して，**距離を保った低次元埋め込み**を求める手法である．ここで，$d_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$

はユークリッド距離とする．

このとき，中心化された内積行列 $K$ は次のように距離行列から計算できる：

$$K = -\frac{1}{2} C D^{(2)} C$$

ここで $D^{(2)}$ は要素ごとの二乗距離行列（$d_{ij}^2$），$C = I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$ は中心化行列である．

この $K$ に対して固有値分解を行うことで，MDS による低次元埋め込みが得られる．

**■カーネル PCA との関係**　一方，カーネル PCA では，データを非線形マッピング $\phi(\boldsymbol{x})$ によって高次元空間に写像し，その特徴空間上の内積行列（カーネル行列）$K$ に対して固有値分解を行う：

$$K_{ij} = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$$

このとき，ユークリッドノルムの恒等式：

$$\|\phi(\boldsymbol{x}_i) - \phi(\boldsymbol{x}_j)\|^2 = K_{ii} + K_{jj} - 2K_{ij}$$

より，距離情報とカーネル行列の間には以下の関係がある：

$$K_{ij} = -\frac{1}{2}\left(\|\phi(\boldsymbol{x}_i) - \phi(\boldsymbol{x}_j)\|^2 - \|\phi(\boldsymbol{x}_i)\|^2 - \|\phi(\boldsymbol{x}_j)\|^2\right)$$

さらに，全体を中心化することで，**MDS の内積行列とカーネル PCA のカーネル行列が一致**する．したがって，**MDS は，ユークリッド距離の二乗から内積行列を計算し，それに対して PCA を適用する手法**とみなすことができ，**カーネル PCA の特別な場合**であると理解できる．

**■まとめ**

- MDS とカーネル PCA はどちらも「中心化された類似度行列」に対する固有値分解に基づいている．
- 距離情報とカーネル（内積）情報の間には明確な変換関係がある．
- 距離行列を基にした MDS は，線形カーネルを用いたカーネル PCA と等価．

## 3.5　独立成分分析（Independent Component Analysis, ICA）

独立成分分析（ICA）は，多変量データを統計的に**独立な成分**の線形和として分解する手法である．主に信号分離や次元削減，特徴抽出などの応用に用いられる．

### 独立性とその定義

ICA の目的は，観測データ $\boldsymbol{x} \in \mathbb{R}^d$ を以下のようなモデルで分解することである：

$$\boldsymbol{x} = A\boldsymbol{s}$$

ここで $A \in \mathbb{R}^{d \times d}$ は混合行列，$\boldsymbol{s} \in \mathbb{R}^d$ は元の独立な成分（ソース）である．ICA では $\boldsymbol{s}$ の成分が統計的に「独立」であることを仮定する．

**■独立性の定義**　確率変数 $s_1, \ldots, s_d$ が独立であるとは，それらの同時分布が個別の周辺分布の積で書けることを意味する：

$$p(s_1, \ldots, s_d) = \prod_{i=1}^{d} p(s_i)$$

この定義により，ICA は「成分ができるだけ独立になるような線形変換を求める」問題として定式化される．

### 独立性の評価指標

実際には真の分布が不明なため，以下のような統計的尺度で独立性を近似的に評価・最大化する：

- **高次のモーメント**（尖度など）：ガウス性からの逸脱度
- **エントロピーに基づく相互情報量**：$I(\boldsymbol{s}) = \sum H(s_i) - H(\boldsymbol{s})$
- **非ガウス性の最大化**（中心極限定理により，線形和はガウスに近づく）

### ガウス成分に関する理論的制限

ICA には以下のような重要な理論的制限がある：

- ガウス分布の成分は，任意の直交変換後も独立であるため，**ガウス成分を分離することは不可能**.
- このため，**独立成分のうちガウス分布に従うものは高々 1 つ**であると仮定するのが通例.
- 逆に，非ガウス性を最大化することで非ガウスな独立成分の抽出が可能になる.

### FastICA と不動点アルゴリズム

FastICA は，非ガウス性（たとえば尖度やネゲントロピー）を最大化することで独立成分を求める高速なアルゴリズムである．主な特徴は以下の通り：

- 観測データ $\boldsymbol{x}$ を白色化（whitening）した後，線形変換によって成分を抽出
- 各成分に対して，非線形関数 $g$ に基づく不動点反復により重みベクトル $\boldsymbol{w}$ を更新
- 更新式の例：

$$\boldsymbol{w}^{\text{new}} = \mathbb{E}[\boldsymbol{x} g(\boldsymbol{w}^\top \boldsymbol{x})] - \mathbb{E}[g'(\boldsymbol{w}^\top \boldsymbol{x})]\boldsymbol{w}$$

- 各 $\boldsymbol{w}$ は直交化して同時に複数の成分を抽出

非線形関数 $g$ の選択（例：tanh, $x^3$ など）は性能に影響を与える.

**■補足：FastICA の更新式の導出 [11]**　FastICA アルゴリズムは，非ガウス性を最大化することで独立成分を抽出する方法である．主に **ネゲントロピー（negative entropy）** を最大化する形で目的関数が設計されている.

**非ガウス性の尺度としてのネゲントロピー**　ネゲントロピー $J(y)$ は以下のように定義される：

$$J(y) = H(y_{\text{gauss}}) - H(y)$$

ここで $H(\cdot)$ はエントロピー，$y_{\text{gauss}}$ は $y$ と同じ分散をもつガウス分布である．ネゲントロピーは常に非負であり，ガウス分布に近いほど小さくなる．

実用上は，$J(y)$ を次のような近似式で計算する：

$$J(y) \approx \left[\mathbb{E}\, G(y) - \mathbb{E}\, G(\nu)\right]^2$$

ここで $G$ は適切な非線形関数（例：$G(y) = \log\cosh(y)$, $G(y) = -\exp(-y^2/2)$ など），$\nu$ は標準正規分布からのサンプルである．

**最適化問題と不動点反復の導出**　FastICA は，白色化済みのデータ $\boldsymbol{x}$ に対して，射影 $y = \boldsymbol{w}^\top \boldsymbol{x}$ の非ガウス性を最大にする方向 $\boldsymbol{w}$ を求める．

今，$J(y)$ を $\|\boldsymbol{w}\| = 1$ のもとで Lagrange 関数を最大化した停留点は

$$\mathbb{E}[\boldsymbol{x} g(\boldsymbol{w}^\top \boldsymbol{x})] + \beta \boldsymbol{w} = 0$$

を満たす ($\beta$ は Lagrange の未定乗数と $J(y)$ の微分から出てくるスカラーを含む)．ここで，$g = G'$ は非線形関数 $G$ の導関数である．例えば $G(y) = \log\cosh(y)$ の場合は $g(y) = \tanh(y)$.

この解を求めるために左辺を $\boldsymbol{w}$ で微分すると，

$$\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top g'(\boldsymbol{w}^\top \boldsymbol{x})] + \beta I$$

となる．さらに，ここでデータが白色化されていることを考慮して，

$$\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top g'(\boldsymbol{w}^\top \boldsymbol{x})] \simeq \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top]\mathbb{E}[g'(\boldsymbol{w}^\top \boldsymbol{x})] = \mathbb{E}[g'(\boldsymbol{w}^\top \boldsymbol{x})] I$$

という近似が成り立つと仮定する．すると Newton 法により

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{\mathbb{E}[\boldsymbol{x} g(\boldsymbol{w}^\top \boldsymbol{x})] + \beta \boldsymbol{w}}{\mathbb{E}[g'(\boldsymbol{w}^\top \boldsymbol{x})] + \beta}$$

という更新式が得られる．右辺全体に $g'(\boldsymbol{w}^\top \boldsymbol{x}) + \beta$ をかけて整理すると $\beta$ が消え，これ全体が新しい $\boldsymbol{w}$ に比例するのでスカラー倍を無視すれば以下の不動点反復式が得られる（スカラー倍については後で正規化の際に調整される）：

$$\boldsymbol{w}^{\text{new}} = \mathbb{E}[\boldsymbol{x} g(\boldsymbol{w}^\top \boldsymbol{x})] - \mathbb{E}[g'(\boldsymbol{w}^\top \boldsymbol{x})] \boldsymbol{w}$$

この更新式は，$\boldsymbol{w}$ を一度に最適化方向に飛ばす**不動点法（fixed-point iteration）**になっており，学習率の調整なしに高速に収束するという利点がある．

また，複数の独立成分を求める際には，各 $\boldsymbol{w}_i$ に対して直交制約（$\boldsymbol{w}_i^\top \boldsymbol{w}_j = \delta_{ij}$）を課し，直交化（たとえば Gram-Schmidt 法）を行うことで同時に複数成分を抽出可能である．

## 因子分析との関係

ICA はしばしば，因子分析（Factor Analysis）と比較される．両者は「潜在変数モデル」に基づいてデータを記述する点で共通するが，仮定や目的は異なる．

| 手法 | 潜在変数の仮定 | 主な目的 |
|---|---|---|
| 因子分析 | 潜在因子は**ガウス分布** | 共分散構造の説明 |
| ICA | 潜在成分は**非ガウスかつ独立** | 独立成分の抽出 |

表 3.1　因子分析と ICA の比較

因子分析は主に分散・共分散構造をモデル化するのに対し，ICA は独立性を重視して情報の分離を目指す．

**■まとめ**　ICA は，独立性というより強い統計的仮定に基づく強力な特徴抽出法であり，音源分離や画像処理など幅広い応用がある．FastICA はその中でも代表的な高速アルゴリズムとして知られている．

**■補足：因子回転と ICA との関係**　因子分析（Factor Analysis）は，観測データ $\boldsymbol{x} \in \mathbb{R}^d$ を潜在変数（因子）$\boldsymbol{z} \in \mathbb{R}^r$ の線形結合と誤差項で表す統計モデルである：

$$\boldsymbol{x} = \Lambda \boldsymbol{z} + \boldsymbol{\epsilon}, \quad \boldsymbol{z} \sim \mathcal{N}(0, I), \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \Psi)$$

ここで $\Lambda$ は因子負荷行列（factor loading），$\Psi$ は誤差の分散共分散行列である．

**因子回転（Factor Rotation）**　因子分析では，$\boldsymbol{z}$ がガウス分布に従うため，$\Lambda$ は以下のような任意の直交変換 $R$ をかけても同じモデルを表現できる：

$$\boldsymbol{x} = \Lambda \boldsymbol{z} = \Lambda R R^\top \boldsymbol{z} = (\Lambda R)(\underbrace{R^\top \boldsymbol{z}}_{\boldsymbol{z}'})$$

このように，因子分析では因子空間に対する**回転の自由度**が存在し，$\Lambda$ の見た目（および $z$ の意味）は一意ではない．

この自由度を利用して，因子の解釈性を高めるために以下のような回転が行われる：

- **直交回転**（Varimax など）：因子間の独立性を保ったまま，負荷のスパース性を促進
- **斜交回転**（Promax など）：因子間の相関を許容し，構造を柔軟に表現

**ICA との関係性**　ICA では，因子（独立成分）に対して **独立性（非ガウス性）を最大化するような回転** を選ぶ．これは，因子分析における「見やすさのための回転」とは目的が異なり，**統計的基準に基づく最適な回転**であると解釈できる．

- 因子分析：回転の自由度を利用して「解釈性」を向上（意味的な単純構造を目指す）
- ICA：独立性を数値的に最大化する回転を求める（モデル選択そのもの）

このように，ICA は因子分析における回転の一つの具体的な解とみなすことができ，特に非ガウス分布を仮定する場合には，ICA の方がより情報量のある分解を与える可能性がある．

## 3.6　階層的クラスタリング（Hierarchical Clustering）

階層的クラスタリングは，観測データを順次マージまたは分割することで，データに階層的なクラスタ構造を構築するクラスタリング手法である．ここでは，特に**凝集型階層的クラスタリング（agglomerative hierarchical clustering）**について説明する．

### 基本アルゴリズム

階層的クラスタリングの基本的な手順は次の通り：

1. 各データ点を 1 つのクラスタとする（初期状態では $n$ 個のクラスタ）
2. すべてのクラスタ間の距離を計算

3. 最も距離が近い 2 つのクラスタをマージ
4. クラスタ数が 1 になるまで，手順 2〜3 を繰り返す

この過程により，クラスタの統合関係が二分木構造として表現され，**デンドログラム（dendrogram）**として可視化される．

### クラスタ間距離（linkage）の定義

クラスタ間の距離（linkage）は，複数のデータ点を含むクラスタ同士の代表距離をどのように定義するかによって異なる．代表的な定義は以下の通り：

- **単連結法（single linkage）**：

$$d(C_1, C_2) = \min_{x \in C_1, y \in C_2} \|x - y\|$$

- **完全連結法（complete linkage）**：

$$d(C_1, C_2) = \max_{x \in C_1, y \in C_2} \|x - y\|$$

- **平均連結法（average linkage）**：

$$d(C_1, C_2) = \frac{1}{|C_1||C_2|} \sum_{x \in C_1} \sum_{y \in C_2} \|x - y\|$$

- **重心法（centroid linkage）**：

$$d(C_1, C_2) = \|\mu_1 - \mu_2\|, \quad \mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

これらの選択はクラスタリング結果の形に大きな影響を与えるため，目的に応じた選択が必要である．

### デンドログラムの読み方

デンドログラムは，クラスタの統合過程を表す木構造の図である．横軸にはデータ点，縦軸にはクラスタ間の距離（または類似度）が示される．

- 各「枝（branch）」の高さは，対応する 2 クラスタの結合距離を表す．
- 水平方向に切ることで，任意のクラスタ数に分割することができる．
- 切る高さが大きいほど，クラスタ間の類似度が低くなる（粗い分類）．

### 手法の注意点と特徴

- 計算量が $O(n^2 \log n)$ と比較的重く，大規模データには不向き
- クラスタの構造が階層的に見えるため，可視化や探索的解析に有効
- 外れ値に弱く，距離のスケーリングに大きく影響を受ける
- クラスタの「数」を事前に指定する必要はないが，分割基準の選択が必要

階層的クラスタリングは，明確なクラスタ数がわからない場合や，クラスタ間の構造の「全体像」を可視化したい場合に有用な手法である．

Program 3.4　階層的クラスタリング

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from scipy.cluster.hierarchy import linkage, fcluster,
    dendrogram
from collections import defaultdict
import matplotlib.cm as cm

# 1. の読み込みと抽出（各数字個ずつ）MNIST10
mnist = fetch_openml('mnist_784', version=1, as_frame=
    False)
X, y = mnist.data, mnist.target.astype(int)

samples_per_class = 10
selected_indices = []
class_counts = defaultdict(int)

```

```
for i, label in enumerate(y):
    if class_counts[label] < samples_per_class:
        selected_indices.append(i)
        class_counts[label] += 1
    if sum(class_counts.values()) >= 10 *
       samples_per_class:
        break

X_subset = X[selected_indices]
y_subset = y[selected_indices]

# 2. 標準化とPCA
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_subset)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# 3. 計算とクラスタ（クラスタに固定）linkage10
linkage_single = linkage(X_pca, method='single')
linkage_complete = linkage(X_pca, method='complete')

clusters_single = fcluster(linkage_single, t=10,
   criterion='maxclust')
clusters_complete = fcluster(linkage_complete, t=10,
   criterion='maxclust')

# 4. 空間でクラスタと文字表示PCA
def plot_clusters(X, labels, clusters, title):
```

```
    import matplotlib.cm as cm
    import matplotlib.colors as mcolors

    plt.figure(figsize=(8, 6))
    unique_clusters = np.unique(clusters)
    colors = cm.get_cmap("tab10", len(unique_clusters))

    # 描画空間を確保するための（透明） scatter
    plt.scatter(X[:, 0], X[:, 1], s=0)  # 表示されないがスケーリングを確保

    # 各点にラベル付きで描画
    for i in range(len(X)):
        plt.text(
            X[i, 0], X[i, 1],
            str(labels[i]),
            color=colors(clusters[i] % 10),
            fontsize=12,
            ha='center', va='center'
        )

    plt.title(title)
    plt.xlabel("PC1")
    plt.ylabel("PC2")
    plt.grid(True)
    plt.show()


plot_clusters(X_pca, y_subset, clusters_single, "PCA +
```

```
    Single Linkage Clustering")
plot_clusters(X_pca, y_subset, clusters_complete, "PCA +
    Complete Linkage Clustering")

# 5. デンドログラムの葉に色付き数字（ラベルによる色分け）
def colored_dendrogram(Z, y_labels, title):
    label_colors = cm.tab10(np.array(y_labels) % 10)

    def llf(id):
        return f"${y_labels[id]}$"

    fig, ax = plt.subplots(figsize=(12, 6))
    dendro = dendrogram(Z, labels=[str(l) for l in
        y_labels],
#                        color_threshold=0,  # ブランチの色
    を無効にする
                        above_threshold_color='gray',
                        leaf_rotation=0,
                        leaf_font_size=10,
                        ax=ax)

    # 葉のテキストをラベルごとに色分け
    xlbls = ax.get_xmajorticklabels()
    for lbl in xlbls:
        digit = int(lbl.get_text())
        lbl.set_color(cm.tab10(digit % 10))

    ax.set_title(title)
    plt.xlabel("Sample Index (Digit)")
```

```
97      plt.ylabel("Cluster Distance")
98      plt.show()
99
100 colored_dendrogram(linkage_single, y_subset, "Dendrogram
        (Single Linkage)")
101 colored_dendrogram(linkage_complete, y_subset, "
      Dendrogram (Complete Linkage)")
```

## 3.7　k-means 法によるクラスタリング

k-means 法は，代表的な非階層型のクラスタリング手法であり，ユークリッド空間上のデータをあらかじめ指定された個数 $k$ のクラスタに分割する．各クラスタは重心（centroid）を持ち，データ点は最も近い重心に割り当てられる．

### 基本的な考え方

与えられたデータ $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$ を $k$ 個のクラスタに分割することを目的とする．各クラスタ $C_1, \ldots, C_k$ には重心（代表点） $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ を対応させ，次の目的関数（分散の和）を最小化する：

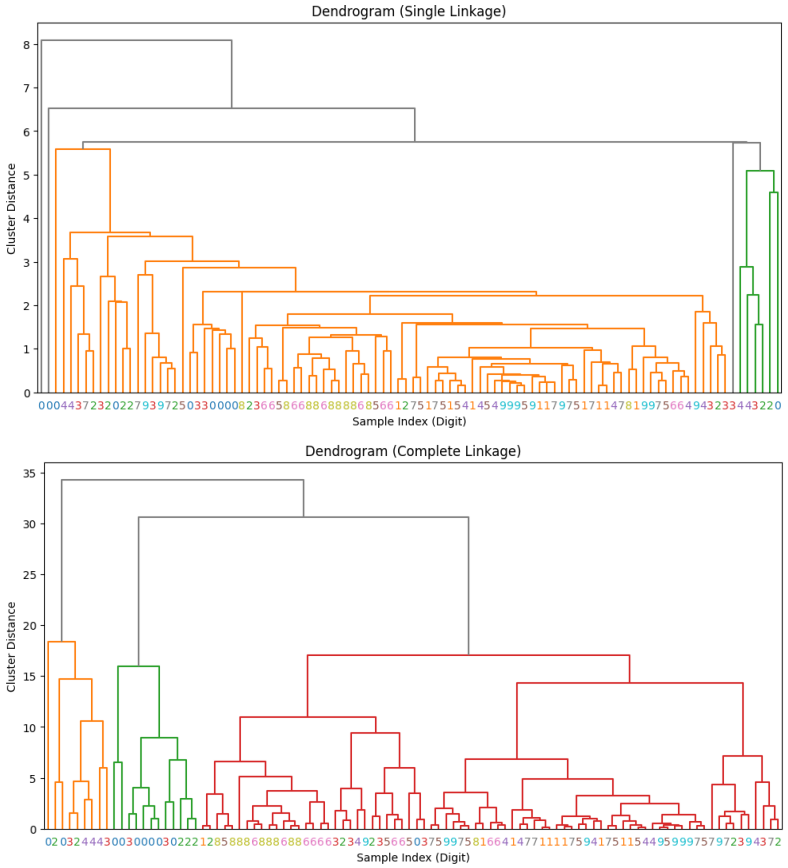
$$J = \sum_{j=1}^{k} \sum_{\boldsymbol{x}_i \in C_j} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|^2$$

これは「各クラスタ内のばらつきをできるだけ小さくする」ようなクラスタリングを意味する．

### アルゴリズムの手順（k-means アルゴリズム）

k-means 法は次のような反復的なアルゴリズムで実装される：

1. 初期重心 $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ をランダムに選択
2. 各データ点 $\boldsymbol{x}_i$ を最も近い重心 $\boldsymbol{\mu}_j$ に割り当てる



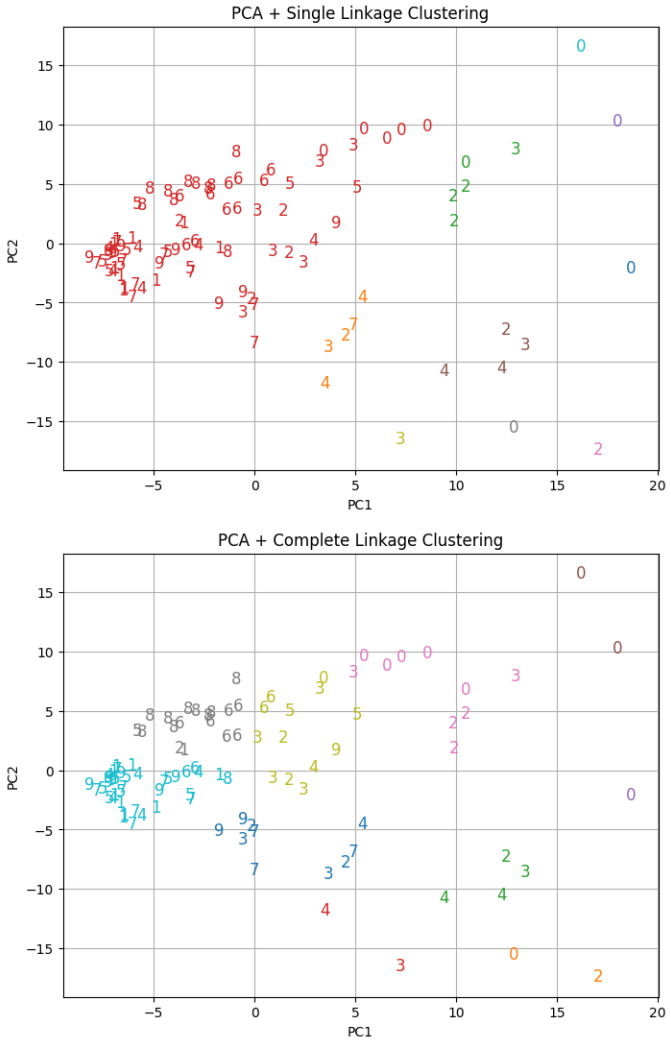




図 3.4　階層的クラスタリング

3. 各クラスタの重心を，割り当てられたデータの平均で更新する：

$$\boldsymbol{\mu}_j = \frac{1}{|C_j|} \sum_{\boldsymbol{x}_i \in C_j} \boldsymbol{x}_i$$

4. クラスタの割り当てが変化しなくなるまで手順 2 – 3 を繰り返す

この手法は，**期待値最大化法（EM 法）**の特殊な場合と解釈できる．

### 特徴と注意点

- クラスタ数 $k$ を事前に指定する必要がある．
- 初期値によって結果が変わるため，複数回の初期化（k-means++ など）を行うのが一般的．
- 各クラスタが球状かつ等分散であることを仮定しており，非凸な形状のクラスタや異なるスケールには弱い．
- 非階層型であるため，計算コストが比較的低く，大規模データに適用しやすい．
- クラスタの境界は，各重心とのユークリッド距離に基づいて線形的に決まる．

■まとめ　k-means 法はシンプルかつ効率的なクラスタリング手法であり，初学者にとっても理解しやすく，基礎的なアルゴリズムとして広く使用されている．ただし，適用するデータの特性や初期値，クラスタ数の選択には注意が必要である．

## 3.8　ガウス混合分布と EM アルゴリズムによるクラスタリング

### ガウス混合分布の定義

ガウス混合分布（Gaussian Mixture Model, GMM）は，複数の正規分布の線形結合によって定義される確率分布である．$K$ 成分の混合モデルは次のように定義される：

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \Sigma_k)$$

ここで，

- $\pi_k$：混合係数（$0 \leq \pi_k \leq 1$, $\sum_k \pi_k = 1$）
- $\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \Sigma_k)$：$k$ 番目の正規分布

これはクラスタリングにおいて，「各クラスタがガウス分布に従う」という仮定の下で，クラスタに所属する確率的なモデルを構築する手法である．

### EM アルゴリズムの一般論

EM（Expectation-Maximization）アルゴリズムは，**潜在変数（未観測変数）または欠損値**を含む確率モデルに対して，**尤度を最大化**するための反復的最適化手法である．

観測データ $\boldsymbol{x}$，潜在変数 $\boldsymbol{z}$ に対して，完全データの対数尤度 $\log p(\boldsymbol{x}, \boldsymbol{z} \mid \theta)$ を最大化したいが，$\boldsymbol{z}$ が観測されないため，以下のように期待値をとった **Q 関数** を最大化する：

$$Q(\theta, \theta^{(t)}) = \mathbb{E}_{\boldsymbol{z} \sim p(\boldsymbol{z} \mid \boldsymbol{x}, \theta^{(t)})} \left[ \log p(\boldsymbol{x}, \boldsymbol{z} \mid \theta) \right]$$

アルゴリズムは以下の 2 ステップを交互に実行する：

E ステップ　：現時点のパラメータ $\theta^{(t)}$ に基づいて $Q$ 関数を計算
M ステップ　：$Q(\theta, \theta^{(t)})$ を $\theta$ に関して最大化して $\theta^{(t+1)}$ を更新

### 指数型分布族と EM アルゴリズム

指数型分布族に属する確率モデルでは，分布は以下のように表現される：

$$p(x \mid \theta) = h(x) \exp\left( \eta(\theta)^\top T(x) - A(\theta) \right)$$

この形をとる場合，期待値 $\mathbb{E}[T(x)]$ によって十分統計量を求めることができ，EM アルゴリズムは解析的に扱いやすくなる．

## GMM への応用と潜在変数の導入

GMM を EM で学習するには，潜在変数 $\boldsymbol{z} = (z_1, \ldots, z_K)$ を導入する．ここで $z_k \in \{0, 1\}$ は，$\boldsymbol{x}$ が $k$ 番目の成分から生成されたかどうかを表す指示変数（one-hot ベクトル）とする：

$$p(\boldsymbol{x}, \boldsymbol{z}) = \prod_{k=1}^{K} \left[\pi_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \Sigma_k)\right]^{z_k}$$

この形式は指数型分布族に属しており，EM アルゴリズムが適用できる．

## GMM の EM アルゴリズム

観測データ $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ に対する GMM の学習は，以下のような EM アルゴリズムで行われる．

**■E ステップ（事後確率の計算）**

$$\gamma_{ik} := p(z_k = 1 \mid \boldsymbol{x}_i, \theta) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_j, \Sigma_j)}$$

**■M ステップ（パラメータの更新）**

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

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i=1}^{n} \gamma_{ik} \boldsymbol{x}_i$$

$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{n} \gamma_{ik} (\boldsymbol{x}_i - \boldsymbol{\mu}_k)(\boldsymbol{x}_i - \boldsymbol{\mu}_k)^\top$$

これらの式に基づき，E ステップと M ステップを交互に実行してパラメータを収束させる．

**■まとめ**　GMM は，各クラスタが異なる分散をもつ連続的な分布であることを表現でき，k-means 法の確率的な一般化とみなすことができる．EM アルゴリズムを用いることで，観測データと潜在変数を同時に扱う柔軟な学習が可能となる．

Program 3.5 非階層的クラスタリング

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from matplotlib.patches import Ellipse
from collections import defaultdict

# 1. MNIST 読み込み＆各クラス個抽出10
mnist = fetch_openml('mnist_784', version=1, as_frame=
    False)
X, y = mnist.data, mnist.target.astype(int)

samples_per_class = 10
selected_indices = []
class_counts = defaultdict(int)

for i, label in enumerate(y):
    if class_counts[label] < samples_per_class:
        selected_indices.append(i)
        class_counts[label] += 1
    if sum(class_counts.values()) >= 10 *
        samples_per_class:
        break

X_subset = X[selected_indices]
```

```
y_subset = y[selected_indices]

# 2. 標準化＋PCA
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_subset)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# 3. k-means クラスタリング
k = 10
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans_labels = kmeans.fit_predict(X_pca)

# 4. クラスタリングGMM
gmm = GaussianMixture(n_components=k, covariance_type='full', random_state=42)
gmm_labels = gmm.fit_predict(X_pca)

# 5. 楕円描画関数（用）GMM
def draw_ellipse(position, covariance, ax, edgecolor):
    if covariance.shape == (2, 2):
        U, s, _ = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
        ell = Ellipse(
            xy=position,
            width=width,
            height=height,
```

```
55                 angle=angle,
56                 edgecolor=edgecolor,
57                 facecolor='none',
58                 lw=2
59             )
60             ax.add_patch(ell)
61
62 # 6. 描画関数（数字・色・中心・楕円）
63 def plot_clusters(X, labels, clusters, centers=None,
       covariances=None, title=""):
64     plt.figure(figsize=(8, 6))
65     cmap = plt.get_cmap("tab10")
66     colors = [cmap(cl % 10) for cl in clusters]
67
68     # 各点を数字で表示
69     for i in range(len(X)):
70         plt.text(X[i, 0], X[i, 1], str(labels[i]),
71                  color=colors[i], fontsize=12,
72                  ha='center', va='center')
73
74     # 中心点（代表点）の表示
75     if centers is not None:
76         plt.scatter(centers[:, 0], centers[:, 1],
77                     c='black', s=100, marker='x', label=
                        'Cluster centers')
78
79     # GMM 楕円の表示
80     if covariances is not None:
81         for i in range(k):
```

```
82              draw_ellipse(centers[i], covariances[i], plt
                    .gca(), edgecolor=cmap(i))
83
84      plt.xlabel("PC1")
85      plt.ylabel("PC2")
86      plt.title(title)
87      plt.grid(True)
88      plt.legend()
89      plt.show()
90
91  # 7. 結果表示
92  plot_clusters(X_pca, y_subset, kmeans_labels,
93                centers=kmeans.cluster_centers_,
94                title="K-means Clustering on PCA-reduced
                    MNIST")
95
96  plot_clusters(X_pca, y_subset, gmm_labels,
97                centers=gmm.means_, covariances=gmm.
                    covariances_,
98                title="GMM Clustering on PCA-reduced MNIST
                    (with Ellipses)")
```

**■補足：クラスタ数の選択**　クラスタリングにおいてクラスタ数 $k$ を適切に選ぶことは，結果の解釈や性能に大きな影響を与えるが，多くの場合は事前に「正しい」クラスタ数が与えられているとは限らない．本節では，クラスタ数を決定するための代表的な方法をいくつか紹介する．

**エルボー法（Elbow method）**　$k$ を変化させながら，k-means 法などで得られる**クラスタ内誤差平方和（Within-Cluster Sum of Squares, WCSS）**を評価し，その減少量の変化が鈍化する「ひじ（elbow）」のような点をクラスタ数の候補とする．

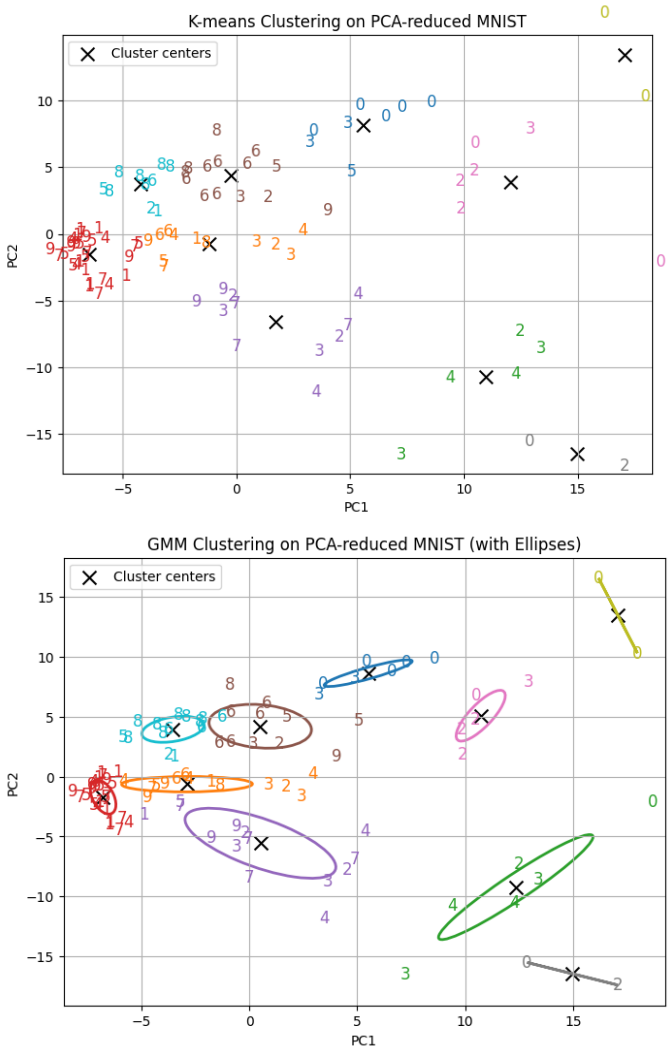



図 3.5　非階層的クラスタリング

$$\mathrm{WCSS}(k) = \sum_{j=1}^{k} \sum_{\boldsymbol{x}_i \in C_j} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|^2$$

この方法は視覚的な判断に依存するが，クラスタ数が多すぎると改善が頭打ちになるという性質を利用する．

**シルエット係数（Silhouette coefficient）**　各データ点に対して，そのクラスタ内の平均距離 $a_i$ と，最も近い他クラスタの平均距離 $b_i$ を計算し，以下のような指標を定義する：

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

この $s_i$ を全体で平均したシルエットスコアが最大となる $k$ を選択する．$s_i$ は $[-1, 1]$ の範囲にあり，1 に近いほど良いクラスタリングとされる．

**情報量基準（AIC/BIC）**　ガウス混合モデルのような確率モデルベースのクラスタリングでは，以下のような**モデル選択基準**を用いてクラスタ数を決定することがある：

- 赤池情報量基準（AIC）：

$$\mathrm{AIC} = -2 \log L + 2p$$

- ベイズ情報量基準（BIC）：

$$\mathrm{BIC} = -2 \log L + p \log n$$

ここで，$L$ はモデルの対数尤度，$p$ は自由度（パラメータ数），$n$ はデータ数である．BIC はパラメータの数に対するペナルティがより強く，過学習に対してより保守的である． ただし，ガウス混合分布モデルには特異性があるため，厳密な意味では AIC, BIC の満たすべき仮定を満たすわけではないことに注意する．

**ベイズモデリングによるアプローチ**　ベイズ的なクラスタリングの枠組みでは，クラスタ数を固定せず，**確率変数として扱う**ことができる．たとえばディリクレ過程混合モデル（DPMM）などはクラスタ数を事前に指定する必要がなく，データから自動的に推定される．

このようなモデルでは，クラスタ数の事後分布やマルコフ連鎖モンテカルロ法（MCMC）を用いた推定が行われる．ベイズモデリングによるアプローチの詳細は，次章で詳しく述べる．

**■まとめ**　クラスタ数の選択には，モデルに応じてさまざまな手法が存在する．k-means のような非確率的手法ではエルボー法やシルエット係数が，GMM のような確率的手法では AIC や BIC が活用される．また，ベイズ的な手法を用いることで，クラスタ数を未知の変数として柔軟に扱うことが可能となる．

## 3.9　スペクトラルクラスタリングとその他のクラスタリング手法

### スペクトラルクラスタリングの考え方

スペクトラルクラスタリングは，データの類似度をもとに構成されたグラフ構造の固有ベクトル（スペクトル）を利用してクラスタリングを行う手法である．非線形な構造や非凸なクラスタにも適用できる柔軟な手法として知られている．

**グラフと類似度行列**　データ点 $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ に対し，類似度（親和度）行列 $W \in \mathbb{R}^{n \times n}$ を定義する．典型的には RBF カーネルを用いて，

$$W_{ij} = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right)$$

のように定める．

**グラフラプラシアンの定義**　類似度行列 $W$ に対応する次数行列 $D$ を $D_{ii} = \sum_j W_{ij}$ として定義し，**非正規化グラフラプラシアン**は次のように与えられる：

$$L = D - W$$

また，よく用いられる正規化ラプラシアンには以下の 2 つがある：

- 対称正規化ラプラシアン：

$$L_{\mathrm{sym}} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2}$$

- ランダムウォークラプラシアン：

$$L_{\mathrm{rw}} = D^{-1} L = I - D^{-1} W$$

**直感的な説明**　グラフラプラシアンは，グラフの平滑性や接続性を表現する演算子であり，固有ベクトルの構造にはグラフのクラスタ構造が反映される．特に，$L$ の最小固有値に対応する固有ベクトル（Fiedler ベクトル）は，グラフを 2 つの部分に分割する情報を持つ．

### スペクトラルクラスタリングのアルゴリズム

以下は対称正規化ラプラシアンを用いた代表的なスペクトラルクラスタリングの手順である：

1. 類似度行列 $W$ を定義（RBF カーネルなど）
2. $L_{\text{sym}} = I - D^{-1/2} W D^{-1/2}$ を計算
3. $L_{\text{sym}}$ の小さい方から $k$ 個の固有ベクトルを列とする行列 $U$ を作成
4. $U$ の各行を正規化して新たなデータ点とし，k-means 法でクラスタリング

この手法は，非凸形状や非線形な分離を持つクラスタ構造にも対応可能である．また，カーネル PCA と同様に，非線形な空間への写像を通じてクラスタ構造を明らかにする点で，**カーネル法との関連性**が深い．

### スペクトラルクラスタリングの特徴

- 類似度に基づくグラフ構造からクラスタリングを行う
- 非線形クラスタ・非凸構造にも対応可能
- 固有値分解を伴うため計算コストは $O(n^3)$ と高く，大規模データには適さない
- 類似度行列やカーネル幅 $\sigma$ の選択が性能に大きく影響

### その他の代表的なクラスタリング手法

階層的クラスタリング，k-means 法，GMM に加えて，以下のようなクラスタリング手法も広く用いられている：

- **DBSCAN（Density-Based Spatial Clustering of Applications with Noise）**：密度に基づいてクラスタを定義し，任意の形状のクラスタや外れ値の検出が可能．クラスタ数を事前に指定する必要がない．
- **Mean Shift**：データ密度のモード（ピーク）を探索し，密度の高い領域をクラスタとする非パラメトリック手法．クラスタ数の指定不要だが，カーネル幅

の選定が重要.

- **HDBSCAN（Hierarchical DBSCAN）**：DBSCAN の拡張で，階層的構造と安定なクラスタの自動抽出を両立．ノイズへの耐性が高い.
- **OPTICS（Ordering Points To Identify the Clustering Structure）**：距離密度に基づくクラスタリングで，密度の異なるクラスタに対応可能．クラスタ構造の可視化にも適する.

これらの手法は，非パラメトリック性，非凸クラスタの検出，外れ値への耐性など，k-means や GMM とは異なる特性を持つ．問題設定やデータの特性に応じて使い分けが重要である.

# 参考文献


[1] James, G., Witten, D., Hastie, T., Tibshirani, R. (2023) An Introduction to Statistical Learning, Second Edition, Springer. https://www.statlearning.com

[2] Bishop, C.M. (2026) Pattern recognition and machine learning, Springer (ビショップ：パターン認識と機械学習（上下），丸善)

[3] Kamishima, T., Akaho, S., Asoh, H., Sakuma, J. (2012). Fairness-aware classifier with prejudice remover regularizer. In Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2012, Bristol, UK, September 24-28, 2012. Proceedings, Part II 23 (pp. 35-50). Springer Berlin Heidelberg.

[4] 萩原克幸 (2022)，入門 統計的回帰とモデル選択，共立出版.

[5] 赤穂昭太郎 (2008), カーネル多変量解析，岩波書店.

[6] Wahba, G. (1990). Spline models for observational data. Society for industrial and applied mathematics.

[7] 金森敬文, 竹之内高志, 村田昇 (2009). パターン認識 (R で学ぶデータサイエンス 5) 共立出版.

[8] Cover, T., Hart, P. (1967). Nearest neighbor pattern classification. IEEE transactions on information theory, 13(1), 21-27.

[9] 青嶋誠，矢田和善 (2019)，高次元の統計学，共立出版

[10] 小西貞則. (2010). 多変量解析入門: 線形から非線形へ. 岩波書店.

[11] Hyvärinen, A. (2013). Independent component analysis: recent advances. Philosophical Transactions of the Royal Society A: Mathematical, Physical

and Engineering Sciences, 371(1984), 20110534.

[12] S. Watanabe, M. Opper (2010), Asymptotic equivalence of Bayes cross validation and widely applicable information criterion in singular learning theory. Journal of machine learning research, 11(12).