第 2 章

教師あり学習

2.1　線形モデル

2.1.1　重回帰モデル

単回帰モデルを拡張し、複数の説明変数を用いて目的変数を予測するモデルが重回帰モデルである。モデルは次のように表される：

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + \varepsilon$$

ここで誤差項 ε は独立同分布の正規分布 $\mathcal{N}(0, \sigma^2)$ に従うと仮定する。この仮定の下では、最小二乗法でパラメータを推定することは、最尤推定と一致する。

最小二乗法では次の目的関数を最小化する：

$$\min_{\boldsymbol{\beta}} \|\boldsymbol{y} - X\boldsymbol{\beta}\|^2$$

このとき、正規方程式により解は次のように求まる（$X^T X$ が正則な場合）：

$$\boldsymbol{\beta} = (X^T X)^{-1} X^T \boldsymbol{y}$$

2.1.2　多項式回帰

説明変数の多項式を用いることで、非線形な関係も線形回帰の枠組みで扱うことができる。例えば 3 次の多項式回帰は次のようになる：

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \varepsilon$$

変数変換により、パラメータに対しては線形性が保たれているため、最小二乗法をそのまま適用できる。

2.1.3　正則化

モデルの柔軟性が高すぎると、訓練データに過剰に適合する（過学習）リスクがある。これを防ぐために、パラメータにペナルティを課す「正則化」が用いられる。

リッジ回帰

リッジ回帰では、L2 ノルムによってパラメータの大きさを抑制する。目的関数は以下の通り：

$$\min_{\boldsymbol{\beta}} \left(\|\boldsymbol{y} - X\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|^2 \right)$$

このときの解は以下のように計算される：

$$\boldsymbol{\beta}_{\text{ridge}} = (X^T X + \lambda I)^{-1} X^T \boldsymbol{y}$$

ここで λ は正則化の強さを調整するハイパーパラメータであり、I は単位行列である。

スパース回帰（Lasso）

Lasso 回帰では、L1 ノルムによる正則化を行い、いくつかの係数をゼロにする効果を持つ。目的関数は以下の通り：

$$\min_{\boldsymbol{\beta}} \left(\|\boldsymbol{y} - X\boldsymbol{\beta}\|^2 + \lambda \sum_j |\beta_j| \right)$$

これにより、重要な説明変数の選択（変数選択）を同時に行うことができる。

2.1.4　具体例：California Housing データによる回帰問題

データセットの概要

California Housing Dataset は、米国カリフォルニア州の各地域における住宅に関する統計情報を用いて、**住宅価格の中央値**を予測する回帰問題である。

このデータセットは、1990 年の国勢調査データに基づいており、scikit-learn の `fetch_california_housing` 関数で取得できる。

- サンプル数：20,640
- 特徴量数：8（すべて数値）
- 目的変数：住宅の中央値価格（単位：$100,000）

特徴量の説明

各サンプルは、ある地域の統計的な特徴を表しており、以下の変数が含まれる：

- `MedInc`：地域の世帯収入の中央値
- `HouseAge`：住宅の築年数の中央値
- `AveRooms`：世帯あたりの平均部屋数
- `AveBedrms`：世帯あたりの平均寝室数
- `Population`：地域の人口
- `AveOccup`：世帯あたりの平均居住者数
- `Latitude`：緯度
- `Longitude`：経度

これらの特徴量はすべて連続値であり、**前処理として標準化（平均 0、分散 1）**を行うことで、多くの回帰モデルが適切に動作する。

Program 2.1　リッジ回帰と LASSO 回帰

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.datasets import fetch_california_housing
```

```
4  from sklearn.linear_model import Ridge, Lasso
5  from sklearn.metrics import mean_squared_error
6  from sklearn.preprocessing import StandardScaler
7
8  # データの読み込み
9  data = fetch_california_housing()
10 X, y = data.data, data.target
11 feature_names = data.feature_names
12
13 # 学習・テストデータの選択
14 k, m = 100, 300
15 np.random.seed(42)
16 indices = np.random.permutation(len(X))
17 train_idx, test_idx = indices[:k], indices[k:k + m]
18 X_train, y_train = X[train_idx], y[train_idx]
19 X_test, y_test = X[test_idx], y[test_idx]
20
21 # 標準化
22 scaler = StandardScaler()
23 X_train_std = scaler.fit_transform(X_train)
24 X_test_std = scaler.transform(X_test)
25
26 # 正則化パラメータ
27 alphas = np.logspace(-3, 2, 100)
28
29 # との誤差と係数追跡RidgeLasso
30 ridge_mse_train, ridge_mse_test = [], []
31 lasso_mse_train, lasso_mse_test = [], []
32 ridge_coefs, lasso_coefs = [], []
```

```
33
34 for alpha in alphas:
35     ridge = Ridge(alpha=alpha).fit(X_train_std, y_train)
36     lasso = Lasso(alpha=alpha, max_iter=10000).fit(
           X_train_std, y_train)
37
38     ridge_mse_train.append(mean_squared_error(y_train,
           ridge.predict(X_train_std)))
39     ridge_mse_test.append(mean_squared_error(y_test,
           ridge.predict(X_test_std)))
40
41     lasso_mse_train.append(mean_squared_error(y_train,
           lasso.predict(X_train_std)))
42     lasso_mse_test.append(mean_squared_error(y_test,
           lasso.predict(X_test_std)))
43
44     ridge_coefs.append(ridge.coef_)
45     lasso_coefs.append(lasso.coef_)
46
47 # プロット
48 fig, axs = plt.subplots(2, 2, figsize=(12, 10))
49
50 # Ridge: 訓練テスト/ MSE
51 axs[0, 0].plot(alphas, ridge_mse_train, label='Train MSE
       ', linestyle='--')
52 axs[0, 0].plot(alphas, ridge_mse_test, label='Test MSE')
53 axs[0, 0].set_xscale('log')
54 axs[0, 0].set_xlabel('Alpha')
55 axs[0, 0].set_ylabel('MSE')
```

```
axs[0, 0].set_title('Ridge: Train & Test MSE vs Alpha')
axs[0, 0].legend()
axs[0, 0].grid(True)

# Lasso: 訓練テスト/ MSE
axs[0, 1].plot(alphas, lasso_mse_train, label='Train MSE', linestyle='--')
axs[0, 1].plot(alphas, lasso_mse_test, label='Test MSE')
axs[0, 1].set_xscale('log')
axs[0, 1].set_xlabel('Alpha')
axs[0, 1].set_ylabel('MSE')
axs[0, 1].set_title('Lasso: Train & Test MSE vs Alpha')
axs[0, 1].legend()
axs[0, 1].grid(True)

# Ridge: 係数パス
axs[1, 0].plot(alphas, ridge_coefs)
axs[1, 0].set_xscale('log')
axs[1, 0].set_xlabel('Alpha')
axs[1, 0].set_ylabel('Coefficient Value')
axs[1, 0].set_title('Ridge: Coefficient Paths')
axs[1, 0].legend(feature_names, loc='best', fontsize='small')

# Lasso: 係数パス
axs[1, 1].plot(alphas, lasso_coefs)
axs[1, 1].set_xscale('log')
axs[1, 1].set_xlabel('Alpha')
axs[1, 1].set_ylabel('Coefficient Value')
```

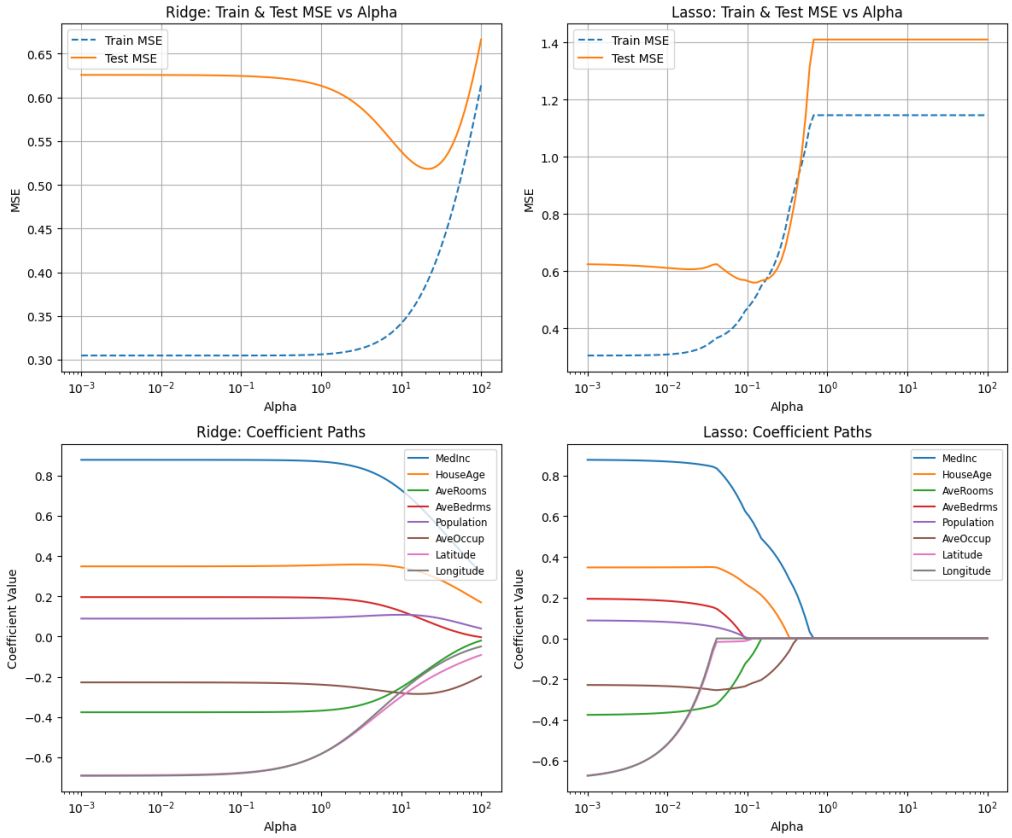

図 2.1　リッジ回帰・Lasso 回帰 ($n = 100$)

```
83 axs[1, 1].set_title('Lasso: Coefficient Paths')
84 axs[1, 1].legend(feature_names, loc='best', fontsize='
       small')
85
86 plt.tight_layout()
87 plt.show()
```

2.1.5　ロジスティック回帰

ロジスティック回帰は、線形モデルを分類問題に拡張したものであり、一般化線形モデル（GLM）の一種である。リンク関数としてロジット関数を用いることで、確率を出力する：

$$P(y = 1 \mid \boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{\beta}^T \boldsymbol{x})}$$

このモデルの学習には、最小二乗法ではなく、対数尤度の最大化を行う。解は解析的には得られないため、数値最適化法が用いられる。

2.1.6　ベイズ推定と正則化の関係

正則化は、ベイズ推定の観点から自然に解釈できる。

- リッジ回帰は、係数 $\boldsymbol{\beta}$ に対してゼロ平均の正規分布を事前分布としたときの MAP 推定に対応する。
- Lasso 回帰は、ラプラス分布（尖った分布）を事前分布としたときの MAP 推定に対応する。

このように、正則化はベイズ的には事前知識をモデルに反映させる手法として位置づけられる。

2.2　基底関数とカーネル法による非線形予測

2.2.1　基底関数による線形モデルの拡張

線形モデルは、説明変数に対して線形な関係を仮定するが、入力を変換することで非線形な関係も表現可能である。具体的には、入力 $\boldsymbol{x} \in \mathbb{R}^d$ を基底関数 $\phi(\boldsymbol{x}) \in \mathbb{R}^M$ によって変換し、その上で線形回帰を行う：

$$f(\boldsymbol{x}) = \sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})$$

ここで、$\phi_j(\boldsymbol{x})$ は入力に対する非線形な変換（例：多項式、ガウス関数、シグモイドなど）である。

この枠組みでは、非線形な入力に対しても予測モデルは依然としてパラメータ $\boldsymbol{w}$ に対して線形であるため、線形回帰と同様の解析手法が使える。

2.2.2 カーネル法の導入

基底関数を使うと柔軟な表現が可能になるが、基底関数の数が多くなると、計算量やメモリの問題が生じる。そこで、明示的に基底関数を用いずに、内積だけを通じて学習を行う方法が「カーネル法」である。

入力 $\boldsymbol{x}, \boldsymbol{x}'$ に対し、以下のような関数 k を定義する：

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}')$$

このような関数 k を「カーネル関数」と呼ぶ。

2.2.3 再生核ヒルベルト空間（RKHS）の簡単な導入

カーネル法の理論的基盤として、「再生核ヒルベルト空間（Reproducing Kernel Hilbert Space; RKHS）」がある。

RKHS は、関数空間の一種であり、各点での評価が内積によって再現できる性質（再生性）を持つ。すなわち、ある関数 $f \in \mathcal{H}$ に対して、以下が成り立つ：

$$f(\boldsymbol{x}) = \langle f, k(\cdot, \boldsymbol{x}) \rangle_{\mathcal{H}}$$

この性質により、学習アルゴリズムがカーネル関数を通して関数空間上の最適化問題として定式化できる。

2.2.4 カーネル回帰と Representer Theorem

リッジ回帰をカーネル法に拡張し、以下の問題を考える：

$$\min_{f \in \mathcal{H}} \sum_{i=1}^{n} (y_i - f(\boldsymbol{x}_i))^2 + \lambda \|f\|_{\mathcal{H}}^2$$

このような無限次元の最適化問題に対して、「Representer Theorem（表現定理）」が重要な役割を果たす。この定理は、最適解が訓練データに対するカーネル関数の線形結合として表されることを保証する：

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x})$$

これにより、未知の関数 f を直接求めるのではなく、係数 α_i を求める有限次元の問題として扱えるようになる。

2.2.5　カーネル法の応用：非数値データへの展開

カーネル関数は、数値ベクトルだけでなく、文字列、グラフ、構造データなどにも定義可能である。たとえば、以下のような応用がある：

- 文字列類似度を測るスペクトラムカーネル（バイオインフォマティクス）
- グラフ構造を対象としたグラフカーネル（化学構造の分類など）
- 木構造や文法に基づいた構文カーネル（自然言語処理）

このように、カーネル法は入力の構造に応じて柔軟に拡張できる点が大きな利点である。

2.2.6　再生核ヒルベルト空間（RKHS）の補足

再生核ヒルベルト空間（RKHS）は、内積を備えた関数空間であり、すべての点 $\boldsymbol{x}$ に対して「点評価関数」 $f \mapsto f(\boldsymbol{x})$ が連続線形関数であるという性質を持つ。

再生核 $k(\boldsymbol{x}, \boldsymbol{x}')$ を持つ RKHS $\mathcal{H}$ では、以下の性質が成り立つ：

- 再生性：$f(\boldsymbol{x}) = \langle f, k(\cdot, \boldsymbol{x}) \rangle_{\mathcal{H}}$
- 対称性と正定値性：$k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$、および $\sum_{i,j} c_i c_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0$

具体例（ガウスカーネル）：

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2\sigma^2}\right)$$

このようなカーネルに対応する RKHS は、非常に滑らかな無限次元の関数空間となる。

2.2.7 Representer Theorem の証明スケッチ

Representer Theorem は、次のような形式の最適化問題：

$$\min_{f \in \mathcal{H}} \sum_{i=1}^{n} L(y_i, f(\boldsymbol{x}_i)) + \lambda \|f\|_{\mathcal{H}}^2$$

に対して、解 $f^* \in \mathcal{H}$ が以下の形で表せることを保証する：

$$f^*(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x})$$

証明スケッチ：

- $\mathcal{H}$ の任意の関数 $f \in \mathcal{H}$ を、訓練データ点に対応するカーネル関数の線形結合部分と、その直交補空間（それらに直交する関数）に分解する。
- 正則化項 $\|f\|_{\mathcal{H}}^2$ は直交分解の二乗和になり、目的関数を最小にするには直交補部分を 0 にするのが最適となる。
- よって、最適解は訓練データのみに依存するカーネルの線形結合で表現できる。

2.2.8 カーネル行列による解法の導出

Representer Theorem によれば、最適な関数 f は次のように表される：

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x})$$

これをすべての訓練データに対して適用し、ベクトル形式で表すと：

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

ここで：

- $K \in \mathbb{R}^{n \times n}$：$(i, j)$ 成分が $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ であるカーネル行列（Gram 行列）- $\boldsymbol{f} = [f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n)]^T$ - $\boldsymbol{\alpha} \in \mathbb{R}^n$：係数ベクトル

正則化付きの二乗誤差損失を最小化する問題：

$$\min_{\boldsymbol{\alpha}} \|\boldsymbol{y} - K\boldsymbol{\alpha}\|^2 + \lambda \boldsymbol{\alpha}^T K \boldsymbol{\alpha}$$

これを解くと、以下の正則方程式が得られる：

$$(K + \lambda I)\boldsymbol{\alpha} = \boldsymbol{y}$$

この連立方程式を解くことで、カーネル回帰モデルが構築できる。

Program 2.2　カーネル回帰

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.linear_model import Ridge, Lasso
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

# データの読み込み
data = fetch_california_housing()
X, y = data.data, data.target
feature_names = data.feature_names

# 学習・テストデータの選択
k, m = 100, 300
np.random.seed(42)
indices = np.random.permutation(len(X))
train_idx, test_idx = indices[:k], indices[k:k + m]
X_train, y_train = X[train_idx], y[train_idx]
X_test, y_test = X[test_idx], y[test_idx]

# 標準化
scaler = StandardScaler()
```

```
23 X_train_std = scaler.fit_transform(X_train)
24 X_test_std = scaler.transform(X_test)
25
26 # 正則化パラメータ
27 alphas = np.logspace(-3, 2, 100)
28
29 # との誤差と係数追跡RidgeLasso
30 ridge_mse_train, ridge_mse_test = [], []
31 lasso_mse_train, lasso_mse_test = [], []
32 ridge_coefs, lasso_coefs = [], []
33
34 for alpha in alphas:
35     ridge = Ridge(alpha=alpha).fit(X_train_std, y_train)
36     lasso = Lasso(alpha=alpha, max_iter=10000).fit(
          X_train_std, y_train)
37
38     ridge_mse_train.append(mean_squared_error(y_train,
          ridge.predict(X_train_std)))
39     ridge_mse_test.append(mean_squared_error(y_test,
          ridge.predict(X_test_std)))
40
41     lasso_mse_train.append(mean_squared_error(y_train,
          lasso.predict(X_train_std)))
42     lasso_mse_test.append(mean_squared_error(y_test,
          lasso.predict(X_test_std)))
43
44     ridge_coefs.append(ridge.coef_)
45     lasso_coefs.append(lasso.coef_)
46
```

```
47 # プロット
48 fig, axs = plt.subplots(2, 2, figsize=(12, 10))
49
50 # Ridge: 訓練テスト/ MSE
51 axs[0, 0].plot(alphas, ridge_mse_train, label='Train MSE
       ', linestyle='--')
52 axs[0, 0].plot(alphas, ridge_mse_test, label='Test MSE')
53 axs[0, 0].set_xscale('log')
54 axs[0, 0].set_xlabel('Alpha')
55 axs[0, 0].set_ylabel('MSE')
56 axs[0, 0].set_title('Ridge: Train & Test MSE vs Alpha')
57 axs[0, 0].legend()
58 axs[0, 0].grid(True)
59
60 # Lasso: 訓練テスト/ MSE
61 axs[0, 1].plot(alphas, lasso_mse_train, label='Train MSE
       ', linestyle='--')
62 axs[0, 1].plot(alphas, lasso_mse_test, label='Test MSE')
63 axs[0, 1].set_xscale('log')
64 axs[0, 1].set_xlabel('Alpha')
65 axs[0, 1].set_ylabel('MSE')
66 axs[0, 1].set_title('Lasso: Train & Test MSE vs Alpha')
67 axs[0, 1].legend()
68 axs[0, 1].grid(True)
69
70 # Ridge: 係数パス
71 axs[1, 0].plot(alphas, ridge_coefs)
72 axs[1, 0].set_xscale('log')
73 axs[1, 0].set_xlabel('Alpha')
```

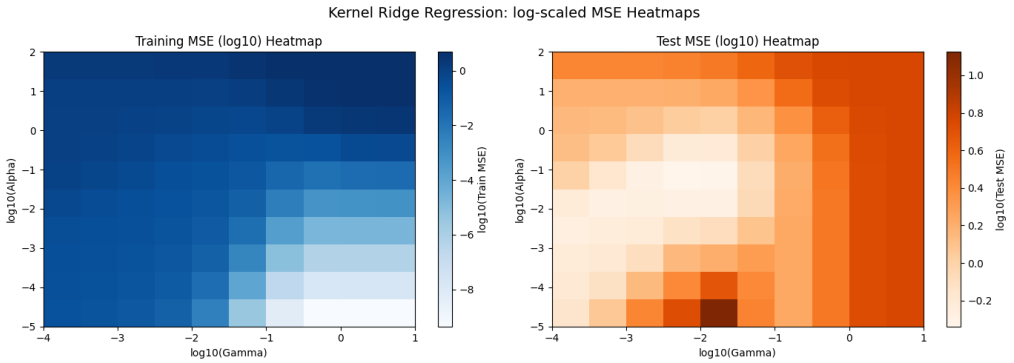

図 2.2　カーネルリッジ回帰 ($n = 100$)

```
74 axs[1, 0].set_ylabel('Coefficient Value')
75 axs[1, 0].set_title('Ridge: Coefficient Paths')
76 axs[1, 0].legend(feature_names, loc='best', fontsize='
       small')
77
78 # Lasso: 係数パス
79 axs[1, 1].plot(alphas, lasso_coefs)
80 axs[1, 1].set_xscale('log')
81 axs[1, 1].set_xlabel('Alpha')
82 axs[1, 1].set_ylabel('Coefficient Value')
83 axs[1, 1].set_title('Lasso: Coefficient Paths')
84 axs[1, 1].legend(feature_names, loc='best', fontsize='
       small')
85
86 plt.tight_layout()
87 plt.show()
```

2.2.9　補足：線形モデルにおける交差検証誤差の効率的計算

重回帰モデルやカーネルリッジ回帰では、最適な出力予測ベクトル $\hat{\boldsymbol{y}}$ が以下のような形で書ける：

$$\hat{\boldsymbol{y}} = H\boldsymbol{y}$$

ここで、H は入力行列 X のみに依存する行列（ハット行列、またはスムージング行列と呼ばれる）であり、$\boldsymbol{y}$ には依存しない。この H の対角成分 H_{ii} は、i 番目のデータに対する予測の影響度（自己影響）を表す。

Leave-one-out 交差検証（LOOCV）

交差検証は、汎化性能を評価するために用いられる手法であり、データを訓練用と検証用に分割して誤差を評価する。特に Leave-one-out 交差検証（LOOCV）は、n 個のデータのうち 1 つをテストに用い、残り $n-1$ 個で学習を行い、これを n 回繰り返す方法である。

この方法は、汎化誤差に近い評価を得られる一方で、通常は n 回の学習が必要であり計算コストが高い。しかし、線形モデルで $\hat{\boldsymbol{y}} = H\boldsymbol{y}$ の形を取る場合には、次の公式により 1 回の学習結果から LOOCV 誤差を計算することができる：

$$\mathrm{CV}_{\mathrm{LOO}} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{1 - H_{ii}}\right)^2$$

ここで、$\hat{y}_i = (H\boldsymbol{y})_i$ は全データを用いて学習したときの i 番目の予測値である。(証明は Wahba 1990, 赤穂 2008, 萩原 2022 など参照) これは PRESS (PREdiction Sum of Squares) 統計量とも呼ばれている（一般の k-fold CV についても同様の式が一応存在する）.

■まとめ　線形モデルで予測が $\hat{\boldsymbol{y}} = H\boldsymbol{y}$ の形で書ける場合には、LOOCV 誤差は H の対角成分と全データでの予測値を使って効率的に計算できる。この性質は、計算コストの削減だけでなく、解析的な誤差評価の導出にも有用である。

2.3 サポートベクターマシン（SVM）

2.3.1 マージン最大化による分類

サポートベクターマシン（Support Vector Machine; SVM）は、分類問題において「マージン（margin）」と呼ばれる量を最大化することにより、分類器の汎化性能を高めることを目的とした手法である。

ここでは、2 クラス分類（$y_i \in \{-1, +1\}$）を考える。

分類平面は、次のような線形関数によって定義される：

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$$

このとき、すべてのデータが以下の不等式を満たす場合、線形分離が可能である：

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1$$

この条件のもとで、マージン（境界から最も近い点までの距離）を最大化するためには、以下の最適化問題を解く：

$$\min_{\boldsymbol{w}, b} \ \frac{1}{2}\|\boldsymbol{w}\|^2 \quad \text{subject to } y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 \quad (\forall i)$$

この最適化により、分類平面の傾きを小さく（＝マージンを広く）することができる。

2.3.2 ソフトマージンと損失関数の定式化

実際のデータは完全に線形分離できるとは限らないため、**ソフトマージン**として誤分類を許容する形式に拡張する。これにより損失関数が導入され、以下のような目的関数になる：

$$\min_{\boldsymbol{w}, b} \ \frac{1}{2}\|\boldsymbol{w}\|^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b))$$

ここで：

- 第 1 項：2 次正則化項（モデルの複雑さの抑制）
- 第 2 項：**ヒンジ損失関数**（hinge loss）

$$\ell_{\text{hinge}}(y, f(\boldsymbol{x})) = \max(0, 1 - yf(\boldsymbol{x}))$$

- $C > 0$：誤分類に対するペナルティの強さを調整するパラメータ

ヒンジ損失は、分類境界から離れていないデータ点に対してのみ損失を与える「区分線形関数」であり、サポートベクトルの選定に寄与する。

2.3.3　カーネルによる非線形分類

SVM はカーネル法と組み合わせることで、非線形な分類問題にも対応できる。入力を明示的に変換せずに、内積の代わりにカーネル関数 $k(\boldsymbol{x}, \boldsymbol{x}')$ を用いることで、次のような非線形分類器が得られる：

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b$$

このとき、係数 α_i の多くはゼロになる。これにより、分類器は訓練データのうち一部の点（**サポートベクトル**）のみに依存する。

2.3.4　サポートベクトルと解のスパース性

SVM の双対問題を解くことで得られる解の特徴として、**スパース性（疎性）**がある。すなわち、学習によって得られる係数 α_i は、多くが 0 であり、非ゼロのものだけが分類器に寄与する。

このようなデータ点を「**サポートベクトル**」と呼び、分類境界に近い、あるいは誤分類されている点に対応する。これにより、学習済みモデルはシンプルで効率的になり、予測計算も高速である。

まとめると：

- SVM はマージン最大化により汎化性能の高い分類器を構成する。
- 区分線形のヒンジ損失と 2 次正則化によって損失を定式化する。

- カーネル法によって非線形分類も可能となり、訓練データの一部（サポートベクトル）のみがモデルに寄与する。

2.3.5 カーネル SVM の双対定式化と凸最適化の概要

SVM は主問題（primal problem）を双対問題（dual problem）に変換することで、効率的に解くことができる。特にカーネルを用いた場合、双対問題の形式が重要になる。

ソフトマージン SVM の双対問題は、以下のように定式化される：

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned}$$

この問題は **凸 2 次計画問題（quadratic programming, QP）** であり、グローバルな最適解が保証される。

双対解 $\boldsymbol{\alpha}$ を用いて、分類関数は以下のように表現される：

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b$$

この形式により、入力空間での非線形な分類境界を効率的に学習することができる。

2.3.6 SVM とロジスティック回帰の比較：損失関数の観点から

SVM とロジスティック回帰は、どちらも分類問題を扱うが、損失関数の設計に大きな違いがある。

- **ロジスティック回帰**：確率的モデルであり、ロジスティック関数を用いてクラスの事後確率をモデル化する。損失関数は以下の **対数損失（log loss）**：

$$\ell_{\log}(y, f) = \log(1 + \exp(-yf))$$

- **SVM**：マージン最大化を目的とし、損失関数は **ヒンジ損失**：

$$\ell_{\text{hinge}}(y, f) = \max(0, 1 - yf)$$

- **違いの直感**：ロジスティック損失は全ての誤差に滑らかにペナルティを与えるが、SVM のヒンジ損失は「マージンの外側」にある点（分類に十分余裕がある点）には損失を与えない。

ロジスティック回帰は出力を確率として解釈可能だが、SVM は分類境界に特化しており、確率の解釈は難しいという違いもある。

2.3.7　サポートベクトルと決定境界の直感的理解

SVM では、分類境界（decision boundary）はマージンの幅を最大化するように決定される。このとき、分類器の決定に直接影響するのは、マージン上またはマージン内にあるデータ点だけである。

これらの点を「**サポートベクトル**」と呼び、それ以外の点（分類に余裕がある点）は分類境界に寄与しない。

- サポートベクトルは、分類平面からの距離が最も近い点。
- サポートベクトルの位置が変われば、分類境界も変わる。
- 結果として、モデルは「データの重要な一部」にのみ依存するため、**スパース性**を持ち、計算効率も高くなる。

図示すると、分類境界の両側に平行なマージン線が引かれ、それに接する点がサポートベクトルとして強調される。

2.4　決定木とアンサンブル学習

2.4.1　決定木の定義と最適化

決定木（Decision Tree）は、データを特徴ごとに条件分岐させて分類または回帰を行うモデルである。分類では「はい／いいえ」の質問を繰り返してデータを分類し、回帰では数値予測を行う。

決定木は、根（root）から始まり、各内部ノードで特徴量に基づく条件分岐を行い、葉（leaf）に到達したところで予測を出力する。

分割基準と目的関数：

ノードでの最適な分割を決めるためには、「純度（purity）」の尺度を用いて評価を行う。代表的な指標は以下の通り：

- **ジニ不純度（Gini impurity）**：

$$G(t) = \sum_{k=1}^{K} p_k(1 - p_k) = 1 - \sum_{k=1}^{K} p_k^2$$

- **エントロピー（情報利得）**：

$$H(t) = -\sum_{k=1}^{K} p_k \log p_k$$

ここで p_k はノード t におけるクラス k の出現確率である。

各ノードでは、特徴量としきい値の組み合わせを全探索し、**純度の改善（不純度の減少）**が最大になる分割を選択する。

剪定（pruning）：

木を成長させすぎると過学習するため、適切な深さで止める必要がある。これには：

- 事前剪定（max depth, min samples split など）
- 事後剪定（学習後に不要な枝を削除）

がある。

2.4.2　具体例：Breast Cancer Wisconsin データによる識別問題

データセットの概要

Breast Cancer Wisconsin (Diagnostic) Dataset は、乳がん診断に関する特徴量を用いて、腫瘍が良性か悪性かを識別する分類問題である。UCI Machine Learning Repository で広く使われているベンチマークデータセットの一つである。

- サンプル数：569

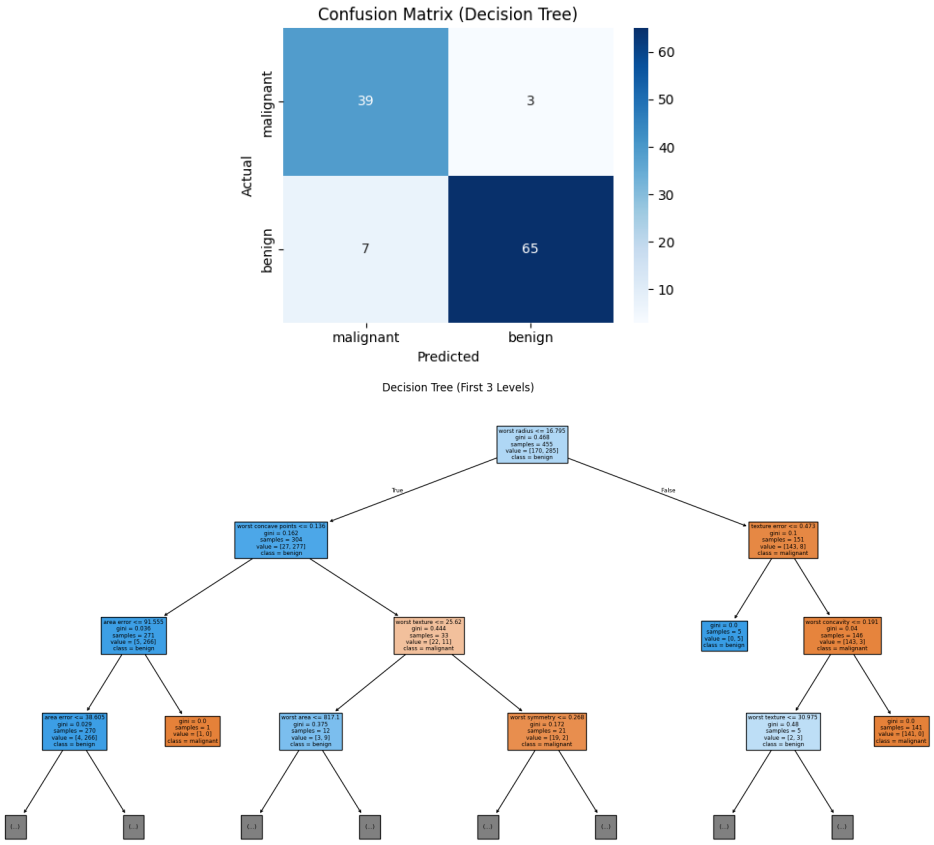

図 2.3　決定木

- 特徴量数：30（細胞核の形状・テクスチャ・大きさ・滑らかさなど）
- クラスラベル：
  - 0：悪性（malignant）
  - 1：良性（benign）

すべての特徴量は数値（実数）であり、スケーリング処理を行うことで多くの分類器に適用可能となる。

決定木の結果

Program 2.3　決定木

表 2.1　Breast Cancer Wisconsin データセットの特徴量一覧（インデックス付き）

インデックス	カテゴリ	特徴量名（日本語訳）
0	mean	radius_mean（半径の平均）
1	mean	texture_mean（テクスチャの平均）
2	mean	perimeter_mean（周囲長の平均）
3	mean	area_mean（面積の平均）
4	mean	smoothness_mean（平滑度の平均）
5	mean	compactness_mean（緻密さの平均）
6	mean	concavity_mean（凹度の平均）
7	mean	concave_points_mean（凹点数の平均）
8	mean	symmetry_mean（対称性の平均）
9	mean	fractal_dimension_mean（フラクタル次元の平均）
10	se	radius_se（半径の標準誤差）
11	se	texture_se（テクスチャの標準誤差）
12	se	perimeter_se（周囲長の標準誤差）
13	se	area_se（面積の標準誤差）
14	se	smoothness_se（平滑度の標準誤差）
15	se	compactness_se（緻密さの標準誤差）
16	se	concavity_se（凹度の標準誤差）
17	se	concave_points_se（凹点数の標準誤差）
18	se	symmetry_se（対称性の標準誤差）
19	se	fractal_dimension_se（フラクタル次元の標準誤差）
20	worst	radius_worst（最大半径）
21	worst	texture_worst（最大テクスチャ）
22	worst	perimeter_worst（最大周囲長）
23	worst	area_worst（最大面積）
24	worst	smoothness_worst（最大平滑度）
25	worst	compactness_worst（最大緻密さ）
26	worst	concavity_worst（最大凹度）
27	worst	concave_points_worst（最大凹点数）
28	worst	symmetry_worst（最大対称性）
29	worst	fractal_dimension_worst（最大フラクタル次元）

複雑な識別器の比較

このデータセットに対して、以下の代表的な分類モデルを適用し、性能を比較する。

- **正則化ロジスティック回帰**：L2 正則化付きの線形分類器
- **SVM（RBF カーネル）**：非線形分類に対応するサポートベクターマシン
- **ランダムフォレスト**：多数の決定木によるバギング型アンサンブル
- **勾配ブースティング**：残差を逐次的に学習するブースティング型アンサンブル

ハイパーパラメータと交差検証による評価

各モデルに対して、主要なハイパーパラメータ（例：正則化強度、木の深さ、学習率など）を変化させ、**5 分割交差検証（5-fold cross-validation）**により汎化性能を評価する。

評価指標には、**ROC AUC（受信者操作特性曲線の下面積）**を用いる。これはクラス不均衡にも頑健であり、識別性能の全体的な比較に適している。

アンサンブルモデルの解釈性

ランダムフォレストおよび勾配ブースティングモデルにおいては、**特徴量の重要度（feature importance）**を計算することで、どの特徴が分類に大きく寄与しているかを可視化できる。

さらに、**SHAP（SHapley Additive exPlanations）**を用いることで、各予測に対してどの特徴量がどのように影響したかを定量的に説明できる。SHAP は以下のような可視化を可能にする：

- **summary plot**：全体の特徴量重要度と影響方向を可視化
- **waterfall plot**：個々の予測に対する特徴量の貢献を可視化

分類性能の詳細評価

最良のモデルについては、以下のような観点で分類性能を詳しく評価する：

- **混同行列**：分類結果の概要（TP, TN, FP, FN）を可視化
- **感度（sensitivity）と特異度（specificity）**の算出
- **ROC 曲線**および **Precision-Recall 曲線**の描画

これにより、単純な正解率では捉えきれない分類性能の詳細な分析が可能となる。

Stratified k-fold 交差検証

機械学習において、**交差検証（cross-validation）**は、モデルの汎化性能を評価するための標準的な手法である。その中でも、**Stratified k-fold 交差検証**は、**クラスラベルの不均衡がある分類問題**において特に有効な交差検証法である。

■k-fold 交差検証の概要　k-fold 交差検証では、以下の手順でモデル評価を行う：

1. データセットをほぼ同じサイズの k 個のブロック（fold）に分割する。
2. 各回で 1 つのブロックを**検証用**、残りの $k-1$ 個を**訓練用**とし、モデルを学習・評価する。
3. これを k 回繰り返し、各回の評価結果の平均を最終的な性能指標とする。

この方法により、データの分割による偏りを平均化し、**安定した性能評価**が可能になる。

■Stratified k-fold の特徴　通常の k-fold 検証では、各 fold に含まれるクラス分布が不均衡になることがある。特に、少数クラスの割合が大きく変動すると、評価結果が不安定になる可能性がある。

これに対し、**Stratified k-fold** では：

- 各 fold において、元のデータセットと**同じクラス分布**を保つようにサンプリングを行う。
- 少数クラスの学習や評価に偏りが生じにくく、**より信頼性の高い評価**が得られる。

■実装例（scikit-learn）　Python の scikit-learn では、以下のようにして Stratified k-fold を指定することができる：

```
1 from sklearn.model_selection import StratifiedKFold
2
3 cv = StratifiedKFold(n_splits=5, shuffle=True,
     random_state=42)
4 scores = cross_val_score(model, X, y, cv=cv, scoring='
     accuracy')
```

このようにして、交差検証におけるデータ分割の**公平性**と**再現性**を確保することができる。

■まとめ　Stratified k-fold 交差検証は、特に分類問題においてクラスの不均衡がある場合に有効であり、モデルの比較やハイパーパラメータの選択時に広く用いられる。アンサンブル学習の評価でも、**公平な性能評価を行うために重要な前処理ステップ**である。

2.4.3　アンサンブル学習の基本的な考え方

アンサンブル学習（Ensemble Learning）は、複数のモデル（基学習器）を組み合わせて、個々のモデルよりも高い精度を実現する手法である。

バギング（Bagging：Bootstrap Aggregating）

バギングは、同じ学習アルゴリズムを用いて、元の訓練データから複数のサブセット（ブートストラップ標本）を作成し、それぞれに学習を行う。

- 各モデルは独立に学習される（並列処理が可能）- 最終的な予測は分類では多数決、回帰では平均をとる- **モデルの分散を低減**し、過学習を防ぐ効果がある

代表例：**ランダムフォレスト**（次節で詳述）

ブースティング（Boosting）

ブースティングは、複数の弱い学習器（性能のよくないモデル）を逐次的に学習させて、それらを組み合わせて性能の良い強い学習器を構成する方法である。

- 各ステップでは、前のモデルの誤りを重視して新しいモデルを学習する- 予測は加重平均（または加重多数決）で行われる- **バイアスを低減**する効果があるが、過学習に注意が必要

代表例：**勾配ブースティング**（次節で詳述）

バギング vs ブースティング（まとめ）：

- バギング：高分散なモデルに対して効果的（例：決定木）
- ブースティング：高バイアスな問題に対して効果的（例：線形的に分離困難なデータ）
- バギングは独立に学習、ブースティングは逐次的に学習

2.4.4 勾配ブースティングの具体的アルゴリズム（回帰：二乗誤差の場合）

ここでは、損失関数として **二乗誤差**を用いる回帰問題を考え、勾配ブースティングの具体的な学習ステップを説明する。

■**設定**　訓練データ：$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$

目的：予測関数 $f(\boldsymbol{x})$ を学習し、出力 $f(\boldsymbol{x}_i)$ が y_i に近づくようにする。

損失関数：

$$L(y, f(\boldsymbol{x})) = \frac{1}{2}(y - f(\boldsymbol{x}))^2$$

■アルゴリズム（学習ステップ）

1. **初期化**：

$$f_0(\boldsymbol{x}) = \arg\min_c \sum_{i=1}^n \frac{1}{2}(y_i - c)^2 = \frac{1}{n}\sum_{i=1}^n y_i$$

（全体の平均で初期化）

2. **for** $m = 1, 2, \ldots, M$（反復ステップ） **do**：

（a）**残差の計算（負の勾配）**：

$$r_i^{(m)} = - \left. \frac{\partial L(y_i, f(\boldsymbol{x}_i))}{\partial f} \right|_{f=f_{m-1}} = y_i - f_{m-1}(\boldsymbol{x}_i)$$

（b）**残差に対する回帰木の学習**：回帰木 $h_m(\boldsymbol{x})$ を学習し、$r_i^{(m)} \approx h_m(\boldsymbol{x}_i)$ を満たすようにする。

（c）**モデルの更新**：

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \eta \cdot h_m(\boldsymbol{x})$$

（$\eta \in (0, 1]$ は学習率。小さくすると学習は遅いが過学習しにくくなる。）

3. **最終モデル**：

$$f_M(\boldsymbol{x}) = f_0(\boldsymbol{x}) + \sum_{m=1}^{M} \eta \cdot h_m(\boldsymbol{x})$$

■ポイント

- 各ステップでは、「現在のモデルがうまく予測できていない誤差（残差）」を次のモデルで修正していく。
- 二乗誤差の場合、勾配は単に残差 $y - f(\boldsymbol{x})$ になるため、非常に直感的。
- 弱学習器（例：浅い決定木）を多数重ねることで、複雑で精度の高いモデルが得られる。

2.4.5　XGBoost：実用的な勾配ブースティングの高速実装

XGBoost（eXtreme Gradient Boosting） は、勾配ブースティングを高速かつ高精度に実装した実用的な手法であり、機械学習コンペティションなどで広く用いられている。

XGBoost の主な特徴は以下の通り：

- **2 次近似による目的関数の最適化**：勾配（1 次導関数）だけでなく、ヘッセ行列（2 次導関数）も用いることで、より安定かつ効率的に木の分割を決定する。

- **正則化項の導入**：各木の複雑さ（ノード数や葉の重み）に対するペナルティを目的関数に加え、過学習を防ぐ：

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

（T：葉の数、w_j：葉の出力値）

- **欠損値処理や並列計算の最適化**：自動的に欠損値に対応し、高速な分割探索を並列で行う。
- **扱いやすさと拡張性**：回帰・分類に加え、ユーザー定義の損失関数やカスタム目的関数も利用可能。

要するに、XGBoost は「精度」「速度」「汎用性」を兼ね備えた勾配ブースティングの実装として、実践的な場面で非常に有効な手法である。

Program 2.4　識別手法の比較

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import seaborn as sns
4  from sklearn.datasets import load_breast_cancer
5  from sklearn.model_selection import cross_val_score,
       StratifiedKFold
6  from sklearn.preprocessing import StandardScaler
7  from sklearn.linear_model import LogisticRegression
8  from sklearn.svm import SVC
9  from sklearn.ensemble import RandomForestClassifier,
       GradientBoostingClassifier
10
11 # データ読み込み
12 data = load_breast_cancer()
13 X, y = data.data, data.target
14 feature_names = data.feature_names
```

```
15
16 # 標準化
17 scaler = StandardScaler()
18 X_std = scaler.fit_transform(X)
19
20 # 交差検証設定
21 cv = StratifiedKFold(n_splits=5, shuffle=True,
       random_state=42)
22
23 # 各モデルとハイパーパラメータ設定
24 models = {
25     'LogisticRegression': [
26         LogisticRegression(C=c, penalty='l2', solver='
              liblinear') for c in [0.01, 0.1, 1, 10, 100]
27     ],
28     'SVM (RBF)': [
29         SVC(C=c, gamma=g, kernel='rbf', probability=True
              ) for c in [0.1, 1, 10] for g in [0.01, 0.1,
               1]
30     ],
31     'RandomForest': [
32         RandomForestClassifier(n_estimators=100,
              max_depth=d, random_state=42) for d in [2,
              4, 6, 8, None]
33     ],
34     'GradientBoosting': [
35         GradientBoostingClassifier(n_estimators=100,
              learning_rate=lr, max_depth=d, random_state
              =42)
```

```
        for lr in [0.01, 0.1, 0.2] for d in [2, 3, 4]
    ]
}

# 結果保存
results = []

print("=== Cross-Validation Performance (ROC AUC) ===")
for name, model_list in models.items():
    for model in model_list:
        scores = cross_val_score(model, X_std, y, cv=cv,
            scoring='roc_auc')
        results.append((name, str(model.get_params()),
            np.mean(scores)))
        print(f"{name:20s} | mean AUC: {np.mean(scores)
            :.4f}")

# グラフ化（平均スコアの比較）
import pandas as pd

df_results = pd.DataFrame(results, columns=["Model", "
    Params", "ROC AUC"])
plt.figure(figsize=(10, 5))
sns.boxplot(data=df_results, x="Model", y="ROC AUC")
plt.title("Model Comparison (ROC AUC, 5-fold CV)")
plt.xticks(rotation=20)
plt.tight_layout()
plt.show()
```

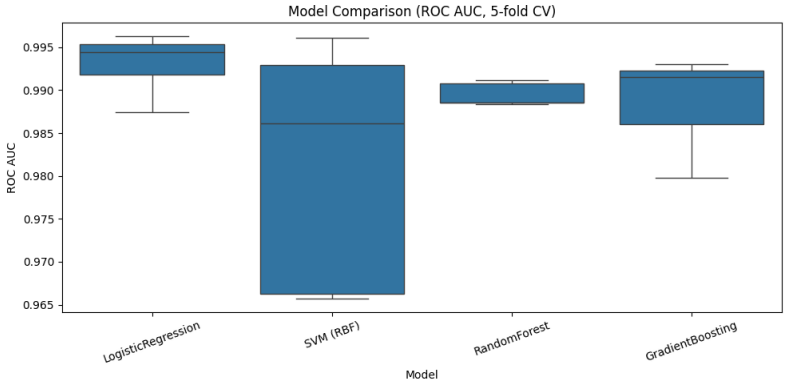

図 2.4　識別手法の比較

2.4.6　アンサンブル学習における変数重要度とモデルの解釈性

■動機：説明性と解釈性の重要性　機械学習モデルが高い精度を持つことは重要だが、**なぜその予測がなされたのか**を理解することも、実務では非常に重要である。特に以下のような文脈で、**モデルの説明性（explainability）や解釈性（interpretability）**が求められる：

- 医療・金融・法務などの高リスク分野における意思決定の裏付け
- モデル改善のための特徴量の選定や理解
- ステークホルダーへの結果の説明

このような背景から、**どの特徴量が予測に大きく寄与したか**を評価する指標として「変数重要度（feature importance）」が注目されている。

2.4.7　変数重要度の一般的な考え方

変数重要度とは、各特徴量が予測に与える寄与の大きさを定量的に評価したものである。一般的な評価方法には以下がある：

- **摂動ベース（permutation importance）**：特定の特徴量の値をランダムに入れ替えたときに、モデルの性能がどれだけ低下するかを見る。
- **SHAP（SHapley Additive exPlanations）**：単一の予測に対して、各特

徴量がどれだけ予測値に寄与したかを、ゲーム理論に基づく公正な方法で分配する。

SHAP は以下のような特徴を持つ：

- 予測単位での寄与の分解が可能
- 全体で平均すればグローバルな重要度になる
- アンサンブル木モデルとの相性が良く、高速な専用実装（TreeSHAP）が存在

2.4.8 決定木アンサンブルに特化した重要度の評価

ランダムフォレストや勾配ブースティングなどの**決定木ベースのアンサンブル学習モデル**では、モデルの構造を利用した変数重要度の指標が自然に得られる。

■1. 分割による不純度減少の累積（Gain ベース） 各特徴量について、**その特徴でノードを分割した際に得られる不純度の減少量（例：ジニ不純度の減少、エントロピーの減少）**を木全体にわたって合計する：

$$\text{Importance}(x_j) = \sum_{\text{分割に使われたノード}} \text{不純度の減少量}$$

この指標は簡単に得られ、高速に計算可能だが、以下のような注意点がある：

- 値のスケールやカテゴリ数の多さに敏感（偏りが出ることがある）
- 他の特徴と相関がある場合、重要度が過小または過大評価されることがある

Program 2.5 変数重要性

```
1 # ランダムフォレストと勾配ブースティングの学習
2 rf = RandomForestClassifier(n_estimators=100, max_depth
      =4, random_state=42)
3 gb = GradientBoostingClassifier(n_estimators=100,
      learning_rate=0.1, max_depth=3, random_state=42)
4
```

```
rf.fit(X_std, y)
gb.fit(X_std, y)

# 特徴量重要度の表示
plt.figure(figsize=(10, 4))
plt.barh(feature_names, rf.feature_importances_)
plt.title("Random Forest Feature Importance")
plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 4))
plt.barh(feature_names, gb.feature_importances_)
plt.title("Gradient Boosting Feature Importance")
plt.tight_layout()
plt.show()
```

■2. TreeSHAP による厳密な貢献度推定　TreeSHAP は、SHAP のアイデアを決定木アンサンブルに特化して効率的に計算する手法である。各予測に対して、特徴量の貢献度を定量的に割り当てることができる。

特徴：

- 各特徴量の影響を厳密に分離して評価
- モデル全体の平均で集約すれば、グローバルな重要度が得られる
- 実行速度も高く、大規模データにも適用可能

■まとめ

- 決定木アンサンブルでは、構造情報を活かして変数重要度を定量化できる。
- より正確かつ局所的な解釈には SHAP（特に TreeSHAP）が有効。
- モデルの解釈性は、単なる精度評価を超えて、実用において重要な視点である。

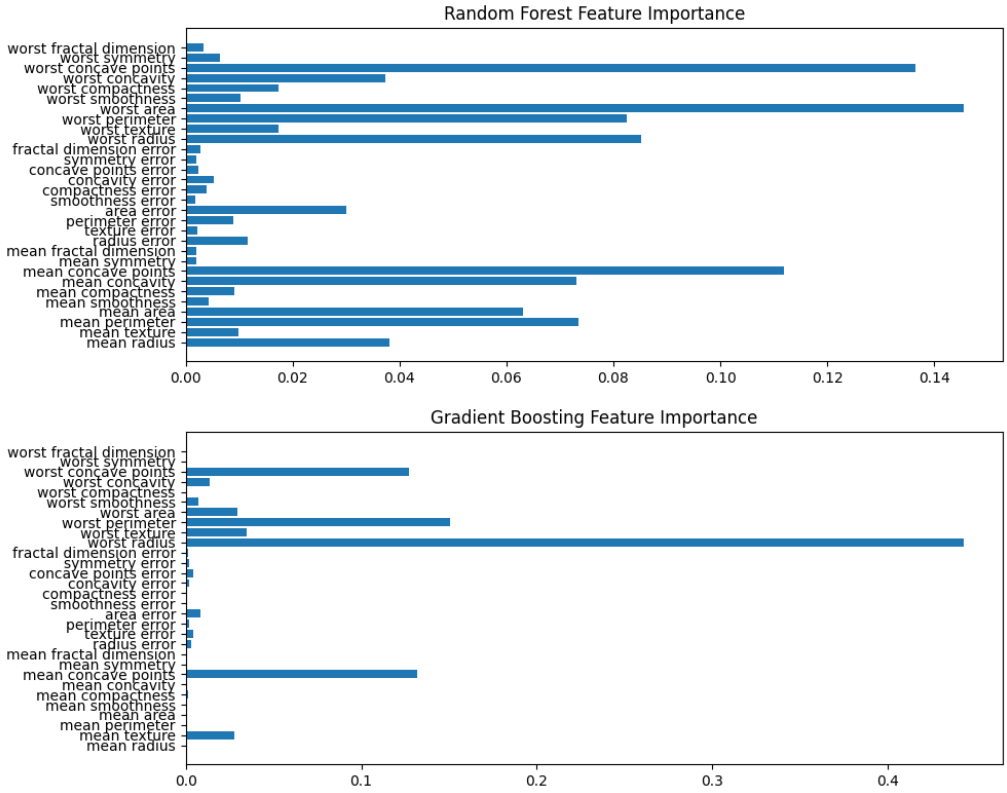

図 2.5　変数重要性

2.4.9　補足：Shapley 値の数学的定義

SHAP における特徴量の貢献度の評価は、ゲーム理論における **Shapley 値**（Shapley value）に基づいている。

Shapley 値は、協力ゲームにおいて、各プレイヤー（ここでは特徴量）が最終的な成果（ここでは予測値）にどれだけ貢献したかを、公平に分配するための理論である。

Shapley 値の定義：

特徴量の集合を $N = \{1, 2, \ldots, d\}$ とし、任意の部分集合 $S \subseteq N \setminus \{j\}$ に対する貢献関数（予測値の期待値）を $v(S)$ とする。

特徴量 j に対する Shapley 値 ϕ_j は次のように定義される：

$$\phi_j = \sum_{S \subseteq N \setminus \{j\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \left(v(S \cup \{j\}) - v(S) \right)$$

ここで：

- $v(S)$：特徴量集合 S の情報のみを使ったときの予測値の期待値
- $v(S \cup \{j\}) - v(S)$：特徴量 j を追加することで得られる **限界貢献**
- 係数は全ての特徴量の順列における順序に基づく重み（公平性の保証）

この定義は、以下の公正性を満たすことが知られている：

- **効率性**：すべての特徴量の貢献度の合計が予測値と一致する
- **対称性**：同じ貢献をする特徴量には同じ値が割り当てられる
- **ダミー性**：貢献しない特徴量の値はゼロになる
- **加法性**：複数モデルに対して Shapley 値を加えると全体の貢献が合計される

SHAP はこの Shapley 値を機械学習の予測に適用し、各予測に対する特徴量の寄与を正確に定量化する方法として利用されている。

Program 2.6　SHAP

```
# 値の可視化SHAP
import shap

# Gradient Boosting に対して SHAP 実行 () TreeExplainer
explainer = shap.Explainer(gb, X_std)
shap_values = explainer(X_std)

# サマリープロット（グローバルな重要度）
shap.summary_plot(shap_values, X_std, feature_names=
    feature_names)

# ウォーターフォールプロット（局所的説明）
shap.plots.waterfall(shap_values[0], max_display=10)
```

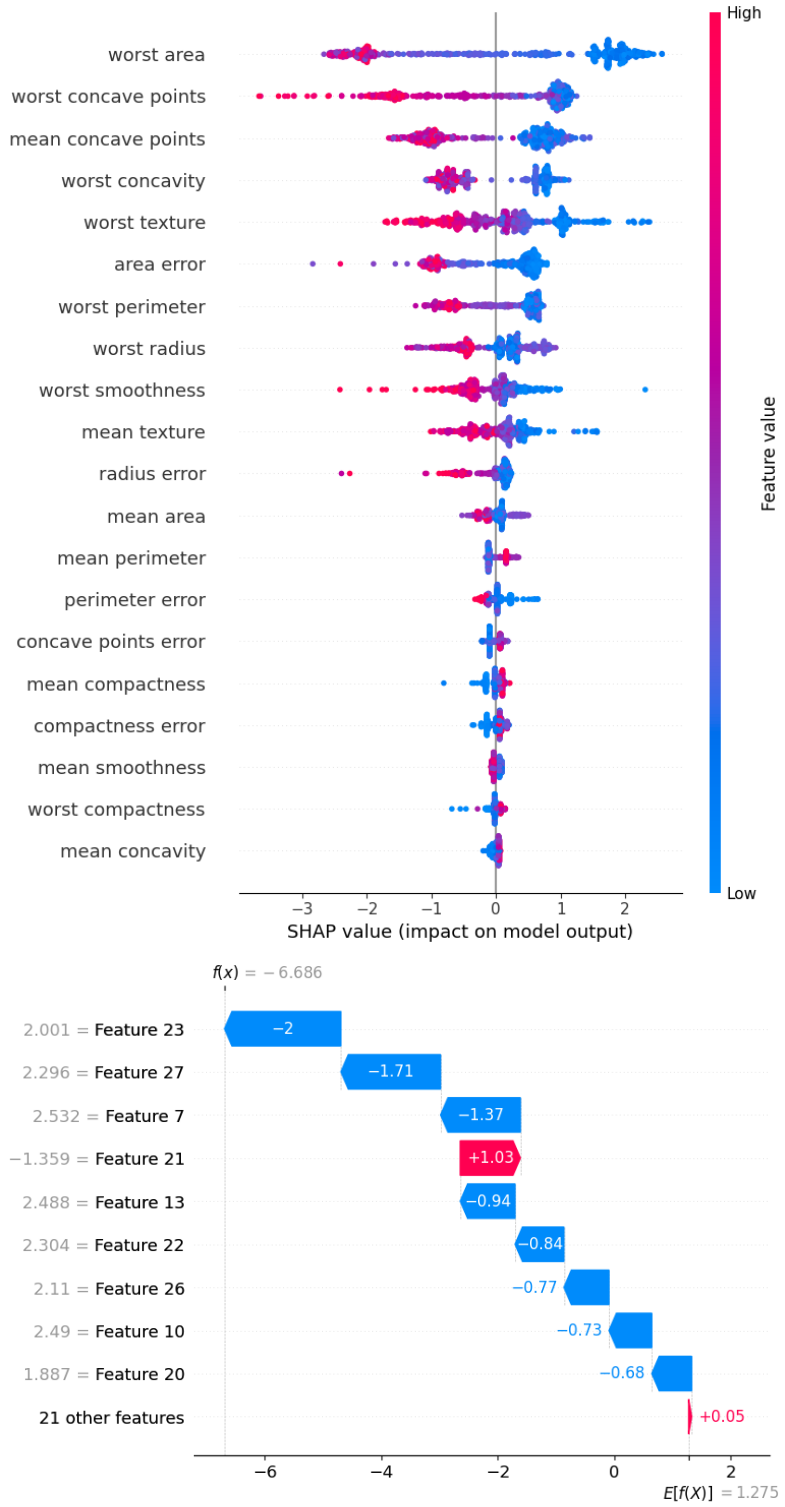
worst area
worst concave points
mean concave points
worst concavity
worst texture
area error
worst perimeter
worst radius
worst smoothness
mean texture
radius error
mean area
mean perimeter
perimeter error
concave points error
mean compactness
compactness error
mean smoothness
worst compactness
mean concavity
−3
−2
−1
0
1
2
SHAP value (impact on model output)
High
Feature value
Low
f(x) = − 6.686
2.001 = Feature 23
2.296 = Feature 27
2.532 = Feature 7
−1.359 = Feature 21
2.488 = Feature 13
2.304 = Feature 22
2.11 = Feature 26
2.49 = Feature 10
1.887 = Feature 20
21 other features
−2
−1.71
−1.37
+1.03
−0.94
−0.84
−0.77
−0.73
−0.68
+0.05
−6
−4
−2
0
2
E[f(X)] = 1.275

図 2.6　SHAP

2.5　多クラス分類と誤り訂正符号による拡張

2.5.1　多クラス分類の基本的アプローチ

多クラス分類とは、ラベルが 2 つ以上ある分類問題（例：クラス数 $K \geq 3$）を扱うものである。多くの機械学習アルゴリズム（特に識別器）は 2 クラス分類を前提としているため、**多クラス分類に拡張するための方法**が重要となる。

2.5.2　One-vs-Rest（OvR）

One-vs-Rest（あるいは One-vs-All）は、各クラス k に対して「クラス k vs その他すべて」の 2 クラス分類器を学習する方法である。

- 合計 K 個の分類器を学習
- 予測時には、すべての分類器の出力を比較し、最もスコアの高いクラスを選択

利点：

- 実装が簡単で汎用的な 2 値分類器をそのまま利用可能

欠点：

- クラスの不均衡に弱い（ある vs 多数）
- 各分類器のスコアが比較可能であることが前提となる

2.5.3　One-vs-One（OvO）

One-vs-One は、すべてのクラスのペアに対して分類器を学習する方法である。クラス数が K のとき、分類器の数は：

$$\frac{K(K-1)}{2}$$

- 予測時には、全ての分類器で多数決を行う

利点：

- 各分類器は 2 つのクラスだけに集中できるため、学習がしやすい
- クラス間の分離が鮮明な場合に有効

欠点：

- 分類器数が多くなり、学習・推論コストが高い
- 投票が同数になった場合の扱いが不明確

2.5.4　ECOC（誤り訂正出力符号：Error-Correcting Output Codes）

ECOC は、**多クラス分類を複数の 2 値分類問題に分解する**ための汎用的なフレームワークであり、誤り訂正符号（error-correcting codes）のアイデアに基づいている。

■基本的な考え方　各クラスに対して、長さ L の**バイナリ符号（コードワード）**を割り当て、それぞれのビットごとに 2 クラス分類器を学習する。

- 各ビット位置に対して、全体を 2 つのグループ（ラベル）に分けて分類器を学習
- 新しいデータに対する予測は、L 個の分類器の出力ビット列と各クラスのコードワードの距離（例：Hamming 距離）で比較して、最も近いクラスを選ぶ

■誤り訂正符号としての性質　ECOC の核心は、「コード間距離を大きく取ることで誤分類を訂正できる」ことである。具体的には：

- クラスのコード間の Hamming 距離が d 以上ある場合、$\lfloor (d-1)/2 \rfloor$ 個までの分類器の誤りを訂正できる
- よって、分類器にある程度の誤りが含まれても、最終予測は正しくなる可能性が高い

■利点と応用

- 複雑なクラス間関係を捉える柔軟な設計が可能
- クラス数が多い場合でも、誤りに対して頑健な分類が可能
- OvR や OvO も ECOC の特別な場合とみなすことができる

■まとめ

- 多クラス分類は 2 値分類器を工夫して組み合わせることで実現可能
- OvR と OvO は簡便かつ実用的だが、それぞれトレードオフがある
- ECOC は符号理論の考え方を応用し、頑健性の高い多クラス分類手法として汎用的に使える

2.5.5　補足：誤り訂正符号としての数式的性質

ECOC は、符号理論における誤り訂正符号と同様に、「クラスのコードワード同士の距離」を利用して分類誤りの訂正を行う。

■定義：Hamming 距離　2 つのバイナリ列 $\boldsymbol{c}_1, \boldsymbol{c}_2 \in \{0,1\}^L$ の間の Hamming 距離 d_H は、異なるビットの数である：

$$d_H(\boldsymbol{c}_1, \boldsymbol{c}_2) = \sum_{i=1}^{L} \mathbb{I}[c_{1i} \neq c_{2i}]$$

■誤り訂正可能な条件　すべてのクラス間のコードワードの最小 Hamming 距離を $d_{\min}$ としたとき、ECOC は最大で

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor$$

個の分類器の誤りを訂正できる。これは、符号理論における誤り訂正の基本原理と一致している。

この性質により、分類器の一部が誤った予測をしても、最終的な予測結果は正しいクラスに復元される可能性が高まる。

2.5.6 補足：不均衡データにおける多クラス分類の難しさ

多クラス分類では、**クラス間のデータ数の偏り（不均衡性）**が大きな問題となる。これは以下のような影響を及ぼす：

- **学習のバイアス**：多数派クラスに対して偏った分類器が学習されやすい
- **スコアの不公平性**：特に One-vs-Rest では、正例と負例の数の比率が極端になる
- **評価指標の選択が重要**：単純な精度（accuracy）では性能を正しく測れない

このような問題に対処するためには：

- **データの再サンプリング**（オーバーサンプリング／アンダーサンプリング）
- **重み付き損失関数**（クラスごとの誤分類に異なる重みを設定）
- **適切な評価指標**（F1 スコア、AUC、マクロ平均など）

が必要となる。多クラス分類では、単に分類器を構築するだけでなく、**クラス不均衡の影響を考慮した学習と評価**が重要である。

2.6 最近傍識別器（k-nearest neighbor 法）

2.6.1 基本的な定義

最近傍法（nearest neighbor classifier）は、教師あり学習の中でも非常に単純かつ直感的な識別法である。分類対象となる入力ベクトル $\boldsymbol{x} \in \mathbb{R}^d$ に対して、訓練データ内の「近くにある」サンプルのラベルに基づいてクラスを予測する。

一般に、k 個の最近傍データ点を用いて多数決を行う手法を **k-最近傍法（k-NN）**と呼ぶ。訓練データを $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ とすると、分類手順は以下の通りである：

1. 入力点 $\boldsymbol{x}$ に対し、訓練データの中からユークリッド距離などの距離尺度に基づいて最も近い k 個の点を探索する。
2. それらの点に対応するラベル $\{y_{i_1}, \ldots, y_{i_k}\}$ の多数決により、$\boldsymbol{x}$ のラベルを決

定する。

$k = 1$ のときは **1-最近傍法**と呼ばれ、最も近い 1 点のラベルをそのまま割り当てる。

2.6.2　ベイズ識別器との性能比較

理論的には、ベイズ識別器（事後確率が最大のクラスを選ぶ識別器）が誤識別率の下限を与える。しかし、k-NN 法は次のような興味深い性質を持つ：

- 訓練データ数 $n \to \infty$, $k \to \infty$, $k/n \to 0$ の極限で、k-NN 法の誤識別率はベイズ識別器の誤識別率 R^* に対して

$$R_{\mathrm{kNN}} \to R^* \quad \text{（一致性 consistency）}$$

- 特に、1-最近傍法（$k = 1$）については次のクラシカルな不等式が知られている（Cover and Hart, 1967）：

$$R_{\mathrm{1NN}} \leq 2R^*(1 - R^*)$$

  この不等式は、1-NN 法が「ベイズ識別器の 2 倍以下の誤識別率」に抑えられることを保証する。

このように、k-NN 法は非常に単純な非パラメトリック手法でありながら、理論的にも優れた性質を持つ。

2.6.3　計算アルゴリズム上の工夫

k-NN 法では、予測時に毎回すべての訓練データとの距離を計算する必要があり、計算コストが高くなる（$O(n)$）。このため、以下のような高速化手法が考案されている：

- **kd-tree（k 次元木）**や **Ball-tree** による空間分割と近傍探索の効率化
- **近似最近傍探索（Approximate Nearest Neighbor, ANN）**：高速だが近似的な手法（例：LSH, HNSW）
- **データの次元削減**（例：PCA）によって距離計算の次元数を削減する

また、次元が高くなると距離尺度の有効性が低下する（次元の呪い）。このため、高次元データへの適用には特別な注意が必要である。

2.6.4　k-NN 法の特徴と注意点

- モデルの学習という概念はなく、すべての訓練データがそのまま予測に使われる。
- ハイパーパラメータ k の選択が性能に大きく影響する。
- データ数が多い場合には、計算・メモリのコストが高くなる。

シンプルで実装しやすく、かつ一定の理論的保証もあるため、分類タスクの初期ベースラインや比較対象として広く用いられている。

Program 2.7　kNN

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# データの読み込み
data = load_breast_cancer()
X, y = data.data, data.target

# 訓練・テストに分割（層化抽出）
X_train, X_test, y_train, y_test = train_test_split(
    X, y, stratify=y, test_size=0.2, random_state=42
)

```

```
18 # 特徴量の標準化
19 scaler = StandardScaler()
20 X_train_std = scaler.fit_transform(X_train)
21 X_test_std = scaler.transform(X_test)
22
23 # を変化させたときのテスト誤差を計算k
24 k_values = range(1, 31)
25 test_errors = []
26
27 for k in k_values:
28     model = KNeighborsClassifier(n_neighbors=k)
29     model.fit(X_train_std, y_train)
30     y_pred = model.predict(X_test_std)
31     test_error = 1 - accuracy_score(y_test, y_pred)
32     test_errors.append(test_error)
33
34 # 結果のプロット
35 plt.figure(figsize=(8, 5))
36 plt.plot(k_values, test_errors, marker='o')
37 plt.xlabel("Number of Neighbors (k)")
38 plt.ylabel("Test Error Rate")
39 plt.title("k-NN Classifier: Test Error vs. k (Breast
       Cancer Dataset)")
40 plt.grid(True)
41 plt.tight_layout()
42 plt.show()
```

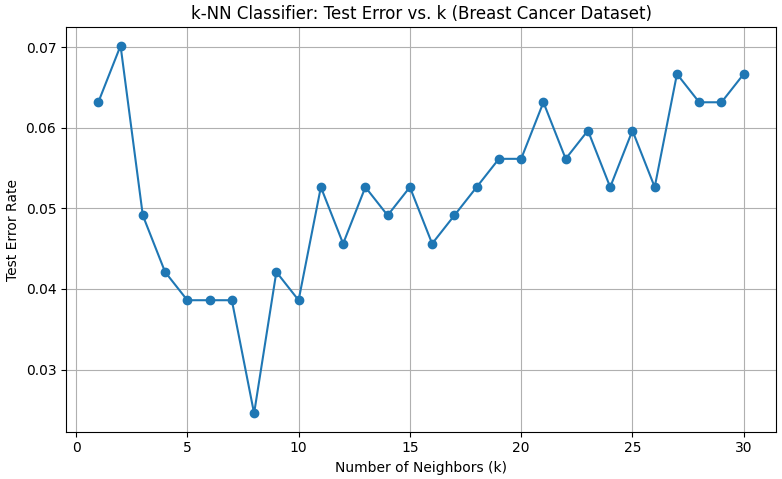

図 2.7　k-Nearest Neighbor 法

参考文献

[1] G. James, D. Witten, T. Hastie, R. Tibshirani “An Introduction to Statistical Learning, Second Edition”, Springer 2023

[2] C.M. Bishop, “Pattern recognition and machine learning”, Springer 2006 (ビショップ：パターン認識と機械学習（上下），丸善)

[3] 萩原，入門 統計的回帰とモデル選択，共立出版 2022

[4] 赤穂，カーネル多変量解析，岩波書店 2008

[5] 青嶋，矢田，高次元の統計学，共立出版 2019

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