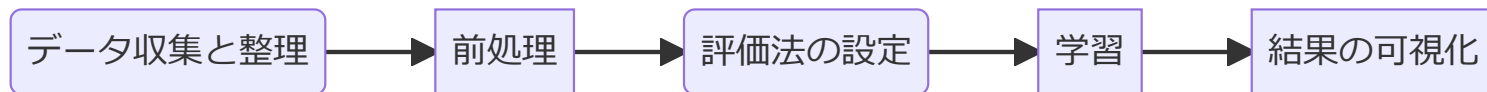


機械学習の基本的な手順

- 一般的な機械学習の手順
 - 前処理・学習・結果の可視化はライブラリによる支援が可能

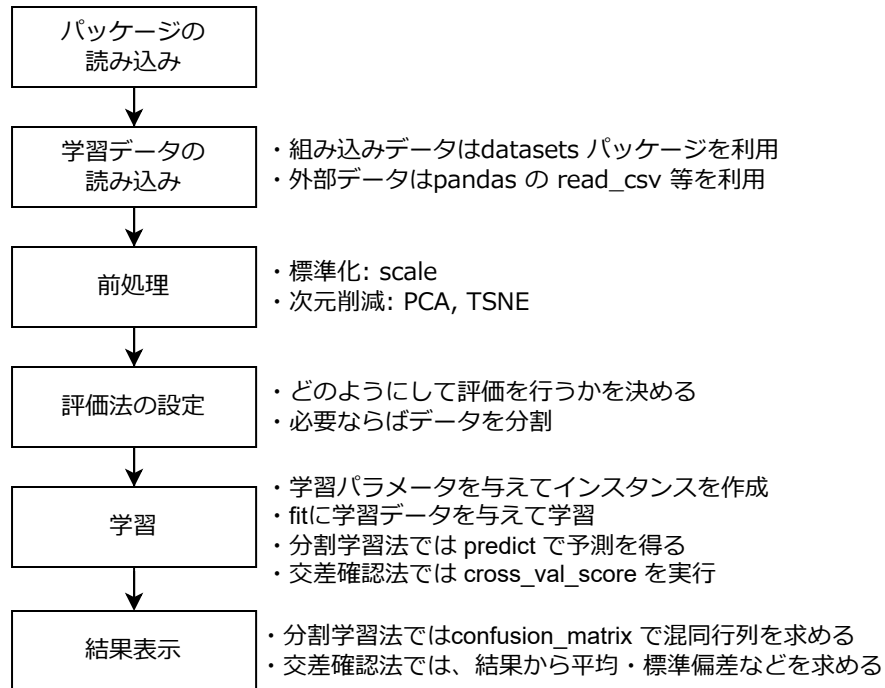


Pythonによる機械学習の実装

- Pythonを使うメリット
 - データ処理や機械学習のパッケージが充実
 - numpy : 多次元配列を効率よく扱う
 - scipy : 高度な数値計算
 - pandas : データの読み込み・解析を支援
 - scikit-learn : 多くの機械学習アルゴリズム
 - tensorflow : 深層学習
 - グラフ表示などの可視化が容易
 - matplotlib : グラフ描画
 - Jupyter Notebookで実行手順を記録しながらコーディングが可能

Pythonによる機械学習の実装

■ scikit-learnを用いた機械学習の手順



scikit-learnを用いた機械学習の手順

- パッケージの読み込み
 - 機械学習のコードにはnumpyは必須
 - 入力したデータの分析や前処理を行うにはpandasは必須
 - データや結果の可視化を行うにはmatplotlib.pyplotは必須
 - scikit-learnは大きなパッケージではなくクラスや関数を個別に指定

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report
```

データの読み込み

- サンプルデータ: iris
 - 3種類のアヤメ(setosa, versicolor, virginica)を萼(がく; sepal) の長さ・幅、花びら(petal)の長さ・幅の4つの特徴から分類する
 - 各クラス50事例ずつで計150事例のデータ数

index	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

データの読み込み

- scikit-learnでのデータの持ち方
 - パターン行列: X
 - 全データの特徴ベクトルを列方向に並べたもの
 - irisデータの場合は150事例、4特徴の150行4列の行列
 - 正解: y
 - 正解ラベルを整数値にしてデータの数だけ並べたもの
 - irisデータの場合は150個の数字(0,1,2のいずれか)が並ぶベクトル

データの読み込み

- numpyのndarray (n次元テンソル)として読み込む方法

```
iris = load_iris()  
X = iris.data  
y = iris.target  
print(iris.DESCR)
```

- load_iris関数の戻り値はBunchオブジェクト
 - Bunchオブジェクト
 - 特徴ベクトル, 正解データ, 特徴名, データの説明などのさまざまな情報を属性として持つ
 - Xやyはndarrayなので、scikit-learnの学習データとして用いることができる

データの読み込み

- pandasのDataFrameおよびSeriesとして読み込む方法

- データロード関数の引数: `as_frame=True`

```
iris = load_iris(as_frame=True)
```

- 実データでは、異常値・欠損値・記述ミス・不要な特徴の混入などへの対処が必要
→ numpyでは不十分

- pandas: データ分析・操作ライブラリ

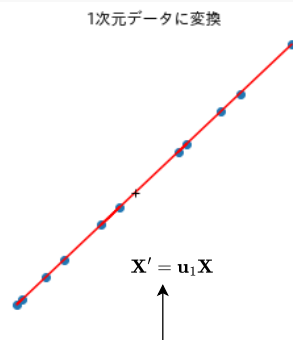
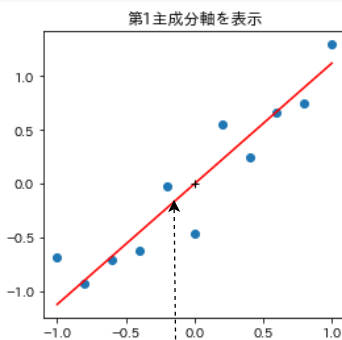
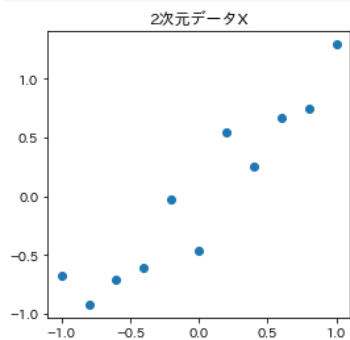
- 統計的分析: `describe`, `hist`, ...
- 異常値・欠損値(NA)処理: `query`, `dropna`, `fillna`, ...

前処理

主成分分析(PCA)

- 高次元空間上のデータの散らばりをできるだけ保存する低次元空間への写像を求める
- データの次元削減に有効

```
pca = PCA(n_components=1) # n_components: 削減後の次元数  
X2 = pca.fit_transform(X)
```



共分散行列 Σ を計算 \longrightarrow Σ を固有値分解
 $\Sigma = U\Lambda U^{-1}$

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \quad U = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix}$$

$\lambda_1 > \lambda_2$

前処理

- t-SNE
 - 高次元空間での類似度を反映した低次元空間への写像を求める
 - データの可視化に有効

```
tsne = TSNE(perplexity=5) # perplexity: 考慮する近傍のデータ数 (5~50程度の値で全データ数が多いほど大きく)  
X3 = tsne.fit_transform(X)
```

t-SNEの考え方

- 元の高次元空間
 - どのくらいの範囲のデータを類似度計算の対象とみなすかをパラメータ perplexity で与える
 - データ \mathbf{x}_i と \mathbf{x}_j の類似度を、 \mathbf{x}_i が与えられた時に近傍として \mathbf{x}_j を選択する条件付き確率 p_{ij} とする
 - p_{ij} : 平均を \mathbf{x}_i 、分散を perplexity に基づいて求めた σ^2 とする正規分布に基づいて計算
- 削減後の低次元空間
 - データ \mathbf{y}_i と \mathbf{y}_j の類似度 q_{ij} を、自由度1のt分布に基づいて計算
 - t分布は正規分布よりも値の大きい範囲が広い
- 最適化
 - p_{ij} と q_{ij} を近くするために、両分布間の距離(KL-divergence)を最小化するように $Y = \{\mathbf{y}_1, \dots, \mathbf{y}_n\}$ の位置を逐次更新

$$KL(P, Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

前処理

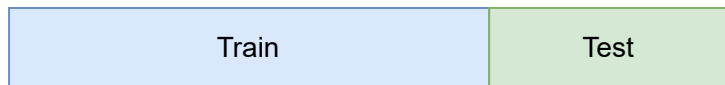
- 特徴のスケーリング
 - 特徴の各次元のスケールが著しく異なると、特徴の扱いが不公平になる
 - 標準化:すべての次元を平均0、分散1に揃える
 - 各次元(軸)に対して平均値を引き、標準偏差で割る

$$x'_i = \frac{x_i - m_i}{\sigma_i} \quad m_i, \sigma_i : \text{軸}i\text{の平均、標準偏差}$$

```
X_scaled = StandardScaler().fit_transform(X)
```

評価基準の設定と学習

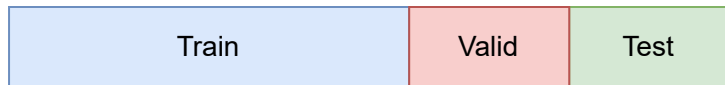
- 分割学習法(データ数が多いとき)
 - データを学習用と評価用に適切な割合で分ける



- 実験の再現性を確保するためにはrandom_stateを固定しておく

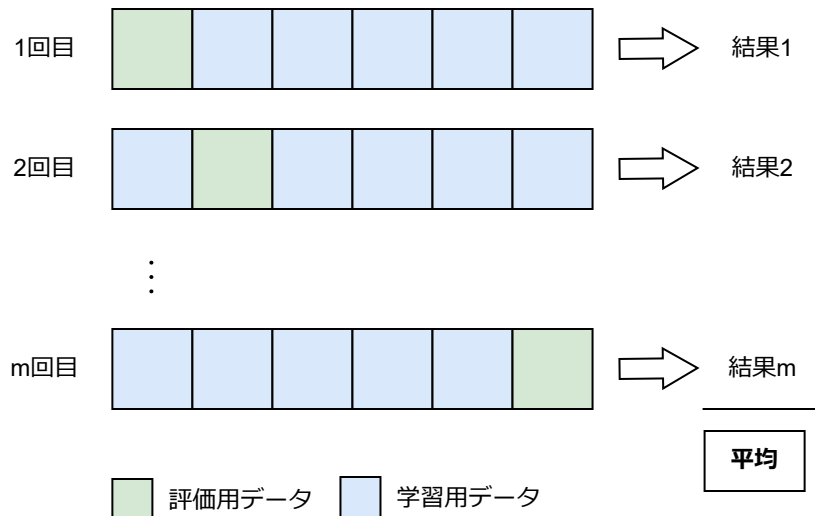
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.33, random_state=7)
```

- ハイパーパラメータを調整する場合は、学習用・検証用・評価用に分ける



評価基準の設定

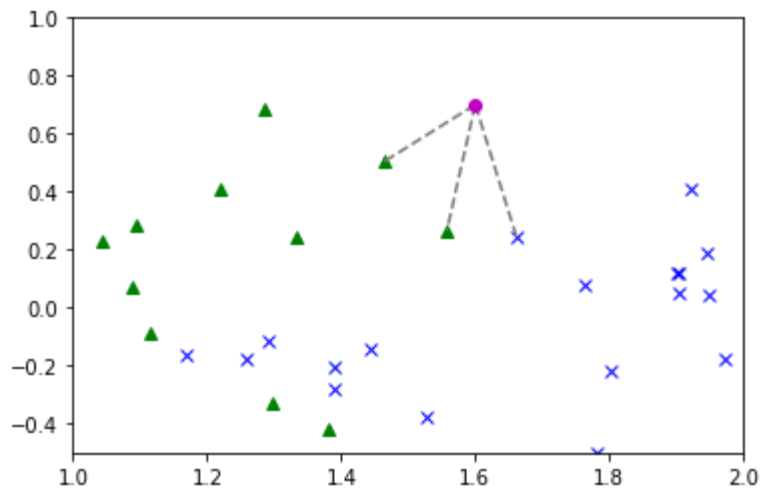
- 交差確認法(データ数が少ないとき)
 - データを m 個の集合に分割し、 $m - 1$ 個の集合で学習、残りの1個の集合で評価を行う
 - 評価する集合を入れ替え、合計 m 回評価を行う
 - 分割数をデータ数とする場合を一つ抜き法とよぶ
 - 学習用データで交差確認によりハイパーパラメータ調整を行い、評価用データで評価してもよい



評価基準の設定と学習

- k-NN法

- 識別したいデータの近傍のk個の学習データを探し、属するクラスの多数決で識別



評価基準の設定と学習

- k-NN法のパラメータ
 - 近傍として探索するデータ数: k
 - k が1の場合にもっとも複雑な境界
 - k が増えるに従って境界は滑らかになるが、大き過ぎると識別性能が低下する
 - 距離尺度
 - 通常はユークリッド距離
 - 値を持つ次元が少ない場合はマンハッタン距離
 - 探索方法
 - 入力と全データとの距離を計算してソート
 - データが多い場合は事前にデータを木構造化

評価基準の設定と学習

- 学習を行うインスタンスの生成

- モデルの構成に関するパラメータ(ハイパーパラメータ)は、インスタンス生成時に与える

```
clf = KNeighborsClassifier(n_neighbors=3)
```

- 詳しくはAPIドキュメントを参照

アルゴリズムの詳細な説明や、
事例が記載されているページ
へのリンク

インスタンス生成時に
指定するパラメータの
説明

sklearn.neighbors.KNeighborsClassifier

```
class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None)
```

Classifier implementing the k-nearest neighbors vote.
Read more in the [User Guide](#).

Parameters:	n_neighbors : int, default=5 Number of neighbors to use by default for <code>kneighbors</code> queries.
	weights : {'uniform', 'distance'} or callable, default='uniform' Weight function used in prediction. Possible values:

デフォルト引数の値が示されている。
*以降は必ずキーワード
引数で指定する。

結果の表示

- 学習したモデル
 - 式、木構造、ネットワークの重み、etc.
- 性能
 - 正解率、適合率、再現率、F値
 - グラフ
 - パラメータを変えたときの性能の変化
 - 異なるモデルの性能比較

結果の表示

- 混同行列

	予測+	予測-
正解+	true positive (TP)	false negative (FN)
正解-	false positive (FP)	true negative (TN)

- 正解率

- 正解の割合。unbalanced dataには不適

$$Accuracy = \frac{TP+TN}{TP+FN+FP+TN}$$

- 適合率

- 正例の判定が正しい割合

$$Precision = \frac{TP}{TP+FP}$$

- 再現率

- 正しく判定された正例の割合

$$Recall = \frac{TP}{TP+FN}$$

- F値

- 適合率と再現率の調和平均

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

結果の表示

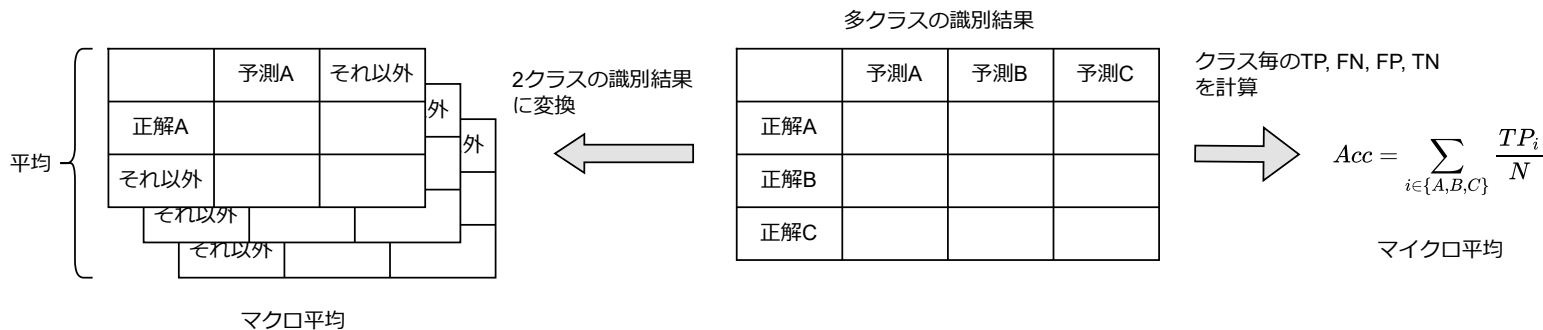
■ 多クラス識別の評価法

○ マクロ平均

- ひとつのクラスを正、残りのクラスを負とした混同行列を作成し、クラスごとの適合率や再現率を求め、平均を計算する
- すべてのクラスを平等に評価している

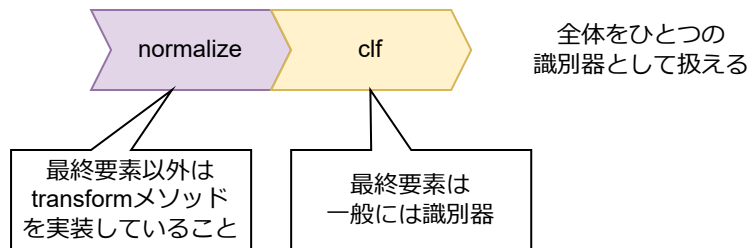
○ マイクロ平均

- クラスごとにTP, FN, FP, TNを求め、それらを足し合わせて適合率や再現率を求める
- クラス毎の事例数を評価に反映させている



パイプライン

- パイプラインとは
 - 複数の前処理と学習モジュールなど、連続した処理ををパイプラインとして結合して、ひとつの識別器のインスタンスとみなせる
- パイプラインのメリット
 - 処理をカプセル化して実行を簡単にできる
 - ハイパーパラメータ調整を一度に行える
 - テストデータが混入していないことを保証できる



まとめ

- 機械学習の基本的な手順
 - 前処理
 - 標準化、次元削減
 - 評価基準の設定
 - 分割法、交差確認法
 - 学習
 - 結果の可視化

実開発ではこれ以前のデータ収集・整理の段階が最も時間がかかることも多い