**データベース処理とデータ分析**

**９． Pythonによるデータ分析：データフレーム、Irisデータセットの可視化、メッシュグリッドと混合ガウス分布**

URL: http://www.kkaneko.jp/cc/dbenshu/index.html

9.1 データ構造の基礎

9.1.1 データフレーム (DataFrame)

Pythonでは表形式のデータを「データフレーム」と呼ぶ．同様のデータ構造は，以下のように各システムで異なる名称で呼ばれている：

リレーショナルデータベース：テーブル (Table)

Microsoft Excel：ワークシート (Worksheet)

Python：データフレーム (DataFrame)

9.1.2 pandasデータフレームを使用した散布図の作成

pandas のデータフレームを使用して2次元平面上の5つの点を散布図として描画する．まずデータフレームを作成し，x列とy列にそれぞれ5つの数値データを追加する．x列には0から4までの連続した整数，y列には4から7までの値を格納する．matplotlibライブラリのpyplotモジュールを用いて，データフレームの2つの列を指定して散布図を作成し表示する．

import pandas as pd

import matplotlib.pyplot as plt

# データフレームを作成

A = pd.DataFrame()

A['x'] = [0, 1, 2, 3, 4]

A['y'] = [5, 7, 5, 6, 4]

# scatter()関数でデータフレームの2列を指定して散布図をプロット

plt.scatter(A['x'], A['y'])

plt.show()

9.1.3 Irisデータセットを使用した散布図の作成

scikit-learnライブラリに含まれるIrisデータセットを使用する．このデータセットは機械学習の基礎研究でよく使用される標準データセットであり，アヤメの花の特徴量（がく片と花弁の長さ・幅）と品種の情報が含まれている．pandasのデータフレームの列には，数値や文字列による名前（列名）を付与できる．例えば，Irisデータセットでは，sepal\_length，sepal\_width，petal\_length，petal\_width，speciesの5つの列名が定義されている． このプログラムは，scikit-learnライブラリからIrisデータセットを読み込み，データフレームとして整形した後，がく片の特徴を散布図として可視化する．まず，load\_iris関数でデータセットを取得し，データフレームを作成する．次に，品種情報をカテゴリカルデータとしてspecies列に追加する．がく片の長さ（sepal length）とがく片の幅（sepal width）を用いて散布図を作成し表示する．

#

from sklearn.datasets import load\_iris

import pandas as pd

import matplotlib.pyplot as plt

# scikit-learnからIrisデータセットを読み込み

iris = load\_iris()

# データフレームを作成

df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

df['species'] = pd.Categorical.from\_codes(iris.target, iris.target\_names)

# がく片の長さと幅を使用して散布図をプロット

plt.scatter(df['sepal length (cm)'], df['sepal width (cm)'])

plt.show()

9.1.4 Irisデータセットを使用した品種別の散布図作成

Irisデータセットのがく片の特徴を品種別に色分けして散布図として表示する．がく片の長さ（sepal length）とがく片の幅（sepal width）を用いて散布図を作成する．この際，iris.targetに含まれる品種のインデックス情報を用いて点の色分けを行い，グラフを表示する．

from sklearn.datasets import load\_iris

import pandas as pd

import matplotlib.pyplot as plt

# scikit-learnからIrisデータセットを読み込み

iris = load\_iris()

# データフレームを作成

df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

df['species'] = pd.Categorical.from\_codes(iris.target, iris.target\_names)

# 品種別に色分けした散布図をプロット（iris.targetで色分け）

plt.scatter(df['sepal length (cm)'], df['sepal width (cm)'], c=iris.target)

# 凡例を追加（品種名を表示）

plt.colorbar(label='Species')

plt.show()

9.1.5 主成分分析による次元削減

Irisデータセットに主成分分析（PCA）を適用して4次元の特徴量を2次元に削減し，その結果を散布図として可視化する．まず，scikit-learnのPCAクラスを使用して2つの主成分を抽出する．4つの特徴量（がく片と花弁の長さ・幅）から成る元データに対してPCAを実行し，2次元の座標データを得る．次に，第1主成分と第2主成分を軸とする散布図を作成する．Set1カラーマップを使用して品種別の色分けを行う．品種名の凡例を追加してグラフを表示する．

from sklearn.datasets import load\_iris

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

# scikit-learnからIrisデータセットを読み込み

iris = load\_iris()

# データフレームを作成

df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

df['species'] = pd.Categorical.from\_codes(iris.target, iris.target\_names)

# PCAの実行（4次元から2次元に削減）

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(iris.data)

# PCA結果の散布図をプロット

scatter = plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=iris.target)

# 品種名の凡例を追加

plt.legend(handles=scatter.legend\_elements()[0],

 labels=iris.target\_names,

 title='品種')

plt.show()

9.1.5 Irisデータセットに対するPCAと可視化

Irisデータセットに主成分分析（PCA）を適用して4次元の特徴量を2次元に削減し，その結果を散布図として可視化する．グラフ領域を確保し，がく片の長さと幅を用いて散布図を作成する．この際，Set1カラーマップを使用して品種別の色分けを行う．さらに，各データポイントの近くに品種名を表示し，軸ラベルとして「がく片長 (cm)」と「がく片幅 (cm)」を，タイトルとして「Irisデータセット：がく片の特徴量の散布図（品種名付き）」を設定する．最後に，品種名の凡例を追加する．

from sklearn.datasets import load\_iris

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

# scikit-learnからIrisデータセットを読み込み

iris = load\_iris()

# データフレームを作成

df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

df['species'] = pd.Categorical.from\_codes(iris.target, iris.target\_names)

# PCAの実行（4次元から2次元に削減）

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(iris.data)

# グラフ領域のサイズを設定

plt.figure(figsize=(10, 6))

# PCA結果の散布図をプロット（Set1カラーマップを使用）

scatter = plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=iris.target, cmap='Set1')

# 各データポイントに品種名を表示

for i in range(len(df)):

 plt.annotate(df['species'][i],

 (X\_pca[i, 0], X\_pca[i, 1]), # インデックスの修正

 xytext=(5, 5), # テキストのオフセット

 textcoords='offset points', # オフセットの単位

 fontsize=8) # フォントサイズ

# グラフのX軸とY軸にラベルを追加（日本語）

plt.xlabel('第１主成分')

plt.ylabel('第２主成分')

# グラフのタイトルを追加（日本語）

plt.title('Irisデータセット：PCAによる2次元可視化（品種名付き）')

# 品種名の凡例を追加

plt.legend(handles=scatter.legend\_elements()[0],

 labels=iris.target\_names,

 title='品種')

# グリッド線を表示

plt.grid(True)

# グラフを表示

plt.show()

9.2 メッシュグリッドと応用

9.2.1 numpy.arange関数による数値列の生成

Pythonでは，区間[s, t)の数値列（sで始まり，tの値を超えない数値列）は，numpy.arangeを用いて生成できる．

numpy.arange関数を使用して2種類の数値列を生成する．まず，変数aには開始値0，終了値10，刻み幅2の数値列を生成して格納する．これにより，0から始まり10未満までの偶数列（0, 2, 4, 6, 8）が得られる．次に，変数bには開始値0，終了値10，刻み幅1の数値列を生成して格納する．これにより，0から始まり10未満までの連続した整数列（0, 1, 2, 3, 4, 5, 6, 7, 8, 9）が得られる．

import numpy as np

# 開始値0，終了値10，刻み幅2の数値列を生成（偶数列）

a = np.arange(0, 10, 2) # [0, 2, 4, 6, 8]

print(a)

# 開始値0，終了値10，刻み幅1の数値列を生成（連続整数列）

b = np.arange(0, 10, 1) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(b)

9.2.2 メッシュグリッドの生成

メッシュグリッドは，2次元平面上の格子点を表現するデータ構造である．

numpy.arange関数を用いて格子点のx座標とy座標を表す数値列を生成し，それらからメッシュグリッドを作成する．まず，変数xxとyyに-1から1未満までの刻み幅0.1の数値列を生成する．次に，numpy.meshgrid関数を用いてこれらの数値列から2つの2次元配列pxとpyを生成する．pxは各列が同じx座標値を持ち，pyは各行が同じy座標値を持つ．

import numpy as np

# x方向とy方向の数値列を生成（-1から1未満まで，刻み幅0.1）

xx = np.arange(-1, 1, 0.1) # x方向の座標値の配列

print("x方向の数値列:")

print(xx)

yy = np.arange(-1, 1, 0.1) # y方向の座標値の配列

print("\ny方向の数値列:")

print(yy)

# meshgrid関数で2次元配列を生成

px, py = np.meshgrid(xx, yy) # メッシュグリッドの生成

# x座標の2次元配列を表示（各列が同じx座標値）

print("\nx座標の2次元配列:")

print(px)

# y座標の2次元配列を表示（各行が同じy座標値）

print("\ny座標の2次元配列:")

print(py)

9.2.3 メッシュグリッドを用いた2次元関数の可視化

メッシュグリッドは2次元関数の可視化に有用である．

メッシュグリッドを用いた f(x, y) = x² + y² の表示プログラム：

import numpy as np

import matplotlib.pyplot as plt

# 2次元関数f(x,y)の定義

def f(x, y):

 return x \* x + y \* y

# メッシュグリッドの生成（-1から1未満まで，刻み幅0.1）

xx = np.arange(-1, 1, 0.1)

yy = np.arange(-1, 1, 0.1)

px, py = np.meshgrid(xx, yy)

# グラフ領域を2分割して描画

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1) # カラーマップ表示

plt.pcolor(px, py, f(px, py))

plt.colorbar()

plt.subplot(1, 2, 2) # 等高線表示

plt.contour(px, py, f(px, py))

plt.show()

9.2.5 PCAで2次元に削減したIrisデータに対する混合ガウス分布

PCAで2次元に削減したIrisデータに対して混合ガウス分布（GMM）をフィットさせ，その確率密度関数w表示する．まず， GaussianMixtureモデルを作成し，Irisデータセット全体を用いて学習を行う．次に，-4から4までの範囲で細メッシュグリッドを生成し，各格子点でGMMの確率密度を計算する．可視化では，12x5インチの図領域に2つのサブプロットを作成する．左側のサブプロットでは，確率密度をカラーマップとして表示し，その上にPCA変換後のデータ点を品種別に色分けして散布図として重ねる．右側のサブプロットでは，確率密度の等高線を表示し，同様にデータ点を重ねる．両方のサブプロットにカラーバーを追加し，確率密度の大きさを示す．

import numpy as np

from sklearn.mixture import GaussianMixture

from sklearn.decomposition import PCA

from sklearn.datasets import load\_iris

import matplotlib.pyplot as plt

# scikit-learnからIrisデータセットを読み込み

iris = load\_iris()

# PCAの実行（4次元から2次元に削減）

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(iris.data)

# 混合ガウス分布モデルの学習

gmm = GaussianMixture(n\_components=3, random\_state=42)

gmm.fit(X\_pca)

# メッシュグリッドの作成と確率密度の計算

x, y = np.mgrid[-4:4:.01, -4:4:.01]

pos = np.dstack((x, y))

z = np.exp(gmm.score\_samples(pos.reshape(-1, 2))).reshape(x.shape)

# グラフ領域を2分割して描画

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1) # 確率密度のカラーマップと散布図

plt.contourf(x, y, z, levels=20, cmap='viridis')

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=iris.target, cmap='Set1', alpha=0.5)

plt.colorbar()

plt.subplot(1, 2, 2) # 確率密度の等高線と散布図

plt.contour(x, y, z, levels=20, cmap='viridis')

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=iris.target, cmap='Set1', alpha=0.5)

plt.colorbar()

plt.show()

9.3 クラスタリング分析

9.3.1 Irisデータセットに対するk-meansクラスタリング

Irisデータセットの全特徴量に対してk-meansクラスタリングを適用し，その結果を表示する．まず，scikit-learnのKMeansクラスを使用して，クラスタ数を3に設定したクラスタリングを実行する．表示では，データ点をクラスタ別に色分けした散布図を作成する．

from sklearn.cluster import KMeans

from sklearn.datasets import load\_iris

import pandas as pd

import matplotlib.pyplot as plt

# データの準備

iris = load\_iris()

X = iris.data # 全特徴量を使用

# k-meansクラスタリングの実行

kmeans = KMeans(n\_clusters=3, random\_state=42)

clusters = kmeans.fit\_predict(X)

# グラフの作成とクラスタリング結果の可視化

plt.scatter(X[:, 0], X[:, 1], c=clusters, cmap='viridis')

# クラスタ中心の表示

plt.scatter(kmeans.cluster\_centers\_[:, 0],

 kmeans.cluster\_centers\_[:, 1],

 c='red',

 marker='x',

 s=200,

 linewidths=3,

 label='クラスタ中心')

# グラフの設定と表示

plt.title('Irisデータセット：k-meansクラスタリング結果（次元削減後）')

plt.legend()

plt.show()