PCA と DBSCAN による時系列クラスタリング

scikit-learn* 向けインテル® エクステンションを利用したアルゴリズムの高速化

Bob Chesebrough インテル コーポレーション シニア AI ソリューション・アーキテクト

この記事では、次元縮小に主成分分析（PCA）、クラスタリングにノイズを含むアプリケーションの密度ベースの空間クラスタリング（DBSCAN）を用いて、時系列データをクラスタリングする方法について考察します。この手法は、ラベル付きデータなしで、都市の交通量などの時系列データ内のパターンを識別します。パフォーマンス向上のため、scikit-learn* 向けインテル® エクステンション（英語）を使用します。

時系列データには、人間の行動、機械、その他の測定可能なソースによる反復パターンがしばしば見られます。このようなパターンを手作業で識別することは困難です。PCA や DBSCAN などの教師なし学習アプローチは、これらのパターンの発見を可能にします。

手法

データ生成

時系列パターンをシミュレーションするため、合成波形データを生成します。データは 3 つの異なる波形で構成され、それぞれにノイズを追加することで、実世界の変動をシミュレーションします。Gaël Varoquaux 氏が作成した scikit-learn の凝集型クラスタリングの例を使用します（**図 1**）。これは、BSD-3-Clause（英語）または CC0（英語）ライセンスの下で利用可能です。

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)
n_features = 2000
t = np.pi * np.linspace(0, 1, n_features)

def sqr(x):
    return np.sign(np.cos(x))

X = []
y = []
for i, (phi, a) in enumerate([(0.5, 0.15), (0.5, 0.6), (0.3, 0.2)]):
    for _ in range(30):
        phase_noise = 0.01 * np.random.normal()
        amplitude_noise = 0.04 * np.random.normal()
        additional_noise = 1 - 2 * np.random.rand(n_features)
        additional_noise[np.abs(additional_noise) < 0.997] = 0
        X.append(12 * ((a + amplitude_noise)

                       * (sqr(6 * (t + phi + phase_noise)))

                       + additional_noise))
        y.append(i)

X = np.array(X)
y = np.array(y)

plt.figure()
plt.axes([0, 0, 1, 1])
for l in range(3):
    plt.plot(X[y == l].T, alpha=0.5, label=f'Waveform {l+1}')
plt.legend(loc='best')
plt.title('Unlabeled Data')
plt.show()
```

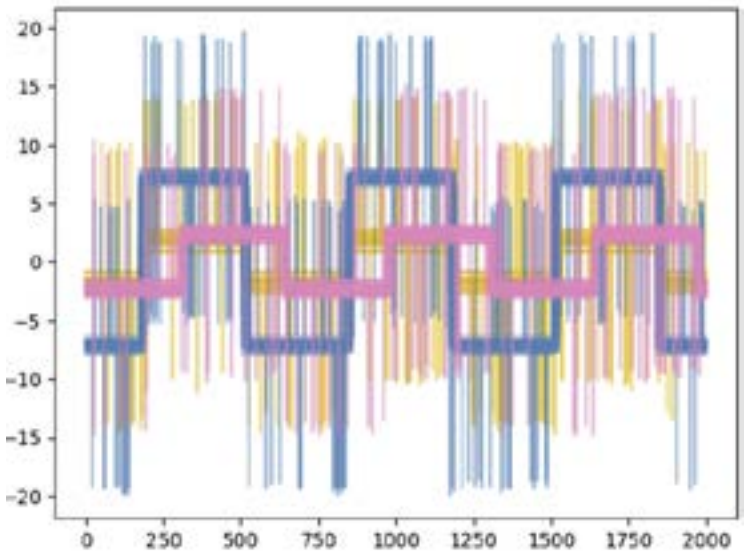

図 1. Gaël Varoquaux 氏が作成した scikit-learn 凝集型クラスタリング・アルゴリズムから著者が生成したコードとプロット

scikit-learn* 向けインテル® エクステンションを利用したアルゴリズムの高速化

PCA と DBSCAN はどちらも、scikit-learn* 向けインテル® エクステンションを用いたパッチ適用スキームによって高速化できます。scikit-learn（sklearn とも呼ばれる）は、マシンラーニング（ML）用の Python モジュールです。scikit-learn* 向けインテル® エクステンションは、シングルノード構成およびマルチノード構成のインテルの CPU および GPU 上で scikit-learn アプリケーションをシームレスに高速化するインテルの AI ツール（英語）の 1 つです。scikit-learn 推定器に動的にパッチを適用することで、ML のトレーニングと推論を同等の数学的精度で最大 100 倍向上させます（**図 2**）。

図 2. scikit-learn* 向けインテル® エクステンションの GitHub リポジトリー（英語）

scikit-learn* 向けインテル® エクステンションは scikit-learn API を使用し、scikit-learn をインポートする前に Python アプリケーションを数行変更することで有効にできます。

```
from sklearnex import patch_sklearn
patch_sklearn()
```

PCA を用いた次元縮小

2,000 個の特徴を含む 90 個のサンプルをクラスタリングする前に、データセットの分散の 99% を維持しながら PCA を用いて次元を縮小します。

```
from sklearn.decomposition import PCA

pca = PCA(n_components=4)
XPC = pca.fit_transform(X)

print("Explained variance ratio:", pca.explained_variance_ratio_)
print("Singular values:", pca.singular_values_)
print("Shape of XPC:", XPC.shape)
```

ペアプロットを使用して、縮小されたデータからクラスターを探します（**図 3**）。

```
import pandas as pd
import seaborn as sns

df = pd.DataFrame(XPC, columns=['PC1', 'PC2', 'PC3', 'PC4'])
sns.pairplot(df)
plt.show()
```

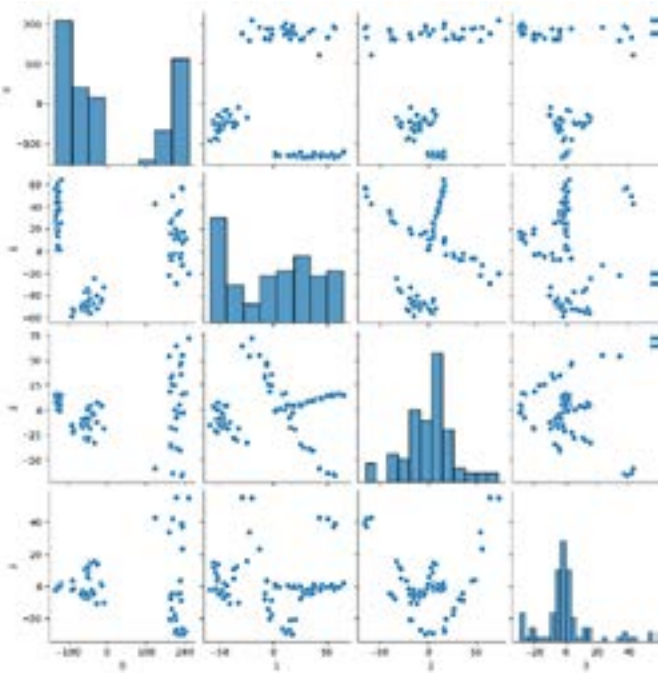

図 3. 次元縮小後のデータからクラスターを見つける

DBSCAN を用いたクラスタリング

ペアプロットから、PC1 と PC2 はクラスターを適切に分離しているため、これらの要素を DBSCAN クラスタリングに使用します。DBSCAN EPS パラメーターの値も推定できます。PC1 と PC0 の図から、観察されたクラスターでは 50 が妥当な分離距離であることが分かります。

```
from sklearn.cluster import DBSCAN

clustering = DBSCAN(eps=50, min_samples=3).fit(XPC[:, [0, 1]])
labels = clustering.labels_

print("Cluster labels:", labels)
```

クラスター化されたデータをプロットすると、DBSCAN がクラスターをどの程度正確に識別したかを確認できます (**図 4**)。

```
plt.figure()
plt.axes([0, 0, 1, 1])
colors = [ "#f7bd01" , "#377eb8" , "#f781bf" ]
for l, color in zip(range(3), colors):
    plt.plot(X[labels == l].T, c=color, alpha=0.5, label=f' Cluster {l+1}' )
plt.legend(loc=' best' )
plt.title( 'PCA + DBSCAN' )
plt.show()
```

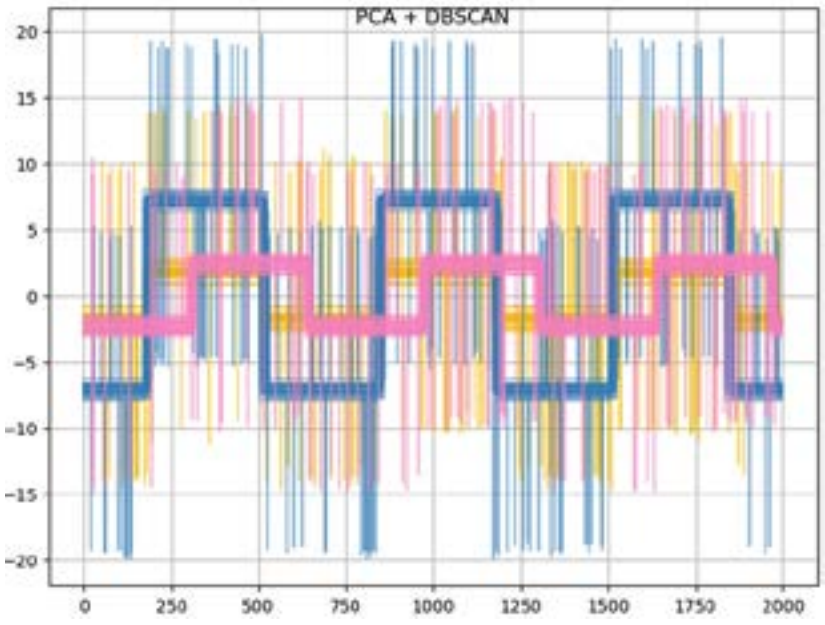

図 4. 前のコード例を使用して生成されたクラスター化データのプロット

元のデータとの比較

図 4 から分かるように、DBSCAN は妥当な色のクラスターを検出し、元のデータ（**図 1**）と一致しています。このケースでは、クラスタリングによってデータ生成に使用された基本パターンが完全に復元されました。次元縮小に PCA を使用し、クラスタリングに DBSCAN を使用することで、時系列データ内のパターンを効果的に識別し、ラベル付けすることができます。このアプローチにより、ラベル付きサンプルがなくても、データの基本構造を発見することができます。