

データ分析（2）

《学修項目》

- クラスター分析
- 次元削減
- パターン発見（アソシエーション分析）

《キーワード》

階層的クラスター分析、非階層的クラスター分析、重心法、群平均法、ウォード法、樹形図（デンドログラム）、k-means法、主成分分析、行列の固有値分解、カーネル法、ガウスカーネル、相関ルール、バスケット分析、支持度、確信度、リフト値、Aprioriアルゴリズム

《参考文献、参考書籍》

- [1] 東京大学MIセンター公開教材「1-4 データ分析」《利用条件CC BY-NC-SA》
- [2] 応用基礎としてのデータサイエンス（講談社 データサイエンス入門シリーズ）
- [3] データサイエンスの考え方 社会に役立つAI×データ活用のために（オーム社）
- [4] Pythonによるあたらしいデータ分析の教科書 第2版（翔泳社）
- [5] 数理・データサイエンス・AI公開講座（放送大学）

1. クラスター分析

1.1 概要 [1][2]

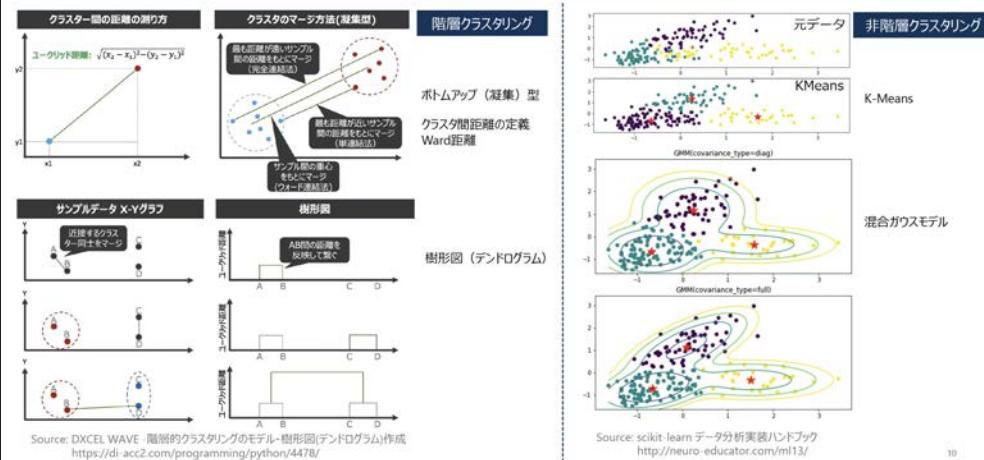
企業は、市場の見込み客を特定することができれば効率的なマーケティング戦略が可能になる。クライアントの属性情報や、利用履歴の情報を用いて、顧客を似た者どうしのグループに細分化し、グループごとの注文・消費行動などの特徴を調べ、効果的な見込み客の特定を行うことがある。このようなグループ化をセグメンテーション(segmentation)という。セグメンテーションを行う際に用いられる分析手法がクラスター分析（cluster analysis）である[2]。

クラスター分析は、説明変数の情報を利用せず、データを似たもの同士に分類する（教師なし学習）。どんなデータを似ていると考えるかは、2点間の距離やクラスター間の距離（類似度）の定義による。距離の定義によって結果が異なることがあることに注意する。

代表的なクラスター分析の方法として、階層的クラスター分析と非階層的クラスター分析がある[1]。

クラスタリング (clustering)

クラスタリングは、機械学習の中の「離散変数/教師なし学習」の一つである。正解ラベルが無く、データから隠れた関係性や構造を見つけるアルゴリズムである。



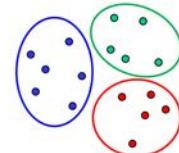
階層クラスタリング、非階層クラスタリング

クラスターとは

k 次元のベクトルで表されるデータを考えます。

$$x_n = (x_{n1}, \dots, x_{nk})$$

例えば、 n 番目の学生の数学、物理、生物、…、英語の成績などです。学生が N 人いると N 個のベクトル x_1, \dots, x_N が得られます。



k 次元空間の似た点の集まりを **クラスター** と呼びます。クラスター分析では、データから図の様に近いものを一つのクラスターにして、いくつかのグループに分けます。ただし、いくつのクラスターが存在するか前もってわかっているわけではないので、2点間の近さやクラスター間の近さをどのように定義するかによって結果が変わってきます。

1.2 階層的クラスター分析[1]

1.2.1 距離の定義

例えば平面上の2点 $P(x_1, y_1), Q(x_2, y_2)$ に対して、様々な距離が定義される。通常使われるユーリッド距離である。マンハッタン距離は通常、 p が1または2の場合が用いられ、これはそれぞれマンハッタン距離とユーリッド距離に対応する。特殊な場合であるが、 p が無限に発散する場合はチェビシェフ距離が得られる。

- ユーリッド距離 $d(P, Q) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$
- マンハッタン距離 $d(P, Q) = |x_1 - y_1| + |x_2 - y_2|$
- 次数 p のマンハッタン距離 $d(P, Q) = (|x_1 - y_1|^p + |x_2 - y_2|^p)^{\frac{1}{p}}$
- チェビシェフ距離 $d(P, Q) = \max(|x_1 - y_1|, |x_2 - y_2|)$

1.2.2 クラスター間の距離（類似度）

階層クラスタリングを行う場合、サンプル同士が何をもって「似ている（もしくは似ていない）」と判断するかによって計算方法が異なる。クラスター間の距離（類似度）には、最短距離法、最長距離法、重心法、群平均法、ウォード法など様々な類似度が提案されている。

- **最短距離法（単連結法）**：2つのクラスタ間で一番近いデータ同士の距離を、クラスタ間の距離として採用する
- **最長距離法（完全連結法）**：最短距離法とは逆の手法。クラスタを構成する要素同士のすべての距離の中で最長のものを、クラスタ間の距離として採用する
- **重心法**：各クラスタを構成する要素の重心位置を求め、その重心間の距離を、クラスタ間の距離として採用する
- **群平均法**：2つのクラスタを構成するデータのすべての組み合わせの距離を求め、その平均をクラスタ間の距離とする
- **ウォード法**：それぞれのデータの平方和（それぞれのデータと平均値の差を二乗した値の和）を求め、平方和が小さなものからクラスタを作っていく手法（i.e. 平方和が小さいほどデータのばらつきは小さい）

クラスター間の距離

二点間の距離としては通常、次に示すユークリッド距離を使います。

$$x_n = (x_{n1}, \dots, x_{nk}), \quad y_n = (y_{n1}, \dots, y_{nk})$$
$$d(x, y) = \sqrt{(x_1 - y_1)^2 + \dots + (x_k - y_k)^2}$$

・ただし、距離の定義としては、このほかマンハッタン距離、ミンコフスキーリー距離、チェビシェフ距離などがあります。

クラスター間の距離（類似度）

クラスター間の距離の定義としては、最短距離法、最長距離法、重心法、群平均法、ウォード法などいろいろ提案されており、どれを選ぶかによってクラスター分析の結果が異なります。

The diagram shows two sets of data points represented by colored ovals.
 1. **最短距離法 (Shortest Distance):** Shows a horizontal arrow pointing from the left cluster to the right cluster, indicating the distance between the closest points in each cluster.
 2. **最長距離法 (Longest Distance):** Shows a horizontal arrow pointing from the left cluster to the right cluster, indicating the distance between the farthest points in each cluster.
 3. **重心法 (Centroid Method):** Shows arrows from the center point (centroid) of each cluster to the center point of the other cluster.
 4. **群平均法 (Average Method):** Shows arrows from every point in one cluster to every point in the other cluster.
 5. **ウォード法 (Ward's Method):** Shows arrows from every point in one cluster to every point in the other cluster, with a formula for the linkage coefficient: $\times \frac{n_1 n_2}{n_1 + n_2}$.

1.2.3 階層型クラスタリング（デンドログラム）

階層的クラスタリングでは、最初は全てのデータが異なるクラスターに属するものとし、 $(\text{クラスターの数}) = (\text{データ数})$ とする状態を考える。

次にすべてのクラスター間で最も距離が小さな2つを探し、それらを1つのクラスターに統合し、クラスター間の距離を更新する。以下、このステップを繰り返すことによって、ボトムアップ的にひとつずつクラスター数を減らしていく。

このクラスター統合のときに、U型にふたつのクラスターを繋ぎ、縦の高さをクラスター間の距離とする（重要）。この操作を続けていくと、最後にクラスターが全体でひとつの木

構造をなすことになる。 クラスター間の関連度も含めて表現した木のことを、**樹形図（デンドログラム）** と言う。

階層的クラスタリング

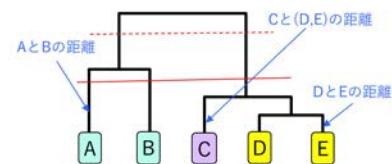
階層的クラスタリングでは、最初は全てのデータが異なるクラスターに属するものとし、(クラスターの数) = (データ数)とする状態を考えます。

次にすべてのクラスター間で最も距離が小さな2つを探し、それらを一つのクラスターに統合し、クラスター間の距離を更新します。以下、このステップを繰り返すことによってボトムアップにひとつずつクラスター数を減らしていきます。

このクラスター統合のときに△型にふたつのクラスターを繋ぎ、縦の高さをクラスター間の距離とします。この操作を続けていくと、最終的にクラスターがひとつになると、**樹形図（デンドログラム）** ができます。

既に樹形図が与えられている場合には、横線を引いて上からだんだん下げていくと2クラスター、3クラスターの順に分類ができます。

右図の場合は2クラスターの場合は、{A,B}, {C,D,E}、3クラスターの場合は、{A}, {B}, {C,D,E}であることがわかります。



統計ソフトウェア R の USArrests データには、米国50州における犯罪の人口10万人当たり件数が整理されている。このデータを犯罪種別ごとに標準化して階層的クラスター分析によって作成したデンドログラムを横向きに表示したものが図1.4.13である。このデンドログラムのある高さで4つのクラスター群を定義し、4色で色分けしている。

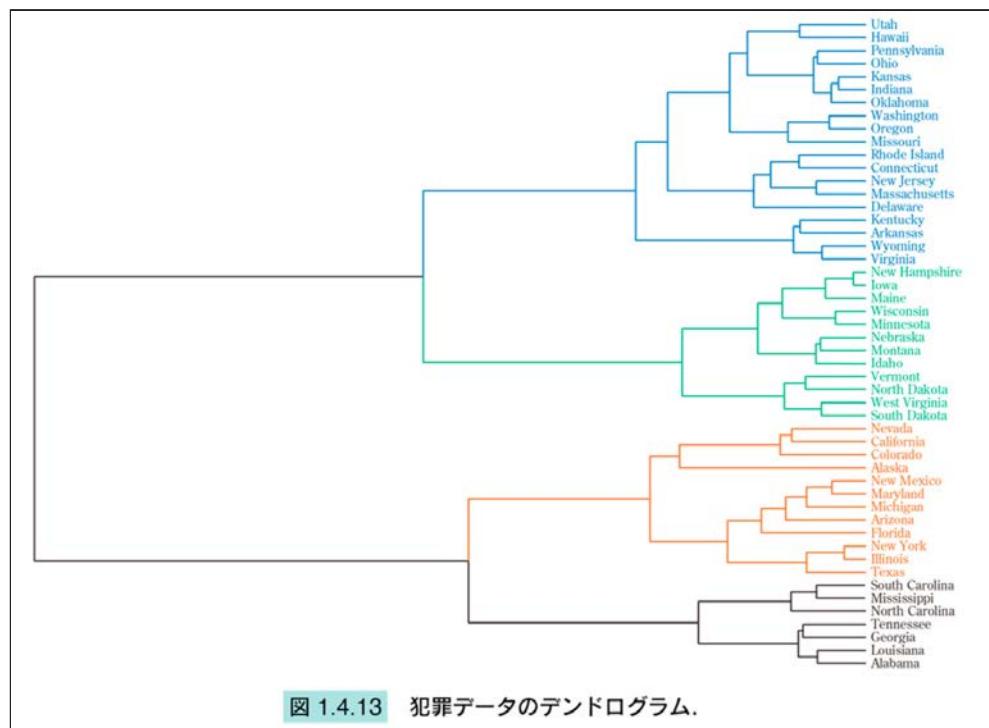


図 1.4.13 犯罪データのデンドログラム。

USArrests 犯罪データ(R)のデンドログラム例 ([2]より引用)

クラスターができたら、クラスターごとに各変数の代表値や標準偏差などを計算し各クラスターの特徴を把握することが重要である。このデータの場合、オレンジと黒のクラスター群は全体的に犯罪件数が多い。緑や青のクラスターは相対的には犯罪件数が少ないなどがわかる。

- 一般社団法人データマーケティングラボラトリー | クラスタ分析

Pythonによる階層クラスター分析 デンドログラムの実装例

階層クラスター分析の具体的な計算方法についてPythonのSciPyライブラリを用いて解説する。データは、「社会人対象：副業の有無(1/0)・収入・性別(1/0)・年齢・結婚有無(1/0)」のダミーデータ (n=21)である（量的変数、質的変数混在であることに注意）。

- CSVデータ
- Stata | 階層クラスター分析

```
In [ ]: # CSVデータを カレントディレクトリ直下のフォルダ(一時作業領域)へダウンロードする。
!wget -nc https://raw.githubusercontent.com/MDASH-shinshu/MDASH-T-DS/main/5/
# wgetしなくても,Google colab の左メニュー [ファイル] アイコンをクリックして,ブラウザへファイルを
# ファイル (udon.csv) がダウンロード・配置できたことを確認する
!ls -al ./
```

```
In [2]: # 全ての警告メッセージを非表示(オプション)
import warnings
warnings.simplefilter('ignore')
```

```
In [3]: # オリジナルのCSVファイルをpandasで読み込んでデータフレームdfに格納

# ライブラリのインポート
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats
from scipy.cluster.hierarchy import linkage, dendrogram

# データの読み込み
df = pd.read_csv('sample_data1.csv')
df.columns = ['no', 'side_job', 'income', 'sex', 'age', 'married']
df.head()
##df.describe()
```

```
Out[3]:   no  side_job  income  sex  age  married
          0         1      580    1    32       0
          1         0      430    0    28       0
          2         1      800    1    45       1
          3         1      780    0    36       0
          4         1      690    0    42       1
```

```
In [4]: # 列noを削除後、全ての列で標準化実行
df = scipy.stats.zscore(df.drop('no', axis=1))
```

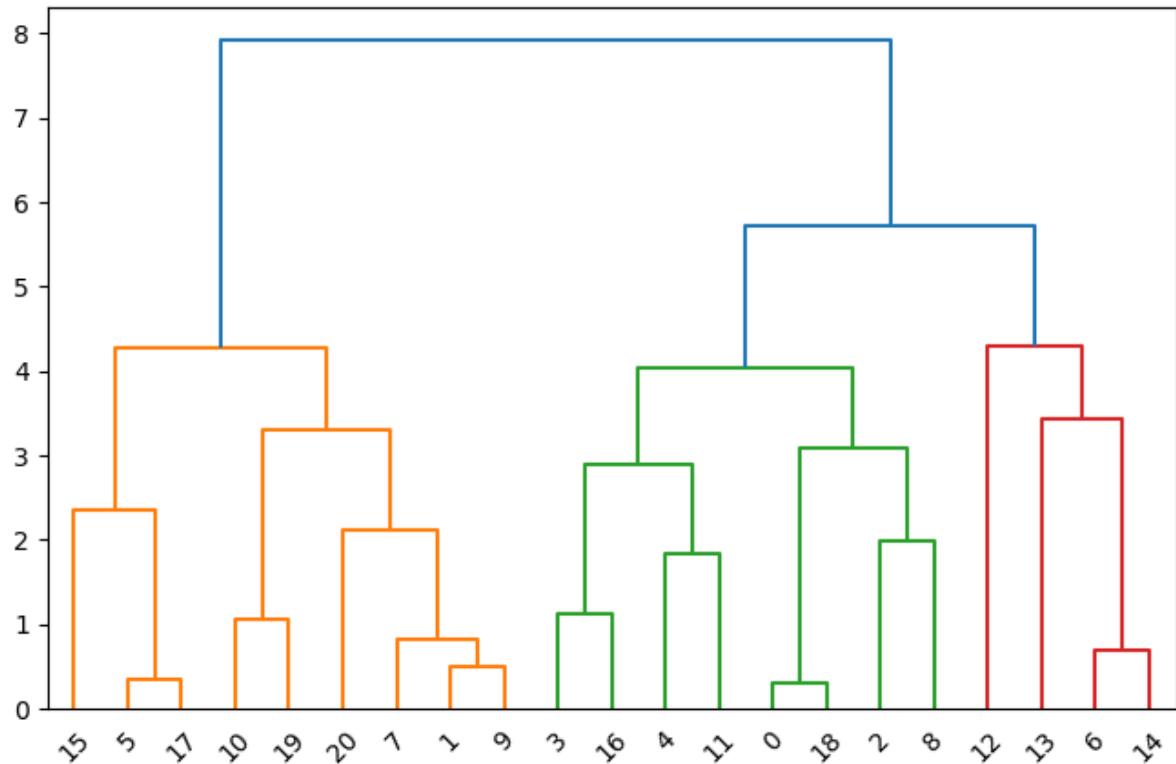
```
In [5]: # SciPyライブラリ::cluster.hierarchyによるデンドログラム生成例 - コード引用 https://corvee
# クラスタリングの実行(ユークリッド距離、ウォード法)
cluster = linkage(df, method='ward', metric='euclidean')

# 描画領域の定義
```

```
plt.figure(num=None, figsize=(8, 5))
```

```
# 樹形図の作成・出力  
dendrogram(cluster)  
plt.show()
```

```
# 縦軸を5の位置で切ると3つに分類することができた。それぞれのクラスタ群が何によってグループ化  
# されているのか、元のデータに基づいて考察して欲しい(副業ありの場合、高収入な傾向ではある)。
```



1.3 非階層クラスター分析 (k-means) [2]

階層的クラスタリングは全データバラバラの状態から、1つずつクラスターを形成していく、最終的に樹形図ができる。しかし全データ間の距離を計算する必要があり、クラスター間の距離の定義によっては毎回距離を更新していく必要があるために、ビッグデータへの適用が困難である。

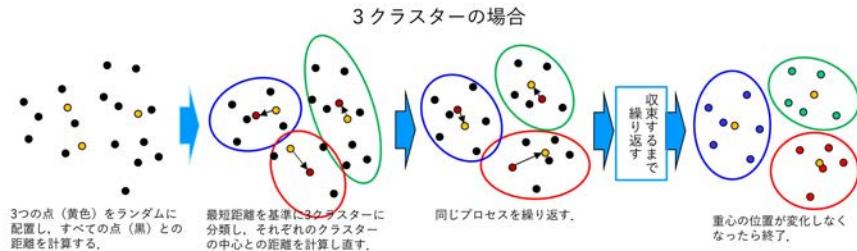
非階層的クラスタリングの代表的な **k-means** 法では、樹形図（階層的な構造）を想定せず、予めクラスタ数を与え、適当に定めた初期クラスターから繰り返し計算することで、与えられた距離を最小にするようにクラスターを求めるものである[2]。

非階層的クラスタリング

階層的クラスタリングは全データバラバラの状態から、一つずつクラスターを形成していき、また最終的に樹形図ができるので、直感的にもわかりやすいです。

しかしながら、この方法ではまず全データ間の距離を計算する必要があり、またクラスター間の距離の定義によっては、毎回距離を更新していく必要があるために、巨大なデータに適用することは困難です。

非階層的クラスタリングの代表的な *k-means* 法では、樹形図（階層的な構造）を想定せず、予め定めたクラスター数について、適当に定めた初期クラスターから繰り返し計算によって、与えられた距離を最小にするようにクラスターを求めます。



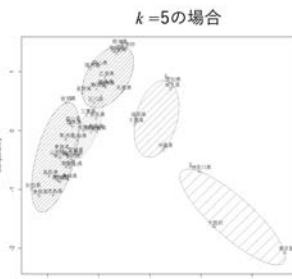
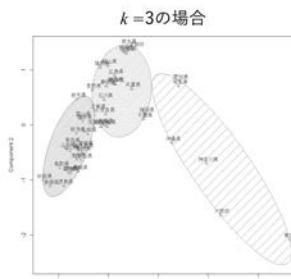
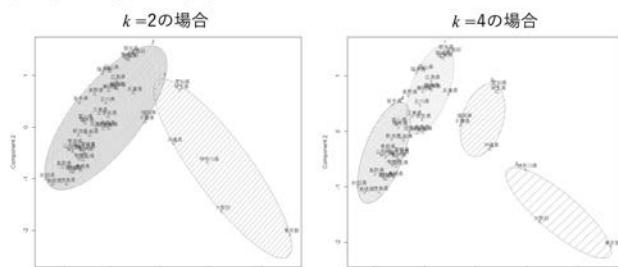
東京大学 数理・情報教育研究センター 北川源四郎 2021 CC BY-NC-SA

53

非階層的クラスタリングの例

階層的クラスタリングでも用いた多変量データのうち1, 3, 4, 5, 6番目の5変量データに *k-means* 法を適用した結果です。 *k*としては2, 3, 4, 5 の4つの場合を示しています。右図では5次元のうち、第1主成分と第2主成分だけを表示しています。

大雑把にいと沖縄を除き、*k*=2 の場合は首都圏・近畿・中部の人口密度が高い県とそれ以外にグループ化されています。*k*=3 ではそれ以外が2つに分けられ、*k*=4 の場合は、さらに大都市圏が2分割されています。



東京大学 数理・情報教育研究センター 北川源四郎 2021 CC BY-NC-SA

54

Pythonによる非階層クラスター分析 *k-means* の実装例

非階層クラスター分析の具体的な計算方法について Scikit-learn ライブリを用いて解説する。

日本の47都道府県の人口千人あたりの新型コロナウイルスの累積感染者数（2021年4月当時）のデータ

- オリジナルデータとプログラム例（公開教材[1]ただし実装はR）
- CSVデータ
- *k-means* 法のpythonによる実装とクラスター数の決定方法 エルボー法、シルエット分析

```
In [ ]: # 日本語フォントをpipでインストールする(最初に1度だけでよい)
!pip install japanize-matplotlib
# seabornで日本語ラベルが付けられるようにする
import japanize_matplotlib
```

```
In [ ]: # CSVデータを カレントディレクトリ直下のフォルダ(一時作業領域)へダウンロードする。
!wget -nc https://raw.githubusercontent.com/MDASH-shinshu/MDASH-T-DS/main/5/
# wgetしなくても,Google colab の左メニュー [ファイル] アイコンをクリックして,ブラウザへファイルを
# ファイル (udon.csv) がダウンロード・配置できることを確認する
!ls -al ./
```

```
In [8]: # ライブライリのインポート
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA, KernelPCA
```

```
In [9]: # オリジナルのCSVファイルをpandasで読み込んでデータフレームdfに格納
df = pd.read_csv('Tokyo_Covid-19_regression.csv', header=1)
df.columns = ['prefecture', 'positive_rate', 'population', 'density', 'aging_rate', 'density_central', '2hours', '4hours']
df.head()
```

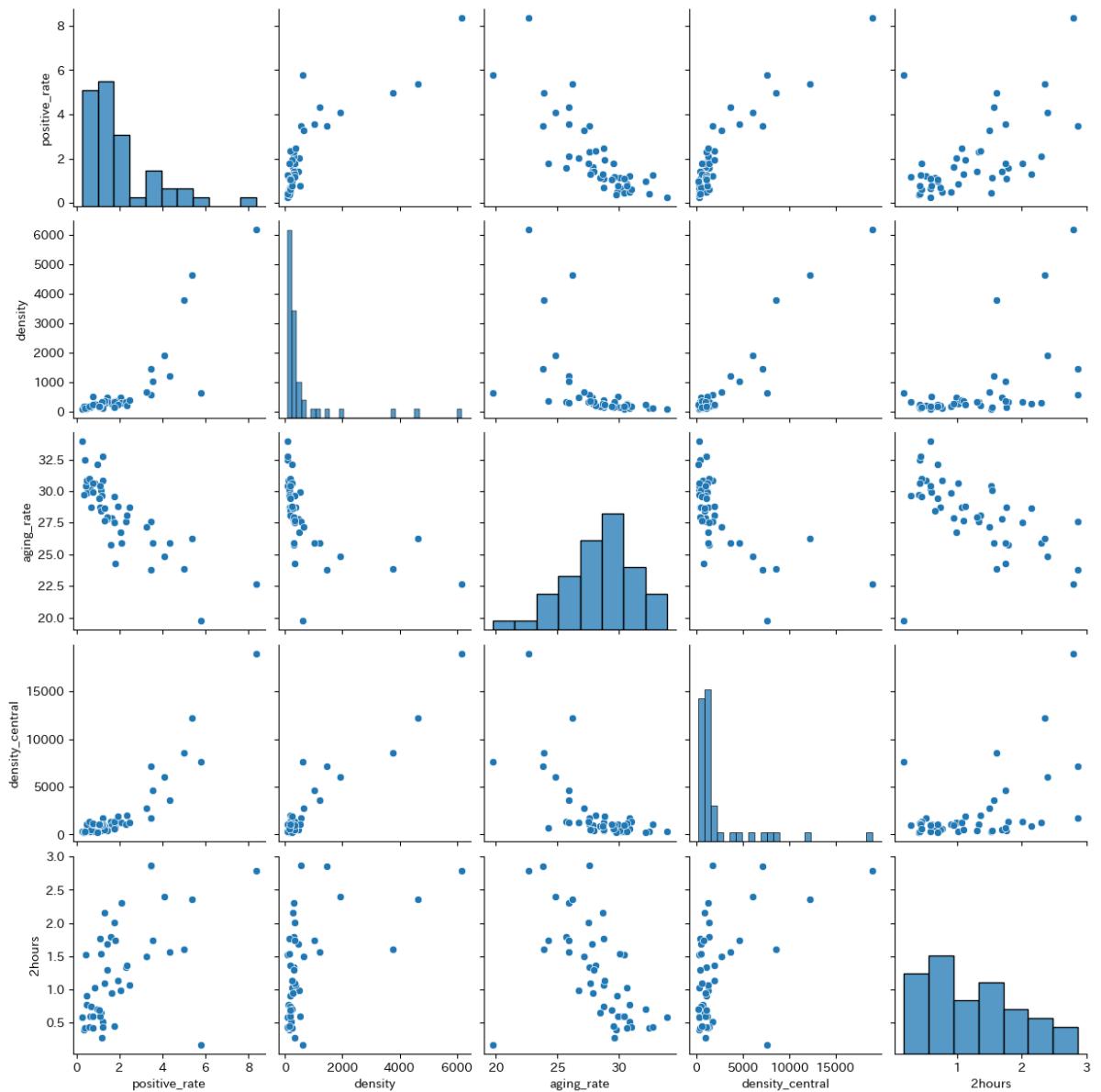
```
Out[9]: prefecture  positive_rate  population  density  aging_rate  density_central  2hours  4hour
0 青森県  0.63  1308  135.6  30.15  331.16  0.59  6.5
1 岩手県  0.43  1280  83.8  30.41  328.63  1.53  13.2
2 宮城県  1.59  2334  320.5  25.74  1388.68  1.79  15.3
3 秋田県  0.26  1023  87.9  33.92  335.55  0.58  5.7
4 山形県  0.48  1124  120.5  30.84  650.37  0.77  7.1
```

```
In [10]: # 特徴量とする列を指定する('positive_rate', 'density', 'aging_rate', 'density_centeral', '2hours', '4hours')
df5 = df.iloc[:, [1, 3, 4, 5, 6]]

# 特徴量の標準化
sc = StandardScaler()

# 学習データを変換器で標準化
X_std = sc.fit_transform(df5.values)
```

```
In [11]: # seaborn.pairplotを使って、取り出した5つの説明変数間でのペアプロットを描画してみる
pg = sns.pairplot(df5)
```



```
In [12]: # 削除後の次元を2に指定し、主成分分析器作成(PCAあるいはKernelPCA gamma: カーネル感度の指定
decomp = PCA(n_components=2)
#decomp = KernelPCA(n_components=2, kernel='rbf', gamma=0.4, random_state=0)

# 主成分分析の実行(次元削減)
X_decomp = decomp.fit_transform(X_std)
```

```
In [13]: # sklearn.cluster.KMeansを使用してクラスタリング実行
# initを省略すると、k-means++法が適応される(randomではk-means法が適応)

# n_clustersによってクラスタ分割数を指定
model = KMeans(n_clusters=3, random_state=0, init='random')
model.fit(X_decomp)

clusters = model.predict(X_decomp) # データが属するクラスターのラベルを取得
print(clusters)

# model.cluster_centers_ でクラスター重心の座標を取得できる
df_cluster_centers = pd.DataFrame(model.cluster_centers_)
df_cluster_centers.columns = ['x1', 'x2']
print(df_cluster_centers)
```

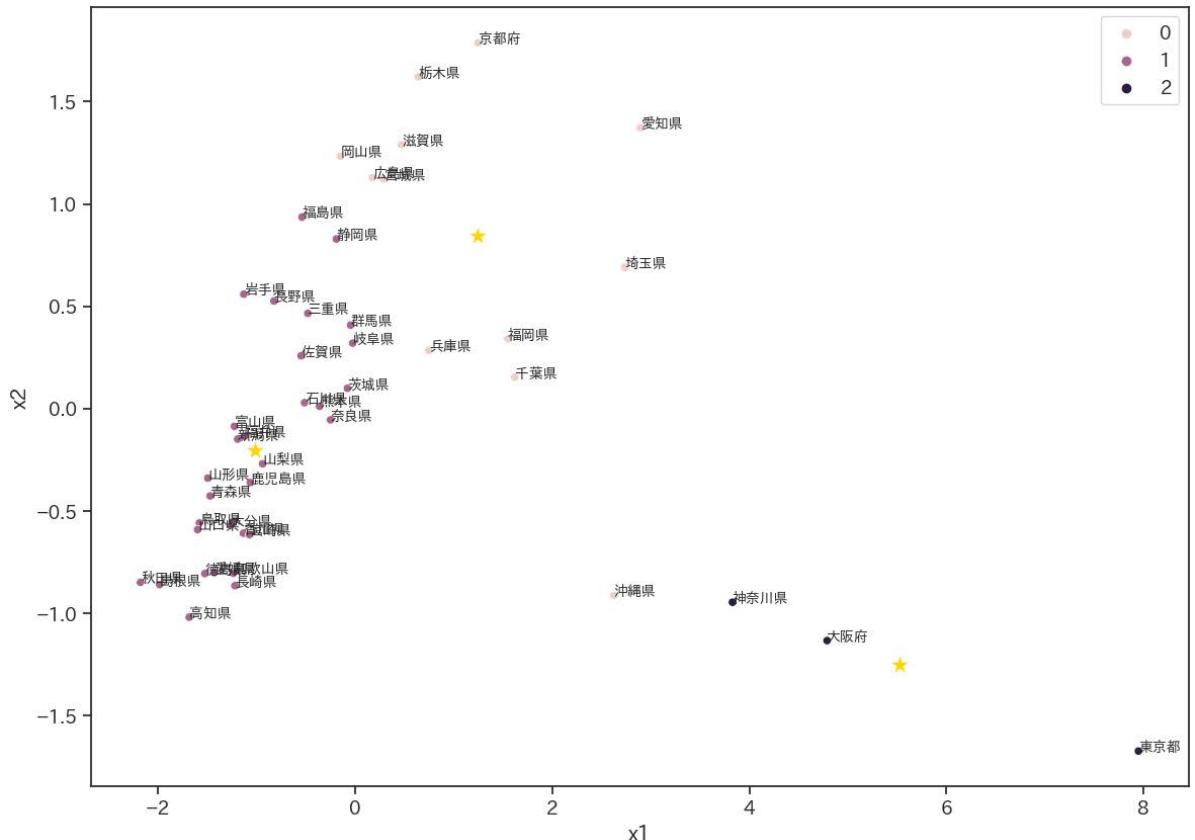
```
[1 1 0 1 1 1 1 0 1 0 0 2 2 1 1 1 1 1 1 1 0 1 0 0 2 0 1 1 1 1 0 0 1 1 1 1  
1 0 1 1 1 1 1 0]  
      x1          x2  
0  1.237425  0.842513  
1 -1.013455 -0.204954  
2  5.522671 -1.252195
```

In [14]: # seabornのプロット点にラベル付けするための関数

```
def label_point(x,y,id,ax):  
    df_tmp=pd.concat({'x':x,'y':y,'id':id}, axis='columns')  
    for i, point in df_tmp.iterrows():  
        ax.text(point['x'], point['y'], point['id'],  
        fontsize=10\  
    )
```

In [15]: # seabornで日本語ラベルが付けられるようにする

```
import japanize_matplotlib  
  
# 主成分分析後、クラスタ分類後のデータを結合する  
df_decomp = pd.DataFrame(X_decomp, columns=['x1', 'x2'])  
df_decomp['class'] = clusters  
  
# 図、フォントのサイズを設定  
plt.figure(figsize=(14, 10))  
sns.set(style="ticks", font_scale=1.2, palette='bright', color_codes=True, f  
  
# クラスタリングデータをプロット  
ax = sns.scatterplot(data=df_decomp, x='x1', y='x2', hue='class')  
# ラベル付け  
label_point(df_decomp.iloc[:,0], df_decomp.iloc[:,1], df.iloc[:,0], ax)  
  
# クラスター重心をプロット  
sns.scatterplot(data=df_cluster_centers, x='x1', y='x2', s=200, marker='*'  
  
# クラスタリング結果を総合的に可視化する  
plt.show()
```



2. 次元削減[1][2]

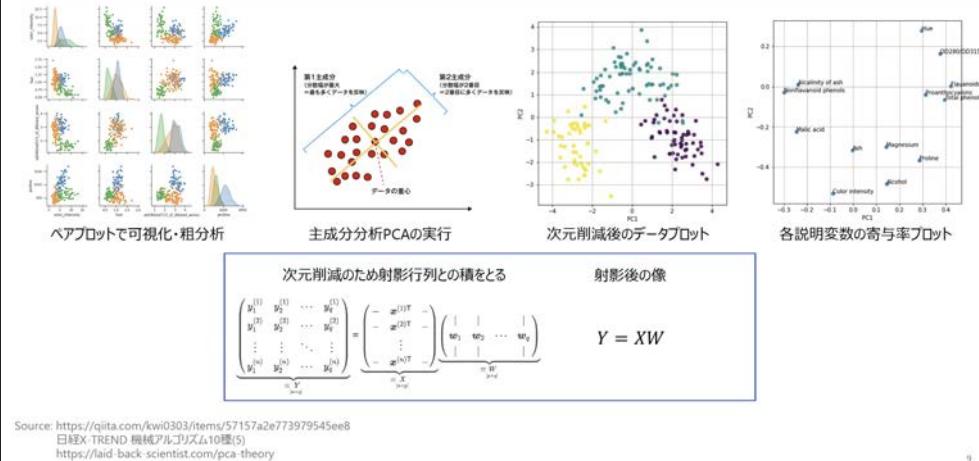
2.1 PCA(Principle Component Analysis) 主成分分析

動機：多次元のデータの取り扱い

- 多次元のデータは、さまざまな場面で現れます.
 - たとえば、100人の「身長、体重、体温、血圧、脈拍、酸素濃度、血糖値」のデータは、7次元空間の中の100個の点の集合とみなせます.
 - 高次元のデータは、低次元のデータに（近似的に）変換してから扱うと、さまざまな点で便利です.
 - データの可視化が、容易になります.
 - 少ない数の因子で、データを説明できるようになります.
 - データの容量が小さくなり、計算機で扱いやすくなります.
 - このように、与えられたデータをより低次元のデータに近似的に変換することを、次元削減とよびます.
 - 元のデータがもつ情報を完全に保ったまま次元削減を行うことは、一般に、不可能です. したがって、元のデータがもつ情報をなるべく保持したまま次元を減らすことが、次元削減の目標です.

次元削減、主成分分析(PCA)

次元削減は「連続変数/教師なし学習」の一つである。主成分分析は次元削減法のひとつアルゴリズムである。可視化とデータ圧縮の両方に利用される。



次元削減、主成分分析(PCA)

2.2 特徴選択、特徴抽出、寄与率

次元削減 (dimensional reduction)

次元削減は、機械学習の中の「連続変数/教師なし学習」の一つである。データ可視化、データ圧縮（訓練時間の短縮）、過学習抑制で前処理で使用する。

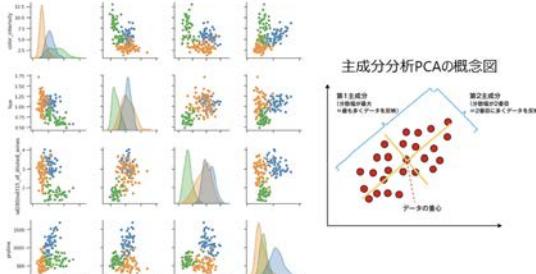
特徴選択 feature selection

- 特徴量の中から、モデルを作成する上で重要な特徴量だけを残し、残りの特徴量の次元を削減する手法
- 例えば scikit-learn サンプル学習セット「ワイン分類」の場合、13個の特徴量の中から、2個の特徴量「色の濃さ、プロリン（アミノ酸）」だけを選択し、3種類のワイン銘柄{0,1,2}に（ある程度の精度で）分類できる。
- 回帰分析の学習時に変数を自動選択する手法として、L1正則化法（Lasso回帰：損失関数は各パラメータの重み付き絶対値和）がある。正則化は過学習を抑制する目的でも用いられる。

特徴抽出 feature selection

- すべての特徴量について使う/使わないという 1/0 重み付けではなく、すべての特徴量を使って新たな軸（主成分）を作成する方法がある（主成分分析法 PCA）。
- 主成分分析は、データの元の情報（分散）を多く持つ順に、 n 個の特徴量から、第1主成分、第2主成分、..., 第 n 主成分の新たな軸を生成する。これらの主成分の中から分散が少ない軸を削減することで、次元削減を実現する。
- 主成分は元の特徴量の線形和であり、 n 次元の特徴量 x は $n \times k$ の射影行列 W を用いて直交する主成分 z (k 次元) に変換される。

$$z = xW$$



Source: ワインデータセット (13変数 178個体、3ラベル) から
プロリン x に関する3変数のペアplotを抜粋したもの
<https://qiita.com/kwi0303/items/57157a2e77397954ee8>

Source: 日経X-TREND
機械アルゴリズム10種(5)から引用
<https://laid-back-scientist.com/pca-theory>

20

特徴選択、特徴抽出

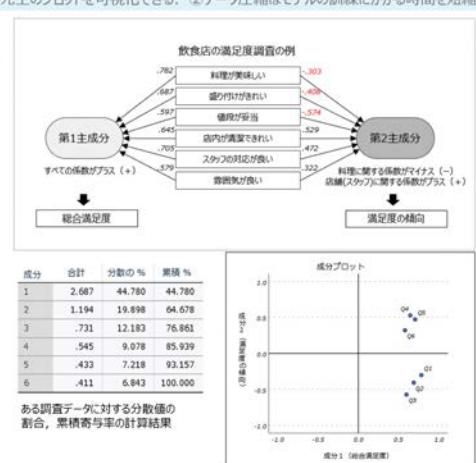
主成分分析(PCA)

主成分分析は次元削減法のひとつアルゴリズムである。可視化とデータ圧縮の両方に利用される。(1)可視化は元の次元を2次元あるいは3次元に次元削減する結果、前処理で作成した主成分を軸にプロットを作成することで、超次元上のプロットを可視化できる。(2)データ圧縮はモデルの訓練にかかる時間を短縮する。

- 主成分分析とは：相間の強い観測変数を統合して、新たな合成変数を作成する分類のための分析手法である。
- 多数の量的変数を少数の量的変数に縮約する目的で利用される。例えば、(右図にあるように)「飲食店の満足度調査」データ分析で、複数ある満足度の項目や評価項目をまとめる場合である。
- 主成分分析を実行すると、各観測変数に係る（係数、寄与率）が付けられ、主成分に呼ばれる合成変数が作成される。このとき、元のデータの持つ情報の損失が「なるべく小さなうちに」主成分を作成していく（i.e. 分散が大きい方向へ主成分軸を生成）
- 最初に作られる主成分を第1主成分、次に作られる主成分を第2主成分…と呼び、各主成分の構造を確認して分析者が名称をつけていく（各寄与率の様相によって、主成分の意味が判然とする場合）。

- 例) 第1主成分はすべての観測変数の係数がプラス (+) であるため、すべての項目の満足度が高い人口が第1主成分の得点が高くなる。つまり、第1主成分は「総合満足度」を表していると解釈することができる。
- 第2主成分は料理に関する係数がマイナス (-) で、店舗やスタッフに関する係数がプラス (+) である。よって、第2主成分は「料理重視なのかあるいはサービス重視なのか」の「満足度の傾向（の方向）」を表していると解釈することができる。

Source: StatsGrid IBM SPSSを利用したデータ分析、情報活用支援から引用
<https://www.stats-guild.com/analytics/15794>



ある調査データに対する分散値の割合、累積寄与率の計算結果

21

主成分分析、寄与率

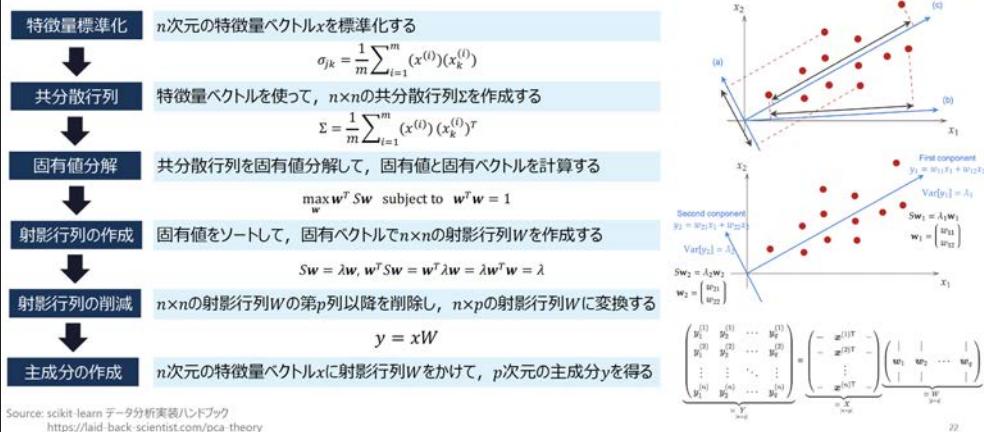
2.3 主成分分析(PCA)の計算ステップ、分散最大化

主成分分析：分散最大化としての導入

- n 次元のデータを 1 次元に圧縮することを考えましょう.
- m 個のデータ点
$$\mathbf{x}_i = [x_{i1} \ x_{i2} \ \cdots \ x_{in}] \quad (i = 1, \dots, m)$$
が与えられているとします.
- 以降の表記を簡潔にするため、データ点は行ベクトルで表します.
- 1 次元への圧縮は、 $\|\mathbf{w}\| = 1$ を満たす列ベクトル $\mathbf{w} \in \mathbb{R}^n$ を用いて $x_1\mathbf{w}, x_2\mathbf{w}, \dots, x_m\mathbf{w}$ と表せます.
- 以下では、データの情報をできるだけ失わないため、 $x_1\mathbf{w}, x_2\mathbf{w}, \dots, x_m\mathbf{w}$ の分散が最大になるように \mathbf{w} を選ぶことを考えます.
- このような次元削減の手法を、主成分分析とよびます.
- 主成分分析の解説の前に、予備知識として、行列の固有値分解と特異値分解についてまとめておきましょう.

主成分分析(PCA)の計算ステップ²²

主成分分析は、多次元データに内包する情報を「分散」で捉えて、情報を失うことなく新たな特徴量を定義していく方法である。データの分散が最大となる射影軸を求め、そこに射影するような行列を求めるのが主題となるが、このために観測データから得られる共分散行列を用いる。最大化のために、固有値問題へ置き換えて所望の解を得る。

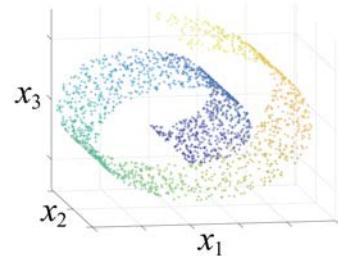
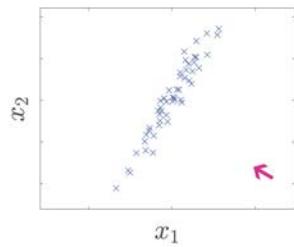


主成分分析(PCA)の計算ステップ、分散最大化

2.4 カーネルPCA、多様体埋め込み

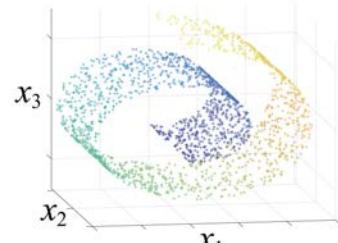
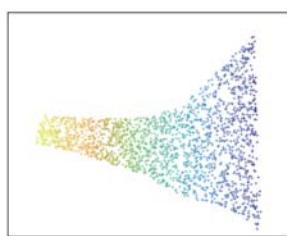
非線形の次元削減：導入 (1/2)

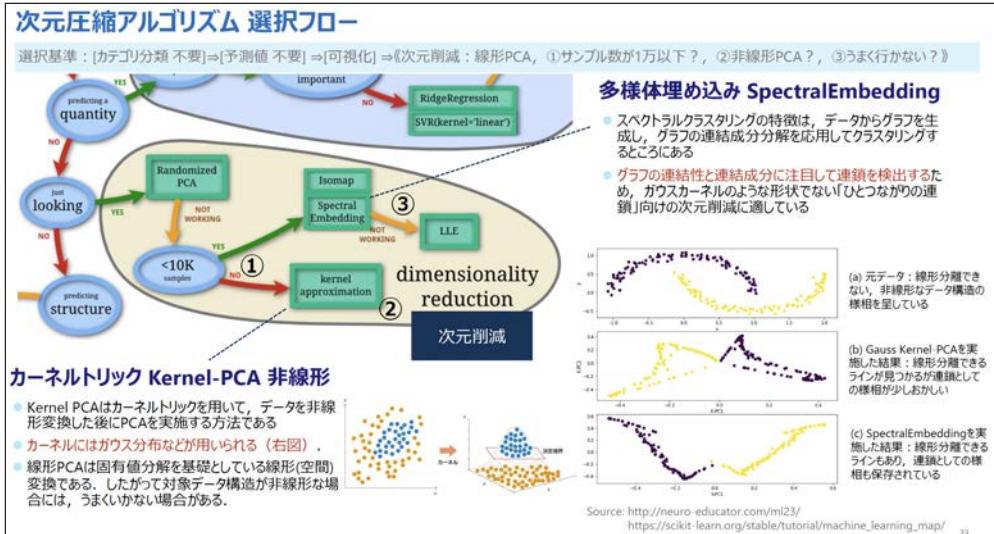
- 主成分分析は、「データを見る方向」を上手に選ぶことで、分散をなるべく保ったまま次元を削減する手法でした。「見る方向」を定めることは、主成分分析が本質的に線形変換を行っていることを意味します。
- このため、主成分分析は、たとえば左図のデータならうまく扱うことができます。
- 右図のデータ（スイスロールとよばれます）は、どうでしょうか。
 - これは3次元のデータですが、データ点は（2次元的な）曲面上に載っているように見えます。



非線形の次元削減：導入 (2/2)

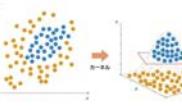
- 右図の「スイスロール」の曲面を（ある方法で）2次元平面上に広げると、左図のようになります。これで、次元が1つ削減できました。
- 曲面を平面に広げるのは非線形な変換ですので、主成分分析では扱うことができません。
- 以下では、非線形な次元削減の手法として、カーネル主成分分析をとりあげます。
- その解説の前に、予備知識として、カーネル法の概要をみておきましょう。





カーネルトリック Kernel-PCA 非線形

- Kernel PCAはカーネルトリックを用いて、データを非線形変換した後にPCAを実施する方法である
- カーネルにはガウス分布などが用いられる（右図）。
- 線形PCAは固有値分解を基礎としている線形（空間）変換である。したがって対象データ構造が非線形な場合には、うまくいかない場合がある。



Source: <http://neuro-educator.com/ml23/>
https://scikit-learn.org/stable/tutorial/machine_learning_map/

23

カーネルトリック、KernelPCA、多様体埋め込み

PythonによるKernelPCAの実装例

KernelPCAの具体的な計算方法について Scikit-learnライブラリを用いて解説する。

- [sklearn.decomposition.KernelPCA](#)
- [Qiita - Kernel PCA](#)
- [KernelPCAでカーネル主成分分析をする](#)

実装例 (circlesデータ)

scikit-learnのデータに用意されているmake_circlesを使って生成したデータで、線形分離不可能なデータを線形分離できるように変換してみる。

```
In [16]: # 実装例(circlesデータ) - コード引用 https://tech.nkhn37.net/scikit-learn-kernelpca.html

import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import make_circles
from sklearn.decomposition import KernelPCA

np.random.seed(0)

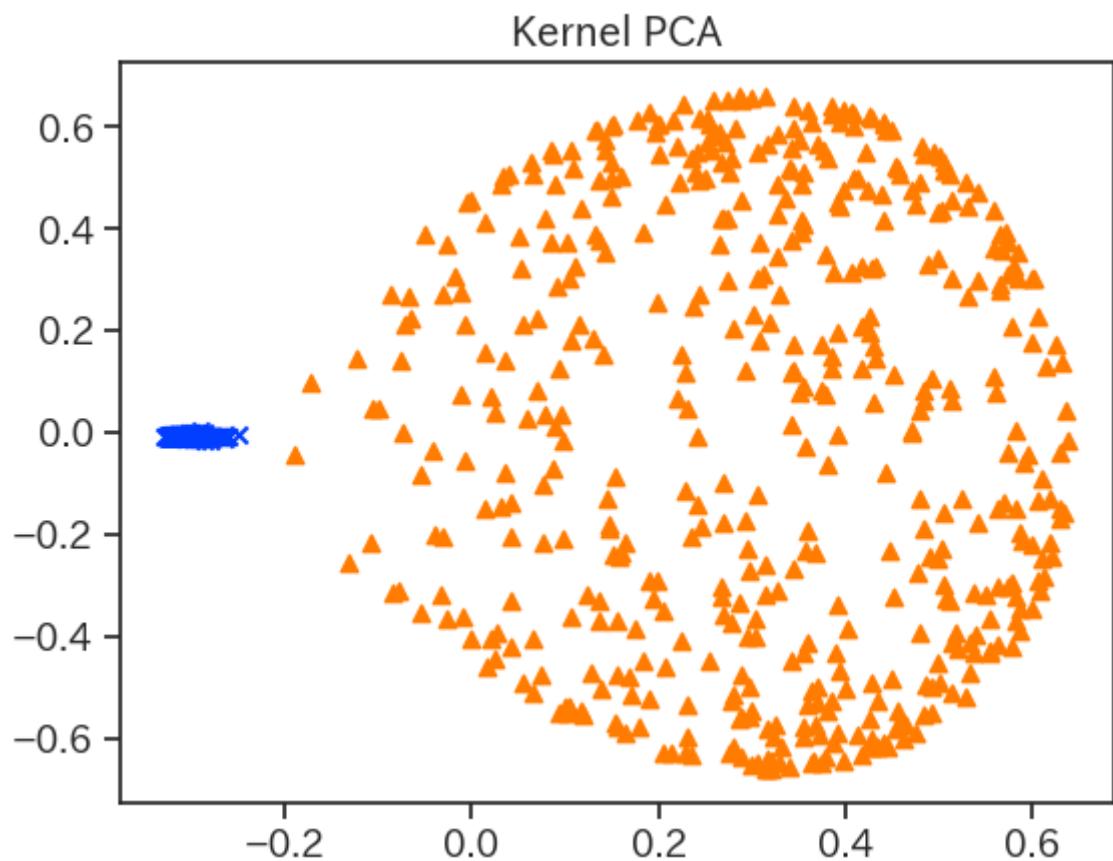
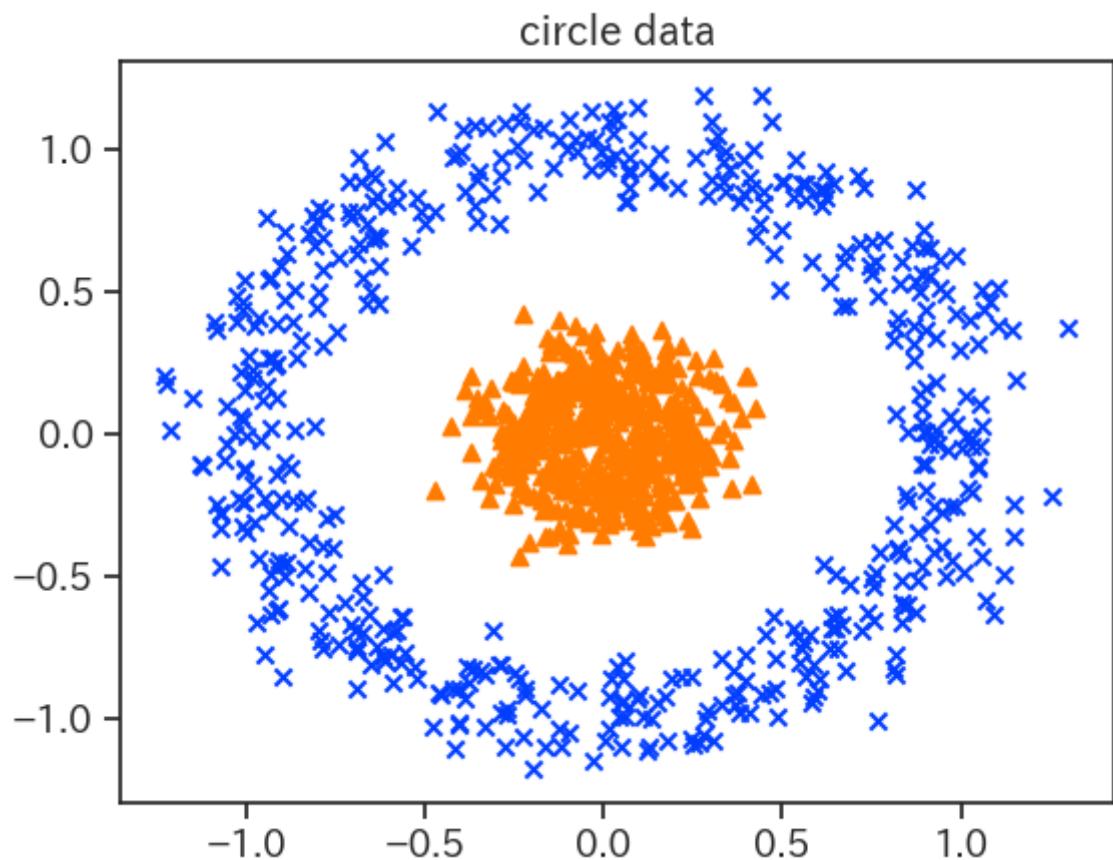
# データの用意・表示 (内側の円と外側の円で分かれているデータ)
data, label = make_circles(n_samples=1000, noise=0.1, factor=0.2)
plt.scatter(data[label == 0, 0], data[label == 0, 1], marker="x")
plt.scatter(data[label == 1, 0], data[label == 1, 1], marker="^")
plt.title("circle data")

# カーネルPCAのモデル生成・学習

# KernelPCAのハイパーコンフィグレーション:
## n_components: 主成分の数を指定
## kernel: カーネル関数を指定します。今回は'rbf' (動径基底関数)を指定。
## gamma:rbfのパラメータを指定
k_pca = KernelPCA(n_components=2, kernel="rbf", gamma=15)
k_pca.fit(data, label)

# 主成分軸への変換
trans_data = k_pca.transform(data)
```

```
# 変換結果を表示
plt.figure()
plt.scatter(trans_data[label == 0, 0], trans_data[label == 0, 1], marker="x")
plt.scatter(trans_data[label == 1, 0], trans_data[label == 1, 1], marker="^")
plt.title("Kernel PCA")
plt.show()
```



```
In [17]: # 実装例(moonsデータ) - コード引用 https://tech.nkhn37.net/scikit-learn-kernelpca
```

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import make_moons
from sklearn.decomposition import KernelPCA

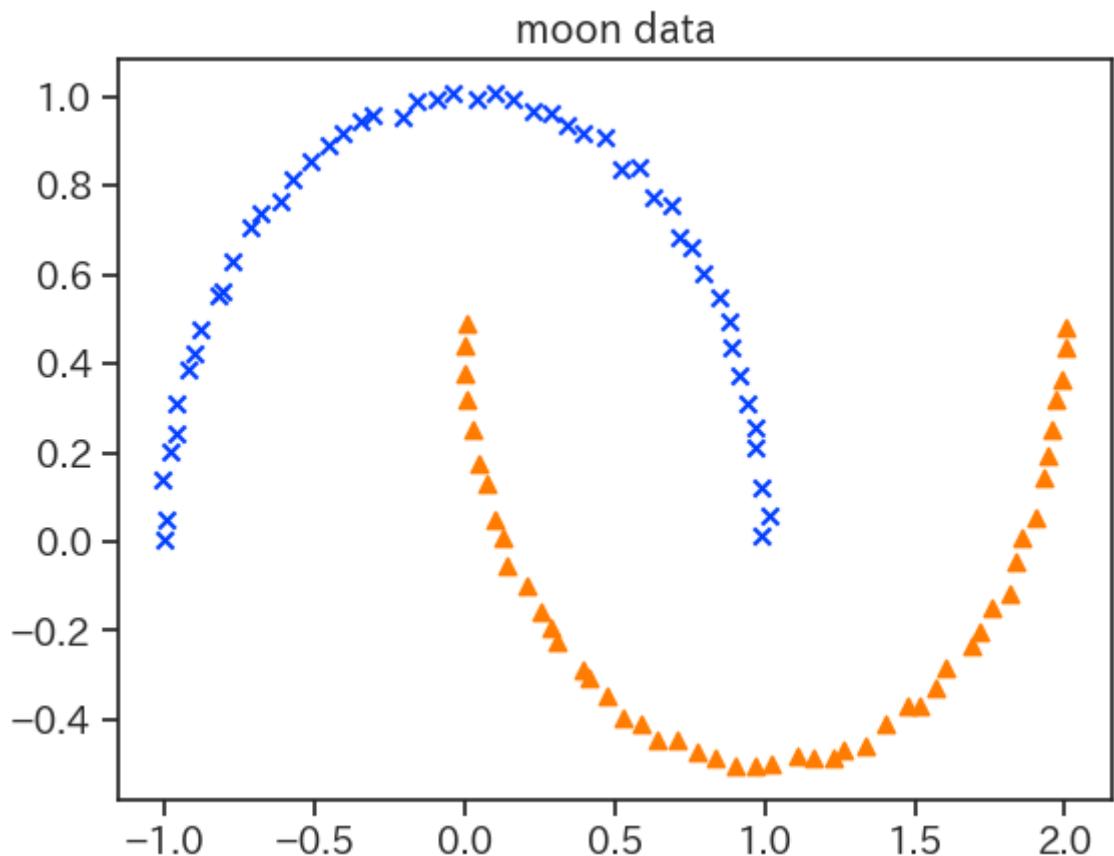
np.random.seed(0)

# データの用意・表示 (2つのムーン)
data, label = make_moons(n_samples=100, noise=0.01)
plt.scatter(data[label == 0, 0], data[label == 0, 1], marker="x")
plt.scatter(data[label == 1, 0], data[label == 1, 1], marker="^")
plt.title("moon data")

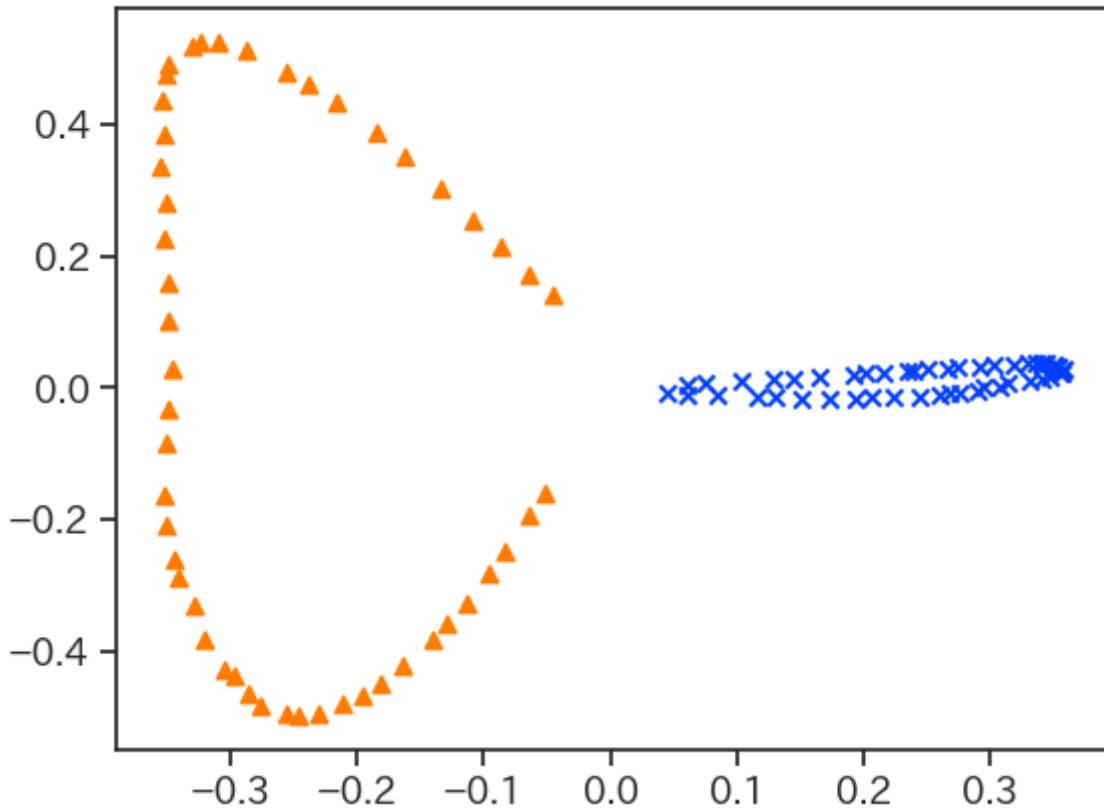
# カーネルPCAのモデル生成・学習
k_pca = KernelPCA(n_components=2, kernel="rbf", gamma=15)
k_pca.fit(data, label)

# 主成分軸への変換
trans_data = k_pca.transform(data)

# 変換結果を表示
plt.figure()
plt.scatter(trans_data[label == 0, 0], trans_data[label == 0, 1], marker="x")
plt.scatter(trans_data[label == 1, 0], trans_data[label == 1, 1], marker="^")
plt.title("Kernel PCA")
plt.show()
```

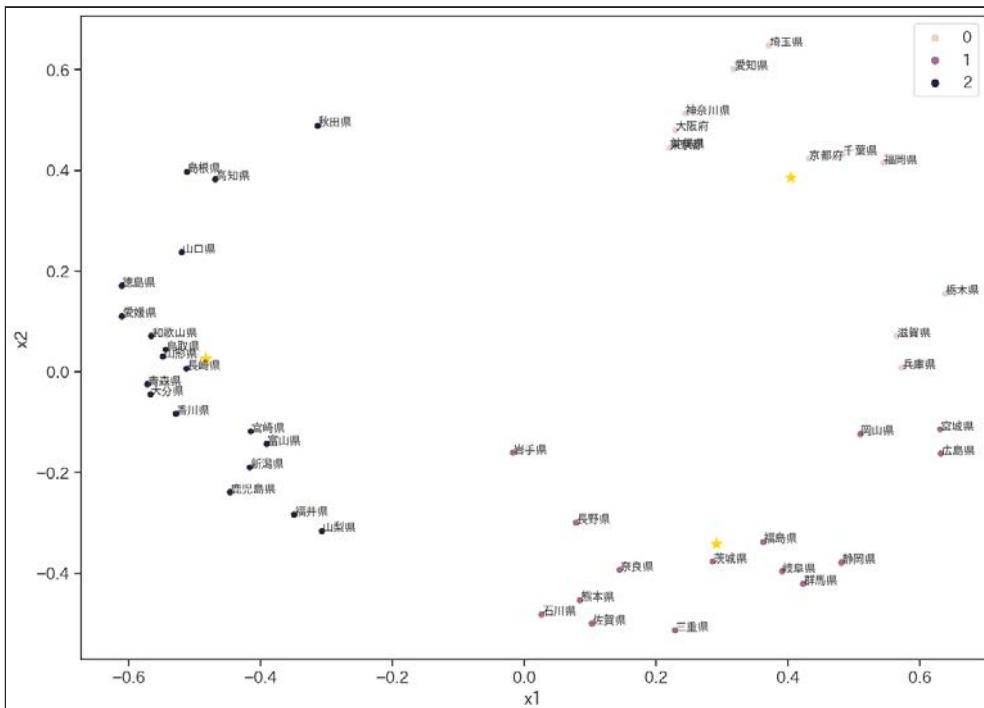


Kernel PCA



*前節のクラスタ分析のPythonコードでは、通常のPCAとともに、KernelPCAがそのまま適用できるようにコメントアウトされている。ここまで学習したからには、カーネルトリックによる分離能力について実際に評価してみると良い（カーネル感度 gammaのハイパーパラメータは要調整）。

```
decomp = KernelPCA(n_components=2, kernel='rbf', gamma=0.4,
random_state=0)
```



sec.1.3 の次元削減手法を KernelPCAによって実行した後、k-meansクラスタリングした結果

3. パターン発見（アソシエーション分析） [1][2]

販売・取引に関するデータから、意思決定に役に立つ規則や注目すべきパターンを発見する方法として、アソシエーション分析がある。

支持度、確信度、リフト値などの概念と相関ルール抽出や頻出パターン発見の方法としてのAprioriアルゴリズムを紹介する。

3.1 アソシエーション分析の概要 [1][2]

消費者の興味ある購買行動パターンをアソシエーションルール(association rule)といい、アソシエーションルールを見つけ出し、顧客の将来の行動を予測するための分析をアソシエーション分析(association analysis)と呼ぶ。

特に、商品の同時購買のパターンを発見するための分析は、マーケットバスケット分析(market basket analysis)とも呼ばれる[2]。

POSデータからのパターン発見

コンビニなどの店舗で買い物をするとレジで様々な情報が入力されており
POS (Point of Sale) データが得られます。POSデータはチェーン店などを中心に販売戦略に活かされています。

POSデータのうち、どのような商品と一緒に購入したかの部分だけからも重要な情報が得られることがあります。例えば、下記のような販売データがあったとします。このような販売データが大量にあったとき、その中から販売促進に役に立ちそうな情報を取り出す方法がアソシエーション分析です。バスケット分析と呼ばれることもあります。

ID	バスケット
1	{A, B, C, D}
2	{A, B, E}
3	{B, C, F, G}
4	{D, E, F, G, H}
5	{B, C, E, H}

パン、ジュース、牛乳などの商品は**アイテム**と呼ばれます。
一人のお客はいくつかのアイテムを購入するので、それを

{A, B, C, D}のように表して**バスケット**と呼びます。また、 $I = \{A, B\}$ のようにアイテムを集めたものを**アイテム集合**と呼びます。

多くのIDのバスケットの中に同じアイテム集合（例えばIとJ）が現れる場合、IとJは一緒に買われる可能性が高いことがわかるので、並べて陳列するなどの販売促進策が考えられます。アソシエーション分析によって、紙オムツとビールが頻出アイテムとして見出されたというのが有名な例です。

アソシエーション分析

アソシエーション分析では、膨大な販売データに見られる商品Bを購入すると商品Cも買う傾向があるというような、**関連性や規則性、相関ルール** (association rule) を見出すことを目的とします。

販売データは以下のような情報を持ちます。

- ID 通し番号, 何番目の購入(バスケット)かを n で表す
 $j(n)$ n 番目の客が購入した商品数
 I_n $\{I_1, \dots, I_{j(n)}\}$ n 番目のバスケット内のアイテム

アソシエーション分析では、全バスケットの中でアイテム集合 I が購入されたときに、アイテム集合 J が購入される傾向があるときに $I \Rightarrow J$ と表現します。

アイテムとアイテム集合

相関ルールではアイテム集合 I と J について、 $I \Rightarrow J$ という関係を考えます。ただし、 I と J は $\{A\}$ のような単独のアイテムの場合も $\{A, B, E\}$ のような複数のアイテムの集合の場合もあります。

相関ルールは I ならば必ず J が起こるというルールではないので、次頁のように**支持度、確信度、リフト値**などの指標が考慮されます。

これらの定義には以下の記号が使われます。

- N 全データ数 (バスケットの数)
 I, J アイテム集合
 $k(I)$ 全データ中にアイテム集合 I が現れる回数

ID	バスケット
1	{A, B, C, D}
2	{A, B, E}
3	{B, C, F, G}
\vdots	...
N	{B, C, E, H}

3.2 支持度、確信度、リフト値[1][2]

支持度 (support) : すべての購買履歴のうちで、商品Aと商品Bを同時に購買しているものの割合。つまり、

$$\text{支持度} = \frac{\text{商品 } A, B \text{ の同時購買履歴数}}{\text{全購買履歴数}}$$

確信度(confidence) : 商品Aの購買履歴のうちで、商品Bも同時に購買しているものの割合。つまり、

$$\text{確信度} = \frac{\text{商品 } A, B \text{ の同時購買履歴数}}{\text{商品 } A \text{ の購買履歴数}}$$

リフト値(lift)：確信度と商品Bの購買率の比。つまり、

$$\text{リフト値} = \frac{\text{確信度}}{\text{商品 } B \text{ の購買履歴数} / \text{全購買履歴数}}$$

例えば リフト値が1より大きいときは、「商品Aを買う人は、商品Aを買わない人よりも相対的に商品Bを買う傾向にある」といえる。

例) イタリアンの料理店で、チーズとパスタを注文した客は、高い確率でワインも注文するであろう[*]。

- [DXCEL WAVE | アソシエーション・バスケット分析実践法](#)

支持度, 確信度, リフト値

支持度：全データ中に I と J が同時に現れる割合

$$S(I \Rightarrow J) = \frac{k(I \cap J)}{N}$$

確信度： I が出現した時に J も同時に現れる割合

$$C(I \Rightarrow J) = \frac{k(I \cap J)}{k(I)}$$

リフト値：支持度と全データの中でアイテム集合 J が出現する割合の比

$$L(I \Rightarrow J) = \frac{C(I \Rightarrow J)}{k(J) / N} = \frac{k(I \cap J)N}{k(I)k(J)}$$

- 支持度は全体の中で I と J が同時に現れる割合で、因果関係を意味しているわけではないが、シンプルで分かりやすい指標です。
- 確信度は I と J の関係以上に J の出現割合に依存する指標なので注意が必要です。
- リフト値が1より大きければアイテム集合 J の販売に I の関連性があると考えられます。

3.2 Aprioriアルゴリズム

Aprioriとは、アソシエーション分析を実践するためのアルゴリズムのことである。アソシエーション分析において高速に相関ルールや頻出アイテム集合を検出するために1990年代前半にIBM研究所で開発されたアルゴリズムで、その後開発されたアルゴリズムの基礎ともなっている[1]。

Aprioriは「頻出アイテム（商品・サービス）の組み合わせを検出し、アソシエーションルールを決定するための評価指標（Support, Confidence, Liftなど）を算出する機能」を有した最も有名なアソシエーション分析アルゴリズムの1つである[*]。

- [DXCEL WAVE | アソシエーション・バスケット分析実践法](#)

Aprioriアルゴリズム

販売データの中に頻繁に表れる頻出パターンや相関ルールを求めるためには、すべてのアイテム集合がデータの中に何回現れるかを調べることになります。これは一見すると簡単そうに見えますが、ビッグデータの場合には組み合わせ爆発によって計算が困難になります。Aprioriアルゴリズムはアソシエーション分析において高速に相関ルールや頻出アイテム集合を検出するために1990年代前半にIBM研究所で開発されたアルゴリズムで、その後開発されたアルゴリズムの基礎ともなっています。

Aprioriアルゴリズムでは、支持度(support)と確信度(confidence)の閾値を指定するとそれ以上の値の相関ルールを列挙します。支持度を小さく設定しすぎると大量のルールを列挙するので注意が必要です。

東京大学 数理・情報教育研究センター 北川源四郎 2021 CC BY-NC-SA

62

Python mlxtendライブラリによる Aprioriアルゴリズム/アソシエーション分析の実装例

Aprioriアルゴリズム/アソシエーション分析の具体的な計算方法についてPython mlxtendライブラリを用いて解説する。データは、KaggleのApriori Association Grocery Storeから、あるスーパーマーケットにおける購買データ(max32商品、n=9835)である（すべて質的変数であることに注意）。

- [Kaggle | Apriori Association Grocery Store](#)
- [Kaggle | Grocery Products Purchase Data \(CSV\)](#)
- [Mlxtend \(machine learning extensions\)](#)

目的：スーパーマーケットで売られている商品の組み合わせに対してアソシエーション分析を適用し、強い関係性を示す（高頻度で一緒に購入される）商品の組み合わせをアソシエーション・ルールとして抽出する。

- [コード引用 DXCEL WAVE | アソシエーション・バスケット分析実践法](#)

```
In [ ]: # mlxtendパッケージをインストール(最初の1回だけ)
!pip install mlxtend
```

```
In [ ]: # CSVデータを カレントディレクトリ直下のフォルダ(一時作業領域)へダウンロードする。
!wget -nc https://raw.githubusercontent.com/MDASH-shinshu/MDASH-T-DS/main/5/
# wgetしなくても,Google colab の左メニュー「[ファイル]」アイコンをクリックして,ブラウザへファイルを
# ファイル (udon.csv) がダウンロード・配置できたことを確認する
!ls -al ./
```

```
In [20]: # 全ての警告メッセージを非表示(オプション)
import warnings
warnings.simplefilter('ignore')
```

```
In [21]: # ライブドリのインポート
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
```

```
In [22]: # オリジナルのCSVファイルをpandasで読み込んでデータフレームdfに格納
data = pd.read_csv('Grocery_Products_Purchase.csv', header=1)
data.columns = ['Product1', 'Product2', 'Product3', 'Product4', 'Product5', 'Product6', 'Product7', 'Product8', 'Product9', 'Product10', 'Product11', 'Product12', 'Product13', 'Product14', 'Product15', 'Product16', 'Product17', 'Product18', 'Product19', 'Product20', 'Product21', 'Product22', 'Product23', 'Product24', 'Product25', 'Product26', 'Product27', 'Product28', 'Product29', 'Product30', 'Product31', 'Product32']
data.head()
```

```
Out[22]:
```

	Product1	Product2	Product3	Product4	Product5	Product6	Product7	Product8	Product9	Product10	Product11	Product12	Product13	Product14	Product15	Product16	Product17	Product18	Product19	Product20	Product21	Product22	Product23	Product24	Product25	Product26	Product27	Product28	Product29	Product30	Product31	Product32
0	tropical fruit	yogurt	coffee	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN			
1	whole milk	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN			
2	pip fruit	yogurt	cream cheese	meat spreads	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN			
3	other vegetables	whole milk	condensed milk	long life bakery product	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN			
4	whole milk	butter	yogurt	rice	abrasive cleaner	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN			

5 rows × 32 columns

データ前処理

```
In [ ]: # データ加工 / トランザクション形式データセットに変換(可変長のリスト形式)

# レコード数
record_len = len(data)
# カラム数
column_len = len(data.columns)

# トランザクション形式に加工
transactions = []
for i in range(record_len):
    # データをリスト型に変更
    values = [str(data.values[i,j]) for j in range(column_len)]
    values_notnull = []
    for check in values:
        if check != 'nan':
            values_notnull.append(check)
    transactions.append(values_notnull)

# 変換結果の表示
## print(transactions)
```

```
In [24]: # データ加工 / テーブル形式データセットに変換(各商品別に顧客の購買の有無をTrue/Falseで表現)
from mlxtend.preprocessing import TransactionEncoder

# データをテーブル形式に加工
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_array, columns=te.columns_)
```

```
df.head()
```

Out[24]:

	Instant food products	UHT-milk	abrasive cleaner	artif. sweetener	baby cosmetics	baby food	bags	baking powder	bathroom cleaner	beef
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False
4	False	False	True	False	False	False	False	False	False	False

5 rows × 169 columns

Aprioriアルゴリズムの実行

In [25]: # 商品の組み合わせ別に支持度(Support)を算出する

```
from mlxtend.frequent_patterns import apriori

freq_items = apriori(df,
                      min_support = 0.01,           # データフレーム
                      use_colnames = True,          # 結果として生成されるデータのSupport
                      max_len      = None,          # 出力値のカラムに購入商品名を表示
                      )                            # 生成されるitemsetsの個数

# 結果出力(支持度(>=min_support)が高い順に商品名が出てくる)
freq_items = freq_items.sort_values("support", ascending = False).reset_index()
print(freq_items)
```

```
support           itemsets
0    0.255542      (whole milk)
1    0.193512      (other vegetables)
2    0.183954      (rolls/buns)
3    0.174395      (soda)
4    0.139516      (yogurt)
..   ...
328   0.010067     (sausage, frankfurter)
329   0.010067     (curd, whole milk, yogurt)
330   0.010067     (rolls/buns, curd)
331   0.010067     (tropical fruit, napkins)
332   0.010067     (hard cheese, whole milk)
```

[333 rows × 2 columns]

In [26]: # モデル作成 / アソシエーション・ルール決定に用いる評価値算出

```
from mlxtend.frequent_patterns import association_rules

# アソシエーション・ルール抽出
df_rules = association_rules(freq_items,                 # supportとitemsetsを持つ
                             metric = "confidence",    # アソシエーション・ルールの評価
                             min_threshold = 0.1,       # metricsの閾値
                             )
```

```
# 支持度の高い組み合わせ別に、評価値が出力される 信頼度(confidence) リフト値(lift) でソートし
df_rules
```

Out [26]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
0	(other vegetables)	(whole milk)	0.193512	0.255542	0.074842	0.386758	1.513480
1	(whole milk)	(other vegetables)	0.255542	0.193512	0.074842	0.292877	1.513480
2	(rolls/buns)	(whole milk)	0.183954	0.255542	0.056640	0.307905	1.204909
3	(whole milk)	(rolls/buns)	0.255542	0.183954	0.056640	0.221647	1.204909
4	(whole milk)	(yogurt)	0.255542	0.139516	0.056030	0.219260	1.571575
...
455	(whole milk, yogurt)	(curd)	0.056030	0.053285	0.010067	0.179673	3.371961
456	(curd)	(whole milk, yogurt)	0.053285	0.056030	0.010067	0.188931	3.371961
457	(curd)	(rolls/buns)	0.053285	0.183954	0.010067	0.188931	1.027059
458	(napkins)	(tropical fruit)	0.052369	0.104942	0.010067	0.192233	1.831802
459	(hard cheese)	(whole milk)	0.024507	0.255542	0.010067	0.410788	1.607518

460 rows × 9 columns

In [27]: # 分析モデルより得られた結果(df_rules)からデータ抽出し、最終的なアソシエーション・ルールを決定して
抽出条件として今回は 信頼度>0.2 かつ リフト値>3.0 を設定する

```
results = df_rules[(df_rules['confidence'] > 0.2) & # 信頼度
                   (df_rules['lift'] > 3.0)] # リフト値

# 結果出力
results.loc[:, ["antecedents", "consequents", "confidence", "lift"]]
```

Out [27]:

	antecedents	consequents	confidence	lift
162	(beef)	(root vegetables)	0.331395	3.040058
300	(tropical fruit, other vegetables)	(root vegetables)	0.342776	3.144460
301	(tropical fruit, root vegetables)	(other vegetables)	0.584541	3.020692
415	(other vegetables, citrus fruit)	(root vegetables)	0.359155	3.294710
416	(root vegetables, citrus fruit)	(other vegetables)	0.586207	3.029300
445	(other vegetables, yogurt)	(whipped/sour cream)	0.234192	3.266730

レコメンド戦略に向けた考察

上記の結果において、例えば163行目を見ると、「beef」を購入する顧客は「root vegetables(根菜)」を普通より3倍高い確率で購入している（リフト値=3.040058）ことが

分かります。

ステーキとかローストビーフを作る際には、ジャガイモとか人参、にんにくやセロリと一緒に買って調理に使ったり付け合せにしたりしますよね。

レコメンド機能実装やクロスセルを促す際は「もし顧客がbeefを購入する場合、root vegetablesをいくつかレコメンドする」というアソシエーション・ルールが適用できそうです。

もちろんローストビーフであれば赤ワインが欠かせない訳で、更に確信度とリフト値を探ってエビデンスが得られれば、店舗のお肉売り場の近くにローストビーフのレシピやトップ、同時に赤ワインを薦めるようなトップがあったり、ワイン売り場への案内があったりすると効果的と言えそうです（ここから先はデータサイエンスではなく、経済・マーケティングの領域となる）。

memo