機械学習
教師なし学習２：
クラスタリング

Pythonによる機械学習入門

第１５章

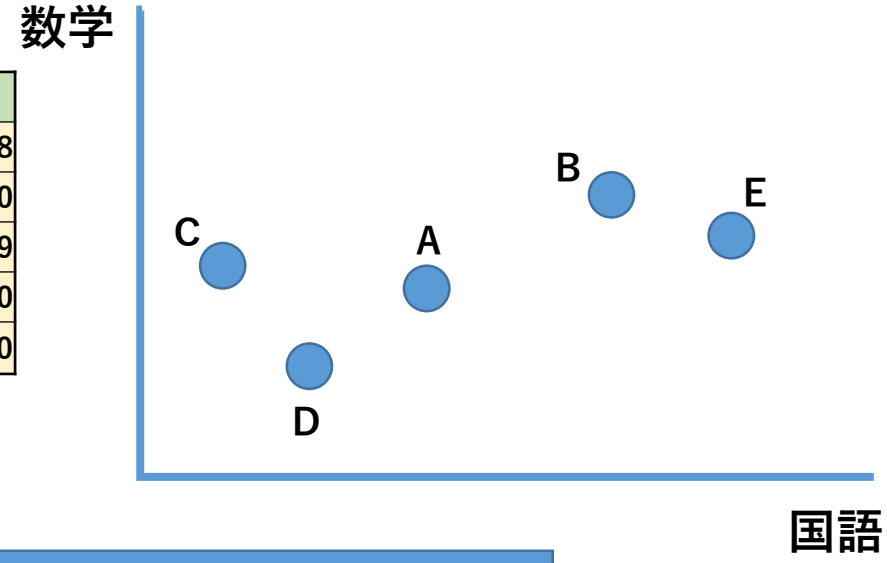
CONTENTS	
１５．１	クラスタリングの概要
１５．２	データの前処理
１５．３	クラスタリングの実行
１５．４	結果の評価
１５．５	第１５章のまとめ
１５．６	練習問題
１５．７	練習問題の解答

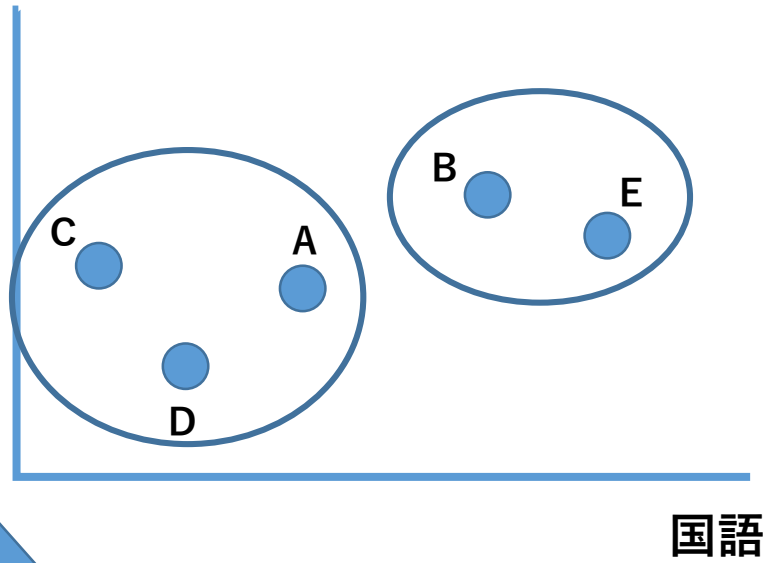
１５．１　クラスタリングの概要　P530～P539

１５．１．１　クラスタリングとは　P530～P539

機械学習におけるクラスタリング　データを似ているもの同士でグループ分けする分析

氏名	国語	数学
Aさん	63	68
Bさん	85	70
Cさん	58	69
Dさん	60	50
Eさん	90	70





特徴量が国数英理社保の６科目になると
散布図を描くことができない

**教師なし学習の
クラスタリングを利用する**

クラスタリングを用いることで各データの特徴量を調べて、距離が近いデータ同士がグループになるようにグループ分けすることができる

１５．１．２ k-means法 P533～P539

k-means法 クラスタリングのいくつかの手法のうち一番有名な手法

k-means法によるクラスタリングの流れ

■ 手順０：クラスタ数 k を決める

k-means法では、分析前に何個のクラスタを作成するかを決める必要がある

■ 手順１：クラスタの個数分、ランダムにデータを選ぶ

P534　図15-6

代表点 ランダムに選ばれたデータ この代表点をもとに、それ以外のデータをクラスタのどれかに所属させる

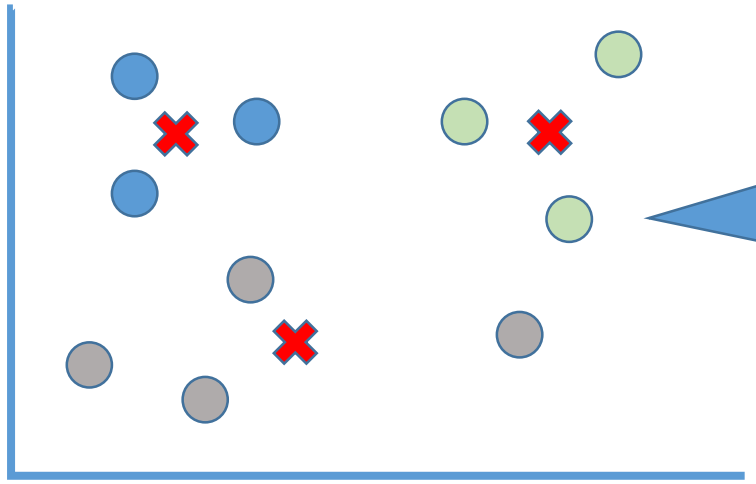
■ 手順２：データと各代表点の距離をもとに、各クラスタに所属させる

P535　図15-7

P535

■ 手順３：各クラスタの中心点を計算して、その中心点を代表点として更新する

P536　図15-9



代表点を各クラスタの中心点に変更したので、各データの所属するクラスタも変わる可能性がある

■ 手順４：データと各クラスタの代表点との距離をそれぞれ計算して、一番近い代表点のクラスタに所属させる

P537　図15-11

■ 手順５：クラスタの中心点を再計算して、代表点を更新する

P537　図15-12

■ 手順６：手順４，手順５を繰り返す

何度か繰り返していくと、中心点が変化しなくなる。
その最終形が求めるべきクラスタ

１５．２　データの前処理　P540～P543

１５．２．１　データの読み込み　P540～P541

コード15-1 データの読み込み

```
# pandasをインポート
import pandas as pd
# Wholesale.csvを読み込みデータフレームにする
df = pd.read_csv('Wholesale.csv')
# 読み込んだデータフレームを表示
df.head(3)
```

実行結果

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844

- Channel：顧客の業態（サービス業、小売業）
- Region：地域（リスボン、ポルト、その他）
- Fresh：生鮮食品の販売数
- Milk：乳製品の販売数
- Grocery：食料雑貨品の販売数
- Frozen：冷凍食品
- Detergents_Paper：洗剤と紙製品の販売数
- Delicassen：惣菜の販売数

１５．２．２ 欠損値の確認 P541～P542

コード15-2 欠損値の確認

```
# 欠損値を確認
df.isnull().sum()
```

実行結果

```
Channel            0
Region             0
Fresh              0
Milk               0
Grocery            0
Frozen             0
Detergents_Paper   0
Delicassen         0
dtype: int64
```

Channel列もRegion列も数値の列であるが、文字列データであるところ、１から整数を割り振っているにすぎない。

Chnnel列とRegion列を機械学習で利用するためには、ダミー変数化を行う必要がある

ただし、ダミー変数化を行うと列数が一気に増えるため、結果の考察やグラフ化したときの見やすさに支障が出る可能性がある。

コード15-3 ChannelとRegion列を削除

```
# Channel列とRegion列を削除
df = df.drop(['Channel', 'Region'], axis = 1)
```

今回は削除

コード15-2 データを標準化する

```
# 標準化モジュールをインポート
from sklearn.preprocessing import StandardScaler
# 標準化インスタンスを生成
sc = StandardScaler()
# 標準化する（fitとtransformを一括して実行）
sc_df = sc.fit_transform(df)
# カラム名を追加してデータフレーム化
sc_df = pd.DataFrame(sc_df, columns=df.columns)
```

１５．３　クラスタリングの実行　P544～P547

１５．３．１　モジュールのインポート　P544～P544

コード15-5 モジュールのインポート

```
# KMeansをインポート
from sklearn.cluster import KMeans
```

１５．３．２　モデルの作成　P544～P547

コード15-6 モデルの作成

クラスタ数の指定

乱数の固定

```
# k-meansモデルを作成
model = KMeans(n_clusters = 3, random_state = 0)
```

k-means法のモデル作成

```
変数 = KMeans( n_clusters = ●, random_state = ▲ )

※ from.sklearn.cluster import KMeans を事前に実行済み
※ n_clusterには、クラスタ数を指定。
※ random_state には乱数固定のための整数を指定。
```

コード15-7 モデルに学習させる

```
# モデルに学習させる
model.fit(sc_df)
```

コード15-8 クラスタリングの結果を確認

```
# クラスタリングの結果を確認
model.labels_
```

コード15-9 クラスタリング結果を追加

```
# クラスタリングの結果を cluster列として追加
sc_df['cluster'] = model.labels_
# 先頭の２行を表示
sc_df.head(2)
```

実行結果

元データ０行目のクラスタ番号

```
array([1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 2, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0,
       0, 0, 0, 2, 0, 0, 1, 1, 1, 0, 1, 1, 2, 0, 1, 1, 1, 2, 1, 0, 1, 2,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 2, 2, 1,
       1, 1, 1, 1, 2, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 2, 0, 2, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0,
       1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 2, 1, 0, 0, 0, 0, 1, 0, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 1, 1, 1, 2, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 2, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0,
       0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1])
```

実行結果

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen	cluster
0	0.052933	0.523568	-0.041115	-0.589367	-0.043569	-0.066339	1
1	-0.391302	0.544458	0.170318	-0.270136	0.086407	0.089151	0

モデルの学習とクラスタ番号の確認

・モデルの学習
モデル変数.fit(特徴量のデータ)
※ 引数には、データフレームや２次元の numpy 配列を指定できる

・クラスタ番号の確認
モデル変数.labels_
※ ０から始まる整数が割り振られている

１５．４　結果の評価　P548～P556

１５．４．１　クラスタの特徴量考察　P548～P551

クラスタリングでは、どのデータが何番クラスタであるかを求めることはできる

しかし

「●番クラスタはどういう共通点を持ったデータの集団なのか」まではわからない

しかし

クラスタリングの結果をもとに、分析者が自分で考察して、各クラスターの特徴を把握する必要がある。

クラスタごとに特徴量の平均値を集計する

コード15-10 groupbyメソッドでクラスタごとに集計する

```
# groupbyメソッドでクラスタごとに平均値を集計する
sc_df.groupby('cluster').mean()
```

実行結果

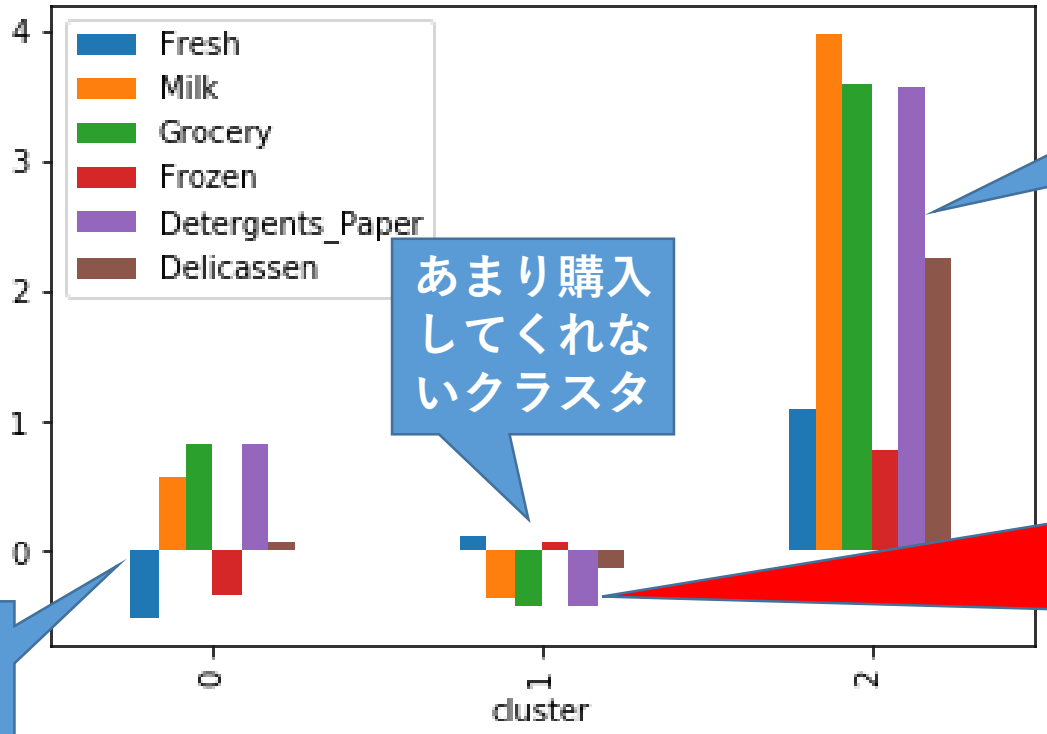
cluster	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	-0.503804	0.574921	0.823225	-0.332913	0.816078	0.075524
1	0.128126	-0.359900	-0.428714	0.082307	-0.425531	-0.118121
2	1.090044	3.983203	3.584579	0.777993	3.566641	2.256182

コード15-11 棒グラフで表示する

```
# コマンド（同一タブにグラフを表示）
%matplotlib inline
# クラスターごとに平均値を集計
cluster_mean = sc_df.groupby('cluster').mean()
# 集計結果を棒グラフで表示
cluster_mean.plot(kind = 'bar')
```

実行結果

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b8d6540eb8>
```



たくさん購入するクラスタ

あまり購入してくれないクラスタ

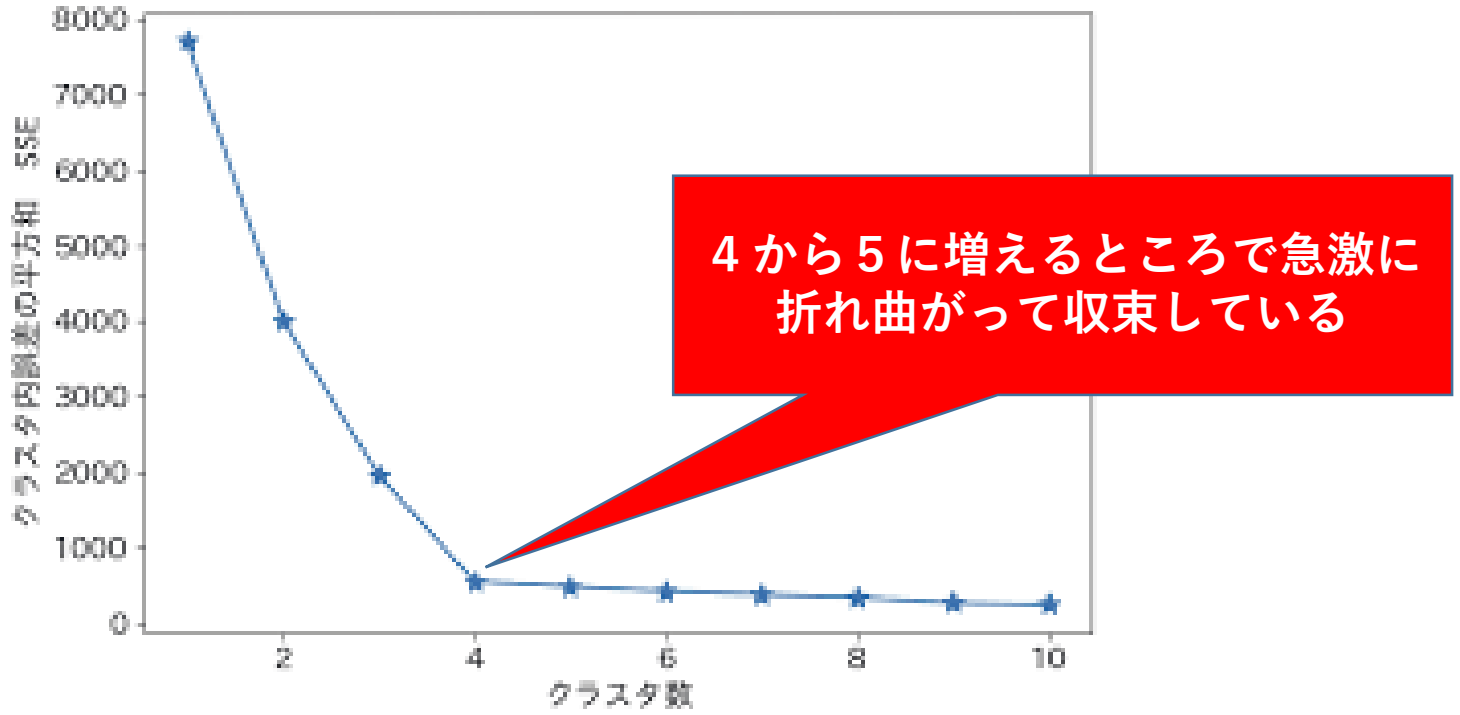
平均的なクラスタ

標準化加工をしたデータをクラスタリングしているのでマイナスの値になることもある。
標準化後データは平均値が必ず0になるので、マイナスは、全体平均よりも低いことを意味する。

１５．４，２ クラスタ数の決定～エルボー法 P551～P556

エルボー法：最適なクラスタ数を決定する方法 → k-means法による結果をもとに、クラスタ内誤差平方和（SSE）を計算する

最適なクラスタ数：クラスタ数とSSEの変化を折れ線グラフで表現する



コード15-12 クラスタ数２～３０でSSEを調べる

```
# SSEを保存する空リスト
sse_list = []
# クラスタ数2～30でSSEを調べる
for n in range(2, 31):
    # KMeans法モデル生成
    model = KMeans(n_clusters = n, random_state = 0)
    # 学習
    model.fit(sc_df)
    # SSEの計算
    sse = model.inertia_
    # SSEをリストsse_listに追加
    sse_list.append(sse)
sse_list
```

実行結果

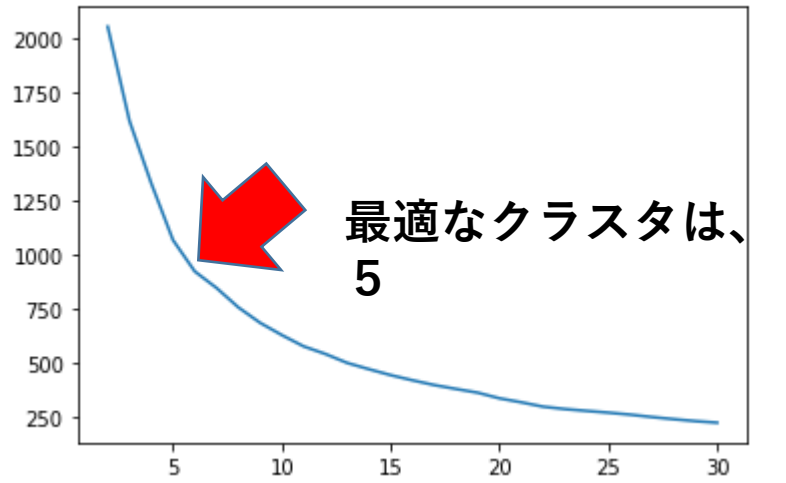
```
[2056.4923628163815,
 1619.952782172456,
 1331.5862623916175,
                :
                :
 240.34634808250857,
  231.32527278823156,
  224.09122126073785]
```

コード15-13 折れ線グラフを描画する

```
# リストをシリーズに変換する
se = pd.Series(sse_list)
# range関数で2～30の整数列を作る
num = range(2, 31)
# シリーズのインデックスを変更
se.index = num
# シリーズの値を基に折れ線グラフの表示
se.plot(kind = 'line')
```

実行結果

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b8d6651a58>
```



コード15-14 結果をCSVファイルに書き出す

```
# クラスタ数 5 でKMeansモデルを作成
model = KMeans(n_clusters = 5, random_state = 0)
# 学習
model.fit(sc_df)
# クラスタリングの結果を追加
sc_df['cluster'] = model.labels_
# データフレームをCSVファイルに変換して保存
sc_df.to_csv('clustered_Wholesale.csv', index = False)
```