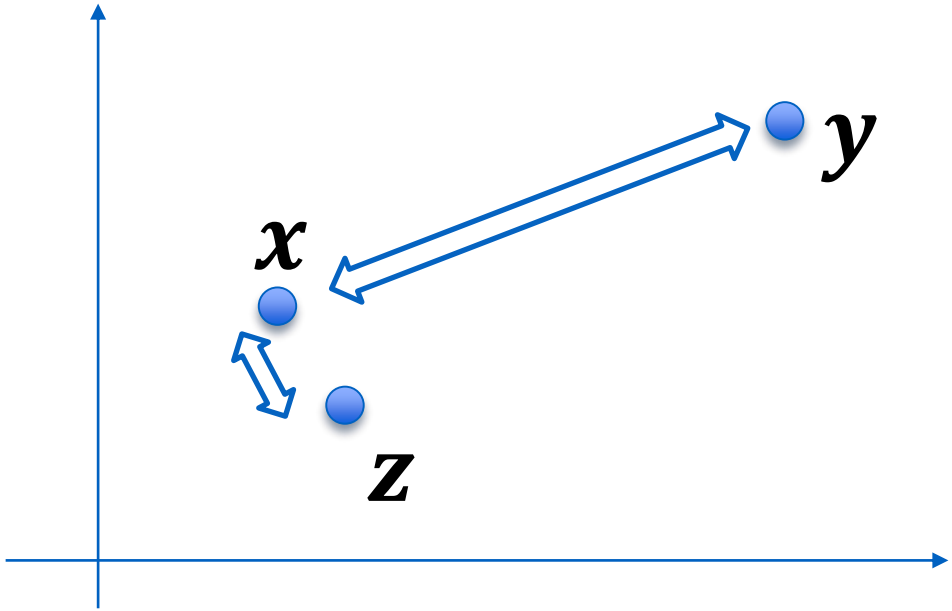
「距離とクラスタリング」

九州大学 大学院システム情報科学研究院
情報知能工学部門
データサイエンス実践特別講座
備瀬竜馬, Diego Thomas, 正井克俊

データ（ベクトル）間の距離、類似度

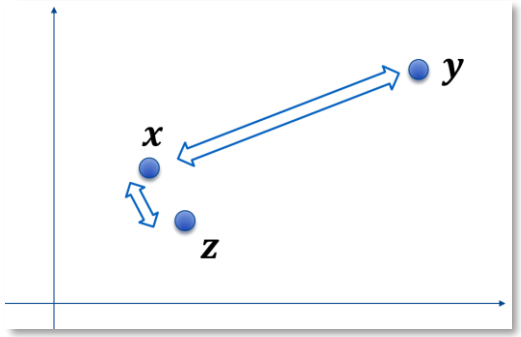
データ解析における「距離」とは？

- データ解析における「距離」
  - 要するにデータ間の類似度　（似てない具合）
  - 距離が小さい2データは「似ている」
  - 単位がある場合もない場合も



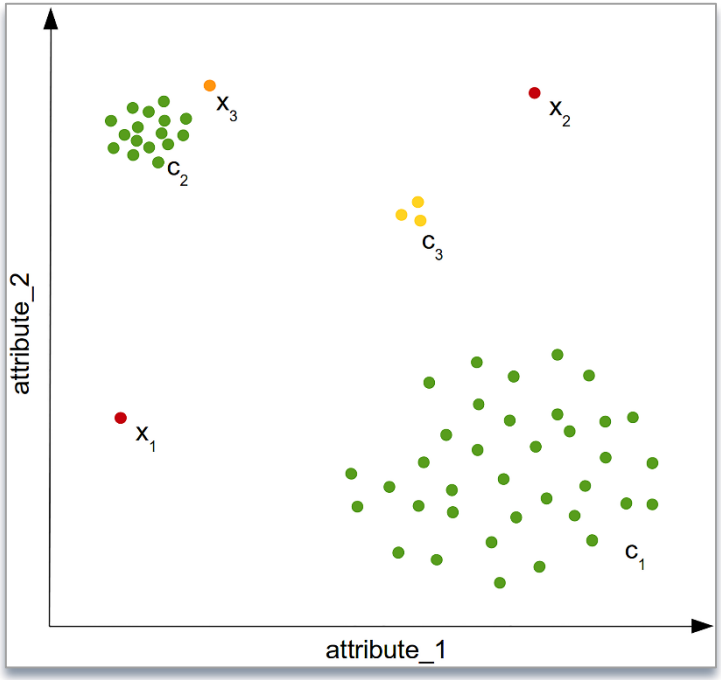
距離がわかると何に使えるか？
実は超便利！

- データ間の「類似度」を定量的に比較できる
  - 「xとyは全然違う/結構似ている」「xとyは28ぐらい違う」
  - 「xにとっては，yよりもzのほうが似ている」



- データ集合のグルーピングができる
  - 似たものとどおしでグループを作る

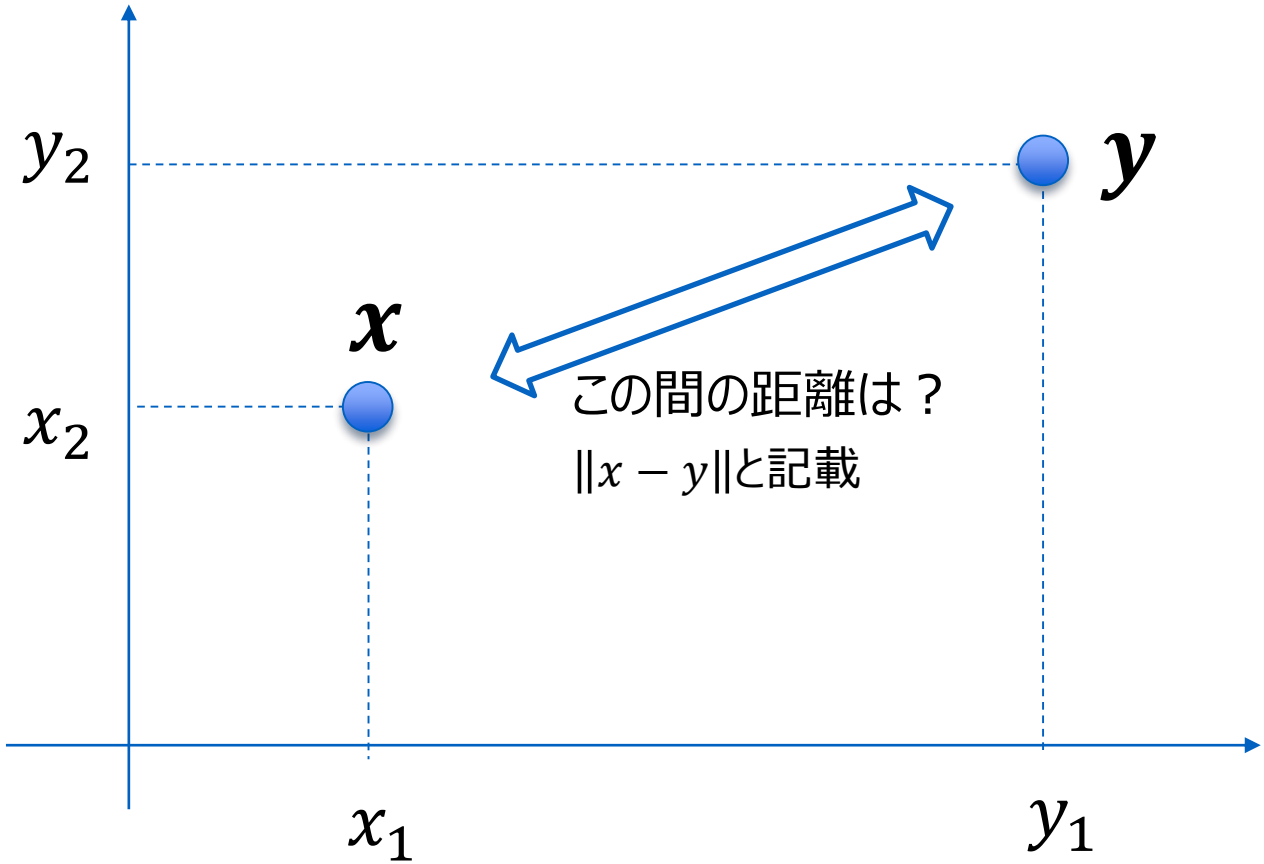
- データの異常度が測れる
  - 他に似たデータがたくさんあれば正常，一つもなければ異常



[Goldstein, Uchida, PLoSONE, 2016]

ユークリッド距離（1）

- 地図上の2点　$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$



ユークリッド距離 (2)

- 2次元の場合

$\boldsymbol{x}$と$\boldsymbol{y}$の距離の二乗 $=$

$$\begin{array}{cc} \boldsymbol{x} & \boldsymbol{y} \\ \downarrow & \downarrow \\ \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} & \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \end{array}$$

要素の差の二乗

$+$

要素の差の二乗

- 3次元の場合

$\boldsymbol{x}$と$\boldsymbol{y}$の距離の二乗 $=$

$$\begin{array}{cc} \boldsymbol{x} & \boldsymbol{y} \\ \downarrow & \downarrow \\ \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} & \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \end{array}$$

要素の差の二乗

$+$

要素の差の二乗

$+$

要素の差の二乗

ユークリッド距離（3）

- D次元の場合

$\boldsymbol{x}$と$\boldsymbol{y}$の距離の二乗 $=$

$$\begin{array}{cc} \boldsymbol{x} & \boldsymbol{y} \\ \downarrow & \downarrow \\ \begin{pmatrix} x_1 \\ \vdots \\ x_D \end{pmatrix} & \begin{pmatrix} y_1 \\ \vdots \\ y_D \end{pmatrix} \end{array}$$

要素の差の二乗

$+$

$\vdots$

$+$

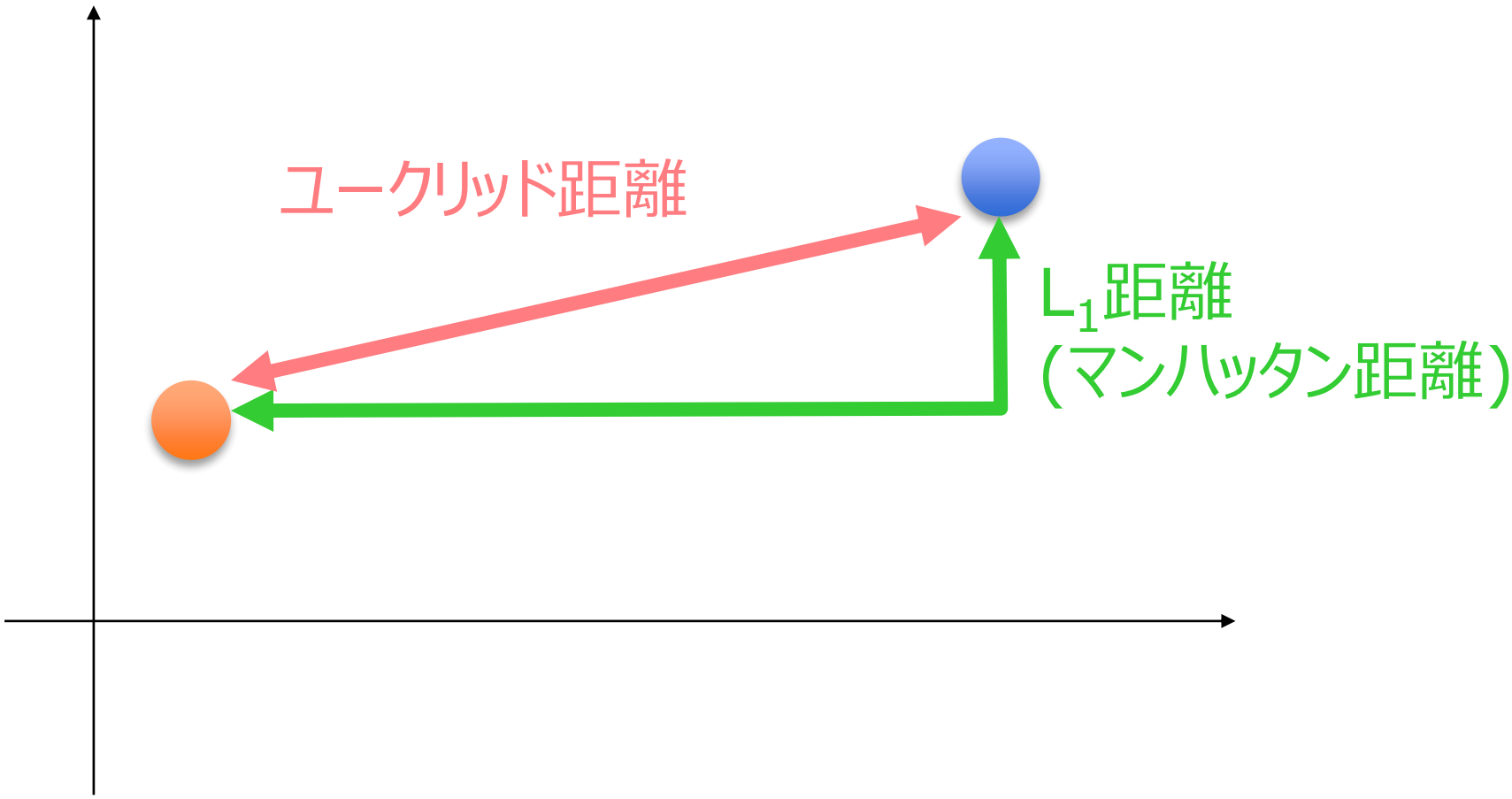
要素の差の二乗

というわけで，何次元ベクトルでも距離は計算可能

もちろん1次元ベクトル(数値)間の距離も計算可能

$$(x_1 - y_1)^2$$

マンハッタン距離



マンハッタン？

- 斜めには行けない街
  - 平安京距離
  - 平城京距離
  - 札幌距離

  でもいいかも

- 「市街地距離」と呼ばれることも

Google map

マンハッタン距離:プログラミング（7）

- N次元の場合

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix}, \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_3 \end{pmatrix}$$

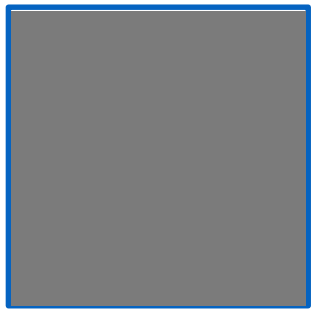
$$\boldsymbol{x}\text{と}\boldsymbol{y}\text{の距離} = \underbrace{|x_1 - y_1|} + \cdots + \underbrace{|x_N - y_N|}$$

各要素の差の絶対値の和

np.linalg.norm(x-y, ord=1) で計算できる
※ ユークリッド距離が「L2」ノルム、
マンハッタン距離が「L1」ノルムとも呼ばれるため

ユークリッド距離で測る類似度

256×256

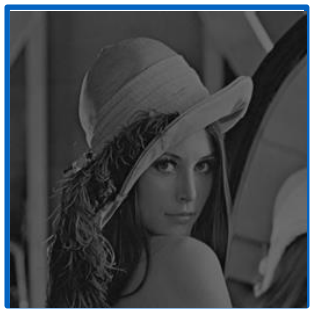


65536次元
ベクトル $\boldsymbol{z}$

256×256

65536次元
ベクトル $\boldsymbol{x}$

256×256



65536次元
ベクトル $\boldsymbol{y}$

画像間距離
$\|\boldsymbol{x} - \boldsymbol{y}\|$

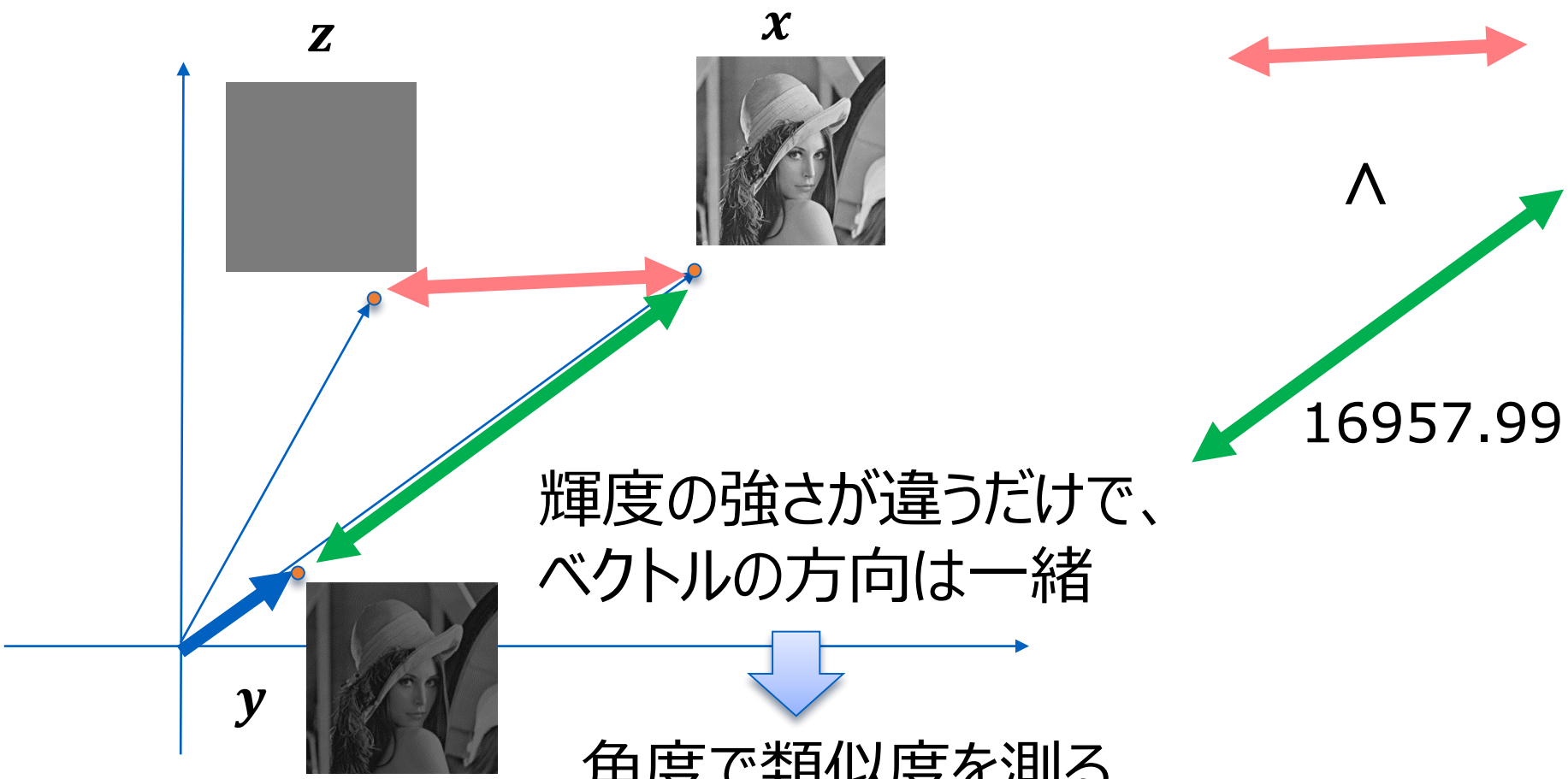
画像間距離
$\|\boldsymbol{x} - \boldsymbol{y}\|$

12249.03 $<$ 16957.99

人の目には $\boldsymbol{z}$ より $\boldsymbol{y}$ の方が $\boldsymbol{x}$ に類似しているが、
ユークリッド距離だと、 $\boldsymbol{z}$ の方が $\boldsymbol{x}$ からの距離が近い

他の類似度

イメージ図



輝度の強さが違うだけで、
ベクトルの方向は一緒

角度で類似度を測る

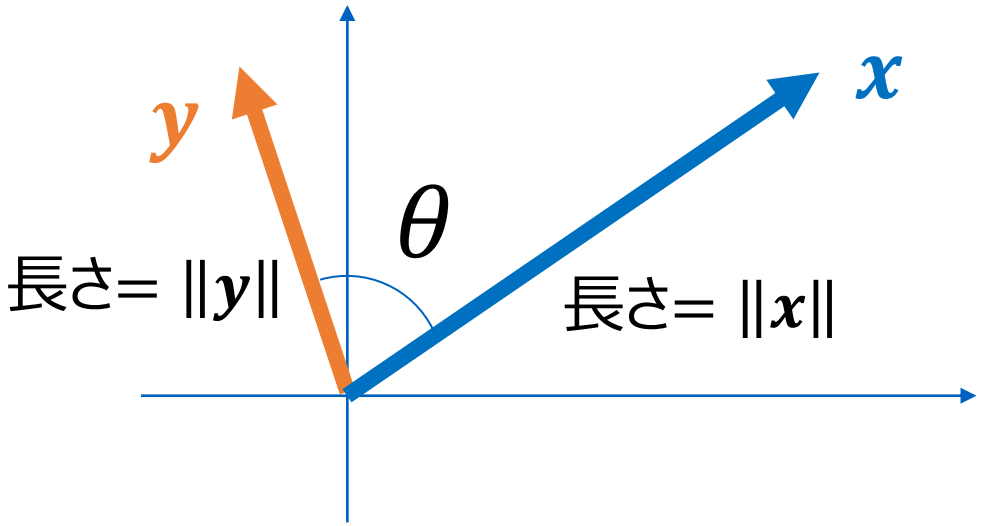
内積

- 内積は角度を考慮した類似度

$$\boldsymbol{x} \cdot \boldsymbol{y} = \|\boldsymbol{x}\| \|\boldsymbol{y}\| \cos\theta$$

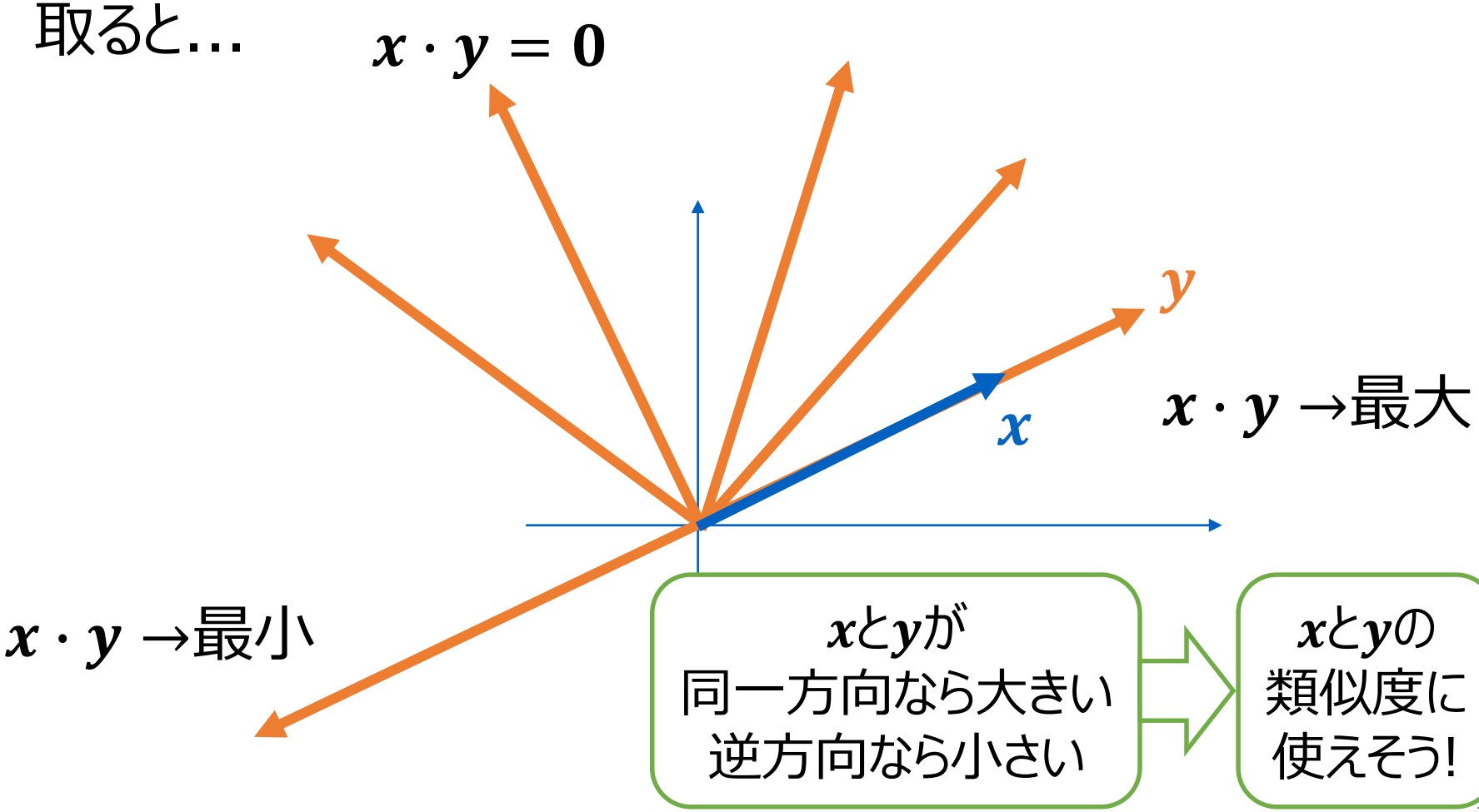
$$\cos\theta = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\| \|\boldsymbol{y}\|}$$

各成分２乗して足して
ルートとったやつ



内積

- 一定の大きさのベクトル$\boldsymbol{y}$を回転させながら$\boldsymbol{x}$と内積を取ると…



正規化相関（コサイン類似度）

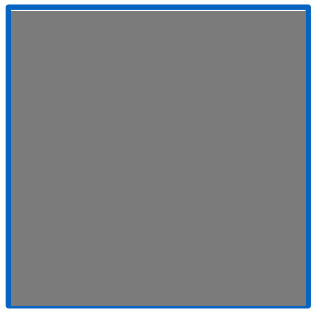
- 正規化相関

$$\cos\theta = \frac{\boldsymbol{x}\cdot\boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|}$$

```
import numpy as np
# xとyの定義
x = np.array([60,180])
y = np.array([60,150])

#コサイン類似度
D = np.dot(x,y) # 内積
xd = np.linalg.norm(x) #||x||
yd = np.linalg.norm(y) #||y||
C = D/(xd*yd) # コサイン類似度

print(C) # 表示
```

正規化相関で測る類似度

256×256

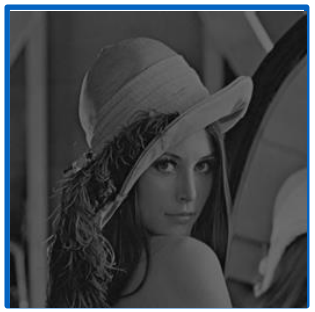


65536次元
ベクトル $\boldsymbol{z}$

256×256

65536次元
ベクトル $\boldsymbol{x}$

256×256



65536次元
ベクトル $\boldsymbol{y}$

画像間
コサイン類似度

画像間距離
コサイン類似度

0.932 < 1.0

まとめ:
numpy を使えば簡単

```
a=np.array([1,2,3,4,5])
b=np.array([2,2,3,3,4])
```

- ユークリッド距離

```
np.linalg.norm(a-b)
1.7320508075688772
```

- マンハッタン距離

```
np.linalg.norm(a-b, ord=1)
3.0
```

- 内積

```
np.dot(a,b)
47
```

- コサイン

```
np.dot(a,b)/(np.linalg.norm(a)*np.linalg.norm(b))
0.9778941948273628
```

練習１：2つのデータ間の各類似度：ユークリッド距離、マンハッタン距離、内積、コサイン類似度を求めよう

$\boldsymbol{x} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}, \boldsymbol{y} = \begin{pmatrix} 6 \\ 1 \end{pmatrix}$のとき 各類似度は？

$\boldsymbol{x} = \begin{pmatrix} 3 \\ 5 \\ 2 \end{pmatrix}, \boldsymbol{y} = \begin{pmatrix} 6 \\ 1 \\ 2 \end{pmatrix}$のとき 各類似度は？

距離を使ったちょっと高度な可視化（１）

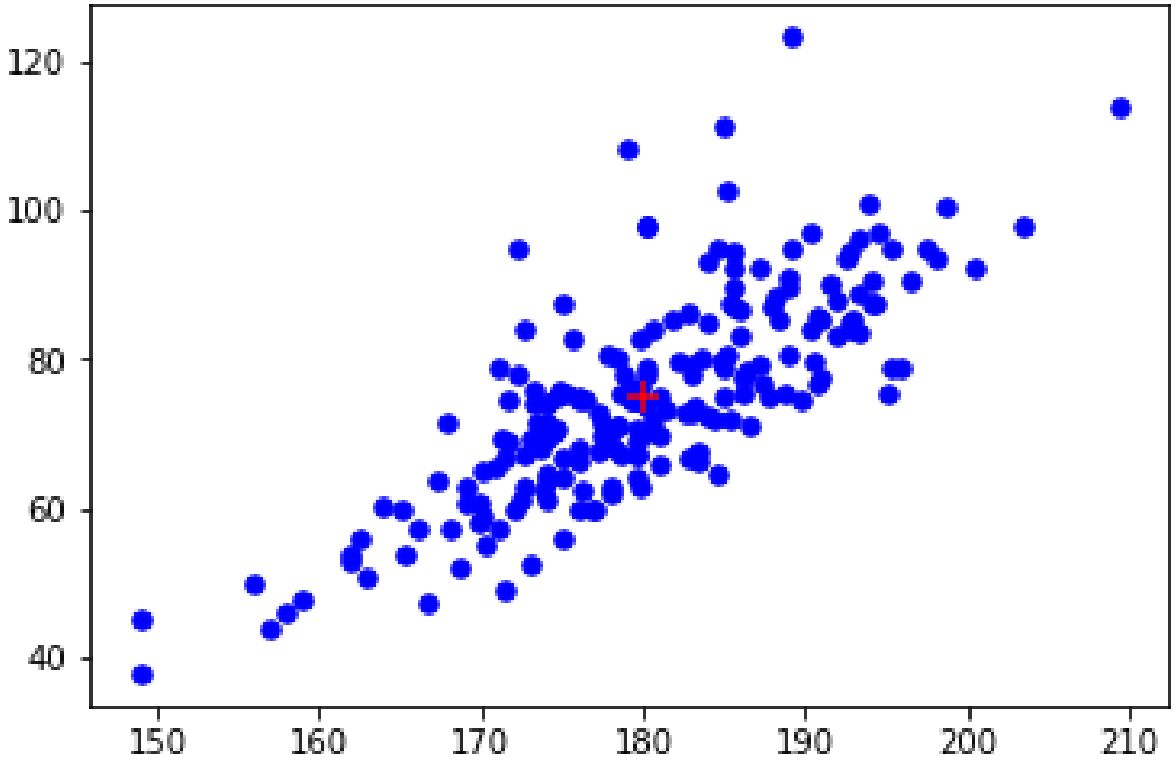
- csvファイル” height_weight.csv”を読み込み、「横軸：身長、縦軸：体重」としてプロットして、分布の形状を確かめたい。ある点Aを別の色で表示してみたい。

```
import pandas as pd
f = pd.read_csv("./height_weight.csv")
vlist = np.array(f)

# 分布の可視化

A = np.array([180,75])
f.plot(kind='scatter', x='Ht',y='Wt', c='blue',marker='o',s=30)
plt.scatter(A[0], A[1],c='red',marker='+',s=100)
```

こうすればpandasで読み込んだデータが一発でnumpy のデータセット（複数のベクトル）にできます

色　　マーカーの形　　マーカーのサイズ

可視化結果



距離を使ったちょっと高度な可視化（２）

- A=(180,75)からの各点へのユークリッド距離をそれぞれ算出し、最も距離が遠い順に並び替えて、距離に応じて色を変えて表示

```
# compute distances
dlist = []
for row in vlist:
    d = np.linalg.norm(A-row[[2,3]])
    dlist.append(d)

# 距離が大きい順に並び替え
sinds = np.argsort(dlist)[::-1]
```

小さい順に並べたときのインデックスを返す関数

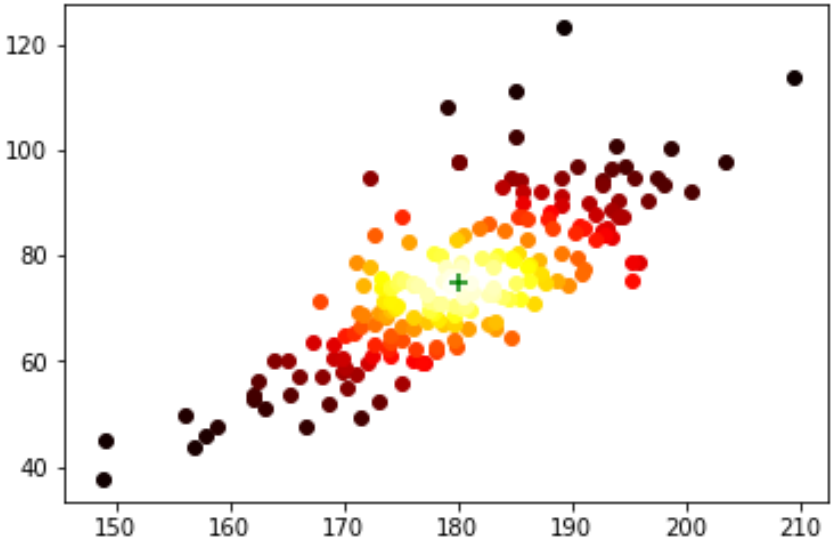
逆から並べる

例：np.argsort([2,1,3])=[1,0,2]

距離を使ったちょっと高度な可視化（２）

- 距離に応じて色を変えて表示

```
# visualize
import matplotlib.cm as cm
import matplotlib.pyplot as plt
count = 0
# 距離の近い順にプロット
for i in sinds:
    row = vlist[i]
   # 並び順に応じて色を決定
    c = cm.hot(count/len(vlist))
   # 決定した色を指定してプロット
    plt.scatter(row[2],row[3],color=c)
    count += 1
# 点Aの可視化
plt.scatter(A[0], A[1], c='green',
marker='+', s=60)
```



遠いほど赤い

K-Means

データのクラスタリング(分類)

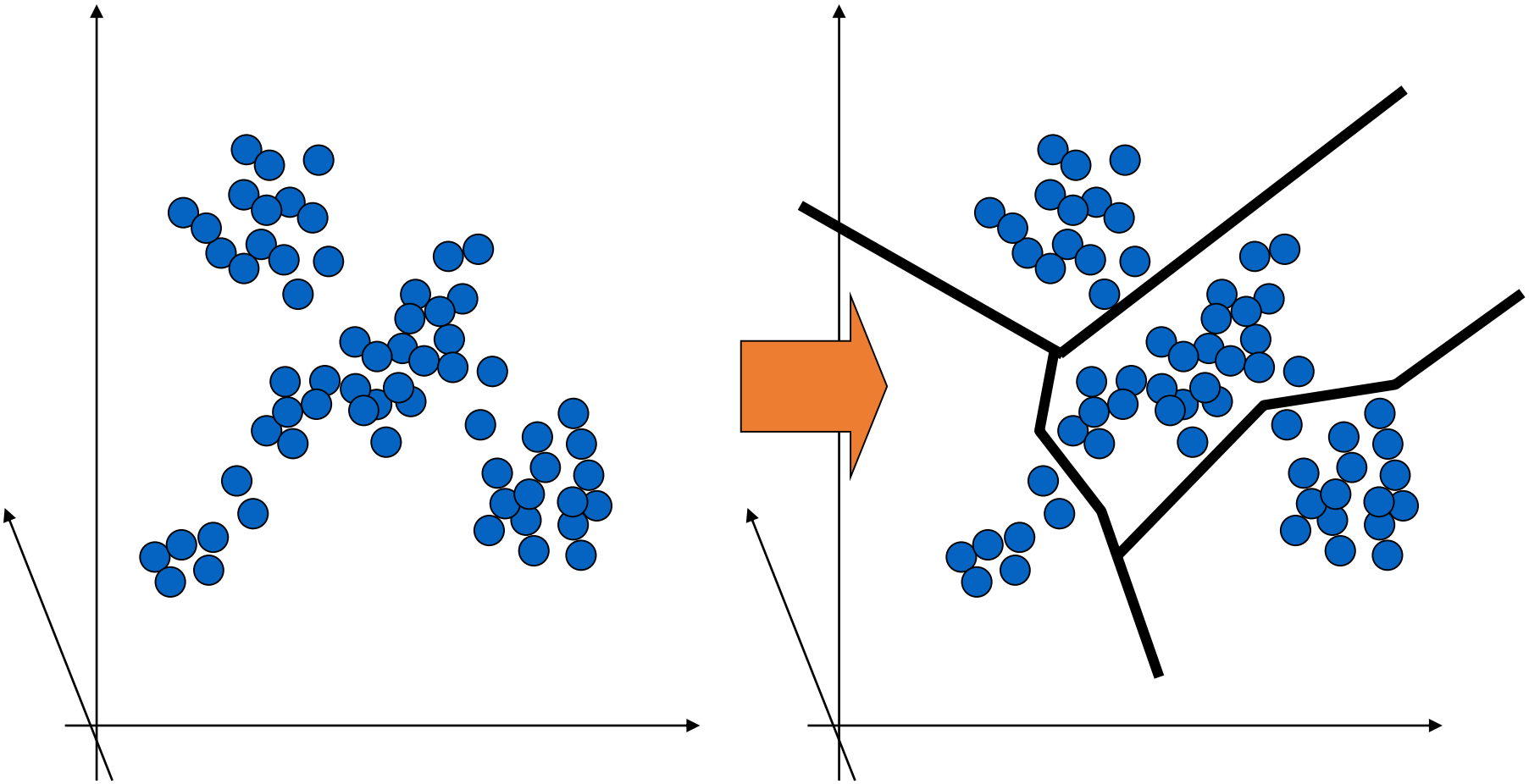
データ集合をグルーピングする

- 我々は「距離」や「類似度」を手に入れた

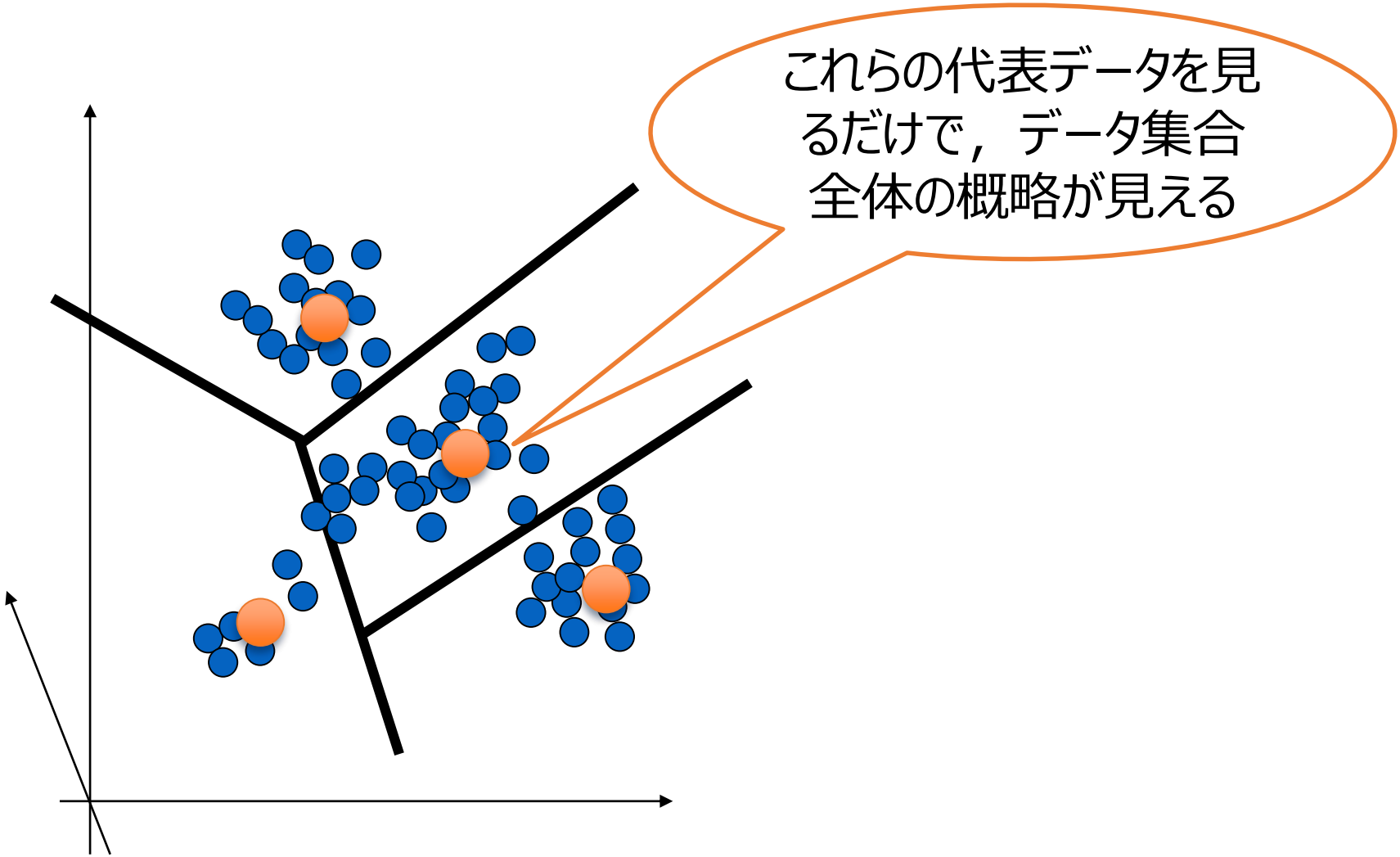
- 結果，「与えられたデータ集合」を「それぞれ似たデータからなる幾つかのグループに分ける」ことが可能に！

クラスタリング(clustering)＝
データの集合をいくつかの部分集合に分割する(グルーピング)

- 各部分集合＝「クラスタ」と呼ばれる

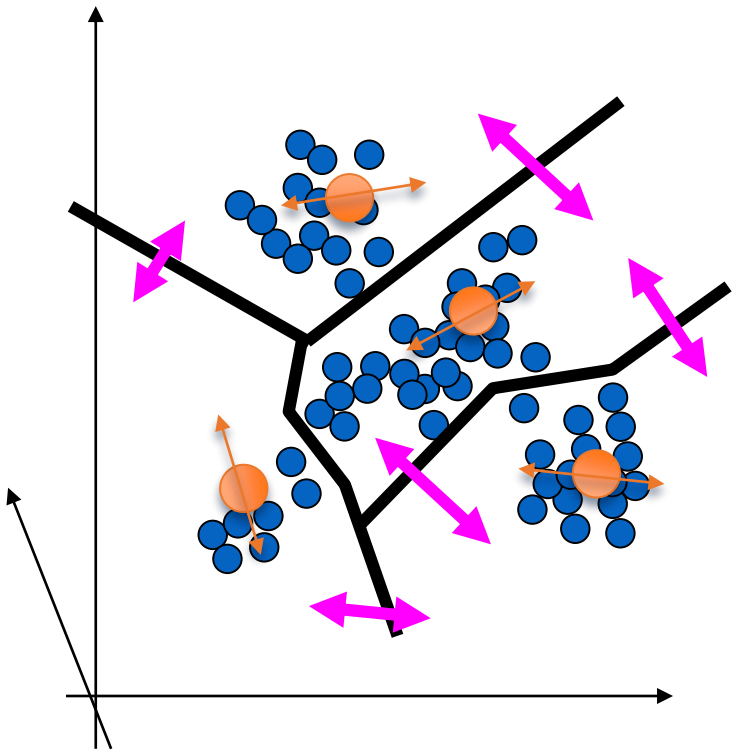


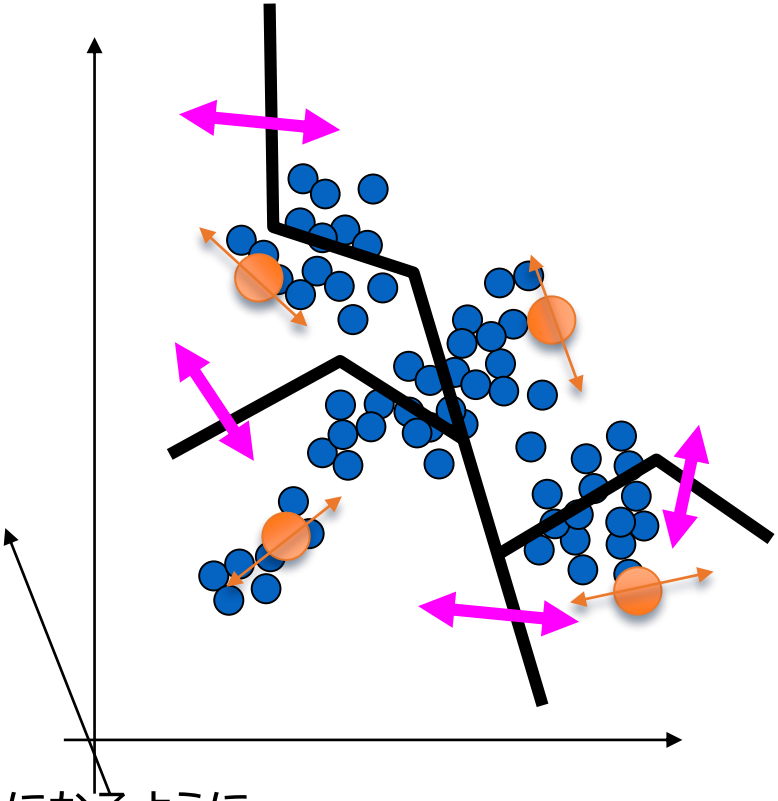
各クラスタから代表的なデータを選ぶと…



どういう分割がよいのか？

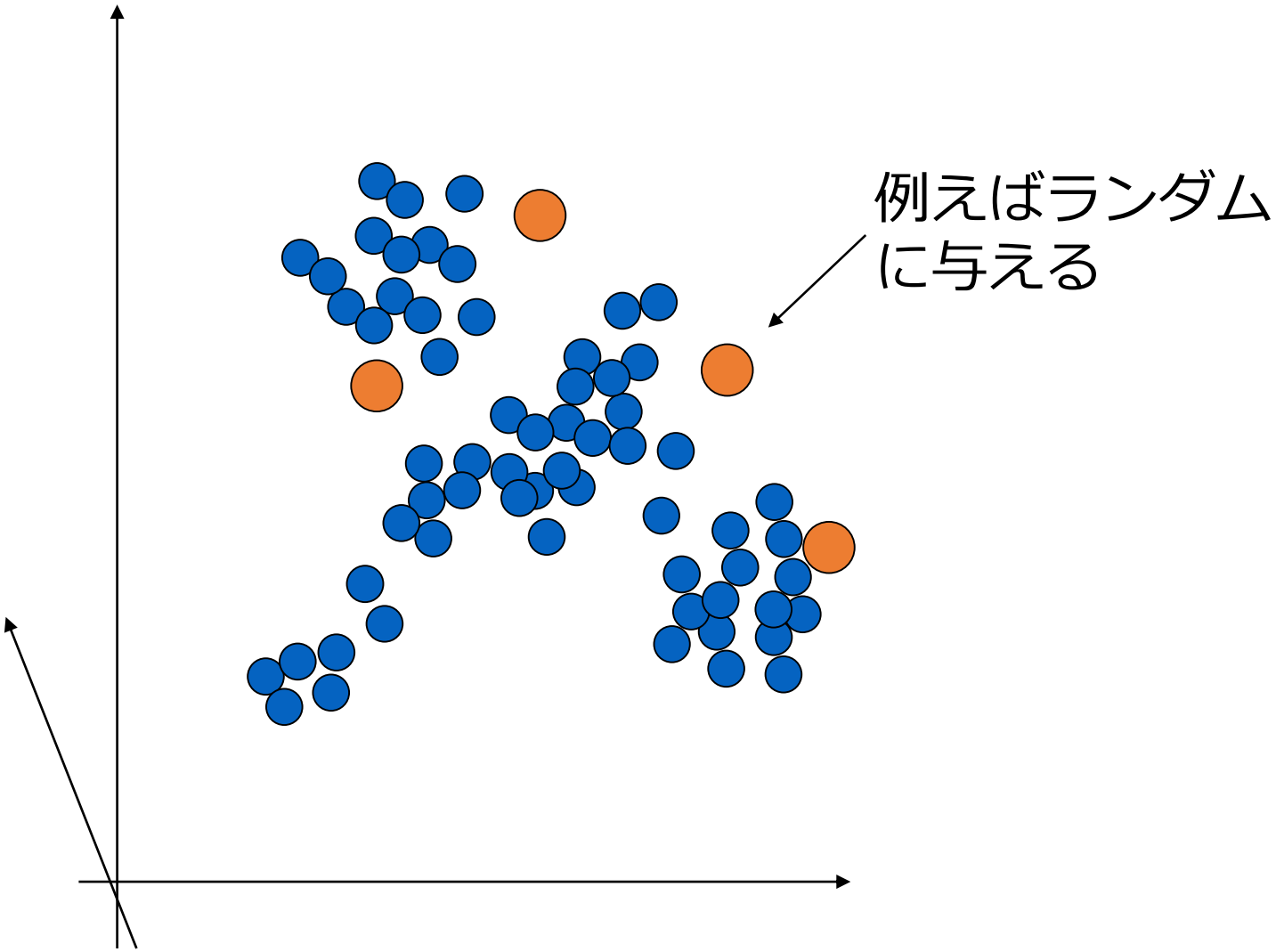
- N個のデータをK個に分割する方法はおよそK^N通り
  - 100個のデータを10分割→およそ10^{100}通り



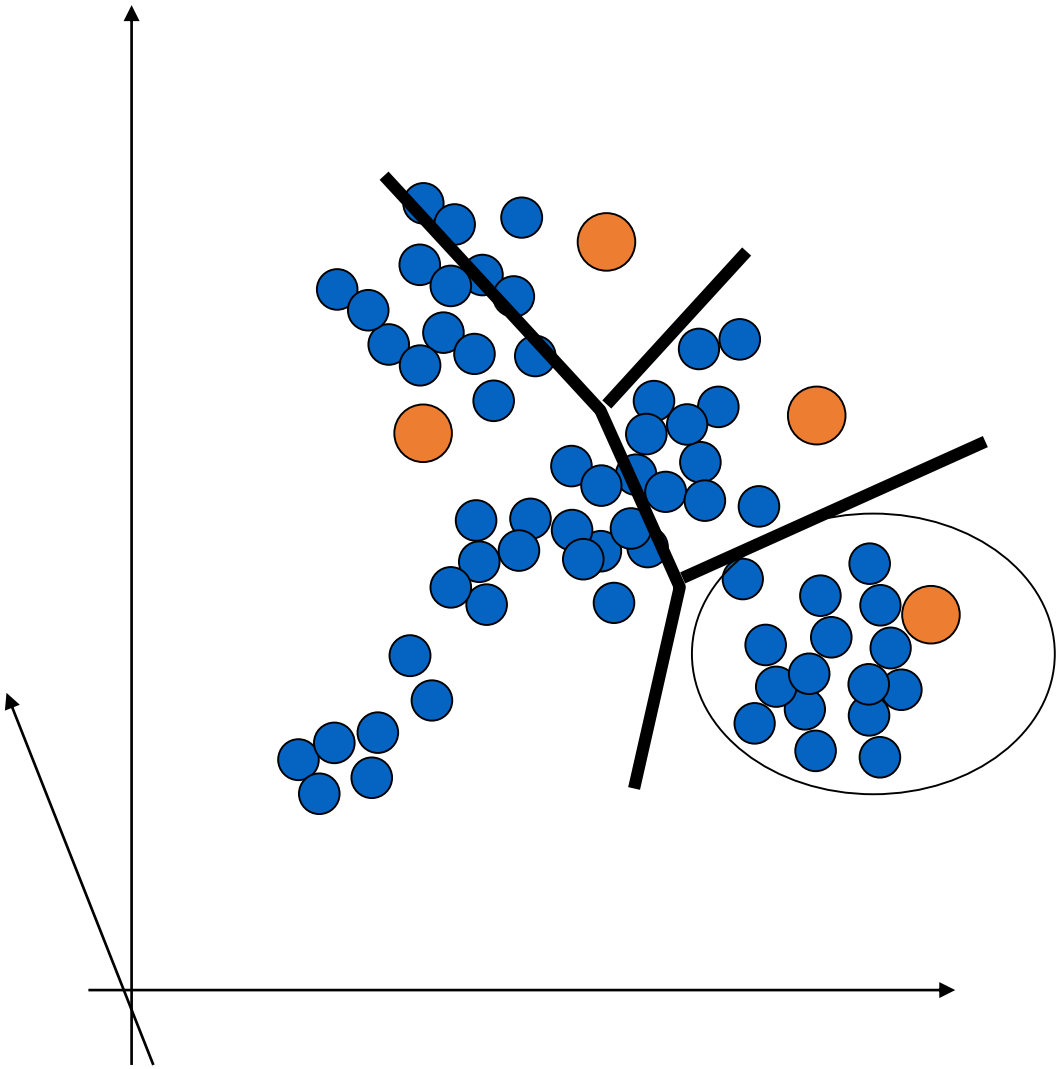


- 近くにあるデータが，なるべく同じ部分集合になるように

代表的なクラスタリング法：K-means法（0）初期代表パターン

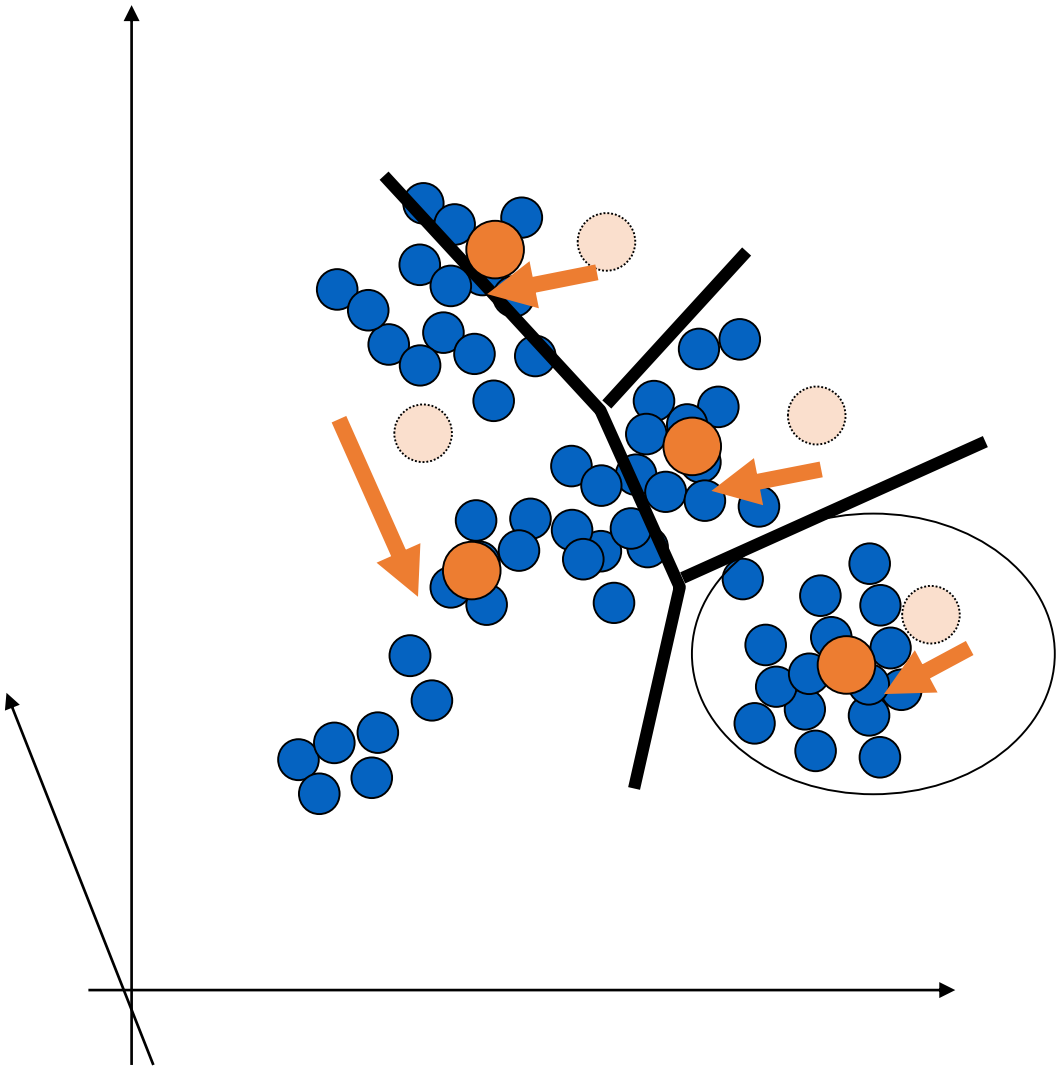


代表的なクラスタリング法：K-means法（1）学習パターン分割



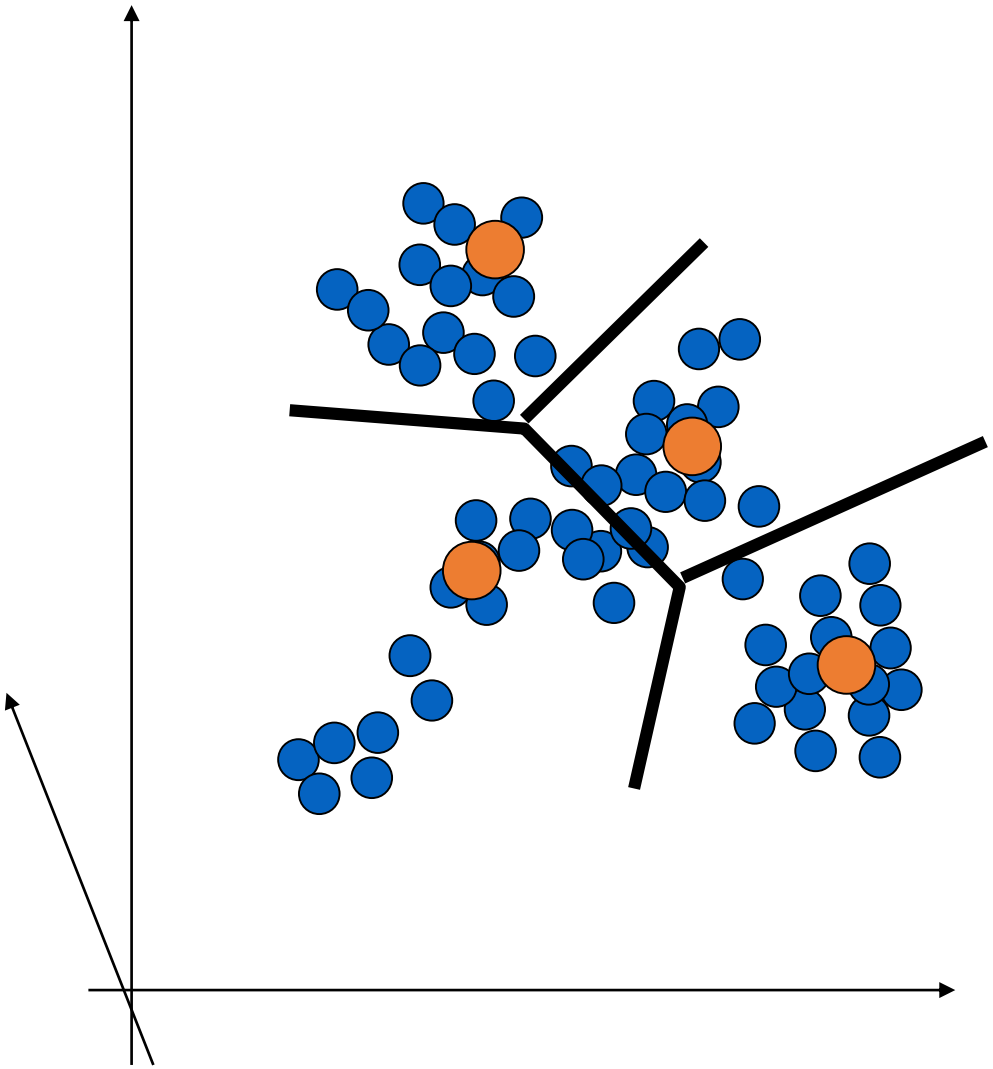
代表パターンの
ファンクラブ
（支持者集合）

代表的なクラスタリング法：K-means法（2）代表パターン更新

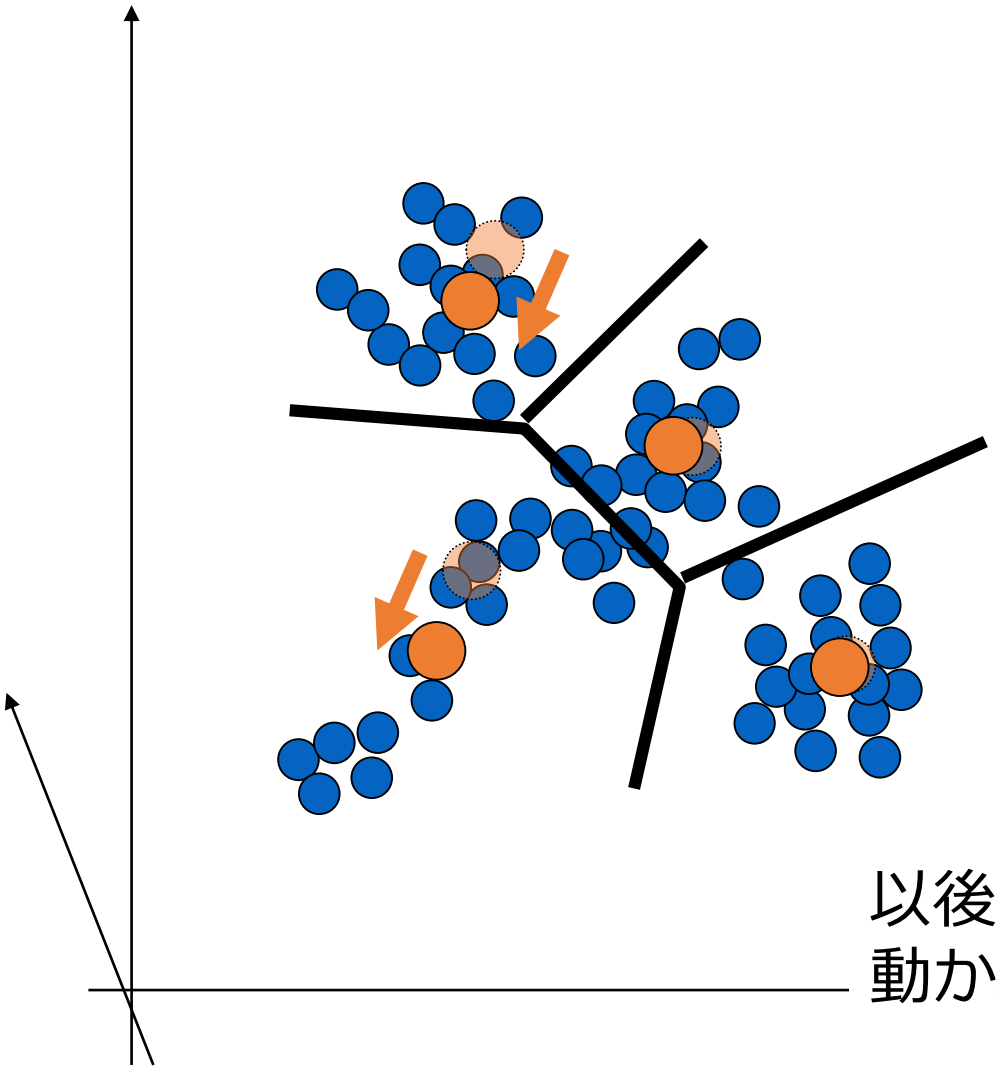


この中の学習パターンの平均に移動

代表的なクラスタリング法：K-means法（1）学習パターン分割



代表的なクラスタリング法：K-means法（2）代表パターン更新

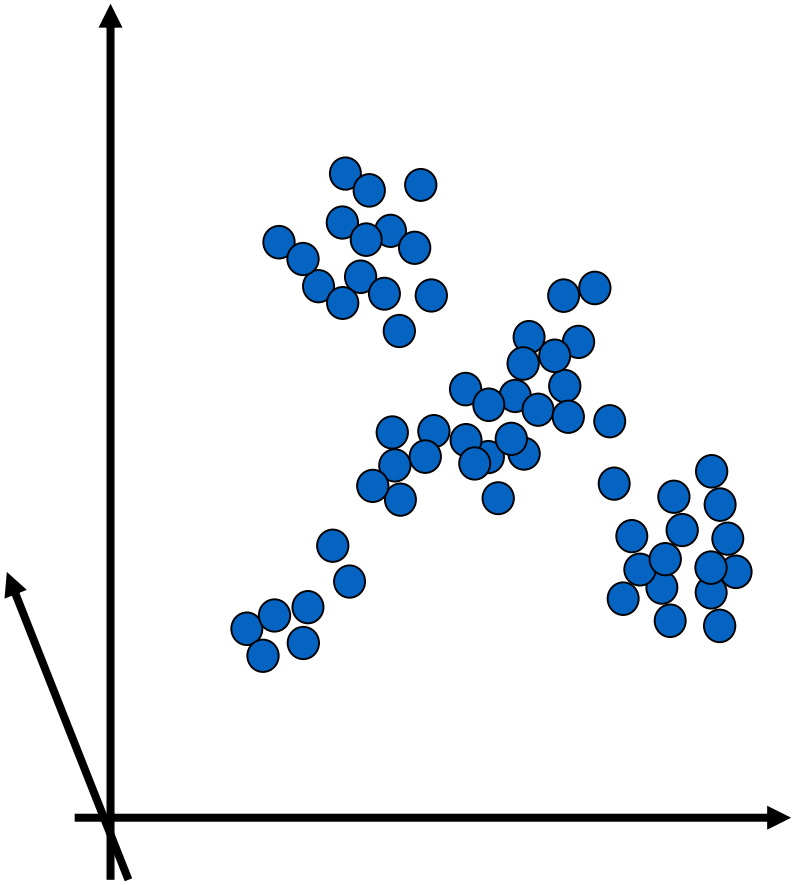


以後，代表パターンが動かなくなるまで反復

K-means：プログラミング

- ステップ0：
  - データ入力
  - クラスタ数 K の決定

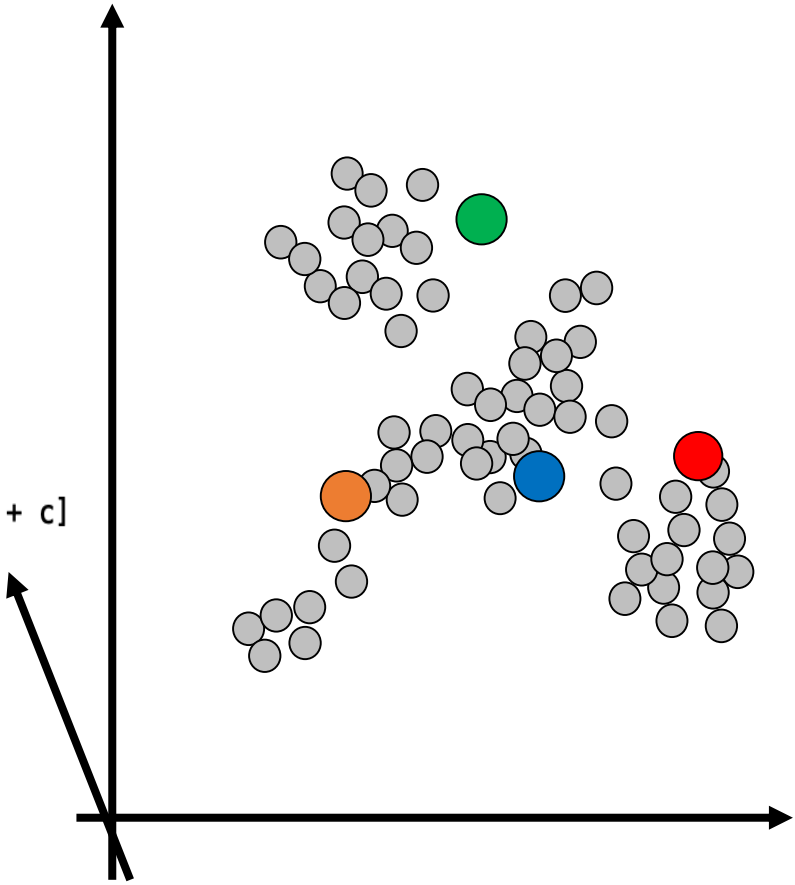
```
import pandas as pd
f = pd.read_csv('data.csv', header=None)
vlist = np.array(f)
# kの決定
K = 4;
```



K-means：プログラミング

- ステップ1：
  - 初期代表点の決定（ランダム）

```
# random値の範囲決定
a = np.min(vlist[:,0])
b = np.max(vlist[:,0])
c = np.min(vlist[:,1])
d = np.max(vlist[:,1])
#初期点(K個)をランダムに決定
CList = np.c_[(b-a)*np.random.rand(K) + a, (d-c)*np.random.rand(K) + c]
plt.scatter(vlist[:,0], vlist[:,1])
plt.scatter(CList[:,0], CList[:,1],c='blue',marker='+',s=100)
orgCList = CList
```



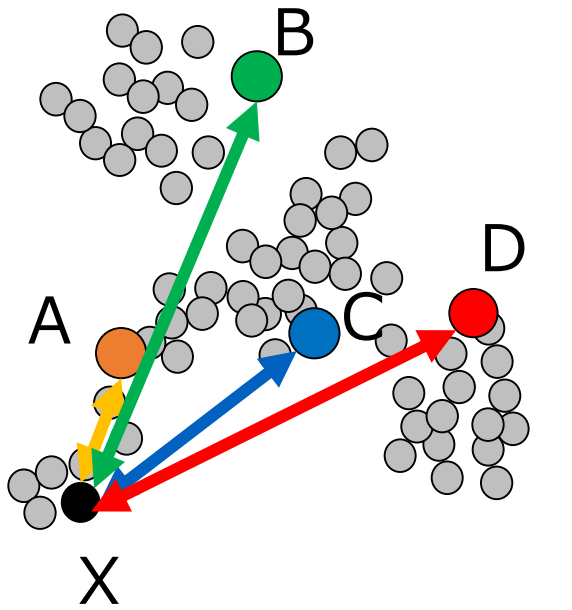
K-means：プログラミング

- ステップ１：
  - 代表点の支持者決定

⇒最も近い代表点を支持する

距離を活用！

  - ある点Xに着目
    - 各代表点までの距離を算出

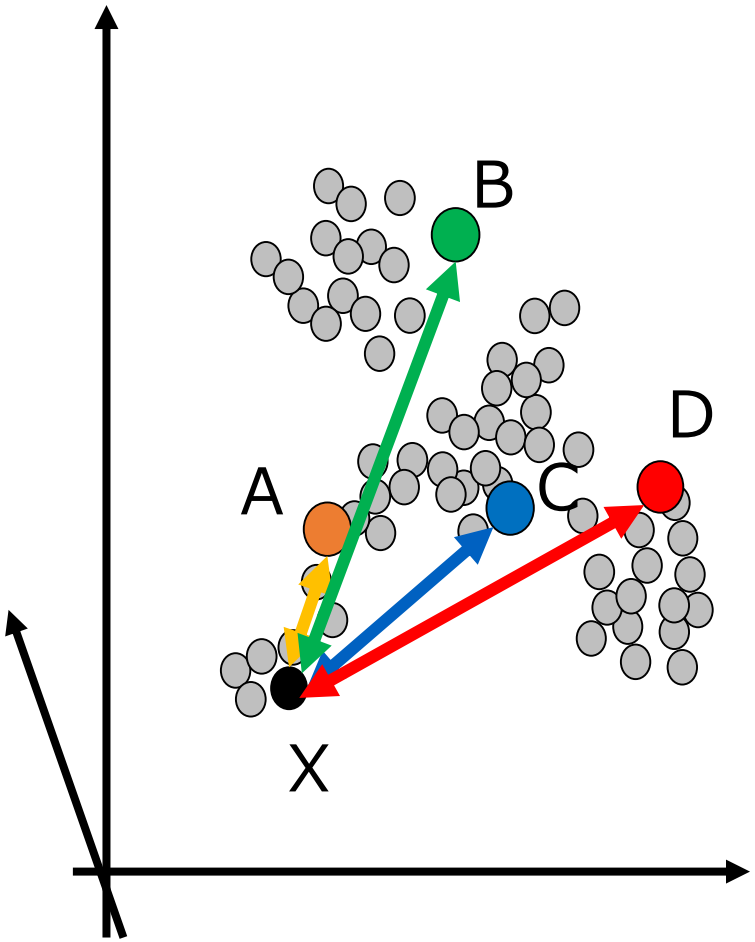



```
＃XとAの距離の計算
Ad = numpy.linalg.norm(X-A)
```

K-means：プログラミング

- ステップ１：
  - ある点Xに着目
  - 距離が最も近い代表点を決定

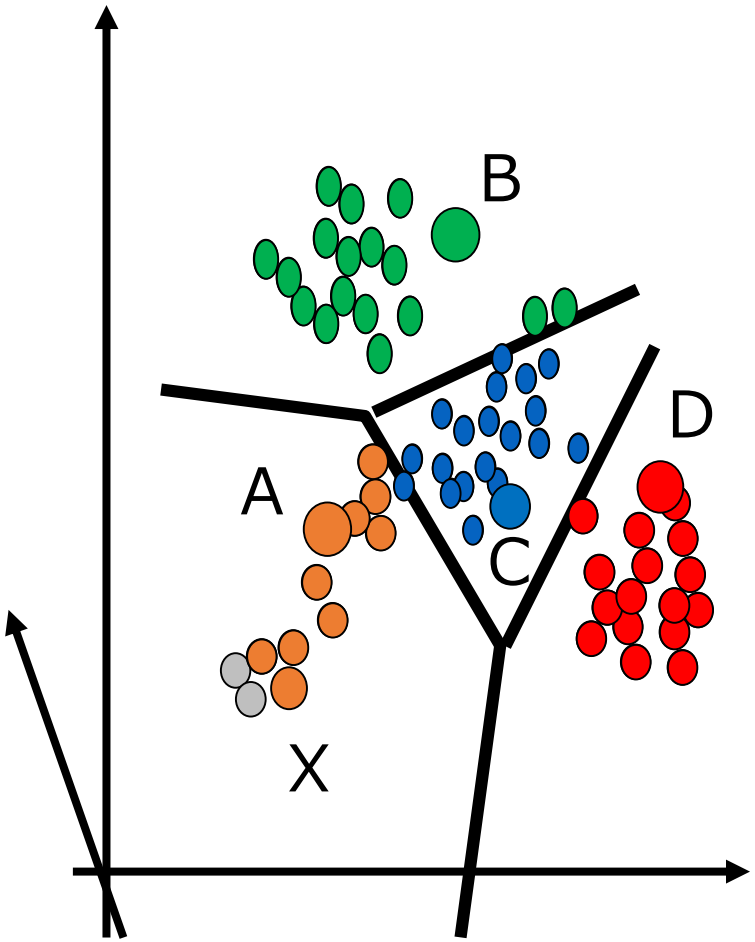
```
def nearest(X,CList):
    minD = float("inf");
    minId = 0
    for ii in range(0,len(CList)):
        C = CList[ii]
        d = np.linalg.norm(X-C)
        if d < minD:
            minD = d
            minId = ii
    return minId
```



K-means：プログラミング

- ステップ１：
  - 全ての点で代表点を決定

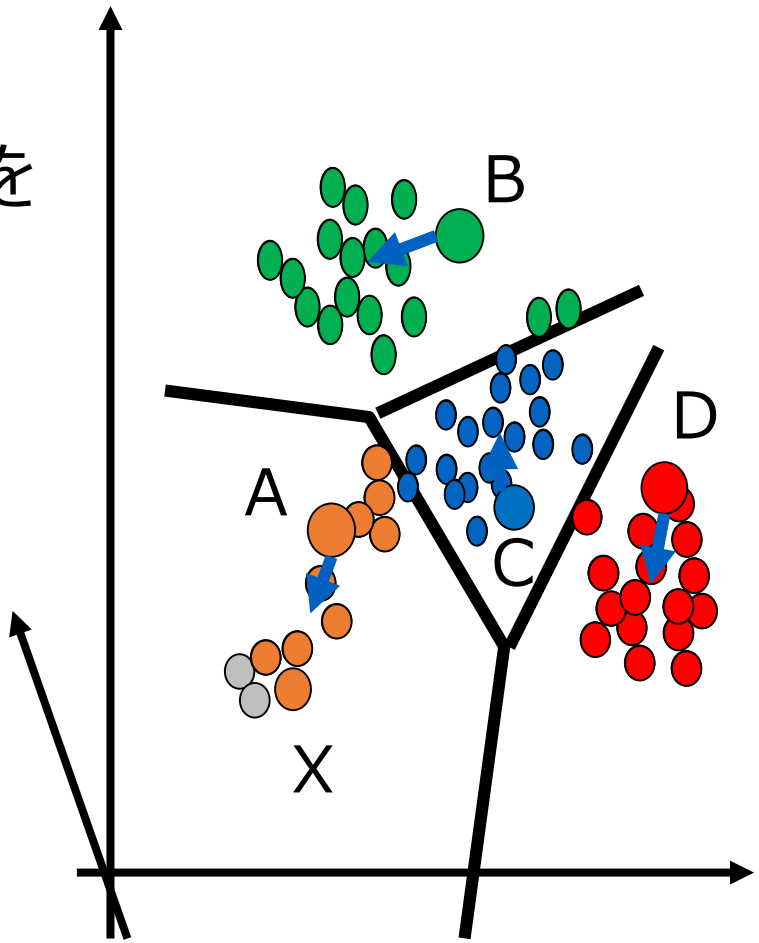
```
def selectCluster(vlist,CList):
    cIList = []
    for row in vlist:
        cIList.append(nearest(row,CList))
    return cIList
```



K-means：プログラミング

- ステップ2：代表点を更新
  - 各クラスタの支持者群の平均値を代表点とする。

```
def updataCenter(vlist,CIList,K):
    CList = []
    for k in range(K):
        Z = []
        for j in range(len(CIList)):
            if CIList[j] == k:
                Z.append(vlist[j])
        mc = np.mean(Z,0)
        CList.append(mc)
    return CList
```

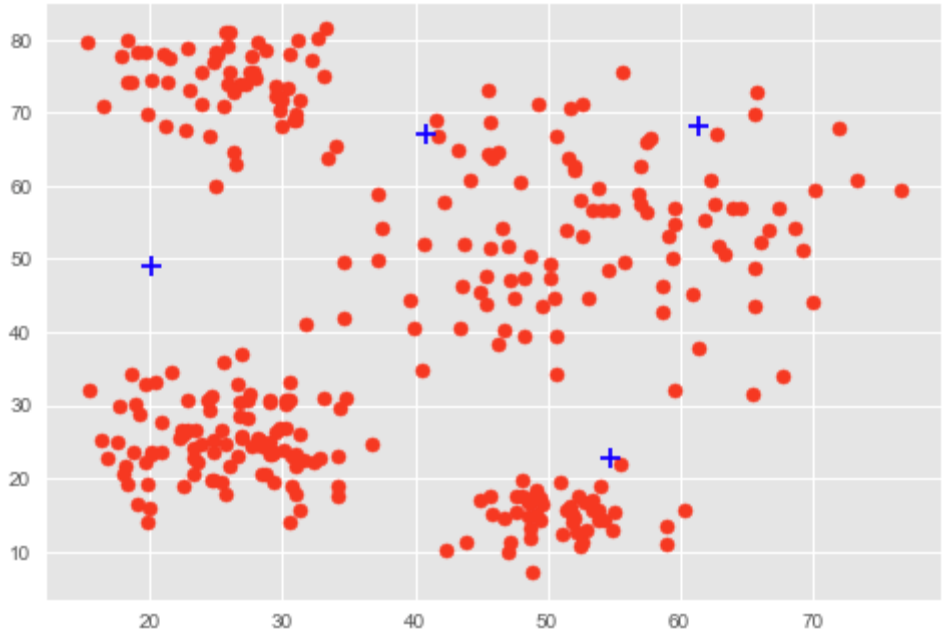


K-means：プログラミング

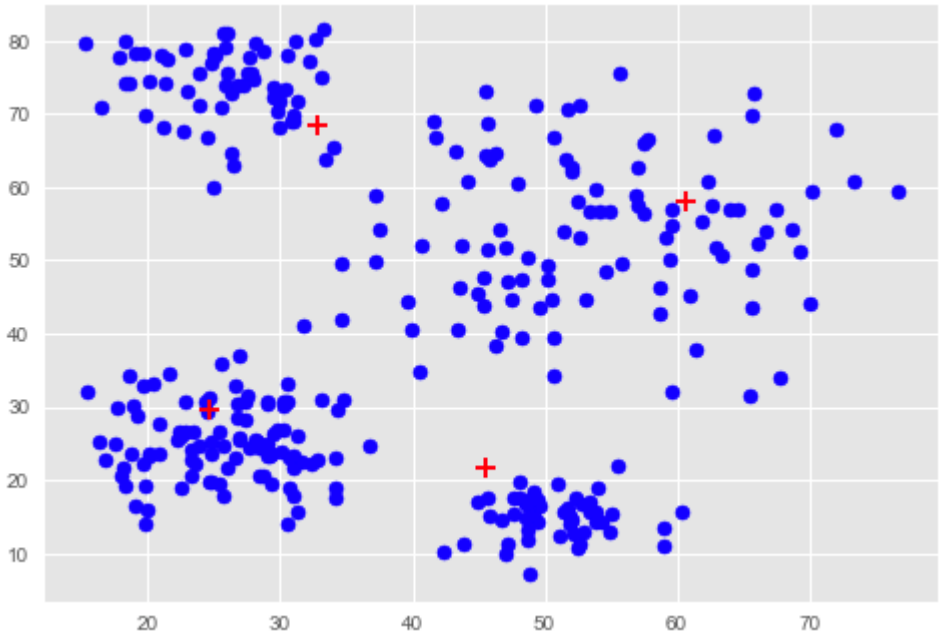
- アルゴリズム
  - ステップ0
  - 繰り返し
    - 代表点の選択
    - 代表点の更新
    - 終了チェック

```
Err = 0.01
maxittr = 100
itt = 0
while itt < maxittr:
    end_flag = True
    preCList = CList
    # 指示クラスタの決定
    CIList = selectCluster(vlist,CList)
    # クラスタ中心の更新
    CList = updataCenter(vlist,CIList,K)
    # 更新の際の移動距離の算出
    for j in range(len(CList)):
        cc = CList[j]
        pre = preCList[j]
        dd = np.linalg.norm(cc-pre)
        print(dd)
        if dd > Err:
            end_flag = False
    itt += 1
    # 可視化
    plt.scatter(vlist[:,0], vlist[:,1], c='blue')
    for k in range(len(CList)):
        mc = CList[k]
        plt.scatter(mc[0], mc[1],c='red',marker='+',s=100)
    plt.show()
    plt.pause(1)
    if end_flag == True:
        break
```

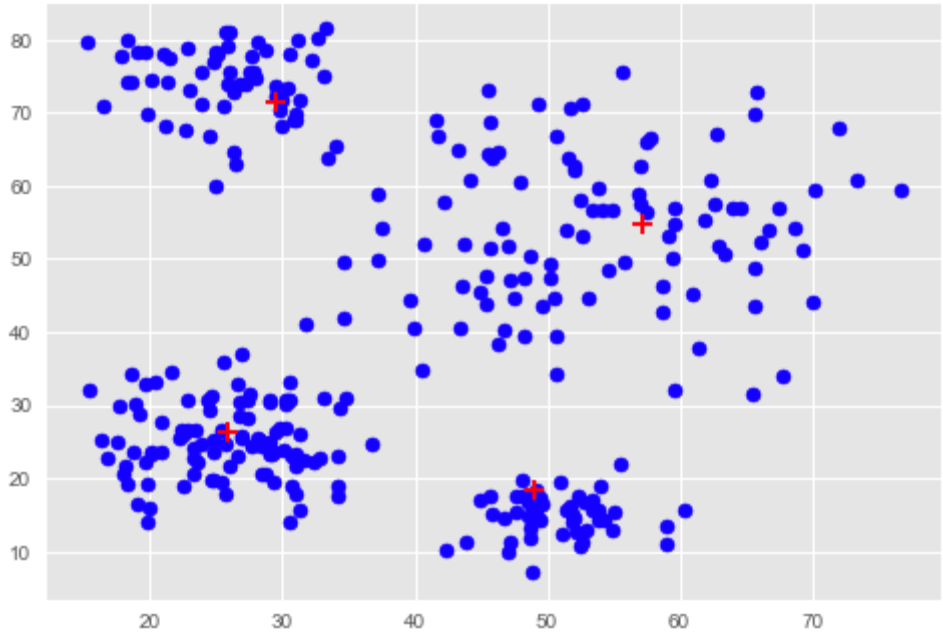
K-meansクラスタリング



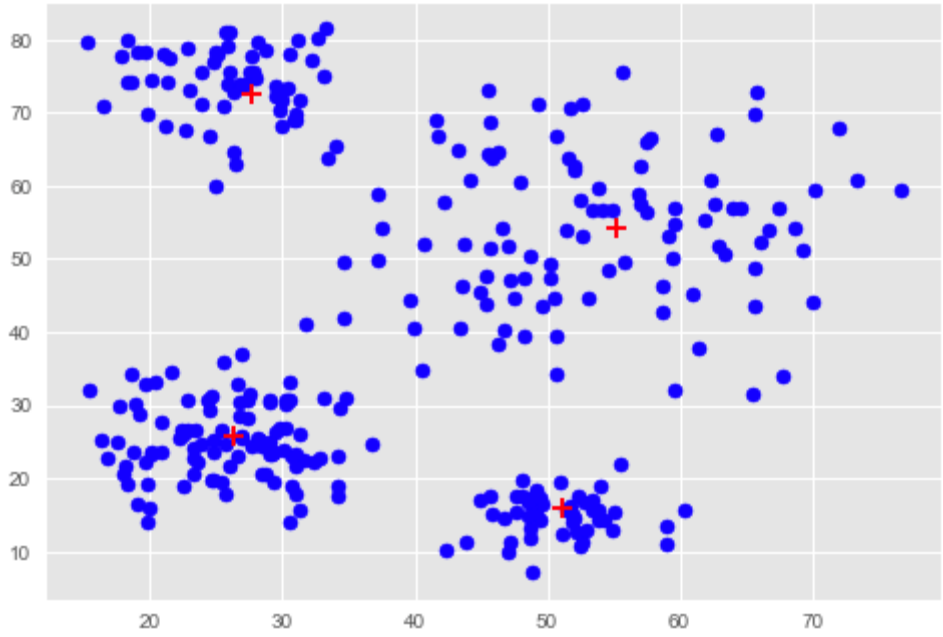
K-meansクラスタリング



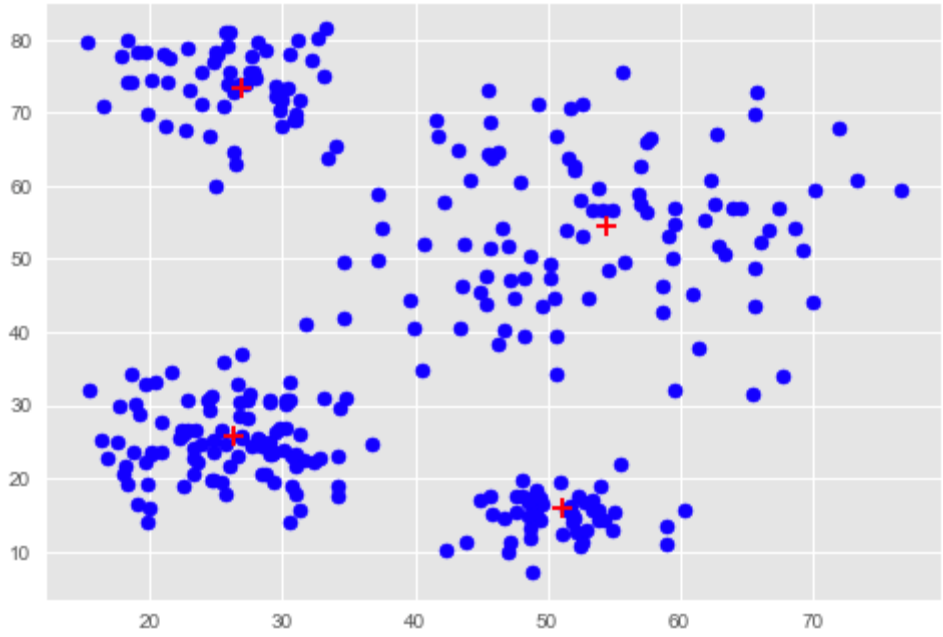
K-meansクラスタリング



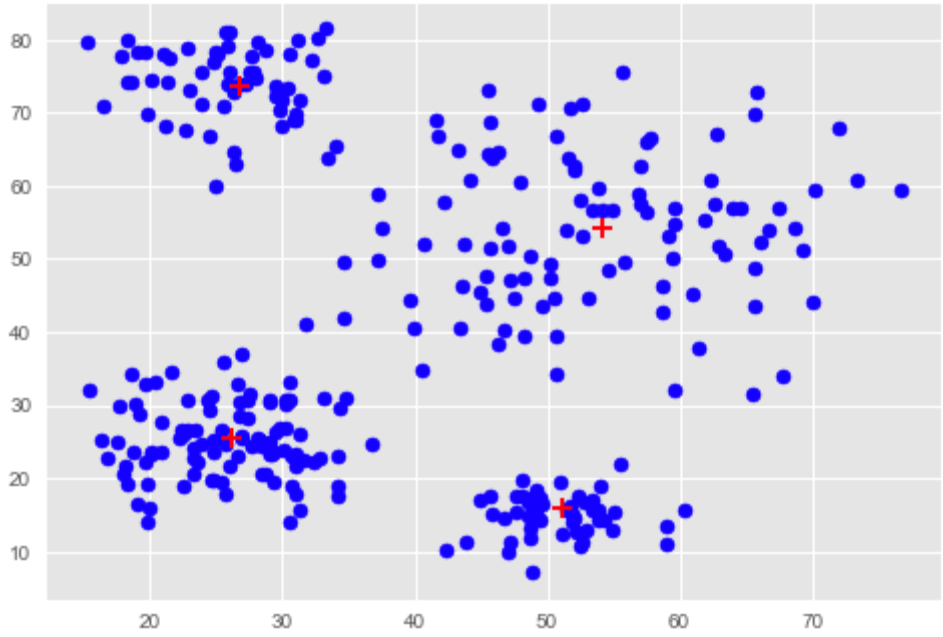
K-meansクラスタリング



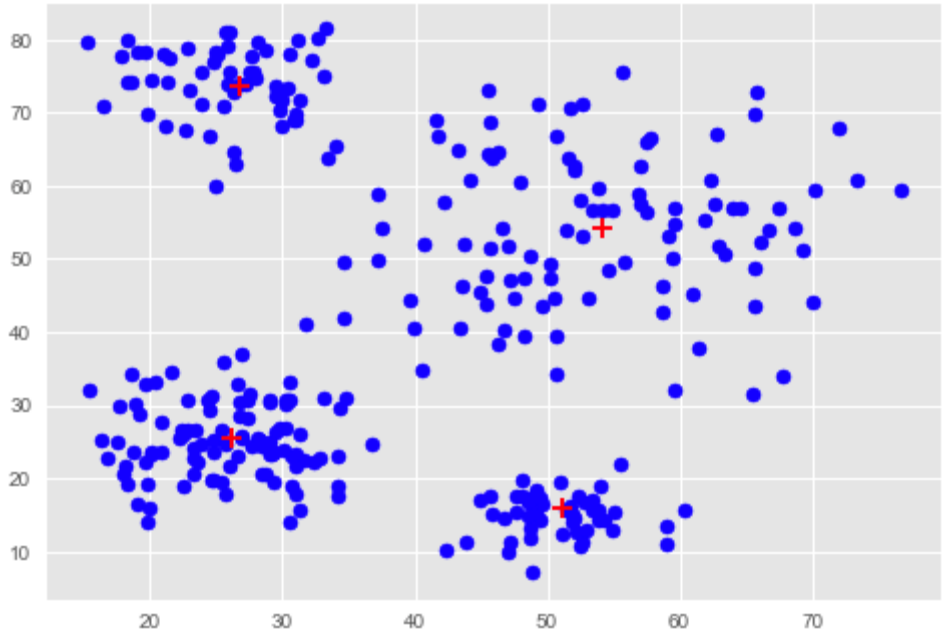
K-meansクラスタリング



K-meansクラスタリング



K-meansクラスタリング



ライブラリを使った K-Means クラスタリング

```
from sklearn.cluster import KMeans
km_model = KMeans(n_clusters=4) #K=4のkmeansモデルの初期化
km_model.fit(vlist) #実際にデータをクラスタリング
```

```
In [564]: km_model.cluster_centers_
Out[564]:
array([[26.15670902, 25.59839271],
       [53.97625751, 54.52132754],
       [51.44211711, 16.33067878],
       [26.76710588, 73.68744256]])
```

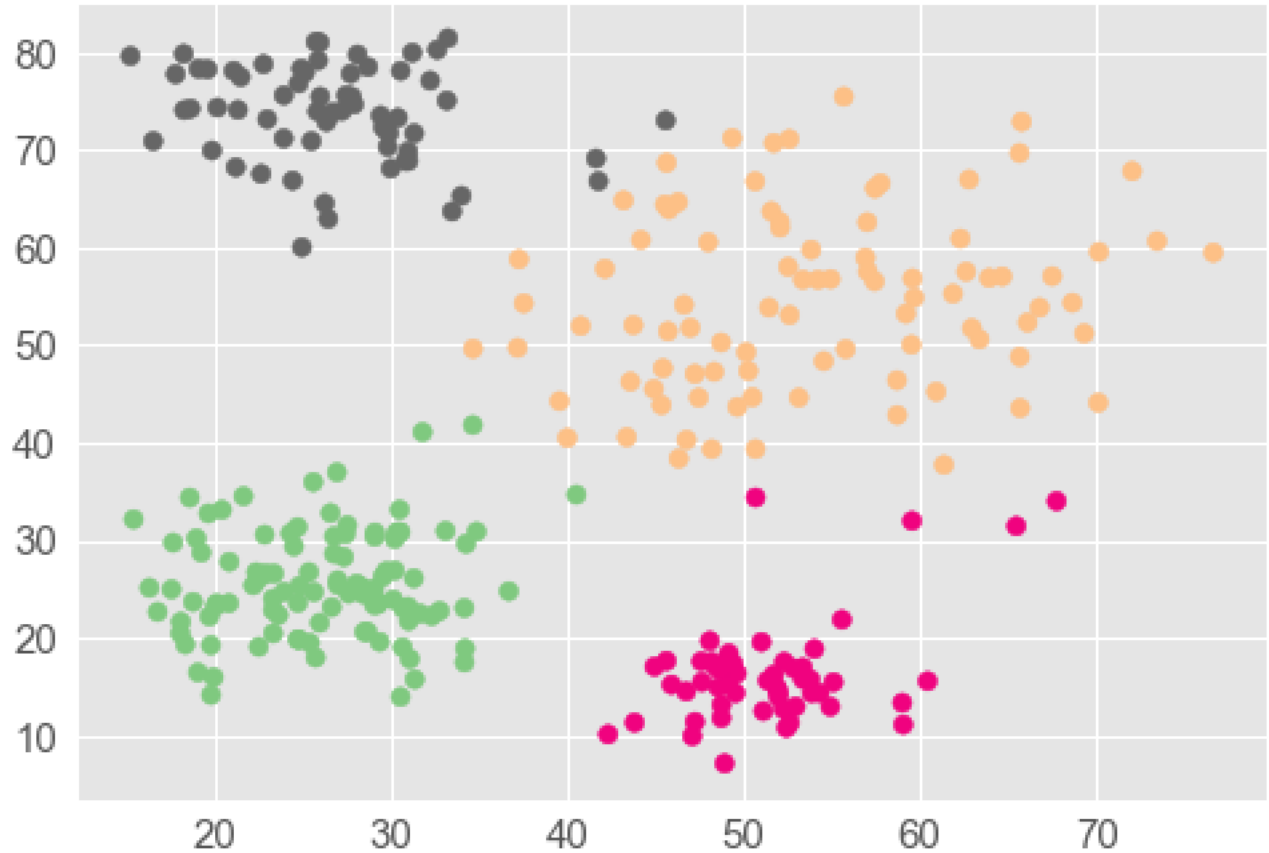
クラスタの中心たち

```
In [567]: km_model.labels_
Out[567]: array([0, 0, 0, ..., 3, 3, 3], dtype=int32)
```

各データがどのクラスタか
（クラスタ０～クラスタ３）

可視化も簡単

```
labels=km_model.labels_ # 各データがどのクラスタかが入っている

plt.scatter(vlist[:,0], vlist[:,1], c=labels,cmap=cm.Accent)
```



演習

演習 1

- csvファイル” height_weight.csv”を読み込み、A=(180,75)からの各点への正規化相間（類似度）をそれぞれ算出し、類似度の小さい順に並び替えて表示
  - ヒント）1行目は項目名なので、無視

- 正規化相間の大きさで色を変えてプロットせよ
  - ヒント）import matplotlib.cm as cm
    cm.hot

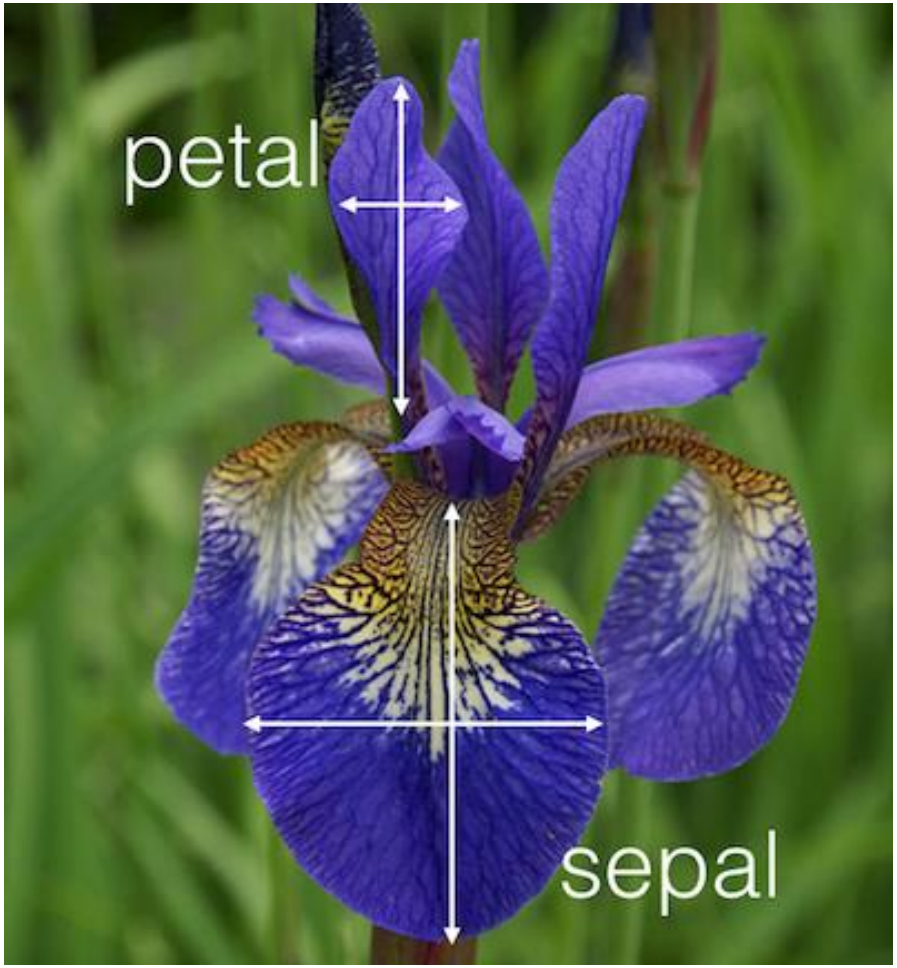
演習 2

- クラスタリングの対象となるcsvデータ“data.csv”を読み込み

- K=2, 3, 4 でそれぞれ K-means クラスタリングせよ
  - 余裕がある人は自分で K-means を実装せよ

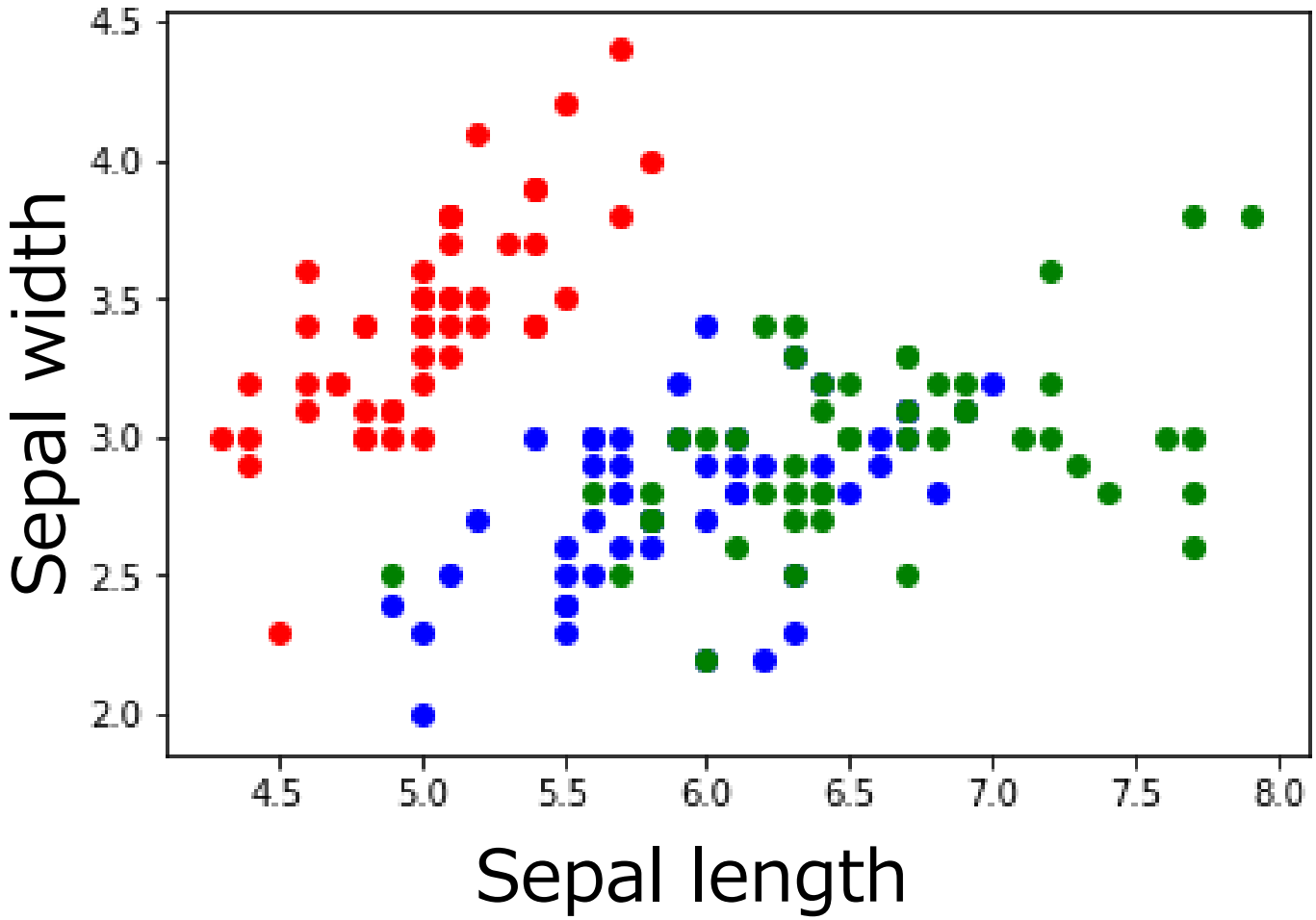
アイリスデータ（１）

- 3クラス
  - Iris-setosa
  - Iris-versicolor
  - Iris-virginica
- 指標
  - sepal length
  - sepal width
  - petal length
  - petal width

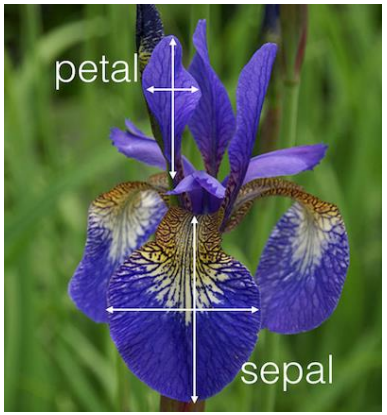


https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

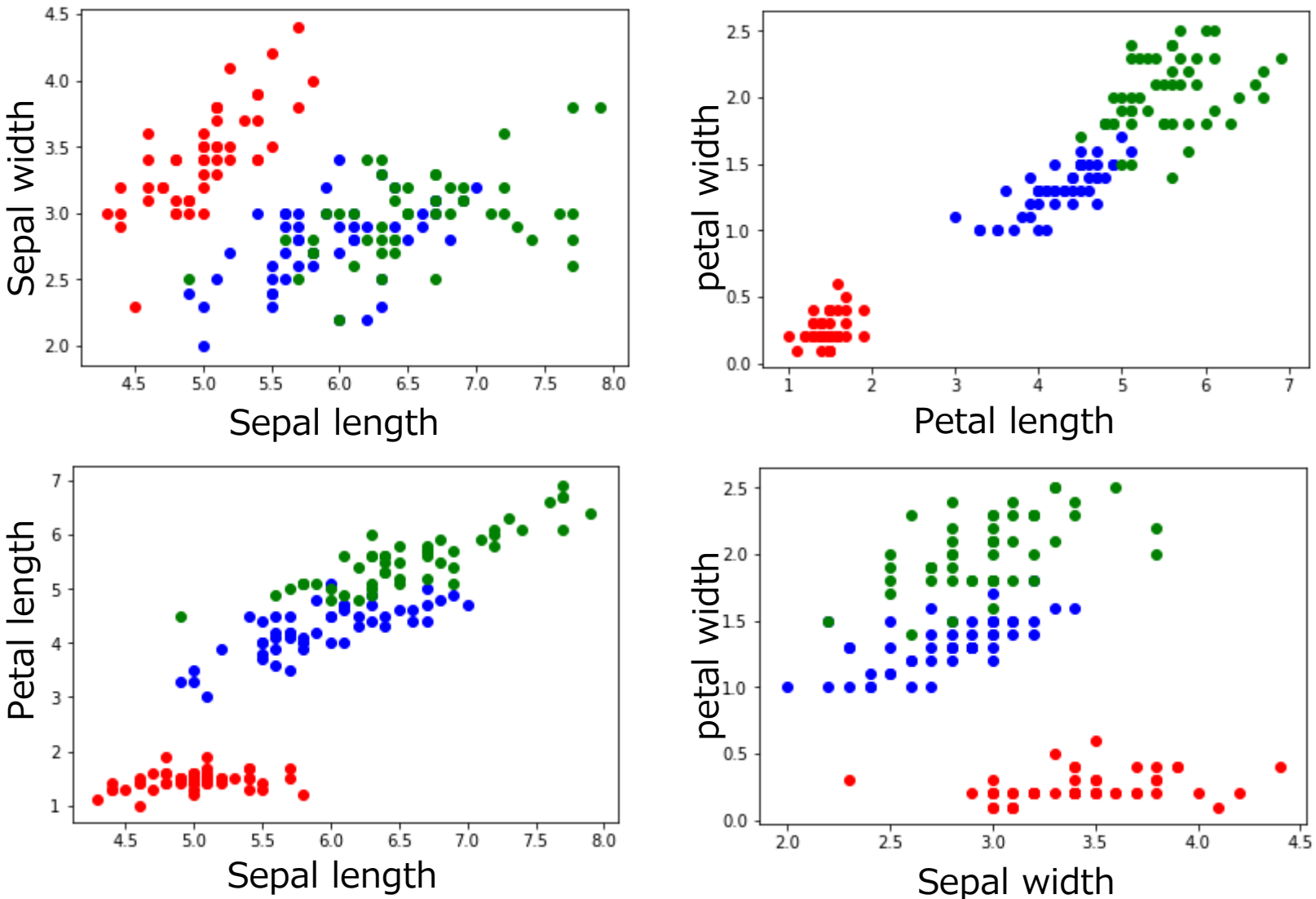
アイリスデータ（２）



- Iris-setosa
- Iris-versicolor
- Iris-virginica



アイリスデータ（３）


Sepal width
Sepal length
petal width
Petal length
Petal length
Sepal length
petal width
Sepal width
setosa
versicolor
virginica

演習 3

- “iris.csv”を読み込み、以下の２つの指標だけで、K= 3 でクラスタリング可能か可視化せよ
- 指標 1
  - sepal length
  - sepal width
- 指標 2
  - sepal width
  - petal width

・自分で K-means を実装
・ライブラリの K-means
どっちもやってみましょう

実際のクラスと
見比べてみよう