「画像処理応用」

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

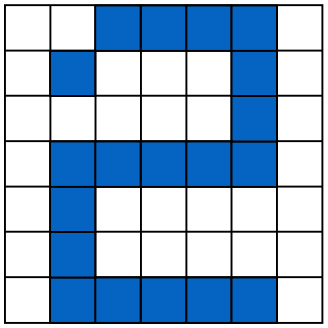
本日の内容

- 画像応用
  - 類似度
  - クラスタリング
  - 画像情報の定量化＋データ解析（検定）

画像応用
類似度

画像のベクトル表現(vector representation of image)

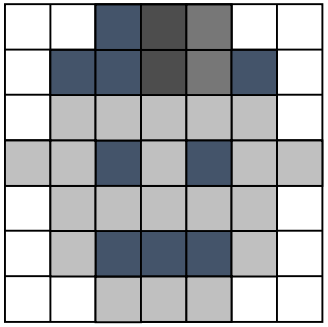
- 2値画像画像の場合の例



7×7画像

$$x = \underbrace{(0,0,1,1,1,1,0,0,1,\ldots,1,0)^T}_{\text{49次元ベクトル}}$$

- 多値画像画像の場合の例

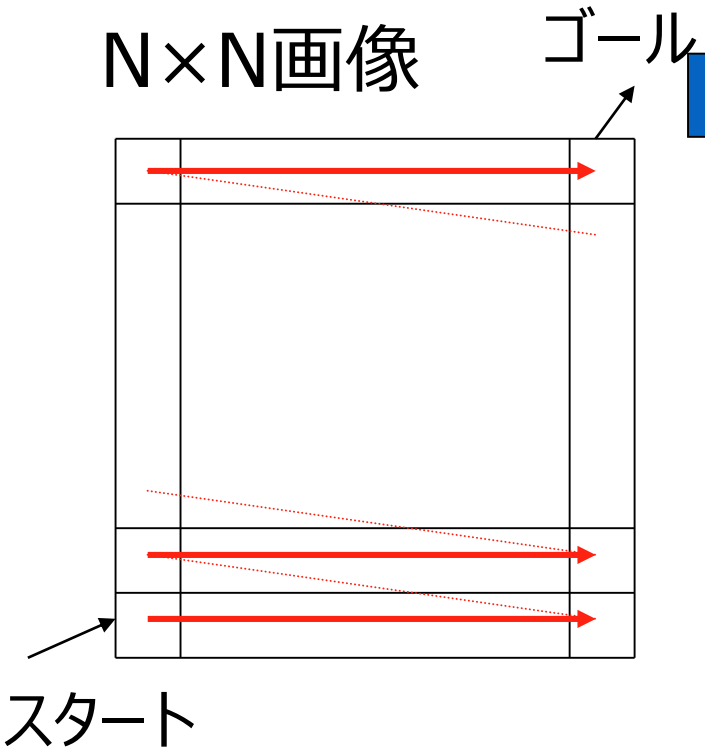


7×7画像

$$x = \underbrace{(0,0,255,213,182,0,0,\ldots,0)^T}_{\text{49次元ベクトル}}$$

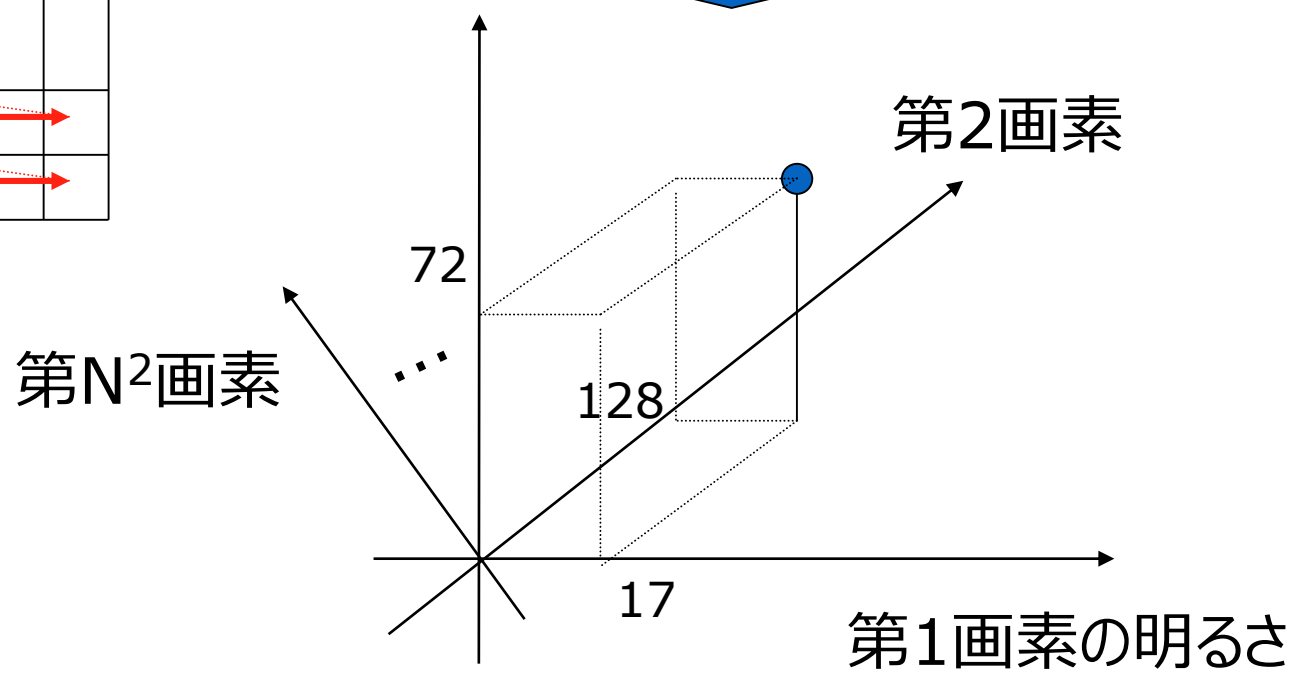
“画像空間” (image space)

N×N画像

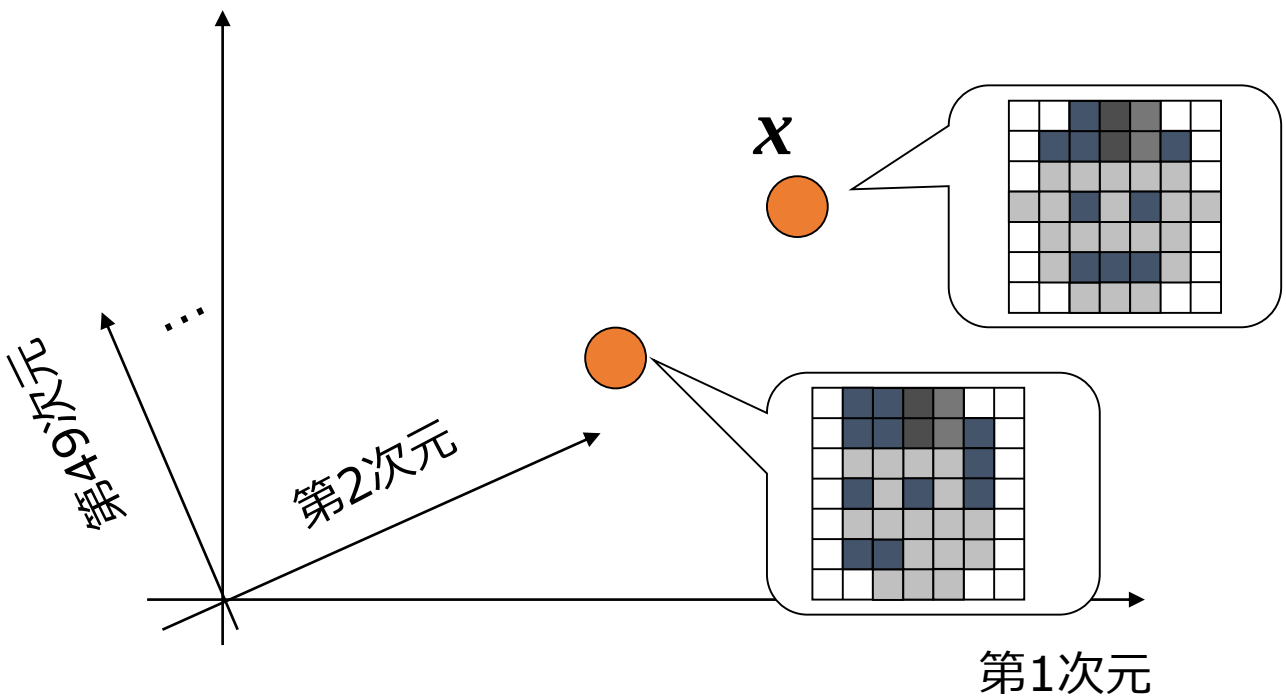


(17, 128, 72, ..., 153)

N^2次元ベクトル

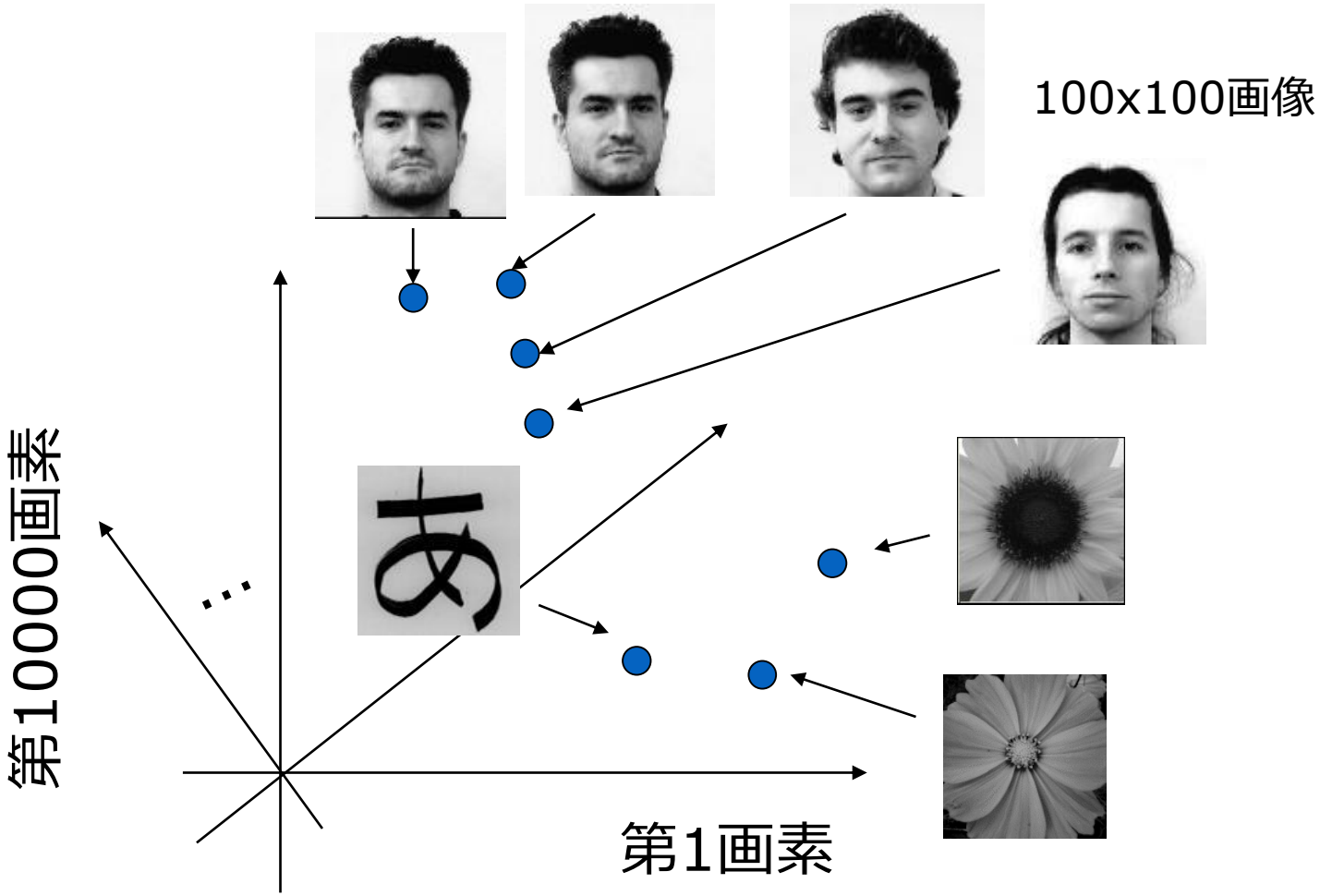


画像空間:任意の１点＝1画像



A point in the image space corresponds to an image

画像空間

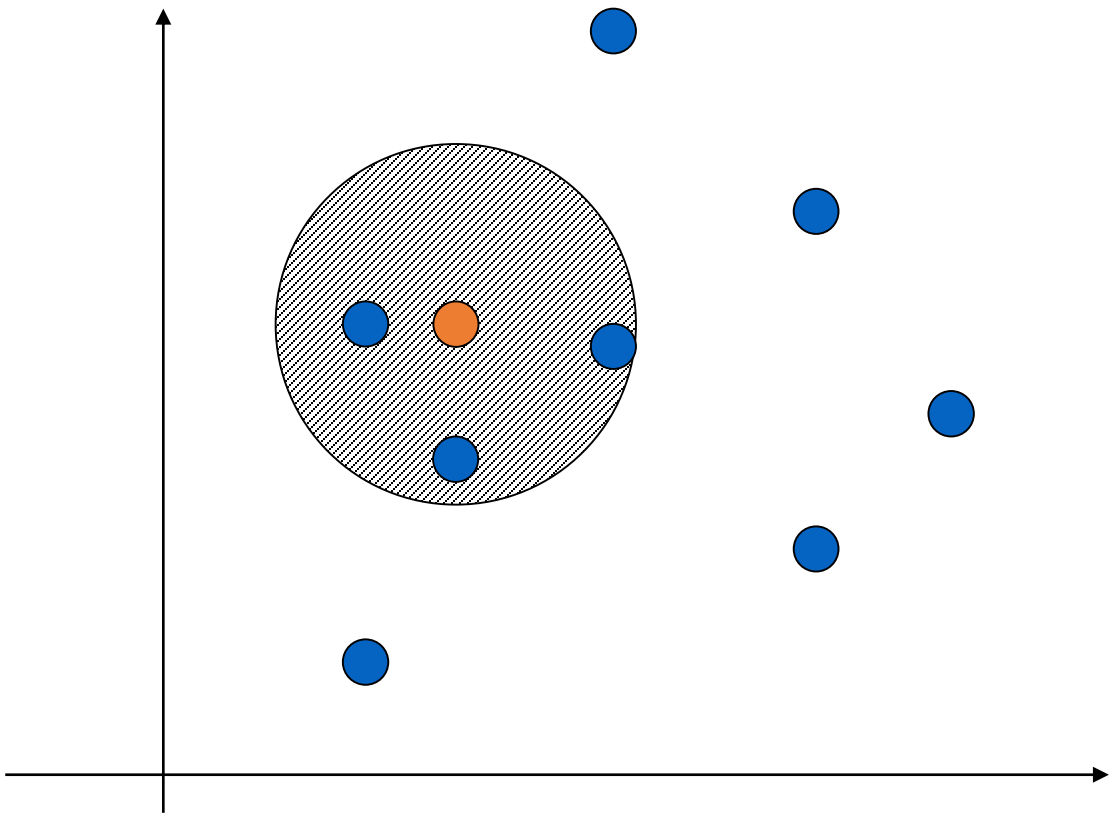


任意の100 x 100画像は10000次元空間で表せる！

画像空間で遊ぶ(play with image space)

- 近い画像を探す (範囲探索)

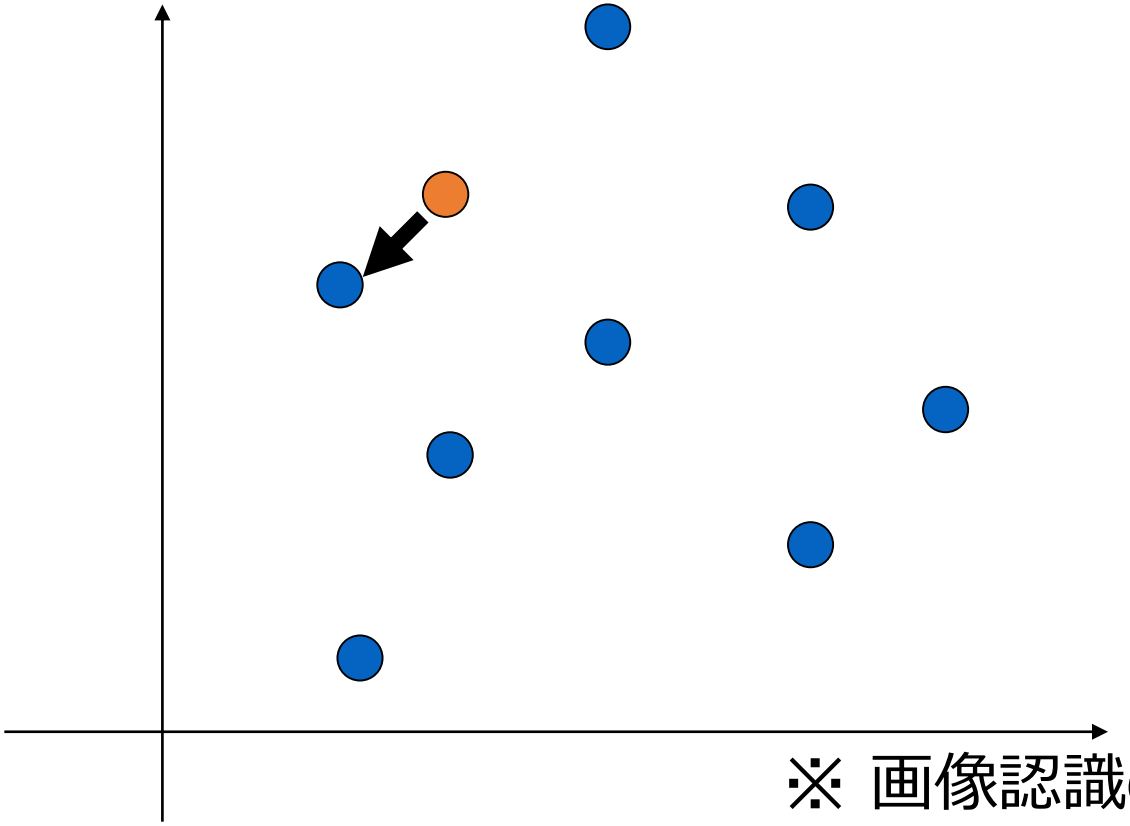
search for similar images



画像空間で遊ぶ（つづき）

- 最も近い画像を探す (最近傍探索)

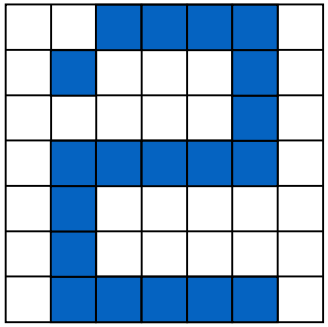
search for the most similar image (the nearest neighbor)

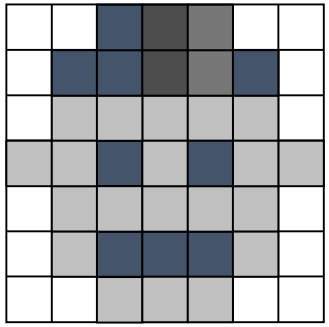


※ 画像認識の原理

練習 1

- 下記の画像をラスタ走査





- 類似度を求める

テンプレートマッチング

- 最も近い画像を探す (最近傍探索)

search for the most similar image (the nearest neighbor)

- クエリ―画像と最も近い領域を探す

クエリー
画像

テンプレートマッチング

クエリー
画像

- 画像の中から最も近い画像を探す
- クエリ―画像と同じ大きさの画像(パッチ)を切り取り、画像間の距離を測る。
- 少しづつ切り取る場所をずらして、一番距離が近い場所を求める

画像間の距離(似てる具合)

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

$$\|\boldsymbol{x} - \boldsymbol{y}\|$$

numpy.linalg.norm(x-y)

38574.86

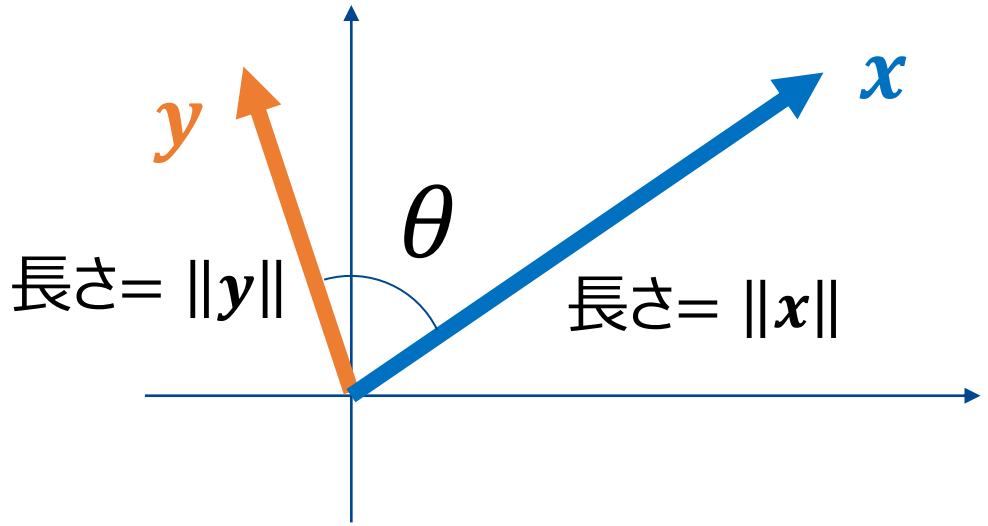
正規化相関

- なんかcos θ も出てきたような…

→ その通り．これ↓をつかって内積計算もできます．

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

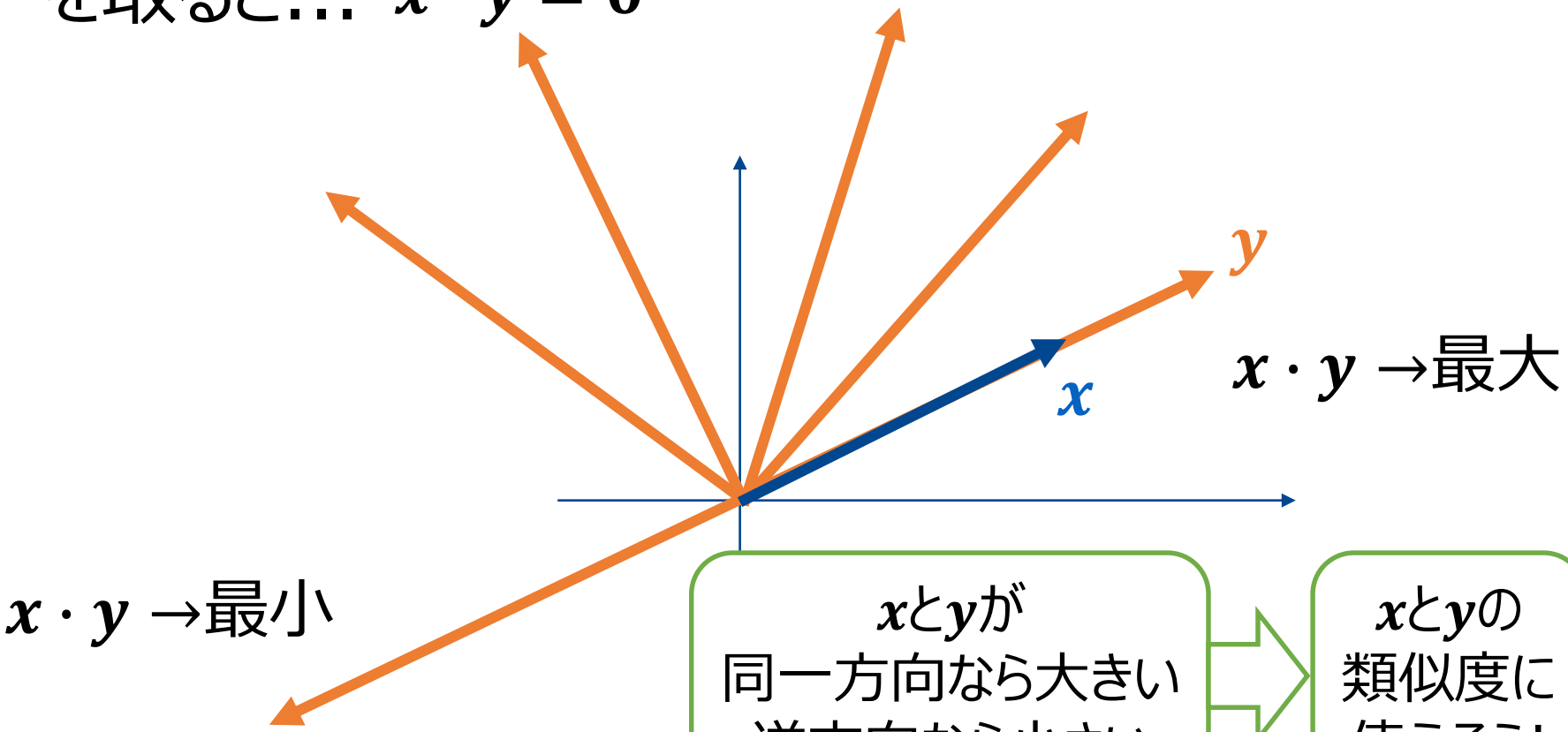
$$\cos\theta = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\| \|\boldsymbol{y}\|}$$



- もっと大事なのは：
  - $-1 \leq \cos\theta \leq 1$
  - θが0度（同じ向きのベクトル）：

思い出そう：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… $\boldsymbol{x}\cdot\boldsymbol{y}=\mathbf{0}$



$\boldsymbol{x}$と$\boldsymbol{y}$が
同一方向なら大きい
逆方向なら小さい

$\boldsymbol{x}$と$\boldsymbol{y}$の
類似度に
使えそう!

テンプレートマッチング

```
from skimage import io
lenna = io.imread('LENNA.bmp')
template = io.imread('lenna_template.tif')

io.imshow(lenna)

io.imshow(template)
```

http://scikit-image.org/docs/stable/auto_examples/

テンプレートマッチング

```
N,M = lenna.shape
wx,wy = template.shape
hwx = (wx-1)/2
hwy = (wy-1)/2
D = np.zeros(lenna.shape)
S = np.zeros(lenna.shape)
a = template.reshape([template.size])
a = a.astype(np.float64)
minsim = 1
for i in range(hwx,N-hwx):
    for j in range(hwy,M-hwy):
        tmp = lenna[i-hwx:i+hwx+1,j-hwy:j+hwy+1]
        b = tmp.reshape([tmp.size])
        b = b.astype(np.float64)
        dis = np.linalg.norm(a-b)
        sim = np.dot(a,b)/(np.linalg.norm(a)*np.linalg.norm(b))
        D[i,j] = dis
        S[i,j] = sim
        if minsim > sim:
            minsim = sim
```

Lennaの画像サイズを取得

テンプレート画像サイズを取得

D：距離行列

S：類似度行列

Lennaと同じサイズの画像（2次元配列）を初期化

テンプレート画像を一次元配列に変換

各画素ごとに距離と類似度を算出

着目画素周辺のテンプレート画像と同じ大きさのパッチ画像を取得

パッチ画像をベクトルへ変換

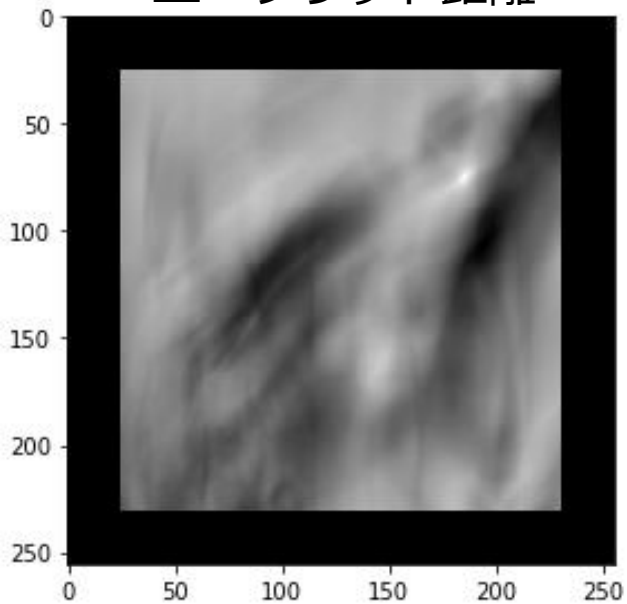
距離

類似度

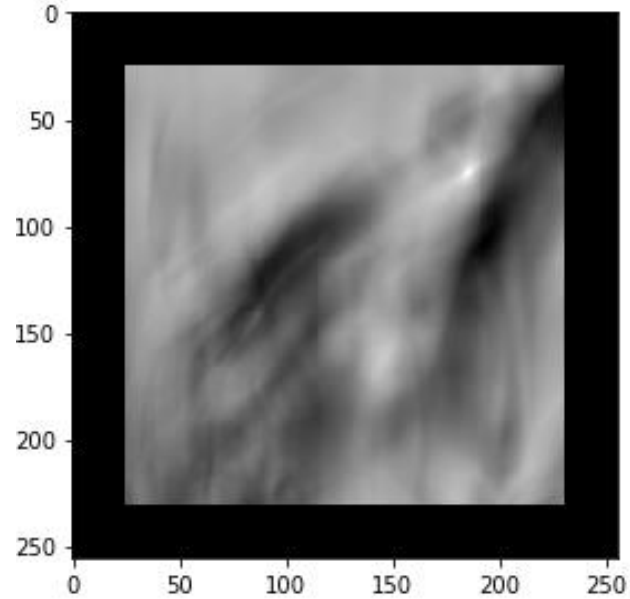
テンプレートマッチング

```
# visualization
D = D/np.max(D)
io.imshow(1-D)

S2 = (S-minsim)/(np.max(S)-minsim)
S2[S2<0] = 0
io.imshow(S2)
```

ユークリッド距離

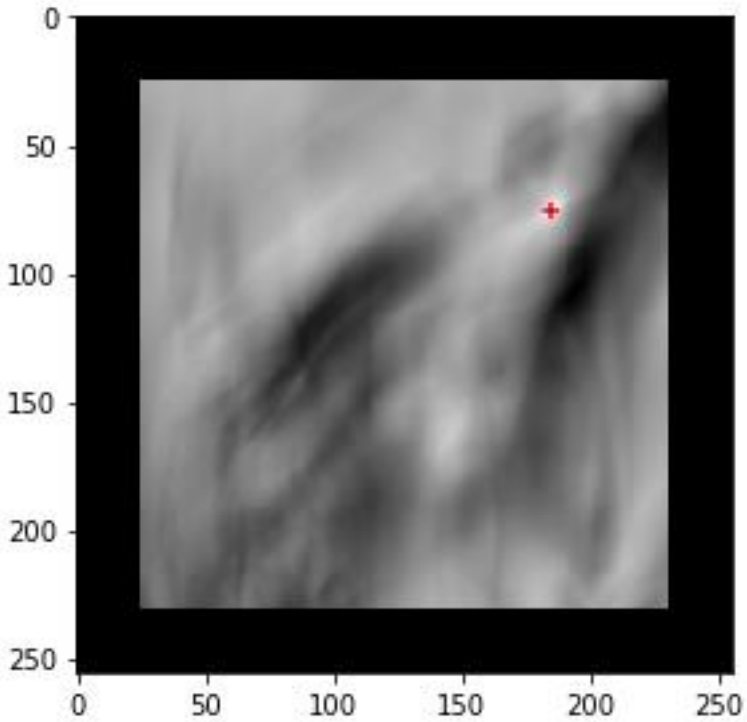


正規化相関



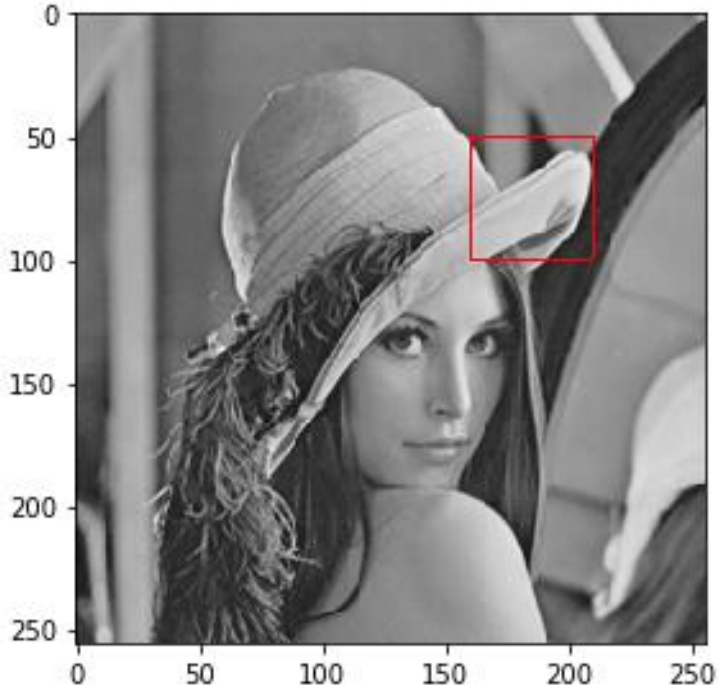
テンプレートマッチング

```
from skimage.color import gray2rgb
iy,ix = np.unravel_index(S.argmax(), S.shape)
io.imshow(S2)
plt.scatter(ix,iy,c='red',marker='+')
```



テンプレートマッチング

```
im = gray2rgb(lenna)
im[iy-hwy:iy+hwy,ix-hwx,:] = [255,0,0]
im[iy-hwy:iy+hwy,ix+hwx,:] = [255,0,0]
im[iy-hwy,ix-hwx:ix+hwx,:] = [255,0,0]
im[iy+hwy,ix-hwx:ix+hwx,:] = [255,0,0]
io.imshow(im)
```



練習 2

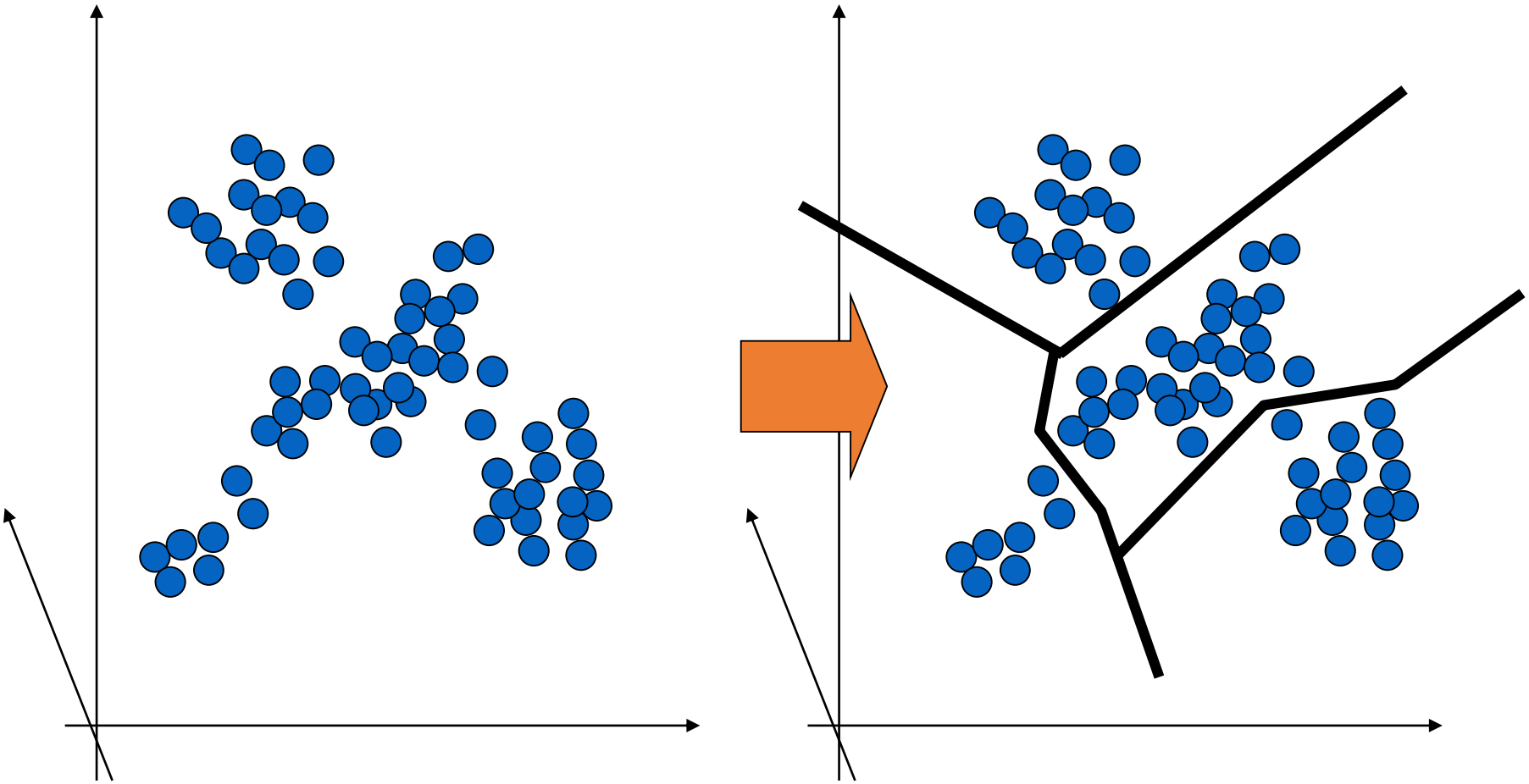
- “lenna_template2.bmp”でテンプレートマッチング

- ユークリッド距離可視化
- 緑でテンプレートマッチング可視化

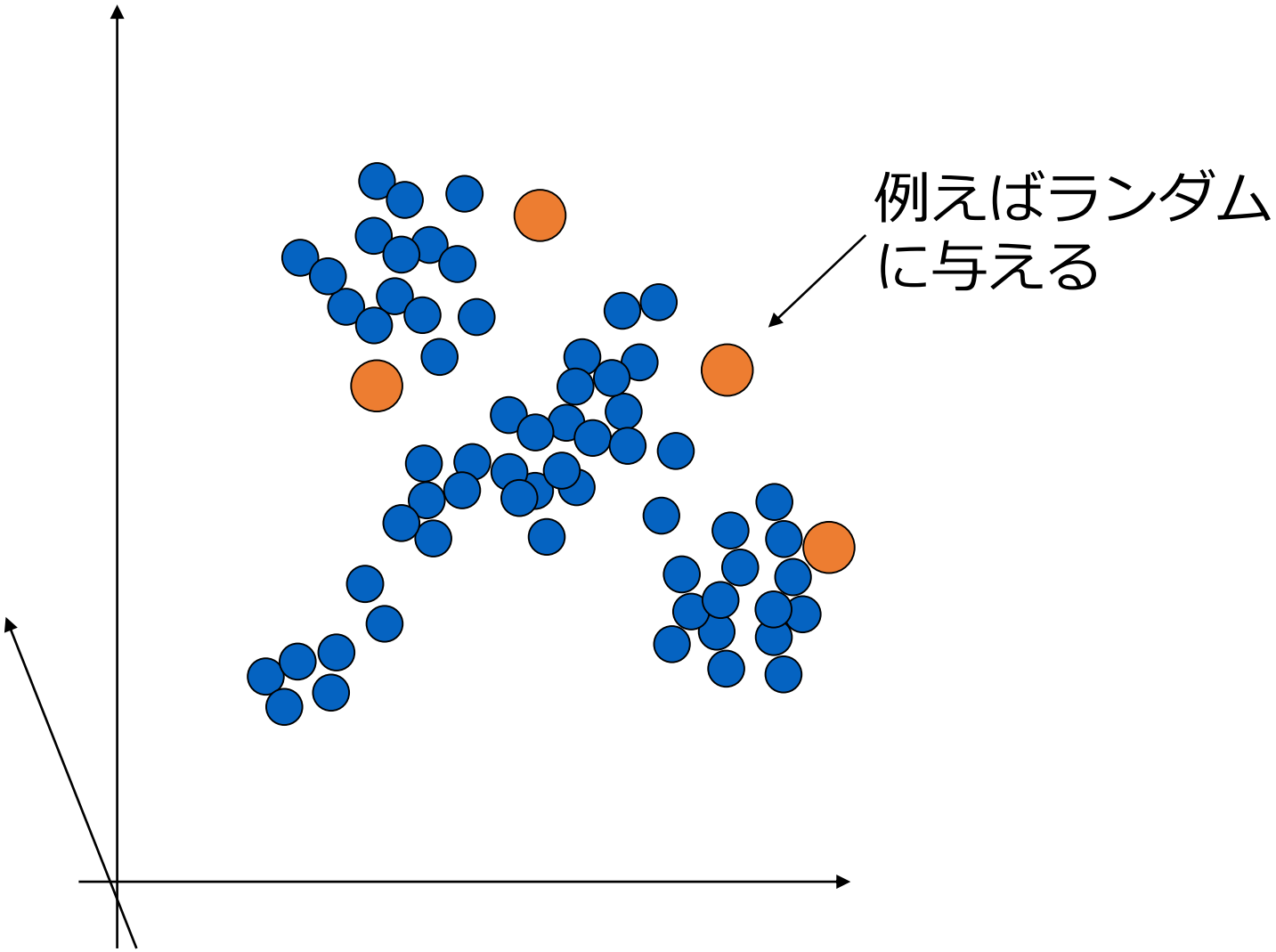
画像応用
クラスタリング

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)

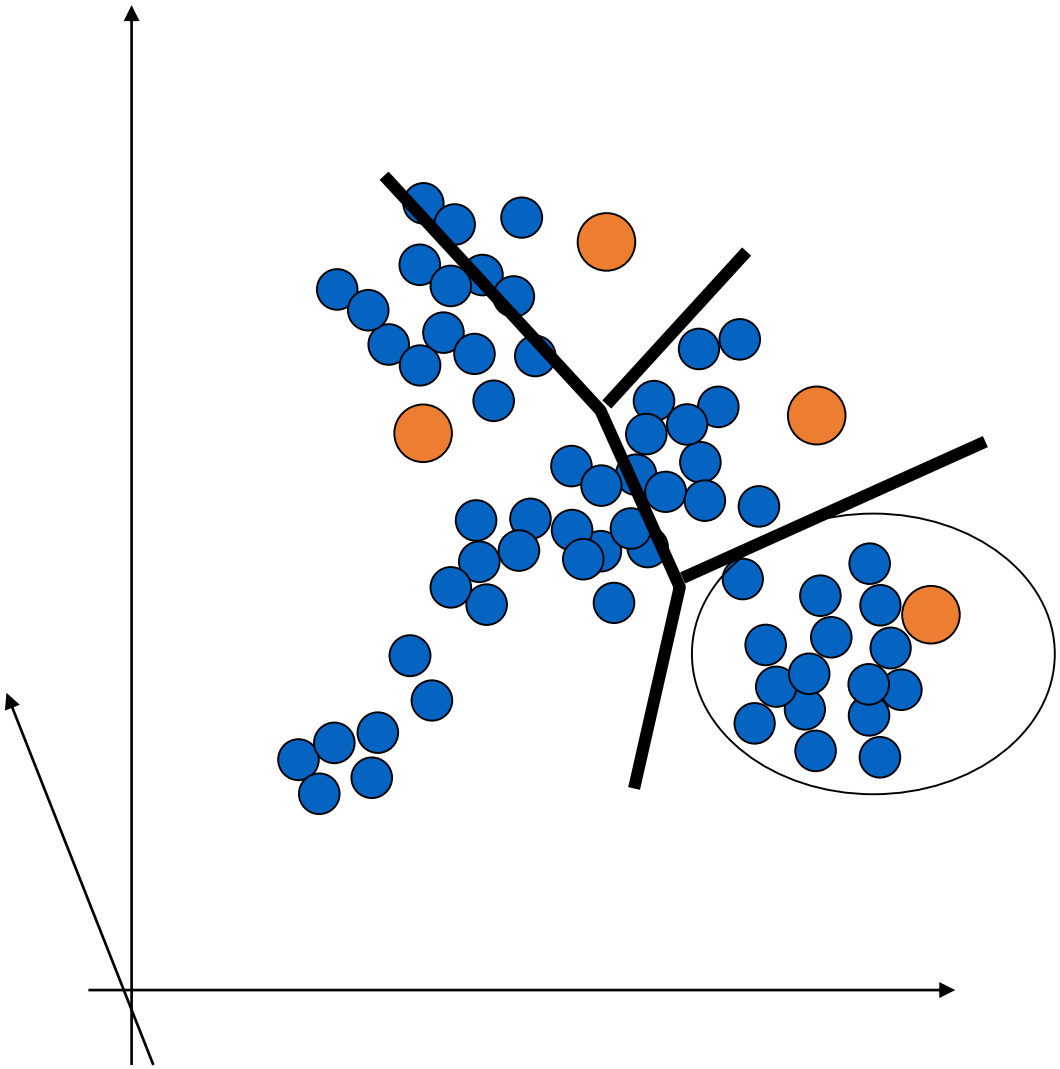
- 各部分集合＝「クラスタ」と呼ばれる



代表的なクラスタリング法：K-means法（0）初期代表パターン

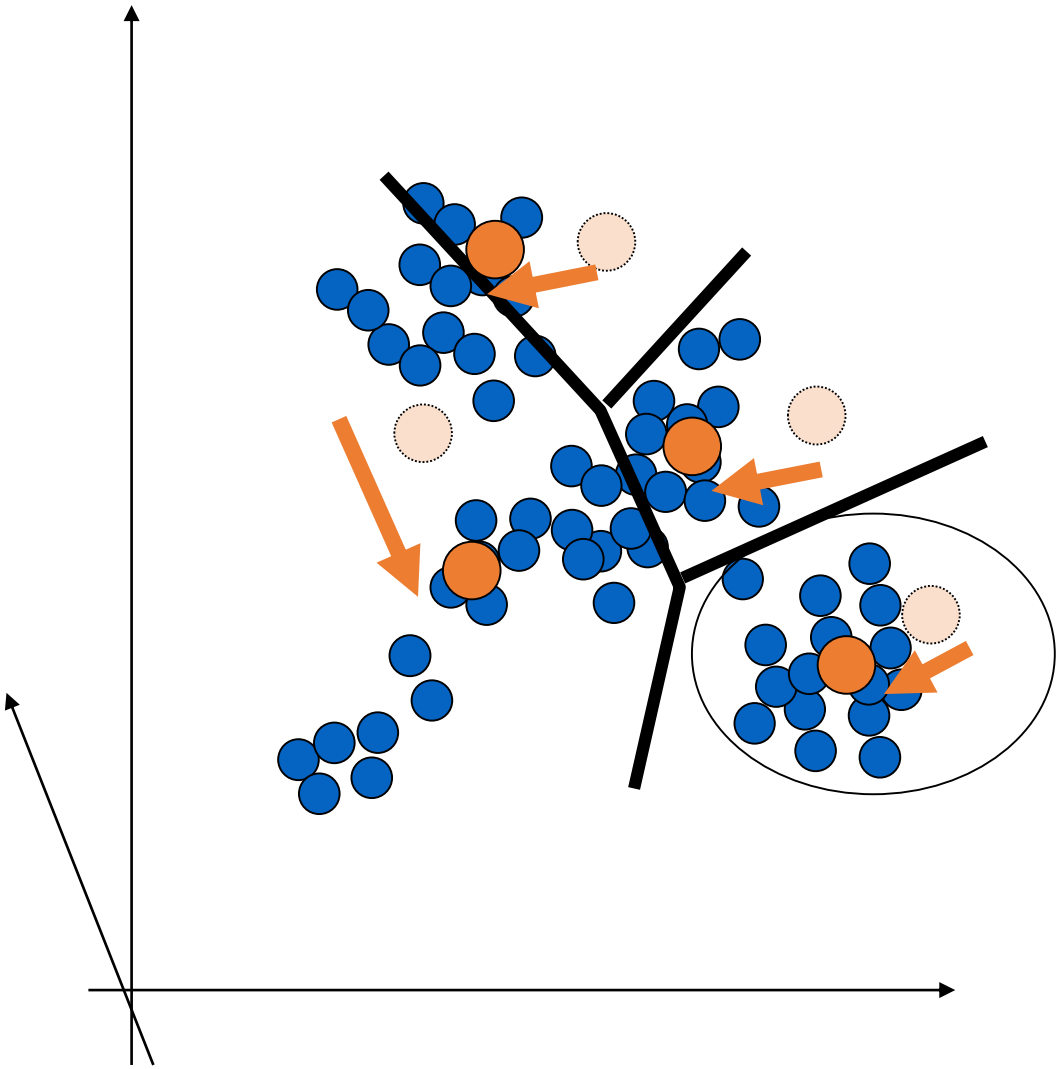


代表的なクラスタリング法：K-means法（1）学習パターン分割



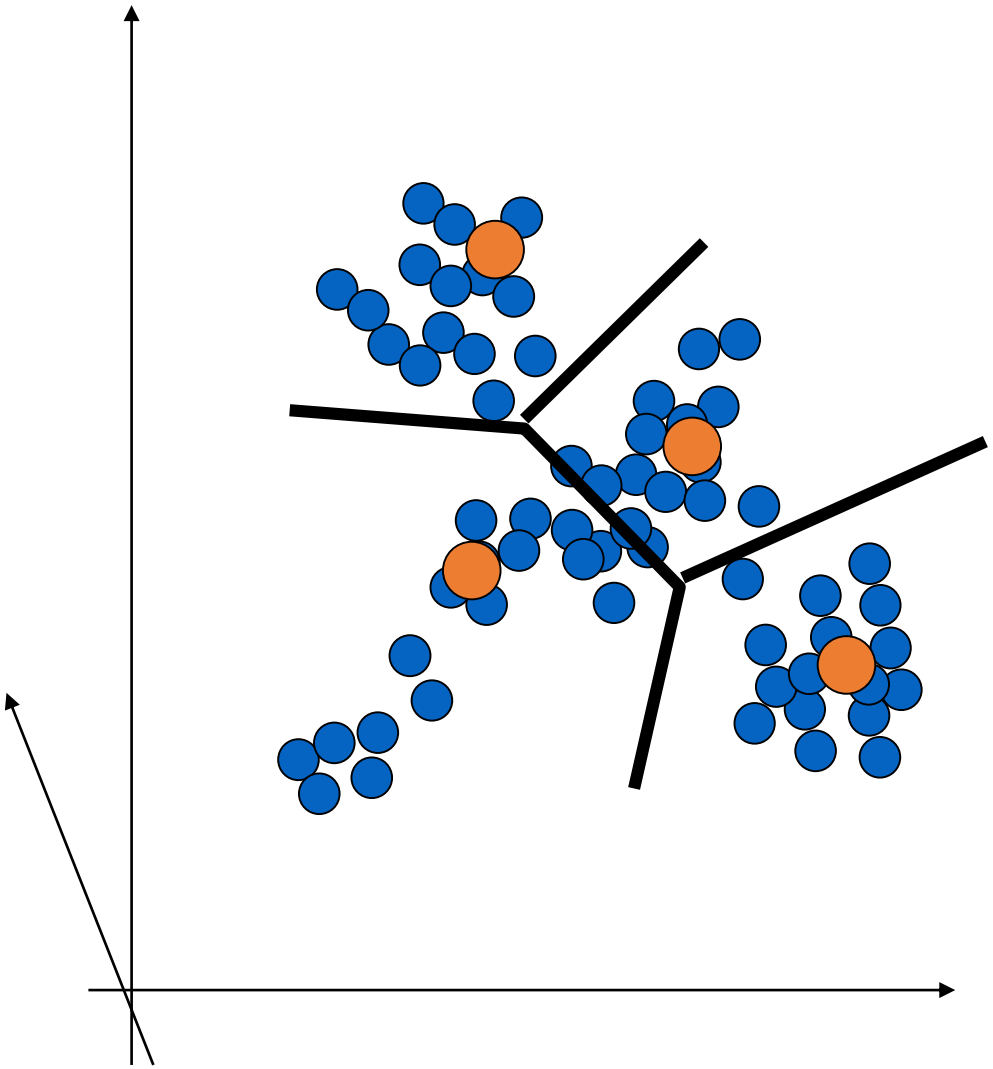
代表パターンの
ファンクラブ
（支持者集合）

代表的なクラスタリング法：K-means法（2）代表パターン更新

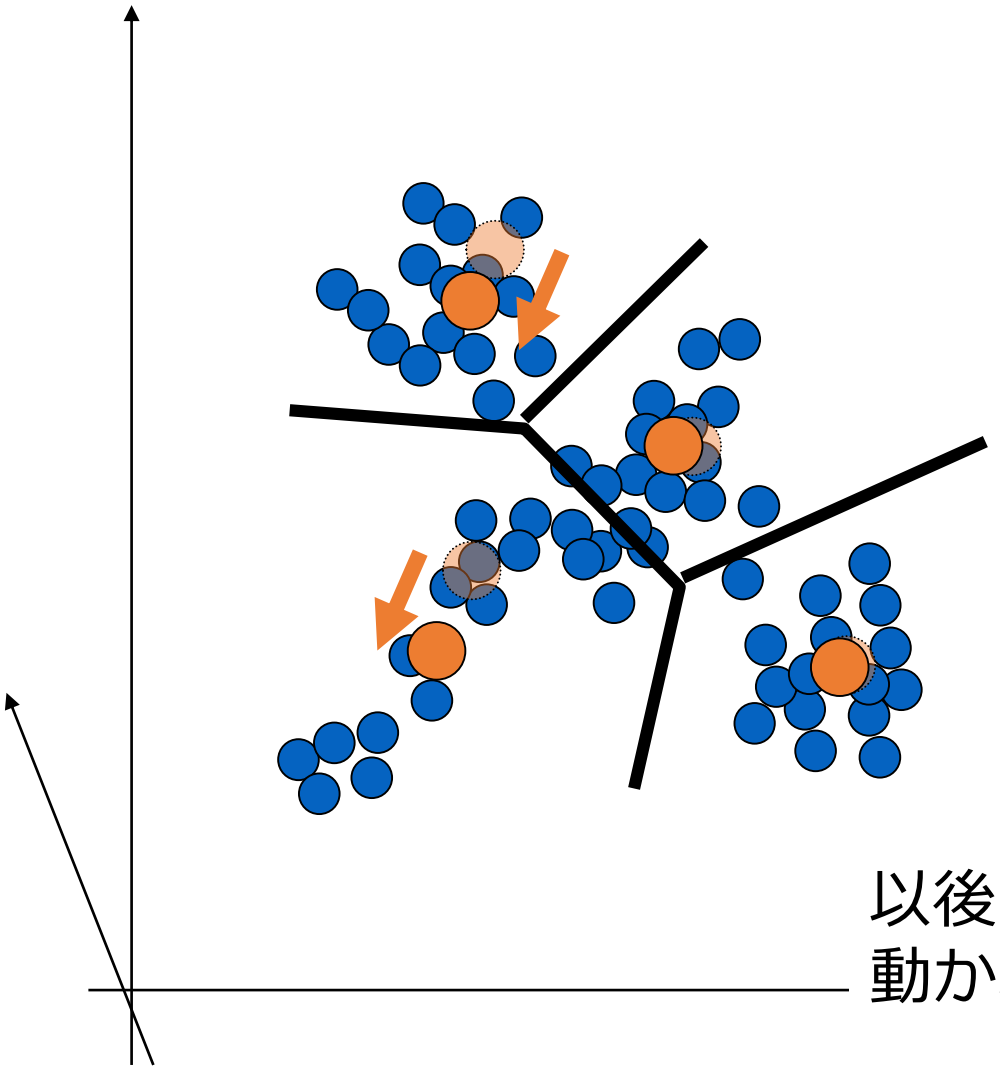


この中の学習パターンの平均に移動

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



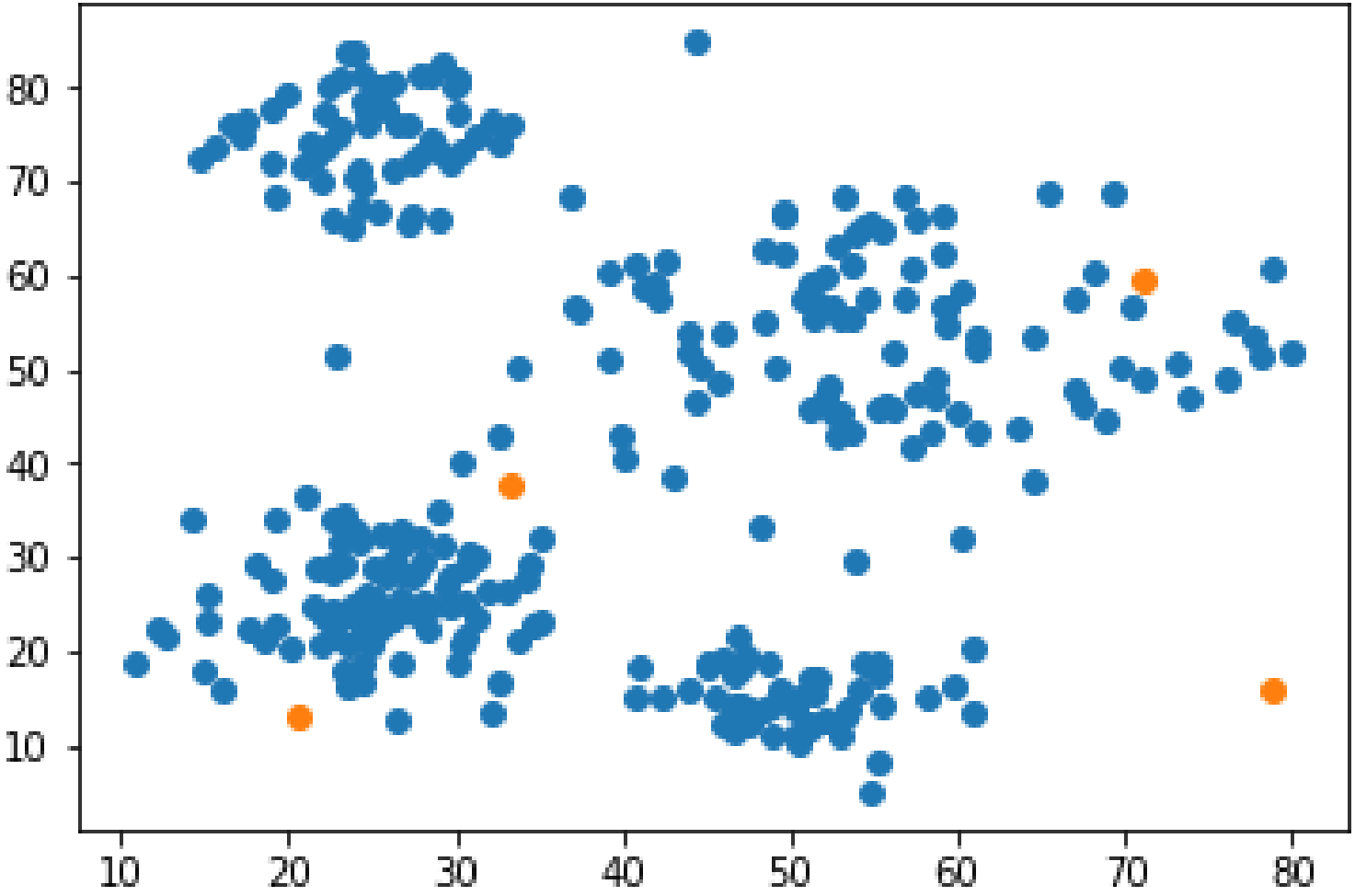
以後，代表パターンが動かなくなるまで反復

K-means：プログラミング

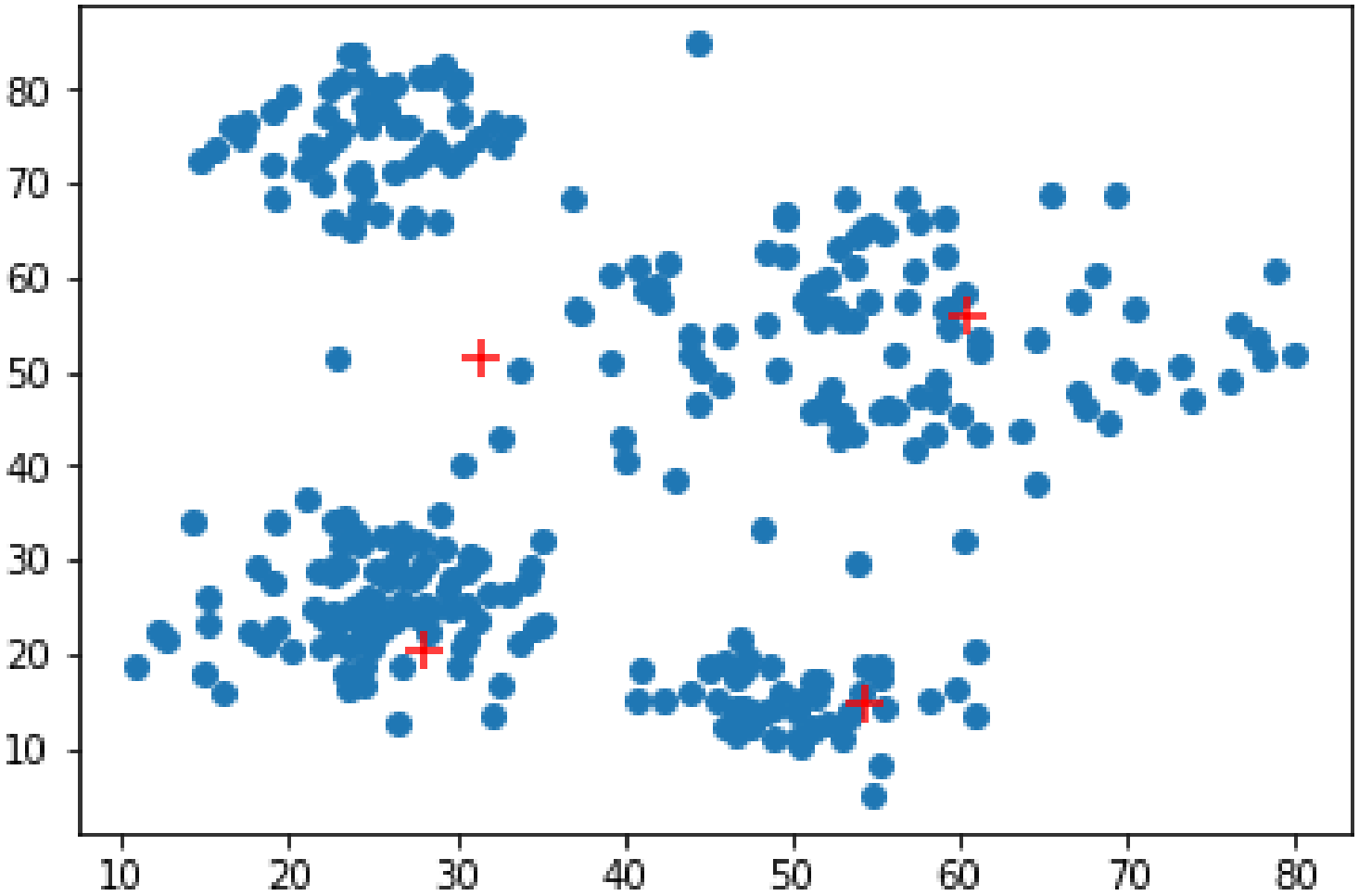
- アルゴリズム
  - ステップ0
  - 繰り返し
    - 代表点の選択
    - 代表点の更新
    - 終了チェック

```
Err = 0.1
dd = 10000;
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)
        if dd > Err:
            end_frag = False
    itt += 1
    if end_flag == True:
        break
```

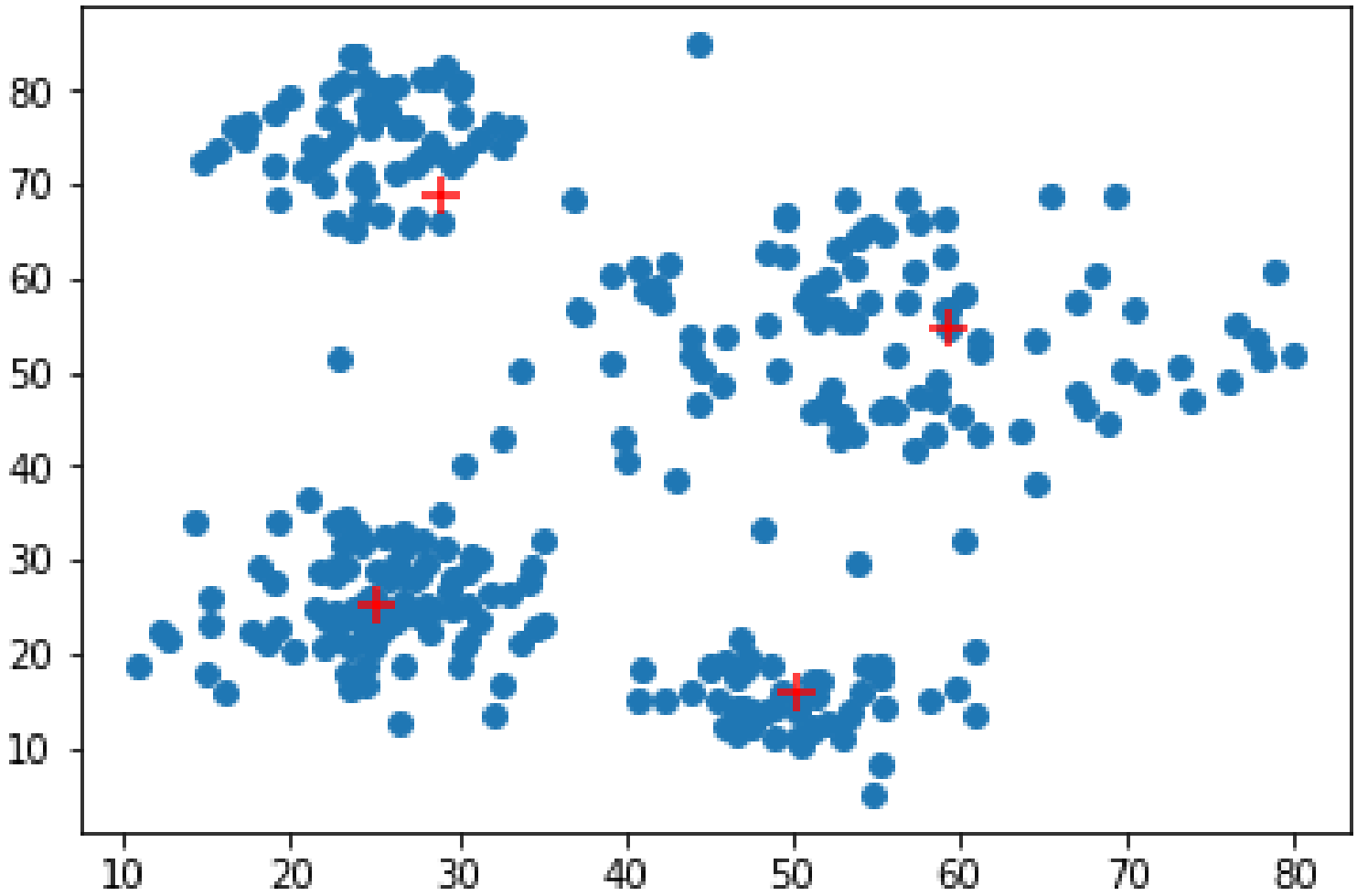
K-meansクラスタリング



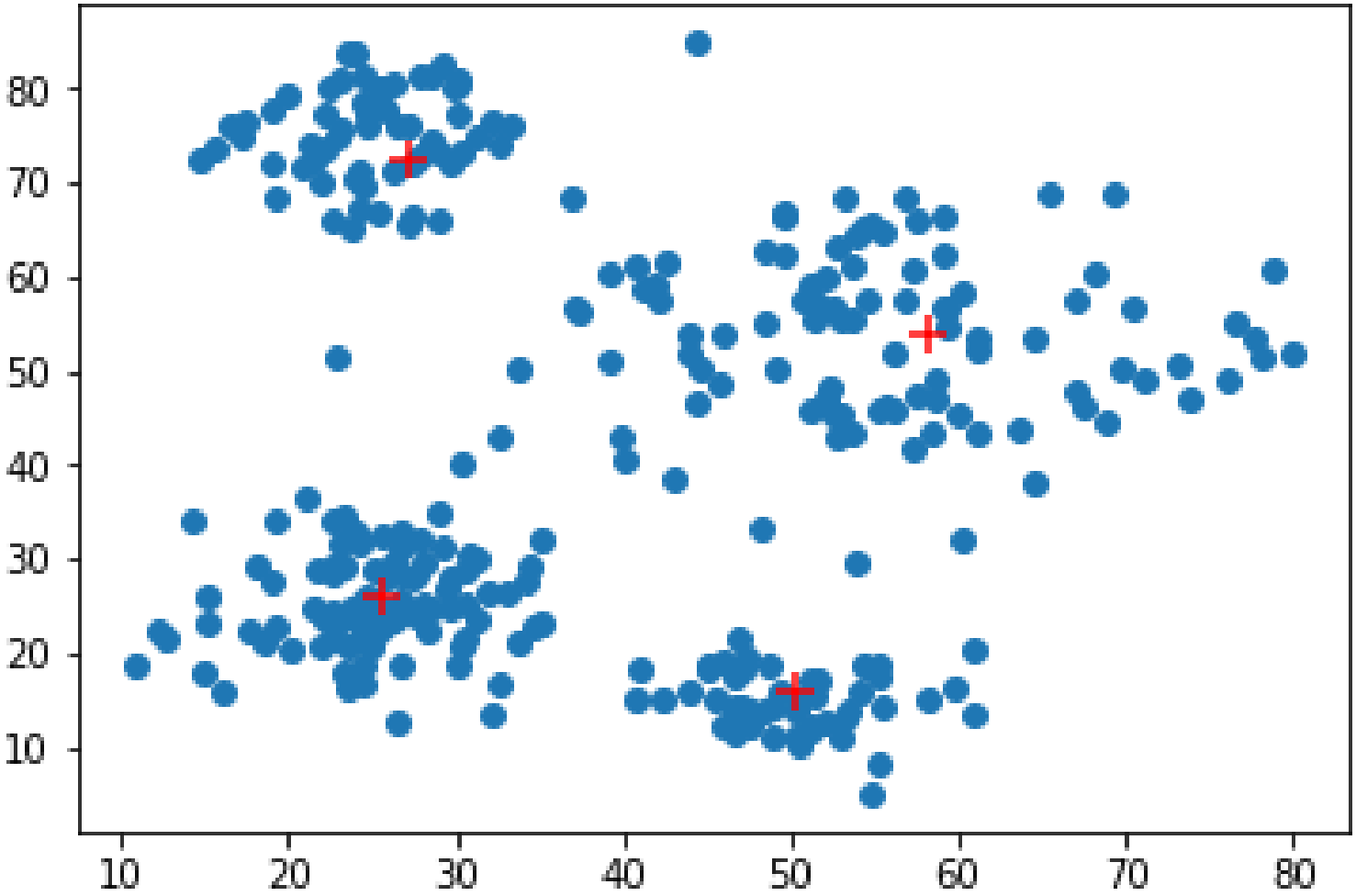
K-meansクラスタリング



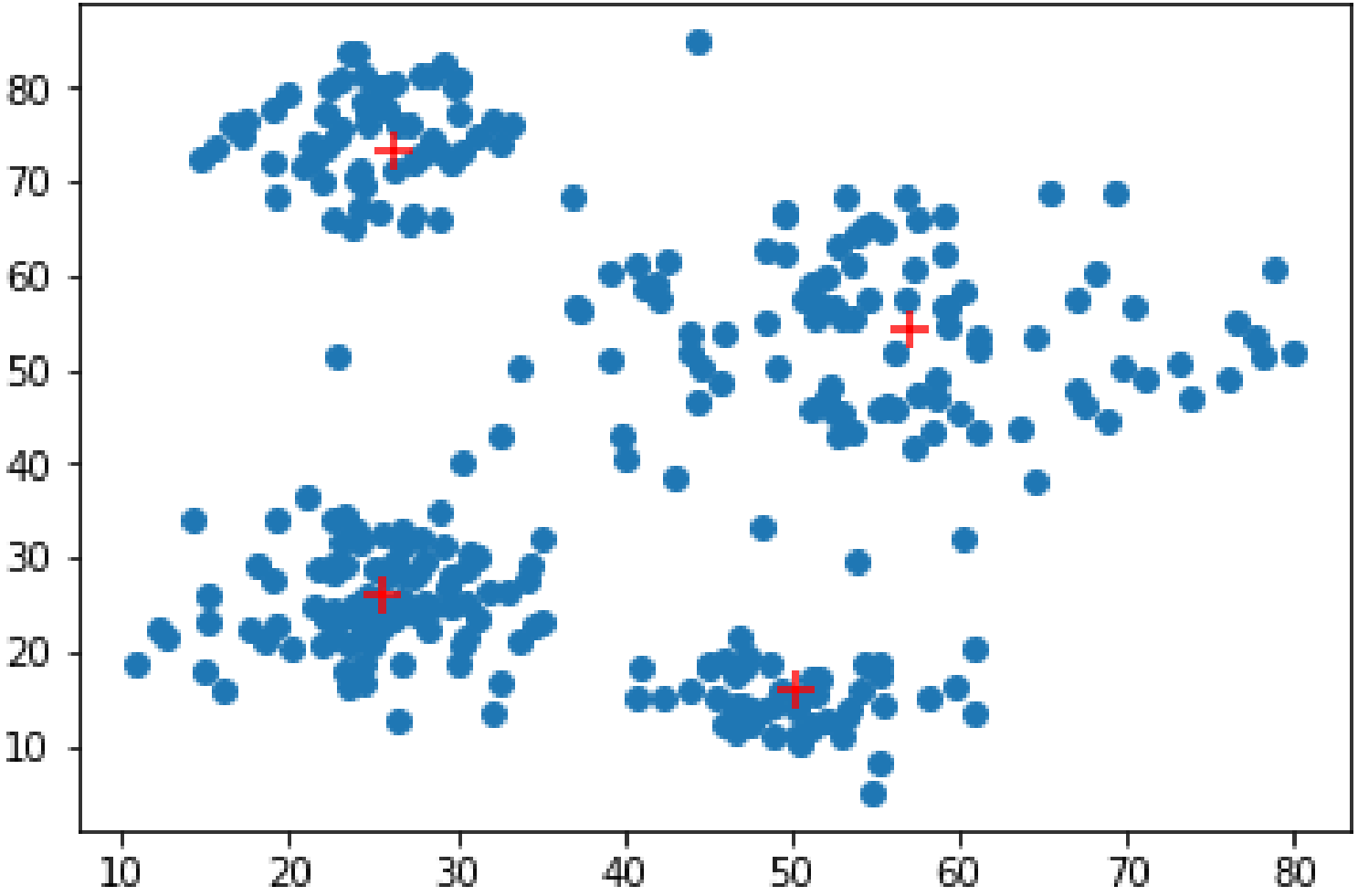
K-meansクラスタリング



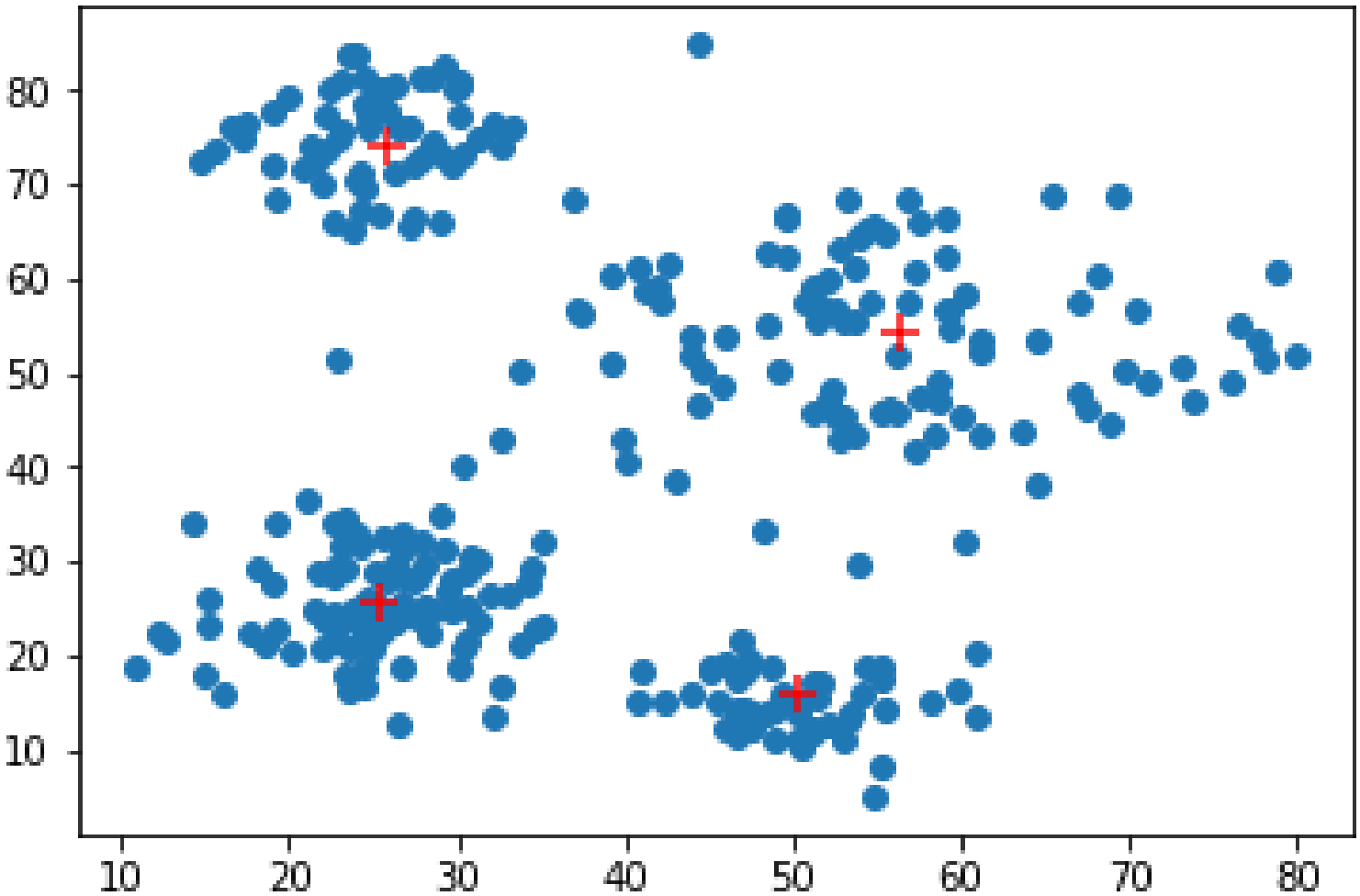
K-meansクラスタリング



K-meansクラスタリング



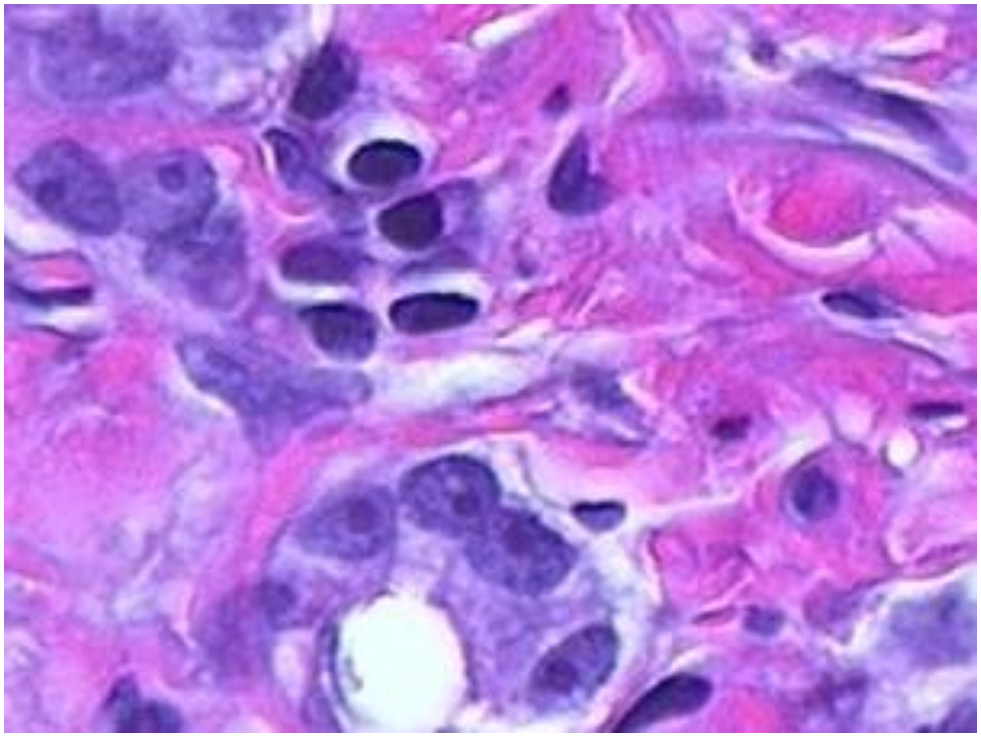
K-meansクラスタリング



画像中の色のクラスタリング（1/4）

- K-meansを使って画像中の色を分けてみよう！
- ‘hestain.png’画像中の色を3つに分けよう

```
from skimage import io
from skimage import color
he = io.imread('hestain.png')
io.imshow(he)
plt.show()
```

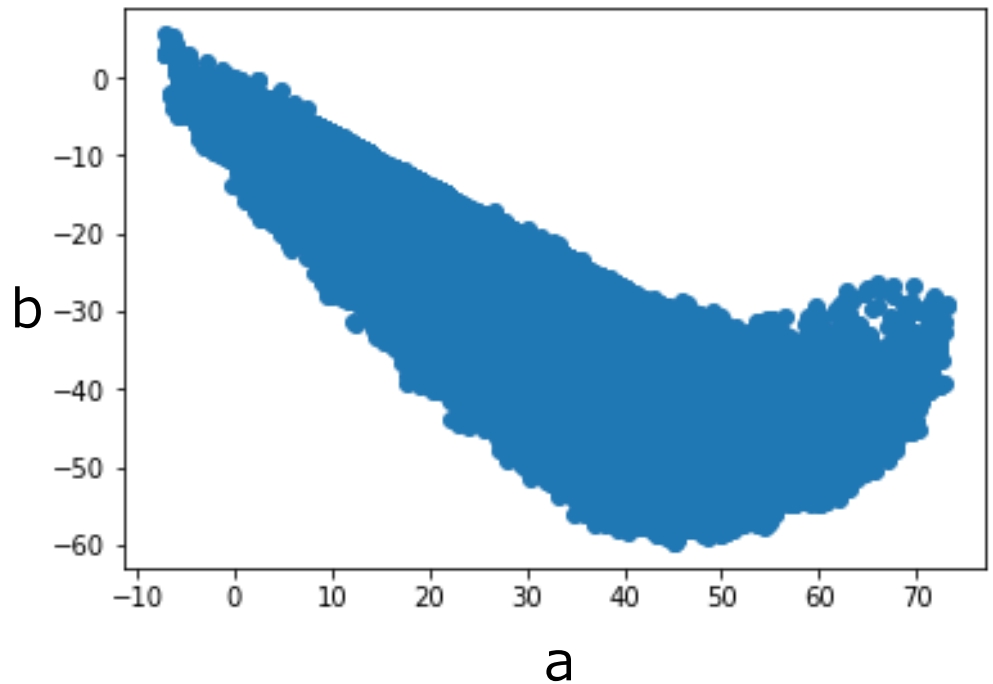


https://jp.mathworks.com/help/images/examples/color-based-segmentation-using-k-means-clustering.html

画像中の色のクラスタリング（2/4）

- Lab色空間に変換
  - L：明度
  - a, b：色情報

```
# L*a*b色空間へ変換
helab = color.rgb2lab(he)
N, M, _ = helab.shape
a = helab[:,:,1]
b = helab[:,:,2]
a = a.reshape(a.size)
b = b.reshape(b.size)
ab = []
for i in range(len(a)):
    ab.append([a[i],b[i]])
ab = np.array(ab)

# scattering
plt.scatter(ab[:,0],ab[:,1])
```



https://ja.wikipedia.org/wiki/Lab%E8%89%B2%E7%A9%BA%E9%96%93

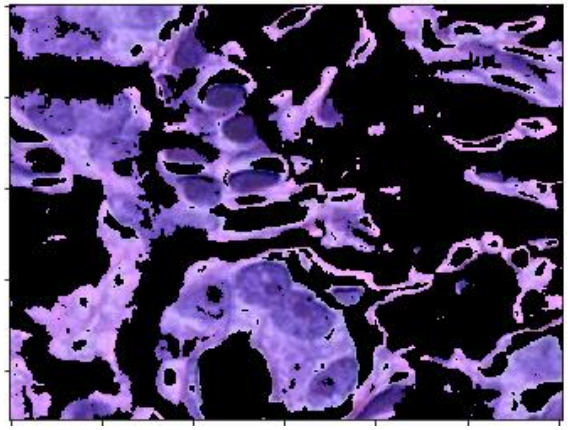
画像中の色のクラスタリング（3/4）

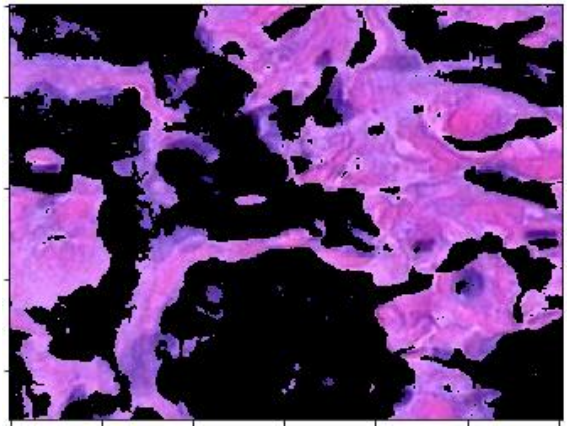
```
# cluster 数
K = 3

# 初期化: random値の範囲決定
vlist = ab
inds = np.round(len(vlist[:,1])*np.random.rand(3)).tolist()
inds = [int(inds[0]), int(inds[1]), int(inds[2])]
CList = vlist[inds]
orgCList = CList

# k-means clustering
Err = 0.1
dd = 10000;
maxittr = 100
itt = 0
while  itt < maxittr:
    end_flag = True
    preCList = CList
    CIList = selectCluster(vlist,CList) # 指示クラスタの決定
    CList = updataCenter(vlist,CIList,K) # クラスタ中心の更新
    # 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)
        if dd > Err:
            end_frag = False
    itt += 1
    if end_flag == True:
        break
```

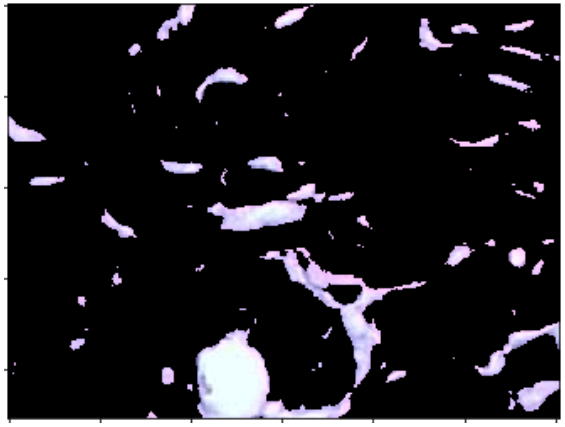
画像中の色のクラスタリング（4/4）

- K-meansを使って画像中の色を分けてみよう！

```
# 色でセグメント化するイメージの作成
CIList = np.array(CIList)
pixelLabels = CIList.reshape(N,M)
io.imshow(pixelLabels)

# cluster1
he = io.imread('hestain.png')
c = he
c[pixelLabels!=0] = 0
io.imshow(c)
plt.show()
```

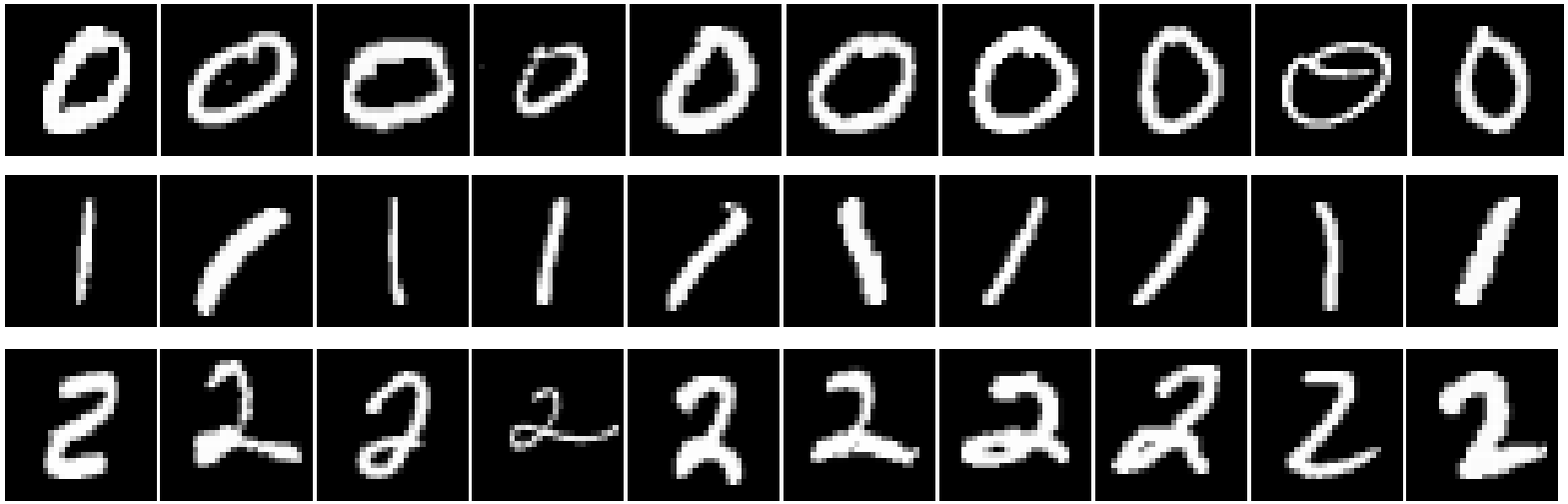






手書き文字のクラスタリング（1/）

- 0から2の手書き文字画像をK-meansでクラスタリング
- 28×28の画像（784次元）



https://www.kaggle.com/c/digit-recognizer/data

練習問題：K-means

```python
import os
infolder0 = './numbers/0/'
infolder1 = './numbers/1/'
infolder2 = './numbers/2/'

data=[]
IDs = []
# 0
data0=[]
infiles0 = os.listdir(infolder0)
for i in range(len(infiles0)):
    infile = infolder0 + infiles0[i]
    im = io.imread(infile)
    N,M = im.shape
    im = im.reshape(im.size)
    a=[]
    for j in range(len(im)):
        a.append(im[j])
    data0.append(a)
    data.append(a)
    IDs.append(0)
```

```python
# 1
data1=[]
infiles1 = os.listdir(infolder1)
for i in range(len(infiles1)):
    infile = infolder1 + infiles1[i]
    im = io.imread(infile)
    N,M = im.shape
    im = im.reshape(im.size)
    a=[]
    for j in range(len(im)):
        a.append(im[j])
    data1.append(a)
    data.append(a)
    IDs.append(1)

# 2
data2=[]
infiles2 = os.listdir(infolder2)
for i in range(len(infiles2)):
    infile = infolder2 + infiles2[i]
    im = io.imread(infile)
    N,M = im.shape
    im = im.reshape(im.size)
    a=[]
    for j in range(len(im)):
        a.append(im[j])
    data2.append(a)
    data.append(a)
    IDs.append(2)
```

練習問題：K-means

```
# clustering
# kの決定
K = 3;

# 初期化
# random値の範囲決定
vlist = np.array(data)
CList = np.round(np.c_[255*np.random.rand(K,N*M)])
orgCList = CList

Err = 0.1
maxittr = 100
itt = 0
while  itt < maxittr:
    end_flag = True
    preCList = CList
    CIList = selectCluster(vlist,CList) # 指示クラスタの決定
    CList = updataCenter(vlist,CIList,K) # クラスタ中心の更新
    # 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)
        if dd > Err:
            end_flag = False
    itt += 1
    if end_flag == True:
        break
```

画素数N*M次元のK個のベクトルを値をランダムに作成

規定回数まで繰り返し

支持クラスタの決定

クラスタ中心の更新

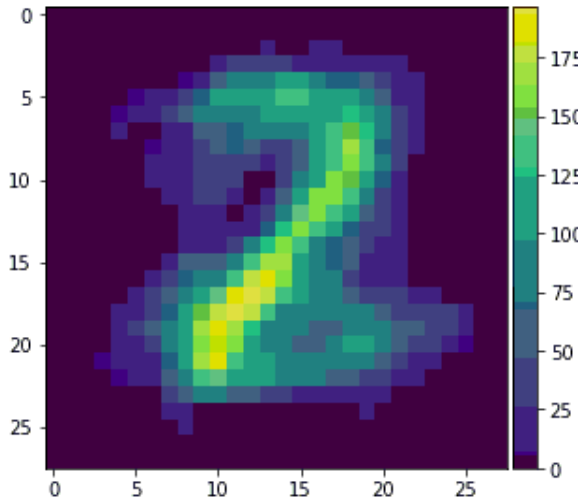
クラスタ中心の移動距離の算出

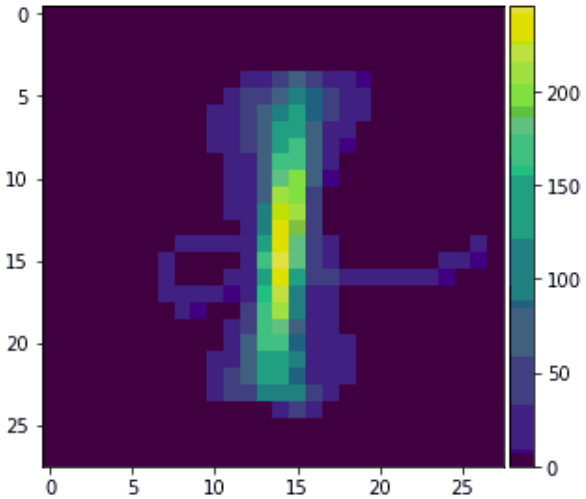
クラスタ中心の移動距離が一定以下のとき、終了

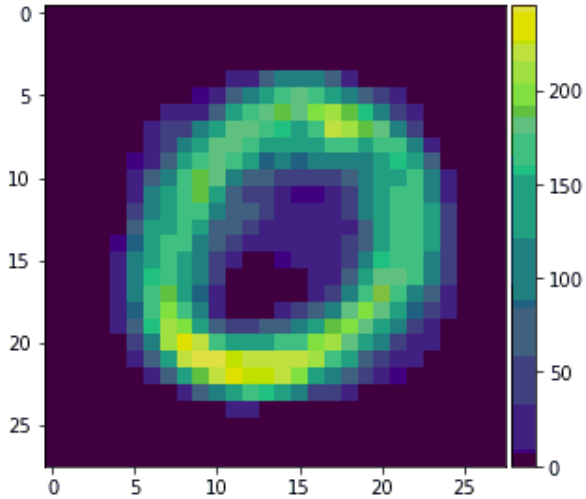
練習問題：K-means結果

- クラスタリングの代表画像の可視化

```
# visualization
cim0 = CList[0]
cim1 = CList[1]
cim2 = CList[2]
cim0 = cim0.reshape(N,M)
cim1 = cim1.reshape(N,M)
cim2 = cim2.reshape(N,M)

io.imshow(cim0)
plt.show()
io.imshow(cim1)
plt.show()
io.imshow(cim2)
plt.show()
```



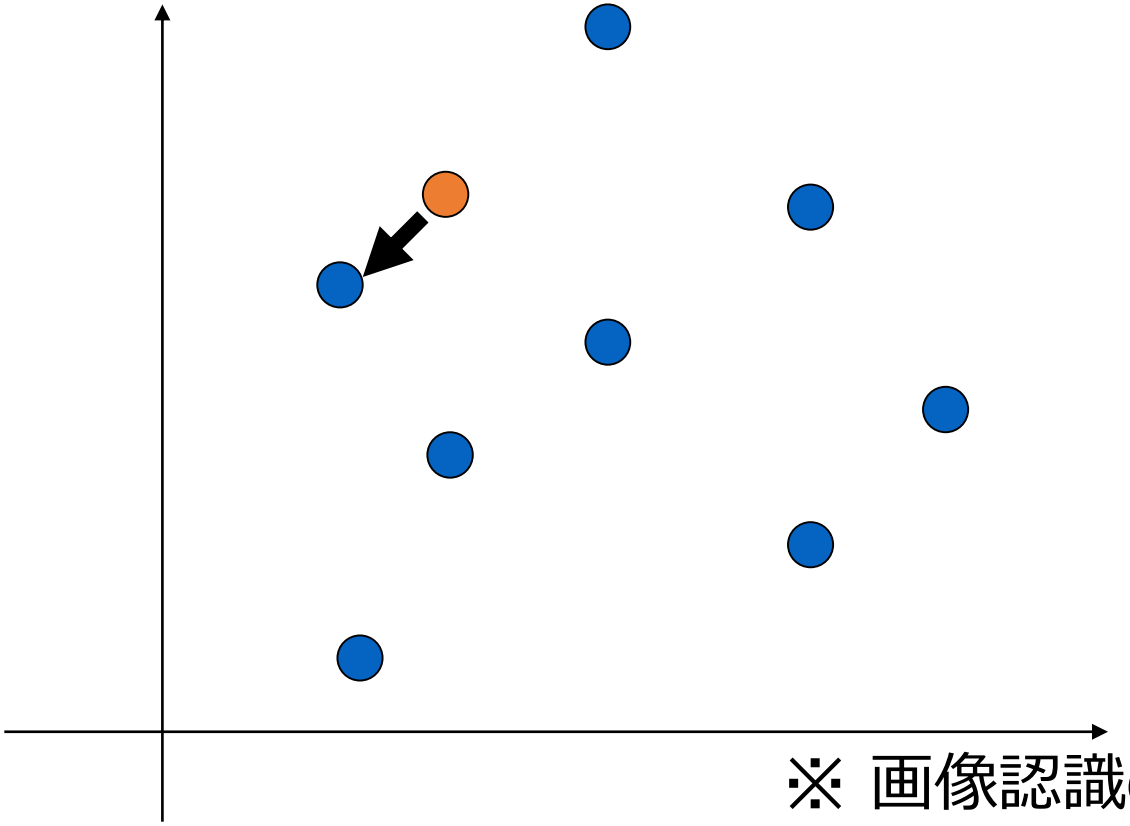




画像空間で遊ぶ

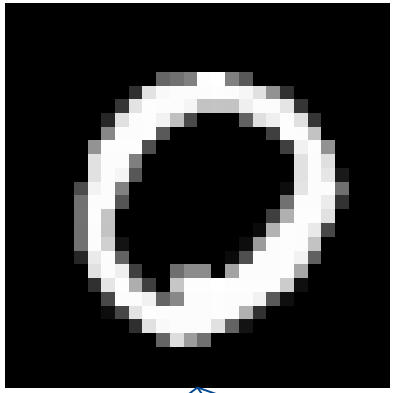
- 最も近い画像を探す (最近傍探索)

search for the most similar image (the nearest neighbor)



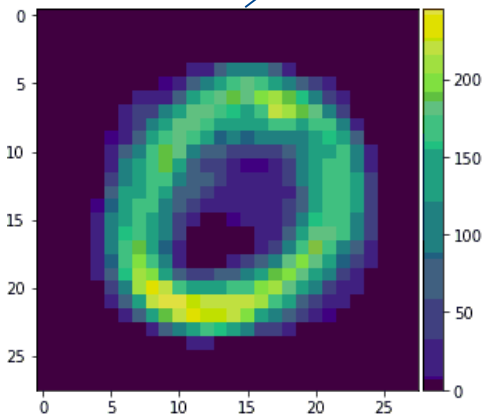
※ 画像認識の原理

最近傍探索

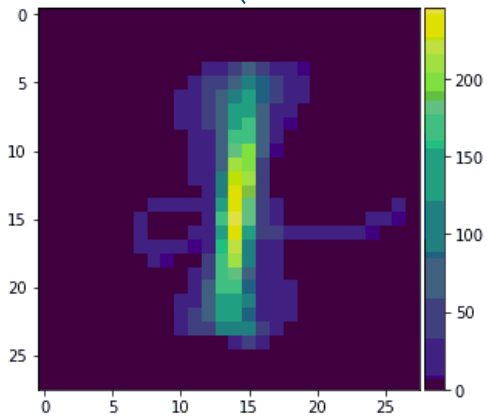


```
np.linalg.norm(cim0-im0)
np.linalg.norm(cim1-im0)
np.linalg.norm(cim2-im0)
```

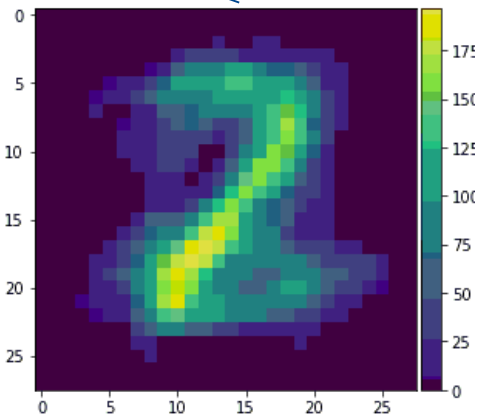
1373



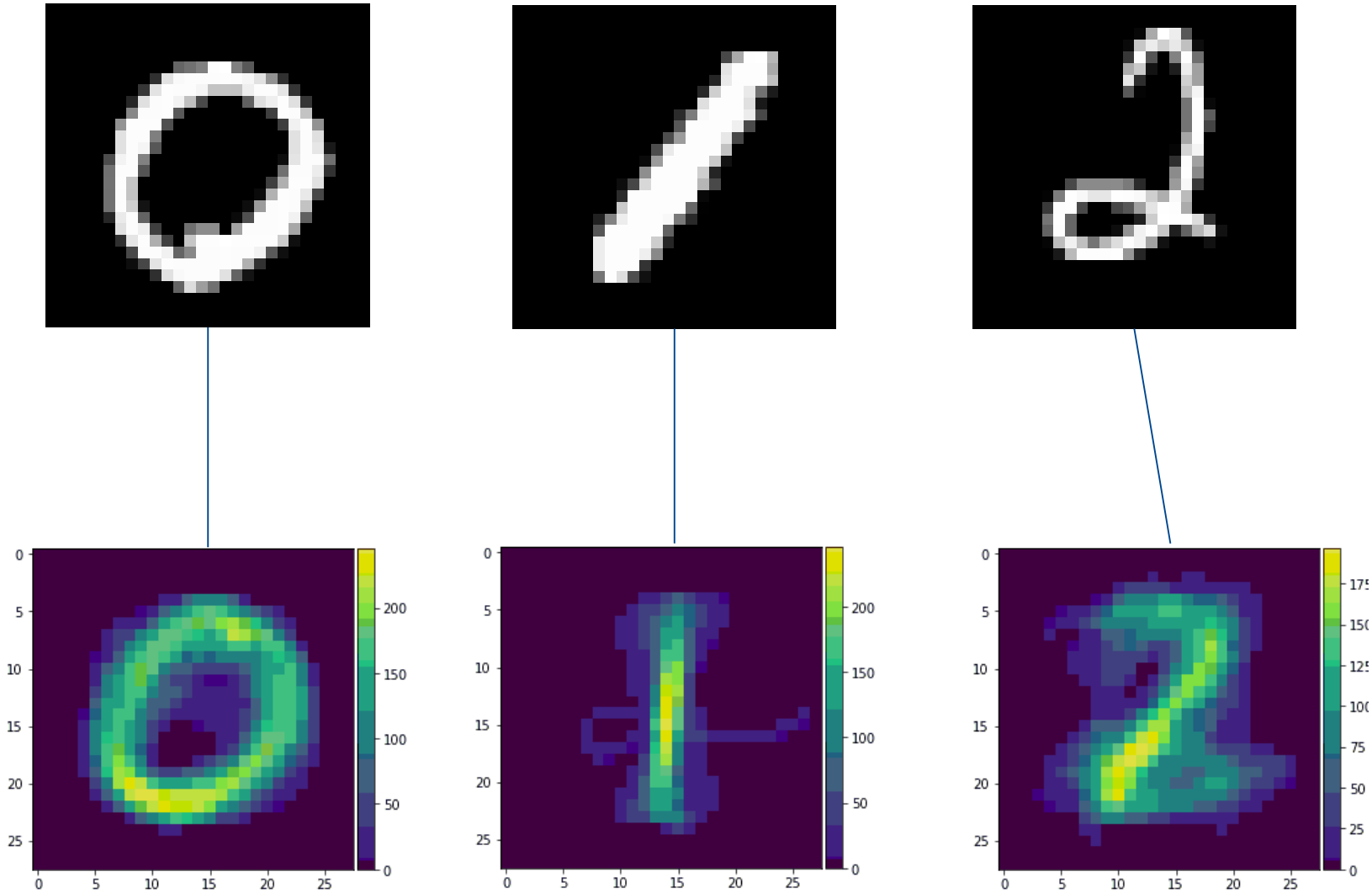
2643



2650



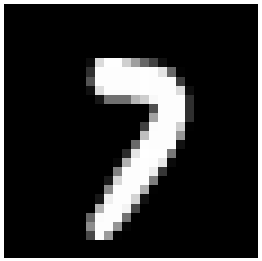
最近傍探索

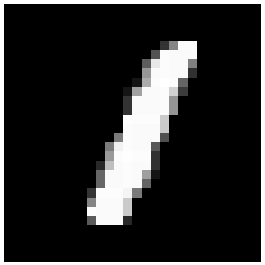


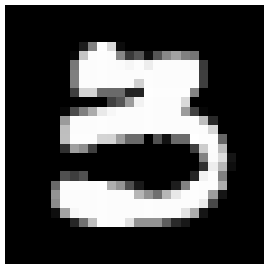
練習3

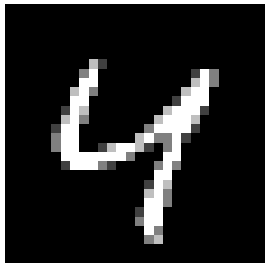
- “numbers/7/number029.tif”,“numbers/1/number1-08.tif”, “numbers/3/number015.tif”, “numbers/4/number014.tif”の画像を読み込んで

- “7” の画像の最近傍探索を行う









画像応用
画像情報の定量化＋データ解析

コインの粒度分布測定

- 画像中のコインのサイズ分布を計算しよう！
  - コインをセグメンテーションして、コインの大きさを定量化
  - 2つの画像のコインサイズを比べよう

coins1

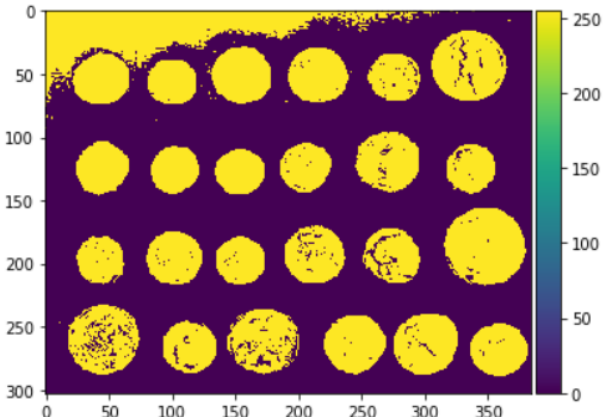
coins2

http://scikit-image.org/docs/stable/auto_examples/

Segmentation

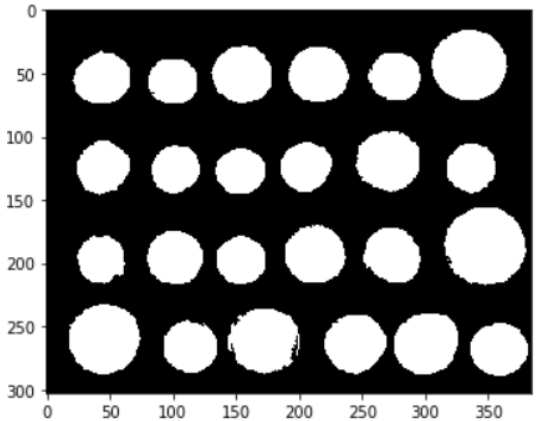
- 閾値処理＋Fill

```
#Segmentation
segmented_img = np.zeros(coins1.shape)
mask = coins[:,:] > 0.43*255
segmented_img[mask] = 255
# fill
from scipy import ndimage as ndi
fill_coins = ndi.binary_fill_holes(segmented_img)
```



- 小さい領域（ノイズ）の除去

```
# labeling and remove small and large blobs
from scipy import ndimage as ndi
label_objects, nb_labels = ndi.label(fill_coins)
sizes = np.bincount(label_objects.ravel())
print(sizes)
mask_sizes = np.zeros(sizes.shape)
for i in range(len(sizes)):
  mask_sizes[i] = 100<sizes[i] and sizes[i]<5000
labeled_coins1 = mask_sizes[label_objects]
labeled_coins1, N1 = ndi.label(labeled_coins1)
```



サイズを測定

```
# size を測定
N = np.max(labeled_coins1)
sizeList1 = []
sizeTh = 500
for i in range(1,N+1):
    ix,iy = np.where( labeled_coins1 == i )
    if len(ix)<sizeTh:
        continue
    sizeList1.append(len(ix))

N = np.max(labeled_coins2)
sizeList2 = []
for i in range(1,N+1):
    ix,iy = np.where( labeled_coins2 == i )
    if len(ix)<sizeTh:
        continue
    sizeList2.append(len(ix))
```

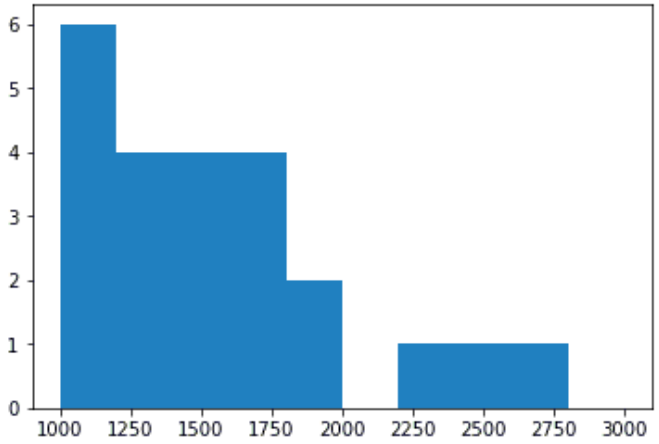
ラベルごとに座標を取得

小さすぎるのはノイズとして除去

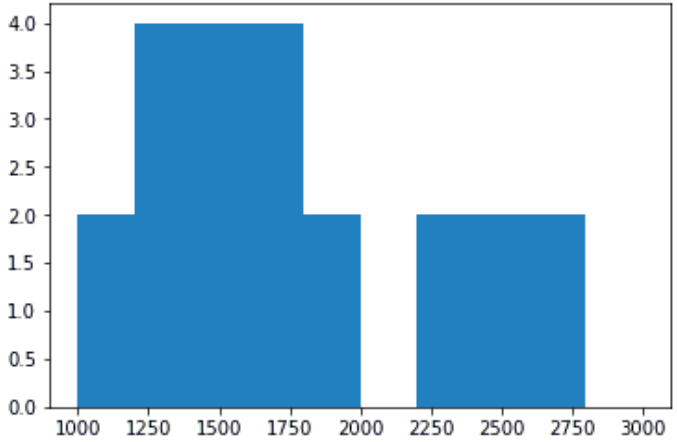
小さすぎるのはノイズとして除去

サイズの分布

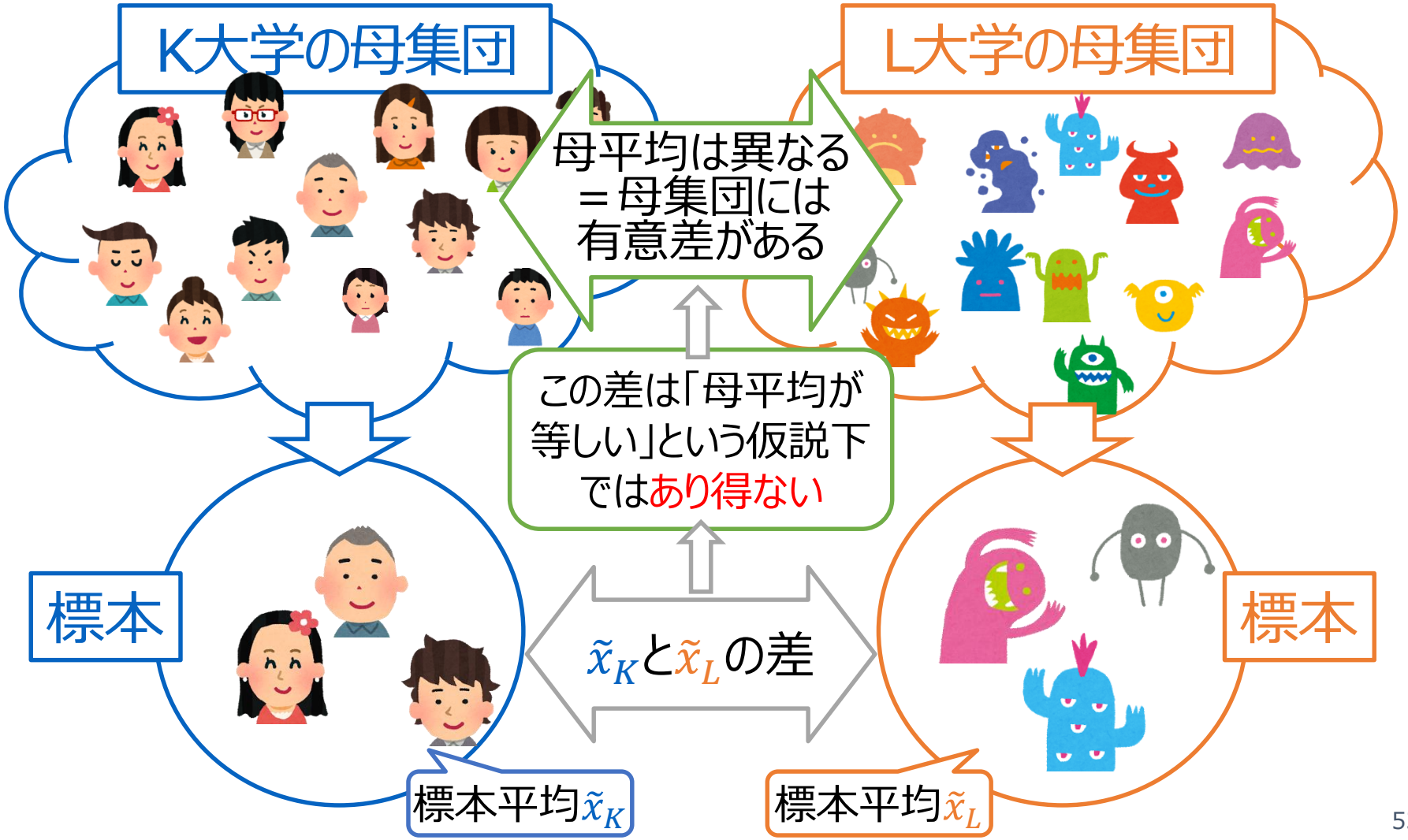
```
# mean and variance
print 'coins1:\n mean:',np.mean(sizeList1),', var:',np.var(sizeList1)
print 'coins2:\n mean:',np.mean(sizeList2),', var:',np.var(sizeList2)

# histogram
plt.hist(sizeList1,range=[1000,3000])
plt.show()
plt.hist(sizeList2,range=[1000,3000])
plt.show()
```



coins1:
 mean: 1622.625 , var: 286133.567708
coins2:
 mean: 1863.0 , var: 374745.5



平均の差の検定～実際の攻め方

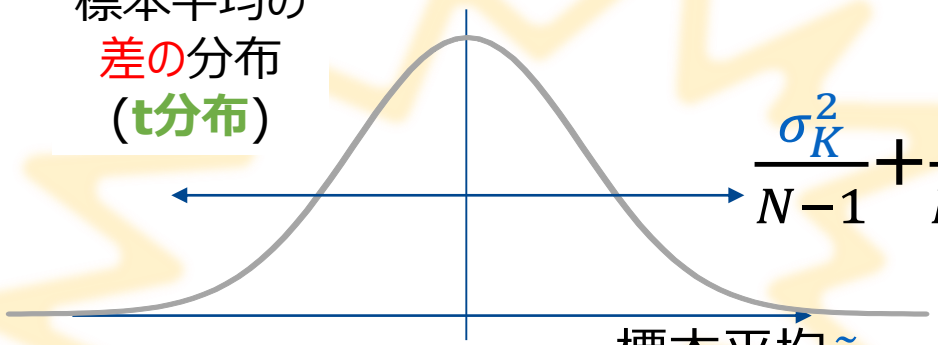


標本平均の差をテストしたいんです：t分布する二つの値の「差」も，やはりt分布

K大学

L大学

標本平均の
差の分布
(**t分布**)



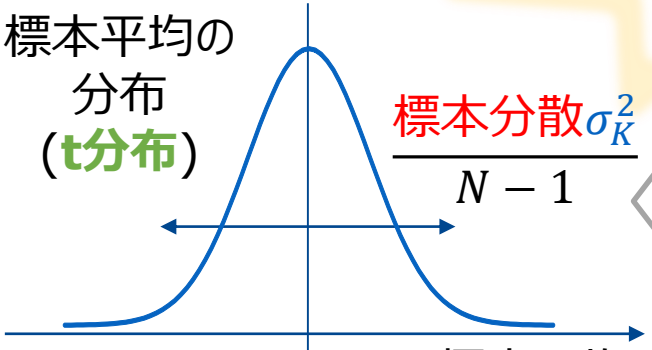
$$\frac{\sigma_K^2}{N-1}+\frac{\sigma_L^2}{N-1}$$

$\mu_K - \mu_L$

標本平均$\tilde{x}_K - \tilde{x}_L$

標本平均の
分布
(**t分布**)

$$\frac{\text{標本分散}\sigma_K^2}{N-1}$$



μ_K

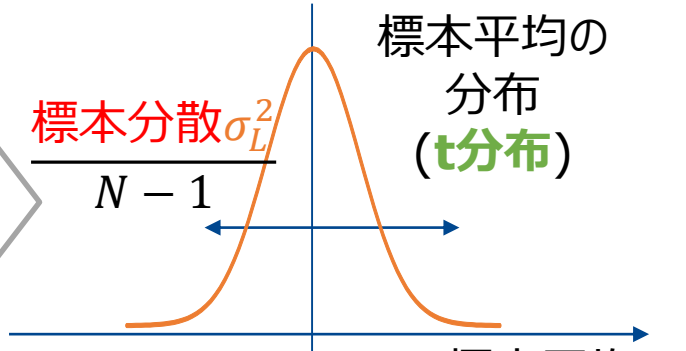
標本平均$\tilde{x}_K$

母平均(未知)

$\tilde{x}_K$と$\tilde{x}_L$の
差をチェックしたい

$$\frac{\text{標本分散}\sigma_L^2}{N-1}$$

標本平均の
分布
(**t分布**)



μ_L

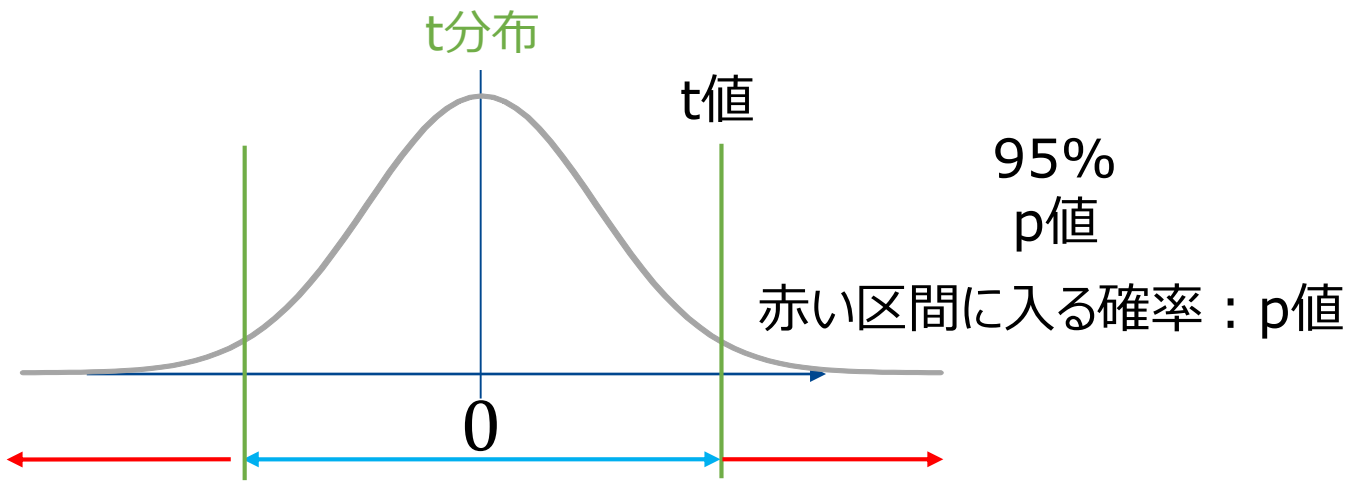
標本平均$\tilde{x}_L$

母平均(未知)

Python で t 検定

- Python で実行するのはとても簡単！

1. 差があるか確認したいデータを２つ用意
2. 有意水準を決定：5%
3. 2つのデータを関数に放り込む
4. t, p値が出る
5. p値が0.05(5%)以下なら、有意に差がある



サイズの差の検定（t検定）

```
# t 検定
from scipy import stats
t, p = stats.ttest_ind(sizeList1, sizeList2,equal_var=False)
print t
print p
```

t値：-1.5409266009999318
P値：0.13035168300344802

5%棄却域で
有意差なし

演習問題

演習１：テンプレートマッチング

- 最も近い画像を探す (最近傍探索)

search for the most similar image (the nearest neighbor)

coins1.tif

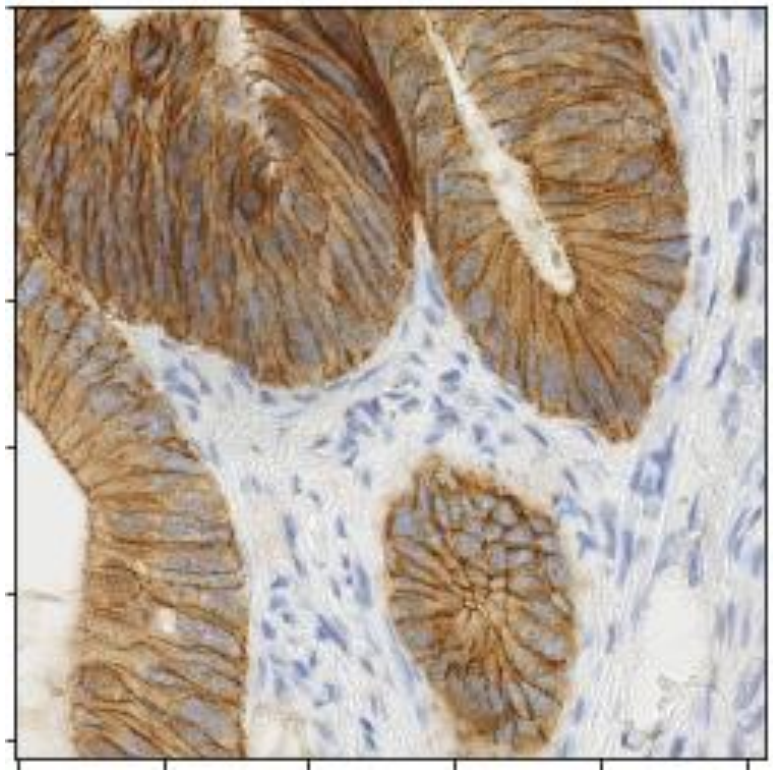
- クエリ―画像と最も近い領域を探す

クエリー
画像

coin_template.tif

演習２：k-means

- data.immunohistochemistry()で読み込んだ画像の色を3色にクラスタリングしよう！

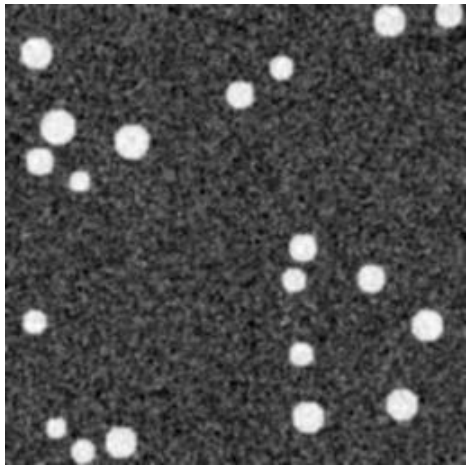


演習３：粒度分布測定

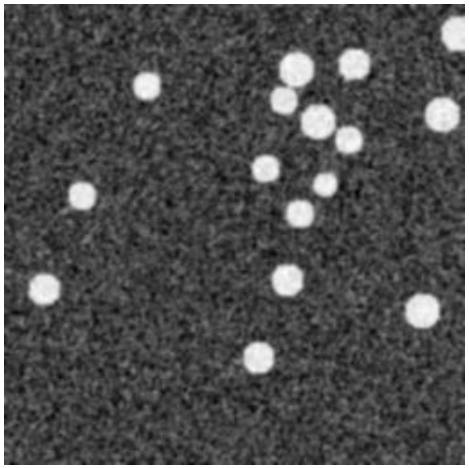
- 画像中の粒子のサイズ分布を計算しよう！
  - 粒子をセグメンテーションして、大きさを定量化
  - 2つの画像のサイズの差の検定を行おう
  - ヒント：閾値は170(particles1)、200(particles2)を使おう

※）切れない場合でも取り合えず、そのまま検定してみよう。

particles1.tif

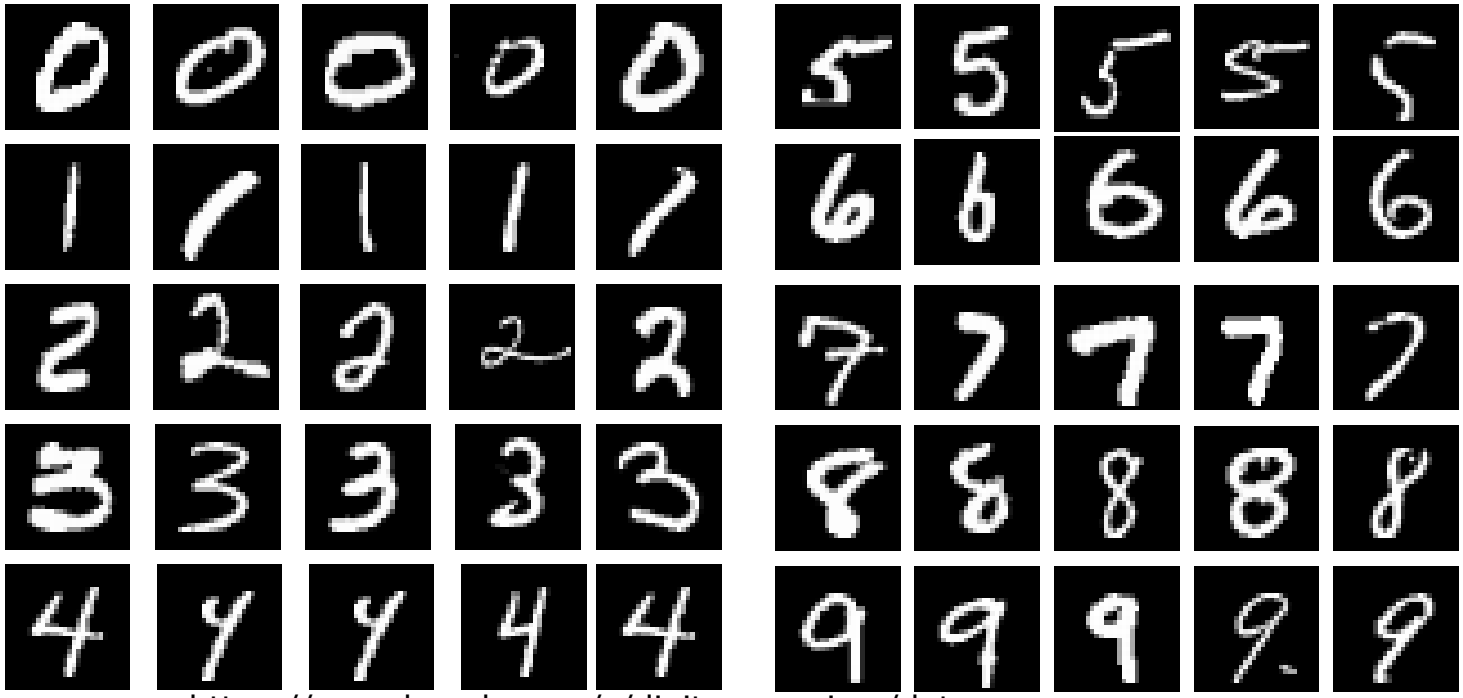


particles2.tif



演習4：主成分分析

- 0から9の手書き文字画像を主成分分析
- 28×28の画像（784次元）



https://www.kaggle.com/c/digit-recognizer/data