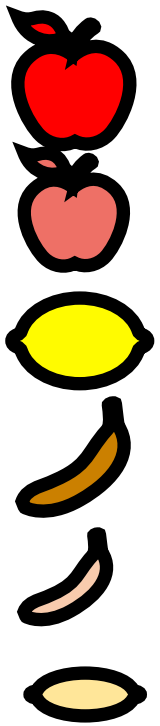
医療とAI・ビッグデータ応用

教師なし機械学習2
(クラスタリング)

統合教育機構
須藤毅顕

前回の教師なし機械学習（次元削減）



	色合い	大きさ	甘さレベル
りんご1	10	100	9
りんご2	8	88	6
れもん1	9	80	2
バナナ1	5	73	7
バナナ2	7	50	4
れもん2	6	40	1

入力

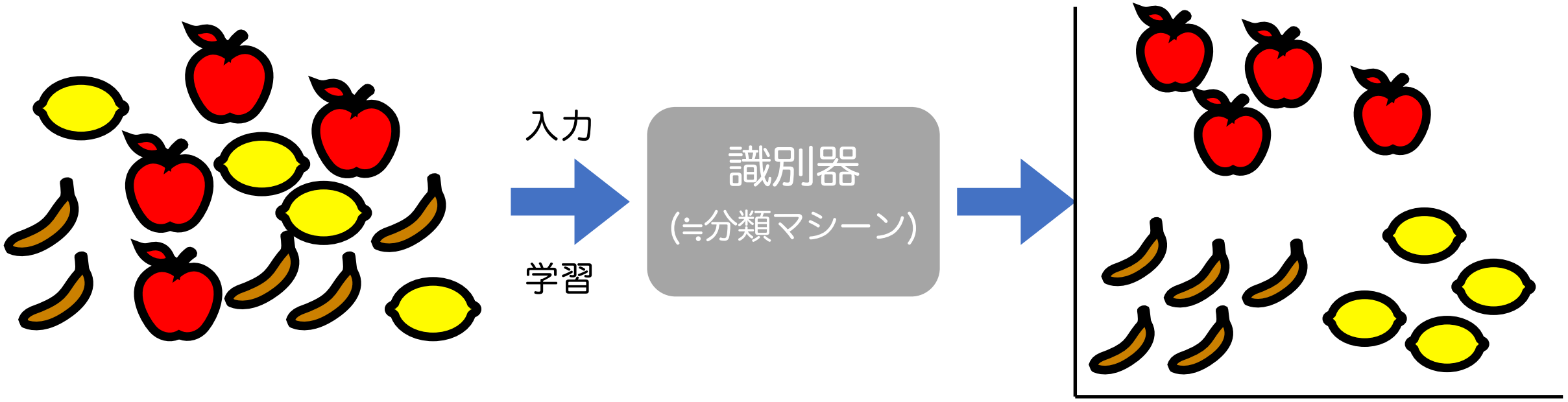
学習

識別器
(≒分類マシーン)

	果物の評価
りんご1	9
りんご2	6
れもん1	2
バナナ1	7
バナナ2	5
れもん2	1

教師なし学習は色合い、大きさ、甘さレベルという特徴から果物の評価という
新たな1つの情報を作り出す

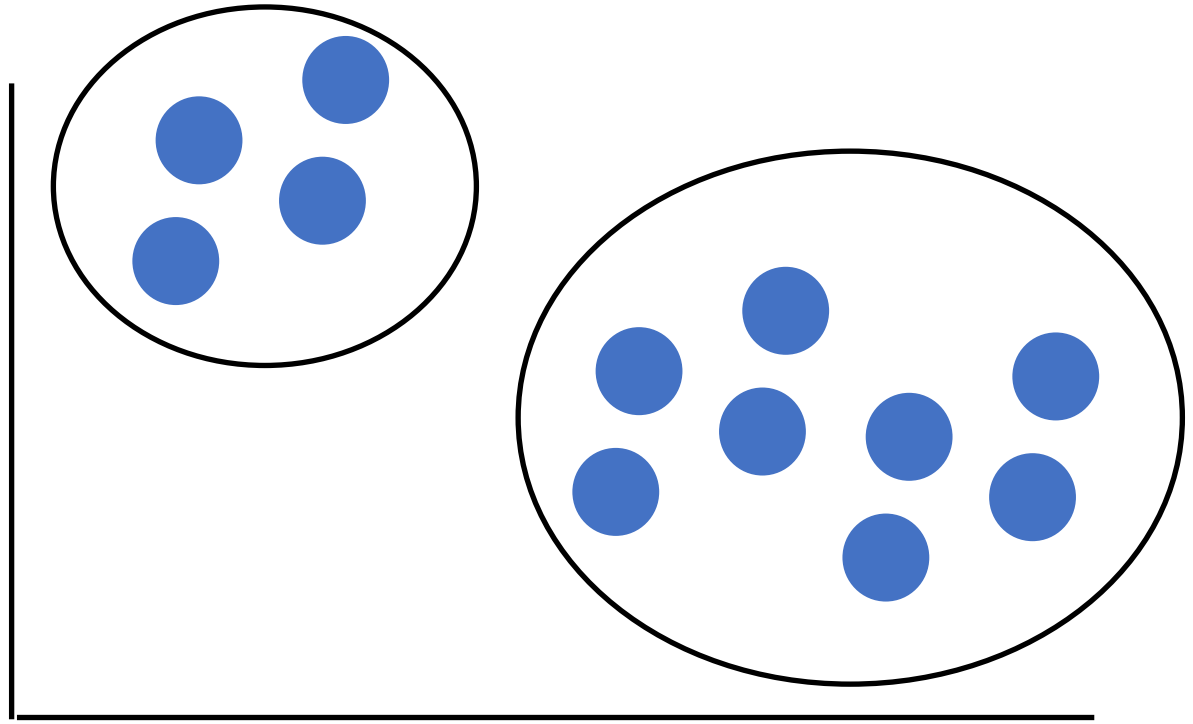
今回の教師なし機械学習（クラスタリング）



教師なし学習は正解を与えず学習させて識別器を作る(法則を見つけさせる)

クラスタリング

データの似ているもの同士でグループ分けする手法

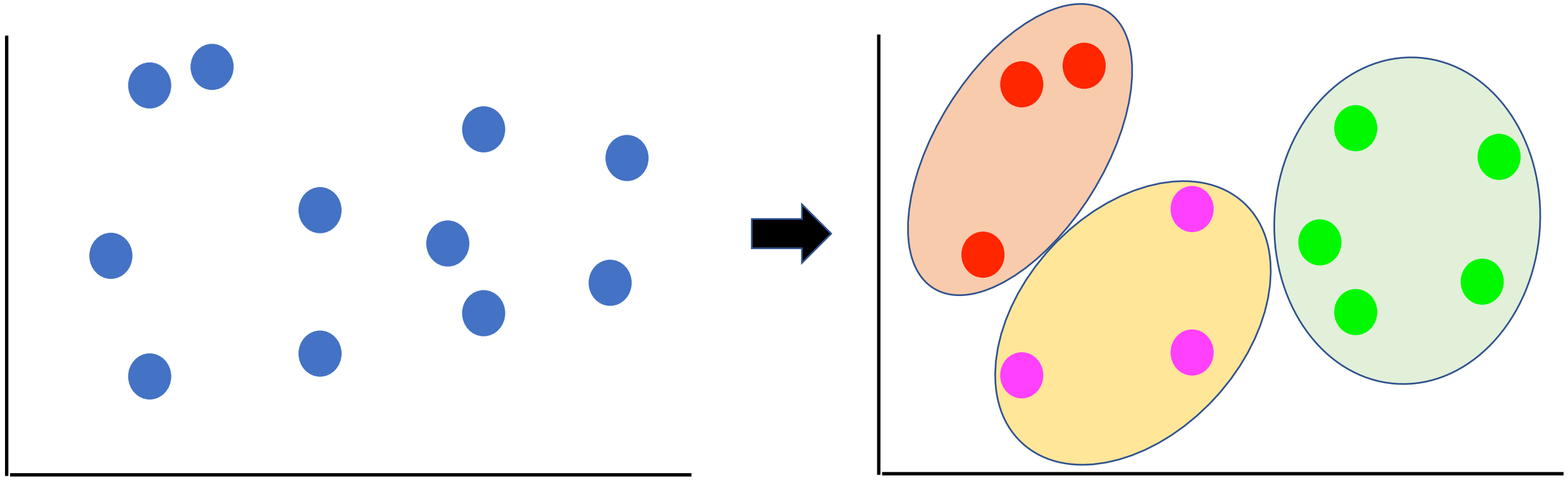


どのような法則でクラスタリングを行うか

k-means 法

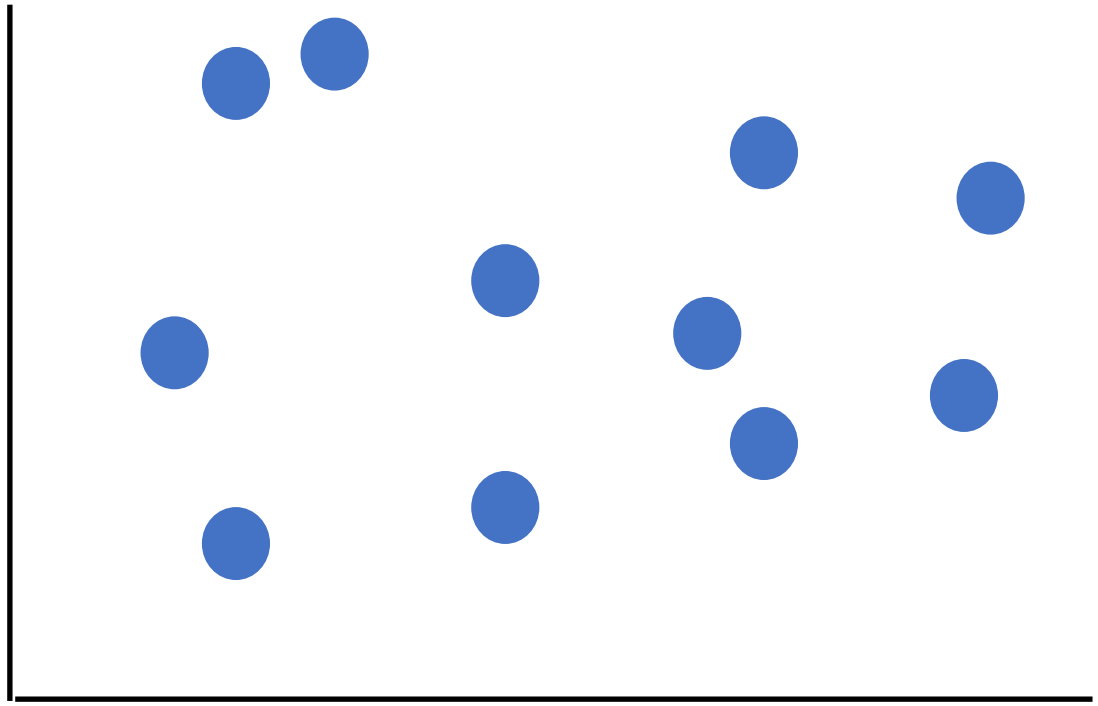
クラスタリングでも最も基本的なアルゴリズム

k個のクラスタ(グループ)を作成するのでk-means法と呼ばれる



k-means 法

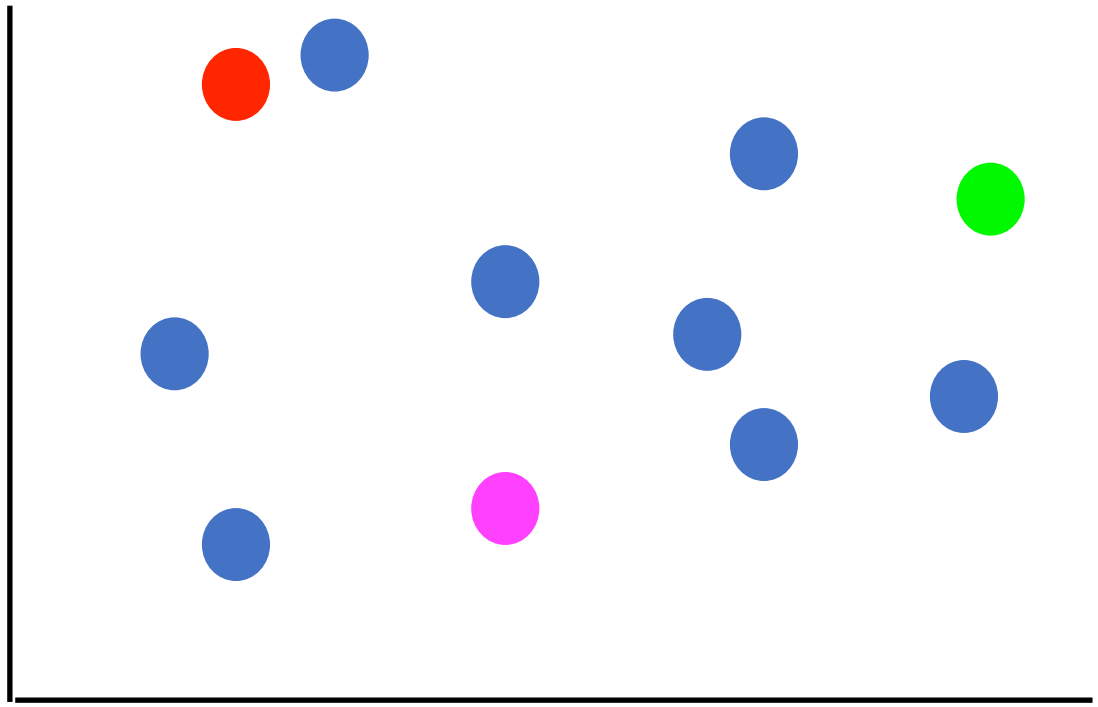
①分けるクラスタ数を決定する(今回は３とする)



k-means 法

①分けるクラスタ数を決定する(今回は３とする)

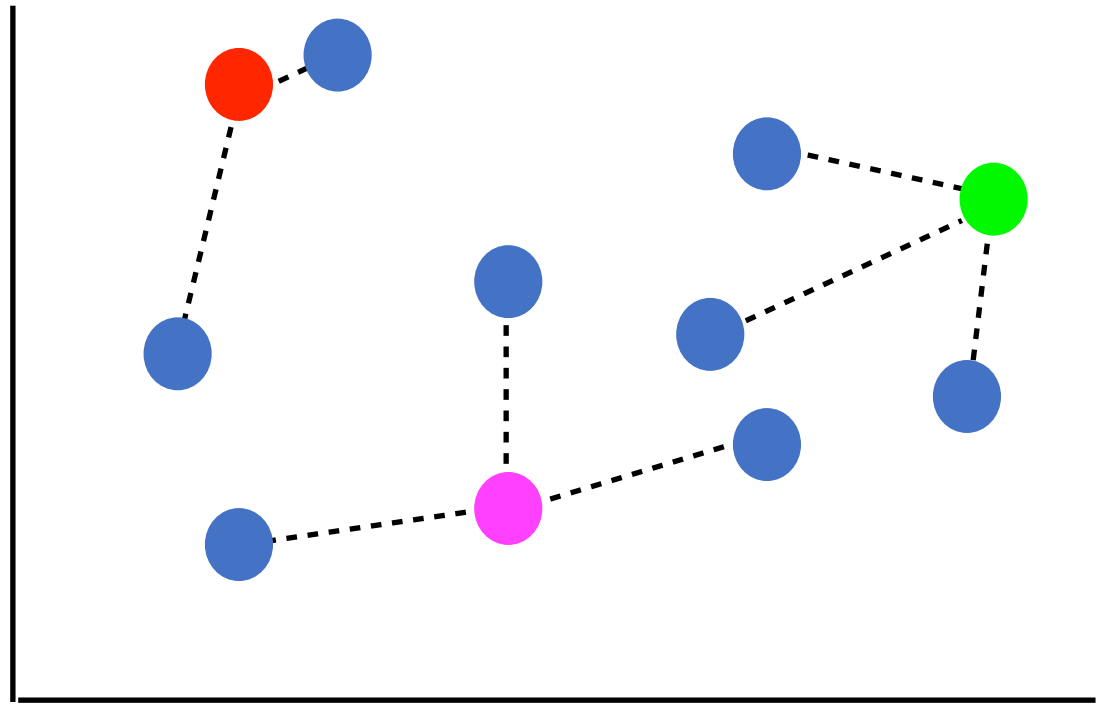
②ランダムにクラスタの個数分データを選ぶ



この３点を代表点とする

k-means 法

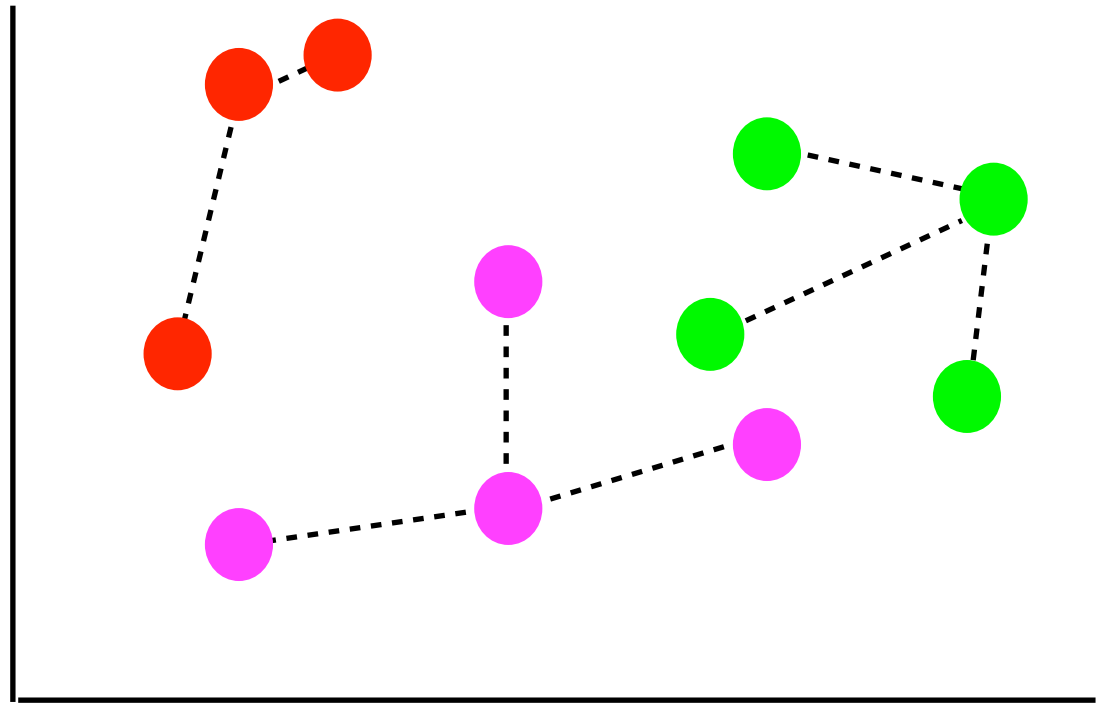
③データを各代表点の距離をもとに各クラスタに所属させる



各データは一番近い距離の代表点の所属となる

k-means 法

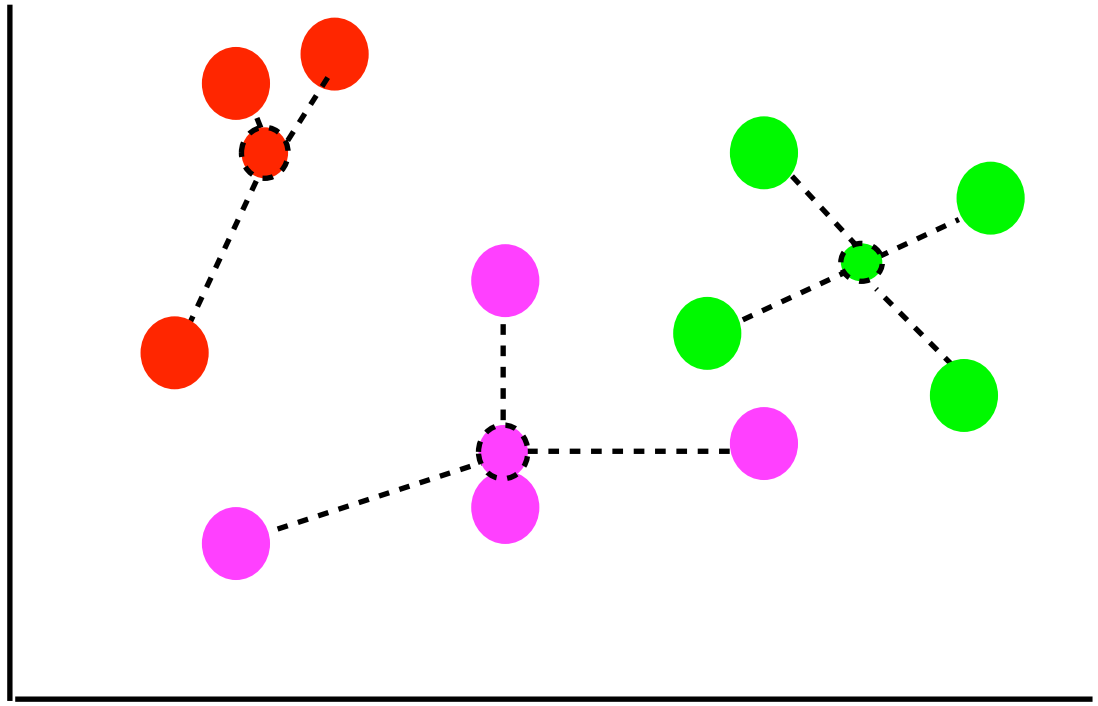
③データを各代表点の距離をもとに各クラスタに所属させる



各データは一番近い距離の代表点の所属となる

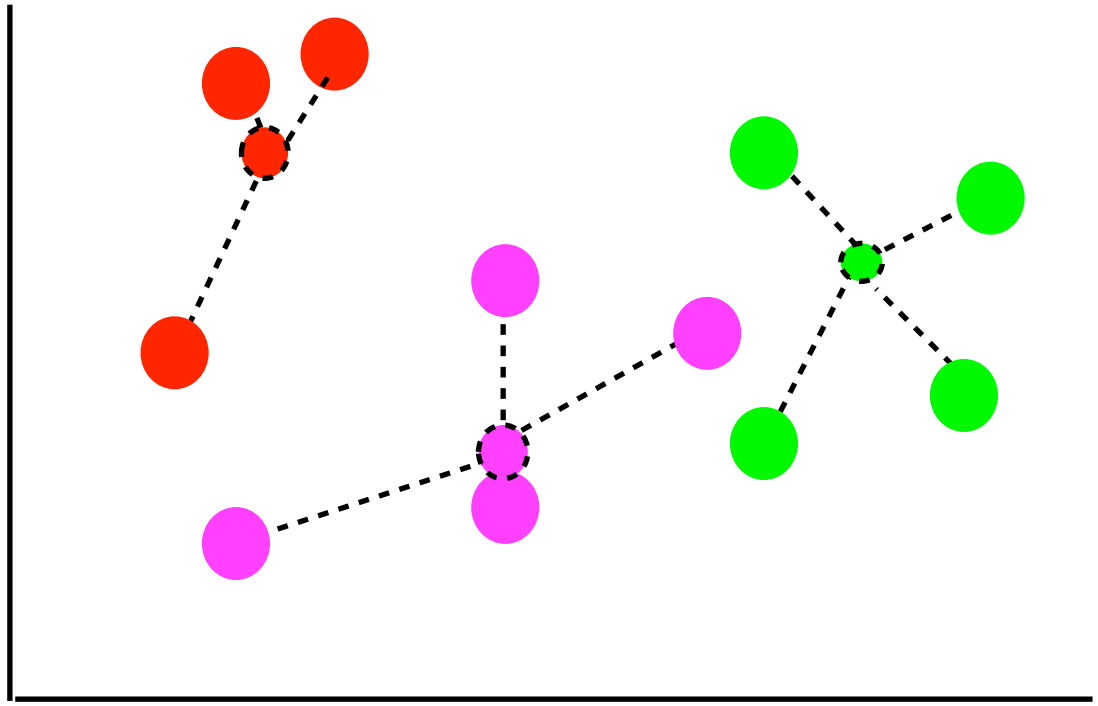
k-means 法

④各クラスタの重心を新たな代表点とする



k-means 法

⑤再度、代表点をもとに所属するクラスタを変えます

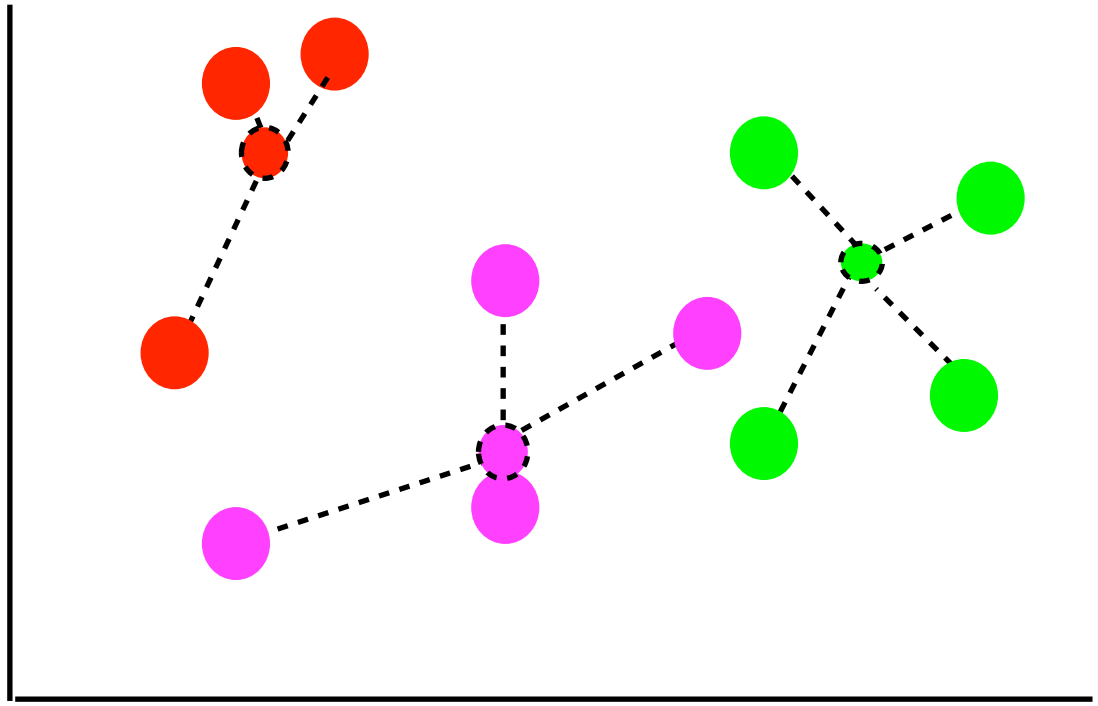


距離が変わるので所属するクラスタが変わります

k-means 法

新たなクラスタに従って再度重心の計算、クラスタの変更を行う

以下、これを繰り返し、代表点が変わらなくなったら終了



アヤメのデータを読み込む

```
from sklearn.datasets import load_iris
iris = load_iris()
iris.data
```

アヤメのデータを読み込む

4つの特徴量
（がく片の長さ、がく片の幅、
花びらの長さ、花びらの幅）
を取り出してiris.dataに代入

がく片の長さ　がく片の幅　花びらの長さ　花びらの幅

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
```

がく片の長さと幅のデータを取得する

がく片の長さと幅だけを取り出す

```
gaku = iris.data[:,0:2]
gaku
```

```
array([[5.1, 3.5],
       [4.9, 3. ],
       [4.7, 3.2],
       [4.6, 3.1],
       [5. , 3.6],
       [5.4, 3.9],
       [4.6, 3.4],
       [5. , 3.4],
       [4.4, 2.9],
       [4.9, 3.1],
       [5.4, 3.7],
       [4.8, 3.4],
       [4.8, 3. ],
       [4.3, 3. ],
       [5.8, 4. ],
       [5.7, 4.4],
       [5.4, 3.9],
       [5.1, 3.5],
       [5.7, 3.8],
       [5.1, 3.8],
       [5.4, 3.4],
```

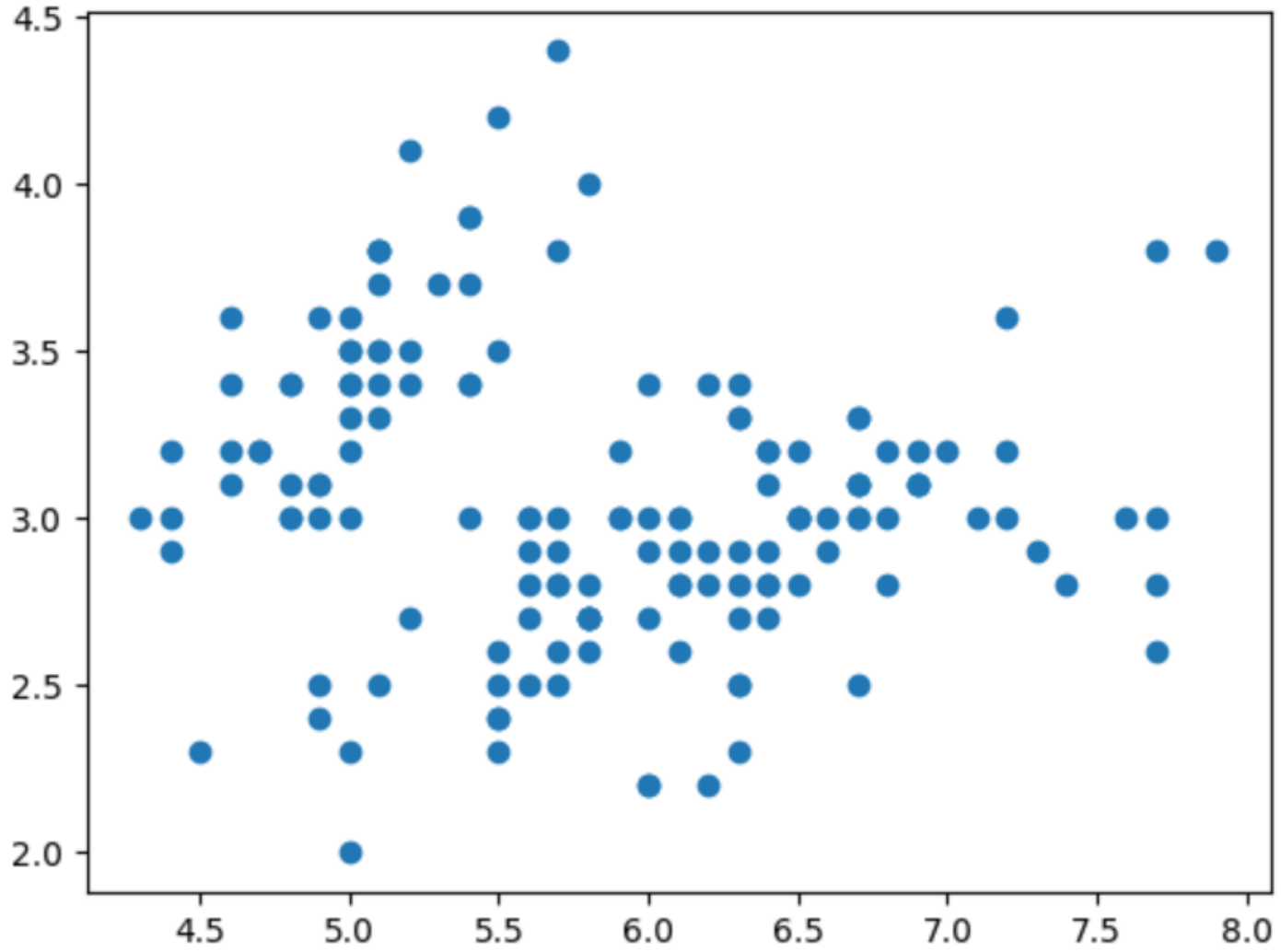
がく片の長さ　がく片の幅　花びらの長さ　花びらの幅

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
```

がく片の長さと幅のデータを図示する

```
import matplotlib.pyplot as plt
plt.scatter(gaku[:,0],gaku[:,1])
plt.show()
```

x軸にがく片の長さ、
y軸にがく片の幅として図示



がく片の長さと幅のデータを図示する

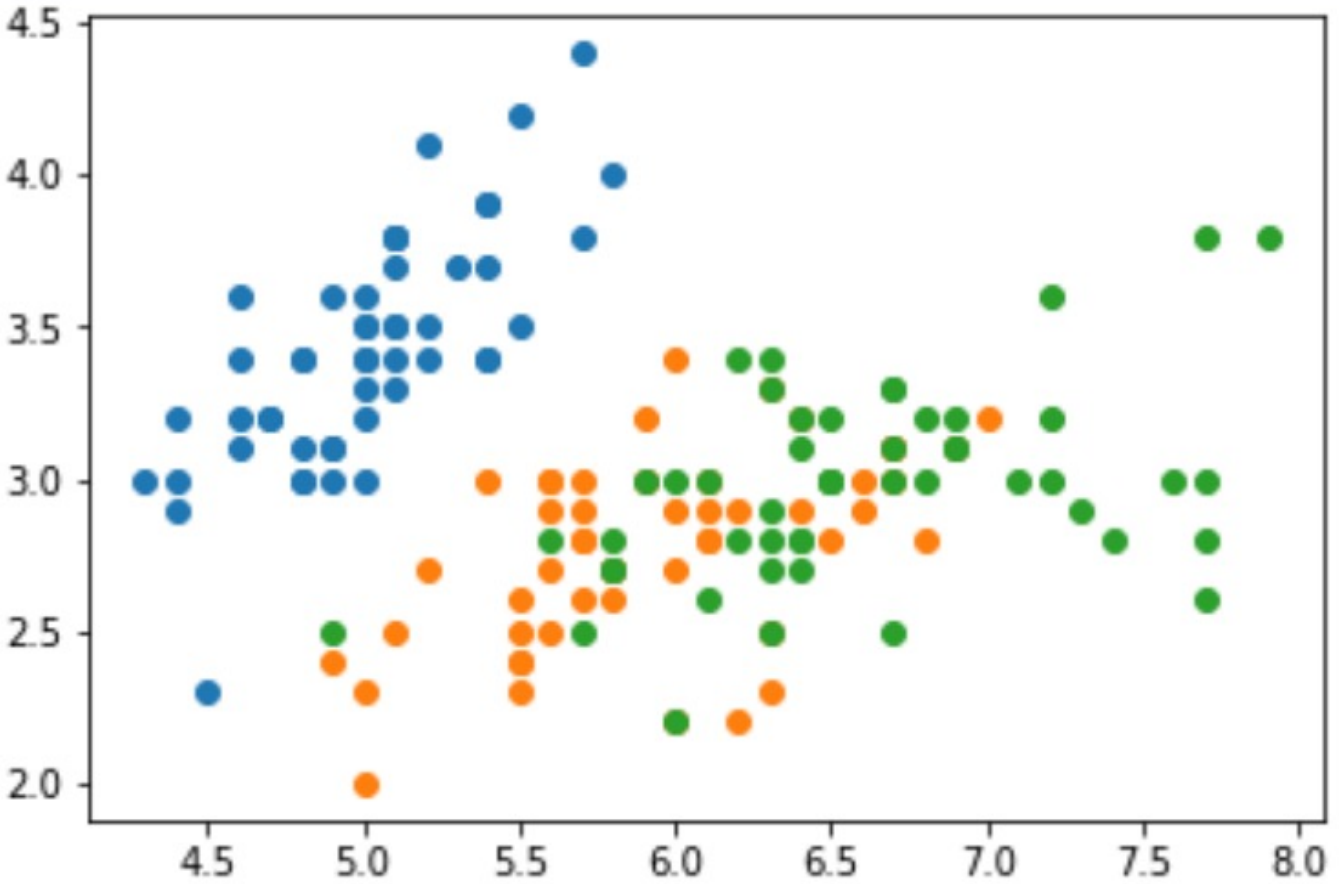
アヤメの種類で分ける

```
import matplotlib.pyplot as plt
plt.scatter(gaku[:,0],gaku[:,1])
plt.show()
```

(全ての行,萼片の長さの列, 全ての行, 萼片の幅の列)

```
import matplotlib.pyplot as plt
plt.scatter(gaku[0:50,0],gaku[0:50:,1])
plt.scatter(gaku[50:100,0],gaku[50:100,1])
plt.scatter(gaku[100:150,0],gaku[100:150,1])
plt.show()
```

(ヒオウギアヤメの萼片の長さ, ヒオウギアヤメの萼片の幅)
(ブルーフラッグの萼片の長さ,ブルーフラッグの萼片の幅)
(バージニカの萼片の長さ, バージニカの萼片の幅)



色を指定しないと勝手に色が
割り当てられます

k-means法を実行する

モデル名=Kmeans()で実行
n_clusters=~~でクラスタリングしたい数を入力する
random_state=0は皆同じ結果になるためのランダムの指定
クラスタリング結果はmodel.predict()で表示される

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3,random_state=0)
model.fit(gaku)
cluster=model.predict(gaku)
print(cluster)
```

```
[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 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 0 2 0 2 0 2 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0
 2 2 2 2 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 0 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
```

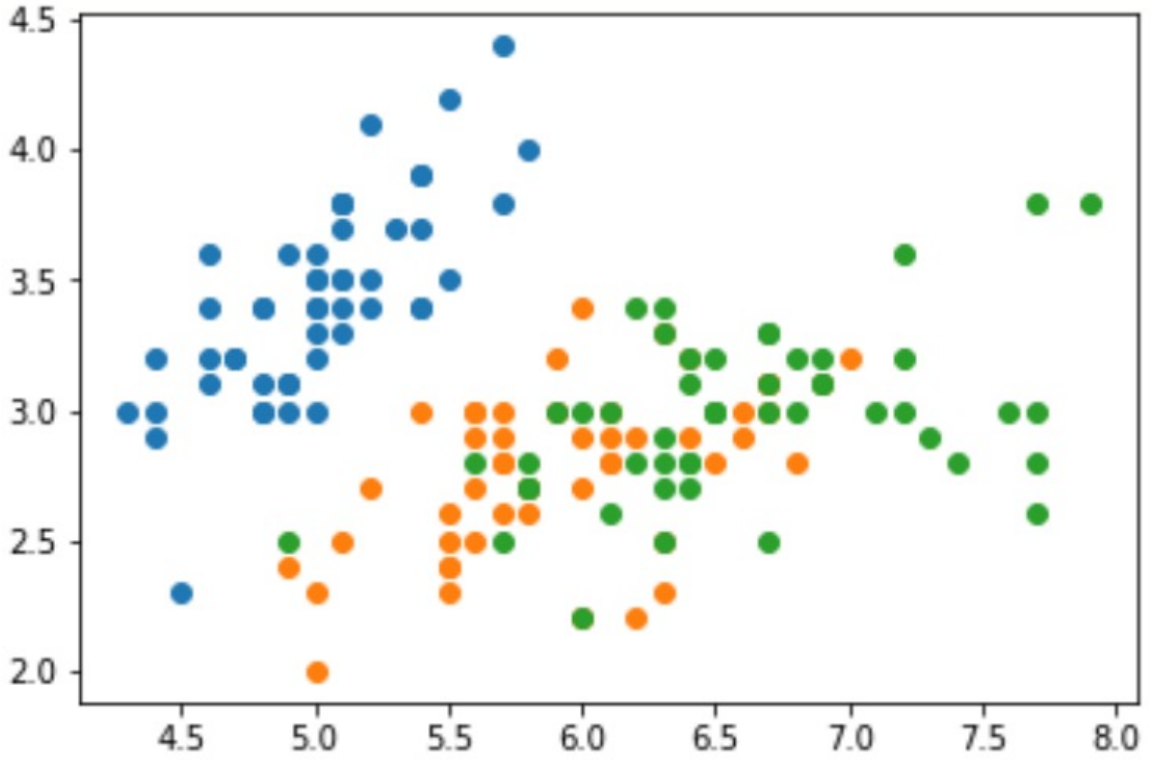
クラスタリングの結果、150個のデータが0，1，2に分けられた。

がく片の長さと幅でクラスタリングを図示

クラスター0,1,2毎にがく片の長さと幅で図示したい

```
import matplotlib.pyplot as plt
plt.scatter(gaku[0:50,0],gaku[0:50:,1])
plt.scatter(gaku[50:100,0],gaku[50:100,1])
plt.scatter(gaku[100:150,0],gaku[100:150,1])
plt.show()
```

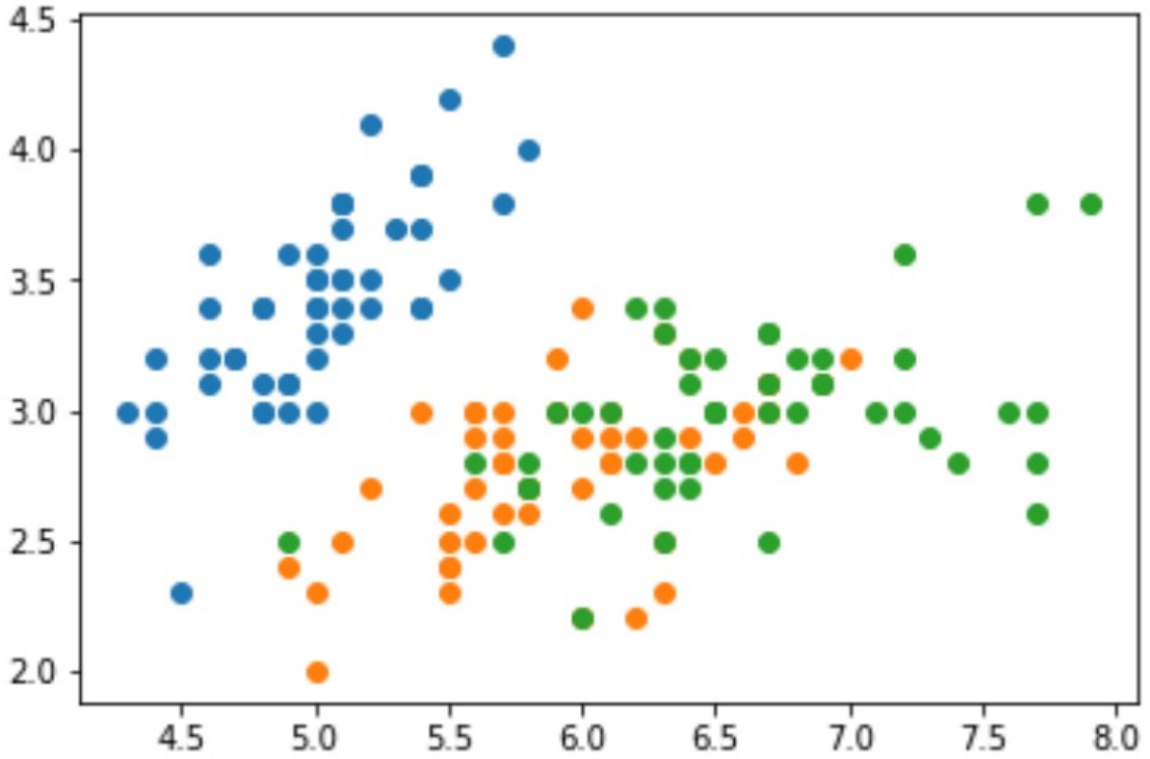
plt.figure()
plt.scatter(クラスタ0のがく片の長さ,クラスタ0のがく片の幅)
plt.scatter(クラスタ1のがく片の長さ,クラスタ1のがく片の幅)
plt.scatter(クラスタ2のがく片の長さ,クラスタ2のがく片の幅)
plt.show()



がく片の長さと幅でクラスタリングを図示

クラスター0,1,2毎にがく片の長さと幅で図示したい

```
import matplotlib.pyplot as plt
plt.scatter(gaku[0:50,0],gaku[0:50:,1])
plt.scatter(gaku[50:100,0],gaku[50:100,1])
plt.scatter(gaku[100:150,0],gaku[100:150,1])
plt.show()
```




```
plt.figure()
plt.scatter(クラスタ0のがく片の長さ,クラスタ0のがく片の幅)
plt.scatter(クラスタ1のがく片の長さ,クラスタ1のがく片の幅)
plt.scatter(クラスタ2のがく片の長さ,クラスタ2のがく片の幅)
plt.show()
```

(gaku[ヒオウギアヤメの行,0], gaku[ヒオウギアヤメの行,1])

(gaku[0:50,0], gaku[0:50,1])

gaku[クラスタ0の行,0], gaku[クラスタ0の行,1])

(gaku[????,0], gaku[????, 1])

がく片の長さと幅でクラスタリングを図示

前回と似たやり方で

```
gaku
```

```
array([[5.1, 3.5],
       [4.9, 3. ],
       [4.7, 3.2],
       [4.6, 3.1],
       [5. , 3.6],
       [5.4, 3.9],
       [4.6, 3.4],
       [5. , 3.4],
       [4.4, 2.9],
       [4.9, 3.1],
       [5.4, 3.7],
       [4.8, 3.4],
       [4.8, 3. ],
       [4.3, 3. ],
       [5.8, 4. ],
       [5.7, 4.4],
       [5.4, 3.9],
       [5.1, 3.5],
       [5.7, 3.8],
       [5.1, 3.8],
       [5.4, 3.4],
       [5.1, 3.7],
       [4.6, 3.6],
```

```
cluster
```

```
array([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, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 2, 0, 2, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 2,
       0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2,
       2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2, 2,
       2, 0, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0], dtype=int32)
```

gakuは150個のアヤメの萼片の長さと幅 (150,2)
clusterは150個のアヤメのクラスタリング結果 (150,)

がく片の長さと幅でクラスタリングを図示

クラスタが0か否か(True or False)

```
cluster == 0
```

```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False,  True,
       False,  True, False,  True, False,  True,  True,  True,  True,
        True,  True, False,  True,  True,  True,  True,  True,  True,
        True,  True, False, False, False, False,  True,  True,  True,
        True,  True,  True,  True,  True, False,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True, False,  True, False, False, False, False,  True, False,
       False, False, False, False, False,  True,  True, False, False,
       False, False,  True, False,  True, False,  True, False, False,
        True,  True, False, False, False, False, False,  True,  True,
       False, False, False,  True, False, False, False,  True, False,
       False, False,  True, False, False,  True])
```

cluster == 0　は150行分ののTrue(クラスタが0)かFalse(クラスタが１か２）の配列になる

がく片の長さと幅でクラスタリングを図示

gakuの全ての行の1列目(がく片の長さ)

```
gaku[:,0]
```

```
array([5.1, 4.9, 4.7, 4.6, 5. , 5.4, 4.6, 5. , 4.4, 4.9, 5.4, 4.8, 4.8,
       4.3, 5.8, 5.7, 5.4, 5.1, 5.7, 5.1, 5.4, 5.1, 4.6, 5.1, 4.8, 5. ,
       5. , 5.2, 5.2, 4.7, 4.8, 5.4, 5.2, 5.5, 4.9, 5. , 5.5, 4.9, 4.4,
       5.1, 5. , 4.5, 4.4, 5. , 5.1, 4.8, 5.1, 4.6, 5.3, 5. , 7. , 6.4,
       6.9, 5.5, 6.5, 5.7, 6.3, 4.9, 6.6, 5.2, 5. , 5.9, 6. , 6.1, 5.6,
       6.7, 5.6, 5.8, 6.2, 5.6, 5.9, 6.1, 6.3, 6.1, 6.4, 6.6, 6.8, 6.7,
       6. , 5.7, 5.5, 5.5, 5.8, 6. , 5.4, 6. , 6.7, 6.3, 5.6, 5.5, 5.5,
       6.1, 5.8, 5. , 5.6, 5.7, 5.7, 6.2, 5.1, 5.7, 6.3, 5.8, 7.1, 6.3,
       6.5, 7.6, 4.9, 7.3, 6.7, 7.2, 6.5, 6.4, 6.8, 5.7, 5.8, 6.4, 6.5,
       7.7, 7.7, 6. , 6.9, 5.6, 7.7, 6.3, 6.7, 7.2, 6.2, 6.1, 6.4, 7.2,
       7.4, 7.9, 6.4, 6.3, 6.1, 7.7, 6.3, 6.4, 6. , 6.9, 6.7, 6.9, 5.8,
       6.8, 6.7, 6.7, 6.3, 6.5, 6.2, 5.9])
```

cluster == 0の時にTrueのデータを取り出す

```
gaku[cluster==0,0]
```

```
array([5.5, 5.7, 4.9, 5.2, 5. , 5.9, 6. , 6.1, 5.6, 5.6, 5.8, 6.2, 5.6,
       5.9, 6.1, 6.3, 6.1, 6. , 5.7, 5.5, 5.5, 5.8, 6. , 5.4, 6. , 6.3,
       5.6, 5.5, 5.5, 6.1, 5.8, 5. , 5.6, 5.7, 5.7, 6.2, 5.1, 5.7, 5.8,
       4.9, 5.7, 5.8, 6. , 5.6, 6.3, 6.2, 6.1, 6.3, 6.1, 6. , 5.8, 6.3,
       5.9])
```

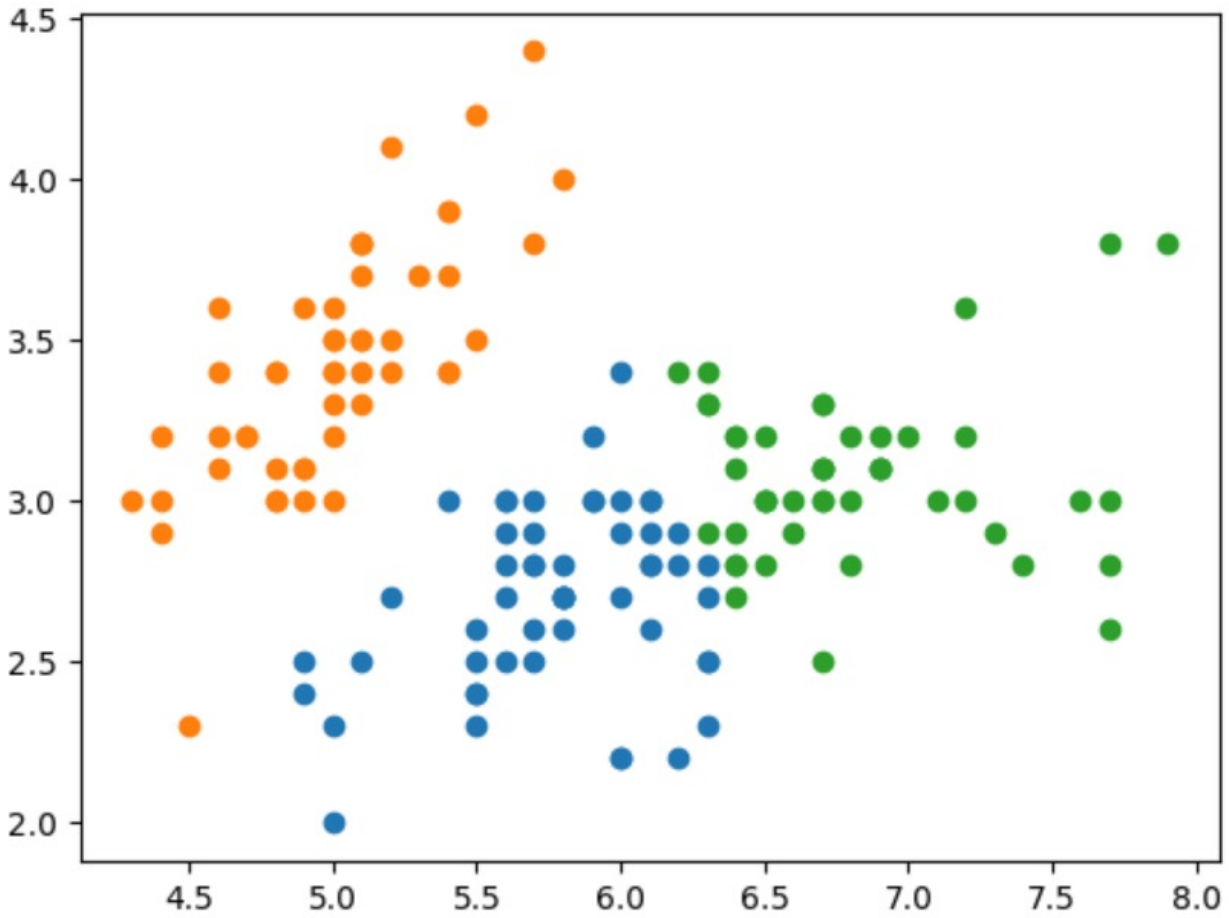
これでクラスタが0の
がく片の長さのみ取り出せた

前回の次元削減の時のスライドも参考にしてください

がく片の長さと幅でクラスタリングを図示

```
plt.scatter(gaku[cluster==0,0],gaku[cluster==0,1])
plt.scatter(gaku[cluster==1,0],gaku[cluster==1,1])
plt.scatter(gaku[cluster==2,0],gaku[cluster==2,1])
plt.show()
```

←クラスタが0の時のがく片の長さ(x)と幅(y)
←クラスタが1の時のがく片の長さ(x)と幅(y)
←クラスタが2の時のがく片の長さ(x)と幅(y)

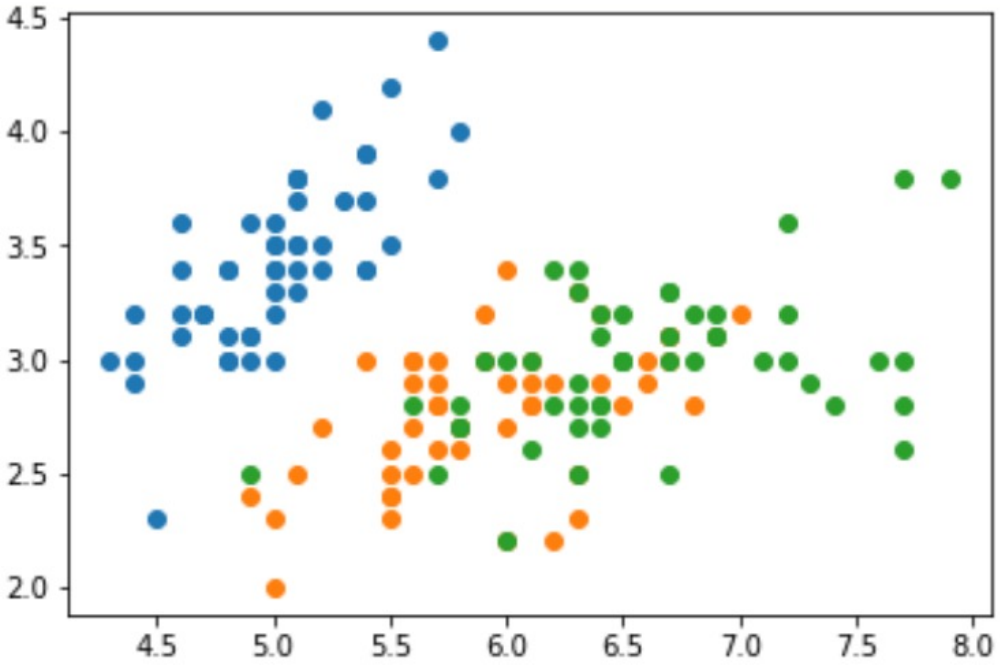


クラスタリングは距離に基づいて
データを決めた個数のグループに分ける

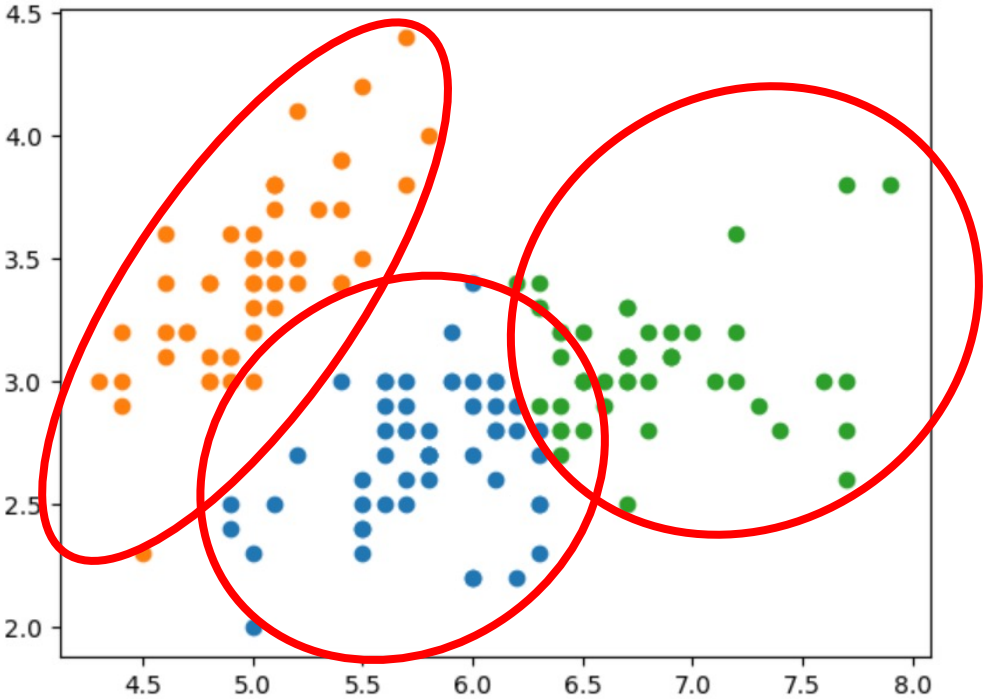
アヤメの正解ラベルを見ずに3つのグループ
に分けられた

がく片の長さと幅でクラスタリングを図示

アヤメの種類で色分け



クラスタリングの結果(クラスタ0,1,2)で色分け



クラスタリングは距離に基づいてデータを決めた個数のグループに分ける

(補足)クラスタの重心を図示

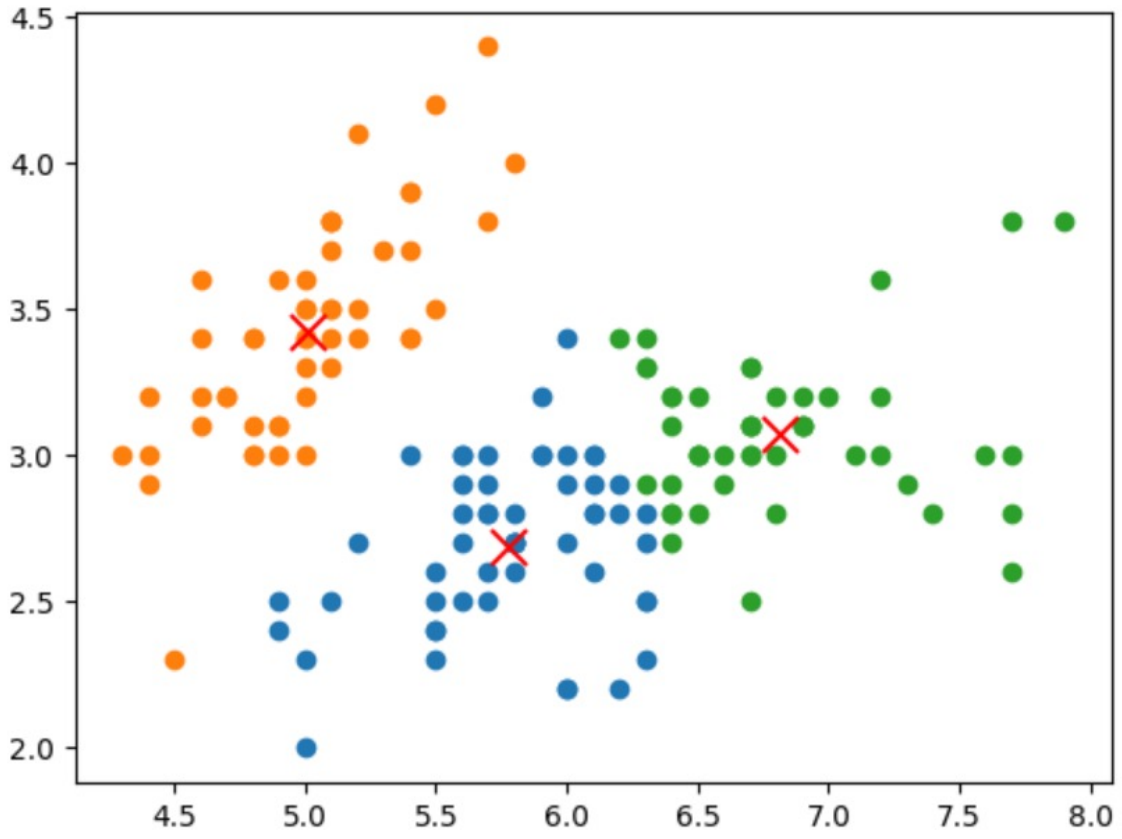
cluster_centers_でクラスタの重心を取得できる

```
model.cluster_centers_
```

```
array([[5.77358491, 2.69245283],
       [5.006     , 3.428     ],
       [6.81276596, 3.07446809]])
```

クラスタ数3にしているので、
(5.77, 2.69),(5.01, 3.43),(6.81, 3.07)
の3点で(3,2)の配列になっている

```
import matplotlib.pyplot as plt
plt.scatter(gaku[cluster==0,0],gaku[cluster==0,1])
plt.scatter(gaku[cluster==1,0],gaku[cluster==1,1])
plt.scatter(gaku[cluster==2,0],gaku[cluster==2,1])
centers = model.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='r', s=150, marker='x')
plt.show()
```



確かに各クラスタの中心に位置していることが確認できる

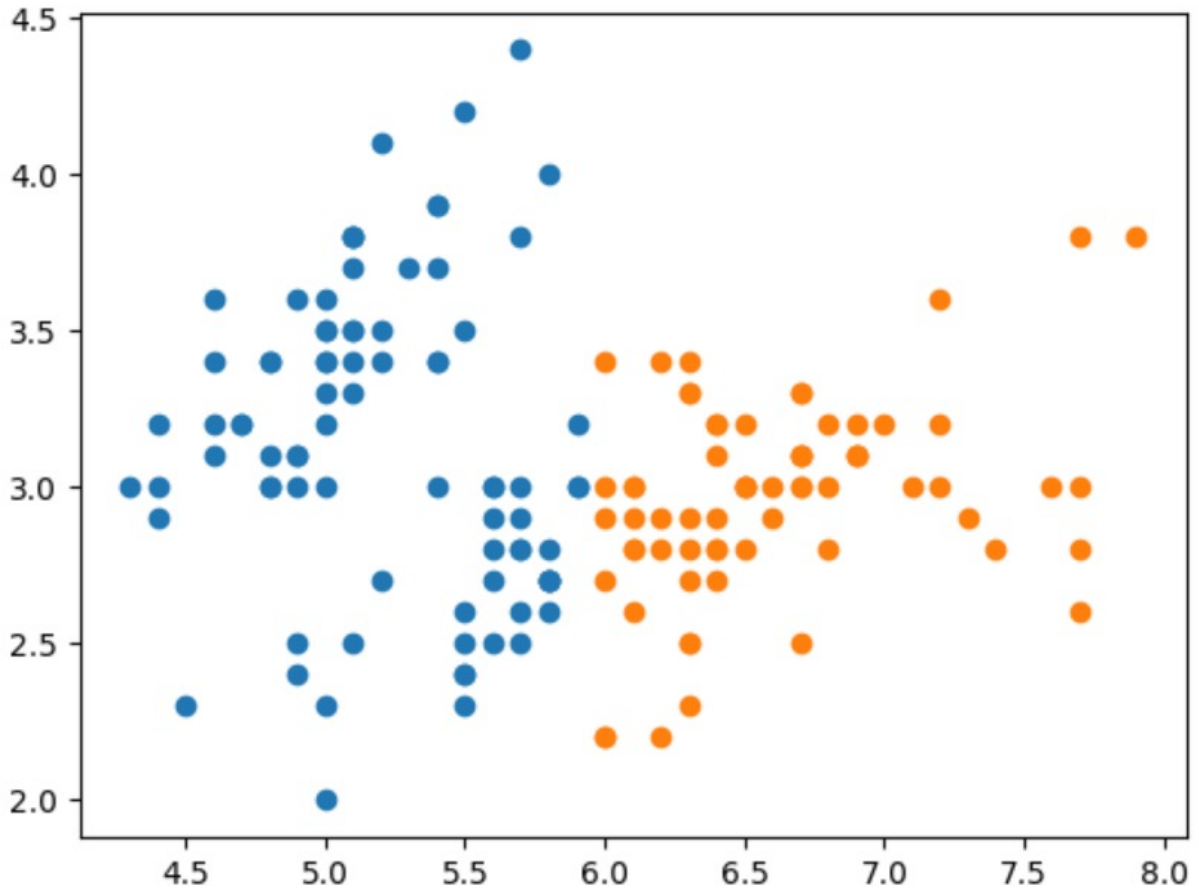
クラスタ数を変える

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=2,random_state=0)
model.fit(gaku)
cluster=model.predict(gaku)
print(cluster)

plt.scatter(gaku[cluster==0,0],gaku[cluster==0,1])
plt.scatter(gaku[cluster==1,0],gaku[cluster==1,1])
plt.show()
```

クラスタ数2の結果

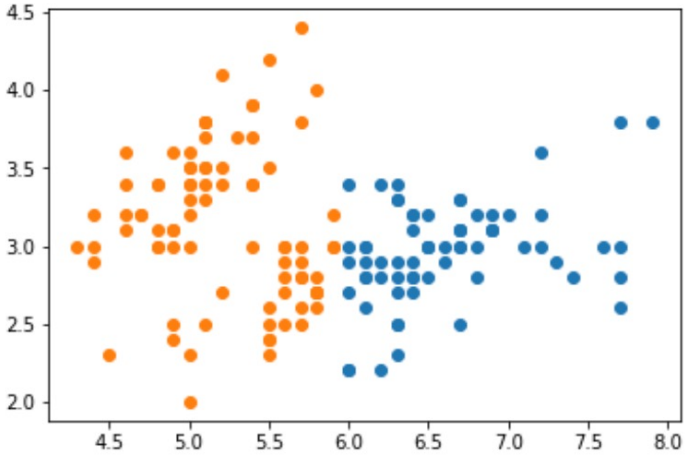
```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 0 1 1 0 1 0 0 1 0 0 1 1 1
 1 1 1 1 1 0 0 0 0 1 0 1 1 1 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 1 1 1 0 1 1 1 1
 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
 1 0]
```



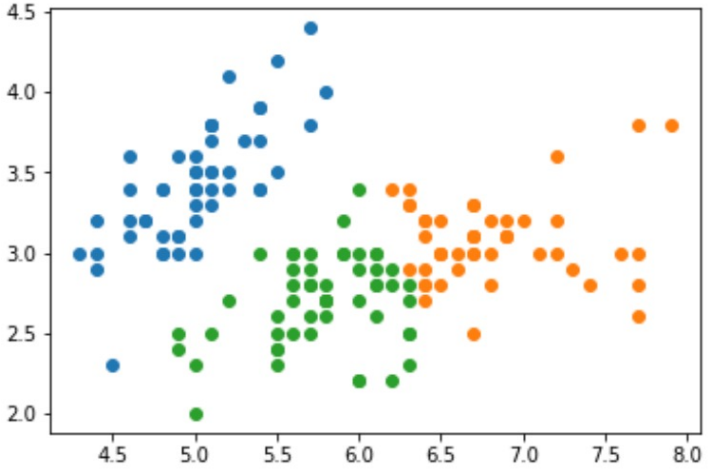
散布図の結果

クラスタ数を変える

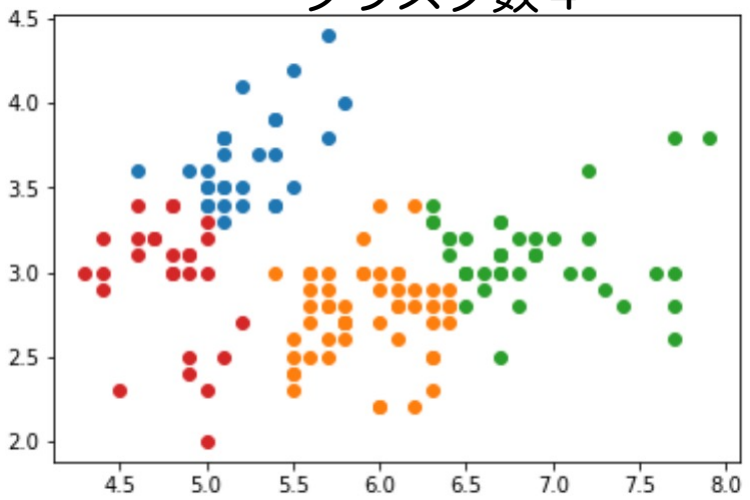
クラスタ数２



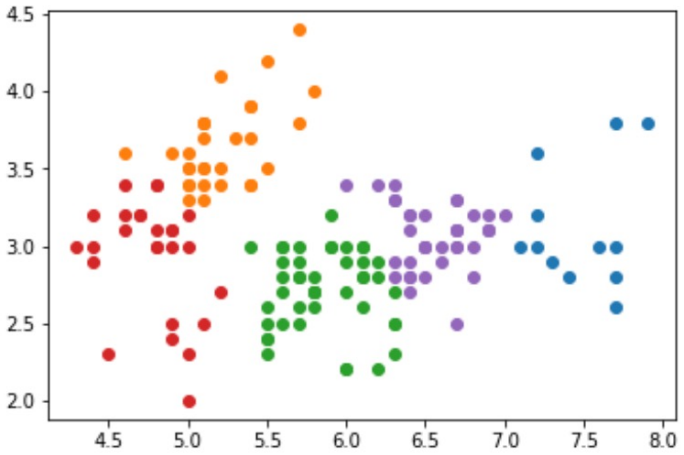
クラスタ数３



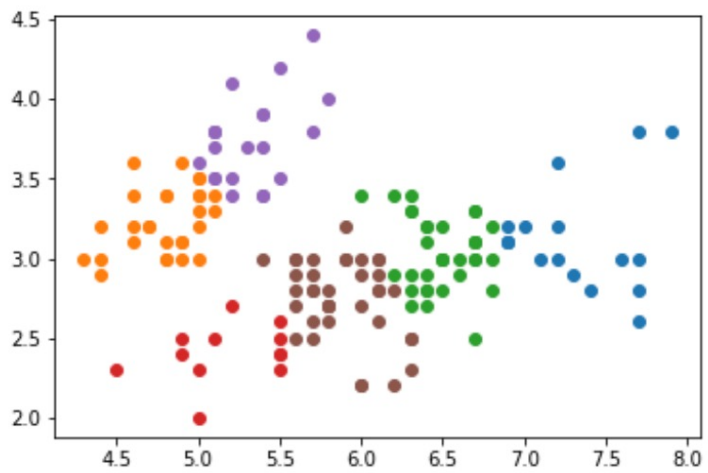
クラスタ数４



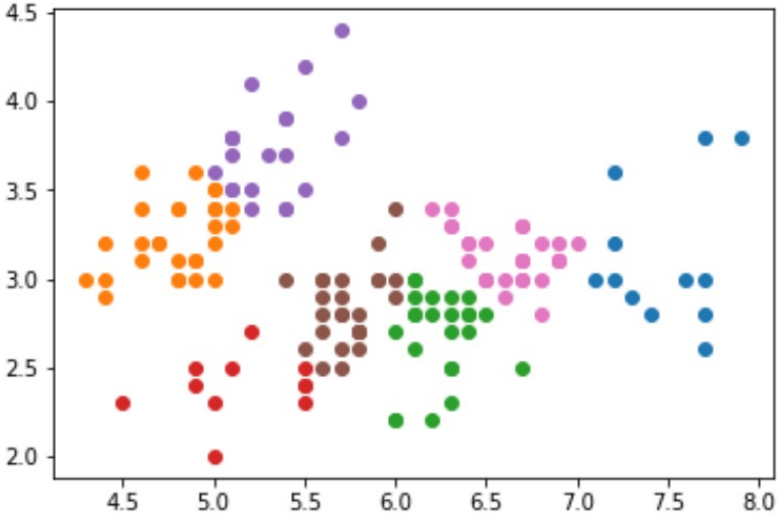
クラスタ数５



クラスタ数６



クラスタ数７



3次元以上のクラスタリング

がく片の長さと幅(の2次元の特徴量)でクラスタリングを行ったが、
3次元以上の特徴量でも同様にクラスタリングが可能

```
from sklearn.datasets import load_iris
iris = load_iris()
iris.data
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1]
```

iris.dataは特徴量4の(150,4)

```
gaku = iris.data[:,0:2]
gaku
```

```
       [5.8, 4. ],
       [5.7, 4.4],
       [5.4, 3.9],
       [5.1, 3.5],
       [5.7, 3.8],
       [5.1, 3.8],
       [5.4, 3.4],
       [5.1, 3.7],
       [4.6, 3.6],
       [5.1, 3.3],
       [4.8, 3.4],
       [5. , 3. ],
       [5. , 3.4],
       [5.2, 3.5],
```

gakuは特徴量2の(150,2)

gakuをiris.dataに変更して
クラスタリング

3次元以上のクラスタリング

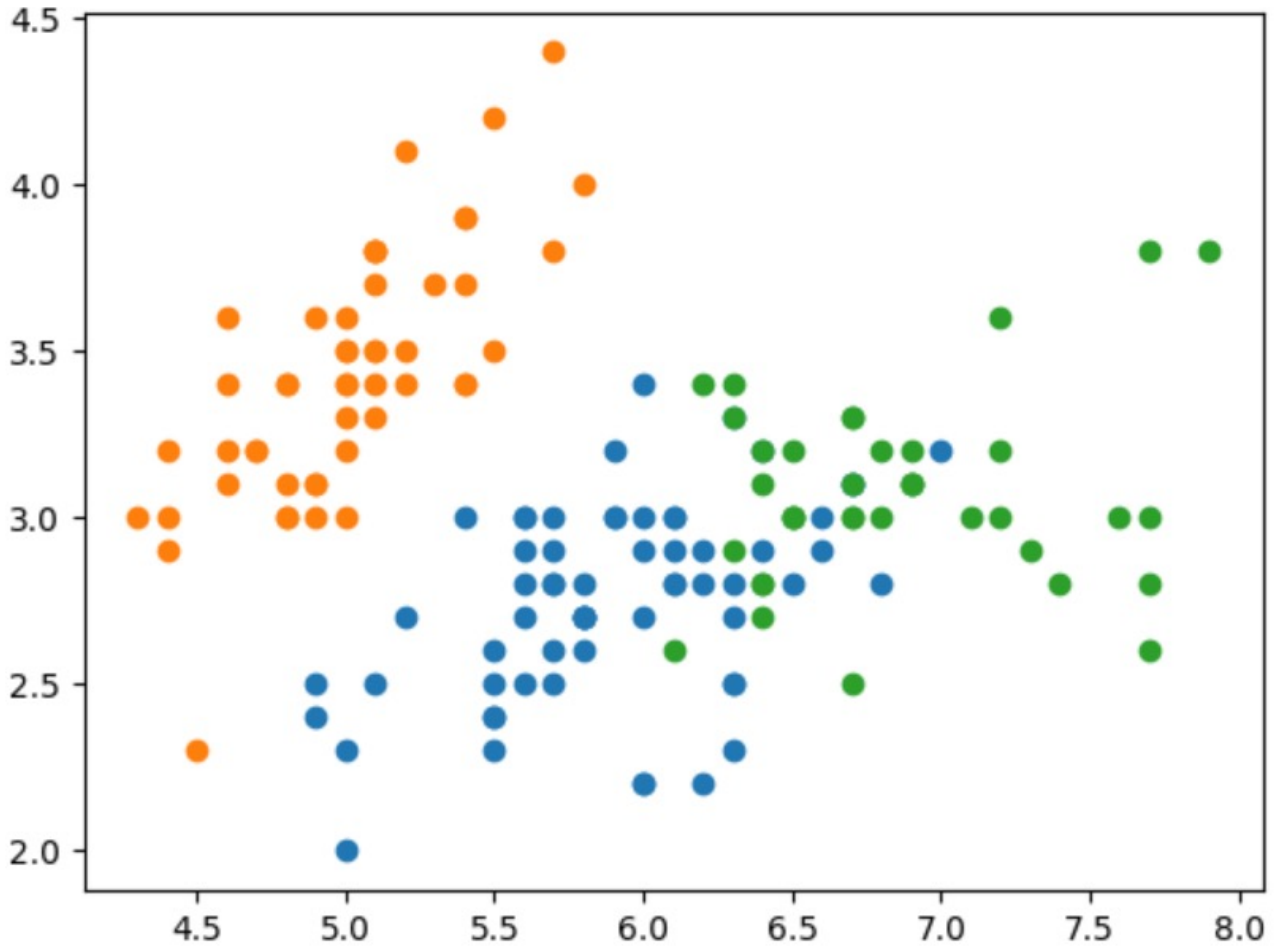
がく片の長さと幅(の2次元の特徴量)でクラスタリングを行ったが、
3次元以上の特徴量でも同様にクラスタリングが可能

```
model = KMeans(n_clusters=3,random_state=0)
model.fit(iris.data)
cluster=model.predict(iris.data)
print(cluster)

plt.scatter(iris.data[cluster==0,0],iris.data[cluster==0,1])
plt.scatter(iris.data[cluster==1,0],iris.data[cluster==1,1])
plt.scatter(iris.data[cluster==2,0],iris.data[cluster==2,1])
plt.show()
```

クラスタ数3の結果

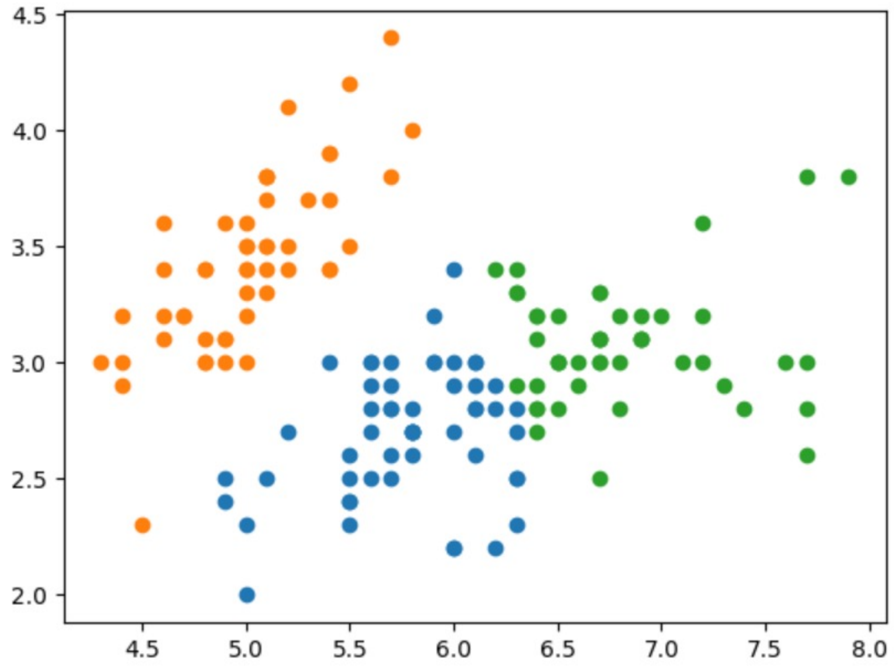
```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
```



3次元以上のクラスタリング

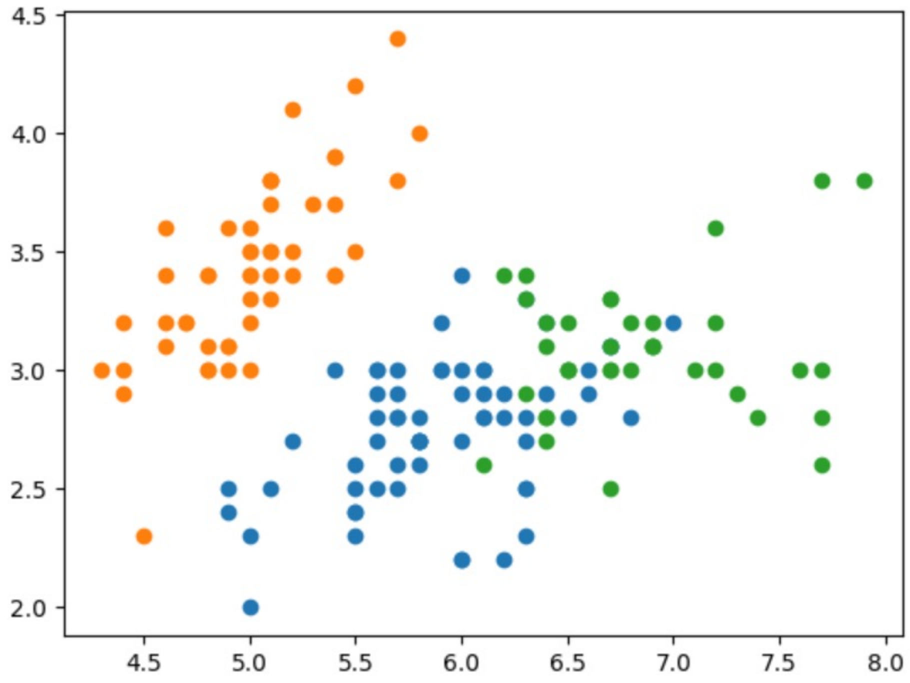
同じクラスタ数(0,1,2)だが、学習するデータ量が違う(2次元と4次元)。
4次元のデータによるクラスタリングを2次元で可視化しているのでやや混ざっている部位もあり。

```
[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 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 0 2 0 2 0 2 0 0 0 0 0 0 2 0 0 0 0 0 0 0
 2 2 2 2 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 0 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
```



がく片の長さと幅でクラスタリングし、がく片の長さと幅で図示

```
[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 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
```

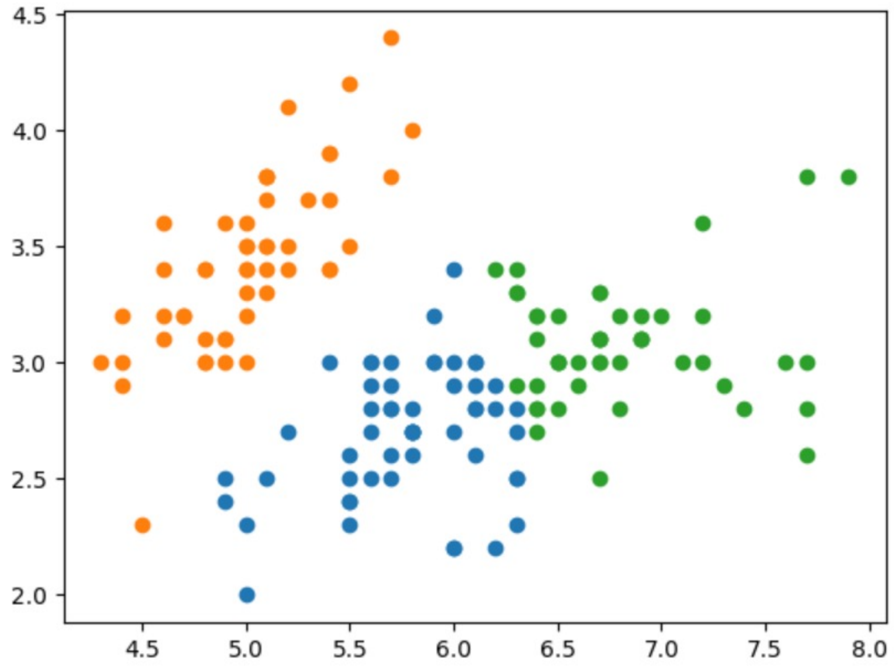


がく片の長さと幅、花びらの長さと幅でクラスタリングし、がく片の長さと幅で図示

3次元以上のクラスタリング

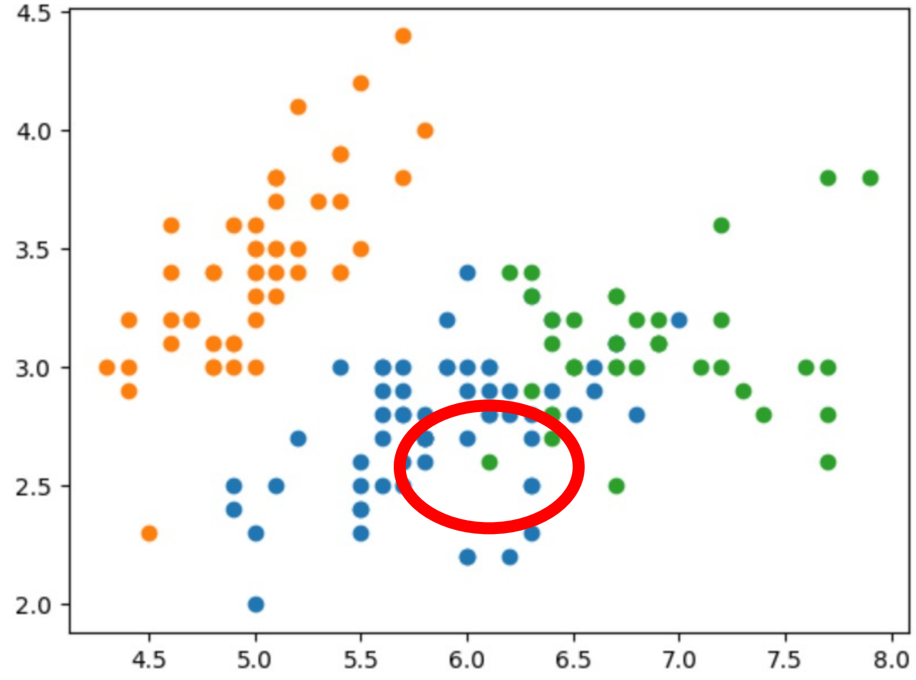
同じクラスタ数(0,1,2)だが、学習するデータ量が違う(2次元と4次元)。
4次元のデータによるクラスタリングを2次元で可視化しているのでやや混ざっている部位もあり。

```
[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 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 0 2 0 2 0 2 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0
 2 2 2 2 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 0 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
```



がく片の長さと幅でクラスタリングし、がく片の長さと幅で図示

```
[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 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
```



がく片の長さと幅、花びらの長さと幅でクラスタリングし、がく片の長さと幅で図示

3次元以上のクラスタリング結果を2次元で(きれいに)可視化したい

データに基づいてグループ分けしているので、結果はグループに分かれてほしい
→4次元は可視化出来ない(のできれいに分かれていない)　→　次元削減

```
model = KMeans(n_clusters=3,random_state=0)
model.fit(iris.data)
cluster=model.predict(iris.data)
print(cluster)

plt.scatter(iris.data[cluster==0,0],iris.data[cluster==0,1])
plt.scatter(iris.data[cluster==1,0],iris.data[cluster==1,1])
plt.scatter(iris.data[cluster==2,0],iris.data[cluster==2,1])
plt.show()
```

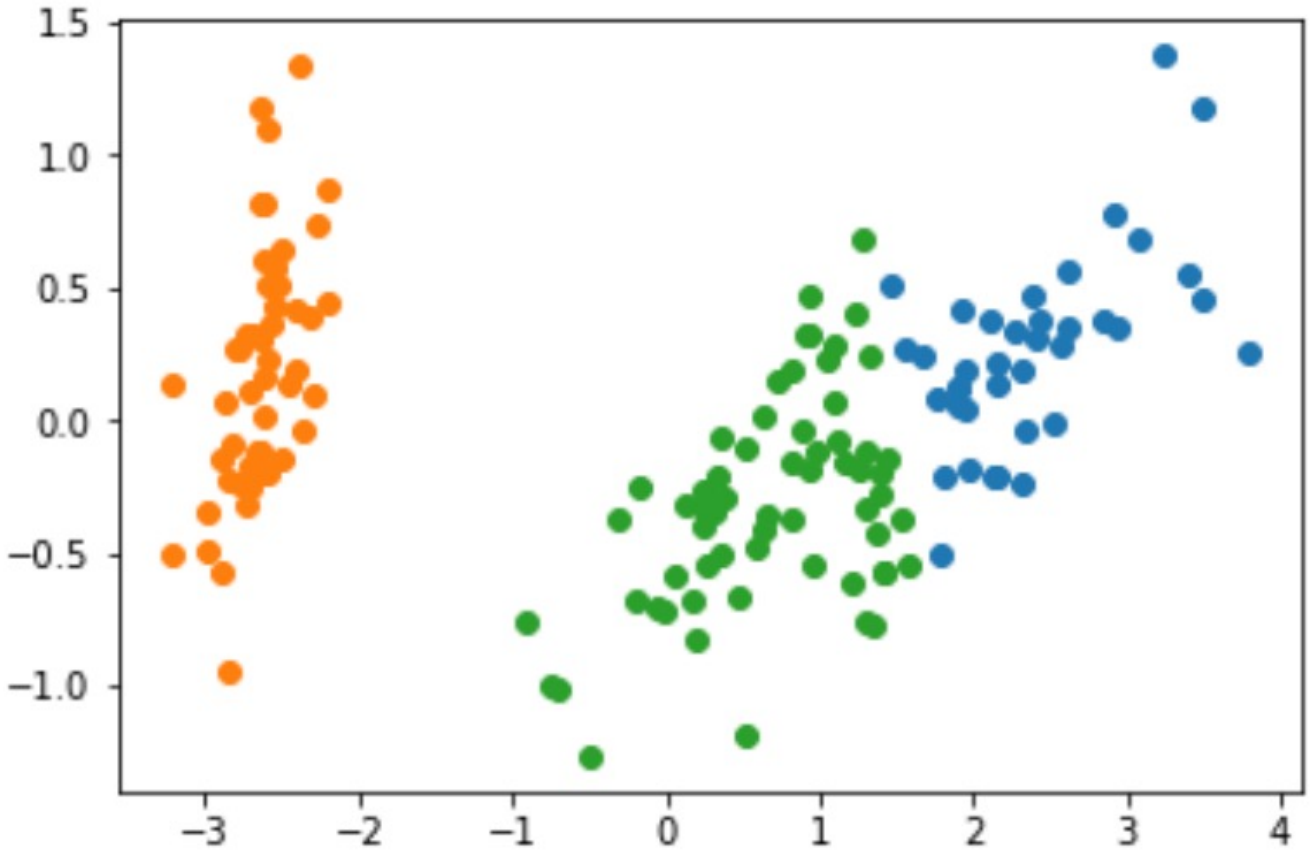
```
model = KMeans(n_clusters=3,random_state=0)
model.fit(iris.data)
cluster=model.predict(iris.data)
print(cluster)

from sklearn.decomposition import PCA
model2 = PCA(n_components=2)
result = model2.fit_transform(iris.data)

plt.scatter(result[cluster==0,0],result[cluster==0,1])
plt.scatter(result[cluster==1,0],result[cluster==1,1])
plt.scatter(result[cluster==2,0],result[cluster==2,1])
plt.show()
```

150個のデータを4つの説明変数で3つのグループに分ける（クラスタリング）
4つの説明変数を２つの新たな説明変数にしてＸ軸、Ｙ軸に設定（次元削減）

3次元以上のクラスタリング結果を2次元で(きれいに)可視化したい



150個の正解を与えていないデータをクラスタリングによって3つグループに分けた
（次元削減は2次元で可視化するために使用）

教師無し機械学習のまとめ

次元削減
目的：多次元のデータを2~3次元に減らす
結果：2次元や3次元の特徴量
使用例)
→データの前処理
解析しやすくする

可視化することが出来る
→データの全体像の把握
→きれいに分かれるかどうかはやってみないとわからない
分かれればその集団は他と違う特徴をもっていることが分かる

クラスタリング
目的：多次元のデータを決めた数でグルーピングする
結果：クラスタ1,2,3などのクラスタ番号
使用例)
→正解が分からなくてもグループに分けられる
似ているデータ群を抽出出来る
・アンケート結果から3つのグループに分ける
グループから特徴を抽出できる
・大量の記事をクラスタリングしてキーワードを抽出する

課題

・WebClassにある課題15をやりましょう

締め切りは1週間後の7/25の23:59です。
締め切りを過ぎた課題は受け取らないので注意して下さい。
(1週間後に正解をアップします)