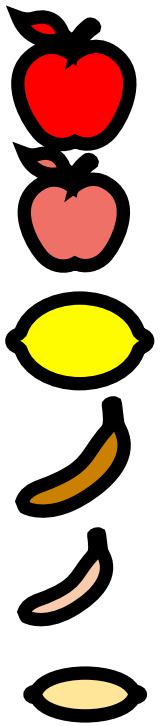
教師なし機械学習②

本スライドは、自由にお使いください。
使用した場合は、このQRコードからアンケート
に回答をお願いします。

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

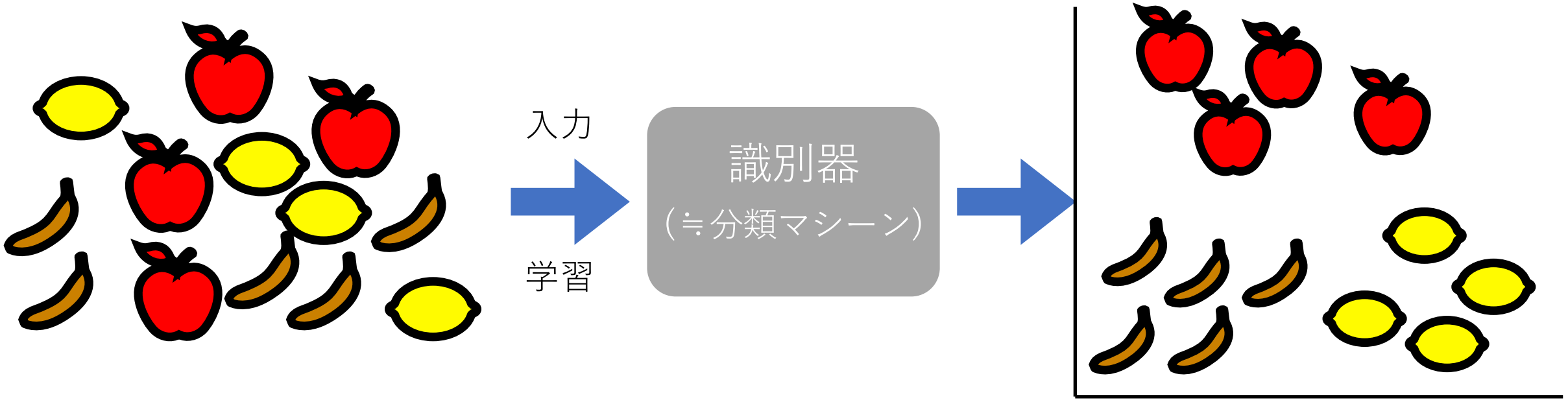
学習

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

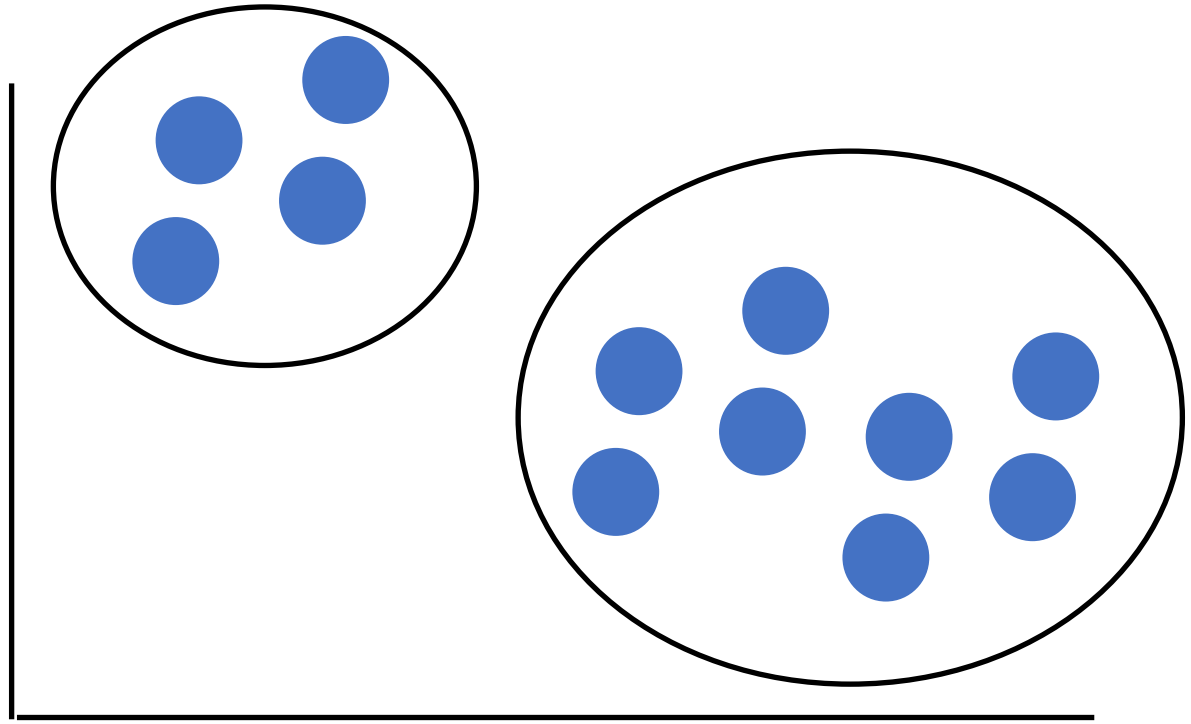
今回の教師なし機械学習（クラスタリング）



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

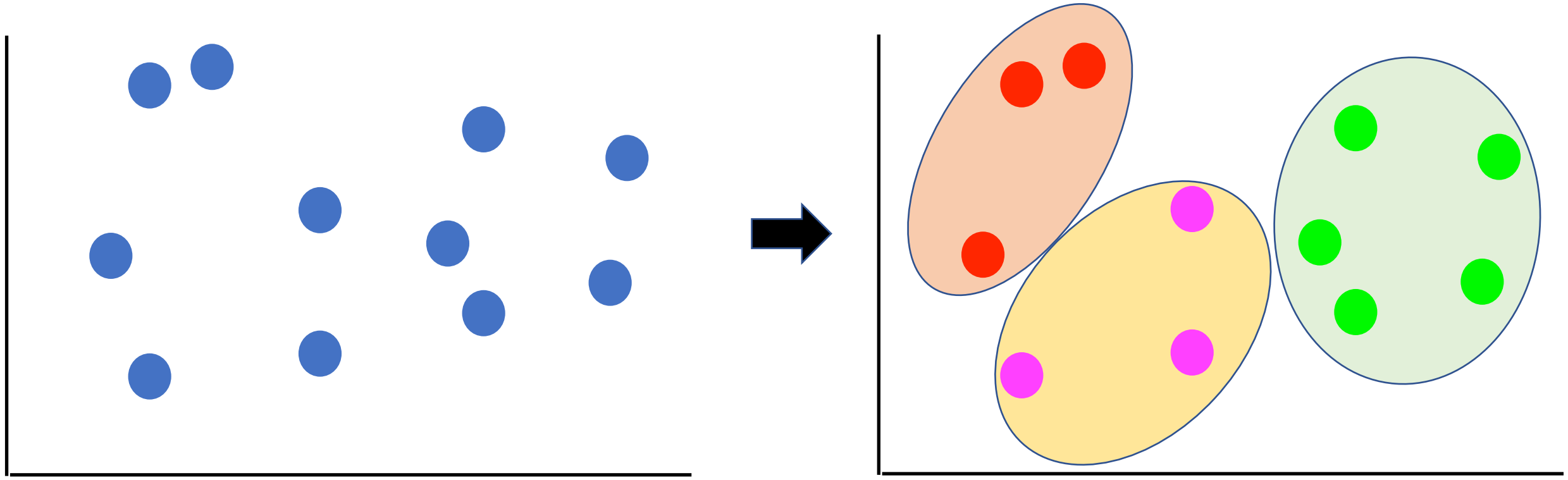


どのような法則でクラスタリングを行うか

k-means 法

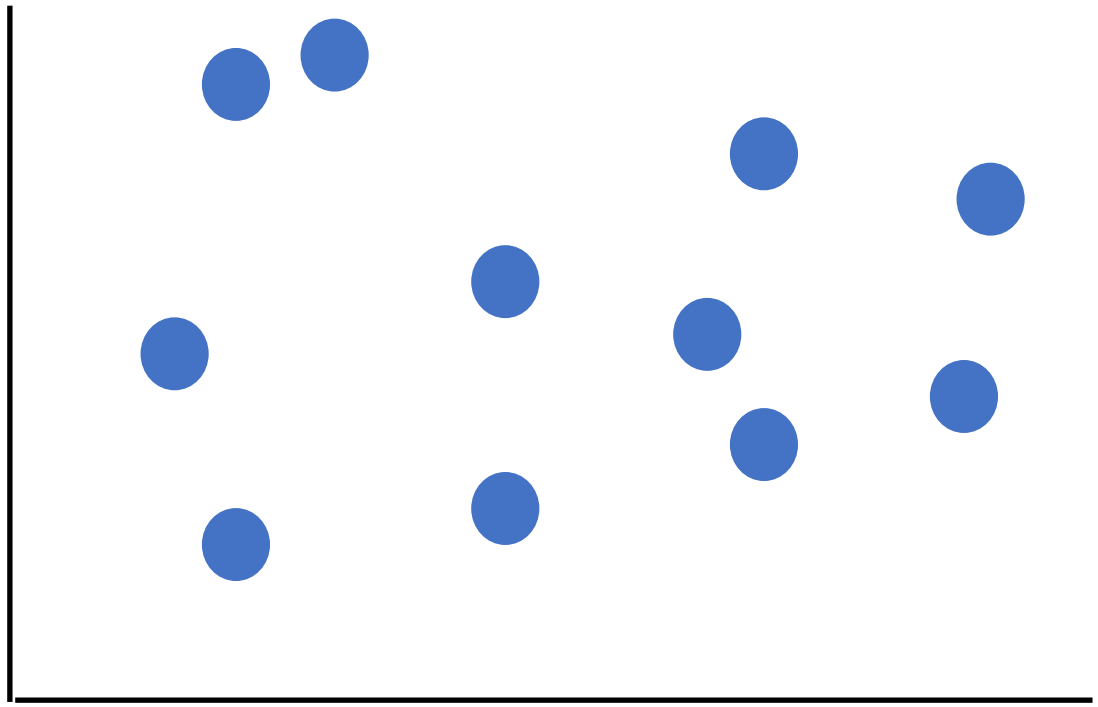
クラスタリングでも最も基本的なアルゴリズム

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

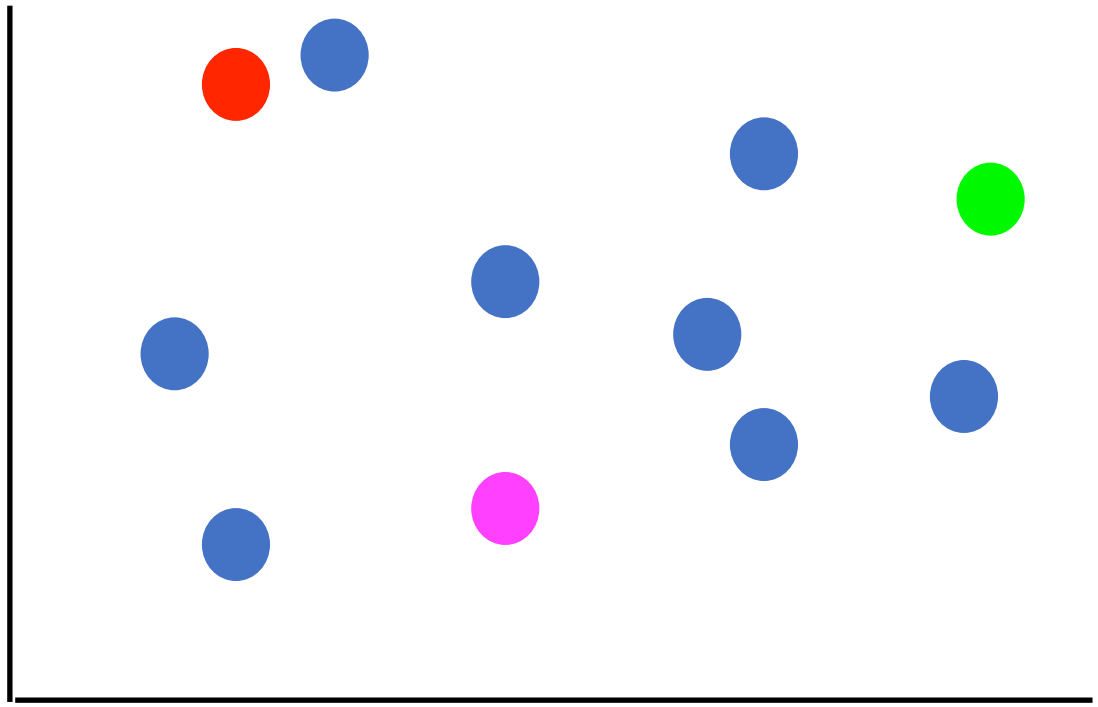
①分けるクラスタ数を決定する(今回は 3 とする)



k-means 法

①分けるクラスタ数を決定する(今回は３とする)

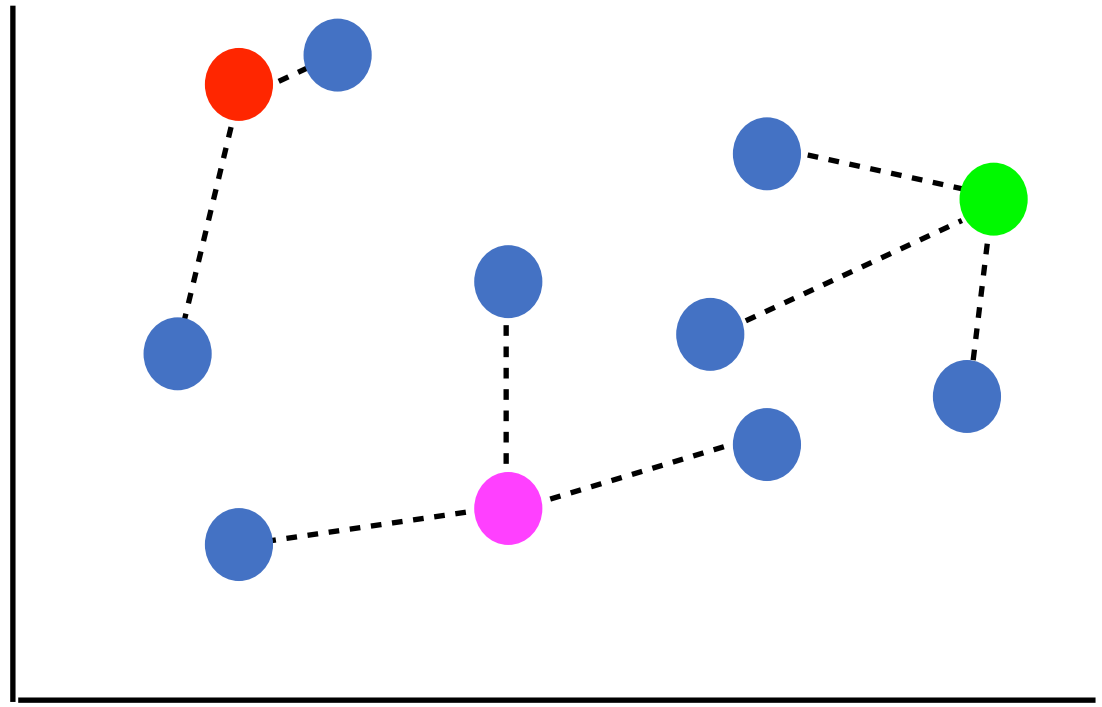
②ランダムにクラスタの個数分データを選ぶ



この３点を代表点とする

k-means 法

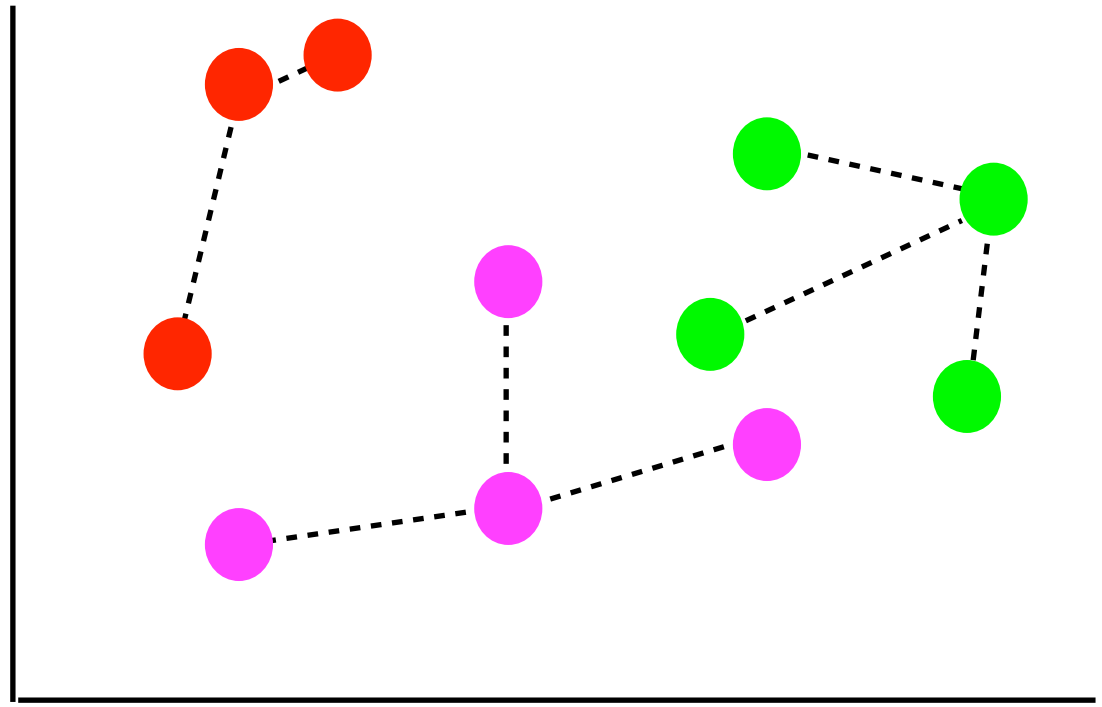
③データを各代表点の距離をもとに各クラスタに所属させる



各データは一番近い距離の代表点の所属となる

k-means 法

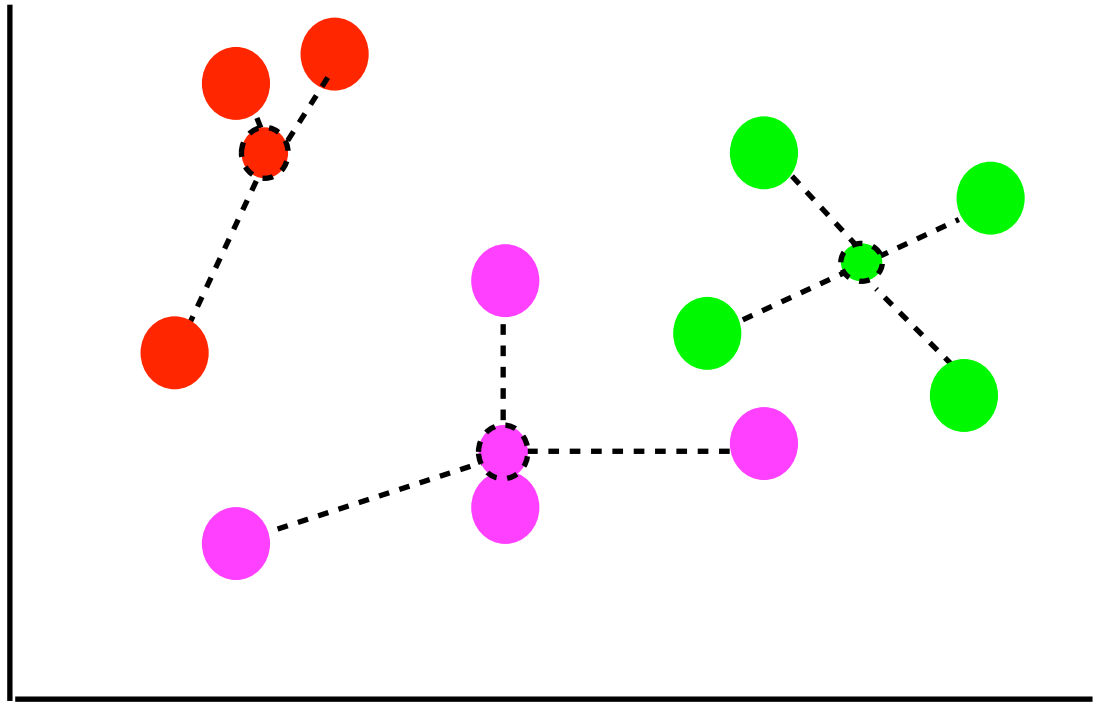
③データを各代表点の距離をもとに各クラスタに所属させる



各データは一番近い距離の代表点の所属となる

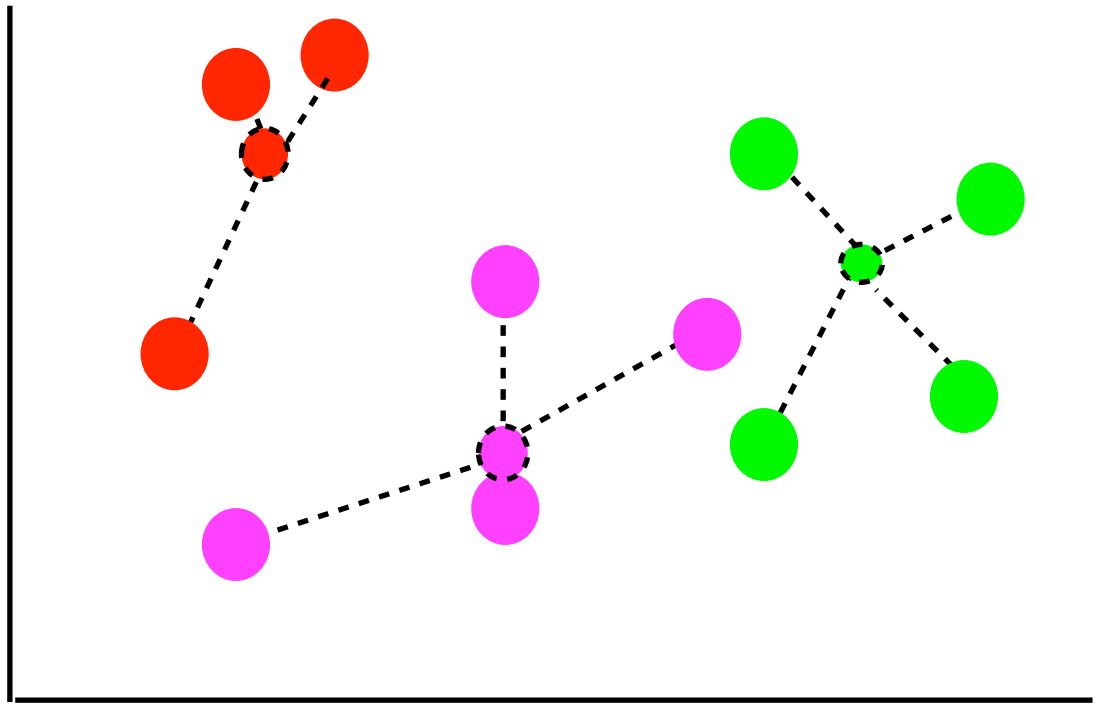
k-means 法

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

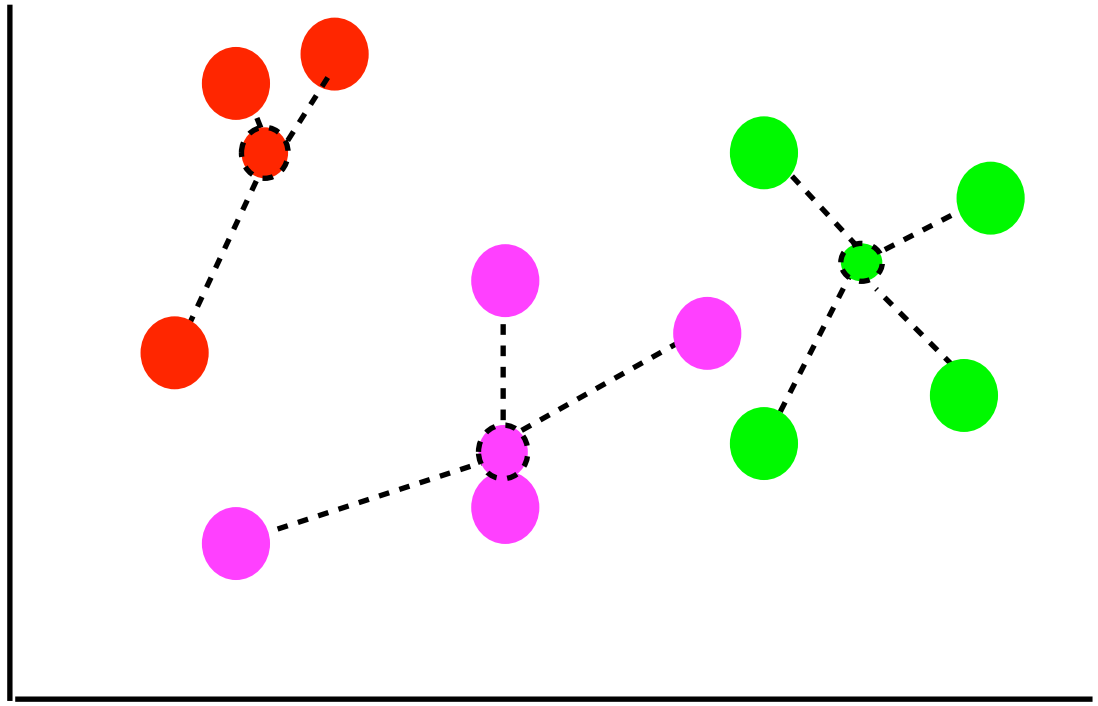


距離が変わるので所属するクラスタが変わります

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()
data = 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dataに代入

アヤメのデータを読み込む

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

data - NumPy object array

	がく片の長さ	がく片の幅	花びらの長さ	花びらの幅
	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5	3.4	1.5	0.2
8	4.4	2.9	1.4	0.2
9	4.9	3.1	1.5	0.1
10	5.4	3.7	1.5	0.2
11	4.8	3.4	1.6	0.2
12	4.8	3	1.4	0.1

Format Resize Background color

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 = data[:,0:2]
```

x軸にがく片の長さ、y軸にがく片の幅として図示

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

data - NumPy object array

	がく片の長さ	がく片の幅	花びらの長さ	花びらの幅
	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5	3.4	1.5	0.2
8	4.4	2.9	1.4	0.2
9	4.9	3.1	1.5	0.1
10	5.4	3.7	1.5	0.2
11	4.8	3.4	1.6	0.2
12	4.8	3	1.4	0.1

Format Resize Background color

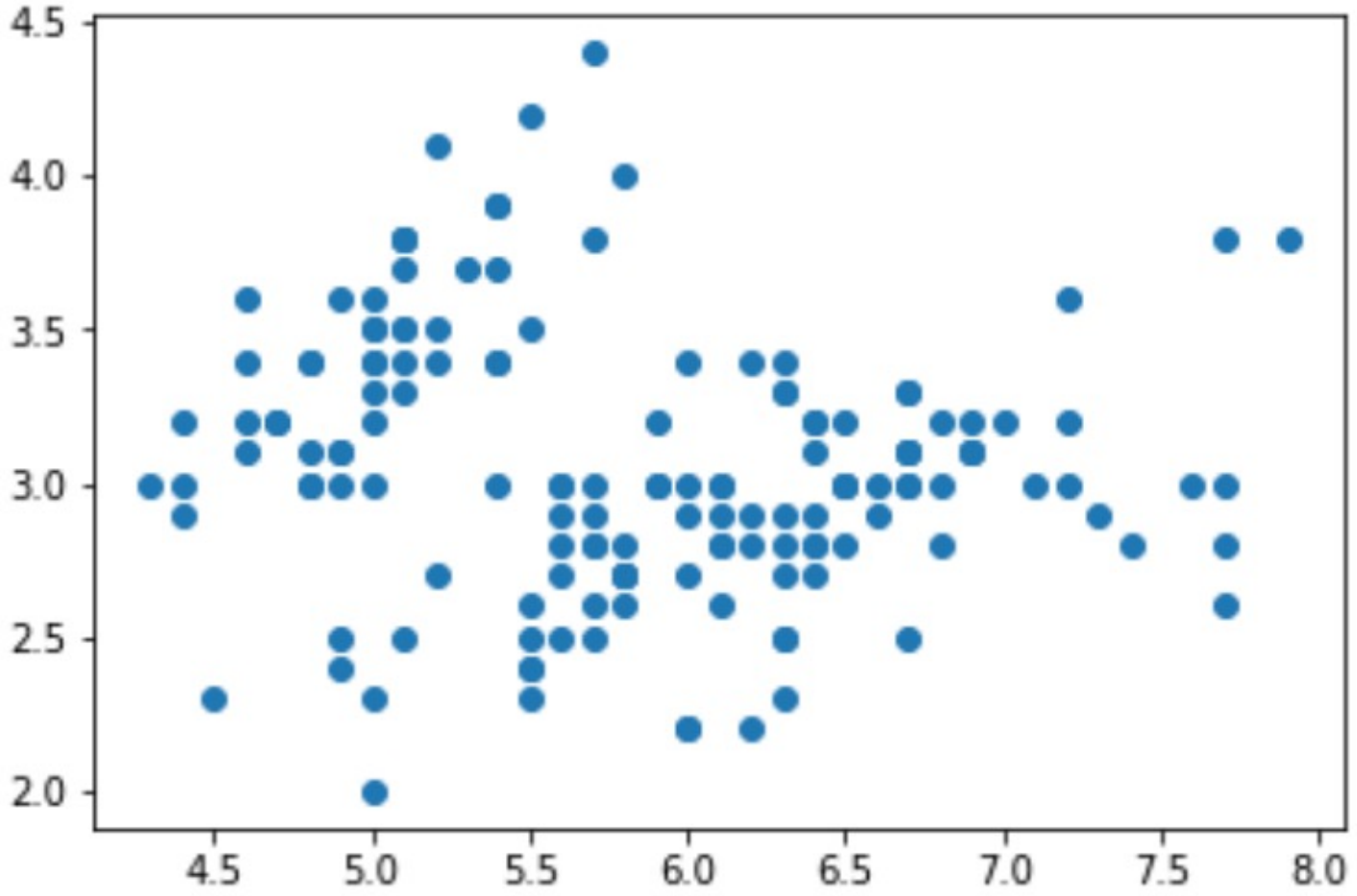
がく片の長さと幅のデータを図示する

がく片の長さと幅だけを取り出す

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

x軸にがく片の長さ、y軸にがく片の幅として図示

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



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.figure()
plt.scatter(gaku[:,0],gaku[:,1])
plt.show()
```

```
plt.figure()
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()
```

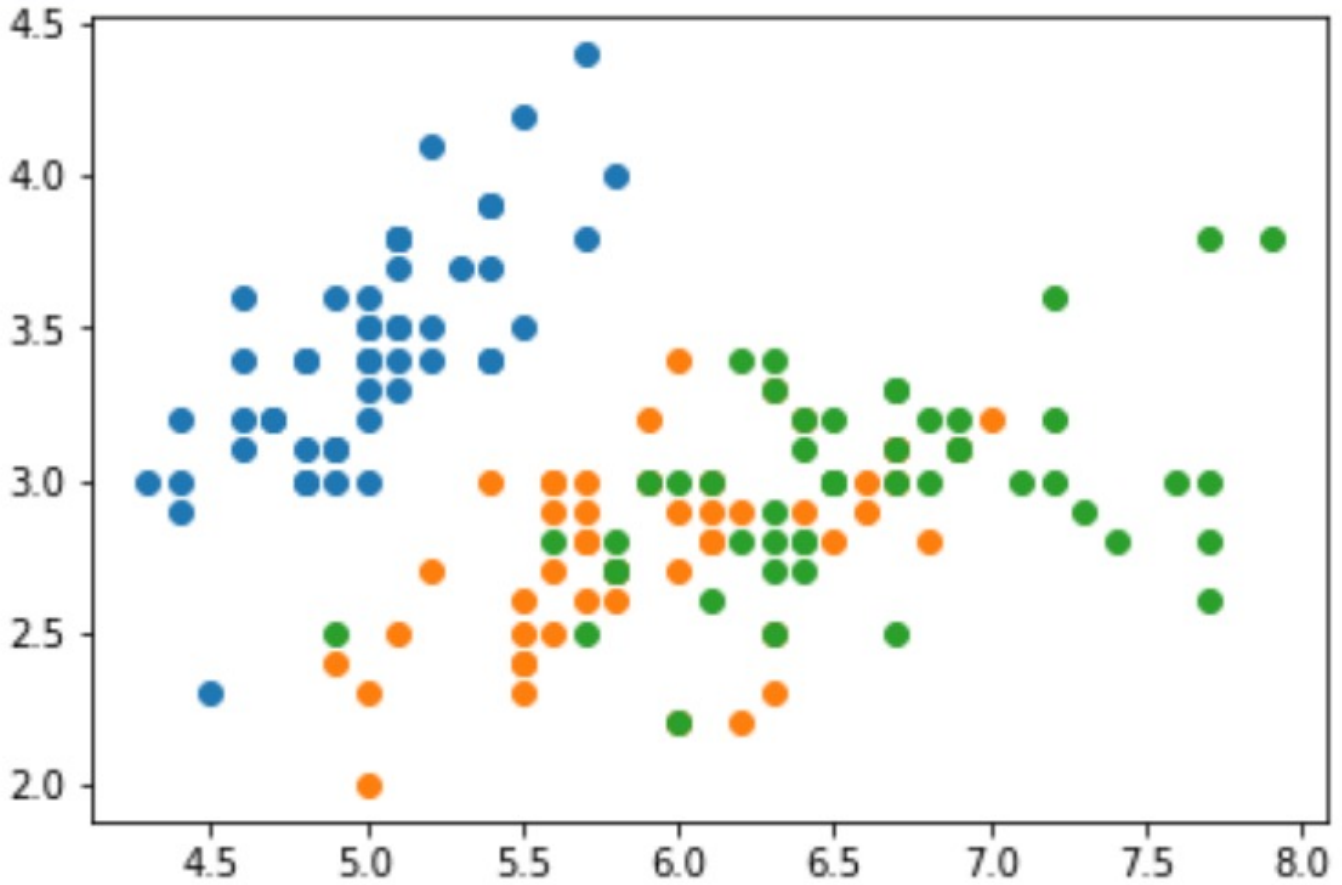
色を指定しないと勝手に色が
割り当てられます

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.figure()
plt.scatter(gaku[:,0],gaku[:,1])
plt.show()
```

```
plt.figure()
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()
```



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

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

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

k-means法を実行する

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

```
In [52]: print(cluster)
[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 2 1 2 1 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2
 1 1 1 1 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1 1 2 1 1 1 1
 1 1 2 2 1 1 1 1 2 1 2 1 2 1 1 2 2 1 1 1 1 1 2 2 1 1 1 2 1 1 1 2 1 1 1 2 1
 1 2]
```

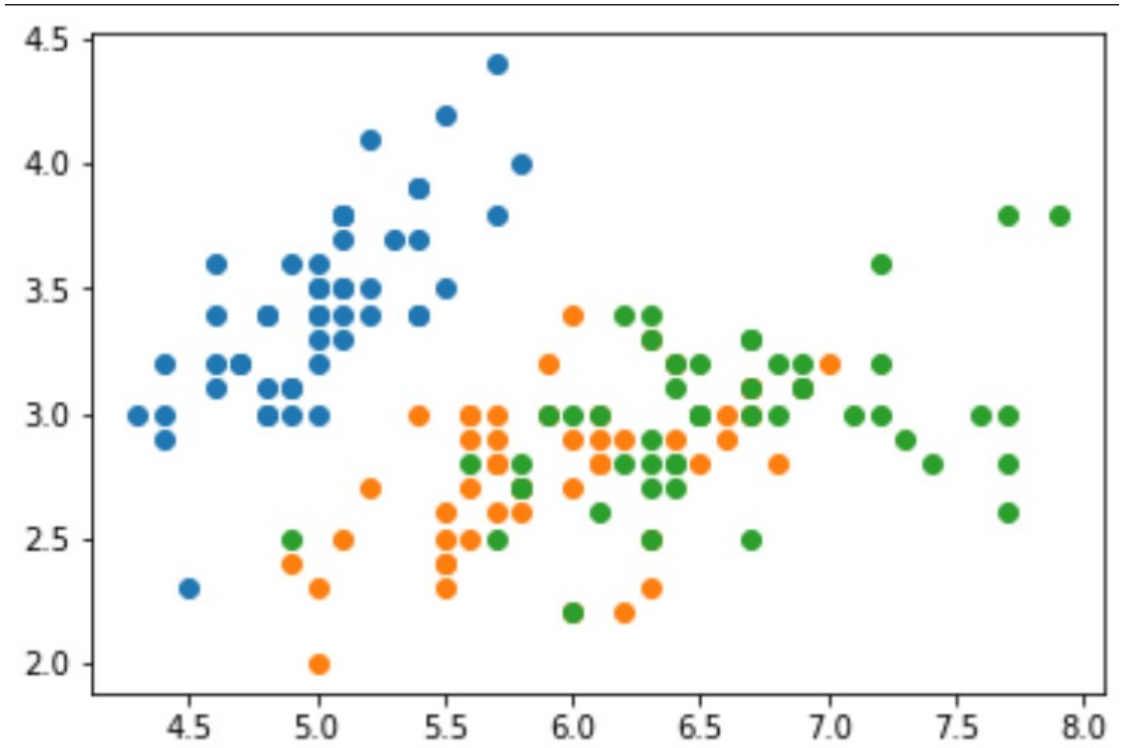
モデル名=Kmeans()で実行
n_clusters=~~でクラスタリングしたい数を入力する
random_state=0は皆同じ結果になるためのランダムの指定
クラスタリング結果はmodel.predict()で表示される

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

```
plt.figure()
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

前回と別のやり方で

```
print(gaku)
```

がく片の長さと幅でクラスタリングを図示

前回と別のやり方で

```
print(gaku)
```

```
In [53]: print(gaku)
[[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. ]
```

がく片の長さと幅でクラスタリングを図示

前回と別のやり方で

```
print(gaku)
```

先頭4行を抜き出す

```
test = gaku[0:4]
print(test)
```

```
In [53]: print(gaku)
[[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. ]
```

```
In [54]: test = gaku[0:4]

In [55]: print(test)
[[5.1 3.5]
 [4.9 3. ]
 [4.7 3.2]
 [4.6 3.1]]
```

がく片の長さと幅でクラスタリングを図示

前回と別のやり方で

```
print(gaku)
```

先頭4行を抜き出す

```
test = gaku[0:4]
print(test)
```

1列目だけ抜き出す

```
print(test[ : ,0])
```

```
In [53]: print(gaku)
[[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. ]
```

```
In [54]: test = gaku[0:4]

In [55]: print(test)
[[5.1 3.5]
 [4.9 3. ]
 [4.7 3.2]
 [4.6 3.1]]
```

```
In [84]: print(test[:,0])
[5.1 4.9 4.7 4.6]
```

がく片の長さと幅でクラスタリングを図示

前回と別のやり方で

```
print(test[ : ,0])
```

```
In [84]: print(test[:,0])
[5.1 4.9 4.7 4.6]
```

がく片の長さと幅でクラスタリングを図示

前回と別のやり方で

```
print(test[ : ,0])
```

```
print(test[ : ,0] == 4.7)
print(test[ : ,0] < 5)
```

```
In [84]: print(test[:,0])
[5.1 4.9 4.7 4.6]
```

がく片の長さと幅でクラスタリングを図示

前回と別のやり方で

```
print(test[ : ,0])
```

```
print(test[ : ,0] == 4.7)
print(test[ : ,0] < 5)
```

```
In [84]: print(test[:,0])
[5.1 4.9 4.7 4.6]
```

```
In [85]: print(test[:,0] == 4.7)
[False False  True False]
In [86]: print(test[:,0] < 5)
[False  True  True  True]
```

配列に(==,<,>)などの代入演算子を使うとTrueとFalseに置き換わる

がく片の長さと幅でクラスタリングを図示

前回と別のやり方で

```
print(test[ : ,0])
```

```
print(test[ : ,0] == 4.7)
print(test[ : ,0] < 5)
```

```
print(test[[0, 2],0])
print(test[[0,1,2,3],0])
```

```
In [84]: print(test[:,0])
[5.1 4.9 4.7 4.6]
```

```
In [85]: print(test[:,0] == 4.7)
[False False  True False]
In [86]: print(test[:,0] < 5)
[False  True  True  True]
```

がく片の長さと幅でクラスタリングを図示

前回と別のやり方で

```
print(test[ : ,0])
```

```
print(test[ : ,0] == 4.7)
print(test[ : ,0] < 5)
```

```
print(test[[0, 2],0])
print(test[[0,1,2,3],0])
```

```
In [84]: print(test[:,0])
[5.1 4.9 4.7 4.6]
```

```
In [85]: print(test[:,0] == 4.7)
[False False  True False]

In [86]: print(test[:,0] < 5)
[False  True  True  True]
```

```
In [64]: print(test[[0,2],0])
[5.1 4.7]

In [65]: print(test[[0,1,2,3],0])
[5.1 4.9 4.7 4.6]
```

範囲を”:”で指定せずに[数字,数字,,,]でその行(or列)番号のみ抜き出す

がく片の長さと幅でクラスタリングを図示

前回と別のやり方で

print(test[: ,0])

```
In [84]: print(test[:,0])
[5.1 4.9 4.7 4.6]
```

print(test[: ,0] == 4.7)

print(test[: ,0] < 5)

```
In [85]: print(test[:,0] == 4.7)
[False False  True False]

In [86]: print(test[:,0] < 5)
[False  True  True  True]
```

print(test[[0, 2],0])
print(test[[0,1,2,3],0])

```
In [64]: print(test[[0,2],0])
[5.1 4.7]

In [65]: print(test[[0,1,2,3],0])
[5.1 4.9 4.7 4.6]
```

print(test[[False,False,True,True],0])

がく片の長さと幅でクラスタリングを図示

前回と別のやり方で

```
print(test[ : ,0])
```

```
print(test[ : ,0] == 4.7)
print(test[ : ,0] < 5)
```

```
print(test[[0, 2],0])
print(test[[0,1,2,3],0])
```

```
print(test[[False,False,True,True],0])
```

```
In [84]: print(test[:,0])
[5.1 4.9 4.7 4.6]
```

```
In [85]: print(test[:,0] == 4.7)
[False False  True False]
In [86]: print(test[:,0] < 5)
[False  True  True  True]
```

```
In [64]: print(test[[0,2],0])
[5.1 4.7]

In [65]: print(test[[0,1,2,3],0])
[5.1 4.9 4.7 4.6]
```

```
In [66]: print(test[[False,False,True,True],0])
[4.7 4.6]
```

範囲を":"で指定せずに[True(or False),...]でTrueのみを抜き出す

がく片の長さと幅でクラスタリングを図示

前回と別のやり方で

print(test[: ,0])

```
In [84]: print(test[:,0])
[5.1 4.9 4.7 4.6]
```

print(test[: ,0] == 4.7)

print(test[: ,0] < 5)

```
In [85]: print(test[:,0] == 4.7)
[False False  True False]

In [86]: print(test[:,0] < 5)
[False  True  True  True]
```

print(test[[0, 2],0])
print(test[[0,1,2,3],0])

```
In [64]: print(test[[0,2],0])
[5.1 4.7]

In [65]: print(test[[0,1,2,3],0])
[5.1 4.9 4.7 4.6]
```

print(test[[False,False,True,True],0])

```
In [66]: print(test[[False,False,True,True],0])
[4.7 4.6]
```

print(test[test[:,0] < 5,0])

がく片の長さと幅でクラスタリングを図示

前回と別のやり方で

print(test[: ,0])

```
In [84]: print(test[:,0])
[5.1 4.9 4.7 4.6]
```

print(test[: ,0] == 4.7)

print(test[: ,0] < 5)

```
In [85]: print(test[:,0] == 4.7)
[False False  True False]

In [86]: print(test[:,0] < 5)
[False  True  True  True]
```

print(test[[0, 2],0])
print(test[[0,1,2,3],0])

```
In [64]: print(test[[0,2],0])
[5.1 4.7]

In [65]: print(test[[0,1,2,3],0])
[5.1 4.9 4.7 4.6]
```

print(test[[False,False,True,True],0])

```
In [66]: print(test[[False,False,True,True],0])
[4.7 4.6]
```

print(test[test[:,0] < 5,0])

=

print(test[[False,True,True,True],0])

```
In [73]: print(test[test[:,0] < 5,0])
[4.9 4.7 4.6]
```

全ての行の[True(or False),...]でTrueのみを抜き出す

がく片の長さと幅でクラスタリングを図示

前回と別のやり方で

```
print(cluster)
```

クラスタリングの結果

```
print(cluster == 0))
```

クラスタが0か否か(True or False)

```
In [92]: 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 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 0 0 0 2 0 0 0 0
 0 0 2 2 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0
 0 2]
```

```
In [93]: print(cluster == 0)
[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 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 False False False False False
 False False False False False False False False False False False False
 False False False False  True False  True  True  True  True False  True
  True  True  True  True  True False False  True  True  True  True False
  True False  True False  True  True False False  True  True  True  True
  True False  True  True  True  True False  True  True  True False  True
  True  True False  True  True False]
```

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

前回と別のやり方で

```
print(cluster)
```

クラスタリングの結果

```
print(cluster == 0))
```

クラスタが0か否か(True or False)

```
print(gaku[:,0])
```

gakuの全ての行の1列目(がく片の長さ)

```
print(gaku[cluster == 0,0])
```

cluster == 0の時にTrue
のデータを取り出す

これでクラスタが0の
がく片の長さのみ取り出せた

```
In [93]: print(cluster == 0)
[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 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 False False False False False
 False False False False False False False False False False False False
 False False False False  True False  True  True  True  True False  True
  True  True  True  True  True False False  True  True  True  True False
  True False  True False  True  True False False  True  True  True  True
  True False  True  True  True  True False  True  True  True False  True
  True  True False  True  True False]
```

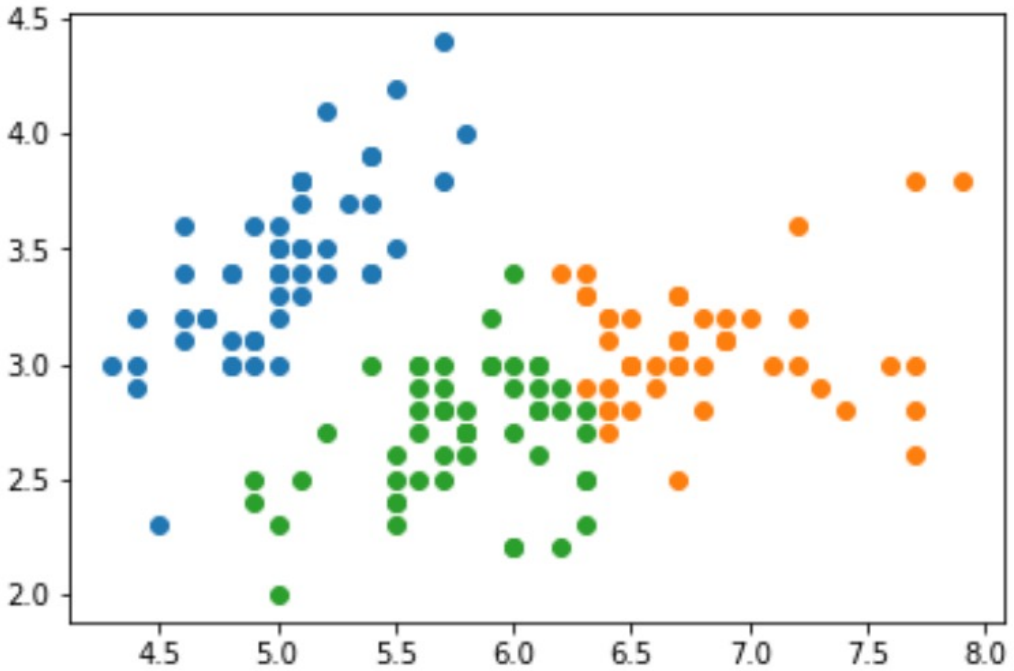
```
In [159]: print(gaku[:,0])
[5.1 4.9 4.7 4.6 5.  5.4 4.6 5.  4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.  5.  5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.
 5.5 4.9 4.4 5.1 5.  4.5 4.4 5.  5.1 4.8 5.1 4.6 5.3 5.  7.  6.4 6.9 5.5
 6.5 5.7 6.3 4.9 6.6 5.2 5.  5.9 6.  6.1 5.6 6.7 5.6 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]
```

```
In [160]: print(gaku[cluster == 0,0])
[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. ]
```

がく片の長さと幅でクラスタリングを図示

```
plt.figure(figsize=(6,4))
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)



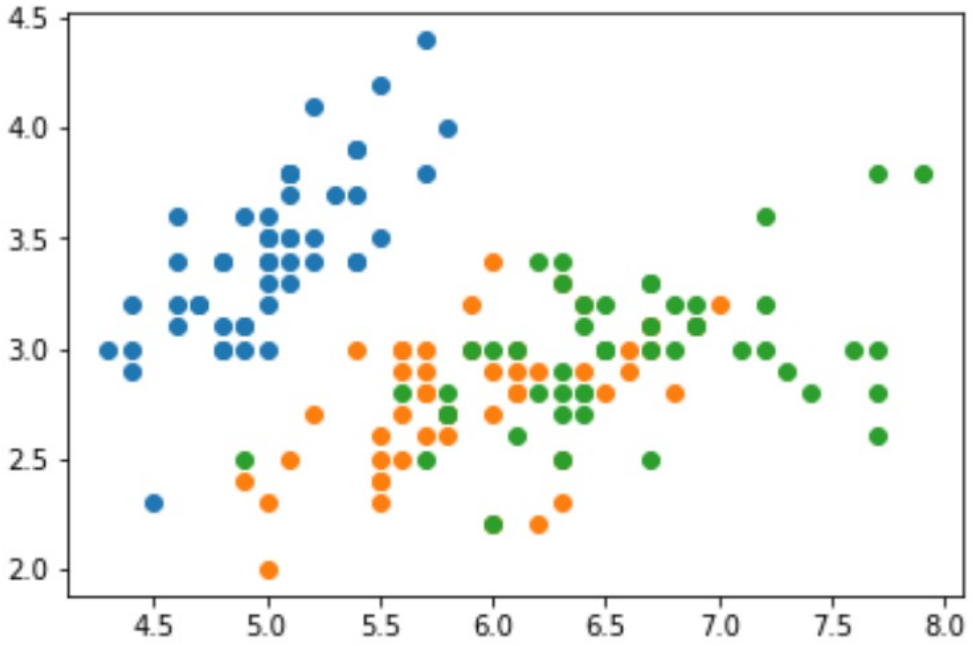
クラスタリングは距離に基づいてデータを決めた個数のグループに分ける

がく片の長さと幅でクラスタリングを図示

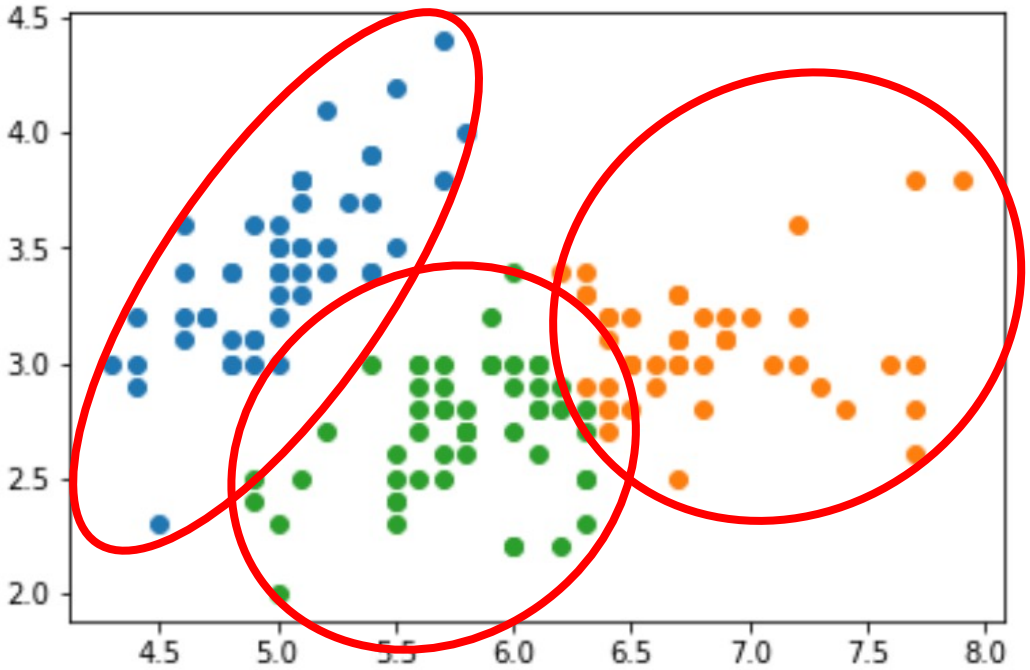
```
plt.figure(figsize=(6,4))
plt.scatter(gaku[cluster == 0,0],gaku[cluster == 0,1])
plt.scatter(gaku[cluster == 1,0],gaku[cluster == 1,1])
plt.scatter(gaku[cluster == 2,0],gaku[cluster == 2,1])
plt.show()
```

←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)

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

クラスタ数を変える

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

plt.figure()
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()
```

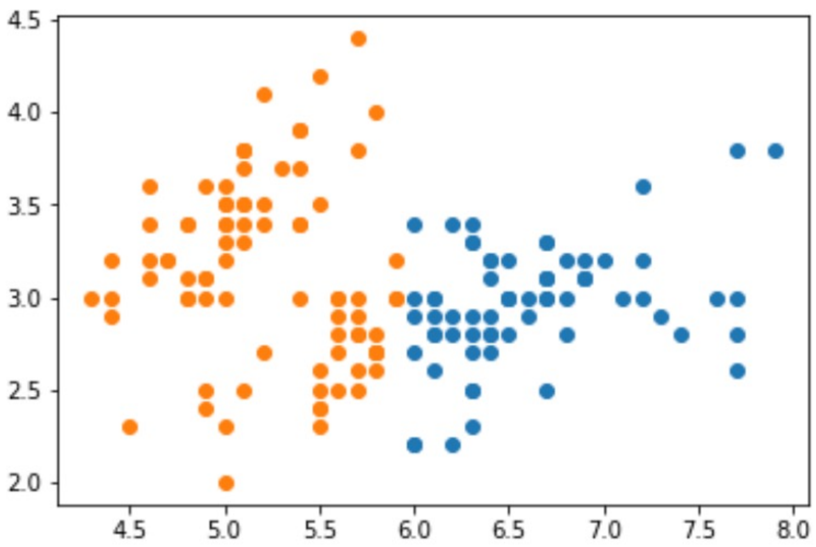
```
model = KMeans(n_clusters=2,random_state=0)
model.fit(gaku)
cluster = model.predict(gaku)
print(cluster)

plt.figure()
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()
```

クラスタ数を変える

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

plt.figure()
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()
```

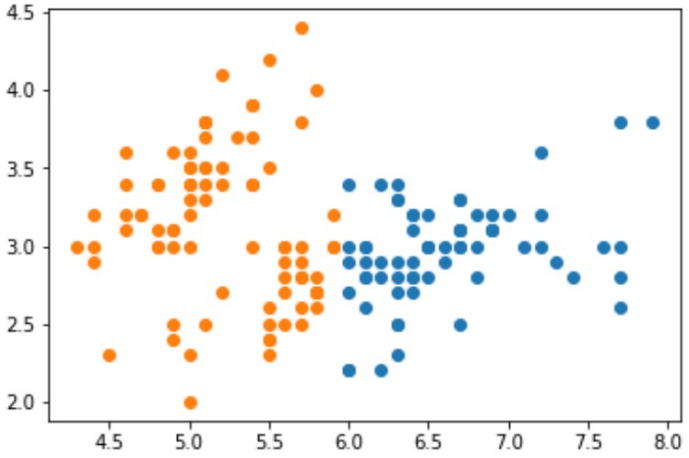
```
model = KMeans(n_clusters=2,random_state=0)
model.fit(gaku)
cluster = model.predict(gaku)
print(cluster)

plt.figure()
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()
```

```
In [77]: 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 1 0 0 0 1 0 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 1 0 0 0
 0 0 0 0 0 1 1 1 1 0 1 0 0 0 1 1 1 0 1 1 1 1 1 0 1 1 0 1 0 0 0 0 1 0 0 0 0
 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 1]
```

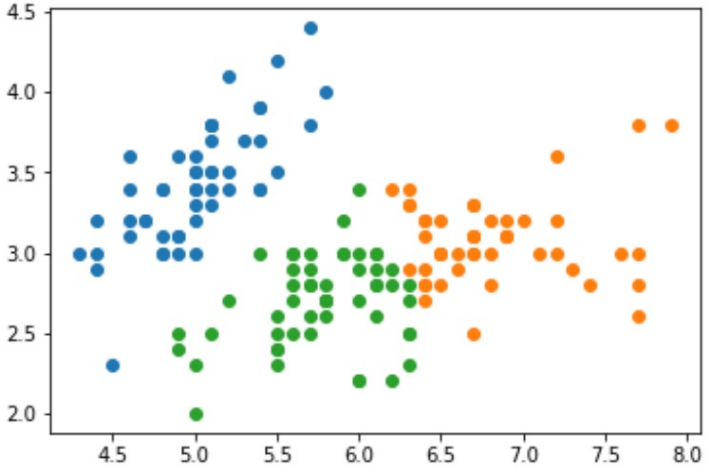


クラスタ数を変える

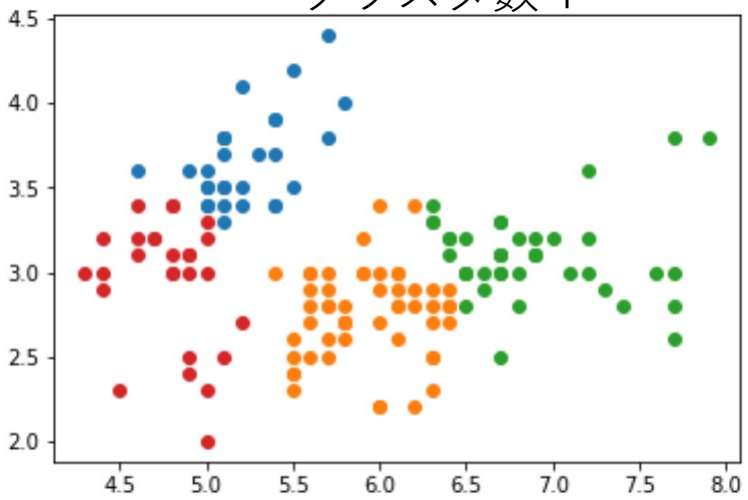
クラスタ数２



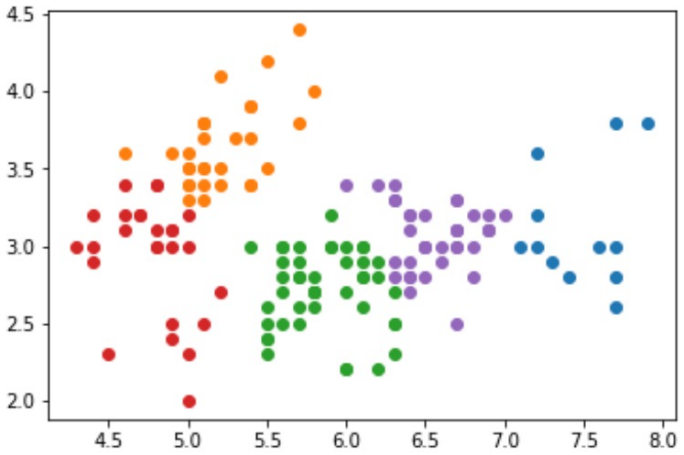
クラスタ数３



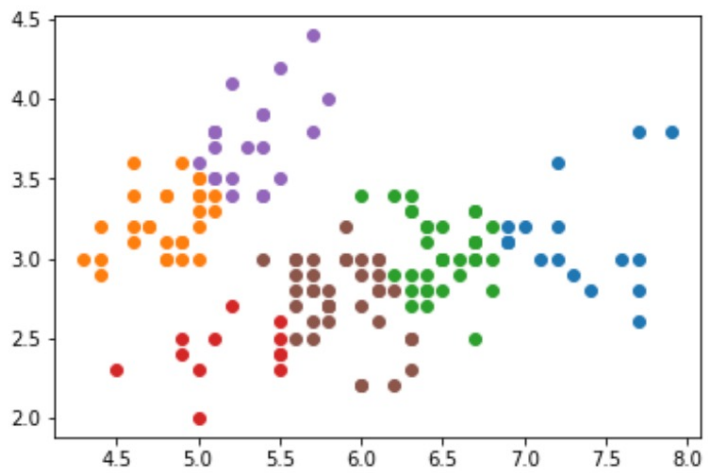
クラスタ数４



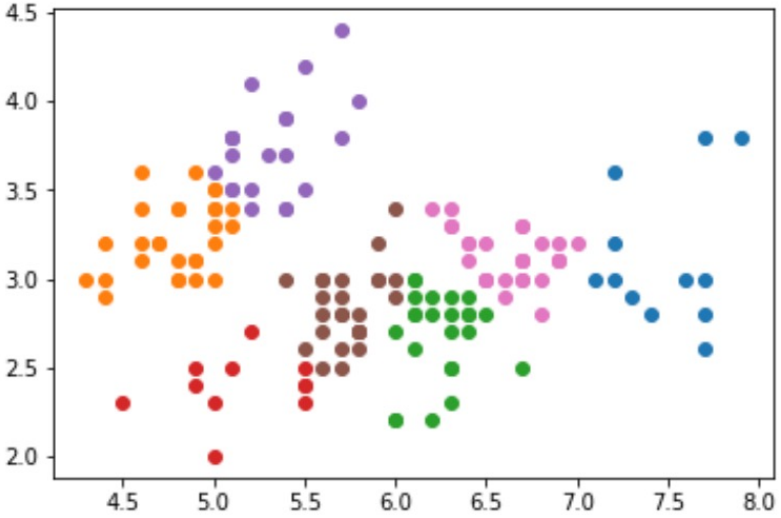
クラスタ数５



クラスタ数６



クラスタ数７

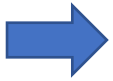


3次元以上のクラスタリング

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

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

```
model = KMeans(n_clusters=3,random_state=0)
model.fit(gaku)
cluster = model.predict(gaku)
print(cluster)
plt.figure()
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()
```

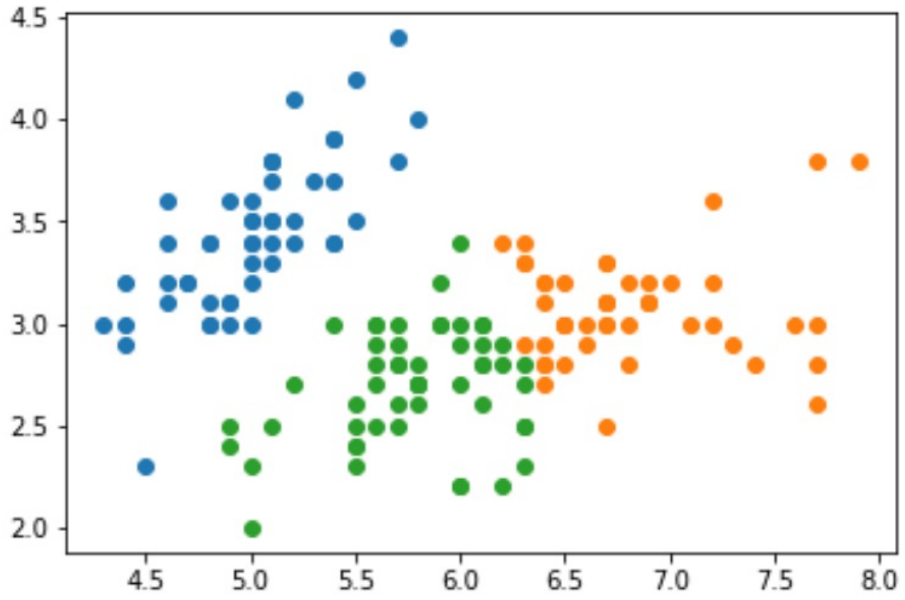


```
model = KMeans(n_clusters=3,random_state=0)
model.fit(data)
cluster = model.predict(data)
print(cluster)
plt.figure()
plt.scatter(data[cluster == 0,0], data[cluster == 0,1])
plt.scatter(data[cluster == 1,0], data[cluster == 1,1])
plt.scatter(data[cluster == 2,0], data[cluster == 2,1])
plt.show()
```

3次元以上のクラスタリング

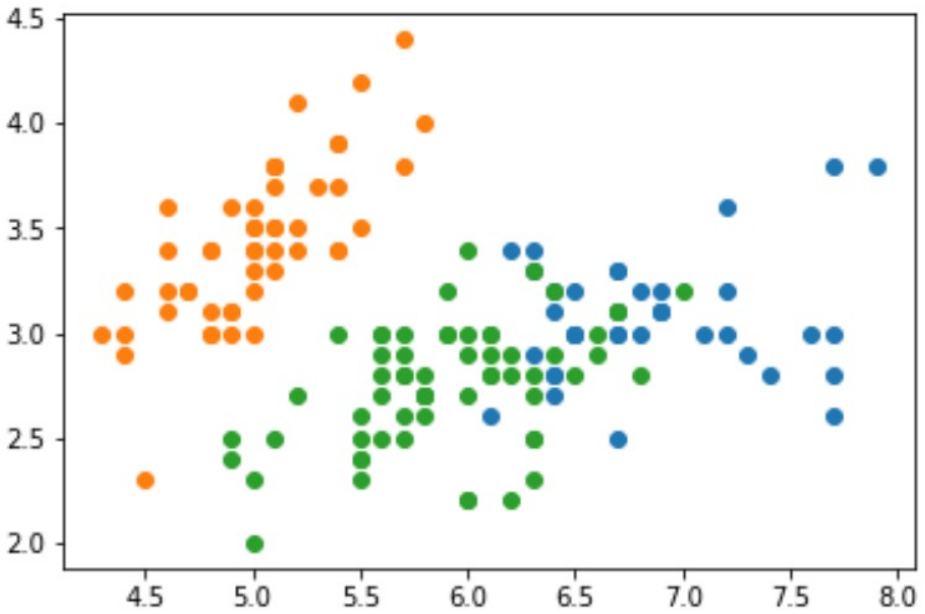
同じクラスタ数(0,1,2)だが、学習するデータが変わり(2次元→4次元)、クラスタ番号の割り振りも変わっている。
4次元のデータによるクラスタリングを2次元で可視化しているのでやや混ざっている部位もあり。

```
In [85]: print(cluster)
[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 0 1 1 1 2 1 2 1 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2
 1 1 1 1 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1 1 2 1 1 1 1 1
 1 1 2 2 1 1 1 1 2 1 2 1 2 1 1 2 2 1 1 1 1 1 2 2 1 1 1 2 1 1 1 2 1 1 1 2 1
 1 2]
```



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

```
In [87]: 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 1 1 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 0 0 2 0 0 0 0
 0 0 2 2 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0 0 0 2 0 0
 0 2]
```



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

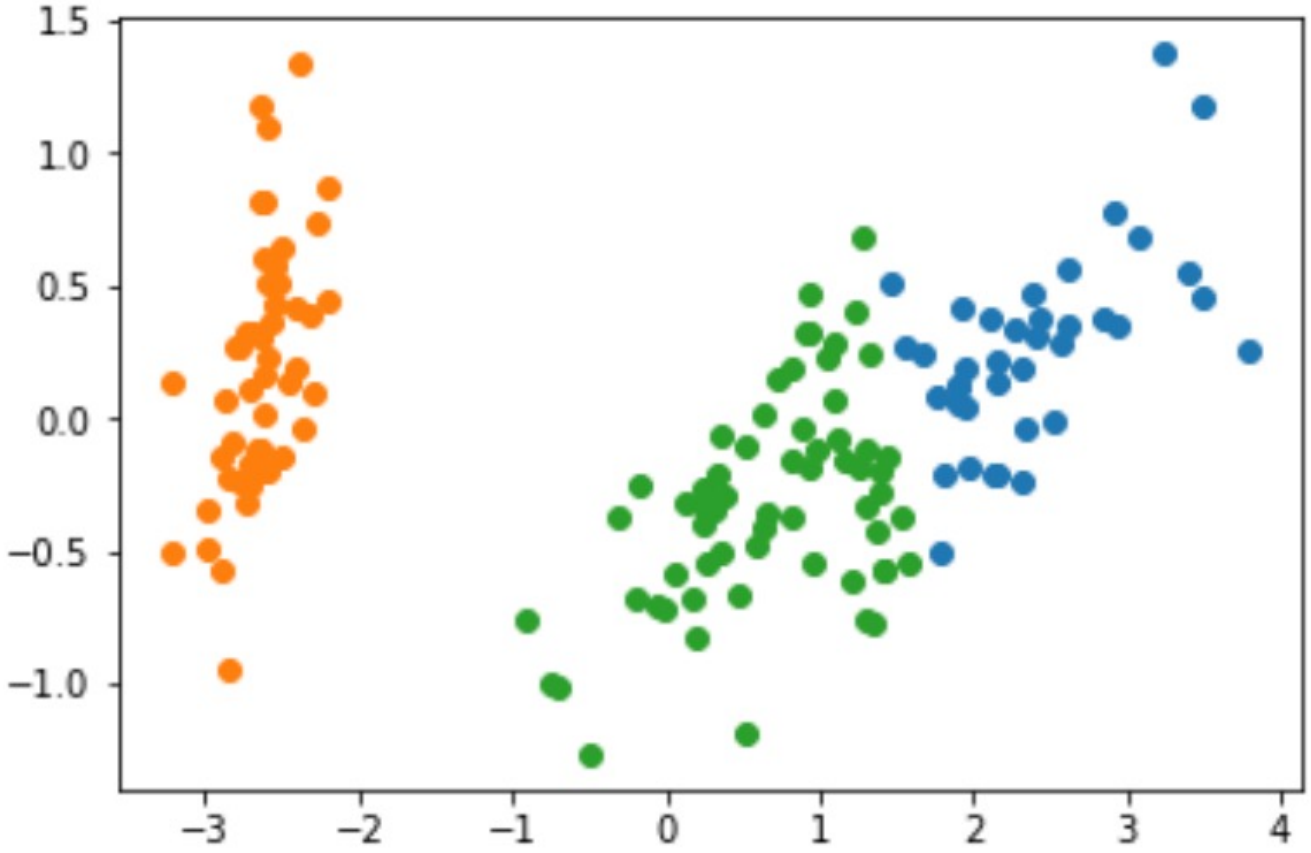
4次元は可視化出来ない　→　次元削減

```
model = KMeans(n_clusters=3,random_state=0)
model.fit(data)
cluster = model.predict(data)
print(cluster)
plt.figure()
plt.scatter(data[cluster == 0,0], data[cluster == 0,1])
plt.scatter(data[cluster == 1,0], data[cluster == 1,1])
plt.scatter(data[cluster == 2,0], data[cluster == 2,1])
plt.show()
```

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

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

plt.figure()
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

アヤメのデータの花びらの長さと幅を使ってKMeans法によるクラスタリングを実施して下さい
・クラスタ数を3つにしてrandom_stateは１にして下さい
　X軸に花びらの長さ、Y軸に花びらの幅にして散布図を提出して下さい
　ファイル名は”cluster1_学籍番号_名前.png”にしてください

・クラスタ数を5つにしてrandom_stateは4にして下さい
　X軸に花びらの長さ、Y軸に花びらの幅にして散布図を提出して下さい
　ファイル名は”cluster2_学籍番号_名前.png”にしてください

・次の講義の感想を記載して下さい

最後に2つのアンケートに答えてください
【全学科共通科目　授業終了後アンケート】
　「医歯学融合教育」→「IL2200071 全学科共通科目 授業終了後アンケート 2022」
　　→「医療とAI・ビッグデータ応用」
【授業後アンケートのお願い】
　このWebClass上(医療とAI・ビッグデータ応用)の一番上にある追加アンケート