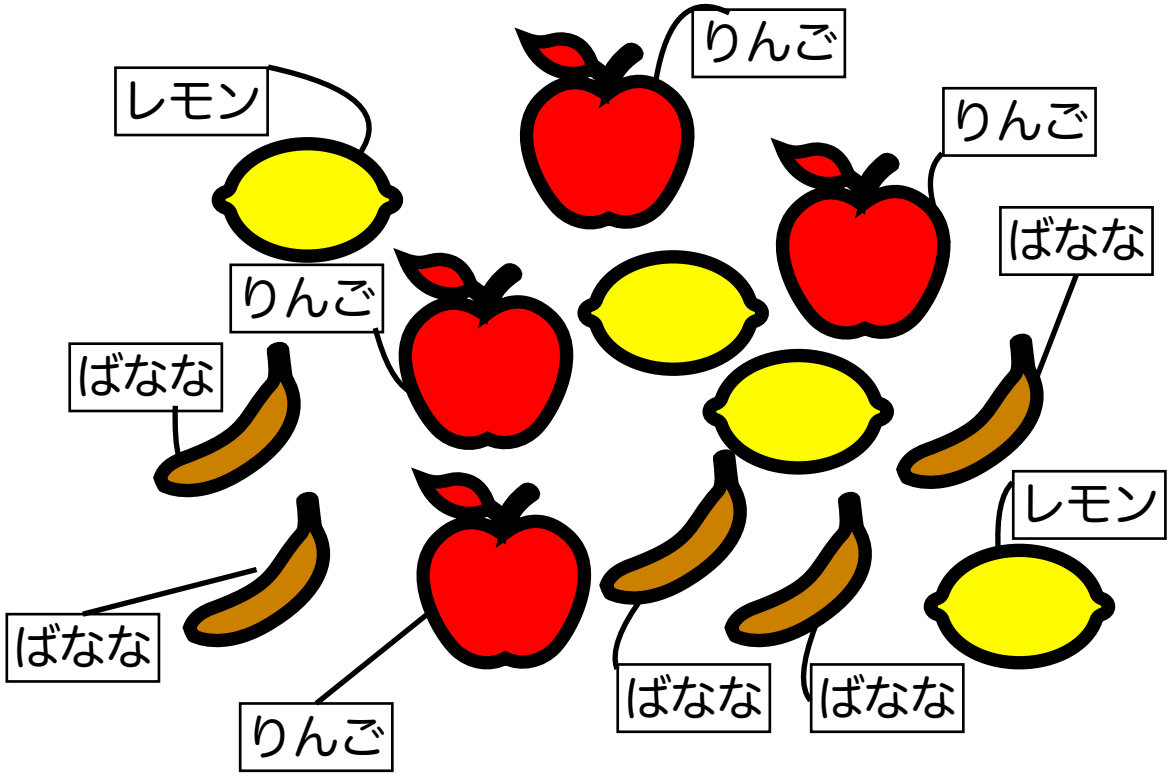
医療とAI・ビッグデータ応用

教師なし機械学習1
(次元削減)

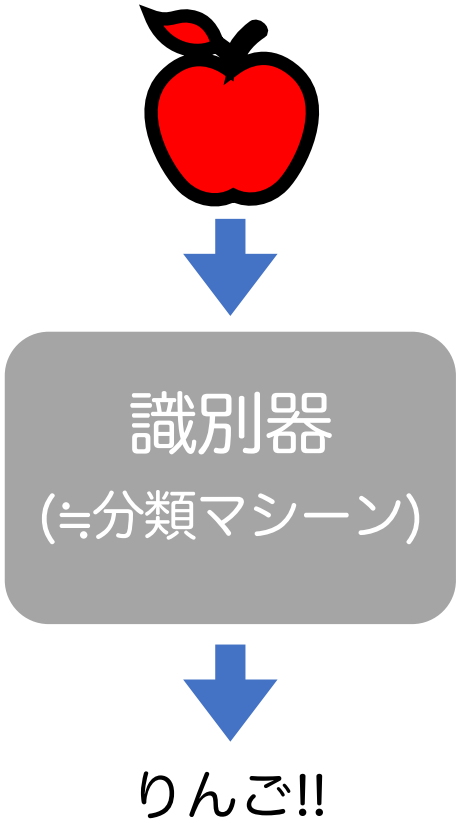
統合教育機構
須藤毅顕

教師あり機械学習の予測



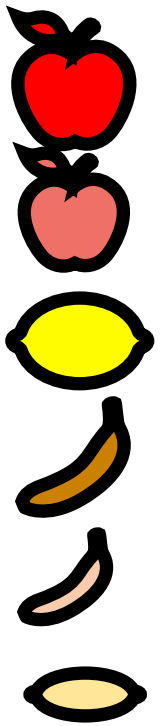
入力

学習



教師あり学習は正解をセットで学習させて識別器を作る

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

あやめ(iris)のデータの読み込み

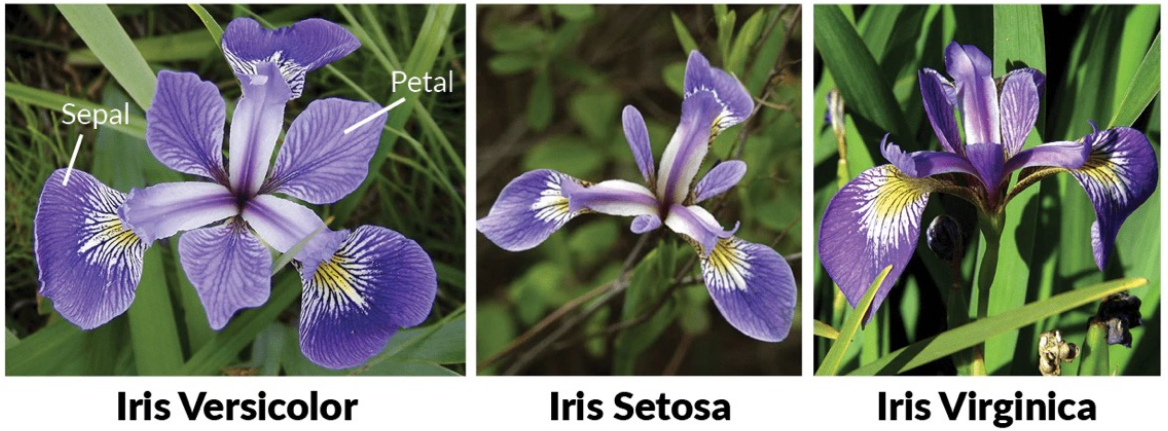
あやめのデータセットはscikit-learnに入っている

```
from sklearn.datasets import load_iris
```

```
iris = load_iris()
iris
```

load_iris()で読み込みirisという変数に格納
そして表示

あやめのデータ



変数4つ
- Sepal(がく片)の長さと幅
- Petal(花びら)の長さと幅

正解が3種類
- ヒオウギアヤメ(0)
- ブルーフラッグ(1)
- バージニカ(2)

あやめ(iris)のデータの中身

```
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],
```

辞書型のような構造

{key:value, key:value, ...}

後半部分

```
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]]),
 'target': array([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, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 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, 2, 2, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 150 (50 in each of three classes)\n    :Number of Attributes: 4 numeric, predictive
attributes and the class\n    :Attribute Information:\n        - sepal length in cm\n        - sepal width in cm\n        - petal length in cm\n        - petal width in cm\n        - class:\n                - Iris-Setosa\n
- Iris-Versicolour\n                - Iris-Virginica\n                \n    :Summary Statistics:\n\n    ============== ==== ==== ======= ===== ====================\n                    Min  Max   Mean    SD   Class
Correlation\n    ============== ==== ==== ======= ===== ====================\n    sepal length:   4.3  7.9   5.84   0.83    0.7826\n    sepal width:    2.0  4.4   3.05   0.43   -0.4194\n    petal length:   1.0  6.9   3.76
1.76    0.9490  (high!)\n    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n    ============== ==== ==== ======= ===== ====================\n\n    :Missing Attribute Values: None\n    :Class Distribution: 33.3%
for each of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The dataset is taken\nfrom
Fisher\'s paper. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning Repository, which has two wrong data points.\n\nThis is perhaps the best known database to be found in the\npattern recognition
literature.  Fisher\'s paper is a classic in the field and\nis referenced frequently to this day.  (See Duda & Hart, for example.)  The\ndata set contains 3 classes of 50 instances each, where each class refers to a\ntype
of iris plant.  One class is linearly separable from the other 2; the\nlatter are NOT linearly separable from each other.\n\n.. topic:: References\n\n   - Fisher, R.A. "The use of multiple measurements in taxonomic
problems"\n     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n     Mathematical Statistics" (John Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene
Analysis.\n     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.\n   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System\n     Structure and Classification Rule for Recognition in
Partially Exposed\n     Environments".  IEEE Transactions on Pattern Analysis and Machine\n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions\n
on Information Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptual clustering system finds 3 classes in the data.\n   - Many, many more ...',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'iris.csv',
 'data_module': 'sklearn.datasets.data'}
```

あやめ(iris)のデータの中身

```
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],
```

辞書型のような構造

{key:value, key:value, ...}

```
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],
```

iris.dataでkey(‘data’)のvalue(中身)を取り出せます

あやめ(iris)のデータの中身

```
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],
```

辞書型のような構造

{key:value, key:value, ...}

```
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],
```

iris.dataでkey(‘data’)のvalue(中身)を取り出せます

```
iris.data.shape
```

```
(150, 4)
```

iris.dataはnumpy配列になっているので、shapeでその形状を確認すると150行4列の2次元配列

あやめ(iris)のデータの中身

```
iris.target
```

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

iris.targetでkey(‘target’)の中身を取り出せます(1次元配列(numpy配列))

```
        [6.3, 2.5, 5. , 1.9],
        [6.5, 3. , 5.2, 2. ],
        [6.2, 3.4, 5.4, 2.3],
        [5.9, 3. , 5.1, 1.8]]),
 'target': array([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, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 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, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 150 (50 in each of three classes)\n    :Number of Attributes: 4 numeric, predictive
attributes and the class\n    :Attribute Information:\n        - sepal length in cm\n        - sepal width in cm\n        - petal length in cm\n        - petal width in cm\n        - class:\n                - Iris-Setosa\n
- Iris-Versicolour\n                - Iris-Virginica\n                \n    :Summary Statistics:\n\n    ============== ==== ==== ======= ===== ====================\n                    Min  Max   Mean    SD   Class
Correlation\n    ============== ==== ==== ======= ===== ====================\n    sepal length:   4.3  7.9   5.84   0.83    0.7826\n    sepal width:    2.0  4.4   3.05   0.43   -0.4194\n    petal length:   1.0  6.9   3.76
1.76    0.9490  (high!)\n    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n    ============== ==== ==== ======= ===== ====================\n\n    :Missing Attribute Values: None\n    :Class Distribution: 33.3%
for each of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The dataset is taken\nfrom
Fisher\'s paper. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning Repository, which has two wrong data points.\n\nThis is perhaps the best known database to be found in the\npattern recognition
literature.  Fisher\'s paper is a classic in the field and\nis referenced frequently to this day.  (See Duda & Hart, for example.)  The\ndata set contains 3 classes of 50 instances each, where each class refers to a\ntype
of iris plant.  One class is linearly separable from the other 2; the\nlatter are NOT linearly separable from each other.\n\n.. topic:: References\n\n   - Fisher, R.A. "The use of multiple measurements in taxonomic
problems"\n     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n     Mathematical Statistics" (John Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene
Analysis.\n     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.\n   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System\n     Structure and Classification Rule for Recognition in
Partially Exposed\n     Environments".  IEEE Transactions on Pattern Analysis and Machine\n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions\n
on Information Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptual clustering system finds 3 classes in the data.\n   - Many, many more ...',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'iris.csv',
 'data_module': 'sklearn.datasets.data'}
```

先ほどのirisの出力結果(後半部分)

あやめ(iris)のデータの中身

```
iris.DESCR
```

```
'.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Characterist
ics:**\n\n    :Number of Instances: 150 (50 in each of three classes)\n    :Number of Attr
ibutes: 4 numeric, predictive attributes and the class\n    :Attribute Information:\n
- sepal length in cm\n        - sepal width in cm\n        - petal length in cm\n        -
petal width in cm\n        - class:\n                - Iris-Setosa\n                - Iris
-Versicolour\n                - Iris-Virginica\n                \n    :Summary Statistics:
\n\n    ============== ==== ==== ======= ===== ====================\n                    M
in  Max   Mean    SD   Class Correlation\n    ============== ==== ==== ======= ===== =====
===============\n    sepal length:   4.3  7.9   5.84   0.83    0.7826\n    sepal width:
2.0  4.4   3.05   0.43   -0.4194\n    petal length:   1.0  6.9   3.76   1.76    0.9490  (h
igh!)\n    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n    ============== =
=== ==… '
```

iris.DESCRでkey(‘DESCR’)
の中身を取り出せます
これは説明文書で文字列データ

先ほどのirisの出力結果(後半部分)

```
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]]),
 'target': array([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, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 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, 2, 2, 2, 2]),
 'frame': None,
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 150 (50 in each of three classes)\n    :Number of Attributes: 4 numeric, predictive
attributes and the class\n    :Attribute Information:\n        - sepal length in cm\n        - sepal width in cm\n        - petal length in cm\n        - petal width in cm\n        - class:\n                - Iris-Setosa\n
- Iris-Versicolour\n                - Iris-Virginica\n                \n    :Summary Statistics:\n\n    ============== ==== ==== ======= ===== ====================\n                    Min  Max   Mean    SD   Class
Correlation\n    ============== ==== ==== ======= ===== ====================\n    sepal length:   4.3  7.9   5.84   0.83    0.7826\n    sepal width:    2.0  4.4   3.05   0.43   -0.4194\n    petal length:   1.0  6.9   3.76
1.76    0.9490  (high!)\n    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n    ============== ==== ==== ======= ===== ====================\n\n    :Missing Attribute Values: None\n    :Class Distribution: 33.3%
for each of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The dataset is taken\nfrom
Fisher\'s paper. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning Repository, which has two wrong data points.\n\nThis is perhaps the best known database to be found in the\npattern recognition
literature.  Fisher\'s paper is a classic in the field and\nis referenced frequently to this day.  (See Duda & Hart, for example.)  The\ndata set contains 3 classes of 50 instances each, where each class refers to a\ntype
of iris plant.  One class is linearly separable from the other 2; the\nlatter are NOT linearly separable from each other.\n\n.. topic:: References\n\n   - Fisher, R.A. "The use of multiple measurements in taxonomic
problems"\n     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n     Mathematical Statistics" (John Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene
Analysis.\n     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.\n   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System\n     Structure and Classification Rule for Recognition in
Partially Exposed\n     Environments".  IEEE Transactions on Pattern Analysis and Machine\n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions\n
on Information Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptual clustering system finds 3 classes in the data.\n   - Many, many more ...',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'iris.csv',
 'data_module': 'sklearn.datasets.data'}
```

あやめ(iris)のデータの中身

```
print(iris.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================
```

print(iris.DESCR)でみやすくなります

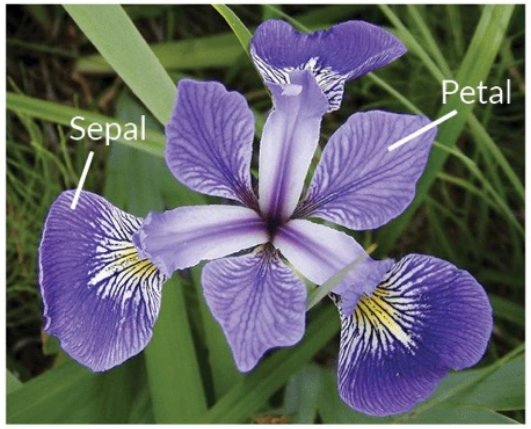
150件データで1クラス50件の3クラス

4つの特徴量
- sepal length(cm)：萼片の長さ
- sepal width(cm)：萼片の幅
- petal length(cm)：花びらの長さ
- petal width(cm)：花びらの幅

3つのクラス
- Iris-Setosa：ヒオウギアヤメ(0)
- Iris-Versicolour：ブルーフラッグ(1)
- Iris-Virginica：バージニカ(2)

あやめ(iris)のデータの中身

あやめのデータ



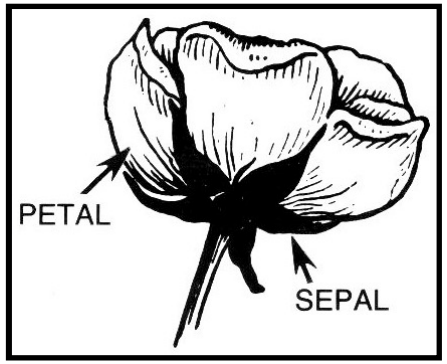
Iris Versicolor

Iris Setosa

Iris Virginica

変数4つ
- Sepal(がく片)の長さと幅
- Petal(花びら)の長さと幅

正解が3種類
- ヒオウギアヤメ(0)
- ブルーフラッグ(1)
- バージニカ(2)



print(iris.DESCR)でみやすくなります

150件データで1クラス50件の3クラス

4つの特徴量
- sepal length(cm)：萼片の長さ
- sepal width(cm) ：萼片の幅
- petal length(cm)：花びらの長さ
- petal width(cm) ：花びらの幅

3つのクラス
- Iris-Setosa ：ヒオウギアヤメ(0)
- Iris-Versicolour：ブルーフラッグ(1)
- Iris-Virginica ：バージニカ(2)

あやめ(iris)のデータの中身

`iris.data` (150,4)の2次元配列

```
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],
```

萼片の長さ / 萼片の幅 / 花びらの長さ / 花びらの幅

←1個目のアヤメ
←2個目のアヤメ
←3個目のアヤメ
…
…
…
←150個目のアヤメ

iris.dataが150個のアヤメの特徴量
iris.targetが150個のアヤメの正解ラベル

`iris.target` (150,)の1次元配列

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

iris.targetの
1-50が0　(=ヒオウギアヤメ)
51-100が1　(=ブルーフラッグ)
101-151が2 (=バージニカ)

あやめ(iris)のデータの中身

```
iris.data
```

(150,4)の2次元配列

```
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],
```

←1個目のアヤメ
←2個目のアヤメ
←3個目のアヤメ
…
…
…
←150個目のアヤメ

萼片の長さ	萼片の幅	花びらの長さ	花びらの幅

iris.dataが150個のアヤメの特徴量
iris.targetが150個のアヤメの正解ラベル

```
iris.target
```

(150,)の1次元配列

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

iris.targetの
1-50が0　　(=ヒオウギアヤメ)
51-100が1　(=ブルーフラッグ)
101-151が2 (=バージニカ)

```
print(iris.data[0])
print(iris.target[0])
```

```
[5.1 3.5 1.4 0.2]
0
```

iris.data[0]が1個目のアヤメの特徴量
iris.target[0]が1個目のアヤメの正解ラベル

次元削減の1つ：主成分分析

主成分分析は、教師なしの機械学習で、多次元の特徴量の「データの相関関係」を
失わないでデータの特徴を圧縮することが出来る。

最大のメリットは多次元なのを２次元や３次元にすることが出来るので可視化出来る。

	がく片の長さ	がく片の幅	花びらの長さ	花びらの幅	アヤメの種類
0	5.1	3.5	1.4	0.2	ヒオウギアヤメ
1	4.9	3.0	1.4	0.2	ヒオウギアヤメ
2	4.7	3.2	1.3	0.2	ヒオウギアヤメ
3	4.6	3.1	1.5	0.2	ヒオウギアヤメ
4	5.0	3.6	1.4	0.2	ヒオウギアヤメ
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	バージニカ
146	6.3	2.5	5.0	1.9	バージニカ
147	6.5	3.0	5.2	2.0	バージニカ
148	6.2	3.4	5.4	2.3	バージニカ
149	5.9	3.0	5.1	1.8	バージニカ

	主成分１	主成分２
0		
1		
2		
3		
4		
...		
145		
146		
147		
148		
149		

or

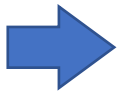
	主成分１	主成分２	主成分３
0			
1			
2			
3			
4			
...			
145			
146			
147			
148			
149			

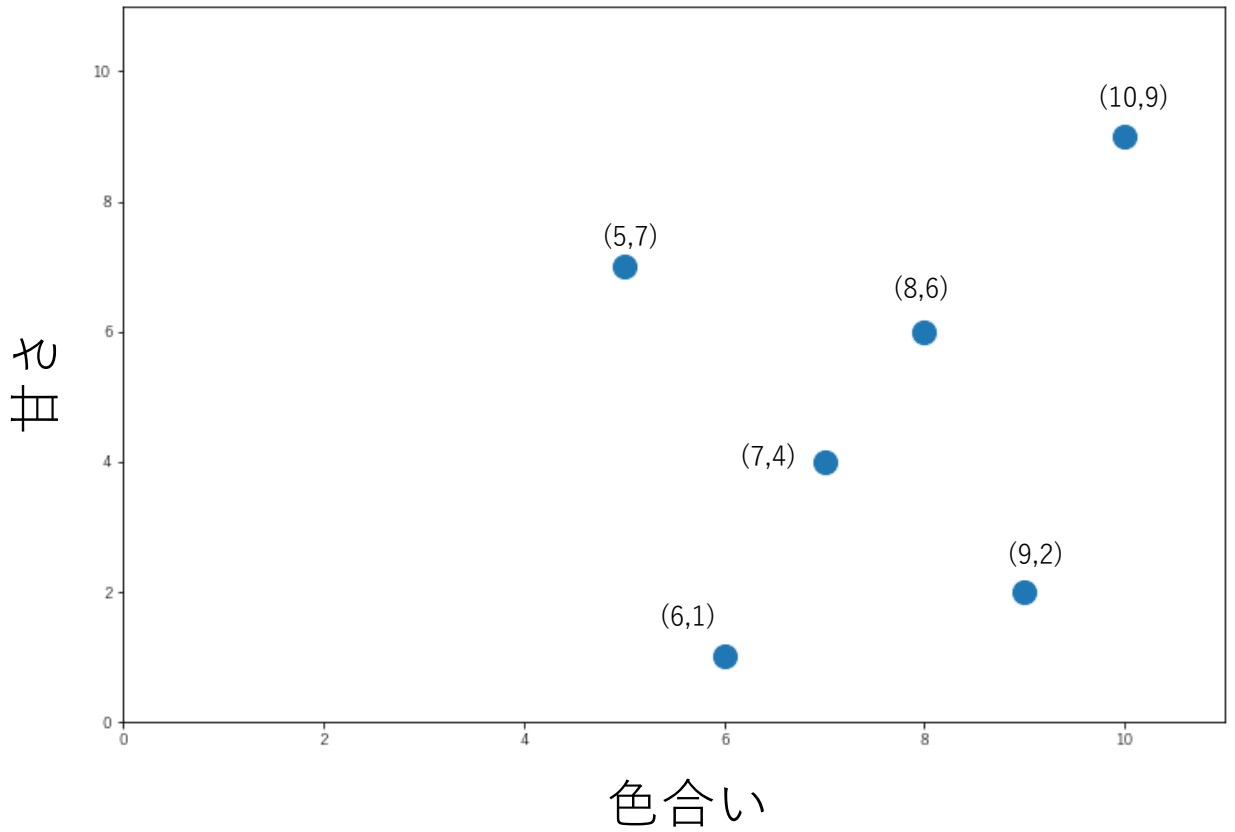
4つの特徴量を

2つや3つの特徴量に減らすことが出来る

どうやって新しい特徴量を作成しているのか？

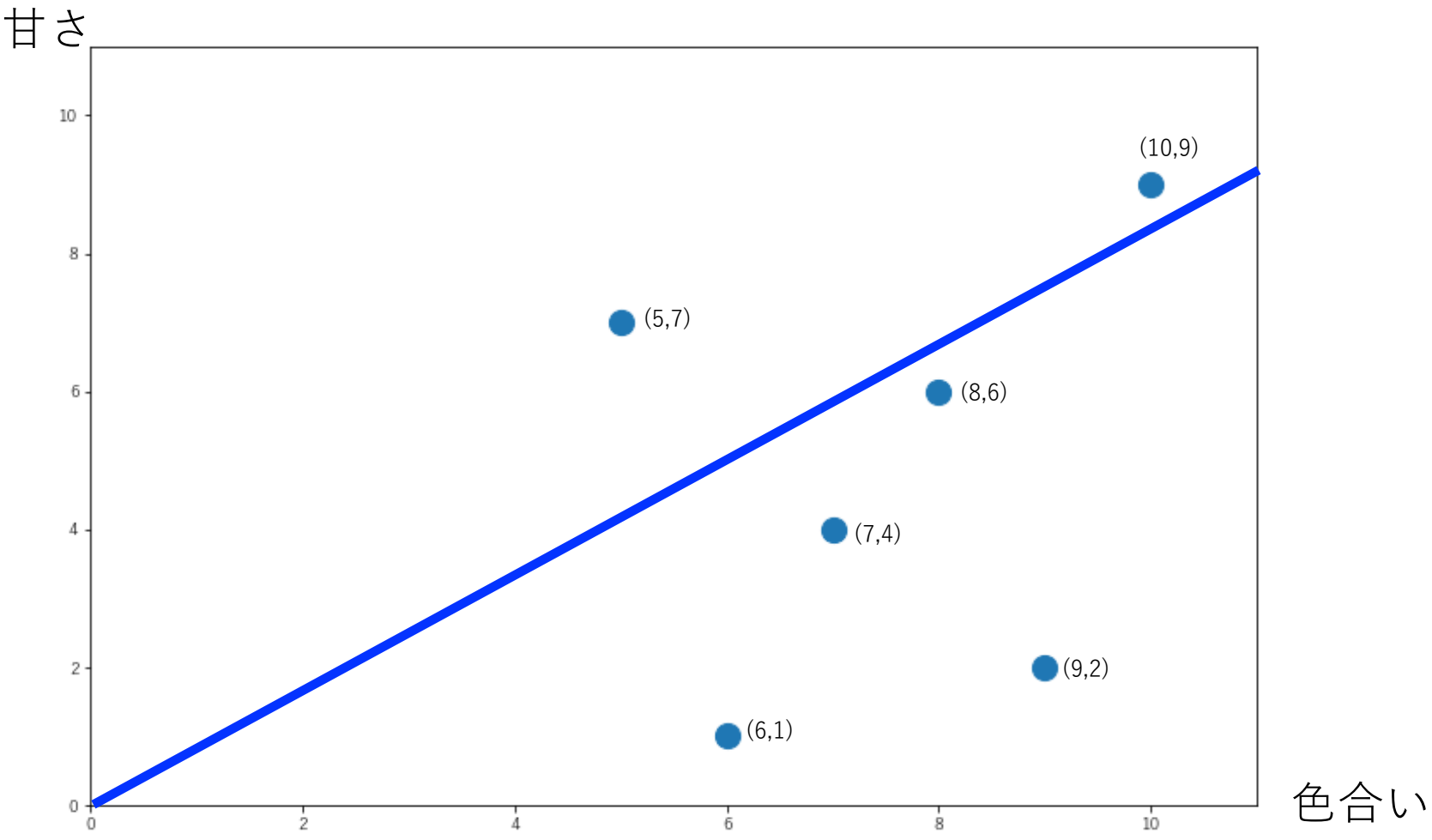
	色合い	大きさ	甘さレベル
りんご1	10	100	9
りんご2	8	88	6
れもん1	9	80	2
バナナ1	5	73	7
バナナ2	7	50	4
れもん2	6	40	1





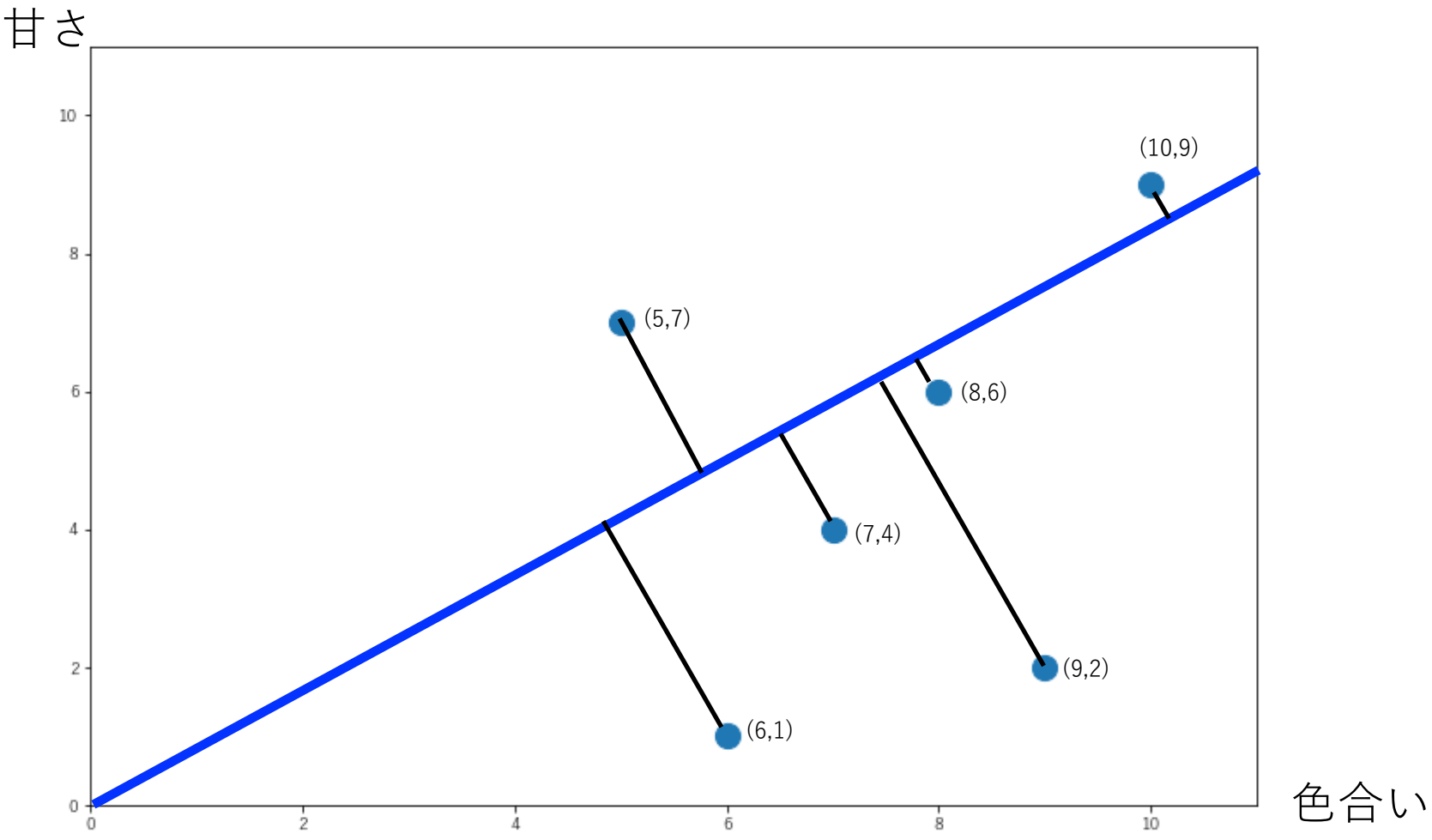
甘さと色合いの2つの特徴量から新たな1つの特徴量を作ることを考えます
甘さと色合いに着目して散布図を作ります

どうやって新しい特徴量を作成しているのか？



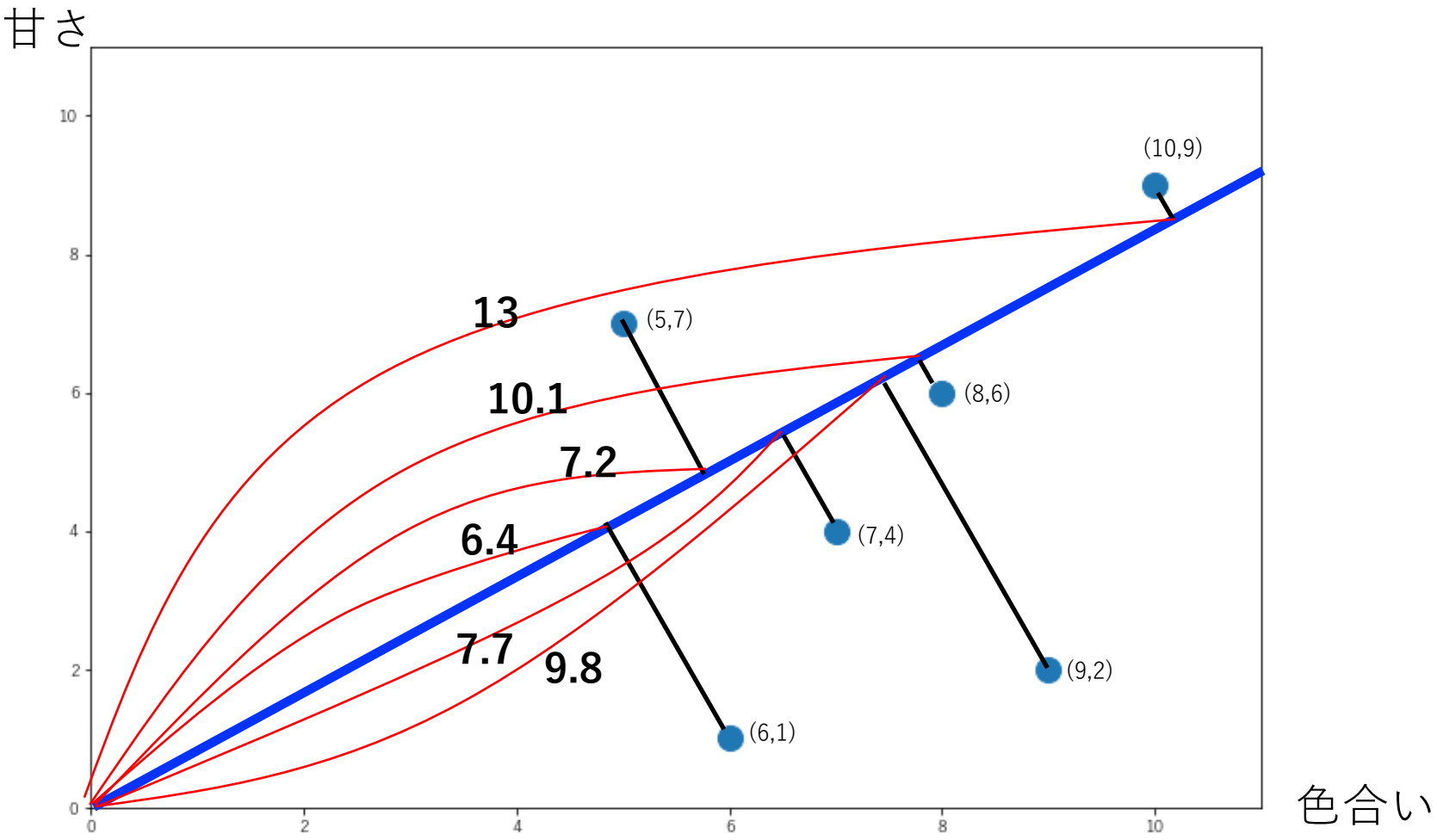
原点を通る新しい軸を作成する（とりあえず）

どうやって新しい特徴量を作成しているのか？



各点を新しい軸上に移す

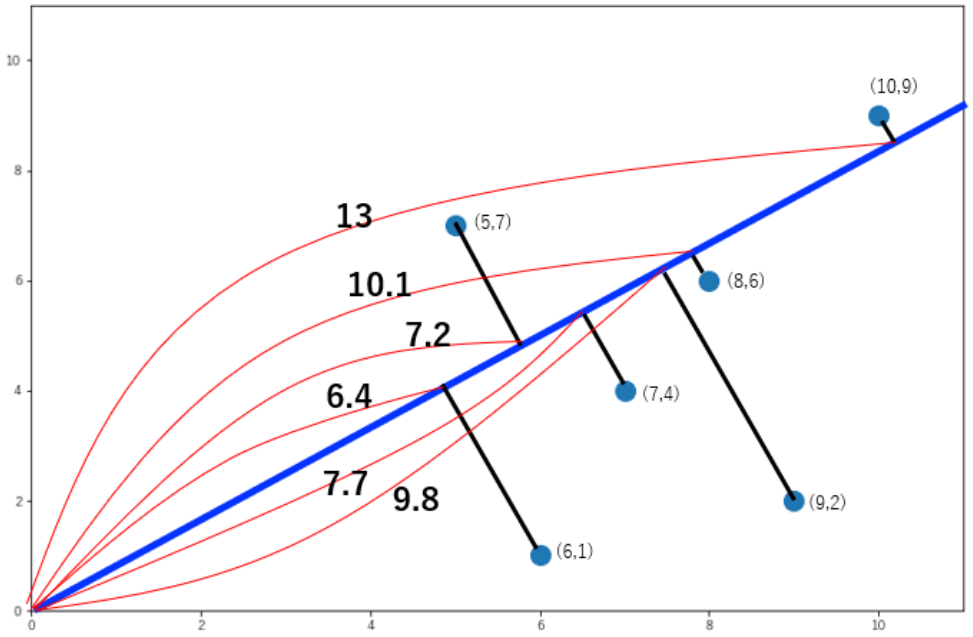
どうやって新しい特徴量を作成しているのか？



この時の原点からの距離が新しい特徴量の値になります

どうやって新しい特徴量を作成しているのか？

甘さ



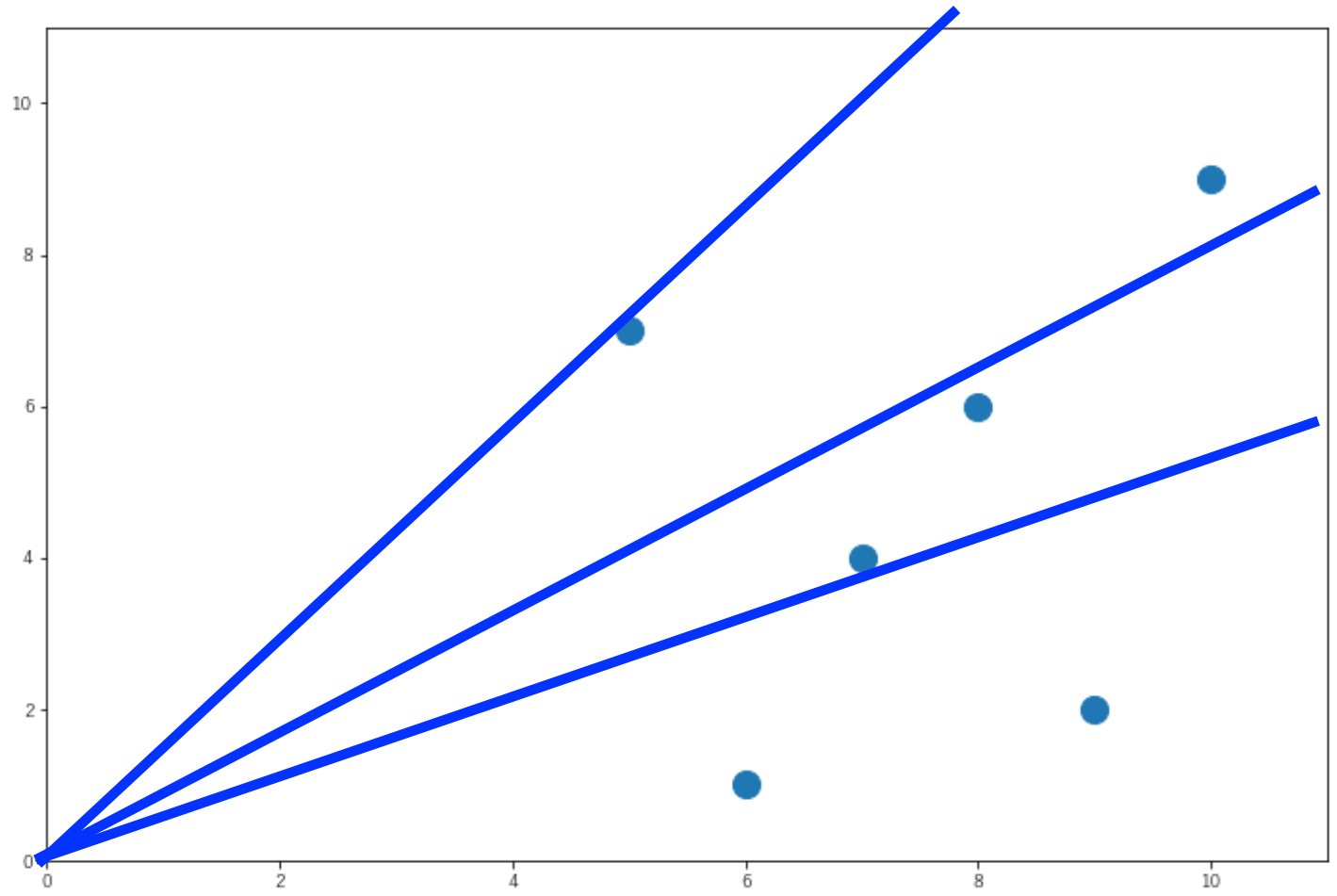
色合い

	色合い	甘さレベル
りんご1	10	9
りんご２	8	6
れもん1	9	2
バナナ1	5	7
バナナ2	7	4
れもん2	6	1

	新たな特徴量
りんご1	13
りんご２	10.1
れもん1	9.8
バナナ1	7.2
バナナ2	7.7
れもん2	6.4

この時の原点からの距離が新しい特徴量の値になります
これで2つの特徴量が1つに集約されたことになります

新しい軸はどのようにして決まるのか

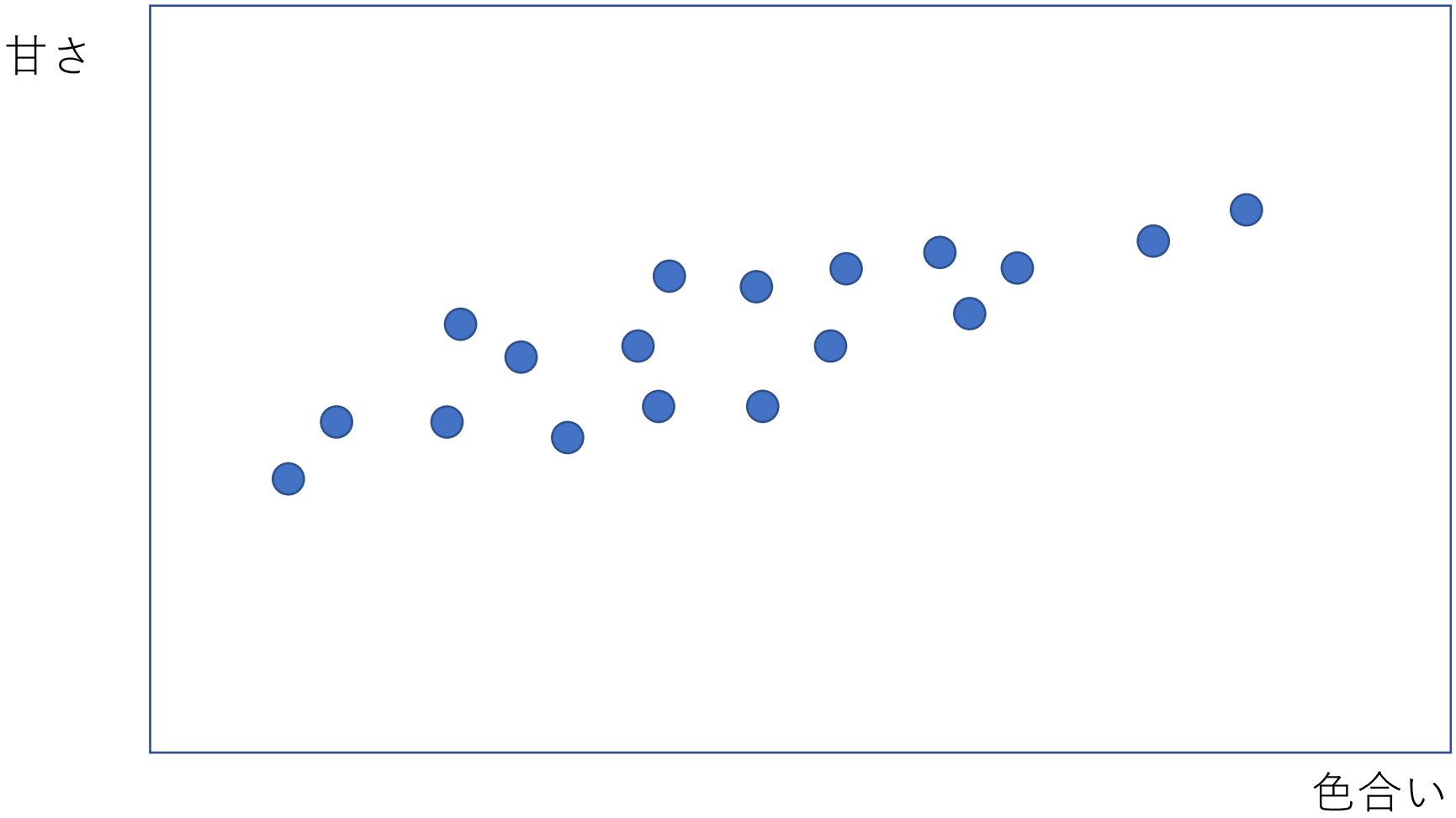


モデルがデータを学習することによって最適な軸を決定している

具体的には、データを移した時の分散(ばらつき)が最大になる様な軸を選ぶ

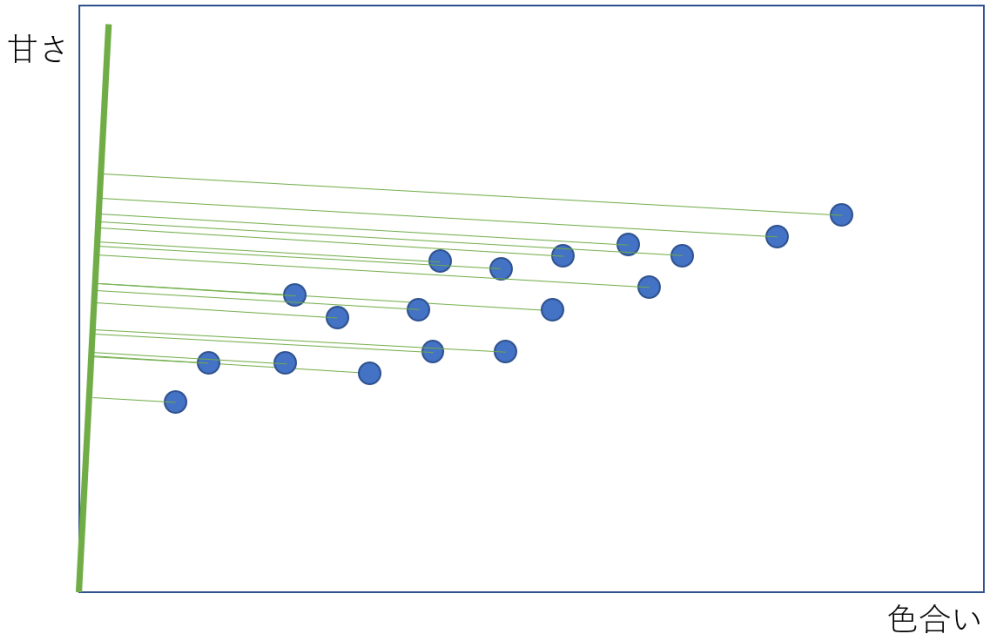
新しい軸はどのようにして決まるのか

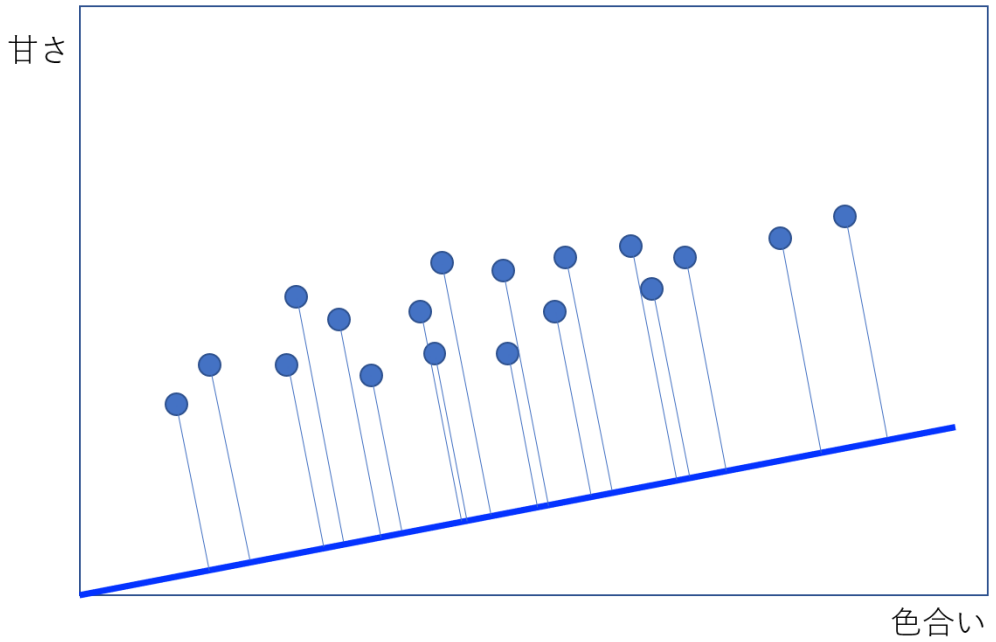
例えば次のようなデータがあったとします



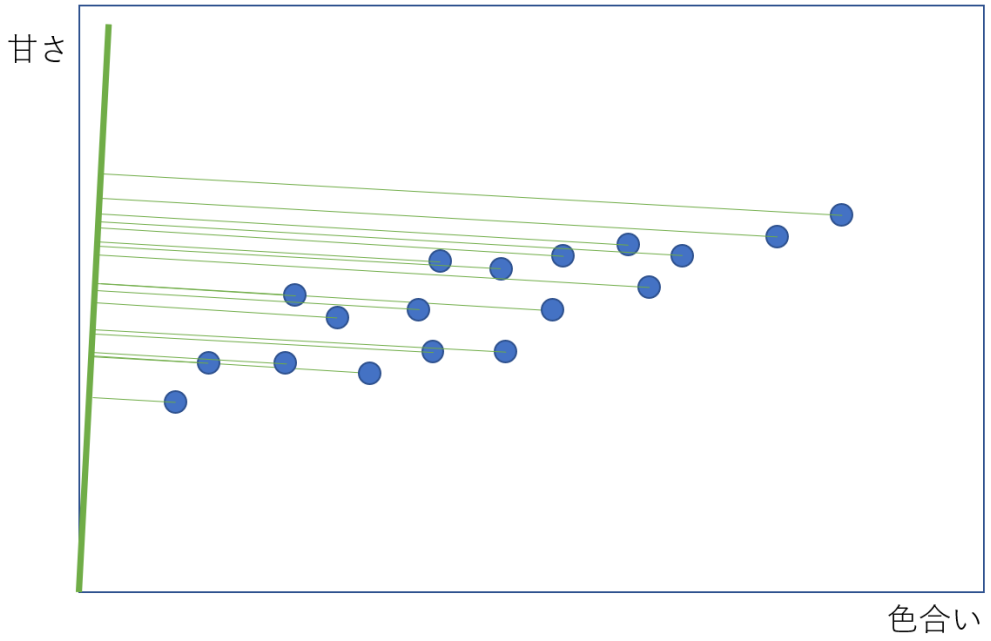
目的は2つの変数(甘さと色合い)から新たな変数を作りたい

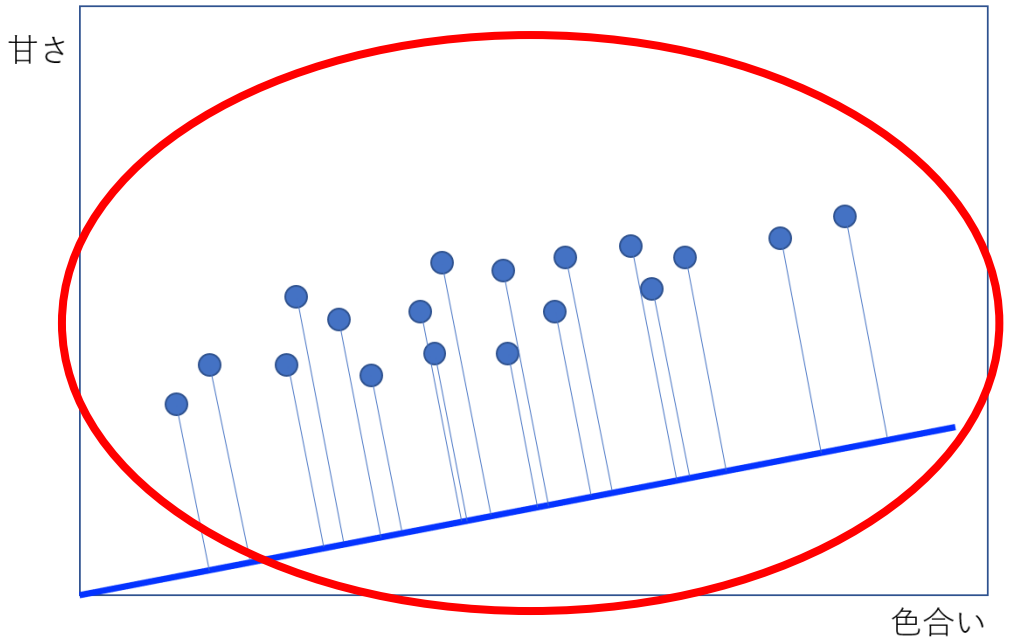
次の2本の軸はどちらの方が甘さと色合いの情報がより保存されているでしょうか？
主成分分析は「データの相関関係」を失わないでデータを圧縮することを考えます。



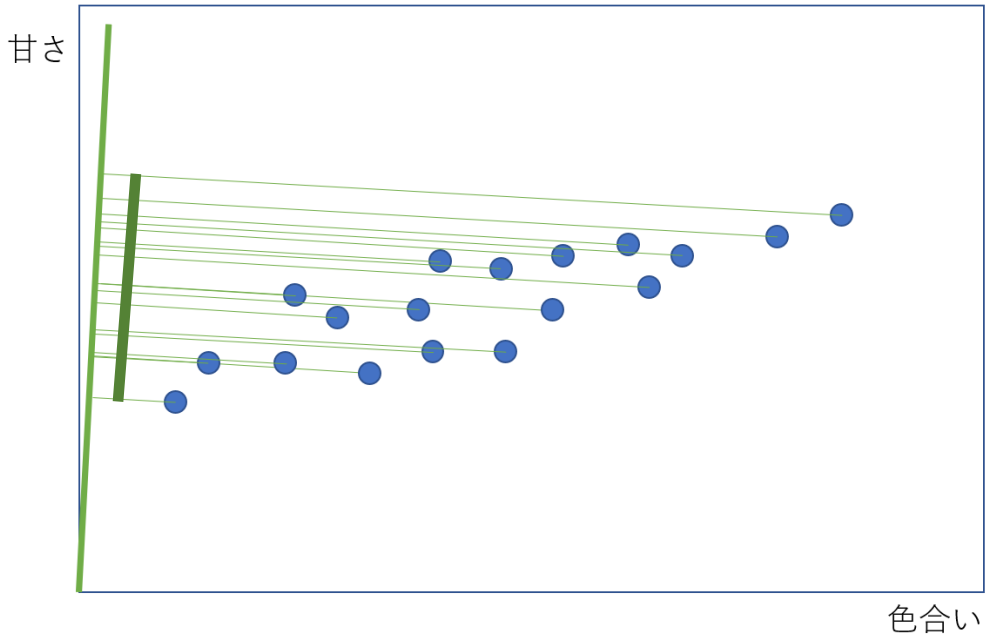


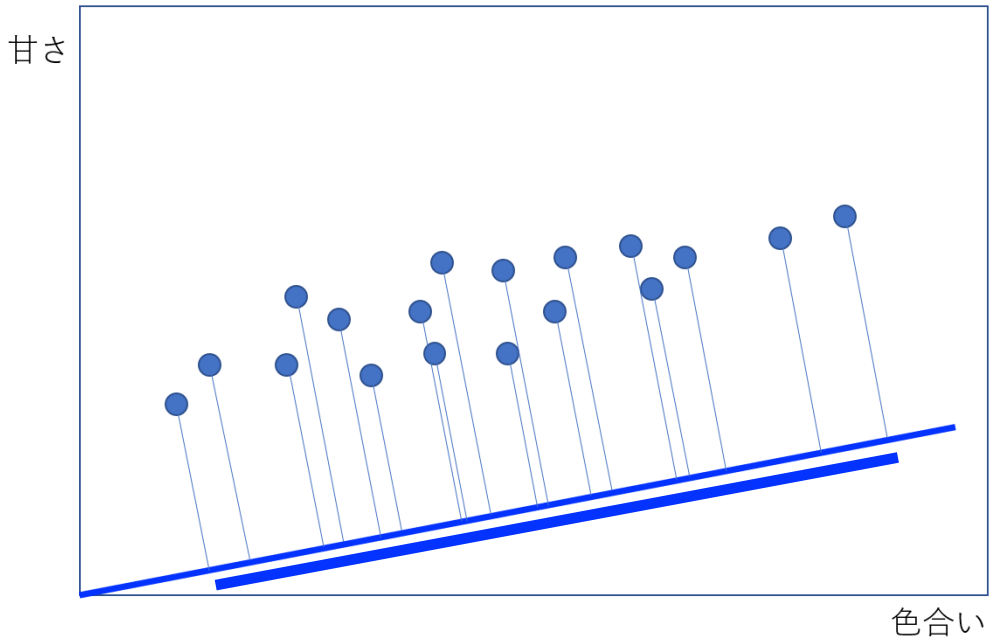
次の2本の軸はどちらの方が甘さと色合いの情報がより保存されているでしょうか？
主成分分析は「データの相関関係」を失わないでデータを圧縮することを考えます。





次の2本の軸はどちらの方が甘さと色合いの情報がより保存されているでしょうか？
主成分分析は「データの相関関係」を失わないでデータを圧縮することを考えます。



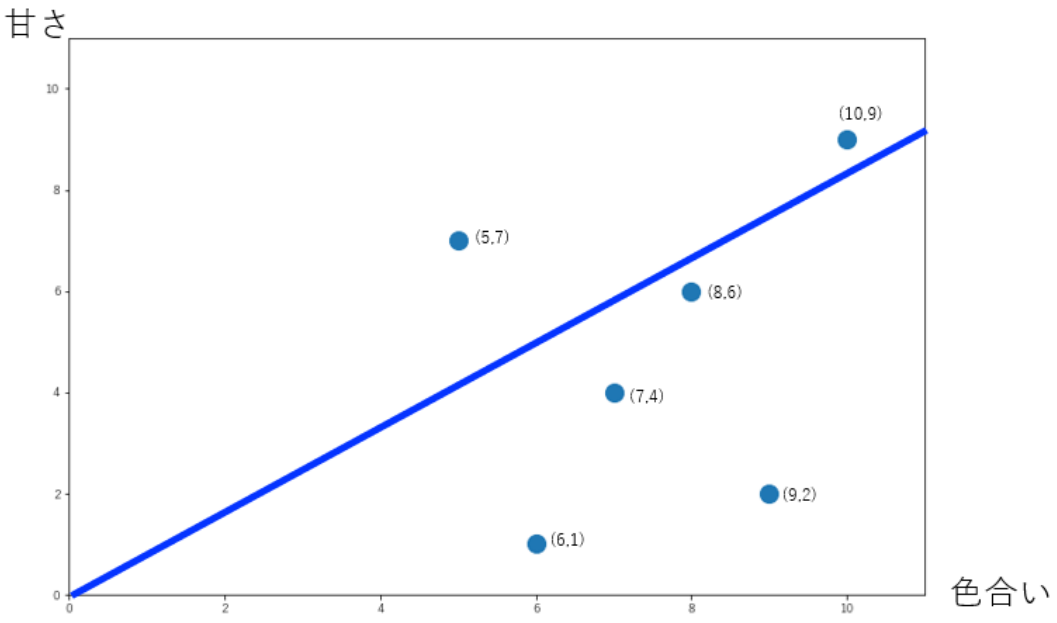


　緑の軸の方がばらつきが小さい
＝似たデータになってしまっている
＝失っている情報が多い
→ばらつき(＝分散)が大きい軸ほどより2つの変数の情報が保存された新しい軸

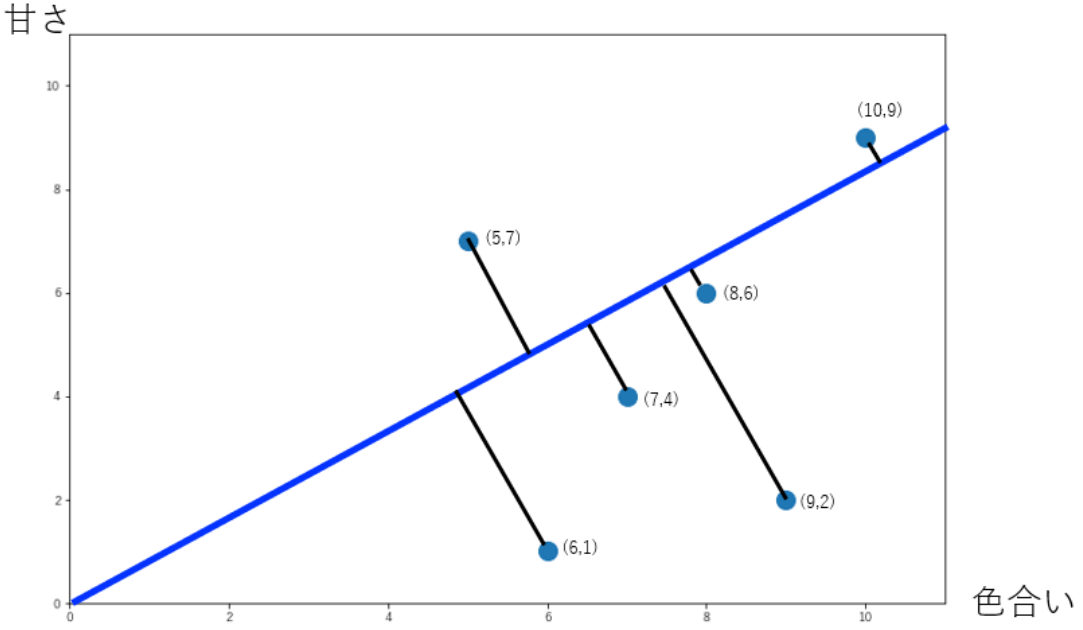
主成分分析(PCA)を実行する

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

モデル名 = PCA()でモデルを作成
(深層学習のmodel = Sequential()と同様)
n_components=~は、作りたい主成分の数を入力する
モデル名.fit_transform(特徴量)で学習と新しい軸へのあてはめを行う



学習（＝新しい軸の作成）



軸へのデータのあてはめ

主成分分析(PCA)を実行する

```
result.shape
```

```
(150, 2)
```

```
result
```

```
array([[-2.68412563,  0.31939725],
       [-2.71414169, -0.17700123],
       [-2.88899057, -0.14494943],
       [-2.74534286, -0.31829898],
       [-2.72871654,  0.32675451],
                    ...
       [ 1.41523588, -0.57491635],
       [ 2.56301338,  0.2778626 ],
       [ 2.41874618,  0.3047982 ],
       [ 1.94410979,  0.1875323 ],
       [ 1.52716661, -0.37531698],
       [ 1.76434572,  0.07885885],
       [ 1.90094161,  0.11662796],
       [ 1.39018886, -0.28266094]])
```

実行結果を変数resultに格納
resultはnumpy配列で、(150,2)の2次元配列

主成分分析(PCA)を実行する

```
result.shape
```

```
(150, 2)
```

```
result
```

```
array([[-2.68412563,  0.31939725],
       [-2.71414169, -0.17700123],
       [-2.88899057, -0.14494943],
       [-2.74534286, -0.31829898],
       [-2.72871654,  0.32675451],
       ...
       [ 1.41523588, -0.57491635],
       [ 2.56301338,  0.2778626 ],
       [ 2.41874618,  0.3047982 ],
       [ 1.94410979,  0.1875323 ],
       [ 1.52716661, -0.37531698],
       [ 1.76434572,  0.07885885],
       [ 1.90094161,  0.11662796],
       [ 1.39018886, -0.28266094]])
```

実行結果を変数resultに格納
resultはnumpy配列で、(150,2)の2次元配列

```
result[:,0]
```

```
array[-2.68412563, -2.71414169, -2.88899057, -2.74534286, -2.72871654,
 -2.28085963, -2.82053775, -2.62614497, -2.88638273, -2.6727558 ,
 -2.50694709, -2.61275523, -2.78610927, -3.22380374, -2.64475039,
 -2.38603903, -2.62352788, -2.64829671, -2.19982032, -2.5879864 ,
 -2.31025622, -2.54370523, -3.21593942, -2.30273318, -2.35575405,
 -2.50666891, -2.46882007, -2.56231991, -2.63953472, -2.63198939,
 -2.58739848, -2.4099325 , -2.64886233, -2.59873675, -2.63692688,
 -2.86624165, -2.62523805, -2.80068412, -2.98050204, -2.59000631,
 -2.77010243, -2.84936871, -2.99740655, -2.40561449, -2.20948924,
 -2.71445143, -2.53814826, -2.83946217, -2.54308575, -2.70335978,
  1.28482569,  0.93248853,  1.46430232,  0.18331772,  1.08810326,
  0.64166908,  1.09506066, -0.74912267,  1.04413183, -0.0087454 ,
 -0.50784088,  0.51169856,  0.26497651,  0.98493451, -0.17392537,
  0.92786078,  0.66028376,  0.23610499,  0.94473373,  0.04522698,
  1.11628318,  0.35788842,  1.29818388,  0.92172892,  0.71485333,
  0.90017437,  1.33202444,  1.55780216,  0.81329065, -0.30558378,
 -0.06812649, -0.18962247,  0.13642871,  1.38002644,  0.58800644,
  0.80685831,  1.22069088,  0.81509524,  0.24595768,  0.16641322,
  0.46480029,  0.8908152 ,  0.23054802, -0.70453176,  0.35698149,
  0.33193448,  0.37621565,  0.64257601, -0.90646986,  0.29900084,
  2.53119273,  1.41523588,  2.61667602,  1.97153105,  2.35000592,
```

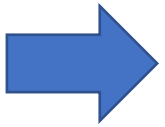
result[:,0]は行の全て、列は1列目を抽出＝第一主成分
(result[:,1]は行全て、列は2列目を抽出＝第二主成分)

主成分分析(PCA)を実行する

元の特徴量(iris.data)
150行4列

```
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],
```



新たな特徴量(result)
150行2列

```
result
```

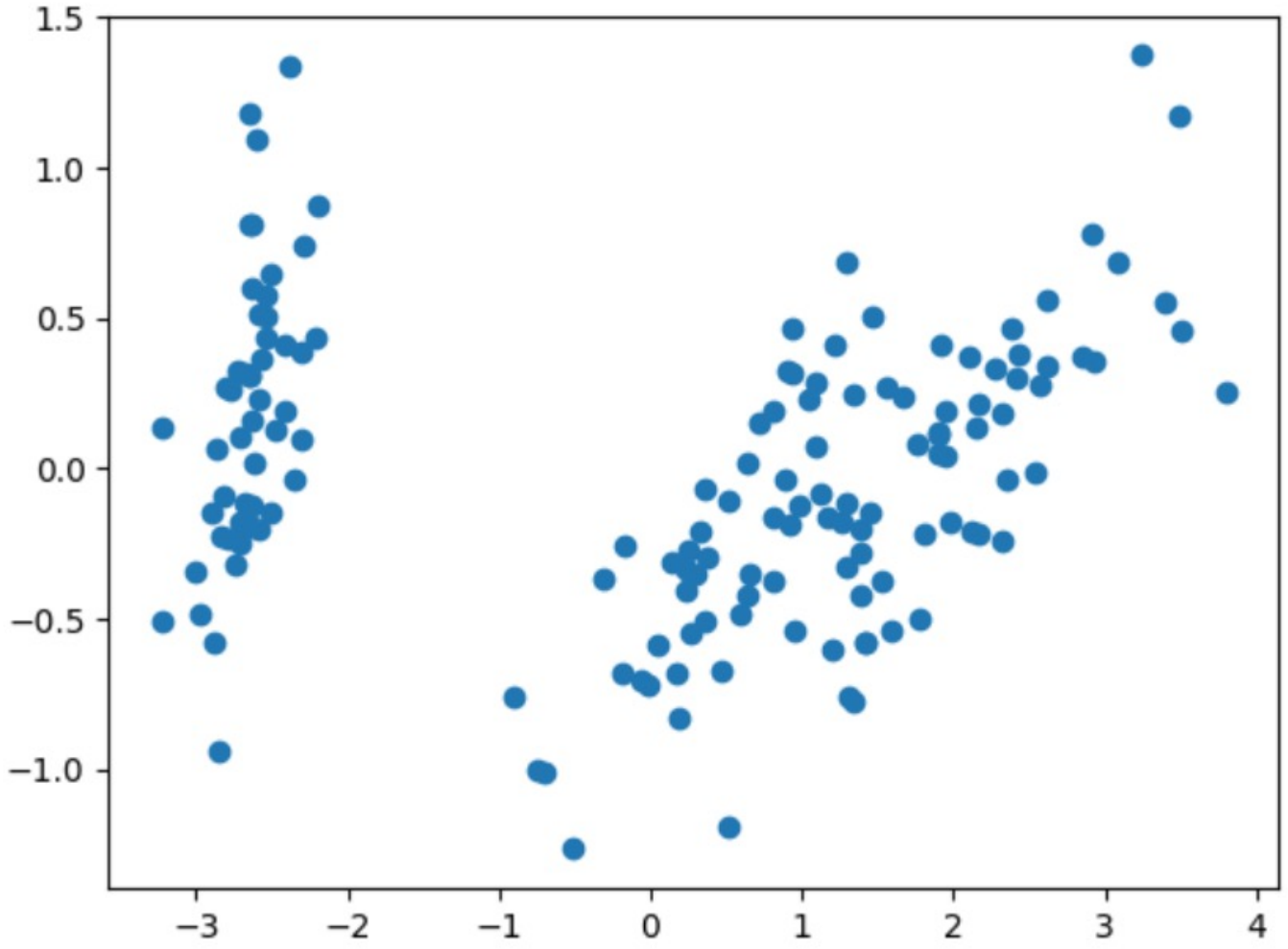
```
array([[-2.68412563,  0.31939725],
       [-2.71414169, -0.17700123],
       [-2.88899057, -0.14494943],
       [-2.74534286, -0.31829898],
       [-2.72871654,  0.32675451],
```

特徴量が4つから新たな特徴量２つ（主成分１と主成分2）になった

ex) 1つ目のアヤメ(=ヒオウギアヤメ)は
がく片の長さ＝5.1、がく片の幅＝3.5、花びらの長さ＝1.4、花びらの幅＝0.2
→　主成分１=-2.68412563、主成分２＝0.31939725　となった

主成分分析(PCA)の結果を図示する

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



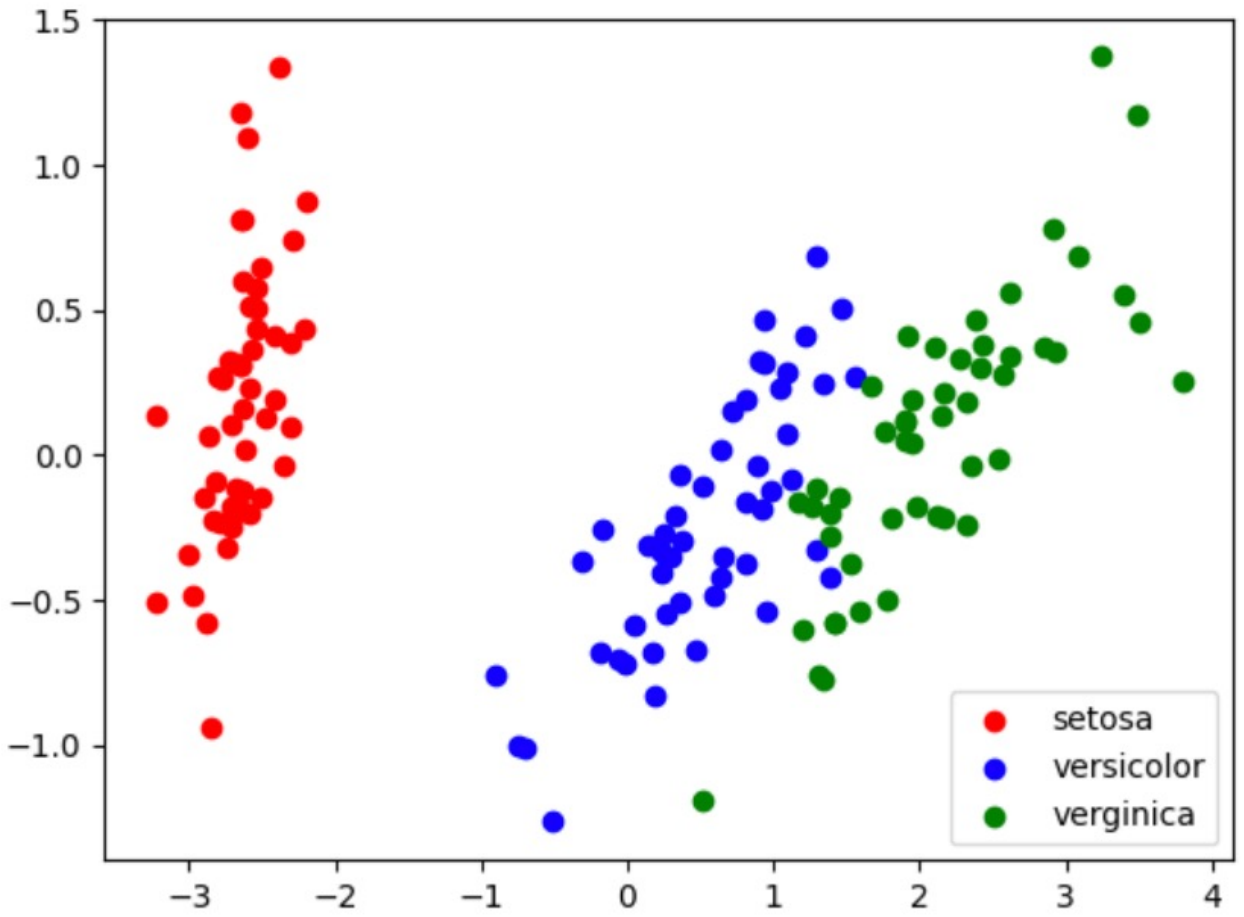
第1主成分であるresult[:,0]、
第2主成分であるresult[:,1]
を元に散布図を描画

結果のグラフは横軸が第1主成分、
縦軸が第2成分になっている

このままではどの点がどのアヤメか
分からない

アヤメの種類で色分けする

```
import matplotlib.pyplot as plt
plt.scatter(result[0:50,0],result[0:50,1],c='r',label='setosa')
plt.scatter(result[50:100,0],result[50:100,1],c='b',label='versicolor')
plt.scatter(result[100:150,0],result[100:150,1],c='g',label='verginica')
plt.legend()
plt.show()
```

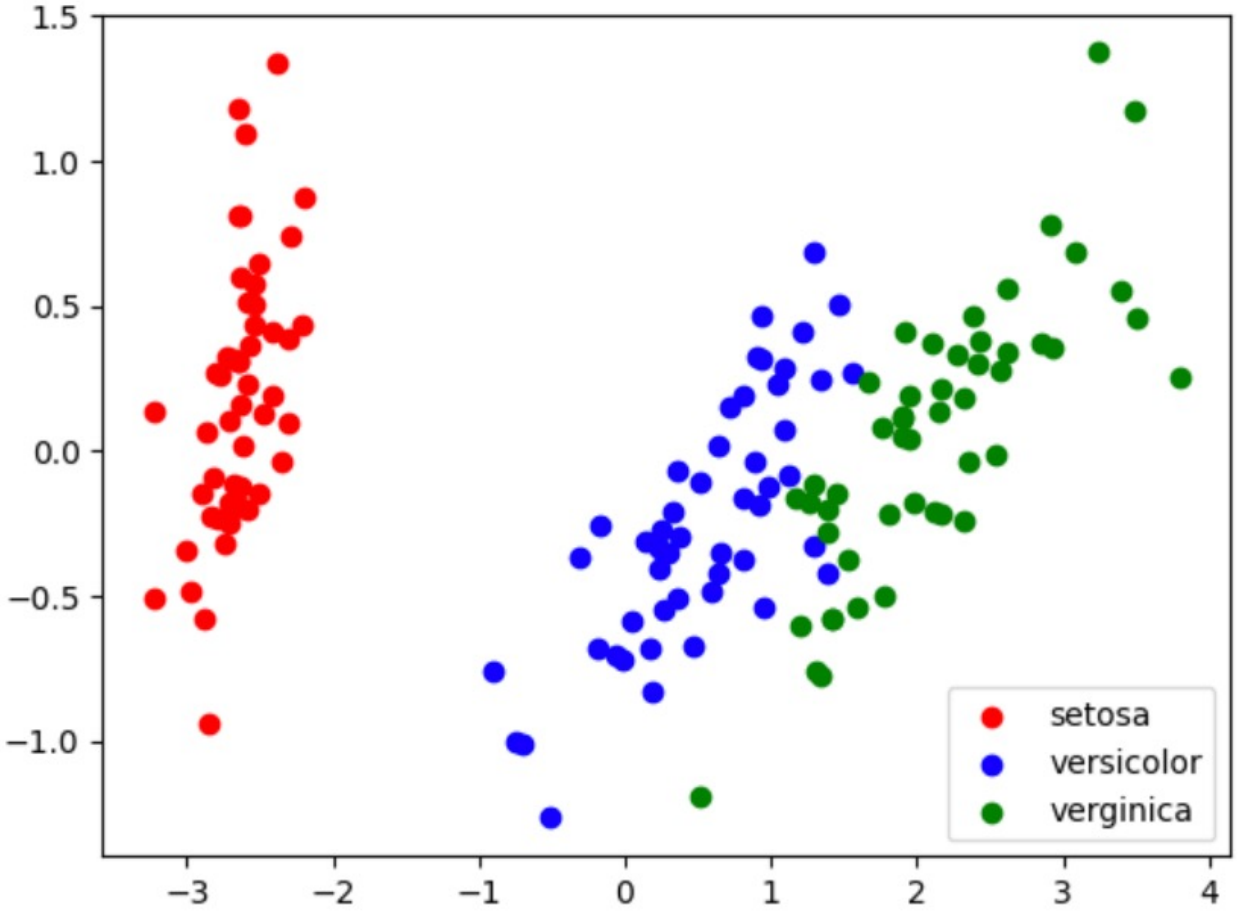


result[0:50,0],result[0:50,1]はそれぞれ
1から50行目までの第1主成分と第2主成分

1から50行目のヒオウギアヤメを赤
51から100行目のブルーフラッグを青
101から150行目のバージニカを緑
にしている

c=~~は点の色、label=~~は点の名前を指定
labelを追加した場合はplt.legend()を追加

主成分分析の結果



4つの特徴量から新たな2つの主成分で3つのアヤメが大体分類されることが分かった
第2主成分はいずれのアヤメもばらついているが、第1主成分の値で3つは大まかに分類出来る
（第1主成分の値：ヒオウギアヤメ<ブルーフラッグ<バージニカ）

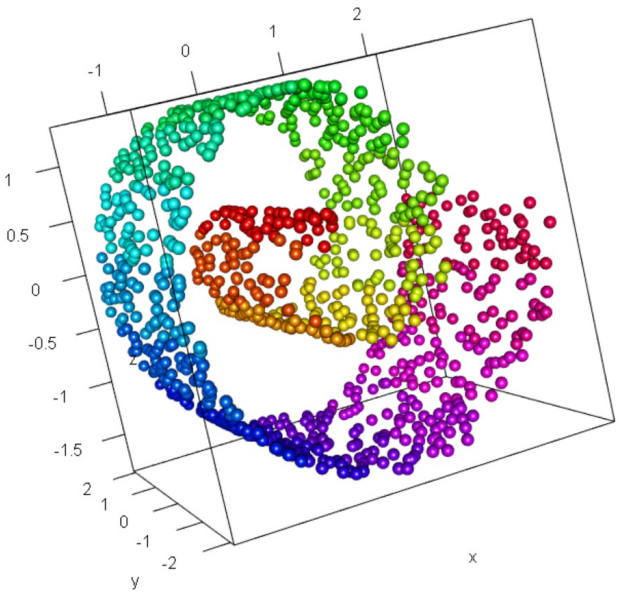
他の次元削減の実施

MDS(多次元尺度構成法)
：データ同士の近さを元に、近いもの同士は近くに配置し、
遠いものは遠くに配置する手法

t-SNE
：非常に多次元のデータも2、３次元に落とし込める方法
近くの点との関係をなるべく維持するように変換する
複雑なデータだとPCAよりも高精度に次元削減を行えることが多い

UMAP（紹介のみ)
：tSNEよりもさらに実行速度が速い次元削減手法
近年ゲノム解析でよく使用されている

PCAはこのような
非線形なデータが苦手　→
図は有名なswiss roll(ロールケーキ)というデータ



他の次元削減の実施

やり方は大きくは変わらない
model=MDS()とする

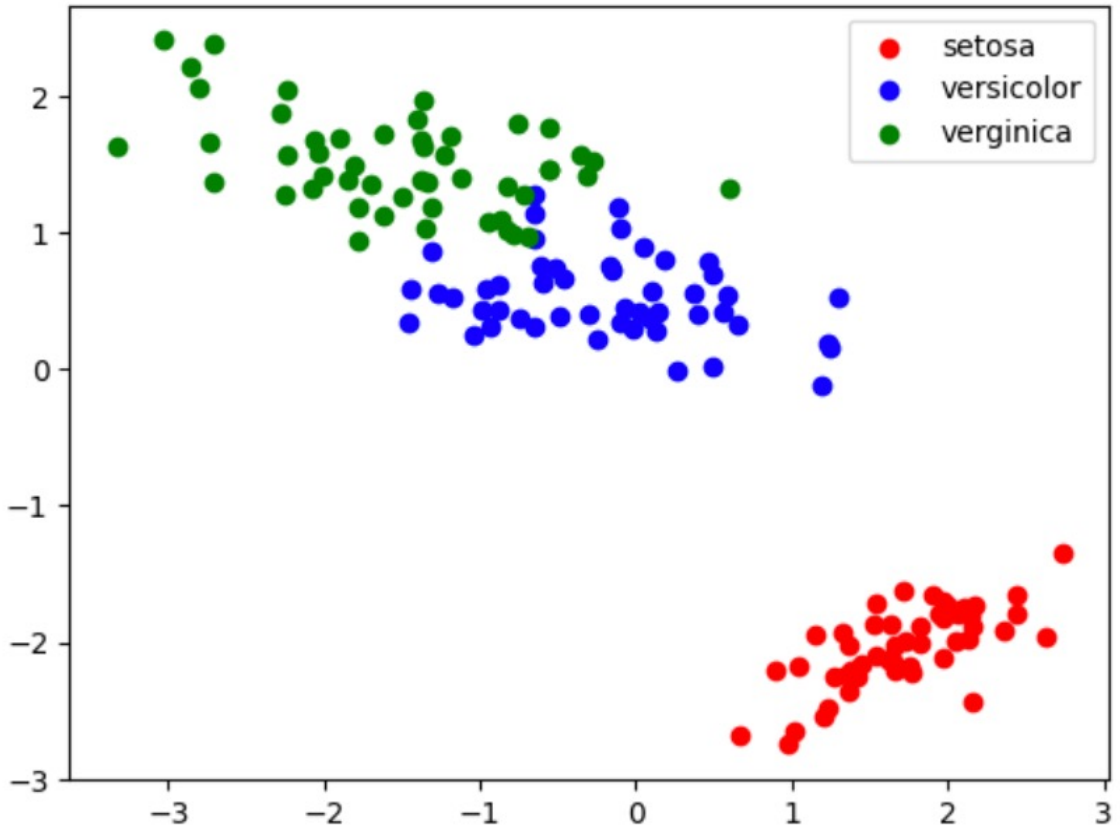
```
from sklearn.manifold import MDS

model = MDS(n_components=2)
result = model.fit_transform(iris.data)
```

```
/usr/local/lib/python3.10/dist-packages/sklear
  warnings.warn(
```

アルゴリズムが違うだけで、目的はPCAと一緒
(特徴量4つの(150,4)を特徴量2つの(150,2)にしている)
なので、同じように図示(先ほどのコードを再利用)

```
import matplotlib.pyplot as plt
plt.scatter(result[0:50,0],result[0:50,1],c='r',label='setosa')
plt.scatter(result[50:100,0],result[50:100,1],c='b',label='versicolor')
plt.scatter(result[100:150,0],result[100:150,1],c='g',label='verginica')
plt.legend()
plt.show()
```

MDSを実行する



結果は毎回違います

他の次元削減の実施

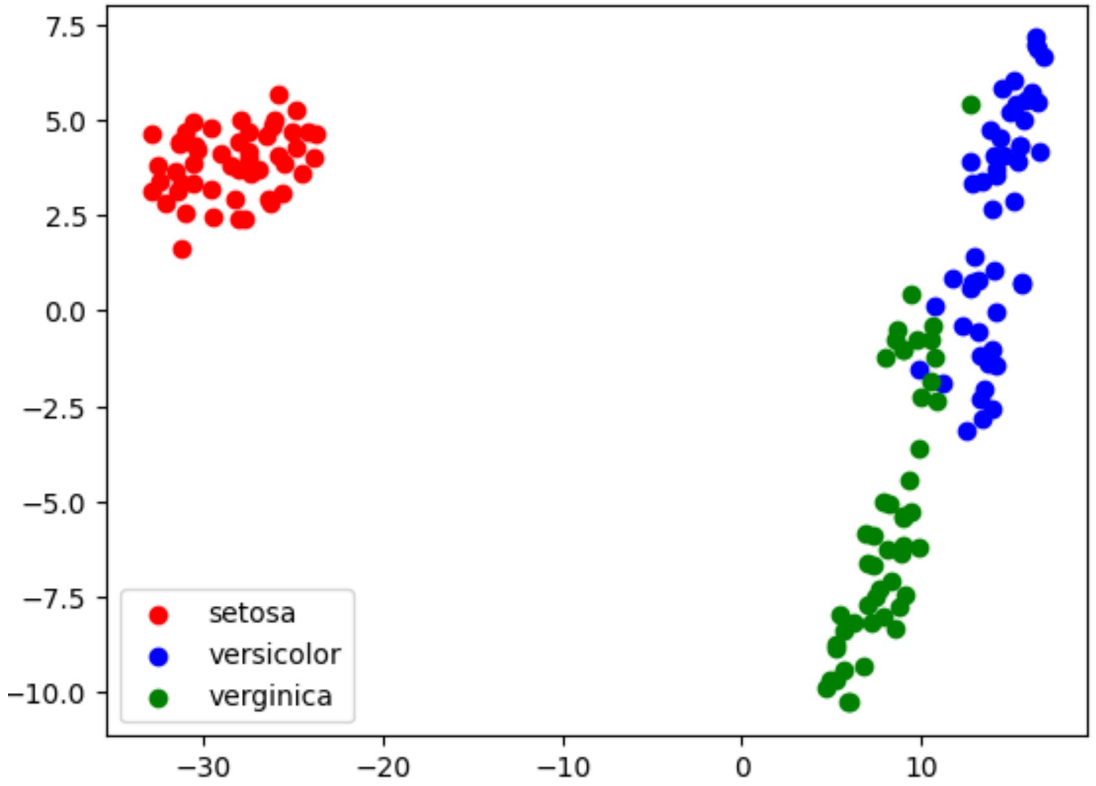
```
from sklearn.manifold import TSNE
model = TSNE(n_components=2)
result = model.fit_transform(iris.data)
```

t-SNEを実行する

やり方は大きくは変わらない
モデル名をTSNE()とする

作図は同様

```
import matplotlib.pyplot as plt
plt.scatter(result[0:50,0],result[0:50,1],c='r',label='setosa')
plt.scatter(result[50:100,0],result[50:100,1],c='b',label='versicolor')
plt.scatter(result[100:150,0],result[100:150,1],c='g',label='verginica')
plt.legend()
plt.show()
```



他の次元削減の実施

```
!pip install umap-learn
from umap import UMAP
model = UMAP(n_neighbors=15)
result = model.fit_transform(iris.data)
```

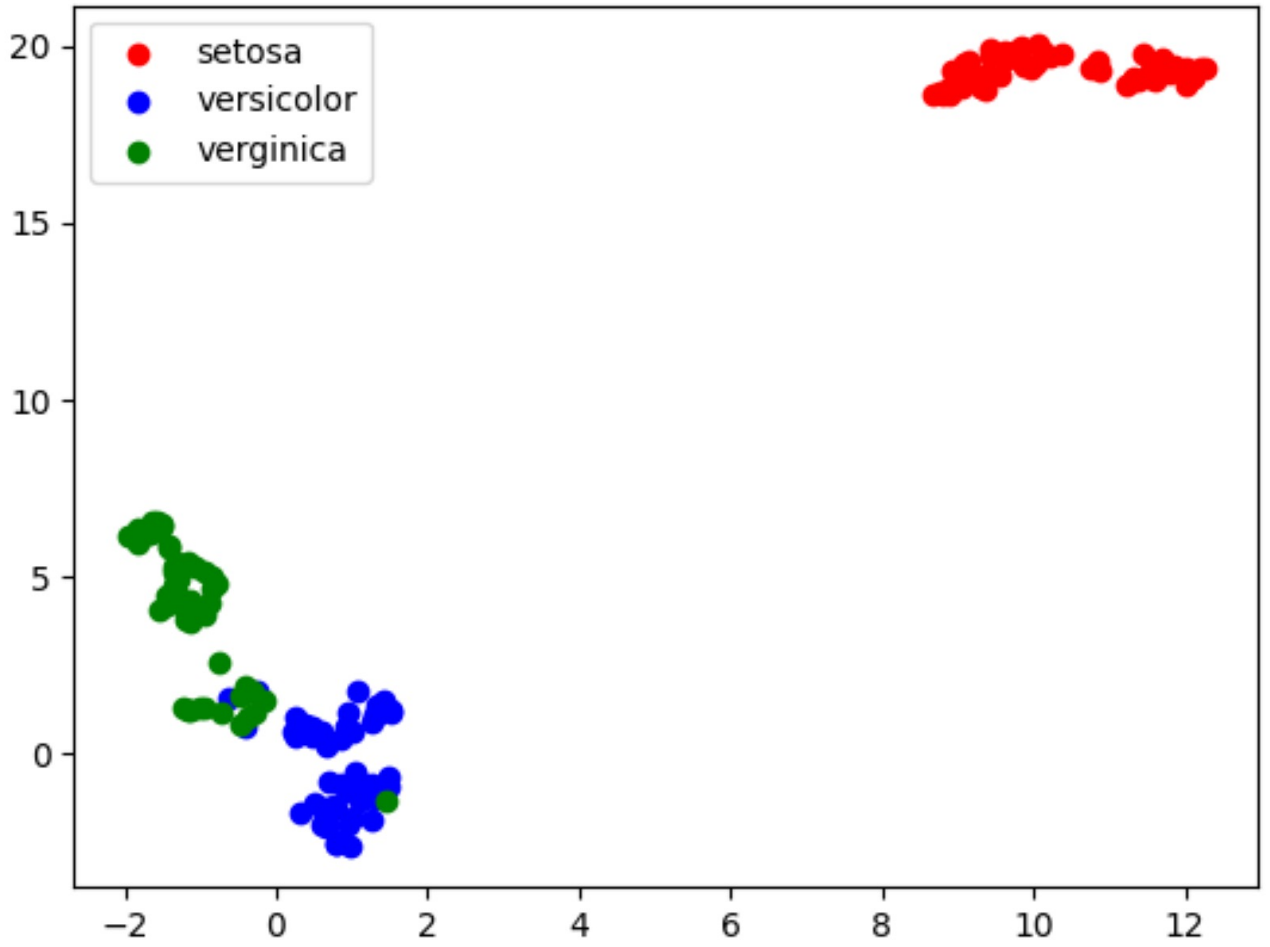
```
Collecting umap-learn
  Downloading umap-learn-0.5.5.tar.gz (90 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 90.
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.17 in /us
```

やり方は大きくは変わらない
モデル名をUMAP()とする

n_componentsのような指定が無くても
2つの特徴量に削減される

UMAPを実行する

(UMAPはumap-learnという
ライブラリのインストールが必要)



mnistのデータでやってみよう

mnistの読み込み

```
from keras.datasets import mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
```

教師なし機械学習なので、x_trainしか使わない。
x_trainのデータ(60000,28,28)を(60000,784)に変換

```
x_train.shape
```

```
(60000, 28, 28)
```

```
x_train = x_train.reshape(60000,784)
x_train.shape
```

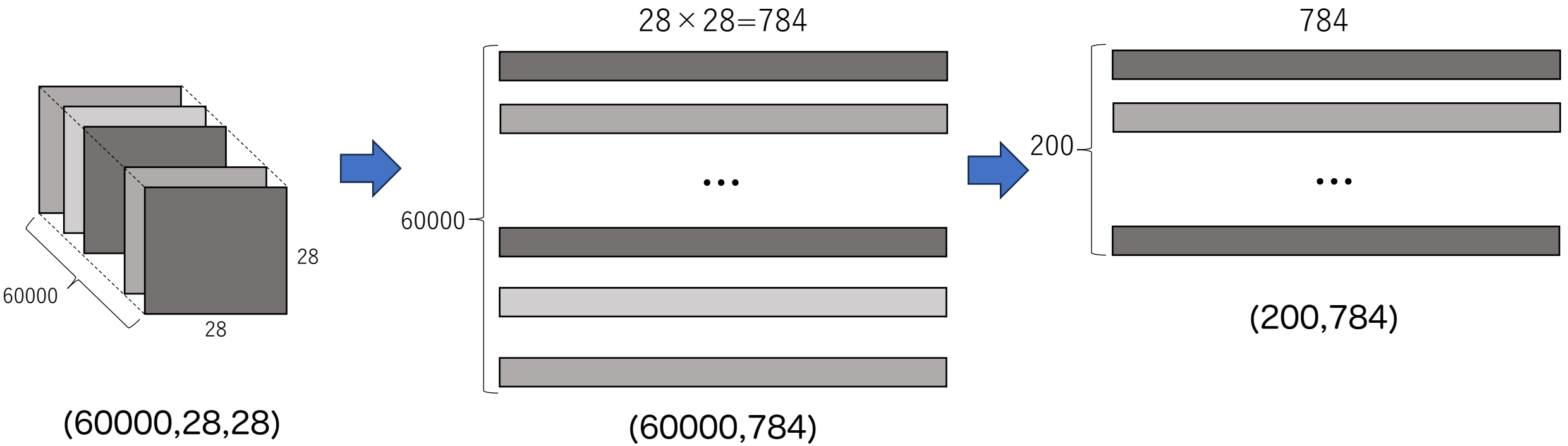
```
(60000, 784)
```

mnistのデータでやってみよう

```
x200 = x_train[0:200]
x200.shape
```

```
(200, 784)
```

data(60000,784)の先頭200個を取り出す



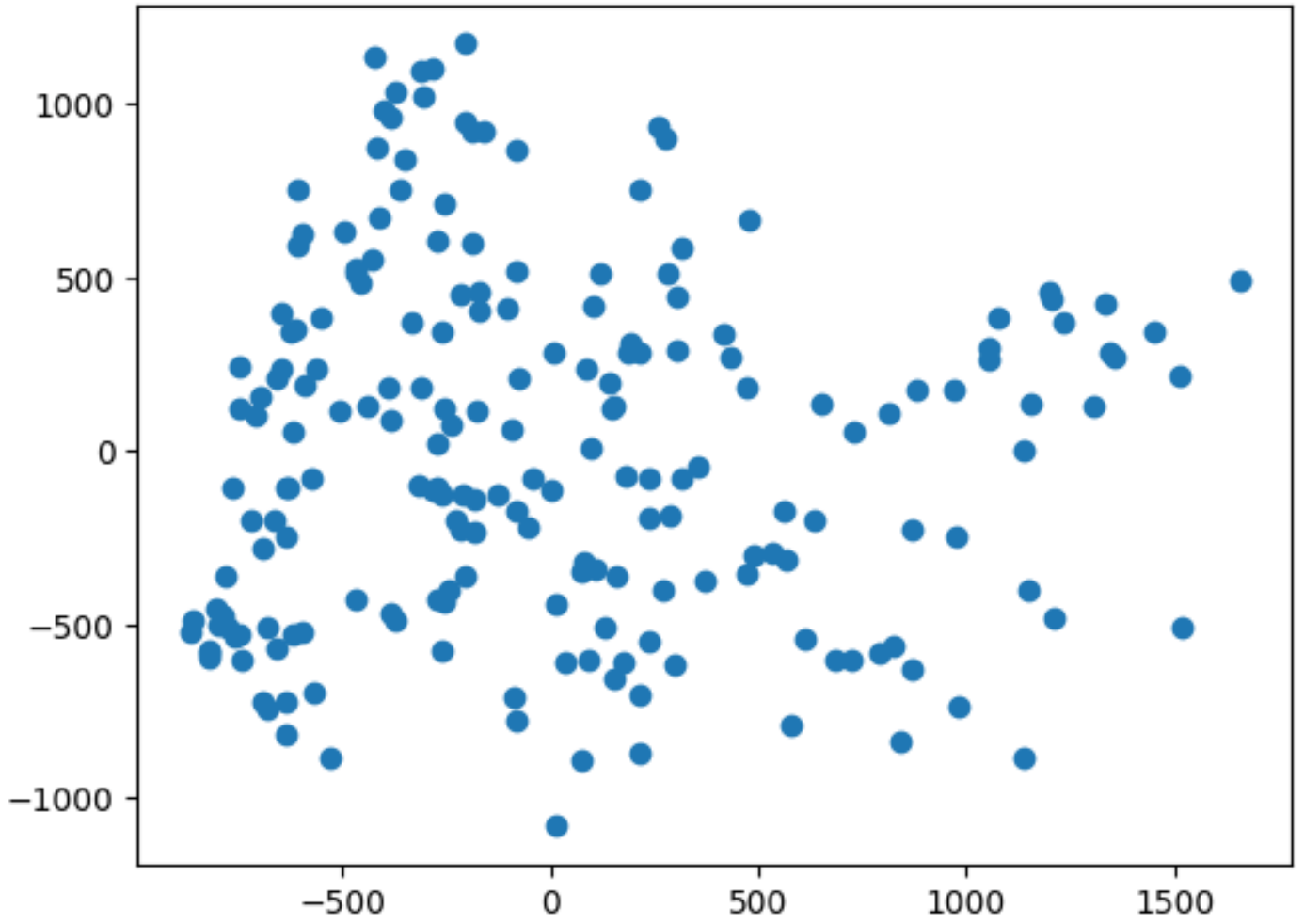
mnistのデータでPCA

```
from sklearn.decomposition import PCA
model = PCA(n_components=2)
result = model.fit_transform(x200)
```

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

784次元を2次元に次元削減
した先頭200個のデータ

このresultもラベルで色分けしたい



mnistのデータでPCA

60000個のラベル

```
[23] y_train.shape

     (60000,)
```

先頭200個のラベル

```
[25] y200 = y_train[0:200]
     y200.shape

     (200,)
```

```
y200

array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4, 3, 5, 3, 6, 1, 7, 2, 8, 6, 9, 4, 0,
       9, 1, 1, 2, 4, 3, 2, 7, 3, 8, 6, 9, 0, 5, 6, 0, 7, 6, 1, 8, 7, 9,
       3, 9, 8, 5, 9, 3, 3, 0, 7, 4, 9, 8, 0, 9, 4, 1, 4, 4, 6, 0, 4, 5,
       6, 1, 0, 0, 1, 7, 1, 6, 3, 0, 2, 1, 1, 7, 9, 0, 2, 6, 7, 8, 3, 9,
       0, 4, 6, 7, 4, 6, 8, 0, 7, 8, 3, 1, 5, 7, 1, 7, 1, 1, 6, 3, 0, 2,
       9, 3, 1, 1, 0, 4, 9, 2, 0, 0, 2, 0, 2, 7, 1, 8, 6, 4, 1, 6, 3, 4,
       5, 9, 1, 3, 3, 8, 5, 4, 7, 7, 4, 2, 8, 5, 8, 6, 7, 3, 4, 6, 1, 9,
       9, 6, 0, 3, 7, 2, 8, 2, 9, 4, 4, 6, 4, 9, 7, 0, 9, 2, 9, 5, 1, 5,
       9, 1, 2, 3, 2, 3, 5, 9, 1, 7, 6, 2, 8, 2, 2, 5, 0, 7, 4, 9, 7, 8,
       3, 2], dtype=uint8)
```

先頭200個の正解ラベル
(1つ目の画像は5, 2つ目の画像は
0, ..., 200個目の画像は2)

mnistのデータでPCA

```
y200 == 0
```

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

y200 == 0
とすると、y200(200個の数字のデータ)のうち、

0と等しければTrue,
0と等しく無ければFalse

が返ってきますnumpy配列)

”==“等しい、”>”大きい、“<“小さいなどを比較演算子と言います

numpy配列でTrueだけ抜き出す

```
import numpy as np
test = np.array([1,2,3,4,5])
test
```

```
array([1, 2, 3, 4, 5])
```

```
true_false = np.array([True,False,True,False,True])
true_false
```

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

```
test[true_false]
```

```
array([1, 3, 5])
```

test

[1 , 2 , 3 , 4 , 5]

true_false

[True,False,True,False,True]

test[true_false]

[1 , 3 , 5]

numpy配列でTrueだけ抜き出す

第１主成分を取り出してresult1に代入

```
result1 = result[:,0]
result1
```

(200,)

```
array([ 3.69042055e+02,  1.05540642e+03, -1.03942075e+02, -6.59462988e+02,
       -4.25982148e+02,  1.89542498e+02, -6.19111451e+02,  8.69051072e+02,
       -7.81486729e+02, -4.96525277e+02,  2.14912864e+02, -6.62952615e+02,
        5.79201334e+02,  8.45672674e+01, -7.86051951e+02, -4.20579990e+02,
        9.61637449e+01, -2.60347671e+02, -8.44706128e+01, -7.46599062e+02,
        2.79621049e+02,  1.05183831e+03, -6.49147638e+02, -6.79366168e+02,
       -2.25514337e+02,  1.13727774e+03, -5.89625657e+02,  1.21077361e+03,
        4.74306669e+02, -7.49757350e+02, -2.02082715e+02,  1.31372667e+01,
       -9.39728235e+01, -6.26401328e+02,  8.82811286e+02, -6.30434967e+02,
        1.61209327e+02,  1.07877215e+03, -3.10096989e+02,  2.12301829e+02,
       -8.22747095e+02,  7.46722708e+01, -7.08739847e+02, -5.76518458e+02,
       -2.52441047e+02, -2.56990872e+02, -5.49491504e+01, -2.54094832e+02,
       -1.82181897e+02,  6.83137399e+02,  1.77733554e+02,  1.65795746e+03,
        2.73856204e+02, -7.64082193e+02, -2.72471567e+02, -7.49196019e+01,
        1.20576150e+03, -5.94516916e+02,  3.01586880e+02, -5.71117649e+02,
        1.20858698e+02, -6.21018205e+02,  5.34650464e+02,  1.35521202e+03,
        3.15181555e+02, -6.36017541e+02,  3.56570551e+02, -8.22630886e+02,
        2.71463345e+02,  1.51107526e+03, -8.22408500e+01, -6.09703652e+02,
       -8.61350490e+02, -4.38553778e+01,  2.34853270e+02,  1.15345437e+03,
        1.58225416e+01, -6.37941684e+02, -7.58247453e+02, -6.06105496e+02,
        6.34359678e+02,  1.34782609e+03,  1.14675780e+03,  1.52414744e+02,
       -1.71108658e+02,  7.82594925e+01, -2.10733628e+02, -3.03298169e+02,
        1.14076409e+03, -2.84858095e+02,  4.30953645e+02, -3.73348697e+02,
       -2.18692083e+02,  2.39554842e+02, -1.25997348e+02,  1.45170085e+03,
       -6.47448450e+02,  7.30031928e-01,  1.53117227e+02, -8.05503884e+02,
       -2.41119248e+02, -2.04083808e+02, -7.73023412e+02, -4.19030445e+02,
       -6.93372877e+02, -6.34026417e+02,  5.62965166e+02,  6.11797467e+02,
        4.72527320e+02, -2.74512647e+02,  1.04529835e+02,  9.69781835e+02,
       -8.64938064e+02, -5.27627766e+02,  1.20098466e+03, -3.82913980e+02,
       -7.96208568e+01,  8.67790417e+02,  8.15634557e+02,  1.33107334e+03,
        7.31228974e+02,  1.52049829e+03, -2.61748299e+02, -3.98108311e+02,
       -7.98860883e+02,  9.16563623e+01,  3.18126470e+02, -4.54235672e+02,
       -6.79519826e+02,  1.07304194e+02,  3.43201289e+01, -3.52585840e+02,
       -3.81869870e+02, -6.58745892e+02, -2.73636151e+02,  3.00717138e+02,
       -8.92949178e+01, -3.74521322e+02,  8.26939687e+02, -4.12275926e+02,
       -6.34916494e+02, -6.14360344e+02, -4.66612472e+02, -1.80261952e+02,
       -2.41571663e+02, -5.63835655e+02,  1.32646007e+02, -1.77969718e+02,
       -3.90216203e+02,  7.21761173e+02, -3.83081576e+02,  1.92104301e+02,
       -6.88977736e+02, -6.99920053e+02, -8.02039647e+01,  1.48769675e+02,
        6.48964056e+02,  4.90262299e+02, -1.71213525e+02,  2.86262357e+02,
       -2.82424095e+02,  9.82138603e+02, -2.62980055e+02, -5.53574260e+02,
        4.76171090e+02,  1.80944468e+02, -4.70555093e+02, -1.88960760e+02,
       -4.68920842e+02,  1.30461881e+03, -1.86366459e+02,  5.66553380e+02,
       -1.58442988e+02, -7.19908349e+02, -8.18741266e+02, -3.14390057e+02,
       -4.40518816e+02, -7.40423388e+02,  3.05827516e+02,  9.75502063e+02,
       -5.94876186e+02,  4.14736031e+02,  7.90704407e+02, -3.59793351e+02,
       -7.46093407e+02, -2.05071109e+02, -2.15388126e+02,  8.40038901e+02,
        1.40595638e+02,  2.34872660e+02,  2.15677838e+02, -5.09724869e+02,
        1.23196281e+03, -3.13103346e+02,  2.12032797e+02, -3.32520611e+02,
        2.60904311e+02, -2.69951054e+02,  7.27878343e+01,  1.03893916e+01])
```

200個のうち、正解ラベルが0のものがTrue

```
y200 == 0
```

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

result1のうち、正解ラベルが0のものだけ抜き出す

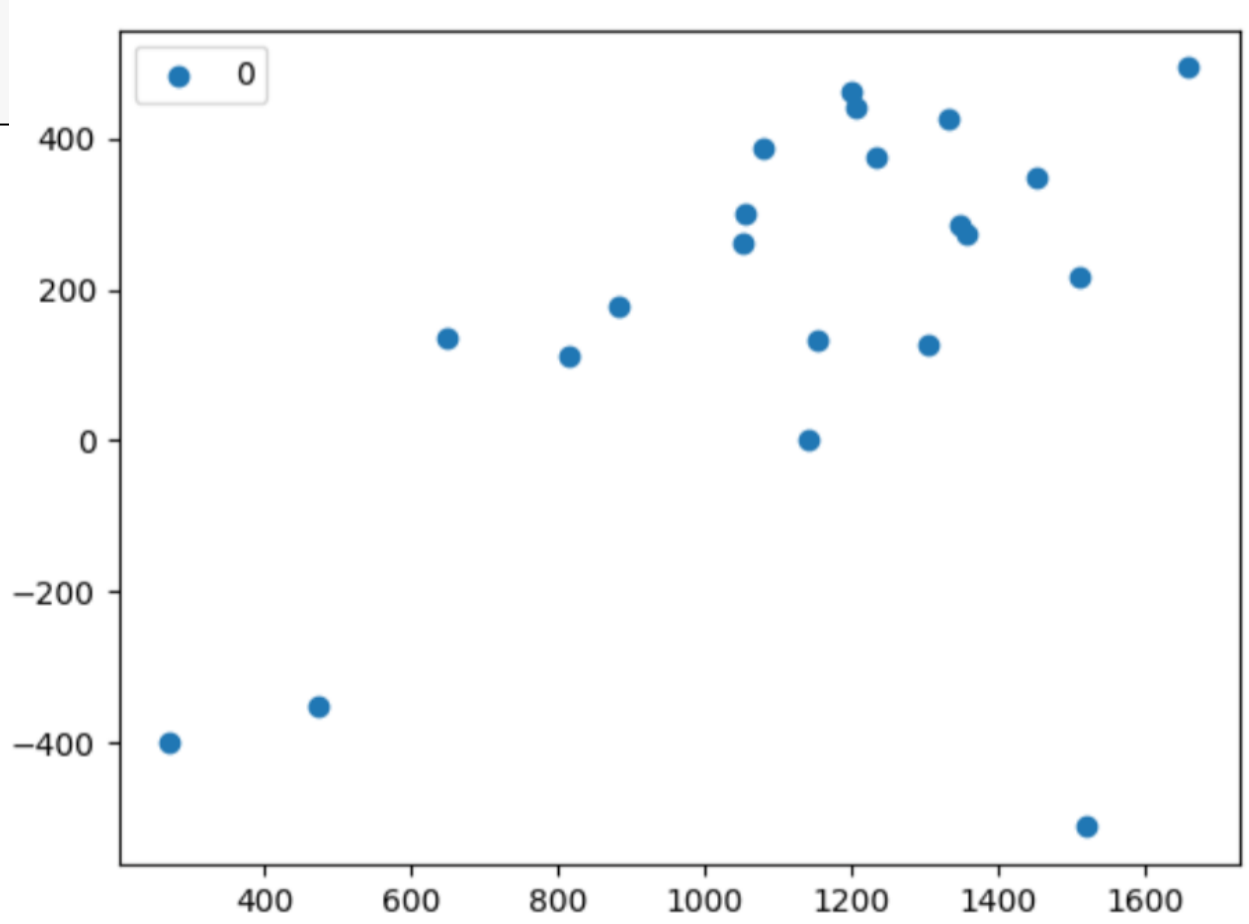
```
result1[y200 == 0]
```

```
array([1055.4064223 , 1051.83830511,  882.81128644, 1078.77215007,
       1657.95746338, 1205.76150277, 1355.21202486,  271.46334478,
       1511.07525658, 1153.45436922, 1347.82609121, 1140.76408833,
       1451.70084762,  472.52732048, 1200.98466365,  815.63455652,
       1331.07334107, 1520.49828785,  648.96405579, 1304.61880826,
       1231.96280974])
```

numpy配列でTrueだけ抜き出す

```
result2 = result[:,1]
```

```
plt.scatter(result1[y200 == 0],result2[y200 == 0],label='0')
plt.legend()
plt.show()
```



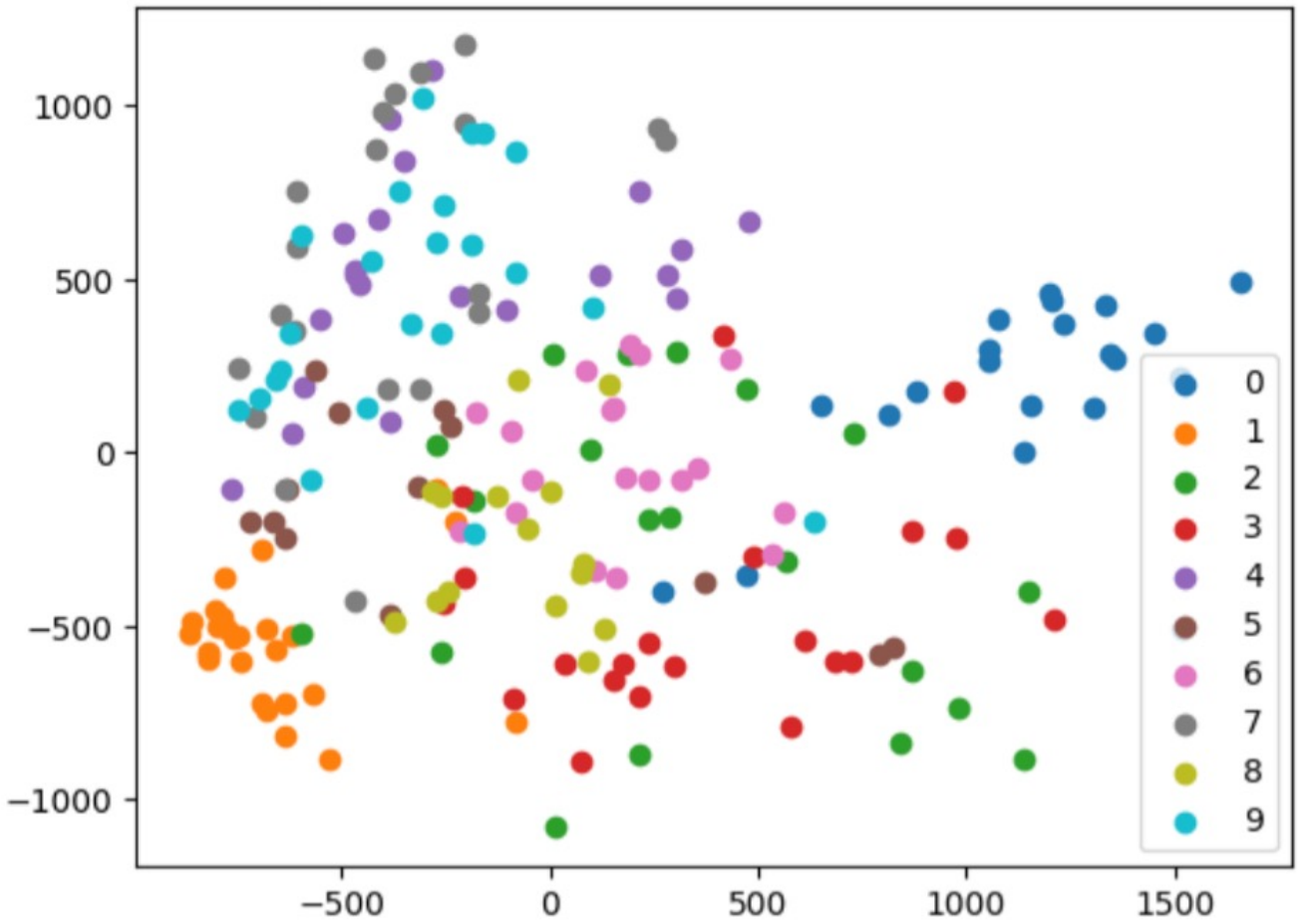
第2主成分も取り出してresult2に代入

先頭200個のうち、ラベルが0の点
のみをプロット
(label=“ラベル名”でplt.legend()とすると、
点に名前(ラベル名)をつけられる)

これをラベル0~9で繰り返して
1つの図で表示する

mnistでPCAの結果

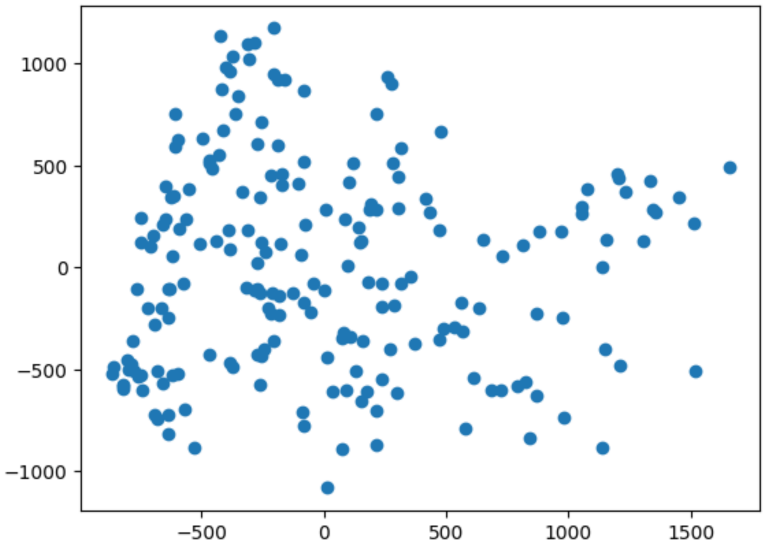
```
for i in range(10):
  plt.scatter(result1[y200 == i],result2[y200 == i],label=i)
plt.legend()
plt.show()
```



for文でrange(10)として、0から9をiに代入

plt.scatter()のみを繰り返し処理

最後に図を表示



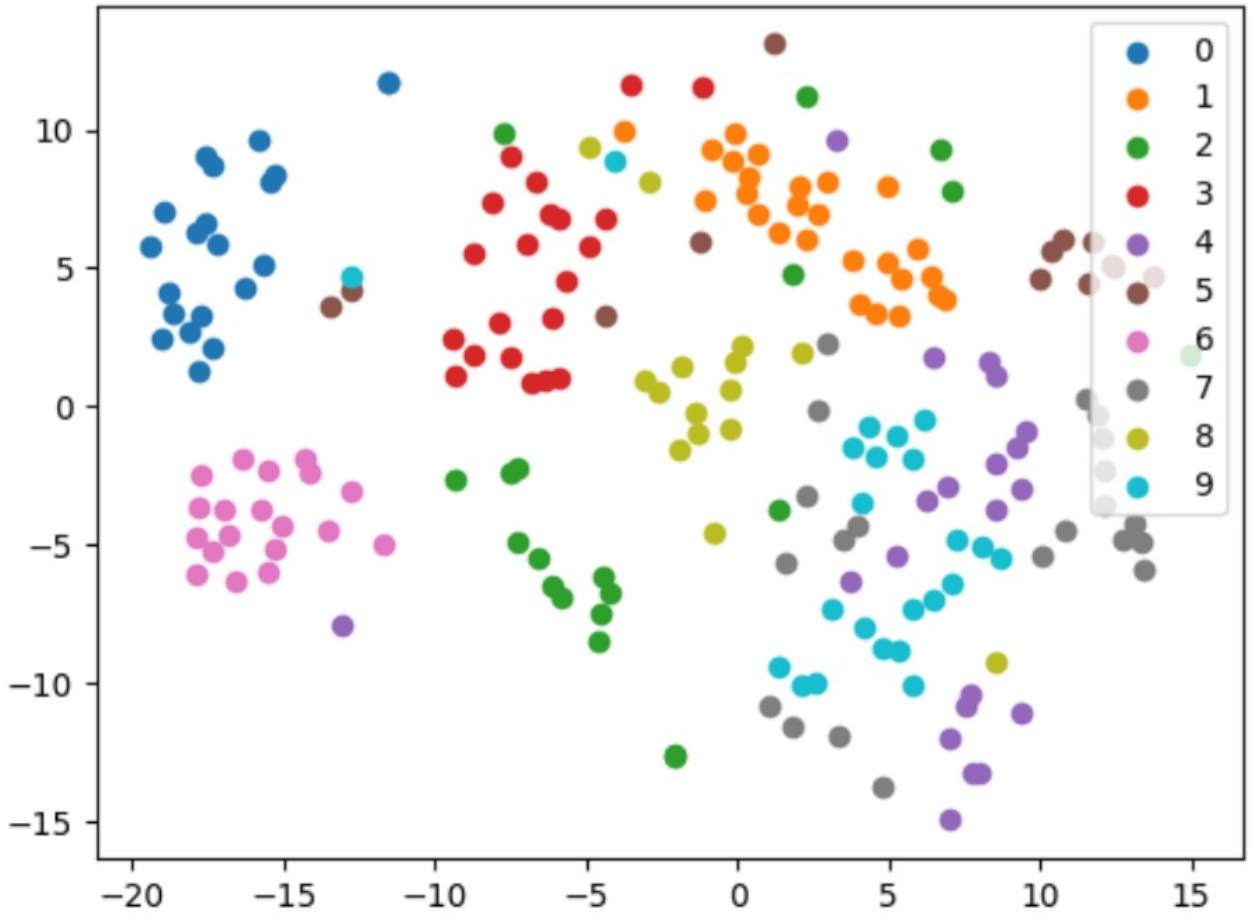
あまり綺麗に分かれていない

mnistでtSNEの結果

```
model = TSNE(n_components=2)
result = model.fit_transform(x200)
```

```
result1 = result[:,0]
result2 = result[:,1]
```

```
for i in range(10):
  plt.scatter(result1[y200 == i],result2[y200 == i],label=i)
plt.legend()
plt.show()
```



mode = TSNE()に変更して再度実行

TSNEの方がきれいに分かれそう
(結果は学習の度に変わるので皆同じ図にはなりません)

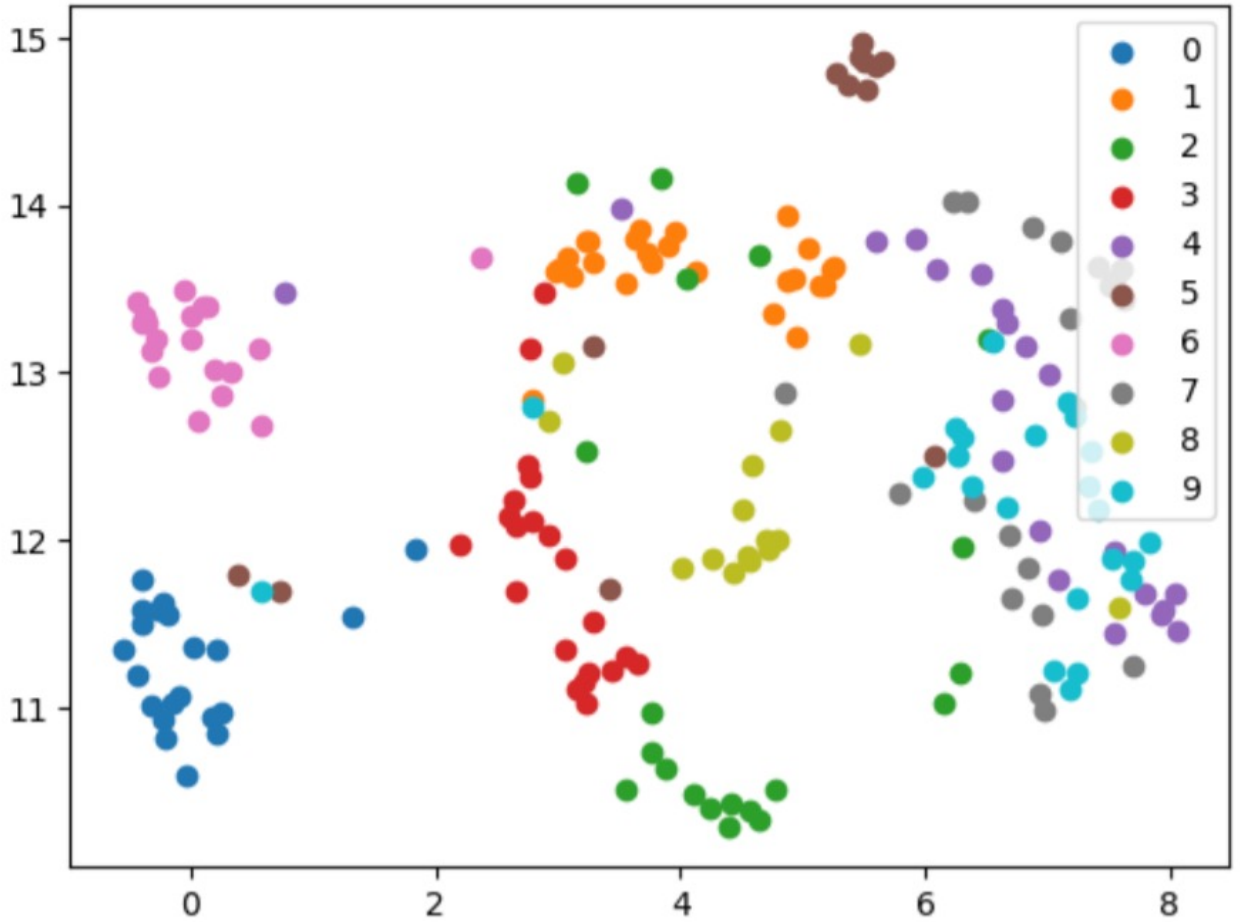
mnistでUMAPの結果

```
model = UMAP()
result = model.fit_transform(x200)
```

```
result1 = result[:,0]
result2 = result[:,1]
```

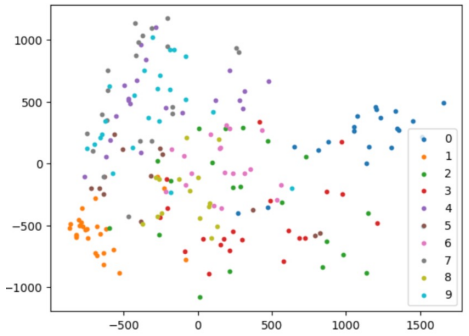
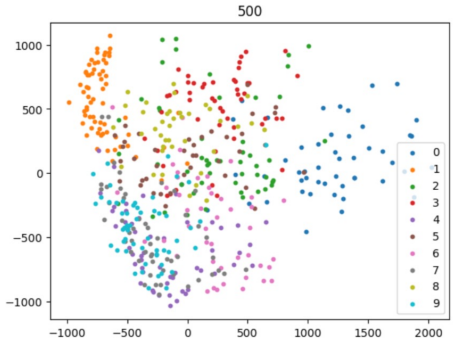
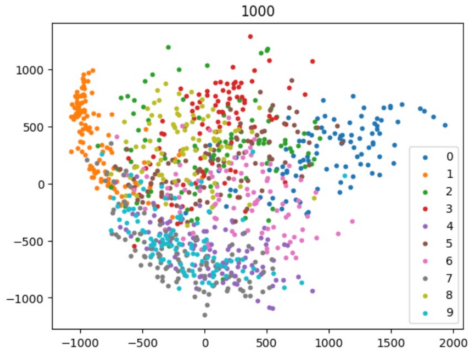
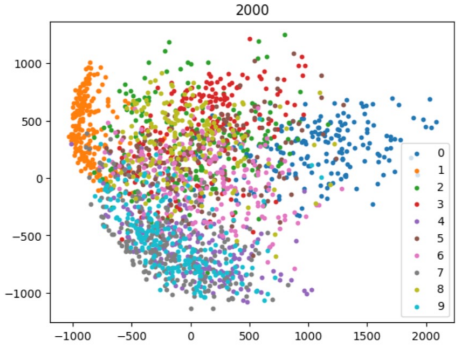
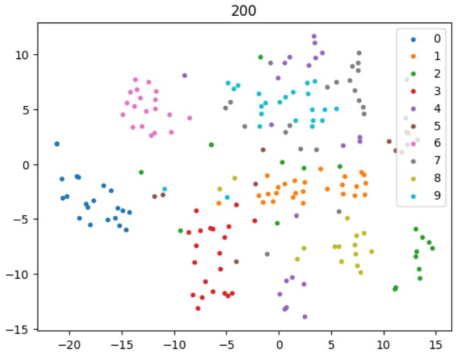
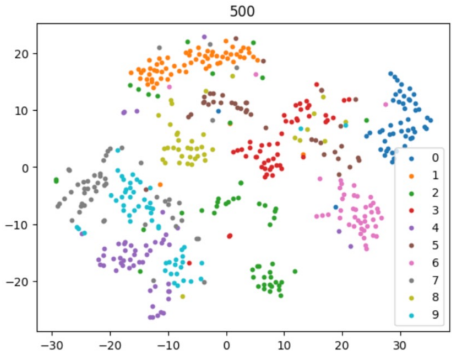
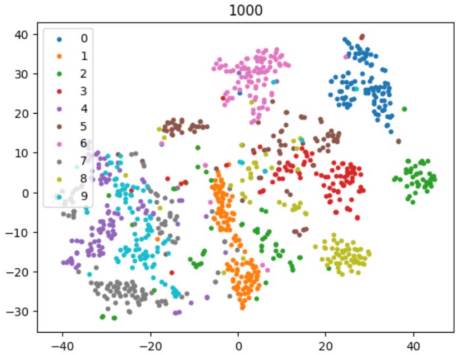
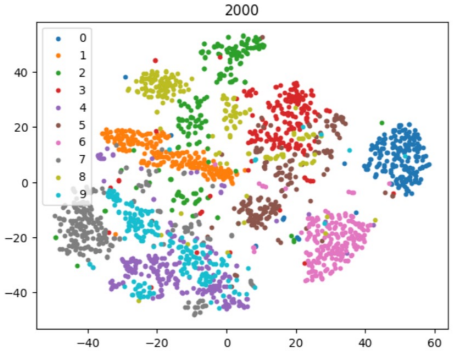
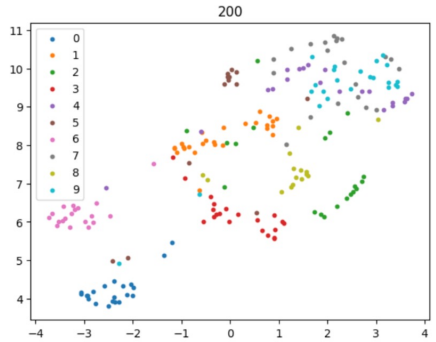
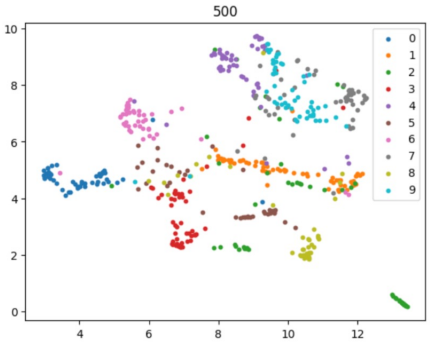
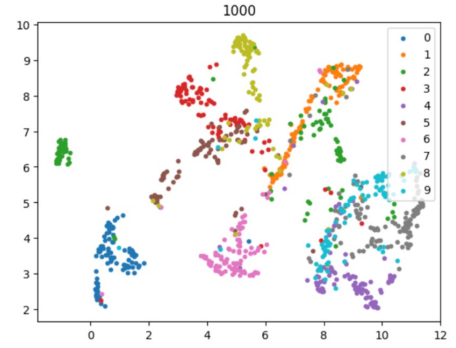
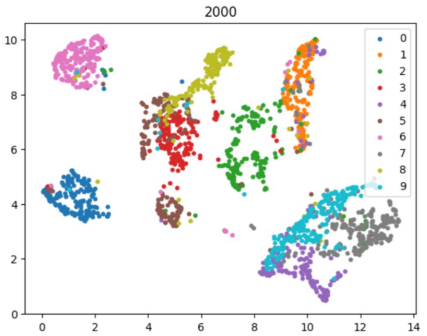
```
for i in range(10):
  plt.scatter(result1[y200 == i],result2[y200 == i],label=i)
plt.legend()
plt.show()
```

mode = UMAP()に変更して再度実行



UMAPもPCAよりきれいに分かれそう
(結果は学習の度に変わるので皆同じ図にはなりません)

n数(使用するデータの数)を変えた時の比較

	n = 200	n = 500	n = 1000	n = 2000
PCA				
tSNE				
UMAP				

```
plt.scatter(result1[y == i],result2[y == i],label=i,s=10)
```

s=10として点のサイズを小さくしている

まとめ

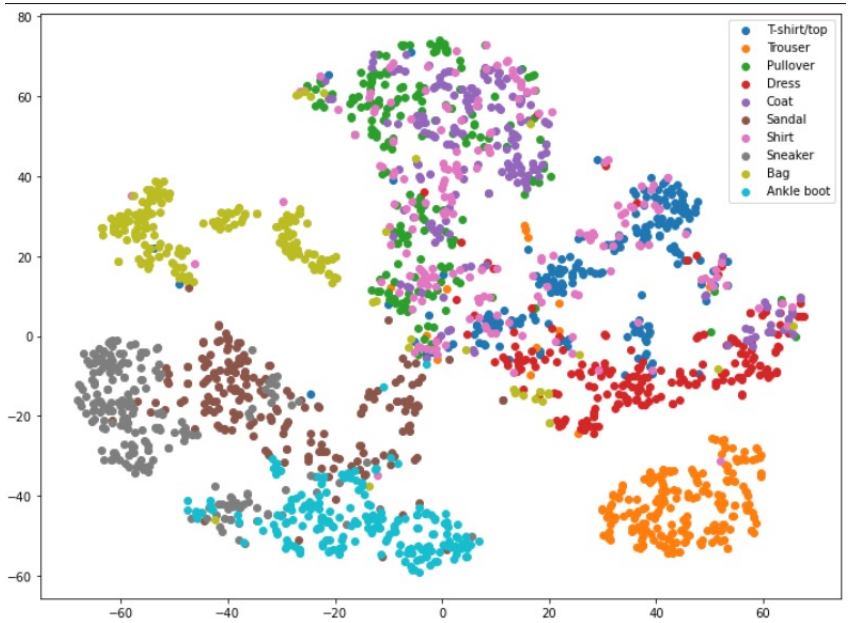
・次元削減は高次元のデータを2次元や3次元のデータに圧縮する教師なし機械学習の手法
　(ここでの次元は2次元配列などのデータの構造を指す次元ではなく、特徴量の数)

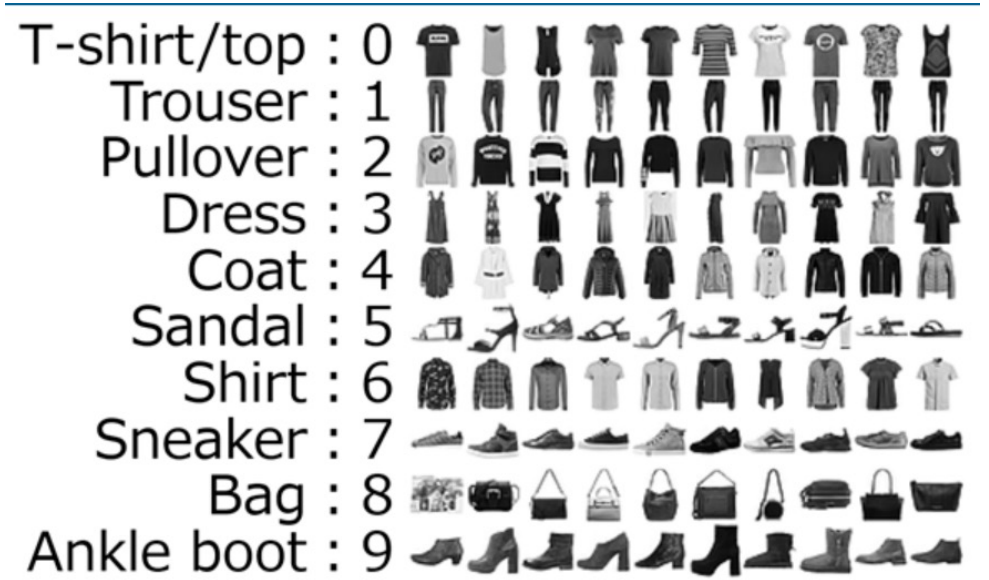
・今回の高次元の次元削減(784次元→2次元)ではtSNEやUMAPの方がうまく分類できていることが分かりました
　(画像のデータかつ1次元配列に変換(28,28)→(784,)しているにもかかわらず)

・tSNEやUMAPは可視化によく使われる手法であり、主成分分析より優れた解析手法というわけではありません

課題

fashionMNISTの学習用のデータ(x_train)を使って教師無し学習をしてください
最初の2000個のデータを使用してください
極力きれいに分類された画像を提出して下さい
ラベルは0~9でいいです
tSNEかUMAPで実施してください(皆同じ画像にはなりません)





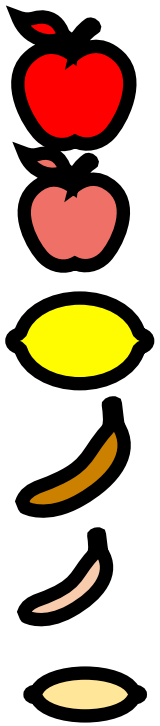
今回は、作成した図を保存してwebclassに提出してください
notebook上の画像を右クリックでダウンロードできます。
kadai13_学籍番号_名前.pngとすること(2/8 23:59まで)

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

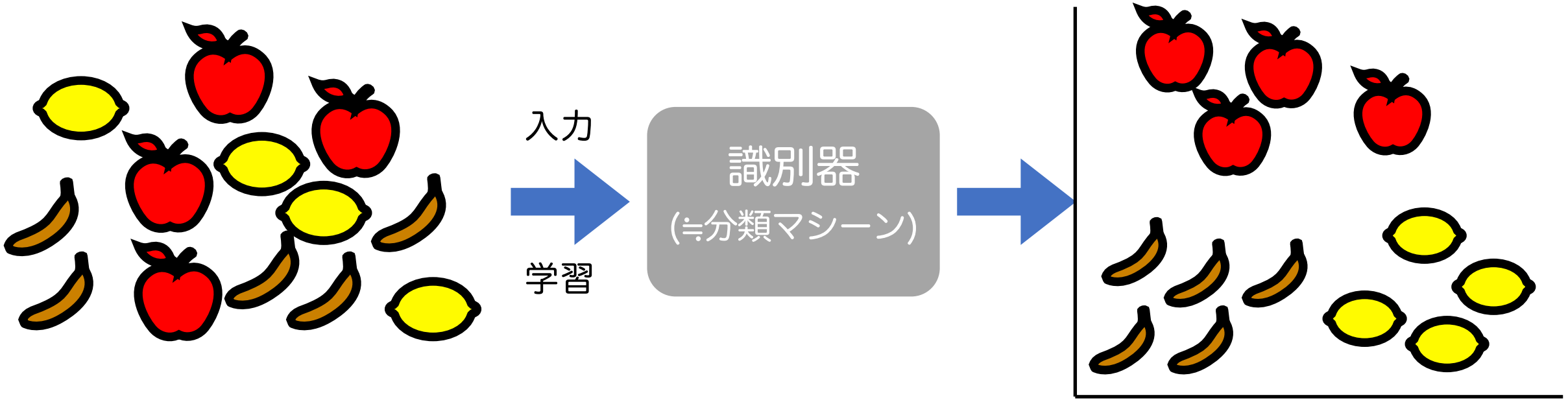
学習

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

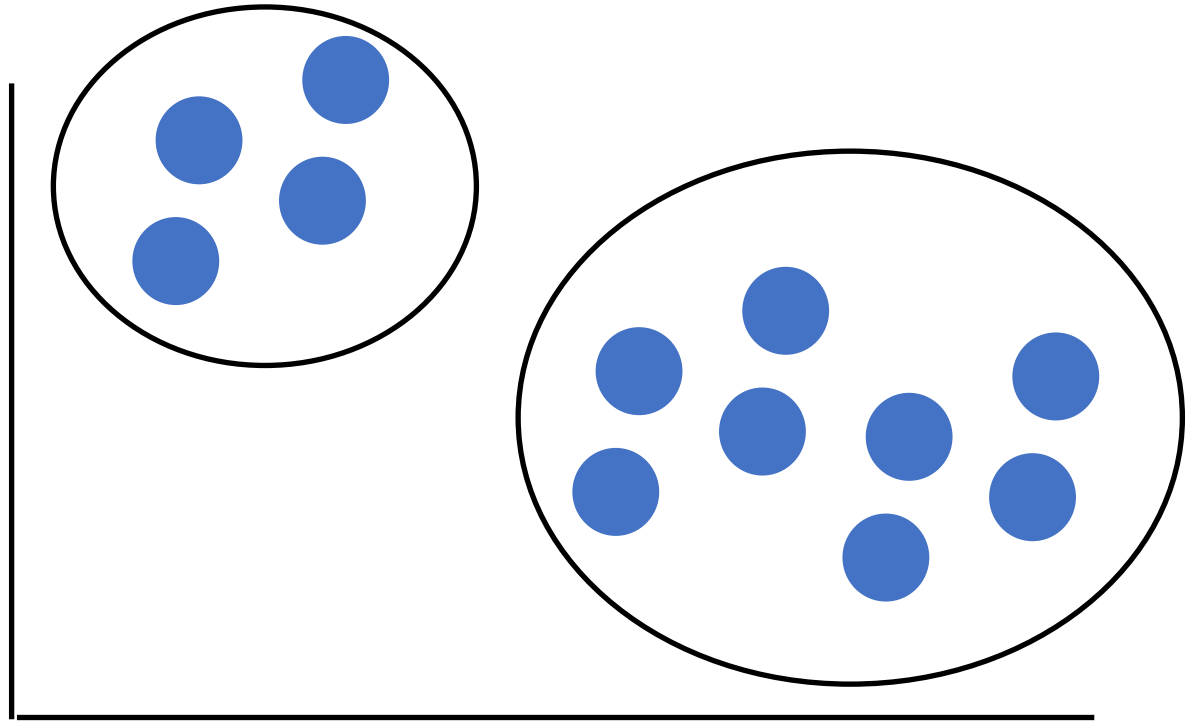
今回の教師なし機械学習（クラスタリング）



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

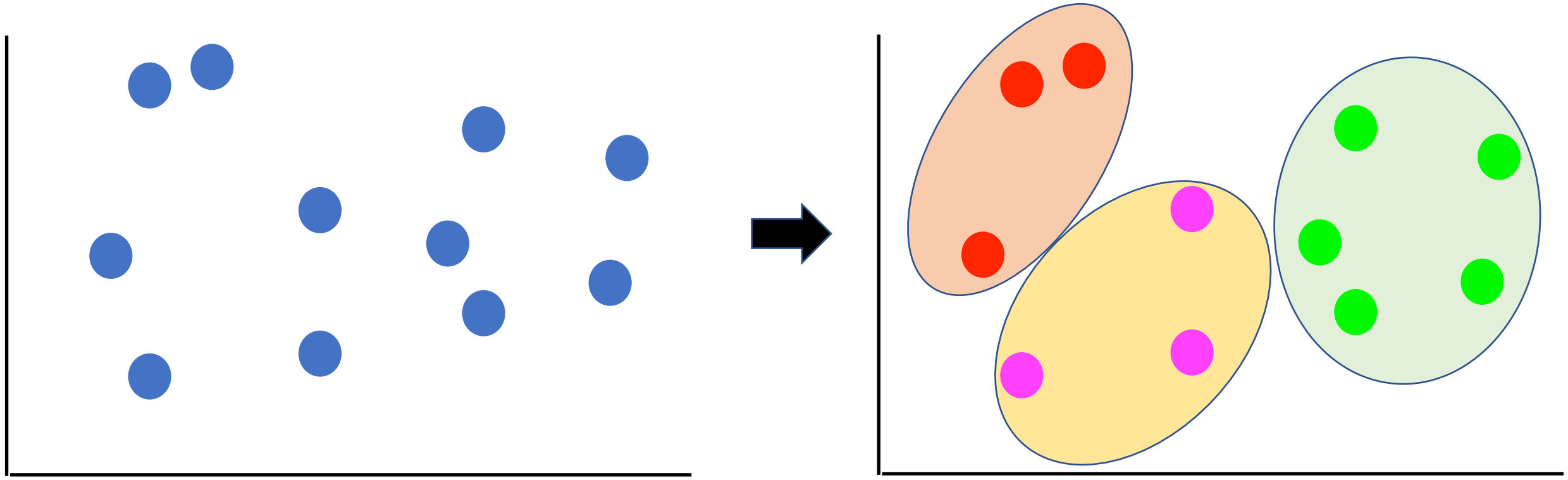


どのような法則でクラスタリングを行うか

k-means 法

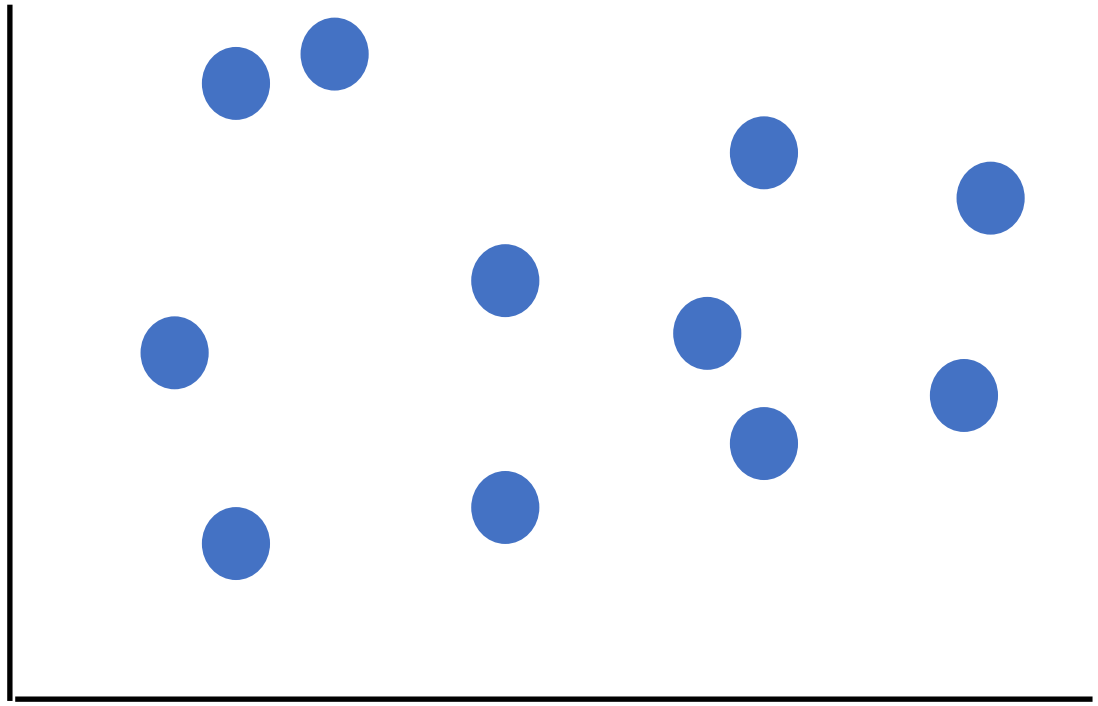
クラスタリングでも最も基本的なアルゴリズム

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

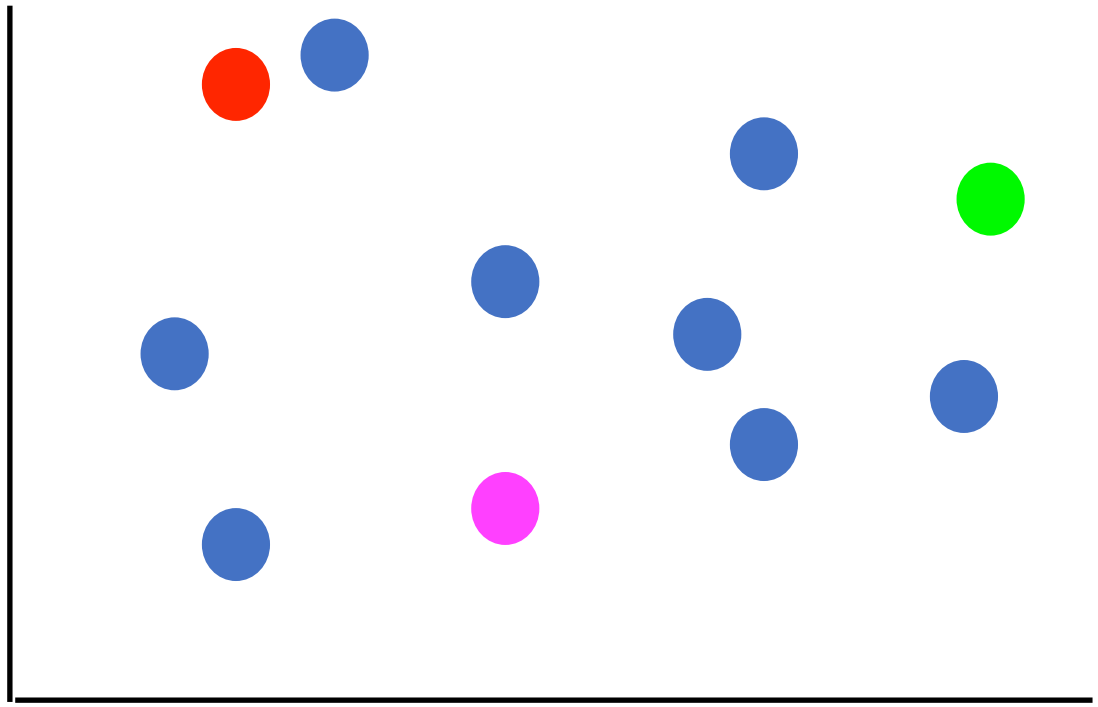
①分けるクラスタ数を決定する(今回は３とする)



k-means 法

①分けるクラスタ数を決定する(今回は３とする)

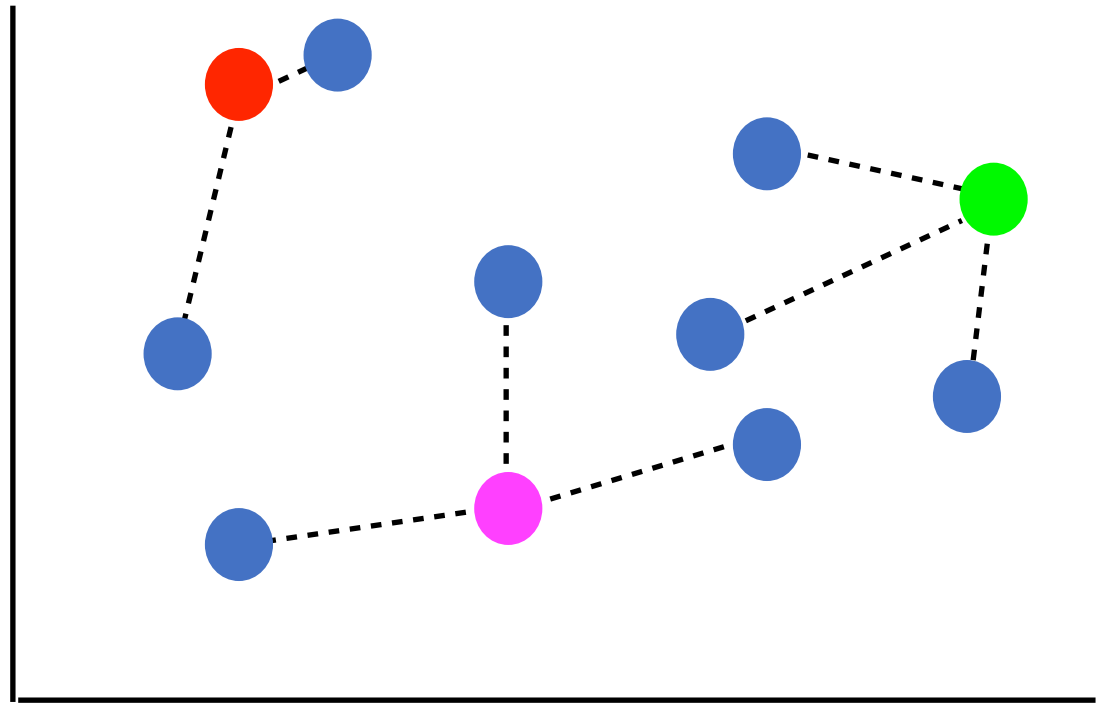
②ランダムにクラスタの個数分データを選ぶ



この３点を代表点とする

k-means 法

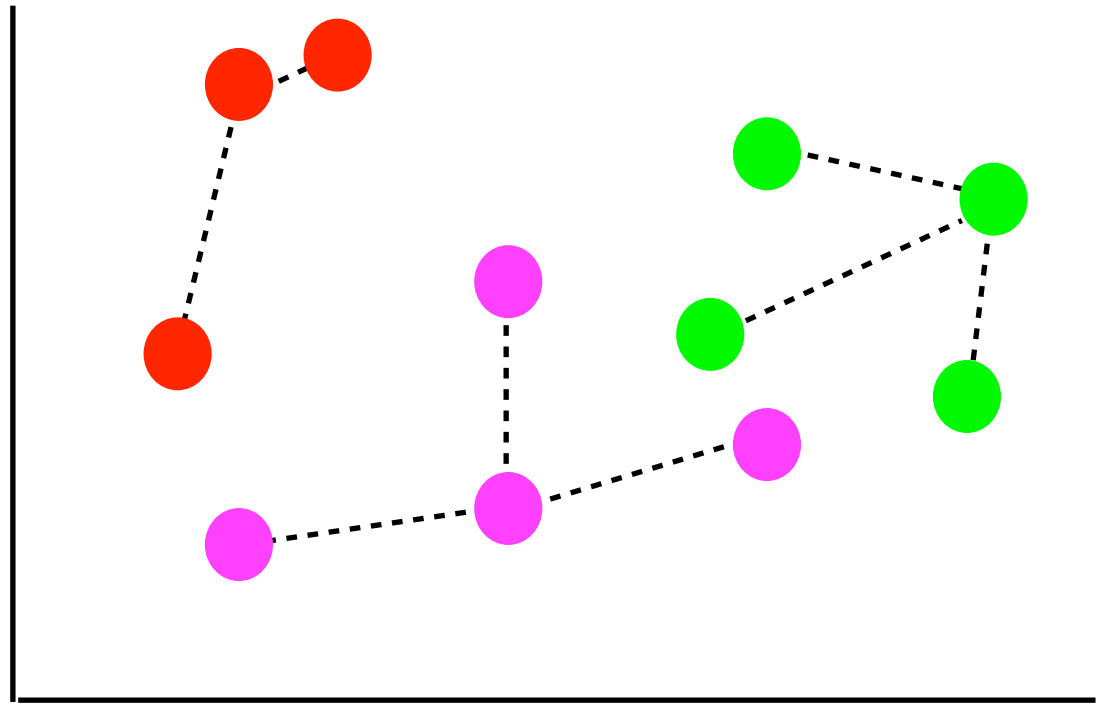
③データを各代表点の距離をもとに各クラスタに所属させる



各データは一番近い距離の代表点の所属となる

k-means 法

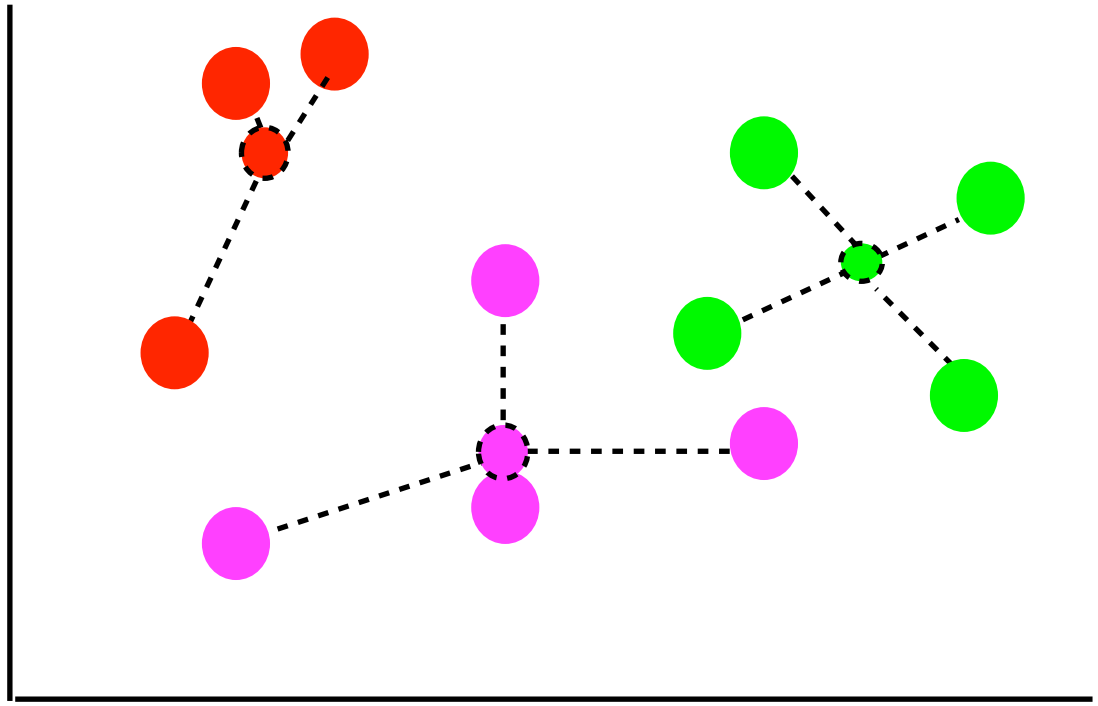
③データを各代表点の距離をもとに各クラスタに所属させる



各データは一番近い距離の代表点の所属となる

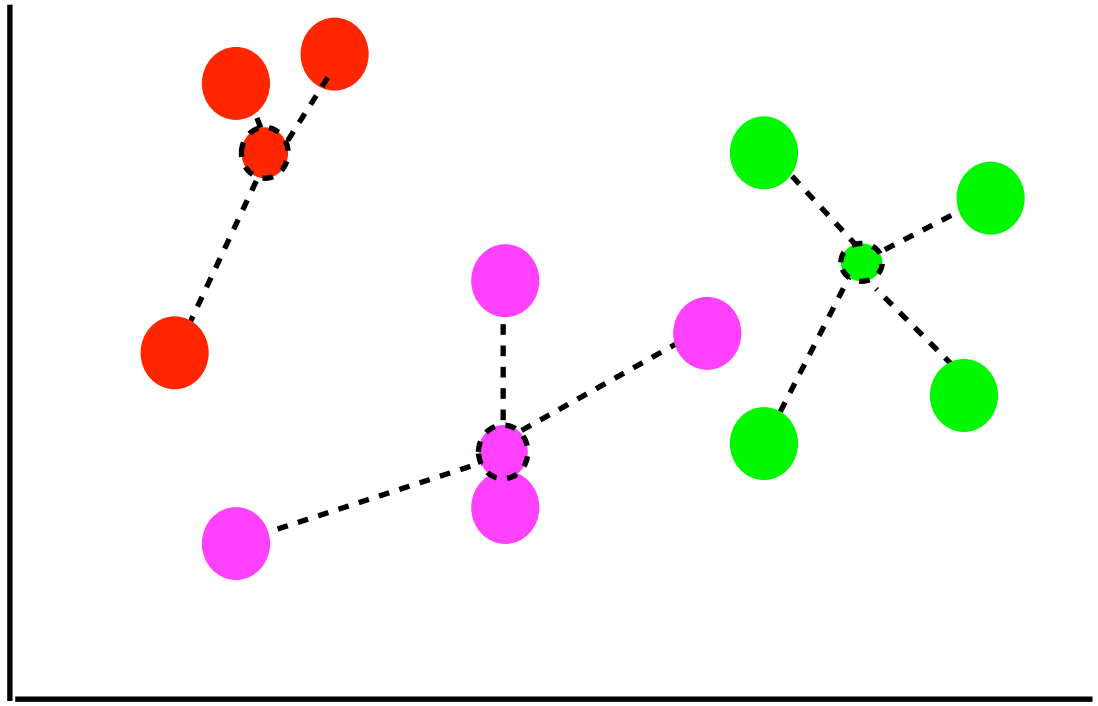
k-means 法

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

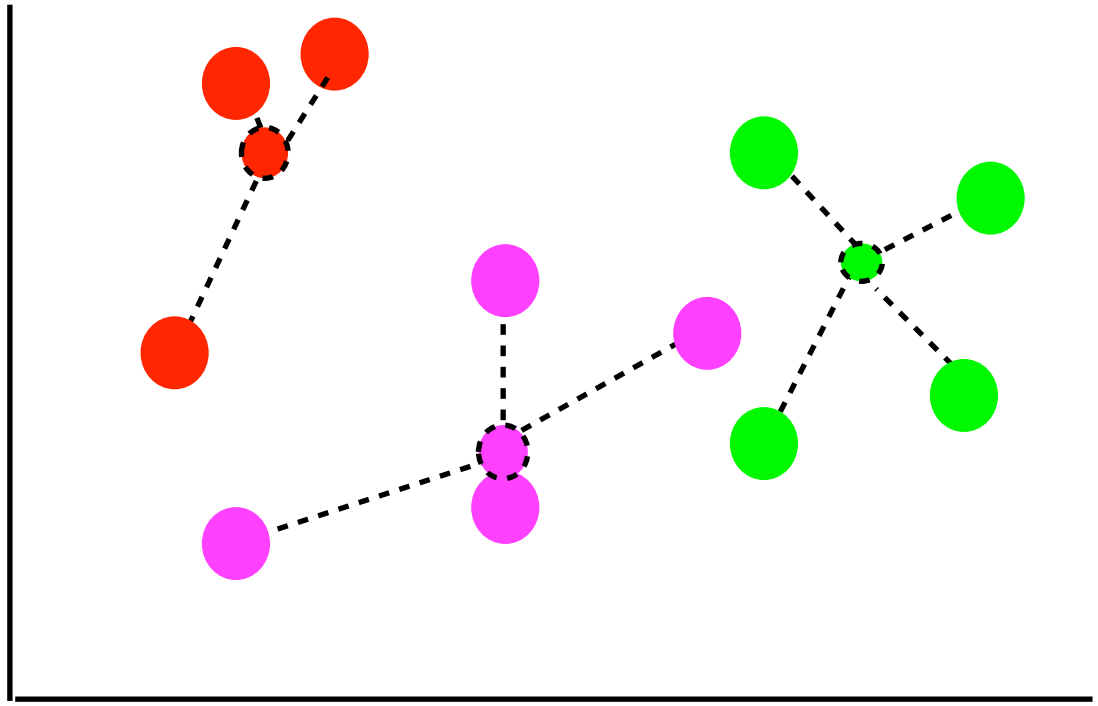


距離が変わるので所属するクラスタが変わります

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],
```

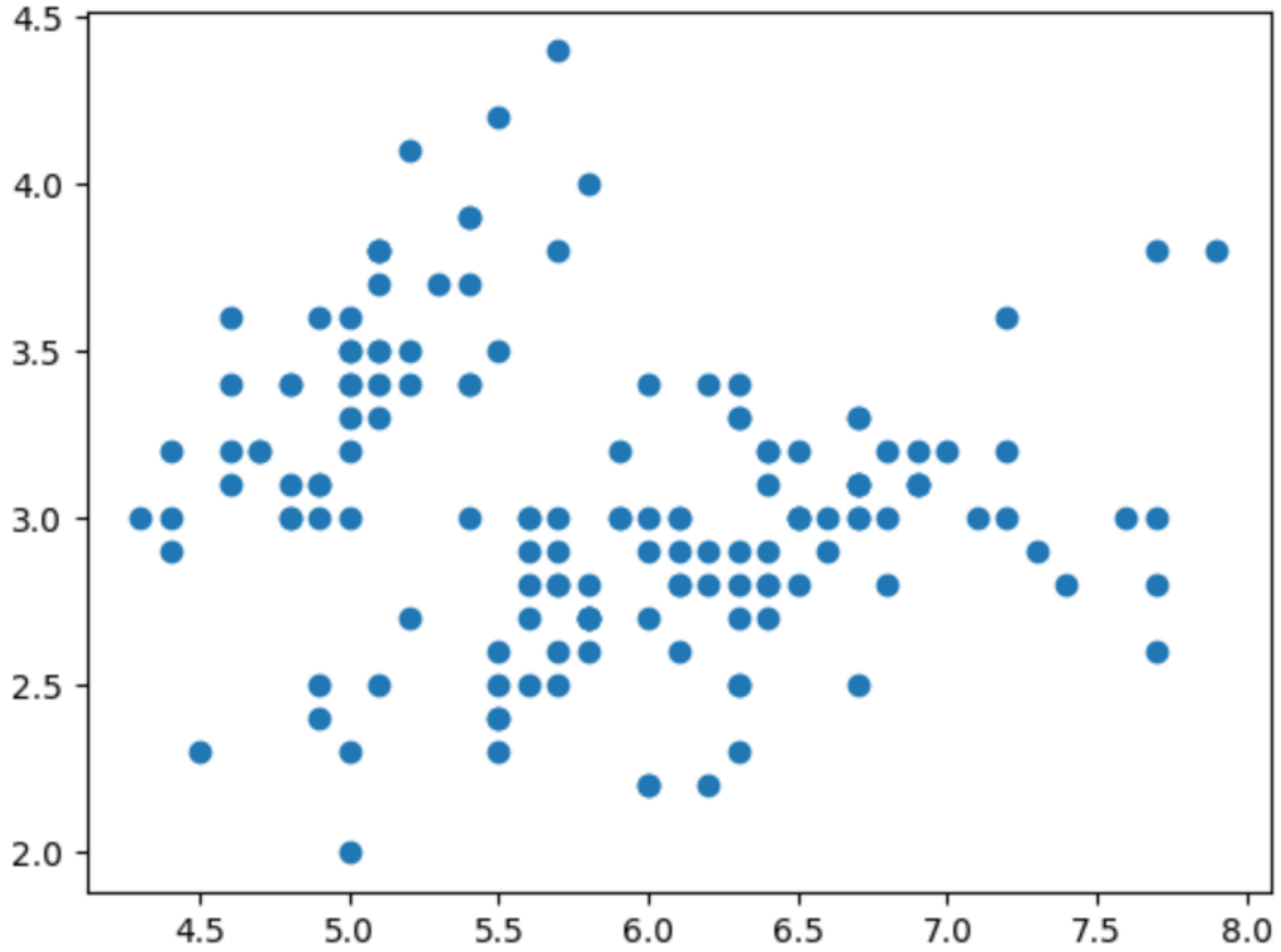
がく片の長さ　がく片の幅　花びらの長さ　花びらの幅

```
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()
```

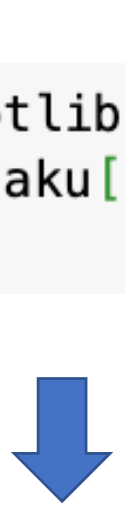
x軸にがく片の長さ、
y軸にがく片の幅として図示



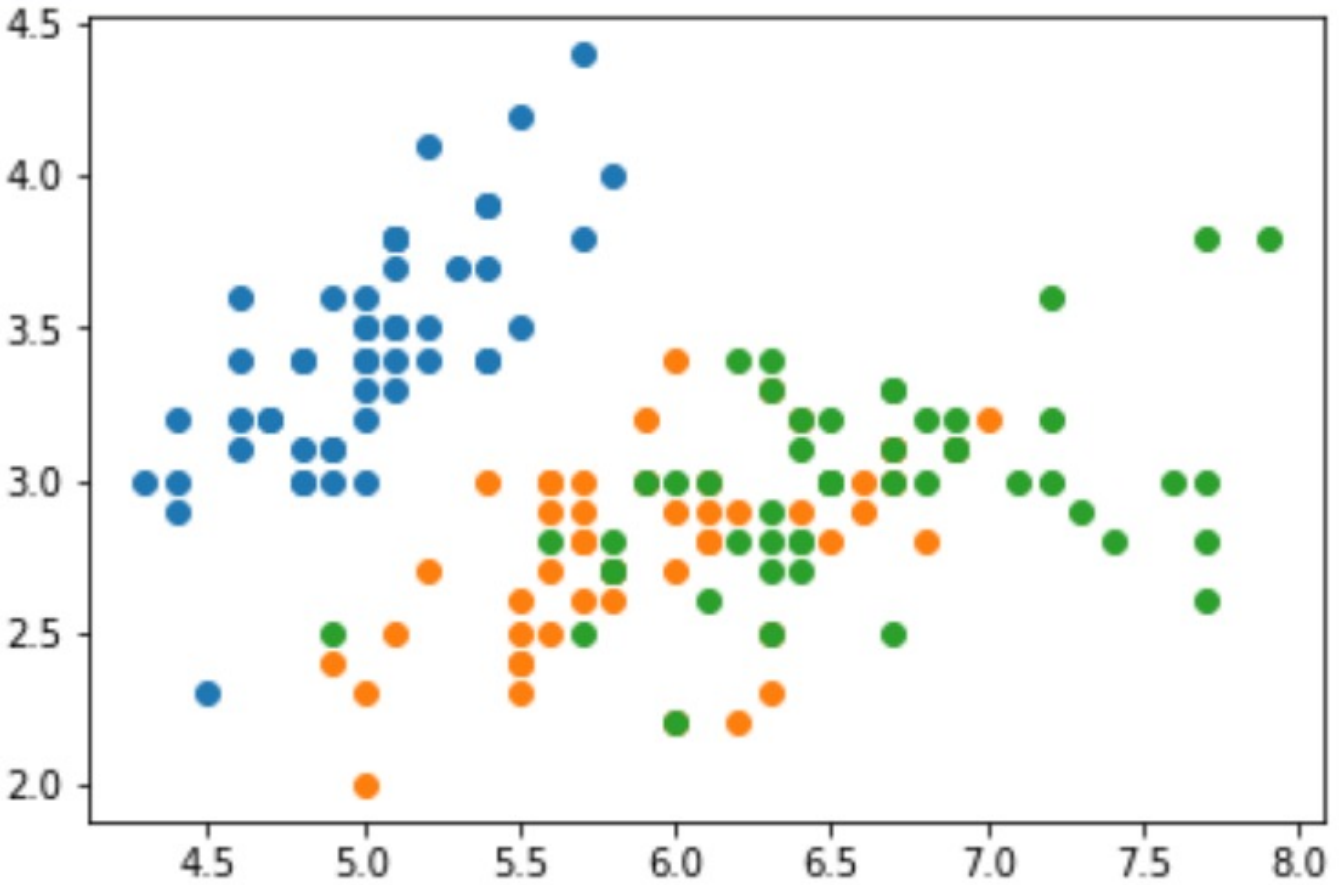
がく片の長さと幅のデータを図示する

アヤメの種類で分ける

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



```
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()
```



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]
```

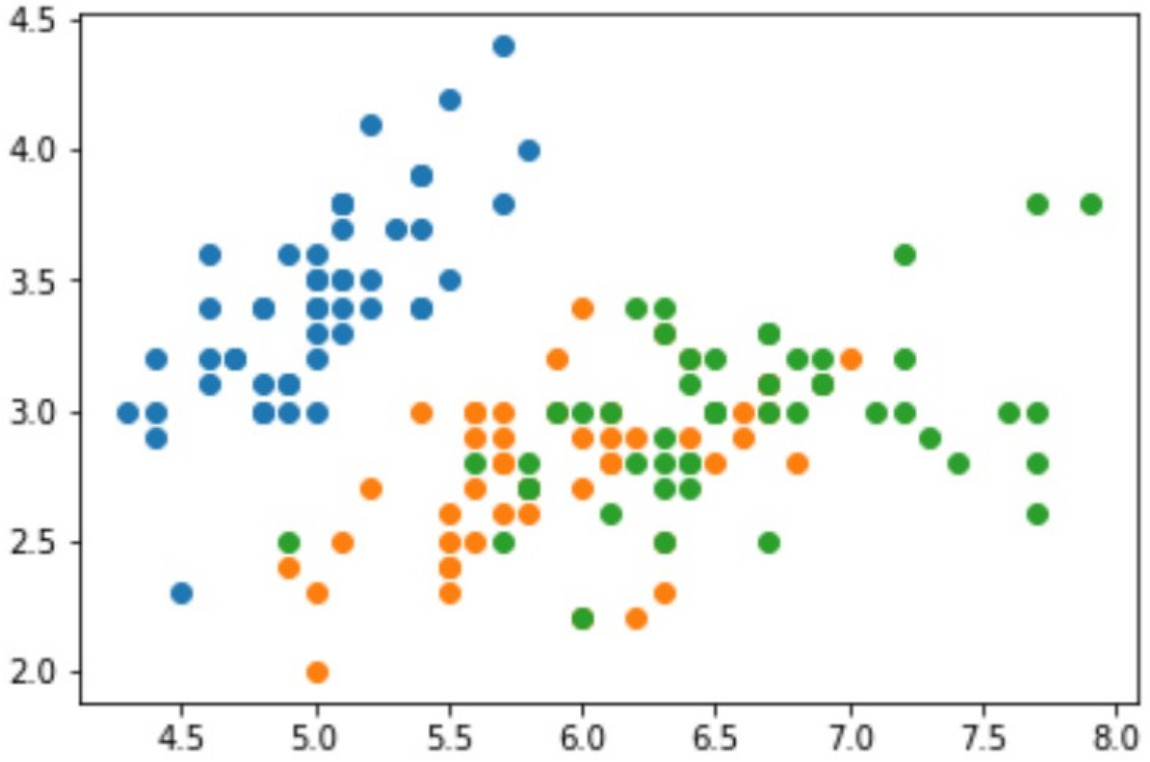
クラスタリングの結果、150個のデータが0，1，2に分けられた。

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

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])
```

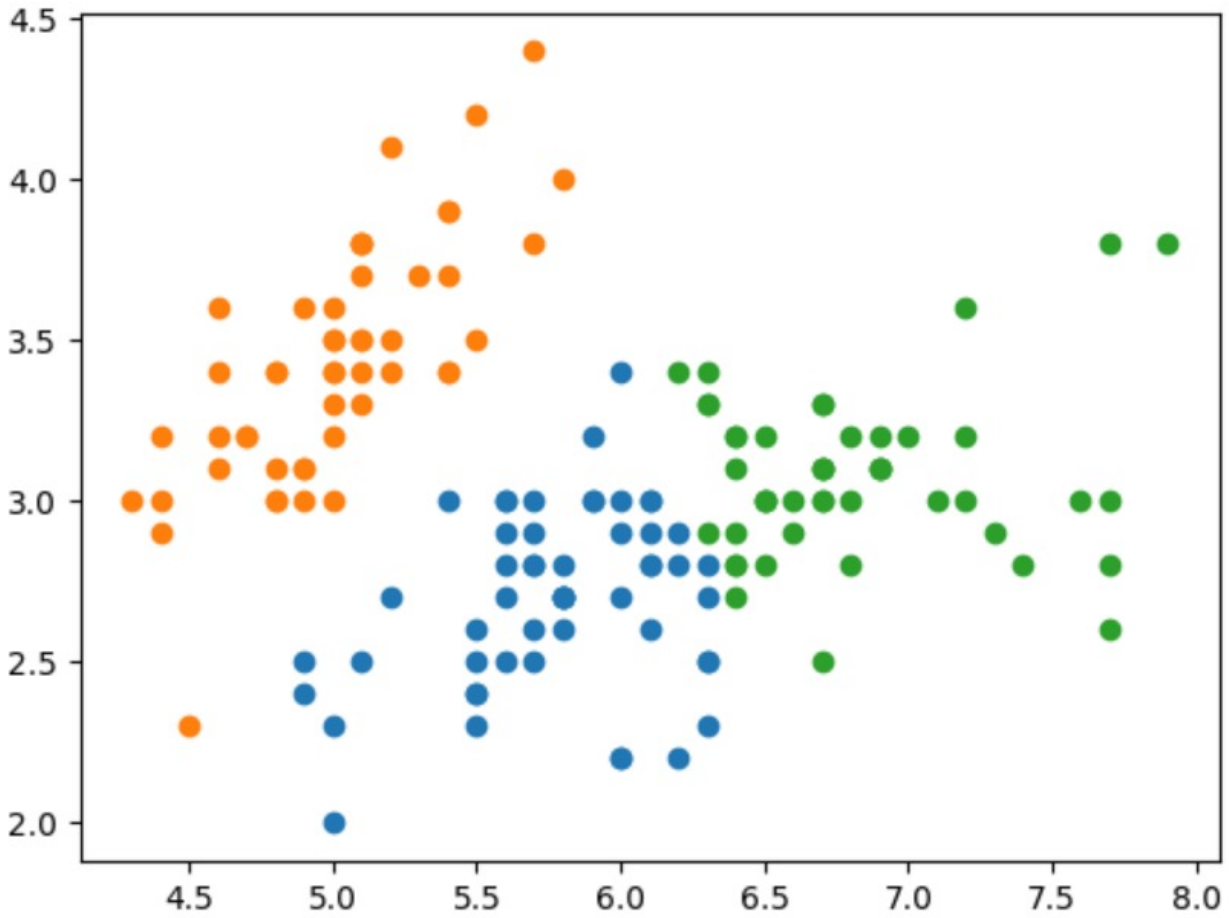
これでクラスタが0の
がく片の長さのみ取り出せた

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)

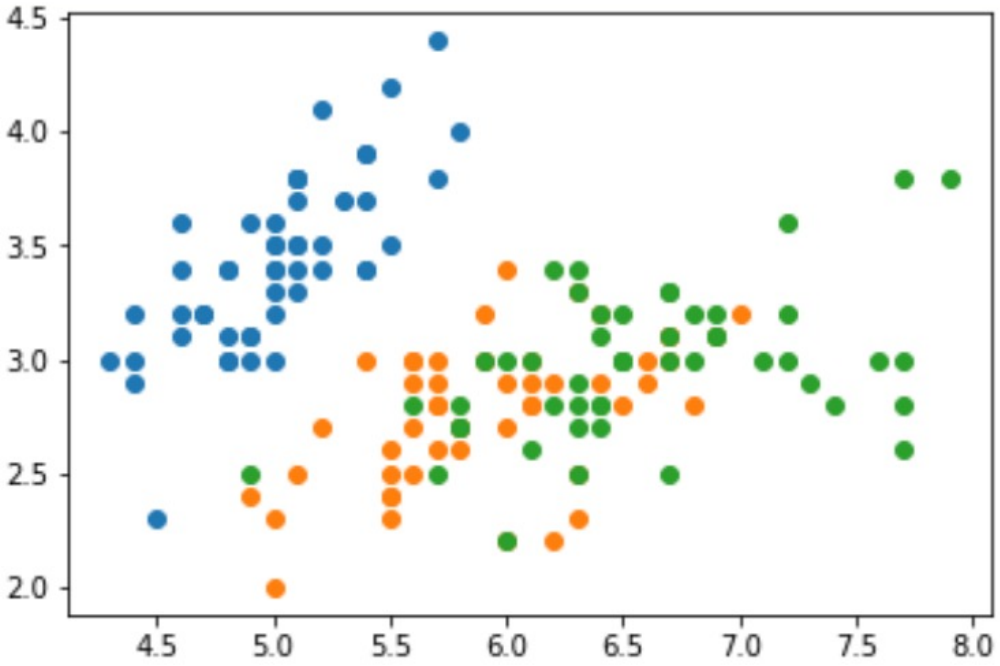


クラスタリングは距離に基づいて
データを決めた個数のグループに分ける

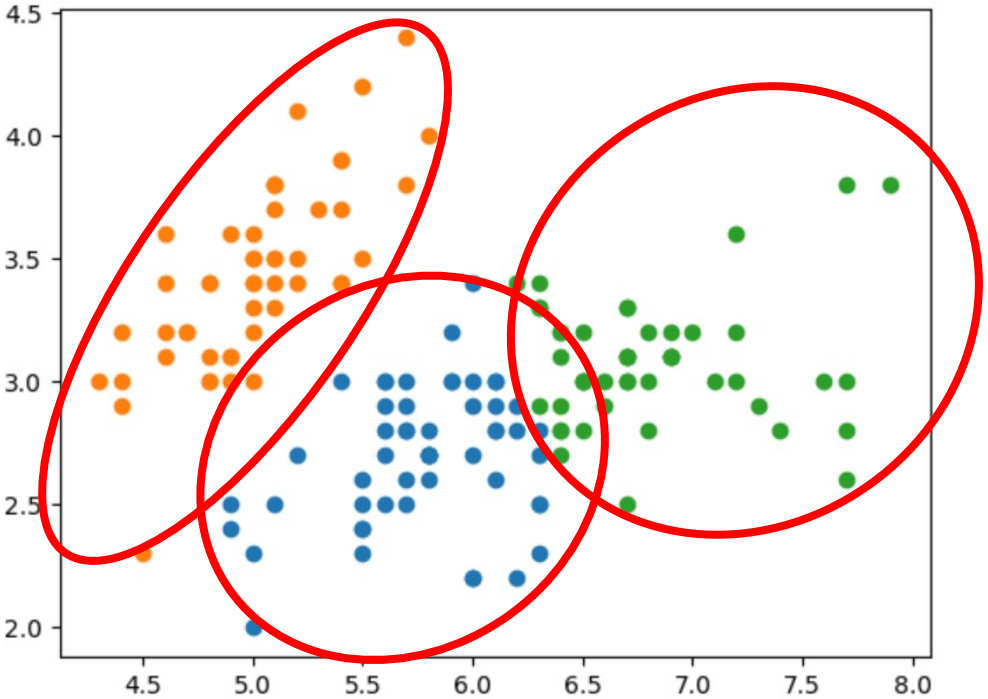
アヤメの正解ラベルを見ずに3つのグループ
に分けられた

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

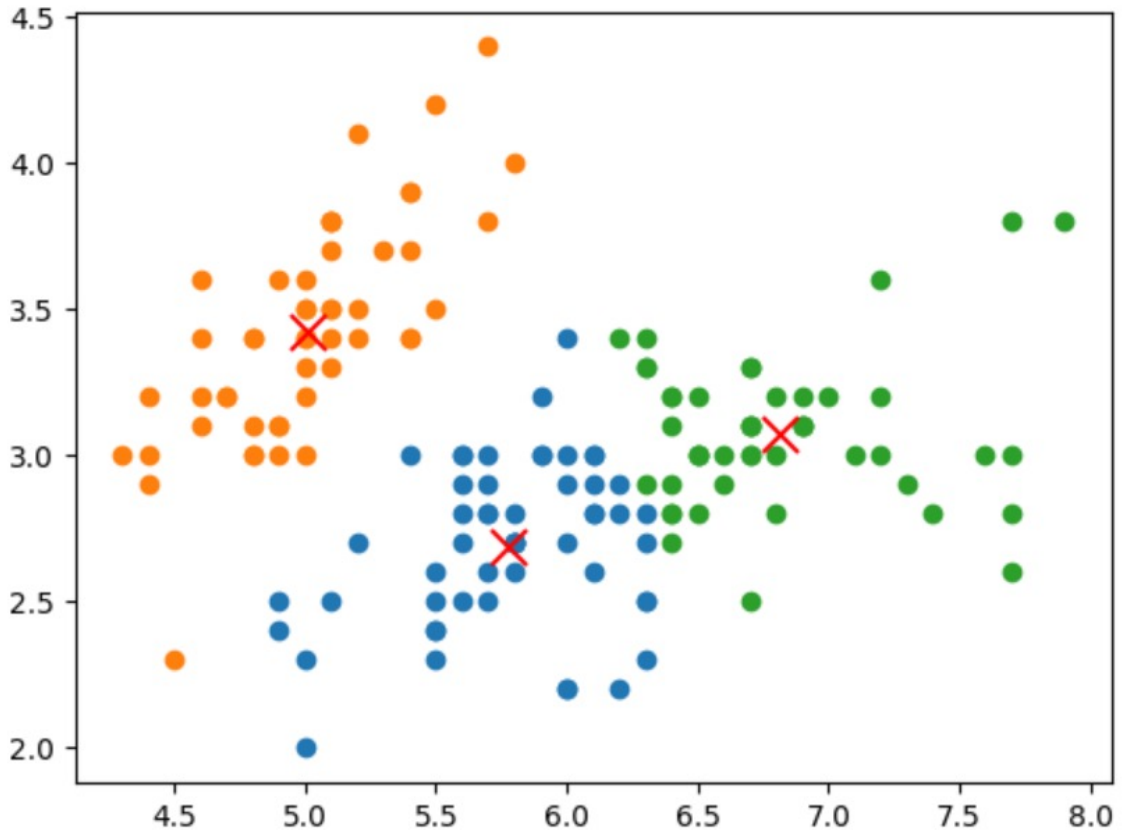
cluster_centers_でクラスタの重心を取得できる

```
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()
```



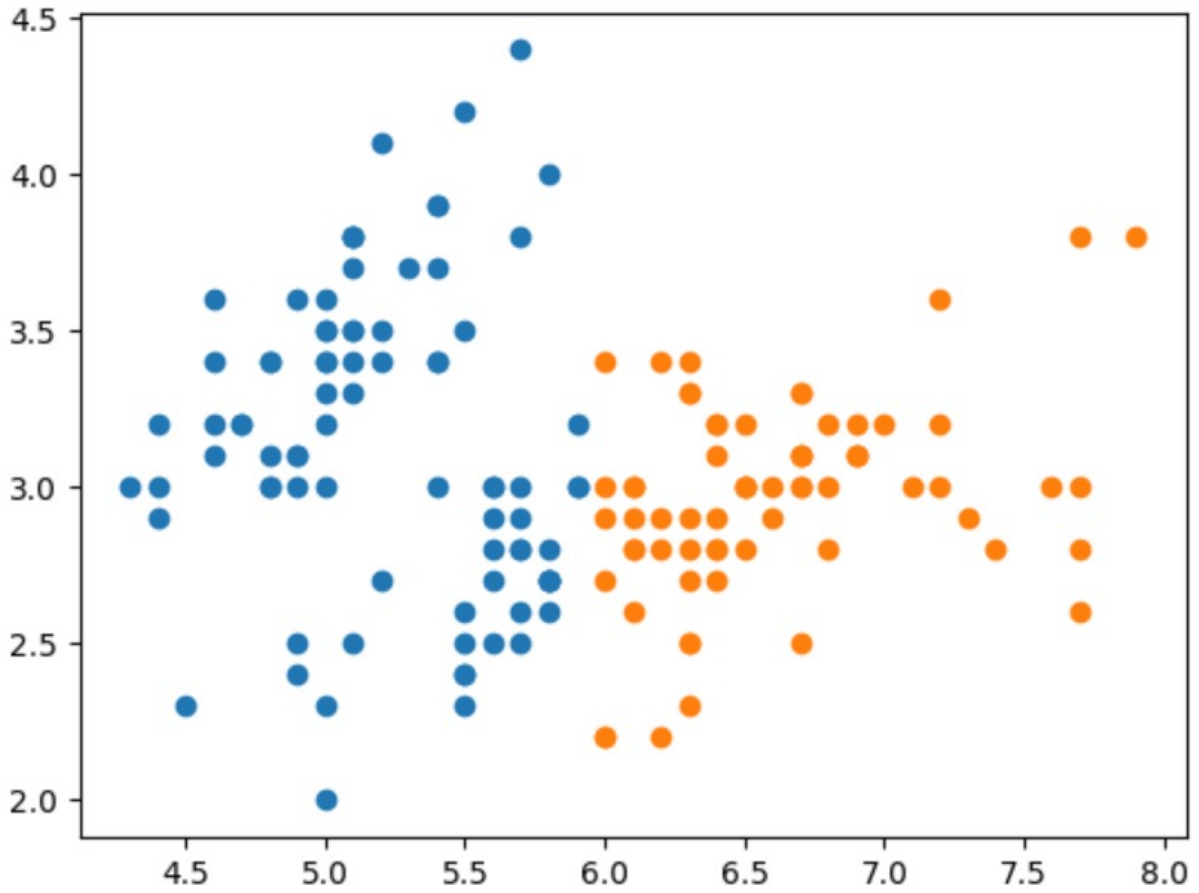
確かに各クラスタの中心に位置していることが確認できる

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

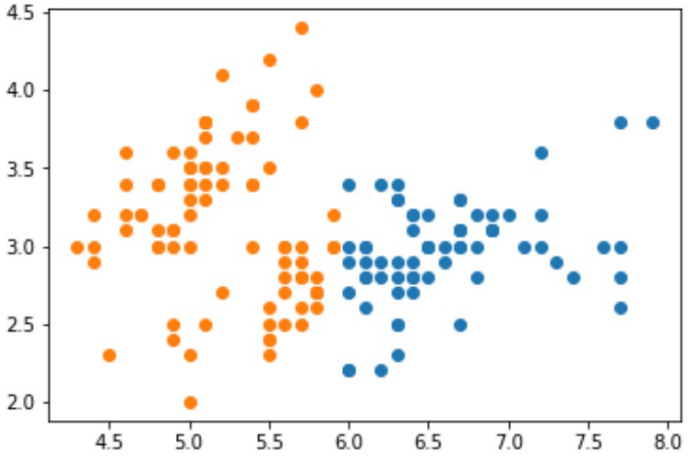
```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 0 1 1 0 1 0 0 1 0 0 1 1 1
 1 1 1 1 1 0 0 0 0 1 0 1 1 1 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 1 1 1 0 1 1 1 1
 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1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



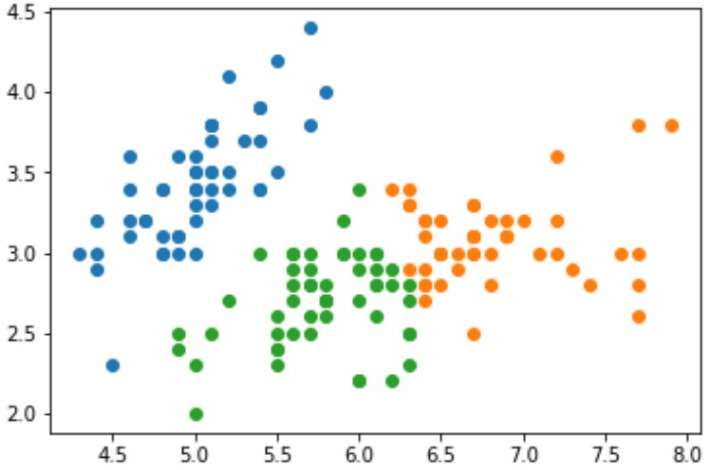
散布図の結果

クラスタ数を変える

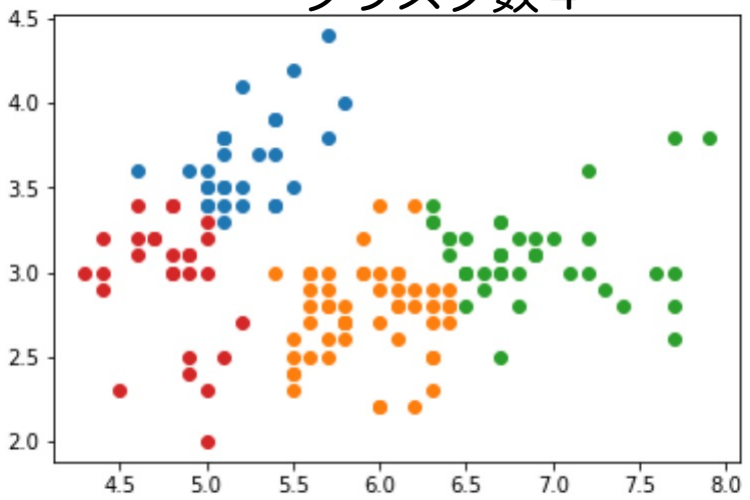
クラスタ数２



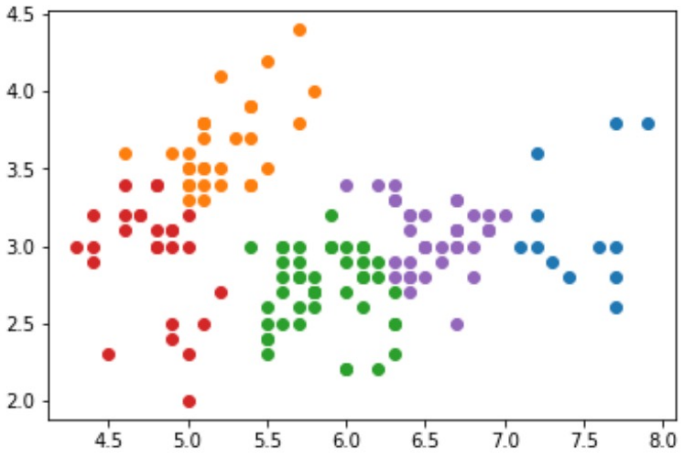
クラスタ数３



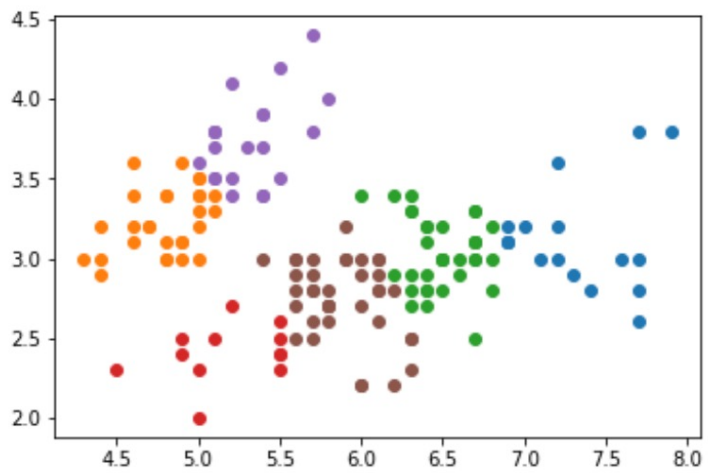
クラスタ数４



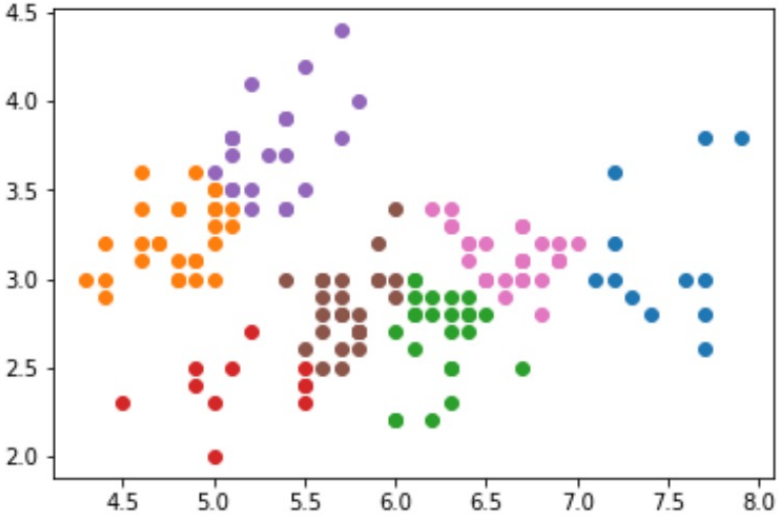
クラスタ数５



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

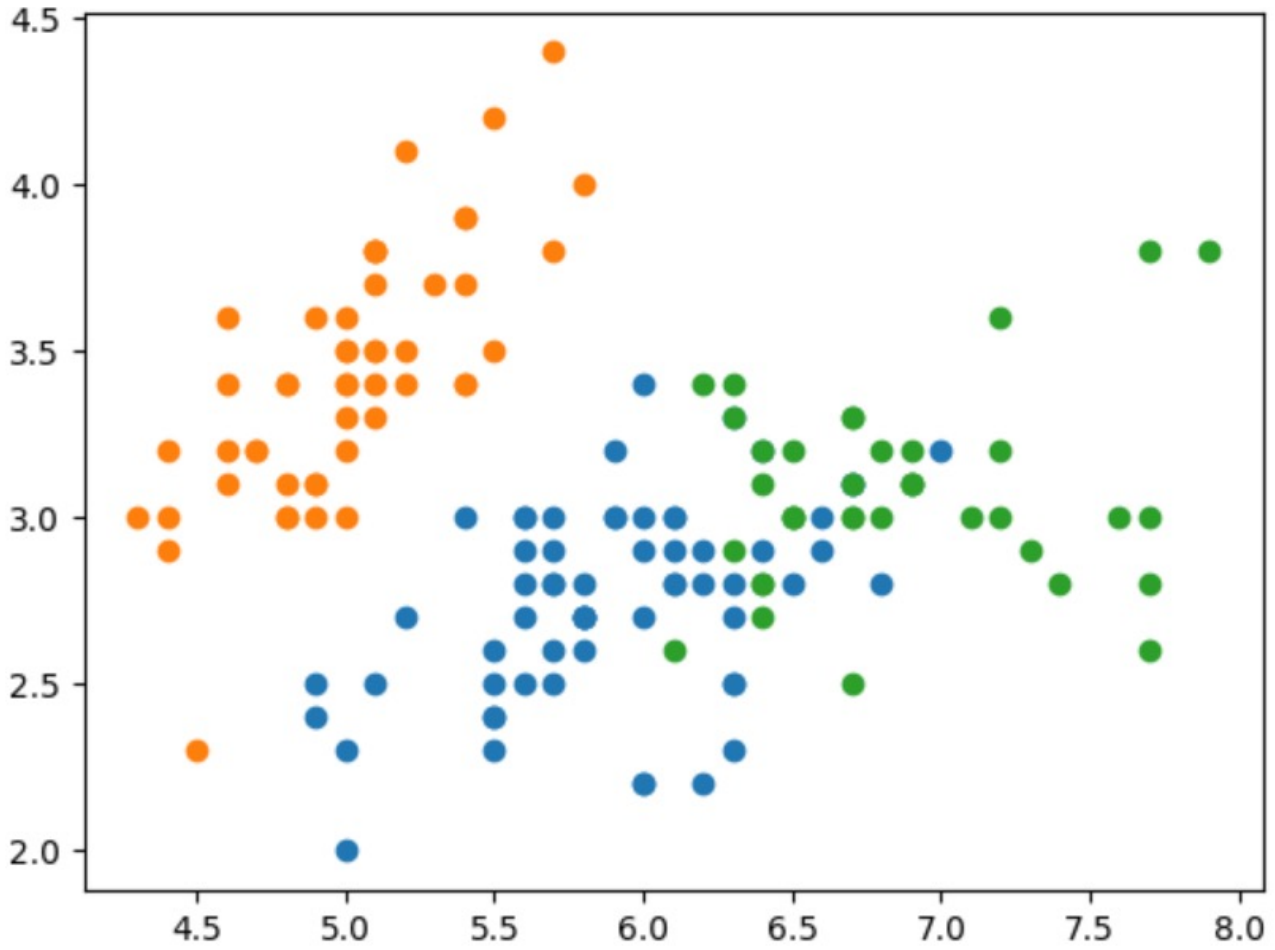
3次元以上のクラスタリング

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

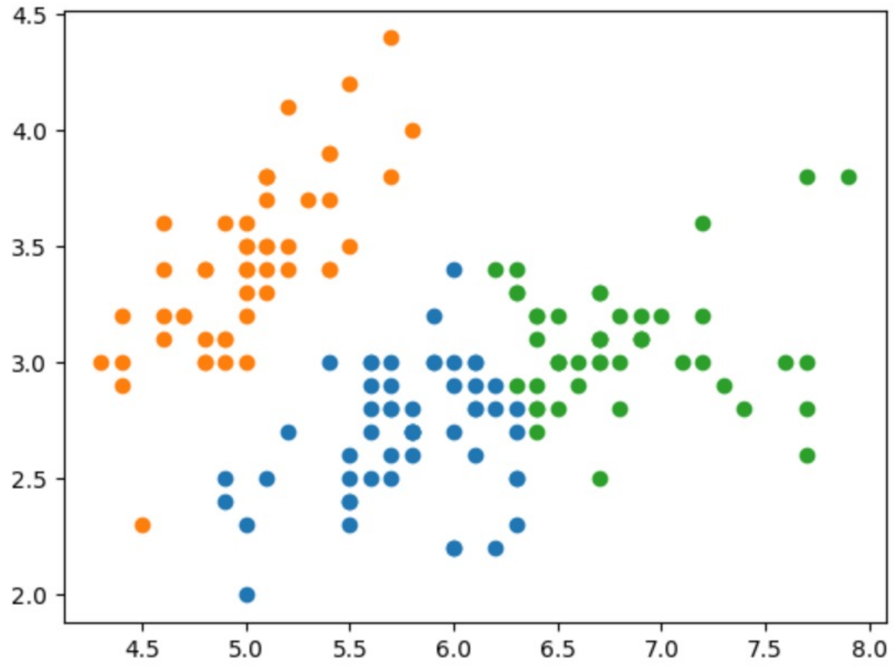
```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2
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



3次元以上のクラスタリング

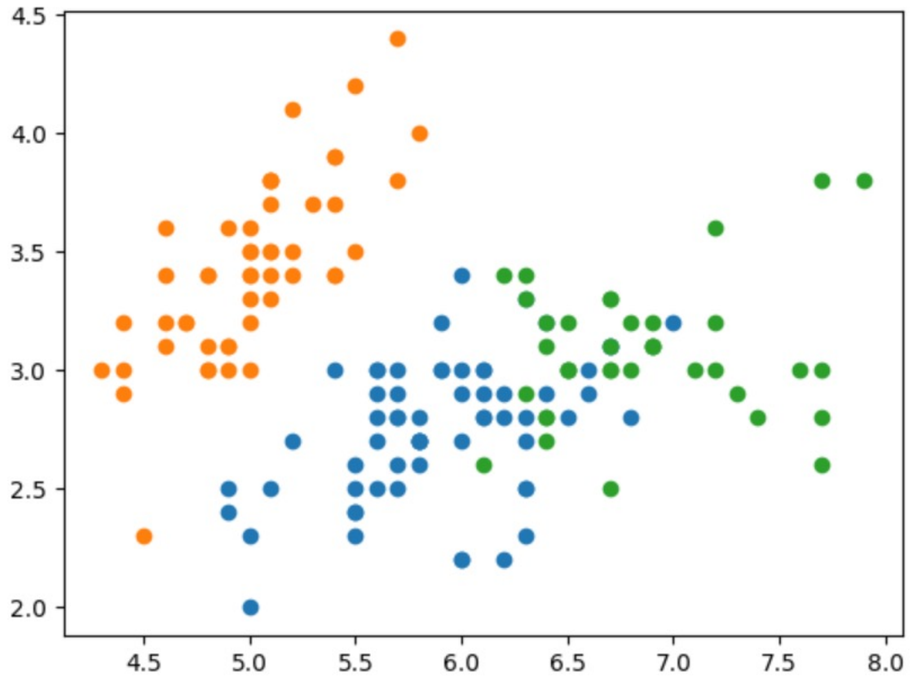
同じクラスタ数(0,1,2)だが、学習するデータ量が違う(2次元と4次元)。
4次元のデータによるクラスタリングを2次元で可視化しているのでやや混ざっている部位もあり。

```
[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 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]
```

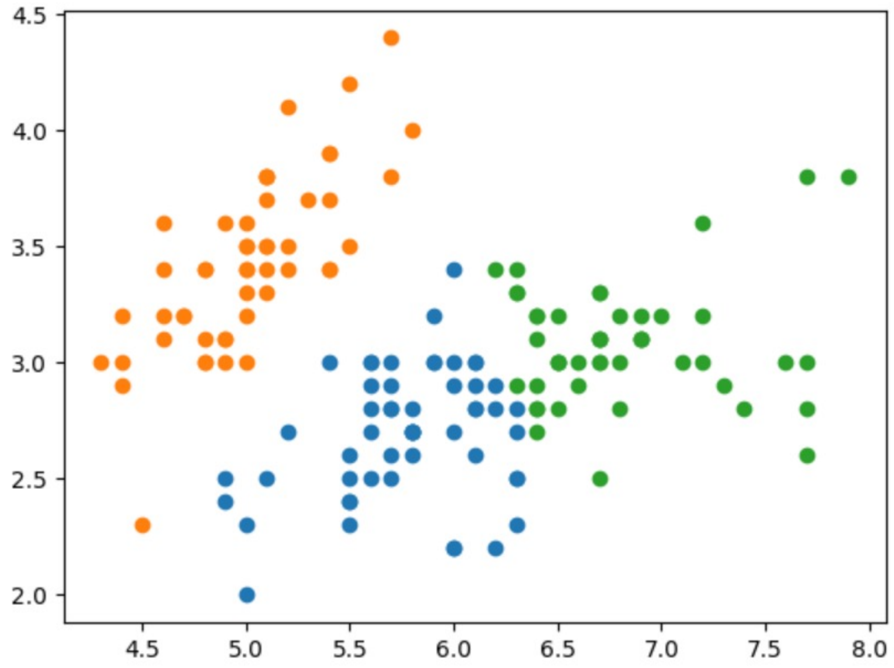


がく片の長さと幅、花びらの長さと幅でクラスタリングし、がく片の長さと幅で図示

3次元以上のクラスタリング

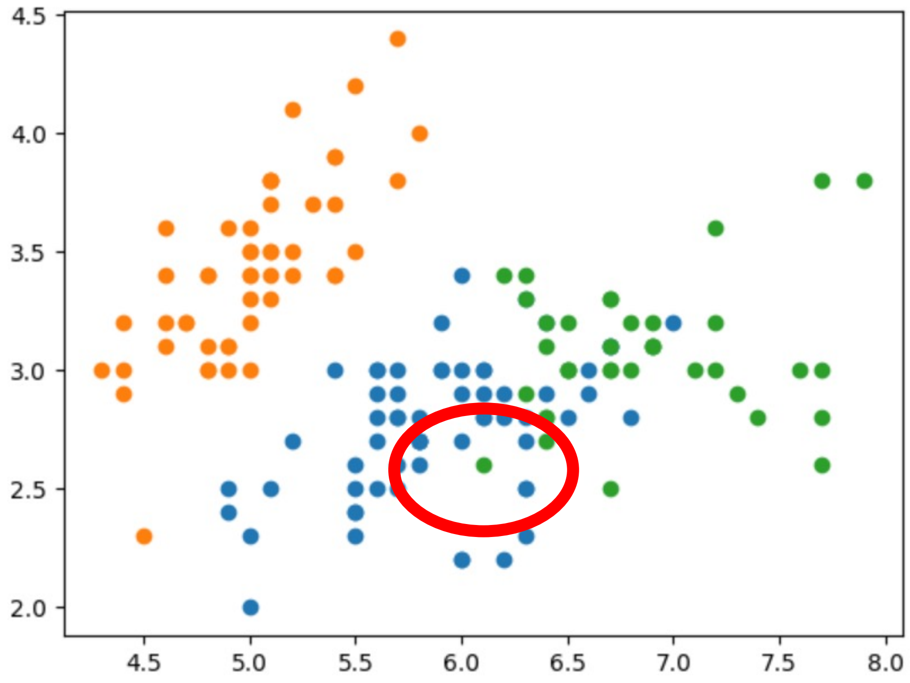
同じクラスタ数(0,1,2)だが、学習するデータ量が違う(2次元と4次元)。
4次元のデータによるクラスタリングを2次元で可視化しているのでやや混ざっている部位もあり。

```
[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

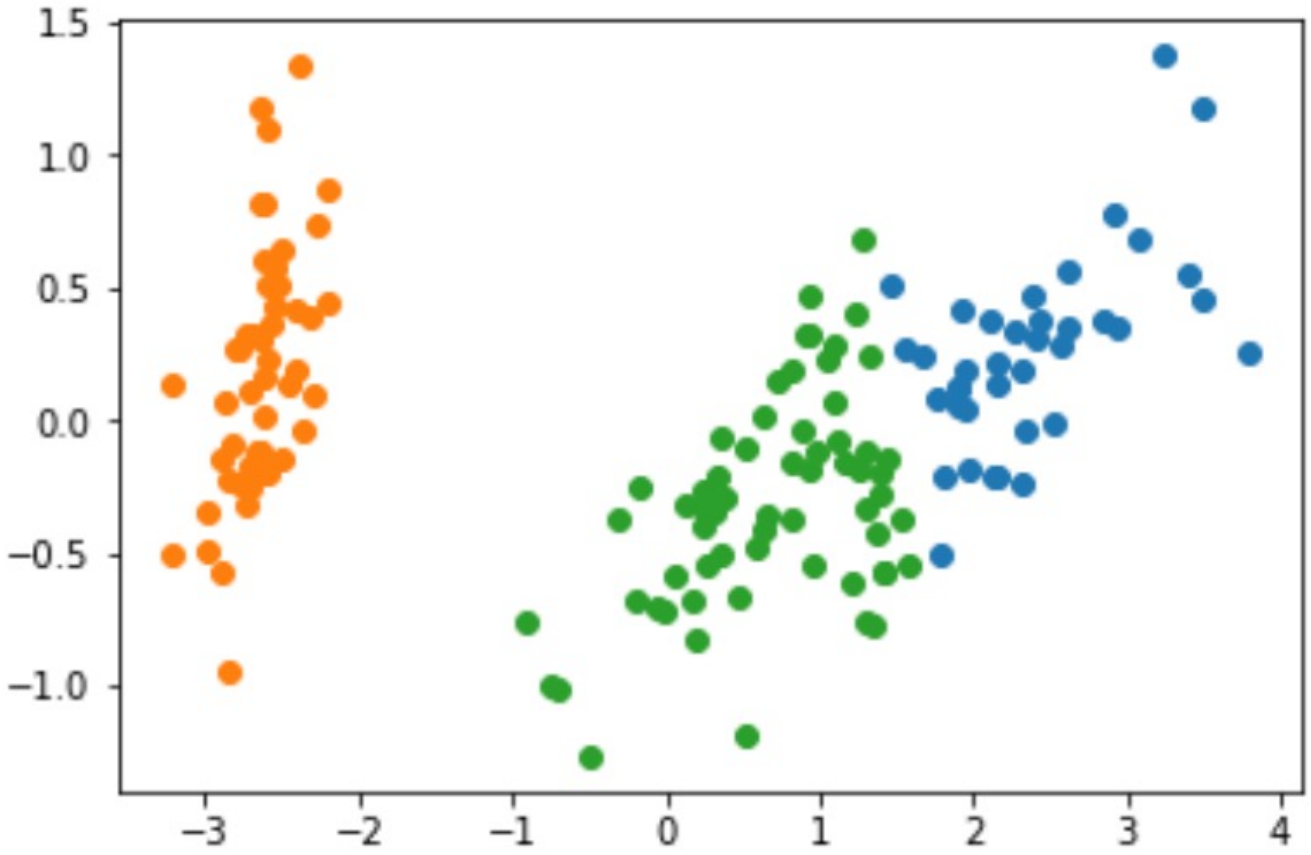
4次元は可視化出来ない　→　次元削減

```
model = KMeans(n_clusters=3,random_state=0)
model.fit(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

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”kadai14_1_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”kadai14_2_学籍番号_名前.png”にしてください

最後にアンケートに答えてください
【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