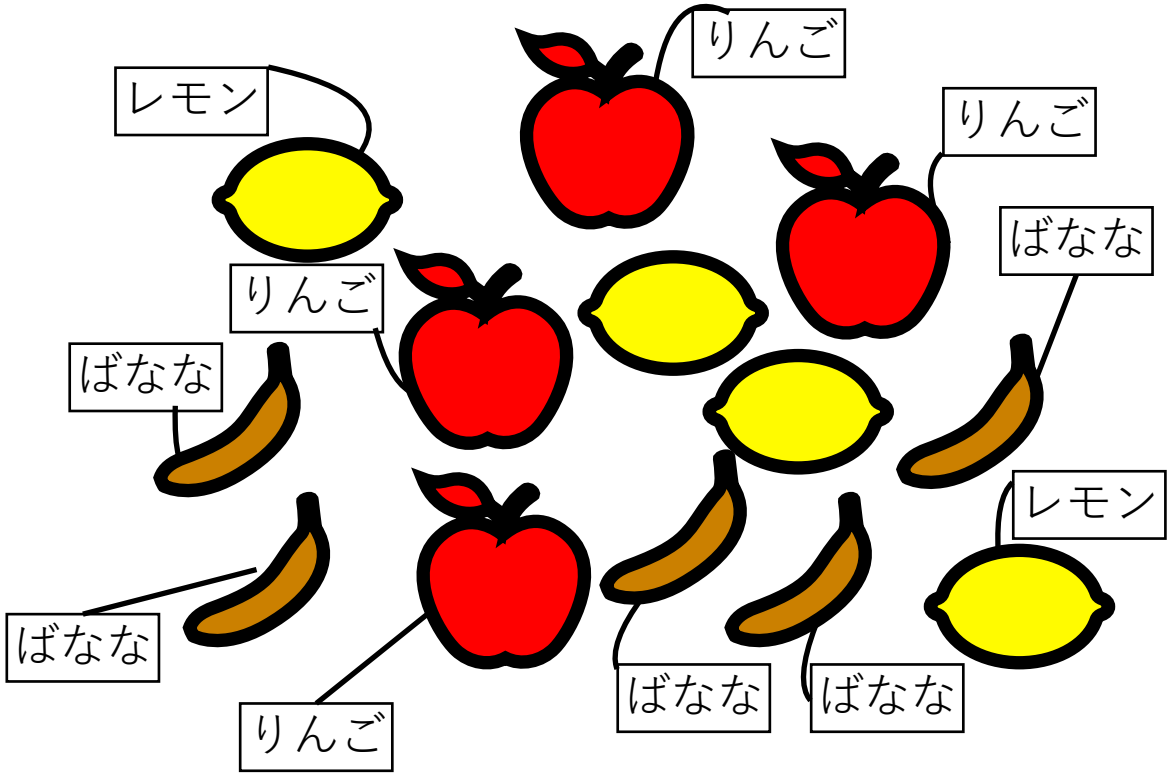
教師なし機械学習

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

教師あり機械学習の予測



入力

学習

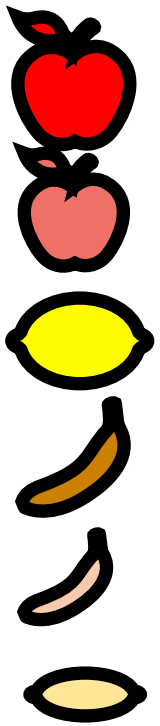
識別器

（≒分類マシーン）

りんご!!

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

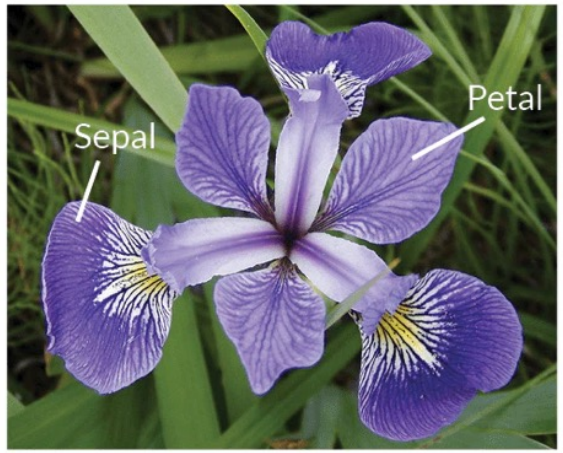
識別器
（≒分類マシーン）

	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

次元削減の代表例：主成分分析をアヤメのデータで実施してみる

あやめのデータ



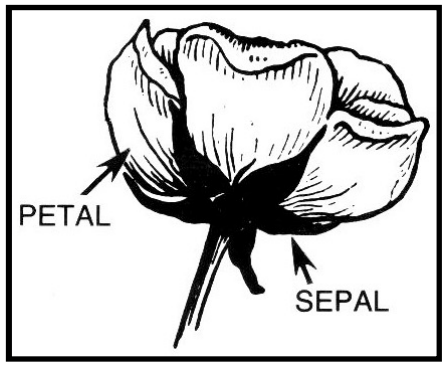
Iris Versicolor
ブルーフラッグ

Iris Setosa
ヒオウギアヤメ

Iris Virginica
バージニカ

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)



	A	B	C	D	E	F
1	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_数字	アヤメの種類
2	5.1	3.5	1.4	0.2	0	ヒオウギアヤメ
3	4.9	3	1.4	0.2	0	ヒオウギアヤメ
4	4.7	3.2	1.3	0.2	0	ヒオウギアヤメ
5	4.6	3.1	1.5	0.2	0	ヒオウギアヤメ
6	5	3.6	1.4	0.2	0	ヒオウギアヤメ
7	5.4	3.9	1.7	0.4	0	ヒオウギアヤメ
8	4.6	3.4	1.4	0.3	0	ヒオウギアヤメ
9	5	3.4	1.5	0.2	0	ヒオウギアヤメ
10	4.4	2.9	1.4	0.2	0	ヒオウギアヤメ
11	4.9	3.1	1.5	0.1	0	ヒオウギアヤメ
12	5.4	3.7	1.5	0.2	0	ヒオウギアヤメ
⋮						
50	5.3	3.7	1.5	0.2	0	ヒオウギアヤメ
51	5	3.3	1.4	0.2	0	ヒオウギアヤメ
52	7	3.2	4.7	1.4	1	ブルーフラッグ
53	6.4	3.2	4.5	1.5	1	ブルーフラッグ
54	6.9	3.1	4.9	1.5	1	ブルーフラッグ
55	5.5	2.3	4	1.3	1	ブルーフラッグ
56	6.5	2.8	4.6	1.5	1	ブルーフラッグ
57	5.7	2.8	4.5	1.3	1	ブルーフラッグ
58	6.3	3.3	4.7	1.6	1	ブルーフラッグ
⋮						
141	6.9	3.1	5.4	2.1	2	バージニカ
142	6.7	3.1	5.6	2.4	2	バージニカ
143	6.9	3.1	5.1	2.3	2	バージニカ
144	5.8	2.7	5.1	1.9	2	バージニカ
145	6.8	3.2	5.9	2.3	2	バージニカ
146	6.7	3.3	5.7	2.5	2	バージニカ
147	6.7	3	5.2	2.3	2	バージニカ
148	6.3	2.5	5	1.9	2	バージニカ
149	6.5	3	5.2	2	2	バージニカ
150	6.2	3.4	5.4	2.3	2	バージニカ
151	5.9	3	5.1	1.8	2	バージニカ

(入門で扱っていたデータ)150行✖6列

アイリスのデータセットはscikit-learnに入っている

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

アイリスのデータセットはscikit-learnに入っている

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

```
In [82]: print(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],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
```

```
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': 'C:\\WPy64-3771\\python-3.7.7.amd64\\lib\\site-packages\\sklearn\
\datasets\\data\\iris.csv'}

In [83]:
```

load_iris(=iris)の中身

```
{     ‘data’ : 特徴量のデータ,
      ‘target’ : 正解ラベル(ヒオウギアヤメ=0, ブルーフラッグ=1, バージニカ=2),
‘target_names’ : array[‘setosa’, ‘versicolor’, ‘virginica’],
      ‘DESCR’: load_irisの説明                }
```

これらのデータは変数名.data、変数名.target、変数名.target_names、変数名.DESCRと書くことで取得できます(中身は全てnumpy配列)

```
print(iris.data)
print(iris.target_names)
print(iris.target)
```

load_iris(=iris)の中身

```
{     'data' : 特徴量のデータ,
        'target' : 正解ラベル(ヒオウギアヤメ=0, ブルーフラッグ=1, バージニカ=2),
'target_names' : array['setosa', 'versicolor', 'virginica'],
        'DESCR': load_irisの説明              }
```

これらのデータは変数名.data、変数名.target、変数名.target_names、変数名.DESCR
と書くことで取得できます(中身は全てnumpy配列)

```
print(iris.data)
print(iris.target_names)
print(iris.target)
```

```
In [83]: print(iris.data)
[[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]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
```

	A	B	C	D	E	F
1	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_数字	アヤメの種類
2	5.1	3.5	1.4	0.2	0	ヒオウギアヤメ
3	4.9	3	1.4	0.2	0	ヒオウギアヤメ
4	4.7	3.2	1.3	0.2	0	ヒオウギアヤメ
5	4.6	3.1	1.5	0.2	0	ヒオウギアヤメ
6	5	3.6	1.4	0.2	0	ヒオウギアヤメ
7	5.4	3.9	1.7	0.4	0	ヒオウギアヤメ
8	4.6	3.4	1.4	0.3	0	ヒオウギアヤメ
9	5	3.4	1.5	0.2	0	ヒオウギアヤメ
10	4.4	2.9	1.4	0.2	0	ヒオウギアヤメ
11	4.9	3.1	1.5	0.1	0	ヒオウギアヤメ
12	5.4	3.7	1.5	0.2	0	ヒオウギアヤメ

(入門で扱っていたデータ)

load_iris(=iris)の中身

{　‘data’ : 特徴量のデータ,
‘target’ : 正解ラベル(ヒオウギアヤメ=0, ブルーフラッグ=1, バージニカ=2),
‘target_names’ : array[‘setosa’, ‘versicolor’, ‘virginica’],
‘DESCR’: load_irisの説明　　}

これらのデータは変数名.data、変数名.target、変数名.target_names、変数名.DESCR
と書くことで取得できます(中身は全てnumpy配列)

```
print(iris.data)
print(iris.target_names)
print(iris.target)
```

```
In [84]: print(iris.target_names)
['setosa' 'versicolor' 'virginica']

In [85]: print(iris.target)
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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]
```

次元削減の例：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

4つの特徴量を

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

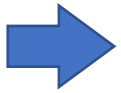
or

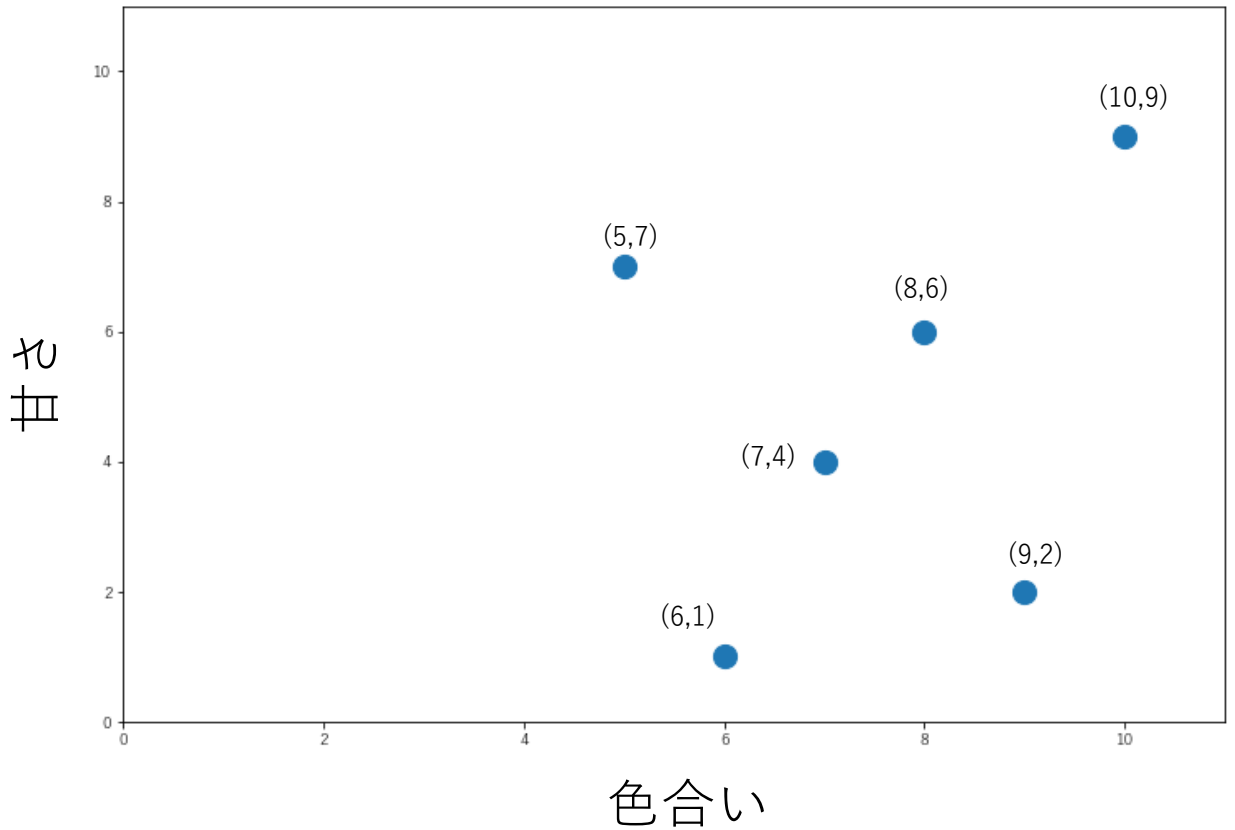
	主成分１	主成分２	主成分３
0			
1			
2			
3			
4			
...			
145			
146			
147			
148			
149			

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？

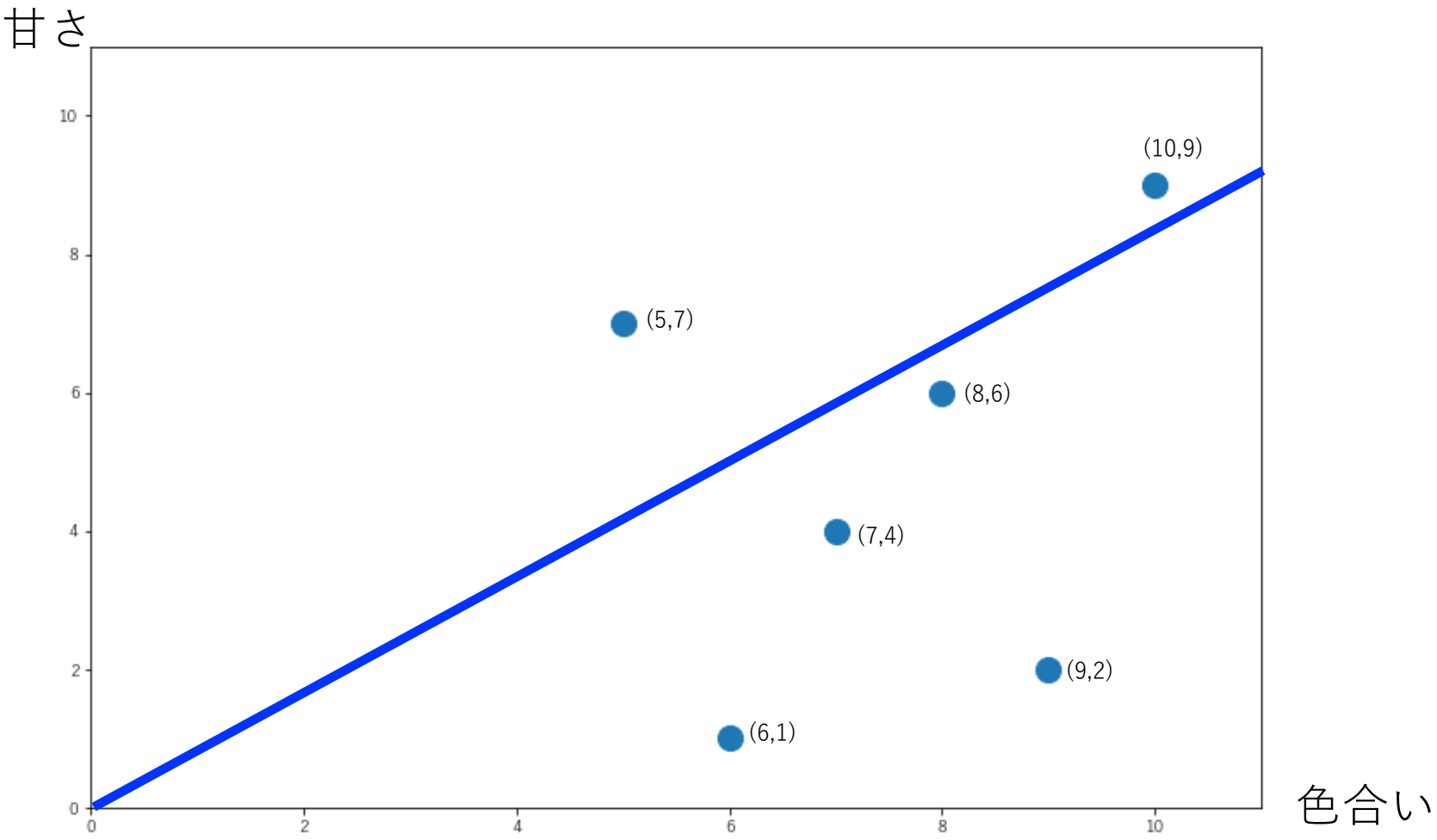
	色合い	大きさ	甘さレベル
りんご1	10	100	9
りんご2	8	88	6
れもん1	9	80	2
バナナ1	5	73	7
バナナ2	7	50	4
れもん2	6	40	1





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

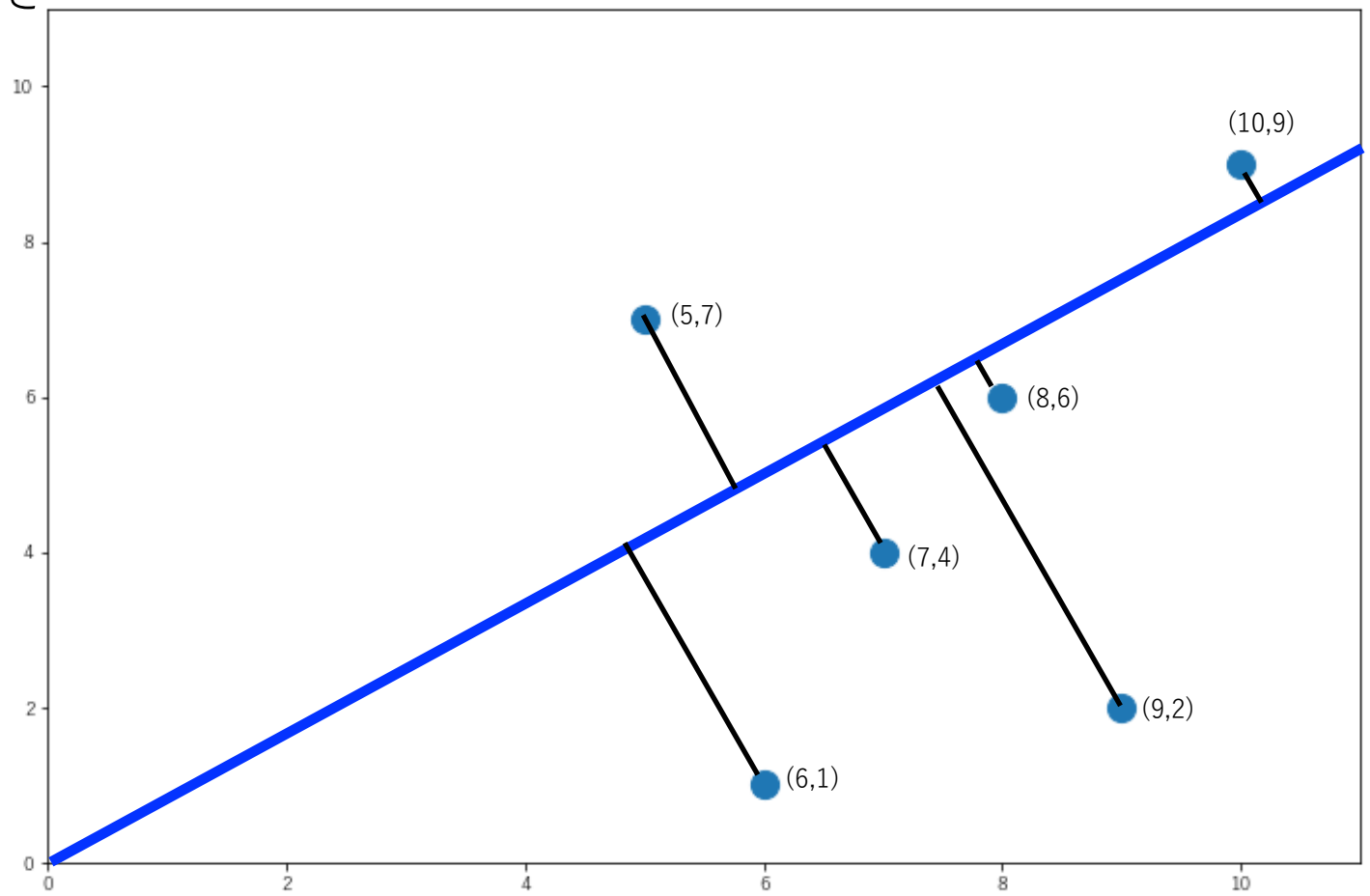
どうやって新しい特徴量を作成しているのか？



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？

甘さ

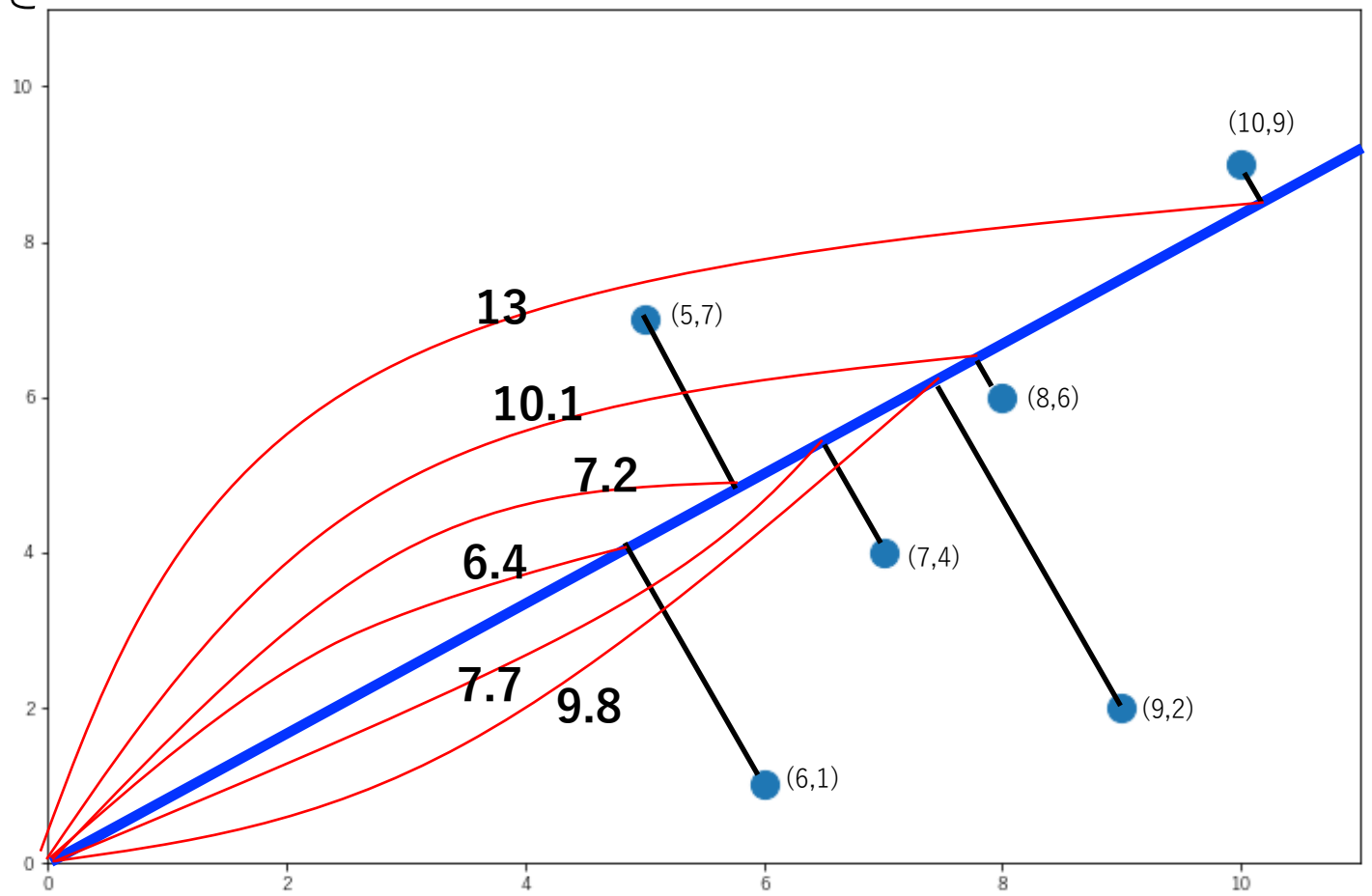


色合い

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？

甘さ

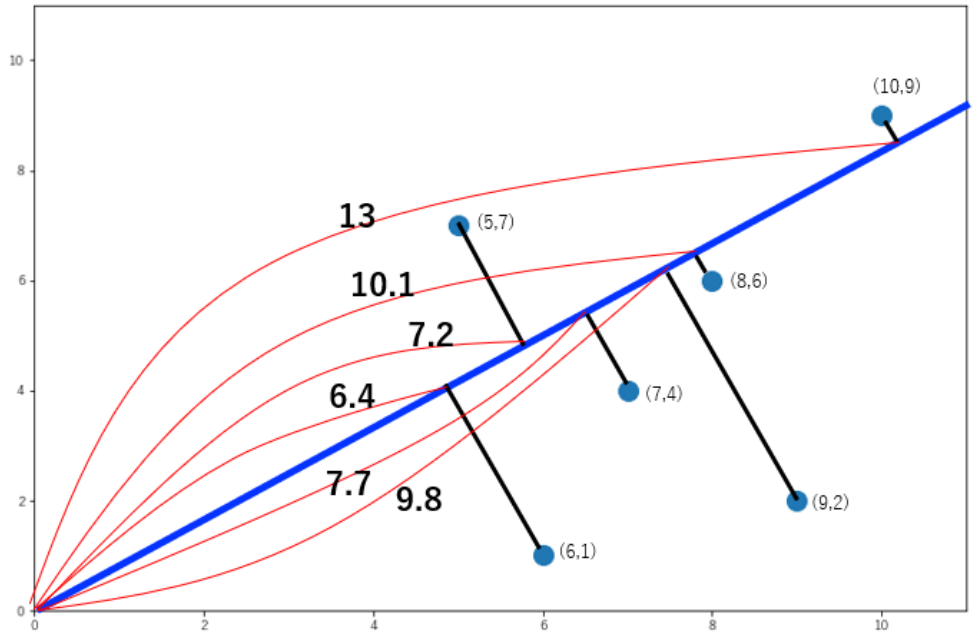


色合い

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



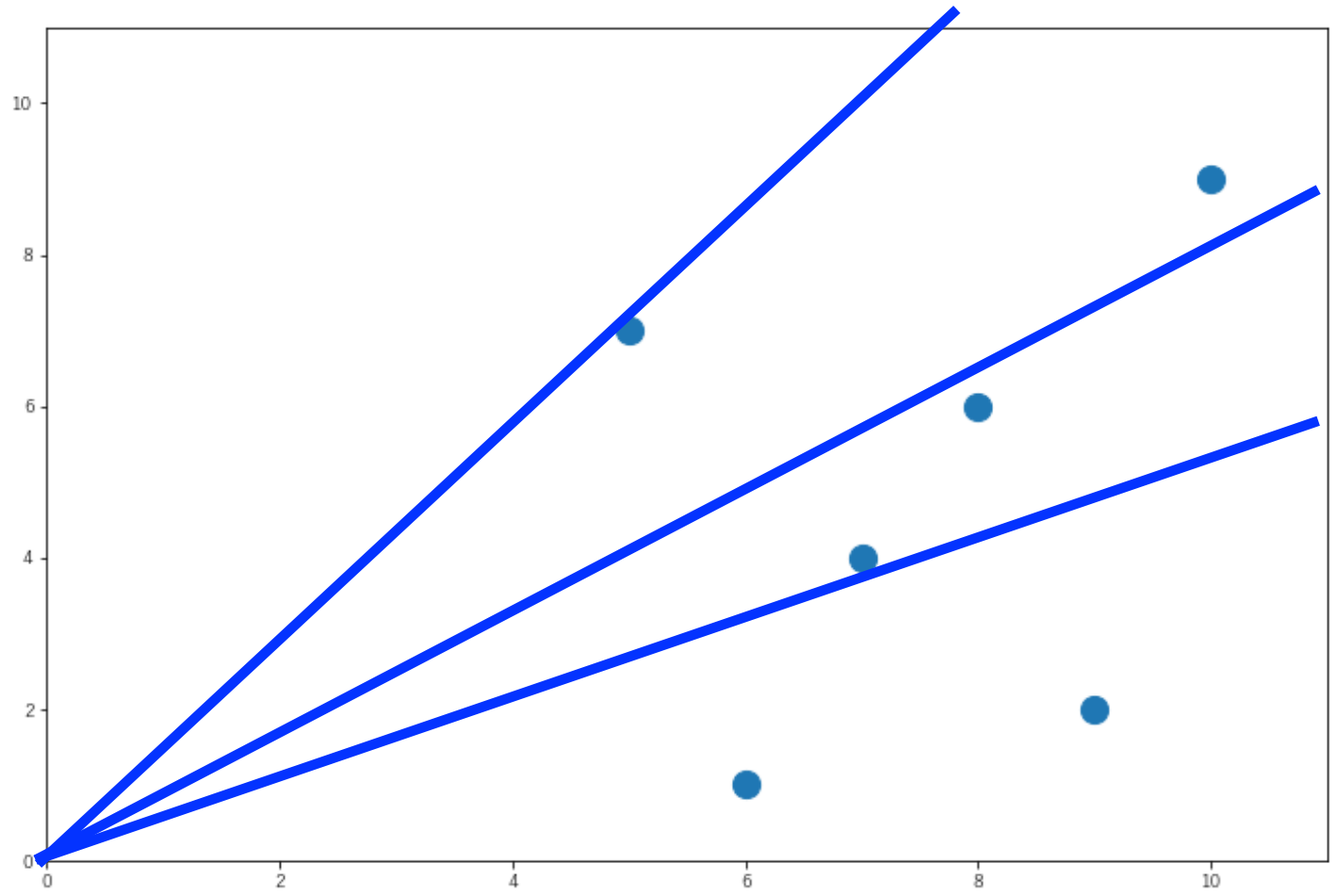
色合い

	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

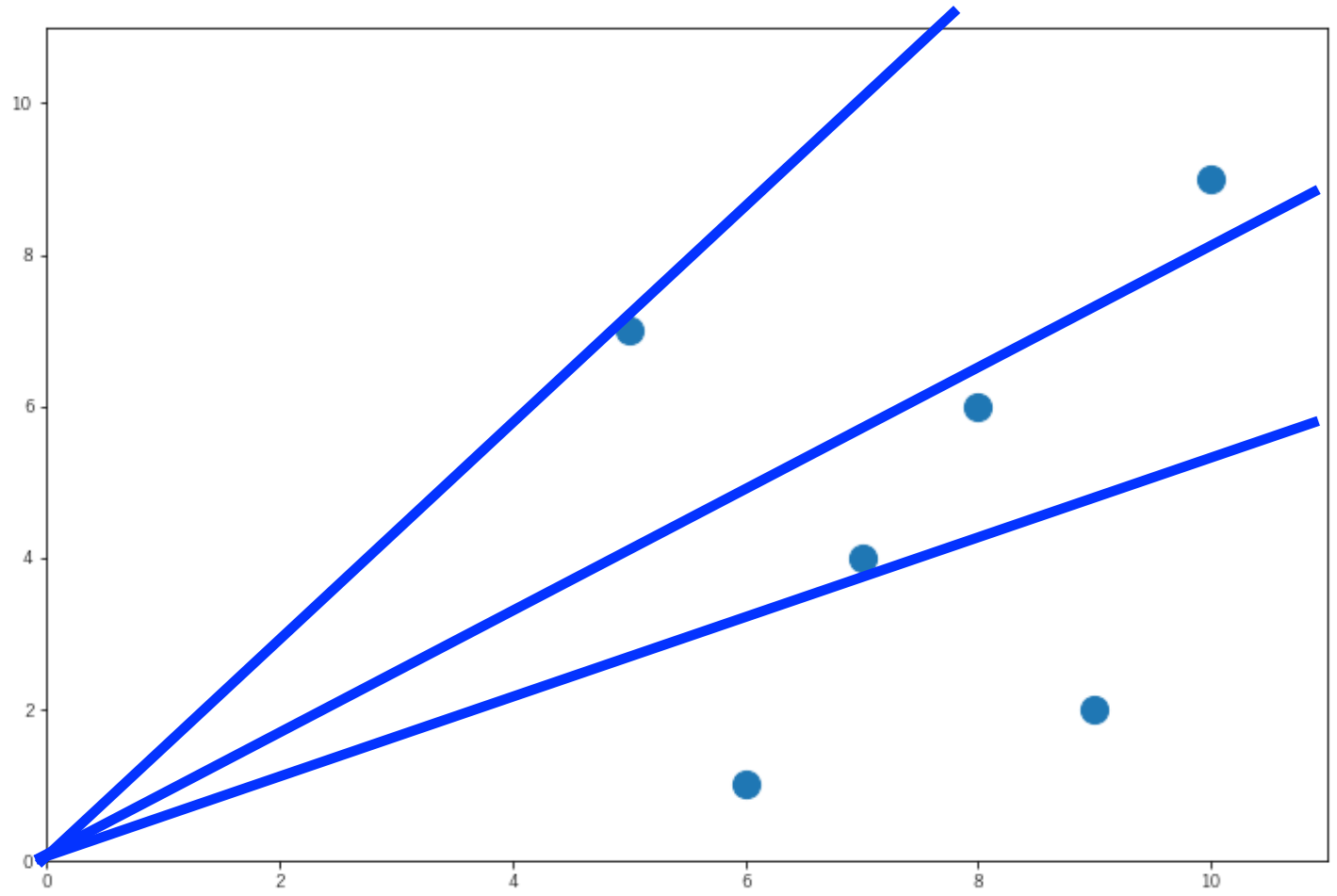
	新たな特徴量
りんご1	13
りんご２	10.1
れもん1	9.8
バナナ1	7.2
バナナ2	7.7
れもん2	6.4

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

この軸はどの様にして決まるのか



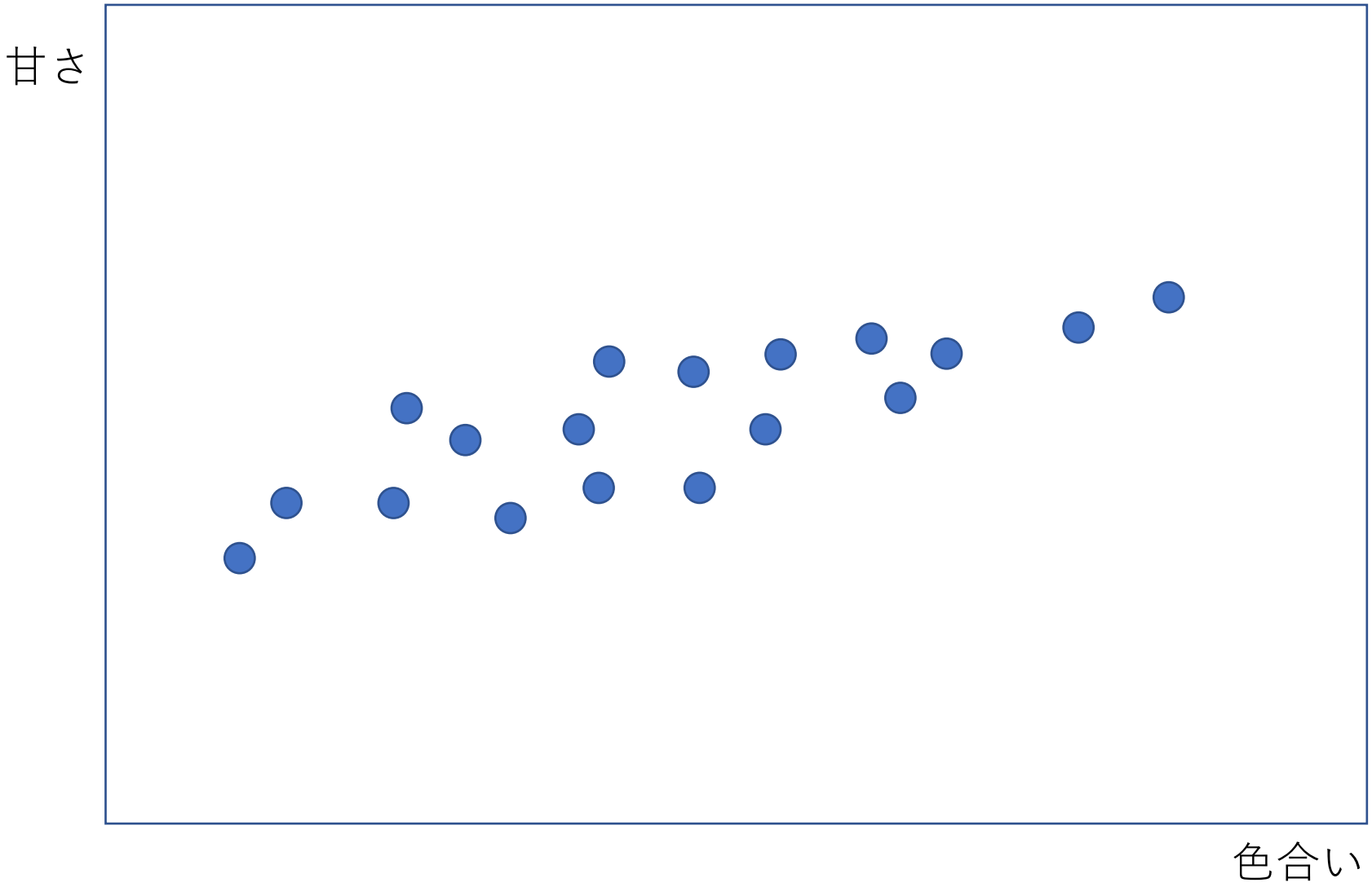
この軸はどの様にして決まるのか



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

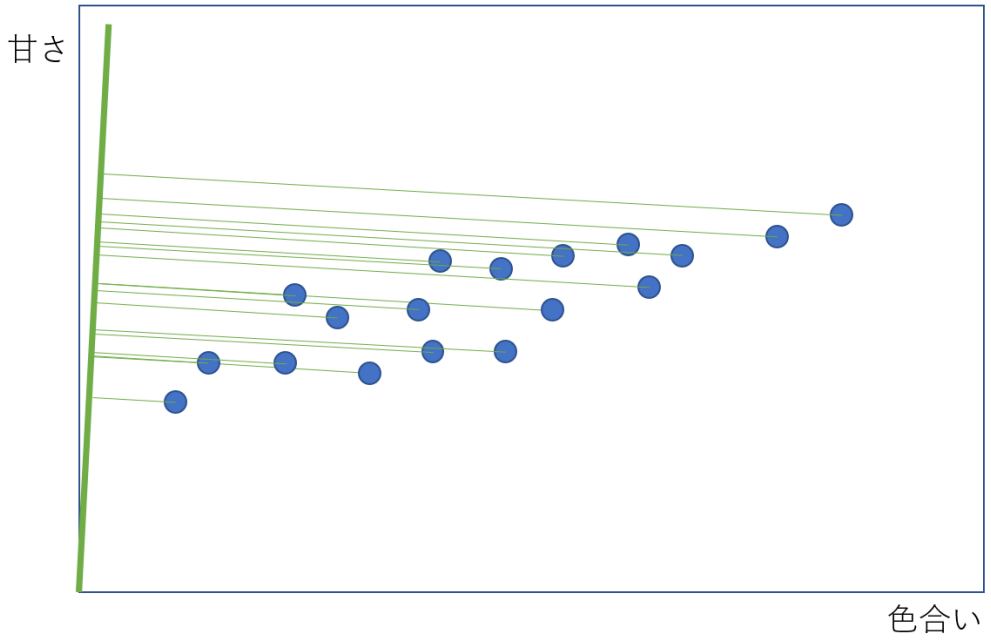
具体的には、データを移した時の分散(ばらつき)が最大になる様な軸を選ぶ

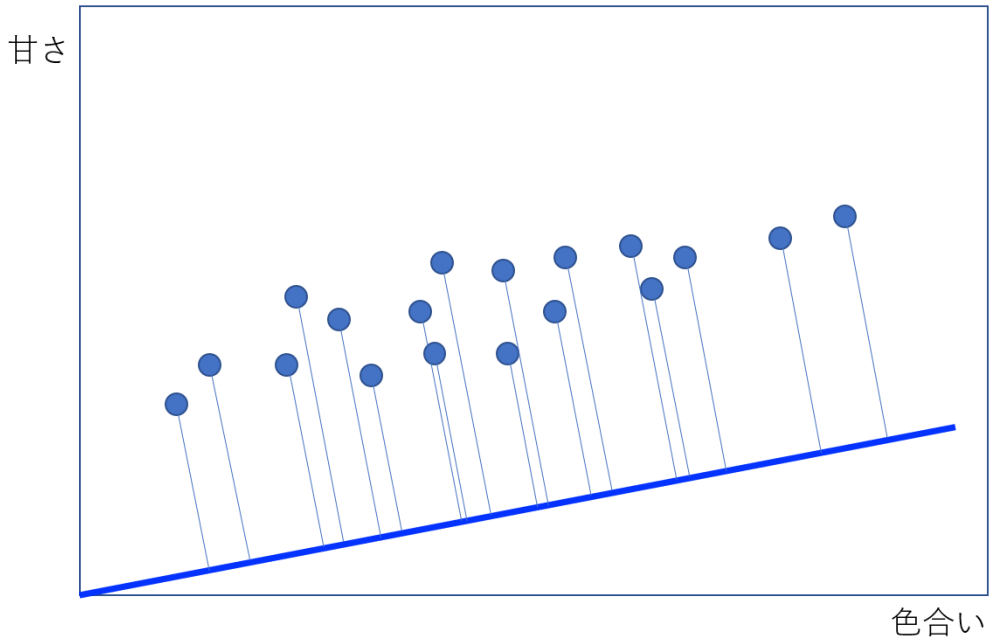
例えば次のようなデータがあったとする



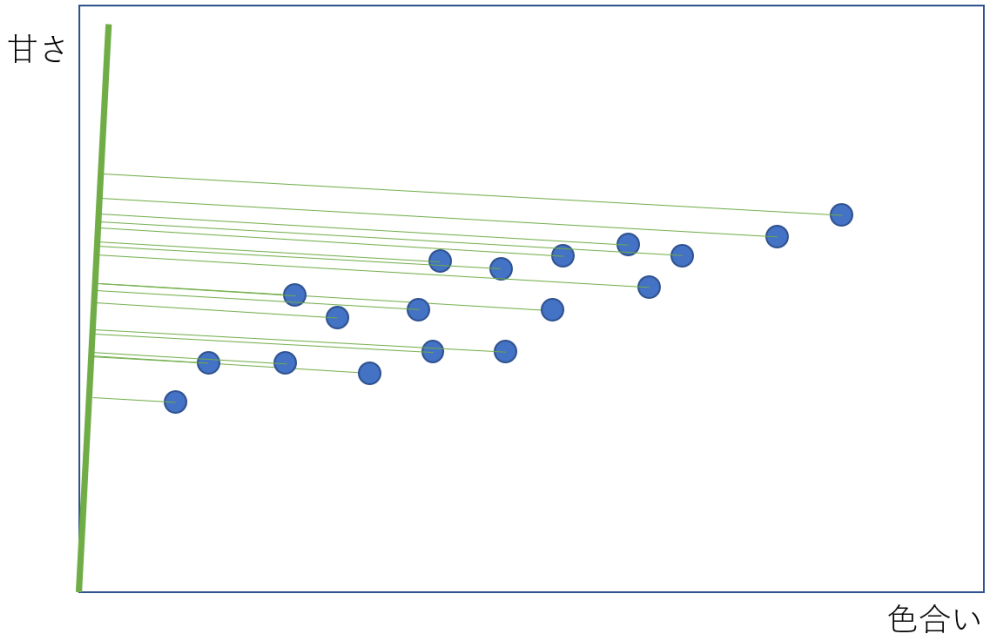
目的は2つの変数(甘さと色合い)から新たな変数を作りたい

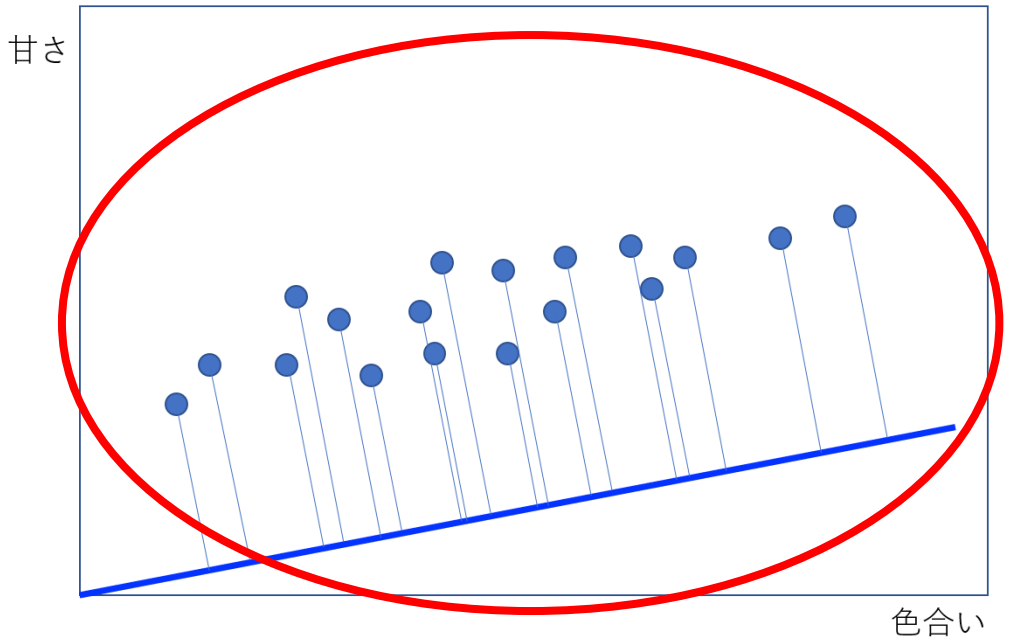
次の2本の軸はどちらの方が甘さと色合いの情報がより保存されているでしょうか？
主成分分析は「データの相関関係」を失わないでデータを圧縮することを考えます。



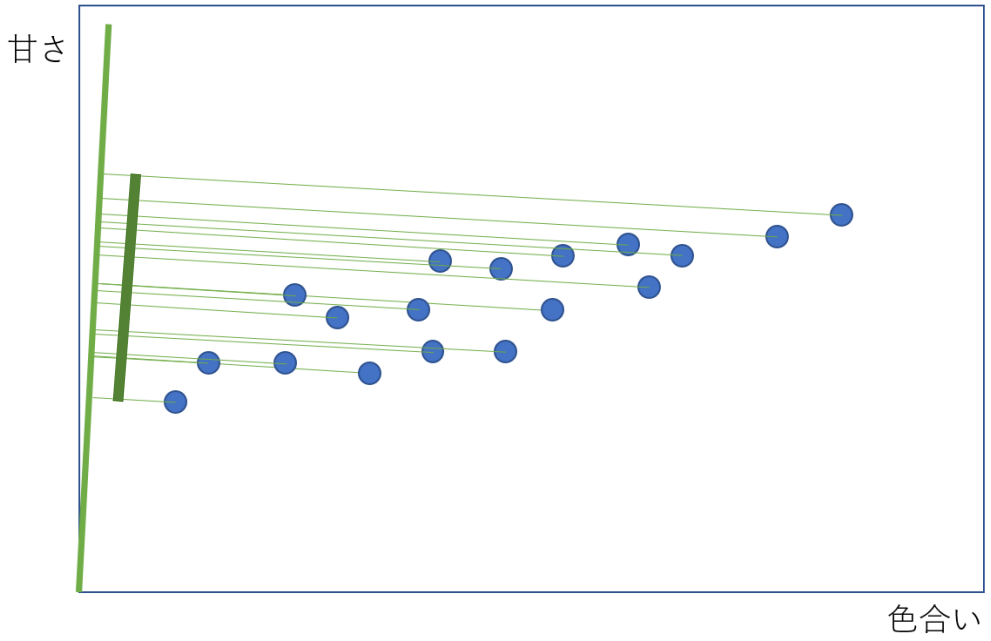


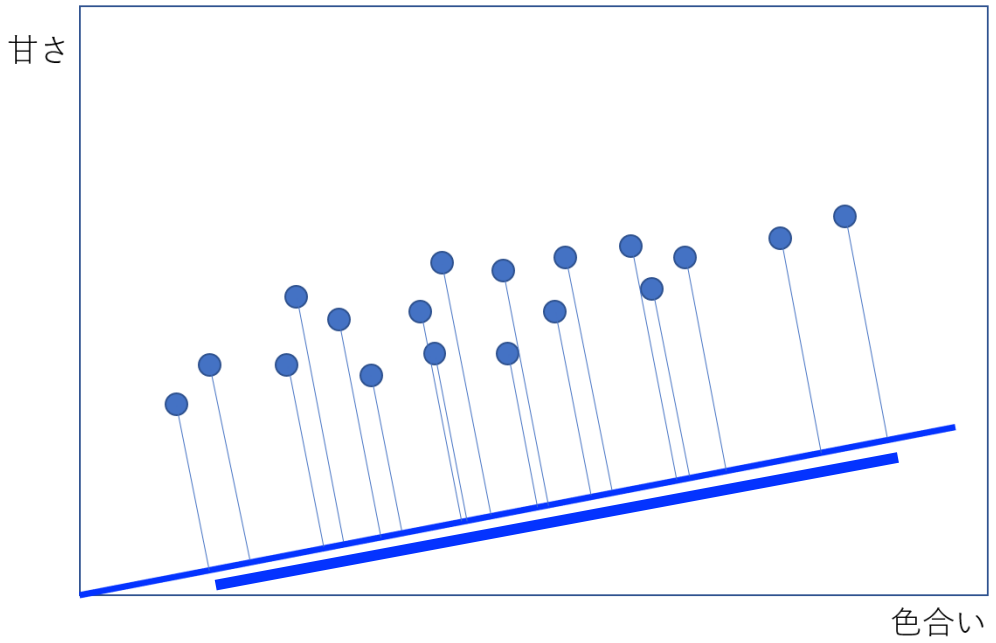
次の2本の軸はどちらの方が甘さと色合いの情報がより保存されているでしょうか？
主成分分析は「データの相関関係」を失わないでデータを圧縮することを考えます。





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。



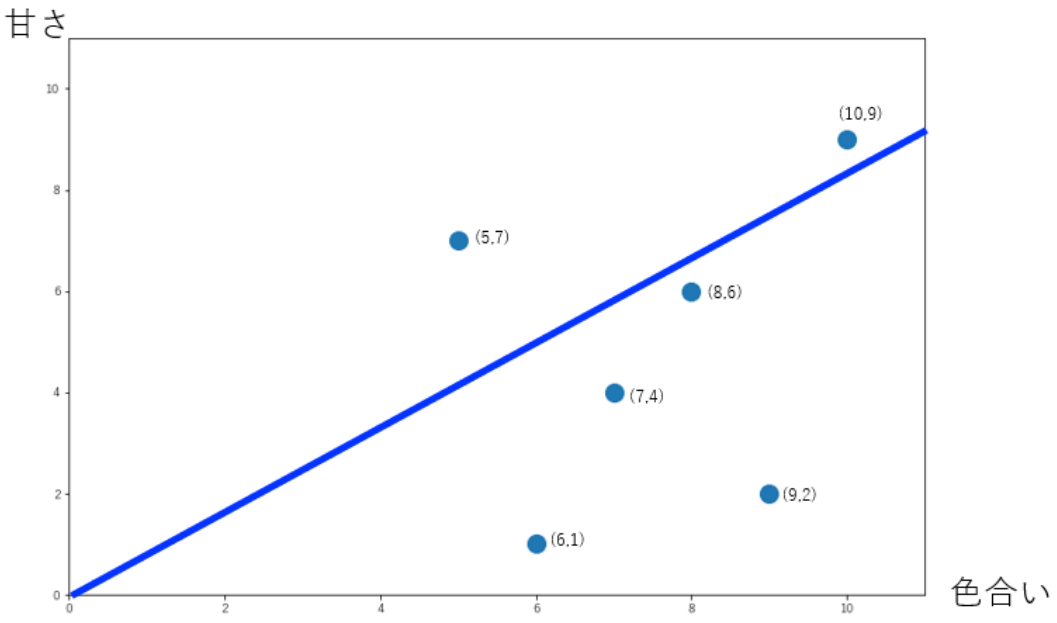


　緑の軸の方がばらつきが小さい
＝似たデータになってしまっている
＝失っている情報が多い
→ばらつき(＝分散)が大きい軸ほどより2つの変数の情報が保存された新しい軸

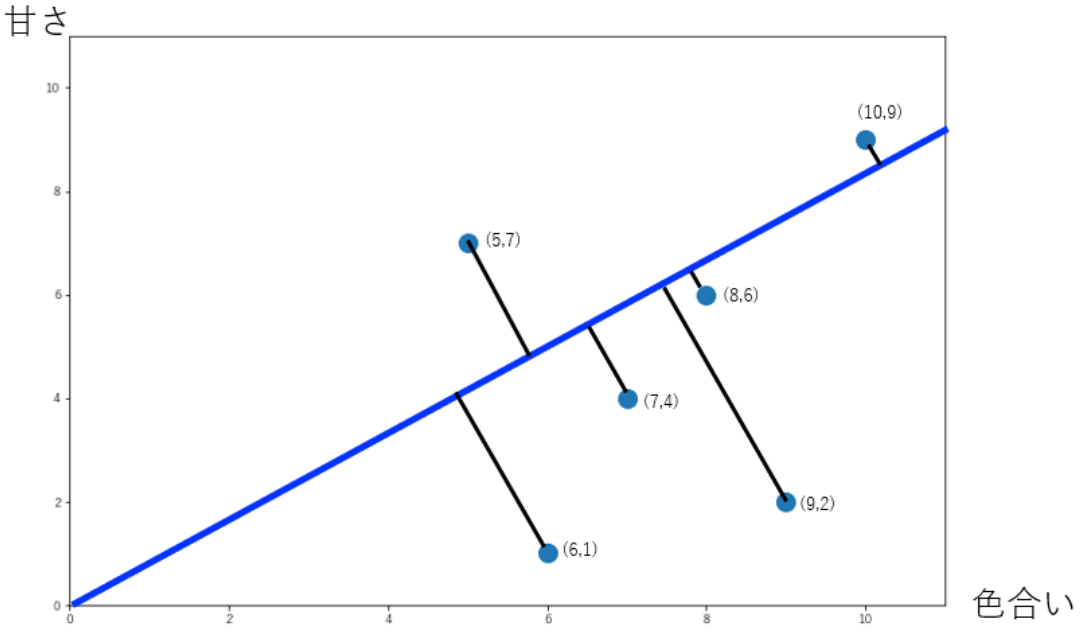
主成分分析(PCA)を実行する

```
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

```
print(type(result))
print(result.shape)
print(result)
```

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

```
In [95]: print(type(result))
<class 'numpy.ndarray'>

In [96]: print(result.shape)
(150, 2)

In [97]: print(result)
[[-2.68412563  0.31939725]
 [-2.71414169 -0.17700123]
 [-2.88899057 -0.14494943]
 [-2.74534286 -0.31829898]
 [-2.72871654  0.32675451]
 [-2.28085963  0.74133045]
 [-2.82053775 -0.08946138]
 [-2.62614497  0.16338496]
 [-2.88638273 -0.57831175]
 [-2.6727558  -0.11377425]
 [-2.50694709  0.6450689 ]
 [-2.61275523  0.01472994]
 [-2.78610927 -0.235112  ]
 [-3.22380374 -0.51139459]
 [-2.64475039  1.17876464]
 [-2.38603903  1.33806233]
 [-2.62352788  0.81067951]
 [-2.64829671  0.31184914]
 [-2.19982032  0.87283904]
 [-2.5879864   0.51356031]
 [-2.31025622  0.39134594]
```

主成分分析(PCA)を実行する

```
print(type(result))
print(result.shape)
print(result)
print(result[:,0])
```

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

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)

```
In [98]: print(result[:,0])
[-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  3.39703874  0.52123224  2.93258707
  2.32122882  2.91675097  1.66177415  1.80340195  2.1655918   1.34616358
  1.58592822  1.90445637  1.94968906  3.48705536  3.79564542  1.30079171
  2.42781791  1.19900111  3.49992004  1.38876613  2.2754305   2.61409047
  1.25850816  1.29113206  2.12360872  2.38800302  2.84167278  3.23067366
  2.15943764  1.44416124  1.78129481  3.07649993  2.14424331  1.90509815
  1.16932634  2.10761114  2.31415471  1.9222678   1.41523588  2.56301338
  2.41874618  1.94410979  1.52716661  1.76434572  1.90094161  1.39018886]
```

主成分分析(PCA)を実行する

元の特徴量(iris.data)
150行4列

```
In [99]: print(iris.data)
[[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]
```

新たな特徴量(result)
150行2列

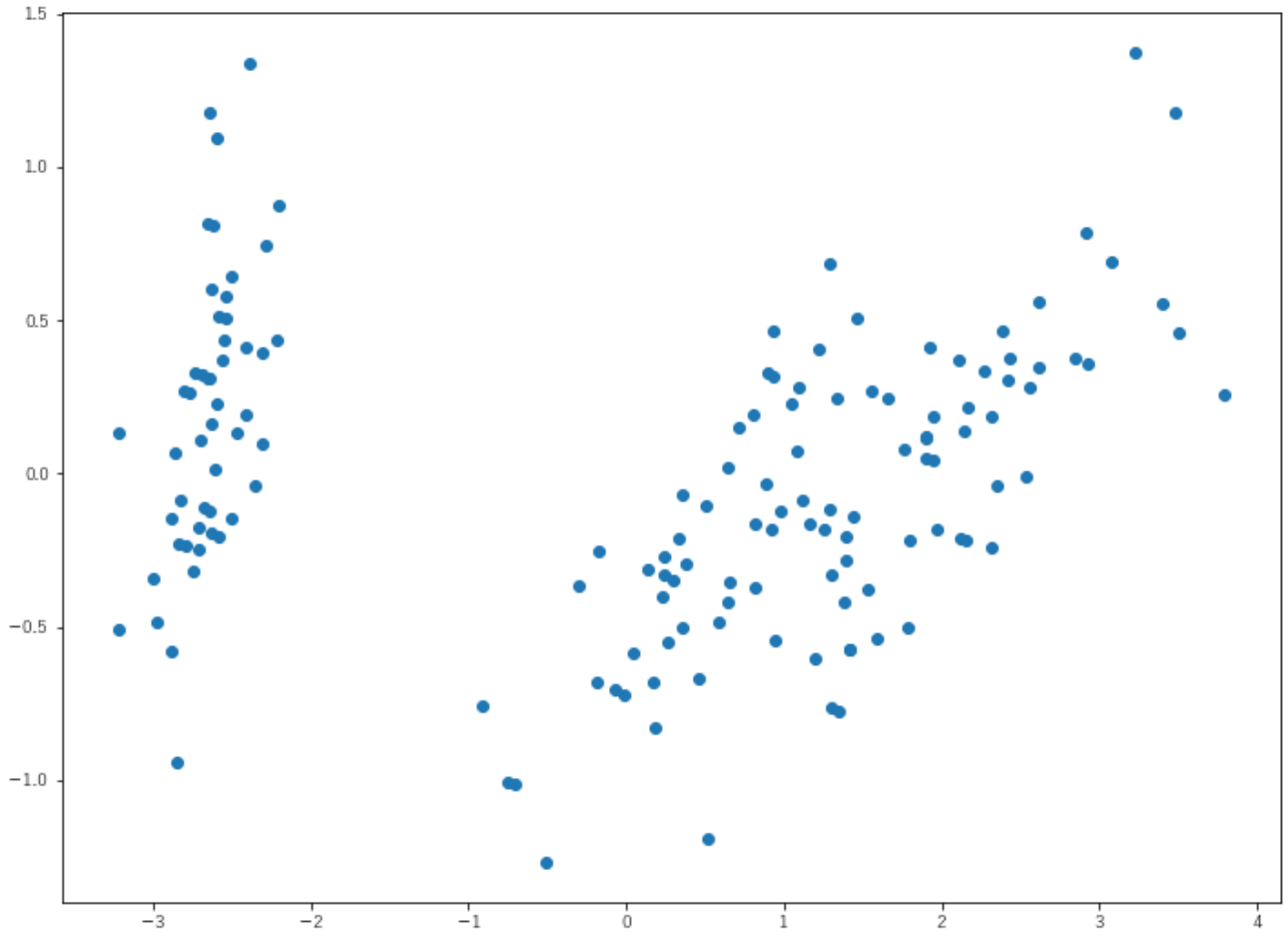
```
In [100]: print(result)
[[-2.68412563  0.31939725]
 [-2.71414169 -0.17700123]
 [-2.88899057 -0.14494943]
 [-2.74534286 -0.31829898]
 [-2.72871654  0.32675451]
 [-2.28085963  0.74133045]
 [-2.82053775 -0.08946138]
 [-2.62614497  0.16338496]
 [-2.88638273 -0.57831175]
 [-2.6727558  -0.11377425]
 [-2.50694709  0.6450689 ]
 [-2.61275523  0.01472994]
```

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.figure(figsize=(12,9))
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

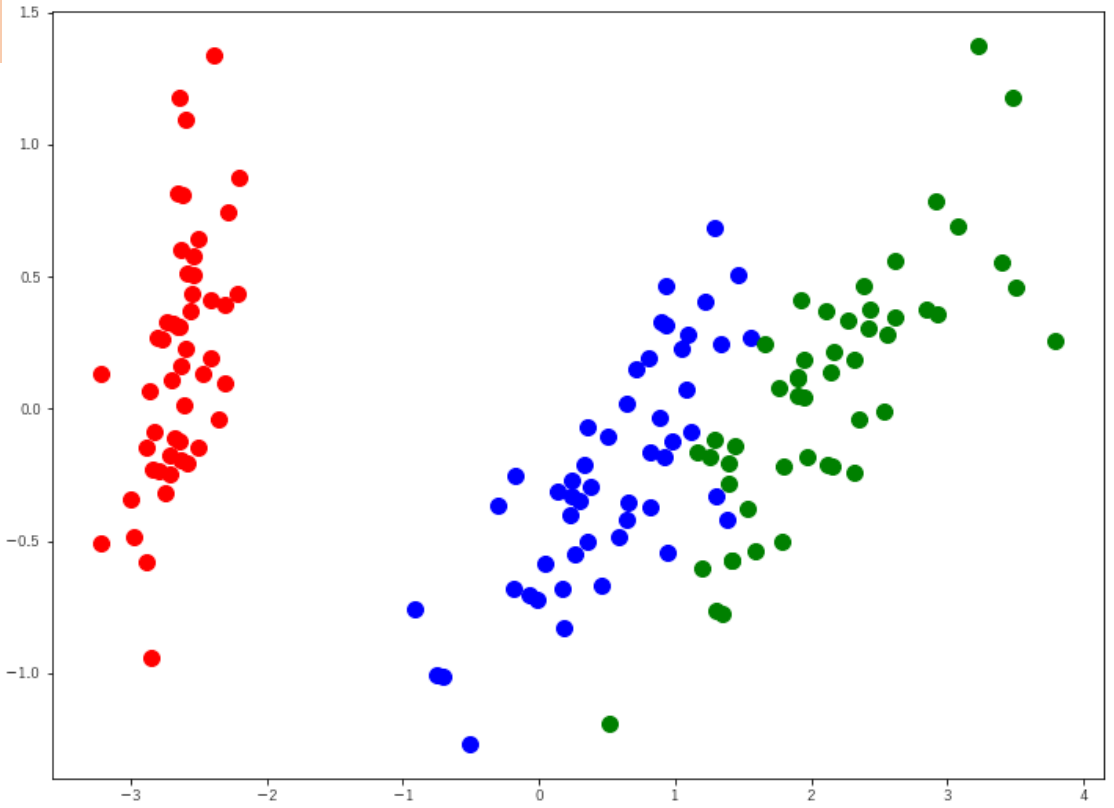
主成分分析(PCA)を実行する（アヤメの種類で分ける）

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12,9))
plt.scatter(result[0:50,0], result[0:50,1], c='red', s=100)
plt.scatter(result[50:100,0], result[50:100,1], c='blue', s=100)
plt.scatter(result[100:150,0], result[100:150,1], c='green', s=100)

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、s=~~は点のサイズを指定

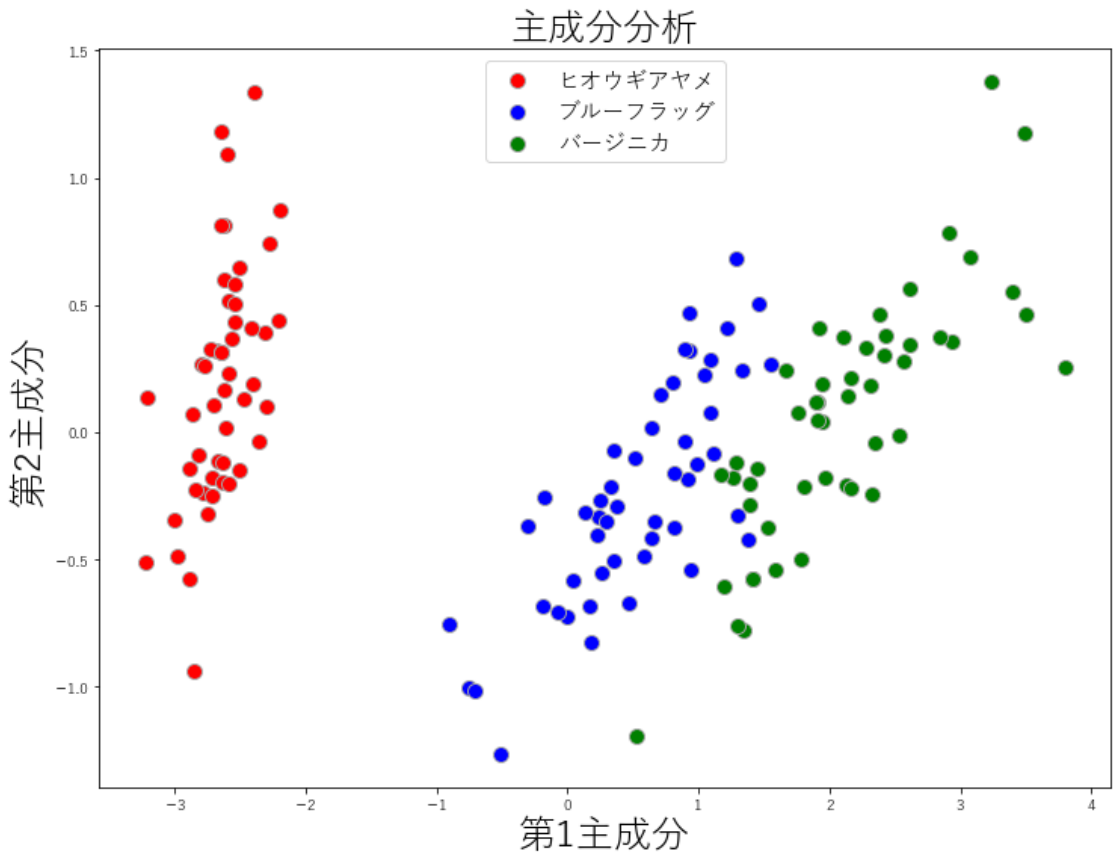


（参考）

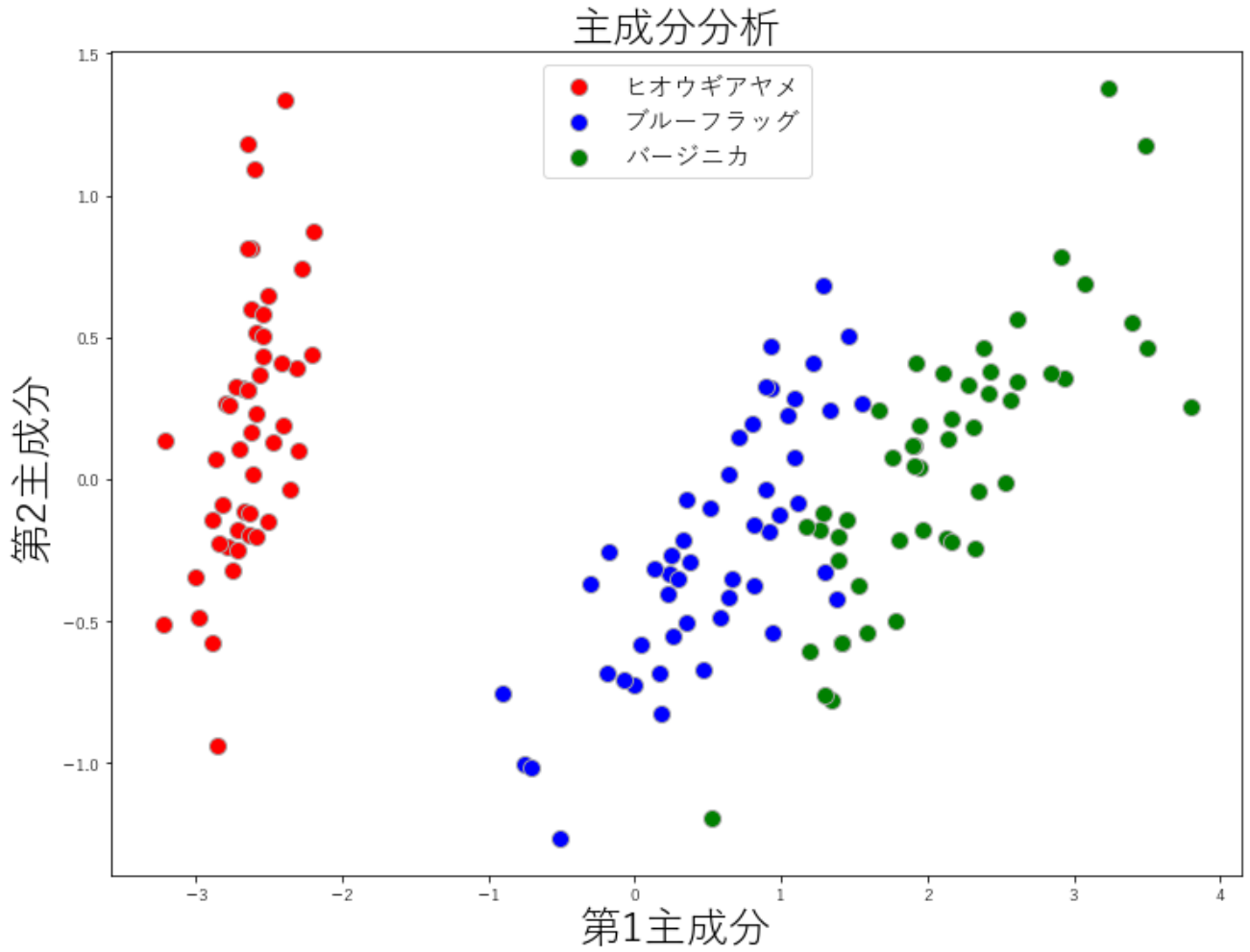
主成分分析(PCA)を実行する

```
import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams['font.family']='sans-serif'
rcParams['font.sans-serif']=['Hiragino Maru Gothic Pro', 'Yu Gothic', 'Meirio']
plt.figure(figsize=(12,9))
plt.scatter(result[0:50,0], result[0:50,1], c='red', label='ヒオウギアヤメ',edgecolor='darkgray', s=100)
plt.scatter(result[50:100,0], result[50:100,1], c='blue', label='ブルーフラッグ',edgecolor='darkgray', s=100)
plt.scatter(result[100:150,0], result[100:150,1], c='green', label='バージニカ',edgecolor='darkgray', s=100)
plt.title('主成分分析',fontsize=25)
plt.xlabel('第1主成分',fontsize=25)
plt.ylabel('第2主成分',fontsize=25)
plt.legend(fontsize=15,loc='upper center')
plt.show()
```

タイトル、x,y軸の名前、点の名前を追加



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）

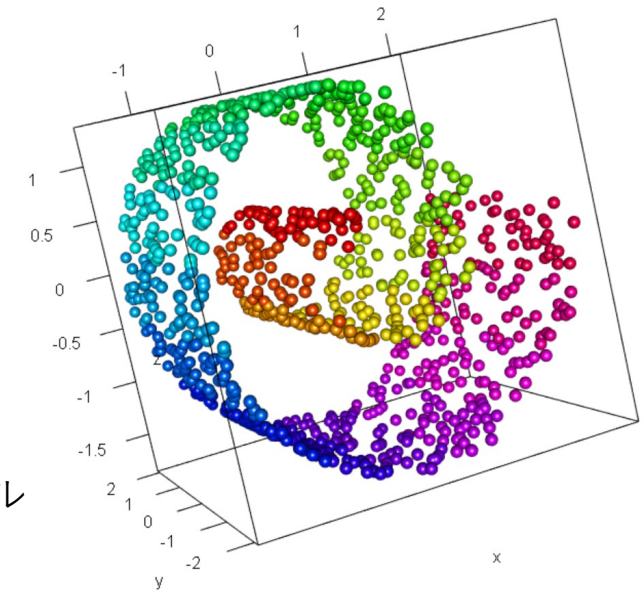
他の次元削減の実施

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

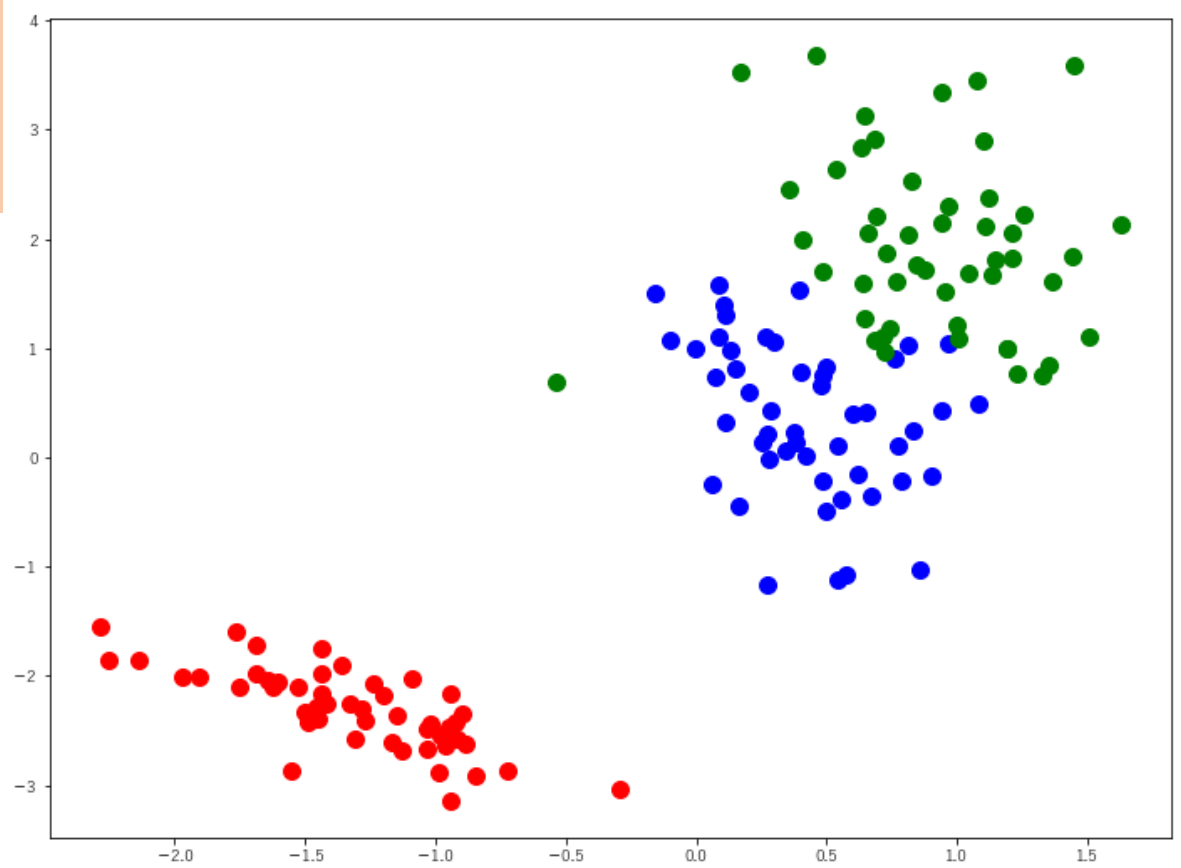
UMAP（紹介のみ）
：tSNEよりもさらに実行速度が速い次元削減手法
　近年ゲノム解析でよく使用されている

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



MDSを実行する

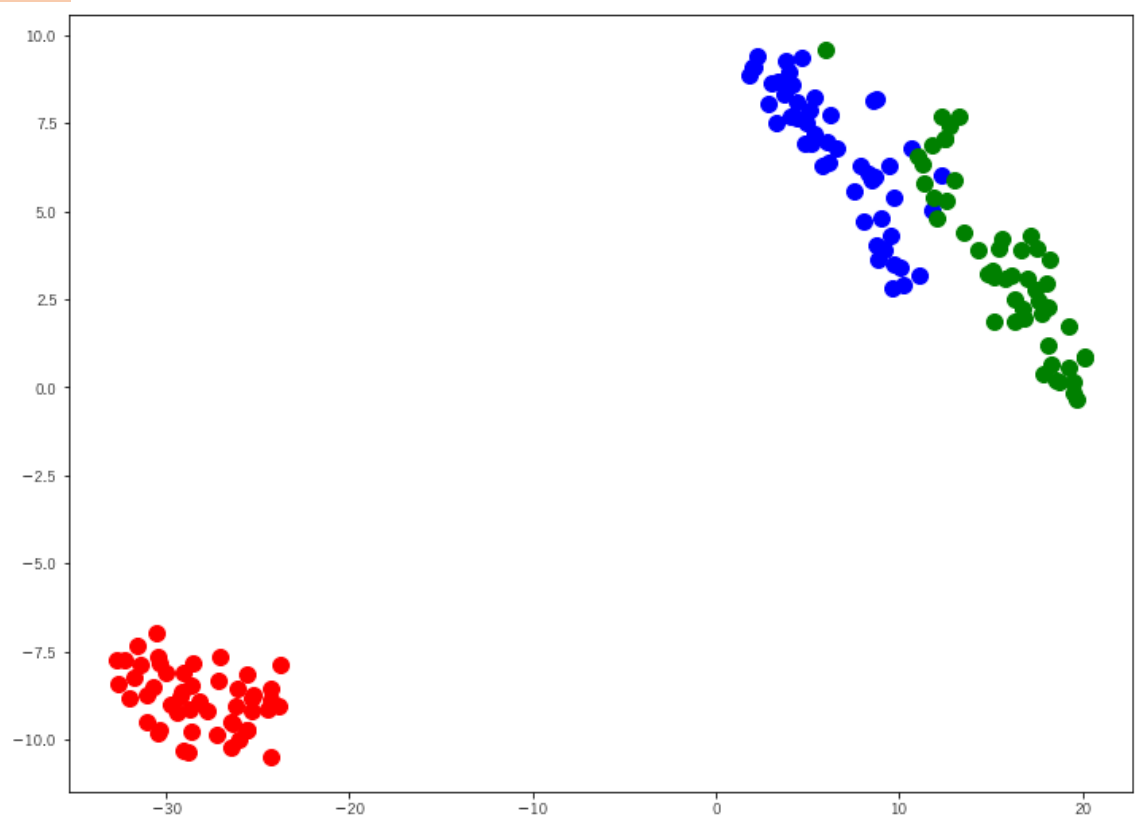
```
from sklearn import manifold

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

plt.figure(figsize=(12,9))
plt.scatter(result[0:50,0], result[0:50,1], c='red', s=100)
plt.scatter(result[50:100,0], result[50:100,1], c='blue', s=100)
plt.scatter(result[100:150,0], result[100:150,1], c='green', s=100)

plt.show()
```



やり方は大きくは変わらない
モデル名をmanifold.MDS()とする

t-SNEを実行する

```
from sklearn.manifold import TSNE

model = TSNE(n_components=2,perplexity=25)
result = model.fit_transform(iris.data)

plt.figure(figsize=(12,9))
plt.scatter(result[0:50,0], result[0:50,1], c='red', s=100)
plt.scatter(result[50:100,0], result[50:100,1], c='blue', s=100)
plt.scatter(result[100:150,0], result[100:150,1], c='green', s=100)

plt.show()
```



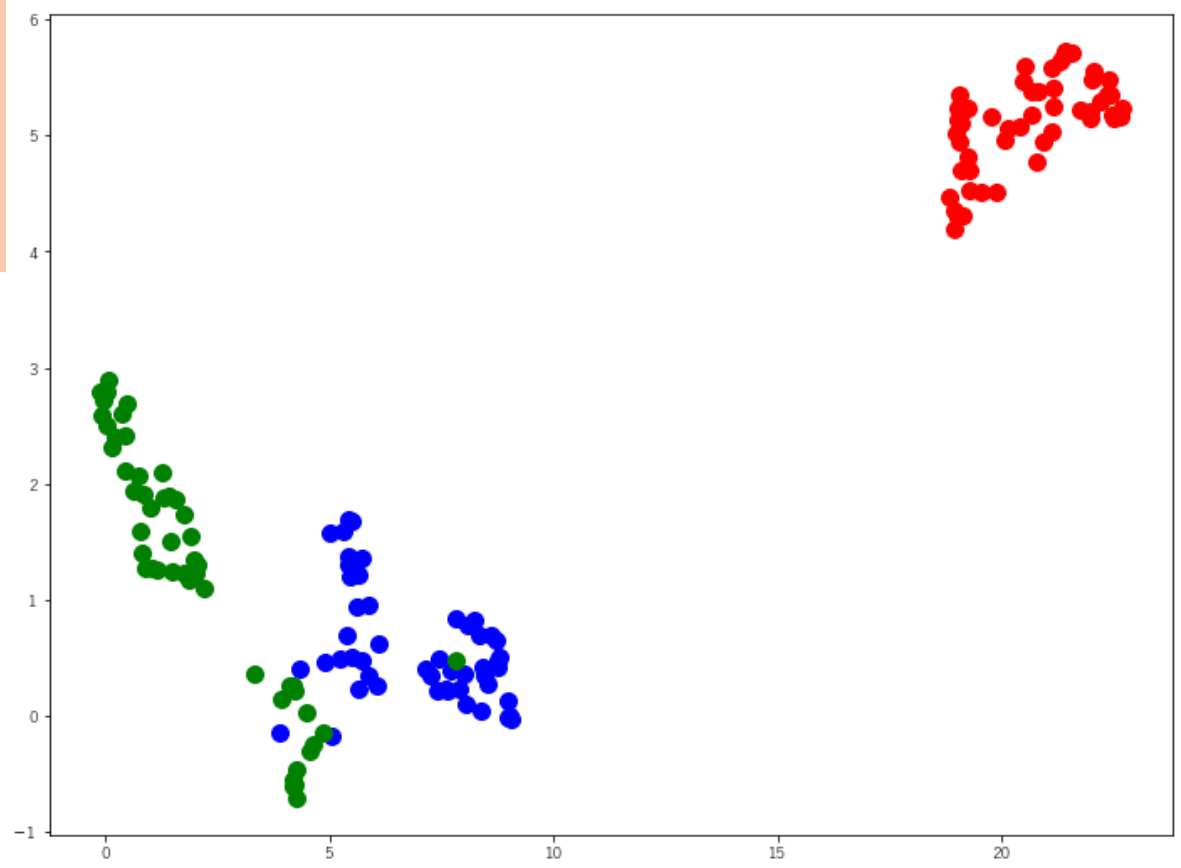
やり方は大きくは変わらない
モデル名をTSNE()とする

perplexityはどれだけ近くの点を考慮するかの指標
一般的に5~50程度にする

(参考)UMAPを実行する

(macの人はumap-learnというライブラリのインストールが必要)

```
import umap
# Conda install, conda install -c conda-forge umap-learn

model = umap.UMAP(n_neighbors=15)
result = model.fit_transform(iris.data)

plt.figure(figsize=(12,9))
plt.scatter(result[0:50,0], result[0:50,1], c='red', s=100)
plt.scatter(result[50:100,0], result[50:100,1], c='blue', s=100)
plt.scatter(result[100:150,0], result[100:150,1], c='green', s=100)

plt.show()
```



やり方は大きくは変わらない
モデル名をumap.UMAP()とする

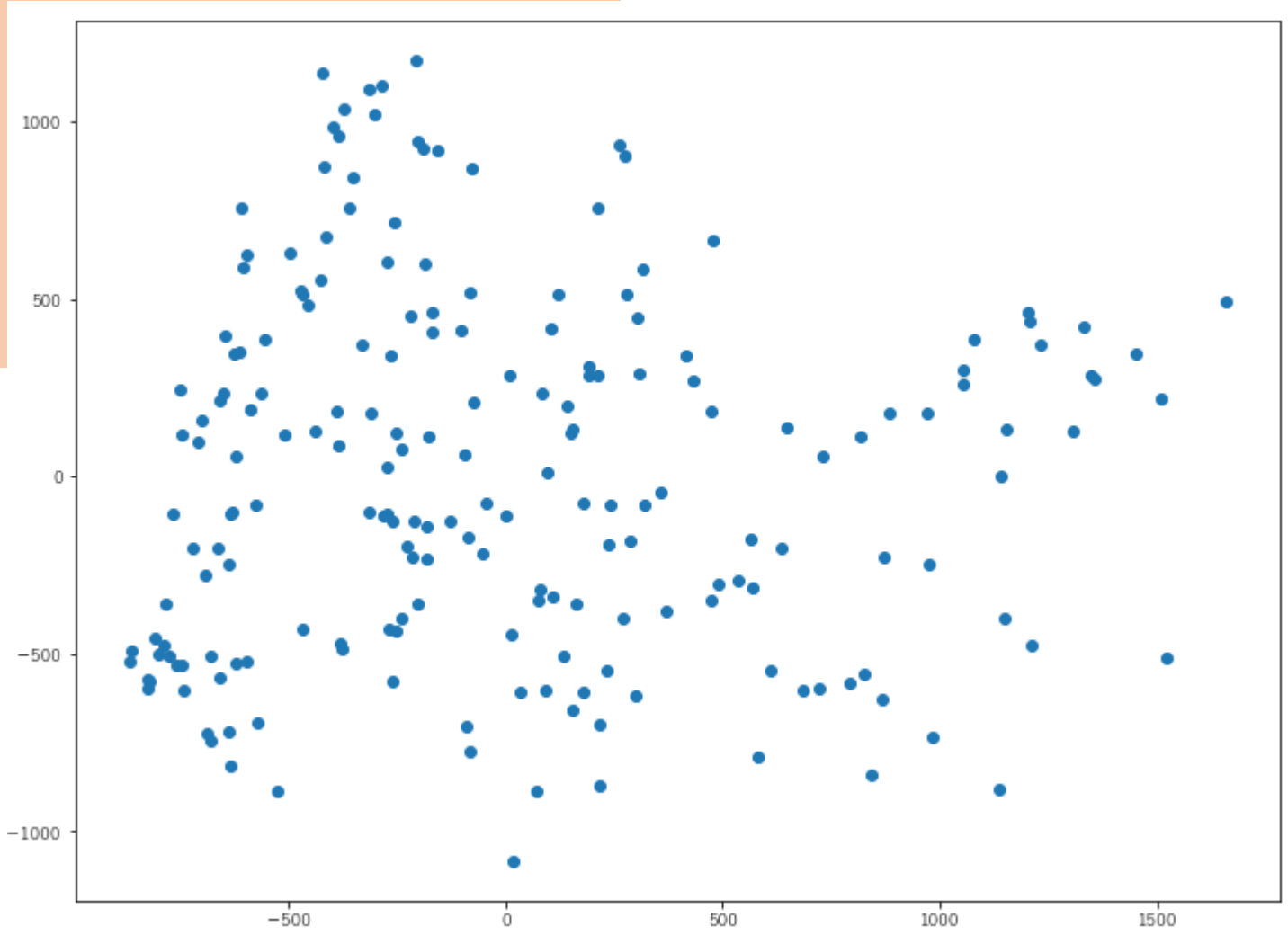
n_neighborsがtSNEのperplexityに似たパラメータで、
2~100の値が推奨されており、デフォルトが15

mnistのデータでやってみよう

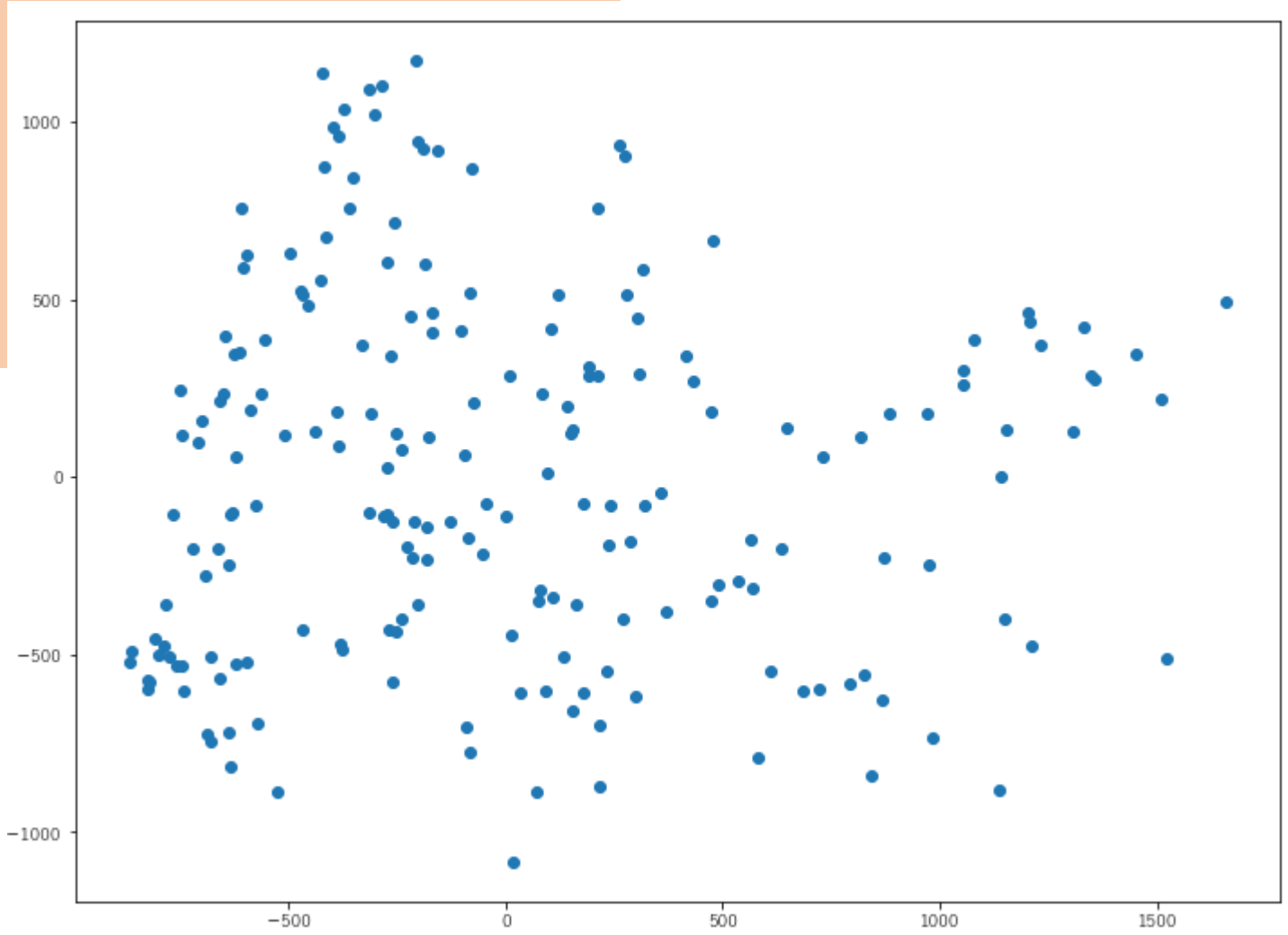
```
from tensorflow.keras.datasets import mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(60000,784)
data = x_train[0:200]
```

x_trainのデータを(60000,28,28)を(60000,784)に変換して先頭200個を取り出します

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

model = PCA(n_components=2)
result = model.fit_transform(data)

plt.figure(figsize=(12,9))
plt.scatter(result[:,0],result[:,1])

plt.show()
```



```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

model = PCA(n_components=2)
result = model.fit_transform(data)

plt.figure(figsize=(12,9))
plt.scatter(result[:,0],result[:,1])

plt.show()
```



数字ごとに色を変えたい

plt.scatter(result[:,0],result[:,1])

全ての行ではなく、
ラベル0のみの行、
ラベル1のみの行、、、にしたい

dataはx_trainの先頭200行

```
print(data.shape)
```

```
In [6]: print(data.shape)
(200, 784)
```

resultはdata使って次元削減(784→2次元)

```
print(result.shape)
```

```
In [7]: print(result.shape)
(200, 2)
```

y_train[0:200]はdataに対応する正解200個

```
print(y_train[0:200].shape)
```

```
In [8]: print(y_train[0:200].shape)
(200,)
```

dataはx_trainの先頭200行

```
print(data.shape)
```

```
In [6]: print(data.shape)
(200, 784)
```

resultはdata使って次元削減(784→2次元)

```
print(result.shape)
```

```
In [7]: print(result.shape)
(200, 2)
```

y_train[0:200]はdataに対応する正解200個

```
print(y_train[0:200].shape)
```

```
In [8]: print(y_train[0:200].shape)
(200,)
```

200行の中で正解ラベルの数字が0の行は

```
list0 = [i for i in range(0,200) if y_train[i] == 0]
```

dataはx_trainの先頭200行

```
print(data.shape)
```

```
In [6]: print(data.shape)
(200, 784)
```

resultはdata使って次元削減(784→2次元)

```
print(result.shape)
```

```
In [7]: print(result.shape)
(200, 2)
```

y_train[0:200]はdataに対応する正解200個

```
print(y_train[0:200].shape)
```

```
In [8]: print(y_train[0:200].shape)
(200,)
```

200行の中で正解ラベルの数字が0の行は

list0 = [i for i in range(0,200) if y_train[i] == 0]

[処理内容 for 変数 in 連続したデータ if 条件式]

dataはx_trainの先頭200行

```
print(data.shape)
```

```
In [6]: print(data.shape)
(200, 784)
```

resultはdata使って次元削減(784→2次元)

```
print(result.shape)
```

```
In [7]: print(result.shape)
(200, 2)
```

y_train[0:200]はdataに対応する正解200個

```
print(y_train[0:200].shape)
```

```
In [8]: print(y_train[0:200].shape)
(200,)
```

200行の中で正解ラベルの数字が0の行は

list0 = [i for i in range(0,200) if y_train[i] == 0]

[処理内容 for 変数 in 連続したデータ if 条件式]

0から199(200未満)を順にiに代入し、

dataはx_trainの先頭200行

```
print(data.shape)
```

```
In [6]: print(data.shape)
(200, 784)
```

resultはdata使って次元削減(784→2次元)

```
print(result.shape)
```

```
In [7]: print(result.shape)
(200, 2)
```

y_train[0:200]はdataに対応する正解200個

```
print(y_train[0:200].shape)
```

```
In [8]: print(y_train[0:200].shape)
(200,)
```

200行の中で正解ラベルの数字が0の行は

list0 = [i for i in range(0,200) if y_train[i] == 0]

[処理内容 for 変数 in 連続したデータ if 条件式]

0から199(200未満)を順にiに代入し、

もしiが y_train[i] == 0 を満たすなら、

dataはx_trainの先頭200行

```
print(data.shape)
```

```
In [6]: print(data.shape)
(200, 784)
```

resultはdata使って次元削減(784→2次元)

```
print(result.shape)
```

```
In [7]: print(result.shape)
(200, 2)
```

y_train[0:200]はdataに対応する正解200個

```
print(y_train[0:200].shape)
```

```
In [8]: print(y_train[0:200].shape)
(200,)
```

200行の中で正解ラベルの数字が0の行は

list0 = [i for i in range(0,200) if y_train[i] == 0]

[処理内容 for 変数 in 連続したデータ if 条件式]

0から199(200未満)を順にiに代入し、

もしiが y_train[i] == 0 を満たすなら、

iをリストに加える

dataはx_trainの先頭200行

```
print(data.shape)
```

```
In [6]: print(data.shape)
(200, 784)
```

resultはdata使って次元削減(784→2次元)

```
print(result.shape)
```

```
In [7]: print(result.shape)
(200, 2)
```

y_train[0:200]はdataに対応する正解200個

```
print(y_train[0:200].shape)
```

```
In [8]: print(y_train[0:200].shape)
(200,)
```

200行の中で正解ラベルの数字が0の行は

```
list0 = [i for i in range(0,200) if y_train[i] == 0]
```

[処理内容 for 変数 in 連続したデータ if 条件式]

0から199(200未満)を順にiに代入し、
もしiが y_train[i] == 0 を満たすなら、
iをリストに加える

例)

```
test = [i+2 for i in range(0,5) if i > 1]
print(test)
```

```
In [104]: print(test)
[4, 5, 6]
```

0から4を順にiに代入し、もしiが1より大きいならi+2をリストに加える

```
list0 = [i for i in range(0,200) if y_train[i] == 0]
print(list0)
```

y_train[i] == 0を満たす
行番号のリスト

```
In [107]: print(list0)
[1, 21, 34, 37, 51, 56, 63, 68, 69, 75, 81, 88, 95, 108, 114, 118, 119, 121,
156, 169, 192]
```

```
list0 = [i for i in range(0,200) if y_train[i] == 0]
print(list0)
```

y_train[i] == 0を満たす
行番号のリスト

```
In [107]: print(list0)
[1, 21, 34, 37, 51, 56, 63, 68, 69, 75, 81, 88, 95, 108, 114, 118, 119, 121,
156, 169, 192]
```

```
list0x = [result[i,0] for i in list0]
list0y = [result[i,1] for i in list0]
print(list0x)
print(list0y)
```

list0を順にiに代入し、
(条件は無しで)
result[i,0]をリストに加える

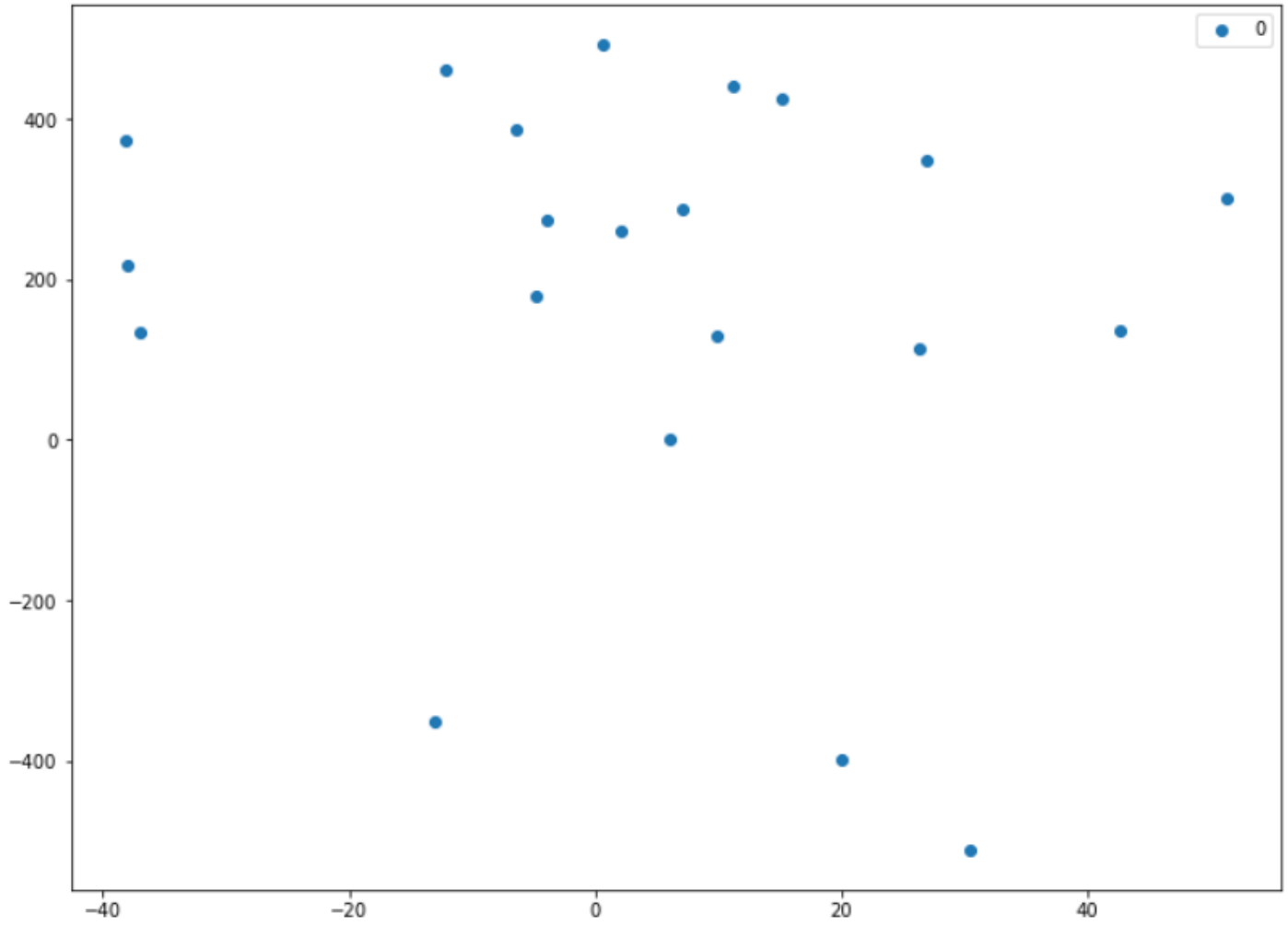
result[i,0]は主成分1の値
result[i,1]は主成分2の値

```
In [111]: print(list0x)
[51.332474, 2.1552584, -4.7430816, -6.370416, 0.71120286, 11.277737,
-3.9621105, 20.11414, -38.018253, -36.952633, 7.03827, 6.1027784, 26.887995,
-13.001846, -12.093183, 26.346466, 15.243017, 30.470387, 42.712387, 9.910969,
-38.13865]

In [112]: print(list0y)
[300.4944002283623, 260.78518204532946, 177.82982829804016,
387.0684566254843, 492.66122212907743, 439.5157792625104, 272.9074953623049,
-399.6526036019192, 217.53204002756158, 133.36910057443333,
286.05742729562115, 1.6411752193475335, 346.801085883006, -350.4735967296917,
461.61778199614577, 112.59004181828587, 424.1843688838097,
-510.4544584221686, 136.5182270893607, 128.1959395730891, 373.413531142822]
```

このlist0x、list0y正解ラベル0の主成分１と２のデータなので図示

```
plt.figure(figsize=(12,9))
plt.scatter(list0x,list0y,label="0")
plt.legend()
plt.show()
```

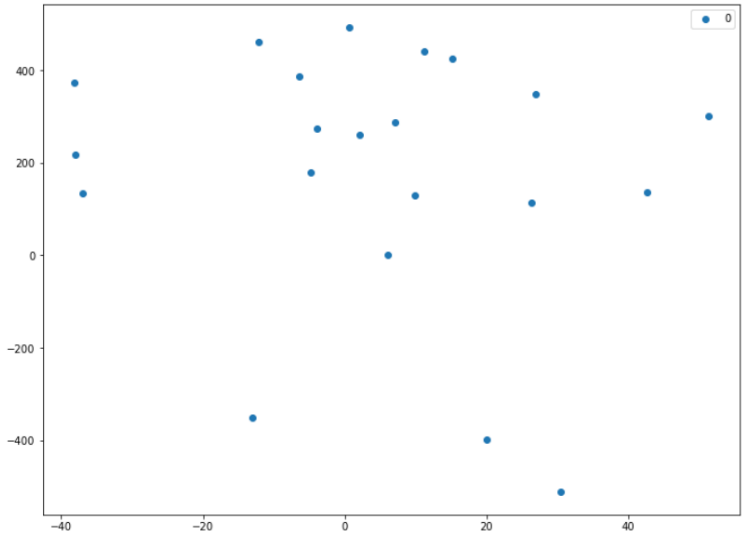
先頭200行のうち、
正解ラベルが0の散布図



これを0から9まで繰り返す

先頭200行のうち、
正解ラベルが0の散布図

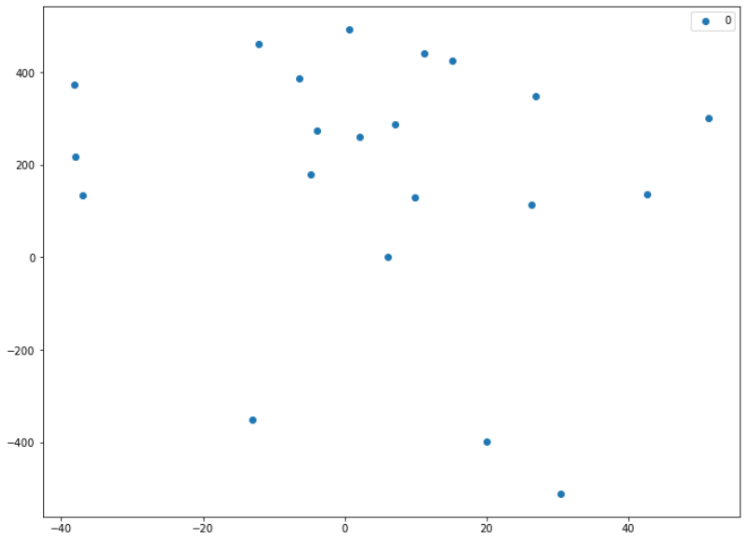
```
plt.figure(figsize=(12,9))
list0 = [i for i in range(0,200) if y_train[i] == 0]
list0x = [result[i,0] for i in list0]
list0y = [result[i,1] for i in list0]
plt.scatter(list0x,list0y,label="0")
plt.legend()
plt.show()
```



これを0から9まで繰り返す

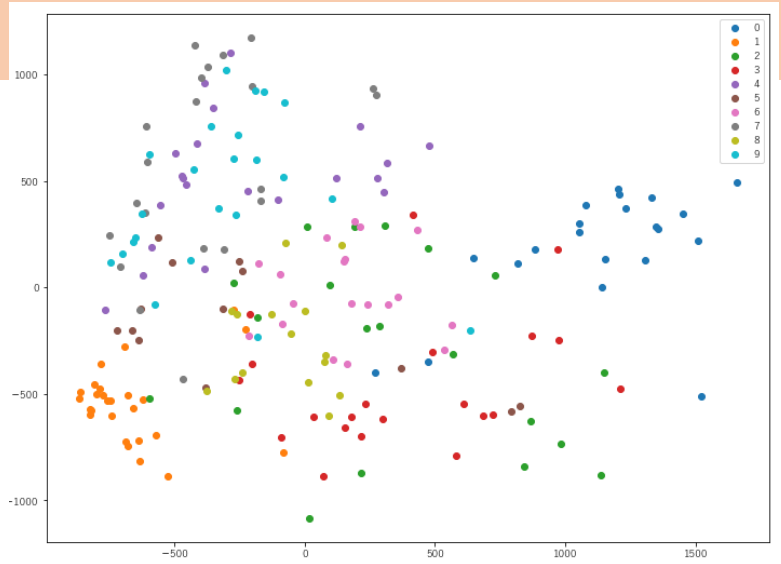
先頭200行のうち、
正解ラベルが0の散布図

```
plt.figure(figsize=(12,9))
list0 = [i for i in range(0,200) if y_train[i] == 0]
list0x = [result[i,0] for i in list0]
list0y = [result[i,1] for i in list0]
plt.scatter(list0x,list0y,label="0")
plt.legend()
plt.show()
```

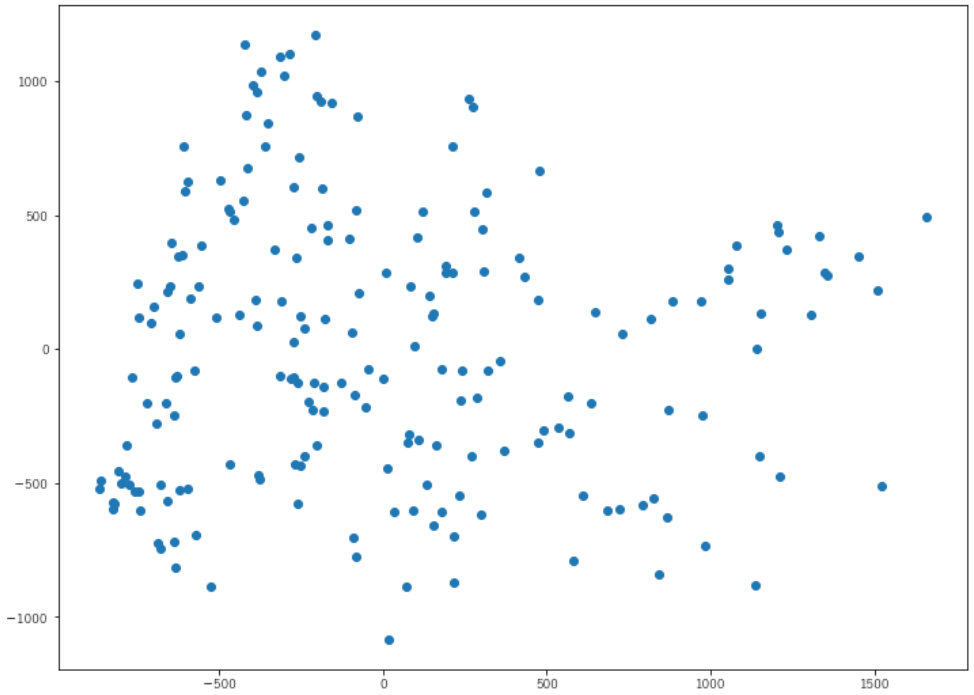


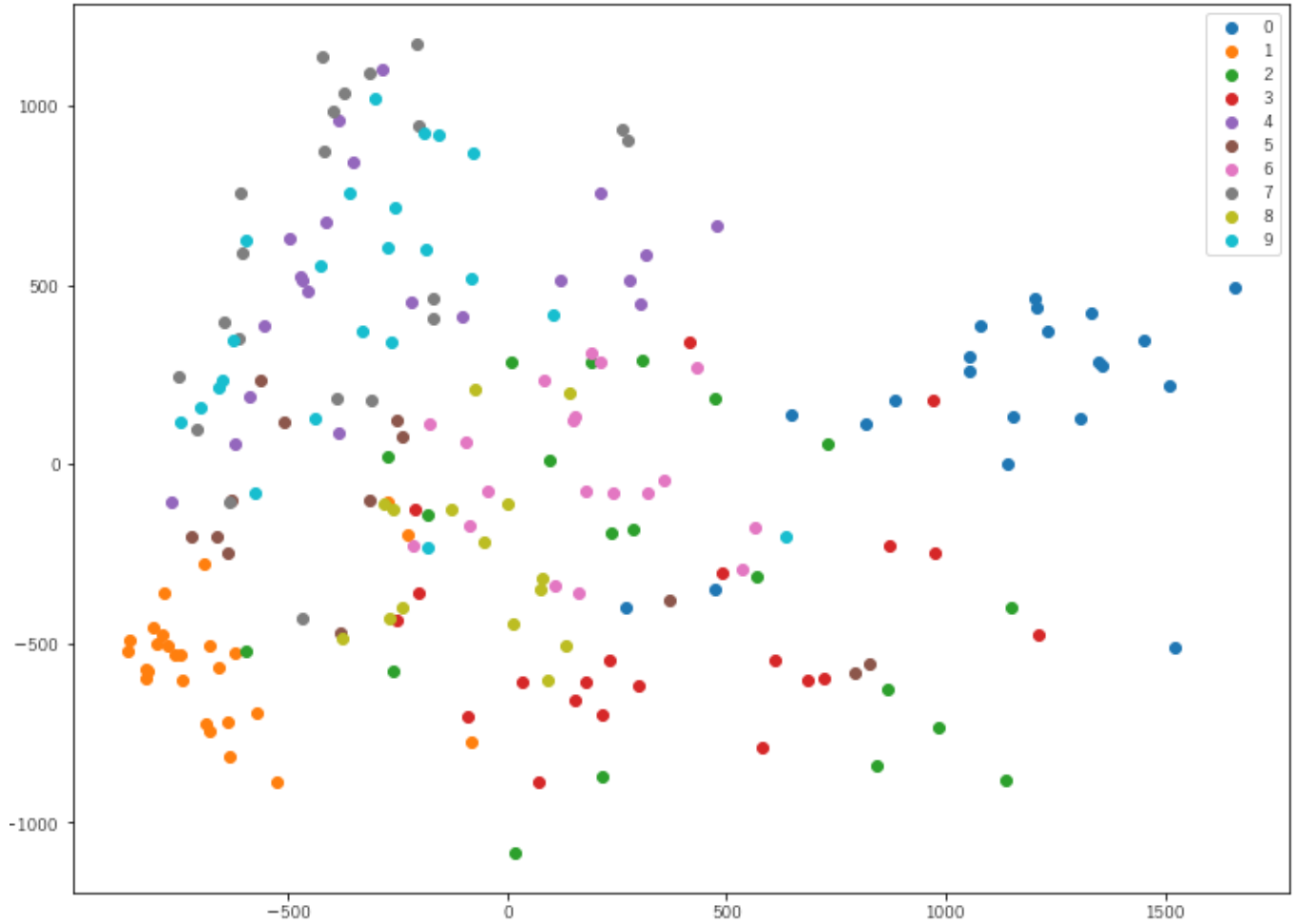
先頭200行の正解ラベルで色分けした散布図

```
plt.figure(figsize=(12,9))
for j in range(0,10):
    list0 = [i for i in range(0,200) if y_train[i] == j]
    list0x = [result[i,0] for i in list0]
    list0y = [result[i,1] for i in list0]
    plt.scatter(list0x,list0y,label=f"{j}")
plt.legend()
plt.show()
```



主成分分析では、あまりきれいに
分かれていないことが分かる





TSNEでも確認

違うのはモデル選択の1行のみ

```
model = PCA(n_components=2)
```

```
model = TSNE(n_components=2,perplexity=15)
```

作図はそのまま

```
plt.figure(figsize=(12,9))
for j in range(0,10):
    list0 = [i for i in range(0,200) if y_train[i] == j]
    list0x = [result[i,0] for i in list0]
    list0y = [result[i,1] for i in list0]
    plt.scatter(list0x,list0y,label=f"{j}")
plt.legend()
plt.show()
```

TSNEでも確認

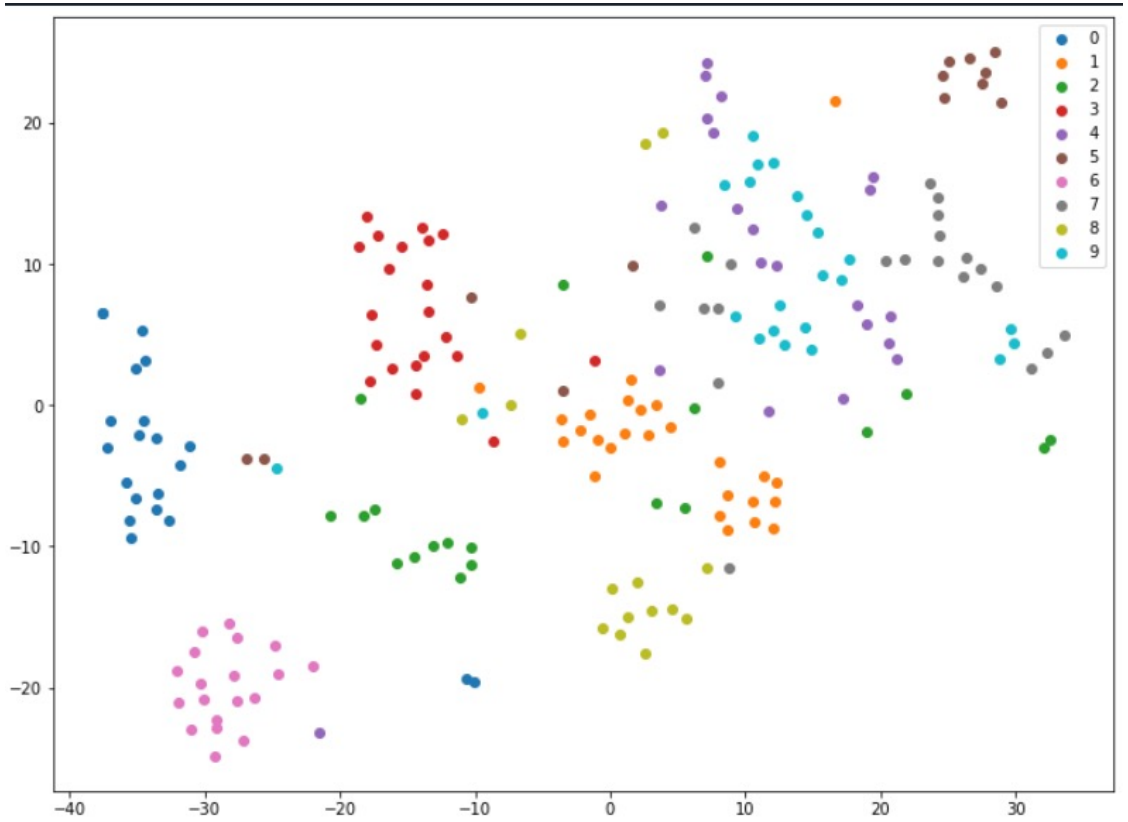
違うのはモデル選択の1行のみ

```
model = PCA(n_components=2)
```

```
model = TSNE(n_components=2,perplexity=15)
```

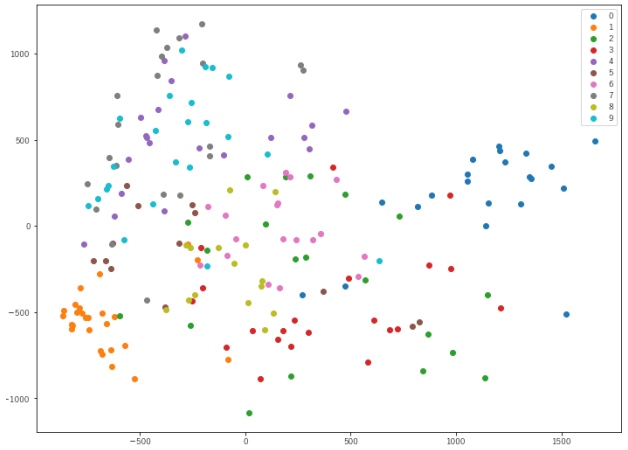
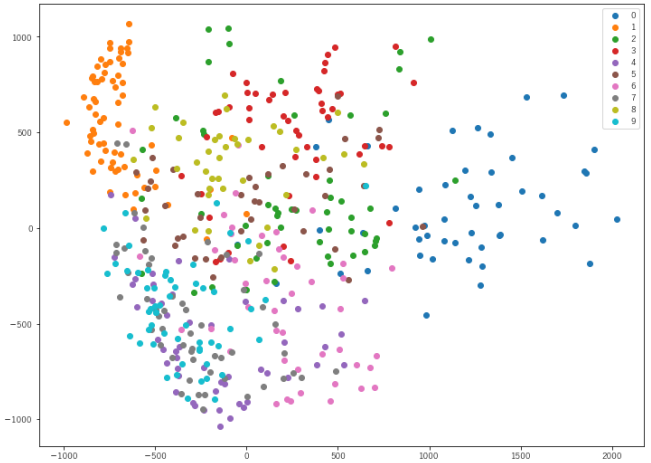
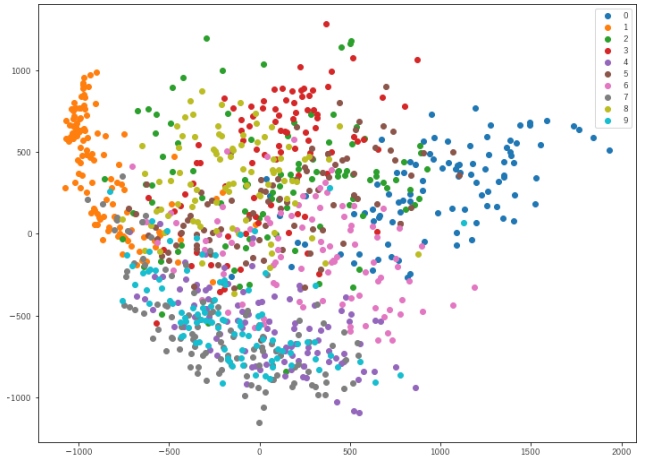
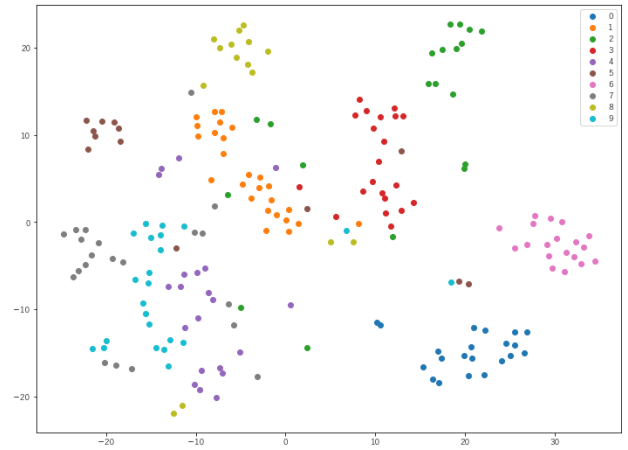
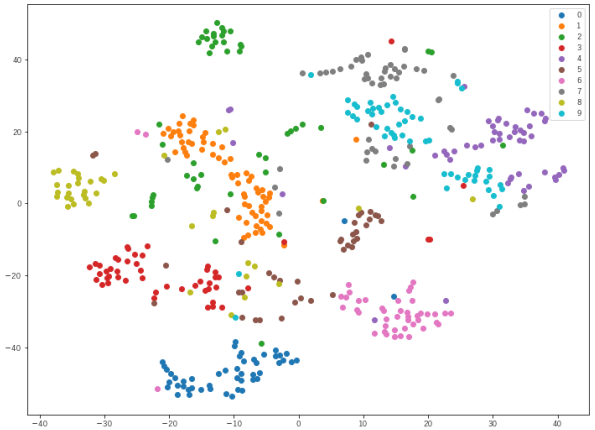
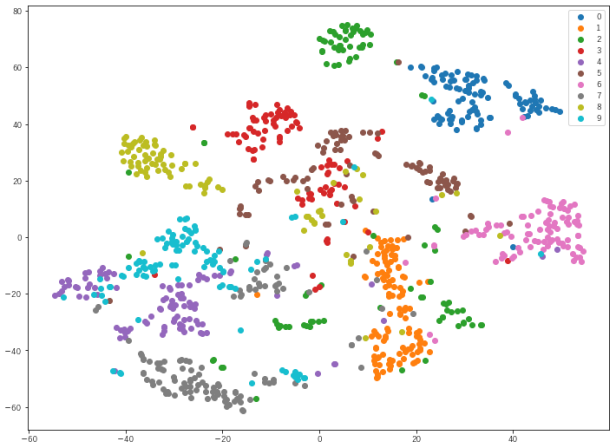
作図はそのまま

```
plt.figure(figsize=(12,9))
for j in range(0,10):
    list0 = [i for i in range(0,200) if y_train[i] == j]
    list0x = [result[i,0] for i in list0]
    list0y = [result[i,1] for i in list0]
    plt.scatter(list0x,list0y,label=f"{j}")
plt.legend()
plt.show()
```



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)

n数(使用するデータの数)を変えたときの比較

	n = 200	n = 500	n = 1000
PCA			
tSNE			

今回の高次元の次元削減(784次元→2次元)ではtSNEの方がうまく分類できていることが分かりました（tSNEは可視化によく使われる手法で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
皆同じ画像にはなりません
（余力のある人はラベルを置き換えられるか試してみましょう）