主成分分析

青木繁伸

2020 年 3 月 17 日

1　目的

主成分分析を行う。

2　使用法

```
import sys
sys.path.append("statlib")
from multi import pca
pca(dat, npca = None, verbose = True)
```

主成分負荷量や主成分得点をプロットする。

```
import sys
sys.path.append("statlib")
from multi import pca_plot
pca_plot(obj, type="fl", ax1=1, ax2=2, color="black", color2="blue")
```

2.1　引数

dat	2 次元配列またはデータフレーム
npca	求める主成分の個数
verbose	最小限のプリント出力をする（デフォルトは True）
obj	pca() の戻り値
type	type="fl"（デフォルト）なら主成分負荷量，type="fs" なら主成分得点をプロットする。
ax1	横軸にとる主成分の番号
ax2	縦軸にとる主成分の番号
color	点の色
color2	点のそばに付けるテキストの色

2.2　戻り値の名前

"r"	相関係数行列

"fl"　　主成分負荷量
"eval"　固有値
"fs"　　主成分得点
"names"　変数名

3　使用例

データは出力中に変数名を使うのでデータフレームとして与えるのを基本とするが，二次元配列または二重リストでもかまわない。

```
dat = [[144.0, 129.0, 419.9, 128.5, 192.6, 139.8],
       [177.0, 225.7, 331.2, 124.7,  83.1, 100.6],
       [355.9, 175.3, 173.5, 106.1, 136.5, 302.7],
       [207.1, 165.2, 254.3,  94.7, 197.1, 275.1],
       [212.9, 104.8, 158.6, 156.3, 267.1,  49.1],
       [371.5, 195.5, 152.4, 233.1,  34.9, 190.5],
       [246.1, 121.5, 121.1,  -1.4, 165.0, 110.4],
       [73.5,  33.2, 140.5, 221.2, 275.6,  -7.1],
       [131.3, 162.0, 365.1, 323.7, 146.1, 215.0],
       [155.4, 291.9, 194.6, 403.8, 222.7, 192.1],
       [322.4, 142.5, 211.9, 330.1, 249.2, 190.3],
       [236.0, 260.8, 224.4, 275.7, 226.8, 221.6],
       [240.1,  38.2, 323.2,  27.3, 265.3, 288.2],
       [211.1, 194.4, 148.4, 139.8, 187.7, 220.6],
       [144.4, 251.9, 100.7, 164.8, 158.6, 138.4]]

import sys
sys.path.append("statlib")
from multi import pca

a = pca(dat)
```

```
                PC1     PC2     PC3  Contribution
         x1  -0.703   0.457  -0.279   0.780
         x2  -0.667  -0.634   0.009   0.847
         x3   0.082   0.115   0.949   0.920
         x4  -0.131  -0.834   0.135   0.731
         x5   0.699  -0.003   0.043   0.491
         x6  -0.672   0.326   0.417   0.732
  Eigenvalue   1.904   1.425   1.171
Contribution    31.7    23.8    19.5
Cum.contrib.    31.7    55.5    75.0
```

```
import pandas as pd
```

```
dat = pd.read_csv("data/pca.csv")

import sys
sys.path.append("statlib")
from multi import pca

a = pca(dat)
```

	PC1	PC2	PC3	PC4	PC5	Contribution
X1	-0.685	0.112	0.232	0.483	-0.148	0.791
X2	-0.260	-0.295	0.490	-0.239	-0.266	0.523
X3	-0.610	-0.356	0.315	-0.117	-0.091	0.621
X4	-0.501	-0.215	-0.631	-0.013	-0.323	0.800
X5	-0.512	0.507	-0.424	-0.239	0.031	0.757
X6	-0.503	0.459	0.503	-0.082	0.103	0.733
X7	-0.338	0.595	-0.326	0.345	-0.211	0.738
X8	-0.186	0.088	0.129	0.350	0.761	0.760
X9	-0.185	0.404	0.306	-0.684	0.072	0.764
X10	-0.289	-0.312	-0.487	-0.505	0.294	0.759
X11	-0.310	-0.754	0.096	0.141	-0.117	0.707
X12	-0.361	-0.370	-0.160	0.056	0.522	0.568
Eigenvalue	2.169	2.067	1.712	1.356	1.217	
Contribution	18.1	17.2	14.3	11.3	10.1	
Cum.contrib.	18.1	35.3	49.6	60.9	71.0	

3.1 主成分負荷量のプロット

```
import sys
sys.path.append("statlib")
from multi import pca_plot

pca_plot(a)
```

3.2 主成分得点のプロット

```
a["fs"][:10, :]
```

```
array([[-2.86420705, -1.46241849,  1.77974374, -0.40197819, -0.4250754 ],
       [ 1.52272112, -2.06429251, -0.41259397, -0.86183071,  0.05210972],
       [ 2.89579476, -0.13033549,  2.13700131,  2.93963898,  0.71626212],
       [-2.70314081, -3.25410773,  0.26091312,  0.24192352,  0.93422987],
       [ 2.84713757, -0.77312347,  0.23167944, -1.86666494, -2.43766724],
       [-0.49259372,  2.60845544,  1.11427029,  0.00976229,  1.29631144],
```

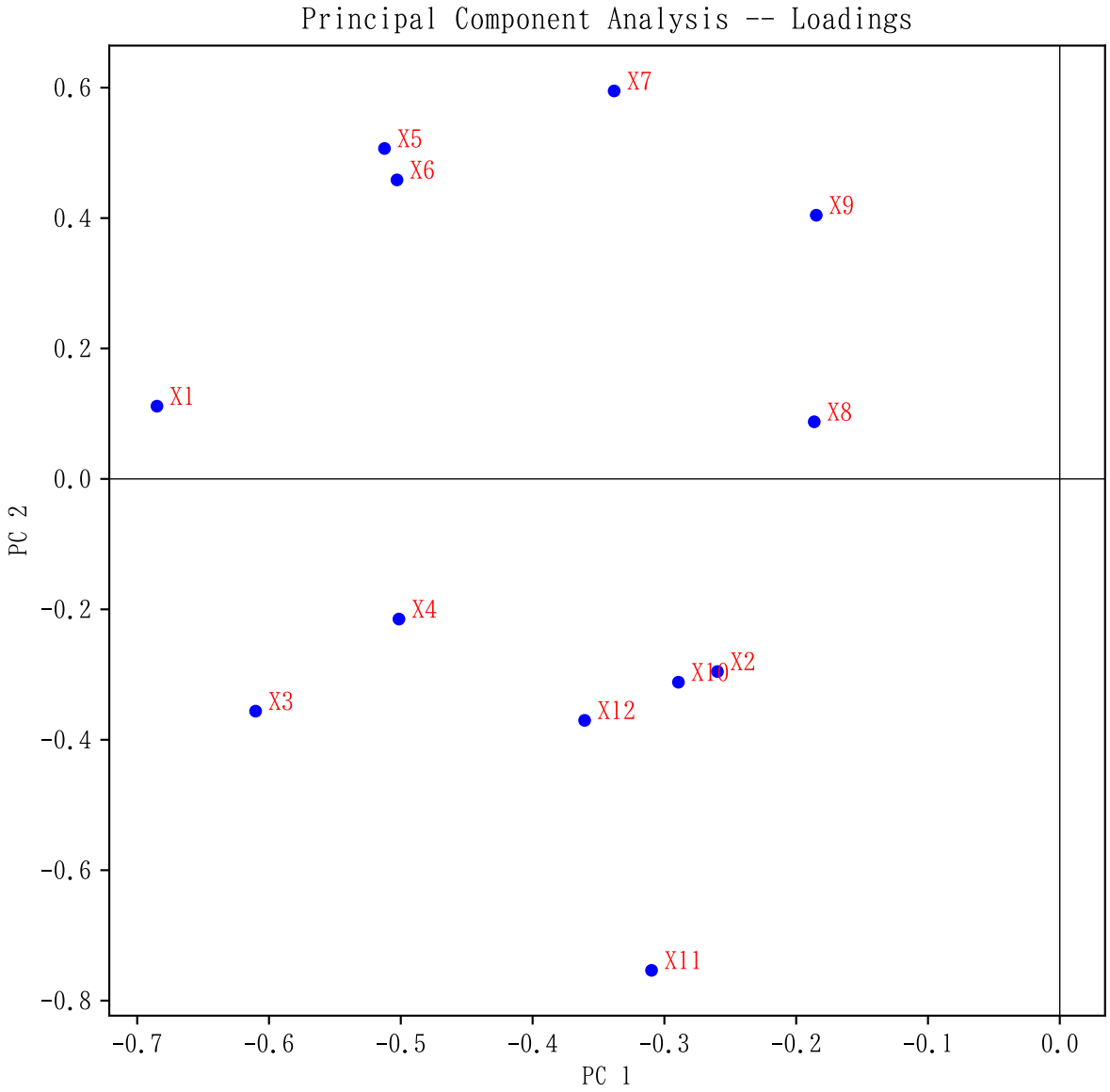

図 1　主成分負荷量のプロット

```
[-2.16493281,  3.05588295, -1.06013102,  0.99986893, -1.52711794],
[ 0.22612907,  1.9550428 , -0.842728  ,  2.27232159,  0.46735961],
[ 1.25732067,  1.66787266, -1.34398459, -1.98807916,  0.55570689],
[-0.95597517,  1.02109904,  1.17036492, -1.09529223,  0.82745354]])
```

```
pca_plot(a, type="fs")
```

3.3　バイプロット

```
from multi import pca_plot
```

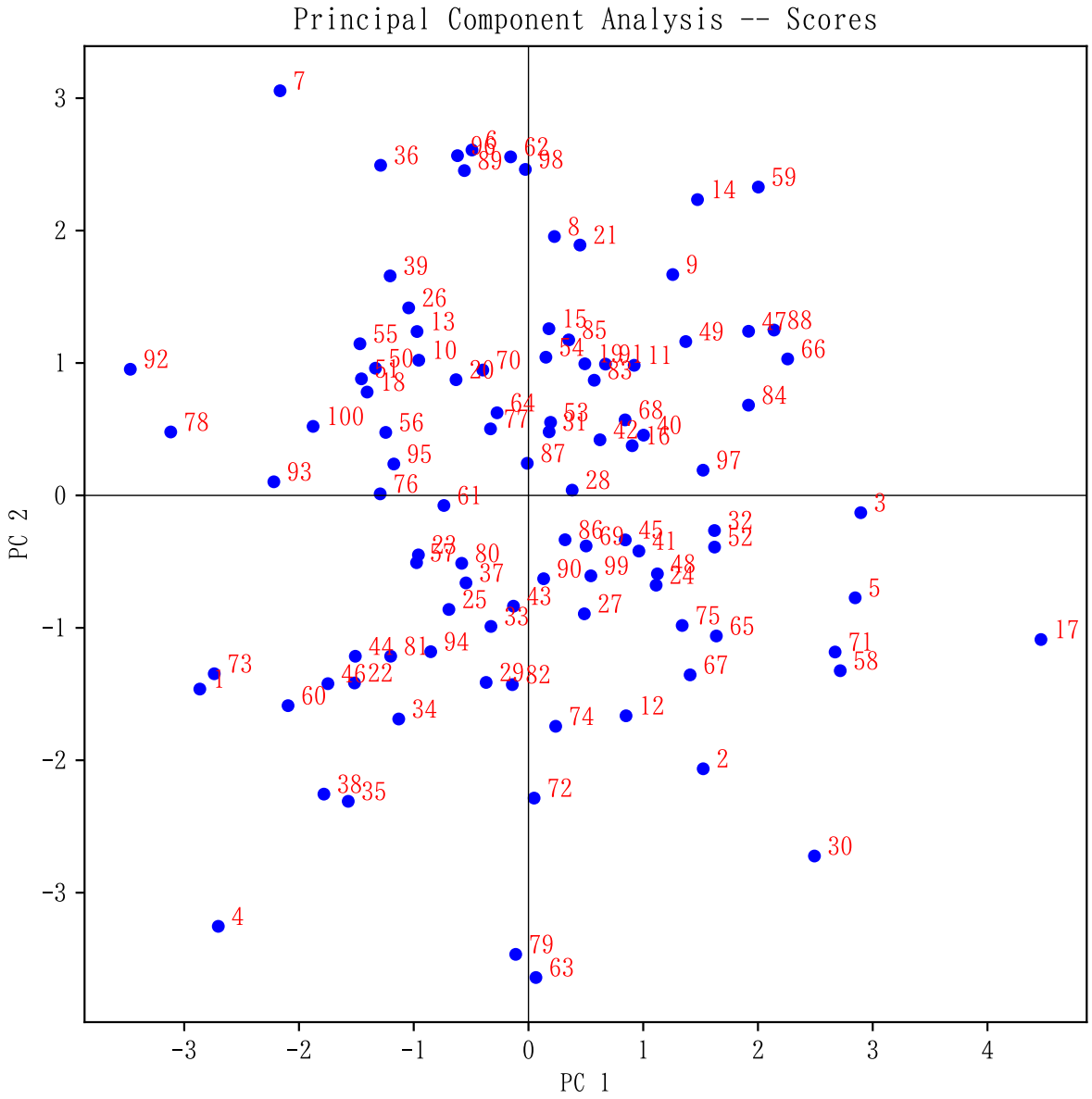

図 2　主成分得点のプロット

```
pca_plot(a, type="biplot")
```

3.4　ケース数が変数の個数より少ない場合

R の prcomp() と同じく，このような場合にも対応している。

```
dat = [[1,2,3,2,4,5,4],
       [3,2,5,4,5,3,5],
       [2,1,2,4,3,5,4],
       [1,2,3,2,3,4,3]]
```

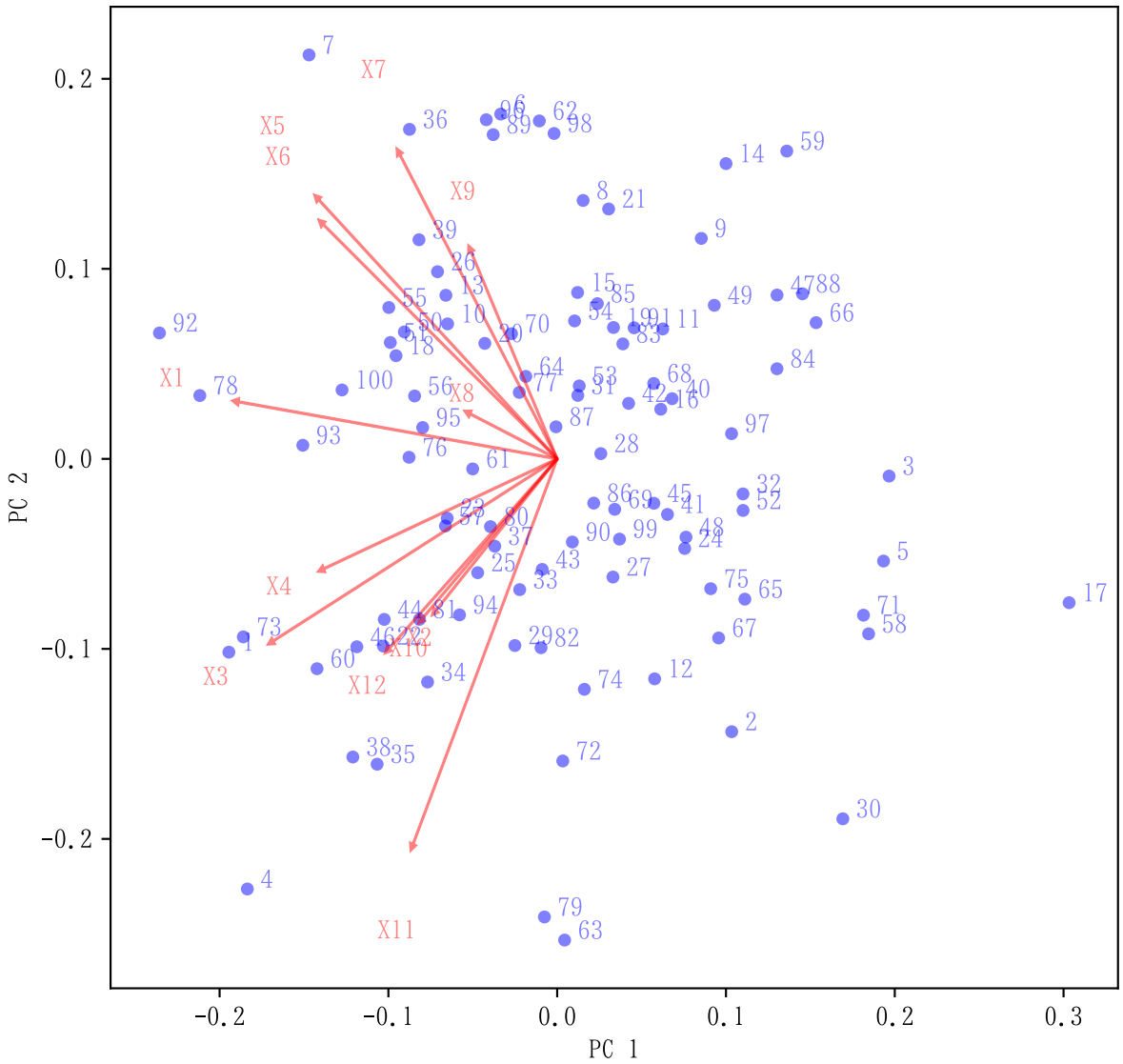

図 3　バイプロット

```
a = pca(dat)
```

	PC1	PC2	Contribution
x1	0.879	-0.457	0.981
x2	0.304	0.951	0.998
x3	0.910	0.408	0.995
x4	0.597	-0.791	0.982
x5	0.905	0.243	0.878
x6	-0.810	-0.314	0.755
x7	0.875	-0.298	0.855
Eigenvalue	4.291	2.153	
Contribution	61.3	30.8	
Cum.contrib.	61.3	92.1	

4 既存の Python の sklearn.decomposition モジュールの PCA クラスによる解析

データ入力

```
import pandas as pd
dat = pd.read_csv("data/pca.csv")
variables = dat.columns # 変数名
n, p = dat.shape # サンプルサイズ, 変数の個数
```

正規化

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
normalized_dat = sc.fit_transform(dat)

import numpy as np
normalized_dat *= np.sqrt((n-1)/n) # np.std(x, ddof = 1) で基準化したいとき
```

分析

```
from sklearn.decomposition import PCA
N_COMPONENTS = 5
pca = PCA(n_components=N_COMPONENTS)
pca.fit(normalized_dat)
```

```
PCA(copy=True, iterated_power='auto', n_components=5, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

主成分

```
# print(pca.components_)
print("{0:>12s}".format("Variable"), end = " ")
for j in range(N_COMPONENTS):
    print("{0:>7s}".format("PC"+str(j+1)), end = " ")
print()
for i in range(p):
    print("{0:>12s}".format(variables[i]), end = " ")
    for j in range(N_COMPONENTS):
        print("{0:7.3f}".format(pca.components_[j, i]), end = " ")
    print()
```

```
    Variable     PC1     PC2     PC3     PC4     PC5
          X1  -0.465  -0.078   0.177   0.415   0.134
          X2  -0.176   0.205   0.374  -0.205   0.241
```

```
          X3  -0.414   0.248   0.241  -0.101   0.083
          X4  -0.340   0.149  -0.482  -0.011   0.292
          X5  -0.348  -0.352  -0.324  -0.205  -0.028
          X6  -0.341  -0.319   0.384  -0.071  -0.093
          X7  -0.230  -0.414  -0.249   0.296   0.191
          X8  -0.127  -0.061   0.099   0.301  -0.689
          X9  -0.125  -0.281   0.234  -0.588  -0.065
         X10  -0.196   0.217  -0.372  -0.434  -0.266
         X11  -0.210   0.524   0.073   0.121   0.106
         X12  -0.245   0.258  -0.122   0.048  -0.473
```

主成分負荷量と固有値，寄与率，累積寄与率

```
# print(pca.explained_variance_)
# print(pca.explained_variance_ratio_)
eigenvalues = np.sqrt(pca.explained_variance_) # 固有値
loadings = (pca.components_.T * eigenvalues).T # 主成分負荷量
contribution = np.sum(loadings**2, axis=0) # 寄与率

def footer(name, x, dec = 3):
    print("{0:>12s}".format(name), end = " ")
    for j in range(len(x)):
        print("{0:7.{1:d}f}".format(x[j], dec), end = " ")
    print()

print("{0:>12s}".format("Variable"), end = " ")
for j in range(N_COMPONENTS):
    print("{0:>7s}".format("PC"+str(j+1)), end = " ")
print("Contribution")
for i in range(p):
    print("{0:>12s}".format(variables[i]), end = " ")
    for j in range(N_COMPONENTS):
        print("{0:7.3f}".format(loadings[j, i]), end = " ")
    print("{0:7.3f}".format(contribution[i]))
footer("  Eigenvalue", pca.explained_variance_)
footer("Contribution", pca.explained_variance_ratio_ * 100, dec = 1)
footer("Cum.contrib.", np.cumsum(pca.explained_variance_ratio_ * 100),
     dec = 1)
```

```
    Variable     PC1     PC2     PC3     PC4     PC5 Contribution
          X1  -0.685  -0.112   0.232   0.483   0.148   0.791
          X2  -0.260   0.295   0.490  -0.239   0.266   0.523
          X3  -0.610   0.356   0.315  -0.117   0.091   0.621
          X4  -0.501   0.215  -0.631  -0.013   0.323   0.800
          X5  -0.512  -0.507  -0.424  -0.239  -0.031   0.757
          X6  -0.503  -0.459   0.503  -0.082  -0.103   0.733
          X7  -0.338  -0.595  -0.326   0.345   0.211   0.738
```

X8	-0.186	-0.088	0.129	0.350	-0.761	0.760
X9	-0.185	-0.404	0.306	-0.684	-0.072	0.764
X10	-0.289	0.312	-0.487	-0.505	-0.294	0.759
X11	-0.310	0.754	0.096	0.141	0.117	0.707
X12	-0.361	0.370	-0.160	0.056	-0.522	0.568
Eigenvalue	2.169	2.067	1.712	1.356	1.217	
Contribution	18.1	17.2	14.3	11.3	10.1	
Cum.contrib.	18.1	35.3	49.6	60.9	71.0	

主成分得点

```
score = normalized_dat @ pca.components_.T
score[:5, :]
```

```
array([[-2.86420705,  1.46241849,  1.77974374, -0.40197819,  0.4250754 ],
       [ 1.52272112,  2.06429251, -0.41259397, -0.86183071, -0.05210972],
       [ 2.89579476,  0.13033549,  2.13700131,  2.93963898, -0.71626212],
       [-2.70314081,  3.25410773,  0.26091312,  0.24192352, -0.93422987],
       [ 2.84713757,  0.77312347,  0.23167944, -1.86666494,  2.43766724]])
```

主成分得点の分散は，固有値に等しい。

```
score.var(axis=0, ddof=1)
```

```
array([2.16863969, 2.0669161 , 1.71206571, 1.35615656, 1.21728342])
```