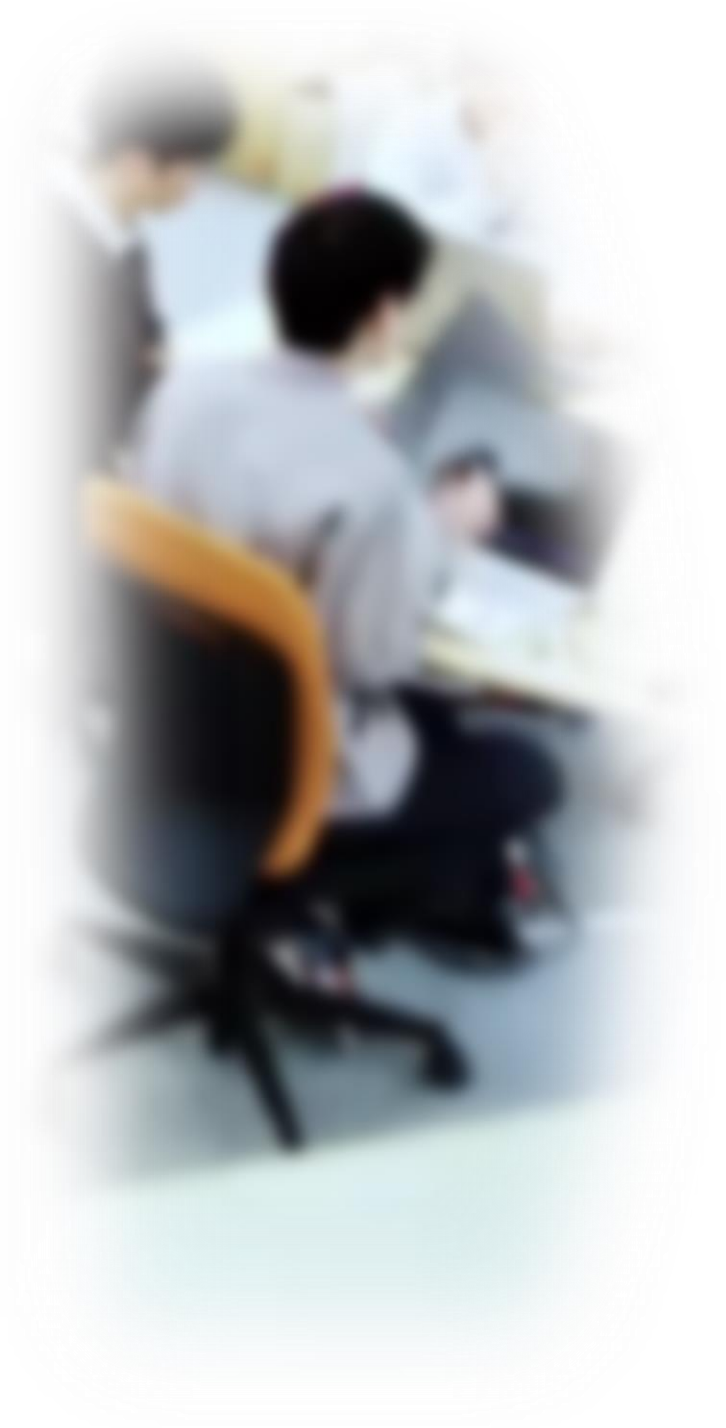
pd-3. 主成分分析，次元削減

（Python による ICT システム）

URL: https://www.kkaneko.jp/de/pd/index.html

金子邦彦



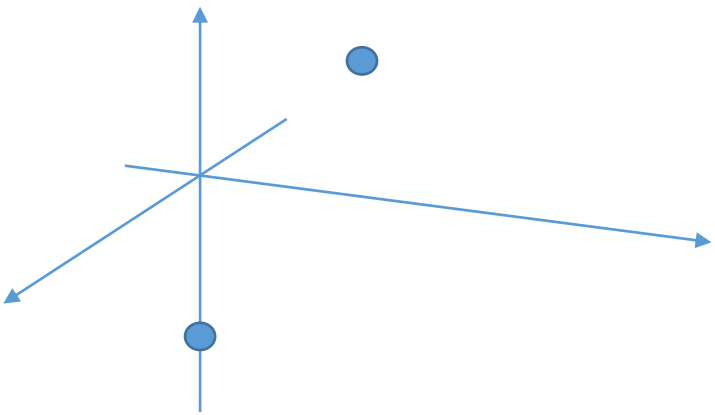
アウトライン

1. 主成分分析と次元削減
2. Iris データセットでの主成分分析の実行
3. 主成分分析と外れ値

3-1 主成分分析と次元削減

データの次元

データの**次元**：**データの表現に必要な最小の情報の数**



空間の中の２つの点

x	y	z
0	-20	0
10	20	0.1

２つのデータ「0 -20 0」と「10 20 0.1」

次元数：３

データの次元

さまざまなデータについて、**次元**を考えることが可能

温度	湿度
25	60
28	70

2 つのデータ「25 60」と「28 70」

次元数：2

次元削減

データの**次元削減**を考えることが可能

属性 z を削除

x	y	z
0	-20	0
10	20	0.1

x	y
0	-20
10	20

元データ：　次元数は**3**

次元数は**2**

次元削減の効果

- **可視化のため**

データを散布図などのグラフにするとき、データを２次元や３次元に次元削減

- **本質でない情報の除去のため**

データにノイズが含まれていたり、分析のために不要なものが含まれている場合、次元削減を行う

- **計算の効率化のため**

次元削減によりデータ全体のサイズが少なくなり、計算の効率化ができる

次元削減の手法①

次元削減の単純な方法

- 属性の削除

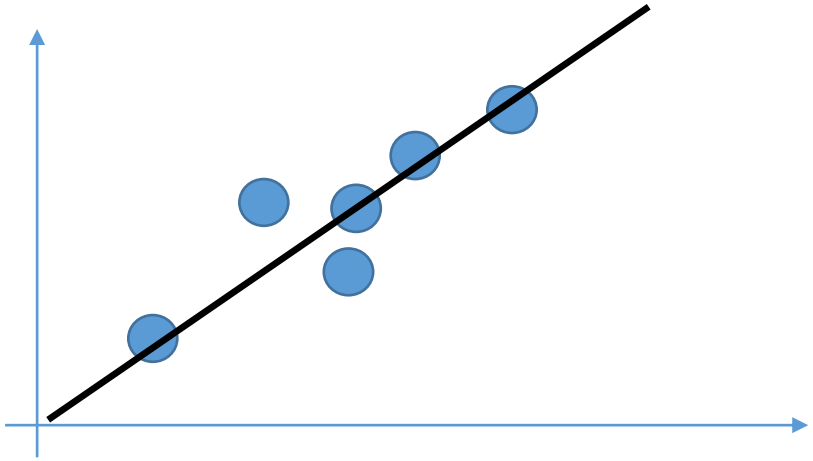
x	y	z
0	-20	0
10	20	0.1

次元数は**3**

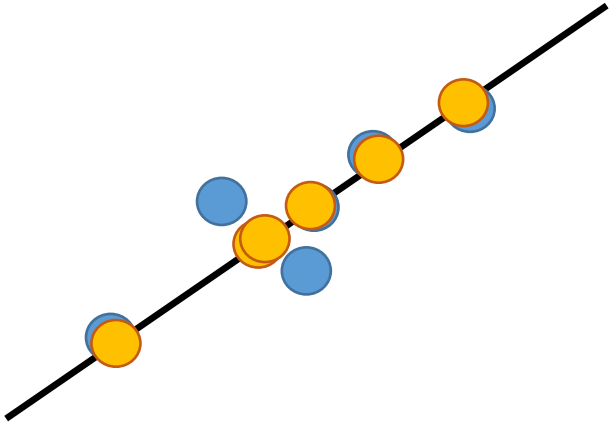
x	y
0	-20
10	20

次元数は**2**

- 近似直線への投影



次元数は**2**



次元数は**1**

次元削減の手法②

次元削減にはさまざまなアプローチがある

主成分分析（PCA）

データの分散が最大になる方向に軸を見つける（**データの特徴**を最もよく表す**軸**を見つける）．

- **T-SNE**

  データ間の距離を用いる．距離を保つようにしながら次元削減.

- **Linear Discriminant Analysis**

  教師有りの機械学習の技術を使用.

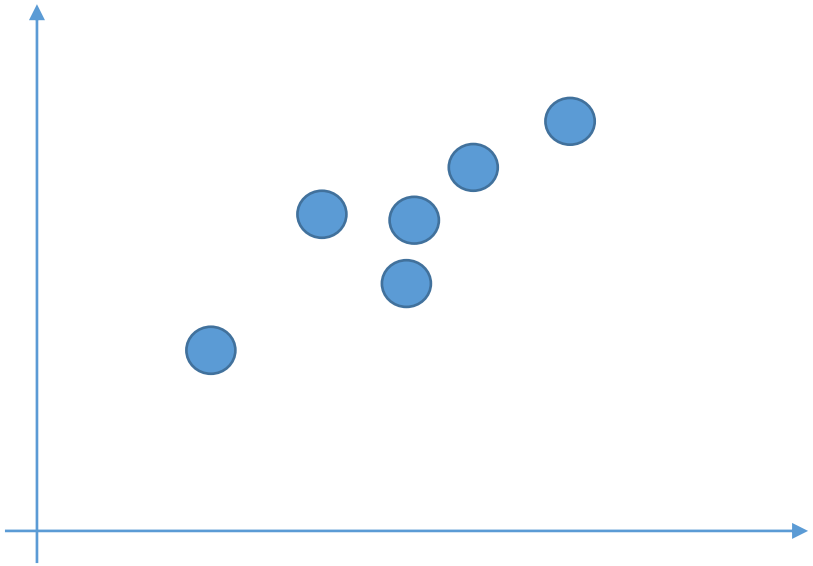
  「データは、種類ごとの正規分布」，「各種類のデータは同じ形状の分布である（共通の共分散行列を持つ）．ただし，平均は違ってもよい」という性質の成り立つデータに有効

- **QDA**

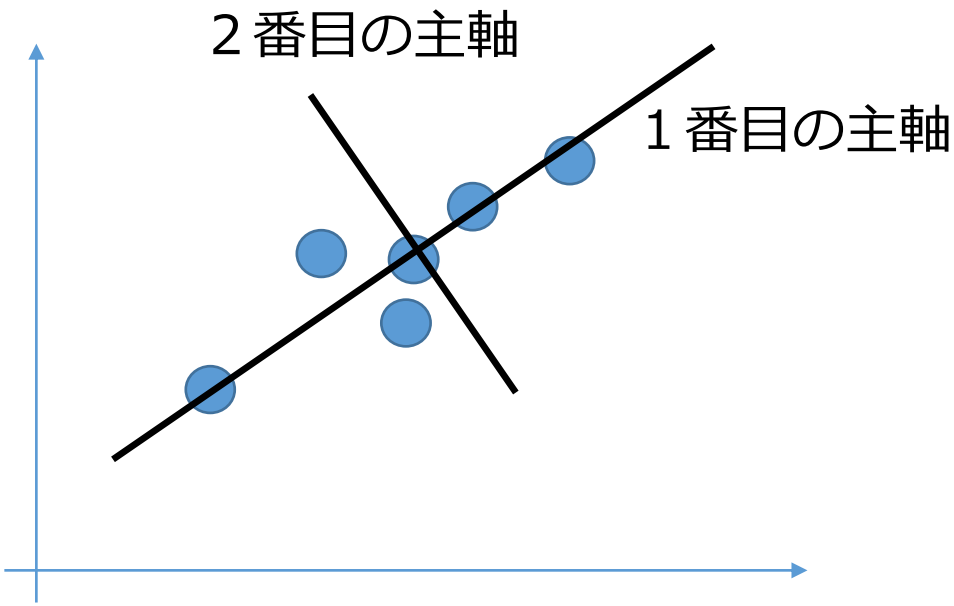
  LDAの改良．「共通の共分散行列を持つ」という仮定を置かない

主成分分析と主軸

- **主成分分析**では、**データの分散が最大となる方向の軸**を**主軸**という
- **主成分分析**では、**元のデータの次元数**と**同じ数**の**主軸**を作成できる



次元数は**2**



主軸を**2**つ

主成分分析と主軸

主成分分析では、**元のデータの次元数**と**同じ数**の**主軸**を作成できる

① 1番目の**主軸**は，データの分散が最大になるような方向の軸

② 2番目の**主軸**は，1番目の**主軸**とは異なる方向で、その方向における分散が最大

1番目の**主軸**に対する**データの成分を取り除いた**残りのデータから行う

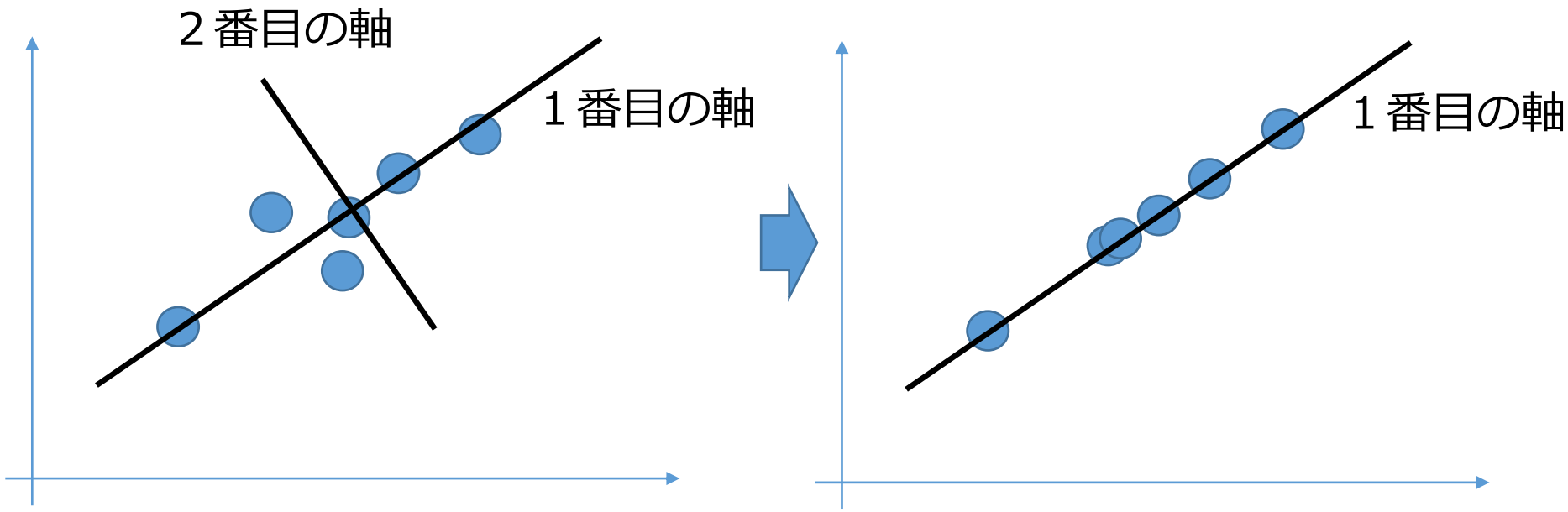
③ 3番目以降も同様

3番目の**主軸**は1番目と2番目の**主軸**に対する**成分を取り除いた**残りのデータから、その方向における分散が最大となるように選ぶ

得られた各主軸は互いに直交（各段階で「成分を取り除く」ので）

主成分分析

- **主成分分析**では，得られた**主軸**の中から、上位の**主軸**を**選び**，下位の**主軸**を削除.
- **選ばれた主軸**に，元データを投影することで，次元削減を行う．この投影は線形変換である.



元データ：　次元は**2**

次元は**1**

ランダムデータの散布図（Google Colaboratory）

```
import numpy as np
import matplotlib.pyplot as plt
x = np.random.normal(0, 1, 100)
y = 3 * x + np.random.normal(0, 1, 100)
# グラフとラベル
plt.axis('equal')
plt.scatter(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('random data')
plt.show()
```

ランダムデータ

xは平均0，標準偏差1

Yはxの3倍にノイズを追加

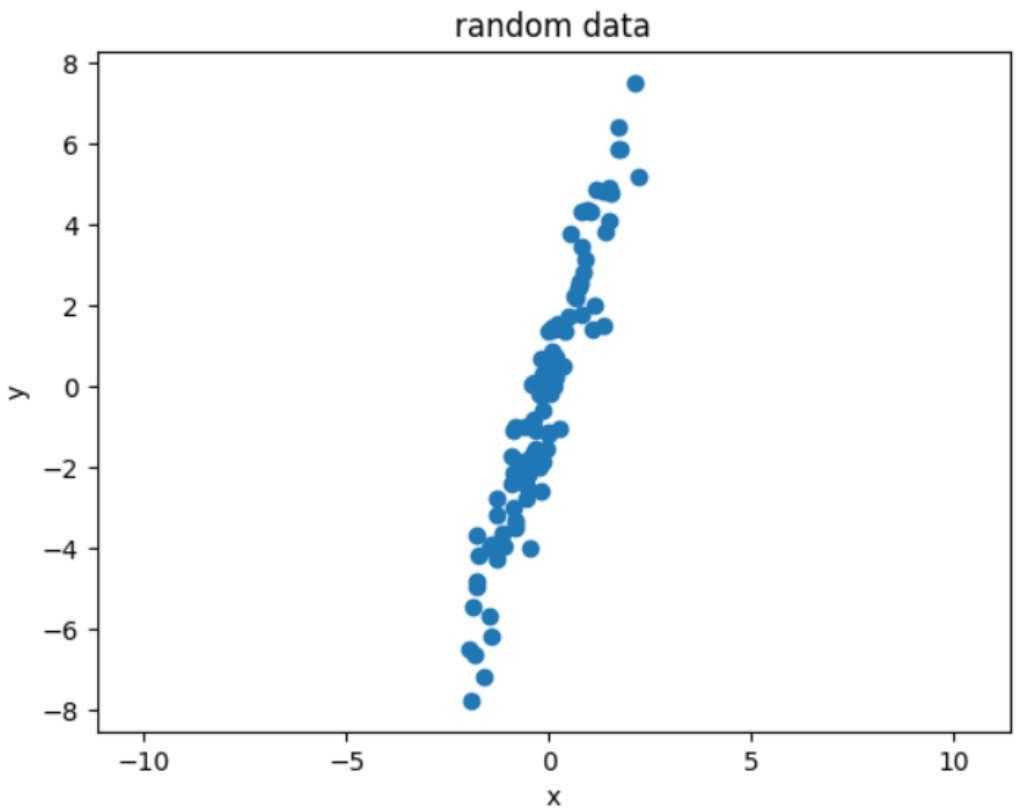
データの個数は100

散布図

コード

ランダムデータの散布図（Google Colaboratory）

```python
import numpy as np
import matplotlib.pyplot as plt
x = np.random.normal(0, 1, 100)
y = 3 * x + np.random.normal(0, 1, 100)
# グラフとラベル
plt.axis('equal')
plt.scatter(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('random data')
plt.show()
```



ランダムデータの主成分分析（Google Colaboratory）

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

x = np.random.normal(0, 1, 100)
y = 3 * x + np.random.normal(0, 1, 100)
data = np.column_stack((x, y))

pca = PCA(n_components=2)
pca.fit(data)
components = pca.components_
explained_variance = pca.explained_variance_

print(f'Principal components:¥n{components}')
print(f'Explained variance: {explained_variance}')
plt.axis('equal')
plt.scatter(x, y)
plt.quiver(0, 0, components[0, 0], components[0, 1], color='r')
plt.quiver(0, 0, components[1, 0], components[1, 1],
color='b')
plt.title('PCA')
plt.show()
```

ランダムデータ

xは平均0，標準偏差1

Yはxの3倍にノイズを追加

データの個数は100

主成分分析

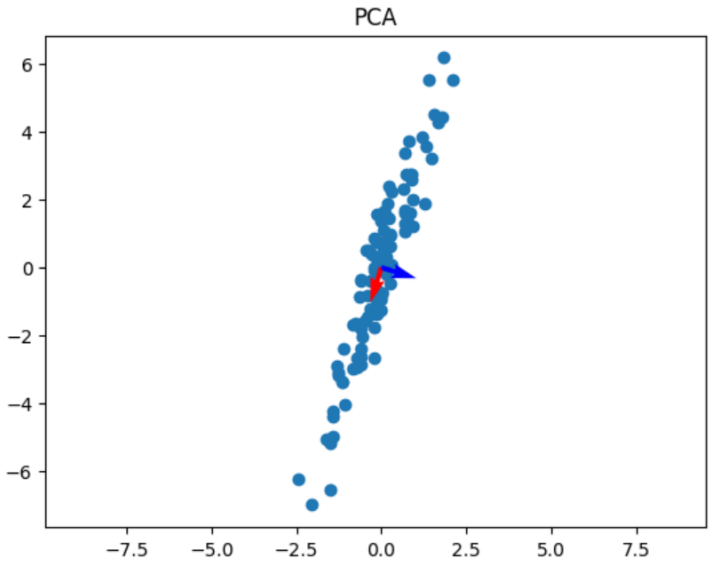
結果表示

プロット

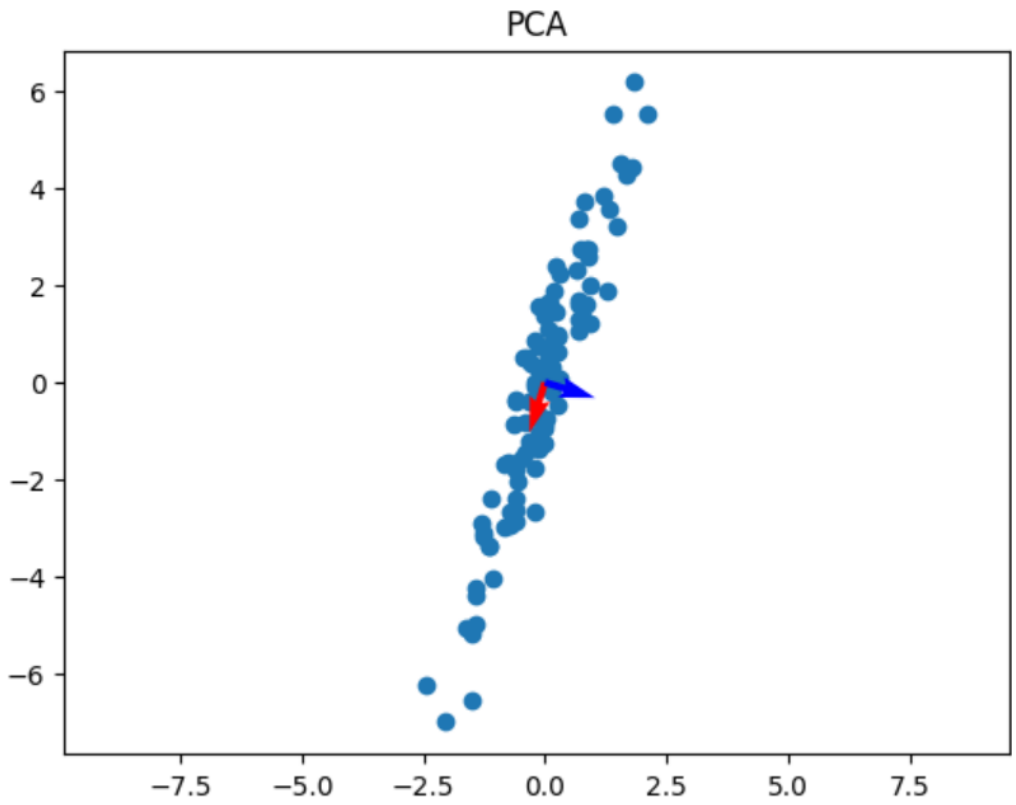
ランダムデータの主成分分析（Google Colaboratory）

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

x = np.random.normal(0, 1, 100)
y = 3 * x + np.random.normal(0, 1, 100)
data = np.column_stack((x, y))

pca = PCA(n_components=2)
pca.fit(data)
components = pca.components_
explained_variance = pca.explained_variance_

print(f'Principal components:¥n{components}')
print(f'Explained variance: {explained_variance}')
plt.axis('equal')
plt.scatter(x, y)
plt.quiver(0, 0, components[0, 0], components[0, 1], color='r')
plt.quiver(0, 0, components[1, 0], components[1, 1], color='b')
plt.title('PCA')
plt.show()
```

```
Principal components:
[[-0.29555466 -0.95532583]
 [ 0.95532583 -0.29555466]]
Explained variance: [7.87225366 0.0835529 ]
```



主成分分析の例



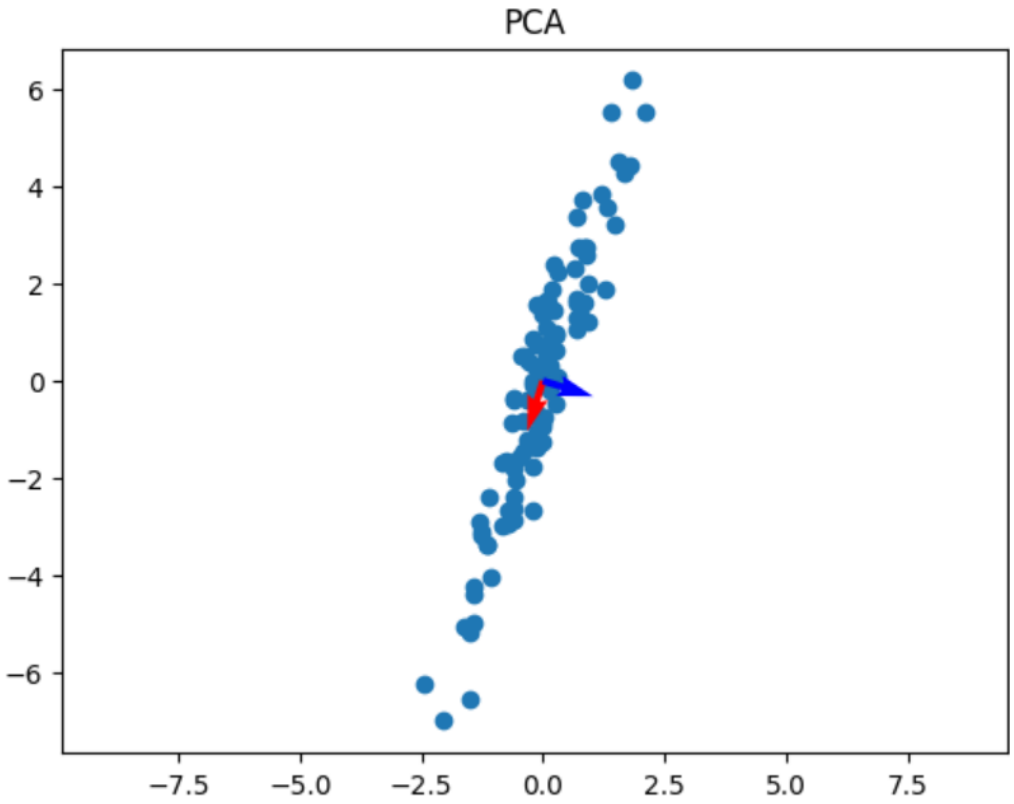
1番目の主軸

```
Principal components:
[-0.29555466 -0.95532583]
[ 0.95532583 -0.29555466]
```

２番目の主軸

主成分分析の結果

主成分分析の実行について



1番目の主軸

```
Principal components:
[-0.29555466 -0.95532583]
[ 0.95532583 -0.29555466]
```

2番目の主軸

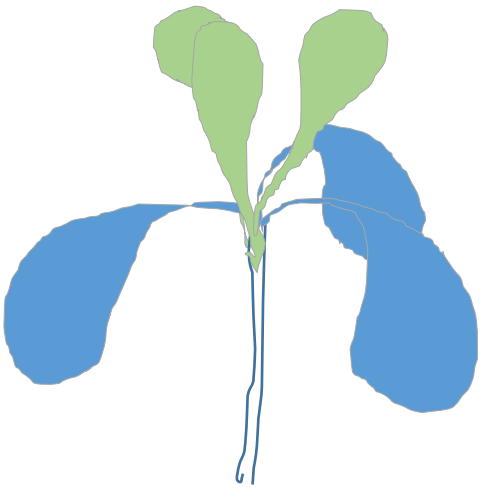
主成分分析の結果

- **スケーリング**: 主成分分析の分散の算出において，それぞれの値の単位をそろえるスケーリングが重要.
- このプログラムでは、説明の簡単のため、スケーリングを行っていない

はとても重要です。特に、異なる尺度の特徴があるときには、それぞれの特徴をすることが一般的

3-2 Iris データセットでの主成分分析の実行

アヤメ属 (Iris)



- 多年草
- 世界に 150種. 日本に 9種.
- 花被片は 6個
- 外花被片（がいかひへん） Sepal
  3個（大型で下に垂れる）
- 内花被片（ないかひへん） Petal
  3個（直立する）

Iris データセット

```
[[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]
```

0列 1列 2列 3列

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

Iris データセット

- **3種のアヤメ**の外花被辺、内花被片の幅と長さを計測したデータセット

  Iris setosa

  Iris versicolor

  Iris virginica
- データ数は 150 (50 × 3)
- 作成者：Ronald Fisher
- 作成年：1936

今から行うこと

```
[[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]
```

0列 1列 2列 3列

Iris データセット

元データは４次元

４次元を２次元に削減する

２つの方法

①Iris データセットの０，１列目

②主成分分析により２次元に次元削減

Iris データセットのロード（Google Colaboratory）

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
x = iris.data
y = iris.target
print(x)
print(y)
```

インポート

x, y にロード、

（x は２次元の配列、

y は１次元の配列）

表示

コード

Iris データセットのロード（Google Colaboratory）

```python
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
x = iris.data
y = iris.target
print(x)
print(y)
```

```
[[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]
```

Iris データセットの０，１列目（Google Colaboratory）

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
x = iris.data
y = iris.target
# グラフ
plt.scatter(x[:, 0], x[:, 1])
# ラベル
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('Iris dataset')
# 表示
plt.show()
```

インポート

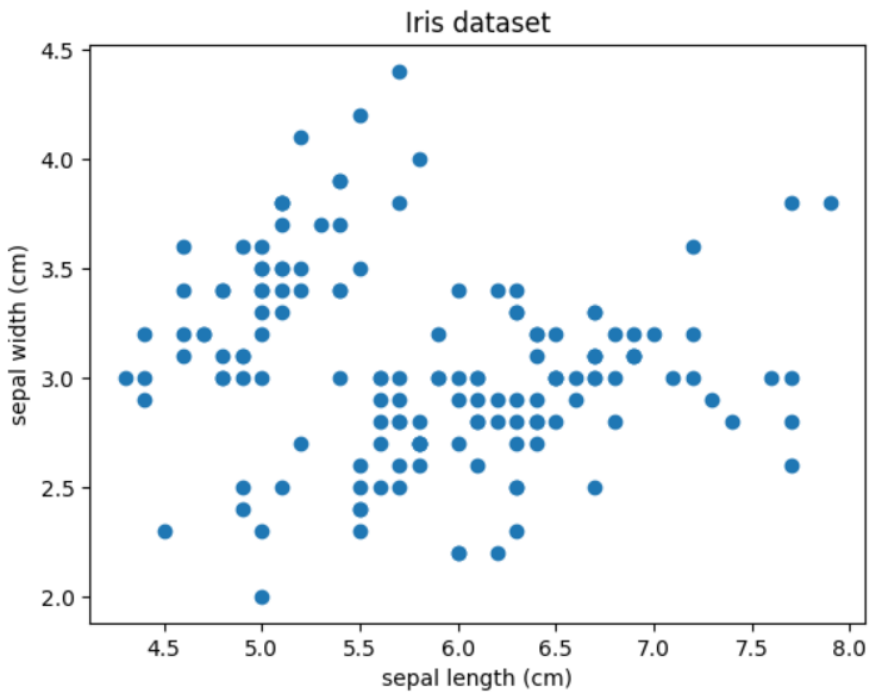
Irisデータセットをロード

グラフの設定
（0列1列が縦横に）

表示

Iris データセットの０，１列目（Google Colaboratory）

```python
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
x = iris.data
y = iris.target
# グラフ
plt.scatter(x[:, 0], x[:, 1])
# ラベル
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('Iris dataset')
# 表示
plt.show()
```



主成分分析により２次元に次元削減（Google Colaboratory）

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

iris = load_iris()
data = iris.data
y = iris.target

scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)

plt.scatter(data_pca[:, 0], data_pca[:, 1])
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('PCA of IRIS dataset')
plt.show()
```

インポート

Iris データセットをロード

平均0、標準偏差1にスケーリング

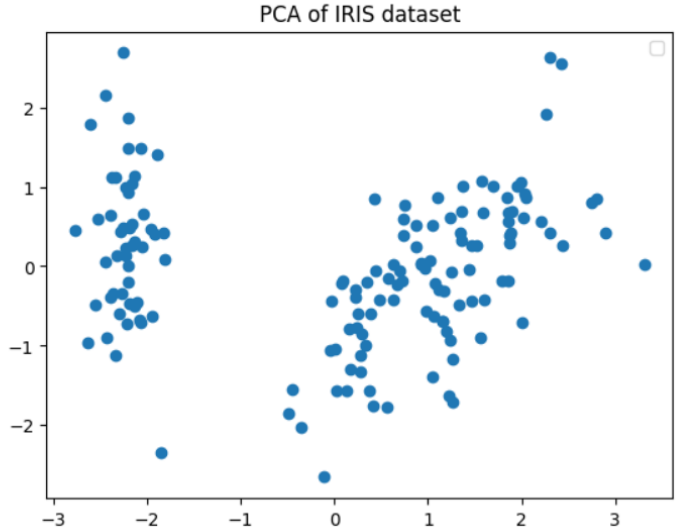
主成分分析．２次元に次元削減

散布図

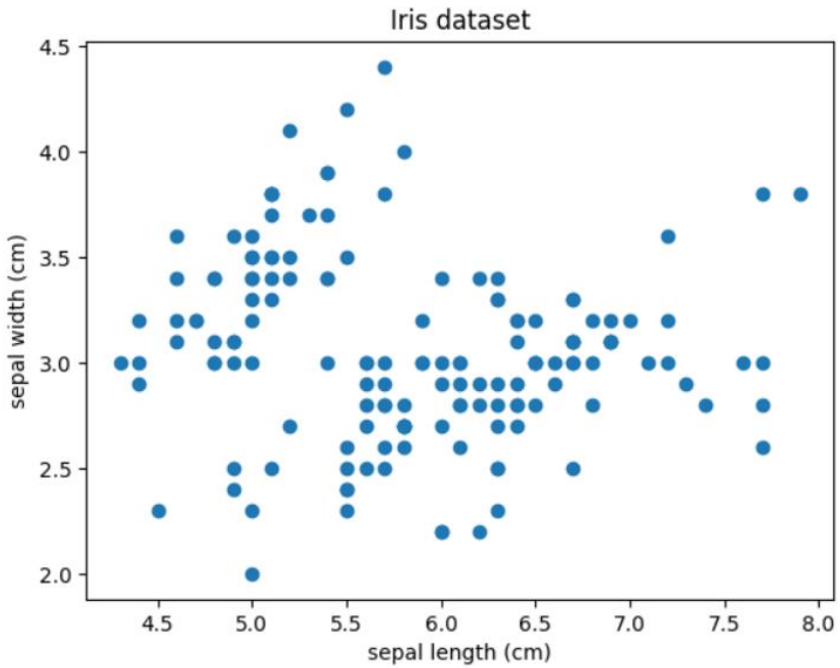
主成分分析により２次元に次元削減（Google Colaboratory）

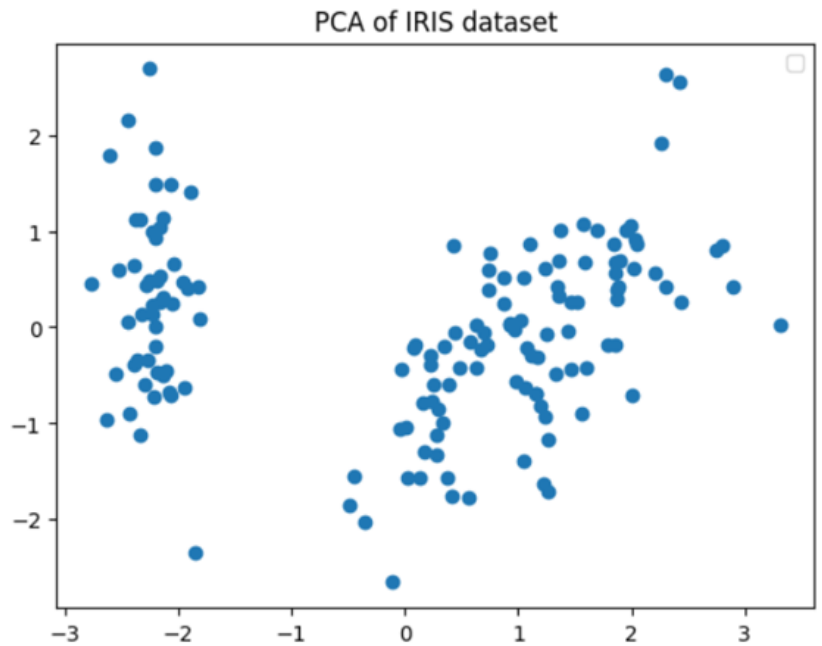
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

iris = load_iris()
data = iris.data
y = iris.target

# Data Scaling
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

pca = PCA(n_components=2)  # reduce the dimension to 2
data_pca = pca.fit_transform(data_scaled)

plt.scatter(data_pca[:, 0], data_pca[:, 1])
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('PCA of IRIS dataset')
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Not
```



４次元を２次元に削減する２つの方法





①Iris データセットの０，１列目　　**②主成分分析により２次元に次元削減**

Iris データセットの４列のデータをプロットするとき、
主成分分析での次元削減の方が有効

3-3 主成分分析と外れ値

ノイズ、外れ値、計測漏れ

データサイエンスは，便利であるが，万能ではない

- ノイズを含むデータについて，**ノイズがランダム**であれば，**平均**に**影響はない**
- **ノイズがランダムでない場合**，ノイズが悪影響を及ぼす

ノイズの他にも、**外れ値**（他の値と比べて、異常に離れた値）、**計測漏れ**（データが空，データが０）も、次元削減に悪影響

外れ値や**計測漏れ**などの**不適切なデータ**は、**手作業や、適切な分析手法で取り除く必要あり**

外れ値の検出、欠損データの取り扱い

- **外れ値の検出**：データが**正規分布**に従っていると**仮定**し，「**平均から何倍の標準偏差以上離れているデータを外れ値**」とするなど

- **欠損データの取り扱い**：**無視**したり，**平均値や中央値で補完**する．欠損データにパターンがある場合には，欠損データの原因を調べる手が仮になる場合がある．

- 異常検出：統計，機械学習の手法がある

主成分分析を利用した外れ値の検出

考え方

①元データ（外れ値を含む）を主成分分析で処理

②①の結果をもちいて、外れ値と、それ以外を区別

主成分分析を利用した外れ値の検出（Google Colaboratory）

```
!pip install pyod
import numpy as np
import matplotlib.pyplot as plt
from pyod.models.pca import PCA as PCA_pyod
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

iris = load_iris()
data = iris.data
y = iris.target

scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

pca = PCA_pyod(n_components=2, contamination=0.1)
pca.fit(data_scaled)

y_train_pred = pca.labels_  # binary labels (0: inliers, 1: outliers)
y_train_scores = pca.decision_scores_  # raw outlier scores

outlier_mask = y_train_pred != 0  # True for outliers
inlier_mask = np.logical_not(outlier_mask)  # True for inliers

data_pca_manual = np.dot(data_scaled, pca.components_.T)

plt.scatter(data_pca_manual[inlier_mask, 0], data_pca_manual[inlier_mask, 1], color='b', label='Inliers')
plt.scatter(data_pca_manual[outlier_mask, 0], data_pca_manual[outlier_mask, 1], color='r', label='Outliers')
plt.legend()
plt.title('Robust PCA of IRIS dataset')
plt.show()
```

インストール

インポート

Iris データセットをロード

平均0、標準偏差1にスケーリング

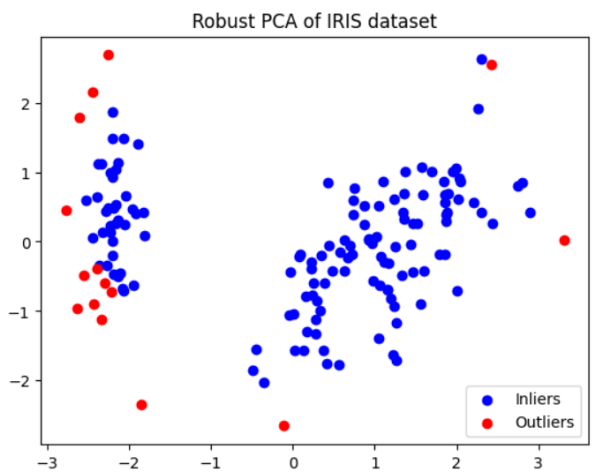
主成分分析．２次元に次元削減

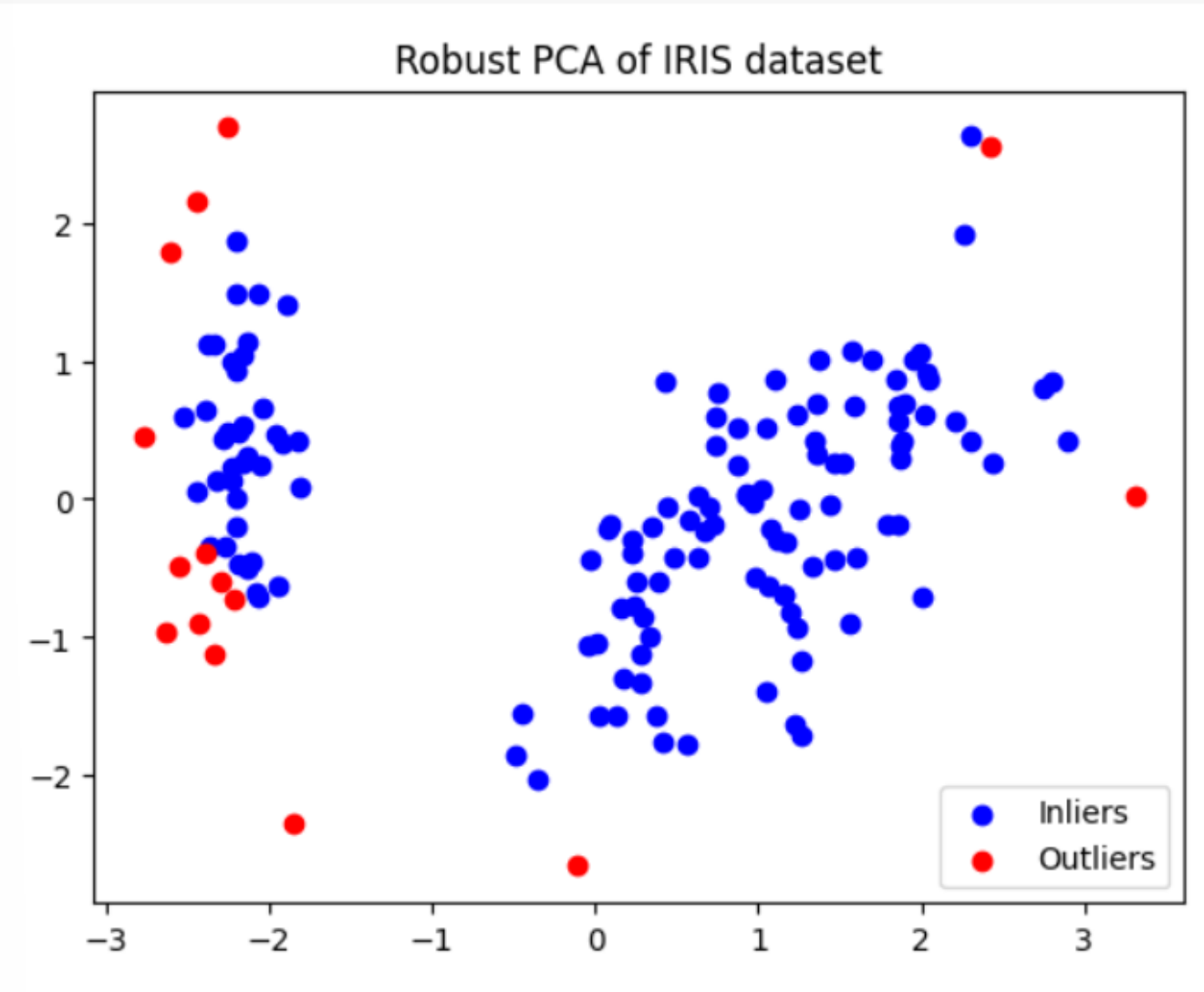
外れ値の検出

散布図

主成分分析を利用した外れ値の検出（Google Colaboratory）

```python
import numpy as np
import matplotlib.pyplot as plt
from pyod.models.pca import PCA as PCA_pyod
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

iris = load_iris()
data = iris.data
y = iris.target

scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

pca = PCA_pyod(n_components=2, contamination=0.1)  # contamination parameter is used to control the amount of outliers
pca.fit(data_scaled)

y_train_pred = pca.labels_  # binary labels (0: inliers, 1: outliers)
y_train_scores = pca.decision_scores_  # raw outlier scores

outlier_mask = y_train_pred != 0  # True for outliers
inlier_mask = np.logical_not(outlier_mask)  # True for inliers

data_pca_manual = np.dot(data_scaled, pca.components_.T)

plt.scatter(data_pca_manual[inlier_mask, 0], data_pca_manual[inlier_mask, 1], color='b', label='Inliers')
plt.scatter(data_pca_manual[outlier_mask, 0], data_pca_manual[outlier_mask, 1], color='r', label='Outliers')
plt.legend()
plt.title('Robust PCA of IRIS dataset')
plt.show()
```





赤は、外れ値と検出されたもの。
青はそれ以外

このプログラムの利用上の注意点

- 外れ値の割合を指定

次では「0.1」を設定．外れ値は全体の 10パーセントだろうと設定

pca = PCA_pyod(n_components=2, **contamination=0.1**)

- このプログラムは「外れ値を含んだ場合でも、なるべく正当な主成分分析を行う」という**ロバスト主成分分析**を行っている

全体まとめ

- **主成分分析と次元削減**：**主成分分析（PCA）**は**次元削減**の一手法．**多次元データを低次元データに変換**.
- **主成分分析の特徴**：**主軸**は，データの分散を最大限に保つように選ばれる．**元のデータセットの情報を可能な限り保持**するように**次元削減**を行う.
- **外れ値：**他の値と比べて、異常に離れた値．取り除くことが必要
- **主成分分析のよる外れ値の検出**：**主成分分析**により、**外れ値の検出**を行う技術がある