+ コード + テキスト

RAM ディスク

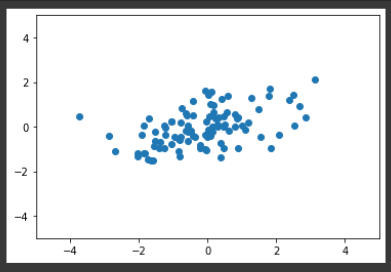
主成分分析

```
[1]  1 #%matplotlib inlineは、バックエンドをアウトプット行出力用に設定するという意味
     2 %matplotlib inline
     3 import numpy as np
     4 import matplotlib.pyplot as plt
```

訓練データ生成

```
[2]  1 n_sample = 100
     2
     3 def gen_data(n_sample):
     4     mean = [0, 0]
     5     cov = [[2, 0.7], [0.7, 1]] #共分散？
     6     #np.random.multivariate_normal:多変量正規分布からランダムなサンプルを抽出
     7     #引数:平均値、共分散、サンプル数
     8     return np.random.multivariate_normal(mean, cov, n_sample)
     9
    10 def plt_data(X):
    11     plt.scatter(X[:, 0], X[:, 1])
    12     plt.xlim(-5, 5)
    13     plt.ylim(-5, 5)
```

```
[3]  1 X = gen_data(n_sample)
     2 plt_data(X)
```



学習

訓練データ$X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^{\mathrm{T}}$に対して$\mathbb{E}[\boldsymbol{x}] = \boldsymbol{0}$となるように変換する。

すると、不偏共分散行列は$\boldsymbol{Var}[\boldsymbol{x}] = \frac{1}{n-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$と書ける。

$\boldsymbol{Var}[\boldsymbol{x}]$を固有値分解し、固有値の大きい順に対応する固有ベクトルを第1主成分($\boldsymbol{w}_1$), 第2主成分($\boldsymbol{w}_2$), ...とよぶ。

```
[4]  1 n_components=2
     2
     3 def get_moments(X):
     4     mean = X.mean(axis=0)
     5     stan_cov = np.dot((X - mean).T, X - mean) / (len(X) - 1)
     6     return mean, stan_cov
     7
     8 def get_components(eigenvectors, n_components):
```

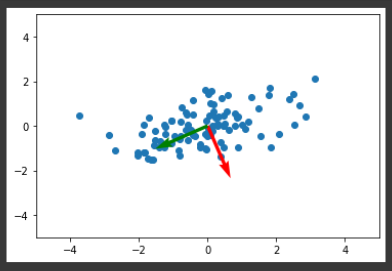
+ コード + テキスト

RAM ディスク

```
[4]  8 def get_components(eigenvectors, n_components):
     9 #     W = eigenvectors[:, -n_components:]
    10 #     return W.T[::-1]
    11     W = eigenvectors[:, ::-1][:, :n_components]
    12     return W.T
    13
    14 def plt_result(X, first, second):
    15     plt.scatter(X[:, 0], X[:, 1])
    16     plt.xlim(-5, 5)
    17     plt.ylim(-5, 5)
    18     # 第１主成分
    19     #matplotlib.pyplot.quiver()はベクトル場を可視化する。
    20     #X, Yはベクトルの開始点、U, Vはベクトルの成分
    21     plt.quiver(0, 0, first[0], first[1], width=0.01, scale=6, color='red')
    22     # 第２主成分
    23     plt.quiver(0, 0, second[0], second[1], width=0.01, scale=6, color='green')
```

```
[5]  1 #分散共分散行列を標準化
     2 meean, stan_cov = get_moments(X)
     3 #固有値と固有ベクトルを計算
     4 #np.linalg.eigh() : 複素エルミート（共役対称）または実対称行列の固有値と固有ベクトルを返します。
     5 eigenvalues, eigenvectors = np.linalg.eigh(stan_cov)
     6
     7 # print(eigenvalues)
     8 # print(eigenvectors)
     9 components = get_components(eigenvectors, n_components)
    10
```

```
plt_result(X, eigenvectors[0, :], eigenvectors[1, :])
```



変換（射影）

元のデータをm次元に変換(射影)するときは行列Wを$W = [\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_m]$とし、データ点$\boldsymbol{x}$を$\boldsymbol{z} = W^{\mathrm{T}}\boldsymbol{x}$によって変換(射影)する。

よって、データXに対しては$Z = X^{\mathrm{T}}W$によって変換する。

[6]
```
def transform_by_pca(X, pca):
    mean = X.mean(axis=0)
    return np.dot(X-mean, components)
```

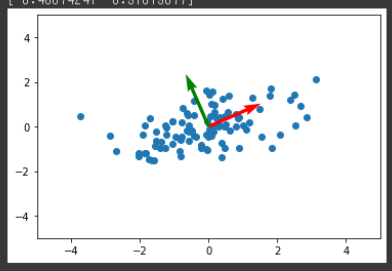
[7]
```
Z = transform_by_pca(X, components.T)
plt.xlim(-5, 5)
plt.ylim(-5, 5)
```

```
(-5.0, 5.0)
```

0 秒　完了時間: 21:43

```
[12]  print(pca.components_[0, :])
      print(pca.components_[1, :])
      plt_result(X, pca.components_[0, :], pca.components_[1, :])
```

```
[0.91619077 0.40074241]
[-0.40074241  0.91619077]
```



```
[13]  from sklearn.decomposition import PCA
      pca = PCA(n_components=2)
      pca.fit(X)

      plt_result(X, pca.components_[0, :], pca.components_[1, :])

      print('components: {}'.format(pca.components_))
      print('mean: {}'.format(pca.mean_))
      print('covariance: {}'.format(pca.get_covariance()))
```

```
コンポーネント：[[0.91619077 0.40074241]
 [-0.40074241 0.91619077]]
平均：[-0.13306501 -0.04539885]
共分散：[[1.66343942 0.52986926]
 [0.52986926 0.68379985]]
```