z79bhpq5z

February 24, 2025

1 PCA : Feature extraction technique

[171]:
```
from PIL import Image
import IPython.display as display

# Open and display the image
img = Image.open("C:/Users/lenovo/Downloads/pca pic.jpg")
display.display(img)
```

Summarizing the PCA approach

Listed below are the 6 general steps for performing a principal component analysis, which we will investigate in the following sections.

1. Take the whole dataset consisting of d-dimensional samples ignoring the class labels
2. Compute the d-dimensional mean vector (i.e., the means for every dimension of the whole dataset)
3. Compute the scatter matrix (alternatively, the covariance matrix) of the whole data set
4. Compute eigenvectors ($\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_d$) and corresponding eigenvalues ($\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \ldots, \boldsymbol{\lambda}_d$)
5. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a $d \times k$ dimensional matrix $\boldsymbol{W}$ (where every column represents an eigenvector)
6. Use this $d \times k$ eigenvector matrix to transform the samples onto the new subspace. This can be summarized by the mathematical equation: $\boldsymbol{y} = \boldsymbol{W}^T \times \boldsymbol{x}$ (where $\boldsymbol{x}$ is a $d \times 1$-dimensional vector representing one sample, and $\boldsymbol{y}$ is the transformed $k \times 1$-dimensional sample in the new subspace.)

[221]:
```
import pandas as pd

df = pd.read_csv(
    filepath_or_buffer='https://archive.ics.uci.edu/ml/
 ↪machine-learning-databases/iris/iris.data',
    header=None,
    sep=',')
```

```
df.columns=['sepal_len', 'sepal_wid', 'petal_len', 'petal_wid', 'class']
df.dropna(how="all", inplace=True) # drops the empty line at file-end

df.tail()
```

[221]:

	sepal_len	sepal_wid	petal_len	petal_wid	class
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

[ ]:
```
# import pandas as pd
# import numpy as np
# df = pd.read_csv\
# (r'C:\Users\lenovo\anaconda3\pkgs\bokeh-3.3.
  ↪4-py311h746a85d_0\Lib\site-packages\bokeh\sampledata\_data\iris.csv')
# df.head()
```

[222]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   sepal_len  150 non-null    float64
 1   sepal_wid  150 non-null    float64
 2   petal_len  150 non-null    float64
 3   petal_wid  150 non-null    float64
 4   class      150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

[223]:
```
# split data table into data X and class labels y
X = df.iloc[:,0:4].values
y = df.iloc[:,4].values
```

[54]:
```
X
```

[54]:
```
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],
[5. , 3. , 1.6, 0.2],
[5. , 3.4, 1.6, 0.4],
[5.2, 3.5, 1.5, 0.2],
[5.2, 3.4, 1.4, 0.2],
[4.7, 3.2, 1.6, 0.2],
[4.8, 3.1, 1.6, 0.2],
[5.4, 3.4, 1.5, 0.4],
[5.2, 4.1, 1.5, 0.1],
[5.5, 4.2, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.2],
[5. , 3.2, 1.2, 0.2],
[5.5, 3.5, 1.3, 0.2],
[4.9, 3.6, 1.4, 0.1],
[4.4, 3. , 1.3, 0.2],
[5.1, 3.4, 1.5, 0.2],
[5. , 3.5, 1.3, 0.3],
[4.5, 2.3, 1.3, 0.3],
[4.4, 3.2, 1.3, 0.2],
[5. , 3.5, 1.6, 0.6],
[5.1, 3.8, 1.9, 0.4],
[4.8, 3. , 1.4, 0.3],
[5.1, 3.8, 1.6, 0.2],
[4.6, 3.2, 1.4, 0.2],
[5.3, 3.7, 1.5, 0.2],
[5. , 3.3, 1.4, 0.2],
[7. , 3.2, 4.7, 1.4],
[6.4, 3.2, 4.5, 1.5],
[6.9, 3.1, 4.9, 1.5],
[5.5, 2.3, 4. , 1.3],
```

```
[6.5, 2.8, 4.6, 1.5],
[5.7, 2.8, 4.5, 1.3],
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
```

```
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
[6.8, 3. , 5.5, 2.1],
[5.7, 2.5, 5. , 2. ],
[5.8, 2.8, 5.1, 2.4],
[6.4, 3.2, 5.3, 2.3],
[6.5, 3. , 5.5, 1.8],
[7.7, 3.8, 6.7, 2.2],
[7.7, 2.6, 6.9, 2.3],
[6. , 2.2, 5. , 1.5],
[6.9, 3.2, 5.7, 2.3],
[5.6, 2.8, 4.9, 2. ],
[7.7, 2.8, 6.7, 2. ],
[6.3, 2.7, 4.9, 1.8],
[6.7, 3.3, 5.7, 2.1],
[7.2, 3.2, 6. , 1.8],
[6.2, 2.8, 4.8, 1.8],
[6.1, 3. , 4.9, 1.8],
[6.4, 2.8, 5.6, 2.1],
[7.2, 3. , 5.8, 1.6],
[7.4, 2.8, 6.1, 1.9],
[7.9, 3.8, 6.4, 2. ],
[6.4, 2.8, 5.6, 2.2],
[6.3, 2.8, 5.1, 1.5],
[6.1, 2.6, 5.6, 1.4],
[7.7, 3. , 6.1, 2.3],
[6.3, 3.4, 5.6, 2.4],
[6.4, 3.1, 5.5, 1.8],
[6. , 3. , 4.8, 1.8],
[6.9, 3.1, 5.4, 2.1],
[6.7, 3.1, 5.6, 2.4],
[6.9, 3.1, 5.1, 2.3],
[5.8, 2.7, 5.1, 1.9],
[6.8, 3.2, 5.9, 2.3],
[6.7, 3.3, 5.7, 2.5],
[6.7, 3. , 5.2, 2.3],
[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]])
```

[178]:
```
y
```

[178]:
```
array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
```

```
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica'], dtype=object)
```

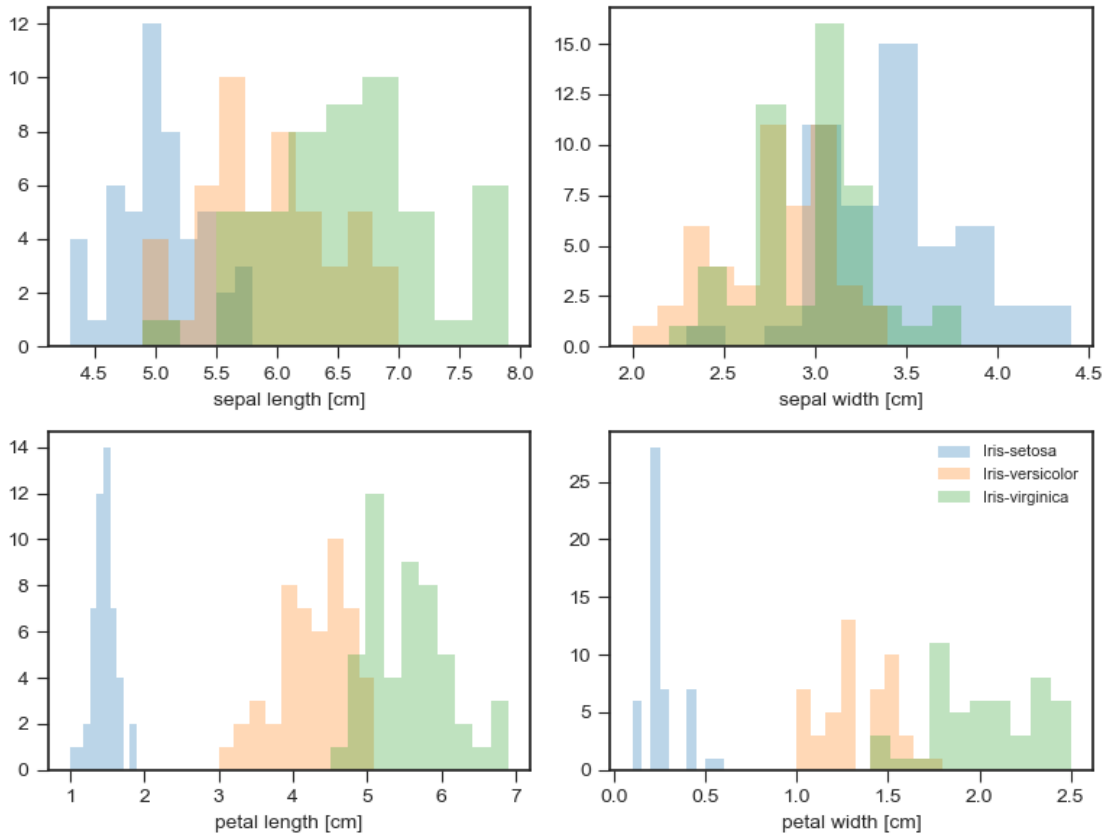
1.1 Exploratory Visualization

[225]:
```
from matplotlib import pyplot as plt
import numpy as np
import math

label_dict = {1: 'Iris-Setosa',
              2: 'Iris-Versicolor',
              3: 'Iris-Virgnica'}

feature_dict = {0: 'sepal length [cm]',
                1: 'sepal width [cm]',
                2: 'petal length [cm]',
                3: 'petal width [cm]'}
#temporarily apply a specific plotting style (seaborn-v0_8-ticks) within a with␣
 ↪block.
with plt.style.context('seaborn-v0_8-ticks'):
    plt.figure(figsize=(8, 6))
    for cnt in range(4):
        plt.subplot(2, 2, cnt+1)
        for lab in ('Iris-setosa', 'Iris-versicolor', 'Iris-virginica'):
            plt.hist(X[y==lab, cnt],
                     label=lab,
                     bins=10,
                     alpha=0.3,)
        plt.xlabel(feature_dict[cnt])
    plt.legend(loc='upper right', fancybox=True, fontsize=8)

    plt.tight_layout()
    plt.show()
```



[56]:
```
# you can see different styles available:
import matplotlib.pyplot as plt
print(plt.style.available)
```

```
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery', '_mpl-gallery-
nogrid', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight',
'ggplot', 'grayscale', 'seaborn-v0_8', 'seaborn-v0_8-bright',
'seaborn-v0_8-colorblind', 'seaborn-v0_8-dark', 'seaborn-v0_8-dark-palette',
'seaborn-v0_8-darkgrid', 'seaborn-v0_8-deep', 'seaborn-v0_8-muted',
'seaborn-v0_8-notebook', 'seaborn-v0_8-paper', 'seaborn-v0_8-pastel',
'seaborn-v0_8-poster', 'seaborn-v0_8-talk', 'seaborn-v0_8-ticks',
'seaborn-v0_8-white', 'seaborn-v0_8-whitegrid', 'tableau-colorblind10']
```

1.2 Standardizing the data

[142]:
```
from sklearn.preprocessing import StandardScaler
X_std = StandardScaler().fit_transform(X)
X_std
```

```
[142]: array([[-9.0068e-01,  1.0190e+00, -1.3402e+00, -1.3154e+00],
               [-1.1430e+00, -1.3198e-01, -1.3402e+00, -1.3154e+00],
               [-1.3854e+00,  3.2841e-01, -1.3971e+00, -1.3154e+00],
               [-1.5065e+00,  9.8217e-02, -1.2834e+00, -1.3154e+00],
               [-1.0218e+00,  1.2492e+00, -1.3402e+00, -1.3154e+00],
               [-5.3718e-01,  1.9398e+00, -1.1697e+00, -1.0522e+00],
               [-1.5065e+00,  7.8881e-01, -1.3402e+00, -1.1838e+00],
               [-1.0218e+00,  7.8881e-01, -1.2834e+00, -1.3154e+00],
               [-1.7489e+00, -3.6218e-01, -1.3402e+00, -1.3154e+00],
               [-1.1430e+00,  9.8217e-02, -1.2834e+00, -1.4471e+00],
               [-5.3718e-01,  1.4794e+00, -1.2834e+00, -1.3154e+00],
               [-1.2642e+00,  7.8881e-01, -1.2266e+00, -1.3154e+00],
               [-1.2642e+00, -1.3198e-01, -1.3402e+00, -1.4471e+00],
               [-1.8700e+00, -1.3198e-01, -1.5107e+00, -1.4471e+00],
               [-5.2506e-02,  2.1700e+00, -1.4539e+00, -1.3154e+00],
               [-1.7367e-01,  3.0908e+00, -1.2834e+00, -1.0522e+00],
               [-5.3718e-01,  1.9398e+00, -1.3971e+00, -1.0522e+00],
               [-9.0068e-01,  1.0190e+00, -1.3402e+00, -1.1838e+00],
               [-1.7367e-01,  1.7096e+00, -1.1697e+00, -1.1838e+00],
               [-9.0068e-01,  1.7096e+00, -1.2834e+00, -1.1838e+00],
               [-5.3718e-01,  7.8881e-01, -1.1697e+00, -1.3154e+00],
               [-9.0068e-01,  1.4794e+00, -1.2834e+00, -1.0522e+00],
               [-1.5065e+00,  1.2492e+00, -1.5676e+00, -1.3154e+00],
               [-9.0068e-01,  5.5861e-01, -1.1697e+00, -9.2055e-01],
               [-1.2642e+00,  7.8881e-01, -1.0560e+00, -1.3154e+00],
               [-1.0218e+00, -1.3198e-01, -1.2266e+00, -1.3154e+00],
               [-1.0218e+00,  7.8881e-01, -1.2266e+00, -1.0522e+00],
               [-7.7951e-01,  1.0190e+00, -1.2834e+00, -1.3154e+00],
               [-7.7951e-01,  7.8881e-01, -1.3402e+00, -1.3154e+00],
               [-1.3854e+00,  3.2841e-01, -1.2266e+00, -1.3154e+00],
               [-1.2642e+00,  9.8217e-02, -1.2266e+00, -1.3154e+00],
               [-5.3718e-01,  7.8881e-01, -1.2834e+00, -1.0522e+00],
               [-7.7951e-01,  2.4002e+00, -1.2834e+00, -1.4471e+00],
               [-4.1601e-01,  2.6304e+00, -1.3402e+00, -1.3154e+00],
               [-1.1430e+00,  9.8217e-02, -1.2834e+00, -1.3154e+00],
               [-1.0218e+00,  3.2841e-01, -1.4539e+00, -1.3154e+00],
               [-4.1601e-01,  1.0190e+00, -1.3971e+00, -1.3154e+00],
               [-1.1430e+00,  1.2492e+00, -1.3402e+00, -1.4471e+00],
               [-1.7489e+00, -1.3198e-01, -1.3971e+00, -1.3154e+00],
               [-9.0068e-01,  7.8881e-01, -1.2834e+00, -1.3154e+00],
               [-1.0218e+00,  1.0190e+00, -1.3971e+00, -1.1838e+00],
               [-1.6277e+00, -1.7434e+00, -1.3971e+00, -1.1838e+00],
               [-1.7489e+00,  3.2841e-01, -1.3971e+00, -1.3154e+00],
               [-1.0218e+00,  1.0190e+00, -1.2266e+00, -7.8892e-01],
               [-9.0068e-01,  1.7096e+00, -1.0560e+00, -1.0522e+00],
               [-1.2642e+00, -1.3198e-01, -1.3402e+00, -1.1838e+00],
               [-9.0068e-01,  1.7096e+00, -1.2266e+00, -1.3154e+00],
```

```
[-1.5065e+00,  3.2841e-01, -1.3402e+00, -1.3154e+00],
[-6.5835e-01,  1.4794e+00, -1.2834e+00, -1.3154e+00],
[-1.0218e+00,  5.5861e-01, -1.3402e+00, -1.3154e+00],
[ 1.4015e+00,  3.2841e-01,  5.3541e-01,  2.6414e-01],
[ 6.7450e-01,  3.2841e-01,  4.2173e-01,  3.9577e-01],
[ 1.2803e+00,  9.8217e-02,  6.4908e-01,  3.9577e-01],
[-4.1601e-01, -1.7434e+00,  1.3755e-01,  1.3251e-01],
[ 7.9567e-01, -5.9237e-01,  4.7857e-01,  3.9577e-01],
[-1.7367e-01, -5.9237e-01,  4.2173e-01,  1.3251e-01],
[ 5.5333e-01,  5.5861e-01,  5.3541e-01,  5.2741e-01],
[-1.1430e+00, -1.5132e+00, -2.6032e-01, -2.6239e-01],
[ 9.1684e-01, -3.6218e-01,  4.7857e-01,  1.3251e-01],
[-7.7951e-01, -8.2257e-01,  8.0709e-02,  2.6414e-01],
[-1.0218e+00, -2.4339e+00, -1.4664e-01, -2.6239e-01],
[ 6.8662e-02, -1.3198e-01,  2.5122e-01,  3.9577e-01],
[ 1.8983e-01, -1.9736e+00,  1.3755e-01, -2.6239e-01],
[ 3.1100e-01, -3.6218e-01,  5.3541e-01,  2.6414e-01],
[-2.9484e-01, -3.6218e-01, -8.9803e-02,  1.3251e-01],
[ 1.0380e+00,  9.8217e-02,  3.6490e-01,  2.6414e-01],
[-2.9484e-01, -1.3198e-01,  4.2173e-01,  3.9577e-01],
[-5.2506e-02, -8.2257e-01,  1.9438e-01, -2.6239e-01],
[ 4.3217e-01, -1.9736e+00,  4.2173e-01,  3.9577e-01],
[-2.9484e-01, -1.2830e+00,  8.0709e-02, -1.3075e-01],
[ 6.8662e-02,  3.2841e-01,  5.9225e-01,  7.9067e-01],
[ 3.1100e-01, -5.9237e-01,  1.3755e-01,  1.3251e-01],
[ 5.5333e-01, -1.2830e+00,  6.4908e-01,  3.9577e-01],
[ 3.1100e-01, -5.9237e-01,  5.3541e-01,  8.7755e-04],
[ 6.7450e-01, -3.6218e-01,  3.0806e-01,  1.3251e-01],
[ 9.1684e-01, -1.3198e-01,  3.6490e-01,  2.6414e-01],
[ 1.1592e+00, -5.9237e-01,  5.9225e-01,  2.6414e-01],
[ 1.0380e+00, -1.3198e-01,  7.0592e-01,  6.5904e-01],
[ 1.8983e-01, -3.6218e-01,  4.2173e-01,  3.9577e-01],
[-1.7367e-01, -1.0528e+00, -1.4664e-01, -2.6239e-01],
[-4.1601e-01, -1.5132e+00,  2.3872e-02, -1.3075e-01],
[-4.1601e-01, -1.5132e+00, -3.2966e-02, -2.6239e-01],
[-5.2506e-02, -8.2257e-01,  8.0709e-02,  8.7755e-04],
[ 1.8983e-01, -8.2257e-01,  7.6276e-01,  5.2741e-01],
[-5.3718e-01, -1.3198e-01,  4.2173e-01,  3.9577e-01],
[ 1.8983e-01,  7.8881e-01,  4.2173e-01,  5.2741e-01],
[ 1.0380e+00,  9.8217e-02,  5.3541e-01,  3.9577e-01],
[ 5.5333e-01, -1.7434e+00,  3.6490e-01,  1.3251e-01],
[-2.9484e-01, -1.3198e-01,  1.9438e-01,  1.3251e-01],
[-4.1601e-01, -1.2830e+00,  1.3755e-01,  1.3251e-01],
[-4.1601e-01, -1.0528e+00,  3.6490e-01,  8.7755e-04],
[ 3.1100e-01, -1.3198e-01,  4.7857e-01,  2.6414e-01],
[-5.2506e-02, -1.0528e+00,  1.3755e-01,  8.7755e-04],
[-1.0218e+00, -1.7434e+00, -2.6032e-01, -2.6239e-01],
```

```
[-2.9484e-01, -8.2257e-01,  2.5122e-01,  1.3251e-01],
[-1.7367e-01, -1.3198e-01,  2.5122e-01,  8.7755e-04],
[-1.7367e-01, -3.6218e-01,  2.5122e-01,  1.3251e-01],
[ 4.3217e-01, -3.6218e-01,  3.0806e-01,  1.3251e-01],
[-9.0068e-01, -1.2830e+00, -4.3083e-01, -1.3075e-01],
[-1.7367e-01, -5.9237e-01,  1.9438e-01,  1.3251e-01],
[ 5.5333e-01,  5.5861e-01,  1.2743e+00,  1.7121e+00],
[-5.2506e-02, -8.2257e-01,  7.6276e-01,  9.2230e-01],
[ 1.5227e+00, -1.3198e-01,  1.2175e+00,  1.1856e+00],
[ 5.5333e-01, -3.6218e-01,  1.0469e+00,  7.9067e-01],
[ 7.9567e-01, -1.3198e-01,  1.1606e+00,  1.3172e+00],
[ 2.1285e+00, -1.3198e-01,  1.6153e+00,  1.1856e+00],
[-1.1430e+00, -1.2830e+00,  4.2173e-01,  6.5904e-01],
[ 1.7650e+00, -3.6218e-01,  1.4448e+00,  7.9067e-01],
[ 1.0380e+00, -1.2830e+00,  1.1606e+00,  7.9067e-01],
[ 1.6438e+00,  1.2492e+00,  1.3311e+00,  1.7121e+00],
[ 7.9567e-01,  3.2841e-01,  7.6276e-01,  1.0539e+00],
[ 6.7450e-01, -8.2257e-01,  8.7643e-01,  9.2230e-01],
[ 1.1592e+00, -1.3198e-01,  9.9011e-01,  1.1856e+00],
[-1.7367e-01, -1.2830e+00,  7.0592e-01,  1.0539e+00],
[-5.2506e-02, -5.9237e-01,  7.6276e-01,  1.5805e+00],
[ 6.7450e-01,  3.2841e-01,  8.7643e-01,  1.4488e+00],
[ 7.9567e-01, -1.3198e-01,  9.9011e-01,  7.9067e-01],
[ 2.2497e+00,  1.7096e+00,  1.6722e+00,  1.3172e+00],
[ 2.2497e+00, -1.0528e+00,  1.7858e+00,  1.4488e+00],
[ 1.8983e-01, -1.9736e+00,  7.0592e-01,  3.9577e-01],
[ 1.2803e+00,  3.2841e-01,  1.1038e+00,  1.4488e+00],
[-2.9484e-01, -5.9237e-01,  6.4908e-01,  1.0539e+00],
[ 2.2497e+00, -5.9237e-01,  1.6722e+00,  1.0539e+00],
[ 5.5333e-01, -8.2257e-01,  6.4908e-01,  7.9067e-01],
[ 1.0380e+00,  5.5861e-01,  1.1038e+00,  1.1856e+00],
[ 1.6438e+00,  3.2841e-01,  1.2743e+00,  7.9067e-01],
[ 4.3217e-01, -5.9237e-01,  5.9225e-01,  7.9067e-01],
[ 3.1100e-01, -1.3198e-01,  6.4908e-01,  7.9067e-01],
[ 6.7450e-01, -5.9237e-01,  1.0469e+00,  1.1856e+00],
[ 1.6438e+00, -1.3198e-01,  1.1606e+00,  5.2741e-01],
[ 1.8862e+00, -5.9237e-01,  1.3311e+00,  9.2230e-01],
[ 2.4920e+00,  1.7096e+00,  1.5016e+00,  1.0539e+00],
[ 6.7450e-01, -5.9237e-01,  1.0469e+00,  1.3172e+00],
[ 5.5333e-01, -5.9237e-01,  7.6276e-01,  3.9577e-01],
[ 3.1100e-01, -1.0528e+00,  1.0469e+00,  2.6414e-01],
[ 2.2497e+00, -1.3198e-01,  1.3311e+00,  1.4488e+00],
[ 5.5333e-01,  7.8881e-01,  1.0469e+00,  1.5805e+00],
[ 6.7450e-01,  9.8217e-02,  9.9011e-01,  7.9067e-01],
[ 1.8983e-01, -1.3198e-01,  5.9225e-01,  7.9067e-01],
[ 1.2803e+00,  9.8217e-02,  9.3327e-01,  1.1856e+00],
[ 1.0380e+00,  9.8217e-02,  1.0469e+00,  1.5805e+00],
```

```
 [ 1.2803e+00,  9.8217e-02,  7.6276e-01,  1.4488e+00],
 [-5.2506e-02, -8.2257e-01,  7.6276e-01,  9.2230e-01],
 [ 1.1592e+00,  3.2841e-01,  1.2175e+00,  1.4488e+00],
 [ 1.0380e+00,  5.5861e-01,  1.1038e+00,  1.7121e+00],
 [ 1.0380e+00, -1.3198e-01,  8.1960e-01,  1.4488e+00],
 [ 5.5333e-01, -1.2830e+00,  7.0592e-01,  9.2230e-01],
 [ 7.9567e-01, -1.3198e-01,  8.1960e-01,  1.0539e+00],
 [ 4.3217e-01,  7.8881e-01,  9.3327e-01,  1.4488e+00],
 [ 6.8662e-02, -1.3198e-01,  7.6276e-01,  7.9067e-01]])
```

1.3 1 - Eigendecomposition - Computing Eigenvectors and Eigenvalues

[ ]:
```
# Calulate covraince matrix
```

[179]:
```
import numpy as np
mean_vec = np.mean(X_std, axis=0)
cov_mat = (X_std - mean_vec).T.dot((X_std - mean_vec)) / (X_std.shape[0]-1)
print('Covariance matrix \n%s' %cov_mat)
#.shape[0] → Returns the number of rows
```

```
Covariance matrix
[[ 1.0067 -0.1184  0.8776  0.8234]
 [-0.1184  1.0067 -0.4313 -0.3686]
 [ 0.8776 -0.4313  1.0067  0.9693]
 [ 0.8234 -0.3686  0.9693  1.0067]]
```

[180]:
```
X_std.shape[0]
```

[180]:
```
150
```

[227]:
```
#or directly you can use cov function

cov_mat = np.cov(X_std.T)
print('NumPy covariance matrix: \n%s' %np.cov(X_std.T))
```

```
NumPy covariance matrix:
[[ 1.0067 -0.1184  0.8776  0.8234]
 [-0.1184  1.0067 -0.4313 -0.3686]
 [ 0.8776 -0.4313  1.0067  0.9693]
 [ 0.8234 -0.3686  0.9693  1.0067]]
```

[183]:
```
# Calculate eigen values and eigen vectors
#np.linalg.eig is a function in NumPy used to compute the eigenvalues and␣
 ↪eigenvectors of a square matrix.

eig_vals, eig_vecs = np.linalg.eig(cov_mat)
print('Eigenvectors \n%s' %eig_vecs)
```

```
print('\nEigenvalues \n%s' %eig_vals)
```

```
Eigenvectors
[[ 0.5211 -0.3774 -0.7196  0.2613]
 [-0.2693 -0.9233  0.2444 -0.1235]
 [ 0.5804 -0.0245  0.1421 -0.8014]
 [ 0.5649 -0.0669  0.6343  0.5236]]

Eigenvalues
[2.9381 0.9202 0.1477 0.0209]
```

1.3.1 Correlation Matrix

Especially, in the field of "Finance," the correlation matrix typically used instead of the covariance matrix. However, the eigendecomposition of the covariance matrix (if the input data was standardized) yields the same results as a eigendecomposition on the correlation matrix, since the correlation matrix can be understood as the normalized covariance matrix.

1.3.2 Eigendecomposition of the standardized data based on the correlation matrix:

np.corrcoef() computes the Pearson correlation coefficient between variables. It measures how strongly two variables are related, ranging from -1 (perfect negative correlation) to 1 (perfect positive correlation).

[148]:
```
cor_mat1 = np.corrcoef(X_std.T)

eig_vals, eig_vecs = np.linalg.eig(cor_mat1)

print('Eigenvectors \n%s' %eig_vecs)
print('\nEigenvalues \n%s' %eig_vals)
```

```
Eigenvectors
[[ 0.5211 -0.3774 -0.7196  0.2613]
 [-0.2693 -0.9233  0.2444 -0.1235]
 [ 0.5804 -0.0245  0.1421 -0.8014]
 [ 0.5649 -0.0669  0.6343  0.5236]]

Eigenvalues
[2.9185 0.914  0.1468 0.0207]
```

1.3.3 Eigendecomposition of the raw data based on the correlation matrix:

[187]:
```
cor_mat2 = np.corrcoef(X.T)

eig_vals, eig_vecs = np.linalg.eig(cor_mat2)

print('Eigenvectors \n%s' %eig_vecs)
print('\nEigenvalues \n%s' %eig_vals)
```

```
Eigenvectors
[[ 0.5224 -0.3723 -0.721   0.262 ]
 [-0.2634 -0.9256  0.242  -0.1241]
 [ 0.5813 -0.0211  0.1409 -0.8012]
 [ 0.5656 -0.0654  0.6338  0.5235]]

Eigenvalues
[2.9108 0.9212 0.1474 0.0206]
```

We can clearly see that all three approaches yield the same eigenvectors and eigenvalue pairs: (1) Eigendecomposition of the covariance matrix after standardizing the data. (2) Eigendecomposition of the correlation matrix. (3) Eigendecomposition of the correlation matrix after standardizing the data.

1.3.4 Singular Value Decomposition

While the eigendecomposition of the covariance or correlation matrix may be more intuitiuve, most PCA implementations perform a Singular Value Decomposition (SVD) to improve the computational efficiency. So, let us perform an SVD to confirm that the result are indeed the same:

[150]:
```
u,s,v = np.linalg.svd(X_std.T)
u
```

[150]:
```
array([[-0.5211, -0.3774,  0.7196,  0.2613],
       [ 0.2693, -0.9233, -0.2444, -0.1235],
       [-0.5804, -0.0245, -0.1421, -0.8014],
       [-0.5649, -0.0669, -0.6343,  0.5236]])
```

1.3.5 2 - Selecting Principal Components

In order to decide which eigenvector(s) can dropped without losing too much information for the construction of lower-dimensional subspace, we need to inspect the corresponding eigenvalues: The eigenvectors with the lowest eigenvalues bear the least information about the distribution of the data; those are the ones can be dropped. In order to do so, the common approach is to rank the eigenvalues from highest to lowest in order choose the top k eigenvectors.

1) eig_vals: An array of eigenvalues obtained from np.linalg.eig().
2) eig_vecs: A matrix of eigenvectors, where each column corresponds to an eigenvector.
3) eig_vals[i] gives the i-th eigenvalue.
4) eig_vecs[:, i] gives the i-th eigenvector.
5) np.abs(eig_vals[i]) (Absolute Eigenvalues): Eigenvalues can sometimes be negative due to numerical issues, but in PCA, we consider variance, which is always non-negative.
6) np.abs(eig_vals[i]) ensures all eigenvalues are positive.

[189]:
```
eig_vals
```

[189]:
```
array([2.9108, 0.9212, 0.1474, 0.0206])
```

[191]:
```
eig_vecs
```

[191]:
```
array([[ 0.5224, -0.3723, -0.721 ,  0.262 ],
       [-0.2634, -0.9256,  0.242 , -0.1241],
       [ 0.5813, -0.0211,  0.1409, -0.8012],
       [ 0.5656, -0.0654,  0.6338,  0.5235]])
```

[194]:
```
# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]
```

[152]:
```
eig_pairs
```

[152]:
```
[(2.918497816531995, array([ 0.5211, -0.2693,  0.5804,  0.5649])),
 (0.91403047146807, array([-0.3774, -0.9233, -0.0245, -0.0669])),
 (0.1467568755713152, array([-0.7196,  0.2444,  0.1421,  0.6343])),
 (0.020714836428619227, array([ 0.2613, -0.1235, -0.8014,  0.5236]))]
```

1) eig_pairs is a list of tuples, where:
2) eig_pairs[i][0] is an eigenvalue.
3) eig_pairs[i][1] is the corresponding eigenvector.
4) key=lambda x: x[0]: Uses the eigenvalue (first element of each tuple) as the sorting key.
5) reverse=True: Sorts from highest to lowest eigenvalue (descending order).

[228]:
```
# Sort the (eigenvalue, eigenvector) tuples from high to low
#lambda x: x[0] is a lambda function (anonymous function) that takes an input x␣
 ↪and returns the first element (x[0]) of x.
eig_pairs.sort(key=lambda x: x[0], reverse=True)
eig_pairs
```

[228]:
```
[(2.9108180837520536, array([ 0.5224, -0.2634,  0.5813,  0.5656])),
 (0.9212209307072243, array([-0.3723, -0.9256, -0.0211, -0.0654])),
 (0.1473532783050957, array([-0.721 ,  0.242 ,  0.1409,  0.6338])),
 (0.020607707235624852, array([ 0.262 , -0.1241, -0.8012,  0.5235]))]
```

[229]:
```
# Visually confirm that the list is correctly sorted by decreasing eigenvalues
print('Eigenvalues in descending order:')
for i in eig_pairs:
    print(i[0])
```

```
Eigenvalues in descending order:
2.9108180837520536
0.9212209307072243
0.1473532783050957
0.020607707235624852
```

1.3.6 Explained Variance

After sorting the eigenpairs, the next question is "how many principal components are we going to choose for our new feature subspace?" A useful measure is the so-called "explained variance,"

which can be calculated from the eigenvalues. The explained variance tells us how much information (variance) can be attributed to each of the principal components.

[230]:
```
tot = sum(eig_vals)
var_exp = [(i / tot)*100 for i in sorted(eig_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
```

1) eig_vals is a list or array of eigenvalues from the covariance matrix.
2) tot stores the sum of all eigenvalues, which represents the total variance in the dataset.
3) sorted(eig_vals, reverse=True): Sorts eigenvalues in descending order (largest variance first).
4) (i / tot) * 100: Computes the percentage of variance explained by each eigenvalue.
5) var_exp: Stores the explained variance ratio for each principal component.
6) np.cumsum(var_exp): Computes the cumulative sum of explained variance.
7) cum_var_exp[i]: Represents the total variance explained by the first (i+1) principal components.

[231]:
```
var_exp
```

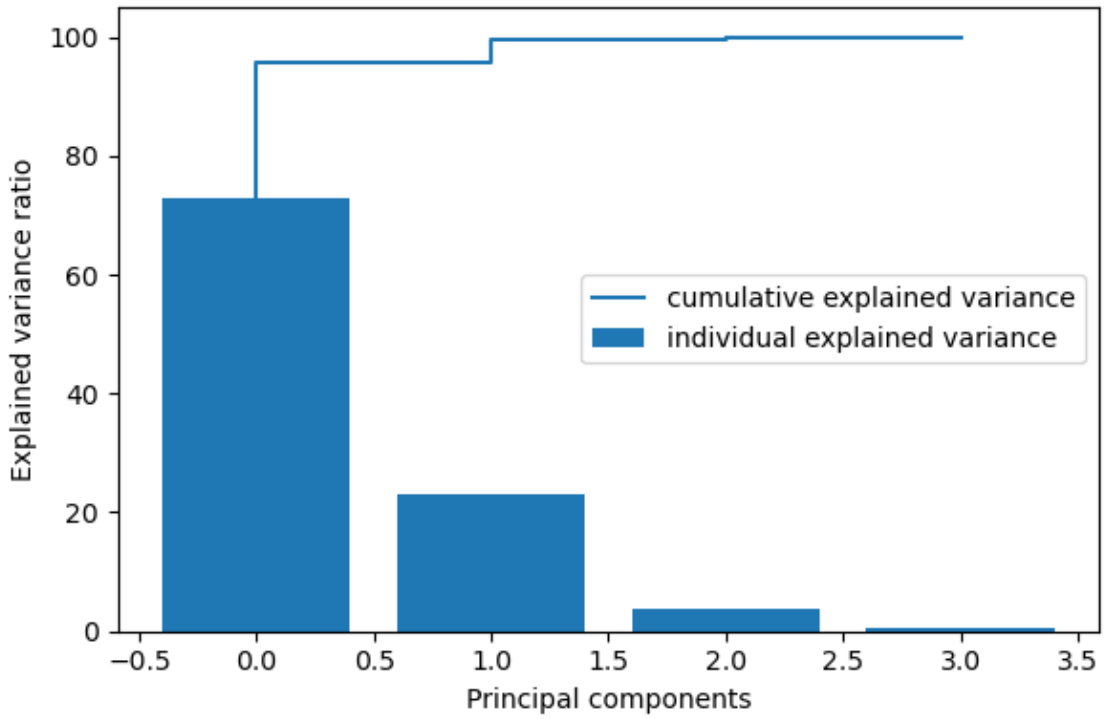
[231]:
```
[72.77045209380137, 23.030523267680618, 3.683831957627394, 0.5151926808906215]
```

[232]:
```
cum_var_exp
```

[232]:
```
array([ 72.7705,  95.801 ,  99.4848, 100.    ])
```

[199]:
```
plt.figure(figsize=(6, 4))

    plt.bar(range(4), var_exp,align='center',
            label='individual explained variance')
    plt.step(range(4), cum_var_exp,
             label='cumulative explained variance')
    plt.ylabel('Explained variance ratio')
    plt.xlabel('Principal components')
    plt.legend(loc='best')
    plt.tight_layout()
```



The plot above clearly shows that most of the variance (72.77% of the variance to be precise) can be explained by the first principal component alone. The second principal component still bears some information (23.03%) while the third and fourth principal components can safely be dropped without losing to much information. Together, the first two principal components contain 95.8% of the information.

1.3.7 3 - Projection Onto the New Feature Space

It's about time to get to the really interesting part: The construction of the projection matrix that will be used to transform the Iris data onto the new feature subspace. Although, the name "projection matrix" has a nice ring to it, it is basically just a matrix of our concatenated top k eigenvectors.

Here, we are reducing the 4-dimensional feature space to a 2-dimensional feature subspace, by choosing the "top 2" eigenvectors with the highest eigenvalues to construct our d×k -dimensional eigenvector matrix W .

[236]:
```
eig_pairs
```

[236]:
```
[(2.9108180837520536, array([ 0.5224, -0.2634,  0.5813,  0.5656])),
 (0.9212209307072243, array([-0.3723, -0.9256, -0.0211, -0.0654])),
 (0.1473532783050957, array([-0.721 ,  0.242 ,  0.1409,  0.6338])),
 (0.020607707235624852, array([ 0.262 , -0.1241, -0.8012,  0.5235]))]
```

[235]:
```
eig_pairs[0]
```

[235]:
```
array([ 0.5224, -0.2634,  0.5813,  0.5656])
```

[237]:
```
eig_pairs[1]
```

[237]:
```
(0.9212209307072243, array([-0.3723, -0.9256, -0.0211, -0.0654]))
```

[238]:
```
eig_pairs[0][1]
```

[238]:
```
array([ 0.5224, -0.2634,  0.5813,  0.5656])
```

[239]:
```
eig_pairs[1][1]
```

[239]:
```
array([-0.3723, -0.9256, -0.0211, -0.0654])
```

[242]:
```
eig_pairs[1][1].size
```

[242]:
```
4
```

[157]:
```
#stack() function is used to stack arrays in sequence horizontally (column wise)
matrix_w = np.hstack((eig_pairs[0][1].reshape(4,1),
                      eig_pairs[1][1].reshape(4,1)))
print('Matrix W:\n', matrix_w)
```

```
Matrix W:
 [[ 0.5211 -0.3774]
 [-0.2693 -0.9233]
 [ 0.5804 -0.0245]
 [ 0.5649 -0.0669]]
```

1) eig_pairs[i][0] is the eigenvalue (importance of the principal component).
2) eig_pairs[i][1] is the corresponding eigenvector (direction of the principal component).
3) eig_pairs[0][1].reshape(4,1) # First principal component (Column Vector)
4) eig_pairs[1][1].reshape(4,1) # Second principal component (Column Vector)
5) .reshape(4,1) converts the eigenvectors from row vectors (shape: (4,)) to column vectors (shape: (4,1)) for proper matrix operations.
6) np.hstack() horizontally stacks the top two eigenvectors, forming Matrix W.
7) Matrix W (4×2) contains the top 2 principal components as columns.

1.3.8 3 - Projection Onto the New Feature Space

In this last step we will use the 4×2 -dimensional projection matrix W to transform our samples onto the new subspace via the equation Y=X×W, where Y is a 150×2 matrix of our transformed samples.

[158]:
```
Y = X_std.dot(matrix_w)
Y
```

```
[158]: array([[-2.2647, -0.48  ],
               [-2.081 ,  0.6741],
               [-2.3642,  0.3419],
               [-2.2994,  0.5974],
               [-2.3898, -0.6468],
               [-2.0756, -1.4892],
               [-2.444 , -0.0476],
               [-2.2328, -0.2231],
               [-2.3346,  1.1153],
               [-2.1843,  0.469 ],
               [-2.1663, -1.0437],
               [-2.3261, -0.1331],
               [-2.2185,  0.7287],
               [-2.6331,  0.9615],
               [-2.1987, -1.8601],
               [-2.2622, -2.6863],
               [-2.2076, -1.4836],
               [-2.1903, -0.4888],
               [-1.8986, -1.405 ],
               [-2.3434, -1.1278],
               [-1.9143, -0.4089],
               [-2.207 , -0.9241],
               [-2.7743, -0.4583],
               [-1.8187, -0.0856],
               [-2.2272, -0.1373],
               [-1.9518,  0.6256],
               [-2.0512, -0.2422],
               [-2.1686, -0.5271],
               [-2.1396, -0.3132],
               [-2.2653,  0.3377],
               [-2.1401,  0.5045],
               [-1.8316, -0.4237],
               [-2.6149, -1.7936],
               [-2.4462, -2.1507],
               [-2.11  ,  0.4602],
               [-2.2078,  0.2061],
               [-2.0451, -0.6616],
               [-2.5273, -0.5923],
               [-2.4296,  0.9042],
               [-2.1697, -0.2689],
               [-2.2865, -0.4417],
               [-1.8581,  2.3374],
               [-2.5536,  0.4791],
               [-1.9644, -0.4723],
               [-2.1371, -1.1422],
               [-2.0697,  0.7111],
               [-2.3847, -1.1204],
```

```
[-2.3944,  0.3862],
[-2.2294, -0.998 ],
[-2.2038, -0.0092],
[ 1.1018, -0.863 ],
[ 0.7313, -0.5946],
[ 1.241 , -0.6163],
[ 0.4075,  1.7544],
[ 1.0755,  0.2084],
[ 0.3887,  0.5933],
[ 0.7465, -0.773 ],
[-0.4873,  1.8524],
[ 0.9279, -0.0322],
[ 0.0114,  1.034 ],
[-0.1102,  2.6541],
[ 0.4407,  0.0633],
[ 0.5621,  1.7647],
[ 0.7196,  0.1862],
[-0.0334,  0.439 ],
[ 0.8754, -0.5091],
[ 0.3503,  0.1963],
[ 0.1588,  0.7921],
[ 1.2251,  1.6222],
[ 0.1649,  1.3026],
[ 0.7377, -0.3966],
[ 0.4763,  0.4173],
[ 1.2342,  0.9333],
[ 0.6329,  0.4164],
[ 0.7027,  0.0634],
[ 0.8743, -0.2508],
[ 1.2565,  0.0773],
[ 1.3584, -0.3313],
[ 0.6648,  0.2259],
[-0.0403,  1.0587],
[ 0.1308,  1.5623],
[ 0.0235,  1.5725],
[ 0.2415,  0.7773],
[ 1.0611,  0.6338],
[ 0.224 ,  0.2878],
[ 0.4291, -0.8456],
[ 1.0487, -0.5221],
[ 1.0445,  1.383 ],
[ 0.0696,  0.2195],
[ 0.2835,  1.3293],
[ 0.2791,  1.12  ],
[ 0.6246, -0.0249],
[ 0.3365,  0.9884],
[-0.3622,  2.0192],
```

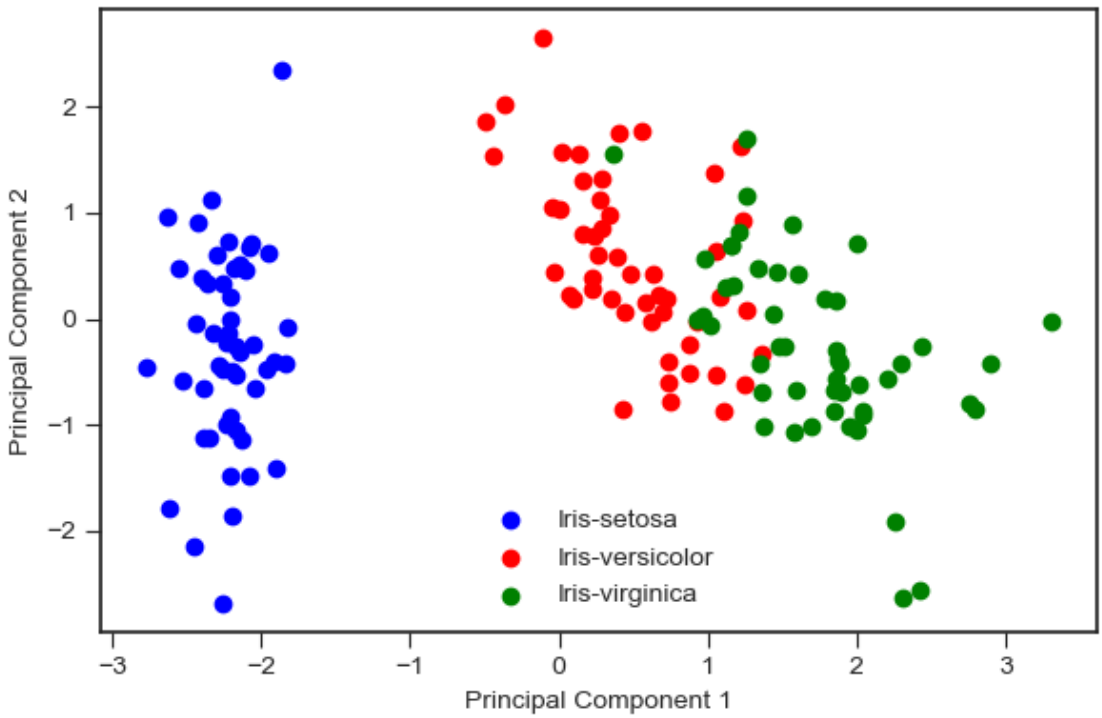
```
[ 0.2886,  0.8557],
[ 0.0914,  0.1812],
[ 0.2277,  0.3849],
[ 0.5764,  0.1549],
[-0.4477,  1.5438],
[ 0.2567,  0.5989],
[ 1.8446, -0.8704],
[ 1.1579,  0.6989],
[ 2.2053, -0.562 ],
[ 1.4402,  0.047 ],
[ 1.8678, -0.295 ],
[ 2.7519, -0.8004],
[ 0.367 ,  1.5615],
[ 2.3024, -0.4201],
[ 2.0067,  0.7114],
[ 2.2598, -1.921 ],
[ 1.3642, -0.6928],
[ 1.6027,  0.4217],
[ 1.8839, -0.4192],
[ 1.2601,  1.1623],
[ 1.4676,  0.4423],
[ 1.5901, -0.6762],
[ 1.4714, -0.2556],
[ 2.4263, -2.5567],
[ 3.3107, -0.0178],
[ 1.2638,  1.7067],
[ 2.0377, -0.9105],
[ 0.978 ,  0.5718],
[ 2.8977, -0.4136],
[ 1.3332,  0.4818],
[ 1.7007, -1.0139],
[ 1.9543, -1.0078],
[ 1.1751,  0.3164],
[ 1.021 , -0.0643],
[ 1.7883,  0.1874],
[ 1.8636, -0.5623],
[ 2.436 , -0.2593],
[ 2.3049, -2.6263],
[ 1.8627,  0.1785],
[ 1.1141,  0.2929],
[ 1.2025,  0.8113],
[ 2.7988, -0.8568],
[ 1.5763, -1.0686],
[ 1.3463, -0.4224],
[ 0.9248, -0.0172],
[ 1.852 , -0.6761],
[ 2.0148, -0.6139],
```

```
        [ 1.9018, -0.6896],
        [ 1.1579,  0.6989],
        [ 2.0406, -0.8675],
        [ 1.9981, -1.0492],
        [ 1.8705, -0.387 ],
        [ 1.5646,  0.8967],
        [ 1.5212, -0.2691],
        [ 1.3728, -1.0113],
        [ 0.9607,  0.0243]])
```

[159]:
```
import matplotlib.pyplot as plt
print(plt.style.available)
```

```
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery', '_mpl-gallery-
nogrid', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight',
'ggplot', 'grayscale', 'seaborn-v0_8', 'seaborn-v0_8-bright',
'seaborn-v0_8-colorblind', 'seaborn-v0_8-dark', 'seaborn-v0_8-dark-palette',
'seaborn-v0_8-darkgrid', 'seaborn-v0_8-deep', 'seaborn-v0_8-muted',
'seaborn-v0_8-notebook', 'seaborn-v0_8-paper', 'seaborn-v0_8-pastel',
'seaborn-v0_8-poster', 'seaborn-v0_8-talk', 'seaborn-v0_8-ticks',
'seaborn-v0_8-white', 'seaborn-v0_8-whitegrid', 'tableau-colorblind10']
```

[218]:
```
#seaborn-v0_8-ticks' is a Seaborn-based Matplotlib style that enhances the look␣
 ↪of tick marks and grids.
with plt.style.context('seaborn-v0_8-ticks'):
    #Creates a new figure with a specific size of 6 inches wide × 4 inches tall.
    # zip() pairs labels (lab) with corresponding colors (col):
    plt.figure(figsize=(6, 4))
    # loops through the three species of the Iris dataset and assigns colors to␣
 ↪each.
    for lab, col in zip(('Iris-setosa', 'Iris-versicolor', 'Iris-virginica'),
                        ('blue', 'red', 'green')):
        plt.scatter(Y[y==lab, 0],
                    Y[y==lab, 1],
                    label=lab,
                    c=col)
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.legend(loc='lower center')
    plt.tight_layout()
    plt.show()
```



1) 0 → First Principal Component (X-axis)
2) 1 → Second Principal Component (Y-axis)
3) X[y == lab, 0] → Filters the first component for a specific species
4) X[y == lab, 1] → Filters the second component for the same species

Now, what we got after applying the linear PCA transformation is a lower dimensional subspace (from 3D to 2D in this case), where the samples are "most spread" along the new feature axes.

2 Shortcut - PCA in scikit-learn

[167]:
```
from sklearn.decomposition import PCA as sklearnPCA
sklearn_pca = sklearnPCA(n_components=2)
Y_sklearn = sklearn_pca.fit_transform(X_std)
```

[168]:
```
with plt.style.context('seaborn-v0_8-ticks'):
    plt.figure(figsize=(6, 4))
    for lab, col in zip(('Iris-setosa', 'Iris-versicolor', 'Iris-virginica'),
                        ('blue', 'red', 'green')):
        plt.scatter(Y_sklearn[y==lab, 0],
                    Y_sklearn[y==lab, 1],
                    label=lab,
                    c=col)
    plt.xlabel('Principal Component 1')
```

```
    plt.ylabel('Principal Component 2')
    plt.legend(loc='lower center')
    plt.tight_layout()
    plt.show()
```