Dimensionality reduction

Feature extraction with **PCA**
(an unsupervised learning technique)

see Ch. 05 in book “Python Machine Learning” by Raschka & Mirjalili

Feature extraction / feature selection

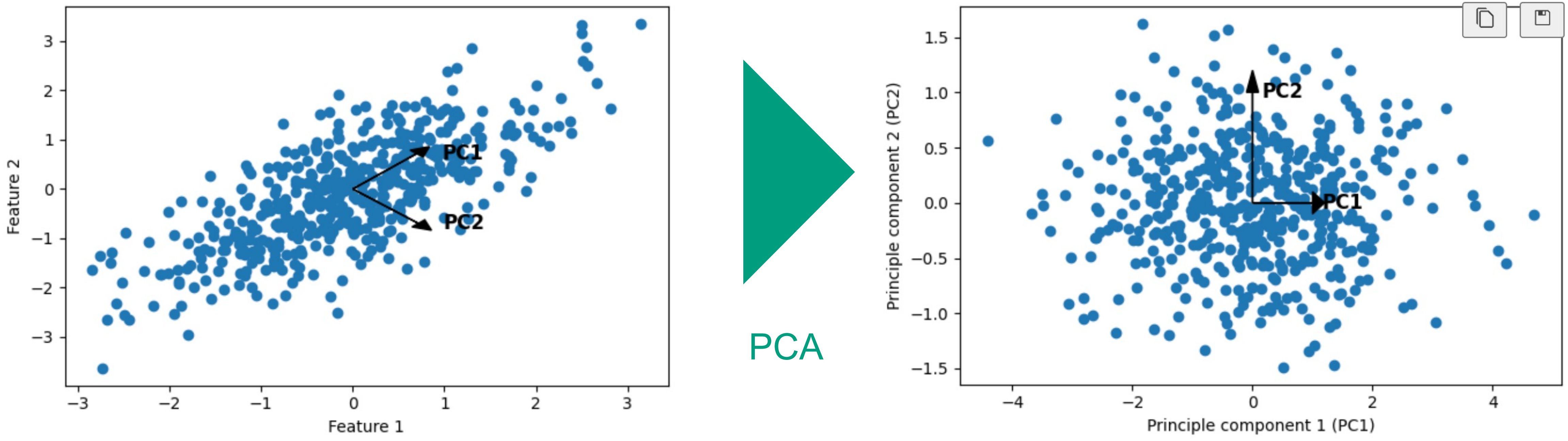
- In the previous lecture, we learned about techniques for **feature selection** (e.g. with SBS/SFS) where we selected a subset of the original features to reduce the number of features
- Here, we learn how to reduce dimensionality of a dataset by **feature extraction:** which describes techniques for computing **new** features from existing features

Unsupervised learning

- The goal of **unsupervised learning** is to **identify hidden patterns** or **structure** in the data
- We only work with the design matrix $\boldsymbol{X}$
- We do not know or do not use the labels/outcome $\boldsymbol{y}$

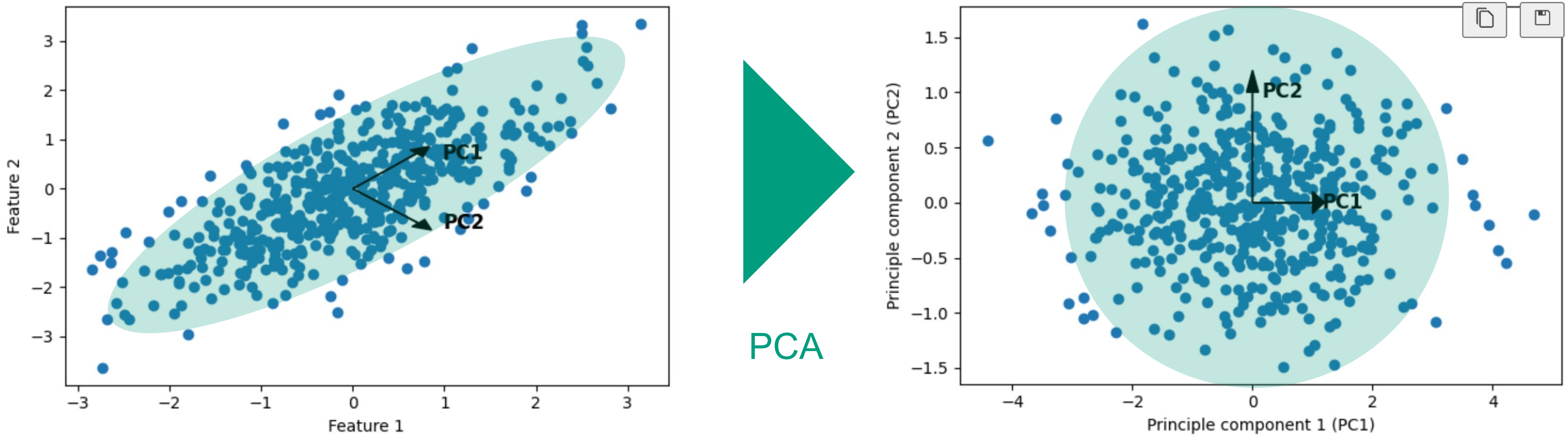
Principal Component Analysis (PCA)

- A **linear transformation** technique often used for **dimensionality reduction**
- **Identifies patterns/structure** in the data based on the **covariance** (centered data) or **correlation** (standardized data) matrix of the data



Principal Component Analysis (PCA)

- A **linear transformation** technique often used for **dimensionality reduction**
- **Identifies patterns/structure** in the data based on the **covariance** (centered data) or **correlation** (standardized data) matrix of the data



Principal Component Analysis (PCA)

- A **linear transformation** technique often used for **dimensionality reduction**
- **Identifies patterns** in the data based on the **covariance** (centered data) or **correlation** (standardized data) matrix of the data
- **Unsupervised learning technique**
- Used for **feature extraction:** computing **new** features from existing features
- In the context of **dimensionality reduction,** we perform **data compression** by finding new features such that a few features contain most of the relevant information

Principal Component Analysis (PCA)

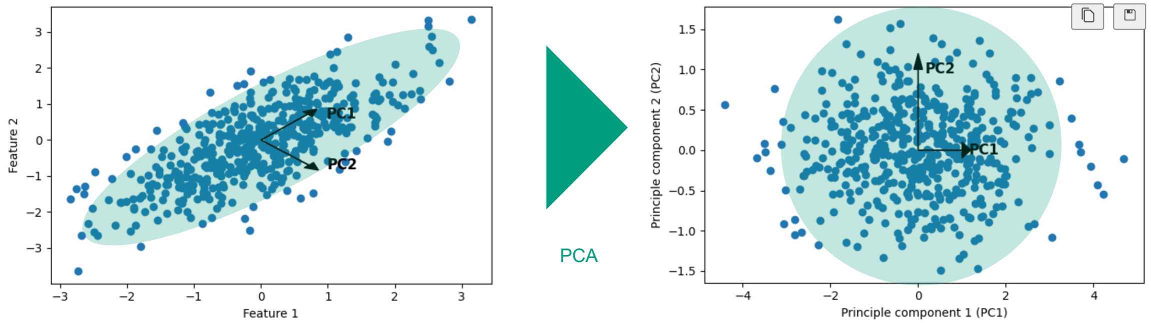
- Applied as **pre-processing technique** for other models and for **data visualization**
  - May help to prevent overfitting (especially for model without regularization option)
  - Denoising of original data
  - Make high-dimensional data suitable for visualization

- Widely used across different fields, examples:
  - Explorative data analysis (EDA)
  - Noise reduction in signal data (e.g. stock market prices, imagine intensity-time signals, etc.)
  - Analysis of genome and gene expression data (very high-dimensional data)

Principal Component Analysis (PCA)

Algorithm is explained in:

`05_PCA.ipynb`



Example applications:

`05_PCA_logreg_wine.ipynb` → Preprocessing / feature extraction

`05_PCA_wine_explained_variance.ipynb` → Feature extraction / importance

`05_PCA_handwritten_digits.ipynb` → Visualization

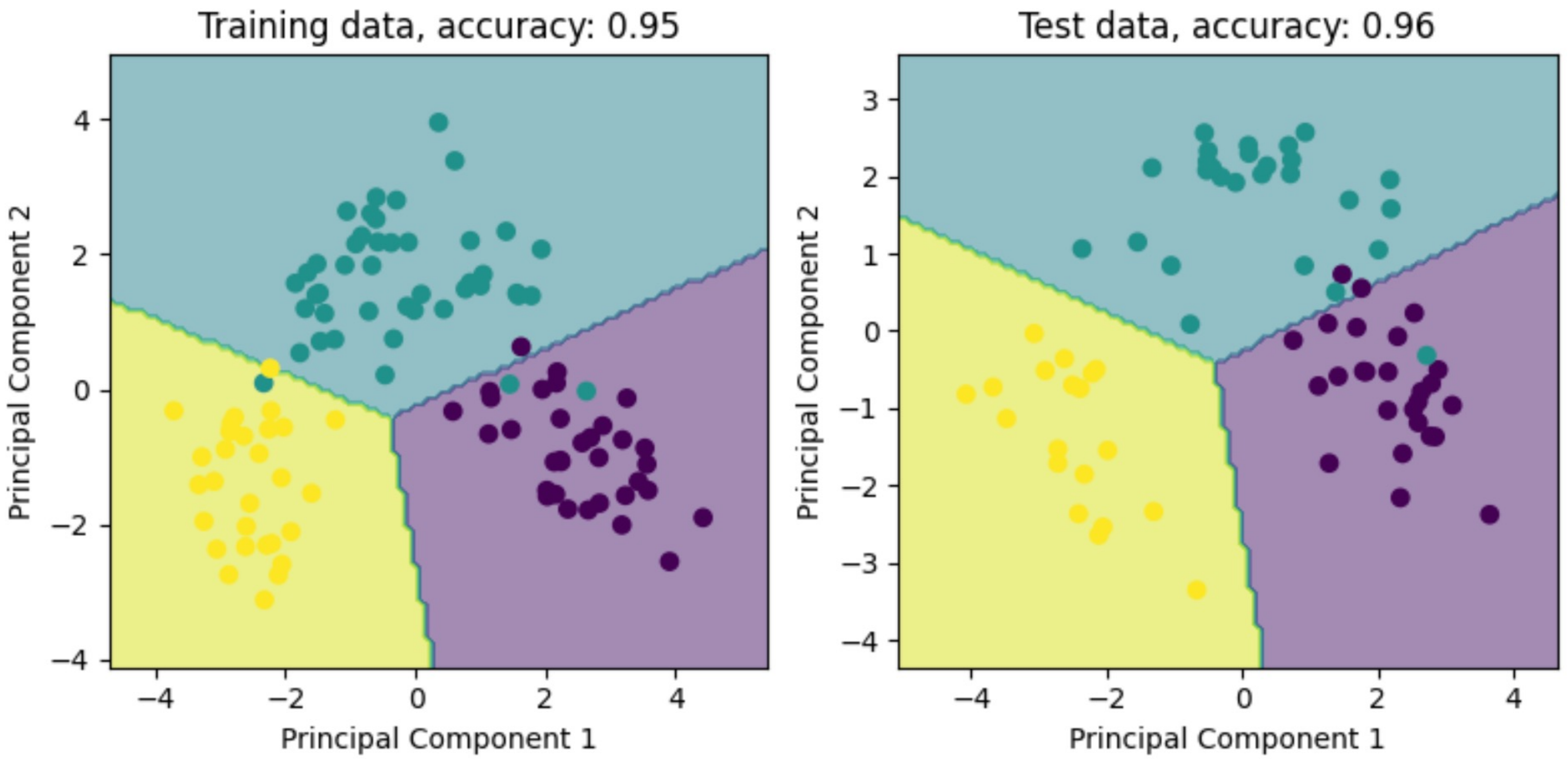
Principal Component Analysis (PCA)

Summary of steps (centering but not standardization is included in `PCA` from `sklearn`)

1. **Center** (covariance matrix) or **standardize** (correlation matrix) the data. [→ Always at least center (subtract mean), but don't standardize if the scale carries significant meaning, e.g. several features measuring similar things (signal intensity at different locations)]
2. Compute **covariance/correlation matrix** ($\boldsymbol{\Sigma} = \frac{1}{n-1} \boldsymbol{X}^T \boldsymbol{X}$)
3. Compute the **eigendecomposition** of $\boldsymbol{\Sigma}$ [→ 2./3. may be replaced by one step by performing a singular value decomposition directly on the centered data]
4. Rank **eigenvectors** according to the value of the corresponding **eigenvalues** of $\boldsymbol{\Sigma}$
5. Keep $k \leq m$ **features**
6. **Transform data set** with m features to the new k features (**principal components**)

Principal Component Analysis (PCA)

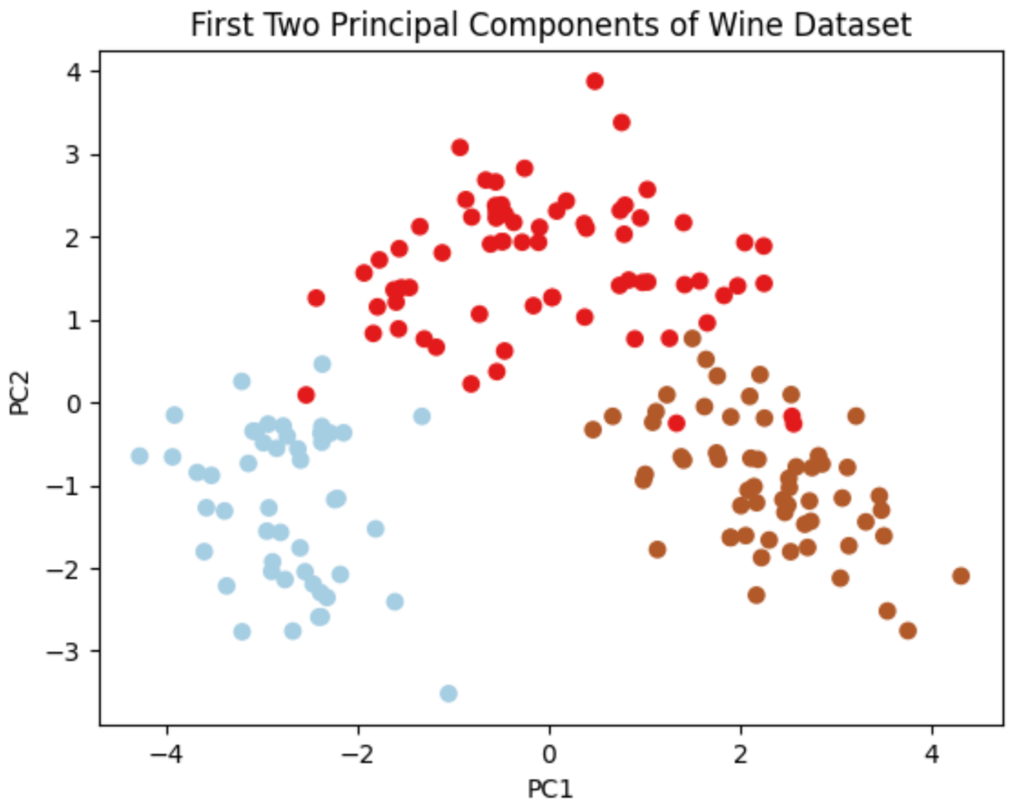
`05_PCA_logreg_wine.ipynb`

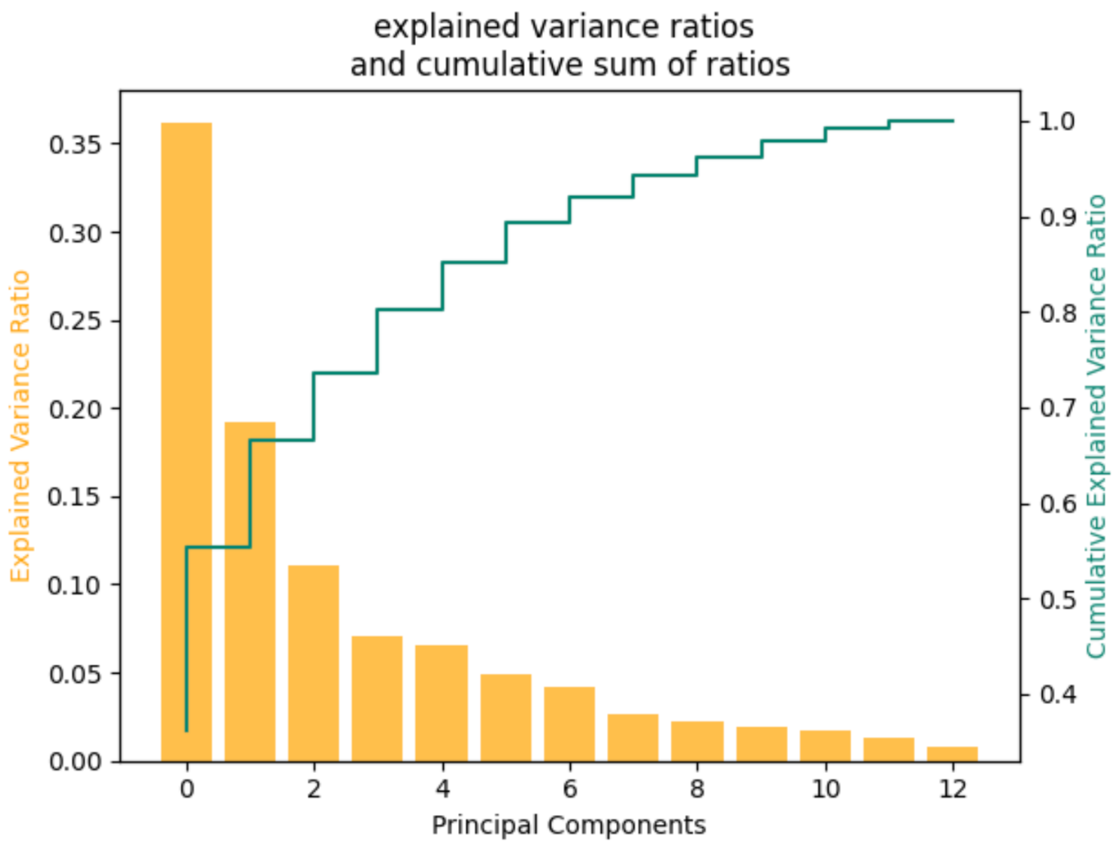


Principal Component Analysis (PCA)

`05_PCA_wine_explained_variance.ipynb`

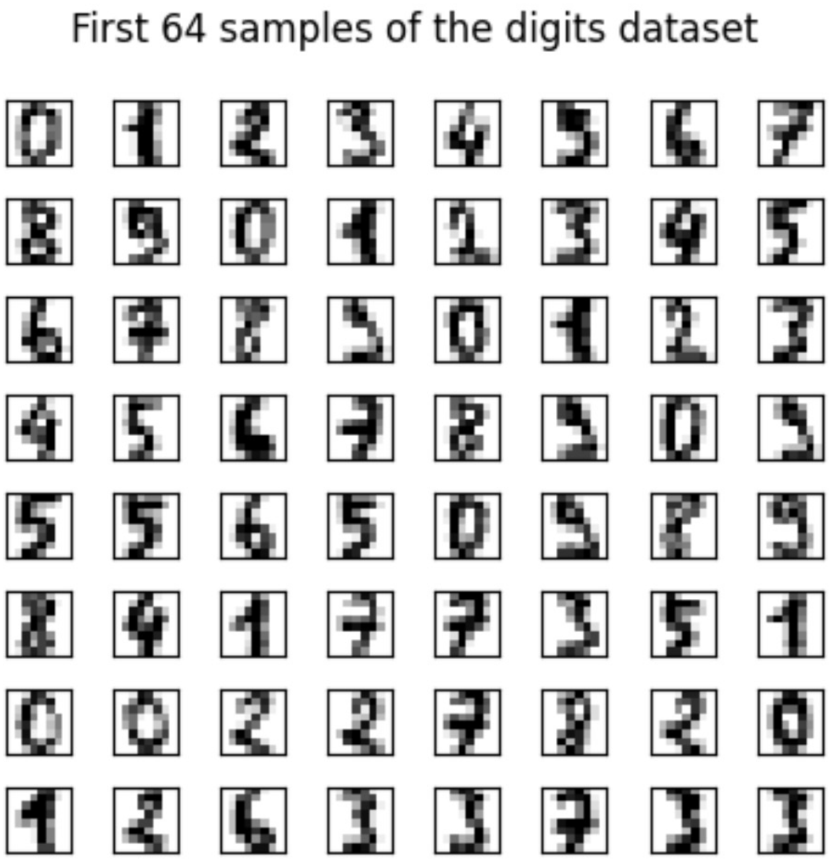
Explained variance ratio = $\frac{\lambda_j}{\sum_{j=1}^{m} \lambda_j}$



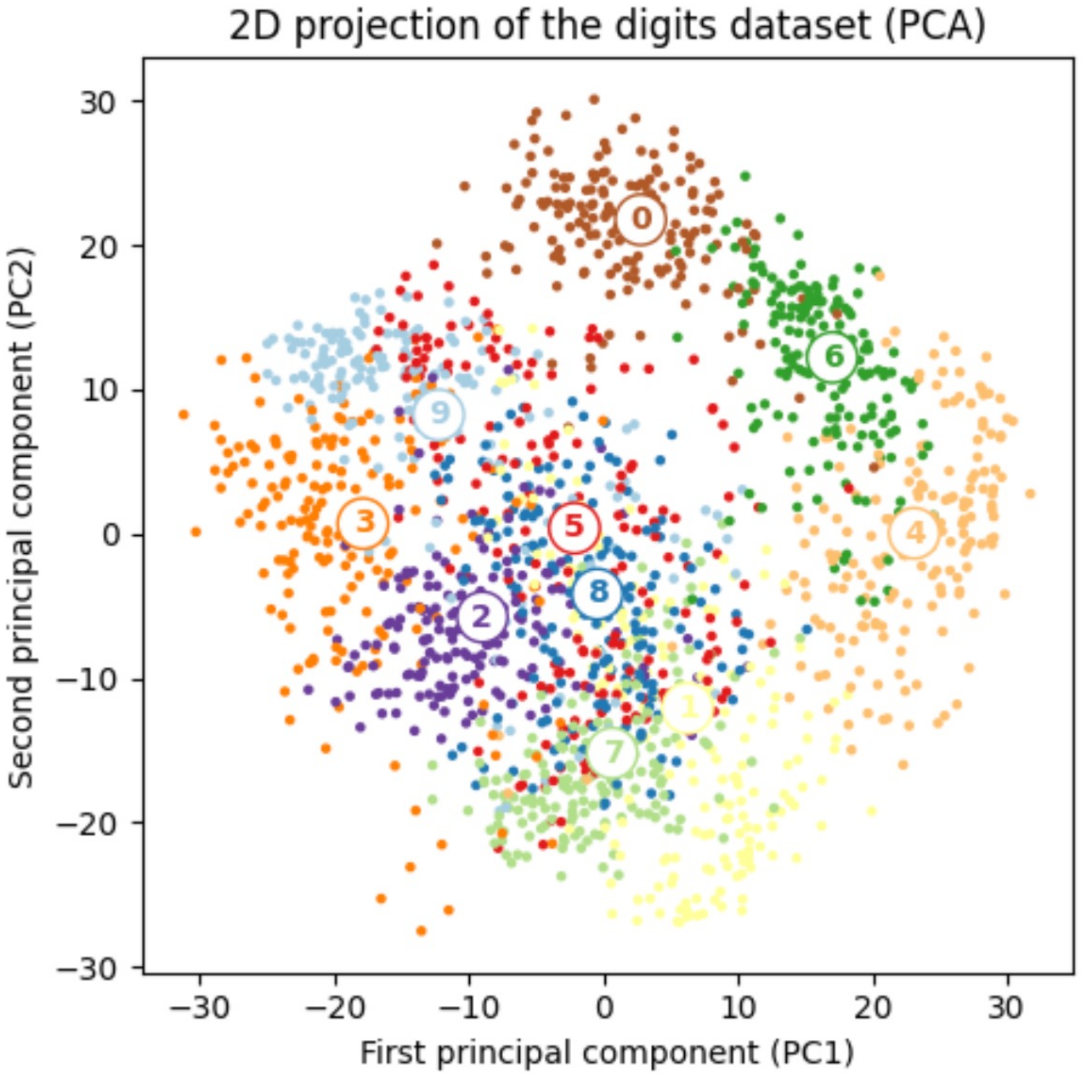


Principal Component Analysis (PCA)

`05_PCA_handwritten_digits.ipynb`



PCA
(k=2)



Why use PCA? Advantages of PCA

- **Reduces** the **number of features** which can lead to
  - a significant speed-up of the model training and prediction and
  - a reduction in required memory to store the data (data compression)
- May **reduce noise** in the data (e.g. common for signal data) which may improve performance of an ML model that is sensitive to noise
- Makes it possible to **visualize high-dimensional data sets** using only the features with highest variance

Why not always use PCA? Disadvantages of PCA

- **Principal components** are difficult to interpret (no obvious connection to the original features)
- **Principal components** with a **low explained variance** may, in fact, be important for class separation. PCA detect features with high variance. It doesn't use class labels (is unsupervised). In case highly discriminative features (separating classes) have low variance, PCA discards important information.
- **PCA** is a linear model, but relations between features may be more complex than linear combinations of the original features
- The **PCA** result highly **depends** on the **scaling of variables**. For example, going from *km* to *cm* for a length feature will increase importance of the feature by a lot

Principle component analysis in other fields

All based on the same principle and mathematics!

- **Eigenvalue decomposition** (for symmetric matrices) of $\boldsymbol{X}^T\boldsymbol{X}$ ($\boldsymbol{X}$ centered)
- **Singular value decomposition** of $\boldsymbol{X}$ (singular values are square root of eigenvalues)
- **Proper Orthogonal Decomposition** (POD): surrogate models and reduced order models (ROM) in engineering and robotics
- **Karhunen–Loève transformation**: signal processing
- **Spectral decomposition / modal analysis**: audio and vibration signal processing, structural dynamic (e.g. eigenmodes of vibrating drum or string)
- **Principal axis transformation:** mechanical engineering, structural mechanics

Kernel PCA

Nonlinear feature extraction PCA + Kernel trick

Kernel Principal Component Analysis

- **Kernelized, nonlinear variant** of **PCA**
- **Non-zero eigenvalues** of $\boldsymbol{X}^T\boldsymbol{X}$ and $\mathbf{X}\mathbf{X}^\mathbf{T}$ are the same

$$(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{v} = \lambda\boldsymbol{v} \quad \Longrightarrow \quad \boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{v} = \lambda\boldsymbol{X}\boldsymbol{v} \quad \Longrightarrow \quad (\boldsymbol{X}\boldsymbol{X}^T)\boldsymbol{u} = \lambda\boldsymbol{u}; \quad \boldsymbol{u} \coloneqq \boldsymbol{X}\boldsymbol{v}$$

- $\boldsymbol{X}\boldsymbol{X}^T$ is the **kernel matrix** when using the **linear kernel function** $\boldsymbol{\kappa}\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \boldsymbol{x}^{(i)^T}\boldsymbol{x}^{(j)}$. It contains the inner products of all pairs of samples in the data set.
- **Basic idea:** Replace $\boldsymbol{X}\boldsymbol{X}^T$ with the kernel matrix of a nonlinear kernel function to perform PCA in some high-dimensional (possibly infinite dimensional) feature space
- The hope is that in this transformed feature space (nonlinear transformation) the data may be decomposable into linear combinations of the transformed features

Kernel Principal Component Analysis

- **Kernelized, nonlinear variant** of **PCA**
- **Non-zero eigenvalues** of $\boldsymbol{X}^T\boldsymbol{X}$ and $\mathbf{X}\mathbf{X}^{\mathbf{T}}$ are the same

$$(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{v} = \lambda\boldsymbol{v} \quad \Longrightarrow \quad \boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{v} = \lambda\boldsymbol{X}\boldsymbol{v} \quad \Longrightarrow \quad (\boldsymbol{X}\boldsymbol{X}^T)\boldsymbol{u} = \lambda\boldsymbol{u}; \quad \boldsymbol{u} \coloneqq \boldsymbol{X}\boldsymbol{v}$$

- $\boldsymbol{X}\boldsymbol{X}^T$ is the **kernel matrix** when using the **linear kernel function** $\boldsymbol{\kappa}\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \boldsymbol{x}^{(i)^T}\boldsymbol{x}^{(j)}$. It contains the inner products of all pairs of samples in the data set.

$$K_{ij} = \kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \begin{bmatrix} \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}) & \cdots & \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(n)}) \\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}^{(n)}, \mathbf{x}^{(1)}) & \cdots & \kappa(\mathbf{x}^{(n)}, \mathbf{x}^{(n)}) \end{bmatrix}$$

$$\kappa(\mathbf{x}, \mathbf{x}') : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

(Kernel matrix)

(Kernel function)

Kernel Principal Component Analysis

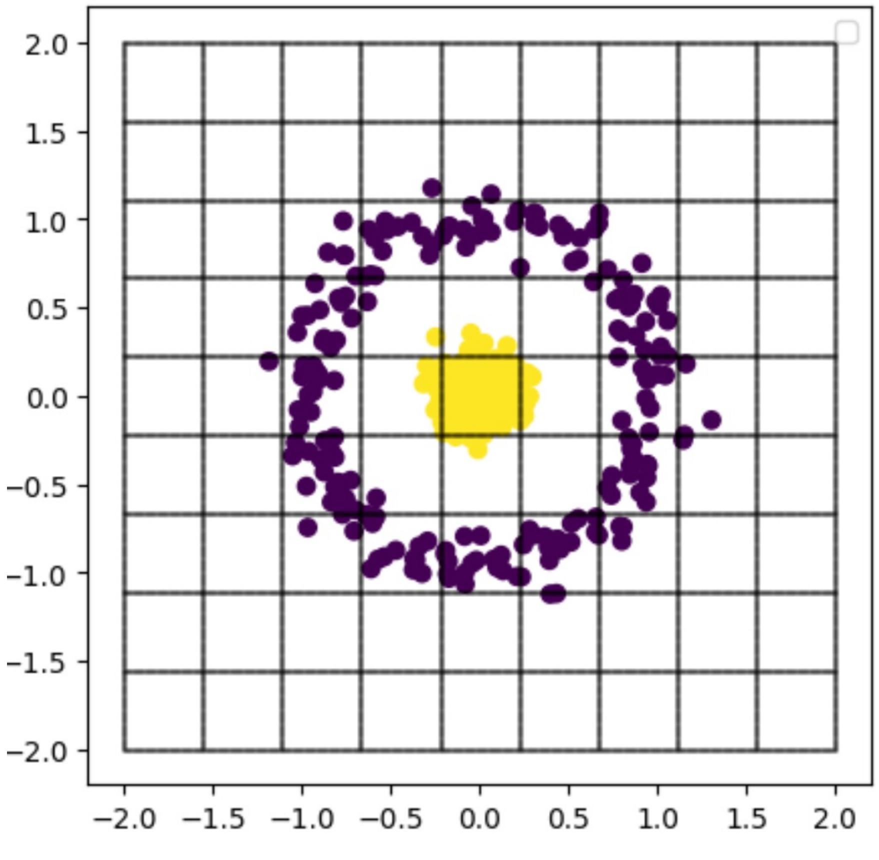
- **Kernelized, nonlinear variant** of **PCA**
- **Non-zero eigenvalues** of $\boldsymbol{X}^T\boldsymbol{X}$ and $\mathbf{X}\mathbf{X}^\mathbf{T}$ are the same

$$(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{v} = \lambda\boldsymbol{v} \quad \Longrightarrow \quad \boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{v} = \lambda\boldsymbol{X}\boldsymbol{v} \quad \Longrightarrow \quad (\boldsymbol{X}\boldsymbol{X}^T)\boldsymbol{u} = \lambda\boldsymbol{u}; \quad \boldsymbol{u} \coloneqq \boldsymbol{X}\boldsymbol{v}$$

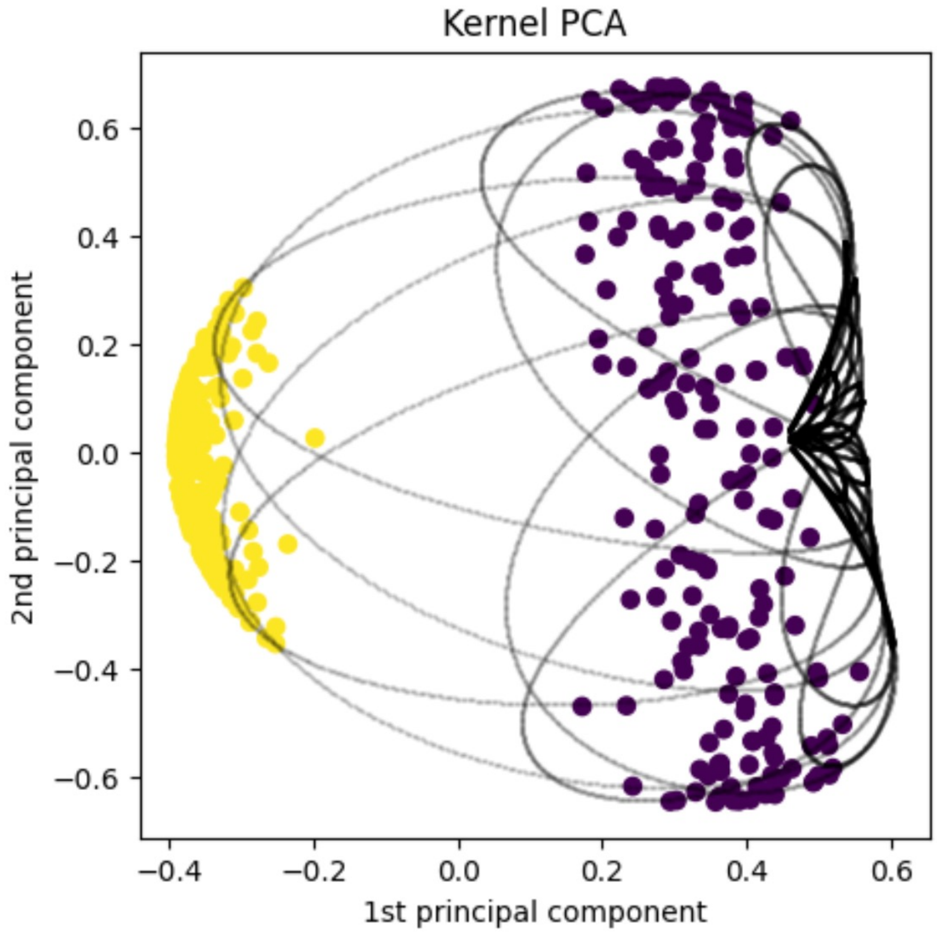
- $\boldsymbol{X}\boldsymbol{X}^T$ is the **kernel matrix** when using the **linear kernel function** $\boldsymbol{\kappa}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \boldsymbol{x}^{(i)^T}\boldsymbol{x}^{(j)}$. It contains the inner products of all pairs of samples in the data set.
- **Basic idea of Kernel PCA:** Replace $\boldsymbol{X}\boldsymbol{X}^T$ with the kernel matrix of a nonlinear kernel function to perform PCA in some high-dimensional (possibly infinite dimensional) feature space
- The hope is that in this transformed feature space (nonlinear transformation) the data may be decomposable into linear combinations of the transformed features

Kernel Principal Component Analysis

05_KernelPCA.ipynb



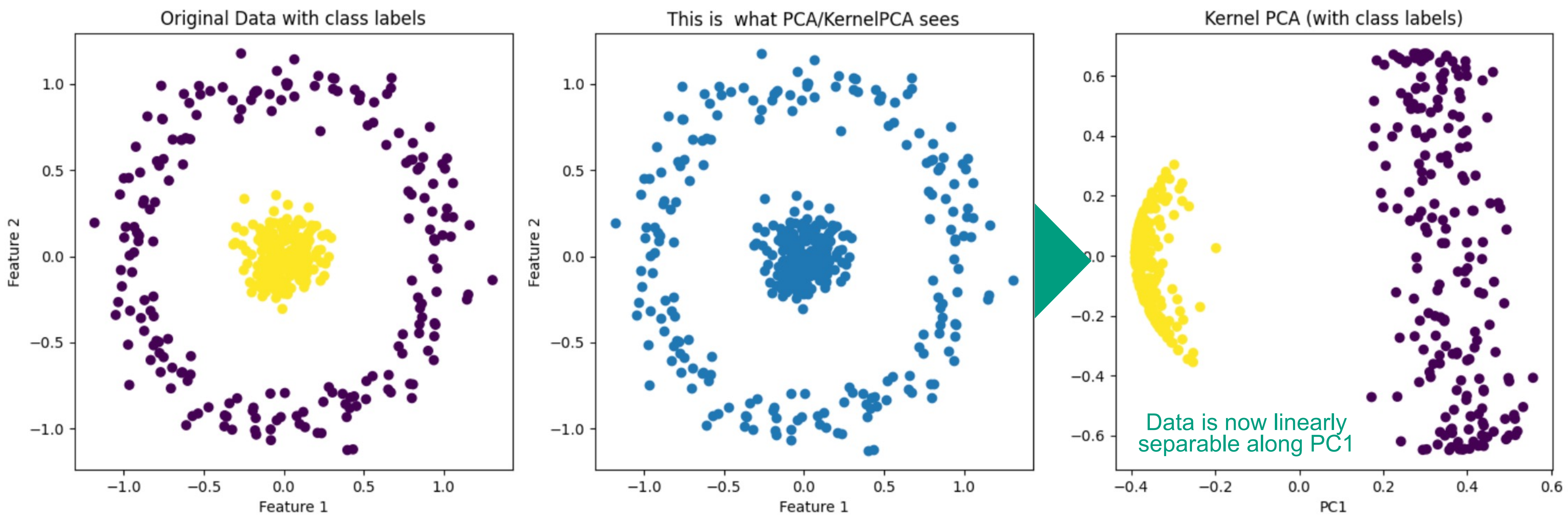
Kernel
PCA



Kernel Principal Component Analysis

`05_KernelPCA.ipynb`

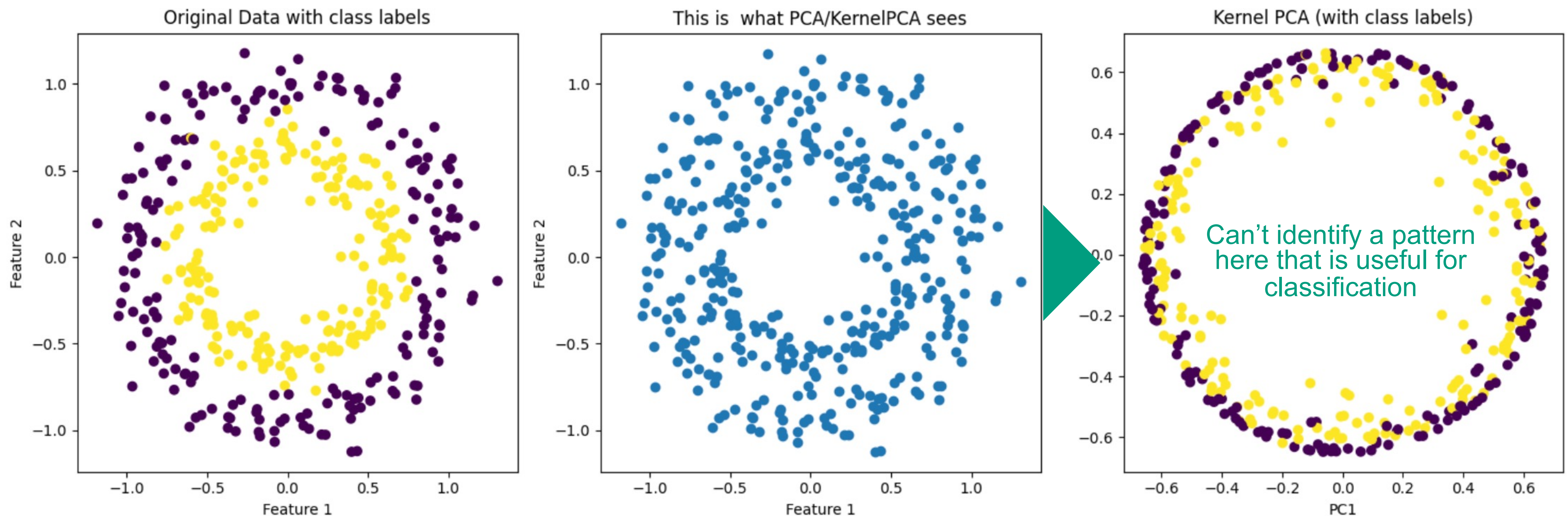
Be aware that PCA is an unsupervised learning technique! → Doesn't use labels



Kernel Principal Component Analysis

`05_KernelPCA.ipynb`

Be aware that PCA is an unsupervised learning technique! → Doesn't use labels



Kernel Principal Component Analysis

- Difficult to interpret
- Very sensitive to kernel parameters
- Can identify possible nonlinear feature interactions / nonlinear patterns in the data
- Could be combined in a pipeline with a performant linear classifier

- Whether no PCA, linear, or kernel PCA is the best choice depends on the data set and the relationships between features