THE UNIVERSITY *of* EDINBURGH

Week 2: Dimension Reduction

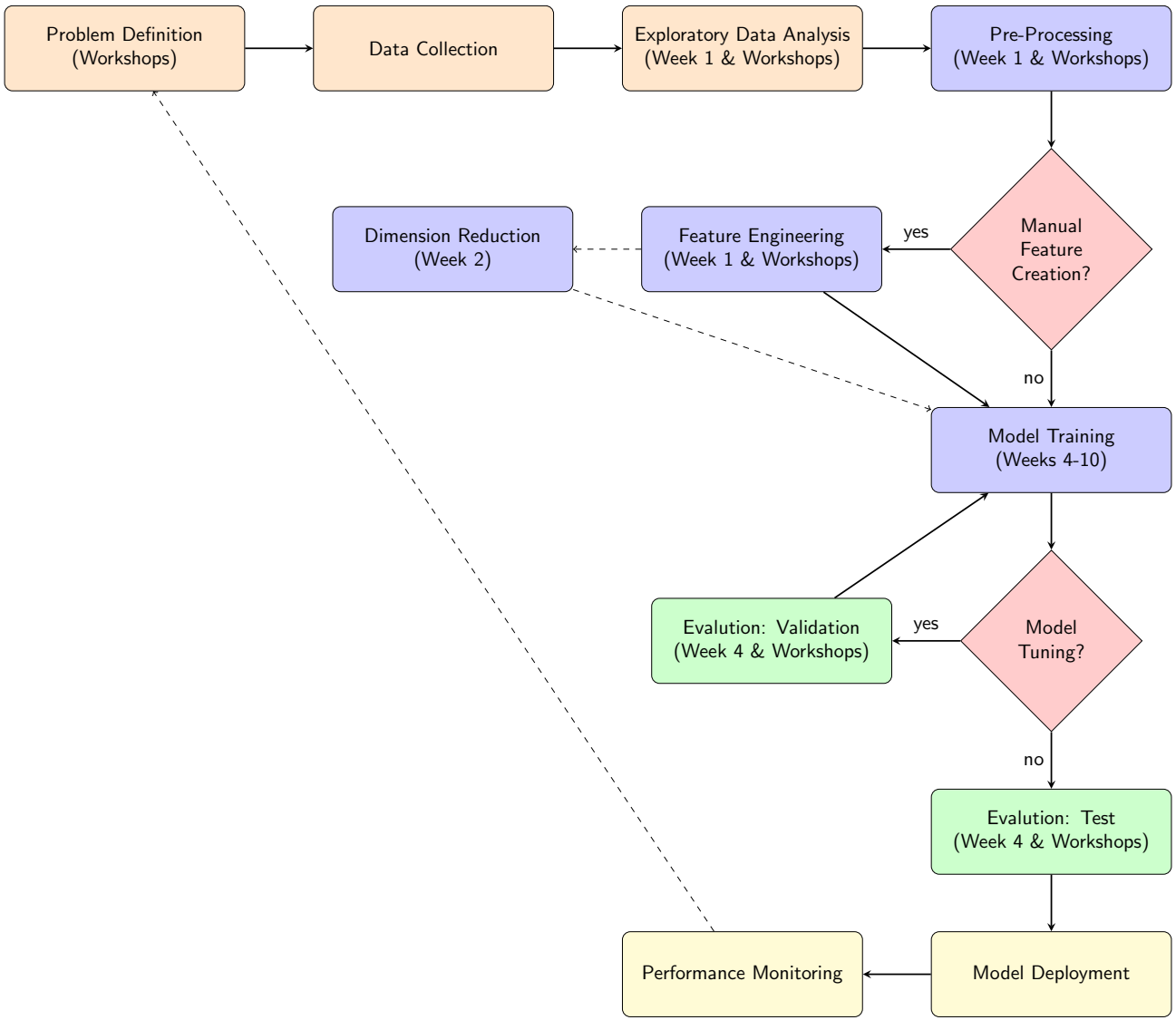
Dr Sara Wade
sara.wade@ed.ac.uk

19 Jan, 2026

Outline

Outline

ML Pipeline - Recap



Unsupervised Learning - Recap

In unsupervised learning, we only observe features $\boldsymbol{x}_n \in \mathbb{R}^D$ for $n = 1, \ldots, N$. The **feature matrix**[1] $\mathbf{X}$ is an $N \times D$ real-valued matrix:

$$\mathbf{X} := \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,D} \\ x_{2,1} & x_{2,2} & \ldots & x_{2,D} \\ & & \vdots & \\ x_{N,1} & x_{N,2} & \ldots & x_{N,D} \end{bmatrix}.$$

We aim to use $\mathbf{X}$ to learn how to represent or find interesting patterns in the data.

[1] also called the **design** or **input** matrix

Dimension Reduction

In many problems, $D \gg 1 \Rightarrow$ Is there any informative way to visualize and/or summarize such data?

Examining pair plots even for moderate D is not practical, e.g. $D = 10$ there are 45 pairs of features! And they can be uninformative, containing only a small fraction of the total information present in the data.

In practice, many of the features in high-dimensional problems are highly correlated and there are many recurring *patterns*.

Extended Yale Face Database

The images all share common features, orientations, etc. There may be a better low-dimensional representation than the high-dimensional raw pixel intensities.

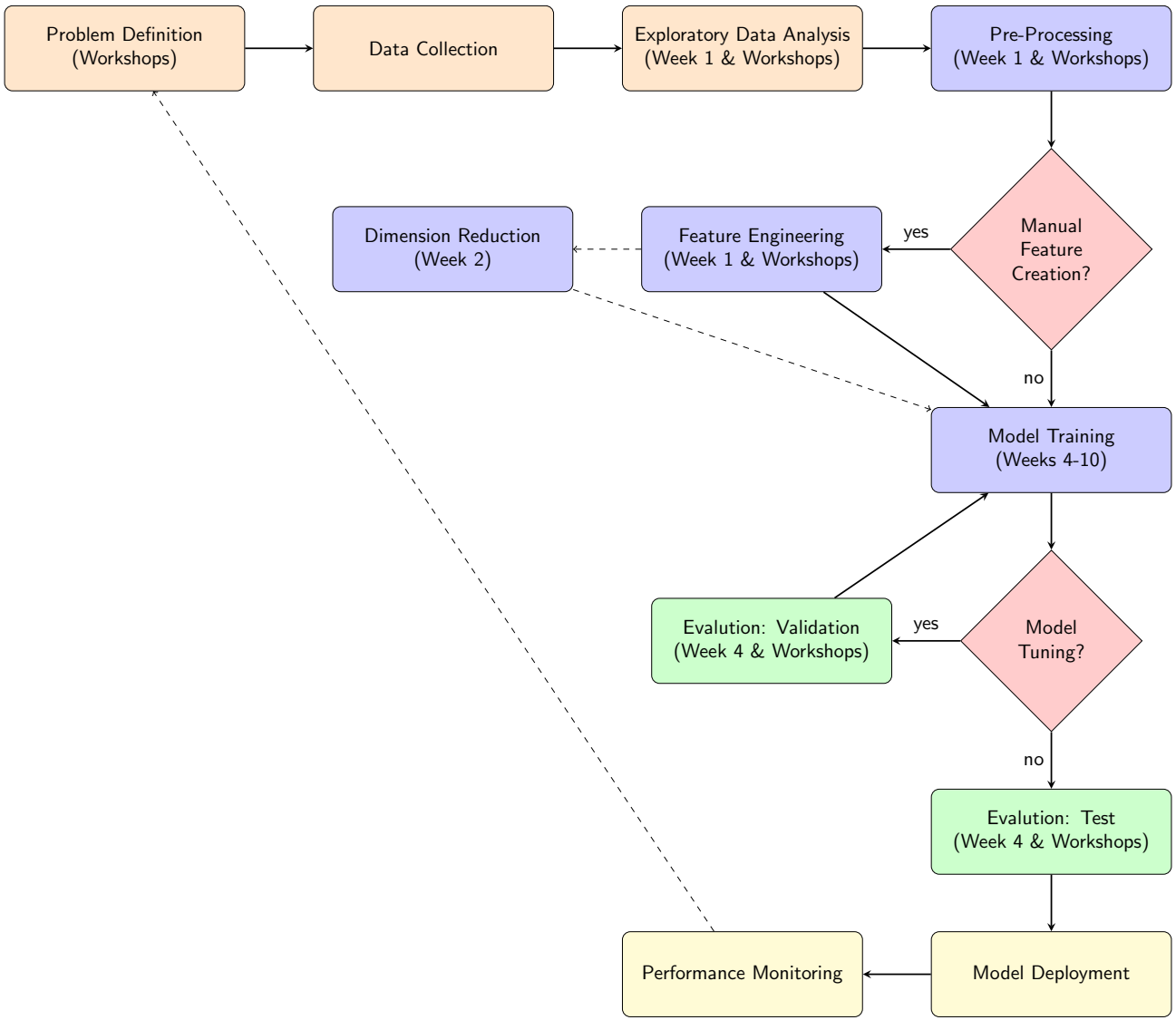
Dimension Reduction

Dimension reduction aims to learn a mapping from the high-dimensional space of observed features, $\boldsymbol{x} \in \mathbb{R}^D$, to a low-dimensional latent space, $\mathbf{z} \in \mathbb{R}^L$, that captures most of the information in the data.

This is useful for:

- Visualizing and summarizing the data in EDA,
- Creating a low-dimensional representation of the inputs in feature engineering,
- Presenting and visualizing the solution.

ML Pipeline - Where is Dimension Reduction useful?



Outline

Principal Component Analysis

Principal Component Analysis (PCA) is the simplest and most widely-used form of dimensionality reduction.

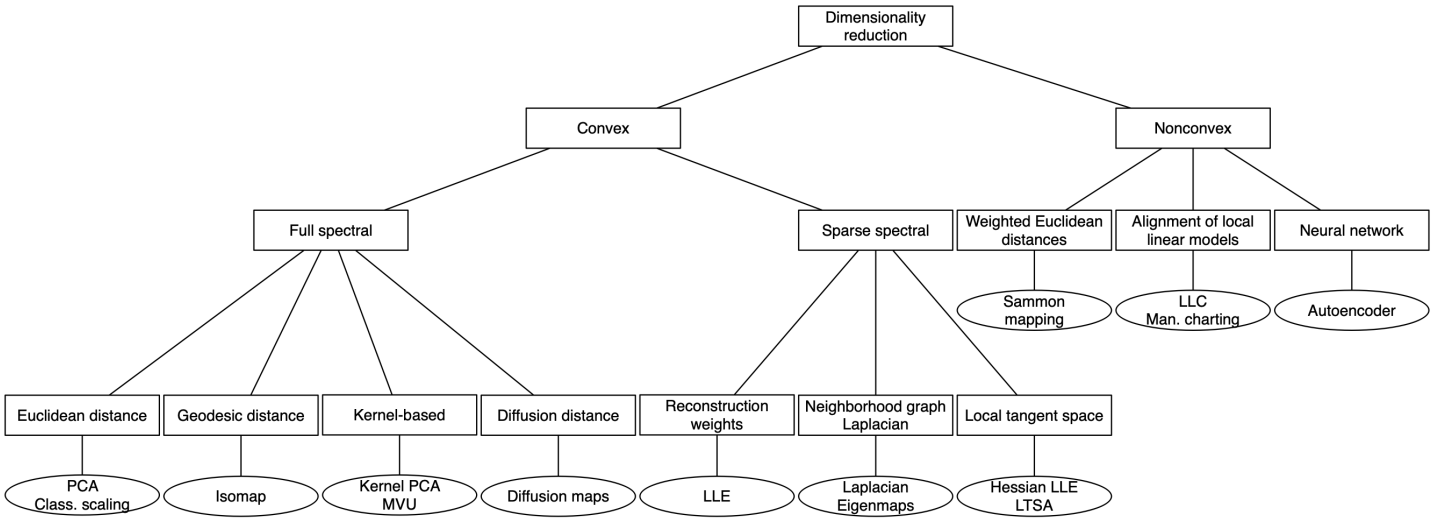


Figure 1: Taxonomy of dimension reduction techniques from van der Maaten & Postma (2009)

Principal Component Analysis

Aim: find a linear and orthogonal projection of the high-dimensional $\boldsymbol{x}$ into a low-dimensional $\mathbf{z}$, that provides a *good approximation* by explaining most of the variability.

- Let $\mathbf{W}$ is a $D \times L$ orthonormal matrix, with $L < D$.
- Encoder: linearly project $\mathbf{z} = \mathbf{W}^T \boldsymbol{x}$.
- Decoder: unproject $\widehat{\boldsymbol{x}} = \mathbf{W}\mathbf{z}$.
- PCA finds the solution that minimizes the reconstruction error:

$$\mathcal{L}(\mathbf{W}) = \frac{1}{N} \sum_{n=1}^{N} \|\boldsymbol{x}_n - \widehat{\boldsymbol{x}}_n\|_2^2.$$

Decoder

The decoder assumes each $\boldsymbol{x}_n$ is *explained* by a weighted combination of basis vectors, $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_L$, with $\boldsymbol{w}_l \in \mathbb{R}^D$ the lth column of $\mathbf{W}$ and weights $\mathbf{z}_n \in \mathbb{R}^L$:

$$\boldsymbol{x}_n \approx \widehat{\boldsymbol{x}}_n = \sum_{l=1}^{L} z_{n,l} \boldsymbol{w}_l.$$

Let $\mathbf{Z}$ be the $N \times L$ matrix, with:

- rows $\mathbf{z}_n^T$ corresponding to the low-dimensional representation of the nth data point,
- columns referred to as the **scores** of the **principal components** (or latent factors).

The columns $\boldsymbol{w}_l$ of $\mathbf{W}$ are referred to as the **loadings** of the lth principal component.

PCA Solution

PCA minimizes the (average) reconstruction error:

$$\mathcal{L}(\mathbf{W}, \mathbf{Z}) = \frac{1}{N} \sum_{n=1}^{N} \|\boldsymbol{x}_n - \mathbf{W}\mathbf{z}_n\|_2^2,$$

subject to the constraint that $\mathbf{W}$ is an orthonormal matrix.

Solution: $\mathbf{W}$ is the $D \times L$ matrix containing the first L eigenvectors with the largest eigenvalues of the empirical covariance matrix:

$$\boldsymbol{\Sigma} = \frac{1}{N} \sum_{n=1}^{N} (\boldsymbol{x}_n - \overline{\boldsymbol{x}})(\boldsymbol{x}_n - \overline{\boldsymbol{x}})^T,$$

where $\overline{\boldsymbol{x}}$ denotes the empirical mean.

Step 1 of Proof: What is the optimal encoding?

WLOG, assume $\mathbf{X}$ is **centred** to have zero mean.

Taking the derivative with respect to $\mathbf{z}_n$ and equating to zero gives:

$$\frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{Z})}{\partial \mathbf{z}_n} = \frac{1}{N}\left(-2\mathbf{W}^T \boldsymbol{x}_n + 2\mathbf{z}_n\right) = 0,$$
$$\Rightarrow \mathbf{z}_n = \mathbf{W}^T \boldsymbol{x}_n.$$

Note: Thus, each element of $\mathbf{z}_n$ is a normalized linear combination of the observed data:

$$z_{n,l} = w_{1,l}x_{n,1} + w_{2,l}x_{n,2} + \ldots + w_{D,l}x_{n,D}.$$

Step 2 of Proof: What is the optimal $\mathbf{W}$?

Plugging in $\mathbf{Z} = \mathbf{W}^T\mathbf{X}$, the loss is:

$$\begin{aligned}\mathcal{L}(\mathbf{W}) &= \frac{1}{N}\sum_{n=1}^{N}\|\boldsymbol{x}_n - \mathbf{W}\mathbf{W}^T\boldsymbol{x}_n\|_2^2. \\ &= \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n^T\boldsymbol{x}_n - \boldsymbol{x}_n^T\mathbf{W}\mathbf{W}^T\boldsymbol{x}_n.\end{aligned}$$

Proof proceeds by finding $\boldsymbol{w}_1$, then $\boldsymbol{w}_2$ given $\boldsymbol{w}_1$, and so on...

Optimal $\boldsymbol{w}_1$

$$\begin{aligned}\mathcal{L}(\boldsymbol{w}_1) &= \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n^T \boldsymbol{x}_n - \boldsymbol{w}_1^T \boldsymbol{x}_n \boldsymbol{x}_n^T \boldsymbol{w}_1 \\ &= \text{const} - \boldsymbol{w}_1^T \boldsymbol{\Sigma} \boldsymbol{w}_1 .\end{aligned}$$

Using Langrange multipliers and taking the derivative and equating to zero, we have:

$$\begin{aligned}&\frac{\partial \tilde{\mathcal{L}}(\boldsymbol{w}_1)}{\partial \boldsymbol{w}_1} = -2\boldsymbol{\Sigma}\boldsymbol{w}_1 + 2\lambda_1 \boldsymbol{w}_1 = 0, \\ &\Rightarrow \boldsymbol{\Sigma}\boldsymbol{w}_1 = \lambda_1 \boldsymbol{w}_1 .\end{aligned}$$

Thus, $\boldsymbol{w}_1$ is an eigenvector of $\boldsymbol{\Sigma}$. Moreover, left multiplying by $\boldsymbol{w}_1^T$ shows that is the eigenvector corresponding to the largest eigenvalue.

Alternative Interpretation

Minimizing the reconstruction error $\Leftrightarrow$ maximizing the empirical variance of the projected data:

$$\widehat{\mathrm{V}}(Z_1) = \frac{1}{N}\sum_{n=1}^{N} z_{n,1}^2 - \left(\frac{1}{N}\sum_{n=1}^{N} z_{n,1}\right)^2 .$$

Thus, we can also interpret PCA as finding the directions of maximal variance.

Optimal $\boldsymbol{w}_2$

$$\begin{aligned}\mathcal{L}(\boldsymbol{w}_2) &= \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n^T \boldsymbol{x}_n - \boldsymbol{w}_1^T \boldsymbol{x}_n \boldsymbol{x}_n^T \boldsymbol{w}_1 - \boldsymbol{w}_2^T \boldsymbol{x}_n \boldsymbol{x}_n^T \boldsymbol{w}_2 \\ &= \text{const} - \boldsymbol{w}_2^T \boldsymbol{\Sigma} \boldsymbol{w}_2.\end{aligned}$$

Using Langrange multipliers to impose the constraints, we want to optimize:

$$\tilde{\mathcal{L}}(\boldsymbol{w}_2) = -\boldsymbol{w}_2^T \boldsymbol{\Sigma} \boldsymbol{w}_2 + \lambda_2(\boldsymbol{w}_2^T \boldsymbol{w}_2 - 1) + \lambda_{1,2}(\boldsymbol{w}_2^T \boldsymbol{w}_1 - 0).$$

The solution is given by the eigenvector with the second largest eigenvalue:

$$\Rightarrow \boldsymbol{\Sigma} \boldsymbol{w}_2 = \lambda_2 \boldsymbol{w}_2.$$

Geometric Intrepretation

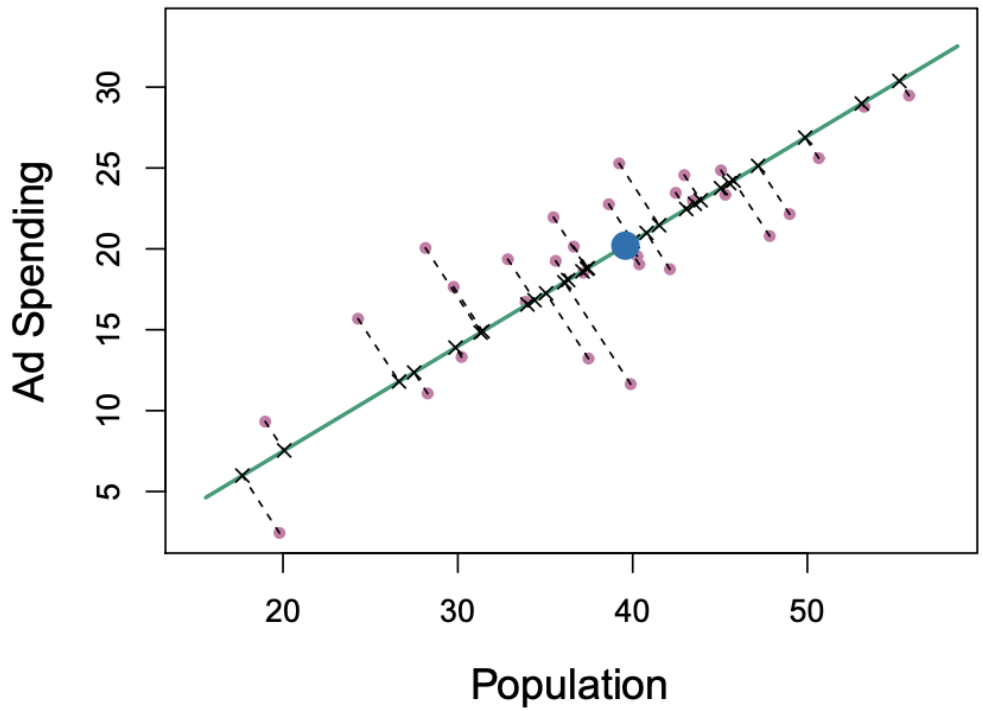
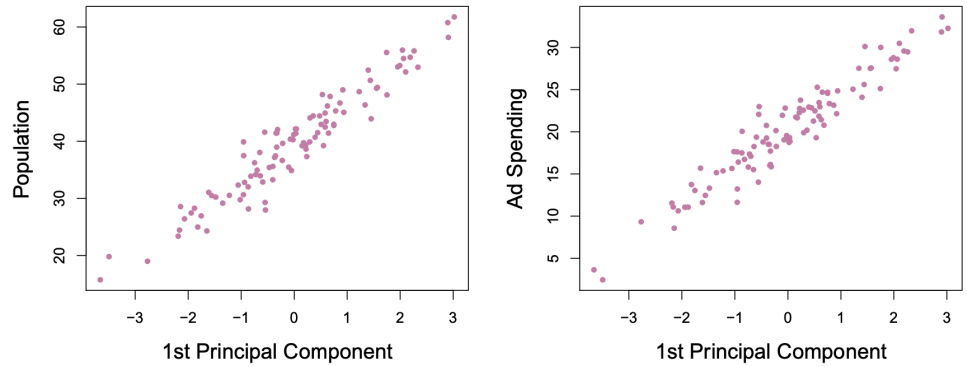
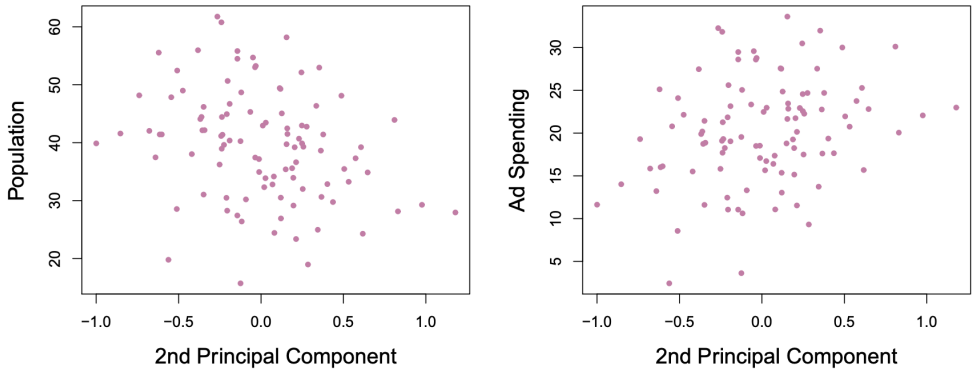


Figure 2: Population size vs ad spending for cities, with mean (blue circle). Green line: first principal component. Black crosses: reconstructions.

Geometric Interpretation



(a) First principal component



(b) Second principal component

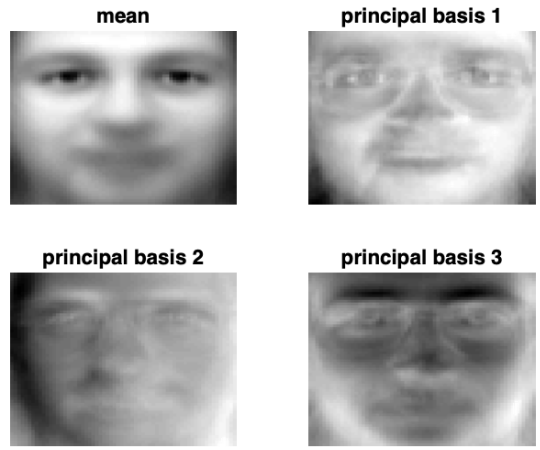
Figure 3: The loadings of the 1st PC: $w_{1,1} = 0.839$ and $w_{2,1} = 0.544$.

Interpreting the Basis Vectors

The vectors $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_L$ can be thought of as being the dominant patterns in the data, and they describe the main sources of variation.

In high-dimensions, in can be hard to interpret the latent dimensions. But, for images, we can plot each $\boldsymbol{w}_l$ as an image.

Olivetti Face Database

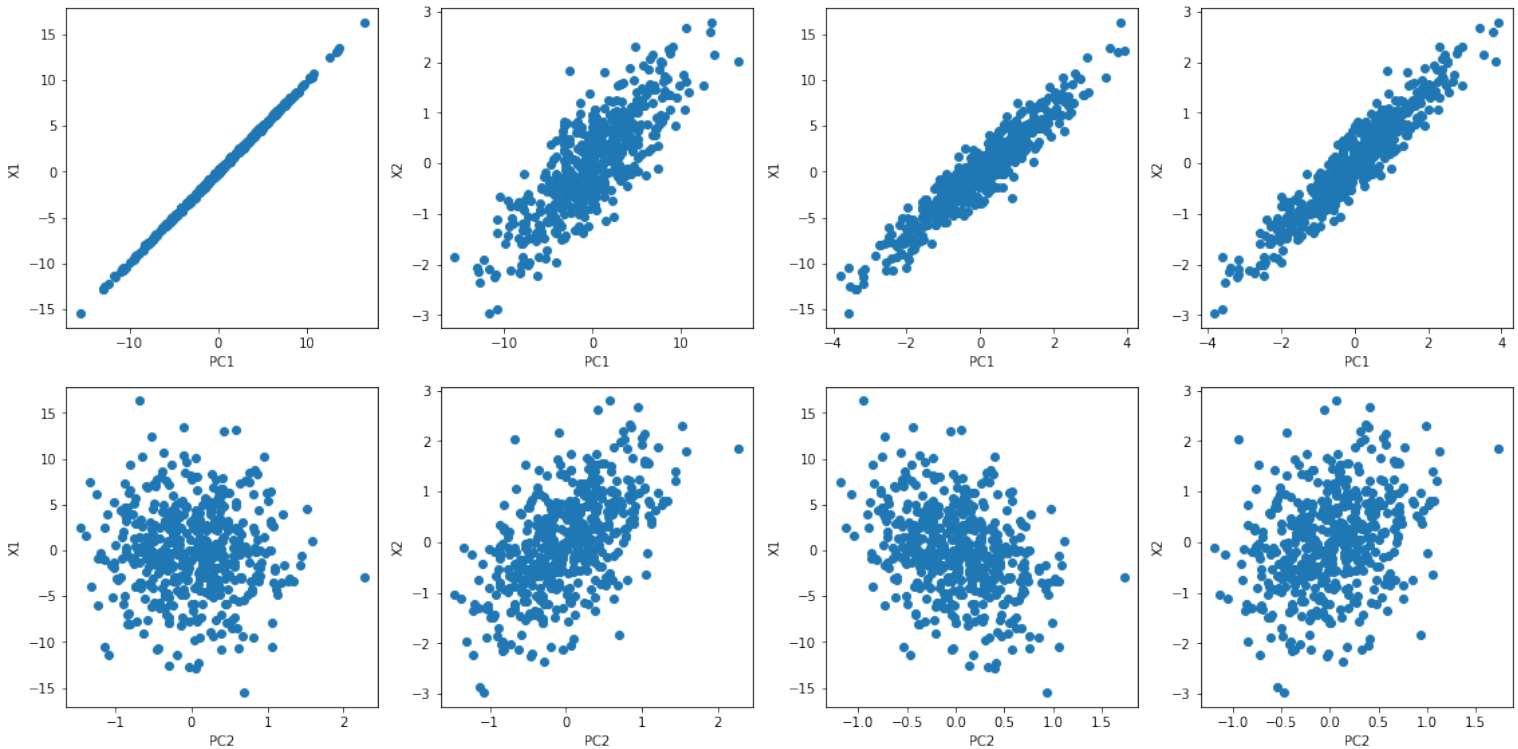


Covariance vs. Correlation Matrix

PCA results will be dominated by the features with the highest variance.

- typically, the data are first standardized, i.e. eigen decomposition of the correlation instead of the covariance matrix.
- but if the features are all **measured on the same units** (e.g. imaging data, gene expression data, etc.), it may preferable to use the covariance matrix.

Covariance vs. Correlation Matrix



(a) No Standarization (b) With Standarization

Figure 4: Data are generated from a bivariate normal with covariance matrix $[25, 4; 4, 1]$.

High-dimensional Data

We focused on the eigen decomposition of $\Sigma = 1/N\,\mathbf{X}^T\mathbf{X}$, but if $D > N$, it is faster to work with the $N \times N$ **Gram matrix**[2] $\mathbf{X}\mathbf{X}^T$.

Eigen decomposition of the Gram matrix: $\mathbf{X}\mathbf{X}^T\mathbf{U} = \mathbf{U}\mathbf{\Lambda}$.

Pre-multiplying by $\mathbf{X}^T$, shows that $\mathbf{X}^T\mathbf{U}$ are the eigenvectors of $\mathbf{X}^T\mathbf{X}$, with eigenvalues given in $\mathbf{\Lambda}$, and the normalized eigenvectors are

$$\mathbf{V} = \mathbf{X}^T\mathbf{U}\mathbf{\Lambda}^{-\frac{1}{2}}.$$

[2] this trick is also used in **kernel PCA**

Computing PCA using SVD

Singular value decomposition (SVD) of $\mathbf{X}$:

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T,$$

where $\mathbf{U}$ is an $N \times N$ orthonormal matrix, $\mathbf{V}$ is a $D \times D$ orthonormal matrix, and $\mathbf{D}$ is an $N \times D$ matrix with the structure (assuming $N > D$):

$$\mathbf{D} = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_D \\ & \cdots & [\mathbf{0}] & \cdots \end{bmatrix}.$$

The **reduced SVD** avoids computing the unnecessary elements by factorizing $\mathbf{X}$ as

$$\mathbf{X} = \mathbf{U}_1\mathbf{D}_1\mathbf{V}^T,$$

where $\mathbf{U}_1$ is the $N \times D$ matrix containing the first D columns of $\mathbf{U}$ and $\mathbf{D}_1$ is a diagonal $D \times D$ matrix containing the first D rows of $\mathbf{D}$.

Computing PCA using SVD

The SVD decomposition of $\mathbf{X}$ provides a decomposition of Σ:

$$\begin{aligned} N\Sigma = \mathbf{X}^T\mathbf{X} &= \mathbf{V}\mathbf{D}_1\mathbf{U}_1^T\mathbf{U}_1\mathbf{D}_1\mathbf{V}^T \\ &= \mathbf{V}\mathbf{D}_1\mathbf{D}_1\mathbf{V}^T. \end{aligned}$$

- the eigenvectors of Σ are equal to $\mathbf{V}$ (the right singular vectors of $\mathbf{X}$).
- the eigenvalues of Σ are the squared singular values divided by N.
- The encoding $\mathbf{Z}$ can be computed as:

$$\mathbf{Z} = \mathbf{X}\mathbf{W} = \mathbf{U}_1\mathbf{D}_1\mathbf{V}^T\mathbf{W} = \tilde{\mathbf{U}}\tilde{\mathbf{D}},$$

  where $\tilde{\mathbf{U}}$ contains the first L columns of $\mathbf{U}$ and $\tilde{\mathbf{D}}$ is a diagonal matrix containing only the first L singular values.

$\Rightarrow$ SVD is often preferred over an eigen decomposition for computational reasons.

How many principal components are needed?

Unfortunately, there is no single answer to this question.

In general, we would like the smallest number required to get a good understanding of the patterns in the data.

- Often, we decide by examining a **scree plot** and finding the *elbow* in the plot.
- If all components show interesting patterns, we may also examine subsequent components.
- If want want accurate reconstructions, we may want more components (e.g. to achieve a specific proportion of variance explained).
- An alternative is *approximate Bayesian model selection* based on *probabilistic PCA* (available in sklearn's `PCA` transformer with `n_components='mle'`).

Proportion of Variance Explained

Proportion of variance explained (PVE) by the lth principal component:

$$\text{PVE}_l = \frac{\sum_{n=1}^{N} \mathbf{z}_{n,l}^2}{\sum_{n=1}^{N} \boldsymbol{x}_n^T \boldsymbol{x}_n} = \frac{\lambda_l}{\sum_{j=1}^{D} \lambda_j}.$$

Scree plot: displays the eigenvalues λ_l of $\boldsymbol{\Sigma}$ against l in order of decreasing magnitude.

We select the value L at the point (elbow) where the PVE drops off.

Scree Plot

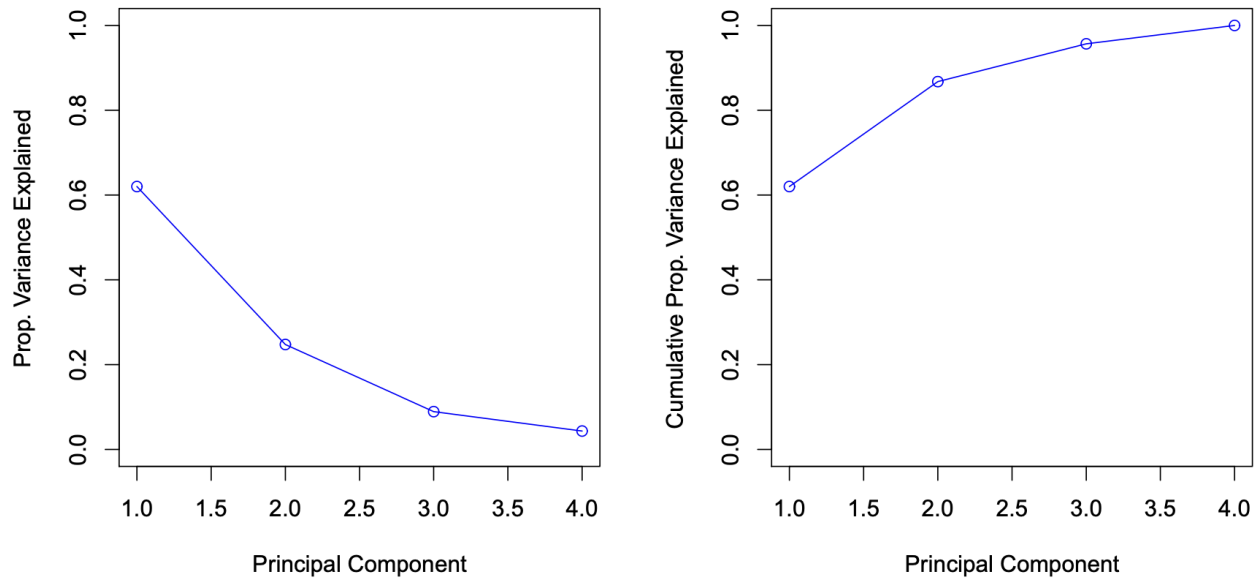


Figure 5: A good amount of variance is explained by the first two PCs and there is an elbow after the second PC.

Outline

Beyond PCA

Many other dimension reduction techniques are available in `sklearn`:

- Matrix factorization
- Manifold learning

In practice, often PCA reduction is first performed, followed by nonlinear reduction (Statistical exploration of the Manifold Hypothesis by Whiteley, Gray, and Rubin-Delanchy (2025)).

Kernel PCA, uses the **kernel trick** to extend PCA from linear to nonlinear reduction and is available in `sklearn.decomposition.KernelPCA`.

Kernels

A kernel $k(\boldsymbol{x}, \boldsymbol{x}')$ is a real-valued function of two inputs $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$, that can be defined over general input spaces $\mathcal{X}$, such as strings.

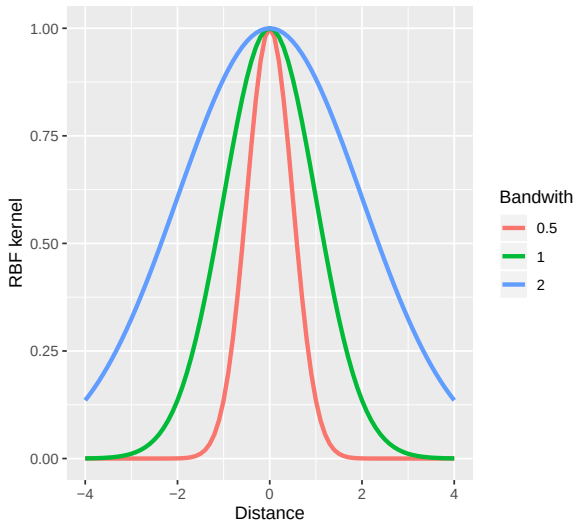
Kernels are typically assumed to be symmetric, i.e. $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$, and non-negative, i.e. $k(\boldsymbol{x}, \boldsymbol{x}') \geq 0$.

Kernels: Radial Basis Functions

The **radial basis function** (RBF) kernel is defined as:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2\ell^2}(\boldsymbol{x} - \boldsymbol{x}')^T(\boldsymbol{x} - \boldsymbol{x}')\right),$$

with length-scale or bandwidth parameter $\ell > 0$. It has a bell-curve shape, with the spread controlled by the bandwidth.

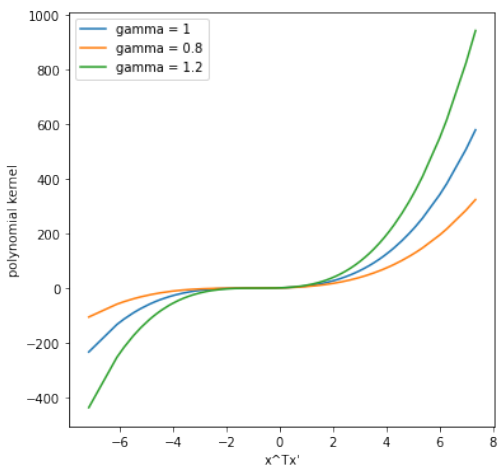


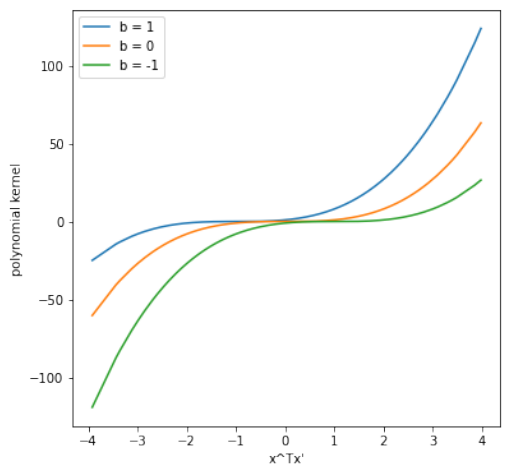
Kernels: Polynomial

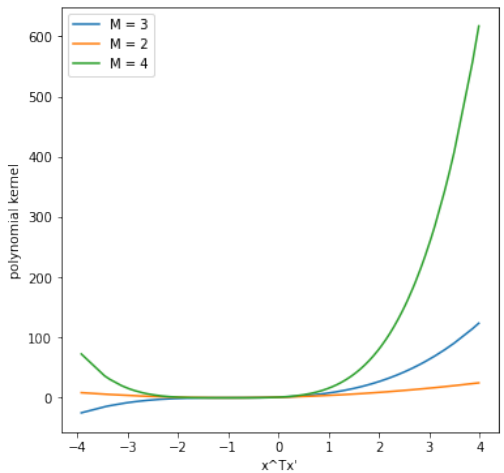
The **polynomial** kernel is:

$$k(\boldsymbol{x}, \boldsymbol{x}') = (-\gamma \boldsymbol{x}^T \boldsymbol{x}' + b)^M.$$

It has polynomial shape with parameters γ, b, and M determining the steepness, position, and degree.







Kernel Trick

Kernel trick:

1. Rewrite all computations to depend on the inputs through their inner products.
2. Replace the all inner products $\boldsymbol{x}^T \boldsymbol{x}'$ with the kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$.

Simple trick that can be used to increase flexibility in many machine learning algorithms, in regression, classification, dimension reduction, and clustering.

Kernel Trick

We require the kernel to be a **Mercer kernel**: the **Gram matrix**, defined by

$$\mathbf{K} = \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \vdots & & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix},$$

must be positive definite for any set of inputs $\{\boldsymbol{x}_n\}_{n=1}^N$, e.g. the polynomial and RBF are Mercer kernels.

Mercer's Theorem: for all Mercer kernels, $k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}')$.

$\rightarrow$ implicitly replace $\boldsymbol{x}$ with new features $\phi(\boldsymbol{x})$, yet through the kernel function, we avoid having to work in the new high-dimensional feature space (e.g. RBF corresponds to infinite-dimensional features).

Polynomial Kernel Feature Space

Consider $M = 2$, $\gamma = 1$, and $b = 1$ and $\boldsymbol{x} \in \mathbb{R}^2$:

$$k(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^T \boldsymbol{x}')^2 = (1 + x_1 x_1' + x_2 x_2')^2$$

This can be written as $k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}')$, where

$$\phi(\boldsymbol{x}) = \left(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2\right)^T$$

contains all terms up to degree $M = 2$.

Kernel PCA

1. Replace[3] $\mathbf{X}\mathbf{X}^T$ with $\mathbf{K}$.
2. Let $\mathbf{U}$ be the orthonormal matrix of the first L eigenvectors and $\mathbf{\Lambda}$ be the diagonal matrix of the first L eigenvalues of $\mathbf{K}$, with normalized eigenvectors:

$$\mathbf{V} = \mathbf{\Phi}^T \mathbf{U} \mathbf{\Lambda}^{-\frac{1}{2}},$$

with feature matrix $\mathbf{\Phi} = [\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_N)]^T$. Then, the encoding of $\boldsymbol{x}_n$ is:

$$\mathbf{z}_n = \mathbf{V}^T \phi(\boldsymbol{x}_n) = \mathbf{\Lambda}^{-\frac{1}{2}} \mathbf{U}^T \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_n) \end{bmatrix}.$$

[3] Note: we actually use $\tilde{\mathbf{K}} = \mathbf{C}\mathbf{K}\mathbf{C}$ for the centered features, with centering matrix $\mathbf{C} = \mathbf{I}_N - \frac{1}{N}\mathbf{1}_N\mathbf{1}_N^T$

Kernel PCA

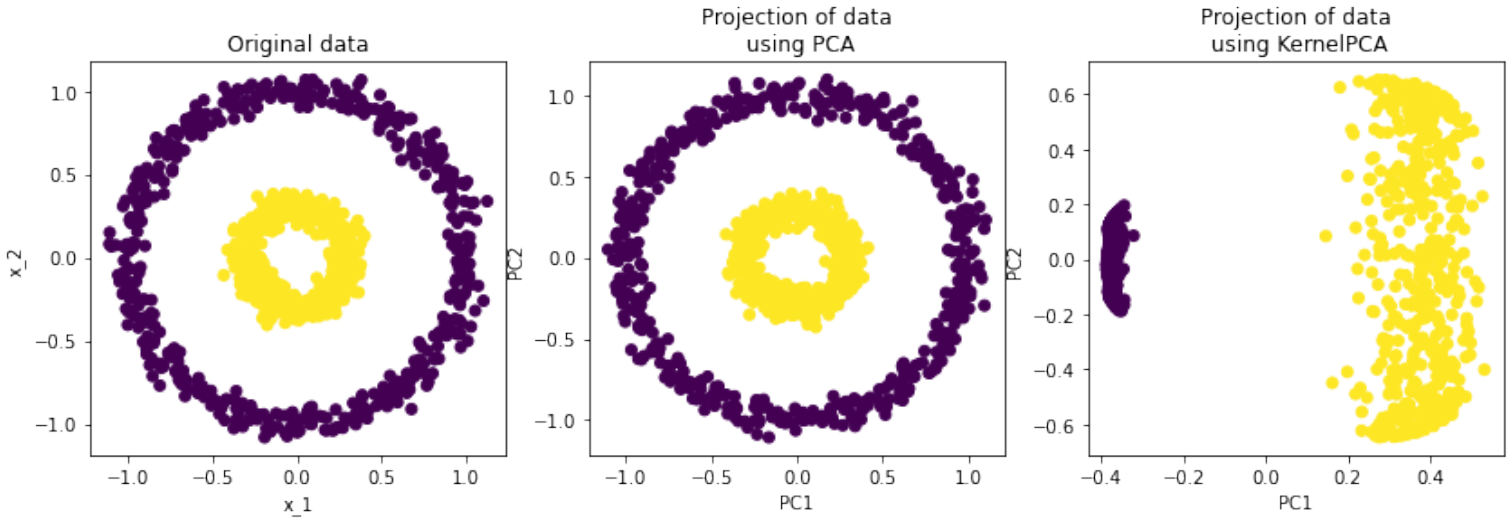


Figure 6: Kernel PCA with RBF separates the classes/circles along the first dimension.

Kernel PCA is available in `sklearn.decomposition.KernelPCA`.