Machine Learning for researchers (300001030) Unit 3. Dimensionality Reduction Methods

Javier Vales Alonso

Doctorate transversal activity

2020

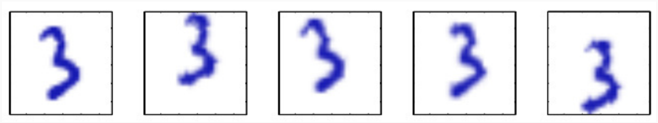
Technical University of Cartagena

How to study this unit?

1. Do a first reading of the unit's slides. Try to focus on the general ideas and do the first review over the maths involved.
2. Next, read and run the notebooks section by section. Leave the questions indicated there for later.
3. Do a more in-depth review of the maths with the slides and the notebook side by side. Try to understand all the developments involved. In case of doubts, read the suggested references (see Syllabus) or contact the teacher.
4. Finally, answer the exercises in the notebook and submit them back through AV.

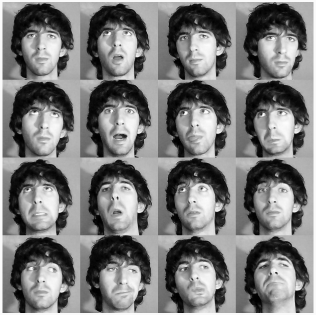
Introduction

Why dimensionality reduction?



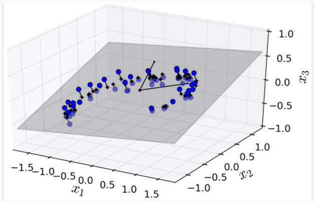
- In figure above each "data point" is a 100$\times$100 image, so they belong to a 10000-dim space.
- Using such a high dimension seems quite inefficient since all figures are very similar, just scaling, rotations, and translations of the same base figure.
- Numbers written by different people will have additional degrees of freedom, but, even though, much lower than that of a random image.

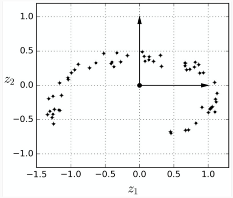
Why dimensionality reduction? (II)



- The same happens with the faces in the left figure (see link).
- The degree of freedom can be high, but it is still much smaller than the original data dimension.

Why dimensionality reduction? (III)





- Mathematically, the data points live within (or close to) a subspace (or manifold) inside the data space.
- For example, we can *approximate* the 3D points shown in the top figure by projecting them onto the plane (bottom figure). This way, we obtain the 2D representation (shown below).

Why dimensionality reduction? (IV)

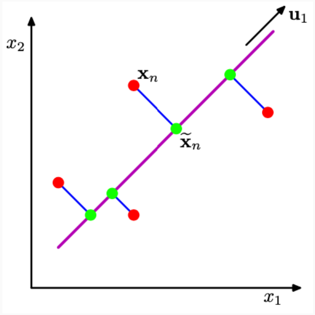
- High-dimensional spaces are counter-intuitive. In a M-dimensional unit hypercube pick a random point inside: as M grows it is likely that it is close to some hypercube edge (think about people, as you consider more and more characteristic -e.g., height, working times, salary, ideologies, etc.- it is likely that they have some extreme characteristic).
- As another example, the distance between points chosen at random inside the 2D unit cube is roughly 0.52. For the 3D unit cube, about 0.66. For a 1 million *unit* dimensional cube, it is 408.25 (much longer than the 1 unit edge length!!)
- In short, high dimensional spaces datasets are extremely *sparse*. Reducing dimensions is *critical* to increase predictions' reliability by avoiding large extrapolations about data.

Principal component analysis

Principal component analysis

- PCA (aka Karhunen-Loève transformation) is a natural technique used for dimensionality reduction.
- Different definitions give rise to the same algorithm.
- It was developed during the first half of the 20th century independently by Hotelling and Pearson.
- Its applications include lossy data compression, feature extraction, data visualization, and its probabilistic formulation allows the implementation of generative models.

Maximum variance formulation



- Given a set of observations $\{\boldsymbol{x}_n\}$ for $n = 1, \ldots, N$ of dimensionality D. Our goal is maximizing the variance of the projected data $\{\tilde{\boldsymbol{x}}_n\}$ onto a space with given dimensionality $M<D$.
- In the example, it means finding the best line to project the data in terms of variance.
- Maximizing the variance allows preserving as much information as possible in the reduced dimensional set

Maximum variance formulation (II)

Assume $M=1$ and let $\boldsymbol{u}_1$ denote a unit length vector. The mean of the projected data is $\boldsymbol{u}_1^T\overline{\boldsymbol{x}}$, being $\overline{\boldsymbol{x}}$ the sample mean of the data set:

$$\overline{\boldsymbol{x}} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n \tag{1}$$

and the sample variance:

$$\frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{u}_1^T\boldsymbol{x}_n - \boldsymbol{u}_1^T\boldsymbol{x}_n)^2 = \boldsymbol{u}_1^T\boldsymbol{S}\boldsymbol{u}_1 \tag{2}$$

with $\boldsymbol{S}$ being the data covariance matrix:

$$\boldsymbol{S} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n - \overline{\boldsymbol{x}})(\boldsymbol{x}_n - \overline{\boldsymbol{x}})^T \tag{3}$$

Maximum variance formulation (III)

Therefore, we have to maximize $\boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1$ with respect to $\boldsymbol{u}_1$ and subject to $\boldsymbol{u}_1^T \boldsymbol{u}_1 = 1$ (unit length vector). Introducing the constraint as a Lagrange multiplier leads to maximizing:

$$\boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1 + \lambda_1 (1 - \boldsymbol{u}_1^T \boldsymbol{u}_1). \tag{4}$$

By setting its derivative equal to zero (see last clause in link and note that covariance matrix is symmetric), we obtain:

$$\boldsymbol{S} \boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1. \tag{5}$$

That is, $\boldsymbol{u}_1$ is an eigenvector of $\boldsymbol{S}$ and variance is given by the eigenvalue $\lambda_1 = \boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1$. Maximizing it requires to select the largest eigenvalue. The associated eigenvector is called the *first principal component*.

Maximum variance formulation (IV)

For $M > 1$, the largest M eigenvalues of $\boldsymbol{S}$ have to be selected and the projection is given by the associated eigenvectors $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_M$ of $\boldsymbol{S}$ (note that eigenvectors are orthogonal, so $\boldsymbol{u}_i^T \boldsymbol{u}_j = 0$ for each $i \neq j$).

The projected point $\tilde{\boldsymbol{x}}_n$ is given by:

$$\tilde{\boldsymbol{x}}_n = \overline{\boldsymbol{x}} + \sum_{i=1}^{M} [\boldsymbol{u}_i^T (\boldsymbol{x}_n - \overline{\boldsymbol{x}})] \boldsymbol{u}_i \tag{6}$$

Later we will demonstrate this expression. This projection has a small error (if M is large enough) compared to the original point.

An alternative method to determine the best projection would be to minimize the sum of these errors. Noteworthy, the result is the same, giving rise to an alternative PCA formulation (described in the next section).

Minimum-error formulation

Let $\{\boldsymbol{u}_i\}$ for $i=1,\ldots,D$ be a basis of orthonormal vectors. Any data point $\boldsymbol{x}_n$ can be approximated by:

$$\tilde{\boldsymbol{x}}_n = \sum_{i=1}^{M} z_{ni}\boldsymbol{u}_i + \sum_{i=M+1}^{D} b_i\boldsymbol{u}_i, \tag{7}$$

where the $\boldsymbol{z}_{ni}$ coefficients are particular for the nth data point and the b_i are constant.

As in the autoencoder approach presented in Unit 2, the loss or cumulative error is:

$$J = \sum_{n=1}^{N} (\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n)^T (\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n) \tag{8}$$

Minimum-error formulation (II)

J depends on the coefficients $\boldsymbol{z}_{ni}$, the constants b_i, and the basis $\{\boldsymbol{u}_i\}$. Setting derivatives with respect to $\boldsymbol{z}_{nj}$ and b_j to zero gives:

$$\frac{\partial J}{\partial z_{nj}} = 0 \implies -(\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n)^T \frac{\partial \tilde{\boldsymbol{x}}_n}{\partial z_{nj}} = 0 \implies z_{nj} = \boldsymbol{x}_n^T \boldsymbol{u}_j \tag{9}$$

$$\begin{aligned} &\frac{\partial J}{\partial b_j} = 0 \implies -\sum_{n=1}^{N} (\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n)^T \frac{\partial \tilde{\boldsymbol{x}}_n}{\partial b_j} = 0 \implies \\ &\sum_{n=1}^{N} \boldsymbol{x}_n^T \boldsymbol{u}_j - \sum_{n=1}^{N} \sum_{i=1}^{M} z_{ni} \boldsymbol{u}_i^T \boldsymbol{u}_j - \sum_{n=1}^{N} \sum_{i=M+1}^{D} b_j \boldsymbol{u}_i^T \boldsymbol{u}_j \implies \\ &N b_j = \sum_{n=1}^{N} \boldsymbol{x}_n^T \boldsymbol{u}_j \implies b_j = \overline{\boldsymbol{x}}^T \boldsymbol{u}_j \end{aligned} \tag{10}$$

Minimum-error formulation (III)

Thus, the error in the approximation to $\boldsymbol{x}_n$ is orthogonal to $\tilde{\boldsymbol{x}}_n$ and given by:

$$\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n = \sum_{i=M+1}^{D} \{(\boldsymbol{x}_n - \overline{\boldsymbol{x}})^T \boldsymbol{u}_i\} \boldsymbol{u}_i \tag{11}$$

Hence, the cumulative error, J, can be written as:

$$J = \sum_{n=1}^{N} \sum_{i=M+1}^{D} (\boldsymbol{x}_n^T \boldsymbol{u}_i - \overline{\boldsymbol{x}}^T \boldsymbol{u}_i)^2 = \sum_{i=M+1}^{D} \boldsymbol{u}_i^T \boldsymbol{S} \boldsymbol{u}_i \tag{12}$$

Minimum-error formulation (IV)

J still has to be minimized with respect to the basis $\{\boldsymbol{u}_i\}$. Let assume D=2 and M=1 and recall that basis vectors have unit length. Then, the constrained optimization can be expressed again with a Lagrange multiplier, which yields:

$$\frac{\partial \boldsymbol{u}_2^T \boldsymbol{S} \boldsymbol{u}_2 + \lambda_2(1 - \boldsymbol{u}_2^T \boldsymbol{u}_2)}{\partial \boldsymbol{u}_2} = 0 \implies \boldsymbol{S}\boldsymbol{u}_2 = \lambda_2 \boldsymbol{u}_2 \qquad (13)$$

Thus, $\boldsymbol{u}_2$ is an eigenvector of $\boldsymbol{S}$ and $J = \lambda_2$ is minimized by selecting λ_2 as the smallest eigenvalue of $\boldsymbol{S}$.

For arbitrary M, D values the $D - M$ smallest eigenvalues have to be selected and $J = \sum_{i=M+1}^{D} \lambda_i$.

Minimum-error formulation (V)

We have seen that $\boldsymbol{x}_n$ is:

$$\tilde{\boldsymbol{x}}_n = \sum_{i=1}^{M} z_{ni}\boldsymbol{u}_i + \sum_{i=M+1}^{D} b_i\boldsymbol{u}_i,$$

That is,

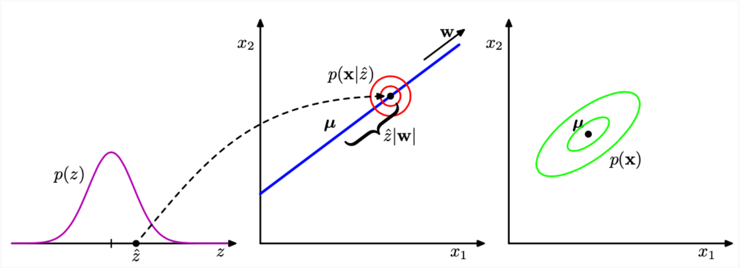
$$\tilde{\boldsymbol{x}}_n = \sum_{i=1}^{M}(\boldsymbol{u}_i^T\boldsymbol{x}_n)\boldsymbol{u}_i + \sum_{i=M+1}^{D}(\boldsymbol{u}_i^T\overline{\boldsymbol{x}})\boldsymbol{u}_i,$$

Since $\overline{\boldsymbol{x}} = \sum_{i=1}^{M}(\boldsymbol{u}_i^T\overline{\boldsymbol{x}})\boldsymbol{u}_i + \sum_{i=M+1}^{D}(\boldsymbol{u}_i^T\overline{\boldsymbol{x}})\boldsymbol{u}_i$, this leads to:

$$\tilde{\boldsymbol{x}}_n = \overline{\boldsymbol{x}} + \sum_{i=1}^{M}[\boldsymbol{u}_i^T(\boldsymbol{x}_n - \overline{\boldsymbol{x}})]\boldsymbol{u}_i$$

Probabilistic PCA

Probabilistic PCA



PPCA assumes that a point $\boldsymbol{x}$ is *generated* by first drawing a M-dimensional value $\boldsymbol{z}$ for a latent (hidden to observer) $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|0, \boldsymbol{I})$ variable and then drawing a value for $\boldsymbol{x}$ from the Gaussian variable $p(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}, \sigma^2 \boldsymbol{I})$. This model is governed by parameters $\boldsymbol{W}$, $\boldsymbol{\mu}$, and σ. $\boldsymbol{W}$ is a $D \times M$ matrix, while $\boldsymbol{\mu}$ is D-dimensional vector, and σ is a scalar. Contrary to PCA, which projects points onto a M-dimensional subspace, PPCA is better explained as a mapping from a latent space into the data space.

Probabilistic PCA (II)

It can be shown that the predictive distribution $p(\boldsymbol{x})$ is a Gaussian variable $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{C})$ with covariance matrix $\boldsymbol{C}=\boldsymbol{W}\boldsymbol{W}^T + \sigma^2\boldsymbol{I}$.

Given a set of observed data points $\{\boldsymbol{x}_n\}$, maximum-likelihood (ML) estimators for the parameters (not unique[1]) are:

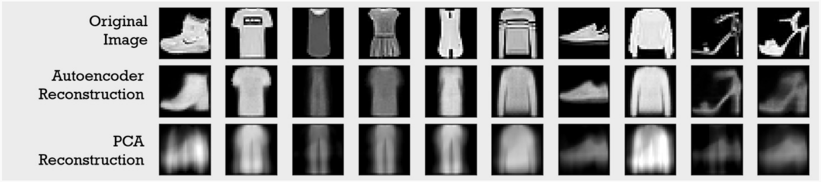
- $\widehat{\boldsymbol{\mu}}=\overline{\boldsymbol{x}}$
- $\widehat{\boldsymbol{W}}=\boldsymbol{U_M}(\boldsymbol{L_M} - \sigma^2\boldsymbol{I})^{1/2}$, where $\boldsymbol{L_M}$ is a diagonal matrix with the M largest eigenvalues of $\boldsymbol{S}$, and $\boldsymbol{U_M}$ is a $D \times M$ matrix whose columns are the M associated eigenvectors.
- $\widehat{\sigma}=\frac{1}{D-M}\sum_{i=M+1}^{D}\lambda_i$

[1] Any rotation applied to $\widehat{\boldsymbol{W}}$ is also a ML estimator of $\boldsymbol{W}$.

Applications of PCA

Feature extraction

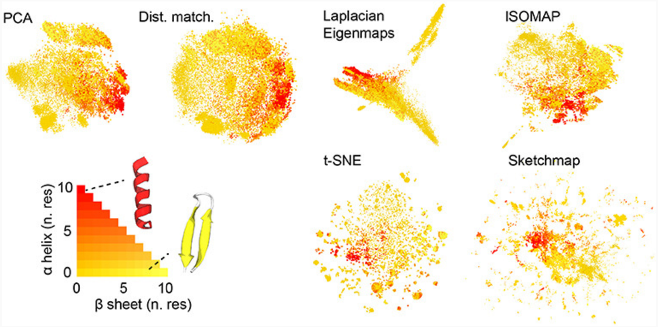
As the autoencoder model described in the Unit 2, PCA can be used to obtained a low dimensional data representation (coding) which can used as a feature space, for lossy data compression, denoising or other applications. As an example, next figure shows how the data are reconstructed from a PCA and autoencoder representations of same code dimensionality.



Data visualization

Usually, the length of the PCA coding M is selected such that $(\sum_{i=1}^{M} \lambda_i)/(\sum_{i=1}^{D} \lambda_i)$ is over some ratio, e.g., 0.9 (thus, the coding conserves at least 90% of the original data set variance).

Nevertheless, it is also possible to set M to 2 or 3-dimensions, so data can be visualized to try to gain some insight on it. The next figure shows an example for a protein dynamic problem.



Further reading

Further reading

As discussed in Unit 1, it is possible in many problems to transform a parameter-based model to an instance-based one, and, by using the kernel trick, do all computations in the input space without transformations to/from the feature space. PCA also allows a kernel-based formulation, known as kernel-PCA (see Bishop 12.3 or Geron ch. 8). The advantage of kernel-PCA is that it allows complex non-linear projections.

Many other dimensionality reduction techniques exist. For example, autoencoders, already discussed in Unit 2. Other methods are indicated in the figure of slide 21.