Independent Component Analysis: algorithms and applications

Kelthoum KERBOUA[a], Fatima BALDE[b]

[a]Ecole Normale Supérieur Paris-Saclay - Telecom Paris, kelthoum.kerboua@telecom-paris.fr, France
[b]Ecole Normale Supérieur Paris-Saclay - Telecom Paris, fatima.balde@telecom-paris.fr, France

Abstract

This work was carried out as part of the Paris Saclay MVA master's program for the Introduction to Probabilistic Graphical Models and Deep Generative Models module. It studies the "Independant component analysis" introduced in article (1) . We present the method that we implement ourselves, some of its applications and its limitations.

Keywords: Independent Component Analysis (ICA), Blind Source Separation, Denoising, Artefact Removal, Classification, Principal Component Analysis (PCA), Kernel PCA, Dimensionality Reduction

1. Introduction

ICA, which stands for Independent Component Analysis, was originally developed to solve problems similar to the cocktail party problem.

That problem can be summed up as follows: Given two microphones placed at two locations not far apart and two people speaking at the same time, we record a signal on each microphone, that signal represents the combination of the two coming from these individuals and we would like to recover the original ones.
Formally, we record $x_1 = a_{11}s_1 + a_{12}s_2$ and $x_2 = a_{21}s_1 + a_{22}s_2$ with s_1 and s_2 being the source signals we want to recover. However, ICA is not limited to this type of source separation problem, in fact it can be used for many other applications.

So, in order to carry out a complete study of ICA , we will in the next section explain in depth the modelling of the method. In section 3 , we present its implementation with the FastICA algorithm. In section 4, we look at some of the applications for which ICA can be used. Then we look at some of the limitations of this method and finally, in the last section, we compare ICA with existing methods that have applications in common with it.

2. ICA

2.1. Definition of the model

The ICA model is a generative statistical model that seeks to estimate latent variables from observations. Observations x_j are modelled as random variables equal to a linear combination of statistically independent sources s_i , $x_j = a_{j1}s_1+a_{j2}s_2+...+a_{jn}s_n$
If we denote X the vector whose components are the x_j , S the vector whose components are the s_j , A the matrix containing the a_{ij}, we can write the model as a matrix product X=AS
As well as assuming that our sources are statistically independent, we also assume that they are non-Gaussian (at most one can be Gaussian). This assumption is fundamental for ICA. In fact, if we give ourselves two Gaussian sources s_1 and s_2, and an orthogonal mixing matrix A, then the observations x_1 and x_2 will also be Gaussian and independent. Their joint distribution is then a symmetric Gaussian and therefore contains no information about the directions of the mixing matrix A, therefore A cannot be estimated.

There are also some ambiguities in the definition of ICA. The variance of the sources cannot be calculated exactly, dividing the variance by a value would be equivalent to multiplying the mixing matrix A by a constant. To avoid this ambiguity about the variance, we will assume that $E(s_i^2) = 1$, but this does not solve the problem about the sign of s_i. Also, the order of the components is not predefined. If A and S are the solution of ICA, then AP^{-1} and PS are also solution for P being a permutation matrix.

2.2. Non Gaussianity

Let's take the problem our ICA model is trying to solve: x=As.
First, we recall that the Central Limit Theorem stipulates that the sum of random variables i.i.d (independent identically distributed) tends towards a Gaussian variable. As a result, the linear combination of independent sources s_i generally has a distribution closer to the Gaussian than any of the original sources.
Let w be a vector, and $y = w^T x = w^T As = z^T s$ with $z^T = w^T A$, y is then a linear combination of the i.i.d sources s_i. Thus according to the Central Limit Theorem, y is closer to a Gaussian than each s_i unless it's equal to one of them.
Ideally, we would like w to be one of the row of the inverse of A, so that way, y should be equal to a source s_i. Therefore, the ICA problem is equivalent to finding the vectors w that maximise the non-Gaussianity of y so that it is as close as possible to the sources s_i.

2.3. Measure of non Gaussianity

Several methods can be used to measure the non-Gaussianity of a variable:

- The Kurtosis is defined by $kurt(y) = E(y^4) - 3(E(y^2))^2$. For a Gaussian random variable, we have $E(y^4) = 3(E(y^2))^2$, therefore the kurtosis coefficient is equal to zero. For almost all other variables the kurtosis is non-zero, making it a good measure of non-Gaussianity. Although very simple to compute, kurtosis can have drawbacks when calculated from samples (sensitivity to outliers, etc.).

- The negentropy is defined as $J(y) = H(y_{gauss}) - H(y)$ with H(y) being the entropy (i.e. the degree of information that the observations of the variable y provide) and y_{gauss} a Gaussian random variable of the same covariance matrix as y. It's a non-negative measure of non-Gaussianity. In fact, Gaussian variables have the highest entropies of all random variables of equal variance, so J(y) is only zero if y is Gaussian.
  However, computing the negentropy is not easy, therefore approximations to this value are needed. In the case of ICA, the most commonly used approximation is $[E(G(y)) - E(G(\nu))]^2$ where ν is a Gaussian variable of zero mean and unit variance. Furthermore, there is usually a choice between two non-quadratic functions G: $G_1(u) = \frac{1}{a_1} \log\cosh(a_1 u)$ where $1 \leq a_1 \leq 2$ (usually $a_1 = 1$) or $G_2(u) = -\exp -u^2/2$. The use of this approximation is recommended since it's easy to compute and robust.

3. FastICA

3.1. Preprocessing

To simplify the ICA algorithm, several prepocessing steps will be carried out:

- Centering: the observations are first centered ($E(X) = 0$).

- Whitening: This linear process, which follows the centering, allows the variances of the observations to be decorrelated and normalised ($E(XX^T) = Id$). Given the observations, we compute the SVD decomposition of the covariance matrix $E(XX^T) = EDE^T$, the whitening of X is then $\tilde{X} = ED^{-1/2}E^T X$

- Other prepocessing techniques, such as bandpass filters, can be applied in the case of time series data.

3.2. FastICA for one unit

The FastICA algorithm predicts directions w maximising the non-Gaussianity of $w^T x$ using the negentropy approximation defined above.

The algorithm is defined as follows for one unit:

1. Choose an initial (e.g. random) weight vector w.
2. Let $w_+ = E(xg(w^T x)) - E(g'(w^T x))w$
3. Let $w = w_+ / \|w_+\|$
4. If $< w_{old}, w_{new} > \neq 1$ (w_{old} and w_{new} don't point in the same direction) go back to 2

3.3. FastICA for several units

To predict several directions, we need to run the FastICA algorithm for one unit, several times. However, to avoid coming up with the same vector, the weight vectors w_i obtained must be decorrelated.

To do this, we can apply the Gram-Schmidt decorrelation process. Given the vectors $w_1, w_2, ..., w_p$, we estimate w_{p+1} with the FastICA and then apply these two steps:

1. Let $w_{p+1} = w_{p+1} \sum_{j=0}^{p} (w_{p+1}^T w_j) w_j$
2. Let $w_{p+1} = w_{p+1} / \sqrt{w_{p+1}^T w_{p+1}}$

4. Applications

The ICA model presented above has numerous applications, some of which we were able to test.

4.1. Simple Examples

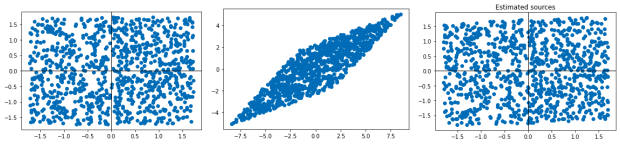

Figure 1: From left to right sources, observations, estimated sources computed with FastICA

Figure 2: Mix of images and extraction of sources with FastICA

4.2. Reducing noise in images

To denoise images with ICA , we use the Sparse Code Shrinkage method defined in (2). The model of a noisy image can be described as $z = x + \mu$ (where z is the noisy image, x is the true image and μ is Gaussian noise).
The matrix W is assumed to be such that $Wx = s$. Applying W to the noisy image gives $Wz = s + W\mu$. It was shown in (2) that W can be estimated using the fastICA on z .
Therefore, $Wz = s + W\mu$ represents a noisy version of the sources ($W\mu$ is Gaussian since W is orthogonal).
The model thus obtained can be written more simply as $y = s + \nu$.
To find the source s we will apply a maximum likelihood estimation.

Given p as the probability density of s and $f = -log(p)$, the maximum likelihood of s is given by $\hat{s} = argmin_u \frac{1}{\sigma^2}(y-u)^2 + f(u)$. A simple derivative calculation gives $\hat{s} - y = \sigma^2 * f'(\hat{s})$ assuming that f is differentiable.
An approximation of $\hat{s}$ is given by $\hat{s} = y - \sigma^2 f'(y)$ replacing $f'(\hat{s})$ by $f'(y)$ that can be observed.
Since this approximation is inacurate towards 0, we shrink the coefficients and $\hat{s} = max(0, |y| - \sigma^2 * |f'(y))|$
In our example, we use the Laplacian as our data distribution. Intuitively, since the Laplacian is a function with a sharp peak at 0, we can assume that the small values of y correspond to pure noise.
The laplacian density distribution for unit variance is: $p(s) = \frac{1}{\sqrt{2}} e^{-\sqrt{2}|s|}$, therefore, we have: $f'(y) = \sqrt{2}$.

The sparse code shrinkage algorithm can be summarised as follows:

- Decompose the image into several windows z_i, that we flatten
- Compute W using $z = (z_1, ..., z_n)$ in the FastICA algorithm
- Compute $y = Wz$
- Find the maximum likelihood estimate $\hat{s} = g(y)$ with $g(y) = max(0, |y| - \sigma^2 * \sqrt{2})$
- Reconstruct the denoised windows $X = W\hat{s}$

We implemented this denoising technique by extracting 20 sources using ICA, for 8*8 size windows on a image with a gaussian noise of variance 50. It should be noted that we have tried several ways of decomposing the image into windows of size 8*8. In fact, when we choose windows that do not overlap, the reconstructed image presents discontinuities between the edges of the windows.
We tried the same denoising technique on a larger image where the size of the window (8*8) would be relatively smaller than that of the image, so discontinuities are less visible, but we don't always have access to large images with high pixel resolution and noise is less noticeable on large images.
We have therefore tried to work with sliding windows to avoid discontinuities. When the image is reconstructed, the patches superimposed on each pixel are averaged. The latter technique gives the smoothest results.

Figure 3: Image Denoising using Sparse Code Shrinkage (image size:1024 x1024)

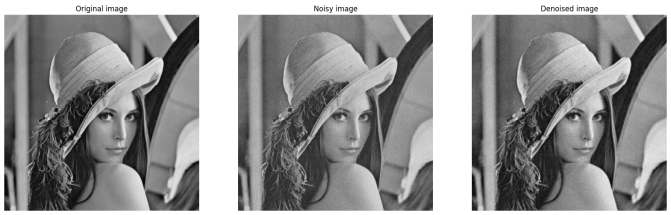

Figure 4: Image Denoising using Sparse Code Shrinkage (image size:3072 x3072)

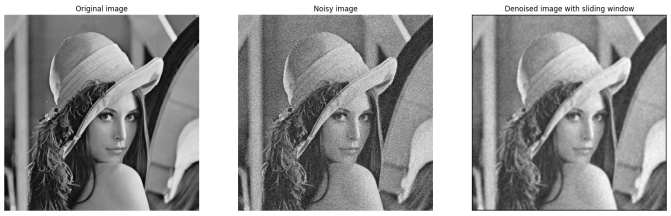

Figure 5: Image Denoising with sliding windows using Sparse Code Shrinkage

4.3. Heartbeat removal from MEG

MEG (Magnetoencephalography) signals are measurements of the magnetic fields generated by neural activity in the brain. These signals are used in neuroscience to study brain function and connectivity. Artefacts in MEG signals can arise from various sources, such as environmental noise, eye movements, muscle activity, heartbeats or non-neural physiological processes. These artefacts can contaminate the neural signals of interest, making it challenging to accurately interpret brain activity. We assume that those artefact are generated by anatomically and physiologically separated processes and therefore independent. By applying ICA, it becomes possible to identify and isolate the sources of artefacts from the neural signals, allowing for the removal or correction of unwanted interference.

To test the effectiveness of this method, we retrieved 12 MEG signals from the dataset provided by the Python library, MNE. These signals were selected because they showed clear signs of heartbeats and artifacts. The independent components are calculated using the FastICA algorithm. It is difficult to identify the origin of the artifacts and therefore to classify them. However, the signal corresponding to the cardiac artifact is easily identifiable and its extraction is therefore the aim of the maneuver.

The first figure shows MEG signals polluted by artifacts, clearly containing heartbeats. The second figure shows the independent components extracted from the dataset. The cardiac artefact is visible in first place.

4.4. Face Recognition

Another widely used application of ICA is face recognition described in the article (3). Here we use the YaleB32*32 face dataset containing 2414 facial images (size 32*32) of 38 individuals (64 near frontal images under different illuminations per individual). To perform face recognition, we separate our dataset into a training set with 80% of the shuffled dataset and a

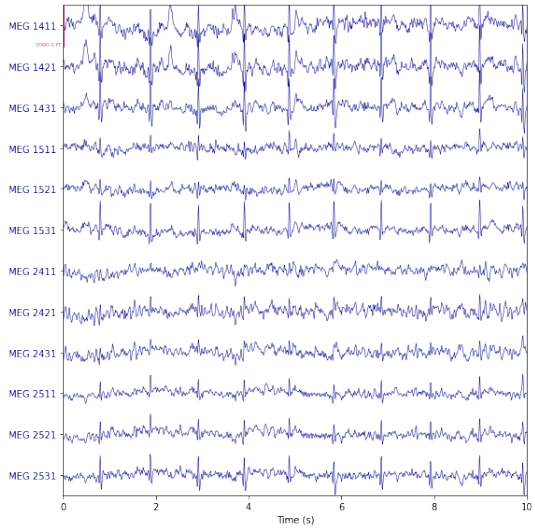

Figure 6: MEG signals with artefact and heartbeats

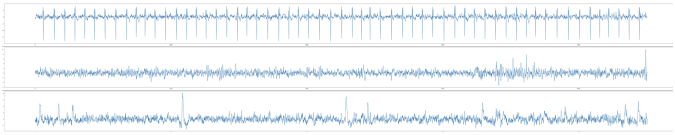

Figure 7: Independent components extracted from MEG signals

test set, then extract 150 independent sources S from the training set using FastICA. The features used for the classification correspond to the mixing matrix coefficients estimated for the training X_{train} and testing X_{test} set as $W = X * S^{-1}$ where S^{-1} is the generalized inverse (pseudo-inverse) of the non-square matrix S. A SVM classifier was then trained with the training features to identify the individual and tested with the test features. The results were greatly improved by performing histogram equalisation on the images beforehand. In fact, it reduces the effects of brightness differences between images, one of the main difficulties for the classification.

We obtain 99,6% of accuracy on the training set and 95% on the testing set.

4.5. Blind Source Separation

As mentioned in the introduction, ICA was introduced to solve blind source separation problems. We then repeated the cocktail party experience by mixing 3 audio sources and applying ICA to extract them. As can be seen in the figure above, the sources can be found from the mixtures, and even when listening, the audios and the ICA source estimates correspond perfectly.

5. Limitations of ICA

5.1. Gaussian or correlated sources

As mentioned in section 2, the ICA method is based on two assumptions: the original sources are non-Gaussian (at most one can be) and independent. This is the first limitation of this

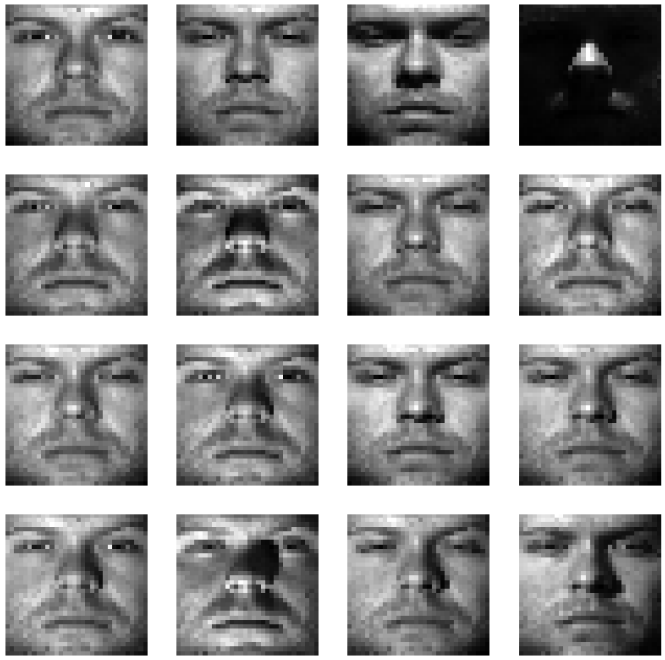

Figure 8: Examples from the dataset YaleB32*32

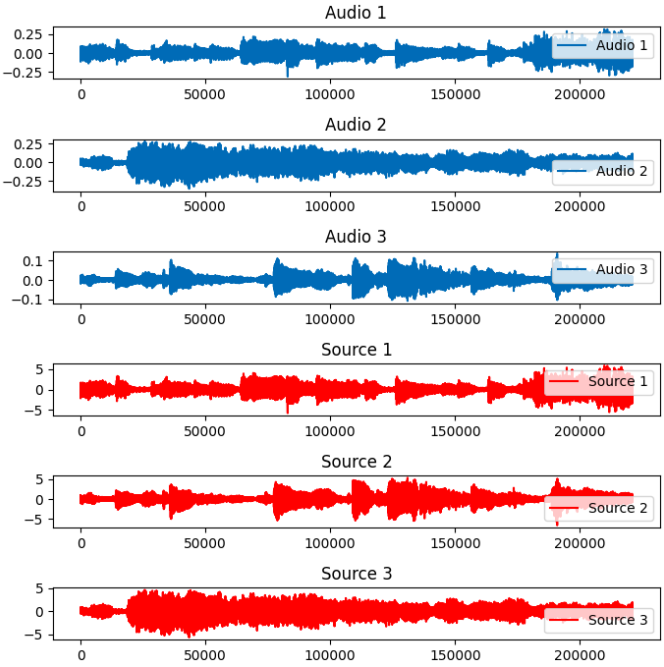

Figure 9: Blind Source Separation: Original sources and extracted sources by FastICA

technique. As a first step, we were able to test the performance of the FastICA algorithm by generating sources with Gaussian distributions. The figure below shows that the original sources are not found at the output of the algorithm.

We then tested FastICA on non-linearly correlated sources. Here's how we define our 3 original sources:
For $f = 5$ and $t \in [0, 1]$ we have $s_1 = sin(2*\pi*f*t)$, $s_2 = sin(2*\pi*f*t+0.5*sin(2*\pi*5*t))$ and $s_3 = sin(2*\pi*f*t+sin(2*\pi*3*t))$

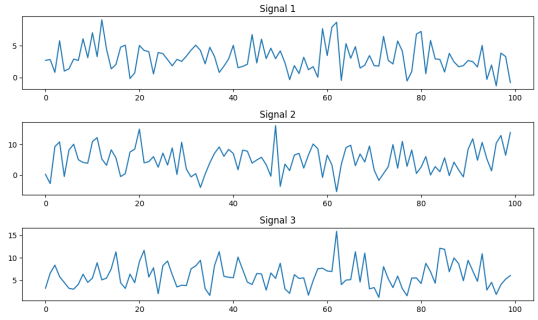

Figure 10: Experiments with Gaussian distributions: original sources

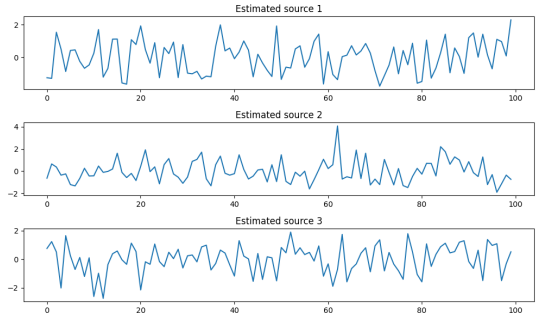

Figure 11: Experiments with Gaussian distributions: estimated sources

Once again, we don't reconstitute our original sources at the output.

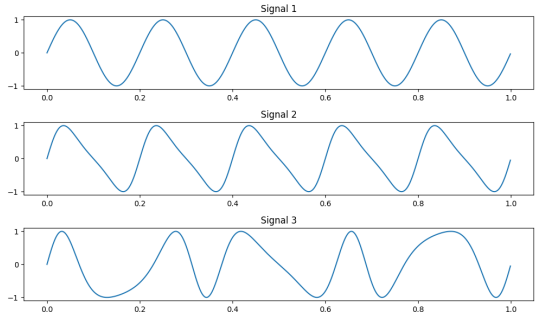

Figure 12: Experiments with correlated sources: original sources

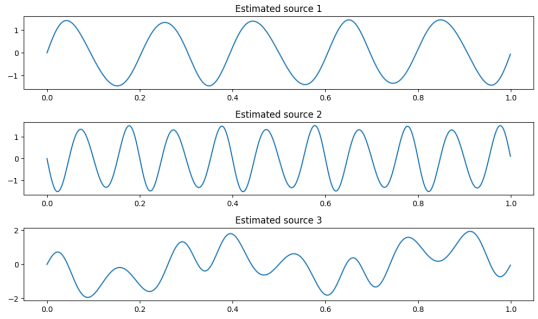

Figure 13: Experiments with correlated sources: estimated sources

5.2. Instability of ICA

Despite the important applications of ICA and its effectiveness in many cases (such as those mentioned above), this method is somewhat unstable. In fact, when the number of sources is greater than the number of observations, it is almost impossible to find these sources. You need at least as many sources as observations for ICA to work. We did the experiment by creating 3 mixtures of 5 audios. In this case, ICA sent back estimated sources that were mixtures of the original sources, but not the sources themselves.

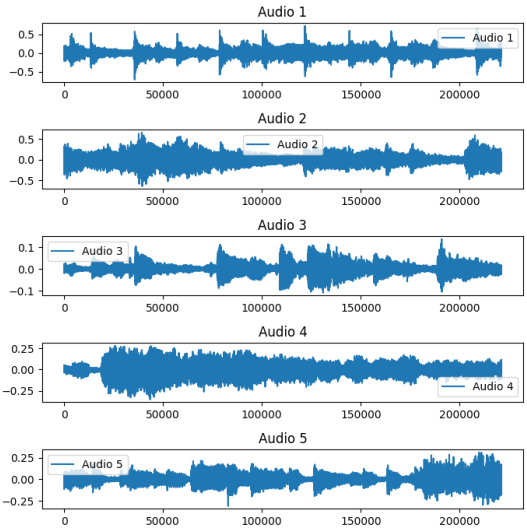

Figure 14: Instability of ICA: Original sources

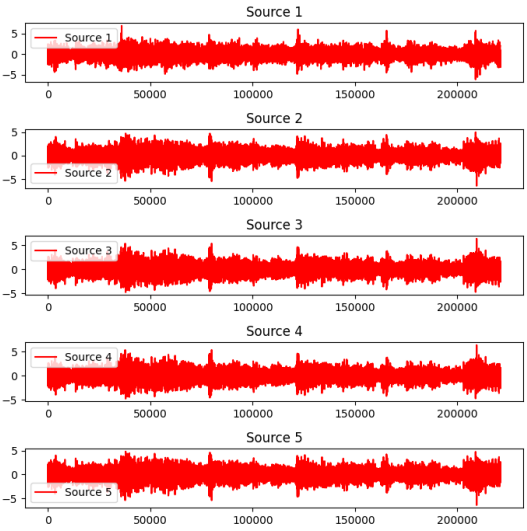

Figure 15: Instability of ICA: Estimated sources by FastICA

6. Comparison with other methods

Independent Component Analysis (ICA) differs from Principal Component Analysis (PCA) in that its main objective is to

extract independent components, rather than to reduce dimensionality by maximizing variance. In the context of dimensionality reduction, however, ICA can be compared with PCA and Kernel PCA. PCA, a precursor in this field, seeks to maximize linear variance, Kernel PCA extends this approach to non-linear spaces through the use of kernels, while ICA focuses on the separation of underlying sources. What they have in common is their ability to explore and extract meaningful structures from data, although their specific approaches and objectives differ. In practice, ICA can be used for dimensionality reduction while shedding particular light on the independent nature of the extracted components.
With this in mind, we have chosen to evaluate the performance of ICA for dimension reduction against other methods, in particular PCA and kernel PCA, in order to compare their effectiveness in this particular context.

6.1. PCA

Principal Component Analysis (PCA) is a dimensionality reduction technique widely used to extract the most significant features from a data set. It works by finding the principal components, which are linear combinations of the original variables, ordered by their decreasing variance. These components capture the main directions of variation in the data.
In practice, PCA transforms the original data into a new feature space where the dimensions are ordered by the size of their contribution to the total variance. As a result, PCA returns the principal components (orthogonal to each other), the eigenvalues (indicating the relative importance of the components), and the data projected into the new space. PCA is widely used for visualization, data compression and simplifying statistical analysis.

The PCA algorithm consists of several stages:

1. Data standardization: Center and scale the data.
2. Calculation of the covariance matrix: Measure linear relationships between original variables.
3. Calculation of eigenvectors and eigenvalues: Identify principal directions and their significance.
4. Eigenvector sorting: Order principal components by decreasing importance (eigenvalues).
5. Data projection: Transform data into the new space defined by the principal components.
6. Choosing the number of principal components: Select the number of components according to the proportion of variance to be preserved.

Thus, the directions returned by the ICA and the PCA are not the same. As explained, the PCA looks for orthogonal directions of greater variability of the data, whereas ICA extracts independent and not necessarily orthogonal sources, which linearly generate the data.

So, for a number of components lower than the number of samples, the PCA algorithm seeks to preserve the greatest amount of information by projecting the data into a space with the greatest variability. This notion of preserved information is not specified in the ICA, but the assumption of independence and non-orthogonality of the sources gives greater freedom in the choice of directions returned, which may be more appropriate in certain cases, as shown in the figure below.

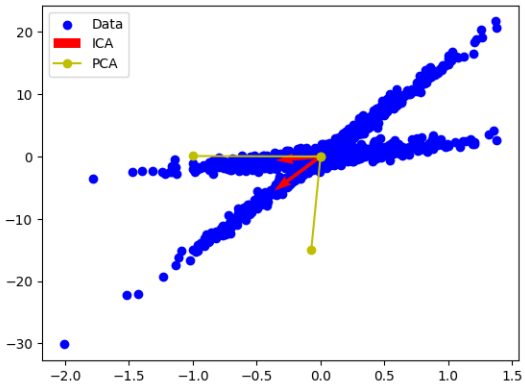

Figure 16: Directions returned by ICA and PCA

6.2. Image compression using PCA and ICA

Dimensionality reduction is often applied to data compression. We wanted to evaluate and compare the performance of the PCA and ICA algorithms, in this case on image compression of the MNIST dataset. The MNIST dataset comprises 70,000 images of handwritten digits, with 60,000 images used for model training (training set) and 10,000 separate images reserved for performance evaluation (testing set). Each image, measuring 28x28 (= 784) pixels, represents a single digit from 0 to 9. To achieve this, 200 principal components with PCA and 200 sources with ICA are extracted from the training dataset, and the images are then reconstructed as a linear combination of just these components. The figures below show the reconstitution of the test image from the components found on the training images. It's clear that the reconstruction after PCA is much better than that after ICA, which can be explained by the fact that the aim of ICA is not really to reduce dimensionality in contrast to PCA, but to extract independent sources and thus separate information. In addition, in contrast to ICA, we can estimate the amount of information preserved with just 200 components after PCA, in this case 46.67% (variance explained).

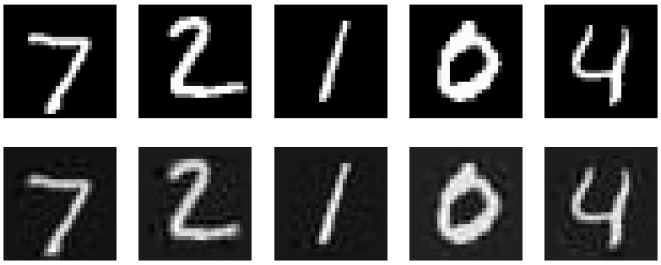

Figure 17: On the first line, the images from the testing set, on the line below, the reconstruction after compression by PCA

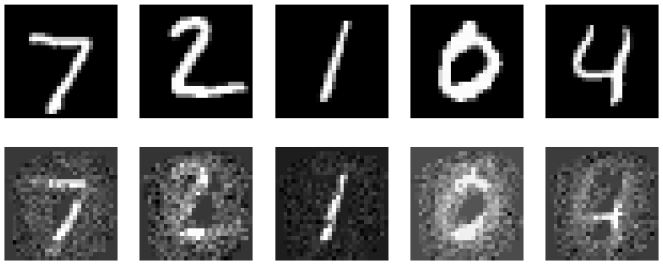

Figure 18: On the first line, the images from the testing set, on the line below, the reconstruction after compression by ICA

6.3. *Kernel PCA*

Kernel Principal Component Analysis (Kernel PCA) is an extension of the PCA algorithm, that enables non-linear data to be processed by introducing kernels. Unlike linear PCA, Kernel PCA projects data into a higher-dimensional feature space, enabling the efficient separation of complex structures. During projection into the feature space, Kernel PCA uses kernel functions to measure similarities between the original data pairs, generating non-linear representations. This makes it possible to capture more complex intrinsic structures that would not be well represented in the original feature space. Kernel PCA offers greater flexibility for modeling non-linear relationships, making it a wise choice when data present complex shapes, non-linear clusters or non-Gaussian distributions.

The Kernel PCA algorithm involves the following steps:

1. Calculation of the kernel matrix: Evaluate the similarity between each pair of samples using a kernel function.
2. Centering the kernel matrix: To obtain a centered representation, adjust the kernel matrix by subtracting the mean of each column and row.
3. Calculating eigenvectors and eigenvalues: Determine the eigenvectors and eigenvalues of the centered kernel matrix.
4. Sorting eigenvectors: Order the eigenvectors according to their eigenvalues to obtain the principal components.
5. Data projection : Project the original data into the new feature space defined by the extracted principal components.

In our implementation, we'll use the RBF kernel for projection into feature space. The Radial Basis Function (RBF) kernel, also known as the Gaussian kernel, is commonly used in the context of Support Vector Machines (SVM) and Kernel Principal Component Analysis (Kernel PCA). It measures the similarity between data pairs by assigning decreasing weights as a function of Euclidean distance.

Next, we introduce an example of how the Kernel PCA can be used more effectively to classify data. The original data is generated as a circle, which means that the classes are not linearly separable in the original space. PCA seeks to find a linear transformation that captures the maximum variance, which may not be optimal for non-linear data. KPCA, on the other hand, introduces a kernel that allows data to be projected into a space where class separation becomes easier.

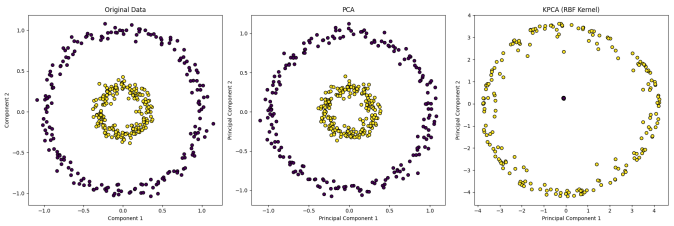

Figure 19: Comparaison PCA/KPCA

6.4. *Classification with PCA, KPCA & ICA*

The PCA, Kernel PCA and ICA algorithms are difficult to compare, as their applications differ according to the task at hand (dimensionality reduction, source separation, etc.) or the type of data being processed (independent, Gaussian, small quantities, complex and non-linear shapes). We have tried to assess the context in which each of these algorithms performs best, by means of data classification experiments.
The Iris dataset, provided by Python in the scikit-learn module, is a collection of measurements of four characteristics (sepal length and width, and petal length and width) of three different iris species (setosa, versicolor and virginica). Comprising 150 samples, with 50 samples per class, the Iris dataset does not contain a huge amount of data or features. The dataset is divided into training (70%) and testing (30%) sets. Classification was performed using a trained SVM after the extraction of 2 components by PCA, KPCA and ICA. We obtain the following results:

- Classification with PCA: Training accuracy: 0.962 / Test accuracy: 0.822
- Classification with ICA: Training accuracy: 0.686 / Test accuracy: 0.733
- Classification with KPCA: Training accuracy: 0.514/ Test accuracy: 0.511

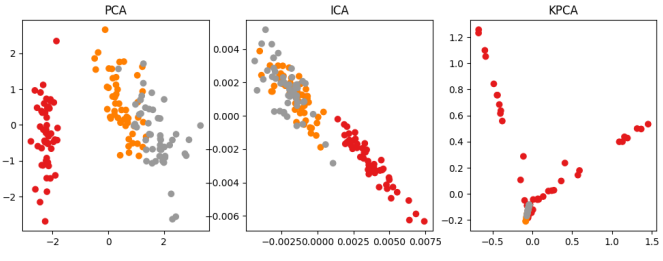

Figure 20: Extraction of 2 components from the dataset iris with PCA, KPCA and ICA

The data are not very complex, which means that the PCA performs better than all the others, and there's no need to use the KPCA in this case. The ICA still gives satisfactory results, which can be explained by the fact that the number

of extracted sources is very close to the starting dimension. The better performance of the PCA compared to the ICA can be explained by the fact that the PCA is more stable than the ICA and performs better on datasets with little data.

The digits dataset, provided by Python via the scikit-learn module, is a collection of grayscale images of handwritten digits from 0 to 9. Each image has a resolution of 8x8 pixels, giving a total of 64 features per sample. Comprising 1,797 samples, the dataset is more complex and complete than the iris dataset.

- Classification with PCA: Training accuracy: 0.596/ Test accuracy: 0.585
- Classification with ICA: Training accuracy: 0.239/ Test accuracy: 0.198
- Classification with KPCA: Training accuracy: 0.634 /Test accuracy: 0.615

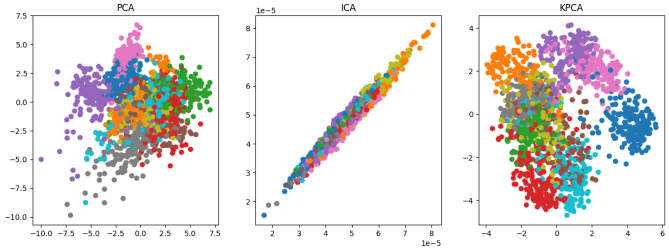

Figure 21: Extraction of 2 components from the dataset digits with PCA, KPCA and ICA

In this case, KPCA performs much better than all the other algorithms, which can be explained by the fact that the data are not linearly separable and contain more complex structures. ICA performs very poorly. Indeed, only 2 components are extracted, and ICA is not adapted to dimension reduction but to source separation.

References

[1] E. Oja A. Hyvarinen. Independent component analysis: algorithms and applications, 2000.

[2] E. Oja A. Hyvarinen, P. Hoyer. Image denoising by sparse code shrinkage, 1999.

[3] Oscar Deniz, Modesto Castrillón Santana, and Marycarmen Hernández. Face recognition using independent component analysis and support vector machines . volume 2091, pages 59–64, 08 2001.