Technical University of Denmark

Technical University of Denmark

Department of Applied Mathematics and Computer Science

02458 Cognitive Modeling

Assignment 3 Report

Homework 3 PCA and ICA

Jan Heimes

Examiner:

Tobias Andersen

processing period: 3 weeks

date of submission: 12.11.2020

I. Table of contents

II. Nomenclature

Acronyms

CDF	Cumulative distribution function
TPR	True Positive Rate
X	Sensor activation x
A	Weights / Mixing matrix
S	Stimulus
MEA	Mean absolute error
MSE	Mean squared error

III. List of Figures

IV. List of Tables

1 PCA

We load the image of Mona Lisa to python, by using the *imread-function* and and transform it to grayscale by using *rgb2gray*. We now have the image represented in python as an array of values.

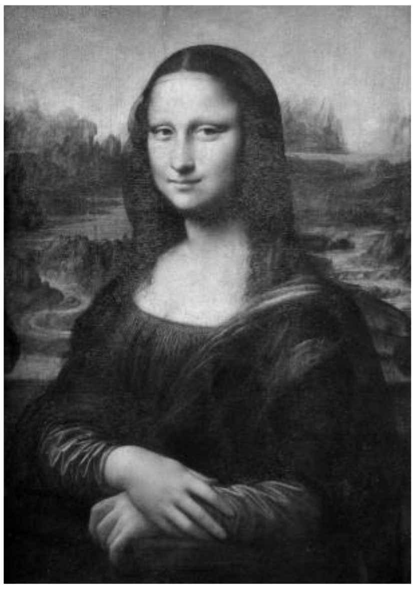


Figure 1 An image of Mona Lisa converted into grayscale.

We use extract patches 2d to 'extract' patches of size 10x10. This leaves us with a matrix of size (894621; 10; 10), which we'll reshape into a (894621; 100) matrix. To perform the principal component analysis (PCA), we use *sklearns* submodule PCA and the function *fit_transform()* to fit the model and apply dimensionality reduction.
We now have the **X** and **W** matrices and can perform a reconstruction of the Mona Lisa using the PCA. This is done by $\mathbf{X} * \mathbf{W}^{\mathbf{T}}$. Below is the reconstructed Mona Lisa.

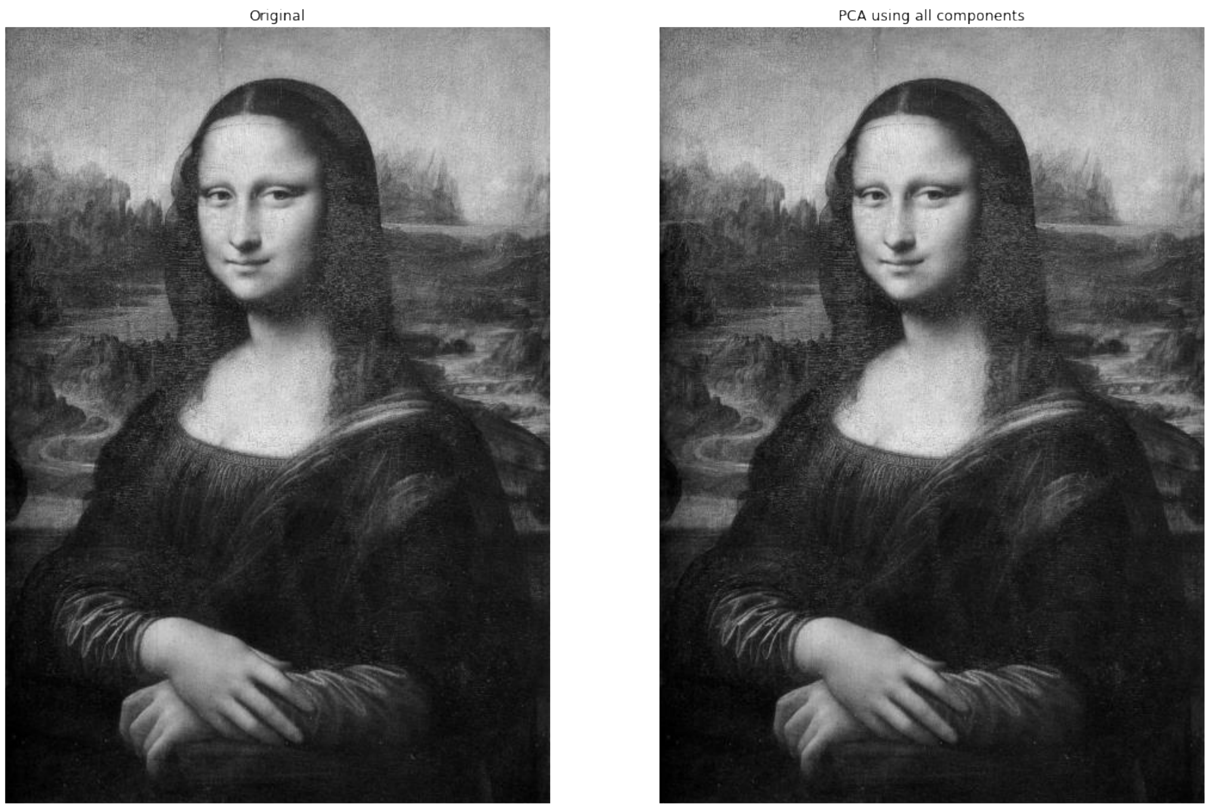
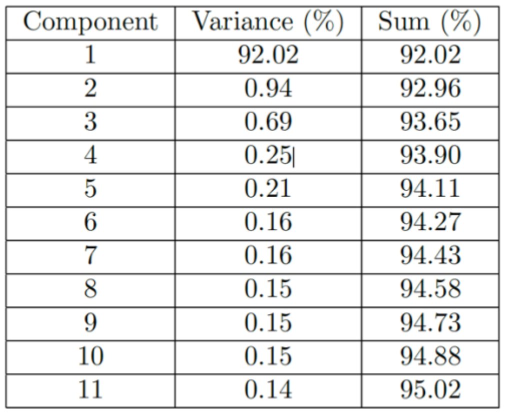


Figure 2 Image of Mona Lisa reconstructed besides the original black and white

It's very difficult to notice any difference in the original and the reconstructed image of Mona Lisa. However, using an error estimator, such as mean squared error (MSE), it is clearer that the two images aren't identical. Using MSE we get an error of 6479:9.

Below is a table showcasing the eigenvalues required to explain 95%, which we found to be 11 components.



Component	Variance (%)	Sum (%)
1	92.02	92.02
2	0.94	92.96
3	0.69	93.65
4	0.25	93.90
5	0.21	94.11
6	0.16	94.27
7	0.16	94.43
8	0.15	94.58
9	0.15	94.73
10	0.15	94.88
11	0.14	95.02

Table 1 As seen in above table, the first PCA explains 92% of the variance, but it will take 11 components to explain 95% of the variance.

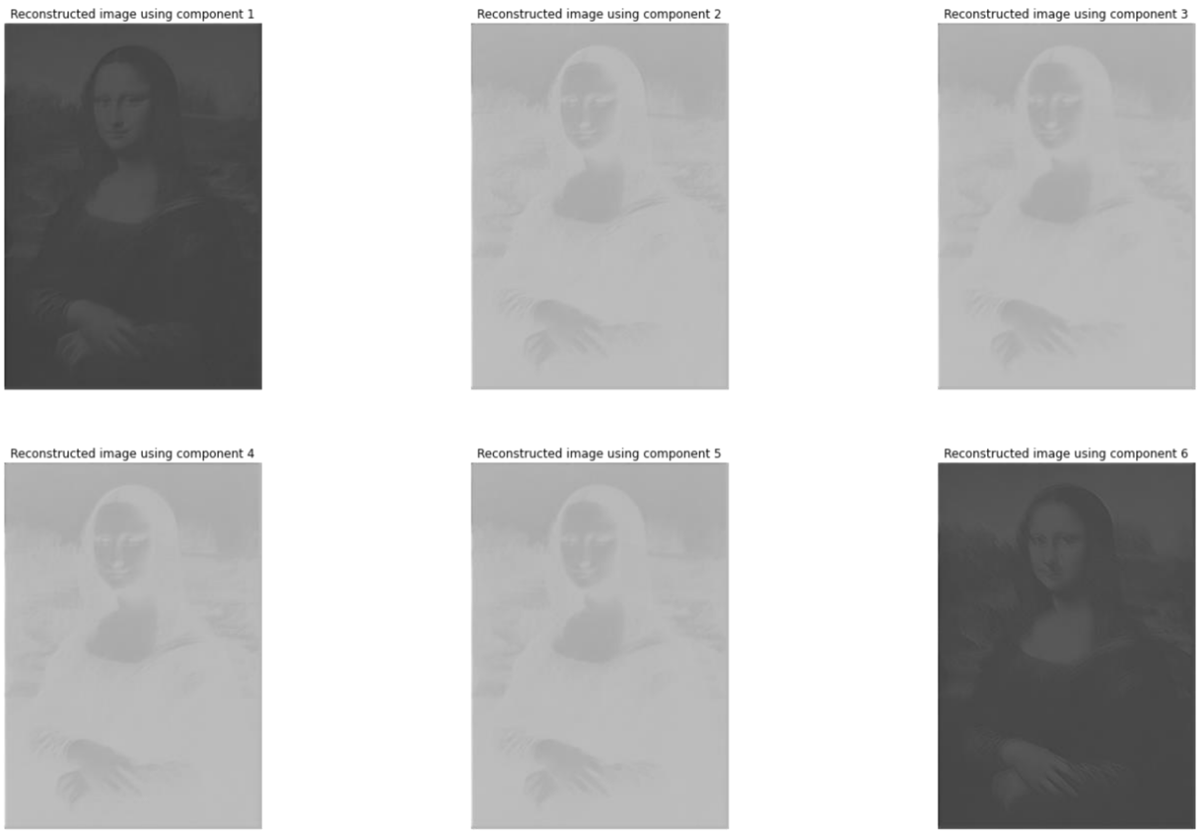

Reconstructed image using component 1
Reconstructed image using component 2
Reconstructed image using component 3
Reconstructed image using component 4
Reconstructed image using component 5
Reconstructed image using component 6

Figure 3 Reconstruction of Mona Lisa using the first 6 PC

Lastly, we want to reconstruct the image of Mona Lisa using all of the six first components together. Here it is easier to see a difference between the two images. It can also be measured using the MSE. Using MSE we get an error of 6683:76. This is a bit higher that before, where we used all the components, which is to be expected.

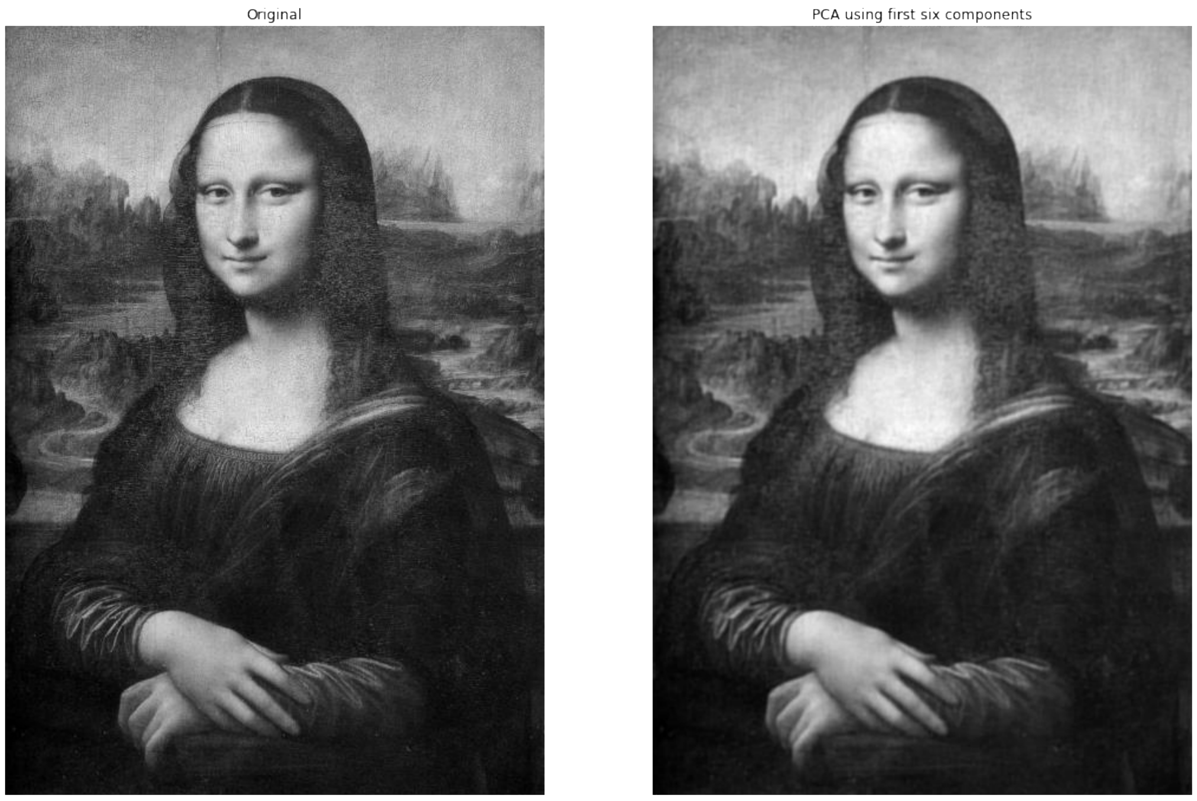


Figure 4 Reconstruction of Mona Lisa using six first components agains the original image of Mona Lisa

2 ICA

2.1 Creating Laplace distribution

First of all, it is needed to create 2 times 10000 samples from a Laplacian distribution. Since we worked with Python instead on MATLAB, we did not use the given *randpl* function. Instead we have used the *Laplace* function of the library *numpy*. However, we have created the Laplacian distribution with the function in MATLAB, as well to verify, that both approaches enable the same *Laplace* distribution, on which our further ICA process is set.
The two distributions are shown below. For further references the MATLAB and Python code are attached.

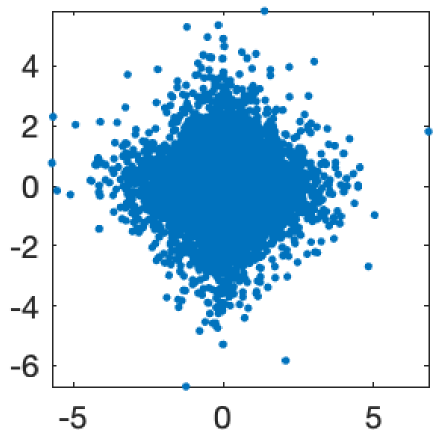


Figure 5 Laplace distribution created with MATLAB and the randpl function

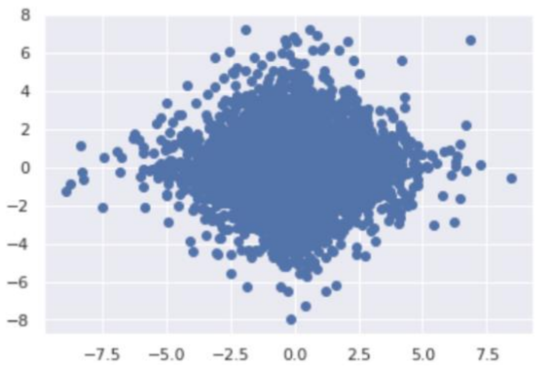


Figure 6 Laplace distribution created with Python and the laplace function of numpy

Hence the figure verifies, that both functions create a similar Laplace distribution. Hence it is verified, that the Laplace function of *numpy* is correct. In the following we are working with Python.

Afterwards we are mixing the distributions and creating a 2x2 Matrix **A** with random values to multiply with the Stimulus to get the sensor activation **X**.

$$\mathbf{X} = \mathbf{A} * \mathbf{S}$$

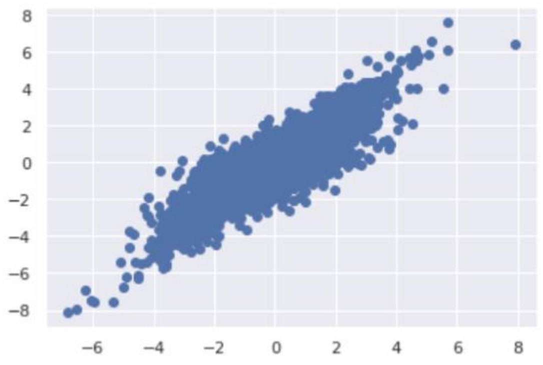


Figure 7 Distribution of X

2.2 PCA on the Laplace distribution

As we know from the Lecture about PCA and ICA we know, that PCA does not perform well on data, which has a covariance of 0. We can see that our Laplace distributed Stimulus is similar to the shape presented in the lecture with 0 covariance.

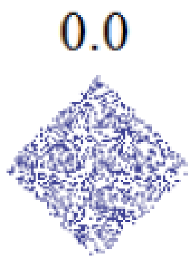


Figure 8 Complex distribution with 0 covariance

Hence, we can assume that the PCA won't do a great job on our data.
As expected with the PCA we obtain a Graph as shown below. The PCA did not separate the two distributions. It says that they totally depend on each other and you can represent one with the other.

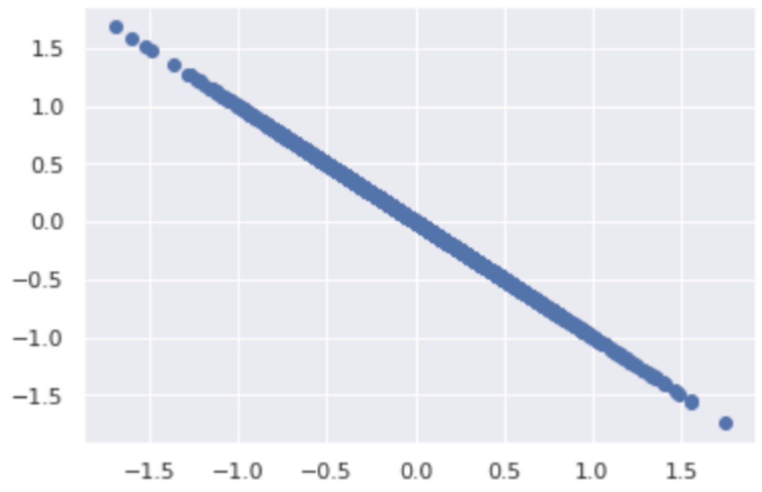


Figure 9 PCA on the mixed signals S

2.3 FastICA

Instead of the *FastICA* from MATLAB we are using the *FastICA* Function imported from *sklearn* for Python.
As a result, we can observe that the *FastICA* did a good job. It has recreated our Stimulus **S** from the beginning pretty well, based on the sensor activation **X** created above and shown in Figure 7. Instead of PCA, which uses Gaussian distributions and minimizes covariance, ICA maximizes kurtosis on non-orthogonal components. The distribution now appears independed.

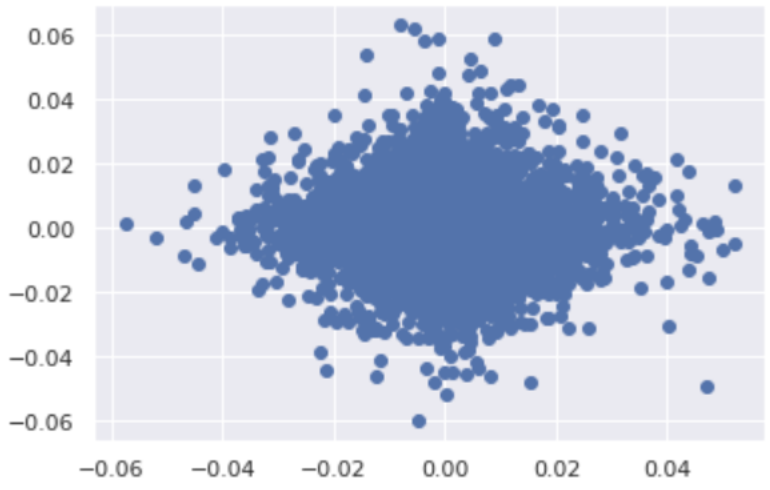


Figure 10 Recovered Stimulus distribution with ICA

While comparing the original samples with the ones created by the ICA it is to observe, that the proportions look in general correct. However, plot of the initial Stimulus represents more the shape of a proper square, while the *FastICA* created diamond shaped distribution. It looks like the distribution is wider in the x direction than in the y direction. Also, the x and y axis have different units due to the fact, that the ICA normalized the to perform.
The ICA can go wrong if we have gaussian signals, because the components that are obtained could have been produced by some other arbitrary mixing of the sources too and therefore ICA is not possible to figure out the original sources.

2.4 Analyzing sound

For the last exercise we have not used the guitar sounds. Instead some other wav files, because the given wav files have been 24-byte files and that has caused some errors.

We read in the wav files as lists of floats. We can get the params from the first and second record. While analyzing them we can see that it has only channel (mono sound). It has a frame rate of 44100. This means that the sound is represented each second by 44100 integers. The file has a total of 264515 integers/frames, which means its length in seconds is around 6 seconds.

We can plot the mix one as well as mix two based on their arrays, which contain the integers, which represent the sound. Normalizing the original signal array allows us to easier compare the resulting sound created with the ICA with the original signals afterwards. We plot the values of the signal over the timing, when they occur.

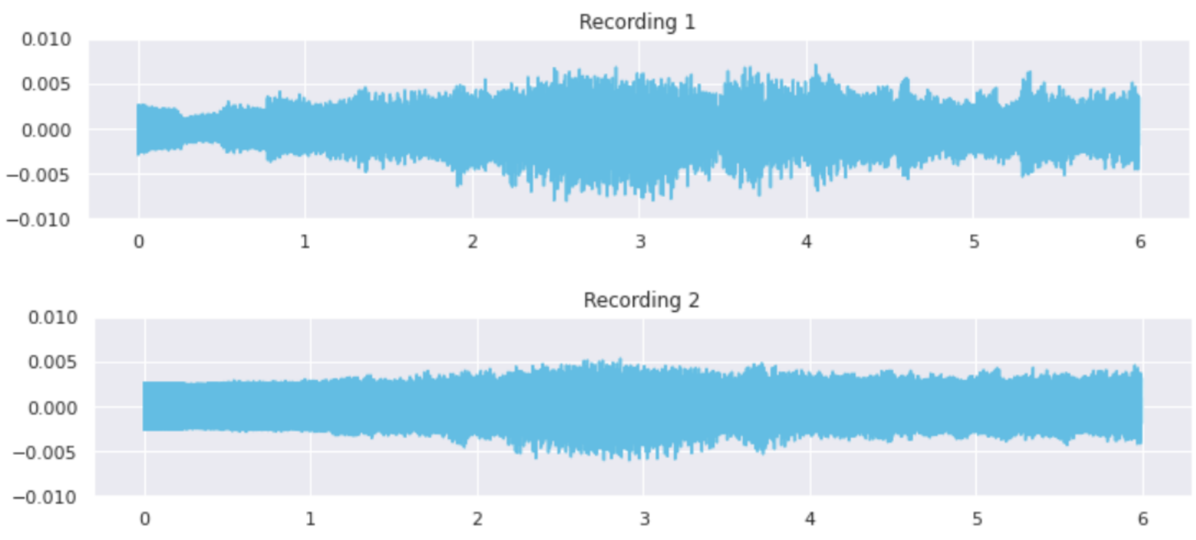


Figure 11 Plot of the two sound files

We can see the classic shape of a sound file within an normalized amplitude of 0.1 to -0.1.

Afterwards we are creating the dataset based on the using the *FastICA* as we have already done previously.

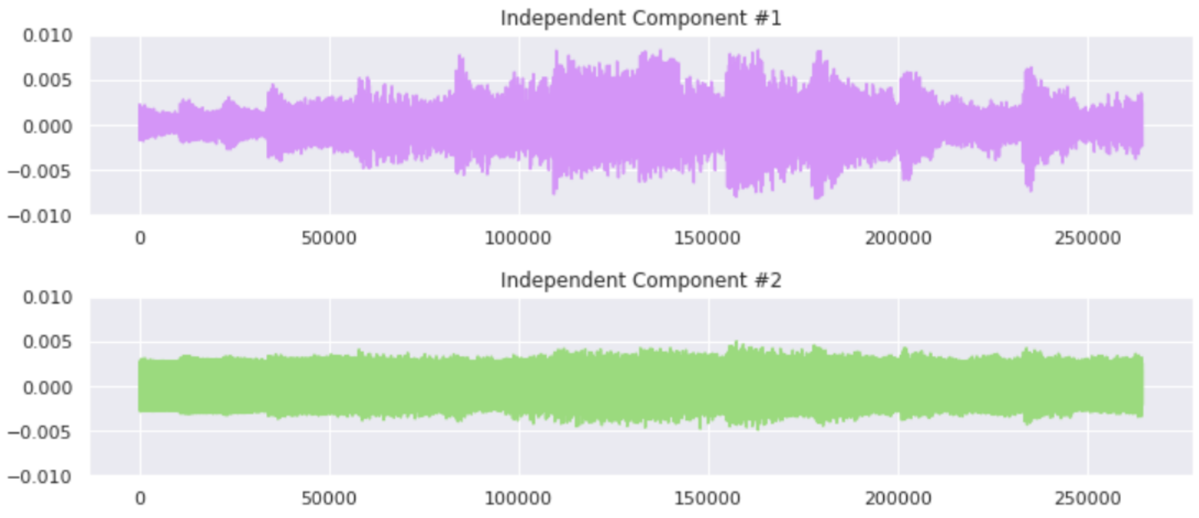


Figure 12 Resulting sound plots with ICA

As we can see from Figure 12 is that ICA does a good job on reconstruction the sound. The first plot has more different aplitudes than the second. The second is differes less in terms of the sound aplitude. The ICA is able to reconstrut this correctly. However, for the first sound the ICA creates a more abstract plot than it has originally been.

The mean absolute error:

$$MEA = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

For the first sound we get a MEA of 0.00280

For the second sound we get a MEA of 0.00048

Hence, we can verify that the ICA does a precise reconstruction of the original sound. Now, we save the files as wave files. This requires: convert them to integer (so we can save as PCM 16-bit Wave files), otherwise only some media players would be able to play them. A basic mapping can be done by multiplying by 32767. The sounds will be a little faint, we can increase the volume by multiplying by a value with 100.

We can conclude that both the mathematical comparison about MEA and the comparison of us that we hear agree. The resulting melody and original melody match. There is some background noise in the resulting melody and the volume of the different instruments relative to each other does not match entirely. However, it can be said that ICA produces very good results for this problem.

But even if the sounds are flipped, since the signals are oscillatory it corresponds to flipping a cosine wave (it just becomes a sine wave, i.e. it introduces a small temporal delay). This is not noticeable to the ear as the sound sampling is 44kHz so 1/44k of a second difference is negligible. So, in this case we would have a good qualitative error (no difference when we listen) but high MSE. In a nutshell it depends on how we calculate the quantitative error and whether our ICA got the right sign or not. In our case the two results do not differ but, ICA does not know the original sign, hence if we would do a mean squared error (MSE) and the sign of the reconstructed sound is flipped, it will have very large errors. However, if we use MAE your two methods will not differ.