CS408: Statistical Pattern Recognition

Independent Component Analysis & Kernel PCA

Project Report

Ameya Vadnere (170010002) & Deepak H R (170010026)

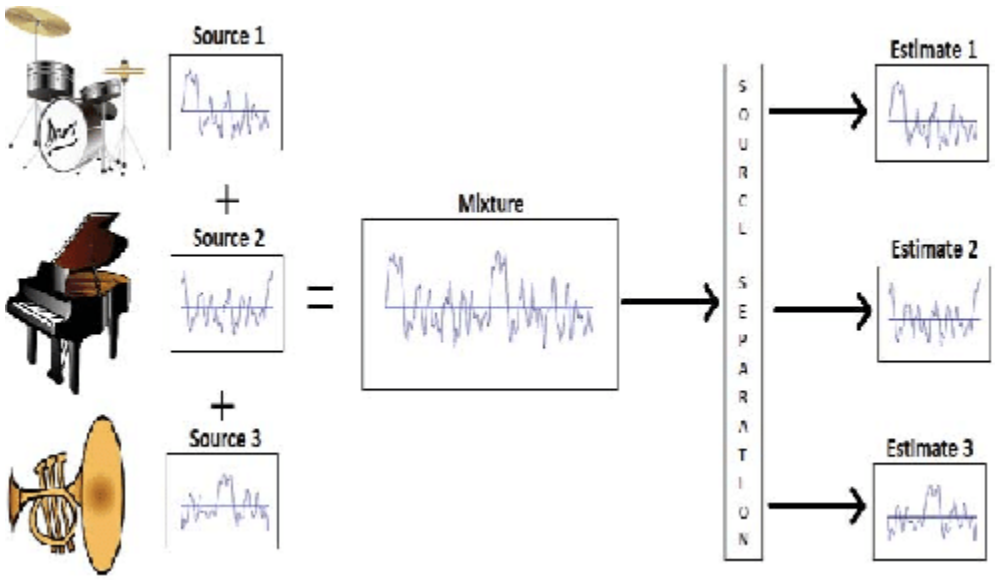
7th April, 2021

Independent Component Analysis

Introduction and Motivation

Blind Source Separation Problem

Source separation, blind signal separation (BSS) or blind source separation, is the separation of a set of source signals from a set of mixed signals, without the aid of information (or with very little information) about the source signals or the mixing process. The objective is to recover the original component signals from one or more mixture signals. The classical example of a source separation problem is the **cocktail party problem**, where several people are talking simultaneously in a room (for example, at a cocktail party), and a listener is trying to follow one of the discussions. The human brain can handle this sort of auditory source separation problem, but it is a difficult problem in digital signal processing.



Cocktail Party Problem

At a cocktail party, there is a group of people talking at the same time. Suppose we record their speech using multiple microphones. Each recording will consist of a mixture of speeches of various people. The cocktail party problem deals with isolating one or each individual's speech out of the mixture. Blind Source Separation can be used to separate the individual sources by using mixed signals.

Independent Component Analysis (ICA) provides a solution to this problem. ICA can be performed through various algorithms like FastICA, Projection pursuit and Infomax. Our focus in this project is ICA through the FastICA algorithm, which extracts independent components from mixed signals by maximizing the non-Gaussianity, which we will soon discuss.

Independent Component Analysis

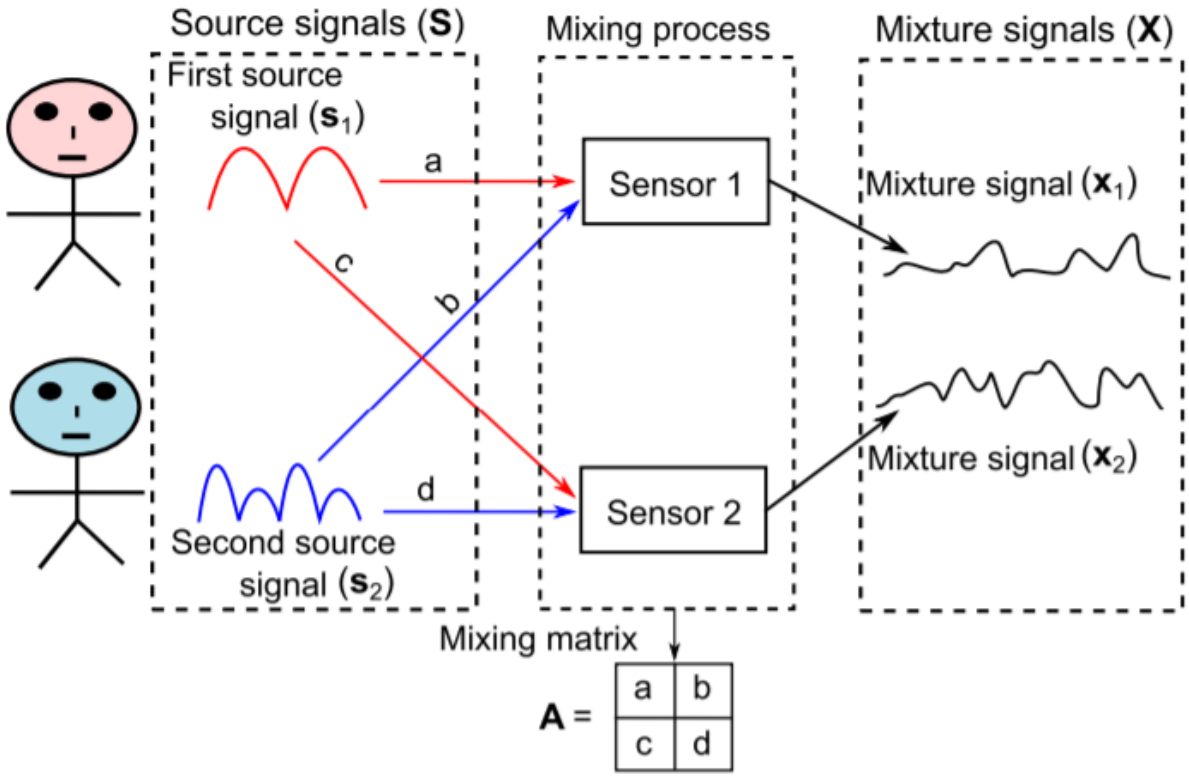
Defining ICA

Assume that in a cocktail party, you have two microphones recording time signals. Let the amplitudes of these mixed recorded signals be $x_1(t)$ and $x_2(t)$ at the time index t. Each of these recorded signals is a weighted sum of the speech signals emitted by the two signals, which we denote by $s_1(t)$ and $s_2(t)$, which are the original source signals. Expressing this as linear equations, we get:

$$x_1(t) = a_{11}s_1 + a_{12}s_2 \quad (1)$$

$$x_2(t) = a_{21}s_1 + a_{22}s_2 \quad (2)$$

Given only $x_1(t)$ and $x_2(t)$, we want to figure out $s_1(t)$ and $s_2(t)$.



Writing Eq. 1 and Eq. 2 in matrix form, we now have:

$$\mathbf{x} = \mathbf{A}\mathbf{s}.$$

Where **x** is a k-dimensional vector, where k corresponds to the number of microphone recordings. **A** is a k x p matrix. And **s** is a p-dimensional vector, where p is the total number of source signals, or in this case, the number of people talking.

We can also incorporate the time index in the above equation. In that case, the equation would become:

$$\mathbf{X} = \mathbf{AS}$$

Considering the signals are recorded over N time steps, the shape of **X** is k x N, **A** is k x p and **S** is p x N.

Throughout this project, we will not deal with underdetermined and overdetermined systems of linear equations. We assume that k = p.

The above equation models our ICA approach. Given only **X,** we need to estimate the mixing matrix **A**, after which we can compute its inverse **W**, thus:

$$\mathbf{S} = \mathbf{WX}$$

Assumptions in ICA

- **Independence of Source Signals:**

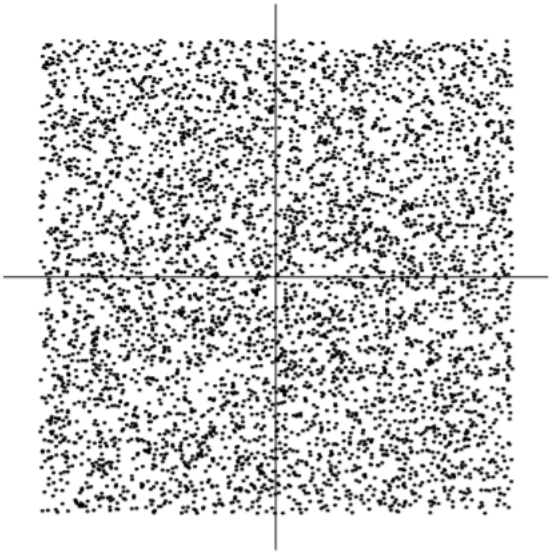
  It turns out that while solving the above problem (2-person), it is enough to assume that $s_1(t)$ and $s_2(t)$ at each time index t are statistically independent. This is not an unrealistic assumption in many cases, but it might not hold in practice depending on the situation.

- **Non-Gaussianity of Source Signals:**

Consider two independent components have the following distribution:

$$p(s_i) = \begin{cases} \frac{1}{2\sqrt{3}} & \text{if } |s_i| \leq \sqrt{3} \\ 0 & \text{otherwise} \end{cases}$$

Their joint distribution would look like this:

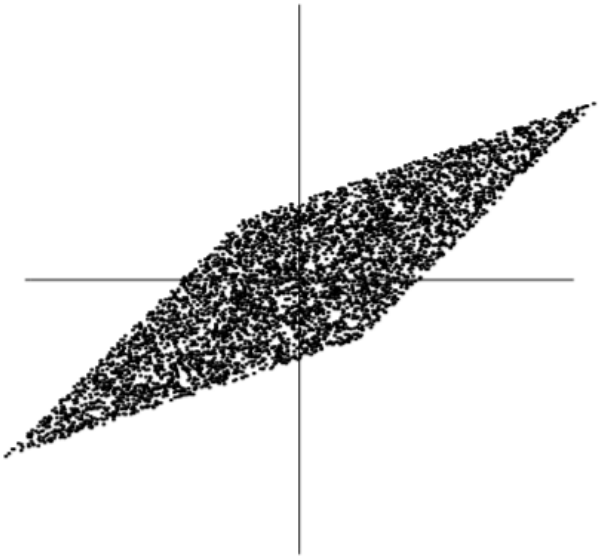


The horizontal and vertical axes are s_1 and s_2 respectively.

Now, suppose we mix s_1 and s_2 with the matrix:

$$\mathbf{A}_0 = \begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix}$$

We get x_1 and x_2 (which are not statistically independent). Their joint distribution would look like this:



Intuitively, from the above figure, you can obtain some information about the mixing matrix **A**. The edges of the parallelogram are in the direction of the columns of **A**. So, roughly one can know something about the matrix **A**, given the above information. Note that this is only a rough and tenuous explanation for conveying the intuition.

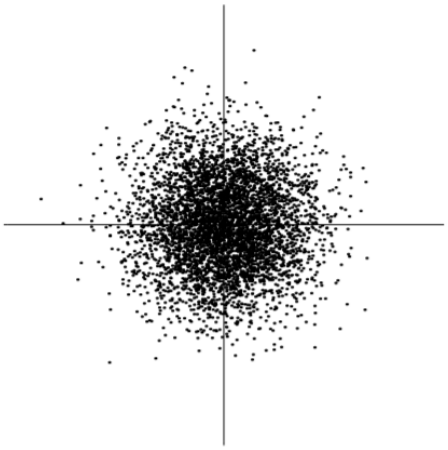
Now, assume that both the signals are Gaussian and that the mixing matrix **A** is orthogonal:

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

After mixing, we get x_1 and x_2, which are Gaussian, uncorrelated and of unit variance. Their joint density is given by:

$$p(x_1, x_2) = \frac{1}{2\pi} \exp\left(-\frac{x_1^2 + x_2^2}{2}\right)$$

It would look like this:



Intuitively speaking, the above distribution contains no information about the columns of **A**, because it is rotationally symmetric. Hence, it is not possible to estimate **A**.

A rigorous argument: It can be proved that no matter what the orthogonal mixing matrix is, an orthogonal transformation applied to (s_1, s_2) will yield the same distribution $p(x_1, x_2)$. So, the mixed signals we observed after mixing could have occurred from any orthogonal transformation (orthogonal matrix).

Ambiguities of ICA

1. We cannot determine the variances of the independent components.

   This is because we only know **X**, and not **A** and **S**. Suppose we have one of the sources $\mathbf{s_i}$. If we multiply $\mathbf{s_i}$ by some scalar, we can also divide the corresponding column in $\mathbf{a_i}$ in **A** to get the same $\mathbf{x_i}$ in **X**. So we decide to fix the variance of the independent components as 1.

2. We cannot determine the signs of the independent components.

   As explained above, if we multiply $\mathbf{s_i}$ by -1, we can make corresponding changes in **A** to get the same **X**. This ambiguity is, fortunately, insignificant in most applications.

3. We cannot determine the order of the independent components.

   We can account for the permutations in **S** by accordingly setting the columns in **A**.

Principles of ICA Estimation

Why maximize non-Gaussianity

Intuitively speaking, the key to estimating the ICA model is non-Gaussianity.

According to the Central Limit Theorem, the distribution of a sum of independent random variables tends towards a Gaussian distribution, under certain conditions. Thus we can say that a sum of two or more random variables is more Gaussian in nature than any of the original random variables, a useful fact our further approach is based upon.

Now consider the ICA model $\mathbf{x} = \mathbf{As}$ (We are dropping the time index). Note that $\mathbf{W} = \mathbf{A}^{-1}$. For simplicity, let us assume that all the independent components have identical distributions. We want to estimate one of the independent components. Let $\mathbf{w}$ be one of the rows in $\mathbf{W}$. Now consider the linear combination of the x_i, let's denote this by $y = \mathbf{w}^T\mathbf{x}$, where $\mathbf{w}$ is to be determined. If $\mathbf{w}$ were one of the rows of $\mathbf{A}^{-1}$, we would have y as the independent component we want to estimate. So, how can we determine $\mathbf{w}$ so that it would equal one of the rows of $\mathbf{W}$, the inverse of $\mathbf{A}$? In practice, we cannot determine such a $\mathbf{w}$ exactly, but we can arrive at a good approximation. We'll now see how we can use the Central Limit Theorem to do this.

Let's define $\mathbf{z} = \mathbf{A}^T\mathbf{w}$. Thus, we have $y = \mathbf{w}^T\mathbf{x} = \mathbf{w}^T\mathbf{As} = \mathbf{z}^T\mathbf{s}$. Observe that y is a linear combination of all the s_i, with the weights defined by z_i. So, according to the Central Limit Theorem, $\mathbf{z}^T\mathbf{s}$ is more Gaussian than any of the individual s_i. Observe that $\mathbf{z}^T\mathbf{s}$ will become least Gaussian when it in fact equals one of the s_i. In that case, only one of the elements z_i will be non-zero. Thus, our objective now becomes to maximize the non-Gaussianity of $\mathbf{z}^T\mathbf{s} = \mathbf{w}^T\mathbf{x}$. We know $\mathbf{x}$, thus if we find a $\mathbf{w}$ which maximizes the non-Gaussianity of $\mathbf{w}^T\mathbf{x}$, we would in fact get s_i, which is one of the independent components.

A measure of non-Gaussianity: Negentropy

There are several measures of the non-Gaussianity of a random variable. For this project, we will focus on Negentropy.

Entropy H(**y**) (differential entropy) for a continuous random vector **y** with density f(**y**) is defined as:

$$H(\mathbf{y}) = -\int f(\mathbf{y}) \log f(\mathbf{y}) \mathrm{d}\mathbf{y}.$$

A fundamental result in Information Theory states that **a Gaussian random variable has the largest entropy amongst all random variables of equal variance**.

We now define the notion of Negentropy *J*:

$$J(\mathbf{y}) = H(\mathbf{y}_{gauss}) - H(\mathbf{y})$$

Here, $\mathbf{y}_{\text{gauss}}$ is a Gaussian random variable with the same covariance matrix as **y**. Negentropy is always non-negative, and it is zero if and only if y has a Gaussian distribution.

The advantage of using negentropy as a measure of non-Gaussianity is that it is well justified by statistical theory. However, calculation of negentropy is computationally very difficult. Hence, we use simpler approximations of negentropy, which are discussed further.

Approximations of negentropy

Based on the maximum-entropy principle, a good approximation to negentropy has been proposed as:

$$J(y) \approx \sum_{i=1}^{p} k_i [E\{G_i(y)\} - E\{G_i(\nu)\}]^2,$$

Here, k_i are some positive constants, y is a variable with zero mean and unit variance, v is a standardized Gaussian variable (zero mean and unit variance), and G_i are some non-quadratic functions.

If we use a single non-quadratic function G, the approximation becomes:

$$J(y) \propto [E\{G(y)\} - E\{G(\nu)\}]^2$$

Using appropriate functions for G can lead to good results. In particular, choosing G that does not grow too fast can yield robust estimators. The following choices of G have proved effective:

$$G_1(u) = \frac{1}{a_1} \log\cosh a_1 u, \quad G_2(u) = -\exp(-u^2/2)$$

Here, $1 <= a_1 <= 2$ (some suitable value)

Preprocessing for ICA

Centering

To center **x** is to subtract its mean vector from itself (can be generalized for various time steps), i.e **mean(m)** = E(**x**). After the mixing matrix is found, one can get the mean of the independent components by doing $\mathbf{A}^{-1}\mathbf{m}$.

Whitening

After centering, it is also useful to whiten the observed variables. We transform the observed vector **x** linearly such that the new vector **x-tilde** which is white, i.e its components are uncorrelated and their variances equal 1.

$$E\{\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T\} = \mathbf{I}.$$

One popular method for whitening is the Eigen-Value Decomposition (EVD) of the covariance matrix.

$$E\{\mathbf{x}\mathbf{x}^T\} = \mathbf{E}\mathbf{D}\mathbf{E}^T,$$

After we get **E** and **D,** whitening can now be done by:

$$\tilde{\mathbf{x}} = \mathbf{E}\mathbf{D}^{-1/2}\mathbf{E}^T\mathbf{x}$$

Thus,

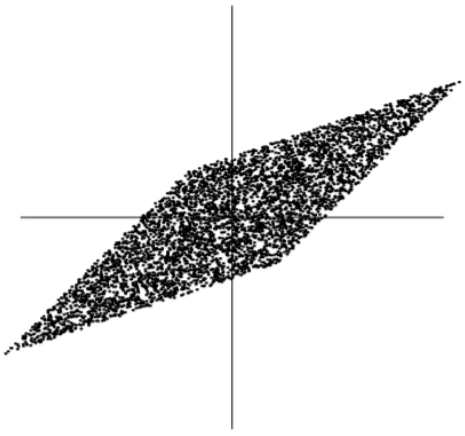
$$\tilde{\mathbf{x}} = \mathbf{E}\mathbf{D}^{-1/2}\mathbf{E}^T\mathbf{A}\mathbf{s} = \tilde{\mathbf{A}}\mathbf{s}$$

Whitening is useful because now, the new mixing matrix $\tilde{\mathbf{A}}$ is orthogonal.

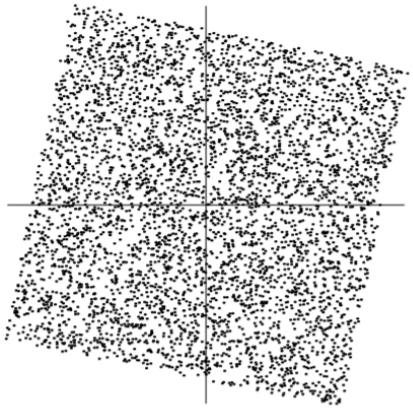
$$E\{\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T\} = \tilde{\mathbf{A}}E\{\mathbf{s}\mathbf{s}^T\}\tilde{\mathbf{A}}^T = \tilde{\mathbf{A}}\tilde{\mathbf{A}}^T = \mathbf{I}.$$

Now, instead of estimating n^2 parameters, we have to estimate only about $n(n-1)/2$ parameters, thus reducing it significantly.

Before whitening (x_1 and x_2):



After whitening:



The FastICA Algorithm

FastICA for one unit

The following algorithm finds a direction, i.e a unit vector **w** such that the projection $\mathbf{w}^T\mathbf{x}$ has maximum non-Gaussianity. Non-Gaussianity here is measured by the approximation of negentropy as discussed above. Also, recall that the variance of $\mathbf{w}^T\mathbf{x}$ is constrained to 1, for whitened data, this is equivalent to constraining the norm of **w** to 1.

The FastICA is a fixed-point iterative algorithm for finding a maximum for the non-Gaussianity of $\mathbf{w}^T\mathbf{x}$. It can also be derived using Newton's iterative method. We denote g as the derivative of G, the non-quadratic function mentioned above:

$$g_1(u) = \tanh(a_1 u),$$
$$g_2(u) = u \exp(-u^2/2)$$

Here, $1 <= a_1 <= 2$ (typically 1). The basic form of the FastICA algorithm is as follows:

1. Choose an initial (e.g. random) weight vector **w**.
2. Let $\mathbf{w}^+ = E\{\mathbf{x}g(\mathbf{w}^T\mathbf{x})\} - E\{g'(\mathbf{w}^T\mathbf{x})\}\mathbf{w}$
3. Let $\mathbf{w} = \mathbf{w}^+ / \|\mathbf{w}^+\|$
4. If not converged, go back to 2.

This follows from the maximization of the approximation of negentropy of $\mathbf{w}^T\mathbf{x}$, i.e $E\{G(\mathbf{w}^T\mathbf{x})\}$, which can be further solved using Newton's method.

This algorithm estimates one of the independent components.

FastICA for several units

Now, to estimate several independent components, we need to run the one-unit FastICA algorithm using several units, i.e with weight vectors $\mathbf{w_1}$, $\mathbf{w_2}$... $\mathbf{w_n}$. To prevent different vectors from converging to the same maxima we must decorrelate the outputs $\mathbf{w_1}^T\mathbf{x}$, $\mathbf{w_2}^T\mathbf{x}$, ... $\mathbf{w_n}^T\mathbf{x}$ after every iteration.

We estimate the components one by one. When we have estimated p independent components, or p vectors $\mathbf{w_1}$, $\mathbf{w_2}$ .... $\mathbf{w_n}$, we run the one-unit FastICA to get $\mathbf{w_{p+1}}$, and after each iterative step, we subtract the projections $\mathbf{w_{p+1}}^T\mathbf{w_j}\mathbf{w_j}$ (j= 1..p) of the previous p estimated vectors from $\mathbf{w_{p+1}}$, and then renormalize $\mathbf{w_{p+1}}$.

1. Let $\mathbf{w}_{p+1} = \mathbf{w}_{p+1} - \sum_{j=1}^{p} \mathbf{w}_{p+1}^T \mathbf{w}_j \mathbf{w}_j$
2. Let $\mathbf{w}_{p+1} = \mathbf{w}_{p+1} / \sqrt{\mathbf{w}_{p+1}^T \mathbf{w}_{p+1}}$

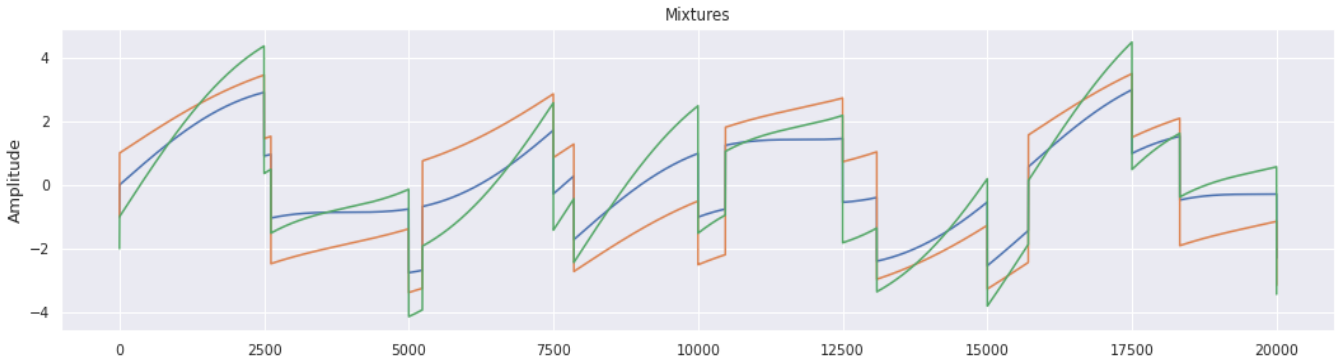
Implementation and Results

ICA on synthetic signals

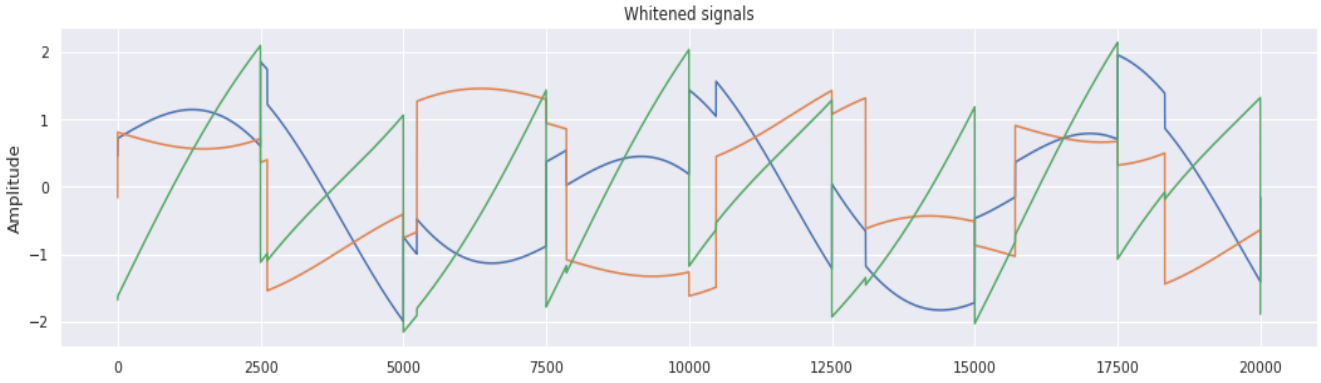
We decided to run FastICA on synthetically generated and mixed signals. With 20000 time steps, we generated 3 signals: A sine wave signal, a sawtooth signal and a square wave.

To generate 3 synthetic mixtures, we used the mixing matrix:

```
A = [[1, 1, 1], [0.5, 2, 1.0], [1.5, 1.0, 2.0]]
```

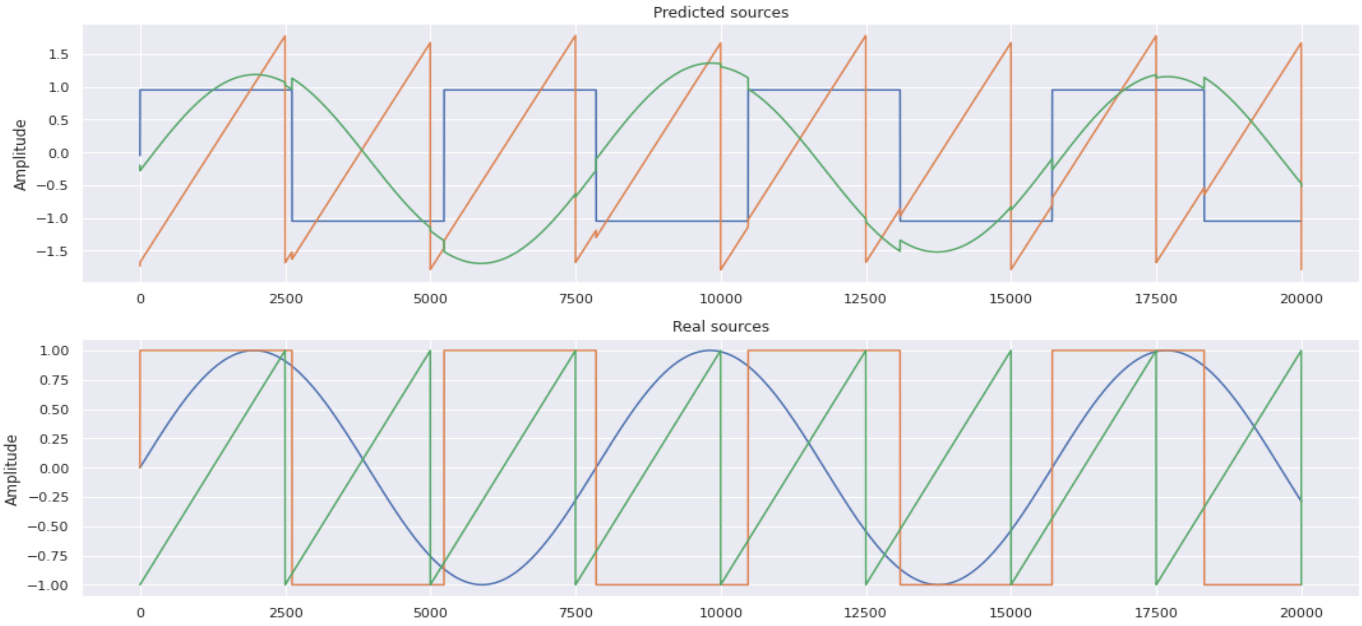


After whitening the signal would look like this:



Thus, only whitening is not sufficient.

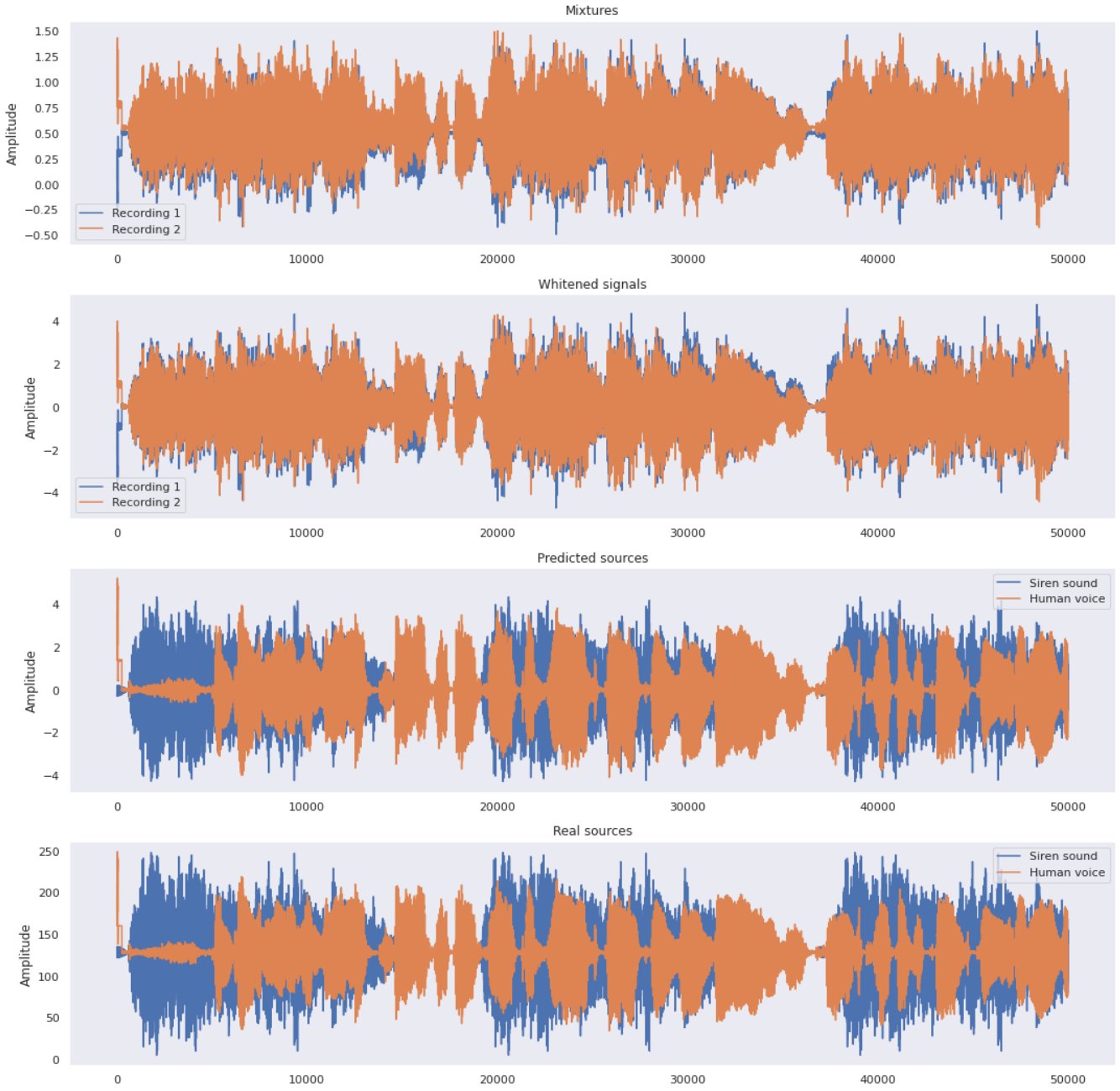
After running the algorithm for 1000 iterations, we get the separated signals:



As can be observed, the algorithm performs reasonably well. This can be tested for several mixing matrices as well. It was also observed that the algorithm performed better when the number of time steps was increased from 10000 to 20000.

ICA on mixed audio signals

We further applied FastICA over 2 mixed audio signals consisting of a person speaking with a shrill siren buzzing in the background.

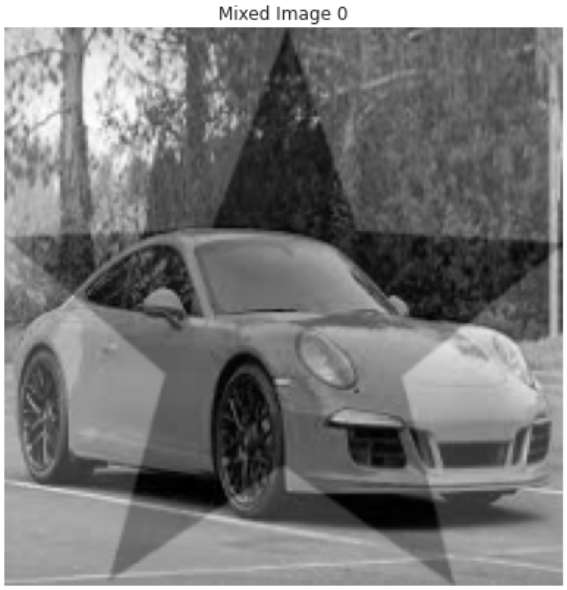


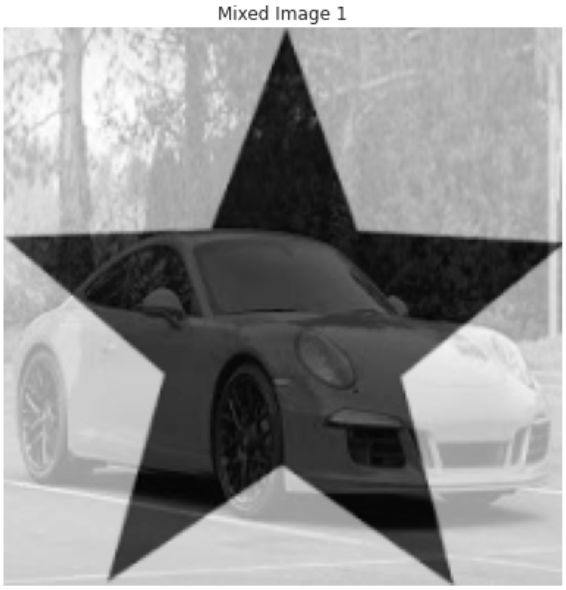
As observed, the predicted source signals are very close to the real sources.

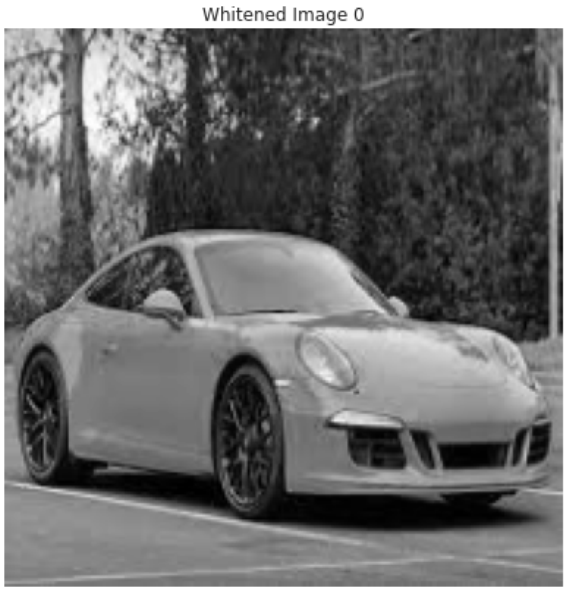
ICA on Images

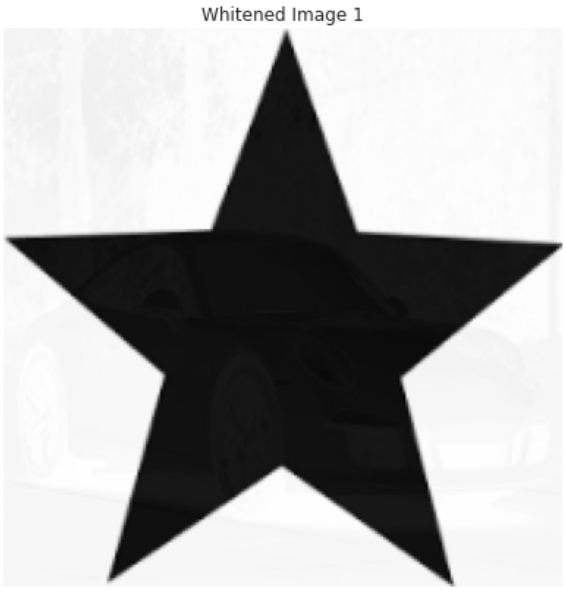
We had two separate grayscale images. We resized these images to bring them to the same size. We then mixed the two images using a mixing matrix to generate two mixed images, which we then tried to separate.

The mixing matrix was: `([[1, 2], [3, 1]])/4`



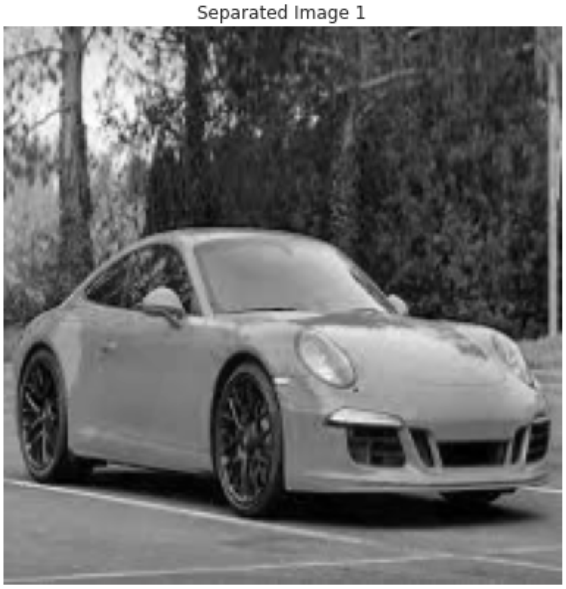






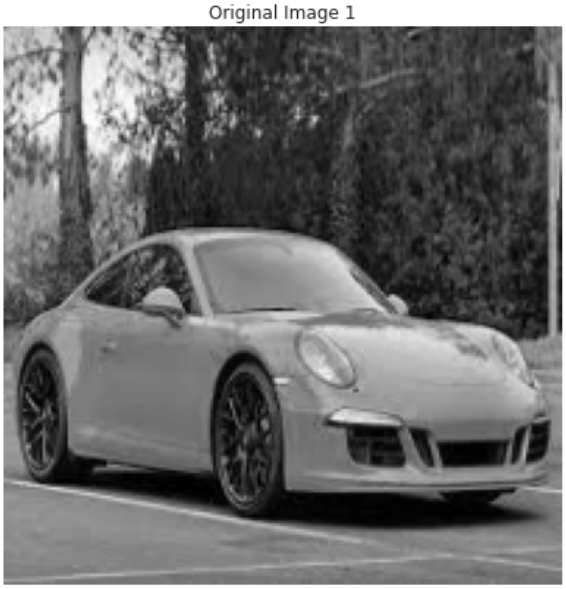
Separated Image 0

Separated Image 1



Original Image 0

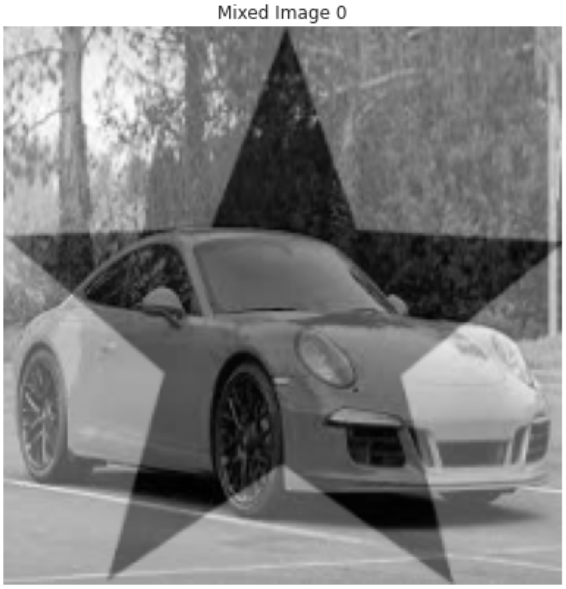
Original Image 1



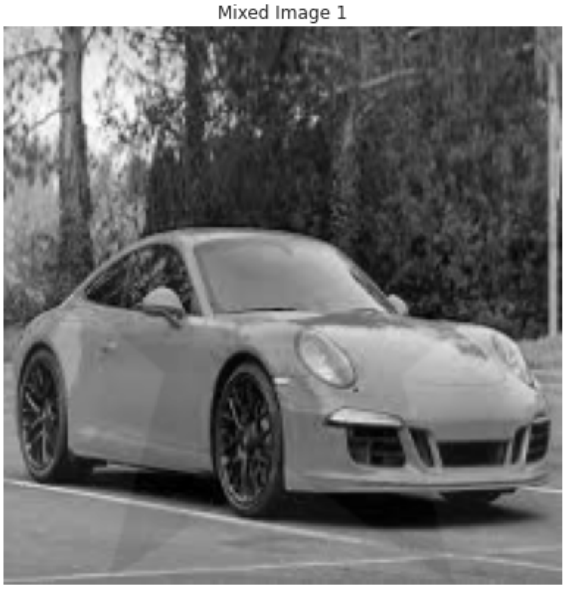
Trying with a different mixing matrix: `[[1, 1.1], [1, 10]]/10.`

This mixing matrix is meant to produce one mixed image which has almost similar ratio of both images, and the other mixed image which has a higher contribution from the 2nd image.

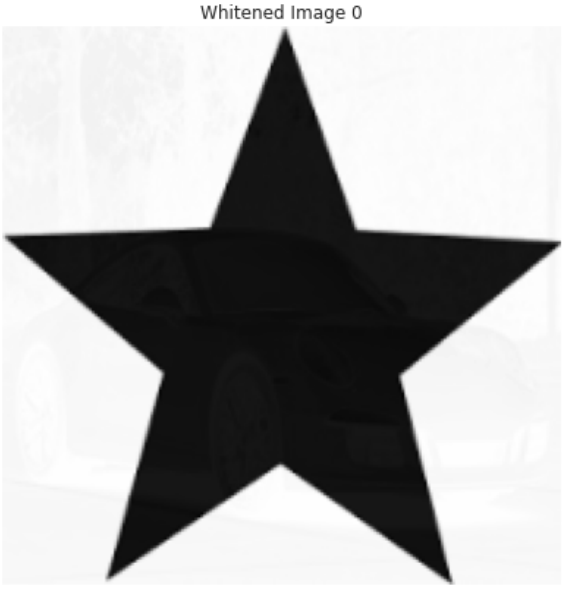
Mixed Image 0



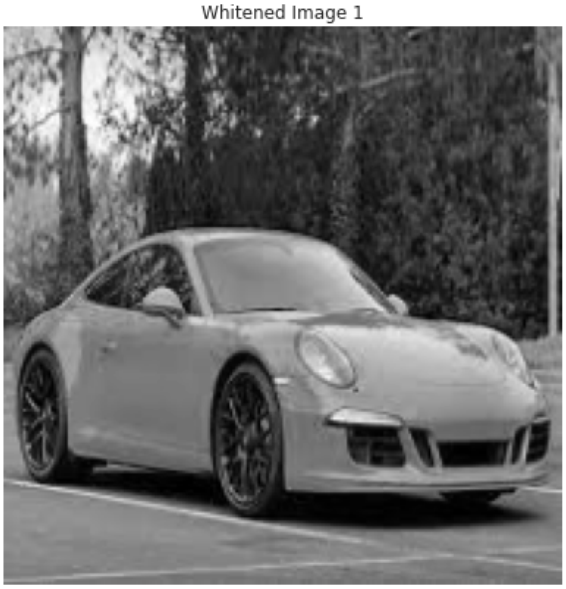
Mixed Image 1



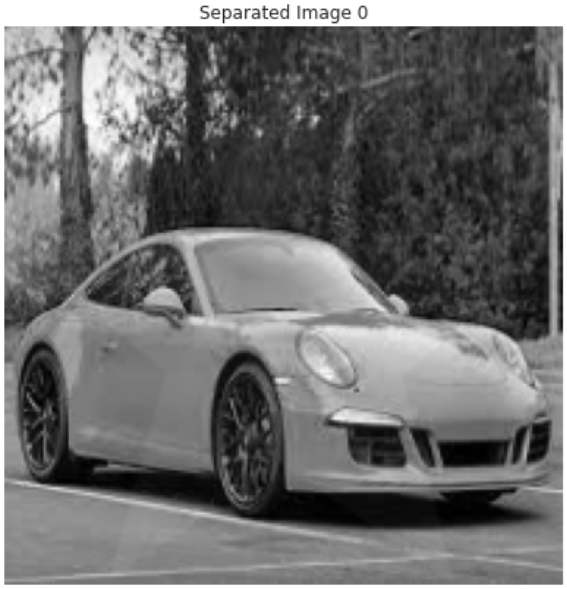
Whitened Image 0



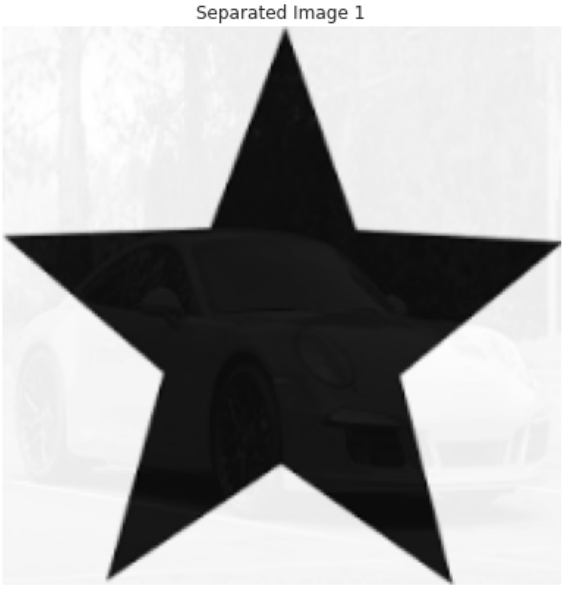
Whitened Image 1



Separated Image 0

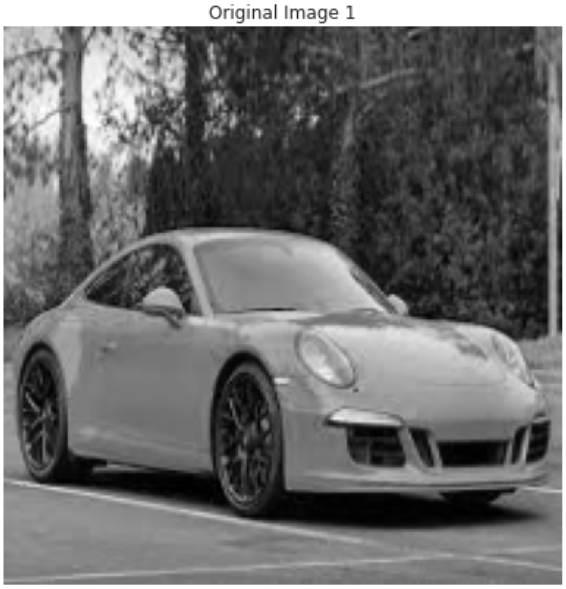


Separated Image 1



Original Image 0

Original Image 1



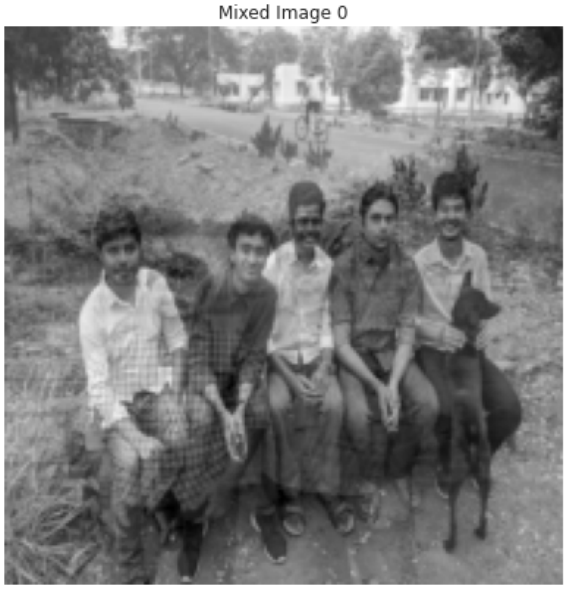
In this case, the separation is slightly less distinct.

We also tried doing this with another set of images.

Mixing matrix: `[[1, 2], [2, 1]]/3`

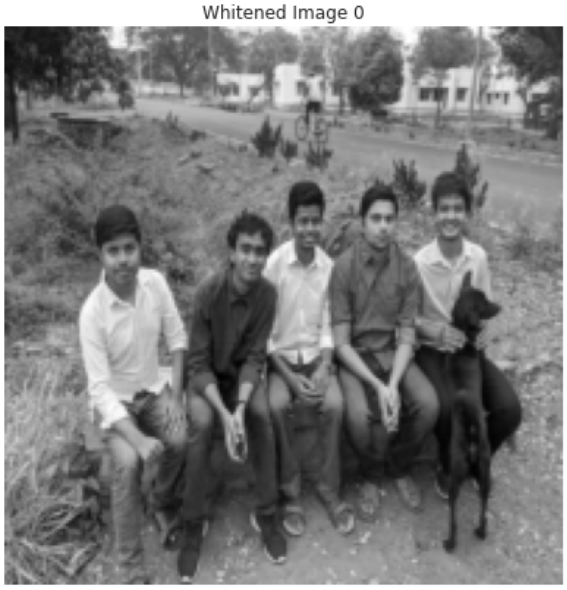
The images were resized to (200, 200)

Mixed Image 0

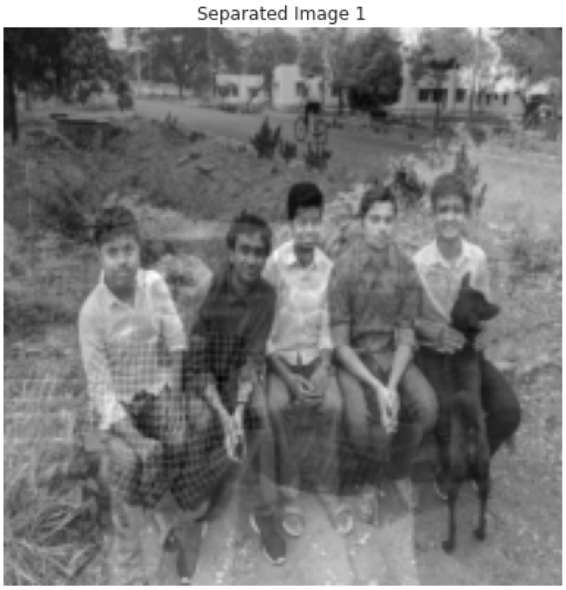


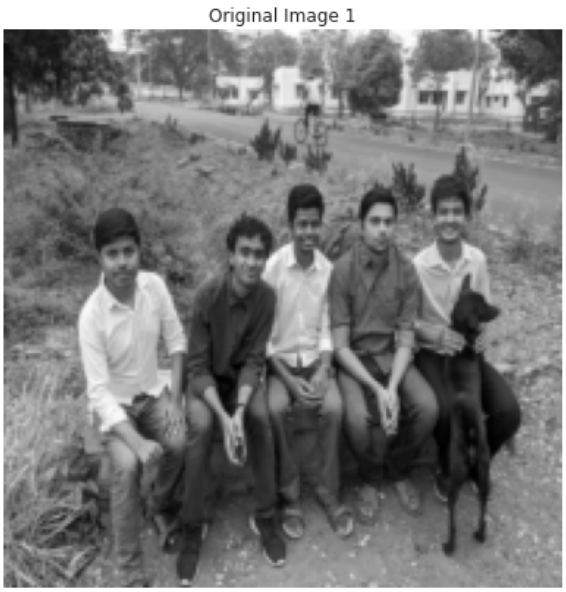
Mixed Image 1

Whitened Image 0



Whitened Image 1

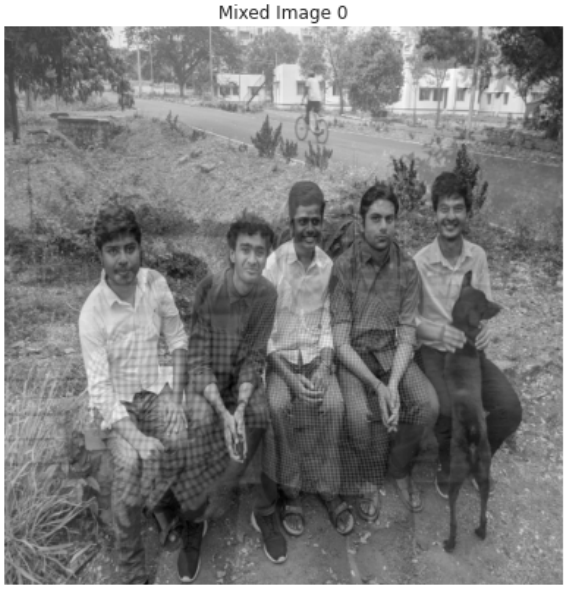




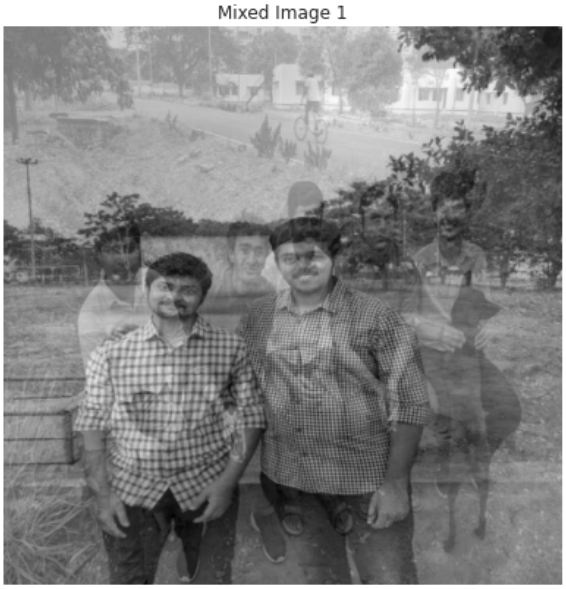
Surprisingly, whitening works well here, but ICA does not give good results.

So, we tried keeping the size of the images to (500, 500).

Mixed Image 0

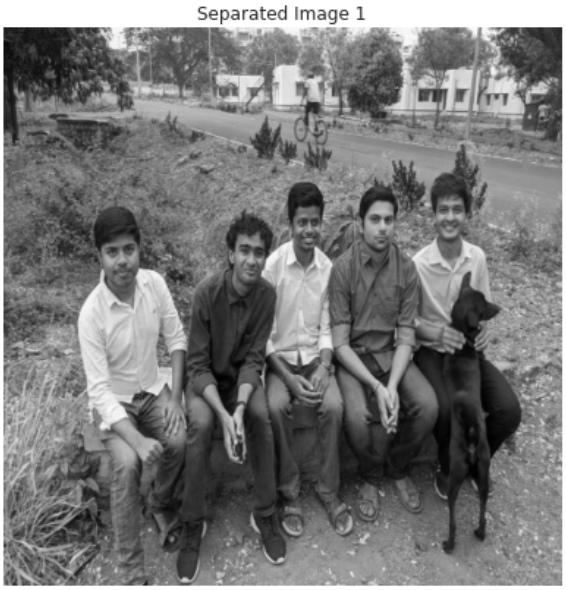


Mixed Image 1



Whitened Image 0

Whitened Image 1



We believe it is performing well because now there are more samples to take the expectation of the approximation non-Gaussianity of $\mathbf{w}^T\mathbf{x}$, i.e E(G($\mathbf{w}^T\mathbf{x}$)), thus leading to better convergence.

Applications of ICA

- Optical imaging of neurons
- Face recognition
- Removing noise from natural images
- Mobile phone communication
- Predicting stock market prices
- Removing artifacts, such as eye blinks from Electroencephalography data
- Studies of the resting state network of the brain

Conclusion

Thus, ICA is a very useful statistical technique that can help us in separating independent components from mixed signals. FastICA is a method of performing ICA, which is computationally very efficient and reasonably effective, hence the name. It has widespread applications in different fields such as audio processing, biomedical signal processing, image processing, telecommunications, and econometrics.

Bibliography/References

- FastICA paper: https://www.cs.helsinki.fi/u/ahyvarin/papers/NN00new.pdf
- https://en.wikipedia.org/wiki/Independent_component_analysis
- https://towardsdatascience.com/independent-component-analysis-ica-in-python-a0ef0db0955e
- https://doi.org/10.1016/j.aci.2018.08.006

Kernel PCA

Direct PCA

- Assumes that when we encounter D dimensional data, the points lie mainly in a linear subspace with dimension < D
- Tries to find a new *orthogonal* ordered set of axes such that the data has the highest variation along the first axis, the second-highest variation along the second axis and so on.
- This ensures that taking the first d' (<= d) axes captures the maximum variation in the data, giving the least reconstruction error.

Mathematical formulation

Given data points as rows, with features as columns, we can write the feature-feature covariance as follows. This is assuming that the data is mean-normalized across all of its features.

$$X = \begin{bmatrix} -x_1^T- \\ -x_2^T- \\ \cdots \\ -x_n^T- \end{bmatrix} \qquad Cov(X) = X^T X$$

$$Cov(XA) = A^T X^T X A$$

If we do a linear transformation of the data with a matrix A, then we would arrive at the equation above.

If we use the right singular vectors V of the matrix X for transformation, we would have diagonalized the covariance of the data in the transformed space.

$$X = U\Sigma V^T$$

$$Cov(XV) = V^T X^T X V = \Sigma^2$$

Transforming the data using first d' eigenvectors yields the feature variances as the first d' squared eigenvalues.

Project training data	Project unseen test data
$Z = XV = U\Sigma$	$Z = YV$

Dual PCA

This method is preferred when the dimension of the data is very high compared to the number of examples. The time complexity of SVD is $O(n^2d + d^3)$.

If the number of dimensions is very large compared to the number of training examples, (image/text data) then SVD on the data matrix might not be a good idea. The following algorithm mitigates the same by doing eigenvalue decomposition on the matrix XX^T. Since the matrix V is not known, we express it in terms of U and **Σ**. Note that U is orthogonal while **Σ** is diagonal.

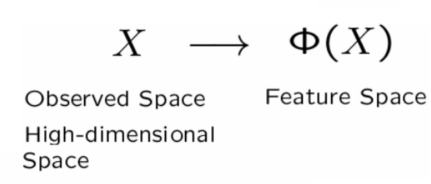
$$XX^T = U\Sigma^2U^T$$

Project training data	Project unseen test data
$Z = XV = U\Sigma$	$Z = YV = YX^TU\Sigma^{-1}$

Here, U is *(n x d')* orthogonal matrix while **Σ** is *(d' x d')* diagonal matrix.

Note that both parts of the algorithm - computing EVD and projecting unseen data is expressed as inner products of the data points. This means that it is sufficient for us to define a space in terms of the inner products of the vectors in that space. This gives motivation for the kernelization of this algorithm.

Kernel PCA



We can use a function **ϕ** to transform the given data to a higher-dimensional space, so that the data may have better interpretability. We can then apply PCA on this transformed data to obtain the first few components of interest.

The Dual PCA's formulation implies that we do not need to explicitly define the function **ϕ**. We only need the closed form for computing the dot products of the vectors in the features space. We call this the kernel function. An example is shown below. The rest of the math will be similar to the dual PCA algorithm except that the data **X** will be replaced by its transformation **ϕ(X)**.

$$\phi_1([x_1, x_2]^T) = [1, x_1, x_2, \sqrt{2}x_1x_2]$$
$$K_1(x, y) = \phi_1(x)^T \phi_1(y) = (x^T y)^2$$

Common Kernels:

- *Linear* $K_{ij} = \langle X_i, X_j \rangle$
- *Polynomial* $K_{ij} = \left(1 + \langle X_i, X_j \rangle\right)^p$
- *Gaussian* $K_{ij} = e^{\frac{-\|X_i - X_j\|^2}{2\sigma^2}}$

$$\Phi(X)\Phi(X)^T = kernel(X, X) = U\Sigma^2 U^T$$

Project training data	Project unseen test data
$Z = \Phi(X)V = U\Sigma$	$Z = \Phi(Y)V = k(Y, X)U\Sigma^{-1}$

Here, X and Y are matrices holding the data points (vectors) as rows.

We can appreciate why a gaussian kernel effectively maps to an infinite-dimensional space with the following proof.

4.3.3 The RBF Kernel

The function $k(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2/2\sigma^2}$ known as a *Radial Basis Function* (RBF) is a kernel function but with an infinite expansion. Without loss of generality let $\sigma = 1$, then we have:

$$
\begin{aligned}
e^{-\|\mathbf{x}-\mathbf{y}\|^2/2} &= e^{-\|\mathbf{x}\|^2/2} e^{-\|\mathbf{y}\|^2/2} e^{\mathbf{x}^\top \mathbf{y}} \\
&= \sum_{j=0}^{\infty} \frac{(\mathbf{x}^\top \mathbf{y})^j}{j!} e^{-\|\mathbf{x}\|^2/2} e^{-\|\mathbf{y}\|^2/2} \\
&= \sum_{j=0}^{\infty} \left(\frac{e^{-\frac{\|\mathbf{x}\|^2}{2j}}}{\sqrt{j!}^{1/j}} \frac{e^{-\frac{\|\mathbf{y}\|^2}{2j}}}{\sqrt{j!}^{1/j}} \mathbf{x}^\top \mathbf{y} \right)^j \\
&= \sum_{j=0}^{\infty} \sum_{\sum_i n_i = j} \frac{e^{-\frac{\|\mathbf{x}\|^2}{2j}}}{\sqrt{j!}^{1/j}} \binom{j}{n_1, \ldots, n_k}^{1/2} x_1^{n_1} \cdots x_k^{n_k} \frac{e^{-\frac{\|\mathbf{y}\|^2}{2j}}}{\sqrt{j!}^{1/j}} \binom{j}{n_1, \ldots, n_k}^{1/2} y_1^{n_1} \cdots y_k^{n_k}
\end{aligned}
$$

From which we can see that the entries of the feature map $\phi(\mathbf{x})$ are:

$$
\phi(\mathbf{x}) = \left(\frac{e^{-\frac{\|\mathbf{x}\|^2}{2j}}}{\sqrt{j!}^{1/j}} \binom{j}{n_1, \ldots, n_k}^{1/2} x_1^{n_1} \cdots x_k^{n_k} \right)_{j=0,..,\infty, \sum_{i=1}^{k} n_i = j}
$$

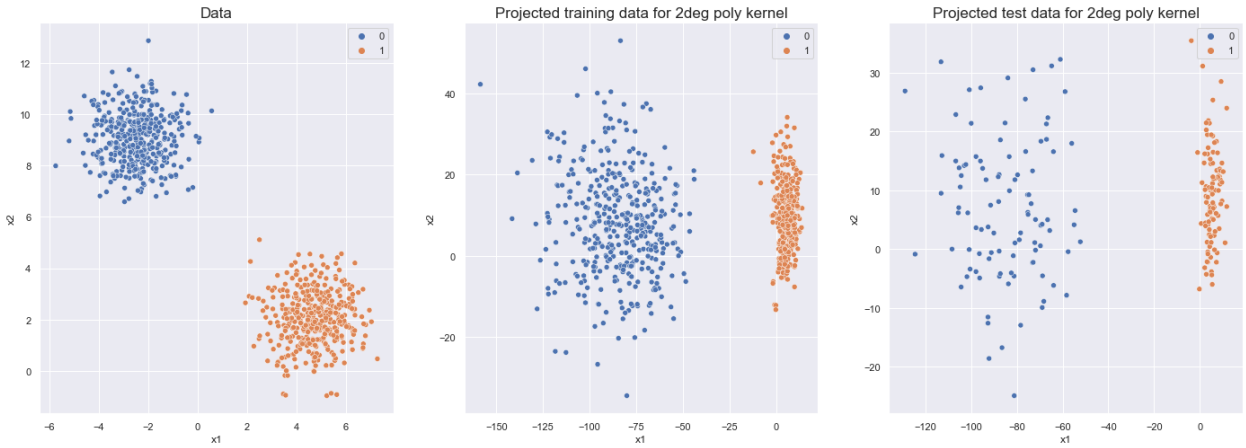
Performance on synthetic data

Blobs

No kernel

Without any kernel, Kernel PCA is just direct PCA. We can clearly see the first component is along the direction of the difference of blob means and the second axis is perpendicular to the first axis.

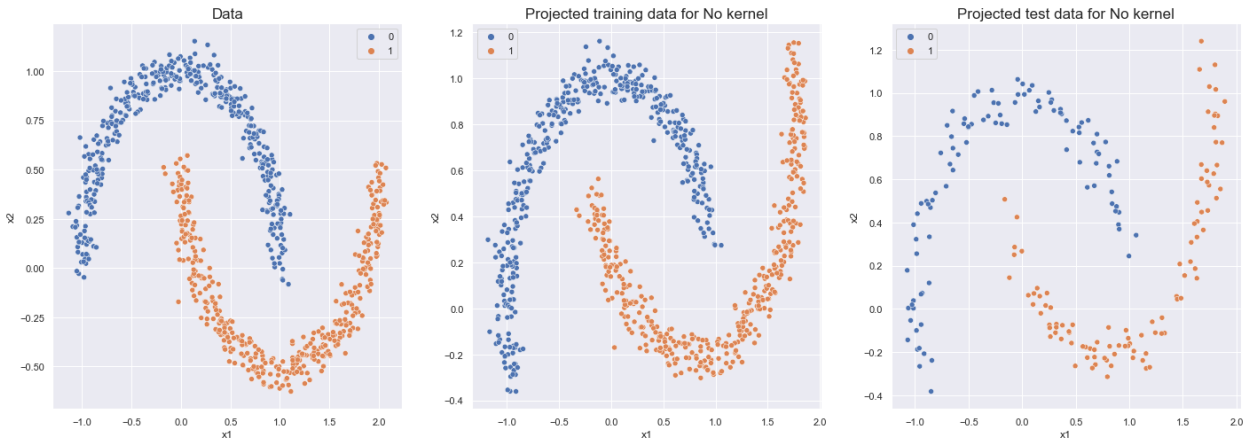
Quadratic kernel



Using the same data as input to a degree 2 polynomial kernel PCA, we can see that the blob closer to the origin is condensed to a smaller place while the other blob is spread out. Because of using the square of the features as the components, the within-class variation will be more for the blob further from the origin.

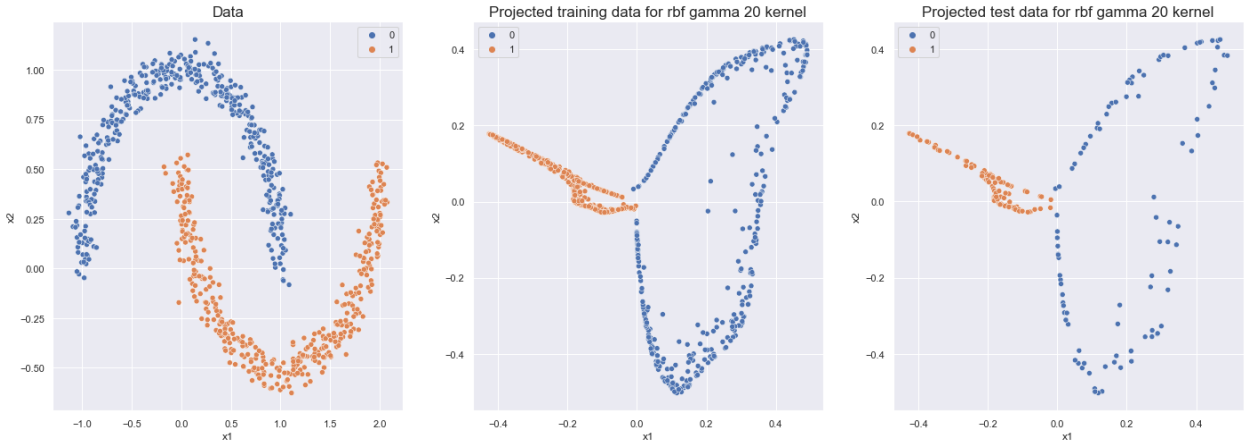
Moons

No kernel



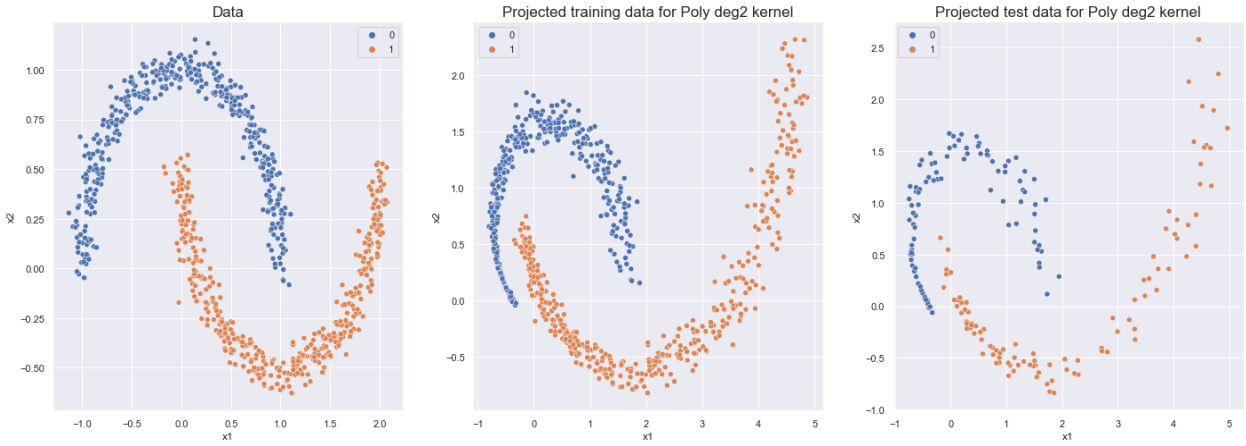
The moons dataset is not linearly separable. Any sort of rotation/scaling without a kernel will not change this

Gaussian Kernel



The Gaussian kernel with a large value of gamma does a good job of separating the data. A high value of gamma means that in the transformed space, points further away from each other will lie in a perpendicular plane. Since we have taken only the first two components for plotting, we can see that the orange dots lie in kind of a horizontal (x2 x3)plane and the blue values lie in the (x1x2) plane.

Quadratic kernel

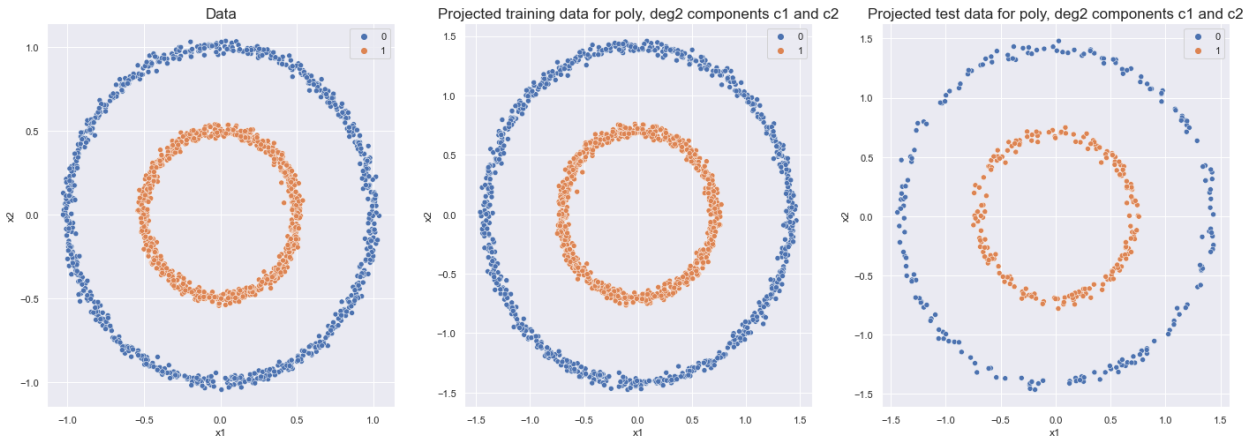


We can see the cluster closer to the origin(blue) has shrunk, whereas the orange cluster has been spread out in the kernel space.

Circles

No kernel: Radially symmetric, every orthogonal basis will give the same plot.

Quadratic Kernel



This was a surprise at first glance. Why would a quadratic kernel not separate the circle's data? After close inspection, we can see that the outer rim of the data lies approximately on a unit circle. This means that the variance in the higher-order terms will be less than the original terms. Note that the quadratic kernel has the transformation function phi([x,y]) = $(1,\ 1.4x,\ 1.4y,\ 1.4xy,\ x^2,\ y^2)$

The components to be looked at are x^2 and y^2. When we plot the fourth and fifth components of the decomposition, we get the desired results as shown above.

	ncomponents	test_acc	train_acc
0	1.0	0.4800	0.501875
1	2.0	0.4625	0.510000
2	3.0	0.4475	0.478750
3	4.0	0.4675	0.501250
4	5.0	1.0000	1.000000
5	6.0	1.0000	1.000000
6	7.0	1.0000	1.000000

Running logistic regression on the circles dataset also shows the expected results - fifty/fifty accuracy until both the fourth and fifth components were included.

Performance on real-world datasets

Experimental Setup

For analysing the performance of the decomposition, we use a K-Nearest-Neighbour model with k=1 to predict the class of a test example. Since we are not estimating the optimal hyperparameters here, we split the data into a training set for KNN density estimation and a test set for reporting performance. We used 20% of the data for testing.

Olivetti faces

The dataset has 10 images each of 40 people, photographed at different times of the day with varying lighting, illumination and facial expressions. The image size is 64 *x* 64, giving us 4096 dimensions to start with.

No kernel

Since we do not supply the class information before PCA decomposition, the dimensions across which maximum variation is observed may not be due to the faces being different, but due to the variance of other factors - illumination, expression, a subject wearing glasses. Thus the first few dimensions, need not in general, completely capture the between-class variation

	dims	train_acc	test_acc
0	10	1.0	0.8625
1	25	1.0	0.9250
2	50	1.0	0.8750
3	100	1.0	0.8500
4	300	1.0	0.8125

among the training examples. Including more components causes the k-nearest neighbour model to overfit to the training data. However, it is worth noting that the training images indeed clustered well in the initial subspace itself.

Polynomial Kernel

Quadratic Kernel

	dims	train_acc	test_acc
0	10	1.0	0.8875
1	25	1.0	0.9375
2	50	1.0	0.8875
3	100	1.0	0.8250
4	300	1.0	0.8000

Cubic Kernel

	dims	train_acc	test_acc
0	10	1.0	0.825
1	25	1.0	0.950
2	50	1.0	0.850
3	100	1.0	0.800
4	300	1.0	0.800

Degree 4 kernel

	dims	train_acc	test_acc
0	10	1.0	0.8625
1	25	1.0	0.9375
2	50	1.0	0.8500
3	100	1.0	0.7500
4	300	1.0	0.7750

We notice a similar trend here, but the superior performance can be attributed to the features in their respective space having higher variance are indeed the discriminatory features necessary for good classification performance.

Gaussian Kernel

Gamma = 5

	dims	train_acc	test_acc
0	10	1.0	0.025
1	25	1.0	0.025
2	50	1.0	0.025
3	100	1.0	0.025
4	300	1.0	0.025

Gamma = 0.001

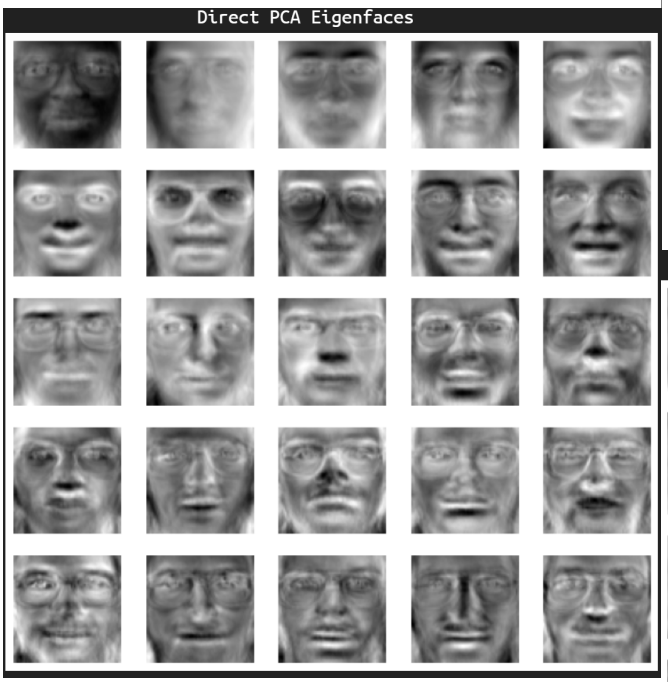
	dims	train_acc	test_acc
0	10	1.0	0.8625
1	25	1.0	0.9375
2	50	1.0	0.8750
3	100	1.0	0.8750
4	300	1.0	0.7500

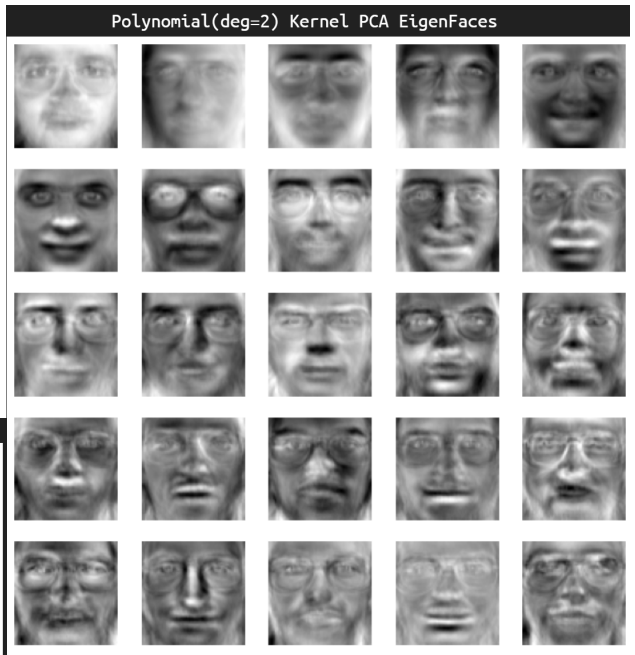
The gamma value refers to how fast the dot product between examples decays with their Euclidean distance. Looking at the low test accuracies, we can conclude that the data is very sparsely distributed in the transformed space - ie, the

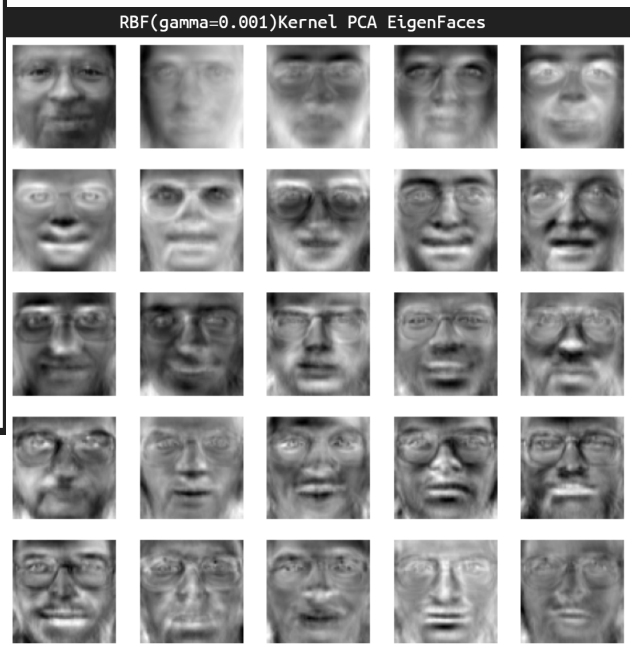
effective density estimated by the KNN would have contours of small areas. Thus increasing the value of gamma improves the performance of the KNN model.

EigenFaces

We supply the identity matrix as input to the kernel PCA model to arrive at the component vectors (eigenfaces) used as an orthogonal basis to project the data into the transformed feature space. When we do not use the kernel, these refer to the vanilla eigenfaces.







Digits Dataset

No kernel

	dims	train_acc	test_acc
0	5	1.0	0.900000
1	10	1.0	0.977778
2	25	1.0	0.972222
3	40	1.0	0.975000
4	64	1.0	0.938889

Polynomial kernel

Quadratic Kernel

	dims	train_acc	test_acc
0	5	1.0	0.855556
1	10	1.0	0.963889
2	25	1.0	0.983333
3	40	1.0	0.980556
4	64	1.0	0.983333

Cubic Kernel

	dims	train_acc	test_acc
0	5	1.0	0.863889
1	10	1.0	0.961111
2	25	1.0	0.972222
3	40	1.0	0.983333
4	64	1.0	0.988889

Degree 4 kernel

	dims	train_acc	test_acc
0	5	1.0	0.847222
1	10	1.0	0.963889
2	25	1.0	0.966667
3	40	1.0	0.972222
4	64	1.0	0.969444

Gaussian Kernel

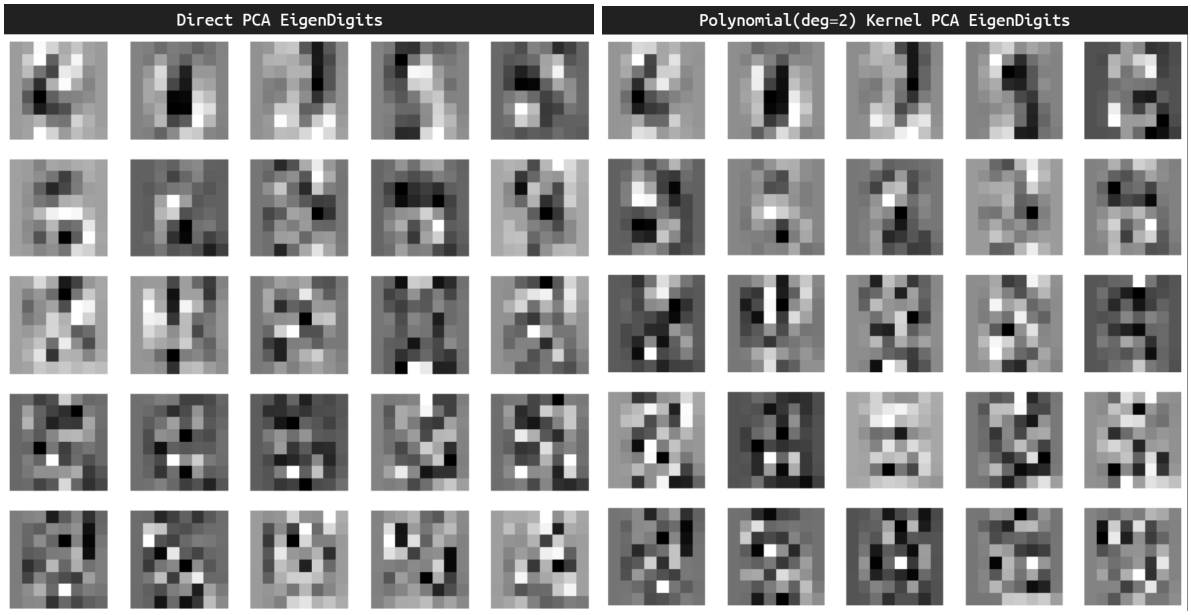
Gamma = 5

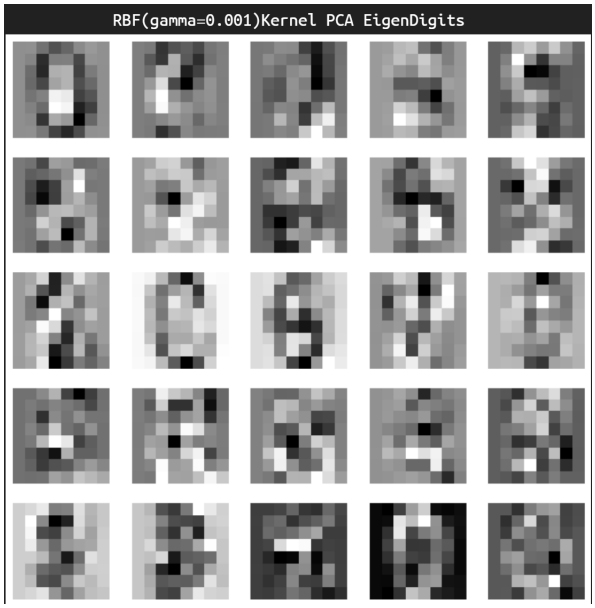
	dims	train_acc	test_acc
0	5	1.0	0.1
1	10	1.0	0.1
2	25	1.0	0.1
3	40	1.0	0.1
4	64	1.0	0.1

Gamma = 0.001

	dims	train_acc	test_acc
0	5	1.0	0.863889
1	10	1.0	0.958333
2	25	1.0	0.977778
3	40	1.0	0.986111
4	64	1.0	0.986111

EigenDigits





References

1. AliGhodsi's course: http://www.math.uwaterloo.ca/~aghodsib/courses/f10stat946/
2. Introduction to Machine learning https://arxiv.org/pdf/0904.3664v1.pdf
3. Kernel tricks and nonlinear dimensionality reduction via RBF kernel PCA https://sebastianraschka.com/Articles/2014_kernel_pca.html

Code at
https://github.com/ameyvadnere/CS418-Statistical-Pattern-Recognition-Lab