Independent Component Analysis & Kernel PCA

Ameya Vadnere and Deepak HR
170010002 and 170010026

Kernel PCA

Algorithm 1 - 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.

$$X = \begin{bmatrix} -x_1^T - \\ -x_2^T - \\ \dots \\ -x_n^T - \end{bmatrix}$$

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

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

Algorithm 1 - Direct PCA

- Assume features of the data are mean normalized, for the covariance formula shown to hold.
- If we use the matrix of transformation as V, then we would have diagonalized the covariance matrix.
- V refers to the right singular vectors of the data matrix X. The corresponding singular values squared correspond to the variance of the data measured along the singular vector.
- To reduce dimensionality while retaining maximum data variance, we use the first d' singular vectors having the largest absolute singular value.

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

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

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

Project train data	Project test data
$Z = XV = U\Sigma$	$Z = YV$

Algorithm 2 - Dual PCA

- SVD time complexity is $O(n^2d + d^3)$
- If number of dimensions is very large (text/image data), it is time consuming to run SVD on the data matrix X.
- Can express V in terms of U, **Σ** and X, by decomposing eigenvectors of XX^T
- For projecting train/test data we truncate consider vectors from U corresponding to the first d' largest squared singular values from **Σ**.
- Both of the steps eigenvalue decomposition and test data projection use only example dot products, paving way for the kernelization of this algorithm.

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

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

Project train data	Project test data
$Z = XV = U\Sigma$	$Z = YV = YX^T U\Sigma^{-1}$

Algorithm 3 - Kernel PCA

- Use a function **ϕ** to transform the given data to a higher-dimensional space, so that the data may have better interpretability.
- Apply PCA on this transformed data to obtain the first few components of interest.
- The Dual PCA's formula allows to not define the transformation function **ϕ**. Defining the inner products of data points in the transformed space is enough.

$$X \longrightarrow \Phi(X)$$

Observed Space — Feature Space

High-dimensional Space

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

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* $K_{ij} = e^{\frac{-\|X_i - X_j\|^2}{2\sigma^2}}$

Algorithm 3 - Kernel PCA

- Need to adjust kernel matrix such that transformed features have zero mean. Here, $I_{1/n}$ denotes a diagonal matrix with value 1/n.
- Remaining math is follows from Dual PCA algorithm
- Decompose eigenvectors and eigenvalues of the kernel matrix K.

$$\tilde{\phi}(x_k) = \phi(x_i) - \frac{1}{n}\sum_{k=1}^{n}\phi(x_k)$$

$$\tilde{K}(x_i, x_j) = \tilde{\phi}(x_i)^T \tilde{\phi}(x_j)$$

$$= \left(\phi(x_i) - \frac{1}{n}\sum_{k=1}^{n}\phi(x_k)\right)^T \left(\phi(x_j) - \frac{1}{n}\sum_{k=1}^{n}\phi(x_k)\right)$$

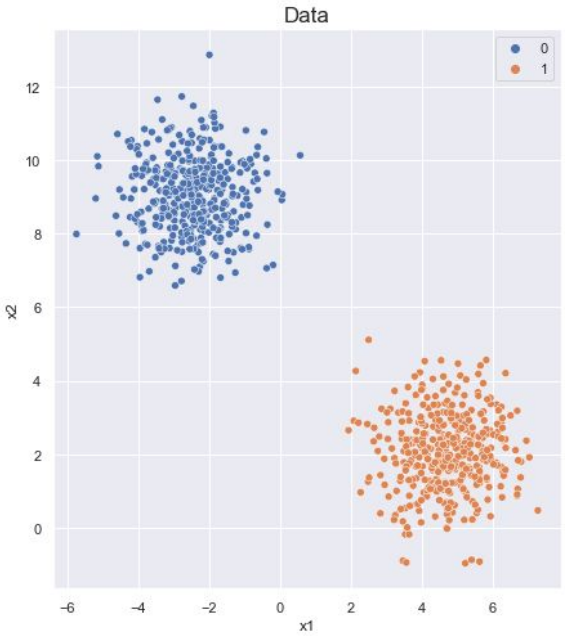
$$= K(x_i, x_j) - \frac{1}{n}\sum_{k=1}^{n}K(x_i, x_k) - \frac{1}{n}\sum_{k=1}^{n}K(x_j, x_k) + \frac{1}{n^2}\sum_{l,k=1}^{n}K(x_l, x_k)$$

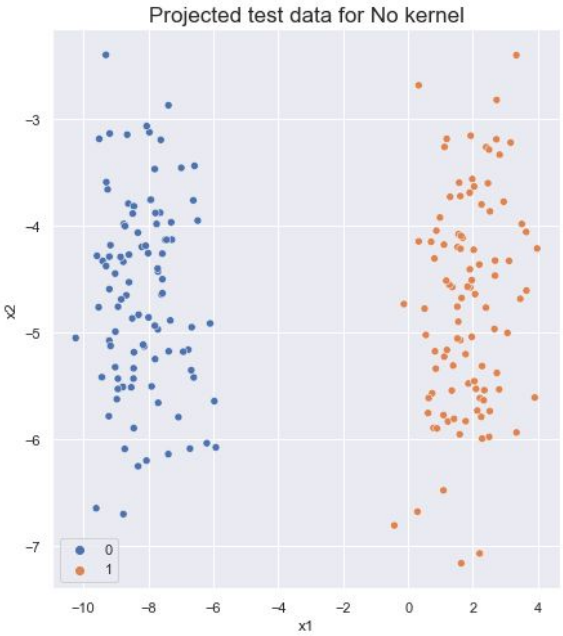
$$\tilde{\mathrm{K}} = \mathrm{K} - \mathbf{2}_{1/n}\mathrm{K} + \mathbf{1}_{1/n}\mathrm{K}\mathbf{1}_{1/n}$$

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

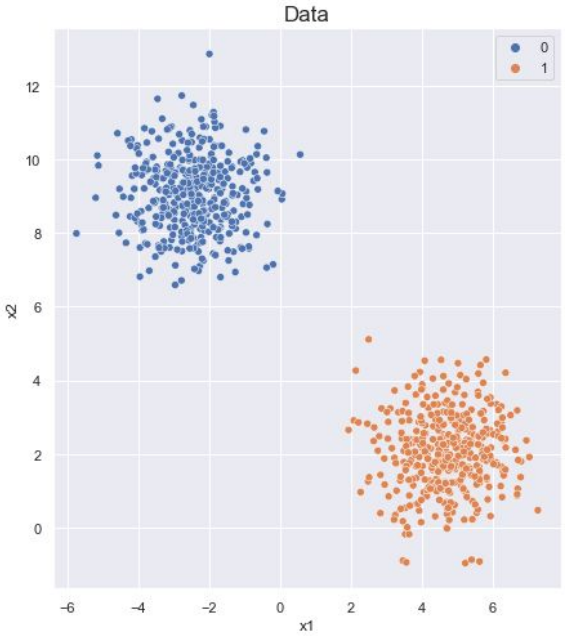
Performance on Synthetic datasets

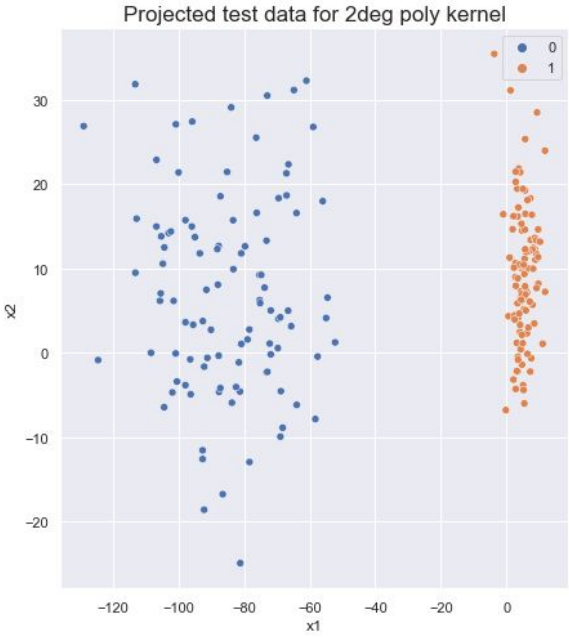
Blobs data - No Kernel



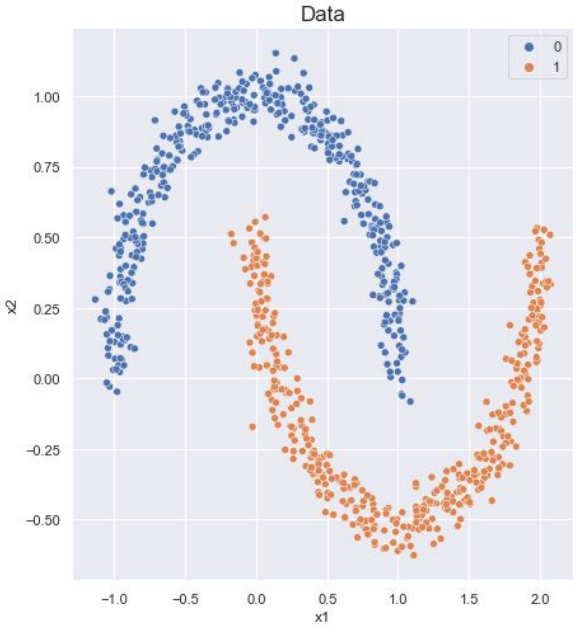


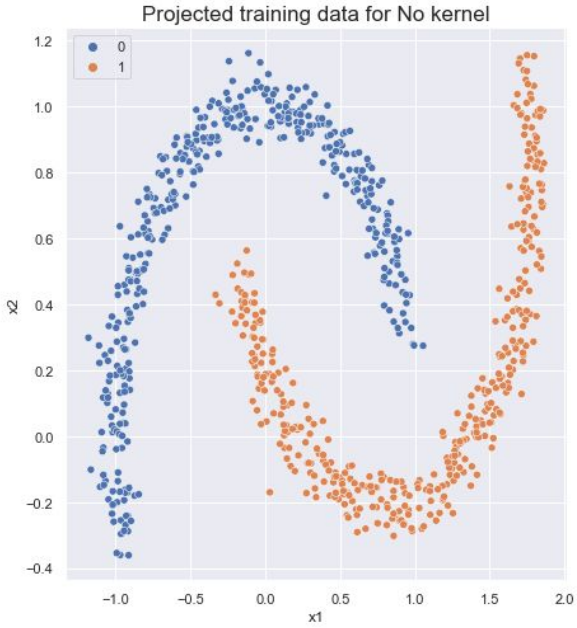
Blobs data - Quadratic Kernel

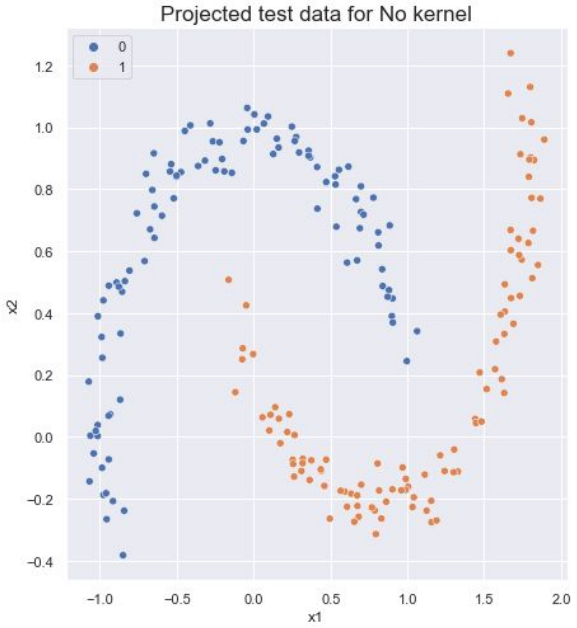




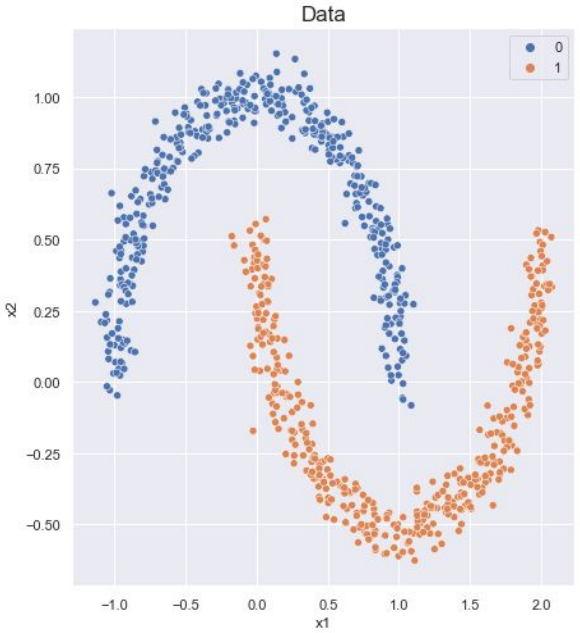
Moons data - No Kernel

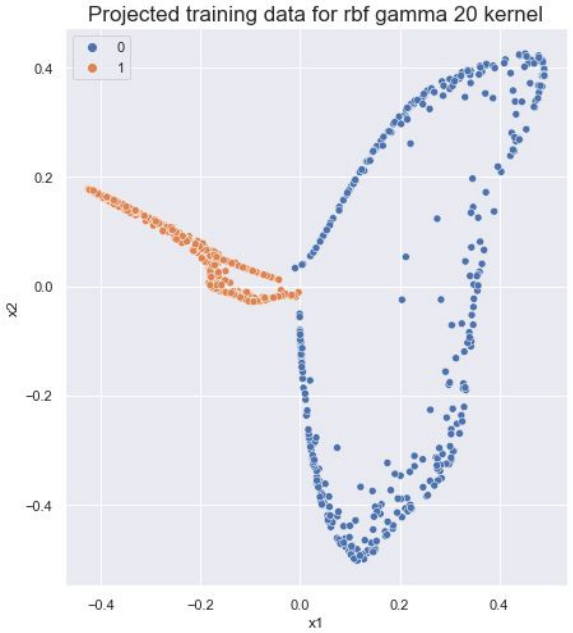


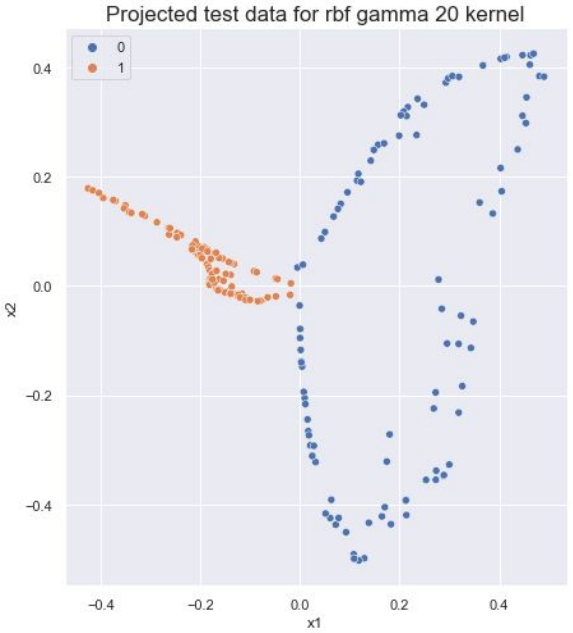




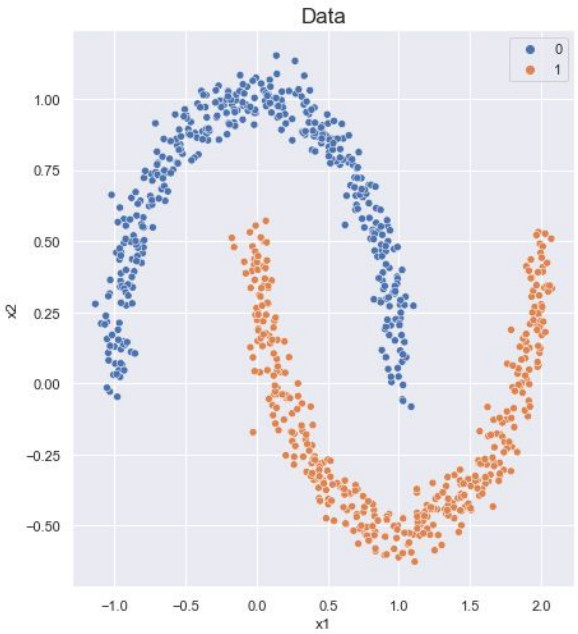
Moons data - Gaussian Kernel

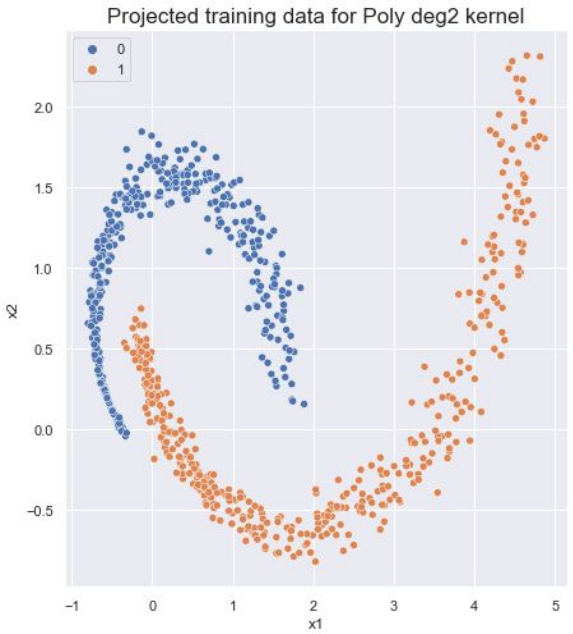


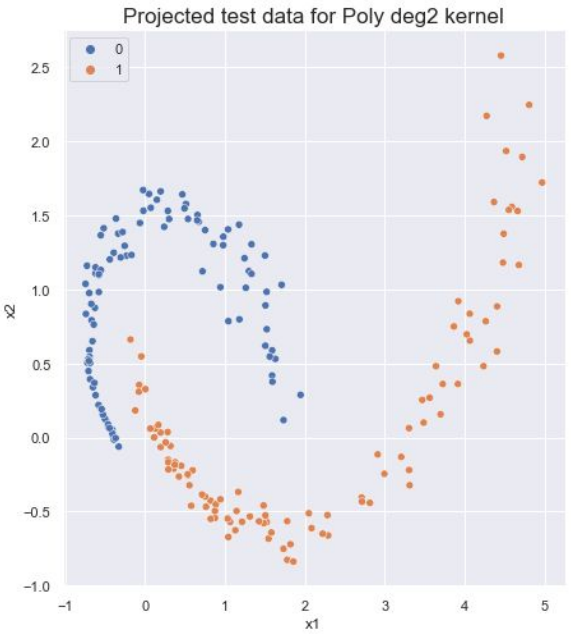




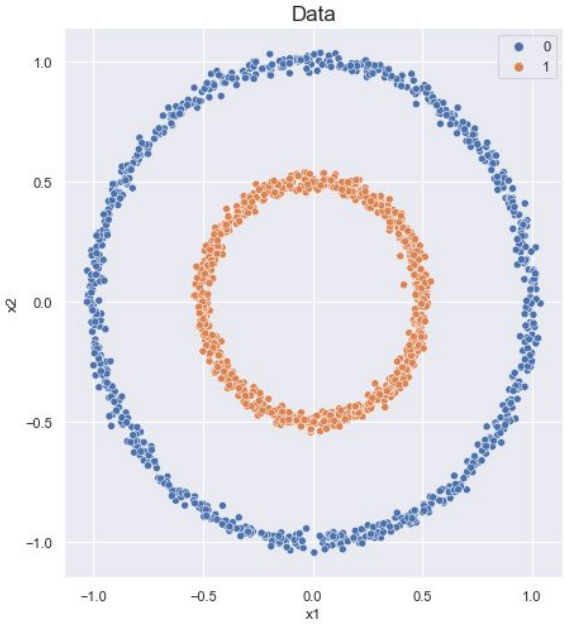
Moons data - Quadratic Kernel

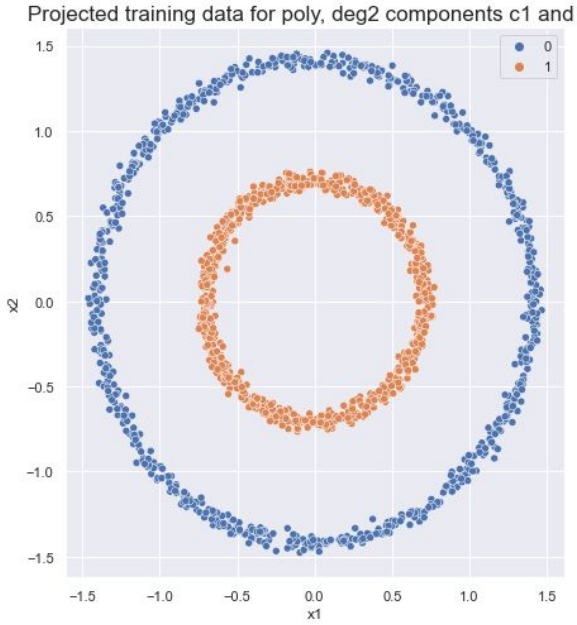


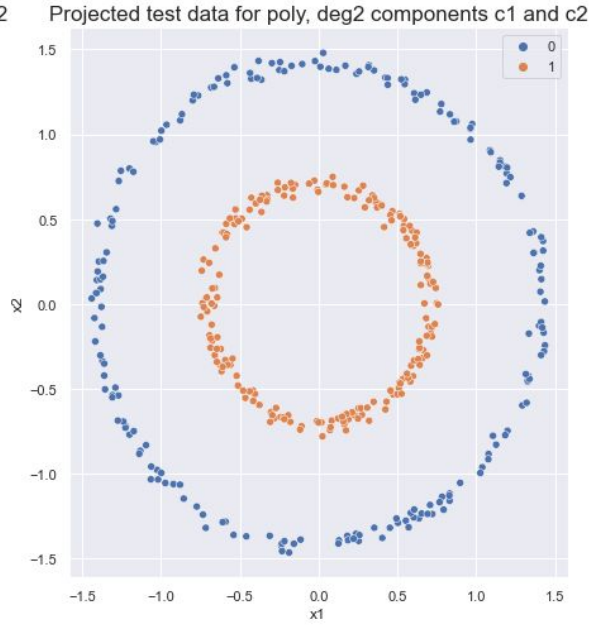




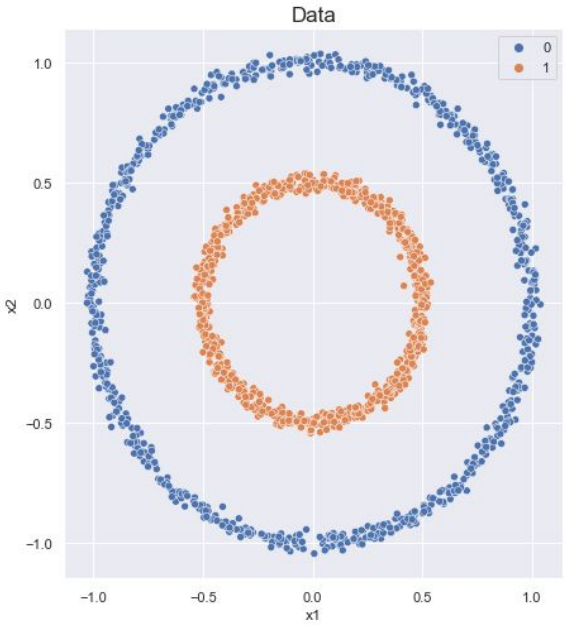
Circles data - Quadratic Kernel (c1, c2)

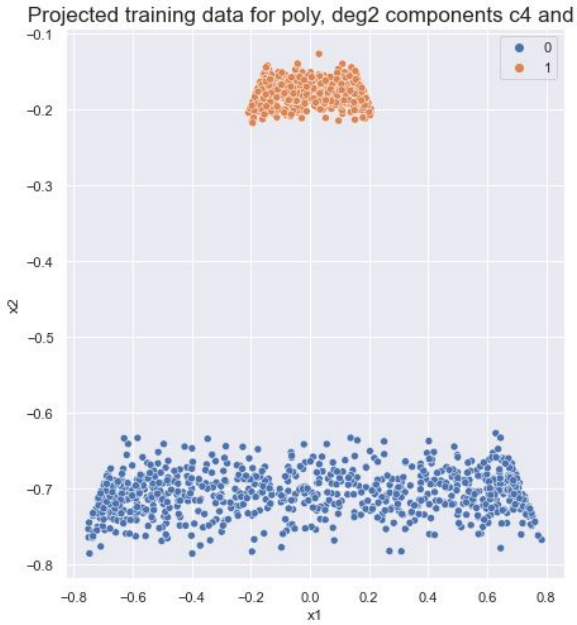


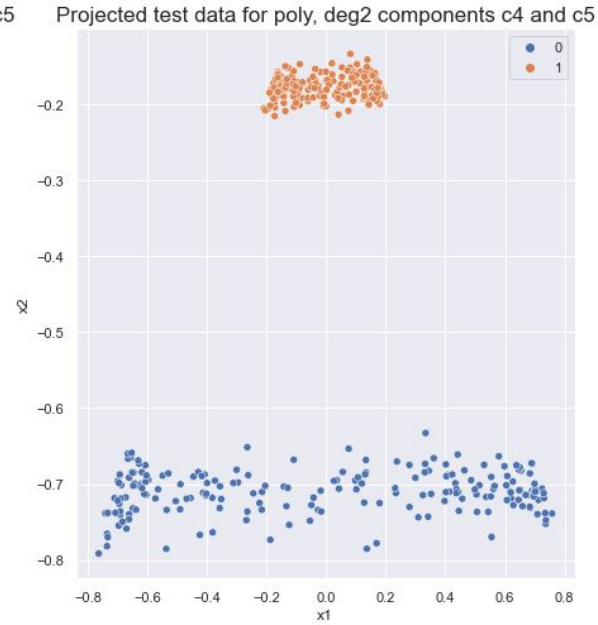




Circles data - Quadratic Kernel (c4, c5)







Logistic regression using different components on circles dataset

- Fifty/fifty accuracy till both fourth and fifth components are included.
- Variation in data versus variation in features that discriminate the class

	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

Performance on Real datasets

Olivetti faces

- The dataset has 10 images each of 40 people
- Images captured 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.

Olivetti faces - No kernel

- Dimensions across which maximum variation need to be due to the faces being different, but due to the variance of other factors - illumination, expression, a subject wearing glasses.
- First few dimensions, need not in general, completely capture the between-class variation
- More components cause KNN to overfit.
- Important to note zero training error.

	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

Olivetti faces - 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

Olivetti faces - Gaussian kernel

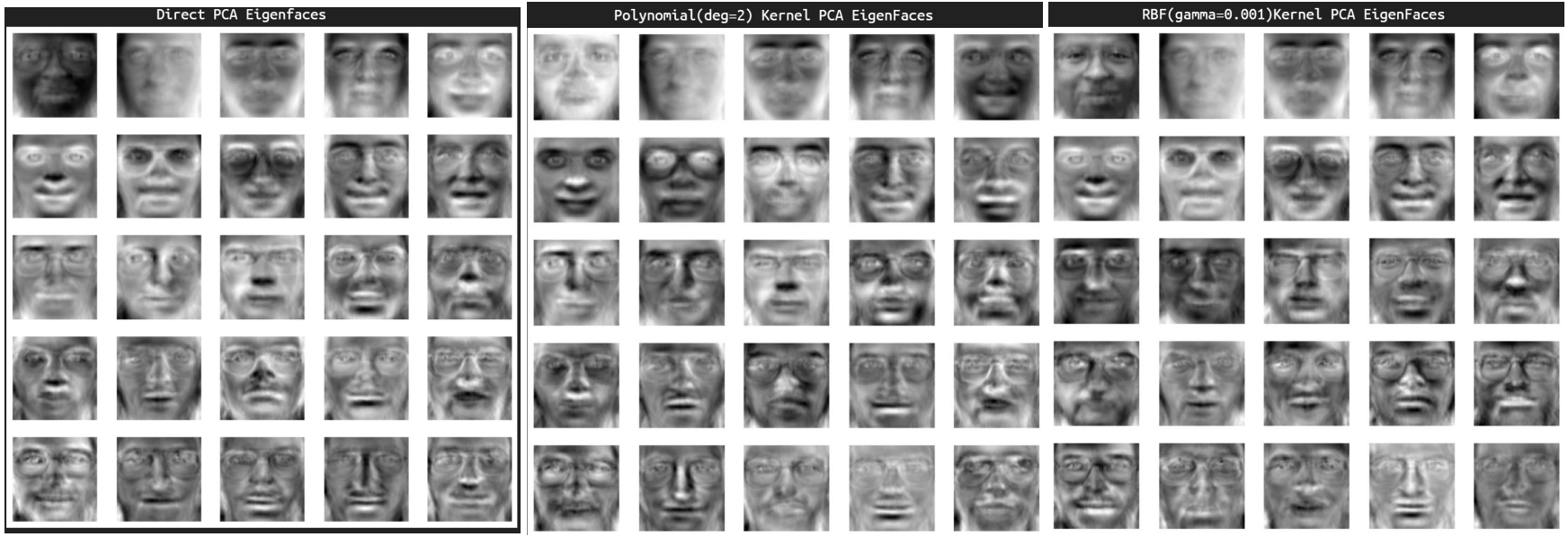
Gamma = 5

	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

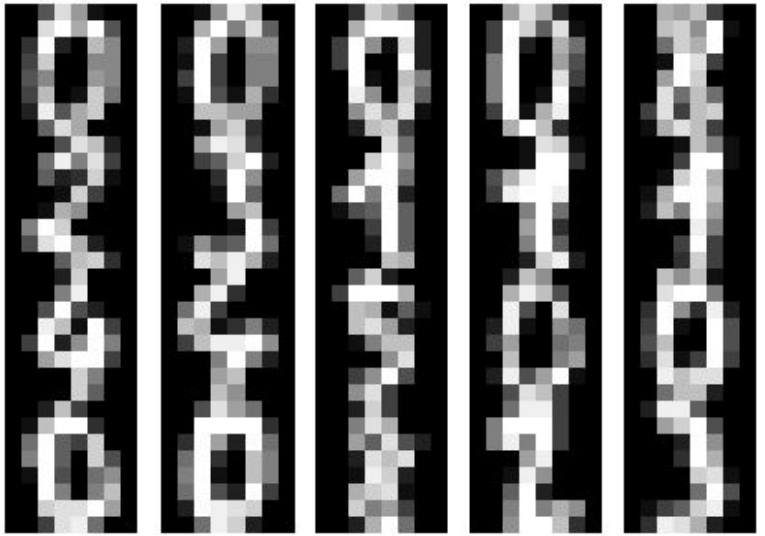
	dims	train_acc	test_acc
0	10	1.0	0.8625
1	25	1.0	0.9375
2	50	1.0	0.8750
3	100	1.0	0.8750
4	300	1.0	0.7500

Olivetti Eigenfaces



Digits

- The dataset has 1797 images belonging to 10 classes
- The image size is 8 x 8, giving us only 64 dimensions to start with.



Digits

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

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

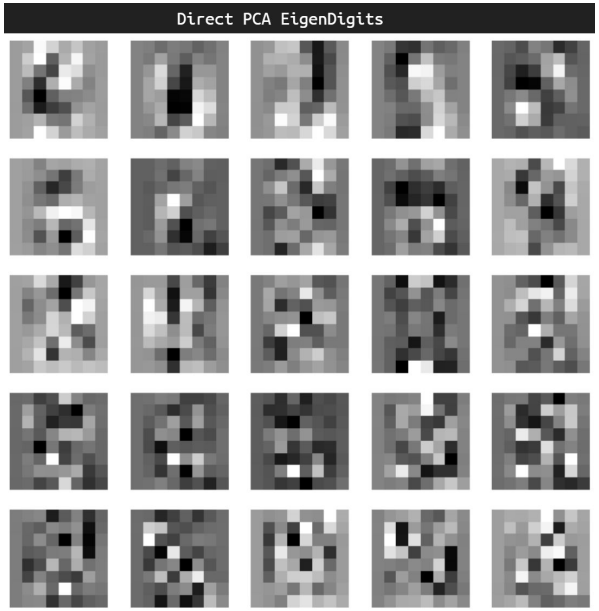
Gamma = 5

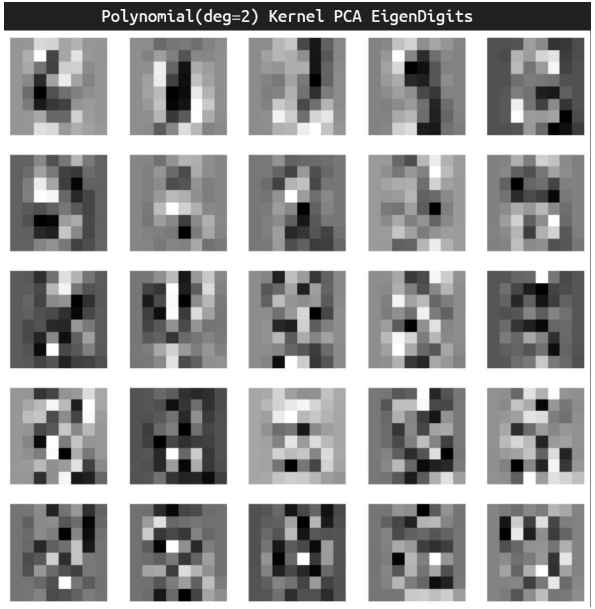
	dims	train_acc	test_acc
0	5	1.0	0.1
1	10	1.0	0.1
2	25	1.0	0.1
3	40	1.0	0.1
4	64	1.0	0.1

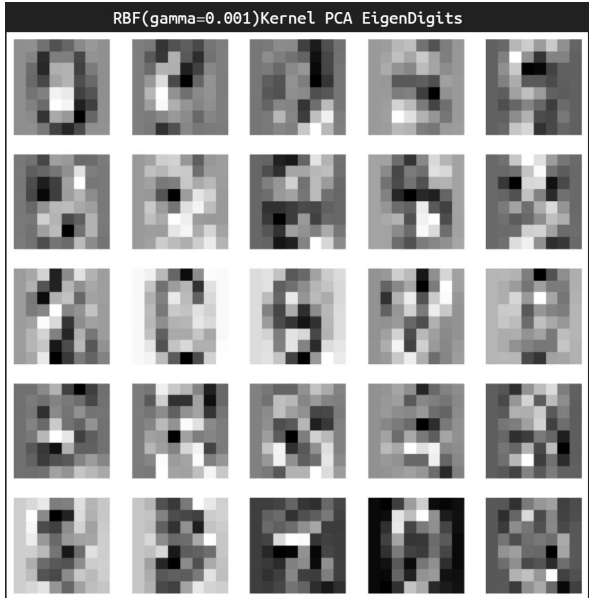
Gamma = 0.001

	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





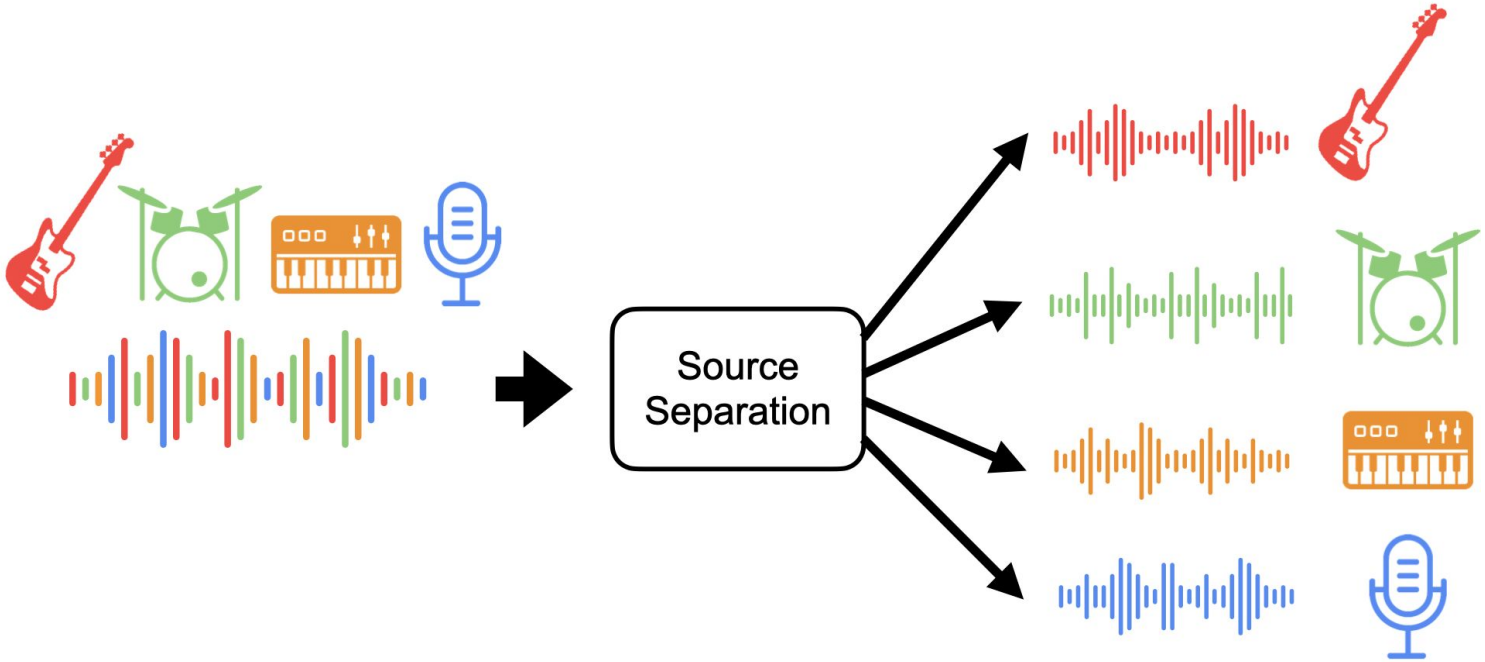


References

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

Independent Component Analysis (ICA)

Blind Source Separation problem



Solution

- Independent Component Analysis provides a solution to this problem.

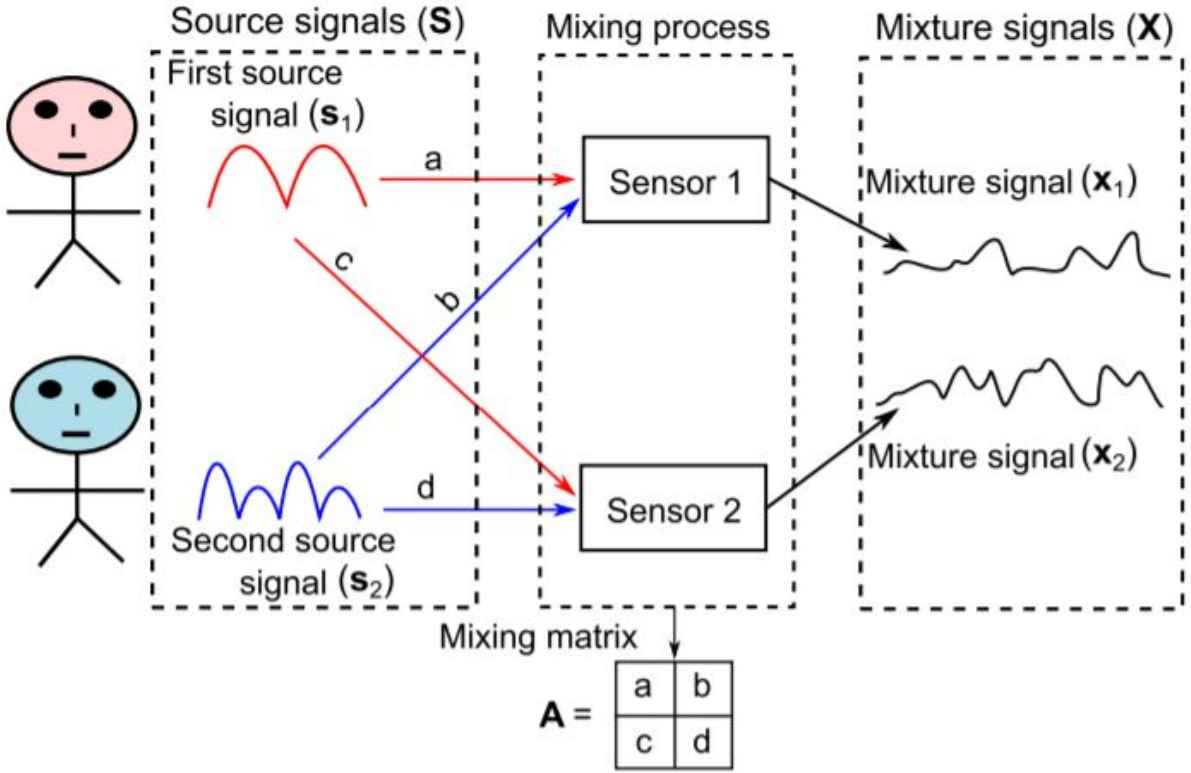
- We will focus on the FastICA algorithm, which extracts independent components from mixed signals by maximizing the non-Gaussianity.

Defining the ICA model

- At some time index t:

$$x_1(t) = a_{11}s_1 + a_{12}s_2$$
$$x_2(t) = a_{21}s_1 + a_{22}s_2$$

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


Source signals (S)
Mixing process
Mixture signals (X)
First source signal (s1)
a
c
b
d
Second source signal (s2)
Sensor 1
Sensor 2
Mixture signal (x1)
Mixture signal (x2)
Mixing matrix
A =
a
b
c
d

$$x_1(t) = a_{11}s_1 + a_{12}s_2$$
$$x_2(t) = a_{21}s_1 + a_{22}s_2$$

- We can write the above system of equations as:

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

- Also incorporating the time index, we have:

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

- The problem becomes:

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

Assumptions in ICA

- Source signals, $s_1(t)$ and $s_2(t)$ are assumed to be statistically independent.

- The signals $s_1(t)$ and $s_2(t)$ should not have Gaussian distribution.

Ambiguities in ICA

- We cannot determine the variances of the independent components.

- We cannot determine the order of independent components.

Principles of ICA estimation

Why maximize non-Gaussianity

- CLT - Sum of independent r.v. converges to gaussian, under certain conditions
- *Reasonable* extension - sum of two or more r.v. is *usually* more Gaussian in nature than any of the original r.v.
- Consider the ICA model $\mathbf{x} = \mathbf{As}$. Let $\mathbf{w}_i$ be one of the rows of $\mathbf{W} = \mathbf{A}^{-1}$. Then $y = \mathbf{w}_i^T\mathbf{x}$, our estimate to some s_i.

Why maximize non-Gaussianity

- Now, define $\mathbf{z = A^T w}$. Hence, we have:
  $y = \mathbf{w^T x = w^T A s = z^T s}$.

- According to CLT, the non-Gaussianity of $\mathbf{w^T x} = \mathbf{z^T s}$ will be maximum only when $\mathbf{z^T s}$ is equal to one of the independent components.

A measure of non-Gaussianity: Negentropy

- Entropy H(**y**): $H(\mathbf{y}) = -\int f(\mathbf{y}) \log f(\mathbf{y}) d\mathbf{y}.$
- Gaussian random variable has the largest entropy amongst all random variables of equal variance.
- Negentropy J(**y**):

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

- Calculation of negentropy is computationally difficult, hence we use an approximation.

Approximation to Negentropy

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

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

Preprocessing for ICA

Centering

- To center **x** is to subtract its mean vector, i.e **m** = E(**x**). This can be generalized across various time steps.

- One can get the mean of the independent components by doing $\mathbf{A}^{-1}\mathbf{m}$.

Whitening

- Linearly transform the observation vector **x** such that the new vector **x-tilde** is white, i.e its components are uncorrelated and their variances equal 1.

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

Whitening

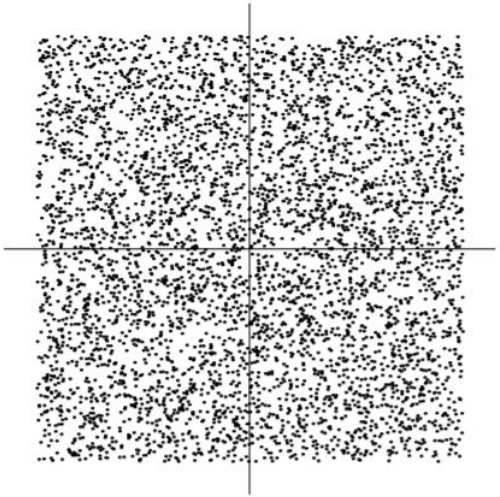
$$E\{\mathbf{x}\mathbf{x}^T\} = \mathbf{E}\mathbf{D}\mathbf{E}^T,$$

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

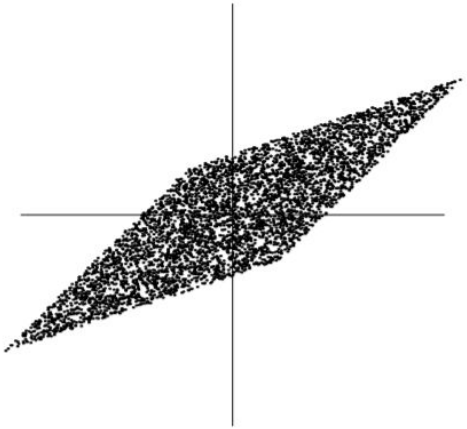
$$\tilde{\mathbf{x}} = \mathbf{E}\mathbf{D}^{-1/2}\mathbf{E}^T\mathbf{A}\mathbf{s} = \tilde{\mathbf{A}}\mathbf{s}$$

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

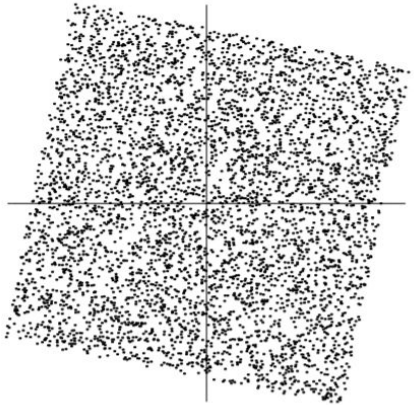
(Orthonormal matrix)



Original Distribution



Mixed Observations



After whitening

The FastICA Algorithm

FastICA for one unit

1. Choose an initial (e.g. random) weight vector $\mathbf{w}$.
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.

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

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

FastICA for multiple units

- We run the FastICA for multiple weight vectors to estimate several independent components. In addition, to prevent different vectors from converging to the same maxima, we need to decorrelate the outputs $\mathbf{w}^T_1\mathbf{x}$, $\mathbf{w}^T_2\mathbf{x}$, ... $\mathbf{w}^T_n\mathbf{x}$.

- So, after each iteration, we perform:

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

$$for\ 1\ to\ the\ number\ of\ components:$$

$$repeat\ until\ w_p^T w_{p+1} \approx 1:$$

$$w_p = \frac{1}{n}\sum_i^n X g(W^T X) - \frac{1}{n}\sum_i^n g'(W^{TX})W$$

$$w_p = w_p - \sum_{j=1}^{p-1}(w_p^T w_j)w_j$$

$$w_p = \frac{w_p}{\|w_p\|}$$

$$W = [w1, w2...]$$

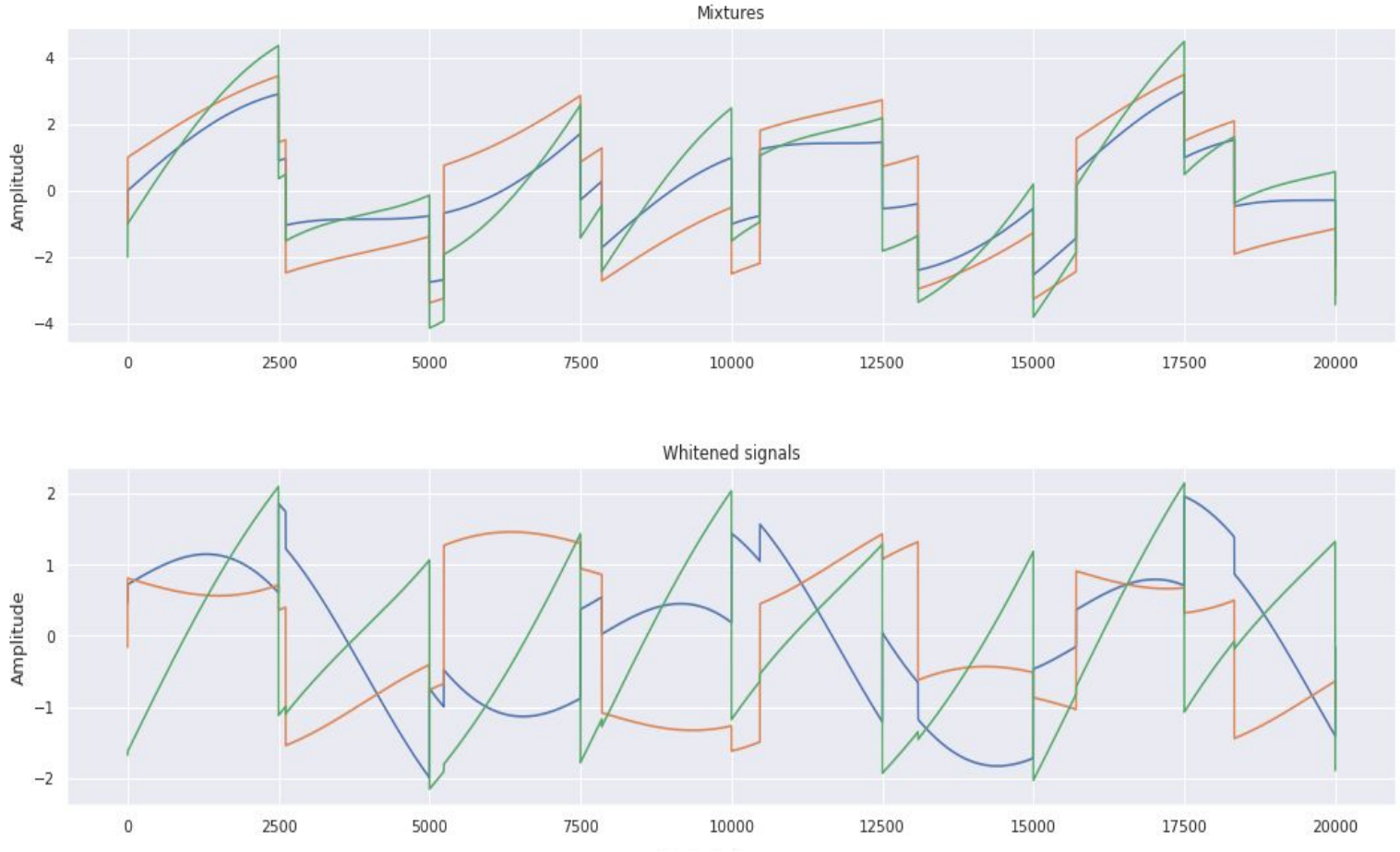
Implementation and Results

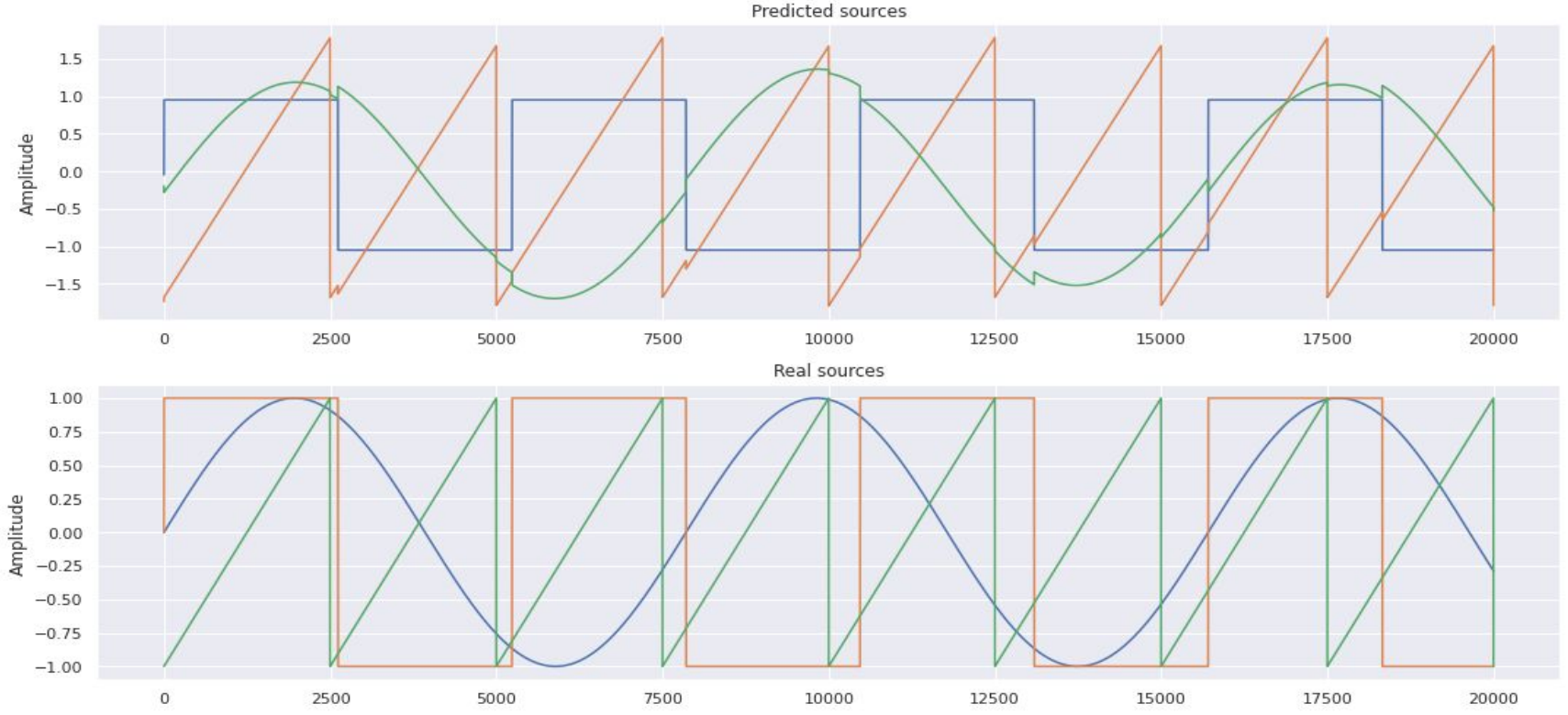
ICA on synthetic signals

- To synthetically generate 3 mixture signals, we mixed 3 separate signals: a sine wave, a square wave and a sawtooth wave over 20000 time steps.

- We then mixed them using the mixing matrix:

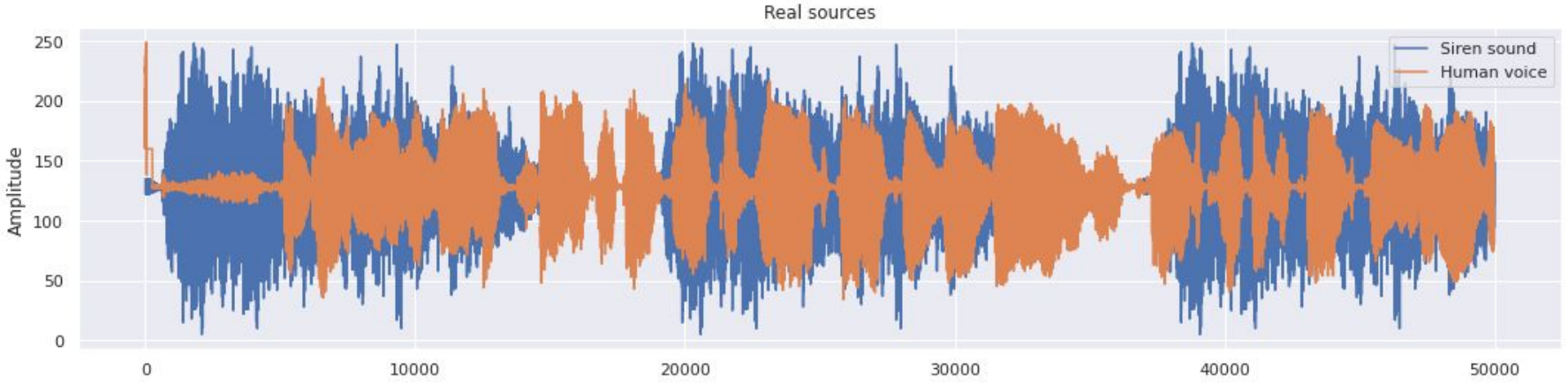
```
A = [[1, 1, 1], [0.5, 2, 1.0], [1.5, 1.0, 2.0]]
```

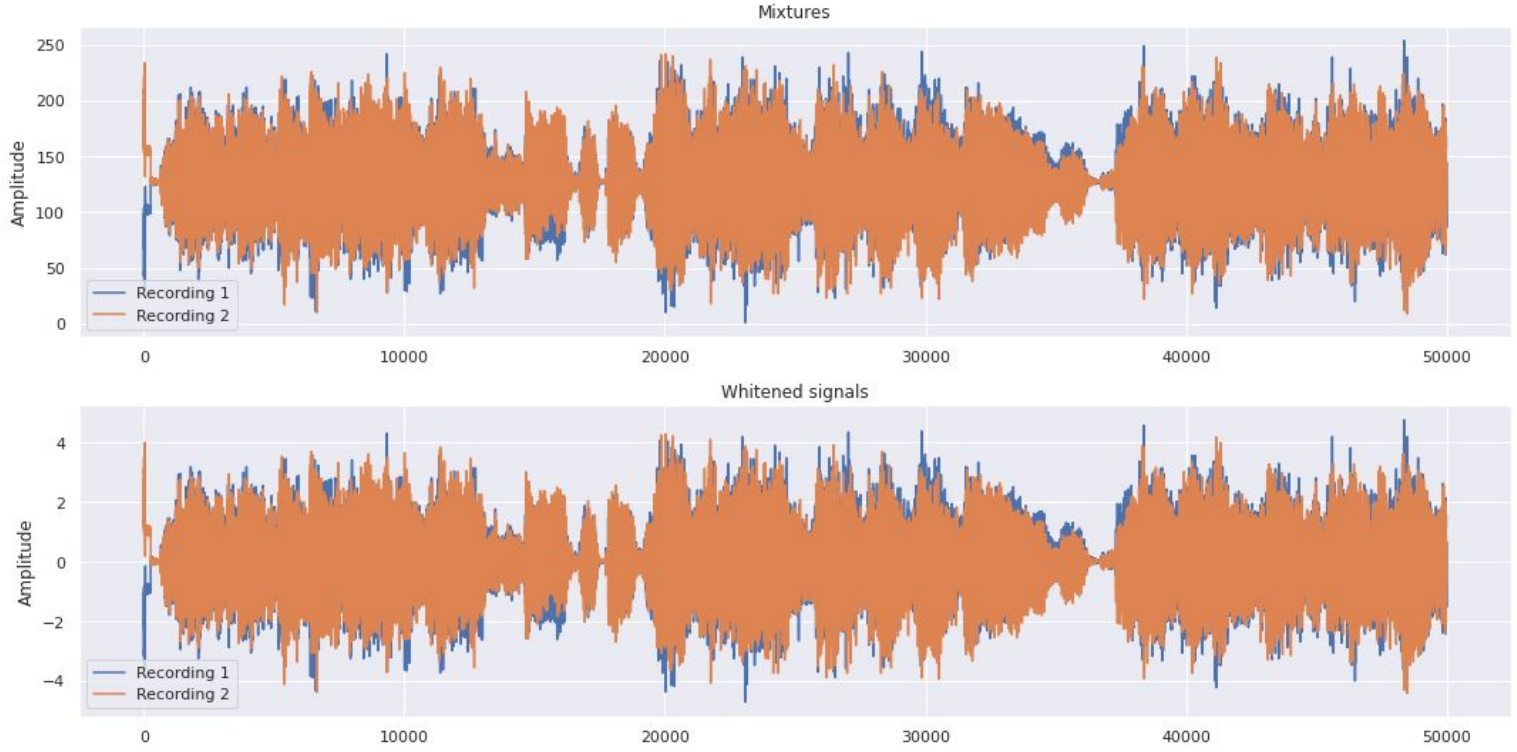

Mixtures
Amplitude
4
2
0
−2
−4
0
2500
5000
7500
10000
12500
15000
17500
20000
Whitened signals
Amplitude
2
1
0
−1
−2
0
2500
5000
7500
10000
12500
15000
17500
20000

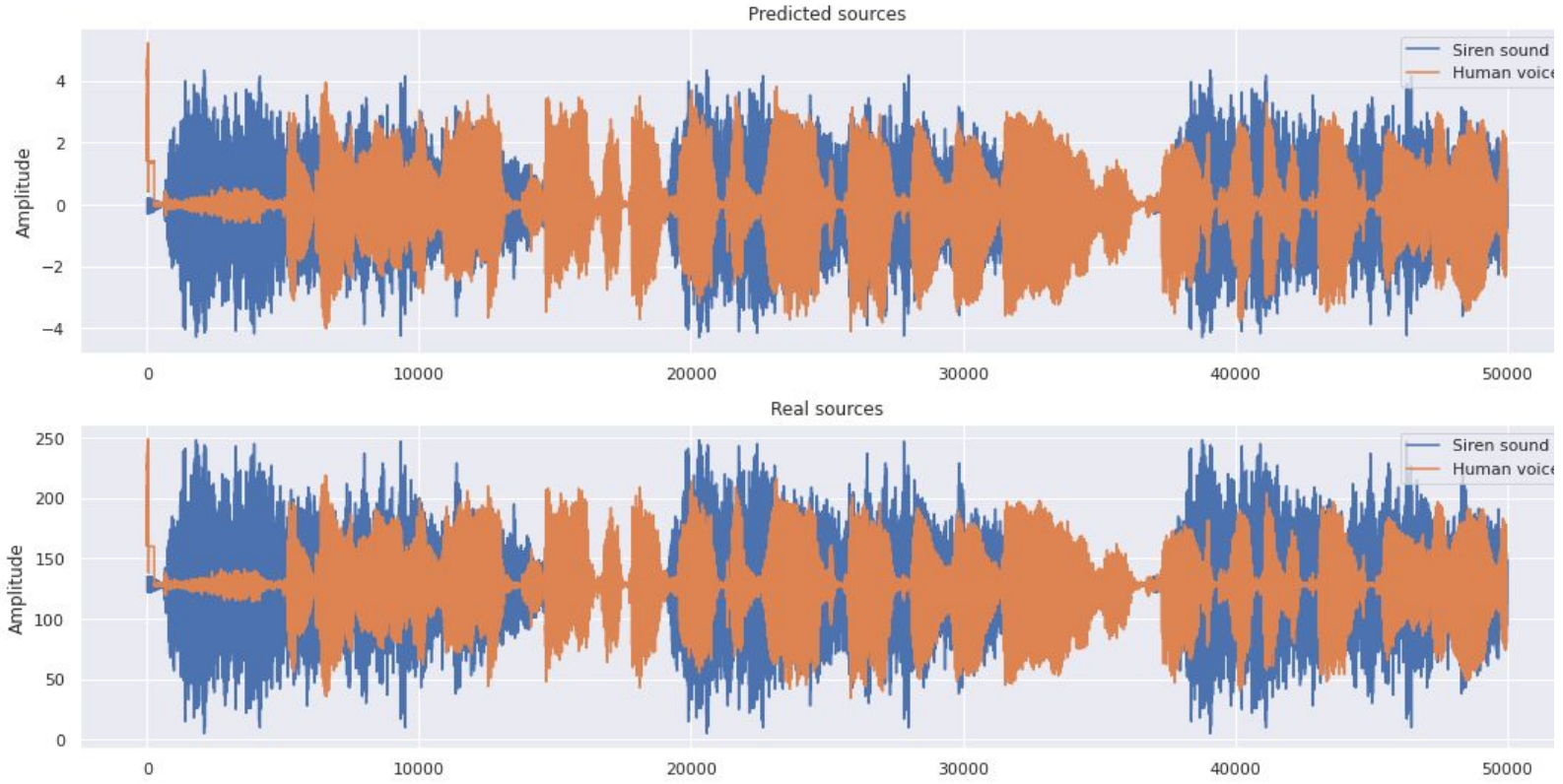

Predicted sources
Amplitude
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
2500
5000
7500
10000
12500
15000
17500
20000
Real sources
Amplitude
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2500
5000
7500
10000
12500
15000
17500
20000

ICA on mixed audio signals

- We had two already mixed signals:

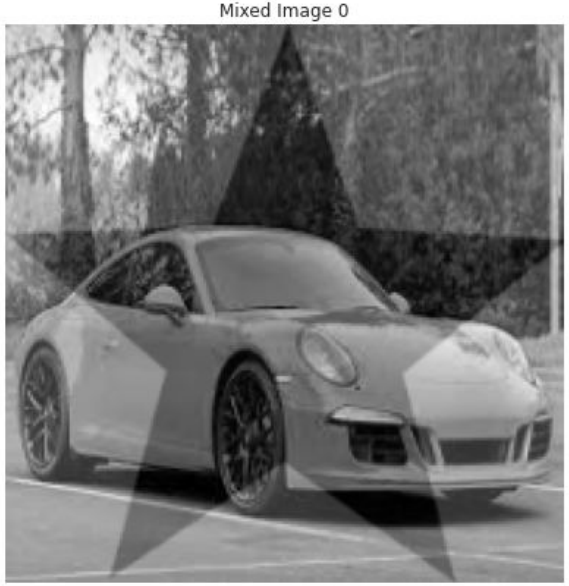


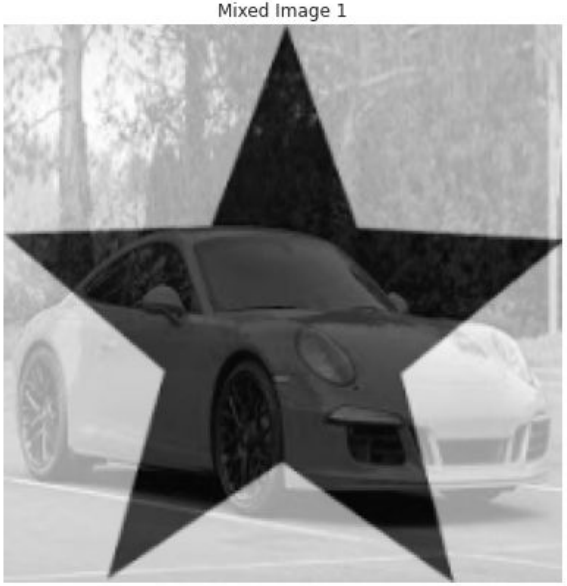

Mixtures
Amplitude
250
200
150
100
50
0
Recording 1
Recording 2
0
10000
20000
30000
40000
50000
Whitened signals
Amplitude
4
2
0
−2
−4
Recording 1
Recording 2
0
10000
20000
30000
40000
50000


Predicted sources
Siren sound
Human voice
Amplitude
4
2
0
−2
−4
0
10000
20000
30000
40000
50000
Real sources
Siren sound
Human voice
Amplitude
250
200
150
100
50
0
0
10000
20000
30000
40000
50000

ICA on images

- We took two grayscale images and mixed them synthetically. Mixing matrix: `[[1, 2], [3, 1]]/4`





Whitened Image 0

Whitened Image 1

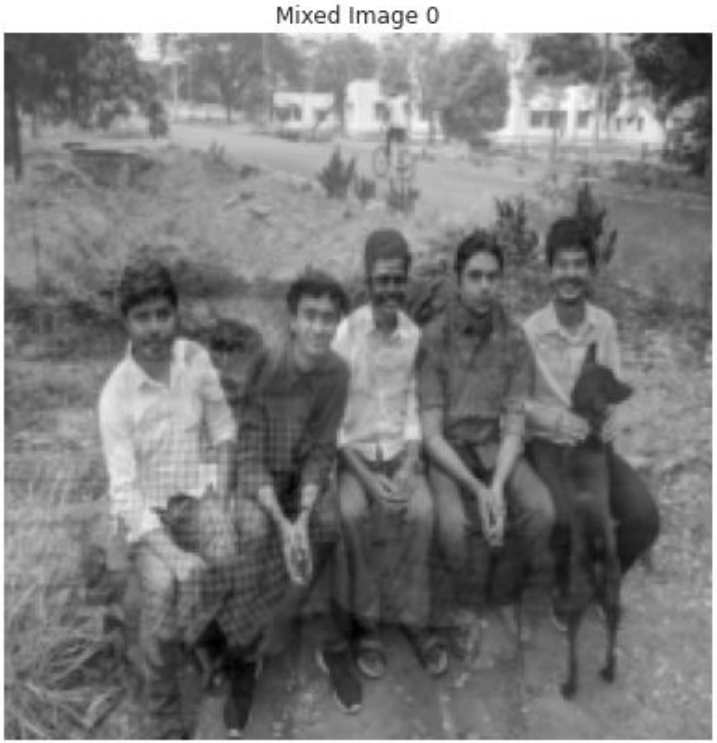
Separated Image 0

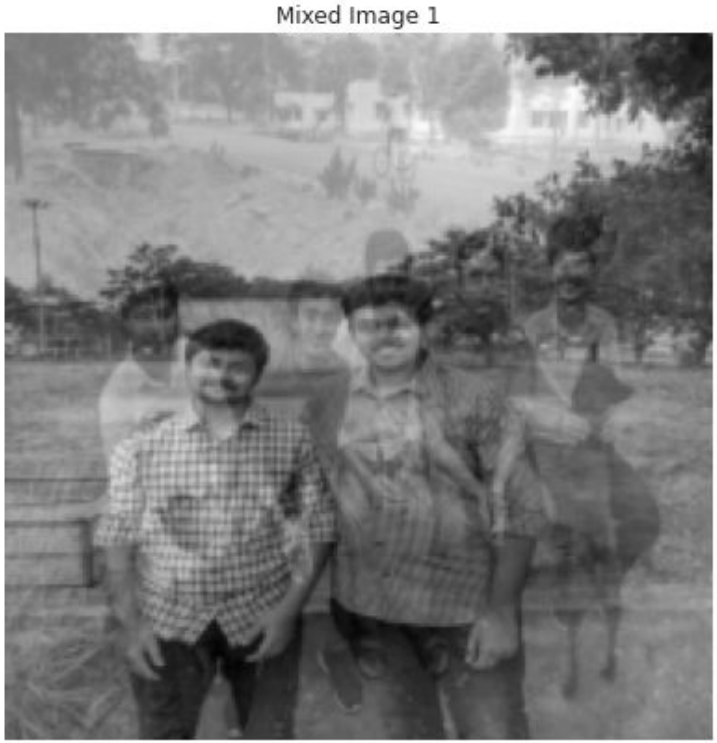
Separated Image 1

Original Image 0

Original Image 1

- 200 x 200 images. Mixing matrix: `[[1, 2], [2, 1]]/3`





Separated Image 0

Separated Image 1

Whitened Image 0

Whitened Image 1

Original Image 0

Original Image 1

- 500x500 images, Mixing matrix: `[[1, 2], [2, 1]]/3`

Mixed Image 0

Mixed Image 1

Whitened Image 0

Whitened Image 1

Separated Image 0

Separated Image 1

Original Image 0

Original Image 1

Conclusion

- Whitening tended to separate independent components properly in case of images, but not in case of audio signals.

- FastICA tended to perform well on high-resolution mixtures of images. We believe that this is due to availability of more number of samples to estimate non-Gaussianity.

- Widespread applications of ICA in audio processing, biomedical signal processing, image processing, telecommunications, etc.

Bibliography

- 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

Thank you