

Reading Assignment 2 - Group 4

Adhokshaja V Madhwaraj

160010032

Abhishek R

160020036

Rohan Sheelvant

160020005

Siddharth Sagar

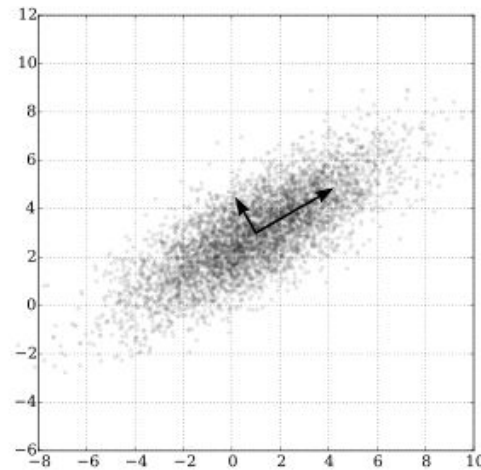
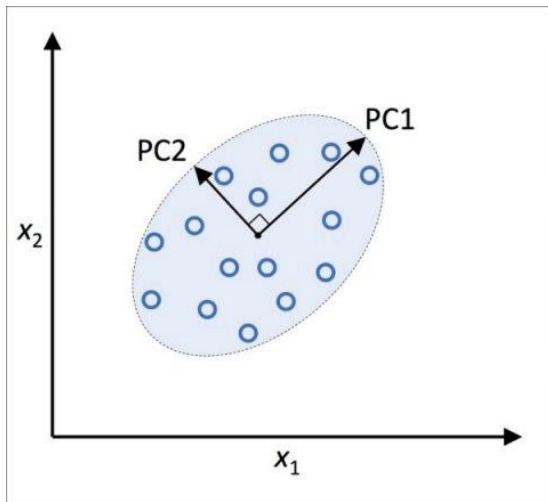
160030034

Principal Component Analysis

INTRODUCTION

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values that are linearly uncorrelated variables called principal components. The first component is chosen such that it has maximum variance (accounts for most of the variability of the data considered), and each of the succeeding components in turn has the highest variance possible in the orthogonal direction to the preceding components. In a nutshell, PCA aims to find the directions of maximum variance in high-dimensional data and projects it onto a new subspace with equal or fewer dimensions than the original one

The curse of dimensionality can be handled very neatly using the method of PCA. Dimensionality reduction is a way to reduce the complexity of a model and avoid overfitting of data. PCA can be used for feature selection and feature extraction. Feature selection is a method in which a subset of features are selected from the original set of features. Feature extraction is the method in which information is derived from the original features, to construct a new feature subspace.



MATHEMATICS AND CONCEPTS

How does PCA work?

- We calculate a matrix that summarizes how the variables relate to one another - Covariance Matrix
- We then break this matrix down into two separate components: Direction (EigenVectors of the data) and Magnitude (EigenValues, how important each direction is)
- We then transform the original data to align with the important directions (choose the number of components according to the requirement and compute power).
- By projecting our data into a smaller space, we're reducing the dimensionality of our feature space... but because we've transformed our data in these different "directions," we've made sure to keep all original variables in our model.

The Mathematics behind PCA

Since the procedure of PCA involves first finding the direction of maximum variance. To begin with, we consider the projection onto a 1-D space ($\mathbf{M} = \mathbf{1}$). The direction of this space will be \mathbf{u}_1 , where \mathbf{u}_1 is a unit vector. Every data point in this space will be denoted by $\text{dot}(\mathbf{u}, \mathbf{x})$.

The mean and variance of the data, can be given as follows:

$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \quad \frac{1}{N} \sum_{n=1}^N \{ \mathbf{u}_1^T \mathbf{x}_n - \mathbf{u}_1^T \bar{\mathbf{x}} \}^2 = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1$$

Where S is the covariance matrix as denoted below,

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T.$$

We try to maximise the value of $\mathbf{u}^T \mathbf{S} \mathbf{u}$. This is done along with a constraint on \mathbf{u}_1 , such that $\|\mathbf{u}_1\| = 1$. Using a Lagrange multiplier for constrained maximisation, we get,

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

We now try to maximise this above quantity, by setting the derivative wrt \mathbf{u}_1 to be 0, we obtain the following expression,

$$\mathbf{S} \mathbf{u}_1 = \lambda_1 \mathbf{u}_1$$

This says that \mathbf{u}_1 is an eigenvector of the covariance matrix of the data S , with λ as the eigenvalue corresponding to the eigenvector \mathbf{u}_1 . Multiply the above expression by \mathbf{u}_1^T on both sides, we get the following,

$$\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 = \lambda_1$$

This eigenvector that we get is known as the 1st Principal Component.

By the definition of PCA, we need to have the succeeding principal components to be orthogonal to the subsequent ones, hence when we consider the general case of reducing the data (which is D -Dimensional) onto a smaller M -Dimensional projection space. The maximised optimal linear projection for which the variance of the projected data is maximised can now be defined as the M eigenvectors of the covariance matrix \mathbf{S} , $\{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_M\}$, and the corresponding eigenvalues $\{\lambda_1, \lambda_2, \dots, \lambda_M\}$. This can be proven by induction.

In the code, we are using SVD (Singular Value Decomposition) to find the eigenvectors and eigenvalues of the covariance matrix S . In our case the covariance matrix S can be decomposed into the form $U D U^*$, where U holds the

eigenvectors, and D is a diagonal matrix which holds the eigenvalues corresponding to the eigenvectors.

Total and Explained Variance

When we select the components, we are looking for those components that have the maximum variance (that is, those components that hold the maximum amount of information). Thus, we need to sort the eigenvectors in descending order of their importance in terms of variance (this is given by the magnitude of the eigenvalue).

Variance Explained Ratios gives us this information, and is simply given by the following expression:

$$\frac{\lambda_j}{\sum_{j=1}^d \lambda_j}$$

Feature Transformation

Suppose we have N data points in our dataset, each of which is D -Dimensional. We need to convert this into M -Dimensional data. We construct a transformation matrix W which consists of the first M important eigenvectors. This W has dimensions of $(D \times M)$.

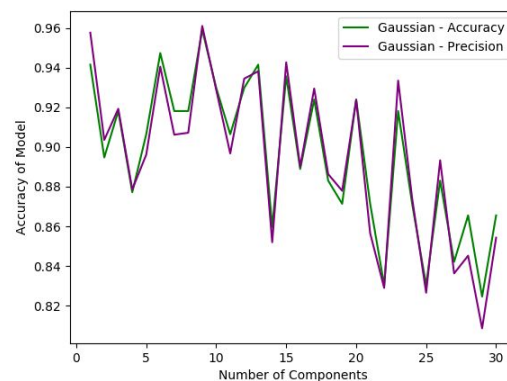
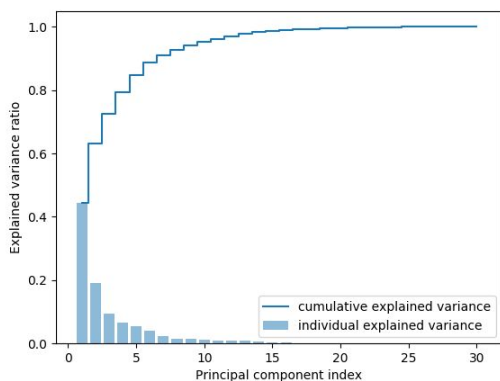
$$X' = WX$$

Here, X' is the modified data, which now has dimensions of $(N \times M)$

ANALYSIS OF PCA ON STANDARD DATASETS

Dataset 1: Breast Cancer Data

Binary Classification of tumours as Malignant or Benign (0 or 1) based on 30 features.



Total number of features: 30

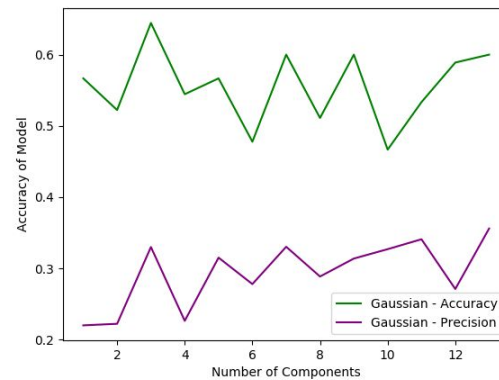
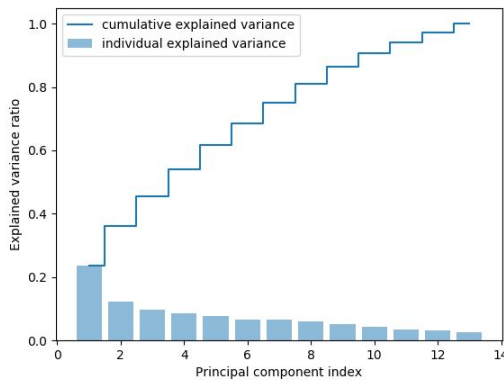
Total number of samples: 569

Optimal number of component: 9

Here we can see the maximum variance (nearly 90%) of the data is captured by 9 components. Accuracy observations and comparisons are tabulated down below.

Dataset 2: Heart Disease Data

Multiclass Classification of Heart Disease from 0-4, based on 13 features



Total number of features: 13

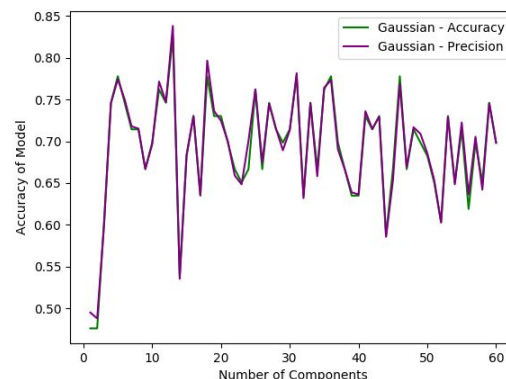
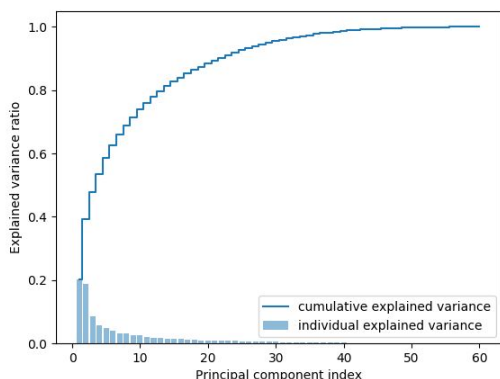
Total number of samples: 297

Optimal number of components: 3

The best performance is obtained when the number of components is 3. Since almost all the other components are almost equal, we see a tumultuous graph, with ups and downs.

Dataset 3: Sonar Data

Binary classification of Sonar data into mines or rocks (mines have metals) based on 60 geological features



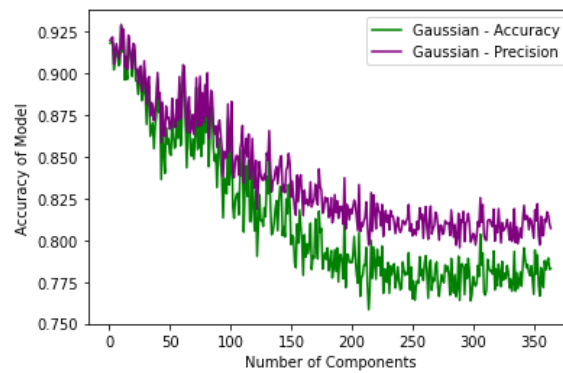
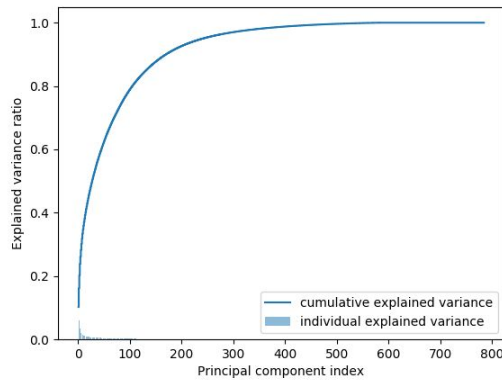
Total number of features: 60

Total number of samples: 208

Optimal number of components: 13

Dataset 4: MNIST Data

Selection of two random handwritten digits for classification



Total number of features: 784

Total number of samples: 12,156

Optimal number of components: 30

Though this graph is hard to read at around 30 components we see a spike in the accuracy and precision, thus confirming that 30 components is sufficient for this task. As we increase the number of components, there is slight overfitting and thus you can see a drop in the accuracy and precision.

Accuracy and Precision Values

Here is an analysis of the accuracy and precision values calculated from the Confusion Matrix from each dataset's classification task. The Classification Method that was used was Gaussian Naive Bayes Classification.

The general trend is that the accuracy and precision that are obtained when using PCA and reducing the number of components (hence features), are higher than that obtained on applying the same classification task on the entire dataset.

Feature extracted Data is after processing with PCA, after multiplying the original data with the transformation matrix that is obtained.

[Continued]

Dataset	Optimal number of Features	Full Data (All Features)		Feature Extracted Data	
		Accuracy %	Precision %	Accuracy %	Precision %
Breast Cancer	9	94.152	94.047	95.906	96.103
Heart Disease	3	40	26.365	64.444	35.593
Sonar Data	13	79.365	78.853	82.539	83.796
MNIST Data	30	90.894	91.276	92.853	92.951

Kernel Principal Component Analysis

Introduction

Standard PCA only allows linear dimensionality reduction. However, if the data has more complicated structures which cannot be well represented in a linear subspace, standard PCA will not be very helpful. Fortunately, kernel PCA allows us to generalize standard PCA to nonlinear dimensionality reduction.

MATHEMATICS AND CONCEPTS

To understand the utility of kernel PCA, particularly for clustering, observe that, while N points cannot, in general, be linearly separated in $d < N$ dimensions, they can almost always be linearly separated in $d \geq N$ dimensions. That is, given N points, \mathbf{x}_i , if we map them to an N -dimensional space with

$\mathbf{x} \rightarrow \phi(\mathbf{x})$ where $\phi(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^N$, which is called “kernel function”,

it is easy to construct a hyperplane that divides the points into arbitrary clusters. Of course, this ϕ creates linearly independent vectors, so there is no covariance on which to perform eigendecomposition explicitly as we would in linear PCA.

Instead, in kernel PCA, a non-trivial, arbitrary ϕ function is 'chosen' that is never calculated explicitly, allowing the possibility to use very-high-dimensional ϕ 's if we never have to actually evaluate the data in that space (This is called Kernel trick). Since

we generally try to avoid working in the ϕ -space, which we will call the 'feature space', we can create the N-by-N kernel

$$K = k(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}), \Phi(\mathbf{y})) = \Phi(\mathbf{x})^T \Phi(\mathbf{y})$$

Often, the mathematical definition of the RBF kernel is written and implemented as

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|_2^2\right)$$

The Mathematics behind PCA

In this section we will discuss the following idea: transformation of the dataset to a new higher-dimensional (in some cases infinite-dimensional) feature space and the use of PCA in that space in order to produce uncorrelated features.

Let us denote a covariance matrix in a new feature space as \mathbf{C}

$$\begin{aligned}\varphi: \mathbb{R}^D &\rightarrow \mathbb{F}, \\ \mathbf{C} &= \frac{1}{N} \Phi^T \Phi\end{aligned}$$

where $\Phi = \{\phi(\mathbf{x}_n)\}_{n=1}^N$. Will consider that the dimensionality of the feature space \mathbb{F} equals to $M > D$.

Eigendecomposition of \mathbf{C} is given by

$$\mathbf{C} \mathbf{v}_i = \lambda_i \mathbf{v}_i, \forall i = 1, \dots, M$$

By the definition of \mathbf{C}

$$\frac{1}{N} \Phi^T \Phi = \frac{1}{N} \sum_{n=1}^N \phi(\mathbf{x}_n) \phi(\mathbf{x}_n)^T,$$

and therefore $\forall i = 1, \dots, M$

$$\frac{1}{N} \sum_{n=1}^N \phi(\mathbf{x}_n) \{\phi(\mathbf{x}_n)^T \mathbf{v}_i\} = \lambda_i \mathbf{v}_i.$$

It is obvious to see, that \mathbf{v}_i is a linear combination of $\phi(\mathbf{x}_n)$ and thus can be written as

$$\mathbf{v}_i = \sum_{n=1}^N a_{i_n} \phi(\mathbf{x}_n).$$

Substituting it to the equation above and writing it in a matrix notation, we get

$$\mathbf{K}\mathbf{a}_i = \lambda_i N \mathbf{a}_i,$$

where \mathbf{K} is a Gram matrix in \mathbb{F} , $\mathbf{K}_{ij} = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$ and \mathbf{a}_i are column-vectors with elements a_{in} . Eigenvectors of \mathbf{C} should be orthonormal, therefore, we get the following:

$$N \lambda_i \mathbf{a}_i^\top \mathbf{a}_i = 1.$$

Having eigenvectors of \mathbf{C} , we can get the projection of an item $\mathbf{x} \in \mathbb{R}^D$ on i -th eigenvector:

$$z_i(\mathbf{x}) = \sum_{n=1}^N a_{in} \phi(\mathbf{x})^\top \phi(\mathbf{x}_n).$$

So far, we have assumed that the mapping $\phi(\cdot)$ is known. From the equations above, we can see, that only a thing that we need for the data transformation is the eigendecomposition of a Gram matrix \mathbf{K} . Dot products, which are its elements can be defined without any definition of $\phi(\cdot)$. The function defining such dot products in some Hilbert space is called kernel. Kernels are satisfied by Mercer's theorem. There are many different types of kernels, there are several popular:

1. Linear: $k(\mathbf{x}_n, \mathbf{x}_m) = \mathbf{x}_n^\top \mathbf{x}_m$;
2. Gaussian: $k(\mathbf{x}_m, \mathbf{x}_n) = \exp(-\gamma \|\mathbf{x}_m - \mathbf{x}_n\|^2)$;
3. Polynomial: $k(\mathbf{x}_m, \mathbf{x}_n) = (\mathbf{x}_m^\top \mathbf{x}_n)^d$.

Using a kernel function we can write new equation for a projection of some data item onto i -th eigenvector:

$$z_i(\mathbf{x}) = \sum_{n=1}^N a_{in} k(\mathbf{x}, \mathbf{x}_n).$$

So far, we have assumed that the columns of Φ have zero mean. Using

$$\hat{\varphi}(\mathbf{x}) = \varphi(\mathbf{x}) - \frac{1}{N} \sum_{n=1}^N \varphi(\mathbf{x}_n),$$

and substituting it to the equation for \mathbf{K} , we get

$$\mathbf{K}_c = \mathbf{K} - \mathbf{1}_N \mathbf{K} - \mathbf{K} \mathbf{1}_N + \mathbf{1}_N \mathbf{K} \mathbf{1}_N,$$

where $\mathbf{1}_N$ is a matrix $N \times N$, where each element equals to $\frac{1}{N}$.

How does KPCA work?

- Calculate \mathbf{K} .
- Calculate \mathbf{K}_c .
- Find the eigenvectors \mathbf{a}_i of \mathbf{K}_c corresponding to nonzero eigenvalues λ_i and normalize them: $\mathbf{a}_i = \frac{1}{\sqrt{\lambda_i N}} \mathbf{a}_i$.
- Sort found eigenvectors in the descending order of corresponding eigenvalues.
- Perform projections onto the given subset of eigenvectors.

ANALYSIS OF KPCA ON STANDARD DATASETS

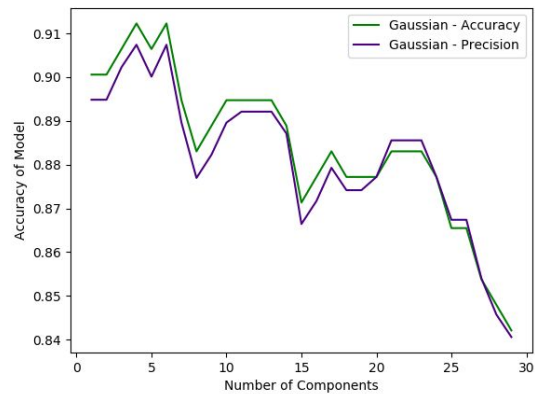
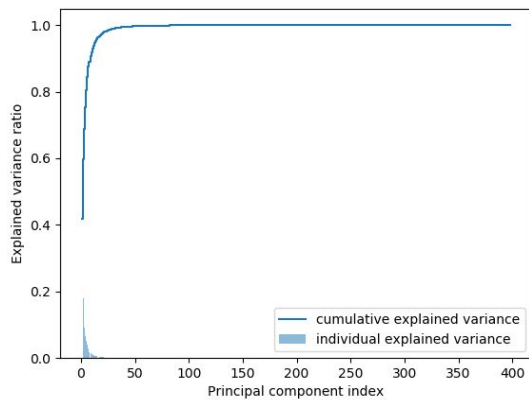
We are using RBF kernel i.e

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|_2^2\right)$$

With gamma = 0.001

Dataset 1: Breast Cancer Data

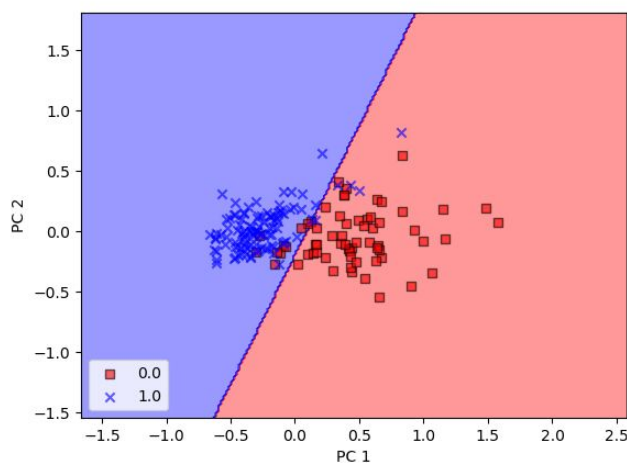
Binary Classification of tumours as Malignant or Benign (0 or 1) based on 30 features.



Total number of features: 30

Total number of samples: 569

Optimal number of component: 4

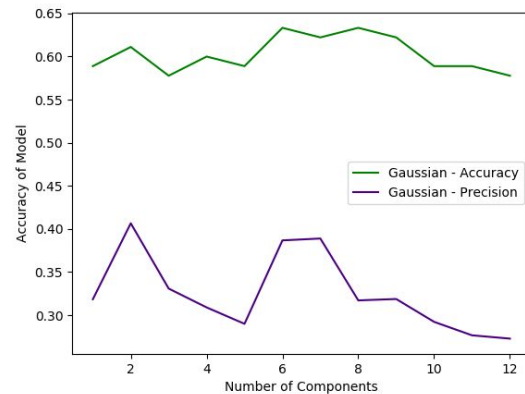
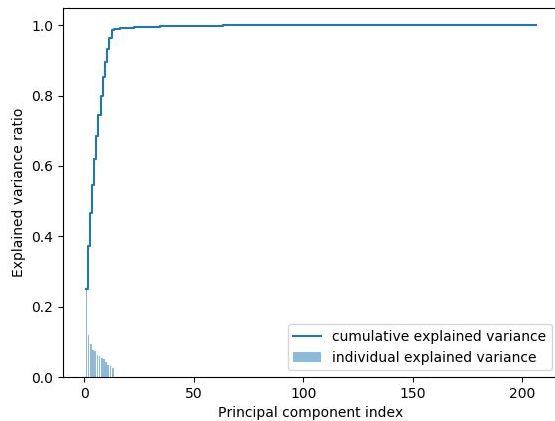


Plot of test data for 2 Principal component and classified based on logistic regression giving 91.22% accuracy

Here we can see that almost 75% of the variance of the Kernel value(after using kernel function on the data) is captured by 4 components. Accuracy observations and comparisons are tabulated down below.

Dataset 2: Heart Disease Data

Multiclass Classification of Heart Disease from 0-4, based on 13 features



Total number of features: 13

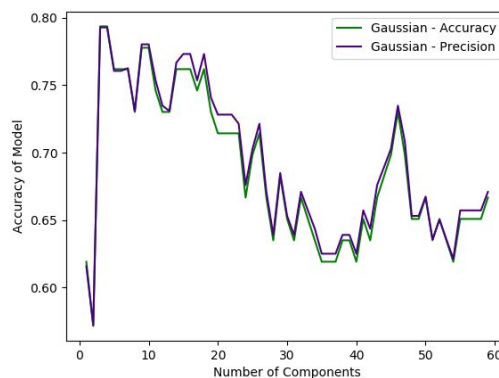
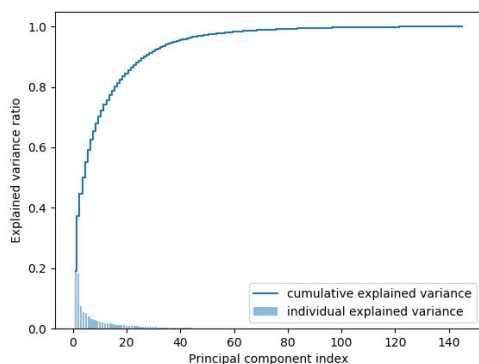
Total number of samples: 297

Optimal number of components: 6

The best performance is obtained when the number of components is 6. The cumulative variance is changing slowly compared to the first data set, so the number of number components for optimal classification also increased.

Dataset 3: Sonar Data

Binary classification of Sonar data into mines or rocks (mines have metals) based on 60 geological features



Total number of features: 60

Total number of samples: 208

Optimal number of components: 3

As we can see from the variance graph that the first 3 components contain almost 45% the variance and the data classification is most promising for using 3 components.

Dataset	Optimal number of Features	Full Data (All Features)		Feature Extracted Data	
		Accuracy %	Precision %	Accuracy %	Precision %
Breast Cancer	4	94.152	94.047	91.22	90.82
Heart Disease	6	40	26.365	63.33	38.67
Sonar Data	3	79.365	78.853	79.36	79.28

APPLICATIONS OF PCA

Using PCA and SVD for Image Compression task

Storage and retrieval of images is an important task on the internet, and many images are created at very high resolutions, and take up a lot of storage. PCA, due to its inherent definition and ability to reduce the dimensionality of images, is an obvious choice for this task of Image Compression and Reconstruction.

The task of Image Compression using these concepts can be implemented in two ways:

1. Using the covariance matrix of the image, then picking the top eigenvalues, and eigenvectors, to recreate the image using lesser data.
2. A directly reduced rank representation of the image can be used using basic Singular Value Decomposition, which gives the same results as the other method.

The computational block in these methods, would be the calculation of the SVD of either the Image or its Covariance matrix. The Complexity of the algorithm depends on the resolution of the image. If the image is of resolution (M pixels X N pixels) is $O(M^2N + N^3)$. This can be changed slightly by using more optimised numerical methods for the same. However in this implementation, standard libraries from Numpy were used.

Mathematics & Intuition behind the process

The image (M X N) can be assumed to be our data matrix **X**, which can be decomposed as the following:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

Where \mathbf{U} is a matrix of eigenvectors of $\mathbf{X}\mathbf{X}^T$, $\mathbf{\Sigma}$ is the diagonal matrix of singular values, and \mathbf{V}^T is the matrix of eigenvectors of $\mathbf{X}^T\mathbf{X}$. The columns of \mathbf{U} are the PCs, the diagonal elements of $\mathbf{\Sigma}$ are proportional to Standard Deviations of the PC.

The important part of this reconstruction procedure is how simply an image of the same resolution can be recreated by dimension-wise reconstruction. A suitable $K < P$, where P is the number of PCs, can be chosen for the compression of the Image. If K is equal to P , then the original picture is obtained.

The number of pixel values that have to be stored reduce drastically in this method. If we choose a particular value of $K < P$ (total dimensions), then the number of values to be stored are:


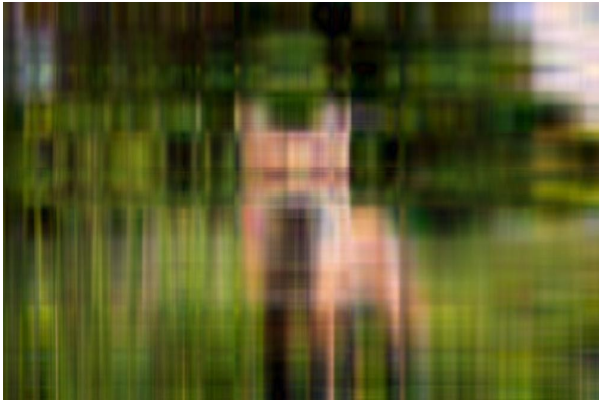
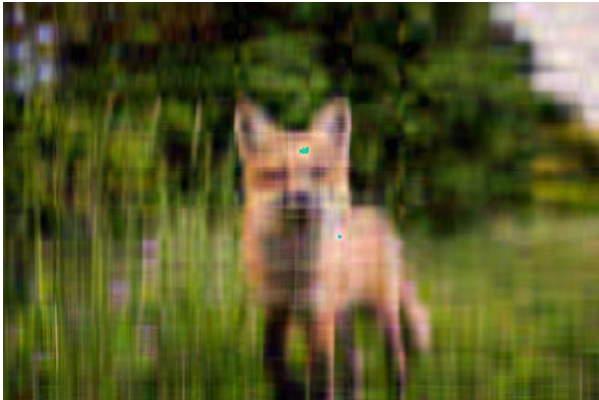
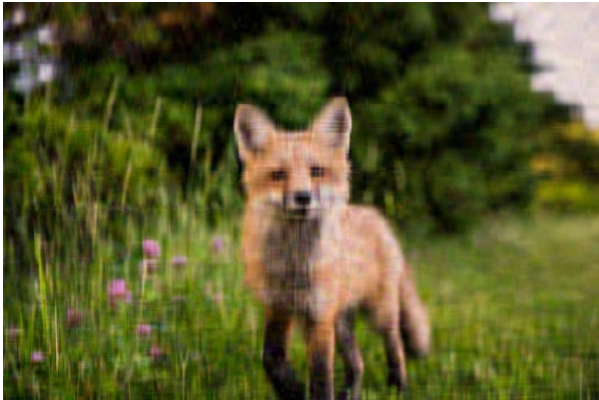


- Dimension of the image : $\mathbf{M} \times \mathbf{N}$
- Number of components : \mathbf{K}
- Size of the \mathbf{U} matrix : $\mathbf{M} \times \mathbf{K}$
- Size of the $\mathbf{\Sigma}$ matrix : $\mathbf{K} \times \mathbf{K}$
- Size of the \mathbf{V} matrix : $\mathbf{N} \times \mathbf{K}$
- Total number of values to be calculated : $\mathbf{K} \times (\mathbf{M} + \mathbf{N} + \mathbf{K})$

Just to give an idea of the numbers we are discussing, if we assume a Full HD image (1080 X 1920), with 200 components (which gives a very good reconstructed image, as will be shown below), the original representation requires **2,073,600** values, and the new representation requires **640,000**. This is nearly **30.86%** of the storage required, giving nearly the same quality to the eyes, at the same resolution.

The reconstruction can also be represented in the following format:

$$\widehat{\mathbf{X}} = \sum_{i=1}^k \sigma_{ii} \mathbf{u}_i \mathbf{v}_i^T.$$

It is however important to know and understand that this compression is lossy, that is information is lost in the compression.

<p>K = 2</p>  <p>Compression Rate : 99.5%</p>	<p>K = 5</p>  <p>Compression Rate : 98.75%</p>
<p>K = 10</p>  <p>Compression Rate : 97.486%</p>	<p>K = 25</p>  <p>Compression Rate : 93.66%</p>
<p>K = 50</p>  <p>Compression Rate : 87.129%</p>	<p>K = 100</p>  <p>Compression Rate : 73.509%</p>
<p>K = 200</p>	<p>ORIGINAL</p>



Here the compression rate can be calculated in the following method:

$$100 \times \left(1 - \frac{\text{Total number of units required for storage}}{\text{Total number of units required for original image}} \right)$$

Thus for an image which has low number of pixels (units) in storage, will have a high compression rate, as can be seen under each image.

An interesting application of these findings is the possibility to use a selection of the PC's as input to greatly reduce the computational burden of machine learning algorithms. For instance, for an algorithmic image classifier.

Using PCA and SVD for Video Compression

An obvious extension to the problem of Image Compression is Video Compression. Again, this problem can be tackled in multiple different ways:

1. Each individual frame of a video can be taken as an individual image and used as input to the Image Compression module built earlier.
2. The entire length of video can be considered as the dataset, with each frame of the video being a sample in it. If N frames are considered, each frame having K pixels, then the ensuing dataset has N X K size, and earlier PCA methods can be used for compression. This method however is very intensive on the RAM as the size can be in GBs for HD (1080p) videos of even a few seconds.
3. The third is to compress the video in small parts (this is considered assuming that there is correlation between the frames which are part of a short clip, as there is not much change in the setting of the video in this interval). Thus we get (number of pixels X fps) size matrices which are still workable in terms of RAM consumption.

For this implementation, I have considered the 1st option, since there was a constraint on memory. A full HD clip of about 5 seconds was considered, and the number of principal components that were used were 50, 100, and 200.

The resulting videos are shared as part of the folder.

Details of Video Compression Task performed

An animated video of around 1 minute was taken, and from it a sample with the following properties was considered:

- Resolution of the video : 1080p Full HD, 1920x1080 pixels RGB
- Frame rate : 24fps
- Time length of sample : 5s
- Total number of frames : $24 \times 5 = 120$ frames
- Principal components considered for analysis : 50, 100, 200
- Corresponding compression rates achieved :

Principal components	50	100	200
Compression Rate	92.645%	85.05%	69.13%

Independent Component Analysis

Introduction

Independent component analysis (ICA) is a computational method for separating a multivariate signal into additive subcomponents. ICA defines a generative model for the observed multivariate data, which is typically given as a large database of samples. In the model, the data variables are assumed to be linear mixtures of some unknown latent variables, and the mixing system is also unknown. The latent variables are assumed non gaussian and mutually independent, and they are called the independent components of the observed data. ICA is a special case of blind source separation. The aim of ICA is to extract useful information or source signals from data. These data can be in the form of images, stock markets, or sounds.

ICA is considered as an extension of the principal component analysis (PCA) technique. However, PCA optimizes the covariance matrix of the data which represents

second-order statistics, while ICA optimizes higher-order statistics such as kurtosis. Hence, PCA finds uncorrelated components while ICA finds independent components.

Principles of ICA estimation

In ICA, the goal is to find the unmixing matrix (\mathbf{W}) and then projecting the whitened data onto that matrix for extracting independent signals. This matrix can be estimated using three main approaches of independence, which result in slightly different unmixing matrices. The first is based on non-Gaussianity. This can be measured by some measures such as negentropy and kurtosis, and the goal of this approach is to find independent components which maximize the non-Gaussianity. In the second approach, the ICA goal can be obtained by minimizing the mutual information. Independent components can be also estimated by using maximum likelihood (ML) estimation. All approaches simply search for a rotation or unmixing matrix. Projecting the whitened data onto that rotation matrix extracts independent signals. The preprocessing steps are calculated from the data, but the rotation matrix is approximated numerically through an optimization procedure.

FastICA

FastICA algorithm extracts independent components by maximizing the non-Gaussianity by maximizing the negentropy for the extracted signals using a fixed-point iteration scheme [18]. FastICA has a cubic or at least quadratic convergence speed and hence it is much faster than Gradient-based algorithms that have linear convergence. Additionally, FastICA has no learning rate or other adjustable parameters which makes it easy to use.

FastICA can be used for extracting one IC, this is called one-unit, where FastICA finds the weight vector (\mathbf{w}) that extracts one independent component. The values of (\mathbf{w}) are updated by a learning rule that searches for a direction which maximizes the non-Gaussianity.

Cocktail Party Problem

The cocktail party problem is the task of hearing a sound of interest, often a speech signal, in this sort of complex auditory setting. The problem is intrinsically quite difficult, and there has been long standing interest in how humans manage to solve it. The goal of this problem is to detect or extract the sound with a single object even though different sounds in the environment are superimposed on one another.

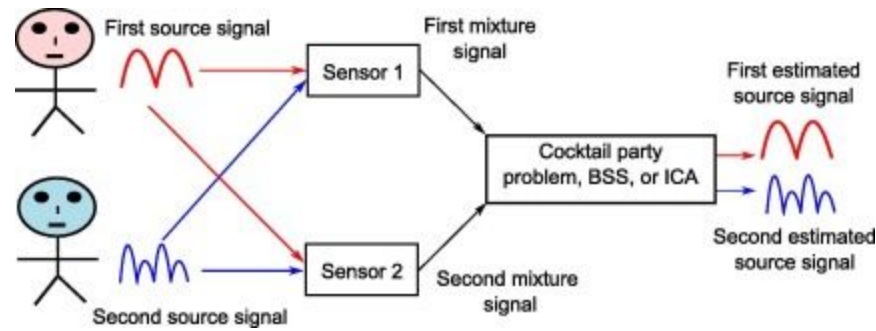
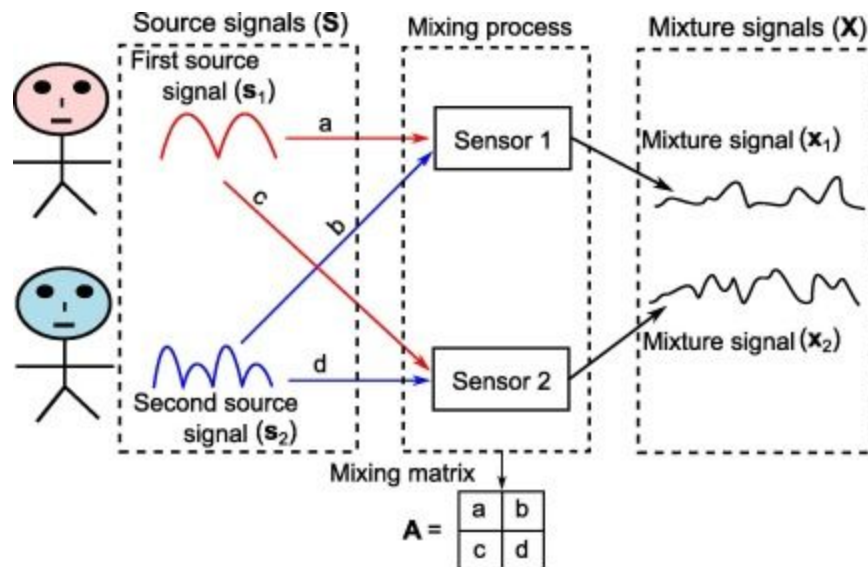


Fig 1.

Fig1 shows an example of the cocktail party problem. In this example, two voice signals are recorded from two different individuals, i.e., two independent source signals. Moreover, two sensors, i.e., microphones, are used for recording two signals, and the outputs from these sensors are two mixtures. The goal is to extract original signals from mixtures of signals. This problem can be solved using independent component analysis (ICA) technique.

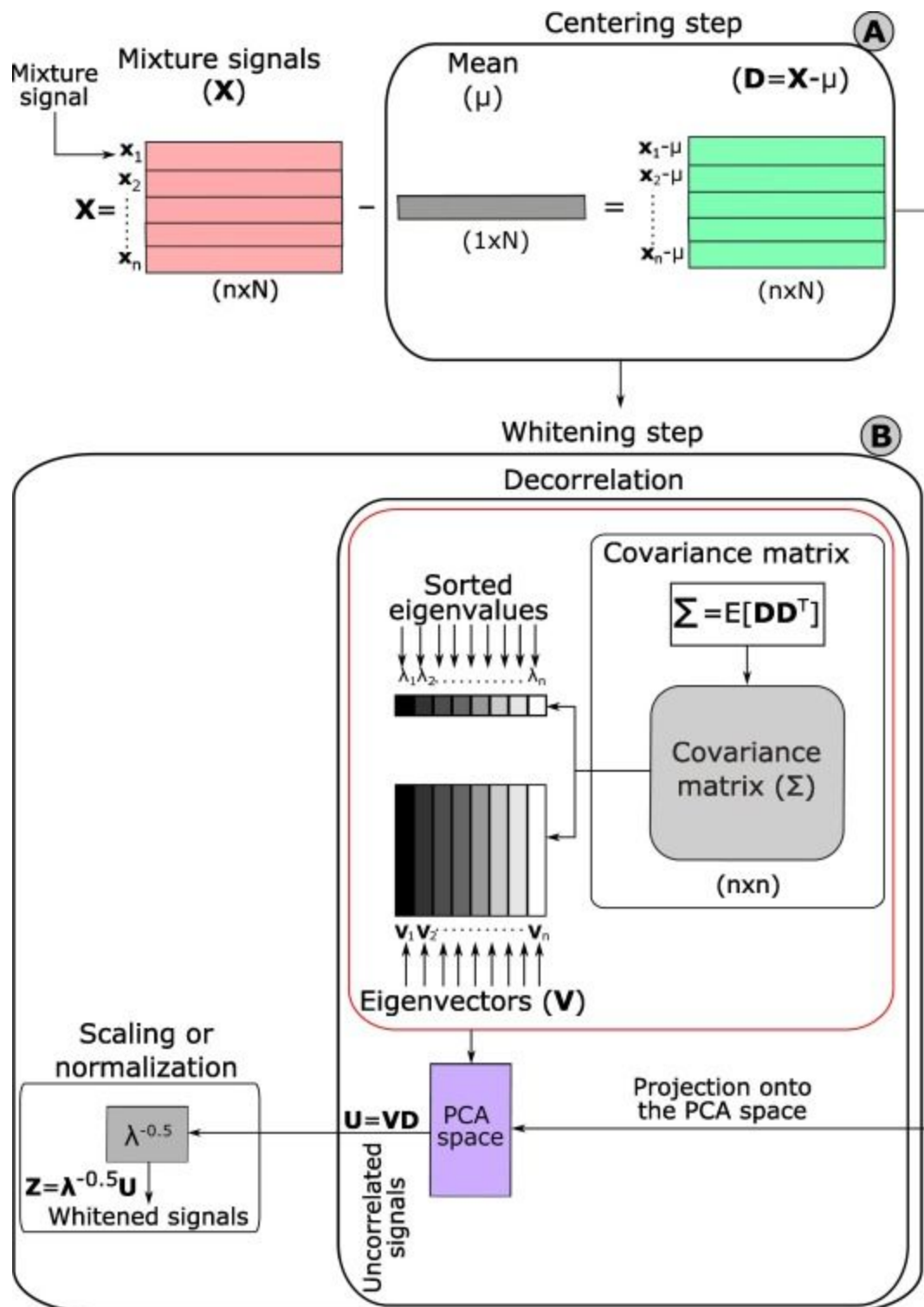


Preprocessing phase

In the preprocessing phase, we have two main steps: centering and whitening.

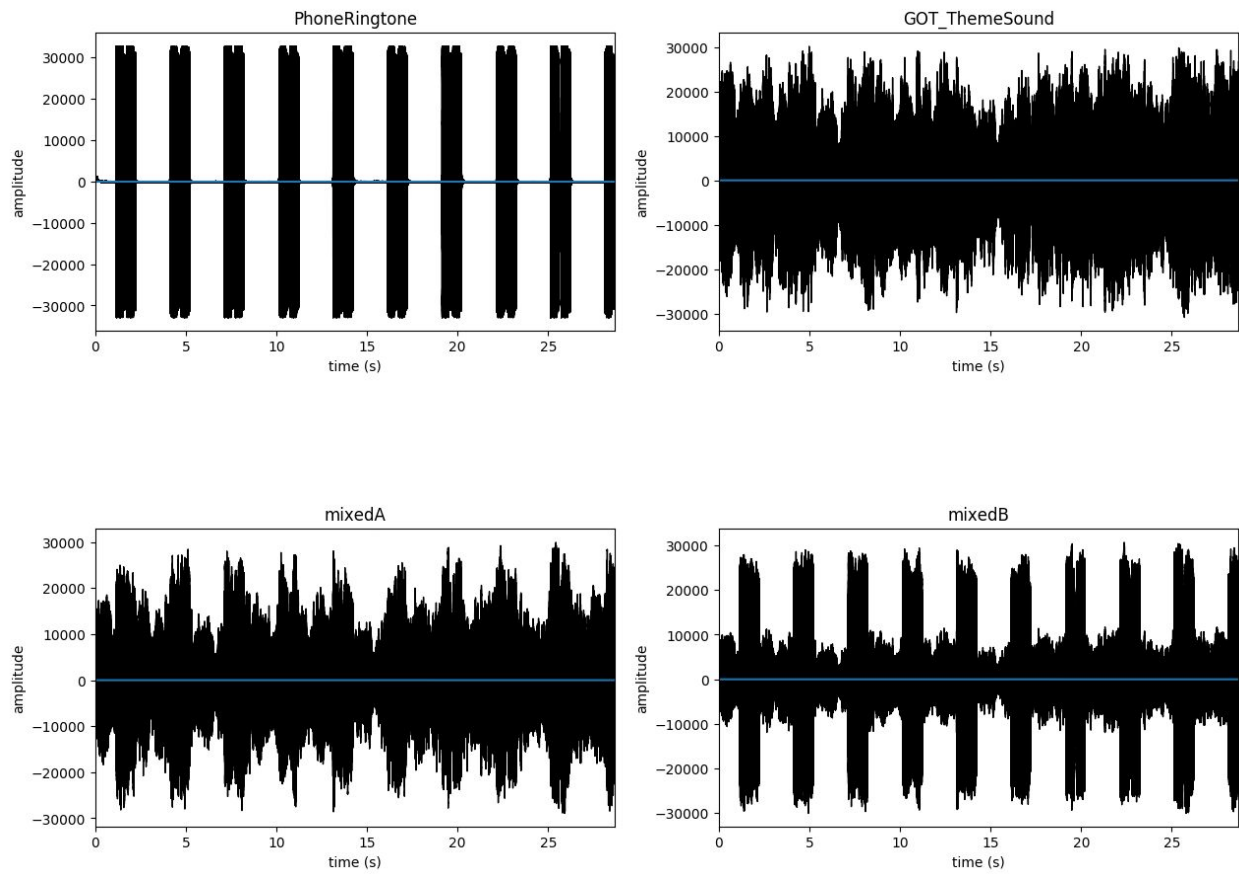
- A. **The Centering Step:** The goal of this step is to center the data by subtracting the mean from all signals. The mean vector can be added back to independent components after applying ICA.

- B. **The Whitening Step:** This step aims to whiten the data which means transforming signals into uncorrelated signals and then rescale each signal to be with unit variance. For decorrelating signals, we make each signal uncorrelated with each other. Two random variables are considered uncorrelated if their covariance is zero. In ICA, the PCA technique can be used for decorrelating signals.



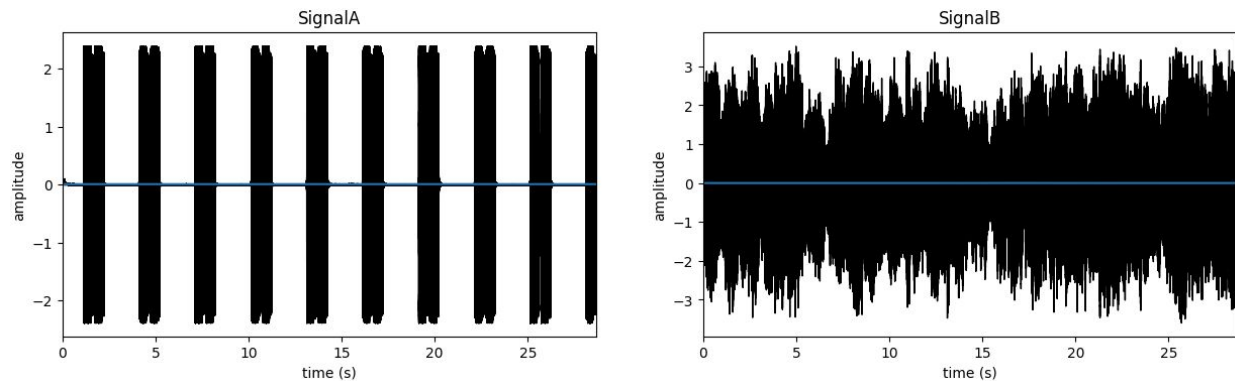
Data

Phone ringtone and GOT (TV series) theme sound are used as original sound waves. These signals are mixed in 0.3,0.7 and 0.6,0.4 proportion to obtain two mixture signals.



Results

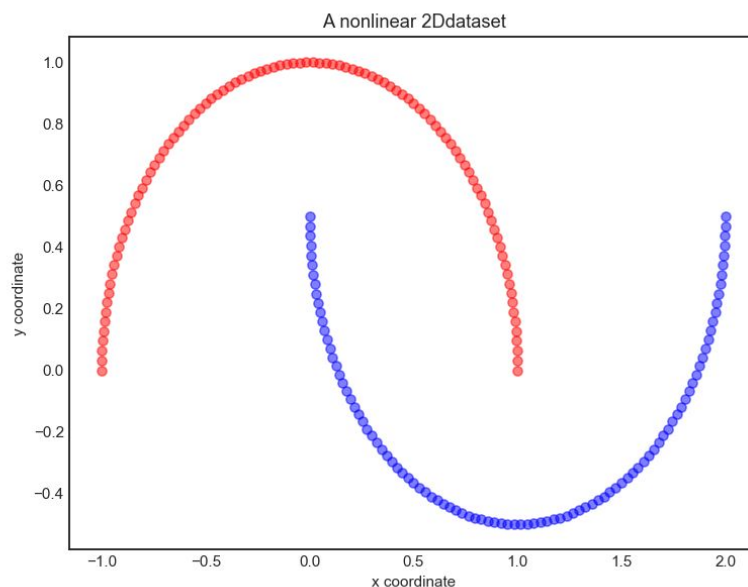
We then apply the ICA algorithm to estimate the original data. Following plot represents the estimated signals.



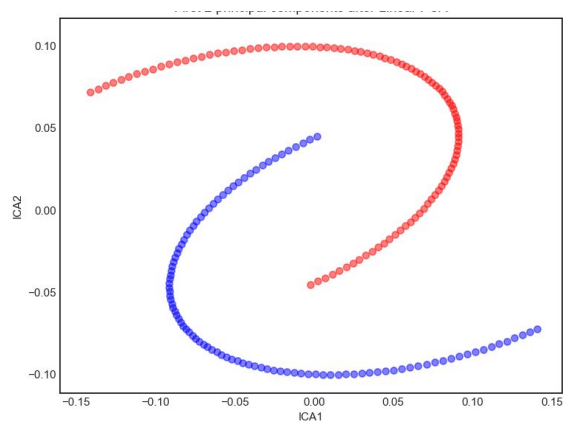
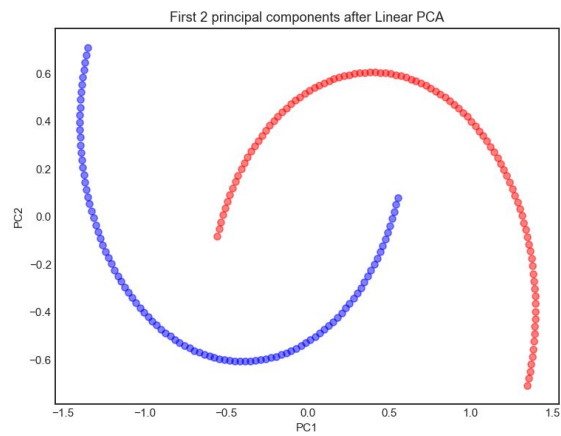
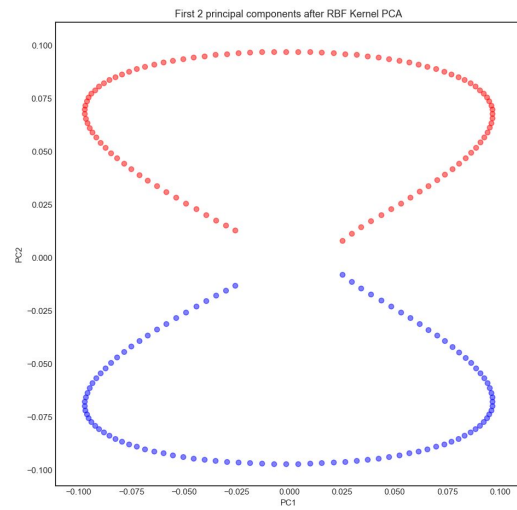
Application of Kernel PCA (RBF Kernel) and FastICA

Facial Recognition - Kernel PCA vs ICA vs PCA

Let us consider a simple case at first where we have a 2D data point that is linearly inseparable which looks like the following.

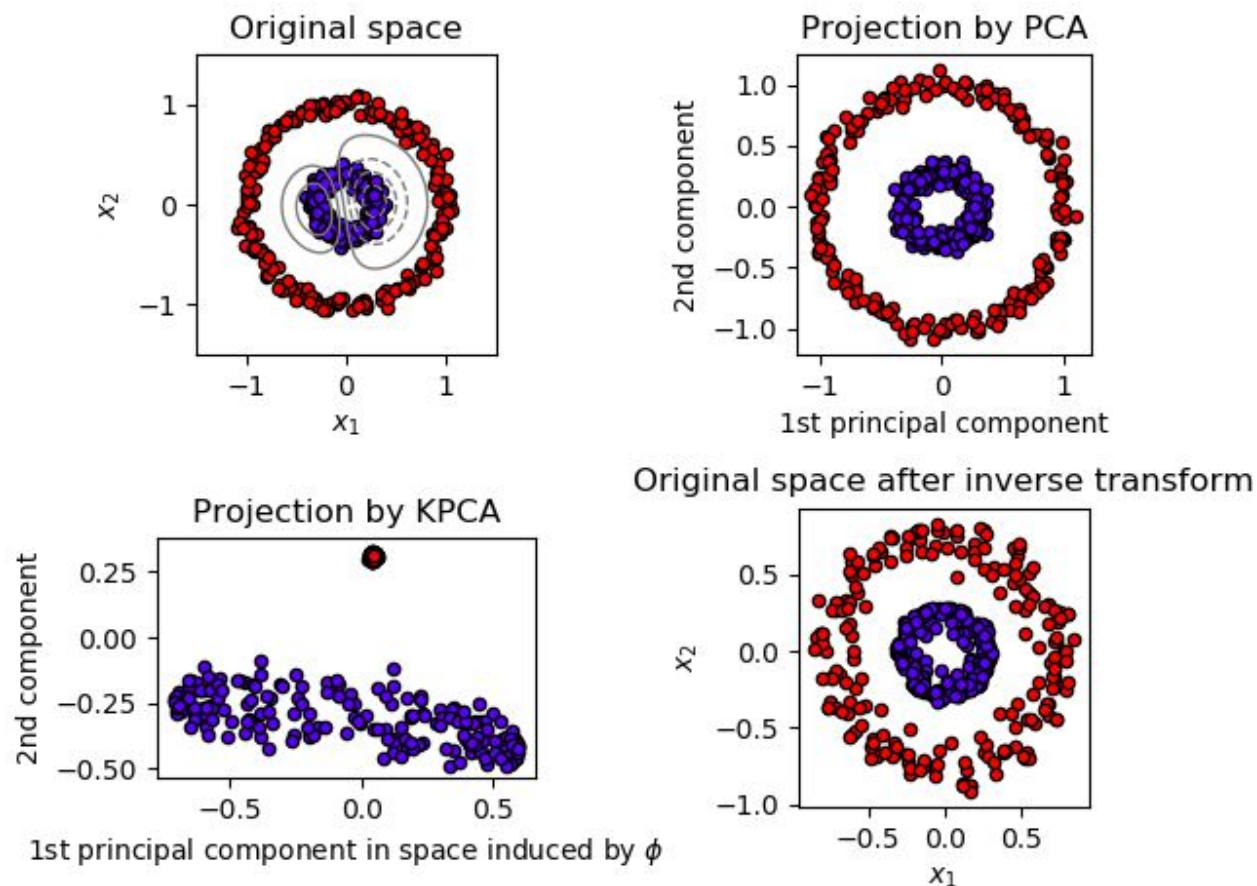


Let's project our data into two dimensional space, using each KPCA, PCA and ICA.



The images are of Kernel PCA, Linear PCA and ICA in that order.

We can see that Kernel PCA transforms the data set into something that is linearly separable.



After using the Inverse transformation we get back the original space in KPCA as shown above figure. But It won't be exact.

Data Set

We use data from LFW datasets. Labeled Faces in the Wild is a database of face photographs designed for studying the problem of unconstrained face recognition. The data set contains more than 13,000 images of faces collected from the web. Each face has been labeled with the name of the person pictured. 1680 of the people pictured have two or more distinct photos in the data set. The only constraint on these faces is that they were detected by the Viola-Jones face detector. We consider only seven faces in our experiment.

Implementation

We are using Kernel Principal Component Analysis followed by SVM.

As we are using the number of pixels as features, it will hard work with that huge number of features. So we use Kernel PCA to reduce the dimension of the features and

the principal axes. Which will decrease the computation and space. Assuming the reduced dimension with covariance matrix is non-linear, we used Kernel PCA.

The implementation is very similar to what we have done earlier, here we do some preprocessing to the image data and apply SVM for classification.

Here we have taken data set from sklearn datasets which contain

n_samples: 1288

n_features: 1850

n_classes: 7

Names of the classes:

1. Ariel Sharon
2. Colin Powell
3. Donald Rumsfeld
4. George W Bush
5. Gerhard Schroeder
6. Hugo Chavez
7. Tony Blair

- Firstly load the dataset and keep in the format no.of samplesXn_features.
- Normalize the data to eliminate the features that are common to all the images.
- Split the data into train and test
- Apply Kernel PCA on the normalized data

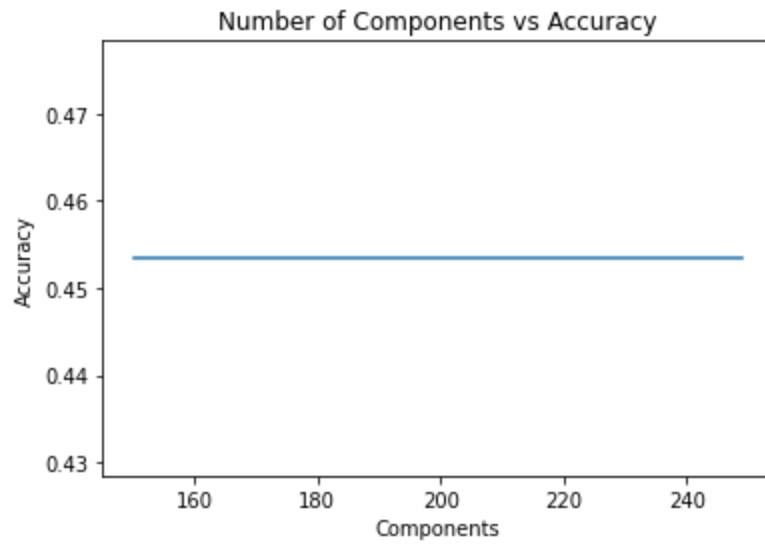
Here again we form a kernel matrix using the RBF function of size no.of_train X no.of_train. Then center the matrix before calculating the eigen_vector and eigen_value of the kernel matrix and choose the K-Best Eigenvectors which form our Eigenface. Which has the size of no_of_train X K.

- Use the resultant matrix and train the SVM model.
- For testing form a kernel matrix from test and train data using RBF kernel then center the matrix. Then multiply the resultant matrix with the Eigenface and predict the class by using the trained SVM model.

Results

Here we have used RBF kernel for Kernel PCA

For gamma = 0.01



There was no change in the accuracy; it was stagnant at 45.34% and everything was classified as one image, in our case it was always classified as George W Bush.

The confusion matrix was as follows:

```
[[ 0  0  0 13  0  0  0]
 [ 0  0  0 60  0  0  0]
 [ 0  0  0 27  0  0  0]
 [ 0  0  0 146 0  0  0]
 [ 0  0  0 25  0  0  0]
 [ 0  0  0 15  0  0  0]
 [ 0  0  0 36  0  0  0]]
```

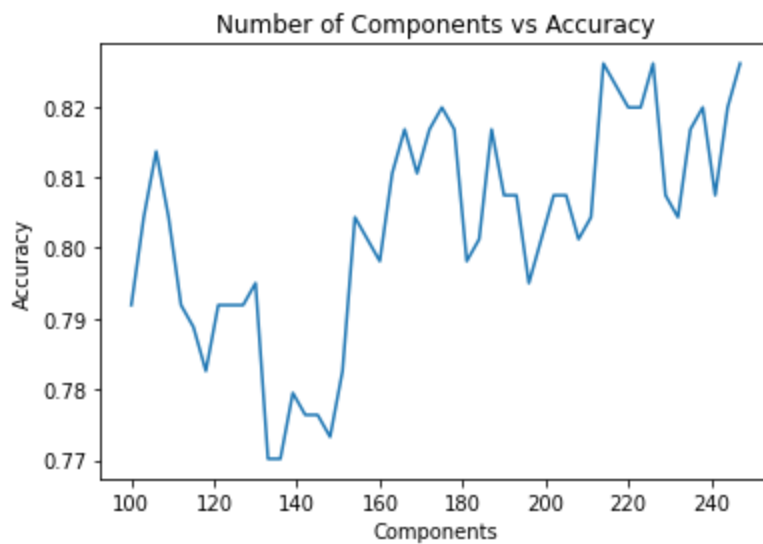
The results were the following:

	<u>precision</u>	<u>recall</u>	<u>f1-score</u>	<u>support</u>
<u>Ariel Sharon</u>	0.00	0.00	0.00	13
<u>Colin Powell</u>	0.00	0.00	0.00	60
<u>Donald Rumsfeld</u>	0.00	0.00	0.00	27
<u>George W Bush</u>	0.45	1.00	0.62	146
<u>Gerhard Schroeder</u>	0.00	0.00	0.00	25
<u>Hugo Chavez</u>	0.00	0.00	0.00	15
<u>Tony Blair</u>	0.00	0.00	0.00	36
accuracy			0.45	322
macro avg	0.06	0.14	0.09	322
weighted avg	0.21	0.45	0.28	322

For gamma = 0.001

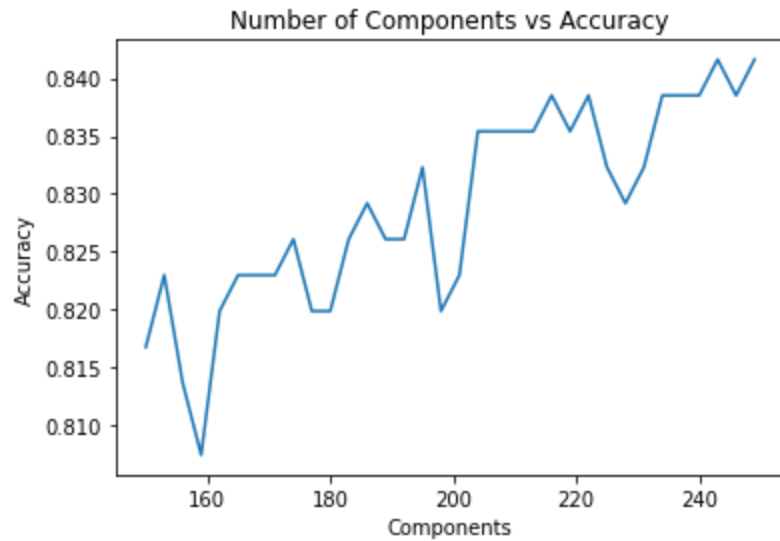
We can observe maximum accuracy of 76.7% for 210 no. of components

For gamma = 0.0001



We can observe maximum accuracy of 82.6% for no. of components 218, 226 and 246.

For gamma = = 0.00001

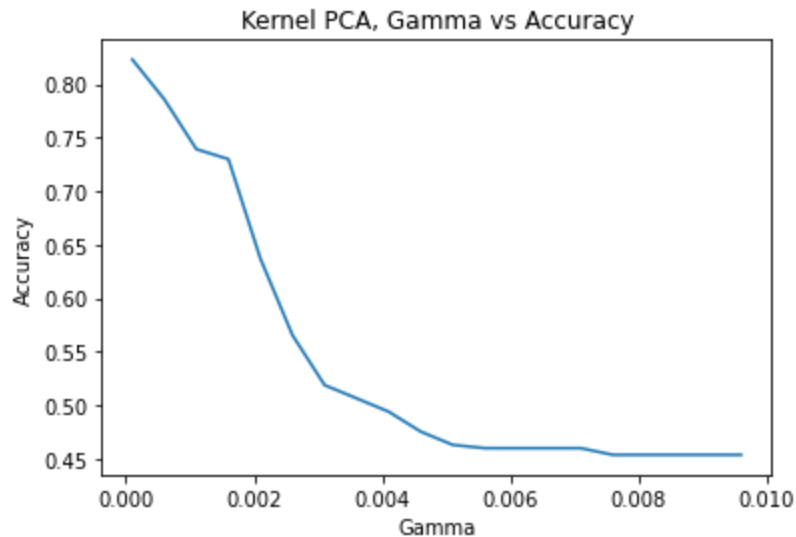


We can observe maximum accuracy of 84.1% for no. of components 246.

Observation

- Gamma plays an important role in kernel PCA, as it very much influences the accuracy of classification.
- As we decreased the gamma the accuracy increased and also the number of components.
- Unfortunately, there is no universal value for the tuning gamma parameter, that works well for different datasets. Finding a value that is appropriate for a given problem requires experimentation.

We now change gamma values and check which value gives us the best accuracy. By running the code we get.



Gamma V Accuracy for 225 components

We can see that as we increase gamma the accuracy reduces.

The starting confusion matrix and its results were as follows

For 0.0001 gamma, accuracy is 0.8229813664596274 confusion matrix is:

```
[[ 9  1  3  0  0  0  0]
 [ 3 52  0  3  0  1  1]
 [ 3  1 18  3  1  0  1]
 [ 2  6  3 129  3  1  2]
 [ 0  1  0  0 20  1  3]
 [ 0  3  0  1  1 10  0]
 [ 1  2  2  2  2  0 27]]
```

	precision	recall	f1-score	support
Ariel Sharon	0.50	0.69	0.58	13
Colin Powell	0.79	0.87	0.83	60
Donald Rumsfeld	0.69	0.67	0.68	27
George W Bush	0.93	0.88	0.91	146
Gerhard Schroeder	0.74	0.80	0.77	25
Hugo Chavez	0.77	0.67	0.71	15
Tony Blair	0.79	0.75	0.77	36

accuracy			0.82	322
macro avg	0.75	0.76	0.75	322
weighted avg	0.83	0.82	0.83	322

The final values were as follows

For 0.0096 gamma, accuracy is 0.453416149068323 confusion matrix is:

```
[[ 0  0  0 13  0  0  0]
 [ 0  0  0 60  0  0  0]
 [ 0  0  0 27  0  0  0]
 [ 0  0  0 146 0  0  0]
 [ 0  0  0 25  0  0  0]
 [ 0  0  0 15  0  0  0]
 [ 0  0  0 36  0  0  0]]
```

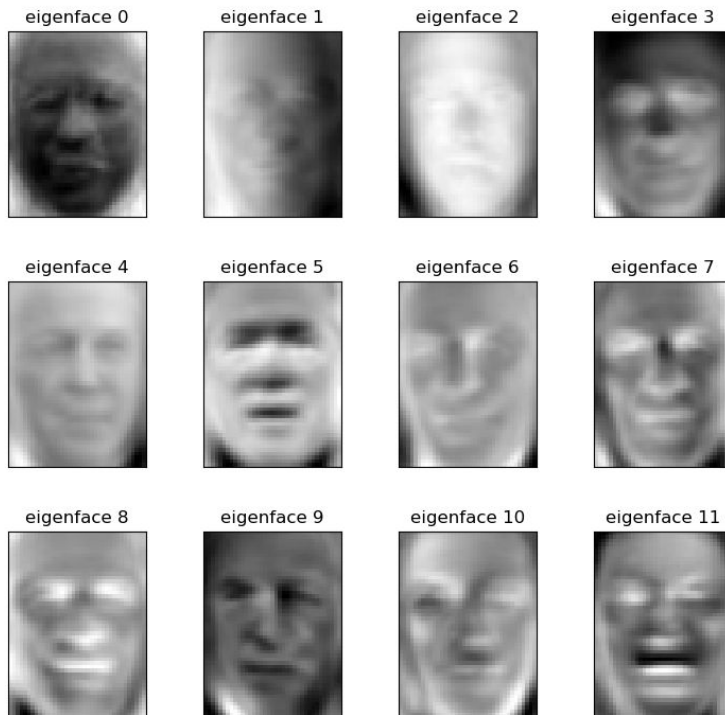
	precision	recall	f1-score	support
Ariel Sharon	0.00	0.00	0.00	13
Colin Powell	0.00	0.00	0.00	60
Donald Rumsfeld	0.00	0.00	0.00	27
George W Bush	0.45	1.00	0.62	146
Gerhard Schroeder	0.00	0.00	0.00	25
Hugo Chavez	0.00	0.00	0.00	15
Tony Blair	0.00	0.00	0.00	36

accuracy			0.45	322
macro avg	0.06	0.14	0.09	322
weighted avg	0.21	0.45	0.28	322

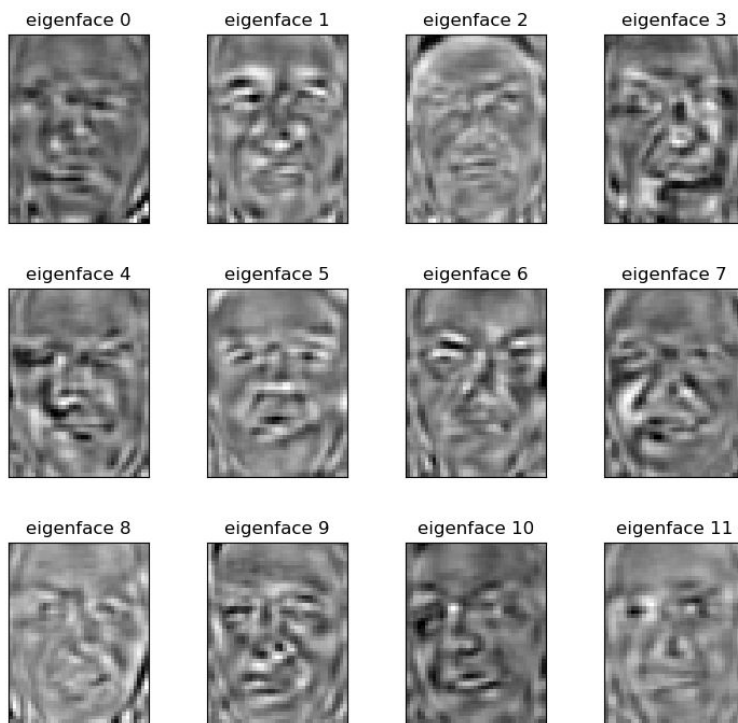
We can see that this is the same as the result before where it would classify all the faces as George W Bush.

EigenFaces

- PCA



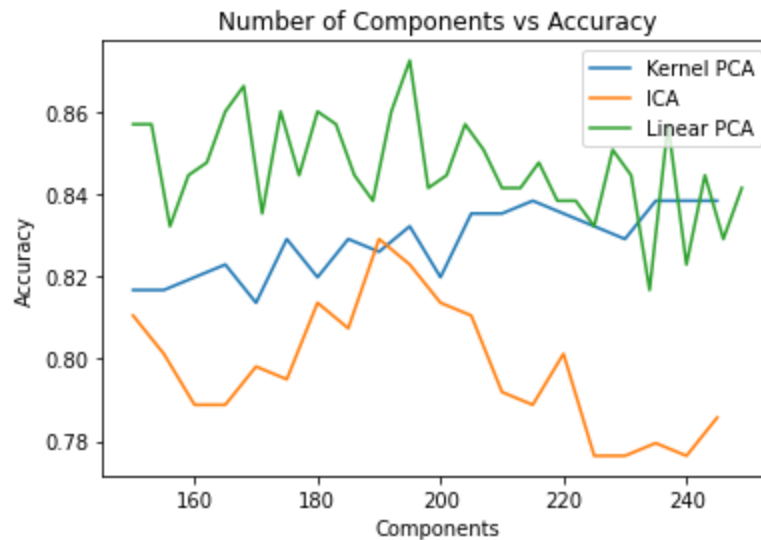
- ICA



The Image does not look as intuitive as PCA eigenfaces. But the structure and relative position of nose, eye's, eyebrows, lips etc; is still maintained.

So we can say PCA works better than ICA, as features are well extracted.

Comparing different methods for different numbers of components.



Conclusion:

1. Linear PCA works best, for the data set we have used, in almost all cases
2. Kernel PCA is least affected by changing the number of components
3. Although all methods give us very good accuracy, we can see that ICA is the least accurate in most cases.