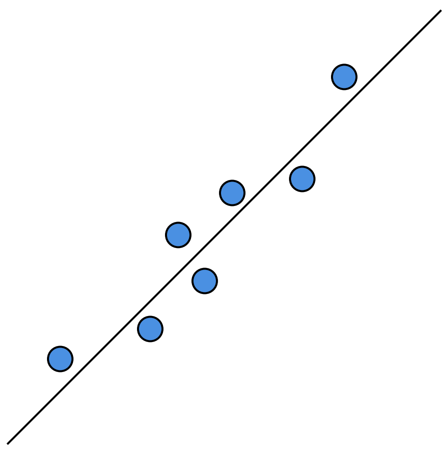
Week-2

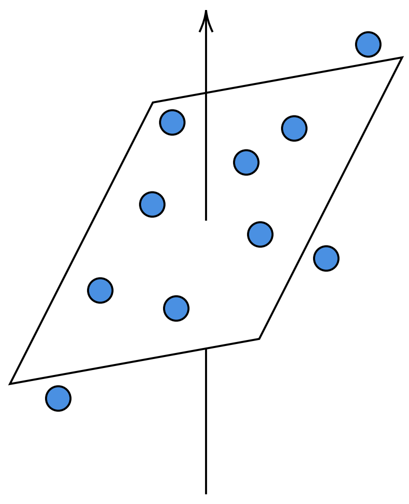
Author: Karthik Thiagarajan

Disclaimer: The notation used in some equations are a bit different from the lectures. The learner is requested to keep this in mind while going through this content.

1. PCA and Linearity

One of the central assumptions in PCA is that the data lies in a linear subspace. The more closely the data clusters around a subspace, the better the results. In $d = 3$, a subspace is a plane or a line passing through the origin. For higher dimensions, a subspace is termed a hyperplane.





For the dataset on the left, one PC would do the job. For the one on the right, two PCs would do the job.

In a real world setting, this scenario would arise when there is a redundancy in the features. This would in turn happen when there is a redundancy in the information being captured. A classic example is the motion of a particle in 3D space along a straight line. Even though a particle moves in a straight line, this information might be recorded using three sensors, one sensor for each direction:

$$\begin{bmatrix} & t=0 & t=1 & t=2 & \cdots & t=T-1 \\ x: & & & & & \\ y: & & & & & \\ z: & & & & & \end{bmatrix}$$

The result is a $3 \times T$ data-matrix. But if we are interested in only the relative position of the particle along the line of motion, we could just do it with one coordinate along the line. This is where PCA would come to our rescue.

Let the particle be traveling at a constant speed u in a direction given by the unit vector $\mathbf{p}$ starting from the origin. In t units, it would have covered ut distance along the vector $\mathbf{p}$. Its position would be $ut\mathbf{p}$. Clearly, we need just a single coordinate to express the relative position of the particle at different time instants.

Expressing this in terms of (x, y, z) coordinates, we have:

$$\begin{aligned} x &= (up_x)t \\ y &= (up_y)t \\ z &= (up_z)t \end{aligned}$$

If we now form the data-matrix $\mathbf{X}$ and perform PCA on it, we will see that the top PC is nothing but the vector $\mathbf{p}$.

2. PCA: Representation Learning

In general, if a dataset mostly lies in a k-dimensional subspace of $\mathbb{R}^d$, then the top-k PCs would be sufficient to represent it. In terms of the amount of information captured, it is:

$$\frac{\lambda_1 + \cdots + \lambda_k}{\lambda_1 + \cdots + \lambda_d}$$

If we are only interested in a representation and not in a full reconstruction, then we can represent the dataset using k coordinates instead of d. Each coordinate would correspond to the scalar projection of the data-point onto one of the principal components.

Let us add the first k PCs as the columns of a matrix $\mathbf{W}$:

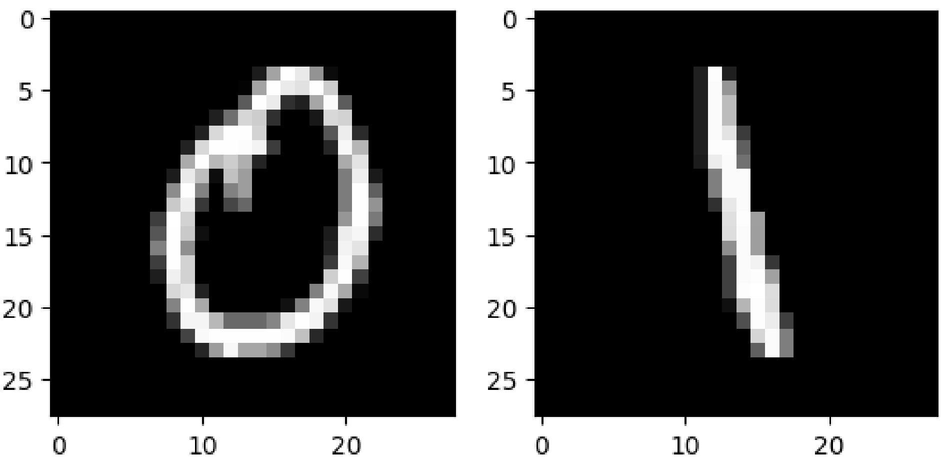
$$\mathbf{W} = \begin{bmatrix} | & & | \\ \mathbf{w}_1 & \cdots & \mathbf{w}_k \\ | & & | \end{bmatrix}, \mathbf{X} = \begin{bmatrix} | & & | \\ \mathbf{x}_1 & \cdots & \mathbf{x}_n \\ | & & | \end{bmatrix}$$

Then the projected dataset is:

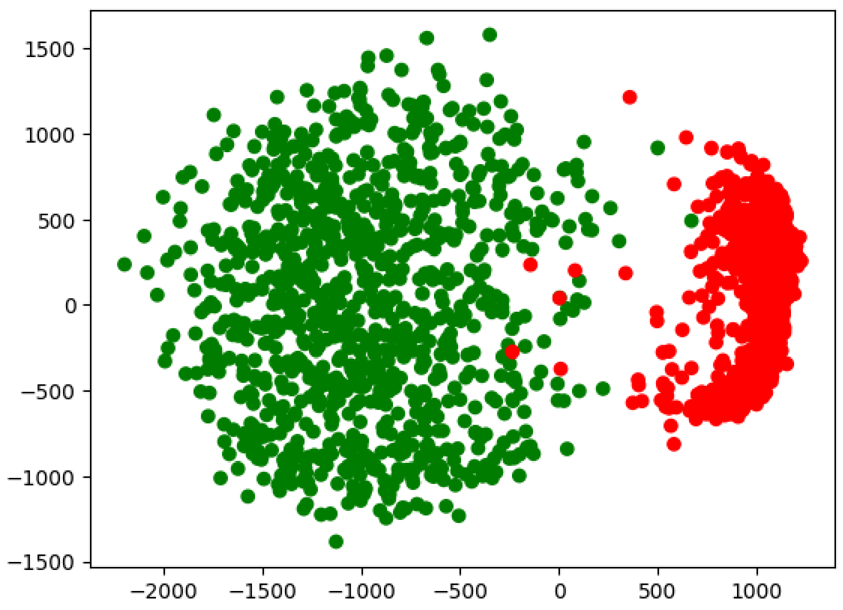
$$\mathbf{X}' = \mathbf{W}^T \mathbf{X}$$

This is a $k \times n$ matrix and contains the scalar projections of the dataset onto the top k PCs. Each column corresponds to one data-point. We can now work with this transformed data-matrix $\mathbf{X}'$ for downstream tasks such as classification and regression.

As an example, assume that we want to classify images of handwritten digits into two classes, 0 and 1. The feature dimension here is 784, which is obtained by arranging the pixels in this 28×28 image in a row or column.



There are 2000 images, 1000 in each class. Running PCA on this dataset and retaining the scalar projections on the top two PCs results in this scatter plot for the data:

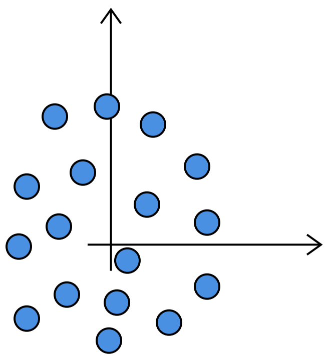


Green corresponds to one of the two classes and red to the other. Notice how the first two PCs have managed to separate the classes quite well. We can now run a standard classification algorithm on this transformed dataset.

3. Problems with standard PCA

3.1. Non-linearity

If a dataset violates this assumption, which is mostly the case in the real world, we can still run PCA, but the gains would be minimal. The rule of thumb is to choose the value of k that captures 95% of the variance. A dataset that violates this assumption would have a high value of k. An example of this is the following:



For a dataset such as the one shown here, notice that the variance is roughly the same along any direction. Any two orthonormal directions would be good choices for the first two PCs. We would need two PCs to represent the data. But we started off with two coordinates to represent each point. There is therefore no gain by performing PCA on such a dataset.

3.2. Computational Cost

The bottleneck in computing the PCs is the eigen-decomposition algorithm used to extract the eigenvectors and the eigenvalues of the covariance matrix. For a $d \times d$ matrix, the number of steps is $O\big(d^3\big)$. That is, if the number of features doubles, the time taken would roughly become eight times. Thus PCA on datasets with huge number of dimensions poses a computational challenge.

4. $\mathbf{XX}^T$ and $\mathbf{X}^T\mathbf{X}$

We can first fix the issue with the cost of doing PCA on datasets with a huge number of features. For such datasets, we have $d \gg n$. What we need are the eigenvectors of the matrix $\mathbf{XX}^T$. Let us now turn our attention to a closely related matrix, $\mathbf{X}^T\mathbf{X}$. Let $(\lambda, \mathbf{v})$ be an eigenpair of $\mathbf{X}^T\mathbf{X}$ with $\lambda \neq 0$:

$$\big(\mathbf{X}^T\mathbf{X}\big)\mathbf{v} = \lambda\mathbf{v}$$

Let us now pre-multiply both sides by $\mathbf{X}$:

$$\begin{aligned}\mathbf{X}\big(\mathbf{X}^T\mathbf{X}\big)\mathbf{v} &= \lambda\mathbf{X}\mathbf{v}\\ \big(\mathbf{XX}^T\big)\mathbf{X}\mathbf{v} &= \lambda(\mathbf{X}\mathbf{v})\end{aligned}$$

We see that $(\lambda, \mathbf{Xv})$ is an eigenpair of $\mathbf{XX}^T$ provided $\mathbf{Xv} \neq \mathbf{0}$. This is indeed the case. As an exercise, show that if $\lambda \neq 0$ then $\mathbf{Xv} \neq 0$. What this demonstrates is the following fact:

$\mathbf{X}^T\mathbf{X}$ and $\mathbf{XX}^T$ have the same non-zero eigenvalues. Further, if $(\lambda, \mathbf{v})$ is an eigenpair of $\mathbf{X}^T\mathbf{X}$, then $\mathbf{Xv}$ is an eigenvector of $\mathbf{XX}^T$, provided $\lambda \neq 0$.

This is good news for us. The matrix $\mathbf{X}^T\mathbf{X}$ has shape $n \times n$. Since

$n \ll d$, performing eigen-decomposition on this matrix is computationally more efficient compared to the same operation on $\mathbf{X}\mathbf{X}^T$. Besides, if know the eigenvector of $\mathbf{X}^T\mathbf{X}$, that easily leads us to the eigenvector of $\mathbf{X}\mathbf{X}^T$.

What remains now is to tie up some loose ends. Recall that the PCs are unit vectors. Let us assume that the eigen-decomposition algorithm returns the following eigenpairs for $\mathbf{X}^T\mathbf{X}$:

$$(\lambda_1, \mathbf{v}_1),\ \cdots, (\lambda_r, \mathbf{v}_r)$$

where $||\mathbf{v}_i|| = 1$ and $\lambda_1 \geqslant \cdots \geqslant \lambda_r > 0$. Here r is the rank of the matrix $\mathbf{X}$, a good fact to keep in mind, but not terribly important. We can now get the eigenpairs of $\mathbf{X}\mathbf{X}^T$ as:

$$(\lambda_1, \mathbf{X}\mathbf{v}_1),\ \cdots, (\lambda_r, \mathbf{X}\mathbf{v}_r)$$

To normalize the vectors, we see that:

$$\begin{aligned} ||\mathbf{X}\mathbf{v}_i||^2 &= \mathbf{v}_i^T\mathbf{X}^T\mathbf{X}\mathbf{v}_i \\ &= \mathbf{v}_i^T\left(\mathbf{X}^T\mathbf{X}\right)\mathbf{v}_i \\ &= \lambda_i\mathbf{v}_i^T\mathbf{v}_i \\ &= \lambda_i \end{aligned}$$

Upon normalization, the eigenpairs become:

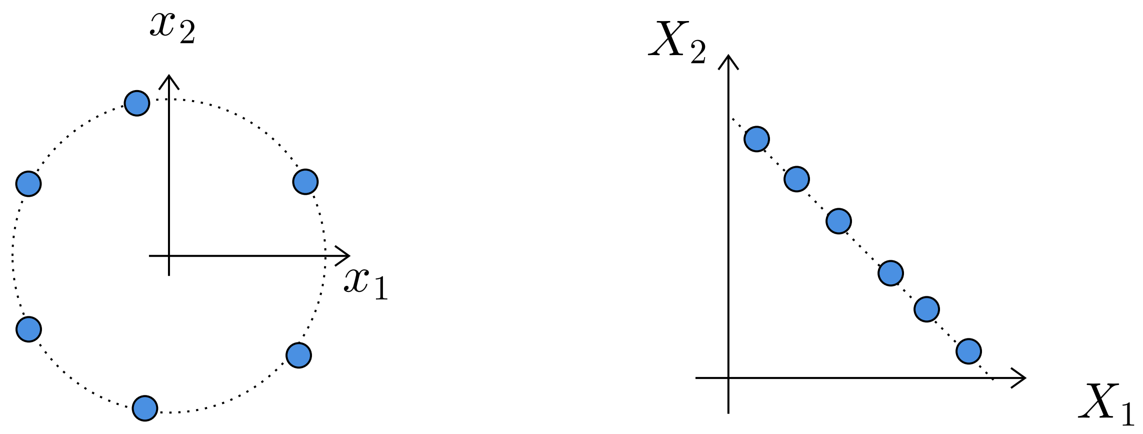
$$\left(\lambda_1, \frac{\mathbf{X}\mathbf{v}_1}{\sqrt{\lambda_1}}\right),\ \cdots, \left(\lambda_r, \frac{\mathbf{X}\mathbf{v}_r}{\sqrt{\lambda_r}}\right)$$

These are the eigenpairs of $\mathbf{X}\mathbf{X}^T$. To get the eigenpairs of the covariance matrix $\mathbf{C} = \dfrac{1}{n}\mathbf{X}\mathbf{X}^T$, we retain the eigenvectors as they are but scale the eigenvalues by $\dfrac{1}{n}$:

$$\left(\frac{\lambda_1}{n}, \frac{\mathbf{X}\mathbf{v}_1}{\sqrt{\lambda_1}}\right), \cdots, \left(\frac{\lambda_r}{n}, \frac{\mathbf{X}\mathbf{v}_r}{\sqrt{\lambda_r}}\right)$$

5. Feature Transformation

Let us now turn our attention to the problem of non-linearity. The strength of PCA lies in the linearity of the dataset. If we can use the existing features to come up with a new set of features in which the dataset is linear, then we can apply PCA on this transformed dataset. Here is an example to show that linearization is possible:



$$X_1 = x_1^2$$
$$X_2 = x_2^2$$

Introducing non-linear features helps in linearization. In this example, just including x_1^2 and x_2^2 helped, but in general, we may not know what to include and what to leave. So a more complete non-linear feature transformation would look like this:

$$\phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1 x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix}$$

This is a polynomial feature transformation of degree 2 as the maximum degree of any component is 2. Notice how it includes all terms of degree

0 and 1 as well. This transformation is a mapping from the feature space $\mathbb{R}^2$ to $\mathbb{R}^6$. In general, a non-linear feature transformation ϕ maps from $\mathbb{R}^d$ to $\mathbb{R}^D$. Our hope is that such a transformation linearizes the dataset in this process. Therefore, a non-linear dataset in $\mathbb{R}^d$ may turn into a linear dataset in $\mathbb{R}^D$ with the help of ϕ.

6. Kernel PCA: Introduction

One way to deal with non-linearity is to transform the features using a non-linear feature transformation ϕ and then apply PCA on the transformed dataset. Let us proceed with this idea. Let $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^D$ be a non-linear transformation. Then abusing notation, we have:

$$\phi(\mathbf{X}) = \begin{bmatrix} | & & | \\ \phi(\mathbf{x}_1) & \cdots & \phi(\mathbf{x}_n) \\ | & & | \end{bmatrix}$$

$\phi(\mathbf{X})$ is the data-matrix in the transformed feature space and is of dimensions $D \times n$. For now, assume that $\phi(\mathbf{X})$ is centered, then the covariance matrix now becomes:

$$\mathbf{C} = \frac{1}{n}\phi(\mathbf{X})\phi(\mathbf{X})^T$$

We can now proceed with PCA. However, note that D could turn out to be very large. Even if we have a small number of features to begin with, a polynomial transformation like the one we saw before would result in a large value of D. Thankfully, we have just seen a workaround for this. We can work with $\phi(\mathbf{X})^T\phi(\mathbf{X})$ instead. This matrix will turn out to be important, hence we are going to use a spacial symbol for it:

$$\mathbf{K} = \phi(\mathbf{X})^T\phi(\mathbf{X})$$

$\mathbf{K}$ is a $n \times n$ matrix. If $\mathbf{v}_1, \cdots \mathbf{v}_r$ are the eigenvectors corresponding to the non-zero eigenvalues of $\mathbf{K}$, then the eigenpairs of $\mathbf{C}$ are:

$$\left(\frac{\lambda_1}{n}, \frac{\phi(\mathbf{X})\mathbf{v}_1}{\sqrt{\lambda_1}}\right), \cdots, \left(\frac{\lambda_r}{n}, \frac{\phi(\mathbf{X})\mathbf{v}_r}{\sqrt{\lambda_r}}\right)$$

Now that we have meddled with the original feature space, it is sufficient to focus only on the scalar representations. The data-matrix after PCA in the transformed space is:

$$\mathbf{X}' = \begin{bmatrix} \phi(\mathbf{x}_1)^T\phi(\mathbf{X})\mathbf{v}_1 & & \phi(\mathbf{x}_n)^T\phi(\mathbf{X})\mathbf{v}_1 \\ \vdots & \cdots & \vdots \\ \phi(\mathbf{x}_1)^T\phi(\mathbf{X})\mathbf{v}_r & & \phi(\mathbf{x}_n)^T\phi(\mathbf{X})\mathbf{v}_r \end{bmatrix}$$

This is of shape $r \times n$.

The one problem that still remains is computing $\phi(\mathbf{X})$. This itself can be quite costly. As a simple example, if $d = 100$, a polynomial transformation of degree 2 would yield nearly $10,000$ features. But a very interesting observation regarding all that has happened so far is this: at an
stage in the algorithm, we only need the dot product between pairs of vectors in the transformed space. That is, we only need $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$. If we can find a way to compute this quantity without explicitly computing $\phi(x)$, then we are in good shape.

7. Kernels

A kernel is precisely such a function. A function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is called a kernel if for some $\phi : \mathbb{R}^d \to \mathbb{R}^D$, we have:

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

for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. To see that such functions exist, consider the following function from $\mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$:

$$k(\mathbf{x}, \mathbf{y}) = \left(1 + \mathbf{x}^T\mathbf{y}\right)^2$$

Let us expand:

$$
\begin{aligned}
k\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}\right) &= (1 + x_1y_1 + x_2y_2)^2 \\
&= 1 + x_1^2y_1^2 + x_2^2y_2^2 + 2x_1y_1 + 2x_2y_2 + 2x_1x_2y_1y_2
\end{aligned}
$$

Now, define $\phi : \mathbb{R}^2 \to \mathbb{R}^6$ as:

$$
\phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \sqrt{2}x_1x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix}
$$

We see that:

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

The kernel just defined is called a polynomial kernel of degree 2. In general, we can define polynomial kernels of degree p as:

$$
k(\mathbf{x}, \mathbf{y}) = \left(1 + \mathbf{x}^T \mathbf{y}\right)^p
$$

If the feature space has dimension d, then the transformed space under this kernel will have dimension $\begin{pmatrix} p + d \\ d \end{pmatrix}$.

8. Kernel PCA: Algorithm

We can now go back to the algorithm and complete its description. We were looking at the matrix $\mathbf{K} = \phi(\mathbf{X})^T \phi(\mathbf{X})$. This is called the Gram matrix or the kernel matrix. This is nothing but the matrix whose entries are dot products between the transformed data-points. If k is a kernel corresponding to ϕ, we have:

$$K_{ij} = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j)$$

We can now compute the eigenvectors of $\mathbf{K}$. Let the eigenpairs of $\mathbf{K}$ be:

$$(\lambda_1, \mathbf{v}_1) \cdots , (\lambda_r, \mathbf{v}_r)$$

where $\lambda_1 \geqslant\cdots\geqslant \lambda_r > 0$ and $||\mathbf{v}_i|| = 1$. Using this, we can find the scalar projection of the data-points in the transformed space after PCA. This is demonstrated for one data-point $\mathbf{x}_i$ on the j^{th} PC:

$$\begin{aligned}\phi(\mathbf{x}_i)^T\phi(\mathbf{X})\frac{\mathbf{v}_j}{\sqrt{\lambda_j}} &= \sum_{p=1}^{n}\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_p)\frac{v_{jp}}{\sqrt{\lambda_j}}\\ &= \sum_{p=1}^{n}k(\mathbf{x}_i, \mathbf{x}_p)\frac{v_{jp}}{\sqrt{\lambda_j}}\end{aligned}$$

The scalar projections of all the data-points on the j^{th} PC can be represented as:

$$\frac{\mathbf{K}\mathbf{v}_j}{\sqrt{\lambda_j}}$$

We can do this for the r PCs with the help of the following matrix product:

$$\mathbf{K}\begin{bmatrix} | & & | \\ \mathbf{v}_1 & \cdots & \mathbf{v}_r \\ | & & | \end{bmatrix}\begin{bmatrix} \frac{1}{\sqrt{\lambda_1}} & & \\ & \ddots & \\ & & \frac{1}{\sqrt{\lambda_r}} \end{bmatrix}$$

This is a $n \times r$ matrix. To get the representation in $r \times n$ form, we just transpose this product. Using the following notation:

$$\mathbf{V} = \begin{bmatrix} | & & | \\ \mathbf{v}_1 & \cdots & \mathbf{v}_r \\ | & & | \end{bmatrix}, \mathbf{D} = \begin{bmatrix} \frac{1}{\sqrt{\lambda_1}} & & \\ & \ddots & \\ & & \frac{1}{\sqrt{\lambda_r}} \end{bmatrix}$$

We get the final representation as:

$$\mathbf{X}' = \mathbf{D}\mathbf{V}^T\mathbf{K}$$

This is the kernel-PCA algorithm. Each column of the matrix $\mathbf{X}'$ represents one data-point. This data-point is represented by r coordinates. We can choose to retain the top k of these r coordinates in a downstream
task such as classification or regression.

9. Kernel Centering

So far we have been assuming that the transformed matrix is mean centered. This is clearly not the case in general. So let us see how we can center the transformed matrix. Let ϕ be a transformation:

$$\phi : \mathbb{R}^d \rightarrow \mathbb{R}^D$$

To help us with the centering, consider the following matrix of 1s divided by n, the number of data-points:

$$\mathbf{1}_{n \times n} = \frac{1}{n}\begin{bmatrix} & \vdots & \\ \cdots & 1 & \cdots \\ & \vdots & \end{bmatrix}$$

The centered dataset can be expressed as:

$$\phi_c(\mathbf{X}) = \phi(\mathbf{X}) - \phi(\mathbf{X})\mathbf{1}_{n \times n}$$

The covariance matrix of the transformed dataset becomes:

$$\mathbf{C} = \frac{1}{n}\phi_c(\mathbf{X})\phi_c(\mathbf{X})^T$$

Let k be a kernel corresponding to the transformation ϕ:

$$k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$$
$$k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T\phi(\mathbf{y})$$

The kernel matrix is:

$$\mathbf{K} = \phi(\mathbf{X})^T\phi(\mathbf{X})$$

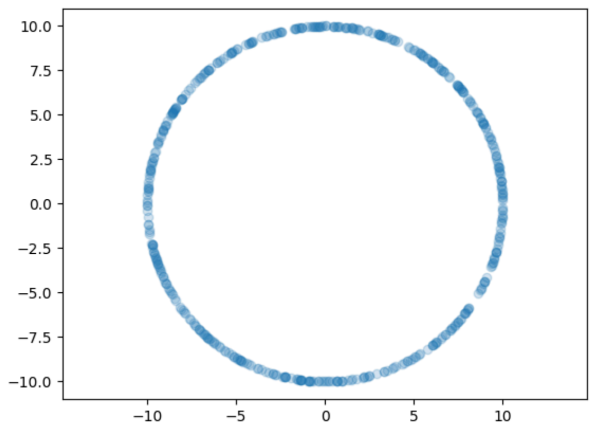
We can now compute the centered kernel in terms of $\mathbf{K}$:

$$\begin{aligned}
\mathbf{K}_c &= \phi_c(\mathbf{X})^T\phi_c(\mathbf{X}) \\
&= [\phi(\mathbf{X}) - \phi(\mathbf{X})\mathbf{1}_{n\times n}]^T[\phi(\mathbf{X}) - \phi(\mathbf{X})\mathbf{1}_{n\times n}] \\
&= \phi(\mathbf{X})^T\phi(\mathbf{X}) - \phi(\mathbf{X})^T\phi(\mathbf{X})\mathbf{1}_{n\times n} \\
&\quad -\mathbf{1}_{n\times n}\phi(\mathbf{X})^T\phi(\mathbf{X}) + \mathbf{1}_{n\times n}\phi(\mathbf{X})^T\phi(\mathbf{X})\mathbf{1}_{n\times n} \\
&= \mathbf{K} - \mathbf{K}\mathbf{1}_{n\times n} - \mathbf{1}_{n\times n}\mathbf{K} + \mathbf{1}_{n\times n}K\mathbf{1}_{n\times n}
\end{aligned}$$

We now replace $\mathbf{K}$ with $\mathbf{K}_c$ in the kernel-PCA algorithm.

10. Example

Consider a dataset in which the data-points lie on a circle with radius 10:



This dataset is clearly non-linear in $\mathbb{R}^2$. Now, consider a polynomial kernel of degree 2. This will correspond to the following feature transformation:

$$\phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} 1 & \sqrt{2}x_1 & \sqrt{2}x_2 & \sqrt{2}x_1x_2 & x_1^2 & x_2^2 \end{bmatrix}^T$$

Consider the following subspace of $\mathbb{R}^6$:

$$S = \left\{\mathbf{x} : \mathbf{w}^T\mathbf{x} = 0\right\}$$

where, $\mathbf{w} = \begin{bmatrix} -100 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}^T$. Note that all the points in the dataset lie in this subspace. This subspace is nothing but the set of all vectors perpendicular to $\mathbf{w}$. The dimension of S is 5. We see that this transformation linearizes the dataset in $\mathbb{R}^6$. If PCA is run on this transformed dataset, the top five PCs would be able to completely reconstruct the data. Therefore, there would be at most five non-zero eigenvalues, as there would be no variance along the sixth direction.