Neural Networks for Signal Processing-I

Prof. Shayan Srinivasa Garani

Department of Electronic System Engineering

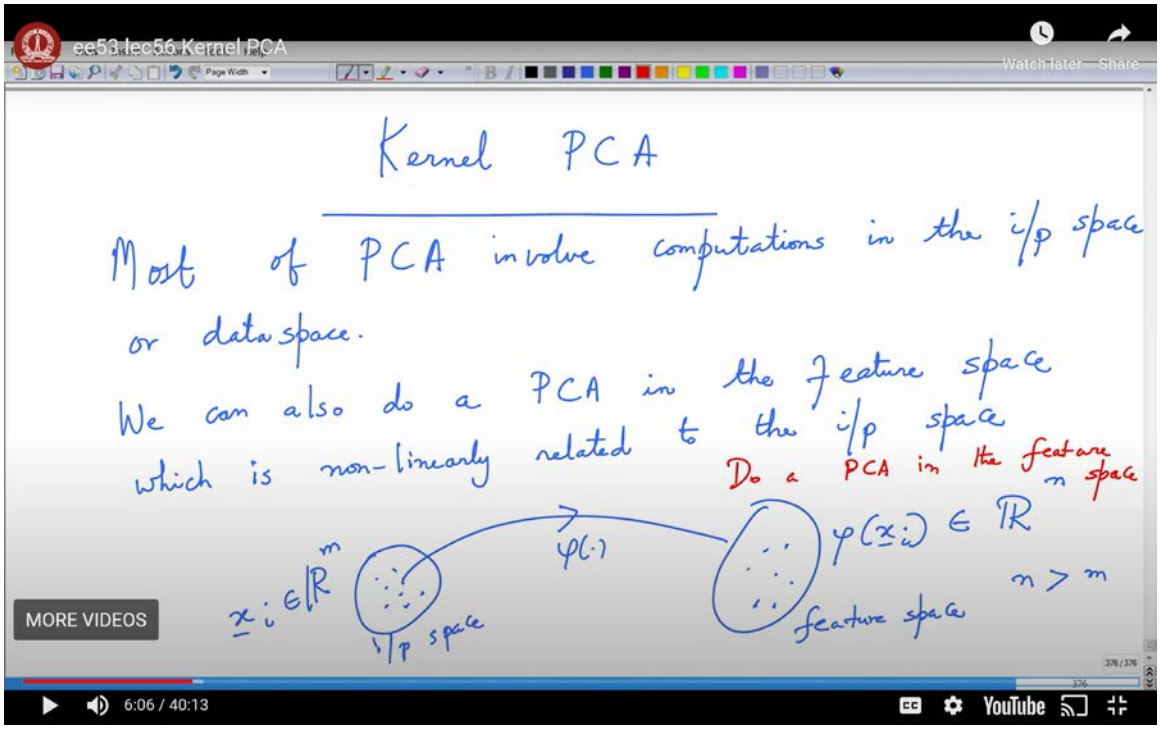
Indian Institute of Science, Bengaluru

Lecture – 56

Kernel PCA

Principal Component Analysis (PCA) is a fundamental tool widely used in data science applications. It is also known as the Karhunen-Loeve (KL) transform. The goal of PCA is to identify the principal directions, or eigen directions, in the data space. This is achieved by using an objective function that minimizes the energy of the components, effectively maximizing the variance captured by the first k components in the eigen space.

(Refer Slide Time: 06:06)



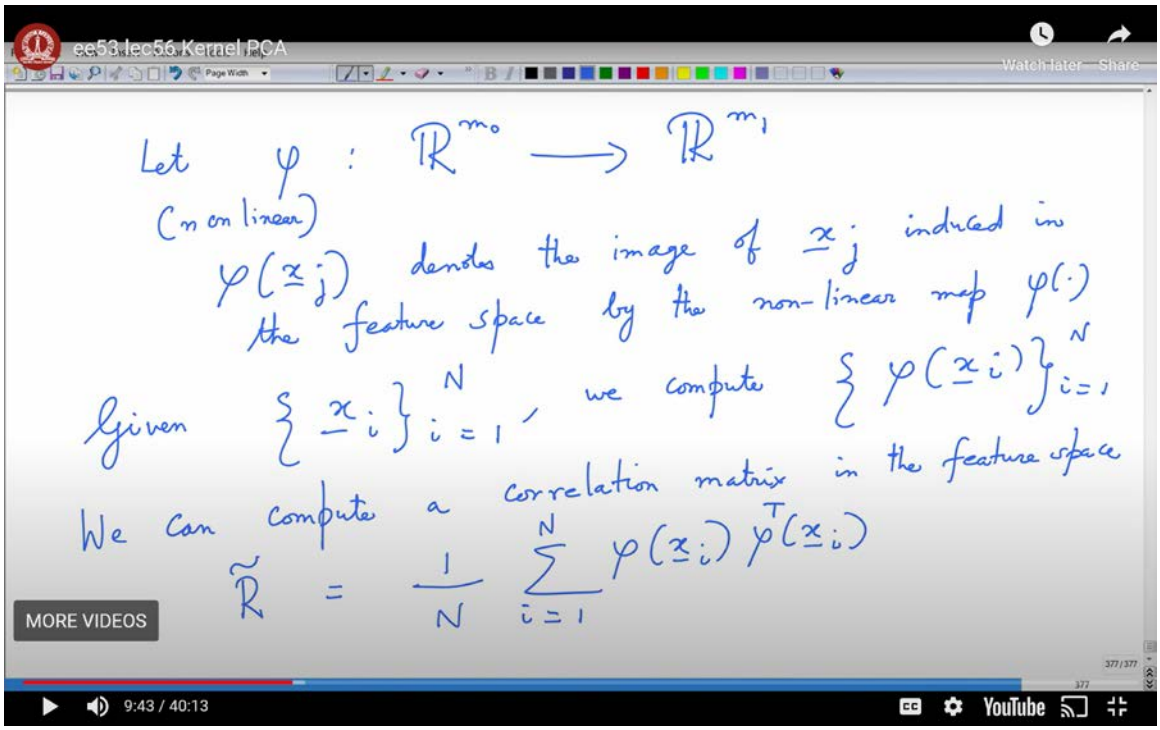
To perform PCA, we start by analyzing the data samples, constructing a covariance matrix from these samples, and then performing an eigen decomposition. This process yields the

eigenvalues and eigenvectors of the data. These eigenvectors define a new basis for the data space, allowing any vector in this space to be expressed in terms of this eigenbasis. By considering energy constraints, we can choose to retain only the most significant components while discarding others, resulting in a more compact representation of the data.

For an in-depth discussion on the derivation and details of the PCA algorithm, refer to the lectures in the "Mathematical Methods and Techniques for Signal Processing" course, particularly lectures 75, 76, and 77.

In this lecture, we will extend the discussion on PCA by incorporating kernels, based on Mercer's theorem. This extension involves applying PCA in a feature space, which is obtained through a non-linear transformation of the input space. This leads us to the concept of Kernel PCA, where PCA is performed in the transformed feature space.

(Refer Slide Time: 09:43)

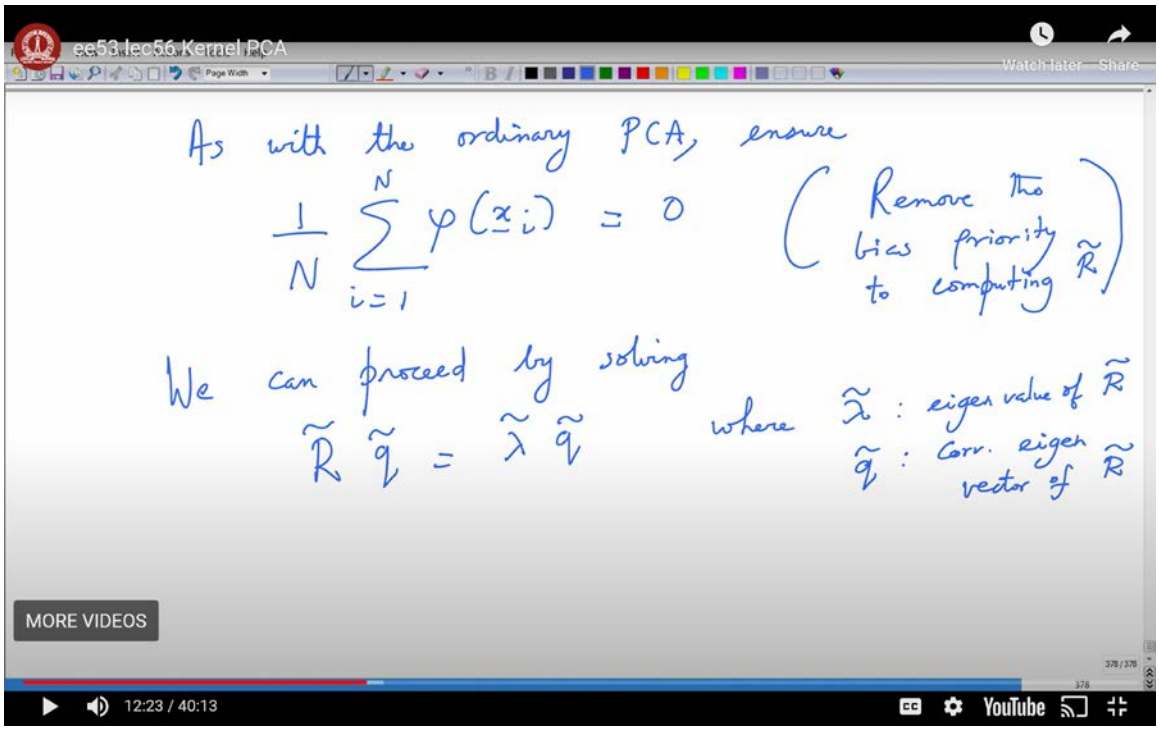


Typically, PCA is computed in the input space (data space). However, we can also apply PCA in the feature space, which is non-linearly related to the input space. To illustrate this, consider a point x_i in the input space, where x_i belongs to R^m. Through a non-linear

transformation, x_i is mapped to the feature space, denoted as $\varphi(x_i)$, which belongs to R^n where n is greater than m. PCA is then performed in this feature space.

We must be aware that in the feature space, we work with inner product kernels due to Mercer's theorem. Let φ denote the transformation from R^{m_0} to R^{m_1}. I have updated the notation here to ensure consistency, where m is replaced with m_0 and n is replaced with m_1.

(Refer Slide Time: 12:23)



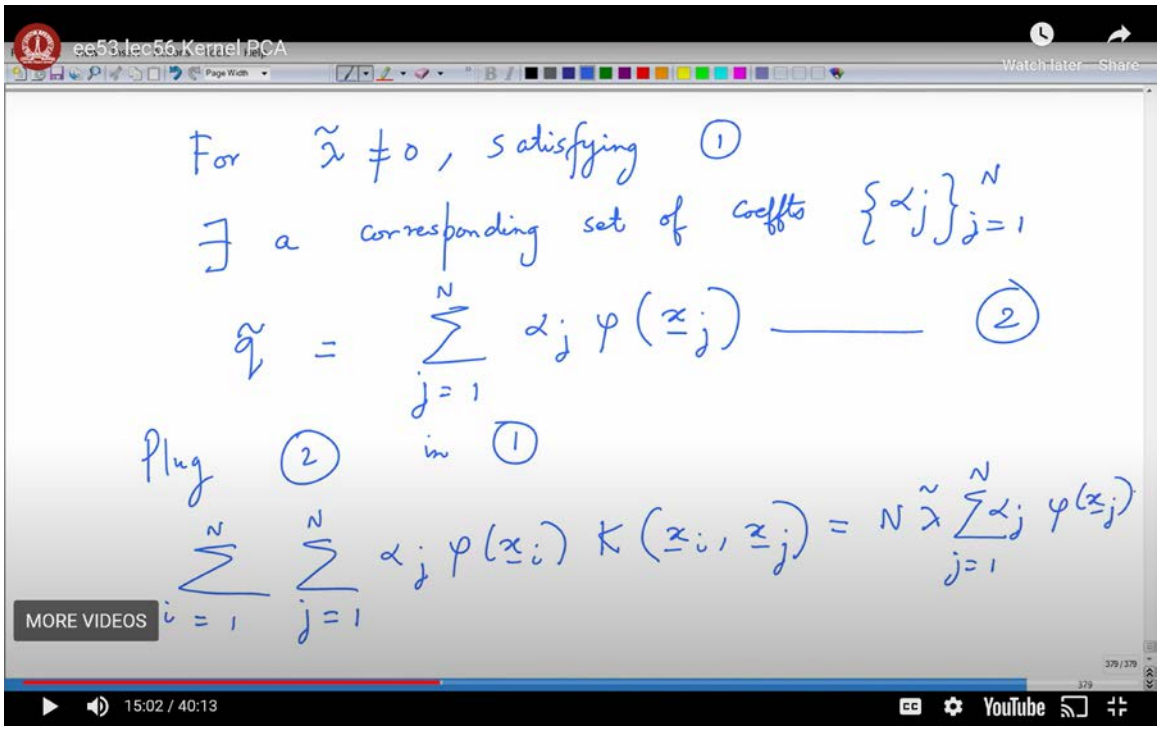
The transformation φ is a non-linear map that takes a vector x_j and maps it to the feature space. Specifically, $\varphi(x_j)$ denotes the image of the vector x_j in this feature space induced by the non-linear map φ. This is the fundamental definition of the image of a point in the feature space.

Given the data points x_i for i = 1 to N, we can compute $\varphi(x_i)$ for all these points. Consequently, we define a correlation matrix in the feature space using the outer products of these mapped vectors. This correlation matrix $\tilde{R}$ is computed as follows:

$$\tilde{R} = \frac{1}{N}\sum_{i=1}^{N} \phi(x_i)\phi(x_i)^{\top}$$

We assume that the mean in the feature space is zero. If not, we compute the bias and subtract it from the data points to ensure that the mean is zero. Thus, the mean of the vectors in the feature space should be zero. Once this setup is complete, we can formulate an optimization problem. The specifics of this optimization problem are detailed in the video slides of the "Mathematical Methods and Techniques for Signal Processing" course, so we will not delve into those details here.

(Refer Slide Time: 15:02)



After setting up the problem, we compute the eigenvalues and eigenvectors of the correlation matrix $\tilde{R}$ by solving the eigenvalue equation:

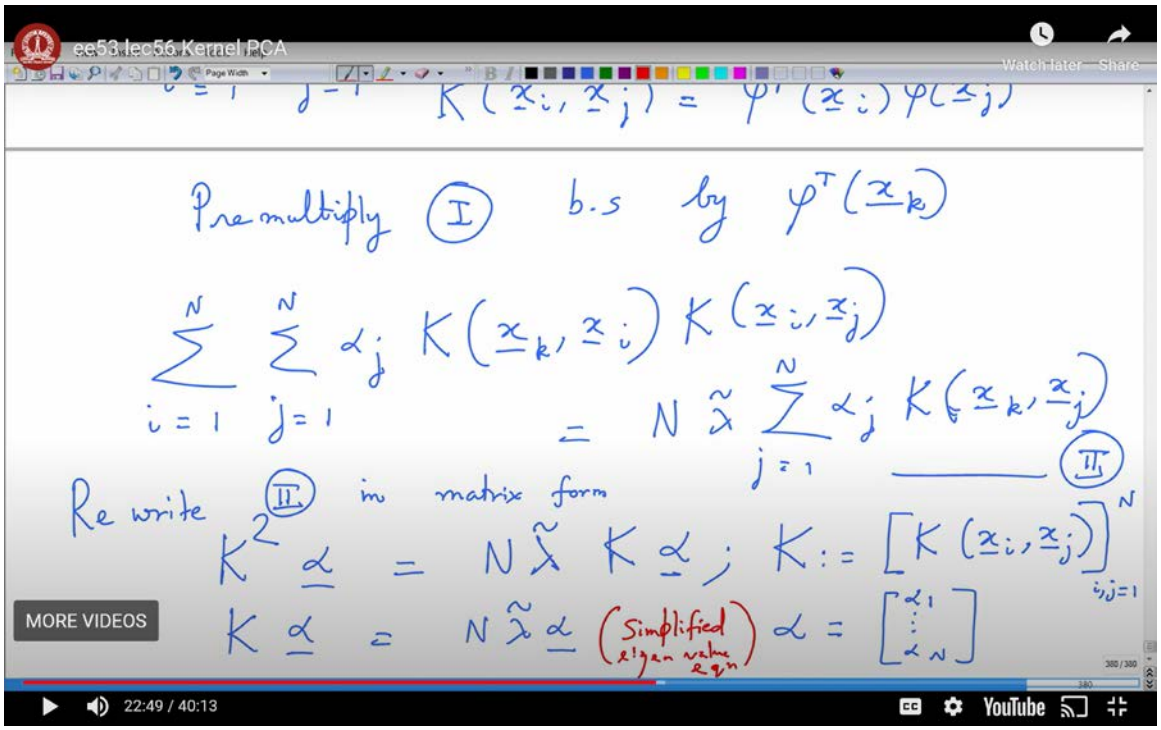
$$\tilde{R}\tilde{Q} = \lambda\tilde{Q}$$

where λ is the eigenvalue of $\tilde{R}$ and $\tilde{Q}$ is the corresponding eigenvector. For a non-zero eigenvalue $\lambda \neq 0$ that satisfies this eigenvalue equation, we can express $\tilde{Q}$ as a linear combination of the elements in the feature space, with coefficients α_j for j = 1 to N.

Substituting $\tilde{Q}$ into the eigenvalue equation, we get:

$$\tilde{R}\tilde{Q} = \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_j \phi(x_i) \cdot \text{kernel}(x_i, x_j) = \lambda \sum_{j=1}^{N} \alpha_j \phi(x_j)$$

(Refer Slide Time: 22:49)



Here, the kernel function $\text{kernel}(x_i, x_j)$ is defined as $\varphi(x_i)^{\mathsf{T}}\varphi(x_j)$. By expanding this matrix $\tilde{R}$ and substituting it into the eigenvalue equation, we simplify to obtain:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_j \phi(x_i) \cdot \text{kernel}(x_i, x_j) = \lambda \sum_{j=1}^{N} \alpha_j \phi(x_j)$$

This step involves straightforward algebra. I will refer to this equation as Equation 1 for reference.

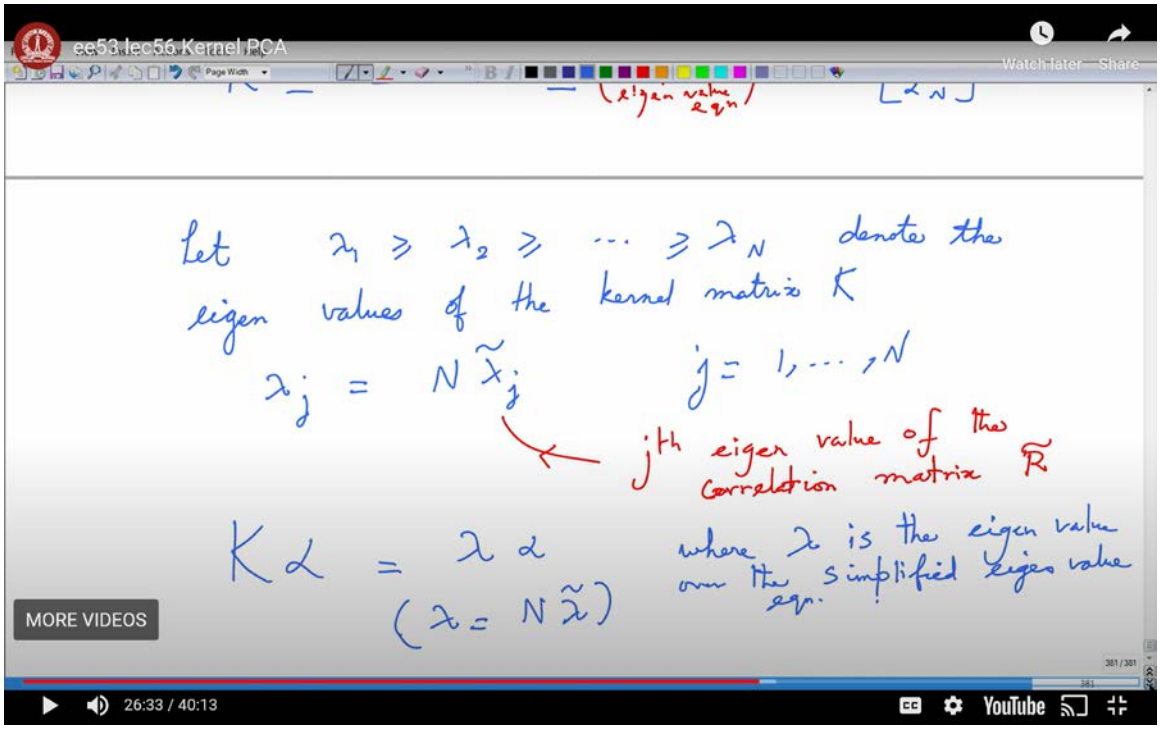
To proceed, let's pre-multiply both sides of Equation 1 by $\varphi(x_k)^\top$. This operation reveals that the kernel function, which is essentially an inner product, simplifies to a scalar value. Hence, when we multiply $\varphi(x_i)\ by\ \varphi(x_k)^\top$, we can use the definition of the kernel function to express this as:

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_j\left(\text{kernel}(x_i,x_k)\cdot\text{kernel}\left(x_i,x_j\right)\right)$$

On the right-hand side of Equation 1, multiplying by $\varphi(x_k)^\top$ gives us:

$$N\cdot\lambda\widetilde{N}\sum_{j=1}^{N}\alpha_j\text{kernel}\left(x_j,x_k\right)$$

(Refer Slide Time: 26:33)



Here, $\text{kernel}\left(x_j,x_k\right)$ is computed as $\varphi\left(x_j\right)^\top\varphi(x_k)$. Thus, this expression simplifies to:

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_j \,\mathrm{kernel}(x_i, x_k)\,\mathrm{kernel}\left(x_i, x_j\right)$$

This correctly uses $\mathrm{kernel}\left(x_k, x_j\right)$ rather than $\mathrm{kernel}(x_k, x_i)$, aligning with the indexing.

The kernel matrix, or Gram matrix, is computed for all pairs of points x_i and x_j where i and j range from 1 to N. This matrix enables us to extract the necessary elements for computation. The $N \times 1$ vector α has its j-th element as α_j.

Rewriting the equation in matrix form, we get:

$$K \cdot \alpha = N \cdot \lambda\widetilde{N} \cdot$$

Let's refer to this as Equation 2. Here, K denotes the Gram matrix comprising the kernel evaluated over all data point pairs x_i and x_j, and α is the column vector stacked with α_1 through α_N.

To simplify this, assuming the inverse of the Gram matrix exists, we pre-multiply both sides by K^{-1}:

$$K \cdot \alpha = N \cdot \lambda\widetilde{N} \cdot$$

This gives us the simplified eigenvalue equation in terms of the kernel matrix K and the number of data samples N. Here, α represents the eigenvector, and $\lambda\widetilde{N}$ denotes the eigenvalues of the Gram matrix K.

For the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_N$ of the kernel matrix K, if we set λ_j to be $N \cdot \lambda_j\widetilde{N}$ for j = 1 through N, we adjust the eigenvalues accordingly. This setting aligns the eigenvalues of the kernel matrix with those of the correlation matrix $\tilde{R}$.

To simplify the eigenvalue equation, we can express it as:

$$K\alpha = N\lambda\tilde{\lambda}$$

This simplifies to:

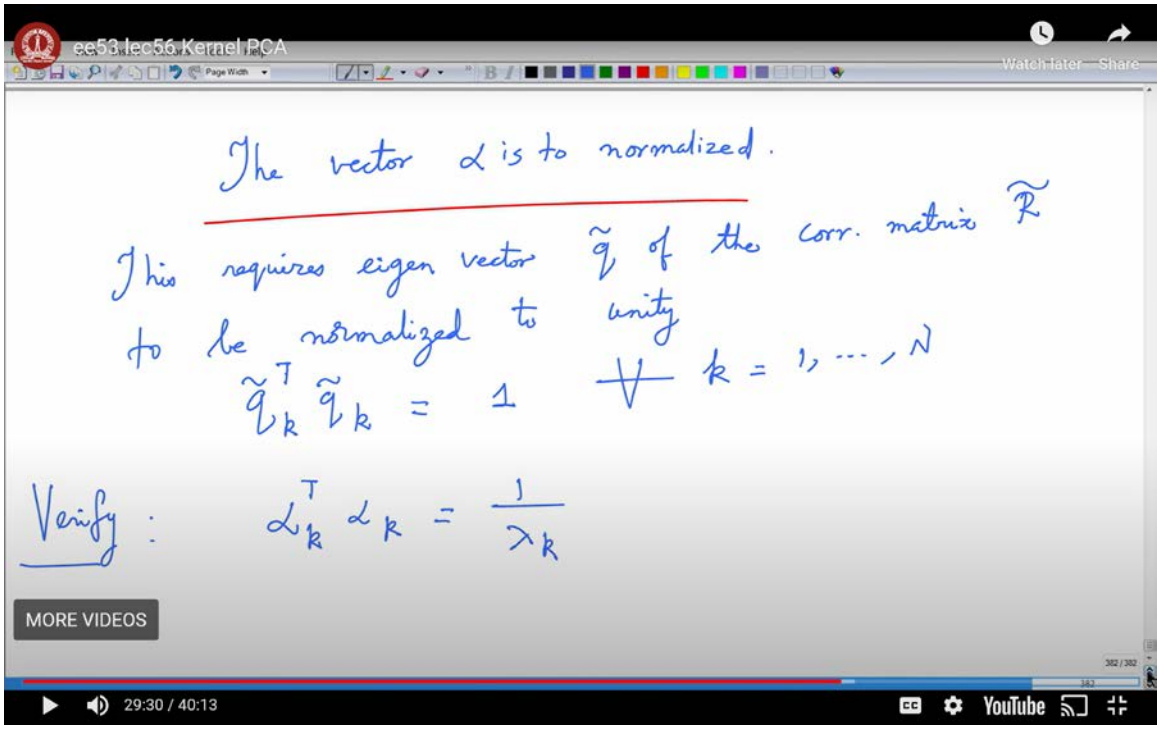
$$K\,\alpha = \lambda\alpha$$

where λ is the eigenvalue defined as $N \times \lambda\tilde{\lambda}$. This is a straightforward process. By substituting λ_j with $N \times \lambda_j\tilde{\lambda}$, we get a more simplified representation:

$$K\,\alpha = \lambda\alpha$$

Here, λ is given by $N \times \lambda\tilde{\lambda}$. In this context, the subscript has been removed because each eigenvector α_j corresponds to an eigenvalue λ_j.

(Refer Slide Time: 29:30)



Additionally, the eigenvector α must be normalized. This means ensuring that the eigenvector $Q\tilde{k}$ of the correlation matrix R $\tilde{R}$ is normalized to unity, which implies:

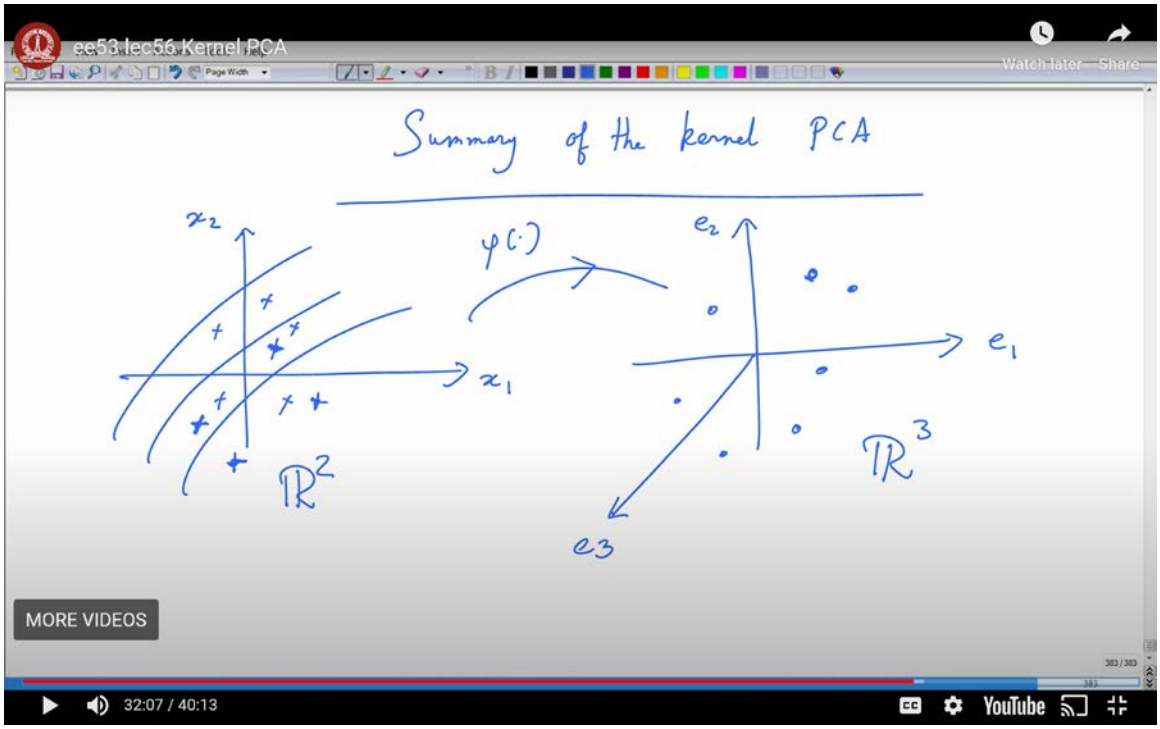
$$Q\widetilde{k^{\top}}Q\tilde{k} = 1$$

for all k ranging from 1 to N. When arranging eigenvalues in descending order, you can formulate it as:

$$\alpha_k^\top \alpha_k = \frac{1}{\lambda_k}$$

This relationship can be verified by normalizing the vector $Q\tilde{k}$. I suggest leaving this verification as an exercise to practice normalization for eigenvectors.

(Refer Slide Time: 32:07)



Now, let's summarize the kernel PCA process. Conceptually, imagine you have points in a 2-dimensional space. These points are non-linearly mapped into a feature space, which could be a 3-dimensional space. In this feature space, you compute the eigenbasis. For instance, if you denote this basis as E, it consists of E_1, E_2, E_3, and you represent the points in this space accordingly.

Here's the algorithm for kernel PCA:

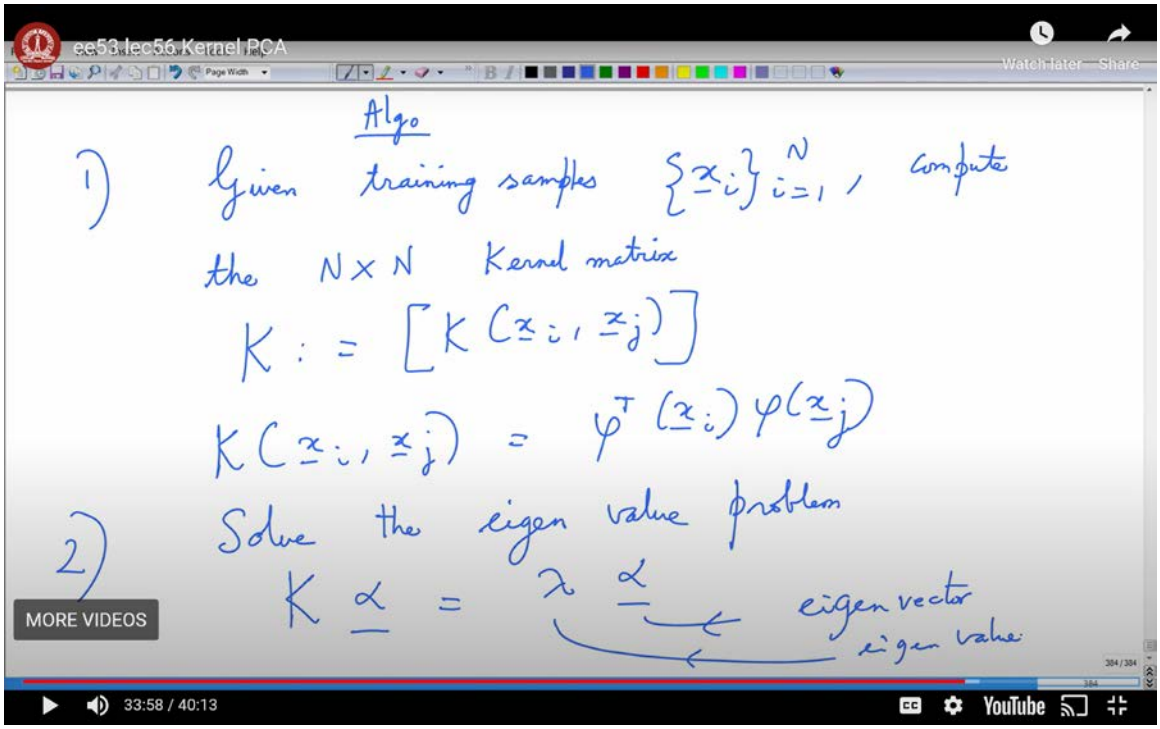
1. Compute the Kernel Matrix: Given training samples x_i for i = 1 to N, calculate the $N \times N$ kernel matrix (or Gram matrix) K, where each element is given by $K(x_i, x_j) = \varphi(x_i)^\top \varphi(x_j)$.

2. Solve the Eigenvalue Problem: Solve the eigenvalue problem:

$$K\,\alpha = \lambda\alpha$$

where λ represents the eigenvalues of the kernel matrix, and α are the corresponding eigenvectors.

(Refer Slide Time: 33:58)



3. Normalize the Eigenvectors: Ensure that the eigenvectors are normalized such that:

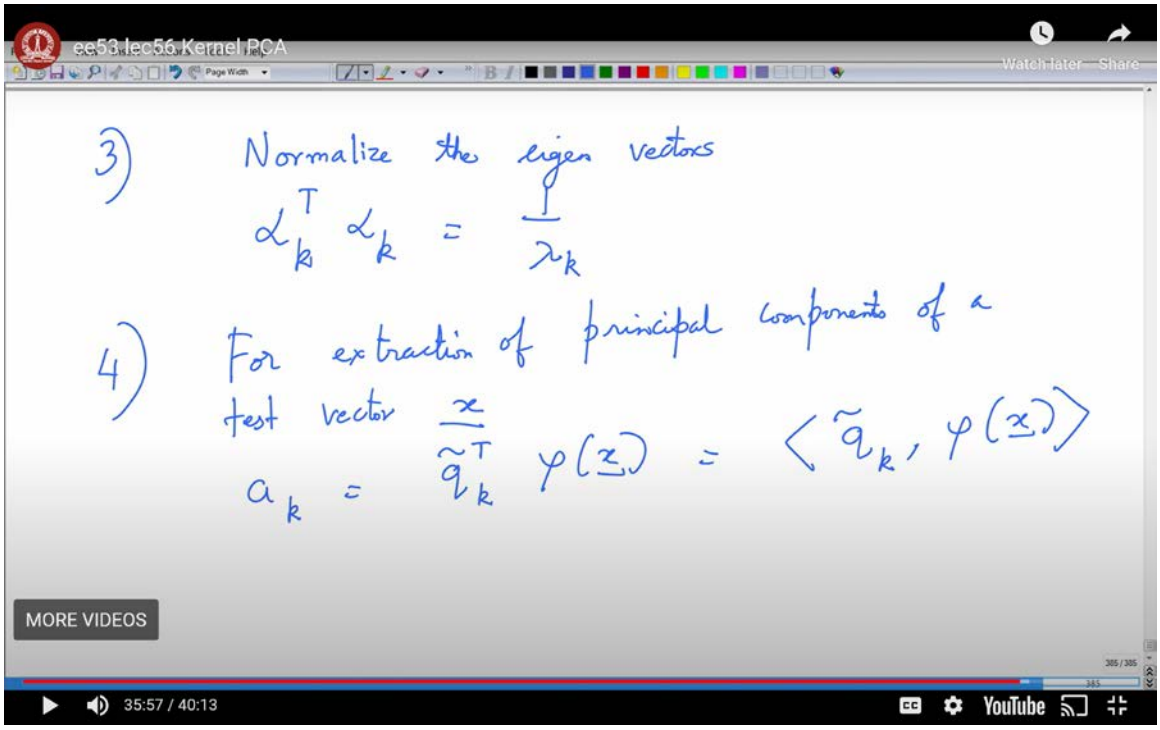
$$\alpha_k^T \alpha_k = \frac{1}{\lambda_k}$$

By following these steps, you can perform kernel PCA to analyze and reduce dimensionality in the feature space.

You can arrange the eigenvalues in descending order and stack the eigenvectors corresponding to these eigenvalues in their respective directions. For extracting the principal components of a test vector X, you need to compute the projection of φ(x) onto

the eigenvectors. This involves taking the inner product of the eigenvectors with the vector in the feature space. By doing this, you can compute the projections and represent the vector in the feature space, leading to a compact representation if needed.

(Refer Slide Time: 35:57)



Now, you might wonder about the necessity of Kernel PCA and its benefits. In the input space, coordinates often have physical meanings; for instance, the first coordinate might represent pressure, the second might represent temperature, and the third might represent distance. By formulating the correlation matrix in the input space, you can visualize correlations across these physical coordinates through PCA.

However, when using Kernel PCA, you perform a non-linear mapping to a higher-dimensional feature space. This mapping helps reveal the geometry of data points in a different space, potentially making patterns more apparent that weren't visible in the original lower-dimensional space. While PCA is a linear transformation and deals with data compression by retaining essential coefficients and discarding the rest, Kernel PCA

allows for better realization and visualization of data geometry, especially when linear transformations in the original space are insufficient.

In applications such as pattern recognition, where you might work with raw images or speech signals, you may need to extract meaningful feature vectors. These features can be obtained through linear or non-linear transformations, depending on your application's requirements. Kernel PCA can then be applied in the feature space to compact the energy or components further.

It's important to note that while Kernel PCA involves a non-linear transformation and might result in coordinates in the feature space that lack physical interpretation, it can still be valuable for dimensionality reduction and optimization of energy in that space. This approach is particularly useful when the original physical representation is less relevant compared to the benefits gained from non-linear dimensionality reduction.

To summarize, Kernel PCA is a powerful tool for applications where non-linear transformations provide better insights into the data. The derivation of Kernel PCA, in terms of the optimization problem, follows a straightforward approach similar to that of ordinary PCA. For further details, you can refer to the videos or consider this as part of a homework exercise.