IIT Madras

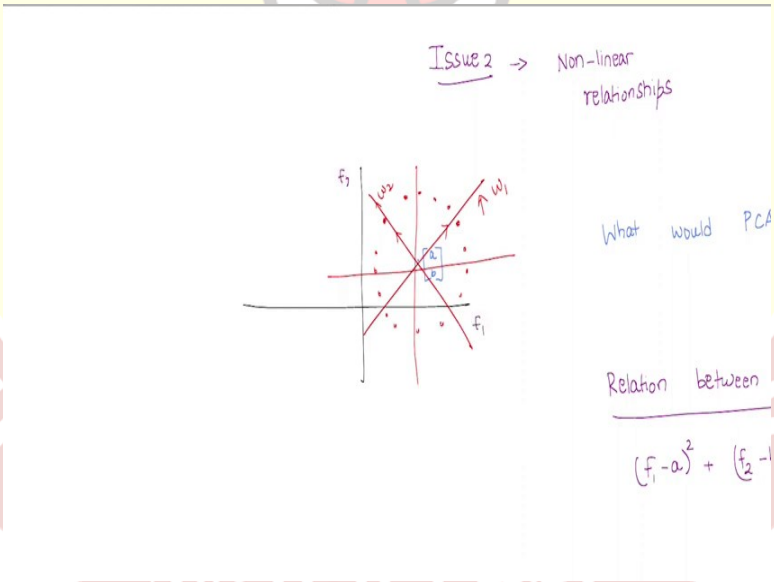
ONLINE DEGREE

Machine Learning Techniques
Professor Arun RajKumar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Feature Transformation

Welcome back, we have been looking at the principal component analysis algorithm. And we were trying to understand some of the issues related with this algorithm, we identified two different issues. One issue was when the number of features is much, much larger than n. That resulted in a computational issue where you had to spend order of d^3 for computing the eigenvectors and Eigen values in general, where d is the dimension of your data.

And we try to solve that issue by saying that instead of computing the Eigen vectors for the covariance matrix, you could compute the Eigen vectors for an associated matrix, which is the $x^T x$ matrix, where x is the data set, a matrix and you can use the Eigen vectors of the $x^T x$ to convert it into the Eigen vectors of the covariance matrix. So, that kind of solved issue one.

(Refer Slide Time: 01:14)



Now, we will look at issue two, which we also mentioned in a previous video. So, the issue two, that we are going to talk about today is the issue of nonlinear relationships among features. So, we know that PCA is very good at identifying the linear relationships among features. But what, if we just had a nonlinear relationship? To motivate this, let us take a simple example, where we have again, a two-dimensional data, where, let us say the data points were like this. That is the

data points all lie on the circle, on the, on the circumference of a circle, let us say centered at a and b.

Now, let us ask the question, what would PCA give? So, if I run the standard PCA algorithm on this data set. What should I expect to see? In other words, what are the most interesting or important directions with respect to, the variance maximization or error minimization that PCA will uncover? What will be the most important direction? So, if you are already seeing it fine. Otherwise, I encourage you to pause and think about this question. I will answer this question now.

So, what would PCA do? PCA would first center this data set, which means that the center will move from a,b to 0, 0, so the origin will be 0, 0. And then what it would do is it would try to find that direction where if you project this data points, the length of the errors is as small as possible, or the variance is as high as possible. Now, because the points are all around the circle, no direction is more important than other direction.

So, I can project my data along this direction, or this direction, or this direction, or this direction. And they would result in more or less the same variance, I say more or less, because depending on the exact data points and how they are spread around the circle, one direction might be slightly better than the other.

But in general, all directions are equally important. So, which means the PCA is going to pick one direction that would be based on how exactly these points are around the circle. But let us say PCA picked this direction as w_1, the most important direction. Now, what would be w_2? Well, w_2 we know has to be perpendicular to this w_1. Well, it would be this direction, let us say, of course, I am assuming a instead of a,b, this is origin centered here.

So, now, if we had to do a dimensionality reduction for this problem, using PCA, then what would happen is the following. So, we would compute the Eigen values, which is simply the variance along each direction that PCA finds and then try to see, how many directions do we need to capture 95 percent of the variance.

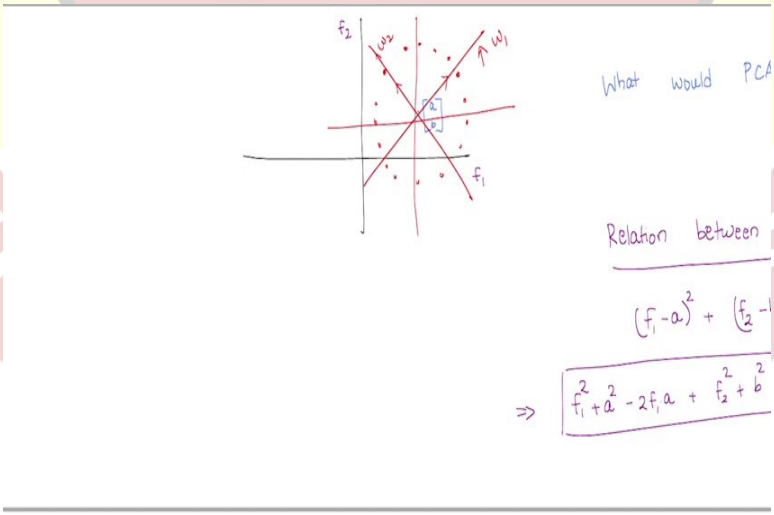
Now, in this particular example, what would happen is your w_1 will have, let us say the most variance. But then because all directions are almost similar to each other w_1's variance or the

variance of the data set along w_1 would be similar to the variance of the data set along w_2 which means that maybe slightly more than 50 percent of the variance would be captured with w_1 and slightly less than 50 percent would be captured by w_2.

Now, this would tell us then that if I use the thumb rule of 95 percent to pick the top k directions, PCA would say in the standard way that you need both directions. Both the directions are important. So, that is what PCA would give. So, it would give w_1 and w_2. And both important, it will think both are important actions.

But the real question is, do we really need two directions here? In other words, do we really need two numbers here. I mean, is the relationship such that you need necessarily to, to capture two directions? Or is there a different way to capture this relationship? So, let us think about that. So, what is the real basic fundamental relationship among the, among these data points, by the virtue of the fact that they lie around a circle, the following is true. The relation between features is the following. So now, any data point I can take in this data set, and it has to satisfy $(f_1 - a)^2 + (f_2 - b)^2 = r^2$, where we say, when I say f1, this is feature 1, the axis y axis is feature 2.

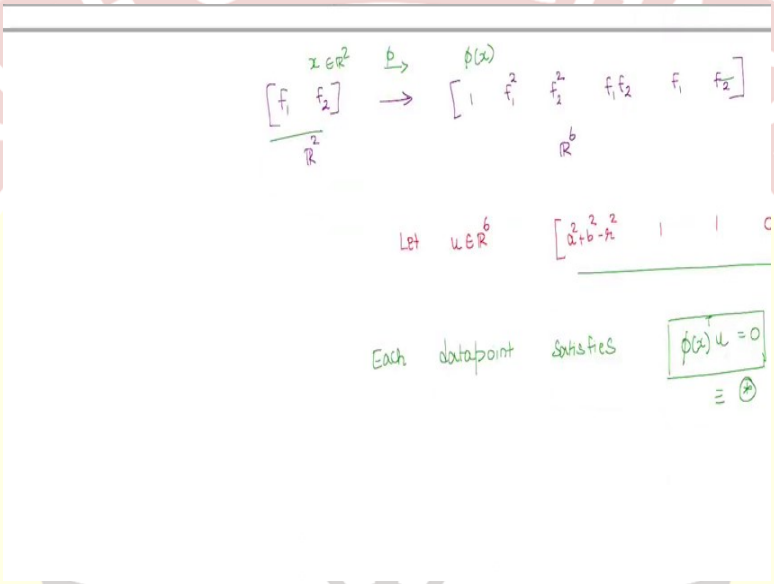
(Refer Slide Time: 06:15)



Why because this is the equation of a circle centered at a and b with radius some radius r. So, which means that every data point is on the circle, and so it satisfies this equation. Now, let us

expand this equation and see what we get this implies, we get $f_1^2 + a^2 - 2f_1\ a + f_2^2 + b^2 - 2f_2 b - r^2 = 0$. So, this is the basic relationship that that all the data points satisfy. And as we can see, this is not a linear relationship. Why? Because this has f1 squared term, f2 squared term and so on. So, this is not a linear relationship. That is that is obvious from the picture, but also from the equations.

(Refer Slide Time: 07:04)



So, now let us do let us try to do the following. Let us take the feature f1, f2, this is just some data point. I mean, maybe f1 is 5, f2 is 10, some numbers, so two numbers. And now what I am going to do is for each point in my data set, I am going to map it to some other data points. So f1, f2, if f1, f2 was my data point in my data set, I am going to map it to a different point.

And that different point looks like this. So, the first coordinate of this map vector is 1, the second coordinate is f_1^2, the third isf_2^2, fourth is $f_1\ f_2$, fifth is just f_1 , sixth is just f_2 (. So, this is a two-dimensional vector that you give me. And then I map it to a 6-dimensional vector, now.

So now, what is the use of doing this? Why am I doing this mapping? So, for this, let us say you consider this vector u in sixth dimension, let us say you have a 6-dimensional vector, and that vector is the following. So that vector is, and I will tell you why we are choosing this vector $[a^2 + b^2 - r^2\ \ 1\ 1\ 0\ -2a\ -2b]$.

So, this is a sixth dimensional vector that I am choosing, which has nothing to do with the data points. So, well, so it is it is a common vector that I am choosing. It has something to do with the data points, we will see what it is, but it does not use the data points. So, to define this vector, I have not used any of the f1 or f2 of the data points.

So, now what to be observed from this, so I have given a data point, I have mapped it to a 6-dimensional vector. And then I have also exhibited a 6-dimensional vector, which I am seeing special with respect to this dataset. So, what is so special about that? Well, we know that each of the data points satisfies this equation here, so the star equation here, so because it satisfies the star equation, now you can see that each data point satisfies, $\phi(x)^T u=0$, where ϕ is the mapping that maps this 2-dimensional data point to a 6-dimensional data point.

Now, once I have mapped f1, f2, which is my specific two-dimensional data point, we put any number here you get a data point on the circle. Well, our data set has if we put any point from the data set, as f1 and f2 then that is the point on the circle. Now, I have mapped that to 6-dimensional feature using ϕ.

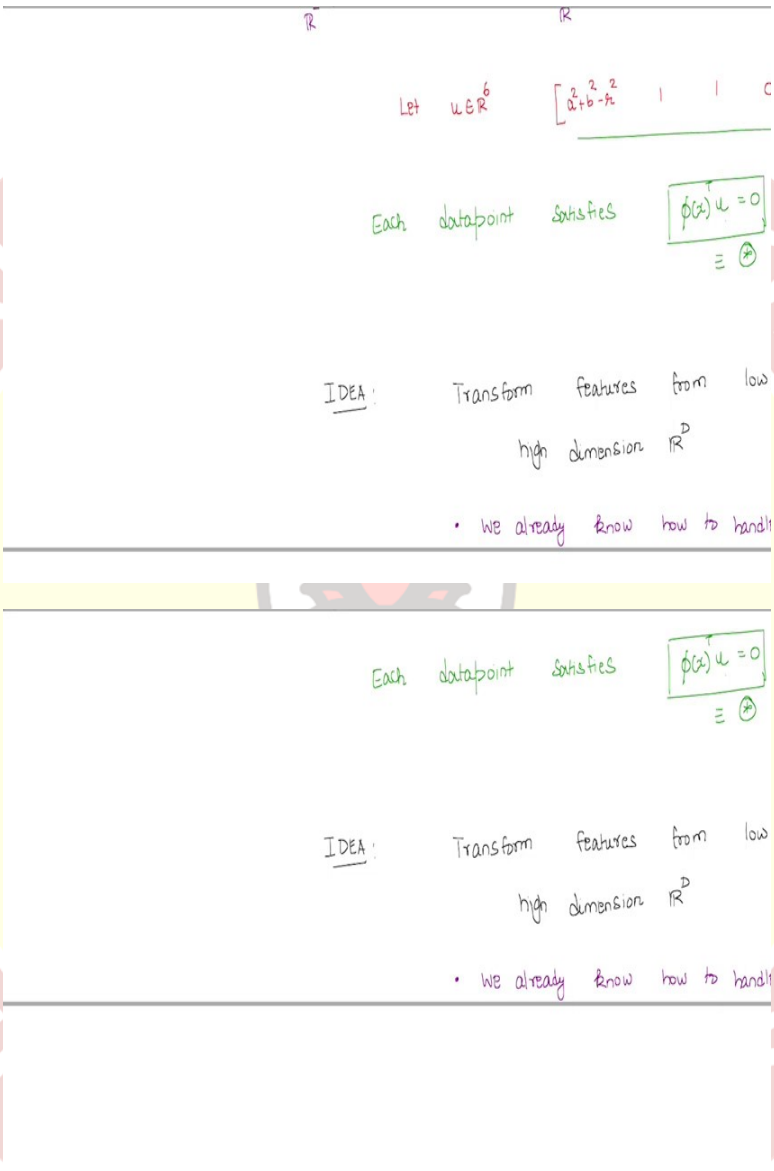
So, ϕ is a map that is that is defined on all of our, but then for those points that are there in my data set, if I use the ϕ map to map it from a two-dimensional space to a 6-dimensional space, then I immediately noticed that all the points in the data set satisfy the equation $\phi(x)^T u=0$. So, that is this implies that the data points are all orthogonal to this vector u in the 6-dimensional space after they have been mapped using this ϕ map. So, that is the data points lie in linear subspace of R^6.

So, that is the interesting point here. So, so, what we have done is, we have mapped our data points to a high dimensional space. Now, you can verify that the dot product of this is exactly this equation here. So, which means that $\phi(x)^T u=0$ is same as equivalent to start. So, this is same as done equal, not in place.

So, because this $\phi(x)^T u=0$ is a linear equation in the higher of x space, basically, what we are saying is that the data lies in some low dimensional space in this map space. So, you have the data in two dimension, you mapped it to six dimension. And now, we suddenly observed that, well, the data is not really, truly in 6 dimension, it is in a low dimensional space in the 6

dimensional space. So, now the question is, well, what, what is the use of this? So why? Fine, so, this is fine.

(Refer Slide Time: 11:56)



So, but how can we convert this into an idea? Well, here is the idea. The idea is the following. Now, given a data transform features, from low dimension R^d to high dimension R^D, so, you have x which is in R^d, and now you apply some ϕ and then map it to D, where you hope that the data lives in a low dimensional subspace. Now, a big low dimensional linear subspace, so, and because the data lives in a low dimensional linear subspace in this map dimension, now, we know already how to extract this low dimensional linear subspace.

So, the important directions corresponding to this low dimensional linear subspace, that is what our PCA anyway does. So now, because you are doing this mapping, you are increasing the dimension. So, which means your d could be really large. But now, we already know how to handle the case when d is much larger.

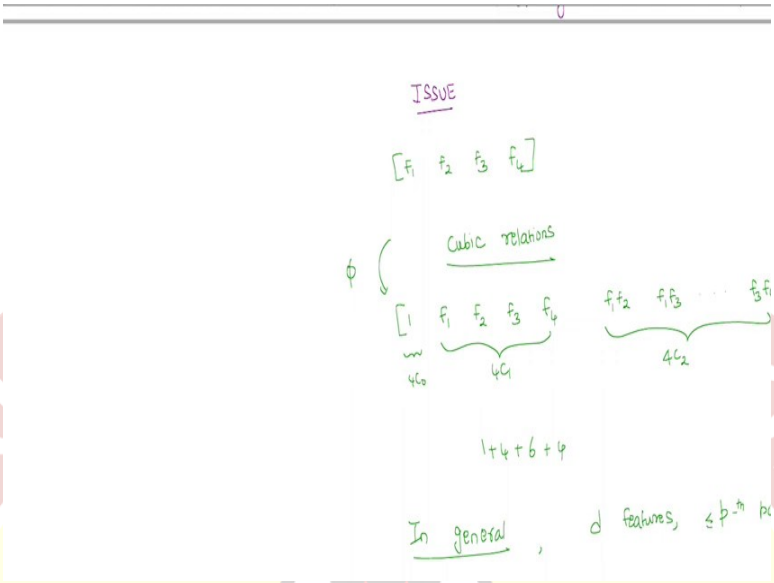
So, we already know how to handle case when d is much, much larger than n. How do we do that? Well, we know that we have to look at the covariance matrix, but we can look at this matrix which is the other way around. So instead of xx^T, you we will look x^Txin the usual PCA.

But now, because the data points are no longer x, they are$\phi(x)$. So, what you do is you look at $\phi(x)^T\phi(x)$. So instead of $\phi(x)\phi(x)^T$, when you say $\phi(X)$, it means that I am applying the phi function to every vector in my in my data set. So that is, that is, I mean, a slight abuse of notation. But that is what I mean when I say ϕ apply to matrix it applies column wise.

So, so this seems like a great idea. It is in the sense that we know how to solve issue one, which is when d is much much larger, we know how to you know, solve that problem. Now, we are saying that in the case of nonlinear relationships, you map your data points to higher dimensional space, where these nonlinear relationships are captured better. Now, where is the non-linearity coming from? Now, the non-linearity is basically, absorbed into this ϕ map. It is a ϕ map has all possible nonlinear relationships captured.

Once you do that, then yes, so in the high dimensional space, you can find a linear relationship using your PCA because B is much much larger than n, you can still run your PCA. So, it seems like a reasonable idea to start with. And it is. So, that is what we are going to see how to convert this into a like a solid idea. But then we will hit a lot of bottlenecks, we will hit a lot of issues, and then we will have to handle them.

(Refer Slide Time: 15:15)



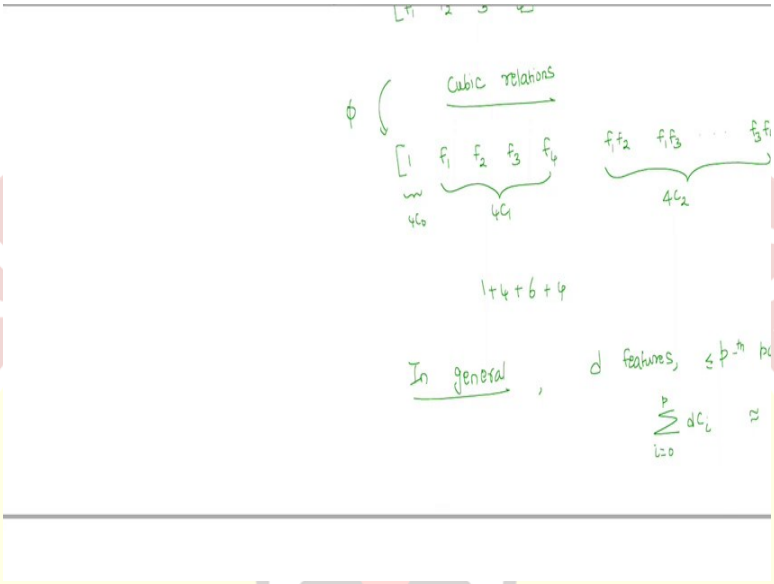
The first issue, the most important issue perhaps is the following. Let us say you had four features, and you wanted to capture cubic relations. Think of these four features as height, weight, a gender, let us say. So, when you say cubic relationships, it means that I should be able to capture relationships such as height x weight x gender, or height x weight^2 x age, sorry, not that that would be fourth power, but let us say heightxweight^2, or weight^{2}xh, it is all possible combinations where if you sum up the power of the combinations, there will be 3. So, in this case, so the number of, so basically, if I do the mapping, ϕ, which captures all cubic relationships in remember, there is a different final from the ϕ that we used earlier.

Now, this ϕ will have all possible cubic relationships. Now, this is going to be a constant f1, f2, f3, f4, well, these are the relationships which are to the power 1. To the power 2 would have f1, f2, f1, f3, and so on, till f3f4. And then there will be $^4C_2$, so this is$^4C_0$, this is $^4C_1$, the 4 is the number of features that you have case, ^4C_k , where k is the number of the power that you are talking about.

This will be $^4C_2$, and then there'll be f1 f2 f3, f1 f2 f4, and so on. So, there will $^4C_3$ of these guys. Then we stop at that, because we only care about cubic features, let us say. So now the total dimension of this map $\phi$is 1+4+6+4. So, this is $^4C_1$, $^4C_2$, $^4C_3$. Now, the question is, how does this grow? So, if I had d features, so in general, if I had d features, and if I care about the pth

power, so d features and want model pth power relationship, up to pth power. Let us say, less than or equal to pth power.

(Refer Slide Time: 17:38)



Then, how many features would the map ϕ have? If you think about it, that it is going to be $\sum_{i=0}^{P} dC_i$. That is what it would be. And one can show that this is something like this grow something like $d^P P$. Now, what does this tell us? This tells us that if d is let us say, 10, and I cared about some kind of 15th power relationship, let us say, so now that would mean I would the map, ϕ would have 10^{15} features, the original dimension had only 10. But then in the map space, these are going to be 10^{15} .

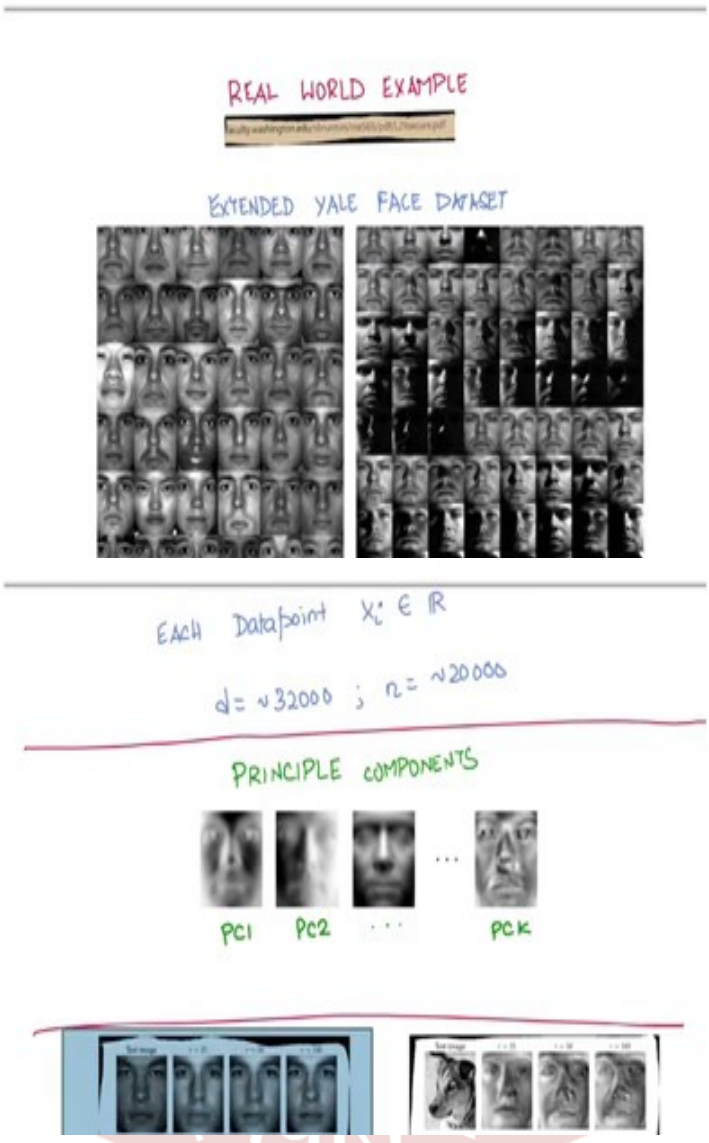
So, if you even if you had just two features, but then if you wanted to for whatever reason, your relationships were so complicated that you need 20th power, then it would be you would have to map your two-dimensional feature to a 2^{20} dimensional space. And that is going to be simply too hard to work with, so as the numbers the power increases, and as the features increase, so this will become harder and harder.

IIT Madras

ONLINE DEGREE

Machine Learning Techniques
Professor Arun Rajkumar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Issues with PCA

(Refer Slide Time: 00:13)



Hello, everyone. Welcome back. So, today, what we will do is continue our discussion on PCA. And last time, we said that we will look at one example for PCA, (real time application), real world application. And we will also look at some issues with PCA. And that will lead us to something more interesting, which not only fixes the issue, but also gives us a general idea which can be used for many supervised learning algorithms that we will see later on.

But before going there, let us start with very interesting, real world application of PCA which is called as the Eigen faces application. So, what is this application? So, to look at this application, let us look at this example. So, this is a real world example, which is called as the YALE FACE DATASET, the extended YALE FACE DATASET.

And the reference to this dataset is in the right here in the Washington University's website. So, it is just a bunch of faces, images of faces, which are of the same size, so all facing the camera images, all grayscale images, and around 20,000 of them. So, that is what this dataset contains. Now, how do you convert an image into features?

Well, the way that this is done for the Yale dataset is a very simple idea, you take, you treat the image itself as a bunch of pixels. So, basically, so each image consists of some mxn, you can think of it as an mxn matrix of pixels. And each pixel value is a grayscale value, which is something some number between 0 and 255.

And now you can, vectorize , this whole matrix mx$n$matrix into a vector of size mnx1. So, it is a, so when you do that, so there are around 32,000 pixels, I am not putting the exact number here it is around 32,000, maybe slightly more, that does not really matter for our discussion here. But there are around 32,000 pixels in each image.

So, each image becomes a point in a 32 dimensional space. Now the question is, well, if you want to get a compressed representation of this data set, so can we use PCA to do this, basically, you treat this as a data set containing 20,000 points, 20,000 faces, and each point is a vector in 32,000 dimensional spaces.

Now, what you could do is just run your standard PCA, whatever we have seen so far, that is compute the covariance matrix, look at its Eigen directions, and each Eigen direction is now going to be in the feature dimension. So, the covariance is a dx$d$matrix. So, which means there will be a 32,000 cross 32,000 matrix, if you look at the Eigen vectors for it.

So, they will also be in 32,000 dimensional spaces. So, the best line that fits will be a line represented by a vector in 32,000 dimensions. Now, what you could do is the following, you can

now again convert this 32,000 dimensional vector back into an image and see how that looks like.

So, the way we went from images to 32,000 dimensional points was map this mx$n$to an mnx1 dimensional vector. Now, we can do the same thing in the reverse as well. So, because there is a specific ordering for these coordinates, where you converted your mxn to mn dimensional vector, you can do the you can go back as well.

And if you did that, we can ask how do, these principal components look like if I visualize them as images. And when you do that, for this dataset, you see, the principal components look like these. So, basically, the first principal component looks like this image here. The second principal component looks like PC2 second image and so on.

Now, let us pause for a second and see what do these principal components intuitively mean? So, why are they looking like the way they are looking? So, basically, PC1 kind of captures that direction, which has the most variance in the data set. That is what we saw PCA does earlier. Now, if you convert this into an image.

So, it is as if this is like the best prototype image that captures most of the variance in your data set because this is a dataset full of human faces. So, now this image captures the faceness of this data set in the best possible way, you could say that. Now as you look at further more principal components, what you observe is that you get more and more details that get added to these directions.

So, for example, PC K, which is some Kth principal component, now has a much more, in some sense a nuanced representation, it captures much more details. So, there is, the noses are more clearer the mouth is more clear and so on. So, now, why does this happen, because, what is common for most of these data points is this template, is the most common template.

In some sense, this is where most of the variance is, but then as you look at other directions, they are trying to find out other characteristics of this data set, which are also important, which is where more and more details get added, as you increase more and more principal components. Now, the question is, what do we do with this dataset?

One question is, so let us say if you want to build a, like a face recognition system, for the employees in a company or something like that, where you have 1000 employees, so, in this case, 20,000 employees

Now do you really want to store 32,000 numbers per employee to capture that person's photograph, so 32,000 pixel details? Do we really need that? Or is there a lower dimensional representation that perhaps, represents what we want really well? So, now PCA would tell us if that is the case.

So, now, what we are trying to do now, what we will try to do now is we will increase the number of principal components that we will retain or in other words, the number of rounds, we will run the algorithm for. Or in other words the number of Eigen directions, that we are going to retain all of these are exactly the same things in our context.

And now, we will try to reconstruct the image, the proxy for, this image using only these retained principal component. In other words, what does this mean? This means that if I just retained one principal component, then for each of these images, I am going to take that image dot product with the first principal component that will give me a number.

And now to reconstruct the proxy, I will multiply this image with just that number. So, all the images will be some scaling of this number. Now, if I am reconstructing with two images, it means that I will take the dot product with the top two Eigen directions, principal components, and then I linearly combine these dot products, using with these images to get a new image.

So, now the question is, as I increase the number of Eigen directions, how do these images start looking like and that the Yale folks did that as well. And what you see is something like this? So, for example, if you let us focus on the left hand side image, first now here is a test image. So, let us say this is an image that is not in your dataset, maybe again, in a face recognition context, this could be some person who walks into your company.

And then your inbuilt CCTV camera or something like that clicks a picture of this person, and then you get an image of this sort. So, now, what you are trying to do is, you are trying to reconstruct this image using only the top few principal components. This is the image reconstructed using just 25 principal components, the next one using 50, 100, and so on.

Now, focus on the original image, and the reconstruction using 1600 principal Components, that is pretty much close. So, you can already tell this is exactly the same person. In fact, you do not really need 1600, well, is 25 enough? Perhaps not.

So, if you only use 25 components, it is not very clear if this is exactly the same person, it is clear that it is a person. But it is not clear if it is exactly the same person. But then if I use 200, components, let us say, or even 300, 400, or somewhere between 200 and 400, pretty much all the important details of this person specific to this person is also captured in some sense.

So, you only need something like 300 directions to capture this person's, essential features that we need to say if this is the person or not, you do not necessarily need all the way to 1600. Well, if you go to 1600, you are adding more and more, finer details. So, maybe there are more details about how the faces how the eyes and noses are, and so on, which is perhaps not necessary to really say if this is the same person or not, only 300 numbers are sufficient.

So, which means to say that the original image had 32,000 pixels, but now what are we saying? We are saying that well, just 300 numbers, you can perhaps recognize each of these persons pretty well. You do not need 32,000 numbers. So, that is like 100 times again, that you can get by doing this PCA. Well, here is another way to look at this.

So, here is a say a different image, which is not characteristic of the data set from which (you are running) your running PCA and getting the principal components. So, this is a dog image. So, now this also has 32,000 pixels. Now, what are we doing, we are only taking the top 25 and trying to reconstruct this dog, the proxy for this dog in that 22,000 dimensional space that best represents this dog.

But because the 25 dimensional space that best represents a dog that is what we are trying to find and because these 25 directions are the top 25 directions with respect to human faces, if you are trying to project our dog onto this 25 dimensional human space, you are not really going to see that dog reconstructed that well.

So, you are only going to see something like, the best human representation of this dog. So, that is what you are essentially getting. You are not really getting the dog reconstruction that well, now as you keep going, so for r equals 100, well, still not that good for r equals 200 still pretty

bad, 400 is pretty bad. 1600 still. So, when you have 1600 dimensions, (then the dog), the reconstructed dog pretty much looks like the dog.

Now, the point I am trying to make here, and this is important in understanding what is happening here is that, because the data set on which we got these principal components from content, only human faces, it is trying to capture the most important directions with respect to human faces, which means if you start with some other the dogs face, or even a tree, you are not going to get good reconstruction with small number of principal components.

So, if your goal was to distinguish a man, or a face, man or a woman, from a dog, so a human from a dog, then you do not necessarily perhaps need even 200 components. I mean, even very small number of components is good enough. Even with 25 components, you can say if it is a human face, or a dog face.

So, because the dogs faces with on the 25 dimensional subspace looks nothing like human or nothing like dog, so you still might get some distinguishing capability. On the other hand, if you really want to say it, is this particular human, well, you still do not need 1600. So, you only need something like 200, 200 to 300.

But then with 200 to 300, the dog does not look anything like the dog, with 1600 you are somehow able to reconstruct both the human faces and the dog's face correctly. So, which means there is enough information in the top 1600 directions that can represent perhaps any face. So, anything that has eyes, nose, and mouth, perhaps is captured in that, 1600 dimensions.

So, you do not, now you would not really be able to, I mean the dog looks like the dog, the human looks like the human. So, with 1600, you kind of get most of the information. Whereas if you took a tree, perhaps you would be much more dimensions to reconstruct the tree 1600 may not be enough, or if you just took a random image so, an image which where each of the pixel values were absolutely random.

Now, how many dimensions do you think , you would need to reconstruct this image? Of course the image does not have any structure. So, it would not even be possible to say if the reconstructed image is similar to the original image or not. But then if you want to exactly

reconstruct this, you would need all 32,000 dimensions. But remember, if you have 32,000 dimensions, you can reconstruct any image so, because you have complete information.

So, for any set of 32,000 , dimensional vector, what we are asking is with smaller numbers, when can you reconstruct well? Well, if it is a human face, you can reconstruct well, with just 200 to 300 dimensions, if it isa dog face you perhaps need 1500 to 1600. If it is a tree, maybe you need 2000 3000. I do not know, I have not tried it on this dataset, but that would be my guess. But if it is a random image, you would need more.

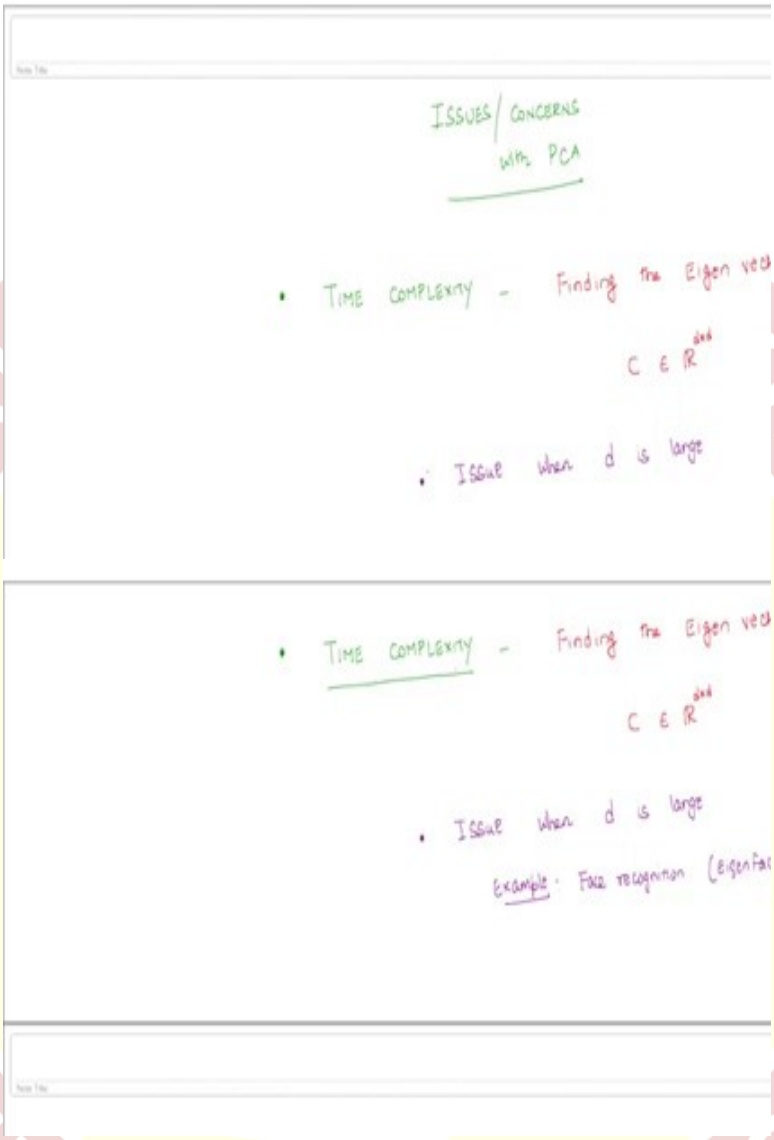
So, 32,000 in the best the worst case so, you would need to store them. So, this is a very good example of PCA which is called Eigen faces, which can recognize faces, specifically human faces. Of course, we will use this as like a pre processing technique to store only the top projections on the top few Eigen directions, and then use it later on to train a supervised learning algorithm. That is the basic idea.

So, given a new test image, you will again project it onto the top few Eigen directions with respect to the human faces and then run your supervised learning algorithm to see if the, if you got the correct answer or not. And, because this works, because there is some structure to these natural images so, there is some underlying subspace , that is the technical word to use.

So, there is some underlying linear subspace where most of the information is captured of the human faces. And so, this algorithm works. So, that is a nice, example of an application of PCA, there are many more and just wanted to show you one example. Now, let us continue our discussion to understand what are some, shortcomings of the PCA algorithm itself and how we can fix those shortcomings.

And as I said, the more we try to understand these shortcomings, and then try to fix them, that will lead us to something very, very interesting, which is useful mode in, I mean, in general for machine learning applications.

(Refer Slide Time: 16:09)



So, the next thing that we are going to look at is issues or concerns with PCA. So, we will, point out two main issues or concerns with PCA and then try to fix both of them in the course of time, and the first issue , it is not like a problem, it is but then it is like a concern definitely is the time complexity. So, how much time does it take to run PCA?

So, we have an algorithm. As computer scientists typically, you should care about how long would, it take to run this algorithm. So, what is the most time consuming step in PCA? Well PCA is just constructing a covariance matrix and then computing the Eigen directions of it. So, finding the Eigen vectors is the time consuming step of finding the Eigen vectors and Eigen

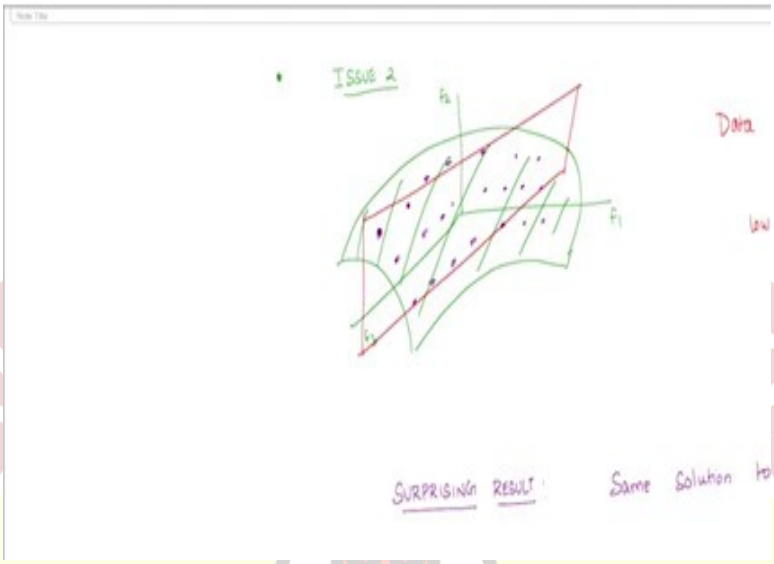
values of covariance matrix. Now, there are a lot of special purpose numerical linear algebra solvers, which can perhaps do this in the most efficient way possible. But what I am going to suggest now is a general time taken for finding the Eigen values and eigenvectors of any matrix, in particular the covariance matrix.

So, we care about the covariance matrix, which is a $d x d$ matrix. And typically, let us say typically, because, of course, there might be special purpose softwares which do this faster, but usually it takes order of d^3 time, so, this is the time complexity. If you are not familiar with order notation, that is fine.

So, you can think of this as d grows , d is the dimension of the data set, , the time taken grows cubic in d. So, it would be order of d^3. So, this is a problem this is an issue when d is large. So, when d is large, so, question is if d the number of features is large, for instance, in the Eigen faces applications that we saw the number of features was around 32,000 the number of data points was smaller than the number of features but only perhaps 20,000 or something like that.

So, if the number of features is large, d is orders of 10s of 1000s or even millions, then d^3 is going to be a million cube, which is huge, perhaps huge for computing these Eigen vectors and Eigen directions. So, an example of where this can happen is face recognition. Eigen faces, so, that we saw where the number of features was 30,000 this is problem number 1. So, we would not really say at this point how to fix it. We will talk about this in a minute, but then I will point out one more issue and then revisit this. What is problem number 2?

(Refer Slide Time: 19:43)



Problem number 2 is an unrelated problem to problem number 1, at least at this point, we will feel like that. This is issue 2. Which is - so what is PCA trying to find PCA is trying to find some linear combination of these features, which are very important. So, if there is weight, if there is height, maybe weight plus height is a very important feature, or two times weight plus three times height is very important, and things like that.

But perhaps let us say you had weight, height and something like a body mass index or something like that. Now, the body mass index might not linearly depend on weight and height, maybe it is, it is something like weight over height squared, I might get the formula wrong, but then there is a quadratic dependency with respect to how the body mass index is related to the weight and height. So, it is not linear.

So, which means that, if we for instance, if you plot in 3-D, Feature 1, Feature 2, Feature 3, and let us say we observed that the data points fell something like this. Now, what does, what am I trying to allude to here is that it could be the case that, you have some manifold, or some curved space, which is a subset of your original space, which is R^3.

So, this is a curved space. And, all your data points lie along the curved space, let us say, now, the fundamental structure in your data is not a linear structure. Why is it curved? It is curved

because these features themselves have a nonlinear relationship. Every F 3 is F 1 x F 2+ $F2^2$ or something like that.

So, maybe in your application, that is what is the fundamental equation that relates these features or the most important equation, now this is a structure, just that PCA would not find this, which because PCA is not looking for these kinds of relationships. PCA is only looking for relations which are linear. So, how can you linearly combine?

How can you weight each of these features, add them up and then create new features? Not all applications may have such relationships. In other words, so if you run a PCA, so you might get something, you will still get some output. So, you will get let us say, if this is like a 2 dimensional manifold inside a 3 dimensional space, let us forget the technical definition of manifold. It is a curved space for us.

But then essentially, think of it as a blanket, which is curved, and all the points are on this blanket. But now if you run PCA, you are still going to get some results. So, PCA might, I mean, there is nothing stopping us from running PCA on this. PCA might say that there is some plane, which is the best plane that represents this data.

Now, if you compute your error with respect to this plane for these data points, that error is still going to be high, it is not going to be insignificant, which means that we might be fooled into thinking that well, there is a third dimension, which is also important for this data set. And we might add one more dimension of course then you will be able to exactly reconstruct this dataset if you have 3 dimensions, that is, you cannot go further I mean, you can reconstruct any data set.

So, error will be 0, but then PCA will not know that there is a fundamentally mean essentially a 2 dimensional structure that is, hiding in this data set. So, the problem though is the data may not necessarily lay or lie in a low dimensional and the key word is not just low dimensional, low dimensional linear subspace.

So, when I say subspace, I just mean can think of it as 2 dimensional planes passing through origin, or line passing through origin and the higher dimensional analog of those. So, both of these issues that we have identified are relevant issues. One is time complexity, when the number of features is large you have a time complexity issue.

The second is the features themselves are non-linearly related. Then also you have an issue. So both of these cases, in one case, PCA will work but then will take a lot of time. In the other case, PCA will work but then may not give you the desired result that you really need. Here is a surprising fact. And this is perhaps the punch line of this video is that we will have the same solution to both these issues.

At this point, this might seem odd. So, these two issues do not seem related at all. One is the time complexity issue, the other is a question about the structure. And, the surprising thing is that if you solve one, you have actually solved the other as well, in the sense that you can use the solution to one to intelligently to solve the second problem as well.

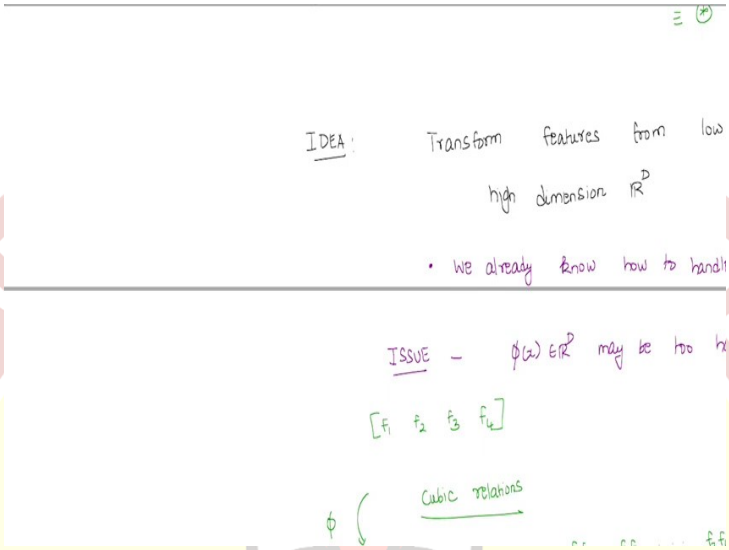
We would not break the suspense as to how that is done at this point. But what I will do is we will first consider issue 1, try to solve that and then make some observations which will help us in solving issue 2 also. So, let us start with the solving issue 1 and how to go about issue 1.

IIT Madras

ONLINE DEGREE

Machine Learning Techniques
Professor Arun Raj Kumar
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Kernel Functions

(Refer Slide Time: 0:13)

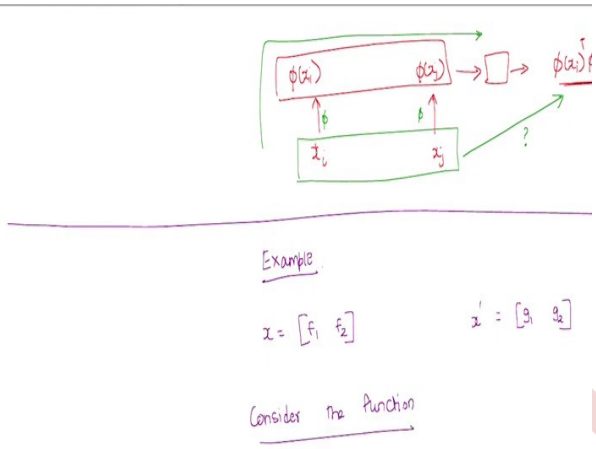


So, this is. So, this is a major issue. So, what is the issue. The issue is the following. So, then the issue is $\phi(x)$ in R^D may be too hard to compute. When you say hard it is perhaps not in the hardness sense but then the number of features might be prohibitively large that you do not necessarily want to compute $\phi(x)$.

So, the question then is well unless I compute $\phi(x)$ it does not look like I am going to be able to apply my PCA. So, I know of course I can apply my PCA when d is much much large by looking at the matrix $\phi(x)^T\phi(x)$but then even if you cannot compute $\phi(x)$then does it mean that everything is doomed or is there a way out.

Now, here let us notice the fact that the Eigen directions that we are going to compute in PCA is only of that of $\phi(x)^T\phi(x)$ this is the matrix for which we will compute the Eigen directions. So, essentially what you really need is $\phi(x)^T\phi(x)$. So, meaning you need a matrix where you need the pairwise dot product between any two pair of items.

(Refer Slide Time: 1:43)



So, if I give you an x_i and x_j. So, what you what is essentially happening is if I give you x_i and some x_j you are mapping x_i to high dimensional space and you are mapping x_j to a higher dimensional space and then both of these are given as input and then we are essentially computing $\phi(x_i)^T\phi(x_j)$. Now, this would be our k_{ij} and we will take the , Eigen vector vectors of k. So, which means we essentially need $\phi(x_i)^T\phi(x_j)$. So, the question is this is one way to go achieve this.

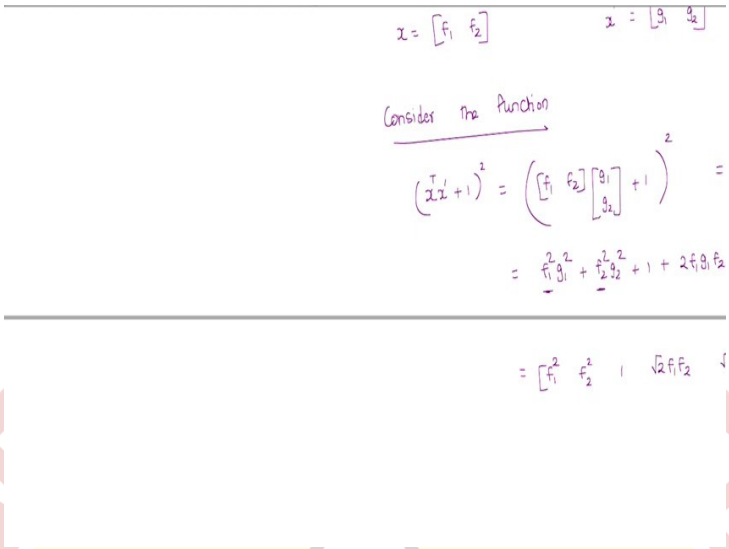
So, I this is the input x_i x_j and map it to $\phi(x_i)$, $\phi(x_j)$ with which I can compute this. Now, is there a direct way to go here meaning without going this way where I compute the ϕ first. So, I apply the ϕ explicitly compute $\phi(x_i)$, $\phi(x_j)$ and then compute the dot product is there a way I can directly compute the dot product because that is what I need anyway. So, I need to know the dot product in the higher dimensional space.

So, is there a way I can compute this dot product without explicitly computing this ϕ. So, this becomes an important question to us because unless we answer this question we would not be able to really apply PCA to you know, non-linear relationships that matter. So, in the sense that if you really want to capture some high dimensional, some complicated p-th power relationship then you must be able to do this but how do we do this.

So, is this even possible this sounds like magic. So, without computing $\phi(x)$ how do I compute $\phi(x_1)^T\phi(x_1)$ is it even possible let us ponder about this for a minute. So, let us take

an example. So, here is an example let us say x is [f_1 f_2] and x' is [g_1 g_2] and now let us say consider this function.

(Refer Slide Time: 4:03)

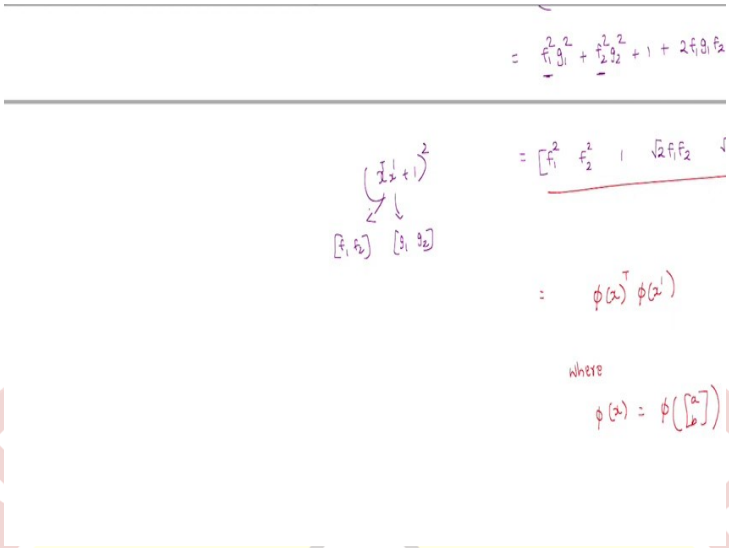


I am just pulling this function out of the hat but let us say we consider we want you to consider this function for the moment. So $(x^T x' + 1)^2$ let us say this is the function that we want to care about. So now, what is this this is just $\left([f_1\ f_2]\begin{bmatrix} g_1 \\ g_2 \end{bmatrix} + 1\right)^2$ this is the dot product. Now, that is $(f_1 g_1 + f_2 g_2 + 1)^2$ that is $f_1^2\ g_1^2 + f_2^2\ g_2^2 + 1 + 2\ f_1\ g_1\ f_2\ g_2\ + 2\ f_1 g_1 + 2 f_2\ g_2$.

So, so x is $[f_1\ f_2]x'$ is $\begin{bmatrix} g_1 \\ g_2 \end{bmatrix}$ and $(x^T x' + 1)^2$ is this complicated looking thing. Now, let us say I collect terms which are dependent on f separately and I look at them as a vector they look like $[f_1^2\ f_2^2\ 1\ \sqrt{2}\ f_1\ f_2\ \ \sqrt{2}\ f_1\ \ \sqrt{2}\ f_2]$ I will tell you why the square root I am doing is a square root here let us say this is 1 vector and now I am dotting this with another vector which is

$$\begin{bmatrix} g_1^2 \\ g_2^2 \\ 1 \\ \sqrt{2} g_1 g_2 \\ \sqrt{2} g_1 \\ \sqrt{2} g_2 \end{bmatrix}.$$

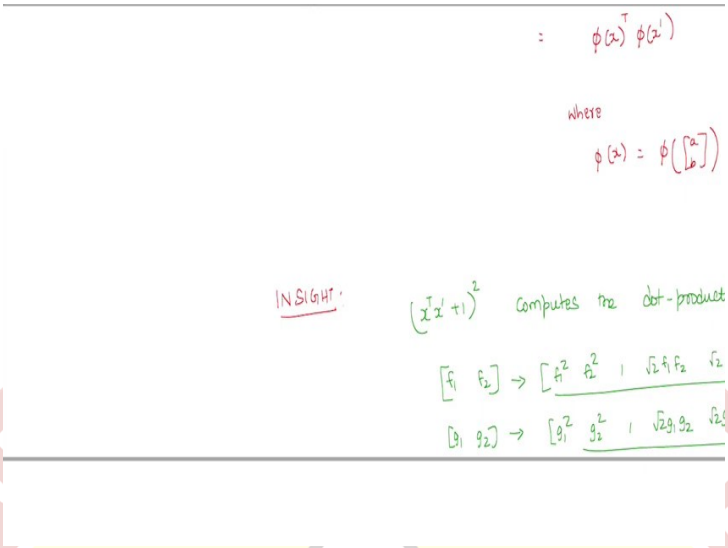
(Refer Slide Time: 5:53)



Now, you can see you can verify that this dot product is exactly the same as $(x^T x' + 1)^2$ wherever x is $[f_1 \; f_2]$ and x prime is $[g_1 \; g_2]$. Now, interestingly what we have managed to do is that these two vectors are of the same functional form. So, with respect to f and g in other words. So, if I give a general $f_1$and $f_2$then you take the first component square it second component square it then put a 1 then put a square root 2 f_1 f_2 and so on you get this mapping.

If you do the same thing for g_1 g_2 you get this mapping. So, basically I can think of this as phi of x transpose phi of x prime where phi of x equals phi of basically x$\phi(x)^T \phi(x')$ is some a,b some vector a,b is just the vector $[a^2 \;\; b^2 \;\; 1 \;\; \sqrt{2}a \;\; b \;\; \sqrt{2}a \;\; \sqrt{2}b]$ of course if a and b are f_1 and f_2 I get this vector if a and b are g_1 and g_2 then I get this vector and then I dot that I get what I want. So, basically what we are able to do. Now, is what we have been able to do is the following.

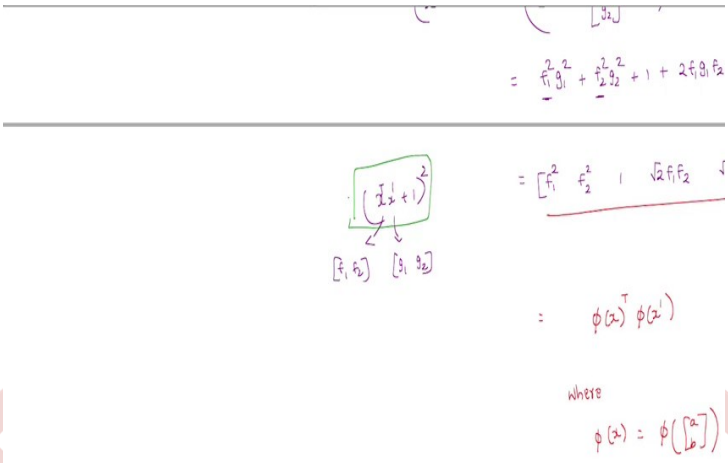
(Refer Slide Time: 7:23)



So, this function that I have asked you to consider and this is the main insight. So, at least is an example nevertheless this gives us an insight of an examples give you most insights. So, $(x^T x' + 1)^2$ is an interesting function because these computes the dot product in a transformed space.

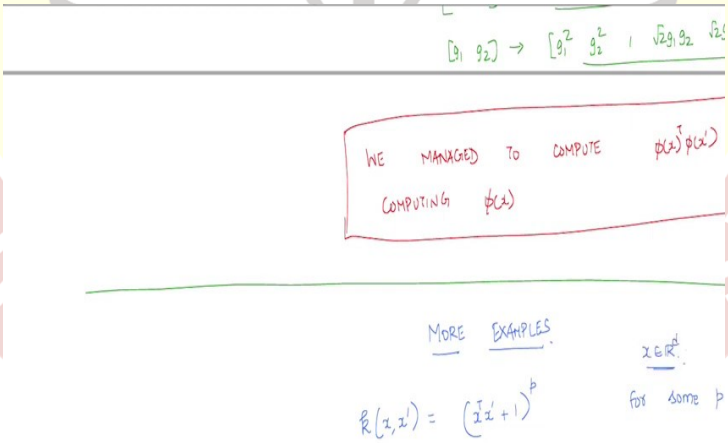
So, where $[f_1 \ f_2]$ get mapped to $[f_1^2 \ f_2^2 \ 1 \ \sqrt{2} f_1 f_2 \ \sqrt{2} f_1 \ \sqrt{2} f_2]$ and similarly $[g_1 \ g_2]$gets mapped $[g_1^2 \ g_2^2 \ 1 \ \sqrt{2} g_1 g_2 \ \sqrt{2} g_1 \ \sqrt{2} g_2]$. So, so basically, we have mapped x which is in R^2 to $\phi(x)$ which is in R^6 and then we have managed to compute the dot product in the mapped space without explicitly computing these two guys. So, I never had to compute $\phi(x)$.

(Refer Slide Time: 8:50)



So, because given x and x' I am just doing this calculation. This calculation is $x^T x'$ which is a two-dimensional vector plus 1 whole square. So now, that is a low dimensional calculation but then the number that results can be interpreted as if I did a high dimensional transformation of x and x' to $\phi(x)$and $\phi(x')$respectively and then I to a dot product in the high dimensional space.

(Refer Slide Time: 9:16)



So essentially, we managed to compute $\phi(x)^T \phi(x')$ without explicitly computing $\phi(x)$and that this was stressing. So, let me put that down. So, we managed to compute $\phi(x)^T \phi(x')$ without explicitly computing $\phi(x)$. So, this should give us some hope that we can actually you know continue this technique and then try to see if we can solve the PCA problem when

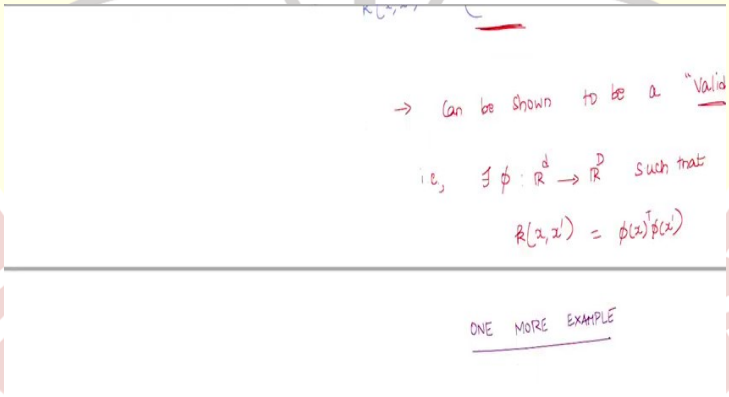
you have non-linear linear relationships among features. So, we will do that but before that. So, this is just an example that we to.

So, where $(x^T x' + 1)^2$is just one example which tries to map a two-dimensional feature to a six-dimensional space but are there other examples. So, because we have a technique or a technique to solve once you have this power to compute dot products in a transformed space it makes sense to look for other examples where you know you can do this general.

So, of course mapping in from 2 to 6 dimension is not that useful I mean I might as well do the exact mapping and then compute the covariance matrix and do all sorts of things but what would be important is if this can be done for a general d dimensional feature where you need the p dimension p-th power. So, can you do that it is the next question we will ask and the answer to that interestingly is, yes and we look at some more examples.

Now, more examples here is another example. So, let us say we have the function some function k which takes x and $x'$and then it can calculate the following x transpose $(x^T x' + 1)^p$. So, for some p greater than or equal to 1. Now, x can be in in general d dimension.

(Refer Slide Time: 11:38)



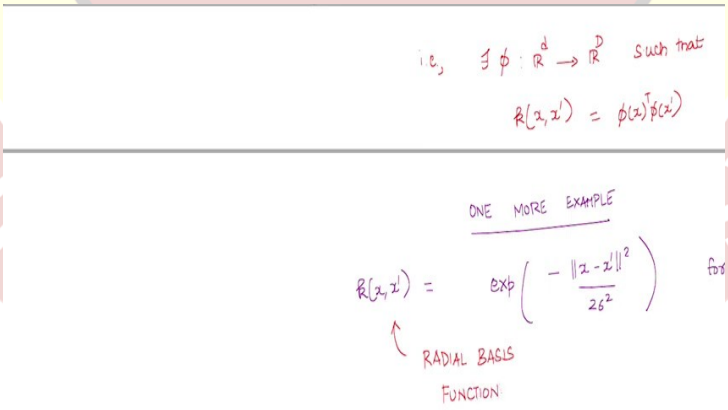
Now, the interesting thing is that and we would not really prove this but then this can be shown to be a valid map, valid function. what does it mean to say a function is valid for us. So, when will we say we are looking at valid functions well for us valid means the following. So, that means that there exists someϕ.

So, and this ϕ typically, is from d dimension to some D dimension such that you know if I just compute $k(x,x')$ which is this quantity here with same as computing $\phi(x)^T\phi(x')$ it is as if I went to this high dimension using ϕ and then did a dot product if such a ϕ exists for a k that I give you then well k is a valid function because otherwise if this does not happen I cannot rely on k to compute my K and then take its Eigen vectors and so, on for PCA.

So, this function here is in fact a valid function and it is a worthy exercise to argue this. So, I will give this as an exercise compute explicit of ϕ for p =3 and p= 4. How how does these pis look like and what happens to their dimension. So, I mean to convince yourself that this is indeed a valid ϕ, you need to comp be able to compute this ϕ. So, if you can I mean that is one way to convince yourself if you can exhibit a ϕ then that that ϕ that you have exhibited is a certificate of validity for this mapping.

So, that there is a transformation where this function is exactly computing the dot products remember not all functions are going to compute dot products in some high dimensional space. I mean I could have given you an arbitrary function but then if you want to convince yourself then one way to do that would be to actually exhibit this file exactly like how we did for p=2 when you had just two features. So, so please do this as an exercise very I mean this will reveal some insights about how the dimension increases as you increase d and p.

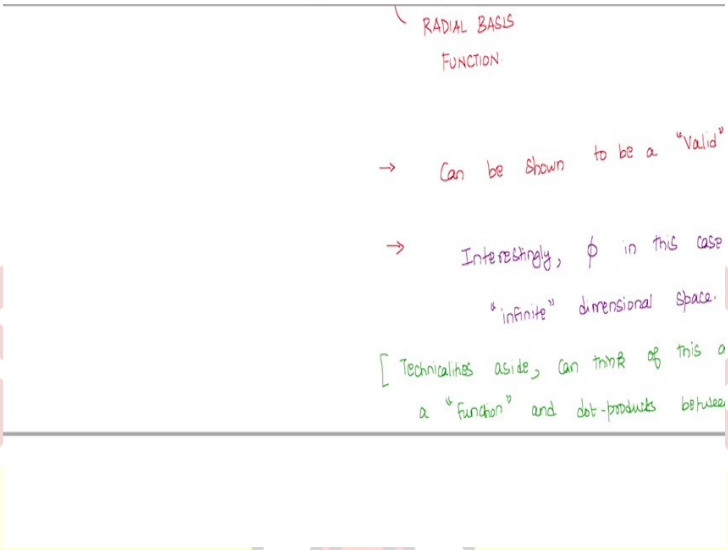
(Refer Slide Time: 14:23)



One more example and this is perhaps another famous mapping that people typically use in machine learning is the following. So, you your $k(x,x')$ in this case $\exp\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right)$ for some σ

greater than 0. So, this should remind you of Gaussian pdf in fact sometimes this is also called the Gaussian map but the standard term is also called as the Radial Basis Function.

(Refer Slide Time: 15:06)



This is the Radial Basis Function two important points about this. So, the first thing is that can be shown that this is also a valid function to be a valid map again. Valid in the sense that you can exhibit a ϕ where this this is exactly computing the dot products in the higher dimensional space but what is this higher dimension that is actually even more interesting. So, interestingly if you think about this, you might actually be able to kind of see why this makes sense ϕ in this case maps x to 1 infinite dimensional space.

So, the higher dimension that we are talking about here is actually infinite dimension. So, infinite dimensional space. So, earlier we said that the dimension where it gets mapped to increases, increases in certain fashion d to the p and so on in fact in this case you can think of I mean you can argue that the ϕ actually maps every data point on infinite dimensional space.

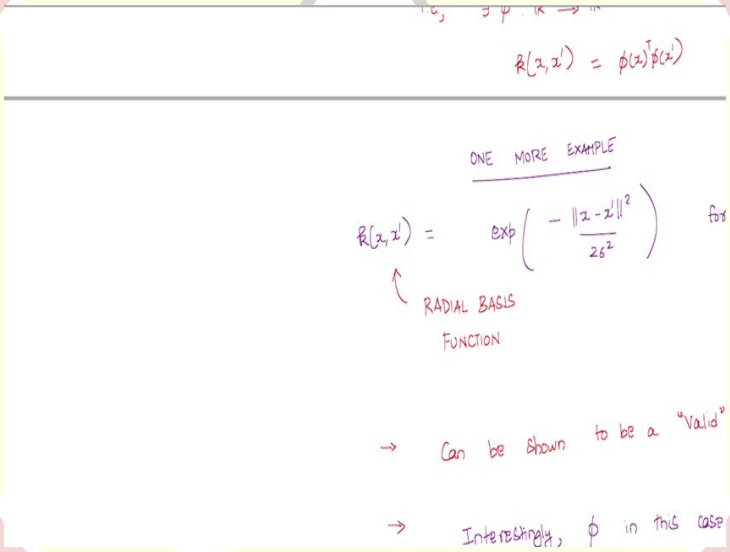
So, well whenever there is infinity involved there are a lot of technicalities but you know technicalities aside we do not want to you know dig deeper right now into the technicalities of what does it mean to say space is infinite dimension and so on. Technicalities aside what you can do is you know can think this think of this as mapping a point a data point to a function and I will tell you what this means in a minute and dot products between functions become integrals.

So, sorry because this is an infinite dimensional space you can think of it as if it is a function. So, what is a function? A function is something where you give an input and then out comes

an output and the input can be any real number whereas it's infinite of them and then in fact uncountably infinite of them and then the output is also a real number let us say. Now, you can think of the whole function itself as an infinite dimensional vector.

So, if you kind of I mean loosely put all the inputs in a vector then all the outputs also become a huge infinite dimensional vector of course you cannot put down everything you cannot write down everything if it is uncountably infinite and that is why you should think of this mapping itself as mapping a point to a function but that is just technicality. Now, how do you do dot products after you map points to functions well because this is infinite dimensional your dot products is essentially a sum but instead of sum in the continuous case it becomes integrals.

(Refer Slide Time: 18:39)

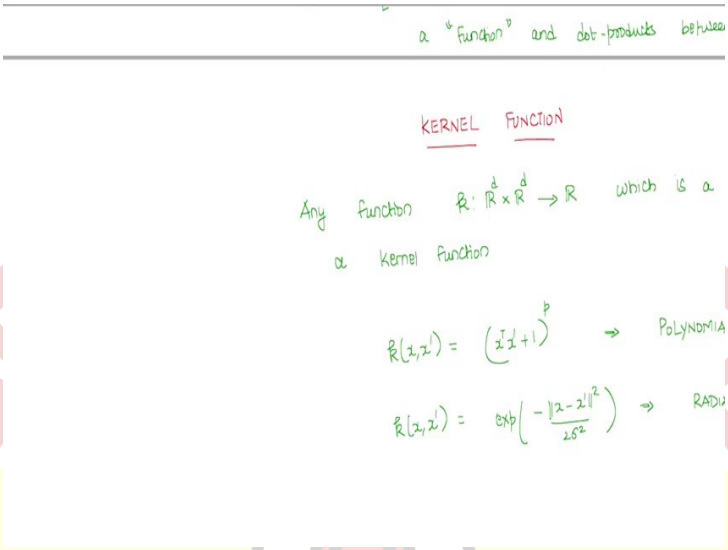


So, that is what you do to exhibit that this is in fact I mean a valid map another way to think of why this should be a valid map is to see that your e power x itself can be written as a power series expansion where you have kind of all polynomial powers. So, of course with decreasing contributions nevertheless, you have all powers embedded into e^x itself. So, in some sense you can think of it as you know mapping it to an infinite dimensional space where you have all powers contributing in different ways. So, that is one way to think about this as well.

So, what we have done. Now, is we have kind of given you multiple examples, the host family of examples, polynomial maps and then there is this radial basis map and so on but in

general can we give all these valid maps a name, yes sure we should give these valid maps a name because they will become useful and the name that is used is what is called as Kernels.

(Refer Slide Time: 19:27)



So, this is a Kernel Function. Kernel is a is a multiple , I mean it has multiple meanings in mathematics but for our purposes in machine learning we are going to think of Kernel functions as functions which are valid in the way that we described earlier that is they compute dot products in some high dimensional space.

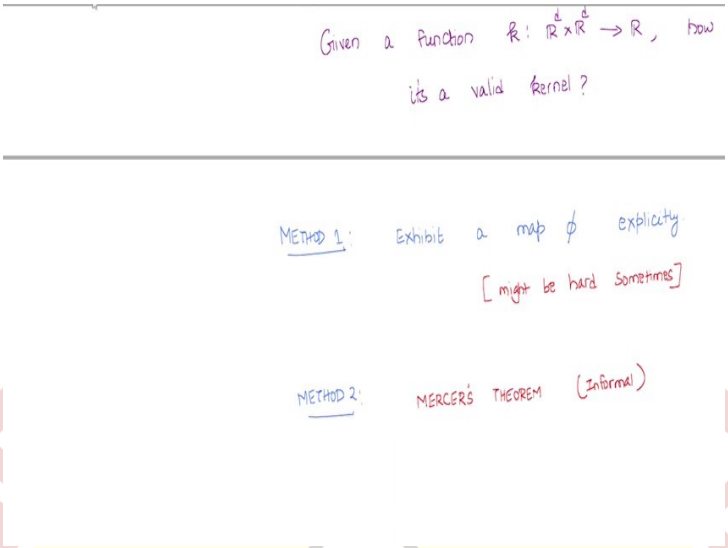
So, we write this as follows any function again loosely writing these definitions given a function k which takes 2d dimensional input and then outputs a number any function such function which is a valid map. So, I am not defining valid again that is what we did earlier that means that there exists a ϕ such that this has this dot product interpretation is called a Kernel function.

So, so what is k (x) if $k(x,x')$ happens to be $(x^T x' + 1)^p$ which is what we saw earlier. So, this is called as a polynomial curve with power p. So, if $k(x,x')$is

$$\exp\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right)$$

. Now, this is called as a radial basis or sometimes it is also called as a Gaussian Kernel. So now, these are specific examples but I think the more pertinent question is I mean it is not necessary that these are the only Kernels which are useful or relevant for our problem. So, maybe there are different non-linear nonlinear combinations which might help us.

(Refer Slide Time: 21:45)



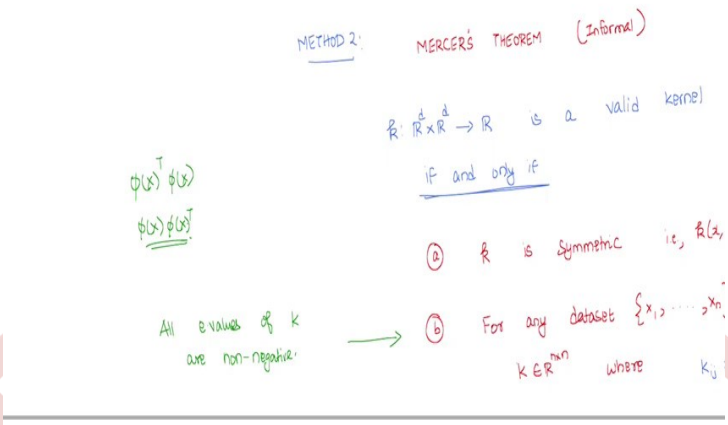
So, in general we want to ask the question given a function. So, let us say I give you a function k which takes in 2d dimensional vectors as input and then computes a number. Now, how can we say it is a valid Kernel? So, if I give you a function and I ask you well hey is I claim that this is a valid Kernel.

Now, how would you be convinced that this is in fact a valid Kernel well we have seen one method already to convince ourselves that. So, that is method 1 which is the standard method where you are going to say that hey you think this is a Kernel well which means there must be some mapping file. So, if I can exhibit such a mapping exhibit a map ϕexplicitly like how we did for the Quadratic Kernel.

So, the power 2 Kernel we exhibited that $\phi(x)$explicitly and said that $(x^T x' + 1)^2$ is actually a dot product if you think of it as this map. So, in the mapped space. So, that is one way to, one way to argue this. So, but sometimes this might be hard. So, might be hard sometimes to come up with this explicit mapping might be harder or rather hard sometimes.

So, if this is not always possible is there an alternate way we can convince ourselves that a function given function is Kernel because not all functions are Kernels. So, is there a different way to argue this, yes there is and this is given by what is called as the Mercer's theorem we will not dwell deep into this theorem in this course but then I will at least give you a informal version of this theorem.

(Refer Slide Time: 23:48)



So, so that we get the essential understanding of this theorem I would not state it in full rigorous mathematical detail but then we will give enough mathematics. So, that we know in practice what it means k which is R^d x R^d to R. I have given a function k and then I am asked if this is a valid Kernel Function is a valid Kernel, Mercer's theorem says it is a valid Kernel if and only if which is a very powerful theorem. So, that is why it is a powerful theorem because it characterizes validity of a Kernel if and only if two things happen a k is symmetric what does that mean that means that $k(x,x')$ should be same as $k(x',x)$that is this.

Now, why should a Kernel be symmetric, well we know that the Kernel is going to compute dot products in some high dimensional space which means $k(x,x')$ has to be $\phi(x)^T\phi(x')$ for some ϕ. Now, because dot product is a symmetric operation. So, $k(x,x')$ should also be same as $\phi(x')^T\phi(x)$ but then $\phi(x')^T\phi(x)$ equals $k(x',x)$. So, it necessarily has to be symmetric.

So, if it is not symmetric if I give you a function which is not symmetric in other words if you can find x and x' where $k(x,x')$ is not same as $k(x',x)$ then you can immediately dismiss my claim that k is a Kernel but let us say if it is a symmetric function then that does not immediately mean it is a Kernel already you will have to check one more condition and that condition is the following again I will write this in a machine learning way of looking at things but there is a formal way to state this as well which I am not doing .

So, for any data set let us say you consider some data set which has n points there is no restriction on what this data set should be because we could we could have gotten any data

set the matrix k which is an n x n matrix where you have k_{ij}. So, k_{ij} is the Kernel is evaluated at x_i , x_j . So, I give you a data set you are trying to claim if a k is valid Kernel or not.

Now, construct a K matrix where capital K_{ij} is the Kernel evaluated at x_i , x_j. So now, of course this is a symmetric matrix because k itself is a symmetric we have already let us say verified k is symmetric but can we say more about this matrix well the Mercer's theorem says that this matrix is positive semi-definite well what does that mean, that means that all the Eigen values so, this just means that all Eigen values of k are non-negative. So, if I give you a data set I mean you can pick any data set construct the K matrix as K_{ij}equals the Kernel evaluated at x_i , x_j.

So, that would be a symmetric matrix because Kernel is a symmetric function and now, you compute the Eigen values of this matrix and the claim is that all the Eigen values have to be non-negative. So, well Mercer's theorem says that these two conditions are not only necessary but they are also sufficient. It is slightly involved to argue the sufficiency side but at least the necessity side it is easier to argue and we will do that in a minute. For instance, we already argued why k should be symmetric it is necessary for k should be symmetric because dot product is a symmetric operation.

Now, can we argue the necessity for the Eigen values of K to be non-negative well we can do that because if I mean we need k to be of the form k_{ij} to be of the form $\phi(x_i)^T\phi(x_j)$ So, which means K should be $\phi(X_i)^T\phi(X_j)$for some matrix or some phi. So, we do not know what the ϕ is but then there is a ϕ. Now, if you look at the Eigen values of this ϕ we know that they would be same at least the non-zero Eigen values of

$\phi(x)^T\phi(x)$

.

So, $\phi(x)^T\phi(x)$the non-zero Eigen values of this is same as the non-zero Eigen values of

$\phi(x)\phi(x)^T$. But then this is just the covariance matrix or the scaled covariance matrix in the

higher dimensional space and we know that the scaled covariance matrix has the property that you know all Eigen values are non-negative.
So, simply because they compute you know, the Eigen values equal to the variance in a corresponding Eigenvectors direction and variance is a positive quantity. So, and so the Eigen values of $\phi(x)^T\phi(x)$should also be non-negative which means that if k is a valid Kernel then

the K matrix should have necessarily non-zero Eigen non-negative Eigen values it could be zero but then it has to be non-negative.

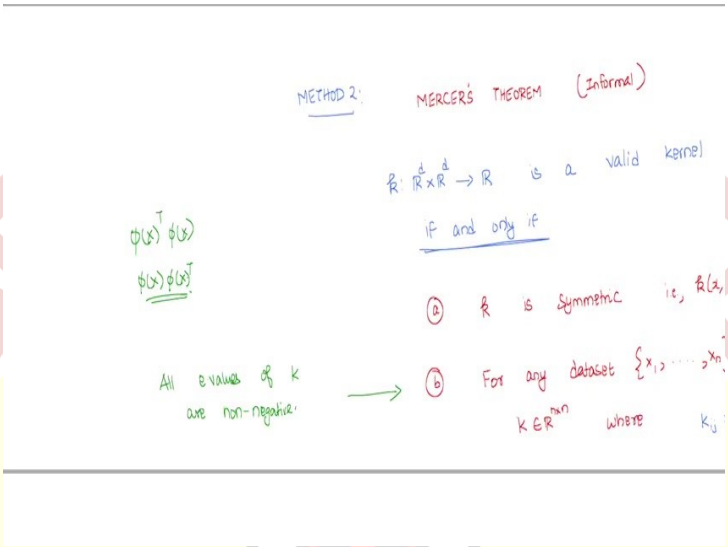
Now, this this is a argument for necessity. So, both a and b are necessity condition necessary conditions for k to be a valid Kernel but Mercer's theorem says that these are enough. So, these are also sufficient for k to be a valid Kernel. So, we would not prove Mercer's theorem but then that is important to know.

IIT Madras

ONLINE DEGREE

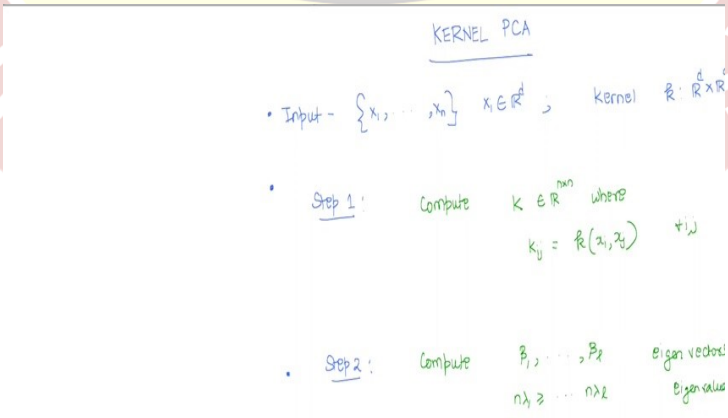
Machine Learning Techniques
Professor Arun Raj Kumar
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Kernel PCA

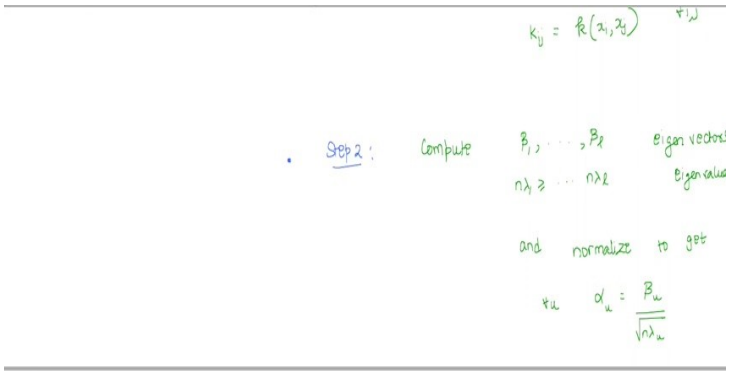
(Refer Slide Time: 0:14)



So, now putting all these things together, let us actually try to come up with the main algorithm that we wanted to do. We wanted to do PCA, but then where we have these nonlinear relationship between features, and we have discussed kernels as a trick or method to achieve that. So, let us put down this algorithm so that we see everything else in place.

(Refer Slide Time: 0:36)





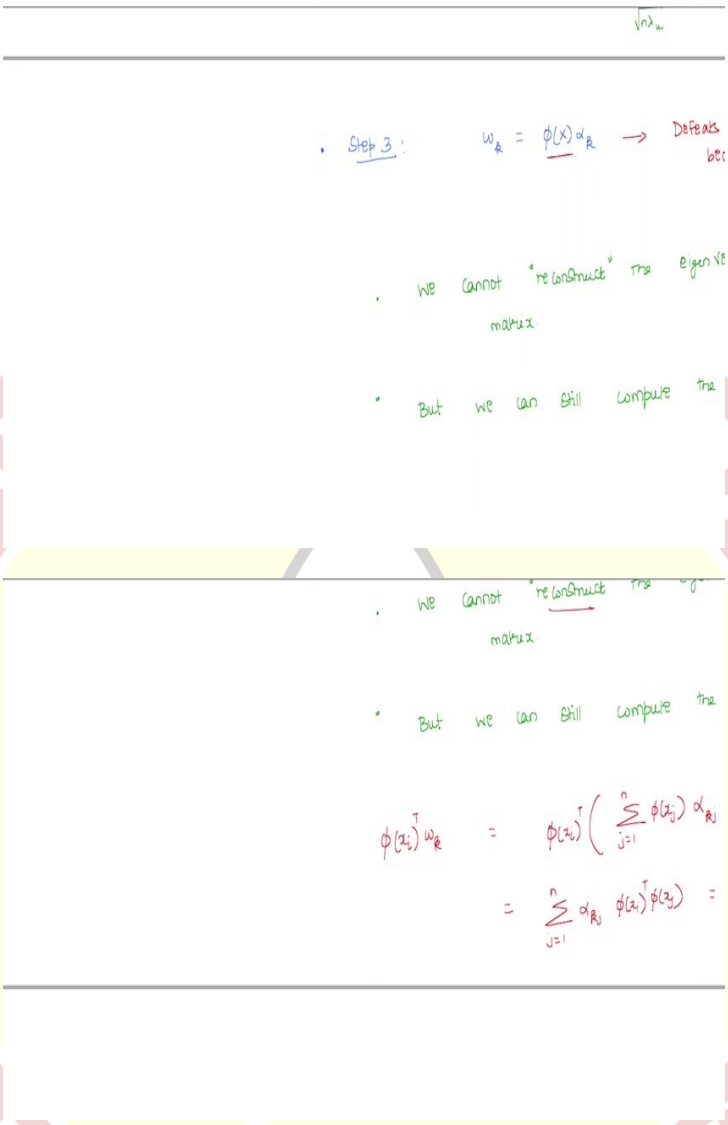
And that will make things clearer. So, this algorithm is what we are going to call as kernel PCA. So, what is the input, the input to this algorithm is as usual the data set $\{x_1 \ldots . x_n\}$ where all x_i's are in R^d. Now in addition to this, we also have a kernel function given to you some kernel K, which is which takes any two d dimensional data points and then maps it to a real number. So, that is, we know that it is a valid kernel.

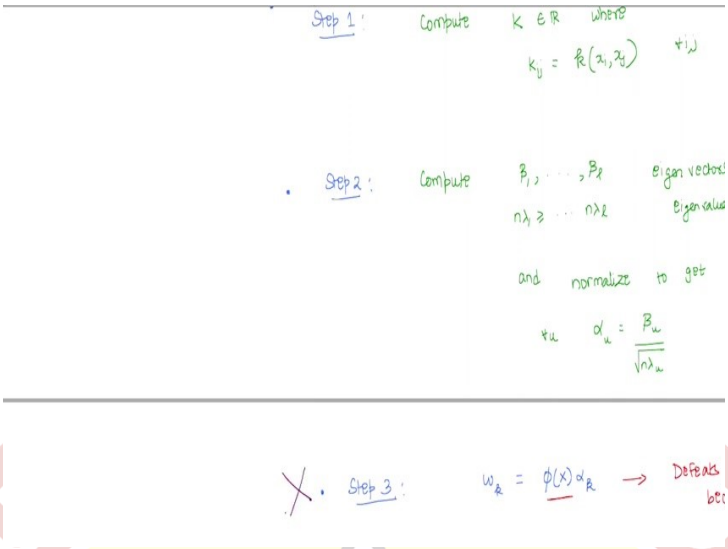
Let us say, we have tested Mercer's theorem and then argued that it's a valid kernel. So, that is given to us as input. Now, what is the step one? Well, step one, you will do the following. So, compute K. So, K in n x n, where Kij equals your kernel evaluated at the point x_i , x_j, so ∀ ij. So, once we compute the kernel, then what is the next step? Well, we compute the Eigen vectors and Eigen values of this kernel matrix.

So, compute, L=let us call them β_1 to β_k, β_1 or β_k does not matter. And $n\lambda_1$ to $n\lambda_k$ as Eigen vectors and Eigen values of k. Once you have that, now, you can also normalize because we saw this last time, you need a normalization factor for βs because the length of αs is not same as one, which is what the length of β is. So, you need a normalization, you can normalize to get α_k is $\frac{\beta_k}{\sqrt{n\lambda_k}}$.

So, you do that, so you have got an α_k is for all k. Maybe I will put this as α_u ,. So, that just confused with other k that we have just an index. So, with how many of our Eigen vectors you want, so that that many alphas you get? So, what would be the step three?

(Refer Slide Time: 3:11)





Well, in the original, when we are solving issue one, we said that step 3 now, is you get w_k you got that as

$x\alpha_k$ here, it is going to be $\phi(x)\alpha_k$. But there is a problem here. So, what is this telling us? Well, if you want w_k then you have to combine your data points in the higher dimensional space using the coefficients that you get from the Eigen computation of your kernel matrix.

But, the whole point of going through the kernel matrix was to avoid the computation of $\phi(x)$in the first place, we know $\phi(x)$cannot be computed. In some sense, if ϕ is maps into an infinite dimensional space then you cannot even compute it let alone computational issues. It is simply impossible. So, now, doing this now, this defeats the purpose, so this is a bad idea, because it needs $\phi(x)$which is what we are trying to avoid in the first place.

So, this looks like a bad idea, we should not compute w_k. So, what does that tell us? Well, this tells us the following. So, we cannot reconstruct the Eigen vectors of the covariance matrix, why because you need w_k and w_k needs $\phi(x)$, which we are not, let us say allowed to compute. So, that seems like an issue.

But, the good thing is that but we can still compute the compressed representation. So, compute or obtain the compressed representation. What do I mean by this? What I mean is that what you may not be able to do is to compute the Eigen vectors of the covariance matrix, but you can still compute the dot product of a or map data points on the Eigen vectors of the covariance matrix. What does that mean?

Well, how would we actually do the compression? So, if you go back to where we started, we said that we will find these directions and then project our data points along this direction to get these dot products. So, which means that after a in case, if I had the power to compute w_k, I would compute that and then I would do $\phi(x_i)$, which is the data point transpose w_k.

This is the number that I would like to store as the kth most important number corresponding to this data point xi. But I cannot compute w_k. But do I have to is the question, so let us see, this is just $\phi(x_i)^T\left(\sum_{j=1}^{n}\phi(x_j)\alpha_{k_j}\right)$ we know is just some combination of the data points appropriately weighed by alpha k. So, alpha kj this whole thing then is just $\sum_{j=1}^{n}\alpha_{kj}\phi(x_i)^T\phi(x_j)$, I am just bringing the $\phi(x_i)$ inside.

Now, this just becomes $\sum_{j=1}^{n}\alpha_{kj}K_{ij}$. Essentially, this is k of kij. Well, that we already know because now, the dot product of the map data points on to the Eigen direction depends only on the dot products among the data points themselves, which I know already how to compute using my kernel function, I can compute this, this is a good. So, because we cannot of course, reconstruct the Eigen vector explicitly.

But we can compute the dot products or the projections onto this Eigen vectors in the high dimensional space. So, now, typically, the reason why you do kind PCA in general is that, you get these projections and then you map you throw away the original data points and only retain these projections along each of these data points.

That is where the dimensionality reduction happens. Original data point was 100 dimension but then you only care about the top 5. So, you only you map the 100 dimensional data into a 5 dimensional projection onto these 5 most important directions. So, that becomes a 5 dimensional representation of your 100 dimensional data.

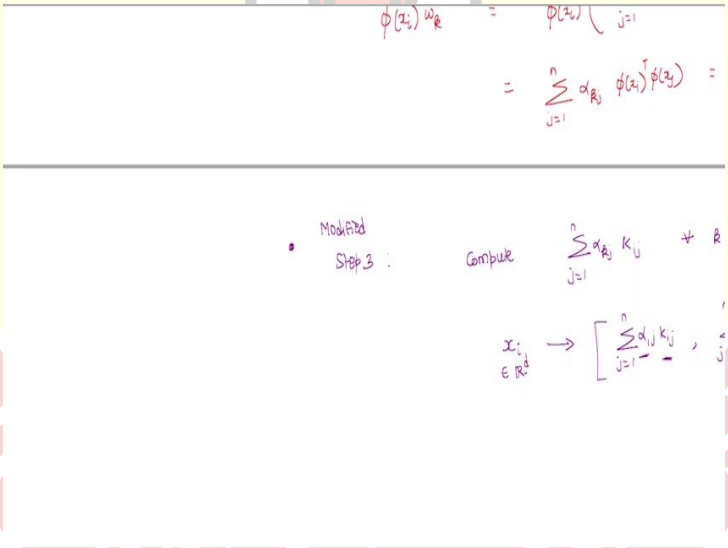
Now, that you can do even here, so though you cannot compute on what it is being projected on, you can compute the value of this projection. That is what we are essentially saying here. So, the reason why this is that, so let me make a note here for downstream tasks. So, if you want to do this as a pre processing step, for some other, so once you learn that a presentation in a low dimensional space, as just the projections onto these Eigen directions.

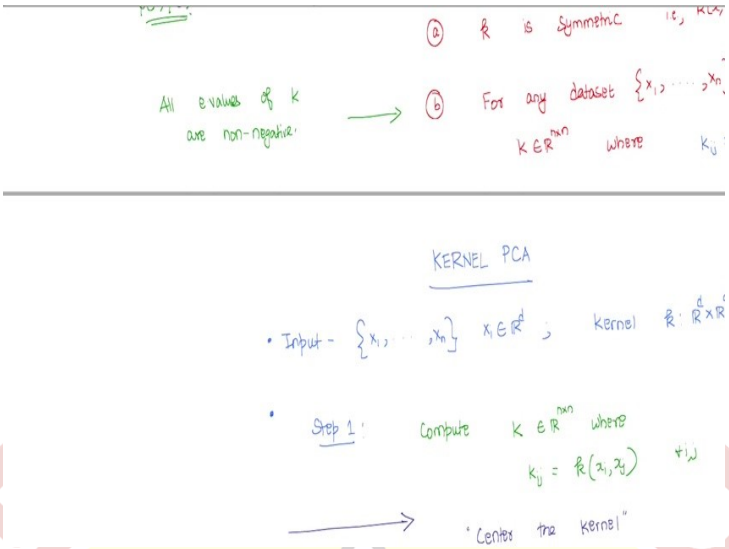
You can use these projection values as, I mean, you can imagine that this is your data point. And then work with this for a downstream task, supervised learning task, which we will see later. So, this is for downstream tasks, this is usually good enough. So, you do not necessarily need to compute the Eigen vectors.

So, you would need the Eigen vectors only if you want to reconstruct the proxy. So, how would you reconstruct the proxy if I give you the projections? Well, you will multiply the projection with the corresponding Eigen vector and then add them up. So, for that, you need the Eigen vector. And if you need the Eigen vector, then you need $\phi(x)$, we have to let go of that. So, we would not be able to reconstruct the proxy in the high dimensional space.

But still, we will be able to store these projections and these predictions are typically good enough for downstream tasks. So, typically, what you do then is just, do not do step 3. So, you cannot do step 3. Instead, your step 3 would just be the following.

(Refer Slide Time: 9:42)





So, modified step 3, well, your modified step 3 would be to compute $\sum_{j=1}^{n} \alpha_{kj} K_{ij}$ for all k, so these are your numbers. So, basically you have your x, which is, f which is an R^d. Now, what you are doing is you are mapping it to the following. So, you are mapping it to $\sum_{j=1}^{n} \alpha_{1j} K_{ij}$, $\sum_{j=1}^{n} \alpha_{2j} K_{ij}$……. whatever top l that you want. So, this is what xi, let us say, where xi data point becomes this, basically.

So, of course, this K ij is just the ith row of your kernel matrix, what you are doing is you are taking the ith row of your kernel matrix, which kind of tells you how similar is the ith data point to each of the other data points with respect to this kernel, and then you are weightinh the similarities in different fashions given by these alphas, and then storing them as the most important, information corresponding to this data point.

So, from d dimension, you can reduce it to l dimension, where l is, I mean, you can either reduce or increase does not matter because you are going to a nonlinear higher dimensional space and then reducing the dimension. So, you might go from 2 to 100 dimensional space, and then reduce it to 10 dimensional space.

So, it might seem like you are actually increased from 2 to 10. But then this increase is necessary to capture the nonlinear structure of your data, it is a low dimensional representation of the maps data points, but which means it might actually increase the original dimension to a higher dimension, nevertheless, it is it might allow you to capture more complicated relationships.

So, this is the general idea of kernel PCA, there is one small important detail that we kind of missed out on that detail, comes somewhere here. In the original PCA, we assumed that we are given a data set and then you center this data set and then compute the covariance or the kernel matrix and then compute the Eigen vectors.

Now, here what we are seeing is that we are given a kernel, if the origin of the data is set to centered fine, we apply the kernel which means it is mapping it to a higher dimensional space using some mapping ϕ. But now, if you think of this data set $\phi(x_1)$,

$\phi(x_2)$

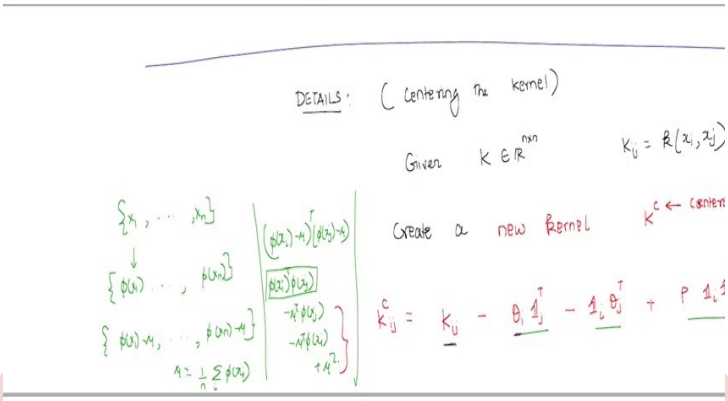
till $\phi(x_n)$ that may or may not be centered. So, that depends on your mapping kernel mapping.

It is a valid kernel, which means there is a ϕ, but then there is nothing that we are seeing that is that will make sure that the ϕ that we have which result in a map data, which is also centered, but then we need the center data only then our directions really make sense. So, we need to center the kernel. So, there is an extra step that is needed here, which is center we need to center the kernel.

So, which means we only have kernel cannot compute $\phi(x)$. So, is there a way to compute slightly different kernel, which will do the same mapping $\phi(x)$, but then after mapping $\phi(x)$, it will also center the data. So, is there a function that we can create which will so I give you a kernel, now that kernel will correspond to a ϕ map.

So, you want to apply that ϕ map, so you use the kernel, but then you also want to do the centering after you apply the phi map which means, can we mimic these two steps of going to a higher dimension and then doing the centering using just a single function kernel function, which is called as centering the kernel, so that that becomes important.

(Refer Slide Time: 14:04)



So, I will just give the details but then these are just details. But, I mean, for the sake of completeness, I am going to give this that you can indeed center the kernels. So, these are details, so how to centering the kernel? The good news is that you can center a kernel without explicitly computing ϕ. So, that is the most important news perhaps.

So, given kernel matrix K, which is an n x n, so give you a kernel function from which you create a kernel matrix K for the data set that you have. So, you have n points in your data set. So, kernel matrix is an n x n matrix, where kij is just your kernel evaluated at xi and xj for all ij. Now create a new kernel.

So, you want to create a new kernel let us call this Kc, c for centered where Kcij is some function of K but then it does not use the explicit mapping $\phi(x)$. It can be done, we would not do the algebra here, but it's just for sake of completeness, I am saying this that this can be done. And it is good to know how this looks like, though it's not so important.

You take Kij, and then you do the following. I will write it down and then make a comment$k_{ij}^{C} = k_{ij} - \theta_i 1_j^T + P1_i\ 1_j^T$. So, basically, you take the original kernel value Kij and then subtract with something, I mean, you need to do some subtraction. So, there is a mean subtraction happening in the higher dimensional space. And then we are trying to mimic what would happen if you do the mean subtraction and then do the dot product.

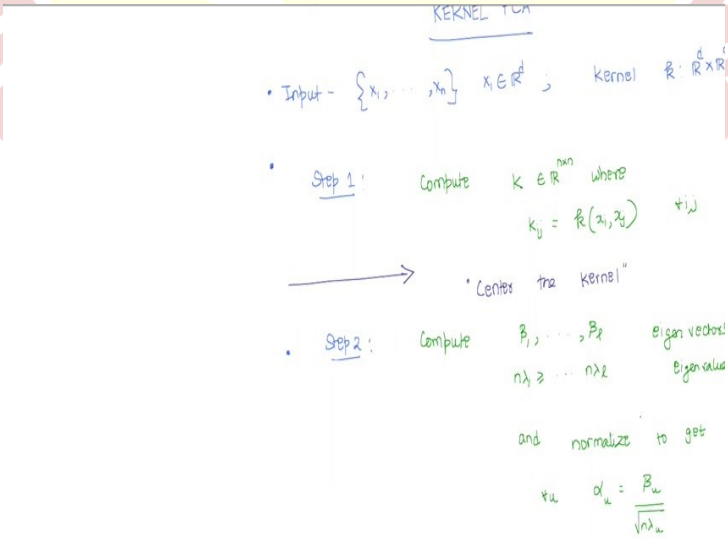
That is essentially what we are saying. So, on that dot product can be achieved by doing some manipulation of the original kernel itself. And that manipulation is, you multiply it with some valueθ_i, , which is just the average row sum corresponding to the ith data point. And then you do some manipulation, I mean, this 1i, 1j just refers to a vector, which just has one in the jth position and zero everywhere else.

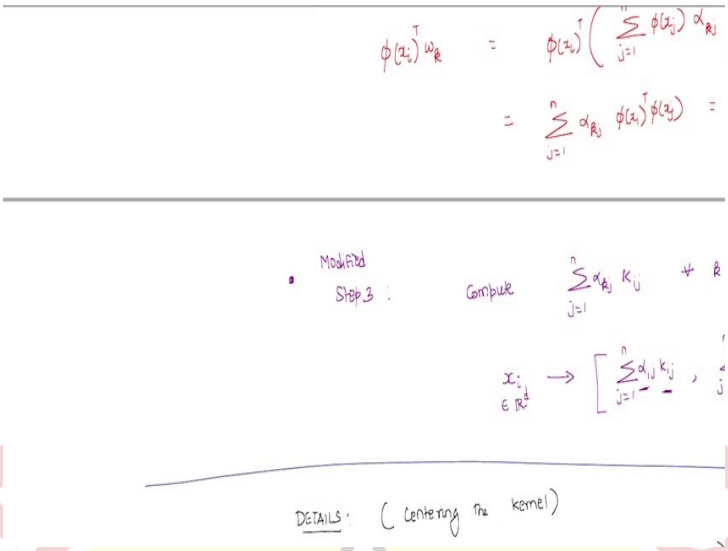
And then that is an n dimensional vector. So, you basically, well, one can algebraically argue that this is exactly same as doing, mapping the following, so you have $\{x_1 \ldots . x_n\}$, you apply your kernel, which will map it to $\{\phi(x_1) \ldots . \phi(x_n)\}$. And then what you do is in the transform space, you do $\{\phi(x_1)\text{-}\mu \ldots . \phi(x_n)\text{-}\mu\}$, where μ is $\frac{1}{n}\sum_{i=1}^{n} \phi(x_i)$. Now, this is your new data point.

Now, if you take the dot product of any two data point here, well, that is going to look like $(\phi(x_i) - \mu)^T(\phi(x_j) - \mu)$. So, that would be $\phi(x_i)^T\phi(x_j) - \mu^T\phi(x_j) - \phi(x_i) + \mu \hat{} 2$. So, there are four terms and those four terms correspond to these four guys, essentially, that is what it is.

So, this Kij corresponds to this, we already know $\phi(x_i)^T\phi(x_j)$is Kij. Now, I mean, it is just a matter of algebra to verify that if I set θ_i and ρas this, so then you will be able to get the other three terms as well. It is easy, I mean, it is just algebra, it is nothing, interesting going on here other than just saying that you can do this. So, that kind of completes the most important ideas for this algorithm.

(Refer Slide Time: 18:37)





And again, just to summarize, what we are doing now is that kernel PCA would be like this, you are given a kernel, compute the kernel and then you center the kernel, then get the Eigen vectors and Eigen values corresponding to the centered kernel, normalize it, get the αs and once you have the αs, do not construct the Eigen vectors instead, map it to these projections onto these Eigen vectors for each of the data points.

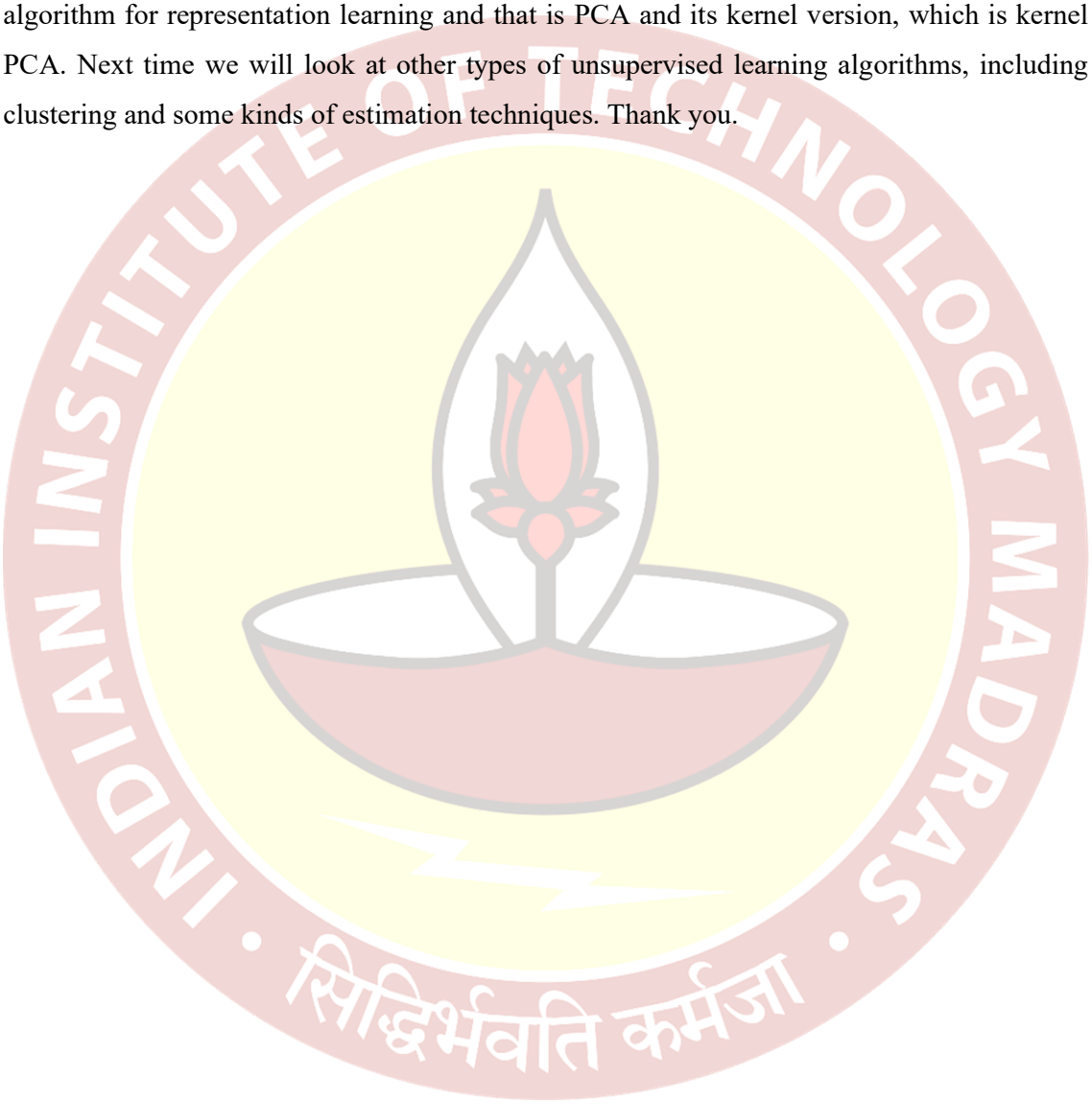
So, for every data point at the end, you have the top K projections and that projections, the set of projections is the representation for these data points as per kernel PCA. So, this is a very interesting technique. So, we have kind of used a lot of linear algebra here. But end of the day, all of this works out and we have been able to solve two important issues that we thought about for PCA, one is high dimensionality and the second is non linearity.

Surprisingly, the trick that solves both of them is the kernel matrix. You can map to higher dimension and then you can achieve nonlinear relationships as well. So, this kind of matrix is nothing specific to PCA you will encounter this object again and again, in this course, especially unsupervised learning as well, where the idea would be typically solve the problem for linear, when the data is linear has a linear relationship, which is easier problem solve. And now use the kernel trick to map it to a nonlinear problem.

So, here the kernel trick works because our whole algorithm is dependent only on pairwise products and not the exact data points themselves, if we know only the pairwise dot products, I can run this algorithm, which is essentially what the kernel matrix is doing in a high dimensional space. So, as we develop algorithms, we will also see if the algorithm depends only on the pairwise dot products.

If they do, then it is it suffices to kernelize this algorithm. Here in PCA, we observed that you can run PCA by only considering pairwise dot products and so we kernelized it and that resulted in kernel PCA. Similarly, we will kernelize slightly different other algorithms that we will encounter in this course.

But that is for later. For now, to summarize, we have a powerful unsupervised learning algorithm for representation learning and that is PCA and its kernel version, which is kernel PCA. Next time we will look at other types of unsupervised learning algorithms, including clustering and some kinds of estimation techniques. Thank you.

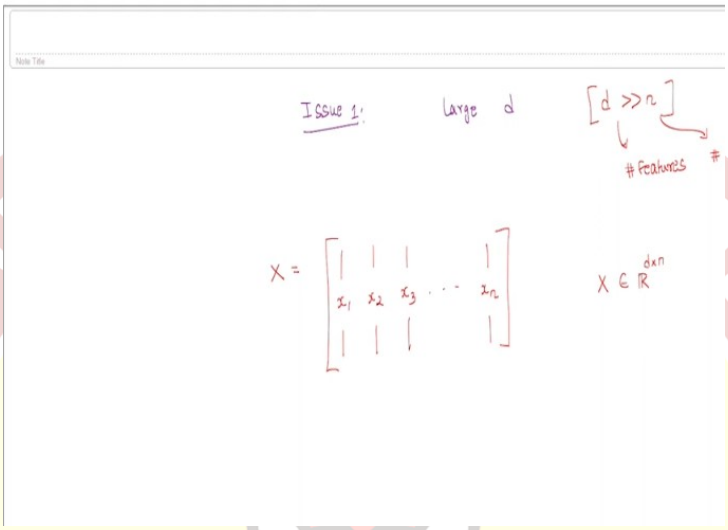


IIT Madras

ONLINE DEGREE

Machine Learning Techniques
Professor Arun Rajkumar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Time-complexity issue with PCA

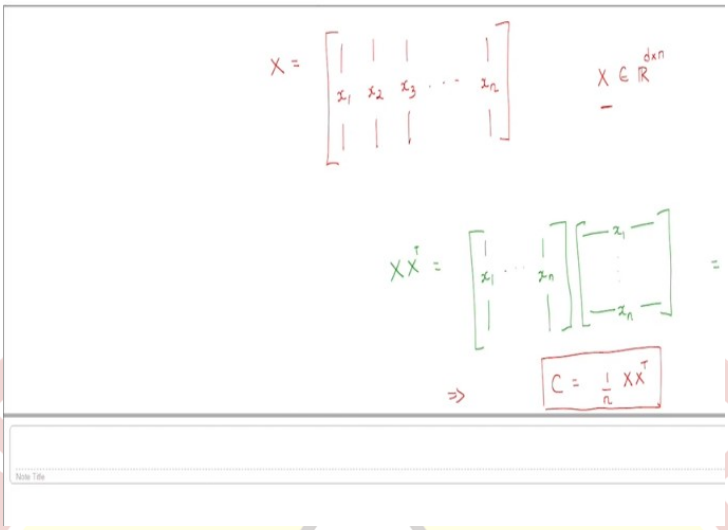
(Refer Slide Time: 00:15)



What is issue 1? It is a large d. In particular you can think of d is much much larger than n. So, d is the number of features and n is the number of data points. So, let us do some, I mean notational simplification. This will really help us, solving this issue. So, let us start by giving some, defining some matrices. The first matrix is the matrix of the data points.

Let us say we have the data set which has $x_1, x_2, x_3 \ldots . x_n$. So, let us say we stack these data points as vectors, d dimensional vectors in columns next to each other. So, that is my data set itself represented as a matrix which is a d x n matrix. Now, the covariance matrix, we know is $\frac{1}{n} \sum_{i=1}^{n} x_i x_i^T$.

(Refer Slide Time: 01:33)

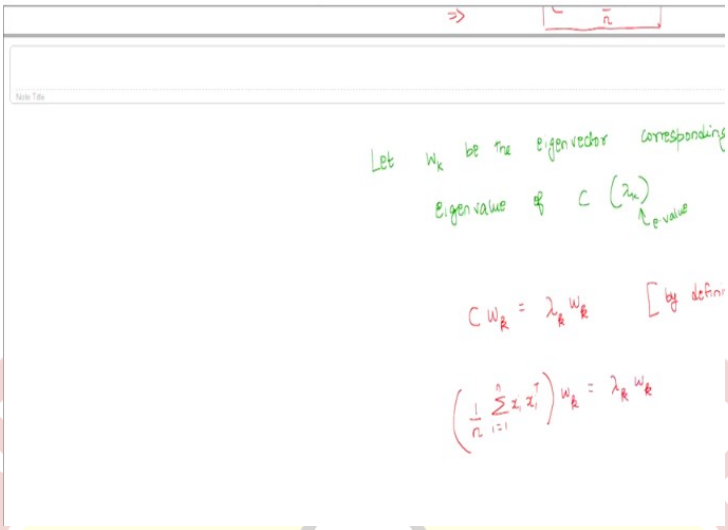


Now, I want to write this covariance matrix in terms of this matrix that I have defined here and that is possible, not too hard to see. So, if you look at XX^T, now, which means that means that you are taking a d x n matrix and then multiplying it with an n xd matrix which is the transpose of the data points where the data points are all rows now.

Now, this is exactly $\frac{1}{n}\sum_{i=1}^{n} x_i x_i^T$. So, this is perhaps you are already seen why this is the k's, if not try to show this. So, try to show this exercise. All I am saying is that your covariance matrix which is $\frac{1}{n}\sum_{i=1}^{n} x_i x_i^T$, this summation can actually be expressed succinctly in matrix notation as $\frac{1}{n}\sum_{i=1}^{n} x_i x_i^T$ transpose. That just implies that our covariance matrix is just $\frac{1}{n}XX^T$.

So, say any d check, X is d x n, $X^T$is n xd, $XX^T$is d x d which is also the dimension of our covariance matrix. Of course, we are dividing it by n that is needed I mean standard definition needs you to either divide by n or (n -1) does not really matter, we will stick to n at least in this course. So, now, what does PCA do? PCA tries to find the Eigen vectors and Eigen values of the covariance matrix or find the best-fit line which happen to be Eigen vectors and Eigen values of the covariance matrix.

(Refer Slide Time: 03:24)



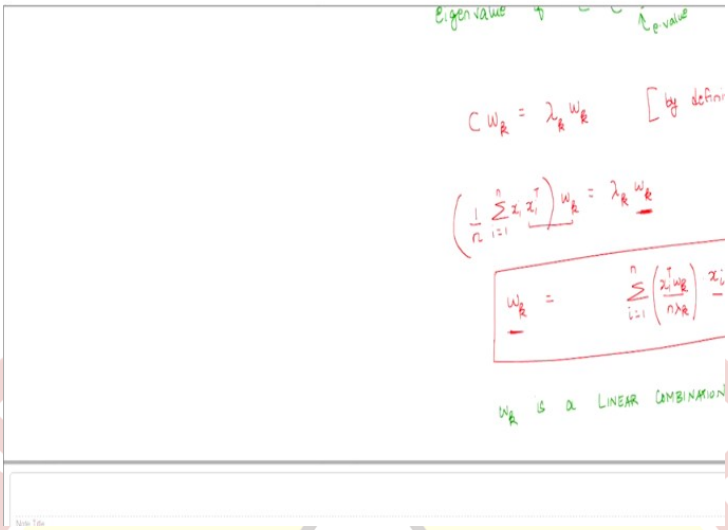
So, let us say w_k be the Eigen vector corresponding to the k'th largest Eigen value of C and let us call this Eigenvalue λ_k. Now, you have a matrix C and then I am saying that you take the Eigen values of this matrix, arrange them in decreasing order, $\lambda_1$is the highest $\lambda_2$is the second highest and so on. And w_k corresponds to the Eigenvector associated with λ_k.

Now, what is the equation that an Eigenvector satisfies and Eigenvector is a special direction for a matrix where if the matrix acts on this vector it just scales this vector by some amount. It does not change the direction. So, the direction is either scaling, it could reverse the direction. So, scaling could be negative that is still okay but it does not change the direction.

So, which means in equations, it means that $Cw_k = \lambda_k w_k$. So, that is the definition of Eigen vector and Eigen value. Now, what we want to understand is, we want to find these Eigen directions which are w_1 to w_k. Now, what can we say about these w_k's? Where do these w_k's live? Where do these Eigen directions live?

Of course, they are all d dimensional vectors but can we say something more about where these Eigenvectors live? That is what we are attempting to find now, Now, for that what we will do is we will replace C with its definition which is $\left(\frac{1}{n}\sum_{i=1}^{n} x_i x_i^T\right) w_k = \lambda_k w_k$.

(Refer Slide Time: 05:50)



Now, what I want to do is this is basically algebra, I will retain this w_k on the left-hand side, bring the λ_k to the other side and combine this x_i and w_k together. I can bring, essentially, bringing that this w_k inside, it does not depend on i so, it can go inside. So, this whole thing becomes $w_k = \frac{1}{n\lambda_k}\sum_{i=1}^{n}(x_i^T w_k)\, x_i$.

If you are not immediately seeing why these two equations mean the same thing, pause this video, just work it out it is. I mean you should be able to convince yourself that these two things are exactly the same. It is just one-step of algebra. Now, mean if you stare at this equation for a while, something interesting becomes clear. So, this is kind of seeing.

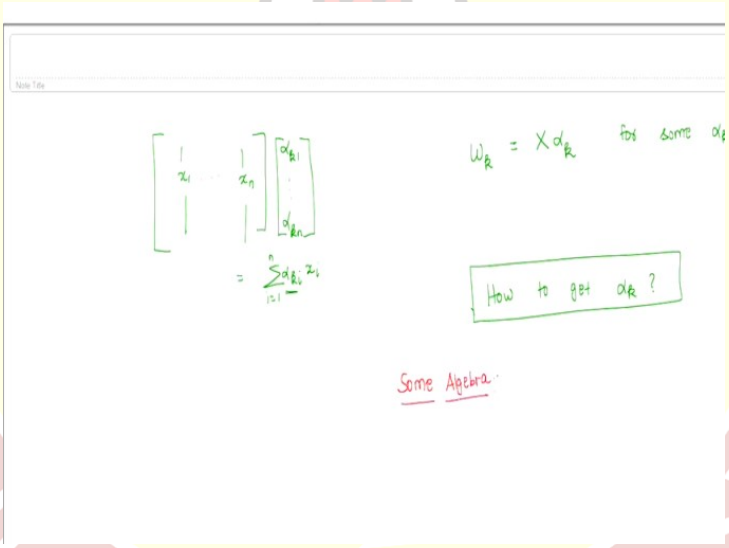
Now, let me, actually, even I can put this in λ_k inside. So, it does not really matter. So, I will tell you why I do this, yeah. So, both are exactly the same. It is a constant. I mean it does not depend on I, you can pull it inside. Now, what is this saying? This is telling me that if I want the k'th Eigen direction, so, Eigen direction corresponding to the k'th largest Eigenvalue.

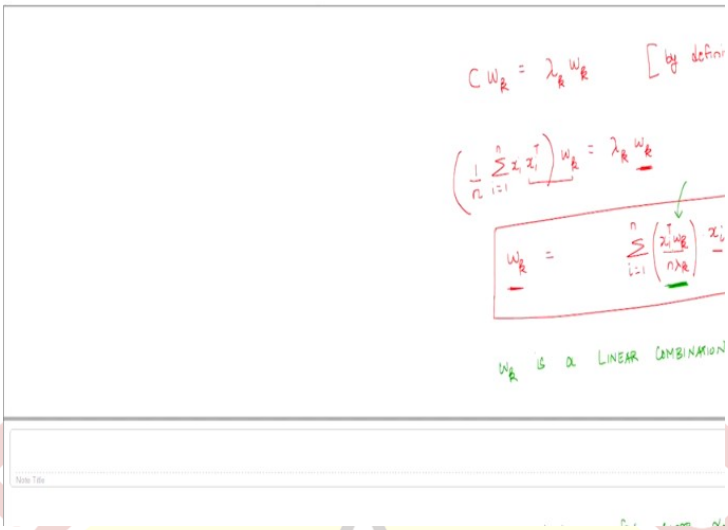
Now, I can express that as a summation of some constant times x_i which means I take the x_i data point multiplied with some number and then add up all these scaled versions of x_i. In other words, this is essentially telling me that my w_k is a linear combination of data points. So, we are going to assume λ_k is not 0 that. So, let us do that without loss of generality in this particular k's. So, because otherwise this is going to not, I mean, this is going to be infinity and then that would not make sense. But for $\lambda_k \neq 0$ this is true.

Now, the interesting thing is that, w_k is a linear combination of data points. Which means that somehow you need to combine your data points to get your Eigen directions. Now, for different Eigenvectors the way you combine these data points are going to be different. Now, what we care about in PCA is to get these Eigen directions which means now we can say that equivalently we can care about getting these combinations of these data points which give these Eigenvectors.

Because once I have the Eigen vectors then I know how to get a compressed representation and all that, I can project each data point on to the Eigenvectors we know that. We know what to do once we have w_k's. But now, to get these w_k's itself what we are saying is that it suffices to find that combination of these data points that will give me the w_k's. So, what does that mean?

(Refer Slide Time: 09:16)





That means that we can say our w_k equals our matrix X, which is remember the matrix of the data point stacked in columns multiplied by some α_k for some α_k in R^n. Now, what does this mean? This simply means that remember, our data point is like this. So, x_1 to, sorry, there matrix X is like this x_1 to x_n.

Now, I am saying there is some α_k which is in R^n which means that it gives some weight to each of this data point. The k says that it corresponds, these weights corresponds to the Eigenvector w_k. So, α_{k1} to α_{kn}. Now, if I multiply this, this is just $\frac{1}{n}\sum_{i=1}^{n} \alpha_{ki} x_i$.

Of course, we know what this α_{ki} is. So, this equation tells us that these weights are exactly $\frac{(x_i^T w_k)}{n\lambda_k}$. But then to get these weights from this equation, it appears that you already need to know w_k. So, this is $\frac{(x_i^T w_k)}{n\lambda_k}$, you need to know $w_k$to find w. That is a chicken and egg problem. We cannot use this equation to directly find these weights because this equation needs what we are trying to find in the first place which is w_k.

So, let us leave this equation out. We are saying here that is there a different way that we can somehow find this α_k's. We first recognize that there are these α_k's which exactly is this but then let us forget that it is this for the moment. But is there a different way you can somehow get these αs.

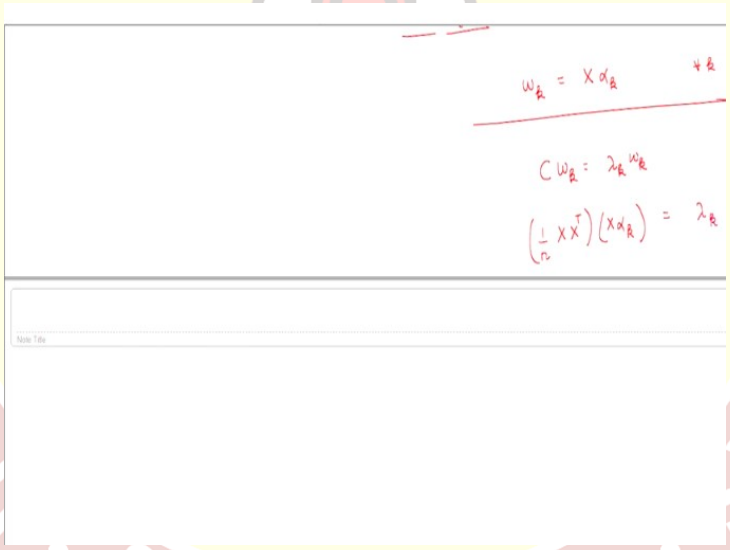
Because, if these weights, how should I weigh these data points to combine them to get my w_k that is all I need. So, I to get my w_k. So, somehow if I can get these αs then I am done.

And there is an α_k for each k. So, remember that. Because for each w_k there is a different set of weights. You have to combine the data points differently to get our w_k's.

Now, how can we get these α_k's? That is the how to get α_k's, this is the question. Once we have this, once if we can somehow efficiently get α_k's which does not require order of d^3 computation then we are in business. Because the whole point was our Eigen vector solvers are going to take order of d^3 where d is the dimension, d is the dimension or the number of features.

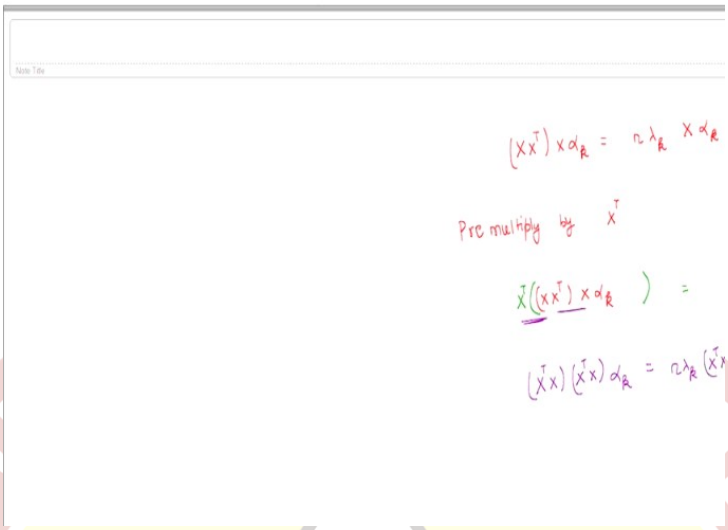
If we can somehow get α without spending d^3 time then that is a good thing to do. So, how can we do that is the question. So, we will do some algebra here. It is more the algebra I will do it for completion sake but then what comes out of it is more important. But I would definitely suggest you to try and follow the algebra as we do it.

(Refer Slide Time: 12:27)



So, we have $w_k = X\alpha_k$. We know that. We do not know what α is but we know that there will exist some α which is what we are trying to find. So, let this be right here. So, we know $Cw_k = \lambda_k w_k$. That is by definition of the Eigenvector, w_k is an Eigenvector. It has to satisfy this. We wrote C as this, $\frac{1}{n}XX^T$.this is something that we wrote earlier. We also are saying $w_k = X\alpha_k$. That is what we observed in just a minute ago. So, this is $\lambda_k X\alpha_k$.

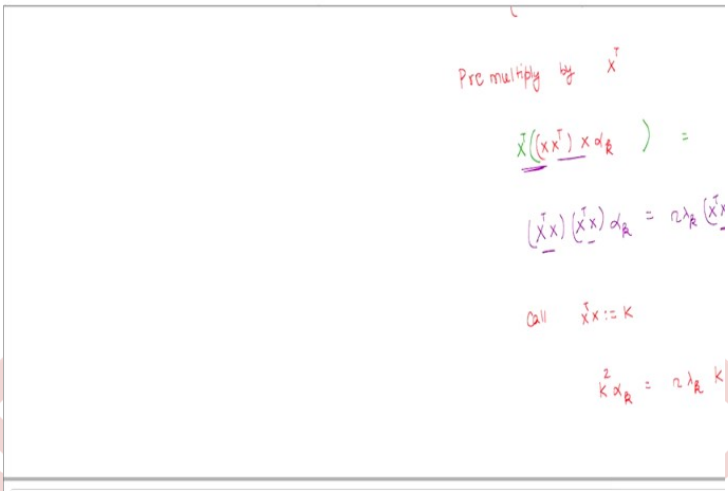
(Refer Slide Time: 13:16)



Let me bring the n to the other side and write this as $XX^T(X\alpha_k) = n\lambda_k X\alpha_k$. Now, what I would do is at this step is pre-multiply this equation by X^T. In other words, I am saying, I will do X^T times whatever was there equals X^T time whatever was there. And what is there was $XX^T(X\alpha_k)$ and we will see why this is useful in a minute, $X\alpha_k$.

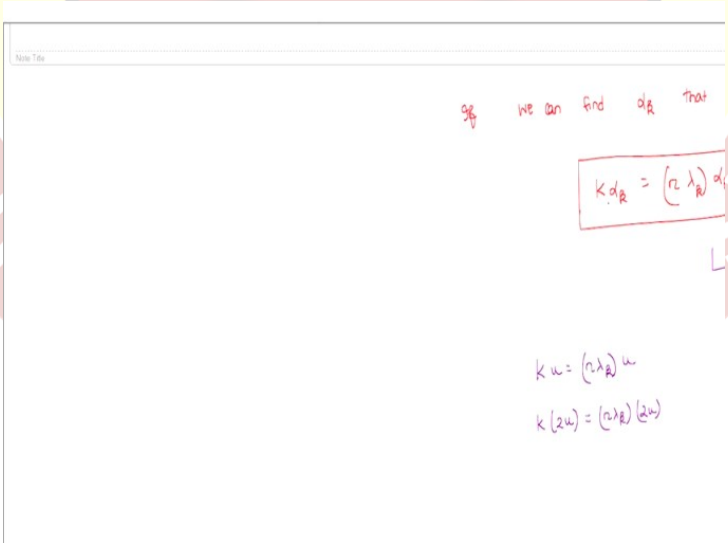
Now, if I rearrange terms, this is I can do this, I mean I cannot, matrix multiplication is not commutative always, so, I cannot swap terms but I can change the brackets. So, it is associative. So, I can change the brackets however I wish. In other words, I can do it $(X^TX)(X^TX)$. Basically, I am combining these two guys and these two guys into α_k equals $n\lambda_k$is a constant that can come outside. This X^T multiplies this X, $(X^TX)\alpha_k$.

(Refer Slide Time: 14:39)



Now, this is a matrix. Now, remember $X$was in R d x n. So, $(X^T X)$ is in, this is X^Tn xd, x is d x n, so, this is n x n. So, that is just to remember. Now, call $X^T X$ let us just give it a name, let us call it K. Then this equation is $K^2 \alpha_k = (n\lambda_k)\alpha_k$. So, we want the α_k somehow and we are saying whichever α_k that is that you need to use to combine these data points to get w_k should satisfy the equation $K^2 \alpha_k = (n\lambda_k)\alpha_k$.

(Refer Slide Time: 15:37)



In other words, if we can find α_k that satisfies, there is a k on both sides, so, I am saying, if we can find an $\alpha_k$that satisfies $K\alpha_k - (n\lambda_k)\alpha_k$then we can multiply by K on both sides and it

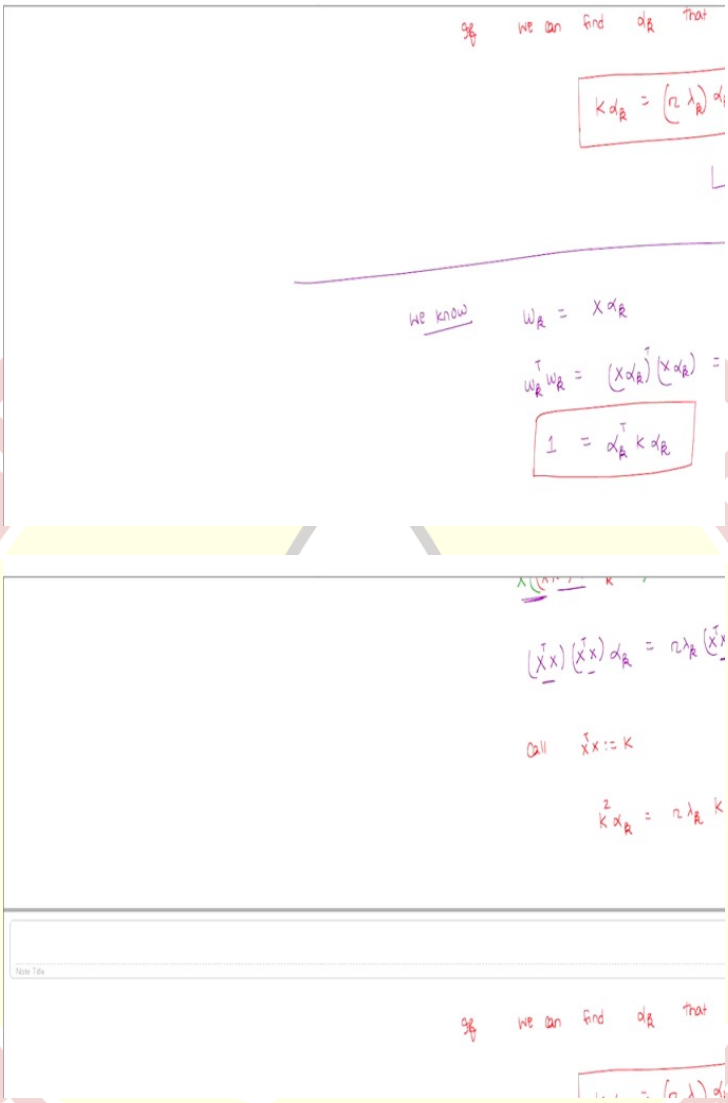
would have satisfied the other equation as well. So, which means that all we need to find is an $\alpha_k$that satisfies $K\alpha_k = (n\lambda_k)\alpha_k$. If we can do that then we are kind of done.

Now, this looks like an Eigen equation. So, this looks like an Eigen equation. In fact, this is an Eigen equation. So, basically, we are saying that, if you take α_k whatever this α_k is, if I apply this K matrix to this α_k, it has to scale this α_k by $n\lambda_k$. So, if I can find such an α_k then I am done. So, this is an Eigen equation. Now, let us say, I give you an α_k. So, I give you something and then claim that, that satisfies this equation.

So, let us say I give you some vector u and then it satisfies this equation that $Ku = (n\lambda_k)u$, let us say. Now, is this u unique? No. So, now, what I can do is I can do $K2u$will satisfy $(n\lambda_k)2u$. Now, what does this tell us that every, I mean, I can scale this u by any number and then it will still satisfy this equation.

So, which means that we need a specific way of combining these data points. So, specific alpha that will give us a w_k. We know that the length of w_k is 1. So, we were looking for directions with length 1. Now, which means that there should be something that we can say about the length of α_k also or in general α_k also. So, which means that you cannot arbitrarily scale this u that satisfies this equation and claim that all of these are α_k's. So, now, what is that? What does that tell us is what we are trying to find out?

(Refer Slide Time: 18:14)

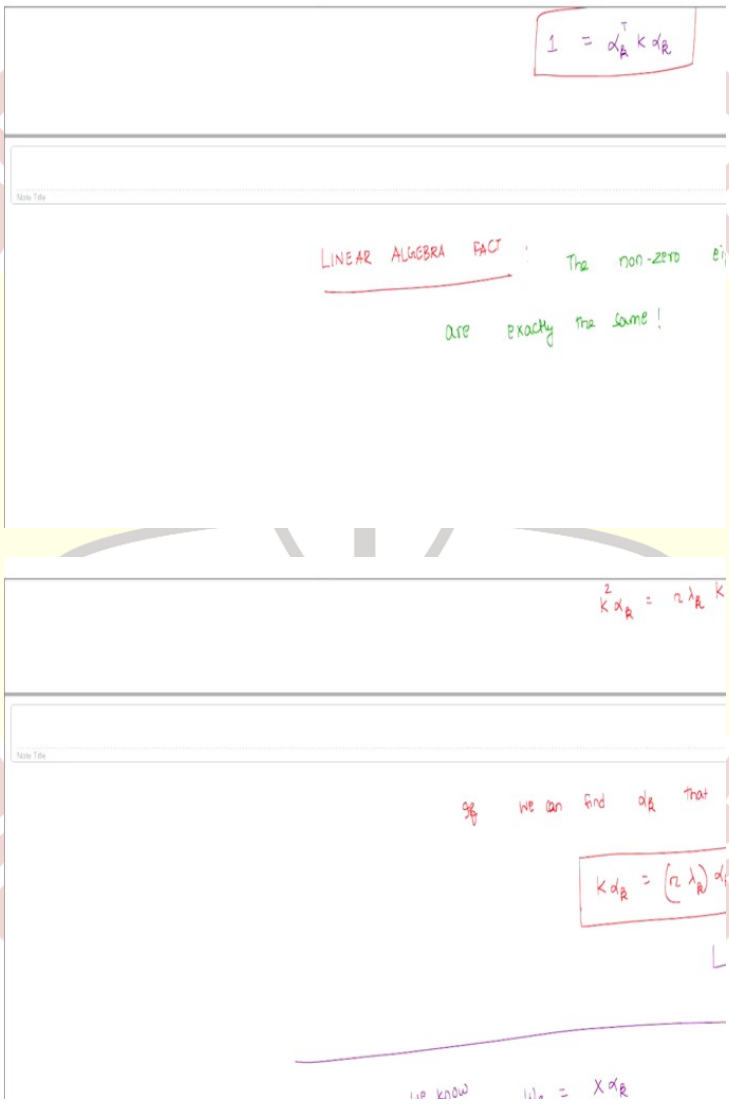


Let us try to find out how the length of w_k gives us an indication of what is the length that we should look for, for α_k. What do we know? So, we know $w_k = X\alpha_k$. We saw that. So, we know that w_k is some combination of this which means ${w_k}^T w_k$which is the length is $(X\alpha_k)^T(X\alpha_k)$. Now, this is ${\alpha_k}^T(X^TX)\alpha_k$. We know that we are looking for w_k's with the length 1 which means the left-hand side is 1. On the right-hand side, this (X^TX) shows up which is what we are calling as K. Which means this is α_k.

So, now, we are saying that we need an α_k that satisfies this equation but it can not be any α_k, such an α_k should also satisfy ${\alpha_k}^T K\alpha_k = 1$. If you find an α_k that satisfies this equation and if that α_k satisfies ${\alpha_k}^T K\alpha_k = 1$then that is the α_k that corresponds to w_k which has length 1.

So, all this is algebra. All that we are saying is that we wanted w_k, we are saying we we can equivalently solve for α_k. And solving for α_k looks like solving for an Eigen equation in K. But then the Eigenvectors length is unspecified. So, we need to normalize the length and then the fact that w_k's of length 1 implies that ${\alpha_k}^T K \alpha_k = 1$. So, we should look for α_k's Eigen equations but then you should also have this property. So, this is first point.

(Refer Slide Time: 20:42)



So, which means, so, now, we need one more important theorem from linear algebra which will actually be very very useful in understanding, in solving, in completing this problem. And that fact, I mean this is a linear algebra fact or actually a theorem but I will state it as a fact for now. And we will see why this fact is useful.

So, essentially before I state this fact, so, let me say why we need this fact. Essentially, we wanted the Eigenvectors of XX^T but now we are saying we can solve the Eigen equation of K, where K is just X^TX. Now, somehow this also involves λ_k. We need to know λ_k which is the Eigen value of XX^T. But then we are only solving the Eigen equation for X^TX.

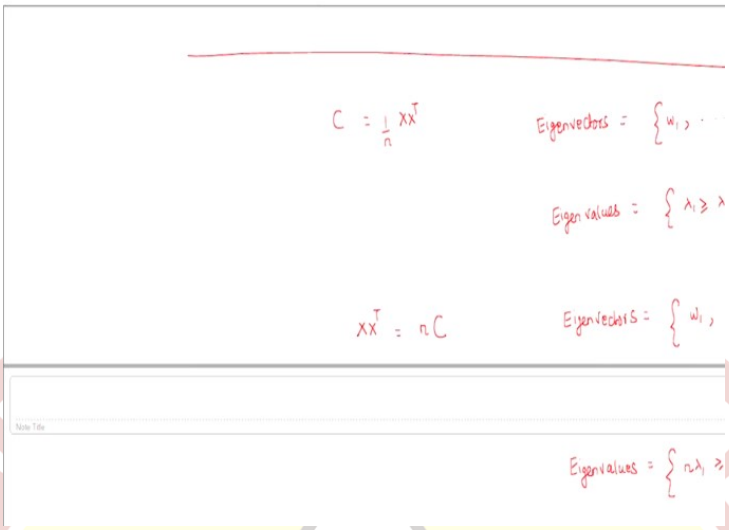
So, now, is there a relation between the Eigen values of XX^T and X^TX? So, because if we solve the Eigen equation for K that will give you a set of Eigen vectors and Eigen values, how are these Eigen values related to the Eigen values that you would have gotten had you solved the Eigen equation for XX^T. That is the question we are asking.

And the answer is this linear algebra fact. The non-zero Eigenvalues of XX^T and X^TX are exactly the same. So, now, to make it precise, so, XX^T e is a vector, it is a matrix in d xd, X^TX is a matrix in n xn. But because both of these come from the underlying x which is d x n their Eigen values are related.

In fact, if you are aware of the singular value decomposition of matrices then you can simply use that to prove this in two steps. I would not do that. Feel free to try this. But what I am saying is that the non-zero Eigenvalues, there might be 0 Eigen values because this is d x d, this is n x n, so, the maximum number of non-zero Eigen values of these matrices will be the minimum of d and n. That is again a linear algebra fact.

So, there is going to be a lot of zero Eigen values if d is large. But which there would not be corresponding Eigen values in n. So, if n is smaller than d then and if your matrix has full length, so, then your number of non-zero Eigen values will be n and they will match with the top n Eigen values of XX^T which is a d x d matrix. So, what does this essentially tell us? So, what is all of this telling us now? It is telling us the following. And this is the most important thing.

(Refer Slide Time: 23:44)

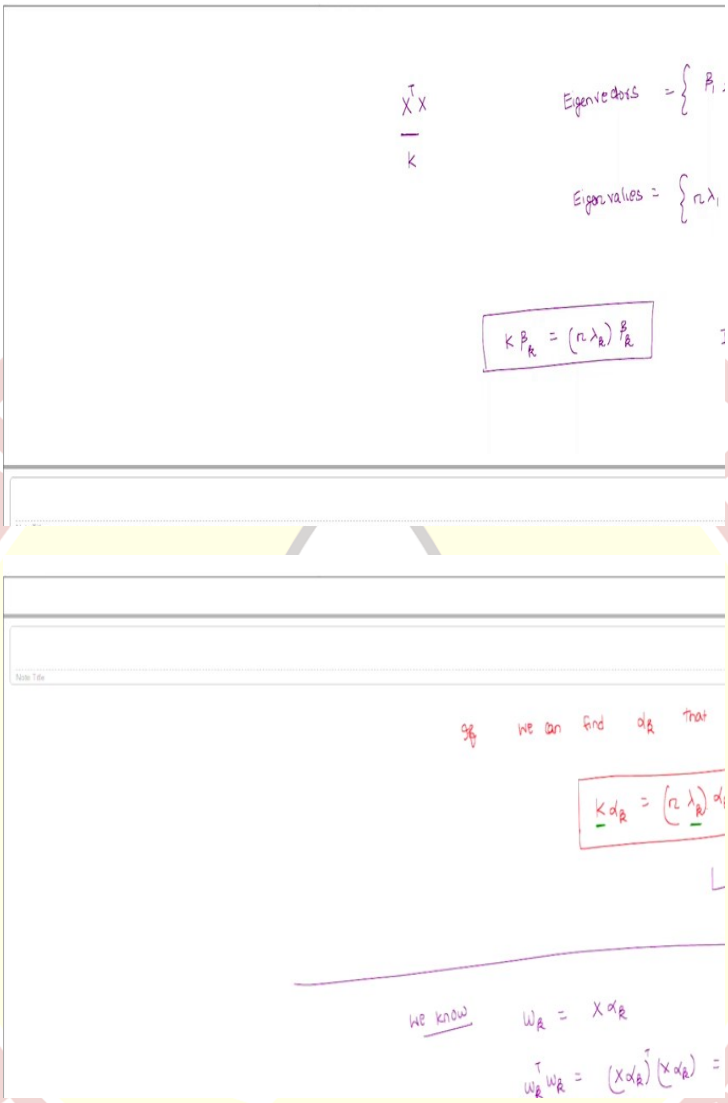


We will now try to put everything together and see all of these together. So, we initially had C which was $\frac{1}{n}XX^T$ whose Eigen vectors and Eigen values we wanted. So, let us say the Eigenvectors of C where w_1 tow_l , some w_l which we want. So, now, for all k in from 1 to l we know w k is 1, the length is 1.

Now, the Eigen values corresponding to this are λ_1, in fact, they are in decreasing order, λ_1> λ_2 >…… λ_l . So, this is what we wanted to get. Now, X, let us look at XX^T, XX^T is just nC. Now, what will be the Eigen vectors of length 1 for XX^T.

They can they will exactly be again w_1 tow_l. It is just X, the matrix is just scaled. So, the Eigen values will scale accordingly but the Eigenvectors of length 1 will stay the same. The Eigen values will nλ_1> nλ_2 >…… nλ_l. So, we need this w_1 tow_l somehow.

(Refer Slide Time: 25:17)



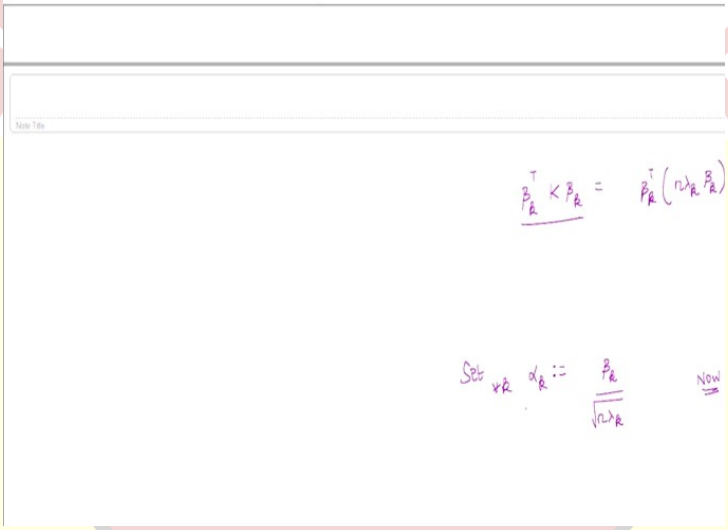
Now, what are we saying? We are saying we need to solve an Eigen equation of X^TX. This is the matrix which we care about. Now, let us say the Eigenvectors of X^TX happen to be some vectors β_1 to β_l, some Eigenvectors. Now, what we know? We know that all the Eigenvectors length is 1. All of these guys are of length 1 and let us say, now, what are the corresponding Eigen values.

Now, the fact, linear algebra fact that I stated before says that $XX^T$and X^TX have exactly the same non-zero Eigenvalues. Which means the Eigen values of X^TX will also be $n\lambda_1 > n\lambda_2 > \ldots\ldots n\lambda_l$. So, this we have. So, now, what does this mean? This means that this is K.

So, which means that $K\beta_k - (n\lambda_k)\beta_k$. So, we have found an Eigen vector β_k which satisfies $K\beta_k = (n\lambda_k)\beta_k$. Now, question is what we wanted. We wanted some α_k's which satisfies $K\alpha_k = (n\lambda_k)\alpha_k$.

Now, here is something which satisfies, we found a β_k which satisfies $K\beta_k - (n\lambda_k)\beta_k$. But, is β_k=α_k? Can we solve the Eigen equation like this and say is $\beta_k$same as α_k? Well we saw that there is a length constraint also on this α_k's. It is not just the fact that it has to satisfy this equation, it also has to satisfy ${\alpha_k}^T K\alpha_k = 1$.

(Refer Slide Time: 27:30)



So, which means, now, let us check with β_k. So, now, what is ${\beta_k}^T K\beta_k$, what is this value? If $\beta_k$was α_k then this has to be 1. But what is this? By the virtue of the fact that β_k is an Eigen vector of K, so, $K\beta_k$is $(n\lambda_k)\beta_k$. So, this is ${\beta_k}^T(n\lambda_k)\beta_k$.

But we know beta k is of length 1. So, this is just $n\lambda_k$. So,

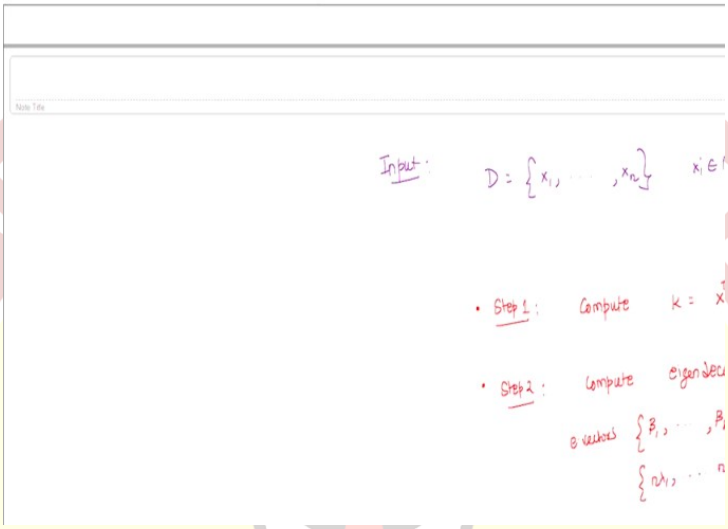
$\beta_k K\beta_k$

is actually $n\lambda_k$. But then we wanted, if we said β_k is α_k then it is not going to work because then no longer ${\alpha_k}^T K\alpha_k = 1$. So, there is a scaling of $n\lambda_k$ that is happening.

So, which means we can set α_k as $\frac{\beta_k}{\sqrt{n\lambda_k}}$. If you do this, now, if once I set this for all k, now, $K\alpha_k = (n\lambda_k)\alpha_k$ and ${\alpha_k}^T K\alpha_k = 1$. Why? Because ${\alpha_k}^T K\alpha_k$ is simply $\frac{{\beta_k}^T K\beta_k}{n\lambda_k}$. But we know the numerator is $n\lambda_k$ that is what we argued here, divided by $n\lambda_k$. This is just 1.

So, now, this is kind of telling us, how to convert your Eigen solutions for this matrix k into the α_k that we really needed. We will talk about why this is a useful thing to do. So, we are kind of going in circles and trying to find α_k but then I will tell you why this is a good thing to do in a minute.

(Refer Slide Time: 29:40)



So, now, here is what is the algorithm that we are proposing. So, our input is just the data set d which is $\{ x_1, \ldots x_n \}$, where all x_i 's are in R^d and the assumption is that d is much much larger than n. So, this is the setup that we are in if we have too many features. That is where the time complexity is a problem. Now, what are we saying?

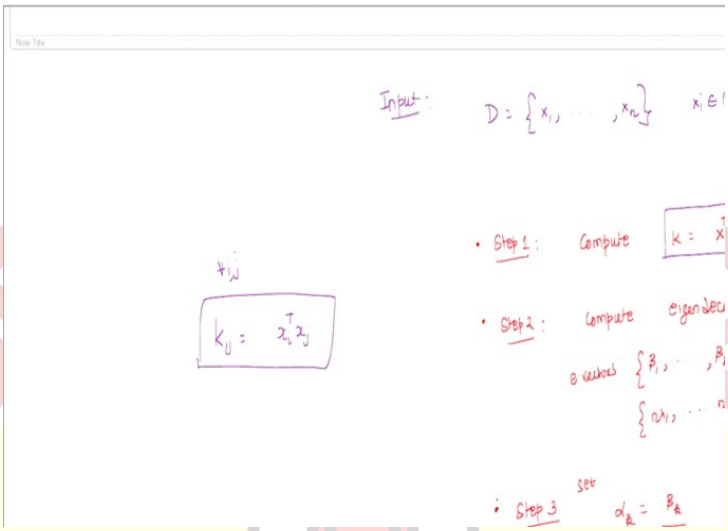
We are saying that step 1, earlier, we would have computed the covariance matrix and then we would have computed the Eigenvectors and Eigenvalues. We no longer want to do that because it is expensive. So, what is the step 1? Step 1 compute $K = X^T X$. So, K is in R^n x n. Step 2. Compute Eigen decomposition of K.

Now, this is still an Eigen decomposition. Well we wanted to avoid a costly Eigen decomposition but then we are saying in step 2, we are doing an Eigen decomposition of K. But note that K is an n x n matrix. Which means the Eigen decomposition of K is only going to take order of n^3.

And if d is much much larger than n, essentially, what you are saying is that instead of doing a d^3 Eigen decomposition you can do an $n^3 d$ Eigen decomposition. So, which might be much

cheaper, in general. So, we do Eigen decomposition of K and we get Eigen vectors $\beta_1$to $\beta_l$with corresponding Eigen values. These are Eigen vectors, nλ_1 …… nλ_l.

(Refer Slide Time: 31:44)



Once you have done this, so, this is an order of n^3 computation. So, once this is done this then we are almost done. So, step 3. We know that theseβ_k's are not exactly α_k's but then you have the Eigen values also with. So, what you can do is set $\alpha_k = \frac{\beta_k}{\sqrt{n\lambda_k}}$ for all k equals 1 to l. That is it.

So, once you have this then you can get back your w's if you want. So, $w_k = X\alpha_k$. So, essentially, what we have done is we have gone in a different row root to find our w_k's. Instead of directly solving the Eigen decomposition, we are solving a different matrix, a related matrix, the Eigen decomposition of which is cheaper.

And then we are getting the Eigenvectors and then converting that into the weights that you need to combine the data points to get the Eigen directions of the covariance matrix. Now, this is pretty much solving problem number 1, issue number 1. Because we are not working with a huge d x d matrix but then we are working with smaller n x n matrix.

Of course, this is helpful only if d is much much larger than n. If n is larger than d, you might as do the covariance matrix decomposition. But then we are in that setup where we are assuming that d is much much larger than n and then we are trying to do this. So, this solves issue 1, the time complexity issue.

Now, note that you cannot, I mean you cannot get away with the Eigen decomposition completely. So, you have to do it. But then because there are only two important parameters in this matrix, one is d, one is n. We are just trying to make it as simpler as possible by picking the smaller one of, smaller of these. So, that is all issue 1.

Now, we want to talk about issue 2 in the next video. But before going there, I would want to make one observation which I will again make in the next video but then I want to hint it at this point and then we will come back and connect it to the next video. Now, what are we saying here, we are given this data set, the first step was to compute Kas $X^T X$. So, what is k_{ij} in that case?

$k_{ij}$is $x_i{}^T x_j$. You can verify that this is what k_{ij}j would be, so, for all i ,j. In some sense, this is capturing the similarity between x_i and x_j, in the dot product sense. More importantly, this also tells us that to solve the PCA problem, you only need these pairwise dot products between the data points. If you have that you have all the information that you need to compute the PCA things.

So, whatever you want in PCA. We will make this more precise next time and then we will see that this important observation of the fact that you only need this kind of a similarity between these data points and not necessarily the data points themselves always plays a very important role in trying to use the solution of issue 1 to actually solve for issue 2 which is a totally different question. It says that what if the data points are not linearly related. It needs some creative thought to take this solution and then solve issue 2 and that is, in fact, one of the very important ideas in classical machine learning and we will talk about that in the next video.

For now, I would want to summarize saying that all we have seen today is this set of videos is to see how we have identified some issues with PCA and then how you can solve the issue of time complexity by converting your original problem into a simpler problem in a computational sense and solving the simpler problem will help you solve the original problem. We will talk more about issue 2 in the next video. Thank you.