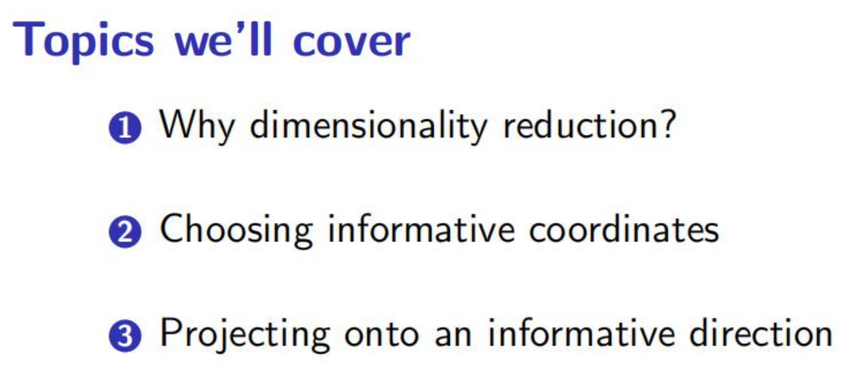
Module 9 - Representation Learning II

- 9.1 Linear Projections
- 9.2 Principal Component Analysis I: One Dimensional Projection
- 9.3 Principal Component Analysis II: The Top k Directions
- 9.4 Case Study: Personality Assessment
- 9.5 Linear Algebra V: Eigenvalues and Eigenvectors
- 9.6 Linear Algebra VI: Spectral Decomposition

总结:

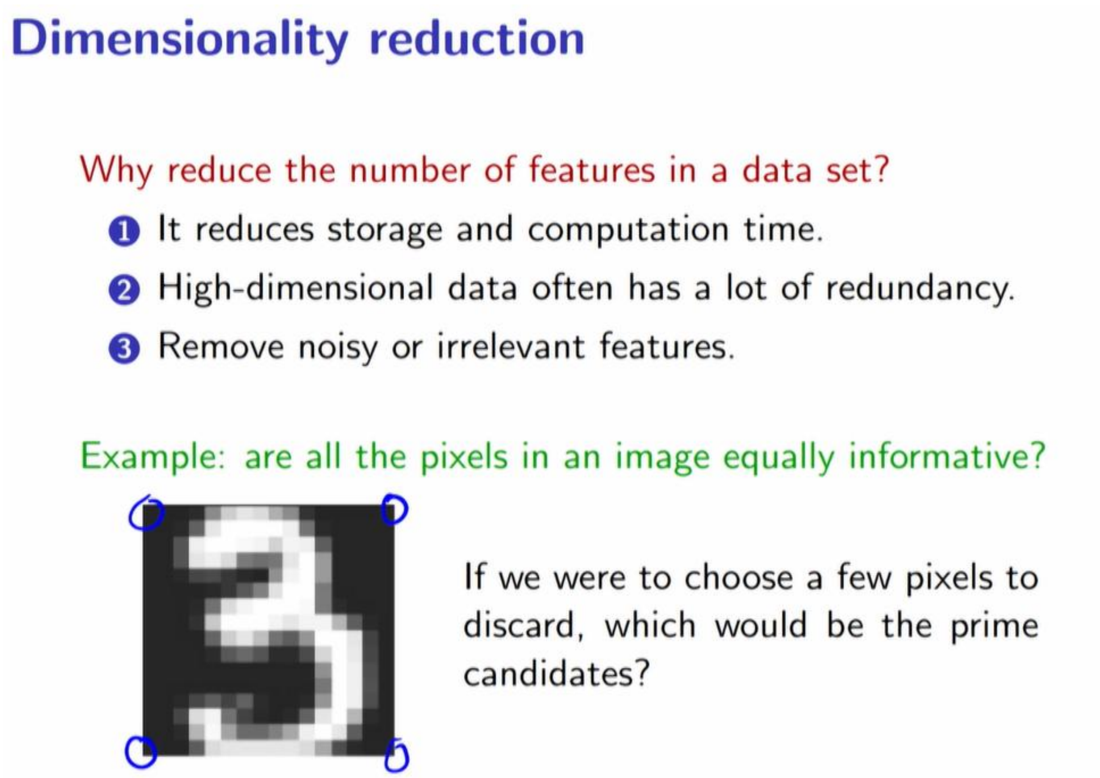
9.1 Linear Projections



Topics we'll cover

1. Why dimensionality reduction?
2. Choosing informative coordinates
3. Projecting onto an informative direction

A very effective form of representation learning is, linear projection.
Today, we'll see what that means. So we'll begin with a little discussion about dimensionality reduction and then we'll look at two approaches. Choosing informative coordinates versus choosing informative directions.



Dimensionality reduction

Why reduce the number of features in a data set?

1. It reduces storage and computation time.
2. High-dimensional data often has a lot of redundancy.
3. Remove noisy or irrelevant features.

Example: are all the pixels in an image equally informative?

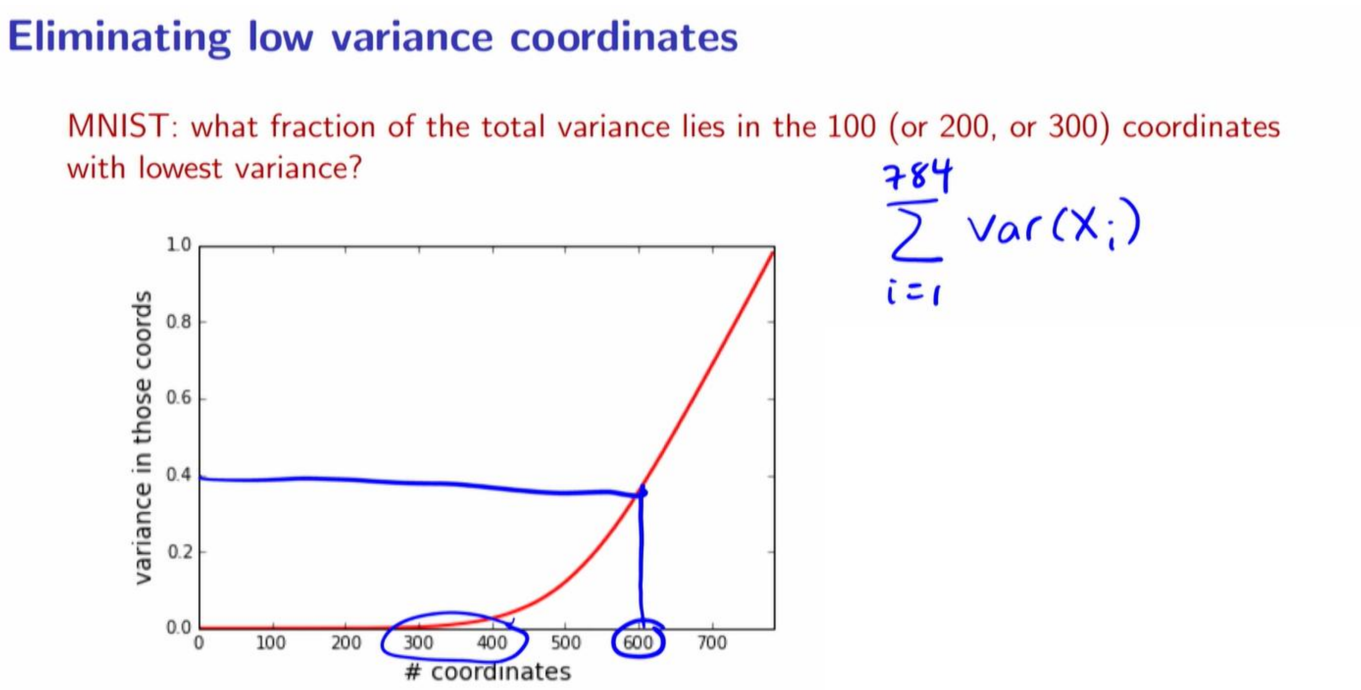
If we were to choose a few pixels to discard, which would be the prime candidates?

When you are given a very high-dimensional data set, a natural question to ask is, does the dimension really need to be this high? Or is there some way to reduce the dimension while retaining all of the information in the data, or at any rate, almost all of the information. If we could produce the dimension in this way it would afford some benefits.

- So first of all, lower dimension would mean quicker computation and perhaps more importantly a smaller memory need.
- Second, high-dimensional data often has a lot of redundancy in it. So for instance, in gene expression analysis there are often groups of genes that act together as a unit, that are co-expressed. Do we really need separate features for each of them? Or is it possible to combine them into a single group feature?
- And third, dimensionality reduction is a great opportunity to jettison irrelevant or noisy features. Okay, if you're looking over somebody's medical record to assess whether they're at risk for some disease. There is a lot of irrelevant information in there, social security numbers, for example. These are pure noise, and we should just get rid of them.

How would we do dimensionality reduction? Let's start with a simple example. So this is a picture of a digit from the Endless Data Set. And if you recall these images are all 28 by 28. So it's 28 pixels across, 28 pixels high. So it's a total of 784 pixels, which means that we can write them as a 784 dimensional vector. We first copy down the first row of pixels and the second row, and so on. So does the dimension really have to be 784 or can we reduce it somehow? Can we throw away some of these pixels? Well, we can definitely get rid of the corner pixels. So this corner here and that one and that one. These pixels have no information, they're always off basically. So we have now reduced the dimension from 784 to 780. It's a bit of a slow start. But maybe we can keep going in this way. Now before we get too excited we should just take a step back and think about what we just did. Why were we able to get rid of these pixels? Well, they always have the same value. Which means that they don't vary at all. In other words they have zero variance. So that gives us an idea, if we're gonna throw away more coordinates, more pixels, why don't we choose the ones with the lowest variance, the least informative pixels?

总结：

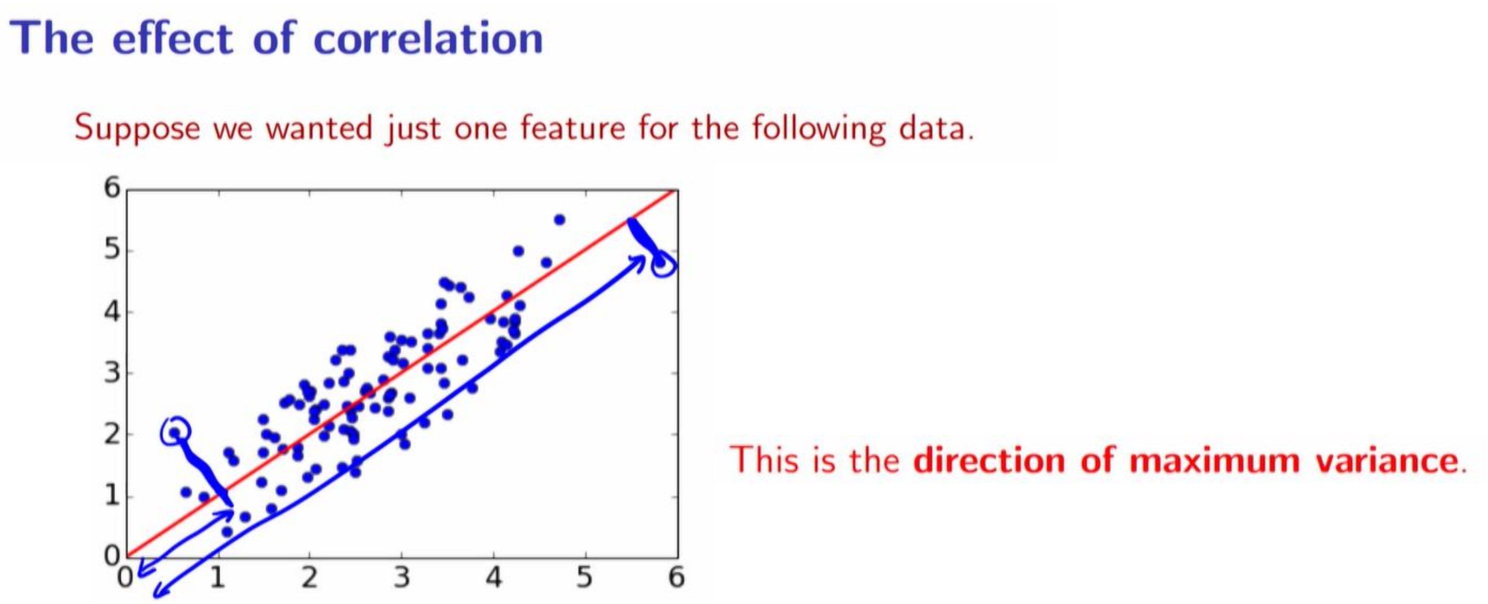


Eliminating low variance coordinates

MNIST: what fraction of the total variance lies in the 100 (or 200, or 300) coordinates with lowest variance?

$$\sum_{i=1}^{784} Var(X_i)$$

And this graph shows you what happens when you do that. So let me explain. Now if we look at all 784 coordinates, if we look at the full representation. The total variance in it, is simply the variance of the first pixel plus the variance of the second pixel, and so on, so the total variance is the variance of pixel number I where we sum I over all of the coordinates. It's the total variance. And what this graph shows, is how much of that variance you lose when you throw away some of the lowest variance coordinates. Okay, so let's say we throw away 600 of the lowest variance coordinates. So we choose the 600 coordinates with the lowest variance, we throw them away. If we do that we lose about 40 percent of the overall variance. Okay? So one of the interesting things here is that we can throw away three to 400 pixels and lose essentially no variance in the process. So this is great. We can reduce the dimension from about 800 down to about 400. We can half the dimension quite easily in this way. Now it'll turn out that we can actually do much much better than this. We can in fact reduce the dimension, not down to 400, but to something much more like 50 or maybe even less. But in order to do that we need a slightly more general type of projection.



The effect of correlation

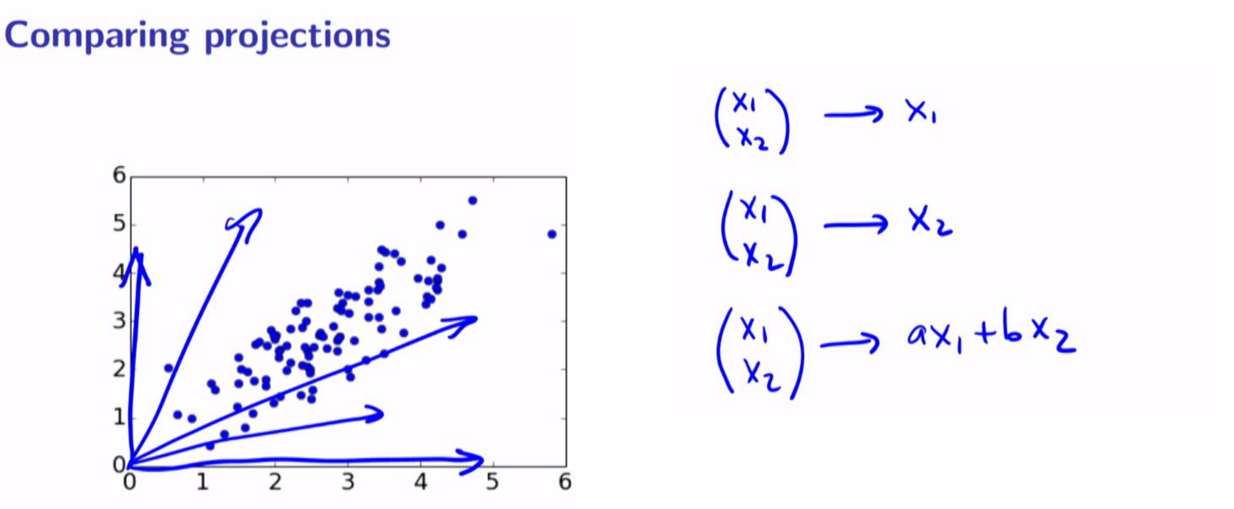
Suppose we wanted just one feature for the following data.

This is the **direction of maximum variance**.

So in order to explain that let's look at this 2D data set over here. So it has two features, X1 and X2. And there's quite a bit of variance along each of these two directions. But these are not directions that we would naturally be inclined to discard at all. But if you look at this data there is quite a lot of correlation between the two features. And so the question is, is there some way to exploit this correlation? And somehow reduce them to just one number instead of two numbers. And there is. The thing to do is to project onto this direction. The one shown in red. So, what does it mean to do this kind of projection? Well let's look at this point for example. What we will do is to project it onto the line. So we drop it onto the red line and we get this valley over here. We replace that 2D point by this distance. So we get just one number. Well let's say we take this point. We will project it onto the line. Projecting onto the line means finding the closest point on the line.

And what that means is that you have to hit the line at a right angle. So you hit the line at a right angle and you replace that two dimensional point by this length over here. And in this way the data set gets boiled out to one dimension. And since the points, by and large, lie fairly close to this line, not a lot of information has been lost in the process. So what exactly is this line, what is this red direction? It turns out that this is the direction of maximum variance.

总结：



Comparing projections

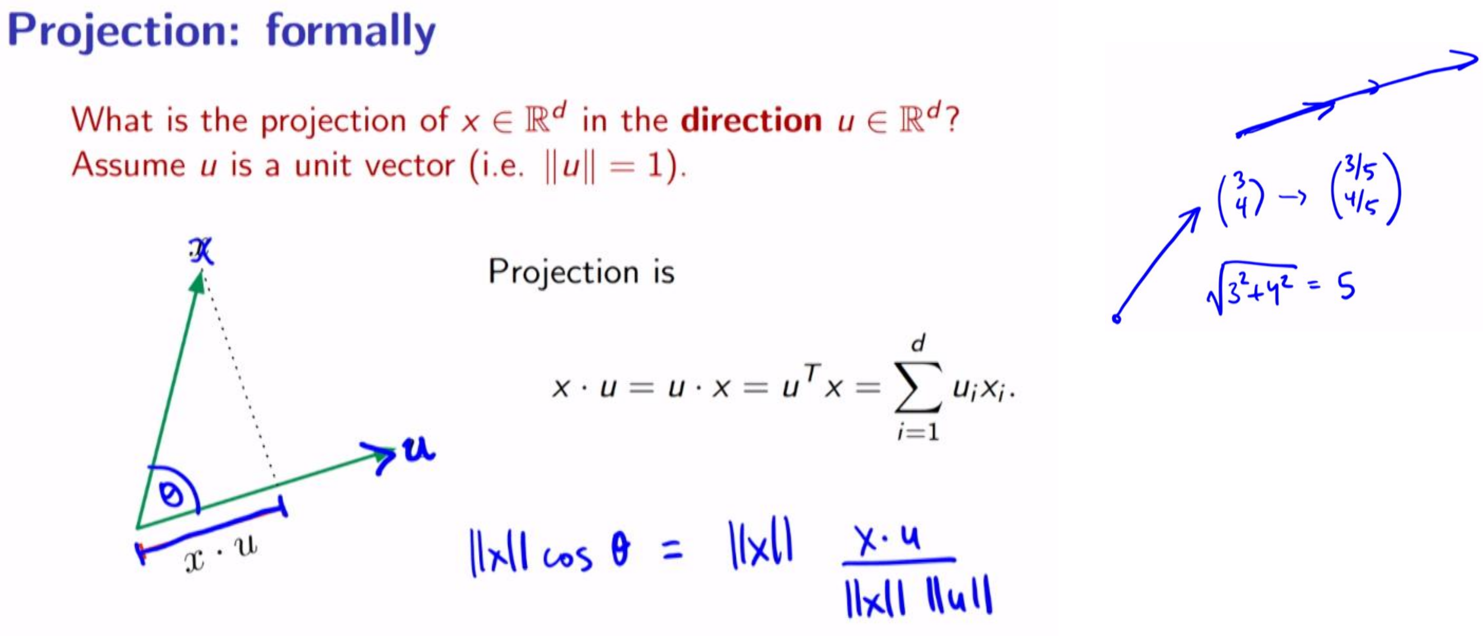
$\begin{pmatrix}x_1\\x_2\end{pmatrix} \longrightarrow x_1$

$\begin{pmatrix}x_1\\x_2\end{pmatrix} \longrightarrow x_2$

$\begin{pmatrix}x_1\\x_2\end{pmatrix} \longrightarrow ax_1 + bx_2$

What does that mean? Well, let's talk a little bit about projections more generally. So once we move away from just looking at coordinates there are infinitely many directions we can project onto. We can project onto this direction, we can project onto this direction, and so on, okay? So let's look at some of these. Suppose we project onto this direction. The X1 direction. So then a point X1 X2 simply gets mapped to it's first coordinate X1. What is the variance of the projected data? It's just the variance of X1. Let's say we project onto this direction, that coordinate axis. Then X1 X2 gets mapped just to X2. What is the variance of the projected data? It's just the variance of X2. But there's no reason why we need to limit ourselves to coordinate projections.

We can also project onto another direction, like this. And if we project onto an arbitrary direction of this kind, then it will turn out that X1 X2 gets mapped to some linear combination of X1 and X2. It gets mapped to a single number that's of this form. AX1 plus BX2 where A and B are sum constants. And again we get a one dimensional value and we can talk about the variance of these numbers. And what we want to find is the direction along which this projected data has the highest variance possible. Now, it will turn out that this is something we can solve quite easily. But before we get into that we need to formalize what exactly a projection is, mathematically.



Projection: formally

What is the projection of $x \in \mathbb{R}^d$ in the **direction** $u \in \mathbb{R}^d$?
Assume u is a unit vector (i.e. $\|u\| = 1$).

Projection is

$$x \cdot u = u \cdot x = u^T x = \sum_{i=1}^{d} u_i x_i .$$

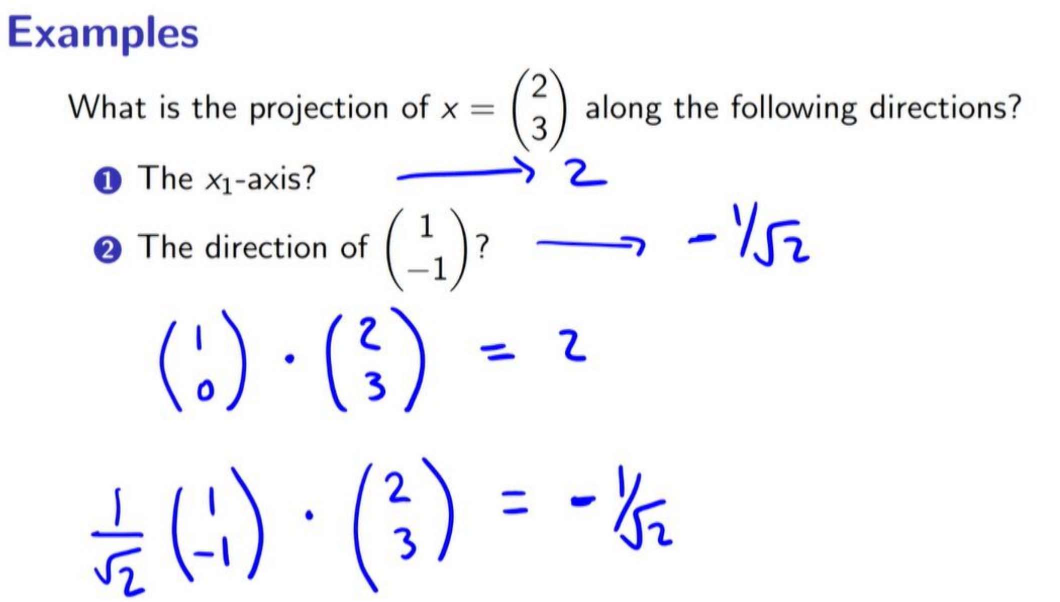
$$\|x\| \cos\theta = \|x\| \frac{x \cdot u}{\|x\| \, \|u\|}$$

$\begin{pmatrix}3\\4\end{pmatrix} \to \begin{pmatrix}3/5\\4/5\end{pmatrix}$

$\sqrt{3^2+4^2} = 5$

We want to project onto some direction. Maybe a direction like this. The length of that particular vector doesn't matter so projecting onto this direction is the same as projecting onto this direction and it's the same as projecting onto that direction, and so on. So for convenience, what we often do is to normalize to unit length. So when we're projecting onto a certain direction, we represent that direction by a unit vector, a vector of length one. For example, let's say we want to project onto the direction three four. So it's a direction something like this. Is that a unit vector? Does it have length one? Well let's see, what is the length of that? It is three squared plus four squared, which is 25, so it's five. The length is five, it's not a unit vector. So to make it into a unit vector we just divide by the length. Instead we use three fifths and four fifths and that's the direction that we're going to project on to.
So, how does one do projections? Now we have some direction U that we are projecting onto. What is the projection of a point X onto that direction? It is simply the dot product between X and U. It's X.U which is the same as U.X which is the same as U transpose X which is the same as X transpose U. That's all it is. This length over here, which is the projection is just X.U. Why is that? Well it's actually very simple to explain. So let's look at this angle, let's call it θ. So what is the projection? The projection is, let's see if you remember trigonometry, the projection is the length of X times the cosign of that angle. And when we were talking about dot products, when we first talked about dot products, we saw that they actually give you the angle between two vectors. We saw a formula that said the cosign of peda is actually X.U divided by the length of X and divided by the length of U. Now U is a unit vector so it's length is one. And so do you see how it all works out? So, projections are just dot products.

总结：



Examples

What is the projection of $x = \begin{pmatrix}2\\3\end{pmatrix}$ along the following directions?

1. The x_1-axis? $\longrightarrow 2$
2. The direction of $\begin{pmatrix}1\\-1\end{pmatrix}$? $\longrightarrow -1/\sqrt{2}$

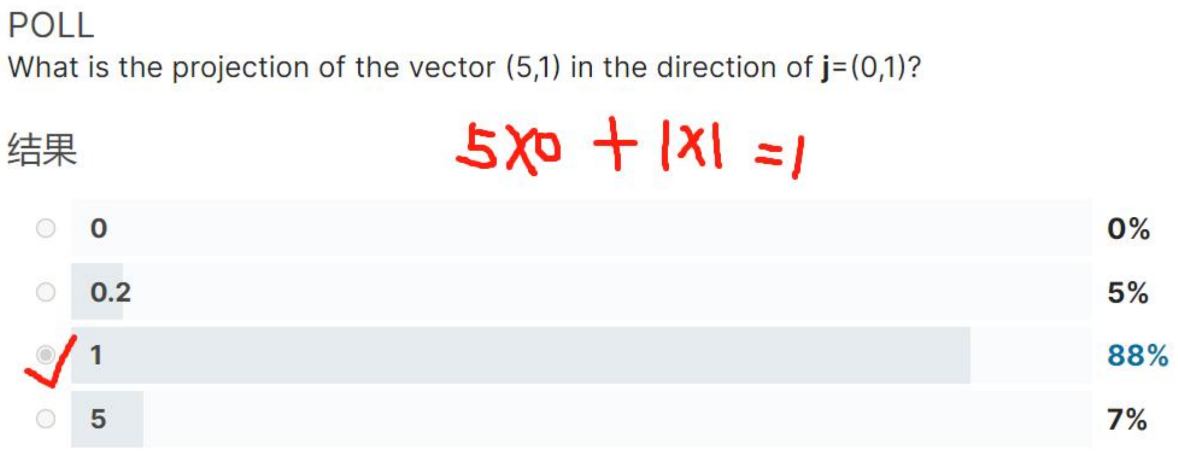
$$\begin{pmatrix}1\\0\end{pmatrix}\cdot\begin{pmatrix}2\\3\end{pmatrix} = 2$$

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}\cdot\begin{pmatrix}2\\3\end{pmatrix} = -1/\sqrt{2}$$

Let's see a couple of examples of this. So here we have a vector X and we want to project it onto two different directions. So let's see, in the first case we want to project it onto the X1 axis. What is that? That is one zero, it's the X1 direction. And to project it onto that direction we just do a dot product. Dot product with two three and we get two. The answer is two. Very simple and what we would have expected.

Now let's do the second one. One negative one, we want to project onto that direction, well wait, we need to make sure that it's a unit vector. What is it's length? Well it's one, plus one square, so it's the square root of two. In order to make it a unit vector we have to divide by it's length. So we divide by the square root of two. Now we can take this and do the dot product. We have a two three, and what do we get? We get two minus three divided by square root two, so minus one over square root two. The answer to this is minus one over square root two. And that's it.

So that's it for today. We formally introduced the notion of projection and found that all it is is dot product. Next time we'll build upon this and we'll find the direction of maximum variance. See you then.



POLL

What is the projection of the vector (5,1) in the direction of **j**=(0,1)?

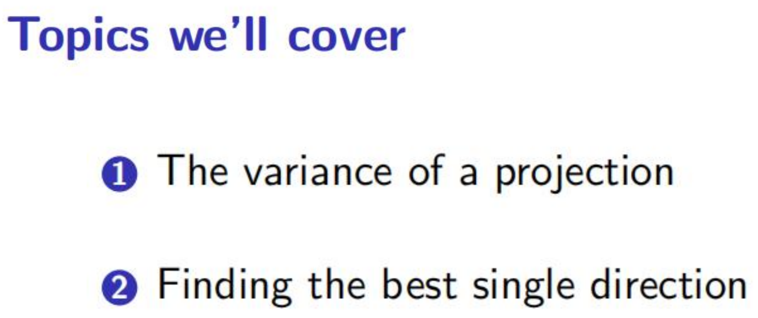
结果

$5\times 0 + 1\times 1 = 1$

○	0	0%
○	0.2	5%
● ✓	1	88%
○	5	7%

总结：

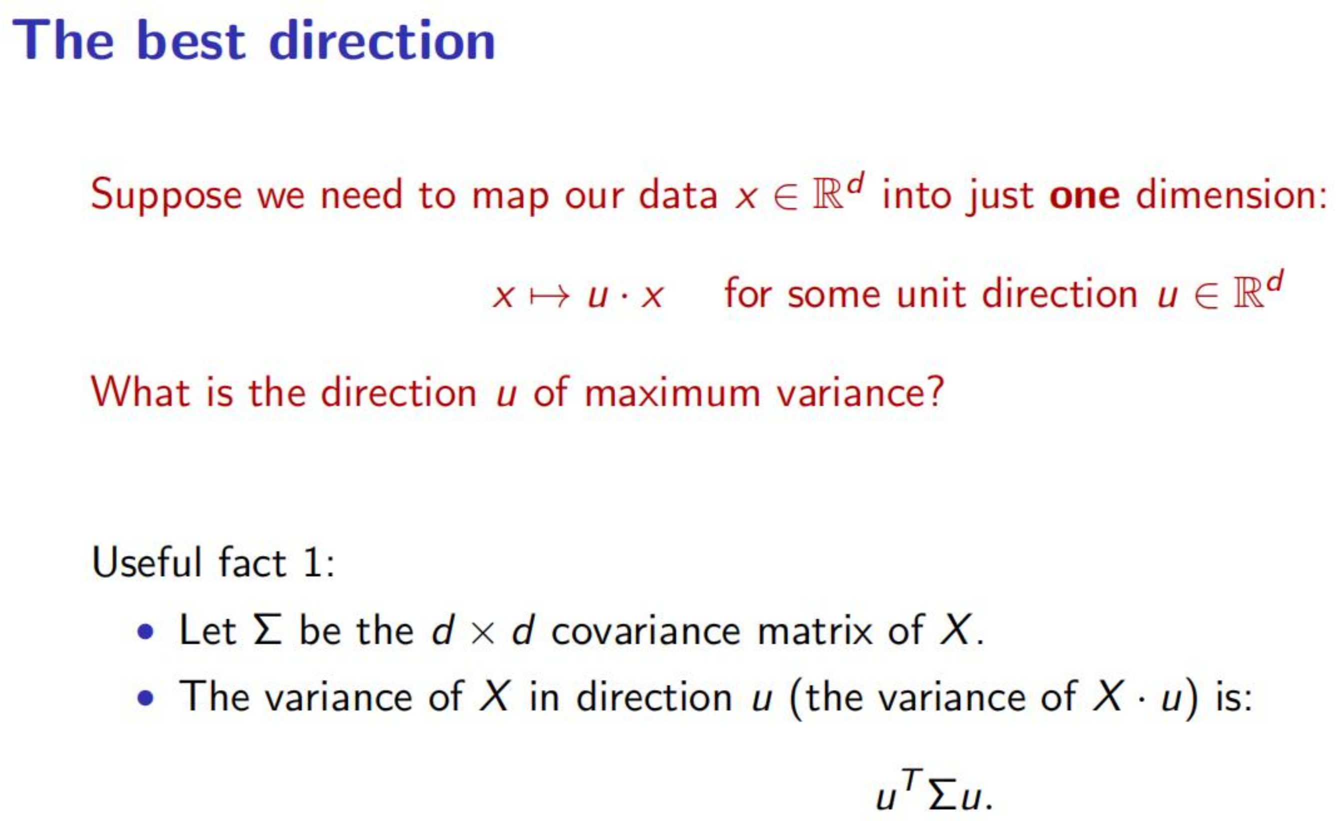
9.2 Principal Component Analysis I: One Dimensional Projection



Topics we'll cover

1. The variance of a projection
2. Finding the best single direction

Last time we talked about what a projection means. What we'll do today is to find the direction of highest variance, The best projection onto a single direction.
So we'll start by looking at the variance of a projection in any direction, and then we'll see how to find the best direction.



The best direction

Suppose we need to map our data $x \in \mathbb{R}^d$ into just **one** dimension:

$$x \mapsto u \cdot x \qquad \text{for some unit direction } u \in \mathbb{R}^d$$

What is the direction u of maximum variance?

Useful fact 1:

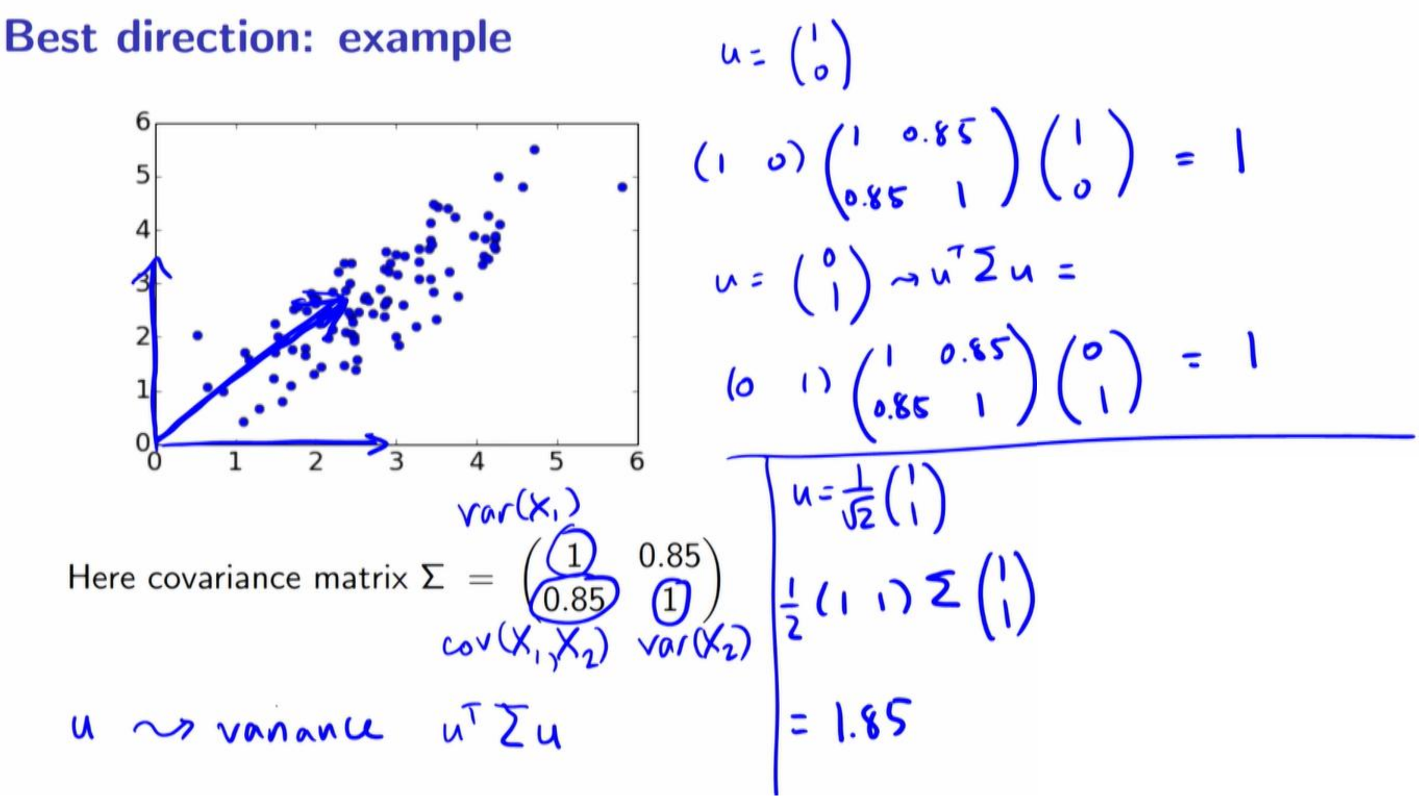
- Let Σ be the $d \times d$ covariance matrix of X.
- The variance of X in direction u (the variance of $X \cdot u$) is:

$$u^T \Sigma u.$$

So suppose we have a data set in d dimensions, and we want to map it down to just one dimension. We wanna project it down into one dimension. So we find a unit vector u and the projection then is just the dot product, u dot x. We wanna find the direction u for which this projected data has as much variance as possible. Is in a sense as informative as possible. What is the direction of maximum variance? How would we find that? Well I suppose we could try many different directions. We could say okay let's try to project onto this direction, we project everything on there and we compute the variance of the projected points. Then we try another direction, we project the points on there, compute their variance. But there are infinitely many directions to choose from. So this would be really a hit and miss process.

It turns out that there is a very simple formula that gives us the variance in any direction. So here it is. Okay, so we have this d dimensional data x, let's look at it's covariance matrix, sigma. Then the variance in any direction u, is simply u transposed sigma u. That's it. Okay simple. But it's also rather abstract, so let's look at a slightly more concrete setting.

总结：



Best direction: example

$u = \begin{pmatrix}1\\0\end{pmatrix}$

$(1\ \ 0)\begin{pmatrix}1 & 0.85\\0.85 & 1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = 1$

$u = \begin{pmatrix}0\\1\end{pmatrix} \rightsquigarrow u^T\Sigma u =$

$(0\ \ 1)\begin{pmatrix}1 & 0.85\\0.85 & 1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix} = 1$

$u = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$

$\frac{1}{2}(1\ \ 1)\Sigma\begin{pmatrix}1\\1\end{pmatrix}$

$= 1.85$

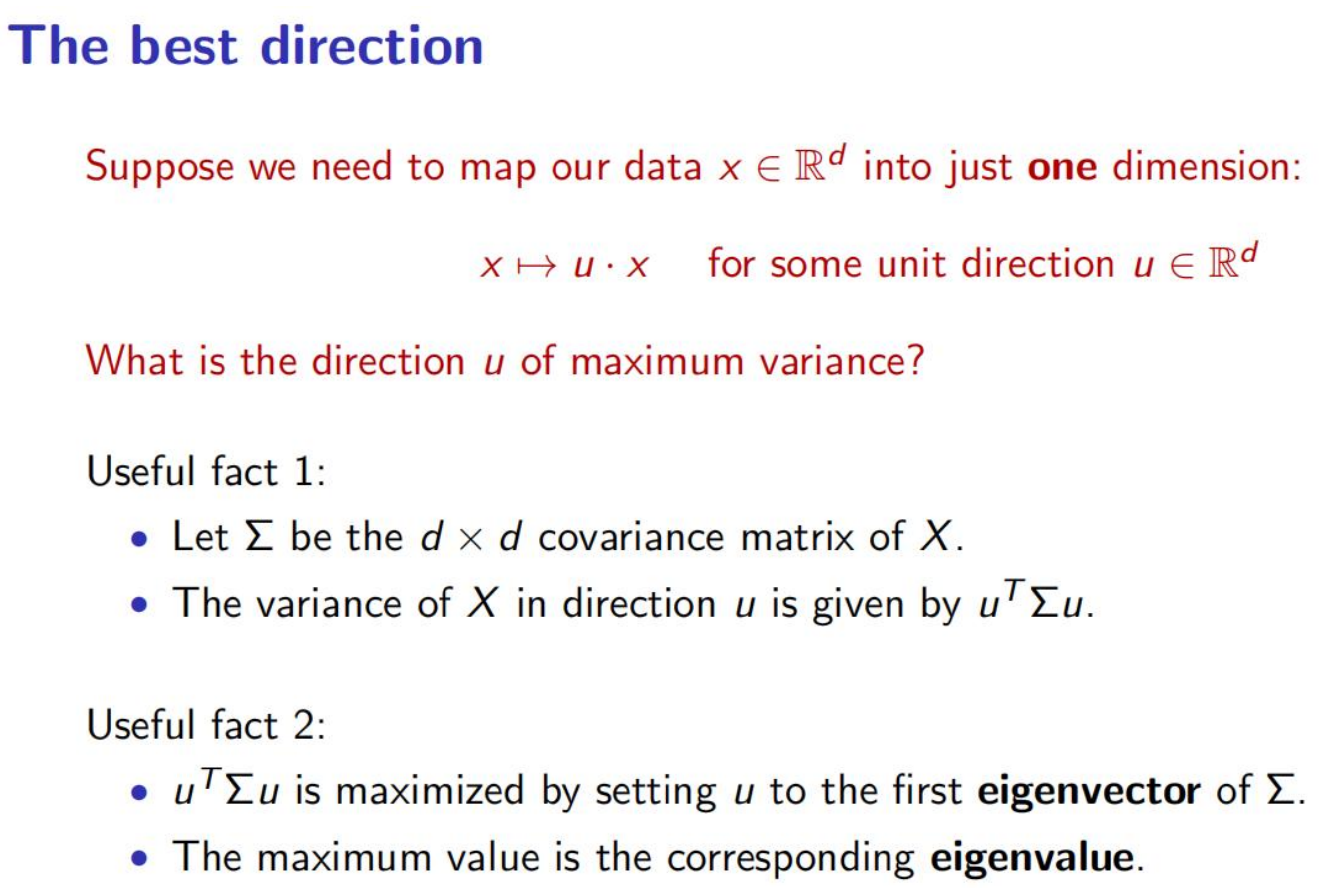
Here covariance matrix $\Sigma = \begin{pmatrix} \overset{\text{var}(X_1)}{\boxed{1}} & 0.85 \\ \underset{\text{cov}(X_1,X_2)}{\boxed{0.85}} & \underset{\text{var}(X_2)}{\boxed{1}} \end{pmatrix}$

$u \rightsquigarrow$ variance $u^T\Sigma u$

So here's a familiar data set, this a two dimensional data set. And so it's covariance matrix is gonna be a two by two matrix. So just to remind you in the covariance matrix, this number over here is the variance of the first feature. So that's the variance of x one. This is the variance of the second feature, so the variance of x two. The off-diagonal numbers are gonna be the same since it's a symmetric matrix. And what the off-diagonal number represents is the covariance of x one and x two. And it's a positive number representing the fact that there's a positive correlation between these two features, the data is sloped upwards. So, what did our formula say? So we said that if we wanna project onto a direction u, then the variance in that direction, the variance of x dot u, is simply u transposed sigma u. Let's try this out okay.

- So for example let's say we wanna project onto the x one axis. In that case the vector we're projecting onto, the projection direction is one, zero. That's the x one direction. And what is x transposed sigma u? Well x transposed is one, zero, sigma is one, 0.85, 0.85, one and one, zero. This works out to one. Is that the variance of the projected data? Yes it's the variance of x one, we already knew that.
- Let's try another direction. Let's say we wanna project onto the x two direction. So zero, one. This time u transposed sigma u is zero, one, one, 0.85, 0.85, one, zero, one. And this is again one. So the projection in that direction is also one which we also knew. Okay so far we haven't learned anything new.
- Let's try another direction, why don't we try this direction over here. The direction one, one. So now u is 1, 1. Oh wait so it has to be a unit vector. So what is the length of this thing? 1, 1, the length is square root two. So in order to make it a unit vector we have to divide it by square root two. So let's put a one over square root two in front of it. So this is the direction we wanna project onto. What is the variance in that direction? It's u transposed sigma u which is, let's pull the square root twos out front so we get 1/2, one, one, sigma, one, one. And what that is is just the sum of the entries of sigma divided by two which is 1.85. So the variance in this direction, if we project onto this direction the projected data has a variance of 1.85. Much higher than either of the coordinate projections. So this is clearly a much better direction to project onto. And in fact this is the direction of highest variance. How can we be sure of that? So let's go back to our slightly more abstract discussion and see how we know.

总结：



The best direction

Suppose we need to map our data $x \in \mathbb{R}^d$ into just **one** dimension:

$$x \mapsto u \cdot x \qquad \text{for some unit direction } u \in \mathbb{R}^d$$

What is the direction u of maximum variance?

Useful fact 1:

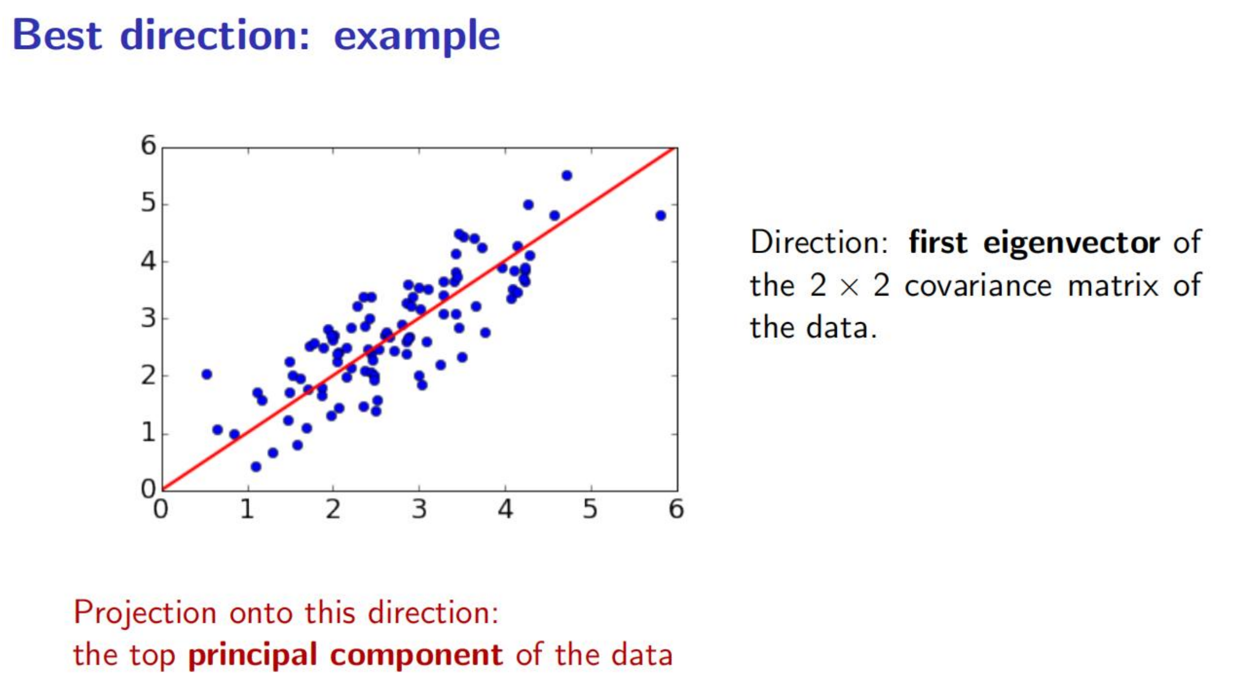
- Let Σ be the $d \times d$ covariance matrix of X.
- The variance of X in direction u is given by $u^T \Sigma u$.

Useful fact 2:

- $u^T \Sigma u$ is maximized by setting u to the first **eigenvector** of Σ.
- The maximum value is the corresponding **eigenvalue**.

So we have this data in d dimensional space, we know that the variance in any direction u is u transposed sigma u. What we wanna do is to find the direction that maximizes this quantity, u transposed sigma u. How do we do that? It turns out there's a very simple answer. That direction, the maximizing direction is simply the first eigenvector of sigma, that's it. Okay but what exactly is the eigenvector? Well this is a very important concept that gets used a lot in machine learning. And so we will be covering it quite carefully soon. But for the time being let me just say a few words about it. So sigma is not just any old matrix. So first of all it is a square matrix, it's d by d. It's symmetric. We saw a little earlier that it's also positive semidefinite. So it has a lot of nice properties. And when a matrix has these kind of nice properties, it has associated with it a set of d special directions, called eigenvectors, along with a set of numbers called eigenvalues. Positive numbers that tell you the stretch of the matrix in each of those directions. So this all sounds rather mysterious right now, and it will hopefully make sense in a week or so.

But, for the time being the main thing to bear in mind is that these eigenvectors, these special directions, are easy to obtain. Python will do it for you for example. You just feed in the covariance matrix and it gives you back these directions. And then you just have to pick the top eigenvector and that is the direction of maximum variance.



Best direction: example

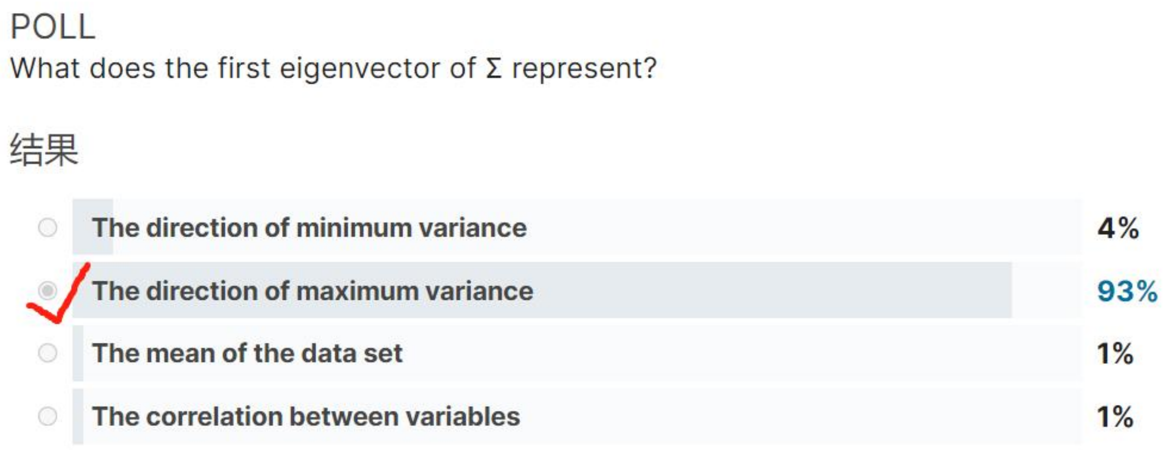
Direction: **first eigenvector** of the 2×2 covariance matrix of the data.

Projection onto this direction: the top **principal component** of the data

总结：

So going back to our example, the direction of maximum variance, the top eigenvector of the two by two covariance matrix in this case, is exactly the direction shown here in red. And when we project the data onto this direction it's often called the top principal component of the data. This is the simplest case of principal component analysis.

So that's it for now. Today we've seen how to find the best single direction on which to project data. Next time we'll generalize this and see what if we wanna project in k directions, what should we set them to? So see you then.



POLL

What does the first eigenvector of Σ represent?

结果

○ The direction of minimum variance 4%

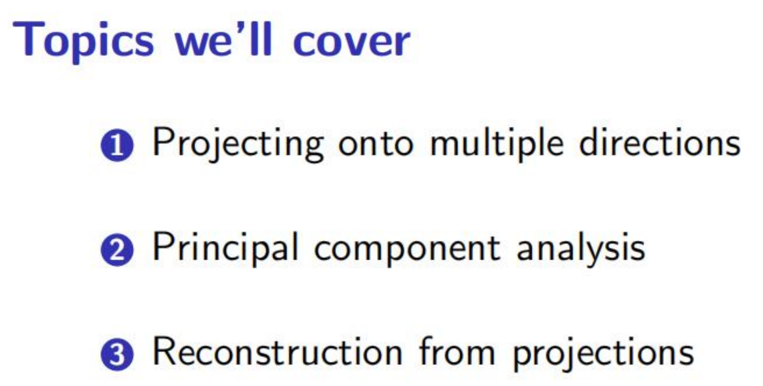
◉ ✓ The direction of maximum variance 93%

○ The mean of the data set 1%

○ The correlation between variables 1%

总结：

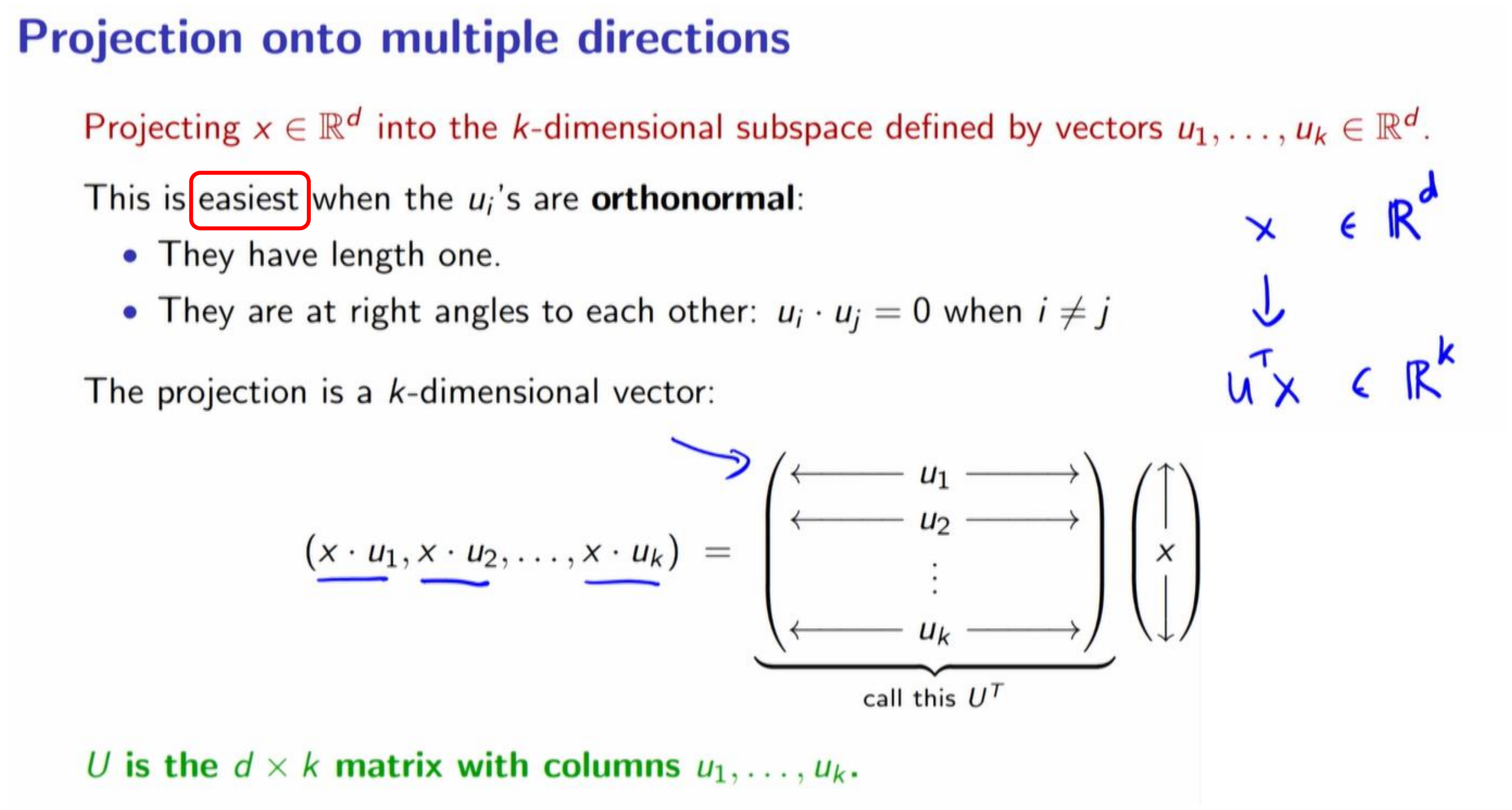
9.3 Principal Component Analysis II: The Top k Directions



Topics we'll cover

1. Projecting onto multiple directions
2. Principal component analysis
3. Reconstruction from projections

Last time, we talked about the best single direction on which to project a data set and the answer was the top eigenvector of the covariance matrix of the data. Today, we'll look at projections onto several directions. We'll start by defining what it means to project onto multiple directions and then, we'll look at the optimal solution, which is called principal component analysis. Finally, we'll look at the reconstruction question, when you're given just the projection of data, how do you reconstruct in the original space?



Projection onto multiple directions

Projecting $x \in \mathbb{R}^d$ into the k-dimensional subspace defined by vectors $u_1, \ldots, u_k \in \mathbb{R}^d$.

This is $\boxed{\text{easiest}}$ when the u_i's are **orthonormal**:

- They have length one.
- They are at right angles to each other: $u_i \cdot u_j = 0$ when $i \neq j$

The projection is a k-dimensional vector:

$$x \in \mathbb{R}^d \quad \downarrow \quad u^T x \in \mathbb{R}^k$$

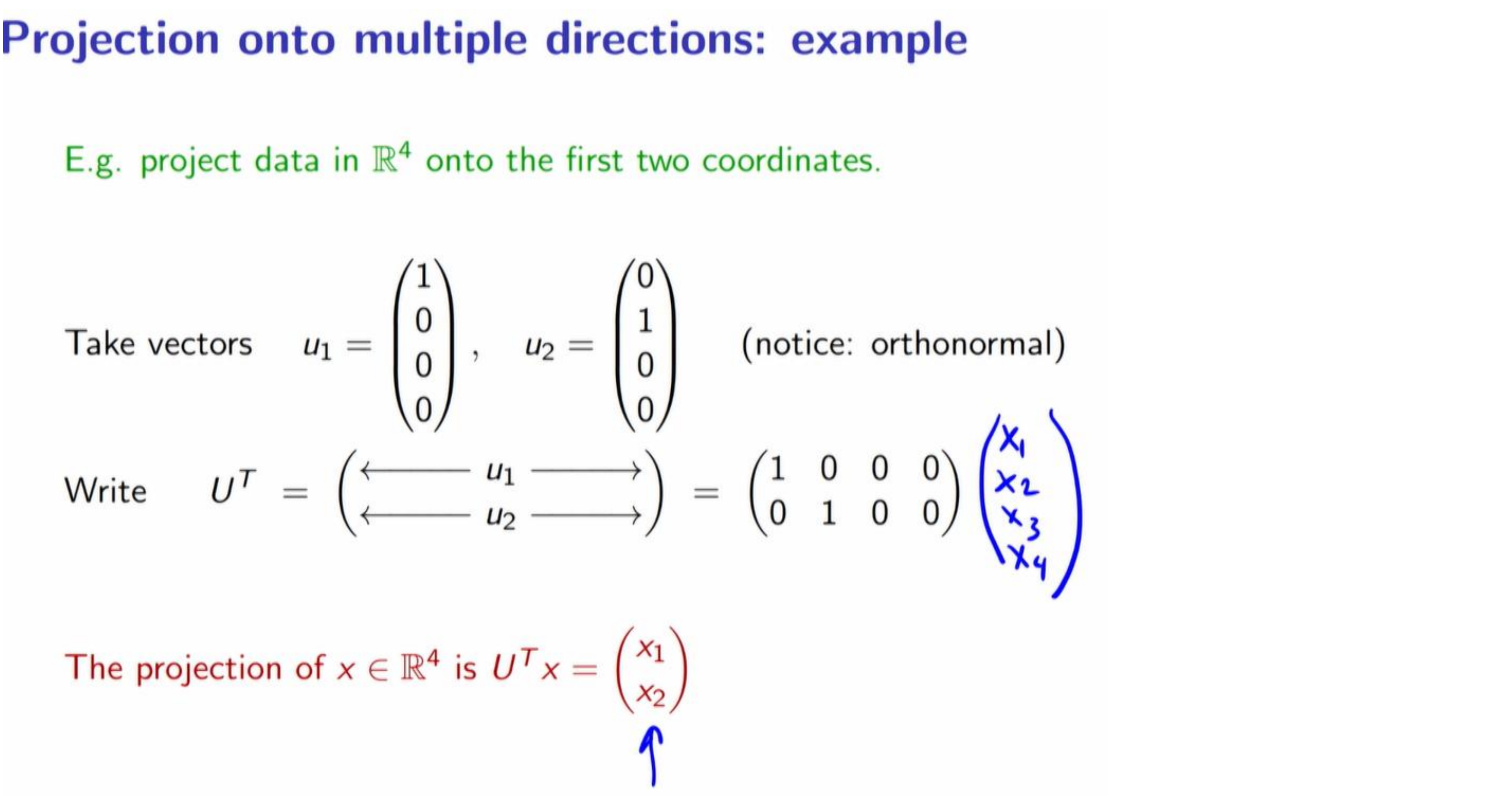
$$(\underline{x \cdot u_1}, \underline{x \cdot u_2}, \ldots, \underline{x \cdot u_k}) = \underbrace{\begin{pmatrix} \longleftarrow & u_1 & \longrightarrow \\ \longleftarrow & u_2 & \longrightarrow \\ & \vdots & \\ \longleftarrow & u_k & \longrightarrow \end{pmatrix}}_{\text{call this } U^T} \begin{pmatrix} \uparrow \\ x \\ \downarrow \end{pmatrix}$$

U **is the** $d \times k$ **matrix with columns** $u_1, \ldots, u_k$.

We're in a situation, where we have a data set that's d dimensional and we wish to reduce the dimension. What we discussed last time was reducing down to one dimension by projecting onto a single direction and we looked at an optimal solution for doing this according to a natural measure of loss. Now, generally one dimension is not going to be enough. No matter how carefully you choose that direction, it's unlikely that just one number is gonna capture everything you need from the original high dimensional data, so we usually need to project onto multiple directions. We need to create a representation that's k dimensional, where k is something more than one and so let's see what that means exactly.

Suppose we choose k different directions now, u1 through uk, what does it mean to project onto those directions? This question is easiest to answer if the directions satisfy certain properties, in particular if they are orthonormal. This means two things. First of all, the directions u1 through uk should all have length 1 that's easy to manage and second, they should be at right angles to each other. They should be orthogonal to each other, their dot product should be zero and that's also something that's easy to generically manage. If these conditions hold, then the projection is very simple. The projection of a point x onto these k directions just consists of k numbers now. The first number is the projection onto u1, x.u1. The second number is the projection onto u2, x.u2, all the way to the last number, the projection onto uk, x.uk, very simple. We can also write this in matrix form, so suppose we take these k different directions u1, u2, u3, uk and stick them together into a matrix, we make them columns of a matrix, then let's call that matrix u. U transpose looks like this thing over here. U transpose has these directions along the rows and then, the projection of x is simply u transpose times x. It's a simple matrix vector product. The projection of the vector x is simply u transpose times x. We have this vector that's originally in d dimensional space and it gets projected down to something that's in k dimensions.

总结：



Projection onto multiple directions: example

E.g. project data in $\mathbb{R}^4$ onto the first two coordinates.

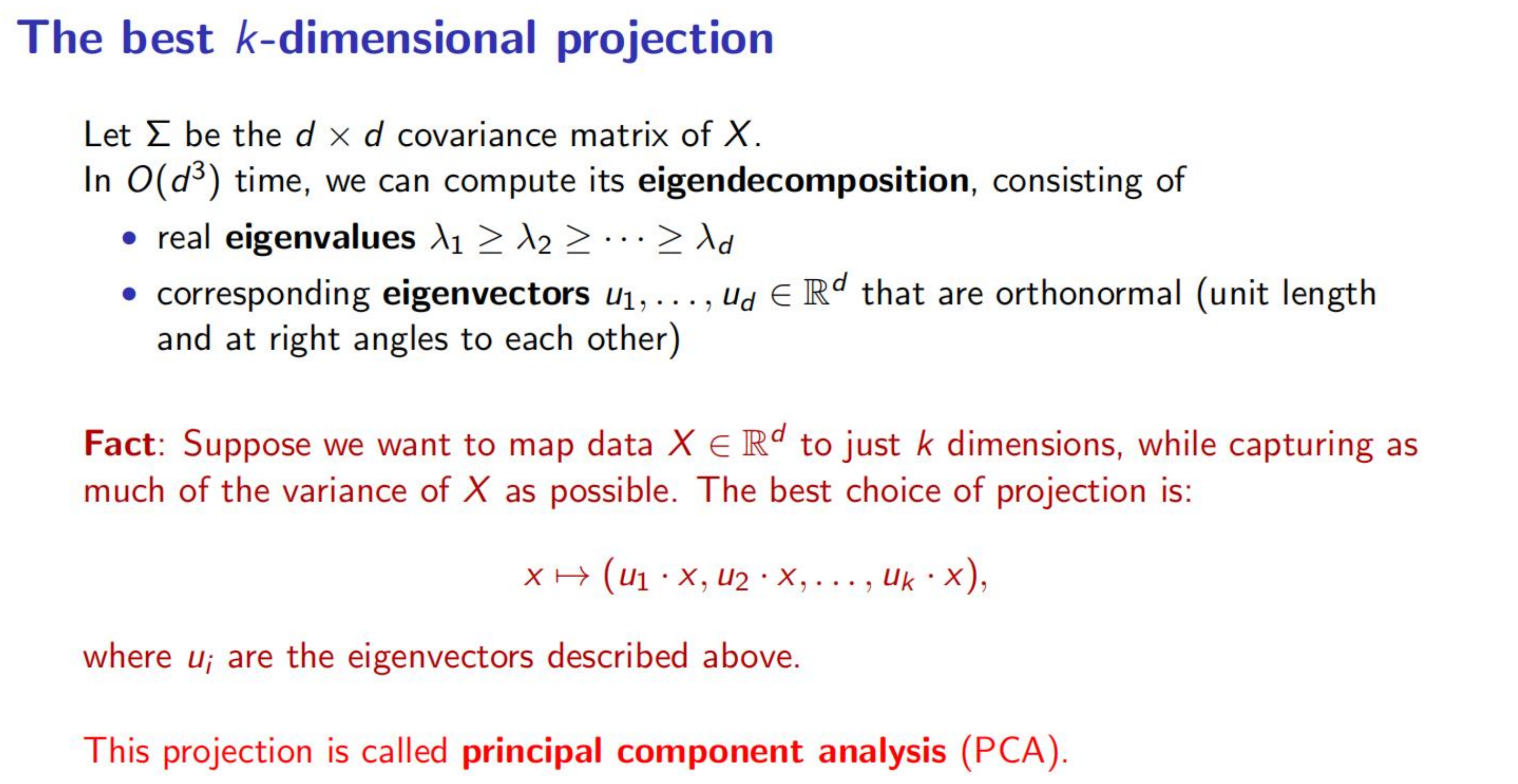
Take vectors $u_1 = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}$, $u_2 = \begin{pmatrix}0\\1\\0\\0\end{pmatrix}$ (notice: orthonormal)

Write $U^T = \begin{pmatrix} \longleftarrow & u_1 & \longrightarrow \\ \longleftarrow & u_2 & \longrightarrow \end{pmatrix} = \begin{pmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\\x_4\end{pmatrix}$

The projection of $x \in \mathbb{R}^4$ is $U^T x = \begin{pmatrix}x_1\\x_2\end{pmatrix}$

How do we find the optimal such projection? Let's start by looking at a small example, just to kind of get the notation straight. Let's say we have data in four dimensional space and we want to project it down to two dimensions and let's do something really simple. Let's just project onto the first two coordinates. Now, this is trivial, we have a point x1, x2, x3, x4 and we just want to spit out x1, x2, okay so we know exactly how to do this. There's nothing to it, but let's just go through anyway and see what the notation looks like in this case. That will help us as we gear up towards the optimal projection.

In this case, we are projecting onto two directions. The first direction is just the x1 direction, so it looks like this 1 0 0 0 and the second direction is the x2 direction, so it's 0 1 0 0. These are the two directions we're projecting onto u1 and u2 and we want to check that these are orthonormal, so are they unit length? Yes, they're both unit vectors. Are they orthogonal to each other? Let's see, their dot product is indeed zero. These vectors are orthonormal, so we are good to go. What we do is we just put these vectors together into a matrix U and then, U^T looks like this. It's this matrix over here and that's going to be our projection matrix. To project the point, we just multiply it by U^T. When we take this matrix and we multiply it by any given point x1, x2, x3, x4, what do we get back? Just the first two coordinates x1 and x2 as expected. This is what the various quantities refer to. We have these individual directions u1, u2 that are unit length and orthogonal to each other and then we put them together into a matrix.



The best k-dimensional projection

Let Σ be the $d \times d$ covariance matrix of X.
In $O(d^3)$ time, we can compute its **eigendecomposition**, consisting of

- real **eigenvalues** $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$
- corresponding **eigenvectors** $u_1, \ldots, u_d \in \mathbb{R}^d$ that are orthonormal (unit length and at right angles to each other)

Fact: Suppose we want to map data $X \in \mathbb{R}^d$ to just k dimensions, while capturing as much of the variance of X as possible. The best choice of projection is:

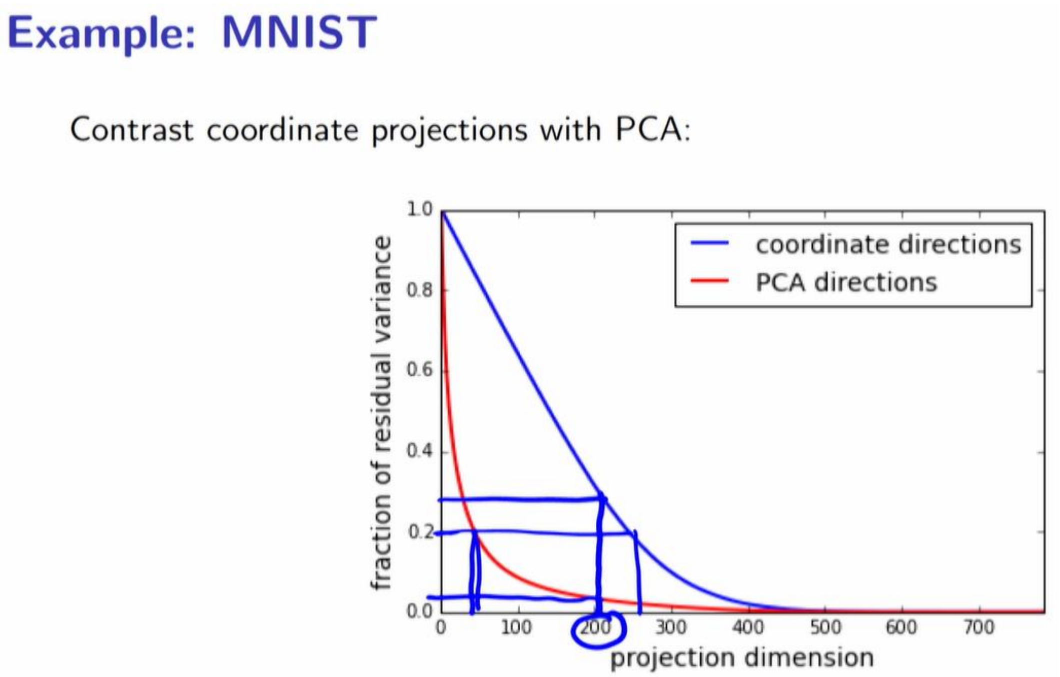
$$x \mapsto (u_1 \cdot x, u_2 \cdot x, \ldots, u_k \cdot x),$$

where u_i are the eigenvectors described above.

This projection is called **principal component analysis** (PCA).

总结：

Let's move on to the optimal projection now. As before, this is given by the eigenvectors of the covariance matrix, so we have a D dimensional data set that means it's covariance is a D by D matrix. This matrix has an associated set of eigenvectors. These are special directions associated with the matrix. Now, this is something we're going to discuss in a little bit more detail soon, but for the time being, these eigenvectors are something that are fairly easy to compute. We can ask Python to do it for us. We just give it the covariance matrix and it gives us back these eigenvectors. Once we have these eigenvectors, these D directions, what do we project onto? The answer is simply to project onto the top k eigenvectors. It's very simple. It's a very simple way to proceed and it's called principle component analysis. We compute the covariance matrix. We ask Python to give us the eigenvectors. We take the top k eigenvectors, call them u1 through uk and our projection is simply a dot product with these.

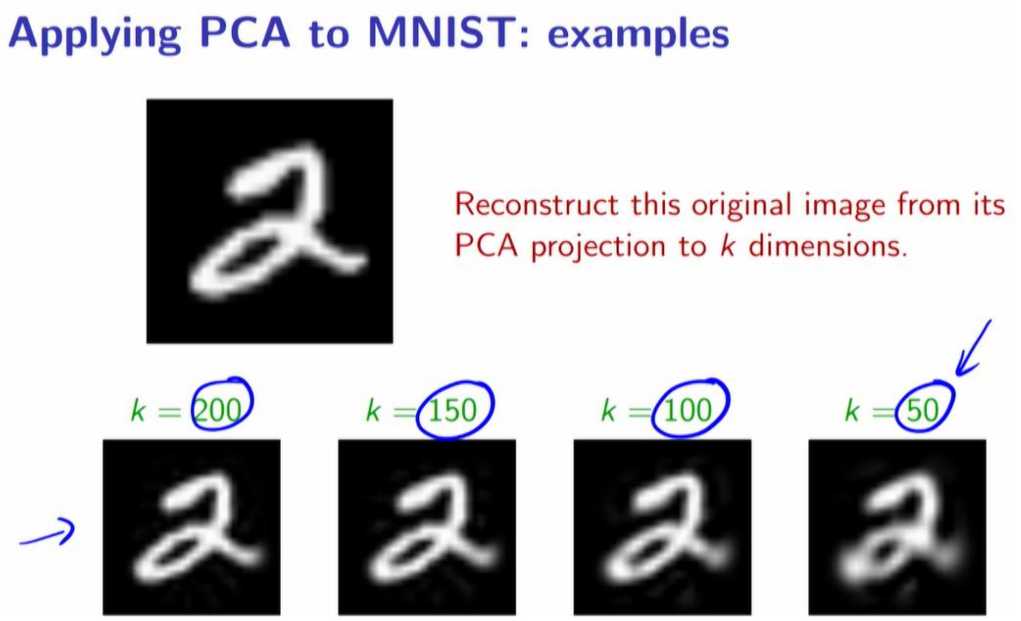


Example: MNIST

Contrast coordinate projections with PCA:

Let's go ahead and see this at work. This is the MNIST dataset and if you recall, this is a dataset that's 784 dimensional. If we want to reduce the dimension, we can reduce it to anything from zero to 784 and that's what's shown along the x-axis over here. Now, the two lines, the blue line shows what happens if you choose coordinate projections that is if you just select a few coordinates, you select pixels, you select the k pixels with the highest variance. The red line shows what happens if you use principle component analysis, so let's interpret this a little bit.

- Suppose we want to project to 200 dimensions, let's see what happens on the blue line, so we go up here. If we project to 200 dimensions by simply picking the 200 pixels that have highest variance, then we end up losing about 30% of the overall variance of the data, but if we project to 200 dimensions by doing principal component analysis, then we lose less than 5% of the original variance of the data, so it's a huge difference.
- Another way to think about it is suppose that we are willing to give up on 20% of the variance in the original data set, okay so that corresponds to this value on the y-axis. Now, we just draw a line across. If we're going to do coordinate projection, so if we're just going to pick pixels, then it turns out we're gonna have to pick about 250 pixels, so we'll be able to reduce the dimension to 250. If we're using principal component analysis, however, which picks arbitrary directions, then we can reduce the dimension to something below 50, so again a huge difference.

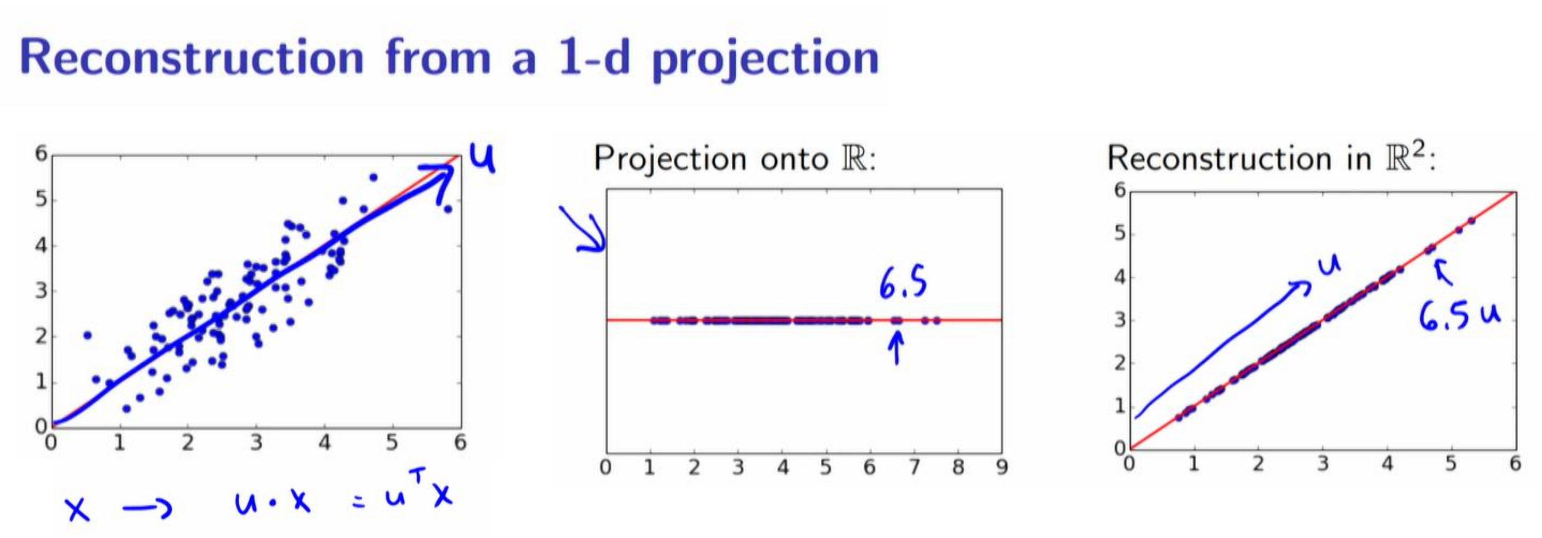


Applying PCA to MNIST: examples

Reconstruct this original image from its PCA projection to k dimensions.

总结：

Now, these numbers can be a little bit hard to interpre tlike what does it mean to lose 20% of the variance in the data, is that good or bad, is that too little or too much? Let's look at something a little bit more concrete. Let's look at some pictures. What's happened over here is that we've taken the MNIST dataset. We computed its covariance matrix for the entire dataset. We recovered the top u eigenvectors and we're going to project onto the top k directions, where k is here either 200 or 150 or 100 or 50. Now, in these pictures, I picked a particular digit, the one you see on the top and I've shown what its projection looks like for different values of k. If we project to 200 dimensions, you get this one over here. It looks just like the original, the one at a 150 dimensions also looks just like the original. Once you get to a 100 dimensions, there's a slight blurring that's setting in and when you go down to 50 dimensions, it's slightly more blurred, but it's still obvious what digit it is. Even projecting down to 50 dimensions is retaining more than enough information, at least for a human to be able to classify this digit. Using this method, we can really reduce the dimension pretty significantly. Now, what exactly are these pictures though? If we look at this last one for example, we have something that's in 784 dimensions and we're projecting down to 50, so what is this thing that I'm displaying over here? What is this reconstruction? What is this picture? After all, I just have 50 numbers.



Reconstruction from a 1-d projection

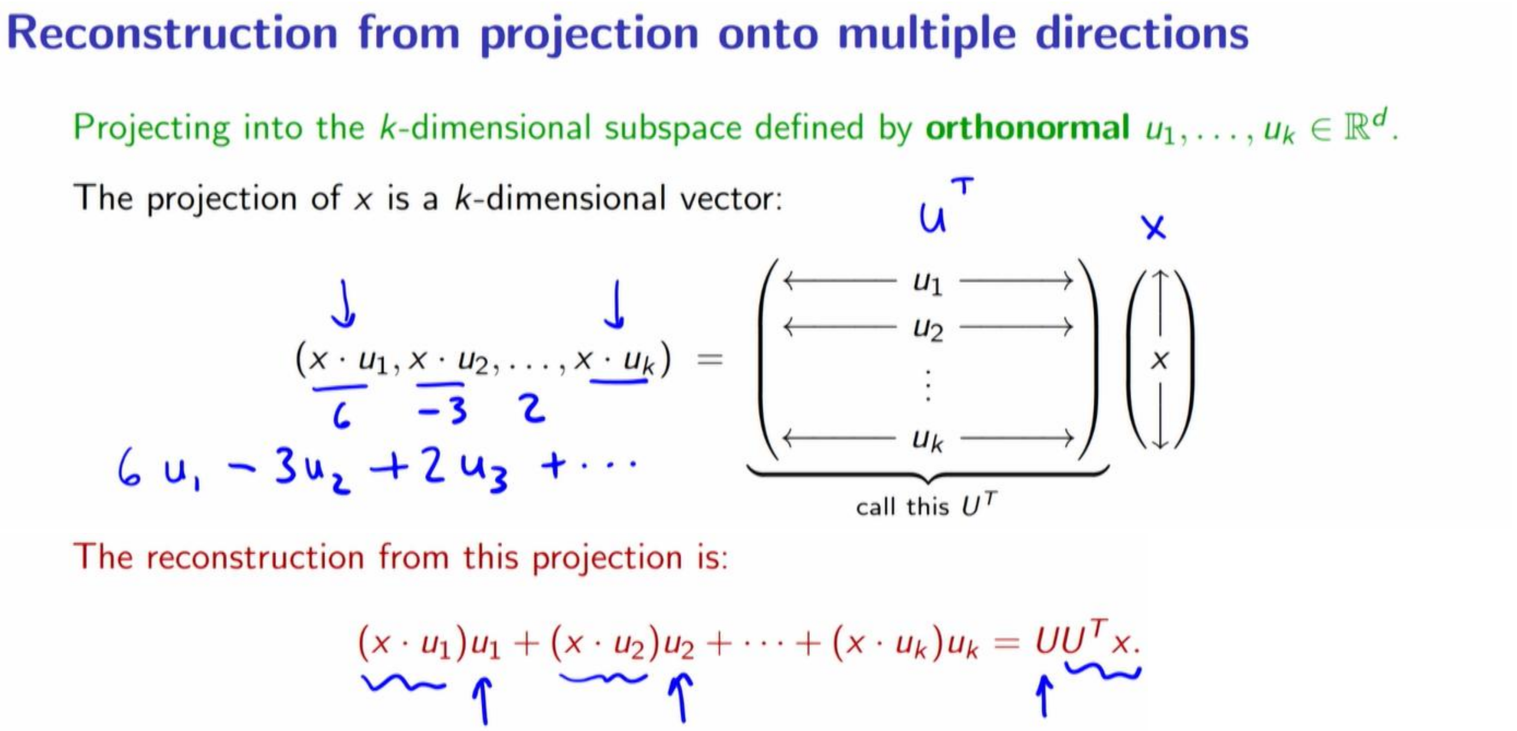
Projection onto $\mathbb{R}$:

Reconstruction in $\mathbb{R}^2$:

$x \longrightarrow u \cdot x = u^T x$

Let's see how to do that. How do you take a 50 dimensional projection and reconstruct something in the original 784 dimensional space? Let's start with a simpler example, just from going down from two dimensions to one dimension.

Here we have a two dimensional data set. We go ahead and we pick a direction to project onto. Let's say that's direction u. We're projecting every point x on to u.x or u transpose x. That's how we project the points and over here, you see the projections. The projections are just single numbers. The points are all laid out on the line. Now, we want to reconstruct. How do we do that? What reconstruction do we use for this point for example? That point has got a value of 6.5. That's just a single number. How do we reconstruct it in 2D? Well, this is what we do, we just say, "Move "6.5 units in the direction u." We find this point over here, which is 6.5 times u. U is a 2D vector and so this gives us a point in 2D space. In this way, we take those one-dimensional projections and map them back up into 2D. Now, all these reconstructions happen to lie along this line, but they are points in two-dimensional space.



Reconstruction from projection onto multiple directions

Projecting into the k-dimensional subspace defined by **orthonormal** $u_1, \ldots, u_k \in \mathbb{R}^d$.

The projection of x is a k-dimensional vector:

$$\begin{array}{ccc} & U^T & x \\ (\overset{\downarrow}{\underset{6}{\underline{x \cdot u_1}}}, \underset{-3}{\underline{x \cdot u_2}}, \ldots, \underset{2}{\overset{\downarrow}{\underline{x \cdot u_k}}}) = & \underbrace{\begin{pmatrix} \longleftarrow & u_1 & \longrightarrow \\ \longleftarrow & u_2 & \longrightarrow \\ & \vdots & \\ \longleftarrow & u_k & \longrightarrow \end{pmatrix}}_{\text{call this } U^T} & \begin{pmatrix} \uparrow \\ x \\ \downarrow \end{pmatrix} \end{array}$$

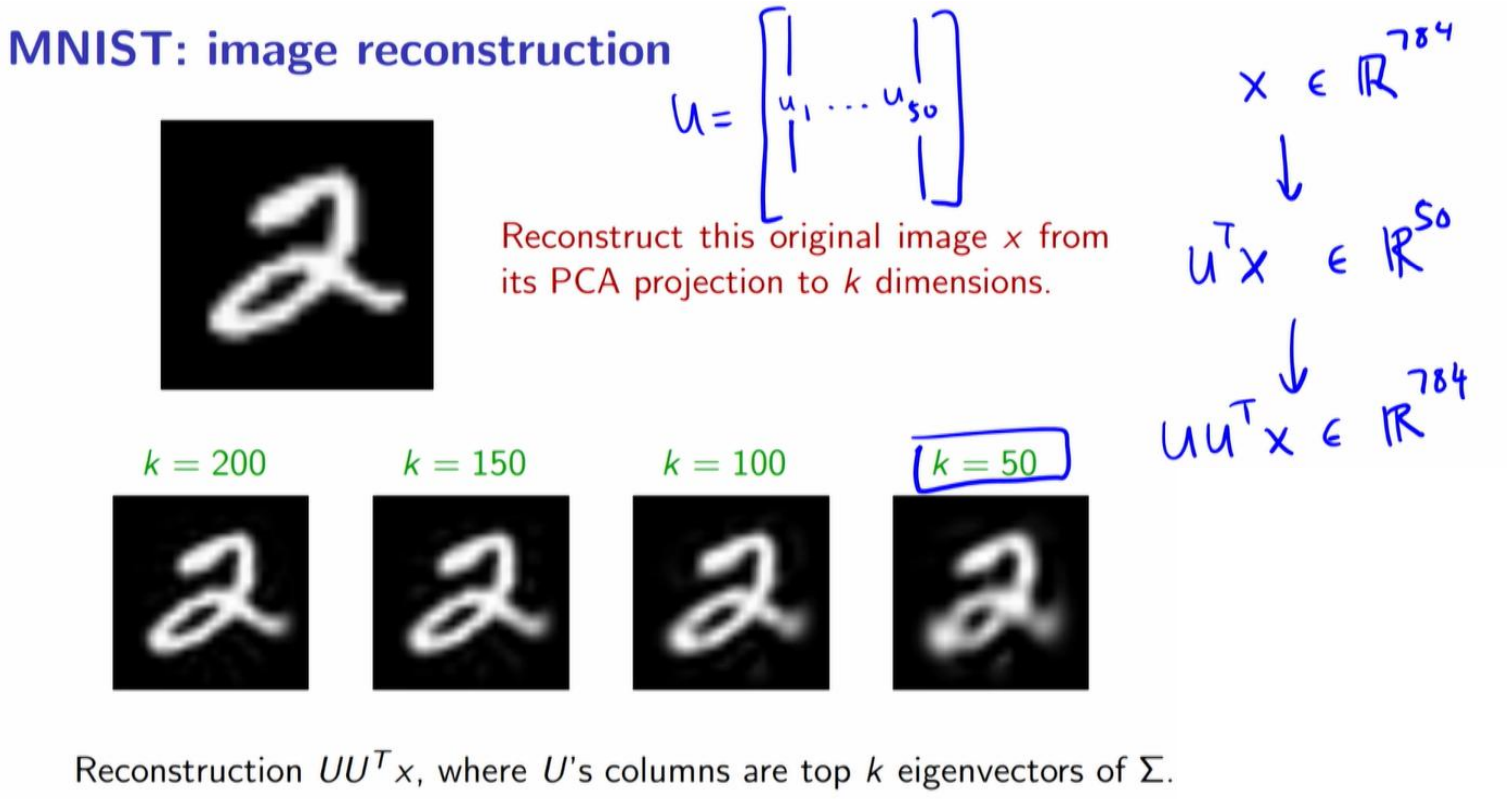
$6u_1 - 3u_2 + 2u_3 + \cdots$

The reconstruction from this projection is:

$$\underset{\sim}{(x \cdot u_1)}\underset{\uparrow}{u_1} + \underset{\sim}{(x \cdot u_2)}\underset{\uparrow}{u_2} + \cdots + (x \cdot u_k)u_k = \underset{\uparrow}{U}\underset{\sim}{U^T x}.$$

总结：

Let's go ahead and generalize this now. Now, we have data in d dimensions and we've projected them down to k dimensions. These are the directions we're projecting onto u1 through uk and the projection was as we saw before. We just take the dot product with these directions. Given these projections, how do we reconstruct? Again, the reconstruction is very simple. We just look at the k numbers in our projection. Let's say the numbers are, I don't know 6 minus 3, 2 and so on. For the reconstruction, we say this, "Okay, move six units "in direction u1, then move minus three units in direction "u2, then move two units in direction u3," and so on. Each of these vectors u1, u2, u3, these are vectors in the original d dimensional space, so this gives us a vector in d dimensions. More generally, what we have is we have a projection that looks like this and for the reconstruction, we say, "Move x.u1 "units in direction u1, "move x.u2 units in direction u2," and so on. Now, this looks especially simple if we write it in matrix vector form. If you recall, we have our vector x and we can take our k directions and stack them as columns of a matrix. That's matrix u, in which case this is u transpose and so the projection is just u transpose times x. The reconstruction takes this projection, u transpose x and just premultiplies it by u. That's it.



MNIST: image reconstruction

$$U = \begin{bmatrix} | & & | \\ u_1 & \cdots & u_{50} \\ | & & | \end{bmatrix}$$

Reconstruct this original image x from its PCA projection to k dimensions.

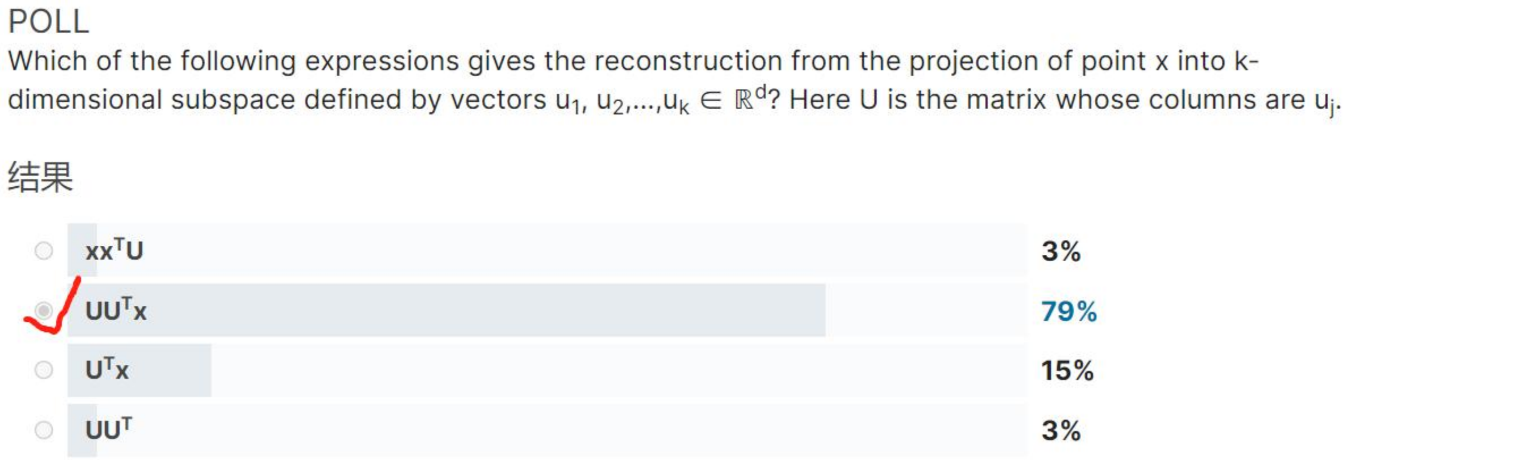
$$\begin{tikzcd} x \in \mathbb{R}^{784} \arrow[d] \\ U^T x \in \mathbb{R}^{50} \arrow[d] \\ UU^T x \in \mathbb{R}^{784} \end{tikzcd}$$

$k = 200$ $k = 150$ $k = 100$ $\boxed{k = 50}$

Reconstruction $UU^T x$, where U's columns are top k eigenvectors of Σ.

Now, let's return to those MNIST pictures and see exactly what was going on over here. We computed the covariance matrix of this dataset and we picked, let's say we're looking at k equals 50, we picked the top 50 eigenvectors of this covariance matrix. Let's call them u1 through u50. This is our matrix u. Now, the way we projected data is that we started with a point in 784 dimensional space. The projection was simply u transpose x, so this is a point in 50 dimensional space, just 50 numbers. Now, when we want to reconstruct something in the original space, we just multiply by the matrix u and that's again in 784 dimensional space. That's how we take a projection and recreate a point in the original space.

That's it for today. If one had to pick a small handful of the absolute, most important primitives for statistics and machine learning, principal component analysis would certainly be on that list. It's an oldie, but a goldie.



POLL

Which of the following expressions gives the reconstruction from the projection of point x into k-dimensional subspace defined by vectors $u_1, u_2, \ldots, u_k \in \mathbb{R}^d$? Here U is the matrix whose columns are u_j.

结果

○ xx^TU 3%

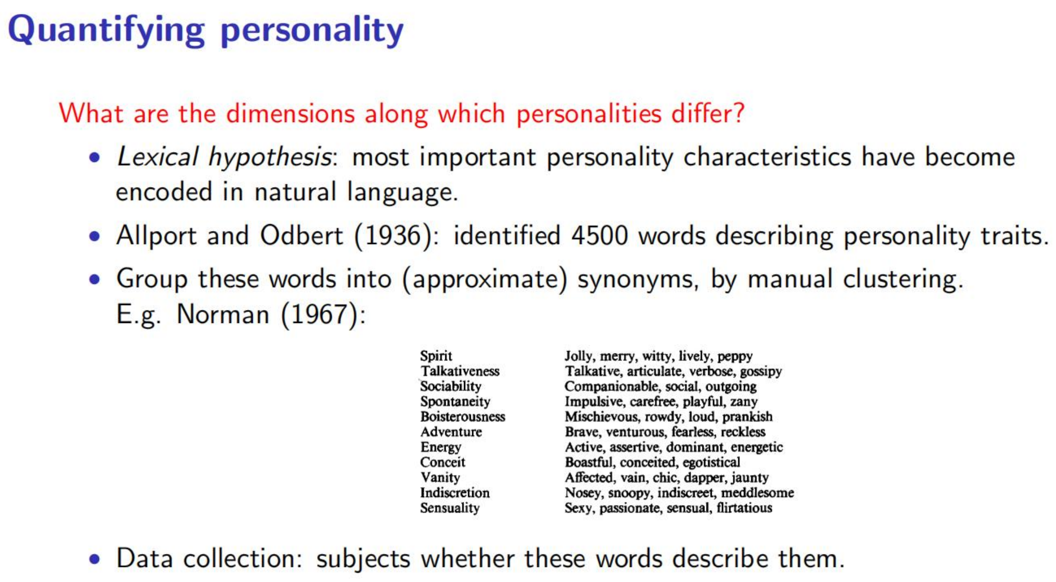
[✓ ● UU^Tx] 79%

○ U^Tx 15%

○ UU^T 3%

总结：

9.4 Case Study: Personality Assessment



Quantifying personality

What are the dimensions along which personalities differ?

- *Lexical hypothesis*: most important personality characteristics have become encoded in natural language.
- Allport and Odbert (1936): identified 4500 words describing personality traits.
- Group these words into (approximate) synonyms, by manual clustering. E.g. Norman (1967):

Spirit	Jolly, merry, witty, lively, peppy
Talkativeness	Talkative, articulate, verbose, gossipy
Sociability	Companionable, social, outgoing
Spontaneity	Impulsive, carefree, playful, zany
Boisterousness	Mischievous, rowdy, loud, prankish
Adventure	Brave, venturous, fearless, reckless
Energy	Active, assertive, dominant, energetic
Conceit	Boastful, conceited, egotistical
Vanity	Affected, vain, chic, dapper, jaunty
Indiscretion	Nosey, snoopy, indiscreet, meddlesome
Sensuality	Sexy, passionate, sensual, flirtatious

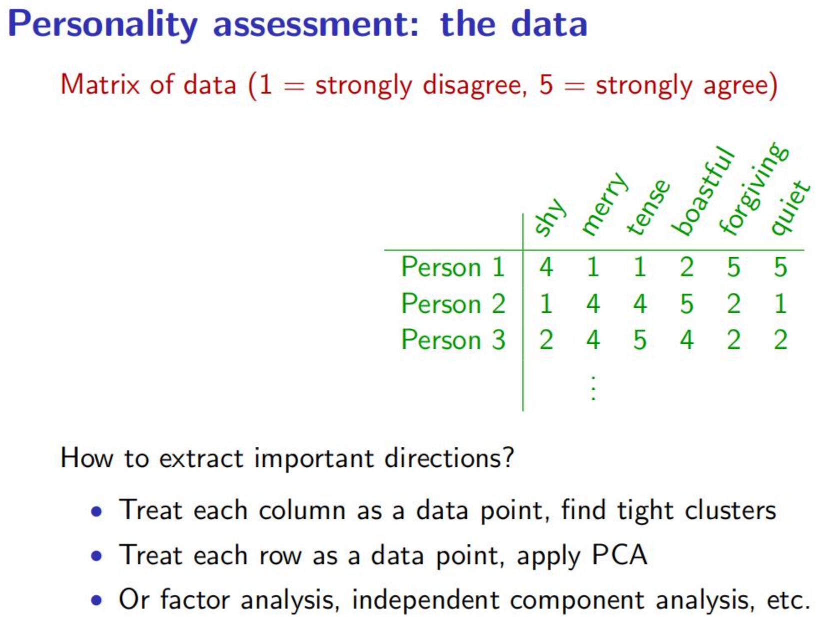
- Data collection: subjects whether these words describe them.

Principal component analysis, or PCA,has been around a very long time. At least a century. It's a basic tool in essentially any domain that uses statistics.

What we'll do today is to look at a case study of a particular application of PCA for quantifying personality. So, we all have different personalities and these personalities seem to be multidimensional. Any two people differ in some ways and are similar in other ways. Is there a way to quantify all this? For instance, can we represent each person's personality as a point in some low-dimensional space? This kind of question has occupied people forever and some of the brightest minds, people like Carl Jung, have weighed in with partial answers and usually qualitative criteria of various sorts.

But now that we live in an era of data collection, it seems plausible that we might be able to make this quantitative. What I'll be talking about today is some progress in this direction. The starting point is what is usually called the **lexical hypothesis**.

This is the simple idea that any personality trait that's of even the remotest interest must have a corresponding English word, like brave, or sensitive, or malicious. So, with this in mind, two researchers in the 1930's, Allport and Odbert, sat down with the English dictionary and just went through it carefully, looking for all words that could be used to differentiate between people. They found 18,000 such words. Now, not all of these referred to personality traits. Some of them were things like short and tall. So, a second pass was made in which this list of 18,000 was further refined to pull out personality traits. That yielded 4,500 words. Now that's still a pretty large list. It turned out though that a lot of these words were synonymous things, like jolly, merry, peppy. So, a different group of researchers, in the 1960's, manually clustered these words into groups of synonyms. This boiled the list down to about 500 distinct personality words. At that point, it was time for data collection and this was done in a very interesting way. People were brought in and each person was shown the list of 500 words and asked, "Which of these words describes you?"



Personality assessment: the data

Matrix of data (1 = strongly disagree, 5 = strongly agree)

	shy	merry	tense	boastful	forgiving	quiet
Person 1	4	1	1	2	5	5
Person 2	1	4	4	5	2	1
Person 3	2	4	5	4	2	2
		⋮				

How to extract important directions?

- Treat each column as a data point, find tight clusters
- Treat each row as a data point, apply PCA
- Or factor analysis, independent component analysis, etc.

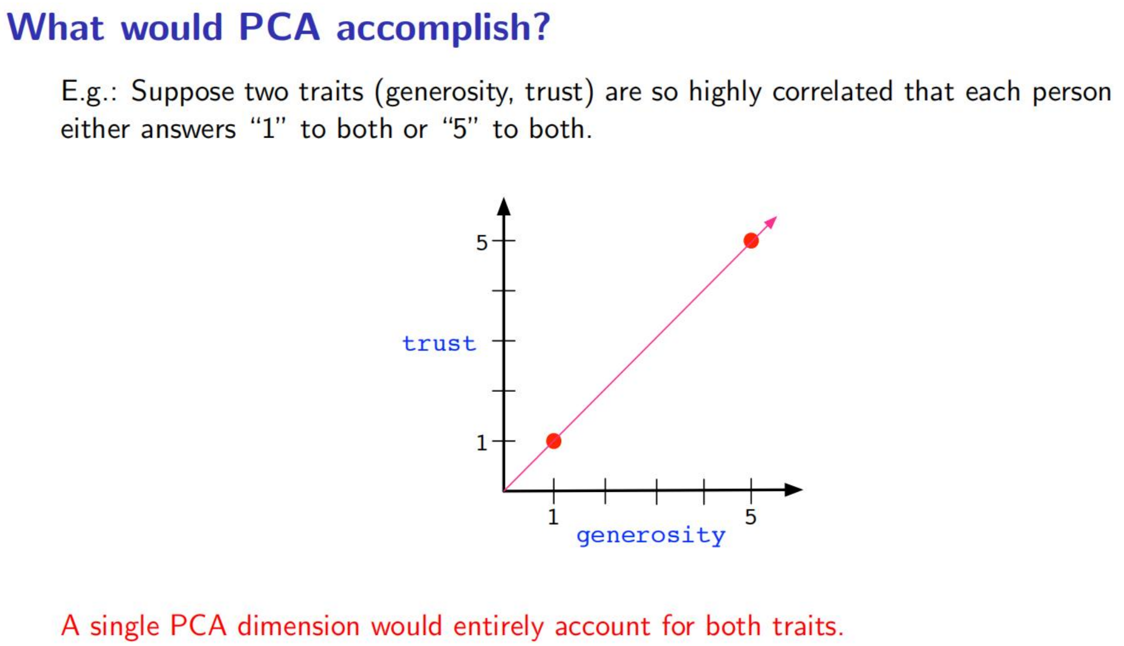
Actually this was done in a slightly more graded way. Each person was given the 500 words, one at a time, say one of the words was shy, and asked does the word shy describe you? Say one if you strongly disagree, five if you strongly agree, or something in between, two, three, or four. In this way, an entire matrix of data was collected. There were 500 words, 500 columns, and each row were the responses for a particular person.

Given data of this kind, how can we get a low-dimensional representation for personality? There are actually many ways to proceed.

- One idea is to think of the columns as being data points and to cluster them. This would give us clusters of similar personality words. For example, suppose it turns out that people who are shy also tend to be forgiving. Then those two words would end up in the same cluster.
- Alternatively, we could think of the data points as being the rows, so there's one data point per person. Then we could apply principal component analysis to this data or factor analysis or some other form of representation learning.

Researchers basically tried all of these and it turned out that to some extent the results were fairly consistent. So, what I'll do is describe the PCA approach today.

总结：

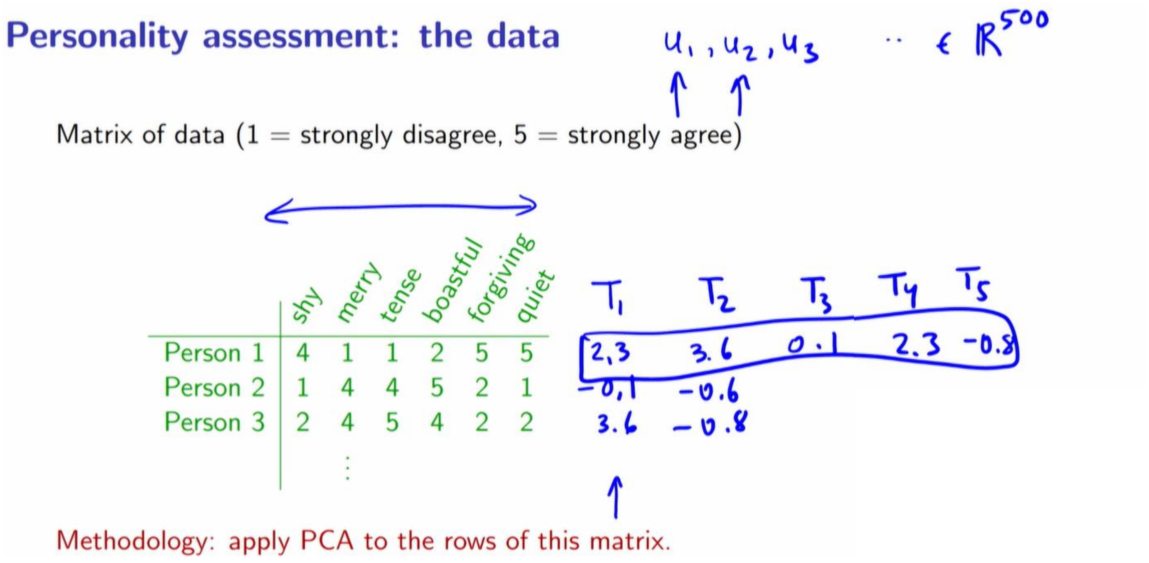


What would PCA accomplish?

E.g.: Suppose two traits (generosity, trust) are so highly correlated that each person either answers "1" to both or "5" to both.

A single PCA dimension would entirely account for both traits.

Let's start with a little bit of intuition. Suppose there are two traits, like generosity and trust, that happen to be very strongly correlated. I don't know if that's actually the case for generosity and trust, this is just a hypothetical example. But if they were perfectly correlated then what would happen is that everybody's answer to generosity would be exactly the same as their answer to trust. So, the answers would all lie along this line, the Y equals X line. If we then asked principal component analysis to find important directions in the data, which direction would it find? It would find that direction. From that one PCA direction, from that one number, we would be able to perfectly reconstruct the generosity value as well as the trust value. This is sort of the intuition that we're trying to get at. But let's see what happens in practice.



Personality assessment: the data

$u_1, u_2, u_3 \cdots \in \mathbb{R}^{500}$

Matrix of data (1 = strongly disagree, 5 = strongly agree)

	shy	merry	tense	boastful	forgiving	quiet	T_1	T_2	T_3	T_4	T_5
Person 1	4	1	1	2	5	5	2,3	3.6	0.1	2.3	-0.8
Person 2	1	4	4	5	2	1	-0,1	-0.6			
Person 3	2	4	5	4	2	2	3.6	-0.8			
		⋮									

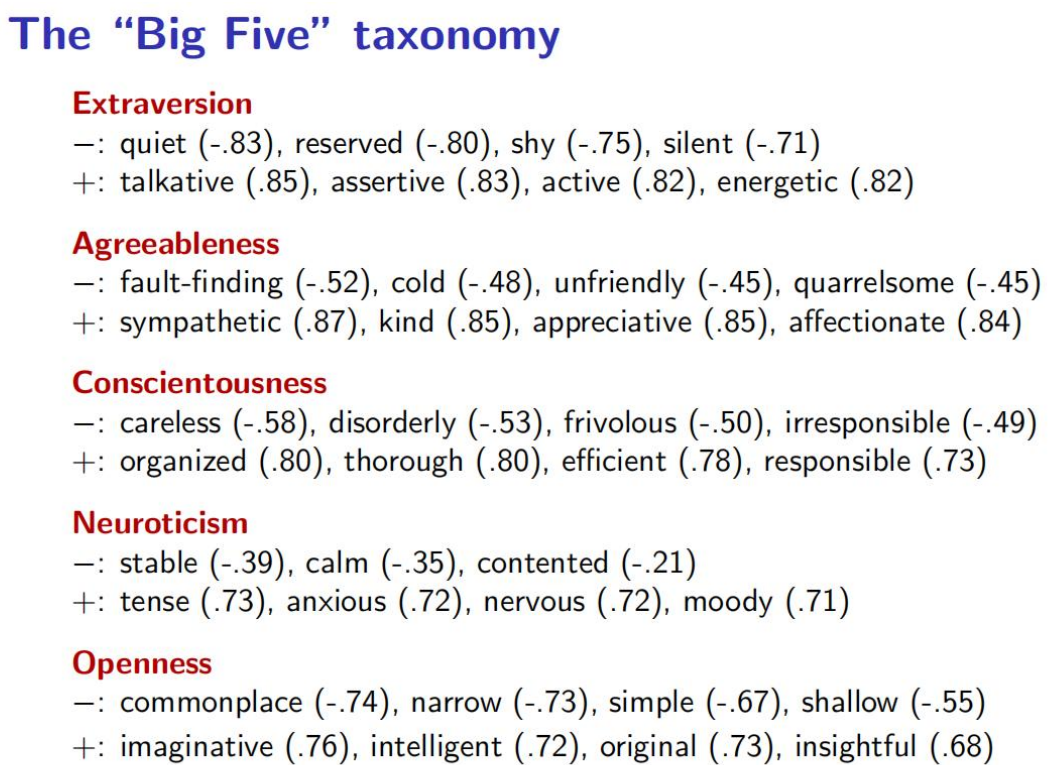
Methodology: apply PCA to the rows of this matrix.

We have this matrix of data and what we'll do is think of each row as a data point. So, we have 500 words and so we have data points in 500 dimensional space. We now apply principal component analysis and we get the principal components. Let's say they are U one, U two, U three, and so on. These are directions in 500 dimensional space. Let's pick the first direction, U one, the first principal component. What we can do is to project the first person onto this component. How? Well, we just take that person's row and take it's dot product with U one. This gives us some number, like maybe two point three. Then we'll look at the projection of the second person onto U one. We take the second person's row and take its dot product with U one and this gives us some other number, maybe negative point one. Then the third person and so on. In this way, we get an entire column of numbers that is essentially a new synthetic personality trait. Let's call it T one for trait number one.

Then we can do the same thing with the second principal component. We take everybody's dot product with the direction U two and we get another bunch of numbers. We'll call this T two. Then we can do T three, and T four, and T five, and so on, and if we go out to T five, then we get five numbers for each person. Five new numbers. Essentially, an embedding of that person into a five-dimensional space. A quantification of their personality.

So, what exactly do these traits refer to? How do we understand what they mean? What is trait T one? This is a trait that was automatically extracted from data. It's something synthetic. How do we understand what aspect of personality it is capturing? In order to figure that out what we can do is just look at that. We can just look at our particular column and then go through the 500 columns that correspond to English words. So, we go through these 500 columns and we see which of them is closest to T one, which of them has the smallest angle with T one, or the largest dot product with T one. By seeing what those words are, we'll get an idea of what aspect of personality T one is capturing. Then we do this for T two, T three, T four, and T five. When we do this, we get what is often called the **Big Five taxonomy**.

总结：



The “Big Five” taxonomy

Extraversion
−: quiet (-.83), reserved (-.80), shy (-.75), silent (-.71)
+: talkative (.85), assertive (.83), active (.82), energetic (.82)

Agreeableness
−: fault-finding (-.52), cold (-.48), unfriendly (-.45), quarrelsome (-.45)
+: sympathetic (.87), kind (.85), appreciative (.85), affectionate (.84)

Conscientousness
−: careless (-.58), disorderly (-.53), frivolous (-.50), irresponsible (-.49)
+: organized (.80), thorough (.80), efficient (.78), responsible (.73)

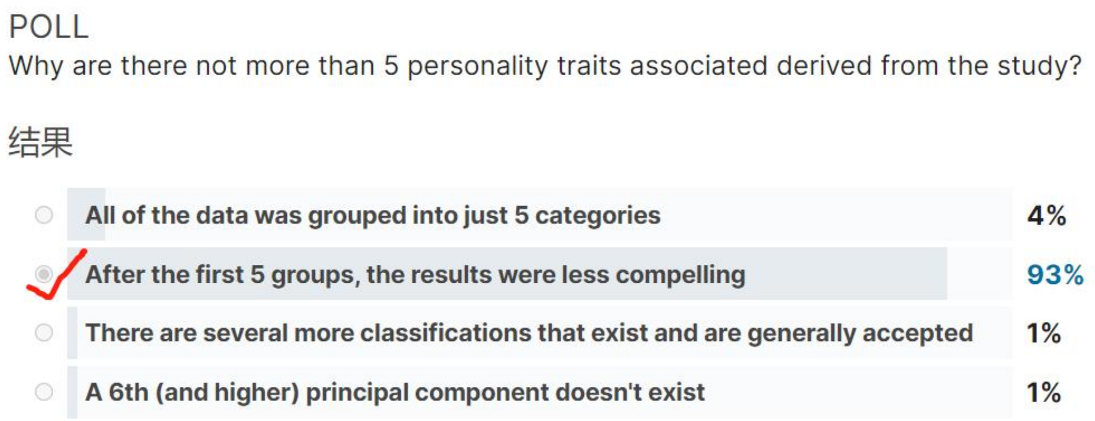
Neuroticism
−: stable (-.39), calm (-.35), contented (-.21)
+: tense (.73), anxious (.72), nervous (.72), moody (.71)

Openness
−: commonplace (-.74), narrow (-.73), simple (-.67), shallow (-.55)
+: imaginative (.76), intelligent (.72), original (.73), insightful (.68)

- The first principal component, T one, turns out to have a high positive dot product with words like talkative, assertive, active, energetic. It has a high negative dot product with words like quiet, reserved, shy, and silent. Just looking at those, we get an intuitive sense for what this particular principle component is capturing. Researchers decided to call this extraversion. This is a name that they cooked up for this synthetically derived trait.
- The second principal component turned out to be highly correlated with sympathetic, kind, appreciative and negatively correlated with fault-finding, cold, quarrelsome. This got called agreeableness.
- The third trait was highly correlated with organized, thorough, efficient, responsible, and negatively correlated with frivolous, careless, and so on. So, it got called conscientousness.
- The fourth one had a large dot product with tense, anxious, nervous, moody, and so that got called neuroticism.
- Then the final one, the fifth one, turned out to have a fairly large dot product, maybe not quite as large as the previous ones, with imaginative, intelligent, original and a negative correlation with words like narrow and shallow. This was called openness.

These all seem quite reasonable and then one might wonder why not keep going? Why did we stop at five? Why not go to the sixth principal component, and the seventh, and the eighth? Well, it turned out that at that point the results started becoming a little less compelling. One of the nice things about the first five traits here, or at least the first four of them, is that similar results were obtained when using other kinds of data analysis, like clustering or factor analysis. That kind of consistency went away though beyond the fifth trait or so. So, this is really the subset that people were able to agree upon. This kind of quantification is now used in many contexts. For example, in computerized matchmaking.

That's it for today. We've seen an application of principal component analysis at work, we've seen how data and PCA can be used to answer some very fundamental questions about human beings. See you next time.



POLL
Why are there not more than 5 personality traits associated derived from the study?

结果

○ **All of the data was grouped into just 5 categories** **4%**

[✓ ● **After the first 5 groups, the results were less compelling**] **93%**

○ **There are several more classifications that exist and are generally accepted** **1%**

○ **A 6th (and higher) principal component doesn't exist** **1%**

总结：

9.5 Linear Algebra V: Eigenvalues and Eigenvectors

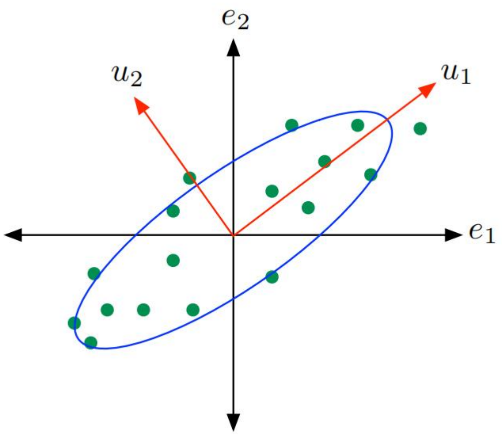
Topics we'll cover

1. Moving between coordinate systems
2. Eigenvectors and eigenvalues of a square matrix
3. Orthonormal basis of eigenvectors for symmetric matrix

The notion of eigenvectors and eigenvalues have been popping up now and again. And we keep postponing the discussion. Well, the time is finally upon us. So what we'll talk about today is finding a natural coordinate system for a data set.

When we think of data, we're used to thinking about it in terms of the features we have selected, the individual coordinates of the data vectors. But it turns out that data analysis often becomes much simpler if we move to an alternative coordinate system, and this is what leads ultimately to the notion of an eigenvector.

Moving between coordinate systems



In this picture, we have a simple two dimensional data set, the two features, and they are positively correlated. If we were to fit a Gaussian to this data, it would look something like this. What I'm going to do is to show you an alternative coordinate system for this data. This one over here. Instead of the regular coordinate axes, e1, e2, we have these tilted axes, u1, u2. We could move quite easily between these two systems. It's a simple rotation. And in many ways, these red coordinates, u one and u two, are more natural for this data.

What's so good about them? If you tilt your head slightly, you'll see that in this new basis, the two coordinates are actually uncorrelated. For instance, we could fit a diagonal Gaussian to them. Things get simpler in this alternative coordinate system.

The linear function defined by a matrix

- Any matrix M defines a linear function, $x \mapsto Mx$. If M is a $d \times d$ matrix, this maps $\mathbb{R}^d$ to $\mathbb{R}^d$.
- This function is easy to understand when M is **diagonal**:

$$e_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

$$Me_1 = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix} = 2e_1$$

$$\underbrace{\begin{pmatrix} 2 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 10 \end{pmatrix}}_{M} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} 2x_1 \\ -x_2 \\ 10x_3 \end{pmatrix}}_{Mx}$$

In this case, M simply scales each coordinate separately.

- General symmetric matrices also just scale coordinates separately... but in a **different coordinate system!**

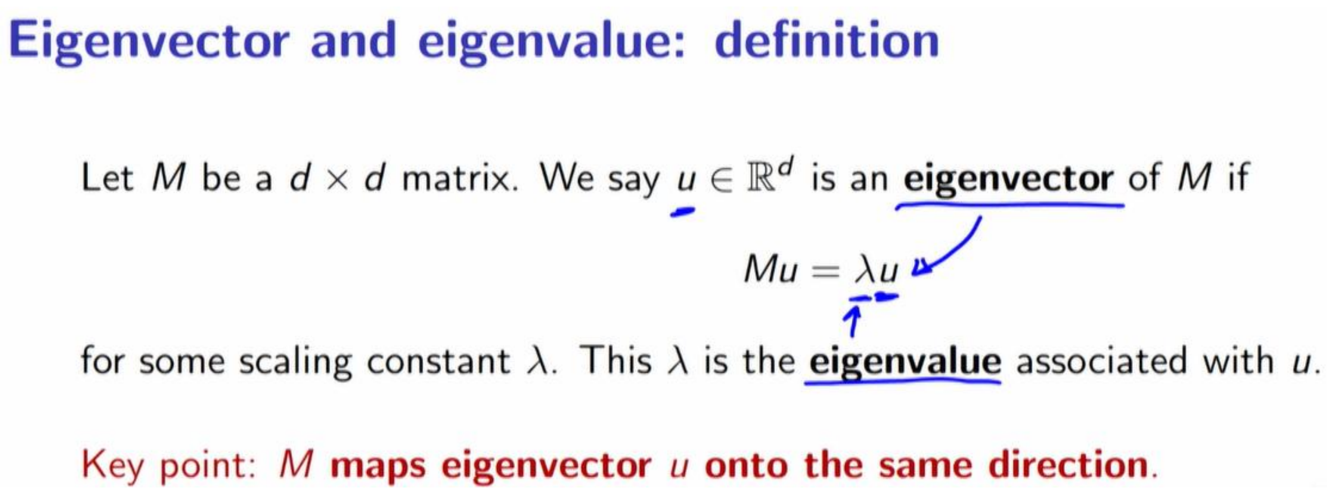
总结：

So let's try and formalize some of this. And for that, we're gonna need linear algebra. So let M be any d by d matrix, any square matrix. And as we've seen a matrix like this defines a linear function. A function that takes a vector x and sends it to some other d dimensional vector Mx. So, okay, it's a linear function. And we've seen examples of this. But what exactly is this function doing? It's hard to visualize or imagine what this transformation is from one d dimensional vector to another. But that's a special case in which these transformations are actually very simple to understand. And that is when the matrix is diagonal.

Okay, so here we have an example of a diagonal matrix, a three by three matrix. So this defines a linear transformation from three dimensional vectors to three dimensional vectors. And because this matrix is diagonal, it's very easy to say what this transformation is. It takes a vector, x one, x two, x three, and simply doubles x one, multiplies x two by minus one, and multiplies x three by 10. A diagonal matrix is easy to understand because it is simply scaling each coordinate separately. The coordinates don't get mixed up at all.

So that's a diagonal matrix. What about a more general symmetric matrix? In that case, there are lots of off diagonal entries and they mix up the different coordinates, which can be confusing. But it turns out that if we simply change the coordinate system, if we move from our own coordinate system, the x one, x two, x three axes to a different coordinate system, a coordinate system that is more natural for the matrix. Well, in that alternative basis, the matrix again becomes something very simple. It becomes a diagonal matrix. So any symmetric matrix is actually once again a simple diagonal matrix if we move to its own coordinate system.

So what might these **natural coordinates** be? How might one define them? Well, if we look at this diagonal example over here, so what we are saying is that for a diagonal matrix, its natural coordinates are just the regular coordinates. So, for example, the x one coordinate, which we can write like this. So what happens when we apply M to e one? Well, we simply double that first coordinate. So we get twice e one. In other words, M sends e one to another vector in exactly the same direction. It does not bleach into other directions. And that's why things are so simple (图中蓝色字迹).



Eigenvector and eigenvalue: definition

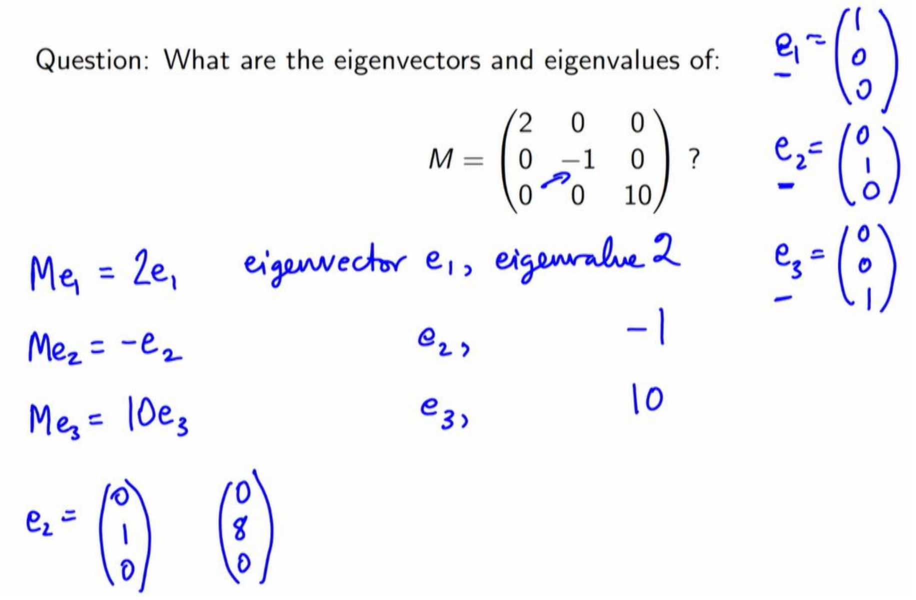
Let M be a $d \times d$ matrix. We say $u \in \mathbb{R}^d$ is an **eigenvector** of M if

$$Mu = \lambda u$$

for some scaling constant λ. This λ is the **eigenvalue** associated with u.

Key point: M **maps eigenvector** u **onto the same direction**.

So this gives us an idea of how we might define the notion of a natural direction. Let M by any d by d matrix. And we'll say u is an eigenvector of M if M sends u to another vector in the same direction. So in some sense, this means that u is a natural direction for M. M keeps u in exactly the same direction. M times u is just a constant, a scaling constant times u. And the scaling constant is called the eigenvalue associated with that particular eigenvector u. Eigenvalues and eigenvectors come in pairs.



Question: What are the eigenvectors and eigenvalues of:

$$M = \begin{pmatrix} 2 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 10 \end{pmatrix} ?$$

$\underline{e}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ $\underline{e}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ $\underline{e}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$

$Me_1 = 2e_1$ eigenvector e_1, eigenvalue 2

$Me_2 = -e_2$ e_2, -1

$Me_3 = 10e_3$ e_3, 10

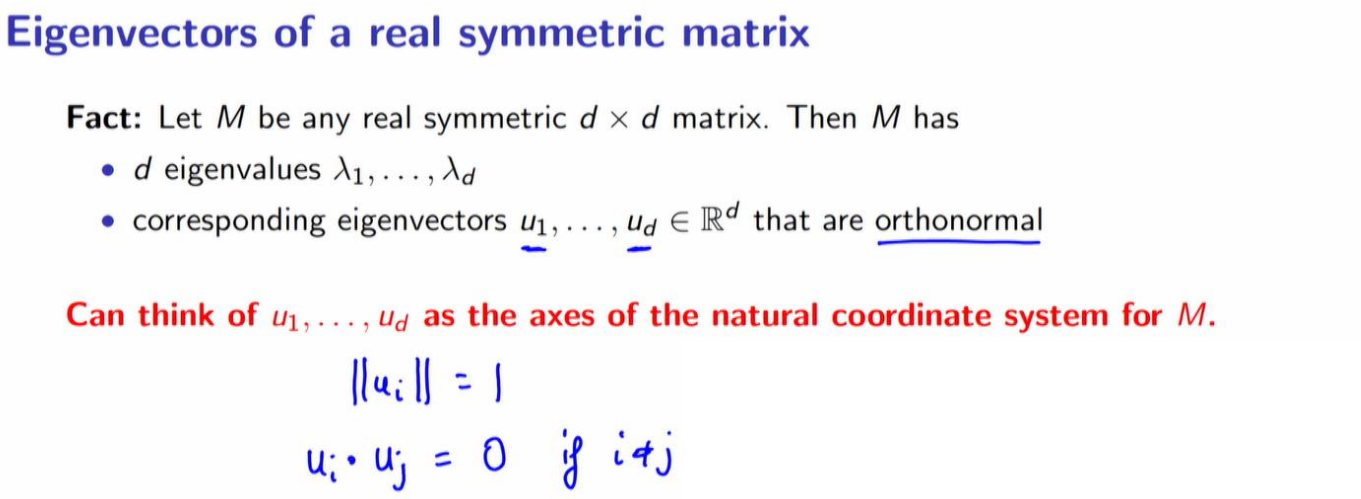
$e_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ $\begin{pmatrix} 0 \\ 8 \\ 0 \end{pmatrix}$

总结:

Let's return to the example of the diagonal matrix we just saw. What are its eigenvectors and eigenvalues? Well, we've suggested that the natural directions for this matrix are just the coordinate directions. So let's try those. The coordinate directions are e one, which is one zero zero, e two, which is zero one zero, and e three, which is zero zero one. What does M do to these directions? So M times e one is simply twice e one. It sends e one in the same direction. And so e one really is an eigenvector of this matrix. So e one is an eigenvector, and the corresponding eigenvalue is just the scaling value, two. So eigenvector, e one, eigenvalue, two. Now, M sends e two to simply minus e two. And so e two is also an eigenvector. And the eigenvalue in that case is the scaling factor minus one. And M sends e three to 10 times e three. So e three is an eigenvector with corresponding eigenvalue, 10. Okay, so e one, e two, and e three are natural directions, or eigenvectors for this diagonal matrix.

Are there any other eigenvectors? There are actually lots of other eigenvectors because if e two is an eigenvector, then any multiple of e two is also by definition automatically an eigenvector. For example, if we take some multiple of it, like 0 8 0, that's also an eigenvector because M times that is minus one times that. So any multiple of e two is also an eigenvector with the same eigenvalue minus one. For this reason, what we often do is to normalize eigenvectors to unit length, just like e1, e2, and e3.

Okay, so let's rephrase the question then. Are there any other unit length eigenvectors of this matrix? And, no, there aren't. So we have a three by three matrix with exactly three unit length eigenvectors. Hmm, is that always the case? Is the number of distinct unit length eigenvectors always just the dimension of the matrix? And for our purposes, the answer is yes.



Eigenvectors of a real symmetric matrix

Fact: Let M be any real symmetric $d \times d$ matrix. Then M has

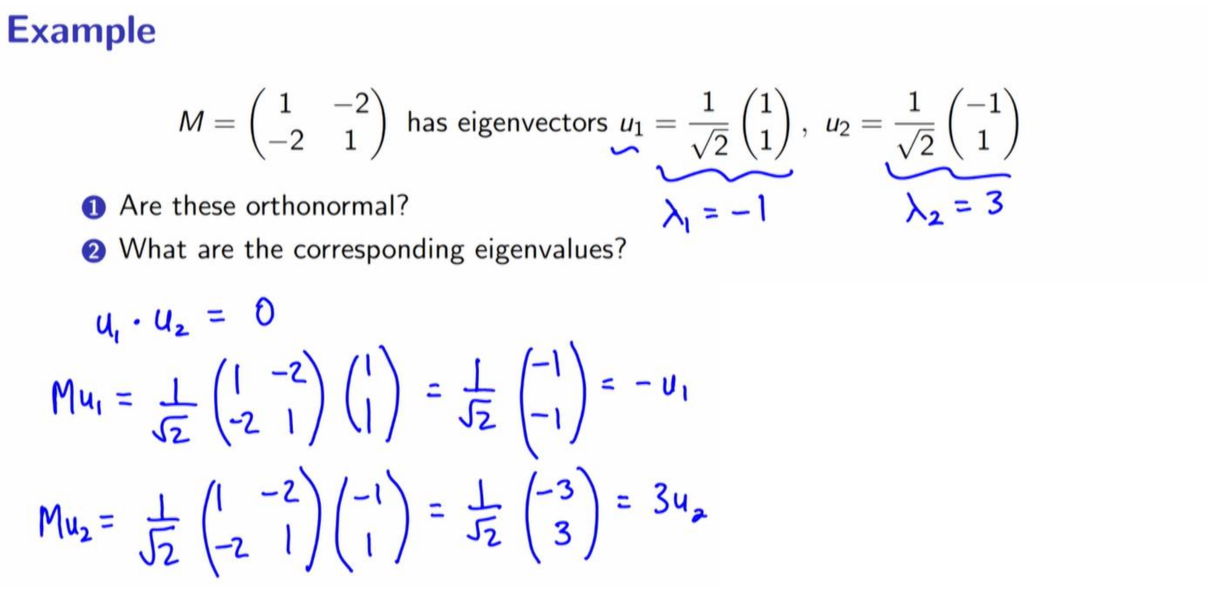
- d eigenvalues $\lambda_1, \dots, \lambda_d$
- corresponding eigenvectors $u_1, \dots, u_d \in \mathbb{R}^d$ that are orthonormal

Can think of $u_1, \dots, u_d$ as the axes of the natural coordinate system for M.

$$\|u_i\| = 1$$

$$u_i \cdot u_j = 0 \quad \text{if } i \neq j$$

The other thing to note over here is that the eigenvectors are all orthogonal to each other, the dot product of any pair of them is zero. Is that also always the case? Are different eigenvectors always orthogonal to each other? And again, the answer is yes. And this means that the eigenvectors constitute a basis. So, let's go ahead and see how this generalizes. So let's take any symmetric square matrix. If it's a d by d matrix, then it is d eigenvectors. u1 through ud. And these are orthonormal. Okay, so in other words, they all have unit length. The length of each of them is one. And any pair of them are orthogonal to each other. So ui dot uj is zero for any i not equal to j. This means that these d eigenvectors constitute a basis, and we'll think of them as the axes of a natural coordinate system for this matrix. It turns out that transforming data into this coordinate system simplifies a lot of different operations.



Example

$$M = \begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix} \text{ has eigenvectors } u_1 = \underbrace{\frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}}_{\lambda_1 = -1}, \quad u_2 = \underbrace{\frac{1}{\sqrt{2}}\begin{pmatrix} -1 \\ 1 \end{pmatrix}}_{\lambda_2 = 3}$$

1. Are these orthonormal?
2. What are the corresponding eigenvalues?

$$u_1 \cdot u_2 = 0$$

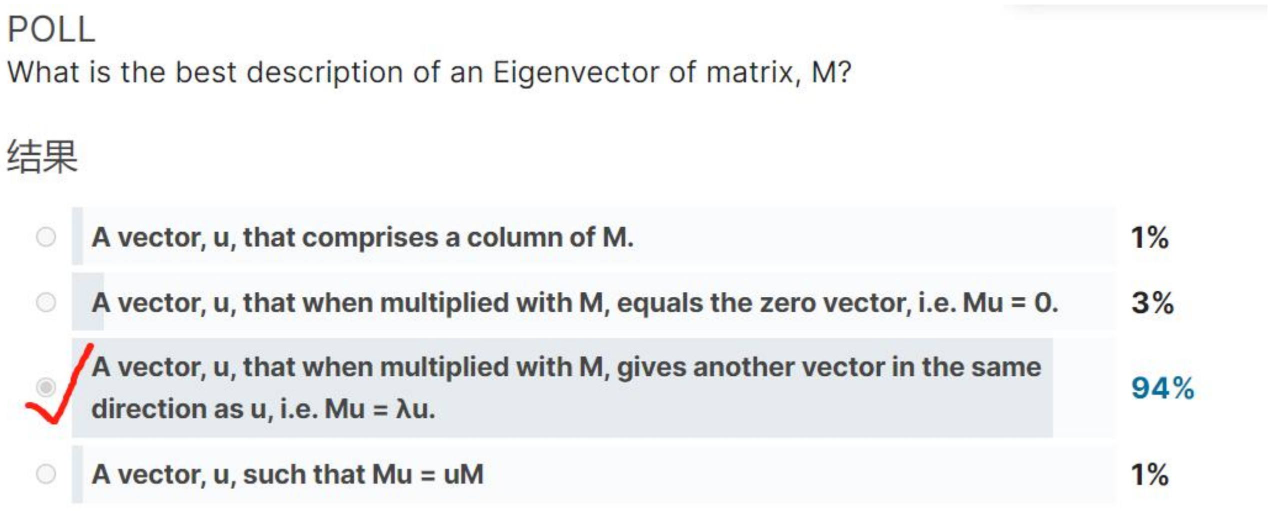
$$Mu_1 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} -1 \\ -1 \end{pmatrix} = -u_1$$

$$Mu_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}\begin{pmatrix} -1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} -3 \\ 3 \end{pmatrix} = 3u_2$$

总结：

So before we get into that. Let's see another simple example. And one in which the eigenvectors are not the coordinate directions. So here's a two by two matrix. And I'm giving you the eigenvectors for the matrix u one and u two. And what we want to do is compute the corresponding eigenvalues. Okay, so these are the eigenvectors. Let's first check that they really do constitute an orthonormal basis. Okay, so we have two of them, which makes sense for a two by two matrix. Are they unit length? Yes, their length of one. Are they at right angles to each other? What is u one dot u two? Well, the dot product in this case is zero. So, indeed, these two eigenvectors are orthonormal. What are their eigenvalues? Okay, so let's see, m times u one. Well, we can bring the square root two in front. So one over square root two times one negative two negative two one times one one. And this works out to negative one negative two plus one. So negative one. Okay, so it sends the vector u one to minus one times the same vector. So the corresponding eigenvalue is negative one. Now let's look at the other vector, M times u two. And again we just pulled the square root two out in front. So we have one minus two, minus two one times negative one one. And we just have the square root two again. And this time we have minus one, so minus three and three. So this is exactly three times u two. And so the second eigenvalue is three. And that's it.

Okay, so that's enough for today. We started by asking about what might be considered a natural coordinate system for a data set? And this led to the notion of an eigenvector. Next time we'll see how any matrix, and any particular a data matrix, can be written in terms of eigenvectors and eigenvalues. This turns out to be a very powerful tool for data analysis called **spectral decomposition**. See you then.



POLL

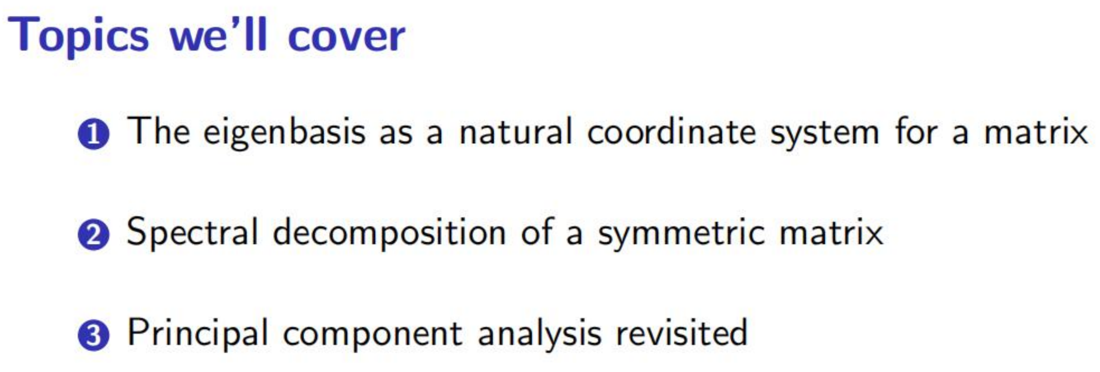
What is the best description of an Eigenvector of matrix, M?

结果

○ A vector, u, that comprises a column of M.	1%
○ A vector, u, that when multiplied with M, equals the zero vector, i.e. Mu = 0.	3%
● A vector, u, that when multiplied with M, gives another vector in the same direction as u, i.e. Mu = λu. ✓	94%
○ A vector, u, such that Mu = uM	1%

总结：

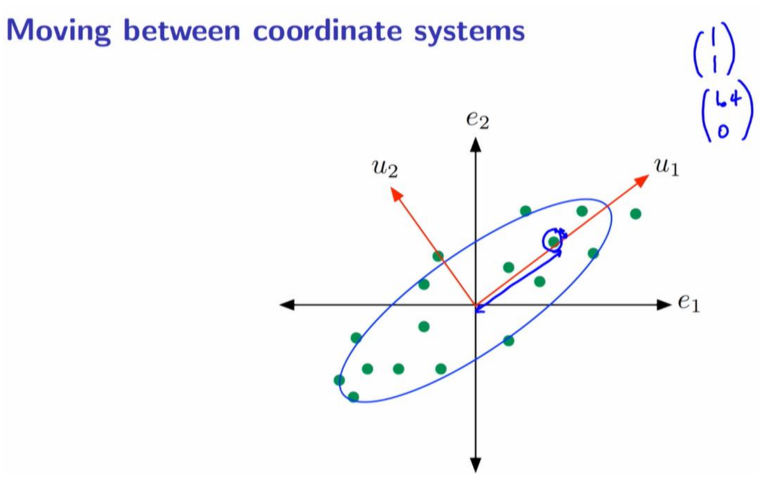
9.6 Linear Algebra VI: Spectral Decomposition



Topics we'll cover

1. The eigenbasis as a natural coordinate system for a matrix
2. Spectral decomposition of a symmetric matrix
3. Principal component analysis revisited

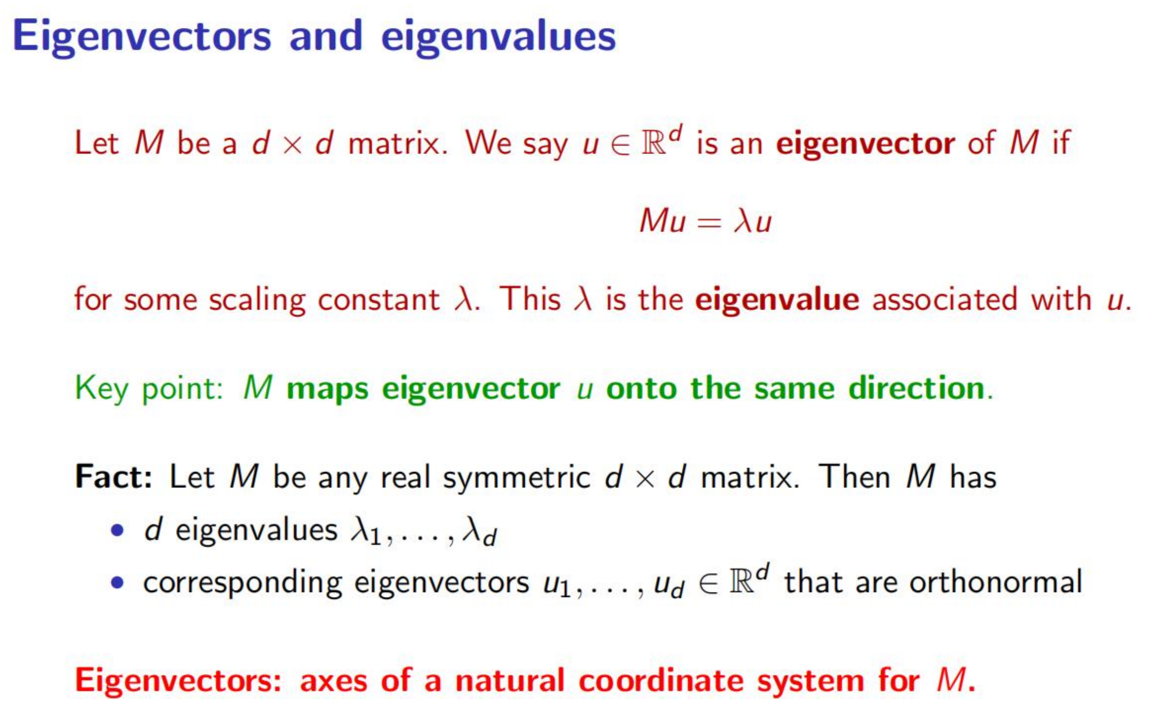
Last time, we introduced eigenvectors and eigenvalues. Today, we'll see how these can be used to understand matrices in a simple and intuitive way, and how this leads to principle component analysis.
So our main result today will be the **spectral decomposition theorem**, which tells you how a matrix can be rewritten in a different way, solely in terms of its eigenvectors and eigenvalues. This is a very powerful tool of data analysis.



Moving between coordinate systems

$\begin{pmatrix}1\\1\end{pmatrix}$
$\begin{pmatrix}1.4\\0\end{pmatrix}$
e_2
u_2
u_1
e_1

So let's start by quickly reviewing our motivation. Here's a simple 2D data set, and the two features are correlated, which is a little bit of a complication. But what if we switch to a different coordinate system, something like this one? What if we switch to these axes? So, first of all, what would this entail? Well, let's take a point, say, this one. In the original coordinate system, this might be something like one, one. In the new coordinate system, what would this be? Well, it would be this length over here, which may be something like 1.4, and this height over here, which is approximately zero. Two different representations for the same point. Now, the way we move back and forth between these representations is in this case just a simple rotation. It's a simple linear transformation. What is the advantage of this alternative coordinate system? Why might we want to work in that basis? Well, if you tilt your head a little bit, you'll see that in this alternative representation, the two coordinates are actually uncorrelated. That correlation goes away. In some sense, the data becomes a little bit simpler in this alternative coordinate system, and therefore, this alternative basis might be a natural place in which to do data analysis. So what are these natural coordinates? So this led us last time to the notion of an eigenvector.



Eigenvectors and eigenvalues

Let M be a $d \times d$ matrix. We say $u \in \mathbb{R}^d$ is an **eigenvector** of M if

$$Mu = \lambda u$$

for some scaling constant λ. This λ is the **eigenvalue** associated with u.

Key point: M **maps eigenvector** u **onto the same direction**.

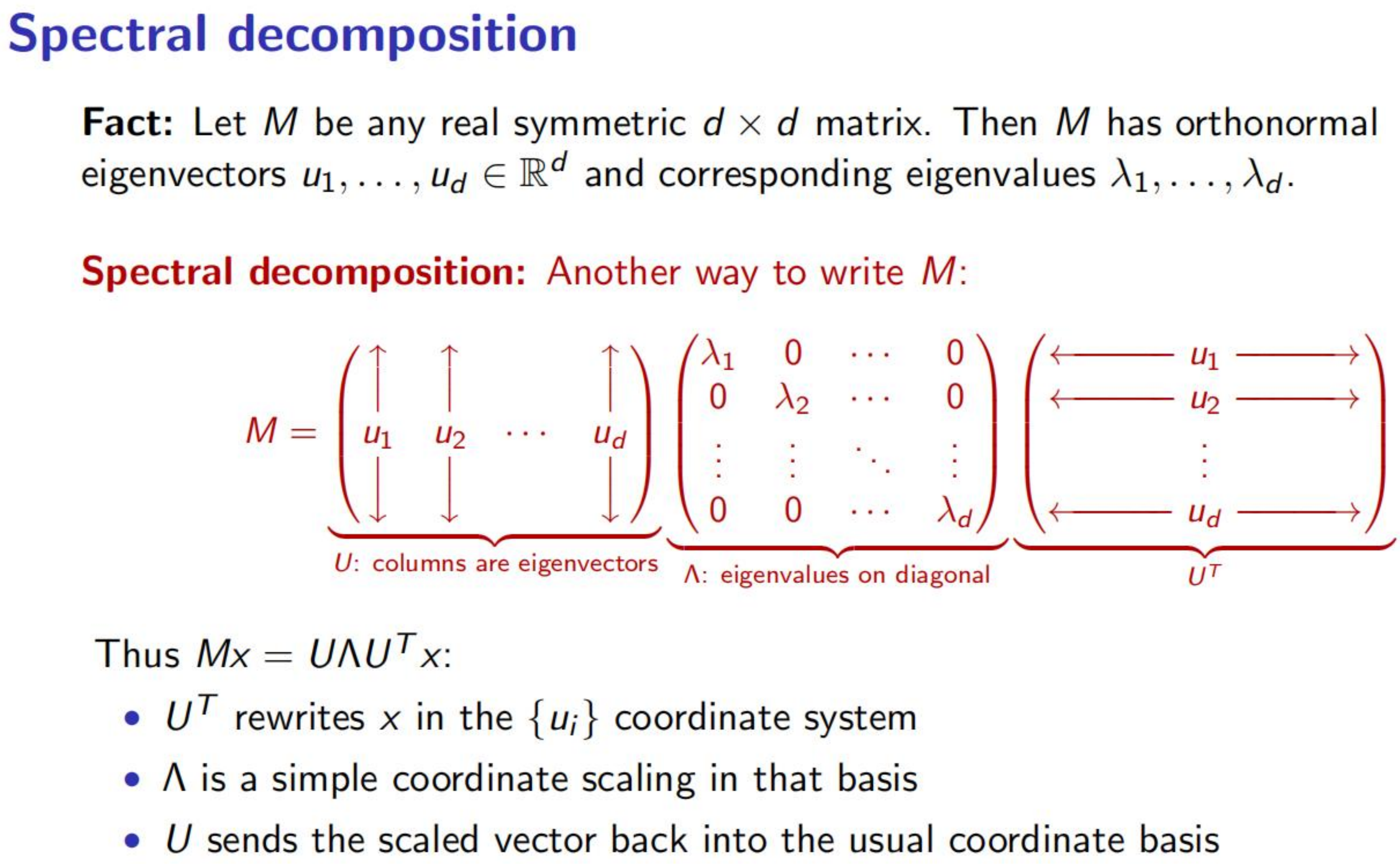
Fact: Let M be any real symmetric $d \times d$ matrix. Then M has

- d eigenvalues $\lambda_1, \ldots, \lambda_d$
- corresponding eigenvectors $u_1, \ldots, u_d \in \mathbb{R}^d$ that are orthonormal

Eigenvectors: axes of a natural coordinate system for M**.**

总结：

For any d by d matrix, M, an eigenvector is a vector, u, that is sent by M to another vector in exactly the same direction. And the scaling factor, lambda, is called the eigenvalue associated with u. Now, what we saw is that the eigenvectors constitute a basis, okay? Specifically, if we take any symmetric matrix, M, a d by d matrix, then it has d eigenvectors that are orthonormal. So each of them is of unit length, and they're at right angles to each other, their dot products are zero, and we can therefore think of u1 through ud as the axes of an alterative coordinate system.



Spectral decomposition

Fact: Let M be any real symmetric $d \times d$ matrix. Then M has orthonormal eigenvectors $u_1, \ldots, u_d \in \mathbb{R}^d$ and corresponding eigenvalues $\lambda_1, \ldots, \lambda_d$.

Spectral decomposition: Another way to write M:

$$M = \underbrace{\begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ u_1 & u_2 & \cdots & u_d \\ \downarrow & \downarrow & & \downarrow \end{pmatrix}}_{U\text{: columns are eigenvectors}} \underbrace{\begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_d \end{pmatrix}}_{\Lambda\text{: eigenvalues on diagonal}} \underbrace{\begin{pmatrix} \longleftarrow & u_1 & \longrightarrow \\ \longleftarrow & u_2 & \longrightarrow \\ & \vdots & \\ \longleftarrow & u_d & \longrightarrow \end{pmatrix}}_{U^T}$$

Thus $Mx = U \Lambda U^T x$:

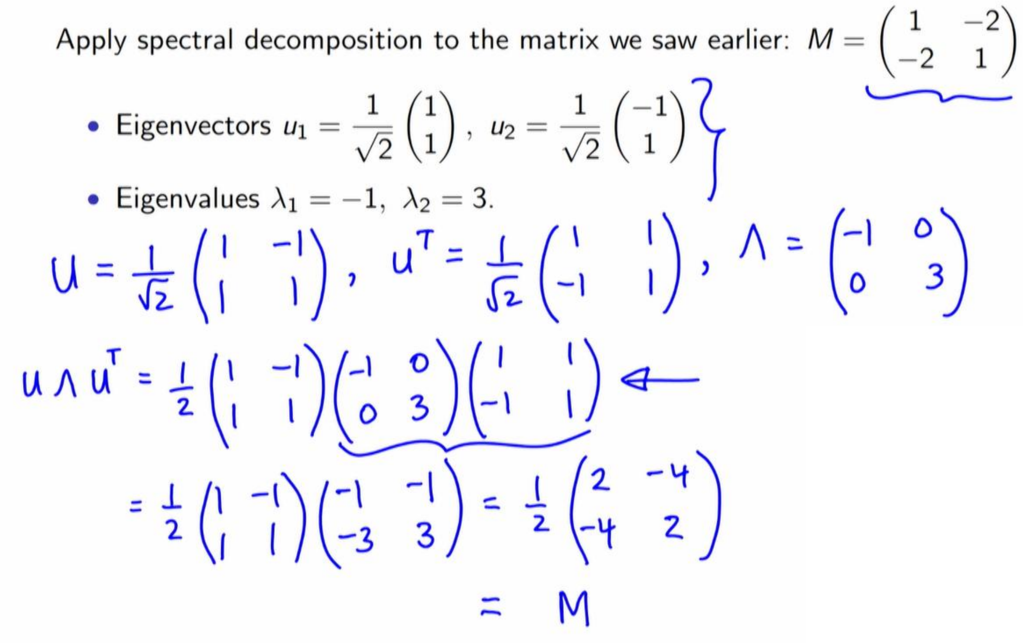
- U^T rewrites x in the $\{u_i\}$ coordinate system
- Λ is a simple coordinate scaling in that basis
- U sends the scaled vector back into the usual coordinate basis

It turns out that a matrix becomes quite simple if rewritten using this coordinate system. And let me explain what this means. We take any symmetric matrix, M. As we've seen, it has these orthonormal eigenvectors, u1 through ud. What we can do is to create a d by d matrix, U, whose columns are the eigenvectors, so the first column is u1, the second column is u2, all the way to ud. And in that case, U transpose looks like this. Its rows are the eigenvectors, so U and U transpose. We can also write down a diagonal matrix, lambda, of all the eigenvalues. If we do that, then it turns out that **M is exactly U times lambda times U transpose**.

- So, first off, this means that M can be recovered exactly from its eigenvectors and eigenvalues. The eigenvectors and eigenvalues contain all the information in M.
- But the second thing, and the more remarkable thing, is that M is much simpler when expressed in this form. So in what sense is it simpler? We started with a d by d matrix, and now we have three d by d matrices. How is that simpler? Well, just look at these matrices. Each of them is very understandable. It's very clear what each of them is doing.
    - U transpose is simply a basis change matrix. It's converting from the regular coordinate basis to the basis of the eigenvectors u1 through ud.
    - U is the basis change in the opposite direction, from the eigenvector basis to the regular coordinate basis.
    - And lambda, well, that's just a diagonal matrix, so it's just scaling each coordinate separately.

M is simply the composition of these three elementary operations. So here is exactly what M is doing to any vector x. It starts by applying U transpose to it. So it takes x, which is a vector in our regular coordinate system, and then it applies U transpose to it, which basically converts it into its own basis, the basis of the eigenvectors. So that's step one, U transpose x. Then, it applies this simple scaling. Now that the point is in its own coordinate system, it simply scales each of the coordinates separately, lambda1 through lambda d, and then finally it takes that answer and converts it back into the basis we understand, the regular coordinate basis. So any matrix can be expressed simply in this way.

总结：



Apply spectral decomposition to the matrix we saw earlier: $M = \begin{pmatrix}1 & -2\\ -2 & 1\end{pmatrix}$

- Eigenvectors $u_1 = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$, $u_2 = \frac{1}{\sqrt{2}}\begin{pmatrix}-1\\1\end{pmatrix}$
- Eigenvalues $\lambda_1 = -1,\ \lambda_2 = 3$.

$$U = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & -1\\ 1 & 1\end{pmatrix},\quad U^T = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ -1 & 1\end{pmatrix},\quad \Lambda = \begin{pmatrix}-1 & 0\\ 0 & 3\end{pmatrix}$$

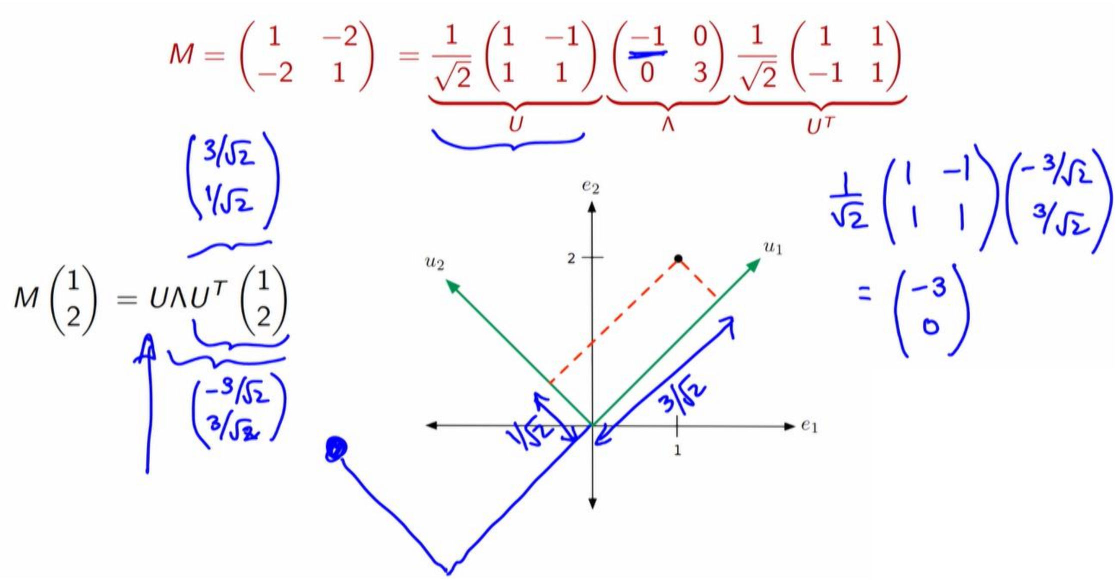
$$U\Lambda U^T = \frac{1}{2}\begin{pmatrix}1 & -1\\ 1 & 1\end{pmatrix}\underbrace{\begin{pmatrix}-1 & 0\\ 0 & 3\end{pmatrix}\begin{pmatrix}1 & 1\\ -1 & 1\end{pmatrix}} \leftarrow$$

$$= \frac{1}{2}\begin{pmatrix}1 & -1\\ 1 & 1\end{pmatrix}\begin{pmatrix}-1 & -1\\ -3 & 3\end{pmatrix} = \frac{1}{2}\begin{pmatrix}2 & -4\\ -4 & 2\end{pmatrix}$$

$$= M$$

Okay, so let's see how the spectral decomposition theorem applies to this matrix, M, that we saw last time, this two by two matrix. So what we did last time is to figure out its eigenvectors and eigenvalues, and this is what they turned out to be. So now let's go ahead and apply this theorem. So we construct the matrix, U, whose columns are the eigenvectors. So let's pull that one over square root two out. The first column is the first eigenvector. The second column is the second eigenvector. Now U transpose, just the transpose of this, so its rows are the eigenvectors. And lambda is just the diagonal matrix of eigenvalues, so it is minus one, three, zero, zero.

Now, let's go ahead and compute the spectral decomposition, U lambda U transpose. So, as usual, we can bring the constants in front. So, we have two square root twos. That just becomes a half. Then our matrices are one, negative one, one, one, negative one, zero, zero, three, and one, negative one, one, one. Let's start by doing these two. What do we get? We have negative one, negative one, negative three, and three. And now let's multiply those two matrices, so we get minus one plus three, two. We get minus one minus three, so minus four. Minus one minus three, minus four. And minus one plus three, two. And so, we indeed get back M. So we had this matrix here, we had this matrix, M, and we were able to rewrite it in this intuitive way, in terms of these three elementary operations, a basis change, a scaling, and moving back to the original basis.



$$M = \begin{pmatrix}1 & -2\\ -2 & 1\end{pmatrix} = \underbrace{\frac{1}{\sqrt{2}}\begin{pmatrix}1 & -1\\ 1 & 1\end{pmatrix}}_{U}\underbrace{\begin{pmatrix}-1 & 0\\ 0 & 3\end{pmatrix}}_{\Lambda}\underbrace{\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ -1 & 1\end{pmatrix}}_{U^T}$$

$$\begin{pmatrix}3/\sqrt{2}\\ 1/\sqrt{2}\end{pmatrix}$$

$$M\begin{pmatrix}1\\2\end{pmatrix} = U\Lambda \underbrace{U^T\begin{pmatrix}1\\2\end{pmatrix}}_{\begin{pmatrix}-3/\sqrt{2}\\ 3/\sqrt{2}\end{pmatrix}}$$

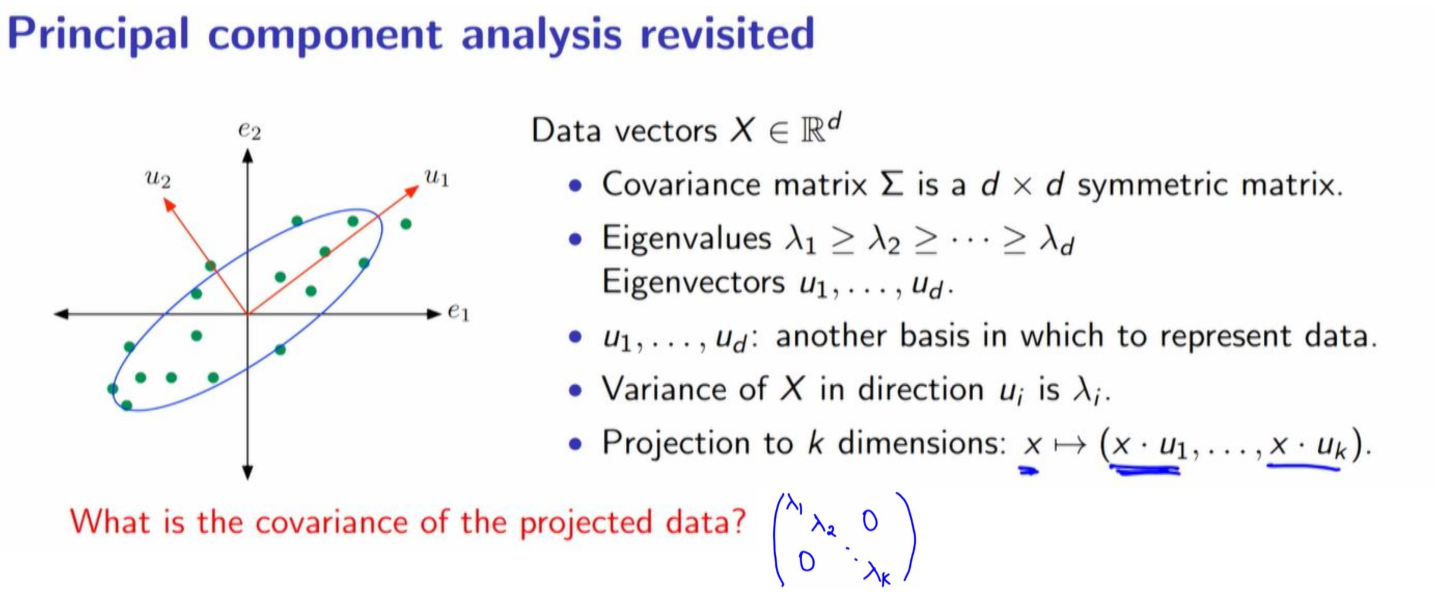
$$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & -1\\ 1 & 1\end{pmatrix}\begin{pmatrix}-3/\sqrt{2}\\ 3/\sqrt{2}\end{pmatrix} = \begin{pmatrix}-3\\ 0\end{pmatrix}$$

So, now let's break the operation of the matrix as a linear function down into these three components, the basis change, the scaling, and then the basis change back. Suppose we are applying the matrix, M, to a vector, one, two. What exactly is going on?

- We can also write M as U lambda U transpose, and the first step is to apply U transpose to this vector, one, two. What that does is to take the vector, one, two, and rewrite it in this alternative eigenvector basis. The eigenvector basis, in this case, has vector u1, which is one, one, and the vector u2, which is negative one, one, all divided by square root two to normalize them to unit length. And what we're gonna do is to take this point, one, two, and rewrite it in this basis. And so these will be the values we get. The first coordinate will be that length, and the second coordinate will be that length. So what are these values? Well, we just need to compute U transpose times one, two. Okay, so let's figure that out. U transpose times one, two is three over square root two and one over square root two. So this distance here is three over square root two, and this distance here is one over square root two. And that is the representation of this point in the new basis. Okay, so we've done this part so far. (后续)

总结：

- Now, we are in the natural basis for the matrix, and in this natural basis, all we need to do is to scale the coordinates separately, and that's what the matrix, lambda, is doing. So what it's saying is take the first coordinate and multiply it by negative one, so we get negative three over square root two. And take the second coordinate and multiply it by three, so we get three over square root two. So in this direction, we go negative three over square root two, and in this direction, we go three over square root two, so we're gonna end up somewhere over there.
- And finally, we return to the original basis by multiplying by U, and let's see what that gives us. So here's U over here. One over square root two times one, negative one, one, one, and then we multiply the vector we have over there, negative three over square root two, three over square root two, and what do we get? Let's see. We get negative three over square root two, negative three over square root two, so that's negative six over square root two, so we get negative three, and so negative three, zero. And that's the answer in our original coordinate axes.



Principal component analysis revisited

Data vectors $X \in \mathbb{R}^d$

- Covariance matrix Σ is a $d \times d$ symmetric matrix.
- Eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$
  Eigenvectors $u_1, \ldots, u_d$.
- $u_1, \ldots, u_d$: another basis in which to represent data.
- Variance of X in direction u_i is λ_i.
- Projection to k dimensions: $\underline{x} \mapsto (\underline{x \cdot u_1}, \ldots, \underline{x \cdot u_k})$.

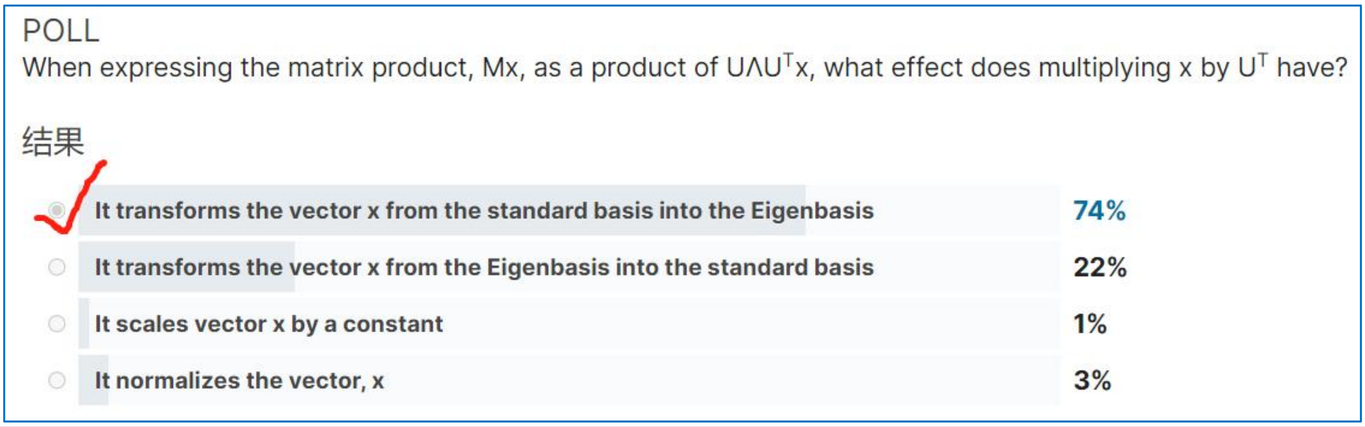
What is the covariance of the projected data? $\begin{pmatrix} \lambda_1 & & & 0 \\ & \lambda_2 & & \\ & & \ddots & \\ 0 & & & \lambda_k \end{pmatrix}$

Now lets see exactly what principle component analysis is doing. So we begin with a bunch of data vectors, X, and our first step is to compute the covariance matrix. The vectors are in d dimensions, and this covariance matrix is a d by d symmetric matrix. And this matrix captures all the correlations between the features. Then we go ahead and find the eigenvectors and eigenvalues. How do we do that? Well, we basically ask the computer to do it for us. There are standard packages that just take the matrix and give us back the eigenvectors and eigenvalues. And since it's a d by d matrix, we get back d eigenvectors, u1 through ud, and the corresponding eigenvalues. Now, these eigenvectors are the axes of the natural coordinate system for this data. They're an alternative basis in which we can represent this data. And in this alternative basis, the coordinates are uncorrelated. Moreover, in terms of what these eigenvalues mean, the variance in the direction u sub i is exactly the eigenvalue, lambda sub i.

Now, what we want to do in principal component analysis is to project down to a certain number of directions, say, k directions, while keeping as much of the variance as possible. So what we do is to sort the eigenvalues into decreasing order, the largest eigenvalue, second largest eigenvalue, and we just keep the top k directions, u1 through uk, and that's our projection. We project to vector, x, in d-dimensional space to x dot u1 all the way to x dot uk, on to the first k of these directions.

what is the covariance of the projected data? Let's see. The first coordinate of the projection is the projection onto u1, and we've seen that the variance in that direction is lambda1. The variance in direction u2 is lambda2. The variance in direction uk is lambda k. And in these directions, these directions, u1, u2, and so on, in these directions, the data is uncorrelated. So the off diagonal entries are zero. And so this is the projection. The projected data has this k by k covariance matrix.

Okay, so that's it for today. We've finally seen the spectral decomposition theorem, which is a very powerful tool of data analysis. It gets used in many different ways, some of them very creative, and what we've tried to do today is to communicate a basic sense of what it's about. See you next time.



POLL

When expressing the matrix product, Mx, as a product of $U\Lambda U^T x$, what effect does multiplying x by U^T have?

结果

[✓ ● It transforms the vector x from the standard basis into the Eigenbasis] 74%

○ It transforms the vector x from the Eigenbasis into the standard basis 22%

○ It scales vector x by a constant 1%

○ It normalizes the vector, x 3%

总结：