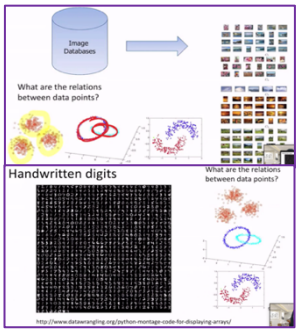
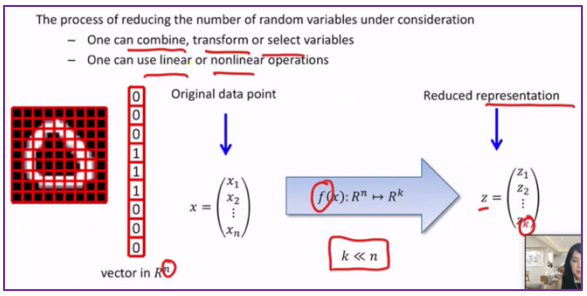
I will talk about dimensionality reduction and principal component analysis, also known as PCA. These are also classified as the unsupervised learning approaches, as we don't have any label information. These methods are used to E4 data and understand patterns.

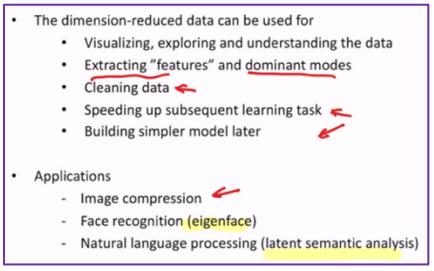
To start with, let's recap what we have talked about so far about **clustering**. The goal of clustering is to identify cluster patterns inside your dataset. For exam-ple, we talk about clustering natural images. You can have a very large image database and have tons of pictures. These pictures are about different contents. And if the picture is about the same content, for example, they're all about a beach, and they will have very similar colors, shapes, and so on, and so, they have high similarity. If you choose the right similarity measure, for example, clean distance, you will be able to see these clustering patterns. How do we identify these clustering pattern using the data using feature vectors only is a question here. That what k-means deals with. We also talk about another type of clustering behavior. This is not based on the Euclidean distance but actually based on the connectivity between the data points, a deforest the data geometry. We talk about spectral clustering, which is a type of method that can discover the non-linear or based on connectivity for clusters. We also demonstrate how to do this for handwritten digits, find out clusters of the handwritten digits. Now we're going to talk about the different problem, which is dimensionality reduction.



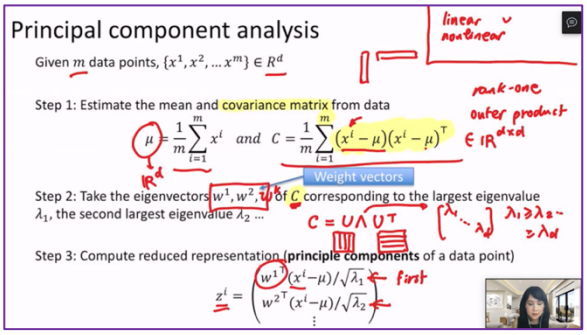
First of all, **what is dimensionality reduction**? Let's look at this amnesty handwritten digits pictures again. As you can see these pictures, even if they're pretty small as a picture, once you vectorize these pictures, which is a typical way we firstly represent these images is to vectorize these little pictures. Even if they're so small, these vectorize the feature vectors are going to be pretty high dimensional, and you can easily have hundreds and even thousands of features for each data points. The original dimension **n** here can be pretty large for your original data. The idea of dimensionality reduction is trying to find a map, which is f, that reduce the dimension of the original data points to a reduced representation and represented by z. The dimension of z is k, and k is typically much smaller than n. So, our goal is to be able to find f that maps this high-dimensional vector into a much lower-dimensional representation. Of course, we want this map to preserve information as much as possible and, or some criteria. What the f does it can either combine, transform, or select the variables out of this long feature vector, and we can use either linear or non-linear operators to f as well. In this and the next lecture, we'll talk about different choices and ways to construct the f, this map to reduce dimensionality and preserves information about the data.



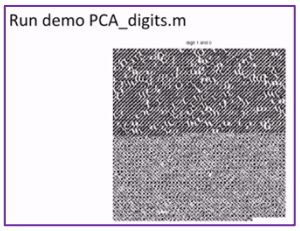
Second, **how do we use dimensionality reduction and why it is useful**? For example, dimensionality reduction can be used for visualizing the data, understanding the data which is quite useful. High-dimensional data are not easy to visualize where we can draw 2-dimensional picture. I can make three-dimensional sculptures, but I can't make the yarn for dimension. So, I could probably make a four-dimensional, like a video of three-dimensional things, but how to visualize even higher-dimensional data is a challenge. If it could project high-dimensional data into lower-dimensional space, then we could visualize it. Another motivation is hopefully we can extract important features and dominant modes from the data but combining features. It is also commonly used as a pre-processing step to clean data, and since we reduce dimensionality, this will also reduce the storage cost and the computational cost or subsequent learning task. If the operation does preserve information needed for subsequent learning task, then we are pretty happy. So, you can compress your data and still do a pretty good job in the subsequent learning tasks. And sometimes this can lead to simpler models as well. **PCA** is used in many places for data compression, data visualization, and, for example, image compression, face recognition. And there's a pretty well-known term called **Eigenface** in face recognition that's basically performing PCA on the data matrix in face recognition. In natural language processing, PCA is also very commonly used, has another name, latent semantic analysis. So, as you can see, PCA is a basic building component for many machine learning tasks, and sometimes they bear different names in different regimes.



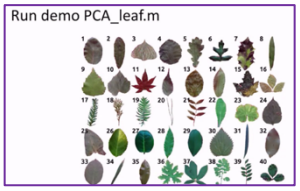
Let's look at the algorithm itself, the **principal component analysis**. I will start to explain why it looks like this and all the mathematics behind it. Assume the usual notation given m data points and each data point is d-dimensional. **First**, we will estimate the mean and covariance matrix of the data. The mean is just the average of the points. This is the first outer Bellman's of data, which is the fundamental quantity and statistics to quantify the data. Next, let me explain the notation a little bit here. The x is a column vector and mu has made from average as a vector. So, x^i - mu is a column vector and x^i - mu^T is going to be a row vector. These two put together, a column vector times a row vector is called the **outer product**. In fact, it is a rank 1 outer product. Essentially this combines n rank 1 outer product based on your data that forms the so-called

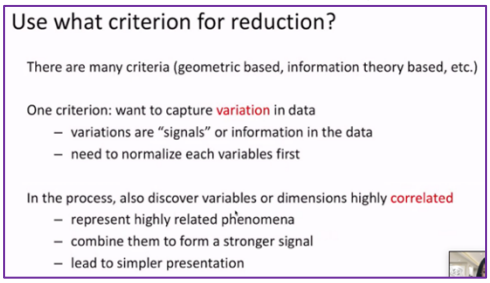


covariance matrix of the data. It basically captures the variability of your data points along different directions, and also captures the correlations, covariance between different coordinates, different features inside this x vector. We'll see this covariance matrix of the data show up quite a few times in our class, and you can see this is one of the fundamental statistics quantities that captures information from the data. Once we have the mean and the covariance of the data calculated, the **next step** we will perform the eigenvalue decomposition of the covariance matrix C, eigendecomposition. We have seen this already in the last lecture, the eigendecomposition of C. And after we have performed the eigendecomposition, then you end up having something like $C = u\lambda u^T$. And u contains all the eigenvectors, and u^T contains all the eigenvector transpose, so each column is a transpose of the corresponding eigenvector. And in the middle, this is a diagonal matrix that contains all the eigenvalues of C. And here, what is the dimension of this covariance matrix C? It should be d-by-d. The C is a d-by-d matrix, and mu is a d-dimensional vector. So, here I have d eigenvalues. Typically, we arrange these d's to be from the largest to the smallest. And so, we will take the eigenvectors w^1, w^2, …, w^k and depending on what the dimension you want to reduce it to is, you can take all the way to w^k. If you want to reduce it to k dimensional representation, you will take top k eigenvectors out of your u. And these correspond to the largest eigenvalues, and the second, and the third, and so, on. And so, once I take these eigenvectors, **step 3,** I'm going to use them as my weight vector w^k to project your data. So, project means I'm going to compute this inner product. So, I'm going to essentially using these w's to combine the original data. So, when you combine it, first you have to subtract the sample mean from each of the sample, and then you will use these weight vectors to correspond to the eigenvectors of the covariance matrix, so one ad then second and so, on. And then once you do this inner product this is going to give you a **number**. And then you will normalize it by divide by the square root of the corresponding eigenvalue. And so, in the end the $\mathbf{z}^i$ is a k dimensional vector. This is the reduced representation of the i's data point. And so, each entry inside this z^i vector will correspond to the projection or the combination of the features using one particular rate. And then we're also going to give some names to the z. . Each of the entries in the **z** are called the **principal components**, and this is the first principal component and the second principal component and so, on. Here's a question for you. We talk about reducing dimensionality using the f, and now we talk about PCA. Do you think PCA correspond to **linear** or nonlinear dimensionality reduction? So, this is a linear dimensionality reduction scheme. Because in your computation for the z, you can see basically you have **performed linear combinations of your f**. So, it's a linear transformation of your x. Fundamentally, this is a linear dimensionality reduction.

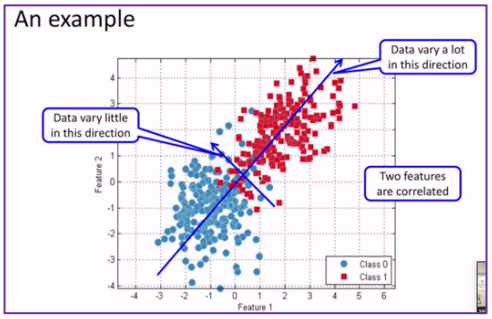


We have introduced the PCA. Then let's first see a demo of our handwritten digits and see what this gives us after you read PCA on these digits. This is the demo for the digits. As usual I'll first load the data, and then I can show you that the place I start to form our PCA is here. First of all, you read the data and once you read the data in here, this is actually an extra step to divide each feature by the standard deviation. This is called the **feature scaling**. It's pretty common in machine learning that you will try to normalize by dividing by the standard deviation of that particular feature so that you don't end up having features with very different ranges of values. Subtract the sample mean from the x, and then this is to find the covariance matrix C. In this case, we're going to find the top 2 eigenvector to project the data to find the top 2 principal components (k=2) of your data points. The eigs in MATLAB actually finds the eigendecomposition, eigs is a faster algorithm that if you only want to find say, a few eigenvectors here, we only need the top 2 eigenvectors. And this gave us the representations. Let me run this part of the code, and it brings up the 0 and the 1 digit. Each little picture represent 1n digits. And we have performed the eigendecomposition of my C. So, let's look at the size first. It's 2. This S is our eigenvalues, Lambda 1 and Lambda 2. And then here we perform the PCA analysis to project each data point. Of course, this is in MATLAB, things are done in matrix vector form pretty compactly, but essentially this project each of the data points using the vector. So, this dim1 and dim2 corresponded to principal components, for the first and the second principal components for each data points. Let's use the principal components to visualize our data. As you can see, these red and blue represent the 2 types of images, the digit 0 and 1, and these are their first 2 principal components. And each dot here represent one data point. So, because you project your original image, the size of the original image is actually 256 after regularization. So, the original size of the picture is 16 by 16. So, we project a 256-dimensional vector representing each image into 2-dimensional data points. So, now I can plot this. Each point here represent one image. So, interestingly, you can see this pattern, seem pretty interesting. Red and blue correspond to the 0 and the 1s. And so, this seems to be separated in the projection space, which is also quite interesting. If I want to perform some classification, I could probably find a decision boundary in this reduced computation as well. And somehow maybe these shapes of the PCA also contain information. That's also why visualizing high dimensional data is quite interesting. Here I like to give another demo for leaf dataset. This dataset consists of samples for different species of leaves. And in this dataset, each sample features are being measured and provided for that particular leaf. There are 14 features together, 8 shape feature and 6 texture features roughly. Let's look at how do we analyze this data. Import the data, as you can see, this also provide some additional information explaining what each feature is about. And we will do the usual PCA analysis. We first scale the features by divide by their standard deviations, then subtract the mean from the data points, find the covariance matrix, perform the PCA to find the top 2 principal components, and then project data. And in addition, we also mark the different species of the leaves. Let's see what pattern can be observed. This is the result. As you can see, after we project the data into two-dimensional space, different colors and patterns represent different leaf species. And so, you can see interestingly that it looks like the principal components have sufficiently captures information about these different species, so they form some clusters in this domain. You see this is one species, and this is another species. And they are pretty close to each other, and they form a cluster in this PCA domain. In this sample, I see the data size is 340 by 16. We have 340 samples out of that, possibly 40 species, and each sample have 16 features.

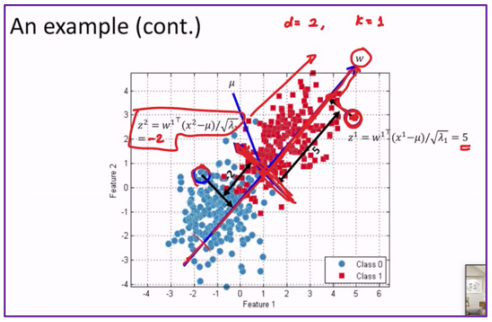




Now we have an algorithm, but the question remains, where do they come from? PCA actually is based on very rigorous derivation solving an optimization problem. The criterion for reducing the dimensionality of the data gives us this optimization problem. When do we reduce the dimensionality of your data, what should be the criteria? For example, there are many different criteria based on preserving the geometry of the data and information content. So, how to define information? There are ways to define through information theory and there's one particular criterion that's pretty popular which is to capture the variation of the data. Because the variation typically contains information or signal that inside the data. Like we mentioned that there's typically pre-processing (normalization). When we do the projection and while keep the variation, we can also possibly discover the variables or dimensions could be correlated. This correlation could represent that these variables could be related to each other. And if we combine the features that seem to be highly related and that can enhance the signal to get a clearer picture. These two give us the motivation to formulate the PCA problem.

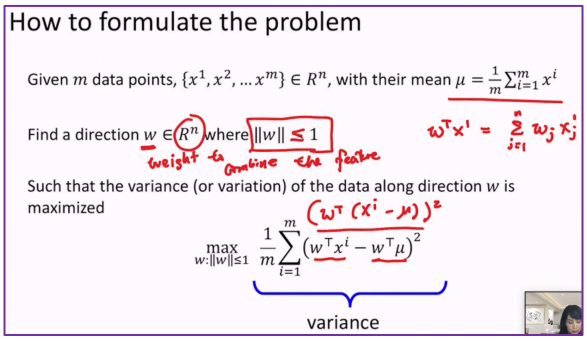


Let's look at the data together a little better picture of what we mean by that. In this **example** we have two classes, one class in red and one class in blue. So, if our goal is to classify these two classes. If you look at the data when plotted in the 2-dimensional space, in this case, the data is 2-dimensional. And you can see along this direction, the data simply have a lot of variability along this direction, the data has very little variability. Moreover, as you can see that because the variability is along this direction, then if I somehow project the data along this direction, along this arrow. So, what do we mean by projection essentially? You find a point on this line that are closest to this data point. So, imagine you're collapsing all the data points onto this arrow line. And so, if I do that, then the blue ones are still going to be in this region and the red ones are more or less in this region. So, I can still be able to separate the two classes after the projection. So, that means long direction is a pretty good direction to project my data that keeps the information and still helps with the classification. And this could be a bad direction to project. What does it mean? If I collapse the data onto short direction, so you can imagine I collapse the data towards this particular direction and then the blue ones are going to spend a whole region here and the red are going to be mixed together with these blue ones, and we're not able to separate them. So, you can see why the direction of projection, the data is quite important in PCA. So, another observation you can have been in this case the two features are quite correlated since it's Feature 1 and Feature 2. And when features are correlated, it means that you can see once one feature gets large, the other feature tends to be large as well. So, the data cloud is oriented in this 45-degree direction. So, that indicated the two features are quite correlated. It means that we can probably combine the two features and preserve information that's exactly what we observe here. If we project the data, combine the feature around this line, then we can preserve most of the information.

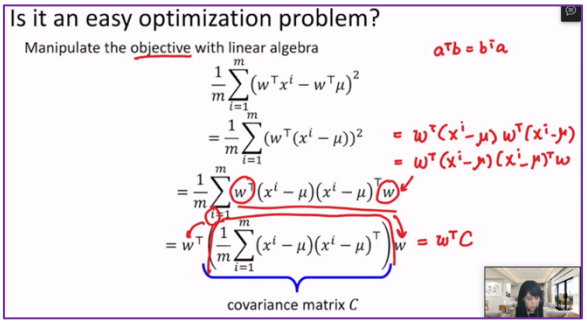


So that's exactly what's going to happen in the PCA. We're going to choose, for example, reducing this case. The original dimension D is 2, and then we reduce it to K =1. And so, which direction should we choose? Essentially, that means we are going to project two-dimensional data on the line and the direction we should choose is going to be along this line here. And so, like we explained before, projection means that I'm going to take one data and then find a point on this line in this direction. There are closest to this particular point. So, of course by simple geometry, this point and this line forms a 90-degree angle, is going to be the point on this line closest to this data point. So, you can imagine after the projection, you're going to have this part pretty much all red and this part pretty much all blue. So, what do we mean by the principal components for these data points? For example, as illustrate here, for these data points and its corresponding score, let's call it z^1 is given by $w^{1T}(x^1 - mu)$. Remember, the sample mean, the mu is the average of all the data points for a data cloud at roughly means the center or centroid of data clouds, so that's in here. So, subtracting the mean and then project it and divide by the square root of the standard deviation. So, in this picture, these λ's actually roughly tells us the spread of the data along these directions. For example, Lambda 1 is roughly going to be the spread of the data along this direction. After we do this, and the z^1, for example, becomes 5. And so, what this really means is roughly it tells you that this data points if you projected in here, and its distance from the center of the data clouds is going to be 5, relatively speaking. And so, if you do it to the other data points, this blue one and after you do the projection, the principal components for this data point is -2. Another thing you observe is one is positive, the other is negative. Indicating that one data point is on one side, the other is on the other side relative to the centroid of the data clouds. Hopefully this example gave us some intuitive understanding what happens in PCA.

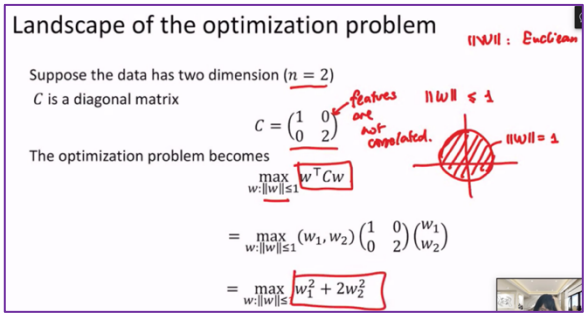
Now let's try to formulate this problem more formally. How do we find the direction to project the data that preserves the variance? In this example, this direction preserves the variance because after you project the data, the red and blue are pretty spread out still if you're projecting along this direction and versus if I project in one direction, they're going to all be collapsed into this region on the line. Let's do it like this. Their variance are going to be smaller. Larger variance is going to be desired. Recall that the data have the sample



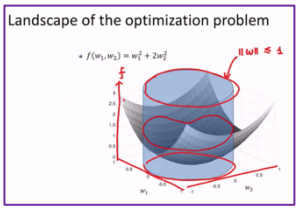
mean and here we will find a direction to project the data. Since the direction to project the data is basically you can also understand it as a weight to combine the features. So, this is used to combine the x. Let me start again. So, for example, $w^T x^1$ means that you're performing this w_j switch on the j's entry; j's feature of this first data point. That's what happens here. Essentially these w's are the weights. And so, to make sure that the problem is meaningful, we will restrict the norm of the w to be less than or equal to 1. And typically, we're going to achieve the equality that the norm, the outer norm of w^1. So, our goal is to find the w, the direction of projection, or the weight vector to maximize the variance roughly preserve information. And what is going to be the variance here? And so, as we explained, the variability can be captured by sum of the squares of the projected data. After you project data and relative to the mean, the centroid. So, this can also be written as $(w^T(x^i - mu))^2$. If I combine the w, geometrically speaking, what this happens is exactly this previous picture here. So, if you look at this data point, I'm interested in finding out after projection how far away it is from the centroid. This captures how much variability this data has after the projection. That's exactly what this expression tells you. I hope you can take 5 seconds to reflect what's happening here from all the intuition, all the way booms end up having this concrete mathematical formulation. This is typically the pattern in machine learning, is you have some math. But from the pretty applied problem to this abstract formulation, is a big leap. Once you have this, the rest of it begin the mathematical problem if you derive the algorithm.



Next, we have this optimization problem to solve. That's it for PCA. We want to find a w to maximize this objective function. Question is, How do we solve it? Let's first manipulate the objective function so that function we're maximizing is called the **objective function**, Bayes linear algebra tricks. And the first step is, like before I combined, pulled out the w, and here I use the trick that I can write this as $w^T(x^i - mu)w^T(x^i - mu)$. So, I have done nothing but just break it down. It's just two scalars. There's the fact that $a^T b = b^T a$. So, I'm going to switch the order, $w^T(x^i - mu)(x^i - mu)^T w$. And so, that's going to be helpful because then I end up having this expression. I have a sum of a bunch of terms look like this. And the sum is, with respect to i, which is my sample index, and w doesn't depend on i. I can put the w out in my expression and bring the sum inside. After this manipulation, what I end up having is I can write my objective function as w transpose this term in the middle is exactly a definition of the colors matrix that we discussed.



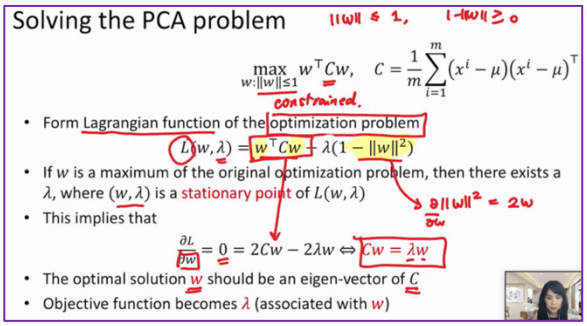
We have shown that this problem becomes the following. We want to find the w with a norm less than equal to 1 such that you maximize this term. And people could ask, why do you restrict the worm to be less than equal to 1? Think about it. You need an upper bound on this w because if you think about, if I don't restrict my w to have an upper bound and I'm trying to maximize this term, then the w is very large. So, in order to make it meaningful, we set an upper bound on the Euclidean norm of w. So, essentially, what does this look like? And this constraint that your w is less than equal to 1, you can imagine this is a ball. This ball correspond to w, the norm is 1. Remember w is the Euclidean norm in distance. And essentially it says, I'm going to consider all the w inside this ball. So, you're saying I'm considering all the double inside this ball and maximizing my objective function. And so, what does this look like? Let's try to make sense of it, to think about a very simple example. Your data have only 2-dimensions and your covariance matrix is diagonal. Again, what does it mean to say your C is diagonal? This means your covariance matrix diagonal; your features are not correlated. So, this is very simple case to be plugging in here, you end up having this expression. This is your objective function, $w_1^2 + 2w_2^2$ subject to the constraints.



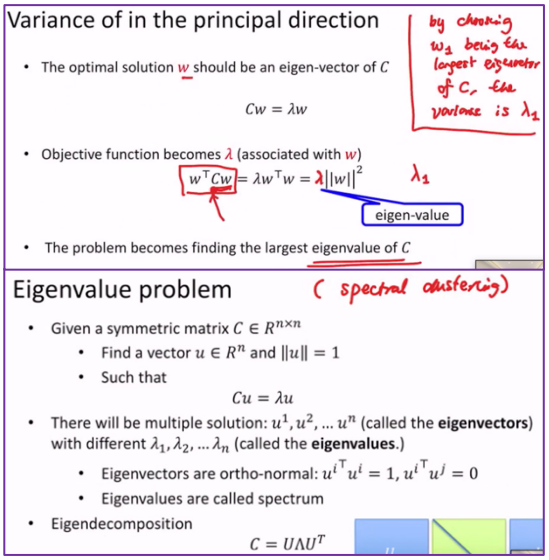
Let's try to plot this. Now, I have a very simple problem I can actually plot it. I'm going to try to plot my objective function; $w_1^2 + 2w_2^2$. So, what does it mean to solve this optimization problem? This is w_1, this is w_2, and this axis is my f. And so, when I plot it this surface here, this is my objective function, and I still have the constraint. Remember that w have to have L2 norm less than equal to 1. And so, essentially, I am trying to find within this cylinder, intersection is my circle is this, and I'm trying to find within this the maximum value of it. So, where does it going to occur? From this plot it looks like it's going to occur somewhere on the boundary.

Let's get back to the original problem of solving the general PCA problem. We have reformulated this problem as the following, which is to maximize this objective function that involves the covariance matrix subject to this constraint. From optimization problem-solving point of view, we are dealing with the so-called a **constrained optimization problem**. The general strategy for solving constraint

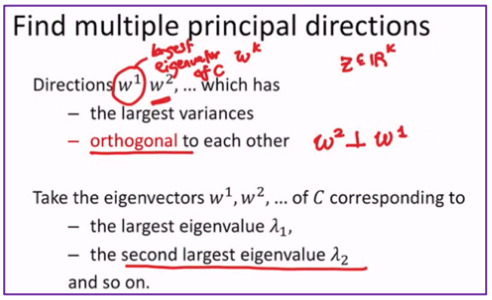
optimization problem involves formula, so-called **Lagrangian function**. Here we will talk about how to solve this specific PCA problem. And for solving the most general optimization problem, we will have a dedicated lecture on this topic later on. Lagrangian function is basically a linear combination, you have objective function, and something consists of your constraints. Remember your constraint is in the norm of w less than or equal to 1, and we can write this as (1 - w norm) ≥ 0. Essentially this λ is the so-called the **Lagrangian multiplier**. We will put this as an optimal variable as well. Roughly speaking, the Lagrangian function measures that if we can allow the constraint to be violated a little bit, the amount of violation is measured by 1 minus the w norm. If we can charge a price, which is this Lagrangian multiplier for the violation, and then what is the corresponding w is that we will achieve by solving this optimization problem. And then maximizing this Lagrangian function with respect to w, and while minimizing it reaches back to the λ (max w, min λ). That's the theory behind it. Basically, we are introducing this wondering function and we'll find a pair of w and Lambda which correspond to the stationary points. So, I mean at min/max problem. So, how do we approach this? We're going to use the usual trick. We're going to take the derivative of the Lagrangian functions just back to w and set it to 0 and using multivariate calculus and this quadratic term, derivative with respect to w is going to be 2Cw. And the term that depends on w is going to be this. So, the derivative of this respect to w is going to be 2w. And so, in the end, this is going to be 2Cw - 2λw. And by setting this to 0, we find a stationary point. That means my w should satisfy this equation, Cw = λw. So, here's a question for you. Do you recognize it? Where have you seen this before?

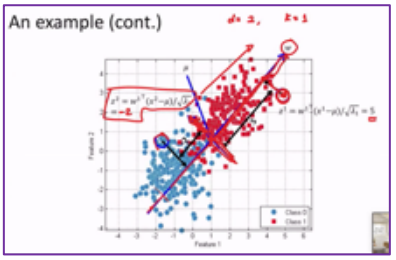


This basically is the definition of eigenvalue and eigenvector pair. Cw is equal to a scalar times w itself. So, w is the eigenvector for C and λ is a corresponding eigenvalue. From this derivation, although it involves some complex optimization theory, but the solution is very intuitive, simple. The optimal solution, the optimal weight back to optimal direction should be the eigenvector of the covariance matrix. And the corresponding objective function associated with this optimal choice of w is λ. That's it. What this tells us is, if you plug in, the optimal vector, optimal solution should be eigenvector of C. And then the objective function for this corresponding w is going to be equal to λ. So, the objective function for this is going to be w^TCw and since Cw = λw, I plug in here, I have $\lambda w^T w$, then λw^2. So, the problem essentially finds the largest eigenvalue of C. If you want to maximize it, I should find the w that gives you the largest λ. The largest Lambda for C is the largest eigenvalue of C. So, after all the mathematical derivation, what does it mean? First of all, this is a recap of what is the eigenvalue problem that we talked about before, talk about this last lecture when we talked about spectral clustering. And as I mentioned, you will see this a few times in our lecture. Pretty useful.

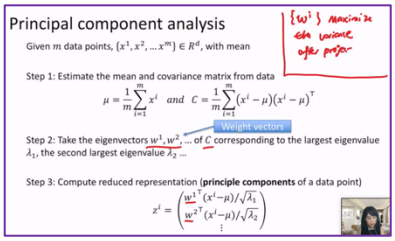


What if you want to find multiple principal directions? We have solved a mathematical problem if you wanted to find one direction to project. If you have one-on-one, how should we choose these directions? That means if you want to project your data and find more than one principal component. Remember, in our formulation, we can't have all the way to w^k. Then your principal component can consist of k components. z can be k-dimensional vector. We have already shown that the directional optimal weight vector and should correspond to the largest eigenvector of your covariance matrix. w^1 is the largest eigenvector of C, then the corresponding variance. After you choose this w from this derivation, the corresponding variance is going to be equal to λ_1. So basically, the message is, by choosing w^1 being the largest eigenvector of C, the variance is λ_1, the largest eigenvalue of your covariance matrix. So now I find the next direction to project my data and to know as much information from data as possible. I shouldn't include any redundancy in my projection, and so, because of that consideration, I will require these directions to be orthogonal to each other. So, w^2 is orthogonal to that w^1. So, if you follow the same derivation we had before, the second principal component will be giving you corresponding variance to correspond to the second largest eigenvalue of your C, and so, on and so, forth. So, this example, if you look at it again, basically, if I project along, this is my w^1. And after I finish this, you say, what's the next direction if I can have another direction to protect my data? And I should choose that to be orthogonal to w^1. So, this is going to be my w^2 but of course, in this problem, the original data is 2-dimensional. And if I project that into two-way vectors, essentially, I get back to my original data, I just change the coordinate system representing my data, so you preserve the entire variance of your data.

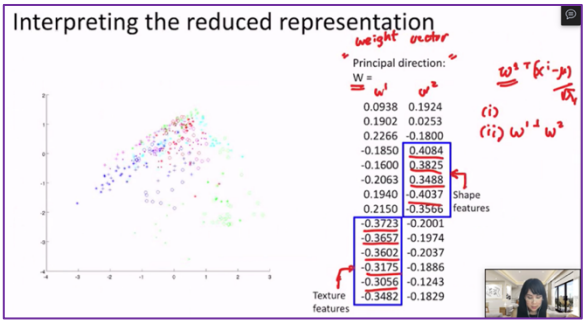




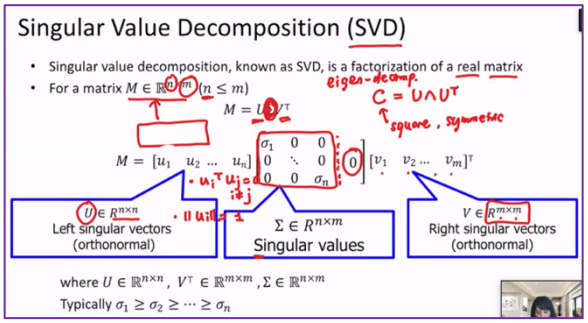
Here's a recap of the principal component analysis. Now hopefully it'll start to make more sense to you. The steps. Why do we form the covariance matrix and perform eigendecomposition of the C to find the eigenvectors? And then compute the reduced representation through the projections using these w's. And most important message here is these w's, these projection directions or weight vectors to combine the features are solved from optimization problem to preserve the variance or maximize the variance after projection. The weight factors themselves actually contain some useful information as well.



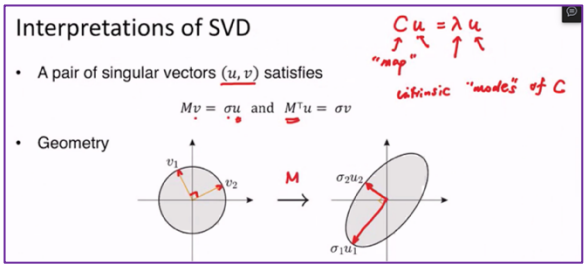
Let's look at this leaf example again. So, if you look at the principal directions, so these weight vectors, w's, also sometimes are referred to as principal directions because these are the directions that gives you the principal components. Remember 'w's are used to combine your features, meaning that we're going to perform w^1, $w^{1T}(x^i - mu) / \sqrt{\lambda_1}$, that's the principal component. So, w's are going to be the weight vector used to combine the features. If we look at this vector, it has some interesting patterns essentially. You can see which part of the w have larger magnitude. These features in the blue box have larger magnitudes than the other one. If we look at what these features are, the features in this box correspond to the texture feature of your leaf. And then the features inside this blue box correspond to the shape features of your leaves. So, interestingly, you can see that from this computation, the PCA has find the optimal way to combine your features so that you can construct different part information using your two components. So, from this interpretation, as you can see, the first principal components, are going to focus more on the texture. And the second principal components is focusing and representing more on the shape features. You can think about why this is the case, for two reasons. First is this weight vector essentially indeed seeming to combine similar features. These features are about texture. These are correlated features, and they are being combined. Second is since we require these directions to be orthogonal to each other so you can see indeed the place, with at the different features in your data. They seem to contain complementary information in the principal component scores.

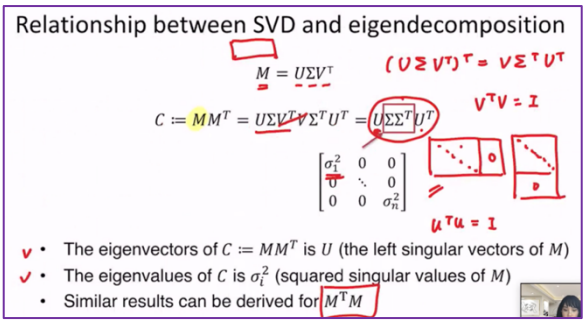


So, we have talked about PCA and the interpretations. The next one I'll talk about is a related decomposition. So, far we talk about PCAs performed based on eigendecomposition. And there is another way to implement PCA based on so-called **singular value decomposition (SVD)**. It is a linear algebra technique to factorize a matrix. Typically, this matrix is real-valued. Remember the eigendecomposition, let me write it here, $C = U\lambda U^T$. We're going to require C to be a square and symmetrical matrix. But for SVD, this can work for rectangular matrices, so it's more general in that sense. Here we're going to assume **n** is less than or equal to **m**, the other case around can be similarly defined. **M** looks like a short fat matrix, and the SVD composition is going to be the product of three matrices, $U\Sigma V^T$. U is a square matrix of n-by-n dimension, and each column of the U is so-called a **singular vector**, the whole measure is called the **left singular vectors**. And the left singular vectors satisfy the requirement that they are orthogonal, meaning $u_i^T u_j = 0$ when i doesn't equal to j. Moreover, they are having unit norm, they're all normalized, so the L2 norm of this vector = 1. Similarly, the other factor on the right, the transpose consist of this vector and the size of the V is m-by-m, a square matrix. So, you can see this time the U and V, the left and right singular vector matrices can be different sizes because your M can be a rectangular matrix. Let's look at the matrix in the middle, it is almost like a diagonal matrix except it's not exactly because this sigma matrix can be not a square matrix, but you can write it as a block matrix, in this case, since n is less than m, the left block is a diagonal matrix, and the right consists of zeros. This sigma consists of so-called the **singular values**, similar to eigendecomposition, we will typically re-arrange these sigma values to be from the largest to smallest.

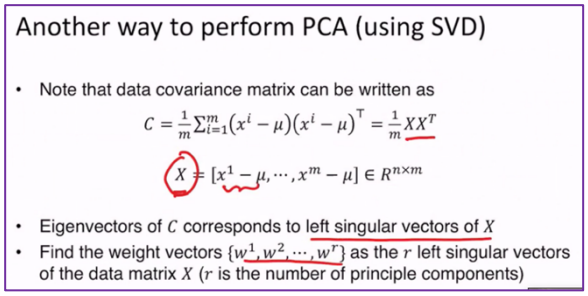


So, how do we understand the SVD? Let's draw an analogy to eigen-decomposition, remembering eigendecomposition, our eigenvalue eigenvector pair have to satisfy $Cu = \lambda U$. What this means is essentially that remember multiplying a matrix can be interpreted as a linear map, if you map your eigenvector, the result is going to be the same eigenvector, but it's going to be stretched to a magnitude equal to the eigenvalue. This use can be interpreted as these intrinsic modes of your matrix, sometimes called of your **linear map C**. In SVD, this concept is generalized. You consider a pair of singular vectors associated with matrix M, $Mv = \sigma u$, and the result can be different from v, but another vector u. Similarly, you can define this for M^T. So, geometrically, what this means is, suppose you have v_1 and v_2 two orthogonal singular vectors and after the map through M, you use M(v_1 and v_2). These are the results, v_1 is mapped into $\sigma_1 u_1$ and v_2 is mapped to $\sigma_2 u_2$, so you can have **rotation** happening and then a **scratch** into an ellipse, and you can see the SVD is the general addition concepts of the decomposition.

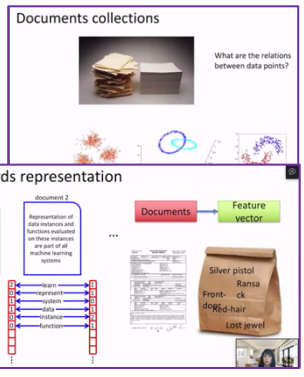




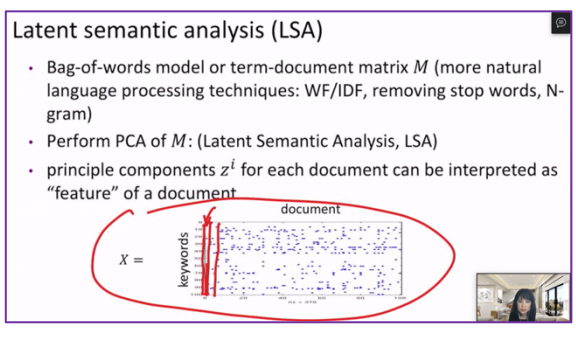
Natural question to ask is how are these two related? We can derive it from here. If you look at the M, and in this case, our M is a short fat matrix, and the SVD decomposition of M is given by $U\Sigma V^T$ and consider the C which is MM^T. Let's see what a C is going to look like. Let's plug in the decomposition for M here, $U\Sigma V^T$ and then M^T. It's going to be $U\Sigma V^T\Sigma$, this is $V\Sigma^T$, U^T. So, when you apply the transpose inside a bunch of matrices multiplying together, you have to shuffle their terms. V is an orthonormal matrix, that means V^TV is equal to Identity, so this term is gone. You end up having $U\Sigma\Sigma^TU^T$. What is Σ? Σ consists of a diagonal matrix and a 0, Σ^T look like the following. So, $\Sigma\Sigma^T$ is going to be a diagonal matrix, and you can see that the 0's are going to knock out each other and so, you're going to have the σ_i^2 on the diagonal and all the other entries are 0. From this derivation what we have shown here is that if you look at this expression, this corresponded to an eigendempensation because U is also an orthonormal matrix satisfy $U^TU = I$ (identity matrix). So, U correspond to the eigenvectors of this particular matrix C and σ_i^2 correspond to the eigenvalues of C. So, similar results can be derived otherwise for M^TM.



This is interesting and how do we use it? Based on this connection, now we find another way to perform PCA. Instead of using eigendecomposition, we can use SVD. First of all, we have to perform an eigendecomposition of the covariance matrix C. The covariance matrix C looks like the following. So, we notice that we can write this more compactly as a matrix product if I define my X as the following. X is a data matrix where each column is one data point, subtract the mean centered data points. If you form this matrix, you can check using linear algebra that XX^T is giving you this exact same expression here. You can see if I have this observation, this is great because in that expression, M correspond to a scaled version of X. If I want to find the eigendecomposition of C, then I can instead find singular valued SVD decomposition. So, how do we use this finding to find another way to perform the principal component analysis? In PCA, we want to find the eigendecomposition of C. The first thing we noticed that is we can write the data covariance matrix as this following form. If it defined a centered data matrix X as the following, so each column of this X is the data point, subtract the mean. Oonce you do this, you realize that my data covariance matrix can be written as the XX^T, and this is almost the same thing as the previous page. M is a scaled version of X. So, that means I can find the eigenvectors of C by finding the left singular vectors of X, and the corresponding weight vectors can be extracted using the singular vectors of X. And you ask why this is preferred sometimes because it saves some computation. Instead of having to form the C and then do eigendecomposition, now, you can directly perform SVD on X, so that gives some speedup and states computation. That finishes all the mathematical descriptions of PCA.

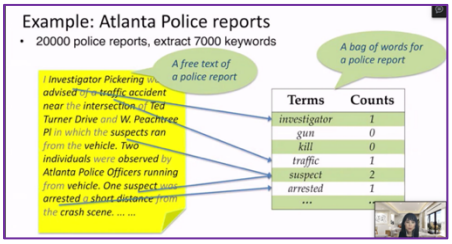


Now we'll focus on one application of PCA, which is a document collection analysis. We will try to find cluster and patterns for the documents, and this is an important application for natural language processing. In this example, each document can be viewed as one data point and the fissures are extracted from the written content of the documents. In national reprocessing, the most common representations of documents, carter so-called the **bag of words model**. This is the type of model where you can extract the features based on the frequency of the keywords. For example, you can predefine some keywords as blurry represents this data instance function, and so, on, and you can count how many times this keyword has appeared in each of the documents. As you can see, the documents with different focus and different content may have a different emphasis in terms of these keywords. So, by counting how many times does keywords appear in your document, you can find some information like about the topics where what document it is. Map the documents into a feature vector consists of the frequency of the keywords is called the bag of words representation.

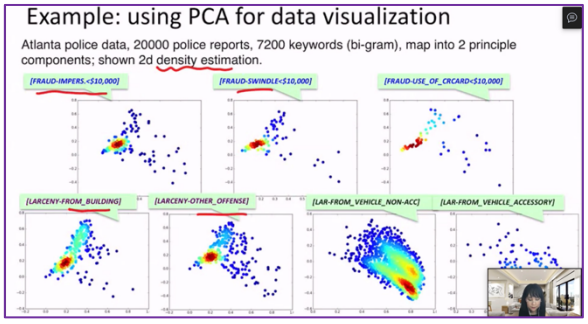


Latent Semantic Analysis (LSA): After you do this, above two, you end up having this M. I'm going to illustrate it here. So, each column corresponds to one document and each row correspond to the current for 8 keywords. You can imagine you have this matrix. That's one data points here. In natural language processing, there are other things to do to extract this X more meaningfully. For example, you will use the word frequency divided by inverse document frequency basically are words that appears everywhere in all the documents. These are having pretty high document frequency that was probably not very informative so, by counting how many times they appeared in all the documents and that's called the **document frequency**. Divide the word frequency you have count by the document frequency, you can reduce the impact imposed on the feature vector from these high frequency documents appear everywhere because they're so common. For example, stop words can even remove them z and so, on. And you can also consider, that is a single keyword, or phrases. Phrases consists of two words or three words

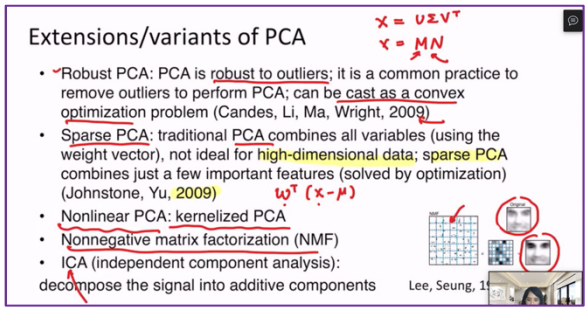
that are called **N-grams**. So, bottom line is you have these ways to extract a feature vector. And then the feature vector helps us to define the documents in here. So, in natural language processing, when interest is to find out the patterns in the documents and in particular the similarity in their classroom patterns. And so, we will combine the features defined by these keywords or transform the keywords counts. And so, that correspond to perform a principal component analysis on this X matrix formed based on the bag-of-words model.



Here's one example. This is a project actually I did with the Atlanta Police Department. And they are interested in finding out the patterns in the crime. Of course, this information is most directly recorded in a police report. And for instance, we take 20,000 of the police report in their database. And each police report is one document. And we can extract some important keywords and then start to convert these police reports into bag-of-words model. Each police report is defined as a feature vector.



Once we have that, we will perform PCA, and find two top principal components of these police reports, and then we can visualize it. We visualize this for different types of crime incidences. And the interesting thing is after this realization can see the different types of crime instance have very different shapes in PCA. And this signature of the different patterns of crimes that's quite interesting. And this is I think fraud type with money less than 10,000 and fraud swindle. This is a different license from building from offense. They have different distributions of the patterns and so, on. And so, in fact when we visualize the PCA principal components of the documents, we also combine it with a density estimation, which is a topic we will discuss later on in this class.



There are different extensions and variance of PCA. Remember we talked about early on that PCA is a linear dimensionality reduction method. So, there are also non-linear PCA called the **kernelized PCA** to go beyond linear dimensionality reduction. We will briefly talk about this when we talk about kernelized methods later on in this class at the general type of method based on kernels. General principle to make linear method non-linear is to introduce kernels. Besides that, there's also robust PCA. PCA actually is known to be robust outliers. You're throwing a few outliers in your data. It's going to disrupt your results. In practice I'm sure as data scientists I don't know it's important to remove the outliers from your data before you perform PCA to get the correct result. In fact, this can also be formulated as a convex optimization problem. And so, for example, this is a paper that studied the robust PCA in a principled way and say how to solve this optimal outlier removal and then still find the PCA optimization problem. There's also sparse PCA. As the traditional PCA we talk about here, you find the weight vector that combines your data this way, this means that you have to consider all the features of your acts. This could be a problem if you have lots of features, especially for high dimensional data where you have a lot of features, maybe some features are not important. You don't even need to consider that. Sparse PCA is designed for these purposes. It helps you to combine just a few important features. These important features are discovered and find from data by summing mentioned problem formulated based on that. And so, here's a paper in 2009 developer related methods and both robust and sparse PCA are pretty popular nowadays and common practice. Besides that, there's also nonnegative matrix factorization (NMF). Basically, in PCA, remember we have our data matrix and then we essentially perform the singular value decomposition of the matrix. In nonnegative matrix factorization, you have a matrix. You would like to factorize the matrix in a different way by requiring the entries in the matrix factor, each entry have to be positive. The application is natural image analysis. The interpretation here is that you have an image and then you want to be able to understand what are the patterns that have appeared in image that compose your image in the end. You want to learn a dictionary that represent different patterns in your image. Meanwhile, you also want to find out the coefficients for each of the patterns imagining, put these patterns together and that will form in a person's face. So, performing this decomposition will help you to identify the templates or the patterns in your image, and then the coefficients help you to combine the features to form an image. And this is useful for understanding image analysis, image denoising, image imputation for missing data, and so on. Besides PCA the goal is to identify the principal components. These components explains most of the variance in your data meaning that you find the direction that captures or maximize the variance after projection. There is a related work with similar goal called independent component analysis, ICA. But the way they decompose the data is different. In ICA, the goal is to decompose the signal into additive components, not necessarily orthogonal or another way of performing data decomposition. You can think about PCA as a way to decompose your data matrix into orthogonal components. And you represent the data using their magnitude along each of the orthogonal directions versus ICA it's another way to decompose your data.

With that we have finished the today's lecture. We have talked about PCA, which is a type of linear dimensionality reduction techniques. And we show how this can be solved using an optimization problem, which in the end becomes solving an identity decomposition of the covariance matrix or SVD of your data matrix to approach it. And we show how this can be used for dimensionality reduction and also visualize your data to find out the patterns. That's it for today. Thank you.