**LORENZO ROSASCO:** If you remember, we did the local method bias-variance. Then we passed to global regularization methods-- least squares, linear least squares, kernel least squares, computations modeling. And that's where we were at. And then we moved on and started to think about more intractable model, and we were starting to think of the problem of variable selection, OK?

And the way we posed it is that, yeah, that you are going to consider a linear model. And I use the weights associated to each variable as the strength of the corresponding variable that you can view as measurements. And your game is not only to build good predictional measurements, but also to tell which measurements are interesting, OK?

And so here, the term "relevant variable" is going to be related to the productivity, the contribution to the productivity of the corresponding function, OK? So that's how we measure relevance of a variable.

So we looked at the funny name. And then we kind of agreed that somewhat it seems that there is a default approach, which is basically based on trying all possible subsets, OK? So this is also called best subset selection. Variable selection is also sometimes called best subset selection.

And this gives you a feeling that what you should do is try all possible subsets and check the one which is best with respect to your data, which, again, would be like a trade-off between how well you fit the data and how many variables you have, OK?

And what I told you last was you could actually see that this trying all possible subsets is related to a form of regularization, where it looks very similar to the one we saw until a minute ago. The main difference here, I put fw, but fw is just the usual linear function.

The only difference is that here, rather than the square norm, we put this functional that is called the 0 norm, which if, as a functional, that given a vector returns the number of entries in

the vector which are different from 0, OK? It turns out that if you were to minimize this, you are solving the best subset selection problem.

Issue here is that-- another manifestation of the complexity of the problem is the fact that this functional here is non-convex. So there is no polynomial algorithm to actually find a solution.

Come to my mind that somebody made the comment during the break. Notice that here I'm a bit passing quickly over a refinement of the question of best subset selection, which is related to, is there a unique subset which is good? Is there more than one? And if there's more than one, which one should I pick, OK?

In practice, these questions arise immediately because if you have two measurements that are very correlated, or even more, if they're perfectly correlated, if you just build the measurements, you might build out of two measurements, a third measurement, which is completely just a linear combination of the first two.

So at that point, what would you want to do? Do you want to keep the minimum number of variables, the biggest possible number of variables? And you have to decide, because all these variables are, to some extent, completely dependent, OK?

So for now, we just keep to the case where we don't really worry about this, OK? We just say, among the good ones, we want to pick one. A harder question would be pick all of them or pick one of them. And if you wanted to pick one of them, you have to tell me which one you want, according to which criterion, OK?

So the problem we're concerned with now is one of, OK, now that we know that we might want to do this, how can you do it in an approximate way that will be good enough, and what does it mean, good enough?

So the simplest way, again, we can try to think about it together, is kind of a greedy version of the brute force approach. So the brute force approach was start from all single variables, then all couples, then all triplets, and blah, blah, blah, blah, OK? And this doesn't work computationally.

So just to wake up, I don't know, how could you twist this approach to make it approximate, but computationally feasible? Let's keep the same spirit. So let's start from few, and then let's try to add more.

So the general idea is I pick one. And once I pick one, I pick another one, just keeping the one I already picked. And then I pick another one, and then I pick another one, and then I pick another one.

So this, of course, will not be the exhaustive search I've done before. It's probably doable. There is a bunch of different ways you can do it. And you can hopefully-- you can hope that under some condition you might be able to prove that it's not too far away from the brute force approach, at least under some condition.

And this is kind of what we would want to do, OK? So we will have a notion of residual. At the first iteration, the residual will be just the output. So just think of the first iteration. You get the output vectors, and you want to explain it. You want to predict it well, OK?

So what you do is that you first check the one variable that gives you the best prediction of this guy, and then you compute the prediction. Then what you want to do at the next round, you want to discount what you have already explained. And what you do is that basically you take the actual output minus your prediction, and you find the residual. And then you try to explain that. That's what's left to explain, OK?

So now you check for the variables that best explain this remaining bit. Then you add this variable to the one you already have, and you have a new notion of residual, which is what you explained in the first round, what you added in explanation of the second round.

And then there's still something left, and you keep on going. If you let this thing go for enough time, you will have the least squares solution. At the end of the day, you will have explained everything.

And each round, notice that you might or not decide to put the variables back in, OK? So you might have that at each step you have just one variable, or you might have that you take multiple steps, but you have fewer variables than number of steps. No matter what, the number of steps would be related to the number of variables that are active in your model, OK?

Does it makes sense? This is the wordy version, but now we go in details, OK? But this is roughly the speaking. So first round, you try to explain something, then you see what's left to explain. And you keep the variables that will explain the rest, and then you need to write.

I'm not sure I used the word, but it's important. The key word here is "sparsity," OK? The fact

that I'm assuming my model to depend on just a few vectors-- sorry a few entries. So it's a vector with many zero entries. Sparsity is the key word to explain this property, which is a property of the problem. And so I build algorithm that will try to find sparse solution explaining my data, and this is one way.

So let's look at this list. So you define the notion of residual as this thing that you want to try to explain, and the first round will be the output. The second round will be what's left to explain after your prediction.

You have a coefficient vector, OK, because we're building a linear function. And then you have an index set, which is the set of variables which are important at that stage. So these are the three objects that you have to initialize. So at the first round, the coefficient vector is going to be 0. The index set is going to be empty. And the residual at the first round is just going to be the output, the output vector.

Then you find the best single variable, and then you update the index set. So you add that variable to the index set. To include such variables, you compute the coefficient vector. And then you update the residual, and then you start again, OK?

If you want, here I show you the first exam-- just to give you an idea. Suppose that this is-- so first of all, notice this, OK? Oh, it's so boring. Forget about anything that's written here. Just look at this matrix. The output vector is of the same length of the column of the matrix, right? So each column of the matrix will be related to one variable.

So what you're going to do is that you're going to try to see which of these best explain my output vector, and then you're going to try to define the residual and keep on going, OK?

So in this case, for example, you can ask, which of the two directions, X1 and X2, best explain the vector Y, OK? So this is the case where it's simple. I basically have this one direction. One variable is this one. Another variable is this one. And then I have that vector. I want to know which direction I should pick to best explain my Y. Which one do you think I should pick?

**AUDIENCE:** X1.

**LORENZO ROSASCO:** I should pick X1? OK. This projection here will be the weight I have to put to X1 to get a good prediction. And then what's the residual? Well, I have to take this X1. I have to-- I have to take Yn. I have to subtract that, and this is what I have left. So this is the simple terms, OK? So

that's what we want to do.

So we said it with hands. We say it with words. Here is, more or less, the pseudocode, OK? It's a bit boring to read. You can see that it's four lines of code anyway. Now that we've said it 15 times, probably it won't be that hard to read because what you see is that you have a notion of residual. You have the coefficient vector, and you have the index set. This is empty. This is all 0's. And this is the first round. It's just the output.

Then what you do is that you start. And the free parameter here is T, the number of iterations, OK? It's going to be lambda, so to say. What you do is-- OK, you have for each j-- so just notation. j runs over the variables, and capital Xj will be the column of the data matrix, the one that corresponds to j's variable, OK?

And then what you do in this line, here I expand it, is to find the coefficient-- sorry, find the error that corresponds to the best variable, OK? If you look, it turns out that it is-- if you assume-- oh, it is equivalent to find the column best correlated with the output is equivalent to find the column that best explains the output, or the residual.

These two things are the same. So here, I write equivalence. So pick the one that you prefer, OK? Either you say I find the column that is best correlated with the output or the residual, or you find the column that best explains the residual in the sense of least squares, OK? These two things are equivalent. Pick the one that you like. And that's the content of this one.

And then you select the index of that column. So you solve this problem for each column. It's an easy problem. It's a one-dimensional problem. And then you pick the one column that you like the most, which is the one that gives you the best correlation, a.k.a. least square error.

Then you add this k to the index set. And then, in this case, it's very simple. I'm just going to-- I'm not going to recompute anything. So what I do is that suppose that at-- you remember the coefficient vector, where it was all 0's, OK? Then at the first round, I compute one number, the solution with, say, the first coordinate, for example. And then I add a number in that entry, OK?

So this is the orthonormal basis, OK? So it has just all 0's, but 1 in the position k. So here I put this number. This is just a typo. And then what you do is that you sum them up, OK? So you have all 0's. Just one number here, the first iteration, then the other one. And then you add this one there, and you keep on going, OK? This is the simplest possible version.

And once you have this, now you have this vector. This is a long vector. You multiply this--

sorry, this should be Xn. Maybe we should take note of the typos because I'm never going to remember all of them.

And then what you do is that you just discount what you explained so far to the solution. So you already explained some part of the residual. Now you discount this new part, and you define the new residual, and then you go back.

This method is-- so greedy approaches is one name, as it often happens in machine learning and statistics and other fields, things get reinvented constantly, a bit because people just come to them from a different perspective, a bit because people just decide studying and reading is not priority sometimes.

And so this one algorithm is often called greedy. It's one example of greedy approaches. It's sometimes called matching pursuit. It's very much related to so-called forward stagewise regression. That's how it's called in statistics. And well, it has a bunch of other names.

Now, this one version is just the basic version. It's the simplex-- it's the simplest version. This step typically remains. These two steps can be changed slightly, OK? For example, can you think of another way of doing this? Let me just give you a hint.

In this case, what you do is that you select a variable. You compute the coefficient. Then you select another variable. You compute the coefficient for the second variable, but you keep the coefficient you already computed for the first variable. They never knew that you took another one because you didn't take it yet.

So from this comment, do you see how you could change this method to somewhat fix this aspect? Do you see what I'm saying? I would like to change this one line where I compute the coefficient and this one line even, perhaps, where I compute the residual to account for the fact that this method basically never updated the weights that it computed before.

You only add a new one. And this seems potentially not a good idea, because when you have two variables, it's better to compute the solution with both of them. So what could you do?

**AUDIENCE:** [INAUDIBLE]

**LORENZO ROSASCO:** Right. So what you could do is essentially what is called orthogonal matching pursuit. So you would take this set, and now you would solve a least square problem with the variables that

are in the index set up to that point, all of them. You recompute everything.

And now you have to solve not a one-dimensional problem, but n times k-dimensional problem, where the k is the-- I don't know, k is a bad name-- with the dimension of the set that could be T or more than T, OK?

And then at that point, you also want to redefine this, because you're not discounting anymore what you already explained, but each time you're recomputing everything. So you just want to do Yn minus the prediction, OK?

So this algorithm is the one that actually has better properties. It works better. You pay the price, because each time you have to recompute the least squares solution. And when you have more than one variable inside, then the problems become big and big.

So if you stop after a few iterations, it's great. But if you start to have many iterations each time, you have to solve a linear system. So the complexity is much higher. This one here, as you can imagine, is super fast.

So that's it. So it turns out that this method is, as I told you, is called matching pursuit, or if not matching pursuit, forward stagewise regression, is one way to approximate a zero solution. And one can prove exactly in which sense you can approximate it, OK? So I think this is the one that we might give you this afternoon, right?

**AUDIENCE:** Orthogonal.

**LORENZO ROSASCO:** Oh, yeah, the orthogonal version, the nicer version. The other way of doing this is the one that basically says, look, here what you're doing is that you're just counting the number of entries different from 0's. If you now were to replace this with something that does a bit more-- what it does is it not only counts, but it actually sums up the weights.

So if you want, in one case, you just check. If a weight is bigger than 0, you count it 1. Otherwise, you count it 0. Here you actually take the absolute value. So instead of summing up binary values, you sum up real numbers, OK? This is what is called the L1 norm. So each weight doesn't count for its sign, but it actually counts for each absolute value.

So it turns out that this one term, which you can imagine-- the absolute value looks like this, right, and now you're just summing them up. So that thing is actually convex. So it turns out that you're summing up two convex terms, and the overall functional is convex.

And if you want, you can think of this a bit as a relaxation of the zero norm. As we say, relaxation in this sense is the strict requirement. So I talked about relaxation before when I said instead of binary values, it takes real values, and you optimize over the reals instead of the binary values.

Here is kind of the same thing. Instead of restricting yourself to this functional, which is binary-valued, now you allow yourself to relax and get real numbers. And what you gain is that this algorithm, the corresponding optimization problem is convex, and you can try to solve it.

It is not still something that we can do-- we cannot do what we did before. We cannot just take derivatives and set them equal to 0 because this term is not smooth. The absolute value looks like this, which means that here, around the kink, is not differentiable.

But we can still use convex analysis to try to get the solution, and actually the solution doesn't look too complicated. Getting there requires a bit of convex analysis, but there are techniques. And the ones that are trendy these days are called forward-backward splitting or proximal method to solve this problem.

And apparently, I'm not even going to show them to you because you don't even see them. But essentially, it's not too complicated. Just to tell you in one word what they do is that they do the gradient descent of the first term, and then at each step of the gradient they threshold.

So they take a step of the gradient, get a vector, look inside the vector. If something is smaller than a threshold that depends on lambda, I set it equal to 0. Otherwise, I let it go, OK? So I didn't put it, I don't know why, because it's really a one-line algorithm. It's a bit harder to derive, but it's very simple to check.

So let's talk one second about this picture, and then let me tell you about what I'm not telling you. So hiding behind everything I said so far, there is a linear system, right? There is a linear system that is n by p or d or whatever you want to call it, with the number of p of variables.

And our game so far has always been, look, we have a linear system that can be not invertible, or even if it is, it might have bad condition number, and I want to try to find a way to stabilize the problem.

The first round, we basically replace the inverse with an approximate inverse. That's the classical way of doing it. Here, we're making another assumption. We're basically saying, look,

this vector, it does look very long, so that this problem seems ill-posed.

But in fact, if only a few entries were different from 0 and if you were able to tell me which one they are, you can go in, delete all these entries, delete all the corresponding columns, and then you will have a matrix. Now he looks short and large. He will look skinny and tall, OK? And that probably would be easier to solve. It would be the case where the problem is one of the linear systema that we know how to solve.

So what we described so far is a way to find solution of linear systems that are-- with the number of equations which is smaller than the number of unknowns, and, by definition, cannot be solved, under the extra assumption that, in fact, there are fewer unknowns than what it looks like. It's just that I'm not telling you which one they are, OK?

You see, if I could tell you, you would just get back to a very easy problem, where the number of unknowns is much smaller, OK? So this is a mathematical effect, OK? And the odds are open, because-- now they're not because people have been talking about this stuff for 10 years constantly.

But one question is, how much does this assumption buy you? For example, could you prove that in certain situations, even if you don't know the entry of this, you could actually solve this problem exactly? So if I give them to you, you can do it, right? But is there a way to try to guess them in some way so that you can do almost as good or with high probability as well as if I tell them in advance?

And it turned out that the answer is yes, OK? And the answer is basically that if the number of entries that are different from 0 is small enough and the columns corresponding to those variables are not too correlated, are not too collinear, so they're distinguishable enough that when you perturb the problem a little bit nothing changes, then you can solve the problem exactly, OK?

So this, on the one hand, is exactly the kind of theory that tells you why using greedy methods and convex relaxation will give you a good approximation to L0, because that's basically what this story tells you.

People have been using this-- and so this is interesting for us-- people have been using this observation in a slightly different context, which is the context where-- you see, for us, Y and X, we don't choose. We get. And whatever they are, they are. And if it's correlate-- if the

columns are nice, nice. But if they're not nice, sorry, you have to live with it, OK?

But there are settings where you can think of the following. Suppose that you have a signal, and you want to be able to reconstruct it. So the classical Shannon sampling theorem results basically tell you that, I don't know, if you have something which has been limited, you have to sample twice the maximum frequency.

But this is kind of worst case because it's assuming that all the bands, all the frequencies, are full. Suppose that now we play-- it's an analogy, OK? But I tell you, oh, look, it's true, the maximum frequency of this. But there's only another frequency, this one. Do you really need to sample that much, or you can do much less?

And so it turns out that basically the story here, as you're answering this question, it turns out that, yes, you can do much less. Ideally, what you would like to say, well, instead of being twice the maximum frequency, if I have just four frequencies different from 0, I'd have to do eight samples, OK?

That would be ideal, but you would have to know which one they are. You don't, so you pay a price, but it's just logarithmic. So you basically say that you can almost-- you have a new sampling theorem that tells you that you don't need to sample that much.

You don't need to sample that low either, which would be, say, the maximum frequency is d. The number of non-zero frequencies is s. So with classical, you would have to say 2d. Ideally, we would like to say 2s. Actually, what you can say is something like 2s log d. So you have a log d price that you pay because you didn't know where they are. But still, it's much less than being linear in the dimension.

So essentially, the field of compressed sensing has been built around this observation, and the focus is slightly different. Instead of saying I want to do a statistical estimation where I can just build this, what you do is say I have a signal. And now I view this as a sensing matrix that I design with the property that I know will allow me to do this estimation well.

And so you basically assume that you can choose those vectors in certain ways, and then you can prove that you can reconstruct with much fewer samples, OK? And this has been used, for example-- I never remember for, as you call in MEG-- in what? No, MRI, MRI.

Two things I didn't tell you about, but it's worth mentioning are, suppose that what I tell you is actually it's not individual entries that are 0, but group of entries that are 0, for example,

because each entry is a biological process.

So I have genes, but genes are actually involved in biological process. So there is a group of genes that is doing something. I have a group of genes that are doing something, and I want to select is not individual genes, but groups. Can you twist this stuff in such a way that you select groups? Yes.

What if the groups are actually overlapping? How do you want to deal with the overlaps? Do you want to keep the overlap? Do you want to cut the overlap? What if you have a tree structure, OK? What do you do with this? So first of all, who gives it information, OK? And then if you have the information, how do you use it, and how are you going to use it?

See, this is the whole field of structure sparsity. It's the whole industry of building penalties other than L1 that would allow you to incorporate this kind of prior information. And if you want as the place, as in kernel methods, the kernel was the place where you could incorporate prior information.

This is the case where, in this field, you can do that by designing a suitable regularizer. And then a lot of the reason is this. So here we'll translate with these new regularizers.

The last bit is that, with a bit of a twist, some of the idea that I show you now that are basically related to vectors and sparsity translate to more general context, in particular that of matrices that have low rank, OK?

The classical example is suppose that I give you-- it's matrix completion, OK? I give you a matrix, but I actually delete most of the entries of the matrix. And I tell you, OK, estimate the original matrix. Well, how can I do that, right?

Well, it turns out that if the matrix itself had very low rank, so that many of the columns and rows you saw were actually related to each other, you might actually be able to do that. And the way you chose the entries to delete or select was not malicious, then you would be able to fill in the missing entries, OK? And the theory behind this is very similar to the theory that allows to fill in the right entries of the vector, OK?

Last bit-- PCA in 15 minutes. So what we've seen so far was a very hard problem of variable selection. It is still a supervised problem, where I give you labels, OK?

The last bit I want to show you is PCA, which is the case where I don't give you labels. And

what you try to answer is actually-- perhaps it's like the simpler question. Because you don't want to select one of the directions, but you would like to know if there are directions that matter. So you allow yourself, for example, to combine the different directions in your data, OK?

So this question is interesting for many, many reasons. One is data visualization, for example. You have stuff that you cannot look at because you have, for example, digits in very high dimensions.

You would like to look at them. How do you do it? Well, you like to find directions. The first direction to project everything, the second direction, three direction, because then you can plot and look at them, OK?

And this is one visualization of these images here. And I'll remember now the code now. It's written here. You have different colors, and what you see is that this actually did a good job. Because what you expect is that if you do a nice visualization, what you would like to have is that similar numbers or same numbers are in the same regions, and perhaps similar numbers are close, OK? So this is one reason why you want to do this.

One reason why you might want to do this is also because you might want to reduce the dimensionality of your data just to compress them or because you might hope that certain dimensions don't matter or are simply noise. And so you just want to get rid of that because this could be good for statistical reasons.

OK, so the game is going to be the following. X is the data space, which is going to be RD. And we want to define a map M that sends vectors of length D into vectors of length k. So k is going to be my reduced dimensionality.

And what we're going to do is that we're going to build a basic method to do this, which is PCA, and we're going to give a purely geometric view of PCA, OK? And this is going to be done by taking first the case where k is equal to 1 and then iterate to go up.

So at the first case, we're going to ask, if I give you vectors which are D dimensional, how can I project them in one dimension with respect to some criterion of optimality, OK? And here what we ask is, we want to project the data in the one dimension that would give me the best possible error.

So I think I had it before. Do I have it-- no, no. This was done for another reason, but it's useful now. If you have this vector and you want to project in this direction, and this is a unit vector, what do you do? I want to know how to write this vector here, the projection.

What you do is that you take the inner product between Yn and X. You get the number, and that number is the length you want to assign to X1, OK? So suppose that w is the direction. And I have a vector x, and I want to give the projection, OK? What do I do? I take the inner product of x and w, and this is the length I have to assign to the vector w, which is unit norm, OK?

So this is the best approximation of xi in the direction of w. Does it make sense? I fix a w, and I want to know how well I can describe x. I project x in that direction, and then I take the difference between x and the projection, OK?

And then what I do is that I sum over all points. And then I check, among all possible directions, the one that give me the best error. So suppose that is your data set. Which direction you think is going to give me the best error?

**AUDIENCE:** Keep going.

**LORENZO ROSASCO:** Well, if you go in this direction, you can explain most of the stuff, OK? You can reconstruct it best. So this is going to be the solution. So the question here is really, how do you solve this problem? You could try to minimize with respect to w.

But in fact, it's not clear what kind of computation you have. And if we massage this a little bit, it turns out that it is actually exactly an eigenvalue problem. So that's what we want to do next. So conceptually, what we want to do is what I said here and nothing more. I want to find the single individual direction that allows me to reconstruct best, on average, all the training set points.

And now what we want to do is just to check what kind of computation these entail and learn a bit more about this, OK? So this notation is just to say that the vector is norm 1 so that I don't have to fumble with the size of the vector.

OK, so let's do a couple of computations. This is ideal after lunch. So you just take this square and develop it, OK? And remember that w is unit norm. So when you do w transpose w, you get 1. And then if you-- and if you don't forget to put your square and if you just develop this, you'll see that this is an equality, OK? There is a square missing here.

So you have xi square. Then you would have the product of xi and this, which will be w transpose xi square. And then you would also have this square, but this square is w transpose xi and then w transpose w, which is 1. And so what you see is that this would create-- instead of three terms we have two because two cancel out-- not cancel out. They balance each other. OK.

So then I'd argue that if instead of minimizing this, because this is equal to this, instead of minimizing this, you can maximize this. Why? Well, because this is just a constant. It doesn't depend on w at all, so I can drop it from my functional.

The solution, the minimum, the minimum will be different, but the minimizer, the w that solves the problem, will be the same, OK? And then here is minimizing something with a minus, which is the same as maximizing the same thing without the minus, OK? I don't ask, so far, so good, because I'm scared.

So what you see now is that basically if the data were centered, basically this would just be a variance. If the data are centered, so there is a minus 0 here, maybe you can interpret this as measuring the variance in one direction.

And so you have another interpretation of PCA, which is the one where instead of picking the single direction with the best possible reconstruction, you're picking the direction where the variance of the data is bigger, OK?

And these two points of view are completely equivalent. Essentially, whenever you have a square norm, thinking about maximizing the variance or minimizing the reconstruction are two complementary dual ideas, OK? So that's what you will be doing here.

One more bit. What about computation? So this is-- so we can think about reconstruction. You can think about variance, if you like. What about this computation? What kind of computation is this, OK? If we massage it a little bit, we see that is just an eigenvalue problem.

So this is how you do it. This actually look-- so it's annoying, but it's very simple. So I wrote all the passages. This is a square, so it's something times itself. This whole thing is symmetric, so you can swap the order of this multiplication. So you get w transpose xi, xi transpose w.

But then this is just the sum that was going to involve these terms. So I can let the sum enter, and this is what you get. So you get w transpose 1/n xi xi transpose w. So this is just a

number. w transpose xi is just a number.

But the moment you look at something that looks like xi, xi transpose, what is that? Well, just look at dimensionality, OK? 1 times d times d times 1 gives you a number, which is 1 by 1. Now you're doing the other way around. So what is this?

**AUDIENCE:** It's a matrix.

**LORENZO ROSASCO:** It's a--

**AUDIENCE:** Matrix.

**LORENZO ROSASCO:** It's a matrix. And it's a matrix which is d by d, and it's of rank 1, OK? And what you do now is that you sum them all up, and what you have is that this quantity here becomes what is called the quadratic form. It is a matrix C, which just looks like this. And it's squeezed in between two vectors, w transpose and w.

So now what you want to do is that you can rewrite this just this way as maximizing the w-- sorry, finding the unit norm vector w that maximizes this quadratic form. And at this point, you can still ask me who cares, because it's just keeping on rewriting the same problem.

But it turns out that essentially using Lagrange theorem, it is relatively simple to do, you can check that-- oh, so boring-- that the solution of this problem is the maximum eigenvector of this matrix, OK? So this you can leave as an exercise.

Essentially, you take the Lagrangian of this and use a little bit of duality, and you show that the [INAUDIBLE] of this problem is just the maximum eigen-- so the eigenvalues-- ugh, the eigenvector corresponding to the maximum eigenvalue of the matrix C. So finding this direction is just solving an eigenvalue problem. OK.

I think just do last few of those slide, kind of cute. It's pretty simple, OK? So the one part, this line after lunch is a bit there for because I'm nice. But really, the only one part which is a bit more complicated is this one here. The rest is really just very simple algebra.

So what about k equal 2? I run out of time. But it turns out that what you want to do is basically if you say that you want to look for a second-- so you look at the first direction. You solve it, and you know that it's the first eigenvector-- the first eigenvector.

And then let's say that you add the constraint that the second direction you find has to be orthogonal to the first direction. You might not want to do this, OK? But if you do, if you say you add the orthogonality constraint, then what you can check is that you can repeat-- sorry, I didn't do it. It's on my note, the one that I have on the website, and the computation is kind of cute.

And what you see is that the solution of this problem looks exactly like the one before, only with this additional constraint, is exactly the eigenvector corresponding to the second largest eigenvalue, OK? And so you can keep on going. And so now this gives you a way to go from k equal to 1 to k bigger than 1, and you can keep on going, OK?

So if you're looking for the direction that maximizes the variance or the reconstruction, they turn out to be the biggest eigen-- the vectors-- ugh, the eigenvectors corresponding to the biggest eigenvalues of this matrix C, which what you can call it as a second moment or covariance matrix of the data.

OK, so this is more or less the end. This is the basic, basic, basic version of this. You can mix this with pretty much all the other stuff we said today. So one is, how about trying to use kernels to do a nonlinear extension of this?

So here we just looked at the linear reconstruction. How about nonlinear reconstruction? So what you would do is that you would first map the data in some way and then try to find some kind of nonlinear dimensionality reduction.

You see that what I'm doing here is that I'm just using this linear-- it's just a linear dimensionality reduction, just a linear operator. But what about something nonlinear? What if my data lie on some kind of structure that looked like that-- our beloved machine learning Swiss roll. Well, if you do PCA, well, you're just going to find a plane that cuts that thing somewhere, OK?

But if you try to embed the data in some nonlinear way, you could try to resolve this. And this has been much of the research done in the direction of manifold learning. Here there are just a few keywords-- kernel PCA is the easiest version, Laplacian map, eigenmaps, diffusion maps, and so on, OK?

I only touch quickly upon random projection. There is a whole literature about those. The idea is, again, that by multiplying the data by random vectors, you can keep the information in the

data and might be able to reconstruct them as well as to preserve distances.

Also, you can combine ideas from sparsity with ideas from PCA. For example, you can say, what if I want to know not only-- I want to find something like an eigenvector, but I would like the entries of the eigenvector to be 0.

So can I add here a constraint which basically says, among all the unit vectors, find the one whose entries are most-- so I want to add an L0 norm or L1 norm. So how can you do that, OK? And this leads to sparse PCA and other structured matrix estimation problems, OK?

So this is, again, something I'm not going to tell you about, but that's kind of the beginning. And this, more or less, brings us to the desert island, and I'm done.