LORENZO ROSASCO: I'm Lorenzo Rosasco. This is going to be a couple of hours plus of basic machine learning. OK. And I want to emphasize a bit, the word, "basic." Because really I tried to just stick to the essentials, or things that I would think of essentials to just start. Suppose that you have zero knowledge of machine learning and you just want to start from zero. OK. So if you already had classes in machine learning, you might find this a little bit boring or at least kind of rehearsing things that you already know.

The idea of looking at machine learning these days is coming from at least two different perspectives. The first one is for those of you, probably most of that are interested to develop intelligent systems in a very broad sense. What happened in the last few years is that there's been a kind of data-driven revolution where systems that are trained rather than programmed start to be the key engines to solve tasks. And here, there are just some pictures that are probably outdated, like robotics. You know, we have Siri on our phone. We hear about self-driving cars.

In all these systems, one key engine is providing data to the system to essentially try to learn how to solve the task. And so one idea of this class is to try to see what does it mean to learn? And the moment that you start to use data to solve complex tasks, then there is a natural connection with what today is called data science, which is somewhat a rapid [INAUDIBLE] renovated version of what we used to call just statistics.

So basically, we start to have tons of data of all kinds. They are very easy to collect, and we are starving for knowledge and trying to extract information from these data. And as it turns out, many of the techniques that are used to develop intelligent systems are the same very technique that you can use to try to extract relevant information patterns, data, from your data.

So what we want to do today is try to see a bit what's in the middle. What is the set of techniques that allows you, indeed, to go from data to knowledge or to acquiring ability to solve tasks.

Machine learning is huge these days, and there are tons of possible applications. There has been theory developed in the last 20, 30 years that brought the field to a certain level of maturity from a mathematical point of view. There have been tons and tons and tons of algorithms developed. OK. So in three hours, there is no way I could give you even just a little view of what machine learning is these days.

So what I did is pretty much this. I don't know if you've ever done this, but you used to do the mixtape, and you try to pick the songs that you would bring with yourself on a desert island. That's kind of the way I thought about what to put in this one [INAUDIBLE] lights that we're going to show in a minute.

So basically, I thought, what are those three, four, five learning algorithms that you should know, OK, if you know nothing about machine learning. And this is more or less at least one part. Of course there are a few songs that stayed out of the compilation, but this is like one selection. OK.

So as such, we're going to start, as I said-- whoop-- simple. And the idea is that this morning you're going to see a few algorithms. And I picked algorithms that are relatively simple from a computational point of view. So the math level is going to be pretty basic. OK. I think I'm going to use some linear algebra at some point and maybe some calculus, but that's about it. So most of the idea here is to emphasize conceptual ideas, the concepts.

And then today, afternoon, there's going to be, basically labs where you sit and you just pick these kind of algorithms and use them, so you immediately see, what does it mean? OK. So at the end of the day, you should have reasonable knowledge about whatever you're seeing this morning.

So this is how the class is structured. It's divided in parts plus the lab. So the first part, what we want to do is start from probably the simplest learning algorithm you can think of to try to emphasize, and use that as an excuse to introduce the idea of bias-variance, trade-off, which, to me, is probably either the, or one of the most fundamental concepts in statistics and machine learning, which is this idea that you're going to see in a few minutes in more detail.

But it's essentially the idea that you never have enough data. OK. And the game here is not about describing the data that you have today, as much as using the data you have today as a basis of knowledge to describe data you're going to get tomorrow. So there is this inherent trade-off between what you have at disposal and what would you like to predict.

And then, essentially it turns out that you have to somewhat decide how much you want to trust the data, and how much you want to somewhat throw away, or regularize, as they say, smooth out the information in your data, because you think that it's actually an accident. It's just because you saw data with aspects today that are not really reflective of the phenomenon that produced them. But it's just because I saw 10 points rather than 100.

The basic idea here is essentially the law of large numbers. When you toss a coin, you might find out that if you toss it just 10 times, it looks like it's not a fair coin, but if you go for 100, or 1,000, you start to see that it converts to 50-50. OK. So that's kind of what's going on here. So the idea is that you want to use some kind of induction principle that tells you how much you can trust the data.

Moving on from this basic class of algorithms, we're going to consider so-called regularization techniques. I use regularization in a very broad sentence. And here we're going to concentrate on least squares essentially because A, it's simple, and it just reduces to linear algebra. And so you don't have to know anything about convex optimization or any other kind of fancy optimization techniques. And B, because it's relatively simple to move from linear models to non-parametric non-linear models using kernels. OK. And kernels are a big field with a lot of math, but you're just going to look more at the recipe to move from simple models to complicated models.

So finally, the last part, we're going to move a bit away from pure prediction. So basically these first two parts are about prediction, or what is called supervised learning. And here we're going to move a bit away from prediction and we're going to ask questions more related to, you have data, and you want to know, what are the important sectors in your data?

So the one key word here is interoperability. You want to have some form of interoperability of the data at hand. You would like to know, not only how you can make good predictions, but what are the important sectors. So you not only want to do good prediction, but you want to know how you make good prediction. What is the important information to actually get good prediction. And so, in this last part we're going to take a peek into this.

And as I said, the afternoon is basically going to be a practical session. If it's all MATLAB I think there is some quick-- if you have never seen MATLAB before, you can play around with just a little bit. But it's very easy and then you've got a few different proposals I think, of things you can do. And you can pick, depending on what you already know and what you can try, you can start from that and be more or less fancy. OK.

So it goes without saying, stop me. I mean, the more we interact, the better it is. So the first part, as I said, the idea is to use so-called local methods as an excuse to understand it by experience. OK. So we're going to introduce the simplest algorithm you can think of, and we're going to use it to understand a much deeper concept.

So first of all, let's just put down our setup. The idea is that we are-- so how many of you had a machine learning class before? All right. So, you won't be too bored. The idea is we want to do supervised learning. So in supervised learning there is an input and an output. And these inputs and outputs are somewhat related. And I'll be more precise in a minute. But the idea is that you want to learn this input-output relationship. And all you have at disposal are sets of inputs and outputs. OK.

So x here is an input, and y is the output. f is a functional relation between the input and the output. All you have in this puzzle are these couples, OK. So I give an input, and then what's the corresponding output? I give another input and I know what's the corresponding output. But I don't give you all of them. You just have n, OK. n is the number of points, and you call this a training set, because it will be the basis of knowledge in which you can try to train a machine to estimate this functional relationship. OK.

And the key point here is that, on the one hand, you want to describe these data. So you want to get a functional relationship that works well that, if you get the next one to give you an f(x1), which is close to y1 and so on. And f(x2), which is close to y2. But more importantly, you want an f, that given a new point that was not here, will give you an output, which is a good estimate of the true output to correspond to that input. OK. This is the most important thing of the setup. OK. The ideal, so-called generalization, if you want prediction. If you want to really do inference. You don't want to do descriptive statistics. You really want to do inferential statistics.

So this is just very, very simple example, but just to start to have something in mind. Suppose that you have-- well, it's just like a toy version of the face recognition system we have on our phones. You know that when you take a picture, you start--

AUDIENCE: Sorry.

LORENZO They really weren't talking. You have something like this. You have a little square appearing

ROSASCO: around a face sometimes. It means that basically the system is actually going inside the image and recognizing faces. OK. So the idea is a bit more complicated than this. But a toy version of this algorithm is, you have an image like this. OK. The image you think of as a matrix of numbers. Now this is color, but imagine it's black and white, OK.

Then it would just contain a number, which is the pixel value with the light intensity of that pixel. And you just have this array. And then if you want you can brutalize it with and just unroll the matrix into a long vector. OK. That gives one vector. So p here would be what? The number of? Just the number of pixels. OK. So I take this image and I unroll it. I take another image and I unroll it. And I take images. And you see, some images here do contain faces. Some of the images do not contain faces. OK. And I here use color to code them.

And now what I have is that images are my inputs, OK, are the x's. So here-- full disclosure, I never use the little arrow above letters to denote vectors. So hopefully it will be clear from the context. When it's really useful I use upper or lower indices. Anyway. So this is the data matrix. Rows are inputs and columns are so-called features or variables, are the entries of each vector. OK. And I have n rows and p columns.

Associated to this, I have my output vector. And what is the output vector? Well in this case, it's just a simple binary vector. And the idea here is, if there is a face, I put 1. If there is not a face, I put minus 1. OK. So this is the way I turn, like an abstract question, recognize faces in images, into some data structure that in a minute we're going to elaborate to try to actually answer the question, whether there is a face in an image or not. OK.

So this first step, it's kind of obvious in this case, but it's actually a tricky step. OK. It's the part that I'm not going to give you any hint about. It's kind of an art. You have data and you have-- at the very beginning you have to turn them into some kind of manageable data structure. OK.

Then you can elaborate in multiple ways. But the very first step is you deciding-- for example, here we decided to unroll all these numbers into vectors. This sounds like a good idea or a bad idea? One thing that you're doing is that the pixel here and the pixel here are probably related. And in this case there is some structure in the image. And so when you take this pixel 136, and you unroll it, it comes here. So they're not close. OK.

Now here it turns out that if you think about it-- you'll see a minute. For those of you who remember, if you just took Euclidean distance, you take product of numbers and you sum them up. That's invariant to the position of the individual pixels. So that's OK. OK. But yet again, there is this intuition that, well, maybe here I'm losing too much geometric information about the context of the image.

And indeed, while this kind of works in practice, but if you want to get better results you have to do the fancy stuff that Andrei was talking about today, looking locally and try to look at collection, try to keep more geometric information. OK. So I'm not going to talk about that kind of stuff. This up to date, a lot of engineering, and some good way to learn it. But we're going to try to just stick to simple representations. OK. So how do you build representation is now going to be part of what I'm going to talk about.

So imagine that either and you stick to this super-simple representation or some friends of yours come in and put the box here in the middle, where you put this array of numbers and you extract another vector much fancier than this that contains some better representation of an image. OK. But then at the end of the day, my job starts when you give me a vector representation that I can trust. And I can basically say that if two vectors seem similar, they should have the same label. And that's the basic idea. OK. All right.

So a little game here is, OK, imagine that these are just the two-pixel version of the images I showed you before. You have some boxes, some circles. And then I give you this one triangle. It's very original. Andrei showed you this yesterday. And the question is, what's the color of that? OK. Unless you haven't slept a minute, you're going to say it's orange. But the question is, why do you think it's orange?

AUDIENCE: [INAUDIBLE]

LORENZO ROSASCO: Say it again?

AUDIENCE: It's surrounded by oranges.

LORENZO ROSASCO: It's surrounded by oranges. OK. And she said, it's close to oranges. So it turns out that this is actually the simplest algorithm you can think of. OK. You check who you have close to you, and if it's orange, you say orange. And if it's blue, you say blue. OK.

But we already made an assumption here, which we ask in the question, which is the nearby things. So we are basically saying that our of vectoral representation is such that, if two things are close-- so I do have a distance, and if two things are close, then they might have the same semantic content. OK. Which might be true or not.

For example, if you take this thing I showed you here, we cannot just draw it, right? We cannot just take 200 times 200 vectors and just look at them and say, yeah, you know, a visual inspection. You have to believe that this distance will be fine. And so the discussion that we just had about what is a good representation is going to kick in. OK.

But the assumption you make-- in this case visually it's very easy, it's low dimension-- is that nearby things have similar labels. One thing that I forgot to tell you in the previous slides, but it's key, is exactly this observation that in machine learning we typically move away from situations like this one, where you can do visual inspection and you have low dimensionality, to kind of a situation like the one I just showed you a minute before, where you have images. And if you have to think of each of these circles as an image, you want to be able to draw it, because it's going to be several hundred typically, or tens dimensional vector. OK.

So the game is kind of different. Can we still do this kind of stuff? Can we just say that closed things should have the same semantic content? That's another question we're going to try to answer. OK. But I just want to do a bit of inception. This is a big deal, OK, going from low dimension to very high dimensions. All right.

But let's stick for a minute to the idea that nearby things should have the same label, and just write the one line, write down the algorithm. It's the kind of case where it's harder to write it down than to code it up or just explain what it is. It's super simple. What you do is, you have data points, Xi. So Xi is the training set, the input data in the training set. X-bar is what I call X-new before. It's a new point.

What you do is that you search. This just says, look for the index of the closest point. That's what you did before. OK. So here, I-prime is the index of the point Xi closest to X-bar. Once you find it, go in your dataset and find the label of that point. And then assign that label to the new point. Does that makes sense? Everybody's happy? Not super-complicated. Fair enough.

How does it work? So let me see if I can do this. This is extremely fancy code. Let's see. All right. So what did I do? Let me do it a bit smaller. So this is just simple two-dimensional datasets. I take 40 points. The dataset looks like this. The dataset is the one on the left. OK. And what I do, I take 40 points. And to make it a bit more complex, I flip some of the labels. OK. So you basically say that the two datasets-- this is called the two moons dataset, or something like this. And what I did is that some of the labels in this sea, I changed color. I changed the label. OK. So I made the problem a bit harder.

And here is what fortunately you don't have in practice. OK. Here we're cheating. We're doing just the simulations. We're looking at the future. We assume that because we can generate this data, we can look at the future and check how we're going to do in future data. So you can think of this as a future data that typically you don't have. So here you're a normal human being. Here you're playing god and looking at the future. OK. Because we just want to do a little simulation.

So based on that, we can just go here and put 1, train, and then test and plot. So what you see here is the so-called decision boundary. OK. What I did is exactly that one line of code you saw before. OK. And what I did is, in this case I can draw it, because it's low dimensional. And basically what I do is that I just put in the regions where I think I should put orange, and the region where it think I should put blue. OK.

And here you can kind of see what's going on. These are actually very good on the data, right? How many mistakes do you make on the new dataset? Sorry, on the training set? Zero. It's perfect. OK. Is that a good idea? Well, when you look at it here, it doesn't look that good. OK. There is this whole region of points, for example, that are going to be predicted to be orange, but they're actually blue. Of course if you want to have zero errors in the training set, there's nothing else you can do, right? Because you see, you have this orange point here. You have these two orange points here. And you want to go and follow them. So there's nothing you can do. So this is the first observation.

The second observation is, the curve, if you look close enough, it's piecewise linear. It's like a sequence of linear pieces stuck together. If we just try to do a little game and generate some new data-- OK, so imagine again, I'm playing god now. I generate the new dataset that it should look like. So take another peek at this. OK. Oop.

So now I generate them. I plot them. I train. And now let's test. OK. If you remember the decision curves you've seen before, what do you notice here?

AUDIENCE: they're different

LORENZO ROSASCO: They're very different. OK. For example, the one before, if you remember, we noticed it was going all the way down here to follow those couple of points. But here you don't have those couple of points. OK. So now, is that a good thing or a bad thing? Well the point here is that because you have so few points, the moment you start to just feed the data, this will happen. OK. You have something that changes all the time. It's very unstable. That's a key word, OK. You have something that you change the data just a little bit, and it changes completely. That sounds like a bad idea. OK.

If I want to make a prediction, if I keep on getting slightly different data and I change my mind completely, that's probably not a good way to make a prediction about anything. OK. And this is happening all the time here. And it's exactly because our algorithm is in some sense is greedy. You just try to get perfect performance on the training set without worrying much about the future. Let's do this just once more.

OK. And we keep on going. It's going to change all the time, all the time. Of course-- I don't know how much I can push this because it's not super-duper fast. But let's try. Let's say 18 by 30. So what I did now is just that I augmented the number of points in my training set. It was 20 or 30, I don't remember. Now it make it 100. So now you should see-- OK. So this is one solution. We want to play the same game. We just want to generate other datasets of the same. So maybe now it might be that I took them all. I don't remember how many there are. No, I didn't take them all.

So, what do you see now? We are doing exactly the same thing. OK. And is this something that you can absolutely not to do in practice, because you cannot just generate datasets. But here what you see is that I just augmented the number of training set points. And what you see is now the solution does change, but not as much. OK. And you can kind of start to see that there is something going on a bit like this here. OK. So this one actually looks pretty bad. Let's try to do it once more.

OK. So again, it does change a lot, but not as much as before. And you roughly see that this guy says that, here it should be orange and here should be blue. OK. So that's kind of what you expect. The more points you get, the better your solution would get. And if I put hear all the possible points, what you will start to see is that the closest point to any point here will be a blue point. OK. So it will be perfect.

So if I ask you if this is a good algorithm or not, what would you say?

AUDIENCE: It's overfitting the data.

LORENZO It's kind of a overfitting the data. But it is not always overfitting the data. If the data are good,

ROSASCO: it's a good idea to fit them. OK. But in some sense, this algorithm doesn't have a way to prevent itself to fall in love with the data when there are very few. And if you have very few data points, you start to just wiggle around, become extremely unstable, change your mind all the time. If the data are enough, it stabilizes, and in some senses, this setting, we're fitting the data, or as she's saying, overfitting the data. It's actually not a bad thing. OK. So this is what's going on here.

AUDIENCE: What do you mean by overfitting?

LORENZO ROSASCO: Fitting a bit too much. So if you look here. So here, if you look what you're doing here, you're always fitting the data OK. But here you're doing nothing else. And so if you have few data points, fitting the data is fine. Sorry, if you have many data points, fitting the data is just fine. If you have few data points, by fitting them you, in some sense, overfit in the sense that when you look at new data points, you have done a bit too much. OK. What you saw before, that you get something that is very good, because it perfectly fits that, but it's overfitting with respect to the future. Whereas here, the fitting on the left-hand side kind of reflects, not too badly the fitting on the right-hand side. OK.

So the idea of overfitting and stability that came out in this discussion are key. OK. If you want everything we're going to do in the next three hours, understand how you can prevent overfitting and build a good way to stabilize your algorithms. OK. So let's go back here. This is going to be quick, because if I ask you, what is this? What would you say?

AUDIENCE: [INAUDIBLE]

[LAUGHING]

LORENZO ROSASCO: So the idea is that, when you have a situation like this, you're still pretty much able to say what's the right answer. And what you're going to do is that you're going to move away from just saying, what's the closest point, and you just look at a few more points. You just don't look at one. OK. You look at, how many? boh? "boh" is very useful Italian word. It means, I don't know.

So these algorithm-- it's called the k nearest neighbor algorithm, it's probably the second simplest algorithm you can think of. It's kind of the same as before. The notation here is a bit boring, but it's basically saying, take the points. Give them new points. Check the distance with everybody. Sort it and take the first k. OK. If it's a classification problem, it's probably a good idea to take an odd number for k, so that you can then just have voting. And basically everybody votes. Each vote counts one. And somebody says blue, somebody says orange, and you make a decision. OK. Fair enough.

Well how does this work? You can kind of imagine. So what we have to do-- so for example here we have this guy. OK. Now let's just put k-- well, let's make this a bit smaller. So we do 40. Generate, plot, train. [INAUDIBLE] test. Plot. OK. Well we got a bit lucky, OK. This is actually a good dataset, because in some sense there are no, what you might call outliers. There are no orange points that really go and sit in the blue.

So I just want to show you a bit about the dramatic effect of this. So I'm going to just try to redo this one so that we get the more-- yeah, this should do. OK. So this is nearest neighbor. This is the solution you get. It's not too horrible. But, for example, you see that it starts following this guy. OK.

Now, what you can do is that you can just go in and say, four. Well, four's a bad idea. Five. You'd retrain them the same. And all of a sudden it just ignores this guy. Because the moment that you put more in, well, you just realize that he's surrounded by blue guys, so it's probably just, his vote just counts one against four. OK. And you can keep on going.

And the idea here is that the more you make this big, the more your solution is going to be, what? Well you say, it's going to be good, but it's actually not true. Because if you start to put k too big, at some point all you're doing is counting how many points you have in class one, counting how many points you have in class two, and always say the same thing. OK. So I'm going to put here, 20.

What you start to see is that you start to obtain a decision boundary, which is simpler, and simpler and simpler. OK. It looks kind of linear here. What you will see is that, suppose that now I regenerate the data. And you remember how much it changed before when I was using nearest neighbor with just k equal to 1.

So of course here, you know, it's probabilistic. OK. So of course I'm going to get a dataset like the one I just showed you minutes ago, and I had it as fast as possible. Because if I pick 10, one is going to look like that and nine are going to look like this. OK. And when they look like this, you see, they kind of start to have this kind of line, like a decision boundary with some twists. But it's very simple. OK.

And if at some point, if I put k big enough-- that is, the number of all points, it won't change any more. OK. It will just be essentially dividing the sets in two equal parts. So does that makes sense? So would it make sense to vote to make different votes? Essentially, the idea is, if the point is closest, his vote should count more than if a point is more far away? Yes, absolutely. Let's say here we're making the simplest thing in the world, the second simplest thing in the world, the third simplest thing in the world. It is doing that. OK.

And you can see that you can go pretty far with this. I mean, it's simple, but these are actually algorithms that are used sometimes. And what you do is that, if you just look at this-- again, these I don't want to explain too much. If you've seen it before, it's simple. Otherwise it doesn't really matter.

But the basic idea here is that each vote is going to be between 0-- so, you see here I put the distance between the new point and all the other points on top of an exponential. So the number I get is not 1, but it is between 0 and 1. If the two points are close, and the limits supposedly are the same, it becomes a 0, and it counts exactly one. If they're very far away, these would be, say, infinity and then we'd be close to 0. So the closest you are, the more you count.

If you want, you can read it like this. You're sitting on a new point, and you put a zooming window. Yeah, like a zooming window of a certain size. And you basically check that everything which is inside this window will be closed. And the more you go farther away-- so the window is like this. And you deform the space so that basically what you say is, things that are far away, they're going to count less.

And if I move sigma here, I'm somewhat making my visual field, if you want, larger or smaller, around this one new point. It's just a physical interpretation of what this is doing. There are 15 other ways of looking at what the Gaussian is doing. Voting, changing the weight of the vote is another one. OK.

Why the Gaussian here? Well, because. Just because. You can use many, many others. You can use, for example, a hat window. And this is part of your prior knowledge, how much you want to weight. If you are in this kind of low dimensional situation, you might have good ways to just look inside the data and decide almost like doing by a visual inspection. Otherwise you have to trust some more broad principles. And it's again back to the problem of learning the representation and deciding how to measure distance, which are two phases of the same story. OK.

And the other thing you see is that, if you start to do these games, you might actually add more parameters. OK. Because we start from nearest neighbor, which is completely parameter-free, but it was very unstable. We added k. We allow ourselves to go from simple to complex, from stability to overfitting. But we introduced a new parameter.

And so that's not an algorithm any more. It's a half algorithm. A true algorithm is a parameter-free algorithm where I tell you how you choose everything. OK. So if they just give you something, say, yeah, there's k, well, how do you choose it? OK. It's not something you can use. And here I'm adding sigma. And again, you have to decide how you use it. OK. And so that's what we want to ask in a minute.

So before doing that, just a side remark is-- we've been looking at vector data. OK. And we were basically measuring distance through just the Euclidean norm, OK, just the usual one, or this version like the Gaussian kernel that somewhat amplifies distances. What if you have strings, for example, or graphs? OK. Your data turns out to be strings and you want to compare them? Say even if they're binary strings, there's no linear structure. You cannot just sum them up. the Euclidean distance doesn't really make a lot of sense.

But what you can do is that as long as you can define a distance-- and say this one would be the simplest one, just the Hamming distance. You just check entries, and if they're the same, you count one. If they're different, you count zero. OK. The moment you can define a distance of your data, then you can use this kind of technique. So this technique is pretty flexible in that sense, that whenever you can give-- you don't need a vectoral representation, you just need a way to measure, say, similarity or distances between things, and then you can use this method. OK. So here I just mentioned this, and that's what most of these classes are going to be, about vector data.

But this is one point where, the moment you have k-- you can think of this case sometimes as a similarity. OK. Similarity is kind of concept that is dual to distances. So if the similarity is big, it's good. The distance small is good. OK. And so here, if you have a way to build the k or a distance, then you're good to go.

And we're not going to really talk about it, but there's a whole industry about how you build this kind of stuff. So we give restraints. Maybe I want to say that I should not only look at the entry of a string, but also the nearby entry when I make the score for that specific. So maybe I shifted a value of the string a little bit. It's not right here. It's in the next position over, so that should come to bits. So I want to do a soft version of this. OK. Or maybe I have graphs, and I want to compare graphs. And I want to say that if two graphs are close, then I want them to have the same label. OK. How do you do that?

The next big question is-- we introduced three parameters. They look really nice, because they kind of allowed us to get more flexible solutions to the problem by choosing, for example, k or the sigma in the Gaussian. We can go from overfitting to stability. But then of course we have to choose the parameter, and we have to find good ways to choose them.

And so there are a bunch of questions. So the first one is, well, is there an optimal value at all? OK. Does it exist? But if it does exist, I can go try to estimate it in some way. If it doesn't, well it does not even make sense. I just throw a random number. I just say, k equals 4. Why? Just because. OK.

So what do you think? It exists or not? What does it depend on? Because that's the next question. What does it depend on? Can we compute it? OK. So let's try to guess one minute before we go and check how we do this. OK. OK.

I have to choose it. How do I choose it? What does it depend on?

AUDIENCE: Size of this.

LORENZO ROSASCO: One thing is the size of the dataset. Because what we saw is that a small k seems a good idea when you have a lot of data, but it seems like a bad idea when you have few. OK. So it should depend. It should be something that scales with n, the number of points, and probably also the training set itself. But we want something that works for all datasets, say, in expectation. So cardinality of the training set is going to be a main factor. What else?

AUDIENCE: The smoothness of the boundary.

LORENZO ROSASCO: The what?

AUDIENCE: The smoothness.

LORENZO This smoothness of the boundary. Yeah. So what he's saying is, if my problem looks like this,

ROSASCO: or if my problem looks like this, it looks like k should be different. In this case I can take any arbitrary high k-- sorry, small k, I guess, or i. It doesn't matter, because whatever you do, you pretty much get the good thing. But if you start doing something like this, then you want-- k is enough, because otherwise you just start to blur everything. And this is exactly what he's saying. If your problem is complicated or it's easy. OK.

And at the same time, this is related to the fact of how much noise you might have in the data, OK, how much flipping you might have in your data. If the problem is hard, then you expect to need a different k. OK. So it depends on the cardinality of the data, and how complicated is the problem? How complicated it is the boundary? How much noise do I have? OK.

So it turns out that one thing you can ask is, can we prove it? OK. Can we prove a theorem that says that there is an optimal k, and it really does depends on this, on this quantities. And it turns out that you can. Of course, as always, to make a theory or to make assumptions, you have to work within a model. And the model we want to work on is the following. You're basically saying, this is the k nearest neighbor solution. So big k here is the number of neighbors, and this is hat because it depends on the data. And what I say here is that I'm just going to look at squared loss error, just because it's easy. And I'm going to look at the regression problem, not just this classification.

And what you do here is that you take expectation over all possible input-output pairs. So basically you say, when I tried to do math, I want to see what's ideal. An ideally I want a solution that does well on future points. OK. So how do I do that? I think the average error over all possible points in the future, x and y. So this is the meaning of this first expectation. Make sense? Yes? No?

So if they fix y and x, this is the error on a specific couple input and output. I give you the input. I do f(kx) and then I check if it's close or not to y. But what I want to do if I want to be theoretical is to say, OK, what I would really like to be small is this error over all possible points. So I take the expectation, not the one on the training set, the one in the future. And I take expectation so that if points are more likely to be simple, they will count more than points that are less likely to be simple. OK.

AUDIENCE: What was Es?

LORENZO ROSASCO: We haven't got to that one yet. OK. So Exy is what I just said. What is Es? It's the expectation over the training set. Why do we need that? Well because if we don't put that expectation, I'm basically telling you what's the good k for this one training set here.

Then I give you another training set and I get another one, which is in some sense is good, but it's also bad, because we would like to have a take-home message that we hold for all training sets. And this is the simplest. You say, for the average training set, this is how I should choose k. That's what we want to do. OK. So the first expectation is to measure error with respect to the future. The second expectation is to say, I want to deal with the fact that I have several potential training sets appearing. OK.

So in the next couple of slides, this red dot means that there are computations. OK. And so I want to do them quickly. And the important thing of this bit is, it's an exercise. OK. So this is an exercise of stats zero. OK. So we don't want to spend time doing that. The important thing is going to be the conceptual parts. I'm going to go a bit quickly through it.

So you start from this, and you would like to understand if there exists-- so this is the quantity that you would like to make small, ideally. You will never have access to this, but ideally, in the optimal scenario, you want k to make this small. OK. Now the problem is that you want to essentially mathematically study this m minimization problem, but it's not easy, because, how do you do this? OK. The dependence of this function on k is complicated. It's that equation we had before, right? So you kind of just take the derivative and set it equal to zero. Let's keep on going into to.

So what we are at is, these are the points I would like to make small. I would like to choose k so that I can make this small. I want to study this from a mathematical point of view. But I cannot just use what you're doing in calculus, which is taking a derivative and setting it equal to zero, because the dependence of these two k, which is my variable, it's complicated. OK. So we go a bit of a round way. We turn out to be pretty universal.

And this is what we are going to do. First of all, we assume a model for our data. And this is just for the sake of simplicity. OK. I can use a much more general model. But this is the model. I'm going to say that my y are just some fixed function of star plus some noise. OK. And the noise is zero mean and variance sigma square for all entries. OK. This is the simplest model. It's a Gaussian regression model.

So one thing I'm doing, and this is like a trick and you can really forget it, but it just makes life much easier is that I take the expectation over xy and a condition here. OK. The reason why you do this is just to make the math a bit easier. Because basically now, if you put this expectation out, and you look just at these quantities, you're looking at everything for fixed x. And these just become a real number, OK, not the function anymore. So you can use normal calculus. You have a real-valued function and you can just use the usual stuff. OK.

Again, I'm going to going a bit quickly over this because it doesn't really matter. So this ingredient one. This is observation two. Observation three is that you need to introduce an object between the solution you get in practice and this ideal function. What is this? It's this kind of, what is called the expectation of my algorithm. What you do is that-- in my algorithm what I do here is that I put Yi, i OK, just the label of my training set. And the label are noisy. But this is an ideal object where you put the true function itself, and you just average the value of the true function.

Why do I use this? Because I want to get something which is in between this f-star and this f-hat. So if you put k big enough-- so if you have enough points, this is going to be-- sorry, if you take k small enough-- so this is closer to f-star than my f-hat, OK, because you get no noisy data. And what I want to do-- oops. What I want to do is that I want to plug it in the middle and split this error in two. And this is what I do. OK. If you do this, you can check that you have a square here. You get two terms. One simplifies, because of this assumption on the noise, and you get these two terms. OK.

And the important thing is these two terms are-- one is the comparison between my algorithm and its expectation. So that's exactly what we called a variance. OK. And one is the comparison between the value of the true function here, and the value of this other function. Sorry, this should be-- oh yeah. This is the expectation, which is my ideal version of my algorithm, the one that has access to the noiseless labels. OK. It's what you call a bias. It's basically because, instead of using the exact value of the function, you blur it a bit by averaging out. OK. You see here, instead of using the value of the function, you average out a few nearby values. So you're making it a bit dirtier.

The question now is, how would these two quantities depend on k? How this quantity depends on k and how this quantity depends on k. OK. And then by putting this together, we'll see that we have a certain behavior of this, and a certain behavior of this. And then balancing this out, we'll get what the optimal value looked like. And this is going to be all useless from-- so these are going to be interesting from a conceptual perspective. We're going to learn something, but we'll still have to do something practical, because nothing of this you can measure in practice.

OK.

So the next question would be, now that we know that it exists and it depends on this stuff, how can we actually approximate it in practice? And cross-validation is going to pop out of the window. OK. But this is the theory that shows you that this would help proving a theory that shows that cross-validation is a good idea, in a precise sense.

The take-home message is, by making this model and using this as an intermediate object, you split the error in two, and you start to be able to study. And what you get is basically the following. This term, by basically using-- so we assume that the data-- I didn't say that, but that's important. We assume that the data are independent with each other. OK. And by using that, you get these results right away, essentially using the fact that the variance of the sum of the independent variable is the sum of the variances. You get these results in one line. OK.

And basically what this shows is that, if k gets big-- so variance is another word for the stability. OK. So if you have a big variance, things will vary a lot. It will be unstable. So what you see here is exactly what we observe in the plot before. If k was big, things are not changing as much. If k was small, things were changing a lot. OK. And this is the one equation that shows you that. OK. And if you just look at that, it would just tell you, the big is better. Big, respect to what? To the noise. OK. If there is a lot of noise, I should make it bigger. If there's more noise, I can make it smaller.

But the point is that we saw before is that the problem of putting k large was that we were forgetting about the problem. We're just getting something that was very stable but could be potentially very bad, if my function was not that simple. OK. This is a bit harder to study mathematically. OK. This is a calculation that I show you because you can do it yourself in like 20 minutes, or less. This one takes a bit more. But he can get the hunch on how it looks like.

And the basic idea is what we already said. If k is small, and the points are close enough, instead of f-star x, we are thinking of f-star Xk, Xi. And the i is closing off. OK. Now if we start to put k bigger, we start to blur that prediction by looking at many nearby points. But here there is no noise. OK. So that sounds like a bad idea. So we expect the error in that case to be either increasing, or at least flat with respect to k. So when we take k larger, we're blurring this prediction, and potentially make it far away from the true one. OK.

And you can make this statement precise. You can prove it. And if you will prove it, it's basically that you have-- what happened? You have linear dependence. So the error here is linearly increasing or polynomially increasing-- in fact I don't remember-- with respect to k. OK. So the reason why I'm showing you this, skipping all these details, is just to give you a feeling of the kind of computation that answered the question if there is a optimal value and what it depends on.

And then at this point, once you get this, you start to see this kind of plot. And typically here I put them the wrong way. But here you basically say, I have this one function I wanted to study, which is the sum of two functions. I have this, and I have this. OK. And now to study the minimum, I'm basically going to sum them up and see what's the optimal value to optimize this too. And the k that optimized this is exactly the optimal k. And you see that the optimal k will behave as we expected. OK.

So here, one ingredient is missing. And it's just missing because I didn't put it in, which is the number of points. OK. It's just because I didn't renormalize things. OK. It should be a 1 over n here. It's just that I didn't renormalize. OK. But you announced it, and it's good, because it's true. There should be a 1 over n there. But the rest is what we expected. OK.

In some sense what we expect is that if my problem is complicated, I need the smaller k. If there is a lot of noise, I need a bigger k. And depending on the number of points, which would be in the numerator here, I can make a bigger or a larger. k. OK.

This plot is fundamental because it shows some property which is inherent in the problem. And the theorem that somewhat is behind it-- intuition I've been saying, repeating over and over, which is this intuition that you cannot trust the data too much. And there is the optimal amount of trust you can of your data based on certain assumptions. OK. And in our case, the assumption where this kind of model. So little calculation I'll show you quickly, grounds this intuition into a mathematical argument. OK.

All right. So we spent quite a bit of time on this. In some sense, from a conceptual point of view, this is a critical idea. OK. Because it's behind pretty much everything. This idea of, how much you can trust or not of the data. Of course here, as we said, this has been informative, hopefully. But you cannot really choose this k, because you would need to know the noise, but especially to know how to estimate this in order to minimize this quantity.

So in practice what you can show is, you can use what is called cross-validation. And in effect, cross-validation is one of a few other techniques you can use. And the idea is that you don't have access [AUDIO OUT] but you can show that if you take a bunch of data points, you split them in two, you use half for the training as you've always done, and you use the other half as a proxy for this future data. Then by minimizing the k-- taking the k that minimized the error on this so-called holdout set, then you can prove it's as good as if you could have access to this. OK. And it's actually very easy to prove. You can show that if you're just split in two, and you minimize the error in second half-- you do what is called the holdout cross-validation-- it's as good as if you'd had access to this. OK. So it's optimal in a way.

Now, the problem with this is that we are only looking at the area and expectation. And what you can check is that if you look at higher order statistics, say that variance of your estimators and so on and so forth, what you might get is that by splitting in two, [AUDIO OUT] big is fine. In practice the difference is small, you might get that the way you split might matter. You might have bad luck and just split in a certain way. And so there is a whole zoology of ways of splitting. And the basic one is, say, split-- this is, for example, the simplest. OK. Split in a bunch of groups. OK. k-fold or v-fold cross-validation. Take one group out of the time. OK. And do the same trick. You know, you train here and calculate the error here for different k's. Then you do the same here, do the same here, do the same here. Sum the errors up, renormalizing, and then just choose the k that minimizes this new form of error.

And if the data there are small, small, small, then typically this set will become very small. And then delimited, it becomes one, the leave one out error. OK. What you do is that you literally leave one out, train on the rest, get there for all the values of k in this case. Put it back in, take another one out, and repeat the procedure.

Now the question that I had 10, 15 minutes ago was, how do you choose v? OK. Shall I make this two? So I just do one split like this? Or shall I make it n, so I do leave one out? And as far as I know there is not a lot of theory that would support an answer to this question. And what I know is mostly what you can expect intuitively, which is, if you have a lot of data points-- what does it mean a lot? I don't know. If you have two million, 10,000, I don't know. If you have a big dataset, typically splitting in two, or maybe doing just random splits is stable enough. What does it mean? That you try, and you look at how much it moves.

Whereas if you have say-- you know, I don't know if it even exists, the implication like, you know, a few years ago there were micro-reapplication where you would have 20, 30 inputs, and you have 20 dimensions. And then in that case, you really don't do much splitting. If you have 20, for example, you try to leave one out and it's the best you can do. And it's already very unstable and sucks. OK. So in this case, there is work to be done. I mean, as far as I know, that's the state of things.

OK. So we introduced a class of very simple algorithms. They seem to be pretty reasonable. They seem to allow us, provided that we have a way to measure distances or similarity, to go from simple to complex. And we have some kind of theory that tells us what is the optimal value of a parameter, a kind of practical procedure to actually choose it in practice. OK.

Are we done? Is that all? do we need to do anything else? What's missing here? One thing that is missing here is that most of the intuition we developed so far are really related to low dimension. OK.

And here, very quickly, if you just do a little exercise where you try to say how big is a cube that covers 1% of the volume of a bigger cube of a unit length? OK. So the big cube is volume 1. The length of that is just 1. And it ask you, how big is this, if it has to cover 1% of the volume? It's really to check that these are just going to be a dth-root where d is the dimension of the cube. And this is the shape of the dth-root. OK.

So if you're in low dimension, basically, 1% is intuitively small within the big cube. But as soon as you're go in higher dimensional, what you see is that the length of the edge of the little cube that has to cover 1% of the volume becomes very close to 1, almost immediately. It's this curve going up. OK.

What does it mean? That if you say, our intuition is, well, 1%. It's a pretty small volume. If I just took the neighbors in 1%, they're pretty close, so they should have the same label. Well, in dimension 10, it's everything. OK. So our intuition-- now you can say that probably there is something wrong with my way of thinking of volume, sure. But the problem is that we have to rethink a bit how you think of dimensions and similarity in high dimension, because things that are obvious low dimensional start to be very complicated. OK.

And the basic idea is that this neighbor technique just looks at what's happening in one region. But what you hope to do is that if your function actually has some kind of global properties-- so, say for example a sign is the simplest example of something which is global, because the value here and the value here are very much related. And then it goes up and it's the same. And then it goes down.

So if you know something like this, the idea is that you can borrow strength from points which are far away. In some sense the function has some similar properties. And so you want to go from a local estimation to some form of global estimation. OK. And instead of making a decision based only on the neighbors of the points, you might want to use points which are potentially far away. OK. And this seems to be like a good idea in high dimensions where the neighboring points might not give enough information . And that's kind of what's called, curse of dimensionality. OK.

So what I want to do next-- we can take a break here-- is discussing least squares and kernel least squares. OK. But what we're going to do is that we're going to take a linear model of our data, and then we are going to try to see how you can estimate and learn. And we're going to look at bit of the computation and a bit of the statistical idea underlying this model. And then we're going to play around in a very simple for way to extend from a linear model to a non-linear model and actually make it non-parametric. I'll tell you what non-parametric means.