PROFESSOR: I want to pick up with a little bit of overlap just to remind people where we were. We had been looking at clustering, and we looked at a fairly simple example of using agglomerative hierarchical clustering to cluster cities, based upon how far apart they were from each other cities. So, essentially, using this distance matrix, we could do a clustering that would reflect how close cities were to one another. And we went through a agglomerative clustering, and we saw that we would get a different answer, depending upon which linkage criterion we used.

This is an important issue because as one is using clustering, one what has to be aware that it is related to these things, and you choose the wrong linkage criterion, you might get an answer other than the most useful.

All right. I next went on and said, well, this is pretty easy, because when we're comparing the distance between two cities or the two features, we just subtract one distance from the other and we get a number. It's very straightforward. I then raised the question, suppose when we looked at cities, we looked at a more complicated way of looking at them than airline distance. So the first question, I said, well, suppose in addition to the distance by air, we add the distance by road, or the average temperature. Pick what you will. What do we do?

Well, the answer was we start by generalizing from a feature being a single number to the notion of a feature vector, where the features used to describe the city are now represented by a vector, typically of numbers. If the vectors are all in the same physical units, we could easily imagine how we might compare two vectors. So we might, for example, we look at the Euclidean distance between the two just by, say, subtracting one vector from the other.

However, if we think about that, it can be pretty misleading because, for example, when we look at a city, one element of the vector represents the distance in miles from another city, or in fact this case, the distance in miles to each city. And another represents temperatures. Well, it's kind of funny to compare distance, which might be thousands of miles, with the temperature which might be 5 degrees. A 5 degree difference in average temperature could be significant. Certainly a 20 degree difference in temperature is very significant, but a 20 mile difference in location might not be very significant.

And so to equally weight a 20 degree temperature difference and at 20 miles distance difference might give us a very peculiar answer. And so we have to think about the question of, how are we going to scale the elements of the vectors?

Even if we're in the same units, say inches, it can be an issue. So let's look at this example. Here I've got on the left, before scaling, something which we can say is in inches, height and width. This is not from a person, but you could imagine if you were trying to cluster people, and you measured their height in inches and their width in inches, maybe you don't want to treat them equally. Right? But there's a lot more variance in height than in width, or maybe there is and maybe there isn't.

So here on the left we don't have any scaling, and we see a very natural clustering. On the other hand, we notice on the y-axis the values range from not too far from 0 to not too far from 1. Whereas on the x-axis, the dynamic range is much less, not too far from 0 to not too far from 1/2. So we have twice the dynamic range here than we have here. Therefore, not surprisingly, when we end up doing the clustering, width plays a very important role. And we end up clustering it this way, dividing it along here.

On the other hand, if I take exactly the same data and scale it, and now the x-axis runs from 0 to 1/2 and the y-axis, roughly again, from 0 to 1, we see that suddenly when we look at it geometrically, we end up getting a very different look of clustering. What's the moral? The moral is you have to think hard about how to cluster your features, about how to scale your features, because it can have a dramatic influence on your answer. We'll see some real life examples of this shortly.

But these are all the important things to think about, and they all, in some sense, tie up into the same major point. Whenever you're doing any kind of learning, including clustering, feature, selection, and scaling is critical. It is where most of the thinking ends up going. And then the rest gets to be fairly mechanical.

How do we decide what features to use and how to scale them? We do that using domain knowledge. So we actually have to think about the objects that we're trying to learn about and what the objective of the learning process is. So continuing, how do we do the scaling? Most of the time, it's done using some variant of what's called the Minkowski metric. It's not nearly as imposing as it looks.

So the distance between two vectors, X1 and X2, and then we use p to talk about, essentially, the degree we're going to be using. So we take the absolute difference between each element of X1 and X2, raise it to the p-th power, sum them and then take the 1 over p.

Not very complicated, so let's say p is 2. That's the one you people are most familiar with. Effectively, all we're doing is getting the Euclidean distance. What we looked at when we looked at the mean squared distance between two things, between our errors and our measured data, between our measured data and our predicted data. We used the mean square error. That's essentially in Minkowski distance with p equal to 2.

That's probably the most commonly used, but an almost equally commonly used sets p equal to 1, and that's something called the Manhattan distance. I suspect at least some of you have spent time walking around Manhattan, a small but densely populated island in New York. And midtown Manhattan has the feature that it's laid out in a grid. So what you have is a grid, and you have the avenues running north-south and the streets running east-west. And if you want to walk from, say, here to here or drive from here to here, you cannot take the diagonal because there are a bunch of buildings in the way. And so you have to move either left or right, or up, or down.

That's the Manhattan distance between two points. This is used, in fact, for a lot of problems, typically when somebody is comparing the distance between two genes, for example, they use a Manhattan metric rather than a Euclidean metric to say how similar two things are. Just wanted to show that because it is something that you will run across in the literature when you read about these kinds of things.

All right. So far, we've talked about issues where things are comparable. And we've been doing that by representing each element of the feature vector as a floating point number. So we can run a formula like that by subtracting one from the other. But we often, in fact, have to deal with nominal categories, things that have names rather than numbers.

So for clustering people, maybe we care about eye color, blue, brown, gray, green. Hair color. Well, how do you compare blue to green? Do you subtract one from the other? Kind of hard to do. What does it mean to subtract green from blue? Well, I guess we could talk about it in the frequency domain, enlighten things. Typically, what we have to do in that case is, we convert them to a number and then have some ways to relate the numbers.

Again, this is a place where domain knowledge is critical. So, for example, we might convert blue to 0, green to 0.5, and brown to 1, thus indicating that we think blue eyes are closer to green eyes than they are to brown eyes. I don't know why we think that but maybe we think that. Red hair is closer to blonde hair than it is to black hair. I don't know.

These are the sorts of things that are not mathematical questions, typically, but judgments that people have to make. Once we've converted things to numbers, we then have to go back to our old friend of scaling, which is often called normalization. Very often we try and contrive to have every feature range between 0 and 1, for example, so that everything is normalized to the same dynamic range, and then we can compare. Is that the right thing to do? Not necessarily, because you might consider some features more important than others and want to give them a greater weight. And, again, that's something we'll come back to and look at.

All this is a bit abstract. I now want to look at an example. Let's look at the example of clustering mammals. There are, essentially, an unbounded number of features you could use, size at birth, gestation period, lifespan, length of tail, speed, eating habits. You name it. The choice of features and weighting will, of course, have an enormous impact on what clusters you get. If you choose size, humans might appear in one cluster. If you choose eating habits, they might appear in another.

How should you choose which features you want? You have to begin by choosing, thinking about the reason you're doing the clustering in the first place. What is it you're trying to learn about the mammals? As an example, I'm going to choose the objective of eating habits. I want to cluster mammals somehow based upon what they eat. But I want to do that, and here's a very important thing about what we often see in learning without any direct information about what they eat. Typically, when we're using machine learning, we're trying to learn about something for which we have limited or no data. Remember when we talked about learning, I talked about learning in which it was supervised, and which we had some data, and unsupervised, in which, essentially, we don't have any labels.

So let's say we don't have any labels about what mammals eat, but we do know a lot about the mammals themselves. And, in fact, the hypothesis I'm going to start with here is that you can infer people's or creatures' eating habits from their dental records, or their dentitian. But over time, we have evolved, all creatures have evolved, to have teeth that are related to what they eat, we can see.

So I managed to procure a database of dentitian for various mammals. There's the laser pointer. So what I've got here is the number of different kinds of teeth. So the right top incisors, the right bottom incisors, molars, et cetera, pre-molars. Don't worry if you don't know about teeth very much. I don't know very much. And then for each animal, I have the number of each kind of tooth. Actually, I don't have it for this particular mammal, but these two I do. I don't even remember what they are. They're cute.

All right. So I've got that database, and now I want to try and see what happens when I cluster them. The code to do this is not very complicated, but I should make a confession about it. Last night, I won't say I learned it. I was reminded of a lesson that I've often preached in 6.00, is that it's not good to get your programming done at the last minute. So as I was debugging this code at 2:00 and 3:00 in the morning today, I was realizing how inefficient I am at debugging at that hour. Maybe for you guys that's the shank of the day. For me, it's too late. I think it all works, but I was certainly not at my best as I was debugging last night.

All right. But at the moment, I don't want you to spend time working on the code itself. I would like you to think a little bit about the overall class structure of the code, which I've got on the first page of the handout. So at the bottom of my hierarchy, I've got something called a point, and that's an abstraction of the things to be clustered. And I've done it in quite a generalized way, because, as you're going to see, the code we're looking at today, I'm going to use not only for clustering mammals but for clustering all sorts of other things as well. So I decided to take the trouble of building up a set of classes that would be useful.

And in this class, I can have the name of a point, its original attributes. That say its original feature vector, an unscaled feature vector, and then whether or not I choose to normalize it. I might have normalized features as well. Again, I don't want you worry too much about the details of the code. And then I have a distance metric, and I'm just for the moment using simple Euclidean distance.

The next element in my hierarchy, not yet a hierarchy-- it's still flat-- is a cluster. And so what a cluster is, you can think of it as, at some abstract level, it's just going to be a set of points, the points that are in the cluster. But I've got some other operations on it that will be useful. I can compute the distance between two clusters, and as you'll see, I have single linkage, Mac Link, max , average, the three I talked about last week.

And also this notion of a centroid. We'll come back to that when we get to k-means clustering. We don't need to worry right now about what that is.

Then I'm going to have a cluster set. That's another useful data abstraction. And that's what you might guess from its name, just a set of clusters. The most interesting operation there is merge. As you saw, when we looked at hierarchical clustering last week, the key step there is merging two clusters. And in doing that, I'm going to have a function called Find Closest, which given a metric and a cluster, finds the cluster that is most similar to that, to self, because as you, again, will recall from hierarchical clustering, that's what I merged at each step is the two most similar clusters. And then there's some details about how it works, which again, we don't need to worry about at the moment.

And then I'm going to have a subclass of point called Mammal, in which I will represent each mammal by the dentitian as we've looked at before.

Then pretty simply, we can do a bunch of things with it.

Before we look at the other details of the code, I want to now run it and see what we get. So I'm just going to use hierarchical clustering now to cluster the mammals based upon this feature vector, which will be a list of numbers showing how many of each kind of tooth the mammals have. Let's see what we get. So it's doing the merging.

So we can see the first step, it merged beavers with ground hogs and it merged grey squirrels with porcupines, wolves and bears. Various other kinds of things, like jaguars and cougars, were a lot alike. Eventually, it starts doing more complicated merges. It merges a cluster containing only the river otter with one containing a Martin and a wolverine, beavers and ground hogs with squirrels and porcupines, et cetera. And at the end, I had it stop with two clusters. It came up with these clusters.

Now we can look at these clusters and say, all right. What do we think? Have we learned anything interesting? Do we see anything in any of these-- do we think it makes sense? Remember, our goal was to cluster mammals based upon what they might eat. And we can ask, do we think this corresponds to that?

No. All right. Who-- somebody said-- Now, why no? Go ahead.

AUDIENCE: We've got-- like a deer doesn't eat similar things as a dog. And we've got one type on the top cluster and a different kind of bat in the bottom cluster. Seems like they would be even closer together.

PROFESSOR: Well, sorry. Yeah. A deer doesn't eat what a dog eats, and for that matter, we have humans here, and while some human are by choice vegetarians, genetically, humans are essentially carnivores. We know that. We eat meat. And here we are with a bunch of herbivores, typically. Things are strange. By the way, bats might end up being in ones, because some bats eat fruit, other bat eat insects, but who knows? So I'm not very happy. Why do you think we got this clustering that maybe isn't helping us very much?

Well, let's go look at what we did here. Let's look at test 0. So I said I wanted two clusters. I don't want it to print all the steps along the way. I'm going to print the history at the end. And scaling is identity. Well, let's go back and look at some of the data here. What we can see is-- or maybe we can't see too quickly, looking at all this-- is some kinds of teeth have a relatively small range. Other kinds of teeth have a big range. And so, at the moment, we're not doing any normalization, and maybe what we're doing is getting something distorted where we're only looking at a certain kind of tooth because it has a larger dynamic range.

And in fact, if we look at the code, we can go back up and let's look at Build Mammal Points and Read Mammal Data. So Build Mammal Points calls Read Mammal Data, and then builds the points. So Read Mammal Data is the interesting piece. And what we can see here is, as we read it in-- this is just simply reading things in, ignoring comments, keeping track of things-- and then we come down here, I might do some scaling.

So Point.Scale feature is using the scaling argument. Where's that coming from? If we look at Mammal Teeth, here from the mammal class, we see that there are two ways to scale it, identity, where we just multiply every element in the vector by 1. That doesn't change anything. Or what I've called 1 over max. And here, I've looked at the maximum number of each kind of tooth and I'm dividing 1 by that. So here we could have up to three of those. Here we could have four of those. We could have six of this kind of tooth, whatever it is. And so we can see, by dividing by the max, I'm now putting all of the different kinds of teeth on the same scale. I'm normalizing. And now we'll see, does that make a difference?

Well, since we're dividing by 6 here and 3 here, it certainly could make a difference. It's a significant scaling factor, 2X. So let's go and change the code, or change the test. And let's look at Test 0-- 0, not "O"-- with scale set to 1 over max. You'll notice, by the way, that rather than using some obscure code, like scale equals 12, I use strings so I remember what they are. It's, I think, a pretty useful programming trick. Whoops. Did I use the wrong name for this? Should be scaling?

So off it's going. Now we get a different set of things, and as far as I know, once we've scaled things, we get what I think is a much more sensible pair, where I think what we essentially have is the herbivores down here, and the carnivores up here. Ok. I don't care how much you know about teeth. The point is scaling can really matter. You have to look at it, and you have to think about what you're doing. And the interesting thing here is that without any direct evidence about what mammals eat, we are able to use machine learning, clustering in this case, to infer a new fact that we have some mammals that are similar in what they eat, and some mammals that are also similar, some groups. Now I can't infer from this herbivores versus carnivores because I didn't have any labels to start with. But what I can infer is that, whatever they eat, there's something similar about these animals, and something similar about these animals. And there's a difference between the groups in C1 and the groups in C0. I can then go off and look at some points in each of these and then try and figure out how to label them later.

OK, let's look at a difference data set, a far more interesting one, a richer one. Now, let's not look at that version of it. That's too hard to read. Let's look at the Excel spreadsheet. So this is a database I found online of every county in the United States, and a bunch of features about that county. So for each county in the United States, we have its name. The first part of the name is the state it's in. The second part of the name is the name of the county, and a bunch of things, like the average value of homes, how much poverty, its population density, its population change, how many people are over 65, et cetera. So the thing I want you to notice, of course, is while everything is a number, the scales are very different. It's a big difference between the percent of something, which will go between 0 and 100, and the population density, which ranges over a much larger dynamic range. So we can immediately suspect that scaling is going to be an issue here.

So we now have a bunch of code that we can use that I've written to process this. It uses the same clusters that we have here, except I've added a kind of Point called the County. Looks very different from a mammal, but the good news is I got to reuse a lot of my code. Now let's run a test. We'll go down here to Test 3, and we'll see whether we can do hierarchical clustering of the counties. Whoops. Test 3 wants the name of what we're doing. So we'll give it the name. It's Counties.Text. I just exported the spreadsheet as a text file. Well, we can wait a while for this, but I'm not going to. Let's think about what we know that hierarchical clustering and how long this is likely to take. I'll give you a hint. There are approximately 3,100 counties in the United States. I'll bet none of you could have guessed that number. How many comparisons do we have to find the two counties that are most similar to each other? Comparing each county with every other county, how many comparisons is that going to be? Yeah.

AUDIENCE: It's 3,100 choose 2.

PROFESSOR: Right. So that will be 3,100 squared. Thank you. And that's just the first step in the cluster.

To perform the next merge, we'll have to do it again. So in fact, as we've looked at last time, it's going to be a very long and tedious process, and one I'm not going to wait for. So I'm going to interrupt and we're going to look at a smaller example.

Here I've just got only the counties in New England, a much smaller number than 3,100, and I'm going to cluster them using the exact same clustering code we used for the mammals. It's just that the points are now counties instead of mammals. And we got two clusters. Middlesex County in Massachusetts happens to be the county in which MIT is located. And all the others-- well, you know, MIT is a pretty distinctive place. Maybe that's what did it. I don't quite think so. Someone got a hypothesis about why we got this rather strange clustering? And is it because Middlesex contains MIT and Harvard both?

This really surprised me, by the way, when I first ran it. I said, how can this be? So I went and I started looking at the data, and what I found is that Middlesex County has about 600,000 more people than any other county in New England. Who knew? I would have guessed Suffolk, where Boston is, was the biggest county. But, in fact, Middlesex is enormous relative to every other county in New England.

And it turns out that difference of 600,000, when I didn't scale things, just swamped everything else. And so all I'm really getting here is a clustering that depends on the population. Middlesex is big relative to everything else and, therefore, that's what I get. And it ignores things like education level and housing prices, and all those other things because the differences are small relative to 600,000.

Well, let's turn scaling on. To do that, I want to show you how I do this scaling. I did not, given the number of features and number of counties, do what I did for mammals and count them by hand to see what the maximum was. I decided it would be a lot faster even at 2:00 in the morning to write code to do it. So I've got some code here. I've got Build County Points, just like Build Mammal Points and Read County Data, like Read Mammal Data. But the difference here is, along the way, as I'm reading in each county, I'm keeping track of the maximum for each feature. And then I'm just going to just do the scaling automatically. So exactly the one over max scaling I did for mammals' teeth, I'm going to do it for counties, but I've just written some code to automate that process because I knew I would never be able to count them.

All right, so now let's see what happens if we run it that way. Test 3, New England, and Scale equals True. I'm either scaling it or not, is the way I wrote this one. And with the scaling on again, I get a very different set of clusters. What have we got? Where's Middlesex? It's in one of these 2 clusters. Oh, here it is. It's C0. But it's with Fairfield, Connecticut and Hartford, Connecticut and Providence, Rhode Island.

It's a different answer. Is it a better answer? It's not a meaningful question, right? It depends what I'm trying to infer, what we hope to learn from the clustering, and that's a question we're going to come back to on Tuesday in some detail with the counties, and look at how, by using different kinds of scaling or different kinds of features, we can learn different things about the counties in this country.

Before I do that, however, I want to move away from New England. Remember we're focusing on New England because it took too long to do hierarchical clustering of 3,100 counties. But that's what I want to do. It's no good to just say, I'm sorry. It took too long. I give up. Well, the good news is there are other clustering mechanisms that are much more efficient. We'll later see they, too, have their own faults.

But we're going to look at k-means clustering, which has the big advantage of being fast enough that we can run it on very big data sets. In fact, it is roughly linear in the number of counties. And as we've seen before, when n gets very large, anything that's worse than linear is likely to be ineffective.

So let's think about how k-means works. Step one, is you choose k. k is the total number of clusters you want to have when you're done. So you start by saying, I want to take the counties and split them into k-clusters. 2 clusters, 20 clusters, a 100 clusters, 1,000 clusters. You have to choose k in the beginning. And that it's one of the issues that you have with k-means clustering is, how do you choose k? We can talk about that later.

Once I've chosen k, I choose k points as initial centroids. You may remember earlier today we saw this centroid method in class cluster. So what's a centroid? You've got a cluster, and in the clusters, you've got a bunch of points scattered around. The centroid you can think of as, quote, "the average point," the center of the cluster. The centroid need not be any of the points in the cluster. So, again, you need some metric. But let's say we're using Euclidean. It's easy to see on the board. The centroid is kind of there.

Now let's assume that we're going to start by choosing k-point from the initial set and labeling each of them as a centroid. We often-- in fact, quite typically-- choose these at random. So we now have k randomly chosen points, each of which we're going to call centroid. The next step is to assign each point to the nearest centroid. So we've got k-centroids. We usually choose a small k, say 50. And now we have to compare each of the 3,100 counties to each of the 50 centroids, and put each one in the correct thing, in the closest. So it's 50 times 3,100, which is a lot smaller number than 3,100 squared.

So now I've got a clustering. Kind of strange, because what it looks like depends on this random choice. So there's no reason to expect that the initial assignment will give me anything very useful.

Step (4) is, for each of the k-clusters, choose a new centroid. Now remember, I just chose at random k-centroids. Now I actually have a cluster with a bunch of points in it, so I could, for example, take the average of those and compute a centroid. And I can either take the average, or I can take the point nearest the average. It doesn't much matter.

And then step (5) is one we've looked at before, assign each point to nearest centroid. So now I'm going to get a new clustering.

And then, (6) is repeat (4) and (5) until the change is small. So each time I do step (5), I can keep track of how many points I've moved from one cluster to another. Or each time I do step (4), I can say how much have I moved the centroids? Each of those gives me a measure of how much change the new iteration has produced. When I get to the point where the iterations are not making much of a change-- and we'll see what we might mean by that-- we stop and say, OK, we now have a good clustering.

So if we think of the complexity each iteration is order k-n, where k is the number of clusters, and n is the number of points. And then we do that step for some number of iterations. So if the number of iterations is small, it will converge quite quickly. And as we'll see, typically for k-means, we don't need a lot of iterations to get an answer. It's typically not proportional to n, in particular, which is very important.

All right. Tuesday, we'll go over the code for k-means clustering, and then have some fun playing with counties and see what we can learn about where we live. All right. Thanks a lot.