JOHN GUTTAG: I'm a little reluctant to say good afternoon, given the weather, but I'll say it anyway. I guess now we all do know that we live in Boston. And I should say, I hope none of you were affected too much by the fire yesterday in Cambridge, but that seems to have been a pretty disastrous event for some.

Anyway, here's the reading. This is a chapter in the book on clustering, a topic that Professor Grimson introduced last week. And I'm going to try and finish up with respect to this course today, though not with respect to everything there is to know about clustering.

Quickly just reviewing where we were. We're in the unit of a course on machine learning, and we always follow the same paradigm. We observe some set of examples, which we call the training data. We try and infer something about the process that created those examples. And then we use inference techniques, different kinds of techniques, to make predictions about previously unseen data. We call that the test data.

As Professor Grimson said, you can think of two broad classes. Supervised, where we have a set of examples and some label associated with the example-- Democrat, Republican, smart, dumb, whatever you want to associate with them-- and then we try and infer the labels. Or unsupervised, where we're given a set of feature vectors without labels, and then we attempt to group them into natural clusters. That's going to be today's topic, clustering.

So clustering is an optimization problem. As we'll see later, supervised machine learning is also an optimization problem. Clustering's a rather simple one.

We're going to start first with the notion of variability. So this little c is a single cluster, and we're going to talk about the variability in that cluster of the sum of the distance between the mean of the cluster and each example in the cluster. And then we square it. OK? Pretty straightforward.

For the moment, we can just assume that we're using Euclidean distance as our distance

metric. Minkowski with p equals two. So variability should look pretty similar to something we've seen before, right? It's not quite variance, right, but it's very close. In a minute, we'll look at why it's different.

And then we can look at the dissimilarity of a set of clusters, a group of clusters, which I'm writing as capital C, and that's just the sum of all the variabilities. Now, if I had divided variability by the size of the cluster, what would I have? Something we've seen before. What would that be? Somebody? Isn't that just the variance?

So the question is, why am I not doing that? If up til now, we always wanted to talk about variance, why suddenly am I not doing it? Why do I define this notion of variability instead of good old variance? Any thoughts?

What am I accomplishing by not dividing by the size of the cluster? Or what would happen if I did divide by the size of the cluster? Yes.

AUDIENCE: You normalize it?

JOHN GUTTAG: Absolutely. I'd normalize it. That's exactly what it would be doing. And what might be good or bad about normalizing it?

What does it essentially mean to normalize? It means that the penalty for a big cluster with a lot of variance in it is no higher than the penalty of a tiny little cluster with a lot of variance in it. By not normalizing, what I'm saying is I want to penalize big, highly-diverse clusters more than small, highly-diverse clusters. OK? And if you think about it, that probably makes sense. Big and bad is worse than small and bad.

All right, so now we define the objective function. And can we say that the optimization problem we want to solve by clustering is simply finding a capital C that minimizes dissimilarity? Is that a reasonable definition? Well, hint-- no. What foolish thing could we do that would optimize that objective function? Yeah.

AUDIENCE: You could have the same number of clusters as points?

JOHN GUTTAG: Yeah. I can have the same number of clusters as points, assign each point to its own cluster, whoops. Ooh, almost a relay. The dissimilarity of each cluster would be 0. The variability would be 0, so the dissimilarity would be 0, and I just solved the problem. Well, that's clearly not a very useful thing to do.

So, well, what do you think we do to get around that? Yeah.

AUDIENCE: We apply a constraint?

JOHN GUTTAG: We apply a constraint. Exactly. And so we have to pick some constraint.

What would be a suitable constraint, for example? Well, maybe we'd say, OK, the clusters have to have some minimum distance between them. Or-- and this is the constraint we'll be using today-- we could constrain the number of clusters. Say, all right, I only want to have at most five clusters. Do the best you can to minimize dissimilarity, but you're not allowed to use more than five clusters. That's the most common constraint that gets placed in the problem.

All right, we're going to look at two algorithms. Maybe I should say two methods, because there are multiple implementations of these methods. The first is called hierarchical clustering, and the second is called k-means. There should be an S on the word mean there. Sorry about that.

All right, let's look at hierarchical clustering first. It's a strange algorithm. We start by assigning each item, each example, to its own cluster. So this is the trivial solution we talked about before. So if you have N items, you now have N clusters, each containing just one item.

In the next step, we find the two most similar clusters we have and merge them into a single cluster, so that now instead of N clusters, we have N minus 1 clusters. And we continue this process until all items are clustered into a single cluster of size N.

Now of course, that's kind of silly, because if all I wanted to put them all it in is in a single cluster, I don't need to iterate. I just go wham, right? But what's interesting about hierarchical clustering is you stop it, typically, somewhere along the way. You produce something called a [? dendogram. ?] Let me write that down.

At each step here, it shows you what you've merged thus far. We'll see an example of that shortly. And then you can have some stopping criteria. We'll talk about that. This is called agglomerative hierarchical clustering because we start with a bunch of things and we agglomerate them. That is to say, we put them together.

All right? Make sense? Well, there's a catch. What do we mean by distance? And there are multiple plausible definitions of distance, and you would get a different answer depending upon which measure you used. These are called linkage metrics.

The most common one used is probably single-linkage, and that says the distance between a pair of clusters is equal to the shortest distance from any member of one cluster to any member of the other cluster. So if I have two clusters, here and here, and they have bunches of points in them, single-linkage distance would say, well, let's use these two points which are the closest, and the distance between these two is the distance between the clusters.

You can also use complete-linkage, and that says the distance between any two clusters is equal to the greatest distance from any member to any other member. OK? So if we had the same picture we had before-- probably not the same picture, but it's a picture. Whoops. Then we would say, well, I guess complete-linkage is probably the distance, maybe, between those two.

And finally, not surprisingly, you can take the average distance. These are all plausible metrics. They're all used and practiced for different kinds of results depending upon the application of the clustering.

All right, let's look at an example. So what I have here is the air distance between six different cities, Boston, New York, Chicago, Denver, San Francisco, and Seattle. And now let's say we're-- want to cluster these airports just based upon their distance.

So we start. The first piece of our [? dendogram ?] says, well, all right, I have six cities, I have six clusters, each containing one city. All right, what happens next? What's the next level going to look like? Yeah?

AUDIENCE: You're going from Boston [INAUDIBLE]

JOHN GUTTAG: I'm going to join Boston and New York, as improbable as that sounds. All right, so that's the next level. And if for some reason I only wanted to have five clusters, well, I could stop here.

Next, what happens? Well, I look at it, I say well, I'll join up Chicago with Boston and New York. All right. What do I get at the next level? Somebody? Yeah.

AUDIENCE: Seattle [INAUDIBLE]

JOHN GUTTAG: Doesn't look like it to me. If you look at San Francisco and Seattle, they are 808 miles, and Denver and San Francisco is 1,235. So I'd end up, in fact, joining San Francisco and Seattle.

AUDIENCE: That's what I said.

JOHN GUTTAG: Well, that explains why I need my hearing fixed.

[LAUGHTER]

All right. So I combine San Francisco and Seattle, and now it gets interesting. I have two choices with Denver. Obviously, there are only two choices, and which I choose depends upon which linkage criterion I use. If I'm using single-linkage, well, then Denver gets joined with Boston, New York, and Chicago, because it's closer to Chicago than it is to either San Francisco or Seattle.

But if I use complete-linkage, it gets joined up with San Francisco and Seattle, because it is further from Boston than it is from, I guess it's San Francisco or Seattle. Whichever it is, right? So this is a place where you see what answer I get depends upon the linkage criteria.

And then if I want, I can consider to the next step and just join them all. All right? That's hierarchical clustering. So it's good because you get this whole history of the [? dendograms, ?] and you get to look at it, say, well, all right, that looks pretty good. I'll stick with this clustering.

It's deterministic. Given a linkage criterion, you always get the same answer. There's nothing random here. Notice, by the way, the answer might not be optimal with regards to that linkage criteria. Why not? What kind of algorithm is this?

AUDIENCE: Greedy.

JOHN GUTTAG: It's a greedy algorithm, exactly. And so I'm making locally optimal decisions at each point which may or may not be globally optimal.

It's flexible. Choosing different linkage criteria, I get different results. But it's also potentially really, really slow. This is not something you want to do on a million examples. The naive algorithm, the one I just sort of showed you, is N cubed. N cubed is typically impractical.

For some linkage criteria, for example, single-linkage, there exists very clever N squared algorithms. For others, you can't beat N cubed. But even N squared is really not very good. Which gets me to a much faster greedy algorithm called k-means.

Now, the k in k-means is the number of clusters you want. So the catch with k-means is if you don't have any idea how many clusters you want, it's problematical, whereas hierarchical, you

get to inspect it and see what you're getting. If you know how many you want, it's a good choice because it's much faster.

All right, the algorithm, again, is very simple. This is the one that Professor Grimson briefly discussed. You randomly choose k examples as your initial centroids. Doesn't matter which of the examples you choose. Then you create k clusters by assigning each example to the closest centroid, compute k new centroids by averaging the examples in each cluster.

So in the first iteration, the centroids are all examples that you started with. But after that, they're probably not examples, because you're now taking the average of two examples, which may not correspond to any example you have. Actually the average of N examples.

And then you just keep doing this until the centroids don't move. Right? Once you go through one iteration where they don't move, there's no point in recomputing them again and again and again, so it is converged.

So let's look at the complexity. Well, at the moment, we can't tell you how many iterations you're going to have, but what's the complexity of one iteration? Well, let's think about what you're doing here. You've got k centroids. Now I have to take each example and compare it to each-- in a naively, at least-- to each centroid to see which it's closest to. Right? So that's k comparisons per example. So that's k times n times d, where how much time each of these comparison takes, which is likely to depend upon the dimensionality of the features, right? Just the Euclidean distance, for example.

But this is a way small number than N squared, typically. So each iteration is pretty quick, and in practice, as we'll see, this typically converges quite quickly, so you usually need a very small number of iterations. So it is quite efficient, and then there are various ways you can optimize it to make it even more efficient. This is the most commonly-used clustering algorithm because it works really fast.

Let's look at an example. So I've got a bunch of blue points here, and I actually wrote the code to do this. I'm not going to show you the code. And I chose four centroids at random, colored stars. A green one, a fuchsia-colored one, a red one, and a blue one. So maybe they're not the ones you would have chosen, but there they are. And I then, having chosen them, assign each point to one of those centroids, whichever one it's closest to. All right? Step one.

And then I recompute the centroid. So let's go back. So we're here, and these are the initial

centroids. Now, when I find the new centroids, if we look at where the red one is, the red one is this point, this point, and this point. Clearly, the new centroid is going to move, right? It's going to move somewhere along in here or something like that, right? So we'll get those new centroids. There it is.

And now we'll re-assign points. And what we'll see is this point is now closer to the red star than it is to the fuchsia star, because we've moved the red star. Whoops. That one. Said the wrong thing. They were red to start with. This one is now suddenly closer to the purple, so-- and to the red. It will get recolored.

We compute the new centroids. We're going to move something again. We continue. Points will move around. This time we move two points. Here we go again. Notice, again, the centroids don't correspond to actual examples. This one is close, but it's not really one of them. Move two more. Recompute centroids, and we're done.

So here we've converged, and I think it was five iterations, and nothing will move again. All right? Does that make sense to everybody? So it's pretty simple.

What are the downsides? Well, choosing k foolishly can lead to strange results. So if I chose k equal to 3, looking at this particular arrangement of points, it's not obvious what "the right answer" is, right? Maybe it's making all of this one cluster. I don't know. But there are weird k's and if you choose a k that is nonsensical with respect to your data, then your clustering will be nonsensical. So that's one problem we have think about. How do we choose k?

Another problem, and this is one somebody raised last time, is that the results can depend upon the initial centroids. Unlike hierarchical clustering, k-means is non-deterministic. Depending upon what random examples we choose, we can get a different number of iterations. If we choose them poorly, it could take longer to converge.

More worrisome, you get a different answer. You're running this greedy algorithm, and you might actually get to a different place, depending upon which centroids you chose. So these are the two issues we have to think about dealing with.

So let's first think about choosing k. What often happens is people choose k using a priori knowledge about the application. If I'm in medicine, I actually know that there are only five different kinds of bacteria in the world. That's true. I mean, there are subspecies, but five large categories. And if I had a bunch of bacterium I wanted to cluster, may just set k equal to 5.

Maybe I believe there are only two kinds of people in the world, those who are at MIT and those who are not. And so I'll choose k equal to 2. Often, we know enough about the application, we can choose k. As we'll see later, often we can think we do, and we don't. A better approach is to search for a good k.

So you can try different values of k and evaluate the quality of the result. Assume you have some metric, as to say yeah, I like this clustering, I don't like this clustering. And we'll talk about do that in detail.

Or you can run hierarchical clustering on a subset of data. I've got a million points. All right, what I'm going to do is take a subset of 1,000 of them or 10,000. Run hierarchical clustering. From that, get a sense of the structure underlying the data. Decide k should be 6, and then run k-means with k equals 6. People often do this. They run hierarchical clustering on a small subset of the data and then choose k.

And we'll look-- but one we're going to look at is that one. What about unlucky centroids? So here I got the same points we started with. Different initial centroids. I've got a fuchsia one, a black one, and then I've got red and blue down here, which I happened to accidentally choose close to one another.

Well, if I start with these centroids, certainly you would expect things to take longer to converge. But in fact, what happens is this-- I get this assignment of blue, this assignment of red, and I'm done. It converges on this, which probably is not what we wanted out of this. Maybe it is, but the fact that I converged on some very different place shows that it's a real weakness of the algorithm, that it's sensitive to the randomly-chosen initial conditions.

Well, couple of things you can do about that. You could be clever and try and select good initial centroids. So people often will do that, and what they'll do is try and just make sure that they're distributed over the space. So they would look at some picture like this and say, well, let's just put my centroids at the corners or something like that so that they're far apart.

Another approach is to try multiple sets of randomly-chosen centroids, and then just select the best results. And that's what this little algorithm on the screen does. So I'll say best is equal to k-means of the points themselves, or something, then for t in range number of trials, I'll say C equals k-means of points, and I'll just keep track and choose the one with the least dissimilarity. The thing I'm trying to minimize. OK?

The first one is got all the points in one cluster. So it's very dissimilar. And then I'll just keep generating for different k's and I'll choose the k that seems to be the best, that does the best job of minimizing my objective function. And this is a very common solution, by the way, for any randomized greedy algorithm. And there are a lot of randomized greedy algorithms that you just choose multiple initial conditions, try them all out and pick the best.

All right, now I want to show you a slightly more real example. So this is a file we've got with medical patients, and we're going to try and cluster them and see whether the clusters tell us anything about the probability of them dying of a heart attack in, say, the next year or some period of time. So to simplify things, and this is something I have done with research, but we're looking at only four features here-- the heart rate in beats per minute, the number of previous heart attacks, the age, and something called ST elevation, a binary attribute.

So the first three are obvious. If you take an ECG of somebody's heart, it looks like this. This is a normal one. They have the S, the T, and then there's this region between the S wave and the T wave. And if it's higher, hence elevated, that's a bad thing. And so this is about the first thing that they measure if someone is having cardiac problems. Do they have ST elevation?

And then with each patient, we're going to have an outcome, whether they died, and it's related to the features, but it's probabilistic not deterministic. So for example, an older person with multiple heart attacks is at higher risk than a young person who's never had a heart attack. That doesn't mean, though, that the older person will die first. It's just more probable.

We're going to take this data, we're going to cluster it, and then we're going to look at what's called the purity of the clusters relative to the outcomes. So is the cluster, say, enriched by people who died? If you have one cluster and everyone in it died, then the clustering is clearly finding some structure related to the outcome.

So the file is in the zip file I uploaded. It looks more or less like this. Right? So it's very straightforward. The outcomes are binary. 1 is a positive outcome. Strangely enough in the medical jargon, a death is a positive outcome. I guess maybe if you're responsible for the medical bills, it's positive. If you're the patient, it's hard to think of it as a good thing. Nevertheless, that's the way that they talk. And the others are all there, right? Heart rate, other things.

All right, let's look at some code. So I've extracted some code. I'm not going to show you all of it. There's quite a lot of it, as you'll see. So we'll start-- one of the files you've got is called

cluster dot pi. I decided there was enough code, I didn't want to put it all in one file. I was getting confused. So I said, let me create a file that has some of the code and a different file that will then import it and use it. Cluster has things that are pretty much unrelated to this example, but just useful for clustering.

So an example here has name, features, and label. And really, the only interesting thing in it-- and it's not that interesting-- is distance. And the fact that I'm using Minkowski with 2 says we're using Euclidean distance.

Class cluster. It's a lot more code to that one. So we start with a non-empty list of examples. That's what init does. You can imagine what the code looks like, or you can look at it. Update is interesting in that it takes the cluster and examples and puts them in-- if you think of k-means in the cluster closest to the previous centroids and then returns the amount the centroid has changed. So if the centroid has changed by 0, then you don't have anything, right? Creates the new cluster.

And the most interesting thing is computeCentroid. And if you look at this code, you can see that I'm a slightly unreconstructed Python 2 programmers. I just noticed this. I really shouldn't have written 0.0. I should have just written 0, but in Python 2, you had to write that 0.0. Sorry about that. Thought I'd fixed these.

Anyway, so how do we compute the centroid? We start by creating an array of all 0s. The dimensionality is the number of features in the example. It's one of the methods from-- I didn't put up on the PowerPoint. And then for e in examples, I'm going to add to vals e.getFeatures, and then I'm just going to divide vals by the length of self.examples, the number of examples.

So now you see why I made it a pylab array, or a numpy array rather than a list, so I could do nice things like divide the whole thing in one expression. As you do math, any kind of math things, you'll find these arrays are incredibly convenient. Rather than having to write recursive functions or do bunches of iterations, the fact that you can do it in one keystroke is incredibly nice. And then I'm going to return the centroid.

Variability is exactly what we saw in the formula. And then just for fun, so you could see this, I used an iterator here. I don't know that any of you have used the yield statement in Python. I recommend it. It's very convenient.

One of the nice things about Python is almost anything that's built in, you can make your own

version of it. And so once I've done this, if c is a cluster, I can now write something like for c in big C, and this will make it work just like iterating over a list. Right, so this makes it possible to iterate over it. If you haven't read about yield, you probably should read the probably about two paragraphs in the textbook explaining how it works, but it's very convenient. Dissimilarity we've already seen.

All right, now we get to patients. This is in the file lec 12, lecture 12 dot py. In addition to importing the usual suspects of pylab and numpy, and probably it should import random too, it imports cluster, the one we just looked at. And so patient is a sub-type of cluster.Example.

Then I'm going to define this interesting thing called scale attributes. So you might remember, in the last lecture when Professor Grimson was looking at these reptiles, he ran into this problem about alligators looking like chickens because they each have a large number of legs. And he said, well, what can we do to get around this? Well, we can represent the feature as a binary number. Has legs, doesn't have legs. 0 or 1. And the problem he was dealing with is that when you have a feature vector and the dynamic range of some features is much greater than the others, they tend to dominate because the distances just look bigger when you get Euclidean distance.

So for example, if we wanted to cluster the people in this room, and I had one feature that was, say, 1 for male and 0 for female, and another feature that was 1 for wears glasses, 0 for doesn't wear glasses, and then a third feature which was weight, and I clustered them, well, weight would always completely dominate the Euclidean distance, right? Because the dynamic range of the weights in this room is much higher than the dynamic range of 0 to 1.

And so for the reptiles, he said, well, OK, we'll just make it a binary variable. But maybe we don't want to make weight a binary variable, because maybe it is something we want to take into account. So what we do is we scale it. So this is a method called z-scaling. More general than just making things 0 or 1.

It's a simple code. It takes in all of the values of a specific feature and then performs some simple calculations, and when it's done, the resulting array it returns has a known mean and a known standard deviation.

So what's the mean going to be? It's always going to be the same thing, independent of the initial values. Take a look at the code. Try and see if you can figure it out. Anybody want to take a guess at it? 0. Right? So the mean will always be 0. And the standard deviation, a little

harder to figure, but it will always be 1.

OK? So it's done this scaling. This is a very common kind of scaling called z-scaling. The other way people scale is interpolate. They take the smallest value and call it 0, the biggest value, they call it 1, and then they do a linear interpolation of all the values between 0 and 1. So the range is 0 to 1. That's also very common.

So this is a general way to get all of the features sort of in the same ballpark so that we can compare them. And we'll look at what happens when we scale and when we don't scale. And that's why my getData function has this parameter to scale. It either creates a set of examples with the attributes as initially or scaled.

And then there's k-means. It's exactly the algorithm I showed you with one little wrinkle, which is this part. You don't want to end up with empty clusters. If I tell you I want four clusters, I don't mean I want three with examples and one that's empty, right? Because then I really don't have four clusters.

And so this is one of multiple ways to avoid having empty clusters. Basically what I did here is say, well, I'm going to try a lot of different initial conditions. If one of them is so unlucky to give me an empty cluster, I'm just going to skip it and go on to the next one by raising a value error, empty cluster. And if you look at the code, you'll see how this value error is used.

And then try k-means. We'll call k-means numTrial times, each one getting a different set of initial centroids, and return the result with the lowest dissimilarity. Then I have various ways to examine the results. Nothing very interesting, and here's the key place where we're going to run the whole thing.

We'll get the data, initially not scaling it, because remember, it defaults to true. Then initially, I'm only going to try one k. k equals 2. And we'll call testClustering with the patients. The number of clusters, k. I put in seed as a parameter here because I wanted to be able to play with it and make sure I got different things for 0 and 1 and 2 just as a testing thing. And five trials it's defaulting to.

And then we'll look at testClustering is returning the fraction of positive examples for each cluster. OK? So let's see what happens when we run it.

All right. So we got two clusters. Cluster of size 118 with .3305, and a cluster of size 132 with a

positive fraction of point quadruple 3. Should we be happy? Does our clustering tell us anything, somehow correspond to the expected outcome for patients here? Probably not, right?

Those numbers are pretty much indistinguishable statistically. And you'd have to guess that the fraction of positives in the whole population is around .33, right? That about a third of these people died of their heart attack. And I might as well have signed them randomly to the two clusters, right? There's not much difference between this and what you would get with the random result.

Well, why do we think that's true? Because I didn't scale, right? And so one of the issues we had to deal with is, well, age had a big dynamic range, and, say, ST elevation, which I told you was highly diagnostic, was either 0 or 1. And so probably everything is getting swamped by age or something else, right?

All right, so we have an easy way to fix that. We'll just scale the data. Now let's see what we get. All right. That's interesting. With casting rule? Good grief. That caught me by surprise.

Good thing I have the answers in PowerPoint to show you, because the code doesn't seem to be working. Try it once more. No. All right, well, in the interest of getting through this lecture on schedule, we'll go look at the results that we get-- I got last time I ran it.

All right. When I scaled, what we see here is that now there is a pretty dramatic difference, right? One of the clusters has a much higher fraction of positive patients than others, but it's still a bit problematic. So this has pretty good specificity, or positive predictive value, but its sensitivity is lousy.

Remember, a third of our initial population more or less, was positive. 26 is way less than a third, so in fact I've got a class, a cluster, that is strongly enriched, but I'm still lumping most of the positive patients into the other cluster.

And in fact, there are 83 positives. Wrote some code to do that. And so we see that of the 83 positives, only this class, which is 70% positive, only has 26 in it to start with it. So I'm clearly missing most of the positives.

So why? Well, my hypothesis was that different subgroups of positive patients have different characteristics. And so we could test this by trying other values of k to see with-- we would get more clusters. So here, I said, let's try k equals 2, 4, and 6. And here's what I got when I ran

that.

So what you'll notice here, as we get to, say, 4, that I have two clusters, this one and this one, which are heavily enriched with positive patients. 26 as before in the first one, but 76 patients in the third one. So I'm now getting a much higher fraction of patients in one of the "risky" clusters.

And I can continue to do that, but if I look at k equals 6, we now look at the positive clusters. There were three of them significantly positive. But I'm not really getting a lot more patients total, so maybe 4 is the right answer.

So what you see here is that we have at least two parameters to play with, scaling and k. Even though I was only wanted a structure that would separate the risk-- high-risk patients from the lower-risk, which is why I started with 2, I later discovered that, in fact, there are multiple reasons for being high-risk. And so maybe one of these clusters is heavily enriched by old people. Maybe another one is heavily enriched by people who have had three heart attacks in the past, or ST elevation or some combination. And when I had only two clusters, I couldn't get that fine gradation.

So this is what data scientists spend their time doing when they're doing clustering, is they actually have multiple parameters. They try different things out. They look at the results, and that's why you actually have to think to manipulate data rather than just push a button and wait for the answer.

All right. More of this general topic on Wednesday when we're going to talk about classification. Thank you.