AUDIENCE: OK. Number (2) -- (2.3). It says if the code [INAUDIBLE] 0 would be the [INAUDIBLE]. I thought it was, you're generating random values for that?

PROFESSOR: Yeah, you were but if you look at what totes 0 is collecting, so if you look at the, where it draws a random number, j is indexing in totes, right? So when j is 0 your standard deviation, which is also being indexed by j, is going to be 0. So you're always going to get the same value. Any more questions? No? OK, that was easy.

So in lecture we were talking a lot about clustering, we've been talking about clustering the past, is it two lectures? And we had two different types of clustering methods, what were they?

AUDIENCE: Hierarchical and--

PROFESSOR: Heirarchical and K-means. Can someone give me a run down of what the steps are in hierarchical clustering?

AUDIENCE: Something that breaks everything down into one cluster [INAUDIBLE]

PROFESSOR: So let's say I have a bunch of data points. What would be the first step? You're going to first assign each point to a cluster. So each point gets its own cluster. And then the next step would be what?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Right, so you're going to find the two clusters that are closest to each other and merge them. So in this very contrived example, would be these guys. And then you're going to keep doing that until you get to a certain number of clusters, right? So you merge these two, then you might merge these two, and you might merge

these two, et cetera, et cetera, right?

AUDIENCE: [INAUDIBLE]

PROFESSOR: So you're going to set the number of clusters that you want at the outset, so I guess, for the mammalian teeth example, there was stopping criteria of two clusters, if I'm not mistaken. So let's take a look at the code here that implements a hierarchical cluster. So this is just some infrastructure code, it builds up the number of points. We have a cluster set class, which we'll go over in a second, and then we just add, for each point we're going to create a cluster object, and add it to the cluster set. Let's take a look at the cluster set.

Cluster set has one attribute, the members attribute, and it just has a set of points, or a set of clusters, actually. And the key method in here-- or the key methods are merge-1 and merge-n. Merge-n is what actually implements the clustering here. So you give it the distance metric that you're going to use for your points, the number of clusters that you want at the end of your clustering, the history tracker, and then you also tell it if you want to print out some debugging information.

So, which apparently is not used to this method, oh, now it is, merge-1. Anyway, so while we have more clusters than the number of clusters we desire, we're going to keep reiterating. And on each step we're going to call this function-- or method called merge-1 and just pass at the distance metric. And all merge-1 is going to do is it's going to, if there's only one-- if there's only one cluster here, then it's just going to return none.

If there are two clusters and its going to merge them, and if there are more than two clusters, it's going to find the closest, according to the distance metric and then merge those two. So then the return value is going to be the two clusters it merged. Let's look at the merge clusters code. All it does is it takes the two clusters and for each point, in both clusters, it adds it to a new list points and it creates a new cluster from those points. And then it removes two clusters from members and adds the newly created cluster. So then, find closest method. So what's this bit of code doing here?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Right, so we'll get to the metric in a second. So it initially finds, it looks at the first two members in the cluster set and it sets minDistance to be that, and it sets toMerge to be these two members. And then it narrates through every possible pair of clusters, in this cluster set and finds the minimum distance according to the metric. So let's look at the cluster class. All the cluster object does or class does is it holds a set of points. It knows the type of point that it's holding because that becomes important, when we talk about the different types of things that we want to cluster, and then it also has something called a centroid.

All a centroid is, is just the middle of the cluster, if you take all of the points and average their distances, or average their location. So these different functions, these just compute metrics about this particular cluster, right? So single linkage dist., all this is going to do is it finds a minimum distance between every pair of points in the cluster. And what does max linkage distance do? I'm sorry I'm mistaken it finds the minimum distance between this-- a point in this cluster and a point in another cluster, I misspoke. So what does max linkage distance do?

AUDIENCE: [INAUDIBLE]

PROFESSOR: The opposite. I have to keep you talking or you'll fall asleep. And then, averageLinkageDist? Same thing. This is why having meaningful function names is important, because it helps you explain code. So it also has this method in here called update and what update does, is it takes a new set of points and it sets the points that this cluster has, to be these new points. And then it computes a new centroid for this cluster. And the return value is the distance of the old centroid from the new centroid. And this becomes important in some of the algorithms. Then there's just some bookkeeping stuff here, like members will just give you all the points in this cluster. You all know what yield does, right?

AUDIENCE: [INAUDIBLE]

PROFESSOR: OK. So yield returns a generator object, which allows you to iterate over elements. So this was asked during the quiz review. What's the difference between range and x range? Right. So if I have a range of values it actually returns a different type. So I can print out this list, right? In this case, it will print out the type of object it is, so this is accomplished using yield.

So if I wanted to write this myself, what this is going to do is going to return something called a generator object. And all it does is, instead of holding all the numbers in memory, it's going to return them one at a time to me. So like when I use range here, it constructs a list and it has all of those integers in memory. If I use xrange it's not going to hold all the integers in memory, but I can still iterate over them one at a time.

AUDIENCE: So within that function [INAUDIBLE] yield a bunch of times before the function, right?

PROFESSOR: Yeah.

AUDIENCE: How is that accomplished? Does it operate [INAUDIBLE] within the way you normally have functions [INAUDIBLE]?

PROFESSOR: Right. So what this tells Python is that when it sees a yield, it's sort of like a return, except it's telling Python that I want to come back to this location at some point. So it a return just returns out of the function completely. What a yield does is it takes the value that is specified after yield, and it returns to the calling place in the program that value. But then, when it comes time to get a new value, it'll return back to where this yield exited.

So kind of a way maybe seeing this is if I iterate over my xrange. Each time it needs new value, it's going to go back inside this function and grab it. So it looks like a function. But what it's actually doing is creating what's called a generator object. And it has these special methods for getting the next value. So it's some nice syntactic sugar. But it's pretty neat. But that's what's going on with this yield statement here is that instead of returning the entire list of points, or instead of doing it in some other

way, all it's doing is just yielding each point one at a time so that you can iterate over them.

So what else? So here's a method for computing the centroid. All we are going to do is total up where each point is. And then, take the average over all the points. Does that makes sense? All right.

So the example we saw was mammal teeth. And the way that that's accomplished in this set of code is we're going to define a sub-class of a class, point, that's call mammal. And what point does is it has a name for a given data point. It has a set of attributes. And then, you can also give it some normalized attributes.

If you don't give it the normalized attributes, then it'll just use the original attributes. So it becomes important when we talk about-- when we do scaling over data, which we'll do shortly. So there's nothing really special about it except for this distance function. It's just defining the Euclidean distance for a given multi-dimensional point. So everyone knows that if you have a point in two dimensions, then if it's an xy point, then it's just x-squared plus y-squared -- square root of. It generalizes to higher dimensions if you weren't already aware. So if I want to find the straight line distance between a point in 3D, it's just going to be x-squared plus y-squared plus z-squared -- square root. That's all. And then, so on and so forth.

So all this does is it sub-classes point. And it has this function, scaleFeatures. And what scaleFeatures does is it'll take a key. And in this case, we have to find two ways of scaling this data, of scaling this point. So we have the identity, which is it's just going to leave every point alone. And then, we have this 1 over max, which is going to scale each attribute by the maximum value in this data set.

And if we look at the data set, we know that our max value is 6. So you could do that automatically. But in this case, we're using prior knowledge of the data set that we have.

So why don't we do a cluster? So this is going to do a hierarchical cluster, right? And what we're going to ask, if I just specify the default parameters, all it's going to

do is it's going to look for two clusters. It's going to use the identity. And it's just going to print out the history, like when it's performed the different merges. Unless I have extraneous code that I'm already running.

So what starts off first is we get a lot of merges with just these single element clusters, right? So I have a beaver with a groundhog, so I guess they're pretty similar in terms of teeth. We have a squirrel with a porcupine, wolf with a bear. And so eventually, though, we start finding clusters-- a wolf and a bear, I guess, are more similar. But they're also similar with a dog. So we're going to start merging multi-point clusters.

So we start seeing to beaver and groundhog cluster is going to get merged with the squirrel and the porcupine cluster. So if you were to visualize this, the reason why it's called hierarchical clustering is-- which one did I say? Beaver, groundhog-- these guys have been merged into a cluster, right? They started out as their own cluster. And they've been merged into their own cluster. And then, the grey squirrel and the porcupine, same thing. They started off with their own clusters at the beginning. They got merged.

And now, what this step is saying is that these two clusters get merged. So we're building this tree, or hierarchy. That's where the hierarchical comes from.

So we use hierarchical clustering a lot in other fields. So in speech recognition, we can do a hierarchical clustering of speech sounds. So if I have say different vowels, and maybe a couple of consonants, I would expect to see, say, these kind of clustered together first. And so what I might see is these would be fricatives. But then, I might have some stops, like "t" and "b" that get merged first.

So it's a way of making these generalized groupings at different levels. I don't know. Does anyone have any real questions about hierarchical clustering? So should I move on to k-means? All right.

So what's the general idea with k-means? So I start off with a set of data points. What's my first step?

AUDIENCE: Choose your total number of clusters?

PROFESSOR: Right, so I'm going to choose a k. So let's say for giggles we're going to choose k equals 3. And then, what's my next step?

AUDIENCE: Choose k's [INAUDIBLE]?

PROFESSOR: So we're going to pick k random points from our data set. All right, and then, what do I do?

AUDIENCE: Cluster?

PROFESSOR: Then you'd cluster. Yeah, all right. So after we've chosen our three centroids here, these become our clusters, right? And we're going to look at each point. And we're going to figure out which cluster they're closest to.

So in this case, this is going to be a pretty easy clustering. So all these points are going to belong here. All these points are going to belong here. All these points are going to belong here, right? And then, we're going to update our centroid for each of these clusters.

And there's going to be a distance that the centroid moves each time we update it. So in this case, the centroid moved quite a bit, right? Then, we're going to find the maximum distance that the centroid moved. And if it's below a certain cut off value, then we're going to say, I've got a good enough clustering. If it's above a certain cut off value, then what I'm going to say is like, this centroid moved quite a bit for this cluster, right?

So I'm going to try another iteration. I'm going to say, for each one of these points, I'm now going to look and try to the closest cluster that it belongs to based on these new centroids. And in this case, nothing's really going to change. So all of the deltas, all of the centroids, are going to stay the same. So it's going to be below the cut off value. And it's going to stop.

So what's an advantage of k-means over hierarchical clustering?

AUDIENCE: More efficient?

PROFESSOR: Yeah. So let's say that I have a million points. If I were to hierarchically cluster these, that means I'd start off with a million clusters. And now, in each iteration, I'm just going to reduce it by 1. I don't know, let's say 3, OK? So on each iteration, we're just reducing it by 1, which, if that's all we were doing, would not be so hard. It doesn't take too long to count down from a million on a computer.

But on each one of these steps, we have to compute the pairwise distance between each cluster. So it's going to be n times n minus 1 comparisons on each step which, in this first case, works out to a lot. And it doesn't get much better. So approximately, right? And it doesn't get much better as we go down.

With k-means, what happens is we just have to perform, on each step, if we have a million points and we have k clusters, we just have to perform k times 1 million comparisons. Because for each point, we need to find the closest centroid approximately. So the upshot is that k-means winds up being a lot more efficient on each iteration, which is why if you have a large number of points, you might want to choose k-means over hierarchical clustering.

What's an advantage, though, of hierarchical clustering over k-means? Even though it's less efficient, what's another--

AUDIENCE: [INAUDIBLE]

PROFESSOR: What's that?

AUDIENCE: More thorough.

PROFESSOR: More thorough.

AUDIENCE: And you can get a lot of different levels that you can look at.

PROFESSOR: Yeah, you can get a lot of different levels. So you can look at the clusterings from different perspectives. But the key thing with--

AUDIENCE: You don't necessarily know how many clusters there actually are. Hierarchical clustering will tell you all of the [INAUDIBLE]. You can just go down the tree and--

PROFESSOR: Right, so you could go down different levels of the tree and pick however many number of clusters you want. But the big reason-- or one of the kind of main advantages that hierarchical clustering has over k-means is that k-means is random. It's non-deterministic. Hierarchical clustering is deterministic. It's always going to give you the same result.

With k-means, because your initial starting conditions are random because you're choosing k random points, the end result will be different each time. And so when we do k-means clustering, this necessarily means that we don't necessarily want to do it once. Like if we choose k equals 3, we might want to do five different k-means clusterings and take the best one. So that's one of the big points with k-means.

There's a degenerate condition with k-means. So if my objective is to-- if my stopping criteria is that the centroid doesn't move, what's a really easy way to make the centroid not move by choosing k? What was it? K equals n, right? So if I have n points and I have k equals n, then all of my points are going to be their own cluster. And every time I update, I'm never going to move my centroid.

So in your problem set, you're going to be asked to compute a standard error for each of the clusters and a total error for the entire cluster. So what that is, is if I have the centroids, and I had each point in the centroids-- so what I'm going to do is I'm going to take the centroid for each cluster. And I'm going to find the distance from each point in the cluster to the centroid. And then, I'm going to sum up all of those distances over the entire cluster. That's going to give me my error. Not sure if this equation's totally right. There might be a like of division in there.

But the general idea, what I'm trying to emphasize, is that we can reduce this number by just increasing the number of k. And if we make k equal n, then this is going to be 0. So like I was saying with statistics, you never want to trust just one number. With k-means, you never just want to trust one clustering or one measurement of error. You want to look at it from multiple perspectives and

advantage points.

So why don't we look at the code and try to match up all of that stuff with what you'll see on your problem set? So the big function to look at for k-means is aptly named k-means. And it's going to take a set of points. It's going to take a number of clusters. It's going to take a cut off value, and a point type, and a variable named maxIters.

So first step, we get our initial centroids. All we're going to do is we're going to sample our points randomly and choose k of them. Our clusters, we're going to represent as a list. And we are going to, for all the points in the initial centroids, we are going to add a cluster with just that point.

And then, we get into our loop here. So what this is saying is, while our biggest change in centroid is greater than the cut off and we haven't exceeded the number of iterations, or the maximum number of iterations, we're going to keep trying to refine our cluster. So that brings up a point I actually failed to mention. Why should we have this cut off point, maxiters?

AUDIENCE: It'll go forever?

PROFESSOR: Yeah, there's a chance that, if our cut-off value is too small, or we have a point that's on a border and likes to jump between clusters and move the centroid just above the cut off point, that we'll never converge to our cut off. And so we want to set up a secondary break. So we have this maxIters, which defaults to 100.

So with this set up, though, there's a couple of things you have to consider. And this is, one, you need to make sure that maxiters is not too small. Because if it's too small, you're not going to converge. And you don't want to make it too large, because then your algorithm will just take forever to run, right? Likewise, you don't want to make your cut off too small, either. So sometimes you have to play around with the algorithm to figure out what the best parameters are. And that's, oftentimes, more of an art than a hard science.

So anyway, continuing on. for each iteration, we're going to set up a new set of

clusters. And we're going to set them initially to have no points in them. And then, for all the points in our data set, we are going to look for the smallest distance.

So that means we are going to look-- we're going to initially set our smallest distance to be the distance from the point to the first centroid. And then, we're just going to interate through all the centroids, or through all the clusters, and then find the smallest distance. Is anyone lost by that? Make sense?

Once we find that, we're just going to add that point to the new clusters. And then, we're going to go through our update. We're going to iterate through each of our clusters. We are going to update the points in the cluster. So remember, the update method sets the points of the cluster to be this new set of points you've given it. And it also updates the centroid, or it updates the centroid, and it returns the delta between the old centroid and the new centroid. So that's where this change variable is coming from.

And then, we're just go look for the biggest change, right? And if, at some point in our clustering, the centroids have stabilized and our clusters are relatively stationary, then our max change will be small. And it'll wind up terminating the algorithm.

And all this function does is, once it's converged or it's gone through the maximum number of iterations, it's just going to find the maximum distance of a point to its centroid. So it's going to look for a point that has a maximum distance from its corresponding centroid. And that's going to be the coherence of this clustering. And then, it's just going to return a tuple containing the clusters and the maximum distance.

So it's not a hard algorithm to understand. And it's pretty simple to implement. There any real questions about what's going on here?

So the example that he went over in lecture with k-means was this counties clustering example. So we had a bunch of data of different counties in the US. And we just played around with clustering them and seeing what we got.

So if we made five clusters, and we clustered on all the features, wanted to see what the distribution would be for, say, incomes, what this function, test, is going to do is it's going to take a k, a cut off, and a number of trials. So remember, we said that because k-means is non-deterministic, we're going to want to run it a number of times to find maybe we get a bad initial set of points for our centroids or for our clusters. And that gives us a bad clustering. So we're going to run it a number of times and try and prevent that from happening.

AUDIENCE: How do we [INAUDIBLE] multiple runs, because [INAUDIBLE] really different clustering happens [INAUDIBLE] after you run it a couple of times?

PROFESSOR: It can be tricky, to be honest. One technique you could use would be to have a training set and a development set. What I mean by that is, you perform a clustering on the training set. And then, you take the development set, and you figure out which clusters they belong to. And then, you measure the error of that development set.

So once you've assigned these development points to the clusters, you measure the distance to the centroid. And you sum up the squared distances, and you sum up over all the clusters. Then if you do that a number of times, what you would do is you'd choose the clustering that gave you the smallest error on your development set. And then, you'd say, that's probably my best clustering for this data. So that's one way of doing it. There's multiple ways of skinning the cat. Was trying to think of a good aphorism.

And actually, that's what's on your problem set. One of the problems on your problem set is to cluster based on a holdout set and see what the effect of the error is on this holdout set. So did that answer your question?

AUDIENCE: Yeah.

PROFESSOR: OK. And then, choosing k, there's different methods for doing it, too. A lot of it is you run a lot of experiments and you see what you get. This is where the research part comes in for a lot of applications. So you can also try some other automatic

methods, like entropy or other, more complicated measurements of error. But don't worry about those.

For our purposes, if you get below the cut off value, and you run a number of iterations, and you've minimized your error on your test set, we'll be happy. We want you to be familiar with k-means, but not experts in it because it's a useful tool for your kit.

Anyway, so all this code is going to do is it's going to run a number of trials. It's going to perform k-means clustering. And it's going to look for the clustering with the smallest maximum distance. So remember, the return value of k-means is the maximum distance from a point to its centroid. We're going to define our best clustering as being the clustering that gives us the smallest max distance. So that's another way you would choose your best clustering.

Yeah, and that's all we're going to do for this bit of code here. We're going to find the average income in each cluster and draw a histogram. So I think this is actually done. So we have five clusters. And what they're showing us is that the clusters-- if we take the average income of the different clusters, they're going to be centered at these average incomes. Let's see what some other plots look like.

This set of examples is using a point type that's called county. And county, like the mammal class, inherits from point. And it defines a set of features that can be used, or a set of-- it calls them filters. But basically, if you pass it one of these filter names, like allEducation, noEducational, wealthOnly, noWealth, what this is doing is it's selecting this tuple of tuples. And so if I say wealthonly, it's going to use this tuple as a filter. And for each element in this tuple of tuples, it has a tuple that has the name of the attribute and whether or not it should be used in the clustering.

So if it has a 1, it should be used. If it has a 0, it shouldn't be used. So if we look at how that's applied, it'll get a filterSpec. And then, it'll just set an attribute called atrFilter. And if we see where that's used, then it's going to iterate through all of the attributes. And if the given attribute has a 1, then it's going to include it in the set of features that are used in this distance computation. So did that make sense at all?

No? All right.

So the idea is to illustrate that if you use different features when you're doing your clustering, you'll probably attain different clusterings. So in that first example I showed, we used all the features. But in this example, we are going to look at only wealth. And that includes the features, if we look at this set of filters here, it's going to include the home value, the income, the poverty. And then, all the other attributes, like population change, it's not include. So those aren't going to be used in the clusterings. So this is going to change what our clusters look like.

So if we look at what we have for our clusters-- well, that's not very clear. Yeah, so let's see what happens. I'm not sure that's really going to show us. Probably better if I show them all at once, right? So this will take a while. Yes?

AUDIENCE: Actually I had just one question [INAUDIBLE] but there is a method showing us [INAUDIBLE] iterative additionally?

PROFESSOR: Mm-hm.

AUDIENCE: [INAUDIBLE]

PROFESSOR: It's like iterValues or iterKeys, yeah.

AUDIENCE: I was wondering how to go about using that in actual code.

PROFESSOR: In actual code? So you know that if you have a dictionary, d, you can do like d.keys. So remember I was demo-ing that code, the difference between range and xrange? Same thing. This is going to return an actual list of all the keys, right? What's d.iterkeys returns is a generator object that gives you, one by one by one, each key in the dictionary. So a lot of times, you guys in your code will use something like for k in d.keys(): to iterate through all the keys in the dictionary.

What this method does, though, is it creates an actual copy of that list of keys, right? So when you call d.keys, if you have a lot of keys in your dictionary, it's going to go one by one by one by one through each of those keys, add it to a list, and then return it to you. What d.iterkeys does is it skips that going one by one by one and

adding it to a new list. It just gives you a generator object which, when you use it in a for loop-- when you use it in a for loop, it's going to just yield the key one at a time without creating a separate list. That make sense?

AUDIENCE: So will it be more efficient?

PROFESSOR: Yeah, it's generally more efficient. And then, there's also, I think, iterValues, which goes through each of the values. And then, I think there's iterItems. And I think, if I'm not mistaken, if you do something just like that, so you do, for k, v in d, this is going to iterate through a tuple that contains the key and the values associated with that key. And it's equivalent to doing this. Make sense?

AUDIENCE: Yeah.

PROFESSOR: So where were we here? Oh, maybe I shouldn't have done this all at once. Why don't we just look at two?

Why don't we take a look at what the average incomes, what the clustering gives us for average incomes for education versus no education. That's probably not going to be a very good comparison. Just doing five trials and two trials for each clustering. So k-means is the efficient one, which means that hierarchical would take a long, long time. There we go.

So these are the average incomes if we cluster with k equals 50 on education. And then, there should be another one. I didn't create the new figure. So apparently there's a bug in the code. I wanted to show the two plots side by side so you could see the differences. Because what you should see is-- we would see a different distribution in incomes among the clusters if we clustered based on no education versus education level. But my code is buggy and not working the way I expected it to. So I apologize.