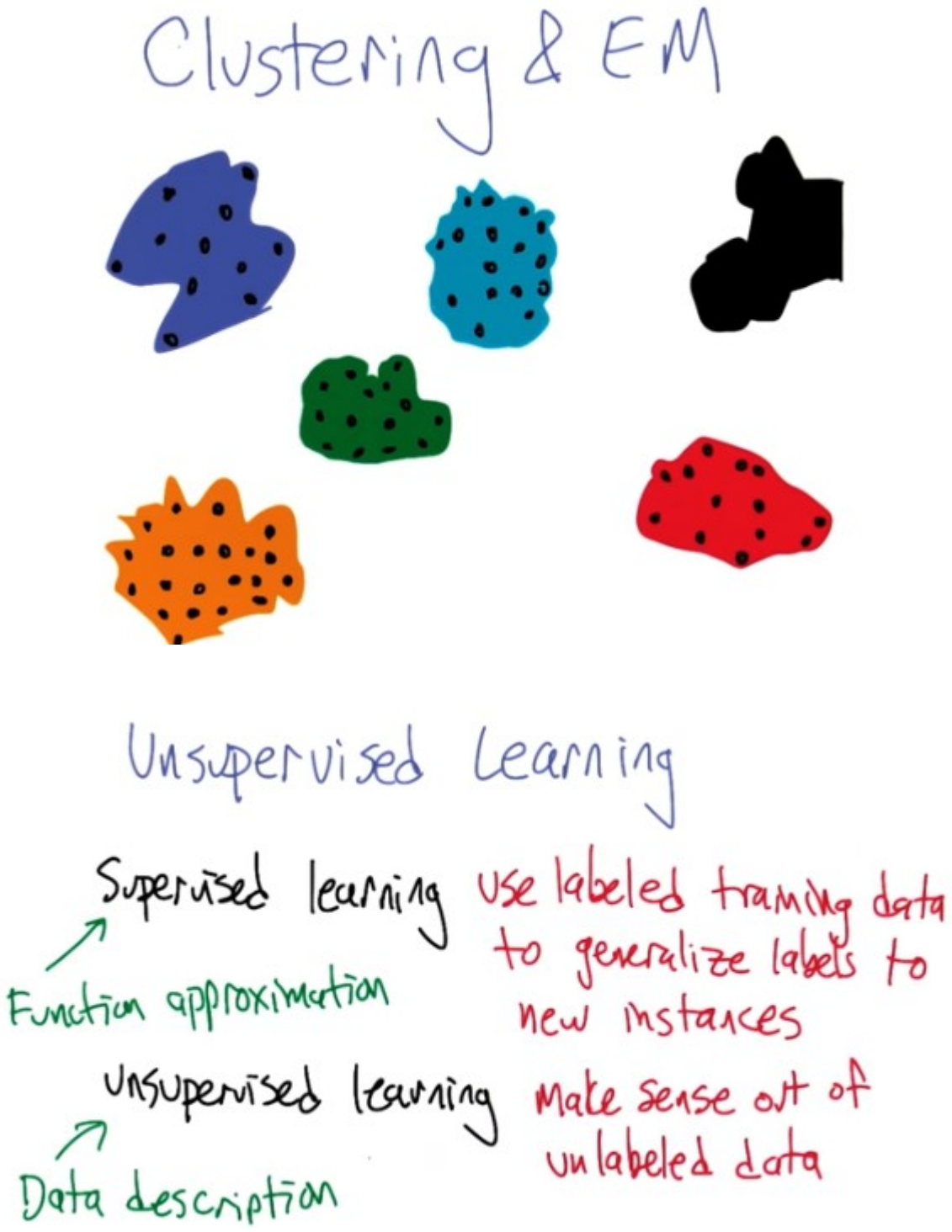
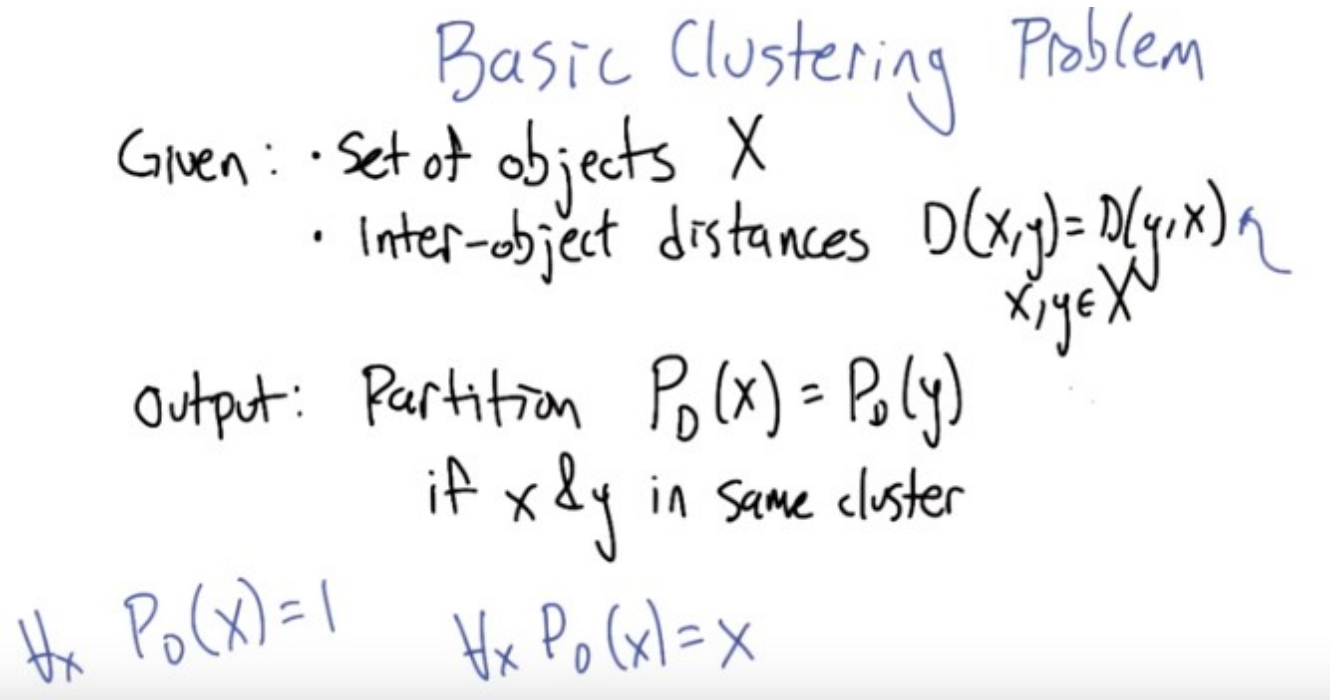


1. >> Dun, duh, dun, dun, duh, dun, dun, dun. Dun, duh, dun, dun, duh, dun, dun, dun. Dun, duh, dun, dun, duh, dun, dun, dun. Dun, duh, dun, dun, duh, dun, dun, dun. >> Hi, I'm Michael Litman and welcome to clustering and expectation maximization. I dunno, I feel a little drama might help kind of wake us up this morning. How are you doing Charles? >> I'm doing fine. I just had a lot of ice cream. >> [LAUGH] Dude, you really shouldn't eat ice cream first thing in the morning. >> It wasn't first thing in the morning. That was after I had a muffin, and a breakfast burrito. So it was third thing in the

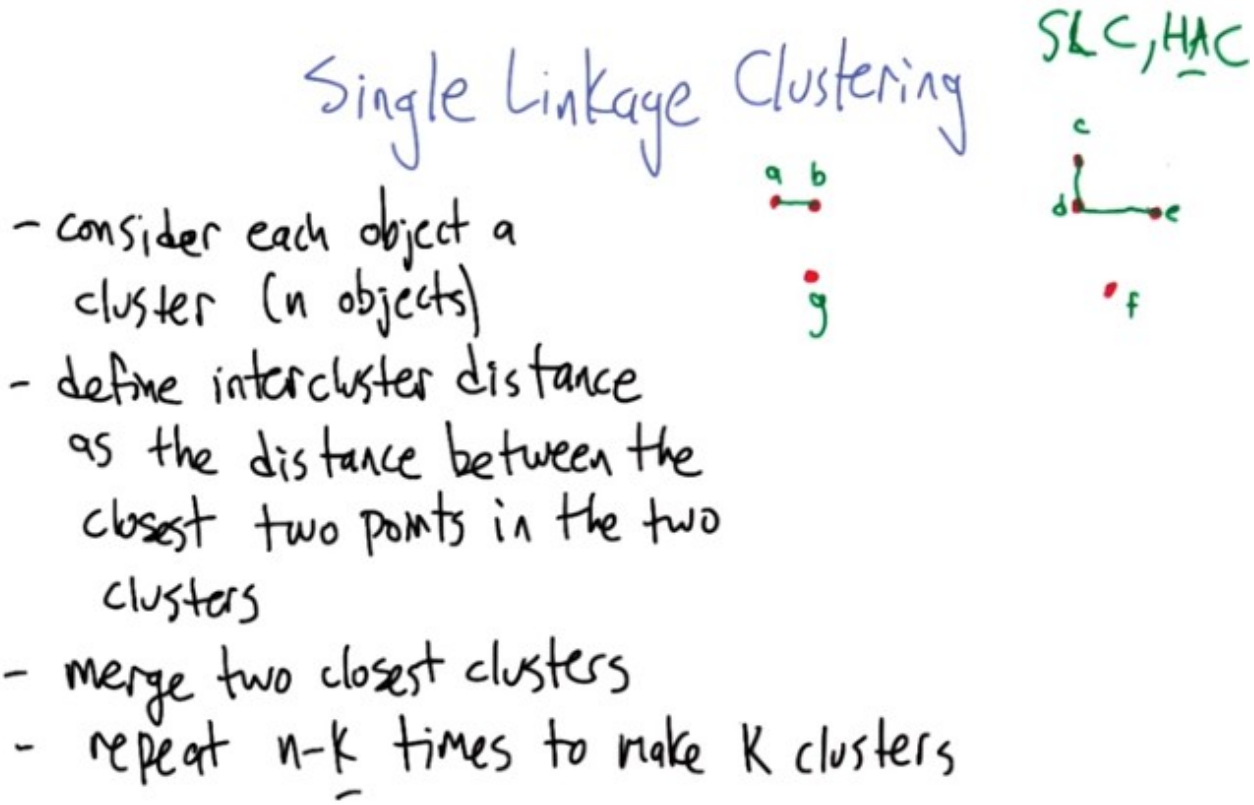
morning. >> You frighten me. As you may recall, the mini course that we are doing now is on unsupervised learning and we haven't really kind of set down and, and introduced these concepts. So, I, I thought it maybe worth taking a step back, even though we are going to talk about some particular algorithms today, to talk more generally about what unsupervised learning is. So up to this point in the first mini course we talked about supervised learning. Supervised learning, in supervised learning we use labeled training data to generalize labels to new instances. And what unsupervised learning is is making sense out of unlabeled data. So when I wrote these down I started to think boy, they're more different than you'd expect just by sticking an un in front. >> How so? >> Well they don't seem to really, the definitions don't seem to have all the much in common. You know, it's not like this one uses labeled training data to generalize labels to new instances, and this one uses unlabeled training data to generalize labels to new instances. Like I would think that you'd have the definition of one. You just stick an extra un in there. >> Well, that's kind of true, right? So you, if you, if you do some data description and make sense out of unlabeled data. And then I give you a new piece of data. Somehow, using that description, you would know how to describe it. >> Maybe. I mean there's definitely some, some unsupervised learning algorithms that do some amount of generalization. >> Mm-hm. >> But but it's, it's not as, it's not as unified. I think, in some sense the concern here is that unsupervised learning is just not as unified a problem, or as well defined, or crisply defined a problem as supervised learning. >> Oh, I think that's definitely true. >> So just as labels, and I think we talked about this in the intro to the whole course. That supervised learning, we can sometimes think of as function approximation, which is learning how to match inputs to outputs. Whereas, unsupervised learning is data description. It's about taking the data you've got, and finding some kind of more compact way of describing it. >> Sure, I buy that. And it makes a lot of sense.



x 和 y 都是 object, X 是 object 的集合.

2. But even though the unsupervised learning problem isn't really that crisply defined, we can define a basic clustering problem and, and be somewhat crisp about it. So, that's what I want to do next. I want to say that one of the classic unsupervised problems is clustering, taking a set of objects and putting them into groups. And so, here's what we're going to assume. We're going to assume that we're given

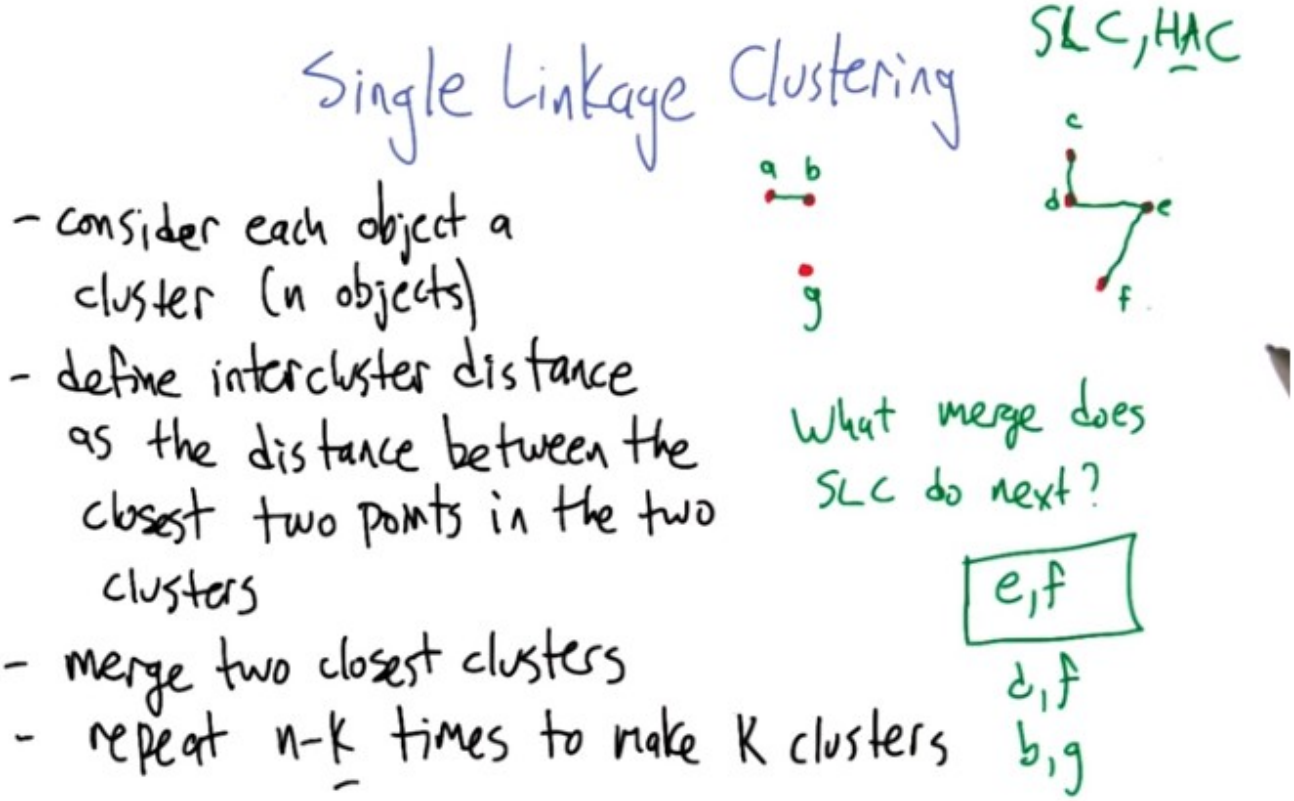
some set of objects X and we're given information about how they relate to each other in the form of inter-object distances. So we're going to assume that for objects x and y in the set of objects we have D(x,y) which is the same as D(y,x). Which says how far apart they are in, in some space. Now they don't actually have to be in some metric space. They just need to have this, this distance matrix D to find. >> So that makes sense, now let me ask you, so you said they don't have to be in a metric space. So you mean it literally does not have to be a real distance? It doesn't have to be a, a metric. >> Yeah, I don't, I don't think we're going to depend on that. I think we just need some kind of way of measuring how far apart things are. And it doesn't mean that they're embedded in some two dimensional space or three dimensional space. There's just numbers that relate them to one another. >> So they don't have to obey the triangle inequality, for example. >> I don't think so. I don't think we're going to depend on that in any way. >> Okay. So this is just like KNN because everything is like KNN. Where you have this notion of similarity and distance and that's where your domain knowledge is. So you've got a bunch of objects and your domain knowledge is these distances, these similarities between objects. >> Exactly, right. And in fact it's, it's surprisingly similar to KNN in that they surprisingly depend on similarness. >> Hm. So that's good. So KNN is similar to everything, so its distance is small to all things. [LAUGH] Wow, that is very meta. >> Alright. So that's, that's the input and the output that a clustering algorithm needs to create is a partition. Which is to say we're going to, I'll write it this way. That P_D, the partition function defined for some particular distance set D, for some object x, is just going to be some label, but it's such that if x and y are going to be assigned to the same cluster, the same partition, then they're going to have the same output of this P_D function. >> Okay. >> That make some sense? >> That does make sense. >> What would be a trivial clustering algorithm based on this definition? Right, it's taking its input the objects and then the distances. And then it spits out partitions. >> A trivial one would be, well, a trivial one would be put everything in the same partition (P_D(x) = 1). >> Yeah. So that would be one. So that would be, that would look like this. P, for all x in the set of objects, PD of x equals, let's say, 1. So that means that all objects are in partition 1. >> We're all humans. >> That's right. >> Okay. Here's another trivial one. Every single object is in its own partition (P_D(x) = x). >> Yes, good, that's the other one I was thinking of. Which maybe we could write like this. We'll just use the object name itself as the name of of the cluster. And now every object is in its own cluster. Now, we didn't define, as part of this, this problem definition, whether we should like the one where everybody's in the same cluster or the one that everyone's in different clusters or something in between, it's not really in the, in this definition at this level of detail. >> Hm. Well, that's fine. I mean, we can all be humans and we can all be unique snowflakes at the same time. [LAUGH]



第二步是求兩個不同的 cluster 之間的距離

3. In think in part because clustering tends not to have one consistent definition it's very algorithm driven. So there's different algorithms for doing clustering and we can analyze each of those algorithms somewhat independently. There isn't 「 really one problem and then there's a bunch of different algorithms for attacking that one problem 」. Each algorithm kind of is its own problem. So, I think the, the easiest way to go through and talk about unsupervised learning and clustering is to start going through some algorithms. So I'm going to start with single linkage clustering, because I think it's in many ways the simplest and most natural. >> Okay. >> And it has some very nice properties. So it's, it's, it's not a terrible idea, it's been very useful in statistics in a very long time. So, so here's how single linkage clustering goes, let's imagine we've got these objects. That we'll call them these little, these little red dots. We're imagine that they're objects and just because it would be hard to write down all the D values between them. Let's just literally use the 2 dimensional space, the distance on the slide here as the distances. Okay. >> Okay. >> So, you can, so if you were, if you were asked to cluster this how would you do it? Like, what do you think the, the groupings ought to be? >> Well, just staring at them they're either two or they're four. >> Mm. >> Or maybe, there's five. So, I would probably put the three on the left that there together, together. And I would put the four on the right that are together, together. Or I might notice that the two at the top of the ones on the right are more together than the other two and sub divide them some more. But, if I were just far enough away, I would definitely say so that seems like a reasonable clustering, but also the one that took all four of those and put them together is a reasonable clustering. >> Right. Yeah, yeah. That means that sort of agrees with what my eyes are thinking as well. Alright, so so it'd be good if we can recover that kind of structure algorithmically, so let's let's talk through this, this algorithm. It's called single linkage clustering, sometimes SLC. >> Or "slick". nice. It's a, it's a hierarchical agglomorative cluster of algorithm, or HAC. >> [LAUGH] I like that acronym even better. >> That's an Andrew Moore joke, for what it's

worth. >> Oh, it's a slick HAC. >> Nice. All right. So here's what we're going to do. We're going to consider each object a cluster to start. We're going to do this interatively. So we start off with N objects, like we have here. One, two, three, four, five, six, seven. And we're going to define the inter cluster distance as the distance between the closest two points in the two clusters. So in the beginning, all the points are in their own clusters. All the objects are in their own clusters. So the inter. Cluster distance is exactly the interobject distance. Okay? >> Okay. >> But, little by little, we're going to actually be aggregating things into larger clusters and we have to define what it means for how, how close is one cluster to another. And we'll say the close, the closest between two clusters is the closest of the closest two points between those two clusters. All right. So, so now what we're going to do is we're going to iterate. We're going to say merge the two closest clusters, because, you know, they belong together. And then we're just going to repeat that n minus k times To leave us with K clusters. So k is now an input to this algorithm. >> Okay. >> All right. So help me out. Let's step through this. What what would we merge first according to this algorithm? >> Well, they're either on cluster I, according to my eyesight, which is, of course, perfect. I would take the two left most ones, upper left most ones. Probably and merge them together. They look closest to me. >> Alright let me label them just to make your life a little bit easier. >> Oh that'd be nice. >> Alright. SO what do you think? >> I would bring A and B togheter. >> ALright. They're now a cluster. So we had seven clusters now we have six. Oh by the way we're trying to get down to two. >> Okay and then I would put C and D together. >> Yeah, that's what I was thinking too. >> Then I would probably want to put c, d, and e together, because they're in alphabetical order. >> [LAUGH]. All right, well, just to be concrete about this, why, what is the next step that we're supposed to do? What we're supposed to do is say which pair of clusters is closest. And again, the, the, distance between, say, I don't know, the a b cluster and the c d cluster is not the average over all the points or the furthest points but the closest ones. So, like I think c and d, the ,the distance between the (a, b) cluster and the (c, d) cluster is exactly the distance between b and d in this case. >> Mm-hm. >> Does that make sense? >> Yeah. >> Alright. So we know what's the currently closest pair of clusters. >> from, from where I am looking, it's either e to the c and d or g to a and b and they look very similar to me. >> E to C and D or G to B and A, that's what you said? >> Yeah. Probably G to B and A, staring at. >> I tried really hard to make it so that they're be an easy answer, but you're right, I, they're really close. So let's, let's I think it doesn't matter too much. we'll, we'll say that that one's neck. So now we'll say, now we'll do a quick quiz. >> Okay. So what merge does, single link clustering do next? Just give the pair of points I would be connecting. >> Okay. >> Say, separated by a comma.



上圖的意思是(e, f), (d, f), (b, g)都差不多, 而不是說連(e, f)的下一步就是連(d, f)

4. Alright. So, what do you think? >> Well I'm actually pulling out a piece of paper >> To measure things on the screen? >> You know, because my screen is actually flat on the ground or flat on the table. So, it looks like e and f are closest. >> Yeah, though I think we probably should be willing to accept ef, df, or bg. >> No. >> Because they're all really similar. >> E and f is close, b and g is close. But d and f is actually pretty, is clearly farther than the rest of them. Proved by. >> D and. Really? because it looks like an equilateral triangle to me. >> Maybe it is, actually. I take it all back. Yeah, actually you're right. You're right. It's the angle. Well actually d and f I think are closer than e and f. >> Alright. Good! So that yeah. So the, so the, I would say that any of those, any of those would be fine next. >> Mm-hm. And, I think interestingly won't change the answer, in the end. >> In the end. Yeah. So let's let's continue with this process now that the quiz explanation is done.

Single Linkage Clustering

SLC, HAC

- consider each object a cluster (n objects)
- define intercluster distance as the distance between the closest two points in the two clusters
- merge two closest clusters
- repeat n-k times to make K clusters ✓

What merge does SLC do next?

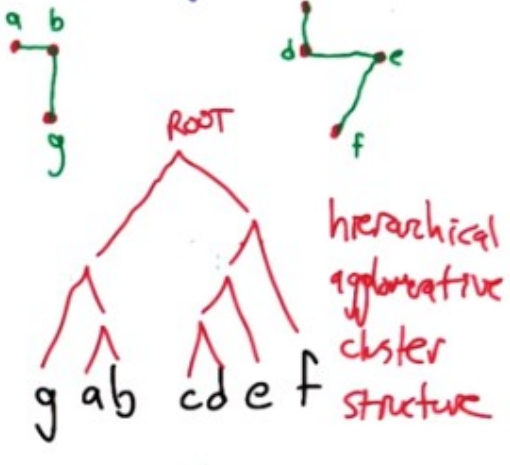
e, f

K=2

Single Linkage Clustering

SLC, HAC

- consider each object a cluster (n objects)
- define intercluster distance as the distance between the closest two points in the two clusters → median, average/mean
- merge two closest clusters
- repeat n-k times to make K clusters ✓



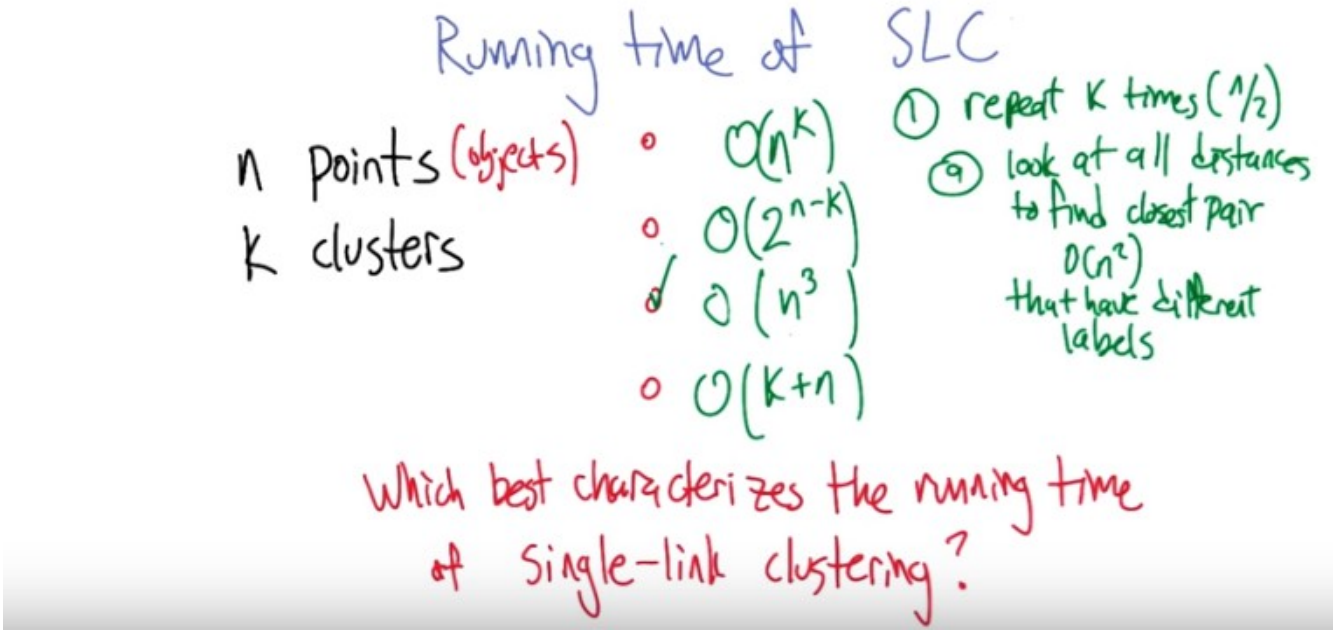
hierarchical agglomerative cluster structure

5. So let's finish this clustering process. So what were you thinking of doing next? >> Well, D and F

and B and G, we both thought were pretty close. So let's just do D and F. >> Well, D and F doesn't matter much anymore because they're already in the same cluster. >> Oh, that's a good point. That's a good point. >> So what is their inter cluster distance? >> It's zero. >> Yes. >> because they're in the same cluster. That's a fine point you make there, Michael. >> Sure. >> How about B and G then? I like B and G. And what do we do next? >> Well, we have two, two nice clusters. So, actually if we were to merge the next cluster. We would end up with, I guess b and d together. Doesn't really matter because it's going to merge the last two clusters together. >> Yeah. So, in particular what I was what I was getting at is we don't do any emerging next because now this repeat loop is done. So that's, this is what we end up with assuming that we had started off setting K equals two. >> Um-hm. >> Then, then we're done at this point. We have what looks like a backwards R and a funny looking, actually they look like Hebrew letters to me but that's probably not a universal perspective. So just one other thing to point out quickly. So we can represent the series of merges that we did using a structure that's kind of like this. We merged a and b first, and then we merged c and d. And then we merged, I think d and e? >> Yeah. >> Or the c, d, and e. And then we merged e and f. And then we merged g and the, and the ab cluster. And then that left us with the two clusters. So, this, in some sense it, we've captured the same information about what's connected with what. But this is kind of a nice way of representing it, because it actually gives us a hierarchical structure. A tree structure. >> A kind of hierarchical agglomerative structure. >> And there's lots of applications where it's actually useful to have that, that extra bit of information. because we can now very easily, look at remember, in the beginning you said that you could see three clusters? Well you can see if you just cut this tree structure here, we get, we get the three. Oh. Not quite. We almost get the three clusters that you wanted. Here it looks like f is separated from the others. It depends which, which distances we thought were closer. >> And also if you combine those two clusters again, the last two clusters, then basically you, you have a true tree with a single root. >> That's right. And, and that's kind of, that summarizes all the different cluster structures that you can get out of single link clustering. >> Hm, I like that. So I have a question, Michael. >> Shoot. >> Well I have two questions. The first question is do you know the difference between further and farther? >> I do, though. I sometimes use them interchangeably. >> Yeah, it turns out interestingly enough that that definitions are further and farther switch every 100 years or so >> Hm >> And unfortunately I was born in a time period where they're switching meanings again and so it drives me nuts. But in any case for our listeners, further farther means physical distance, and further means metaphysical or metaphorical distance. So one lives farther down the road, but goes further into debt. And you should almost never confuse the two, the people currently are using further as if it means farther. So I have a question here for you, which is you define the inter cluster distance as the distance between the closest two points in the two clusters >> Yep, that's what it says right here. >> Yeah, as if that were an immutable fact of the algorithm. Is it, or could we do something else? What would happen if we picked average distance or distance between the farthest two points as opposed to furthest two points or some other measure? >> [LAUGH] Well, we don't know if this is physicaly distance or metaphotical distance, right? Or metaphysical distance, it's just d. >> That's true. >> So it's like, we'll call it forthest, with an o. >> [LAUGH] Please don't. >> The forthest point. Yes good. And average is another one. That's true. Yeah, these are different things you could define. I, my recollection of this is, they're no longer single link clustering. Single link clustering is defined to be closest. But you also have other things like average link clustering, which I think is the one with the average. And then there's maybe max link clustering. I forget exactly what it's called, where it does the forthest. >> I think I saw a talk once as I didn't understand which had to do with median distances. And that somehow you could prove interesting things because of it as opposed to the mean, which is what I think you mean by average there. >> Yeah. Average meaning mean. Yeah, I mean, generally, things like median are really good if the numbers on the distances don't matter, just their orderings. >> Mm. >> So, sometimes people dif, describe that as being metric versus non-metric. Non-metric, median is usually a non-metric statistic. Average is, is a metric statistic. It actually, the, the details of the numbers matter a lot. >>

Okay. Well that all makes sense I guess the basic idea is that what I was saying before that the distance was some notion of where your domain knowledge comes in. I guess it's also true that the, how you define intercluster distance is also a kind of domain knowledge. It's another parameter to the algorithm. That seem fair? >> Yeah. >> Okay.

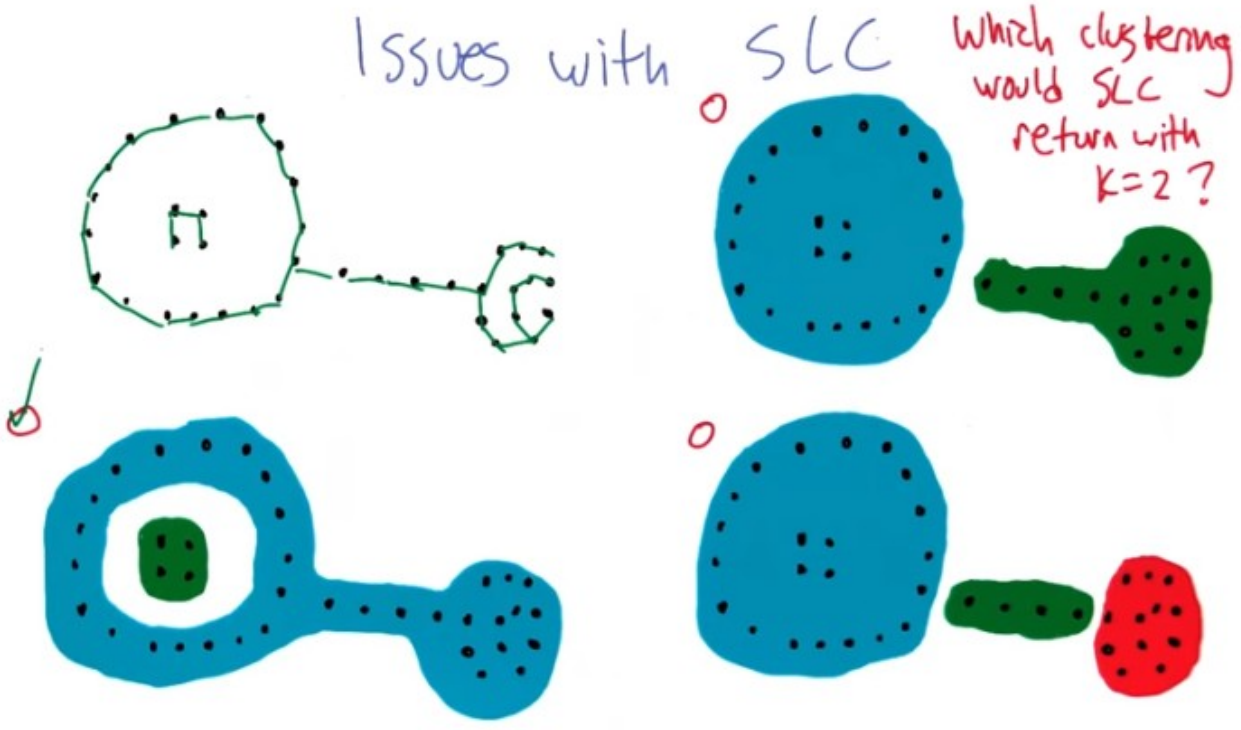
6. So there's a couple of really nice things to be able to say about single-link clustering. So one of them is that it's deterministic, right? So we run the same algorithm, and unless there's ties in the distances, it, it will give us an answer, which is, which is nice. Doesn't require any kind of randomized optimization. Another thing about it is that if you're doing this process in a space where the distances represent edge lengths of the graph then this is actually the same as a minimum spanning tree algorithm. >> Hm. >> And another nice property is that the running time of the algorithm is actually relatively friendly and can be characterized so let's, let's think that through. So let's imagine we've got n points or objects that we want to cluster. And we've got K clusters that we want as our as our target at the end. Which of these different running times best characterizes the running time of single link clustering?



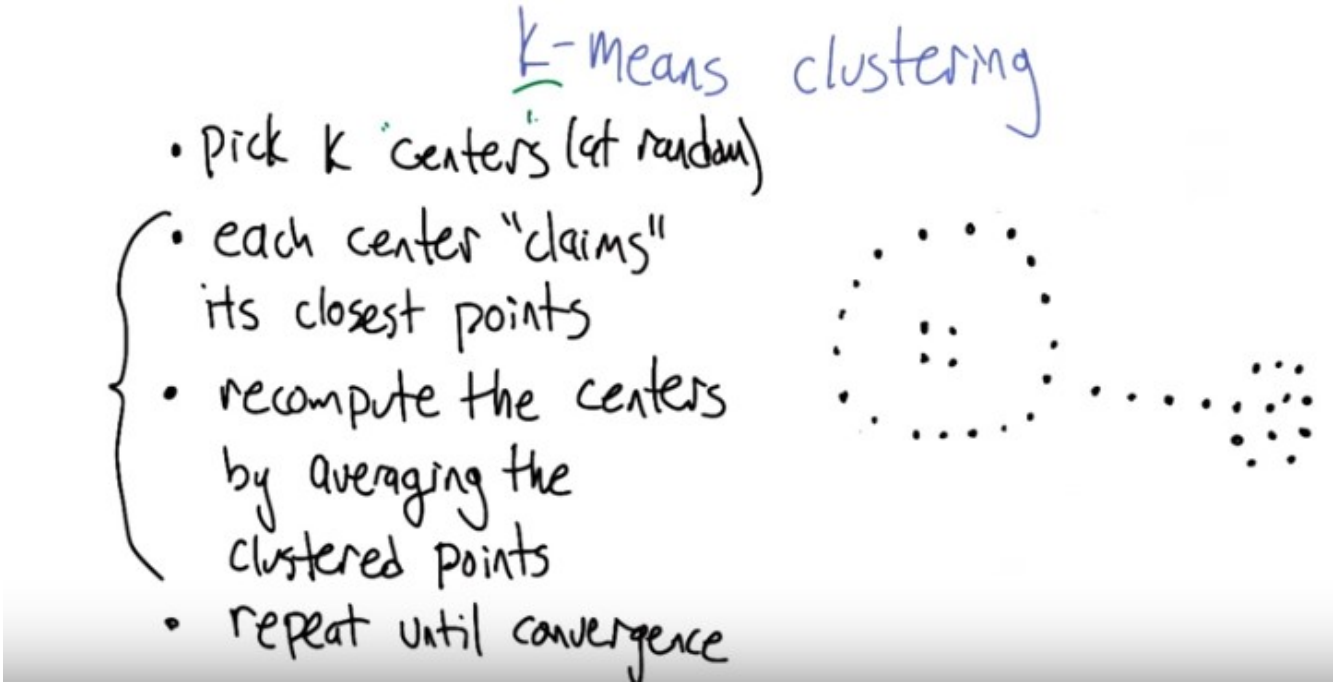
7. All right. What you got? >> So I look at this and believe it or not I, I came to an answer by two completely different ways. >> Okay. And do they, do they agree? >> They, they happen to agree, but one of them is kind of lame. It's sort of the thing that you would do if you were doing the SAT or the GRE and you wanted to quickly get a guess at the answer. >> Oh, I see. >> And it's that k, very small k and very big k are kind of both easy (即時間複雜度小). It's the one in the middle where it gets hard. And so any function that depends on k directly, in a way where as k gets bigger it gets harder, as k gets smaller, it gets easier, is probably not right. Which made me think was probably n cubed. And so then I tried to think about how the algorithm worked. And so, the worst case way of thinking about the algorithm is at every single time you have to figure out which two points are closest to one another. >> Alright, hang on a sec. So we're going to repeat k times, and the hard case is when it's, like, n/2 (意思是 k=n/2 時, 因為前面說了, k 取中間值時是最複雜的), right? >> Right. >> And what, what is it that we do at each step? >> Well, we find the two closest points. >> Good. >> Right, and that, and how long, and yeah, and how long does that take, it's exactly, it's n^2. Because you have to do all possible combinations. It's a little better than n squared, right, it's, you know, but it's big O of n squared. >> Right, and it's possible that if we store those distances really cleverly we can get it down a bit more,

but, yeah, n squared seems like a nice way to characterize it. So don't we need to do some work to merge the clusters together so that the next time through we get the right set of distances that we're looking through. >> We could do that and that would work fine and that's probably the right thing to do, but just kind of at a high level, what you just really want to do is, you just want to find the closest two points that have two different labels. >> Okay. >> So what I associate with every, with every object x is the label that it currently has. >> So you're going to look at each pair. Ignore it if the, the labels are the same, on both of the clusters that the two objects are in. >> Mm-hm. >> And otherwise find the minimum. >> Right. And what, and I can. >> That sort of works. >> And I could do that in time linear in the number of pairs. So it's still n squared, I don't have to sort or anything like that. >> True, though we could, it could be a little bit smarter to not reconsider the same pairs over and over and over again. >> That's right, you could. If you could use, you could probably use something fancy like Fibonacci heaps, or hash tables or something. And in fact, I know that there are people who've gotten entire PhDs on clever ways of doing this without having to consider all points. By dividing points up into little boxes where things tend to be closer to one another. Actually looks a lot, very hierarchical as well. >> Fair enough. >> Okay, so we do, we do, we do a, that takes n squared time. How many times do we do a? >> Well, we do it about n times. >> Alright. So that gets us the n cubed. And is it, is it clear that the other ones are definitely wrong? Well, since we can do this in n squared time, and the number of clusters can be large, the first one is a bad answer. The second one is a bad answer. How about the last one? Is it, is it possible that we could actually do this in linear time? no. Well, for, for, for very small or very big k, maybe. Like, so k equals 2, yes, that's easy, that's linear, k equals n, that's easy. because we started out that way. >> I see, but for in between k's. >> because you still have to find the closest distance, and there's, you have to find distances and the only way to do that is to consider all the distances. >> Yeah, at the very least the, the number of inputs that you need to even sniff at is quadratic. >> Right. >> Because the d can be any kind of structure. So, yeah, okay. So this is really the only answer that, that is even close. But I think it could come down a little bit from n cubed. >> Oh. I'm sure it could. I mean, because I just came up with the silliest, the, the simplest algorithm that you could think of that wasn't absolutely stupid. So I think that would. You could certainly do better than n^3 (因為找 pair 的算法可以稍微優化一下). But not much better than n^3. >> All right. End of quiz. >> Yay.

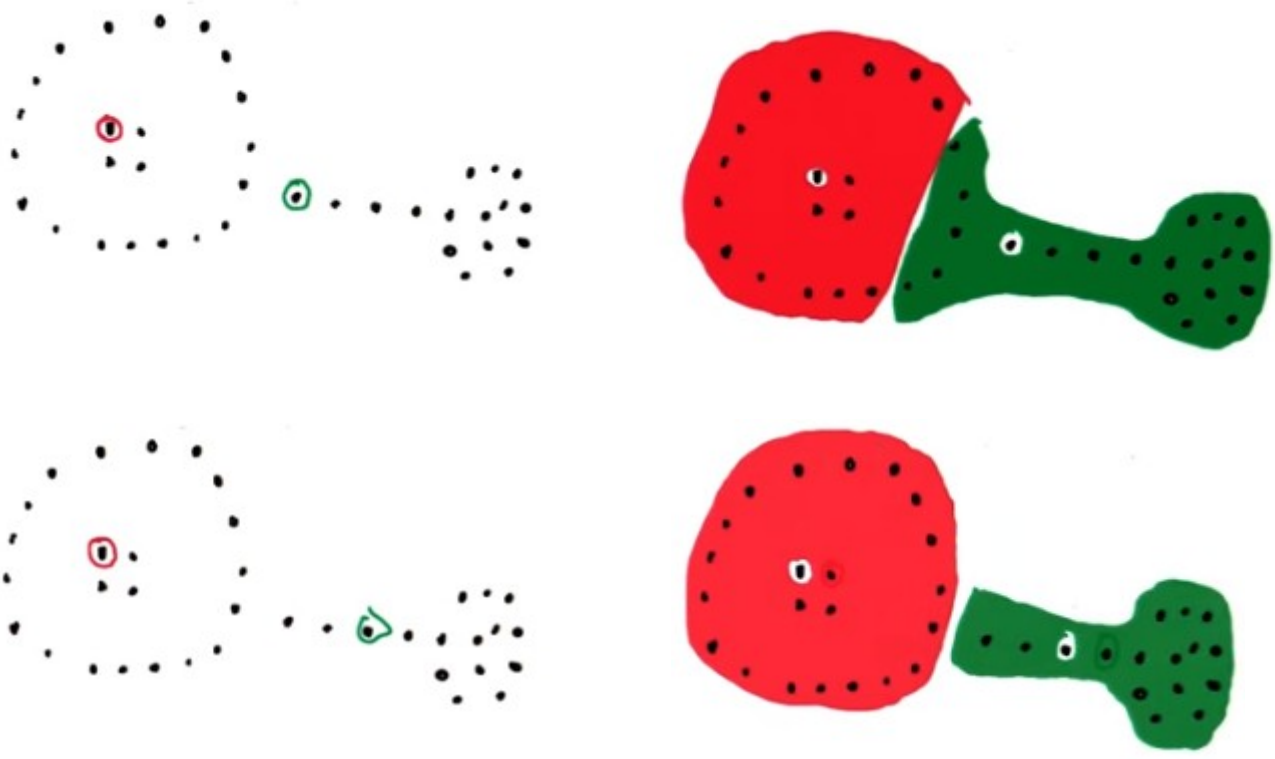
8. Here's another quiz. Just to kind of practice this idea of what singling clustering does and to think about it a little bit more. But also to wonder with, if maybe is it, isn't exactly what we think of when we think of clustering. So so here's a set of points that I drew in the upper left corner here in black. And the question is, if we run singling clustering with K equals two. Which of these three patterns will, will we be getting out? >> Okay, I got it. >> All right, good, let's let's give everybody a chance to think about it. >> Okay. Go.

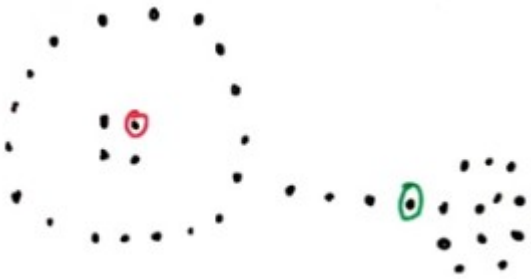


9. >> All right, so, so how does this work, Charles? How do you do single link clustering? Again, assuming the distances are distances in the plane, just because it makes it easy. >> Right. So you just keep finding the clusters that are closest, and you defined it as where the, the two closest points in two clusters are what makes them, is what defines their distance. >> Mm. >> And so if I look at this, if you look at, look at the big outer circle. you'll, you'll notice that each of, you start with any one of those points and it's always going to be closer to one of its neighbors on the circle than to, say, one of those four points in the middle of the circle. Right? >> They're close to each other. >> They're close to each other. >> If we start doing mergings, it, it ought to look something like this, right? Where we've got, I don't know. [SOUND] Maybe. [SOUND] But little by little it's going to be linking these things up. >> Mm-hm. And that's the key thing there that you were drawing before that the bridge between the two big circles, every point there is always going to be closer to one of the points on the outer circle clusters than it's ever going to be. To anything else. And so, you're going to end up making, one big cluster, out of the outer shell. Which is kind of weird. >> It is a little bit weird. You get these kind of stringy clusters because two things are close to each other if they're close anywhere. >> Right. >> So, I don't know. To my eye, the best one is the bottom right. But it wouldn't do that because k equals 2. >> Right. >> So to my, for what's, for what's left, I guess I like the upper right. But that's not what single linkage clustering would do. It would actually kind of walk around the circle and link things up. Like in the lower left.

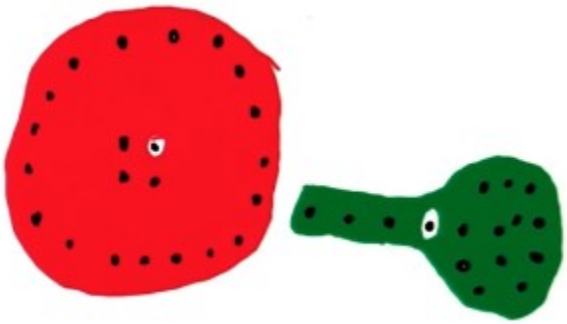


以下小圖順序是:
小圖 → 小圖 →
小圖 → 小圖 →
...





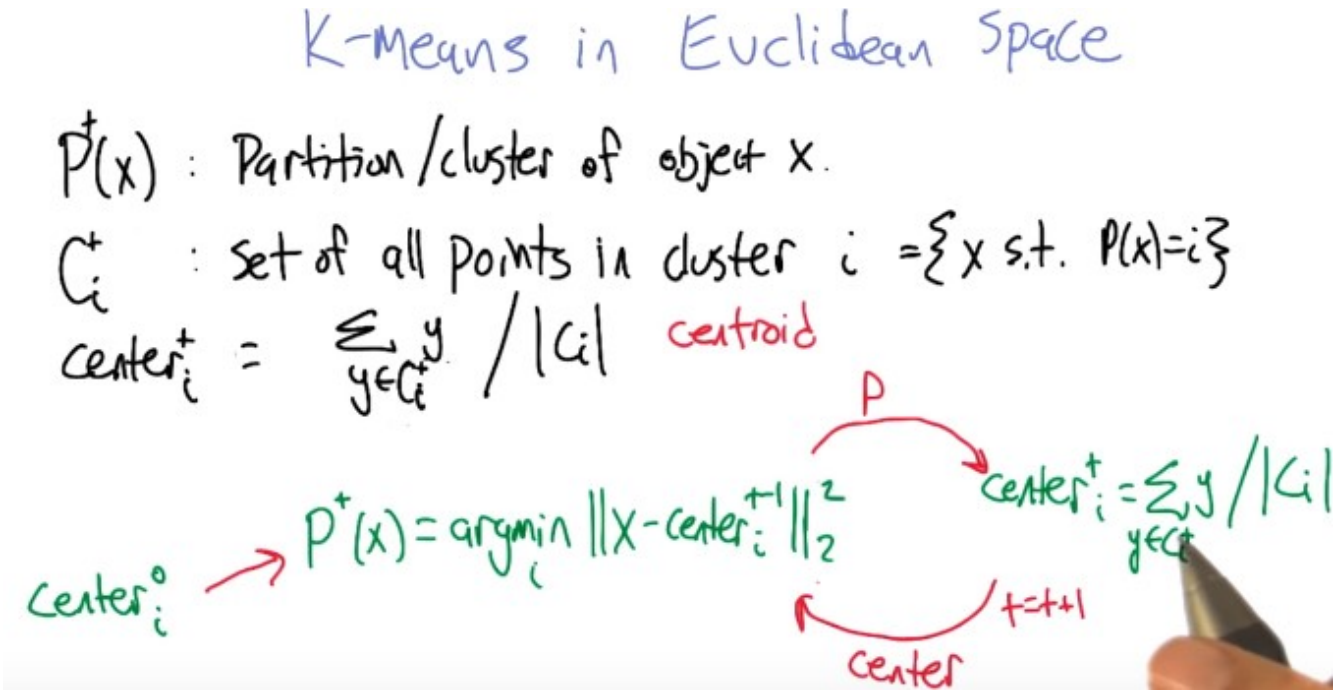
注意上小圖的兩個中心都右移了



上小圖跟它頭上那個圖一樣, 再找中心會不變, 即 converge 了

10. So we're going to talk about a different clustering algorithm that, that at least doesn't have that weird property and it has some other nice, positive properties. But there's trade-offs. There's always trade-offs in clustering. So k-means clustering is what we're going to talk about next. And, and the, the feel, the basic flow of the algorithm is like this. We're going to first pick a k. k is going to be the number of clusters that we're going to try to produce. And the way it's going to do it is by picking k of the points at random to be centers. And then we're going to repeat the following process, each center is going to claim it's closest points. So, we're going to cluster all the points based on how close they are to the k centers that we've defined. Then, we're going to recompute, based on that clustering that we just did. Recompute the centers. And the center would be the average of the points in its cluster. And then we're going to tick tock back and forth. We're going to go back and repeat 'til convergence. For the new centers claiming their closest points. The new groups of points computing their average and back and forth like that. So let's step through an example and then and then we'll try to analyze what this actually does. >> Okay, that makes sense. >> Alright. I'm going to do K equals two. In this example the same set of points that I used in the previous example. And I'll just you know, kind of randomly. Choose two of them to be the two clusters the red cluster and the green cluster alright? >> Okay. >> So that's this, that's this first step done. Now what we're going to do is each of these centers is going to claim. Their closest point. So we are going to take all the points in this space and assign them to one of the other of these two clusters. And it ends up looking like this. Alright? So this is now the points that's closest to the green centers have all been put into a green blob. The ones that are closest to the red center, all get put into the red blob. >> Okay. >> Okay? So, that's, that's step two. Now what we do is recompute the centers. So the center of the red blob is still pretty close to where it was before but if you look at the green blob, the center of the green blob ought to be to the right. You see that? >> I do, I do. In fact it should be a lot more to the right cause there's a whole lot of points on the right. >> Yeah, though, I did this sort of by eye, so it doesn't, yeah, you're right. It should be, it should be closer to the clump on the right, but I, I didn't quite do it that way. >> Okay. >> So I moved it just a little bit more to the right. See that? >> Mm-hm. >> But now we can repeat this process. We can say," Okay, now everybody join the group that you're closest to." And one nice thing that happened now is that the group of points on the left all joined together in red and this sort of weird hammery thing on the right became green. And now we're going to recompute our centers again. We're going to say, where's the center of the clusters given the way that they've been painted. And that, again, move things a little bit to the right in both cases. We ask every point to join the team that they're closest to. And we get that. And that actually turns out to be exactly the same clustering that we had a moment ago. So when we recompute the centers, they remain unchanged. And so we've converged. So it, it seems to have clustered things reasonably given that I didn't actually run this. I just did it by hand. >> Okay. Can I ask a question? >> Oh, please. >> Okay. You seem to have drawn this in such a way that the centers are always one of the

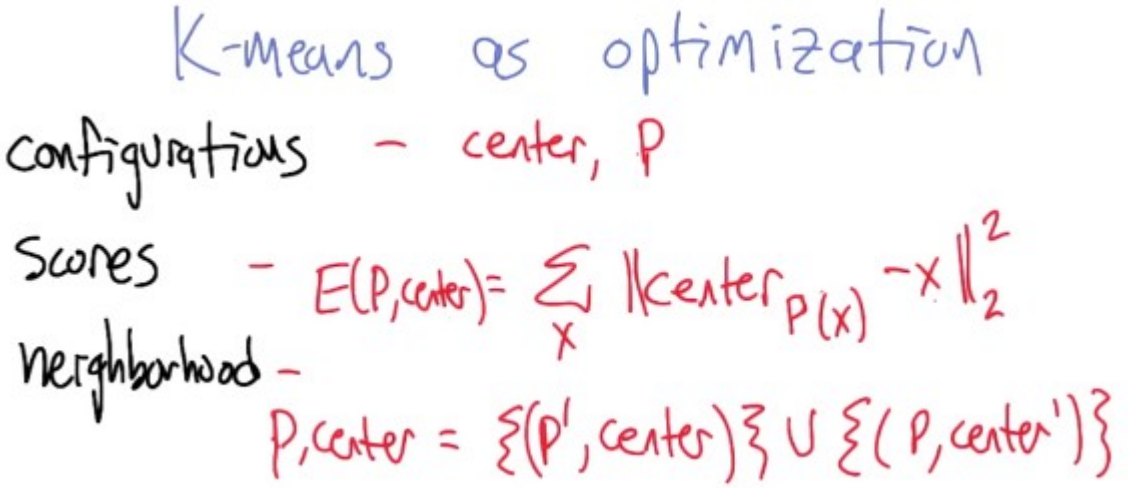
points. But that's not really meant to do, is it? >> No, no. It's definitely not what I meant to do. It's it realy is just. It, it, doesn't. The center is not necessarily a point that's in the collection of objects (即可以是空白處某點). >> Right. >> Yeah thanks for pointing that out. >> Okay, so this makes sense and that look better than what single linkage clustering, or single link clustering, or single linkage or whatever it is called, clustering did. >> At least for this kind of example, yeah, it, it produced kind of more compact clusters without giant holes in them. >> M-hm. >> So so do you have any questions about this? About what it does. >> Yes. So, so, [LAUGH] I think I might have some questions Michael. And they may even be questions you'd like to here. So I asked one question about what the synergy looked like. So I have a couple of questions, one is does this always converge? Does it always come up with a good answer? >> Yes, those are good questions.



是 P^t (x), 而不是 P^+ (x)
是 C^t_i, 而不是 C^+_i
i = {x s.t. P(x) = i}中的 s.t.即 such that

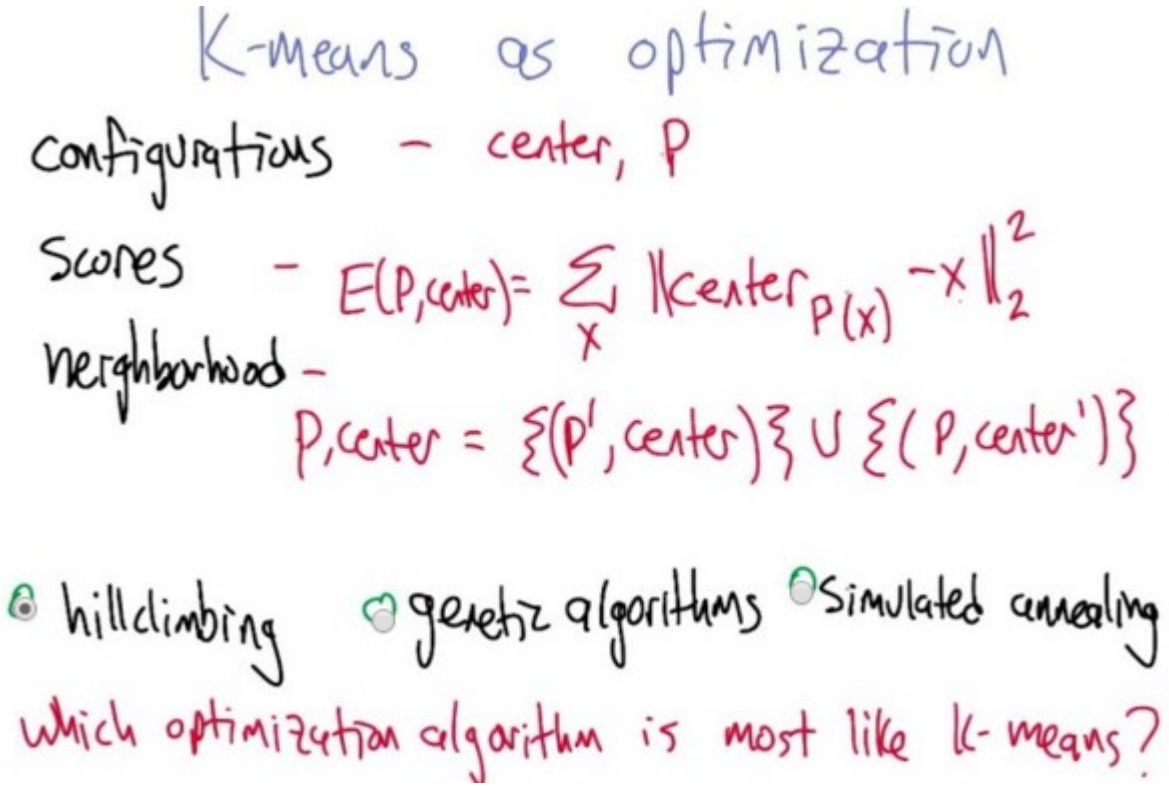
11. Alright, so what I'd like to do is work up to a proof that K-means does something good. [LAUGH] And to do that I think it's helpful to have little bit more precise notation than what I was doing before. So here we go, let's, we're going to work up to doing K-means in a Euclidean space. And the notation that I'm going to use is, we've got at any given moment in time in the algorithm, there's some partition, P^t (x). And these clusters are defined the same way that we did before, which it just returns the, you know, some kind of identifier for cluster x is in and this is at iteration t of K-means. We also have another kind of more convenient way of writing that which is C sub i of t, which is the set of all points in cluster i. It's really just the same as the set x, such that p of x equals i. Does that make sense? >> Yeah, okay that makes perfect sense. You've got some kind of partition, and everything in the same partition belongs together in something you're calling C for cluster. >> Good, and we're also going to need to be able to refer to the center of a cluster because we're in a Euclidean space, it's meaningful to add the objects together. So, take all the objects that are in some cluster, Ci at iteration t and divide by the number of objects in that cluster. So just summing those points together. This is also sometimes called the centroid, right? >> Right, so it's the mean or the centroid. >> Yeah it's like. That's right. I

mean in one dimension it's definitely the mean. In higher dimensions it's like a per dimension mean. >> Oh, so in fact you're going to end up with K means. >> Oh, I see what you did there. >> Mm. >> So now, here's an equation arrow version of the algorithm, that we start off picking centers in some way for iteration zero, and we hand it over to this process that's going to assign the partitions. In particular, the partition of point x is going to be the minimum over all the clusters of the distance between x and the center of that cluster. Alright, so it's just assigning each point to its closest center. Does that make sense? >> Mm-hm, and that all those bars with the sub two (即||_2) just mean Euclidean distance. >> Yeah, we're just, that's right, exactly. We just computing the distance between those two things. And we hand that partition over to the process that computes the center. So it's, it's now using this other representation of the clustering Ci, it's adding the points together, divided by the number of points in that cluster, and it hands those centers back to this first process to recompute the cluster. So we're just going to be ping-ponging back and forth between these two processes. >> Okay. >> Good? >> Sure, and t keeps getting updated after every cycle. >> Right, yes. So, this is where t gets incremented, t plus plus. So this is the algorithm, and an interesting fact about it is: Do you remember when we talked about optimization? >> I do remember when we talked about optimization. >> And optimization is good because it's finding better and better answers. So if we can think of this as being kind of an optimization algorithm then that could be good. It could be that that what we're doing, what we're going around here is not just randomly reassigning things, but actually improving things. >> Okay. Well how are you going to convince me of that? >> Alright, let's do that.



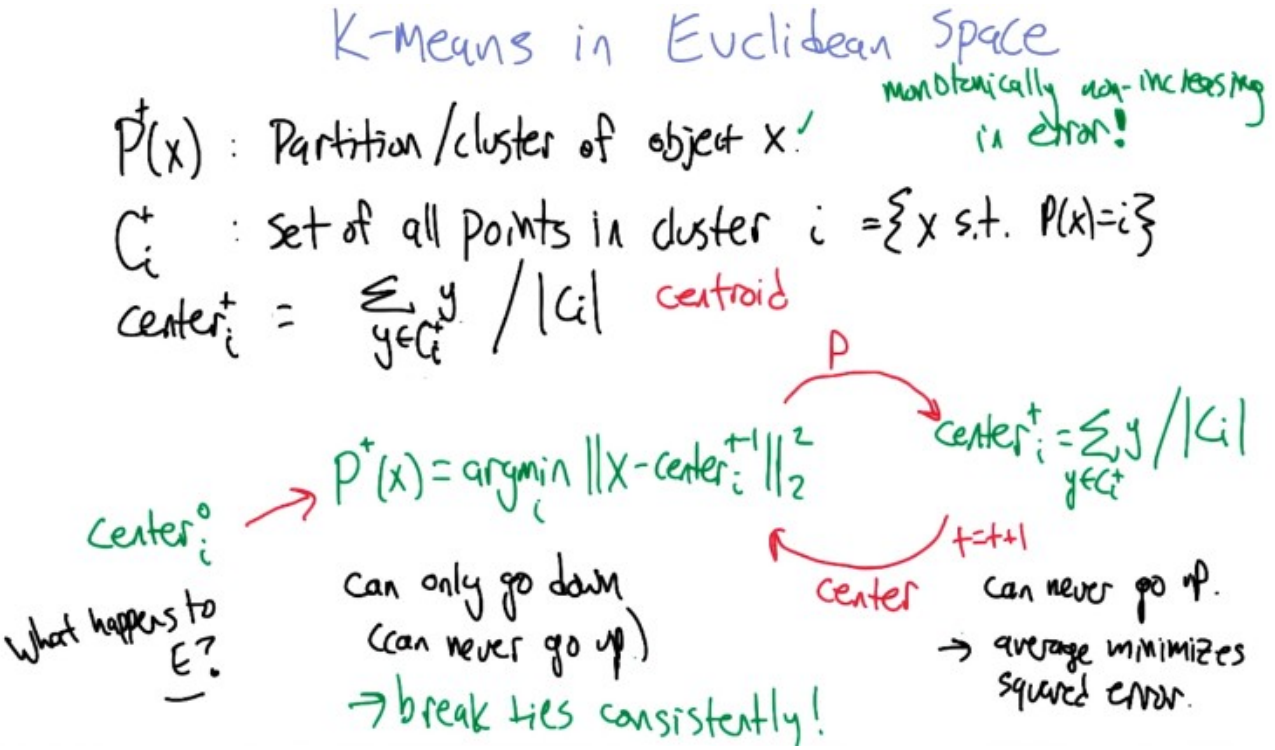
12. We're going to look at K-means as an optimization problem. So remember when we talked about optimization we worried about configurations, I think we called them inputs at that time but I think it's going to be helpful to think of them as, as configurations here. There was a way of scoring configurations and we were trying to find configurations that had high score. And for some of the algorithms we needed a notion of a neighborhood so that we could move from configuration to configuration trying to improve. >> Mm-hm. >> So in this setting, the configuration, the thing that we're optimizing over, is the partitions, the clusters, and we also have this kind of auxiliary variable of where the centers are for those clusters. And what we need now is a notion of scores. How do we score a particular clustering? So do you have any thoughts about what would be a better or worse clustering according to the kind of K-means algorithm? >> Well, the thing about what you were saying earlier, about what we were trying to do with creating these clusters, and I look at this notion of a center, so something pops in my head. So, you would like to have centers or partitions that somehow are good representations of your data, and why does that matter? Because you said in the very beginning that we often think of unsupervised learning as compact representation. So if you want to have a compact

representation it would be nice if you don't throw anything away. So, I'm going to say that a good score will be something that captures just how much error you introduce by representing these points as partitions or in this case as centers. Does that make sense? >> Okay. I guess that's a, that's a perfectly reasonable way to think of it. I, another way to think of it is in terms of error, right. Yeah, which I guess is the same idea. Like if we're, if you think about the object as being represented by the center of its cluster. >> Mm-hm. >> Then we want to know how far away from the center it is. >> Right. And the farther away it is, the more error you have in representing it. >> Right. So, here is a concrete of writing down what we think the scoring function could be. So we're going to say that the error, it's kind of the negative score, right? This is something that we want to minimize even though generally, we've been talking about optimization as maximizing. Here is something we want to minimize. That if you give me a particular way of clustering it and you define the centers based on, say, that cluster, what we're going to do is we're going to sum over all the objects the distance, the square distance actually, between the object and its center. Right? So P(x) is the cluster for x and center sub that is the center for that cluster. >> So that drives home the idea that we're talking about Euclidean distance here because x would have to be in the same space as the centers are by definition. Okay. >> Yeah, and I'm not sure exactly how we're going to define neighborhood. But one way to define neighborhood is that the neighborhood of a configuration, which is a p and a center (即 configuration is a p and a center), is the set of pairs where you keep the centers the same and you change the partitions, or you keep the partitions the same and you move the centers. So you're basically changing one of these at a time. >> Oh, that's very clever, Michael. >> Thanks.



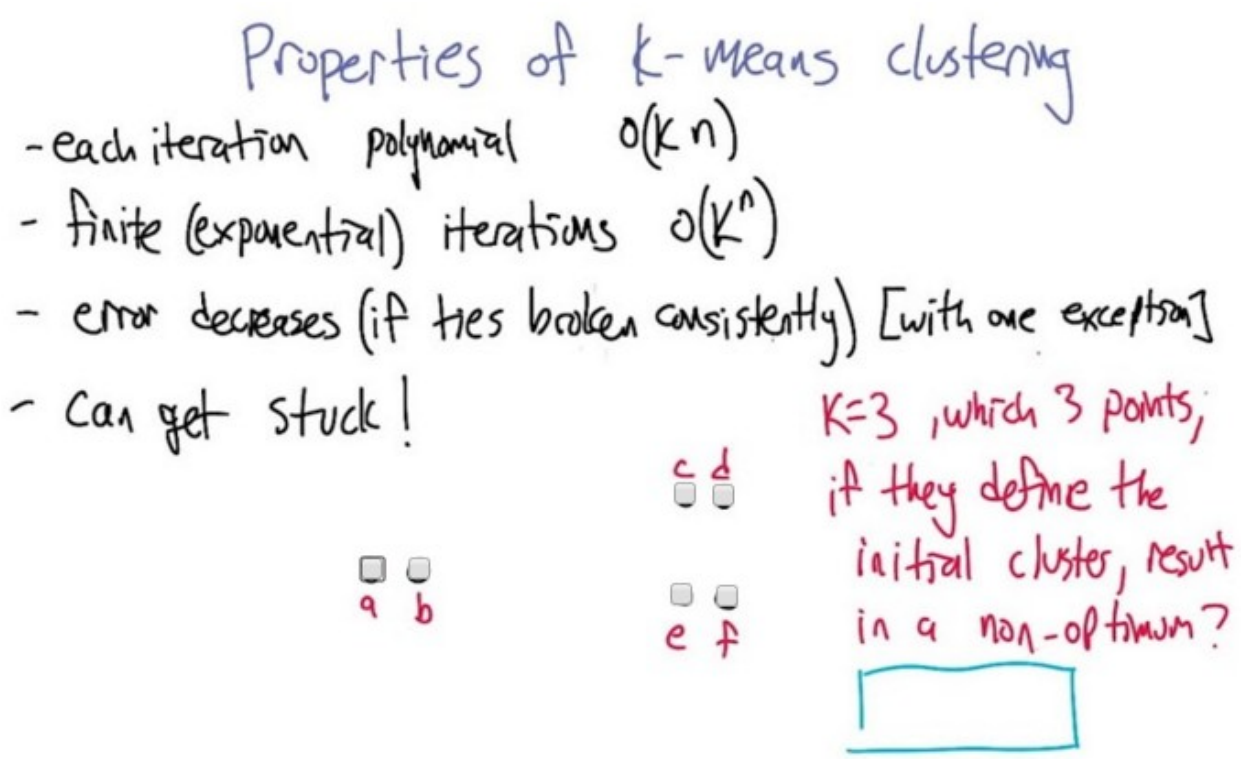
13. Great. So now, looked at this way, you can think of the k means procedure as actually moving around in the space and it follows one of the optimization algorithms that we talked about. So here's three of the algorithms that we talked about: hill climbing, genetic algorithms and simulated annealing. These are all randomized, optimization algorithms and one of these is kind of a lot like K means so which one do you think it is? >> Okay let me think about it.

14. What do you think? >> I think it's hill climbing. >> Me, too. >> It's certainly not simulated annealing because there's no annealing. >> Yup. >> Simulated or otherwise. You might think it's genetic algorithms because you've got all of these centers that are all moving around at once, but they're all a part of the ultimate answer. They're not populations. So it's really hill climbing. You just take a step towards a configuration that's better than the configuration you had before. >> That's right, except what we haven't said yet is the way we defined hill climbing, it actually evaluated the score and chose a neighbor that maximized the score. And we didn't show that these steps actually do that. We just defined them in a particular way. Here's the wicked cool thing. It really does do hill climbing. It really does find the neighbor that maximizes the score, minimizes the error in this case. So let's show that because it's kind of neat. Every time I see this I'm always surprised and delighted. >> Oh, I'm looking forward to being delighted.



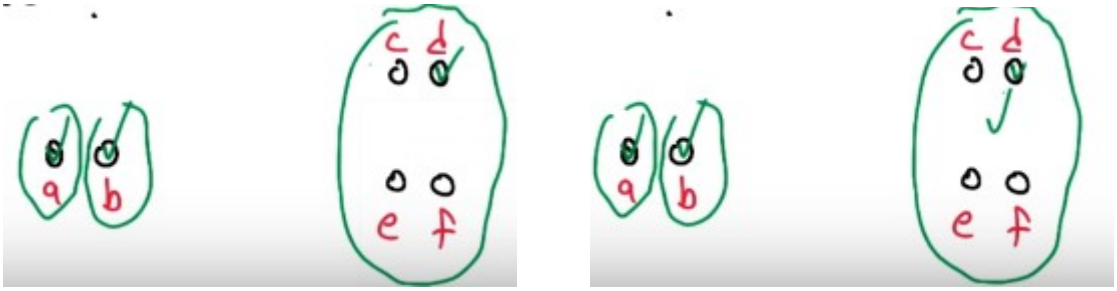
15. Alright so let's see if we can figure out what happens to this E function (即 error) as in one step we update the partitions and in the other step we update the centers. So let's look at the partition one first. The way that this partition is being defined is we, for each point loop over all the clusters and find the cluster whose center is closest to x in terms of squared distance. What happens to E when we do that? Well we move a point from one cluster to a different cluster only if it causes the square distance to the center to go down. So that means that the error, either stays the same if the point stays in the same cluster or it goes down if it goes to a better cluster. >> That makes sense. >> So, can only go down. >> Well, it can never go up. It's different than saying it can only go down. >> Agreed, because it could stay the same. What happens if it stays the same? I guess, not necessarily anything interesting. >> Right. >> But, certainly, when we converged, it stays the same. >> Right. >> Alright. Now let's look at the other side here. So that's what happens when we move things into partitions. And in some sense that seems easy. Because we only move things if it causes the error to go down so it really is a lot like hill climbing >> Mm-hm >> On this side though what happens when we move the centers? So could it be the case that when we move the center to some other place that the error goes up? >> No. >> And why do you say that? >> Because the average is going to be the best way to represent a set of points. On average, we should be able to demonstrate that. >> I think we already have. We did this earlier in,

in the course when we were talking about minimizing the squares. >> You are right. So, basically, you could take that equation and you could just take the derivative of it. Set it equal to zero and it will turn out to be exactly the average. You're right that's exactly right. >> The error equation E that's right, yeah so this is like really kind of neat. When we moved points around (准確地講, 應該是改變點的 partition label) we move it to reduce error and we move centers around we always move it to the center even though this is a continuous space we always jump to the center that actually has minimum error under the assumption that we're holding the partition steady. >> Right. >> So this is just great. >> So put them together, you're guaranteed to be, let's see, what's the math term? Monotonically non-increasing in error. >> Monotonically non-increasing in error, very nice. And does that imply that thing has to converge? Could we be monotonically non-increasing in error forever? >> You could, in some worlds, but not in this world. I, I think I could argue that (以 下 是 argument). >> Alright. >> So a monotonically non-increasing function, is a function that never gets bigger. So you could end up in a case where you hit some point, like say zero error, and you keep going. So, why wouldn't that happen here? Here's the argument. You ready? >> Sure. >> There are a finite number of configurations. >> Hm. >> There have to be a finite number of configurations 'cause there's a finite number of objects, and there's a finite number of labels they can have. >> Mm-hm. >> And once you've chosen a label for a bunch of the objects, the centers are defined deterministically from that. >> Right, so even though it is an infinite space as we're tick-tocking back and forth, if we don't move the partitions then the centers are going to be where they were. So, the centers are quite constrained even though it's continuous. >> Right, so, the only tricky part to that is that you could have a point that can go to either of, let's say, two partitions, because the distance is the same. So you have to have some way of breaking ties, such that you always get the same answer. For example, I will just say that if I, as a point, can go to any of two partitions, I will pick whichever one has the lowest number. >> Good idea. So breaking ties consistently and you gave a particular role for that is going to guarantee that we at least don't kind of spin around not improving. >> Right so let's see if what I just said makes sense. So tell me if you buy this, they have a finite number of configurations. If I always break ties consistently and I never go into a configuration with a higher error, then at some point. Basically, I will never repeat configurations. I'll never go back to a configuration that I was at before. And at some point, I'll have to run out of configurations because there are a finite number of them. >> Yeah, nicely done. >> So it converges, in finite time no less. >> Finite time. Could it be exponential time? because there is a lot of possible partitions. >> So how many different configurations are there, there k^N (因為每個 object 都可以被放进 k 個 partition 中的一個) because you can assign K to the first object, K to the second object. K to the third object, so k times, k times, k times, k times, k all the way up to n, so that's k to the n. So, that's a lot of possible configurations, but regardless, there's a finite number of them and I suppose in practice, it's not like you would look at every single one of them. Because you're going to jump around very quickly because, of the way distance metrics works. They point close together, they're always going to be close together. So you're never going to try, assigning each one to all possible configurations if they're close together. >> Yeah, it tends to converge pretty fast. So let's summarize some of these properties. >> Okay.



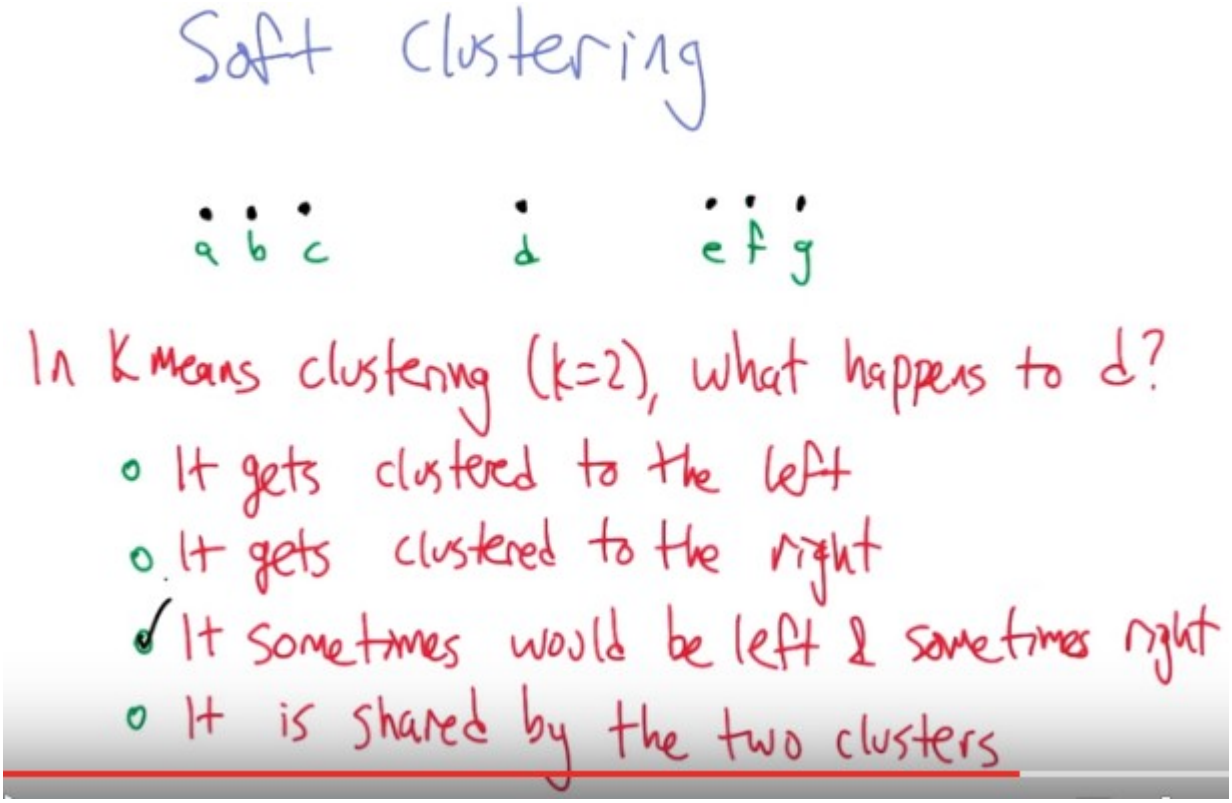
16. Alright, so let's summarize some of the properties that we've been talking about just so that they're actually written out. One is that each iteration as we go through this k-means process. Each iteration is actually polynomial. It's pretty fast. We just have to go through and look at the distance between each center, each of the k centers and each of the n points, to reassign them. And then we have to run through all the points to redefine the clusters. There could be an extra factor of d in here, if these are d dimensional points. >> Sure. >> That makes, that makes sense, alright? Then you were just going through the argument as to why the number of iterations would be finite and exponential. And the exponential is like k to the nth. And what you said was the first of the n objects can be in any of k clusters, and the second can be in any of the k clusters, and the third can be in any of the k clusters. So we get a total of k to the nth different ways of assigning points to the k clusters. >> But I do think what you said in response to that is right which is in practice, you're not going to do an explanation under iteration. >> Right right, in practice it tends be really, really quite low. It just kind of clicks into place. >> Because it's a, it's the same thing for reason why the average is, even though it's a continuous space, it's still finite. Distance is a really, really strong concern. >> Right, and so the error on each iteration is going to decrease if we break ties consistently. And as you've been pointing out to me, there is one exception here whereas if things are assigned to clusters, it could be that things don't improve, that it stays exactly the same, but that can only happen once. Because then the clusters are going to get assigned according to the consistent tie breaking rule, and the next time through we're going to see that we've converged. But there is something that we didn't talk about that I think is important to think through. So here's a set of six points. >> Okay. >> If I was going to ask you to make three clusters of this, what would you do? >> I would put a and b together, c and d together, and e and f together. >> Indeed. Alright. So that is the, that's the optimum here. So, the question is, write down a way of assigning the three cluster centers to points, so that it will be stuck there and not get to the, the clustering that you just found. So, just write down a, b, c, d, e, f. Three of them separated by commas

defining where the centers should start so that it will actually not make progress. >> Okay.

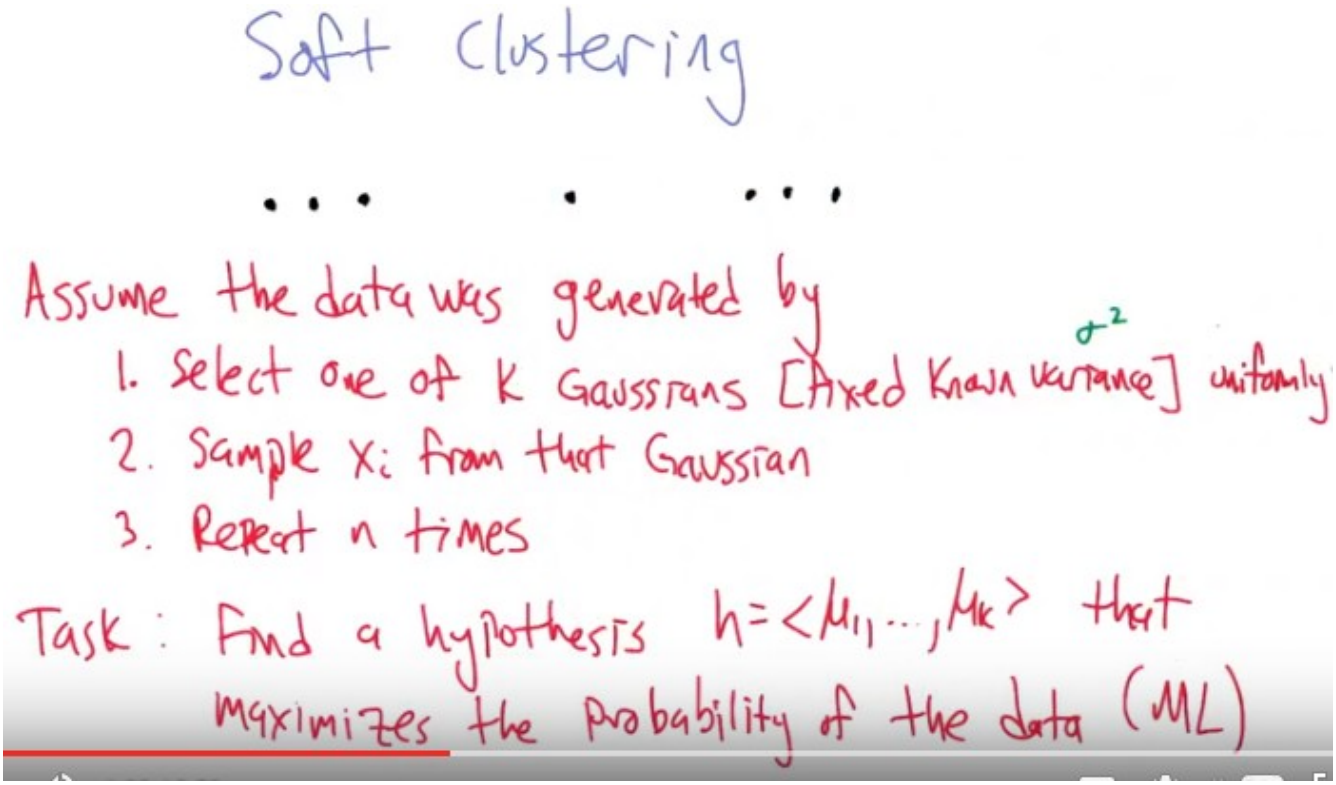


17. Okay, so what do you think? >> I think the answer would be a and b and any of those four: c, d, e or f. >> Alright. So let's think about what would happen in that case. So, if we start off the centers there. The first step is going to be for every point to join whichever cluster it's closest to. So, a is just going to be with a. >> Mm-hm. >> B is just going to be with b. And then d is going to slurp up all these other four points. >> Right. >> All right. So now in the next iteration, it's going to recompute the centers. And a and b aren't going to change. This cluster (右邊那個), the center's going to change to here (四點中心). And now it's never going to make any additional progress. So those are the three clusters it'll find if it starts off with those kind of bad initial points. >> So how would we go about avoiding that sort of thing? >> yes. I was going to ask you exactly that question. So, given that we're thinking about this as a hill climbing process, that's a local optimum. And we had a pretty good way of dealing with local optima and that was random restarts. That's one possibility. Another thing that people sometimes do is they'll do a quick analysis of the space and actually try to find initial cluster centers that kind of are spread out. So pick a cluster center and then pick the next cluster center to be as far away from all the other cluster centers as you can. >> So kind of do like the, I don't know, the convex hole of the space and try to pick points near the corners or something like that? >> Yes, yeah, that's right. >> Hm. So I think you've done another thing too, now that I say that out loud. Which is, you've been choosing these random places to start your centers as always being one of the points. I guess you didn't have to do that. >> Yes. >> But it probably helps that you do. >> It certainly keeps the centers near the points. Another thing you can do to get things started is just randomly assign everybody the clusters but that can often have the property that all of the centers of those clusters end up being kind of really close to each other (why?). So, by picking points to be the clusters it does have a tendency to spread things out. >> Okay, that makes sense. >> That's not a proof, though. >> No, a proof by that makes sense. [LAUGH]

18. All right. We're going to talk about another clustering method. And just to transition from the previous one, let's, let's take a look at this data set here. We've got seven points. A, b, c are all close together on the left. E, f, g are all close together on the right. And d is equidistant from, say c and e. In the k means clustering setting. If we're looking for two clusters. What's going to happen to d? So, one possibility is it ends up with the group on the left. Another possibility is it ends up with the group on the right. Another possibility is that, depending on what happens from the random starting state, it might end up on the left or the right. And then finally, it's shared between the two clusters. because it doesn't really belong in either of them. So it's sort of going to partly join both. >> Oh, I see. So d is exactly in the middle between c and a. Okay. >> That's what I intended to draw, yeah. >> Okay, good. I think I can figure out the answer.



19. Alright. So what do you think? >> I think the answer is, three. It sometimes would be left and sometimes would be right, depending upon where you randomly start. So for example if you start with your random centers on a and b. Then I think what'll happen is, d will end up on the right, as that point gets dragged over towards the weight of the right, it'll drag d with it. If you started out with both points on f and g, then I think d would get dragged to the left cluster as the left most cluster got dragged to the left. I think if you started it with a and g, then it would depend upon how you break ties. And since I always break ties lexiographically, because I like saying the word lexiographically, it would end up on the left. So I think the answer has to be the third choice. But, I'll say one thing, I wish it were the fourth choice. >> I would like to grant this wish by talking about the next clustering algorithm which, in particular, does soft clustering. And, soft clustering allows for the possibility that a point can be shared you know, I'm a little bit of this, I'm a little bit of that. I'm a little bit of country, I'm a little bit of rock and roll.

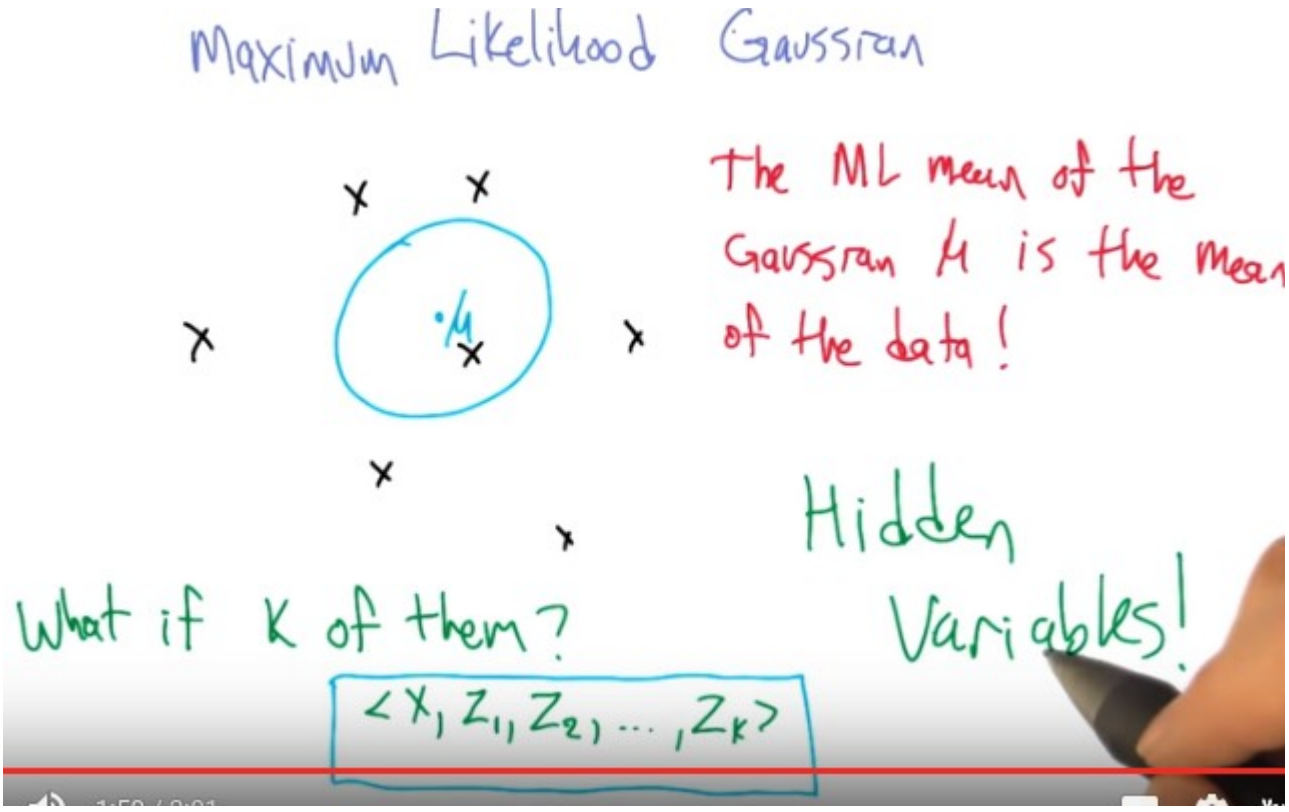


本方法的意思就是: 假定有 k 個 Gaussian 分佈(σ^2 都一樣, μ 不同), 一個 cluster 中的數据就是來自一個 Gaussian 分佈. 現在我們要像 Bayes 一樣, 知道 結果 求 原因的概率. 即我們假定這 k 個 Gaussian 分佈的 μ 為<μ1, μ2, ...μk> (這就是一個 hypothesis), 要求出在這個 hypothesis 的條件下, 出現這些 data 的概率. 更准確地講, 是要求出一個 hypothesis <μ1, μ2, ...μk>, 使得現這些 data 的概率最大.

20. So to do soft clustering, we're going to use a similar trick to what we've used in some of the other lectures, which is to lean on probability theory so that now points instead of coming from one cluster or another cluster can be probabilistically from one of multiple possible clusters. Does that seem like a good idea? >> It does. I feel a song coming on. Lean on probability when you're not strong. [INAUDIBLE] one thing. No, that doesn't work. >> Yeah, that almost worked. >> Almost. >> Yeah. So that's because it's soft clustering or soft assignments instead of hard ones. >> I like it. >> All right. So to do this, we're going to have to connect up a probabilistic generator of process with the data that we actually observed. So let's assume, and there's many ways to go down this road, but we're going to go down the road this way. Assume that the data was generated by, what happens is we're going to select one of k possible Gaussian distributions. So we're going to imagine that the data's going to be generated by draws from Gaussians (Gaussian 即 Gaussain distribution 的意思), from normals. >> Mm-hm. >> Let's assume that we know the variants, sigma square, and that the K Gaussians are sampled from uniformly. >> Okay. And then what we're going to do is that given that Gaussian we're going to select an actual point, an actual data point in the space from that Gaussian. And then we repeat that n times. So if n is bigger than k then we're going to see some points that come from the same Gaussian, and if those Gaussians are well separated they're going to look like clusters. >> Assuming they have very different means. >> Alright. That's what I mean by, we'll separate it. Yeah exactly so. >> Oh, yeah, yeah, yeah, okay. Good. >> Alright, and in particular, what we'd like to do now is say, alright, well, now what we're really happening is, we're given the data, we're thinking kind Bayesianly (即知道

結果, 求原因的概率), right, we're given the data and we want to try to figure out what the clusters would have been to have generated that data. So we're going to try to find a hypothesis, which is this case is just going to be a collection of k means, not to be confused with K-means. >> Mm. >> That maximizes the probability of the data. Right? So find me k mu values, which are the means of those Gaussian distributions. So that the probability of the data given that hypothesis is as high as possible. And this is an ML hypothesis. And ML of course stands for Michael Lipman. >> I don't think that's right. >> Machine learning. >> That's closer, but not quite right. >> Maximum likelihood. >> That I think is correct. >> Alright. So that's now the problem setup. I didn't actually give you an algorithm for doing this, but presumably it's going to depend on various kinds of things and probability theory and optimization, but is it sort of clear what we're shooting for now? >> It is, it is. And I actually think the fact that we're looking for k means probably means that we are going to end up tying it back to k means. Maybe so, but again, it's a softer kind of k means, it's a softer, gentler kind of k means. >> Mm. >> I think some people call it K Gaussians. Does that sound right? >> No. >> Alright then. >> I mean it sounds correct, but it doesn't sound right. >> [LAUGH] Alright then.

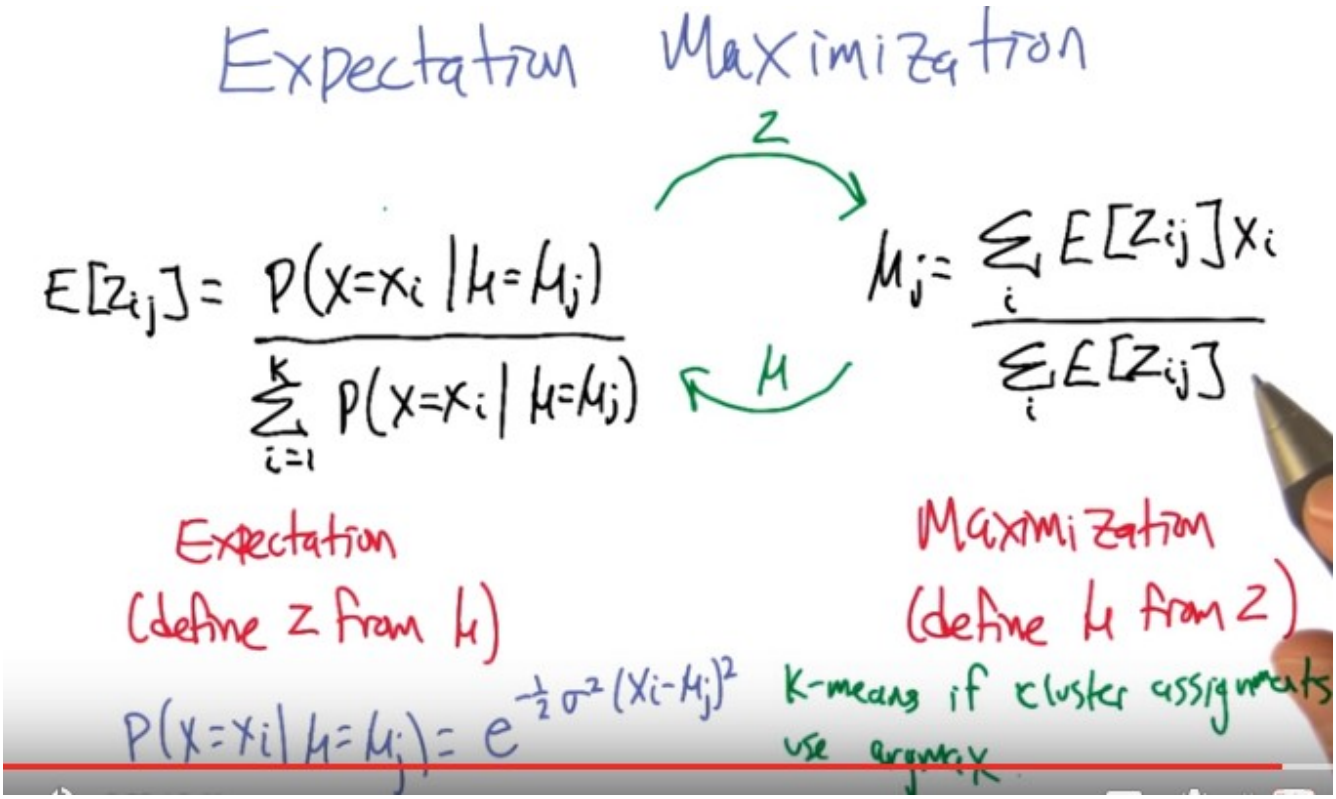
重要: 以下對 EM 算法的講解完全是故弄玄虛, 千萬別看! 直接看 Big Data 這課的 note “7. Clustering”中對 EM 算法的講解, 那裡是小清新, 講得很好.



先看此句話, 就明白是甚麼意思了: If we know that all this data is coming from the same Gaussian, then finding the mean that maximizes the likelihood is just computing the mean of all the points.

21. >> So one of the things that is going to be helpful as a subcomponent to this process is to be able to say. Well, if we've got a set of points in some space, and we're trying to find the (只找出一個) maximum likelihood Gaussian with some known variant. Mode is the maximum likelihood Gaussian. What is the best way of setting the mean to capture this set of points? So fortunately this is really easy to do. And

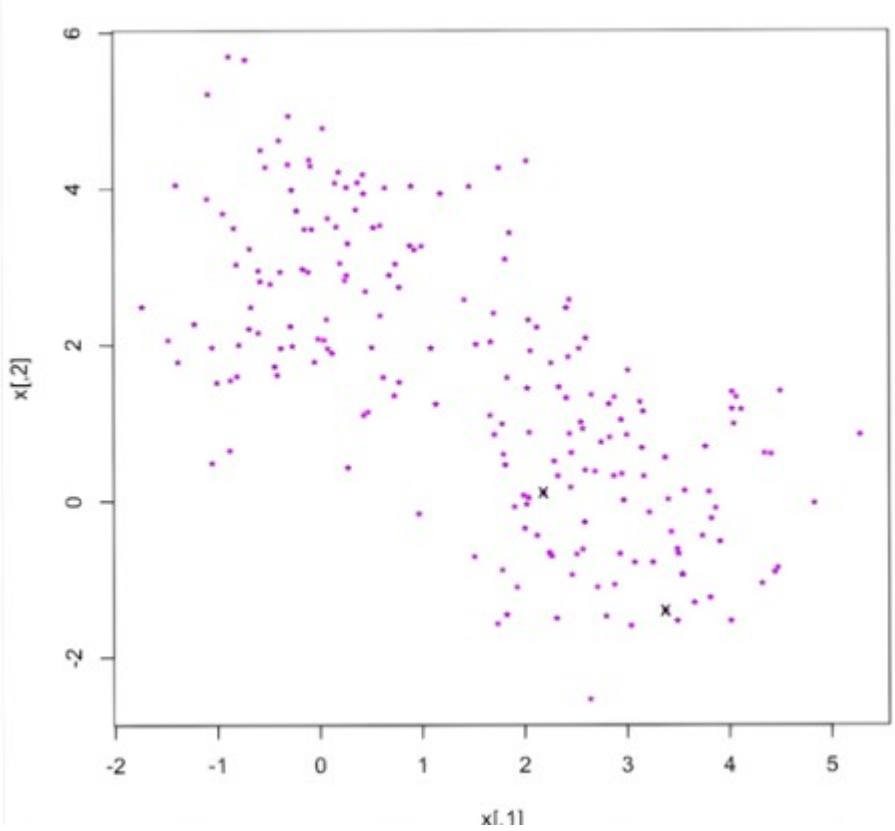
the reason that it works out this way is the same reason that we've talked about in several of the other lessons. But the maximum likelihood mean of the Gaussian, this mu that we want to set, is the mean of the data, the mean is the mean. >> That's pretty mean. >> And it's no mean feat that it works out this way. >> [LAUGH] >> And, [LAUGH] what? >> Oh, just well done. >> So, in particular, this is really easy to compute. If we know that all this data is coming from the same Gaussian, then finding the mean that maximizes the likelihood is just computing the mean of all the points. >> Right, we kind of did that, just a few slides ago. >> Exactly. >> Okay, alright, so given a bunch of data points that I all know came from some Gaussian, I can compute the mean of that Gaussian by actually taking the sample mean, and I could mean it, okay. >> So the tricky thing of course, is what happens if there's k of them. How do we set the k different means? And our answer is going to be, there I just wrote it. Do you see it? >> Nope. >> That's because, it's hidden variables! >> Oh. My favorite kind of variables. >> Variables that you don't have to see. So what we're going to imagine is that the data points, instead of just being X, it's actually X and a bunch of random variables that are indicator variables on which cluster that X came from. So it's not just X anymore it's X and let's say a bunch of zeros and then a one. Corresponding to which cluster generated X. Now of course if we knew that, that would be really useful information. We, we're going to have to do some inference to figure out what those values are. But the concept is that, by adding these extra hidden variables in, it really kind of breaks up the problem in a convenient way. >> Okay.



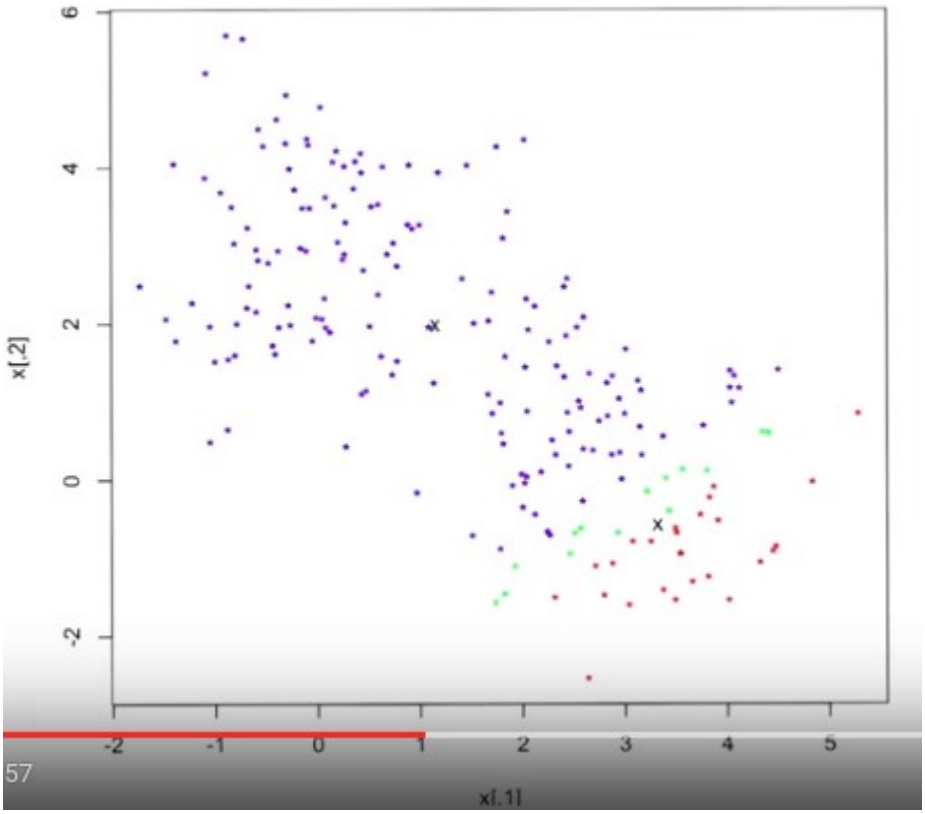
Xi 可以理解為一個數, 即(X1 = 3, X2 = 582...)

22. So, this is going to lead us to the concept of expectation maximization. So, expectation maximization is actually, at an algorithmic level, it's surprisingly similar to K means. So, what we are going to do is, we're going to tick-tock back and forth between 2 different probabilistic calculations. So, you see that? I kind of drew it like the other one. >> Mm Hm. The names of the 2 phases are expectation, and maximization. Sort of you know, our name is our algorithim. >> I like that. >> So,

what they're going to do is, we're going to move back and forth between a soft clustering (算 E[Zij]), and computing the means from the soft cluster. So the soft clustering goes like this. This probablisitic indicator variable, Zij, represents the likelihood that data element i comes from cluster j. And so, the way we're going to do that, since we're in the maximum likelihood setting, is to use Bayes' rule, and say, well, that's going to be proportional to the probability that data element i was produced by cluster j (即已有 cluster j, 則它弄山 data element i 的概率多大, 即 P(X=Xi | μ=μj)). And then we have a normalization factor. Normally, we'd also have the prior (即 P(μ=μj)) in there. So why is the prior gone Charles? >> Well, because you said it was the maximum likelihood scenario (因為 prior 是常數(下句馬上說 assume uniform, 以前講過), 不影响求最大值). >> Yeah, right. We talked about how that just meant that it was uniform and that allowed us to just leave that component out. It's not going to have any impact on the normalization. >> Right. >> So that's what the Z step is. Is if we had the clusters (意思就是下一個 if 的意思), if we knew where the means were, then we could compute how likely it is that the data would come from the means, and that's just this calculation here. So that's computing the expectation. Defining the Z varaibles from the muse. The centers. We're going to pass that information. That clustering information, Z, over to the maximization step. What the maximization is going to say is, okay, well if that's the clustering, we can compute the means from those clusters. All we have to do is just take the average variable value. Right? So the average of the Xi's. Within each cluster J. What's the likelihood it came from cluster J and then again, we have to normalize. If you think of this (E[Zij]) as being a 0 1 indicator variable, then really it is just the average of the things we assign to that cluster. But here, we actually are kind of soft assigning, so we could have half of one of the data points in there, and it only counts half towards the average, and we could have a tenth in another place, and a whole value in another place, and so we're just doing this weighted average of the data points. >> So, can I ask you a question, Michael? >> Yeah, shoot. >> So, this makes sense to me, and I, and I even get that for the Gaussian case, the z i variable will always be non 0 in the end, because there's always some probability. They come from some Gaussian because they have infinite extent. So I, this all makes sense to me. Is there a way to take exactly this algorithm and turn it into k means? I'm staring at it, and it feels like if all your probabilities were ones and zeroes, you would end up with exactly k means. I think. >> I dunno, I never really thought about that. Let's think about that for a moment. Certainly, the case, if all the Z variables (應該是 E[Zij]) were 0, 1, then (the right panel) the maximization set would be the means, which is what k means does. >> Mm-hm. Then, what would happen? We send these means back, and what we do in k-means is we say, each data point belongs to it's closest center. >> Mm-hm. >> Which is very similar actually to what this (the left panel) does. Except that here we then make it proportional (即 E[Zij]在 0 到 1 之間). So I guess it would exactly that if we made these clustering assignments, pushed them to 0 or 1 depending on which was the most likely cluster. Right, so if you made it so that the probability of you being to a cluster actually depends upon all the clusters, and you always got a 1 over 0. Basically you did, this was like a hidden argmax (前面的 hidden variable Zi 應該就是這裡的 Zij) kind of a thing, or a hidden max or something. Then you would end up with exactly k-means. >> I think you're right. >> Huh. >> Yeah, I never thought about that. >> Okay. >> So it really does end up being an awful lot like the k-means algorithm, which is improving in the error metric, this squared error metric. This is actually going to be improving in a probabilistic metric, right. The, the data is going to be more and more likely over time. >> That makes sense.

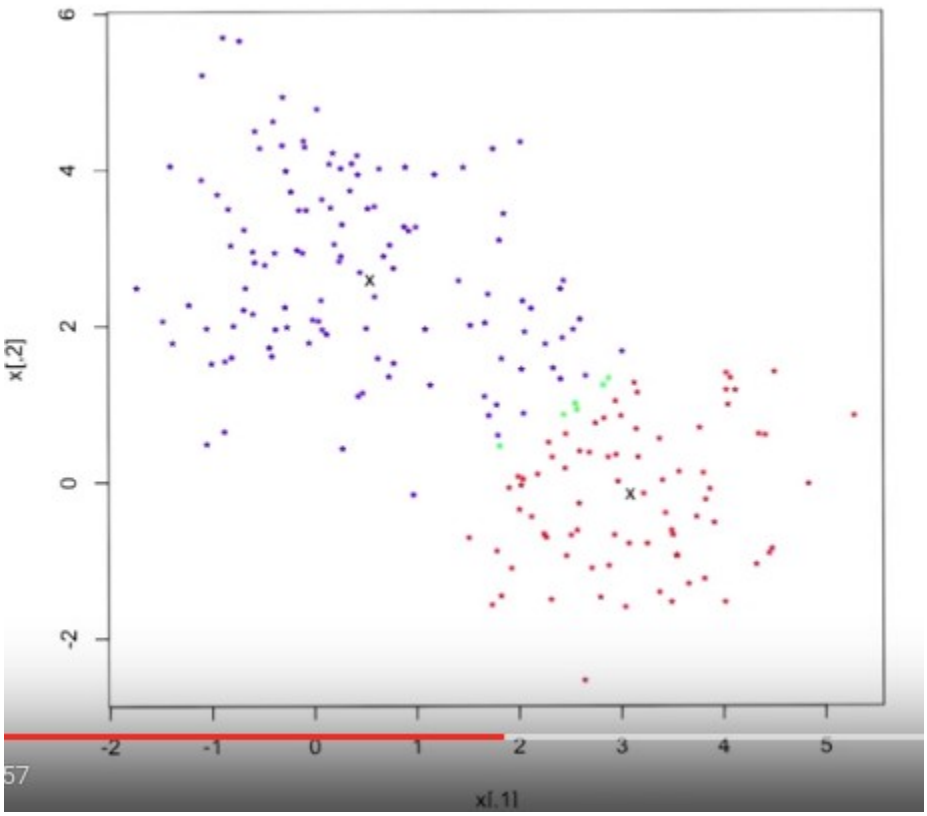


23. So that's sort of EM at the level of the equations, but I thought it would be helpful to actually implement it and kind of mess around a little bit. So do you want to see what happens? >> I would love to see what happens. >> So I generated some data here, which comes actually from two different Gaussian clusters. One that's centered over hereish, and one that's centered over hereish. [LAUGH] Just not to be too specific about it, but you can sort of see that there's two clouds of points, right? One in the upper left and one in the lower right. Can you see it? >> The one in the lower right looks like a clown. I think it's a little bit more of a superhero but that's that's not the point. The point is, is that it's just a bunch of random points. >> Hm. >> That's the point. It's a random point, but it is a point. >> So, your point is that the points are randomly pointed. >> Yeah, I just want to point out that at random. >> I see your point. >> And, so, what I want to do now is run EM and the first thing I need to do is pick the initial centers. Right, so I need to pick the mu's that we think these were generated by. And so a common thing for people to do is just to take two of the points from the data set at random. So I took two points at random and I'll mark them with an x. And now what I'm going to do is run an iteration of EM. And what that means is, I'm going to first do expectation, which is going to assign every one of these points to one of these two clusters. And I'll color them as to which cluster they belong to. And then I will move the centers, re-estimate the means based on that soft assignment. Makes sense.

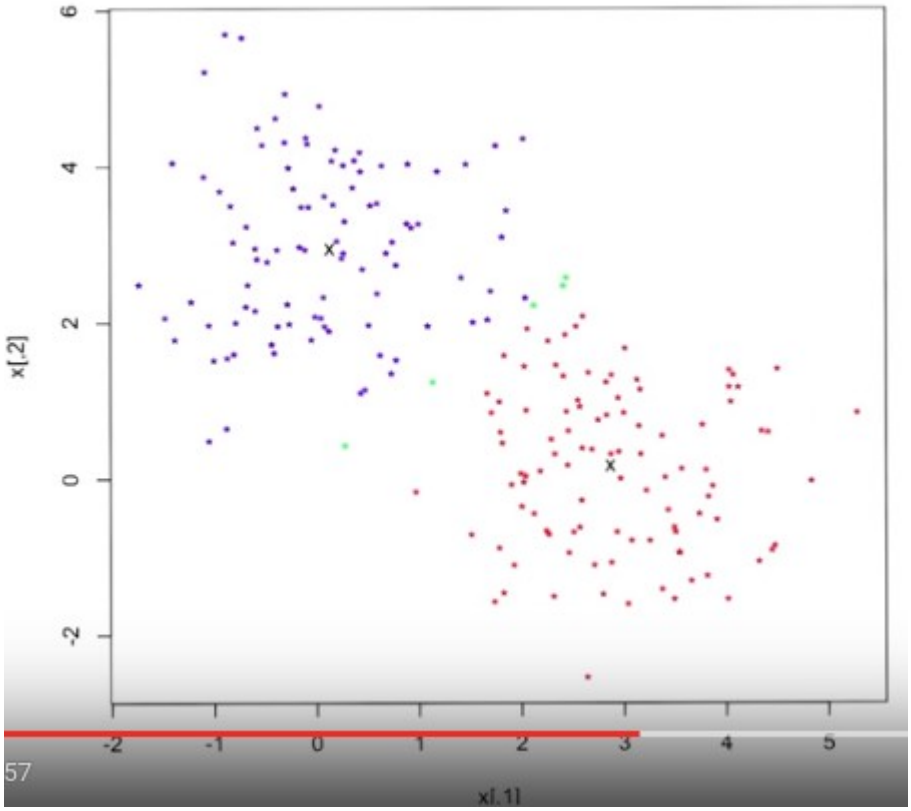


注意此圖中的兩個 x 的位置跟上圖不同, 說明它們已經移到新的 x 去了.

>> Alright, now so what you can see is something kind of interesting happened. Because these centers started off in the same cluster together, many of them were assigned to one cluster, which was the one that was a little bit more above. And then very few of them ended up getting assigned to the lower cluster because very few points were essentially closer to that one than the other one. When I colored the lower ones red and the upper ones blue. And there's this sort of band of 1s in the middle that had intermediate probabilities, they weren't really deeply one cluster or the other. >> They were near 0.5? >> Yeah, exactly, specifically between, including 0.4 to including 0.6. >> Okay that make sense. >> Alright, and so those are the green ones. And then based on the cluster assignments, we estimated the means. So this xis now at the, you know, the mean of the red points. And the green points, because these they're actually shared and this acts as kind of the rest. And you can see, it doesn't really capture yet what the true clustering is because a huge number of the lower right cluster points are blue. They're kind of grouped in with the upper left cluster. But we can run another iteration of the app.



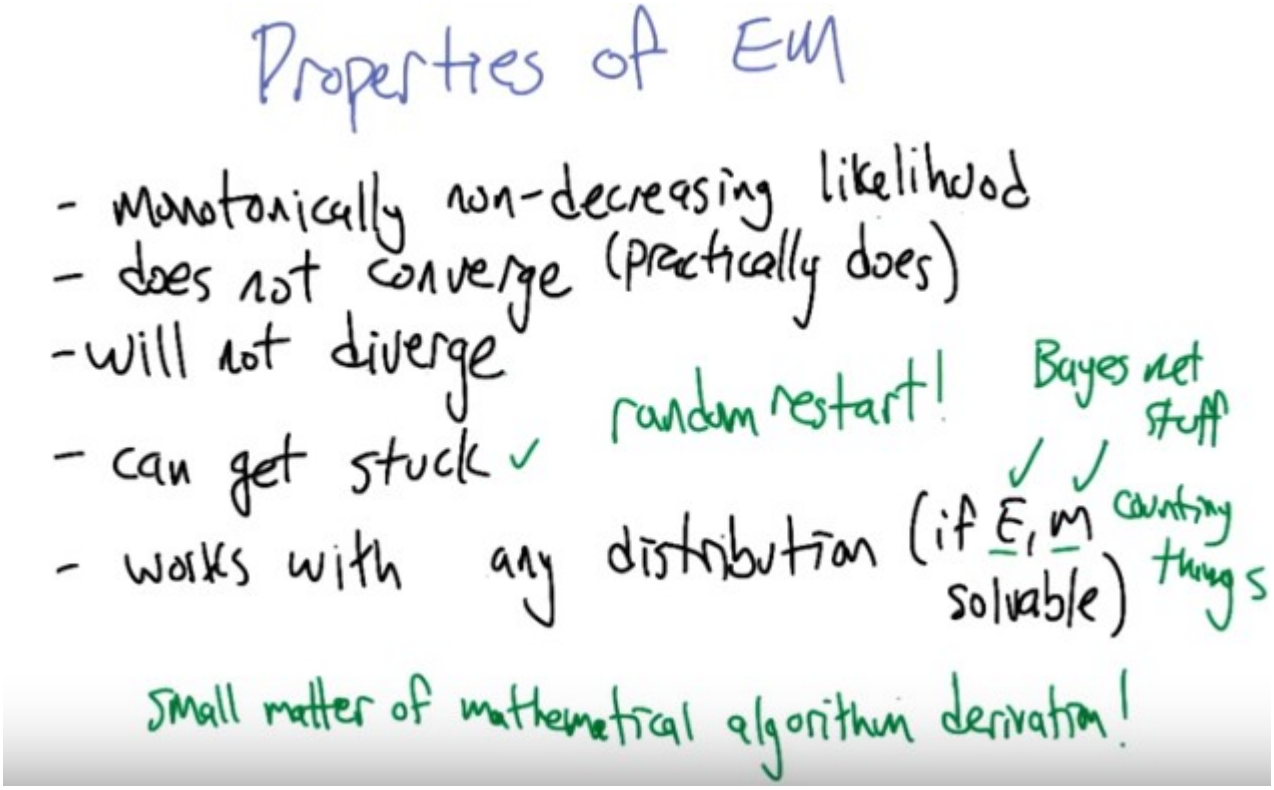
And now things have shifted, right? So now it looks like this lower right cluster has claimed a good amount of the points in the lower right and the blue ones are the blue ones. There's just a few little green scattered between them at the boundary and you always expect some of that to happen. Now, the centers have moved and now they are starting to take on their clusters,



本圖是 converge 後的結果

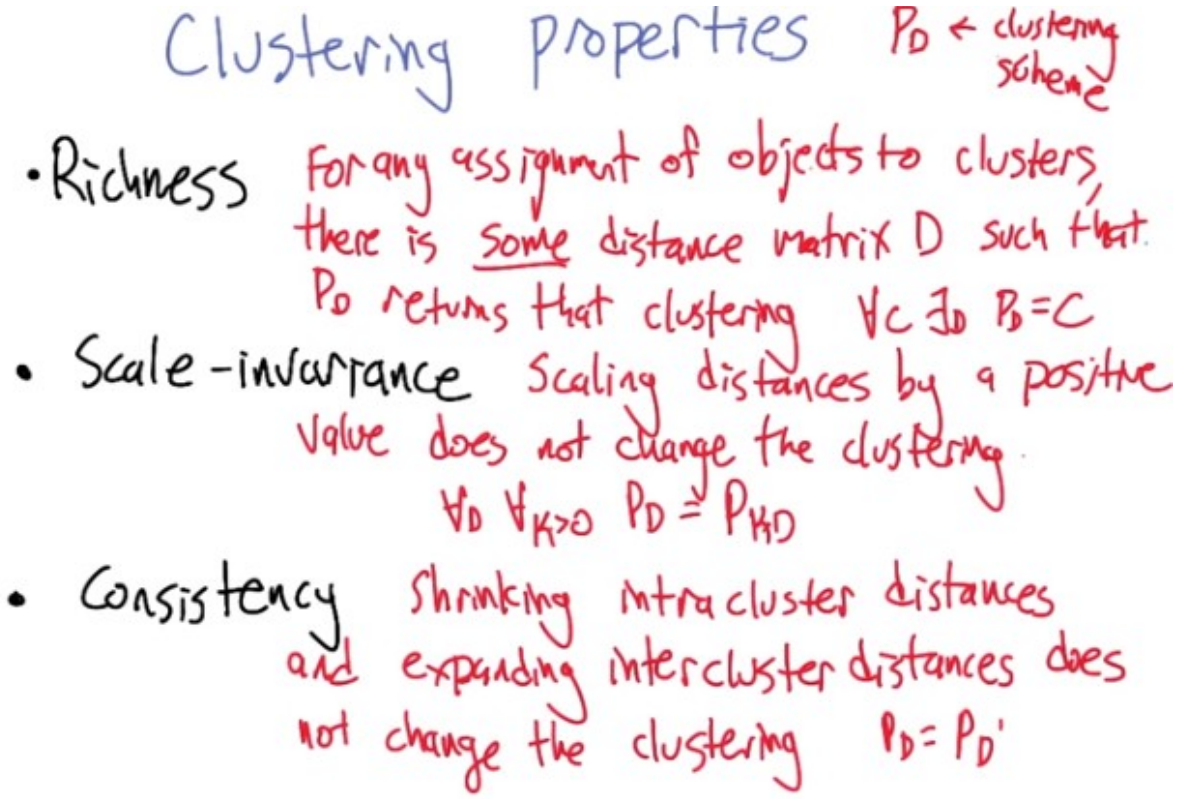
but we just run this a couple more iterations until we see the x is not really moving any where. Alright, seems to have settled down and you can see it really did pull out the lower right cluster, the upper left cluster as a cluster and just a few points at the boundary where it said well I can sort of believe that that's part of the red cluster, I can sort of believe it's also part of the blue cluster, I really can't decide. And you know what? That's probably right too. I mean they could go either way. >> Yeah, exactly. I can sort of hallucinate this green point here as being part of the upper cluster. Just kind of on the fringe. But, you know, it works just as well as being part of the lower cluster. >> Actually Michael I'm kind of curious. Can you, use your magical computer powers to, you know, click on one of those green points and see what the probability is? I could. Yeah, so this, in particular, it's 55% in the first cluster and 45% in the other cluster. >> Hm. >> So, it really is. It's hovering near that boundary. >> That makes sense. What about that red point all the way in to the right? The ones that are all by itself. >> Excellent choice. So, it is actually. 0.9999996 in one cluster. And 1 minus that in the other cluster. This one is pretty certain coming from the second cluster. >> I see two things that came out of this. One is EM is much nicer in that it's not forced to make a decision about where those green points go. So that's sort of soft clustering does that. And that's a good thing. Because we don't have that problem that we had before. But one of the consequences of that, and this is not a bad thing, but it's a thing, is that even points that pretty clearly belong to one cluster or another, given that we're staring at them. They do have a really high probability, 0.9999999996, but they all have some non-zero probability. Of ending

up, of belonging to the other cluster. >> And is that, you think that's a good thing or a bad thing? >> I think it's a good thing. I think it makes sense because Gaussians have infinite extent and even if a point is very, very far away from the center, it has some chance of having been generated from that Gaussian and so, this just tells us what that chance is, it is very, very unlikely. But it is still as non-zero probability match. >> In, in some sense, it's acknowledging truth. Right? Which is that you can't be sure which of the two clusters, this came form, even if I tell you where these clusters are (即上頭的 cluster 也有可能弄出一個在下頭的點). >> Right? >> Yeah, okay. I, I agree that. >> Okay, this is cool. So, EM is nice. So, does EM work for anything other than Gaussians? It does, it actually can be applied in lots of different settings. Let's flip over and talk a little about some properties of EM. >> Okay.



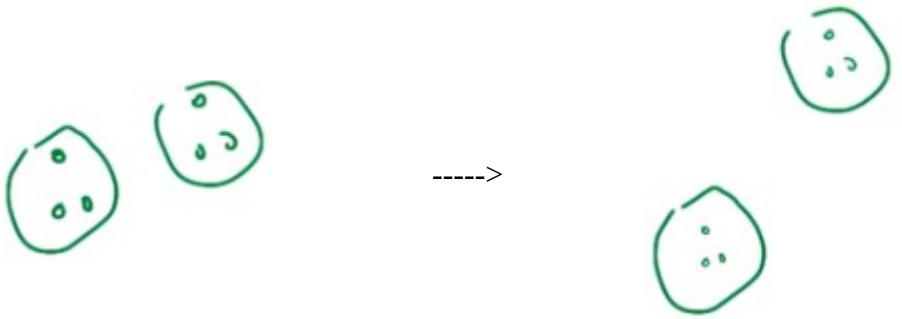
24. So we talked about the equations that defined expectation maximization, and we stepped through an example with some actual data in the sense that it was data points. They weren't actual, measured data points. But what I'd like to talk about now for a moment is some of the properties of the EM algorithm more generally. So one of the things that's good about it is that each time the iteration of EM runs, the likelihood of the data is monotonically non-decreasing, right? So it's not getting worse. Generally it's finding higher and higher likelihoods and it's kind of moving in a good direction that way. Unfortunately, even though that's true, it is not the case that the algorithm has to converge. I mean have you ever seen it not converge? I've never seen it not converge. >> No, because usually there's some kind of step that you take and you just, you make the weight lower and lower. So yeah, or something like that. You know, no, I've never seen it not converge. >> So it doesn't have to converge. I think you can construct really strange examples that make it do that. But on the other hand, so even though it doesn't converge, it can't diverge, right? It can't be that these numbers blow up and become infinitely large because it really is working in the space of probabilities. And it's, it's pretty well behaved as far as that's concerned. >> That's a difference between in k means, right? So, the argument for k means, if I recall, which feels like was about a week ago, when we talked about this. [LAUGH] But of course, it was, it was merely seconds ago. Is that (in k-means) there is a finite number of configurations and k

means and since you are never getting worse in our error metric. So long as you have some way of breaking ties, eventually you have to stop. And, so, that's how you got convergence, right? >> Yeah. >> So, in EM, I guess the configurations are probabilities and I guess there is an infinite number of those. Yet you can do more than guess. >> So in fact, there are an infinite number of those. >> Exactly. >> It wouldn't necessarily follow that you wouldn't converge from that. But that alone is, is one big difference, I guess, between the k means and the, the EM. That's the trade off you get for being able to put probabilities on things. So you've got an infinite number of configurations. You never do worse but you're always trying to move closer and closer. So, I guess what could happen is you could keep moving closer every single time, but because there's a infinite number of configurations, the step by which you get better could keep getting smaller, and so you never actually approach the final best configuration. I suppose that's possible. But for all intents and purposes, it converges. >> Right. Exactly. However, it can get stuck. And this you see all the time. I almost never not see it do this. Which is to say that, if there's multiple ways of assigning things to clusters, it could find a way that doesn't have very good likelihood but can't really improve on very well. So there's a local optima problem that is pretty common, and so what do we do when we, get stuck in local optima with a randomized algorithm? >> Cry. >> No. >> We take all of our toys home and randomly restart. >> There we go. >> Okay. >> And the last thing to mention is this, is, is what you just suggested a moment ago in the previous slide. Which is that, it's nothing, there's nothing specific about Gaussians in here. It really is an algorithm that can be applied anytime that we can work with probability distribution. And so there's just a ton of different algorithms that work in different scenarios by defining different probability distributions, and then all you have to do is figure out what the E step and the M step are. How do you expectation to work out the probability of the latent variables. And then, how do you do maximization to use those latent variables to estimate parameters? And usually it's the case that it's the estimation that's expensive. It's difficult because it involves probabilistic inference. Right? So it's just like Bayes net stuff. >> Mm. >> And the maximization is often just, you know, counting things. But it isn't, in general, the case that it's always harder to do E than M. There's some well-known examples where M is hard and E is actually quite easy. You know, for any given problem you have some work to do to derive what the E and the M steps are. But it's very general, it's a really it's a good tool to have in your toolbox. >> I like that. So, basically it's not that hard because it's just a small matter of math programming. >> Indeed.



25. Alright. So that's all the algorithms that we're going to talk about in terms of unsupervised clustering algorithms. But I would like to kind of pop up a level, and talk a little bit about different properties that clustering algorithms might have. Desirable properties. So, the three that I'm going to talk about are richness. Scale-invariance and consistency, and these are good things to have right? I'd like to be rich and consistent. >> And Lord knows, you'd like to be scale-invariant. >> I would. I wish I were scale-invariant. I guess it would be very hard for me to gain weight if that were true. >> Mm-hm. No matter what the scale is, the number's always the same. >> Alright, so if you have a clustering algorithm, what does it do? It takes a set of distances d and maps them to a set of clusters. >> Or partitions. >> Right. Or a partition. Right. And these are three properties of those kinds of mappings. So richness is the idea that for any assignment of objects to the clusters, there's some distance matrix that would cause your clustering algorithm to produce that clustering. For any clustering that you want to achieve there is some distance matrix where p of d, your clustering algorithm, your clustering scheme, produces that clustering. >> So that's like saying, all inputs are valid and all outputs are valid. >> All inputs, which is to say that the distance matrices, sure, those are valid, and anything could be an output. Any way of clustering could be an output. because the, you know, think of the alternative. The alternative is there are certain clusters that you just can't produce. And that seems wrong, doesn't it? That, it ought to be the case that your clustering algorithm should produce whatever is appropriate, and shouldn't be limited in what it can express. >> Sometimes. You'd even want your algorithms to say, you know what, there's only one cluster here. >> Yeah. I mean, totally. If I showed you a picture, you might look at it. And you'd say I just see one cluster and it looks like a cloud. The second property that we're talking about, the scale invariance. And this is, I think, much more straight forward, at least in terms of conceptually. So, it just means that if I give you a distance matrix and I double all the distances or halve all the distances. It shouldn't change what the clustering is. That the clustering algorithm should be invariant to what the space of the point is, assuming that we keep the relative distances the same. >> So this is the NASA problem. So this says that, if I come up with a bunch of

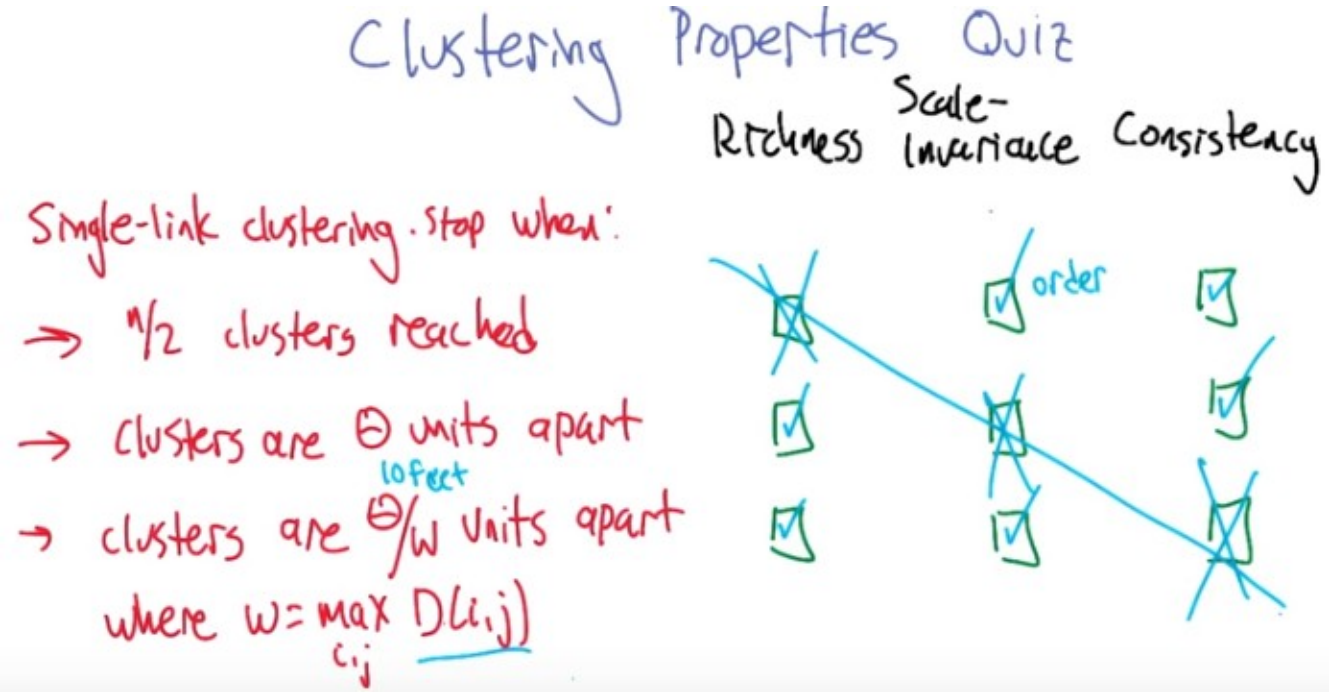
clusterings because I've been measuring points in miles, if I suddenly start measuring them in kilometers, it shouldn't change anything. >> That's right, yeah, change of units. Yeah, that's a really good way of thinking about it. It's not even that I'm scaling the distances, I'm just using inches instead of feet. It should be the case that it's still the same data. All right.



And then the last one (consistency) is maybe the hardest one to visualize. But, it's a really reasonable thing. And you would expect clustering to act this way. So, consistency says that if we have some clustering. If your clustering algorithm produces some clusters. So, let's, let's do a little example of that. Then, shrinking the intracluster distances and expanding the intercluster distances, does not change the clustering. Let me show you that. All right. And now I've edited this so that within the clusters the points have gotten closer together. More so in this one than, than the other. Or not changing them at all. But in this particular case, I shrunk this one a lot, I shrunk this one a little. And then I moved the clusters a little bit farther apart. This changes the distances a bit and we like our clustering algorithm to continue to consider these clusters, right? It shouldn't introduce new clusters because we've shrunken things. And it shouldn't want to join these things (那兩個 cluster, 視頻就是這樣, 不用 double check) together, because they've gotten further apart. So, that's this notion of consistency and a little bit more cumbersome to describe, but very natural thing to think about in the clustering setting. >> Well that makes sense right? So, this is where our domain knowledge comes in so, distances are a measure of similarity, right? >> Yup. >> So, what consistency says if you found that a bunch of things were similar and a bunch of other things were dissimilar. That if you made the things that were similar, more similar and the things that were not similar, less similar. It shouldn't change your notion which things are similar and which things are not. >> Yeah, which things are alike, which things want to be grouped together. Yeah, uh-huh. >> Hm. >> Good so you think you understand these three clustering ideas? >> I believe I do. >> Alright, so let's put them into play a little bit. >> Okay.

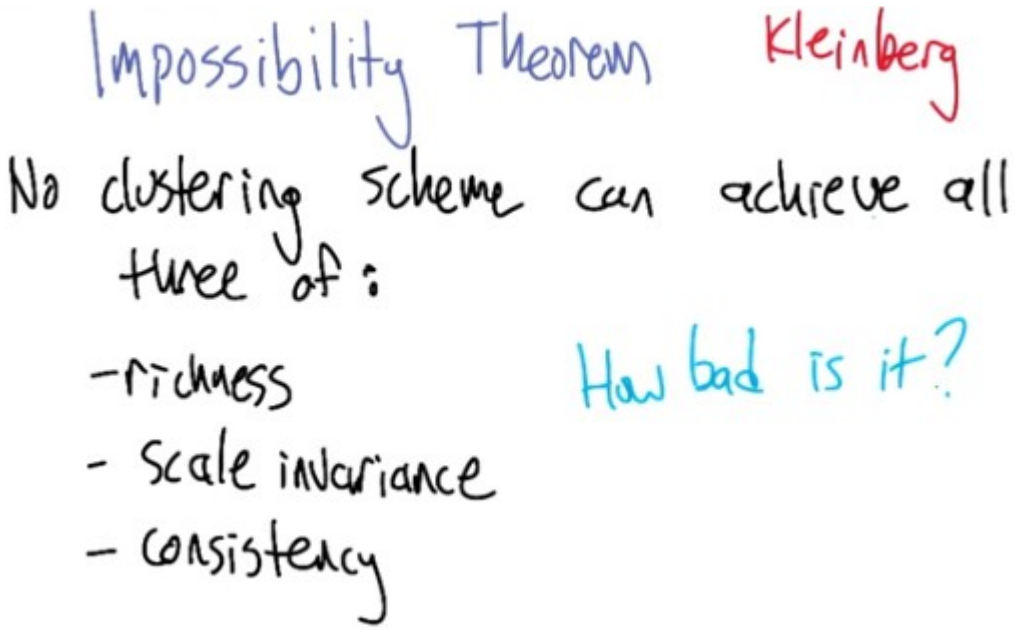
26. And we'll do that with a clustering properties quiz. >> Oh yeah. >> All right, so what I'm going to do, is I'm going to give you three different clustering algorithms. For each one, ask whether it has these properties. Does it have richness? Does it have scale-invariance? Does it have consistency? And so the algorithms are all going to be variations of the first clustering algorithm we talked about, single-link clustering. And so, what we're going to do is we're going to run single link clustering, but to make it a clustering algorithm we have to decide under what conditions are we going to stop building our clusters? And I've got three different conditions, then that defines our three different algorithms. So one is, we've got n items that we're clustering. I'm going to stop when I've got n over 2 clusters. 'Kay? So just keep merging, keep building clusters until you've reached n/2 clusters, and at that point, stop and return what you've got. >> Okay. >> Does that, does that make sense? >> Yes. >> All right, and you remember enough about single-link clustering for that to be meaningful, but that's, it's where were going to start off with everything in its own cluster, and then merge them together by whatever two clusters are closest together, and then iterate. >> Yes. >> Okay. All right, so that's algorithm one.

We're going to stop at n over 2 clusters. The second one is we're going to have some parameter θ, and we're going to keep merging clusters until we'd have to merge clusters that are theta units apart. And once they're theta units apart, we're going to say, nope, that's too far to be part of the same cluster. We're done. >> Okay. >> Okay? So that, again, it's a clustering algorithm, right? It's going to take these distances, and it's going to turn it into groups. >> So that's like, only things that are within ten feet of each other could possibly be clusters. >> Exactly. >> Okay. >> Right. Theta is going to define that. And the last one is very, very similar. We're going to keep doing clusters until we'd have to merge clusters that are farther than theta over ω units apart. And ω in this case is going to be defined to be the largest pair-wise distance over the entire data set. >> That's an omega? >> Yes. >> Okay. >> And that's a capital D, at least now it is. >> [LAUGH] Okay. >> All right, good? So if you understand these algorithms, what, what I'd like you to do is say which of these have the richness property, which of them have scaling variants, which of them have consistency.

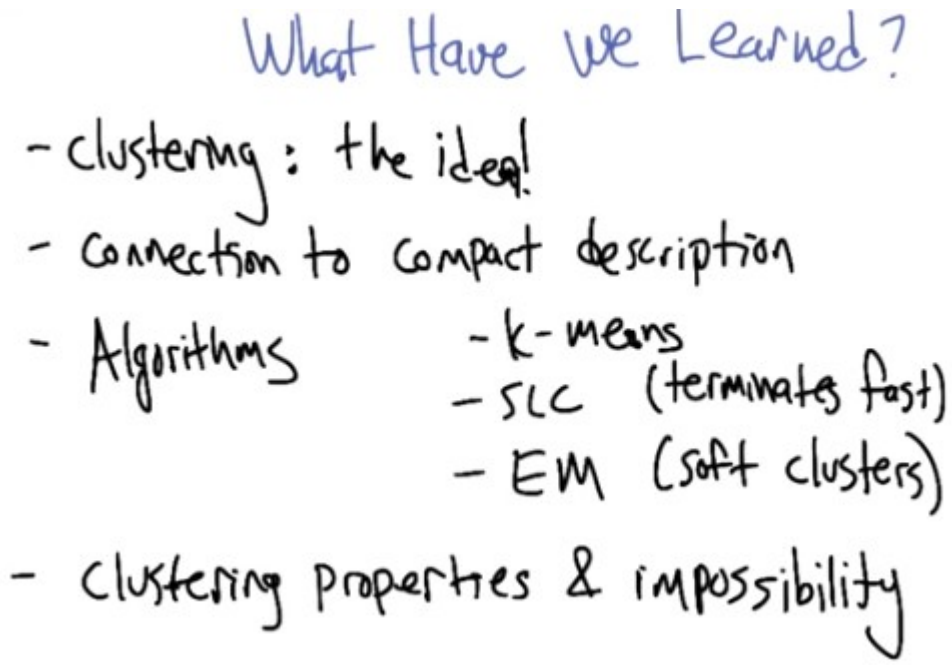


27. All right Charles let's see what you think. >> The first one says we want to have a fixed number of clusters. Well actually the first thing I'll note about that, since we're going to have a fixed number of clusters in doesn't have the richness property. >> Good point. >> Because richness would allow for one cluster or it could have n clusters or it could have n over three clusters or it could have n over two clusters. But here you forced it to always have n over two clusters so it can't represent all possible

clusters. >> Agreed. >> However, you'll notice that there's nothing in here that cares about distances. In the sense that, if I took all of the coints, and I multiplied their distances by two, I would still get the same clusters in exactly the same order. >> That's the important thing. That it cares about distancees but it only cares about the order not the scale. >> Exactly. So that means there's scale-invariance. >> Very good. >> And then by the same argument, this algorithm has consistency because the points that were, clustered together if they got closer together, they would still be picked up. And the ones that were farther apart, well, they'd still get picked up by each other and it, it doesn't matter. So they're definitely consistent. >> That's right. And it's a nice little property of single-link clustering. Let's move on to the second clustering algorithm. >> Nice. Okay so clusters that are theta units apart, well, since theta can change, even if theta's fixed the, the points that I get. Could be various kinds of distances. So, let's imagine that theta was ten feet. If all the points are within ten feet of one another, then they would be one cluster. >> Yup. >> If on the other hand, all the points were more than ten feet apart, you would have N different clusters. And, you could do any of them in between. So, this is rich. >> It is indeed. That's right. We can always muck with the units. Or muck with theta for that matter. So, that we can group the beginning of our clusters in any. Accommodation that we want. >> Yeah, but for exactly the same argument that they're rich, they're not scale invariant. >> Yeah. >> Because I could just take everything and multiply it by theta, multiply it by the distance by theta and now I will suddenly have n then if I had n to begin with, I could divide by theta, and then I would have one. So it's not scale invariant. >> Agreed. >> But the consistency argument still works, because all the points that got clustered together. Because they're within theta of each other. If I made them closer, would still be within theta of each other. And the ones that weren't closer together because they were more than theta apart, would now be even more theta apart and so you do get consistency. >> Agreed. >> Excellent. Ok. >> Alright, last example. >> Clusters that are Theta W units apart where. I'm sorry, Theta Omega units apart. [LAUGH] >> θ divided by ω. >> Yeah. >> Where Omega equals the maximum distance. Well, that's just a way of normalizing distance. In much the same way that I argued for richness of the second algorithm, the same argument applies here (不用管 ω, 只改 θ 即可). >> That it is rich. >> It is rich, because I can just keep shrinking and moving the points around, and it will work out just fine. And so, it's definitely rich. >> We don't control omega, because that's determined by the distances, but we can change the theta so that things are too close or too far. Yeah, okay, I agree with that. >> That's sort of the last thing you said for the second algorithm too. So you do get richness. Now, what's interesting to me here is that unlike in the second algorithm where you didn't have scale and variance, you do get scale and variance here because if I try to make things bigger. I'm also going to make the maximum distance bigger by exactly the same amount and so I will always end up back where I was before. >> Yeah. So whatever you do to scale this, it's going to get undone. Yeah. Very good. >> Right. Exactly. Anything I do to make it, make them far apart will just make them the same distance (因為判斷的標准是 θ/ω, 所以 distance 其實也是 normalized 了的). But at least by omega. >> Agreed. Okay. And, and if we have consistency too, we've got a trifecta. >> We do, except, we don't have consistency. >> What? >> Because if I make the, the clusters that are farther apart, farther apart, then I also change omega. >> And that could actually change the cluster. >> It would because, in fact, imagine that I made the points in a set of clusters. Much closer together, but then I move that cluster, you know, sort of infinitely far apart from the rest of the clusters. Then, suddenly my theta divided by infinity, or near infinity, would make the, the radius of allowable clusters so small that no points would be able to cluster with any other point. And so that would screw up whatever you had before, assuming you had clusters before at all. And so, I can construct a world where consistency would be wrong. Oh that's nice, it made a little diagonal of X's. >> [LAUGH] And I win, three in a row. >> Well done Michael. So, what little tweak do we have to do to these algorithms to get a trifecta? >> Yeah, that would be great, wouldn't it? And that's the final algorithm that we'll talk about. >> Great.



28. >> Charles, I lied. >> No! >> I'm so sorry. In fact, it turns out that there is no tweak that you could do to these algorithms to make the trifecta, to have all 3of these properties. And, in fact, there is no clustering algorithm. It is impossible to derive a clustering algorithm that has these 3 properties. So this is proven by Kleinberg, and what he showed is that, once you define these 3 different properties, richness, scale invariance, and consistency, they are mutually contradictory in a sense. So, just like we saw in the example that we went through in the quiz, where we tweak the algorithm and it gets us one but it loses another, that's a necessary property. You just can't have all these 3 hold at once. >> That's pretty surprising coming from Kleinberg, because John is one of the nicest people I know in machine learning and theory, and you would think that he would have tried to find a possibility theorem, not an impossibility theorem. I'm very disappointed. >> See, he's got a dark side, is what I'm saying. This is a striking and maybe even upsetting result, right? It's saying that, if you actually sit down and say, well, here's what I would like my clustering algorithm to be, which people hadn't really done very often, once you've bothered to go do all that hard work of saying what matters to you and what doesn't matter to you, it turns out you can't have what you want. You can't always have what you want. >> But, can you get what you need? >> In this particular case, maybe. It depends if you only need 2 of these. [LAUGH] >> Well, so how bad is this? So, so, so, I agree, it is, it is a at least to me, a surprising result that you can't have all 3 of them. And it's a little disappointing because you'd love to have an algorithm that does. Because, I think we agree, all 3 of these properties makes sense. At least, certainly, independently they do. >> Right. >> But just how bad is it? I mean when, when you can't have all 3, can you come close to having all 3? Can you have like 2.9 of them? [LAUGH] >> Well, you can definitely have two of them, because that's what we did in the quiz. >> hm. >> Well, it kind of depends on what, you can reinterpret some of these properties and get, and nearly satisfy them. And I can point you to Kleinberg's paper to look into that. But I think I need to be done with this for now. I just wanted to give you a flavor of the idea that, one of the reasons that clustering is hard to pin down is because when you pin it down, it plays dead. >> Hm. >> It's like a, it's like an opossum. When you pin it down, it actually it turns out that it still doesn't really want to do what you want it to do. But in practice, what happens is people do clustering for lots of different reasons. And they're willing to change their clustering, like, look at what actually comes out of it, and change their notion of clustering so that it's more consistent with what they're trying to achieve. It's not great to use in this purely automated fashion, where you just hand it over to the computer and be done with it. But it's still a really powerful thing to do to get to know your data better. >> Okay. I accept that. I feel better now.



29. So that's it for clustering, that's what I wanted to tell you about. Did you want to help me remember what we learned? What, what did we talk about? >> So, we talked about clustering, as an idea, which is sort of a pretty powerful one. And I guess we, we connected that to the notion of compact description. And we talked about a bunch of algorithms, and in particular we talked about k-means and single-linkage clustering. >> That's right, and there was another one. >> EM. >> Yes. >> And the big difference between EM and the other two is that EM makes for soft clusters versus hard clusters. >> Yeah, that's one distinction. You could also say that the difference between single-link clustering (注意是 SLC, 不是 k-means) and the others is it actually terminates in polynomial time. >> Mm. Doesn't k-means terminate kind of fast? >> I, I don't think we know, exactly. I think it could run through a lot of different configurations before it stops. >> So we talked about some algorithms. And then we did something rather startling, and we showed that clustering is in some sense impossible. >> [LAUGH] Right, which is silly because we know that it's possible, we do it all the time. But we can't have those particular three properties all at once. >> Right. >> That axiomatization is mutually contradictory. >> Mm-hm >> Yeah, I think, I think that's about it. So I'm ready to move onto some other unsupervised topics. Huh. Cool. Me too. All right, well thanks, Michael. I learned a lot. >> Sure, yeah. I think next time you're going to tell me about something, features. >> I'm a tell you about something. Features. >> All right! Well, I'm looking forward to something, features. >> Okay! Bye, Michael. >> See you.