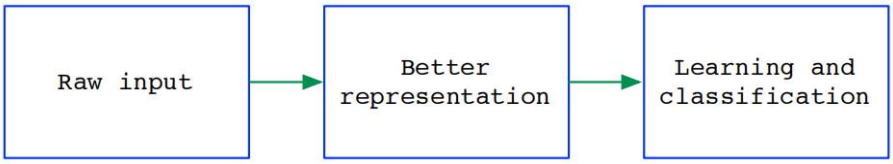
Module 8 - Representation Learning I

- 8.1 Representation Learning
- 8.2 Clustering with the k-Means Algorithm I
- 8.3 Clustering with the k-Means Algorithm II
- 8.4 Clustering with Mixtures of Gaussians
- 8.5 Hierarchical Clustering

总结:

8.1 Representation Learning

Representation learning



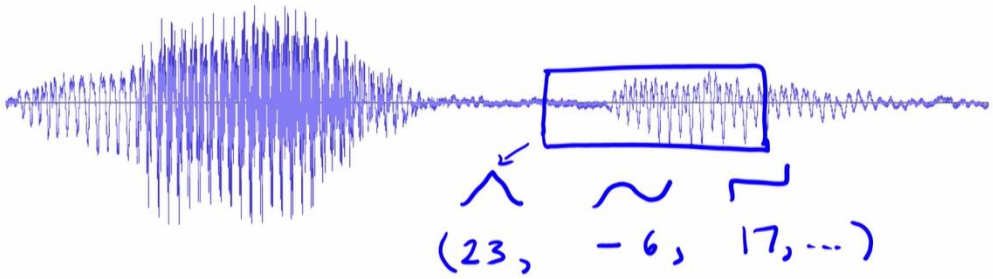
Good representations make learning easier.

- They bring out the true degrees of freedom in the data.
- They capture relevant structure at multiple scales.
- They screen out noisy or irrelevant structure.

We've spent a lot of time on how to build predictive models of data, classifiers, regressors, probability estimators. The whole time, we've been assuming that the representation of the data, the features, the distance function, the similarity function, that this representation is just given to us and is somehow fixed. But in general, this isn't the case. And in fact, the efficacy of these predictive methods, all of them, depends crucially on having a good representation of the data.

In a nutshell, good representations make learning easier. And this happens for a variety of reasons, in a variety of ways. A good representation brings out the true degrees of freedom in the data. It captures structure in the data at many different scales. It helps screen out noise and downweights irrelevant features. Now all of these things sound very good, but what do they mean exactly? What do I mean, for instance, when I talk about the degrees of freedom in data? Let's look at an example to make this a bit clearer.

Degrees of freedom



Usual representation of speech:

- Take overlapping windows of the speech signal
- Apply many filters within each window
- More filters ⇒ higher dimensional

Yet it comes from a physical system with a few degrees of freedom.

Here is a speech signal. It's a one-dimensional time series. When I speak, I agitate the molecules in the air in front of my mouth and this propagates a wave. When this wave reaches your ear, you have this very thin diaphragm in your ear that wiggles in response to this disturbance. And the signal over here, this one-dimensional signal, is literally just the displacement of that diaphragm, it's the way in which it's wiggling, okay? So this is a continuous one-dimensional time series. If we want to process it on a computer, for example, if we want to do speech recognition, we have to somehow digitize it and discretize it. How is that usually done? Well here's one common way. The first thing is to take the speech signal and to divide it into a bunch of overlapping windows, say windows of 25 milliseconds, okay? So you have a particular window, maybe something like this. And then, for that window, you apply a variety of filters. So maybe filter number one looks like this. And filter number two looks like this. And filter number three looks like that. Each filter yields a number, so maybe this yields 23, this yields -6, this yields 17, and so on. And these numbers get stacked together into a vector. In this way, that window of continuous sound gets converted to a point in some higher-dimensional Euclidean space. Now the more filters we use, the higher the dimension.
So we can actually get a representation that's super high-dimensional. But there's something a little bit deceptive about this. After all, even though it might seem very high-dimensional, the speech is produced by a physical apparatus, by my vocal track. And this

总结:

is a physical system that just has a few degrees of freedom.
So is there a way to take this very high-dimensional representation, the concatenation of all these filter values, and somehow recover the underlying degrees of freedom? Is there a way to start with that higher-dimensional vector and somehow simplify it or compress it into something that really brings out the true underlying structure of this data? Now, this really depends, being able to do this really depends on the relationship between these underlying degrees of freedom and the higher-dimensional vector representation. Is it a linear relationship? Is it a non-linear function? Is it some other type of encoding? We will be investigating a lot of these different possibilities.

Multiscale structure

Commonly-occurring structure at many levels.

Another important aspect of representation learning is that the structure that we need to discover is often at multiple different scales. So imagine what's needed, for example, in order to understand an image. There is some very low-level structure, for example, the individual edges(图中屋子轮廓的某条边), like this. Then there's slightly higher-dimensional structure, where, for example, a bunch of edges have been combined to yield a familiar object like a window. And this kind of composition, this combining of simple parts to yield a slightly more complicated object can be applied over and over again to yield progressively more complicated things, like the house, the cart, the horse, and so on. So we're interested in representations that have this multi-scale aspect to them.

Representation learning: goals

Learn underlying degrees of freedom and multiscale structure from the statistics of unlabeled data, e.g.:

- Clustering
- Linear projections
- Embedding and manifold learning
- Metric learning
- Autoencoders

Or learn a representation in tandem with the classifier: deep learning.

总结:

- So in representation learning, the goal, in a nutshell, is to uncover structure in the data. And we'll be focusing on certain key types of structure, especially cluster structure and linear projections.
- Later on, we'll unify a lot of this under the notion of an auto-encoder.
- Now, once a good representation is learned, then it can be used for subsequent predictive modeling, classification, regression, and so on. Or sometimes we don't have a specific predictive task in mind, but we're really interested in understanding the data better.
- At the end, what we'll do is to look at the ways in which we can combine representation learning with classifier learning. And this is what deep learning strives to do.

Okay, so this has been a bit of a road map for the upcoming two or three weeks. What we will do next time is to begin with our first kind of representation learning, clustering.

POLL

What is the goal of representation learning?

结果

总结:

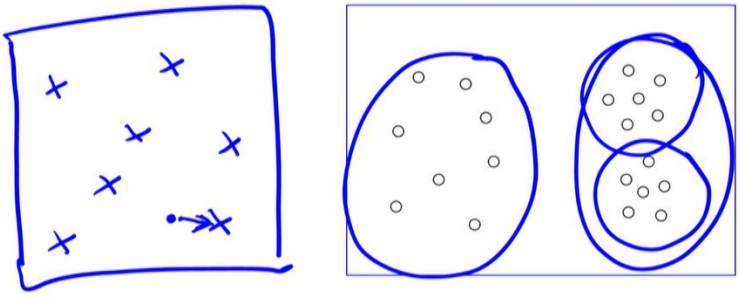
8.2 Clustering with the k-Means Algorithm I

Topics we'll cover

1. The clustering problem
2. Two uses of clustering
3. The k-means cost function and algorithm
4. Initializing Lloyd's algorithm

Our first big topic in representation learning is going to be clustering. So what we'll do today is to define what clustering is and then look at two rather distinct ways in which clustering gets used. Then we'll introduce the immensely popular K-Means cost function and the accompanying algorithm.

Clustering in $\mathbb{R}^d$



Two common uses of clustering:

- Vector quantization
  Find a finite set of representatives that provides good coverage of a complex, possibly infinite, high-dimensional space.
- Finding meaningful structure in data
  Finding salient grouping in data.

So, clustering. In clustering we are given a set of data points saying R D. And we're asked to cluster or group them. Okay, in this data set for example, how many clusters are there? Well, maybe there's one over here and one over here. Or perhaps there's actually two on the right hand side. Perhaps, there's this and this. Both options seem perfectly reasonable. Now there are two rather different ways in which clustering tends to get used.
The first is what might be called **vector quantization**. So, in this setting there is some very large, perhaps infinite or continuous space from which the data has been sampled. And the goal is to find a few representatives for that space. Okay, so there's some very large space. The data has been obtained from that space and the goal is to find a few representative points in that space. Subsequently, any point will simply be assigned to it's closest representative. For instance, this could be the space of all twenty millisecond speech recordings. Okay, so a continuous and very large space. What we want, then, is just a representative subset of these sound clips. And whenever a new sound clip arrives we'll simply replace it by it's closest representative. This is a compact way to encode a sound clip. It's a form of audio compression. Okay.
So, if this is the way in which we're using clustering, what is a good way to measure the quality of the clustering? Well it's quite naturally actually. We would measure it by the amount of distortion induced when you replace points by their closest representatives. So, that's the use of clustering for quantization. And we'll see an example of that later on.

The other big use of clustering is to **find meaningful groups in data**. To find natural clusters in a data set. And this is what would arrive, for example, in exploratory data analysis. So, for example, let's suppose that a supermarket is collecting data on it's customers. Who buys what. Is there a way to cluster the customers that really creates natural groups of people according to their purchase patterns. This could be very useful. For example, if the supermarket is gonna have a sale it could make sure that there are discounts that are appealing to each cluster. Or at least to each of the clusters that are large enough. So, two distinct uses of clustering. It's useful to keep these in mind as we evaluate different clustering algorithms.

总结:

Widely-used clustering methods

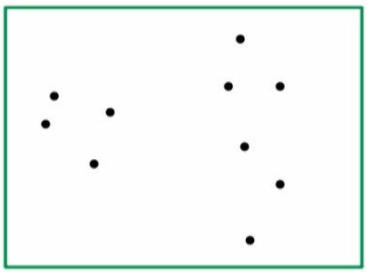
1. K-means and its many variants
2. EM for mixtures of Gaussians
3. Agglomerative hierarchical clustering

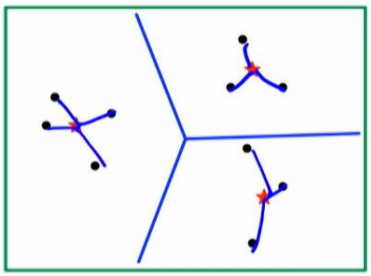
Clustering is an important problem. There are lots and lots of different clustering algorithms that people have come up with. It turns out, however, that there's just a small number of algorithms that have been hugely popular and that are widely used. The foremost amongst these is the K-means algorithm, which is what we'll be talking about today. And, later, we'll get on to some of the other more popular algorithms. Such as EM for mixtures of Gaussians and some of the agglomerative hierarchical clustering algorithms.

The k-means optimization problem

- Input: Points $x_1, \ldots, x_n \in \mathbb{R}^d$; integer k
- Output: "Centers", or representatives, $\mu_1, \ldots, \mu_k \in \mathbb{R}^d$
- Goal: Minimize average squared distance between points and their nearest representatives:

$$\text{cost}(\mu_1, \ldots, \mu_k) = \sum_{i=1}^{n} \min_j \|x_i - \mu_j\|^2$$





The centers partition $\mathbb{R}^d$ into k convex regions: μ_j's region consists of points for which it is the closest center.

Let's start with K-means. Do you remember we talked about two uses of clustering and one of them was vector quantization? So, K-means is something that really seems geared towards this objective. So, in this problem, in the formulation of the optimization problem, what you're given is a set of endpoints in d dimensional space. These are the points to be clustered. And you're also given an integer k. That's the number of clusters, the desired number of clusters. The output is a set of representatives, mu one through mu k, a set of cluster centers. These are just k points in d dimensional space. What we're going to be doing, essentially, is thinking about replacing each point by it's closest representative. So, the goal is to minimize the distortion that gets induced by doing so. So the way we formalize this precisely is that we want a set of centers such that the average distance from a point to it's closest center is small. So, let's look at the cost function. The cost associated with a specific set of centers, mu one through mu k, is a summation over all the data points.

- So, let's say we're looking at point x of i. When we're dealing with point x of i, we look at the closest center, the closest of the k centers. So, we take the min over j and the cost is the distance to that center squared. So the charge for point x of i is the distance to the closest center squared. And we wanna minimize the average charge or the total charge of this kind. So it's a formulation that's very much geared towards factor quantization.
- Here's an example data set with ten points. Let's say we want three clusters in it. Where might we place the cluster centers? So here's an example of where the cluster centers might go. And if we place the centers there what we're essentially saying is that each point in the plane is gonna get sent to it's closest center. So this is gonna divide up the plane in the following way. Any point in that region to the upper right will get sent to that center, and so on and so forth.So what is the k-means cost in this case? The cost is the distance of this point to it's center squared plus this distance squared plus this squared, plus this squared, plus this squared and so on and so forth. It's the sum of these ten distances squared.

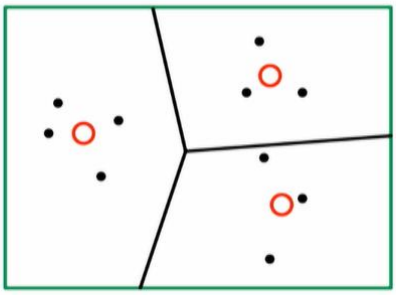
Now, more generally, we will be in R^d and there will be k centers. We assign each point to it's closest center and so this ends up carving d dimensional space into k regions. **It turns out that each of these regions will be a convex region and is often called a Voronoi Cell**. So, this is the k-means optimization problem. This is just stating what we want from the centers. But now we need an algorithm. How do we actually find the centers that minimize this cost function? How do we find the solution to this problem?

总结：

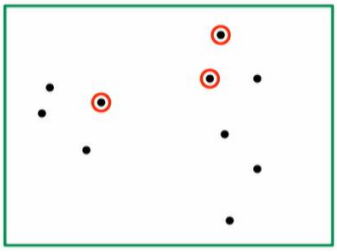
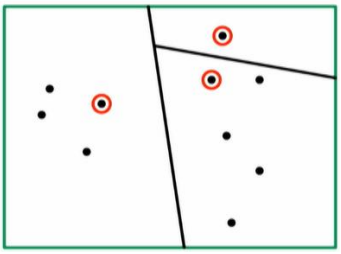
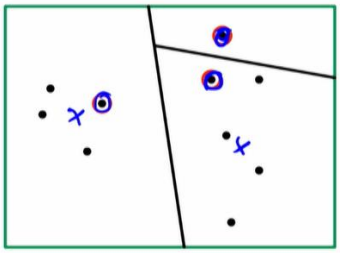
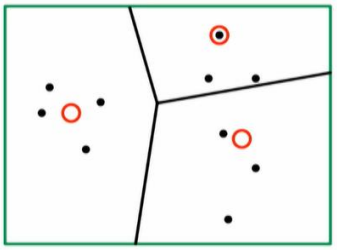
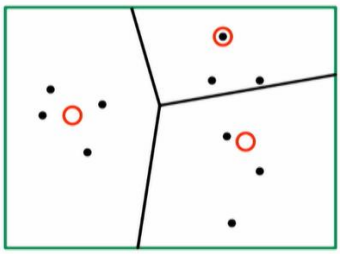
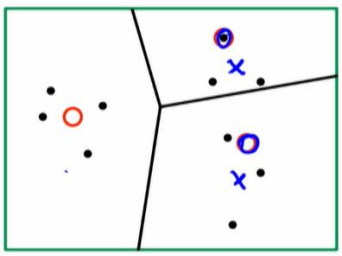
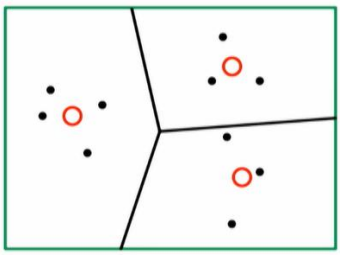
Lloyd's k-means algorithm

The k-means problem is NP-hard. Most popular heuristic: "k-means algorithm".

- Initialize centers $\mu_1, \ldots, \mu_k$ in some manner.
- Repeat until convergence:
  - Assign each point to its closest center.
  - Update each μ_j to the mean of the points assigned to it.



Each iteration reduces the cost $\Rightarrow$ convergence to a local optimum.

So the bad news is that this problem is **NP-hard**. Now what that means, in a nutshell, is that there's no efficient algorithm that's guaranteed to always yield the optimal solution. So we have to look at **heuristics**. And the most popular heuristic and also the most popular clustering algorithm of all time is **Lloyd's algorithm**. Which is often, or typically, just called the **k-means algorithm**. This is a very simple algorithm. It's given in it's entirety over here. And it is a local search algorithm. Now the goal here is to find k centers. And what this algorithm does is to just initialize the k center somehow, so it has k points. And then it just tweaks them a little bit to lower the cost. And then it tweaks them a little more to lower the cost. And it keeps going in this way until it converges. So, it's a local search or a hill climbing algorithm.

Let's look at the specifics. So, we start by initializing the centers in some way. The usual way to do this is to simply pick k of the data points at random and say, "Those are gonna be initial centers." Then, we begin the main loop. And this is the loop of iterative improvements. So, on each iteration we assign each point to it's closest center and then we move that center to the mean of all the points that have been assigned to it. Let's see that in action.

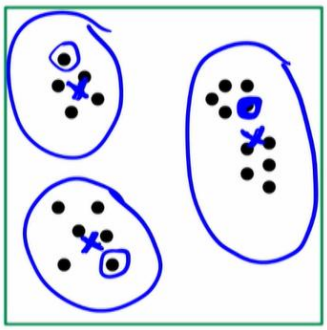
- We have this data set over here with ten points in it. We start by initializing the center somehow. Let's say we just pick three of the points at random. Let's say we have something like this. Those are initial centers. Mu one, mu two, and mu three. Now, we begin the main loop. And on the first iteration we assign every point to it's closest center. So, that means is the space gets carved up somewhat like this. Let's look at the cluster on the left over here. There are four points over here and what we're gonna do now is to move this center, this one over here, to the mean of those four points. We're gonna move it over here. That's definitely an improvement right? Now, let's look at this center up here. It has just one point assigned to it. It's not gonna move. Now, let's look at this one over here. We're going to move it to the mean of the points that have been assigned to it. In this case five points have been assigned to it. And their mean is somewhere over here. That's where that center is gonna move. So, those are the new centers.

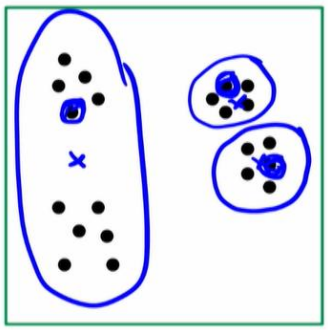
总结:

- Those are our new centers and now we begin another round of k-means. We begin the second iteration. So, once again we assign each point to it's closest center. So we end up with this. This is the way in which the space gets carved up into three regions.And once again is what we're gonna do is we're gonna replace each center by the mean of all the points that have been assigned to it. So let's look at the cluster over here on the left. Four points have been assigned to it and it's already the mean of those four points. So, it's not going to change. Let's look at this center over here. Three points have been assigned to it. And so it'll move to the mean of those three points. Somewhere there. Let's look at this center. Three points have been assigned to it and we'll move to the mean which is somewhere here. So, now, the new centers look like that.
- So, once again we assign each point to it's closest center. So this is what we get. And, once again, we replace each center by the mean of the points assigned to it. And now what happens is that nothing moves. Everything is already the mean of the points that were assigned to it. And so the algorithm converges.

It's a very simple algorithm. What can we say about it? Does it always find the optimal solution to the k-means cost function? No it does not. We wouldn't expect it to given that this is an NP-hard problem. But we can say some things about it. So, in particular, with each successive iteration the cost is guaranteed to go down. So we keep reducing the cost from iteration to iteration. And, eventually, it stops in some sort of **local optimum**. Is it the global optimum? In general, no. So, it's a very simple algorithm. We've specified it fully.

Initialization matters





The one thing that we have left unspecified is the method of initialization. And so I wanna talk about this a little bit more. Because it turns out that it's important to do this right.
And let's see why that's the case. So, let's look at little examples. Here's a data set with, I dunno, roughly, sixteen points. And what we're gonna do is to initialize it two different ways. And we'll look at the resulting k-means clustering.

So let's start on the left. Let's say we want three clusters and that we're going to initialize just by picking three points at random. So maybe the points we pick are, this point over here, and this point over here, and this point over here. So those are our three initial centers. So, now we assign each point to it's closest center. And we end up getting clusters, roughly, of this kind. And now we replace each center by the mean of the points that got assigned to it. So, in this case, it'll move over here somewhere. And, in this case, it'll move over here somewhere. And, in this case, it'll move probably over here somewhere. So the crosses are the new centers. And so now we'll begin a new iteration. What are the new clusters? Well at this point the clusters don't change. Everything remains in the same cluster as it was before. And so those are the final centers as well. So the algorithm ends up with this clustering and this set of three centers.

So, now, let's pick another random initialization. So, again, we pick points at random. Let's say we end up picking this as one of the initial centers. And this as one of the initial centers. And this as one of the initial centers. So those are our three initial centers. Let's see what happens now. So, which points get assigned to this center over here? These points over here. So, that's one of the clusters. Which points get assigned to this center? These points over here. That's another one of the initial clusters. And which points get assigned to that center? These points over here. So these are the three initial clusters. And during the first round of k-means, the means will move. This one over here will get sent to the mean of all these points. Which is somewhere here. This one will get sent to this point here. And this one will get sent very close to where it currently is. So, the x's mark the new means. And now, once again, we see that the clusters will not change on the next iteration. And so the algorithm converges.

So, if you look at these pictures, we had two different initializations and we got two very different answers at the end. Which do you think is the better answer? Well, either way it's a clear sign that it's important to initialize the algorithm well.

总结:

Initializing the k-means algorithm

Typical practice: choose k data points at random as the initial centers.

Another common trick: start with extra centers, then prune later.

A particularly good initializer: **k-means++**

- Pick a data point x at random as the first center
- Let $C = \{x\}$ (centers chosen so far)
- Repeat until desired number of centers is attained:
  - Pick a data point x at random from the following distribution:

$$\Pr(x) \propto \text{dist}(x, C)^2,$$

    where $\text{dist}(x, C) = \min_{z \in C} \|x - z\|$
  - Add x to C

Let's look at some ways in which that's typically done.

- So as I said the most common practice, by far, I believe, is to simply pick k of the data points at random and make those the initial centers.
- Another common trick is to start with extra centers. So if you want ten clusters, let's say. You start with twenty centers. You have extra centers. And now you run k-means for a while. And then you kill ten of the centers. Now your left with just ten centers and you then you'll run k-means until it converges. So, which centers do you kill? The ones you get rid of are the ones that had very few points assigned to them. Or the ones that are very close to other centers. So, that's another common trick for running k-means.
- Now, one very good method of initialization isto use what's called the **k-means plus plus algorithm**. Let me tell you what this is. What happens over is that the initial centers, mu one through mu k, are picked one at a time. So first, we pick mu one by just picking a random data point. We have n data points, we pick one at random, that's mu one. That's our first center. Now we have to pick the second center. And again we pick it at random but this time we favor points that are far away from mu one. The way we do that is that we say that the probability of picking a point x is proportional to the distance from x to mu one squared. If something is very far from mu one there's a higher probability of picking it. And if x is equal to mu one, the probability is zero. It certainly not pick my one again. So in this way, the second, this is the way in which the second center, mu two, is picked. How do we pick the third center? Well again we pick at random from the data set. But now we favor points that are far away from both mu one and mu two. And we keep going in this way until we have all k centers. So a little bit more formally. During this process, at any given time we've picked a bunch of centers c. How do we pick the next center? Well, we pick at random from the data set. From our n points that we've been given. And the probability that we pick a particular point x, is proportional to the distance from x to the centers we have so far squared. It's a very simple initialization procedure. And it turns out to be quite effective.

Well, that concludes our initial treatment of the k-means algorithm. Next time, we'll go into a little bit more depth. We'll see a couple of uses of k-means. And we'll also look some variants that are geared towards massive data sets.

POLL

What is the best description of the cost of a k-means solution?

结果

The average distance between each point and its mean	5%
The sum of the probabilities that each point belongs to the assigned clusters	2%
The sum of the distances between every data point and its mean	7%
✓ The sum of the squared distances between every data point and its assigned cluster center	86%

总结：

8.3 Clustering with the k-Means Algorithm II

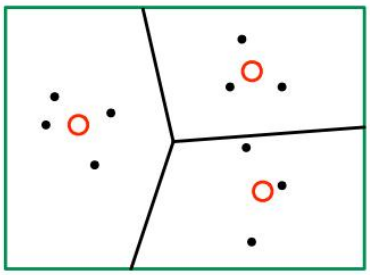
Topics we'll cover

1. Two uses of k-means clustering
2. Clustering in a streaming or online setting
3. The good and bad of k-means

We will now continue with the k-means clustering algorithm. So, to start with, we'll look at two rather distinct applications of k-means clustering, we'll then look at a variant of the algorithm that's motivated by the realities of massive datasets, and finally, we'll end with a little summary of the pros and cons of k-means.

Lloyd's k-means algorithm

- Initialize centers $\mu_1, \ldots, \mu_k$ in some manner.
- Repeat until convergence:
  - Assign each point to its closest center.
  - Update each μ_j to the mean of the points assigned to it.



Each iteration reduces the cost $\Rightarrow$ convergence to a local optimum.

So, if you recall, Lloyd's k-means algorithm is a local search procedure. It initializes the k-centers in some way and then enters this loop where it iteratively keeps improving them until it converges.

Two common uses of clustering

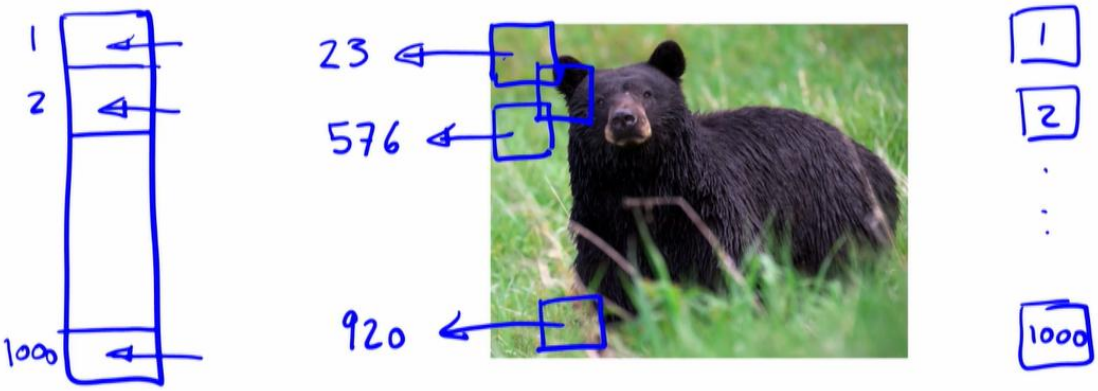
- Vector quantization
  Find a finite set of representatives that provides good coverage of a complex, possibly infinite, high-dimensional space.
- Finding meaningful structure in data
  Finding salient grouping in data.

So, what are some ways in which these clusterings might be used? Well, last time, we spelled out two rather distinct uses of clustering. The first is vector quantization where the goal is to take a large, continuous, infinite space and to find a small set of representatives for this space so as to minimize the distortion induced by this kind of discretization. The second general way in which clustering is used is to find meaningful groups or natural clusters in data. So, let's look at k-means applications of each of these two varieties, starting with the first one.

总结:

Representing images using k-means codewords

How to represent a collection of images as fixed-length vectors?



- Take all $\ell \times \ell$ patches in all images. Extract features for each.
- Run k-means on this entire collection to get k centers.
- Now associate any image patch with its nearest center.
- Represent an image by a histogram over $\{1, 2, \ldots, k\}$.

So, here's a question. How can we represent images as vectors of fixed-length? After all, images come in all shapes and sizes. How do we make them all the same length? How do we put them into vector form? Well, this is a problem for which the computer vision community has developed many different solutions. At a very high level, however, many of these solutions have a somewhat similar flavor and what I'll do is to describe one of them that's based on k-means clustering. So, how does it go? We'll take a picture, for example, this picture over here of a black bear, and let's pull out a small patch from this picture. So, let's fix a certain size, maybe 10 by 10. Let's look at 10 by 10 patches, and here's an example of such a patch. There's a 10 by 10 patch, and there's another 10 by 10 patch, and another such patch, and another patch, and so on. So, let's go ahead and pull out all such patches from this picture. Okay, so we're gonna get a lot of them. And then, let's do this for every single picture we have. So, now we have a vast collection of these 10 by 10 patches. Now, each patch is of a fixed size and so we can represent it by a fixed-length vector. So, we have an enormous collection of these fixed-length patches and we're gonna apply k-means clustering to them. Let's say we use k equals 1,000, so what that gives us, what k-means gives us is that it takes this collection of patches, there might be hundreds of millions of them, and it finds 1,000 representative patches. Now, how are we gonna use these to encode images? How are we gonna use these to come up with a good representation for images? Well, let's go back to this bear picture over here. We have this way of pulling patches out of it, so let's look at this patch over here and let's look at its closest representative. Now, we used k-means to come up with 1,000 patches, one to 1,000. Let's number them. Let's look at this patch on the upper left over here and say, which is the closest representative?
Oh, it's number 23, so we're gonna represent this entire patch by just the number 23. It's the way we compress that patch. Now we look at this other patch and we say, which is the closet representative? Oh, it's number 576, so we represent by simply the number 576. Now we look at this other patch and we say, which is the closet among the thousand representatives that k-means gave us? Oh, it's number 920, so let's represent it by the number 920 and we do this for every patch in the image.

There's lots of these patches, and now, in order to get a vector, what we do is we simply write down a 1,000 numbers. We get a 1,000-dimensional representation and these are frequencies. For the first entry, we say, how many times, what fraction of patches were assigned to representative number one? What fraction of patches were assigned to representative number two? What fraction of patches got assigned to representative number 1,000? So, the entire image, this image of a bear, gets summarized by this 1,000-dimensional vector of probabilities. These fractions add up to one. And representations of these kind have been found to be quite useful in many different vision applications.

总结:

Looking for natural groups in data

"Animals with attributes" data set

- 50 animals: antelope, grizzly bear, beaver, dalmatian, tiger, ...
- 85 attributes: longneck, tail, walks, swims, nocturnal, forager, desert, bush, plains, ...
- Each animal gets a score $(0 - 100)$ along each attribute
- 50 data points in $\mathbb{R}^{85}$

Apply k-means with $k = 10$ and look at grouping obtained.

1. zebra
2. spider monkey, gorilla, chimpanzee
3. tiger, leopard, wolf, bobcat, lion
4. hippopotamus, elephant, rhinoceros
5. killer whale, blue whale, humpback whale, seal, walrus, dolphin
6. giant panda
7. skunk, mole, hamster, squirrel, rabbit, bat, rat, weasel, mouse, raccoon
8. antelope, horse, moose, ox, sheep, giraffe, buffalo, deer, pig, cow
9. beaver, otter
10. grizzly bear, dalmatian, persian cat, german shepherd, siamese cat, fox, chihuahua, polar bear, collie

1. zebra
2. spider monkey, gorilla, chimpanzee
3. tiger, leopard, fox, wolf, bobcat, lion
4. hippopotamus, elephant, rhinoceros, buffalo, pig
5. killer whale, blue whale, humpback whale, seal, otter, walrus, dolphin
6. dalmatian, persian cat, german shepherd, siamese cat, chihuahua, giant panda, collie
7. beaver, skunk, mole, squirrel, bat, rat, weasel, mouse, raccoon
8. antelope, horse, moose, ox, sheep, giraffe, deer, cow
9. hamster, rabbit
10. grizzly bear, polar bear

Now let's look at another kind of use of clustering, defined natural groups in data.
There's a dataset here which you will get to play with as well called Animals with attributes, and this is information about 50 animals, antelopes, grizzly bears, dalmatians, and so on. For each animal, it has 85 features and these features capture things like where does the animal live? In the desert or the rainforest or the ocean? What kind of food does the animal eat, is it a herbivore or carnivore? What is its physical appearance, does it have a long neck? And so on. So, there are 85 features of this kind and so, each animal is essentially represented as an 85-dimensional vector, and we have 50 animals so the dataset consists of 50 points in 85-dimensional space and we can go ahead and apply k-means to it. So, what I did is to do this with k equals 10, so I asked for 10 clusters. And as we discussed last time, each time you run k-means with a different initialization, you potentially get a different answer. So, what I did is to do this twice and you can see the results over here.

- Let's look at the first run of k-means on the left over here. What are the clusters we get? We get cluster number two over here, spider monkey, gorilla, and chimpanzee. That looks pretty good. That makes sense. Cluster number five over here, killer whale, blue whale, humpback whale, that looks pretty decent. Are things that live in the water. What else do we have? Let's look at number three. Tiger, leopard, wolf, bobcat, lion. Seems reasonable to me. Let's look at cluster number 10. It looks a little weird. It's got grizzly bear and polar bear. I can see putting them together, but it also has Siamese cat and Persian cat in it and that doesn't seem right. That cluster looks a little bit dicey. So, some of the clusters are good and they really seem to represent meaningful groups in the data, and some of them are a little bit dicey, and to be honest, this is really the most common situation in clustering. You get quite a bit of valuable information, but it usually isn't exactly aligned with what you might want.
- Now let's look at the second run of k-means. A lot of the clusters are actually the same. So, this cluster, cluster number two is the same one we got last time. Cluster number three has changed a little bit, fox is in there as well. Cluster number five is the same as last time. Oh, cluster number 10 is better, now it's just grizzly bear and polar bear so that's good. So, what happened to Siamese cat and Persian cat? Here they are and uh-oh, they got put in with giant panda, so it's a similar story. Some of these clusters are very good, some of them are a little bit dicey. It's a mixed bag and it's actually hard to tell in this case which of these two clusterings is better.

总结:

Streaming and online computation

Streaming computation: for data too large to fit in memory.

- Make one pass (or maybe a few passes) through the data.
- On each pass:
  - See data points one at a time, in order.
  - Update models/parameters along the way.
- Only enough space to store a tiny fraction of data, or perhaps a short summary.

Online computation: even more lightweight, for data continuously being collected.

- Initialize a model.
- Repeat forever:
  - See a new data point.
  - Update model if need be.

So, very often we have to apply clustering to extremely large datasets. If you think back to our image example, for instance, from each picture we were extracting all 10 by 10 patches and then we were doing this for all the pictures in a collection, so this could easily be a dataset of hundreds of millions of points. In situations like this, when there is so much data that one can't even keep it in memory, what does one do? And people have developed some alternative models of computation that are intended to address these sort of scenarios specifically.

- So, one that's been a very successful model is what's called the **streaming model of computation**. So here, there's a fixed dataset, but it's very, very large. Way too large to fit into memory, so it's sitting on some disk somewhere and all the computer is allowed to do, all you're allowed to do is to do one pass through that data, or maybe two passes or three passes or maybe 10 passes, just a few passes through that data. So, you keep seeing these points in order, the points come by in order, and you can't store all of them. You might be able to store a tiny fraction of them, so if they're end points, maybe you can store a square root end of them or something like that. And as you see each point, you might update the parameters of your model. So the question is, can we do clustering in a setting like this?
- An even more extreme model is the **online model of computation**. So, in this case, there isn't even a fixed dataset. What happens is that data is being collected continuously forever. At any given point in time if a new data point arises, you aren't able to store it, you only have a constant amount of memory, so you see the point. Maybe update your model based on the information in that point and then the point goes away and you never, ever get to see it again, so the data literally just flies by once. That's the online model of computation. So, can we do k-means in these sort of models? 'Cause that would yield a version of the algorithm that's more suitable for very, very large datasets, and actually, there have been several versions of k-means that are of this kind and I'll give you an example.

Example: sequential k-means

1. Set the centers $\mu_1, \ldots, \mu_k$ to the first k data points
2. Set their counts to $n_1 = n_2 = \cdots = n_k = 1$
3. Repeat, possibly forever:
   - Get next data point x
   - Let μ_j be the center closest to x
   - Update μ_j and n_j:

$$\mu_j = \frac{n_j \mu_j + x}{n_j + 1} \quad \text{and} \quad n_j = n_j + 1$$

总结:

This is what is often called **sequential k-means** and it fits into either the streaming or the online model. So you're seeing data points one at a time, you want k-centers, what you do is, the way you initialize it, is you initialize your k-centers to just the first k points that you see and once you get those k points, you also initialize the counts associated with those centers to one. Okay, so these are the counts associated with each of the centers. Now, new points keep arriving, possibly forever, and each time you get a new point x, you take this point and you say, well, which of my centers is it closest to? Let's say it's closest to center number mu j. Let's say it's closest to center mu j, then we're gonna update the count from mu j, so we'll increment n sub j and we will also tweak that center slightly. We will tweak it so as to average in this point x. So, this is a very simple and lightweight version of k-means, and there are, in fact, several other variants of k-means that can also deal with massive datasets quite easily.

K-means: the good and the bad

The good:

- Fast and easy.
- Effective in quantization.

The bad:

- Geared towards spherical clusters of roughly the same radius.

How to accommodate clusters of more general shape?

We've talked a lot about k-means, in large part because it is so popular and so widely-used. What are the pros and cons of this algorithm?

- So, one big pro of the algorithm, one very good thing about it is that it's an extremely simple algorithm and it runs fast. If our goal is to do vector quantization, then the cost function for k-means is really very sensible. It seems geared towards applications of that kind.
- What's bad about k-means? Well, if we think about the other kind of application where what we want is to discover natural clusters in data, it turns out that k-means is really geared towards a very specific type of cluster. It's geared towards finding clusters that are spherical and of roughly the same size. It would be very nice if we could find other algorithms that are similarly simple, but allow more general cluster types and that's something we'll be seeing fairly soon.

That concludes our treatment of k-means. We've spent a lot of time on it because it's a very important algorithm, and what we'll do next is to move on to some of the other key clustering schemes.

POLL
Which is not a benefit of k-means clustering?

结果

It's fast and easy	9%
It's effective at quantization	3%
It's effective with clusters of many different shapes	68%
These are all benefits of k-means clustering	20%

总结:

8.4 Clustering with Mixtures of Gaussians

Topics we'll cover

1. Gaussian mixture models
2. The optimization problem
3. The EM algorithm
4. Examples

We've been talking about the K-means clustering algorithm. Today we'll look at a different algorithm that's also very popular called the EM algorithm. It is based on Gaussian distributions. So we'll begin by introducing Gaussian mixture models which are generative models for data with clusters in it. This will allow us to formalize an optimization problem for clustering and the usual way we solve this problem is via a nice, simple local search algorithm called EM and we'll end with a few examples.

K-means: the good and the bad

The good:

- Fast and easy.
- Effective in quantization.

The bad:

- Geared towards data in which the clusters are spherical, and of roughly the same radius.

Is there is a similarly-simple algorithm in which clusters of more general shape are accommodated?

So we've been talking a lot about K-means and we kept emphasizing its many advantages. Above all else, it's a wonderfully simple algorithm and thus, very attractive to use but it also has its flaws. In particular, it works best when the clusters are roughly the same size and also have spherical shapes. What we'll see today is an algorithm that is also very simple but can accommodate clusters of more general shapes and sizes.

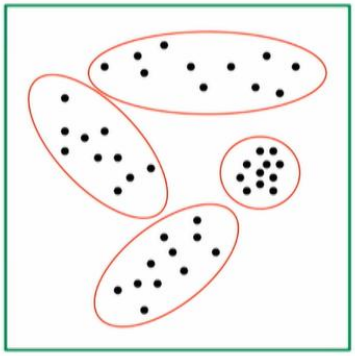
Mixtures of Gaussians

总结：

Mixtures of Gaussians

$\mu_j \in \mathbb{R}^d$

Σ_j $d \times d$ matrix



Each of the k clusters is specified by:

- a Gaussian distribution $P_j = N(\mu_j, \Sigma_j)$
- a mixing weight π_j

$$Pr(x) = \sum_j \underbrace{Pr(\text{cluster } j)}_{\pi_j} \underbrace{Pr(x \mid \text{cluster } j)}_{P_j(x)}$$

Overall distribution over $\mathbb{R}^d$: a **mixture of Gaussians**

$$\Pr(x) = \pi_1 P_1(x) + \cdots + \pi_k P_k(x)$$

So here's the general idea, we have a set of data points that we want to cluster like this for example and we cluster that and then we fit a Gaussian to each cluster. So we might get something like this for example with four clusters. Now as we know Gaussians can have arbitrary ellipsoidal shapes and so we're also going to be able to accommodate clusters of this type.

- So at the end of the process, we'll have these **K clusters and each cluster will be specified by a Gaussian distribution**, P1 through PK. Now in d dimensional space, the Gaussian distribution for the Jth cluster for example will be specified by its mean, miu sub J which is going to be a vector in d dimensional space so miu sub j is a d dimensional vector and it will also be specified by its covariance matrix, sigma sub j which will be a d by d matrix. So these will describe the jth cluster, the distribution of the jth cluster. They'll specify a density for the jth cluster.
- We will also keep track of some **mixing weights** for each of the clusters. So these are the pi sub j's and pi sub j is simply the fraction of the data that comes from that cluster. In this example for instance, all the clusters seem to have roughly the same weight so pi one, pi two, pi three, pi four would all be roughly .25.

So we have a density for each clusterand by combining these densities, we get an overall mixture of Gaussians and this is a distribution over the entire space, over all of our data. The probability the density that it assigns to a specific point x is just pi one times the probability at x, the density at x under distribution p one plus pi two times the density under p two plus pi three times the density p three and so on and so forth. Okay so why is this?

Well the probability of any given point x depends on which cluster it's from. So let's sum over all possible clusters and we'll say what is the probability of being in that cluster and then what is the probability of x given that you are in that cluster? And these are exactly the terms we see below. The probability of cluster j is pi sub j and the probability of x under that cluster is p sub j of x. So each cluster has an associated Gaussian distribution and we combine these k components into an overall mixture of Gaussians. So we take our data and we fit a mixture of Gaussians to it.

总结:

The clustering task

We are given data $x_1, \ldots, x_n \in \mathbb{R}^d$.

For any mixture model $\pi_1, \ldots, \pi_k$, $P_1 = N(\mu_1, \Sigma_1), \ldots, P_k = N(\mu_k, \Sigma_k)$,

$$\begin{aligned}
&\Pr(\text{data} \mid \pi_1 P_1 + \cdots + \pi_k P_k) \\
&= \prod_{i=1}^{n} (\pi_1 P_1(x_i) + \cdots + \pi_k P_k(x_i)) \\
&= \prod_{i=1}^{n} \left(\sum_{j=1}^{k} \frac{\pi_j}{(2\pi)^{d/2} |\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2} (x_i - \mu_j)^T \Sigma_j^{-1} (x_i - \mu_j) \right) \right)
\end{aligned}$$

Find the **maximum-likelihood mixture of Gaussians**:
the parameters $\{\pi_j, \mu_j, \Sigma_j : j = 1 \ldots k\}$ that maximize this function.

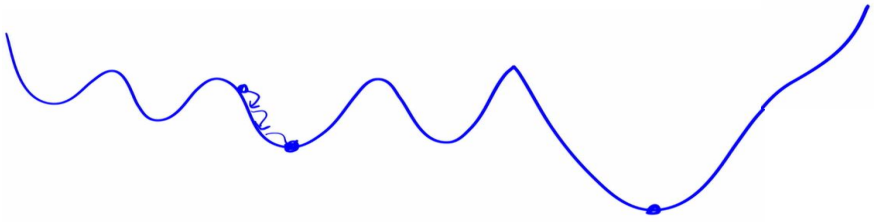
How exactly do we do this? Well let's say we have n data points, let's call it x one through x n. These are points in R d. Okay and at the end of the clustering process, we're going to have a mixture model. We're going to have a bunch of mixing weights, these proportions, pi one through pi k and then we are going to have k Gaussians, one Gaussian describing each cluster. Okay and for each Gaussian we're going to have its mean and its covariance matrix. So this gives us a distribution over R d.

What is the probability of our entire data set under this distribution? Okay so what is the probability of our data of our N data points under this mixture distribution? Well it's just the probability of point number one, x one times the probability of point number two times the probability of point number three and so on. So we have this giant product, this pi sign means product. So we're taking the product from I equals one to n off the probability of the ith data point of x sub I. Okay so we multiply these probabilities together and we get the probability of the overall data set. Now each of these distributions p one, p two and so on is a Gaussian distribution and we've seen the formula for the Gaussian density. If we just plug that formula in, we get this expression below. This is the likelihood of the data under the model pi one through pi k, p one through p k.
And what we want to do is to find the distribution, we want to find the Gaussian mixture model that makes the data maximally likely. We want to find the maximum likelihood mixture of Gaussians. We want to find the parameters pi j, miu j, sigma j where j goes from one to k that maximizes this expression over here.
Now maximizing this is the same as maximizing the log of this thing. And it's kind of helpful to do that because if we **take the log of this**, then this initial product will get converted into a **summation**. That's going to be helpful. So maximizing this thing is the same as maximizing its log and it's the same as minimizing the negative of the log. Maximizing something is the same as minimizing its negative.

Optimization surface

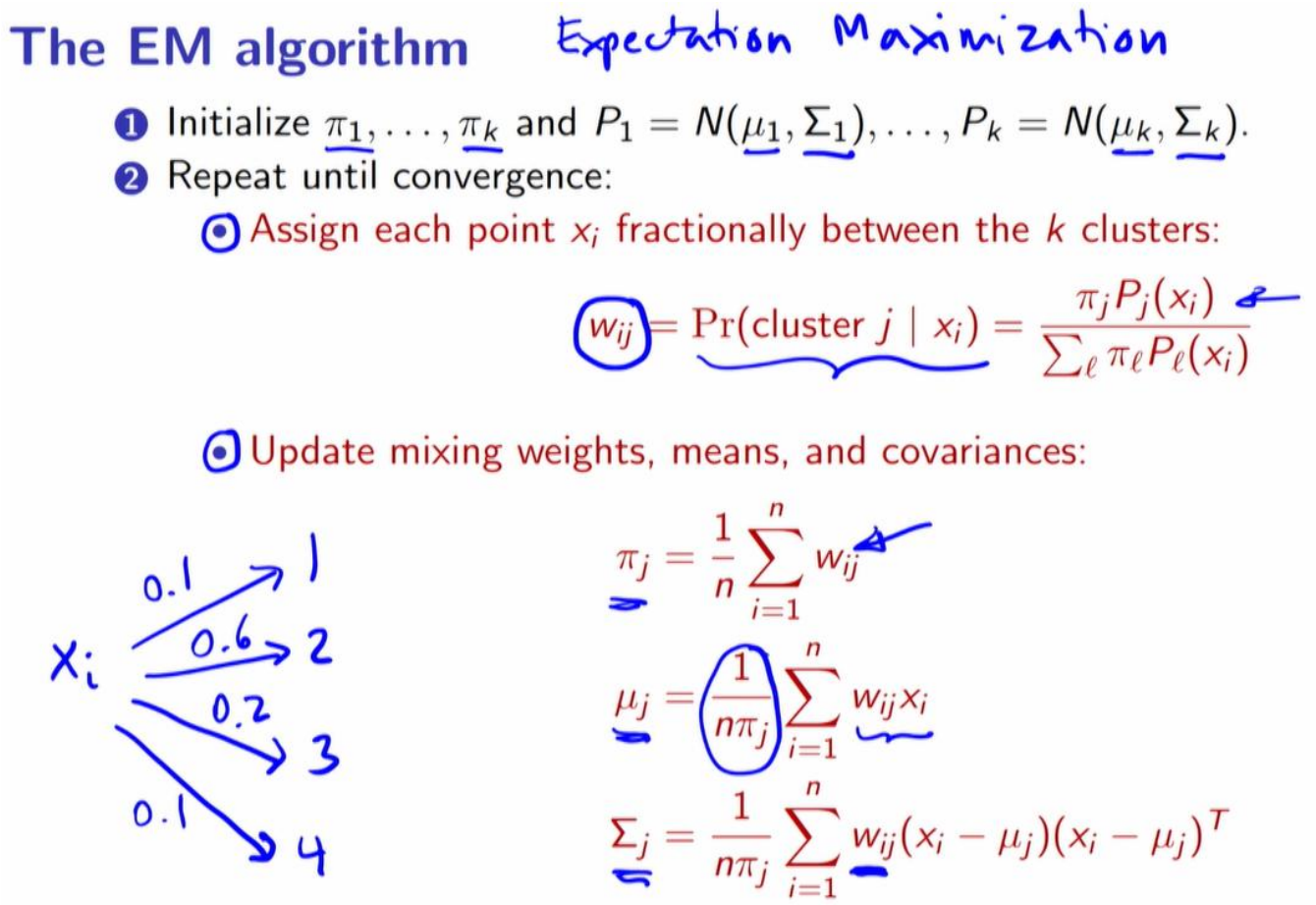
Minimize the negative log-likelihood, Not convex.

$$L(\{\pi_j, \mu_j, \Sigma_j\}) = \sum_{i=1}^{n} \ln \left(\sum_{j=1}^{k} \frac{\pi_j}{(2\pi)^{d/2} |\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2} (x_i - \mu_j)^T \Sigma_j^{-1} (x_i - \mu_j) \right) \right)$$



总结:

So when we do all that, the loss function we end up with is called the negative log likelihood. And this is the loss function. So we want to find a set of parameters. We want to find pis, mus and matrices sigma for which this expression over here gets minimized. Okay so is this an easy thing to minimize? Well it certainly looks like a bit of a mess but maybe it's convex, who knows? Actually it's not. It's not convex at all. Let's write that down. It's emphatically not convex and in fact, if you look at the surface of this loss function, it has lots and lots of local optima. Okay so it's something that looks like this and finding the global optimum, finding that minimum point over there is in p hard. So we're not going to be able to do that in every situation. Instead, the solution strategy that we're going to use is to use a local search procedure. So this is a procedure that starts with some solution that is to say some set of parameters, pi, miu and sigma. Maybe over here and then incrementally improves them. Makes it a little better and then makes it a little better and then ultimately ends up in a local optimum.



Let's see the algorithm that does this. This is called the EM algorithm where EM stands for expectation maximization. Actually EM is not just a single algorithm. It's an entire framework for making algorithms and so there are EM algorithms for a variety of other problems in machine learning and statistics. This however is probably the EM algorithm that's most well-known.

So let's see how it works. **It's a local search procedure**. You start by initializing the parameters in some way. You choose some initial set of weights, let's say you just make them all one over k okay and then you choose some initial Gaussian means. Let's say you just pick k of the data points at random and you choose some initial covariance matrices. Let's say you just take the covariance of the entire data set and set the initial covariance matrices to those. So you start off with some initial parameters and then you enter the main loop, the loop of iterative improvements. At each given time, you have your common set of parameters which is your, just a guess, it's your current guess and now you want to improve them a little bit. This takes place in the form of two steps. *The first step is to cluster the data based on the current set of Gaussians*. Let's see, we have a particular point x sub I and we have to decide which cluster it's going to go to. Is it going to go to cluster one, it it going to go to cluster two, is it going to go to cluster three, is t going to go to cluster four, how do we decide? Well each cluster has an associated Gaussian. So we can talk about the probability that that point is from cluster j. And it's simply this expression over here. It's pi sub j which is the prior probability of that cluster times the density of xi under that particular Gaussian. So roughly this corresponds to how close x sub I is, how likely x sub I is under each individual Gaussian. Okay let's say that the probability that x sub I from cluster one is .1, the probability from cluster two is .6, the probability is from cluster three is .2, and the probability from cluster four is whatever is left which is .1. So which cluster should we assign it to? Cluster two.

What we're going to do is something a little more interesting. We are not going to definitively assign x to any one cluster but rather we'll assign it to all the clusters but with different weights. So we'll say x sub I is assigned to cluster onewith a weight of .1 and it's assigned to cluster two with a weight of .6 and it's assigned to cluster three with a weight of .2 and to cluster four with a weight of .1. These are the weights w I j. It's the weight with which x sub I is assigned to cluster j. Now we have a clustering of the data, or more correctly some sort of fractional clustering. This is often called a **soft clustering**. So we have the soft clustering of the data and on the basis of this, we want to update all the parameters.

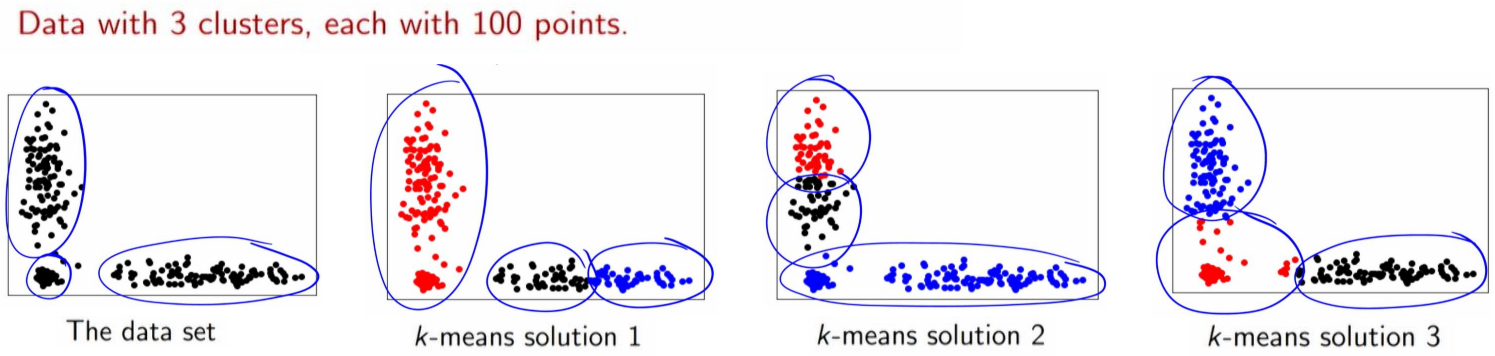
总结:

So we want to say okay this is what the clusters look like now, *now what is a good estimate of the mean of this cluster and the covariance and so on?*

- So let's start with the mixing weights. What proportion of the data comes from cluster j? Well let's look at the proportion of .1 from cluster j plus the proportion of .2 from cluster j plus the proportion of .3 from cluster j and just add these up and divide by n. Okay so that's our updated estimate for pi sub j.
- What should be set to be the mean for cluster j? So the mean is supposed to be the average of all the points in cluster j but if you remember we have assigned every point to cluster j but just with different weights so we need to take a weighted average. Okay and that's what we have over here, a weighted average. The stuff in the front here is just a normalizing factor.
- And it's similar with the covariance matrix. We just take a weighted average.

So that's one round of EM and then we keep repeating this until convergence. That's it. A very simple algorithm.

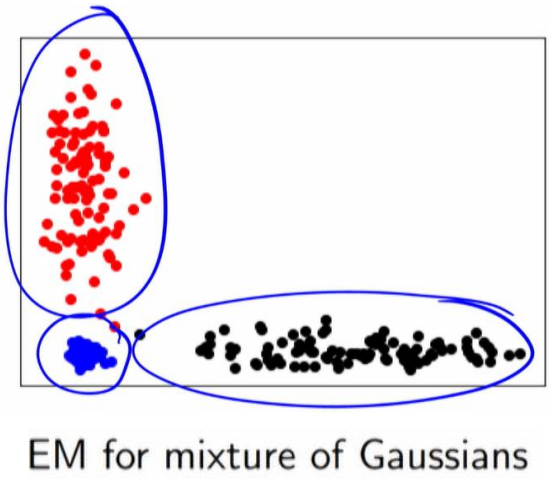
Example



So let's look at an example. Here's a data set in 2D, a toy data set and the intention is for it to have three clusters. Okay so one cluster down here, one cluster over here, one cluster over here and each cluster has got 100 points associated with it. Let's start by running K-means on this and see what happens.

- Okay so we run K-means and this is what it finds. So it finds one cluster like this, one like this, and one like this. Not exactly what we had in mind and if you remember K-means is also a local search algorithm which means that potentially each time you run it, it might produce a different answer.
- So let's run it again. So we run it a second time and indeed it gets a different answer. So this time it has this. This is one cluster, that's one cluster and that's one cluster. Different answer.
- Let's run it again. Okay this time it finds this. This is one cluster, this is one cluster and this is one cluster.

So why is it doing all these strange things? Basically **K-means** can get a little bit confused if the clusters are of very different sizes as is the case over here and different shapes.



Now let's see EM at work. So we run EM and this is what it does. So it finds exactly the right clusters. It finds the ones we had in mind and again it's a local search procedure so we can run it multiple times but if you do that, you find that it fairly consistently finds this same clustering. Why is it able to do this? Why is it able to accommodate these different shapes and sizes? Well it's able to do that because it explicitly models the shape of each cluster by fitting a Gaussian to it.

Okay so that is the EM algorithm in a nutshell. Next time we'll move on to an entirely different type of clustering, one that tries to find hierarchical structure in data.

POLL

The negative log-likelihood function is convex and is solved in polynomial time.

True	22%
✓ False	78%

总结:

8.5 Hierarchical Clustering

Topics we'll cover

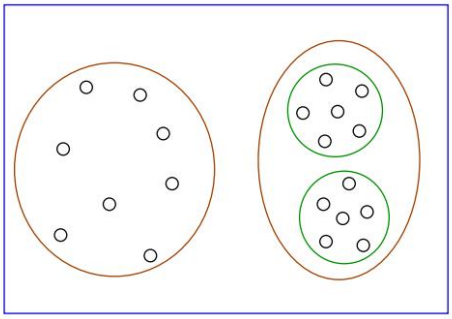
1. What is hierarchical clustering?
2. Single linkage
3. The other linkage schemes

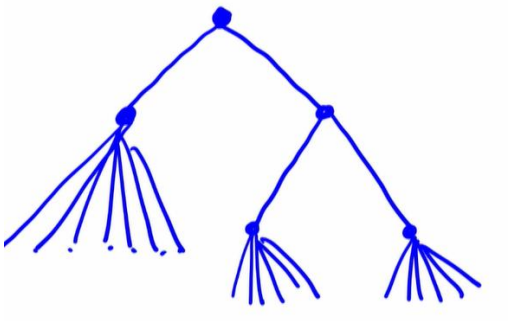
So far, we have talked primarily about two clustering algorithms, K-means and EM. Both of these algorithms recover what might be called **flat clusterings**, that is to say the clusters they find are all at a single scale. But when we introduced representation learning, we emphasized how the structure we're looking for often exists at multiple different scales. Well, the multi-scale version of clustering is what's called **hierarchical clustering**, and that's what we'll look at today.

So we'll begin by introducing the notion of hierarchical clustering, and then we'll go on to look at some popular algorithms for finding clusterings of this kind.

Hierarchical clustering

Choosing the number of clusters (k) is difficult.

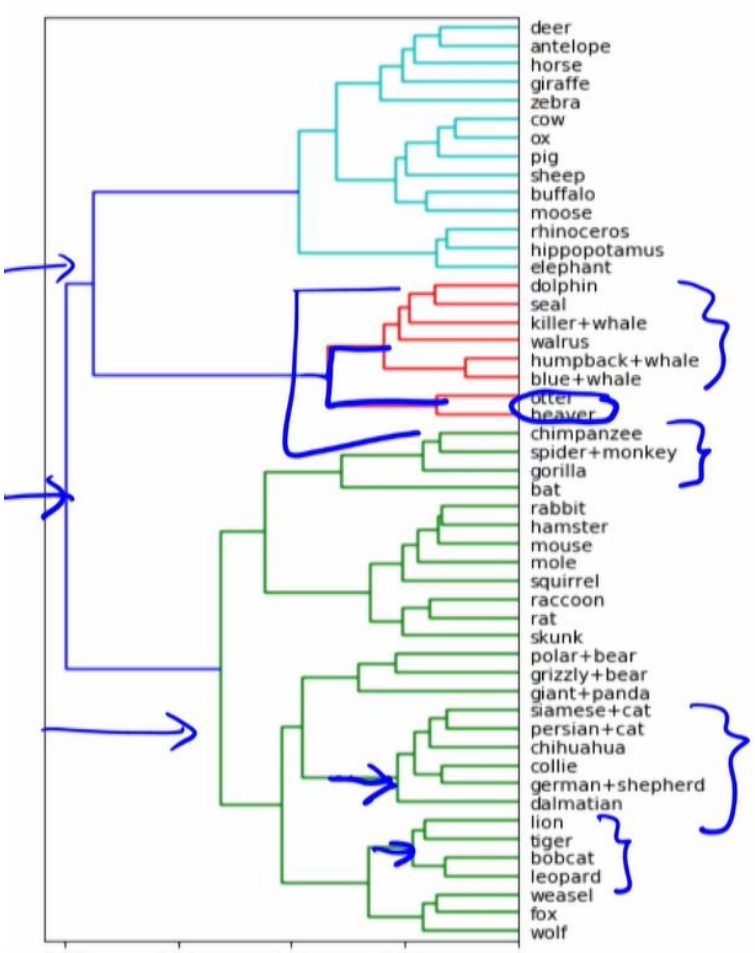




Often there is no single right answer, because of multiscale structure.

Let's look back at this dataset over here. How many clusters are there? Well, there definitely seems to be a cluster on the left. And on the right, is that one cluster or perhaps two clusters? Okay. It's hard to tell. And in fact, there's no right answer. Both choices are perfectly legitimate. Real data often has, often behaves in this way.
There's often cluster structure at multiple different levels, and a hierarchical clustering let's us capture all of these different levels simultaneously. The way a hierarchical clustering is usually depicted is as a **tree**. So at the top, there's a single node, that is one cluster containing all the data points. In this case, it would split into two, where you have the cluster on the left and the cluster on the right. The cluster on the right splits into two again. Now the cluster on the left contains maybe eight data points. So let me just draw them like this. And the clusters on the right, each have about six points. Let's just draw them like this. And the total number of leaves in this tree is then exactly the number of data points, which is 20. Okay, so this tree captures an entire hierarchy of cluster structure in the data.

总结:



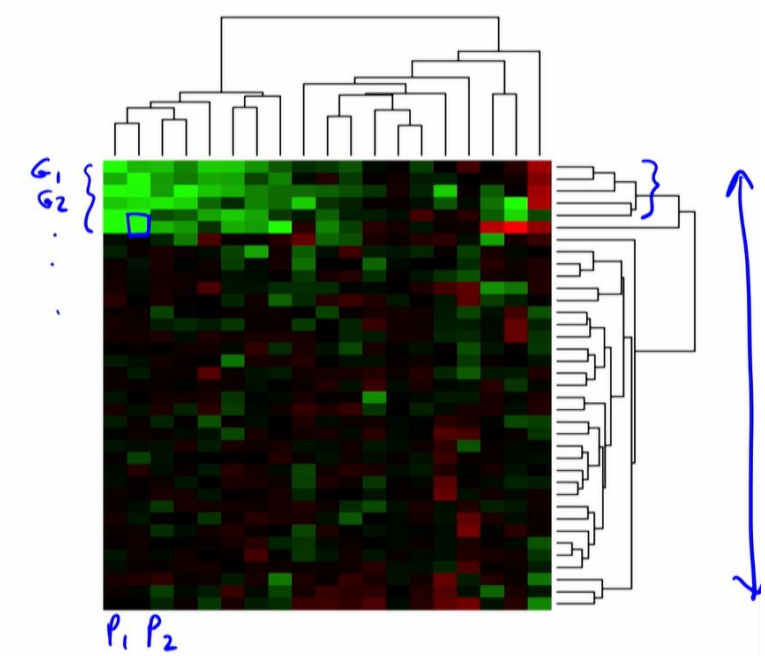
Let's look at a couple of examples. Do you remember the Animals with Attributes dataset? So this was a dataset in which we had information about 50 different animals. And for each animal, there were 85 features. An animal was represented by an 85-dimensional vector, 50 animals or 50 data points in R to the 85. So what I did here was to hierarchically cluster these data points by just using a standard algorithm called average linkage. We'll see the algorithm very soon. But in the meantime, let's just look at the results. It returns a tree with 50 leaves, 50 animals. And I've drawn the tree sideways so that I can spell out the animal names.

Let's look at some of these clusters.

- So over here, on the left, is just one cluster containing all the animals. And then it gets split in two, and then it gets split again and again. Let's take a little bit of a closeup at some of these clusters. Let's look at this cluster over here, the red one. What do we have there? Well, we have dolphin, seal, killer whale, walrus, humpback whale, blue whale, otter, and beaver. That seems pretty reasonable, animals that live in the sea or river or close to the river. The first split in that cluster is into these two groups, and we get this little cluster over here, which contains just otter and beaver. And then we get a larger cluster over here, that contains all the whales and dolphins. Again, a very reasonable way to split that cluster.

- Okay, let's look at some of the other ones. So there's a really big green cluster. So let's look at some of the smaller clusters within it. Over here, we have a cluster of three animals, chimpanzee, spider monkey, and gorilla. Primates, okay, very sensible. Over here, let's look at this.
- Let's look at the cluster represented by this branch over here. It contains Dalmatian, German Shepherd, Collie, Chihuahua, Persian cat, Siamese cat, basically domestic pets. Okay, also reasonable.
- We have another cluster over here that containslion, tiger, bobcat, and leopard, cats. Okay.

This whole thing seems fairly reasonable, We found it automatically just by running a hierarchical clustering algorithm on this animal data.



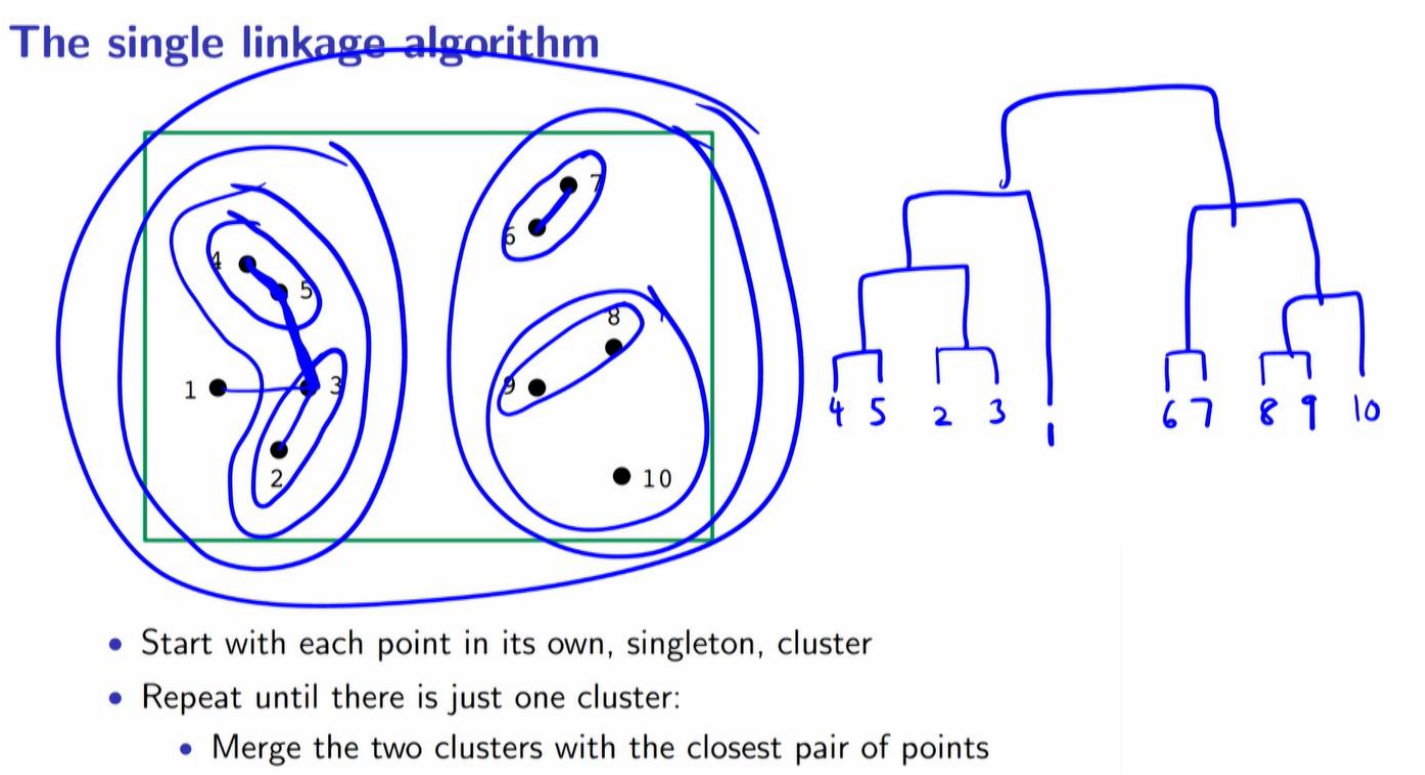
Now one domain in which hierarchical clustering has been quite popular is gene expression analysis. So what you see in this picture over here, let me describe what this is. So along the vertical direction, we have different genes. So let's just call them G one, gene one, gene two, and so on. And along the horizontal axis, we have different samples, which were, say, drawn from different patients. Maybe this is person one, person two, et cetera. Now each pixel, if you look at a given pixel over there, that represents how strongly that gene was expressed in the sample from that particular person. And the range is from bright green to bright red. Now, in these cases, it's common to use lots of genes, thousands of genes, in which case, the data becomes very high-dimensional. And a very nice way to understand it and simplify it is to apply hierarchical clustering.

So what's shown here, along the vertical axis, is a hierarchical clustering of the genes, and the way you obtain that is by simply thinking of each row as a data point.

There's a data point for each gene. And then we just apply a hierarchical clustering algorithm, again, like average linkage, to these points, and this is the tree that results. If you look at a cluster at the top, for example, like look at this cluster, it consists of maybe five or six genes. And these are genes that tend to act together, that are all either not highly expressed or highly expressed. They seem to behave as a unit. So this is a very nice way to understand the data. You start with a bewilderingly large number of genes, and then the hierarchical clustering finds you these gene clusters, these groups of genes that really act together as a unit. And that reduces the number of different things that you have to think about.

总结:

It's also common, actually, to cluster the patients, and the way you do that is to take the same dataset, the same matrix, and now think of each column as being a data point and then apply a hierarchical clustering to those. And the reason this is done, well, often the outcomes are quite interesting. For example, sometimes you think that all these people are suffering from the same disease, but then when you hierarchically cluster these columns, you see that there are actually two rather distinct groups. And this might be evidence that there are actually two subtypes of this disease, and maybe these two groups of people need different treatment.



So how does one do hierarchical clustering? So let's start by looking at the **single linkage algorithm**, which is perhaps the simplest of the hierarchical clustering methods. It's not the best algorithm, though. We will soon come to better alternatives. It's just a useful one 'cause it's rather simple to describe. So how does it work?
So let's say we have 10 data points, the 10 points you see over here. Single linkage works in a bottom-up fashion. We're going to start by thinking of these 10 points as being individual clusters. We're basically at the bottom of the tree, where we have our 10 leaves. Then we're going to merge two of them, and then we're going to merge another two and so on. And in this way, we build the tree in a bottom-up fashion. So we have our 10 clusters over here, and we begin by merging the two that are closest. I think that's four and five. So we start by merging these. Now we have nine clusters. Then we merge the next two that are closest. All right, I think that's six and seven. Now we have eight clusters. Then the next two that are closest, I guess that's three and two. Now we have seven clusters. Now which ones do we merge? So the next closest edge, the next smallest edge, I think, is this one. So we would actually merge these two clusters into a single cluster of four points. And so now the number of clusters we have at this stage is one, two, three, four, five, six. We have six clusters. Then we merge the next two closest points. What is the next closest edge? I think it's this one.
So we're going to create this cluster over here. And then we merge this one. And then we merge this. And then we merge the whole thing into one big cluster. Now this doesn't exactly look like a tree, but it's clear that there's a hierarchical structure over here. So it's easy to recover a tree from this.

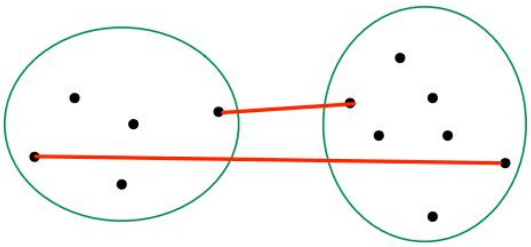
(画树) How do we do that? Well, the first thing we merged was four and five, so let's draw those, four and five. We merged those. Then the next thing was six and seven. Let's draw those. And then we did two and three. Let's draw those. And then we, I think we added, oh, then we merged these two into a bigger cluster, and then we added one to them. And then we did eight and nine. And then we added 10 to it. So the tree looks something like this.
Sometimes one assigns special significance to the heights of different points in the tree, but I'm not doing that over here. So this is the final hierarchical clustering of the data as found by single linkage. Now, there's a whole variety of linkage methods for hierarchical clustering, and they all have roughly the same flavor.

总结:

Linkage methods

- Start with each point in its own cluster
- Repeat until there is just one cluster:
  - Merge the two "closest" clusters

How to measure the distance between two clusters C, C'?



- Single linkage

$$\text{dist}(C, C') = \min_{x \in C, x' \in C'} \|x - x'\|$$

- Complete linkage

$$\text{dist}(C, C') = \max_{x \in C, x' \in C'} \|x - x'\|$$

To be honest, single linkage is probably the worst of them. So they all work in a bottom-up fashion. So you start with each point in its own cluster, and then you merge the two closest points and then the next two closest and so on. And in this way, you build up the tree. So at any given stage, you have a bunch of clusters, and you have to join the two clusters that are closest. How do you measure the distance between two clusters? That's the only degree of freedom that remains, and that particular choice makes all the difference.

So there are a variety of different linkage methods, and the way in which they differ is by the way in which they define the distance between two clusters. At any given stage, we compute these distances, and we merge the pair of clusters that are closest, that have the smallest distance.

- Single linkage, if you give it two clusters, C and C prime, so this is C and this is C prime. Single linkage thinks of the distance between these two clusters as simply being the distance between the closest pair of points in those clusters. It thinks of that as being the distance. So that's, in essence, the smallest possible notion of distance between these two clusters.

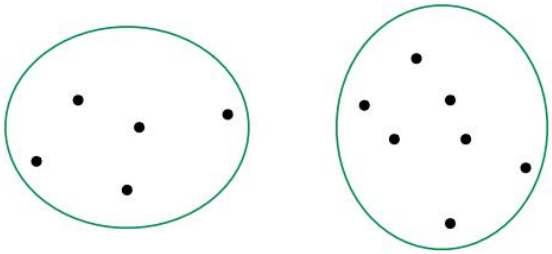
- Complete linkage is a different kind of linkage algorithm that is the other extreme. It defines the distance between two clusters as being the furthest pair between those two clusters. So it's between these two points. If you think of different ways to define distance, this is probably the largest you could come up with. And as you can imagine, complete linkage tends to produce clusters that are fairly tight, that are fairly compact.

So these are two extremal notions. Single linkage defines the distance to be something rather small. Complete linkage defines it to be the distance between the furthest pair of points.

- The most popular clustering methods, the most popular linkage methods take a somewhat intermediate stance. Rather than look at the closest pair or the furthest pair, they look at the average distance between the two clusters, and that's what's called average linkage.

总结:

Average linkage



1. Average pairwise distance between points in the two clusters

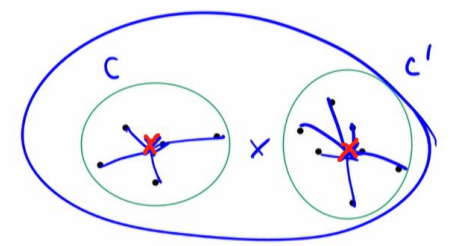
$$\text{dist}(C, C') = \frac{1}{|C| \cdot |C'|} \sum_{x \in C} \sum_{x' \in C'} \|x - x'\|$$

2. Distance between cluster centers

$$\text{dist}(C, C') = \|\text{mean}(C) - \text{mean}(C')\|$$

3. Ward's method: increase in k-means cost from merging the clusters

$$\text{dist}(C, C') = \frac{|C| \cdot |C'|}{|C| + |C'|} \|\text{mean}(C) - \text{mean}(C')\|^2$$



Now it turns out that there are actually several different ways in which you can define this average that are all quite reasonable. And these lead to different average linkage algorithms. So I know of three different ways in which this commonly gets done, so let me tell you what they are.

- The first method is to just look at all pairs of points in the two clusters. So take one point from here and one point from there, look at the distance between them, and then average this over all pairs. So if there are N points on the left and N on the right, there are N squared pairs that you're looking at, and you're taking the average distance. So that's the formula you see over here. You look at all pairs of points, one from the left and one from the right.
- A potentially simpler way to define the distance between two clusters is to say let's just look at their centers. Let's look at their means, and let's just look at the distance between the means. So you look at the cluster on the left. You say that's the mean. Look at the cluster on the right. You say that's the mean. Let's just measure the distance between the means. That's it, that's going to be our notion of distance. So that's also fairly popular.
- There's a third option that's kind of interesting. And this is usually called **Ward's method** for average linkage. And what this does is it's also a kind of average notion, but what it says is that look at the increase in K-means cost from merging these two clusters. Okay, that's going to be the notion of distance. So what exactly does that mean? Well, if you look at this picture, there are two clusters, C and C prime. And if you look at the **K-means cost**, what is the K-means cost? It's just the distance to the closest center. So it would be this distance plus that distance, plus that distance, plus that distance, plus that distance, and you have to square the distances. And then this plus this, plus this, plus this, plus this, plus this, plus this, and you have to square the distances. So that's the starting K-means cost. Now when you merge these two clusters, then you're going to just have one cluster. You're going to just have a big cluster over here. And you're going to have a new mean, which is going to be somewhere in the middle over here. And so the new K-means cost will just be the distance of all the points in C or C prime to that new mean, added up, squared and added up, okay. So you have the cost before, and now you have the new cost, which is going to be larger. What is the difference between those two costs? That's how we're going to define distance. And it turns out that the difference is given exactly by this formula over here. It simplifies to something like this. And actually, it looks a whole lot like option number 2. The main difference is that it's the distance between the means, but this time you have a factor in front that basically is some measure of the size of the clusters, in terms of how many points are in the clusters. So it's kind of like option 2, but it's saying give more emphasis or prefer clusters that have fewer points in them.

So that concludes our discussion of hierarchical clustering. What we'll do next time is to look at an entirely different form of representation learning, projections.

总结:

POLL

How is a hierarchical clustering usually represented?

结果

As a graph	3%
As a tree	92%
As a forest of decision stumps	4%
As a set of overlapping Gaussian distribution functions	1%

总结: