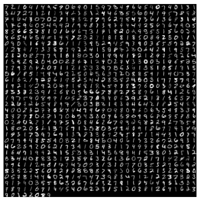
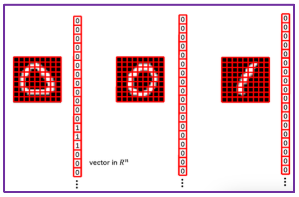
Today we'll talk about clustering, which is a basic unsupervised learning algorithm, meaning that we don't have any label for our data points.

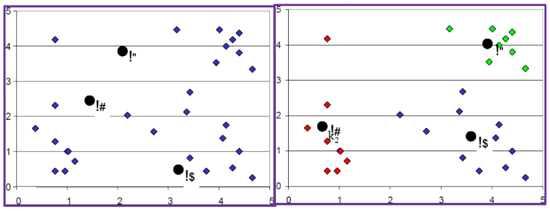
Cluster images: So, I’ll start with that motivating example we talked about briefly during introduction. Suppose you have a pretty large image database, and you collect tons of images, and these images are about different subjects, for example, about a sea, about flowers, sunset, and so on. And without any label about these images, the question is, can we group these pictures into clusters based on their similarity? As you can see clearly, for example, these images about the beach, have similar color and similar shapes, about the seashore and so on. So, clearly there are some similarities between the images. So, based on that, the **goal** of clustering is to divide the object into groups and hopefully the objects within the group are more similar than those across different groups.



Cluster handwritten digits: Here's another mode example of clustering, the handwritten digits. This is a very classic example in machine learning, and this data is the handwritten digits called MNIST dataset. And this dataset is collected from the USPS mails where people read ZIP code on the cover of the envelope, and we take scan pictures of their handwritten digits and do preprocessing and extracting these handwritten digits by different people. As you can see, of course, there are people have different ways of writing digits, there are different ways of writing seven. Slightly different, although you can still recognize this is a seven, but it looks different. And so, the question here is, based on these images we put them together, can we find clusters of images, for example, cluster of images all about seven and cluster of images all about one and two? Can we find an algorithm to perform this task automatically based on the similarities of these little images? And in fact, in this class, we will do multiple exercises, our homework based on this image dataset. This is one of the simpler and widely used datasets for testing too many algorithms because it's fairly simple and has been pretty well preprocessed and so, you can extract features from these images.

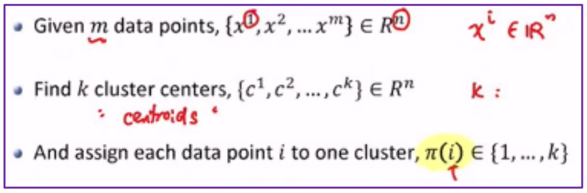


How do we represent these images? We will define features from these images. So, you can look at each tiny little picture in here, it's actually consisting of these pixels. And so, for example, in this **binary image**, and we could say if there is no stroke is a zero, if there's a stroke that it is a one. So, by representing this image by pixels and this is going to give us a matrix is square, and the size of the matrix is the number of pixels along the rows and columns. And the entries in this matrix are going to be zeros and ones. And one thing we typically do is, we'll vectorize this matrix. Meaning that we're going to extract the columns of this matrix and stacking them together to form a pretty long vector. This is called the **vectorize matrix**. After vectorizing the matrix, we end up having a pretty long vector, this vector represent this image. And we're going to define this vector as dimension n, n is the number of pixels in the image. There's notation we're going to see many times in the class representing this is n-dimensional vector in the Euclidean space, the value of each entry is a real value R^n. So, basically you end up having a bunch of zeros and ones after you represent each little image by the vectorized pixels. And so, how do we approach this problem? Before the clustering, each data points now is this little image, it's represented using this long vector. Now we're going to talk about, where do these lectures live? In linear algebra, these vectors define a vector space, these images leading to vector space. And so, you can imagine this vector space is going to be pretty high-dimensional, as we see earlier, is n-dimensional. For example, these little images consist of, I think, nine rows and nine columns roughly, so it's going to be about roughly 90 pixels or something. But you can imagine they live in a high-dimensional space. And then each data points here representing one image. I draw this in a two-dimensional, but you can imagine this is actually happening in a very high-dimensional space. So, before clustering, I don't know what these images are. There could be any of these 10 digits from 0-9. But if you look at where they live in this high-dimensional space, the digits that are similar to each other will be pretty close to each other in this high-dimensional space. So, based on that, can we cluster and divide these data points into groups such that the data points belong to the same cluster will highly be likely to be the same cluster? That's the idea. So, the algorithm we describe next will perform this task.



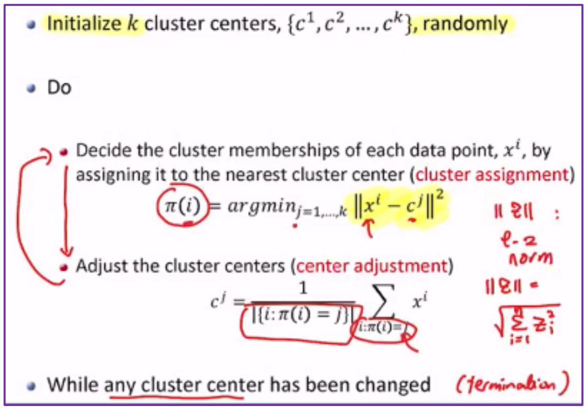
Before clustering, a bunch of vectors: So, to start with, we will assuming the so-called **centroids**. In this example we're going to start to choose three centroids. And why three? For example, you believe that there are roughly three clusters in your data. And so, how to choose the number of clusters is also a question to parameters about two in this algorithm.

After clustering, group assigned: Once, we choose the centroid, then we can assign the data points from the centroids based on how close they are to the centroids. So, for example, if you define Euclidean distance, then we can find out the data points close to each of the centroids and gave it a label this time. So, these data points are close to this first centroid and they're all colored in green as data points belong to this particular cluster. And these ones, based on the calculation, these are the points closest to this particular centroid, and these are close to this particular centroid. And so, after this, you have three assigned groups to each of the centroids. And so, once you have assigned the group to each of the samples, you realize that this initially chosen centroid in these locations may not be the best locations. And so, based on that, we can adjust the centroids based on the assignment. That is simply done by performing the average of the data points and average of the data points in the same group is going to be assigned as the new centroids.



- Given $\underset{\sim}{m}$ data points, $\{x^{①}, x^2, \ldots x^m\} \in R^{ⓝ}$ $x^i \in \mathbb{R}^n$
- Find k cluster centers, $\{c^1, c^2, \ldots, c^k\} \in R^n$ k :
  : centroids .
- And assign each data point i to one cluster, $\pi(i) \in \{1, \ldots, k\}$

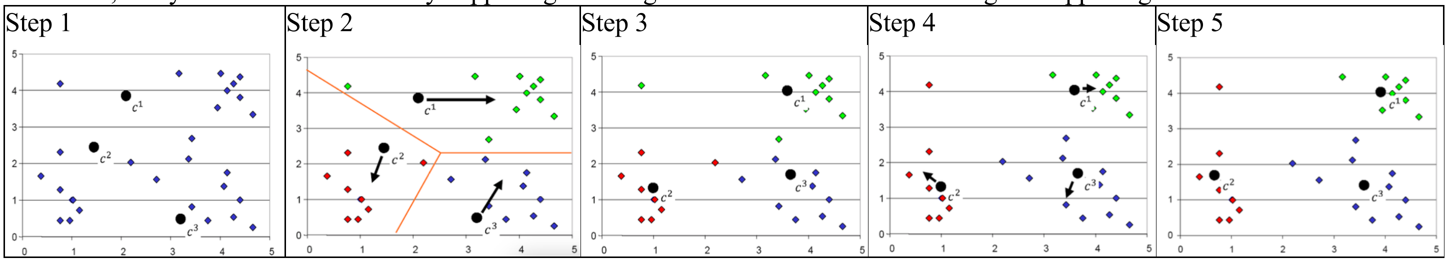
Formal statement and goal of k-means: This very simple idea becomes our first algorithm called the **K-means algorithm**. So, I'm going to describe this algorithm below using pseudocode, as you can see here, we're going to start to introduce some mathematical notation. Given m data points, here this m represents number of data points, so this is going to be the commission in this class, throughout the whole class, m is going to denote the number of the data points. And each of the data points is n-dimensional vector R^n. Remember n is the number of pixels for each image in this example. And one commission we also have is, we're going to put the index for the sample as a superscript to the x. And with understanding that any of these x^i is a n-dimensional vector R^n, they're representing a vectorized image. And next is, we're going to find a k cluster centers. These are the centroids that we were talking about just now, these k cluster centers. And k is the number of clusters. And this k have to be pre-specified. And based on that, we will find an assignment of data points to one particular cluster. And you can imagine this assignment is a function. This function is Pi, its argument is i, and then its outputs is one of the integer values between 1 and k. And so, this basically tells you the i's data point belong to one of these clusters.



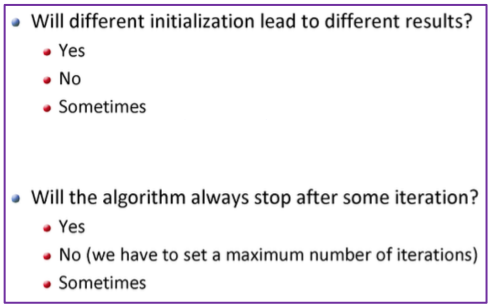
- Initialize k cluster centers, $\{c^1, c^2, \ldots, c^k\}$, randomly
- Do
  - Decide the cluster memberships of each data point, x^i, by assigning it to the nearest cluster center (cluster assignment)
    $$\pi(i) = argmin_{j=1,\ldots,k} \|x^i - c^j\|^2$$
    $\|z\|$: l-2 norm
    $\|z\| = \sqrt{\sum_{i=1} z_i^2}$
  - Adjust the cluster centers (center adjustment)
    $$c^j = \frac{1}{|\{i : \pi(i) = j\}|} \sum_{i:\pi(i)=j} x^i$$
- While any cluster center has been changed (termination)

K-means algorithm: what is this **algorithm**? Here is a **formal description of K-means** and we will first have an initialization. As you can see, a lot of the algorithms we talk about in this class will have some initialization. Here we will initialize the centroids randomly. Your homework, you'll practice K-means yourself and you will see how to randomly assign the centers. For example, we can set these as multivariate normal random vectors for each of the centroids. After that, we will iterate between the two steps in this algorithm. And these two steps correspond to assigning clusters for each data point. And then adjust the centers for the one cluster after we have done the assignment. So, and then you're going to do this. And then coming back and then repeating these two steps. So, if you look into each of the steps in the center assignment. The first thing is we're going to assign data point to a particular center using distance. This notation here represent the l-2 norm of a vector. And so, what does it mean? This z is going to be square root of the sum of the entries of this. That's our usual definition of the distance in the vector space. And so, this is a squared l-2 norm. And so, basically you find out for each data point which centroid is the closest, and then you're going to assign it to the closest centroid. So, this Pi(i) is the argmin finding out the closest, meaning the smallest distance for this particular data points to that particular centroid. After that, we're going to adjust the centers. Once you have these Pi's, these Pi's assign i data point to put a particular cluster. And so, basically this summation here, you can see this representing that finding out all the data point that has been assigned to the j's cluster. Then you will perform an average of this exercise. So, you're going to find out how many points has been assigned to that particular (j's) cluster. And so, this notation here means this. So, this is nothing but performing the average of that. And that's it. So, after the adjustment of the center, and you will realize it's possible that because the center is a different, so the assignment of the points will be changing as well. So, you will go back to the first step and reevaluate exact point of the data points and then go to the next adjust the center as well. You keep doing these two steps until there is no further change. So, there's no change in this cluster centers. And then this is a termination of your algorithm.

K-means: So, let's see what's going to happen after we run this algorithm iteratively. Again, we use this very simple example. This is what it looks like to start with. We have three randomly chosen centroids, and then we will assign the data points to the closest centroids. And then these points in green has been assigned to this first centroid and otherwise second and third, and so on. So, essentially by this assignment, we are performing a partition of the space. So, this partition of the space is formed by finding the closest distance of the data points to its centroids. And after the assignment, the next step is to adjust the centroids and basically performing the average of these points that has been assigned to the first group. And then if you perform that, you can see this tends to initialize the centroids into some position here. And this might be adjusted to here and here and so on. So, after the first assignment and adjusting centroid, this is what's going to happen. And then from here, you can do the same thing again, find out the assignment. And based on the new centroids, as you can see, some points, the assignment will be different. For example, you will see next to the integration. And then based on the new assignment, you will do adjustment of the center again and iterate until there's no further change. So, you can flip through this animation, and you will see what's really happening here to get an idea until no further change is happening to the centroids.

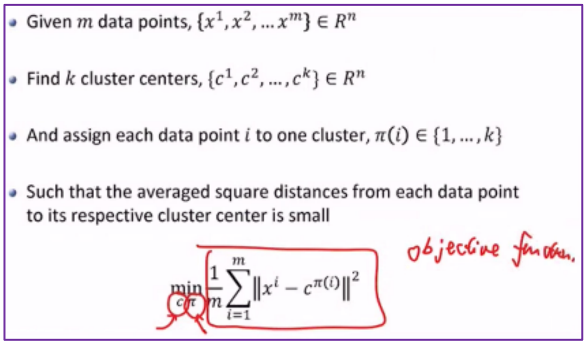


Questions: So, we have to swap it out with them and see the steps. And here is the time for you to think about what possible results are can happen and possible properties of this algorithm. First question, will different initialization lead to different result? Yes, no, or sometimes? And second question is, will the algorithm always stop after some iterations? For the first question, yes. For different initialization we have different results. For the second question, yes. We'll explain the answer to these questions in the next a few slides. So, first of all, give a little bit iteration, iteration because a different initialization leads to different results. But first of all, basically, we are solving the so-called non-convex optimization problem for the K-means. For non-convex optimization problem, there are many local solutions, and we only end up with one of the local solutions. It's only locally optimal. And so, because of that different starting point might lead into different local solutions. So, you might have different results based on different starting points. And second yes, it will stop. The reason is after a finite number of iterations. Since you have a fixed number of data points, there's only a finite number of ways to perform enumeration, and combinations of these points to the centroids. Although it's a big number, but there's a finite number of ways. And our algorithm always try to reduce a certain metric. And so, after many, many iterations, could be many iterations, but it's going to converge because it keeps decreasing until the point you cannot decrease further.



- Will different initialization lead to different results?
  - Yes
  - No
  - Sometimes
- Will the algorithm always stop after some iteration?
  - Yes
  - No (we have to set a maximum number of iterations)
  - Sometimes

Formal statement of K-means objective: So, here's a formal statement of the K-means objective. Given m data points, we want to find k centroids or cluster centers and find out assignment function. And so, what is our objective function? We want to minimize the average square distance from each data point to its respective cluster center. We want to minimize overall sum of squares errors representing these centroids. That's it. So, as you can see, this optimization problem involves two things that we want to minimize. One is the Pi, these are the assignment function and the other is c, the centroids. You can see that's why we have these two steps in each iteration. We're basically adjusting the assignment function and also the centroids in this. Then the objective function, the so-called **objective function**, is defined as the sum of the squared l-2 distance here.

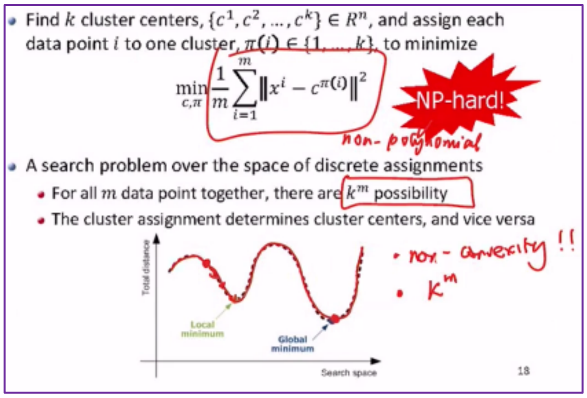


- Given m data points, $\{x^1, x^2, \dots x^m\} \in R^n$
- Find k cluster centers, $\{c^1, c^2, \dots, c^k\} \in R^n$
- And assign each data point i to one cluster, $\pi(i) \in \{1, \dots, k\}$
- Such that the averaged square distances from each data point to its respective cluster center is small

$$\min_{c,\pi} \boxed{\frac{1}{m}\sum_{i=1}^{m}\left\|x^i - c^{\pi(i)}\right\|^2}$$

objective function

Clustering is NP-hard in general: So, we have an optimization problem to solve and how easy or difficult it is to solve this. It turns out, unfortunately, this problem, if you want to solve it exactly, it is **NP-hard**. What does NP mean? It means **non-polynomial**. That means, if you want to solve this problem, find out the exact optimal solution. So, what does it mean? If you plot the value of the objective function, so this is a total variation, this is the objective function. And then you will end up having this shape. So, you can see this is the actual minimum of the objective function that's called the **global minimum**. But there will be some places like a valley, and these are not as small as a global minimum, but they're also pretty small. They're smaller than all the neighbor points. These are called **local minimums**, and so, basically, this objective function is going to take this ugly shape that is pretty hard to find the global minimum. If you start from here and then you go down, you might be trapped in one of the local minimums and couldn't escape from there. So, that's one challenge because of the non-convexity. And because of this, we cannot use a very simple iterative algorithm to find the exact solution. And if you really want to find out the exact solution, because the number of ways to design data points to a centroid is going to be k raised to power m. So, there's number of possibilities. Why? Because for each data point, we can design into one of the centers. You have k centroids. So, you will have k times k times k, multiply this by m, m is a number of samples. So, there are total k raised to power m possibilities, which is an astronomical number. So, that's why it's a very difficult problem, is NP-hard to solve.



- Find k cluster centers, $\{c^1, c^2, \dots, c^k\} \in R^n$, and assign each data point i to one cluster, $\pi(i) \in \{1, \dots, k\}$, to minimize

$$\min_{c,\pi} \boxed{\frac{1}{m}\sum_{i=1}^{m}\left\|x^i - c^{\pi(i)}\right\|^2}$$

NP-hard!

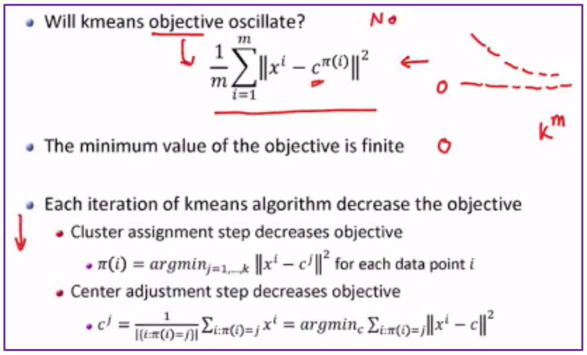
non-polynomial

- A search problem over the space of discrete assignments
  - For all m data point together, there are $\boxed{k^m \text{ possibility}}$
  - The cluster assignment determines cluster centers, and vice versa

- non-convexity !!
- k^m

18

Convergence of K-means algorithm: So, because of that, we're not going to solve an exact global optimal solution. We're only going to solve a local solution, and then this heuristic algorithm end up solving this problem proximately actually give rise as K-means algorithm. So, the convergence, we're talking about, whether or not is it going to converge and finite number of steps. It actually will converge to finite number of steps, and so, the first thing is you can think about question. Will K-means objective function oscillate? And the answer is no. It will not oscillate; it will be monotonically decreasing. If you look at this objective function, this is the objective function. And why? Because if you think about the two steps you're doing in an iteration,



- Will kmeans objective oscillate? No

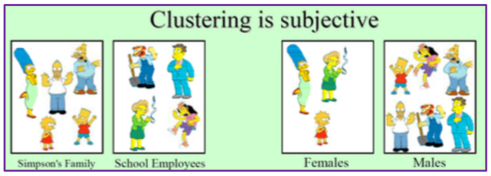
$$\frac{1}{m}\sum_{i=1}^{m}\left\|x^i - c^{\pi(i)}\right\|^2 \leftarrow$$

0

k^m

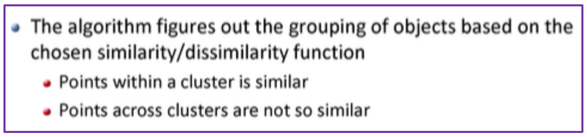
- The minimum value of the objective is finite 0
- Each iteration of kmeans algorithm decrease the objective
  - Cluster assignment step decreases objective
    - $\pi(i) = argmin_{j=1,\dots,k} \left\|x^i - c^j\right\|^2$ for each data point i
  - Center adjustment step decreases objective
    - $c^j = \frac{1}{|\{i:\pi(i)=j\}|}\sum_{i:\pi(i)=j} x^i = argmin_c \sum_{i:\pi(i)=j}\left\|x^i - c\right\|^2$

they're always going to make the change only if you're going to improve, reduce this objective function. If your new or new centroid is not going to reduce objective function, you're not going to make the change. Because of that property, your objective function is always going to get monotonically reducing. The second question is the minimum value of the objective function. Look at this, if you choose any c and any Pi, what is the smallest positive value that your objective function could be? It's going to be zero because the squared numbers cannot be negative. So, sum them together, the smallest you can possibly be zero. So, basically, we're forming a sequence of objective functionalities in that equation. And then they're monotonically approaching to zero. Because of that, there is some theory in mathematical analysis guarantees that your algorithm, it's going to converge in finite number of steps. It's going to converge for sure. And then because you have only finite number of both combinations, which is a large number, which is k raised to power m, and it still is a finite number, so you will converge in finite number of steps. So, this bullet point explains further why each iteration will decrease the objective function.



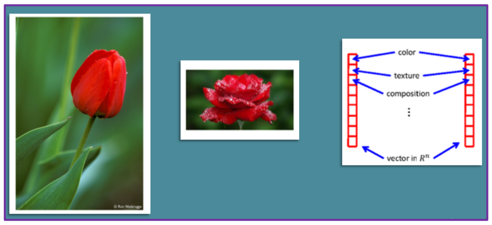
What else needs to be considered? So, now we have a pretty good understanding of paintings. And so, there's one interesting point I would like you to think about, which is, how do we define similarity in the K-means algorithm. So, the definition of similarity is quite subjective if you think about it. And here I get this very interesting example. These are different characters from Simpson family. As you can see, if I want to group these characters into groups, depending on your metric, you could group them differently. And for example, if I define a group as Simpson's family, and school employee, and I have reduced two groups. But if I define the group metric as females, and males, then I will end up having different ways to group them. So, you can see this is going to change the result of my cluster. If I change what is my definition of similar and dissimilar.

You pick your similarity/dissimilarity: Here's another interesting example. A girl with a curly hair, and a puppy with a curly fur. Do you think they are similar or dissimilar, depending on your metric? If I define the similarity as both having curly hairs, and fur and maybe there similarities, is pretty high. But of course, there are different. If you define, then as a human versus non-human objects they are different. So, this example to show you this idea.



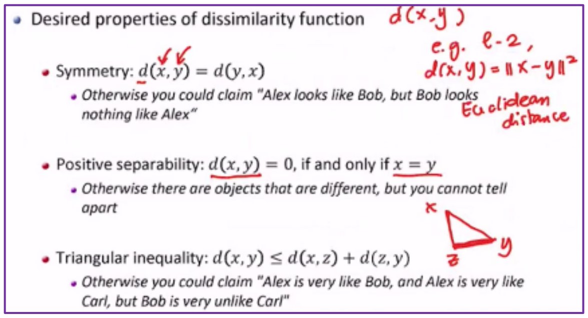
- The algorithm figures out the grouping of objects based on the chosen similarity/dissimilarity function
  - Points within a cluster is similar
  - Points across clusters are not so similar

So, what is clustering in general? So, how do we generalize the K-means algorithm to be able to account for the general definition of similarity. So, as we revisit K-means, you will see we can do this by simply choose the similarity function, and the K-means algorithm. You can pick your own similarity or dissimilarity function. And then the rest of the stuff in K-means will be the same by finding the most similar ones as the centroids, reassign them, and so on.



Images of different sizes: So, again, let's look at some examples illustrating what do I mean by different definitions for similarity functions. And so, for example, how do I define a comparison or similarity between images of different sizes. So, we talk about these amnesty handwritten digits to start ways, but these images are the same size. If I have different images, these are both about flowers. One in about two weeks the other it's about rose. And instead of defining them using raw pixels, I could define them differently. For example, I define the features as the color, the texture, composition, and so on. So, you can see this time these features are not the values of the pixels, but they are something you can extract out of the picture. These are different features. So, that's another term. When you see many terms features. So, this way you can define more abstract way of representing your data.

Object in real life: And then the object in real life, you can also define clearly features, for example, in terms of their family, their gender, the working place, their age, their preference, and so on. And so, you have many different ways defining features. For instance, family, and this is going to end up giving me a vector.

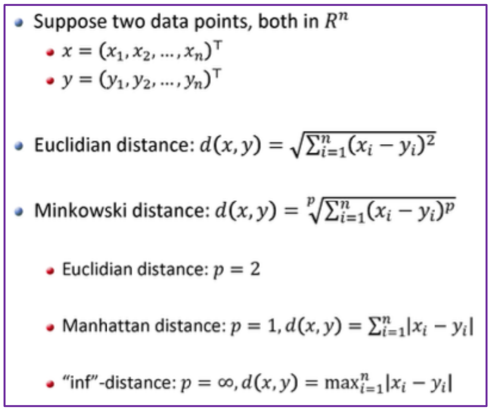


- Desired properties of dissimilarity function $d(x,y)$
  - Symmetry: $d(x,y) = d(y,x)$
    - *Otherwise you could claim "Alex looks like Bob, but Bob looks nothing like Alex"*
  - Positive separability: $d(x,y) = 0$, if and only if $x = y$
    - *Otherwise there are objects that are different, but you cannot tell apart*
  - Triangular inequality: $d(x,y) \leq d(x,z) + d(z,y)$
    - *Otherwise you could claim "Alex is very like Bob, and Alex is very like Carl, but Bob is very unlike Carl"*

e.g. ℓ-2, $d(x,y) = \|x-y\|^2$ Euclidean distance

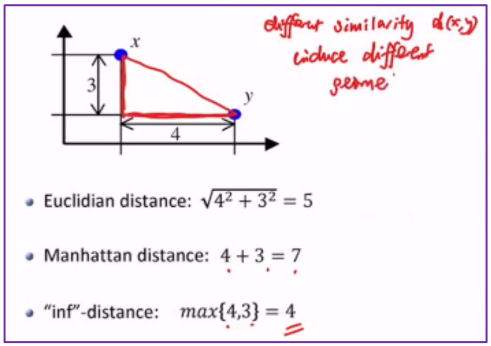
What similarity/dissimilarity function? The question next is, once each data point is represented using a vector, this natural contains values of the features you have chosen. And then measure the similarity between these data points it's a problem of choosing a function to measure the distance between two feature vectors. So, we're going to say how do we define this feature vector? And for example, if you revisit the K-means, you will realize that we actually have made the choice that is similarity function is x minus y square, the squared out two distances. But why did we choose that? You can totally choose it to be something else, and here that's the point we want to generalize. And <u>what is a good similarity function that can be used here to replace the Euclidean or L-2 distance</u>, square. L-2 distance, sometimes called the **Euclidean distance**. This is a square root out of 2 distance or square root instance. So, firstly, it has to satisfy some **symmetry**, and so, d(x, y) is equal to d(y, x). Because you want to make the claim that if a looks like b and then b is the same as a, the argument doesn't hold, then it's tricky, and the second, **positive for separability**. So, if the two objects or two data points have zero distance, this can only happen if these data points are exactly the same. You want to define this way because you want to differentiate the object if they're truly different. If you mark the two objects that are different into zero distance, then that's going to cause the ambiguity here. The next one is **triangle inequality**. So, basically this says if there's x and there's y and there's a z, and you will say the distance between x and y should always be smaller than the distance of xz, and then the distance from z to y, so satisfy the triangle inequality. So, your d satisfy these requirements is going to be a legitimate definition of similarity function.

Distance functions for vectors: Here are some examples of similarity functions. So, again, given two vectors that are n-dimensional and Euclidean distance, or l2 distance that we have used in the K-means. Algorithm so far is based on this definition, the sum of the squared distance of the entries. And this can be generalized to the so-called **Minkowski distance**, and against you, instead of two, I could define as two as some of the p's power of difference in the norm, and then you take the p's square root of that. So, Euclidean distance defined definition is according to the p is equal to 2, and actually, why you take p is equal to 1, it's called the **Manhattan distance**, this is the sum of the absolute difference between x and y entries. And so, why is it called Managed distance? We'll have y example illustrating in the next slide. And by the way, if you set p is equal to infinity, then this distance is called **infinitive distance**, that's basically the maximum absolute distance between entries in the two vectors.



- Suppose two data points, both in R^n
  - $x = (x_1, x_2, \ldots, x_n)^\top$
  - $y = (y_1, y_2, \ldots, y_n)^\top$
- Euclidian distance: $d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$
- Minkowski distance: $d(x,y) = \sqrt[p]{\sum_{i=1}^{n}(x_i - y_i)^p}$
  - Euclidian distance: $p = 2$
  - Manhattan distance: $p = 1, d(x,y) = \sum_{i=1}^{n}|x_i - y_i|$
  - "inf"-distance: $p = \infty, d(x,y) = \max_{i=1}^{n}|x_i - y_i|$

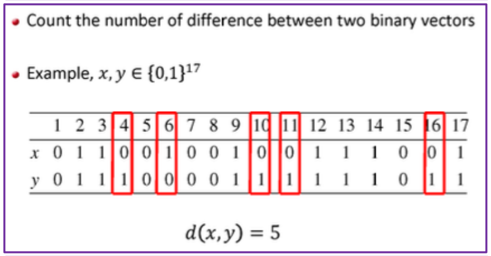
Distance example: Here is my example illustrating the difference between these norms. And so, in this example, we have the two-dimensional vector, x and y, and so, they form this relationship. So, Euclidean distance between x and y, because of the so-called **Pythagorean theorem**, is going to be the distance this way in x and distance in the other coordinate squared, and take the square root is going to be five. And the Manhattan distance between the two is going to be the sum of the absolute difference between the two data points. So, in this particular direction, the difference is 4, in this direction with different 3, so the sum of the difference becomes 7. The infinity distance is going to be the maximum of the difference in each of the coordinates an app before. So, you can see, depending on the choice of really of the p of the metric, you will end up having different values of your difference. In other words, by choosing different similarity measure, it will induce different geometry. So, different d will induce different geometry. And keep this in mind because this concept may show up again when we talk about, for example, feature selection bigger on the last algorithm.



differnt similarity $d(x,y)$ induce differet geome

- Euclidian distance: $\sqrt{4^2 + 3^2} = 5$
- Manhattan distance: $4 + 3 = 7$
- "inf"-distance: $max\{4,3\} = 4$

Hamming distance: So, the distances is induced by this Minkowski definition. In general, Manhattan distance is also called the **Hamming distance**, when the features are binary and says, basically counted the number of entries that are different. So, for example, you have 17-dimensional vectors, x and y, as you can see, and so, the Hamming distance between these two is going to be five as the total number of entries that are different from each other.

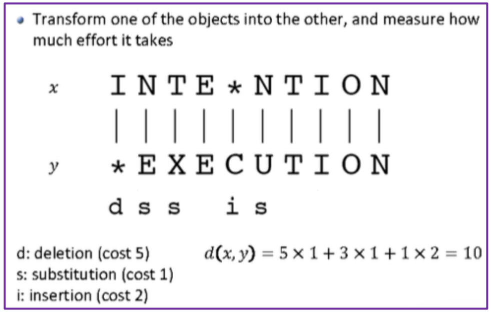


- Count the number of difference between two binary vectors
- Example, $x, y \in \{0,1\}^{17}$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
x	0	1	1	0	0	1	0	0	1	0	0	1	1	1	0	0	1
y	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	1	1

$d(x,y) = 5$

Edit distance: And natural language processing, understanding English, and there's something called the **added distance**. This is the number of edits that has to be made to transform one string train into another train strings. So, for example, this is intention into execution. You can see how many edits, for example, we define deletion that's caused by substitution it's caused one, insertion has caused two, and you have to make one deletion and then three substitutions, and the one insertion, the total added distance becomes 10.



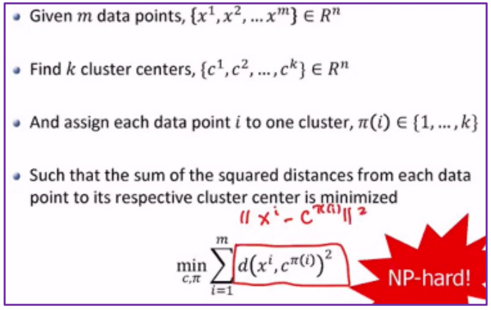
- Transform one of the objects into the other, and measure how much effort it takes

```
x   I N T E * N T I O N
    | | | | | | | | | |
y   * E X E C U T I O N
    d s s   i s
```

d: deletion (cost 5)
s: substitution (cost 1)
i: insertion (cost 2)

$d(x,y) = 5 \times 1 + 3 \times 1 + 1 \times 2 = 10$

Generalization of K-means: You can verify this is the valley distance as well. So, now we have this generalized definition of similarity measures, we can generalize our K-means, so we can see the only difference now from K-means. This is the first important difference, let's put it this way. It's replaced the l2 distance, so before this is the l2 distance between x^i data point from the centroid square. Now, we can replace this with a general definition of the similarity measure.

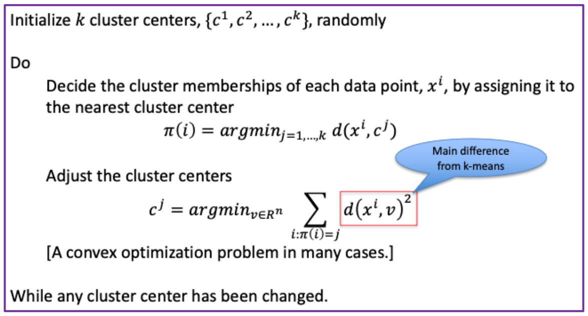


- Given m data points, $\{x^1, x^2, \ldots x^m\} \in R^n$
- Find k cluster centers, $\{c^1, c^2, \ldots, c^k\} \in R^n$
- And assign each data point i to one cluster, $\pi(i) \in \{1, \ldots, k\}$
- Such that the sum of the squared distances from each data point to its respective cluster center is minimized

$\| x^i - c^{\pi(i)} \|^2$

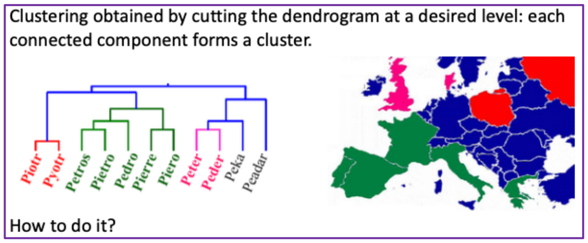
$$\min_{c,\pi} \sum_{i=1}^{m} \boxed{d\left(x^i, c^{\pi(i)}\right)^2}$$

NP-hard!



```
Initialize $k$ cluster centers, $\{c^1, c^2, \dots, c^k\}$, randomly
Do
    Decide the cluster memberships of each data point, $x^i$, by assigning it to the nearest cluster center
        $\pi(i) = argmin_{j=1,\dots,k}\, d(x^i, c^j)$
    Adjust the cluster centers
        $c^j = argmin_{v \in R^n} \sum_{i:\pi(i)=j} \boxed{d(x^i, v)}^2$
    [A convex optimization problem in many cases.]
While any cluster center has been changed.
```

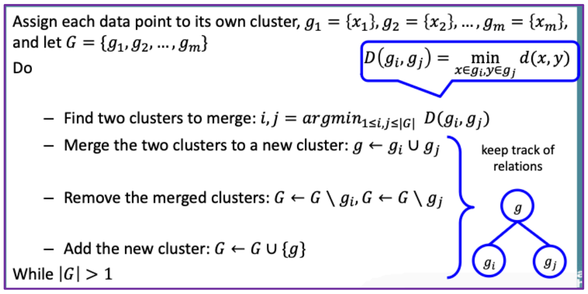
Generalized K-means algorithm: based on that, we're going to do similar two-step iteration, very similar to the K-means as before, but this time is **generalized the K-means.** You're going to initialize k -luster centers randomly and decide the cluster memberships of each data points, but this time you can see the examined is going to be based on the choice of similarity measure, this new d, instead of l2 distance, and then after that, we're going to solve for sub finding the new centroid through this optimization problem. Finding the v, this v is the new centroid that minimize the sum of the squared similarities between the v and each of the points that belong to this particular clusters. And so, this is different from before as well because before in K-means this d is the l2 distance square between x^i, and v. And so, this optimization problem is convex in many scenarios. So, that's going to be a good means, and it's going to be very easy to solve, and there exist computation efficient methods. And so, we start to talk about callbacks, algorithm, and convex optimization. So, we'll make the definition formal in the future lectures as you will see.



Clustering obtained by cutting the dendrogram at a desired level: each connected component forms a cluster.

How to do it?

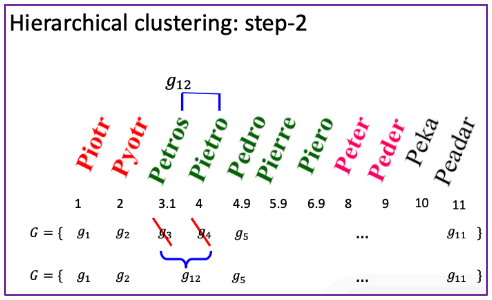
Hierarchical clustering: we have generalized the K-means to generalize K-means by replacing the l2 distance to the general definition of d, now we will talk about another clustering algorithm called **hierarchical clustering**. And so, the difference of this approach from K-means is that we will organize the data in a hierarchical fashion, and this is also called a **dendrogram** in many contexts. For example, here we have different strings, different names, and we will organize these strings by how similar they are, and so, as you can see, we will start to put together the strings that are most similar to each other as one group. These two pairs, and then we will start to group that, maybe this is another pair that's very similar and so on. And based on that, in the next level, we will combine the groups that are bacillus with each other, and so, we end up having a second level. So, the end result of this is you will have this tree structure that tells you how these strange, in this case, each string, my data point are organized in a tree structure. And the patient is, we can perform our clustering because it's very flexible, you could decide to separate the data points at any level of the tree. For example, you say, I decide to cut my tree here, and then that's going to separate your data into two clusters, and if you decide to cut your tree at this level, as you can see, this is going to form 1, 2, 3, 4, and 5 clusters. And the hierarchical clustering is going to give you more information in the sense that you will be able to understand how the data are grouped together in a pretty fine-grained level, and you can do me and zoom out by looking at different levels of the tree. And the right-hand side, I find it pretty interesting example to illustrate how the different countries in the EU in Europe can be organized in a cluster of fashion.



```
Assign each data point to its own cluster, $g_1 = \{x_1\}, g_2 = \{x_2\}, \dots, g_m = \{x_m\}$, and let $G = \{g_1, g_2, \dots, g_m\}$
Do
    – Find two clusters to merge: $i, j = argmin_{1 \le i,j \le |G|}\, D(g_i, g_j)$
    – Merge the two clusters to a new cluster: $g \leftarrow g_i \cup g_j$
    – Remove the merged clusters: $G \leftarrow G \setminus g_i, G \leftarrow G \setminus g_j$
    – Add the new cluster: $G \leftarrow G \cup \{g\}$
While $|G| > 1$
```

$$D(g_i, g_j) = \min_{x \in g_i, y \in g_j} d(x, y)$$

Bottom-up hierarchical clustering: So, here is a **description of a hierarchical clustering**, and it's also an iterative algorithm, we start with the smallest unit. And so, the smallest unit is to say each group only consists of one data points. To start with, we will have m clusters, each cluster only contains one data points. And then from there, we're going to start to form the smallest units cluster based on the choice of D as D is your similarity measure, and so, given as D, the two clusters that are closest to each other, meaning with the d, and then combine them. And so, once you find the g_i and g_j that are closest to each other, we're going to merge them into a cluster. So, these two are close to each other and then we're going to merge them. And then what is the definition of D? If you have only one data points, then that's going to be pretty easy to define the D between two clusters, that's going to be the difference. We've got a D applied on each of the data points, g_i and g_j, but if G contains more than one data points, then this D, g_i, g_j is going to be the smallest. D, this is your choice of similarity. The smallest d between x and y, for any x and y within this group. So, you keep doing this, keep finding of the cluster that are closest to and merge them, and then you will form larger and larger clusters. This gives you this hierarchical tree structure. Until you have finished grouping all the data points, you have constructed this tree.

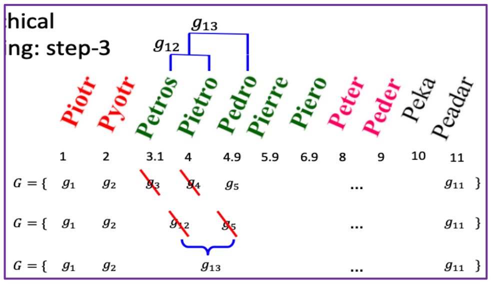


Hierarchical clustering: step-2

$$G = \{\; g_1 \quad g_2 \quad \cancel{g_3} \quad \cancel{g_4} \quad g_5 \quad \dots \quad g_{11} \;\}$$

$$G = \{\; g_1 \quad g_2 \quad \underbrace{\quad}_{g_{12}} \quad g_5 \quad \dots \quad g_{11} \;\}$$

Hierarchical clustering: step-2 So, here's my example, to show you how the clustering is performed on that string data. So, in this example, for the string purposes, we're going to assign one value to each of the strings. So, Piotr has value one, P-Y-O-T-R has value two, and then Petros have 3.1 value, and P-I-E-T-R-O have value four. And the little d between them is going to be the distance between them. And so, in the first two round, you're going examining all the cluster. Each cluster only consists of one string. And you can see which two have the smallest distance. And distance between these two is only 0.9. You can start either from here or start here. But if I have a certain way of writing my algorithm from left to the right, and then you find out these two seems to be the first two cluster, you end up having had the smallest distance. So, I decided to group these two to form g12.


hical
ng: step-3
g_{13}
g_{12}
Piotr
Pyotr
Petros
Pietro
Pedro
Pierre
Piero
Peter
Peder
Peka
Peadar
1
2
3.1
4
4.9
5.9
6.9
8
9
10
11
$G = \{ \; g_1 \; g_2 \; \cancel{g_3} \; \cancel{g_4} \; g_5 \; \ldots \; g_{11} \; \}$
$G = \{ \; g_1 \; g_2 \; \cancel{g_{12}} \; \cancel{g_5} \; \ldots \; g_{11} \; \}$
$G = \{ \; g_1 \; g_2 \; g_{13} \; \ldots \; g_{11} \; \}$

Hierarchical clustering: step-3 And in the next iteration. I will find out how do we find the smallest distance between these newly formed groups. I'm finding this one which has value of 4.9. The difference of this, to the two elements in here, the smallest value is 0.9. So, this achieved next smallest distance in your newly formed groups, so I decide to merge this group with this newly formed group here. So, keep doing this. In the end, you will end up having this result. So, illustrating how this is performed.

Summary: So, we have briefly talked about hierarchical clustering and K-means. So, what is the difference between the two? Let me write it here briefly. So, basically, we talked about K-means and the properties of the K-means algorithm. And then hierarchical clustering. And the K-means typically can work on larger dataset. And as we have explained, you can see it's pretty simple, alternating between the two steps. Versus the hierarchical clustering, typically work with a smaller dataset. Because you have to do a more bottom-up approach, the hierarchical clustering. Because you have to start with each cluster, only contain one data point and then so you can merge them. And so, if you have a very large dataset, it's going to be very expensive to perform hierarchical clustering. So, because that K-means is much more popular for large dataset. And so, that's it for today's lecture. And also, just want to mention that in the homework question you will see, we will talk about a variant of the K-means algorithm. It's called k-medoids and differences in the k-medoids. The way to assign the center, the centroids is going to be different. Instead of choosing the average as the new centroid, in the k-medoid, it's going to pick one of the data points exist in that cluster as a new centroid. And so, you will see this in your homework and you're also K-means yourself and test its performance on a real dataset in the homework. That's it for today.

Demo: Here I would like to show a demo of the K-means algorithm. We'll start with the mnist handwritten digit example. And so, first we'll load the data, which is here, and we will show you what the data look like. And then we start to run the K-means algorithm. So, let's run this code here. So, the first picture here shows the raw data. As you can see, it consists of a bunch of zeros, and bunch of ones. And this is the result after we perform K-means clustering. And these are the two sets of data points being clustered to each of the clusters. As you can see, this seems to be the cluster for the zero and it seems to be the cluster for the ones. It seems like most of the ones are being put into the correct cluster. Although there are some zeros like this because the way they're written, they're like a stroke in this direction. That's being mistaken as a one and put in this cluster. And this is the cluster for the zeros, as you can see these clusters contains mostly zero, although there are some strokes that are because of the way they're written seems to be wide and mistaken as ones. You can see the algorithm, has tried the best, but because of the data is noisy, it's not going to be perfect. And let's see. This is the first demo. Next demo, we want to show you a animation of the process, how K-means converge. And this is also a demo code to show you the step of the K-means. And in this example, the data is generated synthetically. There are six clusters. The cluster centers and so on are generated randomly as well. And so, this is the actual limitation of the K-means. As you can see ultimate between the two steps, assign data points to the current cluster and then finding out new centroids according to the current assignment. And then out between the two, until there's no change between the card center and the passenger. So, you check out the difference between the c in the two iterations. And if this is smaller than 10 to minus six, you're going to stop iteration. Let's see what's going to happen. This is animation, so you can play the code yourself, to see the process. As you can see, as it goes, the centroids are being adjusted and you will look at the points, especially at the boundaries. And the colors are changing, so they are being put into different clusters as iteration is going on. And it's going to converge after a few iterations. Let's check. Here. It's still running after 24 iterations. And so, still making it fine tunes. And finally converge in 27 iterations. That's it. And I would like to say that although here I show the MATLAB demos, but we also provide Python demos and you're welcome to try that as well. And because in this class we emphasize on the algorithm and the principles behind the algorithms, so I will only explain one programming language to show you how this is implemented. But feel free to explore the Python, we'll try to post both versions of demos and for these examples.