Clustering with the K-Means Algorithm

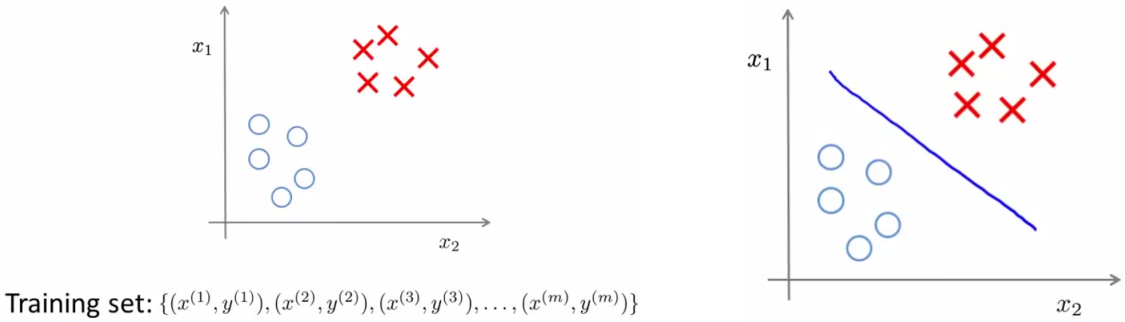
Andrew Ng (video tutorial from Coursera's "Machine Learning" class)

Transcript written*by *José Soares Augusto*, June 2012 (V1.0)

1 Unsupervised Learning_ Introduction

I'd like to start to talk about clustering, our first unsupervised learning algorithm, where we learn from unlabeled data instead of from labeled data. I briefly talked about unsupervised learning at the beginning of the class, but now it's useful to contrast it with supervised learning.

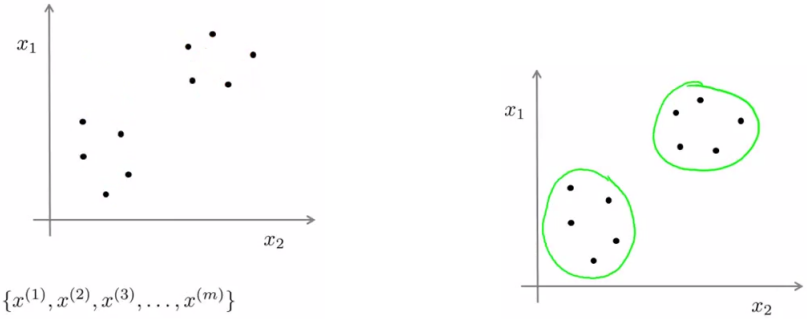
Supervised learning



Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots, (x^{(m)}, y^{(m)})\}$

In a typical supervised learning problem we are given a labeled training set, and the goal is to find the decision boundary that separates the positive and the negative label examples, and to fit a hypothesis to it (blue line).

Unsupervised learning



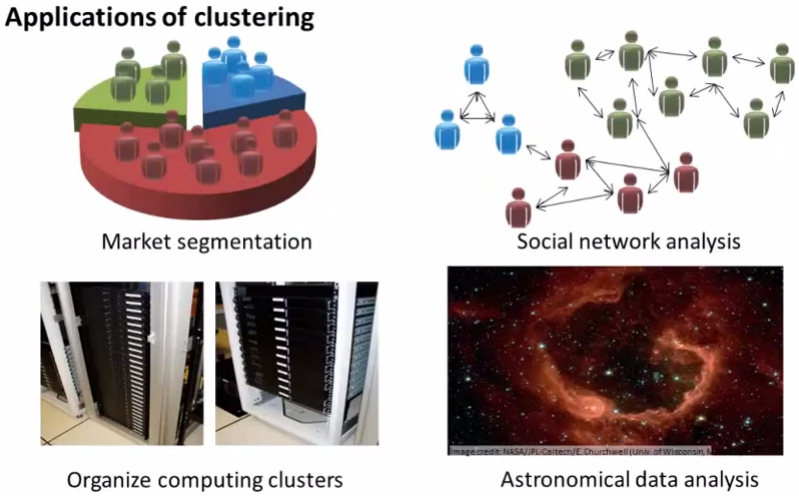
Training set: $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}\}$

In the unsupervised learning problem we're given data that does not have any labels associated with it. And so our training set is written just $x^{(1)}$, $x^{(2)}$ and so on, up to $x^{(m)}$, and we don't get any labels y. So, in unsupervised learning we give this sort of unlabeled training set to an algorithm, and we just ask the algorithm: "find some structure in the data for us."

One type of structure we might have an algorithm find is shown in the data set above, which has points grouped into two separate clusters, and so an algorithm that finds those clusters (green circles) is called a **clustering algorithm**.

This will be our first type of unsupervised learning, although there are other types of unsupervised learning algorithms that find other types of structure, or other types of patterns in the data, other than clusters.

*With help from the English subtitles which are available in the "Machine Learning" site, `https://class.coursera.org/ml/lecture/`.



So what is clustering good for? One application is market segmentation, where you may have a database of customers and you want to group them into different market segments so you can sell to them separately, or serve your different market segments better.

Other application is social network analysis, things like Facebook or Google+, or maybe information about who are the people that you email the most frequently and who are the people that email you the most frequently, and to find coherent groups of people, such as groups of friends in a social network.

One of my friends actually worked on using clustering to organize computer clusters, or data centers, better, because if you know which computers in the cluster tend to work together, you can use that to reorganize your resources, to see how you lay out the networks and how you design your data center and the communication vias.

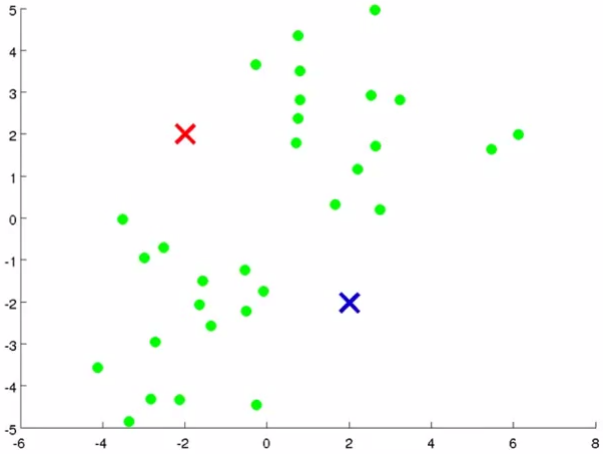
And, lastly, something that actually I worked on: using clustering algorithms on astronomical data to understand galaxy formation.

Clustering is our first example of an unsupervised learning algorithm, and we'll start to talk about a specific clustering algorithm in the next video.

2 K-Means Algorithm

In the **clustering problem** we are given an unlabeled data set and we would like to have an algorithm to automatically group the data into coherent subsets or clusters for us. The K-Means (KM) algorithm is by far the most popular and most widely used clustering algorithm.

The KM clustering algorithm is best illustrated in pictures. Let's say I want to take an unlabeled data set and I want to group the data into two clusters.



(the red and the blue crosses are the randomly initialized centroids of the clusters)

If I run the **K-Means clustering algorithm**, the first step is to randomly initialize two points, called the **cluster centroids**, the two crosses (red and blue) in the picture, and I have two of them because I want to group my data

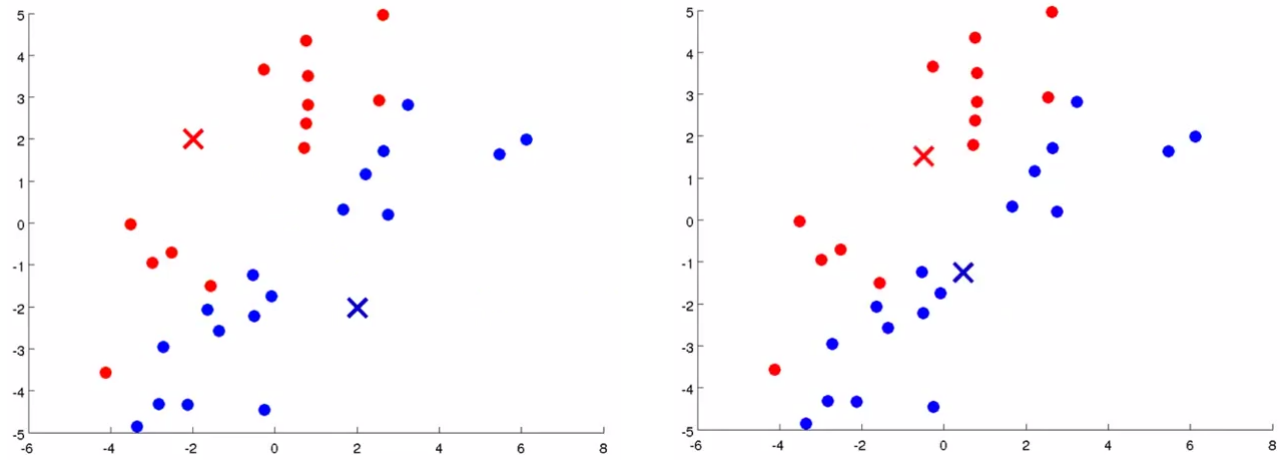


Figure 1: Left: result of the KM first step (cluster assignment step). Right: result of the KM second step (move-centroid step). The raw initial data is shown in the previous picture with the green-dotted points.

into two clusters.

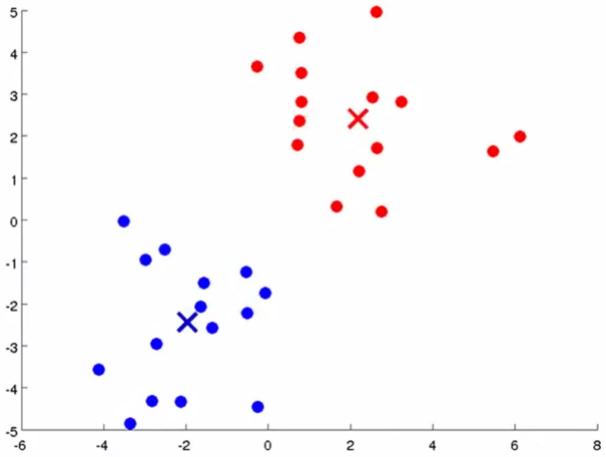
KM is an iterative algorithm, and it does two things in each iteration: the first is a *cluster assignment step*; and the second is a *move-centroid step*.

The **KM cluster assignment step** goes through each of the examples (green dots) and, depending on whether the example is closer to the red cluster centroid or to the blue cluster centroid, it is going to assign each of the data points to one of the two cluster centroids. Specifically, it goes through your data set and color each of the points either red or blue, depending on whether it is closer to the red cluster centroid or to the blue cluster centroid (left of Fig. 1).

The other part in the loop of K-Means is **the move centroid step**. We are going to take the two cluster centroids, that is, the red and the blue crosses, and we are going to move them to the average, or mean, of the points colored with the same colour (right of Fig. 1).

In Fig. 1 we can see the movement of the position of the centroids from the left graphics to the right graphics, i.e. from before to after the move centroid step.

And then we go back to another cluster assignment step, so we're again going to look at all the unlabeled examples and depending on whether it's closer to the red or to the blue cluster centroid, they're recolored either red or blue. And then we do another move centroid step, and another cluster assignment step, and so on, until we arrive at the final cluster arrangement.



Final assignment of the data points to the red and blue clusters, and final positions of the two centroids.

And in fact if you keep running additional iterations of K-Means from here, the cluster centroids will not change any further and the colours of the points will not change any further. And so, at this point K-Means has converged and it's done a pretty good job in finding the two clusters in this data.

Let's write out the K-Means algorithm more formally:

K-means algorithm

Input:
- K (number of clusters)
- Training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

$x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention)

The K-Means algorithm takes two inputs.

The first one is a parameter *K*, which is the number of clusters you want to find in the data. I'll later say how we might go about trying to choose *K*, but for now let's just say that we've decided we want a certain number of clusters and we're going to tell that to the algorithm.

And KM also takes as input an unlabeled training set $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$, and because this is unsupervised learning we don't have the labels y anymore. And for unsupervised learning with K-Means we will use the convention that $x^{(i)} \in \mathbb{R}^n$.

This is what the K-Means algorithm does:

K-means algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$
Repeat {
    for i = 1 to m
        $c^{(i)}$:= index (from 1 to K) of cluster centroid
            closest to $x^{(i)}$
    for k = 1 to K
        μ_k := average (mean) of points assigned to cluster k

}

In the first step KM randomly initializes *K* cluster centroids, $\mu_1, \mu_2, \cdots, \mu_K$, where all $\mu_K \in \mathbb{R}^n$. And so in the earlier diagrams of the example the cluster centroids corresponded to the locations of the red cross and of the blue cross. More generally, we would have eventually *K* cluster centroids rather than just 2.

Then, the inner loop of K-Means repeatedly does the following:

- first, for each of the training examples, it sets the variable $c^{(i)}$ to be the index *i*, from 1 through *K*, of the cluster centroid closest to $x^{(i)}$. This is the **cluster assignment step**, where we take each of the examples and colour it either red or blue, depending on which cluster centroid it is closest to. So $c^{(i)}$ is going to be a number from 1 to *K* that tells us if is $x^{(i)}$ is closer to the red cross or is closer to the blue cross. Another way of writing this is that for computing $c^{(i)}$ I'm going to measure the distances $||x^{(i)} - \mu_k||$, from $x^{(i)}$ to each of my cluster centroids μ_k (where the lowercase *k* indexes the different centroids). So, $c^{(i)} = \min_k ||x^{(i)} - \mu_k||^2$ and, by convention, people actually writes the minimization of the squared distance instead of $c^{(i)} = \min_k ||x^{(i)} - \mu_k||$ which, obviously, leads to the same result.
- The other step in the loop of K-Means is the **move centroid step**. And what that does is for each cluster centroid μ_k, for $k = 1, 2, \cdots, K$, it sets μ_k equal to the average of the points assigned to that cluster. So, as a concrete example, let's say that one of my cluster centroids, μ_2, has training examples $x^{(1)}$, $x^{(5)}$, $x^{(6)}$, and $x^{(10)}$ assigned to it. And this means we got $c^{(1)} = c^{(5)} = c^{(36)} = c^{(10)} = 2$ from the cluster assignment step. Then, in this move centroid step we just compute the average $\mu_2 = \frac{1}{4}[x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}]$, and

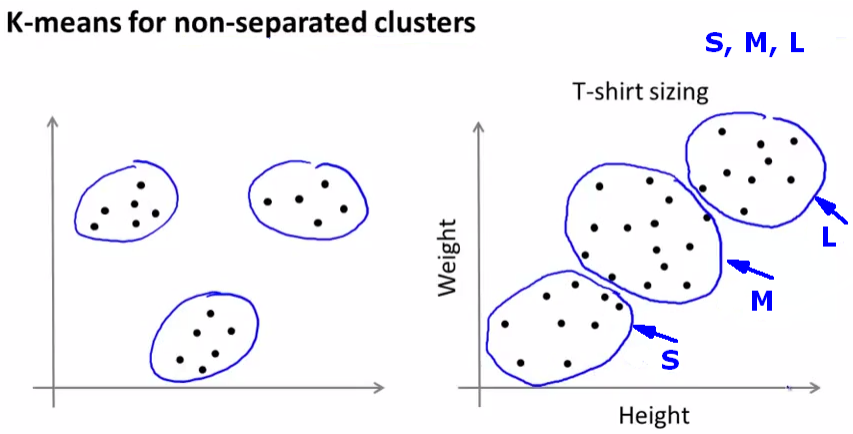


Figure 2: Left: well separated clusters; right: clusters not very well separated.

now $\mu_2 \in \mathbb{R}^n$ is going to be an n-dimensional vector because each of those examples $x^{(i)}$ are n-dimensional vectors. And so I did the move centroid step for my cluster centroid μ_2.

What if there is a cluster centroid with no points assigned to it? In that case the more common thing to do is to just eliminate that cluster centroid. And if you do that, you end up with $K-1$ clusters instead of with K clusters. Sometimes, if you really need K clusters, then the other thing you can do if you have a cluster centroid with no points assigned to it is you can just randomly re-initialize that cluster centroid. But it's more common to just eliminate a cluster if somewhere during K-Means it has no points assigned to its centroid[1]; and that can happen, although in practice it happens not that often.

Other common application of K-Means it's to the problems with non well-separated clusters. Here's what I mean. So far we've been picturing KM and applying it to data sets like that shown in the left of Fig. 2, where we have three pretty well separated clusters and we'd like an algorithm to find the 3 clusters for us.

But it turns out that very often K-Means is also applied to data sets that look like the right of Fig. 2 where there are not several very well separated clusters, and we want to do t-shirt sizing. Let's say you are a t-shirt manufacturer, you've gone to the population that you want to sell t-shirts to and you've collected a number of examples of the height and weight of these people in your population. And so I guess height and weight tend to be positively correlated, so maybe you end up with a data set like that in the right of Fig. 2.

Let's say you want to design and sell t-shirts of three sizes: small, medium and large (S, M and L). So how big should I make my small one? How big should I make my medium? And how big should I make my large t-shirts?

One way to do that would to be to run the K-Means clustering algorithm on the data set, and maybe what K-Means will do is deliver three clusters (blue boundaries).

So, even though the data beforehand didn't seem to have three well separated clusters, K-Means will kind of separate out the data into multiple clusters for you, and what you can do is then look at the lower left people cluster and try to design a small (S) t-shirt, and then design a medium (M) t-shirt for the central cluster and design a large (L) t-shirt for the upper-right cluster.

And this is in fact kind of an example of market segmentation where you're using KM to separate your market into three different segments. So you can design a product separately, S, M and L t-shirts, that tries to suit the needs of each of your three separate sub-populations well.

By now you should know how to implement the K-Means algorithm and get it to work for some problems. In the next few videos what I want to do is really get more deeply into the nuts and bolts of K-Means and to talk a bit about how to actually get it to work really well.

[1] Metaphysical question: if the cluster has no points, how do you calculate its centroid?

3 Optimization Objective

Most of the supervised learning algorithms we've seen – linear regression, logistic regression and so on, – have an optimization objective or cost function that the algorithm is trying to minimize. It turns out that K-Means also has an optimization objective or cost function that is trying to minimize.

I'd like to tell you what that optimization objective is. And the reason I want to do so is because this will be useful to us for two purposes. First, knowing what is the optimization objective of KM will help us to debug the learning algorithm and just make sure that KM is running correctly; and second, and perhaps even more importantly, later we'll talk about how we can use this to help K-Means find better clusters and avoid local optima.

K-means optimization objective

$c^{(i)}$ = index of cluster (1,2,...,K) to which example $x^{(i)}$ is currently assigned

μ_k = cluster centroid k ($\mu_k \in \mathbb{R}^n$)

$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned

Recall that while K-Means is running we're going to keep track of two sets of variables. First, there are the $c^{(i)}$ that keep track of the index k, or number, of the cluster to which an example $x^{(i)}$ is currently assigned. And then, the other set of variables is the $\{\mu_k\}$, where each element is the location of the cluster centroid indexed with k, and k is selected from the K total number of clusters. So $k = 1, 2, \cdots, K$.

The novel notation $\mu_{c^{(i)}}$ will denote the cluster centroid of the cluster to which the example $x^{(i)}$ has been assigned. And to explain that notation a little bit more, let's say that $x^{(i)}$ has been assigned to the cluster number 5. What that means is that $c^{(i)} = 5$. And so, $\mu_{c^{(i)}} = \mu_5$ because $c^{(i)} = 5$. This $\mu_{c^{(i)}}$ is the cluster centroid of cluster number 5, which is the cluster to which my example $x^{(i)}$ has been assigned.

With this notation, we now can write out the optimization objective of the K-Means clustering algorithm.

$$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K) = \frac{1}{m} \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2$$

$$\min_{\substack{c^{(1)},\ldots,c^{(m)},\\ \mu_1,\ldots,\mu_K}} J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K)$$

The cost function $J()$ that KM is minimizing is

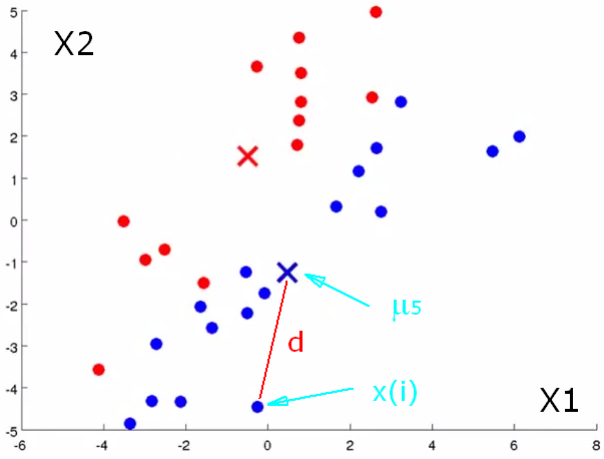
$$J\left(c^{(1)}, c^{(2)}, \cdots, c^{(m)}, \mu_1, \mu_2, \cdots, \mu_K\right) = \frac{1}{m} \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2$$

where the optimization objective is the average of the squared distance between each example $x^{(i)}$ and the location $\mu_{c^{(i)}}$ of the cluster centroid to which $x^{(i)}$ has been assigned.

The minimization is thus written as

$$\min_{c^{(1)},c^{(2)},\cdots,c^{(m)},\mu_1,\mu_2,\cdots,\mu_K} \frac{1}{m} \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2$$

So, to explain this in pictures we get the data set of the former example and assume that there would be at least 5 clusters, i.e. $K = 5$. There are two axis corresponding to the x_1 and x_2 features of the dataset.



So, $x^{(i)}$ is the lower blue example near $x_1 = 0$, and it has been assigned to the cluster centroid μ_5. Then, the corresponding squared distance is d^2, where $d = ||x^{(i)} - \mu_5||$ is the length of the red line drawn in the picture.

And K-Means is trying to find parameters $c^{(i)}$ and $\mu_{c^{(i)}}$ to minimize the cost function $J()$. This cost function is sometimes also called the **distortion cost function** or the **distortion of the K-Means algorithm**.

And just to provide a little bit more detail, here's again the K-Means algorithm:

K-means algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$

Repeat {
  for i = 1 to m
    $c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$
  for k = 1 to K
    μ_k := average (mean) of points assigned to cluster k
}

The **first step** inside the repeat loop, which is the **cluster assignment step** where we assign each point to the cluster centroid by filling $c^{(i)}$, is exactly minimizing $J()$ with respect to the variables $c^{(1)}, c^{(2)}, \cdots, c^{(m)}$ while holding the cluster centroids, $\mu_1, \mu_2, \cdots, \mu_K$ fixed (it's possible to show mathematically this claim).

So, in the cluster assignment step the centroids don't change and we are picking the values of $c^{(1)}, c^{(2)}, \cdots, c^{(m)}$ that minimize the cost function, or distortion function, $J()$.

That has a pretty intuitive meaning: we're assigning the point $x^{(i)}$ to the cluster centroid that is closest to it, because that's what minimizes the squared distance between the point and the set of cluster centroids.

And then, the **second step** of K-Means, the **move centroid step**, chooses the values of μ_k that minimize $J()$. Again I won't prove it, but it can be shown mathematically. So, it minimizes the cost function $J()$ with respect to the locations of the cluster centroids, $\mu_1, \mu_2, \cdots, \mu_K$.

So, what K-Means really is doing is it's taking the set of optimization variables and partitioning it into two halves, $c^{(1)}, c^{(2)}, \cdots, c^{(m)}$ and $\mu_1, \mu_2, \cdots, \mu_K$; first the $\{c^{(i)}\}$ set of variables, and then the $\{\mu_k\}$ set of variables. And what it does is it first minimizes $J()$ with respect to the variables $c^{(1)}, c^{(2)}, \cdots, c^{(m)}$, and then minimizes $J()$ with respect to the variables $\mu_1, \mu_2, \cdots, \mu_K$, and then it keeps on iterating these two sub-minimizations.

And now that we understand that K-Means tries to minimize the cost function $J()$, also called the distortion function, we can use this fact to try to debug our learning algorithm and make sure that our implementation of KM is running correctly.

In the next video we'll see how we can use this to help K-Means find better clusters and also help KM to avoid local optima.

4 Random Initialization

In this video, I'd like to talk about how to initialize K-Means and, more importantly, this will lead us into a discussion of how to make K-Means avoid local optima as well.

Here is the K-Means clustering algorithm that we talked about earlier:

K-means algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$

Repeat {
  for i = 1 to m
    $c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$
  for k = 1 to K
    μ_k := average (mean) of points assigned to cluster k
}

One step that we never really talked much about was how to randomly initialize the cluster centroids μ_k. There are a few different ways to randomly initialize the cluster centroids, but it turns out that there is one method that is much more recommended than most of the other options. So, let me tell you about that option since it's what often seems to work best.

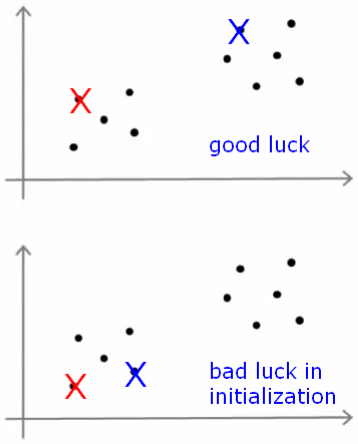
Random initialization

Should have $K < m$

Randomly pick K training examples.

Set $\mu_1, \ldots, \mu_K$ equal to these K examples.

K=2



When running K-Means you should have the number of cluster centroids, K, set to be less than the number of training examples m. It would be really weird to run K-Means otherwise. So, I usually initialize K-Means by randomly picking K training examples and set $\mu_1, \mu_2, \cdots, \mu_K$, equal to these K selections.

Let me show you a concrete example. Let's say that $K = 2$ and so I want to find two clusters. To initialize my cluster centroids I'm going to randomly pick a couple of examples, marked with red and blue crosses. In the top picture case above, I end up choosing a couple of centroids for initialization that looks like a particularly good one.

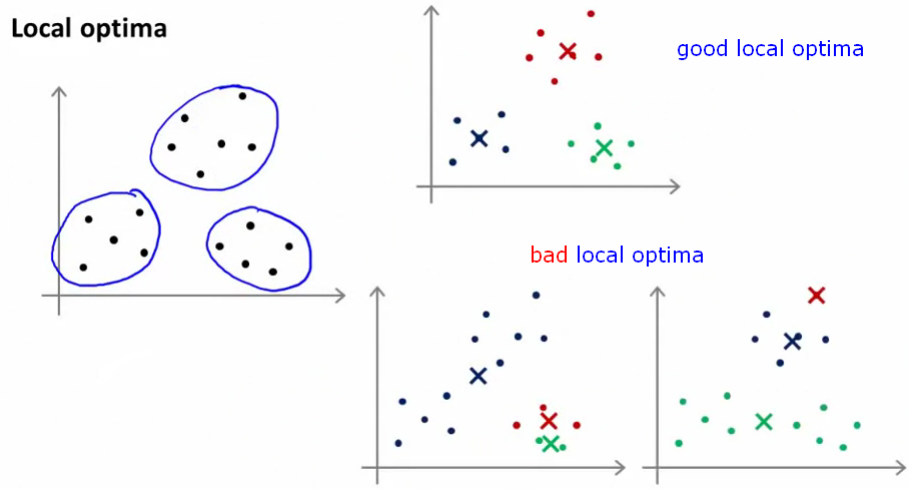
Sometimes I might get less lucky and maybe I'll end up picking the centroids as shown in the down graph (again, the centroids are the red and the blue crosses). This doesn't seem a promising choice.

And so, at initialization your first cluster centroid μ_1 will be equal to $x^{(i)}$ for some randomly value of i, and μ_2 will be equal to $x^{(j)}$ for some different randomly chosen value of j. And so on, if you have more clusters and more cluster centroids.

And, as a sort of a side comment, I should say that in the earlier video where I first illustrated KM with the animation, only for the purpose of illustration, I actually used a different method of initialization for my cluster centroids. But the method I just described above is really the recommended way, and the way that you should probably use when you implement K-Means.

So, as is suggested by these two illustrations of random initialization above, you might really guess that KM

can end up converging to different solutions depending on the random initialization. In fact, KM can end up at different solutions and, in particular, K-Means can actually end up at local optima.



If you're given a data set like this, it looks like there are three clusters and so if you run K-Means and if it ends up at a good local optimum (top case) this might be really the global optimum.

But if you had a particularly unlucky random initialization, K-Means can get stuck at different local optima which are far from the global optimum. In the lower left example, it looks like the blue cluster has captured a lot of points of the left and top, and then the red and the green clusters each has captured a relatively small number of points. And so, this corresponds to a bad local optimum.

The example on the lower right corresponds to a different local optimum of KM; in fact, the red cluster has captured only a single point.

The local optima refer to local minima of the distortion function $J\left(c^{(1)}, c^{(2)}, \cdots, c^{(m)}, \mu_1, \mu_2, \cdots, \mu_K\right)$, and the solutions in the lower graphics, the local optima on the lower examples, correspond to solutions where K-Means has gotten stuck to local optima and it's not doing a very good job minimizing the distortion function $J()$.

So, if you're worried about K-Means getting stuck in local optima, if you want to increase the odds of K-Means finding the best possible clustering, what we can do is try multiple random initializations.

Random initialization

For i = 1 to 100 {

Randomly initialize K-means.
Run K-means. Get $c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K$.
Compute cost function (distortion)
$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K)$
}

So, instead of just initializing K-Means once and hoping that that works, we initialize and run KM lots of times, select the clusters correponding to the iteration where the minimum $J()$ is found, and use that to try to make sure we get as good a local or global optimum as possible.

Concretely, let's say I decide to run KM a hundred times, so I'll execute the loop above 100 times – and it's a fairly typical number of times; in practice it will be something from 50 up to may be 1000.

So, let's say we decide to run K-Means one hundred times. We would randomly initialize K-Means and for each of these 100 random intializations we would run K-Means and that would give us a set of clusters, and a set of cluster centroids, and then we would compute the distortion $J()$, that is compute the cost function on the set of cluster assignments $c^{(1)}, c^{(2)}, \cdots, c^{(m)}$ and cluster centroids $\mu_1, \mu_2, \cdots, \mu_K$ that we've got.

Having done this whole procedure a hundred times, we will have a hundred different ways of clustering the data and then, finally, what you do is select, from these hundred ways you have found of clustering the data, the one that gives the lowest cost, or lowest distortion, $J()$.

Pick clustering that gave lowest cost $J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K)$

And it turns out that if you are running K-Means with a fairly small number of clusters, if the number of clusters is anywhere from 2 up to maybe 10, then doing multiple random initializations can often make sure that you find the better local optimum (even the global optimum), i.e. you find the better clustering of your data. But if K is very large, if $K \gg 10$, if you are trying to find hundreds of clusters, then having multiple random initializations is less likely to make a huge difference and there is a much higher chance that your first random initialization will give you a pretty decent solution already, and doing multiple random initializations will probably give you a slightly better solution, but maybe not that much.

But it's really in the regime of where you have a relatively small number of clusters, especially if you have 2 or 3 or 4 clusters, that random initialization could make a huge difference in terms of making sure you do a good job minimizing the distortion function and giving you a good clustering.

So, that was K-Means with random initialization. If you're trying to learn a clustering with a relatively small number of clusters, 2, 3, 4, 5, maybe, 6, 7, using multiple random initializations can sometimes help you find much better clustering of the data. But even if you are learning a large number of clusters, the random initialization method that I described here should give K-Means a reasonable starting point for finding a good set of clusters.

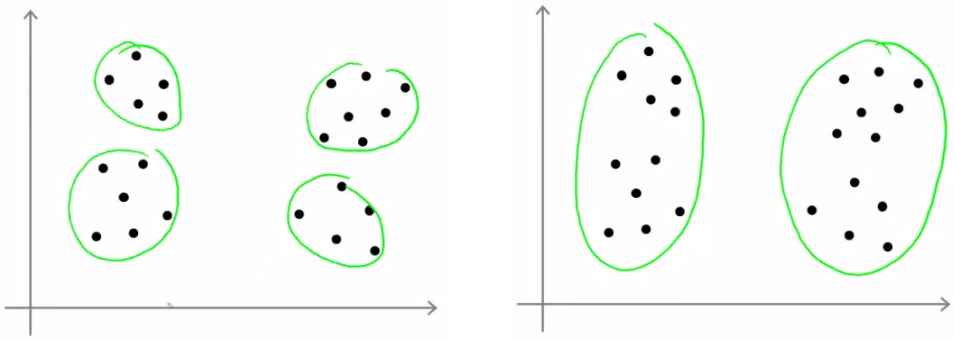
5 Choosing the Number of Clusters

In this video I'd like to talk about one last detail of K-Means clustering, which is how to choose the number of clusters, K. To be honest, there actually isn't a great way of answering this or doing this automatically, and by far the most common way of choosing the number of clusters K is still choosing it manually by looking at visualizations or by looking at the output of the clustering algorithm, or something else.

But I do get asked this question quite a lot, of how do you choose the number of clusters, and so I just want to let you know what are people currently thinking on it, although the most common thing is actually to choose the number of clusters by hand.

A large part of why it might not always be easy to choose the number of clusters, is that it is often generally ambiguous how many clusters there are in the data.

What is the right value of K?



Looking at this data set, some of you may see four clusters and that would suggest using $K = 4$, others may see two clusters and that will suggest $K = 2$, and still others may see three clusters. And so, looking at a data set like this, the true number of clusters actually seems genuinely ambiguous to me, and I don't think there is one right answer.

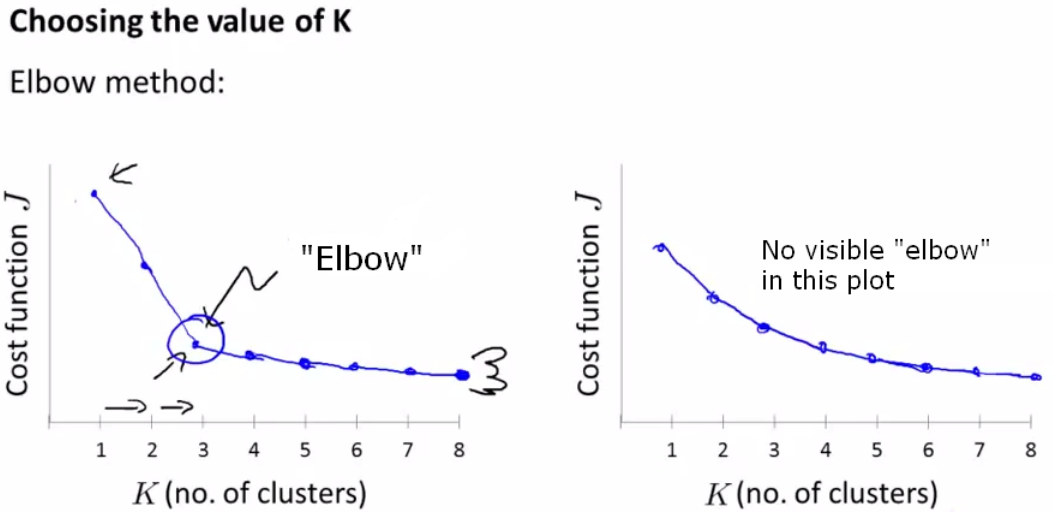
And this uncertainty is a part of supervised learning. We are aren't given labels in the data, and so there isn't always a clear cut answer. And this is one of the things that makes it more difficult to have an automatic algorithm for choosing how many clusters to have.

When people talk about ways of choosing the number of clusters, one method that people sometimes talk about is something called the **Elbow Method**. Let me just tell you a little bit about that, and then mention some of its advantages but also some of its shortcomings.

So, in the Elbow Method we vary K, which is the total number of clusters, and run K-Means. We run KM with one cluster, that means everything gets grouped into a single cluster and we compute the cost function, or

distortion, $J()$, and plot that in the graphics; and then we run KM with two clusters, maybe with multiple random initial centroids, maybe not, and with two clusters we should get, hopefully, a smaller distortion, and so plot that also. And then we run K-Means with three clusters, and hopefully we get even smaller distortion $J()$. And run K-Means with four, five, and so on, clusters.

And so we end up with a curve showing how the distortion $J()$ goes down as we increase the number of clusters (left plot). And so we get a curve with values of $\min J()|_K$, and **what the Elbow Method does** is it says "Well, let's look at this plot. Looks like there's a clear elbow there". Right? This is by analogy to the human arm with an elbow visible.



Then in the left plot you find this sort of pattern where the distortion $J()$ goes down rapidly with K increasing from 1 to 2, and from 2 to 3, and then you reach an elbow at K=3, and then the distortion goes down very slowly after that. And then it looks like maybe using three clusters is the right number of clusters, because that's the elbow of this curve, right? Distortion goes down rapidly until K=3, and really goes down very slowly after that. So let's pick K=3.

If you apply the Elbow Method, and if you get a plot that actually looks like the left one, then that's pretty good, and this would be a reasonable way of choosing the number of clusters.

It turns out the Elbow Method isn't used that often, and one reason is that if you actually use this on a clustering problem, it turns out that fairly often you end up with a curve that looks much more ambiguous (rightmost plot). And if you look at it, maybe there's no clear elbow, but it looks like distortion continuously goes down, maybe $K = 3$ is a good number, maybe $K = 4$ is a good option, maybe $K = 5$ is also not bad.

And so, if you actually do this in a practical problem and your plot looks like the one on the left that's great: it gives you a clear answer. But just as often you end up with a plot that looks like the one on the right, and is not clear where the location of the elbow really is. It makes it harder to choose a number of clusters K using this method. So, maybe **the quick summary of the Elbow Method** is that it is worth the shot, but I wouldn't necessarily have a very high expectation of it working well for any particular problem.

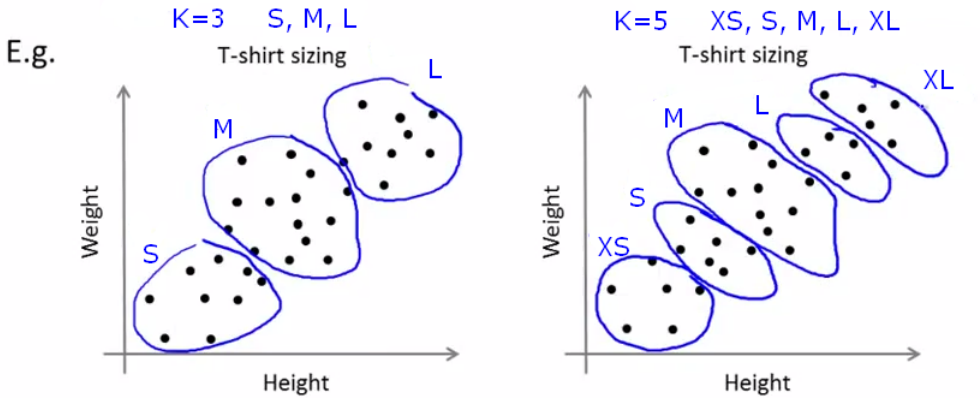
Finally, here's another way of thinking about how you choose the value of K. Very often people are running KM in order you get clusters for some later purpose, or for some sort of downstream purpose. Maybe you want to use K-Means in order to do market segmentation, like in the t-shirt sizing example; maybe you want K-Means to organize a computer cluster better, or maybe for some other different purpose. And so if that downstream purpose, such as market segmentation, gives you an evaluation metric, then often a better way to determine the number of clusters is to see how well different K values serve that later downstream purpose.

Let me step through a specific example. Let me go through the t-shirt size example again, and I'm trying to decide: do I want three t-shirt sizes? So, I choose $K = 3$, then I might have small (S), medium (M) and large (L) t-shirts. Or, maybe, I want to choose $K = 5$, and then I might have extra small (XS), small (S), medium (M), large (L) and extra large (XL) t-shirt sizes. So, you can have like three t-shirt sizes, or four, or five t-shirt sizes. We could also have four t-shirt sizes, but I'm just showing three and five in the picture, just to simplify for now.

So, if I run K-Means with K=3 maybe I end up with that left clustering plot below, with my small (S), my medium (M) and my large (L) t-shirts, whereas if I run K-Means with 5 clusters, maybe I end up with my extra small (XS), my small (S), my medium (M), my large (L) and my extra large (XL) t-shirts in the rightmost plot.

Choosing the value of K

Sometimes, you're running K-means to get clusters to use for some later/downstream purpose. Evaluate K-means based on a metric for how well it performs for that later purpose.



And the nice thing about this example is that this maybe gives us another way to choose whether we want 3, or 4, or 5 clusters and, in particular, what you can do is to think about this from the perspective of the t-shirt business and ask: "Well, if I have five segments then how well will my t-shirts fit my customers, and how many t-shirts can I sell? How happy will my customers be?" And this really makes sense from the perspective of the t-shirt business, in terms of whether I want to have more t-shirt sizes so that my t-shirts fit my customers better, or do I want to have fewer t-shirt sizes, so that I make fewer sizes of t-shirt and I can sell them to the customers more cheaply.

And so, the t-shirt selling business might give you a way to decide between three clusters versus five clusters.

So, that gives you an example of how a later downstream purpose, like the problem of deciding what t-shirts to manufacture, can give you an evaluation metric for choosing the number of clusters K.

For those of you that are doing the program exercises, if you look at this week's program exercise associative K-Means, that's an example of using KM for image compression. And so, if you were trying to choose how many clusters to use for that problem, you could use the evaluation metric of image compression to choose the number of clusters, K. So, how good do you want the image to look, versus how much do you want to compress the file size of the image?

And, you know, if you do the programming exercise what I've just said will make more sense at that time.

So, just to summarize, for the most part the number of clusters K is still chosen by hand, by human input or by human insight. One way to try to define K is to use the Elbow Method, but I wouldn't always expect that to work well. I think the better way to choose the number of clusters is to ask "for what purpose are you running K-Means?" And then to think what is the number of clusters K that serves better whatever later purpose that you actually run the K-Means for.