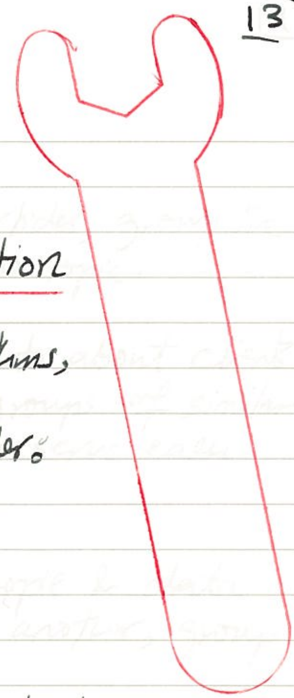


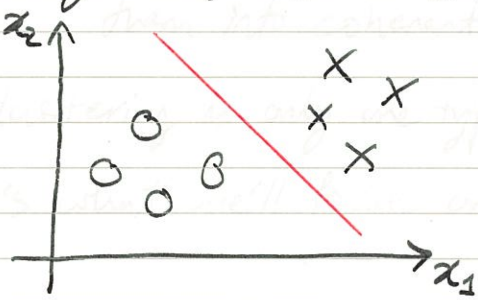
UNSUPERVISED LEARNING

Lecture 77: Unsupervised Learning: Introduction

We will now start learning about clustering algorithms, where the training set is not labeled. As a reminder:

Supervised Learning

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

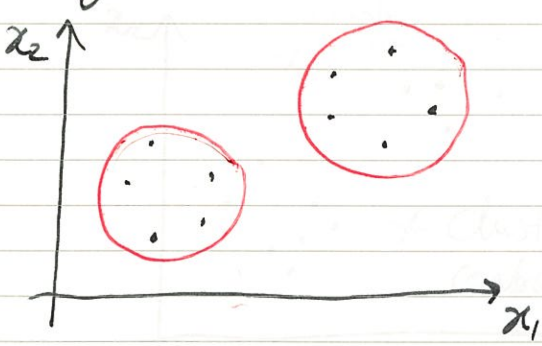


Every example comes with the "right answer", & we fit a model which can predict answers for new examples.

All algorithms we've covered so far (linear regression, logistic regression, neural networks, support vector machines) have been supervised learning algorithms.

Unsupervised learning

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



We're given unlabeled data, and asked to find some structure in the data.

For instance, a clustering algorithm may group the data shown in the figure into the two clusters shown.

Some applications of clustering:

* Given a bunch of documents, for instance news articles, group the similar ones, i.e. those that are about the same topic.

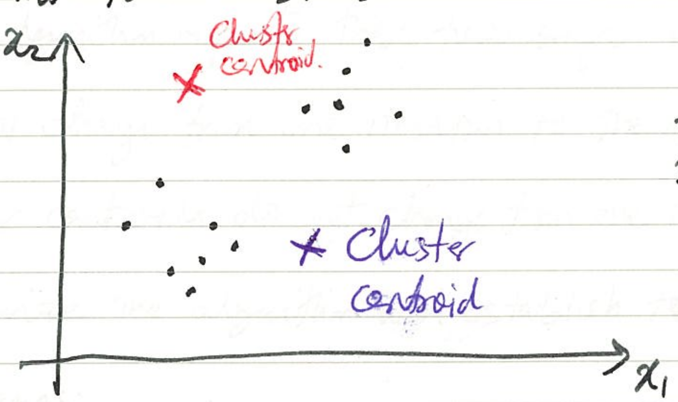
* Market segmentation: Suppose a company has data about clients and their spending habits. It could identify groups of similar clients via a clustering algorithm, to able to serve each group better.

* Social network analysis: Given a bunch of people & data about how often they send messages to one another, group them into coherent friend circles.

Clustering is only one type of unsupervised learning algorithm, but it's what we'll focus on.

Lecture 78: K-means Algorithm

The K-means algorithm is the most popular and widely used clustering algorithm. Let's get right into the algorithm. Suppose we've been given an unlabeled dataset as follows and asked to group the data into two clusters:

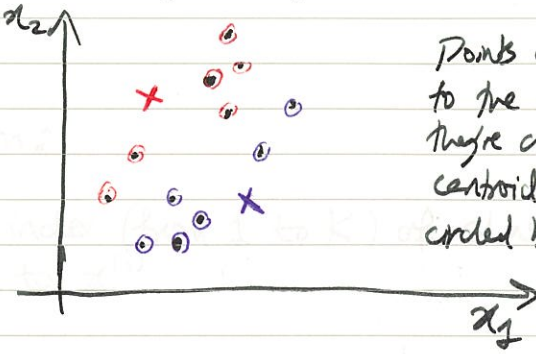


The first step is to initialize two random points, called the cluster centroids.

We initialize two because we want to create two clusters

The algorithm then proceeds iteratively to perform the following two steps:

1) Cluster assignment: Map every example to the cluster centroid it's closest to:



Points circled red are mapped to the red cluster because they're closer to the red cluster centroid. Similarly with points circled blue, because they're closer to the blue cluster centroid.

2) Move centroids: Move every centroid to a new position, determined by the mean of the examples it's mapped to: In the above example, we move the red centroid to the mean of all examples circled red, and move the blue centroid to the mean of all examples circled blue.

The algorithm repeats these two steps until the cluster assignments do not change from one iteration to the next, or equivalently, if the cluster centroids do not change from one iteration to the next. Let's summarize the algorithm and establish terminology commonly used for K-means:

K-means Algorithm

* Randomly initialize K clusters centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$.
* Repeat {

Cluster assignment → ▪ for $i = 1$ to m:

$c^{(i)} :=$ index (from 1 to K) of cluster centroid closest to $x^{(i)}$.

move centroid → ▪ for $k = 1$ to K:

$\mu_k :=$ mean of points assigned to cluster k.

}

We can write $c^{(i)}$ & μ_k as follows:

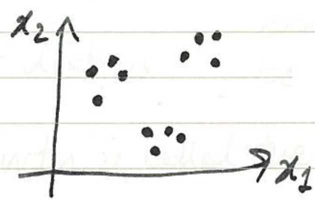
* $c^{(i)} = \underset{k \in K}{\arg\min} \|x^{(i)} - \mu_k\|^2$ → By convention the norm-squared is used.

* Let S_k denote the set of all examples assigned to cluster k:

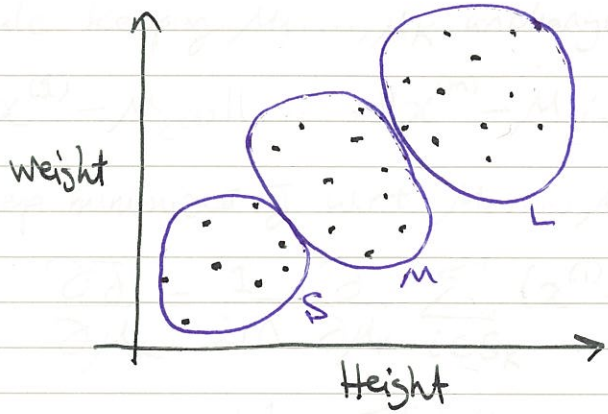
$S_k = \{ i \mid c^{(i)} = k \}$. Then: $\mu_k = \sum_{i \in S_k} x^{(i)} \times 1 / \underbrace{|S_k|}_{\text{Cardinality of } S_k}$.

What if there are no points assigned to a cluster centroid k? It's then common to just remove that cluster & deal with $K-1$ clusters instead.

So far we've only been looking at data that can be clearly separated into different clusters:



K-means clustering can also be applied in cases where obvious groups of well-separable clusters don't exist:



Suppose you're a T-shirt manufacturer & have collected weight & height of clients. You now want to create three T-shirt sizes: Small (S), Medium (M), & Large (L). You could run K-means and perhaps get a result like shown in the figure.

How would you make a prediction if presented with a new example? Andrew didn't discuss this but it would seem natural to assign the new example to a cluster whose centroid it's closest to.

Lecture 79: Optimization Objective

The K-means algorithm described in the previous lecture is a way of minimizing the following cost function:

$$\boxed{J(\underbrace{c^{(1)},\ldots,c^{(m)}}_{\in\{1,2,\ldots,K\}},\underbrace{\mu_1,\ldots,\mu_K}_{\in\mathbb{R}^n})=\frac{1}{2m}\sum_{i=1}^{m}\|x^{(i)}-\mu_{c^{(i)}}\|^2}$$

Cluster centroid of cluster to which example $x^{(i)}$ has been assigned.

Intuitively, we want to split data into K clusters such that the intra-cluster distances of examples is minimized. This cost function is called the distortion of the

K-means algorithm, or the distortion cost function. The cluster assignment step of the K-means algorithm minimizes J w.r.t $c^{(1)},\ldots,c^{(m)}$ while keeping $\mu_1,\ldots,\mu_K$ unchanged: it picks $c^{(1)},\ldots,c^{(m)}$ so that $\|x^{(1)}-\mu_{c^{(1)}}\|,\ldots,\|x^{(m)}-\mu_{c^{(m)}}\|$ is minimized. The centroid moving step minimizes J w.r.t $\mu_1,\ldots,\mu_K$ while keeping $c^{(1)},\ldots,c^{(m)}$ unchanged:

$$\frac{\partial J}{\partial \mu_K} = \frac{1}{2m}\frac{\partial}{\partial \mu_K}\sum_{i\in S_K}(x^{(i)}-\mu_K)\cdot(x^{(i)}-\mu_K)$$

$$= +\frac{1}{m}\sum_{i\in S_K}(\mu_K - x^{(i)})$$

$$= +\frac{1}{m}\left[|S_K|\,\mu_K - \sum_{i\in S_K}x^{(i)}\right]$$

Setting $\frac{\partial J}{\partial \mu_K}=0 \Rightarrow \boxed{\mu_K = \frac{1}{|S_K|}\sum_{i\in S_K}x^{(i)}}$

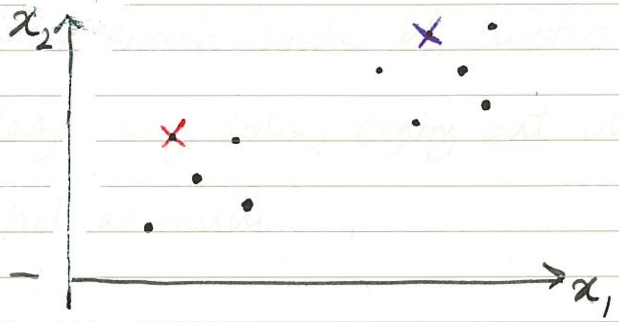
Remember that S_K is the set of all examples assigned to cluster K.

We can use J to debug the K-means algorithm, by computing its value at every iteration and making sure that it's always decreasing.

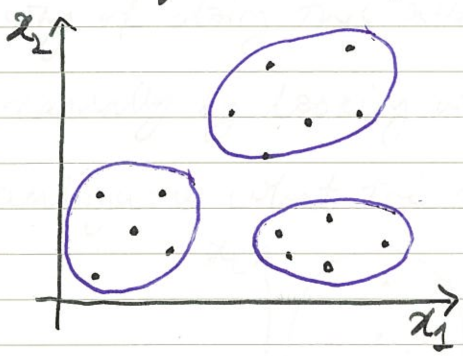
Lecture 80: Random Initialization

One aspect of K-means we didn't talk about is initialization. As we'll see shortly, this is also related to K-means getting stuck in local optima, which we'd like to avoid.

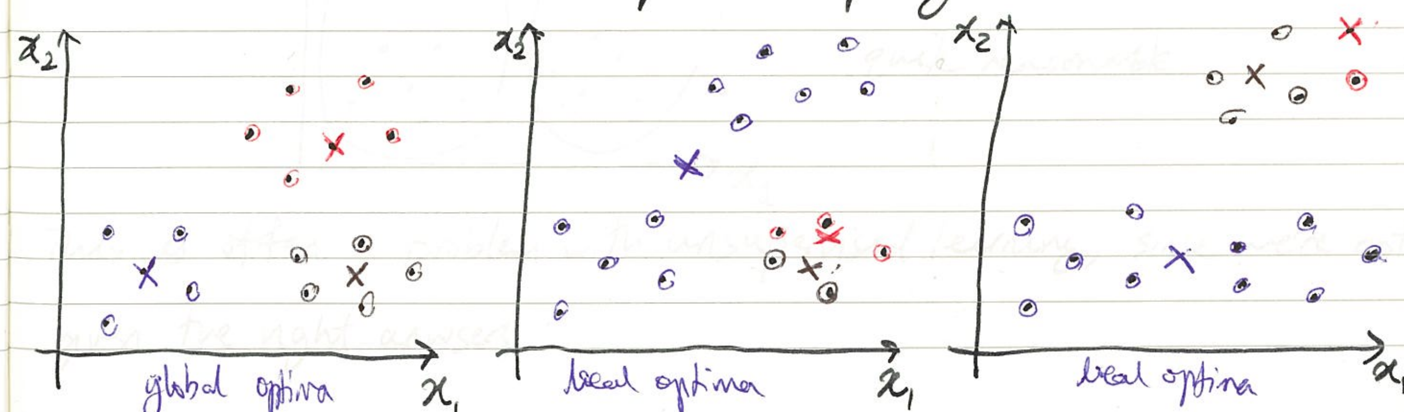
How should we initialize the cluster centroids $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$? There are various approaches, but one that seems to work quite well and is recommended: Randomly pick K training examples & set $\mu_1, \ldots, \mu_K$ to these K examples. For instance, imagine we have $K=2$ in the example below:



If we get unlucky, K-means could get stuck in a local optima (i.e not converge to the right solution) depending on the initialization:



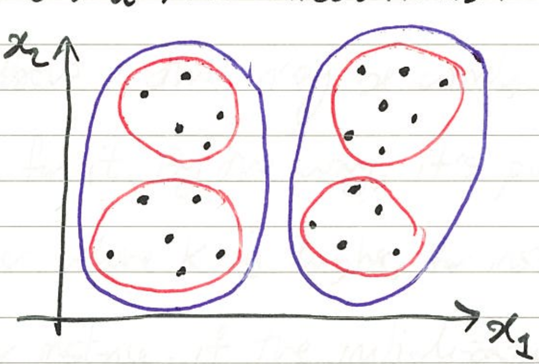
In this example, it's pretty clear that these 3 clusters are the correct solution when $K=3$. But any of the following solutions are possible depending on the initialization



To solve this problem, we can run K-means multiple times with different initializations and pick the one with the lowest value of the cost function $J(c^{(1)},\dots,c^{(m)}, \mu_1,\dots,\mu_K)$. In practice, running K-means with 50-1000 different initializations does the trick. This seems to make a big difference when $K = 2-10$, so when we're fitting for a small number of clusters. When the number of clusters is very large, say 100s, trying out different random initializations may not help as much.

Lecture 81: Choosing the Number of Clusters

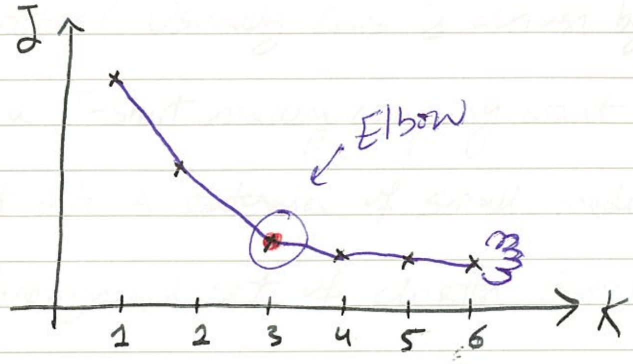
How do we choose the number of clusters? There really isn't a good way of doing this automatically and often one has to do it manually by looking at the data. Sometimes it's genuinely ambiguous what the correct number of clusters should be:



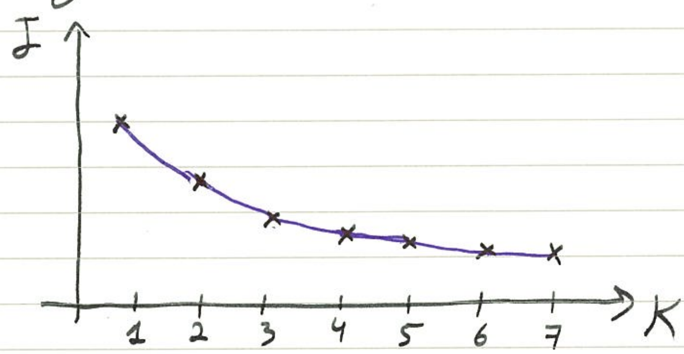
Both of these choices seem quite reasonable.

This is often a problem with unsupervised learning, since we're not given the right answers.

There's a technique called the "Elbow method" which may be worth a shot: we run K-means for different values of K & plot J as a function of K:



If we're lucky, we may get a plot like above which suggests picking where the elbow occurs as the # of clusters. Often, though, the plot doesn't have any well-defined elbow:



So the elbow method may be worth a shot, but we shouldn't have high hopes for it. By the way, it's possible to have a situation where J is higher where K is higher, for instance $J(5) > J(3)$. This can happen, for instance, if the initialization for $K=5$ lead to a bad local optima.

A better way of choosing the number of clusters is to ask: for what purpose are we running K-means? And what # of clusters would serve that purpose better? Usually this is driven by some business logic. For instance, does a T-shirt making company want to cluster their customers by weight-height into 3 categories of small, medium, large or do they want a more fine-grained set of clusters corresponding to extra small, small, medium, large, and extra large?