

# 13: Clustering

[Previous](#) [Next](#) [Index](#)

## Unsupervised learning - introduction

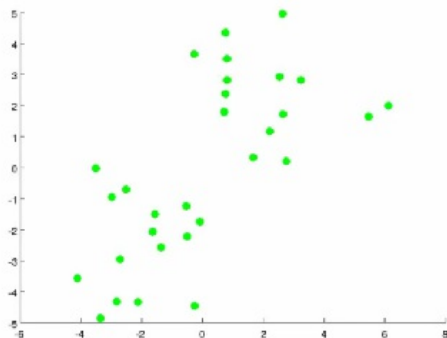
- Talk about **clustering**
  - **Learning from unlabeled data**
- Unsupervised learning
  - Useful to contrast with supervised learning
- Compare and contrast
  - Supervised learning
    - Given a set of labels, fit a hypothesis to it
  - Unsupervised learning
    - Try and determine structure in the data
    - Clustering algorithm groups data together based on data features
- What is clustering good for
  - **Market segmentation** - group customers into different market segments
  - **Social network analysis** - Facebook "smartlists"
  - **Organizing computer clusters** and data centers for network layout and location
  - **Astronomical data analysis** - Understanding galaxy formation

## K-means algorithm

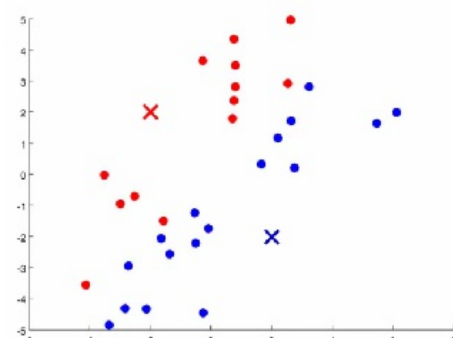
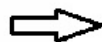
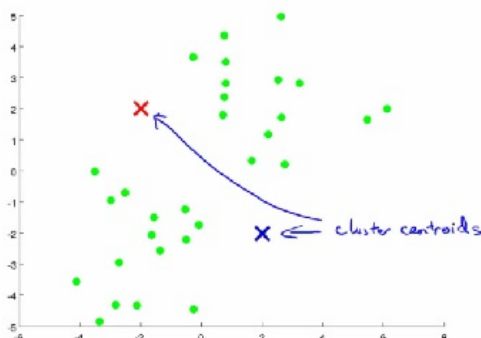
- Want an algorithm to automatically group the data into coherent clusters
- K-means is **by far** the most widely used clustering algorithm

### Overview

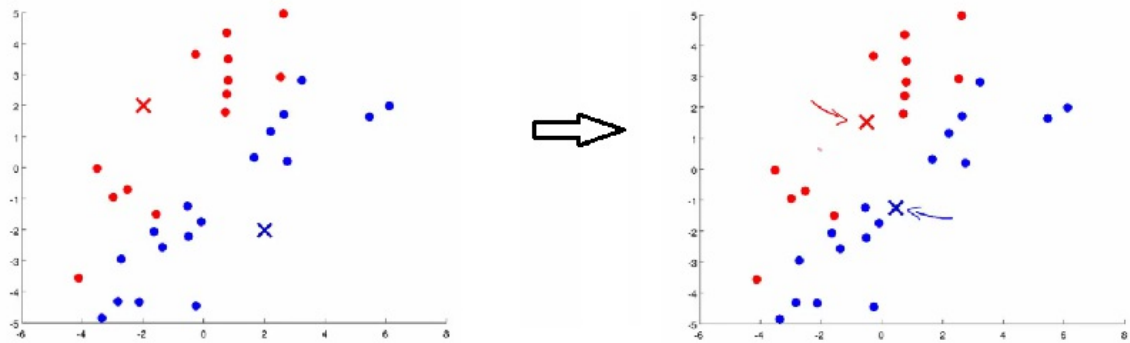
- Take unlabeled data and group into two clusters



- Algorithm overview
  - 1) Randomly allocate two points as the **cluster centroids**
    - Have as many cluster centroids as clusters you want to do ( $K$  cluster centroids, in fact)
    - In our example we just have two clusters
  - 2) Cluster assignment step
    - Go through each example and depending on if it's closer to the red or blue centroid assign each point to one of the two clusters
    - To demonstrate this, we've gone through the data and "coloured" each point red or blue



- 3) Move centroid step
  - Take each centroid and move to the average of the correspondingly assigned data-points



- Repeat 2) and 3) until convergence
- More formal definition
  - **Input:**
    - $K$  (number of clusters in the data)
    - Training set  $\{x^1, x^2, x^3, \dots, x^n\}$
  - **Algorithm:**
    - Randomly initialize  $K$  cluster centroids as  $\{\mu_1, \mu_2, \mu_3, \dots, \mu_K\}$

Repeat {

  for  $i = 1$  to  $m$

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

  for  $k = 1$  to  $K$

$\mu_k :=$  average (mean) of points assigned to cluster  $k$  }

- Loop 1

- This inner loop repeatedly sets the  $c^{(i)}$  variable to be the index of the cluster centroid closest to  $x^i$
- i.e. take  $i^{\text{th}}$  example, measure squared distance to each cluster centroid, assign  $c^{(i)}$  to the cluster closest

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

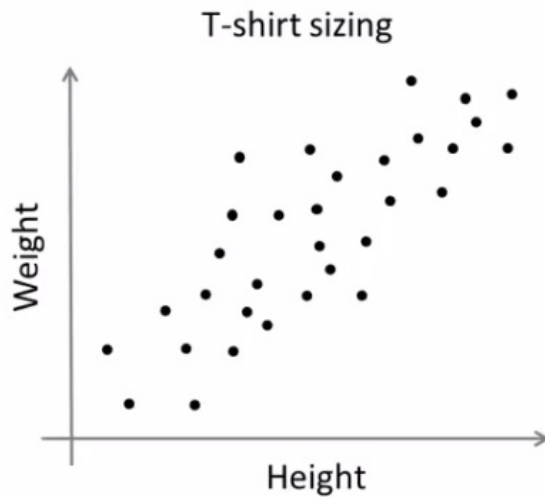
- Loop 2

- Loops over each centroid calculate the average mean based on all the points associated with each centroid from  $c^{(i)}$

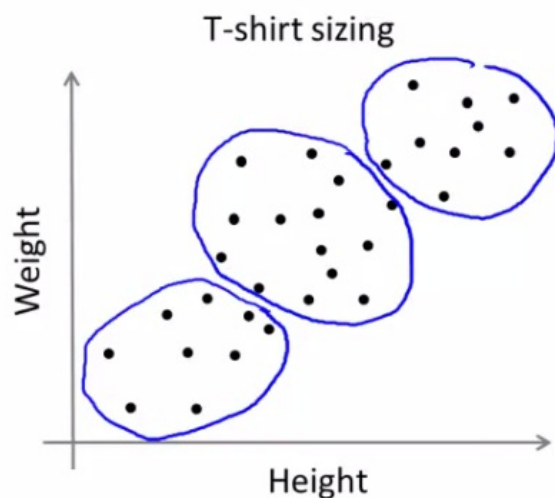
- What if there's a centroid with no data
  - Remove that centroid, so end up with  $K-1$  classes
  - Or, randomly reinitialize it
  - Not sure when though...

### K-means for non-separated clusters

- So far looking at K-means where we have well defined clusters
- But often K-means is applied to datasets where there aren't well defined clusters
  - e.g. T-shirt sizing



- Not obvious discrete groups
- Say you want to have three sizes (S,M,L) how big do you make these?
  - One way would be to run K-means on this data
  - May do the following



- So creates three clusters, even though they aren't really there
- Look at first population of people
  - Try and design a small T-shirt which fits the 1st population
  - And so on for the other two
- This is an example of market segmentation
  - Build products which suit the needs of your subpopulations

## K means optimization objective

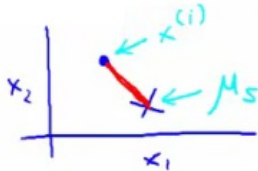
- Supervised learning algorithms have an optimization objective (cost function)
  - K-means does too
- K-means has an optimization objective like the supervised learning functions we've seen
  - Why is this good?
  - Knowing this is useful because it helps for debugging
  - Helps find better clusters
- While K-means is running we keep track of two sets of variables
  - $c^i$  is the index of clusters  $\{1, 2, \dots, K\}$  to which  $x^i$  is currently assigned
    - i.e. there are  $m$   $c^i$  values, as each example has a  $c^i$  value, and that value is one of the clusters (i.e. can only be one of  $K$  different values)
  - $\mu_k$  is the cluster associated with centroid  $k$ 
    - Locations of cluster centroid  $k$
    - So there are  $K$
    - So these the centroids which exist in the training data space
  - $\mu_{c^i}^i$  is the cluster centroid of the cluster to which example  $x^i$  has been assigned to
    - This is more for convenience than anything else
      - You could look up that example  $i$  is indexed to cluster  $j$  (using the  $c$  vector), where  $j$  is between 1 and  $K$
      - Then look up the value associated with cluster  $j$  in the  $\mu$  vector (i.e. what are the features associated with  $\mu_j$ )

- But instead, for easy description, we have this variable which gets exactly the same value
- Lets say  $x^i$  as been assigned to cluster 5
  - Means that
    - $c^i = 5$
    - $\mu_{c^i} = \mu_5$

- Using this notation we can write the optimization objective;

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

- i.e. squared distances between training example  $x^i$  and the cluster centroid to which  $x^i$  has been assigned to
  - This is just what we've been doing, as the visual description below shows;



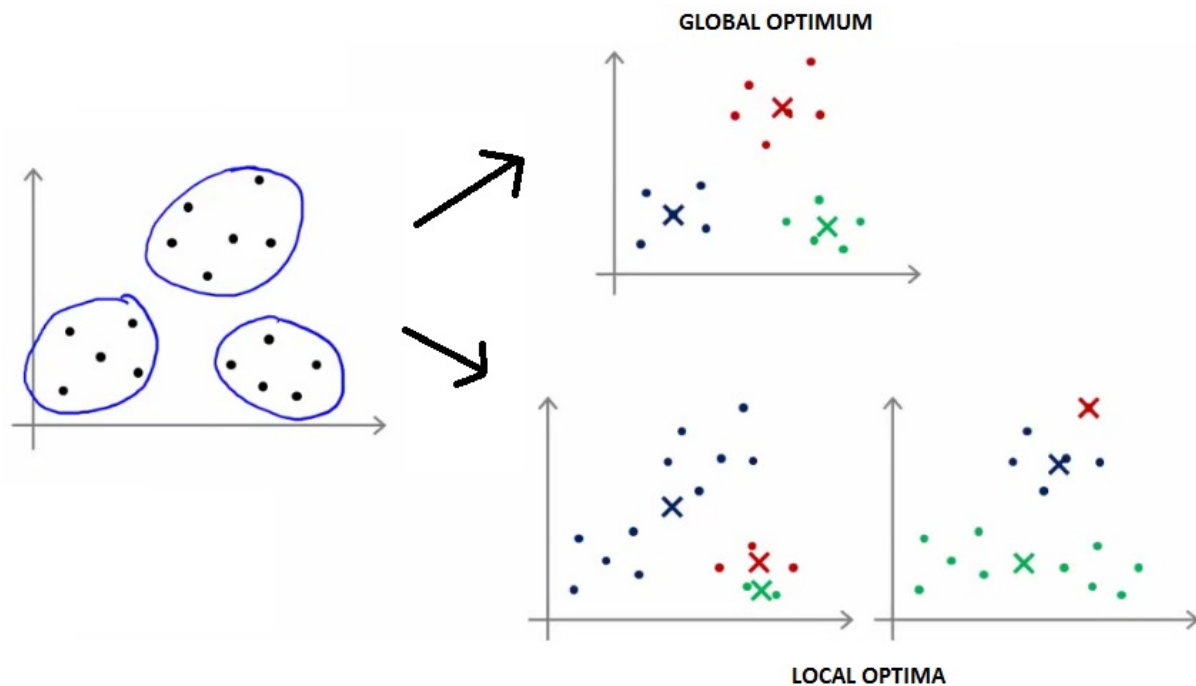
- The red line here shows the distances between the example  $x^i$  and the cluster to which that example has been assigned
  - Means that when the example is very close to the cluster, this value is small
  - When the cluster is very far away from the example, the value is large
- This is sometimes called the **distortion** (or **distortion cost function**)
- So we are finding the values which minimizes this function;

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

- If we consider the k-means algorithm
  - The **cluster assigned step** is minimizing  $J(\dots)$  with respect to  $c^1, c^2 \dots c^i$ 
    - i.e. find the centroid closest to each example
    - Doesn't change the centroids themselves
  - The **move centroid step**
    - We can show this step is choosing the values of  $\mu$  which minimizes  $J(\dots)$  with respect to  $\mu$
  - So, we're partitioning the algorithm into two parts
    - First part minimizes the  $c$  variables
    - Second part minimizes the  $J$  variables
- We can use this knowledge to help debug our K-means algorithm

## Random initialization

- How we initialize K-means
  - And how avoid local optimum
- Consider clustering algorithm
  - Never spoke about how we initialize the centroids
    - A few ways - one method is most recommended
- Have number of centroids set to less than number of examples ( $K < m$ ) (if  $K > m$  we have a problem)0
  - Randomly pick  $K$  training examples
  - Set  $\mu_1$  up to  $\mu_K$  to these example's values
- K means can converge to different solutions depending on the initialization setup
  - Risk of local optimum



- The local optimum are valid convergence, but local optimum not global ones
- If this is a concern
  - We can do multiple random initializations
    - See if we get the same result - many same results are likely to indicate a global optimum
- Algorithmically we can do this as follows;

For  $i = 1$  to 100 {

Randomly initialize K-means.

Run K-means. Get  $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K$

Compute cost function (distortion)

$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$  }

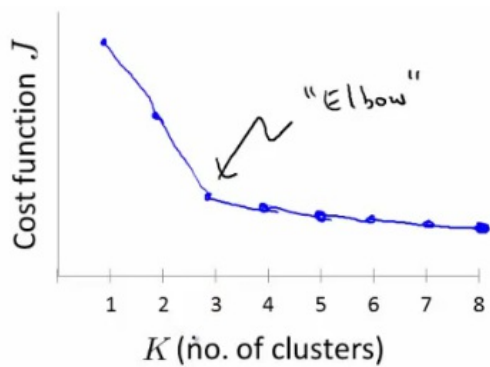
- A typical number of times to initialize K-means is 50-1000
- Randomly initialize K-means
  - For each 100 random initialization run K-means
  - Then compute the distortion on the set of cluster assignments and centroids at convergent
  - End with 100 ways of cluster the data
  - Pick the clustering which gave the lowest distortion
- If you're running K means with 2-10 clusters can help find better global optimum
  - If K is larger than 10, then multiple random initializations are less likely to be necessary
  - First solution is probably good enough (better granularity of clustering)

## How do we choose the number of clusters?

- Choosing K?
  - Not a great way to do this automatically
  - Normally use visualizations to do it manually
- What are the intuitions regarding the data?
- Why is this hard
  - Sometimes very ambiguous
    - e.g. two clusters or four clusters
    - Not necessarily a correct answer
  - This is why doing it automatic this is hard

### Elbow method

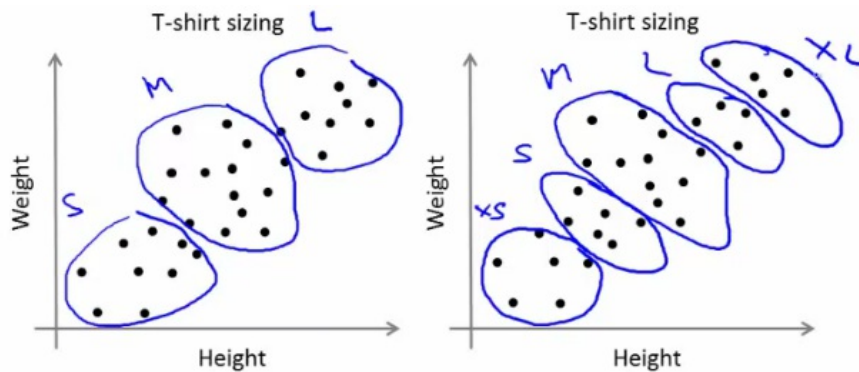
- Vary K and compute cost function at a range of K values
- As K increases  $J(\dots)$  minimum value should decrease (i.e. you decrease the granularity so centroids can better optimize)
  - Plot this (K vs  $J()$ )
- Look for the "elbow" on the graph



- Chose the "elbow" number of clusters
- If you get a nice plot this is a reasonable way of choosing K
- Risks
  - Normally you don't get a a nice line -> no clear elbow on curve
  - Not really that helpful

### Another method for choosing K

- Using K-means for market segmentation
- Running K-means for a later/downstream purpose
  - See how well different number of clusters serve you later needs
- e.g.
  - T-shirt size example
    - If you have three sizes (S,M,L)
    - Or five sizes (XS, S, M, L, XL)
    - Run K means where K = 3 and K = 5
  - How does this look



- This gives a way to chose the number of clusters
  - Could consider the cost of making extra sizes vs. how well distributed the products are
  - How important are those sizes though? (e.g. more sizes might make the customers happier)
  - So applied problem may help guide the number of clusters