Week 10

Data Science Laboratory 1

Dr John Evans
`j.evans8@herts.ac.uk`

Plan for today

The Basic Algorithm

The basic algorithm

The basic algorithm for *k*-means clustering is as follows:

1. First, we choose k initial centroids, where k is our desired number of clusters.
2. Each point is then assigned to the closest centroid, and each collection of points assigned to a centroid is a cluster.
3. The centroid of each cluster is then updated based on the points assigned to the cluster.
4. We repeat the assignment and update steps until no point changes clusters, or equivalently, until the centroids remain the same.

Formally, this is as follows...

k-means clustering

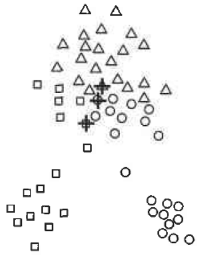
Input: k: the number of clusters,
D: a data set containing n objects

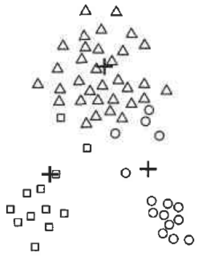
Output: A set of k clusters

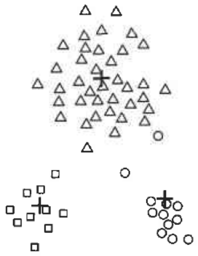
Method:

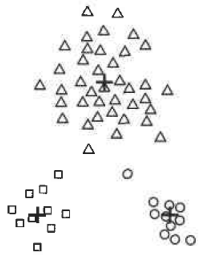
```
Step 1   Arbitrarily choose $k$ objects from $D$ as the initial cluster centres;
Step 2   repeat
Step 3       (re)assign each object to the cluster to which the object is the most similar,
                 based on the mean value of the objects in the cluster;
Step 4       update the cluster means; that is, calculate the mean value of the objects for
                 each cluster;
Step 5   until no change;
```

K-means clustering visualised


(a) Iteration 1.


(b) Iteration 2.


(c) Iteration 3.


(d) Iteration 4.

Previous slide explained

- We start with three centroids and proceed in four steps.
- Note that the centroids are indicated by '$+$', and all points belonging to the same cluster have the same marker shape.
- In the first step, points are assigned based on the initial centroids, which are all in the larger group of points. After the points are assigned to the centroid, the centroid is then updated by taking the mean.
- In the second step, the points are assigned to the updated centroids before the centroids are updated again.
- We note that in Steps 2, 3 and 4, two of the centroids move to the two small groups of points at the bottom of the figures.
- When the *k*-means algorithm terminates in Step 4 (because no changes occur), the centroids have identified the natural groupings of points.

Step 3: Assign points to closest centroid

Let’s look at this in more detail.

Step 3: Assign points to closest centroid

Let's look at this in more detail.

- To do this, we need a proximity measure that quantifies the notion of 'closest' for the specific data under consideration.
- The typical choice when working in Euclidean space is the Euclidean distance, while for documents we often use the cosine similarity. Other proximity measures are available, e.g. Manhattan distance or Jaccard similarity measure. The former would be used for Euclidean data, while the latter for documents.

Step 3: Assign points to closest centroid

Let’s look at this in more detail.

- To do this, we need a proximity measure that quantifies the notion of ‘closest’ for the specific data under consideration.
- The typical choice when working in Euclidean space is the Euclidean distance, while for documents we often use the cosine similarity. Other proximity measures are available, e.g. Manhattan distance or Jaccard similarity measure. The former would be used for Euclidean data, while the latter for documents.
- As the algorithm repeatedly calculates similarity, it is typical to use relatively simple similarity measures. However, in some cases, such as in low-dimensional Euclidean space, it is possible to avoid computing many of the similarities, thereby significantly speeding up the k-means algorithm.
- Another approach to speed up the algorithm is the bisecting k-means strategy. This speeds things up by reducing the number of similarities computed.

Step 4: Update the cluster centroids

Now let’s look at this in more detail.

Step 4: Update the cluster centroids

Now let's look at this in more detail.

- Note that the centroid can vary depending on the proximity measure for the data and the goal of the clustering. The goal of the clustering is typically expressed by an objective function which depends on the proximities of the points to one another, or to the cluster centroids.
- For instance, we may wish to minimise the squared distance of each point to its closest centroid.
- We will now demonstrate this with an example, but the key point is that once we have specified a proximity measure and an objective function, the centroid that we should choose can be determined mathematically.

Example: Data in Euclidean space

- Consider data whose proximity measure is Euclidean distance.
- For our objective function, which measures the quality of a clustering, we use the *sum of the squared error (SSE)*. Here, we calculate the error of each data point (its Euclidean distance to the closest centroid) and then compute the total sum of the squared errors.

Example: Data in Euclidean space

- Consider data whose proximity measure is Euclidean distance.
- For our objective function, which measures the quality of a clustering, we use the *sum of the squared error (SSE)*. Here, we calculate the error of each data point (its Euclidean distance to the closest centroid) and then compute the total sum of the squared errors.
- Given two different sets of clusters that are produced by two different runs of K-means, we choose the one with smallest squared error. This results in the prototypes (centroids) of this clustering being a better representation of the points in their cluster. In this way, clustering becomes an optimisation problem.

Example: Some notation

- x, an object
- C_i, the i^{th} cluster,
- c_i, the centroid of cluster C_i,
- c, the centroid of all points,
- m_i, the number of objects in the the i^{th} cluster,
- m, the number of objects in the data set,
- k, the number of clusters.

Given this notation, the *SSE* is formally defined as follows:

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} dist(c_i, x)^2,$$

where *dist* is the standard Euclidean distance between two objects in Euclidean space. We now want to minimise the total SSE[1].

[1] Some practical approaches to do this include the gradient descent method.

Example: Keeping things simple

Consider the case when the proximity function is Euclidean distance and the objective is to minimise the SSE in the case of one-dimensional data, i.e. we want to minimise,

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} (c_i - x)^2,$$

where C_i is the i^{th} cluster, x is a point in C_i and c_i is the centroid of the i^{th} cluster.

Example: Keeping things simple

Consider the case when the proximity function is Euclidean distance and the objective is to minimise the SSE in the case of one-dimensional data, i.e. we want to minimise,

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} (c_i - x)^2,$$

where C_i is the i^{th} cluster, x is a point in C_i and c_i is the centroid of the i^{th} cluster. For the j^{th} centroid c_j, to minimise we differentiate the SSE and equate to 0:

$$\begin{aligned}
\frac{\partial}{\partial c_j} SSE &= \frac{\partial}{\partial c_j} \sum_{i=1}^{k} \sum_{x \in C_i} (c_i - x)^2 \\
&= \sum_{i=1}^{k} \sum_{x \in C_i} \frac{\partial}{\partial c_j} (c_i - x)^2 \\
&= \sum_{x \in C_j} \frac{\partial}{\partial c_j} (c_j - x)^2 \\
&= \sum_{x \in C_j} 2(c_j - x) \qquad = 0.
\end{aligned}$$

Example continued

- We want to solve this for c_j and recall that we said m_j is the number of objects in the i^{th} cluster (i.e. we are summing m_j times).
- Rearranging therefore gives,

$$m_j c_j = \sum_{x \in C_k} x \quad \Rightarrow \quad c_j = \frac{1}{m_k} \sum_{x \in C_j} x.$$

- In other words, the best centroid for minimising the SSE of a cluster is the mean of the points in the cluster.

Example: Generalised

More generally, using our notation above, the centroid of the i^{th} cluster is given by,

$$c_i = \frac{1}{m_i} \sum_{x \in C_i} x.$$

Example: Generalised

More generally, using our notation above, the centroid of the i^{th} cluster is given by,

$$c_i = \frac{1}{m_i} \sum_{x \in C_i} x.$$

To illustrate this, suppose our cluster has three points:

$$\begin{pmatrix} 1 \\ -1 \end{pmatrix}, \begin{pmatrix} 2 \\ -3 \end{pmatrix}, \begin{pmatrix} 3 \\ -5 \end{pmatrix}.$$

Then,

$$c_i = \frac{1}{3}\left(\begin{pmatrix} 1 \\ -1 \end{pmatrix} + \begin{pmatrix} 2 \\ -3 \end{pmatrix} + \begin{pmatrix} 3 \\ -5 \end{pmatrix}\right) = \frac{1}{3}\begin{pmatrix} 1+2+3 \\ (-1)+(-3)+(-5) \end{pmatrix} = \begin{pmatrix} 2 \\ -3 \end{pmatrix}$$

What have we learned?

- Steps 3 and 4 of the *k*-means algorithm directly attempt to minimise the objective function (in our case, the SSE).
- Step 3 forms clusters by assigning points to their nearest centroid, which minimises the SSE for the given set of centroids.
- Finally, Step 4 recomputes the centroids so as to further minimise the SSE.
- However, the actions of *k*-means in Steps 3 and 4 are guaranteed to only find a local minimum with respect to the SSE because they are based on optimising the SSE for specific choices of the centroids and clusters, rather than for all possible choices.

Choices for proximity function

Common choices for proximity, centroids and objective functions		
Proximity Function	Centroid	Objective Function
Manhattan (L_1)	Median	Minimise sum of the $L-1$ distance of an object to its cluster centroid
Squared Euclidean (L_2^2)	Mean	Minimise sum of the squared L_2 distance of an object to its cluster centroid
Cosine Similarity	Mean	Maximise sum of the cosine similarity of an object to its cluster centroid
Bregman Divergence	Mean	Minimise sum of the Bregman divergence of an object to its cluster centroid

Step 1: Random initialisation of centroids

Let’s look at this in more detail.

Step 1: Random initialisation of centroids

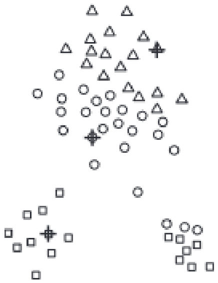
Let's look at this in more detail.

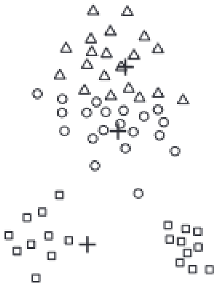
- When random initialisation of centroids is used, different runs of *k*-means typically produce different total SSEs.
- As such, the *k*-means algorithm is not guaranteed to converge to the global optimum, and often terminates at a local optimum.
- Such results inevitably depend on the initial random selection and so it follows that choosing the proper initial centroids is the key step of the basic *k*-means procedure. While a common approach is to choose the initial centroids randomly, the resulting clusters are often poor.

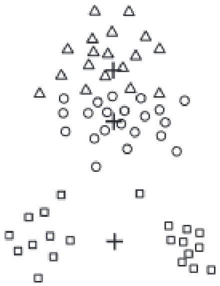
Example: Poor initial centroids

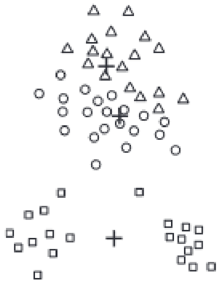
- We use the same data set seen in the earlier visualisation of the K-means algorithm. In that case, we had a convergence to the local optimum.
- Now consider the figure on the next slide.

Example: New figure

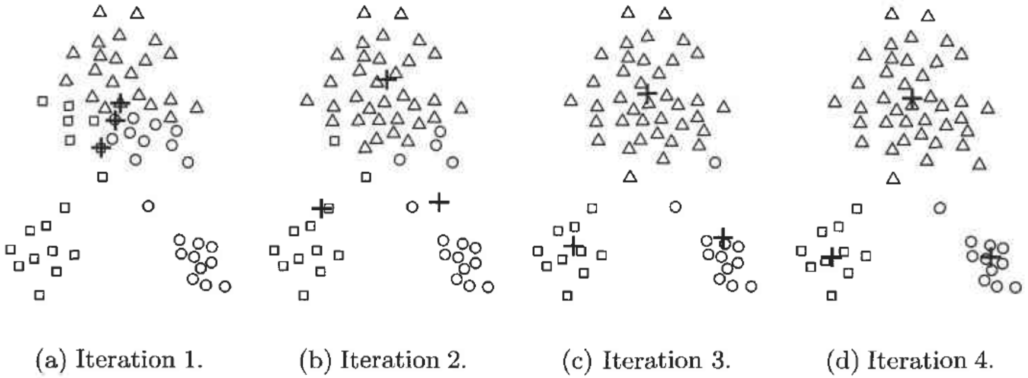

(a) Iteration 1.


(b) Iteration 2.


(c) Iteration 3.


(d) Iteration 4.

Example: Old figure for comparison


(a) Iteration 1. (b) Iteration 2. (c) Iteration 3. (d) Iteration 4.

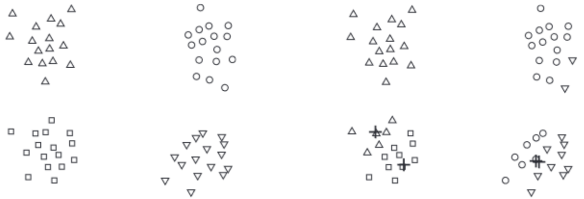
Example: What is going on?

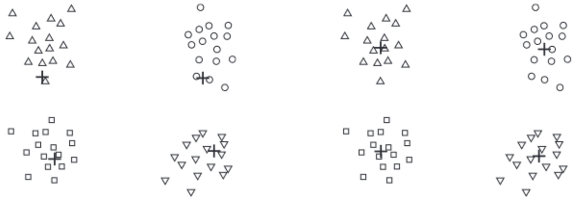
- Both figures show clusters that result from two particular choices of initial centroids.
- In the original figure, even though all the initial centroids are from one natural cluster, the minimum SSE clustering is still found.
- In the new figure however, even though the initial centroids appear better distributed, we obtain a suboptimal clustering with higher squared error.

Example: What is going on?

- Both figures show clusters that result from two particular choices of initial centroids.
- In the original figure, even though all the initial centroids are from one natural cluster, the minimum SSE clustering is still found.
- In the new figure however, even though the initial centroids appear better distributed, we obtain a suboptimal clustering with higher squared error.
- One approach to overcoming this is to perform multiple runs of the algorithm, each time with a different set of randomly chosen initial centroids.
- Then, we select the set of clusters with the minimum SSE.
- While simple, this strategy can struggle depending on the data set and/or the number of clusters sought.

Example: Another problem

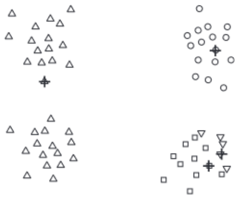

(a) Initial points. (b) Iteration 1.

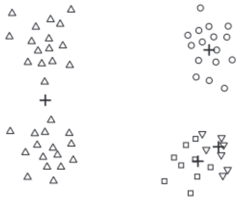

(c) Iteration 2. (d) Iteration 3.

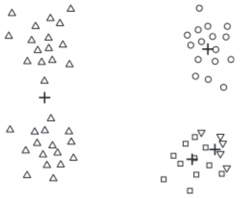
Example: So far so good...

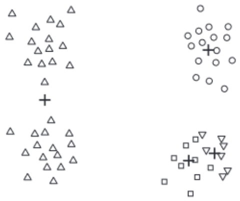
- In this case, we have two pairs of clusters. The clusters in each (top-bottom) pair are closer to each other than to the clusters in the other pair.
- (b)-(d) shows that if we start with two initial centroids per pair of clusters, then even when both centroids are in a single cluster, the centroids will redistribute themselves so that the ‘true’ clusters are found.
- However, if a pair of clusters has only one initial centroid and the other pair has three, then two of the true clusters will be combined and one true cluster will be split.

Example: The problem manifest


(a) Iteration 1.


(b) Iteration 2.


(c) Iteration 3.


(d) Iteration 4.

How do we overcome all this?

1. Take a sample of points and cluster them using a hierarchical clustering technique. Next, *k* clusters are extracted from the hierarchical clustering and the centroids of those clusters are used as the initial centroids. This approach often works well but is practical only if
   - 1.1 the sample is relatively small, e.g. a few hundred/thousand (hierarchical clustering is expensive), and
   - 1.2 *k* is relatively small compared to the sample size.

How do we overcome all this? Another option

2. Select the first point at random or take the centroid of all points.
    - Then, for each successive initial centroid, select the point that is farthest from any of the initial centroids already selected. In this way, we obtain a set of initial centroids that is guaranteed to be not only randomly selected, but also well separated.

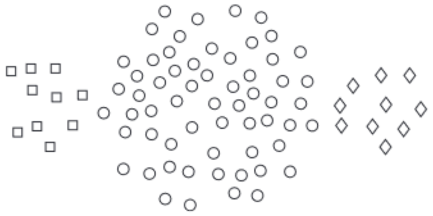
How do we overcome all this? Another option

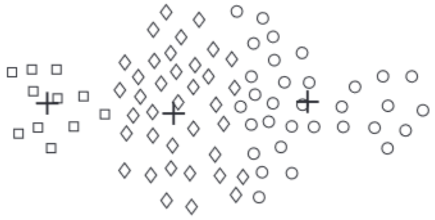
2. Select the first point at random or take the centroid of all points.
    - Then, for each successive initial centroid, select the point that is farthest from any of the initial centroids already selected. In this way, we obtain a set of initial centroids that is guaranteed to be not only randomly selected, but also well separated.
    - Such an approach can select outliers, rather than points in dense regions (clusters). This can lead to a situation where many clusters have just one point (an outlier) which reduces the number of centroids for forming clusters for the majority of points. As might be expected, this is also expensive.
    - To overcome these problems, this approach is often applied to a sample of the points. Because outliers are comparatively rare, they tend to not show up in a random sample. In contrast, points from every dense region are likely to be included unless the sample size is very small.
    - Furthermore, the computation involved in finding the initial centroids is greatly reduced because the sample size is typically much smaller than the number of points.

Yet another problem!

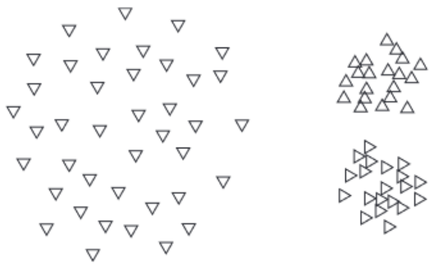
- *k*-means has trouble detecting 'natural' clusters when these are non-spherical, or when clusters have widely different sizes or densities.
- This is visualised in the next few slides.
- In the first, *k*-means cannot find the three natural clusters because one of the clusters (the central one) is much larger than the other two. Thus, the larger cluster is broken, while one of the smaller clusters is combined with a portion of the larger cluster.
- In the second, *k*-means fails because the two smaller clusters are much denser than the larger cluster.
- Finally, in the third, *k*-means finds two clusters that mix portions of the two natural clusters because it is trying to find spherical clusters, but the natural clusters have a spiral shape.

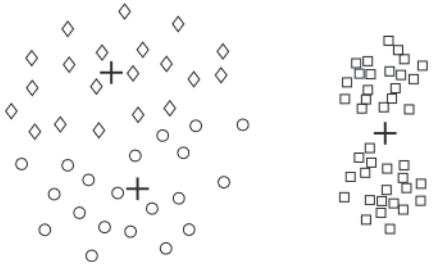
Failure due to clusters having very different sizes


(a) Original points.

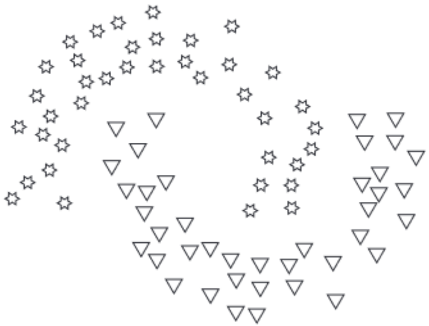

(b) Three K-means clusters.

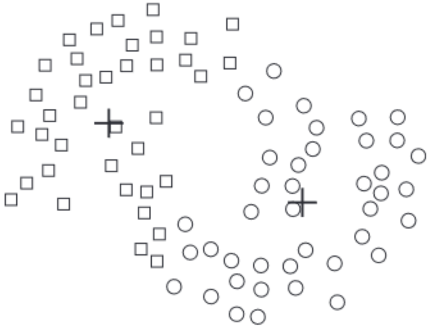
Failure due to clusters having very different densities


(a) Original points.


(b) Three K-means clusters.

Failure due to clusters being non-spherical


(a) Original points.


(b) Two K-means clusters.

Why do we have these problems?

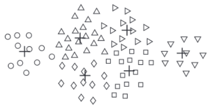
Answer: Because the *k*-means objective function is a mismatch for the kinds of clusters we are trying to find.

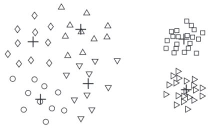
Why do we have these problems?

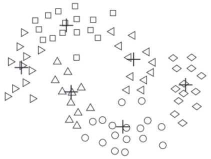
Answer: Because the *k*-means objective function is a mismatch for the kinds of clusters we are trying to find.

- This is because it is minimised by globular clusters of equal size and density, or by clusters that are well-separated.
- To overcome these limitations, we must be willing to accept a clustering that breaks the natural clusters into a number of subclusters. This is shown on the next slide, which demonstrates what happens to our three data sets if instead we find six clusters instead of two or three.
- Each smaller cluster is pure in the sense that it contains only points from one of the natural clusters.

Overcoming these problems


(a) Unequal sizes.


(b) Unequal densities.


(c) Non-spherical shapes.

Some strengths and weaknesses of the *k*-means algorithm

- *k*-means is simple and can be used for a wide variety of data types.
- *k*-means is quite efficient, even though multiple runs are often performed.
- Some variants (such as bisecting *k*-means) are even more efficient, and are less susceptible to initialisation problems.
- *k*-means is not suitable for all types of data, however. It cannot handle non-spherical clusters, clusters of different sizes or clusters of different densities. These limitations can often be overcome by finding pure subclusters if the number of clusters specified is large enough.
- *k*-means also has trouble clustering data that contains outliers. Outlier detection and removal can help significantly in such situations.
- *k*-means is restricted to data for which there is a notion of a centre (a centroid). A related technique known as *k*-medoid clustering does not have this restriction, but it is more expensive.