Clustering is the task of grouping together a set of objects in a way that objects in the same cluster are more similar to each other than to objects in other clusters. Similarity is a metric that reflects the strength of relationship between two data objects. Clustering is mainly used for exploratory data mining. It has manifold usage in many fields such as machine learning, pattern recognition, image analysis, information retrieval, bio-informatics, data compression, and computer graphics.

However, this post tries to unravel the inner workings of K-Means, a very popular clustering technique. The algorithm will help you to tackle unlabeled datasets (i.e. the datasets that do not have any class-labels) and draw your own inferences from them with ease.

K-Means falls under the category of centroid-based clustering. A centroid is a data point (imaginary or real) at the center of a cluster. In centroid-based clustering, clusters are represented by a central vector or a centroid. This centroid might not necessarily be a member of the dataset. Centroid-based clustering is an iterative algorithm in which the notion of similarity is derived by how close a data point is to the centroid of the cluster.

In this post, you will learn about:

* The inner workings of the K-Means algorithm
* A simple case study in Python
* The disadvantages of K-Means

**The inner workings of the K-Means clustering algorithm:**

To do this, you will need a sample dataset (training set):

|  |  |  |  |
| --- | --- | --- | --- |
| **Objects** | **X** | **Y** | **Z** |
| OB-1 | 1 | 4 | 1 |
| OB-2 | 1 | 2 | 2 |
| OB-3 | 1 | 4 | 2 |
| OB-4 | 2 | 1 | 2 |
| OB-5 | 1 | 1 | 1 |
| OB-6 | 2 | 4 | 2 |
| OB-7 | 1 | 1 | 2 |
| OB-8 | 2 | 1 | 1 |

The sample dataset contains 8 objects with their X, Y and Z coordinates. Your task is to cluster these objects into two clusters (here you define the value of K (of K-Means) in essence to be 2).

So, the algorithm works by:

* Taking any two centroids or data points (as you took 2 as K hence the number of centroids also 2) in its account initially.
* After choosing the centroids, (say C1 and C2) the data points (coordinates here) are assigned to any of the Clusters (let’s take centroids = clusters for the time being) depending upon the distance between them and the centroids.
* Assume that the algorithm chose OB-2 (1,2,2) and OB-6 (2,4,2) as centroids and cluster 1 and cluster 2 as well.
* For measuring the distances, you take the following distance measurement function (also termed as similarity measurement function):

        d=|x2–x1|+|y2–y1|+|z2–z1|d=|x2–x1|+|y2–y1|+|z2–z1|

This is also known as the **Taxicab distance** or **Manhattan distance**, where d is distance measurement between two objects, (x1,y1,z1) and (x2,y2,z2) are the X, Y and Z coordinates of any two objects taken for distance measurement.

Feel free to check out other distance measurement functions like [Euclidean Distance, Cosine Distance](https://arxiv.org/ftp/arxiv/papers/1405/1405.7471.pdf) etc.

The following table shows the calculation of distances (using the above distance measurement function) between the objects and centroids (OB-2 and OB-6): 

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Objects** | **X** | **Y** | **Z** | **Distance from C1(1,2,2)** | **Distance from C2(2,4,2)** |
| OB-1 | 1 | 4 | 1 | 3 | 2 |
| OB-2 | 1 | 2 | 2 | 0 | 3 |
| OB-3 | 1 | 4 | 2 | 2 | 1 |
| OB-4 | 2 | 1 | 2 | 2 | 3 |
| OB-5 | 1 | 1 | 1 | 2 | 5 |
| OB-6 | 2 | 4 | 2 | 3 | 0 |
| OB-7 | 1 | 1 | 2 | 1 | 4 |
| OB-8 | 2 | 1 | 1 | 3 | 4 |

The objects are clustered based on their distances between the centroids. An object which has a shorter distance between a centroid (say C1) than the other centroid (say C2) will fall into the cluster of C1. After the initial pass of clustering, the clustered objects will look something like the following:

|  |
| --- |
| **Cluster 1** |
| OB-2 |
| OB-4 |
| OB-5 |
| OB-7 |
| OB-8 |

|  |
| --- |
| **Cluster 2** |
| OB-1 |
| OB-3 |
| OB-6 |

Now the algorithm will continue updating cluster centroids (i.e the coordinates) until they cannot be updated anymore (more on when it cannot be updated later). The updation takes place in the following manner:

http://res.cloudinary.com/dyd911kmh/image/upload/f_auto,q_auto:best/v1526380253/CodeCogsEqn_v5le8c.png(where n = number of objects belonging to that particular cluster)

So, following this rule the updated cluster 1 will be ((1+2+1+1+2)/5, (2+1+1+1+1)/5,(2+2+1+2+1)/5) = (1.4,1.2,1.6). And for cluster 2 it will be ((1+1+2)/3, (4+4+4)/3, (1+2+2)/3) = (1.33, 4, 1.66).

After this, the algorithm again starts finding the distances between the data points and newly derived cluster centroids. So the new distances will be like following:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Objects** | **X** | **Y** | **Z** | **Distance from C1(1.4,1.2,1.6)** | **Distance from C2(1.33, 4, 1.66)** |
| OB-1 | 1 | 4 | 1 | 3.8 | 1 |
| OB-2 | 1 | 2 | 2 | 1.6 | 2.66 |
| OB-3 | 1 | 4 | 2 | 3.6 | 0.66 |
| OB-4 | 2 | 1 | 2 | 1.2 | 4 |
| OB-5 | 1 | 1 | 1 | 1.2 | 4 |
| OB-6 | 2 | 4 | 2 | 3.8 | 1 |
| OB-7 | 1 | 1 | 2 | 1 | 3.66 |
| OB-8 | 2 | 1 | 1 | 1.4 | 4.33 |

The new assignments of the objects with respect to the updated clusters will be: 

|  |
| --- |
| **Cluster 1** |
| OB-2 |
| OB-4 |
| OB-5 |
| OB-7 |
| OB-8 |

|  |
| --- |
| **Cluster 2** |
| OB-1 |
| OB-3 |
| OB-6 |

This is where the algorithm no longer updates the centroids. Because there is no change in the current cluster formation, it is the same as the previous formation.

Now when, you are done with the cluster formation with K-Means you may apply it to some data the algorithm has not seen before (what you call a Test set). Let's generate that: 

|  |  |  |  |
| --- | --- | --- | --- |
| **Objects** | **X** | **Y** | **Z** |
| OB-1 | 2 | 4 | 1 |
| OB-2 | 2 | 2 | 2 |
| OB-3 | 1 | 2 | 1 |
| OB-4 | 2 | 2 | 1 |

After applying K-means on the above dataset, the final clusters will be:

|  |
| --- |
| **Cluster 1** |
| OB-2 |
| OB-3 |
| OB-4 |

|  |
| --- |
| **Cluster 2** |
| OB-1 |

Any application of an algorithm is incomplete if one is not sure about its performance. Now, in order to know how well the K-Means algorithm is performing there are certain metrics to consider. Some of these metrics are:

* Adjusted rand index
* Mutual information based scoring
* Homogeneity, completeness and v-measure

Now that you have got familiar with the inner mechanics of K-Means let's see K-Means live in action.

**A simple case study of K-Means in Python:**

For the implementation part, you will be using the Titanic dataset (available [here](https://www.kaggle.com/c/titanic)). Before proceeding with it, I would like to discuss some facts about the data itself. The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

Now, talking about the dataset, the training set contains several records about the passengers of Titanic (hence the name of the dataset). It has 12 features capturing information about passenger\_class, port\_of\_Embarkation, passenger\_fare etc. The dataset's label is **survival**which denotes the survivial status of a particular passenger. Your task is to cluster the records into two i.e. the ones who survived and the ones who did not.

You might be thinking that since it is a labeled dataset, how could it be used for a clustering task? You just have to drop the 'survival' column from the dataset and make it unlabeled. It's the task of K-Means to cluster the records of the datasets if they survived or not.