**K-Means Clustering**

* K-means is an algorithm that allows you to cluster your data and is a convenient tool for discovering categories or groups in your dataset that you would have otherwise not thought of yourself.
* So, here we have got a scatterplot

Chart, scatter chart

Description automatically generated

* Let’s imagine we have two variables in our dataset, and we decided to plot those two variables on your x and y axis. And as shown above, based on the two variables, all the observations are configured.
* The question here is – can we identify certain groups among the variables, and how do we identify the groups?
* What k-means does is take the complexity out of that decision making and allows to easily identify the clusters of datasets.

Diagram

Description automatically generated

* This is a very simplified example, we only have two dimensions here, so two variables here, but k-means can work with multi-dimensional objects, so, it can work with any number of dimensions.

**Steps for K-Means Clustering –**

**Step 1 –** Choose the number K of clusters

**Step 2 –** Select at random K points, the centroids (not necessarily from your dataset – they don’t have to be the part of the observations plotted on the scatterplot, they can be random x and y values, as long as they equate to the number of clusters you have decided upon)

**Step 3 –** Assign each data point to the closest centroid and that will form the cluster (the distance measured from the data point to the centroid can depend on how you measure the distance, but for this section we will use Euclidean distance).

**Step 4 –** Compute and place the new centroid of each cluster.

**Step 5 –** Reassign data points to new closest centroid.

If Reassignment took place go to step 4, otherwise FINISH.

**Example of K-Means Clustering –**

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

* Instead of calculating distance from each point to the centroid, we will calculate the distance between the two centroids, pass a line through the center of the distance between the two centroids, and from geometry we know that any point on the green line is equidistant to the two centroids. From that, it is obvious that any of points on the scatterplot above the green line is closer to the blue, and any point below the green line is closer to red.

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated

We keep repeating the steps until none of the points need reassignments.

Chart, scatter chart

Description automatically generated

And thus, after no points needed reassignment was needed, we finished after this step.

Diagram

Description automatically generated

* After finishing all the steps and moving past the iterative process of step 4 and step 5, we remove the centroid line, and the centroid points, and have the final cluster.

**Random Initialization Trap:**

Scatter chart

Description automatically generated

* Here we have got a scatterplot with two variables, and in this case, we are going to choose 3 clusters.
* And if we initialize our randomly selected centroids correctly, and if we perform the algorithm, we get the following clusters.

Chart, scatter chart

Description automatically generated

* Now the question is, what if we choose the centroid in a different location, or a different centroid, will the result be different? What would happen if we had a bad random initialization?
* The cluster shown below is what we get when we change the centroids of the algorithm.

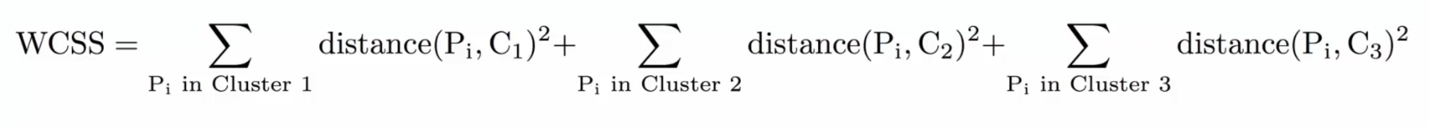
Diagram

Description automatically generated

* And we can see that the three clusters we got here is different from what we got earlier.
* Therefore, what we have now is a situation, or a phenomenon, where the selection of centroids at the start of the algorithm can potentially dictate the outcome of the algorithm and that’s not a good thing as centroids are selected at random.
* There is a correction, a modification, to the k-means algorithm that allows you to correctly select the centroids.
* The solution is k-means ++ algorithm. We won’t be looking at that in this section.
* But the good thing is, the functioning of k-means++ happens in the background. So, the k-means++ happens either in R or Python, you don’t need to actually implement it.

**Choosing The Right Number of Clusters:**

* There is a metric that helps us find the right number of clusters – Within Cluster Sum of Squares and the formula for that is –



This is an example for three clusters.

* At first glance, the formula might look a bit overwhelming, but it is super simple.
* We have three elements, and each sum is calculated for each cluster, in fact it is calculated within that cluster, hence the “Within” in the name of the formula.
* Let’s look at one of them, for instance the middle one. There, we are taking every point inside cluster 2 and we are going to sum the distance between each point in cluster 2, and the centroid of cluster 2.
* Then, we are squaring the distance. So, we are taking the sum of those squared distances.
* We do that for every cluster, as a result we get the total sum, and that is going to be our metric.

It is quite a good metric in terms of understanding, or comparing, the goodness-of-fit between two k-means clustering.

* The higher the number of clusters, the better the goodness-of-fit.
* We can have the same number of clusters as our data points, meaning every single data point has its own cluster, but then the WCSS will equate to 0 because every single point will have its own cluster and the distance between the points and the centroid will be 0, so will be the square.
* So, we can understand that the value of WCSS will come down from a value anywhere between 0 and 1, all the way down to 0 as we increase the number of clusters.
* Thus, the lesser the WCSS the better the goodness of fit.
* To choose the optimal number of clusters, use the elbow method.

Chart, line chart

Description automatically generated

* The chart above shows the WCSS for different number of clusters. So, in that chart look for the elbow – where the value of WCSS drops certain units, but not significantly.
* But this method is quite arbitrary. Sometimes the situations are not that obvious – the elbow is not as evident as it is in this case. Therefore, someone will pick one number of clusters, someone else will pick some other number of clusters. That is a judgment call that you need to make as a data scientist.