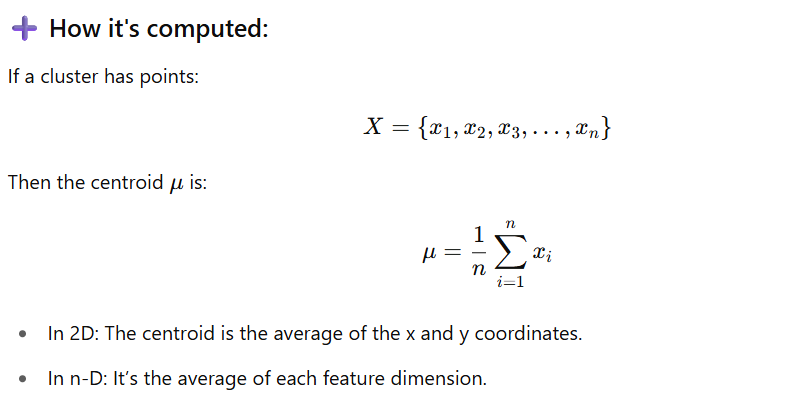
**K-Means clustering**

<https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/>

<https://github.com/campusx-official/100-days-of-machine-learning/tree/main/kmeans>

**1. What is a Centroid?**

A **centroid** is the **center** of a cluster. It is a point that represents the **average of all data points** (in terms of feature values) in that cluster.



**📦 2. What is a Cluster?**

A **cluster** is a **group of data points** that are more similar to each other than to points in other groups, based on a **distance metric** (typically Euclidean distance).

* Each cluster has:
  + A centroid
  + A set of points assigned to that cluster
* The goal of K-means is to **minimize the variance** within each cluster and **maximize the distance between centroids**.

**🔁 3. What is Reassigning Centroids?**

This is a critical step in the **K-means iterative process**. After assigning each point to the nearest centroid (i.e., forming clusters), the algorithm **recalculates** the position of the centroid by taking the **mean of all points assigned** to that cluster.

**✅ Why it's done:**

* To move the centroid closer to the "true center" of the data points currently assigned to it.
* It helps the cluster center better represent the data it owns.

**🔁 How it fits in the full process:**

1. **Initialization**: Choose initial centroids.
2. **Assignment Step**: Assign each data point to the **nearest centroid**.
3. **Update Step** (Reassign Centroids): For each cluster:
   * Compute the **mean position** of all assigned points.
   * **Update** the centroid to this new position.
4. **Repeat** steps 2 and 3 until convergence (no significant change).

**🧠 Intuition Example:**

Imagine 3 people standing in a room representing data points. Their “cluster center” is just the average position of the three. If someone moves in or out of the group (based on who’s closer), the center moves to reflect that — this is **reassigning the centroid**.

**Benefits of K-Means**

1. **Simple and Easy to Understand**
   * Intuitive logic and easy to implement.
2. **Fast and Scalable**
   * Time complexity is **O(n × k × i)** (n = data points, k = clusters, i = iterations).
   * Performs well on large datasets.
3. **Efficient in Practice**
   * When clusters are well-separated and spherical, K-Means performs exceptionally well.
4. **Works with High-Dimensional Data**
   * Can be used effectively in high-dimensional spaces, especially with dimensionality reduction (like PCA).
5. **General-Purpose Clustering**
   * Useful for a wide range of applications: customer segmentation, document clustering, image compression, etc.

**❌ Drawbacks of K-Means**

1. **You Must Specify K**
   * The algorithm requires the number of clusters K to be known in advance.
2. **Sensitive to Initial Centroids**
   * Poor initialization can lead to suboptimal solutions.
   * Solved partially using **K-Means++ initialization**.
3. **Assumes Spherical Clusters**
   * Performs poorly on clusters that are not spherical or vary in size/density.
4. **Not Robust to Outliers**
   * Outliers can significantly skew the centroids, leading to incorrect clustering.
5. **Can Converge to Local Minima**
   * Final result may depend on the initial placement of centroids.
6. **Only Works with Numerical Data**
   * K-Means uses Euclidean distance, so it's not suitable for categorical features without transformation.

**Key Hyperparameters in K-Means**

| **Hyperparameter** | **Description** |
| --- | --- |
| **n\_clusters** | The number of clusters **K** you want to form. |
| **init** | Method for initializing centroids. Common values:  • "random" – choose K random data points as initial centroids.  • "k-means++" – smarter initialization that speeds up convergence (default and recommended). |
| **n\_init** | Number of times the algorithm will run with different centroid seeds. The final result will be the best output (lowest WCSS). Default is 10. |
| **max\_iter** | Maximum number of iterations allowed for each single run of K-means. Default is 300. |
| **tol** | Tolerance to declare convergence — when centroids move less than this value between iterations. Default is 1e-4. |
| **random\_state** | Controls randomness for reproducibility. |
| **algorithm** | K-means computation algorithm:  • "lloyd" – standard K-Means (default)  • "elkan" – faster with sparse data (uses triangle inequality). |

from sklearn.cluster import KMeans

kmeans = KMeans(

n\_clusters=4,

init='k-means++',

n\_init=10,

max\_iter=300,

tol=1e-4,

random\_state=42,

algorithm='lloyd'

)

**What is the Elbow Method**

how Elbow method helps us to know how many clusters should be complete (graph between wcss and number of clusters uses) explain full process to get cluster count

<https://www.analyticsvidhya.com/blog/2021/01/in-depth-intuition-of-k-means-clustering-algorithm-in-machine-learning/>

**What is K-Means Clustering?**

K-means clustering is a popular unsupervised [**machine learning algorithm**](https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/) used for partitioning a dataset into a pre-defined number of clusters. The goal is to group similar data points together and discover underlying patterns or structures within the data.

* Recall the first property of clusters – it states that the points within a cluster should be similar to each other. So, our aim here is to minimize the distance between the points within a cluster.
* There is an algorithm that tries to minimize the distance of the points in a cluster with their centroid – the k-means clustering technique.
* K-means is a centroid-based algorithm or a distance-based algorithm, where we calculate the distances to assign a point to a cluster. In K-Means, each cluster is associated with a centroid.

“**The main objective of the K-Means algorithm is to minimize the sum of distances between the points and their respective cluster centroid.**”

Optimization plays a crucial role in the k-means clustering algorithm. The goal of the optimization process is to find the best set of centroids that minimizes the sum of squared distances between each data point and its closest centroid.

**How K-Means Clustering Works?**

Here’s how it works:

1. **Initialization**: Start by randomly selecting K points from the dataset. These points will act as the initial cluster centroids.
2. **Assignment**: For each data point in the dataset, calculate the distance between that point and each of the K centroids. Assign the data point to the cluster whose centroid is closest to it. This step effectively forms K clusters.
3. **Update centroids**: Once all data points have been assigned to clusters, recalculate the centroids of the clusters by taking the mean of all data points assigned to each cluster.
4. **Repeat**: Repeat steps 2 and 3 until convergence. Convergence occurs when the centroids no longer change significantly or when a specified number of iterations is reached.
5. **Final Result**: Once convergence is achieved, the algorithm outputs the final cluster centroids and the assignment of each data point to a cluster.

**Objective of k means Clustering**

The main objective of k-means clustering is to partition your data into a specific number (k) of groups, where data points within each group are similar and dissimilar to points in other groups. It achieves this by minimizing the distance between data points and their assigned cluster’s center, called the centroid.

Here’s an objective:

* **Grouping similar data points:** K-means aims to identify patterns in your data by grouping data points that share similar characteristics together. This allows you to discover underlying structures within the data.
* **Minimizing within-cluster distance:** The algorithm strives to make sure data points within a cluster are as close as possible to each other, as measured by a distance metric (usually Euclidean distance). This ensures tight-knit clusters with high cohesiveness.
* **Maximizing between-cluster distance:** Conversely, k-means also tries to maximize the separation between clusters. Ideally, data points from different clusters should be far apart, making the clusters distinct from each other.

**What is Clustering?**

Cluster analysis is a technique in [**data mining**](https://www.analyticsvidhya.com/blog/2021/05/introduction-to-data-mining-and-its-applications/) and machine learning that groups similar objects into clusters. K-means clustering, a popular method, aims to divide a set of objects into K clusters, minimizing the sum of squared distances between the objects and their respective cluster centers.

Hierarchical clustering and k-means clustering are two popular techniques in the field of unsupervised learning used for clustering data points into distinct groups. While k-means clustering divides data into a predefined number of clusters, hierarchical clustering creates a hierarchical tree-like structure to represent the relationships between the clusters.

**Example of Clustering**

Let’s try understanding this with a simple example. A bank wants to give credit card offers to its customers. Currently, they look at the details of each customer and, based on this information, decide which offer should be given to which customer.

Now, the bank can potentially have millions of customers. Does it make sense to look at the details of each customer separately and then make a decision? Certainly not! It is a manual process and will take a huge amount of time.

So what can the bank do? One option is to segment its customers into different groups. For instance, the bank can group the customers based on their income:



Can you see where I’m going with this? The bank can now make three different strategies or offers, one for each group. Here, instead of creating different strategies for individual customers, they only have to make 3 strategies. This will reduce the effort as well as the time.

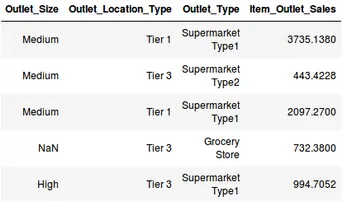
**The groups I have shown above are known as clusters, and the process of creating these groups is known as clustering.** Formally, we can say that:

* Clustering is the process of dividing the entire data into groups (also known as clusters) based on the patterns in the data.
* Can you guess which type of learning problem clustering is? Is it a [**supervised or unsupervised learning**](https://www.analyticsvidhya.com/blog/2020/04/supervised-learning-unsupervised-learning/) problem?

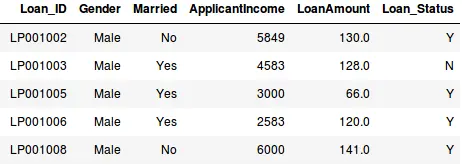
Think about it for a moment and use the example we just saw. Got it? Clustering is an unsupervised learning problem!

How is Clustering an Unsupervised Learning Problem?

Let’s say you are working on a project where you need to predict the sales of a big mart:



Or, a project where your task is to predict whether a loan will be approved or not:



We have a fixed target to predict in both of these situations. In the sales prediction problem, we have to predict the *Item\_Outlet\_Sales* based on *outlet\_size, outlet\_location\_type*, etc., and in the loan approval problem, we have to predict the *Loan\_Status* depending on the Gender, marital status, the income of the customers, etc.

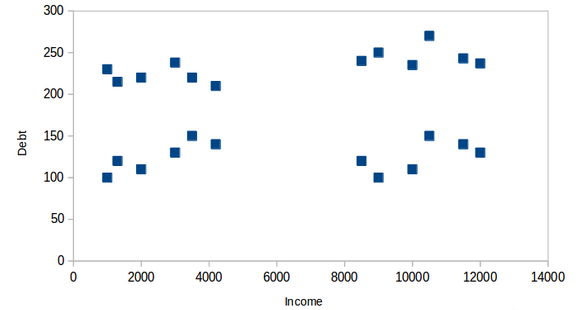
* So, when we have a target variable to predict based on a given set of predictors or independent variables, such problems are called supervised learning problems.
* Now, there might be situations where we do *not* have any target variable to predict.
* Such problems, without any fixed target variable, are known as unsupervised learning problems. In these problems, we only have the independent variables and no target/dependent variable.

**In clustering, we do not have a target to predict. We look at the data, try to club similar observations, and form different groups. Hence it is an unsupervised learning problem.**

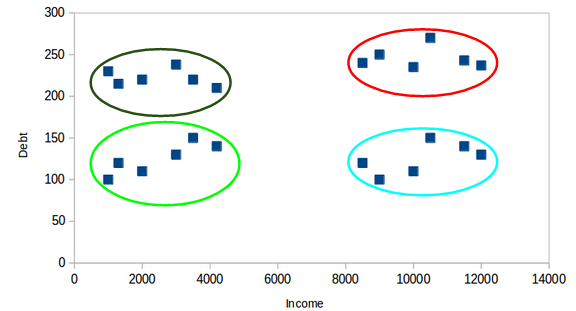
We now know what clusters are and the concept of clustering. Next, let’s look at the properties of these clusters, which we must consider while forming the clusters.

Properties of K means Clustering

How about another example of k-means clustering algorithm? We’ll take the same bank as before, which wants to segment its customers. For simplicity purposes, let’s say the bank only wants to use the income and debt to make the [**segmentation**](https://www.analyticsvidhya.com/blog/2016/02/guide-build-predictive-models-segmentation/). They collected the customer data and used a scatter plot to visualize it:



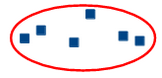
On the X-axis, we have the income of the customer, and the y-axis represents the amount of debt. Here, we can clearly visualize that these customers can be segmented into 4 different clusters, as shown below:



This is how clustering helps to create segments (clusters) from the data. The bank can further use these clusters to make strategies and offer discounts to its customers. So let’s look at the properties of these clusters.

**First Property of K-Means Clustering Algorithm**

All the data points in a cluster should be similar to each other. Let me illustrate it using the above example:

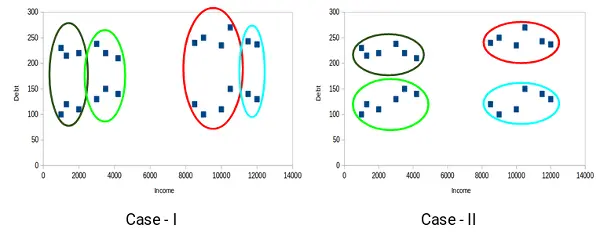


If the customers in a particular cluster are not similar to each other, then their requirements might vary, right? If the bank gives them the same offer, they might not like it, and their interest in the bank might reduce. Not ideal.

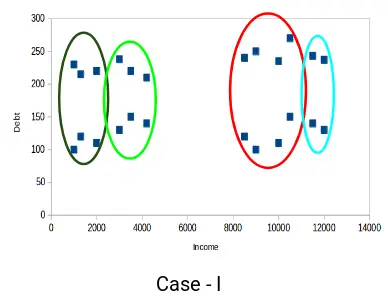
Having similar data points within the same cluster helps the bank to use targeted marketing. You can think of similar examples from your everyday life and consider how clustering will (or already does) impact the business strategy.

**Second Property of K-Means Clustering Algorithm**

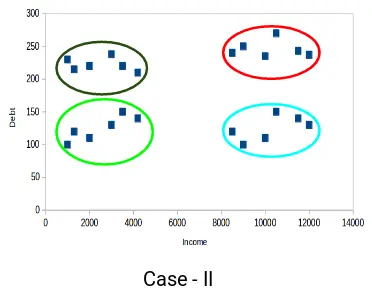
The data points from different clusters should be as different as possible. This will intuitively make sense if you’ve grasped the above property. Let’s again take the same example to understand this property:



Which of these cases do you think will give us the better clusters? If you look at case I:



Customers in the red and blue clusters are quite similar to each other. The top four points in the red cluster share similar properties to those of the blue cluster’s top two customers. They have high incomes and high debt values. Here, we have clustered them differently. Whereas, if you look at case II:



Points in the red cluster completely differ from the customers in the blue cluster. All the customers in the red cluster have high income and high debt, while the customers in the blue cluster have high income and low debt value. Clearly, we have a better clustering of customers in this case.

Hence, data points from different clusters should be as different from each other as possible to have more meaningful clusters. The k-means algorithm uses an iterative approach to find the optimal cluster assignments by minimizing the sum of squared distances between data points and their assigned cluster centroid.

So far, we have understood what clustering is and the different properties of clusters. But why do we even need clustering? Let’s clear this doubt in the next section and look at some applications of [**clustering**](https://github.com/topics/k-means-clustering).

***Clear your Understanding about the***[***Elbow Method for Optimal Cluster Number in K-Means***](https://www.analyticsvidhya.com/blog/2021/01/in-depth-intuition-of-k-means-clustering-algorithm-in-machine-learning/)

Applications of Clustering in Real-World Scenarios

Clustering is a widely used technique in the industry. It is being used in almost every domain, from banking and [**recommendation engines**](https://www.analyticsvidhya.com/blog/2021/07/recommendation-system-understanding-the-basic-concepts/) to document clustering and image segmentation.

**Customer Segmentation**

We covered this earlier – one of the most common applications of clustering is [**customer segmentation**](https://www.analyticsvidhya.com/blog/2021/06/how-to-solve-customer-segmentation-problem-with-machine-learning/). And it isn’t just limited to banking. This strategy is across functions, including telecom, e-commerce, sports, advertising, sales, etc.

**Document Clustering**

This is another common application of clustering. Let’s say you have multiple documents and you need to cluster similar documents together. Clustering helps us group these documents such that similar documents are in the same clusters.

**Image Segmentation**

We can also use clustering to perform image segmentation. Here, we try to club similar pixels in the image together. We can apply clustering to create clusters having similar pixels in the same group.

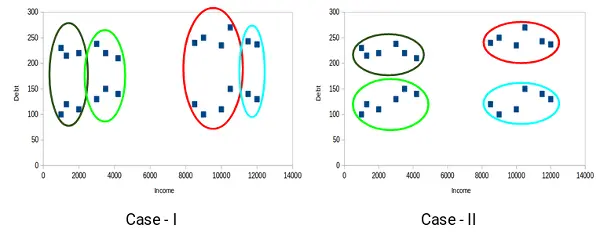
**Recommendation Engines**

Clustering can also be used in recommendation engines. Let’s say you want to recommend songs to your friends. You can look at the songs liked by that person and then use clustering to find similar songs and finally recommend the most similar songs.

There are many more applications that I’m sure you have already thought of. You can share these applications in the comments section below. Next, let’s look at how we can evaluate our clusters.

**Understanding the Different Evaluation Metrics for Clustering**

The primary aim of clustering is not just to make clusters but to make good and meaningful ones. We saw this in the below example:



Here, we used only two features, and hence it was easy for us to visualize and decide which of these clusters was better.

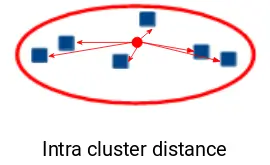
Unfortunately, that’s not how real-world scenarios work. We will have a ton of features to work with. Let’s take the customer segmentation example again – we will have features like customers’ income, occupation, gender, age, and many more. We would not be able to visualize all these features together and decide on better and more meaningful clusters.

This is where we can make use of [**evaluation metrics**](https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/). Let’s discuss a few of them and understand how we can use them to evaluate the quality of our clusters.

Inertia

Recall the first property of clusters we covered above. This is what inertia evaluates. It tells us how far the points within a cluster are. So,inertia actually calculates the sum of distances of all the points within a cluster from the centroid of that cluster. Normally, we use Euclidean distance as the distance metric, as long as most of the features are numeric; otherwise, Manhattan distance in case most of the features are categorical.

We calculate this for all the clusters; the final inertial value is the sum of all these distances. This distance within the clusters is known as intracluster distance. So, inertia gives us the sum of intracluster distances:



Now, what do you think should be the value of inertia for a good cluster? Is a small inertial value good, or do we need a larger value? We want the points within the same cluster to be similar to each other, right? Hence, the distance between them should be as low as possible.

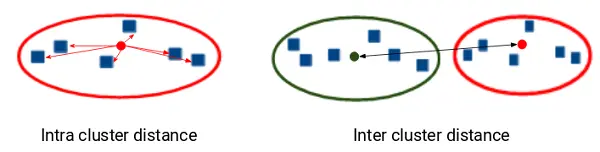
Keeping this in mind, we can say that the lesser the inertia value, the better our clusters are.

Dunn Index

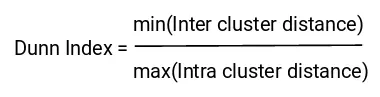
We now know that inertia tries to minimize the intracluster distance. It is trying to make more compact clusters.

Let me put it this way – if the distance between the centroid of a cluster and the points in that cluster is small, it means that the points are closer to each other. So, inertia makes sure that the first property of clusters is satisfied. But it does not care about the second property – that different clusters should be as different from each other as possible.

This is where the Dunn index comes into action.

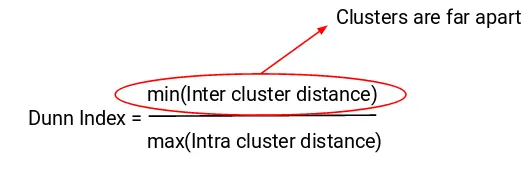


Along with the distance between the centroid and points, the Dunn index also takes into account the distance between two clusters. This distance between the centroids of two different clusters is known as inter-cluster distance. Let’s look at the formula of the Dunn index:

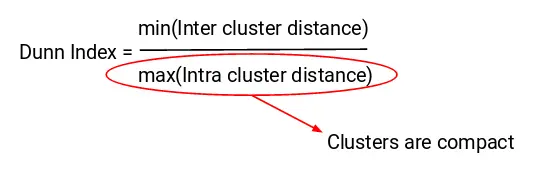


Dunn index is the ratio of the minimum of inter-cluster distances and maximum of intracluster distances.

We want to maximize the Dunn index. The more the value of the Dunn index, the better the clusters will be. Let’s understand the intuition behind the Dunn index:



In order to maximize the value of the Dunn index, the numerator should be maximum. Here, we are taking the minimum of the inter-cluster distances. So, the distance between even the closest clusters should be more which will eventually make sure that the clusters are far away from each other.



Also, the denominator should be minimum to maximize the Dunn index. Here, we are taking the maximum of all intracluster distances. Again, the intuition is the same here. The maximum distance between the cluster centroids and the points should be minimum, eventually ensuring that the clusters are compact.

Silhouette Score

The silhouette score and plot are used to evaluate the quality of a clustering solution produced by the k-means algorithm. The silhouette score measures the similarity of each point to its own cluster compared to other clusters, and the silhouette plot visualizes these scores for each sample. A high silhouette score indicates that the clusters are well separated, and each sample is more similar to the samples in its own cluster than to samples in other clusters. A silhouette score close to 0 suggests overlapping clusters, and a negative score suggests poor clustering solutions.

How to Apply K-Means Clustering Algorithm?

Let’s now take an example to understand how K-Means Clustering actually works:



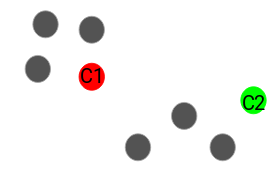
Time needed: 10 minutes

We have these 8 points, and we want to apply k-means to create clusters for these points. Here’s how we can do it.

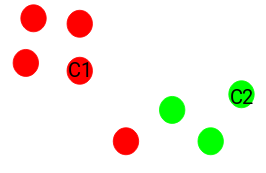
1. **Choose the number of clusters *k***

The first step in k-means is to pick the number of clusters, k.

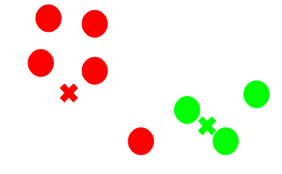
1. **Select k random points from the data as centroids**

Next, we randomly select the centroid for each cluster. Let’s say we want to have 2 clusters, so k is equal to 2 here. We then randomly select the centroid:  
  
Here, the red and green circles represent the centroid for these clusters.

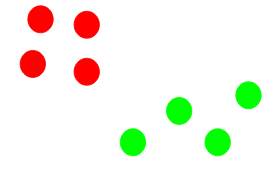
1. **Assign all the points to the closest cluster centroid**

Once we have initialized the centroids, we assign each point to the closest cluster centroid:  
Here you can see that the points closer to the red point are assigned to the red cluster, whereas the points closer to the green point are assigned to the green cluster.

1. **Recompute the centroids of newly formed clusters**

Now, once we have assigned all of the points to either cluster, the next step is to compute the centroids of newly formed clusters:  
  
Here, the red and green crosses are the new centroids.

1. **Repeat steps 3 and 4**

We then repeat steps 3 and 4:  
  
*The step of computing the centroid and assigning all the points to the cluster based on their distance from the centroid is a single iteration*. But wait – when should we stop this process? It can’t run till eternity, right?

**Stopping Criteria for K-Means Clustering**

There are essentially three stopping criteria that can be adopted to stop the K-means algorithm:

1. Centroids of newly formed clusters do not change
2. Points remain in the same cluster
3. Maximum number of iterations is reached

We can stop the algorithm if the centroids of newly formed clusters are not changing. Even after multiple iterations, if we are getting the same centroids for all the clusters, we can say that the algorithm is not learning any new pattern, and it is a sign to stop the training.

Another clear sign that we should stop the training process is if the points remain in the same cluster even after training the algorithm for multiple iterations.

Finally, we can stop the training if the maximum number of iterations is reached. Suppose we have set the number of iterations as 100. The process will repeat for 100 iterations before stopping.

**What is the Elbow Method ?**

The Elbow Method is a technique used in data analysis and [**machine learning**](https://www.analyticsvidhya.com/machine-learning/) for determining the optimal number of clusters in a dataset. It involves plotting the variance explained by different numbers of clusters and identifying the “elbow” point, where the rate of variance decreases sharply levels off, suggesting an appropriate cluster count for analysis or model training.

This method is a visual technique used to determine the best K value for a k-means clustering algorithm. In this method, a graph known as the elbow graph plots the within-cluster-sum-of-square (WCSS) values against various K values. The optimal K value is identified at the point where the graph bends like an elbow.

What is the Elbow Method in K-Means Clustering?

The **elbow method is a graphical representation of finding the optimal ‘K’**in a [**K-means clustering**](https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/). It works by finding WCSS (Within-Cluster Sum of Square) i.e. the sum of the square distance between points in a cluster and the cluster centroid.

Let’s go through the steps involved in K-means clustering for a better understanding:

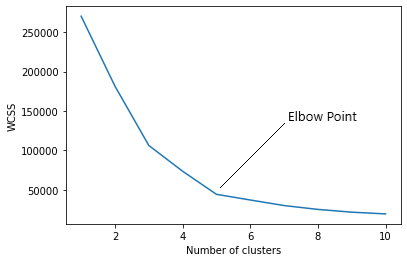
* Select the number of clusters for the dataset (K)
* Select the K number of centroids randomly from the dataset.
* Now we will use Euclidean distance or Manhattan distance as the metric to calculate the distance of the points from the nearest centroid and assign the points to that nearest cluster centroid, thus creating K clusters.
* Now we find the new centroid of the clusters thus formed.
* Again reassign the whole data point based on this new centroid, then repeat step 4. We will continue this for a given number of iterations until the position of the centroid doesn’t change, i.e., there is no more convergence.

Finding the optimal number of clusters is an important part of this algorithm. A commonly used method for finding the optimum K value is**Elbow Method.**

***Checkout this article***[***how K means Clustering Works***](https://www.analyticsvidhya.com/blog/2021/04/k-means-clustering-simplified-in-python/)***!***

K Means Clustering Using the Elbow Method

In the Elbow method, we are actually varying the number of clusters (K) from 1 – 10. For each value of K, we are calculating WCSS (Within-Cluster Sum of Square). WCSS is the sum of the squared distance between each point and the centroid in a cluster. When we plot the WCSS with the K value, the plot looks like an Elbow. As the number of clusters increases, the WCSS value will start to decrease. WCSS value is largest when K = 1. When we analyze the graph, we can see that the graph will rapidly change at a point and thus creating an elbow shape. From this point, the graph moves almost parallel to the X-axis. The K value corresponding to this point is the optimal value of K or an optimal number of clusters.



Now let’s implement K-Means clustering using Python.

Elbow method k means

The elbow method is a common technique used to determine the optimal number of clusters (k) in [**k-means clustering**](https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/). It’s a graphical approach that relies on the idea that as you increase the number of clusters, the sum of squared distances between points and their cluster centers (WCSS) will continue to decrease. This is because you’re essentially splitting the data into increasingly finer groups.

Here’s how it works:

1. **Calculate WCSS for different K values:** You iterate through a range of possible k values (number of clusters). For each k, you perform k-means clustering and calculate the WCSS. This represents the total variance within each cluster.
2. **Plot the WCSS vs. K:** You plot the WCSS values on the y-axis and the number of clusters (k) on the x-axis.
3. **Identify the elbow point:** The ideal k value is identified as the “elbow” of the curve. This is the point where the WCSS starts to decrease at a much slower rate. In essence, you’re looking for the point where adding more clusters isn’t significantly improving the fit within the clusters.

Here are some things to keep in mind about the elbow method:

* **It can be subjective:** The “elbow” might not always be a very sharp bend, and it can be difficult to determine the exact point objectively.
* **Not always the best method:** The elbow method might not be suitable for all datasets, especially for those with high dimensionality or clusters of irregular shapes.
* **Consider other factors:** While the elbow method is a helpful starting point, it’s important to consider other factors like the interpretability of the clusters and the domain knowledge of the data when choosing the final k value.pen\_spark

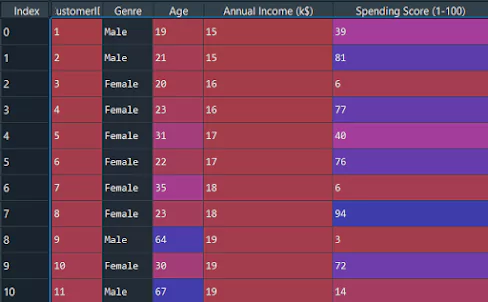
tunesharemore\_vert

***Read More about this article***[***how to choose the value if K in KNN Algorithm***](https://community.analyticsvidhya.com/c/datascience/how-to-choose-the-value-of-k-in-knn-algorithm)

Implementation of the Elbow Method

Sample Dataset

The dataset we are using here is the Mall Customers data ([**Download here**](https://www.kaggle.com/shwetabh123/mall-customers)). It’s unlabeled data that contains the details of customers in a mall (features like genre, age, annual income(k$), and spending score). Our aim is to cluster the customers based on the relevant features of annual income and spending score.



Importing Libraries

First of all, we have to import the essential libraries.

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import sklearnCopy Code

Now let’s import the given dataset and slice the important features.

dataset = pd.read\_csv('Mall\_Customers.csv')

X = dataset.iloc[:, [3, 4]].valuesCopy Code

We have to find the optimal K value for clustering the data. Now we are using the Elbow Method to find the optimal K value.

from sklearn.cluster import KMeans

wcss = [] for i in range(1, 11):

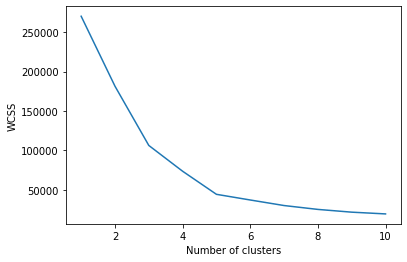
kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 42)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)Copy Code

The “init” argument is the method for initializing the centroid. We calculated the WCSS value for each K value. Now we have to plot the WCSS with the K value.

The graph will be like this:



Training the Model

The point at which the elbow shape is created is 5; that is, our K value or an optimal number of clusters is 5. Now let’s train the model on the input data with a number of clusters 5.

kmeans = KMeans(n\_clusters = 5, init = "k-means++", random\_state = 42)

y\_kmeans = kmeans.fit\_predict(X)Copy Code

y\_kmeans will be:

array([3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0,

3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 1,

3, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 4, 2, 1, 2, 4, 2, 4, 2,

1, 2, 4, 2, 4, 2, 4, 2, 4, 2, 1, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2,

4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2,

4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2,

4, 2])Copy Code

y\_kmeans gives us different clusters corresponding to X. Now, let’s plot all the clusters using matplotlib.

plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s = 60, c = 'red', label = 'Cluster1')

plt.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], s = 60, c = 'blue', label = 'Cluster2')

plt.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], s = 60, c = 'green', label = 'Cluster3)

plt.scatter(X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], s = 60, c = 'violet', label = 'Cluster4')

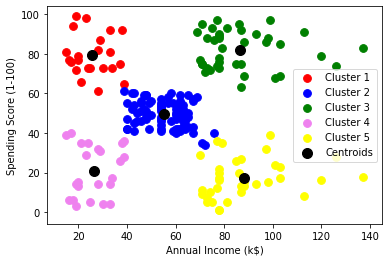
plt.scatter(X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], s = 60, c = 'yellow', label = 'Cluster5')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 100, c = 'black', label = 'Centroids')

plt.xlabel('Annual Income (k$)') plt.ylabel('Spending Score (1-100)') plt.legend()

plt.show()Copy Code

Graph:



Now we will visualize the clusters using the scatter plot. As you can see, there are 5 clusters in total that are visualized in different colors, and the centroid of each cluster is visualized in black color.

Full Code

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd # Importing the dataset

X = dataset.iloc[:, [3, 4]].values

dataset = pd.read\_csv('Mall\_Customers.csv')

from sklearn.cluster import KMeans

# Using the elbow method to find the optimal number of clusters wcss = [] for i in range(1, 11):

wcss.append(kmeans.inertia\_)

kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 42) kmeans.fit(X) plt.plot(range(1, 11), wcss) plt.xlabel('Number of clusters')

y\_kmeans = kmeans.fit\_predict(X)

plt.ylabel('WCSS') plt.show() # Training the K-Means model on the dataset kmeans = KMeans(n\_clusters = 5, init = 'k-means++', random\_state = 42) y\_kmeans = kmeans.fit\_predict(X)

# Visualising the clusters

plt.scatter( X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], s = 60, c = 'blue', label = 'Cluster2')

plt.scatter( X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s = 60, c = 'red', label = 'Cluster1') plt.scatter( X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], s = 60, c = 'green', label = 'Cluster3')

plt.scatter( kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 100, c = 'black', label = 'Centroids')

plt.scatter( X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], s = 60, c = 'violet', label = 'Cluster4') plt.scatter( X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], s = 60, c = 'yellow', label = 'Cluster5') plt.xlabel('Annual Income (k$)') plt.ylabel('Spending Score (1-100)') plt.legend()

plt.show()

**Elbow Method Drawbacks**

The elbow method, while a useful tool for determining the optimal number of clusters in K-means clustering, has some drawbacks:

* **Subjectivity**: The choice of the “elbow point” can be subjective and might vary between individuals analyzing the same data.
* **Non-Gaussian Data**: It assumes that clusters are spherical and equally sized, which may not hold for complex datasets with irregularly shaped or differently sized clusters.
* **Sensitivity to Initialization**: [**K-means**](https://www.analyticsvidhya.com/blog/2021/05/k-mean-getting-the-optimal-number-of-clusters/)itself is sensitive to initial cluster centroids, which can affect the WCSS values and, consequently, the choice of the optimal K.
* **Inefficient for Large Datasets**: For large datasets, calculating WCSS for a range of K values can be computationally expensive and time-consuming.
* **Unsuitable for All Distributions**: The elbow method is not suitable for all data distributions, especially when clusters have varying densities or are non-convex.
* **Limited to K-means**: It specifically applies to K-means clustering and may not be suitable for other clustering algorithms with different objectives.

Despite these drawbacks, the elbow method remains a valuable starting point for selecting the number of clusters, and it often provides useful insights into the data’s underlying structure. However, it’s essential to complement it with other validation techniques when working with more complex datasets or different clustering algorithms.

**Conclusion**

In this article, we covered the basic concepts of the K-Means Clustering algorithm in Machine Learning. We used the Elbow method to find the optimal K value for clustering the data in our sample data set. We then used the matplotlib Python library to visualize the clusters as a scatterplot graph. In the upcoming articles, we can learn more about different [**ML Algorithms.**](https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/)

**Key Takeaways**

* K-Means is a popular unsupervised machine-learning algorithm widely used by Data Scientists on unlabeled data.
* The k-Means Elbow method is used to find the optimal value of the K in the K-Means algorithm.

**Python code –**

**--------elbow method implementation to find K value.**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.datasets import make\_blobs

# --------------------------

# STEP 1: Generate synthetic data

# --------------------------

X, \_ = make\_blobs(n\_samples=300, centers=4, cluster\_std=0.60, random\_state=42)

# --------------------------

# STEP 2: Elbow Method - Calculate WCSS for different K values

# --------------------------

wcss = [] # list to store WCSS for each K

K\_range = range(1, 11)

for k in K\_range:

kmeans = KMeans(n\_clusters=k, init='k-means++', random\_state=42)

kmeans.fit(X)

wcss.append(kmeans.inertia\_) # .inertia\_ gives WCSS

# --------------------------

# STEP 3: Plot the Elbow graph

# --------------------------

plt.figure(figsize=(8, 5))

plt.plot(K\_range, wcss, marker='o', linestyle='--')

plt.title('Elbow Method for Optimal K')

plt.xlabel('Number of Clusters (K)')

plt.ylabel('WCSS (Inertia)')

plt.xticks(K\_range)

plt.grid(True)

plt.show()

import numpy as np

import matplotlib.pyplot as plt

# --------------------------

# STEP 1: Create synthetic data

# --------------------------

np.random.seed(42)  # for reproducibility

# Generate 2D data around three centers

data1 = np.random.randn(100, 2) + [2, 2]

data2 = np.random.randn(100, 2) + [8, 3]

data3 = np.random.randn(100, 2) + [5, 8]

# Combine the data into one dataset

X = np.vstack((data1, data2, data3))

# --------------------------

# STEP 2: Choose K and initialize centroids

# --------------------------

K = 3  # number of clusters

n\_samples, n\_features = X.shape

# Randomly select K unique points as initial centroids

random\_indices = np.random.choice(n\_samples, K, replace=False)

centroids = X[random\_indices]

# --------------------------

# STEP 3: Define helper function to compute distances

# --------------------------

def compute\_distance(a, b):

    # Returns the Euclidean distance between two vectors

    return np.sqrt(np.sum((a - b)\*\*2))

# --------------------------

# STEP 4: Run the K-means algorithm

# --------------------------

max\_iters = 100

for iteration in range(max\_iters):

    # Create a list to store which points belong to which cluster

    clusters = [[] for \_ in range(K)]

    # --------------------------

    # Assignment Step: Assign points to the closest centroid

    # --------------------------

    for point in X:

        distances = [compute\_distance(point, centroid) for centroid in centroids]

        closest\_idx = np.argmin(distances)

        clusters[closest\_idx].append(point)

    # --------------------------

    # Store old centroids to check for convergence

    # --------------------------

    old\_centroids = centroids.copy()

    # --------------------------

    # Update Step: Recompute the centroids

    # --------------------------

    for idx in range(K):

        cluster\_points = np.array(clusters[idx])

        if len(cluster\_points) > 0:

            centroids[idx] = np.mean(cluster\_points, axis=0)

    # --------------------------

    # Convergence Check

    # --------------------------

    diff = np.sum((centroids - old\_centroids)\*\*2)

    if diff < 1e-6:

        print(f"Converged at iteration {iteration}")

        break

# --------------------------

# STEP 5: Visualize the final clusters

# --------------------------

colors = ['r', 'g', 'b']

for idx, cluster in enumerate(clusters):

    cluster = np.array(cluster)

    plt.scatter(cluster[:, 0], cluster[:, 1], c=colors[idx], label=f'Cluster {idx+1}')

plt.scatter(centroids[:, 0], centroids[:, 1], c='black', marker='X', s=200, label='Centroids')

plt.title("K-Means Clustering (Manual Implementation)")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.legend()

plt.grid(True)

plt.show()