Complete Kmeans Clu

Complete Theory

+

Code Implementati

Unsupervised Machine Learning

There are No o/p feature // Here clusters are presents → Group of similar Data points

ex →

In Supervised ML

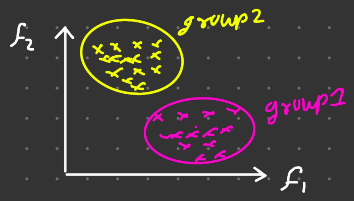
I/p: $f_1, f_2, f_3, \ldots, f_n$

Dependent feature: o/p

In Unsupervised ML

$f_1, f_2, f_3, \ldots, f_n$

Here we have not any output feature

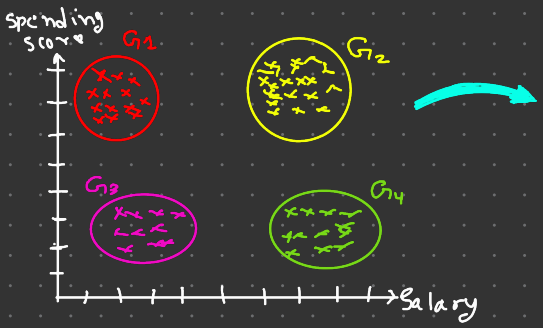


Understand with Usecase :-

↳ Dataset (customer Segmentation)

Salary	Spending score (1–10)
–	–
–	–
–	–
–	–

Matlab, we own a E-commerce website. Now we are Lonching a Brand new iphone 16. So How we target our customer by gifiting them Discount offer. We are Solving this problem with Clustering Technique ☺



Discount given to customers (in %)

G_1 → Salary ↓ Spending score ↑ → 10 %

G_2 → Salary ↑↑ Spending ↑↑ ⇒ 5 %

G_3 → Salary ↓↓ Spending ↓↓ ⇒ 15%

G_4 → Salary ↑↑ Spending ↓↓ ⇒ 30 %

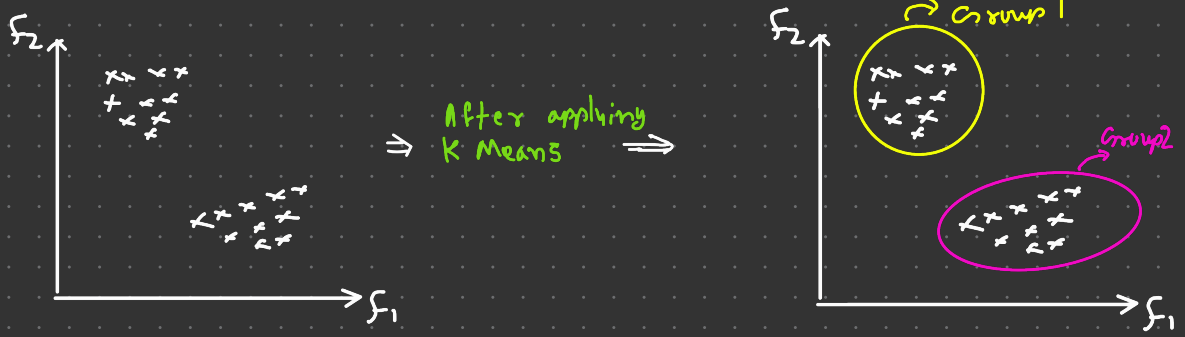
Clustering Algorithms :-

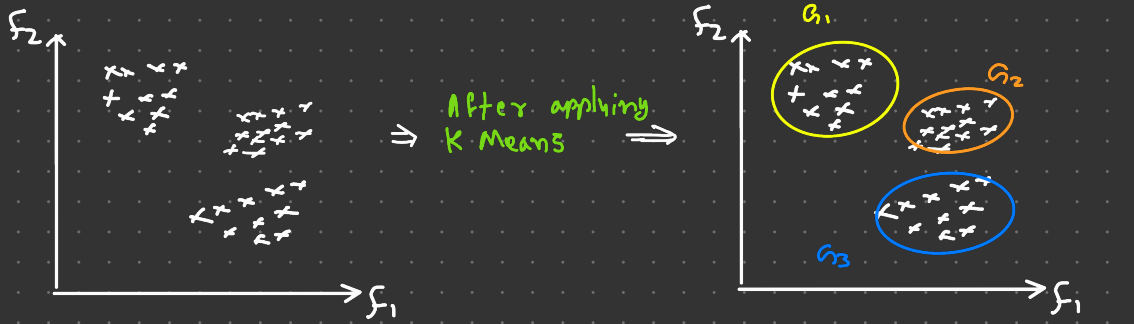
1. K Means Clustering
2. Hierarichal Clustering
3. DB Scan Clustering

1 K Means Clustering

➤ **Geometric Intuition** ↘

It - create groups of Similar looking Datapoints by using clustering





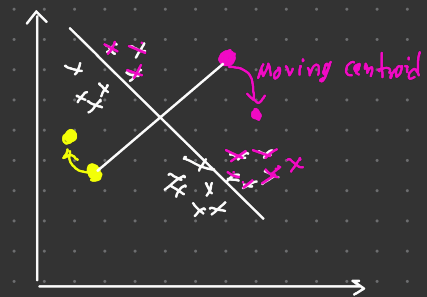
K Means Mathematical Intuition :-

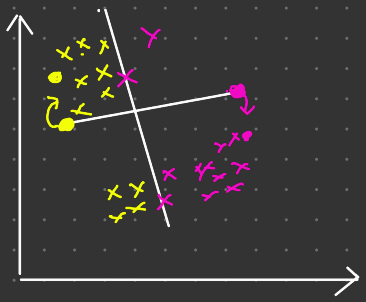
Steps

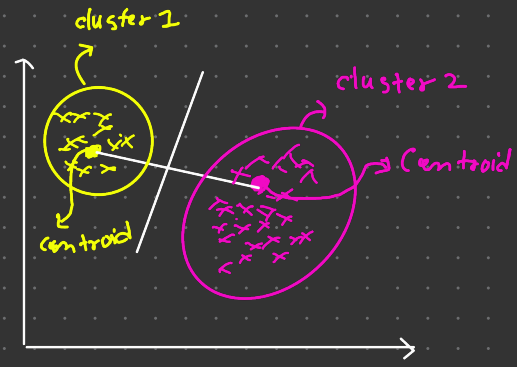
K=2

- Initialize some K → Centroid
- Points that are nearest to the centroid → Make into Groups.
- Move the centroids → By calculating mean

Repeat





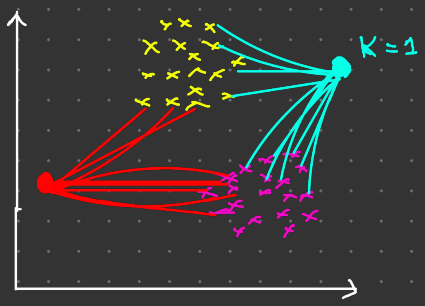


In K Mean clustering we will repeat these 3 Steps until our group is formed.

How do we select the k values ?

WCSS = Within cluster Sum of Squares

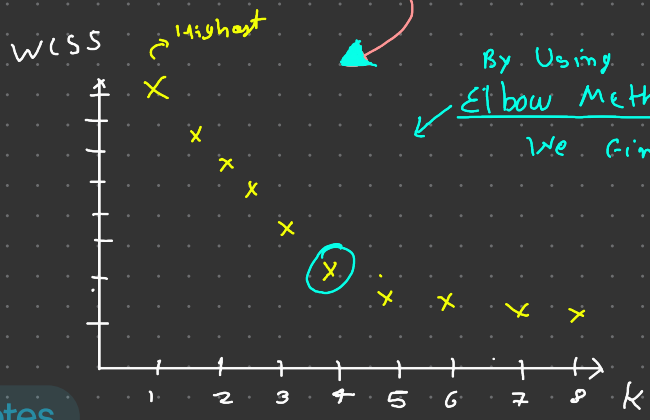
Initailize K=1 to 20



$$WCSS = \sum_{i=1}^{K} \left(\text{Distance between Points to the nearest centroid}\right)^2$$

As the value of K ↑↑ = The value of WCSS ↓↓

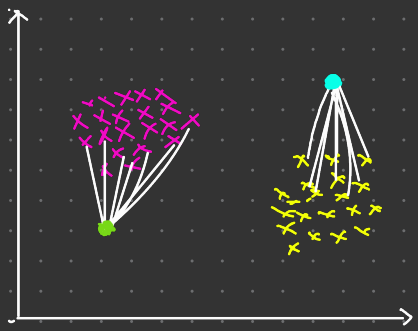
At some time, when K ↑↑ — the value of WCSS going to be Stable



By Using Elbow Method we find K value = 4

K = 4

K=2

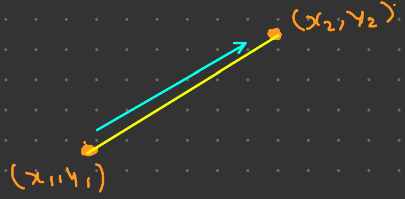


In this case we are finding two K values so Here in elbow Method when we draw the values then we got that

WCSS in K=1 > WCSS in K=2

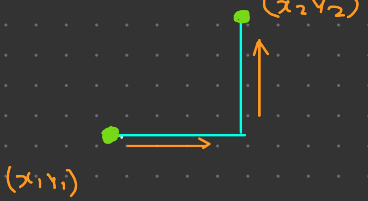
➤ The distance that we get in elbow method that will came through Eucledian and Manhathan Distance.

if we are finding Distance between two Points then,



$$\text{Eucledin Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

OR, Manhathan Distance,



$$MD = |x_2 - x_1| + |y_2 - y_1|$$

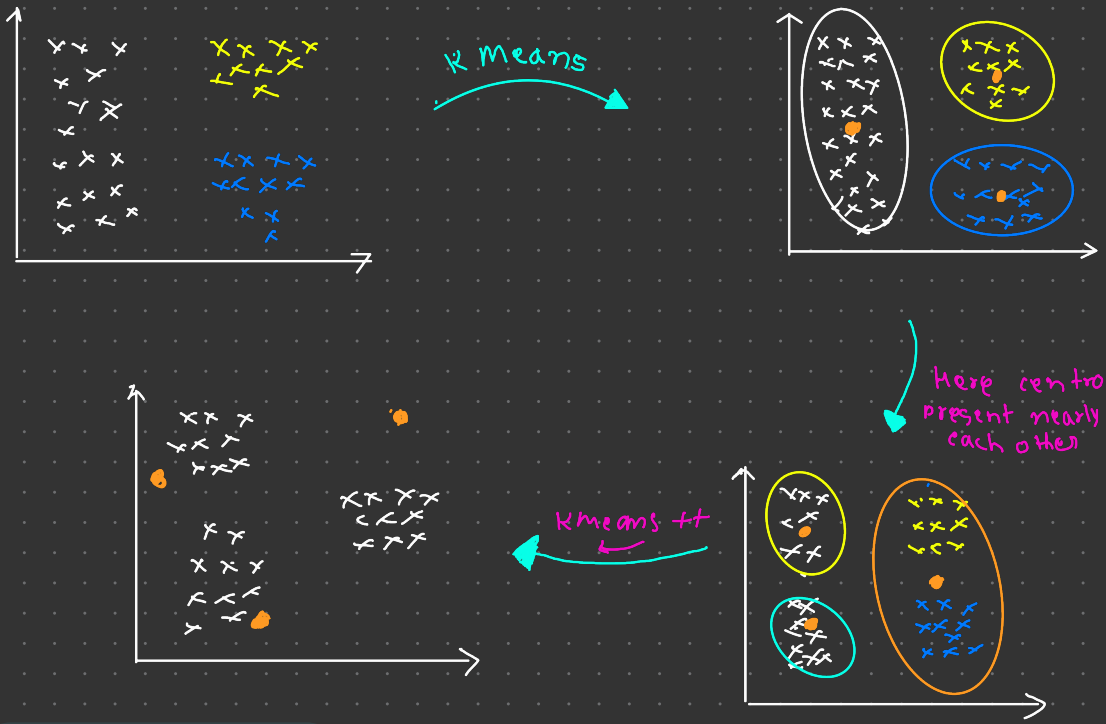
Random Initialization Trap (K Means ++) :-

↳ Random Initialization trap comes into the picture whenever our centroid is nearly intialized and this will fix by a Initialization Technique which is called "K Means++"

K Means++ make sure that as we initialize the points all their centroid are apart form each other. and also when centroids are initialize far apart we find out which particular datapoints are closest to centroid, after which it will try to group them.

K=3

K Means ++ Initialization Technique



Performance Metrix

When we apply clustering we have to validate it to check that it is a proper cluster. To check these things, we use Silhouette score technique of use cards. At the end this will be using to check the models Performace is it working properly or not.

Silhouette Clustering :-

Steps

➤ For each cluster, we compute a(i). Which tells us what will be the distance of all the points from any one point, and by doing the sum (+) this distance, we will find out the mean. And later we will calculate its final average.

$$a(i) = \frac{1}{|C_i| - 1} \sum_{\substack{j \in C_i, \\ i \neq j}} d(i,j)$$

C_i = No. of points belonging to cluster

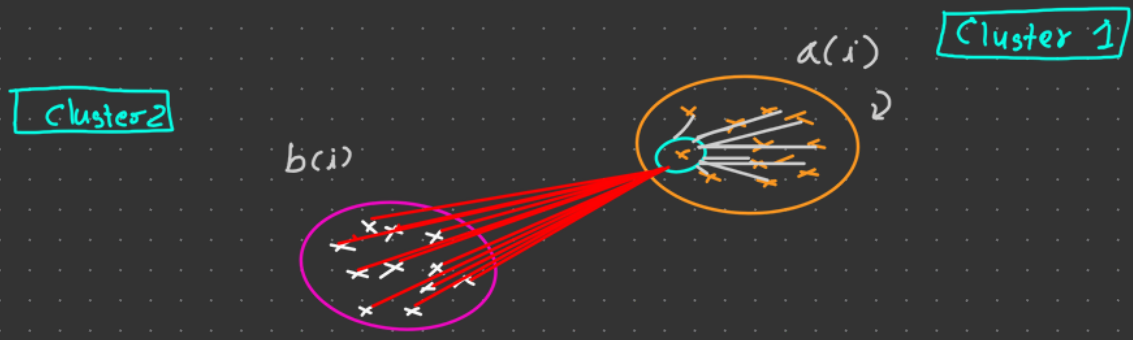
j = other points in the cluster.

a(i) ⟹

a(i) = Internally, whenever we look at any data point from other data points within the same cluster, we will get a(i).

▶ b(i) = We will calculate the distance beteen the points of cluster 2 and the nearest point of cluster 1 and then sum them. and calculate their average.

$$b(i) = \min_{J \neq I} \frac{1}{|C_J|} \sum_{J \in C_J} d(i,J)$$



a(i) > b(i) → We Perform well clustering because here internal distance is very minimum. When compare to other cluster's distance.

a(i) < b(i) → Bad cluster

Silhouette Score :-

$$S(i) = \frac{b(i) - a(i)}{\max [a(i), b(i)]}, \text{ if } |C_z| > 1$$

Here →

$$S(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i), & \text{if } a(i) > b(i) \end{cases}$$

From the above defination it is clear that

$$-1 \leq s(i) \leq 1$$

↳ the more value towards +1, Get more better cluster.

↳ the more value towards -1, Get more bad cluster.

e3uk4rahd

October 17, 2023

1 K Means Clustering Algorithms Implementation

[1]:
```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs       # for generating the dataset form␣
 ↪sklearn
import pandas as pd
import numpy as np
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

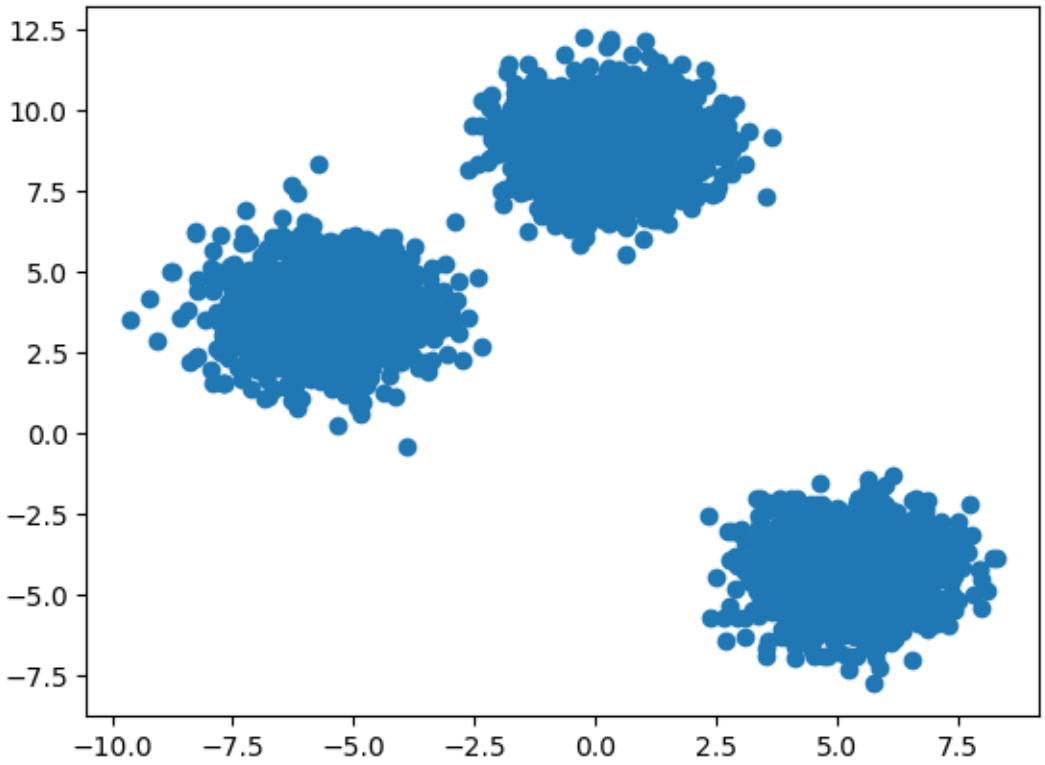
[2]:
```
'''In clustering algorithm we don't use Y value but this library giving Y␣
 ↪values but
 but ignore it completely here.'''

x, y = make_blobs(n_samples=5000, centers= 3, n_features= 2, random_state= 23)
```

[3]:
```
x.shape
```

[3]:
```
(5000, 2)
```

[4]:
```
plt.scatter(x[:,0], x[:,1])
```

[4]:
```
<matplotlib.collections.PathCollection at 0x7fa728eb2320>
```



[5]:
```
# train-test-split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size= 0.33,␣
 ↪random_state= 42)
```

[6]:
```
# importing the Kmeans from sklearn
from sklearn.cluster import KMeans
```

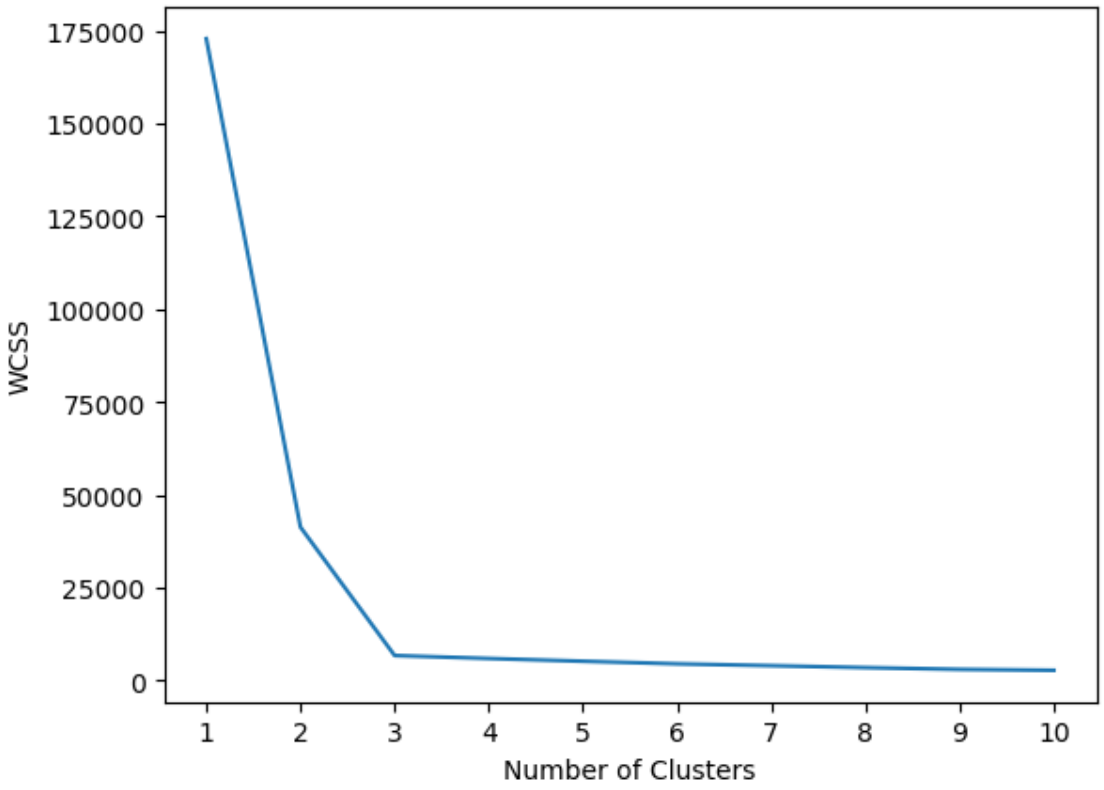
2 Manual Method:

[7]:
```
# by using Elbow Method to select the K value

wcss = []
for k in range(1,11):
    kmeans = KMeans(n_clusters= k, init= 'k-means++')
    kmeans.fit(x_train)    ## applying fit on x_train only
    wcss.append(kmeans.inertia_)
```

[8]:
```
wcss
```

[8]:
```
[172814.17587621388,
 41337.31950532506,
 6747.20268772709,
 5958.951402663642,
 5219.622979783501,
 4520.21339945963,
 4023.4897684702546,
 3500.1695326315553,
 3027.141523665331,
 2784.762579931464]
```

[9]:
```
# Now plot the Elbow curve
plt.plot(range(1,11), wcss)
plt.xticks(range(1,11))
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```



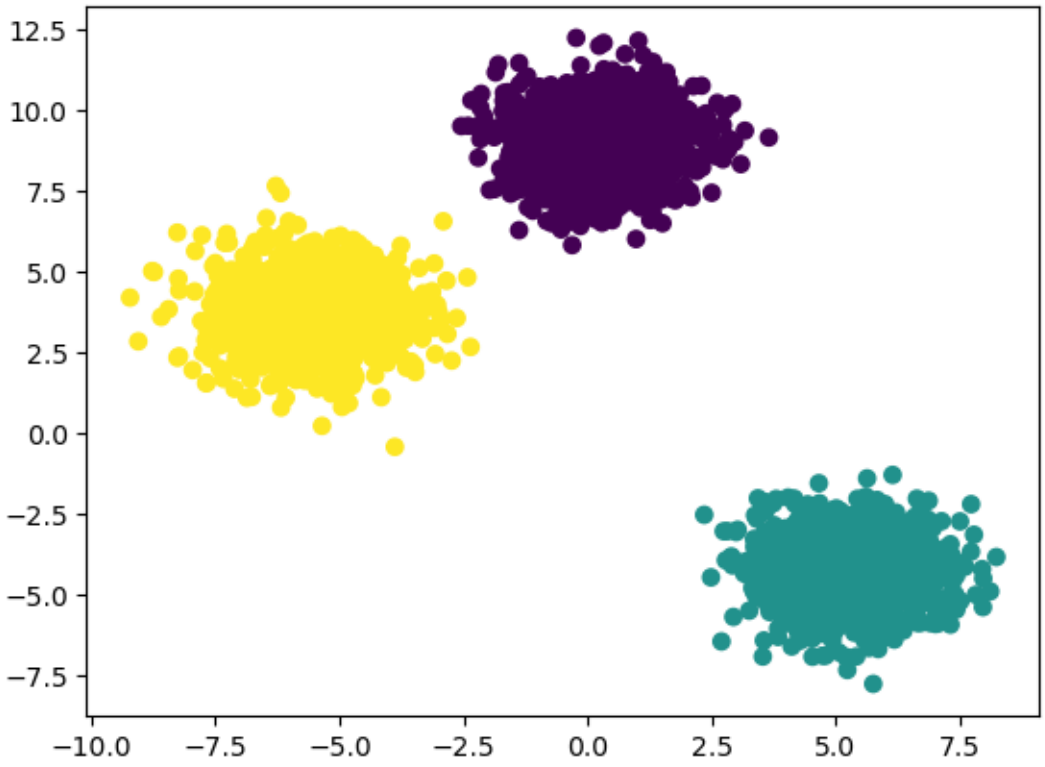
[10]:
```
# by this elbow plot we can see that our k value in 3. now make the model by␣
 ↪using no. of cluester = 3
```

```
kmeans = KMeans(n_clusters= 3, init= 'k-means++')
```

[11]:
```
# labels for train data
y_labels = kmeans.fit_predict(x_train)  ## finding the  laybal of x-train data␣
 ↪(output for the Kmeans clustering)
```

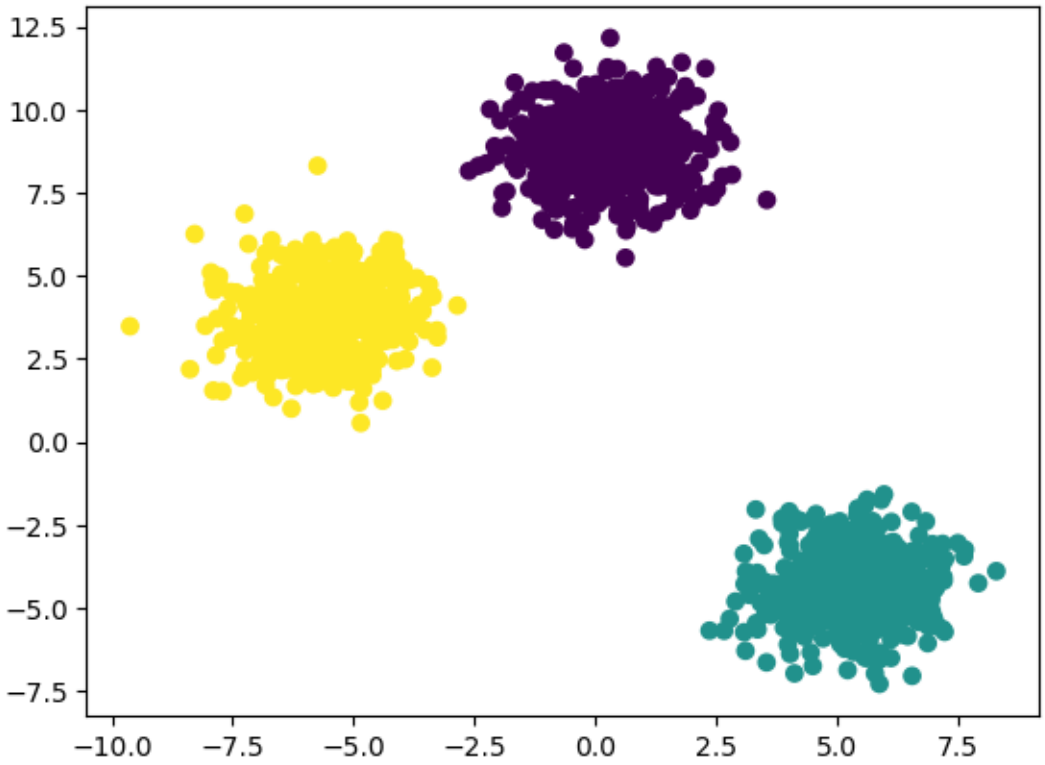
[12]:
```
# checking in the scater plot
plt.scatter(x_train[:,0], x_train[:,1], c = y_labels)
```

[12]: <matplotlib.collections.PathCollection at 0x7fa71fe63b20>



[13]:
```
# labels for test data
y_test_labels = kmeans.predict(x_test)
plt.scatter(x_test[:,0], x_test[:,1], c = y_test_labels)
```

[13]: <matplotlib.collections.PathCollection at 0x7fa71c4e3a00>



[ ]:

3 Method 2: (Automation of Kmeans Clustering)

[14]:
```
# knee locater
!pip install kneed
```

```
Requirement already satisfied: kneed in /opt/conda/lib/python3.10/site-packages
(0.8.5)
Requirement already satisfied: scipy>=1.0.0 in /opt/conda/lib/python3.10/site-
packages (from kneed) (1.9.3)
Requirement already satisfied: numpy>=1.14.2 in /opt/conda/lib/python3.10/site-
packages (from kneed) (1.23.5)
```

[15]:
```
from kneed import KneeLocator
```

[16]:
```
'''
fix terms of kneelocator:-
which one to choose: convex = when WCSS == decrease  and direction == decrease
                     concave = when WCSS== increase and direaction == increase.
```

```
'''
k1 = KneeLocator(range(1,11), wcss, curve= 'convex', direction= 'decreasing' )
print('n_cluster is :',k1.elbow)
```

```
n_cluster is : 3
```

[ ]:

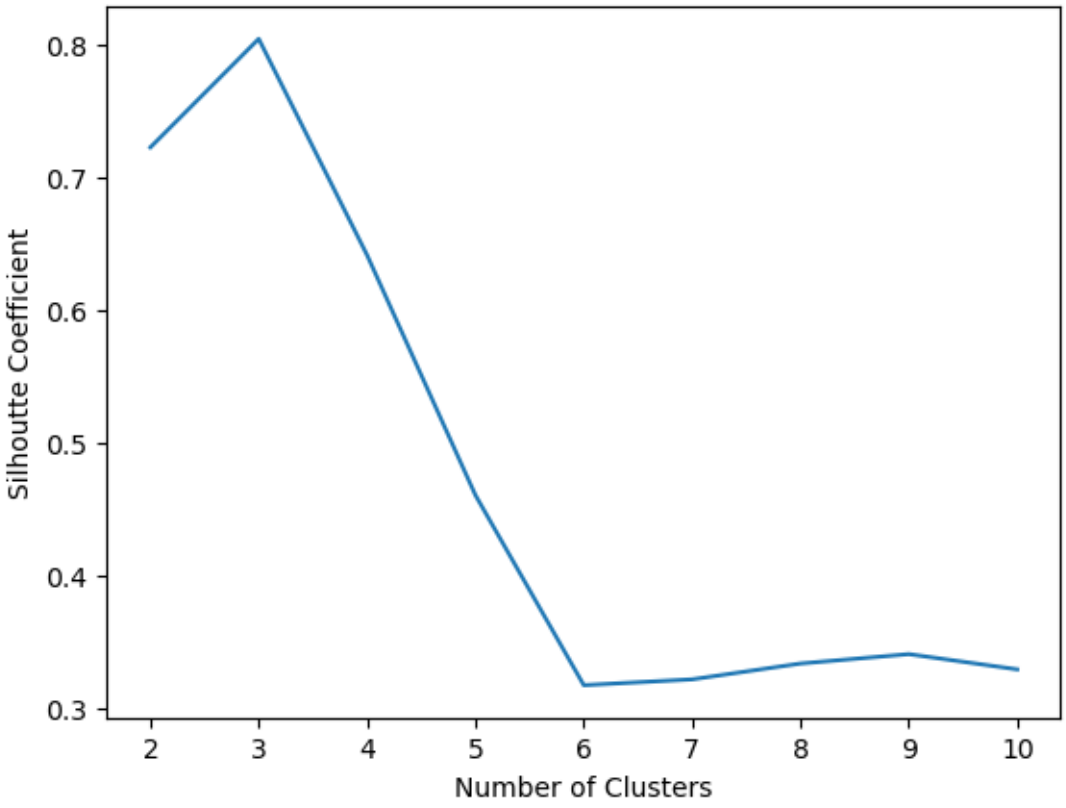
4 Performance Metrics

[17]:
```
# Silhouette score
from sklearn.metrics import silhouette_score

silhouette_coefficient = []
for k in range(2,11):
    kmeans = KMeans(n_clusters= k, init= 'k-means++')
    kmeans.fit(x_train)     ## aplying fitting on x-train
    score = silhouette_score(x_train, kmeans.labels_)
    silhouette_coefficient.append(score)
```

[18]:
```
silhouette_coefficient
```

[18]:
```
[0.7225506868743963,
 0.8043128899815283,
 0.6413848248731825,
 0.46049350524387034,
 0.31728491423500804,
 0.3216752509352751,
 0.33360477825859886,
 0.3406824455710321,
 0.32919764906096666]
```

[19]:
```
# ploting sisilhouette- score
plt.plot(range(2,11), silhouette_coefficient)
plt.xticks(range(2,11))
plt.xlabel('Number of Clusters')
plt.ylabel('Silhoutte Coefficient')
plt.show()
```



5 Observation:

For k = 3 silhouette score is higest and more than 80% which show that for k=3 perform better than other values. and we know that silhouette score value range between -1 to +1 and for k= 3 it more towards to +1.

[ ]:

[ ]:

[ ]:

[ ]: