MACHINE LEARNING

PROJECT 3

SRN: PES1UG19EC192

NAME: PALADI NAVYA SREE

SECTION:C

K MEAN CLUSTERING ALGORITHM:

K-means clustering tries to group similar kinds of items in form of clusters. It finds the similarity between the items and groups them into the clusters. K-means clustering algorithm works in three steps. Let's see what are these three steps.

1. Select the k values.
2. Initialize the centroids.
3. Select the group and find the average.

Let us understand the above steps with the help of the figure because a good picture is better than the thousands of words.

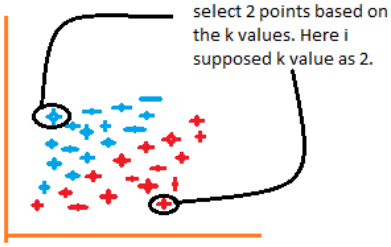


Figure 1

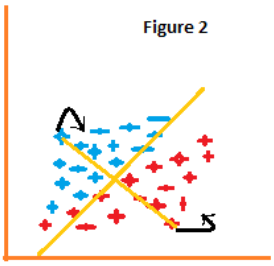
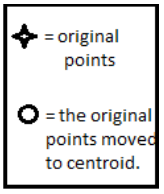
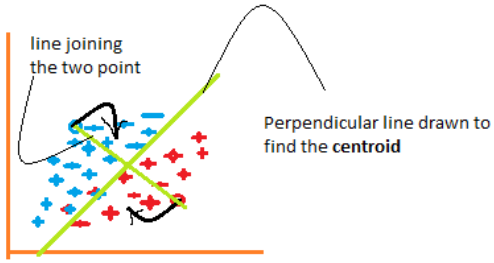


Figure 2



F2: Find the average of all the blue points and red points and move the selected points to **centroid.**



F3: Some of the red points changed to blue points, that means they belong to the group blue now. Again the repeat the same process.

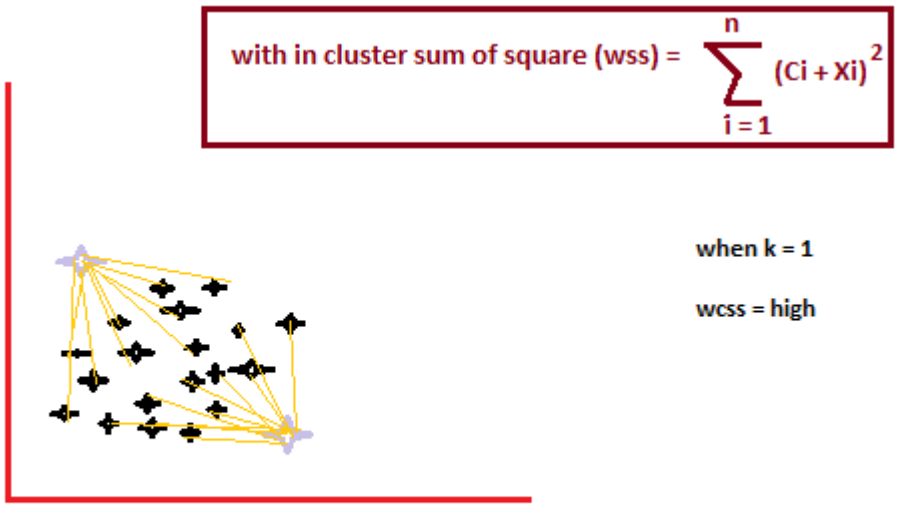
F4: The same process has been applied here. This process will be continued until we get the **two complete different cluster.**

Additional information :

Elbow Method

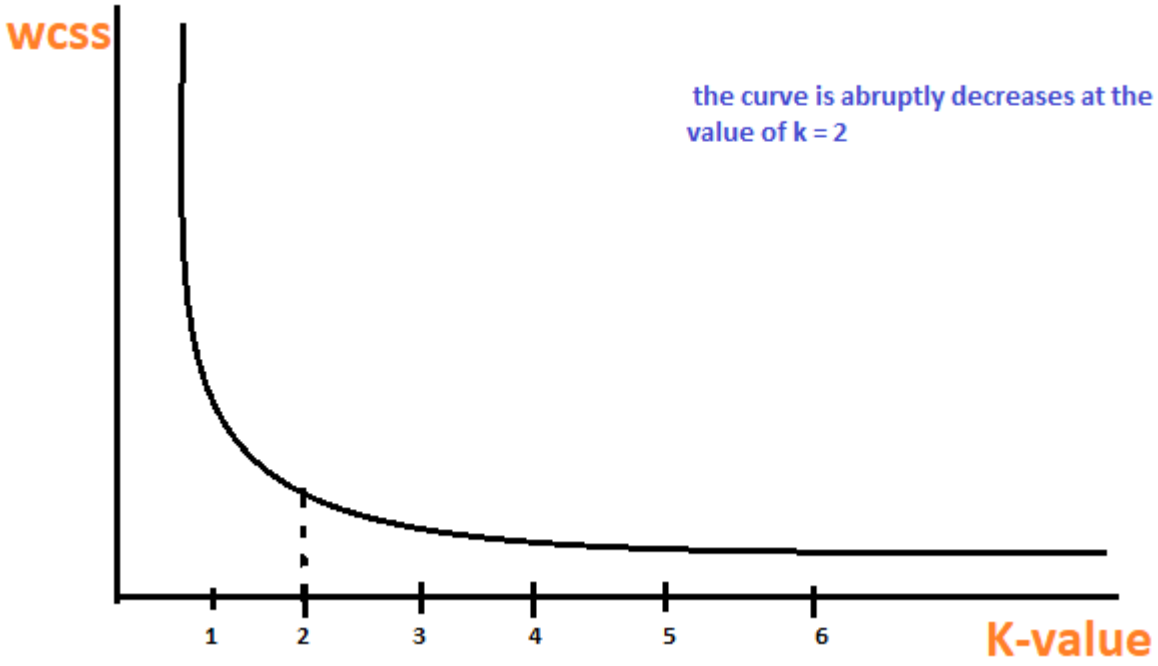
Elbow is one of the most famous methods by which you can select the right value of k and boost your model performance. We also perform the hyperparameter tuning to chose the best value of k. Let us see how this elbow method works.

It is an empirical method to find out the best value of k. it picks up the range of values and takes the best among them. It calculates the sum of the square of the points and calculates the average distance.



When the value of k is 1, the within-cluster sum of the square will be high. As the value of k increases, the within-cluster sum of square value will decrease.

Finally, we will plot a graph between k-values and the within-cluster sum of the square to get the k value. we will examine the graph carefully. At some point, our graph will decrease abruptly. That point will be considered as a value of k.



Code:

```
[1] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive

[2] import numpy as np
    import matplotlib.pyplot as plt
    np.random.seed(42)
    def euclidean_distance(x1, x2):
        return np.sqrt(np.sum((x1 - x2)**2))
    class KMeans():
        def __init__(self, K=5, max_iters=100, plot_steps=False):
            self.K = K
            self.max_iters = max_iters
            self.plot_steps = plot_steps
            # list of sample indices for each cluster
            self.clusters = [[] for _ in range(self.K)]
            # the centers (mean feature vector) for each cluster
            self.centroids = []
        def predict(self, X):
            self.X = X
            self.n_samples, self.n_features = X.shape
```

```
        # initialize
        random_sample_idxs = np.random.choice(self.n_samples, self.K, replace=False)
        self.centroids = [self.X[idx] for idx in random_sample_idxs]
        # Optimize clusters
        for _ in range(self.max_iters):
            # Assign samples to closest centroids (create clusters)
            self.clusters = self._create_clusters(self.centroids)
            if self.plot_steps:
                self.plot()
            # Calculate new centroids from the clusters
            centroids_old = self.centroids
            self.centroids = self._get_centroids(self.clusters)

            # check if clusters have changed
            if self._is_converged(centroids_old, self.centroids):
                break
            if self.plot_steps:
                self.plot()
        # Classify samples as the index of their clusters
        return self._get_cluster_labels(self.clusters)
    def _get_cluster_labels(self, clusters):
        # each sample will get the label of the cluster it was assigned to
        labels = np.empty(self.n_samples)
```

```
            for cluster_idx, cluster in enumerate(clusters):
                for sample_index in cluster:
                    labels[sample_index] = cluster_idx
            return labels
        def _create_clusters(self, centroids):
            # Assign the samples to the closest centroids to create clusters
            clusters = [[] for _ in range(self.K)]
            for idx, sample in enumerate(self.X):
                centroid_idx = self._closest_centroid(sample, centroids)
                clusters[centroid_idx].append(idx)
            return clusters
        def _closest_centroid(self, sample, centroids):
            # distance of the current sample to each centroid
            distances = [euclidean_distance(sample, point) for point in centroids]
            closest_index = np.argmin(distances)
            return closest_index
        def _get_centroids(self, clusters):
            # assign mean value of clusters to centroids
            centroids = np.zeros((self.K, self.n_features))
            for cluster_idx, cluster in enumerate(clusters):
                cluster_mean = np.mean(self.X[cluster], axis=0)
                centroids[cluster_idx] = cluster_mean
            return centroids
        def _is_converged(self, centroids_old, centroids):
```

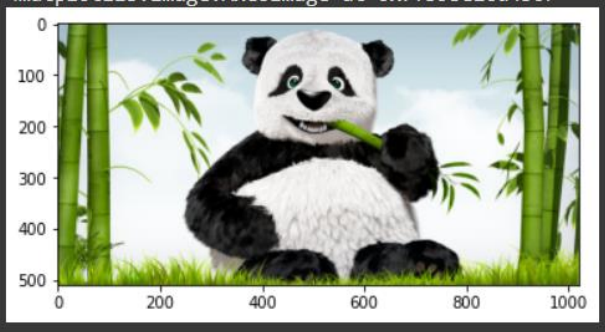
```
            cluster_mean = np.mean(self.X[cluster], axis=0)
            centroids[cluster_idx] = cluster_mean
        return centroids
    def _is_converged(self, centroids_old, centroids):
        # distances between each old and new centroids, fol all centroids
        distances = [euclidean_distance(centroids_old[i], centroids[i]) for i in range(self.K)]
        return sum(distances) == 0
    def plot(self):
        fig, ax = plt.subplots(figsize=(12, 8))
        for i, index in enumerate(self.clusters):
            point = self.X[index].T
            ax.scatter(*point)
        for point in self.centroids:
            ax.scatter(*point, marker="x", color='black', linewidth=2)
        plt.show()
    def cent(self):
        return self.centroids
import cv2
from google.colab.patches import cv2_imshow
image = cv2.imread("/content/drive/MyDrive/panda.jpg")
plt.figure(figsize=(6, 6))
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7f39be718650>
```

[2]


```
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(6, 6))
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7f39be1cd450>
```



```
[4] pixel_values = image.reshape((-1, 3))
    pixel_values = np.float32(pixel_values)
    print(pixel_values.shape)
```

```
(520200, 3)
```

```
[12] k = KMeans(K=5, max_iters=100)
     y_pred = k.predict(pixel_values)
     k.cent()
```

```
array([[102.91201019, 138.4924469 ,  23.10248566],
       [240.27835083, 243.19953918, 244.176651  ],
       [ 27.24910164,  31.98379135,  20.22505379],
       [159.95141602, 172.05422974, 121.10760498],
       [197.66937256, 211.25549316, 214.35362244]])
```

```
[13] centers = np.uint8(k.cent())
     centers
```

```
array([[102, 138,  23],
       [240, 243, 244],
       [ 27,  31,  20],
       [159, 172, 121],
       [197, 211, 214]], dtype=uint8)
```

```
[14] y_pred
```

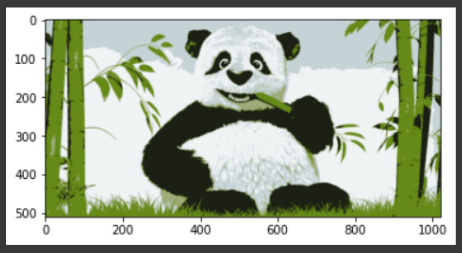
```
[14] y_pred
```

```
array([4., 4., 4., ..., 0., 0., 0.])
```

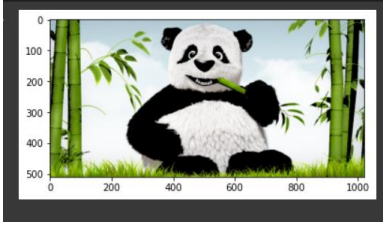
```
[15] y_pred = y_pred.astype(int)
     np.unique(y_pred)
```

```
array([0, 1, 2, 3, 4])
```

```
labels = y_pred.flatten()
segmented_image = centers[labels.flatten()]
segmented_image = segmented_image.reshape(image.shape)
plt.imshow(segmented_image)
plt.show()
```



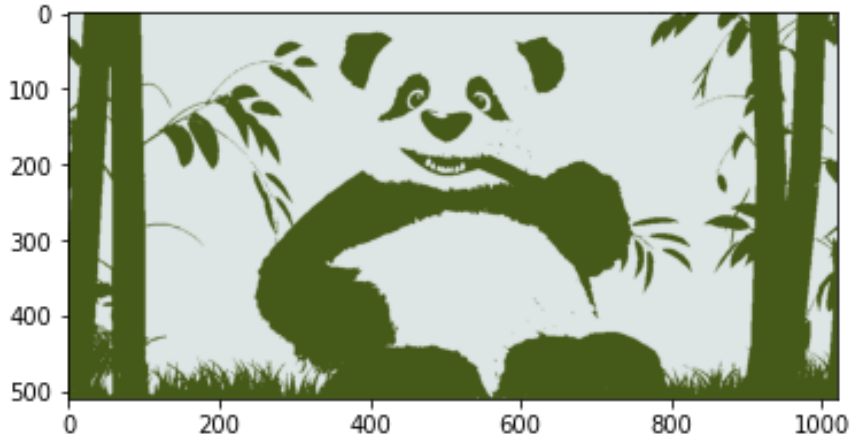

```
masked_image = np.copy(image)
masked_image = masked_image.reshape((-1, 3))
cluster = 2
masked_image[labels == cluster] = [0, 0, 0]
masked_image = masked_image.reshape(image.shape)
plt.imshow(masked_image)
plt.show()
```



Original image:

Segmented Image after 5 clusters:

K=2:


0
100
200
300
400
500
0
200
400
600
800
1000

K=3:


0
100
200
300
400
500
0
200
400
600
800
1000

K=5:


0
100
200
300
400
500
0
200
400
600
800
1000