E0-270: Machine Learning Assignment 2

Report

Rahul Dev Boipai

21514 , CSA

Introduction

K means clustering in an unsupervised technique in machine learning, which is capable of clustering unlabeled datasets very quickly and efficiently. The goal is to partition the data set into K given cluster, such that distance between points inside a cluster is small compared to the distance between points in different clusters. The idea is that a cluster represents a group of data points that are similar to each other and dissimilar to data points in other clusters. The distance between points captures the similarity, the smaller the distance similar are the points.

K means clustering is a popular choice for clustering tasks in many fields, including marketing, biology, and image processing. The algorithm is sensitive to the initial placement of the centroid may be struck in local optima and need many iterations with different initialization to get the best results.

Implementation

K means Clustering Algorithm:

1. Decide how many numbers of clusters(K) to partition the dataset into.
2. Initialize K centroids randomly selected from the dataset.
3. Assign each data point to the cluster, for which it has the least distance from the centroid.
4. Now update the centroid of each cluster by taking the mean of data points in the clusters.
5. Repeats steps 3 and 4 until the centroids no longer move significantly, or a maximum number of iterations is reached.

Once the algorithm has converged, each data point will belong to the cluster whose centroid is closest to it.

Now these centroids can be used for image segmentation and image compression by replacing each data point in a cluster with its centroid.

Code

Kmeans clustering in implemented inside KMeans class in function fit().

K centroid are initialized using np.random.choice() without replacement, so that all centroids be randomly picked and uniquely picked.

```
pixel ,colors = X.shape
        i = np.random.choice(pixel, self.num_clusters, replace=False) # pick without replacement
        self.cluster_centers = X[i]
```

Assign each data point to its closest centroid by calculating the distance between all centroids and selecting the cluster for which distance is minimum. Find the sq_sum using sum() function after calculating the square difference and then square root using np.sqrt(). Clusters are picked using np.argmin() which pick cluster for which distance is minimum.

```
 # Assign each sample to the closest prototype
            sq_sum = ((X[:, np.newaxis, :] - self.cluster_centers)**2).sum(axis=2)
            distances = np.sqrt(sq_sum)
            nearest_clusters = np.argmin(distances, axis=1)
```

Find new centroids by computing the mean of all points using mean() in each cluster and if there are no points in the cluster randomly assign a new centroid value using np. random.choice() from the dataset to that cluster.

```
# Update prototypes
            new_cluster_centers = np.zeros((self.num_clusters, colors))
            cluster = []
            for j in range(self.num_clusters):
                #find all the points in the cluster and its new centre
                cluster = X[nearest_clusters == j]
                if(cluster.size == 0):
                    #if cluster is empty then randomly assign new cluster
                    i = np.random.choice(pixel, 1)
                    new_cluster_centers[j] = X[i]
                else:
                    new_cluster_centers[j] = cluster.mean(axis=0)
```

Check whether centroids are moved or not and update accordingly. To check whether centroid values are updated or not compare each element by using np.alloclose() which take new centroids and old centroids and compare them for amount of change in value. If change is less than epsilon, stop updating else update to new centroid value.

```
# Check for convergence
            if np.allclose(self.cluster_centers, new_cluster_centers, atol=self.epsilon):
                break
            else:
                self.cluster_centers =  new_cluster_centers
```

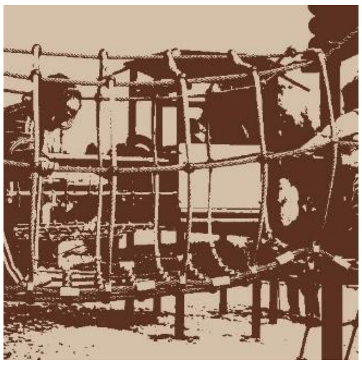
Replacing each point with its corresponding centroid in the cluster for image segmentation by simply changing all point in dataset with centroid of the cluster the point belong to.

```
def replace_with_cluster_centers(self, X: np.ndarray) -> np.ndarray:
        # Returns an ndarray of the same shape as X
        # Each row of the output is the cluster center closest to the corresponding row in X
        sq_sum = ((X[:, np.newaxis, :] - self.cluster_centers)**2).sum(axis=2)
        distances = np.sqrt(sq_sum)
        nearest_clusters = np.argmin(distances, axis=1)
        return self.cluster_centers[nearest_clusters]
```

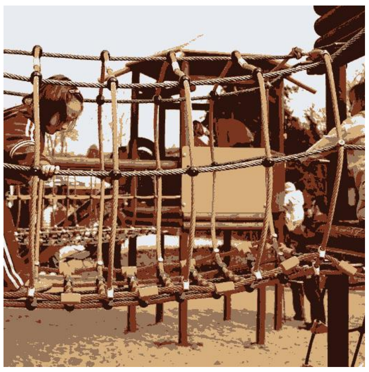
Results

Application of K means clustering algorithm to Image segmentation with different K values:

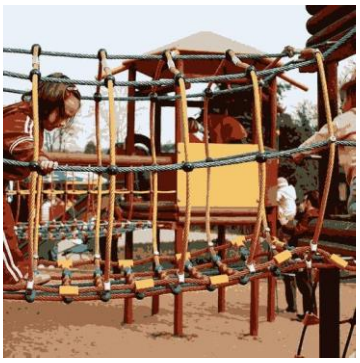
k = 2



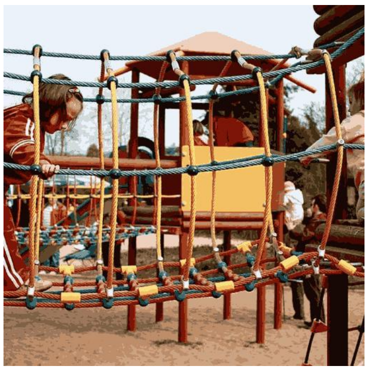
k = 5



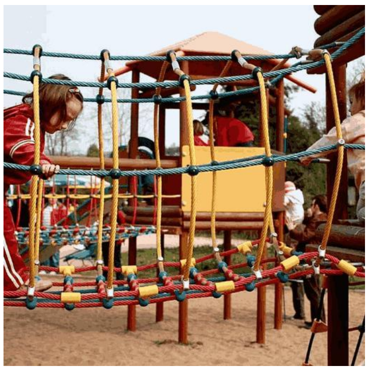
k = 10



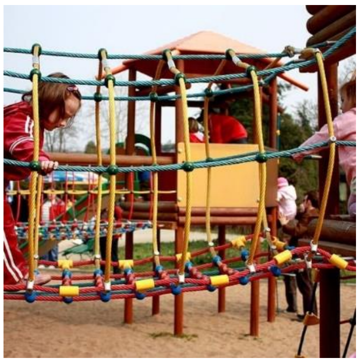
K = 20



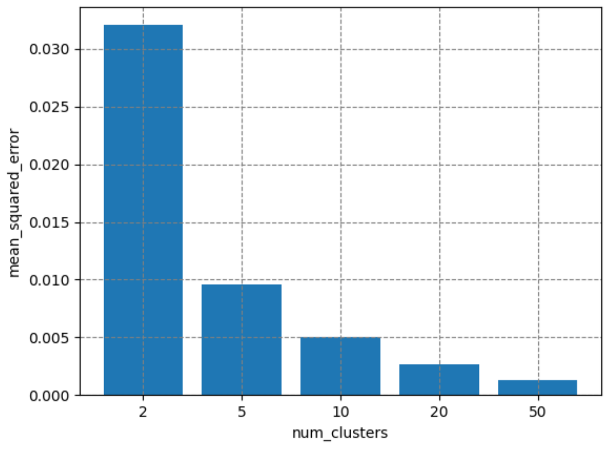
k = 50



Original



The plot of mean squared error as a function of the number of clusters



As the value of k is increased image quality gets close to the original image and the mean squared error also decreases as the number of clusters is increased.