# Image segmentation using K-Means and DBSCAN algorithms

Data Mining Assignment Task – 1

Submitted by: Danial Monachan
Alna Eldhose

## Abstract

This work discusses on how to perform image clustering with two different types clustering algorithms. The algorithms which will be discussed in the project will be K-Means and DBSCAN,
The overview of the project is to take the image as the input from the user and then train and fit the image data to perform clustering and then give the output of the clustered image.

## Problem statement

Design a program with which an input image can be clustered by selecting the appropriate clustering algorithm. The output of the segmentation will be the image divided into segments (cluster). The individual segments are to be represented by different colors.

An example can be seen in figure 1:

(a) Non-segmented image (b) Segmented image

(
Figure 1: Exemplary segmentation (k-means, k=3)

For both algorithms the following parameters should be available and arbitrarily selectable:

- k-means: $k$
- DBSCAN: $\epsilon$, $minPts$

## Data Requirements

Data requirement depends on the data which we are giving as test data for training the model.

Here we will be using Image data as input and train the model to classify the pixels and then plot the data in the clustered form.

The same algorithm mentioned in the source code can also be used to train and test for different datasets other than image such as Iris dataset for example.

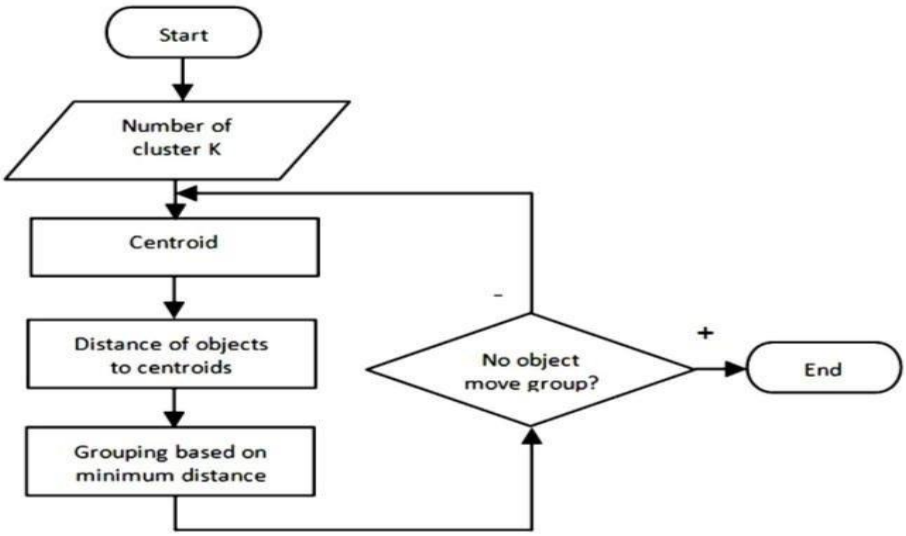
The scope of the project lies for only images and datasets not for videos.

## Tools used

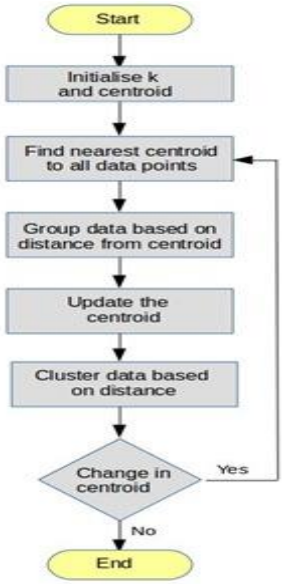
- Python programming language and frameworks such as NumPy, Pandas are used to build the whole model.
    - Jupyter is used as IDE.
    - For visualization of the plots Matplotlib is used.

## Model Flow Chart Representation

K- Means




DBSCAN



## K-means Algorithm

K-Means performs the division of objects into clusters that share similarities and are dissimilar to the objects belonging to another cluster.

Calculating the Euclidean Distance

```
# function to Calculate the Euclidean distance
# This will be used to calculate the the distance between the two data points
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))
```

**Steps**

- Choose the k number of clusters

We have created a class KMeans_algorithm which has the function to take the values of K for the algorithm as parameter.

```
class KMeans_algorithm():
    # creating the init function which defines the parameters so that if any parameter value is left


    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 = []
```

- Choose k random points from the data as centroids
  It is done with the help of the following function.

```
def _get_centroids(self, clusters):
    # Assigning 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
```

- Assign all the points to the cluster centroid which is close
  It is done with the help of the following function.

```
def _create_clusters(self, centroids):
    # Assigning 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 cent(self):
        return self.centroids
```

```
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 _is_converged(self, centroids_old, centroids):
    # distances between each old and new centroids, for all centroids
    distances = [euclidean_distance(centroids_old[i], centroids[i]) for i in range(self.K)]
    return sum(distances) == 0
```

Recompute the centroids of newly formed clusters

- Repeat until maximum iterations are reached or no new centroids are found.

## DBSCAN Algorithm

Based on a set of points, DBSCAN groups together points that are close to each other based on a distance measurement (usually Euclidean distance) and a minimum number of points. It also marks as outliers the points that are in low-density regions.

Unlike K means DBSCAN can find arbitrarily -shaped clusters.

**Parameters:**

The DBSCAN algorithm basically requires 2 parameters:

**eps**: specifies how close points should be to each other to be considered a part of a cluster. It means that if the distance between two points is lower or equal to this value (eps), these points are considered neighbours.

**minPoints**: the minimum number of points to form a dense region. For example, if we set the minPoints parameter as 5, then we need at least 5 points to form a dense region.

### Steps

We are using two functions here one is to calculate the maximum distance between the two data points using the function MaximumDistance

```
def MaximumDistance(P,Q):
    intermediateValues = []
    for i in range(len(P[2])):
        intermediateValues.append(abs(Q[2][i]-P[2][i]))
    return max(intermediateValues)
```

And FindNeighbours function to find the neighbors of the Data point it will help us in identifying whether the data points is a core point or a non-core point.

```
#Finds all neighbor points for a chosen point
def FindNeighbours(Point, Points, eps):
    tempNeighbours = []
    for y in range(len(Points)):
        for x in range(len(Points[0])):
            if MaximumDistance(Point, Points[y][x]) <= eps:
                    tempNeighbours.append(Points[y][x])
#Note: use Max Distance if required
    return tempNeighbours
```

A function dbscan is create which takes the data set, min_points and Epsilon as the input parameters

```
def dbscan(vectors: list, minpts: int, epsilon: int) -> list:
```

The following segment of code helps in identifying the core values and non-core values in order to specify each data point as a part of cluster or to disregard them as noise.

```
#DBSCAN clustering
clusterCounter = 0
for y in range(len(vectors)):
    for x in range(len(vectors[0])):
        if pointsArray[y][x][-1] != "Undefined":
            continue
        # finding neighbours for core pints and assigning noises
        Neighbours = FindNeighbours(pointsArray[y][x], pointsArray, epsilon)
        if len(Neighbours) < minpts:
            pointsArray[y][x][-1] = "Noise"
            continue

        clusterCounter = clusterCounter + 1
        pointsArray[y][x][-1] = str(clusterCounter)
        if pointsArray[y][x] in Neighbours:
            Neighbours.remove(pointsArray[y][x])

        for innerPoint in Neighbours:
            if innerPoint[-1] == "Noise":
                pointsArray[innerPoint[0]][innerPoint[1]][-1] = str(clusterCounter)
            if innerPoint[-1] != "Undefined":
                continue
            pointsArray[innerPoint[0]][innerPoint[1]][-1] = str(clusterCounter)
            NeighboursInner = FindNeighbours(innerPoint, pointsArray, epsilon)
            if len(NeighboursInner) >= minpts:
                Neighbours.append(NeighboursInner)
```

The following code allows us to get the clusters distinguished separately.

```
# code to get the distinct clusters getting created
clusterNumbers = []
for y in range(len(vectors)):
    for x in range(len(vectors[0])):
        if pointsArray[y][x][-1] not in clusterNumbers:
            clusterNumbers.append(pointsArray[y][x][-1])
```

## Conclusion

The above mentioned Algorithms K-Means and DBSCAN are able to successfully distinguish between clusters and produce the clustered image as output.

## References


- https://towardsdatascience.com/create-your-own-k-means-clustering-algorithm-in-python-d7d4c9077670
- https://medium.com/analytics-vidhya/image-segmentation-using-k-means-clustering-from-scratch-1545c896e38e
- https://becominghuman.ai/dbscan-clustering-algorithm-implementation-from-scratch-python-9950af5eed97
- https://www.mathworks.com/discovery/image-segmentation.html