Segmentation algorithms

Ananth Bharathwaj T A (21BCE1079)
Sankari Karthik (21BCE1396)

Code:

K-means:

```
# imports
import numpy as np # for handling image data
from sklearn.datasets import make_blobs # to generate data points for some
visualizations
from collections import defaultdict  # to store cluster information
import matplotlib.pyplot as plt  # for plotting
import random  # to initialize randomly generated clusters
import cv2 # converts image to pixel values

# select k centroids
def get_initial_centroids(X, k):
    """
    Function selects k random data points from X
    """
    initial_centroids = set(np.random.choice(np.arange(len(X)), size=k))
    while len(initial_centroids) != k:
        initial_centroids.add(random.randrange(0, len(X)))
    initial_centroids = [tuple(X[id]) for id in initial_centroids]
    return np.array(initial_centroids)

def get_euclidean_matrix(A_matrix, B_matrix):
    A_square = np.reshape(np.sum(A_matrix * A_matrix, axis=1),
(A_matrix.shape[0], 1))
    B_square = np.reshape(np.sum(B_matrix * B_matrix, axis=1), (1,
B_matrix.shape[0]))
    AB = A_matrix @ B_matrix.T

    C = -2 * AB + B_square + A_square
    return np.sqrt(abs(C))

def get_clusters(X, centroids):
```

```
    """
    Returns a mapping of clusters to points that belong to that cluster
    """
    clusters = defaultdict(list)
    distance_matrix = get_euclidean_matrix(X, centroids)
    closest_cluster_ids = np.argmin(distance_matrix, axis=1)
    for i, cluster_id in enumerate(closest_cluster_ids):
        clusters[cluster_id].append(X[i])
    return clusters

def check_clusters_converged(old_centroids, new_centroids, threshold):
    distances_between_old_and_new_centroids =
get_euclidean_matrix(old_centroids, new_centroids)
    return np.max(distances_between_old_and_new_centroids.diagonal()) <=
threshold

def k_means(X, k, threshold=0.5):
    new_centroids = get_initial_centroids(X, k)

    has_converged = False
    while not has_converged:
        previous_centroids = new_centroids
        previous_clusters = get_clusters(X, previous_centroids)

        new_centroids = np.array([np.mean(c, axis=0) for c in
previous_clusters.values()])

        has_converged = check_clusters_converged(previous_centroids,
new_centroids, threshold)
    return new_centroids

def visualization():
    k = 4
    X, _ = make_blobs(n_samples=1000, n_features=2, centers=k)

    centroids = k_means(X, k)
    clusters = get_clusters(X, centroids)

    plt.rcParams['figure.figsize'] = [10, 5]
    for centroid, points in clusters.items():
```

```
        points = np.array(points)
        centroid = np.mean(points, axis=0)
        plt.scatter(points[:, 0], points[:, 1], marker='o')
        plt.grid()
        plt.scatter(centroid[0], centroid[1], marker='x', color="red")

    plt.show()

def get_segmented_image(image, k):
    image = cv2.imread(image)
    height, width, depth = image.shape

    X = np.reshape(image, (height * width, depth))
    X = np.array(X, dtype=np.int32)

    centroids = k_means(X, k)
    distance_matrix = get_euclidean_matrix(X, centroids)
    closest_cluster_ids = np.argmin(distance_matrix, axis=1)

    X_segmented = centroids[closest_cluster_ids]
    X_segmented = np.array(X_segmented, dtype=np.uint8)

    segmented_image = np.reshape(X_segmented, (height, width, depth))
    plt.imshow(cv2.cvtColor(segmented_image, cv2.COLOR_BGR2RGB))
    plt.savefig('output/30.jpg')
    plt.show()


# visualization()
get_segmented_image("../datasets/resized/30.jpg", 10)
```

Mean shift:

```
import numpy as np
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import cv2

def euclidean_distance(x1, x2):
    """Generates euclidean distance between two points
```

```
    Args:
        x1 (int): Point 1
        x2 (int): Point 2

    Returns:
        float: Euclidean distance between the Points 1, 2
    """
    return np.sqrt(np.sum((x1 - x2)**2))

def gaussian_kernel(distance, bandwidth):
    """Generates gaussian kernel

    Args:
        distance (int): distance between the cluster centre and point in
question
        bandwidth (int): Radius of the cluster

    Returns:
        float: Gaussian kernel
    """
    return (1 / (bandwidth * np.sqrt(2 * np.pi))) * np.exp(-0.5 *
((distance / bandwidth))**2)

def get_new_centroid(x, data, bandwidth):
    """Generates new centroid, based on kernel density estimation

    Args:
        x (int): Point in question
        data (List[int]): Image as an array
        bandwidth (float): Bandwidth of the cluster

    Returns:
        _type_: _description_
    """
    weights = gaussian_kernel(np.linalg.norm(data - x, axis=1), bandwidth)
    return np.sum(data * weights[:, np.newaxis], axis=0) / np.sum(weights)

def mean_shift(data, bandwidth=2, convergence_threshold=0.001,
max_iterations=100):
```

```
    """The main mean shift algorithm

    Args:
        data (List[int]): The image as an array
        bandwidth (int, optional): The width of each cluster. Bigger the
bandwidth, lesser clusters generated. Defaults to 2.
        convergence_threshold (float, optional): The value below which two
clusters are converged. Defaults to 0.001.
        max_iterations (int, optional): The number of iterastions that
clustering is run. Defaults to 100.

    Returns:
        _type_: _description_
    """
    shifted_points = data.copy()
    for i, point in enumerate(data):
        old_point = point
        shift = np.inf
        iteration = 0
        while shift > convergence_threshold and iteration <
max_iterations:
            new_point = get_new_centroid(old_point, data, bandwidth)
            shift = euclidean_distance(new_point, old_point)
            old_point = new_point
            iteration += 1
        shifted_points[i] = old_point
        if (i+1)%1000==0: print(f"Processed {i+1}/{len(data)} points")
    return shifted_points

def visualization():
    """Visualizes the clustering algorithm
    """
    X, _ = make_blobs(n_samples=1000, n_features=2, centers=4)

    shifted_points = mean_shift(X)

    plt.rcParams['figure.figsize'] = [10, 5]
    plt.scatter(X[:, 0], X[:, 1], marker='o', label='Original Data')
    plt.scatter(shifted_points[:, 0], shifted_points[:, 1], marker='x',
color="red", label='Cluster Centers')
```

```
    plt.legend()
    plt.grid()
    plt.show()

def get_segmented_image(image,
bandwidth=5,output_path='segmented_image.png'):
    """Generates the segmented image from a 2D array

    Args:
        image (List[int]): The 2D array to be converted into an image
        bandwidth (int, optional): The bandwidth value used in the
clustering. Defaults to 5.
        output_path (str, optional): The output image filepath. Include
filename in path as well. Defaults to 'segmented_image.png'.
    """
    image = cv2.imread(image)
    height, width, depth = image.shape

    X = np.reshape(image, (height * width, depth)).astype(np.float64)

    shifted_points = mean_shift(X, bandwidth=bandwidth)

    segmented_image = np.reshape(shifted_points.astype(np.uint8), (height,
width, depth))
    plt.imshow(cv2.cvtColor(segmented_image, cv2.COLOR_BGR2RGB))
    plt.savefig(output_path)
    # plt.show()

import os
for image in os.listdir('datasets/resized'):
    path = 'datasets/resized/' + image
    if image in os.listdir('meanshift/output/original'):
        print("Already done with", image)
        continue #['01.jpg', '02.jpg', '03.jpg', '04.jpg', '05.jpg',
'06.jpg', '07.jpg', '08.jpg', '09.jpg', '10.jpg', '11.jpg', '12.jpg',
'13.jpg', '14.jpg', '15.jpg']
    print(path)
    print("Processing", image)
    get_segmented_image(path, bandwidth=15,
output_path=f'meanshift/output/original/{image}')
```

```
    print("Done with", image)
```

Output:

(Complete dataset is 30 images, here is a selection of 5 images. All 30 can be found in our GitHub Repository in the `results` folder)

Original

Benchmark Segmentation

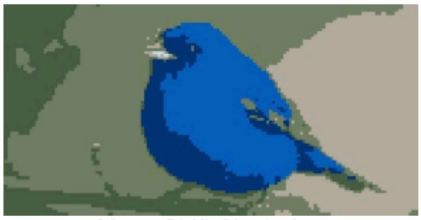
K-Means Clustering

Mean Shift Clustering

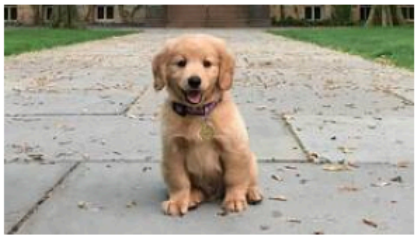
Original

Benchmark Segmentation

K-Means Clustering



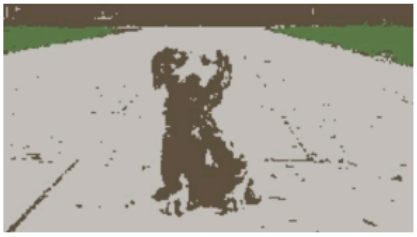
Mean Shift Clustering



Original

Benchmark Segmentation

K-Means Clustering



Mean Shift Clustering

Original

Benchmark Segmentation

K-Means Clustering

Mean Shift Clustering

Original

Benchmark Segmentation

K-Means Clustering

Mean Shift Clustering