hw5

November 15, 2019

1 Computer Vision

2 Jacobs University Bremen

3 Fall 2019

4 Homework 5

This notebook includes both coding and written questions. Please hand in this notebook file with all the outputs and your answers to the written questions.

This assignment covers K-Means and HAC methods for clustering and image segmentation.

```
In [1]: # Setup
        from time import time
        import numpy as np
        import matplotlib.pyplot as plt
        from matplotlib import rc
        from skimage import io

        from __future__ import print_function

        %matplotlib inline
        plt.rcParams['figure.figsize'] = (15.0, 12.0) # set default size of plots
        plt.rcParams['image.interpolation'] = 'nearest'
        plt.rcParams['image.cmap'] = 'gray'

        # for auto-reloading extenrnal modules
        %load_ext autoreload
        %autoreload 2
```

4.1 Introduction

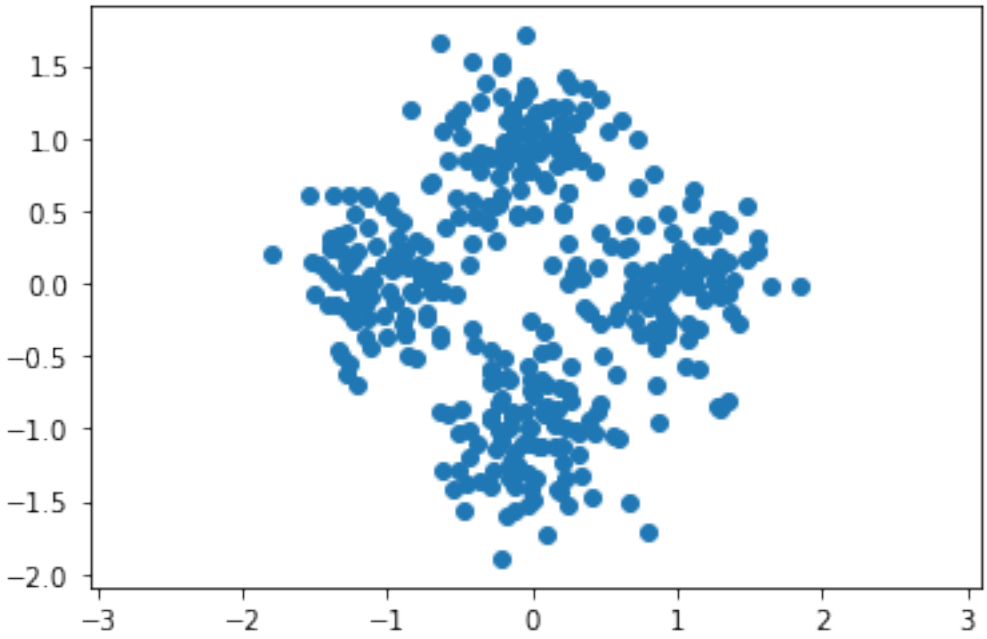
In this assignment, you will use clustering algorithms to segment images. You will then use these segmentations to identify foreground and background objects.

Your assignment will involve the following subtasks: - **Clustering algorithms**: Implement K-Means clustering and Hierarchical Agglomerative Clustering. - **Pixel-level features**: Implement a feature vector that combines color and position information and implement feature normalization.

- **Quantitative Evaluation**: Evaluate segmentation algorithms with a variety of parameter settings by comparing your computed segmentations against a dataset of ground-truth segmentations.

4.2 1 Clustering Algorithms (40 points)

In [2]:
```
# Generate random data points for clustering

# Set seed for consistency
np.random.seed(0)

# Cluster 1
mean1 = [-1, 0]
cov1 = [[0.1, 0], [0, 0.1]]
X1 = np.random.multivariate_normal(mean1, cov1, 100)

# Cluster 2
mean2 = [0, 1]
cov2 = [[0.1, 0], [0, 0.1]]
X2 = np.random.multivariate_normal(mean2, cov2, 100)

# Cluster 3
mean3 = [1, 0]
cov3 = [[0.1, 0], [0, 0.1]]
X3 = np.random.multivariate_normal(mean3, cov3, 100)

# Cluster 4
mean4 = [0, -1]
cov4 = [[0.1, 0], [0, 0.1]]
X4 = np.random.multivariate_normal(mean4, cov4, 100)

# Merge two sets of data points
X = np.concatenate((X1, X2, X3, X4))

# Plot data points
plt.scatter(X[:, 0], X[:, 1])
plt.axis('equal')
plt.show()
```

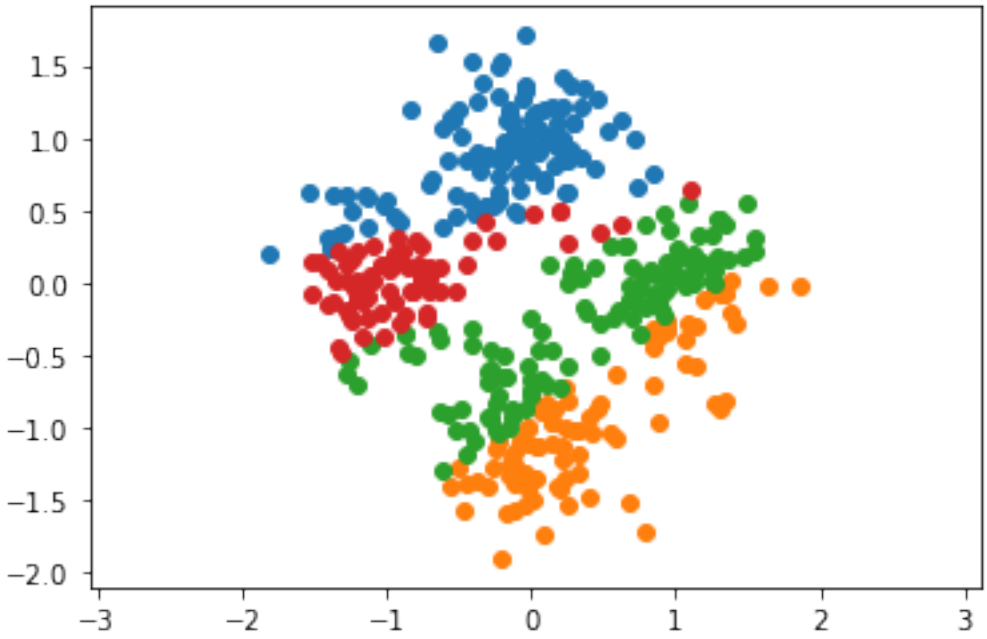


4.2.1 1.1 K-Means Clustering (20 points)

As discussed in class, K-Means is one of the most popular clustering algorithms. We have provided skeleton code for K-Means clustering in the file `segmentation.py`. Your first task is to finish implementing **`kmeans`** in `segmentation.py`. This version uses nested for loops to assign points to the closest centroid and compute a new mean for each cluster.

```
In [3]: from segmentation import kmeans

        np.random.seed(0)
        start = time()
        assignments = kmeans(X, 4)
        end = time()

        kmeans_runtime = end - start

        print("kmeans running time: %f seconds." % kmeans_runtime)

        for i in range(4):
            cluster_i = X[assignments==i]
            plt.scatter(cluster_i[:, 0], cluster_i[:, 1])

        plt.axis('equal')
        plt.show()
```

```
kmeans running time: 0.046874 seconds.
```



We can use numpy functions and broadcasting to make K-Means faster. Implement **kmeans_fast** in segmentation.py. This should run at least 10 times faster than the previous implementation.

```
In [4]: from segmentation import kmeans_fast

        np.random.seed(0)
        start = time()
        assignments = kmeans_fast(X, 4)
        end = time()

        kmeans_fast_runtime = end - start
        print("kmeans running time: %f seconds." % kmeans_fast_runtime)
        print("%f times faster!" % (kmeans_runtime / kmeans_fast_runtime))

        for i in range(4):
            cluster_i = X[assignments==i]
            plt.scatter(cluster_i[:, 0], cluster_i[:, 1])

        plt.axis('equal')
        plt.show()
```

```
kmeans running time: 0.046388 seconds.
1.010480 times faster!
```