Filippo Ficarra: Lab 5 - Image Segmentation

fficarra@student.ethz.ch, 22–938–062.

23/11/2023 - 16:10h

1 Mean-Shift Algorithm

The Mean-Shift Algorithm aims to find modes in a given dataset. It is used to perform unsupervised clustering. The principal benefit of this algorithm is that we don't have to impose a fixed number of cluster a priori but it will just depend on a variable called "bandwidth".

1.1 Distance function

The first function completed is the distance function. This function it will further used in the gaussian kernel to compute the weight of each point. The following function simply computes the eucledian distance of the point $\boldsymbol{x}$ and all points in $\boldsymbol{X}$.

Listing 1: Distance function

```
1  def distance(x, X):
2    return np.sqrt(np.sum((x - X) ** 2, axis=1))
```

1.2 Gaussian Kernel

The Gaussian kernel is used since it provides continuous and smooth distributions and are suitable to extract information also in noisy data.

Give a bandwith, the gaussian kernel is simply the following:

Listing 2: Gaussian Kernel

```
1  def gaussian(dist, bandwidth):
2    K = np.exp(-(dist ** 2) / (2 * bandwidth ** 2))
3    return K
```

1.3 Update point

Updating the point is the part in where the algorithm start to find the clusters. The update of a point $y^{(t+1)}$ is computed as following.

$$y_j^{(t)} \leftarrow \frac{\sum_{i=1}^{n} X_i w_i(j)}{\sum_{i=1}^{n} w_i(j)}$$

where the $w_i(j)$ are the weights for the j-th point computed with the gaussian function above.

1.4 Iterate the algorithm

The final step is to compute the update for each point and compute it for a number of iteration equal to 20 and then compute the centroids:

Listing 3: Mean-shift

```
def meanshift_step(X, bandwidth=1):
  y = X.copy()
  for i in range(X.shape[0]):
      dist = distance(X[i], X)
      weight = gaussian(dist, bandwidth)
      y[i] = update_point(weight, X)
  return y

centroids, labels = np.unique((X / 4).round(), return_inverse=True, axis=0)
```

2 Results

First, the number of colors was 24 so we were constrained with the number of clusters, if the results gave more clusters than the number of colors we would have an error when running. This has been noticed when using different values of the bandwidth in the gaussian kernels. Due to this some of the runs may not succed. Larger the bandwith fewer the number of clusters that the algorithm will discover, since the bandwidth is the neighbourhood in which the gaussian is "searching". The value that gave problem were 1, 2, and 2.5, while all the others generated less cluster than the number of colors. To solve the problem of too many centroids I just took the 24 bigger cluster and merge the other clusters to the nearest. Another method could be do K-means on the centroids with k=24. The result are shown in the images below, As you can see, aggregating large number of clusters could give very bad segmentation, like in bandwidth = 1.

Figure 1: bandwidth = 3, clusters = 15

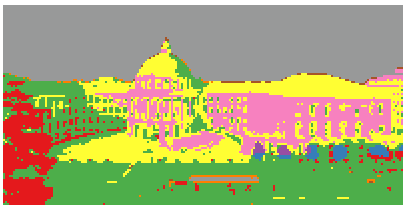

Figure 2: bandwidth = 4, clusters = 9

Figure 3: bandwidth = 5, clusters = 5

Figure 4: bandwidth = 6, clusters = 4

Figure 5: bandwidth = 7, clusters = 3

Figure 6: bandwidth = 1, clusters = 24, original clusters = 282

Figure 7: bandwidth = 2, clusters = 24, original clusters = 44

Figure 8: bandwidth = 2.5, clusters = 24, original clusters = 25