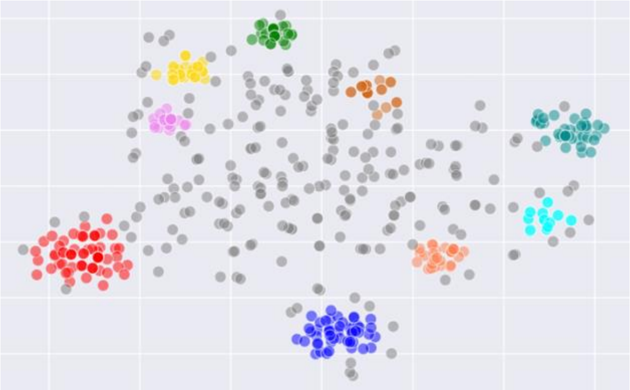
Data Science with Python – Assignment #4

Assignment description

This assignment deals with unsupervised data processing – clustering. In the class we learned about KMeans: a procedure that groups feature vectors into K clusters. Here we will implement its enhanced version, which in practice is more useful for multiple real-world scenarios.

Task: Density-based Clustering (DBC)

KMeans clustering algorithms has two main drawbacks: (1) it requires a known number of clusters as a parameter, and (2) it assigns all data instances into clusters, even if some do not make a natural candidate to enter any cluster – instances that are outliers.



Consider this 2D illustration of feature vectors. There are some groupings that make dense clusters, and many other instances that should be considered as "noise" or outliers: those are not clustered.

Our clustering algorithm will: (1) discover the number of clusters, (2) tolerate outliers, not insisting to assign them to any cluster. In the illustration, groups in color denote outcome clusters, while gray points denote un-clustered outliers.

In this assignment we are clustering images. Images can be represented by a simple pixel representation (e.g., 128X128), where each pixel is assigned a three-valued color based on its RGB values. We will be using a more sophisticated image representation, based on advanced NN architecture – ResNet50. The representations are feature vectors, they were pre-generated and are given to you in this assignment.

Image-based feature vectors (also known as "embeddings") reliably represent the image, in a way that vectors of similar images will be closer in semantic space than those of dis-similar images. As a concrete example, feature vectors of the two images on the right will be more similar to each other than either of them is to the image on the left.

Your task is to implement a density-based clustering algorithm, inspired by KMeans, satisfying the two properties mentioned above: unknown number of clusters and tolerating outliers.

Below is a pseudo-code of the algorithm:

```
input: E (e_1, e_2, …, e_n)  # image feature vectors
input: min_similarity # minimal cosine similarity threshold for an element to enter a cluster
input: min_cluster_size # minimum elements for a group to be considered a cluster

C = {}
while convergence criteria are not met do:
   for each element e_i in E do:
      if max similarity of e_i to any existing cluster centroid > min_similarity then:
         re-assign e_i to its most similar cluster c
         re-calculate centroids of c and of the previous cluster of e_i (if exists)
      else:
         create a new cluster and assign e_i to it
         set the centroid of the new cluster to be e_i
         add the new cluster to C

## elements assigned to clusters of total size < min_cluster_size are considered outliers
return each c in C of size exceeding min_cluster_size
```

Note that `min_cluster_size` is given to you in the config file. The `min_similarity` threshold should be set by you in the code, in a way that optimizes the performance of your clustering module. Once you find the threshold, set it (hardcoded) in your code – it would work for any other image set.

Attached to this assignment:

(1) a folder with flower images so that you can explore them and see how your solution works

(2) a file with images' feature vectors – a serialized pickle file, you have an example using it in `utils.py`

(3) a file with actual cluster assignments (ground truth) as annotated by humans; images that do not fall into any cluster (outliers) are annotated with -1

Additionally, a function evaluating your solution against the true annotation is implemented in the assignment. Please note that it expects input format identical to the output of your clustering module:

```
{<cluster number>: [array of image file names]}, for example:
{0: ['00_001.png', '00_002.png', '00_027.png', …],
1: ['05_016.png', '05_017.png', …],
…}
```

A clustering outcome that has about 550 instances in common with the true clusters, and rand score higher than 0.90, can be considered a good solution. Note that a very small number of instances in common can result in a high rand score, yet – that would not qualify as a good solution. In particular, an empty clustering result would show a rand score of 1.0.

Comments:

(1) Your code should work seamlessly on any input of the given structure. The submission should be ready to be tested on different image datasets, by changing only the config file.

(2) Make sure your code runtime doesn't exceed one minute (invocation → results are printed).

(3) The assignment should be implemented in PyCharm (similarly to assignments #1 and #3).

(4) Do not make any changes to the `main()` function.

Submission

Submit a single zip file – assignment4_ xxxxxxxxx_xxxxxxxxx.zip , where "xxxxxxxxx" stands for a student id. Please specify two student ids (your and your partner's). It should include your solution for the task: a single `main.`py file with your solution for the task.

Grading criteria include: correctness (the major part), code design, readability and documentation.

Good Luck!