Practice 2: Labelling
Part 2: KNN and Shape
Artificial Intelligence Course

Computer Science Department
Universitat Autònoma de Barcelona

1 Introduction

In this Part 2 of the practice, we will continue to work on solving the image labeling problem. We saw how to implement k-means to label the color of clothes. In this second part, we will focus on labeling the clothes' shape, as seen in figure 1. Therefore, now we focus on the implementation of the K-NN algorithm:

K-NN (`k_nn`) Supervised classification method that will be used to assign clothing type labels.

In this part of the practice, we will solve the problem of automatically assigning clothing labels to images. We will use 8 types of garments.

Figure 1: Global goal of the project

2 Required files

We will use the same files we already downloaded for Part 1, but we will focus on the `KNN.py` and `TestCases_knn.py` files.

Since KNN is a supervised learning algorithm, we need the images in the `Images` folder, and the learning and test sets we have in it:

1. `images`: Folder containing the image sets we will use.

   In this folder you will find:

   (a) `Test`: Folder with the set of images that we want to tag and of which we have no information, this will be the testing set (test set).

   (b) `Train`: Folder with the set of images that we can use as a training set. For all of them (train set) we have the class information in the file `Class_labels.cvs`.

   (c) `gt.json`: File containing the information of the images in the *Ground-Truth*.

   (d) `gt_reduced`: File with complementary information about a subset of images of the training set.

2. `test`: Folder containing the set of files needed to perform the tests requested in the test (you do not need to use them in the code, the test functions are automatically loaded in `setUp`).

3. `utils.py`: File containing a number of functions needed to convert color images to other spaces, mainly converting them to grayscale `rgb2gray()` and getting the 11 basic colors `get_color_test()`.

4. `Kmeans.py`: File where you will program the functions needed to implement K-means to extract the predominant colors.

5. `TestCases_kmeans.py`: A file you can use to check whether the functions you are programming in the `Kmeans.py` file give the expected result.

6. `KNN.py`: File where you will program the functions needed to implement K-NN to label the clothe name.

7. `TestCases_knn.py`: A file you can use to check whether the functions you are programming in the `KNN.py` file give the expected result.

8. `my_labeling.py`: File where you will combine the two tagging methods and your enhancements to get the final tagging of the images (third part of the practice).

9. `utils_data.py`: Contains a series of functions you can use for the visualization of results (third part of the practice).

3 What do we have to program?

In this second part of the practice, you will develop the KNN classification algorithm. To carry out the supervised classification of the clothes' shape of the images, we will use as features the pixels of the image after converting it to grayscale. To implement the KNN you have to implement the following four functions:

`_init_train`: Function that ensures the variable `train_data` of class KNN, that contains our training set, has the `float` format, afterward it extracts their features. Thus, `train_data` will have a size $N \times 4800$, where N is the number of images in the training set, and 4800 the number of pixels per each image 80×60.

```
def _init_train(self,train_data):
    """
    initializes the train data
    :param train_data: PxMxN matrix corresponding to P grayscale
       images
    :return: assigns the train set to the matrix self.train_data
       shaped as PxD
    (P points in a D dimensional space)
    """
```

`get_k_neighbours`: Function that takes as input the testing set we want to label (test_data) and do the following:

1. Resize images in the same way we have done with the training set.
2. Computes the distance between the features of `test_data` and those of the `train_data`.
3. Stores at the variable of class `self.neighbors` the labels of the j closest images for each sample in the test.

**** Hint: Calculating distances can have a hight computational cost if done in a loop. To do it, we recommend you use the `cdist` function in the `scipy.spatial.distance` library, which will allow you to do this calculation much faster.*

```
def get_k_neighbours(self, test_data, k):
    """
    given a test_data matrix calculates de k nearest neighbours at
       each point (row) of test_data on self.neighbors
    :param test_data:   array that has to be shaped to a NxD matrix (
       N points in a D dimensional space)
    :param k:  the number of neighbors to look at
    :return: the matrix self.neighbors is created (NxK). the ij-th
       entry is the j-th nearest train point to the i-th test point
    """
```

`get_class`: Function that checks which label has appeared most often in the variable of class `neighbors` for each image in the test set, `test_data`, and returns an `array` with