

PROJECT 2 **Labelling**

SS: Session of Support
Project 2 - Part 2

Artificial Intelligence

2023-2024
Universitat Autònoma de Barcelona

1. We remember...

In this second part, we continue working to solve the **image labeling problem**.

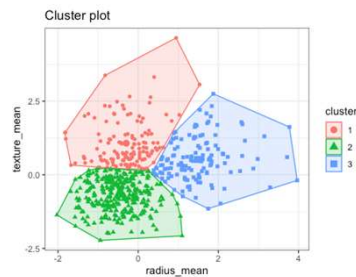
We saw how **k-means** was implemented to **label the color** of the clothing.

Now we will focus on **labeling the shape** of the clothing by implementing the **KNN algorithm**.

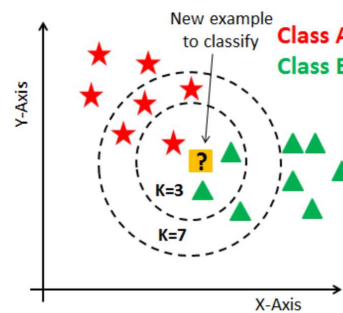


Supervised
classification method

K-means









KNN



2. Required files

Remember that for this second part of the practice you will need:

1.  **Images:** Folder containing the set of images we will use.
 - A.  **gt.json:** File containing information about the class of images to train (Train).
 - B.  **Train:** Folder with the set of images we will use as the **training set**. Information about which class each image belongs to can be found in the file **gt.json**.
 - C.  **Test:** Carpeta amb el set d'imatges que volem etiquetar (**test set**).
2.  **Test:** Folder containing the files necessary to perform the tests in the test scripts.
 **test_cases_knn.pkl**
3. **utils.py:** It contains already implemented functions that you can use.
4. **utils_data.py:** It contains a series of functions necessary for opening and processing images
5. **KNN.py:** File where you will program the necessary functions to implement K-NN for clothing labeling.
6. **TestCases_knn.py:** File with which you verify if the functions in KNN.py give the expected result.

2. KNN Implementation

To implement the KNN, you will need to program the following four functions:

1. `_init_train`
2. `get_k_neighbours`
3. `get_class`
4. Predict

Clarification: To perform supervised classification of the shape of the images, we will use the **pixels of the image** after transforming it into **grayscale** as **features**.

2.1 _init_train

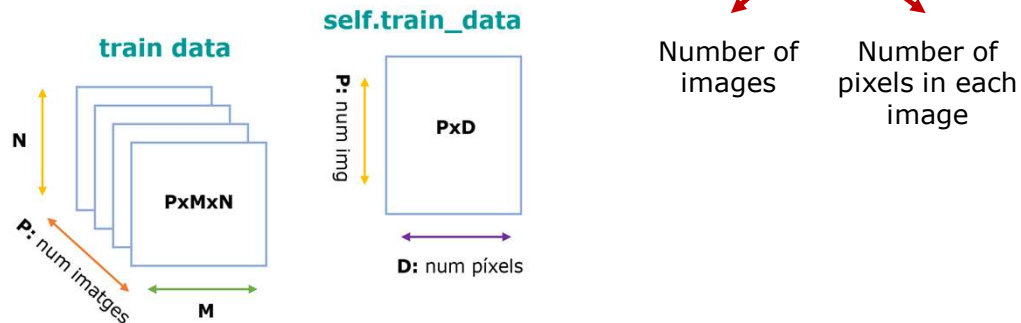
Function that initializes the training data.

```
def _init_train (self, train_data):  
    implementation
```

Parameter: matrix **PxMxN** corresponding to
P grayscale images

Implementation:

- Ensure that train_data has **float** format.
- Extract its **features**.
- Assign the training set to the matrix **self.train_data** with shape **PxD**.
(P points in D-dimensional space)



2.2 get_k_neighbours

Function that, given a test_data matrix, calculates the k nearest neighbors to each point (row) of test_data and stores them in self.neighbors.

```
def get_k_neighbours (self, test_data, k):
```

implementation

Parameters:

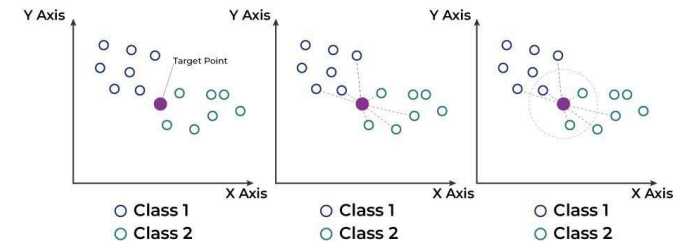
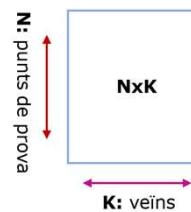
- **test_data:** matrix that needs to be adjusted to an Nx D matrix (N points in a D-dimensional space)
- **k:** number of neighbors to analyze

Implementation:

- Change the **dimensions** of the images (similarly to train_data).
- Calculate **distance** between the samples of test_data and train_data.
- Store in the variable self.neighbors the **K labels** of the **nearest images** for each sample of the test.
- Create the **matrix** self.neighbors (NxK).

*Tip: We recommend using the **cdist** function from the [scipy.spatial.distance](#) library to avoid high computational cost.

self.neighbors



2.3 get_class

Funció que comprova quina és l'etiqueta que més vegades ha aparegut a la variable de classe neighbors per a cada imatge del conjunt de test.

```
def get_class (self):
```

```
    implementation
```

```
    return array
```

Returns: **Numpy Matrix** of Nx1 elements.



N: punts de prova (img)

Classe de la imatge

Implementation:

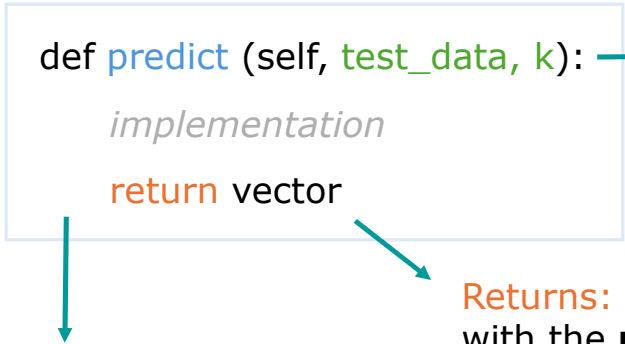
- For each row of self.neighbors, obtain the most voted value (the class to which that row belongs).
- Return an array with this class for each image.

***Clarification:** This array will have as many elements as points have been entered into the predict function.

2.4 predict

Function where we make the prediction of the class to which it belongs.

```
def predict (self, test_data, k):  
    implementation  
    return vector
```



Parameters:

- **test_data:** array which must be adjusted to an NxD matrix (N points in a D-dimensional space)
- **k:** number of neighbors to consider

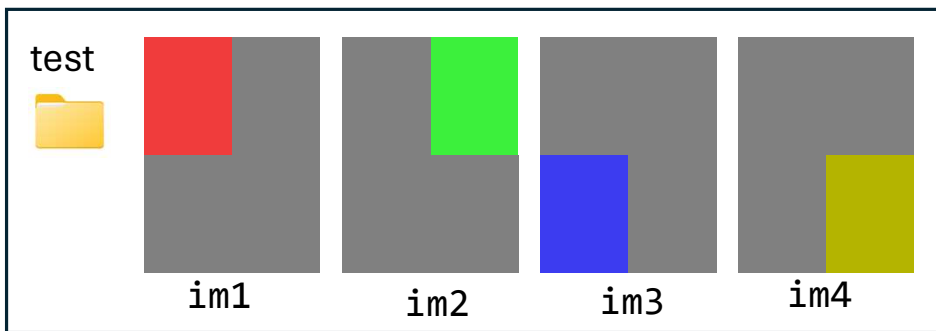
Returns: The output of **get_class**, a vector **Nx1** with the **predicted shape** for each test imagea

Implementation:

- Call the function **get_k_neighbors** to find the **nearest neighbors**.
- Call the function **get_class** to return the **most representative class**.

3. Test your code

Set of images of simple shapes:

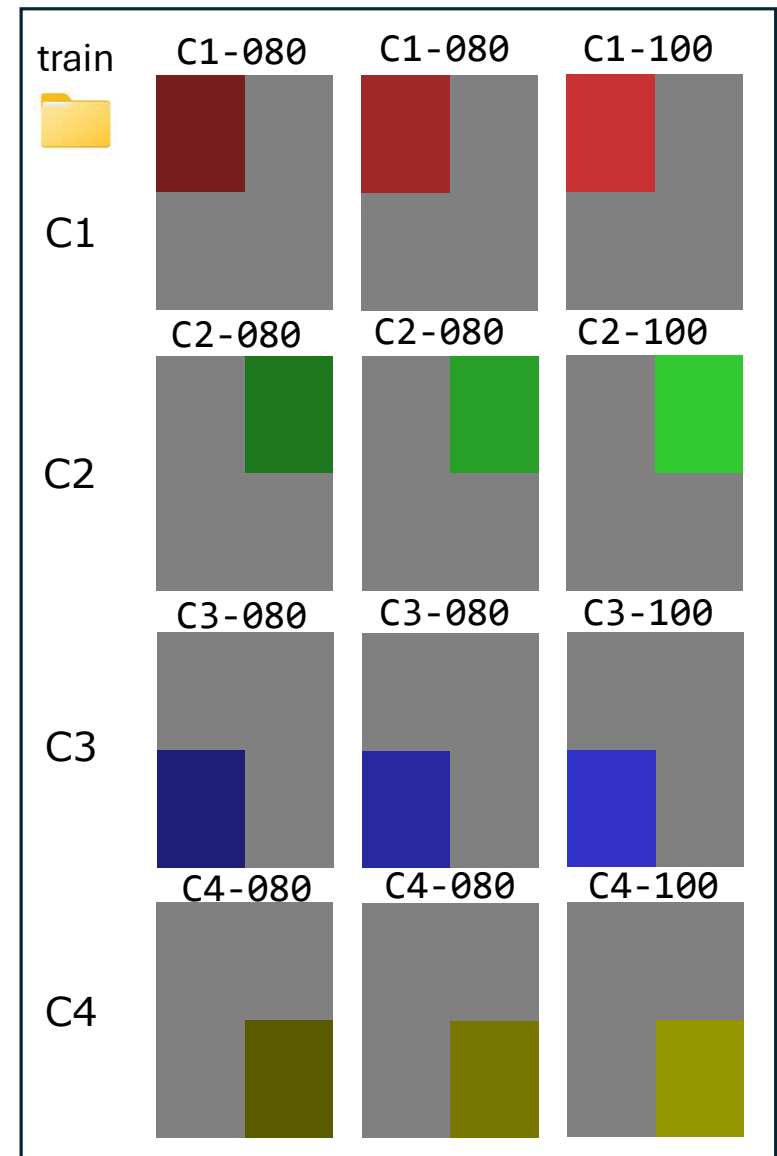


RED				
R	G	B	Grey-level	
120	30	30	60	train
160	40	40	80	train
200	50	50	100	train
240	60	60	120	test

GREEN				
R	G	B	Grey-level	
30	120	30	60	train
40	160	40	80	train
50	200	50	100	train
60	240	60	120	test


BLUE				
R	G	B	Grey-level	
30	30	120	60	train
40	40	160	80	train
50	50	200	100	train
60	60	240	120	test

YELLOW				
R	G	B	Grey-level	
90	90	0	60	train
120	120	0	80	train
150	150	0	100	train
180	180	0	120	test



3. Test your code

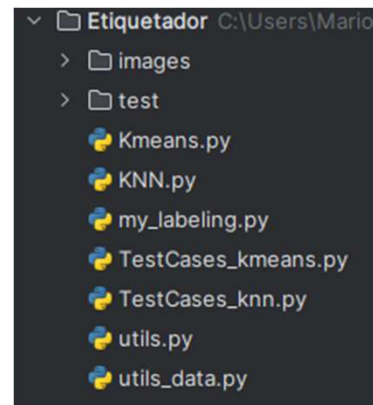
To test your code, you will need to download the file **wooclap_P2_SS2_KNN.zip**, which you will find at **cv.uab.cat** >> Practices >> Project 2. Labelling

1.  **testCases_img_knn:** Folder containing the set of images we will use for testing.
2. [testing_file_knn.py](#): File where you will run the test code to answer the questions in the questionnaire.

3. Test your code

The questionnaire you will undertake next will propose a series of tests for your **KNN.py** functions.

Save the contents of the '**woodclap_P2_SS2_KNN.zip**' file in the same directory as the rest of the code files.



To answer the questionnaire, you will need to test each of the proposed cases.

If you don't answer the questions correctly, you will **need to figure out what is wrong** with your code...

3. Test your code

In the initial part of the code, we find the data loading, label definition, and KNN initialization.

```
import pickle
from KNN import KNN

def load_data_from_pickle(file_path):
    with open(file_path, 'rb') as f:
        data = pickle.load(f)
    return data

if __name__ == '__main__':
    label_map = { 0: 'bottom-left', 1 : 'bottom-right', 2: 'top-left', 3: 'top-right'}

    data = load_data_from_pickle('testCases_img_knn/knn_wooclap_dataset.pkl')
    knn = KNN(data['train_images'], data['train_labels'])
```

3. Test your code

In the second part of the code, we find the calls to execute and answer the questions on Wooclap.

```
# Test1: Check the total number of images in the training data
knn_train_data = knn.train_data
print("Images number on the:", knn_train_data.shape[0])

# Test2: Check the total pixel size of images in the training data.
knn_train_data = knn.train_data
print("Images total pixels size:", knn_train_data.shape[1])

# Test3: Check the total number of test images evaluated to determine neighbor labels
knn.get_k_neighbours(data['test_images'],2)
knn_neighbours_labels = knn.neighbors
print("Amount of testing images evaluated:", knn_neighbours_labels.shape[0])

# Test4: Check the number of neighbor labels selected for each test image using K=2
knn.get_k_neighbours(data['test_images'], 2)
knn_neighbours_labels = knn.neighbors
print("Closest labels selected amount:", knn_neighbours_labels.shape[1])

# Test5: Check the predicted label for the first test image using K=4
label = knn.predict(data['test_images'], 4)
print("Label for k=4: ", label_map[label[0]])

# Test6: Check the predicted label for the last test image using K=6
label = knn.predict(data['test_images'], 2)
print("Label for k=6: ", label_map[label[3]])
```

Delibery of Part 2

For the evaluation of this second part of the practice you will have to upload to the Virtual Campus your **KNN.py** file which has to contain the **NIUs** of all the members of the group in the variable authors and your group in the variable group (at the beginning of the file). NIUs must be listed even if the groups are individual (e.g., [1290010,10348822] or [23512434]).

The delivery has to be before **28/04/2024 at 23:55.**

ATTENTION! It is important that you keep in mind the following points::

1. Code **correction** is done **automatically**, so be sure to upload files with the correct nomenclature and format (Do not change the file name or imports at the beginning of the file.
2. The code is subject to **automatic plagiarism** detection during correction.
3. Any part of the code that is not within the features of the Kmeans.py file **cannot** be **evaluated**, so do not modify anything outside of this file.
4. To prevent code from looping, there is a **time limit** for each exercise, so if your functions take too long it will count as an error.

Recall what is stated in the **Student Guide** about **plagiarism ...**

Notwithstanding other disciplinary measures deemed appropriate, and in accordance with the academic regulations in force, assessment activities will receive a zero whenever a student commits academic irregularities that may alter such assessment. Assessment activities graded in this way and by this procedure will not be re-assessable. If passing the assessment activity or activities in question is required to pass the subject, the awarding of a zero for disciplinary measures will also entail a direct fail for the subject, with no opportunity to re-assess this in the same academic year. Irregularities contemplated in this procedure include, among others:

- the total or partial copying of a practical exercise, report, or any other evaluation activity,
- allowing others to copy,
- unauthorized and/or non-cited use of AI tools (such as, Copilot, ChatGPT or equivalent) to solve exercises or projects or any assessed activity,
- presenting teamwork that has not been entirely done by the members of the team,
- presenting any materials prepared by a third party as one's own work, even if these materials are translations or adaptations, including work that is not original or exclusively that of the student,
- having communication devices (such as mobile phones, smart watches, etc.) accessible during theoretical-practical assessment tests (individual exams).