

# Pràctica 2: Etiquetatge

## Part 2: KNN i Forma

## Intel·ligència Artificial

Departament de Ciències de la Computació  
Universitat Autònoma de Barcelona

### 1 Introducció

En aquesta Part 2 de la pràctica continuem treballant en resoldre el problema d'etiquetatge d'imatge, vam veure com implementar el k-means per etiquetar el color de la roba. En aquesta segona part del projecte ens centrarem en l'etiquetatge de la forma de les peces de roba, tal com es veu a la figura 1. Per això ens centrarem en la implementació de l'algorisme K-NN:

**K-NN (k\_nn)** Mètode de classificació supervisada que es farà servir per assignar les etiquetes de tipus de roba.

En aquesta part de la pràctica resoldrem el problema d'etiquetar automàticament imatges per tipus de peça. Farem servir 8 tipus de peces de roba.



**Etiquetatge**  
→ "Yellow and Green T-shirt"

**8 classes de roba:**

- ✓ Dresses
- ✓ Flip Flops
- ✓ Jeans
- ✓ Sandals
- ✓ Shirts
- ✓ Shorts
- ✓ Socks
- ✓ Handbags



**11 colors bàsics:**

- ✓ Red
- ✓ Orange
- ✓ Brown
- ✓ Yellow
- ✓ Green
- ✓ Blue
- ✓ Purple
- ✓ Pink
- ✓ Black
- ✓ Grey
- ✓ White



Figure 1: Objectius de la pràctica

## 2 Fitxers necessaris

Farem servir els mateixos fitxers que ja varem descarregar per fer la Part 1, però ens centrarem sobretot en el fitxer `KNN.py` i `TestCases_knn.py`. Com que el K-NN és un algorisme d'aprenentatge supervisat necessitarem les imatges que hi ha al directori `Images`, i els conjunts d'aprenentatge i de test que hi tenim:

1. `images`: Carpeta que conté els conjunts d'imatges que utilitzarem.  
Dins d'aquesta carpeta trobareu:
  - (a) `gt.json`: Arxiu que conté la informació sobre la classe de les imatges de Train.
  - (b) `Train`: Directori amb el conjunt d'imatges que podem fer servir com a conjunt d'aprenentatge. De totes elles (train set) tenim informació de a quina classe pertany al fitxer `Class_labels.csv`.
  - (c) `Test`: Directori amb el conjunt d'imatges que volem etiquetar i de les quals no tenim cap informació, això serà el conjunt d'experimentació (test set).
2. `test`: Carpeta que conté el conjunt d'arxius necessari per poder realitzar les proves que es demanen al test (no els heu d'utilitzar en el codi, les funcions de test els carreguen automàticament en el `setUp`).
3. `utils.py`: Conté una serie de funcions necessàries per a convertir les imatges en color en altres espais, principalment passar-les a gris `rgb2gray()` i obtenir la pertinença als 11 colors bàsics `get_color_proba()`.
4. `Kmeans.py`: Arxiu on haureu de programar les funcions necessàries per a implementar el K-means per extreure els colors predominants.
5. `TestCases_kmeans.py`: Arxiu amb el qual podreu comprovar si les funcions que programeu en el fitxer `Kmeans.py` donen el resultat esperat.
6. `KNN.py`: Arxiu on haureu de programar les funcions necessàries per a implementar KNN per etiquetar el nom de la peça de roba.
7. `TestCases_knn.py`: Arxiu amb el qual podreu comprovar si les funcions que programeu en el fitxer `KNN.py` donen el resultat esperat.
8. `my_labeling.py`: Arxiu on combinareu els dos mètodes d'etiquetatge i les vostres millores per obtenir l'etiquetatge final de les imatges (*tercera part de la pràctica*)
9. `utils_data.py`: Conté una serie de funcions necessàries per a la visualització de resultats. (*tercera part de la pràctica*)

### 3 Què s'ha de programar?

En aquesta segona part de la pràctica només programareu l'algorisme de classificació KNN. Per això, haureu de programar les següents quatre funcions:

**\_init\_train:** Funció que s'assegura que la variable `train_data` de la classe KNN, la qual conté el nostre conjunt d'entrenament, tingui el format `float`, i tot seguit n'extreu les característiques. En la part obligatòria de la pràctica, utilitzarem els píxels de la imatge en R, G i B com a cada una de les característiques. Per tant `train_data` tindrà una mida de  $N \times 14400$ , on  $N$  és el número d'imatges del conjunt d'entrenament, i 14400 el número de píxels que té cada imatge  $60 \times 80 \times 3$  (R, G i B).

```
def _init_train(self, train_data):
    """
    initializes the train data
    :param train_data: P×M×N×3 matrix corresponding to P color images
    :return: assigns the train set to the matrix self.train_data
            shaped as P×D
    (P points in a D dimensional space)
    """
```

**get\_k\_neighbours:** Funció que pren com a entrada el conjunt de test que volem etiquetar (`test_data`) i fa el següent:

1. Canvia de dimensions de les imatges de la mateixa manera que ho hem fet amb el conjunt d'entrenament.
2. Calcula la distància entre les característiques del `test_data` amb les del `train_data`.
3. Guarda a la variable de classe `self.neighbors` les K etiquetes de les imatges més pròximes per a cada mostra del test.

*\*\*\* Pista: El càlcul de distàncies pot arribar a ser molt costos computacionalment si es fa en forma de bucle. Per fer-ho us recomanem que utilitzeu la funció `cdist`, de la llibreria `scipy.spatial.distance`, que us permetrà fer aquest càlcul de manera molt més ràpida.*

```
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 N×D matrix (
        N points in a D dimensional space)
    :param k: the number of neighbors to look at
    :return: the matrix self.neighbors is created (N×K). the ij-th
        entry is the j-th nearest train point to the i-th test point
    """
```

**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, `test_data`, i

retorna un `array` amb aquesta classe per a cada imatge. Aquest `array` tindrà tants elements com punts s'hauran entrat a la funció `predict`.

```
def get_class(self):  
    """  
    Get the class by maximum voting  
    :return: numpy array of Nx1 elements.  
            For each of the rows in self.neighbors gets the most  
            voted value (i.e. the class at which that row belongs)  
    """
```

`predict`: Funció que pren com a entrada el conjunt de test que volem etiquetar (`test_data`) i el número de veïns que volem tenir en compte (`k`), busca els seus veïns utilitzant `get_k_neighbours`, i retorna la classe més representativa utilitzant `get_class`. Aquesta funció bàsicament crida a les dues anteriors.

## 4 Entrega de la Part 2

Per a l'avaluació d'aquesta segona part de la pràctica haureu de pujar a l'aula Moodle el vostre fitxer `KNN.py` el qual ha de contenir el vostres NIUs a la variable `authors` i el vostre grup a la variable `group` (a l'inici de l'arxiu). Els NIUs han d'estar en una llista inclús en cas que els grups siguin individuals (p.ex.: `[1290010,10348822]` o `[23512434]`). **L'entrega s'ha de fer abans del dia 24/04/2022 a les 23:55.**

**ATENCIÓ!** és important que tingueu en compte els següents punts:

1. La correcció del codi es fa de manera automàtica, per tant assegureu-vos de penjar els arxius amb la nomenclatura i format correctes (No cambieu el Nom de l'arxiu ni els imports a l'inici d'aquest).
2. El codi esta sotmès a detecció automàtica de plagis durant la correcció.
3. Qualsevol part del codi que no estigui dins de les funcions de l'arxiu `KNN.py` no podrà ser avaluada, per tant no modifiqueu res fora d'aquest arxiu.
4. Per evitar que el codi entri en bucles hi ha un límit de temps per a cada exercici, per tant si les vostres funcions triguen massa les comptarà com a error.