

PROJECT 2

Labelling

SO: Sessió d'Orientació

Project 2 - Part 1

Artificial intelligence

2023-2024
Universitat Autònoma de Barcelona

1. Introducció

En aquesta pràctica, resoldrem un problema simple **d'etiquetatge d'imatges**.

Donat un conjunt d'imatges d'un catàleg de roba, desenvoluparem algorismes per aprendre com etiquetar imatges per tipus i color.

Per simplificar la pràctica, utilitzarem un conjunt petit d'etiquetes:

8 tipus de roba i 11 colors bàsics.

El sistema s'executarà en una imatge donada i retornarà una etiqueta per a un tipus de roba i una o més etiquetes per a colors. Implementareu els següents algorismes vistos a classe de teoria:




1. K-means (k-means): **Mètode de classificació no supervisada per trobar colors predominants.**
2. K-NN (k-nn) Mètode de classificació supervisada per classificar els tipus de roba.

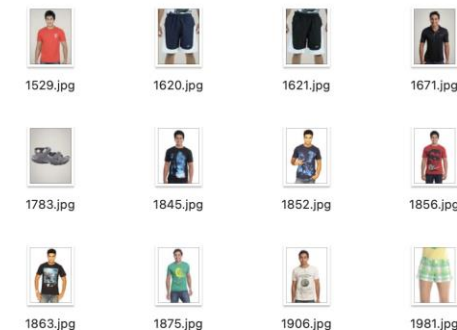
En aquesta **1a part del Projecte**, ens centrarem en

l'algorisme K-means

2. Fitxers requerits

Per dur a terme la pràctica, hauràs de descarregar les següents carpetes:

1.  **Images:** Carpeta que conté el set d'imatges que utilitzarem.
 - A. **gt.json:** Fitxer que conté informació sobre la classe d'imatges a entrenar (Train).
 - B.  **Train:** Carpeta amb el set d'imatges que utilitzarem com a set d'entrenament. Sobre elles (**train set**) trobarem informació sobre quina classe pertany en el fitxer **gt.json**.
 - C.  **Test:** Carpeta amb el set d'imatges que volem etiquetar i de les quals no tenim informació, aquest serà el nostre set d'experimentació (**test set**).




Classes de Train data estan etiquetades per tipus i colors

Conjunt d'imatges de les carpetes Train/Test folders

2. Fitxers requerits

Per dur a terme la pràctica, hauràs de descarregar les següents carpetes:

2.  Test: Carpeta que conté el conjunt de fitxers necessaris per poder realitzar les proves sol·licitades en els scripts de prova (no els has d'utilitzar en els teus scripts, les funcions de prova els carreguen automàticament en el setUp).

`test_cases_kmeans.pkl`

3. `utils.py`: Conté un seguit de **functions** necessaries per convertir les imatges a color en altres espais, bàsicament les convertim a escala de grisos (grey-level) `rgb2gray()` i aconseguim els 11 colors bàsics `get_color_prob()`.

4. `Kmeans.py`: Fitxer a on programareu les funcions necessaries per implementar K-means per extreure els colors predominants.

3. Preliminars

En aquesta pràctica treballarem amb **matrius** per a imatges i també amb **centròids** que utilitzen algorismes iteratius. Per implementar aquests algorismes de manera eficient, et recomanem que tinguis un bon domini de la **llibreria Numpy** per simplificar les teves funcions.

Els següents enllaços contenen alguns exercicis bàsics de Numpy que et poden ajudar a comprendre com funciona aquesta llibreria:

- NumPy Bàsic (41 exercicis amb solució):
<https://www.w3resource.com/python-exercises/numpy/basic/index.php>
- NumPy arrays (192 exercicis amb solució):
<https://www.w3resource.com/pythonexercises/numpy/index-array.php>
- NumPy Sorting and Searching (8 exercicis amb solució):
<https://www.w3resource.com/pythonexercises/numpy/python-numpy-sorting-and-searching.php>
- Altres exercicis: <https://www.w3resource.com/python-exercises/numpy/index.php>

3. Preliminars

Al final dels tutorials, hauries de ser capaç de realitzar les següents operacions amb matrius numpy:

- Redimensionar una matriu
- Calcular la mitjana dels valors d'un array
- Realitzar operacions entre matrius i vectors (per exemple, restar un vector a cada fila d'una matriu)
- Realitzar les dues operacions anteriors només amb certes files d'una matriu.

```
#####
# Code determining the pixels that belong to each
# of the six clusters for 1000 80x60 images.
#####

import time
import random
import numpy as np
K=6
N = 80*60
Iterations = 1000

# Version with python lists.
#####
list = random.choices(range(K), k=N)
t0 = time.time()
grups = {}
for iteration in range(Iterations):
    grups = {}
    for k in range(K):
        grups[k] = [True if elem == k else False for elem
in list]
t1 = time.time()-t0
# Version with numpy matrix
#####
vector_np = np.random.randint(K, N)
t0 = time.time()
for iteration in range(Iterations):
    grups = {}
    for k in range(K):
        grups[k] = vector_np==k
t2 = time.time()-t0

print(f'time to obtain the {K} groups for {N} pixels,
{Iterations} times.')
print(f'          WITHOUT numpy:\t{t1} seconds')
print(f'          WITH numpy:\t\t{t2} seconds')
print(f'WITH is {t1/t2} times faster than WITHOUT
numpy')
```

The execution gives:

```
          WITHOUT numpy: 4.341734886169434 seconds
          WITH numpy: 0.006999969482421875 seconds
WITH is 620.2505449591281 times faster than WITHOUT
numpy.
```

3. Preliminars

Abans de començar a programar, és molt recomanable entendre la classe amb la qual anem a treballar:

KMeans

KMeans És una classe dissenyada per carregar totes les dades, transformar-les i executar l'algorisme K-means.

classe **KMeans**:

Atributs

<code>self.num_iter = INTEGER</code>	<i>Contar iteracions fins arribar al número màxim</i>
<code>self.K = INTEGER</code>	<i>Número de clusters</i>
<code>self.options = {}</code>	<i>Diccionari amb els paràmetres de KMeans</i>
<code>self.centroids = array(2D)</code>	<i>Matriu amb les coordenades dels centròids</i>
<code>self.old_centroids = array(2D)</code>	<i>Matriu amb les coordenades dels centroids antics</i>
<code>self.X = []</code>	<i>Llista de matrius de tots els valors dels píxels</i>
<code>self.labels = array(1D)</code>	<i>Array d'índex dels centroids més propers</i>

3. Preliminars

Abans de començar a programar, és molt recomanable entendre la classe amb la qual anem a treballar:

KMeans

KMeans És una classe dissenyada per carregar totes les dades, transformar-les i executar l'algorisme K-means.

classe **KMeans**:

Funcions

<code>__init__(self, X, K=1, options=None)</code>	<i>Constructor de la classe KMeans</i>
<code>_init_X(self, X)</code>	<i>Inicialitza la matriu de punts</i>
<code>_init_options(self, options=None)</code>	<i>Inicialitza els paràmetres de KMeans</i>
<code>_init_centroids(self)</code>	<i>Inicialitza les classes centròids i centròids antics</i>
<code>get_labels(self)</code>	<i>Assigna el centròid més proper</i>
<code>get_centroids(self)</code>	<i>Reassigna centròids</i>
<code>converges(self)</code>	<i>Verifica si l'algorisme ha arribat al final</i>
<code>fit(self)</code>	<i>Executa l'algorisme KMeans</i>
<code>withinClassDistance(self)</code>	<i>Calcula la intra-clas o WCD</i>
<code>find_bestK(self, max_K)</code>	<i>Troba el valor K òptim</i>

3. Preliminars

Abans de començar a programar, és molt recomanable entendre la classe amb la qual anem a treballar:

KMeans

KMeans És una classe dissenyada per carregar totes les dades, transformar-les i executar l'algorisme K-means.

I altres **funcions externes**:

Funcions

`distance(X, C)` *Calcula la distància entre cada píxel i cada centròid*

`get_colors(centroids)` *Per a cada centròid retorna el color de l'etiqueta*

4. Què és el que hem de programar?

Has de codificar l'algorisme de classificació K-means per trobar els colors predominants en cada imatge.

Haureu de realitzar quatre tasques:

4.1. Funcions per **inicialitzar K-means**

4.2. Funcions **requerides per implementar K-means**

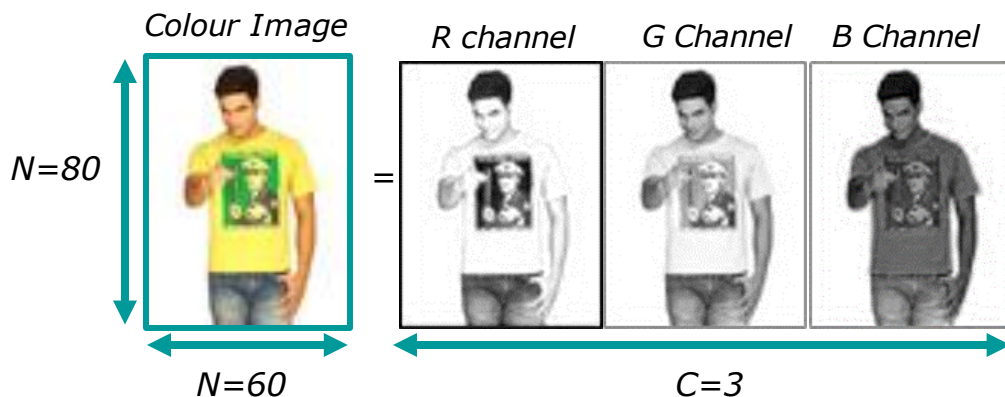
4.3. Funció que **implementa K-means**

4.4. Funcions que **troben el millor K** per aplicar K-Means per trobar els colors predominants

4.5. Funció que **convertirà els colors predominants en etiquetes de noms de colors**

4.1. Funcions per inicialitzar K-means

Sessió de Teoria



$$X = \begin{pmatrix} x_R^1 & x_G^1 & x_B^1 \\ \vdots & \vdots & \vdots \\ x_R^{4800} & x_G^{4800} & x_B^{4800} \end{pmatrix}$$

Sessió pràctica

`_init_options`: Permet indicar paràmetres de funcionament del K-Means. L'entrada és un diccionari que pot contenir les següents claus amb els seus valors possibles:

```
km_init: ['first', 'random', 'custom'],  
tolerance: float,  
maxiter: int,  
fitting: ['WCD', ...].
```

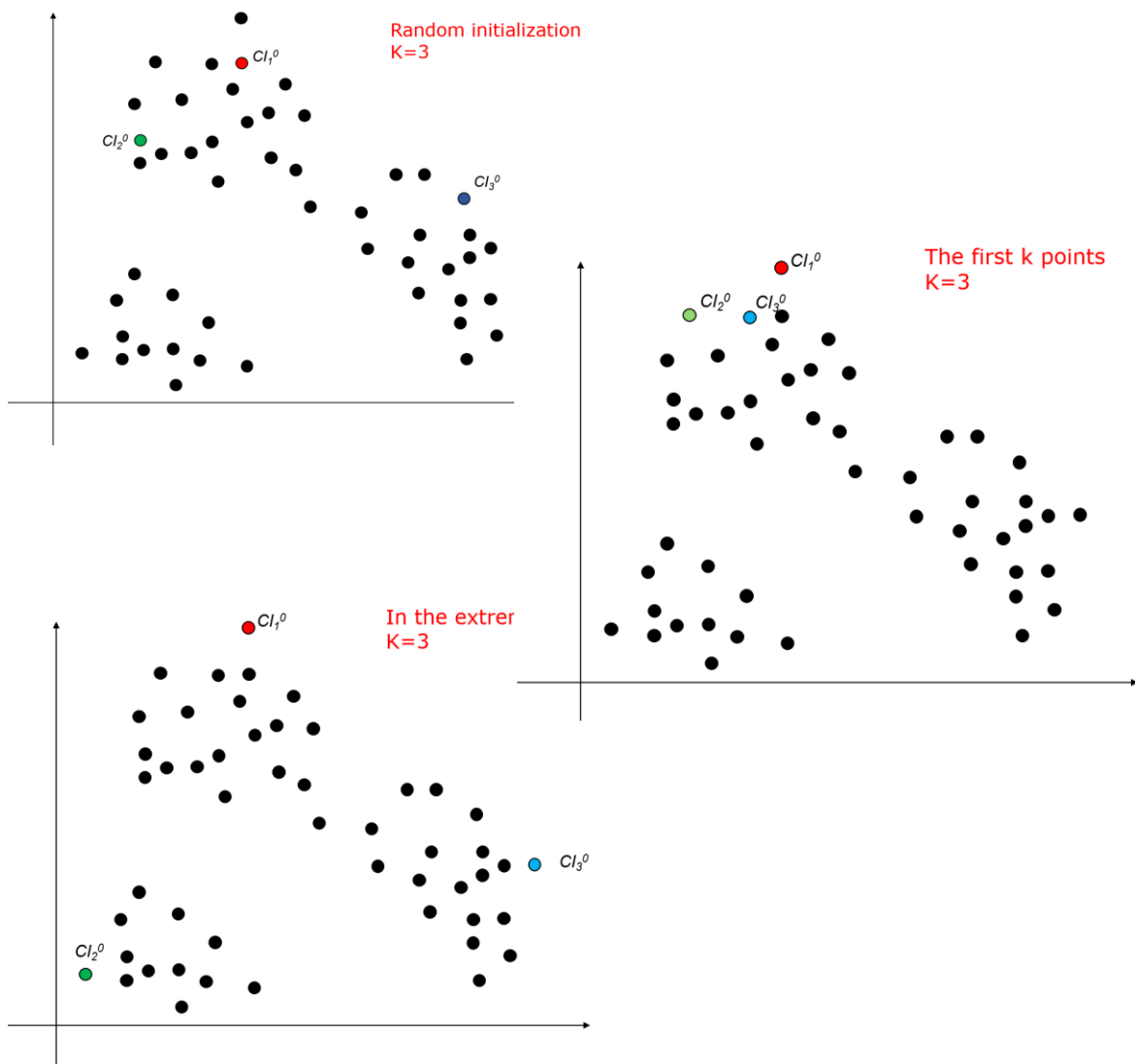
Si n'heu de menester podeu crear tants paràmetres com us siguin necessaris.

`_init_X`: Funció que rep com a entrada una matriu de punts X i fa diverses coses: (a) s'assegura que els valors siguin de tipus `float`; (b) si s'escau, converteix les dades a una matriu de només 2 dimensions $N \times D$; si no és així, i l'entrada és una imatge de dimensions $F \times C \times 3$ ¹ aleshores la transformarà i retornarà els píxels de la mateixa imatge, però en una matriu de dimensions $N \times 3$ i finalment guarda aquesta matriu a la variable `X` de l'objecte `KMeans`, `self.X`.

**** Pista: Podeu utilitzar la funció `reshape` de la llibreria `NumPy` per canviar les dimensions de la imatge d'entrada.*

4.1. Funcions per inicialitzar K-means

Sessió de Teoria



Sessió pràctica

`_init_centroid`: Funció que inicialitza les variables de classe centroids, i `old_centroids`. Aquestes dues variables tindran una mida de $K \times D$ on K és el nombre de centroides que hem passat a la classe `KMeans` i D és el nombre de canals. A continuació assigna valors als centroides en funció de l'opció d'inicialització que hàgim triat:

Opció `'first'`: assigna als centroides als primers K punts de la imatge X que siguin diferents entre ells. Per defecte haureu de programar l'opció `'first'`.

Opció `'random'`: triarà, de forma que no es repeteixin, centroides a l'atzar.

Opció `'custom'`: podrà seguir qualsevol altra política de selecció inicial de centroides que vosaltres considereu.

***Pista: punts distribuïts sobre la diagonal del hipercub de les dades?

4.2. Funcions requerides per implementar K-means

Sessió de Teoria

Definicions prèvies:

X : Conjunt de punts que formen el conjunt d'aprenentatge

$$X = \{\vec{x}^i : \vec{x}^i = (x_1^i \dots x_d^i) \quad \forall i: 1..n\}$$

k : Nombre de classes en que es vol dividir l'espai

C_i : Conjunt de punts de la classe i .

CI_i^t : Centre d'inèrcia de la classe i a l'instant t .

$d(\vec{x}, \vec{y})$: distància entre dos punts.

1. Escollir aleatòriament k punts de X per inicialitzar $CI_i^0, \forall i: 1..k$

2. Repetir

1. Per a (cada classe $C_i, \forall i: 1..k$) fer

1.

$$C_i = \{x \in X : \forall j \neq i, d(x, CI_i^t) \leq d(x, CI_j^t)\}$$

2. Calcular el nou centre:

2. **FPer**

3. $t++$;

3. Fins que (

4. Retornar($CI_i^t = CI_i^{t+1}; \forall i: 1..k$)
 $\{CI_i^t\}_{i: 1..k}$)

FFunció

$$CI_i^{t+1} = \left(\sum_{j=1}^{\#C_i} x_1^j / \#C_i \dots \sum_{j=1}^{\#C_i} x_d^j / \#C_i \right)$$

Sessió pràctica

`fit`: Funció que executa l'algorisme K-means iterant sobre els passos de l'algorisme que son:

1. Per cada punt en X , troba el centròid més proper.

2. Calcula el nou centròid utilitzant la funció `get_centroids`.

3. Incrementa el número d'iteracions en 1

4. Verifica si convergeix, en que que no sigui així, tornem al primer pas.

4.3. Funció que implementa K-means

Sessió de Teoria

Definicions prèvies:

X : Conjunt de punts que formen el conjunt d'aprenentatge

$$X = \{\vec{x}^i : \vec{x}^i = (x_1^i \quad \dots \quad x_d^i) \quad \forall i: 1 \dots n\}$$

k : Nombre de classes en que es vol dividir l'espai

C_i : Conjunt de punts de la classe i .

CI_i^t : Centre d'inèrcia de la classe i a l'instant t .

$d(\vec{x}, \vec{y})$: distància entre dos punts.

Algorisme

Funció k-means (X, k)

1. Escollir aleatòriament k punts de X per inicialitzar

$$CI_i^0, \forall i: 1 \dots k$$

2. Repetir

1. Per a (cada classe $C_i, \forall i: 1 \dots k$) fer

1.

$$C_i = \{x \in X : \forall j \neq i, d(x, CI_i^t) \leq d(x, CI_j^t)\}$$

2. Calcular el nou centre:

$$CI_i^{t+1} = \left(\sum_{j=1}^{\#C_i} x_1^j / \#C_i \quad \dots \quad \sum_{j=1}^{\#C_i} x_d^j / \#C_i \right)$$

2. FPer

3. t++;

3. Fins que (

$$CI_i^t = CI_i^{t+1}; \forall i: 1 \dots k$$

4. Retornar(

$$\{CI_i^t\}_{i: 1 \dots k}$$

FFunció

Sessió pràctica

distance: Funció que pren com a entrada la imatge X ($N \times D$) i els centroides C ($K \times D$), i calcula la distància euclidiana entre cada punt de la imatge amb cada centroide, i ho retorna en forma d'una matriu de dimensió $N \times K$.

get_labels: Funció que per a cada punt de la imatge X , assigna quin és el centroide més proper i ho guarda a la variable de la classe KMeans: **self.labels**. Per tant, aquesta variable serà un vector tan llarg com punts tingui X , i contindrà valors entre 0 i $K-1$.

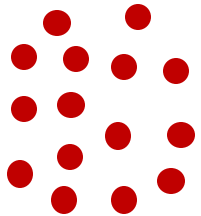
get_centroids: Funció que passa el valor de la variable **centroids** a **old_centroids**, i calcula els nous centroides. Per fer-ho, necessita calcular el centre de tots els punts de X relacionats amb un centroide. I això ho ha de fer per a cada un dels centroides.

converges: Funció que comprova si els **centroids** i els **old_centroids** són els mateixos. En cas afirmatiu voldrà dir que el K-Means ha arribat a una solució, per tant, la funció retornarà **True**, si no **False**. Per accelerar el temps de resposta podeu usar una certa tolerància a la diferència entre **centroids** i **old_centroids**, definida en les opcions d'inicialització i/o en el nombre màxim d'iteracions.

4.4. Funcions que **troben la millor K** per aplicar K-Means per trobar els colors predominants

Sessió de Teoria

Sessió pràctica



Compact class
Good classification



Non-compact class
Bad classification



Low Intra-class
distance



High Intra-class
distance



withinClassDistance: Funció que pertany a la classe `KMeans`, que calcula la seva distància intra-clas o *within-class-distance* (WCD) i actualitza el camp WCD amb el seu valor calculat així:

$$WCD = \frac{1}{N} \sum_{\mathbf{x} \in \mathbf{X}} \text{distancia}(\mathbf{x}, \mathbf{C}_{\mathbf{x}})^2 = \frac{1}{N} \sum_{\mathbf{x} \in \mathbf{X}} (\mathbf{x} - \mathbf{C}_{\mathbf{x}})(\mathbf{x} - \mathbf{C}_{\mathbf{x}})^T \quad (1)$$

on $\mathbf{C}_{\mathbf{x}}$ és el vector que representa el clúster al qual pertany el punt \mathbf{x} .

4.4. Funcions que troben la millor K per aplicar K-Means per trobar els colors predominants

Sessió de Teoria

Donada una imatge

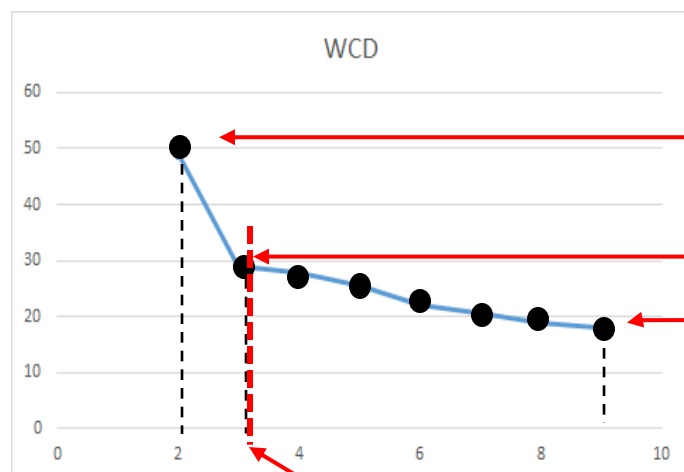


withinClassDistance() per K=2

withinClassDistance() per K=3

⋮

withinClassDistance() per K=9



K ideal

Sessió pràctica

find_bestK: Funció que pertany a la classe `KMeans` i pren com a entrada un número, `max_K`, que indica la màxima K que s'analitzarà. Executa l'algorisme K-Means per a cada K des de 2 fins a `max_K` sobre les dades en les quals s'ha inicialitzat l'objecte `KMeans`. Per a cada K calcula el valor de WCD que haurà d'anar decreixent a mesura que K augmenta. Aleshores calcularem el percentatge de decrement de WCD per a cada K, de la següent manera:

$$\%DEC_k = 100 \frac{WCD_k}{WCD_{k-1}} \quad (2)$$

On WCD_k és la WCD sobre el resultat del K-Means amb nombre de classes K. Ens quedarem amb la K que passi d'un decrement alt a un decrement estabilitzat. Per tant, en el moment que $100 - \%DEC_{k+1}$ sigui més petit que un llindar (per exemple el 20%) aleshores agafarem aquesta K com a K ideal i s'actualitzarà al camp K. En cas de no trobar-ne cap hi posarà `max_K`. Podeu ajustar millor aquest llindar de forma global.

4.5. Funció que **convertirà els colors predominants en noms de les etiquetes dels colors**

Sessió de Teoria

Experiments in
Anthropology

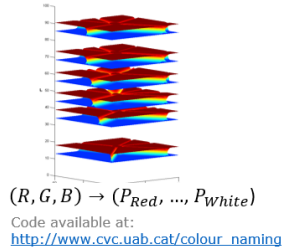
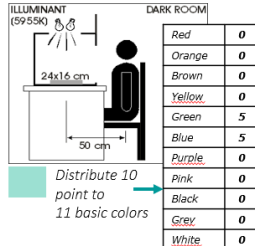
+

Experiments in
Experimental
Psychology

+

Mathematical
Models in
Computer Vision

Studies on 78 languages have shown that there are 11 universal basic color names shared by the most evolved languages



Berlin, B., & Kay, P. (1991)
Basic color terms: Their
universality and evolution.
Univ of California Press.

R. Benavente, M. Vanrell, R. Baldrich
(2006) A dataset for fuzzy color
naming, Color Research and
Applications

R. Benavente, M. Vanrell, R. Baldrich
(2008)
Parametric fuzzy sets for automatic color
naming, Journal of the OSA.

Sessió pràctica

`getcolor`: Funció que pren com a entrada els valors RGB dels centroides i els converteix en etiquetes de color.

Per fer això, passa de l'espai RGB a l'espai d'11 dimensions dels noms de color, on cada dimensió representa la probabilitat que aquest RGB rebi un nom bàsic dels 11 universals.

El codi d'aquesta conversió se us dona fet:

Funció: `get_color_prob()`

Fitxer: `utils.py`

`get_color_prob(A)`: A és una matriu de $K \times D$ i retorna una matriu de $K \times 11$ amb la probabilitat per cada un dels K punts els quals pertany a cada un dels 11 colors bàsic.

$(R, G, B) \rightarrow (P_{Red}, P_{Orange}, P_{Brown}, P_{Yellow}, P_{Green}, P_{Blue}, P_{Purple}, P_{Pink}, P_{Black}, P_{Grey}, P_{White})$

Delivery of Part 1

Per a l'avaluació d'aquesta primer part de la pràctica haureu de pujar al Campus Virtual el vostre fitxer **Kmeans.py** que ha de contenir el **NIU** de tots els membres del grup a la variable authors (a l'inici de l'arxiu). Els NIUs s'hauran d'afegir encara que els grups sigui d'una sola persona (e.g., [1290010,10348822] o [23512434]).

L'entrega s'ha de fer abans del dia **April 14th, 2024 at 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. Si no ho poseu bé la nota serà un 0. (No canvieu el nom del fitxer o els imports al principi del fitxer)
2. El codi està sotmès a **detecció automàtica** de plagis durant la correcció.
3. Qualsevol part del codi que no estigui dins de les funcions de l'arxiu Kmeans.py **no** podrà ser **avaluada**, per tant, no modifiqueu res fora d'aquest arxiu.
4. Per evitar que el codi entri en bucles infinits hi ha un **límit de temps** per a cada exercici, per tant si les vostres funcions triguen massa les considerarà incorrectes.

Recordeu el que es diu a la **guia docent sobre copiar o deixar copiar**

Sense perjudici d'altres mesures disciplinàries que s'estimin oportunes, i d'acord amb la normativa acadèmica vigent, les **irregularitats comeses per l'alumnat** que puguin conduir a una variació de la qualificació es qualificaran amb un zero (0). Les activitats d'avaluació qualificades d'aquesta forma i per aquest procediment no seran recuperables. Si és necessari superar qualsevol d'aquestes activitats d'avaluació per aprovar l'assignatura, aquesta assignatura quedarà suspesa directament, sense oportunitat de recuperar-la en el mateix curs. Aquestes irregularitats inclouen, entre d'altres:

- còpia total o parcial d'una pràctica, informe, o qualsevol altra activitat d'avaluació;
- deixar copiar;
- ús no autoritzat i/o no referenciat de la IA (p. ex, Copilot, ChatGPT o equivalents) per a resoldre exercicis, pràctiques i/o qualsevol altra activitat avaluable;
- presentar un treball de grup no fet íntegrament pels membres del grup;
- presentar com a propis materials elaborats per un tercer, encara que siguin traduccions o adaptacions, i en general treballs amb elements no originals i exclusius de l'alumnat.
- tenir dispositius de comunicació (com telèfons mòbils, smart watches, etc.) accessibles durant les proves d'avaluació teòric-pràctiques individuals (exàmens).

En resum: **copiar, deixar copiar o plagiar** en qualsevol de les activitats d'avaluació equival a un SUSPENS amb **nota inferior o igual a 3,0**.