Pràctica 2: Etiquetatge

Part 1: K-means i color

Intel·ligència Artificial

Departament de Ciències de la Computació
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 els algorismes que permetin aprendre a etiquetar automàticament imatges per tipus de peça i per color. Per simplificar la pràctica, farem servir un conjunt molt reduït d'etiquetes: 8 tipus de peces de roba i 11 colors bàsics. El sistema que desenvoluparem s'executarà sobre una imatge donada, i retornarà una etiqueta de tipus de peça de roba i una o vàries etiquetes de color. Podem veure un esquema general a la figura 1. Per fer-ho implementarem els següents algorismes vists a la teoria:

1. K-means (`k_means`) Mètode de classificació no supervisada que es farà servir per trobar els colors predominants d'una imatge.

2. K-NN (`k_nn`) Mètode de classificació supervisada que es farà servir per assignar les etiquetes de tipus de roba.

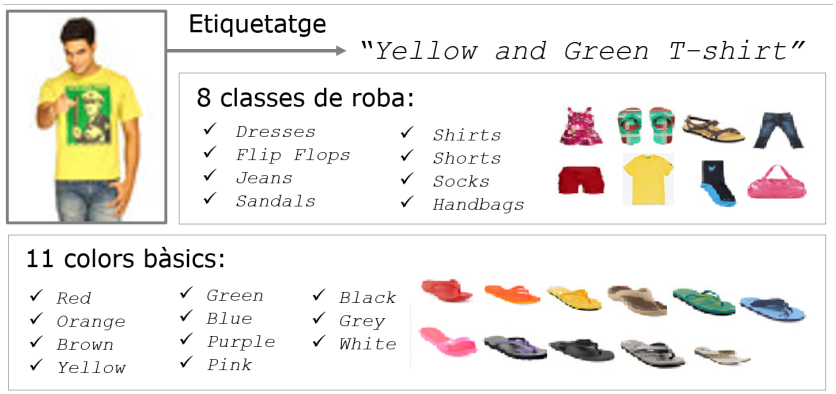

Figure 1: Objectius de la pràctica

En aquesta primera part de la pràctica ens centrarem en l'etiquetatge de color amb l'algorisme Kmeans.

2 Fitxers necessaris

Per a realitzar la pràctica haureu de descarregar les següents carpetes:

1. `Dataset`: Carpeta que conté els conjunts d'imatges que utilitzarem.

   Dins d'aquesta carpeta trobareu:

   (a) `Class_labels.csv`: 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 quína classe pertany al fitxer Class_labels.cvs.

   (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. `Code`: Carpeta que conté els fitxers de python amb els que treballarem. En alguns fitxers hi ha funcions útils que us donem per fer la pràctica i en altres fitxers haureu de programar les vostres funcions. Els fitxers són els següents:

   (a) `utils.py`: Conté una serie de funcions necessaries per a convertir les imatges en color en altres espais, principalment passar-les a gris `rgb2gray()` i obtenir la pertinença a els 11 colors bàsics `get_color_proba()` .

   (b) `Kmeans.py`: Arxiu on haureu de programar les funcions necessàries per a implementar el K-means per extreure els colors predominants.

   (c) `TestCases_kmeans.py`: Arxiu amb el qual podreu comprovar si les funcions que programeu en el fitxer `Kmeans.py` donen el resultat esperat.

   (d) `k_nn.py`: Arxiu on haureu de programar les funcions necessàries per a implementar KNN per etiquetar el nom de la peça de roba (segona part de la pràctica).

   (e) `TestCases_knn.py`: Arxiu amb el qual podreu comprovar si les funcions que programeu en el fitxer KNN.py donen el resultat esperat (segona part de la pràctica).

   (f) `My_labeling.py`: Arxiu on combinareu els dos mètodes d'etiquetatge i les vostres millores per a obtenir l'etiquetatge final de les imatges (segona part de la pràctica

3 Preparació

En aquesta pràctica treballarem amb matrius tant per treballar amb les imatges, com amb els centroides utilitzant algorismes iteratius. Per implementar aquests algorismes de manera

eficient, us recomanem tenir un bon domini de la llibreria Numpy per a simplificar les vostres funcions.

En els següents enllaços podeu trobar alguns exercicis bàsics de `Numpy` que us poden ajudar a entendre com funciona aquesta llibreria:

- `NumPy Basic (41 exercises with solution)`: https://www.w3resource.com/pyt hon-exercises/numpy/basic/index.php
- `NumPy arrays (192 exercises with solution ])`: https://www.w3resource.com/python-exercises/numpy/index-array.php
- `NumPy Sorting and Searching (8 exercises with solution)`: https://www.w3resource.com/p exercises/numpy/python-numpy-sorting-and-searching.php
- Altres exercicis: https://www.w3resource.com/python-exercises/numpy/index.php

En acabar amb els tutorials hauríeu de ser capaços de fer les següents operacions amb matrius `numpy`:

- Canviar les dimensions d'una matriu
- Calcular la mitjana dels valors d'una matriu
- Realitzar operacions entre matrius i vectors (ex: restar un vector a cada fila d'una matriu)
- Realitzar les dues operacions anteriors només amb certes files d'una matriu.

A la taula 1 us posem un exemple de com l'ús de la llibreria `numpy` pot accelerar l'execució d'un algorisme. Us posem dues versions d'un codi que ha de determinar els píxels que pertanyen a cada clúster per a 1000 imatges de 80×60 píxels en 6 clústers. La primera versió usa llistes de `python` i la segona usa matrius amb `numpy`, la primera versió resulta ser aproximadament 620 vegades més costoses que la versió `numpy`. En aquesta pràctica haureu de fer moltes més operacions que les que es fan en aquest codi, per tant és essencial que ho feu d'aquesta manera si voleu fer un codi eficient.

4 Què s'ha de programar?

En aquesta primera part de la pràctica programareu l'algorisme de classificació K-means per a trobar els color predominants de cada imatge. Hem dividit totes les funcions que heu de fer en 4 següents grups:

1. Funcions per inicialitzar el K-means (secció 4.1).
2. Funcions necessàries per implementar el K-means (secció 4.2).
3. Funció que implementa el k-Means (secció 4.3).

```
#####################################################################
# Codi que determina els píxels que pertanyen a cada un de 6 clústers
# per a 1000 imatges de 80x60 píxels.
#####################################################################
import time
import random
import numpy as np
K = 6
N = 80*60
Iteracions = 1000

# Versió amb llistes python
############################
llista = random.choices(range(K), k=N)
t0 = time.time()
grups = {}
for iteracio in range(Iteracions):
    grups = {}
    for k in range(K):
        grups[k] = [True if elem == k else False for elem in llista]
t1 = time.time()-t0

# Versió amb matrius numpy
############################
vector_np = np.random.randint(K, N)
t0 = time.time()
for iteracio in range(Iteracions):
    grups = {}
    for k in range(K):
        grups[k] = vector_np==k
t2 = time.time()-t0

print(f'Temps per a obtenir els {K} grups per a {N} pixels, {Iteracions} vegades.')
print(f'        SENSE numpy:\t{t1} segons')
print(f'        AMB numpy:\t\t{t2} segons')
print(f'AMB és {t1/t2} vegades més ràpid que SENSE numpy')
```

L'execució retorna:

```
        SENSE numpy: 4.341734886169434 segons
        AMB numpy: 0.006999969482421875 segons
AMB és 620.2505449591281 vegades més ràpid que SENSE numpy.
```

Table 1: Exemple de les avantatges d'usar la llibreria numpy.

4. Funcions que troben la millor k per aplicar el K-Means i trobar els colors predominants (secció 4.4).

5. Funció que convertirà els colors predominants en etiquetes de noms de colors (secció ??).

4.1 Funcions d'inicialització

`_init_options`: Permet indicar paràmetres de funcionament del kmenas. L'entrada és un diccionari que pot conteir 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 un 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` del objecte `KMeans, self.X`.

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

`_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 número de centroides que hem passat a la classe `KMeans` i D és el nombre de canals. A continuació assigna valors als centroids en funció de l'opció d'inicialització que haguem triat.

L'opció `'first'` assigna als centroides els primers K punts de la imatge `X` que siguin diferents entre ells. Per defecte haureu de programar l'opció 'first'. L'opció 'random' triarà, de forma que no es repeteixin, centroids a l'atzar. L'opció 'custom' podra seguir qualsevol altra politica de selecció inicial de centroids que vosaltres considereu ****.

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

4.2 Funcions necessàries pel K-means.

Per poder implementar el K-means, necessitarem tota una sèrie de funcions per actualitzar els clústers en cada iteració de l'algorisme. A la figura 2 reproduïm el pseucodi d'aquest algorisme tal com l'hem vist a les classes de teoria. Tot seguit especifiquem les 4 funcions necessàries:

[1] Aquí suposem que N és el número de píxels d'una imatge que té F files i C columnes, per tant $N = F \cdot C$. La dimensió del espai de característiques ve donat per D que en imatges color és 3.

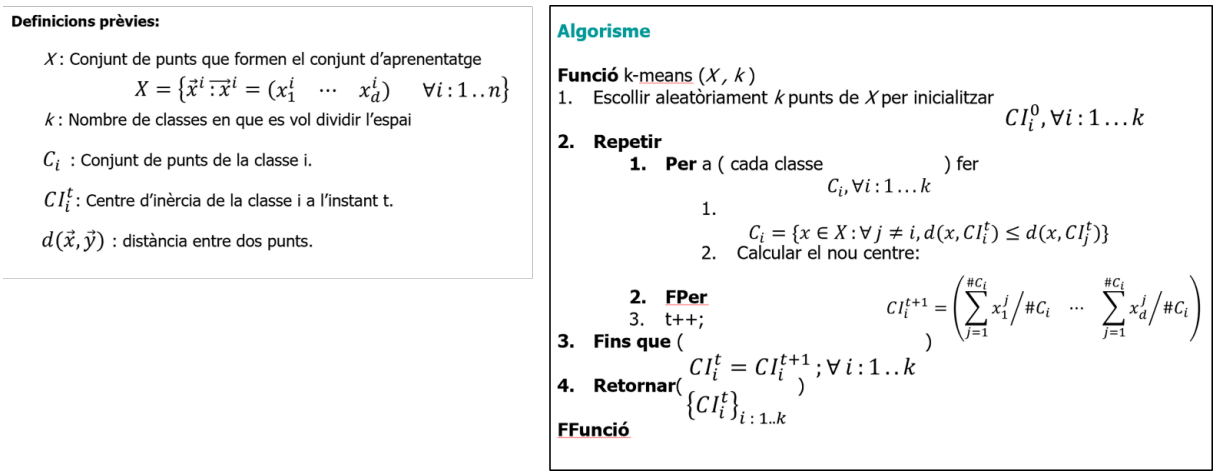

Figure 2: Pseudocodi de l'Algorisme K-means vist a teoria.

`distance`: Funció que pren com a entrada la imatge `X` ($N \times D$) i els centroides `C` ($K \times D$), i calcula la distancia 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 es el centroide més proper i ho guarda a la variable de la classe `KMeans`: `self.labels`. Per tant aquesta variable serà un vector tant 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 centroids. Per a 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 son els mateixos. En cas afirmatiu voldrà dir que el `K-Means` ha arribat a una solució, per tant la funció retornarà `True`, sinó `False`. Per accelerar el temps de resposta podeu usar una certa tolerancia a la diferencia entre `centroids` i `old_centroids`, definida en les opcions d'inicitalització i/o en el número màxim d'iteracions.

4.3 K-means

Ara que ja hem programat totes les funcions necessàries de la classe `KMeans` podem programar la funció `fit` que ens agruparà els punts de la imatge en `K` clusters.

`fit`: Funció que executa l'algorisme Kmeans i itera sobre els passos vists a la figura 2, que són:

1. Per a cada punt de la imatge, troba quin és el centroide més proper.
2. Calcula nous centroides utilitzant la funció `get_centroids`
3. Augmenta en 1 en numero d'iteracions
4. Comprova si convergeix, en cas de no fer-ho torna al primer pas.

4.4 Funcions per trobar la k ideal

Un cop hem programat la funció k-means necessitem automatitzar el paràmetre k que li passarem, i que representarà el nombre de colors predominants de cada imatge. Per tant, necessitem trobar el valor de K que s'adapta millor a la distribució de colors de cada imatge. Per fer això farem servir la distància intra-class per a diferents K, i ens quedarem amb aquella K a partir de la qual el resultat de la distància intra-class s'estabilitza i ja quasi no disminueix. Això ho farem amb les dues funcions següents que seran dues funcions de la classe `Kmeans`:

`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:

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

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

`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 Kmeans per a cada k des de 2 fins a `max_K` sobre les dades en les que 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 Kmeans amb número 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 aquestas 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.

5 Funció per etiquetar el color

Un cop tenim els color predominants d'una imatge ja podem assignar les etiquetes dels noms que corresponen a aquests colors. Per fer això haureu de programar la funció `get_color` dins del fitxer `kmeans.py`. Aquesta funció utilitzarà la funció `get_color_prob(A)` que implementa els resultats de treballs previs que han après com els humans assignem els noms de colors.

`get_color:` 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. Aleshores, per a cada centroide retorna l'etiqueta del nom de color que presenta la màxima probabilitat, i en cas d'empat tornarà la primera de la llista. Per a obtenir la representació de cada color a l'espai 11D cal cridar la funció `utils.get_color_prob(A)` on `A` és una matriu de $K \times D$ i retorna una matriu de $K \times 11$ amb la probabilitat per a cadascun dels K punts de pertànyer a cada un dels 11 colors bàsics.

**** Pista: Dins el fitxer* `utils.py` *trobareu quins són els colors representats per cada una de les 11 probabilitats, dins de la variable colors.*

6 Entrega de la Part 1

Per a l'avaluació d'aquesta primera part de la pràctica haureu de pujar al Campus Virtual el vostre fitxer `Kmeans.py` que ha de contenir els NIAs de tots els membres del grup a la variable authors i el vostre grup a la variable group (a l'inici de l'arxiu). Els NIAs han d'estar en una llista inclús pel cas que els grups siguin individuals (p.ex.: [1290010,10348822] o [23512434])

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 canvieu 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 `Kmeans.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.