

# **INFORME DE LA PRÀCTICA** **ETIQUETATGE**

Intel·ligència Artificial

Enginyeria informàtica

Curs 2020-2021

Joel Prat Vilanova 1568204

Alejo de Urrengoechea 1569298

Arnau Masana Silvente 1569115

# 1. INTRODUCCIÓ

Aquest treball té com a objectiu posar en pràctica diversos algorismes per tal de poder diferenciar i classificar imatges de roba segons el tipus de roba i el color d'aquesta. Aquest projecte abarca 8 peces de roba diferents i 11 colors bàsics. El funcionament del programa ha de ser que a partir d'una imatge donada, ha de ser capaç de retornar una etiqueta amb el tipus de roba corresponent i una o varies etiquetes segons els colors adequats.

El llenguatge de programació utilitzat per a fer el projecte ha estat Python ja que és un llenguatge que disposa de moltes funcionalitats i permet la programació orientada a objectes. Per poder realitzar el que se'ns demana, hem de saber treballar amb matrius, és per això que una de les llibreries que més s'utilitza durant la pràctica és "Numpy".

A nivell tècnic, per a realitzar aquest projecte, els algorismes a posar en pràctica són:

- K-means: és un algorisme de classificació no supervisada que té com a objectiu la partició d'un conjunt en "n" observacions en "k" grups. Aquest algorisme ens resulta útil per a poder trobar els colors predominants d'una imatge.
- K-NN: és un algorisme de classificació supervisada que a partir d'una imatge, busca les següents "K" imatges d'entrenament que més s'hi assemblen, posteriorment, selecciona la classe més abundant entre les "K" possibilitats. Aquest algorisme ens resulta útil per a poder identificar quin tipus de roba és.

Per a utilitzar aquests dos algorismes, el que s'ha fet es programar dues classes, una per a cada algorisme corresponent, amb els seus atributs i mètodes per tal de poder realitzar tot el que se'ns demanava. Aquests mètodes ja se'ns donaven marcats, de tal forma que la nostre feina ha consistit en programar-los seguint les indicacions donades.

# 2. MÈTODES D'ANÀLISI IMPLEMENTATS

Per a poder analitzar el funcionament del programa hem implementat dues funcions de control: una d'anàlisi qualitatiu i una altra d'anàlisi quantitatiu.

- Retrieval\_by\_color

De les funcions qualitatives disponibles hem programat la “Retrieval\_by\_color”, la qual té com a objectiu poder filtrar imatges per colors, és a dir, la funció a partir d'un llistat d'imatges, les etiquetes dels colors predominants de les imatges, i el color o colors que es desitgen filtrar, retorna totes les imatges d'aquells colors.

Per a fer-ho hem recorregut tots els elements de les etiquetes, i quan n'hi havia alguna que coincidia amb el color desitjat, afegim la imatge corresponent a una llista externa per tal de retornar-la.

```

12
13 def Retrieval_by_color(llistaImatges, etiquetes, colors):
14     llista=[]
15     aux = 0
16     for a in etiquetes: #recorrem totes les etiquetes
17         for b in a:     #recorrem cada element de l'etiqueta
18             if(b in colors): #si l'element de l'etiqueta correspon amb el color desitjat entrem a l'if
19                 llista.append(llistaImatges[aux]) #afegim l'element corresponent a una llista auxiliar
20                 aux = aux + 1 #augmentem el comptador
21
22     return llista
23

```



Aquestes dues imatges són resultat de la execució del codi anterior. La primera imatge correspon a filtrar imatges que tinguin el color “Red” en alguna de les seves etiquetes. La segona imatge, tot i que no s'apreci bé, és resultat de la execució del codi amb la intenció de filtrar imatges “Red” i “Blue”, és per això que ens surten tants elements.

- Kmean\_statistics

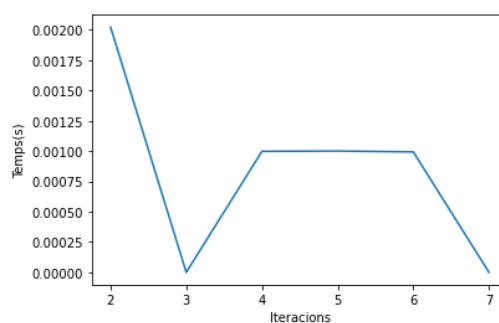
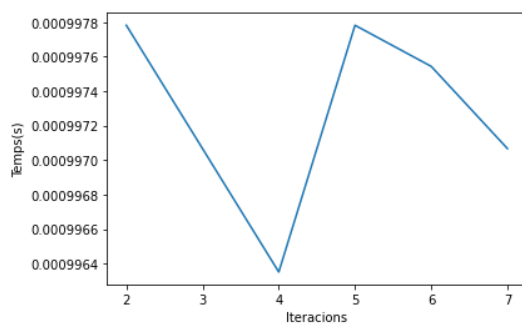
De les funcions quantitatives, hem programat la funció “Kmean\_statistics” la qual ens mostra les estadístiques de l’algorisme k-means. Aquest programa informa del correcte funcionament ja que a partir d’un conjunt d’imatges ens mostra les dades tant de la distància entre clusters com el les iteracions/temps que s’ha produït.

Per a fer aquesta funció el que hem realitzat es un bucle que va executant les funcions fit i whitinClassDistance de la classe KMeans. Això ens retornarà una grafica. Aquesta grafica la traiem de la llista de temps que tarda el programa per a cada K. Per treure les iteracions/temps, en el cas de les iteracions simplement les hem comptat i en el cas del temps hem utilitzat la funció time de la llibreria time. Per a poder veure la gràfica hem inclòs la llibreria “matplotlib.pyplot”, la qual ens permet mostrar dades en forma de gràfiques.

```

25 def Kmeans_statistics(llistaImatges, kmeans, Kmax):
26     K = 2 #inici de K=2
27     i = Kmax-K #calculem el numero de iteracions màximes
28     llista = []
29     llistaIteracions = []
30
31     while (K != Kmax): #loop mentre el numero de iteracions no sigui el màxim
32         tInicial = time.time() #temps inicial de la iteració
33         kmeans[K].fit()
34         wcd = kmeans[K].whitinClassDistance() #amb aquesta comanda i l'anterior calculem la distància
35         tFinal = time.time() #temps final de la iteració
36         Temps = tFinal - tInicial #temps total
37         llista.append(Temps)
38         llistaIteracions.append(K)
39         K = K + 1 # augmentem K, ==> una iteració menys a fer
40
41     plt.plot(llistaIteracions, llista) #afegim les dues llistes a la gràfica
42     plt.ylabel('Temps(s)') #indiquem les dades de l'eix Y
43     plt.xlabel('Iteracions') #indiquem les dades de l'eix X
44     plt.show() #mostrem la gràfica
45

```



Aquestes gràfiques són resultats de diferents execucions del codi anterior. Les dues execucions han estat fetes amb Kmax = 8, per això veiem que les iteracions van de 2 a 7. Aquestes gràfiques ens resulten útils per a veure com varia el temps que ha trigat en realitzar cada iteració. Per a cada execució del codi, els resultats de les taules varien.

### 3. MILLORES SOBRE KMEANS I KNN

A més a més de l'implementació de l'algorisme kmeans, knn i la realització de funcions de control quantitatiu i qualitatiu, el projecte ens demanava que proposéssim millores a aquests algorismes per tal de millorar el seu funcionament.

- Inicializations de Kmeans:

La primera millora realitzada ha estat l'implementació de més mètodes d'inicialització dins de la funció `init_centroids` de la classe `KMeans`. Els tres tipus d'inicialització són el "first", "random" i "custom".

- L'opció `first` s'encarrega d'assignar els centroids als primers K punts de la imatge X que siguin diferents entre ells. Per a realitzar-ho primerament fem un bucle que iteri per a cada element de X. Posteriorment comparem l'element que s'itera amb el centroid, i si no és igual afegim l'element que s'itera a la llista de centroids a la posició corresponent. Aquesta posició ve determinada per un comptador que augmenta en 1 per a cada iteració del bucle.

```
87 x = 0 #comptador a 0, serà utilitzat en els 3 tipus
88 if tipus == 'first':
89     for i in self.X: #recorrem cada element
90         comprobacio = np.equal(i, self.centroids).all(1) #comprovem que l'element iterat no pertanyi als centroids
91         if not any(comprobacio): #si cap element hi pertany (retorna tot false)
92             self.centroids[x] = i #afegim el centroid i a la posició que determini el comptador
93             x = x + 1
94         if x == self.K: #si la x == k sortim del for
95             break
```

- L'opció `random` s'encarrega d'escollir centroids a l'atzar pero sense repetir-ne. Per a fer-ho hem utilitzat la funció `np.random.randint(low=0, high=longitud)` essent `longitud` el número total d'elements d'X. Aquesta funció escull un nombre a l'atzar que posteriorment serà utilitzat. L'opció `random` funciona amb un bucle que itera `self.K` vegades. Dins d'aquest bucle comparem l'element escollit a l'atzar dins del conjunts de valors de X, `self.X[valor random d'abans]`, amb els centroids, i si no n'hi cap d'igual, afegim l'element `self.X[random]` a la llista de centroids a la posició corresponent. Aquesta posició ve determinada per un comptador que augmenta en 1 per a cada iteració del bucle. A cada iteració del bucle tornem a buscar un valor `random`.

```
97 elif tipus == 'random':
98     np.random.seed()
99     longitud = len(self.X) - 1 #-1 ja que recorrerà de 0 a n-1
100     index = np.random.randint(low=0, high=longitud) #escull un valor random entre el rang 0 a longitud
101
102     while self.K != x:
103         comprobacio = np.equal(self.X[index], self.centroids).all(1) #comprovem que l'element random no pertanyi als centroids
104         if not any(comprobacio): #si cap element hi pertany (retorna tot false)
105             self.centroids[x] = self.X[index] #afegim el centroid random d'X a la posició corresponent
106             x = x + 1
107
108     np.random.seed() #tornem a calcular un valor aleatori
109     longitud = len(self.X) - 1
110     index = np.random.randint(low=0, high=longitud)
```

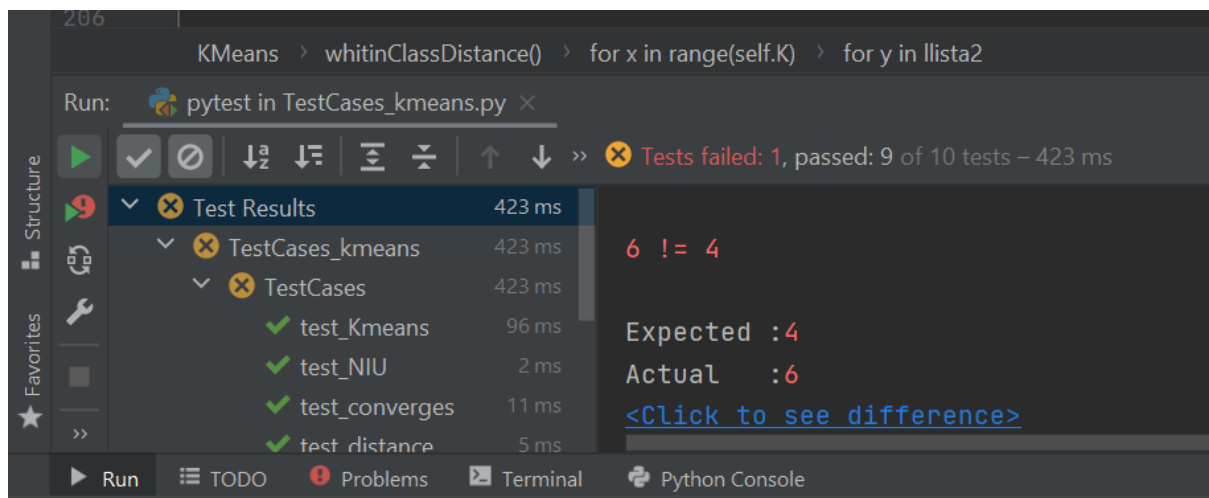
- Diferents heurístiques per BestK:

Aquesta millora consisteix en modificar la funció `WithinClassDistance` de la classe `KMeans`, per tal de canviar les heurístiques de les distàncies. Per a fer-ho, el que hem fet ha estat recórrer el número d'elements de `K`. Posteriorment guardem tots els elements que corresponguin de `self.labels` amb l'iterador. Un cop guardats seleccionem l'`X` corresponent, i recorrem tots els seus elements. Posteriorment calculem l'intraclass i l'interclass multiplicant les matrius corresponents amb les seves transposades. Finalment per a calcular Fischer el que fem és dividir el valor d'intraclass amb el de interclass.

```

191     c = np.array([])
192     for x in range(self.K):
193         llista = np.where(self.labels == x) #seleccionem els valors que compleixin aquesta condició
194         llista2 = self.X[llista[0]] #agafem el primer valor per a poder recórrer les X
195         for y in llista2:
196             valor = y - self.centroids[x]
197             intra = np.matmul(valor, valor.T) #multiplicacio de les matrius
198             inter = np.matmul(self.centroids[x], self.centroids[x].T) #multiplicacio de les matrius
199
200             fischer = intra / inter
201             c = np.append(c, fischer)
202
203     sumaClusters = sum(c)
204     total = sumaClusters / len(c)
205     return total

```



The screenshot shows the test results for the `KMeans` class. The test `test_Kmeans` failed with an assertion error. The error message is: `6 != 4`. The expected value was `4` and the actual value was `6`. The error message is: `Expected :4 Actual :6 <Click to see difference>`.

Aquesta ultima imatge és el resultat de la compilació del codi amb les modificacions anteriors. Aquests són els resultats de la compilació del test `Kmeans`. Com es pot observar, el test esperava un resultat diferent del que ha obtingut, això és degut al canvi d'heurística que hi ha hagut. (En l'entrega del `kmeans.py` tenim aquesta millora comentada per tal d'executar l'original i així passar els testos correctament).

- Find\_BestK:

Aquesta millora consisteix en intentar aconseguir millorar els resultats obtinguts al modificar el paràmetre llinar de la funció Find\_BestK de la classe Kmeans. Aquest valor estava marcat per defecte en un 20%.

Per a poder analitzar quina és la millor K, el que hem fet ha estat modificar el valor del llinar i analitzar els resultats. Per a poder analitzar els resultats ele que hem fet ha estat afegir els valors de K en llistes i comparar els resultats.

```
74     llista=[]
75     for imagen_aux in range(5):
76         x = km.KMeans(test_imgs[imagen_aux]) #inicialitzem l'objecte kmeans
77         x.fit()
78         x.find_bestK(8) #cridema a la funció
79         llista.append(x.K) #afegim resultat
80     print(llibra)
81
```

Valors de K amb un 20%	Valors de K amb un 30%	Valors de K amb un 10%
[5, 4, 2, 5, 3]	[5, 2, 2, 5, 3]	[7, 6, 7, 5, 5]

Veient aquests resultats, podem concloure que com més petit és el llinar, els valors de K que retorna la funció són més grans. De tal forma, sabent que com més gran sigui la K, més grans són els costos d'execució ens interessa tenir un llinar permissiu per tal d'aconseguir millor eficiència.

## 4. CONCLUSIÓ

Aquest treball ens ha servit per a poder posar en pràctica i augmentar els nostres coneixements sobre python i la llibreria numpy. A més a més, ens ha servit per a familiaritzar-nos amb els conceptes teòrics ensenyats a classe. Finalment creiem que el valor més important que hem extret de la pràctica ha sigut poder veure la utilitat real dels algorismes kmeans i knn.