Pràctica 2: Etiquetatge

Part 2: KNN i Forma

Intel·ligència Artificial

2019-2020

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.

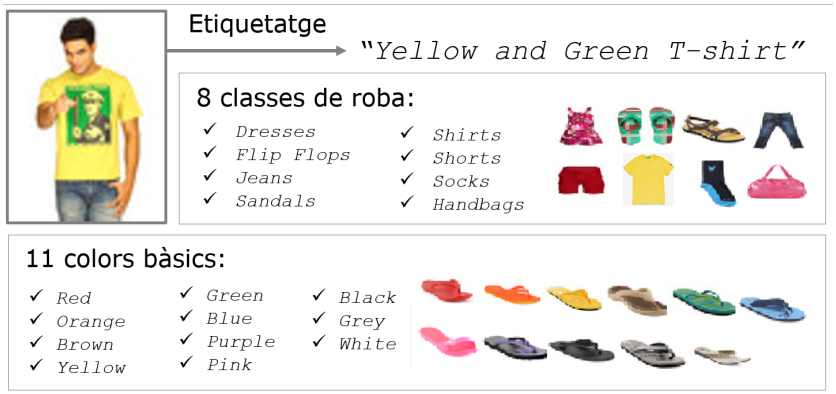

Figure 1: Objectius de la pràctica

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 datasets amb les imatges que utilitzarem.

   Dins d'aquesta carpeta trobareu:

   (a) `Test`: Conjunt d'imatges que volem etiquetar.

   (b) `Train`: Conjunt d'imatges qeu utilitzarem com a model per a la classificació de formas.

2. `test`: Carpeta amb fitxer per a tests.

3. `codi`: en l'arrel del fitxer comprimit trobareu el codi necessari per al projecte i els fitxers que haureu d'omplir (`KNN.py`). Els fitxers necessaris per treballar són:

   (a) `utils.py`: Conté una series de funcions que us poden ser útils principalment per a la part del KMeans.

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

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

   (d) `Kmeans.py`: Arxiu on ja hi haureu programat les funcions necessàries per a implementar Kmeans per classificar el COLOR.

   (e) `KNN.py`: Arxiu on haureu de programar les funcions necessàries per a implementar KNN per classificar el NOM de la peça de roba.

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

3 Què s'ha de programar?

En aquesta segona part de la pràctica només programareu l'algorisme de classificació K-NN. 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 com a cada una de les característiques. Per tant `train_data` tindrà una mida de $N \times 4800$,on N és el número d'imatges del conjunt d'entrenament, i 4800 el número de píxels que té cada imatge.

```
def _init_train(self,train_data):
    """
    initializes the train data
    :param train_data: PxMxNx3 matrix corresponding to P color images
    :return: assigns the train set to the matrix self.train_data
       shaped as PxD
    (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 NxD matrix (
       N points in a D dimensional space)
    :param k:  the number of neighbors to look at
    :return: the matrix self.neighbors is created (NxK). 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 imatgedel 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 NIAs a la variable authors i el vostre grup a la variable group (a l'inici de l'arxiu).

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.