Pràctica 2: Etiquetatge

Part 3: Millores i Avaluació

Intel·ligència Artificial

Grau d'Enginyeria Informàtica

01 de juny de 2023

Grup PRACT2_16

Santiago Atienza Reig 1600088
Ilias Chakrane Hedioued 1569270
Samya Karzazi El Bachiri 1568589

Índex

Introducció a la pràctica

Aquesta pràctica té com a objectius principals resoldre i entendre com classificar imatges segons les seves etiquetes, utilitzant els algorismes K-Means i K-NN (K- Nearest Neighbors). En el nostre cas les imatges que s'hauran de classificar són d'un catàleg de roba i ho haurem de fer segons el tipus de peça (shape) i/o el color (colour).

Per una banda, hem d'implementar l'algorisme K-means, el qual és un mètode de classificació no supervisada que el fem servir per trobar els colors predominants d'una peça de roba. En podem diferenciar 11 colors bàsics:

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

D'altra banda, usarem l'algoritme K-NN, el qual és un mètode de classificació supervisada, per assignar les etiquetes del tipus de roba.

En el nostre cas tenim 8 classes de roba:

- Dresses
- Flip-Flips
- Jeans
- Sandals
- Shorts
- Socks
- Handbags

Partim de la hipòtesi que els dos algoritmes han de funcionar perfectament i no hi ha d'haver cap error a l'hora de detectar el tipus de roba o el color que es tracta.

Un cop tindrem definides les classes amb les funcions adients, per obtenir els resultats dels algoritmes, realitzem unes proves per veure que tan exacte són aquests algoritmes a l'hora de classificar la roba segons la forma i el color. Aquests mètodes d'anàlisi que hem implementat podem classificar-les en:

- Funcions d'anàlisi qualitatiu: Funcions que ens permeten avaluar d'una manera visual els nostres classificadors.
- Funcions d'anàlisi quantitatiu: Funcions que ens permeten avaluar de manera numèrica els nostres classificadors.
- Millores als mètodes de classificació: Funcions que avaluaran com variant diferents paràmetres es poden obtenir millors o pitjors resultats.

Finalment, explicarem les conclusions assolides després de programar les funcions dels dos algoritmes, d'haver realitzat els mètodes d'anàlisi i millorar els mètodes.

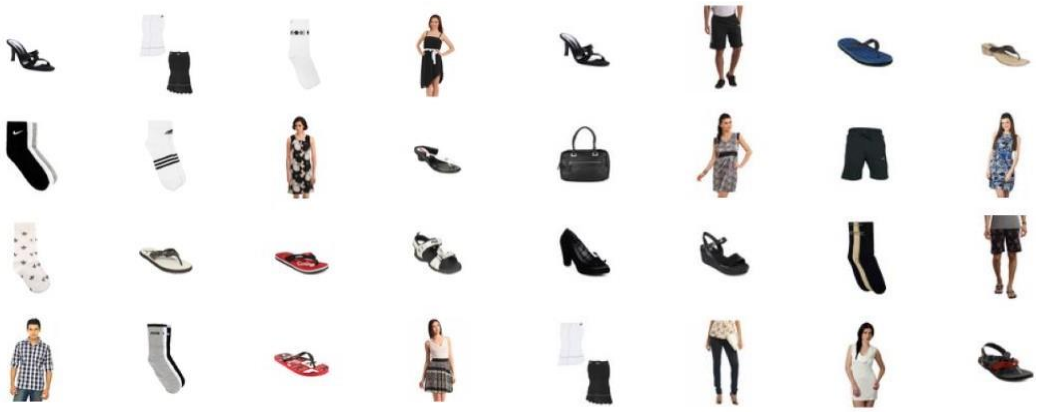
Mètodes d’anàlisi implementats

En aquesta última pràctica hem implementat els següents mètodes d’anàlisi, classificats segons siguin funcions d’anàlisi qualitatiu o quantitatiu.

- ## Funcions d’anàlisi qualitatiu

  La funció implementada en aquest cas és la de retrieval_by_colour(). En aquesta funció li passem per paràmetre una llista d’imatges; les etiquetes després d’aplicar l’algorisme Kmeans, que indiquen els colors; la pregunta que en aquest cas són els diferents colors que volem en la roba; i si passarem un ‘1’ si volem que la peça de roba contingui tots els colors indicats anteriorment o un ‘0’ si volem les peces de roba que tinguin algun dels colors.

  Per exemple, si a la pregunta li passem una llista amb colors com seria [‘White, ‘Black’] i indiquem que volem obtenir les peces de roba que tinguin tots dos colors, obtindrem les Img 1. En canvi, si volem les peces que tinguin algun d’aquests dos colors ens imprimeix per pantalla les imatges que tinguin algun dels colors indicats, com en la Img 2.



Img 1: resultats retrieval_by_colour(), allColours = 1

Img 2: resultats retrieval_by_colour(), allColours = 0

  Hem de tenir en compte que les imatges que visualitzem va en funció del valor de K punts que li estem donant. En el nostre cas ja li hem aplicat la funció findBestK(), per no haver de trobar manualment aquest valor de clústers k.

  Una altra funció que hem implementat és retrieval_by_shape(). Aquesta té els mateixos paràmetres d’entrada que la funció anterior, amb la diferència que les etiquetes les obtenim després d’aplicar l’algorisme KNN i en comptes de preguntar per colors, preguntarem per tipus de roba.

  Després de fer proves amb la funció, si per exemple volem obtenir les xancletes, posarem com a pregunta ‘Flip Flop’ i obtenim el següent resultat:

Img 3: resultats retrieval_by_shape()

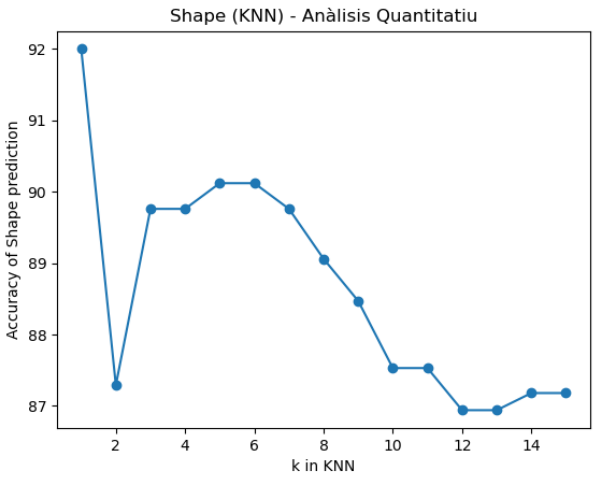
L'última funció que hem implementat en aquest anàlisi qualitatiu ha sigut retrieval_combined(). Aquesta té els mateixos paràmetres d'entrada que les dues funcions anteriors, i a l'hora de posar la pregunta posem color i peça de roba. En l'exemple següent es veu una imatge del que obtenim en preguntar "Black Jeans":

Img 4: resultats retrieval_combined()

▪ Funcions d'anàlisi quantitatiu

En aquesta part hem implementat la funció get_shape_accuracy(), la qual retorna el percentatge d'etiquetes correctes i depèn del valor de K punts que s'estigui examinant. Rep per paràmetre les etiquetes obtingudes després d'aplicar el KNN i el Ground-Truth.

Després de fer proves hem obtingut el següent graf (Img.5) que mostra l'exactitud de l'algorisme KNN en funció de la K que li donem.

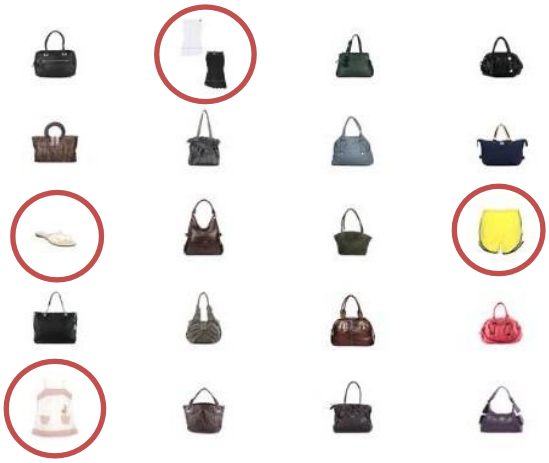


Img 4: resultats retrieval_combined()

Observant el gràfic, veiem que en utilitzar una k=1 o k=5, obtenim uns valors d'accuracy del 92% i 90,2% respectivament, per tant, a l'hora d'aplicar l'algorisme KNN per classificar en funció de la forma de la roba, utilitzarem k=1 o k=5, ja que obtindrem millors resultats en veure 1 o 5 veïns més propers a la imatge que volem classificar.

Per tant, podem afirmar que és un bon classificador de roba, fent servir les etiquetes de cada peça si emprem una K petita. A mesura que anem augmentant el valor de la K, l'exactitud de l'algoritme en encertar la peça de roba que es tracta va disminuint i no té sentit analitzar amb k més grans que 15.

Per acabar de refutar aquesta idea que l'exactitud no és del 100%, provem la funció retrieval_by_shape() amb K = 5 i com a paràmetre pregunta li passem 'Handbags' aconseguirem les següents imatges. Podem veure que no només imprimeix per pantalla bosses sinó que també uns pantalons, unes sandàlies entre d'altres.



Img 5: resultats retrieval_by_shape() per comparar amb el resultat get_shape_accuracy()

També hem implementat la funció get_color_accuracy(), que retorna el percentatge de colors correctes que prediu. Rep per paràmetre els colors obtinguts després d'aplicar KMeans i el Ground-Truth i obtenim el graf mostrat en la img 6.

		K-MEANS ACCURACY PERCENTATGES					
		TEST			TRAIN		
		WhithinClassDistance	InterClassDistance	Fisher	WhithinClassDistance	InterClassDistance	Fisher
km_init	first	77%	77%	78%	61%	61%	61%
	sharding	77%	77%	78%	61%	61%	61%

Img 7: Graf obtingut en aplicar get_color_accuracy()

Observem que no hi ha gaire diferència en els resultats obtinguts després de provar inicialitzacions del Kmeans diferents i càlcul de distàncies entre classes. Cal emfatitzar que si analitzem el temps d'execució en cada cas, veiem que si utilitzem el coeficient de Fisher, que s'explica més endavant, el temps d'execució és menor. A més a més, no hem tingut en compte el valor de la k, ja que directament cridàvem a la funció findBestK_llindar(), que ens donava el millor valor. Per tant, qualsevol inicialització usada seria vàlida i el càlcul de distàncies seria millor utilitzar el coeficient de Fisher per tenir un temps d'execució menor.

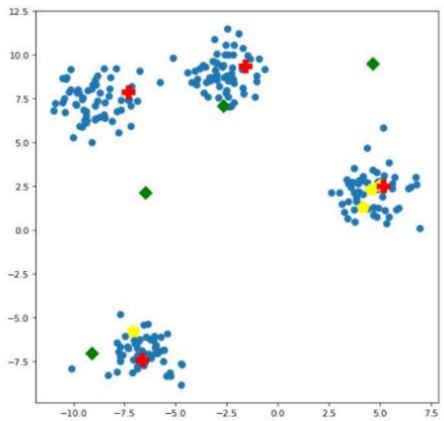
Millores sobre Kmeans i KNN

Per tal de millorar els mètodes de classificació hem realitzat 3 canvis que s'expliquen a continuació:

- Inicialitzacions del Kmeans

Una de les millores que hem implementat en el Kmeans és afegir diferents opcions a l'hora d'inicialitzar els centroides a la funció _init_centroids(). La primera d'elles és l'opció que anomenem "random", que consisteix a escollir de manera aleatòria els centroides, però amb la condició que aquests no es puguin repetir entre ells. D'altra banda, també hem afegit la funcionalitat "Kmeans++", que primer escull un centroide a l'atzar i els posteriors s'escullen a partir d'una probabilitat proporcional segons el càlcul del quadrat de la distància del centroide més proper.

Finalment, hem inclòs l'opció anomenada "naive sharding", on s'agrupen els punts i s'inicialitzen els centroides a partir de la mitjana dels punts centrals de cada grup.



Img 8: Representació de les diferents inicialitzacions aplicades

Per una altra banda, hem decidit implementar també una sèrie d'heurístiques per trobar la millor K que utilitzar. La primera d'elles és InterClassDistance(), que s'encarrega de trobar els centroides de classes que estiguin molt separades entre elles aplicant la fórmula següent: $\sqrt{\sum}(i - j)^2$. A més, hem afegit FisherCoefficient(), que troba el centroide de classes molt separades i compactes entre elles, fent servir la divisió WithinClassDistance() i InterClassDistance().

- # Diferents heurístiques i Find_BestK

La funció find_bestK() és una millora que hem implementat per determinar el valor òptim de K en el nostre algorisme de cerca del color de la peça. Anteriorment, a la pràctica, utilitzàvem un valor de llindar fix del 20% per trobar K.

No obstant això, amb aquesta millora, ara podem passar un paràmetre de llindar com a argument a la funció find_bestK(), cosa que ens permet especificar un valor de llindar diferent del 20% anterior. Això ens ofereix més flexibilitat i control sobre el procés de cerca.

A més a més, també pots d'indicar quin mètode s'usarà per calcular la distància. En el nostre programa, s'han implementat tres mètodes diferents: WithinClassDistance, FisherCoeficient i InterClassDistance. Cada un d'aquests mètodes té les seves pròpies característiques i pot influir en el resultat final de la cerca.

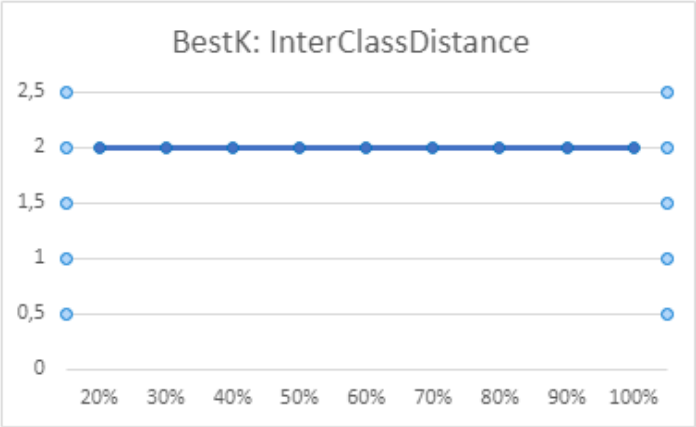
En permetre que l'usuari seleccioni un d'aquests mètodes, podem avaluar la millor K per a cadascuna de les tres heurístiques diferents. Això significa que podem realitzar múltiples proves i comparar els resultats obtinguts per determinar quina de les tres heurístiques és la més efectiva en la cerca del color de la peça.

Img 9: findBestK() amb distancia WhithinClassDistance

Gràfica que mostra la millor K segons al llindar que posem amb la funcio WhithinClassDistance:

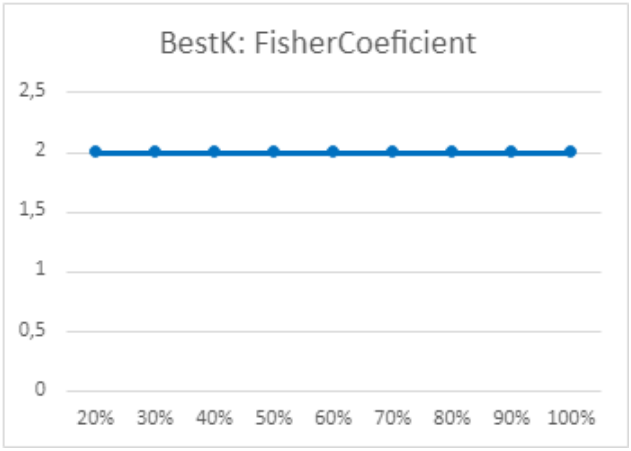
- *InterClass Distance*: En aquest mètode no es modifica cap paràmetre respecte a l'heurística *WithinClass Distance*. Retorna la suma d'aplicar la fórmula √∑(i - j)^2 per cada grup.

  La diferència de resultats depenent de si apliquem o no la millora seran diferents, ja que estarem utilitzant heurístiques diferents.
  Aquesta funció funciona molt bé quan les classes estan molt separades entre elles, en canvi, quan estan molt juntes entre elles realitza una mala classificació.



Img 10: findBestK() amb distancia InterClassDistance

- *Fisher Coeficient*: En aquest mètode tampoc es modifica cap paràmetre respecte a l'heurística WithinClassDistance. El resultat obtingut en utilitzar aquesta millora és la divisió del mètode WithinClassDistance per InterClassDistance. Aquesta funció funciona molt bé quan les classes estan molt separades entre elles i compactes, en canvi, quan estan molt juntes entre elles o poc compactes, realitza una mala classificació.



Img 11: findBestK() amb distancia Fisher Coeficient

La millor K abans de la millora, independentment del mètode que s'esculli, sempre donava el valor k=2. I després hem pogut observar que en funció del llindar que aplicàvem aquest valor de la k variava com es poden veure als gràfics anteriors.

En resum, la funció find_bestK() representa una millora significativa en el nostre programa, ja que ens permet trobar la millor K ajustant el llindar de manera personalitzada i triar entre diferents mètodes de càlcul de distància. Això ens proporciona una major precisió i eficiència en la cerca del color de la peça, permetent adaptar el nostre algorisme a diferents escenaris i requisits específics.

Conclusió global

En aquesta pràctica, hem après principalment a saber quin algoritme utilitzar a l'hora de voler classificar les imatges. Per exemple, utilitzarem el K-means quan volem un algoritme per classificar, en aquest cas imatges, segons un criteri amb aprenentatge no supervisat. En canvi, el KNN quan necessitem un algoritme de classificació amb aprenentatge supervisat.

A més a més, en el cas del K-means hem vist que els mètodes de classificació es poden millorar, segons utilitzem diferents tipus d'inicialització o diferents heurístiques per realitzar el càlcul de la distància. Tanmateix, hem vist que el valor de la K influeix en els resultats que obtenim. Gràcies a la gràfica representada hem vist que en aplicar l'algorisme KNN, els valors que obtindrem seran més exactes si utilitzem els valors de K més petits, en canvi menys exactes quan K és més gran. Per evitar cometre errors a l'hora d'escollir la K, tenim la funció de findBestK() que avalua quin és el millor valor de K per a cada situació que es planteja.

D'altra banda, ens ha faltat implementar les funcions de Kmeans_statistics i Features for KNN, que ens agradaria implementar més endavant per acabar de millorar els algorismes aplicats.

En conclusió, aquesta pràctica ens ha permès ampliar els nostres coneixements sobre la IA en conèixer alguns algoritmes molt útils que tracten aquest àmbit de la informàtica. També ens ha permès aprendre a dominar més el llenguatge Python, que és un dels llenguatges de programació més importants i més utilitzats actualment a l'hora de fer servir la Intel·ligència Artificial.

Bibliografia

[1] Wikipedia, https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm, 2022

[2] Wikipedia, https://en.wikipedia.org/wiki/K-means_clustering, 2022

[3] Janalta Interactive, https://www.techopedia.com/definition/32066/k-nearest-neighbor-k-nn, 2017

[4] Janalta Interactive, https://www.techopedia.com/definition/32057/k-means-clustering, 2017