Informe Proyecto Etiquetador

Marta Matute - 1496672
David Pacheco - 1565824

1 Introducción

El propósito de este proyecto es implementar dos algoritmos que permitan etiquetar imágenes de ropa automáticamente basándose en su color y forma. En total hay 8 tipos de ropa y 11 colores diferentes.

Para poder etiquetar correctamente el color de las imágenes se utiliza el algoritmo $K-means$, un método de clasificación no supervisada. Para la clasificación de la ropa por forma, se utiliza el algoritmo KNN (*K-nearest neighbors*), un algoritmo de clasificación supervisada.

Para la evaluación de la calidad de la clasificación, se emplean el estadístico P@K (*precision at k*) y la *accuracy*. Mencionar también que se valoran el número de *clusters* para el algoritmo $K-means$ así como el de vecinos utilizados por el algoritmo KNN, ambos representados por la letra k. Se mide la tasa de acierto en los dos casos y se busca el valor óptimo.

En cuanto al código, todas las funciones obligatorias han sido implementadas tanto en `K-means.py`, en `KNN.py` como en `my_labelling.py`. Además, también hemos editado las funciones opcionales `find_bestK` y `get_colors_and_probs`. Disponemos además de 2 formas de inicialización: $first$[1], $random$[2] y una alternativa: $crop$[3]. La primera es una inicialización básica en la que se cogen como centroides las k primeros píxeles diferentes de la imagen como centroides; la segunda elige k píxeles aleatorios diferentes de la imagen como centroides y la última reduce el tamaño de la imagen

[1]Con la opción: 'init':'first'
[2]Con la opción: 'init':'random'
[3]Con la opción: 'crop':'cropped

a la mitad para así tener más precisión en la selección de color sin perjudicar la precisión al predecir la forma, además es varias veces más rápido.

2 | Análisis previo K-means y KNN

2.1 | Gráficas rendimiento antes de las mejoras

A continuación se muestran las gráficas obtenidas de ejecutar el algoritmo $K-means$, KNN y $P@K$ del `main.py`.

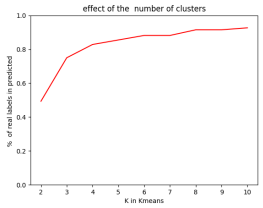

Figura 1: Effect of the number of clusters

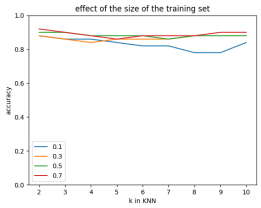

Figura 2: Effect of the size of the training set

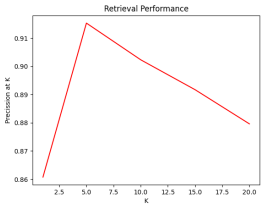

Figura 3: Precision at K

Observando la primera imagen (1), la precisión para k número de *clusters* pequeño es de 0,5 y para $k = 10$ está cerca de 1. Sin embargo, a partir de $k = 5$, observamos que el crecimiento no es tan marcado y por lo tanto podríamos considerar que no tiene tanto sentido seguir incrementando k a partir de ese punto.

Si miramos la segunda (2) nos damos cuenta de que incrementar el tamaño del conjunto de *train* no es tan relevante como podríamos pensar en primera instancia.

Finalmente, la P@K (3) consigue el pico máximo alrededor de $k = 5$ y consigue un máximo de 0,9153349.

2.2 | Class Accuracy

Al ejecutar nuestro código sin cambiar parámetros y simplemente con la implementación recomendada en la guía, podemos ver cuál es el porcentaje de veces que acierta la clase para las imágenes de test. En nuestro caso los resultados que se obtienen son:

Global accuracy on shape prediction: 0.9048178613396005
Class: Dresses, Acc: 0.9354838709677419
Class: Flip Flops, Acc: 0.7731958762886598
Class: Handbags, Acc: 0.946236559139785
Class: Heels, Acc: 0.8787878787878788
Class: Jeans, Acc: 0.967741935483871
Class: Sandals, Acc: 0.851063829787234
Class: Shirts, Acc: 0.9795918367346939
Class: Shorts, Acc: 0.9
Class: Socks, Acc: 0.9148936170212766

Vemos que, para la mayoría de clases, el porcentaje de aciertos supera el 90 % e incluso roza el 98 % para la clase 'Shirts'. En cambio, el porcentaje es significativamente más bajo para otras clases, en particular para 'Flip Flops', 'Heels' y 'Sandals'. Inicialmente, podríamos pensar que este fuese un error del etiquetador (quizá haya una falta de imágenes de *training* o quizá el algoritmo no sea lo suficientemente bueno), sin embargo, al echar un vistazo a las imágenes y sus etiquetas reales, vemos que algunas de las imágenes están etiquetadas de manera confusa incluso para nosotros mismos. Veamos algunos ejemplos de este fenómeno:

Predicted class: Flip Flops
Real class: Heels

Predicted class: Sandals
Real class: Flip Flops

Predicted class: Heels
Real class: Sandals

Predicted class: Flip Flops
Real class: Sandals

Las etiquetas reales pueden pueden generar cierta confusión al ver las imágenes.

De hecho, se da el caso en el que el algoritmo, al igual que nosotros, no es capaz de predecir la clase correctamente. Para este problema no podemos implementar una solución en el código.

2.3 | Color Accuracy

De nuevo sin cambiar ninguno de los parámetros iniciales, tenemos que la precisión a la hora de predecir los colores no es demasiado alta. Vamos a ver precisión para cada uno de los colores en la tabla siguiente:

Global accuracy for colors:	0.4967242378055276
Accuracy for Red:	0.4172661870503597
Accuracy for Orange:	0.06451612903225806
Accuracy for Brown:	0.488
Accuracy for Yellow:	0.6666666666666666
Accuracy for Green:	0.7407407407407407
Accuracy for Blue:	0.7777777777777778
Accuracy for Purple:	0.6078431372549019
Accuracy for Pink:	0.49230769230769234
Accuracy for Black:	0.7205882352941176
Accuracy for Grey:	0.27674418604651163
Accuracy for White:	0.21151586368977673

Observamos que la precisión global es de apenas el 50 %, y para determinados colores (naranja, gris y blanco), la precisión es mucho más baja de lo que podríamos considerar aceptable. En el caso del blanco y el gris se debe al color de fondo de las imágenes y las sombras de las mismas. El caso del naranja es debido al color de piel de la mayoría de modelos que aparecen en las imágenes. Estos errores de predicción podrían llegar a enmendarse mediante herramientas más avanzadas de visión por computador. Por ejemplo, podríamos usar la librería `opencv` para diferenciar los objetos del fondo y las personas de las prendas.

Para mejorar la precisión de los colores que no se ajustan al problema de la detección de objetos, hemos observado que, en general, la mala clasificación se produce por la incapacidad de detectar el color dominante de la imagen. Es decir, nuestro clasificador encuentra acertadamente el conjunto de colores que se encuentran en la imagen. Sin embargo es incapaz de decidir cuál es el que se refiere al objeto principal.

Veamos un ejemplo muy claro de este caso, haciendo un query para 'Red' nos devuelve la imagen:

Predicted colors: [‘White’, ‘Black’, ‘Orange’, ‘Red’]
Real colors: [‘Black’]

Para nosotros es evidente que el producto principal de la foto no coincide con aquello que hemos detectado de color rojo. Para solucionar este problema, hemos implementado algún cambio que veremos más adelante relacionado con la zona donde buscamos el color.

3 | Mejora del código base

Dado que, como hemos visto en el apartado de análisis, a veces salen imágenes erróneas y el color no siempre es acertado, hemos implementado dos funciones que permiten que la búsqueda sea más acertada en general.

3.1 | Devolver únicamente imágenes coincidentes

La primera de ellas es devolver únicamente aquellas imágenes de las que tenemos un mínimo de certeza de que están bien. Al hacer el análisis, nos hemos dado cuenta de que, en ocasiones, se ofrecían como solución algunas imágenes que simplemente tenían probabilidad 0. Este factor dañaba la tasa de aciertos media. Para ello, hemos implementado en `retrieve_image_by_color` una selección de aquellos colores de los que tenga una seguridad no nula de que son correctos.

3.2 | Recorte de la imagen

Como se ha mencionado en la introducción, en ocasiones la predicción de color era errónea ya que ponderaba demasiado el blanco del fondo de la imagen o algún color secundario muy nítido. Para ello, hemos centrado la imagen en aquella parte que

define su color y forma eliminando prácticamente todo el fondo blanco que impedía tener una mayor tasa de acierto al predecir los colores de la ropa. Este cambio ha permitido una ejecución unas 3 veces más rápida en el test del $K-means$ del `main.py` y de unas 5 veces para el caso del test del KNN. Veamos un ejemplo de una imagen antes y después de aplicar el cambio, y los resultados que obtenemos:

Predicted colors: ['White', 'Black', 'Orange', 'Red']
Real colors: ['Black']

Predicted colors: ['Black', 'White', 'Orange', 'Grey']
Real colors: ['Black']

Hay que tener en cuenta también que hemos diseñado el programa de manera que la lista de colores que devuelve para cada una de las imágenes está ordenada de acuerdo con la prevalencia del color en la imagen. Es decir, los colores que tengan más píxeles asociados serán aquellos que aparezcan primeros. Dicho esto, queda en evidencia la mejora de cara a la clasificación de colores. Donde antes obteníamos el color blanco como color principal, ahora obtenemos el color negro (es decir, el correcto); y donde antes teníamos algunos colores de la lista que en nada se correspondían con la prenda principal de la imagen (en nuestro ejemplo el color rojo), ahora tenemos una lista mayoritariamente acertada (con la excepción de los ya mencionados blanco, gris y naranja).

Las precisiones exactas ahora nos han quedado como sigue:

Global accuracy for colors:	0.5241180742371072
Accuracy for Red:	0.4462809917355372
Accuracy for Orange:	0.0888030888030888
Accuracy for Brown:	0.5344827586206896
Accuracy for Yellow:	0.7083333333333334
Accuracy for Green:	0.734375
Accuracy for Blue:	0.8383838383838383
Accuracy for Purple:	0.5510204081632653
Accuracy for Pink:	0.5603448275862069
Accuracy for Black:	0.8295454545454546
Accuracy for Grey:	0.2617924528301887
Accuracy for White:	0.21193666260657734

Hemos aumentado ligeramente la precisión para la predicción de color. Otra observación al hacer este cambio es el número de *clusters* que el programa considera óptimo. Vamos a comparar en una tabla el porcentaje de veces que nuestra imagen tiene 1 sólo color, 2 colores, etc. hasta 5 colores y lo vamos a comparar con el número real que encontramos en las etiquetas de *Ground Truth*.

	1	2	3	4	5
Etiquetas reales	56 %	33 %	9 %	2 %	0 %
Sin recortar la imagen	0 %	14 %	47 %	33 %	6 %
Recortando la imagen	2 %	20 %	52 %	24 %	2 %

Aunque después de haber recortado la imagen el número de colores identificados se aproxima más que considerando toda la imagen, sigue habiendo una diferencia notable. Este ajuste lo discutiremos más adelante, al modificar el parámetro de la máxima K en el algoritmo $K - means$.

4 | Hyperparameter tunning

A continuación se muestran los efectos de cambiar algunos de los parámetros que hemos considerado que podrían ser más relevantes a la hora de alterar los resultados en nuestros algoritmos. Entre ellos se encuentran el parámetro k y el número de imágenes de test.

4.1 | Modificación del número de imágenes de test

Dado que el número de imágenes propuestas para hacer un test en primera instancia eran 50 y hemos pensado que no eran suficientes, hemos probado con 200 para ver el efecto que tenían en la tasa de aciertos (*accuracy*) de los algoritmos $K-means$ y KNN.

El resultado ha sido que simplemente no afecta en la *accuracy*, al menos para este conjunto test y problema dado. El único factor que hemos encontrado diferente ha sido una suavización de las funciones de error. Con 50 imágenes los errores tienen formas más abruptas que con 200. Sin embargo, el porcentaje de acierto se mantiene.

4.2 | Número de vecinos k

También hemos experimentado cambiando el número de vecinos sobre los que implementar el KNN. En el archivo se proponía una $K = 5$ en la función `predict` de KNN, pero hemos probado a implementarla con $K = 3$ y hemos obtenido resultados aún más satisfactorios, con una P@K de unos 93 puntos, cuando con $K = 5$ rondaba los $86-89$ (según la incialización). Si nos fijamos en la precisión global del clasificador para las clases, vemos que inicialmente era de $\approx 0{,}9048$. Con $K = 3$ vecinos pasa a ser del $\approx 0{,}9095$, una pequeña mejora. También hemos probado con $K = 8$ vecinos, pero los resultados no han sido satisfactorios en cuanto a precisión global para las clases, obteniendo un valor de $\approx 0{,}8954$.

4.3 | Número máximo de *clusters* k

Como hemos visto en la sección 3.2, el número de colores que considera nuestro clasificador queda bastante por encima del número real de colores que hay en la imagen. Si restringimos el número máximo de clases que pueda considerar el algoritmo, obtenemos una precisión mayor. Nos queda:

Global accuracy for colors:	0.595321270488984
Accuracy for Red:	0.5625
Accuracy for Orange:	0.0859375
Accuracy for Brown:	0.6290322580645161
Accuracy for Yellow:	0.6785714285714286
Accuracy for Green:	0.7804878048780488
Accuracy for Blue:	0.8689655172413793
Accuracy for Purple:	0.6153846153846154
Accuracy for Pink:	0.7313432835820896
Accuracy for Black:	0.9281767955801105
Accuracy for Grey:	0.45614035087719296
Accuracy for White:	0.21199442119944212

Es decir, la precisión más alta que hemos podido conseguir hasta el momento para la clasificación de colores (casi un 10 % más que con los parámetros y opciones iniciales).

4.4 | Gráficas rendimiento después de las mejoras

Si volvemos a analizar las gráficas obtenidas, pero ahora con todas las mejoras aplicadas observamos que aunque la forma general de las tres es parecida a las anteriores, hemos hecho aumentar la precisión total en un 1.3 % aproximadamente (hemos pasado de un 0,9153349 a un 0,92930866).

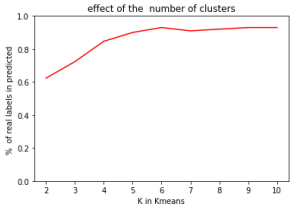

Figura 4: Effect of the number of clusters

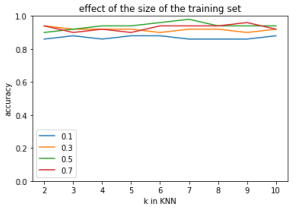

Figura 5: Effect of the size of the training set

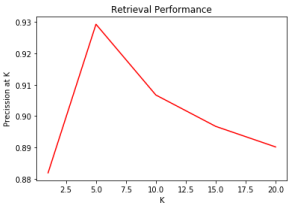

Figura 6: Precision at K

5 | Conclusión

Podemos concluir que, con la implementación básica inicial que habíamos hecho, obteníamos resultados bastantes buenos para la clasificación de forma (alrededor de un 90 % de precisión), pero no muy adecuados para la clasificación de color (alrededor de un 50 %).

Esto nos ha llevado a centrarnos en la mejora de la clasificación de color. El primer cambio que hemos realizado ha sido el de modificar la imagen que el clasificador analiza para que ésta pase a ser únicamente la parte central, donde en la gran mayoría de casos se encuentra el producto. Este cambio, aunque no ha mejorado significativamente la precisión de la clasificación del color, ha hecho que el programa sea mucho más rápido.

El segundo cambio respecto al algoritmo $k - means$ ha sido el de reducir el número máximo de clases que podía identificar, dado que en las etiquetas reales este número era considerablemente más bajo de lo que nuestro clasificador escogía como óptimo. Esto ha hecho que la precisión de color aumente en casi un 10 % respecto al valor inicial, quedando alrededor de un 60 %. Aunque sigue siendo un porcentaje relativamente bajo, hemos visto también que el motivo es la falta de herramientas más avanzadas para separar el producto de otros elementos de la imagen.

Por último, hemos reducido el número de vecinos que mira el algoritmo KNN, lo cual ha hecho subir ligeramente la precisión de la clasificación de la forma.

En resumen, la mejor configuración con nuestra implementación se da cuando añadimos la opción `'crop' : 'cropped'` al definir el objeto $K - means$, cuando reducimos el número màximo de *clusters* y vecinos a 3.