Inteligencia Artificial
Grado en Ingeniería Informática en Sistemas de Información
ENSEÑANZAS PRÁCTICAS Y DE DESARROLLO

EPD5: Machine Learning – Clustering

Objetivos

- Implementación en Python de un algoritmo de clustering.

Bibliografía Básica

The elements of statistical learning: Data Mining, Inference and Prediction. Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Springer, 2017. Disponible online en: https://web.stanford.edu/~hastie/ElemStatLearn/

Ejercicios

Implementar el algoritmo de clustering K-means para un dataset en 2D (fichero "`ex7data2.mat`"). Este algoritmo se basa en la idea de agrupar instancias similares en "clusters". Se trata de un algoritmo iterativo que parte de unos centroides iniciales y los va refinando repetidamente agrupando las instancias en sus centroides más cercanos (**EJ1**) y recalculando los centroides (sus medias) en base a esas asignaciones (**EJ2**). Es importante tener en cuenta que las soluciones del K-means dependen de los centroides iniciales, es por eso que en la práctica en ocasiones se ejecuta el algoritmo varias veces con diferentes inicializaciones de los centroides y se selecciona la solución con el menor valor de la función objetivo que se está minimizando.

EJ1. Implementar la función `findClosestCentroids.py` para asignar cada instancia del conjunto de training a su centroide más cercano, partiendo de los siguientes centroides iniciales: [[3, 3], [6,2], [8, 5]], es decir, K se inicializa a 3. Se usará la distancia euclídea para determinar qué centroide está más cerca. La función debe devolver un array de *m* enteros (siendo *m* el número de instancias del dataset). Cada elemento de dicho array tendrá un valor de 0 a K-1 representando el centroide asociado (recuerde que en Python los índices empiezan en 0).

EJ2. Implementar la función `computeCentroids.py` para actualizar los centroides según la media de las instancias que fueron asignadas en el ejercicio anterior a cada centroide.

EJ3. Ejecutar la función `runKmeans.py` donde se llama a las dos funciones anteriores para ver cómo funciona el algoritmo usando los centroides iniciales especificados en el EJ1 y 10 iteraciones. Visualizar e interpretar los resultados: centroides iniciales vs. Centroides finales.

EJ4. Existen algoritmos que buscan la inicialización óptima de los centroides, como el K-Means++. En este caso, implemente una inicialización los centroides seleccionando al azar K instancias del conjunto de entrenamiento en `kMeansInitCentroids.py`. Se recomienda modificar aleatoriamente los datos usando randint de numpy y seleccionar las K instancias. Después volver a ejecutar el algoritmo completo y ver cómo varían los centroides encontrados.

EJ5. Implementar el algoritmo elbow en `elbowMethod.py` para determinar cuál sería el K óptimo para este conjunto de datos. Pruebe con K desde 1 hasta 10. Visualice la gráfica función coste versus número de clusters.

Problemas

P1. Implementar el algoritmo K-Means usando la librería sklearn con el método "k-means++" para inicializar los centroides. Uitlice los mismos datos que en los Ejercicios. Completar el código para poder determinar el número óptimo de clusters para el conjunto de datos tanto de forma visual como numéricamente con la librería kneed. De forma visual, consiste en dibujar los valores de la función objetivo que se minimiza (a los que accedemos mediante el atributo `.inertia_`) para los valores de K desde 1 hasta 10. De forma numérica, puede consultar el uso de la librería kneed en: https://kneed.readthedocs.io/en/stable/

P2. Se usará el algoritmo K-Means implementado en los ejercicios anteriores (**EJ1-EJ4**) para comprimir una imagen ("`bird_small.mat`"). La imagen está codificada con RGB encoding (255 valores) y se pretende reducir a solo 16 valores, reduciendo así el número de colores presentes en la imagen a solamente los más comunes. Una vez cargada la imagen, se crea

una matriz tridimensional con los dos primeros índices las posiciones de los píxeles de la imagen y en el tercer índice el color. Para ello:

1. En primer lugar, debéis transformar esta matriz para que todos sus valores estén en el rango [0, 1].
2. Después, redimensionar esta matriz en una matriz de m filas y 3 columnas, donde m es el número de pixeles.
3. Aplicar el algoritmo K-Means a esta matriz para encontrar los 16 colores que mejor agrupan los pixeles.

Una vez obtenido los 16 colores, para comprimir la imagen primero encontrar los centroides más cercanos a cada pixel, y reemplazar cada pixel por dicho centroide. Una vez comprimida, recuperar el tamaño original de la imagen y guardar la imagen comprimida. Podéis ver el efecto de la compresión imprimiendo la imagen original y la imagen comprimida.

P3. Repetir el **P2** usando en este caso el algoritmo K-Means de la librería sklearn.