MINERÍA DE DATOS
PROBLEMARIO PARTE 2 – UNIDAD 1

Realiza un script en lenguaje Python que haga lo siguiente:

1. Importa el Dataset Iris

2. Extrae en 3 diferentes arreglos (Xc1, Xc2, Xc3) los valores de cada clase pero solo considera las 2 primeras características (2 primeras columnas)

3. De cada uno de los 3 arreglos extrae el 70 por ciento de los datos y genera 3 arreglos nuevos llamados Xc1_70, Xc2_70 y Xc3_70, haz esto con la función: np.random.choice(arg1,arg2,replace=False) la cual genera índices aleatorios sin reemplazo dentro de un rango, por ejemplo si se requieren 4 valores aleatorios entre 1-10 entonces esta función podría regresar [3,1,7,9], si hay dudas puedes documentarte en internet cómo usar esta función. Esto se hace para generar los índices del 70% de los valores a extraer de cada arreglo.

4. El 30 porciento de los datos restantes de cada arreglo guárdalos en tres arreglos distintos llamados Xc1_30, Xc2_30 y Xc3_30, puedes hacer esto con la función:
np.setxor1d(Xc_index_70,Xc_all_index) que extrae los valores distintos que no están en ambos arreglos, es decir, en este caso, Xc_index_70 es un arreglo que contiene los índices del 70% de los valores extraídos en el paso anterior y Xc_all_index contiene todos los índices del arreglo (arreglos generados en el paso 2), por lo que la función np.setxor1d(,) regresa los valores de los índices del 30% de los datos restantes.

5. Con los índices del 70% de los datos de cada arreglo, esto es con Xc1_index_70, Xc2_index_70, Xc3_index_70 genera los arreglos de etiquetas de clase correspondientes, es decir, genera Yc1_70, Yc2_70 y Yc3_70. Y lo mismo para los arreglos de las etiquetas de clase del 30% de datos restantes, es decir, Yc1_30, Yc2_30 y Yc3_30.

6. En este punto se deben de tener 12 arreglos, los primeros 6 son: Xc1_70, Xc2_70, Xc3_70 con sus respectivas etiquetas de clase Yc1_70, Yc2_70 y Yc3_70. Los otros 6 son: Xc1_30, Xc2_30, Xc3_30 con sus respectivas etiquetas de clase Yc1_30, Yc2_30 y Yc3_30.

7. Une los arreglos correspondientes al 70% de los datos de cada clase con la función np.concatenate(arg1,arg2,axis=0), esto es, une los arreglos Xc1_70, Xc2_70 y Xc3_70, esto para formar un único arreglo llamado Xtrain. Haz lo mismo con los arreglos unidimensionales correspondientes a las etiquetas de clase del 70% de los datos, esto es, concatena en un mismo arreglo Yc1_70, Yc2_70 y Yc3_70, esto para formar un único arreglo llamado Ytrain.

8. Une los arreglos correspondientes al 30% de los datos de cada clase con la función np.concatenate(arg1,arg2,axis=0), esto es, une los arreglos Xc1_30, Xc2_30 y Xc3_30, esto para formar un único arreglo llamado Xtest. Haz lo mismo con los arreglos unidimensionales correspondientes a las etiquetas de clase del 30% de los datos, esto es, concatena en un mismo arreglo Yc1_30, Yc2_30 y Yc3_30, esto para formar un único arreglo llamado Ytest.

9. Inicializa una variable k e igualala a 3, (K=3), y selecciona aleatoriamente 'k' filas de la matriz Xtrain (las filas no se deben

repetir), puedes emplear la función np.random.choice(arg1,arg2,replace=False) para generar los índices de las 'k' filas a seleccionar.

10. Una vez seleccionadas las filas, guarda las 'k' filas en una matriz (arreglo bidimensional) llamada 'centroids' que debe de ser de orden k-filas por 2-columnas (kx2).

11. Crea una matriz de ceros llamada 'square_distance' cuyo tamaño sea del mismo número de filas que Xtrain y k-columnas, si el número de filas de Xtrain es F entonces el orden de 'square_distance' es Fxk. Emplea la función np.zeros() para hacer esto.

12. Calcula la DISTANCIA EUCLIDIANA CUADRADA entre CADA CENTROIDE (cada fila de la matriz 'centroids') y TODAS LAS FILAS de la matriz Xtrain. Estas distancias guárdalas en cada columna de la matriz 'square_distance', es decir, en la i-ésima columna de 'square_distance' (recuerda que tiene k columnas) estarán las 'F' DISTANCIAS EUCLIDIANAS entre TODAS LAS FILAS DE 'Xtrain' y la i-ésima fila de 'centroids'. NOTA IMPORTANTE: Recuerda que la DISTANCIA EUCLIDIANA CUADRADA es ||x-c||^2 donde ||.|| es la distancia euclidana (raíz cuadrada de la suma de las diferencias al cuadrado), por lo que ||x-c||^2 (DISTANCIA EUCLIDIANA CUADRADA) es solamente la suma de las diferencias al cuadrado (sin raíz cuadrada). Para el CÁLCULO DE LA <<OPERACIÓN>> DE LA DISTANCIA EUCLIDIANA CUADRADA puedes utilizar np.sum(arg1,axis=1) para que no tengas la necesidad de emplear ningún bucle (for, while, etc.). SIN EMBARGO para hacer el cálculo de la DISTANCIA EUCLIDIANA CUADRADA ENTRE CADA CENTROIDE Y TODAS LAS FILAS, en el proceso de hacerlo para CADA UNO DE LOS 'k' CENTROIDES SÍ PUEDES UTILIZAR UN ciclo for que itere k veces.

13. Ordena de menor a mayor los valores de CADA FILA de la matriz 'square_distance' y GENERA LA MATRIZ DE ÍNDICES DE LAS COLUMNAS DE LAS POSICIONES ORIGINALES DE LOS VALORES, ejemplo con una matriz 'square_distance' de F=5 filas y k=3 columnas:

SITUACION ORIGINAL (ANTES DE ACOMODAR DE MAYOR A MENOR LOS VALORES DE LAS FILAS)

Índices de las columnas: 0 1 2

square_distance = [1.2, 3.2, 0.5]
[0.2, 1.2, 6.5]
[3.8, 5.2, 2.5]
[3.2, 2.1, 9.5]
[4.2, 0.6, 0.1]

DESPUÉS DE HACER EL SORT (ACOMODO):

Índices de las columnas: 0 1 2

square_distance_sorted= [0.5, 1.2, 3.2]
[0.2, 1.2, 6.5]
[2.5, 3.8, 5.2]
[2.1, 3.2, 9.5]
[0.1, 0.6, 4.2]

matriz_de_indices = [2, 0, 1]
[0, 1, 2]
[2, 0, 1]
[1, 0, 2]
[2, 1, 0]

La matriz de índices contiene la posición (índice) que tenían los valores de las filas de la matriz square_distance_sorted originalmente antes de ser acomodados. Esto significa que la

primera columna de la 'matriz de índice' nos índica cuales Filas (puntos, vectores, patrones, etc.) de Xtrain están más cercanos a cada uno de los k=3 centroides.

14. Guarda en un arreglo llamado 'min_distance' los valores de la primera columna de 'matriz_de_indices'.

15. Genera tres arreglos, llamados min_k0, min_k1 y min_k2 en donde guardes los valores de cada fila de Xtrain que estén más cercanos al í-ésimo centroide. Para esto emplea 'min_distance'. Por ejemplo, en la matriz de índices, en la primera columna, los valores son:
[2,0,2,1,2], por lo que en min_k0 se guardará la fila 1 de Xtrain, en min_k1 se guardará la fila 3 de Xtrain, y en min_k2 se guardarán las filas 0, 2 y 3 de Xtrain.

16. Emplea un tipo de gráfica llamada 'scatter' para visualizar los puntos (vectores, patrones, etc.) de Xtrain pertenecientes a cada uno de los k-centroides. Debes pintar con diferentes colores cada grupo diferente de k-puntos.

17. Grafica los puntos correspondientes a los k-centroides de un color diferente a los puntos y con un tamaño mayor.

NOTA: Las funciones que no específico cómo utilizarlas pueden buscar un ejemplo de su uso en estas páginas:

PARA GRAFICAR-MATPLOTLIB:
https://matplotlib.org/index.html
y en particular para el scatter plot:
https://matplotlib.org/gallery/shapes_and_collections/scatter.html#sphx-glr-gallery-shapes-and-collections-scatter-py

PARA MANEJO DE MATRICES -NUMPY:
DOCUMENTACIÓN COMPLETA: https://docs.scipy.org/doc/
GUÍA RÁPIDA:
https://docs.scipy.org/doc/numpy/user/quickstart.html
PARA USUARIOS DE MATLAB:
https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html

EL SCRIPT SE DEBE DE ENVIAR COMPLETAMENTE FUNCIONAL ANTES DE LAS 11:59 PM DEL JUEVES 14 DE JUNIO DEL 2018

CORREO ELECTRÓNICO: stoviasa@upv.edu.mx
ASUNTO: **MDD-PROB-U1-ApellidoPaterno-ApellidoMaterno**

NOTA-IMPORTANTÍSIMA: SI EL ASUNTO NO VIENE ASÍ, TAL CUAL, NO LO RECIBO. DEBEN DE RESPETAR EL FORMATO DEL ASUNTO. SOLO RECIBO UNA VERSIÓN (LA PRIMERA QUE ENVÍEN) ASÍ QUE NO SE EQUIVOQUEN AL ENVIAR LA PRIMERA VEZ.