Unidad Profesional Interdisciplinaria de Ingeniería campus Tlaxcala

INSTITUTO POLITÉCNICO NACINAL

FUNDAMENTOS DE INTELIGENCIA ARTIFICIAL.

Algoritmo de K Means (Clustering) aplicado a la base de datos de Handwritten.

Equipo:

Navil Pineda Rugerio.

Luis Daniel Islas García.

Miguel Ángel Sánchez Zanjuampa

16 de junio, 2023

Resumen
El algoritmo de K-means clustering es una técnica de aprendizaje no supervisado, en la que los datos no tienen etiquetas, por lo que se busca agrupar elementos según sus características y estructura interna. En esta práctica se realizó un breve análisis de los datos utilizados para el entrenamiento del modelo, y la agrupación en clusters, para identificar cuales eran los datos que más convenían mostrar en las gráficas que más adelante se muestran, así como los criterios de paro del algoritmo. Después se diseña el algoritmo, siguiendo la forma en la que opera K means. Por último, se muestran gráficos en donde se puede apreciar la agrupación de los datos.

Palabras clave: Algoritmo de clasificación, k means clustering, aprendizaje no supervizado, cluster.

Abstract
The K-means clustering algorithm is an unsupervised learning technique, in which the data do not have labels, so it seeks to group elements according to their characteristics and internal structure. In this practice, a brief analysis of the data used for the training of the model and the grouping in clusters was carried out, to identify which data were the most convenient to show in the graphs shown below, as well as the stopping criteria of the algorithm. Then the algorithm is designed, following the way in which K means operates. Finally, graphs are shown where the grouping of the data can be appreciated.

Keywords: Classification algorithm, k means clustering, unsupervised learning, cluster.

I. MARCO TEÓRICO.

"El algoritmo de K-means clustering es una técnica de aprendizaje no supervisado utilizada para agrupar un conjunto de datos en k grupos o clústeres basados en su similitud. Es uno de los algoritmos de clustering más populares debido a su simplicidad y eficiencia en cuanto a implementación.

En la clasificación no supervisada los datos no tienen etiquetas, por lo que se buscan formas de agrupar elementos según sus características, su estructura interna. El algoritmo de k means agrupa objetos en k grupos basándose en sus características. El agrupamiento se realiza minimizando la suma de distancias entre cada objeto y el llamado *centroide* de su grupo o clúster. El centroide es un punto representativo de cada clúster, este algoritmo inicia con un conjunto inicial de centroides elegidos de forma aleatoria.

El algoritmo de k means opera de la siguiente forma:

1. Selecciona k puntos aleatorios como centroides iniciales, estos representan los "centros", en un primer momento, de los clústeres.
2. Para cada punto en el conjunto de datos, calcula la distancia euclidiana entre el punto y cada centroide. Asigna el punto al clúster cuyo centroide esté más cercano.
3. Después de asignar todos los puntos a los clústeres, se recalcula el centroide de cada clúster tomando la media de todos los puntos asignados a ese clúster. El centroide se actualiza como el nuevo centro del clúster.
4. Los pasos 2 y 3 anteriores se repiten iterativamente hasta que se de un caso o criterio de convergencia, que puede ser cuando los centroides ya no cambian significativamente, cuando se ha alcanzado un número máximo de iteraciones, o alguna otra condición aceptada por este algoritmo.
5. Una vez que se tiene el criterio de convergencia, lo centroides finales representan los centros de los clústeres y los demás punto están asignados a sus correspondientes.

Es importante tener en cuenta algunas consideraciones sobre el algoritmo de K-means clustering:

- El número de clústeres, k, debe ser especificado antes de ejecutar el algoritmo. Una elección incorrecta de k puede conducir a resultados subóptimos.
- La elección inicial de los centroides puede influir en los resultados finales. Una inicialización aleatoria puede llevar a diferentes resultados en diferentes ejecuciones.
- El algoritmo es sensible a valores atípicos y a la escala de datos, ya que pueden influir en la posición de los centroides y el cálculo de las distancias, respectivamente.

Objetivo

El objetivo de esta práctica es diseñar un algoritmo k means clustering, en el lenguaje de programación Python, para agrupar un conjunto de datos de Handwritten Dataset, en 10 clústers que corresponden a un número entre 0 y 9.

II. Materiales

- Bibliotecas y herramientas para leer los datos de un archivo, para el manejo de arreglos y la visualización de los datos.
- Funciones y clases para la lectura del archivo y el algoritmo de k means clustering
- IDE Visual Studio Code.

III. Desarrollo

A. Librerías

Se utilizan las librerías numpy para el manejo de arreglos, matplotlib para graficar los datos, la librería pandas para mostrar los datos y facilitar su análisis. Por último, se importa random para una aleatorización de los datos, previa a la agrupación.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import random
```

B. Cargar el DataSet.

Para cargar la información del archivo se creó una clase que tiene un método para leer el archivo y guardar la información en una matriz que contiene todas las imágenes, y un vector que contiene todas las clases correspondientes a las imágenes.

El constructor de la clase no recibe ningún parámetro, y sus atributos se inicializan en None, para que en los métodos posteriores se guarde la información cargada del archivo. También se declaran los métodos setters y getters que devuelven los datos ya aleatorizados.

```
# Leer el archivo de texto
class Data:
    def __init__(self):
        self._data = None
        self._target = None
        self._randomData = None
        self._randomTarget = None

    # Metodos getters
    def getRandomData(self):
        return self._randomData

    def getRandomTarget(self):
        return self._randomTarget
```

Para cargar el archivo se tiene el método load_data, el cual lee el archivo de texto plano y lo separa por líneas, separando cada token cuando encuentra un coma (,) cada una de las líneas del archivo ya sin las comas se guarda en una lista llamada values, esta lista contiene el vector de las imágenes y en la última posición contiene la clase a la que pertenece.

```
# Funcion para leer el archivo y guardar las imagenes y sus clases
def load_data(self):
    with open("optdigits.tes", "r") as datos:
        valores = []
        for linea in datos:
            valores.append([x for x in linea.strip().split(",")])

        # Crear matriz para guardar los valores de data
        data = np.zeros((len(valores), 64))
        for i in range(len(valores)):
            data[i] = valores[i][:64]

        self._data = np.copy(data)

        # Crear matriz para guardar los valores de target (clases)
        target = np.zeros((len(valores)))
        for i in range(len(valores)):
            target[i] = valores[i][64]

        self._target = np.copy(target)


    print("Se cargaron los datos")
```

Para guardar la información de las imágenes y las clases se guarda en una matriz llamada data los valores de values de la posición 0 a la posición 64, lo que corresponde a la imagen, y para las clases, se guardan en un arreglo los valores en la posición 65, que corresponden al número que representa la imagen.

C. Aleatorizar Datos.

Para aleatorizar los datos ya guardados se tiene el método randomizer.

```
# Funcion para aleatorizar la informacion guar
def randomizer(self):
    # Crear matriz auxiliar para permutar fila
    numDatos, numChar = self._data.shape

    randomData = np.copy(self._data)
    # Crear arreglo auxiliar para permutar cla
    numTarget = self._target.shape

    randomTarget = np.copy(self._target)

    # Tomar un valor random desde 0 hasta numD
    a, b = shuffle(randomData, randomTarget)

    # Guardar la matriz con filas intercambiad
    self._randomData = np.copy(a)
    self._randomTarget = np.copy(b)
    print("Datos aleatorizados")
```

Primero, se obtienen las dimensiones del arreglo "data" utilizando la función "shape". Esto permite conocer el número de datos y de características (o columnas) de la matriz. A continuación, se crea un arreglo nuevo llamado "randomData" utilizando la función "np.copy()", que crea una copia en memoria del arreglo original.

Lo mismo se hace con el arreglo "target", excepto que aquí sólo se necesita conocer el número de filas.

Después se utiliza la función shuffle que será explicada a continuación.

```
def shuffle(data, target):
    # Hacer un shuffle de las listas
    dataRand, targetRand = shuffle_lists(data, target)
    return dataRand, targetRand
✓ 0.0s
```

Esta función transforma las copias de la matriz y el arreglo en listas y devuelve los datos mezclados tanto de las imágenes, como de las clases. Para ello también se utiliza la función shuffle_lists.

```
def shuffle_lists(lst, lst2):
    n = len(lst2)
    for i in range(n - 1, 0, -1):
        j = random.randint(0, i)
        lst[i], lst[j] = np.copy(lst[j]), np.copy(lst[i])
        lst2[i], lst2[j] = lst2[j], lst2[i]
    return lst, lst2
```

Esta función obtiene un número aleatorio de 0 a n, donde n es la longitud de la lista de datos, y se van intercambiando los datos tanto de las imágenes como de las clases, finalmente se devuelven esas dos listas, que ya son las listas aleatorizadas.

D. Leer la información del archivo.

Para cargar el archivo, instanciamos una variable de tipo de la clase Data, y llamamos al método load_data. Después se llama al método randomizer para aleatorizar los datos ya guardados.

```
data = Data()
data.load_data()
data.randomizer()
```

Para obtener los datos ya aleatorizados se utiliza el método getRandomData, y se muestran los resultados, para este caso solo son las primeras dos imágenes.

```
# Devuelve los imagenes aleatorias con las que se va a trabajar
datos = data.getRandomData()
datos[:2]
✓ 0.0s
array([[ 0.,  0.,  2., 15., 15.,  4.,  0.,  0.,  0.,  0., 11., 10., 14.,
         9.,  0.,  0.,  0.,  0.,  1.,  0., 11.,  9.,  0.,  0.,  0.,  0.,
         0.,  3., 15.,  4.,  0.,  0.,  0.,  0.,  1., 16., 16., 14.,  6.,
         0.,  0.,  0.,  0.,  8., 13.,  6.,  1.,  0.,  0.,  0.,  0.,  9.,
         7.,  0.,  0.,  0.,  0.,  0.,  1., 15.,  2.,  0.,  0.,  0.],
       [ 0.,  0., 10.,  9.,  0.,  0.,  0.,  0.,  0.,  0.,  8., 16.,  2.,
         0.,  0.,  0.,  0.,  0.,  8., 16.,  6.,  0.,  0.,  0.,  0.,  0.,
         5., 16., 13.,  1.,  0.,  0.,  0.,  0.,  1.,  5., 14.,  6.,  0.,
         0.,  0.,  0.,  0.,  0.,  8., 11.,  0.,  0.,  0.,  0.,  8., 12.,
         9., 16.,  6.,  4.,  0.,  0.,  7., 16., 16., 16., 16., 14.]])
```

También se muestran los resultados de las clases aleatorizadas.

```
# Devuelve las clases de las imager
target = data.getRandomTarget()
target
```

E. Data Frame para visualizar los datos.

Después se utiliza un Data Frame de la librería pandas para mostrar los datos en una tabla, se muestran los valores de los 64 pixeles de las imágenes y las clases a las que corresponden.

```
# Dataframe para visualizar los datos
df = pd.DataFrame(datos)
#df["target"] = target
df.head(10)
```
✓ 0.1s

	0	1	2	3	4	5	6	7	8	9	…	54	55	56	57	58	59	60	61	62	63
0	0.0	0.0	2.0	15.0	15.0	4.0	0.0	0.0	0.0	0.0	…	0.0	0.0	0.0	0.0	1.0	15.0	2.0	0.0	0.0	0.0
1	0.0	0.0	10.0	9.0	0.0	0.0	0.0	0.0	0.0	0.0	…	6.0	4.0	0.0	0.0	7.0	16.0	16.0	16.0	16.0	14.0
2	0.0	0.0	5.0	16.0	14.0	1.0	0.0	0.0	0.0	0.0	…	0.0	0.0	0.0	0.0	3.0	15.0	15.0	5.0	0.0	0.0
3	0.0	0.0	0.0	12.0	7.0	0.0	0.0	0.0	0.0	0.0	…	0.0	0.0	0.0	0.0	0.0	12.0	9.0	0.0	0.0	0.0
4	0.0	1.0	15.0	16.0	4.0	0.0	0.0	0.0	0.0	9.0	…	12.0	1.0	0.0	0.0	13.0	16.0	16.0	15.0	11.0	1.0
5	0.0	0.0	2.0	14.0	13.0	0.0	0.0	0.0	0.0	0.0	…	0.0	0.0	0.0	0.0	0.0	16.0	5.0	0.0	0.0	0.0
6	0.0	0.0	0.0	7.0	14.0	16.0	6.0	0.0	0.0	0.0	…	0.0	0.0	0.0	0.0	0.0	9.0	6.0	0.0	0.0	0.0
7	0.0	0.0	4.0	12.0	1.0	0.0	0.0	0.0	0.0	0.0	…	12.0	0.0	0.0	0.0	5.0	13.0	16.0	11.0	1.0	0.0
8	0.0	0.0	4.0	15.0	16.0	11.0	0.0	0.0	0.0	0.0	…	0.0	0.0	0.0	0.0	6.0	11.0	0.0	0.0	0.0	0.0
9	0.0	0.0	13.0	14.0	8.0	1.0	0.0	0.0	0.0	1.0	…	0.0	0.0	0.0	1.0	13.0	14.0	12.0	12.0	0.0	0.0

10 rows × 64 columns

F. Preparar datos.

Una vez visualizados, ya podemos preparar los datos que se van a utilizar para el entrenamiento en este caso solo necesitamos las imágenes vectorizadas, no ocupamos sus clases, y notamos que es mejor trabajar con datos en las posiciones centrales de la imagen (aproximadamente en columnas de 20 a 40), ya que son esas regiones en las que se encuentra escrito el número. En estas secciones identificadas se puede notar que el valor del pixel es diferente de 0 en la mayoría de los casos.

G. Definición de K y atributos a analizar.

Ahora podemos definir los valores de las columnas centrales que son aquella que más tienen variaciones de datos, y que son las que analizaremos en las gráficas posteriores.

También definimos el valor de K, es decir el número de clústeres que queremos, en este caso 10.

```
K=5
v1 = 26
v2=35
v3=30
```

H. Obtener el valor μ.

En este caso μ representa el centroide aleatorio con el que inicia el algoritmo, para ello se tiene una función que inicializa un vector aleatorio, de 64 posiciones, con números de 0 a 16, ya que las imágenes obtenidas de la base de datos son matrices cuyo valor mínimo es 0 y su valor máximo es 16.

```
mu=[]
for i in range(0,K):
    numrandom = []
    for j in range(64):
        x = np.random.randint(16)
        numrandom.append(x)
    mu.append(numrandom)

mu=np.array(mu)
mu
```

```
array([[ 1, 10,  7,  2, 15, 12,  1,  8, 14,  7, 14, 10, 15,  3, 12, 14,
         0, 13,  0, 12,  4, 15, 12,  3, 10,  7,  2,  8, 13, 15,  3,  2,
        10,  4,  6, 14,  2, 13, 13,  1,  5,  2,  1,  4, 10,  7, 13,  0,
        13, 11,  1,  1, 13,  9,  6, 11,  2,  1, 10, 14, 13,  2, 12,  1],
```

I. Mostrar Data Frame con la clase a la que corresponde el número.

Se muestra un dataframe lleno de ceros que más adelante será utilizado y modificado.

✓ 0.0s

	0	1	2	3	4	5	6	7	8	9	…	55	56	57	58	59	60	61	62	63	64
0	0.0	0.0	11.0	14.0	5.0	0.0	0.0	0.0	0.0	6.0	…	0.0	0.0	0.0	11.0	8.0	4.0	5.0	16.0	1.0	7
1	0.0	0.0	0.0	1.0	13.0	1.0	0.0	0.0	0.0	0.0	…	0.0	0.0	0.0	0.0	2.0	11.0	11.0	1.0	0.0	2
2	0.0	0.0	3.0	12.0	15.0	14.0	3.0	0.0	0.0	1.0	…	0.0	0.0	0.0	3.0	12.0	16.0	13.0	0.0	0.0	2
3	0.0	3.0	14.0	14.0	16.0	16.0	10.0	0.0	0.0	9.0	…	0.0	0.0	3.0	15.0	16.0	4.0	0.0	0.0	0.0	10
4	0.0	0.0	2.0	12.0	16.0	10.0	0.0	0.0	0.0	0.0	…	0.0	0.0	0.0	3.0	13.0	16.0	3.0	0.0	0.0	2

J. Actualización del centroide, para un determinado número de iteraciones.

Ahora debemos encontrar el mejor centroide, esto se hace calculando la distancia euclideana entre cada observación de la base de datos y los centroides que actualmente se han encontrado. Por ello se itera tanto cada uno de los datos, como cada uno de los 10 centroides propuestos. En concreto, es la siguiente formula.

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{n,k} ||x_n - \mu_k||^2$$

En Python lo haremos de la siguiente manera.

Inicializamos variables y listas.

```
plot_counter=0
J_plot=[]
for i in range(1,100):
    J=0
```

Se recorre el data frame que contiene los datos, y se guardan en x el vector de una de las imágenes,

la imagen “actual”, después se calcula la distancia euclideana con cada uno de los centroides, y se obtiene la distancia mínima. Esto también es conocido como función de distorsión.

```
for n in range(0,df.shape[0]):
    x=np.array(df.iloc[n,0:64])
    mins=[]

    for k in range(0,K):
        distance=np.linalg.norm(x-mu[k])
        mins.append(distance**2)

    k_r=np.argmin(np.array(mins)) # Onten
    r[n,k_r]=1
```

Esta medida es la que se busca reducir, y se optimizará con respecto al valor r_nk, siguiendo la siguiente función.

$$r_{n,k} = \begin{cases} 1, & \text{if } k = argmin_j ||x_n - \mu_j||^2. \\ 0, & \text{otherwise.} \end{cases}$$

De acuerdo con esa función solo se rellenará con 1, el data frame previamente definido con 0s, solo en caso de que la distancia sea mínima.

K. Optimización con respecto al centroide.

Se utiliza la siguiente función para calcular el nuevo valor de μ, es decir, el nuevo valor del centroide, estos son ajustados al promedio del número de observaciones, es por ello que lleva como nombre K “means”.

$$\mu_k = \frac{\sum_n r_{n,k} x_n}{\sum_n r_{n,k}}$$

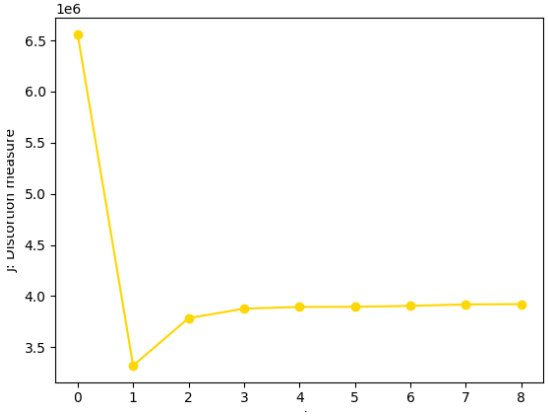
Este proceso se repetirá 100 veces que es el número máximo de iteraciones que se han propuesto.

IV. Resultados

Al ejecutar el algoritmo las 100 veces obtenemos los 9 centroides “definitivos” que hemos encontrado, según nuestra condición de paro, en este caso, un número máximo de iteraciones.

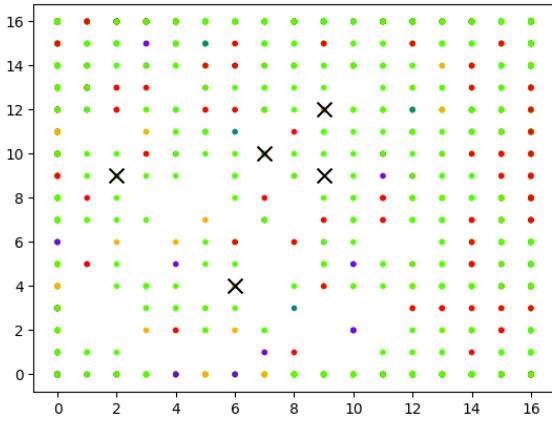
```
6558177.0
3321252.0
3785222.0
3877296.0
3894417.0
3895809.0
3904967.0
3919068.0
3921621.0
```

Ahora podemos observar y analizar cómo cambió la dispersión desde el centroide elegido de forma aleatoria, hasta el último centroide.

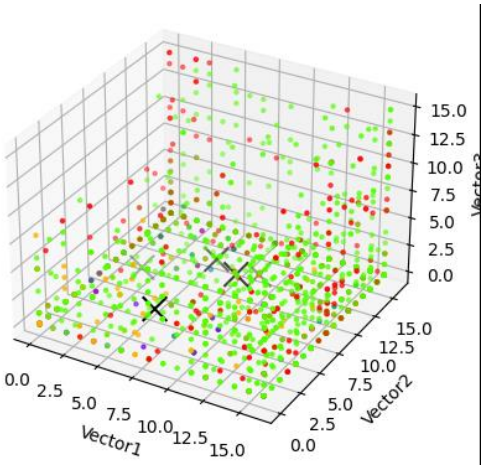


Como observamos hubo un momento en el que ya no cambió demasiado, por ello lo consideramos el centroide “definitivo”.

Ahora podemos graficar la distribución de los datos con un color diferente para cada clase. En esta parte solo graficamos dos pixeles de la imagen, pues solo podemos mostrar en 2D, y marcamos con una X el centroide.



O podemos visualizarlo en 3D.



En esta grafica podemos ver mucho mejor la forma en la que están distribuidas las clases y el centroide que justamente es el promedio de las clases.

V. Conclusiones

En conclusión, el algoritmo de K-means clustering es una técnica eficiente y ampliamente utilizada en el aprendizaje no supervisado para agrupar conjuntos de datos en clústeres basados en su similitud. En el contexto específico del objetivo planteado, el diseño de un algoritmo de K-means clustering en Python para agrupar el conjunto de datos de Handwritten Dataset en 10 clústeres que corresponden a los números del 0 al 9, se espera proporcionar una herramienta útil para el análisis de caracteres escritos a mano.

Referencias

[1] Education Ecosystem (LEDU). (2018, septiembre 12). *Understanding K-means clustering in machine learning*. Towards Data Science. https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1

[2] *kmeans*. (s/f). Unioviedo.es. Recuperado el 17 de junio de 2023, de https://www.unioviedo.es/compnum/laboratorios_py/kmeans/kmeans.html

[3] Rosas, L. Á. A. (2019, noviembre 11). *K Means: Programando el algoritmo desde cero en Python*. Medium. https://medium.com/@alcantararosas/k-means-programando-el-algoritmo-desde-cero-en-python-f633b3a1c125