K-Means

Bruno Gil Ramírez, Joana Michelle Rodriguez Gomez, Angel Molina Guillén, Angel Axel Mendez Meneses

Benemérita Universidad Autónoma de Puebla
Heroica Puebla de Zaragoza

Abstract
En esta práctica se implementó un sistema de recuperación de información utilizando el modelo K-Means. El objetivo de este sistema es clasificar un conjunto de documentos en diferentes categorías para poder recuperarlos de manera eficiente en el futuro. Para ello, se utilizó el algoritmo K-Means para agrupar los documentos en diferentes clusters basados en sus características y se asignó una etiqueta a cada uno de estos clusters.

1 Introduction

La implementación de sistemas de recuperación de información es una herramienta clave en la gestión y organización de grandes cantidades de datos. En el marco de este objetivo, se llevó a cabo una práctica enfocada en la implementación del modelo K-Means para la clasificación y agrupación de información en un sistema de recuperación.

Este modelo se basa en la agrupación de datos similares en "clusters" o grupos, lo que permite la identificación de patrones y relaciones entre los datos. El objetivo de la práctica fue la aplicación y configuración del modelo K-Means, con el fin de obtener una clasificación eficiente y efectiva de los datos a través de la técnica de clustering.

La implementación de este sistema de recuperación de información con el modelo K-Means representa un importante avance en la organización y gestión de grandes conjuntos de datos, permitiendo una rápida identificación y acceso a la información relevante para el usuario. A continuación se presentan los resultados y conclusiones de la práctica llevada a cabo.

2 El funcionamiento de k-means

2.1 ¿Qué es k-means?

k-means es un algoritmo de aprendizaje no supervisado que permite agrupar un conjunto de datos en k grupos o clusters, de acuerdo a su similitud. El objetivo del algoritmo es minimizar la distancia entre los puntos que pertenecen al mismo cluster y maximizar la distancia entre los puntos que pertenecen a clusters diferentes. El algoritmo funciona de la siguiente manera:

1. Se elige el número de clusters k que se desea obtener.

2. Se seleccionan k puntos al azar del conjunto de datos como los centroides iniciales de cada cluster.

3. Se asigna cada punto del conjunto de datos al cluster cuyo centroide está más cerca, usando alguna medida de distancia como la euclídea.

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{1}$$

4. Se recalculan los centroides de cada cluster como el promedio de los puntos asignados a ese cluster.

   Media aritmetica :

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2}$$

   donde $\bar{x}$ es la media aritmética de un conjunto de n observaciones $x_1, x_2, ..., x_n$.

   Moda :

$$Mo = \underset{x_i}{\text{argmax}}, f(x_i) \tag{3}$$

   donde Mo es la moda, es decir, el valor con mayor frecuencia en un conjunto de observaciones $x_1, x_2, ..., x_n$, y $f(x_i)$ es la frecuencia del valor x_i.

5. Se repiten los pasos 3 y 4 hasta que los centroides no cambien significativamente o se alcance un número máximo de iteraciones.

El algoritmo k-means es uno de los métodos más populares y sencillos para realizar agrupamiento de datos, ya que es fácil de implementar y tiene una complejidad computacional baja. Sin embargo, también tiene algunas limitaciones, como la sensibilidad a la elección inicial de los centroides, la dificultad para determinar el valor óptimo de k y la incapacidad para manejar clusters con formas irregulares o tamaños muy diferentes.

2.1.1 Inicialización

Tras disponer de los puntos de los elementos, se calculan los centroides de salida iniciales mediante una función de aleatoriedad.

2.1.2 Asignación

Cálculo de las distancias Euclídeas entre cada objeto y los centroides. Determinación del centroide más próximo al objeto.

2.1.3 Actualización

Se actualizarán los valores de los centroides para acercarse al conjunto de objetos que representará. Se repetirá los pasos 2 y 3 hasta la estabilización de los centroides, no obstante, en nuestro caso hemos establecido una única iteración.

2.2 K-means++

El algoritmo K-means++ es una variante del algoritmo K-means que busca mejorar la inicialización de los centroides iniciales para obtener mejores resultados en la agrupación de datos. En el algoritmo K-means clásico, los centroides iniciales se seleccionan aleatoriamente, lo que puede llevar a una mala agrupación de datos si los centroides iniciales no están bien posicionados.

El algoritmo K-means++ propone una forma de seleccionar los centroides iniciales de manera más inteligente, eligiendo los centroides iniciales de manera que estén "alejados" entre sí. Para ello, el algoritmo sigue los siguientes pasos:

1. Seleccionar el primer centroide aleatoriamente entre los puntos de datos.
2. Para cada punto de datos restante, calcular la distancia al centroide más cercano que se haya seleccionado hasta el momento.
3. Seleccionar el siguiente centroide aleatoriamente, con una probabilidad ponderada por la distancia al centroide más cercano que se haya seleccionado hasta el momento. En otras palabras, los puntos de datos que están más lejos de los centroides que ya se han seleccionado tienen una mayor probabilidad de ser seleccionados como el siguiente centroide.
4. Repetir los pasos 2 y 3 hasta que se hayan seleccionado k centroides.

La probabilidad ponderada se puede calcular utilizando la siguiente ecuación:

$$D(x)^2 = \sum_{x' \in C} \min_{\mu \in C} \|x' - \mu\|^2$$

Donde $D(x)$ es la distancia al centroide más cercano que se haya seleccionado hasta el momento, C es el conjunto de centroides que ya se han seleccionado y μ es un centroide en el conjunto C.

El algoritmo K-means++ ha demostrado ser más efectivo que el algoritmo K-means clásico en la agrupación de datos.

3 Diseño E Implementación

3.1 La clase KMeans_DocsRecovery

```
class KMeans_DocsRecovery:
def __init__(self,dict_documentos, k=3):
    self.dict_documentos=dict_documentos # Diccionario de documentos
    self.X,self.unique_words=self.vectorize_documents(dict_documentos) #
        Representación de documentos como vectores binarios
    self.k=k # Número de clústeres
def addDoc(self, docs: dict): # Agregar un documento al diccionario de
   documentos
    self.dict_documentos = self.dict_documentos | docs # Unir los
       diccionarios de documentos
```

3.1.1 vectorize_documents

```
def vectorize_documents(self,dict_documentos):
        # Representación de documentos como vectores binarios
        vectorizador = CountVectorizer(binary=True, stop_words='english')
        self.X= vectorizador.fit_transform(list(dict_documentos.values())) #
            Representación de documentos como vectores binarios
        self.X=self.X.toarray() # Convertir a array
        self.dict_Vectores = {}; i=0 # Diccionario de vectores
        for vector in self.X: # Recorrer los vectores
            self.dict_Vectores[list(dict_documentos.keys())[i]] = vector.
               tolist() # Agregar el vector al diccionario de vectores
            i+=1
        self.unique_words = vectorizador.get_feature_names_out() # Palabras
           únicas
        return self.X, self.unique_words # Devolver la representación de
           documentos como vectores binarios y las palabras únicas
```

3.1.2 k_medias_mejorado

```
    def k_medias_mejorado(self, max_iter=100, tol=1e-4, seed=0):
        # Inicialización robusta de los centroides
        centroides = []
        np.random.seed(seed) #Esto hace que se fije la semilla para que los
           resultados sean reproducibles.
        centroides.append(self.X[np.random.choice(range(self.X.shape[0]), 1)
           , :][0]) # Esto hace que se elija un vector aleatorio de X y se a
           ñada a centroides
        #Tecnica de inicializacion k-means++
        for i in range(1, self.k):
            distancias = np.array([min([np.linalg.norm(x-c)**2 for c in
               centroides]) for x in self.X]) #Esto hace que se calcule la
               distancia de cada vector a cada centroide, distancias es un
               array de distancias, cada distancia corresponde a un vector.
            probs = distancias / distancias.sum() #Esto hace que se calcule
               la probabilidad de que cada vector sea un centroide, probs es
                un array de probabilidades, cada probabilidad corresponde a
               un vector.
            cumprobs = probs.cumsum() #Esto hace que se calcule la
               probabilidad acumulada de que cada vector sea un centroide,
               cumprobs es un array de probabilidades acumuladas, cada
               probabilidad acumulada corresponde a un vector.
            r = np.random.rand() #Esto hace que se elija un número aleatorio
                entre 0 y 1, r es un número aleatorio entre 0 y 1.
            for j, p in enumerate(cumprobs): #Este bucle hace que se elija
               un vector aleatorio de X y se añada a centroides basandonos
               en probs y r
                if r < p: #Esto hace que se elija un vector aleatorio de X y
                    se añada a centroides basandonos en probs y r.
                    centroides.append(self.X[j]) #Esto hace que se elija un
                       vector aleatorio de X y se añada a centroides
                       basandonos en probs y r.
                    break #Esto hace que se elija un vector aleatorio de X y
                        se añada a centroides basandonos en probs y r.
```

```
        for j in range(max_iter): #Bucle principal del algoritmo
            # Asignación de cada vector al centroide más cercano
            distancias = np.linalg.norm(self.X[:, np.newaxis, :] -
                centroides, axis=-1) #np.linalg.norm es la norma euclidea,
                esto hace que se calcule la distancia de cada vector a cada
                centroide, distancias es un array de distancias, cada
                distancia corresponde a un cluster.
            etiquetas = np.argmin(distancias, axis=1) #np.argmin devuelve el
                 indice del valor minimo de un array, Etiqutas es un array de
                 indices, cada indice corresponde a un cluster.

            # Actualización de los centroides
            nuevos_centroides = np.zeros_like(centroides) #np.zeros_like
                devuelve un array de ceros con la misma forma y tipo que el
                array de entrada, esto hace que se cree un array de ceros con
                 la misma forma y tipo que centroides.
            for i in range(self.k): #Este bucle hace que se calcule el
                centroide de cada cluster basandonos en X[etiquetas == i]
                 nuevos_centroides[i] = np.mean(self.X[etiquetas == i], axis
                    =0) #np.mean es la media aritmética, esto hace que se
                    calcule el centroide de cada cluster basandonos en X[
                    etiquetas == i]

            # Verificación de convergencia temprana
            if np.allclose(centroides, nuevos_centroides, rtol=0, atol=tol):
                 #np.allclose devuelve True si dos arrays son iguales dentro
                de una tolerancia, rtol es la tolerancia relativa y atol es
                la tolerancia absoluta, esto hace que se compruebe si los
                centroides y los nuevos centroides son iguales dentro de una
                tolerancia.
                 print(f"Convergencia temprana en la iteración {j}")
                 break
            centroides = nuevos_centroides #Esto hace que se actualicen los
                centroides.

        # Devolución de las etiquetas finales y los centroides finales
        return etiquetas, centroides
```

3.1.3 k_mode

```
    def k_mode(self, max_iter=100, tol=1e-4,seed=0): #este algoritmo es el
        mismo que el anterior, solo cambia np.linalg.norm por scipy.stats.
        mode, que calcula la moda de un array.
         # Inicialización robusta de los centroides
         centroides = []
         np.random.seed(seed) #Esto hace que se fije la semilla para que los
            resultados sean reproducibles.
         centroides.append(self.X[np.random.choice(range(self.X.shape[0]), 1)
            , :][0])
         #Tecnica de inicializacion k-means++
         for i in range(1, self.k):
             distancias = np.array([min([np.linalg.norm(x-c)**2 for c in
                centroides]) for x in self.X])
             probs = distancias / distancias.sum()
             cumprobs = probs.cumsum()
             r = np.random.rand()
```

```
            for j, p in enumerate(cumprobs):
                if r < p:
                    centroides.append(self.X[j])
                    break

        # Bucle principal del algoritmo
        for j in range(max_iter):
            # Asignación de cada vector al centroide más cercano
            distancias = np.linalg.norm(self.X[:, np.newaxis, :] -
                centroides, axis=-1)
            etiquetas = np.argmin(distancias, axis=1)

            # Actualización de los centroides
            nuevos_centroides = np.zeros_like(centroides)
            for i in range(self.k):
                nuevos_centroides[i] = mode(self.X[etiquetas == i], axis=0,
                    keepdims=True).mode[0]

            # Verificación de convergencia temprana
            if np.allclose(centroides, nuevos_centroides, rtol=0, atol=tol):
                print(f"Convergencia temprana en la iteración {j}")
                break
            centroides = nuevos_centroides

        # Devolución de las etiquetas finales y los centroides finales
        return etiquetas, centroides
```

3.1.4 k_mean_fit

```
    def k_mean_fit(self, Algoritmo='k_medias_mejorado', max_iter=100, tol=1e
        -4, seed=0): #Este metodo hace que se ejecute el algoritmo elegido.


        if Algoritmo == 'k_medias_mejorado': #Este if hace que se ejecute el
            algoritmo elegido.
            etiquetas, centroides = self.k_medias_mejorado( max_iter, tol,
                seed)
        elif Algoritmo == 'k_medias':
            etiquetas, centroides = self.k_medias( max_iter)
        elif Algoritmo == 'k_mode':
            etiquetas, centroides = self.k_mode( max_iter, tol, seed)

        # Impresión de los clústeres
        dict_cluster={}
        for i in range(self.k): #Este bucle hace que se cree un diccionario
            con los clusters y sus vectores.
            indices = etiquetas == i #Este bucle hace que se cree un
                diccionario con los clusters y sus vectores.
            vectores_cluster = self.X[indices] #Esto hace que se cree un
                diccionario con los clusters y sus vectores.
            dict_cluster[i] = vectores_cluster.tolist() #Esto hace que se
                cree un diccionario con los clusters y sus vectores.

        for name,clusters in list(dict_cluster.items()): #Este bucle hace
            que se imprima el cluster y sus vectores.
            print(f"\nClúster {name}: valores: {clusters}")
```

```
            for cluster in clusters:
                for key,value in list(self.dict_Vectores.items()):
                    if value == cluster:
                        print(f"Documento: {self.dict_documentos[key]}")
```

4 Resultados

4.1 Kmeans++

```
from n_sklearn_kmeans import KMeans_DocsRecovery, dict_documentosd)
```

4.2 La Clase KMeans_DocsRecovery()

Se inicializa con el diccionario de documentos y con el numero de clusters que se quieren.

```
k_means = KMeans_DocsRecovery(dict_documentos, k=3)
```

4.3 K_medias y la Semilla

La semilla en np, permite hacer numeros aleatorios controlados y reproducibles

```
k_means.k_mean_fit(Algoritmo='k_medias')

Clúster 0: valores: [[0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0], [0, 0,
    0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0,
     0, 0, 1], [0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0]]
Documento: Este es el primer documento
Documento: Este es el segundo documento
Documento: Este es el tercer documento
Documento: Este es el quinto documento

Clúster 1: valores: [[0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]]
Documento: Este es el cuarto documento

Clúster 2: valores: [[0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 1,
    1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0], [1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1,
     0]]
Documento: El sexto documento es diferente
Documento: El séptimo documento es diferente
Documento: El octavo documento es diferente al sexto y séptimo
```

4.4 k_mean_fit

Este tiene como parametros :Algoritmo='k_medias_mejorado', max_iter=100, tol=1e-4, seed=0 es decir, que siempre que se ejecute sin especificar que algoritmo se quiere, entonces por defecto esta el algoritmo k_medias_mejorado

```
  k_means.k_mean_fit()

  Convergencia temprana en la iteración 1

Clúster 0: valores: [[0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0], [0, 0, 0,
    1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
     1], [0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 1, 1, 1, 1, 0,
     0, 1, 0, 0, 0, 0]]
Documento: Este es el primer documento
Documento: Este es el segundo documento
Documento: Este es el tercer documento
Documento: Este es el cuarto documento
Documento: Este es el quinto documento

Clúster 1: valores: [[0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 1,
    1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0]]
Documento: El sexto documento es diferente
Documento: El séptimo documento es diferente

Clúster 2: valores: [[1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0]]
Documento: El octavo documento es diferente al sexto y séptimo
```

4.5 Utilizar la moda

Se inicializa con el diccionario de documentos y con el numero de clusters que se quieren.

```
   k_means.k_mean_fit(Algoritmo='k_mode')

   Convergencia temprana en la iteración 1

Clúster 0: valores: [[0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0], [0, 0, 0,
    1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
     1], [0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 1, 1, 1, 1, 0,
     0, 1, 0, 0, 0, 0]]
Documento: Este es el primer documento
Documento: Este es el segundo documento
Documento: Este es el tercer documento
Documento: Este es el cuarto documento
Documento: Este es el quinto documento

Clúster 1: valores: [[0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 1,
    1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0]]
Documento: El sexto documento es diferente
Documento: El séptimo documento es diferente

Clúster 2: valores: [[1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0]]
Documento: El octavo documento es diferente al sexto y séptimo


k_means.query_k_mean("sexto documento",Algoritmo="k_mode")

Convergencia temprana en la iteración 2

Clúster 0: valores: [[0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0], [0, 0, 0,
    1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
     1], [0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 1, 1, 1, 1, 0,
     0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]]
```

```
Documento: Este es el primer documento
Documento: Este es el segundo documento
Documento: Este es el tercer documento
Documento: Este es el cuarto documento
Documento: Este es el quinto documento
Documento: sexto documento

Clúster 1: valores: [[1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0]]
Documento: El octavo documento es diferente al sexto y séptimo

Clúster 2: valores: [[0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 1,
    1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0]]
Documento: El sexto documento es diferente
Documento: El séptimo documento es diferente
```

5 Conclusión

En conclusión, K-means es un algoritmo de aprendizaje automático ampliamente utilizado para agrupar datos en clústeres. Utiliza la distancia euclidiana para medir la similitud entre los puntos de datos y busca minimizar la suma de las distancias cuadradas entre los puntos de datos y sus centros de clúster asignados.

K-means es fácil de implementar y escalable a grandes conjuntos de datos. Sin embargo, la elección del número de clústeres (valor K) puede ser un desafío, y los resultados pueden ser sensibles a la inicialización aleatoria de los centros de clúster. Además, K-means no es adecuado para datos con formas irregulares o para datos que tienen diferentes tamaños y densidades de clúster.

A pesar de estas limitaciones, K-means sigue siendo un algoritmo popular para el agrupamiento de datos y ha sido utilizado en una variedad de campos, como la biología, la ingeniería, la astronomía y el análisis de mercado. Es importante entender sus fortalezas y limitaciones al utilizarlo en aplicaciones prácticas.