# Implementación de Algoritmo K-Means

Gómez,Jonathan[1], Pérez Oliva,J. Simón[1], and Romero Caballero,Facundo N.[1]

Universidad Gastón Dachary, Posadas Misiones 3300, Argentina

**Abstract.** A lo largo de este trabajo se plasma el funcionamiento del Algoritmo K-Means, abordando la problemática de clasificar distintos conjuntos de datos de R2 dimensiones en K grupos con cada método. Una de las posibles soluciones a esta problemática es el uso del algoritmo de clustering K-Means mediante los métodos de inicialización aleatorio y heurístico (K-Means ++). Se propone la solución mediante la implementación en el lenguaje de programación Python, dejando explícito a su vez las diferentes librerías utilizadas para poder unir interfaz gráfica con el modelo, y el análisis de los resultados obtenidos en base a las 72 pruebas realizadas.

**Keywords:** K-Means · cluster · centroide · K-Means++ · semilla.

## 1 Introducción

Los problemas de clasificación requieren la aplicación de técnicas que permitan agrupar o clasificar distintos conjuntos de datos en un conjunto de grupos o clases. A lo largo de este trabajo, se plantea el marco teórico referente a la problemática a solucionar, a modo de contextualización,. Luego, se presenta la solución al problema recién planteado mediante la implementación de una aplicación de escritorio desarrollada en el lenguaje de programación Python, gestionando los datasets utilizados mediante el uso de listas, arreglos y dataframes. Además, se consta de un apartado de Simulaciones y Resultados Obtenidos, en el cual se compararán los resultados arrojados, para posteriormente finalizar el trabajo con las conclusiones arribadas en base a las pruebas obtenidas.

## 2 Marco Teórico

### 2.1 Aprendizaje No Supervisado

El aprendizaje no supervisado es un tipo de aprendizaje en el cual el modelo utiliza datos que no poseen una partición dada por un atributo de clase, como sí lo poseen los modelos de aprendizaje supervisado. Una técnica de aprendizaje supervisado es el Clustering, descrito a continuación.

### 2.2 Clustering

Un clúster o clase se define como una colección de datos que poseen semejanzas entre sí o diferencias respecto a datos de otros grupos, todos ellos comprendidos dentro del mismo dataset. Teniendo esto en mente, se define al Clustering como el proceso de organizar instancias de datos de un dataset en grupos o clusters cuyos miembros sean similares, tal como indica [1]. Estas instancias de datos también se conocen como data points u objetos.

### 2.3 Algoritmo K-Means

El algoritmo K-Medias (K-Means), también llamado Algoritmo de Lloyd es un algoritmo de clustering muy utilizado en la minería de datos por su sencillez para realizar segmentación de datos en k clusters (representados por k centroides, los cuales se definen como la media de todos los data points del cluster según la ecuación 1), a fin de clasificarlos y analizarlos posteriormente. A continuación se detallan los pasos para la realización del algoritmo.

$$mj = \frac{1}{|Cj|} . \sum_{x \in Cj} xi \tag{1}$$

Inicialización: Dado el dataset y el número de k clases a utilizar para la clasificación, brindados por el usuario, se inicializan los k centroides (uno por cluster) utilizando los valores de k data points del dataset.

Asignación: Se asigna cada uno de los data points al centroide más cercano, tal como indica [2] siguiendo la ecuación (2).

$$k(n) = argmin[k(d(xn, mj))] \tag{2}$$

siendo $k(n)$ el cluster al que pertenece el punto $xn$ y siendo $d(mj, xn)$ la distancia entre el punto $xn$ y el centroide $mj$, tal como sugiere la ecuación (3) a continuación para distancias euclidianas.

$$d(x_n, m_j) = ||x_n - m_j|| = \sqrt{(x_n 1 - m_j 1)^2 + (x_n 2 - m_j 2)^2 + ... + (x_n r - m_j 1 r)^2} \tag{3}$$

Actualización: Una vez asignados todos los data points, se re computan los centroides utilizando los data points de cada clúster. Este proceso se repite hasta que se cumple alguno de los tres siguientes criterios de parada:

1. No hay más (o mínima cantidad) de reasignaciones de puntos a clusters diferentes.
2. No hay más (o mínima cantidad) de cambios de centroides.
3. Mínimo decrecimiento en la suma del error cuadrático SSE.

$$SSE = \sum_{j=1}^{k} \sum_{c \in C_j} (m_j, x_n)^2 \tag{4}$$

### 2.4 Algoritmo K-Means ++

Dado el problema de complejidad computacional de tipo NP-duro para inicialización de semillas, el cual es una característica del algoritmo k-means estándar, y que es la causa de agrupamientos pobres o malos por parte del mismo, surge la variante k-means ++. Esta última consiste en la realización de los siguientes pasos, siguiendo [3]:

1. Elegir un punto aleatorio del dataset de entrada como centroide, utilizando una variable aleatoria uniforme.
2. Elegir un nuevo centroide, utilizando otro punto del dataset de entrada, pero siguiendo la probabilidad:
$$\frac{D(x^2)}{\sum_{x \in X} D(x^2)} \tag{5}$$
donde $D(x)^2$ es la distancia más corta entre un data point y el centroide más cercano ya elegido
3. Repetir el paso 2 hasta haber elegido los k centroides de clusters.
4. Proseguir con el algoritmo k-means estándar de optimización.

Con todo esto, se pasa de poseer una complejidad computacional NP-duro (k-means estándar) a poseer una complejidad computacional de $O(logk)-competitivo$, lo cual provoca una convergencia más acelerada, y un tiempo de cálculo muchas veces menor que el algoritmo estándar.

### 2.5 Índice Calinski-Harabasz

También conocido como Criterio de Ratio de Varianza, el índice Calinski-Harabasz es una métrica que representa el grado de dispersión entre clusters y a su vez la dispersión interna de los puntos dentro de cada clúster. Este índice se relaciona a un modelo de tal manera que, a mayor valor del índice, más densos serán los clusters y más dispersos estarán los clusters entre sí, denotando una mejor y más pronunciada separación entre los mismos. La formulación matemática del índice, siguiendo a [4], se denota como:

$$s = \frac{tr(B_k)}{tr(W_k)} X \frac{n_E - k}{k - 1} \tag{6}$$

donde $s$ es el índice Calinski-Harabasz, $E$ es un conjunto de datos de tamaño $n_E$, separado en k clusters, $tr(B_k)$ la traza de la matriz de dispersión entre clusters y $tr(W_k)$ la traza de la matriz de dispersión entre elementos de cada clúster, siendo estos últimos términos iguales a

$$W_k = \sum_{q=1}^{k} \sum_{c \in C_q} (x - c_q)(x - c_q)^T \tag{7}$$

$$B_k = \sum_{q=1}^{k} n_q (c_q - c_E)(c_q - c_E)^T \tag{8}$$

siendo $C_q$ un set de datos del clúster $q$, $c_q$ el centroide del clúster $q$, $c_E$ el centro de $E$ y $n_q$ el número de puntos en $q$.

## 3 Descripción de la Solución Implementada

Para la implementación del algoritmo K-Means se utilizaron las siguientes tecnologías: En la presente implementación, el lenguaje de programación utilizado es Python, debido a su sintaxis simple y fácil de utilizar para el desarrollo del núcleo de la aplicación, además de para las representaciones gráficos de dispersión en ventanas a través de librerías.

En conjunto con Python, se utilizó el IDE Visual Studio Code para la programación en Python, debido a su versatilidad y facilidad de uso. Dentro de la implementación propia en Python, los datasets, centroides y distancias se manejan mediante listas de Python. Para el caso del dataset se pasa a manejar mediante una lista llamada "listaPuntos", para el caso de los centroides se pasa a manejar mediante una lista denominada "listaCentroides", y para el caso de las distancias de los puntos a los centroides, se pasa a manejar mediante una lista de nombre "listaDistancias".

Al comienzo del algoritmo, luego de que el usuario provea los k clusters y el método de inicialización (aleatorio o heurístico), se procede a obtener mediante la función random de Python los k puntos iniciales del dataset para el caso de inicialización aleatoria, y utilizando el método de K-Means++ para la inicialización heurística. En ambos casos, se almacenan todos los centroides la lista "listaCentroides".

Luego de esto, lo que se describe a continuación se realiza hasta que se cumpla con los dos primeros criterios de parada. Para cada punto de la lista del dataset, se calculan las distancias hacia los k centroides de la lista de centroides y se almacenan en la lista "listaDistancias" con el identificador hacía que centroide pertenece dicha distancia, después de esto se elige a la menor distancia en dicha lista y se procede a actualizar el valor del punto, el cual indica a que centroide pertenece. Además se van modificando los valores de "cantidadX", "cantidadY" y "cantidadCentroide" de los centroides para su posterior actualización. Por otra parte también se verifica el cambio del centroide de un punto para el primer criterio de parada.

Una vez terminado el punto anterior, se procede a reasignar la posición de de los centroides utilizando dichos valores modificados y aplicándolos en base a la ecuación (1).

Por último se procede a verificar, para el segundo criterio de parada, si hubo modificación en la posición de los centroides, utilizando la lista "listaCentroidesAux", la cual contiene la lista de los centroides con sus posiciones sin actualizar, y la lista "listaCentroidesActu", la cual contiene la lista de los centroides posterior a la reasignación de posiciones.

Este proceso de actualización de centroides se repite hasta cumplir con los criterios de parada número 1 y 2 expuestos en la sección anterior.

Como complemento al núcleo de la aplicación desarrollada, se utilizaron las siguientes librerías:

- PyQt5: librería utilizada para la representación gráfica en ventanas del trabajo [5].
- Matplotlib: librería utilizada para la representación gráfica de listas en 2 dimensiones. Es decir, se utiliza para la representación de gráficos de dispersión [6].
- Random: librería propia de Python para la generación de números pseudo-aleatorios [7].

## 4 Simulaciones y Resultados Obtenidos

De acuerdo con los lineamientos del presente trabajo (los cuales son limitar la cantidad k de clusters de 2 a 5), se corrobora el funcionamiento del algoritmo realizando 72 simulaciones utilizando los 3 datasets, 36 pruebas con inicialización aleatoria y 36 pruebas con inicialización heurística, 12 pruebas por dataset, de las cuales serán 3 pruebas por cada k ($k = 2$; $k = 3$; $k = 4$; $k = 5$). De esta forma el algoritmo debe realizar la clasificación de cada dataset tanto con inicialización aleatoria como heurística considerando a su vez distintos números de clusters a clasificar.

A continuación se presenta, a modo de ejemplo, los resultados de una prueba de resolución de un dataset a fin de denotar gráficamente la interfaz de la aplicación de escritorio desarrollada.

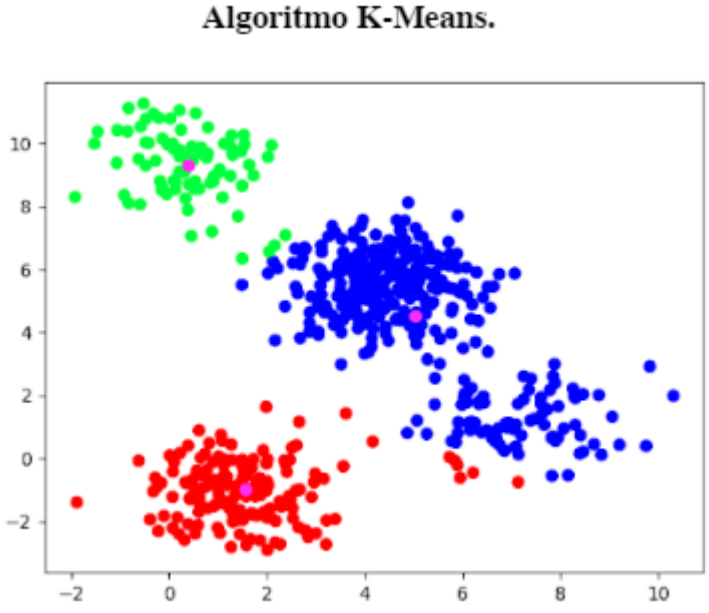

Fig 1. Dataset Nº 1 luego de la ejecución (clasificación) del algoritmo K-Means con $k = 3$.

A continuación se toma una muestra de 72 pruebas (el cual es un número que agrega bastante variedad a la población, aumentando la probabilidad de ocurrencia de casos especiales), utilizando la aplicación desarrollada y tomando los siguientes ítems como datos relevantes de dichas pruebas:

- Cantidad de Iteraciones: cantidad de iteraciones que debe realizar el algoritmo para clasificar exitosamente el dataset en k clusters.
- Cantidad de Cálculos de Distancia: cantidad de veces que el algoritmo realizó cálculos de distancias entre puntos y clusters.
- Cantidad de Clusters Elegidos: cantidad k de clusters elegida a priori para clasificar el dataset.
- Cantidad de Data Points por Cluster: cantidad de elementos que posee cada cluster luego de la clasificación.
- Índice Calinski-Harabasz: puntuación obtenida al ejecutar el Índice Calinski-Harabasz sobre el agrupamiento realizado.

En base a los datos obtenidos de las 72 pruebas realizadas, se obtiene la siguiente tabla con las siguientes medidas de rendimiento y eficiencia de uso de recursos:

- Promedio de Iteraciones (PI): basado en el dato "Cantidad de Iteraciones". Representa la cantidad promedio de iteraciones realizadas por el algoritmo para realizar la clasificación de las 72 pruebas separadas por cantidad de clusters y por tipo de inicialización. Esta medida da un indicio de rendimiento del programa al momento de realizar las corridas.
- Promedio de Cantidad de Cálculos de Distancia (PCCD): basado en el dato "Cantidad de Cálculos de Distancia". Representa la cantidad promedio de cálculos de distancias realizadas por el algoritmo para realizar la clasificación de las 72 pruebas separadas por cantidad de clusters y por tipo de inicialización. Esta medida da un indicio de rendimiento del programa al momento de realizar las corridas.
- Promedio de Índice Calinski-Harabasz (PICH): puntuación obtenida del dato "Índice Calinski-Harabasz". Da un indicio de qué tan bien agrupados están los clusters; a mayor puntuación del índice, mejor agrupados están los clusters.

Dados los resultados obtenidos, se observan los siguientes ítems:

1. Comparando los "PI" con ambos tipos de inicialización, se observa que el promedio de iteraciones por k con inicialización aleatoria es menor que el mismo para inicialización heurística para $k < 4$. Sin embargo, la situación se revierte para $k >= 4$ .
2. Comparando los "PCCD" con ambos tipos de inicialización, se observa que el promedio de la cantidad de cálculos de distancias realizados por k varía

| | Dataset N° 1 | | | | Dataset N° 2 | | | | Dataset N° 3 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | k = 2 | k = 3 | k = 4 | k = 5 | k = 2 | k = 3 | k = 4 | k = 5 | k = 2 | k = 3 | k = 4 | k = 5 |
| PI | 8,67 | 9,33 | 7,33 | 10,33 | 5 | 5 | 11,33 | 13 | 7 | 8 | 13,33 | 16,67 |
| PCCD | 10920 | 17640 | 18480 | 32550 | 6000 | 9000 | 27200 | 39000 | 11200 | 19200 | 42666,67 | 66666,67 |
| PICH | 34,84 | 68,18 | 105,07 | 151,39 | 3168,38 | 2171,76 | 2350,4 | 3472,96 | 168,29 | 558,3 | 1964,51 | 7269,76 |

**Table 1.** Resultados estadísticos obtenidos en base a las 36 pruebas con inicialización aleatoria

| | Dataset N° 1 | | | | Dataset N° 2 | | | | Dataset N° 3 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | k = 2 | k = 3 | k = 4 | k = 5 | k = 2 | k = 3 | k = 4 | k = 5 | k = 2 | k = 3 | k = 4 | k = 5 |
| PI | 7 | 10 | 5 | 15,33 | 7,33 | 4,33 | 10,67 | 14 | 5,67 | 18,33 | 14,33 | 6 |
| PCCD | 630 | 1888 | 3772 | 6280 | 600 | 1798 | 3592 | 5980 | 800 | 2398 | 4792 | 7980 |
| PICH | 34,87 | 76,47 | 103,71 | 168,66 | 1292,09 | 2053,96 | 1848,23 | 2537 | 168,1 | 606,68 | 1258,74 | 1619,08 |

**Table 2.** Resultados estadísticos obtenidos en base a las 36 pruebas con inicialización heurística (K-Means ++)

enormemente entre la inicialización aleatoria y la heurística, incrementándose dicho promedio en 1385,23 por ciento para $k = 2$, 753,45 por ciento para $k = 3$, 726,77 por ciento para $k = 4$ y 682,88 por ciento $k = 5$. Cabe denotar, además, que dicho incremento disminuye conforme se aumenta el número k de clusters, por lo que no se descarta que en algún punto se estabilice el incremento dado.

3. Comparando los "PCCD" con ambos tipos de inicialización, se observa que en todos los casos, indistintamente del tipo de inicialización y el dataset de entrada, siempre ocurre un incremento linealmente proporcional entre el número k de clusters y la cantidad de distancias a calcular.
4. Comparando los "PICH" con ambos tipos de inicialización, se denota que los promedio de índices poseen una variación muy pequeña entre tipos de inicialización. Sin embargo, se observa que entre datasets ocurre una diferencia muy marcada.

## 5 Conclusiones

En este informe se presenta el desarrollo del algoritmo K-Means, implementado como resolución a la problemática de clasificar un conjunto de datos de dos dimensiones que se presenta como entrada, representado gráficamente como un conjunto de puntos en $R^2$ en un espacio euclidiano. A continuación se presentan las conclusiones arribadas del análisis de los resultados obtenidos en la sección anterior, siguiendo el mismo orden de los ítems previamente expuestos.

1. En el ítem 1 de la sección previa se expuso que el promedio de iteraciones por k con inicialización aleatoria es menor que para inicialización heurística para $k < 4$, y la situación se revierte para $k >= 4$. Esto se debe a que la

aleatoriedad de la elección de semillas con inicialización aleatoria es más eficiente para poca cantidad de clusters a clasificar. Sin embargo, se revierte la situación debido a que esta característica se pierde para cantidad de clusters a clasificar más elevada.
2. En el ítem 3 de la sección previa se expuso que el promedio de la cantidad de cálculos de distancias realizados por k varía enormemente entre la inicialización aleatoria y la heurística. Esto se debe al orden de complejidad de cada tipo de inicialización, siendo el heurístico (K-Means ++) mucho más eficiente, con una complejidad de $O(logk)$.
3. En el ítem 4 de la sección previa se expuso que siempre ocurre un incremento linealmente proporcional entre el número k de clusters y la cantidad de distancias a calcular. Esto se debe a que, indistintamente del tipo de inicialización, si se añaden más clusters a clasificar, más distancias tendrá que calcular el algoritmo porque es una característica del propio K-Means.
4. En el ítem 5 de la sección previa se expuso que con ambos tipos de inicialización, se denota que los promedio de índices poseen una variación muy pequeña entre tipos de inicialización, pero entre datasets ocurre una diferencia muy marcada. Esto se debe a que, lo que determina el índice Calinski-Harabasz no es el tipo de inicialización, sino la calidad de agrupamiento de los datos del dataset de entrada.

## References

1. Liu, B., and Mining, W. D. Web Data Mining. Exploring Hyperlinks, Contents, and Usage Data. Ser. Data-Centric Systems and Applications. Chapter 4 - Unsupervised Learning. Springer Berlin Heidelberg.(2011)
2. MacKay, David, An Example Inference Task: Clustering. Information Theory, Inference and Learning Algorithms. Chapter 20, pp. 284-292. (2003). ISBN 0-521-64298-1. MR 2012999.
3. Arthur, D. and Vassilvitskii,"k-means++: the advantages of careful seeding" (2007)
4. Caliński, T., and Harabasz, J.“A Dendrite Method for Cluster Analysis”. Communications in Statistics-theory and Methods 3, pp. 1–27. (1974)
5. Proyecto PyQt. https://pypi.org/project/PyQt5/.
6. Proyecto Matplotlib.https://matplotlib.org/.
7. Documentación de Random - Proyecto Python.https://docs.python.org/3/library/random.html.