

Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

Grado en Matemáticas

Curso 2022-2023

Trabajo Fin de Grado

**ESTUDIO COMPARATIVO DEL ALGORITMO
K-MEANS PARA DISTINTAS DISTANCIAS EN
ESPACIOS MÉTRICOS**

Autor: Marcos Crespo Díaz
Tutora: María Jesús Algar Díaz

Agradecimientos

A mi tutora María Jesús Algar Díaz, por su organización y ética de trabajo ejemplar, que me han facilitado inmensamente la elaboración de este trabajo.

A mis padres y mi hermana, por su ayuda y consejo incondicional.

Y a José Blasco, por su paciencia y generosidad no acotadas superiormente y por sus ánimos que me han acompañado durante este viaje.

“Es probable que ni uno ni otro sepamos nada que tenga valor, pero este hombre cree saber algo y no lo sabe, en cambio yo, así como, en efecto, no sé, tampoco creo saber.”

Platón, *Apología de Sócrates*

Resumen

El algoritmo de K-Means es un conocido algoritmo de análisis clúster que se utiliza en diversas áreas de la ciencia de datos desde mediados del S.XX. Sus fundamentos matemáticos son fácilmente entendibles y muy intuitivos existiendo diversas modificaciones que se le pueden hacer al algoritmo para adaptarlo a diferentes situaciones. Factores como la función de distancia para medir la similitud o el espacio métrico donde habitan los puntos a analizar tienen una gran importancia sobre el resultado de la clusterización. El algoritmo se puede implementar para realizar un estudio comparativo de las distintas variaciones que exponemos, y observar bajo qué condiciones se comporta mejor esta técnica.

Palabras clave:

- K-Means
- Ciencia de datos
- Data Science
- Distancias
- Espacios métricos
- Euclídea
- Manhattan
- Minkowski
- Chebysev
- Python
- Estudio comparativo

Índice de contenidos

Índice de tablas	IX
Índice de figuras	XI
Índice de códigos	XIII
1. Introducción	1
1.1. Objetivos del trabajo	2
1.2. Estructura del documento	3
2. El algoritmo de K-Means	4
2.1. La ciencia de datos	4
2.1.1. Aprendizaje automático. Clasificación.	5
2.2. Contexto histórico del K-Means	7
2.3. El algoritmo	8
3. Espacios métricos	15
3.1. Distancia euclídea	16
3.2. Distancia de Manhattan	18
3.3. Distancia de Chebysev	21
3.4. Distancia de Minkowski	23
4. Estudio comparativo del K-Means en distintos espacios métricos	26
4.1. Implementación del algoritmo	27
4.1.1. K-Means. El algoritmo en Python	27
4.2. K-Means sobre datasets sintéticos	30
4.2.1. Primer ejemplo: Nubes simples	30
4.2.2. Segundo ejemplo: Nubes simples con outliers	40
4.3. K-Means sobre la base de datos MNIST	45
4.3.1. El código	46
4.3.2. Resultados	48
5. Conclusiones	53

6. Trabajos futuros	56
Bibliografía	56
Anexos	60
A. Primer anexo	62
A.1. Códigos	62
A.1.1. Distancias	62
A.1.2. Algoritmos de K-Means para $n=2$	63
A.1.3. Análisis para un dataset simple	66
A.1.4. Análisis para un dataset con outliers	73
A.1.5. Algoritmos de K-Means para dimensión n	80
A.1.6. Análisis para el MNIST	83

Índice de tablas

4.1. Matriz de confusión	33
4.2. Distribución de cada nube para la distancia euclídea en un dataset simple	34
4.3. Matriz de confusión para un dataset simple con la distancia euclídea	35
4.4. Distribución de cada nube para la distancia de Manhattan en un dataset simple	36
4.5. Matriz de confusión para un dataset simple con la distancia de Manhattan	37
4.6. Distribución de cada nube para la distancia de Chebysev en un dataset simple	38
4.7. Matriz de confusión para un dataset simple con la distancia de Chebysev	38
4.8. Distribución de cada nube para la distancia euclídea en un dataset con outliers	42
4.9. Matriz de confusión para un dataset con outliers con la distancia euclídea	42
4.10. Distribución de cada nube para la distancia de Manhattan en un dataset con outliers	43
4.11. Matriz de confusión para un dataset con outliers con la distancia de Manhattan	43
4.12. Distribución de cada nube para la distancia de Chebysev en un dataset con outliers	44
4.13. Matriz de confusión para un dataset con outliers con la distancia de Chebysev	44
4.14. Recuento de etiquetas de MNIST	48
4.15. MNIST para la distancia euclídea	49
4.16. MNIST para la distancia de Manhattan	50
4.17. MNIST para la distancia de Chebysev	51

Índice de figuras

2.1. Dendograma, diagrama de Venn anidado y esquema particional	7
2.2. Diferentes formas de agrupar datos sobre el plano	9
3.1. Esquema Distancia de Manhattan	19
3.2. Esquema Distancia de Chebysev	21
3.3. Circunferencias goniométricas para diferentes órdenes de p	24
4.1. Datos sintéticos con 5 nubes diferenciadas	32
4.2. K-Means con distancia euclídea para el dataset simple	35
4.3. K-Means con distancia de Manhattan para el dataset simple	37
4.4. K-Means con distancia de Chebysev para el dataset simple	38
4.5. Dataset con outliers	41
4.6. K-Means con distancia euclídea para el dataset con outliers	42
4.7. K-Means con distancia de Manhattan para el dataset con outliers	43
4.8. K-Means con distancia de Chebysev para el dataset con outliers	44
4.9. Base de datos MNIST.	46

Índice de códigos

4.1. Distancias en 2 dimensiones	27
4.2. K-Means con distancia euclídea con $n = 2$	28
4.3. Generación de dataset de nubes simples	30
4.4. Función comprobar	32
4.5. K-Means con distancia euclídea para un dataset simple	34
4.6. K-Means con distancia de Manhattan para un dataset simple	35
4.7. K-Means con distancia de Chebysev para un dataset simple	37
4.8. Generación de dataset con outliers	40
4.9. Distancias n-dimensionales	46
4.10. Función comprobar n-dimensional	47
A.1. Funciones de distancia	62
A.2. K-Means para la distancia euclídea y $n=2$	63
A.3. K-Means para la distancia de Manhattan y $n=2$	64
A.4. K-Means para la distancia de Chebysev y $n=2$	65
A.5. Análisis para un dataset simple	66
A.6. Análisis para un dataset con outliers	73
A.7. K-Means para la distancia euclídea y dimensión n	80
A.8. K-Means para la distancia de Manhattan y dimensión n	81
A.9. K-Means para la distancia de Chebysev y dimensión n	82
A.10. Análisis para MNIST	83

1

Introducción

La ciencia de datos es una rama académica eminentemente multidisciplinar. Toma elementos de la estadística, la computación, la algoritmia y las matemáticas para solventar los problemas que generan los grandes volúmenes de datos que recogemos desde mediados del S.XX. Busca analizar, organizar y extraer conocimiento y conclusiones sobre datos en relación a casi cualquier otra rama del conocimiento donde sus técnicas se puedan aplicar, tanto científicas como sociales y económicas.

Todo esto se traduce en el desarrollo de una gran variedad de técnicas y metodologías aplicables sobre conjuntos de datos de diversas índoles, pero todas ellas con el objetivo de extraer algún tipo de información adicional a la preexistente en los datos. De entre todas estas técnicas existe un algoritmo concebido en la década de 1960 llamado el algoritmo de *K-Means*.

El algoritmo de *K-Means* es un conocido algoritmo de análisis clúster. Su principal objetivo es organizar conjuntos de datos en distintos grupos o, como los llamaremos de aquí en adelante, *clústers*. Dicha organización se hace pretendiendo que los elementos más similares entre si pertenezcan al mismo clúster. Este algoritmo se aplica con éxito en diversas disciplinas de ingeniería y ciencias, tales como biología, psicología, medicina, marketing, visión computacional e inteligencia artificial. A continuación, se pueden investigar las clasificaciones resultantes para evaluar si los datos se agrupan acorde a ideas preconcebidas, o para proponer nuevos experimentos y líneas de investigación; es decir, para obtener nueva información acerca de los datos.

1.1. Objetivos del trabajo

Los objetivos de este trabajo serán dos principalmente. En primer lugar, se pretenderá realizar una explicación amplia y profunda del algoritmo de *K-Means*, englobándolo dentro de las diferentes ramas de la ciencia de datos. Una vez finalizado el documento, el lector tendrá una visión completa sobre esta técnica y de manera intuitiva habrá llegado a las conclusiones que llevaron a su desarrollo. A su vez, se intentará dar una amplia visión de los fundamentos matemáticos sobre los que el algoritmo se apoya, justificando su utilidad y casos de uso para cada uno de ellos.

Por otro lado, también se busca no solo dar un enfoque teórico sobre el algoritmo, si no que se pretenderá a su vez aplicar en diversos ambientes. El algoritmo de *K-Means* es un algoritmo vigente en el día a día de investigadores y empresas actuales. Ellos trabajan en distintos entornos computacionales con datos muy diversos, y ponen en práctica algunos de los aspectos que durante este trabajo discutiremos. Esto, como veremos más adelante, pasa por aplicar el algoritmo de distintas formas y variando algunos de los parámetros que más adelante se discutirán.

La metodología que se seguirá aportará al trabajo dos partes diferenciadas. Una primera parte más teórica y formal explicando las bases matemáticas de la técnica, y otra abordando un estudio comparativo del algoritmo. Hacer un estudio comparativo de un algoritmo significa analizar y evaluar el rendimiento del mismo en relación con otros algoritmos que abordan el mismo problema. El objetivo principal de un estudio comparativo es determinar qué algoritmo es más efectivo en términos de rendimiento y eficiencia.

Un estudio comparativo típicamente involucra la selección de un conjunto de algoritmos que abordan el mismo problema, la definición de un conjunto de criterios de evaluación, la ejecución de los algoritmos en un conjunto de datos de prueba y la comparación de los resultados obtenidos por cada algoritmo. Los criterios de evaluación pueden incluir la precisión, el tiempo de ejecución, la escalabilidad y la facilidad de uso, entre otros.

Es importante destacar que los estudios comparativos de algoritmos pueden ser muy útiles para la comunidad científica y para los profesionales que trabajan en el área, ya que permiten una evaluación objetiva de los algoritmos y proporcionan información valiosa para la selección y diseño de algoritmos en diferentes aplicaciones. Sin embargo, es fundamental llevar a cabo los estudios comparativos de manera cuidadosa, ya que una selección inadecuada de criterios de evaluación o un conjunto de datos de prueba no representativo pueden conducir a conclusiones erróneas o engañosas.

1.2. Estructura del documento

Como ya hemos introducido, el trabajo constará de dos partes. La primera parte, más teórica, estará contenida en los capítulos 2 y 3. La segunda parte, el estudio comparativo, vendrá presentada en el capítulo 4. Finalmente, se completará la visión general de la técnica con una sección de conclusiones 5 y de futuras líneas de trabajo 6.

El capítulo 2 se centrará en dar la intuición, definiciones y nociones más básicas que matemáticamente serán necesarias para comprender el algoritmo. Poco a poco se pretenderá deducir un esquema algorítmico muy básico que encapsule correctamente las ideas fundamentales del *K-Means*.

Después, en el capítulo 3 nos adentraremos en los conceptos de distancia y espacio métrico, fundamentales para la sustentación matemática de la técnica. También, sobre estos conceptos se apoyan la mayoría de cambios y adaptaciones que proponemos para el algoritmo en el estudio comparativo. Comentaremos qué aplicaciones pueden tener estos conceptos abstractos en el análisis clúster.

El capítulo 4 conforma el cuerpo del estudio comparativo para los distintos espacios métricos. Haremos un recorrido extenso sobre el código implementado, explicando cómo refleja cada aspecto teórico explicado, y expondremos las diferencias en los resultados para cada una de las variantes del algoritmo a utilizar.

Finalmente, daremos una visión general en un apartado de Conclusiones 5 y completaremos el estudio con una sección de futuras líneas de trabajo 6 que aporten información sobre cuales serían los siguientes aspectos a estudiar sobre el algoritmo.

Todos los códigos utilizados para la elaboración de este trabajo se podrán consultar en los Anexos A, así como las referencias bibliográficas.

2

El algoritmo de K-Means

2.1. La ciencia de datos

Como ya hemos comentado, la ciencia de datos es aplicable en entornos tan distintos como ramas del conocimiento existen, y es por esto por lo que se han desarrollado técnicas que se adaptan a tipos de conjuntos de datos muy diversos. A menudo, para navegar entre tantas técnicas, se suele generar una clasificación; pero hay que entender que al ser la ciencia de datos una rama tan joven, aún no se ha aceptado una clasificación absoluta de su contenido.

El tratamiento estadístico de unos datos posee diversas etapas, lo que nos sugiere quizás una primera clasificación. Distinguimos la etapa de recogida de los datos, la etapa de almacenamiento de los mismos y la etapa de procesado y obtención de información. Es precisamente esta última etapa, la de procesado y obtención de la información, donde nos centraremos en el presente trabajo. La recogida de los datos será labor del ingeniero de datos, que diseñará un esquema correcto para que la información recogida suponga un dibujo completo de la realidad. Es la labor de los ingenieros de sistemas su almacenamiento en distintos formatos y localizaciones. Por último, es el científico de datos el que entrará en la interpretación de dicha información recogida y elaboración de modelos gracias a estos datos y a este marco de trabajo. Cabe mencionar que a menudo, las barreras entre estas etapas están ampliamente desdibujadas y que los equipos de trabajo no están compartimentados. Aún así, se puede abstraer un flujo de trabajo similar en cualquier tarea que involucre el tratamiento estadístico de datos.

Para procesar los datos y obtener información de los mismos existen varios enfoques. De entre ellos, en los últimos años hay uno que está cobrando gran importancia por su aplicación en ramas como el Big Data o la Inteligencia Artificial, y este es el Aprendizaje Automático. El Aprendizaje Automático (Machine Learning en inglés) es una rama que se enfoca en desarrollar algoritmos y modelos matemáticos que permiten a las computadoras aprender a partir de datos y realizar tareas específicas sin ser explícitamente programadas para hacerlo.

En lugar de codificar explícitamente reglas y algoritmos para resolver un problema, en el aprendizaje automático se utilizan algoritmos y técnicas estadísticas para permitir que las computadoras identifiquen patrones en los datos, realicen predicciones y tomen decisiones.

El Aprendizaje Automático se aplica en una amplia selección de tareas de distintas áreas, como el reconocimiento de patrones, la visión por computadora, el procesamiento del lenguaje natural, la medicina, la economía y la física, entre otras. Los ejemplos que encontramos en su uso diario son innumerables incluyendo la detección de fraude en tarjetas de crédito, el reconocimiento de voz y de imágenes, la recomendación de productos y servicios, la predicción de enfermedades y el análisis de datos financieros.

2.1.1. Aprendizaje automático. Clasificación.

En aprendizaje automático podemos dividir las técnicas en dos grandes grupos: aprendizaje supervisado y aprendizaje no supervisado.

Los algoritmos supervisados basan sus conclusiones en datos históricos etiquetados, es decir, con una clasificación o etiqueta previa. Buscan generar una función que prediga cierto comportamiento para nuevos datos. Estas técnicas permiten realizar predicciones futuras basadas en el comportamiento o en las características que se han analizado sobre los datos históricos. Las etiquetas no son más que las salidas que se han obtenido para los datos históricos. Estos algoritmos, a su vez, se clasifican en:

- Modelos de Regresión (regresión lineal, múltiple, logística, redes neuronales, ANOVA, etc.): se encargan de describir la relación entre una variable dependiente en función del resto de variables con el objetivo principal de predecir el comportamiento de los datos para futuras observaciones. Toman los datos etiquetados como datos de entrenamiento para validar el modelo y se aplican después sobre datos nuevos sin etiquetar.
- Modelos de Clasificación (cuantificador bayesiano, análisis discriminante, AdaBoost, árboles de decisión, random forest, etc): tratan de encontrar los patrones subyacentes en las variables explicativas de los datos históricos con

el objetivo de entrenar el modelo para poder realizar predicciones sobre las variables respuesta de los nuevos datos. Para ello, organizan en grupos y clasifican los datos atendiendo a una o varias variables de interés.

Los algoritmos no supervisados no trabajan con datos históricos, si no que extraen conclusiones de conjuntos de datos a priori sin etiquetar. Estos algoritmos no buscan una función que se ajuste a la salida deseada, si no que lo que les interesa es aumentar el conocimiento sobre los datos de los que se dispone. Principalmente buscan patrones o relaciones entre los datos. En particular, estos algoritmos se pueden clasificar en:

- **Análisis Clúster:** los algoritmos tratan de buscar grupos (clústers) dentro del conjunto de datos no categorizados, es decir, que no tienen etiquetas. La búsqueda de estos grupos se hace por similaridad. Aquellos elementos que son más semejantes entre si, se agrupan en un mismo clúster (análisis de correspondencias simple y múltiple, análisis clúster jerárquico y no jerárquico).
- **Reducción de la Dimensionalidad:** el objetivo es escoger un subconjunto de dimensiones en el que los datos se puedan representar mejor que en el espacio inicial (Análisis de Componentes Principales o PCA, análisis factorial, escalamiento multidimensional).

Análisis Clúster Jerárquico vs. Particional

El análisis clúster jerárquico se podría entender intuitivamente como un diagrama de Venn anidado. Se crea un árbol jerárquico de clústers en el que cada observación se agrupa en un clúster individual y se van fusionando los clústers en función de su similitud. Este proceso puede ser aglomerativo (comenzando con clústers individuales y fusionando clústers similares) o divisivo (comenzando con todos los datos en un clúster y dividiéndolos en clústers más pequeños). Tratan de encontrar el número óptimo de clústers entre todas estas divisiones. La representación natural suele ser un dendograma como el que se observa en figura 2.1.

Por otro lado, el Análisis clúster no jerárquico o particional sabe a priori cuántos clúster desea crear y forma grupos disjuntos de elementos desde un principio. No se basa en elegir el número de grupos óptimo para hacer una división natural de los datos, si no que dado un numero de grupos deseados, encuentra la división óptima de los datos atendiendo a una determinada condición de similitud entre ellos. Es dentro de este grupo donde se encuentra el algoritmo de *K-Means*.

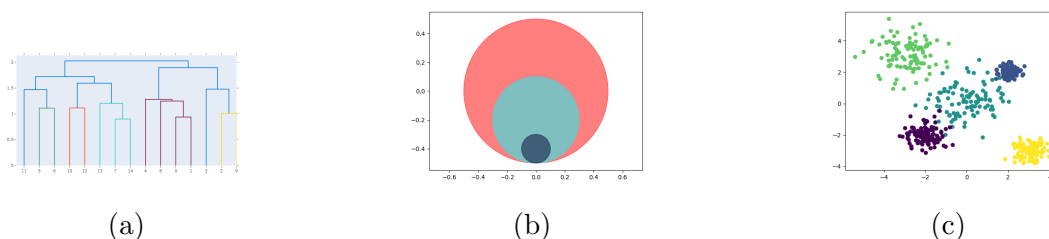


Figura 2.1: Dendrograma, diagrama de Venn anidado y esquema particional

2.2. Contexto histórico del K-Means

El algoritmo de *K-Means* tiene su aparición a mediados del siglo pasado. Diversos autores como [Bock, 2007] señalan que la necesidad de clasificar y obtener patrones e información de conjuntos de datos surgió en diferentes áreas durante las décadas de 1950 y 1960. No solo se interesaron por estas técnicas matemáticos e ingenieros, también lo hicieron psicólogos, taxónomos, biólogos y expertos en marketing. Se publicaron entonces diversos trabajos con objetivos similares a lo que nosotros hemos denominado análisis clúster. Ejemplos de los diferentes nombres que fue tomando son: ‘*Taxonomía Numérica*’ [Sneath et al., 1973] o ‘*Aprendizaje por Observación*’ [Michalski and Stepp, 1983]. Según Jain, la primera vez que se introdujo la palabra ‘clustering’ aplicada al tratamiento estadístico de la información fue en 1954 en un artículo dedicado a la antropología.

Todos estos desarrollos multidisciplinarios paralelos fueron poco a poco confluyendo a una investigación conjunta que se aplica en diferentes ámbitos [Bock, 2007], lo que ayudó a que en las siguientes décadas se empezaran a publicar libros con un enfoque más global y completo. Se publicaron entonces libros como ‘*Algorithms for Clustering Data*’ [Jain and Dubes, 1988] o ‘*Cluster Analysis for Applications*’ [Anderberg, 1973]. La investigación en materia teórica (no tanto en aplicaciones) quedó reservada entonces a las matemáticas, ingeniería y estadística, que tomaron el relevo de la necesaria formalización e investigación de estas nuevas técnicas que habían surgido por necesidad en otras áreas.

En estos primeros trabajos, se empezaba ya a hacer la distinción entre análisis clúster jerárquico y particional. Dentro del particional, el algoritmo del *K-Means* fue propuesto independientemente por diversos investigadores en las décadas de 1950 y 1960 siguiendo distintas líneas de trabajo. Ejemplos de ello pueden ser ‘*Sur la division des corps matériels en parties*’ [Steinhaus et al., 1956]. o ‘*ISODATA, a novel method of data analysis and pattern classification*’ [Ball and Hall, 1965].

El método con el que fue propuesto el algoritmo de *K-Means* en esta etapa prematura se basaba en la distancia euclídea. Esta técnica se asentó por su potencia y sencillez y se utiliza en muchos casos hoy en día. Como abordaremos en el capítulo 3, esta no es la única función de distancia que se puede utilizar, tal como

se recoge en los trabajos de [Kapil and Chawla, 2016], [SAPUTRA et al., 2020] o [Singh et al., 2013], entre otros.

Si bien el algoritmo de *K-Means* tiene más de 50 años, aún constituye una de las bases del análisis clúster particional por su sencillez y eficacia. Forma parte del conocimiento básico de cualquier científico de datos, y su explicación en cualquier manual sobre el tema es imperativa. Además, como comentaremos en el apartado de futuras líneas de trabajo, existe aún margen en la investigación en determinados aspectos muy interesantes del algoritmo.

2.3. El algoritmo

Una vez entendido el contexto bajo el cual surge el algoritmo *K-Means* y sus principales usos, vamos a comenzar con la explicación de la intuición que hay detrás del método.

El algoritmo *K-Means* es una generalización de un razonamiento que surge de manera muy natural. En un conjunto de datos de cualquier dimensión, si fijamos el número de grupos que queremos generar k , el algoritmo nos otorga k grupos en los que todos los elementos dentro de cada grupo son los más similares entre sí.

Algunas introducciones que distintos autores nos aportan sobre algoritmo son:

- ‘... determine a partition of the data into K groups, or *clústers*, such that the data in a *clúster* are more similar to each other than to data in different *clústers*’ [Jain and Dubes, 1988]
- ‘*K-means clustering is a simple and elegant approach for partitioning a data set into K distinct, non-overlapping clústers. To perform K-means clustering, we must first specify the desired number of clústers K ; then the K-means algorithm will assign each observation to exactly one of the K clústers*’ [James et al., 2013]

Existen varios conceptos comunes en los que las definiciones previas incurren. El primero de ellos es sin duda la agrupación, aunque también nos encontramos otros como la similitud. El lector podría ahora plantearse dos preguntas muy elementales: ¿Qué significa agrupar? y ¿Cómo sabemos si dos elementos son lo suficientemente similares como para agruparlos?

La primera condición que uno necesita para agrupar datos es una forma de representarlos. Es por esto que las técnicas de visualización son casi tan importantes en ciencia de datos como las técnicas de aprendizaje. La visualización de datos bidimensionales es la más sencilla y aquella que, teniendo en cuenta cosas

como la escala y la normalización, pueden inducir a menos errores. Existen técnicas para reducir la dimensionalidad de los datos, ya sean lineales o no, pero para simplificar la explicación, nos limitaremos a representar un conjunto de datos que tenga dos dimensiones.

Comencemos suponiendo que poseemos un conjunto de datos de poca dimensionalidad p , esto es, que las variables de las que dependen los datos son pocas, por ejemplo, $p = 2$. Supongamos además que ambas variables son cuantitativas. La representación típica de un conjunto de datos bidimensional es sobre un plano con dos ejes. Un primer acercamiento sobre qué significa agrupar es el que se deriva precisamente de esta idea. Si los datos tienen un aspecto amable, entonces es lógico pensar que se pueden trazar distintos grupos atendiendo a lo que nos sugiera su representación gráfica. Podría haber grupos muy diferenciados, agrupaciones por densidad o quizás la población sea tan homogénea que ninguna clasificación intuitiva resulte satisfactoria ¹.

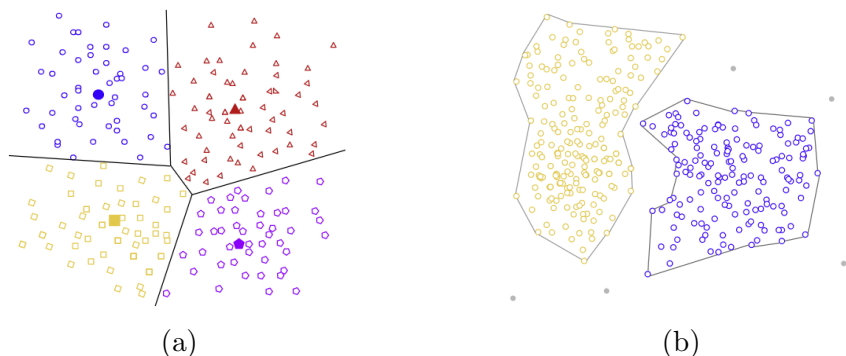


Figura 2.2: Diferentes formas de agrupar datos sobre el plano

Por otro lado, parece claro que en los ejemplos a y b de la figura 2.2 hemos usado diferentes criterios para decidir qué elementos eran similares a otros y debían estar en el mismo clúster. En la primera imagen hemos usado la cercanía en el plano entre elementos, mientras que en el segundo ejemplo hemos usado la cantidad de elementos por unidad en el plano. Esta es la siguiente idea clave que trataremos en este documento. Es posible definir de distintas maneras qué significa que dos elementos de un conjunto sean similares y, por supuesto, podemos estudiar qué definiciones son las más adecuadas para cada conjunto.

Para responder a la segunda pregunta que planteábamos al principio nos hará falta profundizar bastante en el concepto matemático de la distancia. La distancia generaliza matemáticamente conceptos físicos como la cercanía o la similitud, ideas que forman parte de la intuición detrás del algoritmo. En matemáticas, la distancia física a la que estamos acostumbrados recibe el nombre de *'distancia'*

¹En la figura 2.2 observamos agrupaciones en el plano por cercanía a un punto central (a) y por densidad (b).

euclídea’ y las características topológicas que se extraen del concepto de medir una distancia entre dos puntos se han recogido en los llamados ‘*espacios métricos*’.

Existen espacios topológicos a los que si dotamos de una función que satisfaga unas propiedades muy particulares, el espacio con dicha función nos permite relacionar los elementos del conjunto de una manera muy similar a la relación que supone medir distancias entre objetos en el mundo real. Esto nos puede servir para dar nociones de cercanía y de similitud entre elementos, de la misma manera en la que hablamos de dos puntos cercanos en un plano.

Por el momento, y para poder proponer al final de este capítulo un esquema del algoritmo completo, nos bastará con saber que una forma adecuada en matemáticas de expresar la similitud entre dos objetos es la distancia. Imaginemos que en el contexto de un mapa, establecemos que dos ciudades son más similares cuánto más cerca se encuentre una de otra, entonces la distancia usual (a saber, la euclídea) expresa nuestra medida de similitud.

La distancia euclídea fue la función de distancia con la que el algoritmo *K-Means* fue concebido en un primer momento y ha sido objeto de investigación de los últimos años el observar su comportamiento con diferentes funciones. Este será el objetivo del capítulo 3, aunque por el momento daremos todas las definiciones necesarias para plantear el algoritmo basándonos en la distancia euclídea en dimensión 2. A saber:

$$d(A, B) = \sqrt{\sum_{i=1}^2 (a_i - b_i)^2} = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2} \quad (2.1)$$

Siendo $A = (a_1, a_2)$ y $B = (b_1, b_2)$ puntos en \mathbb{R}^2 y $d(A, B)$ la ‘distancia entre A y B’

Definición 1. Dado un conjunto de datos $\mathbf{C} \subset \mathbb{R}^2$, se llama **centroide** $\hat{c} \in \mathbb{R}^2$ al punto que satisface:

$$\hat{c} = \sum_{x \in \mathbf{C}} \frac{x}{|\mathbf{C}|} \quad (2.2)$$

Geométricamente, se podría interpretar el centroide como el centro de masas o la media aritmética entre los puntos. Una definición algo más abstracta que pondremos en contexto más adelante² podría entender el centroide como el punto que minimiza la suma de distancias entre todos los elementos del conjunto. Nótese que el centroide no es necesariamente un elemento del conjunto de datos pero sí del espacio.

El algoritmo se basará en estos centroides. *K-Means* propone que si el número de clústers deseados son k , se fijan k centroides a los que de ahora en adelante

²Ver en el lema 2.3.1

llamaremos \hat{c}_i con $i \in \{1, \dots, k\}$. Estos centroides serán los representantes de cada clúster, y son los elementos con respecto a los cuales se calculará la similitud en cada grupo. Es decir, para cada clúster i , su centroide \hat{c}_i es el elemento ‘más similar’ a todos los integrantes del grupo.

Por tanto, podríamos sintetizar el razonamiento anterior en el siguiente razonamiento algorítmico: Dado un conjunto de datos $\mathbf{C} \subset \mathbb{R}^2$ y $k \in \mathbb{N}$ clúster deseados:

1. Inicializar C_i con $i \in \{1, \dots, k\}$ conjuntos de \mathbb{R}^2 vacíos, que serán nuestros clústers.
2. Tomar k elementos de \mathbf{C} aleatoriamente como centroides \hat{c}_i con $i \in \{1, \dots, k\}$.
3. Calcular la $d(\hat{c}_i, \mathbf{x}) \forall \mathbf{x} \in \mathbf{C}, i \in \{1, \dots, k\}$.
4. Para cada $\mathbf{x} \in \mathbf{C}$, asignar a \mathbf{x} al C_i cuya $d(\hat{c}_i, \mathbf{x})$ sea menor (elemento ‘más similar’).
5. Iterar hasta que la asignación de clúster no cambie:
 - Para cada clúster C_i , recalcular su centroide (1).
 - Asignar a cada $\mathbf{x} \in \mathbf{C}$, el clúster C_i cuya $d(\hat{c}_i, \mathbf{x})$ sea menor.

Sobre la convergencia del *K-Means*

Con el siguiente resultado 2.3.1 nos ayudaremos para demostrar la convergencia del algoritmo *K-Means*. En este primer ejemplo para dimensión 2 con distancia euclídea, aunque su generalización para dimensión n es homóloga.

Lema 2.3.1. Sean $x^1, x^2, \dots, x^m \in \mathbb{R}^2$, con $m \geq 1$ puntos. Sea $\hat{c} = \frac{1}{m} \sum_{i=1}^m x^i$ su centroide, y sea $z \in \mathbb{R}^2$ un punto arbitrario en el espacio 2-dimensional. Entonces:

$$\sum_{i=1}^m \|x^i - z\|^2 \geq \sum_{i=1}^m \|x^i - \hat{c}\|^2.$$

Demostración.

$$\begin{aligned}
\sum_{i=1}^m \|x^i - z\|^2 &= \sum_{i=1}^m \|(x^i - \hat{c}) + (\hat{c} - z)\|^2 \\
&= \sum_{i=1}^m \left(\|x^i - \hat{c}\|^2 + \|\hat{c} - z\|^2 + 2(x^i - \hat{c}) \cdot (\hat{c} - z) \right) \\
&= \sum_{i=1}^m \|x^i - \hat{c}\|^2 + \sum_{i=1}^m \|\hat{c} - z\|^2 + 2 \sum_{i=1}^m (x^i \cdot \hat{c} - x^i \cdot z - \hat{c} \cdot \hat{c} + \hat{c} \cdot z) \\
&= \sum_{i=1}^m \|x^i - \hat{c}\|^2 + m\|\hat{c} - z\|^2 + 2(m\hat{c} \cdot \hat{c} - m\hat{c} \cdot z - m\hat{c} \cdot \hat{c} + m\hat{c} \cdot z) \\
&= \sum_{i=1}^m \|x^i - \hat{c}\|^2 + m\|\hat{c} - z\|^2 \\
&\geq \sum_{i=1}^m \|x^i - \hat{c}\|^2.
\end{aligned} \tag{2.3}$$

Este resultado 2.3.1 nos sugiere entonces que los centroides minimizan la diferencia cuadrada entre todos los puntos dentro del mismo clúster. Este hecho nos servirá si entendemos cuál es el problema de optimización subyacente a los problemas de análisis clúster.

Podemos hablar de la varianza dentro de cada clúster C_k como

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^2 (x_{ij} - x_{i'j})^2 \tag{2.4}$$

En este contexto, queremos que dentro de cada clúster la varianza sea mínima, luego el algoritmo de *K-Means* deberá resolver el siguiente problema de optimización:

$$\underset{C_1, \dots, C_K}{\text{minimize}} \left\{ \sum_{k=1}^K W(C_k) \right\} \tag{2.5}$$

Si se tienen nociones sobre investigación operativa, es fácil darse cuenta que encontrar un mínimo global sobre este problema es bastante complicado a medida que el número de clústers aumenta, por no hablar de la dimensión de los datos. El algoritmo de *K-Means* propone una solución para encontrar óptimos locales, que solo depende de la elección inicial de los centroides.

Por tanto, tenemos por un lado que la varianza dentro del clúster es la mínima posible para esa elección gracias al lema 2.3.1. Por otro lado, en cada iteración por construcción sabemos que la asignación del clúster solo varía si mejora (se reduce) la distancia, esto nos permite establecer que:

Teorema 2.3.2. *Convergencia del K-Means*

El algoritmo K-Means 2.3 converge.

Demostración.

Supongamos que el algoritmo procede de la iteración t a la iteración $t+1$. Basta con demostrar que $W(C_{t+1}) < W(C_t)$. Denotamos que el número de posibles agrupaciones es finito (k^n), el algoritmo debe terminar necesariamente.

Por la construcción del algoritmo, sabemos que termina cuando ningún punto tiene un centroide más cercano que el centroide de su clúster actual, en otras palabras, el clúster actual es localmente óptimo.

Demostramos que la diferencia de cuadrados entre todos los puntos de un clúster y su centroide mejora si existe una reasignación de centroide. Si llamamos $SSQ(C^t, \hat{c}^t)$ (Sum of Squared Errors) a la varianza de clúster C en la iteración t con el centroide \hat{c} de C en la iteración t (sin tener en cuenta el tamaño del clúster), tenemos que:

$$SSQ(C^t, \hat{c}^t) = \sum_{i \in C^t} \|x^i - \hat{c}^t\|^2 \quad (2.6)$$

Nuestro objetivo entonces es demostrar que $SSQ(C^{t+1}, \hat{c}^{t+1}) < SSQ(C^t, \hat{c}^t)$. Lo haremos en dos pasos. En primer lugar, mostramos que $SSQ(C^{t+1}, \hat{c}^t) < SSQ(C^t, \hat{c}^t)$ y, a continuación, mostramos que $SSQ(C^{t+1}, \hat{c}^{t+1}) \leq SSQ(C^{t+1}, \hat{c}^t)$.

El primer paso se sigue directamente de la lógica del algoritmo: C^t y C^{t+1} son diferentes solo si hay un punto que encuentra un centroide más cercano en la iteración $t+1$ que el que le asigna en la t . Luego, si existe un punto $x_i \in C^{t+1}$ que ha cambiado de clúster en la iteración t , su diferencia será menor con ese nuevo centroide y tendremos que:

$$SSQ(C^{t+1}, \hat{c}^t) = \sum_{i \in C^{t+1}} \|x^i - \hat{c}^t\|^2 < \sum_{i \in C^t} \|x^i - \hat{c}^t\|^2 = SSQ(C^t, \hat{c}^t) \quad (2.7)$$

El segundo paso se puede demostrar con ayuda del lema 2.3.1, pues para el clúster en la iteración $t+1$, es su centroide quien minimiza la distancia.

$$SSQ(\mathcal{C}^{t+1}, \hat{c}^{t+1}) = \sum_{i \in \mathcal{C}^{t+1}} \|x^i - \hat{c}^{t+1}\|^2 \leq \sum_{i \in \mathcal{C}^t} \|x^i - \hat{c}^t\|^2 = SSQ(\mathcal{C}^{t+1}, \hat{c}^t) \quad (2.8)$$

Una vez se haya alcanzado la condición de parada, el conjunto de C_i con $i \in \{1, \dots, k\}$ forman nuestros k clústers deseados. Por construcción, resulta evidente que $C_1 \cup C_2 \cup \dots \cup C_K = \mathbf{C}$. Es decir, los clústers generan una partición de \mathbf{C} . También se observa que $C_k \cap C_{k'} = \emptyset$ para cualquier $k \neq k'$. Es decir, los clústers son disjuntos.

El *K-Means* que planteamos en este trabajo es más general que el esquema algorítmico que hemos introducido en este capítulo. Por un lado, su definición se ampliará para espacios n -dimensionales, y por otro, en el capítulo 3 hablaremos de cómo la distancia euclídea no es la única con la que podemos definir el *K-Means*. Al final del próximo capítulo daremos la versión más general posible del algoritmo. Aún así, esta versión simplificada encapsula perfectamente todas las ideas importantes del método.

3

Espacios métricos

Los espacios métricos juegan un papel muy importante en la clusterización y el tratamiento estadístico de los datos. Esto se debe a que la herramienta de la que disponemos los matemáticos para medir la similaridad o regularidad entre diferentes elementos de un conjunto es la distancia. Para ello es necesario, en primer lugar, identificar qué características debemos considerar para comparar los datos y con ellas elaborar una función de similitud, a la que, si cumple una serie de condiciones, llamaremos distancia. En ciencias, existen gran variedad de problemas que han sido estudiados desde el punto de vista de los espacios métricos, ya que el hecho de poder definir una distancia entre elementos de un conjunto nos aporta una manera intuitiva y potente de poder compararlos.

A una función de similitud, para que sea considerada distancia, le vamos a pedir varias condiciones que se derivan de abstraer ciertas características de la distancia euclídea (la natural).

La primera condición que se puede abstraer es que en el plano, dos puntos son el mismo si y solo si la distancia entre ellos es nula. Esto quiere decir que usando cualquier escala, en un plano podemos distinguir que dos puntos son distintos si existe distancia entre ellos, por muy pequeña que sea. Otra característica importante es la que se deriva de la llamada desigualdad triangular. Esto se refiere a que el camino más corto entre dos puntos, es decir, con la distancia mínima, debe ser precisamente la distancia entre esos dos puntos y no se debe reducir pasando por otro punto. Por último, entendemos que la distancia medida entre dos puntos debe ser igual medida desde uno al segundo que viceversa.

Formalmente, se define un espacio métrico como:

Definición 2. [Morris, 1989] Sea X un conjunto no vacío y d una función de valor real definida sobre $X \times X$ tal que para $a, b \in X$:

1. $d(a, b) \geq 0$, y $d(a, b) = 0$ si, y solo si, $a = b$
2. $d(a, b) = d(b, a)$
3. $d(a, c) \leq d(a, b) + d(b, c)$, para toda a, b y c en X (desigualdad triangular).

Entonces d es llamada métrica sobre X , (X, d) es llamado **espacio métrico** y $d(a, b)$ se conoce como la **distancia** entre a y b .

Como bien hemos comentado, si tenemos un espacio de puntos u observaciones, al conjunto de elementos que lo contiene podemos dotarlo de una función que satisfaga las mencionadas características y definir un espacio métrico. En dicho espacio métrico podemos entonces comparar los elementos del conjunto y medir la distancia entre ellos.

Es importante denotar cómo variar la función de distancia es un cambio en un sentido sutil y en otro sentido muy profundo al algoritmo. Por una parte, en la definición de centroide, no participa en ningún momento. Lo mismo pasa en la definición del problema de optimización que el *K-Means* resuelve, la ecuación 2.5. En la varianza del clúster, aunque gráficamente puede parecer que depende de la distancia euclídea, realmente es un estadístico basado en la función de SSQ (Sum of Square Errors), por lo que aunque cambiemos la distancia, estas definiciones no variarán. Lo único que varía es la asignación del clúster, es decir, el cálculo de qué está cerca y qué está lejos. Además, como la convergencia se asegura a raíz de la varianza, la convergencia tampoco queda afectada por el cambio en la distancia. Esto, que a la hora de implementar el algoritmo supone un cambio menor, en la práctica supone que en muchos casos el resultado del análisis clúster sea radicalmente distinto dependiendo de la distancia que utilicemos, como observaremos más adelante.

Procedemos ahora a analizar las funciones de distancia más comunes empleadas con el algoritmo de *K-Means*.

3.1. Distancia euclídea

La distancia euclídea es una medida de la distancia entre dos puntos en un espacio n -dimensional, que se define como la raíz cuadrada de la suma de los cuadrados de las diferencias entre las coordenadas correspondientes de los dos puntos. En otras palabras, si tenemos dos puntos A y B con $A = (a_1, a_2, \dots, a_n)$ y $B = (b_1, b_2, \dots, b_n)$, entonces la distancia euclídea entre ellos se define como:

$$d(A, B) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \dots + (a_n - b_n)^2} = \sqrt{\sum_{i=1}^n (a_i - b_i)^2} \quad (3.1)$$

Se comprueba fácilmente que esta función verifica las condiciones de función de distancia.

La distancia euclídea o euclidiana tiene muchas aplicaciones en matemáticas y física, pues es que encapsula la idea natural de distancia entre dos puntos como la ‘línea recta’ que los une. Se aplica en geometría, análisis numérico, estadística y teoría de la información.

La distancia euclídea es fácil de comprender y representar, aunque computacionalmente no es la operación más eficiente por su naturaleza cuadrática y racional. Veremos en la siguiente sección que en comparación a otras funciones de distancia, puede ser más sensible a valores atípicos en alguna de sus componentes, por lo que no siempre puede ser la más indicada.

Puntos fuertes:

- Es fácil de calcular y entender, lo que lo hace útil en aplicaciones prácticas.
- Funciona bien en problemas con datos robustos y con separación aparente.
- El K-Means se creó para la distancia euclídea, luego garantiza la convergencia y el rigor del algoritmo.

Puntos débiles:

- La distancia euclídea no siempre es la mejor medida de distancia para todos los tipos de datos o problemas. En algunos casos, puede ser demasiado sensible a los valores atípicos y no capturar la estructura subyacente de los datos.
- No suele ser la más adecuada para datos de alta dimensionalidad por su coste computacional.

El esquema algorítmico queda entonces:

El K-Means n-dimensional con la distancia euclídea

Dado un conjunto de datos $\mathbf{C} \subset \mathbb{R}^n$ y $k \in \mathbb{N}$ clústers deseados, se desea resolver el problema de optimización planteado en la ecuación 2.5. Para ello, el *K-Means* se puede escribir como :

1. Inicializar C_i con $i \in \{1, \dots, k\}$ conjuntos de \mathbb{R}^n vacíos, que serán nuestros clústers.
2. Tomar k elementos de \mathbf{C} aleatoriamente como centroides \hat{c}_i con $i \in \{1, \dots, k\}$.
3. Calcular la $d(\hat{c}_i, \mathbf{x}) \forall \mathbf{x} \in \mathbf{C}, i \in \{1, \dots, k\}$. Se toma $d(\hat{c}_i, \mathbf{x})$ como en la ecuación 3.1.
4. Para cada $\mathbf{x} \in \mathbf{C}$, asignar a \mathbf{x} al C_i cuya $d(\hat{c}_i, \mathbf{x})$ sea menor.
5. Iterar hasta que la asignación de clúster no cambie:
 - Para cada clúster C_i , recalcular su centroide(1).
 - Asignar a cada $\mathbf{x} \in \mathbf{C}$, el clúster C_i cuya $d(\hat{c}_i, \mathbf{x})$ sea menor.

Queremos denotar cómo todas las demostraciones y argumentos dados sobre convergencia en 2.3 son generalizables a dimensión n .

3.2. Distancia de Manhattan

La distancia de Manhattan, también conocida como distancia del taxista o distancia L_1 , es una medida de distancia utilizada en espacios métricos que se define como la suma de las diferencias absolutas de las coordenadas entre dos puntos. Es decir, en un espacio de n dimensiones, la distancia de Manhattan entre dos puntos $A = (a_1, a_2, \dots, a_n)$ y $B = (b_1, b_2, \dots, b_n)$ se define como:

$$d(A, B) = |b_1 - a_1| + |b_2 - a_2| + \dots + |b_n - a_n| \quad (3.2)$$

Se comprueba fácilmente que esta función verifica las condiciones de función de distancia.

En ciencia de datos, la distancia de Manhattan se utiliza a menudo para medir la similitud entre dos vectores de características. Al igual que la distancia euclidiana, se puede utilizar en diferentes ámbitos de la física, las matemáticas y la teoría de la información, pero su uso está algo menos extendido.

La distancia de Manhattan es conocida como la ‘distancia del taxista’ o ‘*taxicab distance*’ por su relación con el recorrido que teóricamente un taxista tendría que hacer en una ciudad con manzanas rectangulares en una ciudad como Manhattan visto desde un plano cenital. La distancia toma los valores absolutos en cada componente y los suma, esto es el tamaño de cada una de las calles que tendríamos que recorrer en nuestro teórico entramado de manzanas.

Por la naturaleza de esta función, la distancia de Manhattan es menos sensible a los valores atípicos que la distancia euclídea, lo que la hace más robusta en presencia de datos extremos. Es decir, imaginemos que hay dos puntos muy cercanos en un componente pero muy lejanos en otro, la distancia euclídea al tomar la diagonal entre ambos magnificará dicha diferencia, mientras que la Manhattan suavizará este aspecto, tal como podemos observar en la figura 3.1 [dearC, 2020]. Por esto, la distancia de Manhattan tiene algunos de casos de usos muy característicos, y existen datos donde su uso está muy recomendado. Por ejemplo, en problemas de clasificación de imágenes, donde la distancia euclidiana puede verse afectada negativamente por los píxeles extremos o de ruido, la distancia de Manhattan puede proporcionar una mayor precisión.

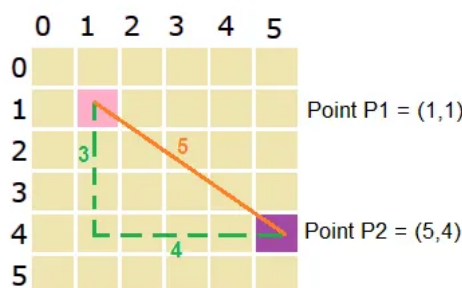


Figura 3.1: Esquema Distancia de Manhattan

Sin embargo, la distancia de Manhattan tiene algunas limitaciones ya que, naturalmente, no siempre es la mejor medida de distancia para todos los tipos de datos. Esta función puede seguir siendo sensible a los datos atípicos dependiendo de su magnitud, y en ocasiones su uso es menos intuitivo, por lo que su uso debe estar justificado y realizarse con cuidado.

Puntos fuertes:

- Es una medida en parte intuitiva, pues posee una justificación natural del problema al que responde.
- La distancia de Manhattan es adecuada para datos que no se distribuyen normalmente, lo que puede ser un problema con otras medidas de distancia como la distancia euclídea.
- Es menos sensible a los valores atípicos que la distancia euclidiana. Los valores extremos tienen un efecto más suave en la medida, lo que puede hacer que sea más robusta en la presencia de datos extremos.
- Se calcula más rápido que la distancia euclidiana, ya que solo implica la suma de las diferencias absolutas de las coordenadas.

Puntos débiles:

- No es adecuada para datos con una alta dimensionalidad, ya que puede perder efectividad a medida que aumenta la cantidad de características.
- Esta distancia no siempre proporciona la misma precisión que la distancia euclídea. Esto se debe a que no siempre su uso está justificado.
- La lógica del algoritmo puede verse ligeramente afectada por su uso, ya que no fue concebido con la misma y existen variaciones del K-Means donde la convergencia no queda afectada.

Para poder aplicar el *K-Means* con esta función de distancia, podemos seguir el siguiente esquema algorítmico.

El K-Means n-dimensional con la distancia de Manhattan

Dado un conjunto de datos $\mathbf{C} \subset \mathbb{R}^n$ y $k \in \mathbb{N}$ clústers deseados, se desea resolver el problema de optimización planteado en la ecuación 2.5. Para ello, el *K-Means* se puede escribir como:

1. Inicializar C_i con $i \in \{1, \dots, k\}$ conjuntos de \mathbb{R}^n vacíos, que serán nuestros clústers.
2. Tomar k elementos de \mathbf{C} aleatoriamente como centroides \hat{c}_i con $i \in \{1, \dots, k\}$.
3. Calcular la $d(\hat{c}_i, \mathbf{x}) \forall \mathbf{x} \in \mathbf{C}, i \in \{1, \dots, k\}$. Se toma $d(\hat{c}_i, \mathbf{x})$ como en la ecuación 3.2.
4. Para cada $\mathbf{x} \in \mathbf{C}$, asignar a \mathbf{x} al C_i cuya $d(\hat{c}_i, \mathbf{x})$ sea menor.
5. Iterar hasta que la asignación de clúster no cambie:
 - Para cada clúster C_i , recalcular su centroide(1).
 - Asignar a cada $\mathbf{x} \in \mathbf{C}$, el clúster C_i cuya $d(\hat{c}_i, \mathbf{x})$ sea menor.

Es necesario comentar cómo, tanto como para la distancia de Manhattan como para las dos siguientes, varios de los argumentos donde nos apoyábamos para demostrar la convergencia en 2.3 se pierden. Esto tiene cierto sentido ya que el algoritmo se concibió únicamente como herramienta a utilizar por la distancia euclídea.

Aún así, realizar las modificaciones necesarias y el estudio pertinente para poder aplicar el algoritmo con estas funciones tiene un interés académico real,

así como un sentido práctico. Por la naturaleza finita de los subconjuntos que se pueden crear con una clusterización, y por el cambio sutil que supone el hecho de variar únicamente la forma de recalcular los centroides, existen pocos casos en los que la convergencia de este algoritmo vaya a suponer un problema. En la mayoría de casos, el hecho de tener este abanico de posibilidades con las que intentar analizar unos datos suponen un beneficio más que un error teórico.

3.3. Distancia de Chebysev

La distancia de Chebysev es una medida de distancia utilizada en espacios métricos que se define como la máxima diferencia entre las coordenadas de dos puntos. Es decir, en un espacio n-dimensional, la distancia de Chebysev entre dos puntos $A = (a_1, a_2, \dots, a_n)$ y $B = (b_1, b_2, \dots, b_n)$ se define como:

$$d(A, B) = \max(|b_1 - a_1|, |b_2 - a_2|, \dots, |b_n - a_n|) \quad (3.3)$$

Se comprueba fácilmente que esta función verifica las condiciones de función de distancia.

A menudo, a la distancia de Chebysev se compara con los movimientos que podría realizar un rey en un tablero de ajedrez. Así, a distancia 1 estaría un cuadrado alrededor de la ficha del rey, otro cuadrado que lo rodea de distancia 2, etc. Ver figura 3.2.


	a	b	c	d	e	f	g	h	
8	5	4	3	2	2	2	2	2	8
7	5	4	3	2	1	1	1	2	7
6	5	4	3	2	1		1	2	6
5	5	4	3	2	1	1	1	2	5
4	5	4	3	2	2	2	2	2	4
3	5	4	3	3	3	3	3	3	3
2	5	4	4	4	4	4	4	4	2
1	5	5	5	5	5	5	5	5	1
	a	b	c	d	e	f	g	h	

Figura 3.2: Esquema Distancia de Chebysev

La distancia de Chebysev se utiliza a menudo para problemas de logística, en los que alguna grúas son capaces de moverse en el eje x, en el eje y y en ambos a la vez, por lo que se transforma en un problema como el del tablero de ajedrez mencionado antes y la distancia de Chebysev ahorra gran cantidad de cálculos.

Además, la distancia de Chebysev es especialmente útil cuando se desea en-

fatizar la similitud en una sola dimensión. Por ejemplo, si se tienen datos de temperaturas en diferentes ciudades, la distancia de Chebysev se puede utilizar para comparar la temperatura máxima en cada ciudad. También es una buena medida de distancia para datos discretos o categóricos, donde las diferencias entre dos valores son 0 o 1.

Por otro lado, esta distancia es bastante más sensible a los datos atípicos que las distancias ya comentadas. Al asignar el valor máximo en cualquiera de las componentes, esto hace que un valor atípico en cualquier coordenada separe mucho a datos que bajo otra definición de distancia no lo estarían. Esto la hace menos robusta en presencia de datos extremos.

En resumen, la distancia de Chebysev es una medida de distancia útil para medir la similitud entre vectores de características en una amplia gama de aplicaciones, y es especialmente útil cuando se desea enfatizar la similitud en una sola dimensión o para datos discretos o categóricos. Sin embargo, su uso debe evaluarse cuidadosamente en función de la naturaleza del problema y el tipo de datos.

Los puntos fuertes de la distancia de Chebysev en la ciencia de datos son:

- Simplicidad: la distancia de Chebysev es fácil de calcular y comprender. Se puede utilizar con datos en diferentes dimensiones y con diferentes tipos de variables, incluyendo variables discretas y continuas.
- Énfasis en una dimensión: la distancia de Chebysev es especialmente útil cuando se desea enfatizar la similitud en una sola dimensión.

Por otro lado, los puntos débiles de la distancia de Chebysev son:

- La distancia de Chebysev no considera la relación entre las diferentes dimensiones (solo el máximo entre ellas), lo que puede hacer que sea menos efectiva en la captura de la estructura subyacente de los datos.
- Problemas con dimensiones escaladas de manera diferente: si las diferentes dimensiones tienen diferentes escalas, la distancia de Chebysev puede dar más peso a las dimensiones con escalas mayores.

Para poder aplicar el *K-Means* con esta función de distancia, podemos seguir el siguiente esquema algorítmico.

El K-Means n-dimensional con la distancia de Chebysev

Dado un conjunto de datos $\mathbf{C} \subset \mathbb{R}^n$ y $k \in \mathbb{N}$ clústers deseados, se desea resolver el problema de optimización planteado en la ecuación 2.5. Para ello, el *K-Means* se puede escribir como :

1. Inicializar C_i con $i \in \{1, \dots, k\}$ conjuntos de \mathbb{R}^n vacíos, que serán nuestros clústers.
2. Tomar k elementos de \mathbf{C} aleatoriamente como centroides \hat{c}_i con $i \in \{1, \dots, k\}$.
3. Calcular la $d(\hat{c}_i, \mathbf{x}) \forall \mathbf{x} \in \mathbf{C}, i \in \{1, \dots, k\}$. Se toma $d(\hat{c}_i, \mathbf{x})$ como en la ecuación 3.3.
4. Para cada $\mathbf{x} \in \mathbf{C}$, asignar a \mathbf{x} al C_i cuya $d(\hat{c}_i, \mathbf{x})$ sea menor.
5. Iterar hasta que la asignación de clúster no cambie:
 - Para cada clúster C_i , recalcular su centroide(1).
 - Asignar a cada $\mathbf{x} \in \mathbf{C}$, el clúster C_i cuya $d(\hat{c}_i, \mathbf{x})$ sea menor.

Mismo comentario sobre su convergencia que en 3.2.

3.4. Distancia de Minkowski

Todas las funciones de distancia previamente mencionadas se pueden generalizar bajo una familia de funciones de distancia llamada la distancia de Minkowski. Esta función es una medida de distancia en espacios métricos que depende de un parámetro $p \in \mathbb{Z}$.

La distancia de Minkowski, para dos puntos $A = (a_1, a_2, \dots, a_n)$ y $B = (b_1, b_2, \dots, b_n)$ se define como:

$$d(A, B) = [|b_1 - a_1|^p + |b_2 - a_2|^p + \dots + |b_n - a_n|^p]^{1/p} \quad (3.4)$$

Donde A y B son los dos puntos que se están comparando, n es el número de dimensiones en el espacio de características y p es un parámetro que controla la ‘forma’ de la distancia.

Nótese que cuando $p = 1$, la distancia de Minkowski es equivalente a la distancia de Manhattan, cuando $p = 2$, es equivalente a la distancia euclidiana y si tomásemos el límite de la expresión cuando p tiende a infinito, tendríamos la distancia de Chebysev.

La distancia de Minkowski solo cumple las condiciones de distancia para valores de $p \geq 1$. Observamos en la siguiente figura 3.3 la forma de varias circunferencias goniométricas en diferentes órdenes de Minkowski [Wikipedia, 2023].

De este diagrama, y de lo expuesto en los puntos anteriores, se deduce que al aumentar el orden de p, la distancia de Minkowski empieza a ser más susceptible a

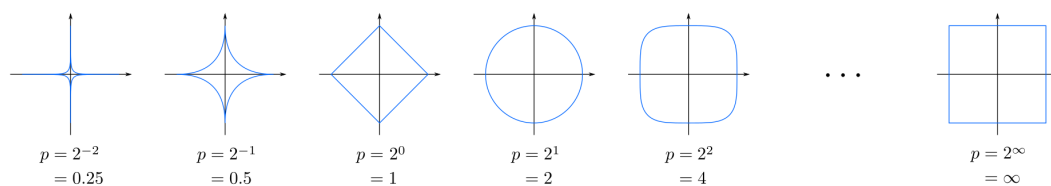


Figura 3.3: Circunferencias goniométricas para diferentes órdenes de p

los valores atípicos en alguna de las componentes de los datos. Es decir, la distancia de Manhattan, para $p = 0$ era menos sensible que la distancia euclídea para $p = 2$. De la misma manera, la euclídea lo será menos que $p = 3$ y así sucesivamente. Dichos cambios son sutiles y nos sugieren que la distancia de Minkowski quizás sea útil para encontrar un orden de p para el cual en nuestro estudio resulte óptimo por el tratamiento de los valores atípicos se refiere. Este estudio se podría hacer por inspección o comparando los resultados hasta encontrar un orden de p que satisfaga nuestras condiciones.

Los puntos fuertes de la distancia de Minkowski en la ciencia de datos son:

- Flexibilidad: La distancia de Minkowski es muy flexible, ya que el parámetro p puede ajustarse según las necesidades del problema.
- Capacidad para capturar diferentes tipos de relaciones: La distancia de Minkowski puede capturar diferentes tipos de relaciones entre las variables.
- Capacidad para manejar diferentes dimensiones: La distancia de Minkowski puede manejar diferentes dimensiones y escalas, lo que la hace adecuada para datos de alta dimensionalidad.

Por otro lado, los puntos débiles de la distancia de Minkowski son:

- Dificultad: Se trata de una función con un parámetro cuya intuición es realmente abstracta.
- Parámetro p : Encontrar el valor del parámetro p puede resultar difícil dependiendo del problema.

Para poder aplicar el *K-Means* con esta función de distancia, podemos seguir el siguiente esquema algorítmico.

El K-Means n-dimensional con la distancia de Minkowski

Dado un conjunto de datos $\mathbf{C} \subset \mathbb{R}^n$ y $k \in \mathbb{N}$ clústers deseados, se desea resolver el problema de optimización planteado en la ecuación 2.5. Para ello, el *K-Means* se puede escribir como:

1. Inicializar C_i con $i \in \{1, \dots, k\}$ conjuntos de \mathbb{R}^n vacíos, que serán nuestros clústers.
2. Tomar k elementos de \mathbf{C} aleatoriamente como centroides \hat{c}_i con $i \in \{1, \dots, k\}$.
3. Calcular la $d(\hat{c}_i, \mathbf{x}) \forall \mathbf{x} \in \mathbf{C}, i \in \{1, \dots, k\}$. Se toma $d(\hat{c}_i, \mathbf{x})$ como en la ecuación 3.4. Se deberá elegir el orden de p que más se ajuste a las necesidades de los datos.
4. Para cada $\mathbf{x} \in \mathbf{C}$, asignar a \mathbf{x} al C_i cuya $d(\hat{c}_i, \mathbf{x})$ sea menor.
5. Iterar hasta que la asignación de clúster no cambie:
 - Para cada clúster C_i , recalcular su centroide(1).
 - Asignar a cada $\mathbf{x} \in \mathbf{C}$, el clúster C_i cuya $d(\hat{c}_i, \mathbf{x})$ sea menor.

Mismo comentario sobre su convergencia que en 3.2.

4

Estudio comparativo del K-Means en distintos espacios métricos

Una vez explicados todos los aspectos teóricos del algoritmo necesarios, es momento de comenzar con el estudio comparativo del *K-Means* en los distintos espacios métricos que hemos introducido.

En primer lugar, para poder plantear y explicar las diferentes cuestiones sobre el *K-Means*, necesitaremos un conjunto de datos o ‘*dataset*’ sobre el que aplicarlo. Primero generaremos un ‘*dataset sintético*’, que no es más que un conjunto de datos generados artificialmente que, si bien no responden a ninguna medición ni experimento en el mundo real, tienen unas características que intentan emular un comportamiento lo más verosímil posible. A este primer dataset lo vamos a dotar de unas características muy básicas con las que podamos observar la intuición del algoritmo con sus diferentes variantes y con las que podamos ejemplificar correctamente los puntos explicados en los anteriores capítulos. Después, pondremos a prueba el método usando el *K-Means* con las distintas distancias sobre la conocida base de datos *MNIST*, para observar su eficacia sobre un caso real.

Todo este estudio se realizará con diversos códigos escritos en Python 3.11.3, con la ayuda de los paquetes `pandas`, `numpy` y `time` en sus versiones más recientes. Los ejemplos se ejecutarán sobre un equipo con Windows 11, un Intel Core i7-10510U y 16Gb de memoria RAM.

Se adjuntarán en los anexos al documento todos códigos utilizados para la elaboración del estudio, aunque algunos segmentos del mismo se irán explicando en las siguientes secciones en favor de una clara comprensión de los resultados.

4.1. Implementación del algoritmo

Existen varias opciones a la hora de implementar un algoritmo tan conocido como el *K-Means*. La primera pasa por elegir el lenguaje de programación más adecuado para la tarea y la segunda tiene que ver con el uso o no de librerías, paquetes o proyectos preexistentes.

En el caso del *K-Means*, para la elaboración de este estudio en el ámbito de la ciencia de datos, se planteaban dos grandes alternativas con dos variantes cada una. La primera de ellas era usar el lenguaje R, usando o no librerías o implementaciones preexistentes. Por otro lado, existía la alternativa de realizar las pruebas en Python, usando o no implementaciones preexistentes. Nuestra elección finalmente fue Python con una implementación propia por diversos motivos, que pasamos ahora a comentar.

Como el estudio que nos ocupa pretende alterar características muy profundas del algoritmo y algo abstractas, no todas las librerías existentes de Python nos permitían realizar estos cambios con libertad absoluta ya que, con buen criterio, esto puede trastocar seriamente la convergencia del algoritmo. Por otro lado, para la programación tan libre que buscábamos, preferimos Python por su estilo sencillo y más interpretado.

Somos plenamente conscientes de que el hecho de realizar una nueva implementación desde cero del algoritmo, podría ocasionar que dicha implementación no fuese la más eficaz computacionalmente hablando. Aún así, nos garantiza que el orden de complejidad será el mismo para cualquiera de las funciones de distancia que usemos, además de otorgarnos un completo control sobre las estructuras de datos con la información de los puntos y sus clústers.

4.1.1. K-Means. El algoritmo en Python

Para la implementación del *K-Means* hemos intentado ser lo más fieles posible a los algoritmos propuestos en 3.1, 3.2 y 3.3. El algoritmo usando la distancia de Minkowski lo apartaremos pues su uso se reserva a casos más específicos.

En primer lugar, definimos las tres funciones de distancia para $n = 2$ ¹, de la siguiente manera:

```
1 import numpy as np
2
3 ### Distancia Euclidea para n=2
4 def euclidean_distance(point1, point2):
5     return np.sum(np.sqrt((point1[0]-point2[0])**2+(point1[1]-
6         point2[1])**2))
```

¹Más adelante lo haremos para dimensión n

```

6
7 ### Distancia de Manhattan para n=2
8 def manhattan_distance(point1, point2):
9     return np.abs((point1[0]-point2[0]))+np.abs((point1[1]-
10         point2[1]))
11 ### Distancia de Chebysev para n=2
12 def Chebysev_distance(point1, point2):
13     return max(np.abs((point1[0]-point2[0])),np.abs((point1[1]-
14         point2[1])))

```

Código 4.1: Distancias en 2 dimensiones

Una vez definidas las distancias, podemos usarlas en Python si las importamos correctamente para aplicarlas a nuestros programas.

Pasamos ahora a explicar la implementación del algoritmo *K-Means* que hemos realizado usando el ejemplo de la distancia euclídea. El resto de versiones, para las otras funciones de distancia, son completamente homólogas y se pueden consultar en el Anexo A en los apartados A.2, A.3 y A.4.

```

1 ### Algoritmo de K-Means para la distancia Euclidea n=2
2 import pandas as pd
3 import numpy as np
4
5 from distances import euclidean_distance
6
7 def kmeans_euclidean(data, k, max_iterations=100):
8     # Inicializacion de los centroides
9     data_df=pd.DataFrame(data)
10    centroids = data_df.sample(k)
11    centroids=centroids.to_numpy()
12    var_centroids=True
13    it=0
14
15    while var_centroids and it in range(max_iterations):
16        # Asignar puntos al cluster mas cercano
17        clusters = [[] for _ in range(k)]
18        copia=[[] for _ in range(k)]
19        for point in data:
20            distances = [euclidean_distance(point, centroid)
21                for centroid in centroids]
22            cluster_index = np.argmin(distances)
23            clusters[cluster_index].append(point.tolist())
24
25        # Actualizar los centroides
26        for i in range(k):
27            if clusters[i]:

```

```

27         copia[i]=centroids[i].copy()
28         centroids[i] = np.mean(clusters[i], axis=0)
29
30     # Condicion de parada
31     var_centroids = any(
32         copia[i][0] != centroids[i][0] or copia[i][1] !=
33         centroids[i][1]
34         for i in range(len(copia))
35     )
36     it+=1
37     return clusters, centroids, it

```

Código 4.2: K-Means con distancia euclídea con $n = 2$

Vemos que para realizar el análisis clúster, la función `kmeans_euclidean()` necesitará 3 parámetros de entrada. Necesitará un vector con los datos, k el número de clústers deseados y un parámetro de iteraciones máximas del algoritmo, para controlar los casos no convergentes si los hubiera, que por defecto se fija en 100.

Comienza entonces el proceso visto en 3.1 y se inicializan los centroides de manera aleatoria. Se toma una muestra de k elementos de los datos con la función `df.sample(k)` de la librería `pandas`. Por otro lado, iniciamos la variable booleana `var_centroids` a `True`, que hace referencia a la ‘variación de centroides’ y que será nuestra condición de parada, al igual que la variable `it` que responde a un contador de ‘iteraciones’.

Después encontramos la parte iterativa del algoritmo. Dentro del primer bucle sucede que mientras los centroides sigan cambiando y no se hayan alcanzado el número máximo de iteraciones (`while var_centroids=True and it in range(max_iteraciones)`) se realizarán los siguientes pasos. Creamos una lista `clusters` con k grupos en los que nuestros puntos irán entrando al asignarse al clúster k -ésimo. Entonces, para cada punto, se calcula su distancia ² a todos los centroides con `euclidean_distance(point, centroid)`.

Después se asigna el punto al clúster cuyo centroide esté más cerca del mismo y, una vez terminado este proceso, se recalculan los centroides haciendo la media aritmética de cada punto en el clúster. Se almacena el estado antiguo de clústers en `copia`.

Se analiza entonces la condición de parada, es decir, se comprueba si alguno de los centroides ha cambiado para que `var_centroids` se mantenga en `True`. Si ninguno ha cambiado, se convierte en `False` y no se ejecuta la siguiente iteración.

²En este caso euclídea, pero con la distancia deseada en las demás variaciones del algoritmo. Simplemente sustituir `euclidean_distance()` por `manhattan_distance()` o `Chebysev_distance()`. Ver Anexo A en los apartados A.2, A.3 y A.4.

La función nos devuelve el último estado de los clústers, centroides y el conteo de la última iteración realizada.

4.2. K-Means sobre datasets sintéticos

Como ya hemos introducido, para mostrar los aspectos más básicos del algoritmo y las principales diferencias entre las funciones de distancia, vamos a aplicar primero la técnica sobre datos generados artificialmente. A este proceso de generación de bases de datos para entrenamiento de algoritmos que se basen en un posible caso real se le denomina ‘dataset sintético’ y constituye una rama de estudio de la ciencia de datos en sí misma. Para nuestro problema, será suficiente que seamos capaces de crear unos conjuntos de datos muy sencillos en 2 dimensiones.

Nuestros datos se van a generar en base a unas nubes de puntos gaussianas. Es decir, diferentes nubes de puntos que sigan una distribución normal multivariante (en este caso en dimensión 2). El objetivo del algoritmo va a ser intentar identificar dichas nubes de puntos con la mayor exactitud posible. Los distintos parámetros de generación de las nubes son: el centro y su varianza o dispersión. Con ellas podremos ajustar casos más simples y otros más complejos. Por último añadiremos datos atípicos que cumplan ciertas condiciones para ver cómo se comportan las distintas variantes del algoritmo.

4.2.1. Primer ejemplo: Nubes simples

Para nuestro primer ejemplo, primero crearemos unas nubes de puntos gaussianas muy simples. Nos hemos decantado por 5 nubes de 100 puntos cada una, haciendo un total de 500 puntos para analizar. En Python, podemos generar estos datos con la ayuda de la librería numpy de la siguiente manera:

```
1 import numpy as np
2 ### Generamos nube de puntos simple
3
4 ## Definimos los parametros de las cinco distribuciones gaussianas
5 mean1 = [-2, -2] # Media de la primera distribucion gaussiana
6 cov1 = [[0.5, 0], [0, 0.5]] # Matriz de covarianza de la primera distribucion gaussiana
7
8 mean2 = [2.5, 3] # Media de la segunda distribucion gaussiana
9 cov2 = [[0.5, 0], [0, 0.5]] # Matriz de covarianza de la segunda distribucion gaussiana
10
```

```

11 mean3 = [0, 0] # Media de la tercera distribucion gaussiana
12 cov3 = [[1, 0], [0, 1]] # Matriz de covarianza de la tercera
    distribucion gaussiana
13
14 mean4 = [-3, 2.5] # Media de la cuarta distribucion gaussiana
15 cov4 = [[0.8, 0], [0, 0.8]] # Matriz de covarianza de la
    cuarta distribucion gaussiana
16
17 mean5 = [2, -3] # Media de la quinta distribucion gaussiana
18 cov5 = [[0.4, 0], [0, 0.4]] # Matriz de covarianza de la
    quinta distribucion gaussiana
19
20 ## Generamos los datos sinteticos para cada distribucion
    gaussiana
21 np.random.seed(0)
22 data1 = np.random.multivariate_normal(mean1,cov1, 100)
23 np.random.seed(1)
24 data2 = np.random.multivariate_normal(mean2,cov2, 100)
25 np.random.seed(2)
26 data3 = np.random.multivariate_normal(mean3,cov3, 100)
27 np.random.seed(3)
28 data4 = np.random.multivariate_normal(mean4,cov4, 100)
29 np.random.seed(4)
30 data5 = np.random.multivariate_normal(mean5,cov5, 100)

```

Código 4.3: Generación de dataset de nubes simples

Vemos como, indicando los centros y la matriz de covarianzas para cada nube, la función `np.random.multivariate_normal()` nos permite generar las nubes con los parámetros deseados. La función `np.random.seed()` almacena la semilla aleatoria de la generación de los puntos para que el estudio sea replicable en distintas ejecuciones.

Los datos aparecen representados por grupos en la figura [4.1](#).

Se observan 5 núcleos claramente diferenciados, de los cuales el más central (verde) tiene algo más de dispersión para ver cómo las distintas distancias lo manejan. Hemos buscado algo de solape entre las nubes para que el algoritmo tenga que trabajar ligeramente y no sea excesivamente sencillo separar las nubes.

Para realizar el análisis y valorar la eficiencia y eficacia de las distintas variantes del algoritmo, vamos a considerar 3 parámetros. El primero el tiempo de ejecución, el segundo las iteraciones realizadas hasta alcanzar un resultado y por último el número de aciertos que ha conseguido el algoritmo. Definimos un acierto en nuestro contexto si el clúster final del punto es el mismo a la nube en el cual se generó.

El tiempo de ejecución lo obtendremos haciendo uso del paquete `time` y su

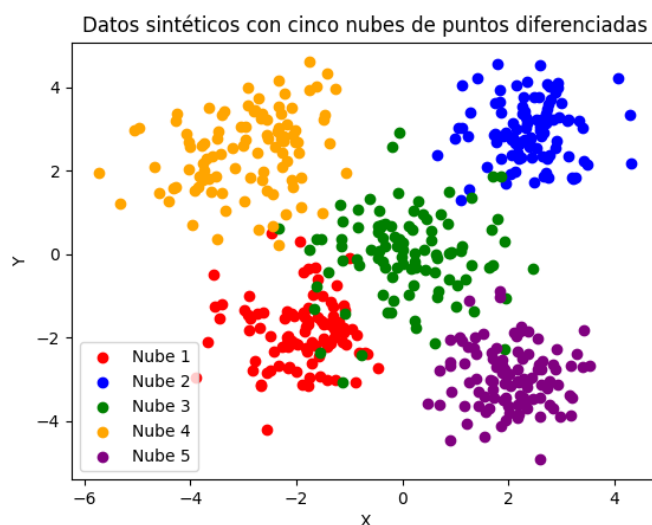


Figura 4.1: Datos sintéticos con 5 nubes diferenciadas

función `time()`, que consigue el tiempo del sistema antes y después de la ejecución del algoritmo. Si los restamos podremos saber cuánto tardó el algoritmo en alcanzar la condición de parada. Recordamos que las iteraciones son uno de los outputs del algoritmo implementado en [A.2](#).

Para el número de aciertos hemos creado la función `comprobar()`. Esta función, para confirmar si la asignación del clúster coincide con la nube generadora, ejecuta las siguientes instrucciones:

```

1     import numpy as np
2
3     def comprobar(lista1, lista2, k=5):
4         acierto=[0 for _ in range(k)]
5         for i in range(k):
6             for elemento in lista1:
7                 for array in lista2[i]:
8                     if np.array_equal(elemento, array):
9                         acierto[i]+=1
10    print(acierto)
```

Código 4.4: Función comprobar

`comprobar()` toma como parámetros de entrada dos listas de puntos y el número de clústers k . Nótese que tanto el output de clústers que nos otorga la función `kmeans_euclidean()` o cualquiera de sus variantes y también las nubes que conforman el dataset son listas de puntos (siendo cada punto un array de dimensión 2). La función desempeña el siguiente razonamiento lógico: para cada punto en la nube 'lista1', mira cuántos de los puntos en cada clúster pertenecen a

esa nube y lo toma como acierto³. Esta función nos sirve para comprobar también que ningún punto se contabiliza más de una vez y que la ejecución es correcta, ya que si la ejecutamos sobre cada nube, la suma de todos los aciertos debe ser igual al número de datos.

Para ver la eficacia del algoritmo utilizaremos una típica matriz de confusión de la tabla 4.1 que evalúe los resultados acertados, falsos negativos y falsos positivos.

	Predicted Condition	
	Positive	Negative
	True Positive (TP) <i>hit</i>	False Negative (FN) <i>Underestimation, type II error</i>
Actual Condition	False Positive (FP) <i>Overestimation, type I error</i>	True Negative (TN) <i>Correct rejection</i>

Tabla 4.1: Matriz de confusión

Los Verdaderos Positivos o ‘True Positive’ (**TP**) serán contabilizados como los puntos que fueron generados por una nube y pertenecen al clúster que agrupa a la mayoría de puntos de dicha nube. Los Falsos Negativos o ‘False Negative’ (**FN**) serán los elementos que perteneciendo a la nube generadora, no han terminado perteneciendo al correspondiente clúster. Los Falsos Positivos o ‘False Positive’ (**FP**) serán los elementos que, si bien se encuentran clasificados en un clúster, no pertenecían a la nube generadora que lo originó.

Los resultados Verdaderos Negativos o ‘True Negative’ (**TN**) no nos interesan en este caso concreto pues significarían todos los puntos que no pertenecían a la nube y el algoritmo deja fuera del clúster correspondiente.

Con estos valores podremos obtener una medida de la calidad de los resultados del algoritmo. Dicha medida se llama **precisión** y se calcula como en 4.1.

$$\frac{TP}{TP + FP} \quad (4.1)$$

Queremos denotar que un análisis de matriz de confusión suele ser un enfoque más adecuado para un trabajo de clasificación, no para uno de clusterización. Si bien no es el enfoque más típico para el *K-Means*, en este caso nos permite hacernos una idea sobre cómo trabajan las variaciones del algoritmo, y nos permitirá sacar conclusiones sobre el manejo que hacen las funciones de distancia sobre varias características de los datos.

³Nótese que la función no sabe *a priori* qué clúster es el que corresponde a cada nube o si existe dicha correspondencia. Tomamos como clúster principal que organiza esa nube el grupo con más aciertos y el resto los contabilizaremos como errores

K-Means con distancia euclídea para un dataset simple

Empezamos ejecutando el algoritmo con la distancia euclídea. Esto es:

```

1 import numpy as np
2 import time
3
4 from Euclidean import *
5 ## Concatenamos los datos en un unico vector
6 total = np.concatenate((data1,data2,data3,data4,data5), axis=0)
7
8 ## K-Means con Euclidean
9 start_time = time.time()
10 clusters_eu, centroids_eu, it_eu = kmeans_euclidean(total,5,
11     max_iterations=1000)
12 end_time = time.time()
13 execution_time = end_time - start_time
14 print(it_eu, execution_time)
15
16 ## Comprobacion de la distribucion de los puntos en cada
17 cluster
18 comprobar(data1,clusters_eu)
19 comprobar(data2,clusters_eu)
20 comprobar(data3,clusters_eu)
21 comprobar(data4,clusters_eu)
22 comprobar(data5,clusters_eu)

```

Código 4.5: K-Means con distancia euclídea para un dataset simple

Que nos da como salida el siguiente resultado *# 10 0.26439976692199707*. Es decir, ha realizado 10 iteraciones en aproximadamente 0.2643 segundos. Nos será posible poner en contexto estos resultados cuando ejecutemos las siguientes pruebas.

La distribución por clústers (columnas) de los puntos pertenecientes a cada nube se encuentra en la tabla 4.2 y en la figura 4.2.

	C1	C2	C3	C4	C5
Nube 1	0	2	0	97	1
Nube 2	0	1	99	0	0
Nube 3	4	85	3	7	1
Nube 4	0	1	0	1	98
Nube 5	98	2	0	2	2

Tabla 4.2: Distribución de cada nube para la distancia euclídea en un dataset simple

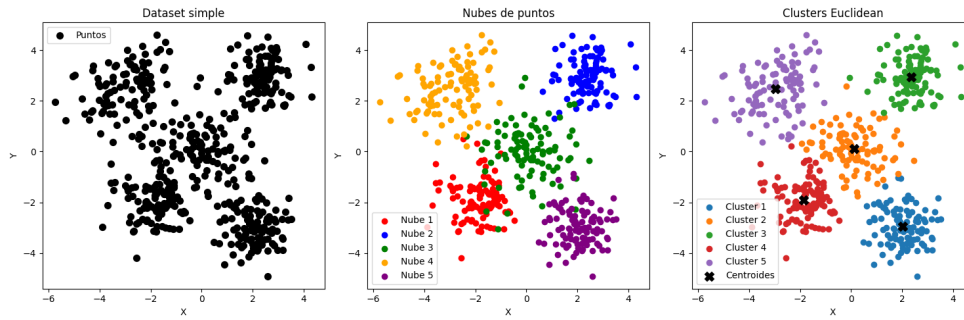


Figura 4.2: K-Means con distancia euclídea para el dataset simple

Vemos que los elementos de la primera nube han sido agrupados en el cuarto clúster. De ese clúster, 97 elementos fueron colocados acertadamente, 3 elementos que debían pertenecer al clúster no lo hacían (False Negative) y en total 8 elementos que no debían pertenecer al clúster pertenecen (False Positive). Para la segunda nube, se encuentra en el tercer clúster y los resultados son: 99 True Positives, 1 False Negative y 3 False Positive. Para la tercera nube (segundo clúster) encontramos: 85 True Positives, 15 False Negatives y 6 False Positives. La cuarta (quinto clúster) ha obtenido: 98 True Positives, 2 False Negatives y 2 False Positives. La ultima nube (primer clúster) consigue unos resultados de: 98 True Positives, 2 False Negatives y 4 False Positives.

Por tanto, en total tenemos el recuento en la tabla 4.3

		Clúster Predicho	
Nube		477 (TP)	23 (FN)
		23 (FP)	-

Tabla 4.3: Matriz de confusión para un dataset simple con la distancia euclídea

Con estos valores de la matriz de confusión podemos calcular que la precisión del algoritmo para esta distancia euclídea en el dataset simple es: **0.954**.

K-Means con distancia de Manhattan para un dataset simple

Para la distancia de Manhattan, tenemos que el código para realizar el análisis clúster es:

```
1 import numpy as np
2 import time
3
4 from Manhattan import *
5
6 ## Concatenamos los datos en un unico vector
7 total = np.concatenate((data1,data2,data3,data4,data5), axis=0)
```

```

8
9 ### K-Means con Manhattan
10 start_time = time.time()
11 clusters_mn, centroids_mn, it_mn = kmeans_manhattan(total,5)
12 end_time = time.time()
13 execution_time = end_time - start_time
14 print(it_mn, execution_time)
15
16 ## Comprobacion de la distribucion de los puntos en cada
   cluster
17 comprobar(data1,clusters_mn)
18 comprobar(data2,clusters_mn)
19 comprobar(data3,clusters_mn)
20 comprobar(data4,clusters_mn)
21 comprobar(data5,clusters_mn)

```

Código 4.6: K-Means con distancia de Manhattan para un dataset simple

Observamos que la única diferencia con 4.5 es la llamada a la función de `kmeans_manhattan()` que calcula las distancias de los puntos al clúster con la distancia de Manhattan implementada en A.1.

Los resultados que nos arroja el código 4.6 son `# 6 0.06431818008422852`. Es decir, 6 iteraciones con un tiempo de ejecución de aproximadamente 0.0643 segundos.

El desglose por clúster de cada nube gaussiana que hemos obtenido se observa en la tabla 4.4 y en la figura 4.3.

	C1	C2	C3	C4	C5
Nube 1	2	0	1	97	0
Nube 2	1	0	0	0	99
Nube 3	85	4	1	7	3
Nube 4	2	0	98	0	0
Nube 5	1	99	0	0	0

Tabla 4.4: Distribución de cada nube para la distancia de Manhattan en un dataset simple

Realizando el mismo análisis confusión que en 4.2.1, es decir, sumando por columnas los FP y por filas los FN, tenemos que la matriz de confusión quedaría como en la tabla 4.5.

Nos aporta una precisión de: **0.956**.

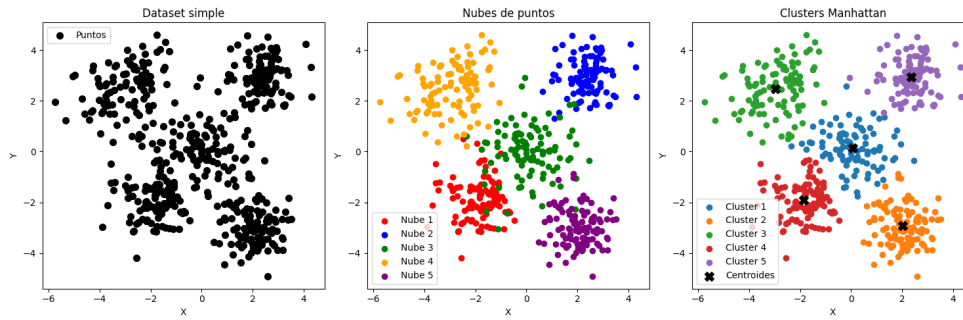


Figura 4.3: K-Means con distancia de Manhattan para el dataset simple

Nube	Clúster Predicho	
	478 (TP)	22 (FN)
	22 (FP)	-

Tabla 4.5: Matriz de confusión para un dataset simple con la distancia de Manhattan

K-Means con distancia de Chebysev para un dataset simple

Por último, para la distancia de Chebysev, el código es:

```

1 import numpy as np
2 import time
3
4 from Chebysev import *
5
6 ## Concatenamos los datos en un unico vector
7 total = np.concatenate((data1,data2,data3,data4,data5), axis=0)
8 ### K-Means con Chebysev
9 start_time = time.time()
10 clusters_ch, centroids_ch, it_ch = kmeans_Chebysev(total,5)
11 end_time = time.time()
12 execution_time = end_time - start_time
13 print(it_ch, execution_time)
14
15 ## Comprobacion de la distribucion de los puntos en cada
   cluster
16 comprobar(data1,clusters_ch)
17 comprobar(data2,clusters_ch)
18 comprobar(data3,clusters_ch)
19 comprobar(data4,clusters_ch)
20 comprobar(data5,clusters_ch)

```

Código 4.7: K-Means con distancia de Chebysev para un dataset simple

Observamos que la única diferencia con 4.5 y 4.6 es la llamada a la función de `kmeans_Chebysev()` que calcula las distancias de los puntos al clúster con la distancia de Chebysev implementada en A.1. Los resultados que nos arroja el algoritmo son: `# 8 0.08747005462646484`. Es decir, 8 iteraciones con un tiempo de ejecución de aproximadamente 0.0875 segundos.

El desglose por clúster de cada nube gaussiana es el contenido de la tabla 4.6 y la figura 4.4.

	C1	C2	C3	C4	C5
Nube 1	97	0	2	1	0
Nube 2	0	0	0	1	99
Nube 3	8	3	1	83	5
Nube 4	1	0	99	0	0
Nube 5	0	97	0	3	0

Tabla 4.6: Distribución de cada nube para la distancia de Chebysev en un dataset simple

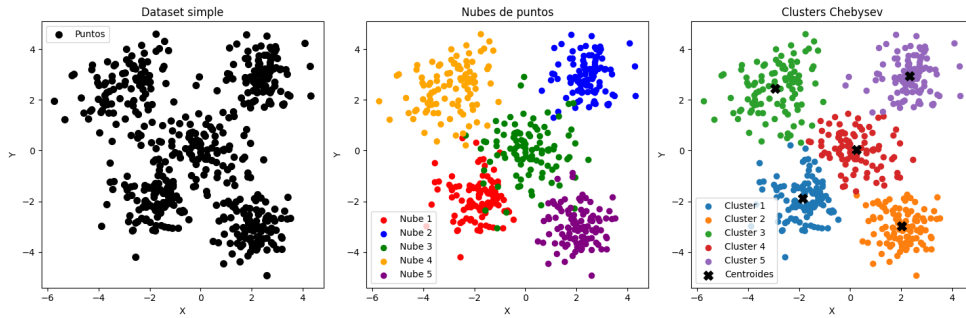


Figura 4.4: K-Means con distancia de Chebysev para el dataset simple

Estos resultados vistos en una matriz de confusión se muestran en la tabla 4.7.

	Clúster Predicho	
Nube	475 (TP)	25 (FN)
	25 (FP)	-

Tabla 4.7: Matriz de confusión para un dataset simple con la distancia de Chebysev

La precisión es: **0.95**.

Resultados

De los resultados obtenidos en los ejemplos anteriores podemos obtener varias conclusiones.

En primer lugar, hay que poner en valor que los resultados de los 3 análisis son muy positivos. Valores de precisión superiores al 0.95 son extremadamente altos y posiblemente no son superiores por el ligero solape que existe entre algunas de las nubes. Esto resalta el buen funcionamiento y la robustez del algoritmo para los casos básicos como el que hemos planteado.

Si los datos no plantean ninguna característica especial en la cual alguna de las distancias destaque, los 3 algoritmos han sido capaces de rescatar de manera más que aceptable las nubes de puntos con sus respectivos centros.

Si consideramos las iteraciones y el tiempo de ejecución, observamos que el algoritmo que más iteraciones ocupó fue el *K-Means* con distancia euclídea, 10. Esto supone una diferencia muy despreciable con las 6 u 8 que realizaron las otras dos alternativas, y más teniendo en cuenta la posibilidad de que la aleatoriedad en la inicialización de los centroides para el algoritmo pudo haber ocasionado un caso menos favorable en términos de convergencia.

El resultado que no es despreciable, y que en los siguientes ejemplos analizaremos más en profundidad, es el tiempo de ejecución. La magnitud del tiempo de ejecución del *K-Means* con distancia euclídea es significativamente mayor, entre 3 y 4 veces mayor a los demás, y dicho aumento no se explica por el mayor número de iteraciones. Si bien las estructuras de datos y su manejo en los tres algoritmos es exactamente el mismo, existe una diferencia sustancial en materia de complejidad computacional en el cálculo de la distancia. Eso es la naturaleza cuadrática de la distancia euclídea, es decir, el hecho de que cada componente esté elevado al cuadrado. Esto supone que para cada dimensión de los datos, se lleve a cabo una operación ‘extra’ que no se está realizando para las otras distancias. Si bien en este caso tan sencillo, computacionalmente esta diferencia es más que asumible, veremos más adelante como afecta a otros casos con mayor dimensión. Este hecho hace patente lo comentado en la sección 3.1 en los puntos débiles de la distancia.

4.2.2. Segundo ejemplo: Nubes simples con outliers

Para este ejemplo, tomaremos el algoritmo para dos dimensiones de la sección anterior esta vez aplicado a un dataset nuevo. Para crear este conjunto de datos, buscaremos que tenga mayor dispersión para otra vez 5 nubes de 100 puntos cada una. Además, añadiremos algunos ‘outliers’ o valores atípicos en las nubes. Buscamos que se evidencien las diferencias y los puntos fuertes de cada una de las funciones de distancia que comentamos en el Capítulo 3.

Para crear las nubes tomaremos un código muy similar al utilizado en 4.3:

```

1 import numpy as np
2
3 ### Generamos nube de puntos simple
4
5 ## Definimos los parametros de las cinco distribuciones gaussianas
6 mean1 = [-3, -3] # Media de la primera distribucion gaussiana
7 cov1 = [[1.5, 0], [0, 1.5]] # Matriz de covarianza de la
   primera distribucion gaussiana
8
9 mean2 = [3, 3.5] # Media de la segunda distribucion gaussiana
10 cov2 = [[1.5, 0], [0, 1.5]] # Matriz de covarianza de la
   segunda distribucion gaussiana
11
12 mean3 = [0, 0] # Media de la tercera distribucion gaussiana
13 cov3 = [[2, 0], [0, 2]] # Matriz de covarianza de la tercera
   distribucion gaussiana
14
15 mean4 = [-3, 2.5] # Media de la cuarta distribucion gaussiana
16 cov4 = [[1.8, 0], [0, 1.8]] # Matriz de covarianza de la
   cuarta distribucion gaussiana
17
18 mean5 = [2.5, -3] # Media de la quinta distribucion gaussiana
19 cov5 = [[2.4, 0], [0, 2.4]] # Matriz de covarianza de la
   quinta distribucion gaussiana
20
21
22 # Generar los datos sinteticos para cada distribucion gaussiana
23 np.random.seed(0)
24 o_data1 = np.random.multivariate_normal(mean1, cov1, 100)
25 np.random.seed(1)
26 o_data2 = np.random.multivariate_normal(mean2, cov2, 100)
27 np.random.seed(2)
28 o_data3 = np.random.multivariate_normal(mean3, cov3, 100)
29 np.random.seed(3)
30 o_data4 = np.random.multivariate_normal(mean4, cov4, 100)

```

```

31 np.random.seed(4)
32 o_data5 = np.random.multivariate_normal(mean5, cov5, 100)
33
34 # Generar algunos datos con medidas atípicas
35 outliers1 = np.array([[6, 6], [-6, -9], [0,
36     1],[10,0],[9,-5],[-8,-2]])
37 outliers2=np.array([[6, 2], [-6, -1], [3,
38     1],[10,4],[9,-4],[-6,-2]])
39 outliers3=np.array([[6, -6], [-6, 4], [0,
40     -1],[10,-3],[4,-5],[3,-2]])
41 outliers4=np.array([[6, 1], [-6, 7], [0,
42     14],[10,-10],[-9,-5],[8,-2]])
43 outliers5=np.array([[6, -2], [-6, -7], [0,
44     -1],[10,1],[9,1],[-8,2]])
45 # Concatenar los datos con medidas atípicas a los datos
    anteriores
46 o_data1 = np.concatenate((o_data1, outliers1), axis=0)
47 o_data2 = np.concatenate((o_data2, outliers2), axis=0)
48 o_data3 = np.concatenate((o_data3, outliers3), axis=0)
49 o_data4 = np.concatenate((o_data4, outliers4), axis=0)
50 o_data5 = np.concatenate((o_data5, outliers5), axis=0)

```

Código 4.8: Generación de dataset con outliers

Observamos que el único fragmento que varía son las medidas atípicas introducidas manualmente. Con las medidas atípicas encontramos que hay un total de 530 observaciones, como se puede ver en la figura 4.5. Además, se ha aumentado considerablemente la dispersión de los datos mediante las magnitudes en las matrices de covarianzas.

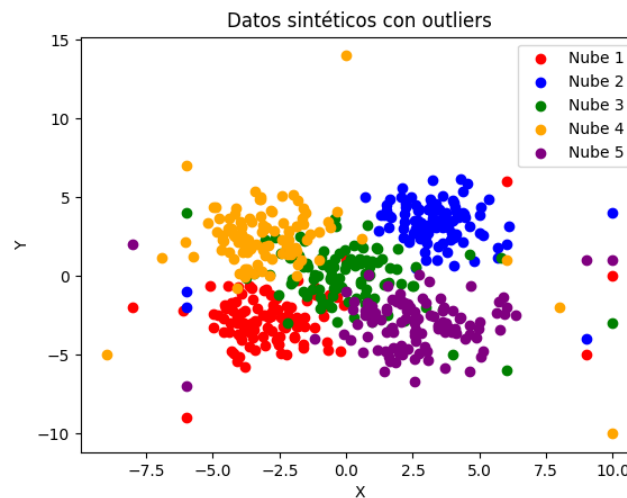


Figura 4.5: Dataset con outliers

Pasamos ahora a realizar un análisis completamente homólogo al del dataset simple. Los códigos utilizados no plantean ninguna diferencia sustancial y se pueden consultar en el Anexo A en el código A.6.

K-Means con distancia euclídea para un dataset con outliers

La salida del algoritmo ha sido: `# 19 0.47806239128112793`. Es decir, ha realizado 19 iteraciones en aproximadamente 0.47806 segundos. Observamos que el tiempo de ejecución y las iteraciones son más elevadas debido a la complejidad de los datos.

La distribución por clústers de los puntos pertenecientes a cada nube es la presentada en la tabla 4.8 y la figura 4.6.

	C1	C2	C3	C4	C5
Nube 1	2	0	1	93	10
Nube 2	102	0	1	2	1
Nube 3	4	13	9	2	79
Nube 4	2	97	2	2	3
Nube 5	2	1	79	2	23

Tabla 4.8: Distribución de cada nube para la distancia euclídea en un dataset con outliers

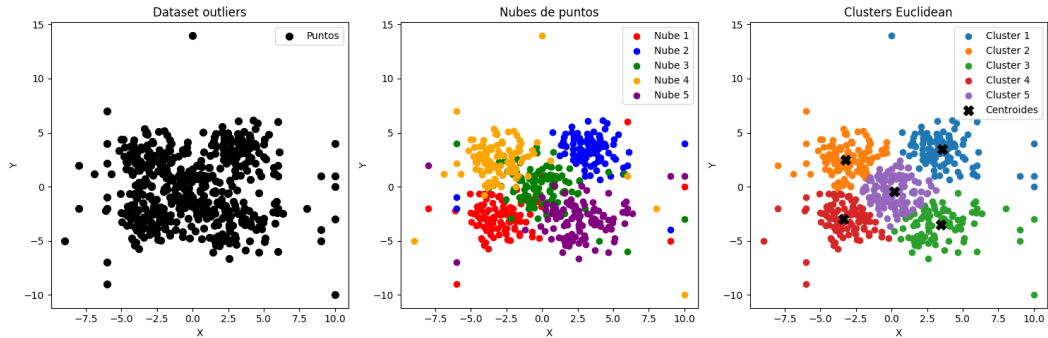


Figura 4.6: K-Means con distancia euclídea para el dataset con outliers

Con estos datos, el análisis de matriz de confusión se resume en la tabla 4.9.

	Clúster Predicho	
Nube	450 (TP)	82 (FN)
	82 (FP)	-

Tabla 4.9: Matriz de confusión para un dataset con outliers con la distancia euclídea

Con estos valores de la matriz de confusión podemos calcular que la precisión del algoritmo para esta distancia euclídea en el dataset con outliers es: **0.849**.

K-Means con distancia de Manhattan para un dataset con outliers

En este caso el algoritmo ha alcanzado la condición de parada con resultados: **# 13 0.17791390419006348**. Ha realizado 13 iteraciones en aproximadamente 0.17781 segundos.

La distribución por clústers de los puntos pertenecientes a cada nube es la que encontramos en la tabla 4.10 y en la figura 4.7:

	C2	C2	C3	C4	C5
Nube 1	9	2	1	0	94
Nube 2	1	102	1	0	2
Nube 3	78	4	9	13	3
Nube 4	8	1	2	93	2
Nube 5	13	2	89	1	2

Tabla 4.10: Distribución de cada nube para la distancia de Manhattan en un dataset con outliers

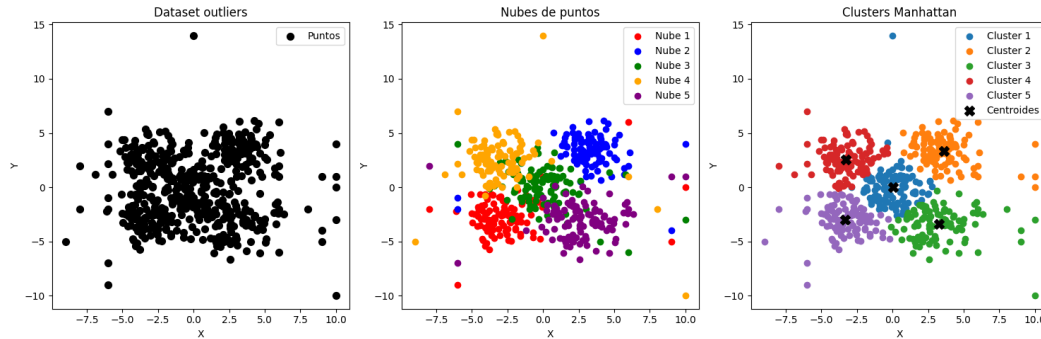


Figura 4.7: K-Means con distancia de Manhattan para el dataset con outliers

Con estos datos, el análisis de matriz de confusión se resume en la tabla 4.11

Nube	Clúster Predicho	
	456 (TP)	76 (FN)
	76 (FP)	-

Tabla 4.11: Matriz de confusión para un dataset con outliers con la distancia de Manhattan

La precisión del algoritmo queda: **0.8603**.

K-Means con distancia de Chebysev para un dataset con outliers

Hemos obtenido [# 14 0.1952073574066162](#). Ha realizado 14 iteraciones en aproximadamente 0.1952 segundos.

La distribución por clústers de los puntos pertenecientes a cada nube se ve en la tabla [4.12](#) y figura [4.8](#).

	C1	C2	C3	C4	C5
Nube 1	3	2	2	98	1
Nube 2	0	6	0	2	98
Nube 3	32	2	37	14	22
Nube 4	1	2	98	3	2
Nube 5	91	13	0	3	0

Tabla 4.12: Distribución de cada nube para la distancia de Chebysev en un dataset con outliers

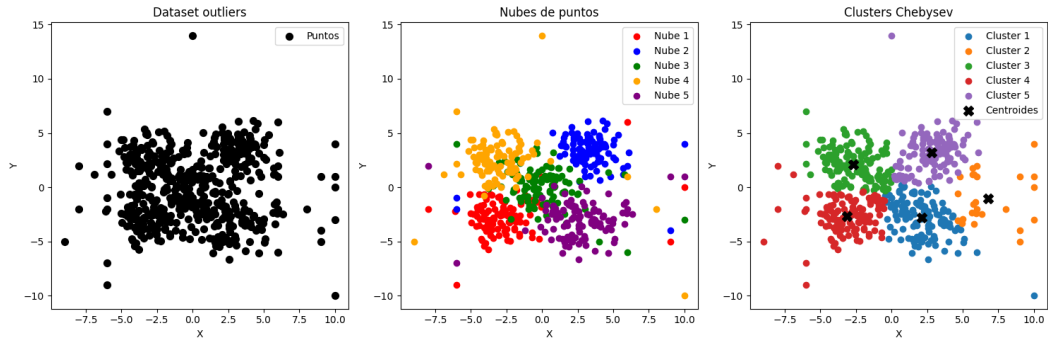


Figura 4.8: K-Means con distancia de Chebysev para el dataset con outliers

Con estos datos, el análisis de matriz de confusión se resume en [4.13](#)

Nube	Clúster Predicho	
	417 (TP)	115 (FN)
	115 (FP)	-

Tabla 4.13: Matriz de confusión para un dataset con outliers con la distancia de Chebysev

Con estos valores de la matriz de confusión podemos calcular que la precisión del algoritmo: **0.7868**.

Resultados

Las nubes de puntos ciertamente tienen un mayor solape y la inclusión de outliers aporta una nueva dimensión en la que los algoritmos han demostrado su manejo de los mismos. Nos damos cuenta que tanto el número de iteraciones que han ocupado los análisis, como el tiempo de ejecución han aumentado considerablemente. El *K-Means* con distancia euclídea sigue teniendo ejecuciones más largas claramente, utilizando un 40 % más de iteraciones para alcanzar tiempos de ejecución del orden de 2 a 3 veces más que sus alternativas. La precisión que ha alcanzado la distancia euclídea es 0,849.

Si nos centramos en la distancia de Manhattan y la distancia de Chebysev, observamos que claramente la distancia de Manhattan ha alcanzado resultados más positivos. La distancia de Chebysev, como se puede observar en el gráfico 4.8, ha interpretado la existencia de un clúster que no se corresponde con ninguna nube de puntos, en parte parece que por el peso que aportan todos esos valores atípicos en el lado derecho del plano.

Por todo ello, podemos afirmar que la distancia de Manhattan es la que mejor se ha comportado en este entorno con outliers. Esto concuerda perfectamente con el desarrollo teórico comentado anteriormente, que nos indicaba que esta distancia era capaz de minimizar en gran medida algunos casos de outliers. Queremos destacar el aspecto de ‘algunos casos’, ya que como podemos observar, la distancia de Chebysev, si bien también matemáticamente contrarresta los datos extremos, en este caso particular no se ha comportado como esperábamos. Se mantiene el mayor coste computacional de la distancia euclídea, que si bien con estos tiempos es absolutamente despreciable, mantiene la idea de que a altas dimensiones cobrará más importancia. Los datos de precisión de la distancia euclídea, no siendo tan buenos como la de Manhattan, son muy positivos.

4.3. K-Means sobre la base de datos MNIST

La base de datos MNIST (Modified National Institute of Standards and Technology) es una colección de imágenes de dígitos escritos a mano. Fue creada por el Instituto Nacional de Estándares y Tecnología de Estados Unidos y se ha convertido en un conjunto de datos ampliamente utilizado en el campo de la inteligencia artificial y el aprendizaje automático. Esta base de datos contiene 60,000 imágenes de entrenamiento y 10,000 imágenes de prueba. Cada imagen es en blanco y negro, tiene un tamaño de 28x28 píxeles y muestra un solo dígito del 0 al 9. Estas imágenes fueron recopiladas de formularios de solicitud y tarjetas postales escritas a mano.

El propósito principal de la base de datos MNIST es proporcionar un conjunto

de datos estándar para probar y comparar diferentes algoritmos de reconocimiento de patrones y aprendizaje automático. Se utiliza comúnmente como punto de partida para los investigadores y estudiantes que desean desarrollar y evaluar modelos de aprendizaje automático para reconocimiento de dígitos.

En nuestro caso, vamos a tomar cada dígito del 0 al 9 como un clúster a determinar, como si fueran cada una de las nubes de los ejemplos anteriores. En ese espacio 28^2 -dimensional, el objetivo del *K-Means* en este caso será intentar discernir cada uno de estos 10 clústers.

En la base de datos observamos una primera columna con una etiqueta que nos indica a priori el dígito con el que estamos trabajando. Esta columna se excluirá del análisis y la usaremos para comprobar nuestros resultados. Después encontramos 784 columnas correspondientes a cada uno de los píxeles que conforman el número. A cada píxel, se le otorga un valor en la escala de grises indicando su color. El aspecto que tiene la base de datos y una representación de lo que podría ser un dato se encuentran en la figura 4.9. Utilizaremos el conjunto de entrenamiento de MNIST, el cual lo componen 60000 datos, teniendo entonces un tamaño de base de datos de 60000×784 .

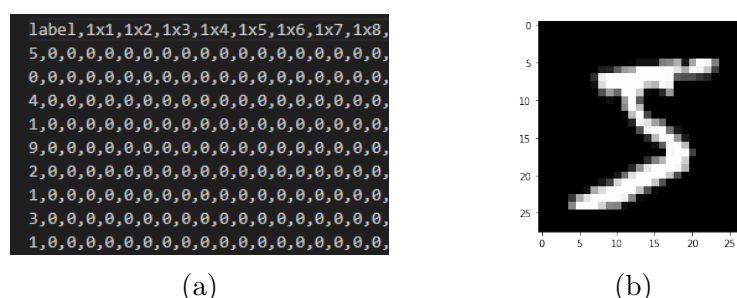


Figura 4.9: Base de datos MNIST.

4.3.1. El código

Para la realización de este análisis vamos a tener que adaptar los scripts utilizados en dimensión 2 utilizados en la sección anterior y adecuarlos para dimensión n .

```
1 import numpy as np
2
3 ### Distancia Euclidea n-dimensional
4 def euclidean_n_distance(point1, point2):
5     if len(point1) != len(point2):
6         raise ValueError("Los puntos deben tener la misma
7             cantidad de dimensiones")
8     squared_distance = sum((coord1 - coord2) ** 2 for coord1,
9         coord2 in zip(point1[1:], point2[1:]))
```

```

8
9     return math.sqrt(squared_distance)
10
11 ### Distancia de Manhattan n-dimensional
12 def manhattan_n_distance(point1, point2):
13     if len(point1) != len(point2):
14         raise ValueError("Los puntos deben tener la misma
15                             cantidad de dimensiones")
16     return sum(abs(coord1 - coord2) for coord1, coord2 in zip(
17         point1[1:], point2[1:]))
18
19 #### Distancia de Chebysev n-dimensional
20 def chebysev_n_distance(point1, point2):
21     if len(point1) != len(point2):
22         raise ValueError("Los puntos deben tener la misma
23                             cantidad de dimensiones")
24     return max(abs(coord1 - coord2) for coord1, coord2 in zip(
25         point1[1:], point2[1:]))

```

Código 4.9: Distancias n-dimensionales

Se observa en 4.9 que la única diferencia con A.1 es la utilización de la función `sum()` para hacer sumatorios y la exclusión de la primera coordenada en `zip(point1[1:], point2[1:])` para ambos puntos por ser la etiqueta.

Una vez definidas las nuevas funciones de distancia, el único fragmento de código que varía es la llamada a la función de distancia entre el punto y el centroide. Por esto, el *K-Means* utilizado en esta sección es homólogo al usado en A.2. Se puede consultar el código exacto en los anexos A.7, A.8 y A.9.

La función que utilizaremos para comprobar los resultados para este análisis sí va a variar un poco con respecto a los ejemplos anteriores. Para este caso, tomaremos la lista que contiene los clústers, y para cada uno de ellos analizaremos todos los puntos que contienen. Como los puntos poseen su etiqueta, vamos a contar dentro de cada clúster cuántos elementos de cada número fueron asignados al clúster, y tomaremos como el número principal al que posea más asignaciones. El resto serán tomados como errores. Todo esto lo representaremos en una matriz de 10×10 en la cual cada fila será un clúster y cada columna un número. En la sección de resultados 4.3.2 se observará con más detalle esta representación. El código lo encontramos en 4.10.

```

1 import numpy as np
2
3 def comprobar_n(clusters):
4     # Crea una matriz de 10x10 llena de ceros
5     conteo = np.zeros((10, 10))
6
7     # Asignamos a cada columna los valores de las etiquetas

```

```

8     for i in range(len(clusters)):
9         for elemento in clusters[i]:
10             index=elemento[0]
11             conteo[i][index] +=1
12     print(conteo)
13     print(conteo.sum())

```

Código 4.10: Función comprobar n-dimensional

4.3.2. Resultados

Al realizar el análisis para las tres funciones de distancia, hemos obtenido la clusterización de los 60000 datos MNIST. En las tablas 4.15, 4.16 y 4.17 están representados los resultados de la función `comprobar()` para cada una de ellas. El código de la ejecución principal se puede consultar en el anexo A.10. Las ejecuciones se han demorado: 4 iteraciones con 723.057 segundos para la distancia euclídea, 4 iteraciones con 511.804 segundos para la Manhattan y 4 iteraciones con 564.501 segundos para la Chebysev. Estos resultados nos confirman la premisa teórica del más alto coste computacional de la distancia euclídea, viendo como ahora supone un aumento de casi el 70 %.

Antes de pasar a analizar los resultados uno por uno, queremos resaltar a modo informativo el desglose por etiquetas de la base de datos MNIST en la tabla 4.14. Esta información nos ayudará a poner en valor la calidad de los resultados que discutiremos a continuación.

	Count
0	5923
1	6742
2	5958
3	6131
4	5842
5	5421
6	5918
7	6265
8	5851
9	5949

Tabla 4.14: Recuento de etiquetas de MNIST

Los resultados que nos otorgan las funciones `comprobar` son:

El *K-Means* con la distancia euclídea parece que ha tenido algunas dificultades para identificar los clústers relativos a algunos números. Podemos analizar la

	0	1	2	3	4	5	6	7	8	9
C1	43	14	2586	51	172	38	4072	12	53	63
C2	14	1	67	51	155	46	2	3914	24	913
C3	9	11	258	1952	127	859	5	10	1214	244
C4	38	10	361	79	5033	664	110	1328	257	4066
C5	2309	0	500	74	32	96	390	12	36	25
C6	75	12	100	2467	0	1277	104	0	374	59
C7	107	173	1183	295	70	681	272	319	3231	197
C8	21	6519	781	378	239	907	711	638	466	331
C9	857	2	114	779	5	841	202	19	176	13
C10	2450	0	8	5	9	12	50	13	20	38

Tabla 4.15: MNIST para la distancia euclídea

tabla 4.15 por filas o por columnas. Si realizamos un análisis por filas, estaríamos comprobando que cada clúster se corresponde con un solo número, es decir, la facilidad del algoritmo para relacionar cada punto con una única etiqueta. En un análisis por columnas, si la mayoría de puntos se encuentran en un único clúster, estamos valorando la facilidad con la que el algoritmo agrupa puntos con la misma etiqueta dentro del mismo clúster.

Con todo ello, vamos a considerar que el algoritmo ha agrupado los puntos de la siguiente manera:

- En el primer clúster o **C1**: puntos con etiqueta 6, aunque ha alcanzado cierto nivel de confusión con el 2.
- En el segundo clúster o **C2**: puntos con etiqueta 7.
- En el tercer clúster o **C3** y en el sexto clúster o **C6**: etiqueta 3, con cierta confusión con el 8 y el 5.
- En el cuarto clúster o **C4**: etiqueta 4, con bastante confusión con el 9.
- En el quinto clúster o **C5** y en el décimo clúster o **C10**: se han agrupado los elementos con etiqueta 0.
- En el séptimo clúster o **C7**: elementos de etiqueta 8 con una confusión moderada con el 2.
- En el octavo clúster o **C8**: etiqueta 1.
- En el noveno clúster o **C9**: sin clasificación aparente.

Parece que el algoritmo no ha sido capaz de distinguir un clúster para los elementos con etiqueta 9, y ha asignado dos clústers para separar el 0. Tomaremos

estos dos clústers como correctos para el 0 (es decir (C5,0) y (C10,0) como aciertos) y todos los elementos del 9 como errores. Sucederá lo mismo con los clústers C3 y C6, tomaremos su agrupación del 3 como correcta y todos los elementos del 5 como errores. Los elementos del C9 serán todos tomados como errores, pues no han obtenido una etiqueta principal aparentemente. El algoritmo tampoco ha podido asignar un único clúster a los datos con etiqueta 2.

Con todo esto, para nuestro análisis de matriz de confusión tomaremos como **True Positives** los elementos en las celdas (C1,6), (C2,7), (C3,3), (C4,4), (C5,0), (C6,3), (C7,8), (C8,1) y (C10,0). Todos ellos sumados hacen un total de **31.947 TP's** y una precisión final del 53,245 %.

	0	1	2	3	4	5	6	7	8	9
C1	35	3	92	38	1576	138	5	4589	89	1730
C2	17	0	101	4	1037	20	262	39	30	651
C3	132	3123	1019	1024	699	977	1809	802	604	566
C4	98	8	113	598	1852	1524	17	251	2433	2601
C5	624	0	180	15	320	131	2968	18	51	94
C6	3458	0	27	7	1	18	46	9	13	17
C7	21	0	201	2085	6	778	3	4	935	68
C8	107	3608	2893	1100	328	953	365	552	1385	196
C9	1172	0	1283	96	23	46	433	1	65	7
C10	259	0	49	1164	0	836	10	0	246	19

Tabla 4.16: MNIST para la distancia de Manhattan

El *K-Means* con la distancia de Manhattan también ha tenido dificultades en la clusterización de algunas etiquetas. Podemos analizar la tabla 4.16 igual por filas o por columnas. De una exhaustivo análisis de los datos consideramos que el algoritmo ha agrupado los puntos de la siguiente manera:

- En el primer clúster o **C1**: puntos con etiqueta 7.
- En el segundo clúster o **C2**: puntos con etiqueta 4 (pocos elementos).
- En el tercer clúster o **C3**: etiqueta 1, con cierta confusión con el 6 y en menor medida con el 2 y el 3.
- En el cuarto clúster o **C4**: etiqueta 9, con mucha confusión con el 8 y algo de confusión con el 4 y 5.
- En el quinto clúster o **C5**: se han agrupado los elementos con etiqueta 6.
- En el sexto clúster o **C6**: elementos con etiqueta 0.
- En el séptimo clúster o **C7**: elementos de etiqueta 3 con una confusión moderada con el 8.

- En el octavo clúster o **C8**: etiqueta 1 con gran confusión con el 2.
- En el noveno clúster o **C9**: sin clasificación aparente. Confusión entre el 0 y el 2.
- En el décimo clúster o **C10**: agrupación de elementos con etiqueta 3 (pocos elementos)

Parece que el algoritmo con la distancia de Manhattan no ha sido capaz de distinguir un clúster para los elementos con etiqueta 2, 5 y 8. Esto se debe a que ha obtenido varios clústers que comparten de manera casi igual proporción de elementos entre dos etiquetas. Los puntos con etiqueta 1 se comparten entre los clústers C3 y C8.

Con todo esto, para nuestro análisis de matriz de confusión tomaremos como **True Positives** los elementos en las celdas **(C1,7)**, **(C2,4)**, **(C3,1)**, **(C4,9)**, **(C5,6)**, **(C6,0)**, **(C7,3)** y **(C8,1)**. Todos ellos sumados hacen un total de **23.469 TP's** con una precisión final del 39,115 %.

	0	1	2	3	4	5	6	7	8	9
C1	2519	1288	697	1055	2738	2198	343	665	2608	790
C2	31	30	352	54	16	31	2168	0	9	0
C3	55	26	247	568	1209	195	11	2538	317	3420
C4	38	767	135	602	461	237	10	118	625	196
C5	91	43	211	49	29	73	2192	0	22	0
C6	3060	601	3548	1793	958	950	1116	92	1188	71
C7	31	348	12	168	398	1476	22	671	775	1007
C8	0	764	0	0	0	0	0	0	0	0
C9	0	1	0	4	14	7	0	2084	3	449
C10	98	2874	756	1838	19	254	56	97	304	16

Tabla 4.17: MNIST para la distancia de Chebysev

Para la distancia de Chebysev, se observa claramente como el algoritmo ha generado varios clústers muy grandes como el C1, el C3 o el C6 que agrupan muchos puntos de distintas etiquetas, dejando más vacíos otros con una clasificación con etiquetas más clara.

- En el primer clúster o **C1**: sin clasificación aparente. Clúster muy grande con muchos puntos de distintas etiquetas.
- En el segundo clúster o **C2**: puntos con etiqueta 6.
- En el tercer clúster o **C3**: etiqueta 9, con cierta confusión con el 7 y en menor medida con el 4.

- En el cuarto clúster o **C4**: sin clasificación aparente.
- En el quinto clúster o **C5**: se han agrupado los elementos con etiqueta 6.
- En el sexto clúster o **C6**: elementos con etiqueta 2 y gran confusión con el 0.
- En el séptimo clúster o **C7**: elementos de etiqueta 5 con una confusión moderada con el 9.
- En el octavo clúster o **C8**: etiqueta 1 con muy pocos elementos.
- En el noveno clúster o **C9**: elementos con la etiqueta 7.
- En el décimo clúster o **C10**: agrupación de elementos con etiqueta 1 con cierta confusión con el 3.

Tomaremos las celdas **(C2,6)**, **(C3,9)**, **(C5,6)**, **(C6,2)**, **(C7,5)**, **(C8,1)**, **(C9,7)** y **(C10,1)**. Todos ellos sumados hacen un total de **18.526 TP's** con una precisión del 30,88 %.

Por los resultados obtenidos en las tres pruebas podemos determinar que la distancia euclídea ha sido la más solvente para este conjunto particular de datos. Su separación de los puntos en función de nuestra idea preconcebida otorgada por las etiquetas ha sido ampliamente superior a la de sus rivales en prácticamente un 20 %. Se podría considerar que este resultado es de una magnitud algo baja, pero esto se debe a la alta dimensionalidad de los datos. Para un conjunto de datos de estas características hemos obtenido unos resultados más que aceptables que nos ayudan a cubrir un amplio abanico de posibilidades. Si se deseara mejorar estos resultados, podríamos cambiar el *K-Means* básico que proponemos en este trabajo por alguna otra variación más avanzada que emplease más centroides o aplicar el *K-Means* de forma iterativa.

Se observa que la Manhattan y la Chebysev no han sido capaces de rescatar muy bien muchos de los clústers y que aquellos que ha rescatado se encuentran bastante mezclados. En particular, la distancia de Manhattan parece que ha dejado algunos clústers con menos elementos que otros, cuando su distribución en la tabla 4.14 es bastante homogéneo. Esta circunstancia se hace aún más acusada en la prueba con la distancia de Chebysev.

5

Conclusiones

En este Trabajo de Fin de Grado, se ha llevado a cabo un análisis exhaustivo del algoritmo de *K-Means* y se han realizado contribuciones significativas para su comprensión y análisis. Se han abordado las bases matemáticas del algoritmo, destacando su rigurosidad y ofreciendo una explicación intuitiva para su comprensión general. Además, se ha introducido el concepto de espacio métrico y se ha adaptado el algoritmo para utilizarlo con diferentes distancias, como la distancia de Manhattan, Chebysev y Minkowski. De esta primera parte teórica del documento podemos destacar todo el trabajo de abstracción que supone comparar dos elementos de un conjunto ‘midiendo’ su similitud. Además, las conclusiones obtenidas sobre convergencia y optimalidad tienen profundas implicaciones en la modificación del algoritmo, y suponen una base teórica sólida que hacen del *K-Means* la herramienta tan polivalente que es.

En la segunda parte del trabajo, el estudio comparativo, hemos alcanzado un control absoluto de todas las características de la técnica gracias a realizar una implementación tan inédita. Si bien esto ha podido tener influencia sobre la eficiencia del algoritmo, para estos volúmenes de datos no se presenta como un problema insalvable. Se pretendía realizar un completo análisis sobre qué tipos de datos son más adecuados para cada función de distancia, pero la realidad es que la naturaleza aleatoria del algoritmo y las diferencias entre ejecuciones hacen muy difícil completar dicha recomendación. Aún así, se han conseguido verificar la mayoría de aspectos fuertes y débiles que teóricamente fueron deducidos, lo cual supone un potente punto de partida. Con los dos primeros ejemplos, más sencillos, se ha constatado que aunque el algoritmo fuese en un primer momento propuesto como una técnica de algún modo euclídea (y así se evidencia en su convergencia),

la función de distancia se puede modificar y la técnica sigue funcionando. Es más, no es solo que funcione, si no que en determinados casos tiene sentido teórico y práctico el comparar los puntos con otras funciones de distancia.

La base de datos MNIST, si bien es muy conocida por su uso para entrenamiento de técnicas de visión artificial y reconocimiento de patrones y no tanto por su aplicación a problemas de clusterización, nos ha servido para tener un conjunto de datos de alta dimensionalidad con una clasificación clara y accesible. Aunque los resultados no aporten una precisión alta, reflejan todos los aspectos teóricos que pretendíamos explicar. Como hemos comentado, quizás con algunas modificaciones se podrían mejorar dichos resultados, pero se ha comprobado el funcionamiento del algoritmo en entornos no solo computacionalmente más demandantes, si no también teóricamente más complejos.

En conclusión, este Trabajo Fin de Grado ha abordado de manera exhaustiva el algoritmo de *K-Means*, proporcionando una comprensión profunda de sus bases matemáticas y explorando adaptaciones para mejorar su rendimiento utilizando diferentes distancias. Se espera que tanto la explicación de su posición dentro del ámbito del aprendizaje automático, así como el contexto histórico aportado sirvan al lector para encuadrar correctamente la importancia de esta técnica. Se ha evidenciado el carácter central que este algoritmo ostenta no solo en el análisis clúster, si no en la ciencia de datos en general.

Asimismo, los experimentos realizados han revelado las limitaciones del algoritmo. Este hecho sin duda propone nuevos escenarios de estudio y mejoras a realizar. Aún así, se ha demostrado la capacidad del algoritmo para identificar agrupaciones en situaciones diversas, lo cual era el principal objetivo que vertebraba este trabajo.

6

Trabajos futuros

Una vez alcanzados los objetivos de este documento, se plantean ahora una serie de futuras líneas de trabajo que consideramos serían interesantes de cara a completar una visión más profunda sobre el *K-Means*.

En primer lugar proponemos adentrarse en los distintos órdenes de la distancia de Minkowski. Elegir un orden de p es un problema de inspección a priori, pero se puede alcanzar ligeras mejorías para cada problema en particular si se encuentra el orden de p óptimo para cada conjunto de datos. Es posible que para los ejemplos que hemos realizado en este documento, para alguno de ellos exista algún orden de p que alcance valores de precisión ligeramente más altos que los 3 propuestos.

Por otro lado, es importante comentar un aspecto teórico del algoritmo que hemos pasado por alto, y es la elección inicial de los centroides. En la inicialización al *K-Means* aleatoriamente se eligen k centroides iniciales y sobre estos se comienza el proceso iterativo. Estos centroides iniciales tienen muchísimo peso sobre el resultado de óptimo local que alcanza la técnica. Si se desea estar plenamente seguro de los resultados obtenidos, lo ideal es ejecutar un número elevado de veces el algoritmo y obtener un resultado promedio de agrupación, pues siempre habrá casos con inicializaciones extremas de los centroides que lleven a óptimos locales muy distintos.

Además, existe gran cantidad de literatura y de investigación en lo referente a distintas técnicas que minimizan el impacto aleatorio de la inicialización de los centroides, y aún hay espacio para el desarrollo de nuevas investigaciones. Algunos ejemplos de ello son [Fabregas et al., 2017] o [Yedla et al., 2010].

Bibliografía

- [Anderberg, 1973] Anderberg, M. (1973). *Cluster Analysis for Applications*. Probability and Mathematical Statistics : a series of monographs and textbooks. Academic Press.
- [Ball and Hall, 1965] Ball, G. H. and Hall, D. J. (1965). Isodata, a novel method of data analysis and pattern classification. Technical report, Stanford research inst Menlo Park CA.
- [Bock, 2007] Bock, H.-H. (2007). Clustering methods: a history of k-means algorithms. *Selected contributions in data analysis and classification*, pages 161–172.
- [dearC, 2020] dearC (2020). Log book-guide to distance measuring approaches for k-means clustering.
- [Fabregas et al., 2017] Fabregas, A. C., Gerardo, B. D., and Tanguilig III, B. T. (2017). Enhanced initial centroids for k-means algorithm. *International Journal of Information Technology and Computer Science*, 1:26–33.
- [Jain and Dubes, 1988] Jain, A. K. and Dubes, R. C. (1988). *Algorithms for clustering data*. Prentice-Hall, Inc.
- [James et al., 2013] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*, volume 112. Springer.
- [Kapil and Chawla, 2016] Kapil, S. and Chawla, M. (2016). Performance evaluation of k-means clustering algorithm with various distance metrics. In *2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)*, pages 1–4.
- [Michalski and Stepp, 1983] Michalski, R. S. and Stepp, R. E. (1983). Learning from observation: Conceptual clustering. *Machine learning: An artificial intelligence approach*, pages 331–363.
- [Morris, 1989] Morris, S. A. (1989). *Topology without tears*. University of New England.
- [SAPUTRA et al., 2020] SAPUTRA, D. M., SAPUTRA, D., and OSWARI, L. D. (2020). Effect of distance metrics in determining k-value in k-means clustering using elbow and silhouette method. In *Sriwijaya International Conference on Information Technology and Its Applications (SICONIAN 2019)*, pages 341–346. Atlantis Press.
- [Singh et al., 2013] Singh, A., Yadav, A., and Rana, A. (2013). K-means with three different distance metrics. *International Journal of Computer Applications*, 67(10).
- [Sneath et al., 1973] Sneath, P. H., Sokal, R. R., et al. (1973). *Numerical taxonomy. The principles and practice of numerical classification*.
- [Steinhaus et al., 1956] Steinhaus, H. et al. (1956). Sur la division des corps matériels en parties. *Bull. Acad. Polon. Sci*, 1(804):801.
- [Wikipedia, 2023] Wikipedia (2023). Minkowski distance — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Minkowski_distance&oldid=1149051293. [Online; accessed 30-May-2023].

- [Yedla et al., 2010] Yedla, M., Pathakota, S. R., and Srinivasa, T. (2010). Enhancing k-means clustering algorithm with improved initial center. *International Journal of computer science and information technologies*, 1(2):121–125.

Anexos



Primer anexo

A.1. Códigos

A.1.1. Distancias

```
1 import numpy as np
2 import math
3
4 ### Distancia Euclidea para n=2
5 def euclidean_distance(point1, point2):
6     if len(point1) != len(point2):
7         raise ValueError("Los puntos deben tener la misma
8             cantidad de dimensiones")
9     return np.sum(np.sqrt((point1[0]-point2[0])**2+(point1[1]-
10         point2[1])**2))
11
12 ### Distancia de Manhattan para n=2
13 def manhattan_distance(point1, point2):
14     if len(point1) != len(point2):
15         raise ValueError("Los puntos deben tener la misma
16             cantidad de dimensiones")
17     return np.abs((point1[0]-point2[0]))+np.abs((point1[1]-
18         point2[1]))
19
20 ### Distancia de Chebysev para n=2
21 def Chebysev_distance(point1, point2):
```

```

18     if len(point1) != len(point2):
19         raise ValueError("Los puntos deben tener la misma
                cantidad de dimensiones")
20     return max(np.abs((point1[0]-point2[0])),np.abs((point1[1]-
                point2[1])))
21
22
23 ### Distancia Euclidea n-dimensional
24 def euclidean_n_distance(point1, point2):
25     if len(point1) != len(point2):
26         raise ValueError("Los puntos deben tener la misma
                cantidad de dimensiones")
27     squared_distance = sum((coord1 - coord2) ** 2 for coord1,
                coord2 in zip(point1[1:], point2[1:]))
28
29     return math.sqrt(squared_distance)
30
31 ### Distancia de Manhattan n-dimensional
32 def manhattan_n_distance(point1, point2):
33     if len(point1) != len(point2):
34         raise ValueError("Los puntos deben tener la misma
                cantidad de dimensiones")
35     return sum(abs(coord1 - coord2) for coord1, coord2 in zip(
                point1[1:], point2[1:]))
36
37 #### Distancia de Chebysev n-dimensional
38 def chebysev_n_distance(point1, point2):
39     if len(point1) != len(point2):
40         raise ValueError("Los puntos deben tener la misma
                cantidad de dimensiones")
41     return max(abs(coord1 - coord2) for coord1, coord2 in zip(
                point1[1:], point2[1:]))

```

Código A.1: Funciones de distancia

A.1.2. Algoritmos de K-Means para n=2

```

1 ### Algoritmo de K-Means para la distancia Euclidea n=2
2 import pandas as pd
3 import numpy as np
4
5 from distances import euclidean_distance
6
7 def kmeans_euclidean(data, k, max_iterations=100):
8     # Inicializacion de los centroides
9     data_df=pd.DataFrame(data)

```

```

10 centroids = data_df.sample(k)
11 centroids=centroids.to_numpy()
12 var_centroids=True
13 it=0
14
15 while var_centroids and it in range(max_iterations):
16     # Asignar puntos al cluster mas cercano
17     clusters = [[] for _ in range(k)]
18     copia=[[] for _ in range(k)]
19     for point in data:
20         distances = [euclidean_distance(point, centroid)
21                     for centroid in centroids]
22         cluster_index = np.argmin(distances)
23         clusters[cluster_index].append(point.tolist())
24
25     # Actualizar los centroides
26     for i in range(k):
27         if clusters[i]:
28             copia[i]=centroids[i].copy()
29             centroids[i] = np.mean(clusters[i], axis=0)
30
31     # Condicion de parada
32     var_centroids = any(
33         copia[i][0] != centroids[i][0] or copia[i][1] !=
34         centroids[i][1]
35         for i in range(len(copia))
36     )
37     it+=1
38 return clusters, centroids, it

```

Código A.2: K-Means para la distancia euclídea y n=2

```

1 ### Algoritmo de K-Means para la distancia Manhattan n=2
2 import pandas as pd
3 import numpy as np
4
5 from distances import manhattan_distance
6
7 def kmeans_manhattan(data, k, max_iterations=100):
8     # Inicializacion de los centroides
9     data_df=pd.DataFrame(data)
10    centroids = data_df.sample(k)
11    centroids=centroids.to_numpy()
12    var_centroids=True
13    it=0
14
15    while var_centroids and it in range(max_iterations):
16        # Asignar puntos al cluster mas cercano

```

```

17     clusters = [[] for _ in range(k)]
18     copia=[[] for _ in range(k)]
19     for point in data:
20         distances = [manhattan_distance(point, centroid)
21                       for centroid in centroids]
22         cluster_index = np.argmin(distances)
23         clusters[cluster_index].append(point.tolist())
24
25     # Actualizar los centroides
26     for i in range(k):
27         if clusters[i]:
28             copia[i]=centroids[i].copy()
29             centroids[i] = np.mean(clusters[i], axis=0)
30
31     #Condicion de parada
32     var_centroids = any(
33         copia[i][0] != centroids[i][0] or copia[i][1] !=
34         centroids[i][1]
35         for i in range(len(copia))
36     )
37     it+=1
38     return clusters, centroids,it

```

Código A.3: K-Means para la distancia de Manhattan y n=2

```

1  ### Algoritmo de K-Means para la distancia Chebysev n=2
2  import pandas as pd
3  import numpy as np
4
5  from distances import Chebysev_distance
6
7  def kmeans_Chebysev(data, k, max_iterations=100):
8      # Inicializacion de los centroides
9      data_df=pd.DataFrame(data)
10     centroids = data_df.sample(k)
11     centroids=centroids.to_numpy()
12     var_centroids=True
13     it=0
14
15     while var_centroids and it in range(max_iterations):
16         # Asignar puntos al cluster mas cercano
17         clusters = [[] for _ in range(k)]
18         copia=[[] for _ in range(k)]
19         for point in data:
20             distances = [Chebysev_distance(point, centroid) for
21                           centroid in centroids]
22             cluster_index = np.argmin(distances)
23             clusters[cluster_index].append(point.tolist())

```

```

23
24     # Actualizar los centroides
25     for i in range(k):
26         if clusters[i]:
27             copia[i]=centroids[i].copy()
28             centroids[i] = np.mean(clusters[i], axis=0)
29
30     # Condicion de parada
31     var_centroids = any(
32         copia[i][0] != centroids[i][0] or copia[i][1] !=
33         centroids[i][1]
34         for i in range(len(copia))
35     )
36     it+=1
37     return clusters, centroids,it

```

Código A.4: K-Means para la distancia de Chebysev y n=2

A.1.3. Análisis para un dataset simple

```

1     import numpy as np
2     import matplotlib.pyplot as plt
3     import time
4
5     from Euclidean import *
6     from Manhattan import *
7     from Chebysev import *
8     from comprobar import *
9
10
11     ### Generamos nube de puntos simple
12
13     ## Definimos los parametros de las cinco distribuciones gaussianas
14     mean1 = [-2, -2] # Media de la primera distribucion gaussiana
15     cov1 = [[0.5, 0], [0, 0.5]] # Matriz de covarianza de la primera distribucion gaussiana
16
17     mean2 = [2.5, 3] # Media de la segunda distribucion gaussiana
18     cov2 = [[0.5, 0], [0, 0.5]] # Matriz de covarianza de la segunda distribucion gaussiana
19
20     mean3 = [0, 0] # Media de la tercera distribucion gaussiana
21     cov3 = [[1, 0], [0, 1]] # Matriz de covarianza de la tercera distribucion gaussiana
22

```



```

23 mean4 = [-3, 2.5] # Media de la cuarta distribucion gaussiana
24 cov4 = [[0.8, 0], [0, 0.8]] # Matriz de covarianza de la
    cuarta distribucion gaussiana
25
26 mean5 = [2, -3] # Media de la quinta distribucion gaussiana
27 cov5 = [[0.4, 0], [0, 0.4]] # Matriz de covarianza de la
    quinta distribucion gaussiana
28
29 ## Generamos los datos sinteticos para cada distribucion
    gaussiana
30 np.random.seed(0)
31 data1 = np.random.multivariate_normal(mean1,cov1, 100)
32 np.random.seed(1)
33 data2 = np.random.multivariate_normal(mean2,cov2, 100)
34 np.random.seed(2)
35 data3 = np.random.multivariate_normal(mean3,cov3, 100)
36 np.random.seed(3)
37 data4 = np.random.multivariate_normal(mean4,cov4, 100)
38 np.random.seed(4)
39 data5 = np.random.multivariate_normal(mean5,cov5, 100)
40
41 ## Representacion grafica de las nubes
42 plt.scatter(data1[:, 0], data1[:, 1], c='red', label='Nube 1')
43 plt.scatter(data2[:, 0], data2[:, 1], c='blue', label='Nube 2')
44 plt.scatter(data3[:, 0], data3[:, 1], c='green', label='Nube 3'
    )
45 plt.scatter(data4[:, 0], data4[:, 1], c='orange', label='Nube 4
    ')
46 plt.scatter(data5[:, 0], data5[:, 1], c='purple', label='Nube 5
    ')
47 plt.xlabel('X')
48 plt.ylabel('Y')
49 plt.title('Datos sinteticos con cinco nubes de puntos
    diferenciadas')
50 plt.legend()
51 plt.show()
52
53
54 ## Concatenamos los datos en un unico vector
55 total = np.concatenate((data1,data2,data3,data4,data5), axis=0)
56
57 ## K-Means con Euclidean
58 start_time = time.time()
59 clusters_eu, centroids_eu, it_eu = kmeans_euclidean(total,5,
    max_iterations=1000)
60 end_time = time.time()
61 execution_time = end_time - start_time

```

```

62 print(it_eu, execution_time)
63
64 ## Comprobacion de la distribucion de los puntos en cada
   cluster
65 print(len(total))
66 print(len(clusters_eu[0]), len(clusters_eu[1]), len(clusters_eu
   [2]), len(clusters_eu[3]), len(clusters_eu[4]))
67 comprobar(data1, clusters_eu)
68 comprobar(data2, clusters_eu)
69 comprobar(data3, clusters_eu)
70 comprobar(data4, clusters_eu)
71 comprobar(data5, clusters_eu)
72
73 ## Representacion grafica en subplots
74 fig, axs = plt.subplots(1, 3, figsize=(15, 5))
75
76 ## Dibujar puntos de cada cluster con colores diferentes en el
   primer subplot
77 axs[0].set_title("Dataset simple")
78 axs[0].scatter(total[:, 0], total[:, 1], s=50, c="black", label="
   Puntos")
79
80 ## Nubes de puntos
81 axs[1].scatter(data1[:, 0], data1[:, 1], c='red', label='Nube 1
   ')
82 axs[1].scatter(data2[:, 0], data2[:, 1], c='blue', label='Nube
   2')
83 axs[1].scatter(data3[:, 0], data3[:, 1], c='green', label='Nube
   3')
84 axs[1].scatter(data4[:, 0], data4[:, 1], c='orange', label='
   Nube 4')
85 axs[1].scatter(data5[:, 0], data5[:, 1], c='purple', label='
   Nube 5')
86 axs[1].set_title('Nubes de puntos')
87
88 ## Dibujar puntos de cada cluster con colores diferentes en el
   segundo subplot
89 axs[2].set_title("Clusters Euclidean")
90 for i, cluster in enumerate(clusters_eu):
91     cluster_points_eu = np.array(cluster)
92     axs[2].scatter(cluster_points_eu[:, 0], cluster_points_eu
  [:, 1], label=f"Cluster {i+1}")
93
94 ## Dibujar centroides con marcadores especiales y colores
   diferentes en el segundo subplot
95 axs[2].scatter(centroids_eu[:, 0], centroids_eu[:, 1], marker="
   X", s=100, c="black", label="Centroides")

```

```

96
97 ## Configuracion adicional de los subplots
98 for ax in axs:
99     ax.legend()
100     ax.set_xlabel("X")
101     ax.set_ylabel("Y")
102
103 plt.tight_layout()
104 plt.show()
105
106 ### K-Means con Manhattan
107 start_time = time.time()
108 clusters_mn, centroids_mn, it_mn = kmeans_manhattan(total,5)
109 end_time = time.time()
110 execution_time = end_time - start_time
111 print(it_mn, execution_time)
112
113 ## Comprobacion de la distribucion de los puntos en cada
cluster
114 print(len(total))
115 print(len(clusters_mn[0]), len(clusters_mn[1]), len(clusters_mn
    [2]), len(clusters_mn[3]), len(clusters_mn[4]))
116 comprobar(data1,clusters_mn)
117 comprobar(data2,clusters_mn)
118 comprobar(data3,clusters_mn)
119 comprobar(data4,clusters_mn)
120 comprobar(data5,clusters_mn)
121
122 ## Representacion grafica en subplots
123 fig, axs = plt.subplots(1, 3, figsize=(15, 5))
124
125 ## Dibujar puntos de cada cluster con colores diferentes en el
primer subplot
126 axs[0].set_title("Dataset simple")
127 axs[0].scatter(total[:,0], total[:,1], s=50, c="black", label="
    Puntos")
128
129 ## Nubes de puntos
130 axs[1].scatter(data1[:, 0], data1[:, 1], c='red', label='Nube 1
    ')
131 axs[1].scatter(data2[:, 0], data2[:, 1], c='blue', label='Nube
    2')
132 axs[1].scatter(data3[:, 0], data3[:, 1], c='green', label='Nube
    3')
133 axs[1].scatter(data4[:, 0], data4[:, 1], c='orange', label='
    Nube 4')

```

```

134 ax[1].scatter(data5[:, 0], data5[:, 1], c='purple', label='
    Nube 5')
135 ax[1].set_title('Nubes de puntos')
136
137 ## Dibujar puntos de cada cluster con colores diferentes en el
    segundo subplot
138 ax[2].set_title("Clusters Manhattan")
139 for i, cluster in enumerate(clusters_mn):
140     cluster_points_mn = np.array(cluster)
141     ax[2].scatter(cluster_points_mn[:, 0], cluster_points_mn
       [:, 1], label=f"Cluster {i+1}")
142
143 ## Dibujar centroides con marcadores especiales y colores
    diferentes en el segundo subplot
144 ax[2].scatter(centroids_mn[:, 0], centroids_mn[:, 1], marker="
    X", s=100, c="black", label="Centroides")
145
146 ## Configuración adicional de los subplots
147 for ax in axs:
148     ax.legend()
149     ax.set_xlabel("X")
150     ax.set_ylabel("Y")
151
152 plt.tight_layout()
153 plt.show()
154
155 ### K-Means con Chebysev
156 start_time = time.time()
157 clusters_ch, centroids_ch, it_ch = kmeans_Chebysev(total, 5)
158 end_time = time.time()
159 execution_time = end_time - start_time
160 print(it_ch, execution_time)
161
162 ## Comprobación de la distribución de los puntos en cada
    cluster
163 print(len(total))
164 print(len(clusters_ch[0]), len(clusters_ch[1]), len(clusters_ch
    [2]), len(clusters_ch[3]), len(clusters_ch[4]))
165 comprobar(data1, clusters_ch)
166 comprobar(data2, clusters_ch)
167 comprobar(data3, clusters_ch)
168 comprobar(data4, clusters_ch)
169 comprobar(data5, clusters_ch)
170
171 ## Representación gráfica en subplots
172 fig, ax = plt.subplots(1, 3, figsize=(15, 5))
173

```

```

174 ## Dibujar puntos de cada cluster con colores diferentes en el
175 primer subplot
176 axs[0].set_title("Dataset simple")
177 axs[0].scatter(total[:,0], total[:,1], s=50, c="black", label="
178 Puntos")
179 ## Nubes de puntos
180 axs[1].scatter(data1[:, 0], data1[:, 1], c='red', label='Nube 1
181 ')
182 axs[1].scatter(data2[:, 0], data2[:, 1], c='blue', label='Nube
183 2')
184 axs[1].scatter(data3[:, 0], data3[:, 1], c='green', label='Nube
185 3')
186 axs[1].scatter(data4[:, 0], data4[:, 1], c='orange', label='
187 Nube 4')
188 axs[1].scatter(data5[:, 0], data5[:, 1], c='purple', label='
189 Nube 5')
190 axs[1].set_title('Nubes de puntos')
191 ## Dibujar puntos de cada cluster con colores diferentes en el
192 segundo subplot
193 axs[2].set_title("Clusters Chebysev")
194 for i, cluster in enumerate(clusters_ch):
195     cluster_points_ch = np.array(cluster)
196     axs[2].scatter(cluster_points_ch[:, 0], cluster_points_ch
197        [:, 1], label=f"Cluster {i+1}")
198 ## Dibujar centroides con marcadores especiales y colores
199 diferentes en el segundo subplot
200 axs[2].scatter(centroids_ch[:, 0], centroids_ch[:, 1], marker="
201 X", s=100, c="black", label="Centroides")
202 ## Configuración adicional de los subplots
203 for ax in axs:
204     ax.legend()
205     ax.set_xlabel("X")
206     ax.set_ylabel("Y")
207 plt.tight_layout()
208 plt.show()
209 ### Representación gráfica en subplots todas juntas
210 fig, axs = plt.subplots(3, 2, figsize=(15, 15))
211 ## Dibujar puntos de cada cluster con colores diferentes en el
212 primer subplot

```

```

209 ax[0][0].set_title("Dataset_1")
210 ax[0][0].scatter(total[:,0], total[:,1], s=50, c="black",
    label="Puntos")
211
212 ## Nubes de puntos
213 ax[0][1].scatter(data1[:, 0], data1[:, 1], c='red', label='
    Nube 1')
214 ax[0][1].scatter(data2[:, 0], data2[:, 1], c='blue', label='
    Nube 2')
215 ax[0][1].scatter(data3[:, 0], data3[:, 1], c='green', label='
    Nube 3')
216 ax[0][1].scatter(data4[:, 0], data4[:, 1], c='orange', label='
    Nube 4')
217 ax[0][1].scatter(data5[:, 0], data5[:, 1], c='purple', label='
    Nube 5')
218 ax[0][1].set_title('Nubes de puntos')
219
220 ax[1][0].set_title("Clusters Euclidean")
221 for i, cluster in enumerate(clusters_eu):
222     cluster_points_eu = np.array(cluster)
223     ax[1][0].scatter(cluster_points_eu[:, 0],
        cluster_points_eu[:, 1], label=f"Cluster {i+1}")
224 ## Dibujar centroides con marcadores especiales y colores
    diferentes en el segundo subplot
225 ax[1][0].scatter(centroids_eu[:, 0], centroids_eu[:, 1],
    marker="X", s=70, c="black", label="Centroides")
226
227
228 ax[1][1].set_title("Clusters Manhattan")
229 for i, cluster in enumerate(clusters_mn):
230     cluster_points_mn = np.array(cluster)
231     ax[1][1].scatter(cluster_points_mn[:, 0],
        cluster_points_mn[:, 1], label=f"Cluster {i+1}")
232 ## Dibujar centroides con marcadores especiales y colores
    diferentes en el segundo subplot
233 ax[1][1].scatter(centroids_mn[:, 0], centroids_mn[:, 1],
    marker="X", s=70, c="black", label="Centroides")
234
235
236 ax[2][1].set_title("Clusters Chebysev")
237 for i, cluster in enumerate(clusters_ch):
238     cluster_points_ch = np.array(cluster)
239     ax[2][1].scatter(cluster_points_ch[:, 0],
        cluster_points_ch[:, 1], label=f"Cluster {i+1}")
240 ## Dibujar centroides con marcadores especiales y colores
    diferentes en el segundo subplot

```

```

241 axes[2][1].scatter(centroids_ch[:, 0], centroids_ch[:, 1],
    marker="X", s=70, c="black", label="Centroides")
242
243 plt.tight_layout()
244 plt.show()

```

Código A.5: Análisis para un dataset simple

A.1.4. Análisis para un dataset con outliers

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import time
4
5 from Euclidean import *
6 from Manhattan import *
7 from Chebysev import *
8 from comprobar import *
9
10
11 ### Generamos nube de puntos simple
12
13 ## Definimos los parametros de las cinco distribuciones gaussianas
14 mean1 = [-3, -3] # Media de la primera distribucion gaussiana
15 cov1 = [[1.5, 0], [0, 1.5]] # Matriz de covarianza de la
    primera distribucion gaussiana
16
17 mean2 = [3, 3.5] # Media de la segunda distribucion gaussiana
18 cov2 = [[1.5, 0], [0, 1.5]] # Matriz de covarianza de la
    segunda distribucion gaussiana
19
20 mean3 = [0, 0] # Media de la tercera distribucion gaussiana
21 cov3 = [[2, 0], [0, 2]] # Matriz de covarianza de la tercera
    distribucion gaussiana
22
23 mean4 = [-3, 2.5] # Media de la cuarta distribucion gaussiana
24 cov4 = [[1.8, 0], [0, 1.8]] # Matriz de covarianza de la
    cuarta distribucion gaussiana
25
26 mean5 = [2.5, -3] # Media de la quinta distribucion gaussiana
27 cov5 = [[2.4, 0], [0, 2.4]] # Matriz de covarianza de la
    quinta distribucion gaussiana
28
29
30 # Generar los datos sinteticos para cada distribucion gaussiana

```

```

31 np.random.seed(0)
32 o_data1 = np.random.multivariate_normal(mean1, cov1, 100)
33 np.random.seed(1)
34 o_data2 = np.random.multivariate_normal(mean2, cov2, 100)
35 np.random.seed(2)
36 o_data3 = np.random.multivariate_normal(mean3, cov3, 100)
37 np.random.seed(3)
38 o_data4 = np.random.multivariate_normal(mean4, cov4, 100)
39 np.random.seed(4)
40 o_data5 = np.random.multivariate_normal(mean5, cov5, 100)
41
42 # Generar algunos datos con medidas atípicas
43 outliers1 = np.array([[6, 6], [-6, -9], [0,
44     1], [10, 0], [9, -5], [-8, -2]])
45 outliers2 = np.array([[6, 2], [-6, -1], [3,
46     1], [10, 4], [9, -4], [-6, -2]])
47 outliers3 = np.array([[6, -6], [-6, 4], [0,
48     -1], [10, -3], [4, -5], [3, -2]])
49 outliers4 = np.array([[6, 1], [-6, 7], [0,
50     14], [10, -10], [-9, -5], [8, -2]])
51 outliers5 = np.array([[6, -2], [-6, -7], [0,
52     -1], [10, 1], [9, 1], [-8, 2]])
53 # Concatenar los datos con medidas atípicas a los datos
54 anteriores
55 o_data1 = np.concatenate((o_data1, outliers1), axis=0)
56 o_data2 = np.concatenate((o_data2, outliers2), axis=0)
57 o_data3 = np.concatenate((o_data3, outliers3), axis=0)
58 o_data4 = np.concatenate((o_data4, outliers4), axis=0)
59 o_data5 = np.concatenate((o_data5, outliers5), axis=0)
60
61 ## Representacion grafica de las nubes
62 plt.scatter(o_data1[:, 0], o_data1[:, 1], c='red', label='Nube
63     1')
64 plt.scatter(o_data2[:, 0], o_data2[:, 1], c='blue', label='Nube
65     2')
66 plt.scatter(o_data3[:, 0], o_data3[:, 1], c='green', label='
67     Nube 3')
68 plt.scatter(o_data4[:, 0], o_data4[:, 1], c='orange', label='
69     Nube 4')
70 plt.scatter(o_data5[:, 0], o_data5[:, 1], c='purple', label='
71     Nube 5')
72 plt.xlabel('X')
73 plt.ylabel('Y')
74 plt.title('Datos sint ticos con outliers')
75 plt.legend()
76 plt.show()
77

```



```

67 ## Concatenamos los datos en un nico vector
68 o_total = np.concatenate((o_data1, o_data2, o_data3, o_data4,
69                             o_data5), axis=0)
70
71 ## K-Means con Euclidean
72 start_time = time.time()
73 clusters_eu, centroids_eu, it_eu = kmeans_euclidean(o_total,5,
74                                                     max_iterations=1000)
75 end_time = time.time()
76 execution_time = end_time - start_time
77 print(it_eu, execution_time)
78
79 ## Comprobacion de la distribucion de los puntos en cada
80 cluster
81 print(len(o_total))
82 print(len(clusters_eu[0]), len(clusters_eu[1]), len(clusters_eu
83 [2]), len(clusters_eu[3]), len(clusters_eu[4]))
84 comprobar(o_data1,clusters_eu)
85 comprobar(o_data2,clusters_eu)
86 comprobar(o_data3,clusters_eu)
87 comprobar(o_data4,clusters_eu)
88 comprobar(o_data5,clusters_eu)
89
90 # Representacion grafica en subplots
91 fig, axs = plt.subplots(1, 3, figsize=(15, 5))
92
93 ## Dibujar puntos de cada cluster con colores diferentes en el
94 primer subplot
95 axs[0].set_title("Dataset outliers")
96 axs[0].scatter(o_total[:,0],o_total[:,1], s=50, c="black",
97               label="Puntos")
98
99 ## Nubes de puntos
100 axs[1].scatter(o_data1[:, 0], o_data1[:, 1], c='red', label='
101 Nube 1')
102 axs[1].scatter(o_data2[:, 0], o_data2[:, 1], c='blue', label='
103 Nube 2')
104 axs[1].scatter(o_data3[:, 0], o_data3[:, 1], c='green', label='
105 Nube 3')
106 axs[1].scatter(o_data4[:, 0], o_data4[:, 1], c='orange', label=
107 'Nube 4')
108 axs[1].scatter(o_data5[:, 0], o_data5[:, 1], c='purple', label=
109 'Nube 5')
110 axs[1].set_title('Nubes de puntos')
111

```

```

102 ## Dibujar puntos de cada cluster con colores diferentes en el
    segundo subplot
103 axs[2].set_title("Clusters Euclidean")
104 for i, cluster in enumerate(clusters_eu):
105     cluster_points_eu = np.array(cluster)
106     axs[2].scatter(cluster_points_eu[:, 0], cluster_points_eu
        [:, 1], label=f"Cluster {i+1}")
107
108 ## Dibujar centroides con marcadores especiales y colores
    diferentes en el segundo subplot
109 axs[2].scatter(centroids_eu[:, 0], centroids_eu[:, 1], marker="
    X", s=100, c="black", label="Centroides")
110
111 ## Configuración adicional de los subplots
112 for ax in axs:
113     ax.legend()
114     ax.set_xlabel("X")
115     ax.set_ylabel("Y")
116
117 plt.tight_layout()
118 plt.show()
119
120 ### K-Means con Manhattan
121 start_time = time.time()
122 clusters_mn, centroids_mn, it_mn = kmeans_manhattan(o_total,5)
123 end_time = time.time()
124 execution_time = end_time - start_time
125 print(it_mn, execution_time)
126
127 ## Comprobación de la distribución de los puntos en cada
    cluster
128 print(len(o_total))
129 print(len(clusters_mn[0]), len(clusters_mn[1]), len(clusters_mn
    [2]), len(clusters_mn[3]), len(clusters_mn[4]))
130 comprobar(o_data1,clusters_mn)
131 comprobar(o_data2,clusters_mn)
132 comprobar(o_data3,clusters_mn)
133 comprobar(o_data4,clusters_mn)
134 comprobar(o_data5,clusters_mn)
135
136 ## Representación gráfica en subplots
137 fig, axs = plt.subplots(1, 3, figsize=(15, 5))
138
139 ## Dibujar puntos de cada cluster con colores diferentes en el
    primer subplot
140 axs[0].set_title("Dataset outliers")

```

```

141  axs[0].scatter(o_total[:,0], o_total[:,1], s=50, c="black",
142                label="Puntos")
143  ## Nubes de puntos
144  axs[1].scatter(o_data1[:, 0], o_data1[:, 1], c='red', label='
145                Nube 1')
146  axs[1].scatter(o_data2[:, 0], o_data2[:, 1], c='blue', label='
147                Nube 2')
148  axs[1].scatter(o_data3[:, 0], o_data3[:, 1], c='green', label='
149                Nube 3')
150  axs[1].scatter(o_data4[:, 0], o_data4[:, 1], c='orange', label='
151                Nube 4')
152  axs[1].scatter(o_data5[:, 0], o_data5[:, 1], c='purple', label='
153                Nube 5')
154  axs[1].set_title('Nubes de puntos')
155  ## Dibujar puntos de cada cluster con colores diferentes en el
156                segundo subplot
157  axs[2].set_title("Clusters Manhattan")
158  for i, cluster in enumerate(clusters_mn):
159      cluster_points_mn = np.array(cluster)
160      axs[2].scatter(cluster_points_mn[:, 0], cluster_points_mn
161                   [:, 1], label=f"Cluster {i+1}")
162  ## Dibujar centroides con marcadores especiales y colores
163                diferentes en el segundo subplot
164  axs[2].scatter(centroids_mn[:, 0], centroids_mn[:, 1], marker="
165                X", s=100, c="black", label="Centroides")
166  ## Configuración adicional de los subplots
167  for ax in axs:
168      ax.legend()
169      ax.set_xlabel("X")
170      ax.set_ylabel("Y")
171  plt.tight_layout()
172  plt.show()
173  ### K-Means con Chebysev
174  start_time = time.time()
175  clusters_ch, centroids_ch, it_ch = kmeans_Chebysev(o_total,5)
176  end_time = time.time()
177  execution_time = end_time - start_time
178  print(it_ch, execution_time)
179  ## Comprobación de la distribución de los puntos en cada
180                cluster

```

```

177 print(len(o_total))
178 print(len(clusters_ch[0]), len(clusters_ch[1]), len(clusters_ch
    [2]), len(clusters_ch[3]), len(clusters_ch[4]))
179 comprobar(o_data1, clusters_ch)
180 comprobar(o_data2, clusters_ch)
181 comprobar(o_data3, clusters_ch)
182 comprobar(o_data4, clusters_ch)
183 comprobar(o_data5, clusters_ch)
184
185 ## Representacion grafica en subplots
186 fig, axs = plt.subplots(1, 3, figsize=(15, 5))
187
188 ## Dibujar puntos de cada cluster con colores diferentes en el
    primer subplot
189 axs[0].set_title("Dataset outliers")
190 axs[0].scatter(o_total[:,0], o_total[:,1], s=50, c="black",
    label="Puntos")
191
192 ## Nubes de puntos
193 axs[1].scatter(o_data1[:, 0], o_data1[:, 1], c='red', label='
    Nube 1')
194 axs[1].scatter(o_data2[:, 0], o_data2[:, 1], c='blue', label='
    Nube 2')
195 axs[1].scatter(o_data3[:, 0], o_data3[:, 1], c='green', label='
    Nube 3')
196 axs[1].scatter(o_data4[:, 0], o_data4[:, 1], c='orange', label=
    'Nube 4')
197 axs[1].scatter(o_data5[:, 0], o_data5[:, 1], c='purple', label=
    'Nube 5')
198 axs[1].set_title('Nubes de puntos')
199
200 ## Dibujar puntos de cada cluster con colores diferentes en el
    segundo subplot
201 axs[2].set_title("Clusters Chebysev")
202 for i, cluster in enumerate(clusters_ch):
203     cluster_points_ch = np.array(cluster)
204     axs[2].scatter(cluster_points_ch[:, 0], cluster_points_ch
       [:, 1], label=f"Cluster {i+1}")
205
206 ## Dibujar centroides con marcadores especiales y colores
    diferentes en el segundo subplot
207 axs[2].scatter(centroids_ch[:, 0], centroids_ch[:, 1], marker="
    X", s=100, c="black", label="Centroides")
208
209 ## Configuracion adicional de los subplots
210 for ax in axs:
211     ax.legend()

```

```

212     ax.set_xlabel("X")
213     ax.set_ylabel("Y")
214
215 plt.tight_layout()
216 plt.show()
217
218
219
220 ### Representacion grafica en subplots todas juntas
221 fig, axs = plt.subplots(3, 2, figsize=(15, 15))
222
223 ## Dibujar puntos de cada cluster con colores diferentes en el
primer subplot
224 axs[0][0].set_title("Dataset outliers")
225 axs[0][0].scatter(o_total[:,0], o_total[:,1], s=50, c="black",
    label="Puntos")
226
227 ## Nubes de puntos
228 axs[0][1].scatter(o_data1[:, 0], o_data1[:, 1], c='red', label=
    'Nube 1')
229 axs[0][1].scatter(o_data2[:, 0], o_data2[:, 1], c='blue', label
    ='Nube 2')
230 axs[0][1].scatter(o_data3[:, 0], o_data3[:, 1], c='green',
    label='Nube 3')
231 axs[0][1].scatter(o_data4[:, 0], o_data4[:, 1], c='orange',
    label='Nube 4')
232 axs[0][1].scatter(o_data5[:, 0], o_data5[:, 1], c='purple',
    label='Nube 5')
233 axs[0][1].set_title('Nubes de puntos')
234
235 axs[1][0].set_title("Clusters Euclidean")
236 for i, cluster in enumerate(clusters_eu):
237     cluster_points_eu = np.array(cluster)
238     axs[1][0].scatter(cluster_points_eu[:, 0],
        cluster_points_eu[:, 1], label=f"Cluster {i+1}")
239 ## Dibujar centroides con marcadores especiales y colores
diferentes en el segundo subplot
240 axs[1][0].scatter(centroids_eu[:, 0], centroids_eu[:, 1],
    marker="X", s=70, c="black", label="Centroides")
241
242
243 axs[1][1].set_title("Clusters Manhattan")
244 for i, cluster in enumerate(clusters_mn):
245     cluster_points_mn = np.array(cluster)
246     axs[1][1].scatter(cluster_points_mn[:, 0],
        cluster_points_mn[:, 1], label=f"Cluster {i+1}")

```

```

247 ## Dibujar centroides con marcadores especiales y colores
    diferentes en el segundo subplot
248 axs[1][1].scatter(centroids_mn[:, 0], centroids_mn[:, 1],
    marker="X", s=70, c="black", label="Centroides")
249
250
251 axs[2][1].set_title("Clusters Chebysev")
252 for i, cluster in enumerate(clusters_ch):
253     cluster_points_ch = np.array(cluster)
254     axs[2][1].scatter(cluster_points_ch[:, 0],
    cluster_points_ch[:, 1], label=f"Cluster {i+1}")
255 ## Dibujar centroides con marcadores especiales y colores
    diferentes en el segundo subplot
256 axs[2][1].scatter(centroids_ch[:, 0], centroids_ch[:, 1],
    marker="X", s=70, c="black", label="Centroides")
257
258 plt.tight_layout()
259 plt.show()

```

Código A.6: Análisis para un dataset con outliers

A.1.5. Algoritmos de K-Means para dimensión n

```

1
2 ### Algoritmo de K-Means para la distancia Euclidea n
    dimensional
3 import pandas as pd
4 import numpy as np
5
6 from distances import euclidean_n_distance
7
8 def kmeans_euclidean_n(data, k, max_iterations=100):
9     # Inicializacion de los centroides
10     data_df=pd.DataFrame(data)
11     centroids = data_df.sample(k)
12     centroids=centroids.to_numpy()
13     var_centroids=True
14     it=0
15
16     while var_centroids and it in range(max_iterations):
17         # Asignar puntos al cluster mas cercano
18         clusters = [[] for _ in range(k)]
19         copia=[[] for _ in range(k)]
20         for point in data:
21             distances = [euclidean_n_distance(point, centroid)
                for centroid in centroids]

```

```

22         cluster_index = np.argmin(distances)
23         clusters[cluster_index].append(point)
24
25     # Actualizar los centroides
26     for i in range(k):
27         if clusters[i]:
28             copia[i]=centroids[i].copy()
29             centroids[i] = np.mean(clusters[i], axis=0)
30
31     # Condicion de parada
32     var_centroids = any(
33         copia[i][0] != centroids[i][0] or copia[i][1] !=
34         centroids[i][1]
35         for i in range(len(copia))
36     )
37     it+=1
38     return clusters, centroids, it

```

Código A.7: K-Means para la distancia euclídea y dimensión n

```

1
2 ### Algoritmo de K-Means para la distancia Manhattan n
3 dimensional
4 import pandas as pd
5 import numpy as np
6
7 from distances import manhattan_n_distance
8
9 def kmeans_manhattan_n(data, k, max_iterations=100):
10     # Inicializacion de los centroides
11     data_df=pd.DataFrame(data)
12     centroids = data_df.sample(k)
13     centroids=centroids.to_numpy()
14     var_centroids=True
15     it=0
16
17     while var_centroids and it in range(max_iterations):
18         # Asignar puntos al cluster mas cercano
19         clusters = [[] for _ in range(k)]
20         copia=[[] for _ in range(k)]
21         for point in data:
22             distances = [manhattan_n_distance(point, centroid)
23                         for centroid in centroids]
24             cluster_index = np.argmin(distances)
25             clusters[cluster_index].append(point)
26
27         # Actualizar los centroides
28         for i in range(k):

```

```

27         if clusters[i]:
28             copia[i]=centroids[i].copy()
29             centroids[i] = np.mean(clusters[i], axis=0)
30
31         #Condicion de parada
32         var_centroids = any(
33             copia[i][0] != centroids[i][0] or copia[i][1] !=
34             centroids[i][1]
35             for i in range(len(copia))
36         )
37         it+=1
38     return clusters, centroids,it

```

Código A.8: K-Means para la distancia de Manhattan y dimensión n

```

1  ### Algoritmo de K-Means para la distancia Chebysev n
2  dimensional
3  import pandas as pd
4  import numpy as np
5  from distances import chebysev_n_distance
6
7  def kmeans_chebysev_n(data, k, max_iterations=100):
8      # Inicializacion de los centroides
9      data_df=pd.DataFrame(data)
10     centroids = data_df.sample(k)
11     centroids=centroids.to_numpy()
12     var_centroids=True
13     it=0
14
15     while var_centroids and it in range(max_iterations):
16         # Asignar puntos al cluster mas cercano
17         clusters = [[] for _ in range(k)]
18         copia=[[] for _ in range(k)]
19         for point in data:
20             distances = [chebysev_n_distance(point, centroid)
21                           for centroid in centroids]
22             cluster_index = np.argmin(distances)
23             clusters[cluster_index].append(point)
24
25         # Actualizar los centroides
26         for i in range(k):
27             if clusters[i]:
28                 copia[i]=centroids[i].copy()
29                 centroids[i] = np.mean(clusters[i], axis=0)
30
31         # Condicion de parada
32         var_centroids = any(

```



```

32         copia[i][0] != centroids[i][0] or copia[i][1] !=
           centroids[i][1]
33         for i in range(len(copia))
34     )
35     it+=1
36     return clusters, centroids,it

```

Código A.9: K-Means para la distancia de Chebysev y dimensión n

A.1.6. Análisis para el MNIST

```

1
2 import pandas as pd
3 import time
4
5 from Euclidean_n import *
6 from Manhattan_n import *
7 from Chebysev_n import *
8 from comprobar_n import *
9
10
11 ### Indicamos la ruta de la base de datos en formato .csv
12 ruta='C:/Users/marco/Documents/Universidad/TFG/mnist/
    mnist_train 1.csv'
13
14
15 ### Cargamos la base de datos y la convertimos al formato
deseado
16 df=pd.read_csv(ruta, header=0)
17 print(df['label'].value_counts())
18 print('Base de datos leida')
19 print(df.shape)
20 print(df.head)
21 lista=df.values.tolist()
22 print('MNIST lista')
23
24
25 ### Realizamos el analisis para la distancia euclidea
26 start_time = time.time()
27 clusters, centroids, it = kmeans_euclidean_n(lista,10)
28 end_time = time.time()
29 print('KMeans euclidean ready')
30 execution_time = end_time - start_time
31 print(it, execution_time)
32
33 print('KMeans euclidean proof')

```

```
34 comprobar_n(clusters)
35
36
37 ### Realizamos el analisis para la distancia de Manhattan
38 start_time = time.time()
39 clusters_mn, centroids_mn, it_mn = kmeans_manhattan_n(lista,10)
40 end_time = time.time()
41 print('KMeans Manhattan ready')
42 execution_time = end_time - start_time
43 print(it, execution_time)
44
45 print('KMeans Manhattan proof')
46 comprobar_n(clusters_mn)
47
48
49 ### Realizamos el analisis para la distancia de Chebysev
50 start_time = time.time()
51 clusters_ch, centroids_ch, it_ch = kmeans_chebysev_n(lista,10)
52 end_time = time.time()
53 print('KMeans Chebysev ready')
54 execution_time = end_time - start_time
55 print(it, execution_time)
56
57 print('KMeans Chebysev proof')
58 comprobar_n(clusters_ch)
```

Código A.10: Análisis para MNIST