UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Sistemas Informáticos y Computación

Análisis de la robustez de los algoritmos de clustering

Trabajo Fin de Máster

Máster Universitario en Ingeniería y Tecnología de Sistemas Software

AUTOR/A: Diaz Ccasa, Naysha Naydu

Tutor/a: Ramírez Quintana, María José

CURSO ACADÉMICO: 2024/2025

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Universitat Politècnica de València

Departamento de Sistemas Informáticos y Computación

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Máster Universitario en Ingeniería y Tecnología de Sistemas Software

Trabajo Fin de Máster

Análisis de la robustez de los algoritmos de clustering

Autor(a): Naysha Naydu Diaz Ccasa
Director(a): María José Ramírez Quintana

Valencia, 09 - 2025

Este Trabajo Fin de Máster se ha depositado en el Departamento de Sistemas Informàticos y Computación de la Universitat Politècnica de València para su defensa.

Trabajo Fin de Máster
Máster Universitario en Ingeniería y Tecnología de Sistemas Software

Título: Análisis de la robustez de los algoritmos de clustering

09 - 2025

Autor(a): Naysha Naydu Diaz Ccasa
Director(a): María José Ramírez Quintana
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Resumen

El *clustering* o agrupamiento es una técnica esencial dentro del aprendizaje no supervisado, utilizada para identificar estructuras dentro de los datos sin necesidad de etiquetas como en el aprendizaje supervisado. Sin embargo, los algoritmos de *clustering* son sensibles a perturbaciones y errores presentes en los datos, lo que plantea la necesidad de evaluar su robustez. En este Trabajo Fin de Máster se realiza un estudio experimental de la estabilidad de dos algoritmos representativos que son el `k-means` y *clustering* jerárquico, frente a la introducción de ruido gaussiano en un rango de 0.2 a 1.4 y aplicado a conjuntos de datos heterogéneos.

Para verificar la estabilidad de los algoritmos se diseñó un flujo metodológico que incluye la selección y preprocesamiento de *datasets*, la estandarización de atributos numéricos, la inyección de perturbaciones controladas y la aplicación de métricas de evaluación de robustez: *Adjusted Rand Index* (ARI), *Normalized Mutual Information* (NMI) y la Proporción de Pares Dañados (PDP). Los experimentos permiten comparar el comportamiento de ambos algoritmos en escenarios con diferente dimensionalidad y complejidad, esto gracias a la variedad de *datasets* que se tiene para el estudio.

Los resultados evidencian que la robustez depende tanto del algoritmo como de las características intrínsecas de los datos, donde `k-means` presenta una degradación más suave y estable en comparación con el *clustering* jerárquico, que muestra pequeñas perturbaciones especialmente en *datasets* con estructuras menos definidas. Asimismo, se observan comportamientos no monótonos, donde niveles intermedios de ruido pueden alterar la coherencia de los *clusters* de forma inesperada.

Este trabajo contribuye a la comprensión de la estabilidad en *clustering* y ofrece un marco experimental replicable para evaluar la robustez de algoritmos no supervisados frente a perturbaciones, con aplicaciones en dominios donde los datos suelen estar afectados por ruido o errores de medición.

Resum

El *clustering* o agrupament és una tècnica essencial dins de l'aprenentatge no supervisat, utilitzada per a identificar estructures dins de les dades sense necessitat d'etiquetes com en l'aprenentatge supervisat. No obstant això, els algorismes de *clustering* són sensibles a pertorbacions i errors presents en les dades, la qual cosa planteja la necessitat d'avaluar la seua robustesa. En este Treball fi de màster es realitza un estudi experimental de l'estabilitat de dos algorismes representatius que són el `k-means` i *clustering* jeràrquic, enfront de la introducció de soroll gaussià en un rang de 0.2 a 1.4 i aplicat a conjunts de dades heterogènies.

Per a verificar l'estabilitat dels algorismes usen *clustering* es va dissenyar un flux metodològic que inclou la selecció i preprocessament de *datasets*, l'estandardització d'atributs numèrics, la injecció de pertorbacions controlades i l'aplicació de mètriques d'avaluació de robustesa: *Adjusted Rand Index* (ARI), *Normalized Mutual Information* (NMI) i la Proporció de Parells Danyats (PDP). Els experiments permeten comparar el comportament dels dos algorismes en escenaris amb diferent dimensionalitat i complexitat, això gràcies a la varietat de *datasets* que es té per a l'estudi.

Els resultats evidencien que la robustesa depén tant de l'algorisme com de les característiques intrínseques de les dades, mentres que `k-means` presenta una degradació més suau i estable a comparació amb el *clustering* jeràrquic, que mostra xicotetes pertorbacions especialment en *datasets* amb estructures menys definides. Així mateix, s'observen comportaments no monòtons, on nivells intermedis de soroll poden alterar la coherència dels clústers de manera inesperada.

Este treball contribuïx a la comprensió de l'estabilitat en *clustering* i oferix un marc experimental reproduïble per a avaluar la robustesa d'algorismes no supervisats enfront de pertorbacions, amb aplicacions en dominis on les dades solen estar afectats per soroll o errors de mesurament.

Abstract

Clustering is an essential technique within unsupervised learning, used to identify structures in data without the need for labels as in supervised learning. However, clustering algorithms are sensitive to perturbations and errors present in the data, which raises the need to assess their robustness. In this Master's Thesis, an experimental study is conducted on the stability of two representative algorithms—`k-means` and hierarchical clustering—when subjected to Gaussian noise in a range from 0.2 to 1.4, applied to heterogeneous datasets.

To verify the stability of the algorithms, a methodological workflow was designed that includes dataset selection and preprocessing, standardization of numerical attributes, injection of controlled perturbations, and the application of robustness evaluation metrics: *Adjusted Rand Index* (ARI), *Normalized Mutual Information* (NMI), and the Proportion of Damaged Pairs (PDP). The experiments allow for a comparison of the behavior of both algorithms in scenarios with different dimensionality and complexity, thanks to the variety of datasets included in the study.

The results show that robustness depends both on the algorithm and on the intrinsic characteristics of the data. `k-means` exhibits a smoother and more stable degradation compared to hierarchical clustering, which shows small perturbations, particularly in datasets with less defined structures. Non-monotonic behaviors are also observed, where intermediate levels of noise can unexpectedly alter cluster coherence.

This work contributes to the understanding of clustering stability and provides a replicable experimental framework for evaluating the robustness of unsupervised algorithms under perturbations, with applications in domains where data are often affected by noise or measurement errors.

Tabla de contenidos

Capítulo 1

Introducción

La agrupación o *clustering* es una técnica del aprendizaje no supervisado que tiene como propósito identificar agrupaciones o estructuras con información similar dentro de los *datasets*. A diferencia de su contraparte, el aprendizaje supervisado, esta técnica no depende de etiquetas de salidas en los datos, necesarias para comprobar que se realizó una clasificación correcta, sino que forma agrupaciones por similitud de los datos. La relevancia del uso de *clustering* radica en que permite organizar, resumir y explorar datos complejos con el objetivo de encontrar patrones que se pasarían por alto en un simple análisis de datos. Dentro del campo de acción se usa en distintas áreas, desde la biología computacional y el análisis de textos hasta la segmentación de clientes en un comercio y la visión por computador.

La calidad de los resultados al aplicar *clustering* se pueden evaluar usando medidas que analizan la compacidad de los *clusters*, es decir los puntos que pertenecen a un mismo *cluster* deben estar lo más cerca posible, por ejemplo un *cluster* de personas que compran productos saludables debe agrupar a clientes que tengan los mismos patrones compra, otra medida es la separación entre *clusters*, que indica si los *clusters* están bien separados unos de otros, por ejemplos si existe un *cluster* de jóvenes compradores online y otro de adultos compradores presenciales ambos grupos deben estar bien separados para que no haya confusión; también puede usarse una medida de validación externa, si se tiene una agrupación de datos correcta se puede evaluar cuanto coincide el *clustering* con la clasificación de referencia. Sin embargo la calidad de resultados no es suficiente porque en contextos reales los datos nunca están libres de errores, entonces si los datos cambien ligeramente, cuanto podría verse afectada la creación de estos grupos cuando se aplique el *clustering*, ya que los agrupamientos dependen de la información de los datos y si estos varían las estructuras haciendo que se rompan o alterando los grupos que se desean descubrir, entonces que técnicas de *clustering* garantizan tener baja sensibilidad a perturbaciones de datos. En este sentido cobra importancia el concepto de robustez en *clustering* que se entiende como la capacidad de mantener coherencia en las agrupaciones frente a perturbaciones, un algoritmo robusto no solo forma *clusters* definidos, sino que se mantiene estable cuando se introducen variaciones controladas dentro de los datos. Otras investigaciones señalan que la robustez es un criterio complementario a la calidad y que resulta esencial en aplicaciones que trabajan en entornos reales donde es común que los datos presenten cierta cantidad de ruido, a pesar de ello en comparación a la gran cantidad de literatura sobre validación de *clusters* los estudios

experimentales de robustez en algoritmos no supervisados son menos frecuentes, esto motiva la necesidad de diseñar un marco metodológico para evaluarla estabilidad de los algoritmos representativos cuando interactúan con datos perturbados por ruido. En este trabajo se aborda este problema a través de un estudio experimental donde se usan dos algoritmos ampliamente conocidos, `k-means` y *clustering* jerárquico, ambos tienen diferentes enfoques para realizar las agrupaciones, pero ambos devuelven una partición del conjunto de datos agrupados en *clusters*. Para introducir ruido se emplea una distribución gaussiana que se aplica a los datos numéricos en cada conjunto de datos y para evaluar la robustes se usan tres métricas *Adjusted Rand Index* (ARI), la Normalized Mutual Information (NMI) y la Proporción de Pares Dañados (PDP) que se irán definiendo más adelante. De esta forma, el trabajo no solo contribuye a entender el comportamiento de algoritmos de *clustering* expuestos en entornos más reales o adversos, sino que ofrece un marco replicable para evaluar la estabilidad en diferentes contextos de conjuntos de datos.

1.1. Motivación

En entornos reales las aplicaciones tienen datos con errores aleatorios, distorsiones o instancias que actúan como ruido en la data por ser poco comunes. En este contexto es necesario analizar que tan estables son los algoritmos de *clustering* frente a las perturbaciones, entonces la robustez es un complemento crítico para medir la calidad de los *clusters* y de esta manera seleccionar algoritmos que no solo agrupen los datos coherentemente, sino que se mantengan estables a pesar de que se introduzcan ruido o se encuentren en ambientes adversos. En este trabajo surge la necesidad de evaluar la estabilidad de dos algoritmos de *clustering* para lo que se diseñó un experimento replicable de esta manera se pueda entender mejor las limitaciones y fortalezas de estos algoritmos, `k-means` y *clustering* jerárquico.

1.2. Objetivos

1.2.1. Objetivos generales

Realizar un estudio experimental usando algoritmos de *clustering* para analizar la robustez cuando se introduce ruido gaussiano en distintos conjuntos de datos heterogéneos.

1.2.2. Objetivos específicos

- Revisión bibliográfica sobre la robustez, modelos de *clustering*, ruidos o perturbaciones y sus tipologías.
- Seleccionar conjuntos de datos representativos que sean adecuados para aplicar las técnicas *clustering* así como para incluir en ellos perturbaciones.
- Limpiar y preprocesar adecuadamente los conjuntos de datos para poder usarlos con el método de *clustering*.
- Definir un procedimiento para introducir distintos niveles de ruido gaussiano en los datos.

- Realizar experimentos para evaluar la robustez de los algoritmos de *clustering* usando métricas de evaluación basadas en la comparación de agrupamientos.
- Analizar los resultados para identificar patrones de estabilidad, fragilidad o posibles comportamientos atípicos.

Capítulo 2

Antecedentes y Trabajos Relacionados

Dentro del contexto de aprendizaje automático, la robustez se define como la capacidad de un modelo a mantenerse estable y confiable cuando se introducen datos con cierto grado de perturbación, ruido o simplemente errores. En las aplicaciones reales, los datos nunca son perfectos, contienen errores de medición, valores atípicos o información contaminada. Si hablamos de un modelo robusto este no debe cambiar los resultados que emite ante pequeñas modificaciones de los datos, pero si ocurre el modelo pierde confianza. Esta idea dio lugar a numerosas investigaciones donde se busca cuantificar, evaluar y mejorar los modelos para que sean más robustos. La mayoría de las investigaciones sobre robustez se han centrado en modelos de clasificación[1]. En este ámbito, la robustez se ha abordado desde distintas perspectivas.

2.1. Estudios de Robustez en Clasificación

Es importante señalar que existen distintas formas de introducir ruido, así como ruido de diferente naturaleza. Por ejemplo, (Fawzi, Moosavi-Dezfooli, y Frossard, 2016) y (Franceschi, Fawzi, y Fawzi, 2018) analizan la robustez de clasificadores frente a perturbaciones aleatorias[2] y adversariales[3]. En estos trabajos se demuestra que la distancia a la frontera de decisión, es decir, el límite que separa diferentes clases en el espacio de entrada del modelo, y su forma (lineal o curva) influyen en la robustez del clasificador: cuando el margen entre clases es estrecho, basta una mínima perturbación para que la instancia sea mal clasificada. Este es un factor clave para entender la vulnerabilidad de los modelos. El estudio concluye que los clasificadores resultan ser mucho más robustos al ruido aleatorio que al ruido adversarial.

Otro estudio que habla sobre la relevancia del ruido es (Zhu y Wu, 2004). En este trabajo se usan 17 *datasets* del repositorio UCI (Kelly, Longjohn, y Nottingham, 2024), aplican diferentes niveles de ruido y comparan sistemáticamente el impacto del ruido

[1]Tarea del aprendizaje supervisado, donde se dispone de etiquetas de salida verdaderas para guiar el aprendizaje.

[2]Ruido, también llamado aleatorio, que se genera sin intención, como para simular errores de lecturas de temperatura y cada medición con alteraciones ligeras y aleatorias sin un patrón estable.

[3]Son cambios pequeños, intencionados y cuidadosamente diseñados que se hacen sobre una instancia de entrada para engañar al modelo a propósito.

en las etiquetas (también conocido como ruido de clase, es un tipo de ruido que altera las etiquetas de salida de los datos) y en los atributos (el *attribute noise* afecta a las variables de entrada), concluyendo que ambos tipos de ruido pueden degradar el rendimiento, siendo el daño producido dependiente de la importancia estadística de los atributos alterados.

En el trabajo de (Petety, Liu, y Koyejo, 2020) se estudia la robustez de clasificadores binarios[4] frente a la introducción ruido simétrico en los atributos, es decir todos los atributos se corrompen con la misma probabilidad y ruido asimétrico independiente (donde cada atributo se corrompe con su propia probabilidad). El estudio muestra que la forma de aplicación de ruido junto con la elección de funciones de pérdida[5], es clave para la robustez. Estas conclusiones son de interés para el análisis de robustez en tareas no supervisadas como el *clustering*, porque sugieren que el comportamiento de un modelo bajo ruido depende no solo del algoritmo, sino también del modelo de perturbación introducido y de la función objetivo utilizada.

Entendiendo que hay distintos tipos de ruido y que se usan de forma distinta según el contexto del experimento, una pregunta interesante es saber cuánto ruido puede tolerar un modelo. Dado que introducir ruido puede modificar su salida, en (Zhao, Yu, y Liu, 2018) se propone una nueva métrica de robustez en clasificación llamada *Angular Breakdown Point* (ABP). Esta métrica computa cuántos *outliers* (instancias que no pertenecen a la misma distribución que el resto de datos) se necesitan para cambiar la orientación del hiperplano de decisión[6]. Según este trabajo, basta con una cantidad mínima de *outliers* para que el hiperplano se desvíe hasta 90°. Aunque este enfoque está pensado para clasificación, la idea puede trasladarse al *clustering* y usar una medida similar a ABP para indicar cuánto ruido es suficiente para romper la coherencia de los grupos.

En (Ljunggren y Ishii, 2021) se evalúa la robustez de varios modelos de clasificación frente a la introducción de ruido en los datos. Se utilizan cuatro conjuntos de datos reales para la detección de spam, diagnóstico de diabetes, cáncer de mama y detección de billetes falsos. El ruido que se introduce en los atributos numéricos es el ruido gaussiano (es el más común para simular el ruido, donde cada valor es alterado según una distribución normal y desviación estándar controladas, tal y como se explica en la Sección 3.5). En este estudio la desviación estándar se basa en la variabilidad de cada atributo. También se distinguen dos tipos de errores: el ruido de etiqueta y el ruido de atributos. Para medir la robustez se utiliza la métrica *Equalized Loss of Accuracy* (ELA), que cuantifica cuánto disminuye la precisión del modelo a medida que se incrementa el nivel de ruido. Se comparan los resultados de clasificación sobre los datos originales sin ruido y los datos perturbados. Los hallazgos muestran que algunos modelos, como Random Forest, son más resistentes al ruido que otros, dependiendo del tipo y magnitud de la perturbación. Este estudio refuerza el uso del ruido gaussiano como protocolo estándar para evaluar robustez, y destaca la importancia de contar con métricas específicas que midan la consistencia del rendimiento ante distintas condiciones de entrada.

Por otro lado, el estudio (Fabra-Boluda, Ferri, Ramírez-Quintana, y Martínez-Plumed,

[4]Tipo de aprendizaje supervisado en el que la clase solo puede tomar dos categorías posibles, ejemplo spam o no spam.

[5]Una fórmula matemática que mide qué tan mal lo está haciendo un modelo con una predicción.

[6]Un hiperplano de decisión es una línea o superficie imaginaria que separa los datos que son de distinta clase.

2024) propone un marco innovador para estudiar la robustez basado en ruido y en la dificultad de las instancias (calculada aplicando la teoría *Item Response Theory* propuesta en (Martínez-Plumed, Prudêncio, Martínez-Usó, y Hernández-Orallo, 2016)), la dificultad es mayor en instancias cercanas a fronteras de decisión o que se comportan como *outliers*. Se utiliza el estadístico *kappa de Cohen*[7] para evaluar la consistencia de las predicciones con y sin ruido. El análisis permite agrupar los modelos en familias según su sensibilidad al ruido y a la dificultad de los ejemplos, mostrando que no existe un modelo universalmente robusto.

Aunque estos estudios hablan sobre clasificación supervisada, las ideas que brindan, como la aplicación de niveles de ruido, uso de distintos tipos de algoritmos de clasificación, conjunto de datos y métricas de evaluación que se adapten al tipo de modelo, se pueden aplicar también a modelos no supervisados. Estos trabajos aportan una base conceptual sólida que inspira el enfoque experimental propuesto en este TFM para estudiar la robustez en modelos de *clustering*.

2.2. Estudios de Robustez en Clustering

Evaluar la robustez de *clustering* es un poco más complejo ya que no hay una etiqueta de salida que indique "verdad" con la cual comparar los resultados. En este caso, la robustez se analiza observando si las agrupaciones se mantienen estables al introducir perturbaciones.

El estudio presentado en (Bittner y cols., 2000) se basa en añadir ruido gaussiano aleatorio (con media 0 y desviación estándar 0.15) a unos datos sobre clasificación molecular del melanoma (usando microarrays de expresión génica), y observar cómo cambian las particiones con el fin de evaluar la robustez de los *clusters* que se forman. La métrica de robustez usada es *Weighted Average of Discrepant Pairs* (WADP) que mide qué proporción de pares de muestras que estaban juntas en el *clustering* original dejan de estarlo tras añadir ruido, muy similar a las métricas de Proporción de Pares Dañado (PDP) que se verá más adelante. El estudio llega a la conclusión que el grupo de 19 melanomas se mantiene estable frente a perturbaciones. El artículo justifica que, si un *cluster* es real y bien definido, debería persistir incluso si se añaden pequeñas perturbaciones. Aunque este trabajo solo analiza un dataset de ámbito biológico para un único valor de ruido, el enfoque metodológico es generalizable y se puede usar tipo de ruido similar para evaluar la estabilidad de los *clusters* en cualquier dominio de datos, tal y como se hace en este TFM.

En (Lu, Phillips, y Langston, 2019) se propone una métrica específica para la robustez en *clustering*. Concretamente, definen una nueva métrica, *robustness*, como la proporción de pares de elementos agrupados que permanecen juntos cuando se cambia de configuración o varían los parámetros del algoritmo. Para la experimentación se usaron 11 algoritmos de *clustering*, incluyendo jerárquicos y `k-means`, y 24 *dataset* biológicos. Su estudio muestra que los métodos jerárquicos muestran una mayor robustez que otros algoritmos, esto marca la importancia de evaluar no solo la calidad de los *clusters*, sino también su estabilidad ante cambios en configuraciones o condiciones de entrada.

[7] Mide consistencia corregida por azar entre dos clasificaciones. Es útil para evaluar robustez porque distingue entre "coincidencia real" y "coincidencia aleatoria"

En (von Luxburg, 2010) se presenta un marco teórico sobre la estabilidad en *clustering* y criterios para validar modelos y seleccionar el número de *clusters*. Principalmente, define la estabilidad como la capacidad de un algoritmo de producir resultados consistentes cuando se aplican ciertas perturbaciones de datos en diferentes muestras. En este trabajo se realiza una revisión exhaustiva sobre la estabilidad en *clustering*, explicando cómo introducir perturbaciones (submuestreo, adición de ruido y proyecciones aleatorias). También muestra diversas métricas para comparar particiones, como el indice de Rand Ajustado (ARI), el índice de Información Mutua Normalizada (NMI) y la variación de información que se definen más adelante. Analiza de manera detallada el caso del algoritmo `k-means`, distinguiendo entre un escenario idealizado (global óptimo) y uno práctico (con inicialización aleatoria y locales óptimos), mostrando que la estabilidad puede verse afectada tanto por la variación muestral[8] como por la presencia de múltiples óptimos locales, que implica que no se puede garantizan encontrar el óptimo global de la función objetivo, sino que dependen de la inicialización de centroides. Por último, resalta que una alta estabilidad no siempre implica el número correcto de *clusters*, pero sí permite descartar configuraciones totalmente inestables. Este trabajo constituye un referente fundamental para evaluar la robustez de los algoritmos de *clustering* en presencia de ruido y es un soporte conceptual directo para la metodología desarrollada en este trabajo.

Este TFM adopta la perspectiva de utilizar métricas de estabilidad y propone un estudio experimental de la robustez del *clustering* ante ruido en las instancias, extendiendo así la línea de trabajos previos desde la clasificación hacia el análisis no supervisado.

[8] Cuando se aplica *clustering* a diferentes muestras del mismo *dataset*, los resultados pueden variar, por ejemplo tomar el 80 % de los datos y aplicar `k-means`, puede que los *clusters* no salgan exactamente iguales que con el 100 %.

Capítulo 3

Robustez en clustering

En este capítulo introducimos los conceptos generales sobre el *clustering* y su evaluación, los algoritmos que vamos a usar en este TFM, e introducimos la forma en la que perturbamos las instancias (mediante la introducción de ruido) y las medidas de evaluación que usaremos para estudiar la robustez de los algoritmos seleccionados.

3.1. Clustering

El *clustering* es una técnica de aprendizaje no supervisado que busca dividir un conjunto de datos en grupos (*clusters*) de manera que los elementos dentro de un mismo grupo son similares entre sí y diferentes con respecto a otros *clusters* (Jain, Murty, y Flynn, 1999). El *clustering* se fundamenta en la idea de similitud/diferencia entre instancias, y dentro de esta técnica existen algoritmos que realizan esta división de grupos cada uno de manera distinta, como k-means, *clustering* jerárquico, DBSCAN, entre otros. Tiene una clara diferencia con los métodos supervisados, porque en las instancias no hay un atributo destacado (también llamado en el aprendizaje supervisado etiqueta o clase) conocido durante la fase de entrenamiento y que sirve para, una vez generado el modelo que estima los valores de ese atributo, validar las respuestas. En vez de ello, todos los atributos tienen la misma importancia y los grupos se infieren a partir de las relaciones entre las instancias. Para evaluar la calidad del agrupamiento se suelen usar dos medidas: el valor intra-cluster(MacQueen, 1967)(Jain y cols., 1999) y el inter-cluster (Jain y cols., 1999):

- Intra-cluster: Los puntos del mismo *cluster* son más parecidos posibles. Mientras menos varianza entre los puntos de un *cluster* y su centroide, el *cluster* sera más homogeneo y más compacto.
- Inter-cluster: Los *cluster* que se forman deben ser lo más distintos posibles, en otras palabras, que tan separados están, se mide la distancia entre los centroides de cada *cluster* (como en `k-means`) o entre los grupos (como en los algoritmos jerárquicos.)

Una buena partición es aquella en la que el valor intra-cluster es bajo y el inter-cluster es alto.

Cuando se distorsionan los datos (por ejemplo, introduciendo ruido, se espera que el intra-cluster aumente, los grupos pierden cohesión, y el inter-cluster disminuya, los

clusters se acercan más o se solapan. Por eso, medir robustez implica ver cómo se degradan estos dos valores al perturbar los datos.

Para determinar la similitud entre las instancias y formar los *clusters*, los algoritmos de *clustering* usan funciones de distancia. Existen muchas funciones que pueden usarse, como la distancia de Manhattan, la distancia Canberra, distancia Hamming o la distancia de Chebyshev, dependiendo del dominio y de las características de los datos. En este TFM usaremos la distancia euclidiana ya que es la que usa la implementación en R del algoritmo `k-means`, que es uno de los algoritmos de *clustering* que vamos a evaluar.

En términos simples, la distancia euclidiana es una generalización del teorema de Pitágoras, ya que mide la longitud del segmento de la recta que une dos puntos. Formalmente (Ecuación 3.1), dados dos puntos en un espacio n-dimensional, p y q, donde $p = (p_1, p_2, ..., p_n)$ e $q = (q_1, q_2, ..., q_n)$, la distancia euclidiana se define como

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{3.1}$$

3.2. Método de agrupamiento k-means

En (MacQueen, 1967) y (Lloyd, 1982), se define el algoritmo de `k-means` como la partición en k grupos sin una jerarquía entre los mismos. En otras palabras, `k-means` busca dividir un conjunto de instancias $X = x_1, x_2, ..., x_n$ en k grupos disjuntos $C_1, C_2, ..., C_k$ que no se solapan entre sí, pero cuya unión forma el conjunto de datos total, es decir, que cada instancia x_i pertenece a un único *cluster*. El objetivo es que las instancias que pertenecen a un clúster sean muy similares entre sí y poco similares a las instancias del resto de *clusters*.

Formalmente, `k-means` particiona un conjunto de datos en k *clusters* minimizando la suma de errores cuadráticos intra-cluster `Sum of Squared Errors` (Lloyd, 1982). Sea C_i el conjunto de puntos asignados al *cluster* i y μ_i el centroide correspondiente, definido como la media aritmética de los puntos en C_i. La distancia euclídea al cuadrado entre un punto $x \in C_i$ y su centroide se expresa como $\|x - \mu_i\|^2$. El objetivo del algoritmo consiste en encontrar la asignación de puntos y centroides que minimice la siguiente función de coste:

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2 \tag{3.2}$$

El algoritmo consiste en cuatro pasos (Lloyd, 1982):

- Elección: Se trata de escoger de forma aleatoria o mediante heuristicas, k elementos que serán los centroides.
- Asignación: usando la distancia euclidiana asignar cada elemento al *cluster* con centroide más cercano.

- Actualización: recalcular cada centroide según la media de los elementos asignados al mismo *cluster*.
- Iteración: Se debe repetir los pasos de Asignación y Actualización hasta que converjan[1].

3.2.1. Ventajas y limitaciones

Como algoritmo representante de los métodos particionales, `k-means` presenta algunas ventajas y limitaciones (Jain y cols., 1999). Este algoritmo es uno de los más utilizados en minería de datos y aprendizaje no supervisado por su simplicidad ya que se puede interpretar e implementar de manera sencilla; además, funciona bien con grandes volúmenes de datos y tiene una complejidad cercana a $O(n \cdot k \cdot d \cdot t)$ siendo n el número de instancias, d la dimensión (número de atributos) y t las interacciones del algoritmo (Jain, 2010).

El algoritmo requiere especificar el valor de k de antemano y no siempre es fácil. Además, la estimación de los centroides depende de la inicialización utilizada; diferentes puntos de partida pueden llevar a que el algoritmo converja en mínimos locales, generando así soluciones distintas (Arthur y Vassilvitskii, 2007). Otra posible desventaja es que asume que los *cluster* son iguales de tamaño, esféricos y convexos, lo que limita mucho su aplicación en distribuciones más complejas. Los *outliers* o el ruido pueden también representar un inconveniente ya que distorsionan los centroides, lo que afecta la calidad de la asignación de los puntos en cada iteración del algoritmo.

3.2.2. Variantes y extensiones

Existen diversas variantes que buscan mejorar el rendimiento del algoritmo original. Un ejemplo es `k-means++` (Arthur y Vassilvitskii, 2007), que optimiza el procedimiento de inicialización de los centroides para reducir la probabilidad de obtener soluciones de baja calidad. En el entorno R, este y otros métodos se encuentran implementados en librerías especializadas, como $ClusterR$.

3.3. Clustering jerárquico

El *clustering* jerárquico, es una técnica de agrupamiento cuya idea principal es construir un dendrograma, el cual representa la relación jerárquica entre los *clusters*. Existen dos enfoques principales (Jain y cols., 1999; Everitt, Landau, Leese, y Stahl, 2011):

- Aglomerativo (bottom-up): cada observación comienza como un *cluster* individual (una hoja en el dendrograma) y, en cada paso, se fusionan los dos *clusters* más similares hasta formar un único *cluster* que contiene todos los elementos.
- Divisivo (top-down): parte de un único *cluster* que contiene todos los elementos y, en cada paso, se va dividiendo en subclusters más pequeños hasta llegar a *clusters* individuales.

[1] La convergencia se alcanza cuando las asignaciones ya no cambian o, según la configuración del algoritmo, se alcanza un número máximo de iteraciones

Para construir un dendrograma es necesario definir un criterio de enlace, `linkage`, que determina cómo calcular la distancia entre dos *clusters*:

- Single-link: utiliza la distancia mínima entre pares de puntos de dos clusters; tiende a generar *clusters* alargados.
- Complete-link: se basa en la distancia máxima entre pares de puntos; produce *clusters* más compactos y esféricos.
- Average-link: considera el promedio de todas las distancias entre pares de puntos de los dos *clusters*.
- texttWard (Jr., 1963): mide el incremento en la varianza intra-cluster al fusionar dos clusters; busca minimizar la varianza interna en cada fusión. Cabe señalar que, en algunos lenguajes como R, la implementación inicial de este método (denominada `ward`) realizaba la fusión de *clusters* minimizando el incremento en la suma total de cuadrados, pero no directamente la distancia euclidiana al cuadrado. Posteriormente, esta discrepancia fue corregida en la versión `ward.D2`, que refleja con mayor fidelidad el concepto original.

3.4. Elección del k óptimo

La elección del número de *clusters* k es un aspecto fundamental tanto para los algoritmos `k-means` como para los métodos jerárquicos (Jain, 2010). En el caso de `k-means`, k debe definirse antes de ejecutar el algoritmo. Para determinar un valor adecuado de k se han propuesto diversos criterios, entre los más utilizados se encuentran: el método del codo o *Elbow Method*, Coeficiente de silueta o *Silhouette Score* y Gap Statistic (Tibshirani et al., 2001).

Para el método del codo (Jain, 2010), se calculan los cuadrados intra-cluster o *WCSS (Within-Cluster Sum of Squares)* igual que en Ecuación 3.2 para diferentes valores de k. A medida que aumenta k, el WCSS disminuye hasta llegar a un punto en el que la mejora se vuelve muy marginal. En este punto se observa que la curva forma un "codo" (de ahí el nombre del método), tomándose ese valor como el k óptimo.

El Coeficiente de silueta (*Silhouette Score*) Ecuación 3.3, evalúa que tan bien está separado cada *cluster*, y qué tan bien está colocado un punto dentro de su *cluster* (respecto a los demás clusters). Sus valores son: $s(i) \approx 1$ cuando el punto está bien agrupado y lejos de los demás clusters; $s(i) \approx 0$ cuando el punto está en la frontera entre dos *clusters* y $s(i) < 0$ si el punto estuviera mejor en otro *cluster* (mala asignación). Véase Ecuación 3.3, donde $a(i)$ es la distancia entre el punto i y los puntos de su mismo *cluster* y $b(i)$ es la distancia mínima entre i con respecto a los puntos de otro *cluster*.

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{3.3}$$

Gap Statistic (Tibshirani, Walther, y Hastie, 2002), compara la dispersión intra-cluster con respecto a una distribución nula aleatoria. El k óptimo se obtiene de maximizar la diferencia entre los datos reales y el modelo nulo.

A diferencia de `k-means`, en el *clustering* jerárquico no es necesario fijar un valor de k al inicio. En este caso, la elección del número de *clusters* se realiza mediante la

decisión del punto de corte en el dendrograma. Algunos criterios utilizados (Everitt et al., 2011) son:

- Altura del dendrograma: se observa el gráfico y se busca una altura de corte donde los *clusters* estén bien separados, identificando "saltos" grandes en la distancia de fusión.
- Silhouette Score: al igual que en `k-means`, se puede calcular la silueta para distintos números de *clusters* obtenidos tras cortar el dendrograma, eligiendo aquel que maximice este índice.
- Gap Statistic: también puede aplicarse a *clustering* jerárquico, comparando la dispersión intra-cluster con la esperada en datos aleatorios, ayudando a determinar un número de *clusters* significativo.
- Método de inconsistencia: analiza cuánto difieren las fusiones de un *cluster* respecto a los niveles inferiores; un *cluster* con un "salto" de inconsistencia elevado puede indicar un punto adecuado para cortar el dendrograma.

3.5. Perturbando las instancias

La robustez de un sistema puede verse como la capacidad de mantenerse inalterado (o poco alterado) ante situaciones que difieren de las consideradas cuando el sistema se diseñó. Para poder estudiar la robustez, necesitamos simular estas alteraciones del entorno en un espacio controlado de experimentación. La forma adoptada en este TFM para simular alteraciones en los datos consiste en añadir a las instancias un cierto nivel de ruido.

El ruido se puede entender como los errores en las mediciones, variaciones aleatorias o datos irrelevantes que alteran la estructura que se analiza. Se refiere a un componente que genera una alteración de la señal o algunos datos considerada verdaderos. Según (Duda, Hart, y G.Stork, 2001), la señal medida x' está compuesta por la señal real más el ruido. Este ruido es inevitable y puede generar errores incluso en modelos teóricamente óptimos. Similar a Ecuación 3.5, el ruido puede manifestarse como errores de medición en la recolección de datos, variabilidad no explicada en las características o información irrelevante que enmascara la estructura real.

Por otro lado, las perturbaciones son modificaciones deliberadas o accidentales que alteran la distribución original de los datos. Estas pueden ser errores en la transmisión, imprecisiones en los instrumentos o cambios introducidos intencionalmente para evaluar la robustez de un algoritmo. En (von Luxburg, 2010), a estabilidad de un algoritmo de *clustering* se puede medir perturbando el conjunto de datos y analizando la variación en los *clusters* obtenidos. En este sentido, las perturbaciones también pueden ser una herramienta metodológica útil.

Como se señala en (Halkidi, Batistakis, y Vazirgiannis, 2001), tanto el ruido como las perturbaciones afectan las métricas de validación de *clusters*, lo que obliga a incluir mecanismos de control y evaluación. La diferencia clave entre ambos conceptos es que el ruido está ligado a la naturaleza de los datos y al proceso de medición, por lo que es inevitable, mientras que las perturbaciones se entienden como cambios o manipulaciones de los datos, que pueden ser accidentales o intencionales.

El ruido gaussiano es una forma específica de ruido aleatorio que sigue una distribución normal, definida por la media μ y la varianza σ^2. Generalmente, la media se toma como cero (ruido centrado), y σ representa la desviación estándar, encargada de la dispersión del ruido. La función de densidad de probabilidad se expresa como:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.4}$$

En una distribución normal aproximadamente el 68 % de los datos caen en el rango $[\mu-\sigma, \mu+\sigma]$, el 95 % dentro de $[\mu-2\sigma, \mu+2\sigma]$ y el 99.7 % en $[\mu-3\sigma, \mu+3\sigma]$. Si se el ruido gaussiano es $N(0,1)$, media en cero y desviación estándar en 1, entonces el 68 % está entre -1 y $+1$, el 95 % en -2 y $+2$ y el 99.7 % entre -3 y $+3$, esto significa que la mayoría de las veces el ruido será pequeño cercano a la media, pero ocasionalmente pueden aparecer valores extremos.

La aplicación del ruido gaussiano a una señal o valor real x se representa como

$$x' = x + \epsilon \quad \text{con} \quad \epsilon \sim \mathcal{N}(\mu, \sigma^2) \tag{3.5}$$

donde x' es el valor observado y ϵ es un valor aleatorio generado según la distribución normal definida anteriormente.

Según (Bishop, 2006), la distribución gaussiana es el modelo más utilizado para ruido aditivo en problemas de aprendizaje estadístico, ya que muchos procesos aleatorios tiendan a comportarse de forma aproximadamente normal y facilita el modelamiento y análisis en algoritmos de clasificación y *clustering*.

En *clustering*, el ruido gaussiano puede difuminar los límites entre *clusters*, aumentar la superposición de datos y afectar la calidad de las particiones obtenidas.

3.6. Métricas de evaluación de la robustez en clustering

En este trabajo evaluamos la robustez del *clustering* usando tres métricas complementarias, lo que permite cuantificar tanto la similitud entre particiones, como la perdida de coherencia entre las instancias agrupadas. En (Ben-Hur, Elisseeff, y Guyon, 2002) y (von Luxburg, 2010) se definen distintas métricas para la evaluación de la estabilidad del *clustering* después de perturbar el conjunto de datos, ya sea por *sub-sampling* o por la agregación de ruido en las instancias. La idea es comparar el *clustering* original con el que se obtiene tras aplicar la perturbación. En este trabajo usaremos las siguientes formas de evaluar la robustez:

- Medir la similitud entre dos particiones obtenidas con el conjunto de datos original D_x versus el conjunto de datos perturbado DR_x: para ello, se cuentan cuantos pares continúan perteneciendo al mismo grupo antes y después de agregar el ruido al *dataset* original.
- Medir la cantidad de información compartida entre dos particiones obtenidas con el conjunto de datos original D_x versus el conjunto de datos perturbado DR_x: La idea es medir cuánto conocer la partición original D_x ayuda a predecir la partición después de agregar ruido. Esta incertidumbre se cuantifica mediante

la entropía[2]. A mayor desorden en los datos la entropía aumenta, lo que refleja que resulta más difícil determinar a qué *cluster* pertenece cada instancia.

- Medir la proporción de pares de instancias que se encontraban juntas en el *clustering* original y terminan separadas tras añadir la perturbación a D_x para obtener DR_x.

3.6.1. Índice de Rand Ajustado

El *Adjusted Rand Index* o ARI por sus siglas, es propuesto por (Hubert y Arabie, 1985) mide la proporción de pares de elementos en la que dos particiones[3] coinciden. Esta coincidencia se refiere a que ambos elementos estén juntos en ambas particiones o separados en ambas. El índice se ajusta corrigiendo el valor esperado de las particiones formadas al azar.
El *Rand Index* (RI) (Ecuación 3.6) es una proporción de pares en las que coinciden dos particiones, siendo a el número de pares (i, j) que pertenecen al mismo *cluster* en ambas particiones, b el número de pares en distintos *clusters* en ambas particiones, c número de pares juntos en la partición original pero no en la partición nueva y d el número de pares separados en la original y separados en la nueva partición.

$$RI = \frac{a+b}{a+b+c+d} \tag{3.6}$$

Pero como RI no ajusta el azar, se ajusta la fórmula dando lugar a la medida ARI (Ecuación 3.7) donde $\mathbb{E}[RI]$ es el valor esperado de RI si las asignaciones a los *clusters* fueran aleatorias

$$ARI = \frac{RI - \mathbb{E}[RI]}{1 - \mathbb{E}[RI]} \tag{3.7}$$

ARI corrige el azar y los resultados al aplicar esta métrica se pueden interpretar de la siguiente manera:

- 1 indica alta concordancia, que las particiones son idénticas.
- 0 indica que se forma agrupaciones aleatorias o tiene una concordancia al azar.
- Menores a 0, indica una agrupación formada peor que al azar.

3.6.2. Normalized Mutual Information

El *Normalized Mutual Information*, NMI, mide la cantidad de información que comparten dos particiones. Es decir, si tengo la información de una partición cuanto se podría predecir de la otra partición (Strehl y Ghosh, 2002). Los conceptos previos para entender esta métrica son la entropía (H) Ecuación 3.8 que mide la incertidumbre de una partición, si una partición divide un grupo en partes muy equilibradas la entropía es alta, donde $P(U_i)$ es la proporción de elementos en el clúster i de la partición U.

[2]En el sentido de la teoría de información introducida por (Shannon, 1948), la entropía es una medida de incertidumbre asociada a una variable aleatoria. En el contexto del *clustering*, representa la probabilidad promedio de asignar correctamente una instancia a un *cluster*.

[3]resultados de los clusterings

$$H(U) = -\sum_i P(U_i) \log P(U_i) \tag{3.8}$$

Otro concepto importante es la información mutua, $I(U;V)$ Ecuación 3.9 que mide cuanta incertidumbre se reduce en U al conocer V y viceversa

$$I(U;V) = \sum_i \sum_j P(U_i \cap V_j) \log \frac{P(U_i \cap V_j)}{P(U_i)P(V_j)} \tag{3.9}$$

Entendiendo estos conceptos se define de mejor manera NMI, como la información mutua no está normalizada se divide entre la entropía de ambas particiones, dando así una ecuación de NMI usando la media aritmética Ecuación 3.10, aunque no es la única variante, pero si la más común.

$$NMI(U,V) = \frac{2I(U;V)}{H(U) + H(V)} \tag{3.10}$$

Existen otras variantes (Vinh, Epps, y Bailey, 2010) como la geométrica Ecuación 3.11, todas variantes están acotadas entre 0 y 1, sin embargo, la variante geométrica se suele usar en artículos científicos por ser más simétrica y evitar sesgos cuando la entropía de particiones es muy diferente, mientras que la aritmética es más intuitiva y fácil de calcular.

$$NMI(U,V) = \frac{I(U;V)}{\sqrt{H(U) \cdot H(V)}} \tag{3.11}$$

$$NMI(U,V) = \frac{I(U;V)}{\text{máx}\{H(U), H(V)\}} \tag{3.12}$$

Dentro de la librería `aricode` se puede elegir el tipo de la variante que se quiere usar (Chiquet, Rigaill, Sundqvist, Dervieux, y Bersani, 2023), el valor por defecto es `max` Ecuación 3.12 aunque también existe otros: `max, min, sqrt, sum joint`, sin importar la variante que se elija el resultado de NMI se interpreta de la siguiente manera:

- 0, sin información compartida
- 1, perfecta concordancia, son particiones idénticas y se comparte la información

3.6.3. Proporción de Pares Dañados

La proporción de pares dañados por sus siglas PDP, mide la fracción de pares de instancia que se encontraban juntas después de hacer *clustering* a una partición sin ruido y que después de agregar ruido terminan en *clusters* distintos, se define con la Ecuación 3.13 donde U representa a la partición original, V es la participación alterada con ruido, i, j índices de dos instancias distintas siendo $i < j$, (i, j) representa un par de instancias. El denominador el total de pares que estaban juntos en la partición original U, con $U_i = U_j$ indica que i e y pertenecen al mismo *cluster*, mientras que el numerador representaría el número de pares rotos con $V_i \neq V_j$.

$$PDP(U \to V) = \frac{|\{(i,j) : i < j,\ U_i = U_j,\ V_i \neq V_j\}|}{|\{(i,j) : i < j,\ U_i = U_j\}|} \tag{3.13}$$

Aunque la métrica PDP no se trata tanto en artículos científicos como ARI o NMI, pero se inspira de métricas basadas en comparar pares (Hubert y Arabie, 1985) como el *Rand Index*. Mientras que ARI y NMI evalúan la similitud global entre particiones, comparan todas la relaciones posibles entre todos los pares de instancias de manera conjunta, PDP es diseñada para capturar la perdida local de coherencia cuando los datos se perturban *cluster* por *cluster*, es decir se fija en un solo subconjunto en concreto, los pares que si están juntos en la partición original, no le importa si dos instancias si dos nuevas instancias se juntaron solo se fija en las rupturas respecto a la partición base. Para la interpretación de la siguiente métrica se tiene:

- 0, si ningún par se rompió, los pares originales permanecen juntos
- 1, si todos los pares se rompieron
- NA, caso extremo, no existen pares juntos en el *clustering* base.

Finalmente, esta combinación de tres métricas nos da una evaluación integral de la robustez, si al perturbar los datos con ruido el algoritmo de *clustering* cambia los resultados o se mantiene, ARI compara las dos particiones, antes y después del ruido. En cuanto a NMI, si agregamos ruido y volvemos a calcular el *clustering*, el resultado de las particiones puede que comparta mucha formación (NMI alto) dando un posible indicativo para asegurar la robustez o por lo contrario (NMI bajo) indicaría que las etiquetas cambiaron mucho y finalmente con PDP más enfocado en la ruptura de pares de forma más local.

Capítulo 4

Diseño de experimento

Este capítulo describe el diseño del experimento propuesto de manera detallada, también incluye la metodología que se siguió para obtener los resultados deseados. En el contexto del experimento, se busca ver qué algoritmo tiene mayor robustez, entendiendo por robustez la capacidad de un algoritmo de mantener la coherencia de sus agrupaciones cuando se introduce ruido a los atributos. En las próximas secciones, se muestra el flujo de trabajo que se siguió y los componentes metodológicos del estudio.

4.1. Flujo metodológico del experimento

El flujo para este experimento se compone de los siguientes pasos:

1. Selección del *dataset* original, en adelante denominado D_x donde x es el número de *dataset* dentro de la colección de *dataset*.
2. Limpieza de cada *dataset* original, que comprende la selección de las columnas relevantes para el *clustering*, así como la numeración de las columnas binarias no numéricas seleccionadas.
3. Creación de los conjuntos de datos perturbados, en adelante $D_{R:x}$, inyectando ruido gaussiano sobre atributos numéricos, para distintos valores de ruido (tal y como se explica en el siguiente capítulo).
4. Aplicación de estandarización para todos los *datasets*.
5. Determinación del valor de k (número de *clusters*) óptimo para cada D_x.
6. Aplicación de los algoritmos de *clustering* `k-means` y *clustering* jerárquico.
7. Comparación de los agrupamientos generados con el *dataset* sin ruido D_x y los *datasets* con ruido D_{R-x} con las siguientes métricas de evaluación de robustez (definidas en la Sección 3.6): ARI, NMI y PDP.

Se ejemplifica en la Figura 4.1, los pasos que están punteados (estabilización de datos y luego introducción de ruido a los datos) es un paso que se descarta y se explica mas adelante en la Sección 4.5.

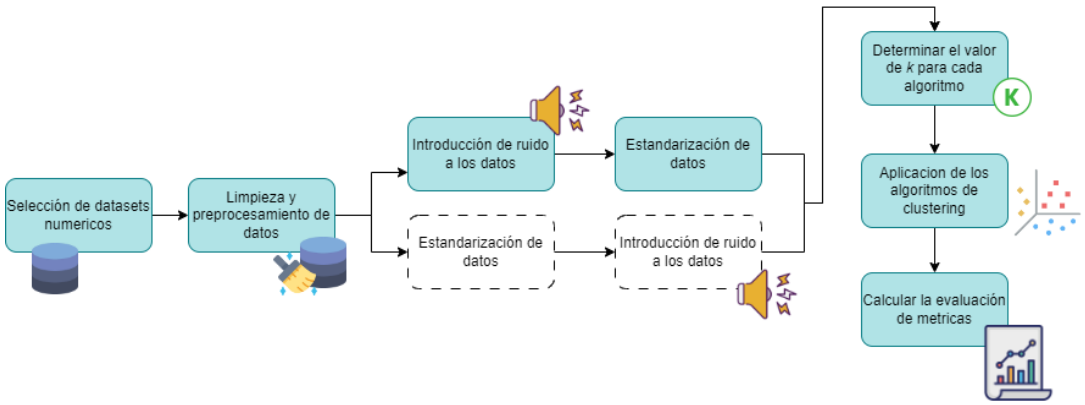

Figura 4.1: Flujo metodológico del experimento. Los pasos 3 y 4 varían en orden creando una bifurcación debido a que se probaron ambos caminos dentro del experimento.

4.2. Selección de datasets

Para este trabajo se buscó 20 *datasets*, estos *datasets* deben cumplir ciertas características para poder trabajar con los datos y se pueda aplicar un algoritmo de *clustering*. Se vio por conveniente hacer una selección de diversos *datasets*, que por la naturaleza del experimento son numéricos, de acceso público y fueron obtenidos de las plataformas UCI *Machine Learning* y *Kaggle*. Los criterios que se usaron para la elección son los siguientes:

- Los datos del *datasets* deben ser en su mayoría numéricos, esto refiere que puede haber características de otro tipo de dato, pero estas no deben ser atributos clave dentro del *dataset*, en lugar de eso debe ser atributos prescindibles para poder quitarlos. Los atributos deben ser adecuados para algoritmos que usan la distancia euclidiana como `k-means` o *clustering* jerárquico que se usan en este trabajo.
- Los atributos pueden ser categóricos si solo poseen dos categorías, de la misma manera se permite atributos binarios.
- El tamaño de los *datasets* debe ser variado, pero en rangos moderados, se plantea un rango de 200 a 60 mil instancias, esto para poder manejar los datos en el servidor.
- La data debe ser real de distintos campos del conocimiento. Para dar una variedad al momento de realizar los experimentos, se usa dominios de medicina, energía, calidad de alimentos, etc.

En la Tabla 4.1 se encuentran los *datasets* que se recopilaron, la información que se muestra es sobre la data cruda, antes de pasar los datos por cualquier filtro o cualquier modificación de preprocesamiento, es un resumen de la cantidad de atributos y características que se manejaran dentro del proyecto.

Nro	Nombre	Fuente	Instancias	Atributos
1	Seeds Dataset	UCI	210	8
2	Wholesale Customers	UCI	440	8
3	Banknote Authentication	UCI	1372	5
4	Travel Review Ratings	UCI	5456	26
5	EastWestAirlines - Hierarchical *clustering*	Kaggle	3999	12
6	Dataset of Songs in Spotify	Kaggle	42305	22
7	Power Plant Data	Kaggle	6410	5
8	California Housing Prices	Kaggle	20640	10
9	Heart Failure Clinical Records	UCI	299	13
10	Smoke Detection Dataset	Kaggle	62630	16
11	Wine Quality	UCI	6497	12
12	Dry Bean	UCI	13611	17
13	HTRU2	UCI	17898	9
14	Concrete Compressive Strength	UCI	1030	9
15	Gas Turbine CO and NOx Emission Data Set	UCI	36733	11
16	Breast Cancer Wisconsin (Diagnostic)	UCI	569	32
17	Rice (Cammeo and Osmancik)	UCI	3810	8
18	Credit Card Dataset for *clustering*	Kaggle	8950	18
19	Yeast	UCI	1484	10
20	Predict Diabetes	Kaggle	768	9

Tabla 4.1: Listado de *datasets* utilizados antes de cualquier limpieza

El primer *dataset* de nuestra colección para este experimento es *Seeds Dataset*, que contiene las mediciones geométricas de tres variedades de granos de trigo distintas: Kama, Rosa y Canadian, con 70 elementos cada una, esa información se obtuvo de UCI y cuenta con siete atributos que fueron recolectados usando técnicas de *soft X-ray* y el paquete GRAINS.

El segundo *dataset* es *Wholesale Customers* contiene la información de un distribuidor mayorista, esta información incluye los datos del cliente y abarca los gastos anuales en unidades monetaria en diversas categorías de productos. Los atributos incluyen productos frescos, lácteos, alimenticios, congelados y delicatessen; canales de clientes como hotel, restaurante y cafés; y información sobre la región. Algo a destacar es que el *dataset* contiene dos atributos categóricos: región y canal.

Para el *dataset Banknote Authentication* la información se obtuvo de evaluaciones con datos de billetes auténticos y falsos. El *dataset* contiene las características que obtuvieron de imágenes de 400x400 pixeles y se usó *Wavelet Transform* como herramienta de extracción de características de imágenes. Solo contiene las características

no se tiene la información de las imágenes como tal.

En el *dataset Travel Review Ratings* se encuentran las opiniones de Google atractivos en todo Europa, estos representados por sus categorías como playas, iglesias, restaurantes, etc. Estas opiniones oscilan entre uno a cinco, pero dentro del *dataset* se usa una media de la valoración de los usuarios por categoría.

Nuestro primer *dataset* del repositorio de *Kaggle* es *EastWestAirlines - Hierarchical clustering* que contiene información de pasajeros que pertenecen al programa de "viajero frecuente"de una línea aérea, la información incluye la cantidad de millas, como se acumularon o gastaron, con la finalidad de poder indicar grupos de pasajeros y segmentar ofertas según su cantidad de millas.

El siguiente es *Dataset of Songs in Spotify*, un *dataset* muy interesante que contiene las características musicales de distintos *tracks* de la plataforma de Spotify. En concreto se incluyen características como *danceability*, *energy*, *acousticness*, etc. aunque contiene datos categóricos también como *genre* y otros textuales como URI o *song_name*. Este *dataset* puede usarse para agrupar canciones como bailables, tristes, relajantes, entre otras categorías.

Power Plant Data es un *dataset* que contiene información de una central hidroeléctrica, los datos se encuentran desde el 2006 al 2011, los datos son medias horarias de variables ambientales, temperatura ambiente (TA), presión ambiente (PA), humedad relativa (HR) y vacío de escape (V). La última variable sirve para predecir la producción horaria neta de energía eléctrica (PE) de la central.

El *dataset California Housing Prices* contiene información del censo de 1990 en los distritos de California, la data contiene información de longitud, latitud, total_rooms, median_house_value, etc, posee datos numéricos, pero también categóricos, por otro lado, dentro del mismo repositorio de *Kaggle* se recomienda realizar preprocesamientos previos para trabajar con los datos.

Un *dataset* medico es *Heart Failure Clinical Records* que contiene información de registros clínicos de pacientes, incluyen información de insuficiencia cardíaca, anemia y otros, aunque es pequeño está muy bien definido y no todos sus datos son numéricos, posee tres atributos binarios que se pueden manejar para *clustering*.

Smoke Detection Dataset es un *dataset* creado a partir de la extracción de un dispositivo IoT, contiene información de incendios y distintos muestreos en distintos entornos que garantiza un buen conjunto de datos. El conjunto es de 60 mil lecturas con 16 características, contiene características de tipo fecha, contadores y etiquetas que se pueden tratar para este trabajo más adelante.

El *dataset Wine Quality* es una colección de registros de calidad de vinos, dentro del conjunto existen dos tipos de vinos, tintos y blancos, procedentes del norte de Portugal. Contiene un promedio de cuatro mil instancias y once características que son de tipo físico químicas y sensoriales, no hay información sobre tipos de uva, marcas o precios del vino.

Dry Bean es un *dataset* extraído del repositorio UCI, la información que contiene es de 16 características extraídas de imágenes de siete variedades de judías secas, estos datos son dimensionales y de forma. Todos los datos son numéricos y tiene un aproximado de trece mil instancias.

HTRU2 es un conjunto de datos que muestra datos de candidatos a pulsares, tipos de estrellas de neutrones raras de gran interés científico, que fueron recogidos durante sondeo. El *dataset* este compuesto por 1639 instancias positivas (pulsares), 16,259 negativas (ruido/RFI) y todas ellas etiquetas por humanos.

El *dataset Concrete Compressive Strength* se obtuvo del repositorio de UCI, y posee datos cuantitativos reales, no hay variables categóricas. La información que tiene es sobre la resistencia del concreto. Se dispone de un aproximado de mil instancias y 8 características, este *dataset* esta originalmente está diseñado para regresión, pero se puede trabajar para *clustering*.

Gas Turbine CO and NOx Emission Data Set extraído del repositorio UCI, se usa para analizar contaminantes como CO y NOx, los datos se obtuvieron de sensores de una turbina de gas durante 5 años agrupadas por hora. Son mediciones, variables relacionadas al ambiente y funcionamiento de la turbina. Todas continuas reales excepto por el *year*, consta de 36 mil instancias con doce atributos El siguiente *dataset* se obtuvo de UCI, el *Breast Cancer Wisconsin (Diagnostic)* tiene un aproximado de 500 instancias y 30 características, contiene información extraída de imágenes digitalizadas de aspirados con aguja fina (FNA) de masas mamarias, cada instancia es un examen clínico, posee etiquetas de si se clasifican en malignos o benignos. *Rice (Cammeo and Osmancik)* es un *dataset* de UCI, contiene información de dos tipos de granos cultivadas en Turquía, la información se obtuvo de imágenes de donde se extrajeron siete características morfológicas por grano de arroz, contiene alrededor de tres mil instancias, algunas enteras y otras continuas.

El *Credit Card Dataset for clustering* extraído de *Kaggle* tiene información del comportamiento de aproximadamente nueve mil clientes con tarjetas de crédito en un rango de 6 meses y tiene 18 variables entre comportamiento y financiero, las variables son de tipo continuo, binario y enteros.

El *dataset Yeast* del repositorio UCI, contine información para predecir la localización celular de proteínas, contiene cerca 1.5 mil instancias y cada una de ellas es una proteína con características bioquímicas. Son ocho atributos que describen propiedades biológicas derivadas de secuencias de aminoácidos y todas de tipo continuo.

El *dataset Predict Diabetes* que se extrajo de *Kaggle*, fue recopilado por *National Institute of Diabetes and Digestive and Kidney Diseases*, el objetivo es predecir si una mujer puede tener diabetes en base a ciertas características médicas, los datos son solo de mujeres mayores de 21 años, contiene 768 instancias, cada una un registro de paciente y contiene medidas médicas.

4.3. Preparación de data

En esta subsección se realiza la limpieza de datos de cada uno de los elementos del conjunto de 20 *datasets*, esta limpieza es para poder trabajar con los algoritmos de *clustering* más adelante. Todo *dataset* debe pasar por una revisión para asegurar que no exista presencia de valores nulos. Por otro lado, cada *dataset* es distinto y pertenece a un dominio específico del conocimiento, por tanto, cada conjunto tiene su propia estructura, formato y características que deben abordarse de manera individual para su posterior uso. Cada *dataset* fue sometido a un preprocesamiento de manera individual, dentro de este proceso se incluye:

- Eliminación de las columnas categóricas o no numéricas, por ejemplo, fechas, texto o ubicaciones textuales.
- Eliminación de instancias con datos faltantes.
- Normalización mediante escalado estándar, para no afectar la distancia euclidiana al tener atributos de diferente escala.

A continuación, se evalúa los distintos métodos para la preparación de data que se siguieron en este trabajo:

4.3.1. Carga de datasets

Al realizar la búsqueda de *datasets* en distintos repositorios los formatos que se descargan tienen distintos formatos: `.txt`, `.csv`, `.arff`, `.data` y `.xls`. Los textos planos (`.txt`) usa delimitadores por espacios pero también pueden personalizarse, los *Comma-Separated Values* (`.csv`) también pueden separarse por coma o punto y coma, los *Attribute-Relation File Format* contiene datos, metadatos en un texto plano estructurado y su forma de carga varia; el formato de `.data` es propio de repositorios como UCI y está asociado a otro archivo: `.name` para la cabecera y finalmente el formato de Excel (`.xls`) contiene hojas, tipos de celdas, fórmulas entre otras características, igual que el anterior formato se cargan de manera distinta. Se debe considerar estos formatos al momento de la carga de datos, delimitadores y títulos de calabacera para evitar la pérdida de datos.

4.3.2. Limpieza de elementos de la colección

Después de tener los *datasets* cargados en memoria, para poder continuar con el flujo se debe analizar cada *dataset* de manera individual y eliminar atributos que no son relevantes para realizar el *clustering* o que tienen demasiados valores nulos. En este punto también se eliminan los atributos que representan la clase (algunos *datasets* correspondían a tareas de clasificación), ya que, de usarse en clustering, podrían dar lugar a agrupaciones artificiales. Respecto a los atributos categóricos, algunos se numeran para usarse con la distancia euclidiana, como se explicará más adelante.

Dataframe	Numéricas	Enteros	Categóricas	Total
*data*_1	7	1	0	8
*data*_2	0	8	0	8
*data*_3	4	1	0	5
*data*_4	24	0	2	26
*data*_5	0	12	0	12
*data*_6	10	4	8	22
*data*_7	5	0	0	5
*data*_8	9	0	1	10
*data*_9	3	10	0	13
*data*_10	8	8	0	16
*data*_11	11	1	0	12
*data*_12	16	0	1	17
*data*_13	8	1	0	9
*data*_14	9	0	0	9

Dataframe	Numéricas	Enteros	Categóricas	Total
*data*_15	11	0	0	11
*data*_16	30	1	1	32
*data*_17	7	0	1	8
*data*_18	14	3	1	18
*data*_19	8	0	2	10
*data*_20	2	7	0	9

Tabla 4.2: Tipos de atributos de la colección por *dataset*

Como puede observarse en Tabla 4.2, hay *datasets* en los que la cantidad de categóricos represente una buena proporción de los atributos, otros en los que proporción es mínima (1 o 2 variables categóricas) y *datasets* que no contienen variables categóricas.

A continuación, se describe el preprocesado realizado dependiendo del tipo del atributo:

- Atributos **numéricos discretos o continuos**: En su mayoría se mantienen, sin embargo se debe considerar la variabilidad, para ver si las columnas aportan información, usando la función `auditar_columnas`[1] se procede a revisar la razón $sd/rango$ y se considera que un atributo tiene baja variabilidad si $sd/rango$ <0.1, media si está entre 0.1 y 0.3, y alta si supera 0.3. Aquellos que tienen baja variabilidad son descartados por no aportar información discriminatoria.
- Atributos **categóricos**: mediante la función `auditar_columnas` verificamos la cantidad de valores únicos y los conteos por para cada valor, si la cantidad de valores únicos es mayor a 2 y menor a 20, se procede a evaluar la distribución para cada valor, revisando los conteos para cada valor único, se descarta el atributo en caso de presentar fuerte desbalance. Un caso particular es si solo posee 2 categóricos, en cuyo caso se trata como binario y se describe más adelante.
- Atributos **binarios**: en este caso se observó que pueden ser numéricos al estar codificados como 0 y 1, pero también pueden ser de tipo textual o carácter, como F (femenino) y M (masculino). En este último caso se aplica la *codificación binaria simple* que realiza una numeración del atributo, asignando un valor numérico a cada valor textual, como se muestra en el siguiente ejemplo para un atributo con valores F y M:

$$\text{Sexo} : \{F, M\} \to \{0, 1\}, \quad \begin{cases} F \mapsto 0 \\ M \mapsto 1 \end{cases} \tag{4.1}$$

- Atributos **cadenas textuales**: el algunos *datasets*, como por ejemplo *Dataset of Songs in Spotify*, existen atributos textuales como el título, álbum y URL, que no aportan información (muchos de sus valores son únicos en el *dataset*) por lo que se **eliminan**.

[1] Función que realiza un análisis de cada atributo de un dataframe, como cantidad de valores únicos, la varianza, desviación estándar (sd) y rango

- Atributos **etiquetas**: como se ha mencionado anteriormente, estos atributos son necesarios para un contexto de clasificación supervisada, pero en clustering no se existen atributos destacados (se usan todos) por lo que se procede a **eliminarlos**.

- Atributos **identificadores**: Atributos como IDs o códigos únicos se eliminan ya que no aportan información útil al proceso de agrupamiento.

La última parte para el preprocesamiento de datos involucra el análisis de NAs o nulos dentro de la data de cada *dataset*. Como los algoritmos que empleamos son `k-means` y *clustering* jerárquico, dependemos de las distancias euclidianas y se necesita datos numéricos completos. Las acciones realizas son:

- Si las columnas poseen demasiados valores nulos o NAs se procede a eliminar la columna o atributo.

- Si las filas poseen valores nulos o NAs se procede a eliminar las instancias.

Finalmente, se procede a la eliminación de duplicados. Aunque el *clustering* soporta datos duplicados, esto genera densidad en ciertas áreas, distorsiona las distancias e influye en los centroides de cada *grupo*.

La Tabla 4.3 muestra el resultado final del preprocesado.

Dataset	**Filas Iniciales**	**Filas Finales**	**% Filas Eliminadas**	**Cols Iniciales**	**Cols Finales**	%**Cols. Eliminadas**
data_1	210	210	0.0%	7	7	0.0%
data_2	440	440	0.0%	7	7	0.0%
data_3	1372	1348	1.7%	5	4	20.0%
data_4	5456	5451	0.1%	26	24	7.7%
data_5	3999	3998	0.02%	12	7	41.7%
data_6	42305	35096	17.0%	22	11	50.0%
data_7	6410	6396	0.2%	5	4	20.0%
data_8	20640	20433	1.0%	10	9	10.0%
data_9	299	299	0.0%	13	12	7.7%
data_10	62630	62628	0.003%	16	12	25.0%
data_11	6497	5318	18.1%	12	11	8.3%
data_12	13611	13543	0.5%	17	16	5.9%
data_13	17898	17898	0.0%	9	8	11.1%
data_14	1030	996	3.3%	9	8	11.1%
data_15	36733	36726	0.02%	11	8	27.3%
data_16	569	569	0.0%	32	30	6.3%
data_17	3810	3810	0.0%	8	7	12.5%
data_18	8950	8636	3.5%	18	14	22.2%
data_19	1484	1453	2.1%	10	7	30.0%
data_20	768	768	0.0%	9	8	11.1%

Tabla 4.3: Comparación del número de filas y columnas; antes y después del preprocesamiento

4.4. Introducción de ruido

El siguiente paso, es generar los conjuntos DR_x aplicando distintos niveles de ruido a cada dataframe D_x de la colección de *datasets* obtenidos tras el paso descrito en la sección anterior. En este TFM aplicamos ruido Gaussiano con media igual a 0 y desviaciones estándar de 0 a 1.5 en intervalos de 0.2 (un total de 8 valores). El ruido Gaussiano es un valor numérico, por tanto, se puede aplicar a los atributos numéricos que se tiene en cada *dataset*. Sin embargo, no se puede realizar un barrido completo por el *dataframe*, ya que después de la limpieza o preprocesamiento los atributos binarios, aunque numéricos, solo pueden tomar los valores enteros 0 y 1, Por ello, una vez identificados los valores binarios por dataframe, se maneja una lista con los nombres de los atributos que son binarios y a qué dataframe pertenecen. En el momento de inyectar el ruido, estos atributos binarios no se consideran, debido a que al aplicar el ruido se realiza una adición pequeña que simula un error o ruido y en los atributos binarios solo se puede manejar enteros y no decimales.

A continuación, se muestra mediante un ejemplo el efecto de introducir ruido gaussiano en la siguiente instancia:

$$\text{instancia} = [2.5,\ 3.0,\ 0,\ 4.1] \tag{4.2}$$

En la que el tercer atributo 0 es binario. El ruido Gaussiano ε se genera usando una desviación estándar de $\sigma = 0.2$ tal y como se ve en 4.4. Como se ha mencionado, no se calcula ruido para el atributo binario.

$$\varepsilon = [\mathcal{N}(0,\ 0.2^2)] = [0.03,\ -0.15,\ 0,\ 0.07] \tag{4.3}$$

Obteniendo la siguiente instancia con ruido:

$$\text{instancia}_{\text{ruido}} = [2.5 + 0.03,\ 3.0 - 0.15,\ 0,\ 4.1 + 0.07] = [2.53,\ 2.85,\ 0,\ 4.17] \tag{4.4}$$

Este proceso se realiza para los 20 *dataset* y los 8 niveles de ruido (7 si excluimos el ruido con desviación estándar igual a 0 ya que en realidad representa la ausencia de ruido), por lo que obtendremos un total de 140 *datasets* perturbados.

4.5. Estandarización de datos

Una vez generados los conjuntos con ruido, se procede a aplicar una estandarización de los datos de tipo escalado, para que todos los atributos tengan media 0 y desviación estándar igual a 1.

$$D_{x-\text{estandarizado}} = \frac{x - \mu}{\sigma} \tag{4.5}$$

La motivación para aplicar la estandarización es para poner en la misma escala los valores de los atributos, para que así al calcular la distancia entre dos instancias, todos los atributos contribuyan de forma uniforme.

Dentro del flujo se analizaron dos caminos posibles:

- Estandarización antes de agregar el ruido: Se analizó este flujo, pero fue descartado, ya que al aplicar el ruido los datos ya no estarían estandarizados (es decir, la media no sería 0 y la desviación estándar no sería 1).
- Estandarización después de agregar el ruido. Este es el flujo seguido. Nótese que si después de agregar ruido se realiza la estandarización, el ruido agregado puede verse diluido si es muy pequeño. No obstante, el estudio incluye la agregación de ruido de forma incremental, lo que nos permite analizar su efecto en el agrupamiento (mucho más evidente para valores grandes de ruido, como se muestra en el capítulo siguiente).

4.6. Búsqueda del k óptimo

Con los datos limpios y normalizados, el siguiente paso es calcular el valor óptimo de k. Son muy conocidos los métodos clásicos como *Elbow* y *Silhouette Score* (Sección 3.4) para el algoritmo `k-means`, mientras que para el *clustering* jerárquico se suele usar el dendrograma. Sin embargo, para calcular un mejor valor de k adecuado para cada algoritmo es conveniente usar la mayor cantidad de métodos y aplicarlos a la data original D_x. De esta forma, se recopilan los valores de k óptimo calculado por cada método y se elige aquel en el que más métodos coinciden (el valor más frecuente).

4.7. Aplicación de *clustering*

El siguiente paso es la aplicación de los algoritmos `k-means` y *clustering* jerárquico a todos los conjuntos de datos generados, tanto originales como los obtenidos por la aplicación del ruido.

4.8. Aplicación de métricas de evaluación

En el último paso de la metodología que se siguió se aplican las métricas de evaluación para comprobar la robustez de los algoritmos de clustering, aplicando las métricas de evaluación definidas en la Sección 3.6. La idea es comparar la agrupación original (obtenida con los datos originales) con la agrupación obtenida con los datos alterados. Los valores obtenidos permiten analizar en qué medida se mantiene la coherencia en los grupos y por tanto evaluar la robustez del algoritmo bajo ciertos niveles de perturbación. Si ARI y NMI se mantienen altas, pero PDP se mantiene baja el *clustering* es robusto, por lo contrario, se puede decir que el algoritmo es sensible a ruido.

Capítulo 5

Experimentos

En este capítulo se implementa los pasos de la metodología descrita anteriormente y la creación de gráficas. Se comienza con la descripción de las herramientas, lenguaje empleado y librerías utilizadas. Se ahondará más en la implementación del código dentro de los pasos de extracción de datos, preprocesamiento y aplicación de ruido. Dentro de los pasos como estandarización, cálculo de los algoritmos de *clustering* y evaluación de métricas se describirán los scripts que se pasaron al servidor para su ejecución y los datos obtenidos.

5.1. Herramientas y tecnologías utilizadas

Para el desarrollo de este trabajo, la implementación se desarrolló en el lenguaje R usando la plataforma de RStudio. Se escogió el lenguaje R por ser ampliamente utilizado en análisis de datos, minería de datos y *Machine Learning*. Al ser un lenguaje interpretativo es ideal para experimentación, facilita las pruebas y ayuda en este trabajo en el manejo de grandes conjuntos de datos y aplicación de técnicas estadísticas como el *clustering*. También tiene una naturaleza orientada a vectores, que hace que el código sea más conciso, legible y eficiente. Rstudio es un IDE diseñado específicamente para trabajar con R y permite usar RMarkdown, mientras el IDE facilita el desarrollo de código, ejecución y visualización de una forma organizada, productiva y profesional, con RMarkdown complementamos la herramienta para crear documentos interactivos y reproducibles.

5.1.1. Paquetes y herramientas asociadas a R

En esta subsección describiremos las librerías que se usaron en R y que están enfocadas en la carga, transformación, análisis, evaluación y visualización de datos para obtener la robustez del *clustering*.

Base R

Estas son un conjunto de funciones incorporadas en el lenguaje R que forman parte del entorno de R, no requieren la instalación de paquetes. Se utilizaron funciones como `k-means()`, `dist()`, `hclust()` y `cutree()` para realizar algunos experimentos, ejecutar los algoritmos de agrupamiento y el cálculo de distancias.

readxl

Encargado de la lectura de archivos en formato `.xls` y `.xlsx`. Se utilizó para importar *datasets* multiformato, especialmente aquellos disponibles en hojas de cálculo como en el repositorio UCI.

farff

Permite la lectura y escritura de archivos en formato `.arff` (Attribute-Relation File Format), comúnmente utilizados en el aprendizaje automático. Un archivo `.arff` posee dos partes, la cabecera (*header*) donde se encuentran los atributos y tipos; la segunda es la de datos, donde están las instancias.

ggplot2

Paquete para generar la visualización avanzada basado en la gramática de gráficos. Permite visualizaciones complejas con poco código, se puede configurar el gráfico y comparar múltiples variables de manera clara. En este trabajo se utilizó principalmente para:

- Graficar la variación de métricas de robustez (ARI, NMI, PDP) frente a los distintos niveles de ruido y sus incrementos.

NbClust

Este paquete de R determina automáticamente del número óptimo de *clusters* utilizando múltiples índices de validación interna, respondiendo a la pregunta "¿Cuál es el número de grupos (k) para particionar un conjunto de datos?". En este trabajo es usado para seleccionar el valor de k tanto en `k-means` como en el algoritmo jerárquico (`Ward.D2`). `NbClust` funciona siguiendo este flujo:

- Obtiene un *dataset* y el rango de valores posible que puede tomar.
- Aplica un algoritmo de *clustering*.
- Calcula automáticamente hasta 30 índices de validación interna como: Calinski-Harabasz, Silhouette, Dunn index y Davies-Bouldin, entre otros.
- Cada índice muestra cuál es su mejor k y se registra el voto.
- Después determina el k óptimo por mayoría de voto entre los índices.

mclust

Este es un paquete especializado en el agrupamiento de modelos, ofrece el modelado y el análisis. Tiene dos funcionalidades, el `Model-Based Clustering`, que estima el número óptimo de grupos, uso de modelos de mezcla gaussiana y selección del modelo más adecuado; y la otra funcionalidad es de medidas de concordancia, que se utilizó específicamente para calcular el ARI que mide la concordancia entre particiones originales y perturbadas.

aricode

Es un paquete de R orientado a la evaluación de calidad en *clustering* mediante métricas de información mutua, implementa métricas basadas en la teoría de la información para comparar dos particiones y evalúa que tan parecidos son dos resultados de *clustering*. Se empleó para calcular el *Normalized Mutual Information* (NMI).

5.1.2. Servidor externo

Por la cantidad de datos que se manejan es necesario una buena capacidad de cómputo, por tanto, dentro del entorno local se ejecutó el flujo normal, que no necesita muchos recursos, mientras algunos scripts por la cantidad de datos y complejidad se ejecutaron en un servidor externo. Estos experimentos se realizaron en el servidor *«cremaet.dsic.upv.es»* a cargo del Departamento de Sistemas Informáticos y Computación de la Universitat Politècnica de València, a continuación describo las características.

Parámetro	**Detalle**
Sistema operativo	Debian GNU/Linux 13 (trixie)
Kernel	Linux 6.12.35 (SMP PREEMPT_DYNAMIC)
Arquitectura	x86_64 (64 bits)
Procesador	AMD Ryzen 9 9950X (16 núcleos físicos / 32 hilos)
Frecuencia base	600 MHz
Frecuencia máxima	5752 MHz
Caché	64 MiB (L3), 16 MiB (L2), 768 KiB (L1d), 512 KiB (L1i)
Memoria RAM total	30 GiB
Libre al momento de ejecución	16 GiB
Swap disponible	31 GiB
Almacenamiento en `/home`	2.7 TiB libres de 3.5 TiB
Virtualización	AMD-V
Versión de R	4.5.0 (publicada el 11 de abril de 2025)
Mitigaciones de seguridad	Soporte actualizado para Spectre, Meltdown y otras vulnerabilidades conocidas.

Tabla 5.1: Características del entorno de ejecución remoto

Para el desarrollo de este trabajo se utilizan dos entornos complementarios, como se mencionó anteriormente:

Parámetro	**Valor**
Entorno local (estación de trabajo)	
Lenguaje R	R version 4.4.2 (2024-10-31 ucrt)
Entorno IDE	RStudio 2025.05.0 (Build 496)
Sistema operativo	Windows 10 (64 bits)
Herramienta de generación de informes	Quarto 1.6.42
Licencia	GNU Affero General Public License v3

Parámetro	Valor
Entorno remoto (servidor para la ejecución de scripts)	
Lenguaje R	R version 4.5.0 (2025-04-11) — *"How About a Twenty-Six"*
Sistema operativo	Linux (x86_64-pc-linux-gnu)
Acceso remoto	vía terminal (usuario `naysha@cremaet`)
Licencia	GNU General Public License v3

Tabla 5.2: Entornos utilizados para el desarrollo y ejecución

5.2. Carga de datasets

Como se describió anteriormente, este proceso es el encargado de cargar la información de los *datasets* para poder trabajar y realizar los experimentos. Para este propósito se crea una lista `datasets_info` que contiene la información de cada *datasets*, con la información de los `headers` y el tipo de separador.

```
datasets_info <- list(
  "1_seeds_dataset.txt" = list(header = FALSE, sep = ""),
  "2_Wholesale customers data.csv" = list(header = TRUE, sep = ","),
  "3_data_banknote_authentication.txt" = list(header = FALSE, sep = ","),
  "4_google_review_ratings.csv" = list(header = TRUE, sep = ","),
  "5_EastWestAirlines.csv" = list(header = TRUE, sep = ","),
  "6_genres_v2.csv" = list(header = TRUE, sep = ","),
  "7_power_plant_data.csv" = list(header = TRUE, sep = ","),
  "8_housing.csv" = list(header = TRUE, sep = ","),
  "9_heart_failure_clinical_records_dataset.csv" = list(header = TRUE, sep
     = ","),
  "10_smoke_detection_iot.csv" = list(header = TRUE, sep = ","),
  "11_winequality-white.csv" = list(header = TRUE, sep = ";"),
  "12_Dry_Bean_Dataset.arff" = list(header = NA, sep = ""),
  "13_HTRU.csv" = list(header = FALSE, sep = ","),
  "14_Concrete_Data.xls" = list(header = TRUE, sep = ","),
  "15_gt_2011.csv" = list(header = TRUE, sep = ","),
  "16_wdbc.data" = list(header = FALSE, sep = ","),
  "17_Rice_Cammeo_Osmancik.arff" = list(header = NA, sep = ""),
  "18_CC GENERAL.csv" = list(header = TRUE, sep = ","),
  "19_yeast.data" = list(header = FALSE, sep = ""),
  "20_diabetes.csv" = list(header = TRUE, sep = ",")
)
```

Listing 5.1: Lista `datasets_info` con informacion de cada `dataset`

En Listing 5.1 la forma de la lista es el nombre del *dataset* y la lista con la información, como *header* indica si el archivo de carga posee títulos para cada *feature* del archivo de carga, esta información se comprueba de forma manual, con respecto a los separadores `sep` se usa en una función auxiliar que extrae el separador de cada archivo de carga.

La función de `detectar_separador` revisa las primeras líneas de los archivos de carga ubicados en una dirección que se pasa como parámetro, se crea un vector con

los separadores más comunes que se usan, después se realiza una búsqueda en las primeras 5 líneas para ver cual `sep` tiene más apariciones. La respuesta es informativa y con ella podemos modificar la información de `datasets_info`. Realizando un bucle con la ruta y los nombres de los archivos de carga se obtienen los separadores correspondientes a cada uno como se ve en Figura 5.1, el primer archivo de carga es de tipo `.txt` y no posee separador, a diferencia del segundo archivo de carga que es un `.csv` y tiene como separador la coma.

```
El dataset 1_seeds_dataset.txt  tiene el separador:
El dataset 2_Wholesale customers data.csv  tiene el separador:  ,
El dataset 3_data_banknote_authentication.txt  tiene el separador:  ,
El dataset 4_google_review_ratings.csv  tiene el separador:  ,
El dataset 5_EastWestAirlines.csv  tiene el separador:  ,
El dataset 6_genres_v2.csv  tiene el separador:  ,
El dataset 7_power_plant_data.csv  tiene el separador:  ,
El dataset 8_housing.csv  tiene el separador:  ,
El dataset 9_heart_failure_clinical_records_dataset.csv  tiene el separador:  ,
El dataset 10_smoke_detection_iot.csv  tiene el separador:  ,
El dataset 11_winequality-white.csv  tiene el separador:  ;
El dataset 12_Dry_Bean_Dataset.arff  tiene el separador:
El dataset 13_HTRU.csv  tiene el separador:  ,
```

Figura 5.1: Resultados del bucle usando la función `detectar_separador` para encontrar separadores por archivo de carga

Con esta información actualizada, el siguiente paso es la carga de los archivos, para esto se usa un bucle que recorre cada elemento de `datasets_info` y usando la función `cargar_archivos` se irán cargando uno a uno cada archivo según su extensión, si ocurre alguna excepción se procede a mostrar un mensaje informativo. Los parámetros que se usan son `stringsAsFactors` para evitar que las columnas se conviertan en factores, `fill` para completar las instancias vacías con `NA`, `strip.white` para eliminar espacios en blanco de adelante y atrás de cada campo, `quote` para evitar que algún separador en este caso las comillas afecten el *parseo*, solo se vio en algunos formatos que la cabecera no son válidos, para esto se usa `check.names` y un parámetro que no usamos con este conjunto de datos pero fue útil para otros *datasets* es `skip` que ayuda a cargar un archivo desde una línea especifica. Para cargar otros archivos como como: `.csv`, `.arff` o `.xls`, se utiliza funciones especiales como `farff::readARFF(file_path)` o `readxl::read_excel(file_path)` que vienen de paquetes instalados con el mismo nombre.

Con estos procedimientos ya se tiene la lista `datasets` que contiene 20 *datasets* que fueron cargados de archivos de distintos formatos. De este punto en adelante se trabajará con esta lista.

5.3. Limpieza de datos

En esta sección abordamos la limpieza de elementos de la colección y el preprocesamiento básico. Debido a que cada elemento de la lista `datasets` es un `dataframe` con sus propias características no es posible crear una función iterativa para eliminar datos que no son útiles para el *clustering*, por este motivo se procedió a analizar cada uno. El preprocesamiento realizado consiste en asignar nombres más legibles a las columnas, sobre todo para aquellos *dataframes* que no poseen `headers`, eliminación de columnas con poca variabilidad debido a que desembocan en atributos categóricos o con información que no es relevante, otro procedimiento usado es la fusión de varios archivos de carga a un `dataframe` porque este se encontró con varios archivos

de carga, como ejemplo el `dataframe` de vinos (data_11) que tenía dos archivos para vinos blancos y vinos tintos; finalmente otro procedimiento es la eliminación de columnas con varios campos faltantes que si no se soluciona en este paso puede llevar a un mal manejo en posteriores pasos, causando que se eliminen instancias útiles.

Nro	attr. Clase	attr. Id	attr. Categórico	attr. sin sin Aporte	No Headers	Vacías/ NA	Fusión
data_1	x				x		
data_2			x				
data_3	x				x		
data_4		x			x	x	
data_5		x	x				
data_6	x	x	x			x	
data_7							
data_8			x				
data_9	x						
data_10	x	x		x			
data_11	x						x
data_12	x						
data_13	x				x		
data_14	x				x		
data_15	x			x			x
data_16	x	x			x		
data_17	x						
data_18	x		x			x	
data_19	x		x	x			
data_20	x						

Tabla 5.3: Procesamiento aplicado por tipo de operación en cada dataframe

Como se puede ver en la Tabla 5.3 cada `dataframe` tuvo un procesamiento distinto que se ira detallando. La eliminación del atributo de clase (attr. Clase) se da en *dataframes* supervisados, en su mayoría de clasificación, donde el atributo de clase representa la etiqueta asignada, casi a todos los *dataframes* de la lista se les elimino este atributo con excepción del *dataframe* número 2, 5 y 7. Con respecto a los atributos categóricos se evalúa su distribución y la cantidad de valores únicos, este es el caso de algunos *dataframes* como el `data_2` que posee dos categóricos: `región` que posee varios valores y se opta por quitarlo por presentar una distribución desbalanceada (3=316; 1=77; 2=47), y `channel` que tiene dos valores más balanceados, por lo que se conserva (Listing 5.2). De manera similar si se tienen atributos de tipo `Id` o que posean muchas instancias vacías son descartados.

```
1         id_name=paste0(default_name,"2")
2         ##analizar columnas
3         auditar_columnas(datasets[[id_name]])
4         datasets[[id_name]] <- datasets[[id_name]][, !(names(datasets[[id_
              name]]) %in% c("Region"))]
5         #refactorizar channel a binario
```

```
6         datasets[[id_name]]$Channel <- as.numeric(datasets[[id_name]]$
              Channel == 2)
7         head(datasets[[id_name]])
```

Listing 5.2: Fragmento de codigo para quitar columnas categoricas

Otra particularidad que se encontró al recolectar *dataset* es que algunos no tienen `headers` o títulos, por tanto, de forma manual se agregan los títulos según la documentación de los *datasets*, para hacer más legible la lectura y análisis posteriores. El agregar los títulos para cada columna, es útil porque ayuda a identificar columnas que no son útiles, como para el `data_4` los atributos denominados `X` y `User` son `ids` y etiquetas de salida, ademas `X` tiene todos sus valores en `null`.

Un caso particular es el *dataframe* `data_9` que contiene algunas variables binarias de tipo numérico para los atributos `anemia, diabetes, high_blood_pressure, sex` y `smoking`. En este caso no se eliminan estos atributos en su lugar tienen un trato especial, no se les agregara ruido, pero sí participan en del *clustering*.

Description: df [6 × 12]

	age <dbl>	anaemia <int>	creatinine_phosphokinase <int>	diabetes <int>	ejection_fraction <int>	high_blood_pressure <int>	platelets <dbl>
1	75	0	582	0	20	1	265000
2	55	0	7861	0	38	0	263358
3	65	0	146	0	20	0	162000
4	50	1	111	0	20	0	210000
5	65	1	160	1	20	0	327000
6	90	1	47	0	40	1	204000

Figura 5.2: Fragmento del `dataframe data_9` que contiene algunos binarios

El *dataframe* data_10, que es una colección de datos de un detector de humo, posee algunos datos temporales `UTC`, contadores `CNT` y etiquetas. En este caso, se eliminan estos datos que no aportan información sobre el estado físico del sistema, sino que crearían nuevos sesgos artificiales, agrupaciones que se basan en el tiempo en lugar de las características del sistema. Después de eliminar estos atributos también debe eliminarse las instancias repetidas con los mismos datos y evitar que datos duplicados alteren los centroides al momento de aplicar el *clustering*.

```
1         id_name=paste0(default_name,"10")
2         cols_quitar <- c("", "Fire Alarm", "UTC", "CNT")
3         datasets[[id_name]] <- datasets[[id_name]][, !(names(datasets[[id_
              name]]) %in% cols_quitar)]
4         head(datasets[[id_name]])
```

Listing 5.3: Fragmento de codigo para limpiar el *dataframe* data_10

El siguiente *dataframe* es el *Calidad de vino*, que se presenta en dos partes, características del vino tinto y del blanco, para poder trabajar con ambos se realizó una unión de ambos *dataframes* y se eliminó el atributo clase, quedando así el *dataframe* data_11.

El *dataset* de *Mediciones de emisiones de CO y NOX* contiene varios subconjuntos agrupados por *year* y los atributos que se obtienen a partir de las condiciones ambientales, CO, NOX y TEY. Estos atributos se eliminan ya que, de usarse para el

clustering, podrían hacer que se crearan grupos en base a atributos creados para la predicción. También se realiza la unión de los *dataset* de los distintos *year*.

El *dataset Yeast* tiene atributos de tipo identificador, textual y categóricos. Este *dataset* tiene la variable `erl` que representa la presencia o no de una señal (HDEL), mientras que `pox` indica una señal con tres tipos de valores. Esta información se obtiene de una función auxiliar `auditar_columnas` que describe los atributos del *dataset* como se ve en Figura 5.3.

columna <chr>	tipo <chr>	unicos <int>	es_entero <lgl>	muestra <chr>
Sequence_Name	character	1462	FALSE	ADT1_YEAST, ADT2_YEAST, ADT3_YEAST, AAR2_YEAST, AATM_YEAST
mcg	numeric	81	FALSE	0.58, 0.43, 0.64, 0.42, 0.51
gvh	numeric	79	FALSE	0.61, 0.67, 0.62, 0.44, 0.4
alm	numeric	53	FALSE	0.47, 0.48, 0.49, 0.57, 0.56
mit	numeric	78	FALSE	0.13, 0.27, 0.15, 0.54, 0.17
erl	numeric	2	FALSE	0.5, 1
pox	numeric	3	FALSE	0, 0.5, 0.83
vac	numeric	48	FALSE	0.48, 0.53, 0.54, 0.49, 0.58
nuc	numeric	68	FALSE	0.22, 0.34, 0.3, 0.27, 0.29
localization_site	character	10	FALSE	MIT, NUC, CYT, ME1, EXC

1-10 of 10 rows | 2-6 of 5 columns

Figura 5.3: Análisis del `dataframe data_19` que muestras los valores únicos y otras características

Como ya se ha explicado, para este trabajo solo se mantienen los categóricos de hasta dos valores. Por tal motivo se eliminan los campos: `Sequence_Name, localization_site y pox` mientras que `erl` se convierte en un binario como se ve en Listing 5.4 y se obtiene el *dataset* `data_19`.

```
1
2     id_name=paste0(default_name,"19")
3     colnames(datasets[[id_name]]) <-  c("Sequence_Name",  "mcg", "gvh", "
          alm",  "mit", "erl", "pox", "vac", "nuc", "localization_site")
4     ##analizar columnas
5     auditar_columnas(datasets[[id_name]])
6     ## convertor a binarios los categoricos
7     datasets[[id_name]]$erl <- ifelse(datasets[[id_name]]$erl == 1, 1, 0)
8     #quitar columnas no necesarias
9     datasets[[id_name]] <- datasets[[id_name]][, !(names(datasets[[id_name
          ]]) %in% c("Sequence_Name", "localization_site", "pox"))]
```

Listing 5.4: Fragmento de codigo para limpiar el *dataframe* data_19 y convertir `erl` a binario

El ultimo *dataset* y otros que no se mencionan a detalle tienen un preprocesamiento muy parecido o más simple de lo descrito anteriormente, sin embargo, para más detalles se da una tabla con las modificaciones en el Anexo7.

5.4. Agregación de ruido

Para el siguiente paso, una parte fundamental es la agregación de ruido a cada uno de los *dataframes* de la colección que se limpió previamente. Primero se crea una secuencia con los 7 niveles de ruido.

```r
agregar_ruido <- function(df, nombre_dataset, noise_sd = 0.1, columnas_
    binarias) {
  df_ruido <- df
  #identificar columnas
  columnas_numericas <- names(df)[sapply(df, is.numeric)]
  columnas_excluir <- columnas_binarias[[nombre_dataset]]
  columnas_validas <- setdiff(columnas_numericas, columnas_excluir)
  =

  # ruido en columnas numericas
  df_ruido[, columnas_validas] <- df[, columnas_validas] +
    matrix(
      rnorm(length(columnas_validas) * nrow(df), mean = 0, sd = noise_
          sd),
      nrow = nrow(df), ncol = length(columnas_validas)
    )

  return(df_ruido)
}
```

Listing 5.5: Función de agregar ruido a la colección de dataframes

En Listing 5.5 la función recibe como parámetros el nombre de *dataframe* para identificarlo, el ruido que se le aplica y el conjunto de atributos que son binarios.

1. Se realiza una identificación de las columnas para un *dataframe* determinado, usando la función `sapply` detectamos las columnas numéricas. Luego usando `setdiff` se realiza la exclusión de las columnas binarias.

2. Se crea y suma el ruido gaussiano a las `columnas_validas`, en este punto se aplica ruido solo a estas columnas.

Para la aplicación de ruido como tal, se obtiene el número de celdas a perturbar con `length(columnas_validas) * nrow(df)`, que son columnas por filas del dataframe, este dato es n. La media `mean` o según la fórmula μ corresponde a cero porque en este experimento no se introducirá sesgo. La varianza que controla el nivel de ruido σ dependerá del valor del parámetro `noise_sd`, estos tres parámetros se usaran para la función `rnorm`, que generara números pseudoaleatorios con distribución gaussiana. Después de obtener el vector con el ruido, se pasa a la función `matrix`, que dará forma a este vector convirtiéndolo en una matriz y se sumará celda a celda con el *dataframe* original (sin ruido), obteniendo un *dataframe* perturbado con cierto nivel de ruido `noise_sd`.

Para la reproducibilidad del experimento, se utiliza $104729L$ como base para la semilla `seed`. Para crear una nueva colección de *dataframes* con cada nivel del ruido, se realiza una doble iteración usando la función de `lapply`.

1. Se realiza una iteración por cada nivel de la secuencia generada de ruido. La función `seq_along(niveles_activos)` genera $1, 2, 3...N$

2. Por cada nivel se estable una semilla, usando la base $104729L$ más un espaciado de un número primo 73, este espaciado es para generar una semilla única en cada nivel y evitar patrones correlativos.

3. El segundo bucle recorre cada `dataframe` de la colección `datasets`

```
datasets_ruido <- lapply(seq_along(niveles_activos), function(i){
  set.seed(base_seed + i*73L)    # semilla por nivel con espaciado primo
  nivel <- niveles_activos[i]

  lapply(names(datasets), function(nombre) {
    agregar_ruido(datasets[[nombre]], nombre_dataset = nombre, noise_sd
        = nivel, columnas_binarias = columnas_binarias)
  }) |> setNames(names(datasets))

})
```

Listing 5.6: Fragmeto para la generacion de la coleccion de datframes con ruido

Al finalizar la ejecución de Listing 5.6, se tendrá una nueva colección organizada por los niveles de ruido y cada uno tendrá una versión alterada del cada uno de los *dataframes* de la colección, como se aprecia en Figura 5.4.

Name	Type	Value
datasets_ruido	list [7]	List of length 7
0.2	list [20]	List of length 20
data_1	list [210 x 7] (S3: data.frame)	A data.frame with 210 rows and 7 columns
data_2	list [440 x 7] (S3: data.frame)	A data.frame with 440 rows and 7 columns
data_3	list [1348 x 4] (S3: data.frame)	A data.frame with 1348 rows and 4 columns
data_4	list [5451 x 24] (S3: data.frame)	A data.frame with 5451 rows and 24 columns
data_5	list [3998 x 7] (S3: data.frame)	A data.frame with 3998 rows and 7 columns
data_6	list [35096 x 11] (S3: data.frame)	A data.frame with 35096 rows and 11 columns
data_7	list [6396 x 4] (S3: data.frame)	A data.frame with 6396 rows and 4 columns
data_8	list [20433 x 9] (S3: data.frame)	A data.frame with 20433 rows and 9 columns
data_9	list [299 x 12] (S3: data.frame)	A data.frame with 299 rows and 12 columns
data_10	list [62628 x 12] (S3: data.frame)	A data.frame with 62628 rows and 12 columns
data_11	list [5318 x 11] (S3: data.frame)	A data.frame with 5318 rows and 11 columns
data_12	list [13543 x 16] (S3: data.frame)	A data.frame with 13543 rows and 16 columns
data_13	list [17898 x 8] (S3: data.frame)	A data.frame with 17898 rows and 8 columns
data_14	list [996 x 8] (S3: tbl_df, tbl, data.	A tibble with 996 rows and 8 columns
data_15	list [36726 x 8] (S3: data.frame)	A data.frame with 36726 rows and 8 columns
data_16	list [569 x 30] (S3: data.frame)	A data.frame with 569 rows and 30 columns
data_17	list [3810 x 7] (S3: data.frame)	A data.frame with 3810 rows and 7 columns
data_18	list [8636 x 14] (S3: data.frame)	A data.frame with 8636 rows and 14 columns
data_19	list [1453 x 7] (S3: data.frame)	A data.frame with 1453 rows and 7 columns
data_20	list [768 x 8] (S3: data.frame)	A data.frame with 768 rows and 8 columns
0.4	list [20]	List of length 20
0.6	list [20]	List of length 20
0.8	list [20]	List of length 20

Figura 5.4: Captura de la estructura de la colección de *dataframes* con ruido

5.5. Escalamiento de datos

Después de tener la colección de *dataframes* sin ruido y la colección de *dataframes* agrupados por la perturbación de diferentes niveles de ruido tenemos un total de 140 *dataframes* perturbados. Considerando nuestros 20 *dataframes* originales donde se tiene un promedio de 11 columnas con una determinada escala para cada una, para poder continuar con los próximos pasos de la metodología se debe realizar un escalamiento, para esto se usa la función `scale`.
Al realizar el escalamiento por defecto, se centran los datos en 0 y ajustar la desviación a 1, el ruido introducido a modificado los datos y al realizar el escalamiento, `scale`), puede absorber el ruido la nueva media y σ cambiaran y las diferencias de los ruidos se esconderán. En esta metodología el ruido se inyecta antes del escalado, lo que hace que su efecto quede normalizado respecto a la variabilidad total del atributo tras la introducción de ruido. Aunque esto atenúa las diferencias absolutas entre niveles de ruido, se mantiene la comparación relativa entre algoritmos bajo un mismo preprocesado, que es el objetivo de la evaluación de robustez. Para realizar el escaldo se excluyen las columnas binarias declaradas por cada *dataset*, manteniéndose sin transformación. El resto de las columnas numéricas se estandarizan para que la distancia euclidiana trate todas las dimensiones en la misma escala.

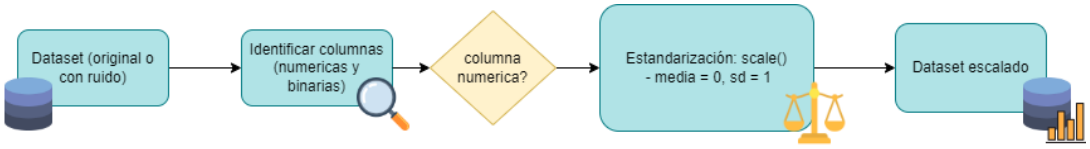

Figura 5.5: Diagrama para el flujo de escalamiento

5.6. Calculo de k óptimo

En este paso ya tenemos la colección de *dataframes* con y sin ruido listos para el *clustering*, sin embargo, se necesita calcular aun los valores de k óptimos para cada algoritmo: `k-means` y `jerárquico`.

Para la obtención del valor de k utilizamos la función `NbClust`, que consta de varios parámetros, para la distancia se usa la distancia euclidiana, el rango de valores que puede tomar k va de entre 2 a 6, y dependiendo del algoritmo que se usa el parámetro `method` varia. Para `k-means` se usa el atributo del mismo nombre mientras que para el jerárquico se usa `ward.D2`. La funcionalidad de `NbClust` es usar un algoritmo `method` y aplicar distintos índices, el parámetro por defecto de `index` es `all` e involucra 26 índices que escogen un valor k óptimo, entre los valores de k de cada índice se escoge el que tenga mayores votos, este será el k óptimo para el experimento.

Como la complejidad del proceso involucra a cada *dataframe* con un tamaño distinto se crea un script para ejecutar en el servidor `cremaet`, para mejorar el proceso y acortar tiempos se realiza un muestreo de datos para los *dataset* más grandes como: `Dataset of Songs in Spotify`, `California Housing Prices`, `Detection Datase` y `Gas Turbine CO and NOx`, para las primeras pruebas se establece un límite de 3000 filas, se fija una semilla para la reproducción de pruebas y si los *dataframes* superan el límite se genera un muestreo aleatorio del *dataframe* para escoger el k óptimo.

5.7. Clustering

Para el siguiente paso, aplicación de *clustering* usaremos dos scripts: `calcular_clustering_sin_ruido` y `calcular_clustering_con_ruido` que tienen el mismo comportamiento, ambos usan el paquete `stats` un paquete estadístico y las funciones `k-means` para el *clustering* del mismo nombre y para el *clustering* jerárquico se hace uso de `dist`, `hclust` y `cutree`.

La configuración para el *clustering* usando `k-means` usa el parámetro `nstart`, el número de inicializaciones distintas o puntos de partida para realizar la prueba y encontrar buenos centros iniciales. Como los *dataframes* tienen distinto tamaño y distinta dificultad al momento de realizar la separación de *clusters*, el valor de `nstart` varia:

$$n_{\text{start}} = \begin{cases} 50 \text{ si difícil o } p > 50,\ 25 \text{ en otro caso,} & n \leq 10\,000 \\ 75 \text{ si difícil o } p > 50,\ 50 \text{ en otro caso,} & 10\,000 < n \leq 30\,000 \\ 100 \text{ si difícil o } p > 50,\ 75 \text{ en otro caso,} & n > 30\,000 \end{cases} \tag{5.1}$$

El siguiente parámetro es el algoritmo que se usa, por defecto se usa el algoritmo de `Hartigan-Wong`, en caso de que la cantidad de datos sea mayor a 20000 y la separación de *clustering* sea difícil o el número de columnas mayor a 50 se usara el algoritmo de `Lloyd`. El parámetro de `iter.max`, el número de pasos que se permite para afinar en cada nueva inicialización de centros, el valor de `iter.max` será 200L si la separación es difícil y el número de columnas es mayor 50 caso contrario el `iter.max` será de 100L.
Para el *cluster* jerárquico se usa varias funciones, `dist`, devuelve la matriz inferior de la distancia de todos los puntos entre sí, el parámetro que se usa es `method` para determinar que la distancia que se usara será la euclidiana, esto alimentara al `hclust` que aplica un método de enlace sobre matriz y crea un dendrograma, el método que se usa en esta función es `ward.D2` para minimiza la varianza intra-cluster. el último paso para el *cluster* jerárquico es realizar el corte, se usa para esto la función `cutree` que realiza el corte a la altura donde se produce k *clusters* y devuelve un vector de asignaciones. Estos scripts devolverán respectivamente los datos almacenados en `resultados_sin_ruido.rds` y `resultados_con_ruido.rds`.

5.8. Métricas de Evaluación

Después de tener los vectores de asignaciones de `k-means` y `k-means$cluster` se pasa a comparar el agrupamiento obtenido con los datos sin suido con cada agrupamiento creado con los datos con ruido. Como el proceso puede ser pesado se creó un script; `evaluate_clustering.R` que se ejecuta en un servidor externo, para que este script se ejecute adecuadamente se necesita los datos extraídos y almacenados en pasos anteriores, `resultados_sin_ruido.rds` y `resultados_con_ruido.rds` almacenados en la ruta de `../data`.
Para obtener la métrica de pares rotos se implementa un doble bucle por todas las parejas, una pareja será un par de índices (i, j) con $i < j$ pero evita duplicados y $i == j$. Si en `labels1` están en el mismo *cluster* se cuenta en dentro de `count_total`

y se realiza una comparación, si en `labels2` ya no se encuentran juntos se incrementa el contador de pareja disrumpida `count_disrupted`, la proporción que se busca es $count_{disrupted}/count_{total}$.

```
pairwise_disruption <- function(labels1, labels2) {
  n <- length(labels1)
  count_total <- 0
  count_disrupted <- 0
  for (i in 1:(n - 1)) {
    for (j in (i + 1):n) {
      if (labels1[i] == labels1[j]) {
        count_total <- count_total + 1
        if (labels2[i] != labels2[j]) {
          count_disrupted <- count_disrupted + 1
        }
      }
    }
  }
  if (count_total == 0) return(NA)
  return(count_disrupted / count_total)
}
```

Listing 5.7: Funcion para calcular la medida de pares rotos

Para el resto de las métricas se utiliza la librería de `mclust` con la función de `adjustedRandIndex` para el ARI y el paquete `aricode` con la función del mismo nombre para `NMI`. Para poder evaluar los resultados se crea un bucle que itera sobre cada nivel del ruido, dentro de cada nivel de ruido recorre cada *dataset* que fue perturbado y finalmente el ultimo bucle recorre los resultados de `k-means` y `hclust`, dentro de estos 3 bucles se asignan valores al conjunto de resultados del *dataset* sin ruido y al conjunto de *dataset* con ruido actual.

Capítulo 6

Análisis de Resultados

En este capítulo realizaremos el análisis de resultados de las métricas ARI, NMI y PDP que se obtuvieron en el capítulo anterior. Para el análisis de cada métrica se hará uso de gráficas para su comparación y análisis.

6.1. Panorama para ARI

Para la evaluación para la métrica ARI, véase Figura 6.1, donde se muestra como el valor de la métrica va descendiendo conforme el ruido va incrementándose, conforme a las primeras estimaciones que se hicieron y en algunos casos se mantiene estable. De manera general se puede observar que los resultados son muy variados, pero aun así muestran ciertas similitudes que analizaremos a continuación:

6.1.1. Casos de coincidencia entre algoritmos

Observando la gráfica para la métrica ARI, Figura 6.1, se puede apreciar que las curvas de evolución son muy similares en los *datasets* 1, 2, 10, 11, 16 y 19. En particular, en `dataset_10` y `dataset_2` ambos algoritmos de *clustering* se mantienen estables a pesar del aumento del ruido. Esto puede ser un indicativo que los *clusters* se encuentren bien separados. Al momento de evaluar las particiones con ruido vs. sin ruido, los pares de elementos que se encuentran en una partición están en su mayoría también juntos en la otra. Por lo tanto, en estos dos datasets ambos algoritmos son robustos. En este tipo de caso, se observa también que el algoritmo de `k-means` supera ligeramente a `hclust`. En casos como el `dataset_19` ambos algoritmos se mantienen con un ARI muy bajo, lo que indica que ambos algoritmos no pueden formar *clusters* similares al original aun cuando se les introduce una cantidad pequeña de ruido.

6.1.2. Casos con divergencia marcada entre algoritmos

Otros casos donde se observa que los algoritmos muestran comportamientos muy diferentes son los resultados de los *datasets* 5, 9, 12 y 20. En estos casos vemos que `k-means` se mantiene más robusto en la mayoría de los casos con respecto a `hclust`, el cual con la mínima cantidad de ruido pierde estabilidad y su valor ARI cae drásticamente. Sin embargo, es en el `dataset_12` en el que es `k-means` el que cae

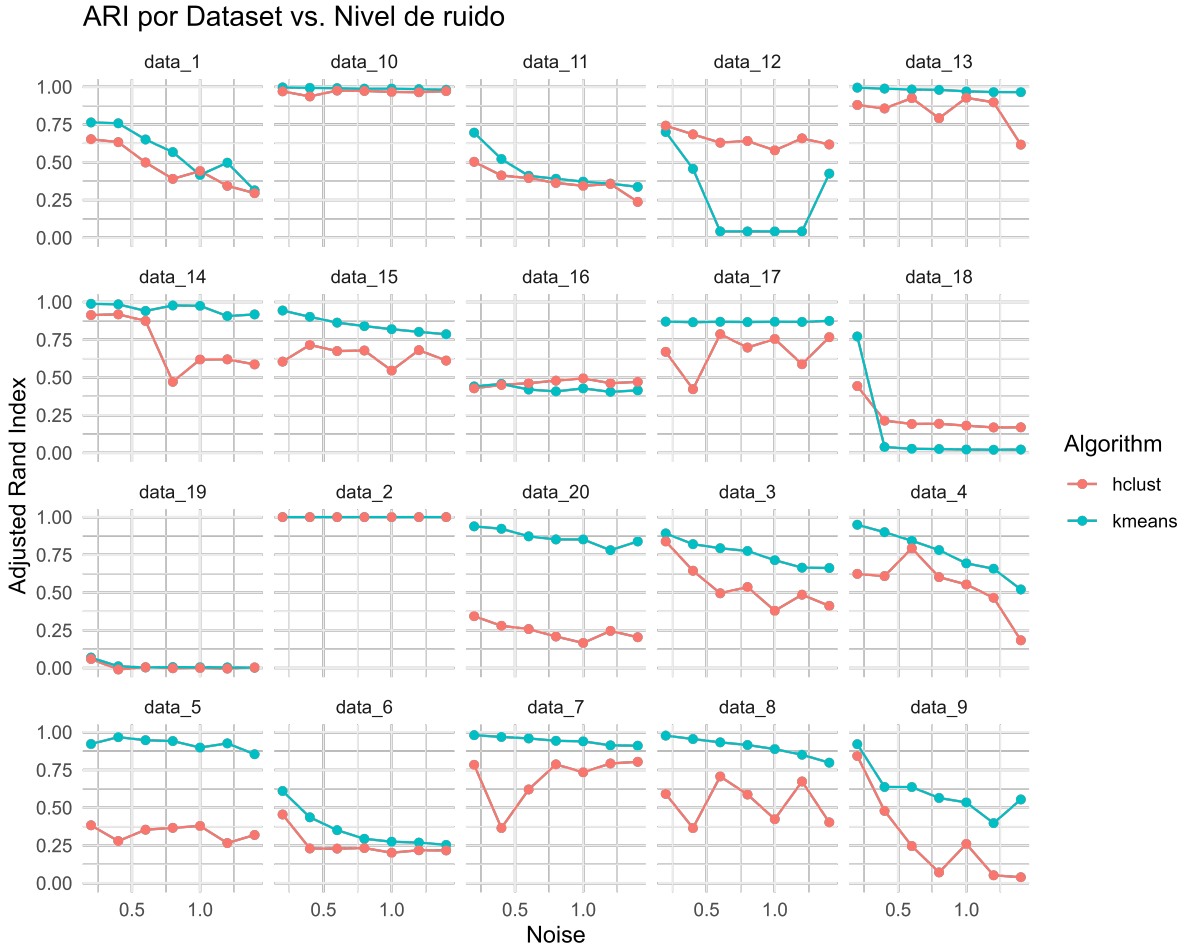

Figura 6.1: Gráficas de la métrica ARI para los 20 *datasets*.

drásticamente en comparación con `hclust`. Para analizar este caso en más detalle véase Figura 6.2 al Figura 6.7, donde se observa la formación de *clusters* y cómo estos van variando conforme se agrega ruido. Para un ruido de 0.6 se ve que el *cluster* rojo crece demasiado, mientras que el azul se reduce, la densidad de los *cluster* se desbalancea, la frontera también se mueve con respecto a la versión original (0 en el eje X) hasta llegar al resultado con ruido 1.4 donde los *clusters* existen pero ya no tienen relación con la versión sin ruido.

6.1.3. Casos con comportamiento estable

Los casos que se podrían considerarse como mostrando un comportamiento normal (la métrica ARI disminuye a medida que aumenta el ruido) poseen una curva decreciente, que se mantiene sin mostrar picos muy marcados, como en los*datasets* 1, 6 y 11. En estos casos sigue siendo mas robusto `k-means` que `hclust`, que es más sensible al ruido.

6.1.4. Casos con comportamiento anormal

Se ve dentro de las gráficas que algunos resultados presentan curvas con mucha variabilidad, lo esperado es que conforme aumente el ruido la métrica ARI baje, pero es raro que aumente si se incrementa el ruido. Los casos para los *datasets* 4, 7, 8, 14 y 17 son muy representativos de este comportamiento. Para el algoritmo de `hclust` se muestra que el ARI posee picos en niveles de ruido intermedios, creando la idea que se recupera la robustez. Sin embargo, este comportamiento se explica por la forma en la que los grupos se construyen con el algoritmo jerárquico aglomerativo, ya que en cada paso del algoritmo se unen los dos grupos con menor distancia de enlazado. Introducir un cierto ruido en los datos, puede hacer que se fusionen grupos con datos diferentes a los unidos cuando no hay ruido o incluso que los grupos se rehagan para valores mayores de ruido. Para observar mejor cómo están formados los `clusters`, véase Figura 6.8 al Figura 6.13, donde se muestra el caso del `dataset_7`. Dos `clusters` (azul y rojo) bastante equilibrados, pero al agregar ruido de 0.4 se produce un cambio drástico haciendo que el `cluster` azul se expanda y reduciendo el `cluster` rojo, se muestra desequilibrio con respecto al caso base. Posteriormente, cuando el ruido se incrementa, se recupera el tamaño y las posiciones de los `clusters` aunque las diferencias son ligeras.

6.1.5. Predominio de k-means y hclust

En la mayoría de los casos `k-means` tiene una robustez mayor a `hclust`, destacan los *datasets* 5, 7, 8, 17, 20 donde `k-means` muestra valores estables y altos, ARI ≈ 1 mientras que `hclust` muestra un valor de ARI que desciende rápidamente llegando a valores medios, ARI ≈ 0.5, con pequeñas cantidades de ruido.
Solo existen dos casos donde `hclust` logra superar a `k-means`, como se mencionó anteriormente el `dataset_12` supera a `k-means` y se mantiene estable, pero en el `dataset_18` aunque es ligeramente superior a `k-means`, ambos algoritmos se ven rápidamente afectados por el ruido, llegando a valores muy bajos de ARI. ≈ 0.

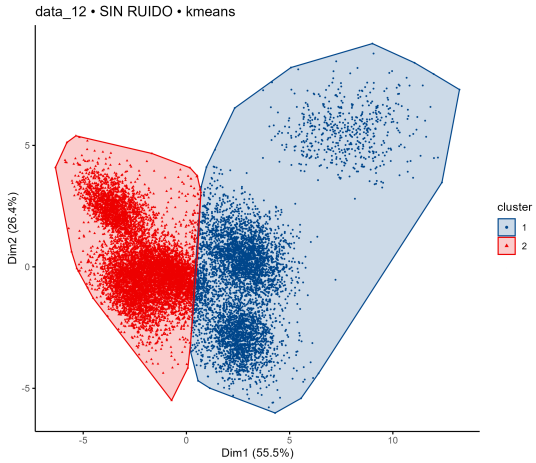

Figura 6.2: Formación de clusters para el dataset_12 sin ruido

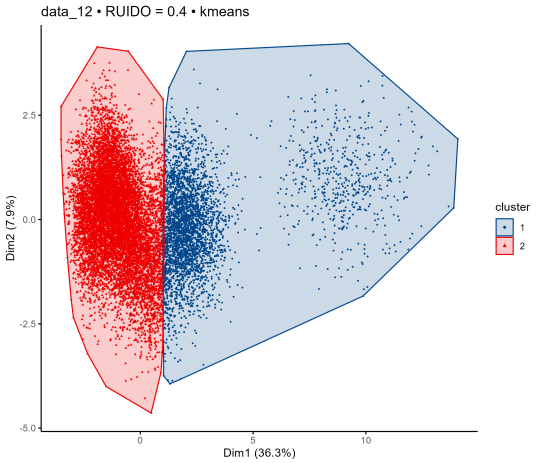

Figura 6.3: Formación de clusters para el dataset_12 con ruido 0.4

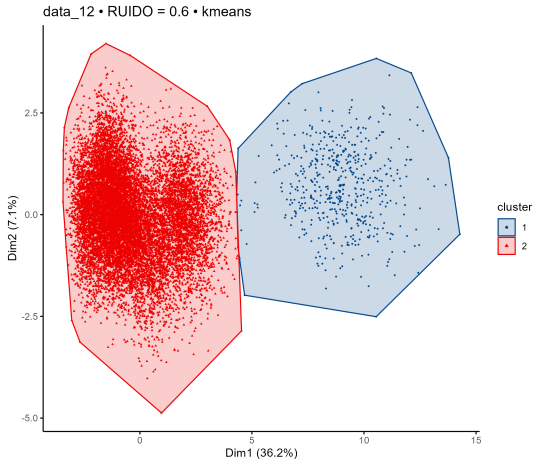

Figura 6.4: Formación de clusters para el dataset_12 con ruido 0.6

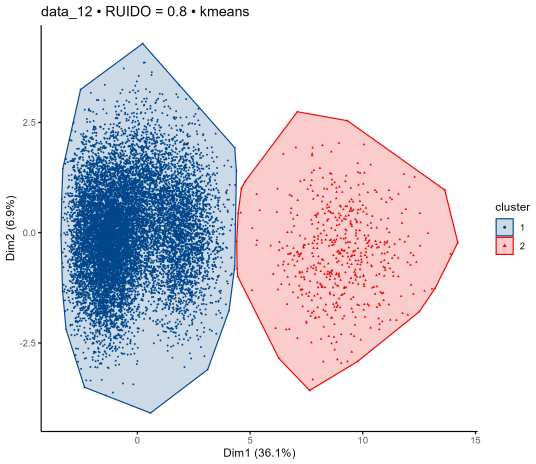

Figura 6.5: Formación de clusters para el dataset_12 con ruido 0.8

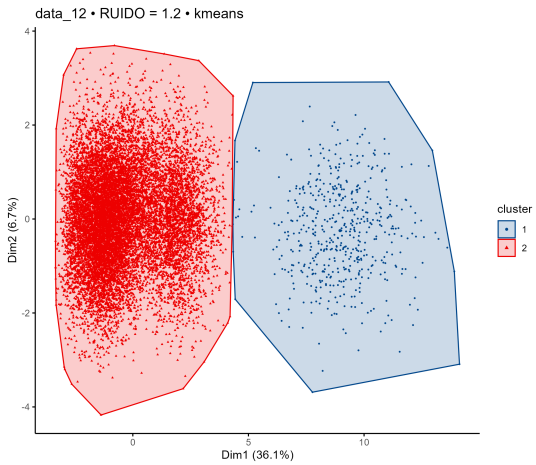

Figura 6.6: Formación de clusters para el dataset_12 con ruido 1.2

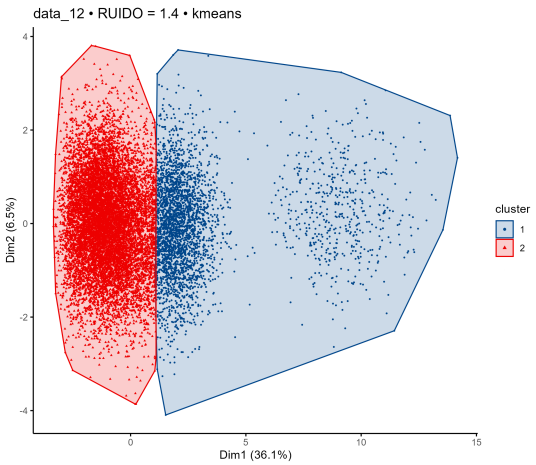

Figura 6.7: Formación de clusters para el dataset_12 con ruido 1.4

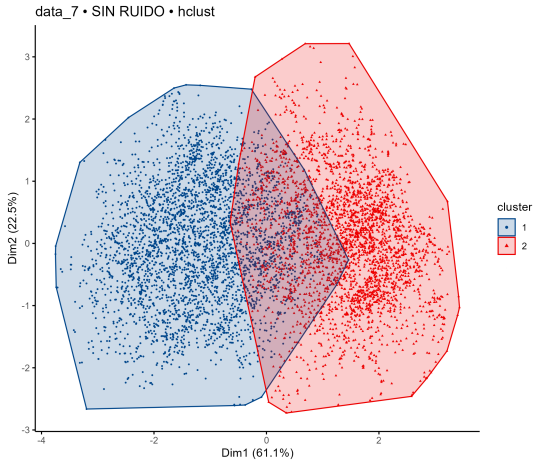

Figura 6.8: Formación de clusters para el dataset_7 sin ruido

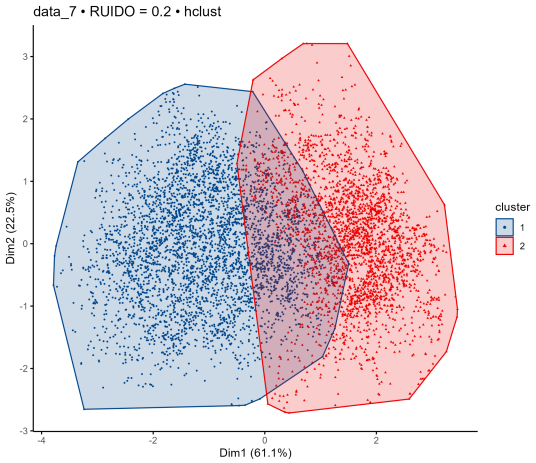

Figura 6.9: Formación de clusters para el dataset_7 con ruido 0.2

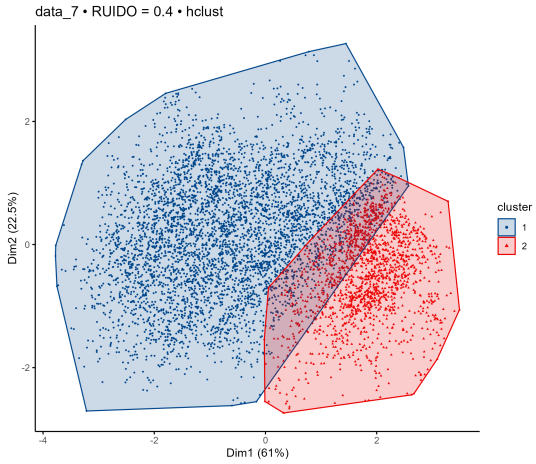

Figura 6.10: Formación de clusters para el dataset_7 con ruido 0.4

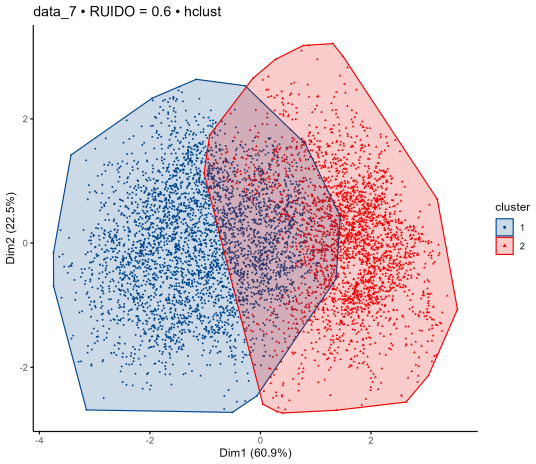

Figura 6.11: Formación de clusters para el dataset_7 con ruido 0.6

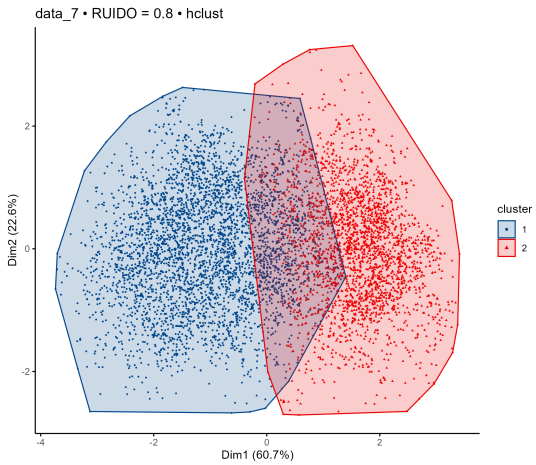

Figura 6.12: Formación de clusters para el dataset_7 con ruido 0.8

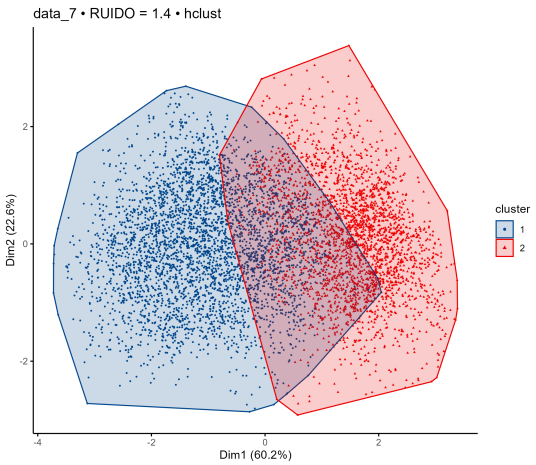

Figura 6.13: Formación de clusters para el dataset_7 con ruido 1.4

6.2. Panorama para NMI

Para la evaluación de la robustez usando la métrica NMI, véase Figura 6.14, donde se puede apreciar que, aunque las curvas no han cambiado drásticamente con respecto a la métrica ARI, sí se ven algunos ligeros cambios. Las curvas en cada resultado son las esperadas, son descendentes conforme el ruido se incrementa y solo en pocos casos se ve una resistencia constante al ruido. Los resultados son muy variados y similares a ARI, por esta razón se especificará solo los casos que difieran o los más resaltantes, siguiendo ciertas similitudes.

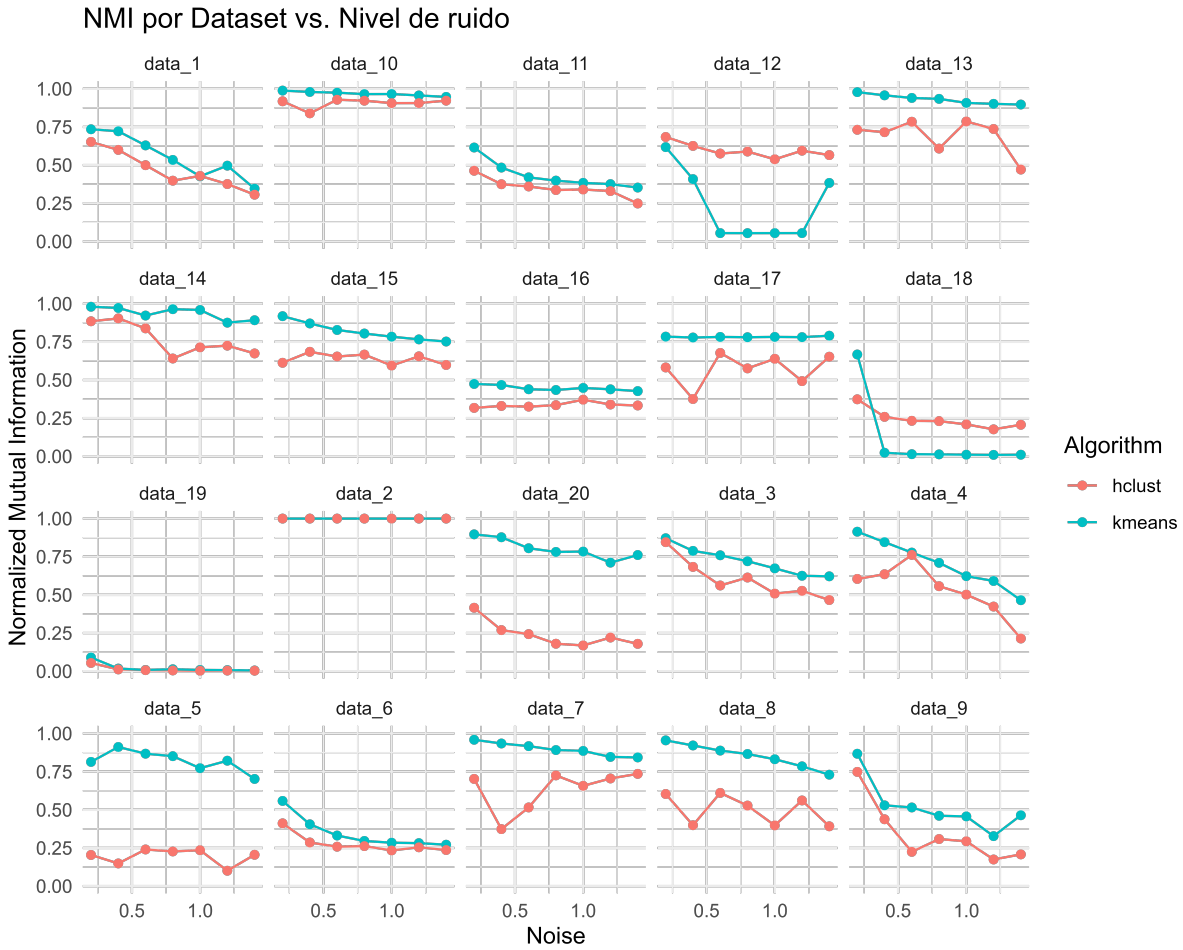

Figura 6.14: Gráficas de la métrica NMI para los 20 *dataframes*.

6.2.1. Casos de coincidencia entre algoritmos

En los *datasets* 10, 11 y 16 el valor de NMI hace que las curvas desciendan ligeramente con respecto a ARI. El resultado de NMI en el `dataset_6` logro hacer coincidir las curvas de los algoritmos `k-means` y `hclust` ligeramente, aunque ambas muestran valores bajos, `k-means` sigue siendo más robusta que `hclust`.

6.2.2. Casos con divergencia marcada entre algoritmos

Los casos con divergencia marcada para NMI y que sean distintos a los resultados mostrados en ARI son pocos. Los *datasets* 5 y 8 tienen una clara diferencia en las curvas, en ambos casos `k-means` sigue siendo más robusto manteniendo valores altos y constantes, NMI ≈ 0.75, a diferencia de `hclust` que desciende muy rápido obteniendo valores bajos (NMI ≈ 0.25).

6.2.3. Casos con comportamiento estable

Los *datasets* con comportamiento normal siguen siendo casi los mismos que en ARI. Aunque en casos como los *datasets* 1, 3 y 4 se observa que las curvas cambiaron ligeramente descendiendo en la escala de valor para NMI. En estos casos, el `k-means` sigue mostrando mejores resultados, con pendientes que bajan ligeramente mientras aumenta el ruido, mientras que `hclust` es más sensible al ruido llegando a valores bajos de NMI ≈ 0.25.

6.2.4. Casos con comportamiento anormal

Para los casos de los *datasets* 7, 8, 13, 14 y 17, las curvas de ambos algoritmos varian ligeramente. El algoritmo `k-means` sigue teniendo mayor resistencia al ruido que `hclust`. En esta métrica se siguen apreciando los picos en niveles de ruido intermedio para `hclust`.

6.2.5. Predominio de k-means y hclust

La resistencia al ruido sigue siendo para `k-means`. En la mayoría de los casos la métrica NMI alcanza valores altos (NMI ≈ 0.75) o se mantiene constante en valores medios (NMI ≈ 0.50). Solo en los casos para los *datasets* 12 y 18 el `hclust` supera a `k-means`, aunque sus valores estén entre media (NMI ≈ 0.50) y valores bajos, llegando a tener un NMI ≈ 0.25.

6.3. Panorama para PDP

Esta métrica presenta un rango distinto a las anteriores, ya que valores cercanos a cero indican mayor robustez frente al ruido. En la Figura 6.15 se observa que, de manera general, `k-means` (azul) tiende a alcanzar valores más bajos de PDP, lo que sugiere una mayor resistencia al ruido en la mayoría de los casos. Por otro lado, `hclust` suele presentar valores más altos y con mayor variabilidad, reflejando mayor sensibilidad ante perturbaciones.

6.3.1. Casos de coincidencia entre algoritmos

En los *datasets* 1, 2, 10, 13, 17 y 19 las curvas muestran trayectorias muy similares para ambos algoritmos. Destacan los `datasets_2` y `10`, que presentan un valor de PDP perfecto, lo cual indica que tanto `k-means` como `hclust` son altamente robustos y prácticamente no se ven afectados por el incremento de ruido. En los demás casos de coincidencia, `k-means` tiende a mantener valores cercanos a cero, lo que confirma su resistencia.

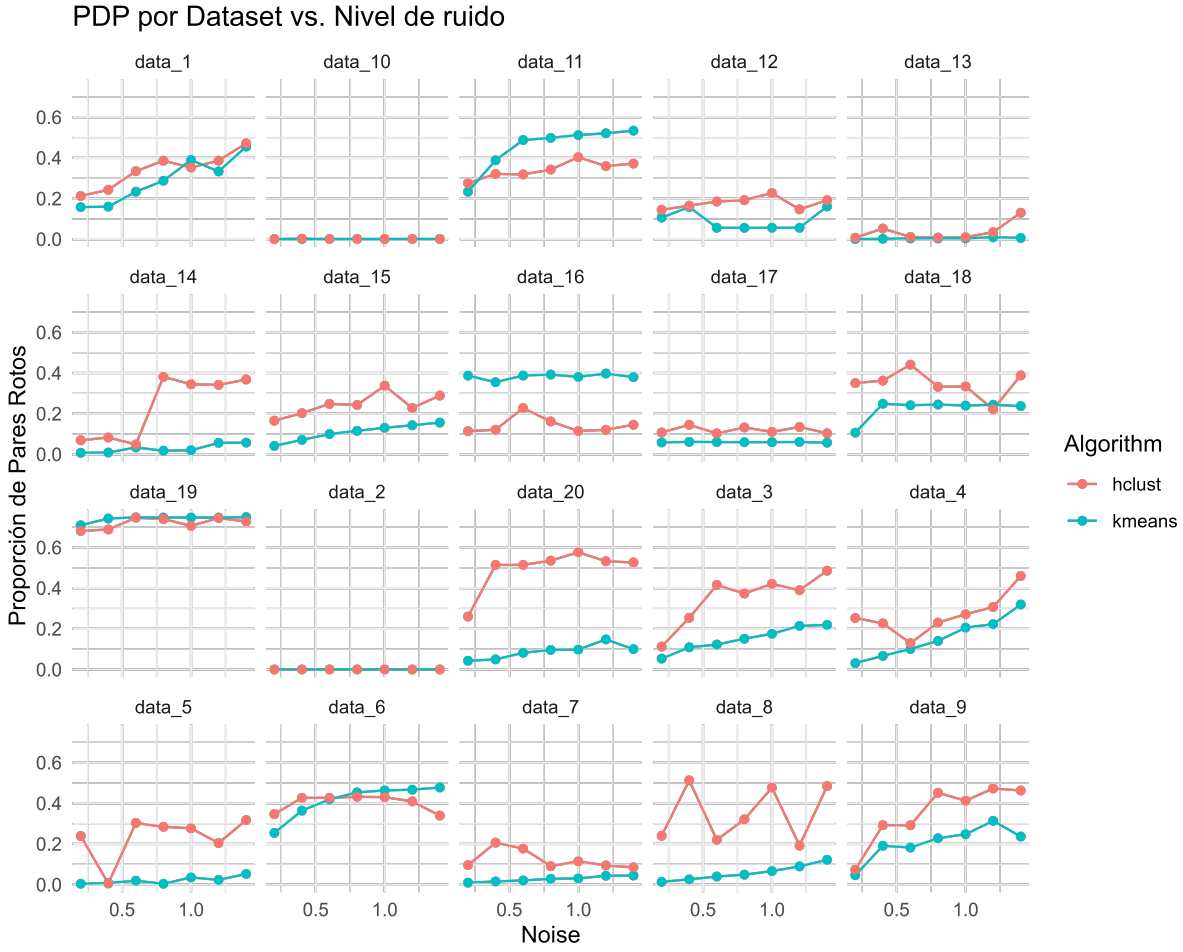

Figura 6.15: Gráficas de la métrica PDP para los 20 *datasets*.

6.3.2. Casos con divergencia marcada entre algoritmos

Los *datasets* 14 y 20 representan los ejemplos más claros de divergencia. En ellos, `hclust` incrementa bruscamente el valor de PDP con la introducción de ruido, mientras que `k-means` conserva valores bajos y estables, evidenciando un mejor desempeño frente a perturbaciones.

6.3.3. Casos con comportamiento anormal

En los *datasets* 5 y 8 se aprecia un comportamiento irregular en `hclust`, que presenta oscilaciones notorias con incrementos y descensos en los valores de PDP conforme se aumenta el ruido. Esto refleja una sensibilidad mayor a pequeñas variaciones, en contraste con la estabilidad relativa de `k-means`.

6.3.4. Casos con comportamiento estable

En varios *datasets*, como el `dataset_3`, se observa un comportamiento esperado: el valor de PDP aumenta progresivamente a medida que se introduce ruido. Este patrón refleja un deterioro gradual de la robustez y es considerado un comportamiento normal en este contexto.

6.3.5. Predominio de k-means y hclust

En la mayoría de los casos, `k-means` mantiene una ventaja clara al presentar valores cercanos a cero y más estables. Por ejemplo, en el `dataset_20`, `k-means` alcanza valores en torno a 0.2 mientras que `hclust` asciende hasta aproximadamente 0.6. Sin embargo, también se identifican casos particulares como el `dataset_16`, donde `hclust` muestra mayor resistencia que `k-means`, manteniéndose cercano a cero mientras este último llega a valores próximos a 0.4.

6.4. Comparativa de las tres métricas: ARI, NMI y PDP

Para poder realizar la comparativa, generamos un gráfico donde se promedia los valores de cada métrica para los 20 *datasets*, agrupando los datos por algoritmo y nivel de ruido, véase Figura 6.16. En el Apéndice 7 se pueden observar los gráficos de niveles de ruido para ver cómo varían las métricas.

La métrica ARI, que define la coincidencia de pares entre las particiones sin ruido y aquellas que tienen diferente nivel de ruido, muestra que tiene valores más altos en general que las métricas NMI y PDP. También se observa claramente la pérdida de calidad cuando aumenta el ruido lo que ayuda a distinguir que algoritmo mantiene la robustez, como se vio en los apartados anteriores. La ventaja de ARI es su sensibilidad para ver la degradación del clustering y mostrar diferencias claras entre los algoritmos. Una desventaja es que si los `clusters` que se van creando están desbalanceados, ARI puede dar valores más bajos, como se vio en el ejemplo del `dataset_12` La métrica NMI que mide cuanta información compartida hay entre `clusters` de las particiones sin ruido y las particiones con distinto nivel de ruido. NMI sigue una tendencia similar a ARI, pero suele ser un poco más estable, es decir la caída de ruido es ligeramente más estable, es decir NMI es menos sensible al deterioro, pero pue-

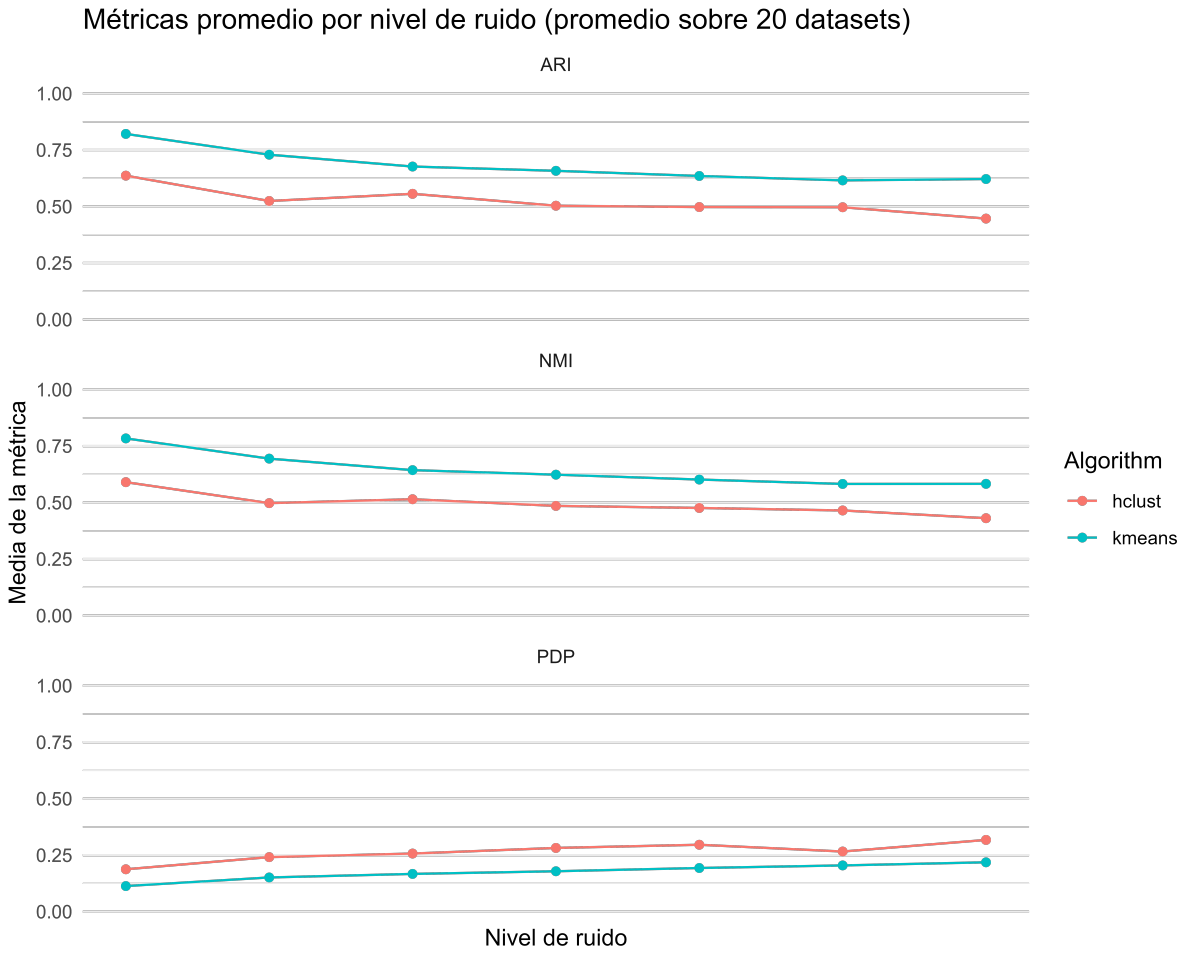

Figura 6.16: Gráficas de las métricas ARI, NMI y PDP agregadas para los 20 *datasets*.

de enmascarar pequeñas diferencias porque no cae tan drásticamente frente al ruido.

PDP, que mide la probabilidad que los puntos cambien de `clusters` frente al incremento de ruido, muestra que a diferencia de ARI y NMI que disminuyen con el ruido, la métrica PDP aumenta con el ruido, sin embargo, los valores son más bajos en general, llegando a valores menores de 0.5 en la mayoría de los casos lo que hace que sea difícil ver la separación entre algoritmos. La ventaja de esta métrica es que complementa a ARI y NMI porque mide la inestabilidad directamente, no refleja calidad de `clusters` con ARI o NMI sino solo estabilidad.

Al realizar la comparativa entre métricas, vemos que ARI es más sensible y puede detectar caídas más abruptas y claras, ideal para observar cual es el punto de quiebre cuando el ruido destruye la estructura del clustering. En cambio, NMI es útil si se quiere generalizar sin que el ruido extremo distorsione demasiado la comparación y finalmente PDP está más relacionada a la estabilidad, no mide calidad, lo que la hace clave para este tipo de experimentos.

Al realizar una comparativa de los algoritmos `k-means` y `hclust`, se observó que kmeans es más robusto por mantener valores más altos para ARI y NMI. Por lo contrario, `hclust` es más sensible y en varios *dataset* cae antes, sin embargo, hay casos donde las medidas de las métricas son muy pequeñas lo que indica que la robustes depende de la estructura de datos del *dataset*.

Capítulo 7

Conclusiones

El presente trabajo tuvo como objetivo analizar la robustez de dos tipos de algoritmos de clustering, los más representativos: `k-means` y clustering jerárquico (hclust) y su reacción frente a la introducción de distintos niveles de ruido en las instancias. Los experimentos realizados se hicieron con niveles de ruido que van de 0.2 a 1.4 sobre un conjunto variado de *datasets*, que presenta distintos números de características e instancias. La robustez se midió mediante métricas de correspondencia externa (ARI y NMI) y una medida interna de estabilidad basada en pares discrepantes (PDP). Los principales hallazgos pueden resumirse en los siguientes puntos:

1. El efecto del ruido sobre la calidad del clustering: Al incrementar el nivel de ruido, las métricas ARI y NMI tienden a descender, este resultado se esperaba, evidenciando una pérdida progresiva de la coherencia de los *clusters*. No obstante, la magnitud de esta degradación varía según el algoritmo y el *dataset* sobre el que se analiza.
    - La comparación entre `k-means` y hclust: Ambos algoritmos tienen formas muy distintas de agrupar y crear conjuntos de datos, pero en términos generales, `k-means` mostró una mayor estabilidad en promedio frente al ruido, especialmente en *datasets* con *clusters* bien definidos.
    - Con respecto a `hclust`, se mostró más sensible a las perturbaciones, llegando en algunos casos a colapsar drásticamente cuando el ruido superaba ciertos niveles, los *clusters* se agrupaban de cualquier manera y perdían coherencia con respecto a la partición base. Sin embargo, en *datasets* con estructuras jerárquicas claras, `hclust` fue capaz de mantener un rendimiento competitivo.
2. El comportamiento de las métricas:
    - Con ARI y NMI se observa una tendencia descendente más pronunciada, lo que refleja la pérdida de correspondencia con las particiones originales a medida que aumenta el ruido. Ambas métricas tienen valores similares, sin embargo, ARI muestra cambios en las curvas más marcados que NMI.
    - Con PDP, se detectaron diversos comportamientos: en algunos casos, los algoritmos tuvieron curvas con valores bajos y que se mantuvieron hasta niveles intermedios de ruido, lo que sugiere cierta capacidad de absorber perturbaciones antes de deteriorarse bruscamente

3. Indicador global de robustez: El análisis de las métricas ARI, NMI y PDP junto con sus valores promedios, permitió obtener una visión más integrada de la robustez. En este sentido, se confirma que ningún algoritmo es universalmente robusto, y que la elección depende de la naturaleza del *dataset* y del nivel de ruido esperado en el escenario de aplicación.
4. Las implicaciones prácticas: Las recomendaciones que se pueden extraer a partir de este estudio se centran en: no solo mirar la calidad del clustering, que tan bien agrupa los datos y la necesidad de evaluar la robustez, qué tanto resiste el algoritmo cuando los datos están contaminados con ruido, antes de elegir un algoritmo en entornos con datos ruidosos. Si bien `k-means` puede ser más consistente en escenarios estándar, `hclust` aunque es más sensible al ruido, puede resultar más útil donde la jerarquía de los datos sea importante.
5. Con respecto al comportamiento con los *datasets*:
    - En *datasets*con *clusters* bien definidos y baja dimensionalidad como Seeds, Banknote Authentication, Rice o Breast Cancer, tanto `k-means` como `hclust` tienen alta coherencia inicial, pero al introducir ruido, `k-means` mantuvo una degradación más suave, mientras que `hclust` fue más sensible.
    - Los *datasets*de alta dimensionalidad o heterogéneos como Travel Review Ratings, Yeast o Spotify Songs, donde los *clusters* son menos nítidos, el deterioro de las gráficas fue más pronunciado.
    - Dentro de los *datasets*para aplicaciones criticas como Heart Failure Clinical Records, Predict Diabetes o Smoke Detection implica riesgos en entornos prácticos al generar *clusters*inestables, esto marca la importancia de elegir bien los algoritmos para aplicaciones de seguridad y salud.
    - En entornos industriales los *datasets*Power Plant, Gas Turbine y Concrete mostraron que sus agrupaciones con ambos algoritmos toleraron niveles bajos de ruido, esto indica que en entornos industriales se puede tolerar un cierto umbral de ruido.
    - Otro dominio interesante se observó con el *datasets*de California Housing y Credit Card Customers que presenta atributos con alta varianza y *outliers*, lo que afecta negativamente la estabilidad, esto implica que la robustez también depende de la limpieza y calidad del *dataset*.

En conclusión, el análisis de los 20 *datasets*confirma que no existe un algoritmo totalmente robusto, sino que depende la interacción entre la naturaleza de los datos (el balance, separación de *clusters* y dimensionalidad) junto con la elección de un algoritmo adecuado y el nivel de ruido introducido.

Referencias

Arthur, D., y Vassilvitskii, S. (2007). k-means++: the advantages of careful seeding. En *Proceedings of the eighteenth annual acm-siam symposium on discrete algorithms* (p. 1027–1035). USA: Society for Industrial and Applied Mathematics.

Ben-Hur, A., Elisseeff, A., y Guyon, I. (2002). A stability based method for discovering structure in clustered data. En *Pacific symposium on biocomputing* (pp. 6–17). United States: World Scientific. Descargado de https://pubmed.ncbi.nlm.nih.gov/11928511/

Bishop, C. (2006, 01). Pattern recognition and machine learning. En (Vol. 16, p. 140-155). doi: 10.1117/1.2819119

Bittner, M., Meltzer, P., Chen, Y., Jiang, Y., Seftor, E., Hendrix, M., ... Trent, J. (2000). Molecular classification of cutaneous malignant melanoma by gene expression profiling. *Nature*, *406*(6795), 536–540. Descargado de https://doi.org/10.1038/35020115 doi: 10.1038/35020115

Chiquet, J., Rigaill, G., Sundqvist, M., Dervieux, V., y Bersani, F. (2023). aricode: Efficient computations of standard clustering comparison measures [Manual de software informático]. Descargado de https://github.com/jchiquet/aricode (R package version 1.0.3)

Duda, R., Hart, P., y G.Stork, D. (2001, 01). Pattern classification. En (Vol. xx).

Everitt, B., Landau, S., Leese, M., y Stahl, D. (2011). *Cluster analysis* (Vol. 5th). doi: 10.1002/9780470977811

Fabra-Boluda, R., Ferri, C., Ramírez-Quintana, M. J., y Martínez-Plumed, F. (2024). Unveiling the robustness of machine learning families. *Machine Learning: Science and Technology*, *5*(3), 035040. Descargado de https://doi.org/10.1088/2632-2153/ad62ab doi: 10.1088/2632-2153/ad62ab

Fawzi, A., Moosavi-Dezfooli, S.-M., y Frossard, P. (2016). Robustness of classifiers: From adversarial to random noise. En *Advances in neural information processing systems.*

Franceschi, L., Fawzi, A., y Fawzi, O. (2018). Robustness of classifiers to uniform ℓ_p and gaussian noise. En *Proceedings of the 21st international conference on artificial intelligence and statistics (aistats).*

Halkidi, M., Batistakis, Y., y Vazirgiannis, M. (2001, diciembre). On clustering validation techniques. *Journal of Intelligent Information Systems*, *17*(2), 107–145. Descargado de https://doi.org/10.1023/A:1012801612483 doi: 10.1023/A:1012801612483

Hubert, L., y Arabie, P. (1985, diciembre). Comparing partitions. *Journal of Classification*, *2*(1), 193–218. Descargado de https://doi.org/10.1007/BF01908075 doi: 10.1007/BF01908075

Jain, A. K. (2010, junio). Data clustering: 50 years beyond k-means. *Pattern Recog-*

nition Letters, *31*(8), 651–666. Descargado de https://www.sciencedirect.com/science/article/pii/S0167865509002323 (Award winning papers from the 19th International Conference on Pattern Recognition (ICPR)) doi: 10.1016/j.patrec.2009.09.011

Jain, A. K., Murty, M. N., y Flynn, P. J. (1999, septiembre). Data clustering: a review. *ACM Comput. Surv.*, *31*(3), 264–323. Descargado de https://doi.org/10.1145/331499.331504 doi: 10.1145/331499.331504

Jr., J. H. W. (1963). Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, *58*(301), 236–244. Descargado de https://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500845 doi: 10.1080/01621459.1963.10500845

Kelly, M., Longjohn, R., y Nottingham, K. (2024). *The uci machine learning repository.* https://archive.ics.uci.edu.

Ljunggren, D., y Ishii, S. (2021). *A comparative analysis of robustness to noise in machine learning classifiers* (Inf. Téc. n.º TRITA-EECS-EX-2021:492). Stockholm, Sweden: KTH Royal Institute of Technology, School of Electrical Engineering and Computer Science. Descargado de https://www.kth.se

Lloyd, S. (1982, marzo). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, *28*(2), 129–137. Descargado de https://doi.org/10.1109/TIT.1982.1056489 doi: 10.1109/TIT.1982.1056489

Lu, Y., Phillips, C. A., y Langston, M. A. (2019). A robustness metric for biological data clustering algorithms. *BMC Bioinformatics*, *20*(Suppl 15), 503. Descargado de https://doi.org/10.1186/s12859-019-3089-6 (From the 14th International Symposium on Bioinformatics Research and Applications (ISBRA'18), Beijing, China, 8-11 June 2018) doi: 10.1186/s12859-019-3089-6

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. En L. M. L. Cam y J. Neyman (Eds.), *Proc. of the fifth berkeley symposium on mathematical statistics and probability* (Vol. 1, p. 281-297). University of California Press.

Martínez-Plumed, F., Prudêncio, R. B., Martínez-Usó, A., y Hernández-Orallo, J. (2016). Making sense of item response theory in machine learning. En *Proceedings of the twenty-second european conference on artificial intelligence* (pp. 1140–1148).

Petety, A., Liu, C., y Koyejo, O. (2020). Attribute noise robust binary classification. En *Proceedings of the aaai conference on artificial intelligence.*

Shannon, C. E. (1948, julio). A mathematical theory of communication. *The Bell System Technical Journal*, *27*(3), 379–423. Descargado de https://doi.org/10.1002/j.1538-7305.1948.tb01338.x doi: 10.1002/j.1538-7305.1948.tb01338.x

Strehl, A., y Ghosh, J. (2002). Cluster ensembles – a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, *3*, 583–617. Descargado de https://doi.org/10.1162/153244303321897735 doi: 10.1162/153244303321897735

Tibshirani, R., Walther, G., y Hastie, T. (2002, 01). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *63*(2), 411-423. Descargado de https://doi.org/10.1111/1467-9868.00293 doi: 10.1111/1467-9868.00293

Vinh, N., Epps, J., y Bailey, J. (2010). Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chan-

ce. *Journal of Machine Learning Research*, *11*, 2837–2854. Descargado de http://jmlr.org/papers/v11/vinh10a.html

von Luxburg, U. (2010). Clustering stability: An overview. *Foundations and Trends® in Machine Learning*, *2*(3), 235–274. Descargado de https://doi.org/10.1561/2200000008 doi: 10.1561/2200000008

Zhao, J., Yu, G., y Liu, Y. (2018). Assessing robustness of classification using angular breakdown point. *The Annals of Statistics*, *46*(6B), 3362–3389. Descargado de https://doi.org/10.1214/17-AOS1661 doi: 10.1214/17-AOS1661

Zhu, X., y Wu, X. (2004). Class noise vs. attribute noise: A quantitative study of their impacts. *Artificial Intelligence Review*, *22*(3), 177–210.

Anexo I: Datasets Utilizados

#	Nombre	Fuente	Inst.	Attr.	Observaciones	¿Num.?	Elim.	Cat.	Bin.
1	Seeds Dataset	U	210	8	NA	SI	class		NO
2	Wholesale Customers	U	440	8	Tiene 2 datos categóricos. El valor de channel tiene varianza aceptable; region se descarta por tener 3 valores.	NO	Region	Channel, Region	SI
3	Banknote Authentication	U	1.4K	5	Se eliminan las etiquetas de clase.	SI	class		NO
4	Travel Review Ratings	U	5.5K	26	Tiene etiquetas, identificadores y atributos textuales.	SI	user, x		NO
5	EastWestAirlines - Heirarchical Clustering	K	4.0K	12	Tiene etiquetas, identificadores y categóricos de más de 2 valores que aportan poca variabilidad.	NO	id, cc1_miles(5), cc2_miles(3), cc3_miles(5), Award	cc1_miles(5), cc2_miles(3), cc3_miles(5)	NO
6	Dataset of Songs in Spotify	K	42.3K	22	Tiene etiquetas, identificadores, atributos textuales, valores nulos y variables irrelevantes.	NO	genre, song_name, Unnamed..0, Unnamed: 0, title, type, id, uri, track_href, analysis_url, key, time_signature	mode, time_signature, genre	SI

#	Nombre	Fuente	Inst.	Attr.	Observaciones	¿Num.?	Elim.	Cat.	Bin.
7	Power Plant Data	K	6.4K	5	El atributo de salida es PE, relacionado a temperatura y presión.	SI	PE		NO
8	California Housing Prices	K	20.6K	10	Se elimina ocean_proximity. Algunos atributos tienen pocos NA.	NO	ocean_proximity		NO
9	Heart Failure Clinical Records	U	299	13	No tiene valores faltantes. Atributo objetivo: DEATH_EVENT.	NO	DEATH_EVENT	anaemia, diabetes, high_blood_pressu sex, smoking	SI
10	Smoke Detection Dataset	K	62.6K	16	Se eliminan datos duplicados, temporales, etiquetas y contadores.	SI	Fire Alarm, UTC, CNT		NO
11	Wine Quality	U	6.5K	12	Se eliminan las etiquetas de clase. Se une el dataset de vino tinto con el de vino blanco.	SI	class		NO
12	Dry Bean	U	13.6K	17	Se eliminan las etiquetas de clase.	SI	class		NO
13	HTRU2	U	17.9K	9	Se eliminan las etiquetas de clase.	SI	class		NO
14	Concrete Compressive Strength	U	1.0K	9	Se eliminan las etiquetas de clase. 'age' tiene 14 valores únicos pero varianza útil.	SI	strength		NO
15	Gas Turbine CO and NOx Emission	U	36.7K	11	Se elimina 'year', 'NOX', 'CO' y 'TEY'.	SI	year, CO, NOX, TEY		NO
16	Breast Cancer (Diagnostic)	U	569	32	Se eliminan las etiquetas de id y clase.	SI	id, diagnosis		NO
17	Rice (Cammeo and Osmancik)	U	3.8K	8	Se eliminan las etiquetas de clase.	SI	class		NO
18	Credit Card Clustering	K	8.9K	18	Se elimina 'ID'. Cuidado con 'MINIMUM_PAYMENTS' con NA. Valores enteros y ruido.	SI	CASH_ADVANCE_1 PURCHASES_TRX, TENURE		NO

#	Nombre	Fuente	Inst.	Attr.	Observaciones	¿Num.?	Elim.	Cat.	Bin.
19	Yeast	U	1.5K	10	Se eliminan las etiquetas de clase. Se elimina 'pox' y se conserva 'erl'.	SI	Pox	Pox, erl	SI
20	Predict Diabetes	K	768	9	Se eliminan las etiquetas de clase.	SI	outcome		NO

Tabla 1: Resumen de los 20 datasets utilizados

Anexo II: Gráficas de niveles de ruido con respecto a las métricas por dataframe

Algoritmo k-means

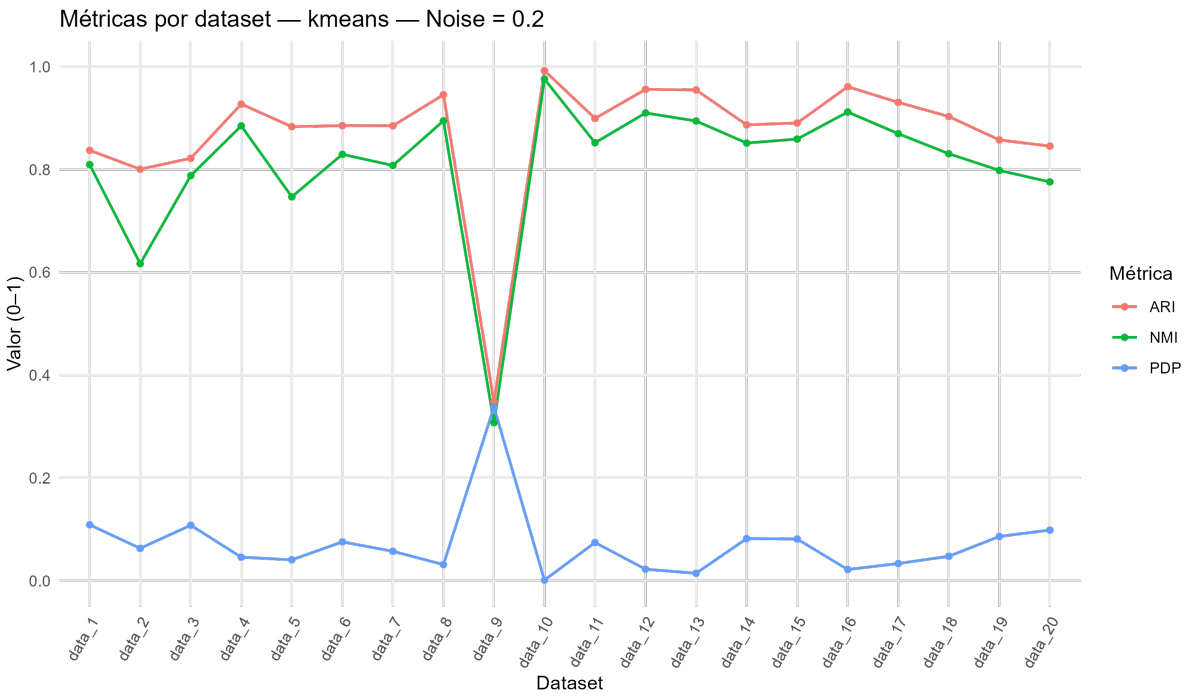

Figura 1: Gráficas de la métricas para los 20 datasets con el algoritmo `k-means` con ruido 0.2

Algoritmo Hclust

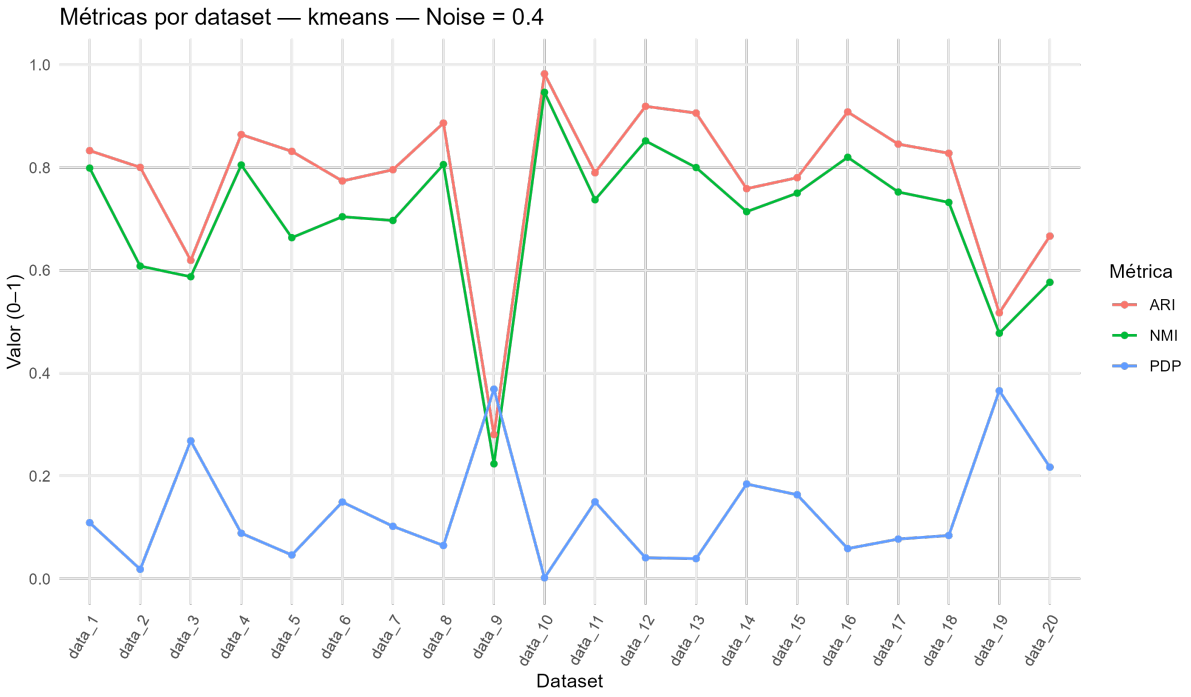

Figura 2: Gráficas de la métricas para los 20 datasets con el algoritmo `k-means` con ruido 0.4

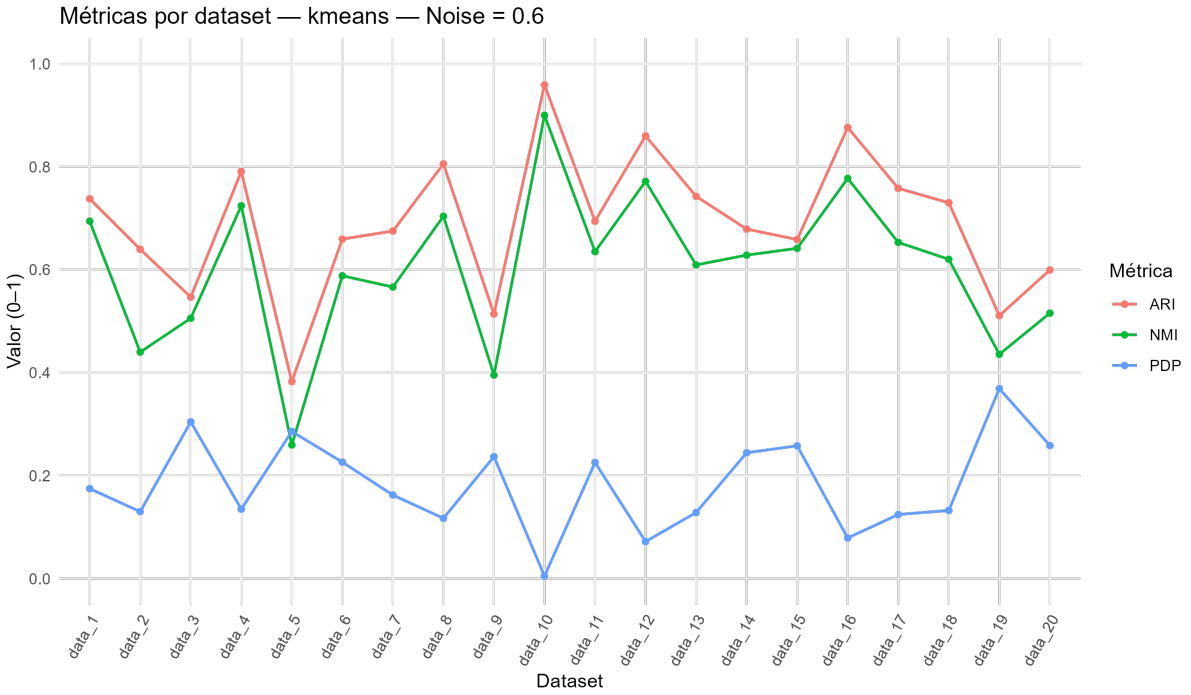

Figura 3: Gráficas de la métricas para los 20 datasets con el algoritmo `k-means` con ruido 0.6

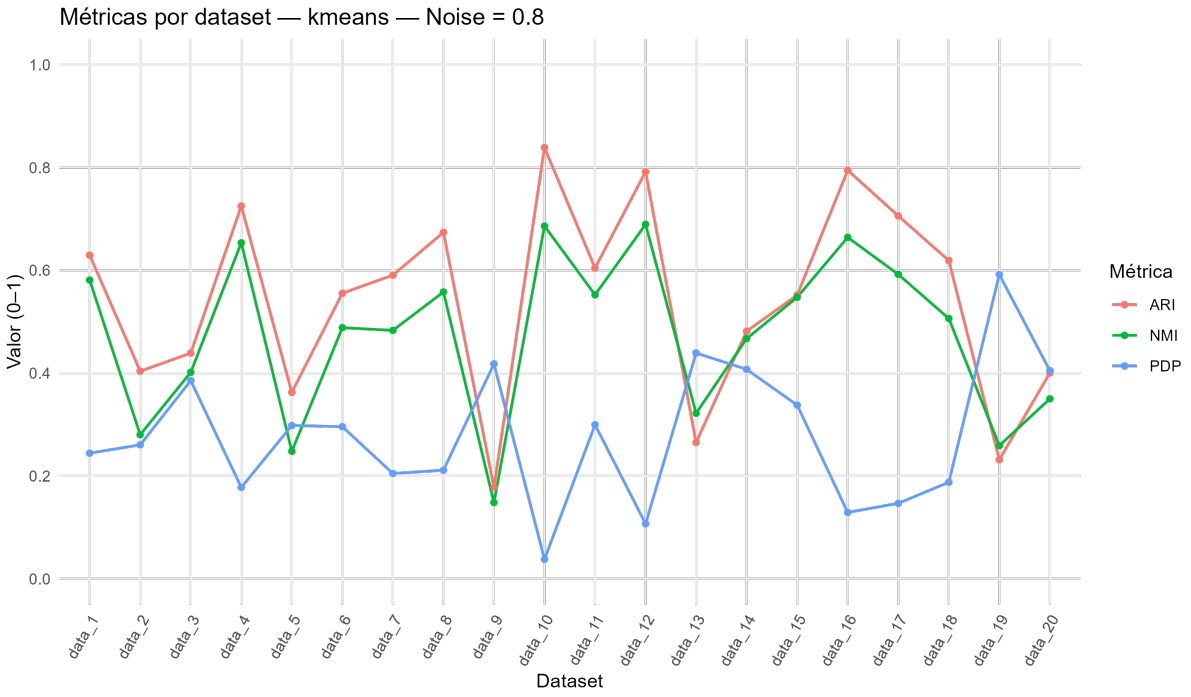

Figura 4: Gráficas de la métricas para los 20 datasets con el algoritmo `k-means` con ruido 0.8

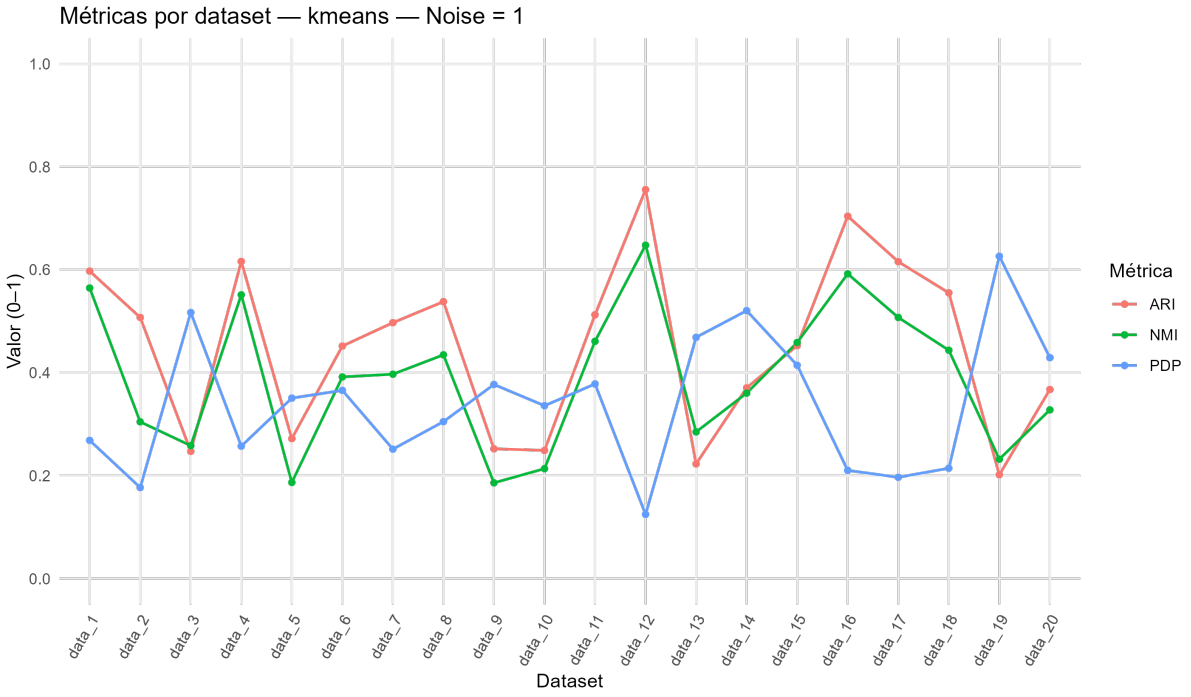

Figura 5: Gráficas de la métricas para los 20 datasets con el algoritmo `k-means` con ruido 1.0

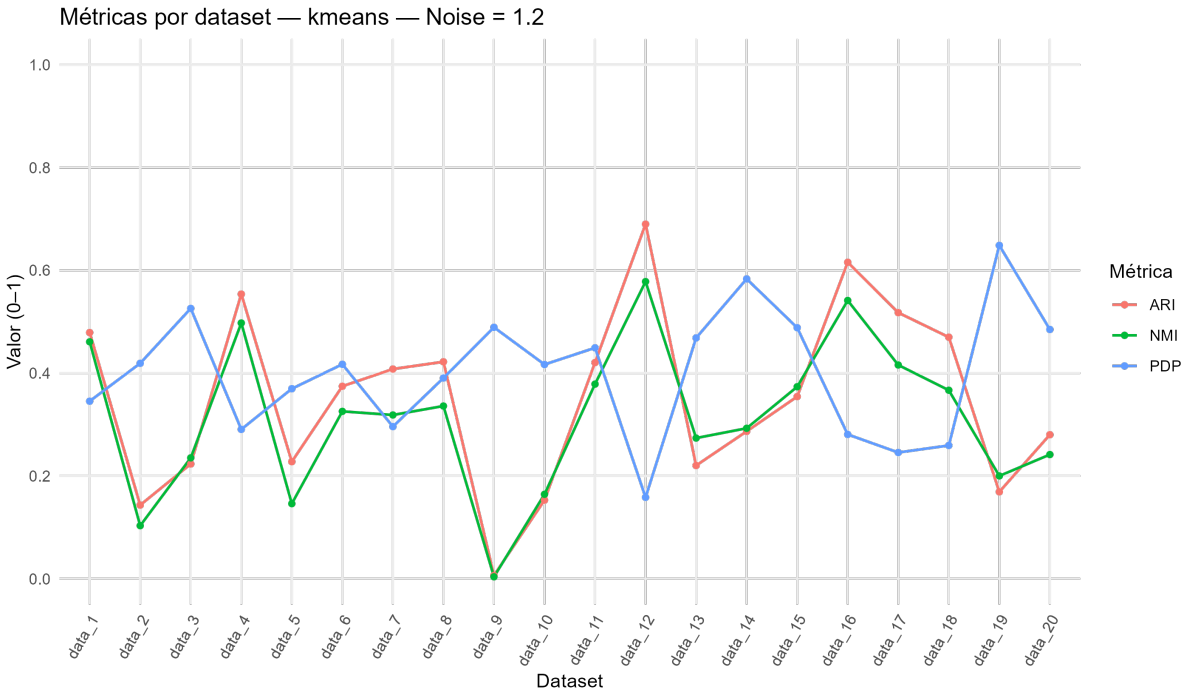

Figura 6: Gráficas de la métricas para los 20 datasets con el algoritmo `k-means` con ruido 1.2

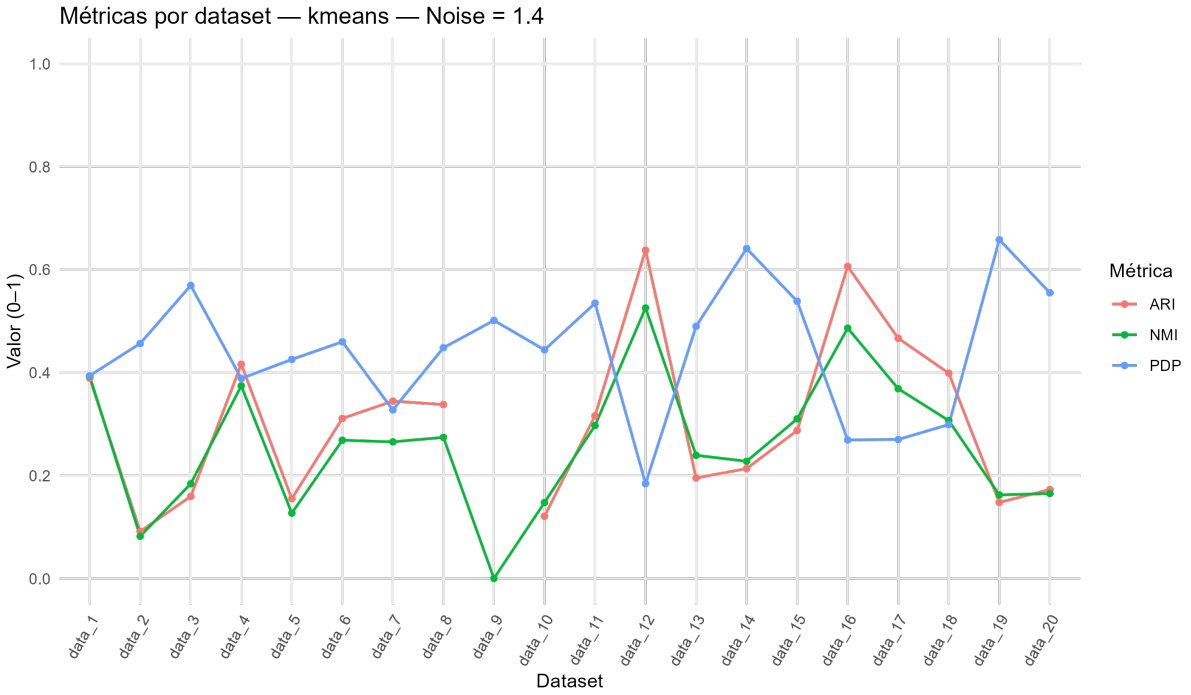

Figura 7: Gráficas de la métricas para los 20 datasets con el algoritmo `k-means` con ruido 1.4

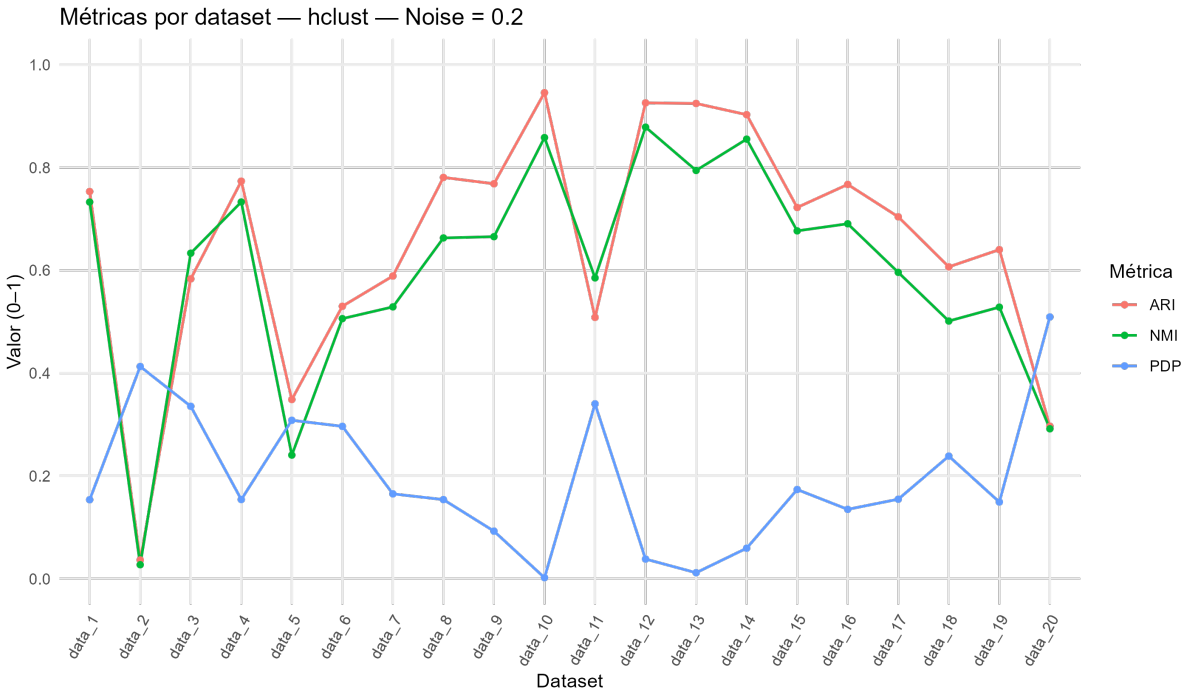

Figura 8: Gráficas de la métricas para los 20 datasets con el algoritmo textthclust con ruido 0.2

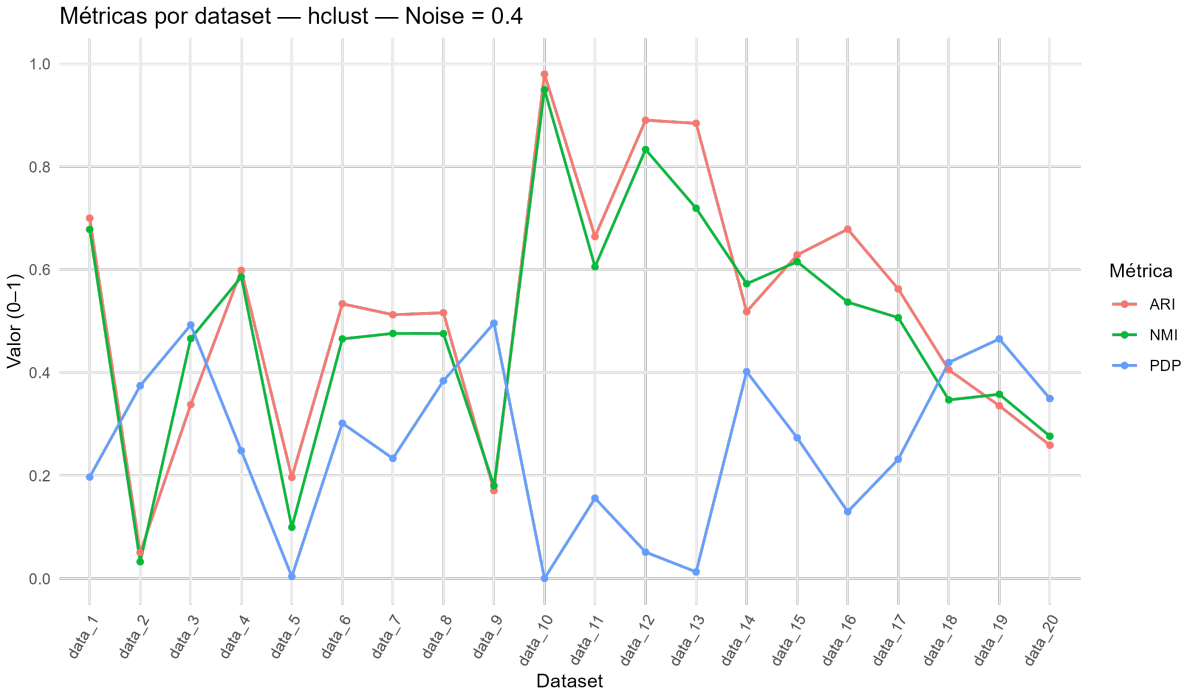

Figura 9: Gráficas de la métricas para los 20 datasets con el algoritmo textthclust con ruido 0.4

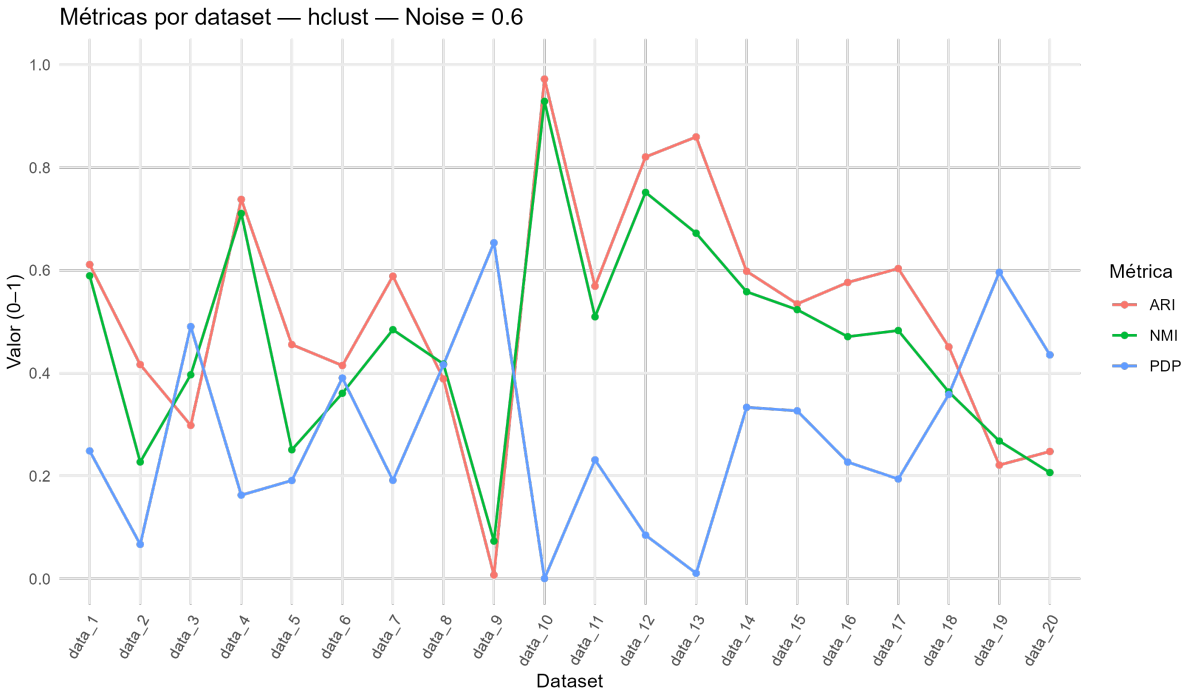

Figura 10: Gráficas de la métricas para los 20 datasets con el algoritmo textthclust con ruido 0.6

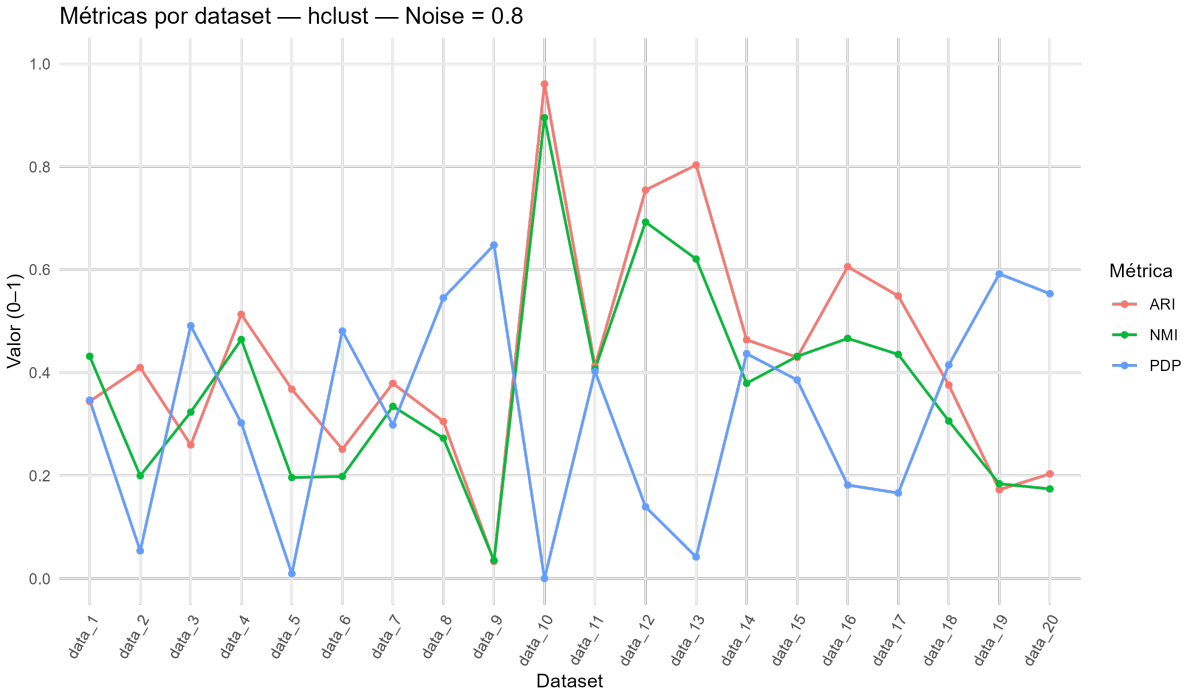

Figura 11: Gráficas de la métricas para los 20 datasets con el algoritmo hclust con ruido 0.8

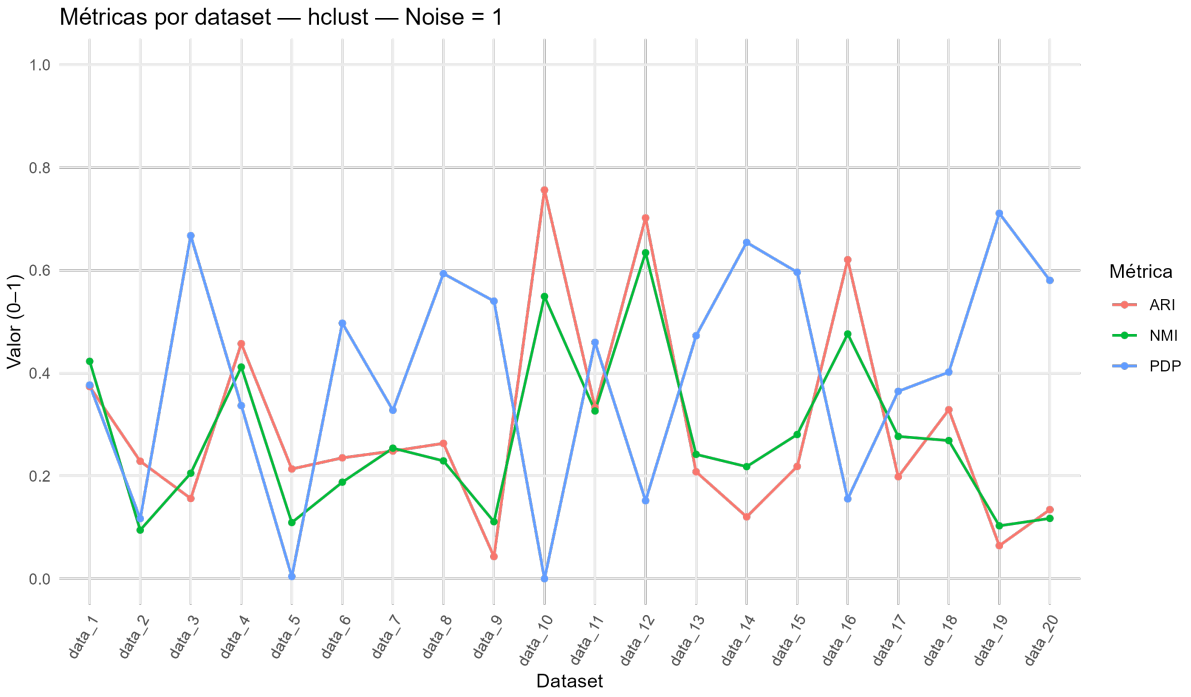

Figura 12: Gráficas de la métricas para los 20 datasets con el algoritmo hclust con ruido 1.0

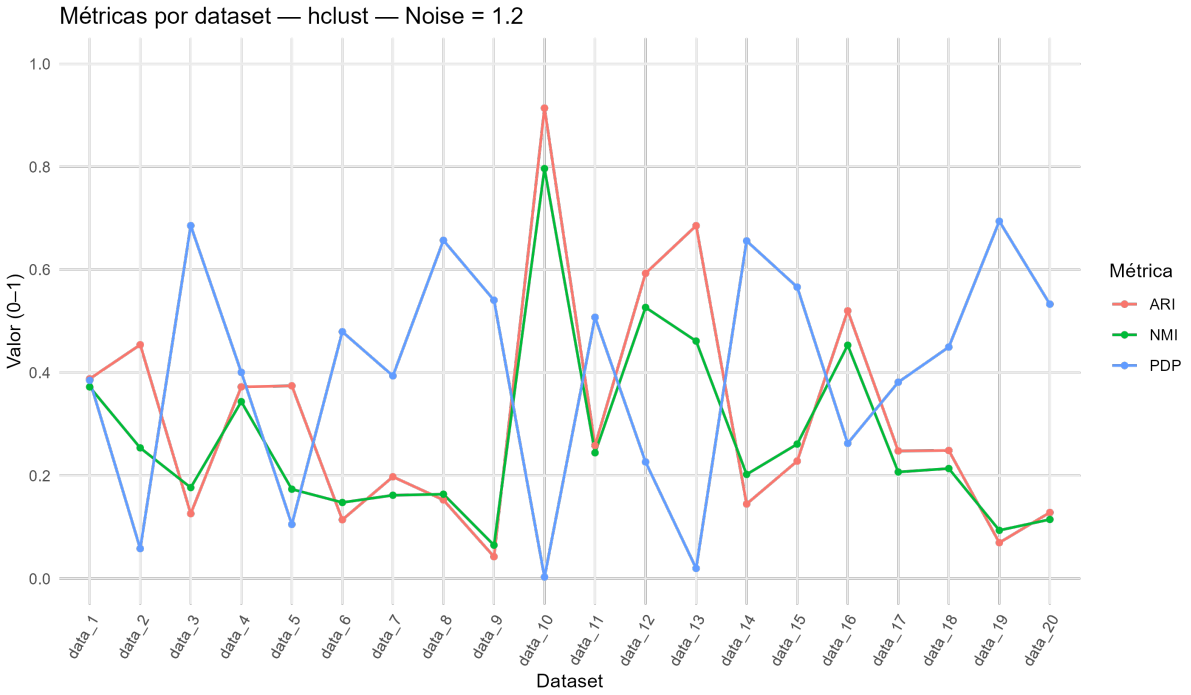

Figura 13: Gráficas de la métricas para los 20 datasets con el algoritmo hclust con ruido 1.2

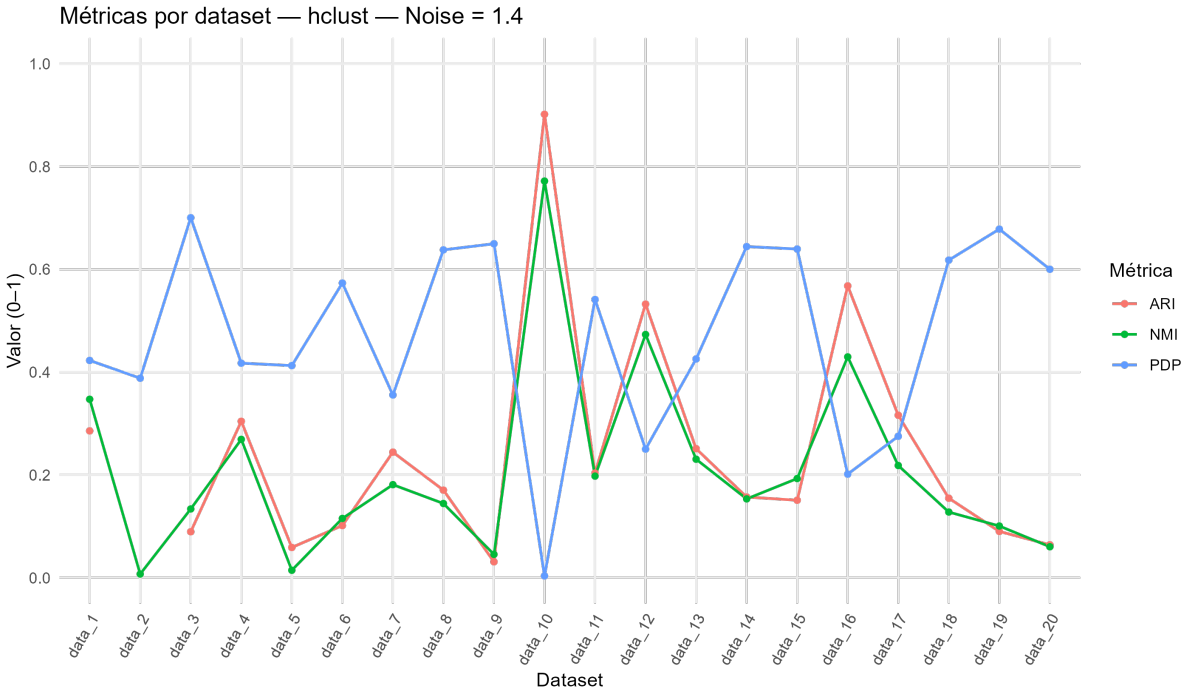

Figura 14: Gráficas de la métricas para los 20 datasets con el algoritmo textthclust con ruido 1.4