



Universidad de Valladolid

GRADO EN ESTADÍSTICA

TRABAJO FIN DE GRADO

**Implementación de técnicas para
Análisis Cluster robusto en torno a
subespacios afines**

Autor

D. Jesús Fernández Iglesias

Tutor

D. Luis Ángel García Escudero

Resumen

La constante generación de conjuntos de datos masivos que se produce en la actualidad ha provocado que el desarrollo de técnicas de aprendizaje automático capaces de extraer conocimiento útil de dicha información sea un campo del conocimiento en auge y en constante desarrollo.

En muchos de estos problemas, las observaciones no tienen asociadas ningún tipo de etiqueta, categoría o clase, únicamente se dispone de los propios datos. Por tanto, la búsqueda de patrones ocultos en los mismos se torna una tarea fundamental.

Con ese fin, el paradigma de aprendizaje no supervisado ofrece una amplia gama de procedimientos que permiten el estudio y agrupación de objetos en base a sus similitudes.

En la intersección de la incesante creación de conjuntos de datos enormes y el aprendizaje no supervisado surge la necesidad de desarrollar e implementar procedimientos computacionalmente eficientes para poder aplicar estas técnicas de aprendizaje no supervisado y en particular de la aplicación de técnicas de análisis *cluster*.

En este trabajo se estudia y se desarrollan versiones computacionalmente eficientes de un procedimiento de análisis *cluster* robusto entorno a subespacios afines. Un enfoque robusto al análisis *cluster* evita que unas pocas observaciones atípicas pueden condicionar de manera muy negativa la detección correcta de *clusters*. La metodología desarrollada examina varias opciones de implementación, explorando el enfoque secuencial, el paralelizado y el híbrido sacando partido a varios lenguajes de programación, además de realizar los correspondientes análisis de eficiencia computacional para determinar qué versión es la más adecuada.

Además, un ejemplo de aplicación real del procedimiento desarrollado es mostrado en el ámbito de la segmentación de imágenes.

Abstract

The constant generation of massive data sets nowadays has made the development of machine learning techniques capable of extracting useful knowledge from such information a booming and constantly developing field of knowledge.

In many of these problems, the observations do not have any kind of label, category or class associated with them; only the data itself is available. Therefore, the search for hidden patterns in the data becomes a fundamental task.

To that end, the unsupervised learning paradigm offers a wide range of procedures that allow the study and clustering of objects based on their similarities.

At the intersection of the incessant creation of huge datasets and unsupervised learning arises the need to develop and implement computationally efficient procedures to be able to apply these unsupervised learning techniques and in particular the application of cluster analysis techniques.

In this paper we study and develop computationally efficient versions of a robust cluster analysis procedure around affine subspaces. A robust approach to cluster analysis avoids that a few outlier observations can condition in a very negative way the correct detection of clusters. The developed methodology examines several implementation options, exploring sequential, parallelized and hybrid approaches taking advantage of several programming languages, as well as performing the corresponding computational efficiency analysis to determine which version is the most suitable.

In addition, an example of real application of the developed procedure is shown in the field of image segmentation.

Agradecimientos

A los profesores del Grado en Estadística que me han acompañado durante estos años por transmitirme la pasión y el conocimiento necesarios para formar mi futuro profesional.

A Luis Ángel, por la constante dedicación y ayuda que han puesto en este proyecto y resolver todas las cuestiones que le he ido planteando de manera rápida y eficaz.

A mis compañeros de carrera por haber compartido esta travesía a mi lado, superando juntos todas las dificultades surgidas.

Y por último, a mi familia, en especial a mis padres, por inculcarme los valores del sacrificio y el esfuerzo tan necesarios, y ser los pilares que me han sostenido durante toda esta etapa.

Índice general

1. Introducción	14
1.1. Contextualización	14
1.2. Contenidos a tratar	15
1.3. Resumen global de resultados	16
1.4. Herramientas utilizadas	16
2. Análisis Cluster y Componentes Principales	18
2.1. Métodos de <i>Clustering</i> jerárquicos	18
2.2. Métodos de <i>Clustering</i> no jerárquicos o particionales	19
2.2.1. <i>K</i> -medias	19
2.3. Análisis en Componentes Principales	20
3. Robustez	24
3.1. Media y mediana	25
3.2. <i>K</i> -medias recortadas	28
3.3. Aplicación de los recortes al ACP	32
4. Clustering robusto entorno a subespacios lineales	36
4.1. Agrupaciones robustas entorno a subespacios afines d -dimensionales	37
4.1.1. Caso I: problema teórico o poblacional	37
4.1.2. Caso II: problema empírico o muestral	38
4.2. Descripción del algoritmo	38
5. Implementación	41
5.1. Mejoras añadidas	41
5.1.1. Subespacios afines de dimensiones diferentes	41
5.1.2. Estimador de mínimo determinante de la covarianza y distancia robusta de Mahalanobis	45
5.1.3. Funcionalidad gráfica	49
5.1.3.1. 2D	50
5.1.3.2. 3D	51
5.2. Pseudocódigo	54
5.3. Computación paralela	58
5.3.1. Búsqueda de subespacios afines inicial	59
5.3.2. Iteración completa de las inicializaciones más prometedoras	62
5.3.3. Flujo general del programa	63
5.4. Integración de R y C++	66
5.4.1. Búsqueda de subespacios afines inicial	68

5.4.2.	Iteración completa de las inicializaciones más prometedoras	72
5.4.3.	Flujo general del programa	75
5.5.	Análisis comparativo de tiempos de ejecución	76
5.5.1.	Secuencial vs paralelización vs integración R & C++	77
5.5.2.	Función <code>trimkmeans</code> vs función <code>trimksubspaces</code>	80
6.	Aplicación real en la segmentación de imágenes	84
6.1.	<i>Clustering</i> de una imagen en 3 grupos.	86
6.2.	<i>Clustering</i> de una imagen en 6 grupos.	87
6.3.	<i>Clustering</i> de una imagen con ruido	88
7.	Conclusiones y líneas futuras	92
7.1.	Conclusiones	92
7.2.	Líneas futuras	93
	Referencias	94

Índice de figuras

2.1. Agrupamiento realizados por k -medias ($k = 3$).	20
2.2. Representación esquemática de las componentes principales z_1 y z_2 de una nube de puntos elíptica en \mathbb{R}^2	22
2.3. Rotación de los datos en el sistema de coordenadas de las componentes principales z_1 y z_2	22
3.1. Representación de los datos x_1, \dots, x_{40}	25
3.2. Representación de los datos x_1, \dots, x_{40} junto con su media, mediana por coordenadas y mediana espacial.	27
3.3. Representación de los datos x_1, \dots, x_{1800}	29
3.4. Agrupaciones realizadas por las k -medias.	29
3.5. Agrupaciones realizadas por las k -medias recortadas ($k = 2, \alpha = 0.05$).	32
3.6. Representación del conjunto de datos x_1, \dots, x_{1000}	33
3.7. Dirección de máxima varianza encontrada por ACP.	34
3.8. Dirección de máxima varianza encontrada por ACP con recortes.	35
5.1. Representación del conjuntos de observaciones x_1, \dots, x_{6000}	42
5.2. Agrupaciones realizadas ($d=(0,0,0)$ y $\alpha = 0.05$).	42
5.3. Agrupaciones realizadas ($d=(1,1,1)$ y $\alpha = 0.05$).	43
5.4. Agrupaciones realizadas ($d=(2,2,2)$ y $\alpha = 0.05$).	43
5.5. Agrupaciones realizadas ($d=(0,1,2)$ y $\alpha = 0.05$).	44
5.6. Representación del conjunto de observaciones x_1, \dots, x_{26000}	46
5.7. Agrupaciones realizadas ($d=(1,2)$ y $\alpha = 0$).	46
5.8. Elipses de tolerancia clásica y robusta sobre un mismo conjunto de datos.	48
5.9. Agrupaciones realizadas ($d=(1,2)$ y $\alpha = 0$) tras reasignar los grupos mediante MCD y Mahalanobis.	49
5.10. Representación bidimensional de las agrupaciones realizadas entorno a subespacios afines de dimensión 0.	50
5.11. Representación bidimensional de las agrupaciones realizadas entorno a subespacios afines de dimensión 1.	50
5.12. Representaciones tridimensionales de las agrupaciones realizadas entorno a subespacios afines de dimensión 0.	51
5.13. Representaciones tridimensionales de las agrupaciones realizadas entorno a subespacios afines de dimensión 1.	52
5.14. Representaciones tridimensionales de las agrupaciones realizadas entorno a subespacios afines de dimensión 2.	53
5.15. Diagrama de flujo de la versión paralela del código.	64
5.16. Diagrama de flujo de la versión que integra R y C++.	67
5.17. <i>Global enviroment</i> de RStudio con funciones definidas en un código fuente C++ accesibles tras compilarle.	75

5.18. Diagrama de cajas del tiempo en escala logarítmica frente a la versión (Exp. 1).	78
5.19. Diagrama de cajas del tiempo en escala logarítmica frente a la versión (Exp. 2).	78
5.20. Diagrama de cajas del tiempo en escala logarítmica frente a la versión (Exp. 3).	79
5.21. Diagrama de cajas del tiempo en escala logarítmica frente a la versión (Exp. 4).	79
5.22. Regresión lineal a los tiempos medidos.	80
5.23. Diagrama de cajas del tiempo en escala logarítmica frente a la función (Exp. 1).	81
5.24. Diagrama de cajas del tiempo en escala logarítmica frente a la función (Exp. 2).	81
5.25. Diagrama de cajas del tiempo en escala logarítmica frente a la función (Exp. 3).	82
5.26. Regresión lineal a los tiempos medidos.	82
6.1. Segmentación de imágenes.	84
6.2. Imagen base a la que se va a aplicar <i>clustering</i> .	85
6.3. Conversión imagen-conjunto de datos.	85
6.4. Imágenes de segmentación obtenidas tras realizar los agrupamientos entorno a dimensiones: 0-0-0, 0-0-1, 0-1-1 y 1-1-1 (de izquierda a derecha y de arriba a abajo).	86
6.5. Imágenes de segmentación obtenidas tras realizar los agrupamientos entorno a dimensiones: 0-0-0-0-0-0, 0-0-0-0-0-1, 0-0-0-0-1-1, 0-0-0-1-1-1, 0-0-1-1-1-1, 0-1-1-1-1-1 y 1-1-1-1-1-1 (de izquierda a derecha y de arriba a abajo).	88
6.6. Imagen de segmentación obtenida tras realizar los agrupamientos con $d = (0, 0, 0, 0)$.	89
6.7. Imagen base con un 2% de píxeles contaminados.	89
6.8. Imágenes de segmentación obtenidas tras realizar los agrupamientos con $\alpha = 0$ y $alpha = 0.02$ ($d = (0, 0, 0, 0)$).	90
6.9. Imagen de segmentación obtenida tras realizar los agrupamientos con $d = (1, 1, 1, 1)$.	90
6.10. Imagen base con un 1.5% de píxeles contaminados.	91
6.11. Imágenes de segmentación obtenidas tras realizar los agrupamientos con $\alpha = 0$ y $alpha = 0.015$ ($d = (1, 1, 1, 1)$).	91

ÍNDICE DE CUADROS

Índice de cuadros

5.1. Tiempo promedio en segundos de las versiones sobre los experimentos.	77
5.2. <i>Intercept</i> y pendiente de las rectas de regresión ajustadas.	79
5.3. Tiempo promedio en segundos de las funciones sobre los experimentos.	81
5.4. <i>Intercept</i> y pendiente de las rectas de regresión ajustadas.	83

1. Introducción

1.1. Contextualización

El aprendizaje automático es una aplicación de la Inteligencia Artificial que provee a los sistemas de la habilidad para automáticamente aprender y mejorar a través de la experiencia y sin ser explícitamente programados. El proceso de aprendizaje comienza con datos, experiencia directa y/o instrucciones, tratando de buscar y encontrar patrones en dicha información para mejorar la toma de decisiones futura.

Dentro del aprendizaje automático, se encuentran varios paradigmas:

- **Aprendizaje supervisado:** se caracteriza por aprovechar la información que proporcionan las etiquetas o los valores de una variable respuesta privilegiada para una muestra de datos o conjunto de entrenamiento. La naturaleza de la variable respuesta permite diferenciar entre problemas de clasificación, donde ésta es categórica, y problemas de regresión, donde ésta es numérica. Algoritmos y técnicas tan conocidas como los árboles de decisión, la regresión logística, SVM (*Support Vector Machines*), *K*-NN, Naive-Bayes, *Bagging* o *Boosting* pertenecen a este paradigma.
- **Aprendizaje no supervisado:** trata de inferir patrones presentes en conjuntos de datos no etiquetados. Se consiguen aprender propiedades estructurales internas muy valiosas del conjunto de datos. Este paradigma será el eje central sobre el que pivote este trabajo.
- **Aprendizaje semisupervisado:** utiliza tanto datos etiquetados como sin etiquetar para generar las hipótesis. Este paradigma se encuentra a mitad de camino entre el aprendizaje supervisado y el no supervisado. Los algoritmos pertenecientes a esta categoría operan con una pequeña cantidad de datos que están etiquetados, mientras que para el resto se desconoce o se omite intencionadamente la etiqueta. Los modelos que se engloban en esta categoría suelen responder a alguno de los siguientes: *generative*, *low-density separation*, *graph-based* o *heuristic*.
- **Aprendizaje por refuerzo:** no se dispone de datos etiquetados, un agente o modelo interactúa con el medio y aprende mediante la realización de acciones que pueden desencadenar recompensas. Se busca maximizar el número de premios. Un ejemplo sencillo para imaginarse el funcionamiento de este tipo de aprendizaje es un robot que está aprendiendo a andar. Cuanto más tiempo aguante andando sin caerse, la recompensa que obtendrá será mayor.

La presencia de tantas aproximaciones para la resolución de problemas de aprendizaje automático ofrece una amplia variedad y diversidad de soluciones, y su continuo e incesante desarrollo permite aplicar estos algoritmos a cada vez más problemas pertenecientes a un amplio abanico de contextos, obteniendo resultados excelentes en muchos casos.

1 INTRODUCCIÓN

1.2. Contenidos a tratar

En este trabajo se van a estudiar y tratar algoritmos de análisis *cluster*. Además, se implementará *software* orientado al desarrollo de técnicas *clustering* robusto en torno a subespacios afines. El enfoque desarrollado generaliza el conocido algoritmo de las k -medias, eliminando tanto la restricción de únicamente poder agrupar datos entorno a puntos o centroides, permitiendo agrupar datos en torno a subespacios lineales, como la restricción de que los k subespacios sean de la misma dimensión, es decir, se permitirá realizar agrupamientos entorno a estructuras de dimensiones diferentes. La robustez se logra aplicando recortes, los cuales permiten descartar aquellas observaciones más atípicas para la tarea de búsqueda y determinación de los subespacios afines.

En el Capítulo 2, se hará una revisión del análisis *cluster*, tratando con más profundidad el algoritmo de las k -medias. También se tratará el Análisis en Componentes Principales(ACP) y su papel como procedimiento para reducir la dimensionalidad del problema. Ambos conceptos convergen en las técnicas que se tratarán a lo largo del trabajo.

En el Capítulo 3, se tratará el concepto de robustez estadística, más en particular, cómo pueden afectar las observaciones atípicas de manera perjudicial en multitud de métodos, y cómo pueden modificarse dichos procedimientos para que sean resistentes a los *outliers*. Además, se expondrán los algoritmos de las k -medias recortadas y el Análisis en Componentes Principales recortadas, procedimientos que incorporan la robustez estadística a los conocidos algoritmos con los que comparten nombre.

El método central sobre el que pivota todo el trabajo será presentado en el Capítulo 4. Se expondrán sus fundamentos y bases teóricas, además de describir un algoritmo que da pie a su implementación a nivel de *software*.

El Capítulo 5 tiene un carácter más tecnológico. Se describirá cómo se ha implementado la versión final del algoritmo aunando los lenguajes de programación **R** y **C++** para conseguir una excelente eficiencia computacional. Además, se hará un estudio comparativo de tiempos de ejecución tanto con otras implementaciones del algoritmo desarrolladas en este mismo trabajo como con *software* disponible en el *CRAN (Comprehensive R Archive Network)* de **R** que realiza funciones equivalentes a una simplificación del procedimiento implementado. También se explicará qué funcionalidad a mayores de lo detallado en el Capítulo 4 ha logrado implementarse para mejorar los resultados.

El Capítulo 6 está dedicado a mostrar un ejemplo del funcionamiento del método implementado en una aplicación real: la segmentación de imágenes. Se aplicará *clustering* a una imagen con el objetivo de agrupar los píxeles en base a sus intensidades en los canales RGB con el fin de generar máscaras de segmentación. Esto nos permitirá observar desde una perspectiva visual cómo afecta el realizar agrupamientos entorno a dimensiones distintas.

El último capítulo de esta memoria está dedicado a exponer las conclusiones del trabajo y a plantear líneas futuras de investigación.

1.3. Resumen global de resultados

En este trabajo se ha logrado desarrollar una versión computacionalmente muy eficiente del algoritmo que agrupa observaciones entorno a subespacios afines de manera robusta. Dicha versión adapta un enfoque híbrido, donde el código es desarrollado en dos lenguajes de programación que interactúan e intercambian información durante la ejecución del procedimiento. Los lenguajes que cooperan son **R** y **C++**, siendo el primero el encargado de llevar a cabo el flujo general del programa y el segundo el que realiza las partes computacionalmente más costosas del algoritmo.

La versión híbrida mejora con creces tanto a una versión que aprovecha el poder de la computación paralela también desarrollada y detallada en este trabajo como a una función disponible en el CRAN de **R**. Los experimentos desarrollados y los resultados obtenidos son expuestos en el Capítulo 5.

Aprovechando la eficiencia de la implementación desarrollada, el procedimiento se aplica a la segmentación de imágenes, las cuales han de ser aplanadas constituyendo en sí mismas un conjunto de datos inmenso para tamaños de imágenes comunes. Se aprecia como la aplicación del algoritmo con distintas configuraciones genera máscaras de segmentación diferentes, remarcando o suavizando partes de intensidad más débil de la imagen, las cuáles pueden ser muy útiles en multitud de dominios de aplicación.

1.4. Herramientas utilizadas

A continuación se muestra la relación de herramientas utilizadas durante la realización del proyecto:

- **R**: es un lenguaje de programación de distribución libre ampliamente usada por la comunidad estadística y en ciencia de datos, está orientado al desarrollo de *software* estadístico y al análisis de datos. En este trabajo, buena parte de los gráficos se han creado con esta herramienta, además del desarrollo íntegro de dos versiones del procedimiento implementado en el Capítulo 5, una secuencial y una paralelizada.
- **C++**: es un lenguaje de programación generalista que extiende al lenguaje *C* para permitir la manipulación de objetos. En este trabajo se ha utilizado para implementar parte de la versión híbrida (combinando dos lenguajes) del análisis *cluster* robusto entorno a subespacios afines. Se trata de uno de los lenguajes de programación más eficientes que existen.

1 INTRODUCCIÓN

- ***RStudio***: es un entorno de desarrollo integrado para el lenguaje de programación R. Contiene herramientas para la gestión, el trazado y la depuración del espacio de trabajo. Los códigos R y C++ han sido desarrollados en este *IDE* (*Integrated Development Environment*).
- ***TeXStudio***: es un editor de \LaTeX de código abierto y multiplataforma. La decisión de utilizar \LaTeX para la redacción de esta memoria es por su característica de producir documentos con alta calidad tipográfica.

2. Análisis Cluster y Componentes Principales

A fecha de realización de este documento, se generan, aproximadamente, 2.5 quintillones de *gigabytes* de información al día. Si queremos obtener conocimiento útil a partir de esta información mediante la aplicación de técnicas de ML (*Machine Learning*) nos encontraremos que la gran mayoría de las observaciones no están etiquetadas en ninguna clase o no se dispone de una variable respuesta medida en ellas.

Es por ello que el aprendizaje no supervisado es cada vez más usado, cobrando especial importancia en áreas como la detección de anomalías, fraudes y defectos, visualización, reducción de la dimensionalidad, sistemas de recomendación... Las técnicas comúnmente aplicadas en estos problemas pueden agruparse principalmente en dos familias:

- **Métodos paramétricos:** en este caso se asume una distribución de probabilidad subyacente a los datos, es decir, los individuos provienen de una población que sigue una distribución probabilística con un conjunto de parámetros fijos. El aprendizaje no supervisado paramétrico requiere de la construcción de modelos, típicamente de mezcla gaussiana (*Gaussian Mixture Models*), y del uso de algoritmos EM (*Expectation-Maximization*).
- **Métodos no paramétricos:** al contrario que en los métodos paramétricos, no se requiere hacer suposiciones acerca de la distribución subyacente a los datos, por tanto, en ocasiones, se denominan *distribution-free methods*.

Dentro de la inmensidad de métodos y aproximaciones que se encuentran dentro del aprendizaje no supervisado, en este trabajo se pondrá el foco en el *clustering*. Su objetivo consiste en agrupar observaciones, compuestas por un conjunto de atributos o variables, únicamente atendiendo a criterios basados en sus diferencias y semejanzas. Los conjuntos obtenidos tras realizar el agrupamiento se denominan *clusters*, conglomerados o grupos.

El principal inconveniente que nos encontramos es la dificultad para definir lo que es un *cluster*. La imprecisión en dicha definición ha traído consigo gran cantidad de métodos.

2.1. Métodos de *Clustering* jerárquicos

En esta familia de métodos, la pertenencia a un *cluster* en un nivel específico de jerarquía condiciona su posible pertenencia en niveles superiores. La manera en la que se construye la jerarquía genera dos tipos de técnicas de *clustering*:

- *Agglomerativos:* su construcción es de abajo hacia arriba. Se parte de *clusters* formados por una única observación. Iterativamente, se van agrupando para formar *clusters* más grandes.

2 ANÁLISIS CLUSTER Y COMPONENTES PRINCIPALES

- *Divisivos*: su construcción es de arriba hacia abajo. Se parte de un *cluster* formado por la totalidad de los individuos. Iterativamente se va dividiendo dando lugar a conglomerados más pequeños según se va descendiendo en la jerarquía.

2.2. Métodos de *Clustering* no jerárquicos o particionales

Esta familia de métodos obtienen una única partición del espacio muestral. Las técnicas de agrupamiento pertenecientes a esta categoría son las más famosas y utilizadas. El máximo exponente es el algoritmo de las k -medias, aunque otros como *DBSCAN* también tienen su importancia.

Respecto al número de grupos en los que se desea fragmentar el conjunto de datos, la estrategia a seguir sigue dos líneas diferenciadas. Algunos de los algoritmos, como es el caso de k -medias, necesitan como condición previa a su uso que se especifique el número de *clusters*. Otros, como es el caso de *DBSCAN*, calculan automáticamente el número de agrupamientos, sin necesidad de que sean especificados previamente. Ambas aproximaciones tienen sus ventajas e inconvenientes, ofreciendo variedad y diversidad de soluciones para multitud de problemas a los que son aplicables este tipo de técnicas. En particular, en este trabajo, se va a exponer con más detalle el algoritmo de las k -medias, ya que guarda relación con el algoritmo que se va a implementar en el Capítulo 5. Concretamente, se trata de un caso particular, donde las dimensiones de los k subespacios afines entorno a los que agrupar son 0 (es decir, k puntos o centroides) y no recorta ninguna observación (ninguna observación es tratada como un punto atípico).

2.2.1. K -medias

El algoritmo de las k -medias es un método de análisis *cluster* particional que trata de segmentar el conjunto de datos $\mathcal{X} = \{x_1, \dots, x_n\}$ en k grupos disjuntos de tal manera que cada observación sea asignada a un único grupo (Figura 2.1). Los pasos de los que consta el algoritmo de las k -medias son los siguientes:

1. Especificar el número de *clusters* k y seleccionar k observaciones aleatorias del conjunto de datos. Estos k puntos son los denominados centroides iniciales.
2. Calcular la distancia euclídea entre cada observación perteneciente a \mathcal{X} y cada uno de los k centroides.
3. Asignar cada observación al *cluster* del que el centroide más cercano es representante.
4. Calcular los nuevos centroides como la media de todas las observaciones pertenecientes al mismo grupo.
5. Repetir los pasos 2, 3 y 4 hasta que los centroides no varíen en 2 iteraciones consecutivas.

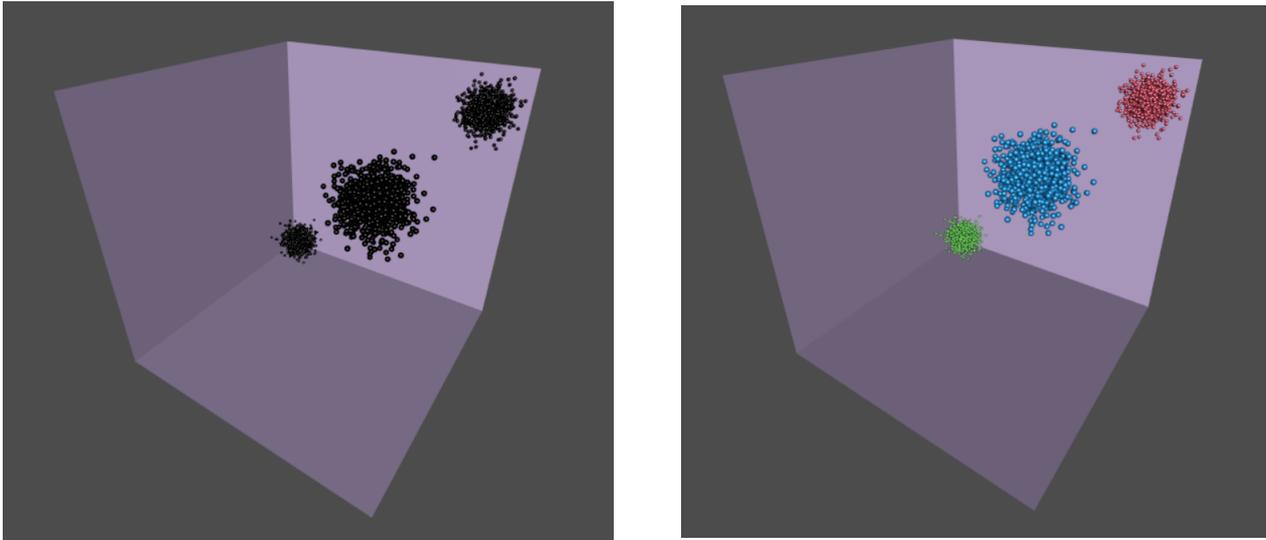


Figura 2.1: Agrupamiento realizados por k -medias ($k = 3$).

Sin embargo, este algoritmo presenta diversos problemas. Entre ellos destacan la sensibilidad a la elección inicial de los centroides, la necesidad de prefiar el número, k , de *clusters*, el hecho de que únicamente se encontrarán *clusters* convexos y la falta de robustez frente a observaciones atípicas.

Como ya se ha mencionado, el algoritmo de las k -medias es un caso particular del algoritmo que se va a detallar en este trabajo, con la configuración de parámetros $(d_1, \dots, d_k) = (0, \dots, 0)$ (representando d_j la dimensión del subespacio afín aproximante j -ésimo) y $\alpha = 0$ (representando la proporción de observaciones que serán recortadas), donde k es el número de *clusters*.

2.3. Análisis en Componentes Principales

Hay muchos problemas candidatos a la aplicación de modelos de aprendizaje automático en los que es necesario tratar con una cantidad enorme de variables medidas. La aplicación directa de algunos de estos algoritmos a conjuntos de datos con esta característica puede causar numerosos problemas.

Por un lado, al disponer de muchas variables explicativas, el número de parámetros del modelo será, por lo general, consecuentemente muy elevado. Para que las estimaciones de dichos parámetros sean suficientemente precisas se requiere una gran cantidad de observaciones, lo cual muchas veces no es posible.

Por otra parte, es más que sabido el peligro que conlleva el incluir demasiadas variables ex-

2 ANÁLISIS CLUSTER Y COMPONENTES PRINCIPALES

plicativas en un modelo, puede darse el problema del sobreajuste. Recordar que el sobreajuste aparece cuando se permite una gran complejidad del modelo ajustado, y éste parece ajustar de manera óptima los datos con los que se han estimado los parámetros. Sin embargo, la capacidad de generalización de un modelo con sobreajuste es mucho menor que lo que hace indicar su funcionamiento sobre los datos iniciales.

Otro problema encontrado prácticamente a diario en la estadística es el de la multicolinealidad. La presencia de un número excesivo de variables explicativas facilita que muchas de ellas tengan una gran correlación entre sí, es decir, multicolinealidad. Por tanto, se puede decir que la presencia de un número muy elevado de variables explicativas actúa como catalizador de la multicolinealidad.

Además, los requerimientos de memoria y capacidad de cómputo son más ambiciosos a medida que aumenta el tamaño de los conjuntos de datos, ya sea debido a un aumento de los casos, de las variables explicativas, o de ambos.

Por estos motivos, en muchos contextos y dominios de aplicación, es necesaria la utilización de técnicas de reducción de la dimensionalidad como paso previo al uso de otros métodos. Algunos clasificadores del paradigma de aprendizaje por *ensembles* como *Random Forest*, los filtros de baja varianza o los filtros de alta correlación pueden utilizarse con el fin de alcanzar este objetivo. Sin embargo, la técnica más conocida y utilizada, y la que se va a tratar con detalle en este trabajo, es el Análisis en Componentes Principales (ACP).

Las componentes principales de un conjunto de observaciones representadas en \mathbb{R}^p son un conjunto de p vectores directores ortogonales entre sí (Figura 2.2). La d -ésima componente principal vendría dada por el d -ésimo vector que mejor recoge la variabilidad de la muestra, siendo ortogonal a los $d - 1$ vectores anteriores. Por tanto, el primer vector será aquel que represente la dirección que recoja la máxima varianza de la nube de puntos.

El fundamento teórico básico que reside en este método es que la variabilidad total del conjunto de datos es invariante frente a la rotación de ejes ortogonales y existen rotaciones que hacen que gran parte de la información contenida en nuestros datos se encuentre disponible en la proyección de dichos datos en el subespacio generado por unos pocos vectores directores (Figura 2.3).

De manera más intuitiva, el Análisis en Componentes Principales puede verse como ajustar un elipsoide p -dimensional a los datos, donde los ejes del elipsoide representan las p componentes principales. Por tanto, si uno de los ejes del elipsoide es pequeño, entonces la proporción de varianza recogida por ese eje también será pequeña.

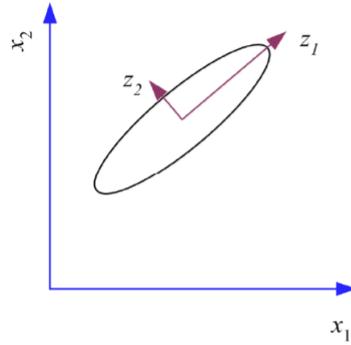


Figura 2.2: Representación esquemática de las componentes principales z_1 y z_2 de una nube de puntos elíptica en \mathbb{R}^2 .

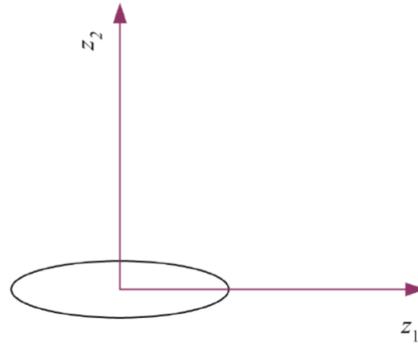


Figura 2.3: Rotación de los datos en el sistema de coordenadas de las componentes principales z_1 y z_2 .

Cada componente principal se obtiene por combinación lineal de las variables originales. Sea (X_1, X_2, \dots, X_p) un vector p -dimensional con las variables explicativas, entonces la primera componente principal Z_1 es la combinación lineal de dichas variables que recoja la mayor cantidad de varianza:

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \dots + \phi_{p1}X_p$$

donde

$$\sum_{j=1}^p \phi_{j1}^2 = 1$$

debido a que la combinación lineal es normalizada. De forma análoga:

2 ANÁLISIS CLUSTER Y COMPONENTES PRINCIPALES

$$Z_d = \phi_{1d}X_1 + \phi_{2d}X_2 + \dots + \phi_{pd}X_p$$

sería la combinación lineal de dichas variable que recoge la d -ésima mayor variabilidad.

Los términos ϕ_{ij} se conocen como *loadings*, y son los pesos en las variables que caracterizan a la componente principal. Cuando es necesario interpretar el significado de una componente principal estos valores son muy útiles, ya que permiten conocer el peso que tiene cada variable independiente en la componente, ayudando a determinar que tipo de información recoge la misma.

Para calcular el valor de los *loadings*, y por tanto la componente principal, es necesario resolver un problema de optimización para maximizar la variabilidad recogida por la componente. En la práctica, esta operación es equivalente a obtener los autovectores de la matriz de varianzas-covarianzas muestral del conjunto de datos ordenados según los autovalores.

Se calcula la matriz de varianzas-covarianzas del conjunto de datos $\mathcal{X} \in \mathbb{R}^{n \times p}$ como:

$$\mathcal{S} = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})(x_i - \bar{x})^T$$

donde x_i es una fila de la matriz de datos \mathcal{X} y \bar{x} contiene las medias aritméticas por columnas de las p variables. Entonces, la descomposición en autovalores y autovectores de la matriz de varianzas-covarianzas puede hacerse de la siguiente manera:

$$\mathcal{S}\mathcal{V} = \mathcal{V}\lambda$$

donde \mathcal{V} es una matriz cuyas columnas son los autovectores de \mathcal{S} , y λ una matriz diagonal con los autovalores. Entonces:

$$\mathcal{S} = \mathcal{V}\lambda\mathcal{V}^{-1}$$

Por tanto, la primera columna de \mathcal{V} asociada al autovalor más alto sería el autovector que nos marca la dirección de la primera componente principal. La columna asociada al segundo autovalor más alto marca la dirección de la segunda componente principal, y así hasta completar las p componentes principales.

Al igual que sucedía con el algoritmo de las k -medias, el Análisis en Componentes Principales es otro caso particular del algoritmo que se trata e implementa en este trabajo. Las componentes principales se obtienen, como caso particular simple, del enfoque adoptado cuando $k=1$. Es decir, si no se busca ninguna estructura de clusters y todas las observaciones están en un mismos cluster.

3. Robustez

Como ya se ha comentado al comienzo del Capítulo 2, diariamente se genera una cascada inmensa de datos. Una parte de esos datos se puede modelizar razonablemente bien por unas pocas distribuciones de probabilidad muy conocidas, permitiendo trabajar con ellos y explotarlos mediante la aplicación de métodos que aprovechan las suposiciones a nivel poblacional realizadas sobre la distribución subyacente que ha generado los datos. Por tanto, en esta situación, los procedimientos desarrollados y estudiados en la estadística clásica son perfectamente válidos.

Sin embargo, en aplicaciones de análisis reales de datos es muy cuestionable el poder justificar el cumplimiento de las hipótesis sobre las distribuciones probabilísticas en las que se basan muchos de estos métodos. Algunos conjuntos de datos presentan ruido, es decir, las observaciones presentan una variabilidad que no puede ser explicada por los modelos que se tratan de ajustar. En general, es imposible evitar la presencia de ruido en un conjunto de datos, ya que se extraen de entornos de producción donde esta característica es innata. La presencia de un ruido o contaminación excesivos puede dañar irreversiblemente un conjunto de datos, y hacerlo inutilizable.

Otros conjuntos de datos se caracterizan por la presencia de valores atípicos o *outliers*, observaciones significativamente diferentes del resto. Por ejemplo, estas observaciones atípicas aparecen cuando nos enfrentamos a datos generados desde distribuciones con colas pesadas (*heavy tailed*) o incluso cuando aparecen puntos que, sin ser extremos, se separan del comportamiento mayoritario del resto de observaciones. La presencia de valores atípicos puede deberse o bien a variabilidad intrínseca de la muestra o bien a un error experimental, es decir, un error de medición, de transcripción... En este último caso, y si se comprueba el error, se suele optar por eliminar el *outlier*, siempre y cuando ese valor no pueda ser corregido.

Los valores atípicos pueden causar serios problemas en el análisis estadístico de los datos. De ahí, surge la necesidad del desarrollo de técnicas estadísticas que sean robustas frente a la presencia de *outliers*. Un método no robusto frente a *outliers* ofrecerá un resultado diferente en función de si los datos que toma como entrada presentan o no este tipo de puntos. La solución calculada cuando hay presencia de valores atípicos no suele ser adecuada. Es lo que se busca precisamente con el desarrollo de este tipo de técnicas, un método robusto frente a *outliers* ofrecerá un resultado similar, y razonable la mayoría de las veces, tanto si los datos que toma como entrada presentan valores atípicos como si no.

Por tanto, se puede concluir que un método estadístico robusto es un procedimiento que mantiene un buen funcionamiento para datos provenientes de una amplia gama de distribuciones de probabilidad, si el modelo que ha generado los datos es próximo a las hipótesis supuestas ideales, aunque éstas hipótesis no se cumplan exactamente. Los métodos no robustos pueden tener un funcionamiento muy malo incluso con muestras de datos que han sido generadas por modelos muy

parecidos al modelo asumido pero que no coinciden con dicho modelo.

3.1. Media y mediana

Una primera y sencilla aproximación para comprender qué diferencias hay entre estadísticos que carecen o presentan robustez estadística es la diferencia entre la media y la mediana. Sea x_1, x_2, \dots, x_{40} una muestra proveniente de una distribución normal multivariante tomando valores en \mathbb{R}^3 . Para simular una situación en la que varios casos de un conjunto de datos presentan ruido, a las 8 últimas observaciones se les añadirá en cada una de las 3 componentes una cantidad aleatoria de ruido proveniente de una distribución uniforme continua (Figura 3.1).

$$x_1, x_2, \dots, x_{32} \sim N \left[\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \right]$$

$$x_{33}, x_{34}, \dots, x_{40} \sim N \left[\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \right] + (U(15, 20), U(15, 20), U(15, 20))'$$

Al generar los datos siempre supondremos que todas las distribuciones que aparezcan han sido generadas de forma independiente entre sí.

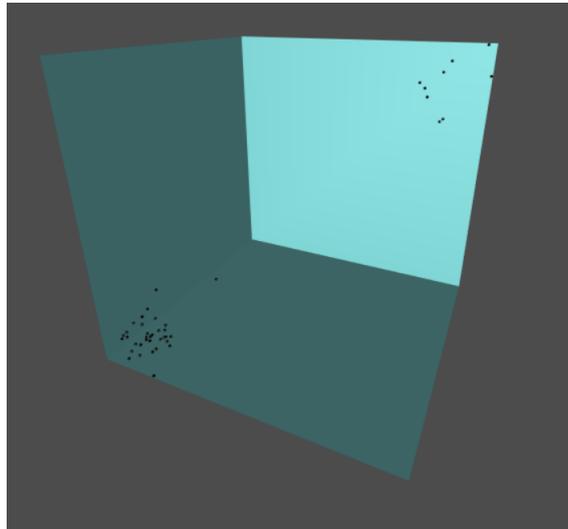


Figura 3.1: Representación de los datos x_1, \dots, x_{40}

La media aritmética es una medida sensible a la presencia de valores atípicos y/o ruido, por lo que es un estadístico no robusto. Si se computa la media aritmética en cada una de las componentes de \mathbb{R}^3 en el conjunto de las 40 observaciones, se obtiene que:

$$\bar{x} = \left(\frac{\sum_{i=1}^{40} x_{i,1}}{40}, \frac{\sum_{i=1}^{40} x_{i,2}}{40}, \frac{\sum_{i=1}^{40} x_{i,3}}{40} \right) = (3.61, 3.84, 3.44)$$

donde $x_i = (x_{i,1}, x_{i,2}, x_{i,3})'$.

El estadístico, si fuese robusto, debería resumir las características de la parte mayoritaria de las observaciones, es decir, aquellas a las que no se les ha añadido una cantidad aleatoria de ruido proveniente de una distribución uniforme. Dichos valores, al provenir de una distribución normal multivariante con media 0 en todas sus componentes y matriz de varianzas-covarianzas la matriz identidad de tamaño 3×3 , deberían ser mucho menores que los valores que toma, y cercanos al 0. Recordar que únicamente el 20% de los puntos han sido modificados añadiéndoles ruido. Los valores atípicos, lejos de tener un papel secundario en el resultado de la métrica, son muy influyentes, y provocan que el valor que toma la métrica no sea adecuado.

La mediana M_e es una medida de la tendencia central, parte una distribución de valores en dos zonas de igual frecuencia. Es decir, deja a ambos lados del corte el mismo número de valores. Si se calcula la mediana coordenada a coordenada de nuestras observaciones se tiene el estadístico mediana por coordenadas (*Coordinate M_e*):

$$\text{Coordinate } M_e = (M_e(x_{i,1}_{i=1}^n), M_e(x_{i,2}_{i=1}^n), M_e(x_{i,3}_{i=1}^n)) = (0.50, 0.29, 0.18)$$

La mediana espacial, también llamada mediana geométrica o *1-median*, es el punto que minimiza la suma de distancias a las observaciones de la muestra. Esta medida, generaliza la mediana clásica, que tiene la propiedad de minimizar la suma de distancias para datos unidimensionales. Por tanto, la mediana espacial es una medida de la tendencia central para datos en \mathbb{R}^p con $p > 1$. La manera de calcularla es:

$$\text{Spatial } M_e = \arg \min_{y \in \mathbb{R}^3} \sum_{i=1}^{40} \|x_i - y\| = (0.12, 0.06, -0.12)$$

En este caso se observa que los valores obtenidos, tanto para la mediana por coordenadas como para la mediana espacial, sí guardan relación con lo esperado, estando más próximos al 80% de observaciones sin presencia de ruido que al 20% que sí. El código **R** para generar los datos aleatorios y computar las 3 métricas puede verse a continuación.

3 ROBUSTEZ

```
library(MASS)
library(Gmedian)
x<-mvrnorm(40, mu = c(0,0,0), Sigma = diag(3))
x[32:40,]<-x[32:40,]+runif(3,15,20)
Mean<-apply(x,2,mean)
CoordinateMedian<-apply(x,2,median)
SpatialMedian<-Gmedian(x, nstart = 10)
```

Gráficamente se aprecia el comportamiento de ambos enfoques de una manera muy clara (Figura 3.2). En naranja se representa la media, que se encuentra a medio camino entre el cúmulo principal de observaciones y los 8 puntos aislados de la esquina superior derecha. La mediana por coordenadas y la mediana espacial están representadas en azul y en rojo respectivamente, ambas se enclavan prácticamente en el centro de la nube de puntos. Al ser la mediana una métrica robusta su comportamiento apenas se ve afectado por la presencia de valores con ruido que van en disonancia con la tendencia global del conjunto de observaciones.

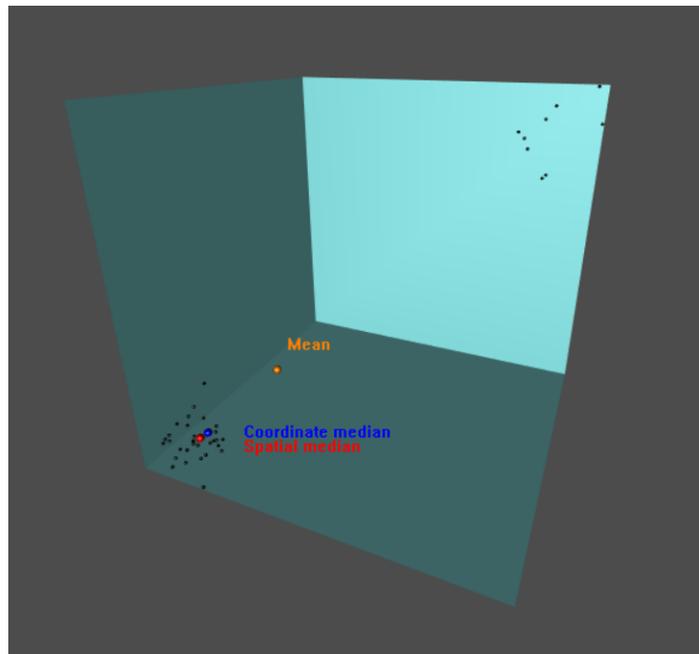


Figura 3.2: Representación de los datos x_1, \dots, x_{40} junto con su media, mediana por coordenadas y mediana espacial.

Con este primer ejemplo del comportamiento de un estadístico robusto frente a uno que no lo es se pretende mostrar la importancia que tiene el desarrollo y uso de técnicas robustas cuando los datos que se utilizan son susceptibles de presentar ruido o puntos atípicos. Los *outliers*, si no son tenidos en cuenta y tratados detenidamente, pueden estropear multitud de métodos que no están diseñados para soportarlos.

3 ROBUSTEZ

observaciones provenientes de la distribución normal multivariante con vector de medias $(\frac{5}{2}, \frac{5}{2}, \frac{5}{2})$. La agrupación que realiza el algoritmo de las k -medias con $k = 2$ se muestra en la Figura 3.4.

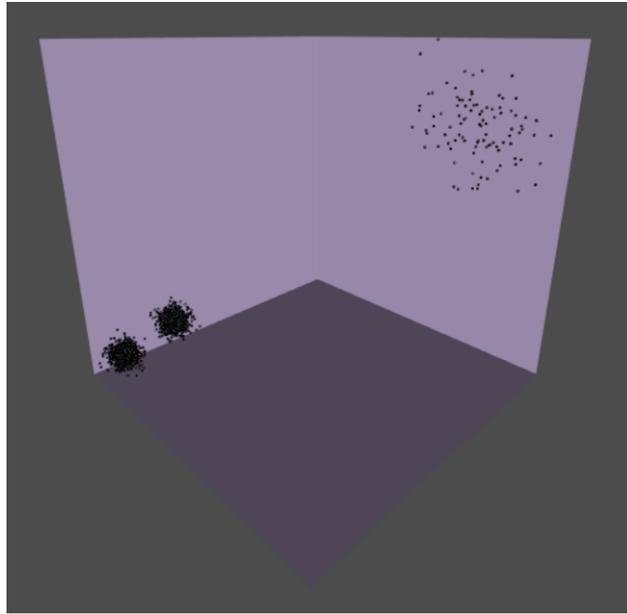


Figura 3.3: Representación de los datos x_1, \dots, x_{1800} .

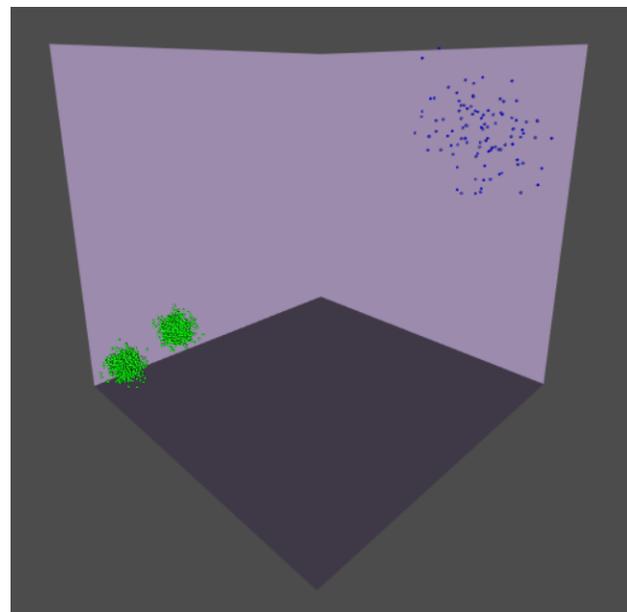


Figura 3.4: Agrupaciones realizadas por las k -medias.

La agrupación realizada por el algoritmo no es la deseada. Bajo el primero de los grupos que encuentra engloba las 1900 observaciones provenientes de las 2 distribuciones normales multivariantes con vectores de medias $(0,0,0)$ y $(\frac{5}{2}, \frac{5}{2}, \frac{5}{2})$. El segundo grupo está formado por la muestra proveniente de la tercera distribución, con vector de medias $(15,15,15)$. Las 100 observaciones extraídas aleatoriamente de esta distribución representan el 5% del total de observaciones del conjunto de datos, y sin embargo condicionan de una manera determinante la solución ofrecida por las k -medias, hasta el punto de que ésta no es razonable.

Como ya se ha comentado, parece lógico pensar que las 950 observaciones provenientes de la normal con vector de medias $(0,0,0)$ y la muestra de 950 observaciones provenientes de la normal con vector de medias $(\frac{5}{2}, \frac{5}{2}, \frac{5}{2})$ se engloben en *clusters* distintos, ya que representan el 90% de las observaciones totales del conjunto de datos. Es deseable que el 10% de observaciones restantes tengan un peso residual en las decisiones tomadas por el algoritmo de las k -medias, y no condicionen la estimación de las coordenadas donde se sitúan los centroides.

Este conjunto de datos simulado artificialmente permite mostrar como el algoritmo de las k -medias, al hacer uso de un estadístico poco robusto frente a las observaciones atípicas y al ruido como lo es la media para calcular las coordenadas en las que se encuentran los centroides, no es robusto. Cuando en el conjunto de datos sobre el que se aplica hay presencia de valores atípicos la solución no converge a la deseada, realizando agrupamientos de observaciones poco razonables, y como catalizador del problema únicamente actúan una pequeña proporción de observaciones.

Se podría pensar que se trata de un problema con 3 *clusters*, pero nótese que las k -medias solo buscan grupos esféricos con la misma dispersión. Si hubiéramos considerado una dispersión mayor en el ruido (generando un ruido de fondo muy disperso) entonces se hubiera necesitado buscar un número muy grande grupos.

La falta de robustez de las k -medias parece esperable por usar normas al cuadrado en la función objetivo. Si buscáramos una robustificación eliminando la penalización cuadrática en la norma, tal como hace la mediana respecto a media, se podría ver que la k -mediana produce una muy tímida robustificación.

Como solución a este problema, y dotando de robustez estadística al algoritmo de las k -medias, Cuesta-Albertos *et al.* (1997) [1] introducen el método *trimmed k-means* o k -medias recortadas, el cual permite que un proporción α de observaciones queden sin ser asignadas a ninguno de los grupos propuestos por el algoritmo, es decir, se evita que determinadas observaciones, anteriormente muy influyentes en el cálculo de los centroides, tengan influencia en el cálculo de las coordenadas de los centroides de los grupos.

Sea x_1, x_2, \dots, x_n un conjunto de datos en \mathbb{R}^p , $\alpha \in [0, 1]$ la proporción de observaciones que serán recortadas y $k \in \mathbb{N}$ el número de *clusters* bajo los que se quiere agrupar las observaciones de

3 ROBUSTEZ

la muestra. Los pasos de los que consta el algoritmo de las k -medias recortadas son los siguientes:

1. Seleccionar k observaciones aleatorias del conjunto de datos. Estos k puntos son los denominados centroides iniciales.

$$m_1, m_2, \dots, m_k \in \mathbb{R}^p$$

2. Calcular la distancia euclídea entre cada observación del conjunto de datos y cada uno de los k centroides. Para cada observación, quedarse con la distancia más pequeña de las k calculadas.

$$d_i = \inf_{j=1, \dots, k} \|x_i - m_j\|^2, \quad i = 1, \dots, n$$

3. Determinar el conjunto C compuesto por las $[n(1 - \alpha)]$ observaciones con menor d_i .
4. Particionar C en $C = \{C_1, \dots, C_k\}$, donde los puntos en C_j son aquellas observaciones que están más próximas al centroide m_j que a cualquier otro centroide m_l , $j \neq l$.

$$C_j := \{x_i \in C : \|x_i - m_j\|^2 = d_i\}$$

5. Calcular los nuevos centroides como la media de todas las observaciones pertenecientes al *cluster* del que el centroide es representante.

$$m_j = \frac{\sum_{i \in C_j} x_i}{|C_j|}, \quad j = 1, \dots, k$$

6. Repetir los pasos 2, 3, 4 y 5 hasta que los centroides no varíen en 2 iteraciones consecutivas.

Las agrupaciones que realiza el algoritmo de las k -medias recortadas, al igual que sucede con el algoritmo de las k -medias son fuertemente dependientes de los centroides iniciales elegidos aleatoriamente. Distintas configuraciones iniciales de los centroides pueden inducir a distintas soluciones. Por ello, para conseguir una agrupación óptima que no dependa de dicha elección inicial, se suele reiniciar el procedimiento varias veces, eligiendo aquella inicialización que alcance el menor valor de una función objetivo tras el proceso iterativo. La función objetivo a minimizar en el caso de las k -medias recortadas es:

$$\frac{1}{[n(1 - \alpha)]} \sum_{j=1}^k \sum_{x_i \in C_j} \|x_i - m_j\|^2$$

Si aplicamos las k -medias recortadas con el parámetro de recorte $\alpha = 0.05$ al conjunto de datos generado a partir de 3 distribuciones normales multivariantes donde el algoritmo de las k -medias no ofrecía una solución óptima, nos encontramos con que ahora los agrupamientos realizados sí son razonables (Figura 3.5).

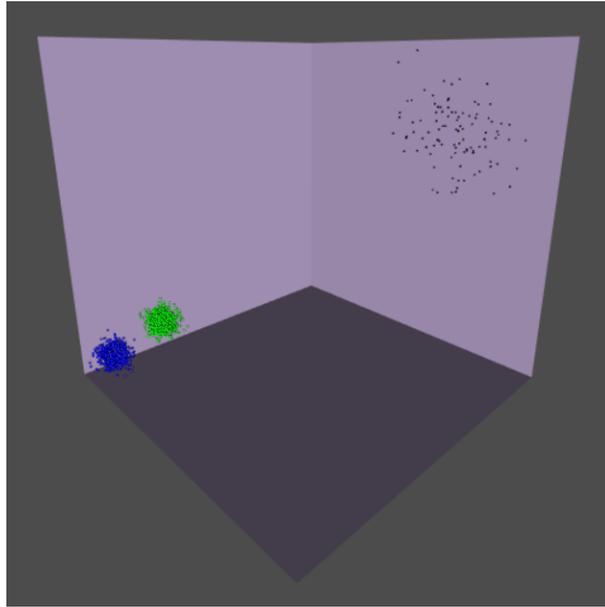


Figura 3.5: Agrupaciones realizadas por las k -medias recortadas ($k = 2$, $\alpha = 0.05$).

Bajo el primero de los grupos se engloban las 950 observaciones provenientes de la distribución normal multivariante con vector de medias $(0,0,0)$, mientras que en el segundo grupo se encuentran las 950 observaciones provenientes de la distribución normal multivariante con vector de medias $(\frac{5}{2}, \frac{5}{2}, \frac{5}{2})$. Las 100 observaciones extraídas aleatoriamente de la última distribución normal multivariante, el 5% del total, son dejadas sin asignar a ningún grupo (graficadas en color negro), por lo que el algoritmo consigue aislarlas y que no intervengan en la determinación de los centroides de los *clusters*.

Este ejemplo sirve para mostrar como el procedimiento de las k -medias recortadas es más robusto a las observaciones atípicas que el algoritmo de k -medias clásico. Las k -medias recortadas han probado contar con buenas propiedades de robustez como se puede ver en [2].

3.3. Aplicación de los recortes al ACP

Al igual que sucede con el algoritmo de las k -medias, el Análisis en Componentes Principales es otro método susceptible de ofrecer malos resultados cuando los datos sobre los que es aplicado presentan valores extremos. Para la determinación de la dirección de máxima varianza de un conjunto de observaciones todos los puntos influyen de la misma manera, y unas pocas observaciones alejadas de la nube de puntos principal pueden acabar desviando dicha dirección de la que cabría esperar únicamente teniendo en cuenta a la nube de puntos principal.

Para ilustrar este fenómeno, se diseña un experimento generando datos artificialmente al igual

3 ROBUSTEZ

que se hizo en el caso de la k -medias. Por ejemplo, supóngase que se tienen 2 submuestras, cada una de ellas proveniente de una distribución normal multivariante tal y como se especifica a continuación:

$$x_1, x_2, \dots, x_{950} \sim N \left[\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 10 & 0 & 0 \\ 0 & \frac{1}{20} & 0 \\ 0 & 0 & \frac{1}{20} \end{pmatrix} \right]$$
$$x_{951}, x_{952}, \dots, x_{1000} \sim N \left[\begin{pmatrix} 10 \\ 10 \\ 10 \end{pmatrix}, \begin{pmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{pmatrix} \right]$$

El primer conjunto está compuesto por una muestra de 950 observaciones, y el segundo conjunto está compuesto por una muestra de 50 observaciones. Los 2 conjuntos se unen en un mismo conjunto de datos, teniendo así 1000 puntos en \mathbb{R}^3 provenientes de 2 distribuciones distintas. La representación gráfica del conjunto de datos se aprecia en la Figura 3.6.

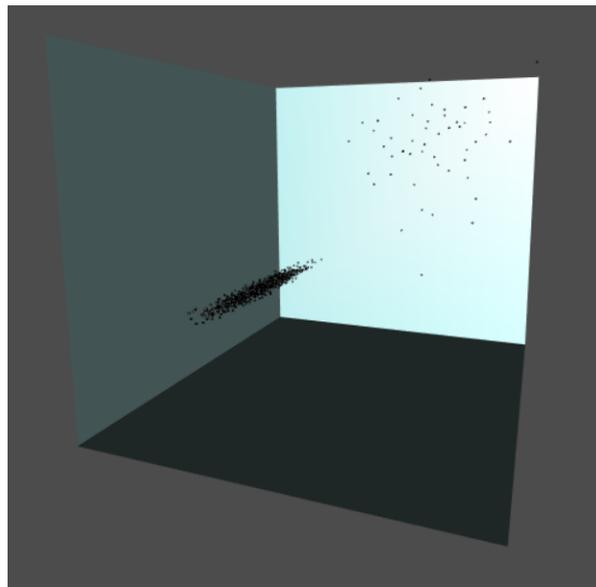


Figura 3.6: Representación del conjunto de datos x_1, \dots, x_{1000} .

Cuando se realiza un ACP sobre el conjunto de datos, se observa que el vector que marca la dirección de máxima varianza de los datos es $\vec{v} = (0.8022, 0.4205, 0.4239)$, representado con una línea azul en la Figura 3.7. Analizando gráficamente la dirección de máxima varianza se aprecia como, lejos de recoger únicamente la información de la nube de puntos principal formada

por las 950 observaciones provenientes de la primera distribución normal, las 50 observaciones extraídas de una normal con vector de medias $(10,10,10)$ son determinantes a la hora de calcular el vector director. La recta cuya dirección es marcada por el vector está claramente inclinada hacia los 50 puntos provenientes de la segunda distribución normal, con lo que se concluye que estas observaciones están influyendo de manera clara en la determinación de la dirección de máxima varianza.

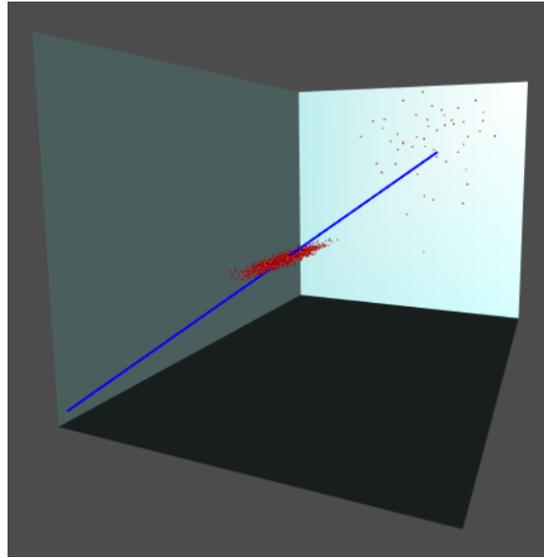


Figura 3.7: Dirección de máxima varianza encontrada por ACP.

Queda claro que es necesario disponer de un procedimiento de cálculo de las componentes principales que sea resistente a las observaciones atípicas, y que en este ejemplo sea capaz de determinar la dirección de máxima dispersión de la nube principal de puntos, sin que observaciones secundarias intervengan en dicha decisión. Para ello, se puede generalizar el algoritmo de las k -medias recortadas de tal manera que, en vez de buscar k grupos, únicamente se busque uno para determinar la dirección de máxima varianza. Las agrupaciones se realizarán entorno a un subespacio afín de dimensión 1, una recta, en vez de un subespacio afín de dimensión 0 como sería un centroide.

Este procedimiento de Análisis en Componentes Principales recortadas es un caso particular del algoritmo que se detallará en el Capítulo 4, al igual que las k -medias recortadas. Concretamente, en el caso de las k -medias recortadas la configuración de parámetros sería $(d_1, \dots, d_k) = (0, \dots, 0)$ (k subespacios afines de dimensión 0 donde k es el número de grupos) y $\alpha > 0$ (la proporción de observaciones a recortar), mientras que en el caso del ACP con recortes la relación de parámetros sería $d = 1$ y $\alpha > 0$.

Si aplicamos el procedimiento con el parámetro de recorte $\alpha = 5\%$ al conjunto de datos

3 ROBUSTEZ

generado a partir de 2 distribuciones normales multivariantes donde el ACP no ofrecía una solución óptima nos encontramos con que ahora la dirección de máxima varianza encontrada si es la esperada, como se aprecia en la Figura 3.8.

La dirección de máxima variabilidad del conjunto de puntos encontrada es mucho más razonable que la propuesta por el la técnica clásica de Análisis en Componentes Principales, debido a que los 50 puntos provenientes de la distribución normal multivariante con vector de medias (10,10,10) no son utilizados para determinar la recta que mejor resume la varianza de la muestra. Al fijar el parámetro de recorte $\alpha = 0.05$ se consigue que esas 50 observaciones no modifiquen la dirección de máxima varianza que encontraría el ACP aplicándose únicamente sobre los 950 puntos provenientes de la distribución normal con vector de medias (0,0,0), y que representa el 95% del conjunto de datos. Frente al vector que marca la dirección de máxima varianza de los datos $\vec{v} = (0.8022, 0.4205, 0.4239)$ encontrado originalmente, el procedimiento robusto encuentra el vector $\vec{v} = (0.9999, -0.0037, 0.0008)$, la variación y la mejoría del ajuste es evidente.

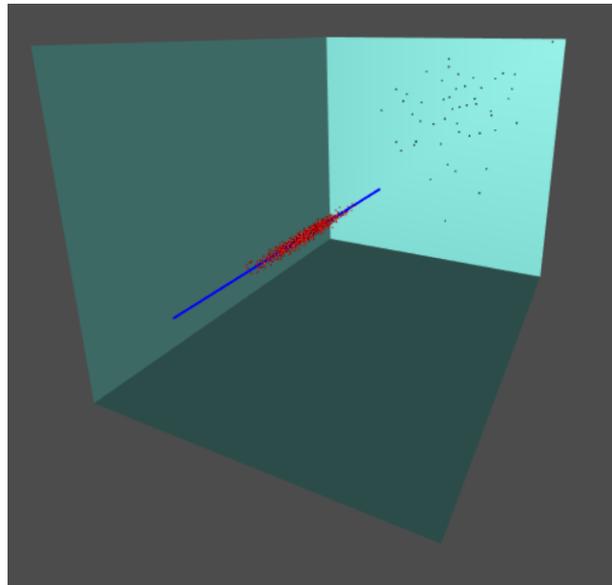


Figura 3.8: Dirección de máxima varianza encontrada por ACP con recortes.

4. Clustering robusto entorno a subespacios lineales

Bajo los métodos de análisis *cluster* no jerárquicos es común que resida el concepto de formar grupos o conglomerados en torno a puntos centrales, los cuáles representan el comportamiento típico de las observaciones pertenecientes al grupo del que dichos puntos son representantes. El máximo exponente de esta familia de métodos es el algoritmo de las k -medias, basado en el criterio de los mínimos cuadrados.

Este criterio presenta mayoritariamente dos problemas. El primero de ellos es la falta de robustez estadística y que ya hemos comentado en secciones anteriores.

El segundo de los problemas que presentan esta familia de métodos es que tienen tendencia a encontrar *clusters* esféricos debido a que realizan la suposición de que el comportamiento de un conglomerado de puntos puede ser resumido por una observación central. Sin embargo, en ocasiones la presencia de agrupaciones en un conjunto de datos es debida a la existencia de relaciones entre las variables medidas que no tienen por qué responder a patrones esféricos. Por ejemplo, dada una muestra de observaciones, se pueden encontrar diferentes estructuras lineales como líneas, planos o hiperplanos alrededor de los cuáles las observaciones pueden ser agrupadas de una manera natural, sin forzar a crear *clusters* inducidos por criterios que favorecen el agrupamiento en estructuras esféricas.

Los primeros intentos de realizar análisis *cluster* entorno a subespacios lineales vinieron por parte de Hosmer (1984), Lenstra *et al.* (1982) y Späth (1982), ajustando una mixtura de dos modelos de regresión lineal simple. Para el problema de estructuras lineales bidimensionales, aproximaciones alternativas fueron presentadas por Murtagh y Raftery (1984) y Phillips y Rosenfeld (1988). DeSarbo y Cron (1988) presentaron una solución para un número general de dimensiones y un número arbitrario de grupos.

Van Aelst *et al.* (2006) [3] abordaron el problema del agrupamiento lineal usando una aproximación basada en regresión ortogonal. Dicho planteamiento obtuvo muy buenos resultados en diversos problemas donde no había presencia de observaciones atípicas. Bajo esta aproximación y buscando un única agrupación, el problema se reduce al Análisis en Componentes Principales clásico, que como ya se ha mostrado en esta memoria también sufre serios problemas de robustez. En muchas de las potenciales áreas de aplicación como la visión artificial, el reconocimiento de patrones o la tomografía la presencia de ruido en los conjuntos de datos es muy frecuente, por lo que urgía mejorar las aproximaciones al agrupamiento entorno a estructuras lineales para que fuesen robustas frente al ruido y los *outliers*.

4.1. Agrupaciones robustas entorno a subespacios afines d -dimensionales

García-Escudero *et al.* (2009) [4] introducen una metodología que busca agrupamientos entorno a estructuras lineales en presencia de observaciones atípicas, dotando de robustez al algoritmo de agrupamiento lineal basado en regresión ortogonal en [3]. El método está basado en la idea de *impartial trimming* (Gordaliza (1991) [5], Cuesta-Albertos *et al.* (1997) [1]), es decir, son los propios datos los que deciden cuáles han de ser las observaciones que tienen que ser eliminadas. El método permite realizar agrupaciones entorno a subespacios afines generales d -dimensionales, no únicamente entorno a líneas rectas. Además, como ya se ha comentado, la robustez estadística se incluye de una manera muy natural utilizando el criterio de los mínimos cuadrados recortados.

Dada una muestra x_1, \dots, x_n de observaciones en \mathbb{R}^p , $0 \leq \alpha < 1$ (la proporción de observaciones que serán recortadas), d (la dimensión de los subespacios afines siendo $0 \leq d < p$) y $k \in \mathbb{N}$ (el número de grupos que se pretenden buscar), se busca la solución al siguiente problema de minimización:

$$\min_{Y \subset \{x_1, \dots, x_n\}, \#Y = [n(1-\alpha)]} \min_{\{h_1, \dots, h_k\} \in A_d} \left(\frac{1}{[n(1-\alpha)]} \sum_{x_i \in Y} \min_{j=1, \dots, k} \{\|x_i - Pr_{h_j}(x_i)\|^2\} \right) \quad (1)$$

donde $A_d := \{h \subset \mathbb{R}^p, h \text{ es un subespacio afín } d\text{-dimensional}\}$ y $Pr_h(\cdot)$ denota la proyección ortogonal en h .

Cualquier solución $\mathcal{H}^0 = \{h_1^0, \dots, h_k^0\}$ del problema induce a una partición de las observaciones no recortadas en k agrupaciones entorno a subespacios afines, de tal manera que la agrupación C_j está formada por todas las observaciones no recortadas que están más cerca de h_j^0 que de cualquiera de los otros $k - 1$ subespacios en \mathcal{H}^0 .

4.1.1. Caso I: problema teórico o poblacional

Asumiendo que $\{x_1, \dots, x_n\}$ es el resultado de una muestra aleatoria X_1, \dots, X_n de una distribución de probabilidad \mathcal{P} , el problema empírico o muestral en (1) admite un homólogo teórico o poblacional. Sea \mathcal{P} una distribución de probabilidad continua en \mathbb{R}^p , $\alpha \in (0, 1)$ y $k \in \mathbb{N}$. Para cada $\mathcal{H} = \{h_1, \dots, h_k\} \subset A_d$ y cada conjunto A tal que $\mathcal{P}(A) = 1 - \alpha$, se mide la k -variación entorno a \mathcal{H} dado A como:

$$V_A(\mathcal{H}) := \frac{1}{1 - \alpha} \int_A d(x, \mathcal{H})^2 d\mathcal{P}(x)$$

con $d(x, \mathcal{H}) = \min_{j=1, \dots, k} \|x - Pr_{h_j}(x)\|$.

Entonces, se obtiene la k -variación dado A haciendo una minimización en \mathcal{H} :

$$V_A := \inf_{\mathcal{H} \subset A_d, \#\mathcal{H}=k} \{V_A(\mathcal{H})\}$$

y finalmente se obtiene la k -variación recortada con recorte α minimizando en A :

$$V_{k,\alpha} := \inf_{A:\mathcal{P}(A)=1-\alpha} (V_A)$$

Resolviendo este doble problema de minimización, se consigue un conjunto óptimo A_0 y k subespacios afines óptimos $\mathcal{H}^0 = \{h_1^0, \dots, h_k^0\}$ tal que $V_{A_0}(\mathcal{H}_0) = V_{k,\alpha}$

Dado que se asume que no existe ninguna variable explicativa que ha de ser resumida por otro conjunto de variables se utilizan distancias ortogonales para medir las discrepancias entre los puntos y los hiperplanos. Esto es, se utiliza el criterio de la regresión ortogonal y no el de la regresión clásica. Además, estas distancias no están escaladas, por lo que se asume igualdad de varianza a lo largo de todos los subespacios. Esta metodología fue extendida en [6] al caso en que sí existe una variable respuesta privilegiada y se usan los residuales típicamente utilizados en Análisis de Regresión.

4.1.2. Caso II: problema empírico o muestral

Siendo $\{X_n\}_n$ una secuencia de vectores aleatorios independientes e igualmente distribuidos extraídos de la distribución \mathcal{P} , la distribución empírica se define como:

$$\mathcal{P}_n(A) = \frac{1}{n} \sum_{i=1}^n I_A(X_i)$$

El problema original (1) tiene un desarrollo idéntico al mostrado en el caso poblacional, pero sustituyendo la distribución desconocida \mathcal{P} por la distribución empírica \mathcal{P}_n . Fijados $X_1 = x_1, \dots, X_n = x_n$, el resultado de existencia de soluciones probada en el caso de distribuciones teóricas \mathcal{P} puede ser utilizada para derivar fácilmente la existencia de soluciones en el caso empírico. Existen formas óptimas de dividir $\{x_1, \dots, x_n\}$ en k grupos de tal manera que el número total de elementos en el total de los grupos sea $[n(1 - \alpha)]$. Para cada una de las particiones, los k subespacios afines óptimos se obtienen recurriendo a la regresión ortogonal de las observaciones en cada grupo.

La referencia [4] muestra una prueba de la convergencia de las soluciones muestrales a las soluciones del problema poblacional, la convergencia entre subespacios afines ha de ser vista como la convergencia de distancias al origen y la posibilidad de elegir secuencias convergentes de los vectores generadores de los subespacios (*spanning vectors*).

4.2. Descripción del algoritmo

El cálculo de los k subespacios afines α -recortados óptimos en el problema empírico tiene una alta complejidad computacional, debido a que para alcanzar ese óptimo es necesario realizar una búsqueda en un espacio combinatorio de subconjuntos, un espacio que generalmente es demasiado

4 CLUSTERING ROBUSTO ENTORNO A SUBESPACIOS LINEALES

amplio. Por tanto, un algoritmo que encuentre la solución óptima siempre realizando una búsqueda de fuerza bruta no es factible, y se abre el camino para un método que encuentre una aproximación adecuada al óptimo.

El algoritmo que se va a describir y posteriormente implementar es una adaptación del propuesto para realizar el cálculo de las k -medias recortadas por García-Escudero *et al.* (2003) [7], y debe ser visto como una combinación del algoritmo de las k -medias clásico y del algoritmo FAST-MCD en Rousseeuw y Van Driessen (1999) [8] para calcular el estimador de mínimo determinante de la covarianza (MCD).

En las k -medias recortadas y en FAST-MCD un paso de concentración es requerido donde se seleccionan las observaciones con menores distancias euclídeas a sus respectivos centros, en este algoritmo se reservan las $[n(1 - \alpha)]$ observaciones con menor distancia ortogonal al subespacio más cercano de entre los k subespacios afines calculados en la iteración anterior. Tras este paso, se obtienen k nuevos subespacios afines resolviendo k problemas de regresión ortogonal.

Dada una muestra $\{x_1, \dots, x_n\}$ de observaciones $\in \mathbb{R}^p$, $0 \leq \alpha < 1$ (la proporción de observaciones que serán recortadas), d (la dimensión de los subespacios afines siendo $0 \leq d < p$), $A_d := \{h \subset \mathbb{R}^p, h \text{ es un subespacio afín } d\text{-dimensional}\}$ y $k \in \mathbb{N}$ (el número de grupos que se pretenden buscar), el algoritmo consta de los siguientes pasos:

1. Realizar un escalado de las variables para evitar problemas de precisión numérica. El método de escalado elegido es robusto, dividiendo cada variable entre su MAD (*Median Absolute Deviation*).
2. Seleccionar k subespacios afines iniciales en A_d . Por ejemplo, seleccionar aleatoriamente $(d + 1) \times k$ observaciones del conjunto de datos y usarlos para obtener k subespacios afines, donde cada uno es determinado por $d + 1$ observaciones. El cálculo del subespacio afín viene dado por el punto medio x_0^j de los $d + 1$ puntos y una matriz U_0^j cuyas columnas son los d autovectores unitarios asociados a los autovalores distintos de 0 de la matriz de varianzas-covarianzas muestral de las $d + 1$ observaciones.
3. Paso de concentración. Sea $H = \{h_1, \dots, h_k\} \subset A_d$ los k subespacios afines calculados en la iteración anterior.
 - a) Calcular las distancias $d_i = d(x_i, H)$, $i = 1, \dots, n$, entre cada observación y el subespacio afín más próximo de entre los k subespacios afines obtenidos en la iteración anterior. Determinar el conjunto C que consiste en las $[n(1 - \alpha)]$ observaciones con menores distancias d_i , donde:

$$d_i^2 = \inf_{j=1, \dots, k} \|\{I - U_0^j(U_0^j)'\}(x_i - x_0^j)\|^2$$

- b) Hacer una partición del conjunto C en $C = \{C_1, \dots, C_k\}$ donde los puntos en C_j son aquellas observaciones que están más cercanas a h_j que a cualquiera de los demás subespacios afines h_l con $l \neq j$, es decir:

$$C_j := \{x_i \in C : d(x_i, h_j)^2 = d_i\}$$

- c) Sea m_j la media muestral de las observaciones en C_j y U_1^j una matriz cuyas columnas son los d autovectores asociados a los d mayores autovalores de la matriz de varianzas-covarianzas muestral de las observaciones en C_j . Los k subespacios afines $H = \{h_1, \dots, h_k\}$ para la siguiente iteración serán los k subespacios afines que pasan por m_1, \dots, m_k y son generados por los vectores disponibles en las columnas de U_1^1, \dots, U_1^k respectivamente.

4. Repetir el paso de concentración varias veces. Posteriormente, calcular el valor de la siguiente función de coste:

$$\frac{1}{[n(1 - \alpha)]} \sum_{j=1}^k \sum_{x_i \in C_j} d_i^2$$

5. Repetir los pasos 2, 3 y 4 varias veces, de tal manera que se inicialice el algoritmo con subespacios afines iniciales diferentes en cada comienzo. Guardar las soluciones que obtengan menores valores en la función de coste, iterar completamente esas soluciones y elegir la óptima en base al criterio de minimizar la función de coste.

Cuando el parámetro de recorte α toma el valor 0, entonces el algoritmo expuesto se reduce al agrupamiento lineal en Van Aelst *et al.* (2006) [3]. Cuando se busca agrupar las observaciones entorno a un único subespacio $k = 1$ se está realizando un Análisis en Componentes Principales recortado en el caso de que $\alpha > 0$ o un Análisis en Componentes Principales clásico en el caso de que $\alpha = 0$. Si se busca agrupar las observaciones entorno a k subespacios de dimensión 0, k centroides, el método se transforma en las k -medias recortadas de la sección 3.2 en el caso de que $\alpha > 0$ y en las k -medias clásicas en el caso de que $\alpha = 0$.

5. Implementación

En este capítulo se va a explicar el desarrollo de dos versiones del método presentado en el Capítulo 4 con el objetivo de optimizar la computación de los subespacios afines. Una versión utiliza el poder de la computación paralela en R, y otra la eficiencia del lenguaje C++. Una versión secuencial en R también ha sido desarrollada pero no será detallada al ser más sencilla y carecer de menos valor que las otras dos versiones comentadas.

Además, se han añadido nuevos pasos y modificaciones para mejorar el algoritmo, obteniendo una versión más completa del mismo. Estas mejoras serán explicadas como paso previo al pseudocódigo desarrollado.

5.1. Mejoras añadidas

5.1.1. Subespacios afines de dimensiones diferentes

La primera de las mejoras consideradas es poder agrupar las observaciones entorno a subespacios afines de diferentes dimensiones. Se rompe con la limitación de que todas las estructuras lineales que se busquen tengan que residir en el mismo subespacio vectorial.

En la versión original del método, se buscaban $H = \{h_1, h_2, \dots, h_k\} \subset A_d$ subespacios afines, donde $h_1 \in \mathbb{R}^p, h_2 \in \mathbb{R}^p, \dots, h_k \in \mathbb{R}^p$ con $p < d$. Ahora, además de seguir contemplando el caso anterior, se permite buscar $H = \{h_1, h_2, \dots, h_k\} \subset A_d$ subespacios afines en \mathbb{R}^p con dimensiones d_1, \dots, d_k (no necesariamente coincidentes) y $d_j < p$ para $j = 1, \dots, k$.

Para ilustrar una situación donde sea útil esta generalización de agrupar entorno a subespacios afines de diferentes dimensiones, se diseña un experimento generando datos artificialmente provenientes de 2 distribuciones normales multivariantes y una mixtura de la distribución normal y la distribución uniforme, tal y como se especifica a continuación:

$$x_1, x_2, \dots, x_{1000} \sim N \left[\begin{pmatrix} -2 \\ -2 \\ -2 \end{pmatrix}, \begin{pmatrix} \frac{1}{5} & 0 & 0 \\ 0 & \frac{1}{5} & 0 \\ 0 & 0 & \frac{1}{5} \end{pmatrix} \right]$$

$$x_{1001}, x_{1002}, \dots, x_{2000} \sim N \left[\begin{pmatrix} 5 \\ 5 \\ 5 \end{pmatrix}, \begin{pmatrix} 10 & 0 & 0 \\ 0 & \frac{1}{20} & 0 \\ 0 & 0 & \frac{1}{20} \end{pmatrix} \right]$$

$$x_{i,1} \sim N \left(12, \frac{1}{10} \right), \quad x_{i,2} \sim U(5, 15), \quad x_{i,3} \sim U(-5, 15), \quad i = 2001, \dots, 6000$$

En la Figura 5.1 se encuentra la representación gráfica conjunta tridimensional de las 3 muestras en \mathbb{R}^3 .

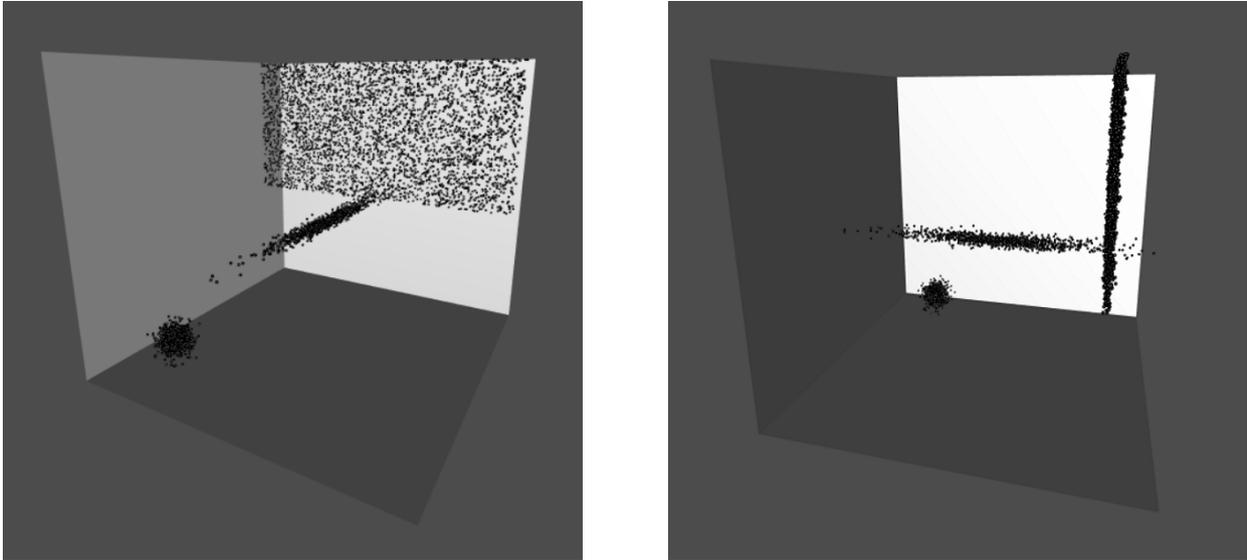


Figura 5.1: Representación del conjuntos de observaciones x_1, \dots, x_{6000} .

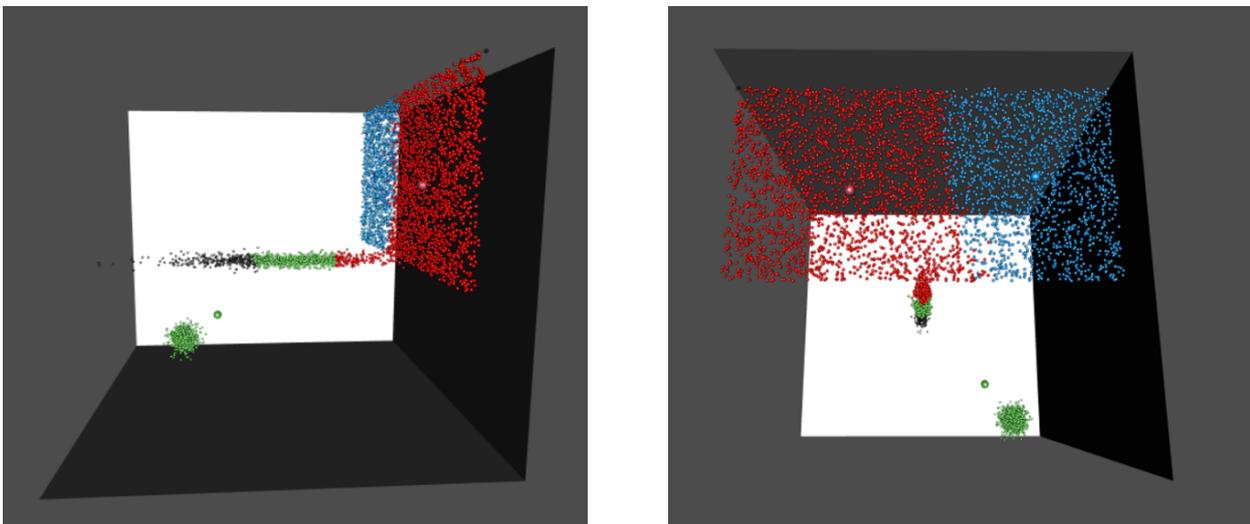


Figura 5.2: Agrupaciones realizadas ($d=(0,0,0)$ y $\alpha = 0.05$).

Manteniendo la restricción de que los subespacios afines sean de la misma dimensión, en este caso solo se podría agrupar las observaciones de 3 maneras diferentes: entorno a 3 subespacios afines de dimensión 0 (Figura 5.2), entorno a 3 subespacios afines de dimensión 1 (Figura 5.3) y entorno a 3 dimensiones afines de dimensión 2 (Figura 5.4).

5 IMPLEMENTACIÓN

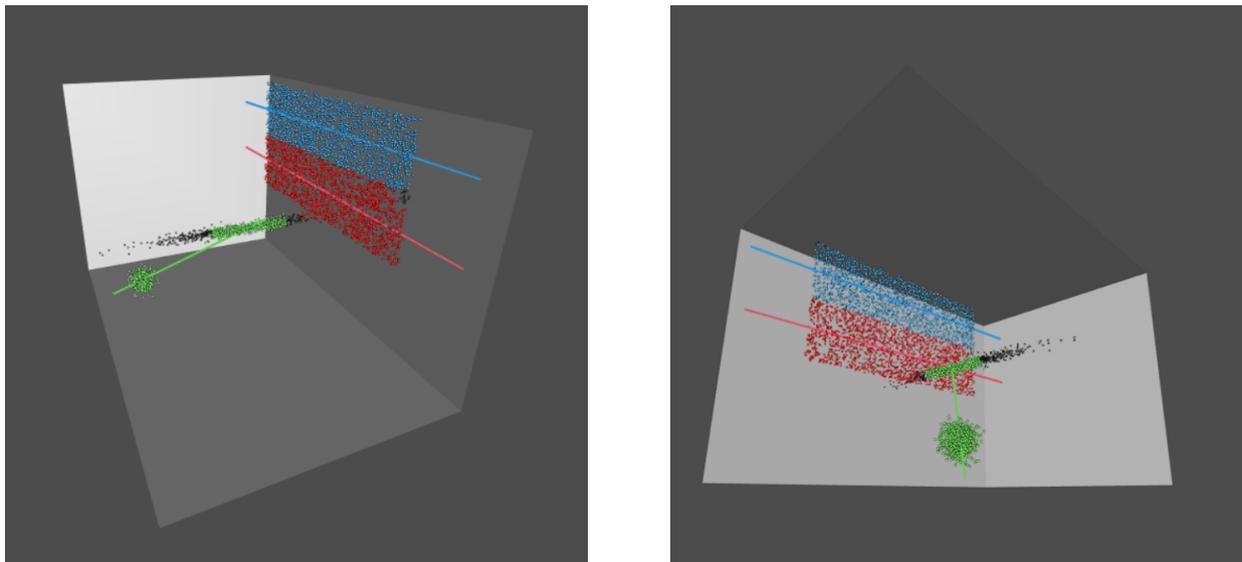


Figura 5.3: Agrupaciones realizadas ($d=(1,1,1)$ y $\alpha = 0.05$).

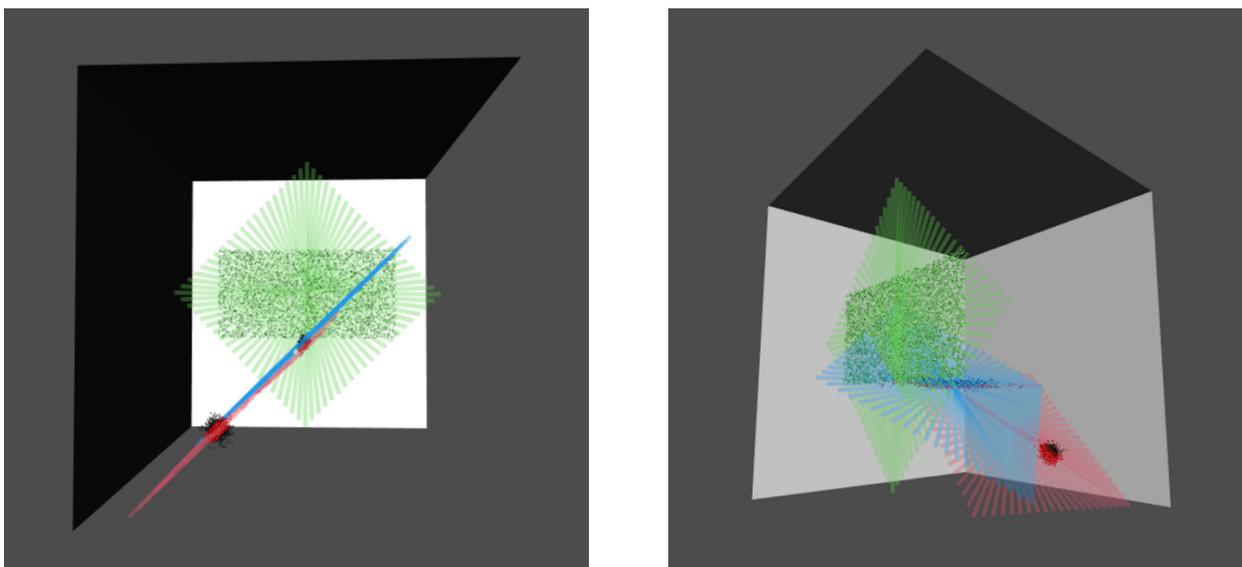


Figura 5.4: Agrupaciones realizadas ($d=(2,2,2)$ y $\alpha = 0.05$).

En los 3 casos se aprecia como las agrupaciones realizadas no son razonables. Lo lógico sería conseguir que las 3 muestras aleatorias simples provenientes de las 3 distribuciones normales multivariantes se encuadraran en 3 *clusters* diferenciados, y con ninguna de las 3 configuraciones mostradas se consigue. Analizando tanto las distribuciones de las que han sido extraídas las muestras como su representación gráfica (Figura 5.1) es fácil darse cuenta que la primera de las muestras

(x_1, \dots, x_{1000}) ha de ser agrupada entorno a un subespacio afín de dimensión 0 (un centroide), la segunda de las muestras $(x_{1001}, \dots, x_{2000})$ ha de ser agrupada entorno a un subespacio afín de dimensión 1 (una recta), y la tercera de las muestras $(x_{2001}, \dots, x_{6000})$ ha de ser agrupada entorno a un subespacio afín de dimensión 2 (un plano).

Para lograr permitir agrupar observaciones entorno a subespacios afines de distintas dimensiones, hay que modificar en tres puntos el algoritmo presentado en la sección 4.2, concretamente:

- En el paso 2, a la hora de seleccionar k subespacios afines iniciales en A_d , ya no hay que seleccionar $(d+1) \times k$ observaciones aleatorias, si no que ahora habrá que seleccionar $(d_1+1), \dots, (d_k+1)$, donde d_k es la dimensión del subespacio k .
- En el paso 2, el cálculo del subespacio afín inicial venía dado por el punto medio m_j de las $d+1$ observaciones iniciales y una matriz U_0^j cuyas columnas eran los d autovectores unitarios asociados a los autovalores distintos de 0 de la matriz de varianzas-covarianzas muestral de las $d+1$ observaciones. Ahora, el cálculo del subespacio afín vendrá dado por el punto medio m_j de las d_j+1 observaciones iniciales y una matriz U_0^j cuyas columnas serán los d_j autovectores unitarios asociados a los autovalores distintos de 0 de la matriz de varianzas-covarianzas muestral de las d_j+1 observaciones, $j = 1, \dots, k$, donde d_j representa la dimensión del subespacio afín j .
- En el paso 3.c), U_1^j ahora tendrá d_j columnas, los d_j autovectores asociados con los d_j mayores autovalores de la matriz de varianzas-covarianzas muestral de las observaciones en C_j , $j = 1, \dots, k$.

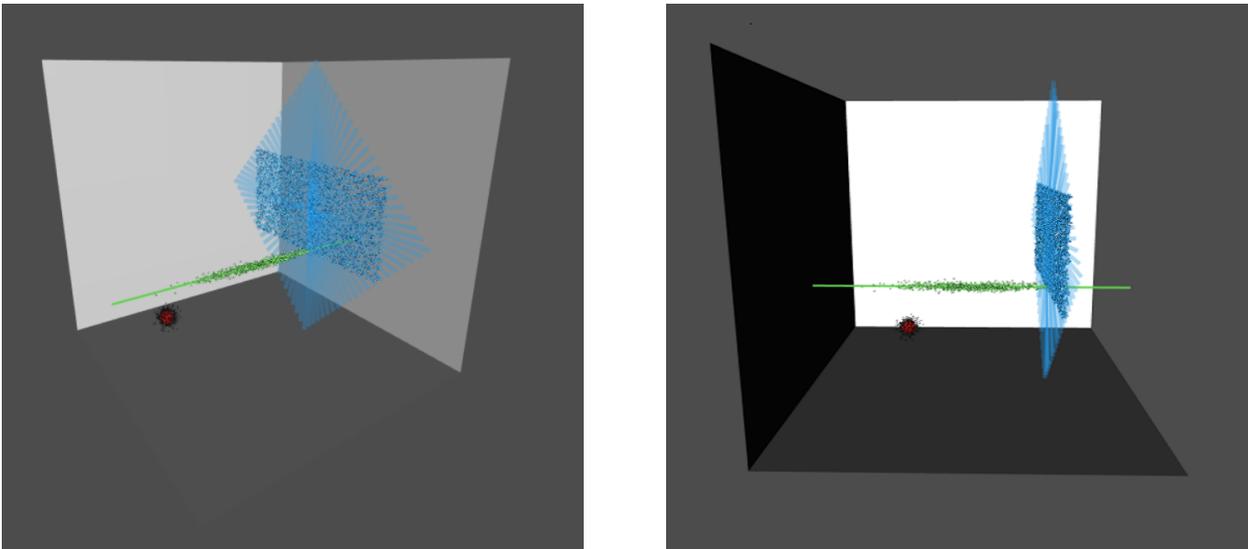


Figura 5.5: Agrupaciones realizadas ($d=(0,1,2)$ y $\alpha = 0.05$).

5 IMPLEMENTACIÓN

Realizando estos cambios en el algoritmo, se pueden agrupar las observaciones de un conjunto de datos entorno a estructuras lineales que no tienen porqué ser de idéntica dimensionalidad. Cuando buscamos agrupar las observaciones del conjunto de datos representado en la Figura 5.1 entorno a 3 subespacios afines de dimensiones 0, 1 y 2 respectivamente (Figura 5.5) se aprecia como el *clustering* realizado es el óptimo, agrupando cada una de los 3 conjuntos de puntos extraídos de distintas distribuciones en 3 grupos diferentes.

5.1.2. Estimador de mínimo determinante de la covarianza y distancia robusta de Mahalanobis

En ocasiones, cuando las agrupaciones encontradas por el algoritmo provocan la intersección de dos o más subespacios afines entre sí, los resultados que se obtienen no son todo lo ideales que se pretendería. Las observaciones que se encuentren en la intersección de dos o más subespacios serán asignadas a aquella estructura lineal con la que tengan una menor distancia ortogonal, induciendo en ocasiones a agrupaciones que no son las deseadas, ya que el criterio que debería primar no es únicamente el de la cercanía al subespacio sino también el de la cercanía a la nube de puntos que conforman cada *cluster*.

Suponer que se tienen un conjunto de observaciones x_i en \mathbb{R}^3 generadas de la siguiente manera (Figura 5.6):

$$x_{i,1} \sim U(-5, 5), \quad x_{i,2} \sim U(-5, 5), \quad x_{i,3} \sim N\left(0, \frac{1}{10}\right), \quad i = 1, \dots, 25000$$

$$(x_{i,1}, x_{i,2}, x_{i,3}) = \lambda_{i-25000} \cdot (0, -7, -10) + \varepsilon_i, \quad \varepsilon_i \sim N \left[\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} \end{pmatrix} \right]$$

$$\lambda_i \in [0.25, 0.251, \dots, 1.249], \quad i = 25001, \dots, 26000$$

Con esta distribución de las observaciones, y buscando agrupaciones entorno a 2 subespacios afines de dimensiones 1 y 2 respectivamente sin aplicar ningún recorte se obtiene la solución mostrada en la Figura 5.7.

A priori, parece que el resultado obtenido es el esperado, las observaciones x_i han sido agrupadas en dos *clusters* diferentes. Sin embargo, poniendo el detalle en la intersección del subespacio representado con una línea roja con el subespacio representado con líneas verdes se ve como muchas de las observaciones son asignadas al subespacio representado con una línea roja, cuando lo lógico sería que por orientación espacial de la nube de puntos verde las observaciones fueran

asignadas al subespacio representado por líneas verdes. Este fenómeno no es debido a un error del algoritmo, y es que la distancia ortogonal de esas observaciones al subespacio representado con una línea roja es menor que la distancia de las observaciones al subespacio representado con líneas verdes (el plano). Queda plasmada la necesidad de desarrollar un último paso con el que dichas observaciones sean reasignadas al subespacio bidimensional, ya que la nube de puntos verde es mucho más cercana a esas observaciones que la nube de puntos roja.

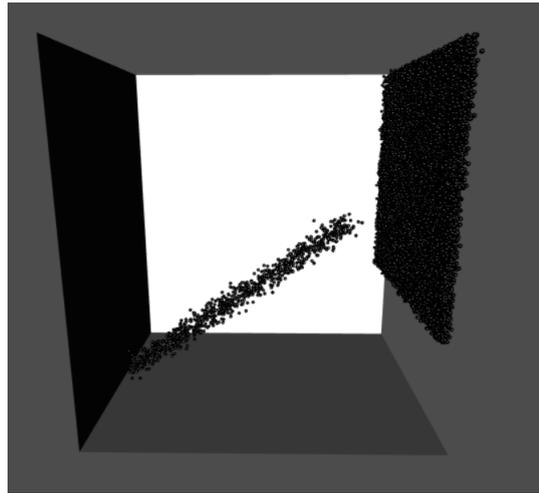


Figura 5.6: Representación del conjunto de observaciones x_1, \dots, x_{26000} .

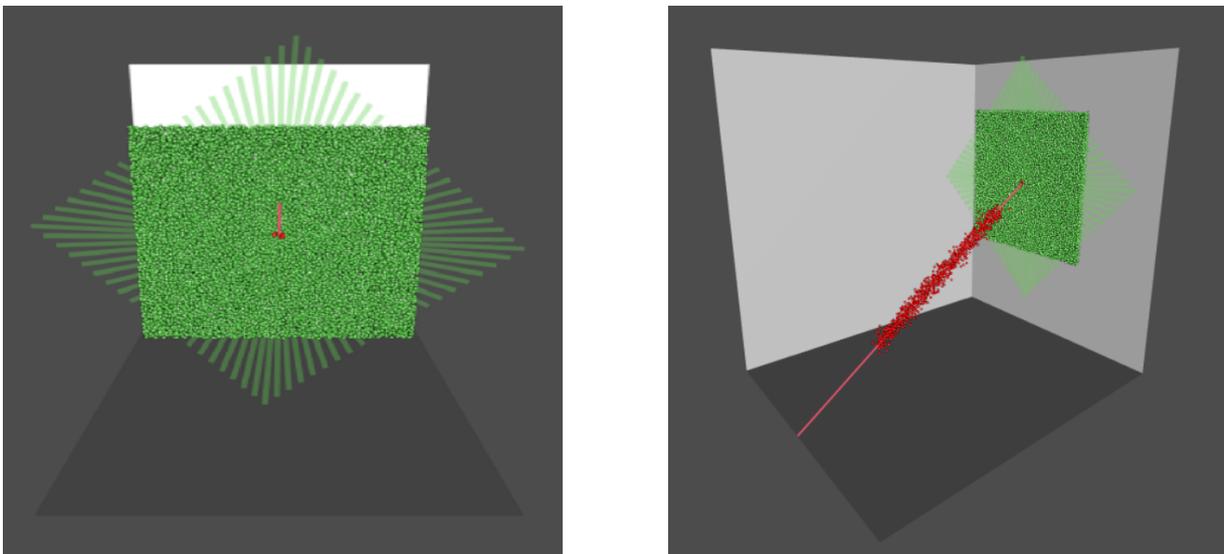


Figura 5.7: Agrupaciones realizadas ($d=(1,2)$ y $\alpha = 0$).

5 IMPLEMENTACIÓN

Para solventar esta problemática y mejorar el funcionamiento del algoritmo en situaciones de intersección de dos o más subespacios afines, se introducirán dos técnicas cuya combinación será añadida como paso final al método descrito en el capítulo anterior: el determinante de mínima covarianza y la distancia de Mahalanobis. El objetivo de la primera será el de estimar de una manera robusta la localización y la matriz de varianzas-covarianzas de la nube de puntos que conforma cada *cluster*, y el objetivo de la segunda será el de calcular la distancia entre cada observación y cada nube de puntos, es decir cada *cluster*, para finalmente realizar la asignación definitiva minimizando dicha distancia.

El MCD (*Minimum Covariance Determinant*) es un método para estimar la localización y la matriz de varianzas-covarianzas de un conjunto de datos tratando de que los puntos anómalos influyan lo menos posible en dicha estimación. La idea que reside detrás de MCD realizar la estimación en base a un subconjunto del total de las observaciones intentando que en dicho subconjunto no incluya ninguna observación atípica que pueda sesgar las estimaciones [9].

Para lograr eso, el escenario ideal consistiría en seleccionar todos los posibles subconjuntos de una nube de puntos de un tamaño especificado, estimar la media muestral y la matriz de varianzas-covarianzas muestral del subconjunto y, finalmente, seleccionar el subconjunto cuyo determinante de la matriz de varianzas-covarianzas muestral sea menor. Dicha matriz de varianzas-covarianzas es multiplicada por un factor de consistencia. El determinante de una matriz de varianzas-covarianzas es indicativo de la dispersión multivariante de la distribución, por lo que minimizar el determinante trata de encontrar la distribución menos dispersa posible, minimizando así el riesgo de presencia de *outliers* que incrementarían notablemente la dispersión y aumentarían consecuentemente el determinante de la matriz de varianzas-covarianzas.

Un ejemplo del resultado que se obtiene mediante la estimación del punto medio y la matriz de varianzas-covarianzas mediante MCD se observa en la Figura 5.8. La elipse magenta representa la elipse de tolerancia al 97.5 %, definida como el conjunto de puntos x cuya distancia de Mahalanobis

$$MD(x) = \sqrt{(x - \bar{x})' S^{-1} (x - \bar{x})}$$

es igual a $\sqrt{\chi_{p,0,975}^2}$. Se observa como la elipse trata de abarcar todas las observaciones. Bajo este criterio, una única observación queda fuera de la elipse y sería considerada como *outlier*. Por otro lado, la elipse de tolerancia robusta utiliza las distancias de Mahalanobis calculadas a partir de las estimaciones de los parámetros de localización y escala ofrecidas por MCD

$$RD(x) = \sqrt{(x - \hat{\mu}_{MCD})' \Sigma_{MCD}^{-1} (x - \hat{\mu}_{MCD})}$$

y la elipse resultante es mucho más pequeña (elipse verde). Con esta versión robusta son muchas más observaciones las consideradas como atípicas, y se comprueba la robustez de los parámetros de localización y escala estimados por MCD.

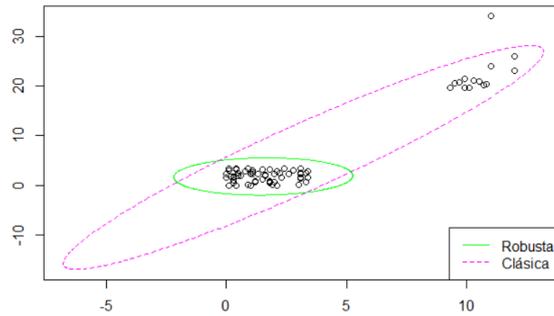


Figura 5.8: Elipses de tolerancia clásica y robusta sobre un mismo conjunto de datos.

Aplicando esta técnica al algoritmo desarrollado en esta sección, una vez se hayan estimado los parámetros de localización y escala de los *clusters*, estos se utilizarán para calcular la distancia con las observaciones mediante la distancia de Mahalanobis, que permite medir la distancia entre un punto y una distribución o nube de puntos, relativa al tamaño de la nube. Por tanto, para el método aquí descrito, se calcularán de manera robusta mediante el determinante de mínima covarianza los parámetros de localización y escala de las k nubes de puntos que representan los k *clusters*: $\hat{\mu}_{C_k}$ y $\hat{\Sigma}_{C_k}$. Posteriormente, para cada observación, se calcularán sus distancias de Mahalanobis con las k nubes de puntos formadas por los k clusters:

$$d_{\text{Mah-Rob}}(x_i, C_k) = \sqrt{(x_i - \hat{\mu}_{C_k})' \hat{\Sigma}_{C_k}^{-1} (x_i - \hat{\mu}_{C_k})}$$

Por tanto, siendo $x_1, \dots, x_n \in \mathbb{R}^p$ el conjunto de observaciones y $C = \{C_1, \dots, C_k\}$ los *clusters* encontrados por el método, la asignación final en *clusters* vendrá dada por:

$$C_j := \{x_i \in C : d_{\text{Mah-Rob}}(x_i, C_j) = \min_{l=1, \dots, k} d_{\text{Mah-Rob}}(x_i, C_l)\}$$

Realizando este paso final, se consigue que cuando los subespacios lineales encontrados por el algoritmo provoquen la intersección de dos o más subespacios afines entre sí, los resultados que se obtenían anteriormente sean corregidos, y las observaciones que se encuentran cerca de la intersección ahora sean asignadas a la nube de puntos que conforme un *cluster* más cercana. En la Figura 5.9 se ven los resultados obtenidos tras aplicar la reasignación final en base al MCD y la distancia robusta de Mahalanobis. Ahora, en la intersección entre los dos subespacios afines, el comportamiento es el esperado, siendo las observaciones fronterizas asignadas a la nube de puntos más cercana (el plano) y no al hiperplano más próximo siguiendo el criterio de la distancia ortogonal.

5 IMPLEMENTACIÓN

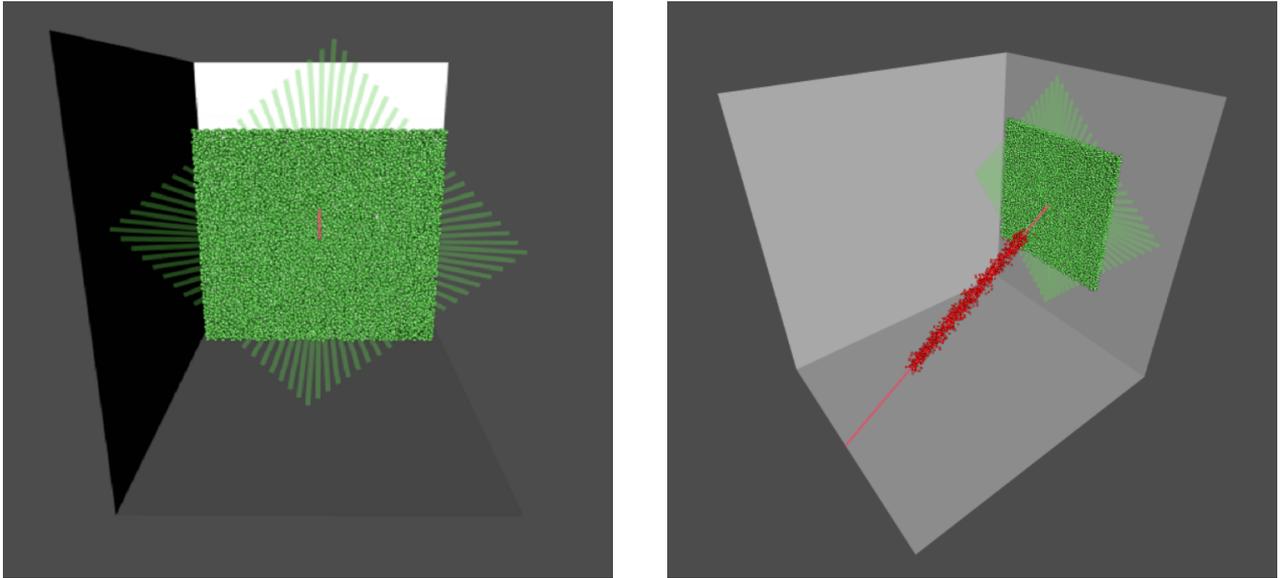


Figura 5.9: Agrupaciones realizadas ($d=(1,2)$ y $\alpha = 0$) tras reasignar los grupos mediante MCD y Mahalanobis.

La función desarrollada en R para realizar la funcionalidad explicada es la siguiente:

```
allocateMahalanobis <- function(n,d,grupos){
  robustMahalanobis <- matrix(rep(0,n*length(d)), nrow = n, ncol = length(d))
  for(i in 1:length(d)){
    mcd <- covMcd(puntos[grupos==i,])
    for(j in 1:n){
      robustMahalanobis[j,i] <- mahalanobis(puntos[j,],mcd$center,mcd$cov)
    }
  }
  gruposRobustos <- apply(robustMahalanobis,1,which.min)
  gruposRobustos[grupos==1000] <- 1000
  return(gruposRobustos)
}
```

5.1.3. Funcionalidad gráfica

Como parte de la funcionalidad añadida a la implementación del algoritmo de *clustering* entorno a subespacios lineales se ha diseñado un sistema para representar gráficamente las agrupaciones realizadas y los subespacios ajustados en dimensiones bajas. La funcionalidad gráfica está disponible cuando las observaciones del conjunto de datos se encuentran en \mathbb{R}^2 y \mathbb{R}^3 . A continuación se detallan las dos opciones.

5.1.3.1 2D

Cuando se pretenden agrupar las observaciones utilizando únicamente la información proporcionada por dos variables los gráficos que se generan son, como es lógico, en dos dimensiones. En este caso, los espacios en los que agrupar las observaciones sólo pueden ser de dimensión 0 y 1, y serán representados mediante puntos y rectas respectivamente.

Un ejemplo de representación gráfica cuando los subespacios afines entorno a los que agrupar son de dimensión 0 se encuentra en la Figura 5.10. Se aprecia como el color en el que se encuentran las observaciones representa su grupo, teniendo el mismo color todos los individuos que pertenecen a un mismo *cluster*. En el centro de los *clusters*, del mismo color, se sitúa un punto más grande que el resto, representando el baricentro o centroide de la agrupación.

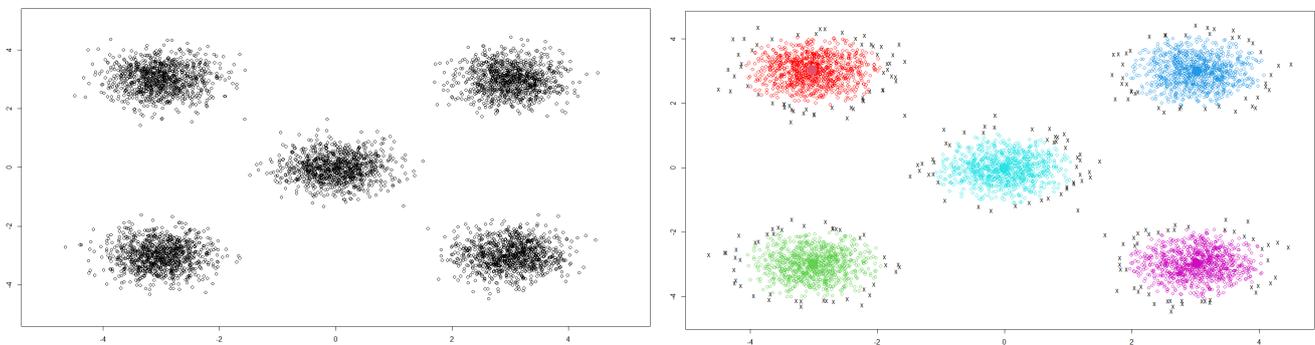


Figura 5.10: Representación bidimensional de las agrupaciones realizadas entorno a subespacios afines de dimensión 0.

Cuando los subespacios afines son de dimensión unitaria éstos quedan definidos por una recta (Figura 5.11). La recta, al igual que los centroides en el caso de subespacios de dimensión 0, está graficada en el mismo tono que los puntos que pertenecen al grupo que representa.

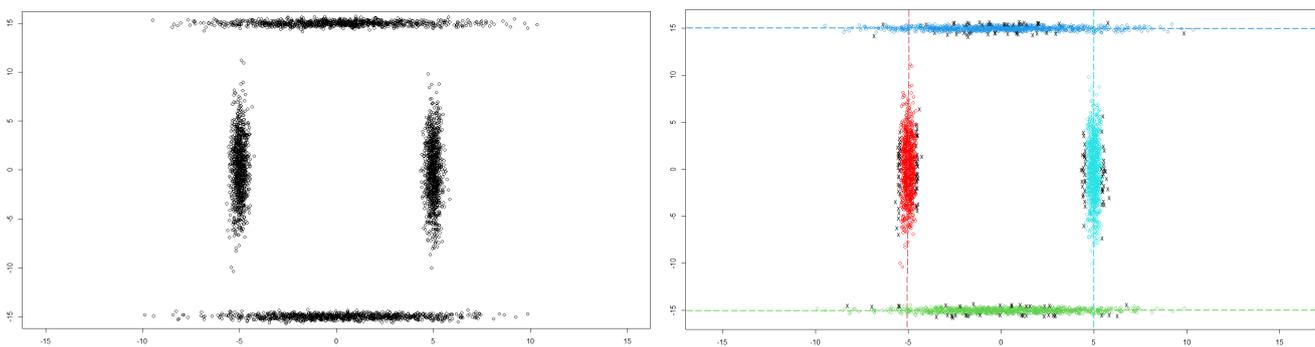


Figura 5.11: Representación bidimensional de las agrupaciones realizadas entorno a subespacios afines de dimensión 1.

5 IMPLEMENTACIÓN

Los puntos negros y con forma de aspa presentes en las representaciones gráficas se corresponden con las observaciones recortadas. Recordemos que dichas observaciones han sido tratadas como puntos atípicos y no han influido en la determinación de los subespacios afines.

5.1.3.2 3D

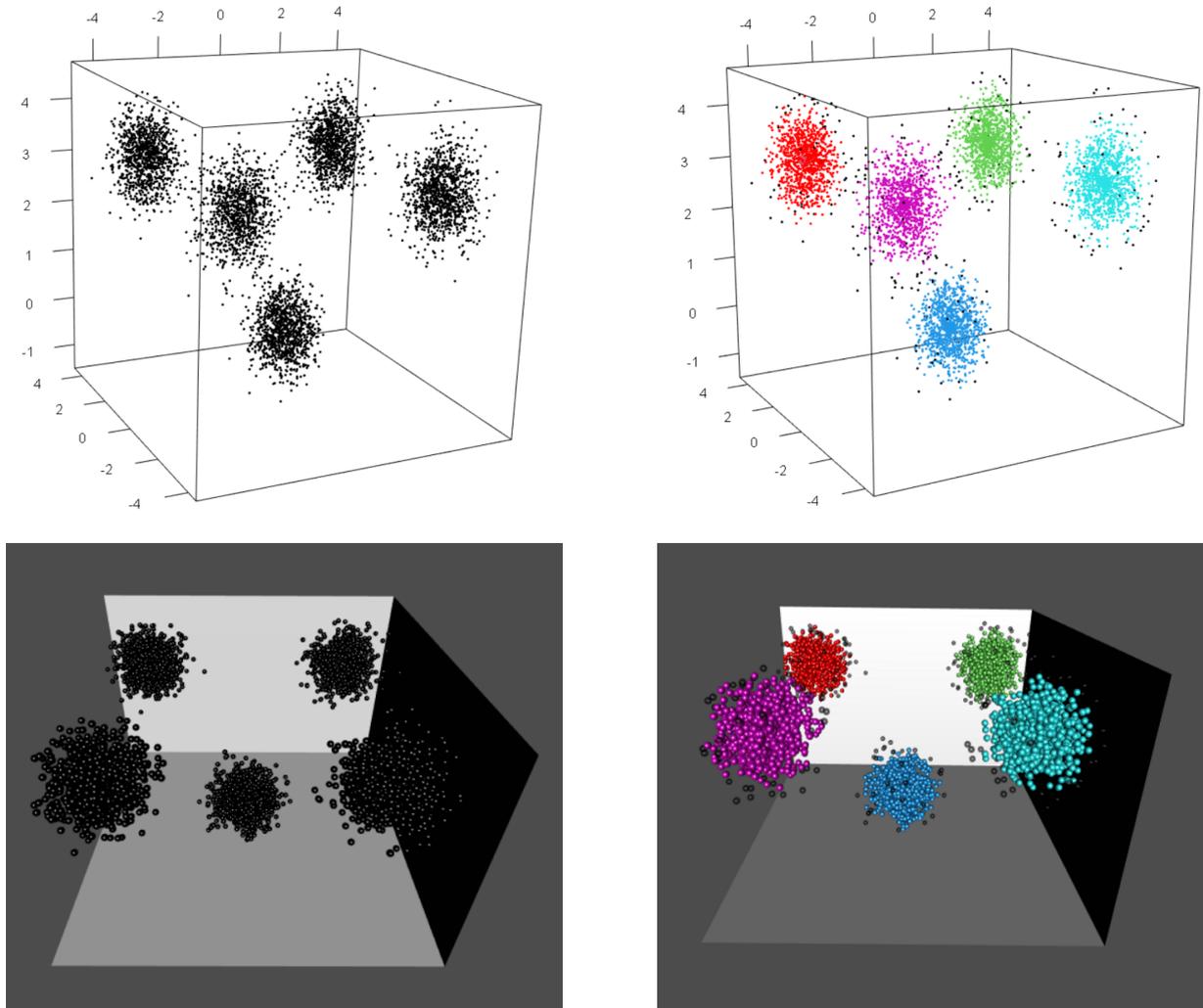


Figura 5.12: Representaciones tridimensionales de las agrupaciones realizadas entorno a subespacios afines de dimensión 0.

Si se pretende realizar el análisis *cluster* robusto en un conjunto de datos en \mathbb{R}^3 , entonces se ofrecen dos posibilidades de representación, una más sencilla y una más estilizada, cada una de las

cuales tiene sus ventajas y desventajas. Ambas representaciones son interactivas y permiten las rotaciones y el aumento sobre una zona concreta de la figura. Para ambos estilos de representaciones se ha utilizado el paquete contribuido `rgl`.

Respecto a la representación más sencilla, la principal ventaja que tiene es que conlleva menor tiempo para su generación que la variante más estilizada y no se requieren especiales prestaciones en cuanto al *hardware* para su generación. Como defecto se puede mencionar su aspecto menos llamativo.

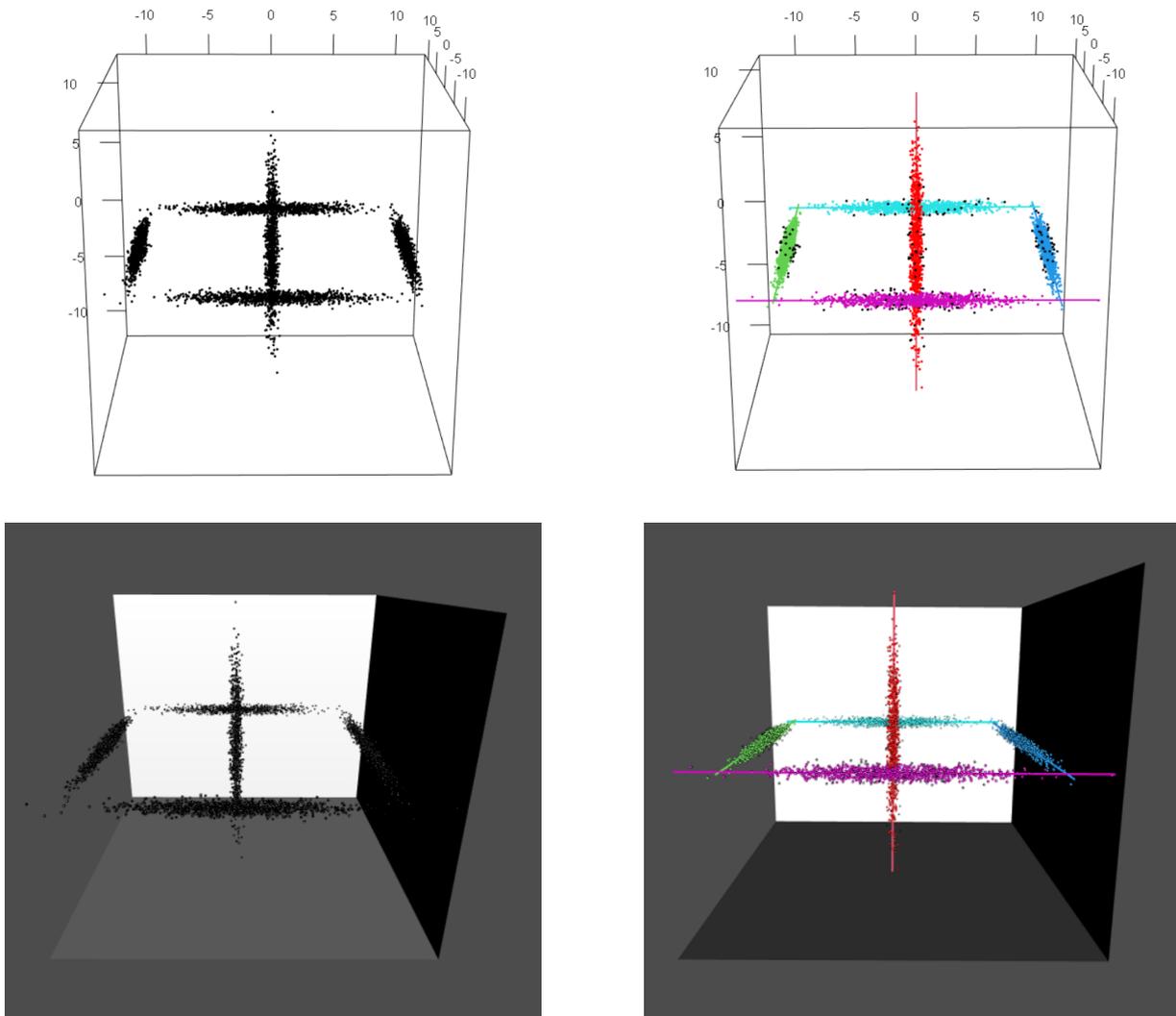


Figura 5.13: Representaciones tridimensionales de las agrupaciones realizadas entorno a subespacios afines de dimensión 1.

5 IMPLEMENTACIÓN

Respecto a la representación más estilizada, su única ventaja y no por ello menos importante es facilitar su visualización. Como hándicaps se encuentran el trabajo nada desdeñable que ha de realizar la tarjeta gráfica de la computadora para generarlo e interactuar con él cuando los conjuntos de datos son de un tamaño considerable.

Las posibles dimensiones de los subespacios entorno a los que agrupar cuando los datos son tridimensionales son 1, 2 y 3. En los dos primeros casos, el modo de representar tanto las observaciones como los subespacios es idéntico al caso bidimensional, mostrando los centroides de los subespacios de dimensión 0 como puntos y los subespacios de dimensión 1 como rectas (Figuras 5.12 y 5.13).

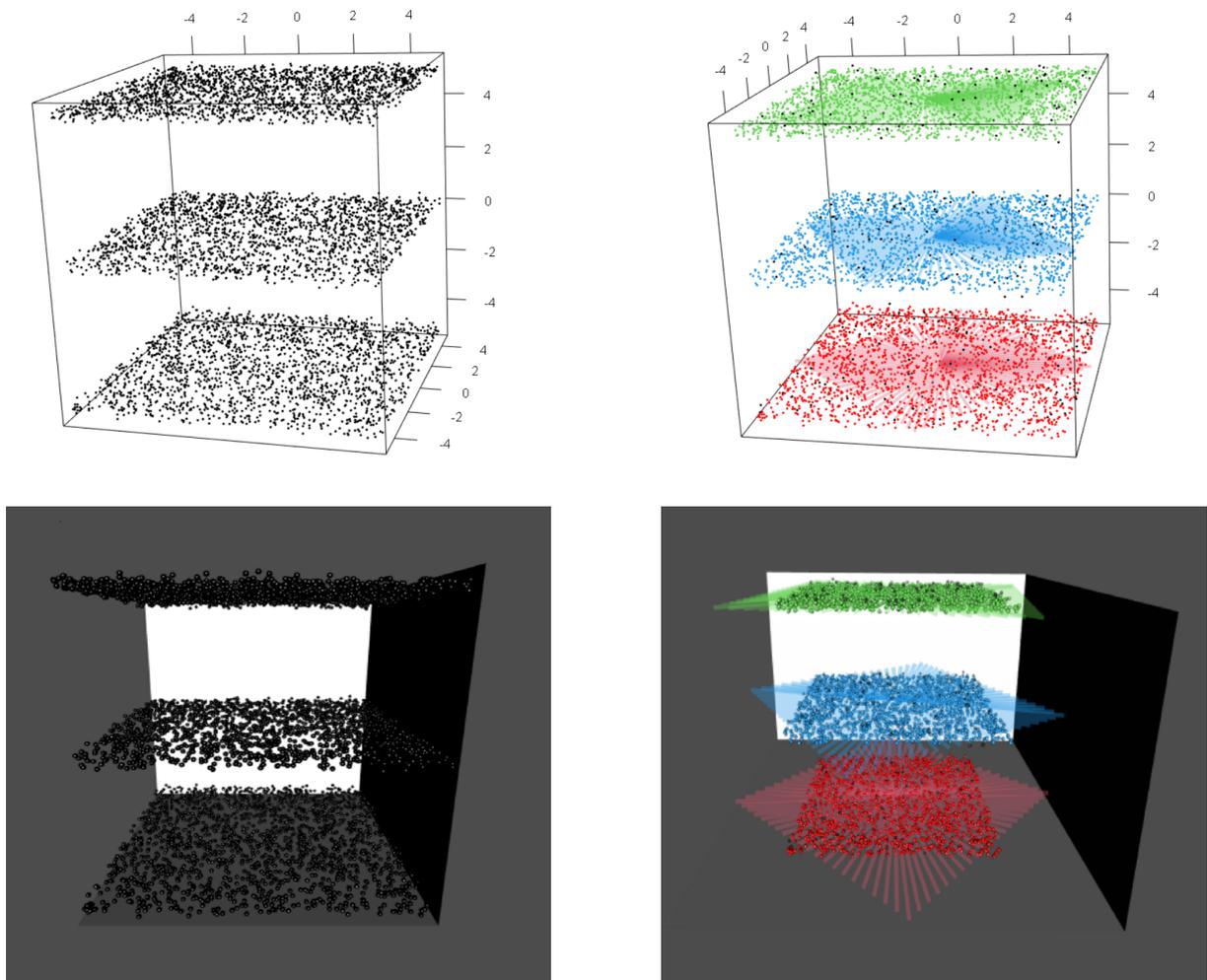


Figura 5.14: Representaciones tridimensionales de las agrupaciones realizadas entorno a subespacios afines de dimensión 2.

Cuando los uno o varios subespacios afines entorno a los que realizar *clustering* son de dimensión 2, entonces dichas estructuras son representadas mediante la rotación en los ejes ortogonales del subespacio de dos rectas (Figura 5.14). Dichas rectas iniciales se corresponden con las dos direcciones de máxima varianza de los puntos que pertenecen al subespacio y que contengan al punto medio de dicha agrupación. Una vez se disponen de esas dos rectas, éstas se rotan como si se tratase de una peonza para generar una estructura con forma de plano que representa al subespacio afín de dimensión de 2. Igualmente, el color del plano es idéntico al color de las observaciones que pertenecen a su grupo.

Al igual que en el caso bidimensional, y para ambos formatos de representaciones, los puntos recortados son representados en negro, esta vez sin forma de aspa.

5.2. Pseudocódigo

Algorithm 1 Robust Linear Clustering

Input: $X = \{x_1, \dots, x_n\}$: conjunto de datos, d : vector con las dimensiones de los subespacios, $\alpha \in [0, 1]$: parámetro de recorte, $nstarts$: número de inicializaciones

Output: *clusters*: grupo al que pertenece cada observación, *fObj*: valor de la función objetivo

```

1: function ROBUSTLINEARSUBSPACESINIT( $X, d, \alpha$ )
2:    $k \leftarrow$  tamaño de  $d$ 
3:    $n \leftarrow$  número de observaciones en  $X$ 
4:    $p \leftarrow$  dimensión de las observaciones en  $X$ 
5:    $\mathcal{D} \leftarrow$  matriz de dimensiones  $n \times k$ 
6:   for  $i \in \{1, \dots, k\}$  do
7:     if  $d[i] == 0$  then
8:        $c \leftarrow$  observación aleatoria de  $X$ 
9:       for  $l \in \{1, \dots, n\}$  do
10:         $\mathcal{D}[l, i] \leftarrow \sqrt{\sum_{j \in \{1, \dots, p\}} (X_{l,j} - c_j)^2}$ 
11:      end for
12:     else
13:        $C \leftarrow$  matriz de dimensiones  $(d[i] + 1) \times p$ 
14:       for  $s \in \{1, \dots, d[i] + 1\}$  do
15:         $C_s \leftarrow$  observación aleatoria de  $X$ 
16:       end for
17:        $\bar{x} \leftarrow$  media de las observaciones en  $C$ 
18:        $\mathcal{U} \leftarrow$  matriz de dimensiones  $p \times d[i]$  con los  $d[i]$  mayores autovectores como columnas
19:       for  $l \in \{1, \dots, n\}$  do

```

5 IMPLEMENTACIÓN

```

20:            $\mathcal{D}[l, i] = \|\{I - \mathcal{U}(\mathcal{U})'\}(X_l - \bar{x})\|^2$ 
21:       end for
22:   end if
23: end for
24: clusters, minimos  $\leftarrow$  vectores  $n$ -dimensionales
25: for  $l \in \{1, \dots, n\}$  do
26:     minimos [ $l$ ]  $\leftarrow$   $\min_{i=1, \dots, k} \mathcal{D}[l, i]$ 
27:     clusters [ $l$ ]  $\leftarrow$   $i$  tal que minimos [ $l$ ] ==  $\min_{i=1, \dots, k} \mathcal{D}[l, i]$ 
28: end for
29: lim  $\leftarrow$   $\{n \cdot \alpha$  mayores valores en minimos $\}$ 
30: clusters [minimos  $\in$  lim]  $\leftarrow$  1000
31: iter  $\leftarrow$  1
32: clustersant  $\leftarrow$  vector  $n$ -dimensional
33: while iter < 10 AND clusters  $\neq$  clustersant do
34:   clustersant  $\leftarrow$  clusters
35:   for  $i \in \{1, \dots, k\}$  do
36:     if  $d[i] == 0$  then
37:        $\bar{x} \leftarrow$  media de las observaciones donde clusters ==  $i$ 
38:       for  $l \in \{1, \dots, n\}$  do
39:          $\mathcal{D}[l, i] \leftarrow \sqrt{\sum_{j \in \{1, \dots, p\}} (X_{l,j} - c_j)^2}$ 
40:       end for
41:     else
42:        $\bar{x} \leftarrow$  media de las observaciones donde clusters ==  $i$ 
43:        $\mathcal{U} \leftarrow$  matriz de dimensiones  $p \times d[i]$  con los  $d[i]$  mayores
         autovectores de la matriz de covarianzas de las observaciones
         donde clusters ==  $i$  como columns
44:       for  $l \in \{1, \dots, n\}$  do
45:          $\mathcal{D}[l, i] = \|\{I - \mathcal{U}(\mathcal{U})'\}(X_l - \bar{x})\|^2$ 
46:       end for
47:     end if
48:   end for
49:   for  $l \in \{1, \dots, n\}$  do
50:     minimos [ $l$ ]  $\leftarrow$   $\min_{i=1, \dots, k} \mathcal{D}[l, i]$ 
51:     clusters [ $l$ ]  $\leftarrow$   $i$  tal que minimos [ $l$ ] ==  $\min_{i=1, \dots, k} \mathcal{D}[l, i]$ 
52:   end for
53:   lim  $\leftarrow$   $\{n \cdot \alpha$  mayores valores en minimos $\}$ 
54:   clusters [minimos  $\in$  lim]  $\leftarrow$  1000
55:   iter  $\leftarrow$  iter + 1
56: end while
57: costfunction  $\leftarrow$   $\frac{\sum_{l \in \{1, \dots, n\}, \text{minimos}_l \neq 1000} \text{minimos}_l^2}{[n[1-\alpha]]}$ 

```

```

58:   return (clusters, cost_function)
59: end function

60: function ROBUSTLINEARSUBSPACES(X, d,  $\alpha$ , clustersant, cost_function)
61:   k  $\leftarrow$  tamaño de d
62:   n  $\leftarrow$  número de observaciones en X
63:   p  $\leftarrow$  dimensión de las observaciones en X
64:    $\mathcal{D}$   $\leftarrow$  matriz de dimensiones  $n \times k$ 
65:   clusters  $\leftarrow$  clustersant
66:   clustersant  $\leftarrow$  vector n-dimensional
67:   flag  $\leftarrow$  0
68:   while flag < 100 AND clusters  $\neq$  clustersant do
69:     clustersant  $\leftarrow$  clusters
70:     for  $i \in \{1, \dots, k\}$  do
71:       if  $d[i] == 0$  then
72:          $\bar{x}$   $\leftarrow$  media de las observaciones donde clusters == i
73:         for  $l \in \{1, \dots, n\}$  do
74:            $\mathcal{D}[l, i] \leftarrow \sqrt{\sum_{j \in \{1, \dots, p\}} (X_{l,j} - c_j)^2}$ 
75:         end for
76:       else
77:          $\bar{x}$   $\leftarrow$  media de las observaciones donde clusters == i
78:          $\mathcal{U}$   $\leftarrow$  matriz de dimensiones  $p \times d[i]$  con los  $d[i]$  mayores
           autovectores de la matriz de covarianzas de las observaciones
           donde clusters == i como columnas
79:         for  $l \in \{1, \dots, n\}$  do
80:            $\mathcal{D}[l, i] = \|\{I - \mathcal{U}(\mathcal{U})'\}(X_l - \bar{x})\|$ 
81:         end for
82:       end if
83:     end for
84:     for  $l \in \{1, \dots, n\}$  do
85:       minimos [l]  $\leftarrow$   $\min_{i=1, \dots, k} \mathcal{D}[l, i]$ 
86:       clusters [l]  $\leftarrow$  i tal que minimos [l] ==  $\min_{i=1, \dots, k} \mathcal{D}[l, i]$ 
87:     end for
88:     lim  $\leftarrow$   $\{n \cdot \alpha$  mayores valores en minimos $\}$ 
89:     clusters [minimos  $\in$  lim]  $\leftarrow$  1000
90:     flag  $\leftarrow$  flag + 1
91:   end while
92:   cost_function  $\leftarrow$   $\frac{\sum_{l \in \{1, \dots, n\}, \text{minimos}_l \neq 1000} \text{minimos}_l^2}{[n[1-\alpha]]}$ 
93:   return (clusters, cost_function)
94: end function

```

5 IMPLEMENTACIÓN

```

95: function MCD_MAHALANOBIS( $X, d, clusters$ )
96:    $k \leftarrow$  tamaño de  $d$ 
97:    $n \leftarrow$  número de observaciones en  $X$ 
98:    $p \leftarrow$  dimensión de las observaciones en  $X$ 
99:    $\Delta \leftarrow$  matriz de dimensiones  $n \times k$ 
100:  for  $i \in \{1, \dots, k\}$  do
101:     $\hat{\mu} \leftarrow$  media de las observaciones donde  $clusters == i$  estimada por MCD.
102:     $\hat{\Sigma} \leftarrow$  matriz de covarianzas de las observaciones donde  $clusters == i$ 
    estimada por MCD.
103:    for  $l \in \{1, \dots, n\}$  do
104:       $\Delta_{l,i} \leftarrow \sqrt{(X_l - \hat{\mu})' \hat{\Sigma}^{-1} (X_l - \hat{\mu})}$ 
105:    end for
106:  end for
107:  for  $l \in \{1, \dots, n\}$  do
108:     $minimos[l] \leftarrow \min_{i=1, \dots, k} \mathcal{D}[l, i]$ 
109:     $clusters_{nuevos}[l] \leftarrow i$  tal que  $minimos[l] == \min_{i=1, \dots, k} \mathcal{D}[l, i]$ 
110:  end for
111:   $clusters_{nuevos} = 1000$  donde  $clusters == 1000$ 
112:  return ( $clusters_{nuevos}$ )
113: end function

114: Escalar robustamente  $X$  dividiendo entre su MAD.
115:  $gr \leftarrow$  matriz de dimensiones  $nstarts \times n$ 
116:  $fObjs \leftarrow$  vector  $nstarts$ -dimensional
117: for  $init \in \{1, \dots, nstarts\}$  do
118:    $gr_{init}, fObjs_{init} \leftarrow$  ROBUSTLINEARSUBSPACESINIT( $X, d, \alpha$ )
119: end for
120:  $gr10, fObjs10 \leftarrow$  agrupaciones y valores de las funciones objetivo de los 10 mejores inicios
    en base a  $fObjs_{init}$ 
121: for  $init \in \{1, \dots, 10\}$  do
122:    $grMin_{init}, fObjsMin_{init} \leftarrow$  ROBUSTLINEARSUBSPACES( $X, d, \alpha, gr10_{init}, fObjs10_{init}$ )
123: end for
124:  $agrupacionOpt, fObjOpt \leftarrow$  agrupaciones y valor de la función objetivo del menor valor en
     $fObjsMin$ 
125:  $clusters \leftarrow$  MCD_MAHALANOBIS( $X, d, agrupacionOpt$ )

```

5.3. Computación paralela

El coste computacional de la implementación del pseudocódigo presentado es muy alto. Las múltiples inicializaciones que han de hacerse debido al carácter aleatorio de la elección de los puntos iniciales que conformarán los subespacios afines iniciales provocan que el procedimiento sea muy pesado. Por ello, y para implementar el algoritmo de una manera eficiente, en este trabajo se sacará partido de la computación paralela.

En la computación paralela, a diferencia de la computación en serie o secuencial, se utilizan varios elementos con capacidad de computación para resolver un problema. Para lograrlo es necesario dividir el programa en partes independientes de modo que cada unidad de procesamiento sea capaz de ejecutar la parte que se le asigne de manera simultánea a la computación de otras partes del algoritmo por parte del resto de unidades de procesamiento. En el lenguaje de programación R existen dos maneras de aplicar la computación paralela a un programa, vía *forking* o vía *sockets*. Las características principales de cada una de ellas son [10]:

- *Forking*: su característica principal es que clona la versión de R que se tiene en el momento anterior a paralelizar el código y la migra a los *cores* que se vayan a utilizar. Entre sus ventajas se encuentran:
 - La ejecución es más rápida que utilizando *sockets*.
 - Más fácil de implementar que los *sockets*.
 - El *workspace* de R existe en las sesiones de los nuevos *cores*.

Por el contrario, tiene las siguientes desventajas:

- únicamente se puede utilizar en sistemas POSIX (Mac, Linux, BSD) y no en Windows.
- Debido a que se clonan las sesiones de R, la generación de números aleatorios puede causar problemas y causar que se generen los mismos en cada *core*.
- *Sockets*: crea una nueva versión de R en cada *core*. Tiene las siguientes ventajas:
 - se puede utilizar en cualquier sistema operativo.
 - No provoca problemas de contaminación entre *cores* al no hacer copias de la sesión original.

Las desventajas son:

- La ejecución es más lenta que utilizando *forking*.
- Más difícil de implementar que los *sockets*.

5 IMPLEMENTACIÓN

- El *workspace* de R no está disponible en las sesiones de los nuevos *cores*, si se disponía de variables creadas en el entorno principal habrá que volver a crearlas en las nuevas sesiones.

Debido a que la migración del código a distintos sistemas operativos es deseable y que la generación de números aleatorios diferentes en cada núcleo del procesador es una parte crucial del algoritmo a implementar se ha optado por utilizar la computación paralela vía *sockets*.

5.3.1. Búsqueda de subespacios afines inicial

La búsqueda de subespacios afines inicial es la parte más costosa del procedimiento y es la que se va a paralelizar. El carácter aleatorio de la selección inicial de observaciones que determinen en primera instancia los subespacios afines hace que se requieran de varias inicializaciones para que haya varias en disposición de ofrecer un buen agrupamiento. La característica de que las inicializaciones sean independientes entre sí permite que se pueda paralelizar sin que surja ningún conflicto de dependencias.

Esta parte se corresponde con la función *ROBUSTLINEARSUBSPACESINIT* del pseudocódigo desarrollado, y la función se llama `rlc_ini`. Lo primero que hay que hacer es adaptar la función y añadirla un nuevo argumento de entrada. El argumento es *dummy* y no tiene ningún uso dentro de la función, únicamente se trata de un vector utilizado para que los métodos de la librería `parallel` que se van a usar sepan cuántas veces deseamos ejecutar la función.

```
rlc_ini<-function(inicializacion ,puntos ,d ,alpha){
  tryCatch(
  {
    .
    .
    .
    return(list(funcion_coste ,grupos))
  },
  error=function(e) NULL
  )
}
```

Todas las tareas realizadas son encerradas dentro de un bloque `tryCatch`, lo que provoca que si algún error salta dentro del bloque éste sea capturado, y no interfiera en la ejecución del resto de inicializaciones que sean asignadas posteriormente al núcleo en cuya sesión se ha producido el error. El error que puede producirse es que, en un determinado punto de la ejecución debido a una mala selección aleatoria inicial de los puntos que generan los subespacios, se intente calcular la matriz de varianzas-covarianzas de entre todos los puntos pertenecientes a un mismo *cluster* cuando a ese *cluster* únicamente pertenece su centroide, caso en el que la matriz de varianzas-covarianzas no podrá calcularse.

En el interior de la función lo primero que se hace es definir algunas variables que posteriormente se van a usar.

```
k<-length(d); n<-dim(puntos)[1]; p<-dim(puntos)[2]
n_atip<-floor(alpha*n)
grupos<-rep(0,n)
x<-matrix(rep(0,k*p), nrow = k, ncol = p)
di<-matrix(rep(0,n*k), nrow = n, ncol = k)
```

Tras esto, se procede a inicializar aleatoriamente k subespacios afines de las dimensiones indicadas en el argumento d , posteriormente se calculan las distancias de cada observación a dichos subespacios iniciales.

```
for(i in 1:k){
  if(d[i] == 0){
    ci<-puntos[round(runif(1,1,n)),]
    x[i,]<-ci
    rest<-(sweep(puntos,2,x[i,]))^2
    for(j in 1:n){
      di[j,i]<-sqrt(sum(rest[j,]))
    }
  } else {
    ci<-matrix(rep(0,(d[i]+1)*p), nrow = d[i]+1, ncol = p)
    for(j in 1:(d[i]+1)){
      ci[j,]<-puntos[round(runif(1,1,n)),]
    }
    x[i,]<-apply(ci,2,mean)
    U<-matrix(rep(0,d[i]*p), nrow = d[i], ncol = p)
    eigen_g<-eigen(cov(ci))
    eigenVecOrd<-sort(eigen_g$values, decreasing = T)
    for(j in 1:d[i]){
      U[j,]<-eigen_g$vectors[,eigenVecOrd[j]==eigen_g$values]
    }
    productoMat<-diag(p)-t(U)%*%U
    rest<-sweep(puntos,2,x[i,])
    for(j in 1:n){
      di[j,i]<-norm(productoMat%*%rest[j,],'0')
    }
  }
}
```

Luego, se asigna cada observación al subespacio afín más cercano. Las $n \cdot \alpha$ observaciones cuyas distancias al subespacio más cercano sean las $n \cdot \alpha$ mayores distancias son asignadas al grupo 1000, indicativo de que dichos puntos son atípicos.

5 IMPLEMENTACIÓN

```
minimos<-apply(di,1,min)
lim<-sort(minimos, decreasing = T)[n_atip]
grupos<-apply(di, 1, which.min)
grupos[minimos>lim]<-1000
asignados<-sum(grupos==1000)
if(asignados<n_atip) grupos[minimos == lim][1:(n_atip-asignados)]<-1000
```

Posteriormente, se realiza el paso de concentración 9 veces, obteniendo una asignación en *clusters* de las observaciones. Si en dos iteraciones consecutivas la asignación de las observaciones a los diferentes grupos no varía, significa que en las posteriores iteraciones tampoco variará, por lo que el bucle se interrumpe y finaliza la ejecución de la función.

```
iter<-1
grupos_ant<-0
while(iter<10 && !all(grupos==grupos_ant)){
  grupos_ant<-grupos
  for(i in 1:k){
    for(j in 1:p){
      x[i,j]<-mean(puntos[grupos==i,j])
    }
    if(d[i] == 0){
      rest<-(sweep(puntos,2,x[i,]))^2
      for(j in 1:n){
        di[j,i]<-sqrt(sum(rest[j,]))
      }
    } else {
      eigen_g<-eigen(cov(puntos[grupos==i,]))
      eigenVecOrd<-sort(eigen_g$values, decreasing = T)
      for(j in 1:d[i]){
        U[j,]<-eigen_g$vectors[,eigenVecOrd[j]==eigen_g$values]
      }
      productoMat<-diag(p)-t(U)%*%U
      rest<-sweep(puntos,2,x[i,])
      for(j in 1:n){
        di[j,i]<-norm(productoMat%*%rest[j,],'0')
      }
    }
  }
  minimos<-apply(di,1,min)
  lim<-sort(minimos, decreasing = T)[n_atip]
  grupos<-apply(di, 1, which.min)
  grupos[minimos>lim]<-1000
  asignados<-sum(grupos==1000)
  if(asignados<n_atip) grupos[minimos == lim][1:(n_atip-asignados)]<-1000
  iter<-iter+1
}
```

Finalmente, se calcula el valor de la función de coste, el cuál será imprescindible para seleccionar las inicializaciones más prometedoras de entre todas las ejecutadas.

```
funcion_coste<-sum((minimos[grupos != 1000])^2)/(n-n_atip)
```

5.3.2. Iteración completa de las inicializaciones más prometedoras

La iteración completa de los inicios más prometedores, la amplia mayoría de las veces, no es un proceso lo suficientemente computacionalmente costoso como para que construir una versión que utilice computación paralela suponga un ahorro en tiempo. Esto es debido a que los inicios seleccionados son, en el mejor de los casos, 10, y la convergencia a la solución no suele demorarse demasiadas iteraciones.

```
rlc<-function(puntos,d,alpha,grupos){
  .
  .
  .
  return(list(funcion_coste,grupos))
}
```

La función `rlc` itera completamente la inicialización que es pasada en el argumento `grupos` hasta que o bien los grupos a los que se asigna cada observación no varían en 2 iteraciones consecutivas o bien se alcanza un número máximo de iteraciones. Ese número máximo de iteraciones ha sido fijado en 100 al considerarse que es número razonable para que los subespacios afines converjan en los óptimos. El código albergado en el cuerpo de la función `rlc` tiene un primer paso de declaración de variables que se van a usar a lo largo de la función.

```
k<-length(d); n<-dim(puntos)[1]; p<-dim(puntos)[2]
flag<-0
x_ant<-0
n_atip<-floor(alpha*n)
x<-matrix(rep(0,k*p), nrow = k, ncol = p)
di<-matrix(rep(0,n*k), nrow = n, ncol = k)
```

Luego llega el bucle que se prolonga hasta que o bien llega la convergencia a los subespacios finales o bien se cumplen las 100 iteraciones.

```
while(flag<100){
  if(flag > 0 & all(x_ant==x)) break
  if(flag > 0) x_ant<-x
  for(i in 1:k){
```

5 IMPLEMENTACIÓN

```
for(j in 1:p){
  x[i,j]<-mean(puntos[grupos==i,j])
}
if(d[i] == 0){
  rest<-(sweep(puntos,2,x[i,]))^2
  for(j in 1:n){
    di[j,i]<-sqrt(sum(rest[j,]))
  }
} else {
  U<-matrix(rep(0,d[i]*p), nrow = d[i], ncol = p)
  eigen_g<-eigen(cov(puntos[grupos==i,]))
  eigenVecOrd<-sort(eigen_g$values, decreasing = T)
  for(j in 1:d[i]){
    U[j,]<-eigen_g$vectors[,eigenVecOrd[j]==eigen_g$values]
  }
  productoMat<-diag(p)-t(U)%*%U
  rest<-sweep(puntos,2,x[i,])
  for(j in 1:n){
    di[j,i]<-norm(productoMat%*%rest[j,],'0')
  }
}
}
}
minimos<-apply(di,1,min)
lim<-sort(minimos, decreasing = T)[n_atip]
grupos<-apply(di, 1, which.min)
grupos[minimos>lim] <-1000
asignados<-sum(grupos==1000)
if(asignados<n_atip) grupos[minimos == lim][1:(n_atip-asignados)]<-1000
flag<-flag+1
}
```

Finalmente, se calcula el valor de la función de coste, el cuál será imprescindible para seleccionar las inicializaciones más prometedoras de entre todas las ejecutadas.

```
funcion_coste<-sum((minimos[grupos != 1000])^2)/(n-n_atip)
```

5.3.3. Flujo general del programa

En la Figura 5.15 se observa el diagrama de flujo del código. La parte que se realiza de manera paralela es la búsqueda de subespacios afines inicial, mientras que el resto de tareas se ejecutan secuencialmente en un sólo núcleo. La iteración completa de las inicializaciones más prometedoras podría realizarse también de manera paralela ya que cada inicialización es independiente de las demás, sin embargo la tarea, en la mayoría de ocasiones, no es lo suficientemente pesada como para que su paralelización más el tiempo necesario para crear el *cluster* de núcleos suponga un ahorro de tiempo respecto a su realización secuencial en un único núcleo.

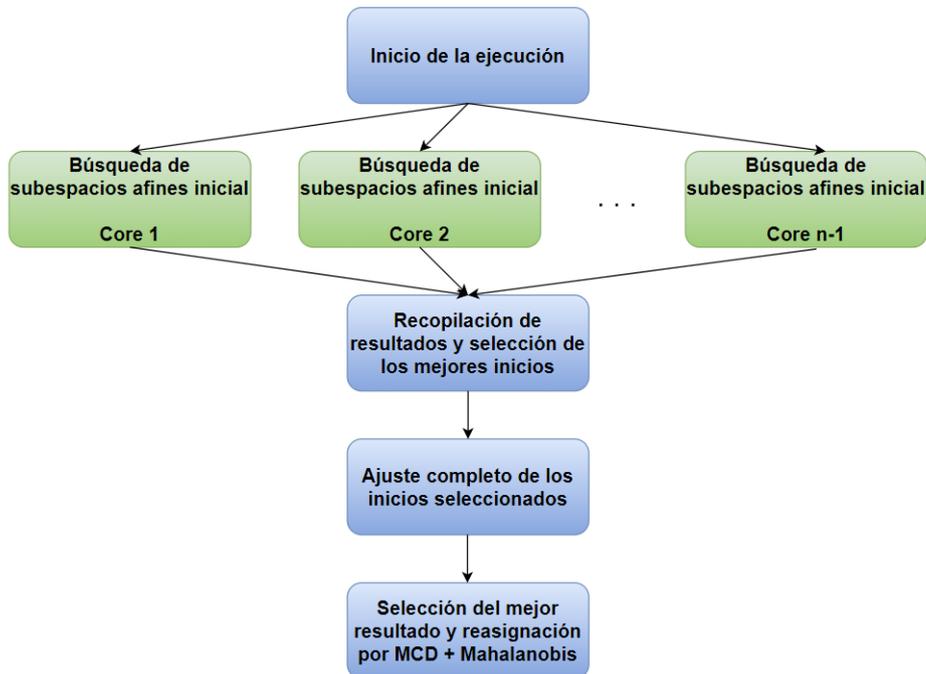


Figura 5.15: Diagrama de flujo de la versión paralela del código.

El esqueleto de la función principal del programa que tiene que ser llamada para realizar el procedimiento es el siguiente:

```

trimksubspaces<-function(puntos,d,alpha,n_starts,plot=NULL, style = 1){
.
.
.
return(list(funcion_coste = optimo, clusters = gruposRobustos))
}
  
```

El primer paso que ha de hacerse es escalar robustamente las variables de los datos que se introduzcan al algoritmo, dividiéndolas entre su desviación mediana absoluta.

```

for(i in 1:dim(puntos)[2]){
puntos[,i]<-puntos[,i]/mad(puntos[,i])
}
  
```

Una vez se disponen de los datos escalados y la función a paralelizar adaptada (sección 5.2.1.1), hay que preparar el entorno de computación paralela. Para ello, hay que seguir los siguientes pasos:

5 IMPLEMENTACIÓN

1. Detectar el número de núcleos disponibles en la máquina donde está ejecutando el código y crear un *cluster* con el número de núcleos deseado. En este caso, todos los núcleos que haya menos 1 para no sobrecargar en exceso la máquina y no comprometer la ejecución de otros procesos coexistentes en segundo plano.
2. Definir una secuencia $\{1, \dots, n\}$ donde n es el número de veces que se quiere ejecutar la función deseada. En nuestro caso, n representa el número de inicializaciones aleatorias del algoritmo.
3. Utilizar una función de la familia `par*apply` a la que se le indica el *cluster* de servidores creado, la secuencia, la función a ejecutar y los argumentos que tiene la función (si los tiene). No hay que olvidar que la función tiene que estar adaptada para soportar el parámetro *dummy* que representa la secuencia. En nuestro caso, la función elegida es `parLapply`, que devolverá los resultados de las n ejecuciones de `rlc_ini` en una lista.
4. Destruir el *cluster* de núcleos creado.

```
if (!require('parallel')) {
  install.packages('parallel')
  library('parallel')
}
cl <- makeCluster(detectCores()-1)
inicializacion<-seq(1,n_starts)
results <- parLapply(cl, inicializacion, rlc_ini, puntos = puntos, d=d, alpha=
  alpha)
stopCluster(cl)
```

Una vez la función `parLapply` ha ejecutado las veces indicadas por el usuario la función `rlc_ini`, se seleccionan las 10 inicializaciones más prometedoras minimizando el valor de la función objetivo. Puede darse el caso de que no haya 10 inicializaciones aleatorias, ya sea o bien porque el usuario ha introducido como argumento `n_starts` un número menor que 10 o bien porque hay muchas inicializaciones que han generado una excepción durante su ejecución por la configuración de los subespacios, la naturaleza de los datos y el parámetro de recorte. En ese caso, se seleccionan todas las disponibles.

Una vez se han iterado completamente las mejores inicializaciones en la función `rlc` se elige la que tiene un menor valor en la función objetivo, y se realiza una última asignación de grupos en base al MCD y la distancia de Mahalanobis robusta, tal y como se detalla en la sección 5.1.2.

```
gruposRobustos<-allocateMahalanobis(dim(puntos)[1],d,grupos)
```

El agrupamiento que devuelve la función `allocateMahanalobis` es el agrupamiento final que devuelve el algoritmo. El valor de la función de coste devuelto no es modificado por las posibles reasignaciones que realice la función `allocateMahanalobis`.

Por último, si se ha indicado que se desea una salida gráfica, se llama a una función que se encarga de realizarla.

```
if(!is.null(plot)){
  if(plot) plottrim3d(gruposRobustos,d, dim(puntos)[2], style)
}
```

El cuerpo de la función que realiza la funcionalidad gráfica no es mostrada en esta memoria al considerarse que no es lo suficientemente compleja.

5.4. Integración de R y C++

En ocasiones, R no es suficientemente rápido. Por mucho que se trate de optimizar todas y cada una de las líneas de código disponibles en un programa, éste es demasiado lento. Para conseguir una versión aún más rápida que la desarrollada mediante la paralelización del código escrito en R y mejorar de manera drástica el rendimiento del procedimiento se desarrollarán las partes más pesadas del algoritmo en otro lenguaje de programación, concretamente C++.

C++ es un lenguaje de programación derivado del lenguaje C cuyo objetivo fue extender el lenguaje C a la orientación a objetos. Una de las características más atractivas de este lenguaje es su eficiencia computacional, motivo por el que se va a utilizar en este trabajo para implementar las porciones más costosas del procedimiento. Debido a que algunas partes del algoritmo no son computacionalmente pesadas y su implementación es mucho más sencilla en R que en C++, se creará un código en formato híbrido, parte del algoritmo será escrito en R y parte en C++.

En la Figura 5.16 se aprecia el *pipeline* de la ejecución del procedimiento en función de si la computación es llevada a cabo por R o por C++. Las partes más costosas del algoritmo, es decir la búsqueda inicial de subespacios afines realizada en varias ocasiones y el ajuste completo de las búsquedas iniciales más prometedoras, van a ser implementadas en C++. El resto del código, encargado de iniciar la ejecución, seleccionar los inicios más prometedores y seleccionar finalmente el mejor resultado, así como la etapa de Mahalanobis y la funcionalidad gráfica, será desarrollado en R.

5 IMPLEMENTACIÓN



Figura 5.16: Diagrama de flujo de la versión que integra R y C++.

Para integrar código C++ en el proyecto de R se hace uso del paquete `Rcpp` [11].

```
if (!require('Rcpp')) {  
  install.packages('Rcpp')  
  library('Rcpp')  
}
```

`Rcpp` ofrece una API (*Application Programming Interface*) que permite intercambiar objetos de R (incluyendo objetos de tipo S3 y S4) entre R y C++, es decir, actúa como *wrapper* entre ambas plataformas. Permite definir funciones en C++ e intercalarlas dentro de un fichero con código R, o bien crear un fichero con código fuente en C++, compilarlo y disponer de las funciones creadas para su uso como si se tratasen de funciones de R. Esta última opción es la escogida en este trabajo.

Otro paquete de gran utilidad que se va a utilizar es `RcppArmadillo` [12], una biblioteca de álgebra lineal con multitud de funciones implementadas en C++ necesarias para la implementación del algoritmo. Por ejemplo, dispone de una función para realizar un Análisis en Componentes Principales, etapa necesaria en el procedimiento a desarrollar. El paquete ha sido desarrollado para proporcionar una sintaxis relativamente similar a la disponible en Matlab.

```
if (!require('RcppArmadillo')) {
  install.packages('RcppArmadillo')
  library('RcppArmadillo')
}
```

5.4.1. Búsqueda de subespacios afines inicial

Como ya se explicó en la sección anterior, la búsqueda de subespacios afines inicial es la parte más costosa del procedimiento, y por ello es candidata a programarse en un lenguaje de programación muy eficiente como lo es C++. El no determinismo que viene inducido por la selección inicial aleatoria de subespacios afines provoca que sea necesario un alto número de inicializaciones para garantizar una solución razonable.

La función, al estar escrita en C++, ha de residir en un fichero con extensión `.cpp`, para que posteriormente el compilador pueda compilarla correctamente. Recordar que esta parte se corresponde con la función *ROBUSTLINEARSUBSPACESINIT* del pseudocódigo desarrollado.

```
// [[Rcpp::export]]
Rcpp::List rlc(Rcpp::NumericMatrix puntos, Rcpp::NumericVector d, double alpha)
{
  .
  .
  .
  return result;
}
```

En la cabecera de la función se encuentra la orden `// [[Rcpp::export]]`, la cuál es necesaria para disponer de la función en la sesión de R desde la que se enlace al fichero `.cpp` en el que se defina la función. Si en ese mismo fichero se define una función sin esa cabecera, entonces no será accesible desde la sesión de R. Esto último es útil, y en este trabajo se ha realizado, en el caso de funciones que sirven de ayuda para realizar una tarea pero que únicamente tienen que ser llamadas desde otro punto del fichero C++.

En este caso no es necesario encerrar la función dentro de un bloque `tryCatch` para evitar que un error en una inicialización afecte a las demás. El *wrapper* que proporciona Rcpp se encarga de gestionar dichos errores y no son propagados hasta el código R que gestiona las llamadas a la función escrita en C++.

En el interior de la función lo primero que ha de hacerse es declarar las variables que serán usadas dentro de la misma.

```
int k = d.length();
```

5 IMPLEMENTACIÓN

```
int n = puntos.nrow();
int p = puntos.ncol();
int n_atip = floor(alpha*n);
Rcpp::NumericVector grupos(n);
Rcpp::NumericMatrix x(k,p);
Rcpp::NumericMatrix di(n,k);
int numRand;
Rcpp::NumericVector minimos(n);
Rcpp::NumericVector minimos2(n);
double lim;
Rcpp::NumericVector temporal;
int asignados = 0;
double funcion_coste = 0;
int iter_ext = 0;
Rcpp::NumericVector gruposAnt(n);
```

Posteriormente, se inicializan los subespacios afines mediante la extracción aleatoria de observaciones del conjunto de datos y se computa la distancia ortogonal de cada observación a cada uno de los k clusters.

```
for(int i = 0; i < k; i++){
  if(d[i] == 0){
    numRand = rand()%n;
    x.row(i) = puntos.row(numRand);
    for(int j = 0; j < n; j++){
      di(j,i) = sqrt(Rcpp::sum(pow(puntos.row(j)-x.row(i),2)));
    }
  } else {

    Rcpp::NumericMatrix ci(d[i]+1,p);
    for(int j = 0; j < (d[i]+1); j++){
      numRand = rand()%n;
      ci.row(j) = puntos.row(numRand);
    }
    for(int j = 0; j < p; j++){
      x(i,j) = Rcpp::mean(ci.column(j));
    }
    Rcpp::NumericMatrix U(d[i],p);
    Rcpp::NumericMatrix eigen_vec(p,p);
    eigen_vec = localpca(ci);
    for(int j = 0; j < d[i]; j++){
      U.row(j) = eigen_vec.column(j);
    }
    Rcpp::NumericMatrix productoMat(p,p);
    arma::mat arma_identidad = arma::eye(p,p);
    arma::mat arma_U = Rcpp::as<arma::mat>(U);
    arma::mat arma_productoMat = arma_identidad-arma_U.t()*arma_U;
```

```

arma::mat arma_puntos = Rcpp::as<arma::mat>(puntos);
arma::mat arma_x = Rcpp::as<arma::mat>(x);

for(int j = 0; j < n; j++){
    di(j,i) = arma::norm(arma_productoMat*(arma_puntos.row(j)-arma_x.row(i)).
t());;
}
}
}

```

Una vez realizada esta etapa, hay que asignar cada observación al subespacio más cercano. Aquellas $\alpha \cdot n$ observaciones con distancias más altas son categorizadas como puntos atípicos, asignándolas al grupo 1000.

```

for(int j = 0; j < n; j++){
    minimos[j] = Rcpp::min(di.row(j));
    minimos2[j] = Rcpp::min(di.row(j));
    temporal = di.row(j);
    grupos[j] = std::min_element(temporal.begin(),temporal.end()) - temporal.
begin();
}
std::sort(minimos.begin(), minimos.end());
if(n_atip>0){
    lim = minimos[n-n_atip];
} else {
    lim = 100000.0;
}
for(int j = 0; j < n; j++){
    if(minimos2[j] > lim){
        grupos[j] = 1000;
        asignados += 1;
    }
}
for(int j = 0; j < n; j++){
    if(minimos2[j] == lim && asignados < n_atip){
        grupos[j] = 1000;
        asignados += 1;
    }
}
}

```

La última etapa es realizar el paso de concentración 9 veces, obteniendo una asignación en *clusters* de las observaciones. Se mantiene la decisión se interrumpir el bucle y la ejecución de la función si en dos iteraciones consecutivas la asignación de las observaciones a los diferentes grupos no varía, significando que en las posteriores iteraciones tampoco variará.

5 IMPLEMENTACIÓN

```
while(iter_ext < 10 && !compara(grupos, gruposAnt)){
    funcion_coste = 0;
    asignados = 0;
    for(int ind = 0; ind < n; ind++){
        gruposAnt[ind] = grupos[ind];
    }
    for(int i = 0; i < k; i++){
        x.row(i) = media(puntos, grupos, i);
        if(d[i] == 0){
            for(int j = 0; j < n; j++){
                di(j,i) = sqrt(Rcpp::sum(pow(puntos.row(j)-x.row(i),2)));
            }
        } else {
            Rcpp::NumericMatrix U(d[i],p);
            Rcpp::NumericMatrix eigen_vec(p,p);
            Rcpp::NumericMatrix puntos_grupo = selecciona_puntos(puntos, grupos, i);
            eigen_vec = localpca(puntos_grupo);
            for(int j = 0; j < d[i]; j++){
                U.row(j) = eigen_vec.column(j);
            }
            Rcpp::NumericMatrix productoMat(p,p);
            arma::mat arma_identidad = arma::eye(p,p);
            arma::mat arma_U = Rcpp::as<arma::mat>(U);
            arma::mat arma_productoMat = arma_identidad-arma_U.t()*arma_U;
            arma::mat arma_puntos = Rcpp::as<arma::mat>(puntos);
            arma::mat arma_x = Rcpp::as<arma::mat>(x);
            for(int j = 0; j < n; j++){
                di(j,i) = arma::norm(arma_productoMat*(arma_puntos.row(j)-arma_x.row(i)
                ).t());
            }
        }
    }
    for(int j = 0; j < n; j++){
        minimos[j] = Rcpp::min(di.row(j));
        minimos2[j] = Rcpp::min(di.row(j));
        temporal = di.row(j);
        grupos[j] = std::min_element(temporal.begin(),temporal.end()) - temporal.
        begin();
    }
    std::sort(minimos.begin(), minimos.end());
    if(n_atip>0){
        lim = minimos[n-n_atip];
    } else {
        lim = 100000.0;
    }
    for(int j = 0; j < n; j++){
        if(minimos2[j] > lim){
            grupos[j] = 1000;
            asignados += 1;
        }
    }
}
```

```

    }
  }
  for(int j = 0; j < n; j++){
    if(minimos2[j] == lim && asignados < n_atip){
      grupos[j] = 1000;
      asignados += 1;
    }
    if(grupos[j] != 1000){
      funcion_coste += minimos2[j];
    }
  }
  funcion_coste /= (n-n_atip)
  iter_ext += 1;
}

```

La asignación de las observaciones a los *clusters* y el valor de la función de coste son introducidos en una lista para ser devueltos como resultado de la ejecución de la función.

```
Rcpp::List result = Rcpp::List::create(grupos, funcion_coste);
```

5.4.2. Iteración completa de las inicializaciones más prometedoras

En la sección anterior se explicó que, en la mayoría de ocasiones, la iteración completa de las inicializaciones más prometedoras no era un proceso lo suficientemente costoso desde el plano computacional como para paralelizarlo entre los distintos núcleos del procesador. Esto sucedía porque al crear el *cluster* de *cores* se necesitaba una cantidad razonablemente alta de tiempo que luego no se veía amortizado al paralelizar la ejecución de la función (en la mayoría de los casos).

Por el contrario, esta problemática no acontece al programar la función en un *script* C++ que, posteriormente, se integra en R mediante un *wrapper*. Las llamadas a las funciones escritas en C++ desde una sesión de R son inmediatas, y es palmaria la mejoría en tiempos de procesamiento que requiere una misma pieza de código escrita en C++ frente a su versión en R.

Por esta razón, esta etapa es implementada en el *script* de C++. Al ser una función que necesita ser llamada desde la sesión de R, hay que incluir en su cabecera la orden `// [[Rcpp::export]]`.

```

// [[Rcpp::export]]
Rcpp::List rlc2(Rcpp::NumericMatrix puntos, Rcpp::NumericVector d, double alpha
, Rcpp::NumericVector grupos) {
  .
  .
  .
  return result;
}

```

5 IMPLEMENTACIÓN

La función `rlc2` itera la asignación a subespacios afines inicial que es pasada en el argumento `grupos` hasta que o bien los grupos a los que se asigna cada observación no varían en 2 iteraciones consecutivas o bien se alcanza un número máximo de iteraciones. El número máximo de iteraciones es fijado en 100 al igual que en el caso de la versión paralelizada del código, permitiendo una posterior comparación de tiempos. Al igual que en el caso de la función anterior, lo primero consiste en declarar las variables que se van a utilizar dentro de la función.

```
int k = d.length();
int n = puntos.nrow();
int p = puntos.ncol();
int n_atip = floor(alpha*n);
Rcpp::NumericMatrix x(k,p);
Rcpp::NumericMatrix di(n,k);
Rcpp::NumericVector minimos(n);
Rcpp::NumericVector minimos2(n);
double lim;
Rcpp::NumericVector temporal;
Rcpp::NumericVector gruposAnt(n);
int asignados = 0;
double funcion_coste = 0;
int flag = 0;
```

Tras esta etapa previa de definición de variables, se procede a, en un bucle `while`, ajustar en profundidad la asignación a subespacios afines pasada como argumento de la función.

```
while(flag < 100 && !compara(grupos, gruposAnt)){
    funcion_coste = 0;
    asignados = 0;
    for(int ind = 0; ind < n; ind++){
        gruposAnt[ind] = grupos[ind];
    }
    for(int i = 0; i < k; i++){
        x.row(i) = media(puntos, grupos, i);
        if(d[i] == 0){
            for(int j = 0; j < n; j++){
                di(j,i) = sqrt(Rcpp::sum(pow(puntos.row(j)-x.row(i), 2)));
            }
        } else {
            Rcpp::NumericMatrix U(d[i], p);
            Rcpp::NumericMatrix eigen_vec(p, p);
            Rcpp::NumericMatrix puntos_grupo = selecciona_puntos(puntos, grupos, i);
            eigen_vec = localpca(puntos_grupo);
            for(int j = 0; j < d[i]; j++){
                U.row(j) = eigen_vec.column(j);
            }
            Rcpp::NumericMatrix productoMat(p, p);
```

```

arma::mat arma_identidad = arma::eye(p,p);
arma::mat arma_U = Rcpp::as<arma::mat>(U);
arma::mat arma_productoMat = arma_identidad-arma_U.t()*arma_U;
arma::mat arma_puntos = Rcpp::as<arma::mat>(puntos);
arma::mat arma_x = Rcpp::as<arma::mat>(x);

    for(int j = 0; j < n; j++){
        di(j,i) = arma::norm(arma_productoMat*(arma_puntos.row(j)-arma_x.row(i)
).t());;
    }
}
}
for(int j = 0; j < n; j++){
    minimos[j] = Rcpp::min(di.row(j));
    minimos2[j] = Rcpp::min(di.row(j));
    temporal = di.row(j);
    grupos[j] = std::min_element(temporal.begin(),temporal.end()) - temporal.
begin();
}
std::sort(minimos.begin(), minimos.end());
if(n_atip>0){
    lim = minimos[n-n_atip];
} else {
    lim = 100000.0;
}
for(int j = 0; j < n; j++){
    if(minimos2[j] > lim){
        grupos[j] = 1000;
        asignados += 1;
    }
}
for(int j = 0; j < n; j++){
    if(minimos2[j] == lim && asignados < n_atip){
        grupos[j] = 1000;
        asignados += 1;
    }
    if(grupos[j] != 1000){
        funcion_coste += minimos2[j];
    }
}
funcion_coste /= (n-n_atip)
flag += 1;
}

```

La asignación de las observaciones a los *clusters* y el valor de la función de coste son introducidos en una lista para ser devueltos como resultado de la ejecución de la función.

5 IMPLEMENTACIÓN

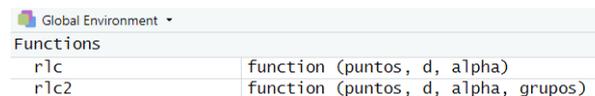
```
Rcpp::List result = Rcpp::List::create(grupos,funcion_coste);
```

5.4.3. Flujo general del programa

El flujo general del programa es dirigido desde la sesión de R. Gracias al *wrapper* proporcionado por el paquete Rcpp las funciones definidas en el *script* C++ son llamadas bajo la misma sintaxis y pautas que rigen las llamadas a funciones de R. El primer paso es compilar el código fuente C++ donde se encuentren las funciones definidas.

```
sourceCpp('script.cpp')
```

Una vez compilado, las funciones estarán disponibles en el *environment* de R, de la misma manera que lo estarían funciones R definidas de la manera clásica (Figura 5.17).



Global Environment	
Functions	
rlc	function (puntos, d, alpha)
rlc2	function (puntos, d, alpha, grupos)

Figura 5.17: *Global environment* de RStudio con funciones definidas en un código fuente C++ accesibles tras compilarle.

El nombre, argumentos y valores de retorno de la función `trimksubspaces` son idénticos a los mostrados en la variante paralelizada, por lo que no se volverán a mostrar. El primer paso a realizar es escalar robustamente las variables de los datos que se introduzcan al algoritmo, dividiéndolas entre su desviación mediana absoluta.

```
for(i in 1:dim(puntos)[2]){  
  puntos[,i] <- puntos[,i] / mad(puntos[,i])  
}
```

Una vez se disponen de los datos escalados y las funciones definidas en el código fuente C++, se computan las asignaciones iniciales (tantas computaciones como indicadas en el argumento `n_starts`).

```
clusters <- matrix(nrow = n_starts, ncol = dim(puntos)[1])  
funciones_coste <- rep(0, n_starts)  
for(inicializacion in 1:n_starts){  
  fit <- rlc(puntos, d, alpha)  
  clusters[inicializacion,] <- fit[[1]]  
  funciones_coste[inicializacion] <- fit[[2]]  
}
```

5.5 Análisis comparativo de tiempos de ejecución

Como ya se explicó en el caso de la variante paralelizada, una vez se han computado las asignaciones iniciales a *clusters* para las distintas inicializaciones, se seleccionan las 10 más prometedoras mediante la minimización del valor de la función objetivo. Puede darse el caso de que no haya 10 inicializaciones aleatorias, ya sea o bien porque el usuario ha introducido como argumento *n_starts* un número menor que 10 o bien porque hay muchas inicializaciones que han generado una excepción durante su ejecución. En ese caso, se seleccionan todas las disponibles.

Los comienzos seleccionados se iteran completamente en la función *rlc2* y se elige el que minimice la función objetivo.

```
for(omega in 1:n_optimos){
  fit<-rlc2(puntos,d,alpha,grupos_optimos[omega,])
  resultados_grupo<-rbind(resultado_grupo,fit[[1]])
  resultados_funcion_coste<-c(resultado_ecm,fit[[2]])
}
optimo<-min(resultados_funcion_coste)
flag<-which.min(resultados_funcion_coste)
grupos<-resultados_grupo[flag,]
```

Por último, se realiza una última asignación de grupos en base al determinante de mínima covarianza y la distancia de Mahalanobis robusta y se realizan los gráficos correspondientes si es que el usuario que ha utilizado el procedimiento así lo desea.

```
gruposRobustos<-allocateMahalanobis(dim(puntos)[1],d,grupos)
if(!is.null(plot)){
  if(plot) plottrim3d(gruposRobustos,d, dim(puntos)[2], style)
}
```

La asignación a subespacios afines obtenida tras realizar la etapa que combina las distancias de Mahalanobis con el determinante de mínima covarianza es la asignación final ofrecida por el algoritmo. Al igual que en la versión paralelizada del código, el valor de la función de coste obtenido tras la selección del mejor ajuste no es modificado por las posibles reasignaciones que se realicen en la etapa donde se computa el MCD.

5.5. Análisis comparativo de tiempos de ejecución

Para cuantificar la eficiencia de las distintas implementaciones del procedimiento de análisis *cluster* entono a subespacios afines se van a realizar dos estudios de medición de tiempos. En el primero, se comparará la celeridad en el ajuste de subespacios de las 3 versiones implementadas (secuencial, paralelizada e híbrida). En el segundo, se comparará la versión implementada más eficiente con una función disponible en el repositorio CRAN de R.

5 IMPLEMENTACIÓN

5.5.1. Secuencial vs paralelización vs integración R & C++

Para realizar la primera comparación y con el objetivo de reflejar la respuesta de las diferentes implementaciones ante problemas de distinto tamaño se diseñan cuatro experimentos de distinta magnitud.

En el primero, el que menos tiempo de cómputo requerirá, se generan 200 observaciones en \mathbb{R}^2 con el fin de agrupar entorno a dos *clusters* de dimensión 0. El segundo consta de 1500 puntos en \mathbb{R}^3 , tratando de agrupar las observaciones entorno a 3 subespacios de dimensiones 0, 1 y 2 respectivamente. Elevando la carga computacional, en el tercero se generan 5000 observaciones en \mathbb{R}^3 , agrupándolas entorno a 5 grupos lineales. Por último, el experimento más exigente de todos consta de 100000 puntos en \mathbb{R}^5 . En este último, las observaciones se agruparán entorno a 10 *clusters* de dimensión 0.

Dado que un computador la presencia de procesos en segundo plano es muy común y éstos pueden afectar al rendimiento del resto de procesos, cada experimento se realiza 5 veces por implementación. Se promedian los 5 tiempos de cómputo y los resultados pueden verse en el Cuadro 5.1.

Versión	Exp. 1	Exp. 2	Exp. 3	Exp. 4
R secuencial	1.70	24.22	150.22	3261.27
R paralelizado (4 núcleos)	1.44	13.05	86.86	1989.58
R & C++	0.09	1.82	9.15	269.38

Cuadro 5.1: Tiempo promedio en segundos de las versiones sobre los experimentos.

La diferencia en el tiempo de computación entre la versión que integra los dos lenguajes de programación y las otras dos versiones es muy marcada, mostrando la tremenda eficiencia de C++ respecto a lo que se logra con R, incluso en su versión paralelizada. En los 4 experimentos, la versión híbrida es respectivamente 16, 7, 9 y 7 veces más rápida que la versión paralelizada. Este ahorro de tiempo computacional puede desempeñar un papel crítico en aplicaciones donde el tiempo sea escaso y muy valioso, por ejemplo en procesos industriales.

A su vez, la versión paralelizada utilizando 4 núcleos del procesador supone un ahorro más que razonable sobre la versión secuencial. En los 4 experimentos, la versión paralelizada es respectivamente un 20 %, 85 %, 72 % y 64 % más rápida que la versión secuencial. Es esperable que a mayor número de *cores* que se utilicen para la computación, mayor será la ventaja sobre la versión secuencial, aunque cabe reseñar que la adición de núcleos para realizar la computación y la reducción del tiempo de procesamiento no guardan una relación lineal, llegando en cierto punto a un estancamiento en el tiempo necesario para ajustar los subespacios afines.

5.5 Análisis comparativo de tiempos de ejecución

En los diagramas de cajas de las Figuras 5.18, 5.19, 5.20 y 5.21 en los que se representa el logaritmo de los 5 tiempos de ejecución frente a la versión de implementación por cada experimento se aprecia como la versión híbrida es la clara vencedora de las comparaciones. Hay mucha más diferencia temporal entre la versión híbrida y la versión paralelizada que entre ésta última y la versión secuencial.

Además, se ajusta una recta de regresión a los tiempos logarítmicos de cada experimento según la versión (Figura 5.22). La evolución de las rectas es muy similar, apreciándose que la que menos pendiente tiene es la ajustada con los 20 tiempos disponibles de la versión paralelizada.

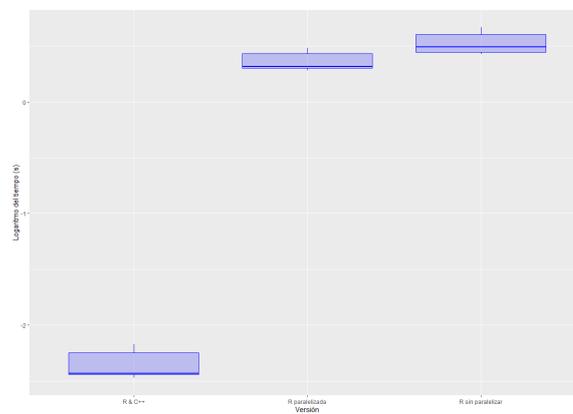


Figura 5.18: Diagrama de cajas del tiempo en escala logarítmica frente a la versión (Exp. 1).

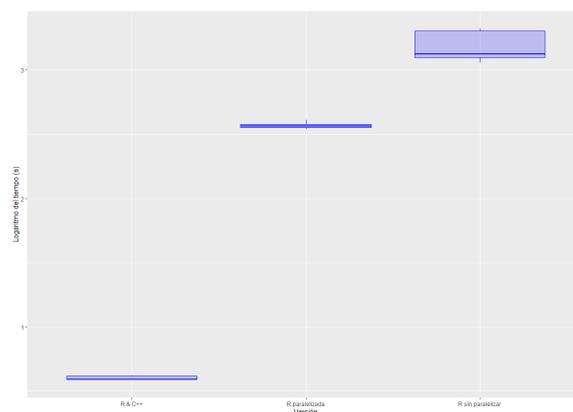


Figura 5.19: Diagrama de cajas del tiempo en escala logarítmica frente a la versión (Exp. 2).

5 IMPLEMENTACIÓN

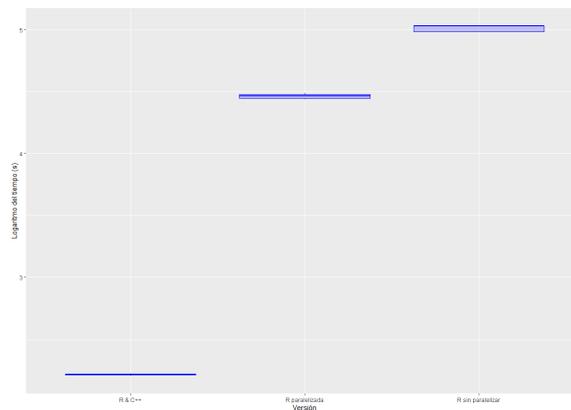


Figura 5.20: Diagrama de cajas del tiempo en escala logarítmica frente a la versión (Exp. 3).

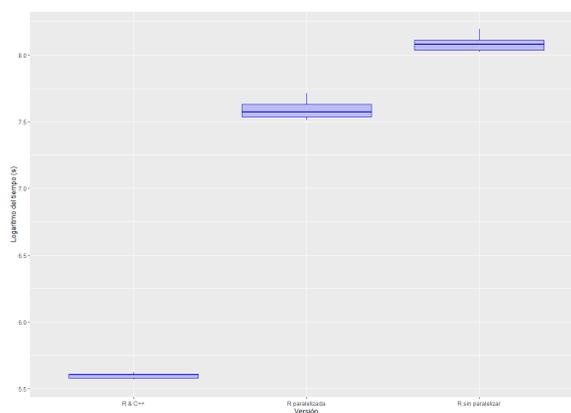


Figura 5.21: Diagrama de cajas del tiempo en escala logarítmica frente a la versión (Exp. 4).

Esta apreciación se confirma viendo la pendiente de cada recta ajustada en el Cuadro 5.2. Si los experimentos realizados y la respuesta de las funciones fuesen un reflejo de la casuística total de posibles combinaciones a darse, esto significaría que, con conjuntos de datos de cada vez más tamaño, la función propia sería la vencedora en cuanto a que tardaría menos tiempo en realizar el *clustering*. Sin embargo, el escaso tamaño muestral de tiempos obtenido (5 mediciones para cada par función-experimento) no permite sacar conclusiones fiables.

Versión	Intercept	Pendiente
R secuencial	-1.927	2.452
R paralelizado (4 núcleos)	-2.150	2.359
R & C++	-4.850	2.546

Cuadro 5.2: *Intercept* y pendiente de las rectas de regresión ajustadas.

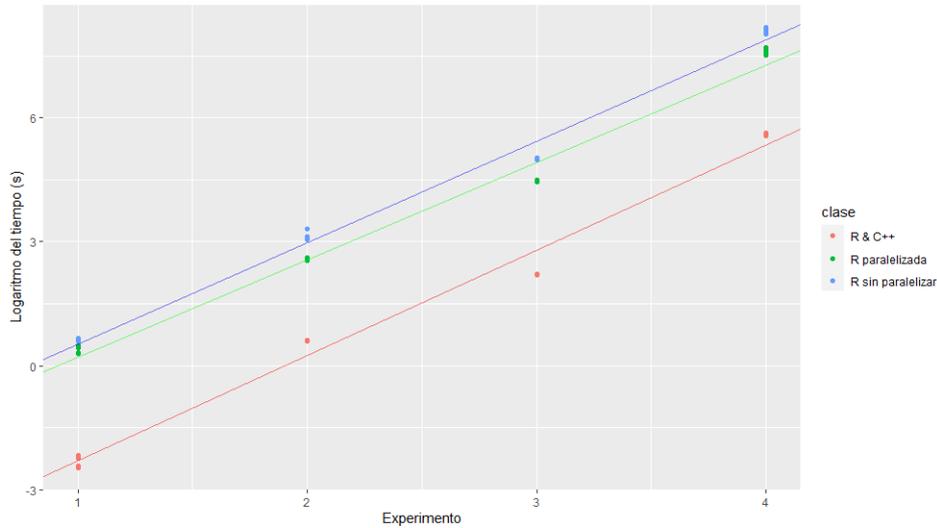


Figura 5.22: Regresión lineal a los tiempos medidos.

5.5.2. Función `trimkmeans` vs función `trimksubspaces`

En esta segunda comparación, se va a enfrentar la versión del algoritmo implementado más eficiente (R & C++) con una función de un paquete disponible en el CRAN (*Comprehensive R Archive Network*) de R. Esta función, llamada `trimkmeans`, pertenece al paquete `trimcluster` [13], desarrollado por C.Hennig, y permite ajustar el método de las k -medias recortadas a un conjunto de datos.

Como la función `trimkmeans` únicamente permite realizar agrupaciones entorno a centroides, los experimentos diseñados para realizar la comparación de tiempos tienen la característica de que los subespacios afines entorno a los que agrupar son de dimensión 0. Cualquier otra combinación de dimensiones de los subespacios soportada por la función `trimksubspaces` (función implementada en este trabajo) no sería soportada por la función del paquete `trimcluster`.

El número de experimentos diseñados es 3. En el primero experimento, computacionalmente el menos costoso, se generan 200 observaciones en \mathbb{R}^2 con el fin de agrupar entorno a dos *clusters* de dimensión 0. El segundo consta de 3000 puntos en \mathbb{R}^2 , tratando de agrupar las observaciones entorno a 2 centroides. Por último, constituyendo el experimento más exigente, en el tercero se generan 50000 observaciones en \mathbb{R}^3 . El número de *clusters* en este último caso será de 5.

Al igual que anterior, cada par función-experimento es realizado 5 veces, recogiendo los tiempos de computación empleados. El promedio de dichos tiempos según función y experimento puede apreciarse en el Cuadro 5.3.

5 IMPLEMENTACIÓN

Versión	Exp. 1	Exp. 2	Exp. 3
<code>trimkmeans</code>	0.687765	26.92146	3650.621
<code>trimksubspaces</code>	0.07548022	1.929447	57.65721

Cuadro 5.3: Tiempo promedio en segundos de las funciones sobre los experimentos.

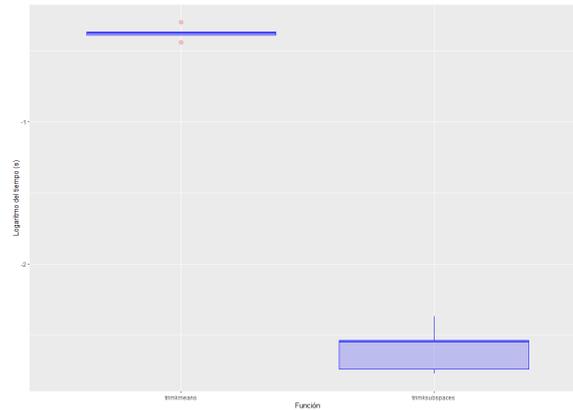


Figura 5.23: Diagrama de cajas del tiempo en escala logarítmica frente a la función (Exp. 1).

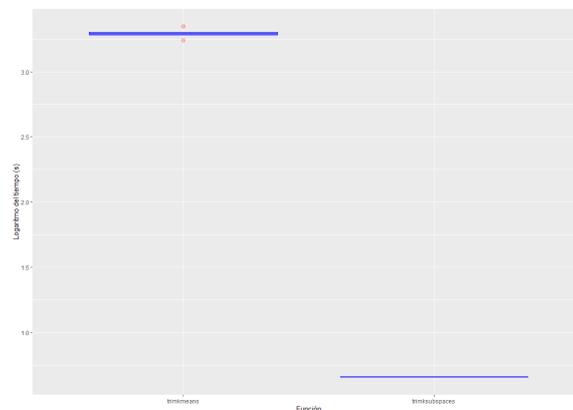


Figura 5.24: Diagrama de cajas del tiempo en escala logarítmica frente a la función (Exp. 2).

La diferencias en el tiempo de computación entre ambas versiones son evidentes. En los 3 experimentos realizados, la función construida en este trabajo es respectivamente 91, 13 y 63 veces más rápida que la función disponible en CRAN, todo ello a pesar de que la función creada es más potente y permite agrupar entorno a subespacios de cualquier dimensión, incluso mezclando dimensiones.

5.5 Análisis comparativo de tiempos de ejecución

En los diagramas de cajas de las Figuras 5.23, 5.24 y 5.25 en los que se representa el logaritmo de los 5 tiempos de ejecución frente a la función utilizada por cada experimento se observa como la función implementada en este trabajo tiene un rendimiento muy superior a la que podemos encontrar en los repositorios de R. Al igual que se hizo con la comparación anterior, se ajusta una recta de regresión a los tiempos logarítmicos de cada experimento según la versión (Figura 5.26). La evolución de las rectas es bastante diferente, apreciándose que la que menos pendiente tiene es la ajustada con los 15 tiempos disponibles de la función desarrollada en ese trabajo.

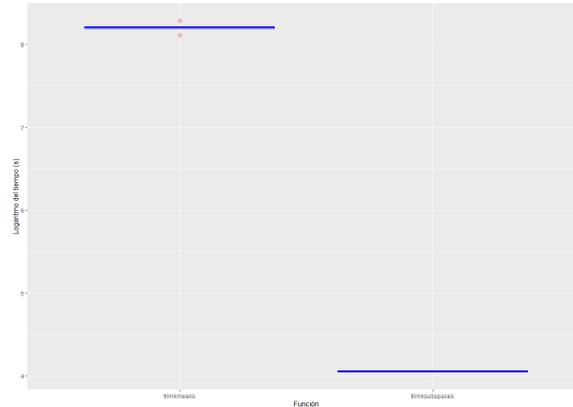


Figura 5.25: Diagrama de cajas del tiempo en escala logarítmica frente a la función (Exp. 3).

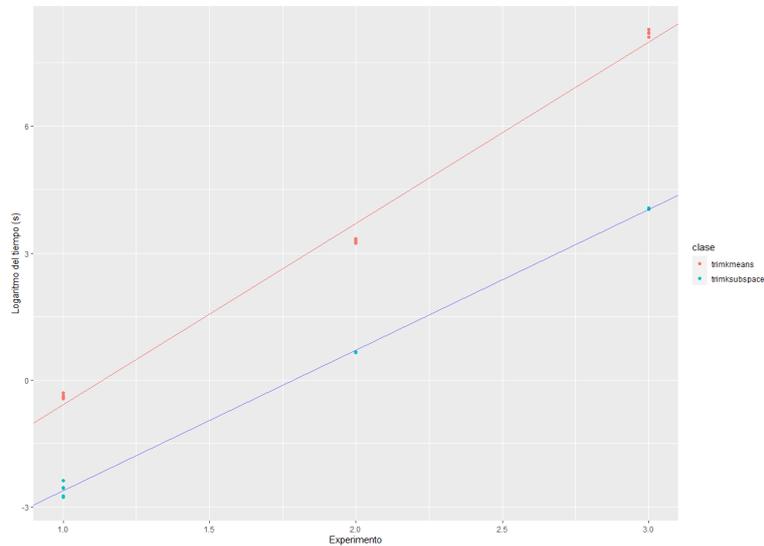


Figura 5.26: Regresión lineal a los tiempos medidos.

Esto se confirma viendo la pendiente de cada recta ajustada en el Cuadro 5.4. Si los experimentos realizados y la respuesta de las versiones fuesen un reflejo de la casuística total de posibles

5 IMPLEMENTACIÓN

combianciones a darse, esto significaría que, con conjuntos de datos de cada vez más tamaño, la función desarrollada en este trabajo sería la que se impondría en cuanto a que tardaría menos tiempo en ajustar los subespacios. El escaso tamaño muestral de 5 tomas de tiempo por cada par función-experimento no permite sacar conclusiones fiables.

Versión	Intercept	Pendiente
<code>trimkmeans</code>	-4.870	4.288
<code>trimksubspaces</code>	-5.944	3.325

Cuadro 5.4: *Intercept* y pendiente de las rectas de regresión ajustadas.

6. Aplicación real en la segmentación de imágenes

El análisis *cluster* tiene una amplia variedad de aplicaciones en el panorama actual científico. Para ilustrar un ejemplo de aplicación real del algoritmo desarrollado se va a realizar segmentación de imágenes, en concreto segmentación basada en colores.

El proceso de dividir una imagen en múltiples regiones o segmentos es conocido como segmentación. El objetivo de esta técnica es dividir una imagen en regiones que, al contener menos información, pueden ser más fáciles de analizar.

Típicamente, el proceso de segmentación de imágenes se utiliza para localizar objetos y fronteras contenidos en una imagen [14]. Ahondando más en el tema, esto se logra etiquetando cada píxel de una imagen en una categoría. Es decir, la segmentación de imágenes puede considerarse un problema de clasificación donde cada píxel ha de ser etiquetado en una categoría distinta, donde todos los píxeles que compartan etiqueta significa que tienen una característica común: pertenecen a un mismo cuerpo o a un objeto de las mismas características típicamente (Figura 6.1).

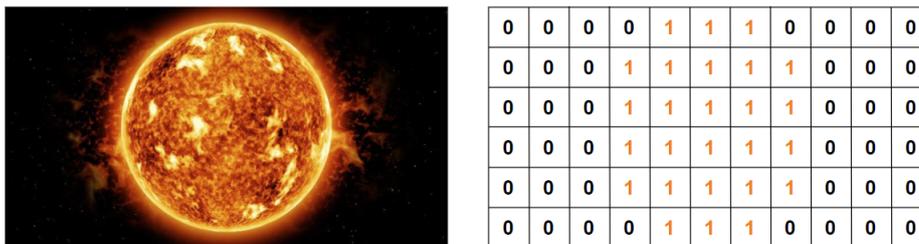


Figura 6.1: Segmentación de imágenes.

La segmentación de imágenes es una tarea vital en una amplia gama de ramas del conocimiento. La segmentación de imágenes médicas [15] o en el mundo del industria [16] tiene mucho interés y multitud de modelos en el estado del arte son desarrollados para estas tareas.

Para automatizar el proceso de segmentación sobre imágenes complejas, lo más común es aprovechar la información proporcionada por los colores presentes en la imagen. Píxeles cercanos en color son asociados a un mismo objeto y se clasifican en una misma categoría.

Es en esta tarea, en la búsqueda de relaciones entre píxeles en base a la información proporcionada por su color, es donde el análisis *cluster* y en concreto el procedimiento desarrollado pueden sacar ventaja sobre el resto de técnicas, al ser un paradigma excelente en la búsqueda de patrones que relacionen elementos entre sí (píxeles en este caso).

La imagen que se va a utilizar para ilustrar la aplicación del análisis *cluster* robusto entorno a

6 APLICACIÓN REAL EN LA SEGMENTACIÓN DE IMÁGENES

subespacios afines es la de la Figura 6.2.

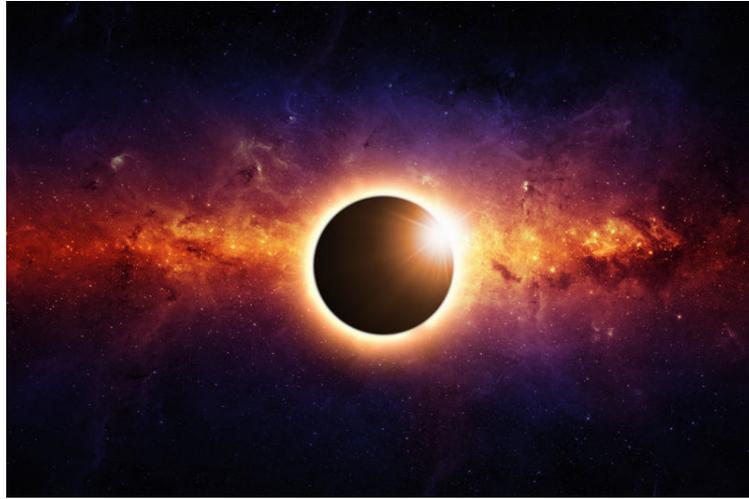


Figura 6.2: Imagen base a la que se va a aplicar *clustering*.

La imagen es de 800x533 píxeles y se encuentra en 3 canales: R (*Red*), G (*Green*) y B (*Blue*). La imagen se aplana, formando un conjunto de datos de 426400 observaciones en \mathbb{R}^3 . Los 426400 puntos representan los 800x533 píxeles, y por cada píxel se recogen 3 valores: su intensidad en el canal R, su intensidad en el canal G y su intensidad en el canal B (Figura 6.3). Los valores de intensidad oscilan entre 0 y 1, correspondiéndose el 1 con el valor más alto posible (típicamente 255) y 0 el valor más bajo posible.

		R	G	B
Pixel 1		1	0	0
Pixel 2		0.34	0.14	0.39
Pixel 3		1	0.91	0
	⋮			
Pixel 426400		0	0	0

Figura 6.3: Conversión imagen-conjunto de datos.

6.1. *Clustering* de una imagen en 3 grupos.

El primer experimento que se va a realizar es el de generar las imágenes de segmentación únicamente clasificando los píxeles en 3 posibles valores. Para ello, se utiliza el procedimiento de análisis *cluster* robusto entorno a subespacios afines desarrollado en su versión en **C++** al ser la más eficiente y la que soporta una mayor carga computacional con diferencia. La decisión de usar esta versión es debido al tamaño masivo del conjunto de datos, con prácticamente medio millón de observaciones.

Las configuraciones de dimensiones de los subespacios afines que se utilizan del algoritmo son todas las posibles para realizar agrupamientos en \mathbb{R}^3 , es decir:

1. Tres grupos entorno a subespacios de dimensión 0 (*k*-medias con $k = 3$).
2. Dos grupos entorno a subespacios de dimensión 0 y un grupo entorno a un subespacio de dimensión 1.
3. Dos grupos entorno a subespacios de dimensión 1 y un grupo entorno a un subespacio de dimensión 0.
4. Tres grupos entorno a subespacios de dimensión 1.

Todas las configuraciones son usadas sin recortar ninguna observación ($\alpha = 0\%$), para no perder información de la imagen, y con 50 inicios aleatorios.

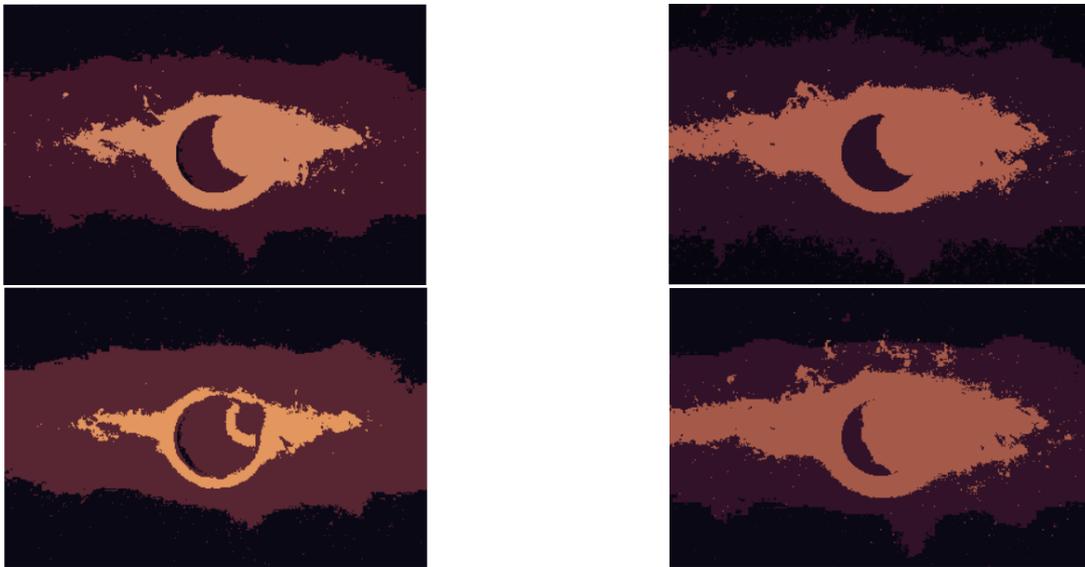


Figura 6.4: Imágenes de segmentación obtenidas tras realizar los agrupamientos entorno a dimensiones: 0-0-0, 0-0-1, 0-1-1 y 1-1-1 (de izquierda a derecha y de arriba a abajo).

6 APLICACIÓN REAL EN LA SEGMENTACIÓN DE IMÁGENES

En la Figura 6.4 se aprecia como las imágenes de segmentación tienden a formar agrupaciones de píxeles bajo los mismos colores más alargadas conforme se utiliza el algoritmo con subespacios de dimensión unitaria en detrimento de subespacios de dimensión 0. Estas agrupaciones pueden aportar un valor añadido diferencial a lo que ya aporta el algoritmo de las k -medias al permitir asociar entre sí partes más débiles de las imágenes que pueden ser objeto de interés en multitud de dominios de aplicación.

6.2. *Clustering* de una imagen en 6 grupos.

El segundo experimento continúa la línea del primero, pero en vez de clasificar los píxeles en únicamente 3 categorías se van a clasificar en 6. Igualmente se utiliza el procedimiento de análisis *cluster* robusto entorno a subespacios afines desarrollado en su versión en C++.

Las configuraciones de dimensiones de los subespacios afines que se utilizan del algoritmo son todas las posibles para realizar agrupamientos en \mathbb{R}^3 , es decir:

1. Seis grupos entorno a subespacios de dimensión cero (k -medias con $k = 6$).
2. Cinco grupos entorno a subespacios de dimensión cero y un grupo entorno a un subespacio de dimensión uno.
3. Cuatro grupos entorno a subespacios de dimensión cero y dos grupos entorno a subespacios de dimensión uno.
4. Tres grupos entorno a subespacios de dimensión cero y tres grupos entorno a subespacios de dimensión uno.
5. Cuatro grupos entorno a subespacios de dimensión uno y dos grupos entorno a subespacios de dimensión cero.
6. Cinco grupos entorno a subespacios de dimensión uno y un grupo entorno a un subespacio de dimensión cero.
7. Seis grupos entorno a subespacios de dimensión uno.

Al igual que en el experimento anterior, todas las configuraciones son usadas sin recortar ninguna observación ($\alpha = 0\%$), para no perder información de la imagen, y con 50 inicios aleatorios.

Observando las imágenes de segmentación generadas en la Figura 6.5, igualmente se observa que los grupos de píxeles que comparten categoría (color) tienden a ser más alargados conforme hay más subespacios afines de dimensión 1 entorno a los que agrupar. Como era de esperar, las imágenes de segmentación generadas por agrupaciones entorno a 6 subespacios son de mayor

calidad que las generadas por agrupaciones entorno a 3 subespacios, generalizando y resumiendo de manera más óptima la información presente en la imagen original.

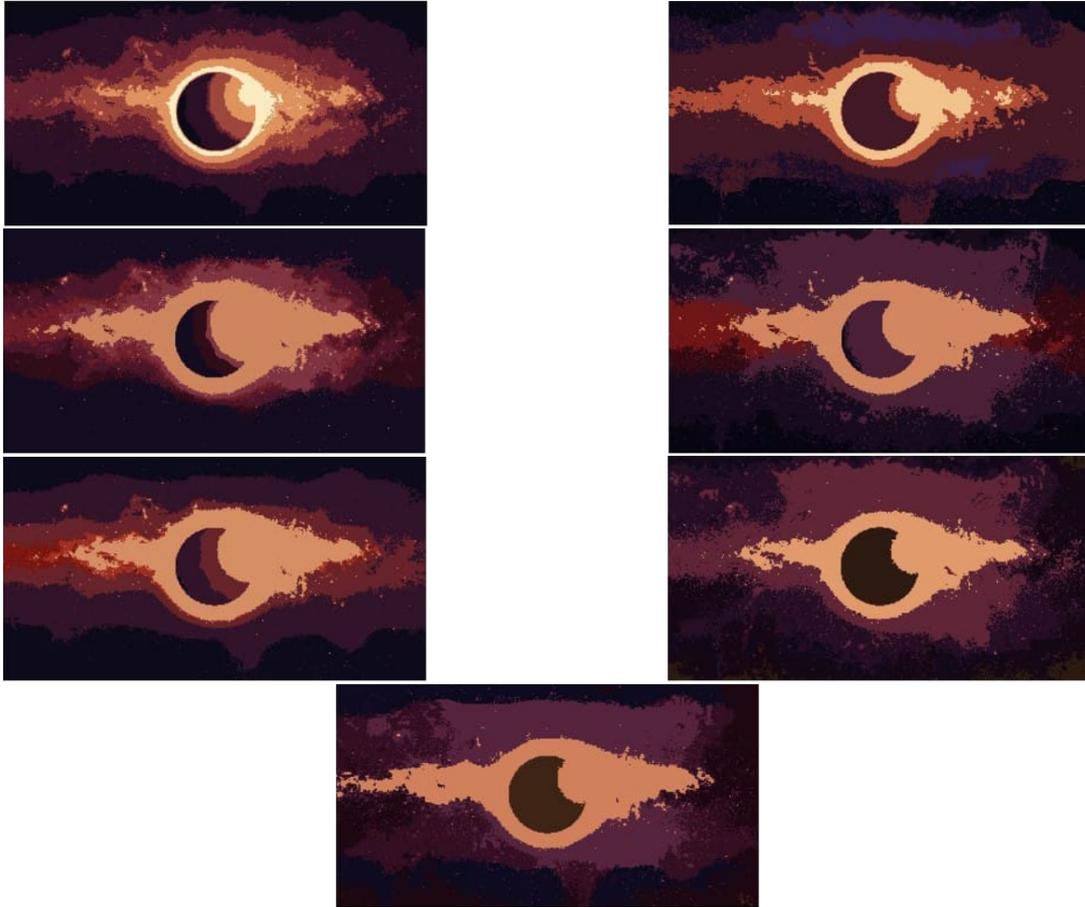


Figura 6.5: Imágenes de segmentación obtenidas tras realizar los agrupamientos entorno a dimensiones: 0-0-0-0-0, 0-0-0-0-1, 0-0-0-0-1-1, 0-0-0-1-1-1, 0-0-1-1-1-1, 0-1-1-1-1-1 y 1-1-1-1-1-1 (de izquierda a derecha y de arriba a abajo).

6.3. *Clustering* de una imagen con ruido

El tercer y último experimento que se va a realizar tiene como objetivo mostrar la robustez estadística del procedimiento desarrollado. Para ello, a la imagen de la Figura 6.2 se le va a añadir una cantidad de ruido aleatorio sobre ciertos píxeles. Posteriormente, se procederá a realizar *clustering* sobre la imagen con ruido con el algoritmo desarrollado, tanto aplicando recortes ($\alpha > 0$) como sin ellos ($\alpha = 0$), con el fin de determinar si en la versión en la que se aplican recortes las observaciones recortadas son aquellos píxeles contaminados.

Para ello, primero se muestra la máscara de segmentación generada a partir de la Figura 6.2

6 APLICACIÓN REAL EN LA SEGMENTACIÓN DE IMÁGENES

con la configuración del algoritmo $d = (0, 0, 0, 0)$ y $\alpha = 0$ (Figura 6.6).



Figura 6.6: Imagen de segmentación obtenida tras realizar los agrupamientos con $d = (0, 0, 0, 0)$.

Posteriormente, se contaminan un 2% de los píxeles de la imagen original, dando lugar a la imagen presentada en la Figura 6.7. Los píxeles con ruido contienen en los canales R y B un valor aleatorio proveniente de una distribución uniforme continua $U(0, 0.05)$, mientras que el canal G contiene un valor aleatorio extraído de una distribución $N(0.9, 0.0004)$.

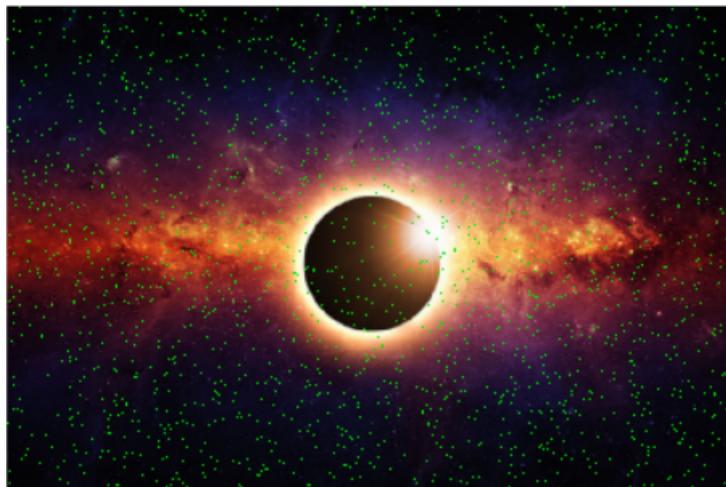


Figura 6.7: Imagen base con un 2% de píxeles contaminados.

Las imágenes de segmentación generadas (Figura 6.8) utilizando configuraciones del procedimiento con $\alpha = 0$ y $\alpha = 0.02$ muestran la robustez del método. En la primera configuración, los píxeles contaminados influyen en la determinación de los subespacios afines y por tanto en

el color de la región de la que forman parte. Sin embargo, asignando a α el valor 0.02 se logra detectar como puntos atípicos a todas las observaciones contaminadas (color blanco en la imagen), y los colores de la máscara de segmentación generada son idénticos a los obtenidos con la imagen original (sin contaminar).

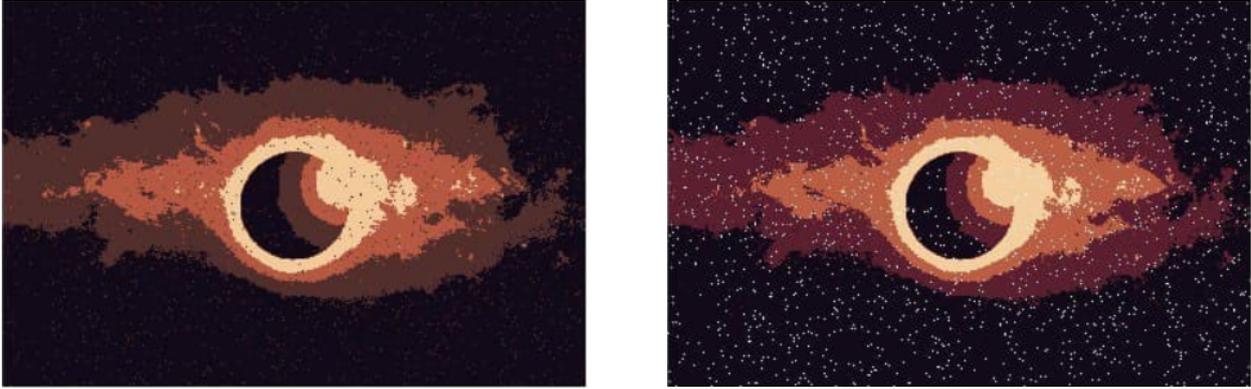


Figura 6.8: Imágenes de segmentación obtenidas tras realizar los agrupamientos con $\alpha = 0$ y $alpha = 0.02$ ($d = (0, 0, 0, 0)$).

De igual manera, y para probar la robustez estadística del procedimiento desarrollado con otra configuración de dimensiones y otra proporción de píxeles contaminados, se genera una máscara de segmentación a partir de la Figura 6.2 con la configuración del algoritmo $d = (1, 1, 1, 1)$ y $\alpha = 0$ (Figura 6.9).



Figura 6.9: Imagen de segmentación obtenida tras realizar los agrupamientos con $d = (1, 1, 1, 1)$.

Ahora, se contaminan un 1.5% de los píxeles de la imagen original, obteniendo la imagen disponible en la Figura 6.10. De igual manera, los píxeles con ruido contienen en los canales R y

6 APLICACIÓN REAL EN LA SEGMENTACIÓN DE IMÁGENES

B un valor aleatorio proveniente de un distribución uniforme continua $U(0, 0.05)$, mientras que el canal G contiene un valor aleatorio extraído de una distribución $N(0.9, 0.0004)$.

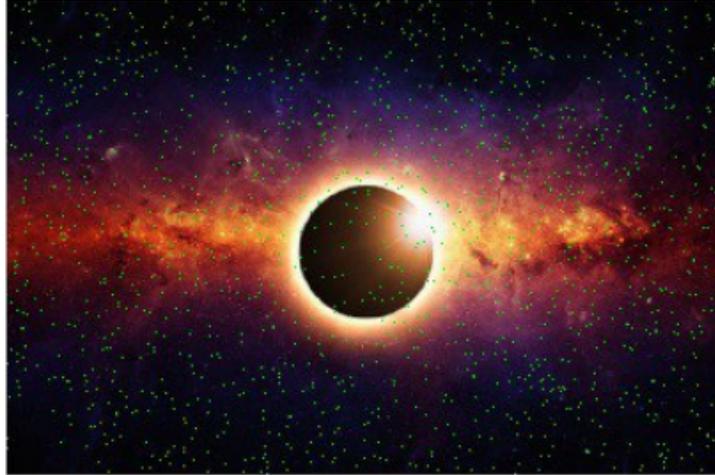


Figura 6.10: Imagen base con un 1.5 % de píxeles contaminados.

Las mismas conclusiones obtenidas con la configuración anterior se pueden apreciar en este caso. Las imágenes de segmentación generadas (Figura 6.11) utilizando configuraciones del procedimiento con $\alpha = 0$ y $\alpha = 0.02$ muestran la robustez del método. En la primera configuración, los píxeles contaminados toman importancia en la determinación del color de la región de la que forman parte. Sin embargo, asignando a α el valor 0.015 se logra detectar como puntos atípicos a todas las observaciones contaminadas (color blanco en la imagen), y los colores de la máscara de segmentación generada son idénticos a los obtenidos con la imagen original (sin contaminar).

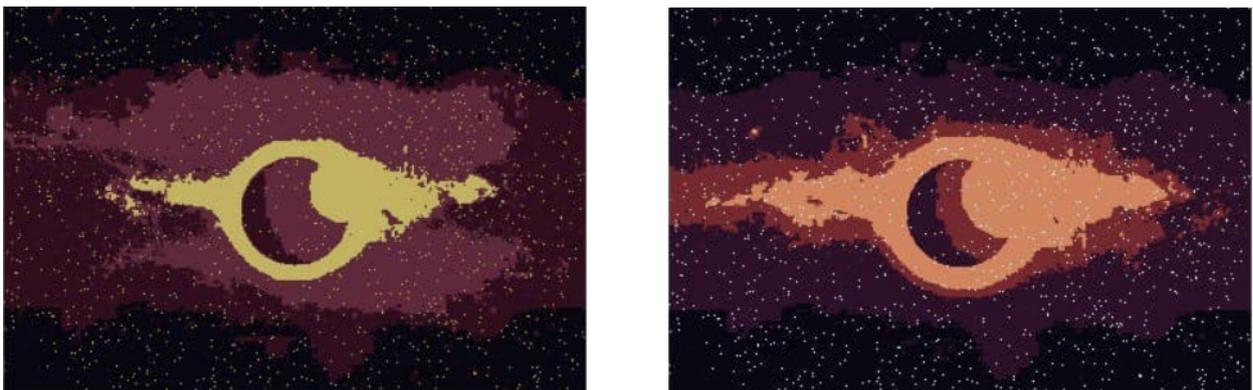


Figura 6.11: Imágenes de segmentación obtenidas tras realizar los agrupamientos con $\alpha = 0$ y $alpha = 0.015$ ($d = (1, 1, 1, 1)$).

7. Conclusiones y líneas futuras

7.1. Conclusiones

En este trabajo se ha estudiado con profundidad un procedimiento para realizar análisis *cluster* robusto entorno a subespacios afines y se han implementado diferentes versiones del mismo con el objetivo de valorar qué aproximación es la más eficiente.

El análisis *cluster* robusto entorno a subespacios afines permite agrupar observaciones alrededor de estructuras lineales. Además, incluyendo el enfoque de los recortes, los propios datos son los que deciden qué observaciones son las más atípicas y distan de la tónica general del conjunto de datos, evitando que dichos puntos atípicos participen en la determinación de los subespacios afines.

Para conseguir una versión funcional del algoritmo aplicable a los conjuntos de datos de gran tamaño presentes en la actualidad se han explorado 3 enfoques: un enfoque secuencial, uno paralelizado y uno híbrido.

En el primero de los enfoques, el secuencial, se implementó de la manera tradicional el procedimiento de análisis *cluster* robusto en R. Se comprobó como dicha implementación era muy lenta, prácticamente inutilizable en conjuntos de datos de tamaño medio y alto.

Tras la primera implementación secuencial se probó un segundo enfoque, utilizando la potencia y las prestaciones que ofrece la computación paralela, también en R. Se comprobó que se producía una mejoría considerable de los tiempos de computación. Concretamente, paralelizando el código en 4 núcleos, se produjo un ahorro de tiempo medio en los 4 experimentos lanzados de entorno al 60 % respecto a la versión secuencial.

Sin embargo, y ante la necesidad de hacer más eficiente aún el procedimiento, se exploró un enfoque híbrido, donde las partes más livianas del algoritmo serían desarrolladas en R y las partes más pesadas en un lenguaje de programación compilado, concretamente C++, un lenguaje que destaca por su eficiencia computacional.

La incorporación de C++ a la implementación en esta versión dual resultó en la obtención de unos tiempos de computación excelentes. Para los 4 experimentos diseñados, esta versión resultó ser en promedio unas 10 veces más rápida que la versión paralelizada, que a su vez ya estaba notablemente optimizada respecto a la versión secuencial.

Además, también se ha mostrado como la comparación entre la implementación híbrida desarrollada y un función de un paquete contribuido disponible en el CRAN de R que realiza un caso particular del problema tratado en este trabajo arroja como ganador a la versión desarrollada en este documento, aún siendo ésta más mucho más completa que su rival.

7 CONCLUSIONES Y LÍNEAS FUTURAS

Por último, se ilustra un campo de interés y un ejemplo de aplicación real del procedimiento desarrollado. Concretamente, en la segmentación de imágenes. Representando cada píxel de una imagen como un vector en \mathbb{R}^3 con los valores de los canales R, G y B y realizando agrupamientos de los píxeles con el procedimiento desarrollado se puede obtener una máscara de segmentación de la imagen original. Variando los parámetros del algoritmo como por ejemplo el número de grupos y las dimensiones de los subespacios entorno a los que agrupar se obtienen diferentes máscaras, las cuales pueden ser de gran utilidad. La segmentación de imágenes tiene multitud de aplicaciones críticas en el mundo de la medicina y de la industria, como lo son la detección de enfermedades y de defectos de fabricación respectivamente.

7.2. Líneas futuras

El principal inconveniente del algoritmo desarrollado, y de multitud de métodos que se encuentran bajo el paraguas del aprendizaje no supervisado, es que hay que determinar de antemano el número de *clusters* entorno a los que se quiere agrupar.

Queda para futuros trabajos el estudio, desarrollo e implementación de técnicas para que el propio algoritmo ayude al usuario a determinar cuál es el número óptimo de subespacios afines entorno a los que agrupar. De igual manera, la dimensión de los subespacios afines también es un hiperparámetro que sería interesante automatizar. No obstante, el disponer de algoritmos como estos (computacionalmente eficientes) es un primer paso necesario para tratar de resolver este problema porque muchos procedimientos para elegir parámetros se basan en monitorizar las particiones obtenidas al mover los mismos.

En el plano computacional, probar una versión paralelizada del enfoque híbrido desarrollado es probable que reduzca aún más los tiempos necesarios para ajustar el procedimiento de análisis *cluster* robusto.

Por último, construir un paquete contribuido de R y subirle al repositorio CRAN sería una interesante línea futura de trabajo para que la comunidad de ciencia de datos, y en concreto la comunidad estadística, pueda aprovechar las ventajas y multitud de aplicaciones de este procedimiento de análisis *cluster*. El disponer de implementaciones computacionalmente eficientes es un primer paso imprescindible para poder desarrollar esta librería.

Referencias

- [1] Cuesta-Albertos, J. A. and Gordaliza, A. and Matrán, C. (1997). Trimmed k-means: an attempt to robustify quantizers. *Ann. Statist.* 25 (2) 553 - 576.
- [2] Garcia-Escudero, L. A. and Gordaliza, A. (1999). Robustness properties of k means and trimmed k means. *Journal of the American Statistical Association*, 94(447), 956-969.
- [3] Van Aelst, S. and Wang, X. and Zamar, R. and Zhu, R. (2006). Linear grouping using orthogonal regression. *Computational Statistics & Data Analysis*. 50. 1287-1312. 10.1016/j.csda.2004.11.011.
- [4] García-Escudero, L. A. and Gordaliza, A. and San Martín, R. and Van Aelst, S. and Zamar, R. (2009). Robust linear clustering. *Journal of the Royal Statistical Society Series B*. 71. 301-318. 10.1111/j.1467-9868.2008.00682.x.
- [5] Gordaliza, A. (1991). Best approximations to random variables based on trimming procedures. *Journal of Approximation Theory*, Volume 64, Issue 2, Pages 162-180.
- [6] García-Escudero, L. A. and Gordaliza, A. and Mayo-Íscar, A. and San Martín, R. (2010). Robust clusterwise linear regression through trimming. *Computational Statistics & Data Analysis*, 54(12), 3057-3069.
- [7] García-Escudero, L. A. and Gordaliza, A. and Matrán, C. (2003). Trimming Tools in Exploratory Data Analysis. *Journal of Computational and Graphical Statistics*. 12. 434-449. 10.1198/1061860031806.
- [8] Rousseeuw, P. J. and Van Driessen, K. (1999). A fast algorithm for the Minimum Covariance Determinant estimator. *Technometrics*, 41:212–223.
- [9] Hubert, M. and Debruyne, M. and Rousseeuw, P. J. (2017). Minimum Covariance Determinant and Extensions.
- [10] Parallel processing in R. Recuperado el 10 de Abril de 2021 de <https://dept.stat.lsa.umich.edu/~jerrick/courses/stat701/notes/parallel.html>
- [11] Rcpp gallery. Recuperado el 28 de Enero de 2021 de <https://gallery.rcpp.org/>
- [12] Eddelbuettel, D. and Francois, R. and Bates, D. and Ni, B. Package RcppArmadillo (2021). Recuperado el 30 de Enero de 2021 de <https://cran.r-project.org/web/packages/RcppArmadillo/RcppArmadillo.pdf>
- [13] Hennig, C. Paquete trimcluster. (2020). Recuperado el 15 de Marzo de 2021 de <https://cran.r-project.org/web/packages/trimcluster/trimcluster.pdf>

REFERENCIAS

- [14] Khattab, D. and Ebied, H. and Hussein, A. and Tolba, M. (2014). Color Image Segmentation Based on Different Color Space Models Using Automatic GrabCut. *TheScientificWorldJournal*. 126025. 10.1155/2014/126025.
- [15] Shafi, A. and Padha, D. (2019). Medical Image Segmentation A Review of Recent Techniques, Advancements and a Comprehensive Comparison. *International Journal of Computer Sciences and Engineering*. 7. 114-124. 10.26438/ijcse/v7i7.114124.
- [16] Chéron, E. and Kleinschmidt, E. J. (1985). A review of industrial market segmentation research and a proposal for an integrated segmentation framework. *International Journal of Research in Marketing*. 2. 101–115. 10.1016/0167-8116(85)90027-8.