

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo de Fin de Grado

Desarrollo de un paquete de clasificación
jerarquizada en R para su posterior aceptación en
CRAN

Autor: Roberto Alcántara Escribano

Tutor/es: Juan José Cuadrado Gallego

TRIBUNAL:

Presidente: Luis Usero

Vocal 1º: Miguel Ángel Herranz

Vocal 2º: Juan José Cuadrado

FECHA: 2 de Octubre de 2020

Contents

1	RESUMEN	4
1.1	Palabras clave:	4
2	INTRODUCCIÓN	4
3	MARCO TEÓRICO Y SOCIAL	5
3.1	Lenguaje R	6
3.2	Análisis clúster	7
3.2.1	Clustering jerárquico	9
3.2.2	Clustering no jerárquico o repartitivo.	11
3.2.3	Clustering jerárquico correlativo.	12
3.2.4	Distancia entre los datos.	13
3.2.5	Proximidad o Similitud entre los clusters.	14
4	PAQUETE R	14
4.1	¿Qué es un paquete?	14
4.1.1	Aportar nuevos paquetes.	17
4.2	Creación del paquete R	18
4.3	Estructura del paquete R	20
4.3.1	Archivo <i>DESCRIPTION</i>	20
4.3.2	Archivo <i>NAMESPACE</i>	26
4.3.3	Directorio <i>R</i>	26
4.3.4	Directorio <i>man</i>	27
4.3.5	Directorio <i>src</i>	27
4.3.6	Directorio <i>data</i>	28
4.3.7	Directorio <i>inst</i>	29
4.3.8	Directorio <i>vignettes</i>	29
4.3.9	Directorio <i>test</i>	29
4.3.10	Directorio <i>demo</i>	30
4.4	Viñetas.	31
4.5	Construcción de paquetes.	31
5	DESCRIPCIÓN EXPERIMENTAL	32
5.1	Software necesario	32
5.2	Creación paquete.	35
5.2.1	Archivos <i>.R</i>	37
5.2.2	Documentación de paquetes.	38
5.2.3	Manual de usuario.	41
5.2.4	Viñetas.	41
5.2.5	Construcción del paquete.	44
5.3	Algoritmos incluidos en el paquete.	45
5.3.1	Clustering Jerárquico Aglomerativo (Agglomerative Hierarchical Clustering)	47
5.3.2	Clustering Jerárquico Divisivo (Divisive Hierarchical Clustering)	49

5.3.3	Clustering Jerárquico por Correlación (Correlative Hierarchical Clustering)	51
5.4	Publicación del paquete en CRAN	53
6	CONCLUSIONES	57
7	VIÑETA	58
8	MANUAL DE USUARIO	95
9	BIBLIOGRAFÍA	164

1 RESUMEN

Castellano: Este documento describe el proceso seguido para desarrollar un paquete en R sobre clasificación jerarquizada. Tanto las necesidades técnicas para su implementación, los elementos que forman el paquete, las bases teóricas que lo sustentan y la documentación asociada a las funciones incluidas en él. Se detallan los archivos necesarios para la ejecución del problema y un manual de usuario para utilizar el paquete sin conocimientos previos.

English: This document describes the process followed in order to develop an R package about hierarchical classification. It includes technical necessities for its implementation, the elements that make the package, the theoretical basis which support it and the associated paperwork to the included functions in it. In addition, the necessary files are detailed so as to execute the problem, and a user manual in order to make use of the package without having previous knowledge.

1.1 Palabras clave:

Paquete R, clasificación jerarquizada, Rstudio, matriz de distancia, dendograma, desarrollo R, CRAN.

2 INTRODUCCIÓN

Este documento recoge el proceso de desarrollo de un paquete en lenguaje R sobre clasificación jerarquizada. Pretende agrupar diferentes algoritmos de procesamiento de información mediante técnicas de análisis clúster. Se utilizará el lenguaje R por estar desarrollado específicamente para el manejo de datos almacenados masivamente y realizar sobre ellos análisis estadísticos y matemáticos.

Este lenguaje es una herramienta utilizada por muchas personas por todo el mundo, y cada vez aumenta más esa comunidad, por eso, es importante mejorar el material ya disponible y aportar contenidos nuevos o más claros, poniéndolos a disposición de todos los usuarios en los sitios web oficiales.

El objetivo principal de este paquete es recoger todos los algoritmos de análisis de clusterización jerarquizada, documentar y explicar su funcionamiento, como adaptar los datos a las características del algoritmo para potenciar su funcionamiento y permitir que, tanto estudiantes de la materia, como usuarios frecuentes de esta tecnología dispongan de esta información. Se incluyen funciones auxiliares que realizan funciones parciales de los algoritmos y otras que explican en detalle los pasos seguidos por el algoritmo, mostrando el estado de los datos y las clasificaciones parciales de los datos. Esto permite al usuario más inexperto comprender el funcionamiento del algoritmo ejecutado con los datos reales y conocidos que ha introducido.

Se añade, además, un nuevo algoritmo de agrupación de datos numéricos, centrado en las propiedades de los objetos en estudio que permita obtener los candidatos más próximos o similares a unos valores dados por el usuario con las características que se buscan. Este algoritmo se detalla en profundidad en secciones posteriores de este documento, tanto su explicación matemática como ejemplos de su aplicación.

CRAN (The Comprehensive R Archive Network) es un conjunto de servidores con almacenamiento de archivos que recogen los paquetes de R desarrollados por usuarios del lenguaje tras ser aprobados por expertos dedicados a este fin. De esta manera, los autores de los paquetes participan en la evolución del lenguaje y de sus contenidos, poniendo a disposición de otros usuarios el trabajo realizado durante sus investigaciones, ofreciendo documentación y mantenimiento para todo aquel que lo necesite. Es importante aportar contenido a esta organización para compartir conocimientos nuevos y que R sea una herramienta cada vez más potente, por eso, que cumpla los requisitos exigidos por CRAN para su posterior publicación será otro de los objetivos principales de nuestro desarrollo.

Posteriormente se presentará el panorama actual que ha promovido la aparición de lenguajes como R y otros medios para el tratamiento de datos. También se detallarán las características técnicas que influyen en el desarrollo del paquete y se describirán en profundidad los diferentes algoritmos recogidos en el producto final.

3 MARCO TEÓRICO Y SOCIAL

Actualmente, la información es uno de los bienes más útiles para las empresas, para los gobiernos o para cualquier individuo que observe el mundo y quiera obtener conclusiones a partir de esas observaciones. La ciencia de los datos (Data Science) es una ciencia que se dedica exclusivamente al estudio de los datos, por eso, está basada en el almacenamiento masivo de estos y en su procesamiento posterior para conseguir tomar decisiones acordes al comportamiento y las relaciones observadas entre ellos.

El almacenamiento masivo de datos exige herramientas de todo tipo que faciliten su interpretación, pues los volúmenes de datos sin procesar no aportan gran valor a la entidad que los ha recogido. Por todo esto, es importante desarrollar sistemas capaces de realizar esta tarea. El problema de todo esto (que supone una ventaja para todos a largo plazo), es que no es posible definir los objetivos de una manera concisa. Los datos son información, pero la interpretación de estos no está estandarizada, por lo que su estudio dependerá del objetivo particular de cada análisis. Por ejemplo, y bajo los mismos datos sobre productos de un supermercado, podríamos obtener información sobre cuales son los productos que más se compran, o realizar un estudio sobre que productos se compran a la vez. Ambas conclusiones se sacan de las mismas muestras, pero no están relacionados entre sí, el objetivo es diferente.

La ciencia de los datos, combina estadística, matemática e informática para facilitar el procesamiento de toda esta información, aparentemente poco expresiva, para obtener conclusiones a partir de ella. Es una ciencia nueva, en constante evolución, buscando exprimir la información que puedan aportar los datos para obtener ventajas en el desarrollo de la actividad, económica principalmente, de cada empresa. Existen grupos investigadores dedicados a mejorar esta ciencia y cada vez es un entorno más amplio y conocido.

Además de la ciencia de los datos, han aparecido nuevos conceptos como el Big Data, los macrodatos y otras ciencias dedicadas al procesamiento de toda la información recogida. Estas actividades cada vez son más importantes en las empresas o en los entornos donde se requieren, por este motivo, en los últimos tiempos existen perfiles dedicados exclusivamente a este fin.

Otras herramientas fundamentales, que han surgido como una consecuencia de la aparición y la evolución de estas nuevas necesidades, son utilizadas por la ciencia de los datos para llevar a cabo su función. La primera, y una de las más importantes, es el lenguaje R, dedicado exclusivamente al manejo de datos. Aunque es un lenguaje relativamente antiguo (desarrollado en 1993) ha aumentado su influencia debido a la importancia del procesamiento de datos en la actualidad; se han ampliado mucho los ámbitos donde se utiliza, y destaca la facilidad en el manejo y procesamiento de grandes volúmenes de datos. También existen otros lenguajes, no dedicados exclusivamente a este fin, pero muy apropiados para el tratamiento de datos, como Python, potenciando sus funciones para el procesamiento de la información.

Tanto Hadoop (Apache Software Foundation, 2006) inicialmente, como Spark (Universidad de Berkeley, 2009), son frameworks nuevos, de no más de quince años. Son plataformas open source de Apache y se dedican específicamente al tratamiento masivo de los datos, a buscar la eficiencia en el procesamiento de las operaciones. Ambos nacieron a partir del auge del análisis de datos, y son las herramientas más utilizadas en las compañías especializadas. Los dos tienen una arquitectura similar, pero Spark apareció como una evolución necesaria que agilizase el procesamiento en operaciones, en mantenimiento y en tiempo de ejecución de Hadoop. Que ambos dispongan de licencia libre ha ayudado a su expansión por todo el mundo, y por este motivo, no existen otros frameworks más utilizados actualmente.

3.1 Lenguaje R

R, como se ha comentado previamente, es un lenguaje dedicado al análisis estadístico y matemático, creado para el estudio de grandes volúmenes de datos. Es una implementación de software libre a partir del lenguaje S (un lenguaje de programación estadística anterior a R y pionero en la materia). Estaba basado en subrutinas implementadas en Fortran, pero el rendimiento y las condiciones de tratamiento de los datos con volúmenes demasiado grandes no eran eficientes. El crecimiento de esta actividad exigía la mejora de las herramientas disponibles, y de ahí la aparición de R.

Esta herramienta incluye muchas opciones que permiten la visualización de los datos y facilita la interpretación de los mismos, ofreciendo distintos resultados que se ajustarán a los objetivos particulares de cada usuario. Dispone de gran cantidad de opciones, es el usuario el que decidirá como aplicarlas.

Forma parte de GNU y se distribuye con licencia GNU GLP (General Public License), es decir, es software libre y esta disponible para todo el que quiera utilizarlo; se garantiza a los usuarios finales que puedan compartirlo y modificarlo sin restricciones, por todo esto ha alcanzado la repercusión de la que dispone actualmente. Este lenguaje fue el más utilizado para el manejo de datos, su procesamiento y la representación gráfica de los mismos.

Por todo ello, será el lenguaje utilizado para el desarrollo de este paquete.

3.2 Análisis clúster

El análisis clúster es una técnica de análisis que tiene como objetivo clasificar objetos formando grupos de objetos (denominados clúster) heterogéneos entre sí, pero en el que los elementos que lo componen deben ser lo más homogéneos posibles. Es una herramienta para estudiar datos conocidos total o parcialmente, pero no es una función para clasificación de datos no conocidos (Los datos se clasifican en función del contenido de cada grupo, y se reorganizan a medida que se añaden nuevos objetos a cada grupo y, por tanto, las medidas de similitud se actualizan en cada iteración).

Debemos destacar que estas técnicas de agrupamiento sirven para observar una realidad almacenada en datos, permite obtener información de los datos, explorar sus relaciones (descubrir si existen patrones ocultos que definan el comportamiento), o supervisar la evolución de las muestras obtenidas. Todas estas técnicas no tienen un objetivo concreto y definido, las conclusiones las obtendrá el analista encargado de obtener los resultados. Por ello, existirán gran cantidad de conclusiones diferentes, en función de la intención del analista, de la estructuración de los datos de la muestra y de las variables utilizadas de cada objeto en la ejecución de los algoritmos.

Por otra parte, los distintos algoritmos utilizan los datos de manera diferente, pueden tener criterios de agrupación diferentes y considerar similitud entre objetos de muchas formas diferentes. Todo esto amplía el número de análisis posibles y por tanto sus interpretaciones posteriores.

Este análisis, exige la definición de las características que van a distinguir unos grupos de otros, y que permitirá clasificar cada objeto en el grupo correspondiente, es decir, se definirán las variables de cada objeto implicadas en el cálculo de la similitud entre objetos.

Antes de profundizar en cuestiones técnicas, es recomendable indicar algunos inconvenientes de estos análisis:

- Como se ha indicado al comienzo, no es una técnica inferencial, es decir, no se deben extrapolar conclusiones a muestras de datos más amplias o formadas por valores diferentes. Aunque los resultados finales, pueden aportar valor al analista encargado si conoce la materia y desea obtener una idea general sobre el comportamiento de datos similares no incluidos en el estudio.
- Las conclusiones obtenibles dependen de las variables y las medidas de similitud impuestas por el analista, por lo que serán diferentes para analistas diferentes.
- Es importante que el analista tenga conocimientos sobre la materia de los datos, de esta manera, podrá decidir si las clasificaciones obtenidas son significativas o no, pudiendo incluso eliminar datos anómalos.

En cualquier técnica de clusterización, el criterio de asignación de los objetos a los grupos es la “distancia”, entendiendo esta como la similitud entre los distintos elementos que forman cada clúster y el objeto a clasificar. Esta similitud, puede estar sujeta a características cuantitativas o cualitativas, dependerá de las decisiones tomadas en el análisis por parte del sujeto que lo realice.

Un ejemplo de esta división en clústeres a partir de características cualitativas sobre el conjunto de toda la población, considerando a cada individuo como un objeto particular con características claramente definidas, sería la agrupación por sexo (hombre o mujer), por nacionalidad (limitando objetos con más de una nacionalidad), o distinguiendo si cada objeto es mayor o menor de edad, por ejemplo. En este caso, definiríamos un clúster por cada valor que pueda tomar la característica clasificadora.

Sin embargo, otra clasificación que atienda a características cuantitativas permitiría la asignación de cada objeto a un clúster en función de su edad. Con este tipo de clasificación, el número de clústeres no está definido, y se ajustará al conjunto de objetos a clasificar y a los criterios de los encargados del análisis (grupos de rangos de edades generalmente). Es posible, como se comentará en apartados posteriores, que los datos se clasifiquen en cada clúster en función de los miembros de cada grupo ya formado, es decir, a partir de un número de grupos indicado por los autores del análisis, los datos se unirán a cada uno de ellos atendiendo a la característica de similitud que se considere, pasando a formar parte del clúster con los elementos más homogéneos con el nuevo objeto.

Las técnicas de análisis de clusterización se dividen en dos grandes grupos según la clasificación final de los objetos, aunque añadiremos un nuevo tipo de clasificación, no sujeto a estas características donde incluiremos nuestro nuevo algoritmo de clasificación:

- Clusterización jerárquica: estas técnicas parten de clústeres simples formados por un único objeto. El algoritmo de clasificación agrupa dos clústeres similares en cada iteración, es decir, comprueba, a partir de los objetos que componen cada clúster, cuál del conjunto total no agrupado tiene una similitud mayor con los elementos que forman el resto de clústeres en cada instante (por medio de la matriz de distancias) y los agrupa. De este modo, la importancia radica en el orden de agrupamiento de los objetos o clústeres, y no de la clasificación en distintos grupos.

Se deduce, por tanto, que únicamente existirá un clúster final, y la clasificación atenderá a la iteración en la que ha sido agrupado cada objeto a partir del clúster donde se incluía. Existe una clasificación jerárquica divisiva que supone un estudio inverso al comentado previamente, en él, se parte de un único cluster con todos los elementos y se dividirán los clusters considerados menos similares, es decir, cuya distancia sea máxima. El algoritmo se explicará en detalle en la sección correspondiente de este mismo documento.

- Clusterización no jerárquica o Repartitiva. Esta técnica, por el contrario, parte de un conjunto de clústeres definidos por el analista. En la primera iteración, todos los objetos creados en este momento actualizarán el valor de los centroides definidos inicialmente, y se recalculará de nuevo la matriz de distancias de todos los objetos a los nuevos clústeres. Después, en las iteraciones del algoritmo de clasificación, los objetos de cada clúster cambiarán o no de clúster en función de la actualización de sus similitudes. El algoritmo finalizará cuando exista una iteración donde no se produzcan cambios de objetos entre clúster, es decir, cuando se alcance el estado de estabilidad en sus distancias.

- Como hemos indicado anteriormente, definiremos un nuevo grupo de algoritmos de clasificación para incluir el nuestro. El objetivo final de nuestro algoritmo será crear diferentes clusters, como ocurría en la clusterización repartitiva, pero ordenados por semejanza a un valor objetivo como en el caso de la clasificación jerárquica.

El algoritmo ordenará todos los valores conocidos, atendiendo a los datos numéricos de los objetos en estudio y al valor objetivo dado. Además, se incluye una normalización de los valores y la posibilidad de otorgar pesos o niveles de importancia en la clasificación a las propiedades de los objetos. Este algoritmo se explicará detalladamente en secciones posteriores.

3.2.1 Clustering jerárquico

Previamente se ha explicado el funcionamiento de este tipo de análisis, que se basa en la agrupación o división jerárquica de los diferentes objetos. La solución final no atiende a que grupo pertenece cada elemento en estudio, si no en que nivel de la jerarquía ha sido añadido o agrupado en los clusters solución si se trata de clasificación jerárquica aglomerativa o en que momento se ha dividido de otros clusters existentes por ser poco similares a los miembros de un

cluster si se trata de clasificación jerárquica divisiva.

Existen dos grandes tipos de métodos jerárquicos:

3.2.1.1 Clustering jerárquico Aglomerativo. [38] Los métodos jerárquicos aglomerativos comienzan con tantos clústeres como objetos formen el conjunto completo a clasificar. En cada iteración, se recalculará la matriz de distancia entre todos los grupos y se unirán los dos clústeres indicados como más similares por la matriz de distancia. Finalizará con un único clúster conteniendo todos los elementos.

Estos algoritmos, constan de 4 pasos diferenciados, que se repiten hasta obtener una solución:

1. Asignamos un cluster a cada elemento del estudio, de tal manera que si existen N elementos, inicializaremos N clusters de un único elemento.
2. Calcularemos la distancia entre todos los clusters existentes en cada iteración, es decir, generaremos una matriz de distancias entre todos ellos. Para realizar este paso, es importante definir la similitud que consideraremos entre los clusters, y la definición que utilizaremos como distancia entre puntos.
3. Seleccionamos la distancia mínima de la matriz y creamos un nuevo cluster uniendo los elementos de los clusters correspondientes a esa distancia mínima previamente calculada. En este momento, los clusters agrupados son eliminados, pues sus elementos se encuentran en el nuevo cluster obtenido.
4. Repetiremos los pasos 2 y 3 hasta obtener un único cluster que incluya todos los elementos del análisis.

3.2.1.2 Clustering jerárquico Divisivo. [38] Por otro lado, este tipo de algoritmos comenzarán con un clúster formado por todos los objetos del análisis y en cada iteración, se dividirá el objeto o subcluster más heterogéneo con respecto al resto. En este ultimo caso, la división afectará al cluster más heterogéneo, y se crearán dos clusters diferentes a partir de uno previo, siendo este el formado por los elementos menos similares entre sí.

Los pasos a seguir para realizar esta clasificación serán:

1. Creamos un cluster que incluya todos los elementos del análisis, de tal manera que si existen N elementos, inicializaremos un cluster con N componentes.

2. Desarrollaremos todas las posibles combinaciones de clusters que se podrán obtener a partir de los clusters existentes anteriormente, sin mezclar elementos de clusters ya divididos. En este paso, es importante no descomponer nuevos clusters que utilicen elementos de dos clusters previos ya divididos, pues en nuestro árbol de búsqueda final se produciría un bucle no permitido que imposibilitaría obtener una solución real y aceptable por el algoritmo.
3. A diferencia de los algoritmos aglomerativos, obtendremos una o varias matrices de distancias entre los elementos de cada cluster, es decir, matrices independientes asociadas a los clusters activos existentes en el paso previo.
4. En este algoritmo, debemos seleccionar la mayor distancia global de las matrices, pues buscamos cual es el subcluster más heterogéneo de todos los posibles, es decir, incluyendo todos los elementos no divididos hasta el momento. Seleccionando éstos de los subclusters, obtenemos un nuevo nivel del árbol de descomposición o del *Dendograma*.
5. Repetiremos los pasos 2, 3 y 4 hasta obtener N clusters formados por un único elemento cada uno. Es decir, cuando no sea posible dividir ningún subcluster en otros más simples.

Aunque son técnicas muy útiles, supone un problema trabajar con muestras que incluyen más de 50 objetos. La interpretación de los datos se complica por perder claridad en los *dendogramas*, aunque es posible aplicarlo a tantos objetos como se desee.

En nuestro paquete incluiremos funciones basadas en ambos tipos, detallando en todo momento la ejecución del algoritmo hasta la solución final.

3.2.2 Clustering no jerárquico o repartitivo.

Por otro lado, esta técnica agrupa los objetos en los clústeres más homogéneos. Es necesario definir el número de clusters que compondrán la solución final, es decir, indicar en cuantos grupos se clasificarán los datos a analizar. A partir de una matriz de distancias que almacena las similitudes entre objetos, y entre objetos individuales y cada cluster, y de una matriz de pertenencia que asigna cada objeto al clúster más similar, agrupamos los distintos elementos en los clusters definidos inicialmente. Dependiendo de los criterios de cada algoritmo, las agrupaciones serán de un modo u otro, pero siempre aparecerán todos los elementos clasificados (excepto por decisiones del analista).

El algoritmo más importante de la clasificación repartitiva o no jerarquizada es *K-means*, muy utilizado en minería de datos.

3.2.3 Clustering jerárquico correlativo.

Esta clasificación se define para poder incluir el nuevo algoritmo descrito e implementado en este paquete. No podemos añadirlo en las secciones anteriores porque incluye características de ambos pero no describe un comportamiento similar a ninguno de los dos previos.

El algoritmo se basa en ordenar jerárquicamente los datos, es decir, a partir de las características numéricas de los objetos en estudio, calcular la proximidad a un objetivo o *target* con las mismas características que los datos iniciales. Calculará la distancia de cada dato al objetivo y ordenará los elementos atendiendo a esta distancia.

Al igual que en el resto de algoritmos, es posible decidir que tipo de distancia se aplicará en los cálculos (*distancia euclidea*, *de Manhattan*, *Octal*, *Camberra* o *de Chebishev*), es decir, como consideraremos el concepto de distancia entre cada objeto y el objetivo. Sin embargo, no existen clusters de más de un elemento, por lo que no será necesario indicar que tipo de similitud o proximidad se aplica en cada caso, la distancia será un valor directo, deducido del cálculo de las distancias previamente mencionadas.

El resultado final constará de una lista ordenada de elementos, así como las distancias de cada uno al objetivo en orden creciente. Este resultado será modificado por el analista, que agrupará los elementos así:

- Atendiendo al número de elementos que formarán cada grupo, es decir, creando clústeres de N elementos a partir de la lista resultado de elementos ordenados. Creará tantos clústeres como necesite hasta agrupar todos los componentes de la lista resultado.
- Atendiendo a la distancia entre los elementos y el objetivo. Dada una distancia d , se agruparán elementos que se encuentren a una distancia d del objetivo; el siguiente grupo contendrá todos los elementos situados a una distancia del objetivo entre d y $2d$. Repetirá este proceso hasta agrupar todos los elementos de la solución final.

Para aumentar las aplicaciones del algoritmo, se incluye la posibilidad de aplicar diferentes pesos a las características de los objetos, es decir, modificar la relevancia de cada característica en el cálculo final de la distancia. Esto permite aplicar una clasificación más precisa, incluyendo todas las características de los objetos en el estudio, pero priorizando las que tengan mayor influencia en la clasificación. Esta opción se aplicará en caso de que el usuario la especifique y utilizará valores introducidos por los analistas expertos en la materia, es decir, la relevancia de cada característica vendrá definida por un profesional del sector de los datos recogidos.

Por otro lado, se incluyen algunas herramientas de normalización de valores, es decir, distribución de los "*pesos*" en valores coherentes, evitando así

que los valores difieran en rangos que eliminen la utilidad del algoritmo.

En la sección técnica que describe la implementación de los paquetes, se explicará en mayor profundidad este algoritmo. Además, se incluyen funciones que describen el proceso seguido hasta obtener una clasificación final.

3.2.4 Distancia entre los datos.

[39] Hemos comentado previamente que las clasificaciones de los datos se realizan atendiendo a la distancia que los separa y al concepto de *proximidad*. En esta sección, definiremos el concepto de distancia entre dos puntos y explicaremos las distancias utilizadas en la implementación de los algoritmos incluidos en el paquete. En la clasificación clásica, las distancias en combinación con la proximidad que comentaremos en apartados posteriores implican algoritmos diferenciados, aunque en esta implementación se ha optado por incluir todos ellos en una misma función, siendo el usuario el que decidirá que características usar.

Las distancias aplicadas en este desarrollo se recogen a continuación, utilizando las variables numéricas de los datos para completar la matriz de distancias que utilizaremos:

- Distancia Euclideana: es la distancia entre dos puntos en un sistema de coordenadas cartesianas. Se deduce a partir del teorema de Pitágoras.
- Distancia Manhattan: define la distancia entre dos puntos como el sumatorio de todas las variaciones entre dos puntos contiguos.
- Distancia de Chebyshev (o de Shebyshov): se utiliza en matemáticas y define la distancia en un espacio vectorial. Considera la distancia entre dos puntos como la mayor diferencia en sus coordenadas. Aplicaremos esta distancia atendiendo a la mayor diferencia entre las dos características numéricas de los datos utilizados en nuestro análisis.
- Distancia de Canberra: Es una distancia entre dos puntos calculada sobre un espacio vectorial. Es una versión ponderada de la distancia de Manhattan definida previamente. Ha sido utilizada para estudios de enfermedades o para detectar problemas de seguridad informática.
- Distancia Octal: Esta distancia es, en realidad, una heurística aplicable en algoritmos de rutas, pero calcula la distancia entre dos puntos en un eje de coordenadas atendiendo a los valores numéricos de los datos, por lo que se ofrece al usuario para utilizarla en sus análisis.

Todas las distancias se crearon hace mucho tiempo, pero el fin de todas ellas es calcular la distancia entre dos puntos con valores numéricos, y es exactamente la necesidad que presentan los algoritmos de clasificación.

3.2.5 Proximidad o Similitud entre los clusters.

[36] Esta sección define el concepto de proximidad entre los clusters y describe las diferentes técnicas utilizadas para ello.

La proximidad o similitud entre dos clusters es el grado de semejanza que existe entre todos los elementos de los clusters que se estudian, es decir, si son parecidos o no. Numéricamente, el cálculo de la semejanza se asocia a la distancia entre los valores de ambos clusters. Podemos describir tres tipos de semejanza:

- Método del Linkage Simple, Enlace Simple o Vecino más próximo (MIN). Define la similitud entre dos clusters como la distancia más pequeña entre todos los elementos de un cluster y de otro, es decir, la distancia entre el componente de un clusters más cercano a un componente del otro cluster.
- Método del Linkage Completo, Enlace Completo o Vecino más alejado (MAX). Define la similitud entre dos clusters como la distancia más grande entre todos los elementos de un cluster y de otro, es decir, la distancia entre los componentes más alejados de cada cluster.
- Método del Promedio entre grupos. Define la similitud entre dos clusters como la distancia media entre todos los elementos de un cluster y de otro, es decir, la suma de distancias entre todos sus puntos entre el número de distancias utilizadas en el cálculo.

4 PAQUETE R

4.1 ¿Qué es un paquete?

Un paquete R es un directorio de archivos y otros directorios que incluyen código en R. Definimos como paquete un conjunto de archivos con código fuente preparados para la creación del paquete, o el propio paquete instalado en un entorno R utilizando CMD u otros editores apropiados como RStudio. Como se detallará en apartados posteriores, es posible crear paquetes binarios para algunos sistemas operativos (Windows, macOS, etc). De este manera, es posible descomprimir el contenido del paquete en cualquier localización sin necesidad de instalarlo desde las fuentes oficiales (opción de usar los paquetes sin conexión, por ejemplo).

Los paquetes de R incluyen funciones para el tratamiento de datos, es decir, incluyen scripts con código utilizable desde los entornos R apropiados.

Es necesario cargar los paquetes en los entornos antes de utilizarlos, por lo que sirven de mecanismo para cargar tanto las funciones que incluye como la documentación de cada una. Es posible añadir datos incluidos en los paquetes, de esta manera, al cargarlos, incluimos también los datos que almacena en nuestro espacio de trabajo.

Existen paquetes que incluyen código fuente en lenguajes permitidos diferentes a R; en los casos donde sea necesaria una compilación previa, debemos atender a las instrucciones indicadas en la página oficial de CRAN (<https://cran.r-project.org/doc/manuals/r-release/R-admin.html>) asociadas a cada sistema operativo.

Una biblioteca en R no es un paquete, y debemos diferenciarlos correctamente para no confundirlos en la documentación del sitio web oficial. En algunos casos se produce la confusión porque utilizamos la sentencia *library("nombrePaquete")* para cargar los paquetes ya instalados, pero hace referencia a que se encuentran instalados en una de ellas. Una biblioteca es un directorio de la máquina donde se instalan los paquetes como directorios genéricos, es decir, la biblioteca será la ruta donde encontraremos los directorios generales de cada paquete ya instalados, dentro de ellos se encuentra el contenido del propio paquete con el código, los datos y la documentación en un formato estandarizado.

Por otro lado, el término biblioteca es utilizado por el sistema operativo como bibliotecas compartidas, dinámicas o estáticas, también conocidas como bibliotecas de enlace dinámico (DLL). Estas bibliotecas compartidas son colecciones de código compiladas que en algunas plataformas reproducen el comportamiento de R. Ambos elementos se vinculan y se cargan en el proceso de instalación básico de R, pero en el sistema operativo macOS existe diferencia entre los objetos compartidos y las bibliotecas dinámicas (atribuye extensiones diferentes), por eso debemos distinguir entre ambos.

A pesar de las diferencias comentadas en el párrafo anterior, la instalación de paquetes debería ser automática en cualquier sistema operativo utilizando la opción desde CMD o el comando *install.packages("nombrePaquetes")* desde el entorno de edición (RStudio, por ejemplo). La instalación utilizará los archivos fuente necesarios para crear el paquete. Aunque no influye en nuestro desarrollo, la instalación de paquetes en el sistema operativo Linux presentará algunos problemas adicionales al instalar nuevos paquetes; es necesario instalar, tanto el paquete deseado como los que provoquen dependencias utilizando la consola y otorgando permisos de administrador.

La creación de paquetes, como es el caso del paquete desarrollado en este documento, consiste en añadir un paquete con código fuente a una biblioteca. Se selecciona un directorio de origen que incluye el contenido del paquete y se crea un archivo *tarball* (archivos con extensión *.tar.gz*) que se podrá distribuir a todos los usuarios, incluyendo su documentación a partir de archivos *.Rd* o de viñetas incluidas en el paquete. De este modo, es posible realizar una instalación de prueba para comprobar si el contenido es correcto, aportando portabilidad y consistencia antes de la publicación oficial del mismo.

Debemos aclarar que no es correcto compilar un paquete fuente como tal, es decir, la instalación del paquete fuente en R no será igual que si existen archivos en otros lenguajes como C o Fortran. En el caso de que existan archivos en otros lenguajes, la instalación requiere de la compilación de esos archivos dentro del propio paquete.

Sin embargo, si es posible hacer una compilación de código R, pero será a nivel de byte en un paquete. Para ello, se utilizan una serie de ventajas incluidas en el compilador de paquetes, habilitado por defecto en todos los paquetes de R con versiones superiores a la 3.5.0. Antes era posible realizar esta compilación también, pero debía habilitarse el permiso necesario del paquete. En conclusión, cuando nos refiramos a la compilación de un paquete, nos referiremos a la compilación de su código R a nivel de byte.

Otro concepto que se debe aclarar debido a actualizaciones recientes, es el de carga del paquete utilizando el comando *library(nombrePaquete)*. Desde que se incluyen los espacios de nombres (namespace). La carga del paquete consiste en cargar el espacio de nombres y después, adjuntar el resto del contenido del paquete para que sea visible en la ruta especificada. La biblioteca realiza ambos pasos consecutivos, pero es posible cargar únicamente el espacio de nombres sin vincular el contenido del paquete. Los detalles de esta última opción no se incluyen en este documento porque no aporta valor a nuestro paquete y requiere de un conocimiento muy avanzado de R.

La carga diferida (lazyload) de código y datos es parte de la instalación. Es una opción configurable, por parte del responsable del paquete, en el caso de los datos, pero en ningún caso lo será para el código. Este proceso de carga se caracteriza porque al cargar el espacio de nombre del paquete únicamente, se crean elementos ficticios en los objetos que aparecen. Cuando se ha producido la carga, se comprueban esos elementos ficticios y se cargan con su contenido real.

Los contenidos que cargamos con los paquetes provienen de bases de datos con estructuras similares, pero diferentes para el código y el contenido de los datos. El código se almacena en el directorio *R*, y los datos en el directorio *data*. Ambas están formadas por dos archivos (*.rdb* y *.rdx*):

- El archivo *.rdb* es una concatenación de objetos serializables.
- El archivo *.rdx* incluye un índice. Este índice se trata de un objeto R también serializado. Esta comprimido e incluye: variables, referencias y comprimidos. Las variables y las referencias son listas numéricas que incluyen las características de los objetos almacenados en el archivo *.rdb*. Los comprimidos en un vector de booleanos que indica si los objetos referenciados han sido comprimidos o no (actualmente casi todos los objetos utilizan esta opción).

Los objetos R incluyen una cabecera de pocos bytes que indican algunas características de los objetos y las condiciones en las que fueron comprimidos. Esta información es importante para el proceso de carga lenta o diferida, porque los objetos de gran tamaño producen errores en este tipo de cargas por incluir objetos demasiado pesados. A pesar de estas complicaciones, la carga de datos desde y hacia las bases de datos se realizan utilizando herramientas dedicadas a este fin, incluidas en los propios paquetes o en el entorno de R. Algunas cargas se guardan en memoria después de la primera carga, pero deben ser muy ligeras; así evitamos la latencia de otros dispositivos de almacenamiento.

4.1.1 Aportar nuevos paquetes.

Después de presentar el lenguaje R y todas sus funciones, argumentaremos la importancia de crear paquetes nuevos a CRAN. Hemos comentado que existen muchos paquetes aplicados a diferentes usos, que permiten tratar, cargar o mostrar los datos de manera variada. La razón principal de continuar creando paquetes propios de R sería que debemos añadir funcionalidades nuevas o que trabajen los datos con técnicas no existentes hasta ahora. Sin embargo, es posible crear contenido ya existente aportando elementos nuevos o modificando paquetes ya existentes centrándonos en otra funcionalidad, como la enseñanza, por ejemplo.

El motivo principal de añadir los paquetes propios a <https://cran.r-project.org> es que los propios usuarios de R contribuyen al crecimiento del lenguaje, y comparten con el resto el trabajo realizado.

Publicar un paquete en CRAN implica cumplir unos requisitos fijados por la organización. Debe incluir una documentación clara de cada función, que se pueda consultar por los usuarios del paquete si tienen dudas con el funcionamiento, el paso de parámetros, el objetivo final de cada función o ejemplos de las llamadas. La unificación de todos estos aspectos permite que todos los usuarios de R puedan acceder a las funciones y a su documentación de manera sencilla y uniforme al resto de paquetes.

Permite compartir con los usuarios del lenguaje R en cualquier parte del mundo la investigación y los progresos realizados utilizando unos estándares que faciliten el acceso, para que todos puedan utilizar las nuevas funciones añadidas. Además, que un paquete se publique en CRAN implica que ha superado los requisitos exigidos por la organización, y por tanto, se trata de código de calidad, que aporta un valor no incluido previamente.

Por último, aprovechando los requisitos impuestos por la organización de incluir el contacto de los autores y el compromiso de realizar un mantenimiento del paquete, se reciben críticas y comentarios sobre el mismo, permitiendo así detectar fallos o añadir nuevas funcionalidades.

4.2 Creación del paquete R

El siguiente paso es la creación del propio paquete. Por línea de comandos de R, existe la instrucción `package.skeleton()`, pero no se recomienda porque añade archivos adicionales que van a tener que mantenerse o eliminarse manualmente por el usuario, por lo que descartaremos esta opción.

Otra posibilidad será utilizar la interfaz gráfica: seleccionamos *File – NewProject*, y aparece la siguiente pantalla:

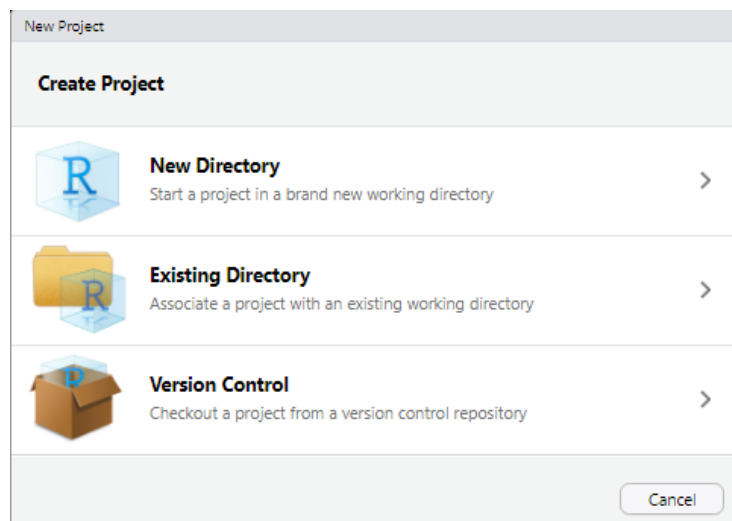


Figure 1: Crear paquete R desde RStudio.

Seleccionamos *New Directory*.

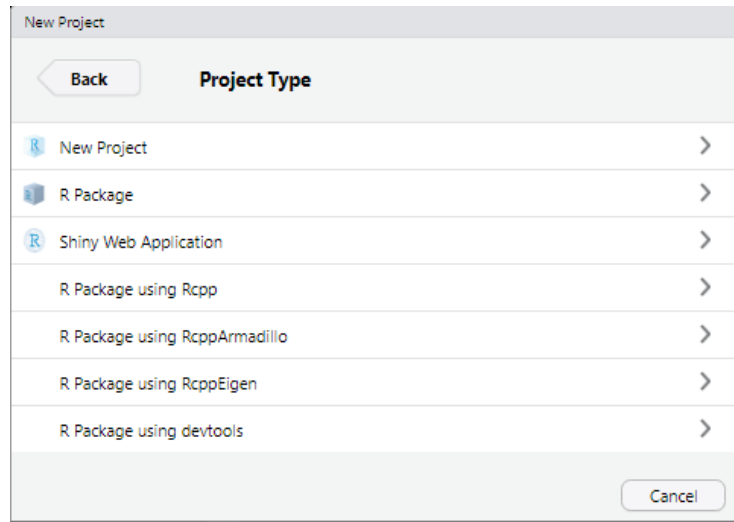


Figure 2: Seleccionamos tipo de proyecto.

Seleccionamos *R Package*.

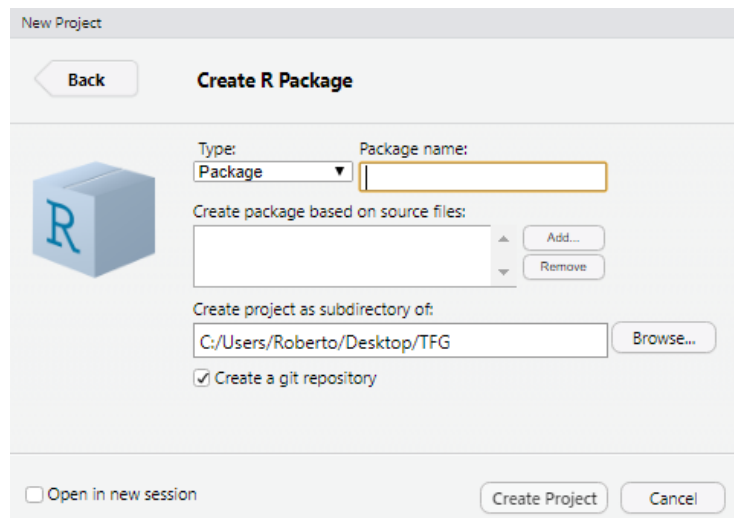


Figure 3: Paquete R

Introduciremos el nombre de nuestro paquete y donde se indica, comprobaremos que el directorio sea el adecuado, aunque ya lo hemos configurado en los pasos iniciales, y aceptamos crear el repositorio de git para nuestro paquete.

Es posible, como se indica en el campo *Create package based on source files* indicar archivos previamente manualmente compilados sobre los que crear el paquete, pero no será nuestro caso.

Este proceso sería equivalente a ejecutar en la consola de RStudio:

```
devtools::create("C:/Users/Roberto/Desktop/TFG/Prueba")
```

Creamos el paquete y comprobamos que se ha realizado correctamente. Para ello, observamos que en la ruta indicada aparece el paquete y está formado por una carpeta `R` que incluirá las funciones que forman el paquete (inicialmente incluye la función `hello.R` como ejemplo), un archivo `DESCRIPTION` que recoge los metadatos del paquete y que se comentará posteriormente más a fondo, y un archivo `NAMESPACE` que también comentaremos posteriormente [10].

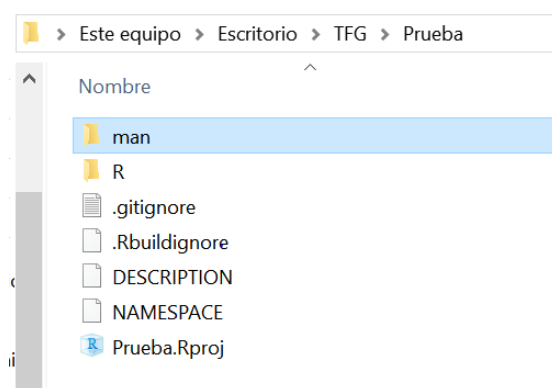


Figure 4: Estructura inicial del paquete.

Se incluye también un archivo `Prueba.Rproj` que utiliza RStudio internamente y que no influirá en la experiencia del usuario. Si creásemos el paquete desde otro entorno, no existiría.

Por último, existe un directorio `man` que incluirá la documentación sobre las distintas funciones del paquete, así como ejemplos y modos de uso.

4.3 Estructura del paquete R

4.3.1 Archivo `DESCRIPTION`.

[5] Este archivo contiene la información general del paquete (autor, versión, etc.). Cuando se crea un archivo con el nombre de `DESCRIPTION` es importante respetar mayúsculas y minúsculas. Este archivo almacena información básica sobre el paquete siguiendo el formato “Archivo Control Debian”, el cual consiste en crear campos con un nombre ASCII, seguido por dos puntos (:) y el valor del campo separado por un espacio entre los dos puntos y el valor. A continuación, esta descripción se muestra en la figura 1.

```

Package: 'Nombre del paquete'
Version: x.x.x
Date: AAAA-MM-DD
Title: 'Título corto del paquete'
Author: 'Autor o autores'
Maintainer: 'Nombre de la persona que mantiene el paquete' <correo@tecnico.com>
Depends: R(>= 3.1.0)
Description: 'Descripción larga del paquete'
License: GPL (>= 2)
URL: 'Dirección web donde se pueda consultar sobre el paquete'
Encoding: 'Tipo de codificación'
RoxygenNote: 5.0.1

```

Figure 5: Formato de un archivo DESCRIPTION básico.

Dentro de los campos obligatorios que se deben definir se encuentran:

- *Package*: nombre del paquete. Sólo debe contener caracteres (ASCII) tipo letras, números y puntos. Debe tener al menos dos caracteres y comenzar con una letra y no terminar nunca con un punto. Será breve y muy claro.
- *Version*: versión del paquete. Este valor es una secuencia de al menos 2 (normalmente 3) valores enteros no negativos separados por puntos o guión (-). En relación a la semántica para generar una versión, el primer número representa el valor de la versión mayor y significa cambios incompatibles del sistema; el segundo número representa la versión menor y representa funcionalidades agregadas compatibles con la versión actual del sistema, y el tercer valor es el parche, y significa los parches o correcciones añadidas de errores a la actual versión. Normalmente en la versión mayor su significado es:
 - 0 = versión alpha.
 - 1 = versión beta.
 - 2 = candidato a liberación.
 - 3 liberación final o estable.
- *License*: licencia para el paquete y es importante asignar un valor, porque sin él podrían existir problemas legales con el paquete. Recordemos que R está bajo la licencia Generic Public Licence (GPL), por lo que su uso y librerías son libres.
- *Description*: descripción completa de la funcionalidad del paquete. En esta sección se pueden utilizar varias sentencias, pero solamente un párrafo. Esta descripción debe ser entendible para cualquier usuario de R. No se debe iniciar la descripción con la frase “This package” o algún similar. Si es necesario aclarar el nombre del paquete se realizará en esta sección.

- *Title*: descripción corta del paquete y tiene como longitud máxima 65 caracteres. Se pueden utilizar letras en mayúsculas en las palabras importantes y no repetir el nombre del paquete.
- *Author*: describe quién escribió o contribuyó de una forma relevante en la creación del paquete. Este es un campo en texto plano con valores permitidos ASCII en forma "Nombre Apellidos". Si existen varios autores entonces serán separados por comas (.). En caso de los autores con dos apellidos (materno y paterno) es válido utilizar el guión (-) para unificarlo en un sólo apellido.
- *Maintainer*: nombre de la persona que está a cargo del mantenimiento del paquete. En este campo sólo se permite un nombre siguiendo el formato del autor y seguido de la dirección de correo electrónico, en corchetes angulares (< >). Fuera de estos campos, los demás son opcionales, aunque muchos paquetes hacen uso de los campos 'Imports', 'Exports' y 'Depends', ya que es muy común hacer uso de otros paquetes para que el nuestro pueda funcionar.

Los campos *Author* y *Maintainer* se sustituyen por un campo '*Authors@R*' que incluya esa misma información. De esta manera, sustituimos la información de los campos previos dedicados a la interpretación humana por una descripción más técnica, que será interpretada por código R para conocer la información de la autoría de los paquetes. Sin embargo, en los campos que sustituye se debe incluir un texto que lo puedan leer humanos. El resultado es la creación de un objeto de la clase *person*, debido a la llamada de una persona o a una serie de llamadas. Finalmente se obtiene un vector de nombres como se muestra en la figura anterior, donde se asocia a cada nombre un rol particular que detallamos a continuación:

- *aut* (autor): para autores completos.
- *cre* (creador): para el mantenimiento del propio paquete.
- *ctb* (colaborador): para otros contribuyentes.
- *cph*: propietario de los derechos de autor u otros participantes.

Por otra parte, los campos 'Author' y 'Maintainer' se generan automáticamente desde allí si es necesario al construir o instalar. Los demás campos que pueden aparecer en el archivo *DESCRIPTION* son los siguientes:

- *Copyright*: campo opcional. Indica si los titulares de los derechos de autor no son los autores reales del paquete. Si es necesario, puede referirse a un archivo de instalación como *inst/COPYRIGHTS*.

- *Date*: campo opcional que indica la fecha de lanzamiento de la versión actual del paquete. Se recomienda utilizar el formato “aaaa-mm-dd” conforme es recogido en el modelo ISO 8601.
- *Depends*: campo que incluye un vector (lista separada por comas) de nombres de otros paquetes de los que depende este. Existirán funcionalidades dentro del paquete que requieren de otros paquetes instalados previamente para el correcto funcionamiento, es decir, las dependencias ente ellos. En nuestro caso, esto ocurre con el paquete *magick* supone una dependencia para las funciones de documentación que muestran imágenes. Estos paquetes se adjuntarán antes del paquete actual, cuando la biblioteca sea llamada o sea requerida. Cada nombre de paquete puede estar seguido por un comentario entre paréntesis que especifique el requisito de la versión. Este comentario debe contener un operador de comparación, espacios en blanco y un numero de versión válida.

Este campo debe especificar, si existe, la versión mínima de R que se requiere para el correcto funcionamiento del paquete. La dependencia de R debe ser a una versión lo más estable (último dígito de la versión igual a 0) y actualizada posible; cuanto más estable sea, menos problemas ofrecerá al incluir el paquete, y la actualidad de la versión facilita que se encuentre en un mantenimiento activo por parte de los responsables. Las instalaciones de R INSTALL verifican si la versión de R usada es lo suficientemente reciente para el paquete que sea instalado, y se adjuntará la lista de paquetes que se especifica antes del paquete actual. Este campo debería utilizarse en la actualidad con poca frecuencia, solo para aquellos paquetes destinados a ponerse en la ruta de búsqueda para que sus instalaciones estén disponibles para el usuario final.

- *Imports*: incluye todos los paquetes que debemos importar, es decir, que se incluyen en el archivo *NAMESPACE*. Sin embargo, no incluirá los paquetes que exijan ser adjuntados. Si el paquete se incluye en el campo *Depends*, no es necesario incluirlo aquí. Debe especificar las versiones adecuadas para cada paquete y se verificarán durante la carga del archivo *namespace*.
- *Suggests*: campo que enumera los paquetes que no son necesarios, es decir, recomendaciones que facilitarían el correcto funcionamiento del paquete o que añadirían funcionalidad adicional, pero que si no aparecen no impiden la instalación adecuada. Utiliza la sintaxis del campo *Depends*.
- *Enhances*: enumera los paquetes modificados por nuestro paquete; es decir, aquellos que utilizan contenido desarrollado en el nuestro, tanto objetos como tratamientos sobre ellos. En este campo, los requisitos de versión se pueden especificar, pero actualmente no se utilizan.

- *LinkingTo*: campo que se utiliza cuando un paquete desea hacer uso de los archivos de encabezado situados en otros paquetes. Se debe crear una lista separada por comas y puede contener un requisito de versión que se verifica durante la instalación.
- *Additional repositories*: lista, separada por comas, de URL de repositorios donde se encuentran los paquetes nombrados en los otros campos. La verificación de R CMD *check* comprueba que los paquetes se pueden encontrar y sean accesibles desde cualquier plataforma, al menos como paquetes fuente.
- *SystemRequirements*: campo que incluye si existen dependencias externas al propio entorno de R. Si existen, se debe añadir un archivo Readme que las describa, pues es común que dificulte la instalación completa de paquete.
- *BugReports*: campo que contiene una URL de contacto. A ella se envían los informes de errores del paquete. Esta opción solo está disponible en versiones superiores a R 3.4.0.
- *Priority*: para paquetes base y contenidos recomendados en la distribución fuente de R o disponible de CRAN y que se recomienda que se incluyan en cada distribución binaria de R.
- *Collate*: campo que regular el orden de los archivos con código R durante el procesamiento previo a la instalación del paquete.
- *LazyData*: este campo controla si los conjuntos de datos y de código R usan la carga lenta o diferida. Actualmente es una propiedad que no se utiliza demasiado.
- *KeepSource*: campo booleano que regula si el código del paquete se obtiene utilizando *keep.resource*.
- *ByteCompile*: campo booleano que controla si el código del paquete debe compilarse en bytes durante la instalación. Como se ha comentado previamente, esto ocurre cuando debemos compilar el paquete con código fuente en R, donde la compilación debe ser a nivel de byte. Además, su valor predeterminado es *byte-compile*.
- *Biarch*: campo que permite seleccionar la opción *INSTALL - force-biarch* para el paquete.

- *BuildVignettes*: campo booleano que define si deseamos construir o comprobar las viñetas existentes durante la instalación por CMD.
- *VignetteBuilder*: este campo aparece en los paquetes con herramientas específicas para la construcción de viñetas, tanto las viñetas del propio paquete como los incluidos en *Depends*, *Imports* o *Suggests*.
- *Encoding*: campo que especifica la codificación del archivo DESCRIPTION cuando incluye caracteres que no son ASCII. Además, se indica la codificación de otros archivos, tanto *NAMESPACE* como archivos con extensiones .R y .Rd. Que existan varias codificaciones dentro del mismo paquete, reduce la probabilidad y exige construcciones particulares para cada codificación.
- *NeedsCompilation*: campo de comportamiento booleano con valores "si" y "no" que indica si el paquete incluye código que debemos compilar. Lo utiliza la instrucción *install.packages* en las versiones de R superiores a la 2.15.2 en plataformas donde los paquetes binarios son el formato más común.
- *OS type*: campo que especifica los sistemas operativos a los que está destinado el paquete.
- *Type*: especifica el tipo de paquete. Por defecto, todos los paquetes son "package", pero existen otros tipos con significados muy variados.
- *Language*: campo lista que incluye los lenguajes distintos al inglés en los que puede aparecer la documentación del paquete. Es importante que los lenguajes aparezcan como los define actualmente el RFC 5646.
- *RdMacros*: lista de paquetes separados por comas de la cual el paquete actual importa definiciones de macro Rd. Estos paquetes deben estar nombrados además en los campos *Depends*, *Imports* o *Suggests*.
- *Built* o *Packaged*: estos campos no deben aparecer, ya que son agregados por herramientas de gestión de paquetes.

Aparte de todos estos campos comentados, todos los repositorios de contenido R pueden incluir campos propios si desean configurar otros aspectos no recogidos.

4.3.2 Archivo *NAMESPACE*.

[7] El objetivo de este archivo es incluir los nombres de las funciones que estarán disponibles para los usuarios cuando carguen el paquete. También tiene como finalidad incluir las funciones de otros paquetes que se podrán usar en la implementación sin que sea necesario utilizar el operador `::`.

El archivo *NAMESPACE* incluye un listado de exports/imports. Por un lado, los exports son aquellas funciones o variables que están definidas en los paquetes o importadas de otros que deben poder usarse por parte del usuario dentro del paquete. Por otro lado, los imports se utilizan cuando el código de las funciones depende de funciones de otros paquetes. No se usan con frecuencia debido a que la buena práctica al programar paquetes es escribir una función escribiendo el nombre del paquete, seguido de `::` y, a continuación, el nombre de la función, pasando los parámetros que necesite entre paréntesis. De esta forma se evitan problemas generados cuando los nombres en las funciones se duplican.

El contenido del archivo es parecido al código R, aunque no se procesa como código R.

Por otra parte, en relación a los espacios de nombres, éstos se fijan una vez que se cargan, lo cual significa que las importaciones y exportaciones no pueden cambiarse, y que los enlaces de variables internas no se pueden modificar tampoco. Además, el espacio de nombres controla la estrategia de búsqueda para las variables usadas por funciones en el paquete. Si no es encontrada localmente, R busca en primer lugar el espacio de nombres del paquete, después las importaciones, seguidamente el espacio de nombres base y finalmente la ruta de búsqueda normal.

4.3.3 Directorio *R*.

Este archivo contiene los archivos de código R, y solo los que tengan extensión *.R*. Estos archivos de código deben empezar con una letra o un dígito ASCII (en mayúsculas o minúsculas), y debe tener una de las siguientes extensiones: *.R*, *.S*, *.q*, *.r* o *.s*. De todas ellas, la extensión más utilizada es la *.R*. Los objetos R deben crearse mediante asignaciones, para que así sea posible leer en dichos archivos utilizando `source("nombreArchivo.R")`, por lo que no debe haber conexión entre el nombre del archivo y los objetos R creados. Los archivos de código R solo deben asignar directamente objetos R. Además, si se necesitan cálculos para crear estos objetos, estos pueden usar el código *earlier* en el paquete, más las funciones en los paquetes de *Depends*, siempre que los objetos creados no dependan de esos paquetes excepto a través de las importaciones del espacio de nombres. Además de las extensiones de archivos mencionadas previamente, también se permiten los archivos que terminan en *.in*. Esto se puede hacer para permitir que un script de configuración genere los archivos adecuados.

Por otro lado, aunque en los archivos de código solo se deben usar caracteres ASCII, se aceptan otros caracteres en los comentarios, pero es posible que no sean legibles en una configuración particular. Aquellos nombres de ob-

jetos que contienen caracteres que no son ASCII normalmente fallan cuando el paquete está instalado.

4.3.4 Directorio *man*.

[2] Uno de los aspectos más relevantes de los paquetes es la documentación. Es útil para los usuarios y para los desarrolladores que extienden o mantienen el paquete en el tiempo. La documentación de objetos es un tipo de documentación de referencia.

R facilita una forma estándar de documentar objetos en un paquete escribiendo archivos con extensión *.Rd* en el directorio *man*. Además, los archivos utilizan una sintaxis personalizada basada en LaTeX. Se puede representar en HTML, texto sin formato y PDF, para que se pueda visualizar.

En el caso de este paquete, se ha utilizado roxygen2, que se encarga de convertir los comentarios incluidos en los archivos de código real, es decir, las funciones incluidas en el paquete en comentarios formateados en archivos *.Rd*. La finalidad es conseguir que la documentación del código sea lo más sencilla posible. Las ventajas que se pueden encontrar al utilizar Roxygen2 en lugar de la escritura de archivos *.Rd* a mano son las siguientes:

- El código y la documentación aparecen unidos y relacionados para que al cambiar el código se recuerde fácilmente que, además, debe actualizar la documentación.
- Roxygen2 comprueba de manera dinámica los objetos que va documentando, por lo que puede omitir textos estándar.
- Sintetiza las diferencias en la documentación de los distintos objetos, de esta manera no necesita aprender muchos detalles.

Es importante conocer que roxygen2 también puede administrar algunos aspectos de *NAMESPACE* o del campo *Collate* del archivo DESCRIPTION.

4.3.5 Directorio *src*

[3] En este subdirectorio se encuentran las fuentes del código compilado. R es un lenguaje de alto nivel, pero a causa a su expresividad, es más lento. Debido a esta razón, la incorporación de un lenguaje compilado de un nivel bajo como C y C++ es un buen complemento para el código R. Aunque frecuentemente C y C++ necesitan un pensamiento más preciso y más líneas de código para resolver el mismo problema, pueden ser órdenes de magnitud más veloces que R.

Por otro lado, las cabeceras del código compilado están en *src*, de forma

opcional un archivo *Makevars* o *Makefile*. Este directorio se utiliza por algunos paquetes para otros fines diferentes a la creación de un objeto compartido, por eso, estos paquetes deben tener los archivos *src/Makefile* y *src/Makefile.win*.

4.3.6 Directorio *data*

[4] Es muy útil incluir datos en un paquete, tanto si está dirigido a una audiencia amplia o a una audiencia más concreta. Hay tres formas principales de incluir datos en un paquete, teniendo en cuenta cual será su objetivo y a que tipo de usuario va a ir dirigido:

- Si el autor del paquete desea almacenar datos binarios y ponerlos a disposición de la persona que vaya a utilizarlo, debe introducirlos en el directorio *.data/*. Este sería el mejor lugar para almacenar los conjuntos de datos.
- Si el autor del paquete desea almacenar datos analizados, pero no ponerlos a disposición del usuario, se deben introducir en el directorio *R/sysdata.rda*. Este es el mejor sitio para colocar los datos que necesitan las funciones.
- Si se necesita almacenar datos sin procesar, hay que introducirlos en el directorio *inst/extdata*.

Por otro lado, existe una alternativa a las tres opciones explicadas previamente; se puede incluir en la fuente del paquete, creándolo a mano o usando la función *dput()* para serializar un conjunto de datos existente en código R.

Referido a cada posible ubicación, la más común para los datos de paquetes es *data/*. Cada archivo de este directorio tiene que ser un archivo *.Rdata* creado con la función *save()* con un solo objeto. Es muy importante incluir el código utilizado para la serialización en la versión fuente del paquete. De esta manera, será más fácil la actualización de la versión de los datos.

Los objetos en *data* siempre se exportan eficazmente, por lo que deben estar documentados. No se documentan los datos directamente, sino que se documenta el nombre del conjunto de datos y R lo almacena automáticamente. Para documentar conjuntos de datos hay dos etiquetas fundamentales (aunque existen otras): *@format* y *@source*.

Por otra parte, cuando las funciones necesiten tablas, es recomendable guardarlos en formato *R/sysdata.rda* para que no estén disponibles para los usuarios del paquete. Estos objetos no deberían exportarse, asique no es necesaria su documentación. También es importante mencionar que los datos del paquete no deben ser más grandes que 1 MB. Esto será más sencillo de llevar a cabo si los datos se sitúan en su propio paquete y no se actualizan frecuentemente.

4.3.7 Directorio *inst*

El contenido de *inst* se copia en el directorio de instalación. Este directorio permite agregar archivos y directorios arbitrarios del nivel superior. Además, nunca debe utilizar un subdirectorio con un nombre igual que un directorio que ya exista.

Los archivos que se agregan a este directorio con más frecuencia son los siguientes:

- *inst/CITATION*: para que se utilice con la función de cita.
- *inst/docs*: esta estructura de carpetas se crea durante la construcción del tarball final. Incluye el contenido de las viñetas ya creadas.
- *inst/extdata*: son datos externos que se añaden a ejemplos y viñetas.
- *NEWS.Rd*: para que lo utilice la función de noticias.

4.3.8 Directorio *vignettes*

Una viñeta es una guía completa del paquete del usuario, ésta explica el problema para el que está diseñado el paquete; para resolver y posteriormente muestra al usuario cómo resolverlo. Cada viñeta proporciona tres cosas: el archivo fuente original, una página HTML o PDF que se pueda leer y un archivo de código R. Una viñeta en concreto se puede leer con *vignette(nombreViñeta)* y ver su código con *edit(vignette("nombreViñeta"))*. Además, una viñeta divide las distintas funciones en categorías útiles y demuestra la coordinación de varias de ellas para resolver problemas.

R extrae el código de las viñetas (que son, como se ha comentado anteriormente, documentos en PDF o HTML). Esta función la realizan los motores de tratamiento de viñetas. Las funciones principales que utilizan estos motores son *tangle* y *Sweave*. La ubicación estándar para los documentos es el subdirectorio *inst/doc* de un paquete fuente.

En los inicios de la documentación en R, la forma única de crear una viñeta era con *Sweave*, que solo funcionaba con LaTeX. En la actualidad cualquier paquete puede proporcionar un motor de viñetas.

4.3.9 Directorio *test*

R permite incluir en el paquete pruebas, tanto unitarias como genéricas. Las pruebas se encargan de asegurar que el código del paquete haga lo que se ha determinado por el usuario. Uno de los errores más comunes entre personas dedicadas a programar R es no automatizar las pruebas. Es de gran relevancia

cambiar de utilizar pruebas informales *ad hoc* para comenzar a emplear pruebas formales automatizadas o pruebas unitarias. Las ventajas que se pueden encontrar al utilizar pruebas automatizadas son las siguientes:

- Menos errores. Al describir el comportamiento del código de un paquete en dos lugares, en el código y en las pruebas, se comparan ambos y, de esta manera, no se repetirán los errores.
- Reinicios más sencillos. Las pruebas facilitarán que el usuario continúe trabajo en el punto donde lo había dejado previamente.
- Mejor estructura de código. El código estará mejor diseñado y, de esta manera, las funciones se trabajarán, se comprenderán y se probarán mejor.
- Código robusto. Se pueden realizar grandes cambios en el paquete sin la preocupación de cometer algún error de forma accidental.

Es importante conocer que existen dos estilos de prueba, *unitizer* y *testthat*. Por un lado, *unitizer* necesita que se revisen los resultados de prueba y que se confirme que son los esperados. Por otro lado, *testthat* requiere que se afirme cuáles son las salidas de prueba previamente. Las ventajas que ofrece el primero de ellos son la sencillez de escritura de las pruebas, no exigen un manejo complejo de las condiciones porque se capturan de forma automática, se pueden revisar las pruebas fallidas en un entorno interactivo y cuando la función cambia legítimamente es sencillo actualizar las pruebas.

Los beneficios al trabajar con *testthat* son la autodocumentación de las pruebas, liberando al desarrollador de esta tarea, y evitando el error humano (que será mucho más probable que el de la máquina). Esto permite que las pruebas estén actualizadas, no sea necesario modificarlas cuando se realizan cambios sobre las funciones reales.

4.3.10 Directorio *demo*

Este recurso se podría considerar un tipo de prueba más. Nos lo ofrece R para aprender a utilizar sus posibilidades. Las demostraciones son ejemplos prácticos en los que se muestran los resultados de aplicar unas funciones determinadas. Este subdirectorio se utiliza para scripts R que prueban funciones del paquete. Las comprobaciones no se realizan de forma automática, para hacer pruebas se tiene que utilizar el código en el directorio *tests*. Es importante saber que los archivos de script deben comenzar con una letra mayúscula o minúscula y tener la extensión .R o .r. Una demostración no tiene una codificación concreta, así que debe ser un archivo ASCII.

Muchos paquetes incorporan demostraciones elaboradas de su funcionalidad, especialmente los que corresponden a gráficos.

4.4 Viñetas.

Una viñeta es un documento PDF o HTML asociada a un paquete. Se obtiene a partir de archivos `.Rmd` que están formados por texto plano y código en R (este código se incluye en un chunk, “`r`”). Los entornos de R tienen motores preparados para distinguir el código y el texto y permiten ejecutarlo. Estas viñetas se recogen en archivos `.Rmd`, pero cuando se crean, aparecen en el subdirectorio `doc`, que al construirse, se situará dentro del subdirectorio `inst`.

Pueden existir diferentes viñetas para un único paquete, en este caso, se creará un índice con todas ellas, pero la estructura y el modo de creación será el mismo en ambos casos.

Las viñetas, cuando son creadas, lo hacen en tres archivos en el subdirectorio `doc` como se ha comentado previamente. Creará un archivo `.html`, otro archivo `.R` y el archivo `.Rmd` de origen que también aparece en el subdirectorio `vignettes`. De la misma manera, al construir el paquete completo, las viñetas son revisadas y creadas de nuevo para ser distribuidas con el paquete.

4.5 Construcción de paquetes.

[31] Después de detallar el procedimiento a seguir para construir el paquete y toda la documentación necesaria, se describe a continuación el último paso, la construcción final del paquete. Para poder realizarla correctamente, debemos comprobar manualmente que el paquete se pueda instalar en un entorno R. Después, debemos revisar el paquete con el comando de R `check()`, este comando permite diferentes parámetros para revisar del mismo modo el manual de usuario que incluye o si se desea que la revisión sea bajo restricciones de CRAN. Se comprueban todos los archivos y subdirectorios del paquete, que no exista ninguno que no deba estar en un lugar determinado, así como todos los ejemplos incluidos en la documentación de las funciones, etc. Con este comando, se ofrecen tres tipos de avisos, los errores que no permiten crear el paquete correctamente, los warnings que indican que algo no conflictivo pero incorrecto se ha producido y las notas que realizan observaciones sobre el paquete.

Después de revisar el paquete, si la respuesta ha sido correcta, podemos construirlo por completo. Para ello utilizamos el comando de R `build()`. De esta manera, obtenemos un archivo `.tar.gz` conocido como *tarball* listo para la distribución y publicación del paquete.

Debemos aclarar que los paquetes pueden distribuirse en archivos `.tar.gz` (*tarballs*) o en formato binario. Los *tarballs* son código fuente, se pueden instalar en cualquier plataforma con las herramientas básicas y es el más utilizado en sistemas Unix. Sin embargo, el formato binario es específico para cada plataforma, por lo que la instalación de cada uno se complica. Los paquetes binarios están formados por versiones comprimidas de paquetes ya instalados en otras máquinas.

5 DESCRIPCIÓN EXPERIMENTAL

A continuación se detallan todos los aspectos teóricos comentados anteriormente, pero aplicados a nuestro paquete. Se indican los pasos seguidos para construir el paquete desde cero, el contenido de cada subdirectorio y de cada archivo, así como las funciones incluidas para la funcionalidad.

5.1 Software necesario

Como se ha descrito en el marco anterior, el lenguaje utilizado para la implementación del paquete será R. Hemos comentado que está dedicado exclusivamente al tratamiento masivo de datos, y existe una comunidad mundial que colabora en el desarrollo del lenguaje añadiendo sus investigaciones y publicando nuevos contenidos.

Para el desarrollo del paquete en sí, utilizaremos RStudio, un entorno de desarrollo dedicado a R y muy superior a otros entornos con los mismos objetivos (ESS, por ejemplo). Utilizaremos el paquete *devtools* [24], elaborado por Hadley Wickham, que facilita en gran medida las tareas para obtener el paquete completo. Este paquete protege de algunos errores que tienen lugar durante la creación, por eso simplifica mucho el desarrollo del paquete. El procedimiento que vamos a seguir se aplica en un entorno Windows; es posible que en otros sistemas operativos los requisitos técnicos difieran.

Es posible crear el paquete desde cero utilizando otras herramientas, pero la abstracción sobre detalles de bajo nivel que ofrecen conjuntamente RStudio y el paquete *devtools* justifica nuestra elección. Podríamos crear el paquete incluso desde los comandos del cmd de Windows, pero exige la instalación de diferentes herramientas que no serán necesarias a partir del entorno de RStudio. Si un posible usuario futuro de este manual no desea utilizar las herramientas indicadas podría obtener toda la información en <https://cran.r-project.org/doc/manuals/r-release/R-exts.html#Creating-R-packages>.

Para la documentación del paquete utilizaremos LaTeX, por lo que debemos añadirlo a nuestro equipo. RStudio reconoce este sistema de estilos de texto como uno de los más útiles, y por ello, incluye facilidades para utilizarlo a partir del entorno de RStudio.

Por otra parte, es posible obtener una consola propia de R en <https://cran.r-project.org/bin/windows/base/> que nos permita lanzar algunas funciones y crear el paquete de ese modo (en caso de elegir esta opción, se recomienda la versión 3.6.1, de mayo de 2019). Es una sustitución aceptada para no añadir compiladores externos al cmd de la propia máquina. En primer lugar, será necesario instalar Rtools. Debemos seleccionar la versión más actualizada, pues será la que más mantenimiento obtenga, no se quede obsoleta y este aceptada por otras herramientas en sus versiones más actualizadas. Del mismo modo, instalaremos la herramienta de LaTeX, descargada desde <http://www.miktex.org>.

Debemos comprobar que todas estas nuevas herramientas se han instalado correctamente, en caso de no ser así, debemos investigar la solución a los problemas; generalmente, las variables de entorno no estarán correctamente configuradas. Si desde la instalación de Rtools autorizamos a la modificación de las variables de entorno, las variables se actualizarán automáticamente [26].

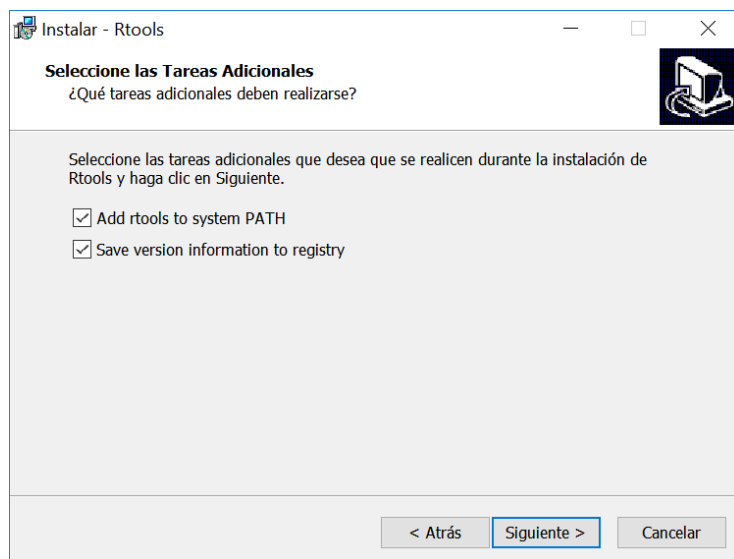


Figure 6: Autorizar la modificación automática de las variables de entorno.

A partir de RStudio, no es necesario realizar estas instalaciones. A continuación, crearemos el paquete a través de su interfaz gráfica, y aprovecharemos las facilidades que nos da para instalar automáticamente muchas de estas herramientas. Es importante disponer de una versión lo más actualizada posible de RStudio. Si elegimos esta opción, RStudio preguntará al usuario si desea añadir las herramientas necesarias.

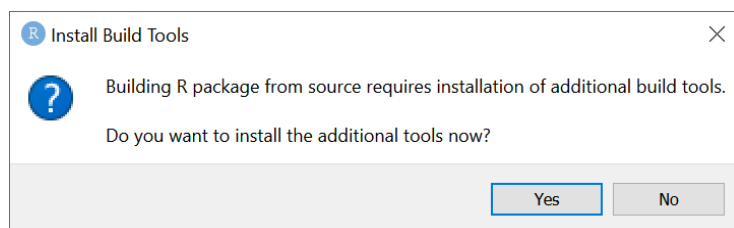


Figure 7: Consulta de RStudio para instalar herramientas automáticamente.

Desde la interfaz de RStudio, crearemos el paquete en el directorio de trabajo que consideremos oportuno, ahí aparecerá la estructura del paquete, con todos los elementos necesarios que se comentarán en apartados posteriores.

Para asegurarnos de cuál es nuestro directorio actual, ejecutamos `getwd()`, si no es el deseado, lo actualizamos con `setwd("nuestraRuta")`.

El primer paso será instalar y cargar el paquete `devtools` en el entorno. Ejecutando los siguientes comandos:

- `install.packages(devtools)`
- `library(devtools)`

Observamos que se añaden algunas dependencias necesarias, evitándose al usuario; de ahí la facilidad de utilizar RStudio.

A continuación, el comando `has_devel()` nos indica si todo se ha instalado correctamente:

```
> install.packages("devtools")
Installing package into 'C:/Users/Roberto/Documents/R/win-library/3.6'
(as 'lib' is unspecified)
also installing the dependency 'remotes'

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.6/remotes_2.2.0.zip'
Content type 'application/zip' length 388823 bytes (379 KB)
downloaded 379 KB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.6/devtools_2.3.1.zip'
Content type 'application/zip' length 341058 bytes (333 KB)
downloaded 333 KB

package 'remotes' successfully unpacked and MD5 sums checked
package 'devtools' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
  C:\Users\Roberto\AppData\Local\Temp\Rtmp6JaK2p\downloaded_packages
> library("devtools")
Loading required package: usethis
Warning messages:
1: package 'devtools' was built under R version 3.6.3
2: package 'usethis' was built under R version 3.6.3
> has_devel()
Your system is ready to build packages!
>
```

Figure 8: Confirmación de la instalación correcta de `devtools`

Por otro lado, como se comentará en secciones posteriores, utilizaremos el paquete `Roxygen2` para facilitar la documentación de las funciones y del paquete en general. Por eso, debemos instalarlo y cargarlo como se muestra en la imagen.

```
> install.packages("roxygen2")
Installing package into 'C:/Users/Roberto/Documents/R/win-library/3.6'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.6/roxygen2_7.1.1.zip'
Content type 'application/zip' length 1176661 bytes (1.1 MB)
downloaded 1.1 MB

package 'roxygen2' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
  C:\Users\Roberto\AppData\Local\Temp\Rtmp6JaK2p\downloaded_packages
> library(roxygen2)
Warning message:
package 'roxygen2' was built under R version 3.6.3
>
```

Figure 9: Instalación y carga correcta del paquete `roxygen2`

Ahora tenemos todos los paquetes auxiliares necesarios para crear el nuestro.

5.2 Creación paquete.

[30] Siguiendo los pasos descritos en la parte teórica de este documento, creamos el paquete utilizando la interfaz de RStudio:

- Seleccionamos *File*.
- *New Project*.
- *New Directory*.
- *R Package*.
- Introducimos el nombre del paquete, el directorio donde se creará, seleccionamos la opción de crear un repositorio de git y confirmamos con *Create project*.

RStudio nos abrirá el espacio de trabajo del nuevo paquete automáticamente si lo hemos seleccionado en el paso anterior. La siguiente imagen muestra la estructura básica del paquete recién creado. [33]

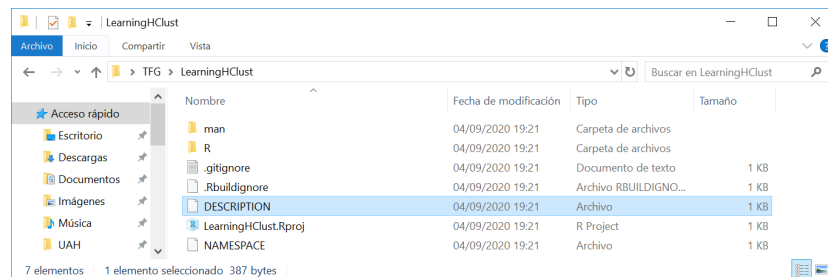


Figure 10: Estructura inicial del paquete creado con RStudio.

Incluye el directorio *R* donde se almacenarán los archivos *.R* con las funciones del paquete, el directorio *man* con archivos *.Rd* que incluirán la documentación de las funciones del directorio anterior. Como ejemplo, se incluye la función *hello* y su correspondiente documentación situadas en los directorios adecuados.

Se incluye el archivo *LearnClust.Rproj* que será un archivo de configuración propio de RStudio, no hay que modificarlo manualmente.

Por último, aparecen los archivos *NAMESPACE* y *DESCRIPTION*. Tienen una estructura inicial que debemos completar según avancemos en el desarrollo del paquete. Inicialmente su aspecto es el siguiente:

```
1 Package: LearningHClust
2 Type: Package
3 Title: What the Package Does (Title Case)
4 Version: 0.1.0
5 Author: Who wrote it
6 Maintainer: The package maintainer <yourself@somewhere.net>
7 Description: More about what it does (maybe more than one line)
8   Use four spaces when indenting paragraphs within the Description.
9 License: What license is it under?
10 Encoding: UTF-8
11 LazyData: true
12 |
```

Figure 11: Estructura inicial del archivo DESCRIPTION

Se incluyen campos obligatorios que debemos rellenar acorde a las restricciones impuestas por CRAN para poder publicarlo después (*Author*, *License*). Existen otros campos que decidiremos arbitrariamente que valor tendrán (*Version*, *Title*). Se incluyen también valores por defecto en algunos campos, que pueden modificarse también en función de las necesidades, pero habitualmente son la opción recomendable (*LazyData*, *Encoding*).

De esta manera, completamos los campos del archivo así:

```
1 Package: LearningHClust
2 Type: Package
3 Date: 2020-09-04
4 Title: Learning Hierarchical Clustering Algorithms
5 Version: 1.0
6 Authors: c(
7   person("Roberto", "Alcantara", role = c("aut", "cre"),
8     email = "roberto.alcantara@uah.es"),
9   person("Juan Jose Cuadrado", role = c("aut"),
10     email = "jjc@uah.es"),
11   person("Universidad de Alcalá de Henares", role = c("aut"))
12 )
13 Author: Roberto Alcantara [aut, cre],
14   Juan Jose Cuadrado [aut],
15   Universidad de Alcalá de Henares [aut]
16 Maintainer: Roberto Alcantara <roberto.alcantara@uah.es>
17 Description: LearningHClust includes classical hierarchical clustering algorithms, agglomerative and divisive clustering. Algorithms are implemented as a theoretical way, step by step.
18   It includes some detailed functions that explain each step. Every function allows options to get different results using different techniques.
19   The package explains non expert users how hierarchical clustering algorithms work.
20 License: Unlimited
21 Encoding: UTF-8
22 LazyData: true
23
```

Figure 12: Campos completados del archivo DESCRIPTION

Se detallan los autores del paquete asignando roles a cada uno de ellos, la persona encargada de mantener el paquete con el correo de contacto, el título del paquete y la versión inicial. Se incluye una pequeña descripción del paquete y se indica el tipo de licencia bajo el que estará recogido su uso.

El archivo *NAMESPACE* inicial tiene este aspecto:

```
1 exportPattern("^[[[:alpha:]]+"]
2 |
```

Figure 13: Estructura inicial del paquete NAMESPACE

Este archivo, incluye la sentencia `exportPattern("[[:alpha:]]+")` que permite hacer visibles todas las funciones contenidas en el paquete. Este archivo deberíamos ir complementandolo progresivamente en paralelo a las funciones que se incluirán en el subdirectorio *R*, pero como vamos a utilizar el paquete auxiliar *Roxygen2* no será necesario realizar los cambios manualmente. En apartados posteriores se mostrará la evolución del archivo según se añadan las funciones.

5.2.1 Archivos .R

[22] A continuación, incluimos en el subdirectorio *R* los archivos .R que tienen implementada la funcionalidad del paquete. Estos archivos tienen una estructura concreta, que permitirá documentar el paquete posteriormente a partir de la información descrita en el archivo del código real. La siguiente imagen muestra el aspecto de un archivo .R, mas adelante comentaremos los elementos que aparecen en él.

```

1 #' @title Matrix distance by distance and approach type.
2 #' @description To calculate the matrix distance by using \code{distance} and \code{approach} type.
3 #' @param list is a clusters list.
4 #' @param distance is a literal. The distance type to be used.
5 #' @param approach is a literal. The approach type to be used.
6 #' @details This function is part of the hierarchical clusterization method. The function calculates the
7 #' matrix distance by using the distance and approach type given.
8 #' @details The \code{list} parameter will be a list with the clusters as rows and columns.
9 #' @details The function avoids distances equal 0 and undefined clusters.
10 #' @examples
11 #'
12 #' data <- c(1,2,1,3,1,4,1,5,1,6)
13 #'
14 #' clusters <- toList(data)
15 #'
16 #' mdAgglomerative(clusters, 'EUC', 'MAX')
17 #'
18 #' mdAgglomerative(clusters, 'CHE', 'AVG')
19 #'
20 #' @export
21
22
23 mdAgglomerative <- function(list,distance,approach){
24   res <- c()
25   for (i in seq_len(length(list))){
26     for (j in seq_len(length(list))){
27       if(is.null(list[[i]]) | is.null(list[[j]])){
28         dist <- 0
29       } else if(i ==j){
30         dist <- 0
31       }
32       else {
33         dist <- clusterDistance(list[[i]],list[[j]],approach,distance)
34       }
35       res <- c(res, dist)
36     }
37   }
38   ret <- matrix(res, nrow=length(list))
39 }
40

```

Figure 14: Ejemplo de un archivo .R

El código R de la función corresponde desde la línea 23 hasta la 39. En ellas se define la función *mdAgglomerative* que calculará la matriz de distancias entre todos los elementos del parametro *list* aplicando las técnicas definidas por *distance* y *approach*. No detallaremos su contenido , pues existen funciones específicas que describen el funcionamiento de todas ellas.

Desde la primera hasta la línea 20 se incluyen características necesarias que documentan la función o todo el contenido del archivo. Rellenando

todos estos apartados, ofrecemos información necesaria para el usuario para conocer como utilizar las funciones incluidas. Estos campos son los siguientes, pero es posible añadir algunos más [28]:

- *@title* indica el título de la función recogida en el paquete.
- *@description* es una breve descripción sobre la función.
- *@param* (una entrada por cada parámetro de la función) detalla las características que debe tener cada parámetro de la función, indicando su tipo de dato o los valores posibles que puede tomar.
- *@details* describe en profundidad el funcionamiento del archivo y aporta información extra para los usuarios.
- *@author* indica quien es el autor o autores del paquete junto a sus emails de contactos.
- *@return* describe el resultado de cada función.
- *@examples* recoge diferentes ejemplos de uso de la función. Muestra como sería una llamada real a la función. Estos ejemplos se utilizarán posteriormente en la revisión del paquete cuando sea creado.
- *@export* es la sentencia que finaliza el espacio dedicado a la documentación y que permite crear automáticamente los archivos que incluyan estos apartados.
- *@importFrom* es una sentencia que aparece en algunos archivos para importar funciones de otros paquetes. Si no se incluye esta opción, no será posible actualizar correctamente el archivo *NAMESPACE*.

5.2.2 Documentación de paquetes.

Cuando hemos completado toda la información del archivo, desde la consola de RStudio ejecutamos la sentencia *devtools::document()* o simplemente *document()* porque ya hemos instalado y cargado el paquete *devtools*. Obtenemos la siguiente respuesta.

```

> devtools::document()
Updating LearningHClust documentation
First time using roxygen2. Upgrading automatically...
Updating roxygen version in C:\Users\Roberto\Desktop\TFG\LearningHClust\DESCRIPTION
Loading LearningHClust
Warning: The existing 'NAMESPACE' file was not generated by roxygen2, and will not be overwritten.
Writing mdAgglomerative.Rd

```

Figure 15: Primera ejecución del comando *document()*

Podemos apreciar el comentario de la consola indicando que es la primera vez que utilizamos el comando. Además, aparece un aviso o warning para que prestemos atención a que el archivo *NAMESPACE* no se ha modificado automáticamente. Esto ocurre porque la estructura básica del paquete no se ha creado con *roxygen2* y nos ofrecía otras opciones, o es posible que el error proceda de no incluir la sentencia *@export* al final de la documentación del archivo .R. Esta cuestión es fundamental en nuestro desarrollo porque queremos aprovecharnos de las ventajas de *Roxygen2*, por eso conseguiremos que se modifique el archivo *NAMESPACE* eliminándolo manualmente y ejecutando de nuevo el comando *document()* [8]. En cada ejecución del comando, aparece una lista con todos los archivos creados y modificados desde la última ejecución. Cada vez que documentemos una o varias funciones, incluiremos la documentación utilizando el comando del mismo modo.

Solucionando este problema, la consola de RStudio no presentará de nuevo el aviso y modificará el archivo *NAMESPACE* automáticamente en cada ejecución. Así, modificaremos el archivo obteniendo la apariencia que se muestra en la imagen:

```

1 # Generated by roxygen2: do not edit by hand
2
3 export(mdAgglomerative)
4 |

```

Figure 16: Actualización automática de *NAMESPACE* con el comando *document()*.

Creando la documentación de las funciones con este paquete, tenemos acceso a la ayuda rápida ofrecida por RStudio utilizando el comando *?nombre-Función()*. Así el usuario puede consultarla desde el propio entorno.

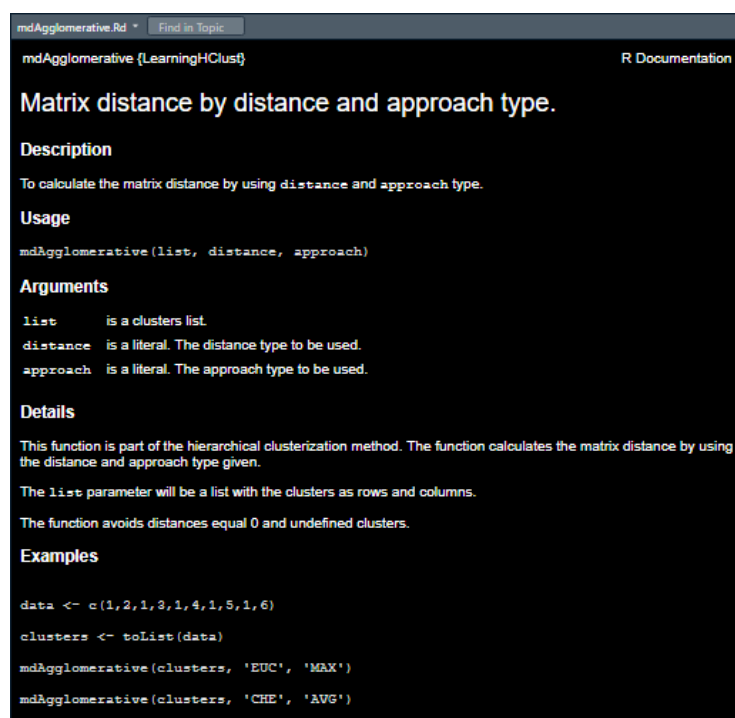


Figure 17: Acceso rápido a la documentación de la función.

Teniendo acceso a la descripción, los detalles y los modos de uso recogidos en el manual del usuario.

El archivo *DESCRIPTION* se modifica automáticamente la primera vez que ejecutamos el comando de la documentación; incluye la referencia a la versión del paquete *roxygen2* que se está utilizando. En nuestro caso particular:

```
RoxygenNote: 7.1.1
```

Figure 18: Actualización de archivo *DESCRIPTION* con la referencia a *roxygen2*.

A continuación, repetimos este proceso para incluir todas las funciones necesarias en el paquete. Incluiremos todos los archivos *.R*, después ejecutaremos el comando *document()* y obtendremos todos los archivos *.Rd* y tendremos disponible la documentación de todas las funciones.

Cada archivo *.R* que completamos con su documentación, crea un archivo *.Rd* en el subdirectorio *man*. Estos archivos incluyen la información que hemos añadido en los archivos *.R* pero formateada de diferente manera [13]. El archivo *.Rd* de la función utilizada como ejemplo en este documento tiene este aspecto:

```

1  % Generated by roxygen2: do not edit by hand
2  % Please edit documentation in R/mdAgglomerative.R
3  \name{mdAgglomerative}
4  \alias{mdAgglomerative}
5  \title{Matrix distance by distance and approach type.}
6  \usage{
7    mdAgglomerative(list, distance, approach)
8  }
9  \arguments{
10   \item{list}{is a clusters list.}
11
12   \item{distance}{is a literal. The distance type to be used.}
13
14   \item{approach}{is a literal. The approach type to be used.}
15 }
16 \description{
17   To calculate the matrix distance by using \code{distance} and \code{approach} type.
18 }
19 \details{
20   This function is part of the hierarchical clusterization method. The function calculates the
21   matrix distance by using the distance and approach type given.
22
23   The \code{list} parameter will be a list with the clusters as rows and columns.
24
25   The function avoids distances equal 0 and undefined clusters.
26 }
27 \examples{
28
29   data <- c(1,2,1,3,1,4,1,5,1,6)
30
31   clusters <- tolist(data)
32
33   mdAgglomerative(clusters, 'EUC', 'MAX')
34
35   mdAgglomerative(clusters, 'CHE', 'AVG')
36 }
37
38

```

Figure 19: Ejemplo de archivo .Rd.

5.2.3 Manual de usuario.

También, aprovecharemos las facilidades de *roxygen2* para crear un manual de usuario. Después de elaborar toda la documentación como se ha descrito previamente, hemos creado un manual de usuario utilizando el comando:

- *devtools::build_manual(path=getwd())*.

Recoge en un archivo .pdf la documentación de todas las funciones, junto al contenido del archivo *DESCRIPTION*. Todas las funciones ordenadas alfabéticamente junto a sus ejemplos, descripción o detalles de los parámetros.

Este manual se incluye en este documento en la sección *MANUAL DE USUARIO*.

5.2.4 Viñetas.

Finalmente, el último elemento de documentación utilizado por CRAN para describir los paquetes son las *viñetas*, combinando texto y código R. A continuación se muestra como se ha desarrollado la viñeta correspondiente a este paquete.

Para la creación de la viñeta, y aprovechando las ventajas de RStudio, se ha utilizado el paquete *rmarkdown* [21]. Por eso, el primer paso será instalar y cargar el paquete con los comandos de R: *install.packages("rmarkdown")* y *library(rmarkdown)*. Para la compilación de la viñeta será necesario el paquete *knitr* [15], que también será instalado y cargado de la misma manera. Debemos actualizar el archivo *DESCRIPTION* como se muestra en la imagen para poder construir las viñetas correctamente.

```
Suggests: knitr, rmarkdown
VignetteBuilder: knitr
```

Figure 20: Actualización de archivo *DESCRIPTION* para crear viñetas.

Desde la interfaz de RStudio crearemos un nuevo archivo .Rmd (o archivo R Markdown) indicando el título, el autor y el tipo de salida del archivo que deseamos [18]. El aspecto inicial del nuevo archivo será:

```
1 ---
2 title: "Untitled"
3 author: "roberto.alcantara"
4 date: "4/9/2020"
5 output: html_document
6 ---
7
8 {r setup, include=FALSE}
9 knitr::opts_chunk$set(echo = FALSE)
10
11
12 ## R Markdown
13 |
```

Figure 21: Estado inicial del archivo .Rmd

Sin embargo, realizaremos algunas modificaciones tanto en la cabecera como en la parte de configuración del paquete para obtener el resultado esperado y acorde a las características de nuestro paquete. En la cabecera indicaremos el título de la viñeta, el tipo de archivo sobre el que imprimiremos la salida y algunas características sobre la configuración de la viñeta (el Título que se mostrará, el método de compilación y la codificación del archivo). En la parte de configuración relacionada con el paquete, incluiremos nuestro paquete para tener acceso a las funciones que se incluyen en él. De esta manera, nuestra viñeta quedará así:

```
1 ---
2 title: "Learning Hierarchical Clusterization"
3 output: rmarkdown::html_vignette
4 vignettes: >
5   %\VignetteIndexEntry{Learning Hierarchical Clusterization}
6   %\VignetteEngine{knitr::rmarkdown}
7   %\VignetteEncoding{UTF-8}
8 ---
9
10 {r setup, include=FALSE}
11 library(LearningHClust)
12 knitr::opts_chunk$set(collapse = TRUE, comment = "#>")
13
14
```

Figure 22: Modificaciones sobre el archivo .Rmd

Ahora, podemos completar la viñeta con el contenido del paquete, definiendo datos que utilizar y mostrando la ejecución de las diferentes funciones contenidas. El usuario final conocerá ejemplos del uso de las funciones y la salida obtenida en cada uno de ellos.

Una vez completada la viñeta, debemos construirla para obtener el archivo de salida necesario. Para este proceso, vamos a utilizar de nuevo el paquete *devtools*, llamando a su función *build_vignettes()*. Esta función construirá la viñeta ejecutando todos los fragmentos de código R, e informará en caso de que exista cualquier problema durante la ejecución (en este caso, la creación se detiene). Ejecutando el comando, obtenemos:

```
> devtools::build_vignettes()
Building LearningHClust vignettes
--- re-building 'learningHClust.Rmd' using rmarkdown
--- finished re-building 'learningHClust.Rmd'

Moving learningHClust.html, learningHClust.R to doc/
Copying learningHClust.Rmd to doc/
Building vignette index
```

Figure 23: Ejecución del comando *devtools::build_vignettes()*

Si no existen fallos en el código, se realiza la compilación del archivo *.Rmd* del subdirectorio *vignettes*. A partir de esta compilación, se crea un subdirectorio *Meta* con un archivo *.rds* que es el código compilado de la viñeta, y que creará, en el subdirectorio *doc*, que es posible que se almacene dentro de un subdirectorio *inst*, el archivo *.html* con la viñeta creada (así como un archivo *.R* y una copia del archivo *.Rmd* contenido en el subdirectorio *vignettes*). En caso de que se produzca algún error, se indicará en la consola y deberá solucionarse.

Cuando se contruye el paquete final, los archivos se reorganizan. El subdirectorio *vignettes* incluye únicamente el archivo *.Rmd*, en el subdirectorio *build* se sitúa el archivo auxiliar *.rds* que permite la creación final de la viñeta, y en el subdirectorio *inst/doc* se incluye la viñeta final en formato *html*, el archivo *.Rmd* y el archivo *.R* que solo incluye el contenido de código R de la viñeta.

Uno de los requisitos exigidos por CRAN para la publicación del paquete es la creación de la viñeta del paquete en formato PDF. En este caso, y para obtener este archivo, debemos modificar el tipo de archivo de salida que deseamos. En nuestro archivo *.Rmd* sustituimos el parámetro *output: rmarkdown::html_vignette* por *output: rmarkdown::pdf_document*, ejecutamos de nuevo el comando de construcción de la viñeta y obtenemos el archivo PDF con el contenido.

El resultado final de la viñeta en formato PDF se encuentra en la sección de este mismo documento *VIIÑETA*.

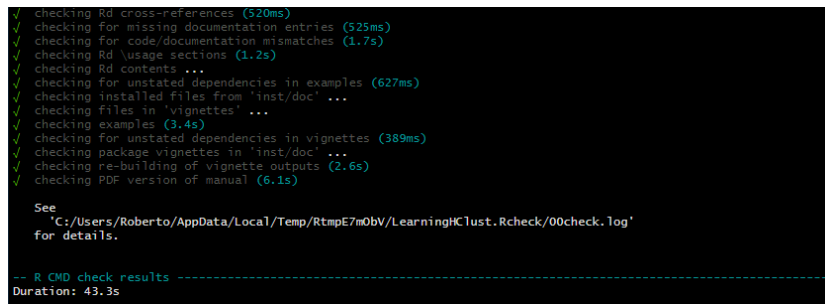
5.2.5 Construcción del paquete.

Después de crear el contenido del paquete, tanto las funciones como toda la documentación relativa a ellas, el último paso será crear el paquete final para poder distribuirlo y publicarlo. Primero, debemos comprobar que el paquete esta correctamente creado. Para ello, debemos ejecutar el comando `check()` que nos informará sobre el estado del paquete; si algo no es adecuado, mostrará el error por pantalla y no permitirá crear el paquete (dependiendo de la gravedad del comentario). En nuestro caso, al igual que en todo el desarrollo, utilizaremos el paquete *devtools* para estos últimos pasos:

```
- devtools::check(  
  cran = TRUE,  
  manual = TRUE,  
  pkg = "C:/Users/Roberto/Desktop/TFG/LearnClust"  
)
```

En el comando se observan algunos parámetros específicos. Utilizando *cran* como "TRUE" exigimos la revisión bajo las condiciones del repositorio oficial de código R, indicando *manual* como "TRUE" exigimos que se revise también el manual de usuario, y por último, especificamos la ruta del paquete que deseamos construir. Este último parámetro no es necesario si se ejecuta sobre un espacio de trabajo del propio paquete.

Cuando ejecutamos el comando aparecen líneas de consola indicando los elementos revisados en cada instante junto al tiempo empleado, así como algunas opciones de la revisión ejecutada. La siguiente imagen es un ejemplo de ello:



```
✓ checking Rd cross-references (520ms)  
✓ checking for missing documentation entries (525ms)  
✓ checking for code/documentation mismatches (1.7s)  
✓ checking Rd \usage sections (1.2s)  
✓ checking Rd contents ...  
✓ checking for unstated dependencies in examples (627ms)  
✓ checking installed files from 'inst/doc' ...  
✓ checking files in 'vignettes' ...  
✓ checking examples (3.4s)  
✓ checking for unstated dependencies in vignettes (389ms)  
✓ checking package vignettes in 'inst/doc' ...  
✓ checking re-building of vignette outputs (2.6s)  
✓ checking PDF version of manual (6.1s)  
  
See  
  'C:/Users/Roberto/AppData/Local/Temp/RtmpE7mDbV/LearningHClust.Rcheck/00check.log'  
for details.  
  
-- R CMD check results -----  
Duration: 43.3s
```

Figure 24: Revisando elementos del paquete antes de construir.

Finalmente, después de obtener un resultado correcto (sin errores), podemos construir el paquete en sí, el archivo *.tar.gz* que incluirá todo el contenido. Ejecutando:

```
- devtools::build()
```

De este modo obtenemos el archivo final *LearnClust_1.0.tar.gz* y la respuesta por consola de que todo el proceso se ha realizado correctamente.

```

> devtools::build()
✓ checking for file 'C:/Users/Roberto/Desktop/TFG/LearningHClust/DESCRIPTION' (390ms)
- preparing 'LearningHClust':
✓ checking DESCRIPTION meta-information ...
- installing the package to build vignettes
✓ creating vignettes (5.8s)
- checking for LF line-endings in source and make files and shell scripts
- checking for empty or unneeded directories
- building 'LearningHClust_1.0.tar.gz'

[1] "C:/Users/Roberto/Desktop/TFG/LearningHClust_1.0.tar.gz"

```

Figure 25: Construyendo el paquete.

5.3 Algoritmos incluidos en el paquete.

El paquete incluye dos tipos de funciones, unas implementan los algoritmos completos y las funciones auxiliares necesarias para realizar los diferentes pasos; y otras, que atienden al mismo nombre añadiendo el sufijo *.details*, describen el proceso seguido hasta obtener la solución final del algoritmo. Existen funciones auxiliares utilizadas en el algoritmo que no realizan una acción concreta de la descripción teórica del algoritmo, pero que son necesarias para la ejecución correcta del algoritmo. Estas funciones no se documentarán, ni incluirán la función paralela con el sufijo *.details*.

Sin embargo, las funciones que realizan pasos explícitos del algoritmo se organizan en archivos independientes e incluyen una documentación individualizada para cada una de ellas. Estas funciones,

- calcularán las distancias entre los clusters,
- localizarán los clusters correspondientes en función de la proximidad entre valores,
- agruparán clusters más próximos,
- definirán los conceptos de distancia entre los datos,
- definirán la similitud entre clusters a partir de los parámetros introducidos por el usuario, o,
- inicializarán los datos para adaptarlos a los formatos requeridos por el algoritmo.

En primer lugar, incluimos las funciones que calculan los diferentes tipos de distancia. Todas ellas han sido definidas en el contenido teórico de este documento. En todas ellas se incluye una función *.details* que explica el modo de cálculo de la distancia. Además, se incluye otra función similar, utilizada en el algoritmo extra incluido en el paquete donde se aplica una ponderación en el valor de la distancia aplicado a cada característica que también nombramos a continuación:

- Distancia euclídea: *euclideanDistance()* y *euclideanDistance.details()*
- Distancia de Manhattan: *manhattanDistance()* y *manhattanDistance.details()*
- Distancia de Canberra: *canberraDistance()* y *canberraDistance.details()*
- Distancia de Chebyshev: *chebyshevDistance()* y *chebyshevDistance.details()*
- Distancia Octal: *octileDistance()* y *octileDistance.details()*

Y las funciones que matizan diferente implicación a cada característica son:

- *euclideanDistanceW()*
- *manhattanDistanceW()*
- *canberraDistanceW()*
- *chebyshevDistanceW()*
- *octileDistanceW()*

Es importante diferenciar ambas funciones, en las del primer grupo se calcula la distancia sobre clusters de dos elementos; sin embargo, las funciones donde se aplica un "peso" particular a cada característica no tienen límite, pueden incluir tantos valores como se desee.

Todos los algoritmos implementan las funciones similares al desarrollo manual del algoritmo, es decir, realizan los pasos como si se tratará de una resolución manual del problema. Detallaremos los algoritmos descritos en el apartado teórico de este documento, así como las funciones complementarias *.details* que permiten conocer el proceso seguido para obtener la solución final mostrando los estados intermedios de los datos y las funciones realizadas en cada momento. Las funciones descriptivas permitirán a usuarios con un conocimiento más básico del clustering jerárquico conocer el proceso en profundidad y facilitar el aprendizaje.

Se describen, a continuación, las funciones individuales de cada algoritmo:

5.3.1 Clustering Jerárquico Aglomerativo (Agglomerative Hierarchical Clustering)

El paquete incluye el algoritmo clásico de clusterización jerárquica aglomerativa. La implementación es exactamente como se define el algoritmo teórico en el apartado *MARCO TEÓRICO Y SOCIAL* de este documento. Otros paquetes incluyen la implementación del algoritmo, pero ninguno de ellos explica el funcionamiento teórico de los mismos. Es una herramienta muy útil para usuarios que estén aprendiendo estos algoritmos y una manera de comparar la ejecución automática con el desarrollo manual. Por otra parte, se detallan todas las funciones que intervienen para poder aplicarlas individualmente y conocer el modo de funcionamiento. No existen otros paquetes en el sitio web oficial de CRAN dedicados a este fin. Destacar que se han aprovechado objetos en R de otros paquetes para facilitar el formato de salida, por ejemplo, los dendogramas.

- *toList*: es la función encargada de inicializar los datos recibidos. A partir de los datos numéricos recibidos como parámetro (que puede ser un vector numérico de longitud par, una matriz con dos columnas o un data frame también con dos columnas), crea una lista inicial de clusters con un único elemento cada uno. Estos clusters serán los que agruparemos en cada iteración del algoritmo. El resultado de su ejecución será una lista con los clusters individuales. Debemos aclarar que en este caso, se añade un elemento adicional a los clusters que indica si en el proceso general el cluster ha sido agrupado ya o debemos seguir calculando su distancia al resto de elementos.
- *mdAgglomerative*: Implementa la función que calcula la matriz de distancias entre todos los clusters. Como parámetros recibirá los clusters existentes, la definición de distancia deseada (euclídea, de manhattan, etc) y el concepto de similitud o proximidad seleccionado por el usuario, pudiendo ser este la distancia entre los elementos más próximos de cada cluster, la distancia entre los dos elementos más lejanos o la media de todas las distancias entre los elementos de ambos clusters. Estos dos últimos parámetros son elegidos por el usuario. El algoritmo completo actualizará automáticamente que clusters están activos o no en un momento determinado. Para evitar problemas en el algoritmo, la función comprueba si las distancias son nulas (por ser dos clusters iguales) o si algún cluster no es correcto.

El resultado de esta función será la matriz completa con las distancias calculadas entre todos los clusters del parámetro *list*.

- *clusterDistance*. Debido a la complejidad total de la función *mdAgglomerative* y a las buenas prácticas recomendadas por CRAN para el desarrollo de código R, se ha modularizado la función y existen otras funciones más simples centradas en una tarea más concreta. Esta función es una de ellas, se encarga de calcular la distancia entre todos los elementos que forman dos clusters en función del tipo de distancia deseado por el usuario. Cuando todos los valores están calculados, se seleccionará el correspondiente a la

definición de similitud indicada también por el usuario utilizando la función *clusterDistanceByApproach*. cuando se obtiene un resultado numérico final, se completa el lugar correspondiente en la matriz de distancias comentada previamente.

- *minDistance*: Es la función encargada de buscar el elemento menor dentro de la matriz de distancias. Esta función recibe la matriz como parámetro y devuelve un valor numérico con el resultado. Evita las distancias nulas entre los clusters, es decir, todas las que tengan valor 0. Utiliza una función auxiliar para comenzar a buscar que selecciona un valor numérico distinto de cero, y después realiza la búsqueda recorriendo toda la matriz.
- *getCluster*: esta función localiza los clusters que tienen una distancia dada, es decir, busca en la matriz de distancias una distancia conocida e incluida en ella. Recibe por parámetros la distancia que debe buscar y la matriz con todas las distancias entre clusters. Si existen dos o más distancias iguales, seleccionará la primera, comenzando por la esquina superior izquierda. Como resultado, obtendremos los dos clusters que se van a unir.
- *newCluster*: función que agrupa dos clusters. Recibe como parámetros todos los clusters existentes y el índice con los dos clusters que se van a agrupar. Los localiza en la lista de clusters y crea uno nuevo. Para la ejecución completa del algoritmo, esta función es la encargada de controlar que clusters continúan o no activos, es decir, si dos clusters se han unido, ya no tendrán importancia para seguir siendo candidatos a agrupamiento. el resultado será una lista actualizada con el cluster creado y los dos clusters agrupados "desactivados".
- *agglomerativeHC*: Es la función principal que realiza el agrupamiento de los clusters, se encarga tanto de recibir y preparar los datos como de realizar los cálculos necesarios para obtener un cluster único con todos los elementos, es decir, de obtener una solución. Exige como parámetros los datos a analizar; en formato vector numérico, matriz de datos con dos columnas o data frames con dos columnas; la definición de distancia entre los clusters o proximidad y el modo de calcular la distancia entre dos puntos cualesquiera. Realiza todas las operaciones de las funciones descritas previamente y comprueba las características de los datos para finalizar la ejecución cuando se obtiene un único cluster con todos los elementos. La función realiza otras operaciones internas para poder presentar la solución de una manera correcta (pero no entraremos en detalles porque no implementan funciones explícitas del algoritmo teórico). La solución final incluye la lista de clusters creados durante la ejecución, una matriz que indica el orden y la agrupación de clusters, y un dendograma que permite visualizar gráficamente estas uniones.

De la misma manera que las funciones recogidas anteriormente, se incluyen otras que atienden a los mismos nombre añadiendo el sufijo *.details* que

detallan el procedimiento de cada función individual. Estas funciones son:

- *toList.details*
- *mdAgglomerative.details*
- *clusterDistance.details*
- *clusterDistanceByApproach.details*
- *minDistance.details*
- *getCluster.details*
- *newCluster.details*
- *agglomerativeHC.details*

Se pueden encontrar todas estas funciones detalladas en la documentación de cada una de ellas, en el manual de usuario o en los ejemplos de la viñeta incluida en este documento.

5.3.2 Clustering Jerárquico Divisivo (Divisive Hierarchical Clustering)

Del mismo modo que el algoritmo anterior, existen otros paquetes que lo implementan, pero el contenido de este está dirigido a aquellos usuarios no expertos que inician el estudio del análisis jerarquizado. Sin embargo, a diferencia que con el algoritmo aglomerativo, no existen demasiados documentos que expliquen teóricamente este algoritmo. En muchas ocasiones aparece después de la explicación teórica del algoritmo aglomerativo y no se profundiza en este; pero existen aspectos que se deben tener en cuenta en este algoritmo. A continuación se detallan las funciones que lo implementan y el papel que realizan dentro del algoritmo general.

- *toListDivisive*: es la función que inicializa los datos de entrada, realiza los mismos pasos que su homóloga aglomerativa. El parámetro de entrada será de nuevo un vector numérico de longitud par, una matriz, o un data frame de dos columnas. El resultado será una lista de clusters de un único elemento, en este algoritmo, será el resultado final.
- *initClusters*: esta función es esencial en el algoritmo divisivo. A partir de los clusters individuales obtenidos con la función anterior, crea todos los clusters posibles de diferentes tamaños hasta obtener el cluster con todos los elementos (que será nuestro estado inicial en la ejecución completa del algoritmo). El motivo de esta función, se debe a que cuando se produce la división de un cluster complejo en dos más simples, debemos comprobar todos los subclusters posibles que se pueden formar dividiendo el ya existente. El resultado será una lista muy extensa con todos los subclusters posibles. Esta función implica un cuello de botella importante para el

algoritmo, es la que más recursos consume y la que limitará el tamaño de los datos procesables por el algoritmo.

- *mdDivisive*: esta función calcula la matriz de distancia entre todos los clusters de la lista activa. Realiza el mismo procedimiento que la función *mdAgglomerative* pero añade la condición de si dos clusters son complementarios o no, que se explicará posteriormente. Descarta las distancias entre clusters con valor 0, y el resultado final es una matriz de distancia entre todos los clusters. La condición de complementariedad provoca una gran cantidad de elementos cuyo valor es cero.
- *complementaryClusters*: es otra función fundamental para el algoritmo divisivo. La función *initClusters* crea todos los subclusters disponibles a partir del cluster inicial que se va a dividir. Con esta función, aplicada sobre el cálculo de la matriz de distancia, comprobamos si los clusters sobre los que se calcula la distancia son complementarios o no, es decir, si es posible dividir el cluster inicial en los dos subclusters siendo esta división válida. Entre ambos deben incluir todos los elementos del cluster raíz que va a dividirse y no tener elementos repetidos. Si no se cumple esta condición, la matriz de distancia no tendrá en cuenta los dos subclusters en cuestión. Los parámetros de entrada serán los dos clusters a comprobar y una lista con los elementos que deben incluir entre ambos.
- *maxDistance*: a diferencia del algoritmo aglomerativo, aquí se dividen los clusters más heterogéneos, es decir, cuya distancia sea mayor. Por eso, esta función busca el mayor valor de toda la matriz de distancia. El parámetro de entrada será únicamente la matriz de distancias, y el resultado final el valor máximo de la matriz.
- *divisiveHC*: esta función incluye el algoritmo de clusterización jerárquica divisiva completo. Esta función recibe los datos de entrada como parámetro, en formato vector numérico, matriz o data frame. Inicializa los datos para poder aplicar el algoritmo sobre ellos. Calcula la matriz de distancia entre todos los clusters y selecciona cuál debe ser la división. Repite después el procedimiento hasta que alcanza una solución (no existen más clusters que se puedan dividir de nuevo). Al igual que el algoritmo aglomerativo, recibe como parámetros las definiciones de distancia entre clusters y proximidad, y las aplica en las funciones internas del algoritmo. El resultado final será una lista de clusters, el primero será el cluster que incluye todos los elementos y finalizará con todos los elementos individuales que no se puedan dividir de nuevo.

Existen funciones que realizan actividades auxiliares en alguna función comentada anteriormente. Utilizará de nuevo las funciones *clusterDistance* y *clusterDistanceByApproach* como ocurría en el algoritmo aglomerativo.

Todas estas funciones, igual que el caso anterior, incluyen unas funciones paralelas con el sufijo *.details* que explican detalladamente el funcionamiento

de cada una:

- *toListDivisive.details*.
- *initClusters.details*.
- *complementaryClusters.details*.
- *mdDivisive.details*.
- *maxDistance.details*.
- *divisiveHC.details*.

5.3.3 Clustering Jerárquico por Correlación (Correlative Hierarchical Clustering)

Este algoritmo no está implementado en ningún otro paquete. Se ha desarrollado mezclando la correlación estadística y la clasificación jerárquica aglomerativa. Se ha considerado un algoritmo útil en la clasificación de elementos (tomados como clusters). De la correlación estadística utilizamos la clasificación de los elementos con respecto a un objetivo o *target*. Por otro lado, el agrupamiento jerárquico aglomerativo nos indicará en qué orden se han producido estas uniones, teniendo en todo momento la distancia entre el objetivo y cualquiera de los elementos de la clasificación general.

El objetivo será un cluster con las mismas características que los datos introducidos en el algoritmo completo. Todas estas condiciones serán comprobadas por el algoritmo para evitar inconsistencias; en cualquier caso, el usuario será consciente del estado de los parámetros introducidos si son ajustados a las características del algoritmo.

Además, se ha añadido como extra la capacidad de ponderar la implicación de cada elemento dentro de un mismo cluster, es decir, existe una variable *pesos* que se aplicará durante el cálculo de la distancia entre cada cluster al objetivo para potenciar según considere el usuario cada propiedad. Para esta ponderación, se aportan opciones aplicables a los valores del peso: permitirá al usuario normalizar los valores introducidos de manera que mantenga la proporción introducida inicialmente, pero con valores normalizados (valores en el intervalo $[0,1]$ cuya suma total sea 1).

Existen diferentes aplicaciones para el algoritmo, permitiendo al usuario conocer los elementos más próximos al objetivo, es decir, los que tienen características similares entre todos ellos, o por el contrario, los últimos elementos de la lista solución serán los que se diferencien más.

A continuación se detallan las funciones de la implementación técnica del algoritmo, explicando la funcionalidad detallada de cada una:

- *initData*: esta función es la encargada de recibir el data frame como entrada del usuario y crear clusters con cada fila del data frame inicial. Inicializa los clusters que luego utilizará el algoritmo completo para calcular distancias y ordenarlos en función de la similitud con el objetivo.
- *initTarget*: esta función recibe y comprueba que el objetivo introducido es correcto para aplicarlo sobre el conjunto de datos principal. Estas comprobaciones incluyen que tenga las mismas características que el bloque de datos principal, mismo número de columnas y una única fila. Para ello recibe como parámetro tanto el *target* introducido como el conjunto de datos completo. Si no cumple las condiciones, inicializará un objetivo con las mismas columnas que el parámetro *data* y una única fila donde todos los elementos que lo formen tendrán valor 0. En cualquier caso, el usuarios conoce la decisión tomada por la función y por tanto, con que objetivo utilizará el algoritmo.
- *normalizedWeight*: es la función encargada de inicializar los valores utilizados en la ponderación. Recibe como parámetro el vector numérico con los pesos implicados, un valor booleano que indica si el usuario desea o no normalizar los pesos aplicados y el conjunto de datos completo para dar forma a este vector numérico. Si los valores introducidos no están definidos, la propia función inicializará el vector con todos los valores igual a 1, es decir, utilizará estos valores para no realizar ninguna modificación sobre la distancia calculada en un primer instante. Si el usuario decide no normalizar los valores, pero se han introducido correctamente, los valores no son modificados y serán los aplicados en los pasos siguientes. Y si decide normalizarlos, que será la opción por defecto, calculará los valores en proporción pero los transformará en valores comprendidos entre 0 y 1.
- *distances*: es la función encargada de calcular la distancia entre dos clusters aplicando la definición de distancia elegida y la ponderación introducida como parámetro. Es la encargada de delegar el cálculo en las funciones comentadas al inicio de esta sección. Los parámetros de entrada serán los clusters sobre los que haremos el cálculo, el vector numérico con los pesos y el tipo de distancia aplicable en cada caso. El resultado final será un valor numérico como resultado de aplicar la distancia y el peso comentado.
- *correlationHC*: esta función incluye el algoritmo completo que hemos definido en este apartado. Realiza tanto las funciones de inicialización de los datos, del objetivo y el tratamiento de los valores para la ponderación, como el cálculo de las distancias entre clusters, la ordenación de las mismas y la relación de cada una de las distancias con el cluster correspondiente. Finalmente, almacena en un único objeto R toda la información de salida (clusters ordenados, distancias ordenadas y representación gráfica), y la entrega al usuario. La representación gráfica del objeto solución será un dendrograma con las características del algoritmo de clasificación aglomerativo, permitirá al usuario conocer cual ha sido el orden de clasificación y

por tanto de proximidad entre todos los clusters.

Se incluyen ejemplos de todas estas funciones, junto a su documentación detallada en el manual de usuario o en la viñeta del paquete, ambos disponibles en este documento.

Al igual que en las funciones de los algoritmos anteriores, este incluye también los correspondientes funciones *.details* que facilitan la comprensión de cada una de ellas:

- *initData.details*.
- *initTarget.details*.
- *normalizedWeight.details*.
- *distances.details*.
- *correlationHC.details*.

5.4 Publicación del paquete en CRAN

[20] El último paso para compartir el paquete creado con la comunidad de CRAN, será publicarlo en su sitio web oficial. Para poder realizar este proceso, el paquete debe satisfacer todos los requisitos indicados por CRAN en <https://cran.r-project.org/doc/manuals/r-release/R-exts.html#Package-namespaces> y en <https://cran.r-project.org/web/packages/policies.html>. Esta última fase consta de tres pasos:

- Carga del paquete (Upload). Se realizará en la página oficial de CRAN que se muestra a continuación:

Submit package to CRAN

Step 1 (Upload)	Step 2 (Submission)	Step 3 (Confirmation)
<p>Your name*: <input type="text"/></p> <p>Your email*: <input type="text"/></p> <p>Package*: <input type="button" value="Seleccionar archivo"/> Ningún archivo seleccionado (* .tar.gz files only, max 100 MB size)</p> <p>Optional comment: <div style="border: 1px solid #ccc; height: 150px; width: 100%;"></div></p>		

*: Required Fields

Before uploading please ensure the following:

- The package contains a DESCRIPTION file.
- DESCRIPTION file contains valid maintainer field "NAME <EMAIL>".
- You submit a tar.gz created with R CMD build.
- You are familiar with the rest of the [CRAN policies](#)

Figure 26: Publicando el paquete.

Se incluirá el nombre del autor del paquete junto a un e-mail válido donde contactar con él. Además se añadirá el tarball creado del paquete después de revisar que se cumplen todos los requisitos.

- Control del contenido básico del paquete (Submission). Se muestran los elementos principales del paquete (imágenes inferiores) para que el autor revise el contenido que se va a publicar. Si todo es correcto, se carga definitivamente el paquete con *submit package*

Submit package to CRAN

Step 1
(Upload)
**Step 2
(Submission)**
Step 3
(Confirmation)

Package successfully uploaded - please review and submit

Your name:

Your email:

Package:

Optional comment:

Seleccionar archivo | Ningún archivo seleccionado
(*tar.gz files only, max 100 MB size)

Re-upload package/Edit information

Detected package information [non-editable]
In case of errors reupload package or contact cran team

Package:

Version:

Title:

Learning Hierarchical Clustering Algorithms

Figure 27: Comprobación del paquete (Parte 1).

Description: LearnClust includes classical hierarchical clustering algorithms, agglomerative and divisive clustering. Algorithms are implemented as a theoretical way, step by step. It includes some detailed functions that explain each step. Every function allows options to get different results using different techniques. The package explains non expert users how hierarchical clusterization algorithms work.

Author(s): Roberto Alcántara [aut, cre]

Maintainer Name: Roberto Alcántara

Maintainer Email: roberto.alcantara@edu.uah

License: Unlimited

Depends: magick

Suggests: knitr, rmarkdown

Imports:

Linking To:

Submit the package here if the above information is correct

Figure 28: Comprobación del paquete (Parte 2).

- Confirmación por parte de CRAN de que se ha recibido el paquete (Confirmation).

Submit package to CRAN

Step 1 (Upload) Step 2 (Submission) **Step 3 (Confirmation)**

The maintainer of this package has been sent an email to confirm the submission. After their confirmation the package will be passed to CRAN for review.

In case of technical problems regarding the website or the submission interface, contact the [CRAN sysadmin team](#).
In case of problems related to the package, its check results or the partly automated check system, contact the [CRAN team](#).

Figure 29: Confirmación del paquete por CRAN.

Cuando el autor publica el paquete en CRAN, la organización envía un correo de confirmación al contacto indicado. Este correo incluye un enlace a una nueva página de CRAN para aceptar, tanto la política de CRAN como la confirmación de que se ha revisado previamente por el autor. Este enlace es el siguiente.

Submit package to CRAN

Step 1 (Upload) Step 2 (Submission) **Step 3 (Confirmation)**

☒ I have read the [CRAN policies](#) and would like to submit this package to CRAN.

☒ I have checked the submission using `R CMD check --as-cran` and a **current version of r-devel**, as mandated by the CRAN Repository Policy. (You could do so using the win-builder service at <https://win-builder.r-project.org>)

Figure 30: Aceptar políticas de CRAN.

Después de aceptar las condiciones indicadas en la Figura 30, y tras un periodo de tiempo reducido, la organización envía un nuevo email que indica el resultado de la primera revisión automática del paquete. Esta revisión comprueba algunos aspectos que deben cumplirse. Si no es así, se indica en un archivo (log del paquete) incluido en el email de confirmación. Si todo ha sido correcto, también se indica que la revisión se ha superado con éxito.

En nuestro desarrollo particular, en un primer momento, el archivo log indicaba una incidencia que tuvo que ser corregida: El contenido del archivo *DESCRIPTION* no podía empezar ni por expresiones como "This package" ni por el nombre del paquete, por lo que hubo que modificar este aspecto y comenzar de nuevo el proceso.

Siempre se incluye una nota en este archivo log relacionada con el *mantenedor* del paquete para gestiones posteriores de los encargados de CRAN.

Por último, habrá que esperar hasta un máximo de 10 días para que se realice la revisión manual del paquete por una persona de la organización encargada de ello. Si existen aspectos que no sean adecuados, lo indicarán, permitirán que sean modificados, y se iniciará el proceso de nuevo.

6 CONCLUSIONES

La realización de este documento y del paquete "LearnClust" han permitido al autor del mismo adquirir y consolidar unos conocimientos previos. Es el último proyecto del Grado en Ingeniería Informática y se han puesto en práctica las distintas habilidades adquiridas en este tiempo.

Centrándonos en el proyecto de este documento, debemos destacar el desarrollo en lenguaje R como la principal habilidad desarrollada, Así como conocimiento sobre la arquitectura de los paquetes y la importancia de cada elemento que lo compone. Fundamental también el desarrollo de aspectos como la documentación de los archivos y de las funciones incluidas. Se ha ampliado el conocimiento sobre el análisis cluster, tanto aglomerativo como repartitivo.

Otras capacidades desarrolladas durante el proyecto han sido la gestión del tiempo y la planificación de tareas (destacando la replanificación de las mismas por diferentes motivos).

Durante todo el proceso, el alumno ha investigado, estudiado en profundidad y aplicado al paquete los requisitos mínimos exigidos por la comunidad oficial del lenguaje R, CRAN, para ser publicado y estar a disposición de todos los usuarios que lo deseen.

No existen en el repositorio oficial de CRAN otros paquetes dedicados a este mismo fin, con una visión didáctica sobre algoritmos teóricos. No siendo de gran complejidad, pero explicados en detalle para usuarios no expertos. Por todo esto, se considera que el paquete aporta nuevo contenido al sitio oficial y puede ser de ayuda para los demás.

En conclusión, ha sido un proyecto interesante, de un desarrollo semiprofesional, que trata de aportar nuevos contenidos a la comunidad del lenguaje R y a los usuarios que lo utilicen, se han consolidado habilidades previas y se han aprendido muchas otras.

7 VIÑETA

VIÑETA LearnClust

Ejemplos de uso

Learning Clusterization

LearnClust package allows users to learn how the algorithms get the solution.

1. The package implements distances between clusters.
2. It includes main functions that return the solution applying the algorithms.
3. It contains .details functions that explain the process used to get the solution. They help the user to understand how it gets the solution.

Datasets:

We initialize some datasets to use in the algorithms:

```
cluster1 <- matrix(c(1,2),ncol=2)

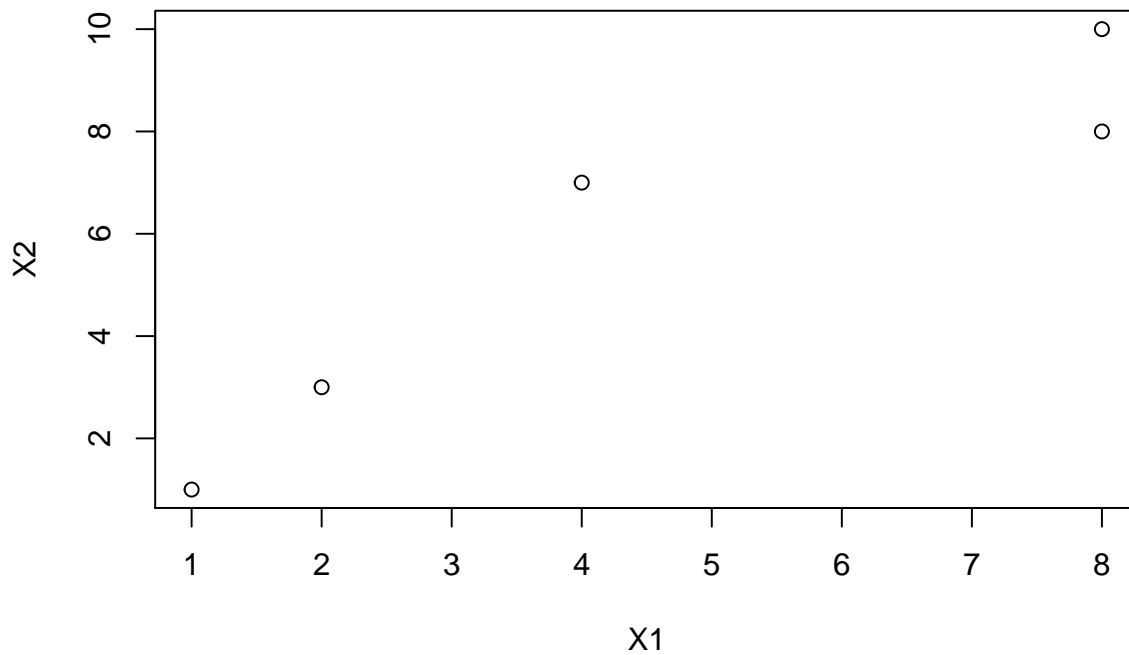
cluster2 <- matrix(c(2,4),ncol=2)

weight <- c(0.2,0.8)

vectorData <- c(1,1,2,3,4,7,8,8,8,10)
# vectorData <- c(1:10)

matrixData <- matrix(vectorData,ncol=2,byrow=TRUE)
print(matrixData)
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    2    3
#> [3,]    4    7
#> [4,]    8    8
#> [5,]    8   10

dfData <- data.frame(matrixData)
print(dfData)
#>   X1 X2
#> 1  1  1
#> 2  2  3
#> 3  4  7
#> 4  8  8
#> 5  8 10
plot(dfData)
```



```
cMatrix <- matrix(c(2,4,4,2,3,5,1,1,2,2,5,5,1,0,1,1,2,1,2,4,5,1,2,1), ncol=3, byrow=TRUE)
cDataFrame <- data.frame(cMatrix)
```

Distances

The package includes different types of distance:

- Euclidean Distance

```
edistance(cluster1,cluster2)
#> [1] 2.236068
```

- Manhattan Distance

```
mdistance(cluster1,cluster2)
#> [1] 3
```

- Canberra Distance

```
canberradistance(cluster1,cluster2)
#> [1] 0.6666667
```

- Chebyshev Distance

```
chebyshevDistance(cluster1,cluster2)
#> [1] 2
```

- Octile Distance

60

```
octileDistance(cluster1,cluster2)
#> [1] 2.414214
```

Each function has a .details version that explain how the calculus is done.

There are functions where some weights are applied to each element. These function are used in the extra algorithm. These functions are:

- Euclidean Distance with weight applied.

```
edistanceW(cluster1,cluster2,weight)
#> [1] 1.843909
```

- Manhattan Distance with weight applied.

```
mdistanceW(cluster1,cluster2,weight)
#> [1] 1.8
```

- Canberra Distance with weight applied.

```
canberradistanceW(cluster1,cluster2,weight)
#> [1] 0.3333333
```

- Chebyshev Distance with weight applied.

```
chebyshevDistanceW(cluster1,cluster2,weight)
#> [1] 1.6
```

- Octile Distance with weight applied.

```
octileDistanceW(cluster1,cluster2,weight)
#> [1] 1.682843
```

Agglomerative Hierarchical Clustering

This algorithm uses some functions according to the theoretical process:

1. We prepare data to be used in the algorithms. We create a cluster with each values. They could be different R types (vector, matrix or data.frame)

```
list <- toList(vectorData)

# list <- toList(matrixData)

# list <- toList(dfData)

print(list)
#> [[1]]
#>      [,1] [,2] [,3]
#> [1,]    1    1    1
#>
#> [[2]]
#>      [,1] [,2] [,3]
#> [1,]    2    3    1
#>
#> [[3]]
#>      [,1] [,2] [,3]
#> [1,]    4    7    1
#>
```

```
#> [[4]]
#>      [,1] [,2] [,3]
#> [1,]    8    8    1
#>
#> [[5]]
#>      [,1] [,2] [,3]
#> [1,]    8   10    1
```

2. We calculate the matrix distance using clusters from the first step. We use the distance and approach type that we want.

```
matrixDistance <- mdAgglomerative(list,'MAN','AVG')
print(matrixDistance)
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]    0    3    9   14   16
#> [2,]    3    0    6   11   13
#> [3,]    9    6    0    5    7
#> [4,]   14   11    5    0    2
#> [5,]   16   13    7    2    0
```

3. We get the minimal value from the matrix distance, that is, the distance between closer clusters.

```
minDistance <- minDistance(matrixDistance)
print(minDistance)
#> [1] 2
```

4. With the minimal distance, we look for the clusters with this distance separation. We take the clusters that will be joined.

```
groupedClusters <- getCluster(minDistance, matrixDistance)
print(groupedClusters)
#> [1] 4 5
```

5. These two clusters will create a new one.

```
updatedClusters <- newCluster(list, groupedClusters)
print(updatedClusters)
#> [[1]]
#>      [,1] [,2] [,3]
#> [1,]    1    1    1
#>
#> [[2]]
#>      [,1] [,2] [,3]
#> [1,]    2    3    1
#>
#> [[3]]
#>      [,1] [,2] [,3]
#> [1,]    4    7    1
#>
#> [[4]]
#>      [,1] [,2] [,3]
#> [1,]    8    8    0
#>
#> [[5]]
#>      [,1] [,2] [,3]
#> [1,]    8   10    0
```

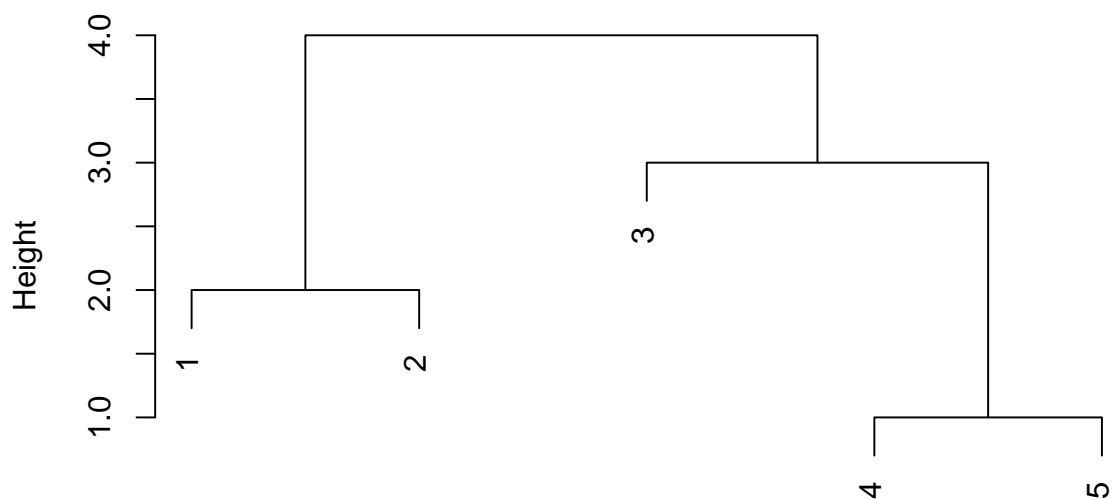
```
#>
#> [[6]]
#>      [,1] [,2] [,3]
#> [1,]    8    8    1
#> [2,]    8   10    1
```

6. We add the new cluster to the solution and repeat from step 2 to 5 until we get only one cluster.

The complete function that implement the algorithm is:

```
agglomerativeExample <- agglomerativeHC(dfData, 'EUC', 'MAX')
plot(agglomerativeExample$dendrogram)
```

Cluster Dendrogram



```
print(agglomerativeExample$clusters)
#> [[1]]
#>   X1 X2
#> 1  1  1
#>
#> [[2]]
#>   X1 X2
#> 1  2  3
#>
#> [[3]]
#>   X1 X2
#> 1  4  7
#>
```



```

#> [[4]]
#>   X1 X2
#> 1  8  8
#>
#> [[5]]
#>   X1 X2
#> 1  8 10
#>
#> [[6]]
#>   X1 X2
#> 1  8  8
#> 2  8 10
#>
#> [[7]]
#>   X1 X2
#> 1  1  1
#> 2  2  3
#>
#> [[8]]
#>   X1 X2
#> 1  4  7
#> 2  8  8
#> 3  8 10
#>
#> [[9]]
#>   X1 X2
#> 1  1  1
#> 2  2  3
#> 3  4  7
#> 4  8  8
#> 5  8 10
print(agglomerativeExample$groupedClusters)
#>   cluster1 cluster2
#> 1         4         5
#> 2         1         2
#> 3         3         6
#> 4         7         8

```

The package includes some auxiliary functions to implement the algorithm. These functions are:

- A function that updates active clusters. If two clusters have been joined, they will not be used again as individual clusters.

```

cleanClusters <- usefulClusters(updatedClusters)
print(cleanClusters)
#> [[1]]
#>      [,1] [,2] [,3]
#> [1,]    1    1    1
#>
#> [[2]]
#>      [,1] [,2] [,3]
#> [1,]    2    3    1
#>
#> [[3]]
#>      [,1] [,2] [,3]

```

```
#> [1,] 4 7 1
#>
#> [[4]]
#> NULL
#>
#> [[5]]
#> NULL
#>
#> [[6]]
#>      [,1] [,2] [,3]
#> [1,] 8 8 1
#> [2,] 8 10 1
```

- Two functions that calculate the distance between clusters using distance and approach values given.

```
distances <- c(2,4,6,8)
```

```
clusterDistanceByApproach <- clusterDistanceByApproach(distances, 'AVG')
print(clusterDistanceByApproach)
#> [1] 5
```

“clusterDistanceByApproach” get the value using approach type. This type could be “MAX”, “MIN”, and “AVG”

```
clusterDistance <- clusterDistance(cluster1, cluster2, 'MAX', 'MAN')
print(clusterDistance)
#> [1] 3
```

“clusterDistance” get the distance value between each element from one cluster to the other ones using distance type. This type could be “EUC”, “MAN”, “CAN”, “CHE”, and “OCT”

Agglomerative Hierarchical Clustering .DETAILS

This algorithm explains every function.

1. How clusters are initialized to be used. Initial data could be different R types (vector, matrix or data.frame)

```
list <- toList.details(vectorData)
#> 'toList' creates a list initializing datas by creating clusters with each one

# list <- toList(matrixData)

# list <- toList(dfData)

print(list)
#> [[1]]
#>      [,1] [,2] [,3]
#> [1,] 1 1 1
#>
#> [[2]]
#>      [,1] [,2] [,3]
#> [1,] 2 3 1
#>
#> [[3]]
#>      [,1] [,2] [,3]
#> [1,] 4 7 1
```

```
#>
#> [[4]]
#>      [,1] [,2] [,3]
#> [1,]    8    8    1
#>
#> [[5]]
#>      [,1] [,2] [,3]
#> [1,]    8   10    1
```

2. How the matrix distance is created.

```
matrixDistance <- mdAgglomerative.details(list,'MAN','AVG')
#>
#> 'mdAgglomerative' creates the matrix distance of every cluster given by 'list' parameter.
#>
#> It usesMANdistance andAVGapproach.
#>
#> It returns the matrix with all the distances between clusters depending on distance and approach.
#>
#> The matrix distance is:
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]    0    3    9   14   16
#> [2,]    3    0    6   11   13
#> [3,]    9    6    0    5    7
#> [4,]   14   11    5    0    2
#> [5,]   16   13    7    2    0
```

3. Choosing the minimal distance avoiding zero values.

```
minDistance <- minDistance.details(matrixDistance)
#>
#> 'minDistance' function gets the minimal value from a matrix.
#>
#> It returns the minimal value avoiding 0 values.
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]    0    3    9   14   16
#> [2,]    3    0    6   11   13
#> [3,]    9    6    0    5    7
#> [4,]   14   11    5    0    2
#> [5,]   16   13    7    2    0
#>
#>
#> 2 is the minimal value of the matrix.
```

4. Using the minimal distance, look for the clusters with this distance.

```
groupedClusters <- getCluster.details(minDistance, matrixDistance)
#>
#> 'getCluster' method searches the clusters which have the distance given.
#> Search for 2 in:
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]    0    3    9   14   16
#> [2,]    3    0    6   11   13
#> [3,]    9    6    0    5    7
#> [4,]   14   11    5    0    2
#> [5,]   16   13    7    2    0
```

```
#>
#> The clusters with the minimum distance are: 4, 5
```

5. With the clusters, it creates a new one and remove the previous from the initial list.

```
updatedClusters <- newCluster.details(list, groupedClusters)
#>
#> 'newCluster' function creates a new cluster from the clusters given.
#>
#> It adds the new cluster to 'list' and disables the clusters used to create the new one.
#>
#> Using 4 and 5 it searches the clusters in 'list' parameter and creates a new one with
#> their components.
#>
#> After the new cluster is created, the initial clusters must be disabled.
#>
#> The new cluster:
#>      [,1] [,2] [,3]
#> [1,]    8    8    1
#> [2,]    8   10    1
#>
#>
#> is added to the list:
```

6. We add the new cluster to the solution and repeat from step 2 to 5 until we get only one cluster.

The complete function that explains the algorithm is:

```
agglomerativeExample <- agglomerativeHC.details(vectorData, 'EUC', 'MAX')
#> Agglomerative hierarchical clustering is a classification technique that initializes
#> a cluster for each data.
#>
#> It calculates the distance between datas depending on the approach type given and
#>
#> it creates a new cluster joining the most similar clusters until getting only one.
#> 'toList' creates a list initializing datas by creating clusters with each one
#>
#> These are the clusters with only one element:
#> [[1]]
#>      [,1] [,2] [,3]
#> [1,]    1    1    1
#>
#> [[2]]
#>      [,1] [,2] [,3]
#> [1,]    2    3    1
#>
#> [[3]]
#>      [,1] [,2] [,3]
#> [1,]    4    7    1
#>
#> [[4]]
#>      [,1] [,2] [,3]
#> [1,]    8    8    1
#>
#> [[5]]
#>      [,1] [,2] [,3]
```

```

#> [1,]    8   10    1
#>
#> In each step:
#> - It calculates a matrix distance between active clusters depending on the approach and distance
#> - It gets the minimum distance value from the matrix.
#> - It creates a new cluster joining the minimum distance clusters.
#> - It repeats these steps while final clusters do not include all datas.
#>
#> -----
#> STEP => 1
#>
#> Matrix Distance (distance type = EUC, approach type = MAX):
#>      [,1]    [,2]    [,3]    [,4]    [,5]
#> [1,]  0.000000 2.236068 6.708204 9.899495 11.401754
#> [2,]  2.236068 0.000000 4.472136 7.810250 9.219544
#> [3,]  6.708204 4.472136 0.000000 4.123106 5.000000
#> [4,]  9.899495 7.810250 4.123106 0.000000 2.000000
#> [5,] 11.401754 9.219544 5.000000 2.000000 0.000000
#>
#> The minimum distance is: 2
#>
#> The closest clusters are: 4, 5
#>
#> The grouped clusters are added to the solution.
#>
#> Grouping clusters 4 and cluster 5, it is created a new cluster:
#> X1 X2
#> 1  8  8
#> 2  8 10
#>
#> The new cluster is added to the solution.
#>
#> -----
#> STEP => 2
#>
#> Matrix Distance (distance type = EUC, approach type = MAX):
#>      [,1]    [,2]    [,3] [,4] [,5]    [,6]
#> [1,]  0.000000 2.236068 6.708204  0  0 11.401754
#> [2,]  2.236068 0.000000 4.472136  0  0 9.219544
#> [3,]  6.708204 4.472136 0.000000  0  0 5.000000
#> [4,]  0.000000 0.000000 0.000000  0  0 0.000000
#> [5,]  0.000000 0.000000 0.000000  0  0 0.000000
#> [6,] 11.401754 9.219544 5.000000  0  0 0.000000
#>
#> The minimum distance is: 2.23606797749979
#>
#> The closest clusters are: 1, 2
#>
#> The grouped clusters are added to the solution.
#>
#> Grouping clusters 1 and cluster 2, it is created a new cluster:
#> X1 X2
#> 1  1  1

```

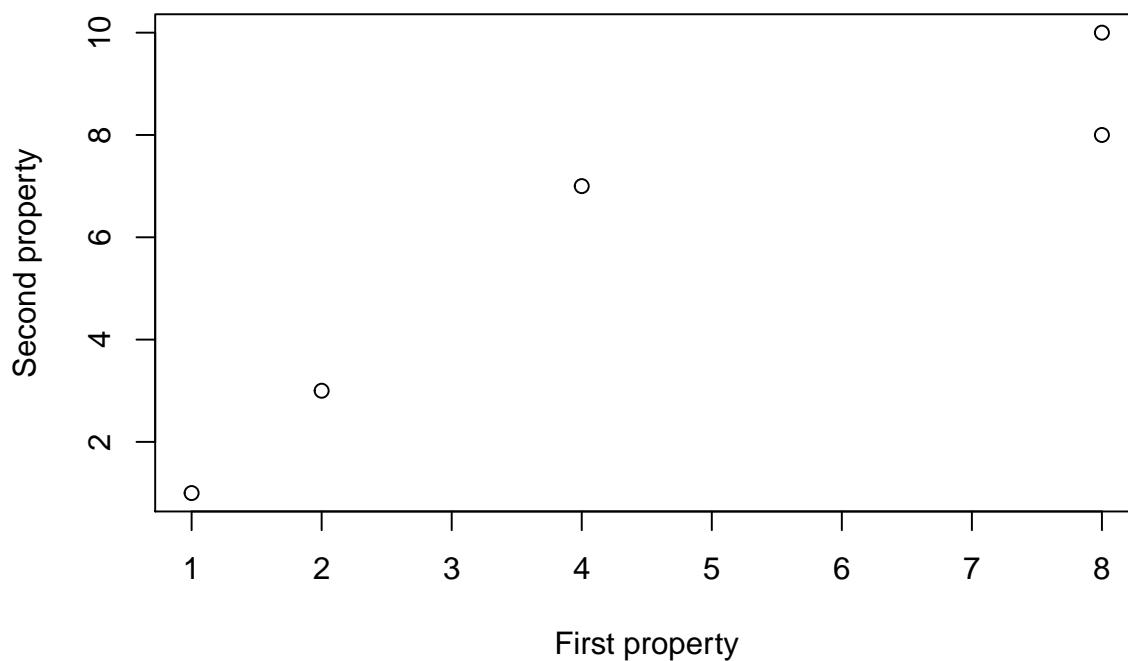
```

#> 2 2 3
#>
#> The new cluster is added to the solution.
#>
#> -----
#> STEP => 3
#>
#> Matrix Distance (distance type = EUC, approach type = MAX):
#>      [,1] [,2]      [,3] [,4] [,5]      [,6]      [,7]
#> [1,]    0    0 0.000000    0    0 0.000000 0.000000
#> [2,]    0    0 0.000000    0    0 0.000000 0.000000
#> [3,]    0    0 0.000000    0    0 5.000000 6.708204
#> [4,]    0    0 0.000000    0    0 0.000000 0.000000
#> [5,]    0    0 0.000000    0    0 0.000000 0.000000
#> [6,]    0    0 5.000000    0    0 0.000000 11.401754
#> [7,]    0    0 6.708204    0    0 11.40175 0.000000
#>
#> The minimum distance is: 5
#>
#> The closest clusters are: 3, 6
#>
#> The grouped clusters are added to the solution.
#>
#> Grouping clusters 3 and cluster 6, it is created a new cluster:
#>   X1 X2
#> 1  4  7
#> 2  8  8
#> 3  8 10
#>
#> The new cluster is added to the solution.
#>
#> -----
#> STEP => 4
#>
#> Matrix Distance (distance type = EUC, approach type = MAX):
#>      [,1] [,2] [,3] [,4] [,5] [,6]      [,7]      [,8]
#> [1,]    0    0    0    0    0    0 0.000000 0.000000
#> [2,]    0    0    0    0    0    0 0.000000 0.000000
#> [3,]    0    0    0    0    0    0 0.000000 0.000000
#> [4,]    0    0    0    0    0    0 0.000000 0.000000
#> [5,]    0    0    0    0    0    0 0.000000 0.000000
#> [6,]    0    0    0    0    0    0 0.000000 0.000000
#> [7,]    0    0    0    0    0    0 0.000000 11.40175
#> [8,]    0    0    0    0    0    0 11.40175 0.000000
#>
#> The minimum distance is: 11.4017542509914
#>
#> The closest clusters are: 7, 8
#>
#> The grouped clusters are added to the solution.
#>
#> Grouping clusters 7 and cluster 8, it is created a new cluster:
#>   X1 X2

```

```
#> 1 1 1
#> 2 2 3
#> 3 4 7
#> 4 8 8
#> 5 8 10
#>
#> The new cluster is added to the solution.
#>
#> This loop has been repeated until the last cluster contained every single clusters.
```

Initial Data Visualization



Divisive Hierarchical Clustering

This algorithm uses some functions according to the theoretical process:

1. We prepare data to be used in the algorithms. We create a cluster with each values. They could be different R types (vector, matrix or data.frame)

```
# list <- toListDivisive(vectorData)

# list <- toListDivisive(matrixData)

list <- toListDivisive(dfData[1:4,])

print(list)
#> [[1]]
#>      [,1] [,2]
#> [1,]    1    1
```

70

```

#>
#> [[2]]
#>      [,1] [,2]
#> [1,]    2    3
#>
#> [[3]]
#>      [,1] [,2]
#> [1,]    4    7
#>
#> [[4]]
#>      [,1] [,2]
#> [1,]    8    8

```

2. With every cluster, the algorithm has to create all possible subclusters joining inicial clusters.

```

clustersList <- initClusters(list)
print(clustersList)
#> [[1]]
#>      [,1] [,2]
#> [1,]    1    1
#>
#> [[2]]
#>      [,1] [,2]
#> [1,]    2    3
#>
#> [[3]]
#>      [,1] [,2]
#> [1,]    4    7
#>
#> [[4]]
#>      [,1] [,2]
#> [1,]    8    8
#>
#> [[5]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    2    3
#>
#> [[6]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    4    7
#>
#> [[7]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    8    8
#>
#> [[8]]
#>      [,1] [,2]
#> [1,]    2    3
#> [2,]    4    7
#>
#> [[9]]

```

71


```

#>      [,1] [,2]
#> [1,]    2    3
#> [2,]    8    8
#>
#> [[10]]
#>      [,1] [,2]
#> [1,]    4    7
#> [2,]    8    8
#>
#> [[11]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    2    3
#> [3,]    4    7
#>
#> [[12]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    2    3
#> [3,]    8    8
#>
#> [[13]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    4    7
#> [3,]    8    8
#>
#> [[14]]
#>      [,1] [,2]
#> [1,]    2    3
#> [2,]    4    7
#> [3,]    8    8
#>
#> [[15]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    2    3
#> [3,]    4    7
#> [4,]    8    8

```

3. We calculate the matrix distance using clusters from second step. We use the distance and approach type that we prefer.

```

matrixDistance <- mdDivisive(clustersList,'MAN','AVG',list)
print(matrixDistance)
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
#> [1,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0
#> [2,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0
#> [3,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0
#> [4,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 10
#> [5,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 10 0
#> [6,] 0.000000 0.000000 0.000000 0 0 0 0 0 7 0 0
#> [7,] 0.000000 0.000000 0.000000 0 0 0 0 7 0 0 0
#> [8,] 0.000000 0.000000 0.000000 0 0 7 7 0 0 0 0

```

```
#> [9,] 0.000000 0.000000 0.000000 0 0 7 0 0 0 0 0
#> [10,] 0.000000 0.000000 0.000000 0 10 0 0 0 0 0 0
#> [11,] 0.000000 0.000000 0.000000 10 0 0 0 0 0 0 0
#> [12,] 0.000000 0.000000 6.666667 0 0 0 0 0 0 0 0
#> [13,] 0.000000 6.666667 0.000000 0 0 0 0 0 0 0 0
#> [14,] 8.666667 0.000000 0.000000 0 0 0 0 0 0 0 0
#> [15,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0
#>      [,12] [,13] [,14] [,15]
#> [1,] 0.000000 0.000000 8.666667 0
#> [2,] 0.000000 6.666667 0.000000 0
#> [3,] 6.666667 0.000000 0.000000 0
#> [4,] 0.000000 0.000000 0.000000 0
#> [5,] 0.000000 0.000000 0.000000 0
#> [6,] 0.000000 0.000000 0.000000 0
#> [7,] 0.000000 0.000000 0.000000 0
#> [8,] 0.000000 0.000000 0.000000 0
#> [9,] 0.000000 0.000000 0.000000 0
#> [10,] 0.000000 0.000000 0.000000 0
#> [11,] 0.000000 0.000000 0.000000 0
#> [12,] 0.000000 0.000000 0.000000 0
#> [13,] 0.000000 0.000000 0.000000 0
#> [14,] 0.000000 0.000000 0.000000 0
#> [15,] 0.000000 0.000000 0.000000 0
```

3. We get the maximal value from the matrix distance, that is, the distance between far away clusters.

```
maxDistance <- maxDistance(matrixDistance)
print(maxDistance)
#> [1] 10
```

4. With the maximal distance, we look for the clusters with this distance separation. We take the clusters that will be divided.

```
dividedClusters <- getClusterDivisive(maxDistance, matrixDistance)
print(dividedClusters)
#> [1] 56
```

5. Two new subclusters will be created from the initial one and added to the solution.

6. We repeat from step 2 to 5 until any cluster could be divided again.

The complete function that implement the algorithm is:

```
divisiveExample <- divisiveHC(dfData[1:4,], 'MAN', 'AVG')
print(divisiveExample)
#> [[1]]
#>   X1 X2
#> 1  1  1
#> 2  2  3
#> 3  4  7
#> 4  8  8
#>
#> [[2]]
#>   X1 X2
#> 1  8  8
#>
#> [[3]]
```

```

#>   X1 X2
#> 1  1  1
#> 2  2  3
#> 3  4  7
#>
#> [[4]]
#>   X1 X2
#> 1  4  7
#>
#> [[5]]
#>   X1 X2
#> 1  1  1
#> 2  2  3
#>
#> [[6]]
#>   X1 X2
#> 1  1  1
#>
#> [[7]]
#>   X1 X2
#> 1  2  3

```

The package uses the same auxiliar functions as the previous to implement the algorithm. These functions are:

- clusterDistanceByApproach
- clusterDistance
- complementaryClusters: checks if the clusters we are going to divide are complementary, that is, every initial cluster is in one or in the other cluster, but never in both. This condition allows to not loose any cluster when the division is done.

```

data <- c(1,2,1,3,1,4,1,5)

components <- toListDivisive(data)

cluster1 <- matrix(c(1,2,1,3),ncol=2,byrow=TRUE)
cluster2 <- matrix(c(1,4,1,5),ncol=2,byrow=TRUE)
cluster3 <- matrix(c(1,6,1,7),ncol=2,byrow=TRUE)

complementaryClusters(components,cluster1,cluster2)
#> [1] TRUE

complementaryClusters(components,cluster1,cluster3)
#> [1] FALSE

```

Its “details” version, explains how the functions checks this condition:

```

complementaryClusters.details(components,cluster1,cluster2)
#>
#> 'complementaryClusters' checks if clusters are complementary.
#>
#> Each element from 'components' list has to be in one cluster, but never included in both.
#>
#> If every element is in one cluster, the function will return 'TRUE',
#>

```

```

#> if not, the result will be 'FALSE'
#>
#> The clusters to be checked are:
#>      [,1] [,2]
#> [1,]    1    2
#> [2,]    1    3
#>      [,1] [,2]
#> [1,]    1    4
#> [2,]    1    5
#>
#> And they have to include:
#> [[1]]
#>      [,1] [,2]
#> [1,]    1    2
#>
#> [[2]]
#>      [,1] [,2]
#> [1,]    1    3
#>
#> [[3]]
#>      [,1] [,2]
#> [1,]    1    4
#>
#> [[4]]
#>      [,1] [,2]
#> [1,]    1    5
#>
#>
#> So, the result is TRUE.
#> [1] TRUE

```

Divisive Hierarchical Clustering .DETAILS

This algorithm explains every function.

1. How clusters are initialized to be used. Initial data could be different R types (vector, matrix or data.frame)

```

# list <- toListDivisive.details(vectorData)

# list <- toListDivisive(matrixData)

list <- toListDivisive(dfData[1:4,])

print(list)
#> [[1]]
#>      [,1] [,2]
#> [1,]    1    1
#>
#> [[2]]
#>      [,1] [,2]
#> [1,]    2    3
#>
#> [[3]]
#>      [,1] [,2]

```

```
#> [1,] 4 7
#>
#> [[4]]
#>      [,1] [,2]
#> [1,] 8 8
```

2. How to create all possible clusters to be divided.

```
clustersList <- initClusters.details(list)
#>
#> 'initClusters' method initializes the clusters used in the divisive algorithm.
#>
#> To know which are the most different clusters, we need to know the distance between
#> every possible clusters that could be created with the initial elements.
#>
#> This step is the most computationally complex, so it will make the algorithm to get the
#> solution with delay, or even, not to find a solution because of the computers capacities.
#>
#> The clusters created using
#> [[1]]
#>      [,1] [,2]
#> [1,] 1 1
#>
#> [[2]]
#>      [,1] [,2]
#> [1,] 2 3
#>
#> [[3]]
#>      [,1] [,2]
#> [1,] 4 7
#>
#> [[4]]
#>      [,1] [,2]
#> [1,] 8 8
#>
#> are:
#> [[1]]
#>      [,1] [,2]
#> [1,] 1 1
#>
#> [[2]]
#>      [,1] [,2]
#> [1,] 2 3
#>
#> [[3]]
#>      [,1] [,2]
#> [1,] 4 7
#>
#> [[4]]
#>      [,1] [,2]
#> [1,] 8 8
#>
#> [[5]]
#>      [,1] [,2]
```

```

#> [1,] 1 1
#> [2,] 2 3
#>
#> [[6]]
#>      [,1] [,2]
#> [1,] 1 1
#> [2,] 4 7
#>
#> [[7]]
#>      [,1] [,2]
#> [1,] 1 1
#> [2,] 8 8
#>
#> [[8]]
#>      [,1] [,2]
#> [1,] 2 3
#> [2,] 4 7
#>
#> [[9]]
#>      [,1] [,2]
#> [1,] 2 3
#> [2,] 8 8
#>
#> [[10]]
#>      [,1] [,2]
#> [1,] 4 7
#> [2,] 8 8
#>
#> [[11]]
#>      [,1] [,2]
#> [1,] 1 1
#> [2,] 2 3
#> [3,] 4 7
#>
#> [[12]]
#>      [,1] [,2]
#> [1,] 1 1
#> [2,] 2 3
#> [3,] 8 8
#>
#> [[13]]
#>      [,1] [,2]
#> [1,] 1 1
#> [2,] 4 7
#> [3,] 8 8
#>
#> [[14]]
#>      [,1] [,2]
#> [1,] 2 3
#> [2,] 4 7
#> [3,] 8 8
#>
#> [[15]]

```

```
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    2    3
#> [3,]    4    7
#> [4,]    8    8
```

3. How the matrix distance is created.

```
matrixDistance <- mdDivisive.details(clustersList,'MAN','AVG',list)
#>
#> 'mdDivisive' creates a matrix distance including every cluster from 'list'.
#>
#>
#> The function checks if the clusters are not valid, if they are the same cluster and
#> if the clusters are not complementary. It will allocate a 0 value if any condition is not 'TRUE'.
#>
#>
#> [[1]]
#>      [,1] [,2]
#> [1,]    1    1
#>
#> [[2]]
#>      [,1] [,2]
#> [1,]    2    3
#>
#> [[3]]
#>      [,1] [,2]
#> [1,]    4    7
#>
#> [[4]]
#>      [,1] [,2]
#> [1,]    8    8
#>
#> [[5]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    2    3
#>
#> [[6]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    4    7
#>
#> [[7]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    8    8
#>
#> [[8]]
#>      [,1] [,2]
#> [1,]    2    3
#> [2,]    4    7
#>
```

```

#> [[9]]
#>      [,1] [,2]
#> [1,]    2    3
#> [2,]    8    8
#>
#> [[10]]
#>      [,1] [,2]
#> [1,]    4    7
#> [2,]    8    8
#>
#> [[11]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    2    3
#> [3,]    4    7
#>
#> [[12]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    2    3
#> [3,]    8    8
#>
#> [[13]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    4    7
#> [3,]    8    8
#>
#> [[14]]
#>      [,1] [,2]
#> [1,]    2    3
#> [2,]    4    7
#> [3,]    8    8
#>
#> [[15]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    2    3
#> [3,]    4    7
#> [4,]    8    8
#>
#> The matrix distance for the list above is:
#>      [,1]      [,2]      [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
#> [1,] 0.000000 0.000000 0.000000    0    0    0    0    0    0    0    0
#> [2,] 0.000000 0.000000 0.000000    0    0    0    0    0    0    0    0
#> [3,] 0.000000 0.000000 0.000000    0    0    0    0    0    0    0    0
#> [4,] 0.000000 0.000000 0.000000    0    0    0    0    0    0    0    10
#> [5,] 0.000000 0.000000 0.000000    0    0    0    0    0    0    10    0
#> [6,] 0.000000 0.000000 0.000000    0    0    0    0    0    7    0    0
#> [7,] 0.000000 0.000000 0.000000    0    0    0    0    7    0    0    0
#> [8,] 0.000000 0.000000 0.000000    0    0    0    7    0    0    0    0
#> [9,] 0.000000 0.000000 0.000000    0    0    7    0    0    0    0    0
#> [10,] 0.000000 0.000000 0.000000    0    10    7    0    0    0    0    0

```



```

#> [11,] 0.000000 0.000000 0.000000 10 0 0 0 0 0 0 0 0
#> [12,] 0.000000 0.000000 6.666667 0 0 0 0 0 0 0 0 0
#> [13,] 0.000000 6.666667 0.000000 0 0 0 0 0 0 0 0 0
#> [14,] 8.666667 0.000000 0.000000 0 0 0 0 0 0 0 0 0
#> [15,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0 0
#>      [,12] [,13] [,14] [,15]
#> [1,] 0.000000 0.000000 8.666667 0
#> [2,] 0.000000 6.666667 0.000000 0
#> [3,] 6.666667 0.000000 0.000000 0
#> [4,] 0.000000 0.000000 0.000000 0
#> [5,] 0.000000 0.000000 0.000000 0
#> [6,] 0.000000 0.000000 0.000000 0
#> [7,] 0.000000 0.000000 0.000000 0
#> [8,] 0.000000 0.000000 0.000000 0
#> [9,] 0.000000 0.000000 0.000000 0
#> [10,] 0.000000 0.000000 0.000000 0
#> [11,] 0.000000 0.000000 0.000000 0
#> [12,] 0.000000 0.000000 0.000000 0
#> [13,] 0.000000 0.000000 0.000000 0
#> [14,] 0.000000 0.000000 0.000000 0
#> [15,] 0.000000 0.000000 0.000000 0

```

4. Choosing the maximal distance, the far away clusters.

```

maxDistance <- maxDistance.details(matrixDistance)
#>
#> 'maxDistance' function gets the minimal value from a matrix.
#>
#> It returns the minimal value avoiding 0 values.
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
#> [1,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0
#> [2,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0
#> [3,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0
#> [4,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 10
#> [5,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 10 0
#> [6,] 0.000000 0.000000 0.000000 0 0 0 0 0 7 0 0
#> [7,] 0.000000 0.000000 0.000000 0 0 0 0 7 0 0 0
#> [8,] 0.000000 0.000000 0.000000 0 0 0 7 0 0 0 0
#> [9,] 0.000000 0.000000 0.000000 0 0 7 0 0 0 0 0
#> [10,] 0.000000 0.000000 0.000000 0 10 0 0 0 0 0 0
#> [11,] 0.000000 0.000000 0.000000 10 0 0 0 0 0 0 0
#> [12,] 0.000000 0.000000 6.666667 0 0 0 0 0 0 0 0
#> [13,] 0.000000 6.666667 0.000000 0 0 0 0 0 0 0 0
#> [14,] 8.666667 0.000000 0.000000 0 0 0 0 0 0 0 0
#> [15,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0
#>      [,12] [,13] [,14] [,15]
#> [1,] 0.000000 0.000000 8.666667 0
#> [2,] 0.000000 6.666667 0.000000 0
#> [3,] 6.666667 0.000000 0.000000 0
#> [4,] 0.000000 0.000000 0.000000 0
#> [5,] 0.000000 0.000000 0.000000 0
#> [6,] 0.000000 0.000000 0.000000 0
#> [7,] 0.000000 0.000000 0.000000 0
#> [8,] 0.000000 0.000000 0.000000 0

```

```
#> [9,] 0.000000 0.000000 0.000000 0
#> [10,] 0.000000 0.000000 0.000000 0
#> [11,] 0.000000 0.000000 0.000000 0
#> [12,] 0.000000 0.000000 0.000000 0
#> [13,] 0.000000 0.000000 0.000000 0
#> [14,] 0.000000 0.000000 0.000000 0
#> [15,] 0.000000 0.000000 0.000000 0
#>
#>
#> 10 is the maximal value of the matrix.
```

5. Using the maximal distance, look for the clusters with this distance.

```
dividedClusters <- getClusterDivisive.details(maxDistance, matrixDistance)
#>
#> 'getCluster' method searches the cluster which have the distance to the target given.
#> Search for 10 in:
#>      [,1]      [,2]      [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
#> [1,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0
#> [2,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0
#> [3,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0
#> [4,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 10
#> [5,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 10 0
#> [6,] 0.000000 0.000000 0.000000 0 0 0 0 0 7 0 0
#> [7,] 0.000000 0.000000 0.000000 0 0 0 0 7 0 0 0
#> [8,] 0.000000 0.000000 0.000000 0 0 0 7 0 0 0 0
#> [9,] 0.000000 0.000000 0.000000 0 0 7 0 0 0 0 0
#> [10,] 0.000000 0.000000 0.000000 0 10 0 0 0 0 0 0
#> [11,] 0.000000 0.000000 0.000000 10 0 0 0 0 0 0 0
#> [12,] 0.000000 0.000000 6.666667 0 0 0 0 0 0 0 0
#> [13,] 0.000000 6.666667 0.000000 0 0 0 0 0 0 0 0
#> [14,] 8.666667 0.000000 0.000000 0 0 0 0 0 0 0 0
#> [15,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0
#>      [,12]      [,13]      [,14] [,15]
#> [1,] 0.000000 0.000000 8.666667 0
#> [2,] 0.000000 6.666667 0.000000 0
#> [3,] 6.666667 0.000000 0.000000 0
#> [4,] 0.000000 0.000000 0.000000 0
#> [5,] 0.000000 0.000000 0.000000 0
#> [6,] 0.000000 0.000000 0.000000 0
#> [7,] 0.000000 0.000000 0.000000 0
#> [8,] 0.000000 0.000000 0.000000 0
#> [9,] 0.000000 0.000000 0.000000 0
#> [10,] 0.000000 0.000000 0.000000 0
#> [11,] 0.000000 0.000000 0.000000 0
#> [12,] 0.000000 0.000000 0.000000 0
#> [13,] 0.000000 0.000000 0.000000 0
#> [14,] 0.000000 0.000000 0.000000 0
#> [15,] 0.000000 0.000000 0.000000 0
#>
#> The cluster with the minimum distance is: 56
```

6. We add the new clusters to the solution and repeat from step 2 to 5 until any cluster could be divided again.

The complete function that explains the algorithm is:

```
divisiveExample <- divisiveHC.details(dfData[1:4,], 'MAN', 'AVG')
#>
#>   Divisive hierarchical clustering is a classification technique that initializes a cluster
#>   with every data.
#>
#>   It calculates the distance between datas depending on the approach type given and
#>   it divides the most different clusters until any cluster can be divided again.
#>
#>   These are the clusters with only one element:
#> [[1]]
#>   X1 X2
#> 1  1  1
#>
#> [[2]]
#>   X1 X2
#> 1  2  3
#>
#> [[3]]
#>   X1 X2
#> 1  4  7
#>
#> [[4]]
#>   X1 X2
#> 1  8  8
#>
#>   And this is the initial cluster with every element:
#>   X1 X2
#> 1  1  1
#> 2  2  3
#> 3  4  7
#> 4  8  8
#>
#>   In each step:
#>   - It calculates a matrix distance between valid clusters depending on the approach
#>   and the distance types.
#>   - It gets the maximal distance value from the matrices of every cluster. We have to
#>   look for the most different clusters available in every cluster.
#>   - It divides the selected cluster in two new and complementary clusters using the maximal
#>   distance between clusters.
#>   - It repeats these steps while there isn't any cluster that can be divided again.
#>
#> -----
#> STEP => 1
#>
#>   The algorithm calculates a matrix distance from every active cluster, but it has to find
#>   the maximal value from all matrix and then chooses which is the selected one.
#>
#>   It gets the maximal value from every matrix and then chooses the maximal between them. So...
#>
#>   Matrix Distance (distance type = MAN, approach type = AVG):
#>           [,1]      [,2]      [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
```

```

#> [1,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0 0
#> [2,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0 0
#> [3,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0 0
#> [4,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0 10
#> [5,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 10 0
#> [6,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 7 0 0
#> [7,] 0.000000 0.000000 0.000000 0 0 0 0 7 0 0 0 0
#> [8,] 0.000000 0.000000 0.000000 0 0 0 7 0 0 0 0 0
#> [9,] 0.000000 0.000000 0.000000 0 0 7 0 0 0 0 0 0
#> [10,] 0.000000 0.000000 0.000000 0 10 0 0 0 0 0 0 0
#> [11,] 0.000000 0.000000 0.000000 10 0 0 0 0 0 0 0 0
#> [12,] 0.000000 0.000000 6.666667 0 0 0 0 0 0 0 0 0
#> [13,] 0.000000 6.666667 0.000000 0 0 0 0 0 0 0 0 0
#> [14,] 8.666667 0.000000 0.000000 0 0 0 0 0 0 0 0 0
#> [15,] 0.000000 0.000000 0.000000 0 0 0 0 0 0 0 0 0
#>      [,12]      [,13]      [,14] [,15]
#> [1,] 0.000000 0.000000 8.666667 0
#> [2,] 0.000000 6.666667 0.000000 0
#> [3,] 6.666667 0.000000 0.000000 0
#> [4,] 0.000000 0.000000 0.000000 0
#> [5,] 0.000000 0.000000 0.000000 0
#> [6,] 0.000000 0.000000 0.000000 0
#> [7,] 0.000000 0.000000 0.000000 0
#> [8,] 0.000000 0.000000 0.000000 0
#> [9,] 0.000000 0.000000 0.000000 0
#> [10,] 0.000000 0.000000 0.000000 0
#> [11,] 0.000000 0.000000 0.000000 0
#> [12,] 0.000000 0.000000 0.000000 0
#> [13,] 0.000000 0.000000 0.000000 0
#> [14,] 0.000000 0.000000 0.000000 0
#> [15,] 0.000000 0.000000 0.000000 0
#>
#> The maximal distance is: 10
#>
#> The most distant clusters are:
#> X1 X2
#> 1 8 8
#>
#> X1 X2
#> 1 1 1
#> 2 2 3
#> 3 4 7
#>
#> The divided clusters are added to the solution.
#>
#> - The second divided cluster is added to the active clusters list because it can be divided again.
#>
#>
#> If the divided clusters can't be divided again, then they don't be added to the active clusters.
#>
#> -----
#> STEP => 2
#>

```

```

#> The algorithm calculates a matrix distance from every active cluster, but it has to find
#> the maximal value from all matrix and then chooses which is the selected one.
#>
#> It gets the maximal value from every matrix and then chooses the maximal between them. So...
#>
#> Matrix Distance (distance type = MAN, approach type = AVG):
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
#> [1,]    0  0.0  0.0  0.0  0.0    6    0
#> [2,]    0  0.0  0.0  0.0  4.5    0    0
#> [3,]    0  0.0  0.0  7.5  0.0    0    0
#> [4,]    0  0.0  7.5  0.0  0.0    0    0
#> [5,]    0  4.5  0.0  0.0  0.0    0    0
#> [6,]    6  0.0  0.0  0.0  0.0    0    0
#> [7,]    0  0.0  0.0  0.0  0.0    0    0
#>
#> The maximal distance is: 7.5
#>
#> The most distant clusters are:
#>   X1 X2
#> 1  4  7
#>
#>   X1 X2
#> 1  1  1
#> 2  2  3
#>
#> The divided clusters are added to the solution.
#>
#> - The second divided cluster is added to the active clusters list because it can be divided again.
#>
#>
#> If the divided clusters can't be divided again, then they don't be added to the active clusters.
#>
#> -----
#> STEP => 3
#>
#> The algorithm calculates a matrix distance from every active cluster, but it has to find
#> the maximal value from all matrix and then chooses which is the selected one.
#>
#> It gets the maximal value from every matrix and then chooses the maximal between them. So...
#>
#> Matrix Distance (distance type = MAN, approach type = AVG):
#>      [,1] [,2] [,3]
#> [1,]    0    3    0
#> [2,]    3    0    0
#> [3,]    0    0    0
#>
#> The maximal distance is: 3
#>
#> The most distant clusters are:
#>   X1 X2
#> 1  1  1
#>
#>   X1 X2

```

```

#> 1 2 3
#>
#> The divided clusters are added to the solution.
#>
#> If the divided clusters can't be divided again, then they don't be added to the active clusters.
#>
#> This loop has been repeated until there aren't any active cluster, that is, any cluster
#> can be divided again.
print(divisiveExample)
#> [[1]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    2    3
#> [3,]    4    7
#> [4,]    8    8
#>
#> [[2]]
#>      [,1] [,2]
#> [1,]    8    8
#>
#> [[3]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    2    3
#> [3,]    4    7
#>
#> [[4]]
#>      [,1] [,2]
#> [1,]    4    7
#>
#> [[5]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    2    3
#>
#> [[6]]
#>      [,1] [,2]
#> [1,]    1    1
#>
#> [[7]]
#>      [,1] [,2]
#> [1,]    2    3

```

Correlative Hierarchical Clustering

This example shows how the algorithm works step by step. 1. Input data is initialized creating a cluster with each data frame row.

```

initData <- initData(cDataFrame)
print(initData)
#> [[1]]
#>      [,1] [,2] [,3]
#> [1,]    2    4    4
#>

```

```

#> [[2]]
#>      [,1] [,2] [,3]
#> [1,]    2    3    5
#>
#> [[3]]
#>      [,1] [,2] [,3]
#> [1,]    1    1    2
#>
#> [[4]]
#>      [,1] [,2] [,3]
#> [1,]    2    5    5
#>
#> [[5]]
#>      [,1] [,2] [,3]
#> [1,]    1    0    1
#>
#> [[6]]
#>      [,1] [,2] [,3]
#> [1,]    1    2    1
#>
#> [[7]]
#>      [,1] [,2] [,3]
#> [1,]    2    4    5
#>
#> [[8]]
#>      [,1] [,2] [,3]
#> [1,]    1    2    1

```

2. The algorithm checks if the input target is acceptable, if not, it initializes the target.

```

target <- c(1,2,3)

initTarget <- initTarget(target,cDataFrame)
print(initTarget)
#>      [,1] [,2] [,3]
#> [1,]    1    2    3

```

3. If users want it, the algorithm will normalize weight's values.

```

weight <- c(5,7,6)

weights <- normalizeWeight(TRUE,weight,cDataFrame)
print(weights)
#> [1] 0.2777778 0.3888889 0.3333333

```

4. It calculates distances between clusters applying weights and distance definition given.

```

cluster1 <- matrix(c(1,2,3),ncol=3)
cluster2 <- matrix(c(2,5,8),ncol=3)

weight <- c(3,7,4)

distance <- distances(cluster1,cluster2,'CHE',weight)
print(distance)
#> [1] 21

```

Finally, the complete algorithm sorts the distances and sort the clusters aswell. It presents the solution as a sorted clusters list, with the distances or using a dendrogram.

```
target <- c(5,5,1)

weight <- c(3,7,5)

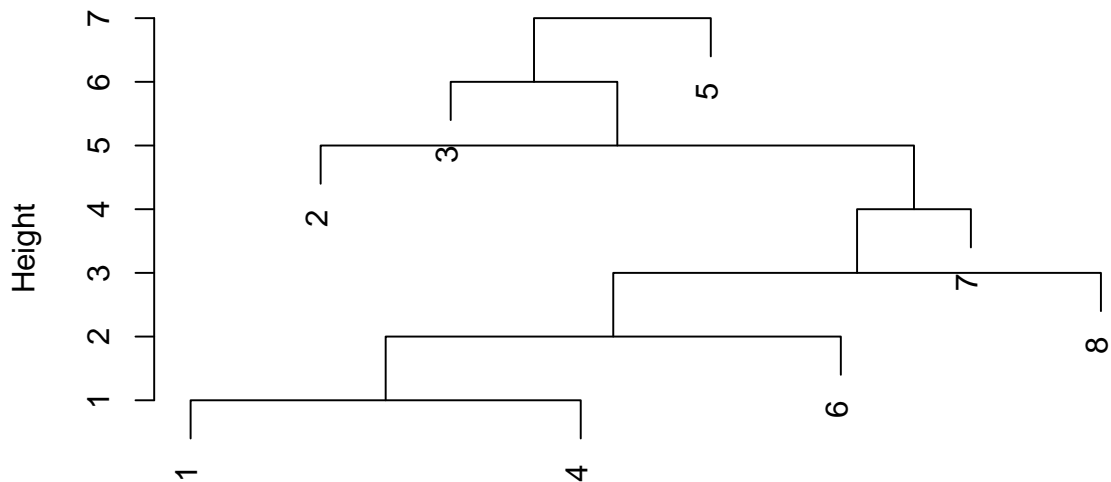
correlation <- correlationHC(cDataFrame, target, weight)

print(correlation$sortedValues)
#>   cluster X1 X2 X3
#> 1         1  2  4  4
#> 2         4  2  5  5
#> 3         6  1  2  1
#> 4         8  1  2  1
#> 5         7  2  4  5
#> 6         2  2  3  5
#> 7         3  1  1  2
#> 8         5  1  0  1

print(correlation$distances)
#>   cluster sortedDistances
#> 1         1           2.294922
#> 2         4           2.670830
#> 3         6           2.720294
#> 4         8           2.720294
#> 5         7           2.756810
#> 6         2           3.000000
#> 7         3           3.316625
#> 8         5           3.855732

plot(correlation$dendrogram)
```


Cluster Dendrogram



Correlative Hierarchical Clustering .DETAILS

This example shows how the algorithm works step by step.

1. How input data is initialized.

```
initData <- initData.details(cDataFrame)
#>
#> This function initializes the input data creating a cluster with each row of the data frame.
#>
#> It gets this data from the user:
#>  X1 X2 X3
#> 1  2  4  4
#> 2  2  3  5
#> 3  1  1  2
#> 4  2  5  5
#> 5  1  0  1
#> 6  1  2  1
#> 7  2  4  5
#> 8  1  2  1
#>
#> Each cluster will be a matrix with a row and the same columns as the initial data frame.
#>
#> Initialized data will be:
#> [[1]]
#>      [,1] [,2] [,3]
#> [1,]    2    4    4
```

```

#>
#> [[2]]
#>      [,1] [,2] [,3]
#> [1,]    2    3    5
#>
#> [[3]]
#>      [,1] [,2] [,3]
#> [1,]    1    1    2
#>
#> [[4]]
#>      [,1] [,2] [,3]
#> [1,]    2    5    5
#>
#> [[5]]
#>      [,1] [,2] [,3]
#> [1,]    1    0    1
#>
#> [[6]]
#>      [,1] [,2] [,3]
#> [1,]    1    2    1
#>
#> [[7]]
#>      [,1] [,2] [,3]
#> [1,]    2    4    5
#>
#> [[8]]
#>      [,1] [,2] [,3]
#> [1,]    1    2    1

```

2. How the algorithm checks if the input target is acceptable, and if not, how it initializes the target.

```

targetValid <- c(1,2,3)

targetInvalid <- c(1,2)

initTarget <- initTarget.details(targetValid,cDataFrame)
#>
#> This function initializes the target and checks if it is a valid target.
#>
#> It gets this target from the user:
#> [1] 1 2 3
#>
#> After transforming the target into matrix, it checks if it is acceptable
#>
#> If the target has the same columns as main data and only one row, it is a valid target!
#>
#> Target used will be:
#>      [,1] [,2] [,3]
#> [1,]    1    2    3

initTarget <- initTarget.details(targetInvalid,cDataFrame)
#>
#> This function initializes the target and checks if it is a valid target.
#>

```

```

#> It gets this target from the user:
#> [1] 1 2
#>
#> After transforming the target into matrix, it checks if it is acceptable
#>
#> If the target does not have the same columns as main data or more than one row,
#> it is not a valid target!
#>
#> The function will initialize as a '0's matrix.
#>
#> Target used will be:
#>      [,1] [,2] [,3]
#> [1,]    0    0    0

```

3. How the normalization process is done.

```

weight <- c(5,7,6)

weights <- normalizeWeight.details(TRUE,weight,cDataFrame)
#>
#> This function normalizes weight values.
#>
#> This are the initial weights:
#> 5
#> 7
#> 6
#>
#> It checks if there is a weights vector. If not, it creates a vector with 3 '1's.
#>
#> Due to the fact that 'normalize' = TRUE, weight vector changes every weight as the
#> initial value divided between the total sum of the vector.
#>
#> FinalWeight[i] = WeightValue[i]/TotalSum
#>
#> These are the new weights:
#> 0.2777777777777778
#> 0.3888888888888889
#> 0.3333333333333333

weights <- normalizeWeight.details(FALSE,weight,cDataFrame)
#>
#> This function normalizes weight values.
#>
#> This are the initial weights:
#> 5
#> 7
#> 6
#>
#> It checks if there is a weights vector. If not, it creates a vector with 3 '1's.
#>
#> Due to the fact that 'normalize' = FALSE, weight vector does not change.
#>
#> These are the new weights:
#> 5

```

```

#> 7
#> 6

weights <- normalizeWeight.details(FALSE,NULL,cDataFrame)
#>
#> This function normalizes weight values.
#>
#> It checks if there is a weights vector. If not, it creates a vector with 3 '1's.
#>
#> Due to the fact that 'normalize' = FALSE, weight vector does not change.
#>
#> These are the new weights:
#> 1
#> 1
#> 1

```

4. How it calculates distances between clusters applying weights and distance definition given.

```

cluster1 <- matrix(c(1,2,3),ncol=3)
cluster2 <- matrix(c(2,5,8),ncol=3)

weight <- c(3,7,4)

distance <- distances.details(cluster1,cluster2,'CHE',weight)
#>
#> This function calculates CHE distance applying weights.
#>
#> It calculates the distance between:.
#> X1 X2 X3
#> 1 1 2 3
#>
#>
#> X1 X2 X3
#> 1 2 5 8
#>
#> Applying these weights:
#> [1] 3 7 4
#>
#> The distance value is: 21.

```

The complete function that explains the algorithm is:

```

target <- c(5,5,1)

weight <- c(3,7,5)

correlation <- correlationHC.details(cDataFrame, target, weight)
#> Correlation hierarchical function is a classification technique that initializes a cluster for each
#>
#> It calculates the distance between clusters and a target given depending on the distance type.
#>
#> The function applies weights to each property from main data to get weighted results.
#>
#> Due to normalized = TRUE, the initial weights change to a [0,1] values.
#>

```

```

#> These are the weight to be used:
#> [1] 0.2000000 0.4666667 0.3333333
#>
#> Initialized data are (more information about how to initialize data in 'initData.details'):
#> [[1]]
#>      [,1] [,2] [,3]
#> [1,]    2    4    4
#>
#> [[2]]
#>      [,1] [,2] [,3]
#> [1,]    2    3    5
#>
#> [[3]]
#>      [,1] [,2] [,3]
#> [1,]    1    1    2
#>
#> [[4]]
#>      [,1] [,2] [,3]
#> [1,]    2    5    5
#>
#> [[5]]
#>      [,1] [,2] [,3]
#> [1,]    1    0    1
#>
#> [[6]]
#>      [,1] [,2] [,3]
#> [1,]    1    2    1
#>
#> [[7]]
#>      [,1] [,2] [,3]
#> [1,]    2    4    5
#>
#> [[8]]
#>      [,1] [,2] [,3]
#> [1,]    1    2    1
#>
#> Initialized target is (more information about how to initialize data in 'initTarget.details'):
#>      [,1] [,2] [,3]
#> [1,]    5    5    1
#>
#> The function calculates the distances between each cluster and the target. It applies
#> weights and uses EUC distance type.
#>
#>
#> The calculated distances are:
#> [1] 2.294922 3.000000 3.316625 2.670830 3.855732 2.720294 2.756810 2.720294
#>
#> The previous distances sorted are:
#>
#> [1] 2.294922 2.670830 2.720294 2.720294 2.756810 3.000000 3.316625 3.855732
#>
#> Then, using sorted distances, the function order the clusters.
#>

```

```

#>
#> Finally, the sorted distances are:
#> cluster sortedDistances
#> 1      1      2.294922
#> 2      4      2.670830
#> 3      6      2.720294
#> 4      8      2.720294
#> 5      7      2.756810
#> 6      2      3.000000
#> 7      3      3.316625
#> 8      5      3.855732
#>
#> The sorted clusters are:
#>
#> cluster X1 X2 X3
#> 1      1 2 4 4
#> 2      4 2 5 5
#> 3      6 1 2 1
#> 4      8 1 2 1
#> 5      7 2 4 5
#> 6      2 2 3 5
#> 7      3 1 1 2
#> 8      5 1 0 1
#>
#> And the final dendrogram is on the image.
#>

```

```
print(correlation$sortedValues)
```

```

#> cluster X1 X2 X3
#> 1      1 2 4 4
#> 2      4 2 5 5
#> 3      6 1 2 1
#> 4      8 1 2 1
#> 5      7 2 4 5
#> 6      2 2 3 5
#> 7      3 1 1 2
#> 8      5 1 0 1

```

```
print(correlation$distances)
```

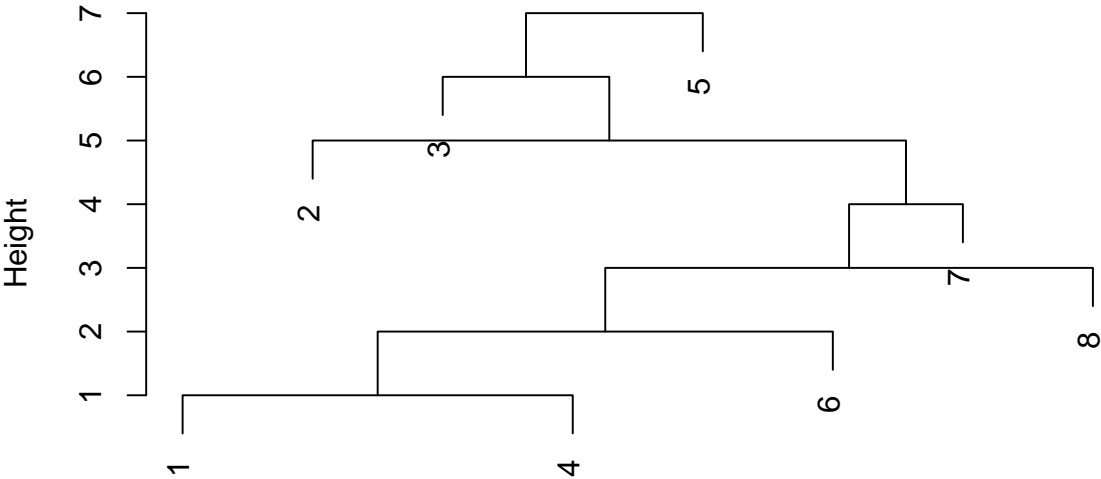
```

#> cluster sortedDistances
#> 1      1      2.294922
#> 2      4      2.670830
#> 3      6      2.720294
#> 4      8      2.720294
#> 5      7      2.756810
#> 6      2      3.000000
#> 7      3      3.316625
#> 8      5      3.855732

```

```
plot(correlation$dendrogram)
```

Cluster Dendrogram



8 MANUAL DE USUARIO

Documentación de funciones

MANUAL DE USUARIO

Cómo usar el paquete

Package ‘LearnClust’

September 30, 2020

Type Package

Date 2020-09-04

Title Learning Hierarchical Clustering Algorithms

Version 1.0

Author Roberto Alcantara [aut, cre],
Juan Jose Cuadrado [aut],
Universidad de Alcala de Henares [aut]

Maintainer Roberto Alcantara <roberto.alcantara@edu.uah.es>

Description Classical hierarchical clustering algorithms, agglomerative and divisive clustering. Algorithms are implemented as a theoretical way, step by step.
It includes some detailed functions that explain each step. Every function allows options to get different results using different techniques.
The package explains non expert users how hierarchical clustering algorithms work.

License Unlimited

Depends magick

Suggests knitr, rmarkdown

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2020-09-30 09:30:03 UTC

R topics documented:

agglomerativeHC	3
agglomerativeHC.details	4
canberradistance	5
canberradistance.details	6

canberradistanceW	7
canberradistanceW.details	8
chebyshevDistance	9
chebyshevDistance.details	10
chebyshevDistanceW	11
chebyshevDistanceW.details	12
clusterDistance	13
clusterDistance.details	14
clusterDistanceByApproach	15
clusterDistanceByApproach.details	16
complementaryClusters	17
complementaryClusters.details	18
correlationHC	19
correlationHC.details	21
distances	22
distances.details	23
divisiveHC	24
divisiveHC.details	26
edistance	27
edistance.details	28
edistanceW	29
edistanceW.details	30
getCluster	31
getCluster.details	32
getClusterDivisive	33
getClusterDivisive.details	34
initClusters	35
initClusters.details	36
initData	37
initData.details	38
initImages	39
initTarget	39
initTarget.details	40
matrixDistance	41
maxDistance	42
maxDistance.details	43
mdAgglomerative	44
mdAgglomerative.details	45
mdDivisive	46
mdDivisive.details	47
mdistance	48
mdistance.details	49
mdistanceW	50
mdistanceW.details	51
minDistance	52
minDistance.details	53
newCluster	54
newCluster.details	55

<i>agglomerativeHC</i>	3
normalizeWeight	56
normalizeWeight.details	57
octileDistance	58
octileDistance.details	59
octileDistanceW	60
octileDistanceW.details	61
toList	62
toList.details	63
toListDivisive	64
toListDivisive.details	65
usefulClusters	66
Index	68

<i>agglomerativeHC</i>	<i>To execute agglomerative hierarchical clusterization algorithm by distance and approach.</i>
------------------------	---

Description

To execute complete agglomerative hierarchical clusterization algorithm choosing distance and approach type.

Usage

```
agglomerativeHC(data, distance, approach)
```

Arguments

data	could be a numeric vector, a matrix or a numeric data frame. It will be transformed into matrix and list to be used.
distance	is a string. It chooses the distance to use.
approach	is a string. It chooses the approach to use.

Details

This function is the main part of the agglomerative hierarchical clusterization method. It executes the theoretical algorithm step by step.

- 1 - The function transforms data in useful object to be used.
- 2 - It creates the clusters.
- 3 - It calculates a matrix distances with the clusters created applying distance and approach given.
- 4 - It chooses the distance value and gets the clusters.
- 5 - It groups the clusters in a new one and updates clusters list.
- 6 - It repeats these steps until an unique cluster exists.

Value

R object with a dendrogram, the grouped clusters and the list with every cluster.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
a <- c(1,2,1,3,1,4,1,5,1,6)

matrixA <- matrix(a,ncol=2)

dataFrameA <- data.frame(matrixA)

agglomerativeHC(a, 'EUC', 'MAX')

agglomerativeHC(matrixA, 'MAN', 'AVG')

agglomerativeHC(dataFrameA, 'CAN', 'MIN')
```

agglomerativeHC.details

To explain agglomerative hierarchical clusterization algorithm by distance and approach.

Description

To explain the complete agglomerative hierarchical clusterization algorithm choosing distance and approach type.

Usage

```
agglomerativeHC.details(data, distance, approach)
```

Arguments

data	could be a numeric vector, a matrix or a numeric data frame. It will be transformed into matrix and list to be used.
distance	is a string. It chooses the distance to use.
approach	is a string. It chooses the approach to use.

Details

This function is the main part of the agglomerative hierarchical clusterization method. It explains the theoretical algorithm step by step.

- 1 - The function transforms data into useful object to be used.
- 2 - It creates the clusters.
- 3 - It calculates a matrix distance with the clusters created by applying the distance and the approach given.
- 4 - It chooses the distance value and gets the clusters.
- 5 - It groups the clusters in a new one and updates clusters list.
- 6 - It repeats these steps until an unique cluster exists.

Value

agglomerative algorithm explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
a <- c(1,2,1,3,1,4,1,5,1,6)

matrixA <- matrix(a,ncol=2)

dataFrameA <- data.frame(matrixA)

agglomerativeHC.details(a,'EUC','MAX')

agglomerativeHC.details(matrixA,'MAN','AVG')

agglomerativeHC.details(dataFrameA,'CAN','MIN')
```

canberradistance

To calculate the Canberra distance.

Description

To calculate the Canberra distance of two clusters.

Usage

```
canberradistance(x, y)
```

Arguments

`x` is a numeric vector or a matrix. It represents the values of a cluster.
`y` is a numeric vector or a matrix. It represents the values of a cluster.

Details

This function is part of the hierarchical clusterization method. The function calculates the Canberra distance value from `x` and `y`.

Value

canberra distance value.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
x <- c(1,2)
y <- c(1,3)

cluster1 <- matrix(x,ncol=2)
cluster2 <- matrix(y,ncol=2)

canberradistance(x,y)

canberradistance(cluster1,cluster2)
```

canberradistance.details

To show the formula and to return the Canberra distance.

Description

To show the formula and to return the Canberra distance of two clusters.

Usage

```
canberradistance.details(x, y)
```

Arguments

`x` is a numeric vector or a matrix. It represents the values of a cluster.
`y` is a numeric vector or a matrix. It represents the values of a cluster.

Details

This function is part of the hierarchical clusterization method. The function calculates the Canberra distance value from `x` and `y`.

Value

canberra distance value with its formula.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
x <- c(1,2)
y <- c(1,3)

cluster1 <- matrix(x,ncol=2)
cluster2 <- matrix(y,ncol=2)

canberradistance.details(x,y)

canberradistance.details(cluster1,cluster2)
```

canberradistanceW	<i>To calculate the Canberra distance applying weights.</i>
-------------------	---

Description

To calculate the Canberra distance between clusters applying weights given.

Usage

```
canberradistanceW(cluster1, cluster2, weight)
```

Arguments

cluster1 is a cluster.
 cluster2 is a cluster.
 weight is a numeric vector.

Details

The function calculates the Canberra distance value from cluster1 and cluster2, applying weights to the cluster's components.

Value

canberra distance applying weights value.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
cluster1 <- matrix(c(1,2),ncol=2)
cluster2 <- matrix(c(1,3),ncol=2)

weight1 <- c(0.4,0.6)
weight2 <- c(2,12)

canberradistanceW(cluster1,cluster2,weight1)

canberradistanceW(cluster1,cluster2,weight2)
```

canberradistanceW.details

To calculate the Canberra distance applying weights .

Description

To explain how to calculate the Canberra distance between clusters applying weights given.

Usage

```
canberradistanceW.details(cluster1, cluster2, weight)
```


Arguments

cluster1 is a cluster.
 cluster2 is a cluster.
 weight is a numeric vector.

Details

The function calculates the Canberra distance value from cluster1 and cluster2, applying weights to the cluster's components.

Value

canberra distance applying weights value. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
cluster1 <- matrix(c(1,2),ncol=2)
cluster2 <- matrix(c(1,3),ncol=2)

weight1 <- c(0.4,0.6)
weight2 <- c(2,12)

canberradistanceW.details(cluster1,cluster2,weight1)
canberradistanceW.details(cluster1,cluster2,weight2)
```

chebyshevDistance *To calculate the Chebyshev distance.*

Description

To calculate the Chebyshev distance of two clusters.

Usage

```
chebyshevDistance(x, y)
```

Arguments

`x` is a numeric vector or a matrix. It represents the values of a cluster.
`y` is a numeric vector or a matrix. It represents the values of a cluster.

Details

This function is part of the hierarchical clusterization method. The function calculates the Chebyshev distance value from `x` and `y`.

Value

Chebyshev distance value.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
x <- c(1,2)
y <- c(1,3)

cluster1 <- matrix(x,ncol=2)
cluster2 <- matrix(y,ncol=2)

chebyshevDistance(x,y)

chebyshevDistance(cluster1,cluster2)
```

`chebyshevDistance.details`

To show the formula of the Chebyshev distance.

Description

To show the formula of the Chebyshev distance of two clusters.

Usage

```
chebyshevDistance.details(x, y)
```

Arguments

`x` is a numeric vector or a matrix. It represents the values of a cluster.
`y` is a numeric vector or a matrix. It represents the values of a cluster.

Details

This function is part of the hierarchical clusterization method. The function calculates the Chebyshev distance value from `x` and `y`.

Value

Chebyshev distance value and formula.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
x <- c(1,2)
y <- c(1,3)

cluster1 <- matrix(x,ncol=2)
cluster2 <- matrix(y,ncol=2)

chebyshevDistance.details(x,y)

chebyshevDistance.details(cluster1,cluster2)
```

chebyshevDistanceW	<i>To calculate the Chebyshev distance applying weights.</i>
--------------------	--

Description

To calculate the Chebyshev distance between clusters applying weights given.

Usage

```
chebyshevDistanceW(cluster1, cluster2, weight)
```

Arguments

cluster1 is a cluster.
 cluster2 is a cluster.
 weight is a numeric vector.

Details

The function calculates the Chebyshev distance value from cluster1 and cluster2, applying weights to the cluster's components.

Value

Chebyshev distance applying weights value.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
cluster1 <- matrix(c(1,2),ncol=2)
cluster2 <- matrix(c(1,3),ncol=2)

weight1 <- c(0.4,0.6)
weight2 <- c(2,12)

chebyshevDistanceW(cluster1,cluster2,weight1)

chebyshevDistanceW(cluster1,cluster2,weight2)
```

chebyshevDistanceW.details

To calculate the Chebyshev distance applying weights.

Description

To explain how to calculate the Chebyshev distance between clusters applying weights given.

Usage

```
chebyshevDistanceW.details(cluster1, cluster2, weight)
```

Arguments

cluster1 is a cluster.
 cluster2 is a cluster.
 weight is a numeric vector.

Details

The function calculates the Chebyshev distance value from cluster1 and cluster2, applying weights to the cluster's components.

Value

Chebyshev distance applying weights value. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
cluster1 <- matrix(c(1,2),ncol=2)
cluster2 <- matrix(c(1,3),ncol=2)

weight1 <- c(0.4,0.6)
weight2 <- c(2,12)

chebyshevDistanceW.details(cluster1,cluster2,weight1)
chebyshevDistanceW.details(cluster1,cluster2,weight2)
```

clusterDistance	<i>To calculate the distance between clusters.</i>
-----------------	--

Description

To calculate the distance between clusters depending on the approach and distance type.

Usage

```
clusterDistance(cluster1, cluster2, approach, distance)
```

Arguments

cluster1	is a matrix
cluster2	is a matrix
approach	is a string. Type of function to apply.
distance	is a string. Type of distance to use.

Details

This function is part of the hierarchical clusterization method. The function calculates the final distance between cluster1 and cluster2 applying the approach definition, using the distance type given.

approach indicates the algorithm used to get the value. distance indicates the distance used to get the value. Possible values: 'MAX', 'MIN', 'AVG'.

Value

Distance between clusters.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
cluster1 <- matrix(c(1,2),ncol=2)
cluster2 <- matrix(c(1,4),ncol=2)

clusterDistance.details(cluster1,cluster2,'AVG','MAN')

clusterDistance.details(cluster1,cluster2,'MAX','OCT')
```

clusterDistance.details

To explain how to calculate the distance between clusters.

Description

To explain how to calculate the distance between clusters depending on the approach and distance type.

Usage

```
clusterDistance.details(cluster1, cluster2, approach, distance)
```

Arguments

cluster1 is a matrix
 cluster2 is a matrix
 approach is a string. Type of function to apply.
 distance is a string. Type of distance to use.

Details

This function is part of the hierarchical clusterization method. The function explains how to calculate the final distance between cluster1 and cluster2 applying the approach definition, using the distance type given.

approach indicates the algorithm used to get the value. distance indicates the distance used to get the value. Possible values: 'MAX', 'MIN', 'AVG'.

Value

Distance between clusters. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
cluster1 <- matrix(c(1,2),ncol=2)
cluster2 <- matrix(c(1,4),ncol=2)

clusterDistance.details(cluster1,cluster2,'AVG','MAN')

clusterDistance.details(cluster1,cluster2,'MAX','OCT')
```

clusterDistanceByApproach

To calculate the distance by approach option.

Description

To calculate the distance depending on option given.

Usage

```
clusterDistanceByApproach(distances, approach)
```

Arguments

distances is a numeric vector.
 approach is a string. Type of function to apply.

Details

This function is part of the hierarchical clusterization method. The function calculates the distance value from distances.

approach indicates the algorithm used to get the value. Possible values: 'MAX', 'MIN', 'AVG'.

Value

max, min or average from a vector.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
distances1 <- c(4,14,24,34)

distances2 <- c(1:10)

clusterDistanceByApproach(distances1, 'MAX')

clusterDistanceByApproach(distances2, 'MIN')
```

clusterDistanceByApproach.details

To explain how to calculate the distance by approach option.

Description

To explain how to calculate the distance depending on option given.

Usage

```
clusterDistanceByApproach.details(distances, approach)
```

Arguments

distances is a numeric vector.
 approach is a string. Type of function to apply.

Details

This function is part of the hierarchical clusterization method. The function explains how to calculate the distance value from distances.

approach indicates the algorithm used to get the value. Possible values: 'MAX', 'MIN', 'AVG'.

Value

max, min or average from a vector.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
distances1 <- c(4,14,24,34)
distances2 <- c(1:10)
clusterDistanceByApproach(distances1, 'MAX')
clusterDistanceByApproach(distances2, 'MIN')
```

complementaryClusters *To check if two clusters are complementary*

Description

To check if two clusters include every element but without repeating anyone.

Usage

```
complementaryClusters(components, cluster1, cluster2)
```

Arguments

components	is an elements list. It contains every component that has to be in one cluster or in the other one. But each element can only be included in one cluster.
cluster1	is a cluster (matrix).
cluster2	is a cluster (matrix).

Details

This function checks if the cluster that will be divided contains the simple elements that they have to include. They have to contain every element, but anyone should be duplicated.

The function will return a boolean value.

Value

Boolean value.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
data <- c(1,2,1,3,1,4,1,5)

components <- toListDivisive(data)

cluster1 <- matrix(c(1,2,1,3),ncol=2)
cluster2 <- matrix(c(1,4,1,5),ncol=2)
cluster3 <- matrix(c(1,6,1,7),ncol=2)

complementaryClusters(components,cluster1,cluster2) #TRUE

complementaryClusters(components,cluster3,cluster2) #FALSE
```

complementaryClusters.details

To explain how and why two clusters are complementary.

Description

To explain how and why two clusters include every element but without repeating anyone.

Usage

```
complementaryClusters.details(components, cluster1, cluster2)
```

Arguments

components	is an elements list. It contains every component that has to be in one cluster or in the other one. But each element can only be included in one cluster.
cluster1	is a cluster (matrix).
cluster2	is a cluster (matrix).

Details

This function checks if the cluster that will be divided contains the simple elements that they have to include. They have to contain every element, but anyone should be duplicated.

The function will return a boolean value.

Value

Boolean value. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
data <- c(1,2,1,3,1,4,1,5)

components <- toListDivisive(data)

cluster1 <- matrix(c(1,2,1,3),ncol=2)
cluster2 <- matrix(c(1,4,1,5),ncol=2)
cluster3 <- matrix(c(1,6,1,7),ncol=2)

complementaryClusters.details(components,cluster1,cluster2) #TRUE

complementaryClusters.details(components,cluster3,cluster2) #FALSE
```

correlationHC

To execute hierarchical correlation algorithm.

Description

To execute hierarchical correlation algorithm applying weights, distance types, ...

Usage

```
correlationHC(
  data,
  target = NULL,
  weight = c(),
  distance = "EUC",
  normalize = TRUE,
  labels = NULL
)
```

Arguments

<code>data</code>	is a data frame with the main data.
<code>target</code>	is a data frame , a numeric vector or a matrix. Default value = NULL.
<code>weight</code>	is a numeric vector. Default value = empty vector.
<code>distance</code>	is a string. The distance type. Default value = Euclidean distance.
<code>normalize</code>	is a boolean parameter. If the user wants to normalize weights. Default value = TRUE.
<code>labels</code>	is a string vector. For the graphical solution. Default value = NULL.

Details

This function execute the complete hierarchical correlation method.

- 1 - The function transforms data in useful object to be used.
- 2 - It creates the clusters.
- 3 - It calculates the distance from the target to every cluster applying distance type given.
- 4 - It orders the distance in increasing way.
- 5 - It orders the clusters according to their distance from the previous step
- 6 - It shows the clusters sorted and the distance used.

Value

R object with a dendrogram, the sorted distances and the list with every cluster.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
data <- matrix(c(1,2,1,4,5,1,8,2,9,6,3,5,8,5,4),ncol= 3)

dataFrame <- data.frame(data)

target1 <- c(1,2,3)

target2 <- dataFrame[1,]

weight1 <- c(1,6,3)

weight2 <- c(0.1,0.6,0.3)

correlationHC(dataFrame, target1)
```

```
correlationHC(dataFrame, target1, weight1)

correlationHC(dataFrame, target1, weight1, normalize = FALSE)

correlationHC(dataFrame, target1, weight2, 'CAN', FALSE)
```

correlationHC.details *To explain how hierarchical correlation algorithm works.*

Description

To explain how the hierarchical correlation algorithm works.

Usage

```
correlationHC.details(
  data,
  target = NULL,
  weight = c(),
  distance = "EUC",
  normalize = TRUE,
  labels = NULL
)
```

Arguments

data	is a data frame with the main data.
target	is a data frame , a numeric vector or a matrix. Default value = NULL.
weight	is a numeric vector. Default value = empty vector.
distance	is a string. The distance type. Default value = Euclidean distance.
normalize	is a boolean parameter. If the user wants to normalize weights. Default value = TRUE.
labels	is a string vector. For the graphical solution. Default value = NULL.

Details

This function explains the complete hierarchical correlation method. It explains the theoretical algorithm step by step.

- 1 - The function transforms data in useful object to be used.
- 2 - It creates the clusters.
- 3 - It calculates the distance from the target to every cluster applying the distance type given.
- 4 - It orders the distance in an increasing way.
- 5 - It orders the clusters according to their distance from the previous step
- 6 - It shows the clusters sorted and the distance used.

Value

R object with a dendrogram, the sorted distances and the list with every cluster. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
data <- matrix(c(1,2,1,4,5,1,8,2,9,6,3,5,8,5,4),ncol= 3)
dataFrame <- data.frame(data)
target1 <- c(1,2,3)
target2 <- dataFrame[1,]
weight1 <- c(1,6,3)
weight2 <- c(0.1,0.6,0.3)
correlationHC.details(dataFrame, target1)
correlationHC.details(dataFrame, target1, weight1)
correlationHC.details(dataFrame, target1, weight1, normalize = FALSE)
correlationHC.details(dataFrame, target1, weight2, 'CAN', FALSE)
```

distances

To calculate distances applying weights.

Description

To calculate distances between two clusters applying weights depending on the distance type.

Usage

```
distances(cluster1, cluster2, distance, weight)
```

Arguments

cluster1 is a matrix.
 cluster2 is a matrix.
 distance is a string. The distance type to apply.
 weight is a numeric vector.

Details

This function calculates distance applying distance type and applying each weight to its characteristic.

Distance type could be EUC, MAN, CAN, CHE or OCT.

Value

Distance value applying weights.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
cluster1 <- matrix(c(2,3))
cluster2 <- matrix(c(4,5))

weight1 <- c(0.6,0.4)
weight2 <- c(2,4)

distances(cluster1, cluster2, 'MAN', weight1)

distances(cluster1, cluster2, 'CHE', weight2)
```

distances.details	<i>To calculate distances applying weights.</i>
-------------------	---

Description

To explain how to calculate distances between two clusters applying weights depending on the distance type.

Usage

```
distances.details(cluster1, cluster2, distance, weight)
```

Arguments

cluster1 is a matrix.
 cluster2 is a matrix.
 distance is a string. The distance type to apply.
 weight is a numeric vector.

Details

This function calculates distance applying distance type and applying each weight to its characteristic.

Distance type could be EUC, MAN, CAN, CHE or OCT.

Value

Distance value applying weights. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
cluster1 <- matrix(c(2,3))
cluster2 <- matrix(c(4,5))

weight1 <- c(0.6,0.4)
weight2 <- c(2,4)

distances.details(cluster1, cluster2, 'MAN', weight1)

distances.details(cluster1, cluster2, 'CHE', weight2)
```

divisiveHC

To execute divisive hierarchical clusterization algorithm by distance and approach.

Description

To execute complete divisive hierarchical clusterization algorithm by choosing distance and approach types.

Usage

```
divisiveHC(data, distance, approach)
```

Arguments

data	could be a numeric vector, a matrix or a numeric data frame. It will be transformed into matrix and list to be used.
distance	is a string. It chooses the distance to use.
approach	is a string. It chooses the approach to use.

Details

This function is the main part of the divisive hierarchical clusterization method. It executes the theoretical algorithm step by step.

- 1 - The function transforms data in useful object to be used.
- 2 - It creates a cluster that includes every simple elements.
- 3 - It initializes possible clusters using the initial elements.
- 4 - It calculates a matrix distance with the clusters created in the 3rd step.
- 5 - It chooses the maximal distance value and gets the clusters to be divided.
- 6 - It divides the cluster into two new complementary clusters and updates the clusters list.
- 6 - It repeats these steps until every cluster can't be divided again. The solution includes every simple cluster.

Value

A list with the divided clusters.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
Juan José Cuadrado <jjcg@uah.es>
Universidad de Alcalá de Henares

Examples

```
a <- c(1,2,1,3,1,4,1,5,1,6)
matrixA <- matrix(a,ncol=2)
dataFrameA <- data.frame(matrixA)
divisiveHC(a,'EUC','MAX')
divisiveHC(matrixA,'MAN','AVG')
```

```
divisiveHC(dataFrameA,'CHE','MIN')
```

divisiveHC.details	<i>To explain the divisive hierarchical clusterization algorithm by distance and approach.</i>
--------------------	--

Description

To explain the complete divisive hierarchical clusterization algorithm by choosing distance and approach types.

Usage

```
divisiveHC.details(data, distance, approach)
```

Arguments

data	could be a numeric vector, a matrix or a numeric data frame. It will be transformed into matrix and list to be used.
distance	is a string. It chooses the distance to use.
approach	is a string. It chooses the approach to use.

Details

This function is the main part of the divisive hierarchical clusterization method. It explains the theoretical algorithm step by step.

- 1 - The function transforms data in useful object to be used.
- 2 - It creates a cluster that includes every simple elements.
- 3 - It initializes possible clusters using the initial elements.
- 4 - It calculates a matrix distance with the clusters created in the 3rd step.
- 5 - It chooses the maximal distance value and gets the clusters to be divided.
- 6 - It divides the cluster into two new complementary clusters and updates the clusters list.
- 6 - It repeats these steps until every cluster can't be divided again. The solution includes every simple cluster.

Value

A list with the divided clusters. Explanation

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```

a <- c(1,2,1,3,1,4,1,5,1,6)

matrixA <- matrix(a,ncol=2)

dataFrameA <- data.frame(matrixA)

divisiveHC.details(a,'EUC','MAX')

divisiveHC.details(matrixA,'MAN','AVG')

divisiveHC.details(dataFrameA,'CHE','MIN')

```

edistance	<i>To calculate the Euclidean distance.</i>
-----------	---

Description

To calculate the Euclidean distance of two clusters.

Usage

```
edistance(x, y)
```

Arguments

x	is a numeric vector or a matrix. It represents the values of a cluster.
y	is a numeric vector or a matrix. It represents the values of a cluster.

Details

This function is part of the hierarchical clusterization method. The function calculates the Euclidean distance value from x and y.

Value

Euclidean distance value.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
x <- c(1,2)
y <- c(1,3)

cluster1 <- matrix(x,ncol=2)
cluster2 <- matrix(y,ncol=2)

edistance(x,y)

edistance(cluster1,cluster2)
```

edistance.details	<i>To show the Euclidean distance formula.</i>
-------------------	--

Description

To show the Euclidean distance formula and to calculate the Euclidean distance of two clusters.

Usage

```
edistance.details(x, y)
```

Arguments

x	is a numeric vector or a matrix. It represents the values of a cluster.
y	is a numeric vector or a matrix. It represents the values of a cluster.

Details

This function is part of the hierarchical clusterization method. The function calculates the Euclidean distance value from x and y.

Value

Euclidean distance value and formula.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
x <- c(1,2)
y <- c(1,3)

cluster1 <- matrix(x,ncol=2)
cluster2 <- matrix(y,ncol=2)

edistance.details(x,y)

edistance.details(cluster1,cluster2)
```

edistanceW	<i>To calculate the Euclidean distance applying weights.</i>
------------	--

Description

To calculate the Euclidean distance between clusters applying weights given.

Usage

```
edistanceW(cluster1, cluster2, weight)
```

Arguments

cluster1	is a cluster.
cluster2	is a cluster.
weight	is a numeric vector.

Details

The function calculates the Euclidean distance value from cluster1 and cluster2, applying weights to the cluster's components.

Value

Euclidean distance applying weights value.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
cluster1 <- matrix(c(1,2),ncol=2)
cluster2 <- matrix(c(1,3),ncol=2)

weight1 <- c(0.4,0.6)
weight2 <- c(2,12)

edistanceW(cluster1,cluster2,weight1)

edistanceW(cluster1,cluster2,weight2)
```

edistanceW.details	<i>To calculate the Euclidean distance applying weights.</i>
--------------------	--

Description

To explain how to calculate the Euclidean distance between clusters applying weights given.

Usage

```
edistanceW.details(cluster1, cluster2, weight)
```

Arguments

cluster1	is a cluster.
cluster2	is a cluster.
weight	is a numeric vector.

Details

The function calculates the Euclidean distance value from cluster1 and cluster2, applying weights to the cluster's components.

Value

Euclidean distance applying weights value. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```

cluster1 <- matrix(c(1,2),ncol=2)
cluster2 <- matrix(c(1,3),ncol=2)

weight1 <- c(0.4,0.6)
weight2 <- c(2,12)

edistanceW.details(cluster1,cluster2,weight1)

edistanceW.details(cluster1,cluster2,weight2)

```

getCluster	<i>To get the clusters with minimal distance.</i>
------------	---

Description

To get the clusters with the minimal distance value. By using the given distance, it gets the matrix index.

Usage

```
getCluster(distance, matrix)
```

Arguments

distance	is a number. It should be in the matrix.
matrix	is a numeric matrix.

Details

This function is part of the hierarchical clusterization method. The function uses the distance value and gets the clustersId with the minimal distance.

For the divisive algorithm, it chooses the distances from a distances list.

Value

numeric vector with two cluster indexes.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
matrixExample <- matrix(c(1:10), ncol=2)

getCluster(2,matrixExample)
```

getCluster.details	<i>To explain how to get the clusters with minimal distance.</i>
--------------------	--

Description

To explain how to get the clusters with the minimal distance value. By using the given distance, it gets the matrix index.

Usage

```
getCluster.details(distance, matrix)
```

Arguments

distance	is a number. It should be in the matrix.
matrix	is a numeric matrix.

Details

This function is part of the hierarchical clusterization method. The function uses the distance value and gets the clustersId with the minimal distance.

For the divisive algorithm, it chooses the distances from a distances list.

Value

Numeric vector with two clusters indexes.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
matrixExample <- matrix(c(1:10), ncol=2)

getCluster.details(2,matrixExample)
```

getClusterDivisive	<i>To get the clusters with maximal distance.</i>
--------------------	---

Description

To get the clusters with the maximal distance value. By using the given distance, it gets the matrix index.

Usage

```
getClusterDivisive(distance, vector)
```

Arguments

distance	is a number. It should be in the matrix.
vector	is a numeric vector

Details

This function is part of the hierarchical clusterization method. The function uses the distance value and gets the clustersId with the minimal distance.

For the divisive algorithm, it chooses the distances from a distances list.

Value

A cluster.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
getClusterDivisive(2,c(1:10))
```

```
getClusterDivisive(6,c(2,4,6,8,10,12))
```

getClusterDivisive.details

To explain how to get the clusters with maximal distance.

Description

To explain how to get the clusters with the maximal distance value. By using the given distance, it gets the matrix index.

Usage

```
getClusterDivisive.details(distance, vector)
```

Arguments

distance	is a number. It should be in the matrix.
vector	is a numeric vector.

Details

This function is part of the hierarchical clusterization method. The function uses the distance value and gets the clustersId with the minimal distance.

For the divisive algorithm, it chooses the distances from a distances list.

Value

A cluster. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
getClusterDivisive.details(2,c(1:10))
```

```
getClusterDivisive(6,c(2,4,6,8,10,12))
```

initClusters	<i>To initialize clusters for the divisive algorithm.</i>
--------------	---

Description

To initialize clusters for the divisive algorithm.

Usage

```
initClusters(initList)
```

Arguments

`initList` is a clusters list. It will contain clusters with one element.

Details

This function will calculate every cluster that can be created by joining initial clusters with each other. It creates clusters from length = 1 until a cluster with every element is created.

These clusters will be used to find the most different clusters that we can create by dividing the initial cluster.

Value

A cluster list.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
data <- c(1:8)
matrix <- matrix(data,ncol=2)
listData <- toListDivisive(data)
listMatrix <- toListDivisive(matrix)
initClusters(listData)
initClusters(listMatrix)
```

`initClusters.details` *To explain how to initialize clusters for the divisive algorithm.*

Description

To explain how to initialize clusters for the divisive algorithm.

Usage

```
initClusters.details(initList)
```

Arguments

`initList` is a clusters list. It will contain clusters with one element.

Details

This function will explain how to calculate every cluster that can be created by joining initial clusters with each other. It creates clusters from length = 1 until a cluster with every element is created.

These clusters will be used to find the most different clusters that we can create by dividing the initial cluster.

Value

A cluster list. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
data <- c(1:8)

matrix <- matrix(data, ncol=2)

listData <- toListDivisive(data)

listMatrix <- toListDivisive(matrix)

initClusters.details(listData)

initClusters.details(listMatrix)
```

initData	<i>To initialize data, hierarchical correlation algorithm.</i>
----------	--

Description

To initialize data, hierarchical correlation algorithm.

Usage

```
initData(data)
```

Arguments

data is a data frame with the main data.

Details

This function is part of the hierarchical correlation method. The function initializes data transforming each row from the data frame into a matrix with every row elements.

Value

A cluster list. Initializing data.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
data <- matrix(c(1,2,1,4,5,1,8,2,9,6,3,5,8,5,4),ncol= 3)

dataFrame <- data.frame(data)

initData(dataFrame)
```

initData.details	<i>To initialize data, hierarchical correlation algorithm.</i>
------------------	--

Description

To explain how to initialize data, hierarchical correlation algorithm.

Usage

```
initData.details(data)
```

Arguments

`data` is a data frame with the main data.

Details

This function is part of the hierarchical correlation method. The function initializes data transforming each row from the data frame into a matrix with every row elements.

Value

A cluster list. Initializing data. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
data <- matrix(c(1,2,1,4,5,1,8,2,9,6,3,5,8,5,4),ncol= 3)
dataFrame <- data.frame(data)
initData.details(dataFrame)
```

initImages	<i>To display an image.</i>
------------	-----------------------------

Description

An auxiliar function to display a picture.

Usage

```
initImages(path)
```

Arguments

path is a file path.

Details

This function is part of the hierarchical clusterization method. The function calculates the Euclidean distance value from x and y.

Examples

```
path <- "../man/images/euclideanDistance.PNG"
initImages(path)
```

initTarget	<i>To initialize target, hierarchical correlation algorithm.</i>
------------	--

Description

To initialize target, hierarchical correlation algorithm. It checks if target is valid, if not, it initializes the target

Usage

```
initTarget(target, data)
```

Arguments

target is a numeric vector, a matrix or a data frame.
data is a data frame with the main data.

Details

This function is part of the hierarchical correlation method. The function initializes `target` and checks if it is a valid target.

The function transforms the target into a matrix. Then, it checks if the target has only one row and the same columns has the main data.

If it is not a valid target, the function will notice the problem and will initialize a new target with every column with value 0.

Value

A cluster.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
data <- matrix(c(1,2,1,4,5,1,8,2,9,6,3,5,8,5,4),ncol= 3)

dataFrame <- data.frame(data)

target1 <- matrix(c(2,3))

target2 <- matrix(c(2,3,6))

initTarget(target1,dataFrame)

initTarget(target2,dataFrame)
```

<code>initTarget.details</code>	<i>To initialize target, hierarchical correlation algorithm.</i>
---------------------------------	--

Description

To initialize target, hierarchical correlation algorithm. It checks if target is valid, if not, it initializes the target

Usage

```
initTarget.details(target, data)
```


Arguments

`target` is a numeric vector, a matrix or a data frame.
`data` is a data frame with the main data.

Details

This function is part of the hierarchical correlation method. The function initializes `target` and checks if it is an acceptable target.

The function transforms the target into a matrix. Then, it checks if the target has only one row and the same columns have the main data.

If it is not an acceptable target, the function will notice the problem and will initialize a new target with every column with value 0.

Value

A cluster. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
data <- matrix(c(1,2,1,4,5,1,8,2,9,6,3,5,8,5,4),ncol= 3)
dataFrame <- data.frame(data)
target1 <- matrix(c(2,3))
target2 <- matrix(c(2,3,6))
initTarget.details(target1,dataFrame)
initTarget.details(target2,dataFrame)
```

matrixDistance	<i>Matrix distance by distance type</i>
----------------	---

Description

To calculate the matrix distance by using distance type.

Usage

```
matrixDistance(list, distance)
```

Arguments

`list` is a clusters list.
`distance` is a literal.

Details

This function is part of the hierarchical clusterization method. The function calculates the matrix distance by using the distance type given.

The `list` parameter will be a list with the clusters as rows and columns.

The function avoids distances equal 0 and undefined clusters.

Examples

```
data <- c(1:10)
clusters <- toList(data)
matrixDistance(clusters, 'EUC')
```

maxDistance	<i>Maximal distance</i>
-------------	-------------------------

Description

Get the matrix maximal value.

Usage

```
maxDistance(matrix)
```

Arguments

`matrix` is a numeric matrix. It could be a numeric vector.

Details

This function is part of the hierarchical clusterization method. The function uses the numeric vector or matrix `matrix` given and return the maximal value. The function avoids distances equal 0, and initialize maximal value with an auxiliar function `initMax`, which gets the first matrix element with a valid distance.

Value

numeric value. Max value from a matrix

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
matrixExample <- matrix(c(1:10), nrow=2)

maxDistance(1:10)

maxDistance(matrixExample)
```

maxDistance.details	<i>Maximal distance</i>
---------------------	-------------------------

Description

To explain how to get the matrix maximal value.

Usage

```
maxDistance.details(matrix)
```

Arguments

`matrix` is a numeric matrix. It could be a numeric vector.

Details

This function is part of the hierarchical clusterization method. The function uses the numeric vector or matrix `matrix` given and return the maximal value. The function avoids distances equal 0, and initialize maximal value with an auxiliar function `initMax`, which gets the first matrix element with a valid distance.

Value

Numeric value. Max value from a matrix. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
matrixExample <- matrix(c(1:10), nrow=2)

maxDistance.details(1:10)

maxDistance.details(matrixExample)
```

mdAgglomerative	<i>Matrix distance by distance and approach type.</i>
-----------------	---

Description

To calculate the matrix distance by using distance and approach type.

Usage

```
mdAgglomerative(list, distance, approach)
```

Arguments

list	is a clusters list.
distance	is a literal. The distance type to be used.
approach	is a literal. The approach type to be used.

Details

This function is part of the hierarchical clusterization method. The function calculates the matrix distance by using the distance and approach type given.

The list parameter will be a list with the clusters as rows and columns.

The function avoids distances equal 0 and undefined clusters.

Value

A matrix distance.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
data <- c(1,2,1,3,1,4,1,5,1,6)

clusters <- toList(data)

mdAgglomerative(clusters, 'EUC', 'MAX')

mdAgglomerative(clusters, 'CHE', 'AVG')
```

mdAgglomerative.details

Matrix distance by distance and approach type.

Description

To explain how to calculate the matrix distance by using distance and approach type.

Usage

```
mdAgglomerative.details(list, distance, approach)
```

Arguments

list	is a clusters list.
distance	is a literal. The distance type to be used.
approach	is a literal. The approach type to be used.

Details

This function is part of the hierarchical clusterization method. The function calculates the matrix distance by using the distance and approach type given.

The list parameter will be a list with the clusters as rows and columns.

The function avoids distances equal 0 and undefined clusters.

Value

A matrix distance. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
data <- c(1,2,1,3,1,4,1,5,1,6)

clusters <- toList(data)

mdAgglomerative.details(clusters, 'EUC', 'MAX')

mdAgglomerative.details(clusters, 'CHE', 'AVG')
```

mdDivisive

Matrix distance by distance and approach type.

Description

To calculate the matrix distance by using distance and approach types.

Usage

```
mdDivisive(list, distance, approach, components)
```

Arguments

list	is a clusters list.
distance	is a string. The distance type to be used.
approach	is a string. The approach type to be used.
components	is a clusters list. It contains every clusters with only one element. It is used to check if complementary condition is 'TRUE'.

Details

This function is part of the divisive hierarchical clusterization method. The function calculates the matrix distance by using the distance and approach types given.

The list parameter will be a list with the clusters as rows and columns.

The function avoids distances equal 0 and undefined clusters.

It also avoids distances between clusters that are not complementary because they can't be chosen to divide all the clusters.

Value

Matrix distance.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
data <- c(1,2,1,3,1,4,1,5,1,6)
clusters <- toList(data)
components <- toList(data)
mdDivisive(clusters, 'EUC', 'MAX', components)
mdDivisive(clusters, 'MAN', 'MIN', components)
```

mdDivisive.details	<i>Matrix distance by distance and approach type.</i>
--------------------	---

Description

To explain how to calculate the matrix distance by using distance and approach types.

Usage

```
mdDivisive.details(list, distance, approach, components)
```

Arguments

list	is a clusters list.
distance	is a string. The distance type to be used.
approach	is a string. The approach type to be used.
components	is a clusters list. It contains every clusters with only one element. It is used to check if complementary condition is 'TRUE'.

Details

This function is part of the divisive hierarchical clusterization method. The function calculates the matrix distance by using the distance and approach types given.

The `list` parameter will be a list with the clusters as rows and columns.

The function avoids distances equal 0 and undefined clusters.

It also avoids distances between clusters that are not complementary because they can't be chosen to divide all the clusters.

Value

Matrix distance. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
data <- c(1,2,1,3,1,4,1,5,1,6)

clusters <- toList(data)

components <- toList(data)

mdDivisive.details(clusters, 'EUC', 'MAX', components)

mdDivisive.details(clusters, 'MAN', 'MIN', components)
```

mdistance	<i>To calculate the Manhattan distance.</i>
-----------	---

Description

To calculate the Manhattan distance of two clusters.

Usage

```
mdistance(x, y)
```

Arguments

<code>x</code>	is a numeric vector or a matrix. It represents the values of a cluster.
<code>y</code>	is a numeric vector or a matrix. It represents the values of a cluster.

Details

This function is part of the hierarchical clusterization method. The function calculates the Manhattan distance value from x and y.

Value

Manhattan distance value.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
x <- c(1,2)
y <- c(1,3)

cluster1 <- matrix(x,ncol=2)
cluster2 <- matrix(y,ncol=2)

mdistance(x,y)

mdistance(cluster1,cluster2)
```

mdistance.details	<i>To explain how to calculate the Manhattan distance.</i>
-------------------	--

Description

To explain how to calculate the Manhattan distance of two clusters.

Usage

```
mdistance.details(x, y)
```

Arguments

x	is a numeric vector or a matrix. It represents the values of a cluster.
y	is a numeric vector or a matrix. It represents the values of a cluster.

Details

This function is part of the hierarchical clusterization method. The function calculates the Manhattan distance value from x and y.

Value

Manhattan distance value and formula.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
x <- c(1,2)
y <- c(1,3)

cluster1 <- matrix(x,ncol=2)
cluster2 <- matrix(y,ncol=2)

mdistance.details(x,y)

mdistance.details(cluster1,cluster2)
```

mdistanceW

To calculate the Manhattan distance applying weights.

Description

To calculate the Manhattan distance between clusters applying weights given.

Usage

```
mdistanceW(cluster1, cluster2, weight)
```

Arguments

cluster1	is a cluster.
cluster2	is a cluster.
weight	is a numeric vector.

Details

The function calculates the Manhattan distance value from cluster1 and cluster2, applying weights to the cluster's components.

Value

Manhattan distance applying weights value.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
cluster1 <- matrix(c(1,2),ncol=2)
cluster2 <- matrix(c(1,3),ncol=2)

weight1 <- c(0.4,0.6)
weight2 <- c(2,12)

mdistanceW(cluster1,cluster2,weight1)

mdistanceW(cluster1,cluster2,weight2)
```

mdistanceW.details	<i>To calculate the Manhattan distance applying weights.</i>
--------------------	--

Description

To explain how to calculate the Manhattan distance between clusters applying weights given.

Usage

```
mdistanceW.details(cluster1, cluster2, weight)
```

Arguments

cluster1	is a cluster.
cluster2	is a cluster.
weight	is a numeric vector.

Details

The function calculates the Manhattan distance value from cluster1 and cluster2, applying weights to the cluster's components.

Value

Manhattan distance applying weights value. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
cluster1 <- matrix(c(1,2),ncol=2)
cluster2 <- matrix(c(1,3),ncol=2)

weight1 <- c(0.4,0.6)
weight2 <- c(2,12)

mdistanceW.details(cluster1,cluster2,weight1)

mdistanceW.details(cluster1,cluster2,weight2)
```

minDistance	<i>Minimal distance</i>
-------------	-------------------------

Description

Get the matrix minimal value.

Usage

```
minDistance(matrix)
```

Arguments

`matrix` is a numeric matrix. It could be a numeric vector.

Details

This function is part of the hierarchical clusterization method. The function uses the numeric vector or matrix `matrix` given and return the minimal value. The function avoids distances equal 0, and initialize minimum value with an auxiliar function `initMin`, which gets the first matrix element with a valid distance.

Value

Numeric value. Min value from a matrix.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
matrixExample <- matrix(c(1:10), nrow=2)

minDistance(1:10)

minDistance(matrixExample)
```

minDistance.details	<i>Minimal distance</i>
---------------------	-------------------------

Description

To explain how to get the matrix minimal value.

Usage

```
minDistance.details(matrix)
```

Arguments

`matrix` is a numeric matrix. It could be a numeric vector.

Details

This function is part of the hierarchical clusterization method. The function uses the numeric vector or matrix `matrix` given and return the minimal value. The function avoids distances equal 0, and initialize minimum value with an auxiliar function `initMin`, which gets the first matrix element with a valid distance.

Value

Numeric value. Min value from a matrix. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
matrixExample <- matrix(c(1:10), nrow=2)

minDistance(1:10)

minDistance.details(matrixExample)
```

newCluster	<i>To create a new cluster.</i>
------------	---------------------------------

Description

To create the cluster formed by the two clusters given. Add the new cluster to list.

Usage

```
newCluster(list, clusters)
```

Arguments

list	is the generic cluster list.
clusters	is a vector with the matrix index clusters.

Details

This function is part of the hierarchical clusterization method.

- 1 - The function maps clusters in list.
- 2 - It creates a new cluster from them.
- 3 - It adds the new cluster to list.
- 4 - It disables the clusters used in the second step.

Value

A list with clusters.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
data <- c(1:10)
list <- toList(data)
clusters <- c(1,2)
newCluster(list,clusters)
```

newCluster.details	<i>To explain how to create a new cluster.</i>
--------------------	--

Description

To explain how to create the cluster formed by the two clusters given. Add the new cluster to list.

Usage

```
newCluster.details(list, clusters)
```

Arguments

list	is the generic cluster list.
clusters	is a vector with the matrix index clusters.

Details

This function is part of the hierarchical clusterization method.

- 1 - The function maps clusters in list.
- 2 - It creates a new cluster from them.
- 3 - It adds the new cluster to list.
- 4 - It disables the clusters used in the second step.

Value

A list with clusters. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
data <- c(1:10)

list <- toList(data)

clusters <- c(1,2)

newCluster.details(list,clusters)
```

normalizeWeight	<i>To normalize weight values.</i>
-----------------	------------------------------------

Description

To normalize weight values if `normalize = TRUE`.

Usage

```
normalizeWeight(normalize, weight, data)
```

Arguments

<code>normalize</code>	is a boolean value.
<code>weight</code>	is a numeric vector.
<code>data</code>	is a data.frame.

Details

This function allows users to normalize weights.

If there is not any weight, the function will create a numeric vector of "1".

If `normalize = TRUE`, the function will make every weight value as a "[0:1]" value.

If `normalize = FALSE`, the function will not make any changes, weights will be the same.

Value

Numeric vector with updated weights.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
data <- data.frame(matrix(c(1:10),ncol = 2))

weight1 <- c(0.6,0.4)
weight2 <- c(2,4)

normalizeWeight(FALSE, weight1, data)

normalizeWeight(TRUE, weight2, data)

normalizeWeight(FALSE, weight2, data)
```

```
normalizeWeight.details
```

To normalize weight values.

Description

To explain how to normalize weight values if normalize = TRUE.

Usage

```
normalizeWeight.details(normalize, weight, data)
```

Arguments

normalize	is a boolean value.
weight	is a numeric vector.
data	is a data.frame.

Details

This function allows users to normalize weights.

If there is not any weight, the function will create a numeric vector of "1".

If normalize = TRUE, the function will make every weight value as a "[0:1]" value.

If normalize = FALSE, the function will not make any changes, weights will be the same.

Value

Numeric vector with updated weights. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
data <- data.frame(matrix(c(1:10),ncol = 2))

weight1 <- c(0.6,0.4)
weight2 <- c(2,4)

normalizeWeight.details(FALSE, weight1, data)

normalizeWeight.details(TRUE, weight2, data)

normalizeWeight.details(FALSE, weight2, data)
```

octileDistance	<i>To calculate the Octile distance.</i>
----------------	--

Description

To calculate the octile distance of two clusters.

Usage

```
octileDistance(x, y)
```

Arguments

x	is a numeric vector or a matrix. It represents the values of a cluster.
y	is a numeric vector or a matrix. It represents the values of a cluster.

Details

This function is part of the hierarchical clusterization method. The function calculates the octile distance value from x and y.

Value

Octile distance value.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
x <- c(1,2)
y <- c(1,3)

cluster1 <- matrix(x,ncol=2)
cluster2 <- matrix(y,ncol=2)

octileDistance(x,y)

octileDistance(cluster1,cluster2)
```

`octileDistance.details`*To explain how to calculate the Octile distance.*

Description

To explain how to calculate the octile distance of two clusters.

Usage

```
octileDistance.details(x, y)
```

Arguments

x	is a numeric vector or a matrix. It represents the values of a cluster.
y	is a numeric vector or a matrix. It represents the values of a cluster.

Details

This function is part of the hierarchical clusterization method. The function calculates the octile distance value from x and y.

Value

Octile distance value.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
x <- c(1,2)
y <- c(1,3)

cluster1 <- matrix(x,ncol=2)
cluster2 <- matrix(y,ncol=2)

octileDistance.details(x,y)

octileDistance.details(cluster1,cluster2)
```

octileDistanceW	<i>To calculate the Octile distance applying weights.</i>
-----------------	---

Description

To calculate the Octile distance between clusters applying weights given.

Usage

```
octileDistanceW(cluster1, cluster2, weight)
```

Arguments

cluster1	is a cluster.
cluster2	is a cluster.
weight	is a numeric vector.

Details

The function calculates the Octile distance value from cluster1 and cluster2, applying weights to the cluster's components.

Value

Octile distance applying weights value.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```
cluster1 <- matrix(c(1,2),ncol=2)
cluster2 <- matrix(c(1,3),ncol=2)

weight1 <- c(0.4,0.6)
weight2 <- c(2,12)

octileDistanceW(cluster1,cluster2,weight1)

octileDistanceW(cluster1,cluster2,weight2)
```

```
octileDistanceW.details
```

To calculate the Octile distance applying weights.

Description

To explain how to calculate the Octile distance between clusters applying weights given.

Usage

```
octileDistanceW.details(cluster1, cluster2, weight)
```

Arguments

cluster1	is a cluster.
cluster2	is a cluster.
weight	is a numeric vector.

Details

The function calculates the Octile distance value from cluster1 and cluster2, applying weights to the cluster's components.

Value

Octile distance applying weights value. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>
 Juan José Cuadrado <jjcg@uah.es>
 Universidad de Alcalá de Henares

Examples

```

cluster1 <- matrix(c(1,2),ncol=2)
cluster2 <- matrix(c(1,3),ncol=2)

weight1 <- c(0.4,0.6)
weight2 <- c(2,12)

octileDistanceW.details(cluster1,cluster2,weight1)

octileDistanceW.details(cluster1,cluster2,weight2)

```

toList	<i>To transform data into list</i>
--------	------------------------------------

Description

To transform data into list.

Usage

```
toList(data)
```

Arguments

data could be a numeric vector, a matrix or a numeric data frame.

Details

This function is part of the agglomerative hierarchical clusterization method. The function initializes data content as a list.

In agglomerative algorithm, it adds a TRUE flag to each element, which indicates that the cluster is not grouped.

Value

A list with clusters.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
data <- c(1:10)

matrix <- matrix(data,ncol=2)

dataFrame <- data.frame(matrix)

toList(data)

toList(matrix)

toList(dataFrame)
```

toList.details	<i>To explain how to transform data into list</i>
----------------	---

Description

To explain how to transform data into list.

Usage

```
toList.details(data)
```

Arguments

data could be a numeric vector, a matrix or a numeric data frame.

Details

This function is part of the agglomerative hierarchical clusterization method. The function initializes data content as a list.

In agglomerative algorithm, it adds a TRUE flag to each element, which indicates that the cluster is not grouped.

Value

A list with cñclusters. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
data <- c(1:10)

matrix <- matrix(data,ncol=2)

dataFrame <- data.frame(matrix)

toList(data)

toList(matrix)

toList(dataFrame)
```

toListDivisive	<i>To transform data into list</i>
----------------	------------------------------------

Description

To transform data into list.

Usage

```
toListDivisive(data)
```

Arguments

data could be a numeric vector, a matrix or a numeric data frame.

Details

This function is part of the divisive hierarchical clusterization method. The function initializes data content as a list.

Value

a list with clusters.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
data <- c(1:10)

matrix <- matrix(data,ncol=2)

dataFrame <- data.frame(matrix)

toListDivisive(data)

toListDivisive(matrix)

toListDivisive(dataFrame)
```

`toListDivisive.details`*To explain how to transform data into list*

Description

To explain how to transform data into list.

Usage

```
toListDivisive.details(data)
```

Arguments

`data` could be a numeric vector, a matrix or a numeric data frame.

Details

This function is part of the divisive hierarchical clusterization method. The function initializes data content as a list.

Value

A list with clusters. Explanation.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
data <- c(1:10)

matrix <- matrix(data,ncol=2)

dataFrame <- data.frame(matrix)

toListDivisive.details(data)

toListDivisive.details(matrix)

toListDivisive.details(dataFrame)
```

usefulClusters	<i>To delete clusters grouped.</i>
----------------	------------------------------------

Description

To delete the clusters already used to create a new one.

Usage

```
usefulClusters(list)
```

Arguments

`list` is a list of clusters.

Details

This function is part of the hierarchical clusterization method. The function updates the cluster list with the clusters used after to calculate de matrix distance.

Value

A list of clusters.

Author(s)

Roberto Alcántara <roberto.alcantara@edu.uah.es>

Juan José Cuadrado <jjcg@uah.es>

Universidad de Alcalá de Henares

Examples

```
data <- c(1:10)
list <- toList(data)
usefulClusters(list)
```

Index

agglomerativeHC, [3](#)
agglomerativeHC.details, [4](#)

canberradistance, [5](#)
canberradistance.details, [6](#)
canberradistanceW, [7](#)
canberradistanceW.details, [8](#)
chebyshevDistance, [9](#)
chebyshevDistance.details, [10](#)
chebyshevDistanceW, [11](#)
chebyshevDistanceW.details, [12](#)
clusterDistance, [13](#)
clusterDistance.details, [14](#)
clusterDistanceByApproach, [15](#)
clusterDistanceByApproach.details, [16](#)
complementaryClusters, [17](#)
complementaryClusters.details, [18](#)
correlationHC, [19](#)
correlationHC.details, [21](#)

distances, [22](#)
distances.details, [23](#)
divisiveHC, [24](#)
divisiveHC.details, [26](#)

edistance, [27](#)
edistance.details, [28](#)
edistanceW, [29](#)
edistanceW.details, [30](#)

getCluster, [31](#)
getCluster.details, [32](#)
getClusterDivisive, [33](#)
getClusterDivisive.details, [34](#)

initClusters, [35](#)
initClusters.details, [36](#)
initData, [37](#)
initData.details, [38](#)
initImages, [39](#)
initTarget, [39](#)

initTarget.details, [40](#)

matrixDistance, [41](#)
maxDistance, [42](#)
maxDistance.details, [43](#)
mdAgglomerative, [44](#)
mdAgglomerative.details, [45](#)
mdDivisive, [46](#)
mdDivisive.details, [47](#)
mdistance, [48](#)
mdistance.details, [49](#)
mdistanceW, [50](#)
mdistanceW.details, [51](#)
minDistance, [52](#)
minDistance.details, [53](#)

newCluster, [54](#)
newCluster.details, [55](#)
normalizeWeight, [56](#)
normalizeWeight.details, [57](#)

octileDistance, [58](#)
octileDistance.details, [59](#)
octileDistanceW, [60](#)
octileDistanceW.details, [61](#)

toList, [62](#)
toList.details, [63](#)
toListDivisive, [64](#)
toListDivisive.details, [65](#)

usefulClusters, [66](#)

9 BIBLIOGRAFÍA

References

- [1] H. Wickham. (Apr 2015). R Packages. https://cran.r-project.org/web/packages/unitizer/vignettes/u1_intro.html
- [2] H. Wickham. (Apr 2015). R Packages. <http://r-pkgs.had.co.nz/man.html>
- [3] H. Wickham. (Apr 2015). R Packages. <http://r-pkgs.had.co.nz/src.html>
- [4] H. Wickham. (Apr 2015). R Packages. <http://r-pkgs.had.co.nz/data.html>
- [5] H. Wickham. (Apr 2015). R Packages. <http://r-pkgs.had.co.nz/description.html>
- [6] H. Parker. (29 Apr 2014). Writing an R package from scratch. <https://hilaryparker.com/2014/04/29/writing-an-r-package-from-scratch/>
- [7] K. Broman. (Aug 2014). Making it a proper package. http://kbroman.org/pkg_primer/pages/proper.html
- [8] Sathish. (Dec 2015). it-swarm.dev <https://www.it-swarm.dev/es/r/namespace-no-generado-por-roxygen2.-saltado-confusion-con-el-libro-de-hadley/1052857995/>
- [9] K. Broman. (Aug 2014). Writing documentation with Roxygen2. http://kbroman.org/pkg_primer/pages/docs.html
- [10] M. García (25 Jan 2017). Hacer y documentar un paquete de R en 20 minutos. https://mauriciogtec.github.io/rGallery/entries/tutoriales/crear_paquetes/crear_paquete.html
- [11] M. Febrero Bande, M. Oviedo de la Fuente. (Octubre 2013) Creación de paquetes, informes y viñetas. http://eio.usc.es/pub/febrero/paquete/Crear_paquetes3.pdf
- [12] H. Wang. <https://www.stt.msu.edu/users/hengwang/R%20package%20tutorial.pdf>
- [13] K. Broman. (Aug 2014). Writing documentation with Roxygen2. https://kbroman.org/pkg_primer/pages/docs.html
- [14] R. M. Ripley (2013) Making an R package <http://portal.stats.ox.ac.uk/userdata/ruth/APTS2012/Rcourse10.pdf>
- [15] K. Broman. (Aug 2014). Writing vignettes http://kbroman.org/pkg_primer/pages/vignettes.html
- [16] RStudio. (Aug 2014). R Markdown. <https://rstudio.com/wp-content/uploads/2015/03/rmarkdown-spanish.pdf>

- [17] J. J. Zubcoff Vallejo, Universidad de Alicante. (2017). Informes con RStudio y R Markdown. https://rua.ua.es/dspace/bitstream/10045/69768/1/Modulo_5_-_Informes_con_R_Markdown.pdf
- [18] T. Boca. (10 Mar 2019). Introducción al uso de Rmarkdown. https://rpubs.com/tereboca/informe_rmakrdwn
- [19] <https://tex.stackexchange.com/questions/399258/user-and-admin-miktex-update>
- [20] CRAN. <https://cran.r-project.org/submit.html>
- [21] RMarkdown RStudio. (2016). <https://rmarkdown.rstudio.com/lesson-1.html>
- [22] H. Wickham. (Apr 2015). R Packages. R code. <https://r-pkgs.org/r.html#style>.
- [23] D. Monheimius. (Oct 2018). LearningRlab: Statistical Learning Functions. <https://cran.r-project.org/web/packages/LearningRlab/index.html>
- [24] RStudio. (Jan 2015). Desarrollo de Paquetes con devtools Hoja de Referencia. <https://rstudio.com/wp-content/uploads/2015/03/devtools-spanish.pdf>
- [25] F. J. Alcaraz Ariza. (14 Feb 2013). Clasificación y ordenación con R. <https://www.um.es/docencia/geobotanica/ficheros/practica2.pdf?q=con-r>
- [26] F. Carmona. (Mar 2013). Creación de paquetes de R en Windows (y Linux). http://www.ub.edu/stat/docencia/Cursos-R/Radvanced/materials/Crear_paquetes_R.pdf
- [27] <https://www.ucm.es/data/cont/docs/1346-2019-04-12-BaSix%20LaTeX%20ba%CC%81sico%20con%20ejercicios%20resueltos%20-%20Noir16.pdf>
- [28] oddworldng. (26 Feb 2018). ¿Cómo documentar un package en R con roxygen2? <https://inebaser.wordpress.com/2018/02/26/como-documentar-un-package-en-r-con-roxygen2/>
- [29] <https://www.ugr.es/~gallardo/pdf/cluster-3.pdf>
- [30] A. Santana, C. N. Hernández. (2016). Creación de paquetes R con Rstudio <http://estadistica-dma.ulpgc.es/cursosR4ULPGC/16-crearPaquetes.html>
- [31] H. Wickham. (Apr 2015). R Packages. Package structure and state. <https://r-pkgs.org/package-structure-state.html>
- [32] <http://wpd.ugr.es/~bioestad/guia-spss/practica-8/>
- [33] O. Perpiñan Lamigueiro. (22 May 2019). Crear paquetes en R. <https://oscarperpinan.github.io/R/Paquetes.html>

- [34] F. Leisch, B. Gruen. (4 Sep 2020). CRAN Task View: Cluster Analysis & Finite Mixture Models. <https://cran.r-project.org/web/views/Cluster.html>
- [35] <https://www.ugr.es/~batanero/pages/ARTICULOS/libroR.pdf>
- [36] <https://www.r-bloggers.com/how-to-perform-hierarchical-clustering-using-r/>
- [37] D. Calvo. (3 Octubre 2016). Análisis Clúster Jerárquico en R. <https://www.diegocalvo.es/analisis-cluster-jerarquico-en-r/>
- [38] M.Á. Jiménez Cuadrillero. (9 Mayo 2018). Clustering Jerárquico en R. <https://rpubs.com/mjimcua/clustering-jerarquico-en-r>
- [39] Amit's Thoughts on Pathfinding (Página 2). <http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html>