Fuzzy C-Means Distribuido En La Nube

Juan Carlos Ruiz González
Grado en Ingeniería Informática
Inteligencia Artificial

Consultor
David Isern Alarcón

Profesor Responsable de la Asignatura
Carles Ventura Royo

Enero de 2017

La frase más excitante que se puede oír en ciencia,
la que anuncia nuevos descubrimientos,
no es ¡Eureka! sino 'Es extraño...'

Isaac Asimov

Todos somos hijos del Informador

Garry Kaspárov

Agradecimientos

A todos aquellos que han permitido
con su silencioso trabajo que escriba estas líneas.
No necesito nombrarlos pues bien saben ellos quienes son.

A David Isern
por sus acertados consejos y buen hacer.

Título del trabajo:	*Fuzzy C-Means Distribuido en la Nube*
Nombre del autor:	*Juan Carlos Ruiz González*
Nombre del consultor:	*David Isern Alarcón*
Nombre del PRA:	*Carles Ventura Royo*
Fecha de entrega:	0 1 / 1 7
Titulación:	*Grado en Ingeniería Informática*
Área del Trabajo Final:	*Inteligencia Artificial*
Idioma del trabajo:	*Castellano*
Palabras clave	*Big Data; Fuzzy C-Means; MapReduce*

Resumen

Big Data y MapReduce son dos conceptos estrechamente unidos en la actualidad. Adaptar técnicas de aprendizaje computacional a estos nuevos paradigmas es una necesidad que contribuye sensiblemente al descubrimiento de nuevo conocimiento.

Por otro lado el algoritmo de clustering difuso Fuzzy C-Means es muy utilizado para procesar imágenes, reconocer patrones y analizar datos.

En este trabajo se adapta el algoritmo Fuzzy C-Means al modelo de programación MapReduce con el objetivo de ser ejecutado en un framework para Big Data.

Se han obtenido dos adaptaciones, la versión uno intentado generar la menor cantidad posible de tráfico de red y la dos intentado tener el mayor porcentaje posible de código a ejecutar en paralelo.

Se han implementado ambas versiones utilizando el framework Hadoop y se han realizado pruebas de validación que demuestran la corrección de las adaptaciones. También se han realizado pruebas de rendimiento que muestran que conforme aumenta el número de clusters la versión con mayor paralelismo tiende a tener un mejor rendimiento.

Abstract

Big Data and MapReduce are two closely related concepts nowadays. Adapting the techniques of Machine Learning to these new paradigms is a necessity that contributes noticeably to discover new knowledge.

On the other hand, the Fuzzy C-Means diffuse clustering algorithm is widely used to process images, recognize patterns and analyze data.

The goal of this work is the implementation of the well-known clustering algorithm as Fuzzy C-Means in a big data-based framework.

Two adaptations have been obtained, the first one trying to generate the least possible amount of network traffic and the second one trying to get the highest possible percentage of code to run in parallel.

Both versions have been implemented using the Hadoop framework and validation tests have been performed to demonstrate the correctness of adaptations. Performance tests have also been performed which show that as the number of clusters increases, the more parallel version tends to have a better throughput.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Tal y como podemos ver en [1] Big Data es hoy en día la nueva frontera donde se decide la competitividad, la innovación y la productividad en múltiples ámbitos. Debido a la ingente cantidad de datos a procesar el uso de la computación distribuida se hace inevitable, y adaptar técnicas de Inteligencia Artificial a sistemas distribuidos es una necesidad objetiva.

La lógica difusa a venido a aportar una herramienta potentísima a la Inteligencia Artificial desde que fue formulada en 1965 por el ingeniero y matemático Lofti A. Zadeh. El uso de esta lógica es común en la creación de sistemas expertos o en el reconocimiento de patrones entre otras aplicaciones y también como no en sistemas de clasificación y agrupamiento.

Dentro del aprendizaje no supervisado Fuzzy C-means es uno de los algoritmos difusos más usado a la hora de agrupar datos cuando no deseamos un agrupamiento exclusivo, es decir, cuando lo que deseamos es que dado un elemento podamos conocer el grado de pertenencia a las distintas agrupaciones en las que se pueda dividir el conjunto inicial de datos.

Adaptar este algoritmo a un entorno distribuido viene casi de suyo como necesidad cuando lo que queremos es obtener resultados a partir de hacer agrupaciones con grandes cantidades de datos propias de Big Data.

Existen diversas técnicas y herramientas que permiten llevar a cabo dicha adaptación y pretendemos estudiar como hacerlo con una de las herramientas y técnica de mayor implantación en la industria a día de hoy para trabajos de Cloud Computing.

1.2 Objetivos del Trabajo

Los objetivos del presente trabajo son en consecuencia:

- Describir el Big Data
- Introducir los conceptos básicos de la lógica difusa
- Describir el Aprendizaje No Supervisado y el Clustering (agrupamiento) como técnicas en Inteligencia artificial
- Describir el Fuzzy C-Means
- Introducir el Cloud Computing
- Caracterizar la técnica MapReduce para computación distribuida
- Adaptar Fuzzy C-Means al paradigma MapReduce
- Implementar y probar dicha técnica en un conjunto de datos

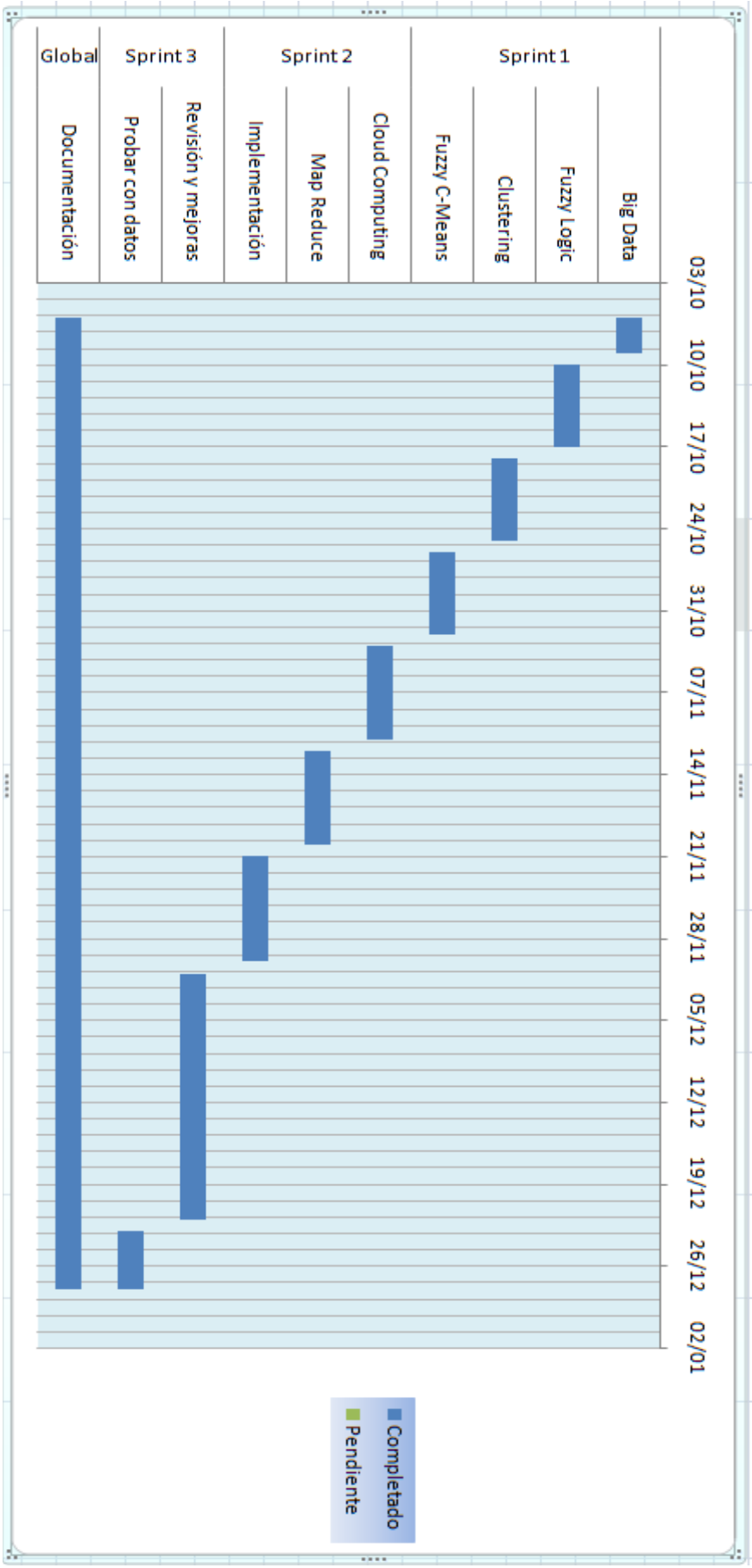
1.3 Planificación del Trabajo

Las tareas a realizar en el trabajo se corresponden con cada uno de los objetivos presentados y cada grupo de objetivos con una de las distintas entregas que se han de realizar.

Cada entrega se va a corresponder con un sprint o etapa que queda descrito con precisión en la siguiente tabla.

Etapa	Proceso	Fecha Inicio	Duración	Fecha Fin	Certificado	Completado	Pendiente
Sprint 1	Big Data	06/10/2016	3	09/10/2016	100%	3,00	0,00
	Fuzzy Logic	10/10/2016	7	17/10/2016	100%	7,00	0,00
	Clustering	18/10/2016	7	25/10/2016	100%	7,00	0,00
	Fuzzy C-Means	26/10/2016	7	02/11/2016	100%	7,00	0,00
Sprint 2	Cloud Computing	03/11/2016	8	11/11/2016	100%	8,00	0,00
	Map Reduce	12/11/2016	8	20/11/2016	100%	8,00	0,00
	Implementación	21/11/2016	9	30/11/2016	100%	9,00	0,00
Sprint 3	Revisión y mejoras	01/12/2016	21	22/12/2016	100%	21,00	0,00
	Probar con datos	23/12/2016	5	28/12/2016	100%	5,00	0,00
Global	Documentación	06/10/2016	83	28/12/2016	100%	83,00	0,00

A partir de la cual se ha realizado el siguiente diagrama de Gantt que nos va a permitir visualizar la información de manera más rápida.


Global
Documentación
Sprint 3
Probar con datos
Revisión y mejoras
Sprint 2
Implementación
Map Reduce
Cloud Computing
Sprint 1
Fuzzy C-Means
Clustering
Fuzzy Logic
Big Data
03/10
10/10
17/10
24/10
31/10
07/11
14/11
21/11
28/11
05/12
12/12
19/12
26/12
02/01
Completado
Pendiente

Los riesgos asociados al proyecto los podemos ver en la siguiente tabla

	Sprint 1	Sprint 2	Sprint 3
RIESGO	Falta de tiempo	Falta de tiempo y Problemas con las tecnologías	Falta de tiempo y Problemas con las tecnologías
GRAVEDAD	Baja	Baja	Alta
SOLUCIÓN	Dedicar más tiempo	Dedicar más tiempo	Buscar ayuda externa (Tutor)
EVALUACIÓN	Continua	Continua	Continua
CERTIFICADO	0,00%	20,00%	0,00%

Los recursos necesarios para llevar a cabo el proyecto son los que se especifican a continuación

- Ordenador con
  - 16 GB de RAM
  - procesador Intel i3
  - Acceso a Internet
- Windows 7
- Virtual Box
- Distribución Hadoop de Cloudera

1.4 Enfoque y método seguido

Como el objetivo primario es la adaptación de un algoritmo comenzamos acercándonos paulatinamente a los conceptos necesarios para entender dicho algoritmo -Fuzzy C-Means- y su utilidad, estos conceptos son el Big Data, la Lógica Difusa y el Clustering. Una vez que tenemos las ideas claras entramos de lleno en el estudio del algoritmo.

A partir de ese punto volvemos a ir acercándonos a otra serie de conceptos, necesarios para entender el mundo al que hay que adaptar el algoritmo, como son el Cloud Computing y el paradigma MapReduce.

Ya en este punto estudiamos con detalle la adaptación del algoritmo Fuzzy C-Means al modelo MapReduce. Ahora toca acercarse al framework que va a permitir la implementación de dicha adaptación, Hadoop.

Finalmente toca implementar las dos versiones estudiadas y llevar a cabo las pruebas de validación y de rendimiento.

1.5 Productos obtenidos

Se obtienen dos adaptaciones de Fuzzy C-Means al paradigma MapReduce, las implementaciones de dichas adaptaciones sobre Hadoop y las pruebas de validación y rendimiento correspondientes.

1.6 Otros capítulos

Capítulo 2. Big Data: se describe qué es Big Data y la importancia que tiene hoy en día tanto económica como científica.

Capítulo 3. Lógica Difusa: se realiza una amplia descripción de la Lógica Difusa y algunos de su usos.

Capítulo 4. Clustering: introducimos el aprendizaje no supervisado y se explica con detalle el Clustering como técnica usual en el aprendizaje computacional.

Capítulo 5. Fuzzy C-Means: estudio detallado del algoritmo.

Capítulo 6. Cloud Computing: presentación de los conceptos necesarios para entender qué es exactamente este modelo de computación.

Capítulo 7. MapReduce: se explica con detalle el paradigma de computación MapReduce.

Capítulo 8: Fuzzy C-Means sobre MapReduce: desarrollamos las adaptaciones del algoritmo a un entorno MapReduce.

Capítulo 9: Implementación y pruebas en Hadoop: se introduce el framework Hadoop, se presentan las implementaciones y las pruebas.

Capítulo 10. Conclusiones: presentamos las conclusiones del trabajo

Capítulo 11. Manual de usuario: explicamos con detalle como utilizar los programas.

2. Big Data

Comencemos nuestro camino estudiando el Big Data. En el mundo actual la cantidad de información digitalizada es enorme ya que prácticamente toda la información que se genera se digitaliza. Tal y como indica la OCDE [2] en sus informes cada día aproximadamente se generan varios Exabytes de datos. Todos estos datos provienen de múltiples actividades que van desde las meramente comerciales a las científicas pasando por las de educación, gubernamentales, defensa y un largo etc.

El reto ante tal cantidad de datos es procesarlos para extraer de ellos nuevo conocimiento y aquí es donde encaja el concepto de Big Data. Este descubrimiento de nuevo conocimiento no es sencillo ya que requiere de técnicas ad hoc creadas a partir de la necesidad de tratar con ingentes cantidades de datos. Pensemos por ejemplo que nos vamos a encontrar con datos estructurados, semiestructurados y no estructurados que hay que manejar con herramientas de computo muy poderosas y aquí es donde juega su papel el Cloud Computing.

Para destacar la importancia de Big Data podemos ver en la figura 1 los diversos ámbitos de estudio de un proyecto de la OCDE [2] para estudiar el impacto de Big Data en diversos sectores de la sociedad y cómo obtener beneficio de ello minimizando sus riesgos.

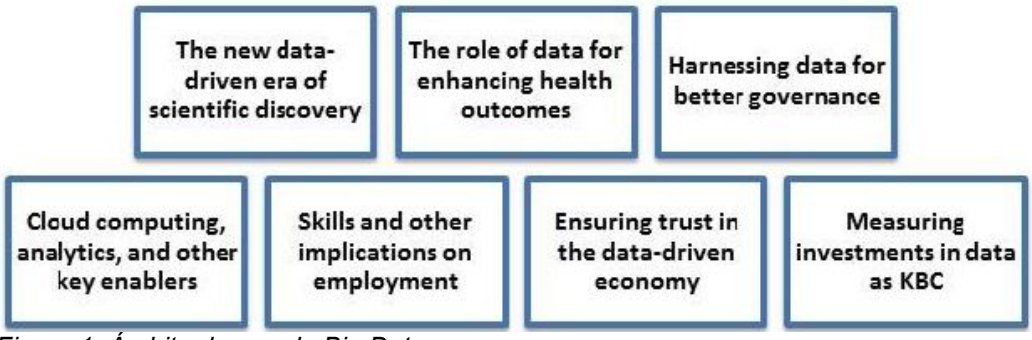


Figura 1. Ámbito de uso de Big Data

Tal y como nos indica IBM [3] en la figura 2, Big Data tiene cuatro dimensiones a tener en cuenta

1. Volumen: "el volumen hace referencia a las cantidades masivas de datos que las organizaciones intentan aprovechar "

2. Variedad: "la variedad tiene que ver con gestionar la complejidad de múltiples tipos de datos, incluidos los datos estructurados, semiestructurados y no estructurados"

3. Velocidad: "el tiempo de espera entre el momento en el que se crean los datos, el momento en el que se captan y el momento en el que están accesibles"

4. Veracidad: "nivel de fiabilidad asociado a ciertos tipos de datos"

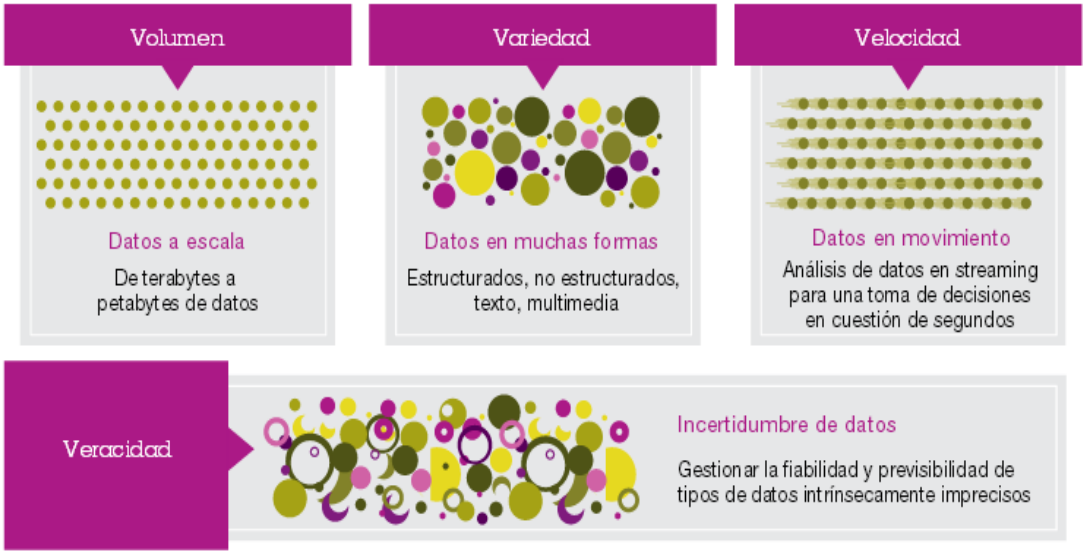


Figura 2. Dimensiones de Big Data

Podemos observar en la figura 3 una comparación que ilustra el posicionamiento de distintos sectores para crecer gracias al Big Data en los Estados Unidos según [4]. Esto denota el gran potencial para genera riqueza que tiene esta nueva manera de entender la información.

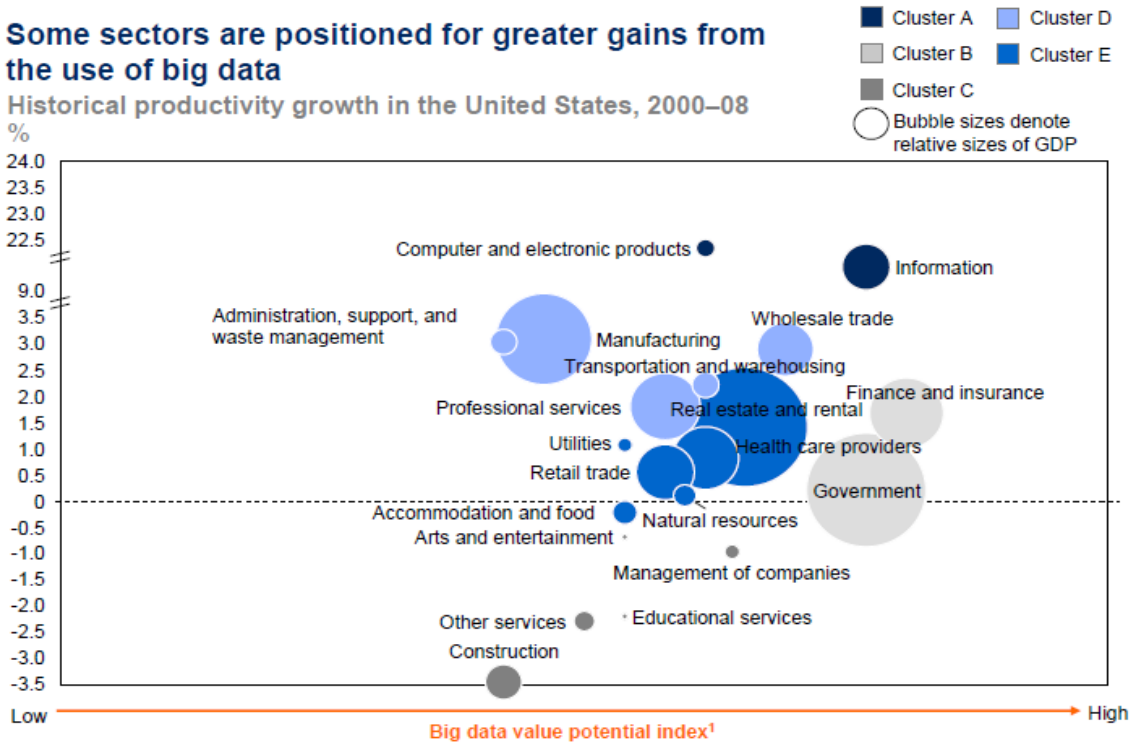


Figura 3. Sectores con potencial de crecimiento gracias al Big Data

La procedencia de esos datos también es algo importante, según nos indica [5] las fuentes fundamentales de datos generados la podemos ver en la figura 4.

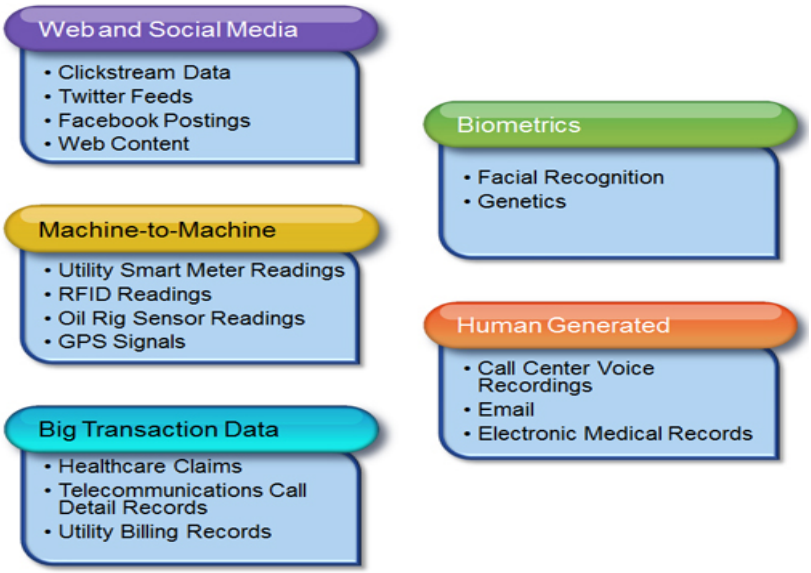


Figura 4. Procedencia de datos para Big Data

- *Web and Social Media*: contenido web e información de redes sociales, blogs, etc.
- *Machine-to-Machine (M2M)*: información de dispositivos como sensores que miden velocidad, temperatura, presión, variables meteorológicas, variables químicas, etc.
- *Big Transaction Data*: registros de facturación, registros llamadas (CDR), etc.
- *Biometrics*: Información biométrica que incluye huellas digitales, escaneo de la retina, reconocimiento facial, genética, etc.
- *Human Generated*: llamadas telefónicas, notas de voz, correos electrónicos, documentos electrónicos, estudios médicos, etc.

En la siguiente figura podemos ver ejemplos de alguna de las procedencias de la marea de datos que recorre el mundo

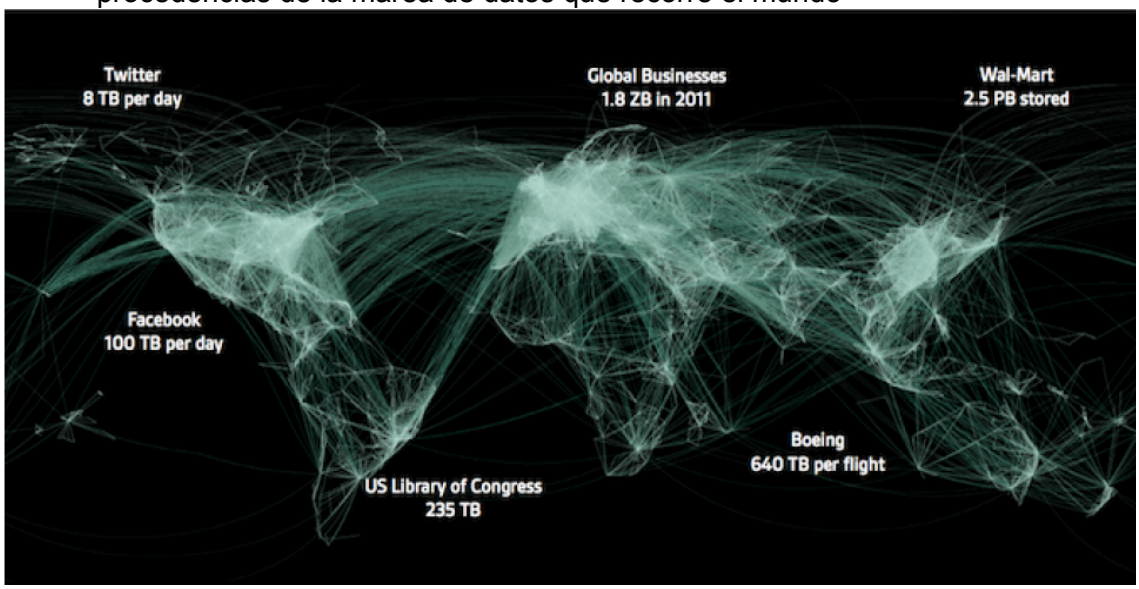


Figura 4 bis. Marea de datos

Una idea de como la tendencia al uso masivo de datos, a partir de los cuales poder aplicar Big Data, es firme en su crecimiento la podemos ver en el informe VNI Mobile Forecast Highlights, 2015-2020 [6] de CISCO donde descubrimos por ejemplo como el tráfico mundial de datos:

- se multiplicará por ocho en el periodo 2015 a 2020
- alcanzará los 30.6 Exabytes por mes en 2020
- será 120 veces mayor en 2020 que una década antes
- el 98% del tráfico de datos será “tráfico inteligente” en 2020
- si lo dividimos entre los usuarios finales el ratio será de aproximadamente 3.5 Gigabytes por usuario y mes.

3. Lógica Difusa

Veamos ahora una introducción a la lógica difusa y démonos cuenta como en la vida diaria es habitual oír frases como:

- “hace calor”
- “está cerca”
- “no era tarde”
- “mantenerlo en el horno hasta que se dore”

Todas estas frases tiene algo en común, la imprecisión. A poco que nos fijemos nos damos cuenta de ella ¿qué es calor? ¿qué es cerca? ¿qué es tarde? ¿qué es dorado? Bien, todo depende del universo de discurso, es decir, para una persona en un momento dado el concepto “calor” no tiene que ser lo mismo que para otra persona.

En la lógica tradicional sólo disponemos de dos posibles valores, True y False, y así a cada sentencia tan solo podemos hacerle corresponder uno de estos dos valores. Es decir, la frase “hace calor” será cierta o falsa, no hay más opción ¿Y cómo le asignamos ese valor? pues estableciendo un límite, digamos que a partir de 30 grados hacer calor, en otro caso no hace calor.

Pero claro, cabe pensar ¿y qué pasa si la temperatura es de 29.9 grados? Nuestra lógica tradicional dirá que no hace calor, pero en realidad está mucho más cerca de hacer calor que de no hacerla. De igual manera la imprecisión está presente en el concepto “cerca” , “tarde” o “dorado”.

Para abordar este tipo de problemas de imprecisión lingüística L.A. Zadeh introdujo los conjuntos difusos [7] en 1965 y a partir de ahí se ha ido desarrollando toda una teoría de conjuntos y lógica difusa para modelar matemáticamente incertidumbre.

En la actualidad tal y como se indica en [8] la lógica difusa se utiliza en diversos ámbitos tales como:

- En procesos complejos, si no existe un modelo de solución sencillo.
- En procesos no lineales.
- Cuando haya que introducir la experiencia de un operador “experto” que se base en conceptos imprecisos obtenidos de su experiencia.
- Cuando ciertas partes del sistema a controlar son desconocidas y no pueden medirse de forma fiable.
- Cuando el ajuste de una variable puede producir el desajuste de otras.
- En general, cuando se quieran representar y operar con conceptos que tengan imprecisión o incertidumbre.

Ejemplos de su uso son los sistemas expertos, las bases de datos, el reconocimiento de patrones, la visión por ordenador o los sistemas de control.

Tal y como dice Zadeh en [7] “un conjunto difuso es una clase de objetos con un grado de pertenencia continuo. Tales conjuntos se caracterizan por una función de pertenencia, función que asigna a cada objeto un grado de pertenencia en el rango de cero a uno”.

Ilustremos este concepto con un ejemplo [9], imaginemos que se establece un universo de discurso donde a partir de 1.80m una persona es alta.

En la lógica clásica al conjunto '**personas altas**' solo pertenecen aquellas personas con altura mayor o igual a 1.80m. Así pues, para saber si una persona es alta hay que aplicar una función que devolverá True (1) si la altura es mayor de 1.80m o False (0) si no lo es. Esta función la podemos dibujar.

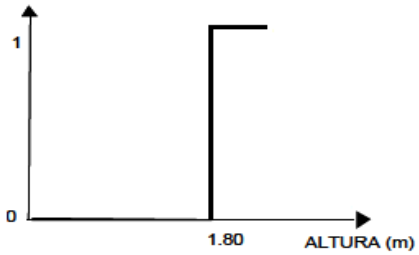


Figura 5. Función discreta

¿Qué pasa si una persona mide 1.79m? Pues claramente no es alto según este modelo ya que le corresponde el valor 0 (False) y evidentemente esta persona está más cerca de ser alto que de no serlo.

Si este mismo conjunto lo fuzzificamos (hacemos difuso) tendremos la siguietne gráfica

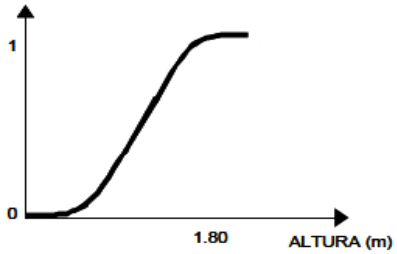


Figura 6. Función difusa

Y aquí ya vemos como a una persona de 1.79m se le asigna un valor muy próximo a 1 lo que indica que está bastante cerca de ser alto.

Imaginemos que ahora disponemos de tres conjuntos de personas **Altas** (más de 1.8m)**, Medias** (de 1.7m a 1.8m)y **Bajas** (menos de 1.7m) [10]**.**

Lo podemos visualizar así con una lógica clásica

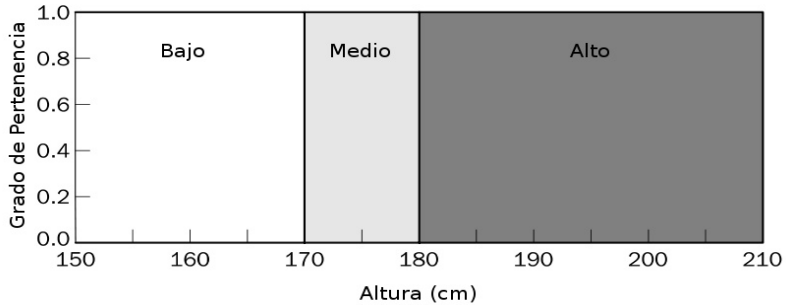


Figura 7. Grados de pertenencia discretos

y así con una lógica difusa

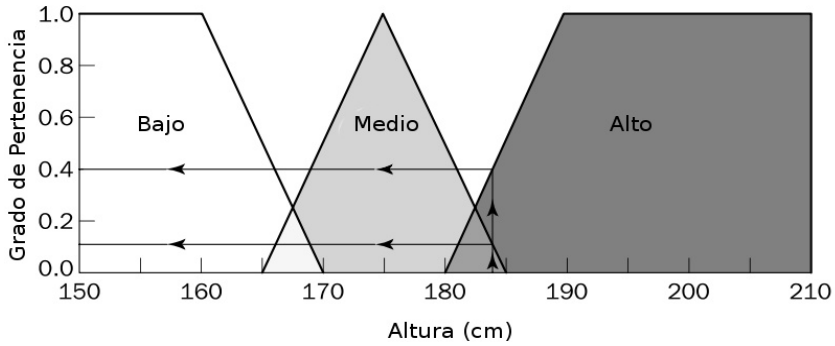


Figura 8. Grados de pertenencia difusos

Observemos como en este último dibujo hay una transición suave entre límites.

Veamos en la siguiente tabla la asignación al conjunto **Persona Alta** que hace la función de pertenencia en lógica clásica (nítida o crisp) y la que hace en la lógica difusa (fuzzy)

Nombre	Altura	Crisp	Fuzzy
Paco	2.05	1	1.0
Juan	1.95	1	1.0
Tomás	1.87	1	0.95
Carlos	1.80	1	0.82
Pedro	1.79	0	0.71
Andrés	1.60	0	0.36

Es claro que Pedro se acerca bastante a ser alto a pesar de medir menos de 1.80m y que Andrés se aleja mucho.

Visto lo anterior vamos a definir matemáticamente los conjuntos difusos, para ello definiremos la función de pertenencia.

Para un conjunto nítido A la función de pertenencia de un elemento x la podemos definir así

$$\mu_A(x) = \begin{cases} 0 & \text{si } x \notin A \\ 1 & \text{si } x \in A \end{cases}$$

Se trata pues de una función escalón centrada en sus valores umbrales

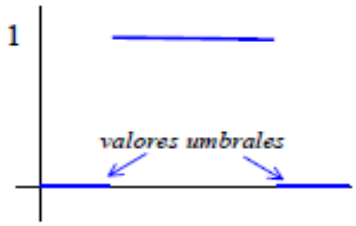


Figura 9. Función escalón

Por lo tanto lo podemos ver así [12]

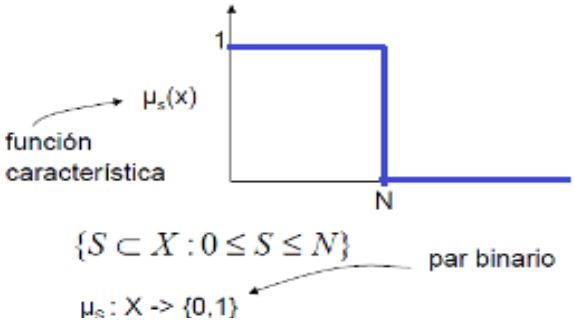


Figura 10. Función de pertenencia nítida

Para un conjunto difuso A la función de pertenencia de un elemento x la podemos definir como un conjunto de pares donde a cada valor de x se le asigna su grado de pertenencia a A. Usando la notación de [7].

$$A=\{(x,\mu A(x)) / x \in U\}$$

Que podemos ver así [12]

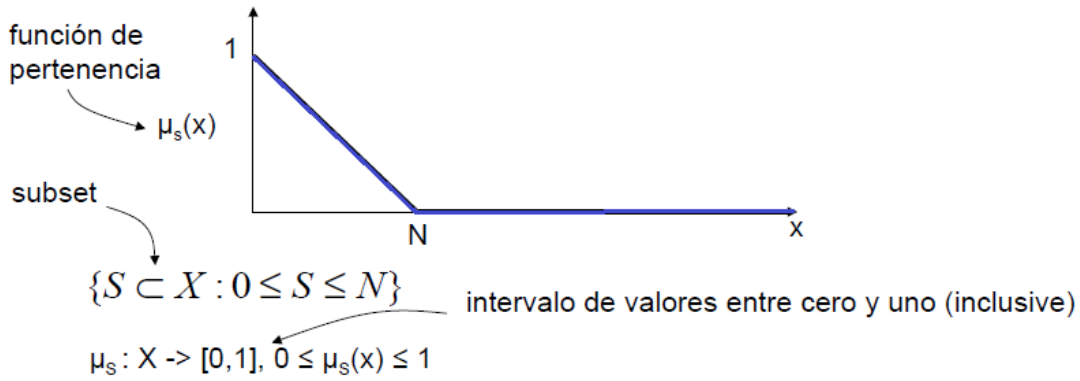


Figura 11. Función de pertenencia difusa

Esta función de pertenencia puede ser cualquiera en teoría pero en la práctica hay algunas que se suelen repetir asiduamente debido a su fácil computo y a que se acomodan perfectamente a la mayoría de los valores lingüísticos que se quieren representar. Veamos algunas de las más habituales [11].

Función Gamma

$$\mu(x) = \begin{cases} 0 & \text{si} \quad x \le a \\ (x-a)/(m-a) & \text{si} \quad x \in (a,m) \\ 1 & \text{si} \quad x \ge m \end{cases}$$

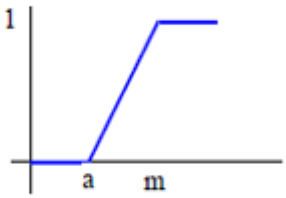


Figura 12. Función Gamma

Función L= 1 - Gamma

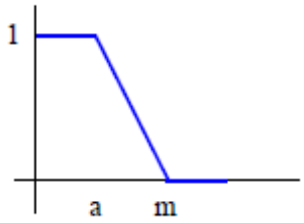


Figura 13. Función L

Función Lambda o triangulo

$$\mu(x) = \begin{cases} 0 & \text{si } x \le a \\ (x-a)/(m-a) & \text{si } x \in (a,m] \\ (b-x)/(b-m) & \text{si } x \in (m,b) \\ 0 & \text{si } x \ge b \end{cases}$$

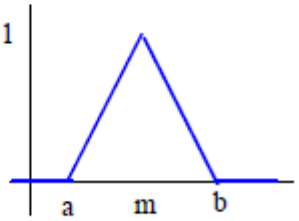


Figura 14. Función Lambda

Función Trapezoide

$$\mu(x) = \begin{cases} 0 & \text{para } x \le a \\ \dfrac{x-a}{b-a} & \text{para } a < x \le b \\ 1 & \text{para } b < x \le c \\ \dfrac{d-x}{b-c} & \text{para } c < x \le d \\ 0 & \text{para } x > d \end{cases}$$

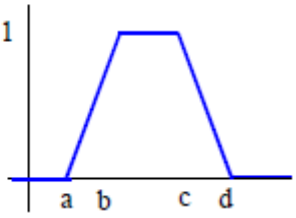


Figura 15. Función Trapezoide

Función S

$$\mu_S(x) = \begin{cases} 0 & \text{para } x \le a \\ 2\left(\frac{x-a}{c-a}\right)^2, & \text{para } a \le x \le \frac{a+c}{2} \\ 1-2\left(\frac{x-a}{c-a}\right)^2, & \text{para } \frac{a+c}{2} \le x \le c \\ 1 & \text{para } x \ge c \end{cases}$$

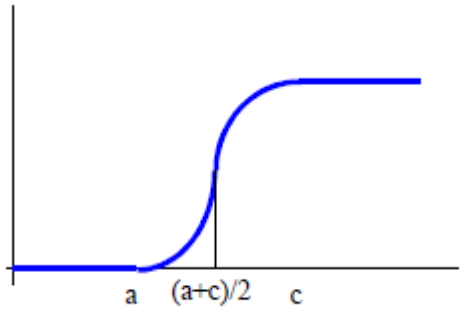


Figura 16. Función S

Función z = 1 – s

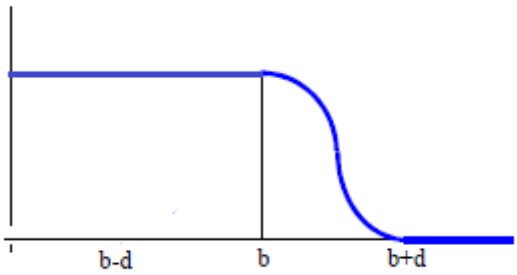


Figura 17. Función Z

Función Л

$$\mu_\Pi(x) = \begin{cases} \mu_S(x) & \text{para } x \le b \\ \mu_Z(x) & \text{para } x > b \end{cases}$$

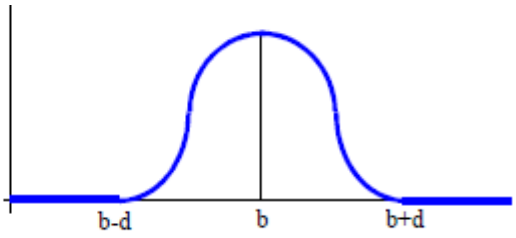


Figura 18. Función Л

Veamos como una función L puede representar el nivel de frío [12]

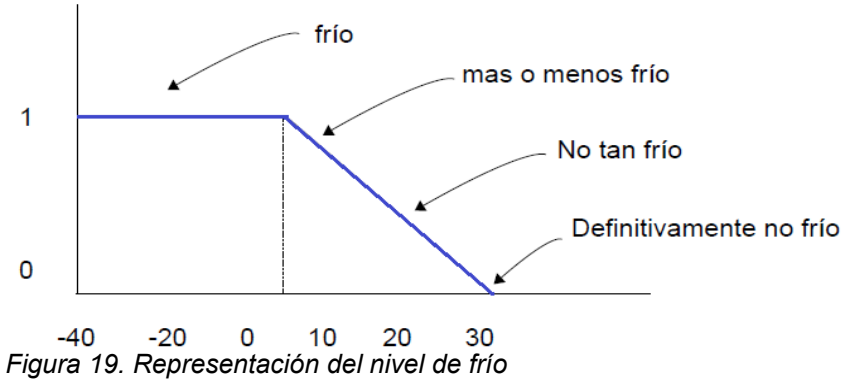


Figura 19. Representación del nivel de frío

Al igual que con los conjuntos nítidos, se pueden realizar las mismas operaciones con los conjuntos difusos. Veamos las tres más comunes [10]

La unión de A y B es un conjunto difuso C tal que:

$$C(x) = Max[A(x),B(x)]$$

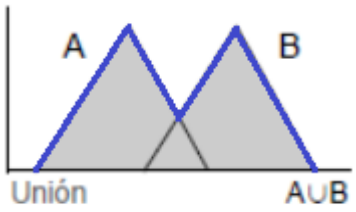


Figura 20. Unión

La intersección de A y B es un conjunto difuso C tal que:

$$C(x) = Min[A(x),B(x)]$$

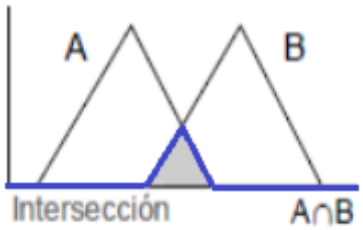


Figura 21. Intersección

El complemento de A es un conjunto difuso C tal que:

$$C(x) = 1 – A(x)$$

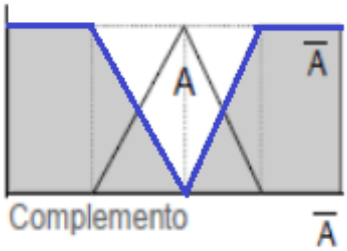


Figura 22. Complemento

Una vez vista esta pequeña introducción al Big Data y a la fabulosa lógica difusa estamos en condiciones de seguir caminando hacia nuestro objetivo, el estudio de un algoritmo de clustering difuso distribuido, así que toca introducirnos en el mundo del clustering o agrupamiento.

4. Aprendizaje No Supervisado. Clustering

Vemos en el diccionario de la RAE [13] en su tercera entrada que el aprendizaje es:

- *"la adquisición por la práctica de una conducta duradera"*

El llevar este concepto a la computación ha dado lugar a múltiples definiciones pero en general se acepta que se trata de conseguir técnicas que permitan a las computadoras mejorar sus resultados o habilidades a partir de la experiencia y entender este proceso como un aprendizaje [14].

Es decir que adquieren nuevo conocimiento ya que también nos dice la RAE [13] que aprender es:

- "*Adquirir el conocimiento de algo por medio del estudio o de la experiencia"*

Disponemos básicamente de dos métodos para adquirir nuevo conocimiento, que son:

- Deducción
- Inducción

La deducción consiste en ir de lo general a lo particular, a partir de premisas verdaderas llegar una conclusión verdadera, no admite grados de veracidad.

$A \rightarrow B$	Si llueve se moja la calle
A	llueve
B	se moja la calle

La inducción es el proceso contrario, de lo particular a lo general, a partir de hechos concretos suponemos una regla general, esta conclusión admite grados de veracidad.

$A \rightarrow B$	Si llueve se moja la calle
B	La calle está mojada
A	ha llovido

Como vemos en este caso la conclusión no tiene por qué ser siempre cierta, de ahí que admita grados. Muy probablemente la calle esté mojada por la lluvia pero también puede ser que alguna vez esté mojada por otra razón, como por haber sido regada. Esto nos lleva a que las hipótesis pueden ser refutadas pero nunca confirmadas.

Y es en el aprendizaje por inducción donde precisamente podemos colocar al Big Data, a partir de los datos recogidos de la realidad pretendemos conseguir nuevo conocimiento, es decir, aprender.

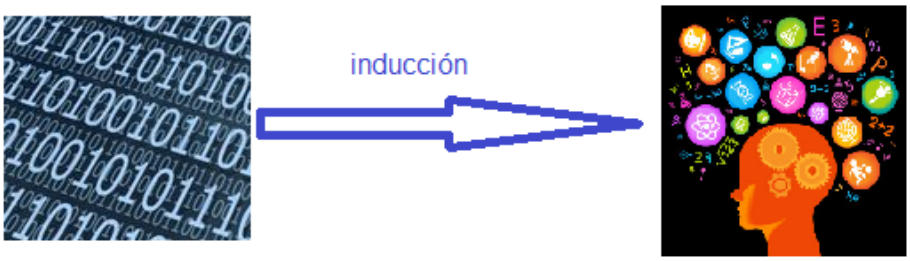


Figura 23. Inducción. Elaboración propia

Este nuevo conocimiento será válido si alcanza una probabilidad de ser cierto aceptable.

Ilustremos esto para dejarlo más claro, acabamos de mencionar que la inducción admite grados de verdad, es decir, a cada nuevo conocimiento inducido le podemos asignar una probabilidad de ocurrencia.

Imaginemos ahora que descubrimos por inducción que en un negocio el 60% de la veces que se compran el artículo A también se compra el artículo B. Podemos pues afirmar que la probabilidad de que se compre el artículo B dada la compra de A es del 0.6.

Parece bastante sensato pensar que existe un relación en este ámbito entre ambos artículos y en consecuencia podemos dar este nuevo conocimiento como válido.

Sin embargo si esta probabilidad fuese de 0.01 también parece bastante sensato pensar que la relación no es tal y que esto no nos aporta un nuevo conocimiento válido.

El aprendizaje por inducción lo podemos clasificar de la siguiente manera [15]

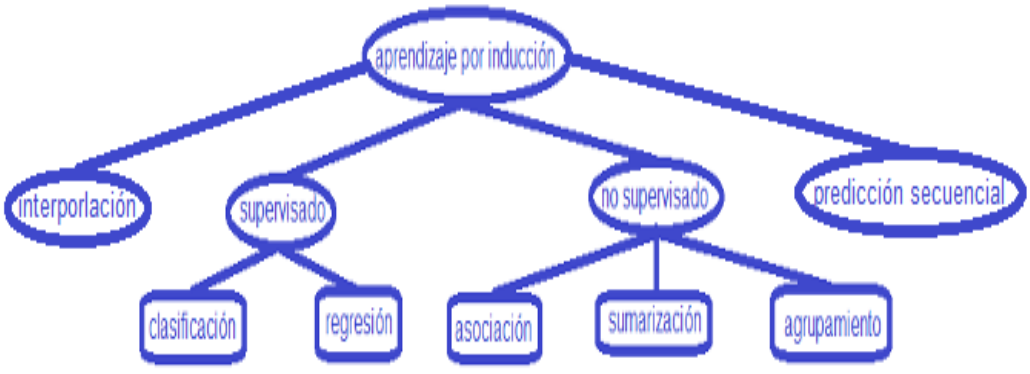
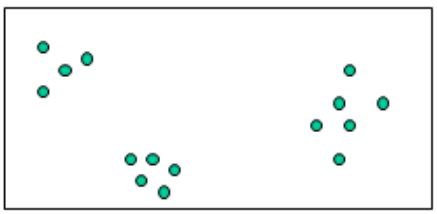


Figura 24. Clasificación del aprendizaje por inducción. Elaboración propia

Veamos esta clasificación más detalladamente.

TECNICA	OBJETIVO Y EJEMPLO
Interpolación	Obtener nuevos puntos a partir de puntos ya conocidos en una función continua  *Figura 25. Interpolación*
Predicción	Predecir el siguiente valor en una serie 1, 2, 3, 5, 7, 11, 13, 17, 19, … ?

TECNICA	OBJETIVO Y EJEMPLO
Clasificación	Obtener la salida esperada para clases discretas Determinar si un objeto es grande o pequeño.
Regresión o Estimación	Obtener la salida esperada para clases continuas Estimar la altura media de una persona a partir de su familia

TECNICA	OBJETIVO Y EJEMPLO
Sumarización	Mostrar los datos de una manera resumida o más simple buscando características comunes el 90% de los clientes son mayores de 25 años
Asociación	Descubrir hechos que ocurren juntos dentro del conjunto {leche, pan} → {arroz} 70%
Clustering o Agrupamiento	Buscar grupos de objetos con semejanzas  ¿Cuántos grupos hay? ¿Qué grupos formo? *Figura 26. Agrupamiento*

CLUSTERING

Como acabamos de decir se trata de obtener agrupaciones de tal manera que los datos dentro del grupo sean lo más parecidos posible y los grupos sean lo más diferentes posibles. Para ello se intenta minimizar la distancia dentro del grupo y maximizar la distancia entre grupos tal y como se describe en [16].

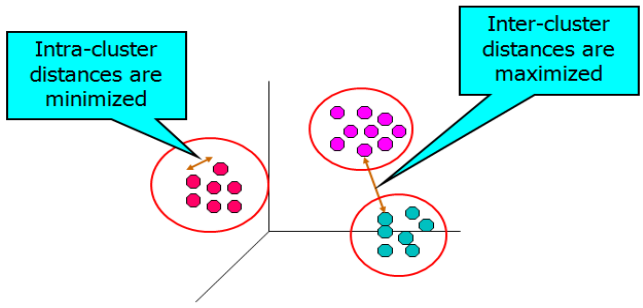


Figura 27. Clustering

Su utilidad es enorme, pensemos en algo tan familiar como agrupar precipitaciones [16]

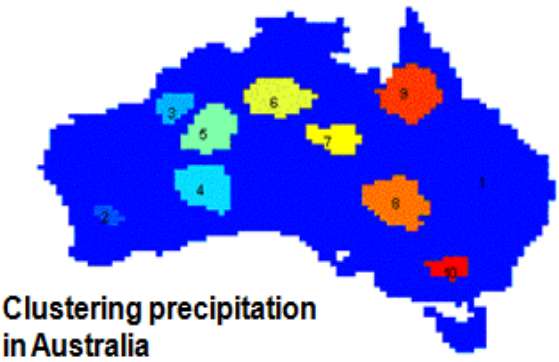


Figura 28. Clustering a partir de Precipitaciones

O en encontrar hoteles similares utilizando las caras de Chernoff [17]

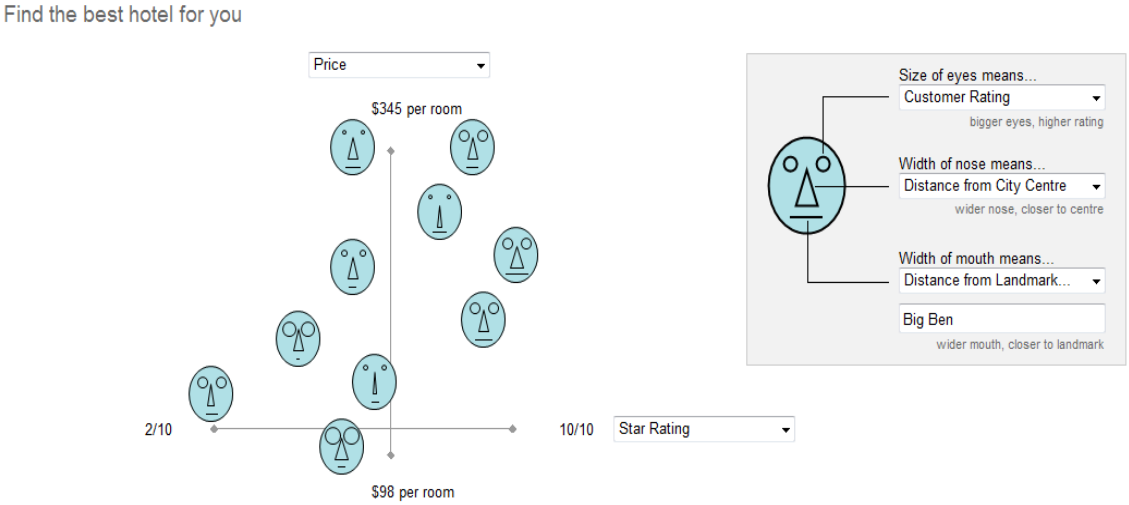


Figura 29. Clustering con caras de Chernoff

Se pueden visualizar en dos dimensiones

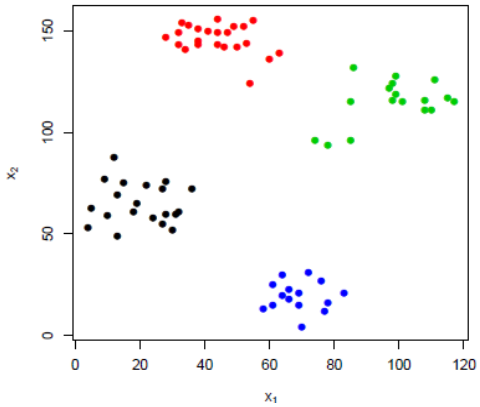


Figura 30. Visualización 2D

O en tres

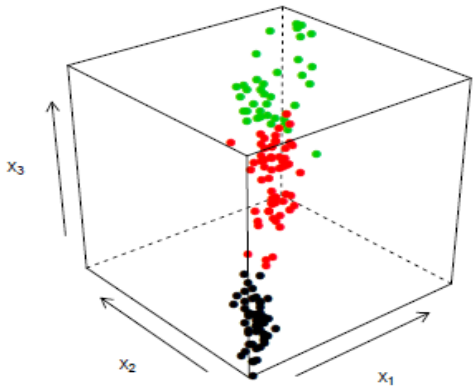


Figura 31. Visualización 3D

La noción de clustering puede ser ambigua tal y como figura en [16] ya que a veces no es claro por qué características agrupar o cuántos grupos hacer.

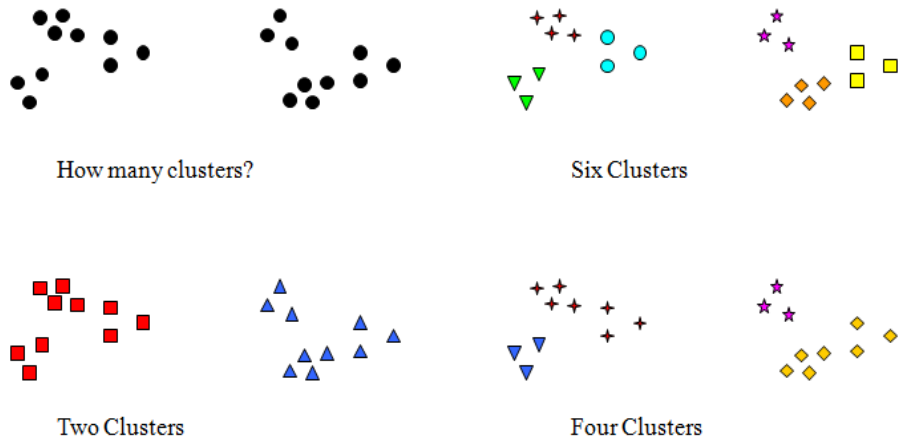


Figura 32. Ambigüedad en los clusters

Y vemos que nos podemos encontrar con distintos tipos de clusters [16], como:

Clusters bien separados donde cada punto de un cluster está más cerca de todos los puntos de su grupo que de cualquier otro punto fuera del grupo

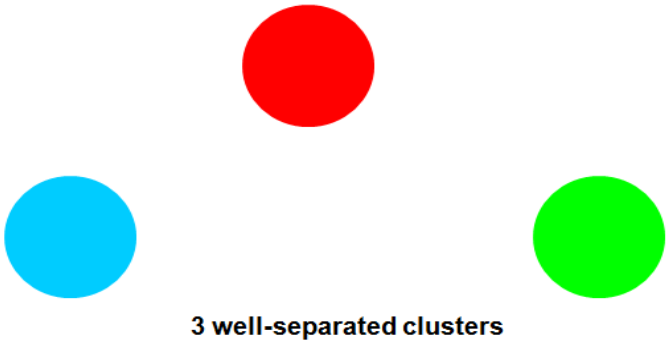


Figura 33. Clusters bien separados

Clusters basados en centros donde cada punto del grupo está más cerca de su centro que de cualquier otro centro. Ese centro suele ser la media o la mediana del grupo

Figura 34. Clusters basados en centros

Clusters continuos donde cada punto del grupo está más cerca de otro punto del grupo que de cualquier punto fuera del grupo.

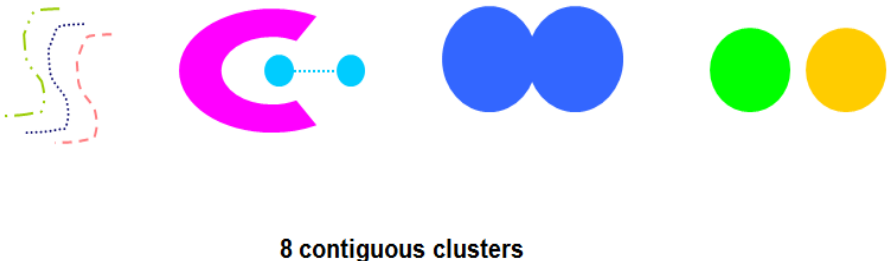


Figura 35. Clusters continuos

Clusters con características o propiedades compartidas

Figura 36. Clusters con propiedades compartidas

Sin olvidar los clusters con obstáculos [18] donde hay zonas del espacio que no pueden ser ocupadas por los objetos

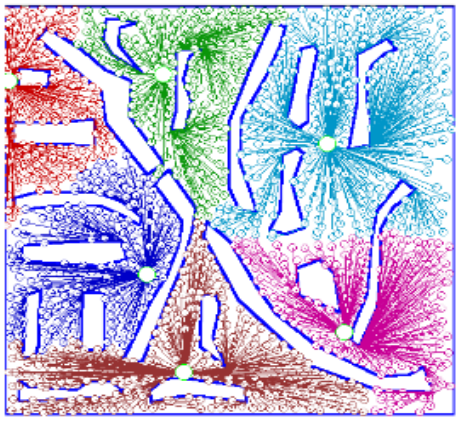


Figura 37. Cluster con obstáculos

Ni por supuesto los cluster difusos [19] donde los objetos tienen un grado de pertenencia a todos los grupos.

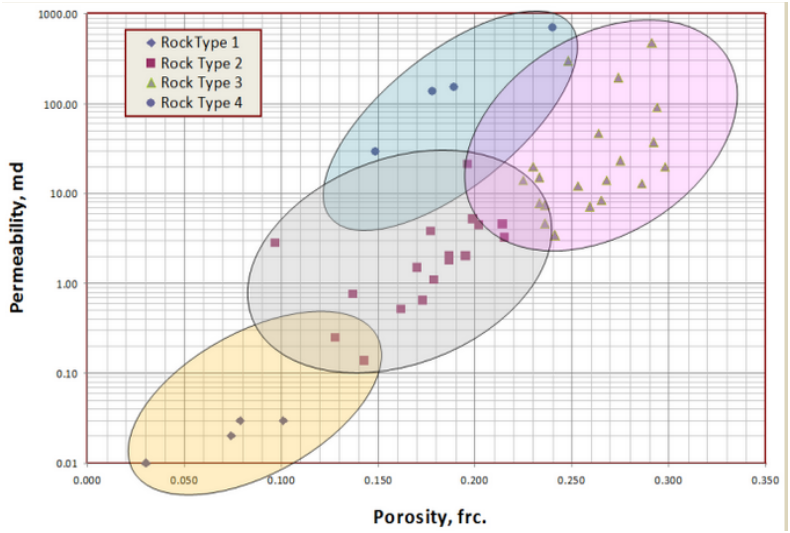


Figura 38. Clusters difusos

Para realizar los cluster disponemos de distintos métodos como son, entre otros, los siguientes:

Métodos basados en particiones donde a priori se conoce el número de particiones o grupos que se desean. Ejemplo con dos grupos [16]

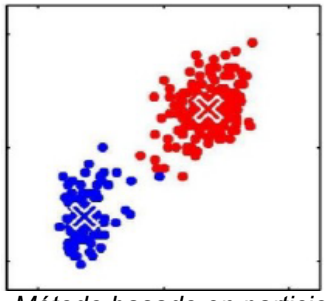


Figura 39. Método basado en particiones

Métodos jerárquicos donde se crea una estructura jerárquica para representar las agrupaciones, esta puede ser de abajo-arriba (agregando) o de arriba-abajo (dividiendo). Se genera un dendograma. Ejemplo donde se han generado tres grupos [19]

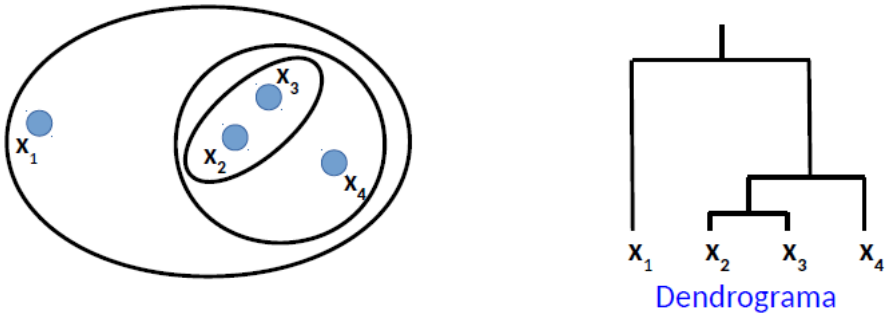


Figura 40. Método jerárquico

Métodos basados en densidad que hacen crecer los clusters siempre cuando la densidad en el entorno del objeto supere un umbral. Permite crear grupos de diversas formas creando regiones de alta densidad de objetos separadas por zonas de baja densidad [16]

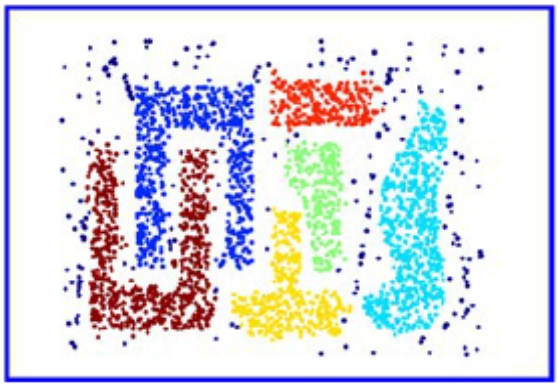


Figura 41. Método basado en densidad

Métodos basados en modelos: se utilizan cuando podemos suponer que los datos siguen una determinada distribución estadística. Ejemplo donde suponemos que siguen una distribución Normal o Gaussiana [19]

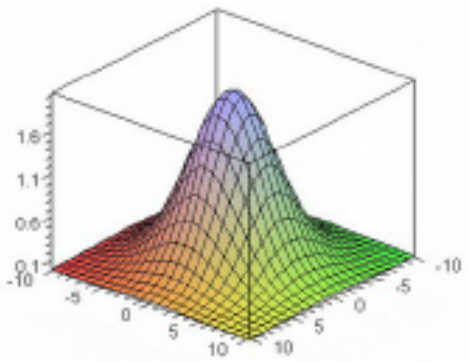


Figura 42. Cluster siguiendo distribución normal en 3D

Veamos un cluster en dos dimensiones

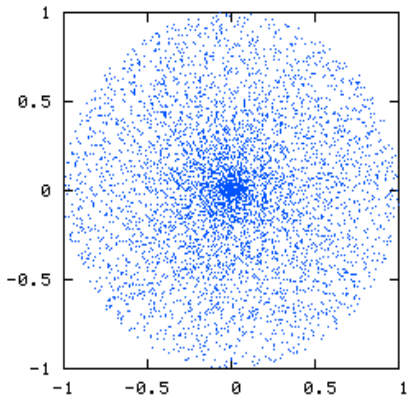


Figura 43. Cluster siguiendo distribución normal en 2D

Y ahora dos grupos

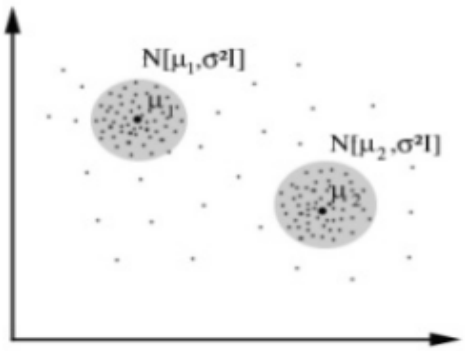
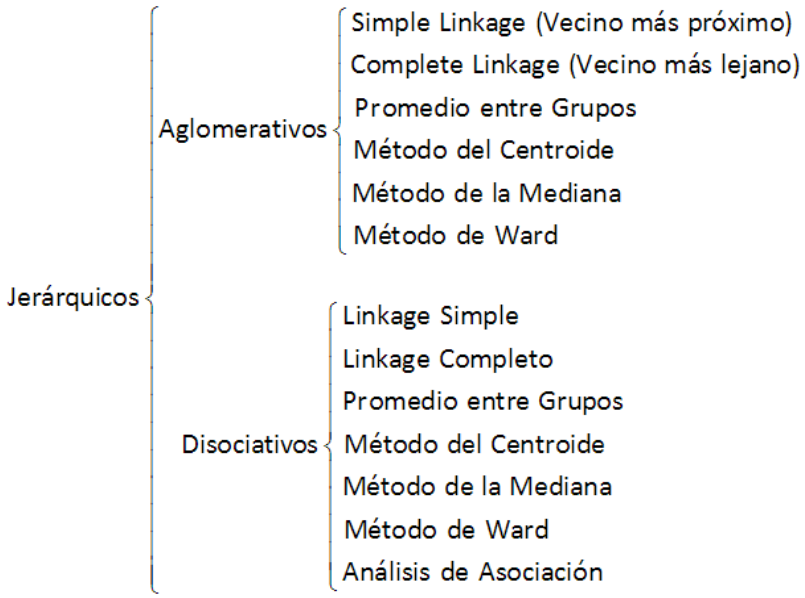
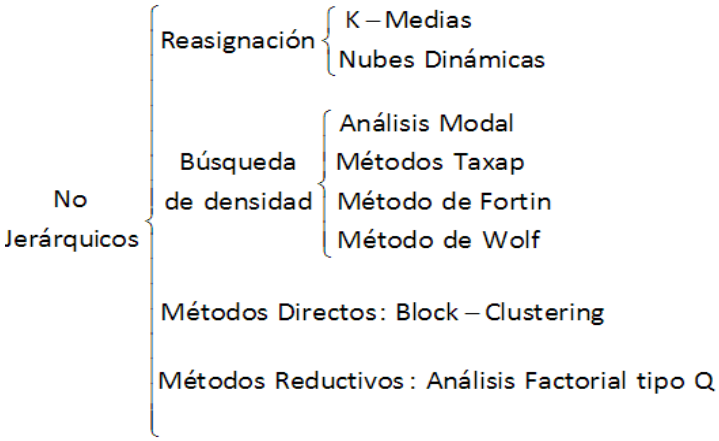


Figura 44. Dos clusters con distribución normal

Estos métodos los podemos especificar tal y como se hace en [20]





Además para poder realizar las agrupaciones es necesario tener medidas que permitan caracterizar de alguna manera las relaciones entre los objetos [20]. La elección de la medida concreta es fundamental a la hora de obtener una buena agrupación. Y las medidas más usuales son:

Distancia Euclídea: $d(x_i, x_j) = \sqrt{\sum_{c=1}^{p} (x_{ic} - x_{jc})^2}$

Distancia de Minkowski: $d_q(x_i, x_j) = \left(\sum_{c=1}^{p} \left| x_{ic} - x_{jc} \right|^q \right)^{\frac{1}{q}}$ donde $q \geq 1$

Distancia d_1 o ciudad (City Block): $d(x_i, x_j) = \sum_{c=1}^{p} \left| x_{ic} - x_{jc} \right|$

Distancia de Tchebychev o del máximo $(q = \infty)$**:** $d_\infty e(x_i, x_j) = \text{máx}(c = 1, \cdots, p) \left| x_{ic} - x_{jc} \right|$

Distancia de Mahalanobis: $D_S(x_i, x_j) = \sqrt{(x_i - x_j)' \, S^{-1} (x_i - x_j)}$

Distancia χ^2: $\chi^2 = m \left[\sum_{i=1}^{p} \sum_{j=1}^{q} \frac{n_j^2}{m_{i\bullet} \, m_{\bullet j}} - 1 \right]$

Ya visto todo lo anterior y teniendo una idea clara de qué es el Big Data, la Lógica Difusa y el Clustering es tiempo de acercarnos al algoritmo concreto que deseamos estudiar y adaptar al mundo de la nube, el Fuzzy C-Means.

Algoritmo que como no podía ser de otra manera es difuso, como su nombre indica, es utilizado en técnicas de Inteligencia Artificial para hacer agrupaciones difusas y obviamente en Big Data donde como ya hemos visto el Clustering es muy útil.

5. Fuzzy C-Means

Fuzzy C-Means o C-Means es uno de los algoritmos para clustering de partición difusa más difundido. Se utiliza en múltiples ámbitos que van desde las ciencias sociales hasta la ingeniería pasando por la ciencia básica. Usual en el reconocimiento de patrones, la segmentación de imagines, agrupamiento de imágenes, Big Data y un largo etc.

La idea como ya sabemos es la obtener particiones difusas a partir de un conjunto de datos. Podemos visualizarlo tal y como se hace en [21]

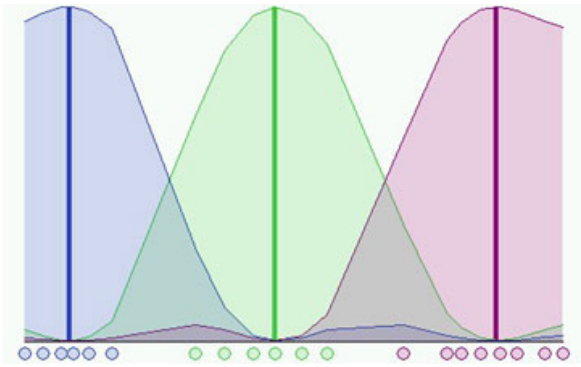


Figura 45. Visualización de particiones difusas

Observemos como estos veinte objetos unidimensionales han acabado agrupados de manera difusa de tal manera que el grado de pertenencia de un objeto decrece conforme nos alejamos del grupo y se incrementa conforme nos acercamos al centro de dicho grupo.

Como ejemplo real vamos a poner un experimento encontrado en [22] de segmentación de imágenes.

Aquí tenemos la imagen PET de un cerebro normal a la izquierda y de un cerebro con Alzheimer a la derecha del que contamos con 266 muestras.

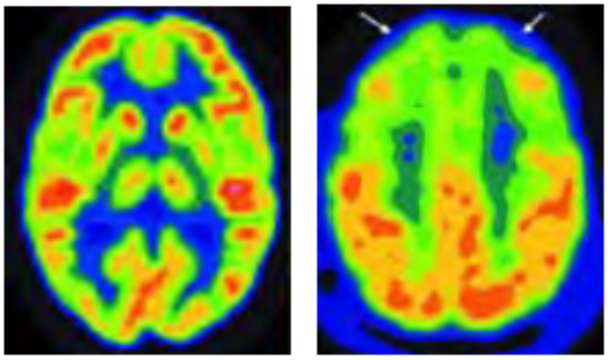


Figura 46. Cluster con metabolismo de la glucosa

Disponemos de 5 grupos, correspondiendo a cada uno un color en la imagen, que representan el nivel del metabolismo de la glucosa.

El algoritmo fué introducido por Ruspini [23] y más tarde ampliado por Dunn [24] y Bezdek [25]. Vamos a describirlo basándonos en definición que hace Bezdek en [26]

Sea **X** un conjunto de **N** objetos, se dice que una partición

P = {C1, C2, …, Cc}

es una partición difusa de **X** si y solo si se cumple que

$$\begin{cases} \sum_{i=1}^{C} u_{ij} = 1,\ j = 1,2,...,N & (1) \\ 1 \geq u_{ij} \geq 0,\ i = 1,2,...,C,\ j = 1,2,...,N & (2) \\ n > \sum_{j=1}^{N} u_{ij} > 0,\ i = 1,2,...,C & (3) \end{cases}$$

atendiendo a que **U**ij es el grado de pertenencia al cluster i del objeto j

Expliquemos esto:

- en (1) vemos como la suma de los grados de pertenencia de un objeto j a los distintos grupos ha de ser igual a 1.
- en (2) vemos como el grado de pertenencia de un objeto j a un cluster i deber estar ente 0 y 1
- en (3) vemos como la suma de todos los grados de pertenencia en un cluster tiene que ser mayor a 0 y menor que N es decir no podemos tener grupos vacíos ni un cluster con todos los elementos.

Evidentemente de lo que se trata es de buscar la mejor partición difusa posible, para ello hay distintas maneras, una de ellas es la de minimizar una función objetivo.

Concretamente la función objetivo a minimizar es la siguiente:

$$J_m(U, \boldsymbol{v}) = \sum_{k=1}^{n} \sum_{i=1}^{c} (u_{ik})^m \, (d_{ik})^2$$

donde ***U*** es la matriz de pertenencias conteniendo el grado de pertenencia de cada objeto a cada grupo. ***V*** es el vector de centros que contiene el centro de cada grupo. ***d*** indica la distancia cuadrada entre el objeto de **k** y el centro ***V*** del cluster i.

$$d_{ik}^2 = ||\boldsymbol{x}_k - \boldsymbol{v}_i||^2$$

m es el factor difuso que indicaría cuanto queremos que se solapen los grupos. Tiene que cumplirse **m >** 1 ya que la partición se vuelve más difusa conforme se incrementa **m** y con **m**=1 la partición dejaría de serlo. Varios estudios indican que el valor más practico está en el intervalo [2, 2.5].

Se trata pues de buscar la partición que haga mínima la distancia de los objetos al centro de su grupo. Esta distancia está ponderada por el grado de pertenencia de cada objeto a un cluster y por el factor difuso como no podía ser de otra manera.

Hay distintas maneras de minimizar esta función pero nosotros aquí al igual que Bezdek en [26] vamos a optar por minimizarla de manera iterativa usando el siguiente teorema:

> una partición difusa puede ser un mínimo local de la función objetivo **J**, para **m** > 1, cuando se cumplen las siguientes condiciones

$$u_{ik} = \frac{1}{\sum_{j=1}^{c} \left(\frac{d_{ik}}{d_{jk}} \right)^{2/(m-1)}}$$

Fórmula 1

$$v_i = \frac{\sum_{k=1}^{n} (u_{ik})^m \, x_k}{\sum_{k=1}^{n} (u_{ik})^m}$$

Fórmula 2

Tengamos en cuenta que || . || es la distancia entre el objeto de **X** y el centro **V** del cluster. Viene dada por una norma que normalmente es la distancia euclidiana pero que puede ser otra en función de como queramos agrupar los objetos, por ejemplo:

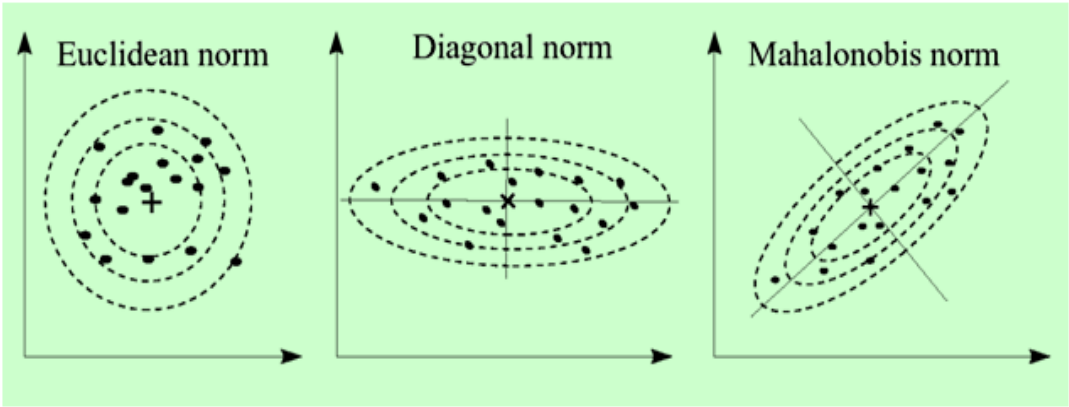


Figura 47. Distintas normas para la medida de la distancia

Con la norma euclidiana como vemos en la figura 47 se generan grupos circulares, con la norma diagonal grupos elípticos con los ejes paralelos al eje de abscisas y de ordenadas y con la norma de Mahalonobis también se generan grupos elípticos pero con los ejes en de manera arbitraria. De igual manera tenemos normas para grupos rectangulares, cilíndricos, etc.

Veamos el algoritmo que nos va a permitir buscar, con una estrategia iterativa, el mínimo de la función objetivo.

INICIO

1. Elegir el número de clusters 1 < ***C*** < N.

   //obviamente ha de ser mayor que 1 y menor que el número de objetos a agrupar

2. Elegir el exponente ***m*** > *1*

   //como hemos mencionado arriba ha de ser mayor que 1 estando su valor más practico entre [2, 2.5]

3. Elegir la norma (cómo medir la distancia)

   //un ejemplo habitual es la distancia euclidiana aunque caben muchas otras tal y como indicamos arriba

4. Elegir la tolerancia ***ε*** para terminar las iteraciones

   //el umbral de parada, normalmente entre [0.001, 0.01]

5. Generar la matriz de pertenencia ***U*** aleatoriamente.

   //matriz donde aparece la pertenencia de cada objeto a los distintos clusters

6. Mientras que la condición de parada sea falsa

   1. Calcular los nuevos centros ***V*** de los grupos

      //aplicando la fórmula 2

   2. Calcular las distancias ***d*** de los objetos a los centros

      //con la norma elegida en el paso 3

   3. Actualizar la matriz ***U*** de pertenencia

      //con la fórmula 1

   4. Comprobar la condición de parada

   Fin Mientras

FIN

La condición de parada suele ser la siguiente:

$$||U^{(b)} - U^{(b+1)}|| < \epsilon$$

que no es otra cosa que ver si la diferencia entre las pertenencias actuales y las anteriores es menor que el valor establecido en el paso 4.

También a veces se impone un número máximo de iteraciones.

6. Cloud Computing.

Acerquémonos ahora al mundo de la Nube una vez que ya conocemos el algoritmo que deseamos adaptar.

COMPUTACIÓN DISTRIBUIDA

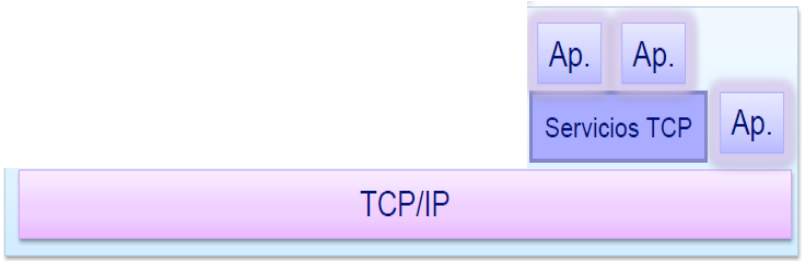
Tal y como vemos en [27] un sistema distribuido es un conjunto de computadores independientes interconectados a través de una red que colaboran con el fin de realizar una tarea.

- Es un sistema heterogéneo (arquitecturas, sistemas operativos, protocolos, ...)
- Diferentes tipos de redes

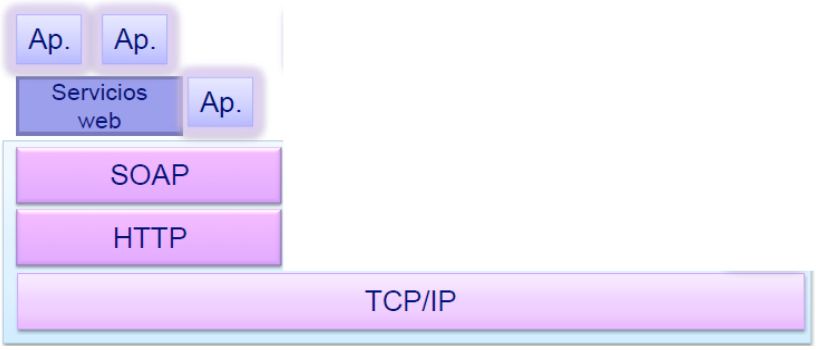
La computación distribuida es aquella que se ejecuta sobre un sistema distribuido. Los programas son bastante autónomos y acceden a la red cuando lo necesitan.

Podemos clasificar la computación distribuida en niveles:

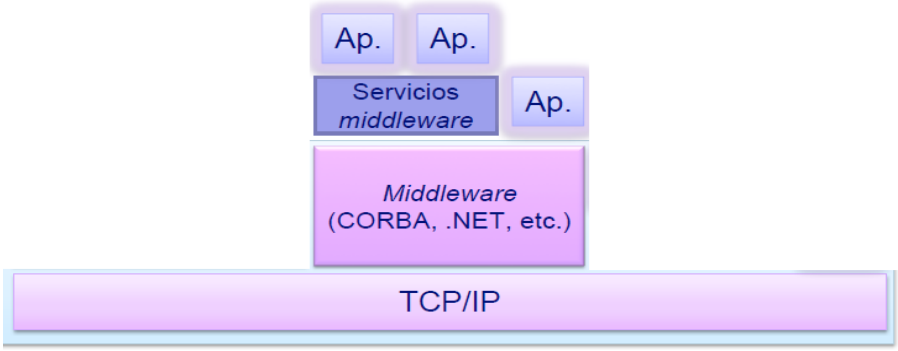
1. Servicios de red: capa de servicios estándar en Internet proporcionado por un servidor como HTTP, DNS, FTP, SMTP.



2. Servicios Web: conjunto de servicios ofertados en la web sobre el protocolo HTTP



3. Aplicaciones de red: aplicaciones específicas desarrolladas por usuarios finales



Estas aplicaciones necesitan de un software especial denominado middleware que es capaz de generar un bus software que proporciona interoperatividad entre objetos en el entorno distribuido [27].

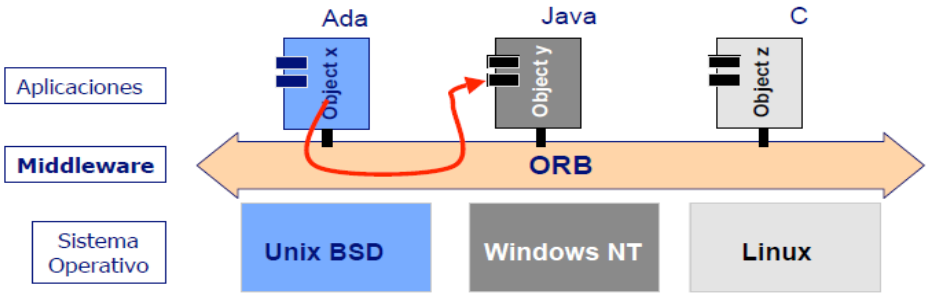


Figura 48. Middleware

podemos clasificar estas aplicaciones de red en

1. Aplicaciones sobre redes WAN: Aplicaciones web, Transacciones distribuidas, Aplicaciones multimedia

2. Aplicaciones sobre redes LAN: Aplicaciones industriales, Sistemas empotrados y para vehículos

Integrándolo todo tenemos la visión conjunta de los tres niveles en los que podemos dividir la computación distribuida

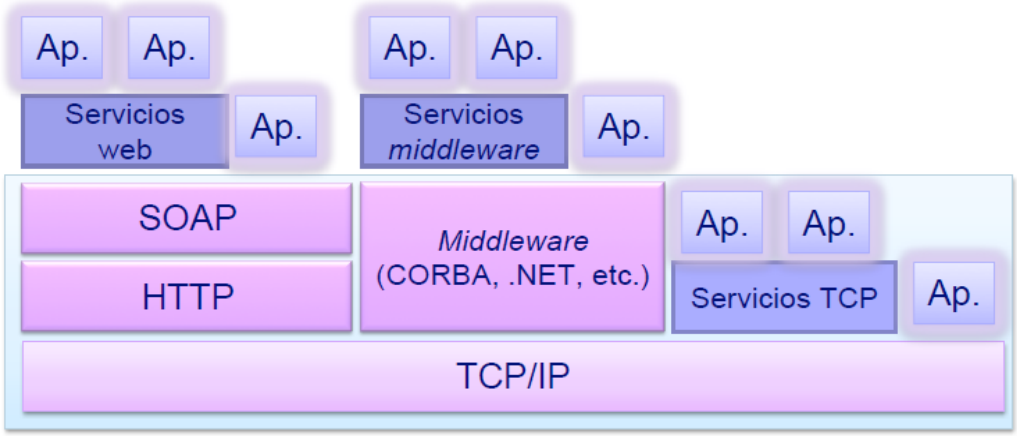


Figura 49. Computación distribuida

Este tipo de computación tiene evidentes ventajas

- Acceder y compartir recursos remotos
- Se puede repartir la carga y tolerar fallos
- Los recursos críticos pueden replicarse.

Una cuestión fundamental en este tipo de computación es la coordinación de procesos para la realización de tareas (sincronización).

Existen diversas técnicas como por ejemplo [28]:

1. Con memoria compartida

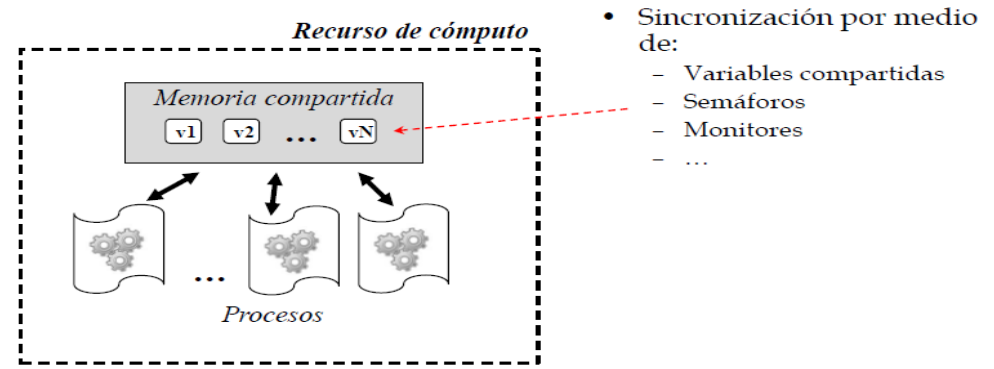


Figura 50. Sincronización con memoria compartida

2. Sin memoria compartida

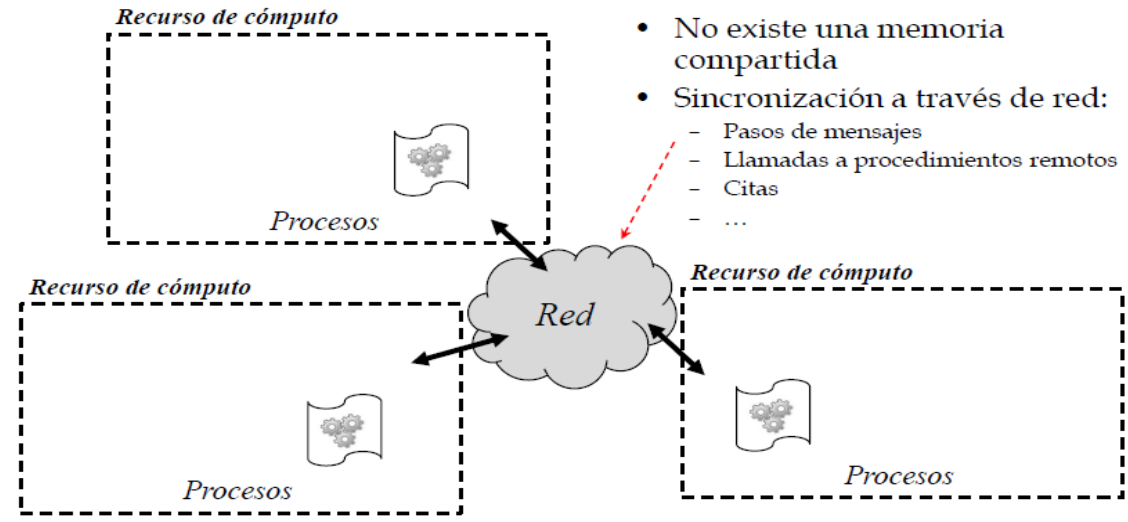


Figura 51. Sincronización sin memoria compartida

3. Combinación de las anteriores

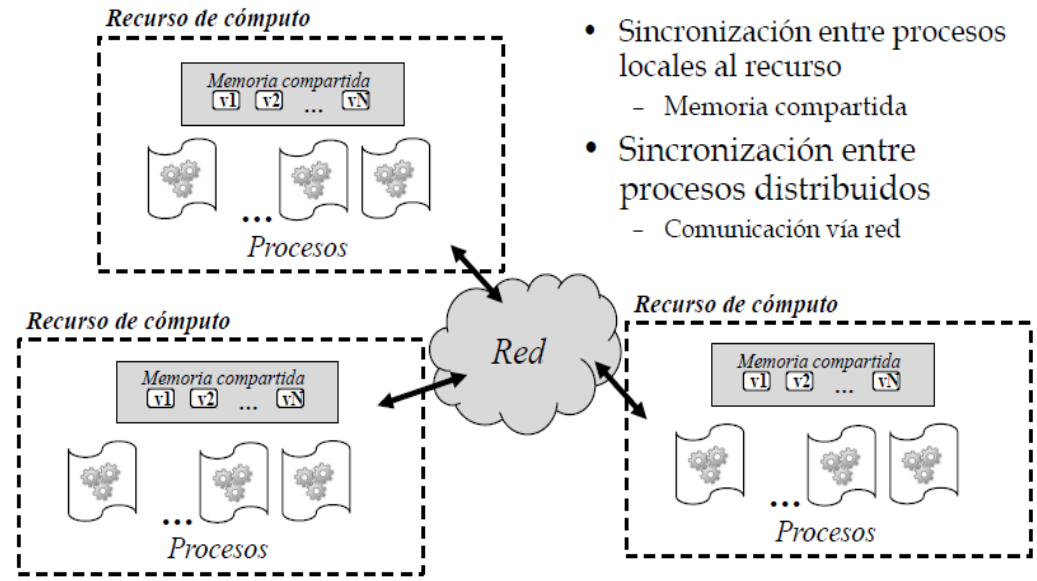


Figura 52. Combinación de sincronizaciones

Por otro lado la comunicación entre los procesos puede ser

1. Síncrona. Es dependiente temporalmente, es necesario que los participantes coincidan en el tiempo. Se necesita un reloj global

(metáfora: llamada telefónica).

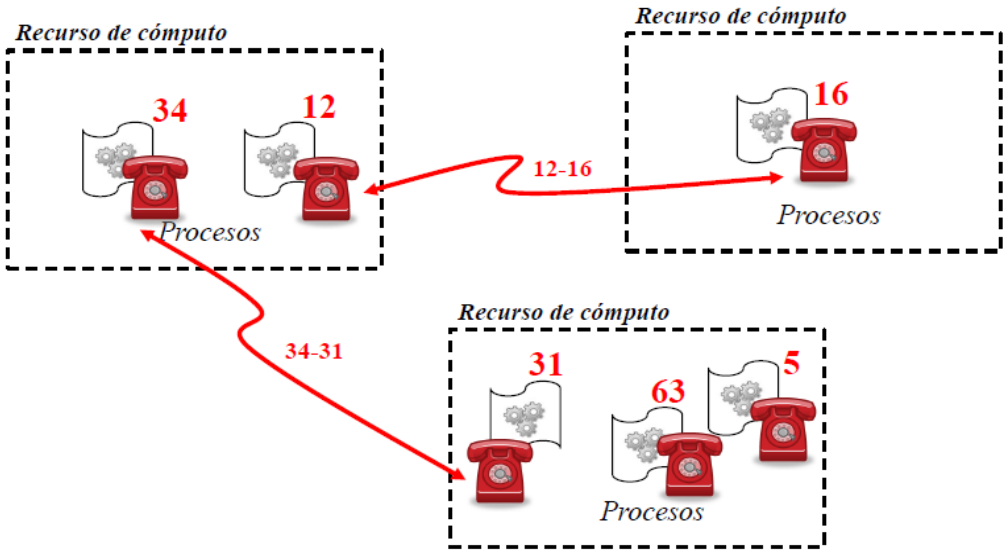


Figura 53. Comunicación síncrona

2. Asíncrona. Es independiente temporalmente, no es necesario que los participantes coincidan en el tiempo. Sin reloj global

(metáfora: servicio postal)

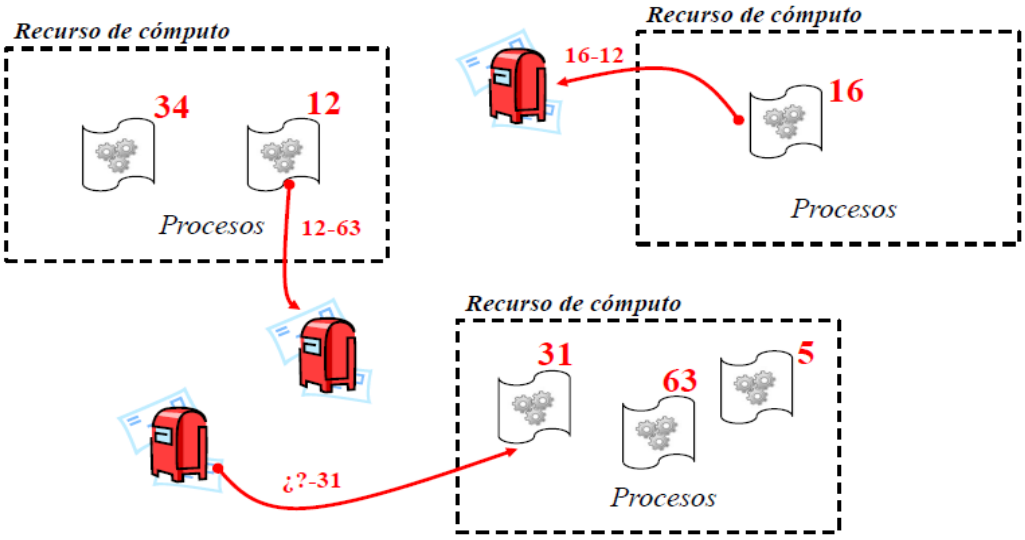


Figura 54. Comunicación asíncrona I

(metáfora: escribir en pizarra)

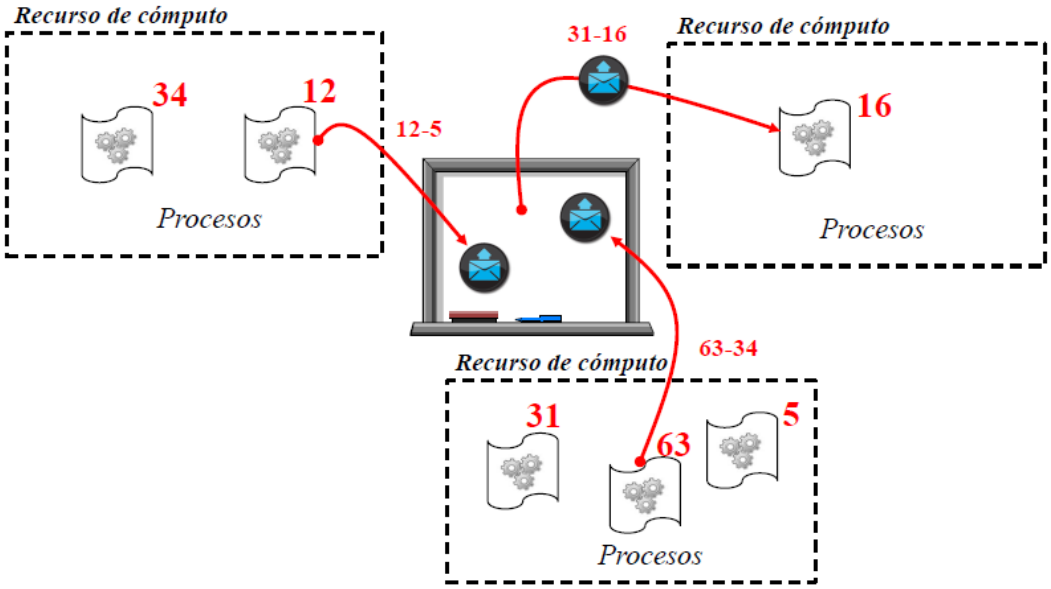


Figura 55. Comunicación asíncrona II

En la actualidad existen diversos paradigmas en la computación distribuida, démosle un pequeño repaso [29]

Cliente-Servidor: donde el proceso de servidor realiza las peticiones en nombre del cliente para compartir recursos en red

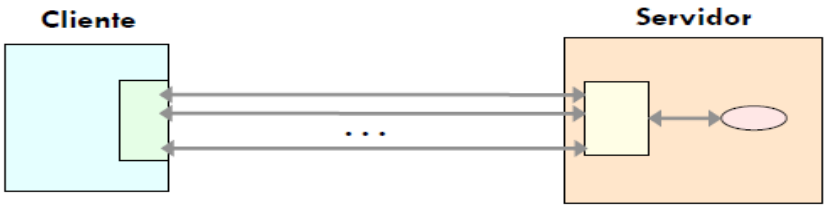


Figura 56. Cliente-Servidor

Sistema de mensajes o *middleware* orientado a mensajes (MOM): donde el sistema de mensajes actúa de intermediario entre los procesos que se comunican tanto punto a punto como con suscriptores.

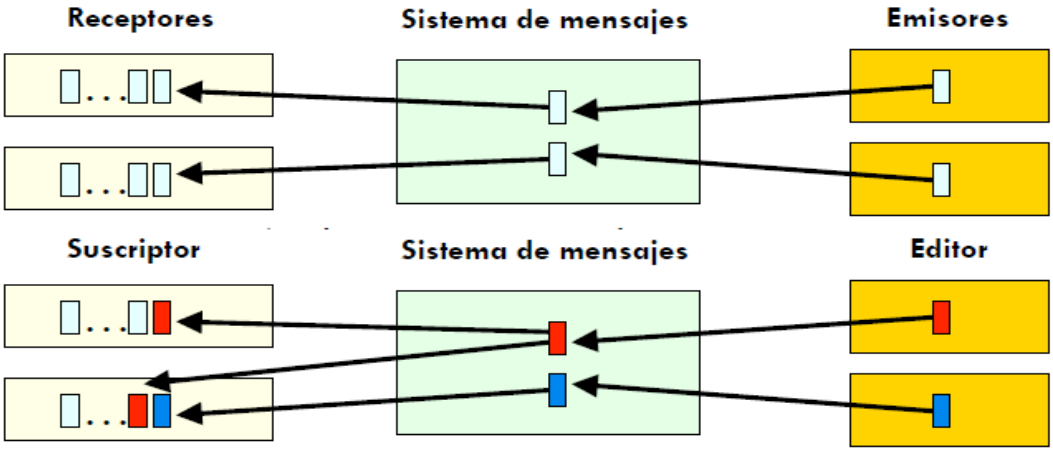


Figura 57. MOM

Agentes móviles: donde un proceso agente se lanza desde una máquina y viaja de forma autónoma por un itinerario accediendo a los recursos de las máquinas que visita

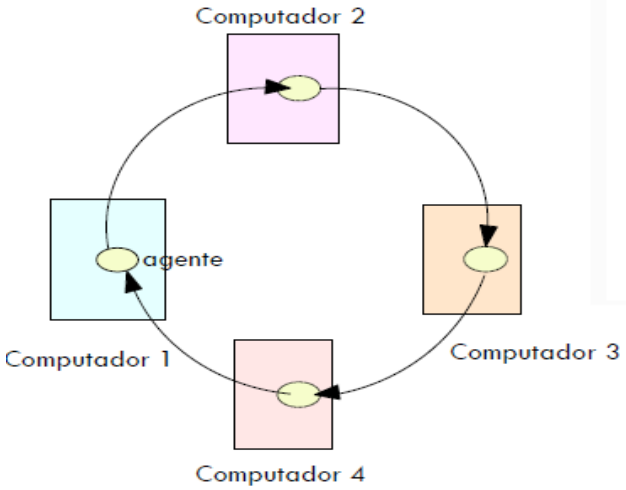


Figura 58. Agentes móviles

Espacio de objetos: donde los procesos colocan objetos que pueden ser accedidos por otros en un espacio común. Base de los sistemas basados en pizarra.

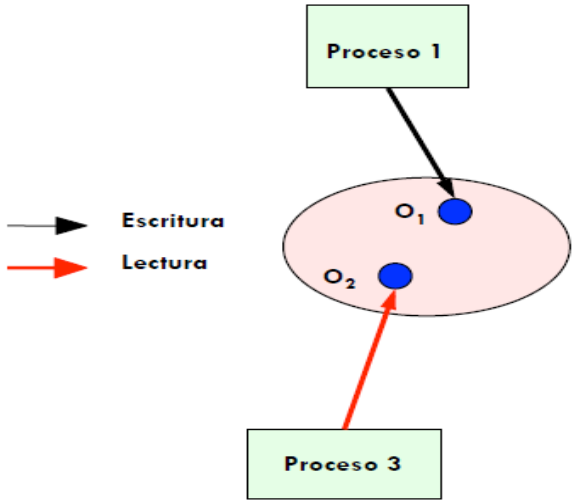


Figura 59. Espacio de objetos

SUPERCOMPUTACIÓN

Hay problemas que requieren de sistemas de grades prestaciones, como son los problemas que requieren mucha cantidad de cómputo y/o los que manejan grandes cantidades de datos [30].

En esta imagen podemos ver algunos ejemplos de este tipo de problemas

Figura 60. Ámbitos de uso de supercomputación

Para este tipo de problemas nació la supercomputación que en un primer lugar se desarrollaba con supercomputadores, máquinas con miles de procesadores en paralelo.

Era claro que el coste de los supercomputadores, muy elevado, debía llevar a buscar alternativas, y la computación distribuida fue la solución.

Destacando sin lugar a dudas los clusters, que son un conjunto de computadoras genéricas conectadas con una rede de alta velocidad y bajo retardo visto como un solo computador.

Los podemos clasificar en

- Cluster para balance de carga: para distribuir la carga de trabajo de manera eficiente
- Cluster de alta disponibilidad: para dar redundancia de datos y servicios de respaldo
- Cluster de alto rendimiento: para la ejecucion de aplicaciones en paralelo

Esto mejora el rendimiento, la escalabilidad, la disponibilidad y la reutlización.

CLOUD COMPUTING

A partir de lo anterior [29] encontramos empresas con exceso de capacidad de cómputo que pueden, de forma rentable, dejar usar sus sistemas a distintos clientes y así no desperdiciar estas capacidades. Este fenómeno se conoce como computación bajo demanda.

Podemos en tanto definir el Cloud Computing como el uso de recursos de computo escalables ofrecidos como un servicio desde fuera del entorno que los usa, a través de pago por uso. Sus principales ventajas son:

- Solo se usa lo que se necesita y se paga solo por lo usado
- Se puede acceder a cualquiera de los recursos que están en la nube en cualquier momento y desde cualquier sitio vía Internet
- No hay que pagar por construir grandes centros de datos, ni por la compleja administración de sistemas, ni por el elevado consumo eléctrico

Ejemplos de empresas que ofrecen estos servicios [30] los podemos ver en la figura 61

Figura 61. Empresas con servicios de Cloud Computing

Tal y como vemos en [31] se ofrecen tres modelos de servicio:

- Software como servicio (SaaS): las aplicaciones son almacenadas en la nube y ofrecidas como servicio a los usuarios. Un ejemplo es Google Docs
- Plataforma como servicio (PaaS): se ofrece una plataforma para construir aplicaciones y correr aplicaciones propias. Un ejemplo es Google App Engine
- Infraestructura como servicio (IaaS): se ofrece capacidad de cómputo y de almacenamiento bajo demanda. Un ejemplo es Amazon Web Service

La infraestructura del Cloud Computing puede visualizarse así

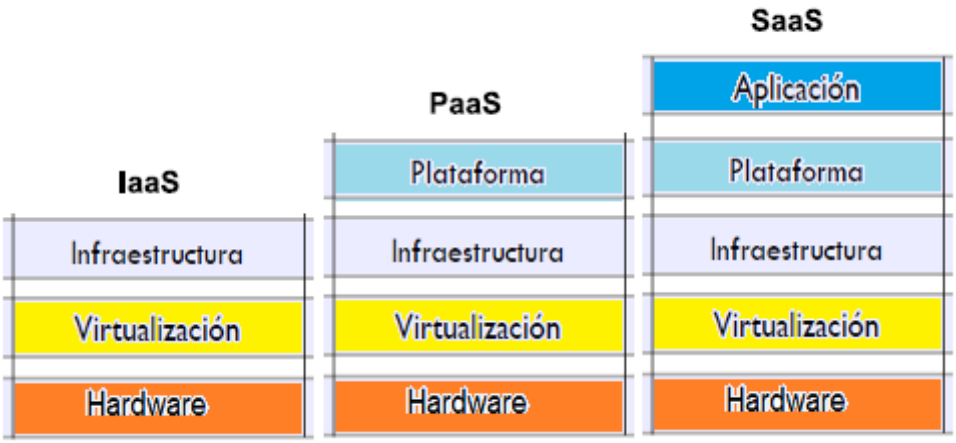


Figura 62. Elaboración propia a partir de [29]. Infraestructura del Cloud Computing

Existen varias formas de desplegar esta infraestructura [32]

1. Cloud privada, en la que los servicios no son ofrecidos al público en general. Las podemos diferenciar:
   a. Cloud propia. La infraestructura es íntegramente gestionada por una organización.
   b. Cloud compartida. La infraestructura es compartida por varias organizaciones.
2. Cloud pública. La infraestructura es operada por un proveedor que ofrece servicios al público en general.
3. Cloud híbrida. Combinación de Clouds individuales propias, compartidas o públicas que permite portar datos o aplicaciones entre ellas.

Podemos ver un esquema general [32]

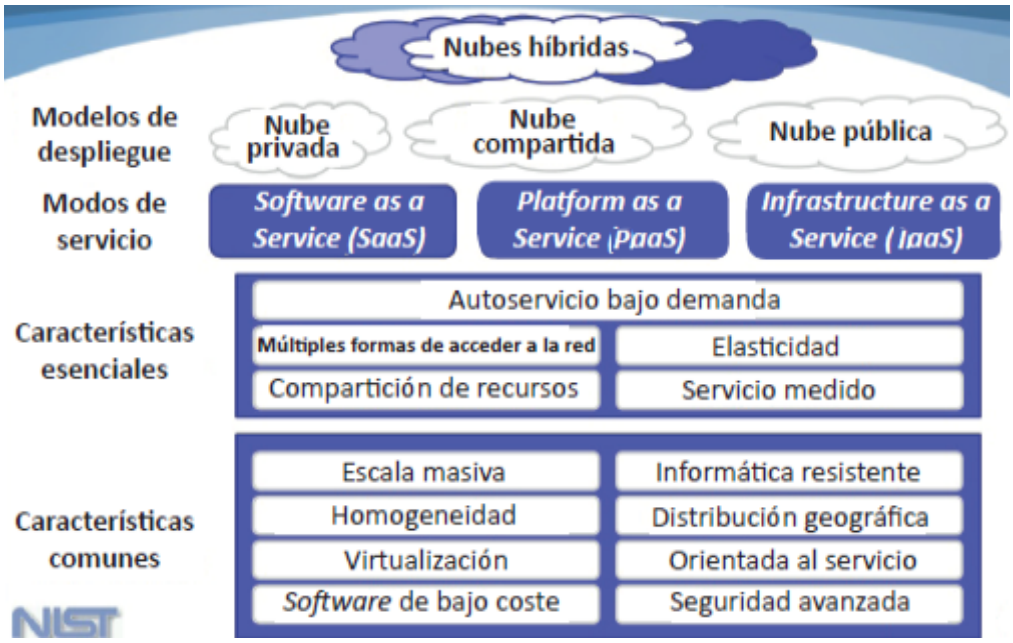


Figura 63. Despliegue de la infraestructura del Cloud Computing

Ya tenemos introducida la computación distribuida, la supercomputación y el Cloud Computing.

Y ahora hay que decir que es precisamente aquí, en una PaaS, donde ubicamos este trabajo ya que será usando una de estas plataformas como vamos a adaptar nuestro algoritmo.

Concretamente la plataforma que vamos a usar es Hadoop, un proyecto de Apache inspirado en el paradigma de programación MapReduce de Google.

Es una plataforma que permite el procesamiento distribuido de grandes cantidades de datos, del orden de petabytes, y que sin duda es lo que se necesita para llevar a cabo las labores propias del Big Data.

Así pues, toca acercarnos a MapReduce para continuar nuestro camino.

7. MAPREDUCE. El paradigma.

En 2004 Jeffrey Dean y Sanjay Ghemawat presenta el paradigma MapReduce tal y como se puede ver en [33]. En este documento se describe el modelo de computación paralela y distribuida que utiliza Google para tratar con grandes cantidades de datos.

Se dieron cuenta que muchas de las tareas que debían resolver tenían características comunes por lo que crean un paradigma y un FrameWork que les permita trabajar con estos problemas similares de una manera más rápida y sencilla.

Esta abstracción la basan en las primitivas *map* y *reduce* de *Lisp* y otros lenguajes funcionales.

En [34] Jeff Dean nos muestra con claridad cual es el problema típico a resolver con esta abstracción:

1. Leer un gran conjunto de datos y dividirlo en registros
2. **Map**: extraer de cada registro algo que te interesa
3. Mezclar y ordenar
4. **Reduce**: agregar, resumir, filtrar o transformar
5. Escribir los resultados

A este conjunto de 5 pasos se le va a denominar Job.

El programador para cada Job debe concentrarse solo en la función *map* y *reduce* ya que el FrameWork se encarga de todo lo demás.

Las funciones trabajan con pares de clave/valor

Función Map: genera claves/valores intermedios

map (K1, V1) → lista (K2, V2)

Función Reduce: combina los valores intermedios para cada clave

reduce (K2, lista (V2)) → (K3, lista (V3))

- Para cada clave de salida K3 se genera una lista de valores
- la lista(V3) suele tener un único valor
- K3 suele ser igual a K2

Visión gráfica.

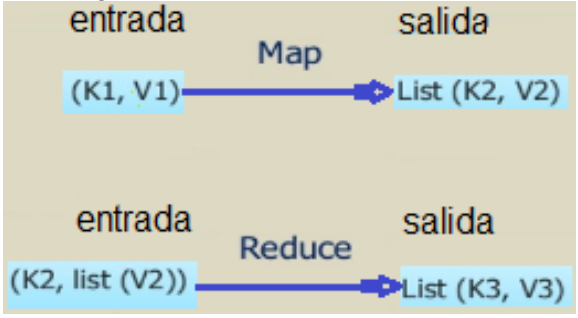


Figura 64. Elaboración propia. MapReduce

Para ilustrar todo lo anterior nada mejor que una vista general de un Job con el ejemplo clásico de contar frecuencias de palabras en una página Web.

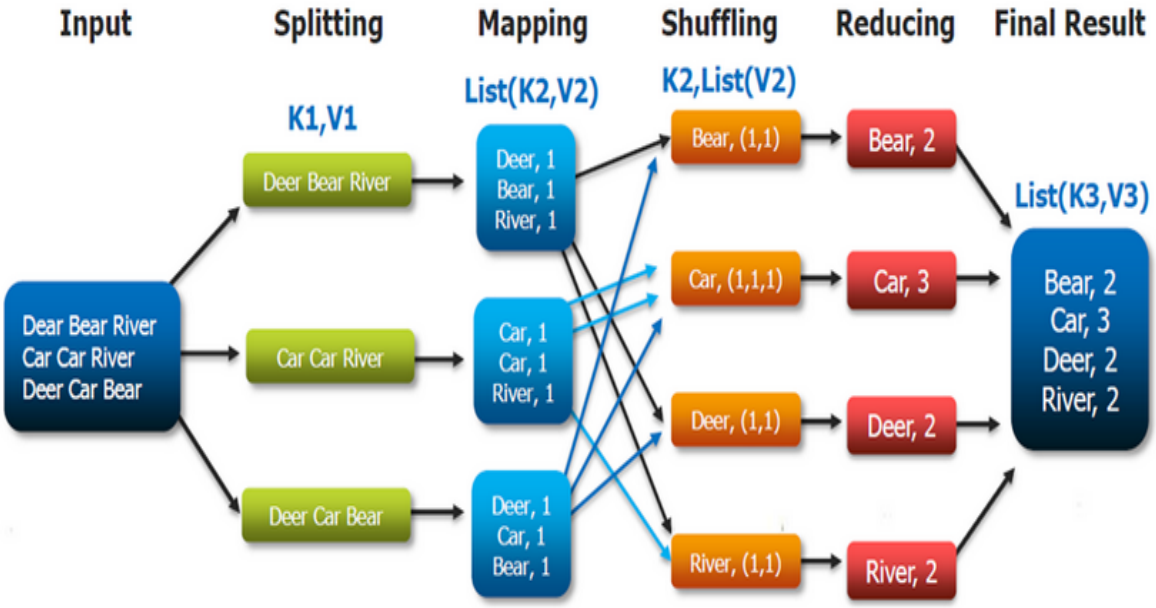


Figura 65 [35]. MapReduce. Contar palabras

Describámoslo:

1. Input: archivo con los datos
2. Spliting: Se divide el conjunto de datos en registros
3. Mapping: la función *map* divide cada registro en palabras. Cada palabra es un clave y el valor es el número 1.
4. Shuffling/sort: se agrupan los valores 1 por clave
5. Reducing: se suman los valores 1 de cada clave
6. Final Result: se presentan los resultados como una lista de palabras y sus valores (número de apariciones)

Ya que el programador tan solo se tiene que ocupar de *map* y *reduce* veamos cómo sería su implementación [34].

```
// input_key: nombre el documento
// input_value: una línea del documento
Map(String input_key, String input_value):
      for each word w in input_values:
            EmitIntermediate(w, "1");
End Map


// key: una palabra, la misma para entrada y salida
// intermediate_values: lista con las apariciones
Reduce(String key, Iterator intermediate_values):
      int result = 0;
      for each v in intermediate_values:
            result += ParseInt(v);
      Emit(AsString(result));
En Reduce
```

Destacar el paralelismo del proceso que evidentemente sólo se puede aplicar a problemas paralelizables, sin dependencia funcional entre partes, como el procesado de grandes cantidades de datos por separado.

Problemas típicos en los que Google usa este paradigma son, entre otros muchos, los siguientes [34]:

Búsqueda distribuida de patrones:

- map emite una línea si esta concuerda con un patrón dado.
- reduce copia dicha línea en la salida

Frecuencia de acceso a URL:

- map procesa los logs de las peticiones Web y emite el par <URL, 1>
- reduce suma todos los valores de una misa URL y emite el par <URL, cuentaTotal>

Grafo de enlace-Web inverso:

- map emite el par <objetivo, fuente> para cada objetivo encontrado en la página fuente
- reduce concatena la lista de todas las fuentes asociadas a cada objetivo y emite el par <objetivo,lista(fuente)>.

Vector de términos por host: un vector de términos resume los términos más importantes que aparecen en un documento como una lista de pares <palabra, frecuencia>

- map emite un par <hostname,vector> para cada entrada del documento, donde el hostname es extraído de la URL del documento.
- reduce junta todos los vectores de términos de un host y emite el par <hostname, vector>

Índice invertido:

- map escanea cada documento y emite una secuencia de pares <palabra, documento ID>
- reduce acepta los pares para una palabra dada, ordena los ID y emite el par <palabra, lista(documento ID)>

Y por fin estamos en condiciones de estudiar la adaptación de nuestro algoritmo a un entorno distribuido en la Nube usando precisamente este paradigma de programación distribuida y paralela desarrollado por Google.

8. FUZZY C-MEANS SOBRE MAPREDUCE

La adaptación de Fuzzy C-Means al paradigma MapReduce tiene distintas variantes, pero antes de presentar alguna de ellas vamos a recordar el algoritmo al que le hemos dado un ligero matiz.

En lugar de comenzar generando aleatoriamente la matriz de pertenencias se hace generando los centros iniciales y se terminan las iteraciones cuando estos centros han convergido.

INICIO

1. Elegir el número de clusters 1 < ***C*** < N.

   //obviamente ha de ser mayor que 1 y menor que el número de objetos a agrupar

2. Elegir el exponente ***m*** > *1*

   //como ya hemos mencionado ha de ser mayor que 1 estando su valor más practico entre [2, 2.5]

3. Elegir la norma (como medir la distancia)

   //en nuestro caso para este trabajo la distancia euclidiana

4. Elegir el umbral ***ε*** para terminar las iteraciones

   //normalmente entre [0.001, 0.01]

5. Generar el vector de centros ***V*** aleatoriamente.

   //vector donde aparecen los centros de cada cluster

6. Hacer

   1. Calcular las distancias ***d*** de los objetos a los centros

      //distancia euclidiana de cada objeto a cada centro

   2. Actualizar la matriz ***U*** de pertenencias

      //con la fórmula 1

   3. Calcular los nuevos centros ***V*** de los grupos

      //con la fórmula 2

   Mientras no haya convergencia de centros

FIN

Habrá convergencia cuando se cumpla

$$||V^{(b)} - V^{(b+1)}|| < \epsilon$$

Es decir, la distancia entre los nuevos centros y los anteriores será menor que el umbral determinado en el paso 4.

Recordamos las fórmulas:

$$u_{ik} = \frac{1}{\sum_{j=1}^{c} \left(\frac{d_{ik}}{d_{jk}} \right)^{2/(m-1)}}$$

Fórmula 1

$$v_i = \frac{\sum_{k=1}^{n} (u_{ik})^m \, x_k}{\sum_{k=1}^{n} (u_{ik})^m}$$

Fórmula 2

A partir de aquí y teniendo en cuenta el funcionamiento del paradigma MapReduce podemos comenzar a pensar en la adaptación.

Existen diversas formas de hacerlo, unas basadas en un único Job y otras usando varios.

En este trabajo optamos por basarnos en un único Job y vamos a presentar dos versiones.

En primer lugar comencemos estudiando las entradas del algoritmo

ENTRADAS

- archivo con los objetos ***O*** a agrupar
- número de clusters ***C***
- archivo con los centros ***V*** aleatorios
- valor del umbral ε de parada por convergencia
- valor del exponente ***m***
- norma para la distancia (en este caso euclidiana)

Vayamos ilustrándolo, Imaginemos que tenemos un archivo con una serie de ***n*** objetos a agrupar en ***c*** clusters.

Objetos: $\mathbf{O_1}$, ... , $\mathbf{O_n}$

Centros: $\mathbf{V_1}$, ... , $\mathbf{V_C}$

Clusters: $\mathbf{C_1}$, ... , $\mathbf{C_C}$

Toca ahora estudiar las funciones *map* y *reduce* de cada versión .

VERSIÓN UNO

El Job irá cogiendo cada uno de los objetos *O* y dándoselos a una función *map* como valor y como clave el desplazamiento del objeto dentro del archivo.

MAP

Recibe: < Desplazamiento **clave**, Objeto $\mathbf{O_k}$ >

1. Calcular la distancia euclidianas $d_{K1}, \ldots, d_{KC}$ del objeto O_k a cada uno de los centros $V_1, \ldots, V_C$
2. Calcular la pertenencia $U_{k1}, \ldots, U_{KC}$ del objeto O_k a cada uno de los clusters $\mathbf{C_1}, \ldots, \mathbf{C_C}$ usando la fórmula 1
3. Calcular la componente $\mathbf{S_{k1} = O_k} * U_{K1}^{m}, \ldots, S_{kC} = O_k * U_{KC}^{m}$ de la sumatoria del numerador de la fórmula 2
4. Emitimos como valor intermedio
   < **k**, $\{U_{k1}, \ldots, U_{kC}, \mathbf{S_{k1}}, \ldots, \mathbf{S_{kC}}\}$ >

FIN MAP

REDUCE

Recibe: < **k**, $\{U_{k1}, \ldots, U_{kC}, \mathbf{S_{k1}}, \ldots, \mathbf{S_{kC}}\}$ >

1. Guardar las pertenencias $U_{k1}, \ldots, U_{kC}$ del objeto **k**
2. Guardar las multiplicaciones $\mathbf{S_{k1}}, \ldots, \mathbf{S_{kC}}$ del objeto **k**

FIN REDUCE

CLEANUP // fin del paralelismo

Una vez guardadas todas las pertenencias y multiplicaciones necesarias para la fórmula 2 calcularemos los nuevos centros de la siguiente manera:

Para Cada Centro **C**

1. Calcular numerador = $\mathbf{S_{1C} + \ldots + S_{nC}}$
2. Calcular denominador = $U_{1C}^{m} + \ldots + U_{nC}^{m}$
3. Realizar la división y obtener el nuevo centro **C** según la formula 2

Fin para

Reescribir el archivo con los nuevos centros.

Si $(C_1^{b+1} - C_1^{b} < \varepsilon \text{ and } \ldots \text{ and } C_C^{b+1} - C_C^{b} < \varepsilon)$

escribir el archivo de pertenencias y finalizar

Si no

volver a la función *map*

FIN CLEANUP

Gráficamente lo podríamos ver como sigue para 4 objetos con 2 centros

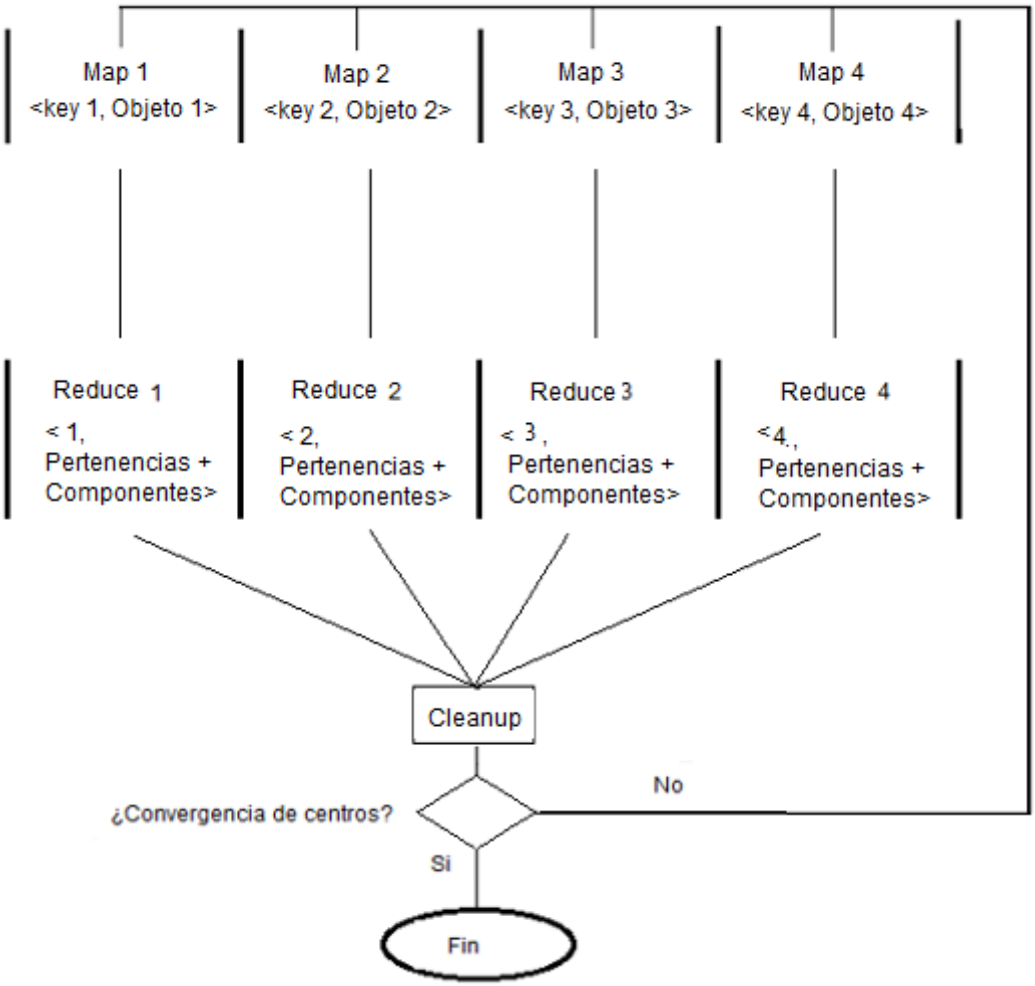


Figura 66. Elaboración propia. Adaptación versión I

VERSIÓN DOS

El Job irá cogiendo cada uno de los objetos *O* y dándoselos a una función *map* como valor y como clave el desplazamiento del objeto dentro del archivo.

MAP

Recibe: < Desplazamiento **clave**, Objeto $\mathbf{O_k}$ >

1. Calcular la distancia euclidianas $d_{K1}, \ldots, d_{KC}$ del objeto O_k a cada uno de los centros $V_1, \ldots, V_C$
2. Calcular la pertenencia $U_{k1}, \ldots, U_{KC}$ del objeto O_k a cada uno de los clusters $\mathbf{C_1}, \ldots, \mathbf{C_C}$ usando la fórmula 1
3. Calcular la componente $\mathbf{S_{k1} = O_k} * U_{K1}^{m}, \ldots, S_{kC} = O_k * U_{KC}^{m}$ de la sumatoria del numerador de la fórmula 2
4. Emitimos valores intermedios

   Para cada centro **C**

   emite < **C**, $\{U_{kC}, \mathbf{S_{kC}}\}$ >

   Fin para

FIN MAP

REDUCE

Recibe: < **C**, $\{ \{U_{1C}, \mathbf{S_{1C}}\}, \ldots, \{U_{nC}, \mathbf{S_{nC}}\} \}$ >

1. Calcular numerador del centro = $\mathbf{S_{1c} + \ldots + S_{nC}}$
2. Calcular denominador del centro = $U_{1c}^{m} + \ldots + U_{nC}^{m}$
3. Realizar la división y obtener el nuevo centro **C** según la formula 2
4. Guardar el nuevo centro C

FIN REDUCE

CLEANUP // fin del paralelismo

Una vez guardados todos los nuevos centros procedemos a:

Rescribir el archivo con los nuevos centros.

Si $(C_1^{b+1} - C_1^{b} < \varepsilon \text{ and } \ldots \text{ and } C_C^{b+1} - C_C^{b} < \varepsilon)$

escribir el archivo de pertenencias y finalizar

Si no

volver a la función *map*

FIN CLEANUP

Gráficamente lo podríamos ver como sigue para 4 objetos con 2 centros

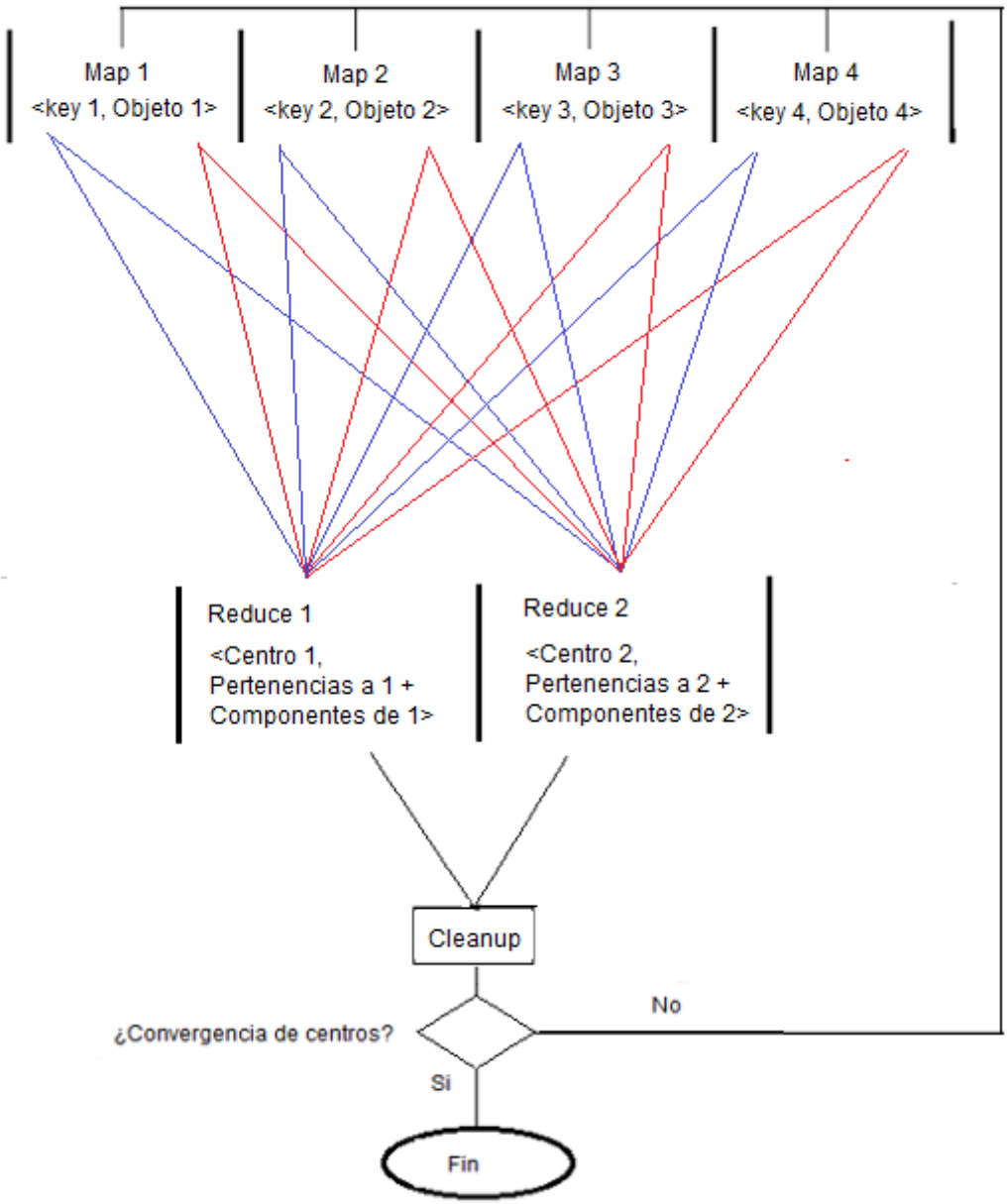


Figura 67. Elaboración propia. Adaptación versión II

ESTUDIEMOS AMBAS VERSIONES

CUALITATIVAMENTE

En las dos versiones se realizan los mismos cálculos en las tareas *map,* se calculan:

1. U_{kC} las distintas pertenencias del objeto k a los distintos clusters
2. $S_{kC} = O_k * U_{KC}^m$ las distintas componentes del denominador necesarias para obtener el centro

Ahora, la diferencia estriba en como se pasa esta misma información a las tareas *reduce*.

En la versión uno se realiza una sola emisión con clave el número de objeto y con valor toda la información obtenida

$< k, \{U_{k1}, \dots, U_{kC}, S_{k1}, \dots, S_{kC}\} >$

Y en la versión dos se pasa esta misma información pero de manera distinta. Usaremos varias emisiones, tantas como centros tengamos, con el número de centro como clave y su correspondiente pertenencia y componente como valor:

Para cada centro **C**

emite $< C, \{U_{kC}, S_{kC}\} >$

Fin para

Sabiendo que habrá tantas tareas *reduce* como claves distintas emitan las tareas *map,* tendremos dos claras diferencias entre ambas versiones:

1. el número de tareas *reduce:* para la versión uno tendremos tantas tareas reduce como objetos y para la versión dos tantas tareas reduce como centros
2. el trabajo que puede realizar cada tarea *reduce* será distinto en las dos versiones ya que le llega distinta cantidad de información.

En la versión uno una tarea *reduce* recibe

$< k, \{U_{k1}, \dots, U_{kC}, S_{k1}, \dots, S_{kC}\} >$

información que hay que guardar de manera adecuada. Una vez finalizado el paralelismo se calculan los nuevos centros con esta información.

En la versión dos una tarea *reduce* recibe

$< C, \{ \{U_{1C}, S_{1C}\}, \dots, \{U_{nC}, S_{nC}\} \} >$

y con esta información se calcula el nuevo centro.

CUANTITATIVAMENTE

Recordemos que ***n*** es el número de objetos a clasificar y ***c*** el número de centros y normalmente ***n >> c***

Para la comparación cuantitativa de las dos adaptaciones tenemos que tener en cuenta dos factores:

1. Nivel de paralelismo: a mayor porcentaje de paralelismo mayor aceleración en el procesamiento de los datos
2. Tráfico de red: a mayor tráfico de red menor aceleración en el procesamiento de los datos

Lo que buscamos es incrementar todo lo posible el paralelismo en una y disminuir todo lo posible el tráfico en la otra.

Detengámonos en el paralelismo: consideramos el paralelismo como función del porcentaje de código paralelo a ejecutar.

- el número de tareas *map es* ***n*** para las dos versiones y con los mismos cálculos en ambas.
- el número de tareas *reduce* es bastante mayor en la versión uno que en la dos, en la versión uno ese número es ***n*** mientras que en la versión dos es ***c***. Sin embargo la cantidad de código a ejecutar en las tareas *reduce* de la versión uno es sensiblemente inferior al que se ejecuta en la versión dos.

Esto nos lleva a colegir que hay más paralelismo en la versión dos que en la uno ya que hay más porcentaje de código en paralelo.

Veamos ahora el tráfico de red: consideramos el tráfico de red como función de la cantidad de emisiones intermedias que se realicen.

- la cantidad de emisiones intermedias que hacemos en la versión uno es ***n*** mientras que en la versión dos en ***cn***, luego el tráfico de red es mayor con la versión dos. Cuantos más centros más tráfico.

  En consecuencia la versión uno implica menos tráfico de red.

Como conclusiones podríamos decir que la versión uno disminuye el tráfico de red y la versión dos aumenta el paralelismo ¿Cuál de las dos es más apropiada? esto va a depender del problema concreto a resolver.

Asumiendo que trabajamos con cantidades masivas de datos la clave diferenciadora estará en el número de grupos que queramos obtener, conforme aumente el número de centros mejorará el paralelismo en la versión dos y se incrementará el rendimiento, aunque esta aceleración siempre irá frenada por el tráfico de red que también se multiplica.

Por lo tanto presuponemos que con una cantidad significativa de centros la balanza se acabará inclinando por la versión dos, pero serán las pruebas de rendimiento las que sin duda nos dirán si esta idea se acerca a la realidad.

9. IMPLEMENTACIÓN Y PRUEBAS EN HADOOP

Una vez adaptado nuestro algoritmo a un entorno MapReduce toca implementarlo y probarlo con un conjunto de datos.

Para dicha implementación vamos a utilizar Hadoop, que tal y como aparece en [36], es un proyecto de Apache™ pensado para computación distribuida confiable y escalable de grandes cantidades de datos.

Se trata de un Framework que permite escalar desde un único servidor hasta miles de máquinas cada una ofreciendo almacenamiento y procesamiento local.

Está pensado asumiendo que los fallos hardware en los sistemas distribuidos son algo común y en lugar de confiar en el hardware para lograr la alta disponibilidad lo que hace es detectar y manejar los posibles fallos desde la capa de aplicación.

Una de las principales ideas que subyacen en el diseño de Hadoop es la movilidad del código.

Se asume que a menudo es mejor mover el código cerca de donde están los datos que mover los datos a donde está el código ejecutándose. Esto es especialmente cierto cuando se trabaja con grandes cantidades de datos como es el caso del Big Data.

Esta idea minimiza la congestión de la red y mejora el rendimiento general del sistema.

La arquitectura Hadoop consta de los siguientes módulos:

1. Hadoop Common: utilidades comunes de apoyo a las demás utilidades.

2. Hadoop Distributed File System (HDFS™): un sistema de ficheros distribuido de alto rendimiento.

3. Hadoop YARN: un framework para la planificación de los Jobs y la gestión de los recursos del cluster

4. Hadoop MapReduce: un sistema YARN para el procesamiento paralelo de grandes cantidades de datos.

Como vemos Hadoop es básicamente HDFS + MapReduce, este último ya lo conocemos, así que acerquémonos ahora a HDFS.

Se trata de la capa de almacenamiento de Hadoop y se encarga del almacenamiento de los datos en cantidades ingentes, del orden de petabytes. Como ya hemos mencionado es un sistema de ficheros distribuido corriendo en un cluster de máquinas.

Los datos están repartidos por todos los nodos y para manejar los posibles fallos existe la redundancia de datos.

Como cada nodo es parte del sistema de almacenamiento se dispone de un demonio en cada máquina. Estos demonios interaccionan para el intercambio de datos.

Para coordinar los nodos existe una arquitectura maestro/esclavo con:

- Un nodo maestro llamado Name Node
- Múltiples esclavos llamados Data Nodes

Destacar que los archivos en HDFS son de una escritura y múltiples lecturas, no se pueden actualizar en consecuencia.

Las distribuciones de Hadoop suelen aportar mucho más, en este trabajo vamos a utilizar la que nos proporciona Cloudera, que consta entre otros de los siguientes componentes:

- Hadoop
- HBase: una base de datos no relacional de tipo clave-valor que permite tener millones de filas y millones de columnas
- Hive: que permite hacer consultas con sintaxis SQL sobre HDFS
- Pig: que permite escribir procedimientos en Pig Latin para analizar y procesar grandes cantidades de datos. Internamente funciona creando secuencias de Jobs. Esto facilita la resolución de problemas al ser un framework en el que se utiliza un lenguaje de alto nivel.
- ZooKeeper: coordinador distribuido de alta confiabilidad usado con HBase, Pig, Hive, etc.

Para este trabajo seguimos los pasos descritos en [37] y creamos un cluster de tres nodos. Y es en este cluster donde vamos a probar el código creado y disponible en:
https://drive.google.com/open?id=0B7LZjmf-ZQ65SDA1OF9WOWdiVkk

PRUEBAS DE VALIDACIÓN

Comprobemos que los productos obtenidos cumplen con lo que se espera de ellos, llevar a cabo perfectamente las adaptaciones de nuestro algoritmo a un entorno distribuido.

Todos los recursos necesarios para estas pruebas se encuentran en: https://drive.google.com/open?id=0B7LZjmf-ZQ65SDA1OF9WOWdiVkk

DATOS PARA LAS PRUEBAS

Elegimos el conjunto de datos "*perfume*" obtenido en el repositorio de Machine Learning de la UCI [38]. Contiene los datos de 20 perfumes, para cada uno de ellos tenemos 28 valores reales que se corresponde con las medidas tomadas en 28 instantes por un medidor de olor OMX-GR. Veamos un extracto.

nombre del perfume | medida 1 | medida 2 | .....medida 28

A	B	C
ajayeb	64.558	64.556
carolina_herrera	63.076	63.072
oudh_ma'alattar	67.197	67.197
constrected	65.151	66.151
asgar_ali	68.209	68.209
bukhoor	71.046	71.046
burberrry	61.096	61.096

Figura 68. Extracto de Perfume

Vamos a utilizar las 28 medidas de cada uno de los 20 perfumes, se dispone en consecuencia de 560 instancias de valores reales.

PREPARACIÓN DE LOS DATOS

Cambiamos el nombre de cada perfume por un valor identificativo del registro, estos valores son los números enteros del 1 al 20 creando así el fichero ***"perfumes"*** de texto plano separado por comas.

CENTROS INICIALES

Debido a la importancia que tiene la elección de los centros iniciales para este algoritmo se decide que sea el usuario el que los elija en lugar de hacerlo aleatoriamente.

Por ello creamos un archivo llamado ***"centros"*** con los centros iniciales. En este caso 3 centros. Aquí vemos un extracto del mismo.

1	65.406	65.571	65.570
2	64.077	64.245	64.078
3	74.722	74.553	74.720

Figura 69. Extracto del archivo centros

El primer centro se corresponde con la media de los 6 primeros objetos, el segundo centro con la media de los 6 siguientes objetos y el tercero con la media de los 6 siguientes.

PARÁMETROS DE ENTRADA

Recordamos que este algoritmo necesita una serie de parámetros de entrada, a saber:

- los centros iniciales
- el parámetro ***m*** (fuzziness parameter).
- El parámetro ε (criterio de parada)

PRUEBAS REALIZADAS

Realizamos el siguiente conjunto de pruebas

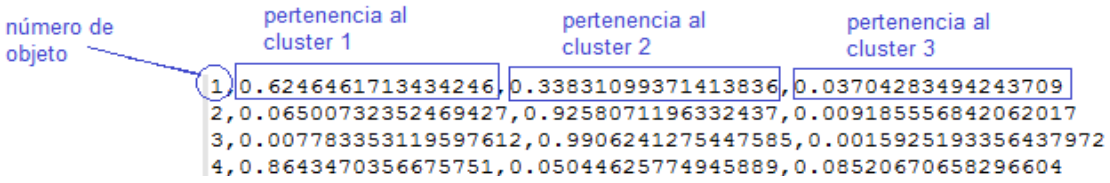
m	ε	PRUEBA
2.05	0.001	1
2.01	0.005	2
2.0	0.01	3

Figura 70. Planificación de pruebas de validación

RESULTADOS

Ambas versiones obtienen los mismos resultados en todas las pruebas.

Vamos a presentar el conjunto de pertenencias obtenido en cada prueba. El conjunto tiene un aspecto tal que así.



La asignación de un objeto a un cluster será como sigue: asignaremos cada objeto al cluster para el que tenga mayor grado de pertenencia. Es decir, en el cluster *C* estarán aquellos objetos cuya pertenencia al cluster *C* sea mayor que su pertenencia a cualquier otro cluster.

Además veremos gráficamente los clusters. Aparecerá cada objeto y su grado de pertenencia al cluster y podremos observar como varía el color en función de la pertenencia.

PRUEBA 1

Conjunto de pertenencias obtenido

1,0.6237158376061157,0.3392356219131263,0.03704854048075797
2,0.06455621747793461,0.926314333198379,0.009129449323686391
3,0.007882546483089201,0.9905036813640656,0.0016137721528452548
4,0.8645969699083415,0.05039501616503039,0.0850080139266282
5,0.9925437633979811,0.005085222072348761,0.0023710145296700087
6,0.9588263058078691,0.02060243759056314,0.020571256601567837
7,0.044653315999699156,0.9487003381290997,0.006646345871201269
8,0.9968471639867157,0.001972782200460858,0.0011800538128234467
9,0.8317986804823,0.057166934436131626,0.11103438508156836
10,0.6377788036391958,0.07679420910088401,0.2854269872599202
11,0.3234488736477509,0.6458726393149874,0.03067848703726163
12,0.2179665061967999,0.6947983105549543,0.08723518324824575
13,0.0017332424782228817,5.810085280662691E-4,0.9976857489937109
14,0.04974663483291963,0.013370422532016549,0.9368829426350638
15,0.03598464162923664,0.013876650752366361,0.950138707618397
16,0.8795896433227006,0.04647205497115233,0.07393830170614694
17,0.910136673964284,0.07109505986808351,0.018768266167632462
18,0.5642401386091336,0.39717276694097514,0.038587094449891335
19,0.9333423970920333,0.05154634387505821,0.015111259032908551
20,0.8098351917001071,0.1607649758267753,0.029399832473117705

Figura 71. Prueba de validación uno

PRUEBA 2

Conjunto de pertenencias obtenido

1,0.624101530082046,0.3388522759166804,0.037046194001273514
2,0.06474292413219898,0.9261043969312148,0.009152678936586263
3,0.007841371644969826,0.9905536806287062,0.0016049477263240409
4,0.8644933849302027,0.05041627512858708,0.08509033994121029
5,0.9925764256227434,0.005061830507498421,0.002361743869758101
6,0.958765382975435,0.020626837163072172,0.020607779861492807
7,0.04480180011510631,0.948531962794105,0.006666237090788791
8,0.9968426978808967,0.0019750932006054717,0.0011822089184978738
9,0.8316859334473551,0.057184939331371615,0.11112912722127334
10,0.637636054539909,0.07679346848792008,0.2855704769721709
11,0.3238580877235717,0.6454394267141013,0.03070248556232701
12,0.21792044928762008,0.6948723687389835,0.08720718197339647
13,0.0017358847794123964,5.818853772148422E-4,0.9976822298433727
14,0.04971344499586799,0.01336166973900799,0.936924885265124
15,0.03598930801331861,0.013878065750925738,0.9501326262357558
16,0.8794899185409576,0.04649480620517634,0.07401527525386607
17,0.9102973206691313,0.07095604018846662,0.0187466391424022

18,0.5646364642534993,0.3967729436229188,0.03859059212358174
19,0.9334745230735958,0.051435063064095746,0.015090413862308546
20,0.8100978403561582,0.1605206307112859,0.029381528932556064

Figura 72. Prueba de validación dos

PRUEBA 3

Conjunto de pertenencias obtenido

1,0.6246461713434246,0.33831099371413836,0.03704283494243709
2,0.06500732352469427,0.9258071196332437,0.009185556842062017
3,0.007783353119597612,0.9906241275447585,0.0015925193356437972
4,0.8643470356675751,0.05044625774945889,0.08520670658296604
5,0.9926223836277368,0.005028926226565056,0.002348690145698261
6,0.9586792694927848,0.020661295215841494,0.02065943529137366
7,0.04501216014268976,0.9482934370494615,0.006694402807848744
8,0.9968362631042437,0.001978433666456458,0.0011853032292998187
9,0.8315266430716828,0.05721032126986327,0.11126303565845401
10,0.6374343920144335,0.07679236577735173,0.28577324220821476
11,0.32443655831584733,0.6448270702164123,0.030736371467740366
12,0.21785529797510472,0.6949770971825416,0.0871676048423537
13,0.0017396265097424255,5.831269019288822E-4,0.9976772465883287
14,0.049666553378731944,0.013349298820310529,0.9369841478009575
15,0.03599592321113414,0.013880070218106848,0.9501240065707589
16,0.8793490205661125,0.04652689937739237,0.07412408005649504
17,0.9105239250058336,0.07075997503566558,0.018716099958500997
18,0.5651962345787529,0.3962082764772784,0.038595488943968685
19,0.9336608631846942,0.05127815037095774 5,0.015060986444348062
20,0.8104685000256332,0.1601758430611683,0.029355656913198486

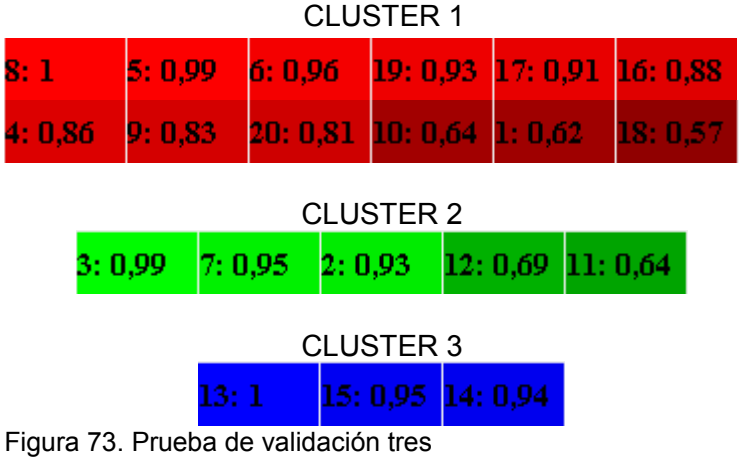


Figura 73. Prueba de validación tres

CONCLUSIONES

Ambas adaptaciones se han comportado exactamente igual, produciendo los mismos resultados. La variación de parámetros no ha provocado más que un ligero cambio en los decimales de los grados de pertenencia. Esto nos indica que los centros, en este caso la media de 6 elementos para cada uno , estaban bien elegidos.

PRUEBAS DE RENDIMIENTO

Comparemos el rendimiento de las dos adaptaciones obtenidas. Vamos a medir el tiempo que emplea cada una de ellas sobre un conjunto de pruebas.

Estas pruebas se han diseñado para ir aumentando paulatinamente la cantidad de computo y así ver como se comportan nuestras adaptaciones. En el banco uno se incrementará la cantidad de instancias a tratar y en el banco dos el número de grupos a formar.

El conjunto de pruebas diseñado es el siguiente

BANCO DE PRUEBAS UNO

VERSIÓN	CENTROS	INSTANCIAS DE TIPO REAL
fuzzy c-means V1	20	15000
fuzzy c-means V2	20	15000
fuzzy c-means V1	20	30000
fuzzy c-means V2	20	30000
fuzzy c-means V1	20	60000
fuzzy c-means V2	20	60000

Figura 74. Banco de pruebas uno

BANCO DE PRUEBAS DOS

VERSIÓN	CENTROS	INSTANCIAS DE TIPO REAL
fuzzy c-means V1	2	60000
fuzzy c-means V2	2	60000
fuzzy c-means V1	10	60000
fuzzy c-means V2	10	60000
fuzzy c-means V1	20	60000
fuzzy c-means V2	20	60000
fuzzy c-means V1	50	60000
fuzzy c-means V2	50	60000

Figura 75. Banco de pruebas dos

Todos los recursos necesarios para estas pruebas se encuentran en: https://drive.google.com/open?id=0B7LZjmf-ZQ65SDA1OF9WOWdiVkk

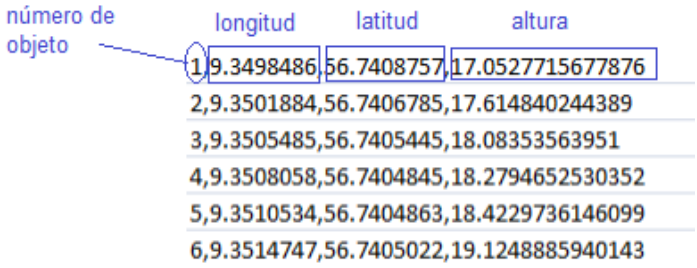
PLATAFORMA PARA LAS PRUEBAS

Disponemos de un cluster de máquinas virtuales con tres nodos sobre VirtualBox con Windows 7 en un procesador Intel I3 con 16 Gb de Ram.

DATOS

Se han construido los ficheros de datos necesarios para los dos bancos de pruebas. Para esta tarea se han utilizado los 5000, 10000 y 20000 primeros registros del conjunto "*3D Road Network (North Jutland, Denmark) Data Set* " disponible en [39].

La interpretación que tiene cada registro de los ficheros construidos es la siguiente



Donde longitud y latitud están expresadas en grados decimales y la altura en metros.

CENTROS INICIALES

Como centros iniciales preparamos cuatro archivos con 2, 10, 20 y 50 centros. Cada centro ha sido obtenido tomando la media de 10 elementos elegidos al azar.

PARÁMETROS DE ENTRADA

Establecemos como parámetros de entrada los siguientes:

- los centros iniciales
- fuzziness parameter $\boldsymbol{m}$ = 2.0
- umbral de parada $\boldsymbol{\varepsilon}$ = 0.01

RESULTADOS DEL BANCO DE PRUEBAS UNO

VERSIÓN	CENTROS	INSTANCIAS	TIEMPO MEDIO POR JOB
fuzzy c-means V1	20	15000	16.1 s
fuzzy c-means V2	20	15000	15.8 s
fuzzy c-means V1	20	30000	16.5 s
fuzzy c-means V2	20	30000	15.6 s
fuzzy c-means V1	20	60000	16.9 s
fuzzy c-means V2	20	60000	14.8 s

Figura 76. Resultados del banco de pruebas uno

RESULTADOS DEL BANCO DE PRUEBAS DOS

VERSIÓN	CENTROS	INSTANCIAS	TIEMPO MEDIO POR JOB
fuzzy c-means V1	2	60000	16.2 s
fuzzy c-means V2	2	60000	16.8 s
fuzzy c-means V1	10	60000	16.6 s
fuzzy c-means V2	10	60000	16.3 s
fuzzy c-means V1	20	60000	17.1 s
fuzzy c-means V2	20	60000	14.7 s
fuzzy c-means V1	50	60000	17.8 s
fuzzy c-means V2	50	60000	14.9 s

Figura 77. Resultados del banco de pruebas dos

Observemos en el banco uno como ambas versiones tienen un comportamiento similar con pocas instancias, pero se comienza a distanciar conforme el número de estas sube hasta que con 60000 instancias el comportamiento de la versión dos es ya superior.

En el banco dos vemos como con pocos centros ambas versiones tienen un comportamiento muy similar, el rendimiento hasta los 10 centros es bastante parecido y a partir de 20 centros el comportamiento de la segunda versión es ya mejor claramente que el de la versión uno.

Todo paree indicar que la versión de mayor paralelismo tiene mejor rendimiento pero hay que señalar aquí que no se trata de un cluster real sino de tres máquinas virtuales, por lo que la penalización por el aumento del tráfico de red que sufre la versión dos no se pude contemplar de manera realista.

10. CONCLUSIONES

Los objetivos planteados al principio del trabajo se han alcanzado y la metodología seguida para ello ha demostrado ser adecuada. Nos hemos ido acercado paulatinamente a la finalidad planteada.

El objetivo principal es la paralelización de Fuzzy C-Means sobre MapReduce para su uso en Big Data.

Para ello nos hemos introducido en el Big Data y hemos visto su importancia en la actualidad, hemos profundizado en la Lógica Difusa y hemos visto algunos ejemplos de sus múltiples usos.

Una vez ahí introducimos el Aprendizaje Computacional y nos centramos en el Clustering, en qué consiste y sus diversos tipos. Ahora nos centramos en un tipo concreto, el Clustering difuso, y estudiamos el algoritmo de agrupación difusa Fuzzy C-Means. Vemos como trabaja y analiza los datos.

A partir de esto estamos en condiciones de acercarnos a la computación distribuida y al Cloud Computing lo que nos permite estudiar MapRedue con garantías, su funcionamiento y ver por qué es tan útil para el Big Data.

Ya podemos plantear las adaptaciones de nuestro algoritmo sobre MapReduce y estudiarlas cualitativamente y cuantitativamente. En una adaptación pretendemos minimizar el tráfico de red y en la otra maximizar el paralelismo.

Finalmente nos acercamos a la tecnología Hadoop para acabar desarrollado las adaptaciones y realizar estudios de validación y rendimiento con la intención de ver las posibilidades que nos ofrecen estas técnicas, Hadoop-MapReduce, sobre grandes cantidades de datos.

Se ha visto en la pruebas de validación la corrección de las adaptaciones y en las de rendimiento como una de las adaptaciones mejora el rendimiento conforme la cantidad de grupos aumenta al igual que lo hace cuando aumenta la cantidad de instancias a tratar. Esta versión es la que contiene más porcentaje de código a ejecutar en paralelo y no la que genera menos tráfico de red. Señalar que las pruebas se han realizado sobre un cluster virtualizado de tres nodos y en consecuencia no se puede contemplar un tráfico de red realista.

En cuanto a la evolución del proyecto se ha constatado la aparición de un riesgo ya previsto, problemas con la tecnología, esto hizo que el Sprint dos de la planificación del trabajo se completara al 80%. La solución para alcanzar el final, dedicar más tiempo, vino en el Sprint tres de la planificación.

Como líneas futuras de trabajo se platean diversas opciones, destacando:

- Crear nuevas adaptaciones pero esta vez basadas en dos Jobs.
- Realizar pruebas de rendimiento de todas las versiones en un cluster real y con millones de registros y estudiar los motivos de las diferencias que hubiere en el rendimiento.
- Estudiar como generar los centros iniciales para este algoritmo ya que de cómo se haga depende su velocidad para la convergencia de centros.
- Estudiar adaptaciones al modelo MapReduce de otros algoritmos de Inteligencia Artificial.

11. MANUAL DE USUARIO

Para comenzar recordar que estamos usando una distribución Hadoop de Cludera http://www.cloudera.com/

1. Para usar correctamente los programas obtenidos lo primero que tenemos que hacer es asegurarnos de que lo datos se encuentran en el formato adecuado.

   FORMATO DE LOS DATOS

   Se necesitan dos archivos de texto plano separado por comas, uno contendrá los datos a agrupar y el otro los centros iniciales.

   El archivo de centros debe llamarse obligatoriamente ***"centros".***

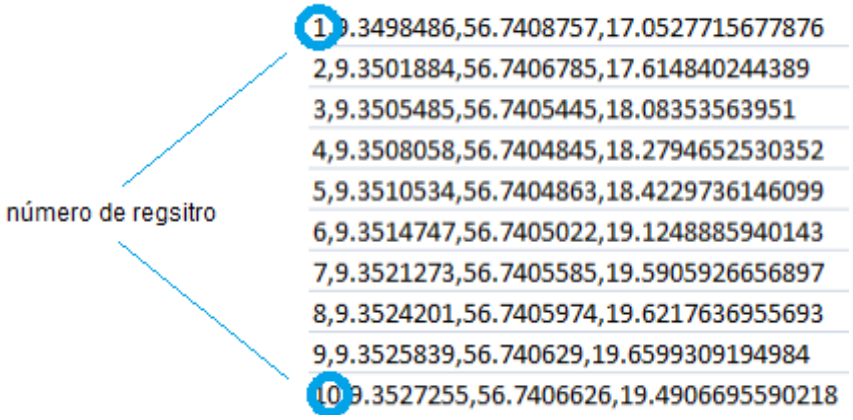
   Estos archivos no podrán contener valores nulos

   Los datos serán de tipo real.

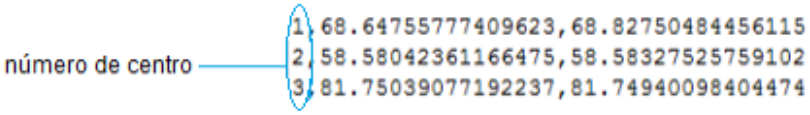
   Cada línea del archivo o registro será un objeto o centro.

   El primer valor de cada registro serán el número de registro y sin posibilidad de repetición.

   Ejemplo de archivo de datos



   Ejemplo de archivo de centros



2. Una vez los datos en el formato adecuado necesitamos cargarlos en el sistema de archivos de Hadoop. Imaginemos que queremos agrupar el archivo *perfume* usado en las pruebas de validación.

CARGA DE LOS DATOS EN HDFS

Par la carga de los datos en el sistema de archivos distribuido de Hadoop:

- copiar al directorio */home/cloudera* el archivo *perfume* y el archivo *centros*
- abrir un terminal, situarnos en */home/cloudera y* ejecutar los siguiente comandos

  hdfs dfs -put perfume

  hdfs dfs -put centros
- para ver que todo ha ido bien ejecutamos

  hdfs dfs -ls

3. Cargados en el sistema de archivos HDFS podemos ejecutar los programas.

EJECUCIÓN DEL PROGRAMA

Para ejecutar el programa debemos copiar a */home/cloudera* el archivo .jar correspondiente.

Ahora debemos ejecutar el siguiente comando

hadoop jar fuzzy.jar StubDriver perfume salida

Donde:

fuzzy.jar es el archivo .jar del programa

StubDriver es la clase que contiene el método *main*

perfume es el nombre del archivo con los objetos a agrupar

salida es el nombre del directorio donde obtendremos las pertenencias de cada objeto a cada grupo.

Una vez ejecutándose el programa pedirá por consola los siguientes valores:

- el valor del parámetro ***m*** comprendido entre [2.0, 2.5]
- el valor del parámetro $\boldsymbol{\varepsilon}$ comprendido entre [0.001, 0.01]

RESULTADOS DEL PROGRAMA

Terminado el programa:

- el archivo "***centros***" contendrá los nuevos centros de los grupos
- en la carpeta ***salida*** tendremos los archivos con los valores de pertenencias (*part-r-NNNNN*). Podemos verlos con el siguiente comando

hdfs dfs -ls salida

4. Finalmente debemos traer al sistema de archivos local los resultados

RECUPERACIÓN DE RESULTADOS

Para recuperar los resultados hemos de:

- borrar el archivo ***centros*** de */home/cloudera* si aún lo tenemos
- desde un terminal en */home/cloudera* ejecutar los siguientes comandos

hdfs dfs -get salida/part-r-NNNNN

hdfs dfs -get centros

12. Bibliografía

1.http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/big-data-the-next-frontier-for-innovation

2.http://www.oecd.org/sti/ieconomy/data-driven-innovation.htm

3.http://www-05.ibm.com/services/es/gbs/consulting/pdf/El_uso_de_Big_Data_en_el_mundo_real.pdf

4.http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/big-data-the-next-frontier-for-innovation

5.http://www.dataversity.net/not-your-type-big-data-matchmaker-on-five-data-types-you-need-to-explore-today/

6. http://www.cisco.com/assets/sol/sp/vni/forecast_highlights_mobile/

7.https://people.eecs.berkeley.edu/~zadeh/papers/Fuzzy%20Sets-Information%20and%20Control-1965.pdf

8. *[Sur, Omron, 1997] Sur A&C, Omron El*ectronics, S.A., "Lógica Fuzzy para Principiantes"

9.http://www.tdx.cat/bitstream/handle/10803/6887/04Rpp04de11.pdf;jsessionid=1216D36147DA7C122E21C0C86E150B46?sequence=4

10.http://www.esi.uclm.es/www/cglez/downloads/docencia/2011_Softcomputing/LogicaDifusa.pdf

11. http://www.lcc.uma.es/~eva/aic/apuntes/fuzzy.pdf

12.http://profesores.elo.utfsm.cl/~tarredondo/info/soft-comp/Introduccion%20a%20la%20Logica%20Difusa.pdf

13. http://dle.rae.es/

14. *Machine Learning,* Tom Mitchell, McGraw Hill, 1997.

15. http://users.dsic.upv.es/~jorallo/cursoDWDM/dwdm-III-3.pdf

16.http://www-users.cs.umn.edu/~kumar/dmbook/dmslides/chap8_basic_cluster_analysis.pdf

17. http://wiki.ead.pucv.cl/index.php/Chernoff_Faces

18. http://elvex.ugr.es/decsai/intelligent/slides/dm/D3%20Clustering.pdf

19. http://www.dc.uba.ar/materias/aa/2015/cuat2/clustering

20http://www.fuenterrebollo.com/Economicas/ECONOMETRIA/SEGMENTACION/CONGLOMERADOS/conglomerados.pdf

21.http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/cmeans.html#dunn

22. https://arxiv.org/ftp/arxiv/papers/1303/1303.0647.pdf

23. E. H. Ruspini, J. Information Sciences,vol. 2, no. 3,(1970)

24. J. C. Dunn (1973): "*A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters*", *Journal of Cybernetics* 3: 32-57

25. J. C. Bezdek (1981): "*Pattern Recognition with Fuzzy Objective Function Algoritms*", Plenum Press, New York

26. http://cb.uu.se/~joakim/course/fuzzy/vt10/CE1/Bezdek_FCM.pdf

27. https://poliformat.upv.es/access/content/group/OCW_6069_2008

28. http://webdiis.unizar.es/asignaturas/pscd/lib/exe/fetch.php?media=contenidos:leccion8.pdf

29.http://ocw.uc3m.es/ingenieria-informatica/desarrollo-de-aplicaciones-distribuidas/materiales-de-clase/paradigmas

30. http://ldc.usb.ve/~yudith/docencia/ci-4822/charlaUCV.pdf

31.http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

32.http://www.gutierrez-rubi.es/wp-content/uploads/2011/05/DT-Cloud_Computing-Ec.pdf

33.https://static.googleusercontent.com/media/research.google.com/es//archive/mapreduce-osdi04.pdf

34.http://static.googleusercontent.com/media/research.google.com/es//pubs/archive/32721.pdf

35.https://wikis.nyu.edu/display/NYUHPC/Big+Data+Tutorial+1%3A+MapReduce

36. http://hadoop.apache.org/index.pdf

37.http://blog.cloudera.com/blog/2014/01/how-to-create-a-simple-hadoop-cluster-with-virtualbox/

38. https://archive.ics.uci.edu/ml/datasets/Perfume+Data

39. https://archive.ics.uci.edu/ml/machine-learning-databases/00246/

40. Tom White (2015): *Hadoop: The Definitive Guide, 4th Edition.* O'Reilly Media