upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Título del proyecto:

"SPECTRAL CLUSTERING"

Daniel Ruiz Albéniz

Tutor: Jesús Villadangos Alonso

Pamplona, 30 de julio de 2010

Agradezco a mi tutor, Jesús, por haberme ofrecido este proyecto para finalizar mis estudios y por su interés, esfuerzo, rapidez a la hora de contestar a los e-mails, aguante y ayuda para que todo salga bien.

Agradezco a mis compañeros de universidad y sobre todo a mis compañeros de laboratorio especialmente a David, Leticia y María, por toda la ayuda mostrada a lo largo de la carrera y por supuesto, por aguantarme después de haber pasado tantas horas juntos.

Agradezco a mis padres, hermana, a mi novia Maitane y a toda mi familia, por preocuparse por mí en los momentos de agobio, por estar siempre pendientes de saber qué tal estoy y por supuesto, por estar siempre a mi lado.

Agradezco a toda la gente, amigos, conocidos en general… que me aguantan tal y como soy, alegre y divertido, y que por supuesto también me aguantan en mis malos momentos.

Gracias a todos.

ÍNDICE:

LISTADO DE FIGURAS:

RESUMEN:

En este capítulo se presenta el objeto del proyecto y se intenta introducir al lector en como se estructura el conocimiento de las personas a medida que avanza la tecnología así como las necesidad de los usuarios de obtener información de una manera clara y precisa.

En primer lugar, el objeto del proyecto consiste en aplicar el algoritmo de agrupación espectral (Spectral Clustering), implementado en matlab, sobre una matriz de datos, página web u otro tipo de aplicación que disponga de un conjunto finito de recursos, para agruparlos en clusters.

El resultado que obtendremos al aplicarlo nos va a permitir identificar una serie de grupos con un comportamiento similar en cuanto a los datos que éstos contienen. Además, obtendremos el tiempo medio de ejecución del algoritmo para agrupar según diferente número de clusters.

Sin embargo, una vez obtenidos los clusters, debemos asegurarnos de que los recursos agrupados dentro de un mismo cluster realmente tienen relación semántica de algún tipo entre sí. Por tanto, necesitamos un paso adicional que consiste en realizar dicha comparación utilizando un analizador semántico. Los resultados que obtendremos se muestran a modo de gráficos en el capítulo 3.

CAPÍTULO 1: INTRODUCCIÓN:

La generación del conocimiento en las organizaciones ha cambiado drásticamente como resultado del surgimiento de nuevos paradigmas asociados a la denominada sociedad de la información y a una nueva economía, basada en el conocimiento. Las nuevas generaciones han entendido que para construir conocimiento hay que ir al ritmo del crecimiento de la tecnología y a la evolución que experimenta la web, es por esto que las estructuras organizacionales se están apoyando en el ambiente colaborativo, cambiando desde sus sistemas de búsqueda de información hasta la forma como se ejecutan todas las acciones relacionadas con ella.

Para nadie es desconocido que la web actual trabaja por medio de léxico, mientras que la web 2.0, la web 3.0 y la web 4.0, que son las propuestas del futuro, se les ha dotado de más significado y semántica.

El desarrollo de las ***tecnologías de la información y la comunicación (TIC)*** modifica la sociedad, y si el medio se transforma, también cambia el acceso a la información, que es hoy mucho más rápido, amplio y preciso. Existe una mayor demanda de información y una participación más directa en la búsqueda de información por parte de los usuarios. Bajo esta dinámica, las TIC pueden facilitar parte de las condiciones relacionadas con la implementación de una gestión del conocimiento, al soportar y potenciar las capacidades de cada uno de los miembros de una organización.

En los últimos años, diversos ***sistemas de organización del conocimiento (SOC)*** se han utilizado con éxito para ordenar la información en la web: desde los tradicionales sistemas de clasificación, hasta las más novedosas redes semánticas.

Por lo anterior, las organizaciones han involucrado en su gestión el enfoque colaborativo que la web brinda, convirtiéndolo en un elemento importante para que sus usuarios y colaboradores encuentren la información de forma fácil, entendible, completa, oportuna, confiable y objetiva.

Aquí es donde toman parte los términos *ontología y folksonomía* como base para la construcción del conocimiento.

Así pues, las ***ontologías*** incluyen definiciones de conceptos básicos en un campo determinado y las relaciones entre ellos, es decir, se encargan de definir los términos utilizados para describir y representar un área de conocimiento. Definen vocabularios que facilitan la búsqueda mediante una herramienta (buscador) que pueda entender, ya que son especificados con la suficiente precisión como para permitir diferenciar términos y reverenciarlos de manera precisa. Además, Las ontologías se caracterizan por tener componentes que sirven para representar el conocimiento de algún dominio.

Por último, destacar que podemos encontrar ontologías en portales web (reglas de categorización para mejorar la búsqueda), colecciones multimedia (búsquedas basadas en contenidos no textuales), etc.

En cuanto a la ***folksonomía***, se emplea para designar a un sistema de etiquetado o clasificación de objetos web no jerárquico que nace de forma natural y democrática de los propios internautas, que son quienes asignan las etiquetas espontáneamente, y de cuya gestión se encarga un sistema automático.

Dicho término, proviene del neologismo inglés "folksonomy", nombre a la categorización colaborativa por medio de etiquetas simples o tags en un espacio de nombres llano, sin jerarquías ni relaciones de parentesco predeterminadas. Es una práctica que se produce en entornos de software social, cuyos mejores ejemplos son los sitios compartidos como *Del.icio.us* (enlaces favoritos), *Flickr* (fotos), *Tagzania* (lugares), *Technoraty* (blogs) o *43 Things* (deseos). A continuación se explican brevemente algunos de los mismos.

-Del.icio.us (http://del.icio.us): los usuarios pueden crear y compartir anotaciones sobre recursos de manera similar al conocido "añadir a favoritos".

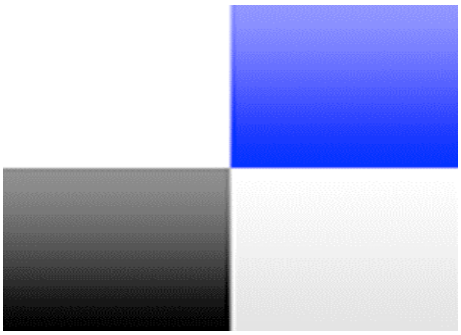


Figura 1.1. Del.icio.us

-Flickr (http://www.flicker.com): es un sistema creado para compartir fotos, en el cual cada usuario puede compartir y etiquetar sus fotografías personales.

Figura 1.2. Flickr

-Technorati (http://www.technorati.com): permite a los autores etiquetar los post de sus propios blogs, agregando información contenida en otros y facilitando su búsqueda.

Figura 1.3. Technorati

-***CiteULike*** (http://www.citeulike.org): clasifica los trabajos que se recomiendan mediante tags o etiquetas, al igual que cualquier otro sitio social relacionado con la ciencia.

Figura 1.4. Cite u Like

En cuanto a su funcionamiento, las folksonomías se apartan de las estructuras jerarquizadas para aproximarse a una organización basada en la colaboración. Son un método de clasificación explotado por varios servicios web donde los usuarios añaden información o fotografías o clasifican páginas web. La clasificación no se realiza a través de una serie de categorías fijas y jerárquicas, como tradicionalmente se ha hecho, sino a través de lo que se denominan *tags o etiquetas* que son añadidas y administradas libremente por las personas que usan los sistemas. Las decisiones de etiquetado las toman los propios usuarios, permitiéndose el uso de más de una *etiqueta* para clasificar cada elemento relacionado con un mismo tema y dichas etiquetas aportan *metadatos* sobre lo que el usuario piensa que es el tema solicitado.

Debido a esto, permite generar datos producidos por la participación de miles de usuarios. Este sistema colaborativo usado por muchas aplicaciones en red materializa la arquitectura de la participación, así como las ideas de la inteligencia colectiva y la intercreatividad. A día de hoy, esto se conoce con el nombre de web 2.0.

En cuanto a las ***ventajas*** que producen las folksonomías, se encuentran la formación de los usuarios en temas específicos, el desarrollo del aprendizaje colaborativo y la habilidad de construir nuevo conocimiento que redundará en un mayor y mejor aprovechamiento para el crecimiento intelectual. Todo esto, sumado, logrará generar una ventaja competitiva sostenible dentro del entorno de competencia en que se desenvuelven tanto los usuarios como las organizaciones.

Sin embargo, no todo es tan bonito como parece, pues también presenta sus ***problemas***, ya que aún se encuentra en fase de exploración pese a que numerosas organizaciones ya las utilicen en sus páginas web y demás.

Así pues, pese a que organicen la información por temas relacionados, son *demasiado genéricas* por no poder emplearse expresiones compuestas en los tags o etiquetas (un único término), que se pierden la especificidad y las relaciones jerárquicas, también existe un uso de *lenguaje incontrolado* que incrementa la *ambigüedad de los conceptos*, ya que para un usuario un tema o un concepto puede significar una cosa completamente diferente que para otro usuario.

De momento, su aplicación presenta muchas debilidades, ya que también ofrece un *lenguaje controlado*, pues *no tiene jerarquías* ni otro tipo de asociaciones más allá de los posibles clusters de recursos que se puedan formar según las clasificaciones o palabras clave de los usuarios.

Debido a esto, el hecho de encontrar un agrupamiento correcto de datos, ha sido el foco de considerables investigaciones en el ámbito del aprendizaje automático y reconocimiento de patrones.

Como veremos en los siguientes capítulos, la agrupación spectral (***Spectral Clustering*** [OSC01] [GBS00], se ha convertido en una de las técnicas de agrupamiento más populares dentro de los diferentes algoritmos que se pueden aplicar a la hora de *solventar los problemas citados* anteriormente.

Dada la sencillez a la hora de aplicar el algoritmo, podremos superar, en cuanto a eficiencia se refiere, utilizando métodos tradicionales del álgebra lineal, a otros algoritmos de clustering como el mítico *k-means* [MJI10] u otros.

El algoritmo ***k-means***, para agrupar una serie de datos, comienza con un conjunto de centros iniciales y se ejecuta de manera iterativa. De este modo, en cada iteración se asocia cada punto al grupo caracterizado por el centro más cercano, y una vez computados todos los puntos, se generan nuevos centros a partir de la media de cada grupo.

El problema que nos muestra este algoritmo es que los resultados obtenidos y la velocidad de convergencia dependen significativamente de los valores iniciales asignados a los centros. Sin embargo, existen otros que inicializan estos valores de manera que la precisión (en cuanto a distancias se refiere) y la velocidad mejoran considerablemente.

Uno de ellos es el algoritmo de **R.Maitra** [MJI10], el cual juega con una serie de modas locales multidimensionales y las ajusta con k-means. Su *elevadísimo coste computacional* genera un número enorme de modas, lo cual hace que su uso sea casi computacionalmente prohibitivo.

Existe otro que se conoce como el k-means++: el algoritmo de **D.Arthur** [MJI10]. Se *selecciona un centro al azar*. Éste consta de "k" iteraciones, en cada una de las cuales se debe calcular la distancia de cada punto al centro obtenido en la iteración anterior, es decir, "n" cálculos de distancias. Por tanto, su complejidad es de *O(nkp)*.

Un algoritmo muy simple es el de **G.C.Tseng** [MJI10]. En él, se obtienen KxP clusters utilizando clustering jerárquico y de entre ellos se seleccionan los "k" más poblados. Su complejidad vendrá dada por la ejecución del algoritmo de cluster jerárquico, ya que la obtención de tamaño de cada cluster es una operación de complejidad *O(n)* y la obtención de los "k" más poblados es una ordenación que simplemente requiere *O(n lg n)* operaciones.

También está el caso del algoritmo de **M.B.Al-Daoud [**MJI10], cuyo coste computacional viene determinado en su primer paso: realizar un análisis de componentes principales sobre los datos de la matriz. Éste coste es de $O(p^3)$.

Así pues, el algoritmo de *inicialización aleatoria* aplicado sobre el k-means, nos devuelve, si comparamos las distancias mínimas obtenidas al clasificar recursos en una matriz completa, la menor distancia y su coste computacional no es muy elevado, a diferencia de aplicar los algoritmos de *Tseng* o el de *Maitra*. En cuanto a rapidez, también se posiciona con mejores resultados otro algoritmo junto con el de inicialización aleatoria: el de *inicialización uniforme*.

Otro algoritmo que pertenece a la familia del k-means (algoritmos de particionado y recolocación) es el ***EM*** [MJM05]. Esto hace que los resultados obtenidos

con uno u otro sean similares, pero destacando que el EM realiza una división más específica sobre los mismos datos que el k-means.

Una buena alternativa que es utilizada en una gran variedad de campos es usar métodos espectrales para el agrupamiento. El ***Spectral Clustering*** usa los vectores propios mayores de una matriz que contiene la distancia entre puntos como veremos en el siguiente apartado, donde se desarrollará todo lo relacionado con el algoritmo implementado. Éste método ha sido utilizado en numerosas aplicaciones, como por ejemplo, en el diseño de visión por computador (VLSI), ofreciendo buenos resultados.

Sin embargo, a pesar de estos resultados empíricos, que en principio parecen muy convincentes, diferentes personas que han estudiado éste prometedor algoritmo no se ponen de acuerdo a la hora de elegir qué vector propio utilizar, al igual que cómo derivar los "clusters" una vez elegidos los mismos. Este desacuerdo ha propiciado que en la actualidad se tienda a enfocar el tema sobre algoritmos que únicamente utilizan un vector propio.

Para finalizar con este capítulo, anteriormente se ha explicado el hecho de que el clustering nos ayuda a buscar relaciones semánticas, pero… *¿cómo asegurarnos de que realmente exista la relación semántica entre todos los tags agrupados en un cluster y los recursos etiquetados por dichos tags*?

La respuesta a esta pregunta la veremos en el capítulo correspondiente al ***analizador léxico***. Con esta aplicación mediremos la similitud que existe entre las diferentes etiquetas y sus recursos asociados, probando si verdaderamente esa relación es correcta o no.

REFERENCIAS:

[OSC01] Andrew Y. Ng, Michael I. Jordan, Yair Weiss (2001). On Spectral Clustering: Analysis and an algorithm. NIPS 2001 Conference.
http://ai.stanford.edu/~ang/papers/nips01-spectral.pdf

[GBS00] R. Kannan, S. Vempala and A.Vetta (2000). On Clusterings: good, bad and spectral. In Proceedings of the 41st Annual Symposium on Foundations of Computer Science, 2000.

[MJI10] Miguel Liroz, Javier Armendáriz, Iria Prieto (2010). Web Semántica: estudio de algoritmos de inicialización para clasificación con k-means. Máster en Tecnologías Informáticas (UPNA), 2010.

[MJM05] Miguel Garre, Juan José Cuadrado, Miguel Ángel Sicilia (2005). Comparación de diferentes algoritmos de clustering en la estimación de coste en el desarrollo de software. Dpto. de Ciencias de la Computación, Universidad de Alcalá, 2005.
http://www.sc.ehu.es/jiwdocoj/remis/docs/GarreAdis05.pdf

CAPÍTULO 2: SPECTRAL CLUSTERING:

En este capítulo se va a mostrar cual es el algoritmo implementado, así como el código del mismo en el lenguaje de programación matlab y los resultados obtenidos al aplicarlo sobre diferentes fuentes tales como un conjunto aleatorio de puntos, una matriz de recursos, etc.

Además, se presentarán las dificultades que ha presentado la versión codificada y las decisiones tomadas para conseguir los resultados más óptimos posibles.

2.1 ALGORITMO:

Dado un conjunto de "n" puntos, donde S= $\{s_1, s_2, \ldots, s_n\}$ en $\Re^l$, podemos agruparlos en C clusters (subconjuntos), de la siguiente manera:

1. Formamos la *matriz afinidad* $A \in \Re^{n \times n}$ definida por: $A_{ij} = e^{-\left(\frac{\|s_i - s_j\|^2}{2\sigma^2}\right)}$ para $(i \neq j)$ y $(A_{ii}=0)$, donde (s_i, s_j) es la distancia euclídea entre los vectores s_i y s_j. σ es un parámetro de escala, cuyo funcionamiento veremos luego.

2. Definir *D* como la *matriz diagonal* con $D_{ii} = \sum_{j=1}^{n} A_{ij}$ y construir la *matriz afinidad normalizada* $L = D^{\frac{-1}{2}} A D^{\frac{-1}{2}}$.

3. Manualmente, definimos un número aleatorio de clusters para realizar la agrupación.

4. Encontrar $x_1, \ldots, x_C$, con los C mayores vectores propios de L y *formar la matriz* $X = [x_1, \ldots, x_C] \in \Re^{n \times C}$.

5. *Renormalizar las filas de X* obteniendo como longitud la unidad: $Y \in \Re^{n \times C}$ así como: $Y_{ij} = \frac{X_{ij}}{\left(\sum_j X_{ij}^2\right)^{1/2}}$.

6. Tratar cada fila de Y como un punto de $\Re^C$ y *agrupar por vía* del *k-means*.

7. *Asignar el punto original* s_i *al cluster C* si, y solo si, la correspondiente fila i de la matriz Y ha sido asignada al cluster C.

2.2 CÓDIGO:

Como se ha dicho anteriormente, en éste apartado vamos a mostrar el código correspondiente al algoritmo implementado. El objetivo es dar a conocer las ideas principales de cada punto para lograr comprenderlo lo mejor posible.

En primer lugar, disponemos de una matriz "*X*" con 12106 filas y 1939 columnas (que se corresponden con un conjunto de recursos y etiquetas respectivamente). Cada celda de dicha matriz tiene un valor entero que significa el número de veces que se ha utilizado una etiqueta para etiquetar un recurso.

Una vez sabido esto, nos disponemos a aplicar el algoritmo de Spectral Clustering sobre la misma para obtener la etiqueta correspondiente a cada recurso y poder analizar las distancias "*intra-cluster*" y "*entre-clusters*" para contrastar posteriormente los resultados obtenidos.

Comenzamos por la declaración de la función *distancias*:

```
function [medias_intra maximos_intra minimos_intra dist_intra t_total etiquetas VMC dist_entre]=distancias(X)
%distancias calcula la distancia intra-clusters y entre-clusters.
%
%   Entradas:   X           :Macro-Matriz de entrada (12106x1939).
%
%   Salidas:    dist_intra  :Distancias entre los recursos que componen un
%                            cluster.
%               dist_entre  :Distancias entre recursos medios de cada
%                            cluster.
%
```

Ésta será la función que calcule las distancias anteriormente nombradas. Por tanto, como entrada tenemos la matriz de datos y las salidas son las correspondientes distancias que queremos obtener. Como datos adicionales para el análisis nos interesan el tiempo total de ejecución, las etiquetas, etc.

```
load('../Pruebas/matrizSinCeros.mat');
disp('Calculando las distancias intra-clusters...');
clusters=input('Indica el numero de clusters: ');
disp('Probando con: '), disp(clusters), disp(' clusters.');

A=X(1:200,:);   %Copio las 200 primeras filas de la "macro-matriz" en 'M'.
B=A;            %Copio la matriz ya que va a sufrir cambios.
tic; %Empezamos a contar el tiempo.
gen_nn_distance(A,5,1,0); %Matriz A; 5 vecinos; tam. bloque 1; resultado en
                          %en fichero '.m'.
load('5_NN_sym_distance.mat'); %Cargo el fichero de salida anterior que tiene
                               %los datos de los vecinos de cada recurso.
[etiquetas]=sc(A,0,clusters); %funcion sc: matriz A; sigma=0; clusters.
```

El primer paso consiste simplemente en cargar la matriz de datos "*matrizSinCeros.mat*" que se corresponde con la que previamente hemos llamado "*X*".

A continuación, vamos a indicar el número de clusters que queremos obtener a la hora de realizar la agrupación (paso 3 en la sección 2.1). A pesar de que éste parámetro puede introducirse manualmente, si queremos obtener un correcto clustering de nuestros recursos, tomaremos dos caminos que nos van a ayudar a escoger el número adecuado de clusters.

El primero de ellos consiste en analizar los *valores propios* de la matriz afinidad. Los diversos análisis realizados en el ámbito del clustering demuestran que si escogemos el mayor valor propio de la matriz "*L*" (ver algoritmo 2.1), éste será un valor propio repetido con valor la unidad (igual a 1), cuya multiplicidad será igual al número de grupos o clusters que debemos escoger.

El segundo consiste en obtener los *vectores propios*, después de ordenar *L* de acuerdo al número de clusters, en el caso ideal tendremos L una matriz diagonal, cuyos valores serán 1, 2, …, C (siendo *C* el número de clusters).

Una vez introducido el número de clusters, para agilizar las pruebas, nos disponemos a seleccionar las 200 primeras filas (por ejemplo) de nuestra "macro-matriz". De esta manera nos será mucho más rápida y sencilla la ejecución del algoritmo.

Llega un punto importante del código: la formación de la *matriz dispersa de distancias* a partir de nuestra matriz inicial.

```
function gen_nn_distance(data, num_neighbors, block_size, save_type)
%gen_nn_distance genera (con los "t" vecinos m·s cercanos de cada dato) una matriz dispersa de distancias.
%
%   Entrada : data         : Matriz NxD de datos, donde N son los datos y D
%                            la dimension de la matriz.
%             num_neighbors: N¨mero de vecinos m·s cercanos a cada dato.
%             block_size   : TamaÒo de bloque para el particionamiento de
%                            la matriz.
%             save_type    : 0 fichero .mat, 1 fichero .txt, 2 para ambos.
%
%   Salida : Fichero .mat, o .txt o ambos.
%
```

Como podemos ver en la declaración de la función, como entrada tenemos: *data* (nuestra matriz X), *num_neighbors* (número de vecinos de cada dato), *block_size* (tamaño de bloque para particionar nuestra matriz inicial) y *save_type* (formato del fichero de salida que cargaremos posteriormente).

```
%
% Divido los datos en bloques y los proceso por separado para un mejor uso
% de la memoria.
%
tic;
disp('Computando las distancias de la matriz no simetrica...')
n = size(data, 1);
num_iter = ceil(n/block_size);
disp(['Numero de Iteraciones ', num2str(num_iter)]);
A = sparse(n, n);
dataT = data';
```

Lo primero que debemos hacer es dividir nuestra matriz inicial en bloques (ya definido su tamaño con la variable de entrada *block_size*) como se muestra, calculando las distancias en la matriz (cuyo tamaño es *NxD*) así como el número de iteraciones que vamos a tener que realizar para ello.

```
% Calculando los datos para la distancia euclÍdea.
%
tmp = full(sum(data.*data, 2));
aa = tmp(:, ones(block_size, 1));
clear tmp;

for i = 1:num_iter
  disp(i);
  start_index = 1 + (i-1)*block_size;
  end_index = min(i*block_size, n);
  num_data = end_index - start_index + 1;



% Selecciono un bloque de datos para trabajar con Él.
block = dataT(:, start_index:end_index);

% Calculo la distancia euclÍdea entre los datos del bloque.
if (num_data < block_size)
  aa = aa(:, 1:num_data);
end
tmp = full(sum(block.*block, 1));
bb = tmp(ones(n, 1), :);
clear tmp;
ab = full(data*block);
dist = aa + bb - 2*ab;
clear bb ab block;
dist = max(dist, 0);
```

Una vez tenemos los bloques, necesitamos saber el número de datos que tiene cada uno de ellos. De esta manera calcularemos, como se observa en el anterior pantallazo, la distancia euclídea entre cada par de puntos.

```
  % Encuentro los vecinos m·s cercanos a cada dato.
  [value index] = sort(dist, 1);
  tempindex = index(2:num_neighbors+1, :);
  rowindex = reshape(tempindex, size(tempindex, 1)*size(tempindex, 2), 1);
  tempindex = repmat(1:num_data, num_neighbors, 1);
  columnindex = reshape(tempindex, size(tempindex, 1)*size(tempindex, 2), 1);
  tempvalue = value(2:num_neighbors+1, :);
  value = reshape(tempvalue, size(tempvalue, 1)*size(tempvalue, 2), 1);
  value = sqrt(max(value, 1.0e-12));
  A(:, start_index:end_index) = sparse(rowindex, columnindex, double(value), n, num_data);
end
clear data dataT tempindex rowindex columnindex tempvalue value;
toc;
```

Es el momento de calcular el número de vecinos de cada dato. Así pues, dependiendo del número introducido de los mismos, lo que haremos será ordenar los valores de la matriz que contiene las distancias euclídeas formada anteriormente, de manera creciente, e ir incluyendo los datos más cercanos en nuestra matriz A, que será la dispersa. Éste proceso se realizará de manera iterativa hasta acabar con el conjunto de valores.

```
% Matriz de distancias SimÈtrica.
disp('Computando la matriz simetrica de distancias...')
A1 = triu(A);
A1 = A1 + A1';
A2 = tril(A);
A2 = A2 + A2';
clear A;
max_num = 100000;
if (n < max_num)
  A = max(A1, A2);
else
  num_iter = ceil(n/max_num);
  B = sparse([]);
  for i = 1:num_iter
    disp(i);
    start_index = 1 + (i-1)*max_num;
    end_index = min(i*max_num, n);
    B = max(A1(:, start_index:end_index), A2(:, start_index:end_index));
    % Matriz temp (temporal) para cuidar el uso de memoria.
    tmpfile = ['tmp_', num2str(i), '.mat'];
    save(tmpfile, 'B');
    clear B;
  end
end
clear A1 A2;
toc;
```

Como se observa en el código implementado, *A1* se corresponde con los elementos situados por encima de la diagonal de la matriz *A*, y por el contrario, *A2* con los situados por debajo. Si el número de filas de nuestra matriz de datos (la inicial) es menor que una constante declarada (en este caso *max_num*=100000), guardaremos en la matriz *A*, el máximo de las dos submatrices anteriores (*A1* y *A2*). En caso contrario, nos situaremos en la iteración que corresponde en cada caso, y calcularemos de nuevo las matrices superior e inferior a la diagonal de *A*, guardando el máximo en la variable *B*.

```
% Concateno todas las matrices temporales.
%
disp('Concatenando matrices temp...');
if (n > max_num)
  A = sparse([]);
  for i = 1:num_iter
    tmpfile = ['tmp_', num2str(i), '.mat'];
    load(tmpfile);
    A = [A B];
    clear B;
  end
end
delete tmp*;

%
% Diagonal de la matriz simÈtrica = 0.
%
n = size(A, 1);
B = spdiags(diag(A), 0, n, n);
A = A - B;
```

Una vez hecho lo más complicado, simplemente concatenamos como se indica las matrices guardadas temporalmente, y hacemos que la diagonal de nuestra matriz simétrica sea 0 por aquello de que la distancia de un punto a si mismo debe ser nula.

```
%
% Guardo la matriz simÈtrica de distancias en .mat o .txt.
%
if (save_type == 0) || (save_type == 2)
  disp('Guardando fichero .mat...');
  outfile = [num2str(num_neighbors), '_NN_sym_distance.mat'];
  save(outfile, 'A');
end

if (save_type == 1) || (save_type == 2)
  disp('Escribiendo fichero .txt...');
  outfile = [ num2str(num_neighbors), '_NN_sym_distance.txt'];
  fid = fopen(outfile, 'w');
  n = size(A, 1);
  for i = 1:n
    if mod(i, 10000) == 0
      disp(i);
    end
    [row_index col_index value] = find(A(:,i));
    fprintf(fid, '%d %d', i-1, length(row_index));
    index = [row_index'-1; value'];
    index = reshape(index, size(index, 1)*size(index, 2), 1);
    fprintf(fid, ' %d:%E', index);
    fprintf(fid, '\n');
  end
  fclose(fid);
end
toc;
```

Finalmente, el resultado quedará almacenado en un fichero "*.mat*", "*.txt*" o en *ambos*, dependiendo de nuestras preferencias. Para ello, habremos introducido 0, 1 o 2 respectivamente en la variable *save_type*, a la hora de llamar a la función.

Volviendo a nuestra función "*distancias.mat*", y ya explicados los pasos a seguir para calcular la matriz dispersa de distancias, la cargaremos con la orden *load* como se indicaba. Nuestro siguiente cometido es obtener los recursos de nuestra matriz de datos agrupados en diferentes clusters, asignando etiquetas a cada uno de ellos para diferenciarlos. Este proceso lo realiza la función "*sc.mat*".

Así pues, el punto de partida son una serie de puntos, los cuales definen las componentes de un vector que denominamos *S*, de forma que se cumple: $S = \{s_1, s_2, ..., s_n\}$, perteneciendo éste a un espacio vectorial $\Re^l$. Nuestra finalidad consiste en agrupar éstos vectores en un número "*C*" de subconjuntos o clusters, los cuales mantengan una relación que determinaremos en el siguiente capítulo mediante el analizador semántico.

La función queda definida de la siguiente manera:

```
function [predict_labels evd_time kmeans_time total_time] = sc(A, sigma, num_clusters)
%sc (Spectral Clustering) usando una matriz dispersa de distancias.
%
%   Entradas: A              : NxN matriz dispersa de distancias (N= nº de datos).
%             sigma          : El valor sigma es usado en similitud computacional.
%                              Si vale 0, aplicamos la tÈcnica "self-tunning".
%             num_clusters   : N`mero de clusters.
%
%   Salidas:  predict_labels : Vector Nx1 que contiene las etiquetas.
%             evd_time       : Tiempo empleado para la descomposicion de los vectores propios.
%             kmeans_time    : Tiempo empleado en el algoritmo k-means.
%             total_time     : Tiempo total empleado.
%
```

Tal y como se muestra, llamamos a la función *sc* con una matriz A cuadrada (*NxN*) que se denomina matriz dispersa de distancias (calculada anteriormente), siendo *N* el número de datos.

Sigma es un parámetro utilizado en similitud computacional y veremos que depende de los datos utilizados. Además, si su valor es 0, aplicaremos una técnica específica denominada “Self-Tunning”.

La variable *num_clusters* simplemente sirve para introducir el número deseado de subconjuntos que queremos obtener.

En cuanto a las salidas, *predict_labels* son las etiquetas asignadas a los diferentes recursos. Por tanto, es una matriz de N filas y una única columna que contiene el número del cluster asignado al recurso.

Por último, *evd_time*, *kmeans_time* y *total_time* son variables que nos muestran el tiempo empleado en la descomposición de los vectores propios, la ejecución del algoritmo k-means y el total de la función sc implementada. Esto nos vendrá bien a la hora de comparar resultados en cuanto a coste computacional.

```
%
%Convierte la matriz dispersa de distancias a una matriz dispersa de similitudes.
%donde S = exp^(-(A^2 / 2*sigma^2)).
%
%Nota: Podemos omitir este paso si A ya es una matriz dispersa de similitudes.
%
disp('Convirtiendo la matriz de distancias a una matriz de similitudes...');
tic;
n = size(A, 1); %Numero de filas de la matriz A.

if (sigma == 0) % "Self-tunning"
  % Contamos los valores que no son 0 de cada columna.
  col_count = sum(A~=0, 1)';
  col_sum = sum(A, 1)';
  col_mean = col_sum ./ col_count;
  [x y val] = find(A);
  A = sparse(x, y, -val.*val./col_mean(x)./col_mean(y)./2);
  clear col_count col_sum col_mean x y val;
else % Matriz Afinidad.
  A = A.*A;
  A = -A/(2*sigma*sigma);
end
```

Tal y como se indica en el punto primero del algoritmo, formamos la matriz Afinidad o de similitudes. Cabe destacar que, como se puede ver en el primer comentario, si los datos se introducen ya en una matriz dispersa de similitudes, en vez de en una simétrica de distancias simplemente, es obvio que puede omitirse este paso.

En cualquier caso, nuestra matriz es $A_{ij} = e^{-\left(\frac{\|s_i - s_j\|^2}{2\sigma^2}\right)}$, con $(i \neq j)$ y $(A_{ii}=0)$. En ésta expresión, $\|s_i - s_j\|^2$ se corresponde con la distancia euclídea entre cada par de vectores (s_i, s_j). Para realizar la función exponencial, lo haremos de manera secuencial para evitar errores en cuanto a la limitación de memoria, puesto que trabajamos con una matriz que contiene un gran número de datos.

```
%La funcion exponencial se hace secuencialmente debido a las limitaciones
%de la memoria.
num = 2000;
num_iter = ceil(n/num);
S = sparse([]);
for i = 1:num_iter
  start_index = 1 + (i-1)*num;
  end_index = min(i*num, n);
  S1 = spfun(@exp, A(:,start_index:end_index));
  S = [S S1];
  clear S1;
end
clear A;
toc;
```

En cuanto al parámetro *sigma* [STSC04] o parámetro de escala, es una medida utilizada cuando dos puntos son considerados similares. En principio, la selección de este parámetro es intuitiva y a tal efecto suele realizarse manualmente. Sin embargo, Ng-Jordan-Weiss [OSC01] sugirió una manera de hacer esta selección de manera automática ejecutando su algoritmo de clustering repetidamente para un número diferente de valores de σ, eligiendo aquel que menos distorsiona los subconjuntos de las filas de la matriz Y.

A priori, parece una buena elección el uso de este método, pero su coste computacional se incrementa notablemente y además seguimos con el problema de tener que indicar manualmente el rango de valores que va a tomar el parámetro σ. Todo ello sumado a que no siempre existe un único valor que nos haga un buen clustering, sino que hay casos en los que varios de ellos se ajustan perfectamente.

Podemos comprobar el efecto que tiene el parámetro de escala sobre un mismo conjunto de datos al variar su valor. Aquí están algunos ejemplos:

$\sigma = 0.041235$

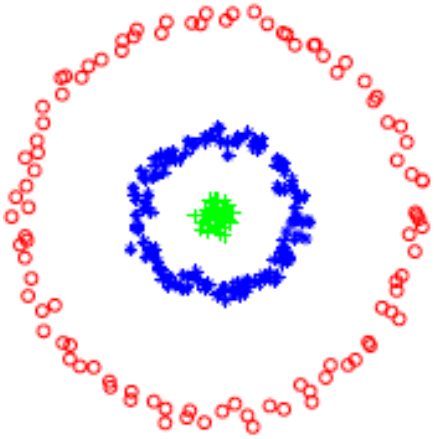


Figura 2.1. $\sigma = 0.041235$

$\sigma = 0.015625$

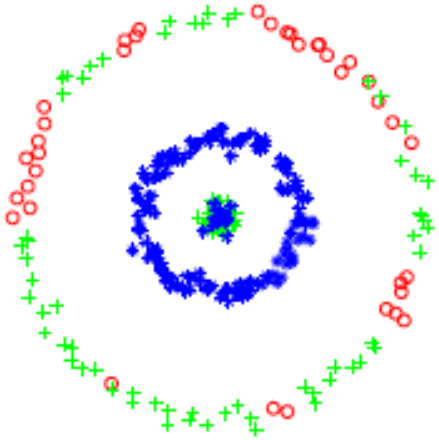


Figura 2.2. $\sigma = 0.015625$

$\sigma = 0.35355$

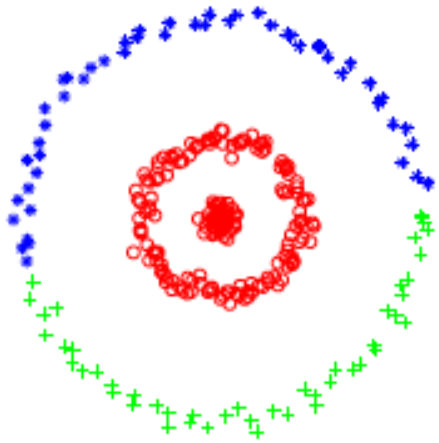


Figura 2.3. $\sigma = 0.35355$

σ = 1

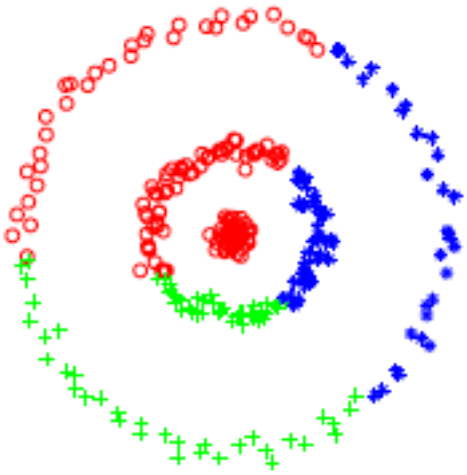


Figura 2.4. σ = 1

De acuerdo a estos ejemplos, lo que se propone es calcular un parámetro de escala "local" para cada punto s_i. La selección de dicho valor puede llevarse a cabo estudiando los "vecinos" de cada punto tal y como se ve en la implementación del algoritmo en matlab y como hemos comentado antes al formar la matriz simétrica de distancias.

Además de los anteriores, veremos los efectos de *sigma* variando su valor al aplicarlo sobre nuestros ejemplos.

```
%
% Matriz laplaciana, L = D^(-1/2) * S * D^(-1/2)
%
disp('Matriz Laplaciana...');
D = sum(S, 2) + (1e-10);
D = sqrt(1./D); % D^(-1/2)
D = spdiags(D, 0, n, n);
L = D * S * D;
clear D S;
time1 = toc;
```

El segundo punto de nuestro algoritmo de Spectral Clustering consiste en construir la matriz afinidad normalizada (*Matriz Laplaciana*), la cual denominaremos "*L*". La variable "*D*" se corresponde con una matriz diagonal, que contiene el grado de cada vértice. Nos será de gran utilidad el cálculo de "*L*" puesto que a partir de ella escogeremos los vectores propios como se ve en el siguiente paso.

```
%
% DescomposiciÛn de vectores propios: if L =
%   D^(-1/2) * S * D(-1/2)    : 'LM' (M·xima Magnitud), o
%   I - D^(-1/2) * S * D(-1/2): 'SM' (MÌnima Magnitud).
%
disp('Realizando la descomposiciÛn de vectores propios...');
OPTS.disp = 0;
[V, val] = eigs(L, num_clusters, 'LM', OPTS);
time2 = toc;
```

Hacemos la descomposición de los vectores propios tal y como se observa. Escogemos los mayores vectores propios ("*LM*"), atendiendo al número de clusters que hemos introducido previamente. Este proceso se corresponde con el paso 4 de la sección 2.1.

Por último, los puntos 5 y 6, consisten en normalizar la matriz (para que la longitud del vector propio sea la unidad) y aplicaremos el algoritmo de agrupación k-means para obtener las etiquetas de cada cluster. En matlab existe una función propia que nos hace este paso automáticamente para ahorrarnos tiempo, pero también es posible encontrarlo implementado ya en numerosos lugares de la web. Aquí se muestra la última parte del código en matlab:

```
%
% k-means.
%
disp('Aplicando el algoritmo kmeans...');
% Normalizamos cada fila de la matriz para que tenga longitud=1.
sq_sum = sqrt(sum(V.*V, 2)) + 1e-20;
U = V ./ repmat(sq_sum, 1, num_clusters);
clear sq_sum V;
predict_labels = k_means(U, [], num_clusters);
total_time = toc;

%
% Calculo y muestro los datos referentes al tiempo.
%
evd_time = time2 - time1
kmeans_time = total_time - time2
total_time
disp('Fin!');
```

Ya solo nos queda comentar la última parte del análisis: el cálculo de las distancias "*intra-cluster*" y "*entre-clusters*".

```
%
%Bucle que calcula las distancias.
%
clear v;
clear dist_intra;
clear dist_entre;
clear VMC;
v=[];
VMC=[];
medias_intra=[];
maximos_intra=[];
minimos_intra=[];
for i=1:clusters
    a=find(etiquetas(:,1)==i);
    for j=1:size(a,1)
        v=[v;B(a(j),:)];
        v(j,:)=v(j,:)/max(v(j,:));
    end
    VMC=[VMC; mean(v)];


    %Matriz distancia INTRA-CLUSTERS.
    dist_intra=[];
    dist_minima=[];
    for k=1:size(a,1)
        for p=1:size(a,1)
            disp('Calculando distancias del recurso: '), disp(a(k,1));
            disp(' al recurso: '),disp(a(p,1));
            dist_intra(k,p)=[sum((X(a(k,1),:)-X(a(p,1),:)).^2).^0.5];
        end
    end

    if(size(dist_intra,1)==1)
        minima_dist=0;
    else
        matriz_ordenada=sort(dist_intra,2);
        matriz_ordenada(:,1)=[];
        minima_dist=min(min(matriz_ordenada));
    end

    medias_intra=[medias_intra;mean(mean(dist_intra))];
    maximos_intra=[maximos_intra;max(max(dist_intra))];
    minimos_intra=[minimos_intra;minima_dist];

    a=[];
end

%Matriz distancia ENTRE-CLUSTERS.
dist_entre=[];
for l=1:clusters
    disp('Calculando distancias del cluster: '), disp(l);
    for t=1:clusters
        dist_entre(l,t)=[sum((VMC(l,:)-VMC(t,:)).^2).^0.5];
    end
end
```

La primera parte de este fragmento de código simplemente se dedica a resetear las variables que vamos a utilizar y a hallar el vector medio de cada cluster ("*VMC*"), el cual es necesario a la hora de calcular las distancias *entre-clusters*. Lo tomaremos como referencia de cada uno de los conjuntos y mediremos distancias en base al mismo.

Para ello, la variable "*a*" es una matriz de una única columna que contiene los recursos que pertenecen a una misma etiqueta. Hacemos un bucle que recorra todos los clusters, y en cada uno de ellos, escogerá el vector que corresponde a sus recursos, haciendo por último, la media de los recursos que pertenecen al mismo cluster, cuyo resultado se guarda en la variable "*VMC"*.

La distancia *intra-cluster* se calcula fijando, de uno en uno, todos los elementos de un mismo cluster, y calculando su distancia al resto mediante la fórmula de la *distancia euclídea* entre dos vectores que matlab nos muestra al introducir en la consola de comandos: *help dist*. La fórmula es:

```
dist_intra(k,p)=[sum((X(a(k,1),:)-X(a(p,1),:)).^2).^0.5];
```

Además de éste dato, también nos interesa saber las distancias máxima, mínima y media entre los recursos de un mismo cluster. Guardamos lo dicho en las variables *maximos_intra*, *minimos_intra* y *medias_intra.*

Finalmente, la distancia *entre-clusters*, se halla de la misma manera que la anterior, con la diferencia que los vectores utilizados son los *VMC* de cada cluster. El resultado que obtendremos será una matriz cuya diagonal es cero, puesto que la distancia de un vector a sí mismo es nula como indicabamos anteriormente. Para ver mejor la diferencia entre las dos funciones que calculan la distancia euclídea, se muestra la utilizada para este último caso:

```
dist_entre(l,t)=[sum((VMC(l,:)-VMC(t,:)).^2).^0.5];
```

2.3 RESULTADOS:

Para concluir este segundo capítulo, se van a mostrar los diferentes experimentos que se han realizado, una vez implementado correctamente el algoritmo del Spectral Clustering en matlab.

En primer lugar, lo que se pretendía, era ver si la asignación de los clusters se realizaba correctamente. Por tanto, se realiza para ello una sencilla prueba que consiste en dibujar una ***nube de puntos aleatorios muy bien diferenciados en dos grupos***, dentro de los ejes de coordenadas *x* e *y*.

El resultado que debemos obtener es tan obvio, que si no fuera así, estaríamos ajustando algún parámetro mal o habría algún error en el código. Sin embargo, lo que obtenemos al realizar tal experimento es la gráfica que se muestra a continuación:

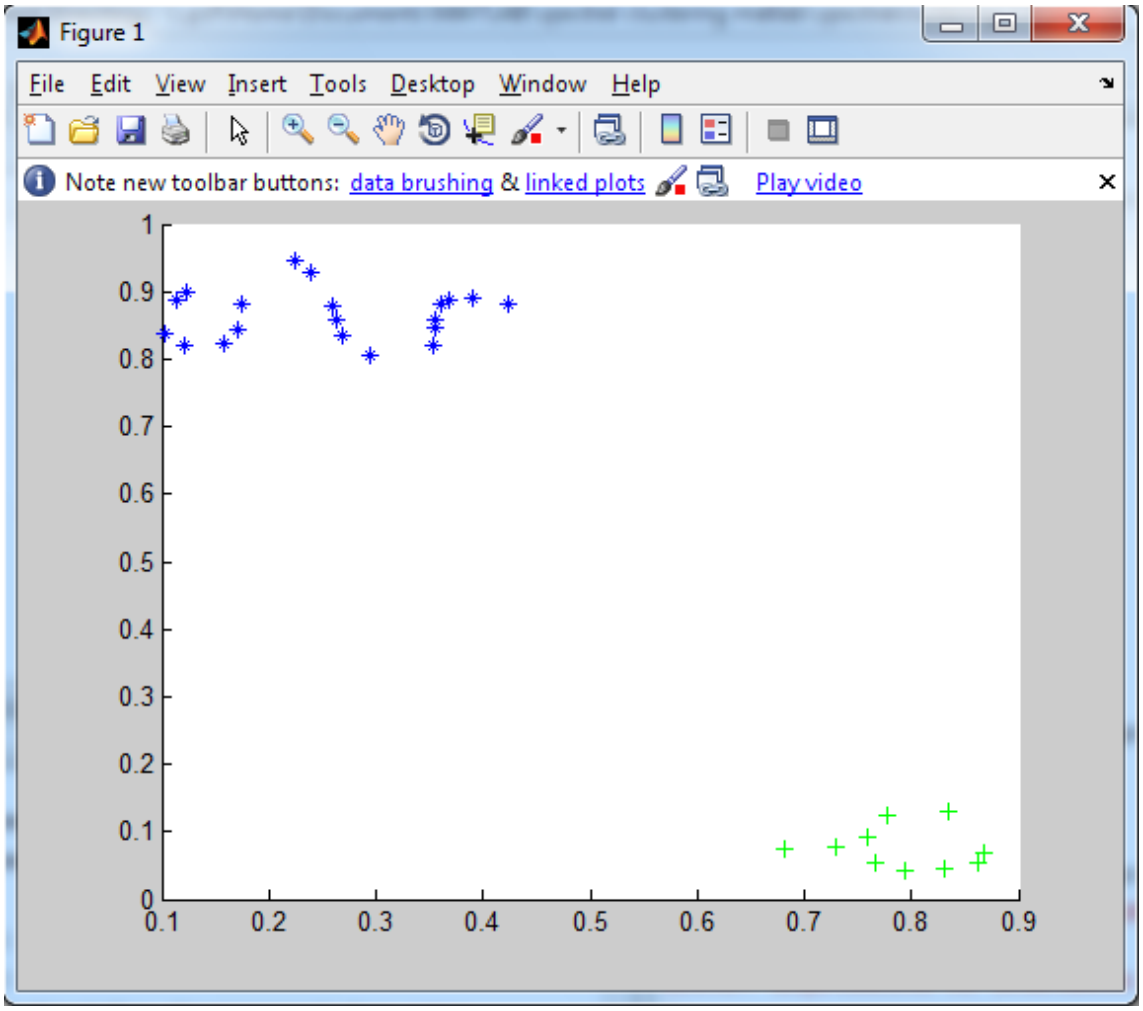


Figura 2.5. Nube de puntos aleatorios.

Los pasos para obtener la gráfica son muy sencillos:

1. **[x,y]=ginput(n);**

   Sirve para poder dibujar con el ratón, una serie de puntos en el plano (x,y). El número de puntos introducidos es la variable *n*.

2. **A=[x,y];**

   La matriz *A* contendrá las coordenadas de los puntos introducidos.

3. **B=A;**

   Simplemente se copian los datos de *A* en *B* porque la matriz *A* va a ser modificada.

4. **scatter(x,y,'*');**

   Es un paso auxiliar, para ver los puntos dibujados anteriormente.

5. **gen_nn_distance(A,5,1,0);**

   Como hemos explicado en el apartado anterior de este capítulo, genera la matriz dispersa de distancias. El *número de vecinos es 5*.

6. **load('5_NN_sym_distance.mat');**

   Cargamos el fichero de salida del apartado anterior.

7. **[predict_labels] = sc(A,0,2);**

   Aplicamos la función Spectral Clustering a la matriz *A*, con la técnica *Self-Tunning*, y *2 clusters*.

8. **grupo1=find(predict_labels(:,1)==1);**
   grupo2=find(predict_labels(:,1)==2);

   Guardamos en las variables grupo1 y grupo2, los recursos pertenecientes a los mismos para colorearlos posteriormente.

9. **scatter((B(grupo1,1)),(B(grupo1,2)),'b*') hold on**
   scatter((B(grupo2,1)),(B(grupo2,2)),'g+');

   Dibuja en la gráfica los puntos pertenecientes al grupo1 de color azul y con forma de '*', y de color verde y con forma de '+' los del grupo2.

Otra prueba, se realizó de la misma manera, solamente que se asignan unos puntos en el plano que determinan unas ***letras***, concretamente la *"O", "A" y "L"*. Solo que ahora tenemos la pequeña dificultad de que hay puntos que están muy próximos pese a pertenecer a diferentes letras.

Por tanto, nuestro objetivo ahora mismo consiste en variar el número de "vecinos" en la función *gen_nn_distance* para ver el efecto que tiene sobre los datos introducidos.

El resultado obtenido al aplicar los pasos anteriores en este caso, nos devuelven las siguientes gráficas:

Resultado para 5 vecinos:

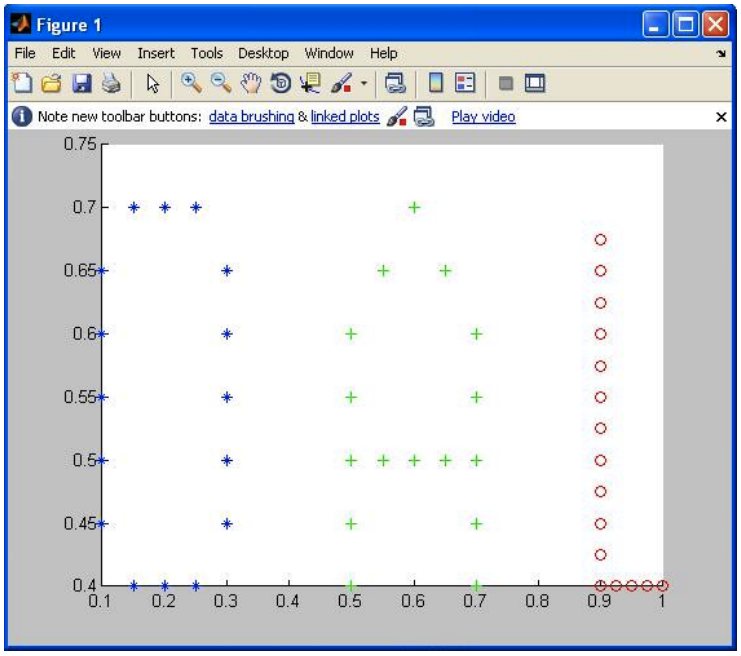


Figura 2.6. Resultado para 5 vecinos.

Resultado para 15 vecinos:

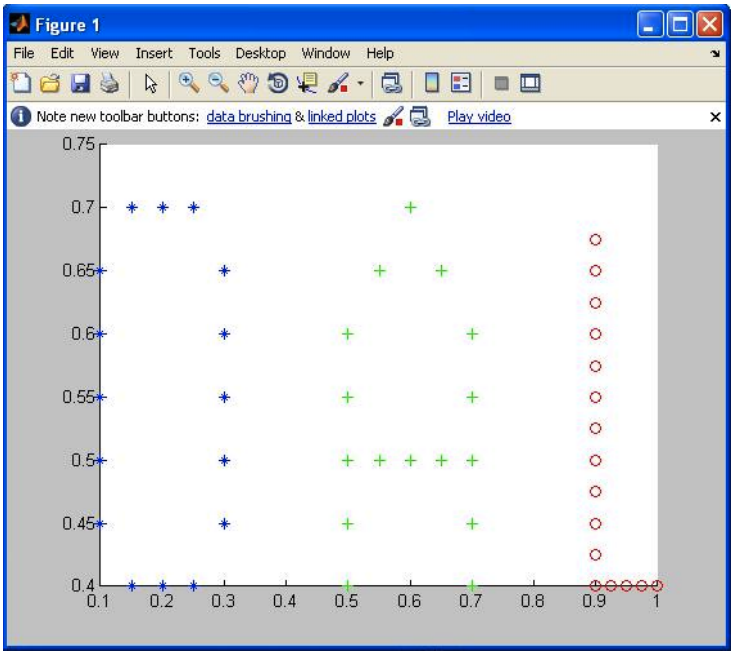


Figura 2.7. Resultado para 15 vecinos.

La dificultad que se nos presentaba en ese aspecto es mínima, ya que pese a haber menos distancia que en el ejemplo anterior entre puntos de diferentes clusters, se observa que es aún y todo ésta es considerable entre las distintas letras, lo que nos devuelve el mismo resultado en ambas gráficas.

Ahora si, procedemos a llevar a cabo el análisis sobre nuestra matriz de datos. Como se ha comentado en el punto anterior de este capítulo, para agilizar el experimento, trabajamos con 200 filas de dicha matriz.

En principio, fijamos un número de clusters igual a 50. De esta manera, veremos que agrupación opta por tomar nuestro algoritmo respecto a los recursos introducidos. Sin embargo, para contrastar los resultados, variaremos dicha cantidad desde los 10 hasta los 100 clusters por ejemplo.

Podríamos poner pantallazos de todos los resultados obtenidos, pero las diferencias se observan mejor en las gráficas que se van a mostrar a continuación.

Esta tabla muestra la ***relación Distancia-NúmeroClusters***. La primera columna contiene el número de clusters seleccionados. Posteriormente, tenemos la distancia media, máxima y mínima en las columnas segunda, tercera y cuarta respectivamente.

10	5.260.645	2,16E+07	24.495
30	4.067.350	2,03E+07	0
50	3.605.101	2,00E+07	0
70	2.867.062	1,99E+07	0
100	1.833.537	1,99E+07	0

La gráfica correspondiente a la tabla es la siguiente:

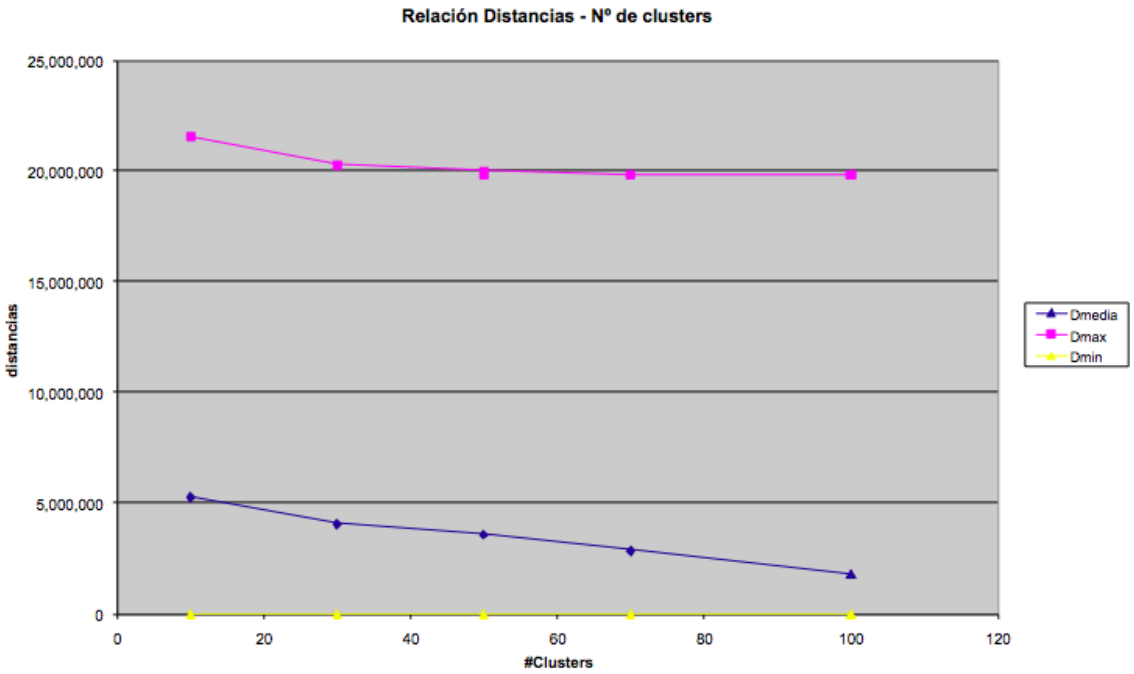


Figura 2.8. Relación Distancia-Nº de clusters.

Como conclusión a los datos obtenidos podemos decir que, tanto la distancia mínima como la máxima, varían muy poco su valor conforme aumentamos el número de clusters. La máxima es en todos los casos un número elevadísimo, lo que nos hace pensar que pese a que dos recursos pertenezcan a un mismo cluster, pueden ser realmente dos recursos muy diferenciados entre sí, manteniendo alguna característica común, claro. Y en cuanto a las mínimas, destacar que en todos los casos excepto el de 10 clusters es cero, de lo que se deduce la lógica de que elementos de un mismo cluster tienen situaciones muy similares.

En cuanto a la distancia media, disminuye conforme aumentamos el número de clusters. Es algo evidente, ya que conforme tenemos más número de clusters, habrá menor distancia entre ellos, lo que implica una mejor agrupación, siempre y cuando trabajemos con los mismos datos.

Vamos a hacer una comparación similar con las **Distancias entre-clusters**. Se han obtenido distancias para los casos de10, 30, 50, 70 y 100 clusters.

10 clusters:

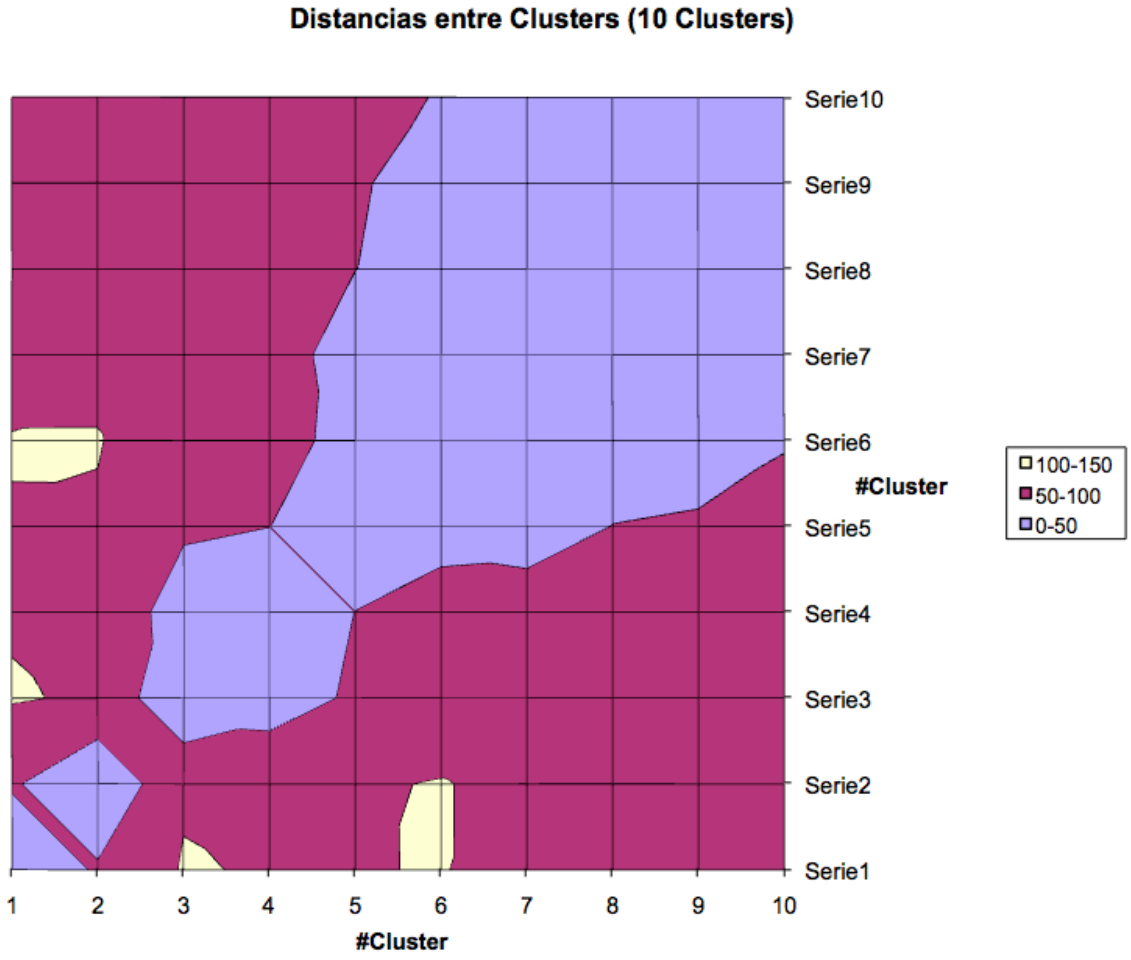


Figura 2.9. Distancia entre clusters (10 Clusters).

Como se puede ver, la franja azul es una distancia mínima, entre 0 y 50, y el hecho de que se sitúe en la diagonal implica que la distancia entre un cluster y sí mismo será 0. Solo se aprecian grandes distancias entre los clusters 2 y 6, así como entre 1 y 3.

30 clusters:

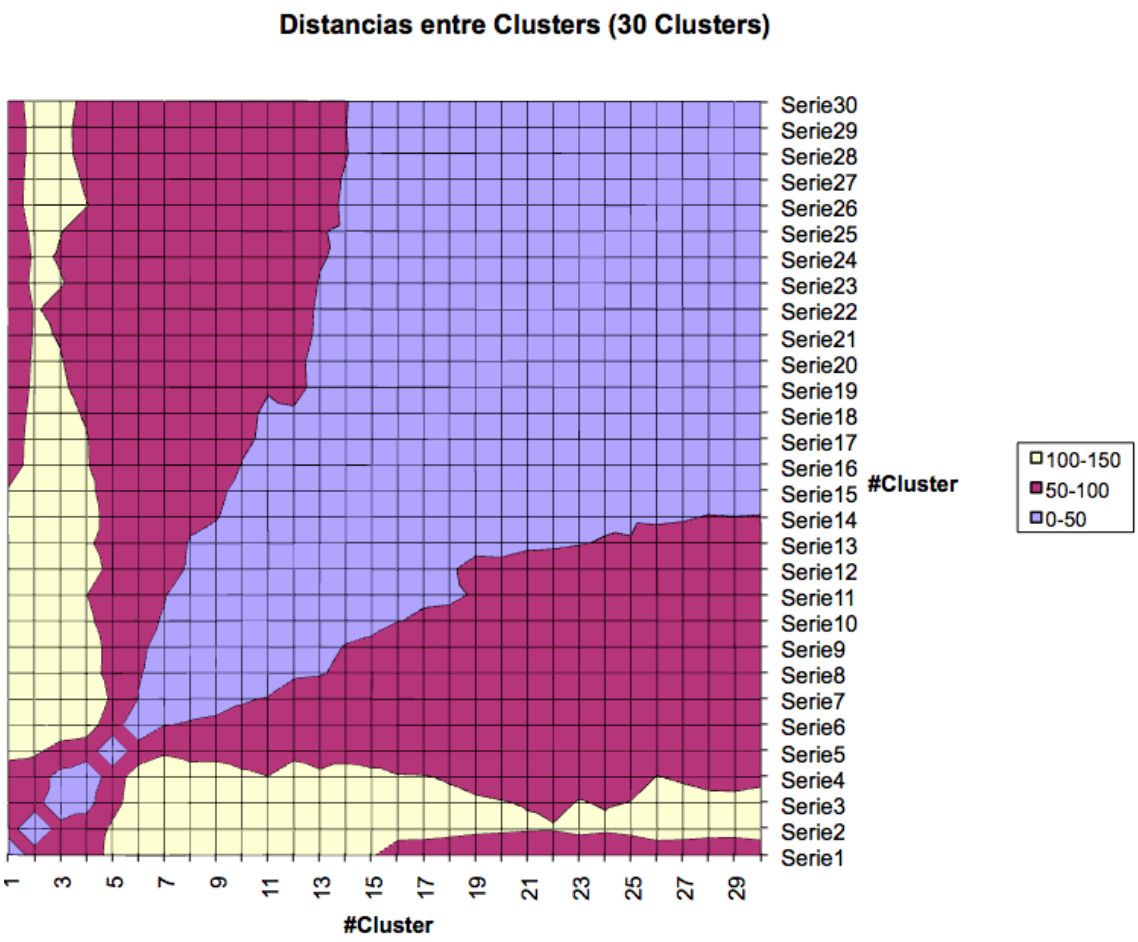


Figura 2.10. Distancia entre clusters (30 Clusters).

La principal diferencia con respecto a la gráfica anterior es que aumenta el número de clusters con una distancia mayor entre ellos, puesto que si hay diez, es lógico que la distancia entre los que más lejos se sitúen será mucho inferior a que si hubiera treinta clusters, como es el caso. Por lo demás, la situación es parecida.

50 clusters:

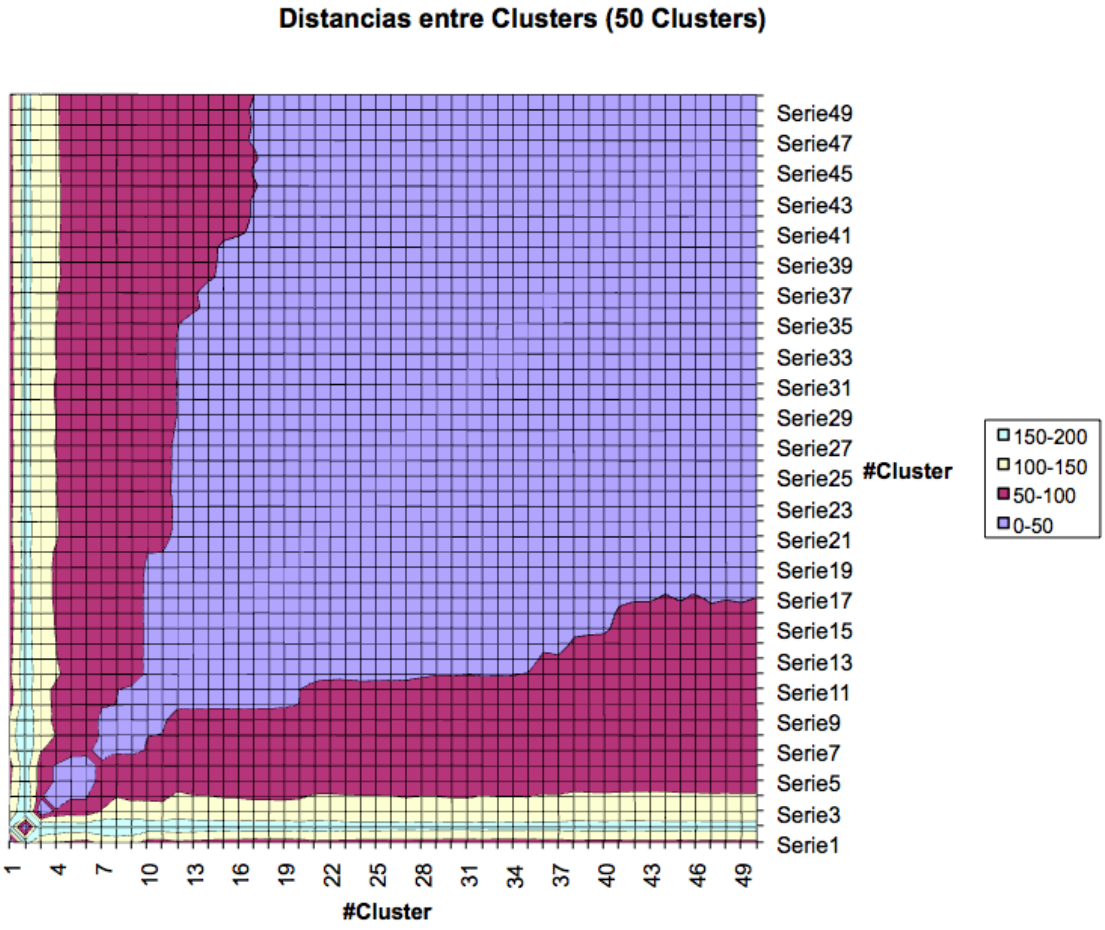


Figura 2.11. Distancia entre clusters (50 Clusters).

Ahora la novedad es que se añade un nuevo color a la gráfica, ya que como se ha dicho antes, aumentamos más el número de clusters y la distancia máxima, aumenta respectivamente. Ante tal necesidad, aparece el color verde. Sobre todo se sitúa en la zona inferior (como es simétrica, también en el lateral izquierdo). También parece a simple vista que la zona "rosa" cada vez va disminuyendo un poco más.

70 clusters:

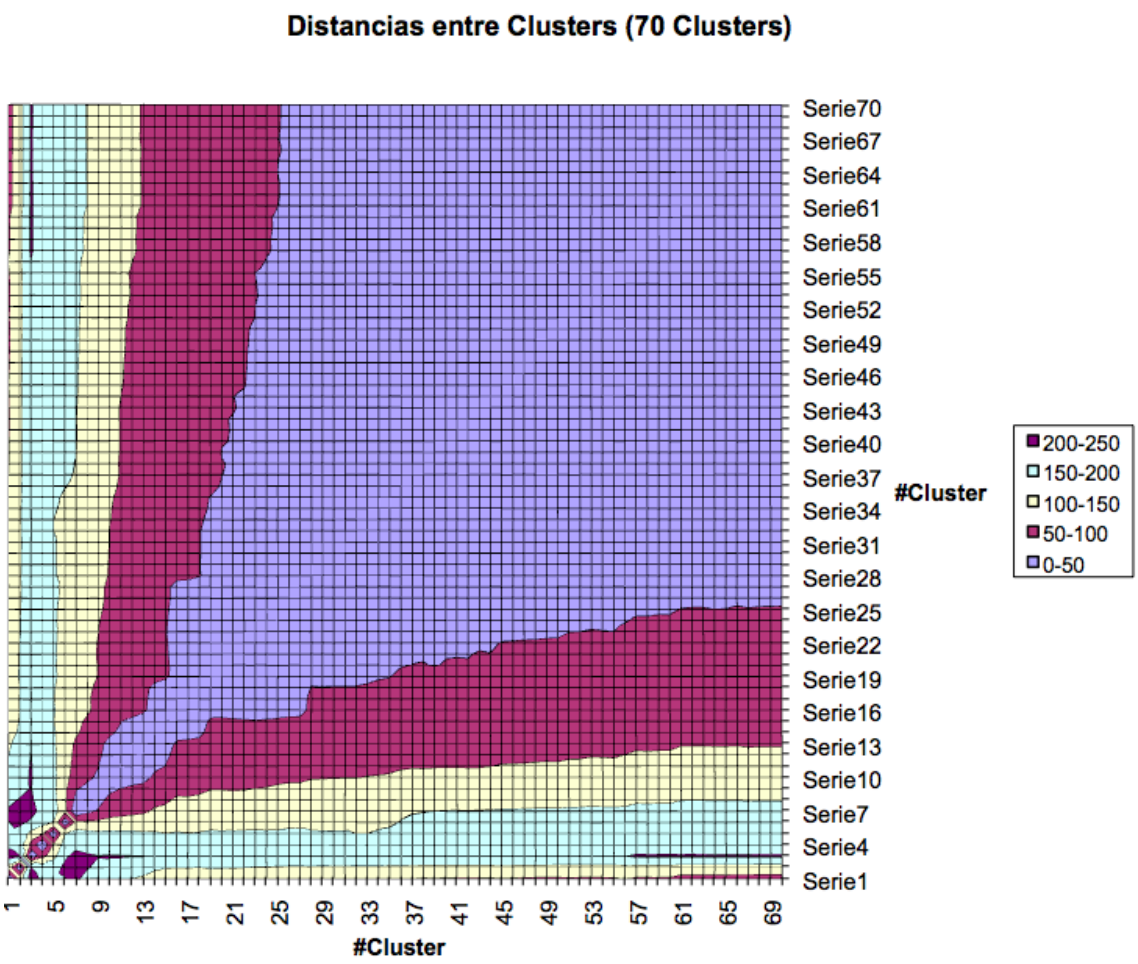


Figura 2.12. Distancia entre clusters (70 Clusters).

Se podría comentar exactamente lo mismo que en el gráfico anterior. Aparece un nuevo color para designar una distancia todavía mayor entre clusters. Además, cada vez se va notando, respecto a las gráficas del principio, que los diversos colores abarcan más o menos el mismo campo, a diferencia de lo que sucedía por ejemplo cuando teníamos 10 clusters, donde predominaba con total claridad el azul y el rosa, con a penas amarillo.

100 clusters:

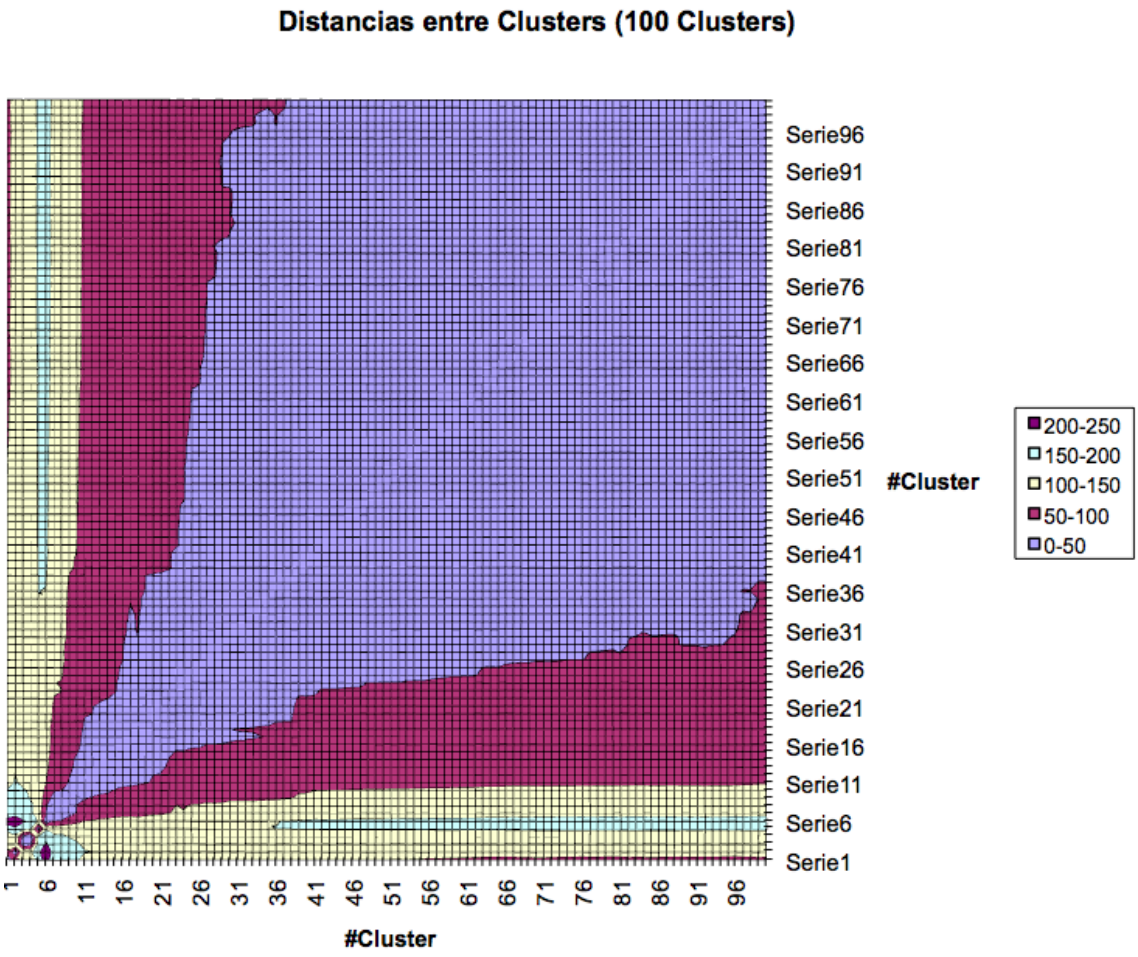


Figura 2.13. Distancia entre clusters (100 Clusters).

Pese a que pueda parecer que existe una gran diferencia con el caso anterior, es engañoso el tamaño de la página a mi parecer. Sigue habiendo más de lo mismo, con la única diferencia de que parece que el color amarillo le ha ganado terreno al verde, que parece haberse desplazado hacia el origen.

REFERENCIAS:

[OSC01] Andrew Y. Ng, Michael I. Jordan, Yair Weiss (2001). On Spectral Clustering: Analysis and an algorithm. NIPS 2001 Conference.
http://ai.stanford.edu/~ang/papers/nips01-spectral.pdf

[STSC04] Pietro Perona, Lihi Zelnik-Manor (2004). Self Tuning Spectral Clustering. Department of Electrical Engineering (California Institute of Technology).
http://www.vision.caltech.edu/lihi/Demos/SelfTuningClustering.html

CAPÍTULO 3: ANALIZADOR LÉXICO:

Para comprobar la similitud existente entre los diferentes recursos pertenecientes a un mismo cluster, utilizaremos una aplicación muy interesante: se trata de un analizador semántico [AML06] que sea capaz de medir la similaridad semántica entre distintos documentos (en nuestro caso cada documento se corresponderá con el contenido de un único recurso), con el objetivo de extraer el mayor conocimiento posible de modo rápido y eficiente.

A continuación se expresan con detalle aspectos sobre el problema con el que nos encontramos, objetivos que se pretenden cumplir y lo más importante: la solución al problema que nos incumbe.

3.1 DESCRIPCIÓN:

Hoy en día Internet es uno de los mayores repositorios de información y cada día crece más y más. Este gran volumen impide que seamos capaces de procesar y manejar con eficiencia la información. De este modo podemos estar trabajando con información redundante, de modo que varias fuentes tratan el mismo contenido y no somos capaces de darnos cuenta de esta situación.

El hecho de que la información que se nos ofrece en las páginas Web esté expresada de distinto modo o que estos clasifiquen el contenido mediante un esquema diferente puede provocar que se pierda información relevante.

Estas situaciones provocan la imposibilidad de realizar un análisis detallado y profundo de la información con la que trabajamos. Y por lo tanto, el conocimiento del tema que se adquiere no es completo.

Para evitar este problema deseamos una herramienta automática, fácil de usar y rápida que nos permita comparar el contenido de los documentos. De este modo el usuario será capaz de descartar documentos ya que el contenido semántico es similar. Para ello se necesita que los resultados obtenidos por el usuario sean inteligibles y manejables.

Así pues, utilizaremos una herramienta software que permita al usuario comparar documentos de texto planos respecto a la semejanza semántica entre ellos. Documentos que se almacenarán en una dirección conocida, o en una base de datos *Semantica* a modo de reglas semánticas, ya que se les ha aplicado algún tipo de filtro.

La comparativa se realizará mediante medidas de similaridad y se presentarán los resultados de forma que el usuario sea capaz de entenderlos de manera fácil y rápida.

Por tanto, esta aplicación nos va a permitir ver si realmente existe una relación vinculante entre dos documentos diferentes, que, como se ha comentado en párrafos anteriores, se corresponden con dos recursos pertenecientes a un mismo cluster.

Aparentemente, nuestro algoritmo ha sido capaz de agruparnos los recursos de la matriz de entrada en un conjunto de clusters, pero hay que comprobar su similitud para obtener conclusiones acerca del agrupamiento realizado.

Como veremos en el próximo capítulo, podremos realizar varios tipos de análisis, según nuestras necesidades de comparación.

3.2 MÉTRICAS A EVALUAR:

Este punto del tercer capítulo tiene como pretensión mostrarnos los diferentes análisis que podemos realizar para obtener los resultados en forma de gráficos de barras, sectores, … y poder comparar los resultados obtenidos con el juicio que haríamos nosotros, en cuanto a la similitud de dos recursos.

En la siguiente sección mostraremos los resultados obtenidos a la hora de aplicarlos sobre los clusters que habremos obtenido previamente al ejecutar nuestro algoritmo.

En cualquier caso, los *tipos de métricas* que podremos evaluar son: ***análisis términos-párrafo***, ***análisis términos-documentos***, ***análisis de términos*** y ***análisis de documentos***.

El análisis de ***términos-párrafos*** mide la importancia de cada uno de los términos que aparecen en el documento, en cada uno de los párrafos.

Por otro lado, en el análisis ***términos-documentos***, lo que comprobaremos será la relevancia de los términos en distintos documentos.

En cuanto al análisis ***de términos***, simplemente decir que el objetivo de este tipo de análisis es comprobar como de relacionados están unos términos con otros en el contexto de un documento. Mediante el mismo, podría determinarse si dos términos pueden tratarse como sinónimos en el contexto o no.

Finalmente, en el análisis **de documentos**, obtendremos la similitud de dos documentos, es decir, si están relacionados entre sí o no.

En nuestras pruebas éste último será el más determinante, ya que compararemos diferentes documentos que se corresponden con el contenido de recursos (páginas web) y nos interesa saber si realmente existe relación o no, verificando de esta forma la correcta inclusión de elementos de un mismo cluster.

Ahora, una vez elegido el tipo de métrica que queramos evaluar, debemos escoger entre varios *modelos de análisis*.

En las medidas que utilizamos para el análisis, se eliminarán los signos de puntuación y el conjunto de palabras definidas como "*stopword*", que son aquellas que no muestran información sobre la semántica del texto. La recopilación se representa como una matriz que variará dependiendo del tipo de modelo a utilizar.

La matriz relaciona los documentos respecto al tópico de cada uno de ellos, de manera que la relación será más fuerte en el caso de que coincidan en el tópico.

El primer modelo de análisis que nos encontramos es el ***Binario***. Este modelo, asume que la similaridad es independiente del número de ocurrencias de los términos que aparecen en los documentos. A la hora de representar los objetos sólo se tiene en cuenta que el término aparece.

En los modelos binarios los términos se representan como vectores, donde la dimensión del vector se corresponde con el número de unidades en que se divide la recopilación de texto que se pretende analizar.

Por tanto podemos definir la recopilación como una matriz multidimensional, de tantas filas como términos distintos aparezcan en la recopilación y tantas columnas como unidades de texto.

La matriz sólo contendrá valores nulos o la unidad. En caso de que el término aparezca en una unidad de texto determinada la celda que está en la fila del término indicado y en la columna de la unidad de texto tendrá valor 1. En caso contrario el valor será 0.

También tenemos el modelo de ***Ocurrencias***. Éste tiene como base dos proposiciones [Lee05] [Ror99]. La primera es que, los documentos relevantes son más parecidos uno con el otro que los que no lo son en los tokens que utiliza. Y la segunda, que los documentos no relevantes se parecen menos y la composición de los términos difiere con los relevantes.

La similaridad mediante estos métodos se mide como el grado en que los documentos comparten los *tokens*, produciendo mayor similaridad cuando el número de *tokens* que aparece en ambos documentos es mayor.

El siguiente modelo es: ***LSA (Latent Semantic Analysis)*** [Lan98] [Fol98]. Es una teoría y un método para extraer y representar de forma automática la similaridad en el contexto de una gran recopilación de textos. Se ha demostrado cualitativamente (e incluso cuantitativamente en ocasiones) que este método es capaz de imitar el funcionamiento de los humanos. Mediante LSA se han logrado grandes resultados en la recuperación de información, siendo importante la capacidad para encontrar sinónimos.

Podemos pensar que LSA representa el significado de una palabra como clase promedio del significado de todas las unidades de texto en las que aparece. El significado de una unidad de texto será el promedio del significado de las clases de las palabras que contiene la unidad.

La capacidad de LSA para representar la similaridad además de depender de la correlación de los dos tipos de significados mencionados, puede variar en función de la dimensionalidad utilizada para representar los valores. Está asumido que la reducción de la dimensionalidad (número de parámetros por los que se describen los términos o documentos) a un número mucho menor cuando ésta es grande puede derivar mejores resultados que la inicial.

Sin embargo, uno de los grandes problemas de este método es encontrar una dimensionalidad óptima, ya que para cada recopilación de texto esta varía.

Por último, el conocido ***Coeficiente de Correlación de Spearman*** [Spe06]. El coeficiente de correlación, R, es un valor que mide el grado de la relación entre dos variables. Este es una técnica no paramétrica del coeficiente de correlación de Pearson (mide la asociación lineal entre dos variables) para medir la fuerza de una relación por pares de 2 variables cuando los datos se encuentran de manera jerarquizada. Este

coeficiente resulta apropiado cuando los datos pueden ser ordenados y para datos que pueden ser agrupados en intervalos que no satisfagan el supuesto de normalidad, es decir, no siguen una distribución normal sino una distribución libre.

El objetivo del calcular el coeficiente consiste en determinar el grado en que dos conjuntos concuerdan o no. Esta técnica se podrá utilizar siempre que los datos se puedan convertir en rangos. Por tanto, elegiremos esta técnica si los datos no están normalmente distribuidos o tienen categorías ordenadas, ya que mide la asociación entre órdenes de rangos.

El valor del grado tiene las siguientes características:

- El valor de R varía entre -1 y 1. El signo indica la dirección de la relación, mientras que el valor absoluto del coeficiente nos informa de la fuerza de la relación. Cuanto más próximo esté el valor de 1, la relación será más fuerte.
- Si las variables son totalmente independientes R=0.
- Si las variables están relacionadas linealmente R=1.

Definimos el grado de relación de las variables como:

- Re1ación Perfecta: R = 1
- Re1ación Excelente: R = 0.9 < = R < 1
- Re1ación Buena: R = 0.8 < = R < 0.9
- Re1ación Regular: R = 0.5 < = R < 0.8
- Relación Mala: R < 0.5

3.3 RESULTADOS:

Nuestro objetivo en este último apartado del tercer capítulo consiste en, una vez ejecutado el algoritmo de spectral clustering (ver Capítulo 2), analizar los clusters obtenidos de manera que, recursos de un mismo cluster, tengan un vínculo que los relacione.

Así pues, comenzamos diciendo que vamos a realizar las pruebas oportunas con 200 filas de nuestra matriz de recursos por el hecho de simplificar el asunto.

Nuestra matriz se encuentra codificada en un fichero llamado "*acoar-method-testclustering.sql*", por lo que, para ver su contenido y poder observar qué páginas contiene cada recurso, hacemos uso de la aplicación "*XAMPP*": servidor apache que contiene MySQL, PHP y Perl. Una vez instalado ya podemos acceder a los datos del fichero anterior.

En la siguiente tabla resumen se muestran algunos de los resultados obtenidos a la hora de ejecutar al algoritmo. Tenemos el número de cluster, recursos pertenecientes al mismo y distancias máximas, mínimas y media intra-cluster.

CLUSTER	RECURSOS	DIST.MÁXIMA	DIST.MÍNIMA	DIST.MEDIA
1	111, 124, 127.	256,1484	189	144,6160
2	32, 122, 170, 194.	1,0938E+03	151,2052	532,2176
47	81, 88, 103.	5,1962	2,4495	2,8537
48	27, 144.	226,6627	226,6627	113,3314
50	110.	0	0	0

El contenido de las páginas web (recursos) debe pasarse a un fichero con formato ".txt" para que el analizador léxico pueda realizar las comparaciones correctamente. Podemos hacer esto mediante una araña (implementándola nosotros mismos o descargando una en la web) o mediante aplicaciones como "*Html2Text*" que nos ahorrarán tiempo de trabajo. En cualquier caso, ambas opciones son válidas.

Una vez visto el planteamiento inicial, aplicamos las métricas convenientes para poder ver si las relaciones entre los elementos del mismo cluster tienen sentido. No olvidemos que, al escoger únicamente 200 filas de nuestra matriz (12106x1939) y hacer 50 clusters de entre dichos recursos, los resultados obtenidos no pueden ser evidentes a primera vista, pero la relación existente es corroborada por nuestro analizador.

En primer lugar, tenemos el *Cluster 1*. En él se encuentran recogidos los recursos 111, 124 y 127 de la matriz, los cuales se corresponden con las siguientes páginas web respectivamente:

- http://pycallgraph.slowchop.com
- http://www.smashingmagazine.com

- http://www.htdp.org

La primera de ellas es la URL de "Python Call Graph", la cual es un módulo de Python que crea/*diseña* "call graphs" (grafos que representan las llamadas de unas subrutinas a otras en un programa) para programas en dicho lenguaje.

Smashing Magazine es un famoso blog que recopila tutoriales y recursos muy útiles para *diseñadores*.

HTDP se corresponde con la web "How To Design Programs", traducido al castellano, "Como *diseñar* programas".

Como se ha marcado en letra cursiva, estos recursos se dedican al *diseño*, pese a que no sean exactamente en los mismos campos. Así pues, puede considerarse una relación totalmente correcta la que ha realizado nuestro algoritmo. Sin embargo, es útil poder ver con una gráfica comparativa, realizada pos nuestro analizador, los términos que más se repiten en ambas páginas.

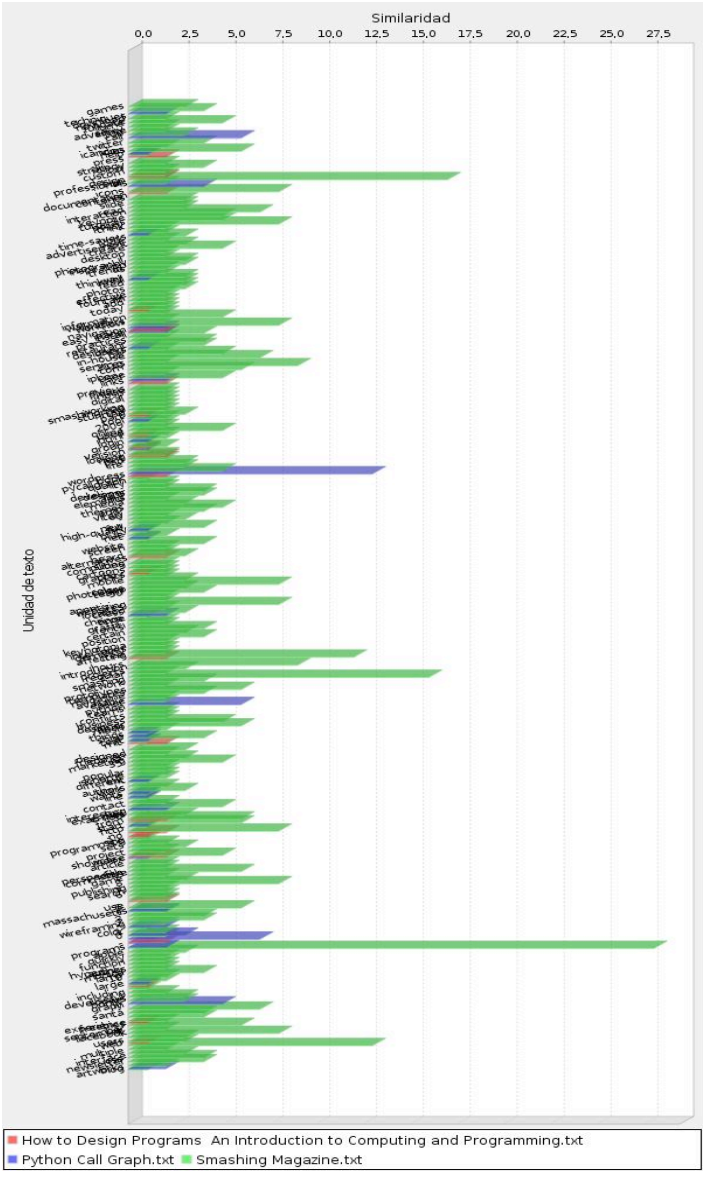


Figura 3.1. Analizador (Cluster 1).

Se puede observar en la gráfica que la mayoría de las barras son de color verde. Esto se debe a que el fichero de texto que contiene la web "Smashing Magazine" es bastante más amplio que los otros dos. Hay gran cantidad de términos que solo pertenecen a uno de los documentos, propios del mismo, así como otros que no tienen significado alguno debido a que tienen que ver con código html u otros (page, html, index, "-", etc).

Lo que nos interesa en este ejemplo es la palabra "*design*". Como bien se ha indicado anteriormente, al ser tres páginas que tienen que ver con el diseño, éste término debería aparecer en ambas. Pues bien, nos encontramos con su aparición 2, 0 y 17 veces respectivamente. Esto indica que están relacionados por el mismo, pero la url de Python Call Graph no está íntimamente relacionada con el diseño de programas en sí, sino que más bien con la creación de esos grafos (de ahí la aparición nula de dicho término).

Lo comentado en el anterior párrafo también tiene que ver con el hecho de que la distancia intra-cluster es algo más de 256, lo cual no es mucho, pero dista mucho de la que se da en el cluster 47 por ejemplo (luego se desarrollará el mismo).

El *Cluster 2* está formado por los recursos 32, 122, 170 y 194. Éstos se corresponden con las URL´s:

- http://www.jetbrains.com/resharper/
- http://www.niallkennedy.com/blog/2009/04/facebook-haystack.html
- http://webdesignledger.com/freebies/70-beautiful-damask-patterns-and-textures
- http://reubenmiller.typepad.com/my_weblog/2008/09/21-unexpected-a.html

Jetbrains se dedica al desarrollo de herramientas inteligentes para mejorar la productividad de programadores de aplicaciones. En concreto, la sección "resharper" contiene una herramienta para visual Studio, la cual puede ser descargada.

En cuanto a Niall Kennedy, se corresponde con su blog, el cual nos habla de la reescritura del almacen de fotos de Facebook. Su página nos habla del tema y posteriormente hay una serie de comentarios de gente que lee su blog.

Web Design Ledger tiene una *estructura muy similar* a la página anterior, pero nos ofrece la descarga de una serie de fondos de escritorio y accesorios de Damasco.

Por último, Reuben Miller, es un blog que nos muestra, con *idéntico estilo* a las otras dos páginas anteriores, los 22 alfabetos más creativos del mundo, pudiéndolos descargar igualmente.

En los párrafos anteriores se ha remarcado que la similitud existente entre estas páginas es *su formato*, casi igual en todas ellas, siendo la que más se diferencia del resto la primera. En cuanto al tema a tratar, no tienen mucha relación aparentemente, salvo que se trata de *herramientas de diseño o algo parecido*. Por tanto, en el análisis semántico pondremos especial atención a palabras clave en cuanto a la estructura o herramientas.

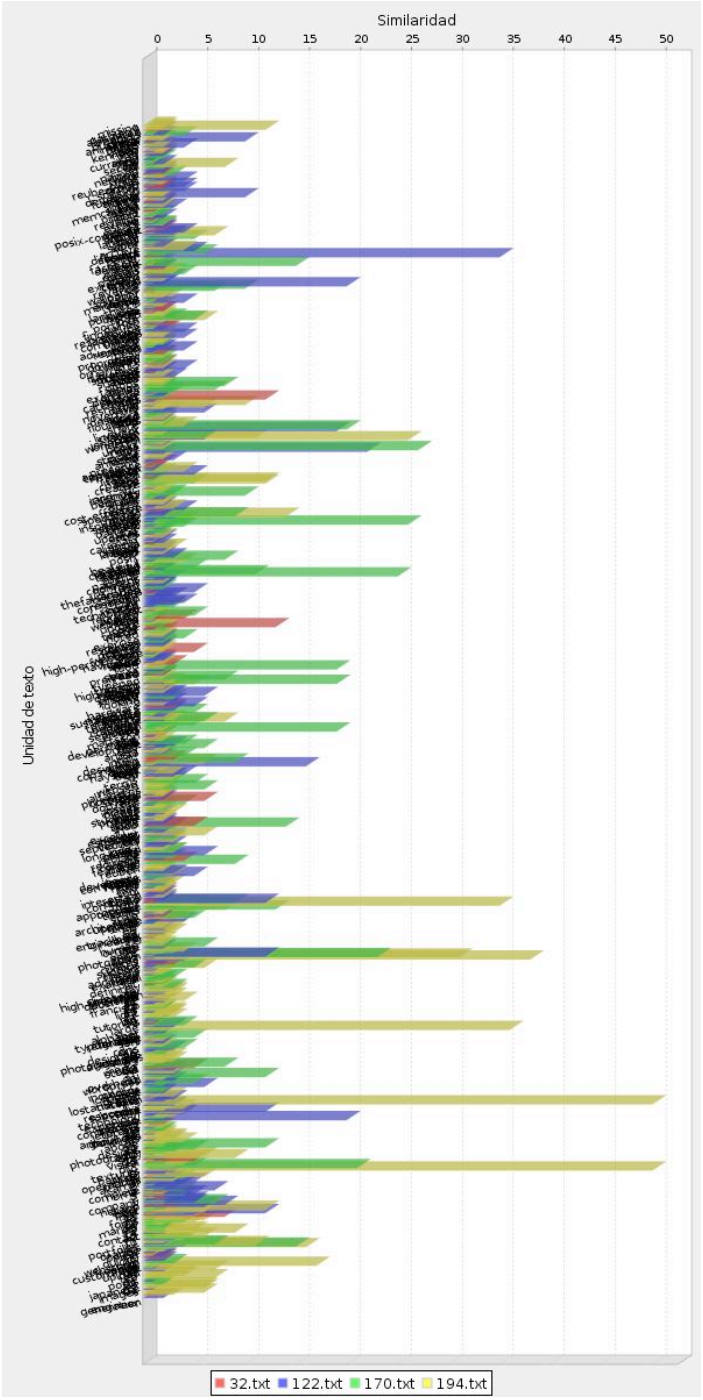


Figura 3.2. Analizador (Cluster 2).

En el dibujo, pese a no apreciarse bien por la gran cantidad de términos, nos encontramos con que, al igual que en el caso anterior y se verá también en el resto, existen términos sin importancia alguna. Sin embargo, términos como "*Tools*" y "*Design*" referidos a herramientas y diseño cobran importancia respecto al resto, ya que aparecen en todas las webs pertenecientes al cluster. También hay palabras que, por su relación en cuanto al formato de la página se repiten. Es el caso de: "*Comments*" (por los comentarios de los usuarios), "*2010*", "*2009*", etc (por el año del comentario en los

blogs), *números del 0 al 9* (por la puntuación dada a la herramienta), "*Search*" (por los motores de búsqueda), etc.

Pasamos a tratar el *cluster 47*, formado por los recursos 81, 88 y 103. Sus páginas son las que siguen:

- http://code.google.com/intl/es-ES/apis/maps/documentation/staticmaps/
- http://www.downloadyoutubevideos.com/
- http://www.w3.org/Amaya/

La primera de las URL se corresponde con el api o manual para *insertar* mapas de Google maps en *nuestras páginas web*.

Downloadyoutubevideos es una web que nos ofrece la posibilidad de poder descargar vídeos de youtube. Podemos darle diversidad de usos a estos vídeos, uno de ellos puede ser, como en el caso anterior, para *insertarlos* en páginas, por ejemplo.

Por último, la web de w3c es una comunidad internacional donde las organizaciones miembro, personal a tiempo completo y el público en general trabajan conjuntamente para desarrollar estándares Web. En concreto la sección "Amaya" se corresponde con un *editor web* de código abierto.

Cabe comentar que nos encontramos con un caso a analizar bastante curioso, ya que las distancias que obtenemos intra-cluster son significativamente más pequeñas que en el resto, lo que a priori nos hace pensar que los recursos tienen que estar íntimamente relacionados entre sí. Las premisas se cumplen ya que todos ellos tienen que ver con el diseño web, pese a que la descarga de vídeos de youtube puede tener muchas otras utilidades.

Por otro lado, tenemos un pequeño problema. La comparación mediante el analizador semántico podría ser una buena manera de corroborar que este cluster está perfectamente formado, pero una el primero de los recursos se corresponde con una página en castellano y los otros dos recursos con páginas en inglés. Esto supone que, a la hora de que nuestra aplicación realice dicha comparación de los términos, en mucho casos estos no van a coincidir porque están en diferentes idiomas.

Sin embargo, tenemos una serie de términos que relacionan sobre todo las dos páginas inglesas. La palabra "*download*" aparece numerosas veces en el recurso de las descargas de vídeos de youtube (nueve veces), al igual que en la página del editor de código abierto Amaya (3 veces). Esto significa que los recursos poseen elementos de descarga, al igual que la api de Google maps, pero nunca coincidirán términos en diferentes idiomas. Lo mismo pasa con términos tales como "*user*", "*software*", "*copyright*", etc.

De todos modos, a continuación se muestra la gráfica que representa la similitud semántica, que pese a no darnos la exactitud que esperábamos por el citado problema, nos asemeja, como se ha comentado antes, dos de los tres recursos de una manera bastante clara.

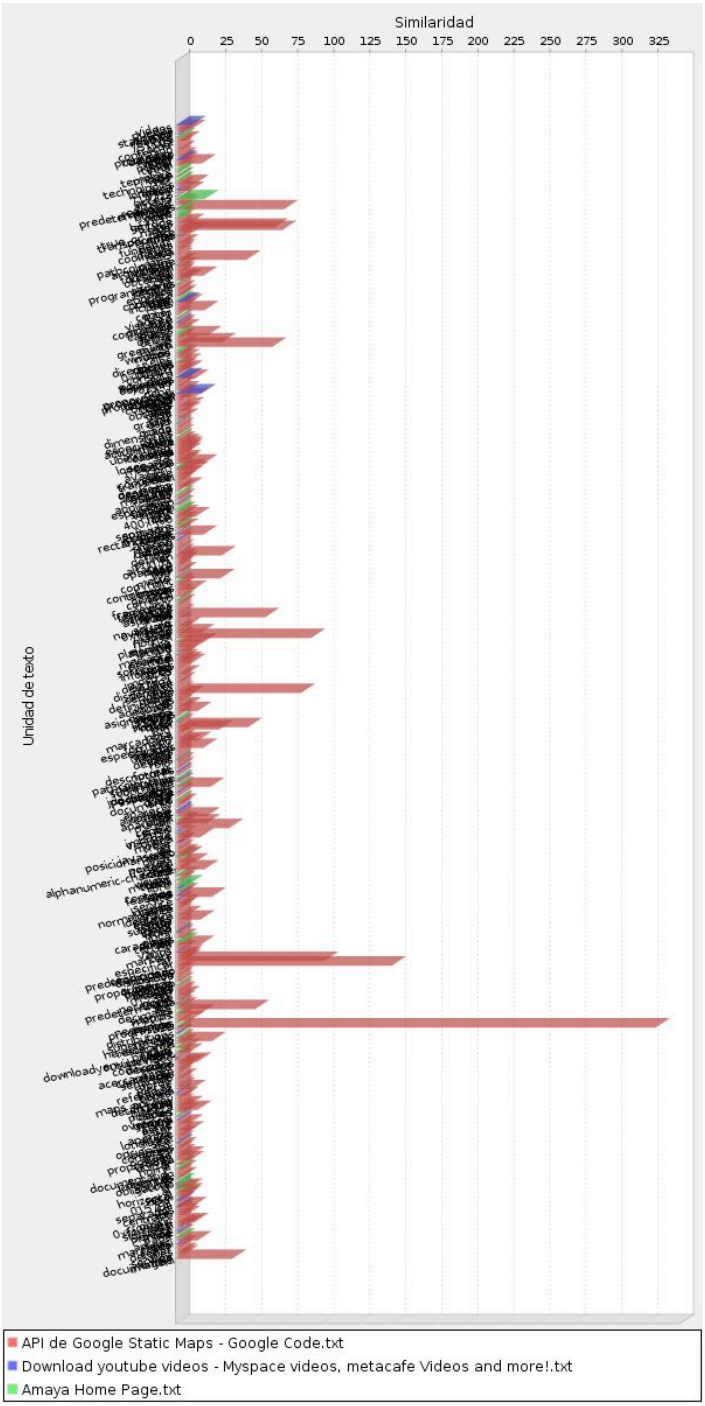


Figura 3.3. Analizador (Cluster 47).

El siguiente cluster que vamos a tratar es el *Cluster 48*. Este cuenta con únicamente dos recursos: el 27 y el 144. Sus páginas respectivas son las que se muestran:

- http://stclairsoft.com/index.html
- http://www.tigerdirect.ca/indexca.asp?

Ambas páginas web tratan sobre *material de informática*, de computadores, pero la primera contiene descargas de *elementos software* gratuitos, a diferencia de la segunda, que es una tienda en Internet de material, tanto hardware como *software*.

Es razonable, a simple vista, que la relación existe en que ambos recursos aglomeran elementos software. Por tanto, nuestro cometido es averiguar que términos se encuentran relacionados. También hay que comentar que el aspecto físico de ambas páginas es similar en cuanto a las etiquetas que poseen: contacto, servicios, etc.

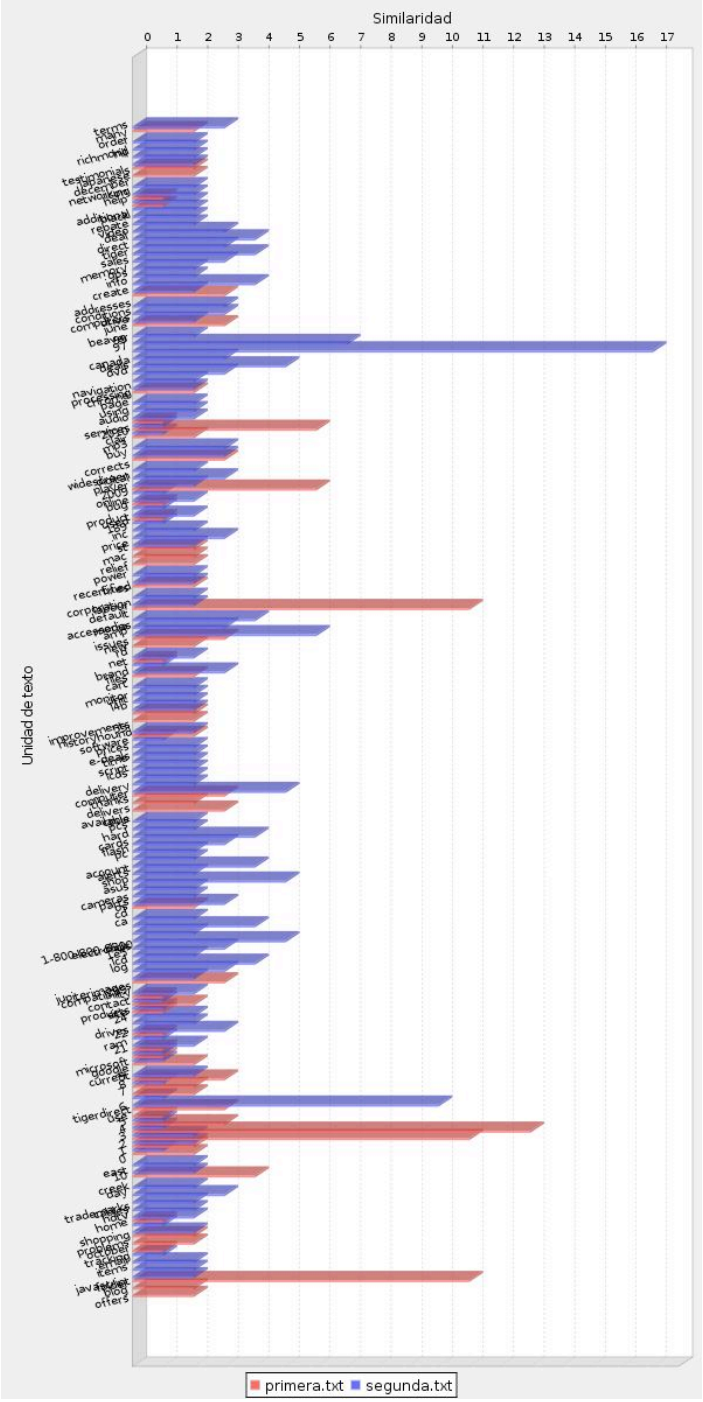


Figura 3.4. Analizador (Cluster 48).

Tras obtener el gráfico del analizador semántico obtenemos unas conclusiones bastante claras. La primera de ellas, se confirma la similitud del contenido de ambos recursos mediante términos como: "*software*" o incluso "*products*" (que también puede incluirse en la segunda reflexión). Ambos aparecen repetidos en las dos webs. Por otra parte, como decíamos antes, la relación que existe físicamente entre ambas se declara en palabras como: "*help*", "*services*", "*contact*", "*home*" y "*products*". Así pues, el análisis demuestra que existe un buen agrupamiento por parte de nuestro algoritmo.

Por último, para acabar con este repositorio de pruebas, decir que existen varios clusters (ver tabla de este mismo apartado) que únicamente contienen un recurso. No tendría sentido valorar su agrupamiento, ya que están alejados del resto y por tanto, los recursos no tienen similitud en cuanto a su contenido.

REFERENCIAS:

[AML06] Amaia Mutuberria Larrayoz (2006). Sistemas de medida de similaridad Semántica entre Documentos. Proyecto Fin de Carrera (I.T.I.G.), UPNA.

CAPÍTULO 4: CONCLUSIONES:

En este capítulo se muestran las conclusiones del trabajo realizado. Estas conclusiones se deberán tener en cuenta a la hora de desarrollar nuevos algoritmos de clustering, así como en los experimentos que se realicen en base al presente.

A la hora de desarrollar el citado proyecto, se han encontrado dificultades sobre todo a la hora de *entender el algoritmo*. Nunca me había topado con algoritmos de clustering, luego comencé por leer información que me supusiera, una vez vista, un esfuerzo menor para entender el algoritmo a implementar. Al principio es complicado, pero una vez que "te pones a ello" se entiende mejor y como consecuencia, se hace más sencillo el trabajo.

En cuanto al hecho de *programar el algoritmo*, no he encontrado tantas dificultades. Simplemente, al comienzo, me familiaricé con matlab y una vez hecho esto no me supuso excesivas complicaciones, siempre dando las gracias a Internet, que nos saca de apuros en muchas ocasiones.

Los *resultados obtenidos* por el algoritmo, a mi parecer, son muy satisfactorios. Al comienzo se ha comprobado (ver *capítulo 2*, sección de *resultados*) que, con ejemplos muy sencillos, el clustering se ajusta a lo esperado, siempre teniendo en cuenta las dificultades en cuanto a la elección de los parámetros que varían estos resultados. Posteriormente, en las pruebas realizadas con la matriz, es más complejo ver a simple vista si los recursos se asocian correctamente en clusters de contenido parecido.

Sin embargo, en este segundo caso, intentamos salir de dudas con el *analizador semántico*, utilizado para ver la correspondencia semántica entre elementos de un mismo cluster. Los resultados, a mi parecer, nos dejan cierta incertidumbre sobre la correspondencia total de los recursos.

Dicha incertidumbre viene dada, en muchos casos, por el hecho de que los términos de páginas en diferentes idiomas no van a coincidir, por tanto el resultado obtenido en cuanto a repetición de términos semejantes se aleja mucho de la realidad.

Pero, por otro lado, el juicio humano si que nos hace optar por un resultado positivo en cuanto a su agrupación, puesto que los recursos agrupados, en todos los casos, coinciden de algún modo, ya sea por su contenido, como por su formato de la página web que representa (recordemos, que los ficheros de texto que contienen los recursos de la matriz, guardan toda su información, ya sea contenido como etiquetas de la web).

CAPÍTULO 5: LÍNEAS FUTURAS:

La investigación y desarrollo tecnológico que se lleva a cabo día tras día, es la principal causa de la aparición de nuevos algoritmos de clustering. Es por este hecho por el que este algoritmo implementado podrá ser de utilidad de cara a una posible ampliación o mejora.

En cuanto al analizador semántico, podremos realizar muchas combinaciones a la hora de, por ejemplo, buscar información. De este modo, una vez tengamos los clusters, podremos ver si determinados términos aparecen en un documento. De este modo, al haber varios recursos en un mismo cluster, a la hora de encontrar la información que necesitábamos, tendremos más variantes donde encontrarla.

En el capítulo anterior veíamos como la métrica utilizada era, en la mayor parte de los caso, la de análisis de documentos. Pues bien, otra posible ampliación podría ser utilizando el resto de métricas comentadas en dicho capítulo para fines distintos al que buscábamos en este caso.

Por último, existe la posibilidad de combinar el algoritmo implementado junto con el analizador semántico para realizar búsquedas más eficientes. Esto significa, que a la hora de buscar algo relacionado con un término, si sabemos que un determinado recurso contiene léxico relacionado con dicho término, entonces el resto de elementos pertenecientes al mismo cluster podrán ser de interés para nuestra búsqueda, ya que el contenido será similar, sino el mismo.

Cabe destacar, que además de lo comentado anteriormente, ampliando la aplicación utilizada, seguramente que todavía podremos disfrutar de más utilidades en un futuro.

CAPÍTULO 6: BIBLIOGRAFÍA:

- [OSC01] Andrew Y. Ng, Michael I. Jordan, Yair Weiss (2001). On Spectral Clustering: Analysis and an algorithm. NIPS 2001 Conference.
- [GBS00] R. Kannan, S. Vempala and A.Vetta (2000). On Clusterings: good, bad and spectral. In Proceedings of the 41st Annual Symposium on Foundations of Computer Science, 2000.
- [MJI10] Miguel Liroz, Javier Armendáriz, Iria Prieto (2010). Web Semántica: estudio de algoritmos de inicialización para clasificación con k-means. Máster en Tecnologías Informáticas (UPNA), 2010.
- [MJM05] Miguel Garre, Juan José Cuadrado, Miguel Ángel Sicilia (2005). Comparación de diferentes algoritmos de clustering en la estimación de coste en el desarrollo de software. Dpto. de Ciencias de la Computación, Universidad de
- [STSC04] Pietro Perona, Lihi Zelnik-Manor (2004). Self Tuning Spectral Clustering. Department of Electrical Engineering (California Institute of Technology).
- [AML06] Amaia Mutuberria Larrayoz (2006). Sistemas de medida de similaridad Semántica entre Documentos. Proyecto Fin de Carrera (I.T.I.G.), UPNA.
- http://ai.stanford.edu/~ang/papers/nips01-spectral.pdf
- http://www.sc.ehu.es/jiwdocoj/remis/docs/GarreAdis05.pdf
- http://ai.stanford.edu/~ang/papers/nips01-spectral.pdf
- http://www.vision.caltech.edu/lihi/Demos/SelfTuningClustering.html

Universidad
Pública de Navarra
Nafarroako
Unibertsitate Publikoa

SPECTRAL CLUSTERING

Daniel Ruiz Albéniz

Tutor: Jesús Villadangos Alonso.

Pamplona, 15 de Septiembre de 2010.

ÍNDICE

RESUMEN

- Estructura del conocimiento de las personas
  - Información *clara y precisa*.
- Algoritmo “Spectral Clustering” en Matlab
  - Grupos con *comportamiento similar*.
  - *Tiempos de ejecución*.
  - *Distancias*.
- ¿Existe relación entre los Clusters?
  - *Analizador Semántico*.

1. INTRODUCCIÓN

- Generación del Conocimiento  Cambio:
  - Nuevos Paradigmas:
    - Sociedad de la Información
  - Nueva Economía:
    - Conocimiento
  - ACTUALIDAD: Web Léxica.
- FUTURO: Web 2.0, Web 3.0, Web 4.0.

- Desarrollo:
  - Tecnologías de la Información y Comunicación (TIC).
  - Sistemas de Organización del Conocimiento (SOC).

1. INTRODUCCIÓN

- Sistemas de Organización del Conocimiento:

  - Ordenan información de la Web
    - Sistemas de Clasificación
    - Redes Semánticas
  - Organizaciones:
    - Enfoque colaborativo de la Web
    - Información para los usuarios:
      - Fácil
      - Entendible
      - Completa
      - Etc.

"ONTOLOGÍA"
"FOLKSONOMÍA"

1. INTRODUCCIÓN

- Algoritmos de Clustering:
  - *K-means:*
    - Centros iniciales ejecutados iterativamente
    - Resultados y Convergencia dependen → Valores iniciales
  - *R. Maitra:*
    - Modas locales multidimensionales → K-means
    - Coste Computacional: Elevadísimo!!
  - *D. Arthur (K-means ++):*
    - Selección de un centro aleatorio
    - 'k' itereaciones → Distancia Punto-Centro
    - Complejidad: O(nkp)
  - ***Spectral Clustering*:**
    - Vectores Propios mayores
    - Más eficiente
    - VLSI, etc.

2. SPECTRAL CLUSTERING

- ALGORITMO:

Dado un conjunto de "n" puntos, donde S= $\{s_1, s_2, \ldots, s_n\}$ en $\Re^l$, podemos agruparlos en C clusters (subconjuntos), de la siguiente manera:

1. Formamos la *matriz afinidad* $A \in \Re^{n \times n}$ definida por: $A_{ij} = e^{-\left(\frac{\|s_i - s_j\|^2}{2\sigma^2}\right)}$ para $(i \neq j)$ y $(A_{ii}=0)$, donde (s_i, s_j) es la distancia euclídea entre los vectores s_i y s_j. σ es un parámetro de escala, cuyo funcionamiento veremos luego.

2. Definir D como la *matriz diagonal* con $D_{ii} = \sum_{j=1}^{n} A_{ij}$ y construir la *matriz afinidad normalizada* $L = D^{\frac{-1}{2}} A D^{\frac{-1}{2}}$.

3. Manualmente, definimos un número aleatorio de clusters para realizar la agrupación.

4. Encontrar $x_1, \ldots, x_C$, con los C mayores vectores propios de L y *formar la matriz* $X = [x_1, \ldots, x_C] \in \Re^{n \times C}$.

5. *Renormalizar las filas de X* obteniendo como longitud la unidad: $Y \in \Re^{n \times C}$ así como: $Y_{ij} = \frac{X_{ij}}{\left(\sum_j X_{ij}^2\right)^{1/2}}$.

6. Tratar cada fila de Y como un punto de $\Re^C$ y *agrupar por vía* del *k-means*.

7. *Asignar el punto original* s_i *al cluster C* si, y solo si, la correspondiente fila i de la matriz Y ha sido asignada al cluster C.

2. SPECTRAL CLUSTERING

- CÓDIGO:
  - Función "*SC*":
    - Parámetro de Escala (*Sigma*):
      - Manualmente
      - Automáticamente: Clustering para distintos valores σ
      - *Efectos*:

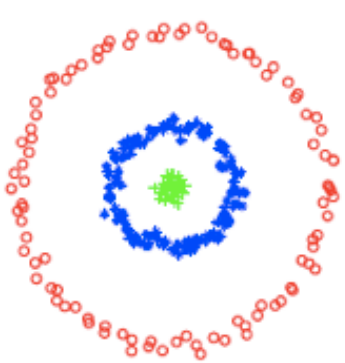


Figura 2.1. σ = 0.041235

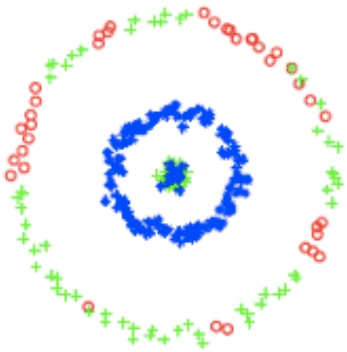


Figura 2.2. σ = 0.015625

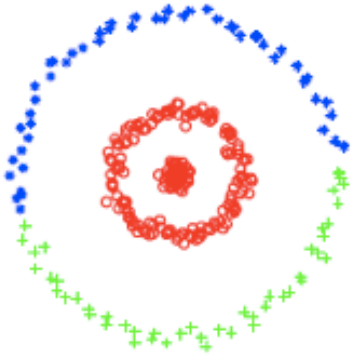
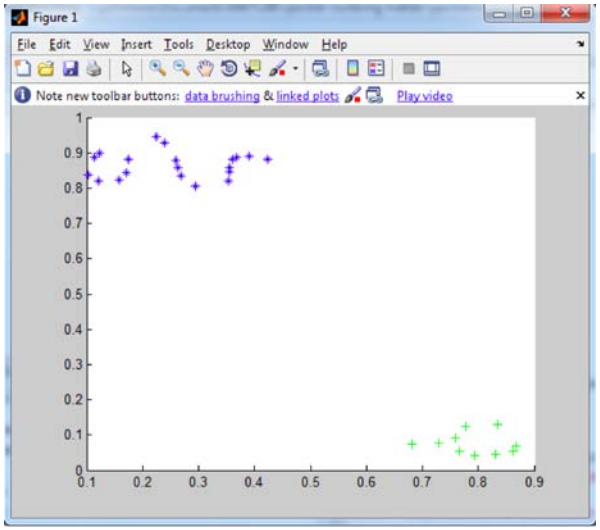


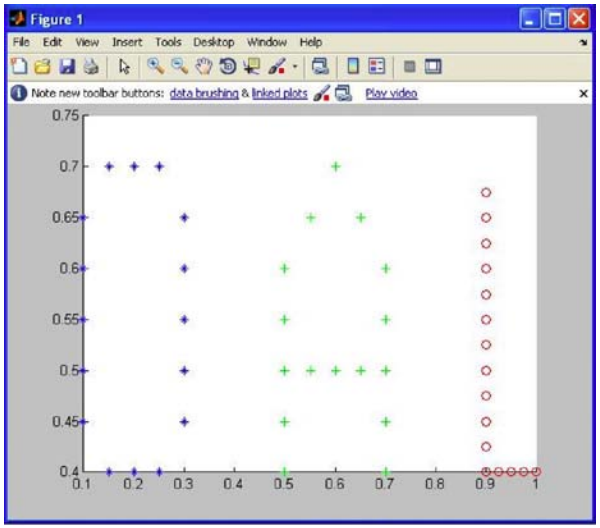
Figura 2.3. σ = 0.35355

      - ¿*Solución*?:
        - Parámetro de Escala… **LOCAL**.

2. SPECTRAL CLUSTERING

- RESULTADOS:
  - Nube de *Puntos Aleatorios* y *Letras*:



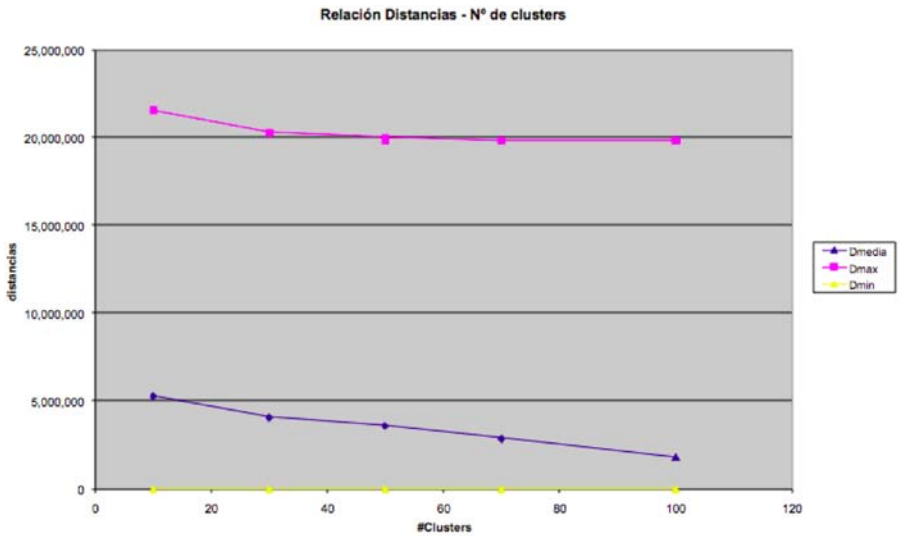


  - **Idéntico Resultado** con distinto "*Número de Vecinos*"

2. SPECTRAL CLUSTERING

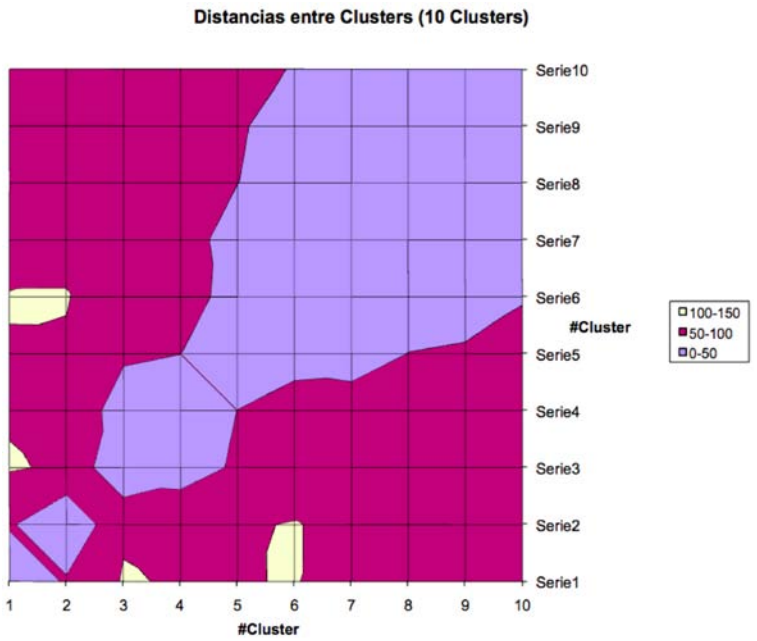
- RESULTADOS:
  - Relación Distancia-Número de Clusters:

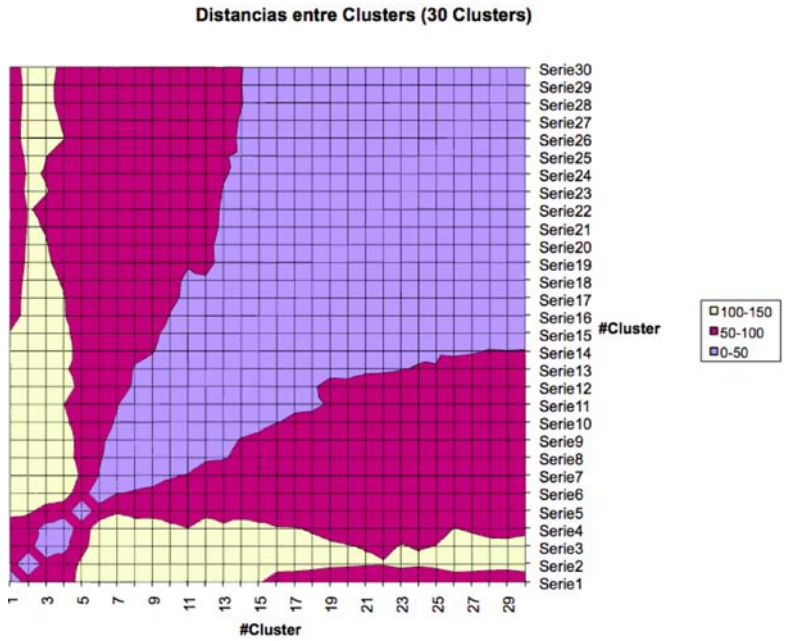
Nº_Clust	D_Med	D_Máx	D_Min
10	5.260.645	2,16e+07	24.495
30	4.067.350	2,03e+07	0
50	3.605.101	2,00e+07	0
70	2.867.062	1,99e+07	0
100	1.833.537	1,99e+07	0



2. SPECTRAL CLUSTERING

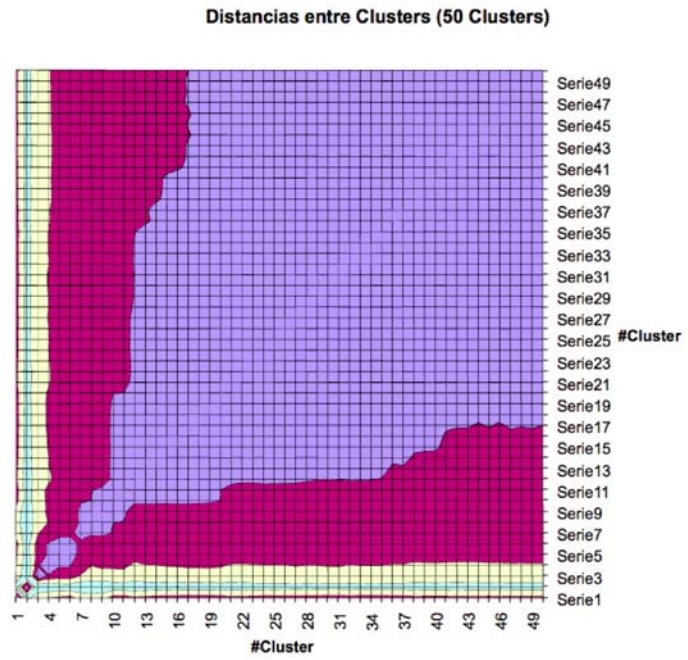
- RESULTADOS:
  - Distancia entre Clusters:

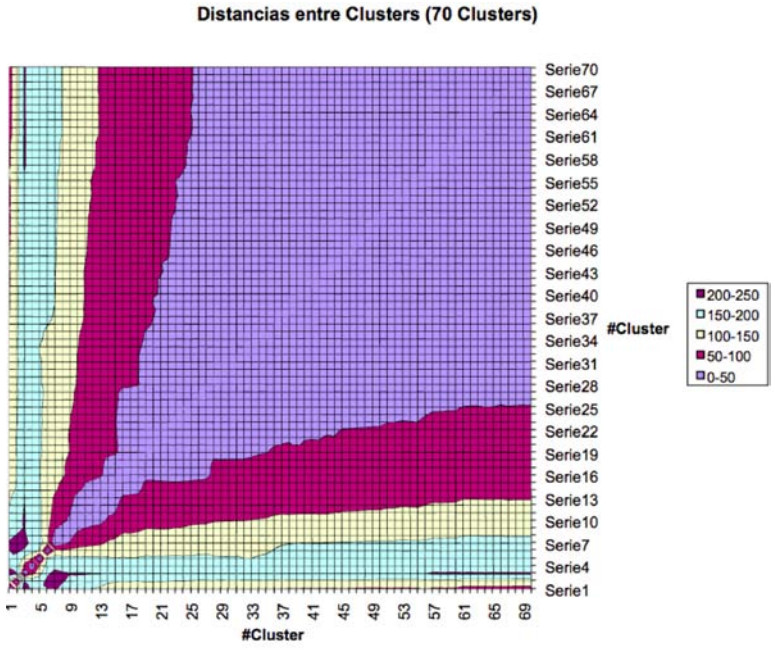




2. SPECTRAL CLUSTERING

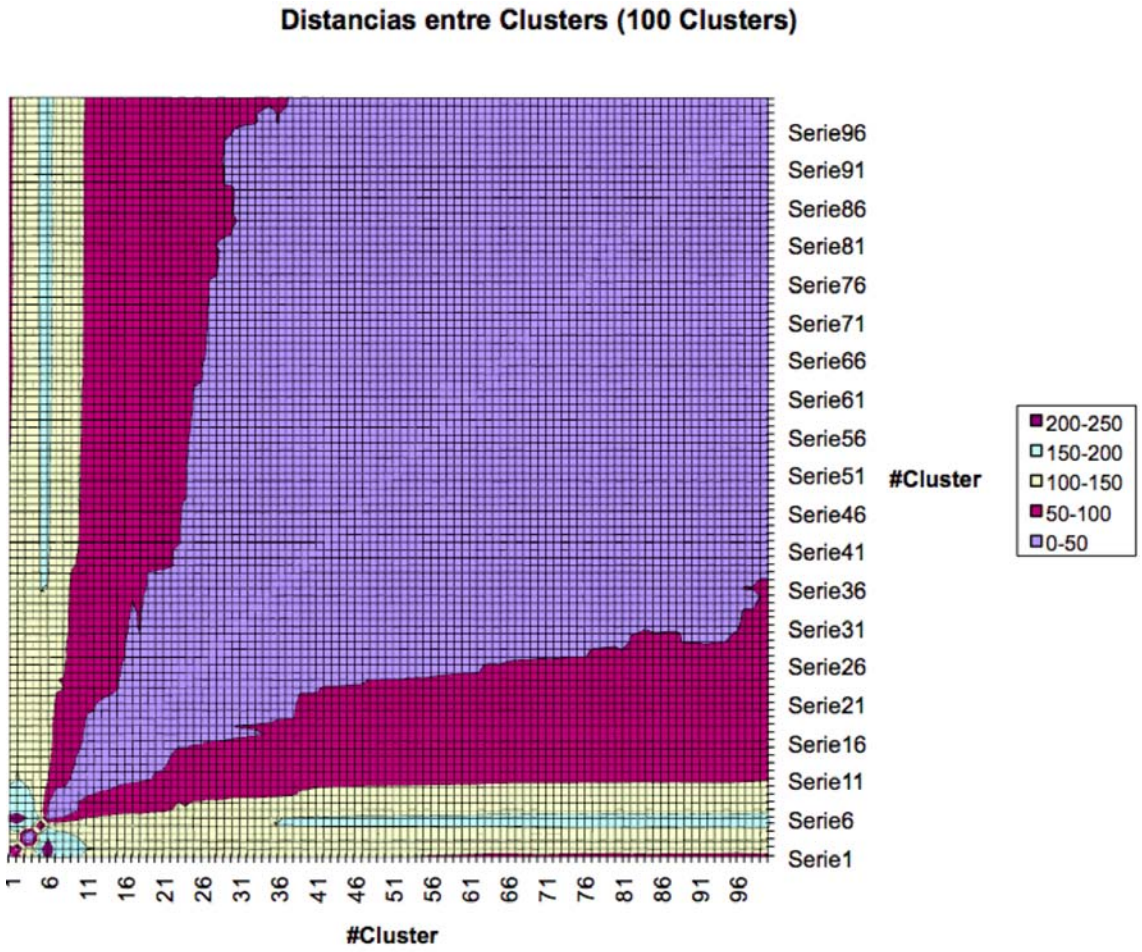
- RESULTADOS:
  - Distancia entre Clusters:





2. SPECTRAL CLUSTERING

- RESULTADOS:
  - Distancia entre Clusters:



3. ANALIZADOR LÉXICO

- DESCRIPCIÓN:
  - Internet crece…  Redundancia!
  - Diferentes esquemas  Pérdida de Información!
  - Imposibilidad de Análisis Profundo
  - *¿Solución?:*
    - Herramienta Software
    - Resultados Inteligibles, Manejables, …
    - ¡¡ **ANALIZADOR LÉXICO** !!
      - ¿Existe una *relación vinculante* entre 2 documentos?

3. ANALIZADOR LÉXICO

- MÉTRICAS A EVALUAR:
  - *Términos-Párrafo*
  - *Términos-Documento*
  - *De Términos*
  - *De Documentos*
- MODELOS DE ANÁLISIS:
  - *Binario:*
    - Similaridad independiente del nº de apariciones del término
  - *De Ocurrencias:*
    - Similaridad depende de los tokens
  - *Latent Semantic Analysis (LSA):*
    - Similaridad depende de la correlación y dimensionalidad
  - *Coeficiente de Correlación de Spearman:*
    - Similaridad depende de la asociación de dos rangos

3. ANALIZADOR LÉXICO

- RESULTADOS:

CLUSTER	RECURSOS	DIST.MÁXIMA	DIST.MÍNIMA	DIST.MEDIA
1	111, 124, 127.	256,1484	189	144,6160
2	32, 122, 170, 194.	1,0938E+03	151,2052	532,2176
47	81, 88, 103.	5,1962	2,4495	2,8537
48	27, 144.	226,6627	226,6627	113,3314
50	110.	0	0	0

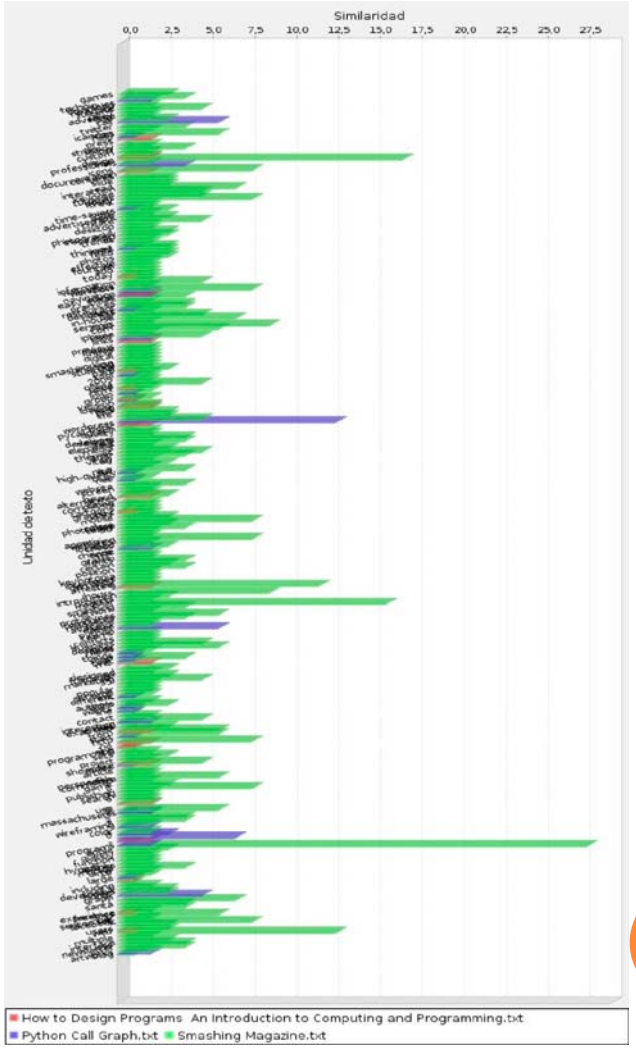
- 200 filas “Macro-Matriz” x 50 Clusters
- “XAMPP”
- Implementación de Araña u otros (Html2Txt, etc)

3. ANALIZADOR LÉXICO

- RESULTADOS:
  - Cluster 1:
    - Recursos:
      - 111: *http://pycallgraph.slowchop.com*
        - Módulo de Python que crea “call graphs”
      - 124: *http://www.smashingmagazine.com*
        - Blog de tutoriales y recursos de diseño
      - 127: *http://www.htdp.org*
        - “How To Design Programs”

3. ANALIZADOR LÉXICO

- RESULTADOS:
  - Cluster 1:
    - Mayoría verde
    - “Design”: 2, 0 y 17 veces
    - Idiomas diferentes
    - Distancia Intra-Cluster: 256,1484



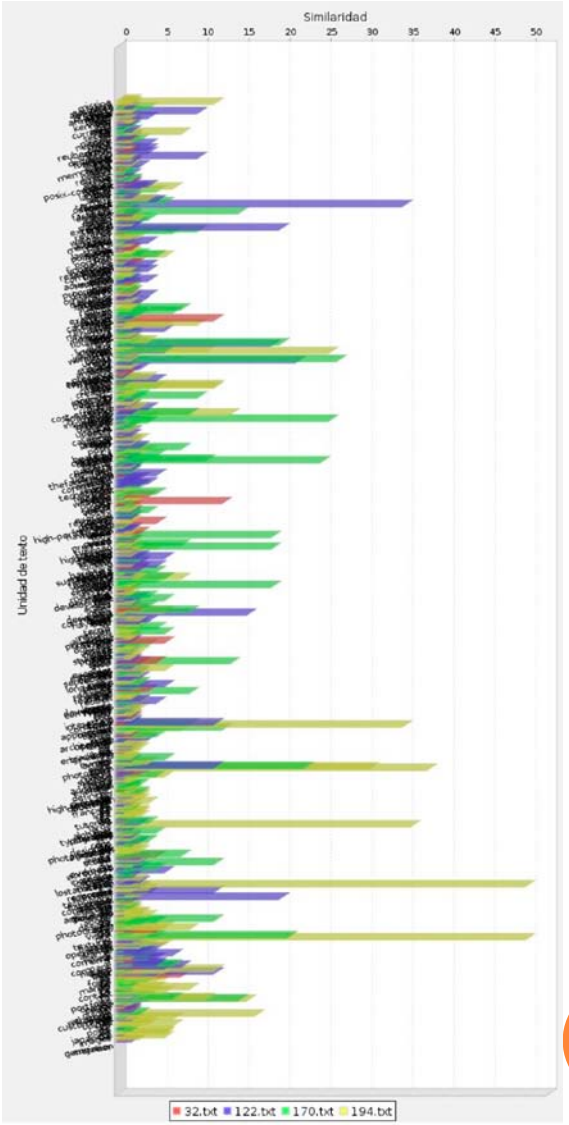
3. ANALIZADOR LÉXICO

- RESULTADOS:
  - Cluster 2:
    - Recursos:
      - 32: *http://www.jetbrains.com/resharper/*
        - Herramientas inteligentes (Visual Studio) para descargar
      - 122: *http://www.niallkennedy.com/blog/2009/04/facebook-haystack.html*
        - Blog (Reescritura del almacén de fotos de Facebook)
      - 170: *http://webdesignledger.com/freebies/70-beautiful-damask-patterns-and-textures*
        - Descarga de fondos y accesorios de Damasco
      - 194: *http://reubenmiller.typepad.com/my_weblog/2008/09/21-unexpected-a.html*
        - Blog (22 alfabetos más creativos del mundo) para descargar

3. ANALIZADOR LÉXICO

- RESULTADOS:
  - Cluster 2:

    - Similitud en el formato
    - Herramientas de diseño o similar
    - “Tools”, “Designs”
    - “Comments”, “2010, …”, “Search”, etc



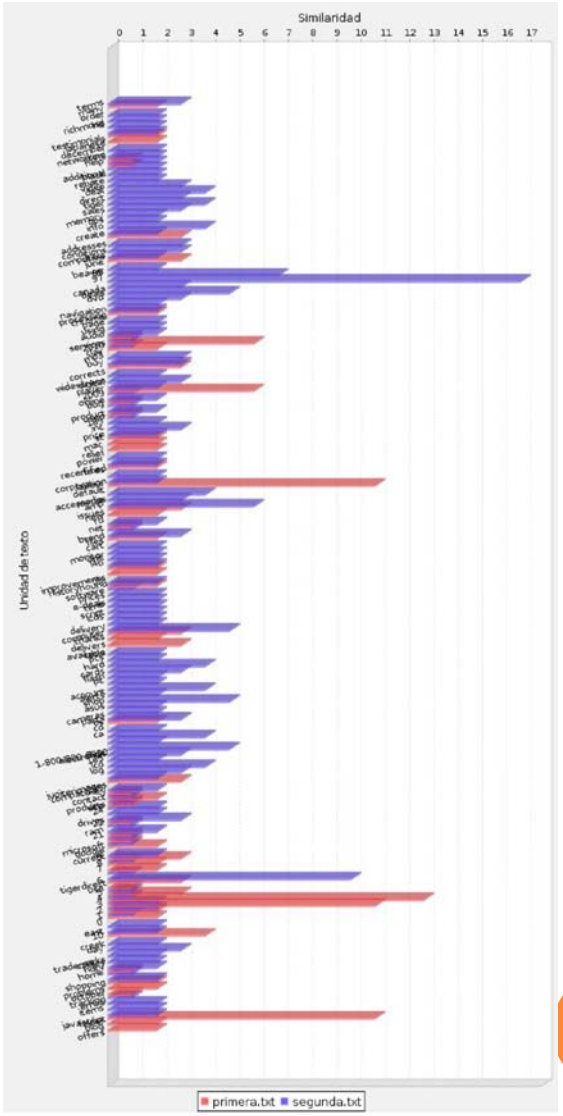
3. ANALIZADOR LÉXICO

- RESULTADOS:
  - Cluster 48:
    - Recursos:
      - 27: http://stclairsoft.com/index.html
        - Descarga de Software gratuito
      - 144: http://www.tigerdirect.ca/indexca.asp?
        - Tienda de hardware y software

3. ANALIZADOR LÉXICO

- RESULTADOS:
  - Cluster 48:

    - Material Informático
    - Elementos software
    - Similar aspecto físico
    - “Software”, “Products”
    - “Help”, “Services”, “Contact”, “Home”
    - Dist. Máx: 226,6627



4. CONCLUSIONES

- Algoritmo "Spectral Clustering":
  - Resultados muy satisfactorios
- Analizador semántico:
  - Cierta incertidumbre
- Bastante similar al juicio humano

5. LÍNEAS FUTURAS

- Diferentes combinaciones
  - Buscar información
- Uso del resto de métricas
- Búsquedas más eficientes
- Ampliación de la aplicación