Universidad Zaragoza
1542

Proyecto Fin de Carrera

Evaluation of a Hierarchical Taxonomy Preparation Method for Document Classification

Autora

Sara Gracia Hernández

Director

Dr. Wolf-Tilo Balke

Ponente

Dr. Eduardo Mena

Escuela de Ingeniería y Arquitectura
Septiembre 2013

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Departamento de Informática e Ingeniería de Sistemas Universidad Zaragoza

AGRADECIMIENTOS

A mi madre **Paqui** y a mi hermana **Beatriz** por confiar en mí;

a mi padre **Miguel Ángel** por darme fuerzas desde la distancia;

a **Eduardo** por ser la constante de mi vida;

a mis abuelos y mis tíos por apoyarme siempre ,

a mis amigas por tantos momentos vividos

a mis amigos Erasmus, por este gran año,

y para terminar, a mi director **Simon Barthel**, que pese a las dificultades del idioma,

ha tenido paciencia y me ha guiado en el trabajo.

Evaluation of a Hierarchical Taxonomy Preparation Method for Document Classification

RESUMEN

Para la realización de entornos virtuales de investigación en el campo de las matemáticas, el **acceso de manera óptima a la literatura** de conocimiento matemático es fundamental. El continuo crecimiento de información ha provocado que el acceso a los segmentos relevantes sea prácticamente imposible, además de ocasionar que el proceso de clasificación de los documentos por parte tanto de los centros de información y las bibliotecas sea cada día más difícil y complejo.

La base de datos **"Zentralblatt Mathematik" de FIZ Karksruhe y el portal Get-Info del TIB Hannover** reciben diariamente grandes cantidades de documentos matemáticos y usan el método de clasificación MSC (Mathematical Subject Classification) para **indexarlos de forma manual**. Debido a esta sobrecarga de información, es necesario apoyar el trabajo de los bibliotecarios para el trabajo de indexación diario.

Por esta razón, el **proyecto** $\boldsymbol{DeLiVer^{Math}}$, dentro del cual se sitúa este Proyecto Final de Carrera, investiga **procesos automatizados para la indexación de contenido** basado en taxonomías e información contextual en el campo de las matemáticas.

Los objetivos de este proyecto han sido **evaluar una aproximación para la introducción de superclases sobre la clasificación MSC**, implementar el método y evaluarlo con datos de una biblioteca digital matemática, para mejorar el rendimiento global de la clasificación y a su vez, examinar la estructura de la taxonomía MSC.

El acceso a la biblioteca digital matemática se ha realizado a través de un corpus de datos perteneciente a un subconjunto de una librería digital matemática. Este corpus está en formato JSON, y está formado por documentos, categorías (MSC) y términos, los cuales son la base del algoritmo que se ha desarrollado.

El algoritmo está definido en el paper *"Hierarchical Taxonomy Preparation for Text Categorization Using Consistent Bipartite Spectral Graph Copartitioning"*. Se ha implementado en *MATLAB* y consiste en la creación de dos matrices base y en aplicar una serie de técnicas de descomposición sobre ellas, entre las que se encuentran principalmente *SVD* (Singular Value Decomposition) y *GSVD* (Generalized Singular Value Decomposition) , para la creación de un vector integrado normalizado, el cual se clusteriza para la obtención del k-particionamiento de categorías. La idea principal del método es aprovechar la información complementaria que se encuentra en ambas matrices, siendo los documentos el puente entre las categorías y los términos, para conseguir un clustering de categorías más razonable y eficaz. Para determinar el número de óptimo de clusters se ha aplicado la técnica *Intra-Cluster Distance Measure* y para realizar el clustering de categorías se ha usado la función *KMEANS*. Una vez que el algoritmo ha sido implementado, el siguiente paso es la construcción de la taxonomía jerárquica; para ello, el algoritmo se aplica de forma recursiva para obtener la jerarquía de categorías.

Para evaluar el algoritmo, se han realizado las siguientes aproximaciones, y se han comparado los resultados con la clasificación de categorías MSC.

- Subconjunto del corpus de datos
- Categorías del primer nivel de MSC
- Subconjunto de categorías, con el mismo primer nivel
- Subconjuntos de categorías bien definidos

INDICE

1. INTRODUCCIÓN

Este Proyecto Final de Carrera surge de la necesidad de facilitar el trabajo a los bibliotecarios de documentación matemática en su trabajo diario de indexación de documentos. Esta necesidad surge principalmente debido al incremento de información que ocurre hoy en día, lo cual afecta de forma directa a los proveedores de esta.

El propósito de este proyecto es obtener una primera aproximación para intentar sustituir el trabajo manual de indexación de documentación matemática hecho por expertos, para poder realizar la clasificación de forma automática de acuerdo a la clasificación MSC. Los primeros experimentos han mostrado que los clasificadores para el primer nivel de las taxonomías tienen una pérdida en el rendimiento global, en comparación con el trabajo manual hecho por los expertos. Es aquí donde este proyecto final de carrera cobra sentido, su propósito es analizar si se pueden obtener mejores resultados si introducimos superclases en la taxonomía como un primer nivel de clasificación, para profundizar después en la jerarquía.

El trabajo realizado en este proyecto ha consistido en evaluar una aproximación para la introducción de superclases sobre la clasificación MSC, implementar el método y evaluarlo con datos de una biblioteca digital matemática, para mejorar el rendimiento global de la clasificación y a su vez, examinar la estructura de la taxonomía MSC.

Como fuente de datos, se ha accedido a un corpus de datos perteneciente a una la biblioteca matemática digital. Este corpus de datos está formado por documentos, categorías (MSC) y términos, los cuales son la base del algoritmo que se ha implementado para conseguir el propósito del proyecto.

Este algoritmo está definido en el paper *"Hierarchical Taxonomy Preparation for Text Categorization Using Consistent Bipartite Spectral Graph Copartitioning"* (para más información sobre el mismo consultar Anexo A). Consiste en la creación de dos matrices base y en aplicar una serie de técnicas de descomposición sobre ellas, para la creación de un vector integrado normalizado, el cual se clusteriza para la obtención del k-particionamiento de categorías. La idea principal del método es aprovechar la información complementaria que se encuentra en ambas matrices, siendo los documentos el puente entre las categorías y los términos, para conseguir un clustering de categorías más razonable y eficaz. . Una vez que el algoritmo ha sido implementado, el siguiente paso es la construcción de la taxonomía jerárquica y su posterior análisis.

1.1 Motivación

Para la realización de entornos virtuales de investigación en el campo de las matemáticas, el acceso de manera óptima a la literatura de conocimiento matemático es fundamental. La construcción de bibliotecas digitales en el campo de las matemáticas incluye tanto la construcción de un vocabulario controlado y una taxonomía con campos bien definidos, así como el desarrollo de métodos para el análisis automatizado de contenidos y asignación de documentos, los cuales son el núcleo del conocimiento matemático.

En este contexto, el proyecto $DeLiVer^{Math}$ se encargará de desarrollar métodos y herramientas para la indexación y recuperación de contenido. El continuo crecimiento de información ha provocado que el acceso a la información relevante sea casi imposible, además ha ocasionado que el desarrollo por parte tanto de los centros de información específicos y de las bibliotecas, sea cada día más difícil y complejo.

El proyecto $DeLiVer^{Math}$ se encarga de investigar los procesos automatizados para la indexación de contenido basado en taxonomías e información contextual en el campo de las matemáticas. Para llevar a cabo sus objetivos, están asociados con el *TIB* y el *FIZ Karlsruhe* , proveedores de documentación dentro del ámbito de las matemáticas, la ingeniería y las ciencias de la naturaleza. La base de datos "*Zentralblatt Mathematik*" de *FIZ Karksruhe* y el portal *Get-Info* del *TIB Hannover*, reciben un gran volumen de documentos matemáticos a diario, y usan la codificación de *MSC* (Mathematical Subject Classification) para indexarlos de forma manual; debido a esta sobrecarga de información, es necesario apoyar el trabajo de los bibliotecarios para el trabajo de indexación diario.

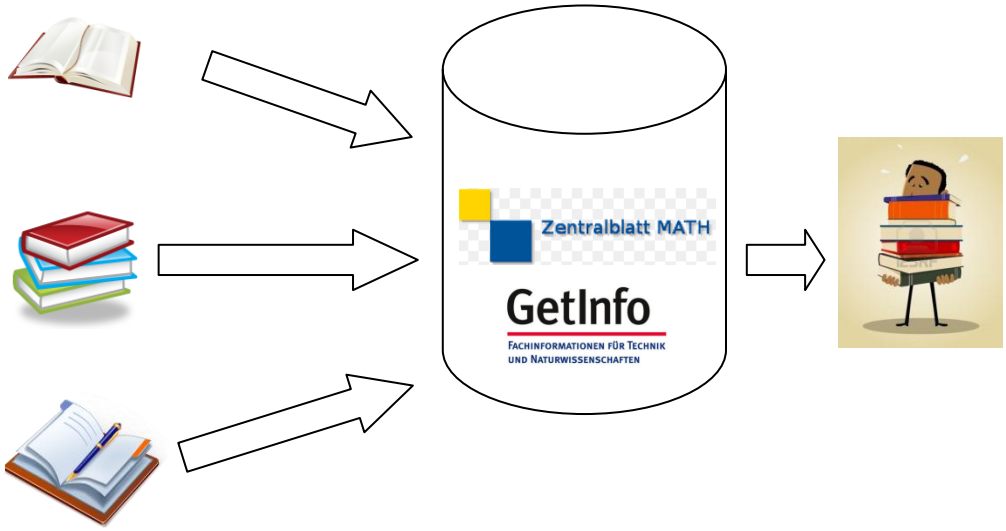


Figura 1: Esquema de llegada e indexación de documentos por bibliotecarios

El uso óptimo de la información requiere la identificación de un subconjunto relevante de información de un gran conjunto de información disponible. El problema radica en que identificación de la información relevante es cada vez más difícil y a su vez, las fuentes de datos de las que disponemos son más heterogéneas. Por lo tanto, se necesita la integración de métodos semánticos y extensiones de los procesos de búsqueda para la construcción de las redes de conocimiento.

Los objetivos del proyecto $DeLiVer^{Math}$ son:

- El desarrollo de un procedimiento semi-automática para apoyar la creación y el mantenimiento de vocabularios controlados y tesauros
- El desarrollo de un proceso totalmente automatizado para la indexación y la selección o clasificación de documentos matemáticas desconocidos con precisión y calidad
- El desarrollo de capacidades de recuperación innovadores e individualmente configurables y de procedimientos de clasificación para el acceso a la información.

Este Proyecto Final de Carrera se sitúa dentro del segundo objetivo del proyecto $DeLiVer^{Math}$, desarrollar un proceso automatizado para la indexación, selección y clasificación de documentos matemáticas desconocidos con precisión y calidad.

1.2 Contexto de desarrollo

Este proyecto forma parte del trabajo de investigación desarrollado por el grupo IFIS[1] de la Universidad Técnica de Braunschweig en el proyecto $DeLiVer^{Math_2}$.

Con el proyecto $DeLiVer^{Math}$ se intenta solventar el problema encontrado a la hora de indexar documentos matemáticos de forma manual. Por ello se propone como solución intentar obtener una primera aproximación para poder sustituir el trabajo manual de indexación de documentación matemática hecho por expertos, para poder realizar la clasificación de forma automática de acuerdo a la clasificación MSC. Los primeros experimentos desarrollados por $DeLiVer^{Math}$ han mostrado que los clasificadores para el primer nivel de las taxonomías tienen una pérdida en el rendimiento global, en comparación con el trabajo manual hecho por los expertos.

Es aquí donde este Proyecto Final de Carrera cobra sentido, el propósito es analizar si se pueden obtener mejores resultados si introducimos superclases en la taxonomía como un primer

[1] *http://www.ifis.cs.tu-bs.de/content/delivermath*
[2] *http://www.l3s.de/delivermath/project.html*

nivel de clasificación, para profundizar después en la jerarquía. Para ello se ha implementado el algoritmo propuesto en el paper *"Hierarchical Taxonomy Preparation for Text Categorization Using Consistent Bipartite Spectral Graph Copartitioning"* cuyo objetivo es preparar una taxonomía jerárquica para categorizaciones de textos, en nuestro caso, documentos matemáticos.

La siguiente figura muestra el esquema que se sigue para poder realizar la categorización de los documentos.

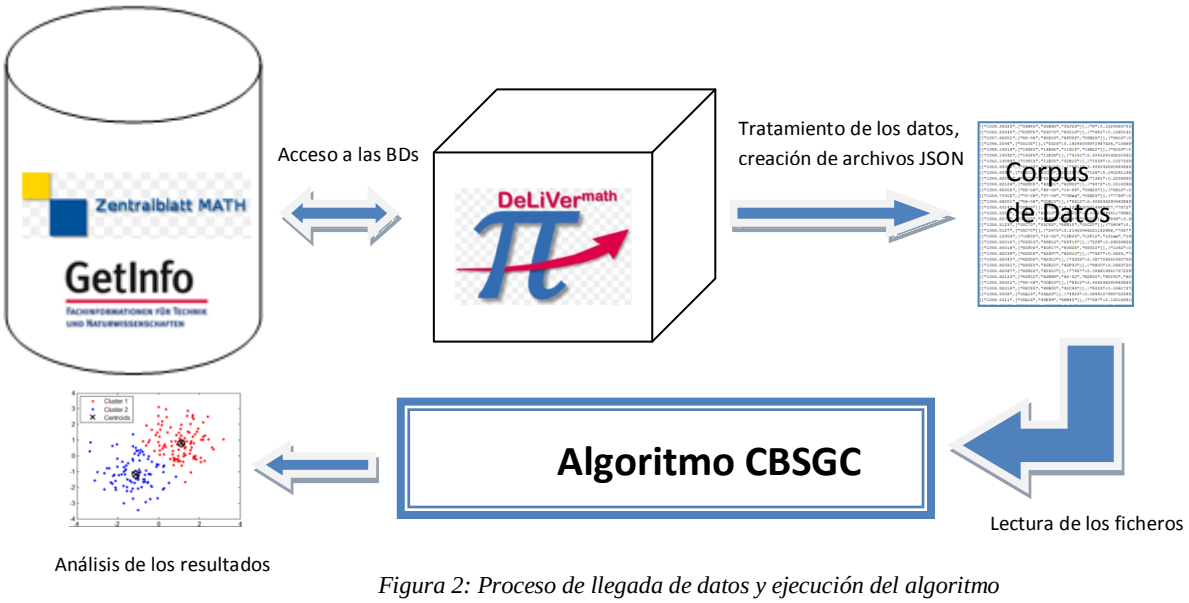


Figura 2: Proceso de llegada de datos y ejecución del algoritmo

El proyecto $DeLiVer^{Math}$ tiene acceso a las bases de datos de documentación matemática Zentralblatt Math [3]y Get-Info. [4]Estas bases de datos contienen registros bibliográficos con reseñas o resúmenes de revistas y publicaciones periódicas escritos por expertos, y sus entradas se clasifican de acuerdo a la codificación MSC.

Desde otros proyectos de $DeLiVer^{Math}$ se realiza el acceso a las bases de datos y se encargan de crear ficheros con la información relevante para la investigación. Estos ficheros son el corpus de datos con el que voy a trabajar en mi proyecto.

El proyecto se desarrola en Python, pero para la implementación del algoritmo que tenemos que usar como base, la función GSVD , indispensable para su cometido, no está presente y no se ha podido conseguir alternativas para que funcionara en Python, por lo que se decidió implementarlo en su totalidad en Matlab. En el apartado de *Implementación* se abordará este tema.

[3] *www.zentralblatt-math.org/zmath/*
[4] *https://getinfo.de/app*

1.3 Objetivos del Proyecto

En base al contexto planteado en el apartado anterior, el objetivo principal de este Proyecto Fin de Carrera es analizar si se pueden obtener mejores resultados en los clasificadores para el primer nivel de las taxonomías si se introduce un nivel de superclases como un primer nivel de clasificación, para profundizar después en la jerarquía.

El objetivo principal de este proyecto engloba a su vez una serie de sub-objetivos los cuales son necesarios para poder alcanzar conclusiones el proyecto.

- Evaluar una aproximación para la introducción de superclases sobre taxonomía
- Implementar el algoritmo descrito en el paper *"Hierarchical Taxonomy Preparation for Text Categorization Using Consistent Bipartite Spectral Graph Copartitioning"* [5]
- Evaluar el algoritmo con datos de una biblioteca digital matemática
- Examinar la estructura de la taxonomía MSC para mejorar el rendimiento global de la clasificación

Para evaluar el algoritmo, se han realizado las siguientes aproximaciones, y se han comparado los resultados con la clasificación de categorías MSC.

- Subconjunto del corpus de datos
- Categorías del primer nivel de MSC
- Subconjunto de categorías, con el mismo primer nivel
- Subconjuntos de categorías bien definidos

[5] *[B. G. B. Gao, T.-Y. L. T.-Y. Liu, G. F. G. Feng, T. Q. T. Qin, Q.-S. C. Q.-S. Cheng, and W.-Y. M. W.-Y. Ma, "Hierarchical taxonomy preparation for text categorization using consistent bipartite spectral graph copartitioning," IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 9. pp. 1263–1273, 2005*

1.4 Estructura de la Memoria

Este documento se divide en dos partes: la primera parte contiene la memoria, que resume el trabajo realizado; la segunda contiene los anexos que explican más en detalle ciertos aspectos de la memoria.

La memoria contiene los siguientes capítulos:

- Capítulo 1 – Introducción: describe la motivación inicial del proyecto, su contexto de desarrollo, los objetivos a alcanzar y la estructura de la memoria.

- Capítulo 2 – Análisis del Problema: en el primer apartado describe el algoritmo que se ha utilizado para realizar la categorización de documentos, explicando el coclustering de documentos-términos y categorías-documentos, las funciones SVD y GSVD y los pasos del mismo; y en el segundo el corpus de datos.

- Capítulo 3 – Diseño: explica el diseño que se ha llevado a cabo para la creación de los vectores y matrices necesarios para el desarrollo del algoritmo CBSGC, y cómo se ha resuelto la parte del clustering para realizar el k-particionamiento de las categorías

- Capítulo 4 – Implementación: explica cómo se ha realizado la implementación para la conseguir la categorización de los documentos aplicando el algoritmo CBSGC, es decir, construcción de vectores, matrices, k-particionamientos y construcción de la jerarquía.

- Capítulo 5 – Resultados Obtenidos: presenta los resultados que hemos obtenidos al aplicar el algoritmos a datos de una librería matemática digital.

- Capítulo 6 – Conclusiones: se la consecución de objetivos, el trabajo futuro a realizar y la valoración personal.

- Capítulo 7- Bibliografía: está compuesto por los documentos que nos han servido de ayuda para la realización de este proyecto

Los anexos son los siguientes:

- Anexo A – Paper "Hierarchical Taxonomy Preparation for Text Categorization Using Consistent Bipartite Spectral Graph Copartitioning" : Contiene el paper de documentación matemática que se ha utilizado para desarrollar este Proyecto Final de Carrera

- Anexo B – Clasificación MSC: Contiene la descripción de la clasificación Mathematical Subject Classification, la cual se usa para realizar la categorización de los documento

- Anexo C- SVD y GSVD: Se explica con mayor detalle el problema SVD y el problema GSVD, citando teoremas y razonando el por qué de su uso en el algoritmo.

- Anexo D – Corpus de Datos: Este anexo contiene información sobre el corpus de datos que hemos usado explicada con detalle.

- Anexo E- k-means: este anexo contiene las pruebas para la selección de los parámetros en la función kmeans de matlab.

- Anexo F- Algoritmo CBSGC: Este anexo contiene los pasos, con todas las funciones implicadas, para la creación de la taxonomía jerárquica

- Anexo G- Gestión del Proyecto: describe la dedicación en tiempo del proyecto, presentando la distribución de este tiempo en las distintas tareas.

2. ANÁLISIS DEL PROBLEMA

Este capítulo consta de dos apartados, en el primero se va presentar el algoritmo Consistent Bipartite Spectral Graph Copartitioning definido en el paper *"Hierarchical Taxonomy Preparation for Text Categorization Using Consistent Bipartite Spectral Graph Copartitioning"* , con el cual se pretende conseguir una primera aproximación para el cálculo de superclases en la taxonomía. En el segundo apartado se va a analizar el corpus de datos con el que se va a trabajar para poder realizar la categorización de los documentos.

2.1 **Algoritmo CBSGC** (Consistent Bipartite Spectral Graph Copartitioning)

Antes de centrarnos en el algoritmo, se va a hacer una breve introducción sobre la base en la que se apoya este, que nos servirá para poder entender qué hace y cómo funciona.

En primer lugar se introducirá la representación de los documentos, la cual es necesaria para poder presentar el problema de coclustering de documento-término, en segundo lugar las categorías, las cuales sirven para representar el problema de coclustering categoría-documento, a continuación se explicará en qué consiste el problema Singular Value Decomposition y el problema Generalized Singular Value Decomposition y finalmente se abordará el algoritmo CBSGC.

2.2.1 Antecedentes

Existen diferentes métodos para abordar el problema de la clasificación multiclase. Una de las estrategias que se puede seguir son las máquinas de soporte vectorial, que utilizan la estrategia *one-against-rest*. La idea es construir un modelo capaz de predecir si un nuevo dato, cuya categoría se desconoce, pertenece a una categoría o a la otra. Este tipo de clasificadores trabajan bien cuando el número de categorías es pequeño, pero desde hace varios años, el problema de las escalas de la clasificación multiclase se está viendo incrementada. Debido a la gran magnitud de datos, trabajar con métodos que se basan en información plana, hacen que la escalabilidad del método se vea afectada y que los tiempos de cómputo y de entrenamiento se vean afectados.

Para abordar este problema se propone el uso de la estructura jerárquica interna existente entre las categorías para dividir la tarea de clasificación. Como resultado se obtiene una máquina

de soporte vectorial jerárquica, en la que cada categoría hijo es entrenado para ser distinguido con otras categorías con el mismo padre.

Sin embargo, no todos los corpus de datos tienen una taxonomía jerárquica dada de forma explícita, por lo que en muchos casos este tipo de clasificación no se puede llevar a cabo. Para afrontar este problema, se necesita extraer una jerárquica de los datos y usarla para organizar los clasificadores jerárquicos, es decir, pre-procesar el corpus de datos para prepararlo para realizar la clasificación jerárquica

La solución adoptada y propuesta en este paper consiste en usar un grafo bipartito para representar la relación existente entre categorías y documentos, y usar otro para representar la relación entre documentos y términos, y a continuación particionar ambos grafos consistentemente resolviendo un problema de descomposición en valores singulares generalizado (GSVD).

2.2.2 Coclustering basado en Documento – Término

La idea en la que se basa el coclustering de documentos- términos es tratar los términos de cada documento como si fueran características.

$D = d_1, d_2, \dots , d_n$ es la representación de los documentos existentes en el corpus y $T = w_1, w_2, \dots , w_t$ los términos.

A partir de esto, podemos representar la relación entre documentos y términos con un grafo bipartito no direccionado (ver figura 3). Este grafo lo representamos con el triplete $G = (D, T, E)$, donde E representa el conjunto de aristas $\{< d_i\, w_j > \mid d_i \,\epsilon\, D, w_j \,\epsilon\, T\}$, y D y T los conjuntos de vértices pertenecientes a documentos y términos respectivamente.

Un eje $< d_i\, w_j >$ existe si y solo si el termino w_j tiene una ocurrencia en el documento d_i y su peso E_{ij} es igual a la frecuencia del término.

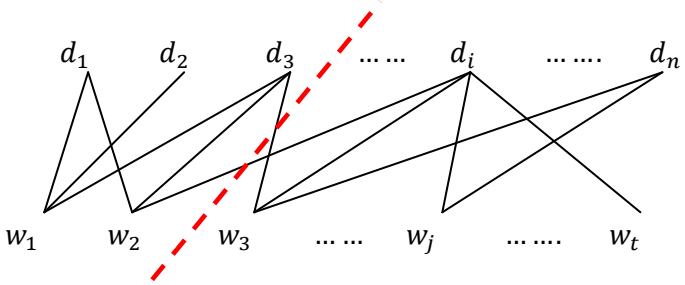


Figura 3: Grafo Bipartito Documento - Término

Con la información de este grafo construiremos la matriz de adyacencia B, la cual representará la relación documentos – términos.

En esta matriz cada elemento B_{ij} representará el peso de la arista E_{ij} que se corresponde a la ocurrencia de los términos, y si no existe valdrá 0.

$$B_{ij} = \begin{cases} E_{ij} & if < i,j > \in E \\ 0 & en\ otro\ caso \end{cases}$$

2.2.3 Coclustering basado en Categoría – Documento

Para el coclustering de categoría-documento introducimos un nuevo componente, las categorías. A cada documento se le asignaran un grupo de categorías pertenecientes al conjunto $C = c_1, c_2, \dots, c_m$, donde *m* es el número de total de categorías.

Podemos representar la relación entre categorías y documentos nuevamente con un grafo bipartito no direccionado (ver figura 4). En este caso el conjunto de aristas *E* del grafo será $\{< c_i\, d_j > | c_i \in C, d_j \in D\}$,

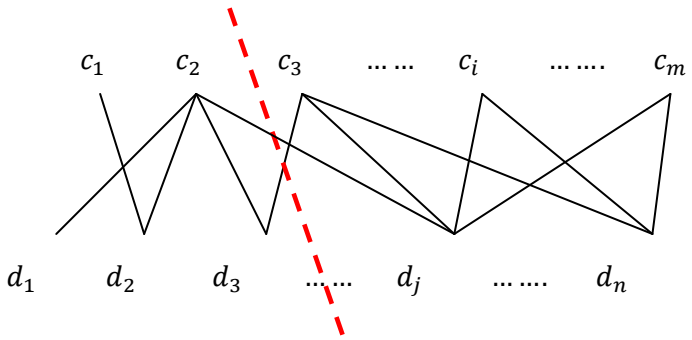


Figura 4: Grafo Bipartito Categoría - Documento

Con la información de este grafo construiremos la matriz de adyacencia A, la cual representará la relación categorías – documentos. En esta matriz, las filas corresponden a categorías y las columnas a documentos. Cada elemento A_{ij} indica la correlación entre el documento d_j y la categoría c_i. Si el documento d_j pertenece a *k* categorías $c_1, c_2, \dots, c_k$, los pesos $A_{1j}, A_{2j}, \dots, A_{kj}$ serán $\frac{1}{k}$, y los otros elementos de la columna *j*th de la matriz serán 0.

2.2.4 Problemas SVD y GSVD

La idea principal del algoritmo es encontrar una partición de los vértices de los grafos bipartitos (ver figura 3 y figura 4), tal que el *corte* (línea de puntos roja), suponiendo que el conjunto total de vértices V que componen el grafo, se particiona en dos subconjuntos V_1 y V_2 (en nuestro caso sería, si nos centramos en el grafo documento-término $V_1 = D$ y V_2 = T, siendo D y T conjuntos de vértices pertenecientes a los documentos y términos respectivamente), pueda ser *minimizado*[6].

$$cut(V_1, V_2) = \sum_{i \in V_1, j \in V_2} B_{ij}$$

En diferentes publicaciones [7] [8] [9], se ha probado que valor propio asociado con el segundo valor propio menor λ_2 del problema de valores propios generalizados produce un embedding óptimo de la partición de los vértices que minimiza el corte. Este tipo de problema, después de una serie de deducciones que no competen al propósito de este proyecto, se puede transformar en un problema de descomposición en valores singulares.

Aplicando SVD se quiere conseguir iterativamente refinar la partición de cada grafo con la información contenida en el otro, de esta manera, el estado estacionario tendrá la misma partición de documentos en ambos grafos y el clustering resultante de categorías será más razonable y eficaz.

Al aplicar SVD sobre el grafo bipartito de documentos- términos (ver figura 3) y sobre el gde de categorías – documentos (ver figura 4) se espera que el particionamiento de documentos en ambos sea el mismo, en cambio esto no ocurre así, por lo que nuestro problema no sería consistente; este problema de inconsistencia se debe sobre todo a que las propiedades para crear las matrices base (ver apartado *Pasos del Algoritmo*) que sirven de entrada al problema SVD son diferentes.

Para solucionar este problema, se propone no usar un embedding óptimo en cada grafo, para de esta manera, intentar conseguir que ambos sean consistentes. Aunque la partición de cada grafo no es la óptima, cuando se consideran los dos grafos como un todo, el resultado de particionamiento es más eficaz. Esto se consigue aplicando GSVD en lugar de SVD

---

[6] *En la teoría de grafos, un corte mínimo de un gráfico es un corte cuyo corte conjunto tiene el menor número de elementos o la suma más pequeña de pesos posibles.*

[7] *I.S. Dhillon, "Coclustering Documents and Words Using Bipartite Spectral Graph Partitioning," Proc. SIGKDD '01, 2001.*

[8] *G.H. Golub and C.F.V. Loan, Matrix Computations, third ed. Johns Hopkins Univ. Press, 1996.*

[9] *J. Shi and J. Malik, "Normalized Cuts and Image Segmentation," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 22, pp. 888-905, 2000.*

Con esta función se consigue, dándole como entrada las matrices A y B normalizadas (matrices $\widehat{A}$ y $\widehat{B}$, ver apartado *Pasos del Algoritmo*)

$$\begin{cases} \hat{A} = U\ C\ X^T \\ \hat{B} = X\ S\ V^T \end{cases}$$

que la matriz *X* resultante represente el particionamiento de documentos que reúne las restricciones de ambos grafos, y que las matrices *U* y *V* integren el particionamiento de categorías y términos respectivamente. De esta manera, conseguimos que el resultado garantice consistencia. En el anexo C se tratan estos dos problemas en mayor profundidad.

2.2.5 Pasos del algoritmo

El algoritmo parte de la creación de las dos matrices que representan el coclustering de documentos –términos y coclustering de categorías –documentos. Estas matrices se normalizan y se les aplica GSVD. Dado que las condiciones de GSVD son menos restrictivas que las de SVD, y que la información que devuelve GSVD no es explícita, hay que encontrar la manera de poder acceder a esa información, eso se consigue aplicando una transformación lineal. Después de esto aplicamos SVD, para poder descomponer la matriz resultante en dos matrices, una que representa el embedding de términos y la otra el categorías, y a continuación las refinamos nuevamente con una nueva transformación lineal. Una vez que tenemos estas dos últimas matrices, se crean los vectores normalizados integrados, y se aplica clustering para obtener el k-particionamiento deseado.

El algoritmo CBSGC arriba descrito, se traduce en los siguientes pasos:

1. Creación de las matrices A y B.

   Matriz A: Cada elemento A_{ij} indica la correlación entre el documento d_j y la categoría c_i. Si el documento d_j pertenece a *k* categorías, los valores serán $\frac{1}{k}$, y los otros elementos de la columna jth de la matriz serán 0.

   Matriz B: Cada elemento B_{ij} representará el peso de la arista E_{ij} que se corresponde a la ocurrencia de los términos, y si no existe valdrá 0.

2. Creación de las matrices normalizadas de *A* y *B*: $\hat{A}$ y $\hat{B}$. Para ello, antes es necesario crear las matrices $P_{A,}P_B, R_A\ y\ R_B$

   Matriz $\boldsymbol{P_A}$: Matriz diagonal donde $P_{A_{ii}} = \sum_j A_{ij}$

   Matriz $\boldsymbol{P_B}$: Matriz diagonal donde $P_{B_{ii}} = \sum_j B_{ij}$

   Matriz $\boldsymbol{R_A}$: Matriz diagonal donde $R_{A_{ii}} = \sum_j A_{ji}$

Matriz $\boldsymbol{R_B}$: Matriz diagonal donde $R_{B_{ii}} = \sum_j B_{ji}$

Matriz $\widehat{\boldsymbol{A}}$: $P_A^{-1/2} A \, R_A^{-1/2}$

Matriz $\widehat{\boldsymbol{B}}$: $P_B^{-1/2} B \, R_B^{-1/2}$

3. Aplicar GSVD a $\hat{A}$ y $\hat{B}$, para obtener las matrices *U*, *X*, *V*, *C* y *S*
4. Para poder utilizar de una forma útil la información de las matrices que devuelve GSVD hay que realizar una transformación lineal adecuada. Para ello creamos la matriz *H*

   Matriz $\boldsymbol{H} = C \, X^T X \, S$
5. Una vez creada la matriz *H*, aplicamos SVD sobre ella para obtener las matrices U_H y V_H
6. Con estas dos nuevas matrices U_H y V_H realizamos una transformación lineal para refinar *U* y *V* , creando U^* y V^*

   Matriz $\boldsymbol{U}^*$ = $U \, U_H$

   Matriz $\boldsymbol{V}^* = V \, V_H$
7. Para realizar el k-particionamiento, seleccionaremos $l = \lceil \log_2 k \rceil$ vectores columna de las matrices U^* y V^*, $u_v = (u_2, u_3, \ldots, u_{l+1})$ y $v_v = (v_2, v_3, \ldots, v_{l+1})$ para formar los vectores normalizados integrados.

   $$\boldsymbol{\Omega_U} = \begin{bmatrix} P_A^{-1/2} \, u_v \\ R_B^{-1/2} \, v_v \end{bmatrix}$$
8. En el último paso, realizaremos el clustering de $\Omega_U[1]$ y de Ω_U [2] para conseguir el k-particionamiento de categorías y términos, aunque para el propósito de nuestro algoritmo, el único que utilizaremos será el de categorías.

Una vez creado el algoritmo, para construir la taxonomía jerárquica, hay que repetir todo el procedimiento, esta vez seleccionando los nuevos clusters como las clases de nivel superior , en lugar de todas las categorías, hasta que el valor de k sea igual a uno, es decir, que sea una hoja.

2.2 Corpus de Datos

El corpus de datos que vamos a utilizar está formado por un subconjunto de documentación matemática perteneciente a opiniones y resúmenes de artículos, provenientes de las bases de datos del ZentralBlatt Math del Fiz Karhsruhe y del Get-Info Portal del TIB de Hannover.

Para trabajar con este tipo de documentos, tenemos acceso a una serie de ficheros, los cuales están formados por reseñas de documentos matemáticos. Cada uno de estos documentos está catalogado dentro de una serie de categorías pertenecientes a la codificación MSC, y las reseñas de cada uno de ellos, formarán los términos de cada documento.

Tenemos acceso a tres tipos diferentes de ficheros (para información más detallada, consultar Anexo D): el primero con las reseñas en formato texto, el segundo con los términos de las reseñas codificados numéricamente y con su número de ocurrencias y la tercera igual que la anterior, pero con las ocurrencias por término normalizadas. Va a ser este tercer fichero el que vamos a usar para desarrollar el algoritmo, ya que la información que aporta, es la que mejor se adapta a este, y los resultados que se obtendrán, al estar normalizados los términos, serán más fiables.

En este fichero, cada línea del mismo es una estructura JSON que representa un documento. Esta estructura consta de dos partes, en la primera encontramos la *metadata*, compuesta por un identificador de documento y el identificador de las categorías a las que pertenece en formato MSC, y en la segunda, el contenido del documento, representado por identificadores de términos y sus ocurrencias.

En la figura 5 podemos ver un ejemplo de un documento que podemos encontrar en el fichero. Como podemos ver está compuesto por un identificador de documento, un conjunto de categorías en codificación MSC y una lista de términos con sus ocurrencias normalizadas.

```
[
    [
        "1039.35023",%Identificador de documento
        ["35B45","35B65","35J25"] %categorias a las que pertenece

    ],
    {
        "8":0.15249857033260467, % Identificador término
                                 % Ocurrencias normalizadas
        "1765":0.15249857033260467,"1850":0.15249857033260467,"3108
        ":0.15249857033260467,"3309":0.15249857033260467,"3370":0.1
        5249857033260467,"5161":0.15249857033260467,"5866":0.152498
        57033260467,"7805":0.15249857033260467,"8492":0.15249857033
        260467,"8799":0.15249857033260467,"9225":0.1524985703326046
        7,"10970":0.15249857033260467,"11188":0.15249857033260467,"
        11566":0.15249857033260467,"11583":0.15249857033260467,"129
        15":0.15249857033260467,"12964":0.15249857033260467,"13893"
        :0.15249857033260467,"15191":0.15249857033260467,"17269":0.
        15249857033260467,"17361":0.457495710997814,"23037":0.15249
        85703326047,"23096":0.15249857033260467,"24444":0.15249857
        033260467,"24807":0.30499714066520933,"24959":0.15249857033
        260467,"27119":0.15249857033260467,"28302":0.15249857033260
        467,"30075":0.15249857033260467,"30077":0.1524985703326046
        "31345":0.15249857033260467
    }
    % Lista de términos con sus ocurrencias normalizadas

]
```

Figura 5: Ejemplo de documento matemático

3. DISEÑO

En este tercer capítulo se explica el diseño que se ha establecido, partiendo del análisis que se ha realizado en el punto anterior sobre el paper matemático utilizado y sobre los ficheros que contienen el corpus de datos, para poder implementar el algoritmo CBSGC. Este algoritmo sirve para realizar la categorización de documentos, y para evaluar y analizar según diferentes si se pueden obtener mejores resultados si introducimos superclases en la taxonomía como un primer nivel de clasificación, para profundizar después en la jerarquía.

Este capítulo consta de cuatro secciones: en la primera se abordará el diseño del corpus de datos, en la segunda las variables y vectores necesarios para crear las matrices del tercer apartado, y finalmente, el k-particionamiento.

3.1 Corpus de Datos

Cada línea de los ficheros de datos que forman el corpus de datos está en formato JSON y presenta la siguiente estructura:

```
[["1039.35023", ["35B45","35B65","35J25"]], {"8":0.15249857033260467,…}]
```

Para poder trabajar con este contenido se ha diseñado una estructura que engloba la información. Cada documento estará formado por su identificador, las categorías a las que pertenece y una lista de términos.

Con esta información podremos acceder a toda la información que aporta el fichero, para poder luego acceder a información concreta que servirá para crear las matrices A y B.

Estructura documento {
    Identificador *cadena*;
    Categorías *vector de cadenas*;
    Términos: *vector de estructuras término: ocurrencia*
}
Estructura términos {
    Identificador término: *cadena*
    Ocurrencia: *float*
{

Figura 6: Estructura de los datos del fichero

3.2 Variables y Vectores

Para la creación de las matrices A y B, necesarias para la ejecución de este método, en primer lugar hemos construido una serie de vectores, los cuales nos van a servir, además de para simplificar la creación de las matrices, para tener un acceso más sencillo y rápido a toda la información que contiene el corpus de datos. Además, se han definido una serie de variables, las cuales guardan el número de documentos, categorías y términos.

Variables

Las cinco variables que se muestran en la siguiente tabla, son los valores a partir de los cuales se crean los vectores. Están directamente relacionadas con los documentos, categorías y términos. Para cada variable mostramos su descripción y de qué tipo es. Los valores que se muestran son los pertenecientes al corpus de datos completo.

		DESCRIPCIÓN	Tipo
VARIABLES	**NCAT**	Número de categorías sin repetición	*tipo* :entero
	NDOC	Número de documentos totales	*tipo*:entero
	NTER	Número de términos sin repetición	*tipo*:entero
	NTCAT	Número de categorías con repetición	*tipo*:entero
	NTTER	Número de términos con repetición	*tipo*:entero

Tabla 1: Descripción de las variables

Debido a que no se pueden completar los cálculos con el corpus de datos original por falta de memoria en nuestro ordenador (ver apartado *Implementación*), se han creado nuevas variables, las cuales reflejan subconjuntos del corpus de datos original.

Como consecuencia de la falta de memoria, para poder realizar las pruebas no se ha simplificado el número de documentos, por ejemplo reduciéndolo a la mitad o a una cuarta parte, lo cuál sería la opción más rápida aunque con ello se vería afectada la creación de nuestra taxonomía, ya que estaríamos obviando la información que nos aportan el resto de documentos y además, no estaríamos eligiendo un subconjunto homogéneo. La solución que se va a adoptar es seleccionar para cada categoría que queramos analizar, un número N de documentos de nuestro corpus, de esta manera conseguiremos que la información aportada sea lo más homogénea posible para todas las categorías. Las nuevas variables, junto a su descripción y tipo, se muestran en la siguiente tabla.

		DESCRIPCIÓN	Tipo
VARIABLES	**NCATINDEX**	Número de categorías sin repetición acotadas según el criterio que utilicemos	*tipo*:entero
	NDOCINDEX	Número de documentos que aparecen relacionadas con las categorías acotadas.	*tipo*:entero
	NTERINDEX	Número de términos sin repetición relacionados con los documentos acotados.	*tipo*:entero
	NCATOTALINDEX	Numero de categorías con repetición acotadas relacionadas con los documentos acotados.	*tipo*:entero
	NTTERINDEX	Número de términos con repetición relacionados con los documentos acotados	*tipo*:entero

Tabla 2: Descripción de las nuevas variables

Vectores

Para la creación de las matrices A y B, se han diseñado ocho vectores, los cuales nos permitirán que el acceso a toda la información contenida en el corpus de datos en formato JSON pueda ser accedida de una forma rápida y eficaz, sin necesidad de tener que recorrer una y otra vez el fichero para poder acceder a los datos. Como se puede observar, los vectores tendrán dimensiones acotadas por los valores de las variables que acabamos de comentar en el apartado anterior.

		DESCRIPCIÓN	Tipo y Tamaño
VECTORES	**VC**	Vector formado por los identificadores de todas las categorías ordenadas sin repetición por orden de aparición	*tamaño*: <1xncat> *tipo:* celda
	VD	Vector formado por los identificadores de todos los documentos ordenados por orden de aparición	*tamaño* <1xndoc> *tipo:* celda
	VT	Vector formado por los identificadores de todos los términos ordenados de menor a mayor sin repetición	*tamaño*: <1xnter> tipo: doble
	VCC	Vector formado por categorías por documento, ordenados según el orden de aparición	*tamaño* <1xndoc> *tipo:* celda
	VCD	Vector formado por el número de categorías por documento, ordenados según el orden de aparición	*tamaño* <1xndoc> *tipo:* celda
	VTTD	Vector formado por términos por documento, ordenados según el orden de aparición	*tamaño* <1xndoc> *tipo:* celda
	VTD	Vector formado por el número de términos por documento, ordenados según el orden de aparición	*tamaño* <1xndoc> *tipo:* celda
	VTT	Vector formado por los identificadores de todos los términos ordenados por orden de aparición , con repetición	*tamaño* <1xntter> *tipo:* celda

Tabla 3: Descripción de los vectores

Debido al problema de falta de memoria para poder completar los cálculos con las matrices, se han tenido que diseñar nuevos vectores. En la tabla 4 se muestra la descripción de los dos vectores que nos van a permitir acotar el número de términos con el que vamos a trabajar, seleccionando los que sean más significativos es decir, a acotando a partir de un número dado de ocurrencias, y en la tabla 5, se puede ver la descripción de los vectores que van a permitir trabajar subconjuntos del corpus, bien sea grupos definidos, seleccionando categorías del primer nivel o trabajando con categorías pertenecientes al mismo grupo de codificación MSC de primer nivel.

Existen investigaciones para seleccionar los términos más significativos de un corpus, y se ha investigado para este proyecto sobre eso, pero al tratar con tan pocos datos, no se consideró relevante seguir adelante con la idea.

	DESCRIPCIÓN	**Tipo y Tamaño**
VOCTER	Vector compuesto por el número de ocurrencias de cada término que aparecen	*tamaño*: <1xnter> *tipo:* celda
VTERCOC	Vector en el cual se acota el vector "vocter" delimitando el número de repeticiones por término, según un valor mínimo y máximo.	*tamaño*: indef *tipo:* celda

Tabla 4: Descripción de los vectores para acotar términos

		DESCRIPCIÓN	**Tipo y Tamaño**
VECTORES	**VID**	Vector formado por los identificadores de los documentos que contienen las categorías con las que se va a trabajar	*tamaño*: <1xndocindex> *tipo:* doble
	VCINDEX	Vector formado por las categorías del corpus de datos seleccionadas	*tamaño* <1xncatindex> *tipo:* celda
	VCCINDEX	Vector formado por las categorías por documento, de los documentos que se han seleccionado en el vector vid	*tamaño*: <1xndoc> tipo: celda
	VCDINDEX	Vector formado por el número de categorías por documento, de los documentos que se han seleccionado en el vector vid	*tamaño* <1xndoc> *tipo:* celda
	VTTINDEX	Vector formado por los identificadores de todos los términos ordenados por orden de aparición , con repetición, que aparecen en los documentos que han sido seleccionados	*tamaño* <1xntterindex> *tipo:* celda

Tabla 5: Descripción de los nuevos vectores para trabajar con subconjuntos del corpus

3.3 Matrices

Este apartado se centra en cómo se han diseñado las matrices del algoritmo CBSGC. Las matrices A y B son la base del mismo, por lo tanto su diseño es una de las partes más importantes del proceso. Para el diseño de ambas, vamos a hacer uso de las variables y de los vectores de datos que se han explicado en el apartado anterior; el resto de las matrices, se construirán principalmente, a partir de estas dos.

En la tabla 6 se muestran las matrices que se han diseñado junto a su descripción, los vectores, variables y matrices que intervendrán para su diseño, y los resultados que se obtendrán al procesar cada matriz. La matriz A al depender de documentos y categorías, hace uso principalmente de vectores relacionados con esas variables, y lo mismo ocurre con la matriz B, aunque en ese caso será relacionado con documentos y términos.

En la columna "Resultados" cada matriz o vector resultado, se representará con su nombre y dimensiones. Para definir el tamaño, los documentos se representan por *m*, las categorías por *n* y los términos por *t* Debido a la gran cantidad de datos con las que se trabaja, en lugar de trabajar con matrices completas, se va a trabajar con matrices dispersas, donde los vectores I, J y D, representan los vectores fila, columna y datos de su matriz dispersa asociada, las dimensiones de los mismos dependerán del volumen de datos de cada matriz, este valor lo representaremos con una *sx*.

		DESCRIPCIÓN	Entrada	Resultados
MATRICES	**Matriz A**	Matriz que representa el coclustering basado en el grafo bipartito documento - término	vid, vc(vcindex), vcd(vcdindex), vcc(vccindex),ncat, ncattotal,ndocs	A(nxm) SparseA(nxm) IA(1x sa) JA(1x sa) DA(1x sa)
	Matriz PA	Matriz diagonal, donde cada elemento es el sumatorio de la fila correspondiente de A	IA, DA, ncat	PA(nxn) SparsePA(nxn) IPA(1xn) DPA(1xn) JPA(1xn)
	Matriz RA	Matriz diagonal, donde cada elemento es el sumatorio de la columna correspondiente de A	JA, DA, ndocs	RA(mxm) SparseRA(mxm) IRA(1xm) DRA(1xm) JRA(1xm)
	Matriz B	Matriz que representa el coclustering basado en el grafo bipartito categoría - documento	vid,vtd,vttd, ndocs,vtercoc,vocter	B(mxt) SparseA(mxt) IB(1x sb) JB(1x sb) DB(1x sb)
	Matriz PB	Matriz diagonal, donde cada elemento es el sumatorio de la fila correspondiente de B	IB, DB, ndocs	PB(mxm) SparsePB(mxm) IPB(1xm) DPB(1xm) JPA(1xm)
	Matriz RB	Matriz diagonal, donde cada elemento es el sumatorio de la columna correspondiente de B	JB, DB, vtercoc	RB(txt) SparseRB(txt) IRA(1xt) DRA(1xt) JRA(1xt)
	Matriz $\widehat{A}$	Matriz A normalizada	SparseRA,SparseA, IPA,JPA,DPA	AA(nxm)
	Matriz $\widehat{B}$	Matriz B normalizada	SparseB, IPB,JPB,DPB,IRB,JRB,DRB	BB(mxt)
	Matriz U,X,V,C y S (GSVD)	Aplicamos Generalized Singular value Decomposition sobre las matrices normalizas de A y B.	AA, BB	U(mxm) X(nxn) V(txt) C(mxm) S(nxn)
	Matriz H	Matriz H es el resultado de aplicar transformación lineal sobre C X y S	C, X, S	H(mxt)
	Matriz (U,S y V) SVD	Aplicamos Singular Value Decomposition sobre la matriz H.	H	UH(mxm) SH(mxt) VH(txt)
	Matriz U* y V*	Refinamiento de U y V	U, UH, V , VH	US(mxm) VS(txt)

Tabla 6: Descripción de las matrices

3.4 K-particionamiento

Los vectores y matrices que se han diseñado hasta el momento son el medio para llegar al último paso del algoritmo, el k-particionamiento, con el cual conseguiremos categorizar las categorías de los documentos del corpus de datos.

Determinar el valor de *k* en los problemas de clustering es un problema complicado, existen muchos algoritmos e investigaciones al respecto que intentar conseguir buenos resultados. En este proyecto se propone como solución la estrategia de enumerar *k*. Se va a testear con diferentes valores de *k* el clustering, desde *k* igual a 1 hasta el número total de categorías del conjunto del corpus que queramos analizar, para crear una curva *Je-k* (siendo *Je* el valor mínimo de la función objetivo del algoritmo de clustering).

Para realizar el clustering, vamos a utilizar *k-means*, método de agrupamiento que tiene como objetivo la partición de un conjunto *n* en *k* grupos. En nuestro problema, dado un conjunto de datos, que será el vector normalizado que integrará las categorías, se obtendrá como resultado el particionamiento de estas y los centroides de los clusters. Con esta información, podremos calcular los distintos valores de *k* y elegir el mejor, utilizando la fórmula *intra-cluster distance measure :*

$$Je = \min \sum_{i=1}^{k} \sum_{y \in \Gamma_i} \|y - \sigma_i\|^2$$

donde $\sigma_1, \sigma_2, \dots, \sigma_k$ son los centros de los clústeres $\Gamma_1, \Gamma_2, \dots, \Gamma_k$. El valor de k será el punto de inflexión de la curva.

Para calcular el punto de inflexión de la curva, la cual estará compuesta por todos los valores *Je* pertenecientes a cada una de las categorías, se aproximará la misma a una gráfica conocida, y se seleccionará el punto, que será el valor óptimo de *k*. Debido a que no se ha podido computar el algoritmo con suficientes datos, las gráficas que se han obtenido no muestran una estructura fija, en la cual se pueda seleccionar el punto de inflexión, lo cual hace que el algoritmo no pueda ser ejecutado en su totalidad de forma recursiva automática . Por ello, se han diseñado un grupo de funciones que permiten seleccionar una serie de puntos candidatos a ser el valor de *k*, y testearlos para decidir cuál es el mejor; en el apartado de *Implementación* se explicará en profundidad cómo se ha llevado a cabo esto. Una vez que seleccionado el valor de *k*, el siguiente paso es clusterizar para obtener el k-particionamiento de categorías.

Para crear la taxonomía jerárquica, hay que repetir el algoritmo recursivamente, hasta que cada subconjuntos este formado por una sola categoría. En la siguiente figura se muestra el diagrama de flujo de la construcción de la taxonomía, nuevamente en el apartado *Implementaciones* se abordara este tema.

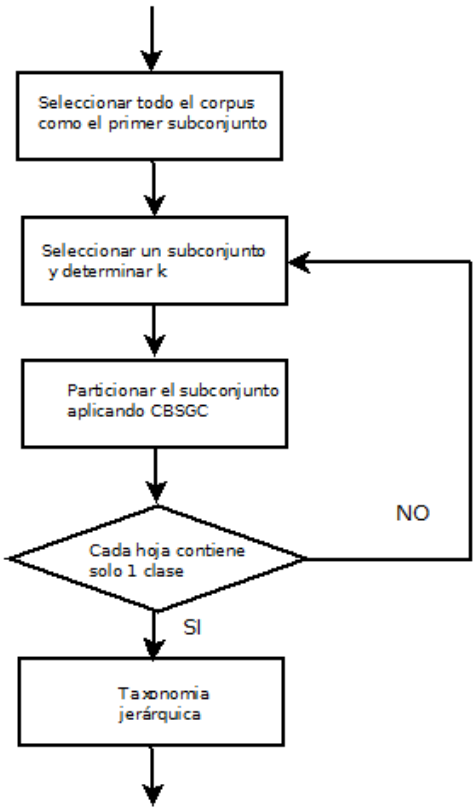


Figura 7: Diagrama de flujo de la construcción de la jerarquía

4. IMPLEMENTACION

Este cuarto capítulo aborda la etapa de implementación del proyecto; ha sido la etapa más complicada y duradera, nos hemos enfrentado a problemas de falta de memoria por parte de nuestro ordenador y a un cambio en el lenguaje de programación, ambos problemas, además de retrasar el proyecto considerablemente, han ocasionado que se haya tenido que dar un nuevo enfoque a la etapa de diseño, y que uno de los objetivos del proyecto, la creación de superclases sobre la taxonomía definida, no se haya podido llevar a cabo.

Este capítulo consta de cuatros secciones: en la primera se tratará el lenguaje de programación, en la segunda la lectura del fichero del corpus de datos, en la tercera la implementación del algoritmo, lo cual incluye vectores, matrices y la parte de clustering, junto a la creación de la curva Je-k y finalmente, la creación de la taxonomía.

4.1 Lenguaje de Programación

En un primer momento, este proyecto final de carrera iba a ser implementado en su totalidad en Python, porque es el lenguaje utilizado para implementar los proyectos del proyecto $DeLiVer^{Math}$ y además, porque tiene características que lo hacen adecuado para este tipo de problema, como lo es el trabajo con matrices y las librerías que posee para clasificación y minería de datos, como Orange y NumPy, o la existencia de la distribución Enthought.

El problema con Python es que no tiene implementada la función GSVD, la cual es indispensable para poder realizar el algoritmo. Para solucionar este contratiempo, se intentaron diferentes alternativas, con la finalidad de conseguir usar Python para desarrollar el proyecto: usar "mlabwrap", que es un *bridge* de Python a Matlab, que permite que Python vea a Matlab como una librería, ya que Matlab sí que tiene la función GSVD; esta solución no se pudo llevar a cabo ya que el wrapper está obsoleto, y no se consiguió que funcionara con las nuevas versiones de Python; y la segunda solución que se propuso fue acceder a librerías LAPACK, para que Python pudiera utilizar la función GSVD de allí, pero estaba también obsoleta.

Debido a estos inconvenientes, se decidió que el proyecto fuera realizado en su totalidad en Matlab, ya que posee en sus librerías las funciones SVD y GSVD, y además tiene funciones de clasificación y minería de datos, y está preparado para trabajar con matrices.

4.2 Lectura del fichero

Para leer el fichero, se ha utilizado la función *parse_json()* . Con esta función función se va a leer cada línea del fichero y se transformará el contenido de las estructuras JSON de nuestro fichero normalizado, a estructuras para poder trabajar en MATLAB.

```
nfile = 'exp-train_proj-abs_vec-conf9-min-3_norm';
fid = fopen(nfile,'r');
InputText=textscan(fid,'%s',1,'delimiter','\n',);
mat = parse_json(InputText{1}{1});
```

En *mat* tendremos la estructura JSON guardada en dos celdas, una para el identificador y otra para las categorías, y una estructura, que contendrá los términos y sus ocurrencias normalizadas. Esta estructura es la base para la construcción de los vectores y matrices necesarios para la implementación del algoritmo y la construcción de la taxonomía jerárquica.

```
mat =

      {1x2 cell}    [1x1 struct]
```

Para accede al identificador del documento

```
mat{1}{1} =

       1039.35023
```

Para acceder a sus categorías

```
mat{1}{2} =

      '35B45'    '35B65'    '35J25'
```

Para acceder a sus términos

```
mat{2} =

      alpha_8: 0.1525
      alpha_1765: 0.1525
      alpha_1850: 0.1525
      alpha_3108: 0.1525 …
```

Se puede observar que los identificadores de los términos, a diferencia de cómo aparecen en el corpus de datos (identificador numérico) aquí aparecen representados por el string “alpha” seguido de un valor numérico. Esto es así ya que en Matlab no se pueden declarar identificadores numéricos, por lo que se ha optado por añadir un string constante a todos los términos para su declaración. Para acceder al valor real se ha implementado la función alpha2num.m, la cual dado un string “alpha_x” devuelve el integer “x”

4.3 Algoritmo

En este apartado se va a abordar la implementación del algoritmo. Se ha divido en tres apartados: implementaciones de los vectores, de las matrices y por último, el k-particionamiento de categorías.

4.3.1 Vectores

En este apartado se va a describir cómo se han implementado las funciones para la creación de los vectores. La tabla 7 muestra todas las funciones relacionadas con vectores que permiten el acceso a la información del fichero, junto a su descripción, los parámetros que tienen de entrada y los resultados que generan.

En la tabla existen dos tipos de funciones: las que sirven para obtener datos intermedios, las cinco primeras funciones que aparecen en la tabla y las que crean los vectores base para poder crear las matrices A y B, todas las que empieza su identificador con *v*. Como se puede observar hay funciones muy similares, se diferencian solamente en alguno de sus parámetros; esta solución de diseño se ha adoptado debido a la falta de memoria de nuestro ordenador y debido a las limitaciones que presenta la función GSVD, puesto que no acepta como entrada matrices dispersas, convirtiéndose así en nuestro cuello de botella. Este problema de memoria hace que no hayamos podido trabajar con el corpus de datos completo, restringe tanto el tamaño de las matrices con las que se puede trabajar, que los resultados que podemos obtener con el corpus de datos que acepta hace que los resultados no sean concluyentes, y que se hayan tenido que buscar otras alternativas para la categorización de los documentos.

Dada esta limitación, en lugar de trabajar con el fichero entero, se han propuesto tres soluciones diferentes para poder sacar resultados: trabajar con todas las categorías pero seleccionando solo su primer nivel, seleccionar categorías pertenecientes al mismo primer nivel, es decir, un subconjunto de categorías que empiecen mismo código y por último, con categorías de un subconjunto concreto. Con estas soluciones, se pretende analizar el algoritmo y la estructura de la codificación MSC, y ver si podría funcionar en futuras investigaciones para ahondar más en el tema, ya que tras la finalización de este proyecto, no se ha llegado a resultados concluyentes.

Para llevar a cabo estas nuevas soluciones, se han tenido que modificar las funciones diseñadas en un primer momento, para adaptarlas a las nuevas necesidades. La función más importante que se ha añadido, es la que crea el vector *vid*, encargada de seleccionar un número N de documentos que pertenecen a las categorías seleccionadas, creando una indexación de categorías con documentos. Con esta función aparte de conseguir terminar la ejecución del

algoritmo, y por lo tanto resultados, hacemos que cada categoría que queramos categorizar, tenga el mismo número de repeticiones en el corpus, y de esta manera asegurar, que los resultados obtenidos sean los más homogéneos posibles.

	DESCRIPCIÓN	Input	Output
countcatdoc1()	Cuenta las categorías de un documento dada la posición del documento	posdoc, ndocs	ncatdoc
countcatdoc2()	Cuenta las categorías de un documento dado el identificador del documento	iddoc, vd	ncatdoc
countdocs()	Cuenta el número de documentos del fichero	nfile	ndocs
poscat()	Dada una categoría y el vector con todas las categorías, devuelve su posición	categoría, vc	poscat
iddoc()	Dado una posición del fichero, devuelve su identificador	nfile, pos	iddoc
vmaxtermvtt()	Devuelve el término máximo y mínimo del fichero, y cuenta el número de términos totales	nfile, ndocs, vid	nter, ntter
vcat()	Crea un vector con las categorías, cuenta las categorías diferentes y las totales. Dependiendo de los parámetros s y c, selecciona las categorías con 5 dígitos o con 2.	nfile, s, ca, categoría	vc,ncat ntcat
vcat1doc()	Crea un vector con el número de categorías por documento	nfile, ndocs	vcd
vccategory()	Crea vectores de número de categorías y categorías por documento, dependiendo de la categoría que se quiera utilizar	vcc, categoría	vccindex vcdindex
vccategorydef()	Crea vectores con el número de categorías y categorías por documento, para un subconjunto de categorías dado.	vcc, vc	vccindex vcdindex
vctindex()	Cuenta el número total de categorías pertenecientes a un subconjunto dado	vid, vccindex, ndocs	ncattotalindex
vdocs()	Crea un vector con los identificadores de todos los documentos	nfile, ndocs	vd
vectvtt()	Crea un vector con todos los términos que aparecen en el fichero	nfile,vid, ndocs	vtt
vindexdoc()	Crea un vector con los identificadores de los documentos que pertenecen a un conjunto de categorías que pertenecen al mismo primer nivel. Se seleccionan un número de documentos por categoría según el valor de maxDC.	vc,vcc, maxDC, ca,vtd	vid
vindexdocdef()	Crea un vector con los identificadores de los documentos que pertenecen a un conjunto de categorías pertenecientes a un subconjunto dado. Se seleccionan un número de documentos por categoría según el valor de maxDC.	vc,vcc, maxDC, vtd	vid
vter1doc()	Crea un vector con el numero de términos por documento	nfile, ndocs	vtd
vterms()	Crea un vector con el número de términos totales	nter	vt

vocurrencesterm()	Crea dos vectores, uno con las ocurrencias de cada término, y el otro con los términos cuya ocurrencia están en un intervalo dado	nter, vtt, min max	vocter, vtercoc

Tabla 7: Funciones para implementar los vectores

4.3.2 Matrices

En este apartado se va a describir cómo se han implementado las matrices necesarias parar realizar la categorización de documentos; en el apartado de *Diseño* se han presentado 19 matrices diferentes, en cambio en este apartado simplemente nos vamos a centrar en la creación de la matriz A y B, y en la normalización de ambas, ya que el resto, aparte de basarse en estas, su implementación simplemente se basa en aplicarles transformaciones lineales, o aplicarles la función GSVD o SVD.

Para la creación de las matrices A y B se han tenido que recorren los diferentes vectores de categorías, documentos y términos, que hemos implementado, para ir completándolas, para ver su definición consultar apartado *Pasos del Algoritmo*. Debido a que son matrices de gran tamaño, rellenarlas enteras no es factible, por lo que se ha optado por crear matrices dispersas, para ello en lugar de rellenar todas las posiciones, hemos tenido que crear tres vectores, uno para los índices, otro para las columnas y el último para los datos; vectores I , J y D.

Debido al cambio de diseño, por problemas de falta de memoria tanto en relación al número de documentos, de categorías y de términos, las matrices de tamaño máximo que permite nuestro ordenador tratar son de apenas diez mil datos por diez mil, si comparamos esto con el fichero de más de cien mil datos, o con los casi cuarenta mil términos, vemos que va a ser imposible complementar los cálculos. Esta gran limitación se ha visto reflejado a la hora de crear las matrices, ya que no podemos utilizar toda la información que hay almacenada en los vectores representando el corpus, sino que hemos tenido que ir seleccionando la información que cumple ciertas características, por ejemplo, términos con un número mayor de ocurrencias, o documentos que posean unas ciertas categorías.

Para la implementación de las matrices diagonales $P_A, P_B,\ R_A\ y\ R_B$, se ha necesitado primero tener implementadas las matrices A y B, ya que se basan en la información que contienen estas. Al igual que con las matrices A y B, se han tenido que crear matrices dispersas de estas cuatro matrices, debido a sus dimensiones.

Para ahorrar memoria se han definido los tamaños de los vectores que forman las matrices dispersas, declarándolos al principio de la función, ya que Matlab al crear vectores que crecen de

forma dinámica, el aprovechamiento que hace de memoria no es bueno, y ralentiza mucho el proceso.

Una vez creadas estas seis matrices, el resto de las matrices $\hat{A}$, $\hat{B}$, U, V, X, S, C, H, UH, VH, SH, US y VS, se crean a partir de estas. La definición de cada una de ellas se puede ver en el apartado *Análisis del Problema*, en el punto *Pasos del Algoritmo*, su implementación es inmediata puesto que o bien la transformación lineal consiste en multiplicar matrices, o bien aplicar a una o varias matrices una función predefinida de Matlab (GSVD o SVD)

Uno de los mayores contratiempos de este proyecto ha sido el tiempo invertido en la creación de las matrices, debido a las dimensiones de estas y al depender cada una de tantas variables hace que no se pueda paralelizar su creación , hacen que los tiempos medios para la creación rondara para cada prueba de tres a cuatro horas. Esto sumado a la limitación de la memoria, ha hecho que no se puedan realizar trabajos en paralelo, incrementando todavía más el periodo de pruebas, además ha habido muchas veces que tras haber creado las matrices iniciales, al aplicar GSVD, ya que este necesita como entradas las matrices en formato completo, se ha producido error de memoria, por lo que se ha tenido que volver a aplicar el algoritmo acotando más la información.

4.3.3 K-particionamiento

En este apartado se va a describir cómo se ha implementado el k-particionamiento de categorías, y cómo se han obtenido e interpretado los resultados.

Este proceso está formado por tres partes: creación y dibujo de la curva *Je-k*, elección de los posibles puntos de inflexión y testeo del clustering con diferentes valores. Para la implementación de la curva *Je-k* , tenemos que aplicar *kmeans* al vector integrado normalizado resultante de aplicar CBSGC. Para esta función hemos elegidos los parámetros siguientes: para el cálculo de la distancia, la *distancia Euclídea* de los puntos; acción por defecto en el caso de que no se pueda realizar clustering con éxito, *singleton*, que crea un nuevo cluster que consiste en el punto más lejando de su centroide; y el parámetro *replicates*, que indica el número de veces que queremos que se repita el clustering, igual a 200. En el anexo E, están las pruebas que se han realizado para seleccionar estos parámetros como los mejores para nuestro problema

```
 [idxa,Ca]  = kmeans(wa,i,'Distance','sqEuclidean','emptyaction',
'singleton','Replicates',200);
```

Una vez que se obtiene el vector *vJe*, que tiene para cada posible valor de *k*, el valor de *Je*, se dibuja la gráfica y se aproxima mediante el comando de Matlab *cftool*

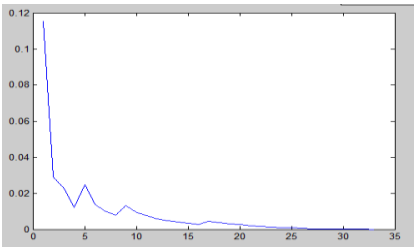


Figura 8: Ejemplo de Curva Je-k

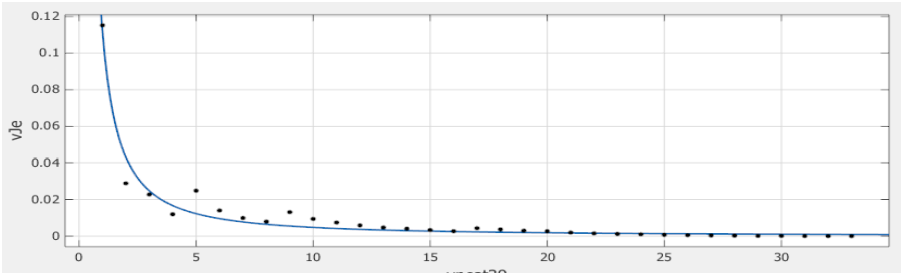


Figura 9: Aproximación de la curva Je-k

Como se puede observar la elección del punto de inflexión no es inmediato, por ello se han implementado las funciones findpeakgraph.m y testingk.m. La primera selecciona los picos de la gráfica devolviendo a qué categoría se refieren y cuál es su valor *Je*, y la segunda para cada valor seleccionado aplica clustering, devolviendo un vector con el clustering resultante de cada posible valor de *k* y las categorías que estarían en cada cluster.

4.4 Taxonomía Jerárquica

En este último apartado se va a tratar la implementación de la taxonomía jerárquica. Una vez que se ha construido el algoritmo, y que se ha seleccionado el valor óptimo de k para realizar el clustering, hay que volver a aplicar el algoritmo hasta crear el árbol completo.

Para ello se han implementado dos funciones nuevas newAAs.m y newvc.m. La primera crea tantas nuevas matrices A y PA como clusters tenemos en el paso anterior, y las normaliza para conseguir matrices $\hat{A}$, y la segunda crea vectores con las nuevas categorías que tendrá cada nueva matriz A.

Para volcar los resultados de la taxonomía se ha creado un procedimiento que rellena un fichero con los resultados que se van obteniendo

```
%--------------------------------------------------------------------
function [ taxonomy ] = createtaxonomy( taxonomy, nivel,k ,C,leaf,vc)
%TAXONOMIA it creates the taxonomy
%           and fill a text file with the information
%--------------------------------------------------------------------
```

A continuación se muestra un ejemplo de fichero resultado:

```
HIERARCHICAL TAXONOMY

Root( 3,C1,C2,C3)
C1( 2,C11,C12)
C11( 2,00,97)
C12( 1,01)
C2(4, C21, C22, C23, C24)
C21(3, 12, 13 14)
C22(4, 64, 65, 73, 82)
```

Figura 10: Ejemplo de fichero resultado

4.4.1 Paralelizar Taxonomía

Hemos optado por la opción de paralelizar la creación de la taxonomía, ya que al ser una estructura jerárquica en la que cada nuevo subárbol no entra en conflicto con la creación de los otros, es la opción más rápida de la que disponemos en contraposición con la opción de crearla de manera secuencial.

Matlab nos ofrece diferentes formas de paralelizar código: el comando *parfor*, que ejecuta un loop de forma paralela, la herramiento *Matlabpool* que habilita clusters para que sean usados en paralelo, y el modo *Interactive Parallel mode (pmode)* que permite trabajar de forma interactiva, con un trabajo desarrollado en paralelo, simultáneamente en varios laboratorios.

Para la creación de nuestra taxonomía vamos a optar por el uso de la herramienta *Matlabpool*. Esta función se usa para reservar un número de "workers" de Matlab, bien para la ejecución de un bucle *parfor* o para paralelizar código (*SPMD* = simple program multiple data). Si *matlabpool* no se está ejecutando, el bucle *parfor* o *spmd* se ejecutarán en serie en el cliente, ya que *matlabpool* es el que inicia la sesión en paralelo y si no se arranca no se ejecutará como tal.

En nuestro código vamos a hacer uso de ambas órdenes: PARFOR que ejecuta un bucle FOR de forma paralela y SPMD que ejecuta regiones de código de forma paralela. Si se pudieran paralelizar todos los bucles del código, es decir cambiar los bucles FOR por bucles PARFOR, éste sería mucho más eficiente y se reduciría el tiempo de computo de forma considerable, pero debido a las dependencias existentes entre las variables, hace que esto no sea posible, y por consiguiente, que la paralelización no se pueda realizar en la gran mayoría de ellos. Este cambio sería positivo, sobre todo, en la creación de los vectores y de las matrices, ya que es ahí donde se pierde más tiempo de cálculo, debido sobre todo al tener que recorrer el fichero del corpus para la creación de los vectores y de recorrer los vectores para la creación de las matrices.

Para la creación de la taxonomía vamos a hacer uso de la paralelización mediante la orden SPMD. Cada vez que ejecutemos el algoritmo CBSGC en un subconjunto de los datos, será una región SPMD la cual se ejecutará de forma simultánea, es decir, iremos creando nuestra taxonomía jerárquica ejecutando en la medida de lo posible, cada nuevo subconjunto de hojas de forma paralela.

Esta limitación viene por el procesador del ordenador que vamos a usar para la creación de la taxonomía. El ordenador que se ha usado es un Packard Bell Easynote TJ66 un procesador Intel Core 2 Duo P8700 (2,53 Ghz, FSB 1066 Mhz, 3 Mb. de cache) con una memoria de 4 Gb. DDR2 a 800 Mhz. Disponemos de 2 núcleos físicos, y 4 núcleos lógicos. Por lo que se podrán realizar cuatro cálculos en paralelos, sin que perdamos eficiencia. Si quisiéramos realizar más operaciones en paralelo, veríamos afectada la velocidad, en lugar de agilizar el proceso, este se vea ralentizado, ya que la capacidad se vería atenuada por un factor de tres.

5. RESULTADOS OBTENIDOS

Una de los mayores inconvenientes de este proyecto ha sido la limitación de memoria, lo que se ha visto directamente reflejado en los resultados que se han obtenido. Al no poder realizarse las pruebas con el corpus al completo, se ha decidido buscar otras alternativas.

Las pruebas realizadas en el paper sobre el que hemos trabajado eran de veintemil mil documentos para categorizar veinte categorías, y nuestro objetivo era categorizar cien mil documentos con seis mil categorías; es evidente que nuestros resultados no iban a ser tan buenos como se esperaría, ya que necesitaríamos por categoría unos diez mil documentos indexados.

Con este proyecto simplemente se va a conseguir una primera aproximación a los objetivos que se pretendían, ya que al no completar los cálculos con el corpus, no se va a poder crear el nivel de superclases esperado.

Se han realizado cuatro experimentos diferentes, cada uno con su número correspondiente de pruebas. Esta parte, quitando la implementación, ha sido la que más tiempo nos ha llevado, ya que no hemos dispuesto de un ordenador preparado para este tipo de cálculos. A continuación se muestra en qué consiste cada experimento, y mostraremos un ejemplo para cada uno que represente los resultados obtenidos.

5.1 Experimento 1

Para este primer experimento se han hecho diversas pruebas, en cada una de ellas se ha seleccionado cien documentos random del fichero principal. En cada uno de estos cien documentos hay cerca de trescientas categorías. Este número elevado de categorías es un problema, ya que no disponemos de suficiente información para que la categorización de documentos sea buena. Se ha hecho la prueba con este número reducido de datos, ya que al incrementar el número de ficheros, el número de categorías se veía también incrementado, además del tiempo de cómputo.

En la figura siguiente se muestran las categorías de un subconjunto de cien ficheros:

'00B15'	'05C70'	'14J28'	'35B65'	'54E35'	'62G07'	'68-01'	'86-06'
'00B25'	'05C75'	'14K22'	'35D05'	'54E40'	'62G08'	'68-06'	'86A17'
'03C35'	'05C80'	'14P15'	'35J25'	'54H05'	'62G10'	'68M99'	'90-06'
'03C50'	'05C83'	'16W30'	'35J85'	'55R12'	'62G20'	'68N99'	'90C30'
'03E35'	'05C85'	'20B25'	'35K50'	'55S10'	'62G30'	'68P20'	'90C40'
'03E72'	'05C90'	'20B27'	'35L70'	'55T15'	'62G32'	'68R10'	'90C55'
'05A05'	'05E05'	'20C30'	'35Q72'	'57-06'	'62H12'	'68T20'	'91A43'
'05A15'	'05E10'	'20E06'	'37-06'	'58-06'	'62H15'	'68T37'	'91B06'
'05A16'	'05E15'	'20F65'	'37A05'	'58D20'	'62H20'	'68U05'	'91B14'
'05A18'	'05E35'	'20G05'	'37A35'	'58J50'	'62H35'	'68U35'	'91C20'
'05A20'	'05E99'	'20M30'	'37A50'	'60C05'	'62J05'	'68U99'	'92C40'
'05B05'	'06A07'	'22C05'	'37B40'	'60E15'	'62K15'	'68W30'	'92C45'
'05B07'	'06A11'	'22E15'	'37E05'	'60F05'	'62L10'	'68W40'	'92D20'
'05B15'	'06A15'	'22E30'	'37F10'	'60F17'	'62M10'	'70-06'	'92D25'
'05B35'	'11E25'	'22E50'	'37N25'	'60G10'	'62M30'	'70Hxx'	'92D40'
'05B40'	'11E39'	'22E60'	'37N99'	'60G35'	'62M99'	'74-06'	'92E10'
'05C07'	'11F70'	'22E65'	'40E99'	'60J05'	'62N01'	'74A15'	'93B05'
'05C10'	'11G15'	'26C10'	'41A21'	'60J65'	'62N02'	'74F10'	'93B12'
'05C15'	'11G50'	'28A05'	'42C05'	'62-02'	'62N05'	'74Nxx'	'93B51'
'05C17'	'12-02'	'28C20'	'43A05'	'62-09'	'62P10'	'81-08'	'93C10'
'05C20'	'12E25'	'30C15'	'43A07'	'62B10'	'62P12'	'81P15'	'93C20'
'05C25'	'12E30'	'30D05'	'43A40'	'62C20'	'62P30'	'81P68'	'93C55'
'05C30'	'12F12'	'32B10'	'43A70'	'62D05'	'62P99'	'81Q05'	'93D20'
'05C35'	'12Lxx'	'33C45'	'43A80'	'62E10'	'65-06'	'81R05'	
'05C38'	'14-06'	'33C90'	'47J15'	'62E15'	'65D18'	'81V55'	
'05C40'	'14G05'	'34C60'	'49J40'	'62E20'	'65M12'	'83C15'	
'05C45'	'14G50'	'34D05'	'52B10'	'62F12'	'65M15'	'83C75'	
'05C69'	'14H45'	'35B45'	'53-06'	'62G05'	'65M60'	'83F05'	

Figura 11: Categorías pertenecientes a un subconjunto de cien documentos random

Aplicamos el algoritmo CBSGC, y creamos la curva *Je-k*, donde se puede observar que no es fácil ni intuitivo seleccionar el punto de inflexión.

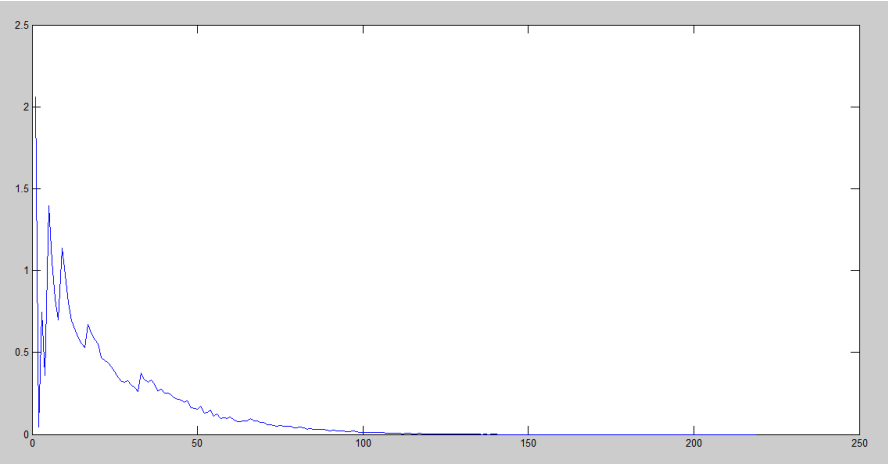


Figura 12: Curva Je-k perteneciente a un subconjunto de cien documentos random

Testeamos k con diferentes valores, y vemos que con valores de 3 y 5 el clustering que realiza es aceptable. En la figura 13 se representa en colores como se han clusterizado las categorías. Podemos observar que con 3 clusters, las categorías que terminan en -06 (Actas, conferencias, coleccciones..) se han agrupado en el mismo cluster. Con 5 clusters podemos observar que nuevamente las categorías que terminan en -06 están agrupadas juntas, y que el resto de las categorías se agrupan en cierta manera, de acuerdo a la división que hace MSC (consultar Anexo C). El hecho que las categorías que terminen en -06 siempre se clustericen aparte, indica que este grupo podía pasar a ser un grupo en el mismo, si consultamos la codificación MSC,

podemos ver que todas las categorías tienen en común el mismo significado para las categorías pertenecientes al grupo – {00,01, 02, 03, 04, 06}

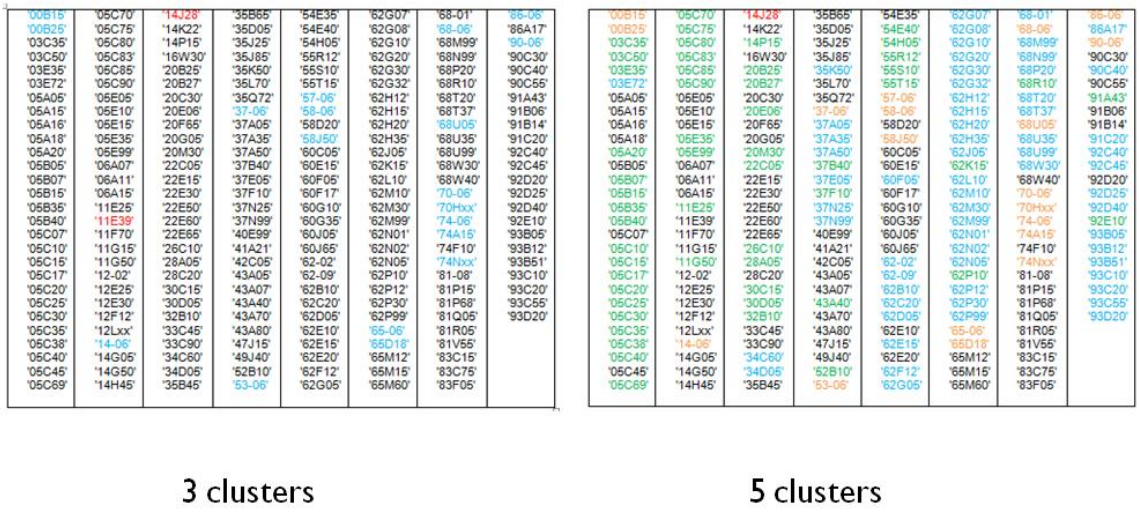


Figura 13: Clustering con k=3 y k=5

En la figura 14 vemos el clustering con k=5, dividido según áreas de MSC. Podemos observar, que aunque el clustering no es óptimo, para la poca cantidad de datos que se han utilizado, los resultados se aproximan bastante a lo que se espera.

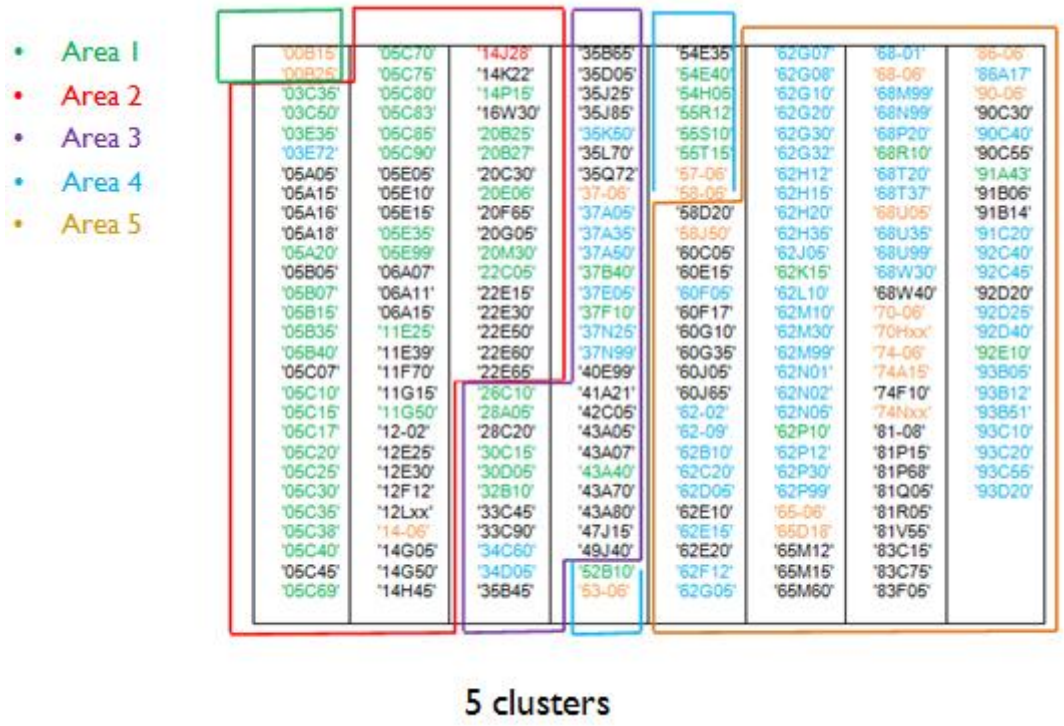


Figura 14: Clustering k=5 dividio por áreas MSC

5.2 Experimento 2

En este segundo experimento, en lugar de trabajar con la codificación MSC completa, sólo se ha trabajado con el primer nivel. Para cada categoría se han seleccionado doscientos documentos, esperando que los resultados se aproximen a la división por áreas realizada por MSC. Se han realizado varias pruebas, seleccionando grupos de documentos random para cada una, a continuación vamos a mostrar y analizar uno de los resultados.

En la siguiente figura se muestra la división por categorías que realizar MSC

'00'	'16'	'33'	'46'	'62'	'85'
'01'	'17'	'34'	'47'	'65'	'86'
'03'	'18'	'35'	'49'	'68'	'90'
'05'	'19'	'37'	'51'	'70'	'91'
'06'	'20'	'39'	'52'	'74'	'92'
'08'	'22'	'40'	'53'	'76'	'93'
'11'	'26'	'41'	'54'	'78'	'94'
'12'	'28'	'42'	'55'	'80'	'97'
'13'	'30'	'43'	'57'	'81'	
'14'	'31'	'44'	'58'	'82'	
'15'	'32'	'45'	'60'	'83'	

Figura 15: División por áreas según MSC

Aplicamos el algoritmo CBSGC, y creamos la curva *Je-k* y la aproximamos a una curva logarítmica, donde podemos ver que nuevamente seleccionar el punto de inflexión no es fácil. Tenemos demasiadas categorías, y pocos documentos por categoría, el algoritmo no es capaz de diferenciar.

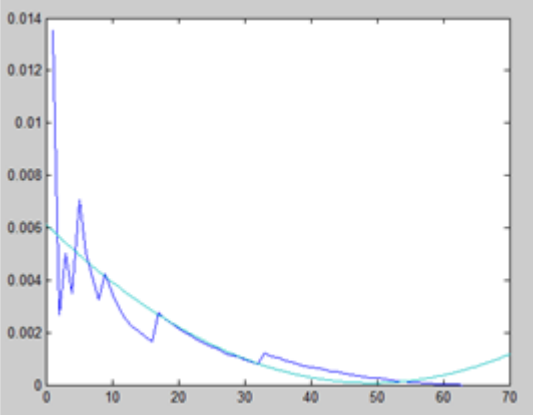


Figura 16: Curva y aproximación de la curva Je-k para el experimento 2

Elegimos como posibles valores de k, k=3 y k=4. El clustering resultante se muestra en la figura siguiente.

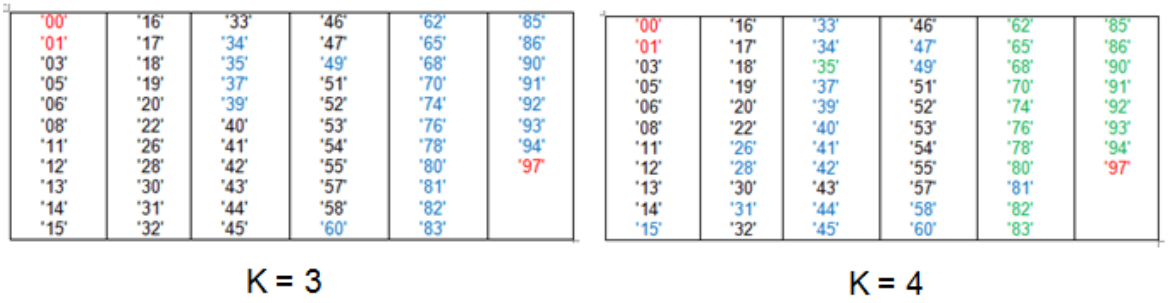


Figura 17: Clustering para k=3 y k=4 para el experimento 2

Podemos ver que con k=3 y k=4, cabe destacar que las categorías 00,01 y 97, pertenecen al mismo cluster, en cambio la categoría 03 no lo está, siendo que 00, 01 y 03 pertenecen al area 1 (General/Fundaciones). Si consultamos en MSC qué tipo de documentos tienen estos grupos vemos que 00 = General, 01 = Historia y Biografía, 03 = Lógica Matemática y Fundaciones, y 97 Educación sobre las matemáticas; la división que hace el algoritmo parece más adecuada, ya que la categoría 97 trata un tema más general como lo es la Educación, y en cambio la categoría 03 podría incluirse dentro del area 2 (Matemática Discreta/Algebra), ya que tiene que ver más con las matemáticas.

Si ahora nos centramos con la división de k= 4, en la figura 18 observamos que la división se asemeja a la división por áreas. Esto es una buena noticia, ya que aunque simplemente hemos utilizado 200 documentos por categoría, la división que hace es similar a la que se hace manualmente con MSC, lo cual cabe indicar, que si utilizáramos más documentos por categorías el clustering mejoraría considerablemente, y se podrían detectar posibles fallos en la clasificación.

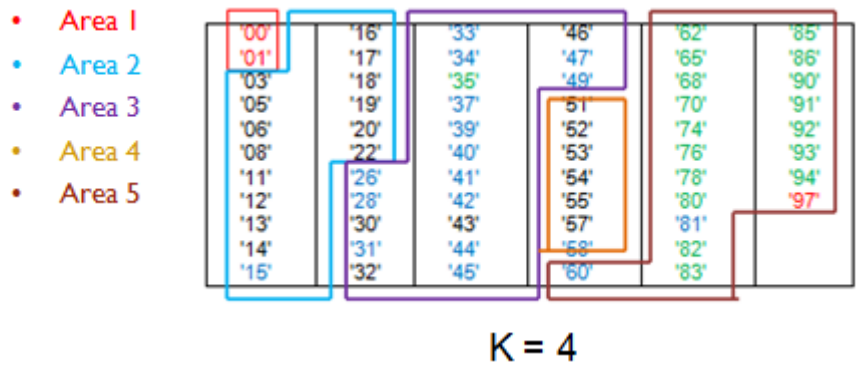


Figura 18: Clustering para k=4 con división por áreas

5.3 Experimento 3

En este tercer experimento, se va a trabajar con categorías que empiezan por el mismo código de primer nivel. Al igual que en el anterior experimento, se van a indexar alrededor de 200 documentos por categoría, dependiendo del número de categorías que tenga cada nivel, ya que si no se producen error por falta de memoria.

Se han hecho pruebas para cada una de las 63 categorías diferentes que posee nuestro corpus, aunque solo vamos a comentar un par que nos han parecido más relevantes, la categoría 39 y la 85.

En la figura 19 podemos ver la tabla con el clustering hecho para las categorías que empiezan por 39, y dos gráficas, Je-k curve y la aproximación de la misma. Vemos que hay tres grupos, por lo que se esperaría que el valor de k=3, en cambioha seleccionado k=4 como el valor óptimo para realizar el clustering y se han dividido las categorías según esta división. Podemos ver que divide las categorías bien para el tercer nivel en el caso de A y B, con algún pequeño error en el caso de B, pero que nuevamente la categoría -06 la clusteriza aparte junto con la terminada en xx (en MSC las categorías que terminan en xx significan que no están en ninguna de las categorías definidas, pero que pertenece al grupo), y que la categoría -02 es un grupo aparte.

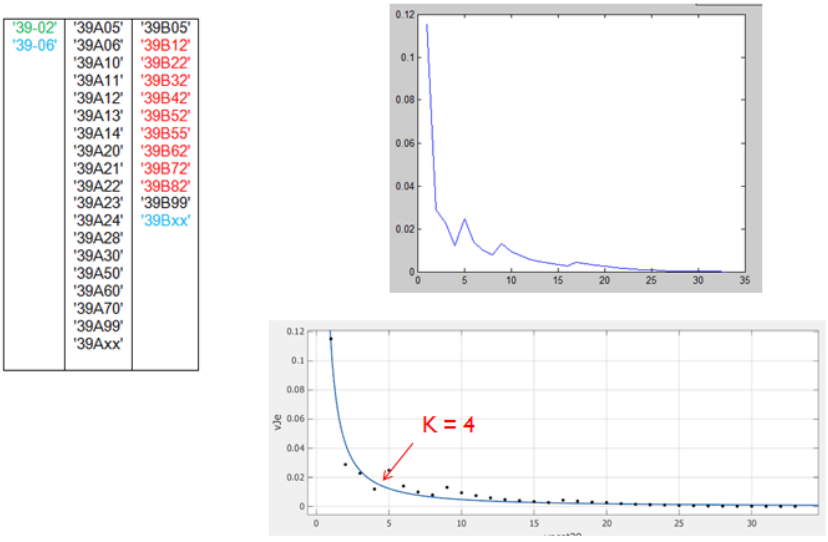


Figura 19: Clustering y grafica Je-k de la categoría 39

En la figura 20 está la tabla con el clustering para las categorías que empiezan por 85, junto a las gráficas Je-k y su aproximación. En esta gráfica se espera que el valor de k=2, y así es, pero al realizar la división de categorías, vemos que agrupa todas juntas excepto a la -06 y a la terminada

el xx. Si probamos con los siguientes valores de k, cada uno de los valores – {00,..,06} lo sitúa en un cluster aparte.

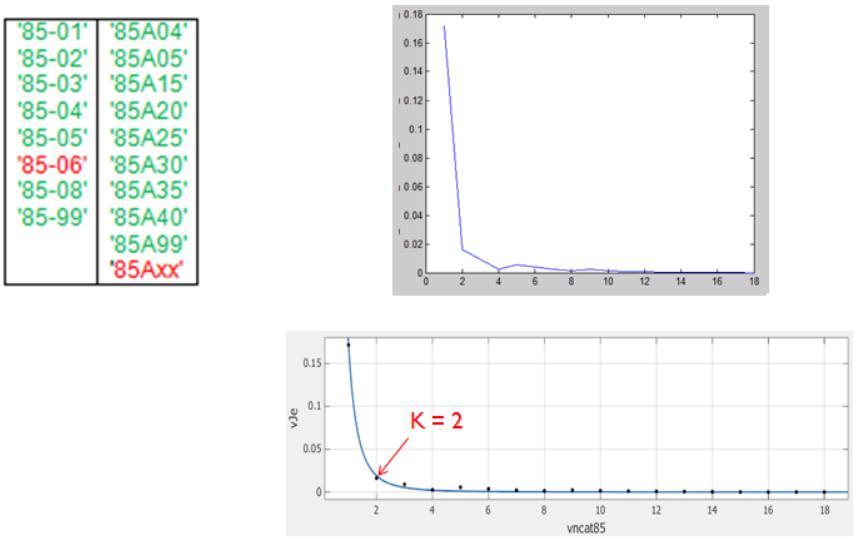


Figura 20: Clustering y grafica Je-k de la categoría 85

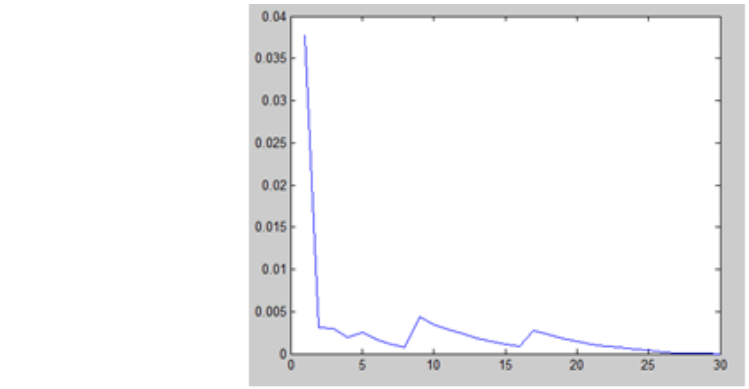
5.4 Experimento 4

En este cuarto y último experimento, se va a trabajar con grupos de categorías definidos. Vamos a probar si el algoritmo es capaz de distinguir diferentes subconjuntos de categorías, dándole a cada grupo de categorías el mismo número de documentos.

Hemos realizado 40 pruebas, cogiendo diferentes subconjuntos de categorías, que pertenezcan a la misma área, a áreas diferentes, que traten temas similares o no, etc. A continuación vamos a comentar las que nos han parecido más relevantes.

En la figura 21, podemos observar como para dos subconjuntos el algoritmo es capaz de distinguir entre la categoría 20 y la 39; en cambio en la figura 22 que también tiene dos subconjuntos, hay una categoría que termina -06 y otra en xx, tal y como ha pasado en los experimentos anteriores, nuevamente han sido agrupadas en clusters aparte.

'20M05'	'39B05'
'20M07'	'39B12'
'20M10'	'39B22'
'20M11'	'39B32'
'20M12'	'39B42'
'20M13'	'39B52'
'20M14'	'39B55'
'20M15'	'39B62'
'20M17'	'39B72'
'20M18'	'39B82'
'20M19'	'39B99'
'20M20'	'39Bxx'
'20M25'	
'20M30'	
'20M32'	
'20M35'	
'20M50'	
'20M99'	



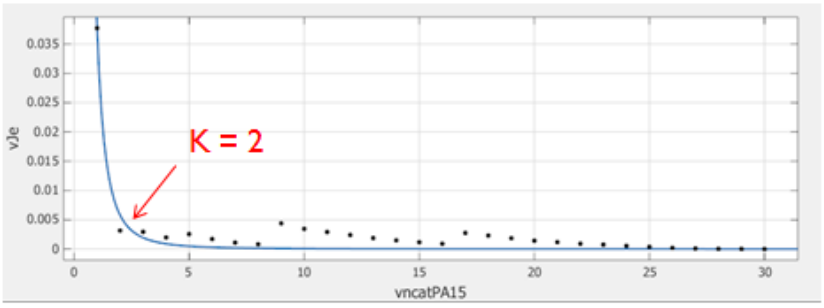
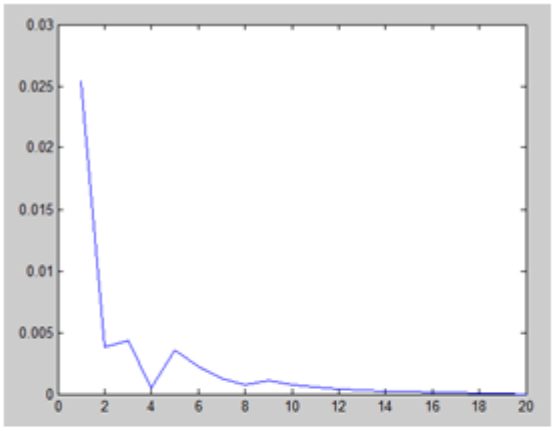


Figura 21: Prueba PA15 -Clustering y grafica Je-k para dos subconjuntos

'26C10'	'53-06'
'26A27'	'53D99'
'26A42'	'53Dxx'
'26A24'	'53C25'
'26A30'	'53C40'
'26A48'	'53C42'
'26A33'	'53B20'
'26A39'	'53D12'
'26A15'	'53D40'
'26D15'	'53C12'



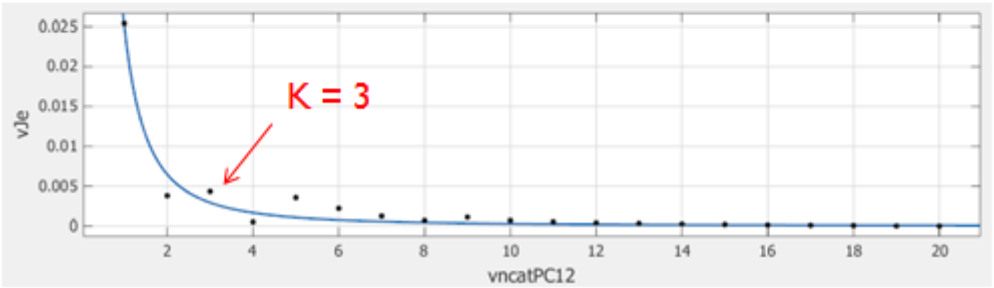


Figura 22: Prueba PC12 -Clustering y grafica Je-k para dos subconjuntos

En la figura 23, se ha hecho el experimento con tres subconjuntos. Vemos que el algoritmo es capaz de distinguir entre la categoría 91 y las otras dos, en cambio entre la categoría 93 y 94, tiene problemas. Cuantas más categorías añadimos a los subconjuntos, más problemas empieza a tener.

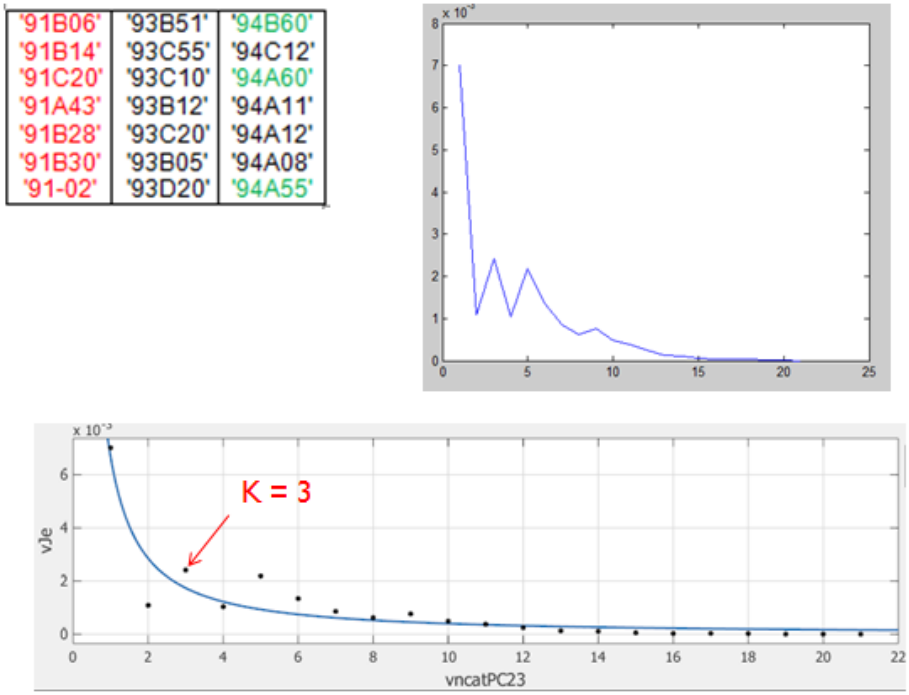


Figura 23: Prueba PC23 -Clustering y grafica Je-k para tres subconjuntos

En la figura 24, se ha hecho el experimento con cuatro subconjuntos, esta vez con categorías que terminan en -06. Seleccionamos k=3, y vemos como las categorías que terminan en -06 se agrupan juntas, la 26 con la 30, y la 53 con la 56.

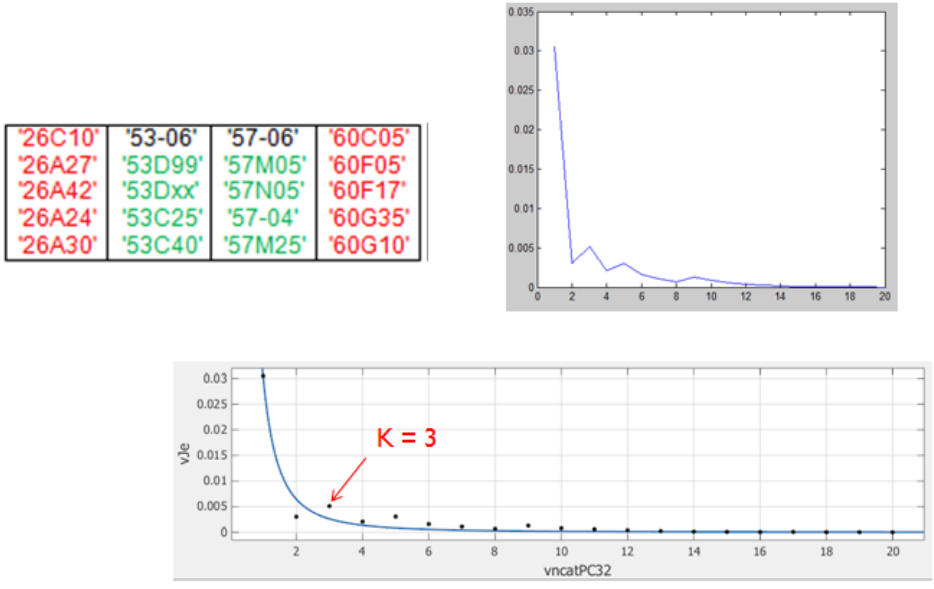


Figura 24: Prueba PC32 -Clustering y grafica Je-k para cuatro subconjuntos

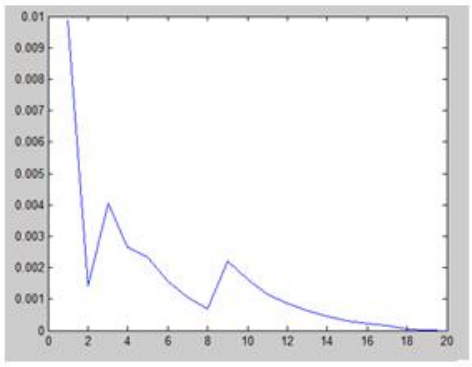
Si en lugar de seleccionar k=3, seleccionamos k=4, vemos que el nuevo cluster es para la categoría que terminar en XX, en lugar de separar las categorías 26 de la 60, o la 53 de la 57.

'26C10'	'53-06'	'57-06'	'60C05'
'26A27'	'53D99'	'57M05'	'60F05'
'26A42'	'53Dxx'	'57N05'	'60F17'
'26A24'	'53C25'	'57-04'	'60G35'
'26A30'	'53C40'	'57M25'	'60G10'

Figura 25: Prueba PC32 con k=5

Por último, en la figura 26, vemos uno de los experimentos con cinco subconjuntos. Cuantas más categorías añadimos a los subconjuntos, más problemas empieza a tener, las categorías 01 y 12 es capaz de diferenciarlas, pero con el resto tiene problemas. Cuantas más categorías tenemos, menos documentos podemos indexar por cada una, en este caso solo 100, ya que si no, no se podía completar los cálculos.

'01A60'	'12E30'	'26C10'	'32B10'	'42C05'
'01-02'	'12-02'	'26A27'	'32G34'	'42B20'
'01A70'	'12E25'	'26A42'	'32A25'	'42B25'
'01A50'	'12F12'	'26A24'	'32A36'	'42A38'



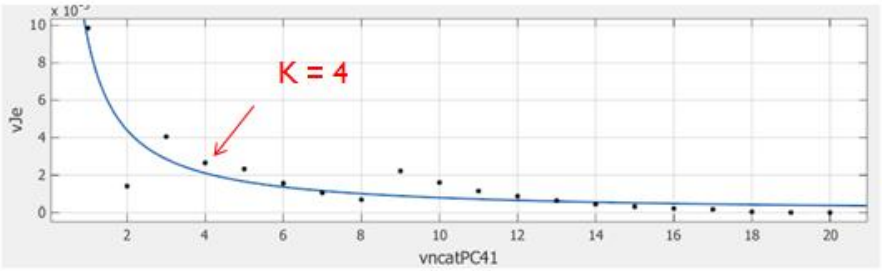


Figura 26: Prueba PC41 -Clustering y grafica Je-k para cinco subconjuntos

5.5 Análisis de los resultados

Aunque no se han podido completar los resultados con todo el corpus de datos, lo cual era indispensable para poder crear un nivel de superclases sobre la taxonomía predefinida, se han podido obtener resultados esperanzadores aplicando este algoritmo.

Lo más importante a destacar es que todas las categorías que pertenecen a los grupos categoría – {00, 01, 02, 03, 04, 06}, pesa más su pertenencia a una colección o a un histórico, que el hecho que sean de álgebra o de funciones reales, por lo que cada una de ellas podría ser una nueva clase.

Las categorías que terminan en codificación -99 o –XX, son categorías que se engloban dentro de una categoría, pero que no tienen lugar dentro de las subcategorías existentes; el algoritmo tiende siempre a separarlas en grupos aparte. Sería conveniente estudiarlas más, para crear un nuevo grupo.

Las categorías que pertenecen a temas generales, tienden a clasificarse juntas, aunque dentro de lo general tengan en particular algo relacionado con las matemáticas.

Se han hecho muchas más pruebas, y en la gran mayoría los resultados han sido siempre similares. Los resultados dependen del número de ficheros que contengan una cierta categoría, y por tanto del número de términos con el que se puede trabajar. Hay categorías que están en muchos ficheros, en cambio otras apenas aparecen, esto condiciona la solución, ya que de algunas tenemos mucha información aportada por los términos, y en otras casi nada.

Para poder obtener mejores resultados habría que evaluar este algoritmo con un ordenador mucho más potente, que pueda hacer frente a los tamaños de las matrices que se requieren e implementar una función Generalized Singular Value Decomposition que pueda trabajar con matrices dispersas, ya que sino el problema se ve muy afectado.

Para concluir se puede decir, que aunque no se ha podido probar con grandes volúmenes de datos, los resultados obtenidos son bastante buenos y esperanzadores; y que habría que hacer un estudio más profundo de la codificación MSC o bien analizar mejor las reseñas que se hacen en los documentos matemáticos.

6. CONCLUSIONES

En este último capítulo se resumen la consecución de objetivos y la valoración personal del trabajo llevado a cabo en este Proyecto Fin de Carrera.

6.1 Consecución de objetivos

El éxito en la realización del proyecto debe analizarse desde la perspectiva de la adecuación de los resultados a los objetivos iniciales, así que vamos a recordarlos y comprobaremos si el proyecto desarrollado los cumple:

- Evaluar una aproximación para la introducción de superclases la clasificación MSC
- Implementar el algoritmo descrito en el paper *"Hierarchical Taxonomy Preparation for Text Categorization Using Consistent Bipartite Spectral Graph Copartitioning"*
- Evaluar el algoritmo con datos de una biblioteca digital matemática
- Examinar la estructura de la taxonomía MSC para mejorar el rendimiento global de la clasificación

Aunque uno de los objetivos principales del proyecto era realización de una aproximación para la introducción de superclases sobre la taxonomía, debido a los problemas que hemos tenido con la falta de memoria, no se han podido completar los cálculos con todo el corpus de datos, por lo que no se han podido obtener las superclases esperadas, pero se han obtenido resultados esperanzadores. El algoritmo se ha implementado con éxito, y se ha evaluado según diferentes aproximaciones: subconjuntos del corpus de datos, categorías del primer nivel de MSC, subconjuntos de categorías, con el mismo primer nivel y subconjuntos de categorías bien definidos. Se ha examinado la estructura de la taxonomía MSC, y se ha observado que la clasificación no es perfecta o bien que la anotación que se hace sobre los documentos no es la adecuada y con las pruebas realizadas, se han probado que en un futuro se podría mejorar el rendimiento global de la clasificación, sobre todo analizando mejor las categorías que pertenecen al grupo –{00,01,02,03,04,06} e intentar localizar mejor aquellas que terminan en -XX

Dado que no se ha podido completar completamente el objetivo de este proyecto, debido a la limitación de memoria, se propone como posible trabajo futuro implementar la función GSVD para datos dispersos, y una vez que esto se consiga, se podrá evaluar con más datos y analizar si la clasificación resultante es mejor.

6.2 Valoración Personal

Este proyecto se ha desarrollado en una universidad extranjera, dentro del marco del programa Erasmus. Sumado al reto que supone realizar un proyecto final de carrera, hay que sumar la dificultad que ha presentado hacerlo en otro idioma, y tener que lidiar con los requisitos que se piden en otras universidades.

Personalmente, me encuentro contenta con el trabajo que he realizado, aunque he tenido muchas dificultades para poder realizarlo, he sabido solventarlas, buscando soluciones, y sacando el proyecto adelante.

7. BIBLIOGRAFÍA

[Paper implementado]

Hierar-chical taxonomy preparation for text categorization using consistent bipartite spectral graph copartitioning

http://research.microsoft.com/pubs/131500/TKDE-FINAL-01490532.pdf

[Papers consultados]

Co-clustering documents and words using Bipartite Spectral Graph Partitioning

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.74.2909&rep=rep1&type=pdf

Exploiting confusion matrices for automatic generation of topic hierarchies and scaling up multiway classifiers

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.16.6404&rep=rep1&type=pdf

Topic hierarchy generation via linear discriminant projection.

http://www.researchgate.net/publication/221301366_Topic_hierarchy_generation_via_linear_discriminant_projection

A comparative study on feature selection in Text Categorization

http://faculty.cs.byu.edu/~ringger/Winter2007-CS601R-2/papers/yang97comparative.pdf

Data Clustering

http://www.cs.rutgers.edu/~mlittman/courses/lightai03/jain99data.pdf

[MSC web] Página web del MSC

http://www.ams.org/mathscinet/msc/msc2010.html

[SVD web] Páginas web sobre Singular Value Decomposition

http://www.mate.unlp.edu.ar/practicas/70_18_0911201012951.pdf

http://www.ehu.es/izaballa/Ana_Matr/Apuntes/lec3.pdf

http://www.utdallas.edu/~herve/Abdi-SVD2007-pretty.pdf

http://www.ling.ohio-state.edu/~kbaker/pubs/Singular_Value_Decomposition_Tutorial.pdf

[GSVD web] Páginas web sobre Generalized Singular Value Decomposition

http://www.utdallas.edu/~herve/Abdi-SVD2007-pretty.pdf

http://www.mathworks.es/es/help/matlab/ref/gsvd.html

http://www.netlib.org/lapack/lug/node36.html

[PYTHON web] Tutoriales de Python http://docs.python.org/2/tutorial/

[MATLAB web] Página web de Matlab www.mathworks.com/

ANEXOS

ANEXO A.

Paper “Hierarchical Taxonomy Preparation for Text Categorization Using Consistent Bipartite Spectral Graph Copartitioning”.

Este anexo incluye el paper de investigación matemática que se ha utilizado como base para desarrollar la evaluación de clasificación de documentación matemática.

El enlace al paper se puede encontrar en formato .PDF en la siguiente dirección: http://research.microsoft.com/en-us/people/tyliu/gao-tkde05.pdf

Hierarchical Taxonomy Preparation for Text Categorization Using Consistent Bipartite Spectral Graph Copartitioning

Bin Gao, Tie-Yan Liu, *Member, IEEE Computer Society*, Guang Feng, Tao Qin, Qian-Sheng Cheng, and Wei-Ying Ma, *Member, IEEE Computer Society*

Abstract—Multiclass classification has been investigated for many years in the literature. Recently, the scales of real-world multiclass classification applications have become larger and larger. For example, there are hundreds of thousands of categories employed in the Open Directory Project (ODP) and the Yahoo! directory. In such cases, the scalability of classification methods turns out to be a major concern. To tackle this problem, hierarchical classification is proposed and widely adopted to get better trade-off between effectiveness and efficiency. Unfortunately, many data sets are not explicitly organized in hierarchical forms and, therefore, hierarchical classification cannot be used directly. In this paper, we propose a novel algorithm to automatically mine a hierarchical structure from the flat taxonomy of a data corpus as a preparation for the adoption of hierarchical classification. In particular, we first compute matrices to represent the relations among categories, documents, and terms. And, then, we cocluster the three substances at different scales through consistent bipartite spectral graph copartitioning, which is formulated as a generalized singular value decomposition problem. At last, a hierarchical taxonomy is constructed from the category clusters. Our experiments showed that the proposed algorithm could discover very reasonable taxonomy hierarchy and help improve the classification accuracy.

Index Terms—Clustering, data mining, singular value decomposition, text processing.

1 Introduction

Multiclass classification has been actively investigated for many years. On one hand, several m-way classifiers such as k-nearest neighbors (k-NN) [5] and Naive Bayes (NB) [5], [18] were developed. On the other hand, people have developed some strategies to extend well-performed binary classifiers such as Support Vector Machines (SVM) [21] to the multiclass case. To the best of our knowledge, the earliest and the most popular strategy is one-against-rest, in which each of the binary classifiers is trained to distinguish one category from all other categories and all these classifiers will make YES/NO decisions on a given instance in the test phase. As a result, the instance will be classified into the category corresponding to the highest confidence score. Many benchmark evaluations [15], [21] have shown that one-again-rest SVM classifiers (denoted by flat SVM) lead to higher classification accuracy as compared to NB and k-NN.

The flat SVM works pretty well when the category number is small. However, in recent years, the problem scale of multiclass classification has been larger and larger. For instance, the well-known large-scale Web directories Open Directory Project (http://dmoz.org/) and Yahoo! directory (http://dir.yahoo.com/) have 118,488 and 292,216 categories, respectively. In such large-scale cases, flat SVM will suffer from its poor scalability because its training complexity is $O(MN)$ and its test complexity is $O(M)$, where M and N are the total numbers of categories and documents in the corpus [16], [24].

To tackle this problem, people proposed using the inner hierarchical structures among the categories to divide the classification task. This resulted in the so-called hierarchical SVM, in which a SVM classifier is trained to distinguish each child category only from other categories with the same parent, rather than from all other categories in the corpus. For the test phase, *pachinko-machine* search is used (i.e., a lower-level SVM classifier is activated only if its parent gives a YES decision). Empirical studies on large-scale data sets have shown that hierarchical SVM often excels flat SVM in both complexity and accuracy [6], [16]: On one hand, local feature selection and parameter tuning might improve the effectiveness; on the other hand, the training complexity is reduced because the training documents of each local task are much less than the flat setting, and the *pachinko-machine* search will significantly reduce the time consumption for testing.

However, as we all know, it is not a necessity that data corpora have explicitly-given hierarchical taxonomies. Therefore, for many cases, hierarchical classification can not be adopted even if we know it is superior to flat classification. To tackle this problem, one possible solution is to mine a hierarchical configuration by our own and use it to organize the hierarchical classifiers. In other words, we need to preprocess the data corpus so as to do some essential preparations for hierarchical classification.

- *B. Gao and Q.-S. Cheng are with LMAM, Department of Information Science, School of Mathematical Sciences, Peking University, Haidian District, Beijing, 100871, P.R. China. E-mail: gaobin@math.pku.edu.cn, qcheng@pku.edu.cn.*
- *T.-Y. Liu and W.-Y. Ma are with Microsoft Research Asia 5F, Sigma Center, No. 49, Zhichun Road, Haidian District, Beijing, 100080, P.R. China. E-mail: {t-tyliu, wyma}@microsoft.com.*
- *G. Feng and T. Qin are with MSP Laboratory, Department of Electronic Engineering, Tsinghua University, Haidian District, Beijing, 100084, P.R. China. E-mail: {fengg03, qinshitao99}@mails.tsinghua.edu.cn.*

Manuscript received 20 Nov. 2004; revised 28 Mar. 2005; accepted 1 Apr. 2005; published online 19 July 2005.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDESI-0440-1104.

To our knowledge, there have been some attempts in this direction, although not many. Vural and Dy [22] proposed a method named divided-by-two (DB2), where the categories of a data corpus is recursively divided into two subsets until each subset consists of only one category. As a result, a binary taxonomy tree is generated. The subset division is done through clustering categories based on the similarities among their documents. Godbole et al. [8], [9] proposed another method to automatically generate the hierarchical taxonomy. They used an NB classifier to quickly compute a category-by-category confusion matrix and adopted a graph-based clustering method to extract a hierarchical taxonomy.

From the above previous works, we can see that clustering might be a very important step for hierarchical taxonomy mining. However, the clustering methods used in these works are not very effective because they have not made full use of the information contained in categories, documents, and terms. This is very similar to what happened in the field of document clustering. As we know, most early document clustering algorithms directly defined the similarity of documents by the similarity of their term representations (i.e., the cosine distance between two TF-IDF [1] vectors) until the concept of document-term coclustering was proposed [2], [3], [25]. In the philosophy of coclustering, the similarity between documents is defined by their term representations while the similarity between terms is defined by their occurrences in documents. In other words, document similarity and term similarity are defined in a reinforcing manner. In such a way, it has been shown that the clustering performance can be greatly improved [2], [3], [25]. Similarly, we believe that, by using the information contained in categories, documents, and terms for category clustering rather than only using the category information or document information, we can get more reasonable results on taxonomy mining. Actually, in our previous work [23], we have proposed a coclustering framework named ReCoM with the similar philosophy. However, in that work, heterogeneous objects were treated as homogeneous objects under some heuristic scaling factors. Sometimes this assumption might be too strong and the tuning of the scaling factor is theoretically difficult. Therefore, we want to avoid it in the method proposed in this paper.

In particular, in this paper, we use one bipartite graph to represent the relationship between categories and documents and use another to represent the relationship between documents and terms. Then, we partition these two bipartite graphs consistently by solving a generalized singular value decomposition problem. Experiments showed that our method could not only lead to a very reasonable taxonomy hierarchy, but also result in higher classification accuracy than previous methods.

The rest of this paper is organized as follows: In Section 2, the background knowledge and related work are introduced. Then, Section 3 describes the proposed method in detail, while experimental results are discussed in Section 4. Some concluding remarks and future work directions are listed in the last section.

2 Related Works

As the proposed method is illumined by the thought of document-term coclustering, in this section, we will introduce some essential background knowledge and review a representative work in this direction.

For the first step, we need to introduce the widely-used document representation in information retrieval and text categorization, vector space model (VSM) [1], because most the previous works on coclustering were built on top of it. The main idea of VSM is to directly treat terms in the documents as features, so as to map a document to a vector in the feature space. Specifically, in VSM, $D = d_1, d_2, \ldots, d_n$ denotes the documents in the corpus and $T = w_1, w_2, \ldots, w_t$ denotes the terms. Then, each document d_i in D can be represented by a t-dimensional vector $d_i = x_{i1}, x_{i2}, \ldots, x_{it}$, where x_{ij} is the term frequency [1] (or TF-IDF [1]) of term w_j in document d_i.

The problem of document-term coclustering is to cluster the documents $D = d_1, d_2, \ldots, d_n$ and terms $T = w_1, w_2, \ldots, w_t$ in a reinforcing manner. There are several methods to tackle this problem [2], [3], [13], [25], among which Dhillon et al. proposed an approach based on bipartite spectral graph partitioning. To illustrate how their algorithm works, we need to introduce some essential knowledge on graph partitioning as below.

A graph $G = (V, E)$ is composed by a set of vertices $V = \{1, 2, \ldots, |V|\}$ and a set of edges $E = \{< i, j > | i, j \in V\}$, where $|V|$ represents the number of vertices. If using E_{ij} to denote the weight of edge $< i, j >$, we can further define the adjacency matrix M of the graph by

$$M_{ij} = \begin{cases} E_{ij} & if \ < i, j > \in E \\ 0, & otherwise. \end{cases} \quad (1)$$

Suppose the vertex set V is partitioned into two subsets V_1 and V_2, then the corresponding *cut* is defined as:

$$cut(V_1, V_2) = \sum_{i \in V_1, j \in V_2} M_{ij}. \quad (2)$$

The above definition can be easily extended to k subsets:

$$cut(V_1, V_2, \ldots, V_k) = \sum_{\eta < \theta} cut(V_\eta, V_\theta). \quad (3)$$

If the vertices of a graph can be decomposed into two disjoint subsets such that no two vertices within the same set are adjacent, the graph is named a bipartite graph. In other words, a bipartite is a triplet $G = (V_1, V_2, E)$, where V_1 and V_2 are two vertex sets, within each of which no vertices are adjacent and E is a set of edges connecting vertices from different vertex sets, i.e., $E = \{< i, j > | i \in V_1, j \in V_2\}$. Specifically, in [2], an undirected bipartite graph, as shown in Fig. 1, is used to represent the relationship between documents and terms. Here, V_1 and V_2 are replaced with D and T, which represent the subsets of document vertices and term vertices, respectively, and E denotes the set of edges $\{< d_i, w_j > | d_i \in D, w_j \in T\}$. An edge $< d_i, w_j >$ exists if and only if the term w_j occurs in document d_i and its weight E_{ij} equals the corresponding term frequency.

If we use B to denote the document-by-term matrix in which B_{ij} equals the edge weight E_{ij}, the adjacency matrix of the bipartite graph, as shown in Fig. 1, can be written as:

$$M = \begin{array}{c} \\ D \\ T \end{array} \begin{array}{c} \begin{array}{cc} D & T \end{array} \\ \begin{bmatrix} 0 & B \\ B^T & 0 \end{bmatrix} \end{array}, \quad (4)$$

where the vertices have been ordered such that the first n vertices index the documents (denoted by D) while the last t index the terms (denoted by T).

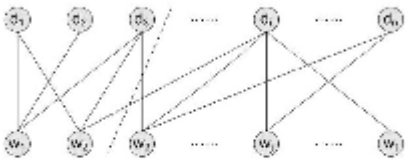


Fig. 1. The document-term bipartite graph.

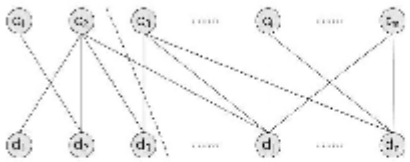


Fig. 2. The category-document bipartite graph.

The key idea of [2] is to find a partition of the vertices in the graph such that the *cut* as defined in (2) (or normalized cut [19], ratio cut [11], min-max cut [4], etc.) can be minimized. For instance, if the dashed line in Fig. 1 shows the very partition that minimizes the *cut*, we will obtain two subsets, $\{d_1, d_2, d_3, w_1, w_2\}$ and $\{d_4, \ldots, d_n, w_3, \ldots, w_t\}$. Therefore, the documents are clustered into two groups, $\{d_1, d_2, d_3\}$ and $\{d_4, \ldots, d_n\}$, while the terms are clustered into $\{w_1, w_2\}$ and $\{w_3, \ldots, w_t\}$ at the same time. It was proved in [2], [10], [19] that the eigenvector associated with the second smallest eigenvalue λ_2 of the generalized eigenvalue problem $L\omega = \lambda Q\omega$ is an optimal embedding for the partition of the vertices that minimizes the *cut*. Here, Q is a diagonal matrix with $Q_{ii} = \sum_k E_{ik}$, $L = Q - M$ is the Laplacian matrix, and ω is a column vector. After some trivial deduction, the above problem can be converted to a singular value decomposition (SVD) problem, which can be computed more efficiently. For the details of this algorithm, please refer to [2], [25]. Although there are some other works on document-term coclustering, since their basic philosophy is not relevant to our proposed method, we will not review them in details in this section. One can find these reference works in [3], [13].

3 Coclustering-Based Hierarchical Taxonomy Mining

As shown in the introduction and the related works, clustering might be a key technology in unsupervised taxonomy mining. In this section, we will also use this methodology to mine hierarchical taxonomy from data corpora. Specifically, we propose a novel clustering algorithm, which coclusters categories, documents, and terms based on consistent bipartite spectral graph copartitioning. We will first explain how this new algorithm is formulated from Section 3.1 to 3.3, and then discuss how to construct the taxonomy hierarchy based on the output of this algorithm in Section 3.4.

Before going into the details of the algorithm description, we need to give some additional notations. Besides the document ($D = \{d_1, d_2, \ldots, d_n\}$) and term ($T = \{w_1, w_2, \ldots, w_t\}$) representations as mentioned in Section 2, for our application, text categorization, we also have one more substance, category. Each document will be assigned a category label from the set $C = \{c_1, c_2, \ldots, c_m\}$, where m is the number of categories. The coclustering will be conducted on all these three substances of C, D, and T.

3.1 Coclustering-Based on Category-Document Bipartite Spectral Graph Partitioning

It is not difficult to acknowledge the analogy between document-category coclustering and document-term coclustering: Categories are similar because the documents in them are similar, while documents are similar because they are likely to be classified into similar categories. Therefore, we can construct a bipartite graph to represent the relationship between categories and documents as well (see Fig. 2).

The category-by-document matrix A can be easily built according to the information from the corpus. In this matrix, rows correspond to categories and columns to documents. Each element A_{ij} indicates the correlation between document d_j and category c_i. If document d_j belongs to k categories $c_1, c_2, \ldots, c_k$, the weights $A_{1j}, A_{2j}, \ldots, A_{kj}$ are set to $1/k$, and the other elements of the jth column of matrix A are set to zero. Then we can write the adjacency matrix of the bipartite graph in Fig. 2 as follows:

$$M = \begin{array}{c} \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cc} C & D \end{array} \\ \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \end{array}, \tag{5}$$

where the vertices have been ordered such that the first m vertices index the categories (denoted by C) while the last n index the documents (denoted by D).

It is easy to understand that the coclustering of categories and documents can also be mapped to solving the generalized eigenvalue problem $L\omega = \lambda Q\omega$. Here, Q is a diagonal matrix with $Q_{ii} = \Sigma_k M_{ik}$ and $L = Q - M$ is the Laplacian matrix. Similar to Dhillon's method [2] described in Section 2, we can solve this problem by computing a singular value decomposition. The corresponding algorithm is given as below:

Algorithm 1.

1. Given A, form its normalized version,
$$\hat{A} = P^{-1/2} A R^{-1/2},$$
where P and R are diagonal matrices with $P_{ii} = \Sigma_j A_{ij}$, $R_{ii} = \Sigma_j A_{ji}$.
2. Compute the left singular vector u_2 and the right singular vector v_2 corresponding to the second largest singular value of $\hat{A}$ and get the eigenvector corresponding to the second smallest eigenvalue of $L\omega = \lambda Q\omega$ as $\omega_2 = [P^{-1/2}u_2 \;\; R^{-1/2}v_2]^T$.
3. Cluster on the one-dimensional data $P^{-1/2}u_2$ and $R^{-1/2}v_2$ to obtain the desired bipartition of categories and documents, respectively.

The same manner as described in [2] can be used to extend this algorithm to its multipartitioning version. While we use the second singular vector to partition the graph into two parts, we could use $l = \lceil \log_2 k \rceil$ singular vectors on each side ($u_2, u_3, \ldots, u_{l+1}$ and $v_2, v_3, \ldots, v_{l+1}$) when partitioning the graph into k parts. In such a way, Algorithm 1 can be updated as below:

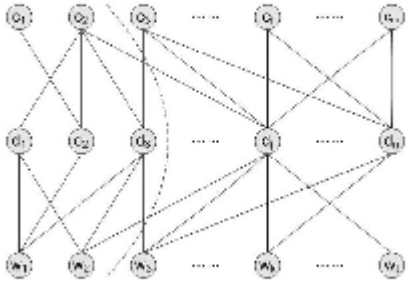


Fig. 3. The category-document-term tripartite graph.

Algorithm 2.

1. Given A, form P, R, and $\hat{A} = P^{-1/2}AR^{-1/2}$.
2. Compute $l = \lceil \log_2 k \rceil$ singular vectors of $\hat{A}$, $u_2, u_3, \ldots, u_{l+1}$ and $v_2, v_3, \ldots, v_{l+1}$, and form Ω as in (6):

$$\Omega = \begin{bmatrix} P^{-1/2}U \\ R^{-1/2}V \end{bmatrix}, \text{ with } \begin{cases} U = [u_2, u_3, \ldots, u_{l+1}] \\ V = [v_2, v_3, \ldots, v_{l+1}] \end{cases}. \quad (6)$$

3. Cluster on the l-dimensional data $P^{-1/2}U$ and $R^{-1/2}V$ to obtain the desired k-partitioning of categories and documents, respectively.

It seems that Algorithm 1 and 2 are natural and workable; however, we have to point out a critical problem of these formulations. Although, in the case of multilabel classification, the above methods might be able to cluster correlated categories together, in the single-label case, the graph is actually made up of m unconnected subgraphs corresponding to the m categories because the weights of those edges between a category and a document which does not belong to the category will be zero. Therefore, we cannot get a desirable partitioning because the connectivity of the graph is not guaranteed. To avoid this situation, one possible way is to further add the term information to the category-document graph so that the graph can be well connected.

3.2 Coclustering Based on Category-Document-Term Tripartite Spectral Graph Partitioning

As the similarity between documents can be defined not only by categories but also by terms, the bipartite graph proposed in Section 3.1 reflects only partial information of the data corpus. To compensate it and avoid the failure case in single-label categorization, a straightforward way is to leverage the term information so as to cocluster the category, document, and term at the same time.

Superimposing Fig. 2 upon Fig. 1, we will obtain a category-by-document-by-term tripartite graph as shown in Fig. 3. Here, a k-partite graph is a graph whose graph vertices can be partitioned into k disjoint sets so that no two vertices within the same set are adjacent.

It is easy to derive the adjacency matrix for the above graph:

$$M = \begin{array}{c} \\ C \\ D \\ T \end{array} \begin{array}{c} \begin{array}{ccc} C & D & T \end{array} \\ \begin{bmatrix} 0 & A & 0 \\ A^T & 0 & \alpha B \\ 0 & \alpha B^T & 0 \end{bmatrix} \end{array}, \quad (7)$$

where α is a weighting parameter and the vertices have been ordered such that the first m vertices index the categories (denoted by C), the next n index the documents (denoted by D), and the last t index the terms (denoted by T).

Up to now, by introducing the term information, we might have the chance to solve the failure case caused by lack of graph connectivity. Then, does everything go smoothly? And can the coclustering be worked out by solving the generalized eigenvalue problem corresponding to this adjacency matrix? Unfortunately, the answers to the above questions are once again negative. In fact, if we move the category vertices in Fig. 3 to the side of the term vertices, the original tripartite graph will turn out to be a bipartite graph. This can also be proved through the following deductions: In the tripartite case, as shown in Fig. 3,

$$Q = \begin{bmatrix} P & 0 & 0 \\ 0 & R & 0 \\ 0 & 0 & S \end{bmatrix} \text{ and } L = \begin{bmatrix} P & -A & 0 \\ -A^T & R & -\alpha B \\ 0 & -\alpha B^T & S \end{bmatrix}, \quad (8)$$

where P, R, and S are diagonal matrices such that $P_{ii} = \Sigma_j A_{ij}$, $R_{ii} = \Sigma_j A_{ji} + \alpha \Sigma_j B_{ij}$, and $S_{ii} = \alpha \Sigma_j B_{ji}$. Let vector $\omega = (x, y, z)^T$, where x, y, and z are column vectors of m, n, and t dimensions, respectively. Then, $L\omega = \lambda Q\omega$ may be written as:

$$\begin{bmatrix} P & -A & 0 \\ -A^T & R & -\alpha B \\ 0 & -\alpha B^T & S \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \lambda \begin{bmatrix} P & 0 & 0 \\ 0 & R & 0 \\ 0 & 0 & S \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}. \quad (9)$$

Assuming that P, R, and S are all nonsingular, the above equations can be rewritten as:

$$\begin{cases} P^{-1/2}Ay & = (1-\lambda)P^{-1/2}x \\ R^{-1/2}A^Tx + \alpha R^{-1/2}Bz & = (1-\lambda)R^{-1/2}y \\ \alpha S^{-1/2}B^Ty & = (1-\lambda)S^{-1/2}z. \end{cases} \quad (10)$$

If we set $u = P^{1/2}x$, $v = R^{1/2}y$, and $s = S^{1/2}z$, we will get the following equations after a little algebraic simplification:

$$\begin{cases} P^{-1/2}AR^{-1/2}v & = (1-\lambda)u \\ R^{-1/2}A^TP^{-1/2}u + \alpha R^{-1/2}BS^{-1/2}s & = (1-\lambda)v \\ \alpha S^{-1/2}B^TR^{-1/2}v & = (1-\lambda)s. \end{cases} \quad (11)$$

Uniting the first and the last formulae in (11) together and considering that P, R, and S are all diagonal matrices, we will have:

$$\begin{cases} \begin{bmatrix} P^{-1/2}AR^{-1/2} \\ S^{-1/2}(\alpha B)^TR^{-1/2} \end{bmatrix} v = (1-\lambda) \begin{bmatrix} u \\ s \end{bmatrix} \\ \begin{bmatrix} P^{-1/2}AR^{-1/2} \\ S^{-1/2}(\alpha B)^TR^{-1/2} \end{bmatrix}^T \begin{bmatrix} u \\ s \end{bmatrix} = (1-\lambda)v. \end{cases} \quad (12)$$

Let $\mu = (u, s)^T$ and

$$F = \begin{bmatrix} P^{-1/2}AR^{-1/2} \\ S^{-1/2}(\alpha B)^TR^{-1/2} \end{bmatrix}.$$

Then, we have:

$$\begin{cases} Fv = (1-\lambda)\mu \\ F^T\mu = (1-\lambda)v. \end{cases} \quad (13)$$

Therefore, $(1-\lambda)$ is the singular value of F and the generalized eigenvalue problem $L\omega = \lambda Q\omega$ is converted to an SVD problem, where $\mu = (u, s)^T$ and v are the left and right singular vectors, respectively. Note that u and s,

representing categories and terms, respectively, are both embedded into the left singular vectors of F. That is, Fig. 3 is actually equivalent to a bipartite graph and we have to distinguish (by the weighting parameter α) the loss of cutting a category-document edge from the loss of cutting a document-term edge since they contribute to the same loss function. However, it is nontrivial to choose a proper value for α: if it is too small, the influence from the matrix B will make little sense and this graph will degenerate to the graph in Fig. 2. On the contrary, if it is too large, we will risk the situation of assigning all category vertices into one subset.

To summarize the above discussions, analyzing the matrix M in (7) by traditional spectral clustering methods does not work as it is expected. To tackle this problem, we will propose a novel algorithm based on consistent bipartite graph copartitioning. In this algorithm, we no longer need to tune the weight to balance document-category and document-term edges. Instead, we use generalized SVD to fuse the two bipartite graphs as shown in Fig. 1 and Fig. 2. The corresponding details will be described in the next section.

3.3 Coclustering Based on Consistent Bipartite Spectral Graph Copartitioning

As aforementioned, the motivation of coclustering categories, documents, and terms is to leverage the complementary information contained in Fig. 1 and Fig. 2. As we know, a document is the bridge between these two bipartite graphs. Therefore, if we can design a method to iteratively refine the partitioning of each graph by the information contained in the other, the stationary state will have the same partitioning of documents in these two graphs and the corresponding category clustering will be more reasonable and effective. We name this stationary state by consistent copartitioning of the two bipartite graphs.

In the following discussions, we will first show that standard SVD on Fig. 1 and Fig. 2 can hardly result in consistent copartitioning and, then, propose a new method to guarantee the consistent copartitioning.

3.3.1 Standard SVD Does Not Guarantee Consistent Copartitioning

To proceed with our discussions, we use M_1 to denote the adjacency matrix of Fig. 2 and M_2 to denote the adjacency matrix of Fig. 1:

$$M_1 = \begin{array}{c} \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cc} C & D \end{array} \\ \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \end{array}, \quad M_2 = \begin{array}{c} \\ D \\ T \end{array}\begin{array}{c} \begin{array}{cc} D & T \end{array} \\ \begin{bmatrix} 0 & B \\ B^T & 0 \end{bmatrix} \end{array}, \tag{14}$$

where A and B are defined as aforementioned and the normalized forms of them are:

$$\hat{A} = P_1^{-1/2} A R_1^{-1/2}, \quad \hat{B} = P_2^{-1/2} B R_2^{-1/2}. \tag{15}$$

Here, P_1, R_1, P_2, and R_2 are diagonal matrices as defined in Algorithm 1.

Due to the spectral graph theory [2], [25], if we conduct the singular value decompositions on $\hat{A}$ and $\hat{B}$ as

$$\hat{A} = U_A \Sigma_A V_A^T, \quad \hat{B} = U_B \Sigma_B V_B^T, \tag{16}$$

unitary matrix U_A will embed the clustering information of categories, V_A and U_B will embed the clustering information of documents, and V_B will be the embedding for term clustering.

Our consistent copartitioning will ask for the same partitioning of documents in both graphs. Therefore, if using SVD for clustering as above, we need to guarantee the clustering results of the corresponding columns of V_A and U_B to be the same. However, it is not difficult to understand that the above consistence cannot be satisfied in many cases due to the different properties of the two pregiven matrices $\hat{A}$ and $\hat{B}$ [10]. To illustrate this, let us see an example as follows. Suppose matrices A and B of a single-labeled problem are:

$$A = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \end{array}\begin{array}{c} \begin{array}{cccc} d_1 & d_2 & d_3 & d_4 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}, \quad B = \begin{array}{c} \\ d_1 \\ d_2 \\ d_3 \\ d_4 \end{array}\begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \begin{bmatrix} 4 & 3 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 & 0 \\ 0 & 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 4 & 3 \end{bmatrix} \end{array}.$$

After forming $\hat{A}$ and $\hat{B}$ and conducting the singular value decompositions on them, we will get:

$$V_A = \begin{bmatrix} 0.70711 & 0 & 0 & -0.70711 \\ 0.70711 & 0 & 0 & 0.70711 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

$$U_B = \begin{bmatrix} 0.70711 & 0 & 0 & -0.70711 \\ 0.70711 & 0 & 0 & 0.70711 \\ 0 & 0.70711 & -0.70711 & 0 \\ 0 & 0.70711 & 0.70711 & 0 \end{bmatrix}.$$

Now, we can see that the document clustering results based on U_B is $\{d_1, d_2; d_3, d_4\}$ while the clustering results based on V_A is $\{d_1, d_2, d_4; d_3\}$. Obviously, these two results are not consistent at all.

To overcome this problem, we suggest that we should relax the unitary constraint on V_A and U_B to get a tradeoff. In other words, for both graphs, we might not use the optimal embedding for clustering provided that these two embeddings can be consistent. As will be shown below, generalized SVD is just such a tool that can meet our requirement. Due to the following theory, we can find a consistent copartitioning of documents in these two graphs based on generalized SVD:

Theorem 1. *If we have $\hat{A} \in R^{m \times n}$ and $\hat{B} \in R^{n \times t}$, $m \leq n \leq t$, then there exists unitary matrices $U \in R^{m \times m}$, $V \in R^{t \times t}$ and invertible matrix $X \in R^{n \times n}$ such that:*

$$\begin{cases} \hat{A} = UCX^T \\ \hat{B} = XSV^T, \end{cases} \tag{17}$$

where $C = diag(c_1, c_2, \ldots, c_m)$, $c_i \geq 0$ and $S = diag(s_1, s_2, \ldots, s_n)$, $s_i \geq 0$.

Proof. Please refer to [10]. □

Theorem 1 indicates that there exists a partitioning of documents (whose embedding is X) that can meet the constraints of both graphs. The corresponding partitioning of categories and terms are embedded in U and V, respectively. However, a follow-up question is whether these embeddings are good and how to refine these embeddings if they are not. This will also be a critical part in the success of our proposed idea of consistent spectral bipartite graph copartitioning.

As shown in Theorem 1, X is an invertible matrix. This condition is much more relaxed as compared to standard

SVD where the embedding of document (i.e., V_A) will be a unitary matrix. Therefore, the coclustering information contained in X is not as explicit as in V_A. While the second singular vector of V_A has been a good embedding for document clustering, the same embedding can only be achieved by considering many nonorthogonal columns in X. As a chain reaction, this will also affect the explicitness of the coclustering information embedded in U. Therefore, we could not use the second columns of U, X, and V directly but should think of a way to extract useful information from them. In the mathematical manner, it is equal to finding a proper linear transformation.

For this purpose, we multiply CX^T and XS to form a mixture matrix H and conduct SVD decomposition on H to get two unitary matrices U_H and V_H:

$$H = CX^TXS, \quad H = U_H\Sigma_H V_H^T. \tag{18}$$

Then, we use U_H and V_H as the linear transformations to refine U and V:

$$U^* = UU_H, \quad V^* = VV_H. \tag{19}$$

Eventually, U^* and V^* are regarded as the modified embeddings for coclustering.

To summarize, the consistent bipartite spectral graph copartitioning algorithm (CBSGC) is given as below:

Algorithm 3.

1. Given A and B, form P_1, P_2, R_1, R_2, and $\hat{A}, \hat{B}$.
2. Compute GSVD of $\hat{A}, \hat{B}$ to get U, X, V, C, and S.
3. Form $H = CX^TXS$ and compute SVD of it to get U_H, V_H.
4. Form $U^* = UU_H, V^* = VV_H$ and take the second column vectors of them, u_2 and v_2, to form the normalized embedding vector
$$\omega_2 = [P_1^{-1/2}u_2 \;\; R_2^{-1/2}v_2]^T.$$
5. Cluster on the one-dimensional data $P_1^{-1/2}u_2$ and $R_2^{-1/2}v_2$ to obtain the desired bipartition of categories and terms, respectively.

Similar to the method in Section 3.1, we can extend the above bipartitioning algorithms to adapt the k-partitioning case. We would like to use $l = \lceil \log_2 k \rceil$ vectors on each side ($u_2, u_3, \ldots, u_{l+1}$ and $v_2, v_3, \ldots, v_{l+1}$) to obtain desired k-partitioning. As the extension form is quite similar to Algorithm 2, we omit the details here.

The reasoning of our consistent copartitioning process can be explained as follows: Since we can hardly get a consistent document partitioning if we partition both graphs optimally (corresponding to SVD decomposition), for compromise, we look for nonoptimal partitioning for each graph which can be consistent with each other. Although the partitioning for each graph is not optimal, when considering the two graphs as a whole, the resulting partitioning may be more effective.

For the above algorithm description, considering that our motivation is to get category clusters, one may argue that another approach to tackle the single-label trap is to use the category-term bipartite graph directly. We agree with that, but want to point out it is nontrivial to build the category-term relationship. If we use some heuristics to get it, we may lose some unrecoverable information. Comparatively, our approach is to use the consistency on the document partitioning to bridge categories and terms. This is a more natural and lossless way, because category and term are actually connected by documents in the real world. And Algorithm 3 is just one of the realizations of our idea. That is, we are not necessarily restricted to matrix multiplication or similar linear operations. The algorithm space that we can explore is very rich, under the same idea of consistency.

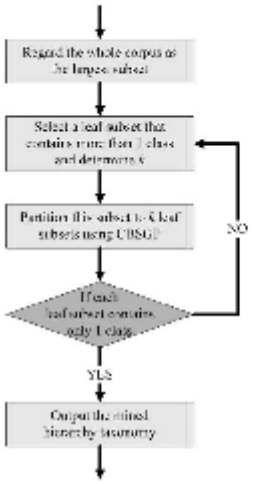


Fig. 4. The flow chart of building hierarchies.

3.4 Building Hierarchical Classifier

After designing the above consistent copartitioning algorithm, we can move onto the part of building the taxonomy hierarchy. As shown in Fig. 4, initially, we regard the whole collection of categories as the root of the hierarchical taxonomy and run the k-partitioning CBSGC algorithm on it to get several subsets of categories in level 2. This process is done recursively until each subset at the leaf nodes of the tree consists of only one category.

It is clear that k for different subsets should not be fixed. However, as we all know, it is a hard problem to determine k in clustering algorithms, although there have been some papers working on it [12]. In the field of spectral clustering, the eigengap [20] is often suggested as a way to determine the number of clusters; however, it does not always work, especially for real-world complex problems [17]. With the above concerns, we adopt a classical strategy to enumerate k in our method. We try different k values ($k = 1, 2, 3, \ldots, K$, where K is the category number of the concerned subset) for clustering to get a Je-k curve, where Je is the minimum value for the objective function of a clustering algorithm. For example, Je for k-means [5] takes the form of:

$$Je = \min \sum_{i=1}^{k} \sum_{y \in \Gamma_i} \|y - \sigma_i\|^2, \tag{20}$$

where $\sigma_1, \sigma_2, \ldots, \sigma_k$ are centers of the clusters $\Gamma_1, \Gamma_2, \ldots, \Gamma_k$. Then, the very k at the inflexion point of the curve is chosen as the number of the clusters. For the above process, one may argue that the time complexity is high because of the enumeration. We agree with that, but want to point out that taxonomy mining could be an offline task. If we can get a reasonable taxonomy hierarchy and can thus improve

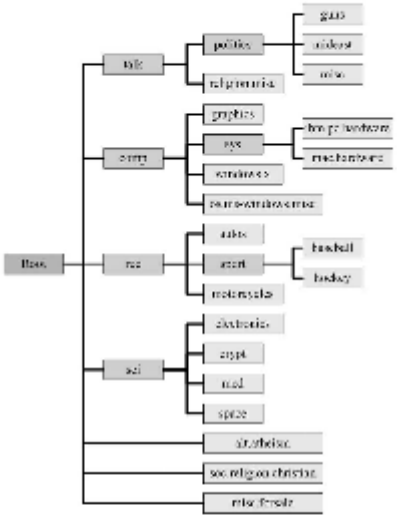


Fig. 5. The natural taxonomy of 20-newsgroups.

the classification efficiency and effectiveness by much, these additional offline data preparations are definitely worthy of being done.

After the hierarchical taxonomy is ready, we can build the hierarchical SVM classifiers. At each node of the mined taxonomy, a SVM classifier is trained to distinguish each of its child categories from other children of it. That is, the one-against-rest strategy is adopted at each node. The training process starts at the root node and covers all the categories using a breadth-first traversal of the taxonomy.

4 Experimental Results

In this section, we conducted experiments on two real data sets to show the outputs of taxonomy mining as well as the classification performance of hierarchical SVM on the mined taxonomy.

4.1 20-Newsgroups

The data set of 20-newsgroups contains about 20,000 articles evenly divided from 20 newsgroups. The human-labeled hierarchical taxonomy of 20-newsgroups is shown in Fig. 5. Our task is to mine this structure out by automatic methods.

As for the preparation, we randomly selected 100 articles per category to form the training set for taxonomy mining and classifier training. Then, we used the remaining 18,000 instances for testing. The experimental settings and external tools are summarized below:

1. For weighting, we used the method described in Section 3.1 to build the category-by-document matrix A and used term frequency to build the document-by-term matrix B. Here, we only used those terms that occur more than 10 times in the data set when generating matrix B.
2. We used the SVD and GSVD functions in the famous MATLAB (http://www.mathworks.com/) software to compute the corresponding matrix decompositions of the matrices.
3. At last, an accelerated k-means algorithm [7] was implemented to get the category clusters.
4. Microsoft Windows Server 2003 running on a workstation with 3.06GHz Xeon(TM) CPU and 3.87GB RAM was used for the experimental platform.

TABLE 1
A Mini Set Selected from 20-Newsgroups

Category Name	Notation
comp.sys.ibm.pc.hardware	C_4
comp.sys.mac.hardware	C_5
talk.politics.guns	C_{17}
talk.politics.mideast	C_{18}
talk.politics.misc	C_{19}

4.1.1 Coclustering Performance

In this section, we picked five categories (See Table 1) from 20-newsgroups to make up a mini set and randomly select 100 articles per category for coclustering. Here, we set $k = 2$ to test the coclustering performance of the four algorithms: Category-Document Bipartite spectral Graph Partitioning (CDBGP), Category-Document-Term Tripartite spectral Graph Partitioning (CDTTGP), Consistent Bipartite Spectral Graph Copartitioning (CBSGC), and Category-by-Category Confusion Matrix (CCCM) [8].

As the mini set is single-labeled, CDBGP made a random partition of the categories according to the arrangement of the documents. In our experiment, it regarded C_5 as a cluster and organized the others together. Then, we tuned different values of α to see the performance of the CDTTGP. From Table 2, we can see that the results are disappointing no matter how this parameter was set. In fact, when α is small, the algorithm degenerated to CDBGP. Otherwise, all categories were assigned to one cluster.

Comparatively, CBSGC and CCCM algorithms both exported a cluster of C_4, C_5 and another of C_{17}, C_{18}, C_{19}. The result was in accordance with our ground truth, which showed that CBSGC and CCCM outperformed the other two methods in category clustering.

4.1.2 The Mined Hierarchical Taxonomy

In this section, we ran the CBSGC algorithm on the whole data set of 20-newsgroups to mine the hierarchical taxonomy. During this process, we used the strategy described in Section 3.4 to determine the cluster number for each nonleaf node. For instance, Fig. 6 showed the *Je-k* curve for the root node, in which the number 4 at the inflexion point of the

TABLE 2
Performance under Different Values of α

α	Cluster 1	Cluster 2
1e-10	$C_4, C_5, C_{17}, C_{18}, C_{19}$	C_{17}
0.01	$C_4, C_5, C_{17}, C_{18}, C_{19}$	C_{17}
0.1	$C_4, C_5, C_{17}, C_{18}, C_{19}$	*null*
1	$C_4, C_5, C_{17}, C_{18}, C_{19}$	*null*
10	$C_4, C_5, C_{17}, C_{18}, C_{19}$	*null*
100	$C_4, C_5, C_{17}, C_{18}, C_{19}$	*null*
1e10	$C_4, C_5, C_{17}, C_{18}, C_{19}$	*null*

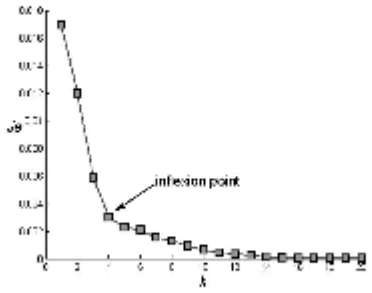


Fig. 6. The *Je-k* curve for the root node.

curve was chosen for *k*. To make the taxonomy compact and depress the computational cost, in our experiment, the recursion stopped when a subset contained no more than three categories. As a result, we mined the taxonomy hierarchy as shown in Fig. 7 for the 20-newsgroups data set.

We can see from this taxonomy that correlated categories such as *rec.**, *talk.**, *comp.**, and *sci.** were clustered together in different levels respectively except for some overlaps between *comp.** and *sci.**, and the single-handed categories such as *alt.atheism*, *misc.forsale*, and *soc.religion.christian* were separated solely from other clusters in certain levels. Further observation shows that low level categories such as *comp.sys.** and *rec.sport.** were also grouped together.

For comparison, we gave in Fig. 8 the taxonomy built in [8] by CCCM. We can see that Group 1 and Group 3 mix different kinds of categories together.

To summarize, the clusters of our mined taxonomy looked more reasonable, and the local properties of this

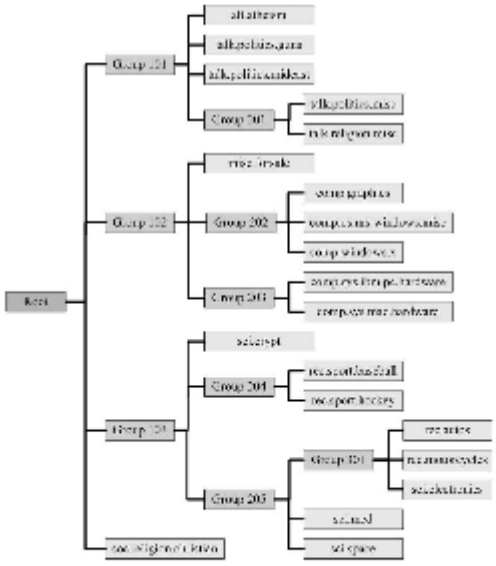


Fig. 7. The mined taxonomy of 20-newsgroups by CBSGC.

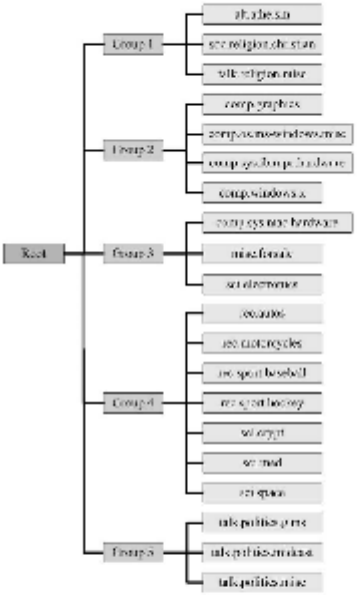


Fig. 8. The taxonomy of 20-newsgroups by CCCM.

hierarchy were more similar to those of the natural taxonomy in Fig. 5. The weakness of CBSGC is that the mined taxonomy often had more levels than the natural one.

4.1.3 Classification Performance

In this experiment, we investigated the accuracy of hierarchical classification with the mined taxonomy. As the baseline, we first constructed a flat SVM classifier. Then, we trained three hierarchical SVM classifiers according to the taxonomies shown in Fig. 8, Fig. 7, and Fig. 5, respectively. The corresponding classification results and time complexities of the above classifiers were summarized in Table 3, in which *Micro-F1* and *Macro-F1* are used for evaluating the classification performance [14].

As can be seen from Table 3, the performance of hierarchical classification based on CBSGC was better than both the flat classifier and the hierarchical classifier based on CCCM. Furthermore, we find that the classification performance of our method was very proximal to the natural hierarchical classifier. This showed in an indirect way that our taxonomy mining algorithm was very effective and the mined hierarchy was reasonable and meaningful. As one can see, the improvement brought by our method is not very significant (about 2 percent). The reason is that the 20-newsgroups corpus is somehow easy to classify. With a baseline of about 0.9 *F1* scores, the space of improvement could only be marginal.

For the time complexity of training and test, we could see that hierarchical classifiers outperformed flat classifiers significantly. Even if we further take the time of taxonomy mining (including the time spent on running generalized SVD and determining the cluster number *k*) into consideration, the overall complexity of CBSGC-based hierarchical SVM was still lower than flat SVM. It is very promising that

TABLE 3
Experimental Results of the Classifiers on 20-Newsgroups

Method	*Micro-F1*	*Macro-F1*	Training Time	Testing Time
Flat Classifier	0.902460	0.900139	2286.202s	61.777s
CCCM based	0.906655	0.906656	413.467s	29.577s
CBSGC-based	0.919102	0.920266	551.546s	29.856s
Natural Hierarchy	0.922720	0.922692	523.074s	28.148s

(The time spent on mining the taxonomy by CBSGC was 273.902s.)

TABLE 4
The Selected Subtree of the Yahoo! Directory

1st level node	2nd level nodes	3rd level nodes
Science	*Mathematics*	*Algebra, Geometry*, etc. (Total 36 nodes.)
	Physics	*Electricity, Magnetism*, etc. (Total 29 nodes)
	Chemistry	*Biochemistry, Chemists*, etc. (Total 26 nodes.)

our proposed method can improve both the effectiveness and efficiency as compared to flat SVM classification.

4.2 A Subtree of the Yahoo! Directory

In this section, we conducted our algorithm on a subtree of the Yahoo! directory. It has been discovered [16] that the distribution of the Web pages (documents) in the Yahoo! directory is seriously skewed; that is, many leaf-node categories of the taxonomy contain very few documents while several common categories consist of thousands of documents. To avoid the influence of this ill distribution and show the validity of our algorithm, we selected the *Science* subtree for our experiment. The root node of the selected corpus was *Science*, and three children *Mathematics*, *Physics*, and *Chemistry* were selected as the second level nodes. Then, all the children of these three nodes were put in the third level and all other nodes were omitted. Table 4 gave a rough view of the selected subtree. (The data was crawled in Oct. 2004.) With such a setting, this subtree contained 91 leaf-node categories and 3,073 Web pages (documents). We randomly chose 80 percent (2,451) documents for training and the rest 20 percent (622) for testing. The weighting process for matrices A and B was the same as in Section 4.1.

4.2.1 Coclustering Performance and the Mined Hierarchical Taxonomy

After applying CBSGC on it, we got the embeddings of the category nodes in Fig. 9, where the second, third, and fourth columns of the embedding matrix U^* were plotted and the ground truth of category labels were shown (children of *Mathematics*, *Physics*, and *Chemistry* were denoted by "x," "+," and "·," respectively).

Figs. 9a, 9b, 9c, and 9d showed the embeddings in different points of view. We could see that most of the nodes that belonged to the same category in level 2 were embedded very close to each other. For example, in Fig. 9d, most mathematical nodes were on the right side of the vertical axis; most physical nodes lay on the left of the vertical axis and below the horizontal axis; and most chemical nodes lay on the left of the vertical axis but over the horizontal axis. We could also see a few error-embedded nodes. For instance, the "x" point near (0.2, 0) in Fig. 9d was *Wavelets*, a child of *Mathematics*, but it seemed more close to the children of *Physics* and *Chemistry*. In some sense, this might also be reasonable because from the bag-of-word view, "wave" can be quite relevant to *Physics*.

Another finding is that there were some nodes far away from the majority, such as the "x" point near (-0.4, 0.1) in Fig. 9d, which was *Mathematics/Education*. One explanation is that these categories had much more training documents than other categories. In other words, this charges upon the skewed category distribution. If we ran k-means on the embedding data as done in Section 4.1, these isolated points would tend to be clustered as separate clusters. To tackle this problem, we used the simplest method to get the clusters, that is, we chose the zero points of different dimensions as thresholds to partition the embedding points, and mined the taxonomy described in Table 5. From this table, we can see the clustering performance is quite good: Most categories are correctly clustered to their ground-truth labels, while only a few categories are wrongly clustered.

4.2.2 Classification Performance

In this section, we examine the classification performance of the mined taxonomy as shown in Table 5. We first constructed a flat SVM classifier composed of 91 one-again-rest SVM classifiers. Then, we trained two hierarchical SVM classifiers according to the taxonomies shown in Tables 4 and 5, respectively. The corresponding classification results and time complexities of the above classifiers were summarized in Table 6, from which we could see that the performance of hierarchical classification based on CBSGC was not only higher than flat SVM, but even better than the hierarchical classifier based on the natural taxonomy. This phenomenon is so promising for automatic taxonomy mining: some local structure of the artificial taxonomy might be unsuitable for machine classification. And for the complexity, we can get very similar conclusion to that from the 20-newsgroups data set: Even if we take the time for taxonomy mining into account, the time complexity of CBSGC-based hierarchical SVM classification is still much lower than flat SVM classification.

5 Conclusion and Future Work

In this paper, our target is to prepare a hierarchical taxonomy automatically in order to applying hierarchical classification. For this purpose, we proposed a method to cluster categories, documents, and terms by consistent

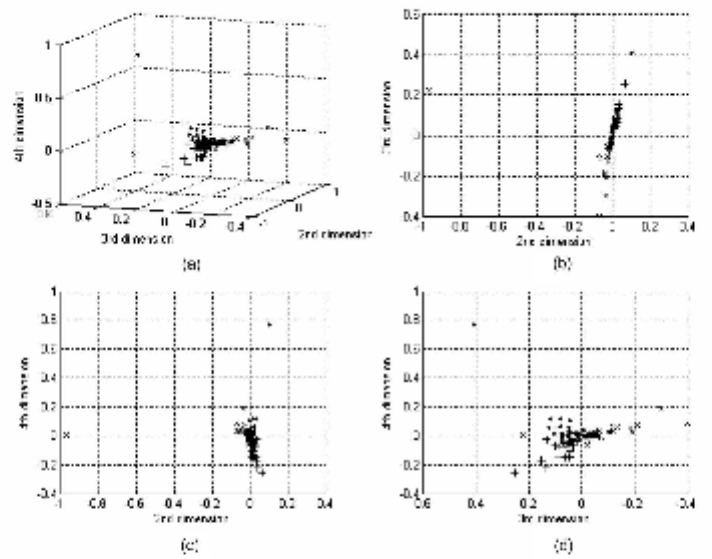


Fig. 9. The embeddings of the category nodes of the Yahoo! subtree.

TABLE 5
The Mined Subtree of the Yahoo! Directory

1st level node	2nd level nodes	3rd level nodes
Science	*Mathematics*	*Algebra, Geometry, etc.* (37 nodes with 2 errors.)
	Physics	*Electricity, Magnetism, etc.* (30 nodes with 2 errors)
	Chemistry	*Biochemistry, Chemists, etc.* (24 nodes with 3 errors.)

TABLE 6
Experimental Results of the Classifiers on Subtree of the Yahoo! Directory

Method	*Micro-F1*	*Macro-F1*	Training Time	Testing Time
Flat Classifier	0.627568	0.436808	1003.997s	9.423s
CBSGC-based	0.664601	0.481123	651.636s	5.312s
Natural Hierarchy	0.661069	0.479177	669.724s	5.093s

(The time spent on mining the taxonomy by CBSGC was 104.517s.)

bipartite spectral graph copartitioning. Compared to previous approaches, this method used more information embedded in the data corpus to assist the taxonomy mining and the hierarchical classifier training. Experimental results showed that the proposed method worked well in category clustering and could produce a very similar hierarchical taxonomy to the natural hierarchy in both the appearance and the corresponding hierarchical classification performance. For future work, we plan to provide some reasonable objective functions for the consistent copartitioning problem and design fast algorithms to get the optimal solution. We would also like to find a more efficient way to select k automatically in the process of generating the hierarchical taxonomy.

Acknowledgments

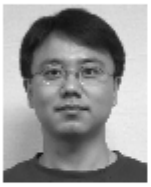
This work was done when the first, the third, and the fourth authors were interns at Microsoft Research Asia. During the formulation of this paper, Hao WAN provided the authors with the code of the hierarchical SVM classifier and helped them understand and debug it.

References

[1] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. ACM Press, 1999.
[2] I.S. Dhillon, "Coclustering Documents and Words Using Bipartite Spectral Graph Partitioning," *Proc. SIGKDD '01*, 2001.
[3] I.S. Dhillon, S. Mallela, and D.S. Modha, "Information-Theoretic Co-Clustering," *Proc. SIGKDD '03*, pp. 89-98, 2003.
[4] C. Ding, X. He, H. Zha, M. Gu, and H. Simon, "A Min-Max Cut Algorithm for Graph Partitioning and Data Clustering," *Proc. IEEE Int'l Conf. Data Mining*, 2001.
[5] R.O. Duda, P.E. Hart, and D.G. Stork, *Pattern Classification*, second ed. John Wiley & Sons Inc., 2001.
[6] S.T. Dumais and H. Chen, "Hierarchical Classification of Web Content," *Proc. SIGIR '00*, 2000.
[7] C. Elkan, "Using the Triangle Inequality to Accelerate *k*-Means," *Proc. Int'l Conf. Machine Learning*, 2003.

[8] S. Godbole, "Exploiting Confusion Matrices for Automatic Generation of Topic Hierarchies and Scaling Up Multi-Way Classifiers," technical report, IIT Bombay, 2002.
[9] S. Godbole, S. Sarawagi, and S. Chakrabarti, "Scaling Multi-Class Support Vector Machines Using Inter-Class Confusion," *Proc. SIGKDD '02*, 2002.
[10] G.H. Golub and C.F.V. Loan, *Matrix Computations*, third ed. Johns Hopkins Univ. Press, 1996.
[11] L. Hagen and A.B. Kahng, "New Spectral Methods for Ratio Cut Partitioning and Clustering," *IEEE Trans. Computer Aided Design*, vol. 11, pp. 1074-1085, 1992.
[12] G. Hamerly and C. Elkan, "Learning the k in k-Means," *Proc. Neural Information Processing Systems Conf.*, 2003.
[13] K. Kummamuru, A. Dhawale, and R. Krishnapuram, "Fuzzy Co-Clustering of Documents and Keywords," *Proc. IEEE Int'l Conf. Fuzzy Systems*, pp. 772-777, 2003.
[14] D.D. Lewis, "Evaluating Text Categorization," *Proc. Speech and Natural Language Workshop*, 1991.
[15] D.D Lewis, Y. Yang, T. Rose, and F. Li, "RCV1: A New Benchmark Collection for Text Classification Research," *J. Machine Learning Research*, vol. 5, pp. 361-397, 2004.
[16] T. Liu, Y. Yang, H. Wan, Q. Zhou, B. Gao, H. Zeng, Z. Chen, and W. Ma, "An Experimental Study on Large-Scale Web Categorization," *Proc. Int'l World Wide Web Conf.*, 2005.
[17] M. Meila and L. Xu, "Multiway Cuts and Spectral Clustering," http://www.stat.washington.edu/mmp/, 2004.
[18] F. Sebastiani, "Machine Learning in Automated Text Categorization," *ACM Computing Surveys (CSUR)*, vol. 34, no. 1, pp. 1-47, 2002.
[19] J. Shi and J. Malik, "Normalized Cuts and Image Segmentation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, pp. 888-905, 2000.
[20] G.W. Stewart and J.G. Sun, *Matrix Perturbation Theory*. Academic Press, 1990.
[21] V.N. Vapnik, *The Nature of Statistical Learning Theory*. Springer Verlag, 1995.
[22] V. Vural and J.G Dy, "A Hierarchical Method for Multi-Class Support Vector Machines," *Proc. Int'l Conf. Machine Learning*, 2004.
[23] J. Wang, H. Zeng, Z. Chen, H. Lu, L. Tao, and W. Ma, "ReCoM: Reinforcement Clustering of Multi-Type Interrelated Data Objects," *Proc. SIGIR '03*, 2003.
[24] Y. Yang, "A Scalability Analysis of Classifiers in Text Classification," *Proc. SIGIR '03*, 2003.
[25] H. Zha, C. Ding, and M. Gu, "Bipartite Graph Partitioning and Data Clustering," *Proc. Conf. Information and Knowledge Management*, 2001.

Bin Gao received the BS degree in mathematics from Shandong University, Jinan, China, in 2001. He is currently a PhD candidate at Peking University, Beijing, China. His research interests are in the areas of pattern recognition, machine learning, data mining, graph theory, and corresponding applications on text categorization and image processing.



Tie-Yan Liu received the BSc, MSc, and PhD degrees all from Tsinghua University in 1998, 2000, and 2003, respectively. After that, he joined Microsoft Research Asia as an associate researcher. His research interests include media content analysis, information retrieval, machine learning, and graph theory. He has had nearly 30 papers published in refereed conferences and journals and has served on the technical program committees, workshop committees, or as a reviewer in a dozen of international conferences. He is a member of the IEEE Computer Society.

Guang Feng received the BS degree from Tsinghua University in 2003 and he is a PhD candidate at Tsinghua University. His research interests are in the areas of machine learning, information retrieval, graph representation, and complex network theory.

Tao Qin received the BS degree from Tsinghua University, Beijing, China, in 2003. He is currently a PhD candidate at Tsinghua University. From September 2003 to June 2004, he was with Microsoft Research Asia, Beijing, as a visiting student with the Media Computing Group. Since July 2004, he has been a visiting student with Web Search and Mining Group. His current interests include machine learning, information retrieval, graph representation and learning, and complex network theory.

Qian-Sheng Cheng received the BS degree in mathematics from Peking University, Beijing, China, in 1963. He is now a professor in the Department of Information Science, School of Mathematical Sciences, Peking University, China, and he was the vice director of the Institute of Mathematics, Peking University, from 1988 to 1996. His current research interests include signal processing, time series analysis, system identification, and pattern recognition. He is the vice chairman of the Chinese Signal Processing Society and has won the Third Chinese National Natural Science Award.

Wei-Ying Ma received the BS degree in electrical engineering from the National Tsing Hua University in Taiwan in 1990 and the MS and PhD degrees in electrical and computer engineering from the University of California at Santa Barbara (UCSB) in 1994 and 1997, respectively. From 1994 to 1997, he was engaged in the Alexandria Digital Library (ADL) project in UCSB while completing a PhD. From 1997 to 2001, he was with HP Labs, where he worked in the field of multimedia adaptation and distributed media services infrastructure for mobile Internet. He joined Microsoft Research Asia in April 2001 as the research manager of the Web Search and Mining Group, leading research in the areas of information retrieval, text mining, search, multimedia management, and mobile browsing. In 2003 and 2004, his research group published nine full papers in SIGIR, five full papers in WWW, and eight full papers in ACM Multimedia conference, which accounted for 5-10 percent of the total number of accepted papers in these conferences. He currently serves as an editor for the *ACM/Springer Multimedia Systems Journal* and associate editor for the *Journal of Multimedia Tools and Applications* published by Kluwer Academic Publishers. He has served on the organizing and program committees of many international conferences including ACM Multimedia, ACM SIGIR, ACM CIKM, WWW, ICME, CVPR, SPIE Multimedia Storage and Archiving Systems, SPIE Multimedia Communication and Networking, etc. He is also the general cochair of the International Multimedia Modeling (MMM) Conference 2005 and the International Conference on Image and Video Retrieval (CIVR) 2005. He is a member of the IEEE Computer Society.

ANEXO B.

Clasificación MSC

Este anexo trata sobre la clasificación MSC. Para crear la taxonomía jerárquica en base a nuestros documentos matemáticas, vamos a utilizar categorías englobadas dentro del sistema de clasificación alfanumérica MSC, en concreto de la versión MSC2010.

Mathematics Subject Classification (MSC)

El MSC es un sistema de clasificación jerárquico alfanumérico, producido en base a la cobertura de las dos principales bases de datos de matemáticas, *Mathematical Reviews* y *Zentralblatt MATH.* Es utilizado en muchas revistas matemáticas. La versión actual es MSC2010.

El sistema consta de tres niveles de estructura, y se puede clasificar en base a dos, tres o cinco dígitos, dependiendo del número de niveles que se utilicen para clasificar

El primer nivel está representado por un número de dos dígitos, el segundo por una letra, y el tercero por otro número de dos dígitos

Primer nivel

En el nivel superior existen 64 disciplinas matemáticas etiquetadas con un número único de 2 dígitos. En este nivel además de encontrarse las áreas básicas de la investigación matemática, están las categorías de "Historia y Biografía", "Educación Matemática", y categorías con similitudes con diferentes ciencias.

Segundo nivel

Los códigos del segundo nivel se representan son una sola letra del alfabeto. Representan áreas específicas relacionadas por el primer nivel de la disciplina. Los códigos de segundo nivel pueden variar de una disciplina a otra.

Además en este nivel se encuentra el código especial “-“, el cual se utiliza para tipos específicos de materiales

Tercer nivel

Los códigos del tercer nivel son los más específicos, normalmente suelen corresponder a un tipo específico de objeto matemático o a un problema conocido o a una área de investigación.

En la tabla siguiente vamos a agrupar las categorías de primer nivel según nombres de áreas que les son comunes

Área	Categorías
General/Fundaciones	00 : General
	01: Historia y Biografía
	03: Lógica Matemática y Fundaciones
Matemática Discreta/Algebra	05: Combinatoria
	06: Teoría del orden
	08: Sistemas generales de álgebra
	11: Teoría numérica
	12: Teoría de campos y polinomiales
	13: Anillo conmutativos y Algebras
	14: Algebra geométrica
	15: Algebra lineal y multilineal; teoría de matrices
	16: Anillos y algebras asociativas
	17: Anillos y algebras no-asociativos
	18: Teorías de categorías; Algebra de homologías
	19: K-teoría
	20: Teoría de grupos y generalizaciones
	22: Grupos topológicos, grupos Lie y análisis
Análisis	26: Funciones Reales, incluye derivadas e integrales
	28: Medida e integración
	30: Funciones complejas: teoría de aproximación en el dominio complejo
	31: Teoría Potencial
	32: Varias variables complejos y espacios analíticos
	33: Funciones especiales
	34: Ecuaciones diferenciales ordinarias
	35: Ecuaciones diferenciales parciales
	37: Sistemas dinámicos y teoría ergódica
	39: Diferencia de ecuaciones y ecuaciones funcionales
	40: Secuencias, series, sumabilidad
	41: Aproximaciones y expansiones
	42: Sistemas armónicos
	43: Análisis de armónicos abstractos
	44: Transformaciones integrales, calculo operacional
	45: Ecuaciones Integrales
	46: Análisis funcional
	47: Teoría operadora
	49: Cálculos de variaciones y control optimo, optimización

Geometría y Topología	51: Geometría
	52: Geometría convexa y discreta
	53: Geometría diferencial
	54: Geometría topológica
	55: Topología Algebraica
	57: Múltiples
	58: Análisis Global, análisis múltiples
Matemática Aplicada /Otros	60: teoría de la probabilidad y sistemas estocásticos
	62: Estadística
	65: Análisis numérico
	68: Ciencias de la computación
	70: Mecánica
	74: Mecánica de sólidos deformables
	76: Fluidos mecánicos
	78: Óptica, teoría electromagnética
	80: Termodinámica clásica, transferencia de calor
	81: Teoría cuántica
	82: Mecánica estadística, estructura de materia
	83: Relatividad y teoría gravitacional
	85: Astronomía y Astrofísica
	86: Geofísica
	90: Investigación de operaciones, programación matemática
	91: Teoría de Juegos, economía, social y ciencias del comportamiento
	92: Biología y otras ciencias naturales
	93: Teoría de sistemas, control, control óptimo
	94: Información y comunicación, circuitos
	97: Educación sobre las matemáticas

Tabla B1: Categorías MSC

Después de analizar las categorías, se ha observado, que dentro de cada categoría, existen categorías del tipo XX – {00, 01, 02, 03, 04,06}, que son comunes a todas ellas

XX-00 Trabajos de referencias generales

XX-01 Exposiciones de enseñanza

XX-02 Exposición de investigación

XX-03 Histórico

XX-04 Computación explícita de máquinas y programas

XX-06 Actas, conferencias, colecciones, etc...

ANEXO C.

SVD y GSVD

En este anexo está formado por dos teoremas pertenecientes al problema de **S**ingular **V**alue **D**ecomposition y al de **G**eneralized **S**ingular **V**alue **D**ecomposition. Contiene la explicación, con un ejemplo, de por qué no se puede aplicar SVD en nuestro problema, y trata más en profundidad el por qué de que sea necesario, tras aplicar GSVD, hacer transformaciones en las matrices.

SVD

Teorema: $If\ we\ have\ A \in\ R^{mxn}\ , m \geq n, then\ exists\ unitary\ matrices\ U \in R^{mxm}\ , V \in R^{nxn}\ and\ diagonal\ matriz\ \Sigma^{mxn}$

$$A = U\,\Sigma\,V^T$$

$$where\ (\sigma_1 \geq\ \sigma_2 \geq \cdots \geq \sigma_n > 0)\ are\ the\ singular\ values\ of\ A\ and\ \ \Sigma = \ diag\ (\sigma_1, \sigma_2, \ldots, \sigma_n),$$

Figura C1: Teorema SVD

El teorema anterior pertenece al problema de Singular Value Decomposition. Como hemos comentado en la memoria, no se puede aplicar únicamente en nuestro algoritmo para realizar la integración de categorías y términos.

En este anexo vamos a ver un ejemplo de por qué SVD no garantiza consistencia en nuestro caso.

Aplicamos SVD a unas matrices $\hat{A}$ y $\hat{B}$ de ejemplo, sabiendo que la primera representa el coclustering entre documentos y términos, y la segunda el coclustering de categorías-documentos; la matriz unitaria U_A integrará la información de clustering de categorías, las matrices $V_A\ y\ U_B$ integrarán la información de clustering de documentos, y la matriz V_B integrará el clustering de términos.

Para que existe una coparticion consistente, debería resultar el mismo particionamiento de documentos en ambos grafos, es decir, garantizar que los resultados de clustering correspondientes a las columnas de $V_A\ y\ U_B$ sean los mismos.

Suponemos A y B son de la siguiente forma:

$$A = \begin{matrix} \\ c_1 \\ c_2 \\ c_3 \end{matrix} \begin{matrix} \begin{matrix} d_1 & d_2 & d_3 & d_4 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{matrix}, \quad B = \begin{matrix} \\ d_1 \\ d_2 \\ d_3 \\ d_4 \end{matrix} \begin{matrix} \begin{matrix} t_1 & t_2 & t_3 & t_4 & t_5 \end{matrix} \\ \begin{bmatrix} 4 & 3 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 & 0 \\ 0 & 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 4 & 3 \end{bmatrix} \end{matrix}$$

Figura C2: Ejemplo matriz A y matriz B

Construimos $\hat{A}$ y $\hat{B}$ y aplicamos SVD, obteniendo:

$$V_A = \begin{bmatrix} 0.70711 & 0 & 0 & -0.70711 \\ 0.70711 & 0 & 0 & 0.70711 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

$$U_B = \begin{bmatrix} 0.70711 & 0 & 0 & -0.70711 \\ 0.70711 & 0 & 0 & 0.70711 \\ 0 & 0.70711 & -0.70711 & 0 \\ 0 & 0.70711 & 0.70711 & 0 \end{bmatrix}$$

Figura C3: Matrices resultantes V_A y U_B

Podemos ver que el clustering de documentos no es el mismo en los dos casos, por lo que los resultados no son consistentes.

GSVD

Teorema: *If we have* $\hat{A} \in R^{mxn}$ *and* $\hat{B} \in R^{nxt}, m \leq n \leq t$, *then there exists unitary matrices* $U \in R^{mxm}$, $V \in R^{txt}$ *and invertible matrix* $X \in R^{nxn}$ *such that*:

$$\begin{cases} \hat{A} = U\ C\ X^T \\ \hat{B} = X\ S\ V^T \end{cases}$$

$$where\ C = diag\ (c_1, c_2, \dots, c_n), c_i \geq 0\ and\ S = diag\ (s_1, s_2, \dots, s_n), s_i \geq 0$$

Figura C4: Teorema GSVD

El teorema indica que existe un particionamiento de documentos, embebido en *X*, que reúne las restricciones de ambos grafos. El particionamiento correspondiente de categorías y términos, esta embebido en *U* y *V* respectivamente.

X es una matriz inversible, lo que es una condición mucho menos restrictiva comparándola con SVD, la cual es unitaria. Además, la información contenida en *X* no está explicita como en V_A. Mientras que el segundo vector singular de V_A tiene embebido el clustering de documentos, en el caso de la matriz *X*, a esta información solo se puede acceder considerando algunas columnas no ortogonales. Es por esta razón, por la que para hay que extraer esta información útil de alguna manera. Esto se consigue encontrando una transformación lineal adecuada.

ANEXO D.

Corpus de Datos

En este anexo complementa al apartado “Corpus de Datos” de la memoria. Se va a explicar cómo es el fichero que contiene el corpus de datos, qué formato tiene, qué información nos aporta y cómo lo vamos a leer para poder trabajar en MATLAB con él.

Descripción del Corpus

Tal y cómo se ha mencionado en la memoria, el propósito de este PFC es ayudar a la organización de documentos matemáticos; para realizar esta labor hay que trabajar con el contenido de estos, ya que analizando la información contenida en sus términos, podremos ser capaces de extraer la información necesaria para poder crear una estructura jerárquica por categorías (MSC, Mathematics Subject Classification).

Para realizar esta labor, dispongo de tres ficheros con documentos, los tres describen el corpus de datos con el que voy a trabajar, aunque representan su contenido de maneras diferentes. El primero contiene las descripciones de los documentos, el segundo representa las ocurrencias por términos de cada documento y el último contiene las ocurrencias de cada documento pero en formato normalizado. Para aplicar el algoritmo con el que he trabajado, únicamente he usado el contenido del último de estos tres ficheros: “*exp-train_proj-abs_vec-conf9-min-3.json”,* ya que tiene la información que necesito en formato normalizado.

Estructura del fichero

Cada línea del fichero es una estructura JSON que representa a un documento. Esta estructura consta de dos partes, en la primera encontramos la *metadata*, compuesta por un identificador de documento y el identificador de las categorías a las que pertenece en formato MSC, y en la segunda, el contenido del documento, representado por identificadores de términos y sus ocurrencias. Será la forma de representar esta segunda parte de la estructura lo que va a variar de un fichero a otro.

En el fichero *“exp-train_proj-abs.json”* aparece en formato de texto, en cambio en los otros dos aparecen los términos denotados con identificadores junto a las ocurrencias

de cada uno, en *"exp-train_proj-abs_vec-conf9-min-3.json"* el número de ocurrencias, y en *"exp-train_proj-abs_vec-conf9-min-3_norm.json"* el número de ocurrencias normalizadas, teniendo en cuenta toda la información del resto de documentos.

A continuación se muestra para cada fichero, la estructura de los documentos.

Fichero "exp-train_proj-abs.json"

```
[
	[
		"1039.35023", %Identificador Documento
		["35B45","35B65","35J25"] %categorías a las que pertenece
	],
	[
		"The authors consider the following elliptic problem $$
		\\lambda    u(x)-\\Delta u(x)-Lu(x)=f(x), \\ x\\in
		S_d,\\quad D_i^2u(x)=0,\\     x\\in
		S_d\\cap\\{x_i=0,1\\},\\ i=1,\\ldots,d $$ where $S_d$ is
		the $d$-dimensional hypercube $[0,1]^d,$ $\\lambda$ is a
		positive constant and $L$ is a second order differential
		operator. They prove that if the coefficients are of class
		$C^{k+\\delta}(S_d),$ with  $k=0,1$  and  $\\delta\\in(0,1),$
		then the problem admits a unique solution $u$ belonging to
		$C^{k+2+\\delta}(S_d).$"
	]
	 %Contenido del documento en formato de texto
]
```

Fichero *"exp-train_proj-abs_vec-conf9-min-3.json"*

```
[
	[
		"1039.35023",%Identificador de documento
		["35B45","35B65","35J25"] %categorías a las que pertenece
	],
	{
		"8":1.0, %Identificador término : Ocurrencia en documento
		"3370":1.0,"5161":1.0,"5866":1.0,"7805":1.0,"8492":1.0,"8799"
		:1.0,"9225":1.0,"10970":1.0,"11188":1.0,"11566":1.0,"11583":1
		.0,"12915":1.0,"12964":1.0,"13893":1.0,"15191":1.0,"17269":1.
		0,"17361":3.0,"23037":1.0,"23096":1.0,"24444":1.0,"24807":2.0
		,"24959":1.0,"27119":1.0,"28302":1.0,"30075":1.0,"30077":1.0,
		"31345":1.0
	}
	% Lista de términos con sus ocurrencias

]
```

Fichero *“exp-train_proj-abs_vec-conf9-min-3_norm.json”*

```
[
	[
		"1039.35023",%Identificador de documento
		["35B45","35B65","35J25"] %categorías a las que pertenece

	],
	{
		"8":0.15249857033260467, % Identificador término
		                         % Ocurrencias normalizadas
		"1765":0.15249857033260467,"1850":0.15249857033260467,"3108":
		0.15249857033260467,"3309":0.15249857033260467,"3370":0.15249
		857033260467,"5161":0.15249857033260467,"5866":0.152498570332
		60467,"7805":0.15249857033260467,"8492":0.15249857033260467,"
		8799":0.15249857033260467,"9225":0.15249857033260467,"10970":
		0.15249857033260467,"11188":0.15249857033260467,"11566":0.152
		49857033260467,"11583":0.15249857033260467,"12915":0.15249857
		033260467,"12964":0.15249857033260467,"13893":0.1524985703326
		0467,"15191":0.15249857033260467,"17269":0.15249857033260467,
		"17361":0.457495710997814,"23037":0.15249857033260467,"23096"
		:0.15249857033260467,"24444":0.15249857033260467,"24807":0.30
		499714066520933,"24959":0.15249857033260467,"27119":0.1524985
		7033260467,"28302":0.15249857033260467,"30075":0.152498570332
		60467,"30077":0.15249857033260467,"31345":0.15249857033260467
	}
	% Lista de términos con sus ocurrencias normalizadas

]
```

El contenido del fichero con el que se ha trabajado sería semejante al que se muestra a continuación (contenido del fichero “*prueba.json”,* el cual ha servido como punto de partida para el trabajo con el corpus de datos real)

```
[["1039.35023",["35B45","35B65","35J25"]],{"8":0.15249857033260467,"13":0
.15249857033260467}]
[["1043.83040",["83F05","83C75","83C15"]],{"9":0.13801311186847084,"1":0.
06900655593423542,"3":0.13801311186847084}]
[["1047.65001",["35B45","35B65","83F05","83C75"]],{"9":0.4082482904638631
,"13":0.4082482904638631}]
[["1048.5048",["35J25"]],{"2":0.19245008972987526,"5":0.19245008972987526
,"9":0.19245008972987526,"12":0.19245008972987526}]
[["1049.14014",["35B45","35B65","35J25","14K22"]],{"6":0.2236067977499789
6,"4":0.22360679774997896}]
[["1049.14032",["83F05","83C75"]],{"7":0.20412414523193154,"10":0.2041241
4523193154,"11":0.20412414523193154}]
[["1052.14069",["83F05","83C75","83C15"]],{"7":0.10273309938750283,"1":0.
10273309938750283,"2":0.10273309938750283}]
[["1053.65002",["35B45","35B65"]],{"2":0.4082482904638631,"12":0.40824829
04638631,"13":0.4082482904638631}]
[["1054.5099",["35B45","35B65","83F05"]],{"11":0.040291148201269014,"1":0
.040291148201269014,"2":0.16116459280507606}]
[["1054.62005",["35B45","35B65","83F05"]],{"9":0.22086305214969307,"4":0.
3312945782245396,"3":0.110431526074846 53}]
```

Este corpus de datos está compuesto por 10 documentos, 7 categorías y 13 términos

Documentos =

```
    {'1039.35023'    '1043.83040'    '1047.65001'    '1048.5048'
'1049.14014'    '1049.14032'    '1052.14069'    '1053.65002'
'1054.50990'    '1054.62005'}
```

Categorías =

```
    {'35B45'  '35B65'  '35J25'  '83F05'  '83C75'  '83C15'   '14K22'}
```

Términos =

```
    {1    2    3    4    5    6    7    8    9   10   11   12    13}
```

Para obtener la información anterior de manera automática se ha tenido que leer el fichero, creando para ello estructuras en Matlab y vectores, Todo esto está explicado en el apartado de Diseño de esta memoria.

ANEXO E.

Clustering-Kmeans

Este anexo trata sobre el clustering con kmeans. Está formado por dos apartados, en el primero encontramos la definición de kmeans y la función en MATLAB, en el segundo apartado hemos realizado pruebas para decidir qué parámetros de esta función son los mejores para aplicarlos a nuestra implementación.

Definición

Para la obtención de la partición de las categorías y de los términos, vamos a hacer uso de la función ***k-means*** que nos ofrece MATLAB. *k -means* es un método de agrupamiento, que tiene como objetivo la partición de un conjunto *n* en *k* grupos en el que cada observación pertenece al grupo más cercano a la media.

Si buscamos la descripción de k-means en MATLAB, obtendremos lo siguiente:

Syntax [IDX,C,sumd,D] = kmeans(X,k)

Description IDX = kmeans(X,k) partitions the points in the n-by-p data matrix X into k clusters. This iterative partitioning minimizes the sum, over all clusters, of the within-cluster sums of point-to-cluster-centroid distances. Rows of X correspond to points, columns correspond to variables. kmeans returns an n-by-1 vector IDX containing the cluster indices of each point. By default, kmeans uses squared Euclidean distances. When X is a vector, kmeans treats it as an n-by-1 data matrix, regardless of its orientation.

[IDX,C] = kmeans(X,k) returns the k cluster centroid locations in the k-by-p matrix C.

[IDX,C,sumd] = kmeans(X,k) returns the within-cluster sums of point-to-centroid distances in the 1-by-k vector sumd.

[IDX,C,sumd,D] = kmeans(X,k) returns distances from each point to every centroid in the n-by-k matrix D.

Parámetros

Parameter	Value	
'distance'	Distance measure, in p-dimensional space. kmeans minimizes with respect to this parameter.kmeans computes centroid clusters differently for the different supported distance measures.	
	'sqEuclidean'	Squared Euclidean distance (default). Each centroid is the mean of the points in that cluster.
	'cityblock'	Sum of absolute differences, i.e., the L1 distance. Each centroid is the component-wise median of the points in that cluster.
	'cosine'	One minus the cosine of the included angle between points (treated as vectors). Each centroid is the mean of the points in that cluster, after normalizing those points to unit Euclidean length.
	'correlation'	One minus the sample correlation between points (treated as sequences of values). Each centroid is the component-wise mean of the points in that cluster, after centering and normalizing those points to zero mean and unit standard deviation.
	'Hamming'	Percentage of bits that differ (only suitable for binary data). Each centroid is the component-wise median of points in that cluster.
'emptyaction'	Action to take if a cluster loses all its member observations.	
	'error'	Treat an empty cluster as an error (default).
	'drop'	Remove any clusters that become empty. kmeans sets the corresponding return values in C and D to NaN.
	'singleton'	Create a new cluster consisting of the one point furthest from its centroid.
'onlinephase'	Flag indicating whether kmeans should perform an online update phase in addition to a batch update phase. The online phase can be time consuming for large data sets, but guarantees a solution that is a local minimum of the distance criterion, that is, a partition of the data where moving any single point to a different cluster increases the total sum of distances.	
	'on'	Perform online update (default).
	'off'	Do not perform online update.
'options'	Structure specifying options for the iterative algorithm used to	

Parameter	Value	
	minimize the fitting criteria. Create the options structure with statset. Applicable statset parameters are:	
	Display	Level of display output. Choices are 'off'(default), 'iter', and'final'.
	MaxIter	Maximum number of iterations allowed. The default is 100.
	UseParallel	If true and if a matlabpool of the Parallel Computing Toolbox™ is open, compute in parallel. If the Parallel Computing Toolbox is not installed, or a matlabpool is not open, computation occurs in serial mode. Default is default, meaning serial computation.
	UseSubstreams	Set to true to compute in parallel in a reproducible fashion. Default is false. To compute reproducibly, set Streams to a type allowing substreams: 'mlfg6331_64' or 'mrg32k3a'.
	Streams	A RandStream object or cell array of such objects. If you do not specify Streams, kmeans uses the default stream or streams. If you choose to specify Streams, use a single object except in the case: You have an open MATLAB® pool UseParallel is true UseSubstreams is false In that case, use a cell array the same size as the MATLAB pool. If a MATLAB pool is not open, then Streams must supply a single random number stream.
'replicates'	Number of times to repeat the clustering, each with a new set of initial cluster centroid positions.kmeans returns the solution with the lowest value for sumd. You can supply 'replicates' implicitly by supplying a 3D array as the value for the 'start' parameter.	
'start'	Method used to choose the initial cluster centroid positions, sometimes known as *seeds*.	
	'sample'	Select k observations from X at random (default).
	'uniform'	Select k points uniformly at random from the range of X. Not valid with Hamming distance.
	'cluster'	Perform a preliminary clustering phase on a random 10% subsample of X. This preliminary phase is

Parameter	Value	
		itself initialized using 'sample'.
	Matrix	k-by-p matrix of centroid starting locations. In this case, you can pass in [] for k, and kmeans infers k from the first dimension of the matrix. You can also supply a 3-D array, implying a value for the'replicates' parameter from the array's third dimension.

Tabla G1: Parámetros función kmeans en Matlab

Para la elección de los parámetros correctos en nuestros datos, se ha probado con diferentes combinaciones hasta obtener la partición deseada.

Las pruebas se han hecho sobre el corpus de datos de prueba, donde las categorías son las siguientes:`{'35B45' '35B65' '35J25' '83F05' '83C75' '83C15' '14K22'}` y donde sabemos que la taxonomía resultante es:

root		
35	83	14
'35B45''35B65''35J25'	'83F05''83C75''83C15'	'14K22'

Tabla G2: Taxonomía jerárquica de ejemplo

Pruebas

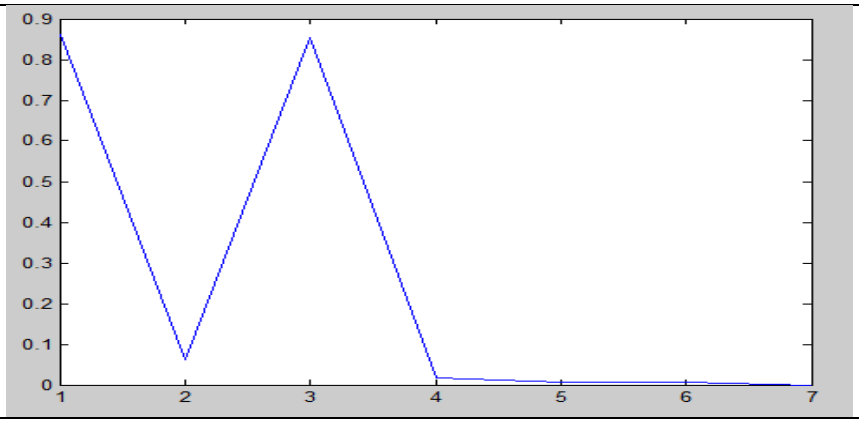
El primer parámetro a modificar va a ser '*distance'*, ya que es el parámetro más restrictivo, lo que nos va a facilitar futuras combinaciones. Cada uno de los posibles métodos a aplicar realizan la medida de las distancias entre puntos, pero con diferentes técnicas: Distancia Euclídea, City Block, Cosenos, Correlation y Hamming.

Para representar los datos se han creado dos tipos de tablas; en la primera se representan los valores que tiene cada uno de los parámetros que están siendo utilizados cuando ejecutamos la función kmeans de Matlab, y en la segunda mostramos para cada valor de K (que será el número de clusters en el que queremos k-particionar nuestros datos) su vector IDX, el cual representa los índices de pertenencia de cada dato, es decir, el contenido de este vector nos indica a qué cluster pertenecerán los datos; el valor de

Je, que es el valor mínimo de la función objetivo del algoritmo y la gráfica K-Je, que nos servirá para calcular el valor óptimo de K.

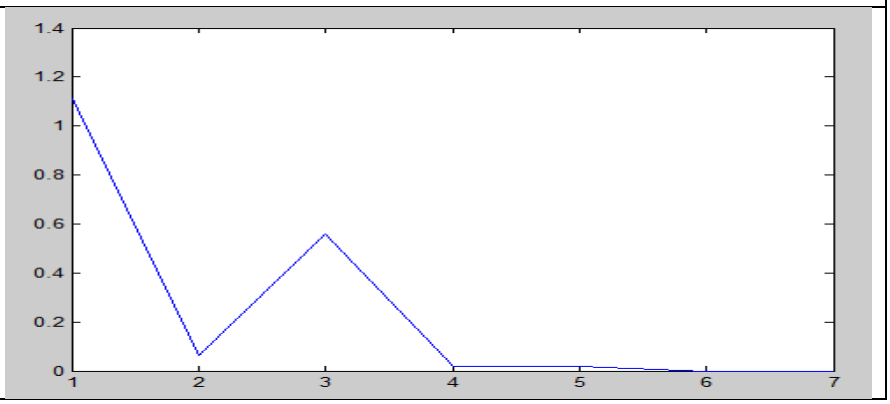
Distancia Euclídea

Parámetros	Valor
Distance	**'sqEuclidean'**
Emptyaction	'error'
Onlinephase	'on'
options	'Display''off'
replicates	0
start	'sample'

K	1	2	3	4	5	6	7
IDX	1 1 1 1 1 1 1	2 2 2 1 1 1 2	2 2 1 3 3 3 2	1 1 4 2 2 2 3	1 4 5 3 3 3 2	3 3 2 4 1 5 6	2 1 4 5 6 3 7
Je	0.8631	0.0612	0.8545	0.0163	0.0179	0.0000	0
Gráfica k-Je							

City Block

Parámetros	Valor
Distance	**'cityblock'**
Emptyaction	'error'
Onlinephase	'on'
options	'Display''off'
replicates	0
start	'sample'

K	1	2	3	4	5	6	7
IDX	1 1 1 1 1 1 1	2 2 2 1 1 1 2	2 2 2 3 3 3 1	1 1 4 2 2 2 3	1 4 5 3 3 3 2	3 3 2 4 1 5 6	2 1 4 5 6 3 7
Je	1.1106	0.0653	0.5597	0.0180	0.0195	0.0000	0
Gráfica k-Je							

Cosine

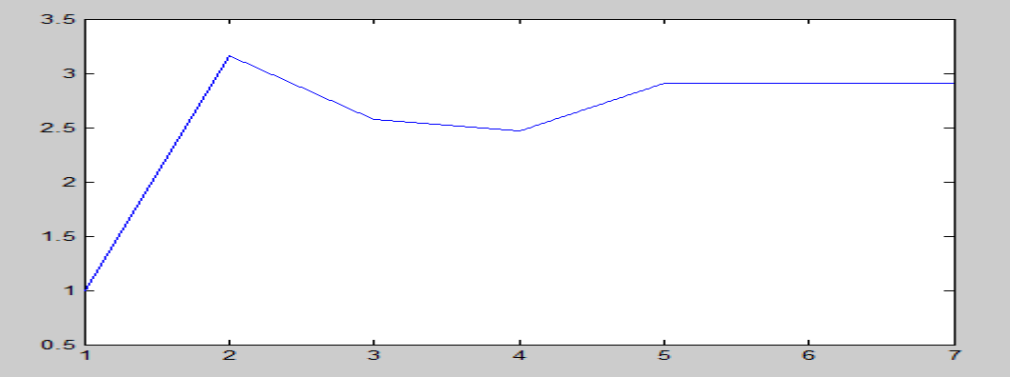
Parámetros	Valor
Distance	**'cosine'**
Emptyaction	'error'
Onlinephase	'on'
options	'Display''off'
replicates	>10
start	'sample'

Error using kmeans (line 323)
An empty cluster error occurred in every replicate.

Error in JeCurve (line 29)
[idxa,Ca] =
kmeans(wa,i,'Distance','cosine','Replicates',20);

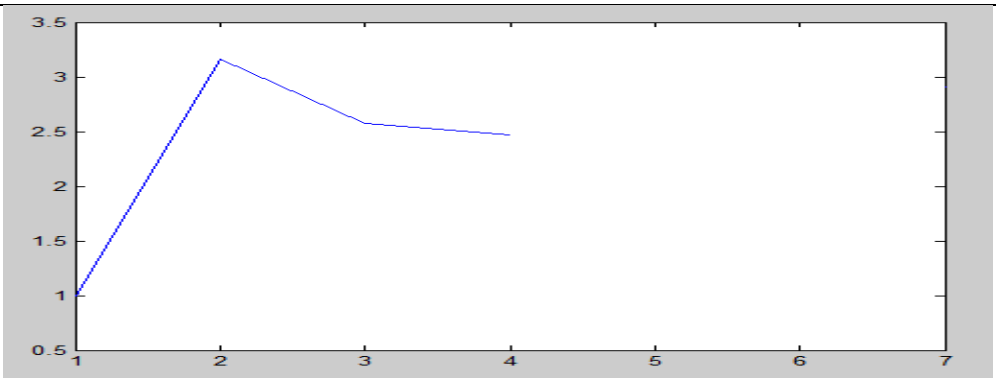
El problema es que al calcular la distancia por cosenos hay clusters que no se pueden crear, y al tener como '*Emptyaction*" '*error'*, provoca que los clústers vacíos los trate como error; para solucionar esto tenemos que modificar la '*Emptyaction'* por '*drop'* o '*singleton'*; con *drop* lo que se hace es eliminar cualquier cluster que sea vacío, devolviendo NaN, *singleton* lo que hace es crear un nuevo clúster lo más distante del centroide

Parámetros	Valor
Distance	**'cosine'**
Emptyaction	'drop' 'singleton'
Onlinephase	'on'
options	'Display''off'
replicates	0
start	'sample' 'uniform'

K	1	2	3	4	5	6	7
IDX	1 1 1 1 1 1 1	2 2 2 1 1 1 2	3 3 1 2 2 2 3	1 1 4 2 2 2 3	2 2 5 3 1 1 4	4 4 1 2 5 5 3	1 1 4 6 2 2 3
Je	0.9923	3.1595	2.5733	2.4660	2.9061	2.9061	2.9074
Gráfica k-Je							

Parámetros	Valor
Distance	**'cosine'**
Emptyaction	'drop'
Onlinephase	'on'
options	'Display''off'
replicates	>5
start	'sample'

K	1	2	3	4	5	6	7
IDX	1 1 1 1 1 1 1	2 2 2 1 1 1 2	3 3 1 2 2 2 3	1 1 4 2 2 2 3	2 2 5 3 1 1 4	5 5 6 1 2 3 4	1 1 7 6 2 1 5
Je	0.9923	3.1595	2.5733	2.4660	NaN	NaN	2.9074

Gráfica k-Je	

Correlation

Parámetros	Valor
Distance	**'correlation'**
Emptyaction	'error' 'drop' 'singleton'
Onlinephase	'on'
options	'Display''off'
replicates	[0 .. N]
start	'sample' 'uniform'

Error using kmeans (line 152)
Some points have small relative standard deviations, making them effectively constant.
Either remove those points, or choose a distance other than 'correlation'.

No se puede usar la el parámetro '*distance'* con la opción '*correlation'*, debido a que los puntos muestran una desviación pequeña, lo que ocasiona valores constantes.

Hamming

Parámetros	Valor
Distance	**'Hamming'**
Emptyaction	'error' 'drop'
Onlinephase	'on'
options	'Display''off'
replicates	0
start	'sample'

Non-binary data cannot be clustered using Hamming distance.

No se puede usar el parámetro '*distance'* con la opción '*Hamming'*, ya que los datos no están en el formato que este método necesita (necesita datos en formato binario)

Después de aplicar las distintas técnicas disponibles para el cálculo de la distancia en un espacio p-dimensional, se ha concluido que las dos técnicas que nos dan unos

resultados cercanos a lo que queremos son '*sqEuclidean'* y '*cityblock'*. Con ellas dos vamos a probar el resto de parámetros hasta conseguir aplicar kmeans de la forma óptima para nuestros datos.

A continuación vamos a modificar el número de REPLICATES, esto es, el número de veces que tiene que repetir el clustering. Es importante que el valor de replicates sea mayor que cero, ya que cuanto mayor sea este valor, mejor será la aproximación a la solución óptima.

Distancia Euclídea

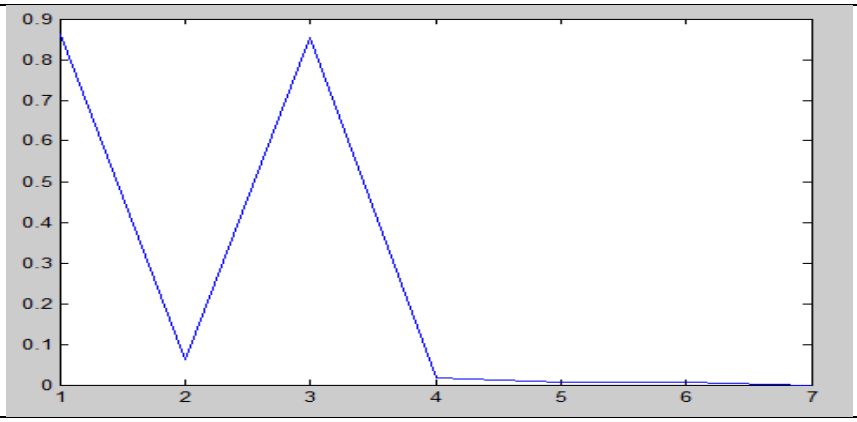
Parámetros	Valor
Distance	**'sqEuclidean'**
Emptyaction	'error' 'sample' 'singleton'
Onlinephase	'on'
options	'Display''off'
replicates	**0**
start	'sample'

Si el número de replicados es cero, vamos a obtener diferentes soluciones cada vez que ejecutemos el método. Esto se debe a la forma que tiene kmeans de calcular los clusters, en cada iteración lo que hace es elegir un conjunto nuevo inicial de centroides; por lo cual si no le decimos que repita el clustering N veces, el resultado obtenido probablemente no será el deseado.

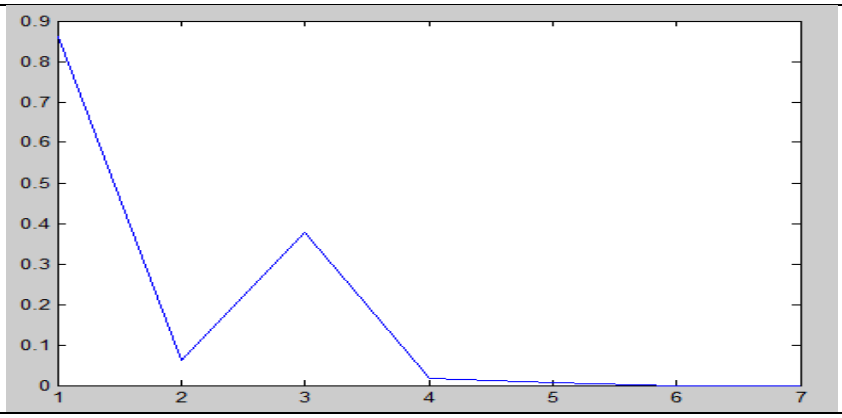
Vamos a comprobar este hecho experimentalmente, ejecutando para nuestros datos de prueba kmeans con Distancia Euclídea, con cero repeticiones; se han obtenido cuatro resultados diferentes para los mismos datos.

Resultado 1

K	1	2	3	4	5	6	7
IDX	1 1 1 1 1 1 1	2 2 2 1 1 1 2	2 2 1 3 3 3 2	1 1 4 2 2 2 3	1 4 5 3 3 3 2	3 3 2 4 1 5 6	2 1 4 5 6 3 7
Je	0.8631	0.0612	0.8545	0.0163	0.0179	0.0000	0

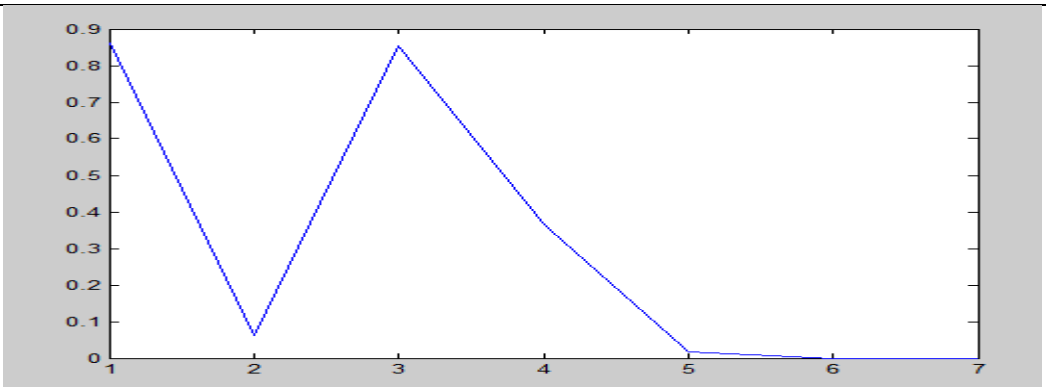
Gráfica k-Je	

Resultado 2

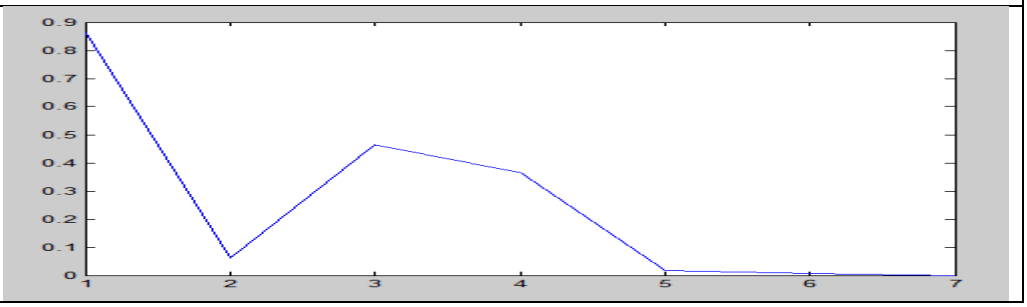
K	1	2	3	4	5	6	7
IDX	1 1 1 1 1 1 1	2 2 2 1 1 1 2	3 3 3 2 2 2 1	1 1 4 2 2 2 3	1 1 2 5 3 3 4	3 3 2 4 1 5 6	2 1 4 5 6 3 7
Je	0.8631	0.0612	0.3775	0.0163	0.0057	0.0000	0
Gráfica k-Je							

Resultado 3

K	1	2	3	4	5	6	7
IDX	1 1 1 1 1 1 1	2 2 2 1 1 1 2	1 1 3 2 2 2 1	1 1 1 3 4 4 3	5 1 4 3 3 3 2	3 3 2 4 1 5 6	2 1 4 5 6 3 7
Je	0.8631	0.0612	0.8545	0.3653	0.0179	0.0000	0

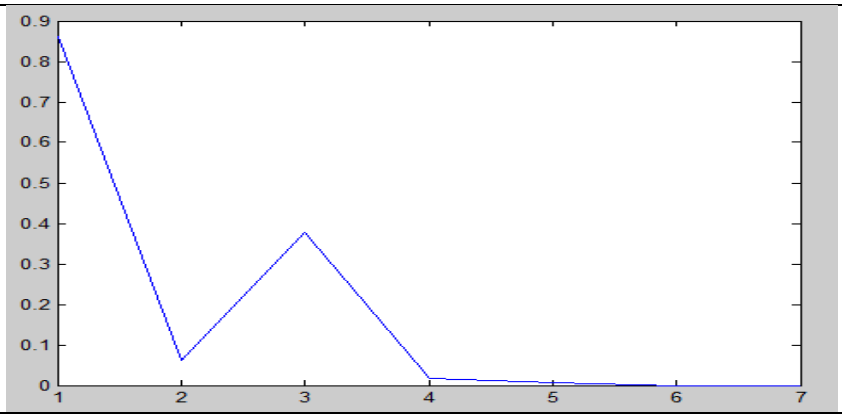
Gráfica k-Je	

Resultado 4

K	1	2	3	4	5	6	7
IDX	1 1 1 1 1 1 1	2 2 2 1 1 1 2	2 2 1 2 2 2 3	4 4 4 2 1 1 3	3 1 2 4 4 4 5	2 5 3 4 6 6 1	2 1 4 5 6 3 7
Je	0.8631	0.0612	0.4641	0.3653	0.0179	0.0000	0
Gráfica k-Je							

En cambio, si fijamos el valor de *replicates* a un número superior a dos, el resultado obtenido siempre va a ser el mismo. En nuestro caso coincide con el *resultado 2*, cuando el valor de *replicates* era cero.

Parámetros	Valor
Distance	'sqEuclidean'
Emptyaction	'error' 'drop' 'singleton'
Onlinephase	'on'
options	'Display''off'
replicates	**>2**
start	'sample'

K	1	2	3	4	5	6	7
IDX	1 1 1 1 1 1 1	2 2 2 1 1 1 2	3 3 3 2 2 2 1	1 1 4 2 2 2 3	1 1 2 5 3 3 4	3 3 2 4 1 5 6	2 1 4 5 6 3 7
Je	0.8631	0.0612	0.3775	0.0163	0.0057	0.0000	0
Gráfica k-Je							

City Block

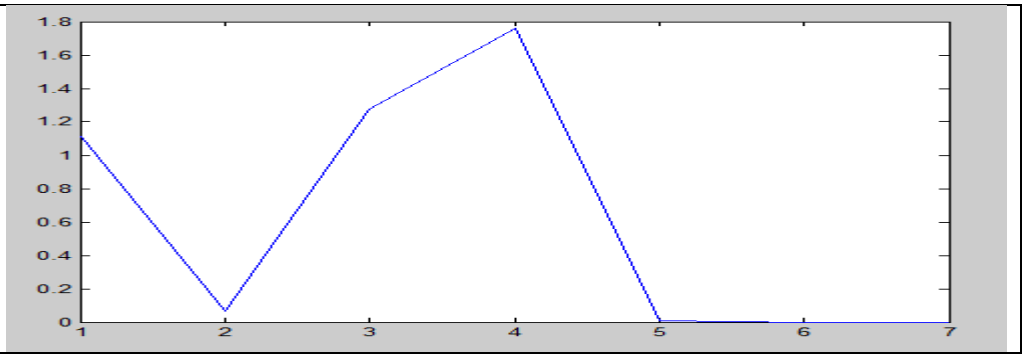
Para la medida City Block vamos a realizar los mismos pasos que hemos hecho con la Distancia Euclídea, y veremos que los resultados son similares.

Si el valor de *replicates* es cero, se obtienen muchas soluciones diferentes, en cambio si fijamos su valor, obtenemos la solución correcta.

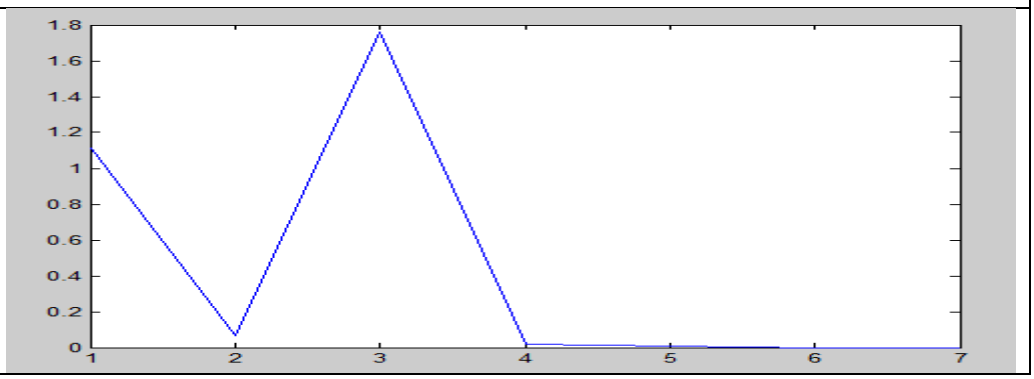
Parámetros	Valor
Distance	'cityblock'
Emptyaction	'drop' 'singleton'
Onlinephase	'on'
options	'Display''off'
replicates	**0**
start	'sample'

Resultado 1

K	1	2	3	4	5	6	7
IDX	1 1 1 1 1 1 1	2 2 2 1 1 1 2	2 2 1 3 3 3 2	1 1 1 4 3 2 1	1 1 2 5 3 3 4	5 5 4 2 1 3 6	2 1 4 5 6 3 7
Je	1.1106	0.0653	1.2753	1.7585	0.0057	0	0

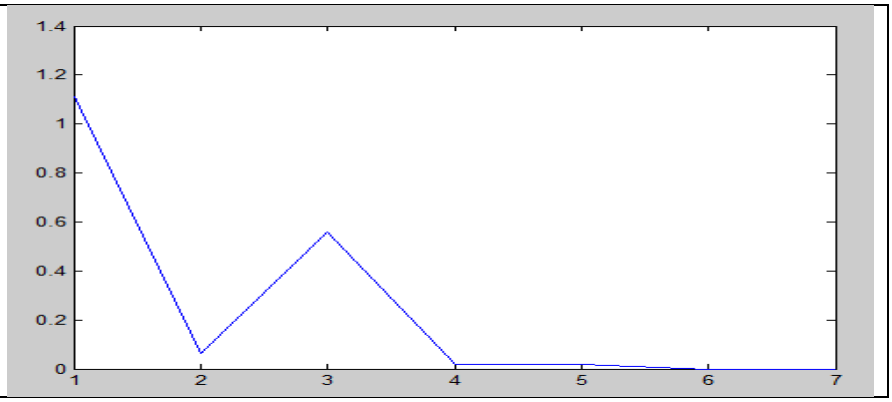
Gráfica k-Je	

Resultado 2

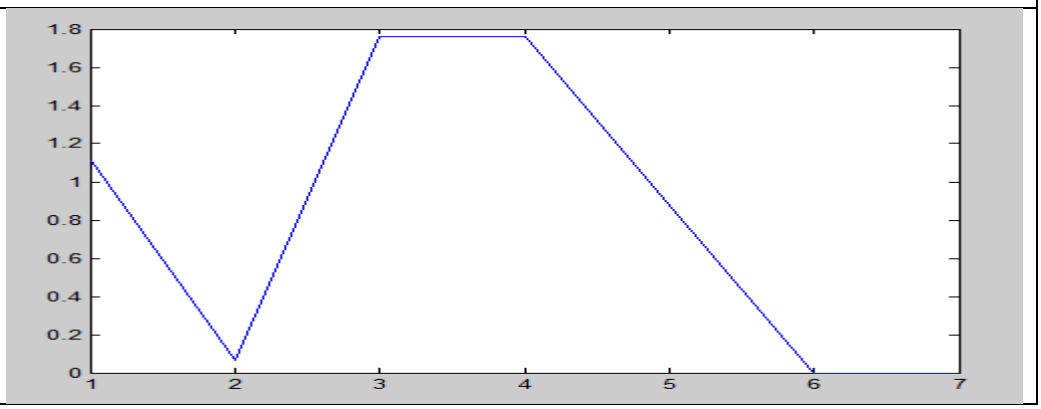
K	**1**	**2**	**3**	**4**	**5**	**6**	**7**
IDX	1 1 1 1 1 1 1	2 2 2 1 1 1 2	2 2 2 1 3 3 2	4 4 2 3 3 3 1	1 1 2 5 3 3 4	5 5 4 2 1 3 6	2 1 4 5 6 3 7
Je	1.1106	0.0653	1.7627	0.0180	0.0057	0	0
Gráfica k-Je							

Resultado 3

K	**1**	**2**	**3**	**4**	**5**	**6**	**7**
IDX	1 1 1 1 1 1 1	2 2 2 1 1 1 2	2 2 2 3 3 3 1	1 1 4 2 2 2 3	1 4 5 3 3 3 2	3 3 2 4 1 5 6	2 1 4 5 6 3 7
Je	1.1106	0.0653	0.5597	0.0180	0.00057	0.0000	0

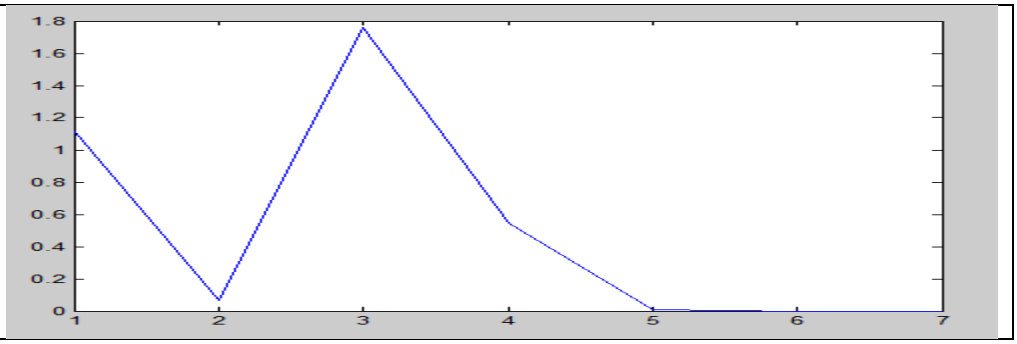
Gráfica k-Je	

Resultado 4

K	1	2	3	4	5	6	7
IDX	1 1 1 1 1 1 1	2 2 2 1 1 1 2	2 2 2 3 1 1 2	4 4 4 3 1 2 4	4 4 4 3 1 2 5	3 3 2 4 1 5 6	2 1 4 5 6 3 7
Je	1.1106	0.0653	1.7627	17585	0.8777	0.0000	0
Gráfica k-Je							

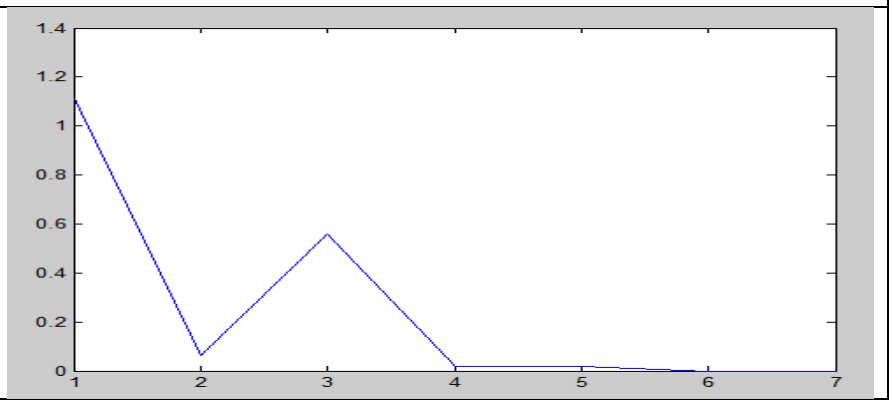
Resultado 5

K	1	2	3	4	5	6	7
IDX	1 1 1 1 1 1 1	2 2 2 1 1 1 2	2 2 2 3 1 1 2	4 4 4 3 2 2 4	4 4 2 3 1 1 5	3 3 2 4 1 5 6	2 1 4 5 6 3 7
Je	1.1106	0.0653	1.7627	0.5459	0.0057	0.0000	0

Gráfica k-Je	

Fijando el valor de *replicates* a un valor superior a cinco, el resultado que obtenemos es constante en cada ejecución. En nuestro caso coincide con el *resultado 3*, cuando el valor de *replicates* era cero.

Parámetros	Valor
Distance	'cityblock'
Emptyaction	'drop' 'singleton'
Onlinephase	'on'
options	'Display''off'
replicates	**>5**
start	'sample'

K	1	2	3	4	5	6	7
IDX	1 1 1 1 1 1 1	2 2 2 1 1 1 2	2 2 2 3 3 3 1	1 1 4 2 2 2 3	1 4 5 3 3 3 2	3 3 2 4 1 5 6	2 1 4 5 6 3 7
Je	1.1106	0.0653	0.5597	0.0180	0.00057	0.0000	0
Gráfica k-Je							

Dependiendo del número de datos por el que este compuesto el corpus con el que vamos a trabajar, habrá que incrementar el número de replicados, ya que habrá que repetir varias veces los cálculos hasta obtener una buena aproximación al valor

óptimo. Aumentar este valor incrementará el tiempo de cómputo, pero es una penalización que hay que asumir, para conseguir un buen resultado.

Analizando los resultados que hemos obtenido para ambos métodos, podemos ver que cuando aumentamos el número de replicates para ambos, los dos convergen a una solución, pero con la Distancia Euclídea, el resultado correcto se consigue con menos iteraciones.

También se puede observar, que en ambos métodos cuando el valor de replicates es cero, se obtienen diferentes soluciones, pero hay una diferencia bastante significativa, aplicando Distancia Euclídea, el punto de inflexión para todas las posibles alternativas siempre ha sido con tres clusters, en cambio con City Block la elección varía entre tres y cuatro clusters.

Por estas dos razones principalmente, se ha decidido que para nuestro corpus de datos, se va a aplicar kmeans con Distancia Euclídea, y con un valor de replicates lo suficientemente alto como para que el resultado sea correcto, pero intentando a la vez que la penalización por tiempo de cálculo reste efectividad al algoritmo; además como emptyaction se usará singleton ,en lugar de error o drop, esto es, en el caso que haya clusters vacíos, en vez de tratarlos como error o eliminarlos, se creará un nuevo cluster con los puntos más alejados; esto se comprobará con los datos del corpus, ya que con los datos de prueba no se aprecian las diferencias de tratar con uno u otro. Se ha decidido no usar drop, ya que existe un bug en la función kmeans de Matlab, que provoca un error de ejecución.

Undefined function or variable "idxBest".

Error in kmeans (line 331)
idx = idxBest;

ANEXO F.
Algoritmo CBSGC

```
%-------------------------------------------------------------------------
%Creación de los vectores y las variables
%-------------------------------------------------------------------------
nfile = 'taxo.json';
s = 2;
ca=0;
maxDC = 100;
%-------------------------------------------------------------------------
[ndocs] = countdocs(nfile)
[vd] = vdocs (nfile,ndocs)
[vtd] = vter1doc(nfile,ndocs)
[vcd] = vcat1doc(nfile,ndocs)
[vc,vcc,ncat,ntcat] = vcat(nfile,ndocs,2,0,'')
[vid] = vindexdoc (vc,vcc,maxDC,0)
[vttd, nter,ntter,~] = maxtermvtt(nfile,ndocs,vid)
[vtt] = vectvtt(nfile,vid,ndocs)
[vt] = vterms (nter)
[vocter,vtercoc] = ocurrencesterm(nter,vtt,0,2000)
[vccindex,vcdindex] = vcindex(vcc,s,ca)
[ncattotalindex] = vctindex(vid,vccindex,ndocs)
[vc,ncat,ntcat] = vcat(nfile,0,1,'');
'creado vector categorias'
[vcc,vcd] = vccategory(vcc,'');
'creado vcc y vcd'
[vid] = vindexdoc (vc,vcc,100,1,vtd);
'indexandos documentos'
[~,c] = size(vid);
ndocs = c;
[ncattotal] = vctindex(vid,vcc,ndocs);
'calculado numero total categorias'
[vtt] = vectvtt(nfile,vid,ndocs);
'creado vtt'
[nter,ntter,~] = maxtermvtt(nfile,ndocs,vid);
'calculado numero terminos'
[vocter,vtercoc] = ocurrencesterm(nter,vtt,0,500);
'calculados vectores de ocurrencias'
%-------------------------------------------------------------------------
%Creación de las matrices
%-------------------------------------------------------------------------
[SparseA,IA,JA,DA] = MatrizA(vid,vc,vcd,vcc,ncat,ncattotal,ndocs);
'matriz A'
[~,IPA,JPA,DPA] = MatrizPA(IA,DA,ncat);
'matriz PA'
[SparseRA,~,~,~] = MatrizRA(JA,DA,ndocs);
'matriz RA'
[AA]  = MatrizAA(SparseRA,SparseA, IPA,JPA,DPA);
'matriz AA'
[SparseB,IB, JB, DB] = MatrizB(vid,vtd,vttd, ndocs,vtercoc,vocter);
'matriz B'
[~,IPB,JPB,DPB] = MatrizPB(IB,DB,ndocs);
'matriz PB'
[~,IRB,JRB,DRB] = MatrizRB(JB,DB,vtercoc);
'matriz RB'
[BB]  = MatrizBB(SparseB, IPB,JPB,DPB,IRB,JRB,DRB);
'matriz BB'
```

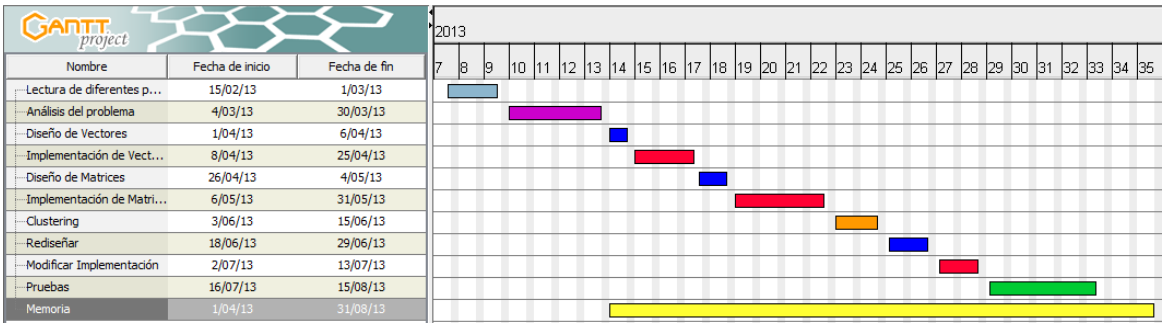
```
[U,V,X,C,S] = fGSVD( AA,BB);
'GSVD'
[H] = MatrizH( C,X,S );
'H'
 [UH, ~, VH] = fSVD(H);
'SVD'
 [US] = fLinearTrans( U,UH);
'transformacion lineal'
%-------------------------------------------------------------------------
%K-particionamiento
%-------------------------------------------------------------------------
vJe = JeCurve(ncat,IPA,JPA,DPA,US,0);
plotJeCurve( vJe,ncat)
[vpeaks,vkopt] = findpeakgraph(vJe,18);
[ vtotal ] = testingK(vc,vkopt,US,IPA,JPA,DPA);
save matvect BB DPA JPA IPA SparseA US ncat ncattotal ndocs ntcat nter
ntter vJe vtotal vc vid vocter vtercoc vtt vcc vcd
%-------------------------------------------------------------------------
%Repetimos, hasta llegar a la raiz la funcion CBSGC
%-------------------------------------------------------------------------
[vMatrixnAA,vMatrixnPA,t] = newAAs(kopt,SparseA,US,IPA,JPA,DPA,ndocsindex
[vMatrixvc,vMatrizncat] = newvc( vc, idxa,kopt)

       %La matriz B permanece constante, el resto de las matrices
       %hay que recalcularlas
       [U2,V2,X2,C2,S2] = fGSVD( vMatrixnAA(1),BB);
       [H2] = MatrizH( C2,X2,S2 )
       [UH2, SH2, VH2] = fSVD(H2);
       [US2,VS2] = fLinearTrans( U2,UH2,V2,VH2);
       vJe =  JeCurve(kopt,vMatrixnPA(1),US2);
       plotJeCurve( vJe,kopt );
       [vpeaks,vkopt] = findpeakgraph(vJe,18);
      [ vtotal ] = testingK(vMatrixvc(1),vkopt,US2,IPA2,JPA2,DPA2);

%-------------------------------------------------------------------------
```

ANEXO G.

Gestión del Proyecto



Nombre	Fecha de inicio	Fecha de fin
Lectura de diferentes p...	15/02/13	1/03/13
Análisis del problema	4/03/13	30/03/13
Diseño de Vectores	1/04/13	6/04/13
Implementación de Vect...	8/04/13	25/04/13
Diseño de Matrices	26/04/13	4/05/13
Implementación de Matri...	6/05/13	31/05/13
Clustering	3/06/13	15/06/13
Rediseñar	18/06/13	29/06/13
Modificar Implementación	2/07/13	13/07/13
Pruebas	16/07/13	15/08/13
Memoria	1/04/13	31/08/13

Figura G1: Diagrama de Gantt para la gestión del proyecto

En la figura anterior se muestra la evolución temporal del proyecto mediante un diagrama de Gantt. Podemos ver que la parte de implementación es la que mayor tiempo nos ha llevado, seguido de cerca de las pruebas. La memoria ocupa la mayoría del tiempo debido a que se ha ido realizando a lo largo del desarrollo del proyecto.

Podemos ver también que hay una tapa de rediseño y de modificación de la implementación, esto se debe, tal y como se ha comentado a lo largo de la memoria, a los problemas de falta de memoria.

Se ha llevado un diario de trabajo, en el que cada día se escribía que se ha hecho, que problemas se han encontrado, posibles soluciones pensadas y trabajo pendiente para el siguiente día; este documento se ha complementado con un diario en el que se apuntaban todas las páginas de internet consultadas, los emails mandados con el director y todos los comandos que se iban aprendiendo de Matlab.

En la siguiente figura se muestra el tiempo dedicado a cada parte del proyecto.

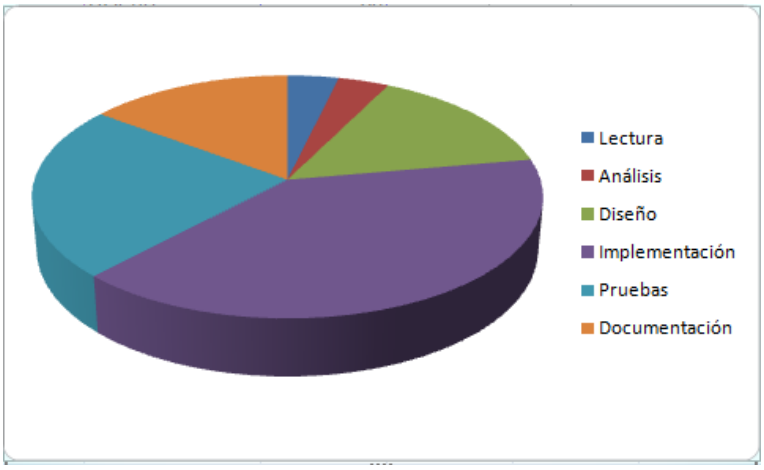


Figura G2: Gráfico que representa las horas invertidas

El número de horas invertidas en cada fase, aproximadamente se refleja en la siguiente tabla.

Fase	Horas
Lectura de papers	20(4%)
Diseño	80(16%)
Vectores	30
Matrices	40
Clustering	10
Implementacion	215(40%)
Vectores	50
Matrices	100
Clustering	40
Taxonomía	25
Pruebas	125(24%)
Documentación	80(16%)
Número total	520

Figura G3: División del trabajo en horas

INDICE DE FIGURAS

INDICE DE TABLAS