Búsqueda y Minería de Información.

Práctica 4

Pareja 09

Ángel Fuente Ortega
Mario Valdemaro García Roque

Ejercicio 1: Recomendación basada en contenido.

Para la implementación de este ejercicio hemos desarrollado 3 clases, 2 de ellas se encuentran englobadas en el paquete lector y la otra en recomendación. Estas 3 clases son:

MoviTags.java:

Esta clase lee el archivo movie_tags.dat guarda los datos en la variable “datos” y los procesa. Con procesar nos referimos a que calcula las similitudes entre películas, gracias a los tags de esta. La forma de hacerlo es mediante el coseno o mediante Jaccard.

La idea que hemos tenido para leer de una forma medianamente rápida la información es insertarla según la leemos en un estructura HashMap<Integer, HashMap<Integer, Integer>>. La idea que subyace en esta estructura es que tenemos un HashMap donde la clave es el Id de la película y el valor es otro HashMap que usa como clave el Id del tag y como valor el peso de ese tag para esa película. Hemos decidido implementar esta estructura ya que nos permite un acceso rápido.

Como ya hemos comentado en esta clase calculamos las similitudes entre películas, estas similitudes las insertamos en una una matriz de Doubles de tamaño NxN donde N es el numero de películas. Hemos decidido usar una matriz y no un HashMap, como el ya comentado anteriormente, ya que nos dimos cuenta que el proceso de cargar las similitudes desde fichero es muy pesado si usamos HashMap. Entonces nos pueden surgir problemas. El principal es que no tenemos un acceso rápido a la información o que esta ocupa demasiado ya que los Id de las películas no están pensados para referirse a una columna/fila concreta de nuestra matriz. Por tanto si queremos hacer referencia directa según el id de la película tenemos mucha información innecesaria. La forma que hemos solucionado el problema de referirnos de forma rápida a las columnas/filas de nuestra matriz y además no ocupar un tamaño excesivo es creando un HashMap que nos indique dada una clave, en este caso el Id de la película, cual es la columna/fila donde esta almacenada su información. Así tenemos un acceso rápido a la información, ya que estamos usando Hases para encontrar la información, no tenemos información que no existe, ya que no tenemos filas/columnas sin correspondencia con un Id, y tenemos un proceso de carga de fichero rápido ya que leer una matriz es bastante mas rápido que un Hash. Aun con todo tenemos información redundante ya que el valor de la matriz [i][j] es el mismo que el de [j][i]. Por tanto la cantidad de información que tenemos en la matriz se podría reducir a N!.

UserRatedMovies.java:

Este es otro lector que procesa la información de user_ratedmovies.dat. La información del fichero se guarda en la variable datos, que es un HashMap<Integer, HashMap<Integer, Double>>. De forma mas concreta esta variable es un Hash que usa el Id como clave y que tiene como valor otro Hash que usa como clave el Id de la película puntuada y tiene como valor esta puntuación.

Existen mas variables y funciones en esta clase, pero no es relevante explicarlas en este apartado (se explicaran en el siguiente).

RecomendadorContenido.java:

Esta clase tiene como objetivo recomendar películas mediante el sistema de similitudes knn basado en contenido. Para ello hace uso de los 2 lectores comentados anteriormente, usándolos como variables de la clase de forma mas concreta usa las similitudes almacenadas en MovieTags y las recomendaciones almacenadas en la clase UserRatedMovies. De tal modo que obtiene las N películas mas similares que ha recomendado el usurio y haz el proceso que ya sabemos de recomendación mediante knn.

Es importante que destaquemos 2 cosas sobre como funcionan los algorimos de recomendación que hemos hecho es casos en los que no puden recomendar. Principalmente estos son 2, que intentemos recomendar una pelicula que no tiene tags por tanto no existe; en ese caso daremos la media de esa pelicula, y que tengamos que recomendar una pelicula que no hay suficeintes peliculas similares. Entonces recomendaremos las N peliculas mas similares que tengan similitud mayor que 0,0 independientemente del numero de vecinos. En el siguiente apartado hacemos algo parecido.

Ejercicio 2: Recomendación basada en filtrado colaborativo.

El esquema que hemos usado para implementar este apartado es bastante parecido al ya comentado en el apartado anterior. Se hace uso aquí de 2 clases:

UserRatedMovies.java

Como ya mencionamos en el apartado anterior vamos a pasar a explicar variables y funciones internas que se hacen uso en este apartado. Ya hemos explicado cosas sobre como se leen los archivos, por tanto nos saltaremos esta parte. En cuanto a como se calculan y almacenan las similitudes basadas en filtro colaborativo decir que el proceso es muy parecido al que ya comentamos en la clase MovieTags.java, es decir existe una matriz de similitudes donde se almacenan estas y un Hash para referirnos mas rápidamente a estos datos. Sin embargo ahora las similitudes no se calculan mediante coseno o jaccard sino mediante cosseno o pearson. Para pearson ha sido necesario desarrollar un hash map adicional que calcule la media de ratings de cada usaurio, siendo la calve el id del usuario y el valor la media de este.

RecomendadorColabarativo.java:

El principio de funcionamiento de este recomendador es bastante parecido al ya explicado en apartado anterior, es decir usamos knn y según las similitudes y los valores de raitngs que tengamos damos valor a un película.

Ejercicio 3: Evaluación.

Básicamente para este apartado hacemos uso de las clases de los apartados anteriores y una nueva clase Evalucion.java esta nueva clase tiene los distintos recomendadores como variables y dos HashMaps donde se almacenan los ratings de test y de train. Estos dos particiones se crean llamando a las función “particion” donde se le pasa un porcentaje que representara el conjunto de Train y divide los ratings en los 2 conjuntos. Una vez dividida la información se calculan las similitudes con el conjunto de Train mediante la forma que se haya elegido (contenido, colaborativo, coseno, etc) y se pasa a calcular el MAE o el RMSE del conjunto de Test.

A continuación mostramos el MAE y el RMSE de las distintas pruebas con un 80% de los ratings en test:

MAE

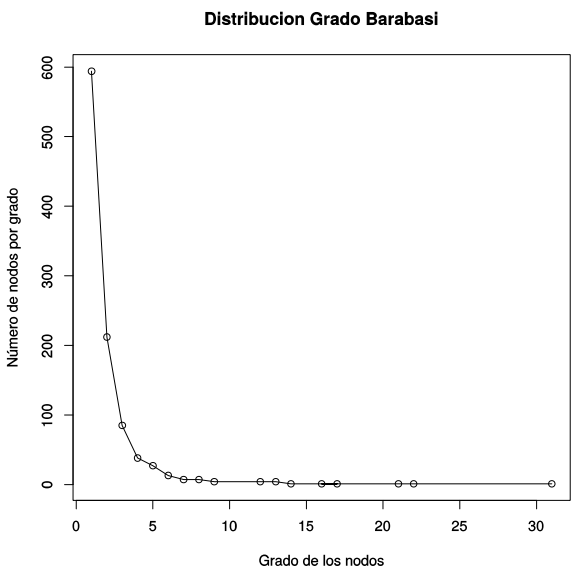
Vecinos	**2**	**16**	**128**
Contenido Coseno	0,750	0,660	0,658
Contenido Jaccard	0,715	0,638	0,652
Colaborativo Coseno	0,787	0,683	0,674
Colaborativo Pearson	0,812	0,679	0,670

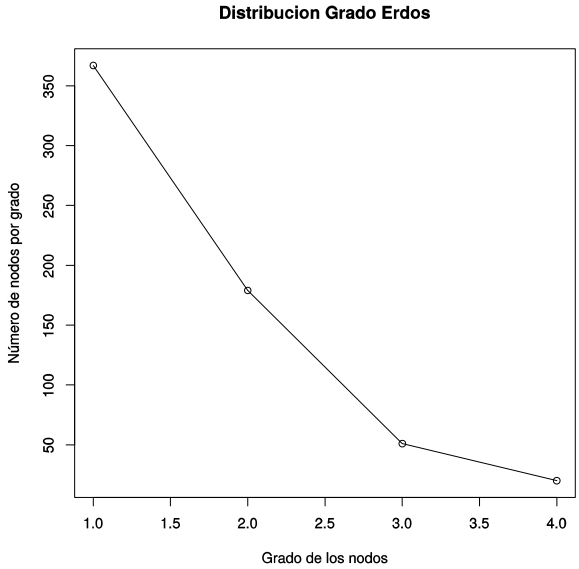
RMSE:

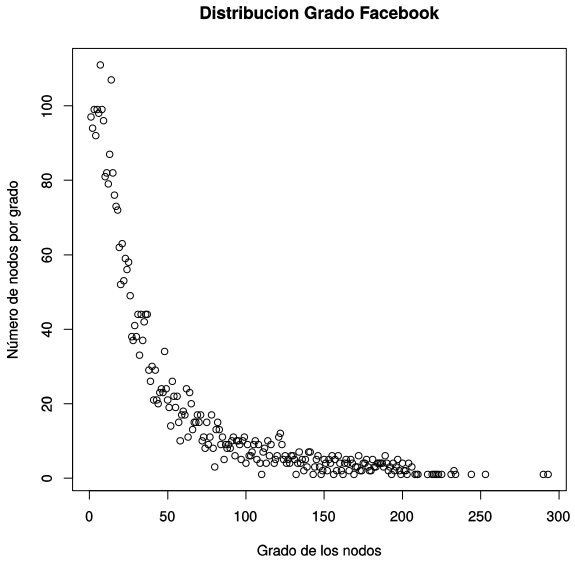
Vecinos	**2**	**16**	**128**
Contenido Coseno	1,008	0,882	0,879
Contenido Jaccard	0,961	0,851	0,868
Colaborativo Coseno	1,034	0,902	0,884
Colaborativo Pearson	1,057	0,889	0,878

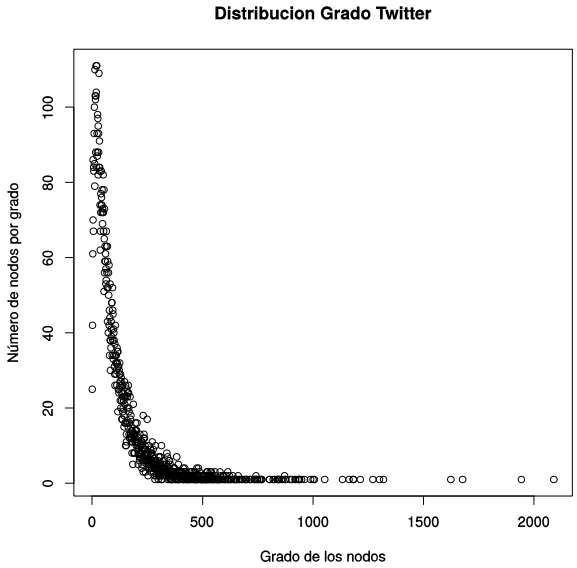
Ejercicio 4: Análisis de redes sociales.

Distribución de grado de las redes de: facebook, twitter, barabasi y erdos.





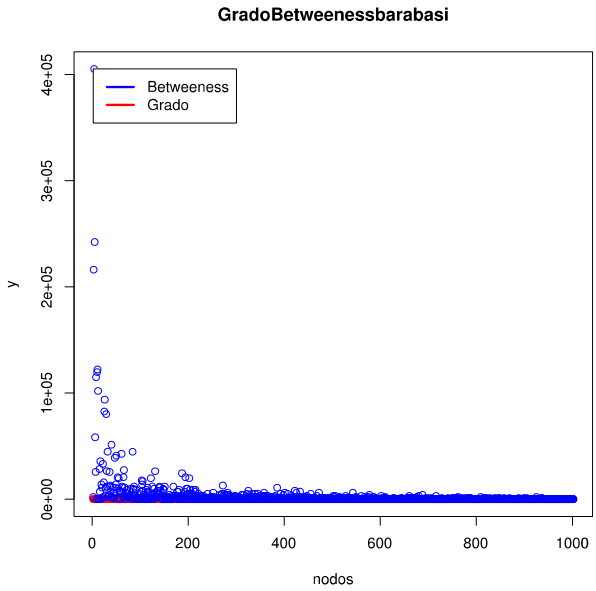


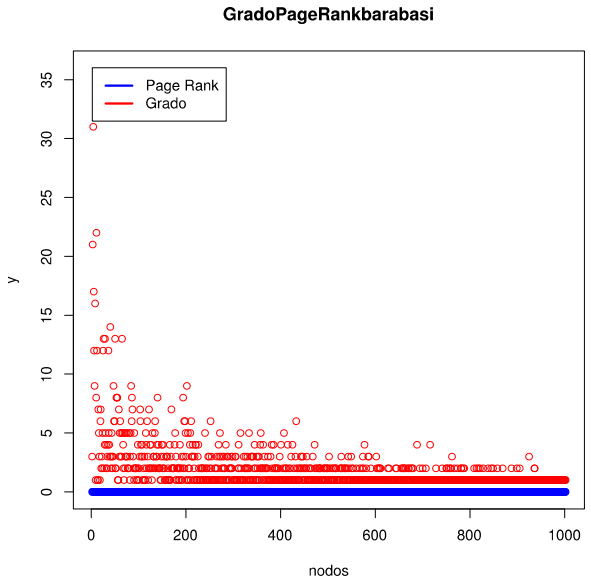


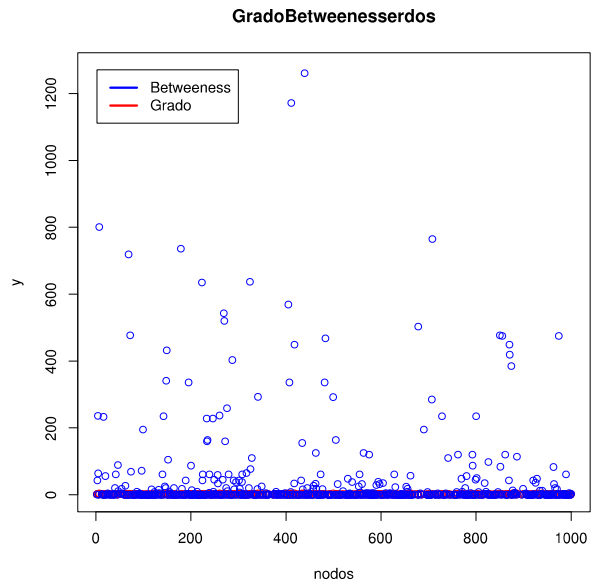
Como se puede ver, quitando el grafo de Erdos, que se genera aleatoriamente, el resto siguen una distribución power-law. Donde unos pocos usuarios tienen mucho grado, y muchos usuarios tienen poco grado, o usuarios.

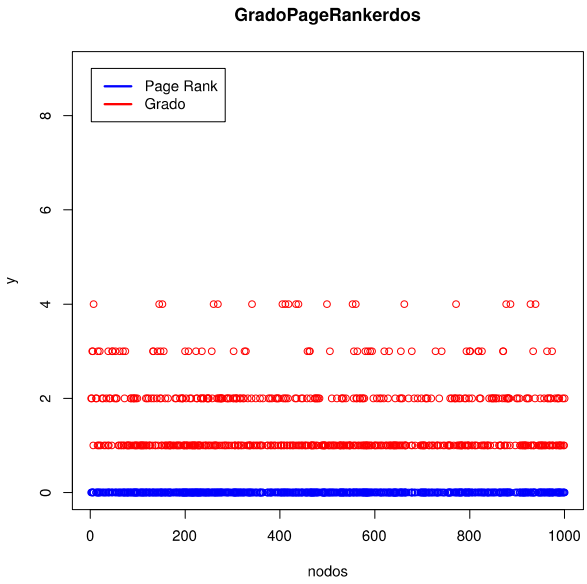
Para los archivos donde se muestra el pageRank y el Betweeness o el clustering de un nodo o del grafo, hay valores para los grafos de erdos y barabasi que son 0. Esto creemos que se debe porque a veces se divide por 0, dando lugar a un NaN. Y esto java lo imprime como un 0.

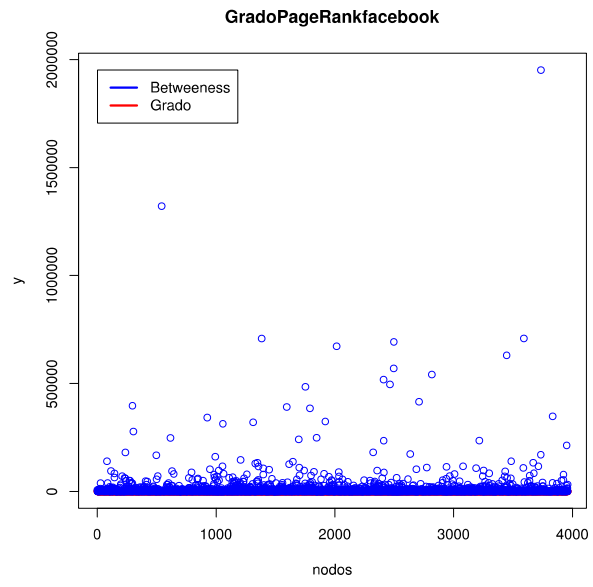
Los scatterplots para todas las redes:

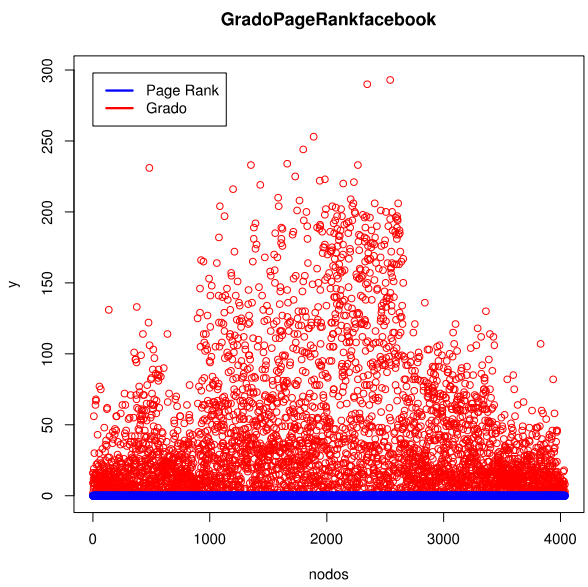


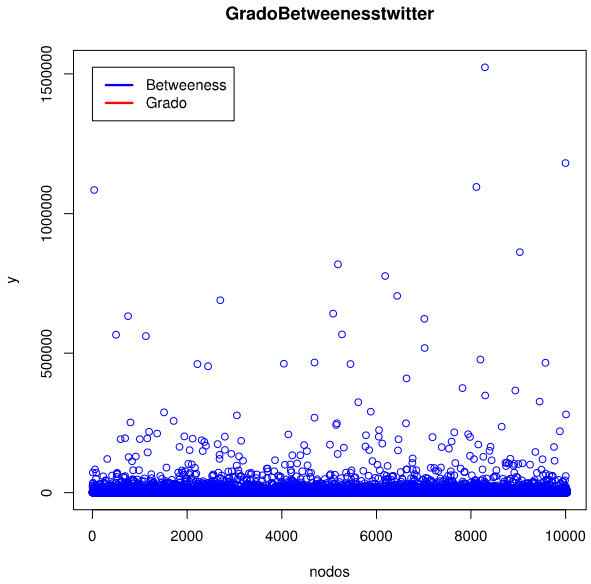


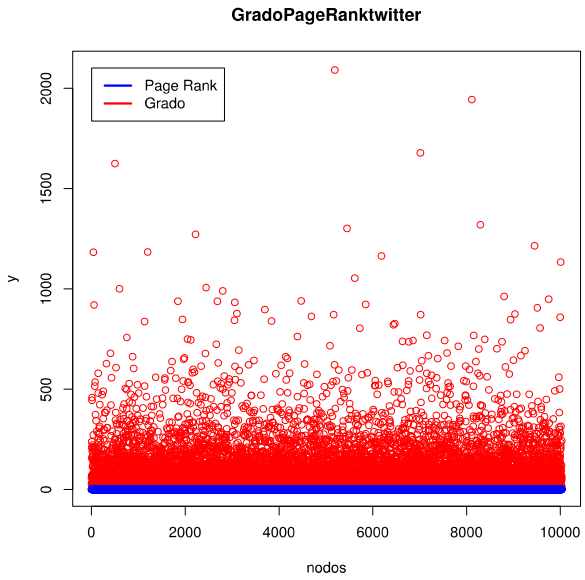












Adjuntamos las gráficas de los scatterplots de las redes small1 y small2, donde puede verse que el grado y el page rank es casi lo mismo.

La paradoja de la amistad: