9-5-2018

Practica 4

Búsqueda y minería de información

Oscar García de Lara Parreño
Jorge Gómez Conde

CONTENIDO

TRABAJO REALIZADO

Hemos realizado completamente los ejercicios 1, 2, 3, 4 y 5 del bloque de recomendación. En el bloque II, análisis de redes sociales hemos realizado los ejercicios 6 y 7.

Más adelante se detallan las implementaciones utilizadas en la práctica.

BLOQUE I - RECOMENDACION

ESTRUCTURAS DE DATOS Y RECOMENDACIÓN SIMPLE

RATINGSIMPL

En esta clase representamos las puntuaciones de cada usuario sobre cada ítem. Utilizamos varias estructuras Map, para guardar la información. Un map que, relacionada cada usuario con sus ítems puntuados, otro map que relaciona cada ítem con los usuarios por los que ha sido puntuado, y finalmente un map que cumple la función de rating, asociando cada usuario con ítem puntuado junto a la su respectiva puntuación en formato Double.

FEATURESIMPL

En la definición de las características (features) de los ítems, hemos utilizado un planteamiento similar al de RatingsImpl. No es necesario guardar tanta información en este caso, como la asociación de cada característica con cada ítem en el que se encuentra, entonces prescindimos de los maps y solamente utilizamos aquel que relaciona los ítems con sus características y sus respectivas valoraciones.

ABSTRACTRECOMMENDER

Esta clase la hemos creado para implementar el método recommend y así extender los demás recomendadores y evitar repetir código. En recommend hemos evitado pedir score de ítems ya puntuados por el usuario.

AVERAGERECOMMENDER

En average básicamente recorremos todos los ítems y cogemos los usuarios que han puntuado cada ítem y si ha superado o igualado el mínimo hacemos la media de sus scores, esta media se almacena en un Map Item/media, esto lo hacemos por eficiencia, así evitamos hacer estos cálculos cuando se llama al método score.

FILTRADO COLABORATIVO KNN

ABSTRACTUKNNRECOMMEND

Para esta parte hemos creado una clase abstracta para User KNN y User KNN normalizada, ya que la parte de guardar las similitudes y la forma de calcular el score es común, para esto sea posible tenemos un método abstracto que devolverá el score final, para ello se le pasa el score del algoritmo, C y el numSimilitudes.

USERKNNRECOMENDER

En este caso el score final es el score que se calcula siguiendo el algoritmo y se le pasa al método abstracto.

NORMUSERKNNRECOMMENDER

El score final será la implementación del método abstracto teniendo en cuenta si ha superado el mínimo si es así, se divide el score entre C para normalizar, si no la supera se devuelve 0.

RECOMENDACIÓN BASADA EN CONTENIDO

CENTROIDRECOMMENDER

Este algoritmo se extiende de la clase mencionada anteriormente, AbstractRecommender. Prácticamente la implementación del algoritmo de Rocchio se encuentran en las clases CosineFeaturesSimilarity y FeatureCentroid

COSINEFEATURESSIMILARITY

Aquí está contenida la función de recomendación basada en contenido de centroides. Extiende la clase abstracta FeaturesSimlarity. El método *sim()* calcula la puntuación de dos vectores indicados por parámetros. Para definir la estructura que guarda los vectores de los usuarios necesarios para esta forma de recomendación utilizamos FeaturesCentroid, y a su vez creamos un método *setFeatures()* distinto al indicado por la clase padre.

FEATURESCENTROID

Extendemos esta clase de FeaturesImpl. Modificamos el método constructor para que inicialice con los nuevos valores de los vectores de usuario. Para ello es necesario proporcionar a la clase la tabla de ratings de cada usuario y las características de cada ítem.

AMPLIACIÓN DE ALGORITMOS

ITEMNNRECOMMENDER

En este algoritmo no era necesaria una versión no normalizada y normalizada, por tanto, hemos optado por no hacer una clase abstracta e implementarlo directamente. El algoritmo es muy similar al de la clase abstracta de UserKNN, lo único que cambia que las similitudes se calculan entre ítems y como indica que no hay que hacerlo con K pues directamente nos ahorramos de hacerlo el ranking y recorremos los getItems(users) al calcular el score

ITEMNNRECOMMENDER CON SIMILITUD DE JACCARD

Tal como está implementado el algoritmo no hay que cambiar nada, solo pasarle otra similitud que en este caso es la de Jaccard para tener en cuenta los atributos de los items.

STUDENTTTEST

Hemos implementado la similitud de Pearson.

EVALUACION

La siguiente tabla es de MovieLens "latest-small" dataset

Recomendación / Métricas	*RMSE*	*P@10*	*Recall@10*	*Tiempo de ejecución*
kNNUser - Cos	21.5587	0.2054	0.1738	27s 614ms
kNNUserNorm - Cos	0.9279	0.0086	0.0101	16s 689ms
ItemNNRecommender - Cos	227.0657	0.1777	0.1590	2min 38s 947ms
ItemNNRecommender - Jaccard	9.9622	0.0241	0.0192	58s 387ms
CentroidRecommender	1.1175	0.0092	0.0083	51s 740ms

Para MovieLens HetRec dataset

Recomendación / Métricas	*RMSE*	*P@10*	*Recall@10*	*Tiempo de ejecución*
kNNUser - Cos	42.522	0.4056	0.1123	5min 55s 229ms
kNNUserNorm - Cos	0.798	0.0223	0.0065	1min 6s 952ms
ItemNNRecommender - Cos	625.5565	0.3589	0.0961	25min 52s 817ms
ItemNNRecommender - Jaccard	43.4612	0.2282	0.07	91min 300ms
CentroidRecommender	3.4456	0.0079	0.0019	25min 44s 123ms

No tiene sentido poner el RMSE de KNN User sin normalizar, Centroide e ItemmNN ya que RMSE necesita que lo que devuelva el algoritmo sea un rating, cosa que estos algoritmos no hacen. Somos conscientes que Item con Jaccard tarda mucho posiblemente por la forma de calcular esta similitud que se hace cada vez que se pide.

BLOQUE II – ANÁLISIS DE REDES SOCIALES

PRELIMINARES

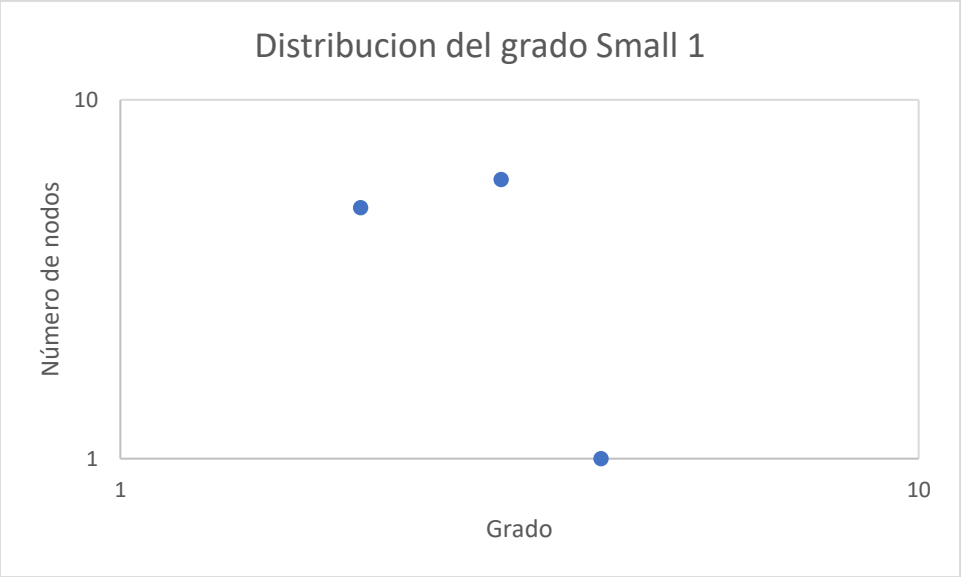
La construcción de los grafos ha sido implementada en la clase MakeGraph. Esta clase contiene un método *main()* que genera ambos grafos, Erdos-Renyi y Barabasi-Albert. Cabe destacar que no se ha utilizado la librería Jung directamente. En su lugar, utilizamos la liberia Agape, en su versión 0.4., propia de la universidad de Orleans, en Francia.

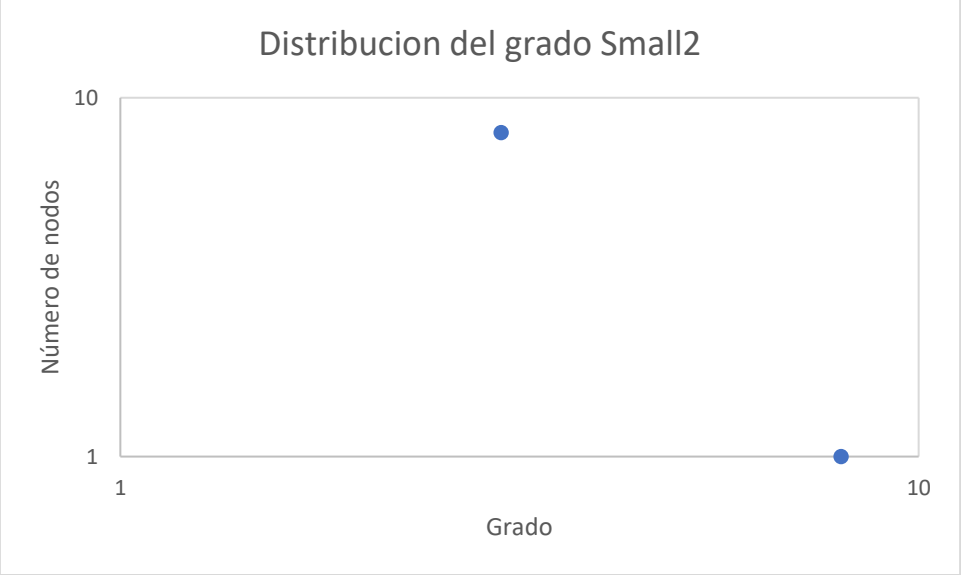
Enlace a la librería: https://traclifo.univ-orleans.fr/Agape/

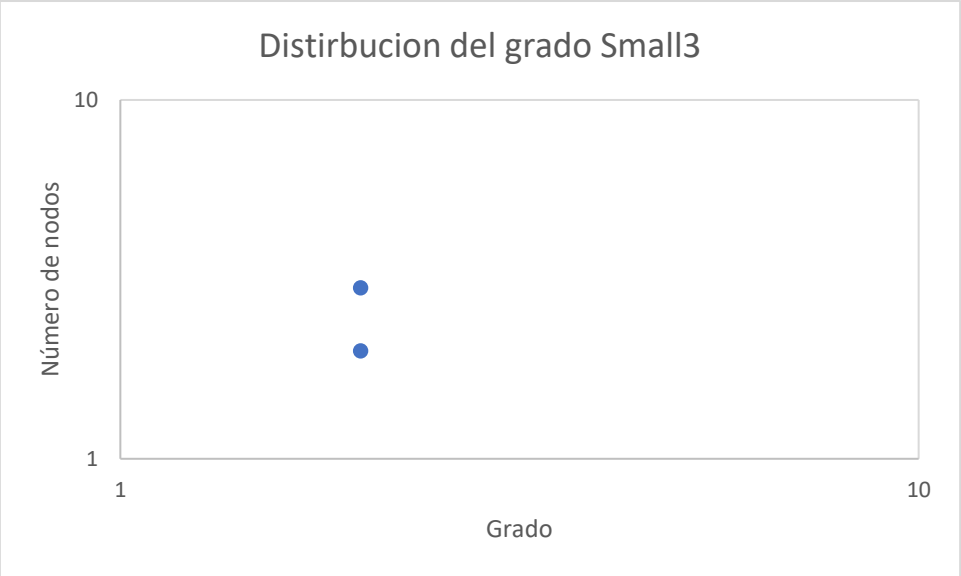
Enlace al tutorial: https://hal.archives-ouvertes.fr/hal-00663866/document

La clase también incluye dos métodos para le impresión de los grafos, uno en pantalla y el otro en un fichero. Ambos métodos se encargan de formatear el nombre de los nodos para que correspondan con los formatos dados en la clase Test.

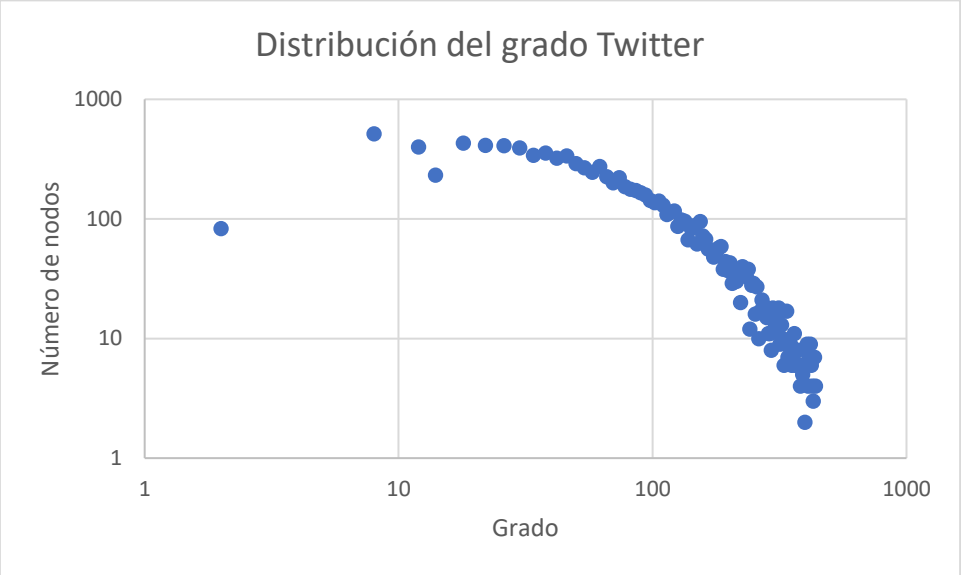
Ahora vamos a mostrar las gráficas de distribución del grado de los distintos grafos.



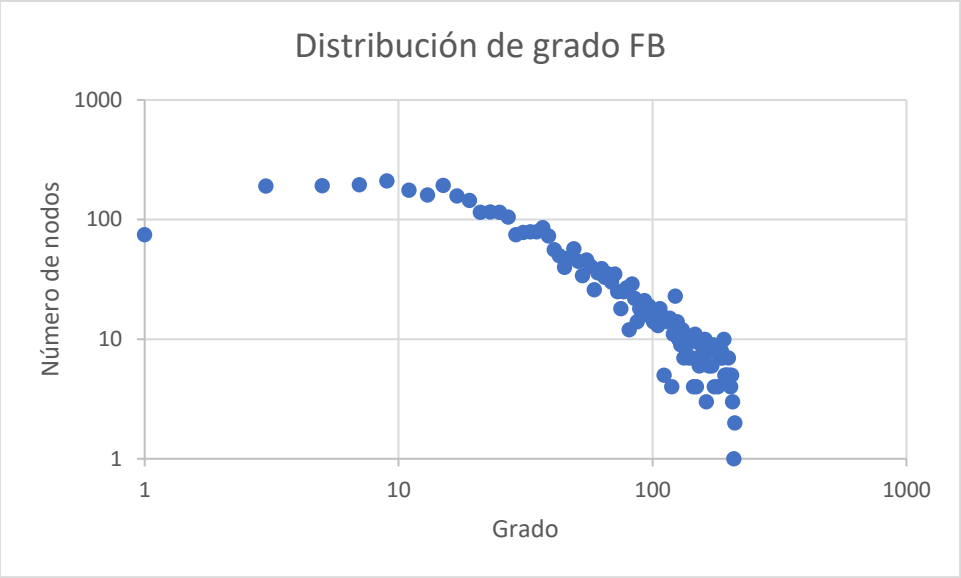




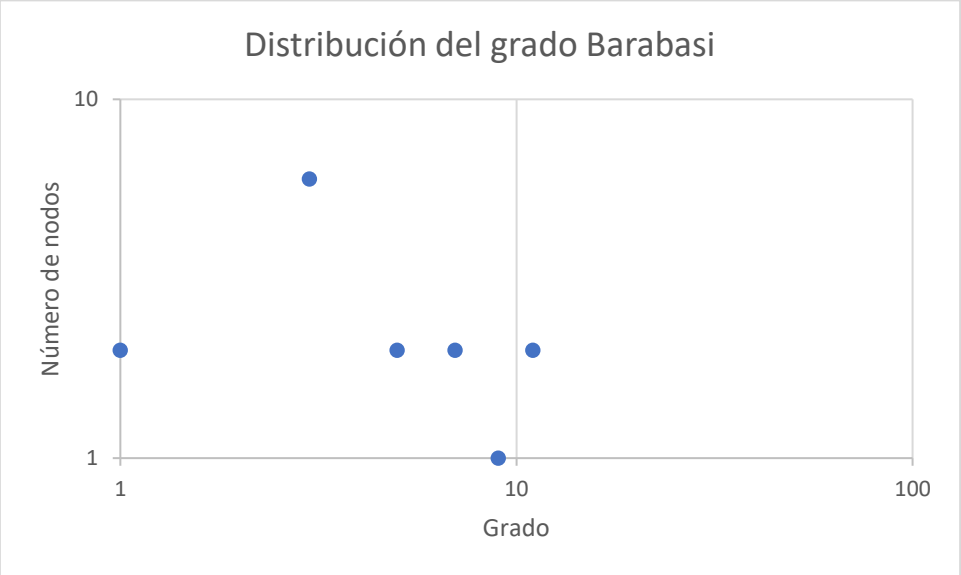
En las tres small no se aprecia que sigan distribución power law, básicamente por la poca cantidad de nodos que presentan

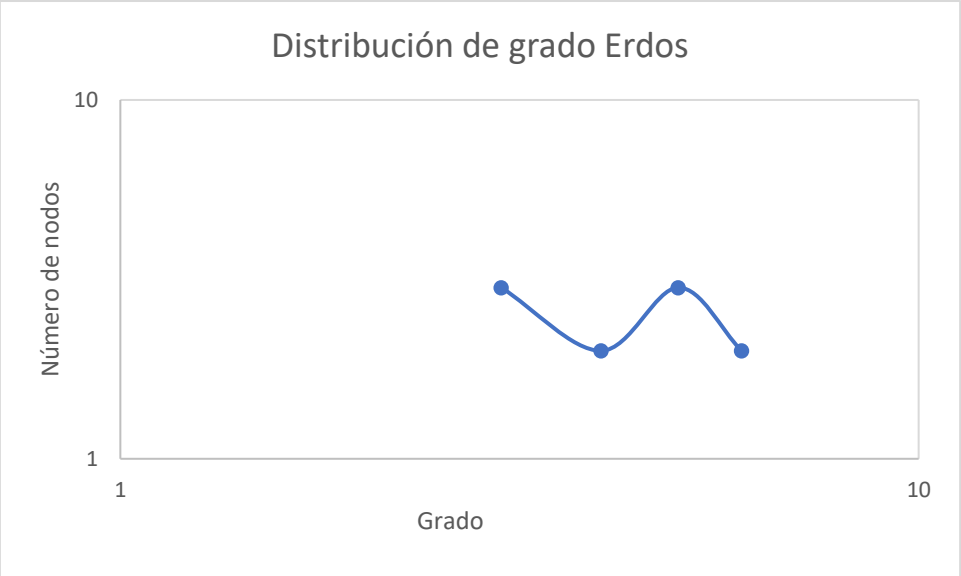


En el caso de Twitter si se parece más a una distribución power law, no es perfecta ya que no es muy estable al principio pero cuanto mas grande es el grado mas decrece.



En FB pasa algo parecido a twitter, es mas estable al principio pero decae mas lentamente de lo esperado pero si que se llega a observar una distribución power law.





Nos pasa lo mismo que en el caso de las Smals, hay demasiado pocos nodos y no se aprecia la distribución.

MÉTRICAS

Métrica	*Facebook*	*Twitter*
UserClusteringCoefficient	1s 98ms	22s 122ms
Embeddedness	22s 797ms	6min 54s 304ms
Assortativity	104ms	260ms
AvgUserMetric	1s 156ms	19s 779ms
ClusterCoefficient	5s 238ms	35s 835ms

Estos tiempos son muy orientativos, ya que dependerán del ordenador utilziado

USERCLUSTERINGCOEFFICIENT

Aplica el algoritmo de coeficiente de clustering local para cada usuario. Calculamos el numero de conexiones existentes en el nodo y dividimos entre la mitad del número de posibles vecinos del nodo, ya que estamos utilizando grafos no dirigidos.

EMBEDDEDNESS

El algoritmo aplicado en esta métrica es simplemente aplicando Jaccard a cada uno de todos los pares posibles de nodos del grafo. A la hora de realizar los cálculos, utilizamos una clase auxiliar EdgeImpl, que implementa un método *equals() y hashCode()* ya que Edge no lo hace, para evitar procesar dos veces el mismo eje. La clase para esta métrica implementa la interfaz LocalMetric.

ASSORTATIVITY

Para calcular la asortividad del grado aprovechamos que hay que recorrer todos los usuarios de la red para calcular los grados al cuadrado y el grado al cubo que sirven para estandarizar la métrica.

AVGUSERMETRIC

La clase implementada para esta métrica se sirve de otra métrica ya implementada, UserClusteringCoefficient. Realizamos la media de los resultados dados por el cálculo del coeficiente de clustering de cada usuario.

CLUSTERCOEFFICIENT

Esta clase aplica el algoritmo de coeficiente de clustering global orientado a grafos no dirigidos. Se calcula el número de triángulos dividido entre número de caminos.