Búsqueda y Minería de Información 2018-2019

Universidad Autónoma de Madrid, Escuela Politécnica Superior

Grado en Ingeniería Informática 4º curso

Práctica 4 – Sistemas de recomendación y análisis de redes sociales

Fechas

- Comienzo: miércoles 3 de abril
- Entrega: viernes 10 de mayo (hora límite 23:55)

Objetivos

Esta práctica reúne dos objetivos: por una parte implementar y evaluar sistemas de recomendación, y por otra, implementar funcionalidades de análisis de redes sociales.

Respecto al primer objetivo, se desarrollarán:

- Algoritmos de recomendación basada en filtrado colaborativo.
- Algoritmos de recomendación basada en contenido.
- Métricas de evaluación de sistemas de recomendación.

Y para el segundo, se implementarán:

- Métricas que se utilizan en el análisis de redes sociales.
- Otras funcionalidades a elección opcional del estudiante, tales como más métricas, la detección de comunidades, la generación aleatoria de redes sociales, o la recomendación de contactos.

Material proporcionado

Se proporciona software y datos para la realización de la práctica, que se divide en dos bloques.

Bloque I – Sistemas de recomendación

- Varias clases e interfaces Java en recsys-src.zip, con las que el estudiante integrará las suyas propias. Igual que en las prácticas anteriores, se incluye una clase `es.uam.eps.bmi.recsys.test.Test` con un programa main que deberá funcionar con las clases a implementar por el estudiante.
- Tres conjuntos de datos que incluyen ratings y etiquetas asignados por usuarios a películas: un conjunto pequeño de datos ficticios, y otros dos conjuntos de datos reales disponibles en la Web de MovieLens.
  - Conjunto mínimo de prueba toy-data.zip. Incluye datos ficticios de ratings y características, así como un split manual fijo.
  - Conjunto de datos real: el disponible en ml-latest-small.zip en https://grouplens.org/datasets/movielens/latest. De los archivos disponibles, se utilizarán sólamente ratings.csv y tags.csv. Se facilita en Moodle una versión de tags.csv con las frecuencias de etiquetas ya agregadas por ítems.
- Un documento de texto "recsys-output.txt" con la correspondiente salida estándar que deberá producir la ejecución del programa "java Test".

Bloque II – Análisis de redes sociales

- Varias clases e interfaces Java en sna-src.zip, con las que el estudiante integrará las suyas propias. Igual que en las prácticas anteriores, se incluye una clase `es.uam.eps.bmi.sna.test.Test` con un programa main que deberá funcionar con las clases a implementar por el estudiante.
- Redes sociales de prueba:
  - Tres redes pequeñas de prueba proporcionadas en graph-small.zip.
  - Redes reales: la red disponible en https://snap.stanford.edu/data/egonets-Facebook.html, y twitter.zip, obtenida en una descarga de Twitter (unos 10 mil usuarios con medio millón de relaciones follow).
  - Al conjunto de redes de prueba, el estudiante añadirá dos redes más, simuladas, en el ejercicio 6.
- Un documento de texto "sna-output.txt" con la correspondiente salida estándar que deberá producir la ejecución del programa "java Test".

Ejercicios

Bloque I – Recomendación

Se implementarán estructuras y diferentes algoritmos para el desarrollo de sistemas de recomendación. Indicación: para los algoritmos de recomendación, se puede implementar el método `recommend(int cutoff)` en una clase abstracta que use el método `score(int user, int item)`, a implementar en las subclases para los diferentes algoritmos. **Importante:** recordar que no deben recomendarse los ítems que los usuarios ya hayan puntuado.

1. Estructuras de datos y recomendación simple (1pt)

Implementar las clases necesarias para manejar **datos de entrada** para los algoritmos de recomendación: a) ratings y b) características de ítems (o usuarios). La funcionalidad a proporcionar se establece en las interfaces `es.uam.eps.bmi.recsys.data.Ratings` y `es.uam.eps.bmi.recsys.data.Features`, para las que se definirán las clases `es.uam.eps.bmi.recsys.data.RatingsImpl` y `es.uam.eps.bmi.recsys.data.FeaturesImpl` que las implementen. Estas clases se utilizarán en la implementación de los algoritmos de recomendación. La interfaz `Ratings` indica además la implementación de un método que divida los ratings aleatoriamente en entrenamiento y test, que se utilizará para evaluar y comparar la efectividad de diferentes algoritmos de recomendación.

Implementar las clases necesarias para almacenar la **salida** de un recomendador, partiendo de la interfaz `es.uam.eps.bmi.recsys.Recommendation`.

Implementar un primer **recomendador simple** por rating promedio en una clase `es.uam.eps.bmi.recsys.recommender.AverageRecommender`. El recomendador sólo recomendará ítems que tengan un mínimo de ratings que se indicará en el constructor (con ello se mejora el acierto de la recomendación). Se proporciona una clase `MajorityRecommender` a modo de ejemplo en el que el estudiante podrá basarse.

2. Filtrado colaborativo kNN (1.5pt)

Implementar dos variantes de filtrado colaborativo mediante **vecinos próximos orientado a usuarios**:

- Implementar la clase `es.uam.eps.bmi.recsys.recommender.UserKNNRecommender` para realizar filtrado colaborativo basado en usuario (sin normalizar por la suma de similitudes). Se sugiere crear los vecindarios "offline" en el constructor del recomendador. Se recomienda asimismo utilizar la clase `es.uam.eps.bmi.recsys.ranking.RankingImpl`, que utiliza un heap de ránking, para construir los vecindarios.
- Implementar una variante normalizada `es.uam.eps.bmi.recsys.recommender.NormUserKNNRecommender`. El algoritmo exigirá un mínimo de ratings por vecinos para aceptar recomendar un ítem (con ello se mejora el acierto de la recomendación).

3. Recomendación basada en contenido (1.5pt)

Implementar un algoritmo de recomendación **basada en contenido** (utilizando etiquetas de los ítems) mediante centroide vectorial de usuario por similitud coseno, en una clase `es.uam.eps.bmi.recsys.recommender.CentroidRecommender`. Para simplificar, no será necesario aplicar *tf-idf* a la frecuencia de las etiquetas.

4. Ampliación de algoritmos (1pt)

Implementar variantes adicionales de los algoritmos de filtrado colaborativo y basados en contenido, tales como:

- Un algoritmo colaborativo por **vecinos próximos orientado a ítem**: `es.uam.eps.bmi.recsys.recommender.ItemNNRecommender`. Mientras que el algoritmo basado en usuario utiliza vecindarios de tamaño k, se sugiere que el algoritmo basado en ítems no acote el vecindario.
- Un algoritmo basado en **contenido** (etiquetas) **por vecinos próximos**. Para este algoritmo podrá servir la misma clase `ItemNNRecommender` del punto anterior, simplemente definiendo una similitud basada en características (en lugar de ratings) de los ítems. Se sugiere probar el algoritmo con una similitud de Jaccard.
- **Otras variantes** adicionales a elección del estudiante: similitud de Pearson, kNN centrado en la media, similitud coseno con *tf-idf* de etiquetas, nuevas variantes normalizadas, etc. Para probar los métodos deberá incluirse en la entrega una ampliación del main de test en una clase aparte `es.uam.eps.bmi.recsys.test.StudentTest` ilustrando la ejecución de las variantes adicionales.

5. Evaluación (1pt)

Se desarrollarán clases que permitan calcular diferentes métricas para evaluar y comparar el acierto de los recomendadores, en particular: RMSE, precisión y recall. **En la memoria se incluirán dos tablas** (una por conjunto de datos) con los valores de las métricas (tres columnas) más tiempo de ejecución (una columna más) sobre todos los algoritmos implementados (filas).

Bloque II – Análisis de redes sociales

Para simplificar, en los ejercicios que siguen supondremos que las redes son no dirigidas (las relaciones en los datos de Twitter son conexiones follow dirigidas, pero ignoraremos igualmente la dirección).

6. Preliminares (1pt)

Generar dos **redes sociales simuladas** siguiendo los modelos de Barabási-Albert y Erdös-Rényi. El tamaño y densidad de los grafos se deja a elección propia. Se pueden utilizar las clases proporcionadas en la librería JUNG: `BarabasiAlbertGenerator` y `ErdosRenyiGenerator`, o programar implementaciones propias (lo cual también es muy sencillo).

Realizar un análisis básico de la **distribución del grado** en las seis redes sociales de la práctica: small x 3, Facebook, Twitter, Barabási-Albert y Erdös-Rényi. Para cada red:

- Generar una gráfica de distribución del grado (utilizando escala log-log cuando ello sea útil) y comprobar en qué medida se observa una distribución power law.
- Verificar si se observa la paradoja de la amistad (en sus diferentes versiones).

Los resultados de este ejercicio no conllevan entrega de software, sino sólo la documentación de los mismos en la memoria.

7. Métricas (2pt)

Se implementarán las siguientes métricas topológicas:

- Coeficiente de **clustering** de un usuario.
- **Arraigo** de un arco (o de un par de usuarios).
- Coeficiente de **clustering** de una red social.
- Coeficiente de **asortatividad** de grado de una red.

8. Ejercicio libre (2pt)

El estudiante desarrollará uno o varios métodos de análisis de redes a su propia elección. Se sugiere por ejemplo:

- Implementación de métricas adicionales a elección del estudiante, tales como betweenness, closeness, modularidad, PageRank, etc., integradas en la misma jerarquía de métricas que el ejercicio anterior. Para este ejercicio deberá incluirse en la entrega una ampliación del main de test en una clase `es.uam.eps.bmi.sna.test.StudentTest` ilustrando la ejecución de las métricas adicionales. (Para la modularidad se incluye en los datos proporcionados una partición por tipos del grafo small3; consultar con los profesores si se desea probar con algún conjunto de datos público más.)
- Detección de comunidades y enlaces débiles.
- Creación de modelos para la generación aleatoria de redes sociales (p.e. amigos de amigos, etc.).
- Recomendación de contactos.

El software que se desarrolle se incluirá en la entrega, y se documentarán en la memoria las pruebas realizadas y los resultados obtenidos. En caso de que proceda mostrar figuras de grafos, se sugiere utilizar las facilidades de visualización de la herramienta Gephi.

Este ejercicio se evaluará en base a la cantidad, calidad e interés del trabajo realizado.

Indicaciones

La realización de los ejercicios conducirá en muchos casos a la implementación de clases adicionales a las que se indican en el enunciado. Algunas vendrán dadas por su aparición en los propios programas `Test`, y otras por conveniencia a criterio del estudiante.

Igual que en prácticas anteriores:

- No deberá editarse el software proporcionado. El software entregado será disjunto al que el profesor ha facilitado; la unión de ambos deberá compilar sin errores.
- Los programas `Test` deberán compilar y ejecutar correctamente “a la primera” (naturalmente con salvedad de los ejercicios que no se hayan implementado).

Para la corrección de la práctica, el profesor unirá los fuentes entregados por el estudiante más los fuentes proporcionados por el profesor, comprobando que no haya archivos .java repetidos ni errores de compilación.

Para el ejercicio 5, se incluirán en la memoria una tabla de resultados en ml-latest-small.zip, con una fila por cada algoritmo de recomendación implementado, con la siguiente estructura (con **cuatro decimales** en los valores de las métricas):

	RMSE	P@10	Recall@10	Tiempo ejecución
kNN usuario				
kNN usuario normalizado				
...				

Para el ejercicio 7, se documentarán en la memoria los **tiempos de ejecución** de las métricas en las redes Facebook y Twitter, en una tabla con la siguiente estructura:

	Facebook	Twitter
Coef. clustering usuario		
Embededness		
Coef. clustering global		
Asortatividad		

Entrega

La entrega consistirá en un único fichero zip con el nombre **bmi-p4-XX.zip**, donde XX debe sustituirse por el número de pareja (01, 02, …, 10, …). Este fichero contendrá:

- Una carpeta **src/** con todos (**y sólo**) los ficheros .java con las implementaciones solicitadas en los ejercicios. Los .java se ubicarán en la ruta apropiada de subcarpetas según sus packages.
- Una carpeta **data/** con los grafos Erdös-Rényi y Barabási-Albert.
- En su caso, una carpeta **lib/** con los .jar adicionales que fuesen necesarios. Se recomienda no obstante consultar con el profesor antes de hacer uso de librerías adicionales.
- Una **memoria bmi-p4-XX.pdf** donde se documentará:
  - En una primera sección, un listado sucinto de qué ejercicios y opciones se han realizado exactamente. Se indicará claramente qué versión de los diferentes algoritmos se ha implementado. Se enfatizarán en su caso los aspectos que puedan evaluarse favorablemente.
  - Comentar en particular las variantes implementadas (en su caso) en el ejercicio 4; los resultados comparativos de las evaluaciones en el ejercicio 5 (qué algoritmos parecen funcionar mejor); el tamaño de los grafos simulados y los resultados de los análisis del ejercicio 6; los tiempos de ejecución en el ejercicio 7; documentar adecuadamente los ejercicios y pruebas que se hayan desarrollado en el ejercicio 8.
  - Y cualquier otro aspecto que el estudiante considere oportuno destacar.

  La calidad de la memoria representará orientativamente el 10% de la puntuación de los ejercicios que se documentan en la misma.

No se deberán incluir en el .zip ninguno de los materiales proporcionados por el profesor (conjuntos de datos, clases Java, etc.), ni los archivos de proyecto Netbeans o Eclipse –ni por supuesto ningún .class!

El fichero de entrega se enviará por el enlace habilitado al efecto en el curso **Moodle** de la asignatura.

Calificación

Esta práctica se calificará con una puntuación de 0 a 10 atendiendo a las puntuaciones individuales de ejercicios y apartados dadas en el enunciado. El peso de la nota de esta práctica en la calificación final de prácticas es del **30%**.

La calificación se basará en el **número** de ejercicios realizados y la **calidad** de los mismos. La puntuación que se indica en cada apartado es orientativa, en principio se aplicará tal cual se refleja pero podrá matizarse por criterios de buen sentido si se da el caso.

Para dar por válida la realización de un ejercicio, el código deberá funcionar (a la primera) integrado con las clases que se facilitan. El profesor comprobará este aspecto añadiendo las clases entregadas por el estudiante a las clases facilitadas en la práctica, ejecutando los programas `Test` así como otros main de prueba adicionales.

La corrección de las implementaciones se observará por la **coherencia de los resultados** (por ejemplo, las métricas sobre los algoritmos de recomendación), y se valorará la eficiencia en tiempo de ejecución (en particular sobre el conjunto de datos grande).