# Búsqueda y Minería de Información 2020-2021

### Universidad Autónoma de Madrid, Escuela Politécnica Superior

### Grado en Ingeniería Informática 4º curso

# Práctica 4 – Sistemas de recomendación y análisis de redes sociales

## Fechas

- Entrega: lunes 10 de mayo (23:59h)

## 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.
- 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 proporcionan software y datos para la realización de la práctica, que se divide en dos bloques.

**Bloque I – Sistemas de recomendación**

- Un esqueleto de clases y funciones en un archivo **recsys.py**, donde el estudiante desarrollará sus implementaciones. De modo similar a las prácticas anteriores, se proporciona un programa **recsys-main.py** que deberá funcionar con el código a implementar por el estudiante.
- Dos conjuntos de datos que incluyen ratings asignados por usuarios a películas: un conjunto pequeño de datos ficticios, y otro conjunto de datos reales disponibles en la Web de MovieLens.
  - Conjunto pequeño de prueba toy-ratings.dat con datos ficticios de ratings, así como un split manual fijo de estos datos en toy-train.dat y toy-test.dat. Se incluyen estos archivos en la carpeta **data**.
  - Conjunto de datos real en la Web de MovieLens, disponible en ml-latest-small.zip en https://grouplens.org/datasets/movielens/latest. De los archivos disponibles, se utilizará sólamente ratings.csv.
- Un documento de texto **recsys-output.txt** con la correspondiente salida estándar que deberá producir la ejecución del programa recsys-main.py.

**Bloque II – Análisis de redes sociales**

- Un esqueleto de clases y funciones en un archivo **sna.py**, donde el estudiante desarrollará sus implementaciones. De modo similar a las prácticas anteriores, se proporciona un programa **sna-main.py** que deberá funcionar con el código a implementar por el estudiante.
- Redes sociales de prueba:
  - Tres redes pequeñas de prueba proporcionadas en la carpeta **graph**.
  - Red real disponible (facebook_combined) en https://snap.stanford.edu/data/egonets-Facebook.html.
  - 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 sna-main.py.

## Ejercicios

## Bloque I – Recomendación

Todo el código implementado en este bloque se incluirá en un archivo **recsys.py**. Los esqueletos que se proporcionan en este archivo son a modo de guía –el estudiante puede modificarlo todo libremente, siempre que el programa **recsys-main.py** funcione correctamente **sin cambios**.

Se implementarán estructuras y diferentes algoritmos para el desarrollo de sistemas de recomendación.

**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** (ratings) para los algoritmos de recomendación. La funcionalidad se implementará en una clase `Ratings`, que permitirá leer los datos de un fichero de texto, añadir ratings y acceder a ellos, así como un método que genere dos particiones aleatorias de entrenamiento y test, para evaluar y comparar la efectividad de diferentes algoritmos de recomendación. **Nota:** si algún archivo de ratings tiene cabeceras, eliminarlas a mano.

Implementar asimismo una clase `Recommendation` para almacenar la **salida** de un recomendador, que consistirá en un diccionario con un ránking por usuario. Se facilita una clase `Ranking` basada en heap, similar a la utilizada para motores de búsqueda en la práctica anterior, para almacenar los ránkings de recomendación –la diferencia es que ahora no se devuelven ránkings en respuesta a consultas, sino un ránking por usuario de forma proactiva y en bloque para todos los usuarios presentes en el conjunto de ratings dado (sin que los usuarios lo "pidan" explícitamente).

Implementar un primer **recomendador simple** por rating promedio en una clase `AverageRecommender`. El recomendador sólo recomendará ítems que tengan un mínimo de ratings, mínimo que se indicará como parámetro 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. También se proporciona `RandomRecommender`, que se utiliza en ocasiones como referencia en experimentos.

### 2. Filtrado colaborativo kNN (3pt)

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

- Implementar la clase `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 `Ranking`, que utiliza un heap de ránking, para construir los vecindarios.
- Implementar una variante normalizada `NormUserKNNRecommender` De forma similar a la recomendación por rating promedio, el algoritmo exigirá un mínimo de ratings de vecinos para aceptar recomendar un ítem (con ello se mejora el acierto de la recomendación).

### 3. Ampliación de algoritmos (1pt)

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

- Un algoritmo colaborativo por **vecinos próximos orientado a ítem**: `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.
- **Otras variantes** adicionales a elección del estudiante, por ejemlpo: similitud de Pearson, kNN centrado en la media.

Para probar los métodos deberá incluirse en la entrega **recsys.py** una función `student_test()` que ilustre la ejecución de las variantes adicionales.

### 4. Evaluación (1pt)

Se desarrollarán clases que permitan calcular métricas para evaluar y comparar el acierto de los recomendadores: se implementarán precisión y recall. **En la memoria se incluirá una tabla** con los valores de las métricas (dos columnas) más el tiempo de ejecución (una columna más) sobre todos los algoritmos implementados (filas).

Opcionalmente, se podrán implementar otras métricas a elección del estudiante (nDCG, etc.), cuya prueba se incluirá en la función `student_test()`.

## Bloque II – Análisis de redes sociales

Todo el código implementado en este bloque se incluirá en un archivo **sna.py**. Los esqueletos que se proporcionan en este archivo son a modo de guía –el estudiante puede modificarlo todo libremente, siempre que el programa **sna-main.py** funcione correctamente **sin cambios**.

Para simplificar, en los ejercicios que siguen supondremos que las redes son no dirigidas.

### 5. **Preliminares** (2pt)

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 puede utilizar para ello cualquier herramienta (como NetworkX, o el entorno interactivo de Gephi), o bien 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, 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.

### 6. **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.

### 7. **Ejercicio libre** (1pt)

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, etc., integradas en la misma jerarquía de métricas que el ejercicio anterior. (Para la modularidad se incluye en los datos proporcionados una partición por tipos del grafo small3; consultar con el profesor 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.

Para este ejercicio deberá incluirse en **sna.py** una función adicional `student_test()` ilustrando la ejecución de las métricas y algoritmos implementados.

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 funciones y/o clases adicionales a las que se indican en el enunciado. Algunas vendrán dadas por su aparición en los propios programas main, y otras por conveniencia a criterio del estudiante.

Igual que en prácticas anteriores, no deberán editarse los programas **recsys-main.py** y **sna-main.py**. Estos programas deberán ejecutar sin errores "a la primera" con el código entregado por el estudiante (naturalmente con salvedad de los ejercicios que no se hayan implementado).

Para el ejercicio 5, se incluirá 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:

| | 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 la red de Facebook, en una tabla con la siguiente estructura:

| | Facebook |
|---|---|
| Coef. clustering usuario | |
| Embededness | |
| Coef. clustering global | |
| Asortatividad | |

## Entrega

La entrega consistirá en un único fichero zip con el nombre **bmi-p3-XX.zip**, donde XX debe sustituirse por el número de pareja (01, 02, …, 10, …). Este fichero contendrá:

- Los ficheros **recsys.py** y **sna.py** con las implementaciones solicitadas en los bloques I y II respectivamente.
- Una carpeta **data/** con los grafos Erdös-Rényi y Barabási-Albert generados en el ejercicio 5.
- Una **memoria bmi-p3-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 la variante implementada (en su caso) en el ejercicio 3; los resultados comparativos de las evaluaciones en el ejercicio 4 (qué algoritmos parecen funcionar mejor); el tamaño de los grafos simulados y los resultados de los análisis del ejercicio 5; los tiempos de ejecución en el ejercicio 6; documentar adecuadamente los ejercicios y pruebas que se hayan desarrollado en el ejercicio 7.
  - 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, programas main), ni archivos de proyecto.

## 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 **40%**.

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 main facilitados así como otros 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.