Sistemas de recomendación

Un sistema de recomendación es básicamente mostrarle elementos a un usuario que le sean útiles.
Los sistemas de recomendación vienen para optimizar eso que uno busca con menos esfuerzo. Si sigo consumiendo en una página gracias al sistema de recomendación cumplió su objetivo.

Tipos de interacción feedback

- **explícito**: el usuario interactúa/da feedback de forma explícita con los elementos.
- **implícito**: el usuario no da un feedback de forma directa sino que se interpretan sus acciones dentro de un sistema como un feedback leve.

Explicit	Implicit
Un usuario no da siempre feedback explícito.	Todos los usuarios tienen feedback implícito.
Tiene más resolución	Solo mide acciones
El usuario puede ingresar incorrectamente	El usuario puede tener ruido en la navegación

El historial de visitas es muy poderoso. Esto es porque tenemos mucha información del historial, más que de las compras. Lo interesante es que para el historial no tuve que subir ningún dato.

¿Qué buscan los algoritmos de recomendación? ensanchar el **funnel** (aumentar la cantidad de gente que logran el objetivo (por ej comprar))

Funnel: Embudo. En el camino se van perdiendo usuarios, por eso se crea el embudo. La idea es que el funnel no sea muy chico, sino ensanchar el funnel.

Es muy importante el tracking en los sistemas de recomendación porque a veces sirve para comparar entre sistemas de recomendación. Se elige qué variante queda. Se ponen a competir distintos sistemas.

Tipos de status a la hora de recomendar

- **Cold start:** el usuario o elemento a recomendar acaban de ser ingresados al sistema. No tienen interacciones ni historial, solo los datos cargados al momento de ingresarlos al sistema
- **Warm start/ Normal State:** el usuario o elemento a recomendar ya tiene interacciones en el sistema y tiene cargada “suficiente” información

¿Cuál es la **base en la cual realizar la recomendación**?

- **user-item**: parado desde el usuario, recomendarle un elemento.
- **item-item**: parado desde un elemento, recomendar otro elemento.

¿Cuáles son las **categorías de sistemas de recomendación**?
basados en popularidad, contenido o interaccion. Se suele usar una mezcla de los 3

serendipity: cuando el algoritmo me muestra algo que era dificil de encontrar y que me gusta. Si tiene un alto serendipity, quiere decir que me recomienda algo que me habría costado encontrarlo sola. (Se busca que tenga una alta serendipity)

Basados en popularidad
Son aquellos que usan datos de popularidad para recomendar. Ejemplos:

popularidad = cantidad reproducida (en el último día/ unidad de tiempo).
popularidad = promedio de puntuación (estrellas/likes vs dislikes).
popularidad = cantidad reproducida/ cantidad mostrada.

Pros
- Fácil de implementar. Fáciles de entender. No son costosos computacionalmente.
- No necesitan info de los usuarios o los elementos. No tienen problema de cold-start con el usuario.
- Pueden influenciar comportamiento (presión social).

Cons
- No se ajustan al usuario ni contexto, a lo sumo aplican algún filtro muy simple (ej: en netflix-> argentina). El filtro tiene que ser trivial.
- No recomiendan inteligentemente, performance mediocre.
- Probablemente la mayoría de las recomendaciones hubiera encontrado sin ayuda de la recomendación. BAJA serendipity

Basadas en contenido
Utilizan atributos del usuario y/o de los elementos, y, en conjunto con un poco de lógica de negocios

Ejemplo: Si le di like a una película, tengo datos sobre la película por ejemplo género.

Pros
- Medianamente fáciles de implementar.
- Fáciles de entender, explicar y debuggear (internamente).
- Se ajustan al usuario/elemento (de la información usada).
- Más serendipity que popular.

Contras
- Tenes que comparar al item con todos los demas
- No funciona para usuarios o elementos que no tienen cargada la información usada por el algoritmo.
- Su eficacia depende directamente de si los datos elegidos (y/o disponibles) pueden captar las preferencias de los usuarios.
- No están usando la información de otros usuarios.

Collaborative
Usa las interacciones de todos los usuarios con todos los elementos.
Somos más propensos a que nos gusten las cosas que le gustaron a personas con gustos similares a los nuestros.
La idea es buscar usuarios con gustos similares y recomendar cosas con las que no se haya interactuado todavía.

Pros
- No necesita la información de los elementos o usuarios, solo las interacciones.
- Funciona bastante bien.
- Permite armar representaciones tanto de los usuarios como de los elementos.

Contras
- Sufre el problema de cold-start.
- Matrices ralas (vacías). Hay información vacía en las matrices si por ejemplo di poco feedback explícito. Hay más ceros que unos.
- Costoso en memoria y procesamiento, pero se puede mitigar.

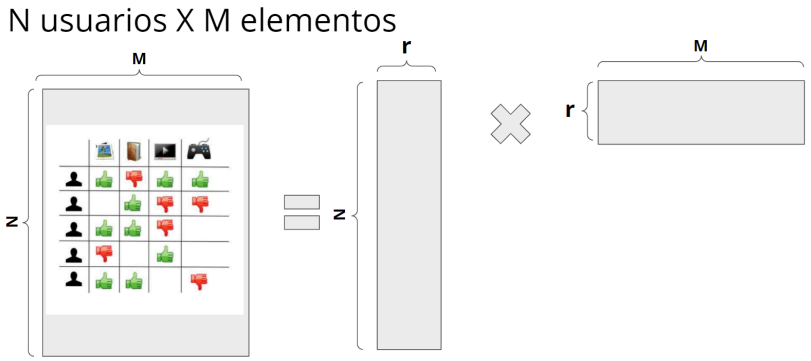
Implementaciones de collaborative

CF (collaborative filtering) simple coseno (user item): si queremos recomendarle ítems, podemos mapear a cada usuario a un vector. **0 representa que no** compro/vio/etc a ese elemento **y 1 que si**. La idea es que usuarios similares tendrán vectores similares. Se buscan usuarios similares y recomendamos elementos con los que el usuario no haya interactuado. Hay que buscar los ceros similares en este vector.
Calculamos la similaridad con otros usuarios con: $\text{simil}(x, y) = \cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{||\vec{x}|| \times ||\vec{y}||}$

Nos quedamos con los k top usuarios. Se rankea el elemento i dependiendo del ranking que le dio cada usuario.

Sin filtering simple coseno (item item): lo mismo que lo anterior, ahora mirado desde el ítem.
sim(x, y) donde x e y son elementos. Conseguimos los k vecinos más similares al ítem x.

Factorizando matrices (svd y svd++):
Descomponen las matrices. La primera de dimensión R y la otra de dimensión M.
Donde falta la opinión, de forma indirecta lo que hago es predecir si al usuario le va a gustar o no cada ítem por la simple descomposición de la matriz
Los usuarios tienen el primer vector (n x r) y cada elemento de la segunda un ítem (r x m).
Lo importante es que no cargamos nada de la información del usuario o del ítem, estamos viendo las interacciones.



Si se hace la distancia coseno del primer vector, tenes usuarios similares. En la dirección horizontal tengo usuarios similares. De la misma manera, le haces distancia coseno al segundo y tienes items similares. En la dirección vertical tengo las características de cada ítem.

Redes neuronales (word2vec): Represento a cada ítem con un id y guardo la secuencia en que el usuario fue interactuando. Cada secuencia de ítems se toma como una frase y el modelo aprende por cercanía. peliculas parecidas tienen parecido vector de palabras, van a estar en la misma vecindad

Métricas: Quiero recomendarle cosas nuevas a un usuario, pero quiero saber si le va a interesar.

Precisión @ K: Tomamos las top N recomendaciones y medimos cuantas de esas son relevantes.

Recall @ K : Tomamos las top N recomendaciones y medimos qué proporción de las relevantes son.

Estas métricas no consideran el orden.

Métricas con rank aware:

MRR: lo que busca es el primer elemento relevante.

MAP @ N: Para cada usuario recomendamos una lista ordenada de largo N. A diferencia de Precision@N queremos considerar el hecho de que es una lista ordenada. Queremos equivocarnos poco en las primeras recomendaciones.

NDCG@N: ganancia acumulada de descuento normalizado

MRR, MAP y NDCG son las más importantes. Si me importa el score, NDCG y sino MAP que me dice si es relevante o no.

Entrenamiento

Si un usuario nunca tuvo una interacción con una película, no podemos inventar que le va a gustar o no. Vamos a censurar reviews. Esos valores que tapamos, los vamos a tratar de predecir y ver que tan lejos estamos cuando predecimos.

Otra opción que podemos hacer es, a partir del historial que tenemos del usuario, comparamos las próximas películas que ve y se comparan con las otras recomendaciones que le doy. Vale lo de train split pero vamos a tener el problema de cold start. Es algo que hay que tener en cuenta.