Tecnológico de Monterrey

Avance de proyecto 4: Sistema de Recomendación

Maestría en inteligencia artificial aplicada
Asignatura : Análisis de grandes volúmenes de datos

Equipo:
Abraham Cabanzo Jimenez - A01794355
Ignacio Antonio Ruiz Guerra - A00889972
Moisés Díaz Malagón - A01208580

16 de junio de 2024

1. Detalle de implementación final de sistemas de recomendación

A continuación se detalla la implementación de los 5 algoritmos implementados. El detalle de código comentado puede encontrarse en el notebook correspondiente a esta entrega:

https://github.com/moisesdiazm/sistema-recomendacion-bigdata-mna/blob/main/Proyecto_Avance_4_%2337.ipynb

Se recomienda seguir el código mientras se lee la descripción de cada algoritmo que se muestra a continuación.

1.1 Algoritmo 1: Sistema básico, recomendador global

Este algoritmo realiza recomendaciones generales con base en la popularidad de las películas. Para ello hace uso de la función *value_counts* de pandas. Esta función toma la base de datos de ratings y agrupa las películas por id y cuenta cuántas veces aparece cada una, que representa la cantidad de veces que fue vista. El recomendador regresa el top 10 de las películas más vistas.

1.2 Algoritmo 2: Sistema de filtro colaborativo utilizando NearestNeighbors

Este algoritmo recomienda películas con base en la matriz de usuario-item, no toma en cuenta las características de las películas. Funciona bajo la premisa de que a usuarios similares les interesan las mismas películas.

El algoritmo se inicializa con la base de datos de películas y ratings, posteriormente se calcula la matriz de usuarios-películas mediante el siguiente procedimiento:

1. Obtenemos los ids de usuarios y películas únicos en cada base de datos
2. Creamos una matriz utilizando la función *csr_matrix*, que genera una matriz dispersa donde los datos serán los ratings, las filas los ids de usuario y las columnas los ids de películas. Guardamos esta matriz en memoria por si se quiere reutilizar más adelante.
3. Calculamos los clusters de vecinos más cercanos utilizando *NearestNeighbors* de Sklearn utilizando métrica de similaridad de coseno. Entrenamos el algoritmo con la matriz de usuario-película.
4. Ejecutamos inferencia en el modelo entrenado dándole como argumento al algoritmo el vector de ratings de todos los usuarios para una película dada.
5. La inferencia muestra los vecinos más cercanos en forma de distancias e índices de las películas. Se ordenan las distancias de mayor a menor y se regresan las 10 primeras, omitiendo la primera que es la más cercana y es la película ingresada como argumento del algoritmo.

1.3 Algoritmo 3: Sistema basado en contenido utilizando one-hot encoding y similaridad de coseno

Este algoritmo recomienda películas con base en la matriz de similaridad entre películas, no toma en cuenta las interacciones de usuarios. Funciona bajo la premisa de recomendar películas similares a una vista. Toma en cuenta las características de las películas. Soporta arranque en frío.

El algoritmo se inicializa con la base de datos de películas únicamente, posteriormente se calcula la matriz de similaridad mediante los siguientes pasos:

1. Los géneros se pre-procesan pues se tienen como una cadena de caracteres separados por el caracter “|”. Se separan para tenerlo como una lista de palabras.
2. Se cuentan el número de géneros únicos y la cantidad de cada uno, se descartan las películas que no tienen género.
3. El título se limpia para que no contenga el año en que se hizo la película.
4. De la fecha de publicación se extrae el año y del año se extrae la década.
5. Se generan columnas para cada género y se asigna un 1 a las películas que contienen dicho género. Esto es, la generación de vectores one-hot para los géneros.
6. Se obtienen mediante la librería pandas los vectores one-hot para la columna de década.
7. Finalmente se concatenan las columnas generadas a los datos originales.
8. Utilizando la librería SKlearn se calcula la similaridad de coseno entre películas. Se genera una tabla donde tanto las columnas como las filas son ids de películas.

Finalmente, para realizar una recomendación, dado un id de película que el usuario ha visto, se obtiene el vector de similaridad correspondiente de la matriz de similaridad. Se ordenan los elementos de mayor a menor y se seleccionan los 10 valores más altos, que corresponde con las películas más similares.

1.4 Algoritmo 4: Sistema de filtro colaborativo complejo, utilizando embeddings y redes neuronales.

El algoritmo de recomendación avanzado es un tipo de sistema de filtro colaborativo basado en un modelo de deep learning. El diseño fue propuesto en He et al. (2017) y consiste en la utilización de redes neuronales para resolver un problema de clasificación binaria, si el usuario tendrá o no interacción con una película dada. Este sistema busca derivar las relaciones entre usuarios y películas que pueden ayudar a determinar si la película será vista o no por el usuario.

La técnica consiste en reemplazar el cálculo de similaridad con una arquitectura de redes neuronales que puede aprender una función arbitraria, y que incluso puede generalizar una factorización de matrices por medio de las no-linealidades que permiten las múltiples capas de perceptrones (He et al., 2017).

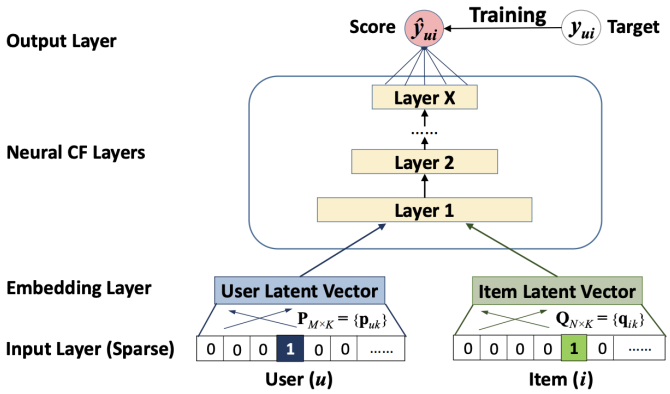

Figura 1. Sistema de recomendación de filtro colaborativo propuesto por He et al. (2017)

El algoritmo comienza representando el id de usuario y el id de la película como un vectores one-hot. Posteriormente ambos vectores son convertidos en embeddings de menor dimensionalidad, se eligió utilizar 10 dimensiones.

Utilizando el framework PyTorch se definen dos capas de tipo Embedding cuyo número de entradas será la cantidad de dígitos 1 y 0 utilizados para representar los ids de usuarios y películas. El tamaño de la salida de esta capa es de 10 elementos, el tamaño de los embeddings. Esta capa es entrenable, lo que significa que sus parámetros son ajustados durante el entrenamiento.

```
   | Name             | Type      | Params
-------------------------------------------
0  | user_embedding   | Embedding | 36.2 K
1  | movie_embedding  | Embedding | 23.7 K
```

Figura 2. Capas de Embedding, con la cantidad de parámetros que se entrenarán.

Posteriormente, los vectores de salida de los embedding del usuario y película son concatenados para formar una sola entrada a las capas de la red neuronal. Considerando que los vectores tienen dimensión de 6 elementos cada uno, el vector de entrada a la red neuronal después de la concatenación tiene una dimensión de 12 elementos.

Se utilizan 3 capas de 64, 32 y 1 neuronas respectivamente. El tipo de capa es Fully Connected, lo que significa que cada una de las neuronas de una capa están conectadas a cada una de las neuronas de la siguiente capa. De esta forma para la primera capa tenemos una dimensión de entrada de 12 elementos (embeddings concatenados) y una salida de 64 elementos (cantidad de neuronas en capa 1). Para la segunda capa tenemos 64 elementos de entrada y 32 de salida, para la tercera capa tenemos 32 entradas y una única salida (una neurona) que será utilizada para predecir un valor entre 0 y 1 que representa la probabilidad de que la película sea vista o no.

Para capas intermedias utilizaremos la función de activación ReLU. En el caso de la salida, debido a que estamos tratando con un problema de clasificación binaria se utiliza la función Sigmoide que entrega valores entre 0 y 1 de forma suavizada.

```
    | Name              | Type       | Params
-----------------------------------------------
  2 | fc1               | Linear     | 832
  3 | fc2               | Linear     | 2.1 K
  4 | fc3               | Linear     | 33
```

Figura 2. Capas Fully Connected, con la cantidad de parámetros que se entrenarán.

La predicción final es comparada con el valor real de que la película fue vista por el usuario. Se calcula la función de pérdida (error) utilizando Binary Cross Entropy. Para la optimización se utiliza Adam de PyTorch que realiza la optimización de gradiente descendente con técnicas que favorecen una convergencia rápida y estable.

Previo al entrenamiento del algoritmo se realizan los siguientes ajustes:

- Dado que se tiene un problema de clasificación binaria, las interacciones a considerar serán de si el usuario vio o no una película en lugar de la calificación asignada. Para ello se asume que si el usuario calificó la película entonces la vio, y si no la calificó no la vio.
- Se realiza una partición de datos en conjunto de entrenamiento y validación. Para ello se considerará la última interacción del usuario como parte del conjunto de validación y el resto como parte del conjunto de entrenamiento.

Para la inferencia se ingresa al algoritmo un usuario dado así como una lista de películas no vistas. Se obtiene para cada una de ellas la probabilidad de que la vea el usuario. Se regresa el top 10 de películas con probabilidad más alta de ser vista por el usuario.

1.5 Algoritmo 5: Sistema basado en contenido utilizando técnicas de NLP

Este algoritmo es un filtro basado en contenido que utiliza técnicas de NLP. Utiliza la base de datos IMDB, y comienza filtrando los textos tal como descripción título y géneros, eliminando caracteres no alfabéticos, signos de puntuación y números. Adicionalmente convierte todo el texto a minúsculas y remueve stopwords que representan palabras comunes del vocabulario y que no aportan información.

Una vez preprocesados los textos, utiliza un vectorizador de tipo TF-IDF, que permite convertir cada palabra de un enunciado en una lista de números. La forma de convertirlos consiste en una multiplicación de dos factores:

- La frecuencia de la palabra en todos los documentos.
- La relación de en cuántas oraciones aparece una palabra contra cuantas oraciones tenemos.

Del tamaño de la matriz generada se observa que se tienen 1000 oraciones y que el vocabulario es de 5714 palabras distintas (tokens), entonces para cada oración se

tiene un vector numérico de tamaño 5714 que representa la importancia de cada palabra dada en la oración.

Se calcula la matriz de similaridad mediante la función *linear_kernel* de SKlearn utilizando como entrada la vectorización hecha por TF-IDF. Nótese que es similar al algoritmo 3, sin embargo en este caso la matriz no fue generada por medio de one-hot encoding, sino utilizando el texto de la descripción de películas, géneros y títulos.

Una vez generada la matriz de similaridad entre Ids de películas, la inferencia sucede al seleccionar un id de película y obtener el top 10 más similar de esta matriz.

2. Evaluación integral del desempeño de los modelos

Para la evaluación de estos modelos (sin considerar el basado en contenido, TF-IDF pues no contamos con datos etiquetados de prueba que nos permitan validar el rendimiento), se utilizan para este proyecto, tres métricas.

Hit Ratio @10

Es una métrica que evalúa la capacidad general del sistema para incluir al menos un ítem relevante en un conjunto fijo de recomendaciones. Es útil para medir la efectividad básica y asegurar que las recomendaciones tienen al menos algo de relevancia. Se decidió utilizarla por tres razones:

Primera, mide la capacidad del sistema para recomendar al menos un ítem relevante (ya visto por el usuario) dentro de las primeras 10 recomendaciones. Esto es directamente aplicable en escenarios donde el objetivo es asegurar que el usuario encuentre algo que le interese en un conjunto limitado de recomendaciones.

Segunda**,** es una métrica fácil de entender y comunicar, tanto a desarrolladores como a partes interesadas no técnicas.

Y la tercera es que nos permite evaluar si el sistema de recomendación está cumpliendo con la tarea básica de incluir ítems relevantes en el conjunto de recomendaciones. Es un buen indicador de si el sistema está haciendo recomendaciones útiles en un sentido general.

$$\text{Hit Ratio@k} = \frac{\text{Number of hits in top-k recommendations}}{\text{Total number of users}}$$

Precision Ratio @10

Nos permite medir la proporción de ítems relevantes en las primeras recomendaciones, proporcionando una evaluación de la precisión y calidad general de las recomendaciones. Ayuda a garantizar que la mayoría de las recomendaciones sean útiles y no solo un pequeño subconjunto. Además, es útil para comparar diferentes modelos y optimizaciones en términos de qué tan bien están filtrando y presentando contenido relevante.

A diferencia del Hit Ratio, que solo necesita una coincidencia, la precisión se enfoca en cuántas de las recomendaciones son realmente útiles. Esto ayuda a evitar que los sistemas presenten muchas recomendaciones irrelevantes, mejorando la experiencia del usuario.

$$\text{Precision@k} = \frac{\text{Number of relevant items in top-k recommendations}}{k}$$

Mean Reciprocal Rank (MRR)

Esta métrica evalúa la calidad de la clasificación de las recomendaciones, priorizando la aparición temprana de ítems relevantes. Es útil para mejorar la experiencia del usuario al asegurar que las recomendaciones más relevantes aparezcan primero.
Además, se enfoca en la posición de la primera recomendación relevante. Esto es importante porque los usuarios tienden a interactuar más con los ítems que aparecen al principio de la lista.
También ayuda a evaluar no solo si se incluyen recomendaciones relevantes, sino también cuán rápidamente aparecen en la lista. Un sistema que coloca ítems relevantes en las primeras posiciones será más valioso para los usuarios.

$$\text{MRR} = \frac{1}{\text{Total number of users}} \sum_{i=1}^{\text{Total number of users}} \frac{1}{\text{Rank of first relevant item for user } i}$$

Los resultados al calcular las métricas de desempeño para nuestros modelos, son las que siguen :

Algoritmo	HitRatio@10	PrecisionRatio@10	Mean Reciprocal Rank (MRR)
Sistema básico: recomendador global (Actividad 4.6)	NA	NA	NA
Sistema de filtro colaborativo: KNeighbors (Actividad 4.6)	57.00%. Es el modelo más efectivo, indicando que más de la mitad de las veces, las recomendaciones incluyen al menos una película relevante.	57.00%. Este indicador sugiere que este modelo es el más adecuado.	18.83%. Es ligeramente menos efectivo que el de la red neuronal.
Sistema basado en contenido: one-hot encoding con similaridad coseno (Actividad 4.6)	52.00%. Es ligeramente menos efectivo pero sigue siendo bastante bueno.	52.00%. Es bueno, pero podría mejorarse.	18.37%. A pesar de no figurar en los primeros lugares, la métrica indica, que este modelo, sigue siendo efectivo, al compararlo con los otros dos modelos.
Sistema de filtro colaborativo complejo: capa de embedding con redes neuronales FC. (Actividad 4.6)	48.00%.sugiere que necesita optimización para superar a los modelos más simples.	45.00%. Estos resultados, sugieren que debe revisarse para alcanzar un mayor rendimiento.	18.88%. Bajo esta métrica, este modelo resultó ser el mejor.

Conclusiones:

Los resultados nos muestran que la personalización es esencial para mejorar el hit ratio, la precisión y el MRR en sistemas de recomendación. Modelos simples, como el Collaborative Basic y el Content Basic, a menudo superan a los más complejos, como los basados en Deep Learning, que requieren ajustes cuidadosos para justificar su complejidad.

Es crucial seguir optimizando los modelos, especialmente los avanzados, para mejorar su rendimiento. Además, combinar diferentes tipos de modelos mediante enfoques híbridos puede aprovechar sus fortalezas y mejorar la calidad de las recomendaciones.

Así pues, no debemos olvidar que la personalización y la combinación de modelos son claves para la efectividad y mejora continua de los sistemas de recomendación.

3. Documentación del código base y algoritmos implementados

La documentación del código se encuentra en el notebook de jupyter. Cada algoritmo cuenta con la documentación detallada de uso, argumentos de entrada y salidas de cada función.

En el Anexo 1 de este documento se puede encontrar la documentación del código para todos los algoritmos implementados.

Referencias:

He, X., Liao, L., Zhang, H., Nie, L., Hu, X. y Chua, T. (2017). *Neural Collaborative Filtering*. Recuperado de: https://arxiv.org/pdf/1708.05031

ANEXO 1: Documentación de algoritmos

ANEXO 1: Documentación de algoritmos

`global_recommender(top_n, movies, ratings)`

Recomendador global de películas. Hace recomendaciones generales con base en la popularidad de las películas, las más vistas. En este caso se considerarán como las mas vistas, las más veces calificadas.

Parameters:

- `top_n` (int): Número de películas a recomendar.
- `movies` (pd.DataFrame): DataFrame con las películas.
- `ratings` (pd.DataFrame): DataFrame con las calificaciones.

Returns:

- `tupla`: Título de las películas recomendadas, id de las películas recomendadas.

class `CollaborativeFilterBasic`

Filtro colaborativo básico, basado en la matriz de usuario-item. No toma en cuenta las características de las películas. Funciona bajo la premisa de que a usuarios similares les interesan las mismas cosas. Aprende de los intereses de la población.

`__init__(movies, ratings)`

Inicializa una instancia del filtro colaborativo básico.

Parameters:

- `movies` (pd.DataFrame): DataFrame con las películas.
- `ratings` (pd.DataFrame): DataFrame con las calificaciones de las películas por usuario.

Returns: None

`find_movieid(title)`

Utiliza la librería fuzzywuzzy para encontrar el id de una película dado su título. Permite encontrar películas incluso si el título está mal escrito o no coincide exactamente.

Parameters:

- `title` (str): Título de la película a buscar.

Returns:

- `int`: Id de la película encontrada.

`recommend(movie_title=None, movie_id=None, k=10)`

Hace recomendación de 10 películas similares a una dada.

Parameters:

- `movie_title` (str): Título de la película a la que se le quiere encontrar películas similares.
- `movie_id` (int): Id de la película a la que se le quiere encontrar películas similares. Si no se proporciona, se buscará el id a partir del título, utilizando la librería fuzzywuzzy.
- `k` (int): Número de películas a recomendar.

Returns:

- `tupla`: Título de las 10 películas recomendadas, id de las 10 películas recomendadas.

`train(k=10, metric='cosine')`

Entrena el modelo de filtro colaborativo básico.

Parameters:

- `k` (int): Número de vecinos más cercanos a considerar.
- `metric` (str): Medida de distancia a utilizar. Puede ser ‘cosine’ o ‘euclidean’.

Returns: None

class `CollaborativeFilterDL`

Algoritmo avanzado. Filtro colaborativo basado en deep learning. Utiliza retroalimentación implícita. Es decir, todas las películas que un usuario califica se consideran como vistas.

Durante el entrenamiento se calculan los embeddings tanto de los usuarios como de las películas así como los pesos de las redes neuronales que modelan la función de similaridad entre los embeddings. Representa un problema de clasificación binaria, donde la etiqueta es 1 si el usuario vio la película y 0 si no la vio. Requiere ser entrenado y actualizado para nuevos usuarios y películas.

`__init__(ratings, movies)`

Inicializa una instancia del filtro colaborativo basado en deep learning.

Parameters:

- `ratings` (pd.DataFrame): DataFrame con las calificaciones de las películas por usuario.
- `movies` (pd.DataFrame): DataFrame con las películas.

Returns: None

`find_movieid(title)`

Utiliza la librería fuzzywuzzy para encontrar el id de una película dado su título. Permite encontrar películas incluso si el título está mal escrito o no coincide exactamente.

Parameters:

- `title` (str): Título de la película a buscar.

Returns:

- `int`: Id de la película encontrada.

`recommend(user_id, model)`

Hace 10 recomendaciones de películas para un usuario dado.

Parameters:

- `user_id` (int): ID del usuario.
- `model` (DLRecommenderModel): Modelo de deep learning entrenado.

Returns:

- `list`: Lista con los 10 títulos de las películas recomendadas.

`train()`

Entrena el modelo de filtro colaborativo basado en deep learning.

Returns: None

class `ContentBasedBasic`

Filtro basado en contenido. No toma en cuenta a los usuarios, sino las características de las películas, pero a diferencia del primer algoritmo, este no utiliza técnicas de NLP.

`__init__(movies)`

Inicializa una instancia del filtro basado en contenido.

Parameters:

- `movies` (pd.DataFrame): DataFrame con las películas.

Returns: None

`find_movie_idx(title)`

Utiliza la librería fuzzywuzzy para encontrar el índice de una película dado su título. Permite encontrar películas incluso si el título está mal escrito o no coincide exactamente.

Parameters:

- `title` (str): Título de la película a buscar.

Returns:

- `int`: índice de la película encontrada, dentro del DataFrame de películas.

`recommend(movie_title=None, movie_id=None, top_n=10)`

Hace recomendación de películas similares a una dada basandose en el contenido de las películas.

Parameters:

- `movie_title` (str): Título de la película a la que se le quiere encontrar películas similares.

- `movie_id` (int): Id de la película a la que se le quiere encontrar películas similares. Si no se proporciona, se buscará el id a partir del título, utilizando la librería fuzzywuzzy.
- `top_n` (int): Número de películas a recomendar.

Returns:

- `tupla`: Título de las películas recomendadas, id de las películas recomendadas.

`train()`

Entrena el modelo de filtro basado en contenido. Calcula la matriz de similaridad.

Returns: None

class`ContentBasedNLP`

Sistema de recomendación basado en contenido. Técnicas de NLP, en específico TF-IDF y similaridad coseno.

`recommend(movie_name)`

Hace recomendación de 10 películas similares a una dada basandose en el contenido de las películas.

Parameters:

- `movie_name` (str): Título de la película a la que se le quiere encontrar películas similares.

Returns:

- `list`: Título de las 10 películas recomendadas.

class `DLRecommenderModel`

Sistema de recomendación que utiliza un modelo de deep learning para implementar un filtro colaborativo que modela una función de similaridad entre vectores (embeddings) de usuarios y películas.

`__init__(n_users, n_movies, ratings, all_movieids, negative_examples_qty)`

Inicializa una instancia del modelo de deep learning.

Parameters:

- `n_users` (int): Cantidad de usuarios.
- `n_movies` (int): Cantidad de películas.
- `ratings` (pd.DataFrame): DataFrame con las calificaciones de las películas por usuario.
- `all_movieids` (list): Lista con los IDs de todas las películas.
- `negative_examples_qty` (int): Cantidad de ejemplos negativos a generar.

Returns: None

`configure_optimizers()`

Configura el optimizador para el modelo.

Returns:

- `torch.optim.Optimizer`: Optimizador a utilizar.

`forward(user_id, movie_id)`

Hace una pasada hacia adelante en el modelo. Tanto para entrenamiento compara inferencia.

Parameters:

- `user_id` (int): ID del usuario.
- `movie_id` (int): ID de la película.

Returns:

- `float`: porcentaje de probabilidad de que el usuario haya visto la película.

`train_dataloader()`

Genera el DataLoader para el entrenamiento del modelo.

Returns:

- `DataLoader`: DataLoader para el entrenamiento del modelo.

`training_step(batch)`

Realiza un paso de entrenamiento en el modelo.

Parameters:

- `batch` (tupla): conjunto de datos de entrenamiento.

Returns:

- `float`: Pérdida en el paso de entrenamiento.

class `RatingsDataset`

Dataset de Pytorch para generar los datos de entrenamiento para el modelo de deep learning. Toma como entrada los ratings de las peliculas y transforma las calificaciones existentes en etiquetas de que el usuario vio la pelicula y genera N ejemplos no existentes para etiquetas de que el usuario no vio la pelicula. Esto servirá para entrenar el modelo de deep learning.

`__init__(ratings, negative_examples_qty, all_movieids)`

Inicializa una instancia del dataset de ratings.

Parameters:

- `ratings` (pd.DataFrame): DataFrame con las interacciones de usuarios.
- `negative_examples_qty` (int): Cantidad de ejemplos negativos a generar.
- `all_movieids` (list): Lista con los IDs de todas las películas.

Returns: None