Recomendación de películas usando modelo Two-Tower

Jorge Velásquez

29 de agosto de 2024

Resumen

Uno de los usos mas comunes y extendidos de los modelos de Machine Learning son los sistemas de recomendación. En este proyecto exploramos una implementacion del modelo Two-Tower para recomendar películas a usuarios en la base de datos MovieLens.

Índice

1. Introducción

En este proyecto, exploramos la arquitectura Two-Tower para sistemas de recuperación a gran escala. El problema de recuperación se define de la siguiente manera: dado un query y un gran conjunto de ítems candidatos, recuperar los k candidatos más relevantes. Los sistemas de recuperación son útiles en muchas aplicaciones del mundo real, como la búsqueda [2] y la recomendación [1].

Un sistema de recomendación completo tendría un segundo modelo de clasificación que toma las salidas de la primera etapa y las ajusta para seleccionar las mejores recomendaciones posibles.

Los datos implicitos son más útiles para esta tarea, por lo que vamos a tratar Movielens como un sistema implícito. Esto significa que cada película que un usuario ha visto es un ejemplo positivo, y cada película que no ha visto es un ejemplo negativo implícito.

Utilizamos las librerías TensorFlow y Keras para construir y entrenar el modelo Two-Tower. La base de datos es MovieLens, un conjunto de datos clásico y ampliamente utilizado en la investigación de sistemas de recomendación. MovieLens contiene una colección de calificaciones asignadas a películas por diferentes usuarios.

2. Metodología

2.1. Descripción del Modelo

El modelo Two-Towers se compone de dos redes neuronales independientes, denominadas "torres", que generan representaciones vectoriales (embeddings) para cada entidad.

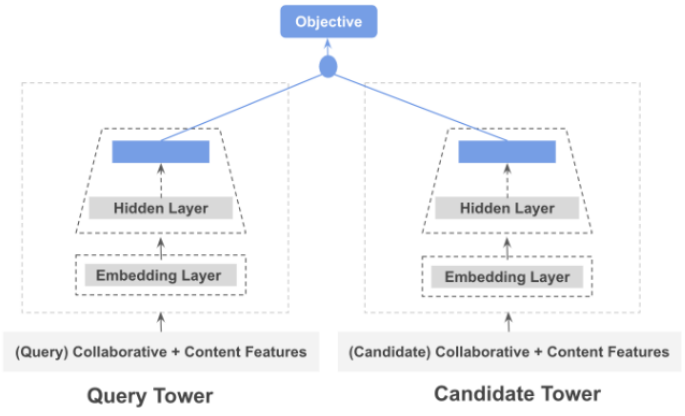

Figura 1: Modelo de dos torres

Dado un par de consulta e ítem representados por vectores de características $x \in X$ y $y \in Y$, respectivamente, la torre izquierda y la torre derecha proporcionan dos funciones de embedding parametrizadas basadas en DNN (Deep Neuronal Network) $u : X \times \mathbb{R}^d \mapsto \mathbb{R}^k, v : Y \times \mathbb{R}^d \mapsto \mathbb{R}^k$, que codifican las características de la consulta y del ítem en un

espacio de embedding de dimensión k. La función de puntuación se calcula luego como el producto punto entre las incrustaciones de la consulta y el ítem en la capa superior, es decir,

$$s(x, y; \theta) = \langle u(x, \theta), v(y, \theta) \rangle$$

.
En este caso las redes neuronales son construidas con ayuda de la librería TensorFlow, en este caso el embedding resultante del output tiene dimensión 32, luego de considerar usuarios y películas únicas, la red neuronal para usuarios tiene 30208 parámetros mientras que la red para películas 53280.

2.2. Preparación de Datos

El conjunto de datos MovieLens fue cargado utilizando la biblioteca TensorFlow Datasets. Se utilizaron dos variantes del conjunto de datos:

- movielens/100k_ratings: Este dataset contiene 100 mil calificaciones que los usuarios han asignado a las películas, junto con información adicional como la marca de tiempo de la calificación
- movielens/100k_movies: Este dataset contiene información sobre 100k películas, incluyendo el ID de la película, el título y los géneros asociados.

Para simplificar el modelo y enfocarnos en las características más relevantes, se realizó un filtrado de las columnas de datos, reteniendo únicamente los campos "user_id" y "movie_title".

Para ajustar y evaluar el modelo, se dividió el conjunto de datos en un conjunto de entrenamiento y otro de evaluación. Utilizamos una partición aleatoria, asignando el 80 % de las calificaciones en el conjunto de entrenamiento y el 20 % en el conjunto de prueba.

2.3. Entrenamiento del Modelo

En nuestros datos de entrenamiento, tenemos pares positivos (usuario, película). Para determinar qué tan bueno es nuestro modelo, necesitamos comparar la puntuación de afinidad que el modelo calcula para este par con las puntuaciones de todos los demás candidatos posibles: si la puntuación para el par positivo es más alta que para todos los otros candidatos, nuestro modelo es preciso. Para esto, podemos usar la métrica tfrs.metrics.FactorizedTopK

Las metricas de recuperacion top-k nos indican si el verdadero positivo se encuentra entre los elementos recuperados en el top-k del conjunto completo de candidatos. Por ejemplo, una métrica de precisión categórica top-5 de 0.2 nos indicaría que, en promedio, el verdadero positivo se encuentra entre los 5 elementos principales recuperados el 20 % del tiempo.

Para la funcion de perdida, utilizamos el objeto Retrieval de la biblioteca TensorFlow Recommenders (TFRS). Esta función de pérdida es una envoltura conveniente que combina la pérdida de entropía cruzada con el cálculo de métricas de recuperación top-k. La

función de pérdida se define como:

$$\mathcal{L}(u, v) = -\log(\sigma(\langle u, v\rangle)) + \sum_{v' \notin S} \log\left(1 - \sigma\left(\langle u, v'\rangle\right)\right)$$

Donde:

- u es el embedding del usuario.
- v es el embedding de la película que el usuario ha visto.
- v' representa los embeddings de las películas que el usuario no ha visto (negativas).
- $\langle u, v\rangle$ denota el producto punto entre los embeddings del usuario y de la película.
- σ es la función sigmoide.
- S es el conjunto de películas que el usuario ha calificado positivamente.

Para entrenar el modelo Adagrad con un learning rate de 0.1, en batches de 8192 por 10 épocas.

3. Resultados

A continuación se muestra la evolucion de la perdida en las épocas.
A medida que el modelo entrena, la pérdida disminuye y se actualiza un conjunto de

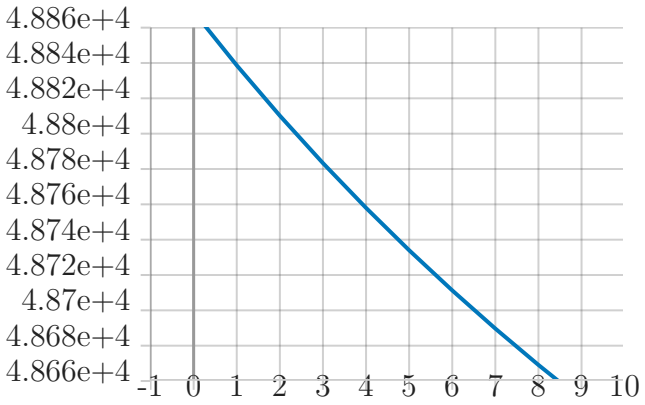

Figura 2: Perdida vs época

métricas de recuperación top-k y como es de esperarse la perdida disminuye rapidamente.

3.1. Evaluación del Modelo

A continuación se muestran las actualizaciones de las métricas de recuperación top-1, top-5 y top-10 a medida que el modelo entrena avanzando en épocas.

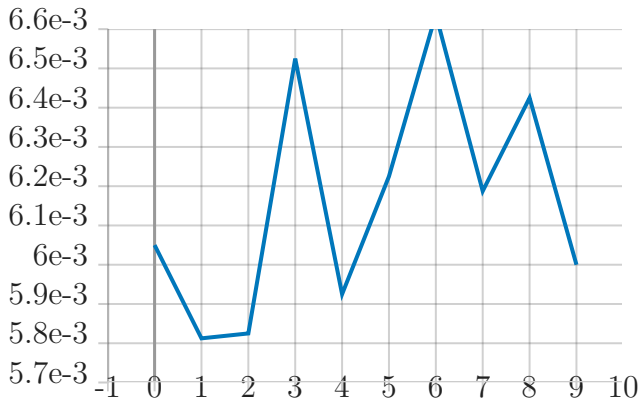

Figura 3: Recuperación top-1

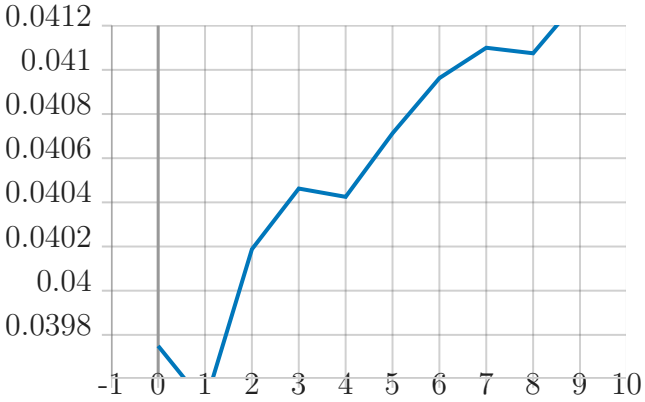

Figura 4: Recuperación top-5

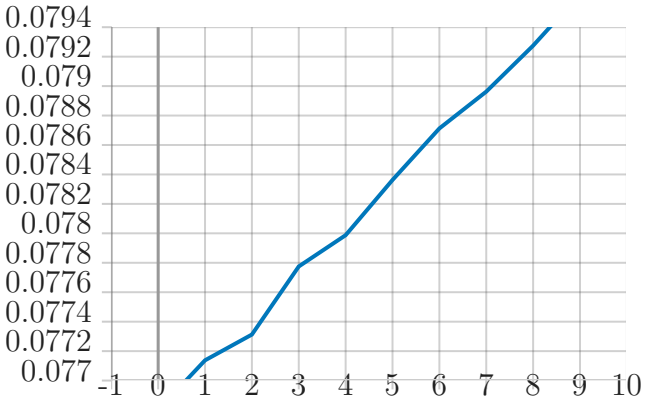

Figura 5: Recuperación top-10

Estos resultados se pueden explicar en la medida en que mientras k es mas grande, el modelo tiene mas oportunidades para incluir la película dentro de las k principales recomendaciones.

3.2. Evaluación

La evaluación se realizo se hizo por batches de 4096 datos, los resultados para las metricas de recuperación top k fueron los siguientes. Volvemos a ver el comportamiento de mejora de rendimiento a medida que aumenta k, sin embargo el rendimiento en el

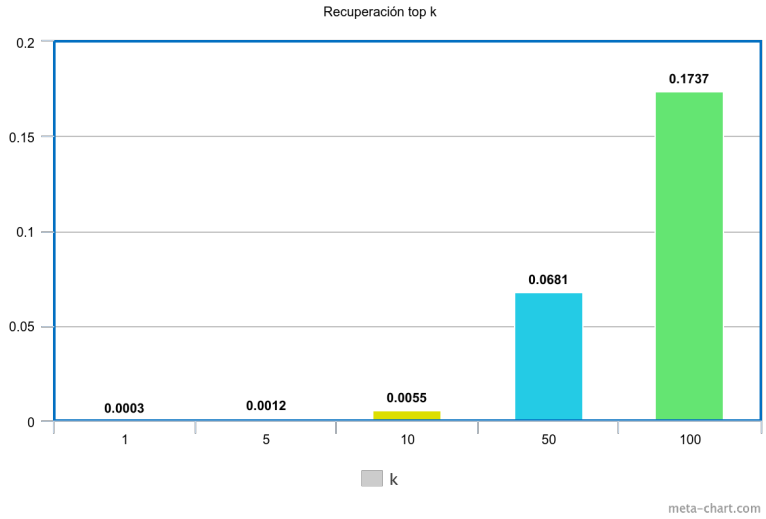

Figura 6: Caption

conjunto de prueba es mucho peor que en el conjunto de entrenamiento. Esto puede deberse a que nuestro modelo funcione mejor con los datos que ha visto porque puede memorizarlos.

4. Conclusiones y Trabajo Futuro

El modelo Two-Tower es una herramienta útil para la tarea de recomendación de items basados en características, en este proyecto realizamos una implementación sencilla que arrojo buenos resultados, una manera de mejorar estos resultados seria incluyendo un segundo modelo al sistema de recomendación que haga un ranking de los items seleccionados por este primer modelo. Estos dos modelos se complementarían de manera eficiente ya que el pobre rendimiento de Two-Tower a la hora de seleccionar pocos items se vería compensado con el modelo de ranking, mientras que el modelo de ranking no tendría que hacer una tarea extensiva de clasificación ya que el modelo de Two-Tower habría hecho un primer filtro de candidatos

Cabe destacar que en este proyecto no se saco todo el provecho a todos los datos existentes en la base de datos, no se tuvo en cuenta el genero, la longitud y otras características de las películas que podrían enriquecer el modelo. Una propuesta interesante es asignarle mas tareas al modelo [3] de esta manear se podria sacar provecho a la naturaleza de distintos datos.

Por ejemplo en nuestro caso, podríamos entrenar el modelo para dos tareas: predecir ratings y predecir vistas. De esta manera la tarea con abundantes datos (cantidad de vista de una película) podría ayudar a mejorar el rendimiento de la tarea con datos escasos (cantidad de ratings) mediante transfer learning.

En conclusión el modelo Two-Tower es un modelo robusto para manejar las complejidades de los sistemas de recomendación en entornos a gran escala. Al gestionar eficazmente el almacenamiento y la recuperación de los vectores de usuarios y elementos, el sistema puede ofrecer recomendaciones oportunas, relevantes y personalizadas.

Referencias

[1] Adams J. Covington, P. and E Sargin. Deep neural networks for youtube recommendations. In *RecSys*, 2016.

[2] He X. Gao J. Deng L. Shen, Y. and G Mesnil. Learn-ing semantic representations using convolutional neural networks for web search. In *WWW*, 2014.

[3] R. Wang. Improving relevance prediction with transfer learning in large-scale retrieval systems. In *ICML Workshop on Adaptive & Multitask Learning: Algorithms & Systems*, 2019.