Recommender Systems Tutorial

Shan-Hung Wu & DataLab
Fall 2023

Overview

In this tutorial, you would design a recommender system that recommends movies to users. When a user queries your system with $(UserID, Timestamp)$, your system should **return a list of 10 movies in their MovieIDs** $(MovieID_1, MovieID_2, \cdots, MovieID_{10})$ which the user might be interested in.

Dataset

Please download the dataset here.

The dataset used in this tutorial is a modified version of the MovieLens Dataset.

The provided dataset consists of three files: `ratings_train.csv`, `users.csv`, and `movies.csv`.

`ratings_train.csv`:

Interactions between users and movies in format $(UserID, MovieID, Rating, Timestamp)$.

- $UserID$: ID of the user.
- $MovieID$: ID of the movie.
- $Rating$: Rating score that the user gives to the movie.
- $Timestamp$: Timestamp of when the user rated the movie.

`users.csv`:

Features of all users in format
$(UserID,$
$Gender,$
$Age,$
$Occupation,$
$ZipCode$
$)$
.

- $UserID$: ID of the user.
- $Gender$: Gender of the user.
- Age: Age interval to which the user belongs. The number represents the starting age of the interval.
- $Occupation$
  : Occupation class number of the user.
- $ZipCode$: ZIP code string of the user.

`movies.csv`:

Features of all movies in format
$(MovieID,$
$Title,$
$Genres)$
.

- $MovieID$
  : ID of the movie.
- $Title$: Title of the movie.
- $Genres$: Genres of the movie. Multiple genres are separated by `|`.

In this tutorial, we provide an example model which implements the simplified version of Funk-SVD mentioned in class.

In [ ]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tqdm import tqdm
```

```
2024-01-05 10:50:19.024112: I tensorflow/core/platform/cpu_feature_guard.cc:182] This Te
nsorFlow binary is optimized to use available CPU instructions in performance-critical o
perations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
```

In [ ]:
```
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        # Currently, memory growth needs to be the same across GPUs
```

```
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        # Select GPU number 1
        tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
        logical_gpus = tf.config.experimental.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
    except RuntimeError as e:
        # Memory growth must be set before GPUs have been initialized
        print(e)
```

Inspect the dataset

Let's load and inspect the three files `ratings_train.csv`, `users.csv`, and `movies.csv` in the dataset.

In [ ]:
```
DATASET_PATH = 'recommender-tutorial-dataset'
USERS_PATH = os.path.join(DATASET_PATH, 'users.csv')
MOVIES_PATH = os.path.join(DATASET_PATH, 'movies.csv')
RATINGS_PATH = os.path.join(DATASET_PATH, 'ratings_train.csv')
```

In [ ]:
```
df_users = pd.read_csv(USERS_PATH)
df_users
```

Out[ ]:

	UserID	Gender	Age	Occupation	ZipCode
0	0	F	1	10	48067
1	1	M	56	16	70072
2	2	M	25	15	55117
3	3	M	45	7	02460
4	4	M	25	20	55455
...	...	...	...	...	...
6035	6035	F	25	15	32603
6036	6036	F	45	1	76006
6037	6037	F	56	1	14706
6038	6038	F	45	0	01060
6039	6039	M	25	6	11106

6040 rows × 5 columns

In [ ]:
```
df_movies = pd.read_csv(MOVIES_PATH)
df_movies
```

Out[ ]:

	MovieID	Title	Genres
0	0	Toy Story (1995)	Animation|Children's|Comedy
1	1	Jumanji (1995)	Adventure|Children's|Fantasy
2	2	Grumpier Old Men (1995)	Comedy|Romance
3	3	Waiting to Exhale (1995)	Comedy|Drama

Out[ ]:

	MovieID	Title	Genres
4	4	Father of the Bride Part II (1995)	Comedy
...	...	...	...
3878	3947	Meet the Parents (2000)	Comedy
3879	3948	Requiem for a Dream (2000)	Drama
3880	3949	Tigerland (2000)	Drama
3881	3950	Two Family House (2000)	Drama
3882	3951	Contender, The (2000)	Drama|Thriller

3883 rows × 3 columns

In [ ]:
```
df_ratings = pd.read_csv(RATINGS_PATH)
df_ratings
```

Out[ ]:

	UserID	MovieID	Rating	Timestamp
0	6039	857	4	956703932
1	6039	2383	4	956703954
2	6039	592	5	956703954
3	6039	1960	4	956703977
4	6039	2018	5	956703977
...	...	...	...	...
939757	5949	1996	3	1046368734
939758	5949	1260	4	1046368750
939759	5949	3151	3	1046368831
939760	5949	3910	4	1046369026
939761	4957	2452	4	1046454260

939762 rows × 4 columns

Let's calculate how many users and movies are in the dataset. There are some $MovieIDs$ that do not correspond to any movie, however, we will just ignore this fact.

In [ ]:
```
M_USERS = max(len(df_users['UserID'].unique()), df_users['UserID'].max() + 1)
N_ITEMS = max(len(df_movies['MovieID'].unique()), df_movies['MovieID'].max() + 1)
print(f'# of users: {M_USERS}, # of movies: {N_ITEMS}')
```

```
# of users: 6040, # of movies: 3952
```

Model

Here we provide an implementation of a simplified version of the Funk-SVD model.

In [ ]:
```
class FunkSVDRecommender(tf.keras.Model):
```

```
    '''
    Simplified Funk-SVD recommender model
    '''

    def __init__(self, m_users: int, n_items: int, embedding_size: int, learning_rate: f
        '''
        Constructor of the model
        '''
        super().__init__()
        self.m = m_users
        self.n = n_items
        self.k = embedding_size
        self.lr = learning_rate

        # user embeddings P
        self.P = tf.Variable(tf.keras.initializers.RandomNormal()(shape=(self.m, self.k)

        # item embeddings Q
        self.Q = tf.Variable(tf.keras.initializers.RandomNormal()(shape=(self.n, self.k)

        # optimizer
        self.optimizer = tf.optimizers.Adam(learning_rate=self.lr)

    @tf.function
    def call(self, user_ids: tf.Tensor, item_ids: tf.Tensor) -> tf.Tensor:
        '''
        Forward pass used in training and validating
        '''
        # dot product the user and item embeddings corresponding to the observed interac
        y_pred = tf.reduce_sum(tf.gather(self.P, indices=user_ids) * tf.gather(self.Q, i

        return y_pred

    @tf.function
    def compute_loss(self, y_true: tf.Tensor, y_pred: tf.Tensor) -> tf.Tensor:
        '''
        Compute the MSE loss of the model
        '''
        loss = tf.losses.mean_squared_error(y_true, y_pred)

        return loss

    @tf.function
    def train_step(self, data: tf.Tensor) -> tf.Tensor:
        '''
        Train the model with one batch
        data: batched user-item interactions
        each record in data is in the format [UserID, MovieID, Rating, Timestamp]
        '''
        user_ids = tf.cast(data[:, 0], dtype=tf.int32)
        item_ids = tf.cast(data[:, 1], dtype=tf.int32)
        y_true = tf.cast(data[:, 2], dtype=tf.float32)

        # compute loss
        with tf.GradientTape() as tape:
            y_pred = self(user_ids, item_ids)
            loss = self.compute_loss(y_true, y_pred)

        # compute gradients
        gradients = tape.gradient(loss, self.trainable_variables)

        # update weights
        self.optimizer.apply_gradients(zip(gradients, self.trainable_variables))
```

```
        return loss

    @tf.function
    def val_step(self, data: tf.Tensor) -> tf.Tensor:
        '''
        Validate the model with one batch
        data: batched user-item interactions
        each record in data is in the format [UserID, MovieID, Rating, Timestamp]
        '''
        user_ids = tf.cast(data[:, 0], dtype=tf.int32)
        item_ids = tf.cast(data[:, 1], dtype=tf.int32)
        y_true = tf.cast(data[:, 2], dtype=tf.float32)

        # compute loss
        y_pred = self(user_ids, item_ids)
        loss = self.compute_loss(y_true, y_pred)

        return loss

    @tf.function
    def eval_predict_onestep(self, query: tf.Tensor) -> tf.Tensor:
        '''
        Retrieve and return the MovieIDs of the 10 recommended movies given a query
        You should return a tf.Tensor with shape=(10,)
        query will be a tf.Tensor with shape=(2,) and dtype=tf.int64
        query[0] is the UserID of the query
        query[1] is the Timestamp of the query
        '''
        # dot product the selected user and all item embeddings to produce predictions
        user_id = tf.cast(query[0], tf.int32)
        y_pred = tf.reduce_sum(tf.gather(self.P, user_id) * self.Q, axis=1)

        # select the top 10 items with highest scores in y_pred
        y_top_10 = tf.math.top_k(y_pred, k=10).indices

        return y_top_10
```

Split datasets

First, we create per-user validation sets with the latest interactions of each user (should also contain sufficient positive interactions). Then, we join all the small validation sets to form a complete validation set.

In [ ]:
```
# let interactions with rating >= 4 be positive interactions
POSITIVE_THRESHOLD = 4

# each per-user validation set should contain at least 5 positive interactions
POSITIVE_PER_USER = 5

train_dataframes = []
val_dataframes = []

for i in tqdm(range(M_USERS)):
    user_all = df_ratings[df_ratings['UserID'] == i]
    user_positive = user_all[user_all['Rating'] >= POSITIVE_THRESHOLD]

    # check if there are enough positive interactions to build a validation set for this
    if len(user_positive) >= POSITIVE_PER_USER:
        split_idx = user_positive.iloc[-POSITIVE_PER_USER].name
```

```
        user_train = user_all.loc[:split_idx]
        user_test = user_all.loc[split_idx:]
        assert user_train['Timestamp'].max() <= user_test['Timestamp'].min()
        train_dataframes.append(user_train)
        val_dataframes.append(user_test)
    else:
        train_dataframes.append(user_all)

# concat all per-user training sets
df_train = pd.concat(train_dataframes).sort_values(by='Timestamp', ascending=True, ignor

# normalize the ratings (may be beneficial to some models)
df_train_norm = df_train
df_train_norm['Rating'] -= 3
df_train_norm['Rating'] /= 2

# concat all per-user validation sets
df_val = pd.concat(val_dataframes).sort_values(by='Timestamp', ascending=True, ignore_in

# normalize the ratings (may be beneficial to some models)
# here we make a copy of the un-normalized validation set for evaluation
df_val_norm = df_val.copy(deep=True)
df_val_norm['Rating'] -= 3
df_val_norm['Rating'] /= 2
```

```
100%|██████████| 6040/6040 [00:12<00:00, 470.58it/s]
```

In [ ]:
```
df_train_norm
```

Out[ ]:

	UserID	MovieID	Rating	Timestamp
0	6039	857	0.5	956703932
1	6039	592	1.0	956703954
2	6039	2383	0.5	956703954
3	6039	2018	1.0	956703977
4	6039	1960	0.5	956703977
...	...	...	...	...
893652	5949	2857	0.0	1046368290
893653	5949	2663	0.0	1046368398
893654	5949	1299	0.5	1046368398
893655	5949	1195	0.5	1046368417
893656	5949	1357	0.5	1046368429

893657 rows × 4 columns

In [ ]:
```
df_val_norm
```

Out[ ]:

	UserID	MovieID	Rating	Timestamp
0	6038	2018	0.5	956706538
1	6038	1251	0.5	956706538
2	6038	922	0.5	956706538

Out[ ]:

	UserID	MovieID	Rating	Timestamp
3	6037	2715	0.0	956707604
4	6037	3547	0.5	956707604
...	...	...	...	...
52084	5949	1996	0.0	1046368734
52085	5949	1260	0.5	1046368750
52086	5949	3151	0.0	1046368831
52087	5949	3910	0.5	1046369026
52088	4957	2452	0.5	1046454260

52089 rows × 4 columns

Evaluation metric

For the evaluation metric, we will use a modified version of
$NDCG$
@10
specifically for this task.

$NDCG@$

$Recall$
@10
(same as
$HitRate$
@10
in this task) will also be calculated.

In [ ]:

```
@tf.function
def log2(x: tf.Tensor) -> tf.Tensor:
    return tf.math.log(tf.cast(x, tf.float32)) / tf.math.log(2.)

@tf.function
def ndcg_at_10(y_true: tf.Tensor, y_pred: tf.Tensor) -> tf.Tensor:
    y_pred = y_pred[:10]
    idx = tf.equal(tf.cast(y_pred, tf.int32), tf.cast(y_true, tf.int32))
    if tf.reduce_sum(tf.cast(idx, tf.int32)) > 0:
        return 1. / log2(2 + tf.argmax(idx))
    else:
        return tf.constant(0.)

@tf.function
def recall_at_10(y_true: tf.Tensor, y_pred: tf.Tensor) -> tf.Tensor:
    y_pred = y_pred[:10]
    idx = tf.equal(tf.cast(y_pred, tf.int32), tf.cast(y_true, tf.int32))
    if tf.reduce_sum(tf.cast(idx, tf.int32)) > 0:
        return tf.constant(1.)
    else:
        return tf.constant(0.)
```

```
def evaluate(model: tf.keras.Model, dataset: tf.data.Dataset) -> tuple:
    '''
    For each data point in the dataset:
    data[0] is the UserID
    data[1] is the MovieID
    data[2] is the Rating
    data[3] is the Timestamp
    '''
    ndcg_scores = []
    recall_scores = []

    for data in tqdm(dataset, desc='Evaluating'):
        # query the model to make predictions if the observed event is a positive intera
        if data[2] >= 4:
            y_pred = model.eval_predict_onestep(tf.gather(data, (0, 3)))
            y_true = tf.gather(data, 1)
            ndcg = ndcg_at_10(y_true, y_pred)
            recall = recall_at_10(y_true, y_pred)
            ndcg_scores.append(ndcg)
            recall_scores.append(recall)

    ndcg_result = tf.reduce_mean(ndcg_scores).numpy()
    recall_result = tf.reduce_mean(recall_scores).numpy()

    return ndcg_result, recall_result
```

Train the model

In [ ]:
```
# hyperparameters
EMBEDDING_SIZE = 256
BATCH_SIZE = 512
N_EPOCHS = 25
LEARNING_RATE = 1e-4
```

In [ ]:
```
# prepare datasets
dataset_train = tf.data.Dataset.from_tensor_slices(df_train_norm)
dataset_train = dataset_train.batch(batch_size=BATCH_SIZE, num_parallel_calls=tf.data.AU

dataset_val = tf.data.Dataset.from_tensor_slices(df_val_norm)
dataset_val = dataset_val.batch(batch_size=BATCH_SIZE, num_parallel_calls=tf.data.AUTOTU

# build the model
model = FunkSVDRecommender(m_users=M_USERS, n_items=N_ITEMS, embedding_size=EMBEDDING_SI

# train the model
train_losses = []
val_losses = []

for epoch in range(1, N_EPOCHS + 1):
    train_loss = []
    val_loss = []
    print(f'Epoch {epoch}:')

    # training
    for data in tqdm(dataset_train, desc='Training'):
        loss = model.train_step(data)
        train_loss.append(loss.numpy())

    # validating
    for data in tqdm(dataset_val, desc='Validating'):
```

```
        loss = model.val_step(data)
        val_loss.append(loss.numpy())

    # record losses
    avg_train_loss = np.mean(train_loss)
    avg_val_loss = np.mean(val_loss)
    train_losses.append(avg_train_loss)
    val_losses.append(avg_val_loss)

    # print losses
    print(f'Epoch {epoch} train_loss: {avg_train_loss:.4f}, val_loss: {avg_val_loss:.4f}')

# plot the training curve
plt.plot(train_losses, label='train_loss')
plt.plot(val_losses, label='val_loss')
plt.legend(loc='upper right')
plt.title('Loss curve')
plt.show()
```

```
Epoch 1:
Training: 100%|██████████| 1746/1746 [00:28<00:00, 60.86it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 369.05it/s]
Epoch 1 train_loss: 0.4015, val_loss: 0.3957

Epoch 2:
Training: 100%|██████████| 1746/1746 [00:25<00:00, 67.68it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 603.00it/s]
Epoch 2 train_loss: 0.3950, val_loss: 0.3930

Epoch 3:
Training: 100%|██████████| 1746/1746 [00:28<00:00, 60.50it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 868.96it/s]
Epoch 3 train_loss: 0.3834, val_loss: 0.3852

Epoch 4:
Training: 100%|██████████| 1746/1746 [00:26<00:00, 65.41it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 834.96it/s]
Epoch 4 train_loss: 0.3599, val_loss: 0.3665

Epoch 5:
Training: 100%|██████████| 1746/1746 [00:25<00:00, 68.71it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 793.52it/s]
Epoch 5 train_loss: 0.3198, val_loss: 0.3361

Epoch 6:
Training: 100%|██████████| 1746/1746 [00:24<00:00, 70.22it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 813.06it/s]
Epoch 6 train_loss: 0.2733, val_loss: 0.3039

Epoch 7:
Training: 100%|██████████| 1746/1746 [00:24<00:00, 70.51it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 832.49it/s]
Epoch 7 train_loss: 0.2374, val_loss: 0.2794

Epoch 8:
Training: 100%|██████████| 1746/1746 [00:24<00:00, 70.37it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 805.53it/s]
Epoch 8 train_loss: 0.2145, val_loss: 0.2626

Epoch 9:
Training: 100%|██████████| 1746/1746 [00:24<00:00, 70.55it/s]
```

```
Validating: 100%|██████████| 102/102 [00:00<00:00, 807.10it/s]
Epoch 9 train_loss: 0.1990, val_loss: 0.2510

Epoch 10:
Training: 100%|██████████| 1746/1746 [00:40<00:00, 42.62it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 853.36it/s]
Epoch 10 train_loss: 0.1870, val_loss: 0.2426

Epoch 11:
Training: 100%|██████████| 1746/1746 [00:24<00:00, 71.15it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 624.12it/s]
Epoch 11 train_loss: 0.1768, val_loss: 0.2361

Epoch 12:
Training: 100%|██████████| 1746/1746 [00:24<00:00, 70.70it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 755.15it/s]
Epoch 12 train_loss: 0.1677, val_loss: 0.2309

Epoch 13:
Training: 100%|██████████| 1746/1746 [00:24<00:00, 70.05it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 812.65it/s]
Epoch 13 train_loss: 0.1593, val_loss: 0.2267

Epoch 14:
Training: 100%|██████████| 1746/1746 [00:25<00:00, 69.64it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 788.51it/s]
Epoch 14 train_loss: 0.1514, val_loss: 0.2232

Epoch 15:
Training: 100%|██████████| 1746/1746 [00:25<00:00, 69.52it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 717.13it/s]
Epoch 15 train_loss: 0.1440, val_loss: 0.2202

Epoch 16:
Training: 100%|██████████| 1746/1746 [00:24<00:00, 70.40it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 825.62it/s]
Epoch 16 train_loss: 0.1369, val_loss: 0.2177

Epoch 17:
Training: 100%|██████████| 1746/1746 [00:24<00:00, 69.98it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 836.00it/s]
Epoch 17 train_loss: 0.1300, val_loss: 0.2156

Epoch 18:
Training: 100%|██████████| 1746/1746 [00:24<00:00, 70.25it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 828.14it/s]
Epoch 18 train_loss: 0.1234, val_loss: 0.2138

Epoch 19:
Training: 100%|██████████| 1746/1746 [00:24<00:00, 71.41it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 773.91it/s]
Epoch 19 train_loss: 0.1170, val_loss: 0.2124

Epoch 20:
Training: 100%|██████████| 1746/1746 [00:24<00:00, 71.06it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 586.83it/s]
Epoch 20 train_loss: 0.1109, val_loss: 0.2113

Epoch 21:
```

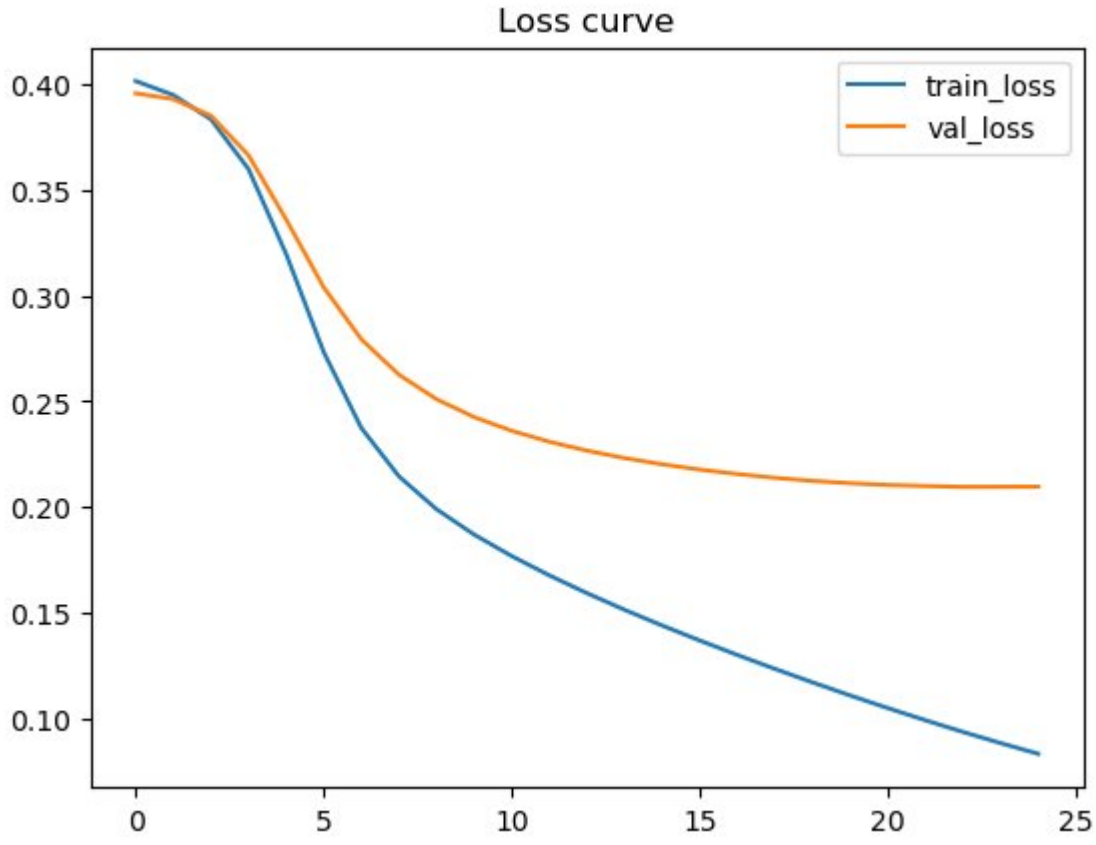
```
Training: 100%|██████████| 1746/1746 [00:24<00:00, 70.84it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 819.87it/s]
Epoch 21 train_loss: 0.1049, val_loss: 0.2104

Epoch 22:
Training: 100%|██████████| 1746/1746 [00:24<00:00, 71.00it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 819.55it/s]
Epoch 22 train_loss: 0.0991, val_loss: 0.2100

Epoch 23:
Training: 100%|██████████| 1746/1746 [00:24<00:00, 70.70it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 846.89it/s]
Epoch 23 train_loss: 0.0936, val_loss: 0.2096

Epoch 24:
Training: 100%|██████████| 1746/1746 [00:40<00:00, 42.61it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 759.77it/s]
Epoch 24 train_loss: 0.0883, val_loss: 0.2096

Epoch 25:
Training: 100%|██████████| 1746/1746 [00:24<00:00, 70.53it/s]
Validating: 100%|██████████| 102/102 [00:00<00:00, 780.51it/s]
Epoch 25 train_loss: 0.0833, val_loss: 0.2097
```



Evaluate the model with the validation set

In [ ]:
```
dataset_eval = tf.data.Dataset.from_tensor_slices(df_val)
dataset_eval = dataset_eval.prefetch(buffer_size=tf.data.AUTOTUNE)
ndcg_result, recall_result = evaluate(model, dataset_eval)
print(f'Evaluation result: [NDCG@10: {ndcg_result:.6f}, Recall@10: {recall_result:.6f}]'
```

```
Evaluating: 100%|██████████| 52089/52089 [02:08<00:00, 404.95it/s]
```

```
Evaluation result: [NDCG@10: 0.021742, Recall@10: 0.042413]
```