DataLab Cup 4: Recommender Systems

Shan-Hung Wu & DataLab
Fall 2023

Platform: Kaggle

Please download the dataset and the environment source code from Kaggle.

Overview

In this competition, your goal is to design a recommender system that suggests news articles to users. The performance of your recommender system will be assessed using a simulation environment.

At each timestep, the simulation environment randomly selects an active user with a given `user_id`. Once you receive this `user_id`, your recommender system must generate a slate **(a list of 5 distinct `item_ids` to recommend to the current user)** and pass it to the environment. The environment then uses its internal information to determine which item the user will choose from the recommended list (with some degree of stochasticity) or decide not to choose any item due to a lack of interest.

Each user has a latent patience value (invisible to your recommender system), which slightly increases when an item is chosen and drastically decreases when no item is chosen in each round. If a user's patience drops below 0 or the user runs out of the time budget (2000 timesteps), the user leaves the environment. The chosen `item_id` (or `-1` if no item is chosen) and whether the current user stays (`True`) or leaves (`False`) are returned as the result of recommending a slate of items. A new user (if any) will be randomly selected for recommendations in the next timestep after the response of the current user is generated.

Your recommender system should continue recommending items to the current user at each timestep as long as there are still active users in the environment. The simulation process terminates after all users have left the system.

Your goal is to maximize the session length of each user. The session length is defined as the number of timesteps a user interacts with your recommender system before leaving the environment. The calculated session length score, normalized to the range of 0 ~ 1, will be provided by the simulation environment after the completion of the simulation process.

In [ ]:
```
import os
import random

import numpy as np
```

```
import pandas as pd
from tqdm import tqdm

from evaluation.environment import TrainingEnvironment, TestingEnvironment
```

In [ ]:
```
# Official hyperparameters for this competition (do not modify)
N_TRAIN_USERS = 1000
N_TEST_USERS = 2000
N_ITEMS = 209527
HORIZON = 2000
TEST_EPISODES = 5
SLATE_SIZE = 5
```

Datasets

In this competition, we won't provide a substantial user-item interaction dataset. Instead, limited information (3 items per user) on historical interactions will be available. To train your recommender system effectively, you need to employ a recommender policy to interact with the training environment and collect additional interaction data.

We will introduce the side-information datasets provided in the following sections.

In [ ]:
```
# Dataset paths
USER_DATA = os.path.join('dataset', 'user_data.json')
ITEM_DATA = os.path.join('dataset', 'item_data.json')

# Output file path
OUTPUT_PATH = os.path.join('output', 'output.csv')
```

User Data

In the **training environment**, there are a total of **1000 users** identified by IDs ranging from 0 to 999. For the **testing environment**, there are **2000 users** with IDs ranging from 0 to 1999. The **testing environment includes the same 1000 users found in the training environment** (user 0 to user 999), and an additional 1000 new users (user 1000 to user 1999) are introduced.

For all 2000 users, we provide you with the **past 3 clicked item IDs of each user**. Let's examine the user dataset.

In [ ]:
```
df_user = pd.read_json(USER_DATA, lines=True)
df_user
```

Out[ ]:

	user_id	history
0	0	[42558, 65272, 13353]
1	1	[146057, 195688, 143652]
2	2	[67551, 85247, 33714]
3	3	[116097, 192703, 103229]
4	4	[68756, 140123, 135289]
...	...	...
1995	1995	[95090, 131393, 130239]
1996	1996	[2360, 147130, 8145]
1997	1997	[99794, 138694, 157888]
1998	1998	[55561, 60372, 51442]
1999	1999	[125409, 77906, 124792]

2000 rows × 2 columns

Item Data

Both the training and testing environments share a common pool of **209527 items** as their item candidate pool. For the side information of these items, we provide text descriptions for each news article. The item dataset is derived from the News Category Dataset. It's important to note that you should only use the dataset provided by us. Utilizing the original dataset, which contains extra information, will be considered as cheating. Let's explore the item dataset.

In [ ]:
```
df_item = pd.read_json(ITEM_DATA, lines=True)
df_item
```

Out[ ]:

	item_id	headline	short_description
0	0	Over 4 Million Americans Roll Up Sleeves For O...	Health experts said it is too early to predict...
1	1	American Airlines Flyer Charged, Banned For Li...	He was subdued by passengers and crew when he ...
2	2	23 Of The Funniest Tweets About Cats And Dogs ...	"Until you have a dog you don't understand wha...
3	3	The Funniest Tweets From Parents This Week (Se...	"Accidentally put grown-up toothpaste on my to...
4	4	Woman Who Called Cops On Black Bird-Watcher Lo...	Amy Cooper accused investment firm Franklin Te...
...	...	...	...
209522	209522	RIM CEO Thorsten Heins' 'Significant' Plans Fo...	Verizon Wireless and AT&T are already promotin...
209523	209523	Maria Sharapova Stunned By Victoria Azarenka I...	Afterward, Azarenka, more effusive with the pr...
209524	209524	Giants Over Patriots, Jets Over Colts Among M...	Leading up to Super Bowl XLVI, the most talked...
209525	209525	Aldon Smith Arrested: 49ers Linebacker Busted ...	CORRECTION: An earlier version of this story i...
209526	209526	Dwight Howard Rips Teammates After Magic Loss ...	The five-time all-star center tore into his te...

209527 rows × 3 columns

Simulation Environments

We offer two simulation environments in this competition: `TrainingEnvironment` and `TestingEnvironment`. The only distinction between the two environments is the number of users, with 1000 for training and 2000 for testing. All public methods for both environments behave the same since they share the same base class.

Important Note: Ensure that you collect interaction data only by accessing the environment through the designated public methods listed below. Directly accessing or modifying any file or code in the `evaluation` directory, or retrieving internal attributes and states of the environment (including all attributes / methods starting with an underscore `_`), will be considered as cheating.

Environment Classes

`class` `TrainingEnvironment`

Class for the training environment. Contains first 1000 users with user ID ranging from 0 to 999.

`class` `TestingEnvironment`

Class for the testing environment. Contains all 2000 users with user ID ranging from 0 to 1999.

Environment Public Methods

Note that both `TrainingEnvironment` and `TestingEnvironment` shares the same set of public methods.

`function` `reset`

```
reset() → None
```

Reset the environment to its initial parameters and states.

`function` `has_next_state`

```
has_next_state() → bool
```

Verify whether the next state exists. The next state is considered to exist if there is at least one user still present in the environment.

Returns:

- `True` if the next state exists, `False` otherwise.

`function` `get_state`

```
get_state() → int
```

Get the current state (the user ID of the current user).

Returns:

- **`int`** : The user ID of the current user, or `-1` if there are no active users in the environment.

`function` `get_response`

```
get_response(slate: list) → tuple[int, bool]
```

Send the recommended slate (list of 5 distinct item IDs) and get the response from

the current user. The internal user state will be updated according to the response, and a random user will be selected to be the next user (next state).

Args:

- `slate` : `list[int]` A list of 5 distinct item IDs to be recommended.

Returns:

- `tuple[int, bool]` : The first entry indicates the `item ID` chosen by the user, or `-1` if the user decides not to choose any item. The second entry represents whether the user is still in the environment after this interaction round. `True` if the user stays, `False` if the user leaves.

Raises:

- `AssertionError` : If the slate length is not 5, contains duplicates or out-of-range item IDs, or if there are no active users in the environment.

`function` get_score

```
get_score() → list[float]
```

Get the normalized session length score (0 ~ 1) for each user.

Returns:

- `list[float]` : A list containing the normalized session length score for each user.

Training

The implementation of the recommender algorithm is left to you. If you're in need of ideas, you can refer to the Recommender Systems Tutorial notebook in Lecture 16. Here, we'll just provide some example use cases of the public methods.

Hint: If you're looking for inspiration, consider starting by collecting interaction data from the environment using your initial recommender policy. Afterward, improve your model with this data, and iterate through this collect-then-train loop.

Important Note: Ensure that you save your model weights after training. You will need to load a set of model weights trained exclusively on the training environment at the beginning of each test episode.

In [ ]:
```
# Initialize the training environment
train_env = TrainingEnvironment()

# Reset the training environment (this can be useful when you have finished one
train_env.reset()
```

```
# Check if there exist any active users in the environment
env_has_next_state = train_env.has_next_state()
print(f'There is {"still some" if env_has_next_state else "no"} active users in

# Get the current user ID
user_id = train_env.get_state()
print(f'The current user is user {user_id}.')

# Get the response of recommending the slate to the current user
slate = [0, 1, 2, 3, 4]
clicked_id, in_environment = train_env.get_response(slate)
print(f'The click result of recommending {slate} to user {user_id} is {f"item {c
print(f'User {user_id} {"is still in" if in_environment else "leaves"} the envir

# Get the normalized session length score of all users
train_score = train_env.get_score()
df_train_score = pd.DataFrame([[user_id, score] for user_id, score in enumerate(
df_train_score
```

```
There is still some active users in the training environment.
The current user is user 412.
The click result of recommending [0, 1, 2, 3, 4] to user 412 is -1 (no click).
User 412 is still in the environment.
```

Out[ ]:

	user_id	avg_score
0	0	0.0
1	1	0.0
2	2	0.0
3	3	0.0
4	4	0.0
...	...	...
995	995	0.0
996	996	0.0
997	997	0.0
998	998	0.0
999	999	0.0

1000 rows × 2 columns

Testing

While testing, you are allowed to update your model. However, please adhere to the following rules:

1. Follow the testing template provided below. Modify only the sections marked as `[TODO]`. Additionally, please carefully follow the instructions specified in each `[TODO]` section. Modifying other sections or not adhering to the instructions is strictly forbidden.

2. Limit model updates to one testing episode. During testing-time updates, follow these steps: (a) **Load your model weights** trained exclusively on the training environment. (b) Run the testing environment and update your model with the collected data **during the testing process**. (c) Obtain the score for this testing episode and **delete your model weights since they now contain some testing information. You should not save the model weights trained on the testing environment for another testing episode. Doing so will be regarded as cheating.**

3. Due to the randomness in the user decision process, **run the testing process 5 times** and calculate the **average session length** for each user as the final score. This part has been covered for you.

After completing the testing process, remember to submit the generated `output.csv` file to the Kaggle competition.

We will illustrate the testing process with a pure random recommender below.

In [ ]:
```python
# Initialize the testing environment
test_env = TestingEnvironment()
scores = []

# The item_ids here is for the random recommender
item_ids = [i for i in range(N_ITEMS)]

# Repeat the testing process for 5 times
for _ in range(TEST_EPISODES):
    # [TODO] Load your model weights here (in the beginning of each testing epis
    # [TODO] Code for loading your model weights...

    # Start the testing process
    with tqdm(desc='Testing') as pbar:
        # Run as long as there exist some active users
        while test_env.has_next_state():
            # Get the current user id
            cur_user = test_env.get_state()

            # [TODO] Employ your recommendation policy to generate a slate of 5
            # [TODO] Code for generating the recommended slate...
            # Here we provide a simple random implementation
            slate = random.sample(item_ids, k=SLATE_SIZE)

            # Get the response of the slate from the environment
            clicked_id, in_environment = test_env.get_response(slate)

            # [TODO] Update your model here (optional)
            # [TODO] You can update your model at each step, or perform a batche
            # [TODO] Code for updating your model...

            # Update the progress indicator
            pbar.update(1)

    # Record the score of this testing episode
    scores.append(test_env.get_score())

    # Reset the testing environment
```

```
    test_env.reset()

    # [TODO] Delete or reset your model weights here (in the end of each testing
    # [TODO] Code for deleting your model weights...

# Calculate the average scores
avg_scores = [np.average(score) for score in zip(*scores)]

# Generate a DataFrame to output the result in a .csv file
df_result = pd.DataFrame([[user_id, avg_score] for user_id, avg_score in enumera
df_result.to_csv(OUTPUT_PATH, index=False)
df_result
```

```
Testing: 10233it [00:19, 526.52it/s]
Testing: 10293it [00:19, 541.22it/s]
Testing: 10285it [00:19, 531.14it/s]
Testing: 10244it [00:19, 533.77it/s]
Testing: 10224it [00:19, 533.37it/s]
```

Out[ ]:

	user_id	avg_score
0	0	0.0025
1	1	0.0027
2	2	0.0025
3	3	0.0025
4	4	0.0025
...	...	...
1995	1995	0.0027
1996	1996	0.0025
1997	1997	0.0025
1998	1998	0.0025
1999	1999	0.0025

2000 rows × 2 columns

Scoring

- Ranking of **private** leaderboard of the Kaggle competition. (80%)
- Report. (20%)

How is the Score For Ranking Calculated:

We will calculate the MAE (Mean Absolute Error) between your submitted `output.csv` and a "ground-truth" of all 1s. The lower the better.

Your Report Should Contain:

- Models you have tried during the competition. Briefly describe the main idea of the model and the reason why you chose that model.

- List the experiments you have done. For instance, data collecting, utilizing the user / item datasets, hyperparameters tuning, training process, and so on.
- Discussions, lessons learned, or anything else worth mentioning.
- **Ensure your report notebook contains your training and testing code. We will re-run your code if we find your score on Kaggle suspicious.**

Please name your report as `DL_comp4_{Your Team name}_report.ipynb.` and submit your report to the eeclass system before the deadline.

What You Can Do

- Implement any recommender models.
- Collect data through accessing the **public methods provided by the environments** (i.e. methods listed in the ***Environment Public Methods*** section) and train your model.
- Use the provided user history data (`dataset/user_data.json`) and item text description data (`dataset/item_data.json`) as auxiliary data to aid your model training.
- Update the model during one testing episode while **following the rules mentioned in the *Testing* section.**
- You can use a pretrained text encoder if you need text embeddings for the item text descriptions. **(This is the only part you can use a pretrained model in this competition.)**

What You CAN NOT Do

- Use any dataset other than the provided ones. Using the original News Category Dataset is also prohibited.
- Use any pretrained recommender models.
- Plagiarize other teams' work.
- Hack our simulation environments. Any attempt of accessing or modifying the data files in the `evaluation` directory, modifying the source code of the environments, accessing or modifying the private attributes and methods (i.e. methods and attributes not listed in the ***Environment Public Methods*** section), not following the rules in the ***Testing*** section, or any other forbidden actions mentioned in the previous section of the notebook will be regarded as cheating.

Competition Timeline

- 2024/01/08 (Mon): Competition launched.
- 2024/01/15 (Mon) 08:00 (TW): Competition deadline.
- 2024/01/16 (Tue) 12:00 (TW): Report deadline.
- 2024/01/16 (Tue) 15:30 (TW): Top-3 teams sharing.

References

1. Misra, Rishabh. "News Category Dataset." arXiv preprint arXiv:2209.11429 (2022).
2. Misra, Rishabh and Jigyasa Grover. "Sculpting Data for ML: The first act of Machine Learning." ISBN 9798585463570 (2021).