data train val test를 나눌때 주의

증상 : 만약 학습을 시키고 점수가 모두 nan이 나온다면

길이 2이하인 데이터를 지우는 클렌징을 나누고나서 진행해줘야한다

아니면 2이하가 안생기게 나눠주는 방법을 고민해보자

In [1]:
```
import subprocess

def get_gpu_info():
    try:
        result = subprocess.check_output(['nvidia-smi'], encoding='ut
        print(result)
    except Exception as e:
        print("GPU 정보를 가져오는 데 실패했습니다:", e)

get_gpu_info()
```

```
Wed Jul 16 03:20:06 2025
+-----------------------------------------------------------------------
----------------------+
| NVIDIA-SMI 570.124.06             Driver Version: 570.124.06     C
UDA Version: 12.8     |
|-----------------------------------------+------------------------+
----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A |
Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage |
GPU-Util  Compute M. |
|                                         |                        |
MIG M. |
|=========================================+========================+
======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |
0 |
| N/A   45C    P8             12W /   70W |       1MiB /  15360MiB |
0%      Default |
|                                         |                        |
N/A |
+-----------------------------------------+------------------------+
----------------------+

+-----------------------------------------------------------------------
----------------------+
| Processes:
|
|  GPU   GI   CI              PID   Type   Process name
GPU Memory |
|        ID   ID
Usage      |
|=======================================================================
======================|
|  No running processes found
|
+-----------------------------------------------------------------------
----------------------+
```

In [2]:
```
import psutil


def get_ram_info():
    mem = psutil.virtual_memory()
    total = mem.total / (1024**3)      # GB 단위로 환산
    available = mem.available / (1024**3)
    used = (mem.total - mem.available) / (1024**3)
    percent = mem.percent

    print(f"총 메모리: {total:.2f} GB")
    print(f"사용 중인 메모리: {used:.2f} GB")
    print(f"사용 가능한 메모리: {available:.2f} GB")
    print(f"사용률: {percent}%")

get_ram_info()
```

```
총 메모리: 17.56 GB
사용 중인 메모리: 1.03 GB
사용 가능한 메모리: 16.53 GB
사용률: 5.9%
```

In [3]:
```
import pandas
import tensorflow

print(pandas.__version__)
print(tensorflow.__version__)
```

```
1.3.3
2.6.0
```

In [4]:
```
import datetime as dt
from pathlib import Path
import os
import time
from datetime import datetime
from IPython.display import display

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [5]:
```
data_path = Path(os.getenv('HOME')+'/aiffel/yoochoose/data/')
train_path = data_path / 'ratings.dat'

def load_data(data_path: Path, nrows=None):
    data = pd.read_csv(data_path, sep='::', header=None, usecols=[0, 1
    data.columns = ['UserId', 'ItemId', 'Rating', 'Time']
    # time 형식을 기존과 맞게 고쳐보기
    data['Time'] = pd.to_datetime(data['Time'], unit='s')
    return data

data = load_data(train_path, None)
data.sort_values(['UserId', 'Time'], inplace=True)  # data를 id와 시간 순
data
```

Out[5]:

	UserId	ItemId	Rating	Time
31	1	3186	4	2000-12-31 22:00:19
22	1	1270	5	2000-12-31 22:00:55
27	1	1721	4	2000-12-31 22:00:55
37	1	1022	5	2000-12-31 22:00:55
24	1	2340	3	2000-12-31 22:01:43
...	...	...	...	...
1000019	6040	2917	4	2001-08-10 14:40:29
999988	6040	1921	4	2001-08-10 14:41:04
1000172	6040	1784	3	2001-08-10 14:41:04
1000167	6040	161	3	2001-08-10 14:41:26
1000042	6040	1221	4	2001-08-20 13:44:15

1000209 rows × 4 columns

In [6]:
```
#데이터 정제
```

In [7]:
```
# 이부분을 이 데이터에 맞게 고치기

# short_session을 제거한 다음 unpopular item을 제거하면 다시 길이가 1인 session
# 이를 위해 반복문을 통해 지속적으로 제거 합니다.
def cleanse_recursive(data: pd.DataFrame, shortest, least_click) -> pd
    while True:
        before_len = len(data)
        data = cleanse_short_session(data, shortest)
        data = cleanse_unpopular_item(data, least_click)
        data = cleanse_short_session(data, shortest) # 재확인
        after_len = len(data)
        if before_len == after_len:
            break
    return data


def cleanse_short_session(data: pd.DataFrame, shortest):
    session_len = data.groupby('UserId').size()
    session_use = session_len[session_len >= shortest].index
    data = data[data['UserId'].isin(session_use)]
    return data


def cleanse_unpopular_item(data: pd.DataFrame, least_click):
    item_popular = data.groupby('ItemId').size()
    item_use = item_popular[item_popular >= least_click].index
    data = data[data['ItemId'].isin(item_use)]
    return data
```

In [8]:
```
# 전체 데이터 세트에 대해 추천 2회 미만, 클릭수 5회 미만 데이터 제거
```

In [9]:
```
data = cleanse_recursive(data, shortest=2, least_click=5)
data
```

Out[9]:

	UserId	ItemId	Rating	Time
31	1	3186	4	2000-12-31 22:00:19
22	1	1270	5	2000-12-31 22:00:55
27	1	1721	4	2000-12-31 22:00:55
37	1	1022	5	2000-12-31 22:00:55
24	1	2340	3	2000-12-31 22:01:43
...	...	...	...	...
1000019	6040	2917	4	2001-08-10 14:40:29
999988	6040	1921	4	2001-08-10 14:41:04
1000172	6040	1784	3	2001-08-10 14:41:04
1000167	6040	161	3	2001-08-10 14:41:26
1000042	6040	1221	4	2001-08-20 13:44:15

999611 rows × 4 columns

In [10]:
```
# 1) 세션 길이 분포 확인
print("검증 세션 수:", data['UserId'].nunique())
print("세션별 길이(상위 10개):")
print(data.groupby('UserId').size().sort_values(ascending=False).head(

# 2) 길이 1인 세션 개수
print("길이 1 세션 비율:",
      (data.groupby('UserId').size() == 1).mean())
```

```
검증 세션 수: 6040
세션별 길이(상위 10개):
UserId
4169    2277
1680    1850
4277    1740
1941    1594
1181    1521
889     1514
3618    1342
2063    1320
1150    1299
1015    1285
dtype: int64
길이 1 세션 비율: 0.0
```

In [11]:
```
def split_by_date(data: pd.DataFrame, n_days: int):
    final_time = data['Time'].max()
    cutoff_time = final_time - dt.timedelta(days=n_days)

    # Train: cutoff 이전의 데이터
    train = data[data['Time'] < cutoff_time]

    # Test: cutoff 이후의 데이터, 단 아이템은 train에 있던 것만
    test = data[(data['Time'] >= cutoff_time) & (data['ItemId'].isin(t

    return train, test
```

In [ ]:

In [ ]:

In [12]:
```
# data에 대한 정보를 살펴봅니다.
def stats_info(data: pd.DataFrame, status: str):
    print(f'* {status} Set Stats Info\n'
          f'\t Events: {len(data)}\n'
          f'\t Sessions: {data["UserId"].nunique()}\n'
          f'\t Items: {data["ItemId"].nunique()}\n'
          f'\t First Time : {data["Time"].min()}\n'
          f'\t Last Time : {data["Time"].max()}\n')
```

In [ ]:

Type *Markdown* and LaTeX: α^2

In [ ]:

In [ ]:

In [13]:
```
def indexing(df, id2idx):
    df['item_idx'] = df['ItemId'].map(lambda x: id2idx.get(x, -1))  #
    return df
```

In [ ]:

In [ ]:

In [14]:
```
class SessionDataset:
    """Credit to yhs-968/pyGRU4REC."""

    def __init__(self, data):
        self.df = data
        self.click_offsets = self.get_click_offsets()
        self.session_idx = np.arange(self.df['UserId'].nunique())  # i

    def get_click_offsets(self):
        """
        Return the indexes of the first click of each session IDs,
        """
        offsets = np.zeros(self.df['UserId'].nunique() + 1, dtype=np.i
        offsets[1:] = self.df.groupby('UserId').size().cumsum()
        return offsets
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [15]:
```
class SessionDataLoader:
    """Credit to yhs-968/pyGRU4REC."""


    def __init__(self, dataset: SessionDataset, batch_size=50):
        self.dataset = dataset

## 이부분에 batch_size 를 마지막에 남은 부분으 처리할수있게 코드 작성
#         self.batch_size = batch_size
        self.batch_size = min(batch_size, len(dataset.session_idx))  #

    def __iter__(self):
        """ Returns the iterator for producing session-parallel traini
        Yields:
            input (B,):  Item indices that will be encoded as one-hot
            target (B,): a Variable that stores the target item indice
            masks: Numpy array indicating the positions of the sessior
        """

        start, end, mask, last_session, finished = self.initialize()
        """
        start : Index Where Session Start
        end : Index Where Session End
        mask : indicator for the sessions to be terminated
        """

        while not finished:
            min_len = (end - start).min() - 1  # Shortest Length Among
            for i in range(min_len):
                # Build inputs & targets
                inp = self.dataset.df['item_idx'].values[start + i]
                target = self.dataset.df['item_idx'].values[start + i
                yield inp, target, mask

            start, end, mask, last_session, finished = self.update_sta

    def initialize(self):
        first_iters = np.arange(self.batch_size)    # 첫 배치에 사용할 세션
        last_session = self.batch_size - 1    # 마지막으로 다루고 있는 세션 
        start = self.dataset.click_offsets[self.dataset.session_idx[fi
        end = self.dataset.click_offsets[self.dataset.session_idx[firs
        mask = np.array([])   # session의 모든 아이템을 다 돌은 경우 mask에 추
        finished = False         # data를 전부 돌았는지 기록하기 위한 변수입니다
        return start, end, mask, last_session, finished

    def update_status(self, start: np.ndarray, end: np.ndarray, min_le
        # 다음 배치 데이터를 생성하기 위해 상태를 update합니다.

        start += min_len   # __iter__에서 min_len 만큼 for문을 돌았으므로 st
        mask = np.arange(self.batch_size)[(end - start) == 1]
        # end는 다음 세션이 시작되는 위치인데 start와 한 칸 차이난다는 것은 session

        for i, idx in enumerate(mask, start=1):  # mask에 추가된 세션 개수
            new_session = last_session + i
            if new_session > self.dataset.session_idx[-1]:  # 만약 새로운
                finished = True
                break
            # update the next starting/ending point
            start[idx] = self.dataset.click_offsets[self.dataset.sessi
            end[idx] = self.dataset.click_offsets[self.dataset.sessior
```

```
        last_session += len(mask)  # 마지막 세션의 위치를 기록해둡니다.
        return start, end, mask, last_session, finished
```

In [ ]:

In [ ]:

In [16]:
```
def mrr_k(pred, truth: int, k: int):
    indexing = np.where(pred[:k] == truth)[0]
    if len(indexing) > 0:
        return 1 / (indexing[0] + 1)
    else:
        return 0


def recall_k(pred, truth: int, k: int) -> int:
    answer = truth in pred[:k]
    return int(answer)
```

In [17]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Dropout, GRU
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from tqdm import tqdm
```

In [18]:
```
def create_model(args):
    inputs = Input(batch_shape=(args.batch_size, 1, args.num_items))
    gru, _ = GRU(args.hsz, stateful=True, return_state=True, name='GRU
    dropout = Dropout(args.drop_rate)(gru)
    predictions = Dense(args.num_items, activation='softmax')(dropout)
    model = Model(inputs=inputs, outputs=[predictions])
    model.compile(loss=categorical_crossentropy, optimizer=Adam(args.l
    model.summary()
    return model
```

In [19]:
```
class Args:
    def __init__(self, tr, val, test, batch_size, hsz, drop_rate, lr,
        self.tr = tr
        self.val = val
        self.test = test
        self.num_items = tr['ItemId'].nunique()
        self.num_sessions = tr['UserId'].nunique()
        self.batch_size = batch_size
        self.hsz = hsz
        self.drop_rate = drop_rate
        self.lr = lr
        self.epochs = epochs
        self.k = k
```

In [ ]:

In [20]:
```
# train 셋으로 학습하면서 valid 셋으로 검증합니다.
def train_model(model, args):
    train_dataset = SessionDataset(args.tr)
    train_loader = SessionDataLoader(train_dataset, batch_size=args.ba

    tf.config.run_functions_eagerly(True)

    for epoch in range(1, args.epochs + 1):
        total_step = len(args.tr) - args.tr['UserId'].nunique()
        tr_loader = tqdm(train_loader, total=total_step // args.batch_
        for feat, target, mask in tr_loader:
            reset_hidden_states(model, mask)  # 종료된 session은 hidden_

            input_ohe = to_categorical(feat, num_classes=args.num_iter
            input_ohe = np.expand_dims(input_ohe, axis=1)
            target_ohe = to_categorical(target, num_classes=args.num_i

            result = model.train_on_batch(input_ohe, target_ohe)
            tr_loader.set_postfix(train_loss=result[0], accuracy = res

        val_recall, val_mrr = get_metrics(args.val, model, args, args.

        print(f"\t - Recall@{args.k} epoch {epoch}: {val_recall:3f}")
        print(f"\t - MRR@{args.k}    epoch {epoch}: {val_mrr:3f}\n")


def reset_hidden_states(model, mask):
    gru_layer = model.get_layer(name='GRU')  # model에서 gru layer를 가
    hidden_states = gru_layer.states[0].numpy()  # gru_layer의 paramet
    for elt in mask:  # mask된 인덱스 즉, 종료된 세션의 인덱스를 돌면서
        hidden_states[elt, :] = 0  # parameter를 초기화 합니다.
    gru_layer.reset_states(states=hidden_states)


def get_metrics(data, model, args, k: int):  # valid셋과 test셋을 평가하는
                                             # train과 거의 같지
    dataset = SessionDataset(data)
    loader = SessionDataLoader(dataset, batch_size=args.batch_size)
    recall_list, mrr_list = [], []

    total_step = len(data) - data['UserId'].nunique()

    for inputs, label, mask in tqdm(loader, total=total_step // args.b
        reset_hidden_states(model, mask)
        input_ohe = to_categorical(inputs, num_classes=args.num_items)
        input_ohe = np.expand_dims(input_ohe, axis=1)
        #
        if input_ohe.shape[0] < args.batch_size:
            pad_len = args.batch_size - input_ohe.shape[0]
            padding = np.zeros((pad_len, 1, args.num_items))
            input_ohe = np.concatenate([input_ohe, padding], axis=0)

        pred = model.predict(input_ohe, batch_size=args.batch_size)


        pred_arg = tf.argsort(pred, direction='DESCENDING')  # softmax

        length = len(inputs)
        recall_list.extend([recall_k(pred_arg[i], label[i], k) for i i
        mrr_list.extend([mrr_k(pred_arg[i], label[i], k) for i in rang
```

```
    print(recall_list)
    print(mrr_list)
    recall, mrr = np.mean(recall_list), np.mean(mrr_list)
    return recall, mrr
```

In [ ]:

In [21]:
```
def test_model(model, args, test):
    test_recall, test_mrr = get_metrics(test, model, args, 20)
    print(f"\t - Recall@{args.k}: {test_recall:3f}")
    print(f"\t - MRR@{args.k}: {test_mrr:3f}\n")
```

In [ ]:
```
import pandas as pd

# 실험 결과를 저장할 리스트 초기화
results_list = []

def run_and_log_experiment(data, n_days, hsz, epochs, batch_size, leas
    """
    주어진 하이퍼파라미터로 전체 실험을 실행하고 결과를 기록하는 함수
    """

    # 1. 현재 실험의 하이퍼파라미터 설정 기록
    params = {
        'n_days': n_days,
        'hsz': hsz,
        'epochs': epochs,
        'batch_size': batch_size,
        'least_click_tr': least_click
    }
    print(f"--- 실험 시작: {params} ---")

    # 2. 데이터 분할 및 추가 정제
    try:
        # 데이터 분할 : 14일 : 2일 분량 테스트
        #tr, test = split_by_date(data, n_days=14)  # 마지막 이틀만 테스트
        #tr, val = split_by_date(tr, n_days=14)  # 마지막 이틀만 테스트용으
        tr, test = split_by_date(data, n_days=n_days)
        tr, val = split_by_date(tr, n_days=n_days)

        # tr = cleanse_recursive(tr, shortest=2, least_click=5)
        # val = cleanse_recursive(val, shortest=2, least_click=1)
        # test = cleanse_recursive(test, shortest=2, least_click=1)
        tr = cleanse_recursive(tr, shortest=2, least_click=least_click
        val = cleanse_recursive(val, shortest=2, least_click=1)
        test = cleanse_recursive(test, shortest=2, least_click=1)

        stats_info(tr, 'train')
        stats_info(val, 'valid')
        stats_info(test, 'test')


        # 데이터 정제 후 세션이 너무 적으면 실험 중단 <==== 확인
        if len(tr) < 10 or len(val) < 10 or len(test) < 10:
            print("데이터 정제 후 분할 데이터별 세션이 너무 적어(<10) 실험을 중단합니
            return

        # train set에 없는 아이템이 val, test기간에 생길 수 있으므로 train data
        id2idx = {item_id : index for index, item_id in enumerate(tr[
        tr = indexing(tr, id2idx)
        val = indexing(val, id2idx)
        test = indexing(test, id2idx)

        # 검증데이터 세션 길이 분포 확인
        print("검증 세션 수:", val['UserId'].nunique())
        print("세션별 길이(상위 10개):")
        print(val.groupby('UserId').size().sort_values(ascending=False

        # 검증데이터 길이 1인 세션 개수
        print("길이 1 세션 비율:",(val.groupby('UserId').size() == 1).mea
        #tr

        tr_dataset = SessionDataset(tr)
```

```
        tr_dataset.df.head(10)
        tr_dataset.click_offsets
        tr_dataset.session_idx
        start = tr_dataset.click_offsets[tr_dataset.session_idx[[[0,1,
        end = tr_dataset.click_offsets[tr_dataset.session_idx[[0,1,2,3
        #start
        #end
        (end - start).min() -1
        inp = tr_dataset.df['item_idx'].values[start + 20]  #20?
        #inp
        target = tr_dataset.df['item_idx'].values[start + 20 + 1] # 20
        #target

        tr_data_loader = SessionDataLoader(tr_dataset, batch_size=batc
        #tr_dataset.df.head(15)

        iter_ex = iter(tr_data_loader)

        inputs, labels, mask =  next(iter_ex)
        print(f'Model Input Item Idx are : {inputs}')
        print(f'Label Item Idx are : {"":5} {labels}')
        print(f'Previous Masked Input Idx are {mask}')


    except Exception as e:
        print(f"데이터 처리 중 오류 발생: {e}")
        return

    # 3. 모델 생성 및 학습
    ## 배치사이즈를 너무 크게두면 밑에 평가 부분에서 nan이 나옵니다
    #args = Args(tr, val, test, batch_size=256, hsz=50, drop_rate=0.1,
    args = Args(tr, val, test, batch_size=batch_size, hsz=hsz, drop_ra
    model = create_model(args)

    print("\n모델 학습 시작...")
    train_model(model, args)

    print("\n모델 테스트 시작...")
    test_model(model, args, test)

    # 4. 성능 평가 및 결과 저장
    print("\n테스트 세트 평가 시작...")
    test_recall, test_mrr = get_metrics(test, model, args, k)
    print(f"Recall@{k}: {test_recall:.4f}, MRR@{k}: {test_mrr:.4f}")

    params['Recall@20'] = test_recall
    params['MRR@20'] = test_mrr
    results_list.append(params)

    print(f"--- 실험 종료 ---\n")


# --- 실험 실행 메인 루프 ---

#run_and_log_experiment(data, n_days, hsz, epochs, batch_size, least_c
# 테스트할 하이퍼파라미터 조합 정의
n_days_options = [30, 90] #14
hsz_options = [50, 100]
epochs_options = [3, 10]
batch_size_options = [128, 256]
least_click_options = [3, 5]
```

```
# 모든 조합에 대해 실험 실행
for n_days in n_days_options:
    for hsz in hsz_options:
        for epochs in epochs_options:
            for batch_size in batch_size_options:
                for least_click in least_click_options:
                    # 원본 데이터(data)를 사용하여 매번 새로운 실험 실행
                    run_and_log_experiment(data, n_days, hsz, epochs,
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.0625, 0, 0,
0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.166666666666
66666, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.05, 0, 0, 0, 0, 0, 0.125, 0, 0,
0, 0, 0, 0, 0, 0]
         - Recall@20 epoch 3: 0.056604
         - MRR@20    epoch 3: 0.017964


Train:  98%|██████████| 7564/7735 [03:15<00:04, 38.61it/s, accuracy=
0.0391, train_loss=5.76]
Evaluation:   8%|▉         | 1/13 [00:00<00:05,  2.22it/s]
```

In [ ]:
```
# 최종 결과를 데이터프레임으로 변환하고 성능이 높은 순으로 정렬
results_df = pd.DataFrame(results_list)
results_df_sorted = results_df.sort_values(by='Recall@20', ascending=F

# 최종 결과 확인
display(results_df_sorted)
```