```
import pandas as pd
import numpy as np
from tqdm.notebook import tqdm

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
import pytorch_lightning as pl

np.random.seed(123)


ratings = pd.read_csv('/content/drive/MyDrive/rating.csv',
                      parse_dates=['timestamp'])


rand_userIds = np.random.choice(ratings['userId'].unique(),
                                size=int(len(ratings['userId'].unique())*0.3),
                                replace=False)

ratings = ratings.loc[ratings['userId'].isin(rand_userIds)]

print('There are {} rows of data from {} users'.format(len(ratings), len(rand_userIds)))
```

```
     There are 6027314 rows of data from 41547 users
```

```
ratings.sample(5)
```

	userId	movieId	rating	timestamp
3840312	26182	3704	4.0	2007-01-31 21:56:52
7608731	52439	3365	4.0	2004-03-21 08:02:56
19363634	134060	1027	3.0	2003-07-15 22:43:45
17181947	118860	2629	1.0	2007-11-29 21:27:08
9344779	64638	4723	2.0	2001-09-10 20:11:41

```
ratings['rank_latest'] = ratings.groupby(['userId'])['timestamp'] \
                                .rank(method='first', ascending=False)

train_ratings = ratings[ratings['rank_latest'] != 1]
test_ratings = ratings[ratings['rank_latest'] == 1]

# drop columns that we no longer need
train_ratings = train_ratings[['userId', 'movieId', 'rating']]
test_ratings = test_ratings[['userId', 'movieId', 'rating']]


train_ratings.loc[:, 'rating'] = 1

train_ratings.sample(5)
```

	userId	movieId	rating
3411906	23263	5481	1
1815983	12245	5464	1
16198592	112109	6	1
13914487	96124	1247	1
1445807	9790	1690	1

```
class MovieLensTrainDataset(Dataset):
    """MovieLens PyTorch Dataset for Training

    Args:
        ratings (pd.DataFrame): Dataframe containing the movie ratings
        all_movieIds (list): List containing all movieIds

    """

    def __init__(self, ratings, all_movieIds):
        self.users, self.items, self.labels = self.get_dataset(ratings, all_movieIds)

    def __len__(self):
        return len(self.users)

    def __getitem__(self, idx):
        return self.users[idx], self.items[idx], self.labels[idx]

    def get_dataset(self, ratings, all_movieIds):
        users, items, labels = [], [], []
        user_item_set = set(zip(ratings['userId'], ratings['movieId']))

        num_negatives = 4
        for u, i in user_item_set:
            users.append(u)
            items.append(i)
            labels.append(1)
            for _ in range(num_negatives):
                negative_item = np.random.choice(all_movieIds)
                while (u, negative_item) in user_item_set:
                    negative_item = np.random.choice(all_movieIds)
                users.append(u)
                items.append(negative_item)
                labels.append(0)

        return torch.tensor(users), torch.tensor(items), torch.tensor(labels)


class NCF(pl.LightningModule):

    """ Neural Collaborative Filtering (NCF)

        Args:
            num_users (int): Number of unique users
```

```
            num_items (int): Number of unique items
            ratings (pd.DataFrame): Dataframe containing the movie ratings for training
            all_movieIds (list): List containing all movieIds (train + test)
    """

    def __init__(self, num_users, num_items, ratings, all_movieIds):
        super().__init__()
        self.user_embedding = nn.Embedding(num_embeddings=num_users, embedding_dim=8)
        self.item_embedding = nn.Embedding(num_embeddings=num_items, embedding_dim=8)
        self.fc1 = nn.Linear(in_features=16, out_features=64)
        self.fc2 = nn.Linear(in_features=64, out_features=32)
        self.output = nn.Linear(in_features=32, out_features=1)
        self.ratings = ratings
        self.all_movieIds = all_movieIds

    def forward(self, user_input, item_input):

        # Pass through embedding layers
        user_embedded = self.user_embedding(user_input)
        item_embedded = self.item_embedding(item_input)

        # Concat the two embedding layers
        vector = torch.cat([user_embedded, item_embedded], dim=-1)

        # Pass through dense layer
        vector = nn.ReLU()(self.fc1(vector))
        vector = nn.ReLU()(self.fc2(vector))

        # Output layer
        pred = nn.Sigmoid()(self.output(vector))

        return pred

    def training_step(self, batch, batch_idx):
        user_input, item_input, labels = batch
        predicted_labels = self(user_input, item_input)
        loss = nn.BCELoss()(predicted_labels, labels.view(-1, 1).float())
        return loss

    def configure_optimizers(self):
        return torch.optim.Adam(self.parameters())

    def train_dataloader(self):
        return DataLoader(MovieLensTrainDataset(self.ratings, self.all_movieIds),
                          batch_size=512, num_workers=2)


num_users = ratings['userId'].max()+1
num_items = ratings['movieId'].max()+1

all_movieIds = ratings['movieId'].unique()

model = NCF(num_users, num_items, train_ratings, all_movieIds)


trainer = pl.Trainer(max_epochs=5
```

```
trainer = pl.Trainer(max_epochs=5,
                     accelerator="gpu",
                     gpus = 1,
                     reload_dataloaders_every_n_epochs=True,
                     progress_bar_refresh_rate=50,
                     logger=False,
                     checkpoint_callback=False)

trainer.fit(model)
```

```
/usr/local/lib/python3.7/dist-packages/pytorch_lightning/trainer/connectors/callback
  f"Setting `Trainer(checkpoint_callback={checkpoint_callback})` is deprecated in v1
/usr/local/lib/python3.7/dist-packages/pytorch_lightning/trainer/connectors/callback
  f"Setting `Trainer(progress_bar_refresh_rate={progress_bar_refresh_rate})` is depr
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name           | Type      | Params
-------------------------------------------
0 | user_embedding | Embedding | 1.1 M
1 | item_embedding | Embedding | 1.1 M
2 | fc1            | Linear    | 1.1 K
3 | fc2            | Linear    | 2.1 K
4 | output         | Linear    | 33
-------------------------------------------
2.2 M     Trainable params
0         Non-trainable params
2.2 M     Total params
8.645     Total estimated model params size (MB)
Epoch 4: 100%                                                    58455/58455 [10:30<00:0
```

```
# User-item pairs for testing
test_user_item_set = set(zip(test_ratings['userId'], test_ratings['movieId']))

# Dict of all items that are interacted with by each user
user_interacted_items = ratings.groupby('userId')['movieId'].apply(list).to_dict()

hits = []
for (u,i) in tqdm(test_user_item_set):
    interacted_items = user_interacted_items[u]
    not_interacted_items = set(all_movieIds) - set(interacted_items)
    selected_not_interacted = list(np.random.choice(list(not_interacted_items), 99))
    test_items = selected_not_interacted + [i]

    predicted_labels = np.squeeze(model(torch.tensor([u]*100),
                                        torch.tensor(test_items)).detach().numpy())

    top10_items = [test_items[i] for i in np.argsort(predicted_labels)[::-1][0:10].tolist(

    if i in top10_items:
        hits.append(1)
    else:
        hits.append(0)
```

```
print("The Hit Ratio @ 10 is {:.2f}".format(np.average(hits)))
```

```
100%                                                  41547/41547 [04:22<00:00, 155.65it/s]
The Hit Ratio @ 10 is 0.86
```