Personalisation_Assignment_2

Korina Kyriaki Zaromytidou

*I had to run the notebook on Colab, as my personal laptop was not running the code. *

Code from Week 6.1- Embeddings for Recommendation Notebook. My response to the tasks are at the bottom of the page.

```
import pandas as pd
```

```
from google.colab import files
uploaded = files.upload()
```

Choose files ratings.csv

- **ratings.csv**(text/csv) - 2438266 bytes, last modified: 03/06/2023 - 100% done

```
Saving ratings.csv to ratings.csv
```

```
import pandas as pd
df = pd.read_csv('ratings.csv')
```

```
len(df)
```

```
100004
```

```
df.tail(100)
```

	userId	movieId	rating	timestamp
99904	671	590	4.0	1065149296
99905	671	608	4.0	1064890575
99906	671	745	4.0	1065149085
99907	671	919	4.0	1065149458
99908	671	1035	5.0	1065149492
...	...	...	...	...
99999	671	6268	2.5	1065579370
100000	671	6269	4.0	1065149201
100001	671	6365	4.0	1070940363
100002	671	6385	2.5	1070979663
100003	671	6565	3.5	1074784724

100 rows × 4 columns

Preprocessing

```
user_ids = df["userId"].unique().tolist()
movie_ids = df["movieId"].unique().tolist()
```

```
len(movie_ids)
```

```
9066
```

```
len(user_ids)
```

```
671
```

```
#Non-sequential list of ids
movie_ids[:6]
```

```
[31, 1029, 1061, 1129, 1172, 1263]
```

Dictionary

```
#Make a dictionary mapping ids (keys) to indexes (values)
user_id_to_index = {x: i for i, x in enumerate(user_ids)}
movie_id_to_index = {x: i for i, x in enumerate(movie_ids)}


#Make a new column in the dataframe which contains the appropriate index for each user and movie
df["user_index"] = [user_id_to_index[i] for i in df["userId"]]
df["movie_index"] = [movie_id_to_index[i] for i in df["movieId"]]


df.head(10)
```

	userId	movieId	rating	timestamp	user_index	movie_index
0	1	31	2.5	1260759144	0	0
1	1	1029	3.0	1260759179	0	1
2	1	1061	3.0	1260759182	0	2
3	1	1129	2.0	1260759185	0	3
4	1	1172	4.0	1260759205	0	4
5	1	1263	2.0	1260759151	0	5
6	1	1287	2.0	1260759187	0	6
7	1	1293	2.0	1260759148	0	7
8	1	1339	3.5	1260759125	0	8
9	1	1343	2.0	1260759131	0	9

**Scaling the ratings* *

```
df["rating"].describe()
```

```
count    100004.000000
mean          3.543608
std           1.058064
min           0.500000
25%           3.000000
50%           4.000000
75%           4.000000
max           5.000000
Name: rating, dtype: float64
```

```
from sklearn.preprocessing import MinMaxScaler
##Pick the range\
df["rating"] = MinMaxScaler().fit_transform(df["rating"].values.reshape(-1, 1))


df["rating"].describe()
```

```
count    100004.000000
mean          0.676357
std           0.235125
min           0.000000
25%           0.555556
50%           0.777778
75%           0.777778
max           1.000000
Name: rating, dtype: float64
```

Training Set

```
from sklearn.model_selection import train_test_split
#Inputs
x = df[["user_index", "movie_index"]]
#Outputs
y = df["rating"]
#Get train-test split
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.1, random_state=42)
```

Making a Custom Model

```
#Install libraries (only do this once!)
!pip install torch torchvision torchaudio
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.0.1+cu118)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.15.2+cu118)
Requirement already satisfied: torchaudio in /usr/local/lib/python3.10/dist-packages (2.0.2+cu118)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.12.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.11.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.2)
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.10/dist-packages (from torch) (2.0.0)
Requirement already satisfied: cmake in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch) (3.25.2)
Requirement already satisfied: lit in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch) (16.0.5)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision) (1.22.4)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from torchvision) (2.27.1)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision) (8.4.(
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch) (2.1.2)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->torchvisi
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests->torcl
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (3.4)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
```

```
#import library
import torch


#Define class and subclass torch.nn.Module
class LouisNet(torch.nn.Module):

    #Override __init__()
    def __init__(self):
        super().__init__()
        print("__init__ called")

    #Override forward()
    def forward(self, inputs):
        print("\nforwards pass (new batch)")
        print(inputs,"\n")
        #return the output (its just the input, unchanged)
        return inputs

#Make a new instance of LouisNet
louisNet = LouisNet()
loss_fn = torch.nn.MSELoss()

#Fake dataset
x = torch.FloatTensor([[1],[2],[3],[4]])
y = torch.FloatTensor([[2],[3],[4],[5]])

#Do a forwards pass
prediction = louisNet(x)
loss = loss_fn(prediction, y)
```

```
__init__ called

forwards pass (new batch)
tensor([[1.],
        [2.],
        [3.],
        [4.]])
```

The Dot Product Recommender Model

```
class RecommenderNet(torch.nn.Module):
    def __init__(self, num_users, num_movies, embedding_size=20):
        super().__init__()
        self.user_embedding = torch.nn.Embedding(num_users, embedding_size)
        self.user_bias = torch.nn.Embedding(num_users, 1)
        self.movie_embedding = torch.nn.Embedding(num_movies, embedding_size)
        self.movie_bias = torch.nn.Embedding(num_movies, 1)
        self.sig = torch.nn.Sigmoid()

    def forward(self, inputs):
        #Split out indexes
        user_indexes = inputs[:, 0]
        movie_indexes = inputs[:, 1]
        #Forward pass on embedding layer
        user_vector = self.user_embedding(user_indexes)
        user_bias = self.user_bias(user_indexes).flatten()
        movie_vector = self.movie_embedding(movie_indexes)
```

```
        movie_bias = self.movie_bias(movie_indexes).flatten()
        #Dot product
        dot = (user_vector * movie_vector).sum(1)
        with_bias = dot + user_bias + movie_bias
        #Activation function
        output = self.sig(with_bias)
        return output
```

Set up model

```
#Pick Embedding size
EMBEDDING_SIZE = 16
#Make new object (calls __init__())
num_users = len(user_ids)
num_movies = len(movie_ids)
model = RecommenderNet(num_users, num_movies, EMBEDDING_SIZE)
```

Training and Datasets in PyTorch

```
from torch.utils.data import DataLoader
from torch.utils.data import Dataset

#Make a subclass to hold our dataset (movie - user pairs (input) and a rating (label))
class MoviesDataset(Dataset):
    def __init__(self, X,y):
        self.X = torch.IntTensor(X)
        self.y = torch.FloatTensor(y)
    def __len__(self):
        return len(self.X)
    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]
```

```
#Use our train - validation split to make DataLoader objects
train_dl = DataLoader(MoviesDataset(x_train.values,y_train.values), batch_size=64, shuffle=True)
validation_dl = DataLoader(MoviesDataset(x_val.values,y_val.values), batch_size=64, shuffle=True)
```

```
epochs = 10
#Use Mean Squared Error as a loss function
loss_fn = torch.nn.MSELoss()
#Use the Adam algorithm to update the weights based on the loss
optimizer = torch.optim.Adam(model.parameters(),lr=0.01)
```

```
#Use a for loop to repeat for the desired number of epochs
for i in range(epochs):

    model.train(True)

    #Use a for loop for each batch (provided by the Dataloader)
    running_loss = 0.0
    for (index, batch) in enumerate(train_dl):

        #Get batch
        inputs, labels = batch
        model.zero_grad()

        #Forward pass
        prediction = model(inputs)

        #Get Loss
        loss = loss_fn(prediction, labels)

        #Update weights (back prop)
        loss.backward()
        optimizer.step()
        running_loss += loss

    avg_loss = running_loss / (index + 1)

    model.train(False)

    #Now try with the validation set (no need to update weights, just get loss)
    running_vloss = 0.0
    for index, vdata in enumerate(validation_dl):
        vinputs, vlabels = vdata
```

```
        voutputs = model(vinputs)
        vloss = loss_fn(voutputs, vlabels)
        running_vloss += vloss

    avg_vloss = running_vloss / (index + 1)
    print('Loss {} Validation Loss {}'.format(avg_loss, avg_vloss))
```

```
Loss 0.1746886521577835 Validation Loss 0.11890841275453568
Loss 0.07875106483697891 Validation Loss 0.0857386589050293
Loss 0.04998217523097992 Validation Loss 0.07208939641714096
Loss 0.0377763994038105 Validation Loss 0.0667073205113411
Loss 0.0315697006881237 Validation Loss 0.06350364536046982
Loss 0.027780719101428986 Validation Loss 0.0638813003897667
Loss 0.025565870106220245 Validation Loss 0.06260982155799866
Loss 0.023830236867070198 Validation Loss 0.06335168331861496
Loss 0.022634232416749 Validation Loss 0.06368756294250488
Loss 0.02177959494292736 Validation Loss 0.062364667654037476
```

Save and Reload models

```
torch.save(model.state_dict(), 'model_weights.pth')
```

```
model = RecommenderNet(num_users, num_movies, EMBEDDING_SIZE)
model.load_state_dict(torch.load('model_weights.pth'))
model.eval()
```

```
RecommenderNet(
  (user_embedding): Embedding(671, 16)
  (user_bias): Embedding(671, 1)
  (movie_embedding): Embedding(9066, 16)
  (movie_bias): Embedding(9066, 1)
  (sig): Sigmoid()
)
```

Accessing the Embeddings

```
num_users, EMBEDDING_SIZE, model.user_embedding
```

```
(671, 16, Embedding(671, 16))
```

```
num_users, EMBEDDING_SIZE, model.movie_embedding
```

```
(671, 16, Embedding(9066, 16))
```

Making Predictions

```
from google.colab import files
uploaded = files.upload()
```

Choose files movies.csv

- **movies.csv**(text/csv) - 458390 bytes, last modified: 03/06/2023 - 100% done

```
Saving movies.csv to movies.csv
```

```
import pandas as pd
movie_data = pd.read_csv('movies.csv')
```

Making predictions and argsort()

```
#movie_data = pd.read_csv('movies.csv')

def get_top_n(user=0, n=10):
    # Get Movie Names
    top_n_indexes = get_top_n_indexes(user, n)
    top_n = get_names_for_indexes(top_n_indexes)
    return top_n

def get_names_for_indexes(indexes):
    return [movie_data[movie_data["movieId"] == movie_ids[i]]["title"].item() for i in indexes]

def get_top_n_indexes(user=0, n=10):
```

```
    # For one user, make a pair with every movie index
    x = torch.IntTensor([[user, i] for i in np.arange(num_movies)])
    # Predict
    predicted_ratings = model(x)
    # Get Top-N indexes
    top_n_indexes = predicted_ratings.argsort()[-n:]
    return top_n_indexes
```

```
import numpy as np
```

```
#Random users top 10
get_top_n(np.random.randint(num_users))
```

```
['Trouble in Paradise (1932)',
 'Audition (Ôdishon) (1999)',
 'Prison Break: The Final Break (2009)',
 'Open Season (2006)',
 'Solaris (Solyaris) (1972)',
 'Old Joy (2006)',
 'Wild Blue Yonder, The (2005)',
 'Compulsion (1959)',
 'Willie & Phil (1980)',
 'Appleseed (Appurushîdo) (2004)']
```

Assessed Assignment 2

Task 1

Diversity

1. Calculate every user's top 10 For each top 10

```
top10_movies_per_user = []  # Create an empty list to store top 10 recommendations for each user

# Iterate over each user
for user in range(num_users):
    # Get the top 10 movie recommendations for the current user
    top10_movies = get_top_n(user, n=10)
    top10_movies_per_user.append(top10_movies)  # Append the top 10 movies to the list
```

```
print(df.head())
```

	userId	movieId	rating	timestamp	user_index	movie_index
0	1	31	0.444444	1260759144	0	0
1	1	1029	0.555556	1260759179	0	1
2	1	1061	0.555556	1260759182	0	2
3	1	1129	0.333333	1260759185	0	3
4	1	1172	0.777778	1260759205	0	4

```
for user_movies in top10_movies_per_user[:2]:
    print(user_movies)
```

```
['Fox and His Friends (Faustrecht der Freiheit) (1975)', 'Mechanic, The (2011)', 'Lady Vengeance (Sympathy for Lady Venge
['Wages of Fear, The (Salaire de la peur, Le) (1953)', 'Arizona Dream (1993)', 'Wild Bunch, The (1969)', 'Roadkill (a.k.a
```

```
top_recommendation_user1 = top10_movies_per_user[0][3]

# Print the top recommendation for the first user
print("Top Recommendation for User 1:", top_recommendation_user1)
```

```
Top Recommendation for User 1: Brother, Can You Spare a Dime? (1975)
```

2. Get the Embedding for each film .

Before getting the function to compute this, I had to run some tests in regards to the size and dimensionality of the lists and tensor, as I had consecutives errors.

```
first_movie_index = 0  # Assuming the index of the first movie is 0
embedding = model.movie_embedding.weight.data[first_movie_index]
print("Embedding for the first movie:", embedding)
```

```
Embedding for the first movie: tensor([-0.2856,  0.2713, -0.9243,  0.2310,  0.1091, -0.0375, -0.3966, -0.3373,
         0.3824,  0.2459,  1.2262, -0.8370, -0.2725,  0.0801,  0.0944,  0.6690])
```

```
first_user_movies = top10_movies_per_user[0]  # Assuming the first user is at index 0
first_movie_name = first_user_movies[0]  # Assuming the first movie is at index 0

if movie_data["movieId"].isin([first_movie_name]).any():
    first_movie_index = movie_data[movie_data["movieId"] == first_movie_name].index[0]
    embedding = model.movie_embedding.weight.data[first_movie_index]
    print("Embedding for the first movie of the first user:", embedding)
else:
    print("Movie not found in the dataset.")
```

```
Movie not found in the dataset.
```

```
print("Movies for the first user in top10_movies:")
print(top10_movies_per_user[0])
```

```
Movies for the first user in top10_movies:
['Fox and His Friends (Faustrecht der Freiheit) (1975)', 'Mechanic, The (2011)', 'Lady Vengeance (Sympathy for Lady Venge
```

```
first_movie_title = top10_movies_per_user[0][0]
first_movie_id = movie_data[movie_data["title"] == first_movie_title]["movieId"].item()
first_movie_index = movie_id_to_index[first_movie_id]
first_movie_embedding = model.movie_embedding.weight.data[first_movie_index]

print("Embedding for the first movie:")
print(first_movie_embedding)
```

```
Embedding for the first movie:
tensor([ 0.8245, -0.3883,  3.0369, -0.4519, -1.5033, -0.6997, -0.4507, -1.6881,
        -0.2778,  2.8513,  1.3007,  1.2187, -0.3915, -1.5134, -1.6128, -0.0153])
```

```
from sklearn.metrics.pairwise import cosine_similarity
similarity_matrices = []

for user_movies in top10_movies_per_user:
    embeddings = []
    for movie_title in user_movies:
        movie_id = movie_data[movie_data["title"] == movie_title]["movieId"].item()
        movie_index = movie_id_to_index[movie_id]
        embedding = model.movie_embedding.weight.data[movie_index]
        embeddings.append(embedding)

    if len(embeddings) > 0:
        embeddings = torch.stack(embeddings)
        similarity_matrix = cosine_similarity(embeddings.detach().numpy())    # Calculation code the cosine similarity  taken
        similarity_matrices.append(similarity_matrix)
```

```
print("Embeddings for user:", embeddings)
```

```
Embeddings for user: tensor([[-1.1492e+00, -5.7702e-01,  3.4236e-01,  8.1605e-01, -4.4123e-01,
         -1.1210e+00,  4.7989e-01,  1.9915e+00,  9.4186e-01, -1.2094e+00,
          7.9946e-01, -6.6219e-01, -9.7850e-01, -5.6021e-01,  1.1827e+00,
          4.4492e-01],
        [-5.8015e-01, -6.2660e-02, -2.0100e-01,  1.1923e+00,  1.3810e-01,
          7.5228e-02, -2.6735e-01,  1.1267e+00,  4.5011e-01, -2.6052e-03,
          1.1260e+00, -9.1311e-01, -1.3383e+00,  2.0558e+00,  2.1987e+00,
         -2.5305e-01],
        [-1.1486e+00,  2.0657e+00, -3.7386e-01,  2.1416e+00,  1.5080e-01,
          1.4257e+00, -3.1463e-01,  1.8389e+00, -1.8259e-01, -6.9103e-02,
         -4.1531e-01, -2.5632e+00,  2.3909e+00, -1.8731e-01, -5.1830e-01,
          2.3298e+00],
        [ 4.8513e-01,  1.0761e+00, -1.1008e+00,  2.2389e+00,  9.8946e-01,
         -1.0311e+00, -1.0798e+00,  9.8449e-01, -2.8686e-01, -3.1155e+00,
          1.6454e-01, -1.6344e+00,  6.4155e-01, -1.1159e+00,  7.7410e-01,
          1.0107e+00],
        [ 7.9166e-01,  4.6288e-01, -1.7652e+00,  2.0637e+00, -1.4961e+00,
          1.7521e+00, -1.2732e+00,  1.3448e-03,  2.2375e+00, -2.7776e-01,
         -2.0809e+00, -6.6303e-01, -1.4064e+00,  2.1963e+00,  5.5491e-01,
         -1.2432e+00],
        [ 8.6273e-01,  1.2248e+00, -1.3818e+00,  2.6373e+00, -2.3372e-02,
```

```
          9.5108e-01,  6.9605e-01, -8.5261e-01, -1.2152e+00,  7.4500e-01,
         -5.6310e-01, -1.4822e+00,  1.2780e+00, -1.2787e-01,  3.4969e-01,
         -6.0165e-01],
        [ 2.3721e-01,  1.0991e+00, -6.1679e-01,  1.3641e+00,  7.2585e-01,
          1.6856e+00,  9.9567e-01,  1.4153e+00, -1.9991e-01, -3.2926e+00,
          2.4992e-01, -2.3061e+00,  1.6731e+00, -1.1905e+00,  1.4193e+00,
         -2.0097e+00],
        [ 5.4769e-01,  6.0897e-01, -3.3514e-01,  1.0775e+00, -1.8738e-02,
          4.4558e-01, -3.0034e-01,  1.2049e+00,  4.2931e-01,  7.8244e-01,
         -4.3605e-01,  1.5462e-01, -1.4606e+00, -1.5958e-01,  1.5940e+00,
          9.3191e-01],
        [ 1.2152e+00, -1.0320e+00, -1.3218e+00,  2.7798e+00, -3.1531e-01,
          9.4870e-01, -1.5954e+00, -7.1949e-02, -1.0215e+00, -1.7631e+00,
         -6.0030e-01, -1.8585e+00, -1.0322e+00,  1.0321e+00, -1.4675e+00,
         -8.3287e-01],
        [ 2.5063e+00,  1.5890e+00, -7.4906e-01,  1.2197e+00,  8.5672e-01,
          2.4128e-01, -2.5668e+00,  3.8478e-01, -3.4791e-02, -1.8684e+00,
         -1.0276e+00,  5.0274e-01, -2.0769e+00, -1.8350e-01,  9.8415e-01,
         -1.9107e-01]])
```

```
print("similarity_matrix:")
print(similarity_matrix)
```

```
similarity_matrix:
[[ 1.0000001   0.5312308   0.13823313  0.42736378 -0.02461929 -0.29471546
   0.28046483  0.35999396 -0.0282383   0.00563443]
 [ 0.5312308   0.99999994  0.08153012  0.18112086  0.4171405   0.04899865
   0.17801307  0.533533    0.20199001  0.19859974]
 [ 0.13823313  0.08153012  1.          0.5151193   0.08912332  0.49796444
   0.42102715  0.18925467  0.19489163 -0.04261153]
 [ 0.42736378  0.18112086  0.5151193   0.9999999   0.07616333  0.3223509
   0.6290594   0.19421129  0.44423807  0.51745945]
 [-0.02461929  0.4171405   0.08912332  0.07616333  1.0000001   0.31056425
   0.14275332  0.41437787  0.5599405   0.46842474]
 [-0.29471546  0.04899865  0.49796444  0.3223509   0.31056425  1.0000001
   0.45087397  0.13566267  0.43487468  0.1161043 ]
 [ 0.28046483  0.17801307  0.42102715  0.6290594   0.14275332  0.45087397
   1.         -0.01927201  0.3172913   0.20947362]
 [ 0.35999396  0.533533    0.18925467  0.19421129  0.41437787  0.13566267
  -0.01927201  1.0000001   0.03977716  0.510338  ]
 [-0.0282383   0.20199001  0.19489163  0.44423807  0.5599405   0.43487468
   0.3172913   0.03977716  1.          0.47308555]
 [ 0.00563443  0.19859974 -0.04261153  0.51745945  0.46842474  0.1161043
   0.20947362  0.510338    0.47308555  0.99999994]]
```

```
from sklearn.preprocessing import MinMaxScaler

difference_matrices = []
scaler = MinMaxScaler(feature_range=(0, 1))    # Code taken from notebook Week 7.2a

for similarity_matrix in similarity_matrices:
    difference_matrix = 1 - similarity_matrix   ## Code similar to Stackoverflow question  https://stackoverflow.com/questions
    difference_matrix = scaler.fit_transform(difference_matrix)
    difference_matrices.append(difference_matrix)
```

```
print(difference_matrices)
```

```
       [0.8625746 , 0.83878714, 0.48710197, 0.4033408 , 0.746947  ,
        0.        , 0.8878653 , 1.        , 0.43881017, 0.5232319 ],
       [0.7976164 , 0.47350654, 0.8859687 , 0.889614  , 0.63357925,
        0.8518144 , 0.        , 0.7841041 , 0.64939994, 1.        ],
       [0.8690858 , 0.6022679 , 0.6276604 , 0.9632567 , 1.        ,
        1.        , 0.8172894 , 0.        , 0.7020731 , 0.4961726 ],
       [0.98758155, 0.85273045, 0.51087606, 1.        , 0.7778304 ,
        0.4350855 , 0.67113864, 0.69611365, 0.        , 0.74640924],
       [0.48868874, 0.87444186, 0.68988854, 0.50373024, 0.827134  ,
        0.5019866 , 1.        , 0.47602597, 0.72223246, 0.        ]],
      dtype=float32), array([[0.        , 0.49292168, 0.8265465 , 0.61984575, 1.        ,
        1.        , 0.7059305 , 0.62790513, 1.        , 0.95372593],
       [0.36206356, 0.        , 0.8809321 , 0.88638955, 0.56885475,
        0.73452544, 0.8064451 , 0.45764732, 0.77609444, 0.7686471 ],
       [0.6656034 , 0.9657925 , 0.        , 0.52485543, 0.88899034,
        0.3877575 , 0.5680258 , 0.7954161 , 0.78299785, 1.0000001 ],
       [0.44228742, 0.86107045, 0.46506363, 0.        , 0.9016389 ,
        0.5233963 , 0.36392704, 0.7905532 , 0.54049915, 0.46281913],
       [0.7913858 , 0.6128903 , 0.8736492 , 1.0000001 , 0.        ,
        0.5324999 , 0.8410382 , 0.5745495 , 0.42797422, 0.5098498 ],
       [1.        , 1.        , 0.4815174 , 0.7335162 , 0.67287016,
        0.        , 0.5387434 , 0.8479948 , 0.5496054 , 0.847771  ],
       [0.5557478 , 0.86433834, 0.55531025, 0.40152183, 0.836649  ,
        0.4241288 , 0.        , 1.        , 0.6639597 , 0.75821763],
       [0.49432185, 0.4905009 , 0.77761024, 0.8722199 , 0.57155097,
        0.6675887 , 1.        , 0.        , 0.93385243, 0.46964952],
       [0.794181  , 0.83912605, 0.7722036 , 0.6015803 , 0.42948592,
        0.4364862 , 0.66980034, 0.9420673 , 0.        , 0.5053795 ],
       [0.7680187 , 0.842691  , 1.        , 0.52232236, 0.51880276,
        0.68269503, 0.77557945, 0.48040372, 0.51244396, 0.        ]],
      dtype=float32)]
```

```python
mean_differences = []

for difference_matrix in difference_matrices:
    mean_difference = difference_matrix.mean()
    mean_differences.append(mean_difference)


for i, mean_difference in enumerate(mean_differences):
    print(f"Mean difference for top 10 list {i+1}: {mean_difference}")
```

```
Mean difference for top 10 list 1: 0.624325692653656
Mean difference for top 10 list 2: 0.6045036315917969
Mean difference for top 10 list 3: 0.6478860974311829
Mean difference for top 10 list 4: 0.6175259947776794
Mean difference for top 10 list 5: 0.6672927737236023
Mean difference for top 10 list 6: 0.6268792748451233
Mean difference for top 10 list 7: 0.6649858355522156
Mean difference for top 10 list 8: 0.6078617572784424
Mean difference for top 10 list 9: 0.653755784034729
Mean difference for top 10 list 10: 0.6241950392723083
Mean difference for top 10 list 11: 0.6150894165039062
Mean difference for top 10 list 12: 0.6247103214263916
Mean difference for top 10 list 13: 0.6609349846839905
Mean difference for top 10 list 14: 0.6345722675323486
Mean difference for top 10 list 15: 0.6485682725906372
Mean difference for top 10 list 16: 0.6284889578819275
Mean difference for top 10 list 17: 0.6864108443260193
Mean difference for top 10 list 18: 0.6349169611930847
Mean difference for top 10 list 19: 0.6554760932922363
Mean difference for top 10 list 20: 0.625062108039856
Mean difference for top 10 list 21: 0.6656431555747986
Mean difference for top 10 list 22: 0.7009323239326477
Mean difference for top 10 list 23: 0.6260776519775391
Mean difference for top 10 list 24: 0.6246715784072876
Mean difference for top 10 list 25: 0.6307580471038818
Mean difference for top 10 list 26: 0.6367862224578857
Mean difference for top 10 list 27: 0.5828424692153931
Mean difference for top 10 list 28: 0.632544755935669
Mean difference for top 10 list 29: 0.6225195527076721
Mean difference for top 10 list 30: 0.6393646001815796
Mean difference for top 10 list 31: 0.6228621006011963
Mean difference for top 10 list 32: 0.6407506465911865
Mean difference for top 10 list 33: 0.6054199934005737
Mean difference for top 10 list 34: 0.5688157081604004
Mean difference for top 10 list 35: 0.664237916469574
Mean difference for top 10 list 36: 0.648007869720459
Mean difference for top 10 list 37: 0.6090980768203735
Mean difference for top 10 list 38: 0.645548939704895
Mean difference for top 10 list 39: 0.6791596412658691
Mean difference for top 10 list 40: 0.6543868780136108
Mean difference for top 10 list 41: 0.6286242008209229
Mean difference for top 10 list 42: 0.6226542592048645
Mean difference for top 10 list 43: 0.641817569732666
Mean difference for top 10 list 44: 0.5907852053642273
Mean difference for top 10 list 45: 0.632124662399292
Mean difference for top 10 list 46: 0.6205264329910278
Mean difference for top 10 list 47: 0.6639276742935181
Mean difference for top 10 list 48: 0.6545956134796143
```

```
Mean difference for top 10 list 49: 0.6496999859809875
Mean difference for top 10 list 50: 0.6442272067070007
Mean difference for top 10 list 51: 0.5367512106895447
Mean difference for top 10 list 52: 0.6305505037307739
Mean difference for top 10 list 53: 0.65578293800354
Mean difference for top 10 list 54: 0.6623494625091553
Mean difference for top 10 list 55: 0.6237037777900696
Mean difference for top 10 list 56: 0.6435391306877136
Mean difference for top 10 list 57: 0.6461787223815918
Mean difference for top 10 list 58: 0.5904294848442078
```

```
print(len(mean_differences)) #Just checking  it matches the users
```

```
671
```

```
#Calaculating the mean difference for the whole dataset (for every top_10)
mean_difference_whole_dataset = sum(mean_differences) / len(mean_differences)
print("Mean difference for the whole dataset:", mean_difference_whole_dataset)
```

```
Mean difference for the whole dataset: 0.638870567038944
```

Conclusion for Diversity:

The mean difference is calculated as 0.63, indicating a moderate level of diversity, on average, among the movies recommended to users. The list of mean differences for each user shows little variation in the diversity measure. This observation suggests that the embeddings generated by the model may contribute to the consistent level of diversity across recommendations. However, if a higher or lower variety of recommendations is desired, adjusting the model parameters could be explored.

Novelty

In the process of calculating the first steps for Novelty, I have experimented with various approaches. Initially, I attempted to use a previously mentioned function and iterated over the dataset to calculate the mean. However, this approach resulted in errors. To better understand the dataset and debug the issues, I conducted several tests.

The code I have implemented for Novelty is a combination of techniques from previous notebooks used in STEM and NLP. As part of this, I utilized a for loop to create a list containing the top 10 recommendations per user. This approach allowed me to extract the necessary information for further analysis.

1. Calculate the 10 top movies per ser

```
top10_movies_per_user_np = np.array(top10_movies_per_user)
```

```
top10_movies_per_user_np.shape
```

```
(649, 10)
```

```
movie_data.columns
```

```
Index(['movieId', 'title', 'genres'], dtype='object')
```

2. For each top 10, get the mean rating for each film (based on the original MovieLens Small dataset (df = pd.read_csv("ml-latest-small/ratings.csv")).

```
df['rating'].isnull().any()
```

```
False
```

```
df['rating'].dtype
```

```
dtype('float64')
```

```
df['rating'].isnull().any()
```

```
False
```

I opted to create a 'for loop' to iterate through the dataset. However, I encountered errors in my code, resulting in "Mean Novelty: nan" as the output. I consulted ChatGPT for debugging. The suggested solution from ChatGPT included additional checks to identify and resolve the errors in my code.(I have indicated/cited the assistance provided by ChatGPT, and I hope that incorporating the recommended code is allowed.)

Below is my initial attempt at using a for loop to iterate through the dataset, which I later modified based on the recommendations from ChatGPT to aid in debugging and resolving the issues.

```
# Testing the code (which I had to then enhance in the following tab)  #
#novelty_per_user = []

#for user_movies in top10_movies_per_user:

   # movie_ratings = df[df['movieId'].isin(user_movies)]['rating']
   # mean_rating = movie_ratings.mean()
   # novelty_per_user.append(mean_rating)


# Calculating  the mean novelty for the whole dataset:
#mean_novelty = np.mean(novelty_per_user)

# Printing the mean novelty for the whoe dataset:
#print("Mean Novelty:", mean_novelty)


novelty_per_user = []


for user_movies in top10_movies_per_user:  # use a for loop to iterate through the previously defined list of the top10_movies
    # Get the ratings for each movie in the user's top 10 movies
    movie_ratings = df[df['movieId'].isin(user_movies)]['rating']    # code from notebook Week 4.2


    valid_ratings = movie_ratings.dropna()   # I was getting errors in my code, so I had to include this as a suggestion from

    if not valid_ratings.empty:  #Checking  if there are valid ratings

        mean_rating = valid_ratings.mean()    #Calculating the mean rating for the user's of the top10 movies
        novelty_per_user.append(mean_rating)  # Append the mean rating to the list of novelty per user

# Calculate the mean novelty for the whole dataset
if novelty_per_user:  # Check if the list is not empty
    mean_novelty = np.nanmean(novelty_per_user)
else:
    mean_novelty = 0.0  # Assign a default value or handle the case when there are no valid ratings

# Print the mean novelty
print("Mean Novelty:", mean_novelty)
```

```
Mean Novelty: 0.0
```

Conclusion

If the code is correct, a novelty of 0 would mean that the films that have been recommended are highly rated by most users. Consequently, this would indicate that the recommendations consist of popular and well-known films, suggesting that it is not providing new or unfamiliar films to users. Instead, it aligns with the existing preferences and tastes of the users, offering suggestions that are already well-liked within the users.

Task 2 ** **

Using a dimensionality reduction approach (PCA? TSNE?), plot the top 30 best rated films on a 2-D graph based on their movie embeddings. Label each point with the title.

There is infact ~400 films that have an average rating of 5 (because some films have only 1 rating). Can you adjust or filter for this?

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

top30_best_rated = df.sort_values(by='rating', ascending=False).head(30)    # Geting  the top30 best rated movies. Code from h
print(top30_best_rated)
num_films = len(top30_best_rated)  # Number of films in the top 30 best rated list
movie_indexes = top30_best_rated['movie_index'].tolist()   # I had to debug this with the help of ChatGPT

movie_30_embeddings = model.movie_embedding.weight.data[movie_indexes].numpy()    # Getting the movie embedding and turning tc

pca = PCA(n_components=2)   # Dimensionality reduction using PCA. Code from Week 3.1
reduced_embeddings = pca.fit_transform(movie_30_embeddings)

plt.figure(figsize=(8, 8))
```

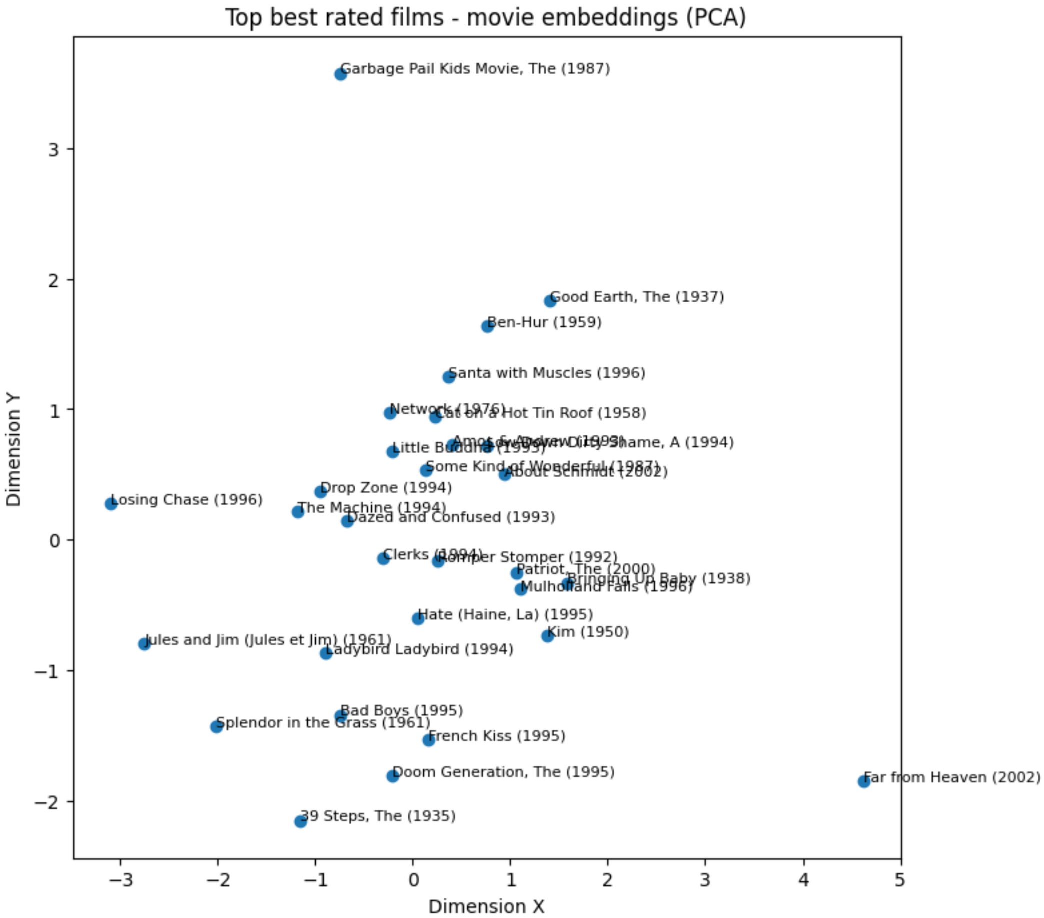
```
plt.scatter(reduced_embeddings[:, 0], reduced_embeddings[:, 1])

#  Adding titles to each scatter plot element https://stackoverflow.com/questions/14432557/scatter-plot-with-different-text-at
for i, title in enumerate(movie_data.loc[movie_indexes, 'title']):
    plt.text(reduced_embeddings[i, 0], reduced_embeddings[i, 1], title, fontsize=8)

plt.xlabel('Dimension X')
plt.ylabel('Dimension Y')
plt.title('Top best rated films - movie embeddings (PCA)')
plt.show()
```

	userId	movieId	rating	timestamp	user_index	movie_index
33889	242	2929	1.0	956687566	241	2667
36867	265	1233	1.0	960056214	264	775
46251	337	1356	1.0	1447176421	336	208
14749	95	3948	1.0	1025556197	94	349
14747	95	3917	1.0	1019023102	94	3002
63929	460	2020	1.0	1072837101	459	235
14741	95	3793	1.0	1018816171	94	598
77745	537	1269	1.0	879503289	536	2189
77744	537	1267	1.0	879503075	536	788
14737	95	3702	1.0	1016316990	94	2751
77742	537	1265	1.0	879521068	536	196
36869	265	1237	1.0	960056005	264	1957
14733	95	3617	1.0	1018816073	94	1062
14732	95	3578	1.0	1018815804	94	468
14730	95	3510	1.0	1018815804	94	124
14729	95	3499	1.0	1019022937	94	1044
14728	95	3481	1.0	1016316859	94	1041
36874	265	1247	1.0	960056180	264	329
77748	537	1273	1.0	879502758	536	4392
88494	587	5013	1.0	1160393838	586	1226
46240	337	589	1.0	1447176450	336	89
46233	336	4406	1.0	995826790	335	2798
14772	95	4369	1.0	1019022782	94	1166
14771	95	4351	1.0	1016317519	94	1163
14770	95	4321	1.0	1025556197	94	629
36878	265	1254	1.0	960056098	264	783
46238	337	329	1.0	1447176408	336	145
14764	95	4140	1.0	1019022672	94	4323
77749	537	1276	1.0	879502488	536	390
77751	537	1282	1.0	879502925	536	199

30



Thank you very much for your great lectures this term! Sorry I couldnt attend (I was teaching the exact same days) - but I watched all your lectures online. :)

0s completed at 8:50 PM