Collaborative Filtering for Movie Recommendations

20150887 이동준

20201219 김인서

Collaborative Filtering

- 사용자, 제품 간의 유사성을 확인하고 이를 바탕으로 사용자 취향에 맞는 아이템을 추천
- 사용자의 과거 경험과 행동 방식에 의존하여 추천하는 시스템

목표

- Movielens 데이터셋을 이용
- 사용자가 시청하지 않은 영화의 평점을 예측하여 높은 평점으로 예측된 영화를 추천하는 것

0. 시작하기

```python
import pandas as pd
from pathlib import Path
import matplotlib.pyplot as plt
import numpy as np
from zipfile import ZipFile

import keras
from keras import layers
from keras import ops
```

1. 데이터 불러오기, 전처리 적용하기

```
# Download the actual data from http://files.grouplens.org/datasets/moviel
# Use the ratings.csv file
movielens_data_file_url = (
    "http://files.grouplens.org/datasets/movielens/ml-latest-small.zip"
)
movielens_zipped_file = keras.utils.get_file(
    "ml-latest-small.zip", movielens_data_file_url, extract=False
)
keras_datasets_path = Path(movielens_zipped_file).parents[0]
movielens_dir = keras_datasets_path / "ml-latest-small"

# Only extract the data the first time the script is run.
if not movielens_dir.exists():
    with ZipFile(movielens_zipped_file, "r") as zip:
        # Extract files
        print("Extracting all the files now...")
        zip.extractall(path=keras_datasets_path)
        print("Done!")

ratings_file = movielens_dir / "ratings.csv"
df = pd.read_csv(ratings_file)
```

```
Downloading data from http://files.grouplens.org/datasets/movielens/ml-latest-small.zip
978202/978202 ━━━━━━━━━━━━━━━━━━━━ 1s 1us/step
Extracting all the files now...
Done!
```

- links
- movies
- ratings
- README
- tags

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
...	...	...	...	...
100831	610	166534	4.0	1493848402
100832	610	168248	5.0	1493850091
100833	610	168250	5.0	1494273047
100834	610	168252	5.0	1493846352
100835	610	170875	3.0	1493846415

100836 rows × 4 columns

1. 데이터 불러오기, 전처리 적용하기

```python
user_ids = df["userId"].unique().tolist()
user2user_encoded = {x: i for i, x in enumerate(user_ids)}
userencoded2user = {i: x for i, x in enumerate(user_ids)}
movie_ids = df["movieId"].unique().tolist()
movie2movie_encoded = {x: i for i, x in enumerate(movie_ids)}
movie_encoded2movie = {i: x for i, x in enumerate(movie_ids)}
df["user"] = df["userId"].map(user2user_encoded)
df["movie"] = df["movieId"].map(movie2movie_encoded)

num_users = len(user2user_encoded)
num_movies = len(movie_encoded2movie)
df["rating"] = df["rating"].values.astype(np.float32)
# min and max ratings will be used to normalize the ratings later
min_rating = min(df["rating"])
max_rating = max(df["rating"])

print(
    "Number of users: {}, Number of Movies: {}, Min rating: {}, Max rating: {}".format(
        num_users, num_movies, min_rating, max_rating
    )
)
```

```
Number of users: 610, Number of Movies: 9724, Min rating: 0.5, Max rating: 5.0
```

2. train, validation 데이터 준비

```
df = df.sample(frac=1, random_state=42)
x = df[["user", "movie"]].values
# Normalize the targets between 0 and 1. Makes it easy to train.
y = df["rating"].apply(lambda x: (x - min_rating) / (max_rating - min_rating)).values
# Assuming training on 90% of the data and validating on 10%.
train_indices = int(0.9 * df.shape[0])
x_train, x_val, y_train, y_val = (
    x[:train_indices],
    x[train_indices:],
    y[:train_indices],
    y[train_indices:],
)
```

```
1 x_train
```

```
array([[ 431, 4730],
       [ 287,  474],
       [ 598, 2631],
       ...,
       [ 589, 5054],
       [ 135,  636],
       [ 274, 3747]])
```

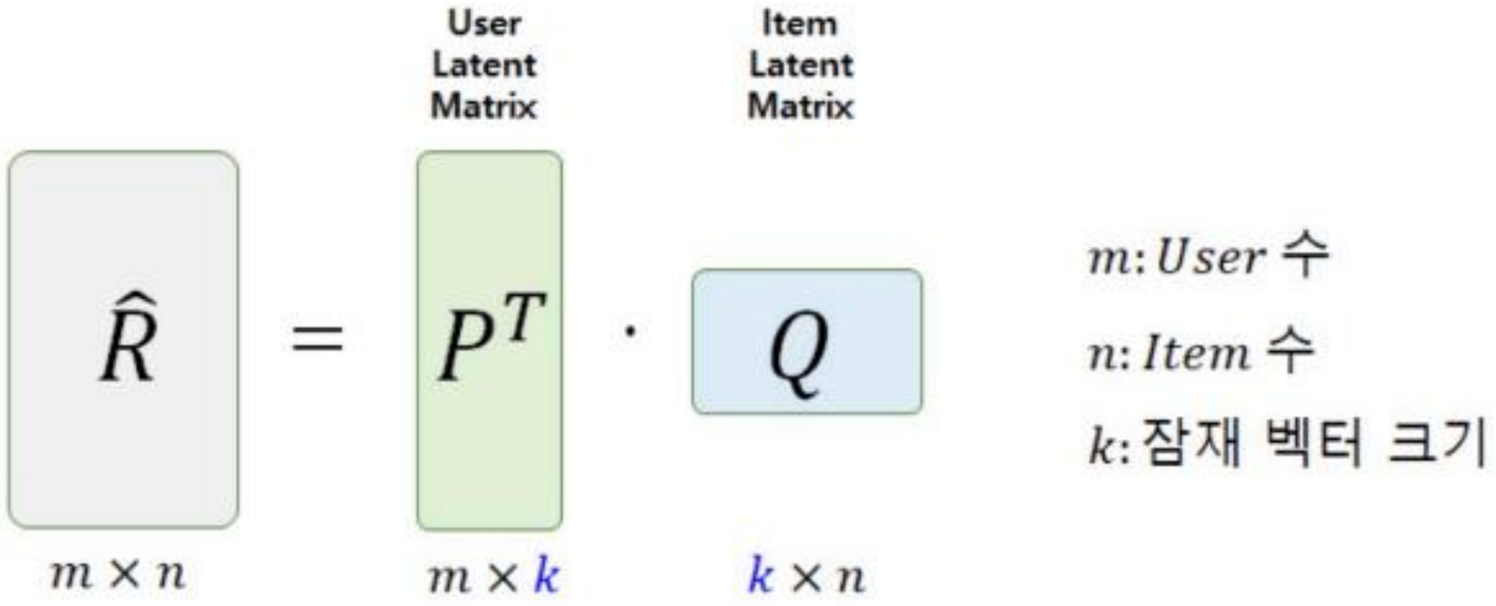
```
1 y_train
```

```
array([0.88888889, 0.55555556, 0.55555556, ..., 0.55555556, 1.
       0.11111111])
```

3. 모델 생성하기

Matrix Factorization

P : 각 사용자의 특성을 나타내는 k개의 요인 값으로 이루어진 행렬
Q : 각 아이템의 특성을 나타내는 k개의 요인 값으로 이루어진 행렬
$\hat{R}$: 평점 예측치



[출처: [추천시스템] 03. Matrix Factorization (velog.io)]()

3. 모델 생성하기

- 잠재 요인 k 개수 설정
- P, Q 행렬 초기화
- 예측 평점 $\hat{R}$ 계산
- 실제값 R과 예측값 $\hat{R}$ 간 오차 계산 및 P, Q 수정

3. 모델 생성하기

```
EMBEDDING_SIZE = 50


class RecommenderNet(keras.Model):
    def __init__(self, num_users, num_movies, embedding_size, **kwargs):
        super().__init__(**kwargs)
        self.num_users = num_users
        self.num_movies = num_movies
        self.embedding_size = embedding_size
        self.user_embedding = layers.Embedding(
            num_users,
            embedding_size,
            embeddings_initializer="he_normal",
            embeddings_regularizer=keras.regularizers.l2(1e-6),
        )
        self.user_bias = layers.Embedding(num_users, 1)
        self.movie_embedding = layers.Embedding(
            num_movies,
            embedding_size,
            embeddings_initializer="he_normal",
            embeddings_regularizer=keras.regularizers.l2(1e-6),
        )
        self.movie_bias = layers.Embedding(num_movies, 1)
```

He_normal

$$W \sim N(0, Var(W))$$

$$Var(W) = \sqrt{\frac{2}{n_{in}}}$$

n_{in}: 이전 layer의 노드 수

[머신러닝 - Weight Initialization(가중치 초기화) 중요성&문제점&종류 : 네이버 블로그 (naver.com)]

3. 모델 생성하기

```
def call(self, inputs):
    user_vector = self.user_embedding(inputs[:, 0])
    user_bias = self.user_bias(inputs[:, 0])
    movie_vector = self.movie_embedding(inputs[:, 1])
    movie_bias = self.movie_bias(inputs[:, 1])
    dot_user_movie = ops.tensordot(user_vector, movie_vector, 2)
    # Add all the components (including bias)
    x = dot_user_movie + user_bias + movie_bias
    # The sigmoid activation forces the rating to between 0 and 1
    return ops.nn.sigmoid(x)
```

```
model = RecommenderNet(num_users, num_movies, EMBEDDING_SIZE)
model.compile(
    loss=keras.losses.BinaryCrossentropy(),
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
)
```

4. 분리된 데이터를 기반으로 모델 훈련

```python
history = model.fit(
    x=x_train,
    y=y_train,
    batch_size=64,
    epochs=5,
    verbose=1,
    validation_data=(x_val, y_val),
)
```

```
Epoch 1/5
1418/1418 ──────────────────────────── 20s 13ms/step - loss: 0.6570 - val_loss: 0.6203
Epoch 2/5
1418/1418 ──────────────────────────── 23s 15ms/step - loss: 0.6157 - val_loss: 0.6171
Epoch 3/5
1418/1418 ──────────────────────────── 34s 10ms/step - loss: 0.6088 - val_loss: 0.6142
Epoch 4/5
1418/1418 ──────────────────────────── 19s 9ms/step - loss: 0.6070 - val_loss: 0.6146
Epoch 5/5
1418/1418 ──────────────────────────── 20s 9ms/step - loss: 0.6067 - val_loss: 0.6126
```

```
1 model.summary()
```

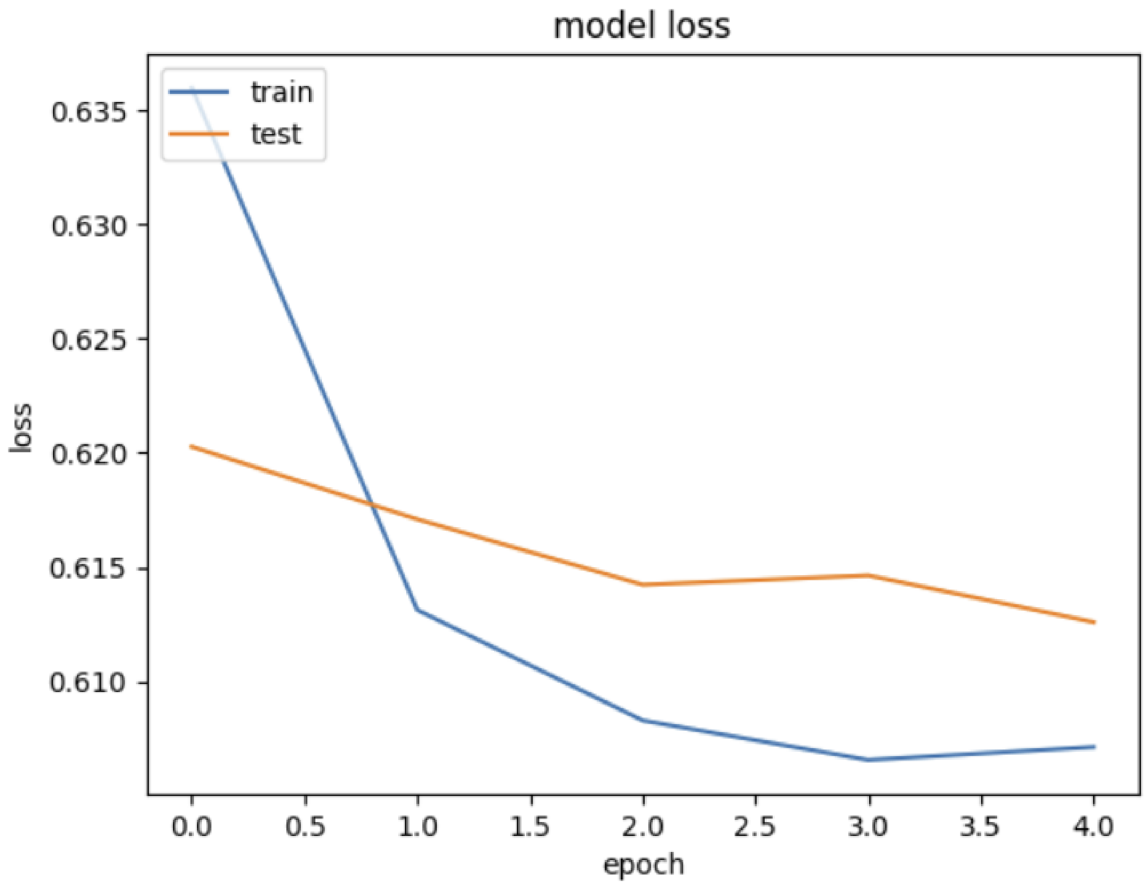
Model: "recommender_net"

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	30,500
embedding_1 (Embedding)	?	610
embedding_2 (Embedding)	?	486,200
embedding_3 (Embedding)	?	9,724

Total params: 1,581,104 (6.03 MB)
Trainable params: 527,034 (2.01 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 1,054,070 (4.02 MB)

5. Train Validation loss 그리기

```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.title("model loss")
plt.ylabel("loss")
plt.xlabel("epoch")
plt.legend(["train", "test"], loc="upper left")
plt.show()
```



6. User에게 상위 10개 추천 영화 제시

```
movie_df = pd.read_csv(movielens_dir / "movies.csv")

# Let us get a user and see the top recommendations.
user_id = df.userId.sample(1).iloc[0]
movies_watched_by_user = df[df.userId == user_id]
movies_not_watched = movie_df[
    ~movie_df["movieId"].isin(movies_watched_by_user.movieId.values)
]["movieId"]
movies_not_watched = list(
    set(movies_not_watched).intersection(set(movie2movie_encoded.keys()))
)
movies_not_watched = [[movie2movie_encoded.get(x)] for x in movies_not_watched]
user_encoder = user2user_encoded.get(user_id)
user_movie_array = np.hstack(
    ([[user_encoder]] * len(movies_not_watched), movies_not_watched)
)
ratings = model.predict(user_movie_array).flatten()
top_ratings_indices = ratings.argsort()[-10:][::-1]
recommended_movie_ids = [
    movie_encoded2movie.get(movies_not_watched[x][0]) for x in top_ratings_indices
]
```

```
print("Showing recommendations for user: {}".format(user_id))
print("====" * 9)
print("Movies with high ratings from user")
print("——" * 8)
top_movies_user = (
    movies_watched_by_user.sort_values(by="rating", ascending=False)
    .head(5)
    .movieId.values
)
movie_df_rows = movie_df[movie_df["movieId"].isin(top_movies_user)]
for row in movie_df_rows.itertuples():
    print(row.title, ":", row.genres)

print("——" * 8)
print("Top 10 movie recommendations")
print("——" * 8)
recommended_movies = movie_df[movie_df["movieId"].isin(recommended_movie_ids)]
for row in recommended_movies.itertuples():
    print(row.title, ":", row.genres)
```

6. User에게 상위 10개 추천 영화 제시

```
262/262 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Showing recommendations for user: 274
====================================
Movies with high ratings from user
--------------------------------
Pulp Fiction (1994) : Comedy|Crime|Drama|Thriller
American Beauty (1999) : Drama|Romance
Fight Club (1999) : Action|Crime|Drama|Thriller
Eternal Sunshine of the Spotless Mind (2004) : Drama|Romance|Sci-Fi
Inglourious Basterds (2009) : Action|Drama|War
--------------------------------
Top 10 movie recommendations
--------------------------------
Ghost in the Shell (Kôkaku kidôtai) (1995) : Animation|Sci-Fi
Philadelphia Story, The (1940) : Comedy|Drama|Romance
Wallace & Gromit: The Wrong Trousers (1993) : Animation|Children|Comedy|Crime
Cinema Paradiso (Nuovo cinema Paradiso) (1989) : Drama
Lawrence of Arabia (1962) : Adventure|Drama|War
To Kill a Mockingbird (1962) : Drama
Amadeus (1984) : Drama
Chinatown (1974) : Crime|Film-Noir|Mystery|Thriller
Manchurian Candidate, The (1962) : Crime|Thriller|War
Great Dictator, The (1940) : Comedy|Drama|War
```

```
1 movie_df
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
9737	193581	Black Butler: Book of the Atlantic (2017)	Action|Animation|Comedy|Fantasy
9738	193583	No Game No Life: Zero (2017)	Animation|Comedy|Fantasy
9739	193585	Flint (2017)	Drama
9740	193587	Bungo Stray Dogs: Dead Apple (2018)	Action|Animation
9741	193609	Andrew Dice Clay: Dice Rules (1991)	Comedy