Part 2: 🎬 Applying NMF to the Movie Ratings Data

In [103…

```
# !unzip ./data/movie_recommendation_files.zip -d ./data/
# !mv ./data/Files/* ./data/
# !rm -d ./data/Files
!find ./data/ -type f -mtime -30 -ls
```

```
       55     10 -rw-------   1 root     root         9367 Jun 23 16:21 ./data/submi
ssion.csv
       60   4601 -rw-------   1 root     root      4711088 Jun 23 17:14 ./data/movie
_recommendation_files.zip
       68    245 -rw-------   1 root     root       250402 Jun 23 17:12 ./data/movie
s.csv
       71   3386 -rw-------   1 root     root      3466730 Jun 23 17:12 ./data/test.
csv
       74   7898 -rw-------   1 root     root      8086756 Jun 23 17:12 ./data/trai
n.csv
       77    108 -rw-------   1 root     root       110238 Jun 23 17:13 ./data/user
s.csv
```

In [105…

```
train = pd.read_csv('./data/train.csv')
users = pd.read_csv('./data/users.csv')
movies = pd.read_csv('./data/movies.csv')
```

🕶️ Create the Utility Matrix

In [106…

```
# Get ID mappings
user_ids = sorted(train['uID'].unique())
movie_ids = sorted(train['mID'].unique())
user_map = {uid: i for i, uid in enumerate(user_ids)}
movie_map = {mid: i for i, mid in enumerate(movie_ids)}

# Shape
n_users = len(user_ids)
n_movies = len(movie_ids)

import numpy as np

# Create full utility matrix
R = np.zeros((n_users, n_movies))

for _, row in train.iterrows():
    u_idx = user_map[row['uID']]
    m_idx = movie_map[row['mID']]
    R[u_idx, m_idx] = row['rating']
```

🍏 Apply NMF to Predict Missing Ratings

In [107…

```python
nmf = NMF(
    n_components=20,
    max_iter=500,
    random_state=42069 # Nice # It never gets old...
    )
W = nmf.fit_transform(R)
H = nmf.components_
R_hat = np.dot(W, H)  # Reconstructed ratings matrix
```

💝 Evaluate on Test Data

In [110…

```python
from sklearn.metrics import mean_squared_error

test = pd.read_csv('./data/test.csv')

y_true = []
y_pred = []

for _, row in test.iterrows():
    u_idx = user_map.get(row['uID'])
    m_idx = movie_map.get(row['mID'])

    if u_idx is not None and m_idx is not None:
        y_true.append(row['rating'])
        y_pred.append(R_hat[u_idx, m_idx])

# Compute RMSE
from math import sqrt
rmse = sqrt(mean_squared_error(y_true, y_pred))
print(f"NMF RMSE on test set: {rmse:.4f}")
```

```
NMF RMSE on test set: 2.8609
```

🔨 Tinkering

In [114…

```python
from sklearn.metrics import mean_squared_error
from math import sqrt
import pandas as pd

# Compute global average rating from training data
global_avg = train['rating'].mean()
print(f"Global average rating: {global_avg:.4f}")

# Predict global mean for all test entries
y_true = test['rating'].tolist()
y_pred = [global_avg] * len(y_true)

# Compute RMSE
baseline_rmse = sqrt(mean_squared_error(y_true, y_pred))
print(f"Global mean baseline RMSE: {baseline_rmse:.4f}")
```

```
Global average rating: 3.5816
Global mean baseline RMSE: 1.1162
```

🍮 Discuss the Results

🧩 NMF vs. Baseline Comparison

The `sklearn` NMF model achieved an RMSE of **2.86**, which is significantly worse than a naive baseline model that predicts the global average rating for every user-item pair.

Model	RMSE
Global Mean Baseline	1.1162
NMF (scikit-learn, k=20)	2.8609

My result highlights a major limitation of the NMF implemented in `sklearn`. NMF assumes all unobserved ratings are `0`. This distorts predicted values.

By populating with the `global_avg`, the baseline model is seemingly unaffected by missing data making it a stronger default approach.

To improve performance, it was recommended to me to use libraries like `Surprise`, which properly handle sparse rating data with implicit masking and bias modeling. Unfortunately, that would have required a downgrade of my `numpy` version. I opted not to do that because I am a chicken.