DATA SOCIETY:

Sentiment Analysis & Recommender Sytems - Part 4

One should look for what is and not what he thinks should be. (Albert Einstein)

Warm up

- Before we start, check out this article on recommender systems and their uses: (An Introduction to Recommender Systems ***+9 Easy Examples) | Iterators***
- In the chat, answer the following questions:
  - What examples did you find the most interesting and relevant to you?
  - How do you plan use recommender systems in your work? What goals would you like to achieve?
  - Can you think of any challenges you might encounter with recommender systems?

Welcome back!

- In the last class, we covered:
  - Recommendation engines and their applications
  - Content-based recommender systems
  - Generating recommendations from the content-based recommender system
- Today, we will learn about:
  - Collaborative filtering recommender system and its types
  - Building item-based collaborative filtering algorithms
  - Model-based collaborative filtering recommender system
  - Evaluating a model using performance metrics

Module completion checklist

Objective	Complete
Describe the collaborative filtering recommender system and its types	
Build an item-based collaborative filtering algorithm	
Summarize a model-based collaborative filtering recommender system and the concept of SVD	
Implement a model-based collaborative algorithm and generate predictions	
Evaluate the model using performance metrics	

Collaborative filtering algorithms

- Content-based recommenders recommend movies from a **particular genre only**
- They don't capture the **taste and preference of user across different genres**
- Collaborative filtering algorithm recommends items across different genres
- It is entirely based on the **past behavior and not on the context**
- It analyzes how **similar the taste of one user is to another**
- For example, if user `X` likes movies 1, 2, 3 and user `Y` likes movies 2, 3, 4, then they have similar taste and `X` should like movie 4 and `Y` should like movie 1

Collaborative filtering algorithm types

- Collaborative filtering can be divided into two types
  - Model-based collaborative filtering, which we will see later today
  - Memory-based collaborative filtering, which we'll cover now
- Memory-based is further divided into:
  - User collaborative filtering, which is based on similarity between the **users**
  - Item collaborative filtering, which is based on the similarity between the **items**

User-based collaborative filtering

- It is based on the idea that similar people will have **similar taste** and would have rated the movies **identically**
- For any given user, this algorithm tries to find similar users and recommends movies which similar users have seen
- This is like **finding a nearest neighbor** to user `A` with a similarity function to measure the distance
- We use the distance to predict the rating that user `A` will give to all items the k neighbors have seen, but `A` has not
- We fetch an item `j` with best predicted rating
- We create a user-item matrix predicting the ratings on items the active user has not seen, based on other similar users

User-based collaborative filtering

- Pros
  - Easy to implement
  - Context independent
  - Compared to content-based, it is more accurate
- Cons
  - **Sparsity:** the percentage of people who rate items is really low
  - **Scalability:** the more K neighbors we consider, the better the classification
  - **Cold start:** new users will have no information about them that we can use

Item-based collaborative filtering

- Item-based is similar to user-based, but it finds the **similarity between items and recommends similar items to the users**
- It finds the similarity between the items first
- It recommends the items similar to the items the user has already seen

Loading the packages

- Let's make sure we have the packages we will need for the data cleaning, plotting and building the recommender system:

```
import os
import pickle
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import wordcloud
from wordcloud import WordCloud, STOPWORDS
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import pairwise_distances
from sklearn.metrics import mean_squared_error
```

Loading the packages

```
from math import sqrt
from scipy.sparse.linalg import svds
from surprise import Reader
from surprise import Dataset
from surprise import SVD
from surprise.model_selection import cross_validate
```

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- We will use the `pathlib` library
- Let the `main_dir` be the variable corresponding to your `skillsoft-sentiment-analysis-2021` folder
- Let `data_dir` be the variable corresponding to your `data` folder

```
# Set 'main_dir' to location of the project folder
from pathlib import Path
home_dir = Path(".").resolve()
main_dir = home_dir.parent
```

```
data_dir = str(main_dir) + "/data"
```

- We'll be using this variable to load the data present in the data folder!

Load the subset of data

- We will use a subset of movies and ratings to implement the collaborative filtering

```
# Read in the datasets.
rating_subset = pd.read_csv(data_dir+'/ratings-subset.csv')
movies_subset = pd.read_csv(data_dir+'/movies-subset.csv')
```

```
# Select only movie ID and title from movies dataset.
movies_subset = movies_subset[['movieId', 'title']]
# Merge both ratings and movies dataframes.
rating_df = pd.merge(movies_subset, rating_subset)
# View the summary and head of the merged dataframe.
print(rating_df.head())
```

```
   movieId             title  userId  rating   timestamp
0        1  Toy Story (1995)       1     4.0   964982703
1        1  Toy Story (1995)       5     4.0   847434962
2        1  Toy Story (1995)       7     4.5  1106635946
3        1  Toy Story (1995)      15     2.5  1510577970
4        1  Toy Story (1995)      17     4.5  1305696483
```

Item-based recommender implementation

- Let's transform our data with users as rows and movie title as columns and the rating as the value

```
userRating = rating_df.pivot_table(index = ['userId'],
                                   columns = ['title'], values = 'rating')

print(userRating.head())
```

```
title   '71 (2014)  ...  À nous la liberté (Freedom for Us) (1931)
userId              ...
1              NaN  ...                                        NaN
2              NaN  ...                                        NaN
3              NaN  ...                                        NaN
4              NaN  ...                                        NaN
5              NaN  ...                                        NaN

[5 rows x 9719 columns]
```

Item correlation matrix

- Let’s find the similarity between items (movies) by using Pearson correlation
- The Pearson correlation measures the linear correlation between two variables and has a value between **+1** and **-1**

```
# corrMatrix = userRating.corr(method = 'pearson', min_periods = 100)
# corrMatrix.to_csv('corrMatrix.csv', index = True, encoding = 'utf-8')
```

```
                                        title  ...  À nous la liberté (Freedom for Us) (1931)
0                                  '71 (2014)  ...                                        NaN
1  'Hellboy': The Seeds of Creation (2004)  ...                                        NaN
2                   'Round Midnight (1986)  ...                                        NaN
3                      'Salem's Lot (2004)  ...                                        NaN
4                 'Til There Was You (1997)  ...                                        NaN

[5 rows x 9720 columns]
```

```
corrMatrix = pd.read_csv(data_dir+ '/corrMatrix.csv')
print(corrMatrix.head())

corrMatrix = corrMatrix.set_index('title')
```

Suggest movies to user

- Now we have a correlation matrix which shows how much each movie is similar to each other movie
- If the similarity value is closer to 1, the more similar the two movies are
- Let's say we want to suggest new movies to user `A`, the item-based filtering works as follows:

Algorithm

- Fetch all the movies (list `Z`) the user `A` has already rated
- For each movie `i` in list `Z`
    - Fetch the row for movie `i` from similarity matrix as list `X`
    - Multiply the rating to the similarity score for each item in list `X`
- Group by `movie_id`
- Fetch top `(rating * similarity_score)` value
- Give the result

Suggest movies to user

- Let’s say we want to recommend movies to user 26
- The index value will be 25 for that person

```
user_corr = pd.Series()
```

```
/opt/conda/envs/skillsoft-sentiment-analysis-2021/bin/python:1: DeprecationWarning: The
default dtype for empty Series will be 'object' instead of 'float64' in a future version.
Specify a dtype explicitly to silence this warning.
```

```
user_id = 25

# Create a list of all films with all correlations multiplied by the rating.
for film in userRating.iloc[user_id].dropna().index:
    corr_list = corrMatrix[film].dropna() * userRating.iloc[user_id][film]
    user_corr = user_corr.append(corr_list)

# Group by movie ID and sum the ratings to remove duplicates.
user_corr = user_corr.groupby(user_corr.index).sum()
```

Suggest movies to user

- Now, we can suggest movies to the user after removing the movies they have already seen and are similar to them

```
# Create a list of movies the user has already seen and remove them.
title_list = []

for i in range(len(userRating.iloc[user_id].dropna().index)):
    if userRating.iloc[user_id].dropna().index[i] in user_corr:
        title_list.append(userRating.iloc[user_id].dropna().index[i])
    else:
        pass
user_corr = user_corr.drop(title_list)
```

Suggest movies to user

```
print('Hi! Based on the films that you have seen,
you might like: \n')
```

```
Hi! Based on the films that you have seen,
you might like:
```

```
for i in userRating.iloc[user_id].dropna().index:
    print(i)
```

```
Ace Ventura: Pet Detective (1994)
Apollo 13 (1995)
Babe (1995)
Batman (1989)
Batman Forever (1995)
Clear and Present Danger (1994)
Cliffhanger (1993)
Die Hard: With a Vengeance (1995)
Disclosure (1994)
Firm, The (1993)
Forrest Gump (1994)
Fugitive, The (1993)
GoldenEye (1995)
Natural Born Killers (1994)
Net, The (1995)
Pulp Fiction (1994)
Quiz Show (1994)
Seven (a.k.a. Se7en) (1995)
Silence of the Lambs, The (1991)
```

```
# Suggest the top 10 movies.
print('\n I would suggest that you watch: \n')
```

```

 I would suggest that you watch:
```

```
for i in user_corr.sort_values(ascending =
False).index[:10]:
    print(i)
```

```
Jurassic Park (1993)
Braveheart (1995)
Aladdin (1992)
Terminator 2: Judgment Day (1991)
Speed (1994)
Fight Club (1999)
Independence Day (a.k.a. ID4) (1996)
Dances with Wolves (1990)
Mask, The (1994)
Matrix, The (1999)
```

Knowledge check 1

Exercise 1

Module completion checklist

Objective	Complete
Describe the collaborative filtering recommender system and its types	✔
Build an item-based collaborative filtering algorithm	✔
Summarize a model-based collaborative filtering recommender system and the concept of SVD	
Implement a model-based collaborative algorithm and generate predictions	
Evaluate the model using performance metrics	

Model-based collaborative filtering algorithm

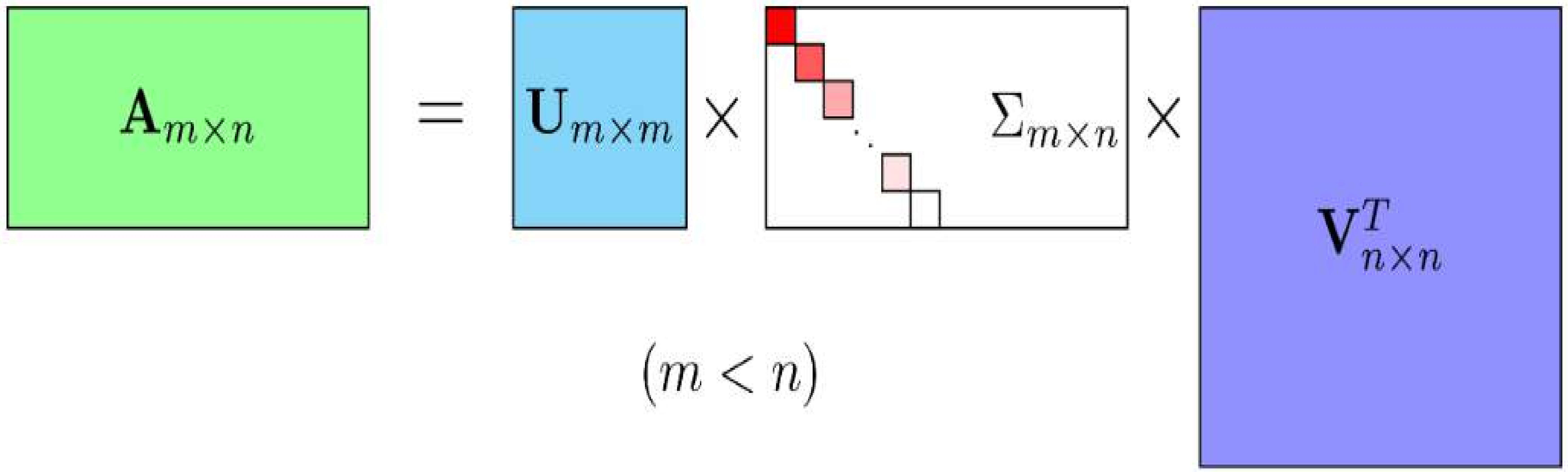
- Memory-based algorithms do not scale well to **massive datasets**
- Rating matrices can overfit to noisy representations
- We need to apply a **dimensionality reduction technique** to derive recommendations
- We do that by factorization, which can:
    - **Discover hidden patterns** and correlations
    - Remove **redundant and noisy features** that are not helpful
    - Interpret and visualize the data easily
- SVD (singular vector decomposition) is a powerful dimensionality reduction technique that is used in modern recommender systems

Model-based collaborative filtering algorithm

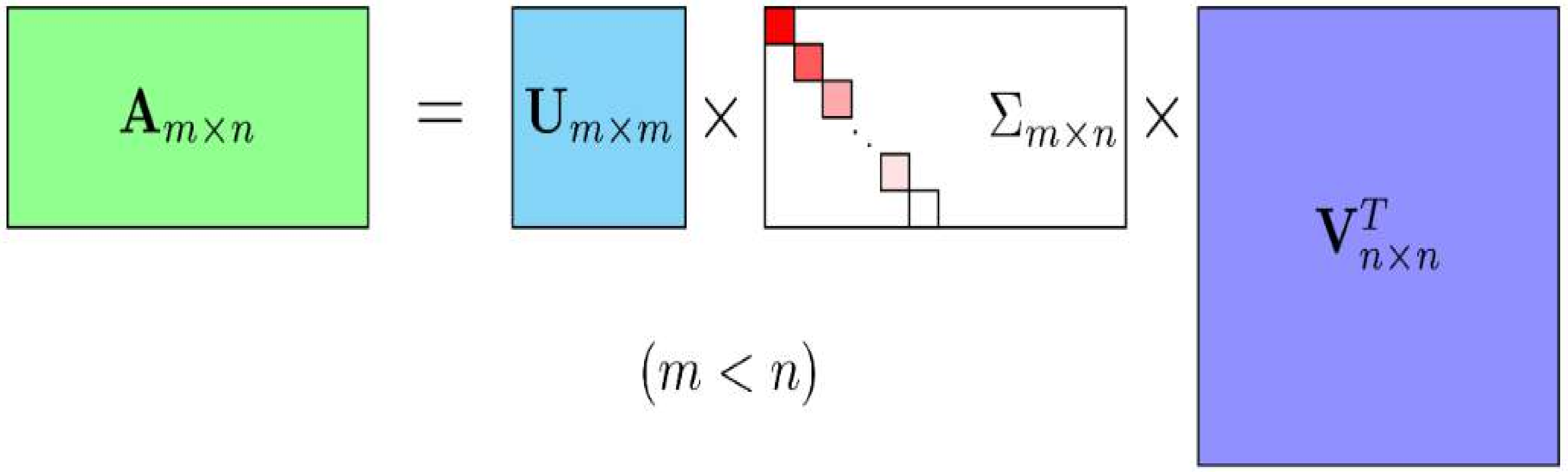
- This type of algorithm deals with **scalability and sparsity**
- It learns the preferences of users from known ratings
- When we have a highly sparse matrix with a lot of dimensions, we can restructure the user-item matrix by performing matrix factorization
- Use this matrix to approximate the original matrix and fill the missing entries in the original matrix

SVD algorithm

- Let's first understand how SVD works
- It decomposes the input matrix into 3 separate matrices and derives the **latent features** which fetches the underlying tastes and preferences vectors


$\mathbf{A}_{m\times n} = \mathbf{U}_{m\times m} \times \Sigma_{m\times n} \times \mathbf{V}^T_{n\times n}$
$(m < n)$

SVD algorithm


$\mathbf{A}_{m\times n} = \mathbf{U}_{m\times m} \times \Sigma_{m\times n} \times \mathbf{V}^T_{n\times n}$
$(m < n)$

- `A` is the input data matrix, which has the **users' ratings**
- `U` is the left singular vectors, which has the **user features matrix**
- `Sigma` is the **diagonal matrix** of singular values, which has the weights/strengths
- V^t is the right singular vectors, which has the movie features matrix

SVD algorithm

- `U` and V^t are **orthonormal** bases and represent different things - **orthonormal** means that the vectors are both orthogonal and are of a unit length (i.e., 1)
- `U` represents how much users like each feature
- V^t represents how relevant each feature is to each movie
- Take these matrices and keep top `k` features which can be thought of as the underlying tastes and preferences vectors

Find total users and movies

- Let's use the larger dataset here, since SVD can handle massive datasets, unlike collaborative filtering

```
# Reading the ratings file.
ratings = pd.read_csv(data_dir+ '/ratings.csv', sep='\t', encoding='latin-1',
usecols = ['user_id', 'movie_id', 'rating'])
```

```
# Find total number of unique users and movies.
n_users = ratings.user_id.unique().shape[0]
n_movies = ratings.movie_id.unique().shape[0]
print('Number of users = ' + str(n_users) + ' | Number of movies = ' + str(n_movies))
```

```
Number of users = 6040 | Number of movies = 3706
```

Data preparation for SVD

- Format the data structure to have one row per user and one column per movie
- Use `pivot_table`

```
Ratings = ratings.pivot(index = 'user_id',columns = 'movie_id',
values = 'rating').fillna(0)

print(Ratings.head())
```

```
movie_id  1     2     3     4     5     ...  3948  3949  3950  3951  3952
user_id                                 ...
1          5.0   0.0   0.0   0.0   0.0  ...   0.0   0.0   0.0   0.0   0.0
2          0.0   0.0   0.0   0.0   0.0  ...   0.0   0.0   0.0   0.0   0.0
3          0.0   0.0   0.0   0.0   0.0  ...   0.0   0.0   0.0   0.0   0.0
4          0.0   0.0   0.0   0.0   0.0  ...   0.0   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

[5 rows x 3706 columns]
```

De-normalize the data and check the amount of sparsity

```
# Normalize the data.
R = Ratings.to_numpy()

user_ratings_mean = np.mean(R, axis = 1)
Ratings_demeaned = R - user_ratings_mean.reshape(-1, 1)

# Check the percentage of sparsity.
sparsity = round(1.0 - len(ratings) / float(n_users * n_movies), 3)
print('The sparsity level of MovieLens1M dataset is ' +  str(sparsity * 100) + '%')
```

```
The sparsity level of MovieLens1M dataset is 95.5%
```

SVD implementation

- Decompose the input matrix with top 50 latent features

```
U, sigma, Vt = svds(Ratings_demeaned, k = 50)

# Convert the sigma matrix to the diagonal matrix form.
sigma = np.diag(sigma)
```

SVD implementation

- We have 3 matrices which can be used to make movie rating predictions for every user
- Add the user means back to get the actual star ratings prediction

```
all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt) + user_ratings_mean.reshape(-1, 1)
```

SVD implementation

- Let's return the list of movies the user has already rated

```
preds = pd.DataFrame(all_user_predicted_ratings, columns = Ratings.columns)
print(preds.head())
```

```
movie_id       1         2         3     ...      3950      3951      3952
0         4.288861  0.143055 -0.195080  ...  0.031912  0.050450  0.088910
1         0.744716  0.169659  0.335418  ... -0.101102 -0.054098 -0.140188
2         1.818824  0.456136  0.090978  ...  0.012345  0.015148 -0.109956
3         0.408057 -0.072960  0.039642  ... -0.026050  0.014841 -0.034224
4         1.574272  0.021239 -0.051300  ...  0.033830  0.125706  0.199244

[5 rows x 3706 columns]
```

SVD implementation

- Let’s write a function to return the movies with the highest predicted rating that the specified user has not already rated

```
def recommend_movies(predictions, userID, movies, original_ratings, num_recommendations):
    user_row_number = userID - 1 # User ID starts at 1, not 0
    sorted_user_predictions = preds.iloc[user_row_number].sort_values(ascending = False) # User ID 
starts at 1
    # Get the user's data and merge in the movie information.
    user_data = original_ratings[original_ratings.user_id == (userID)]
    user_full = (user_data.merge(movies, how = 'left', left_on = 'movie_id', right_on = 
'movie_id').sort_values(['rating'], ascending=False))
    print('User {0} has already rated {1} movies.'.format(userID, user_full.shape[0]))
    print('Recommending highest {0} predicted ratings movies not already 
rated.'.format(num_recommendations))
    # Recommend the highest predicted rating movies that the user hasn't seen yet.
    recommendations = (movies[~movies['movie_id'].isin(user_full['movie_id'])].
         merge(pd.DataFrame(sorted_user_predictions).reset_index(), how = 'left',
               left_on = 'movie_id',right_on = 'movie_id').
         rename(columns = {user_row_number: 'Predictions'}).
         sort_values('Predictions', ascending = False).iloc[:num_recommendations, :-1])
    return user_full, recommendations
```

Recommend movies using SVD

- Recommend 20 movies for user with ID 1310

```
# Reading movies file.
movies = pd.read_csv(data_dir+ '/movies.csv', sep='\t', encoding='latin-1',
usecols = ['movie_id', 'title', 'genres'])
```

```
already_rated, predictions = recommend_movies(preds, 1310, movies, ratings, 20)
```

```
User 1310 has already rated 24 movies.
Recommending highest 20 predicted ratings movies not already     rated.
```

Recommend movies using SVD

```python
# Top 20 movies that User 1310 has rated.
print(already_rated[['user_id', 'title']])
```

```
    user_id                                             title
5      1310                              Say Anything... (1989)
6      1310                          This Is My Father (1998)
7      1310                              Blood Simple (1984)
15     1310                          Good Will Hunting (1997)
1      1310                                    Gandhi (1982)
12     1310                          Fatal Attraction (1987)
11     1310                                 Cape Fear (1991)
20     1310                             Lethal Weapon (1987)
18     1310                               Parenthood (1989)
17     1310                                  Hoosiers (1986)
13     1310                       Places in the Heart (1984)
23     1310                E.T. the Extra-Terrestrial (1982)
10     1310  Star Wars: Episode V - The Empire Strikes Back...
9      1310                              My Left Foot (1989)
8      1310                             Prizzi's Honor (1985)
4      1310                            Broadcast News (1987)
3      1310                        Killing Fields, The (1984)
16     1310                     Brothers McMullen, The (1995)
```

Recommend movies using SVD

```python
# Top 20 movies that User 1310 hopefully will enjoy.
print(predictions[['movie_id', 'title']])
```

```
      movie_id                                            title
1618      1674                                   Witness (1985)
1880      1961                                  Rain Man (1988)
1187      1210   Star Wars: Episode VI - Return of the Jedi (1983)
1216      1242                                     Glory (1989)
1202      1225                                   Amadeus (1984)
1273      1302                           Field of Dreams (1989)
1220      1246                       Dead Poets Society (1989)
1881      1962                       Driving Miss Daisy (1989)
1877      1957                         Chariots of Fire (1981)
1938      2020                       Dangerous Liaisons (1988)
1233      1259                              Stand by Me (1986)
3011      3098                             Natural, The (1984)
2112      2194                         Untouchables, The (1987)
1876      1956                          Ordinary People (1980)
1268      1296                       Room with a View, A (1986)
2267      2352                            Big Chill, The (1983)
1278      1307                    When Harry Met Sally... (1989)
1165      1186                  Sex, Lies, and Videotape (1989)
```

Recommend movies using SVD

- It is a pretty good recommendation - although we did not use any genre for movies, the SVD picked up the **underlying tastes and preferences of the user**
- There are some comedy, drama, and romance movies, which are some genres in the user's top rated movies

Module completion checklist

Objective	Complete
Describe the collaborative filtering recommender system and its types	✔
Build an item-based collaborative filtering algorithm	✔
Summarize a model-based collaborative filtering recommender system and the concept of SVD	✔
Implement a model-based collaborative algorithm and generate predictions	✔
Evaluate the model using performance metrics	

Model evaluation

- How do we evaluate if our recommendation is good and accurate?
- We will use RMSE (root mean square error) to evaluate our SVD model
- By looking into the **RMSE** values, we can evaluate the recommender system
- **Root mean squared error (RMSE)** objectively assesses model performance and is a way to measure error of model while predicting quantitative data
- This is the formula used for RMSE:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

Model evaluation

- We will use the `surprise` package to implement SVD this time instead of manual implementation

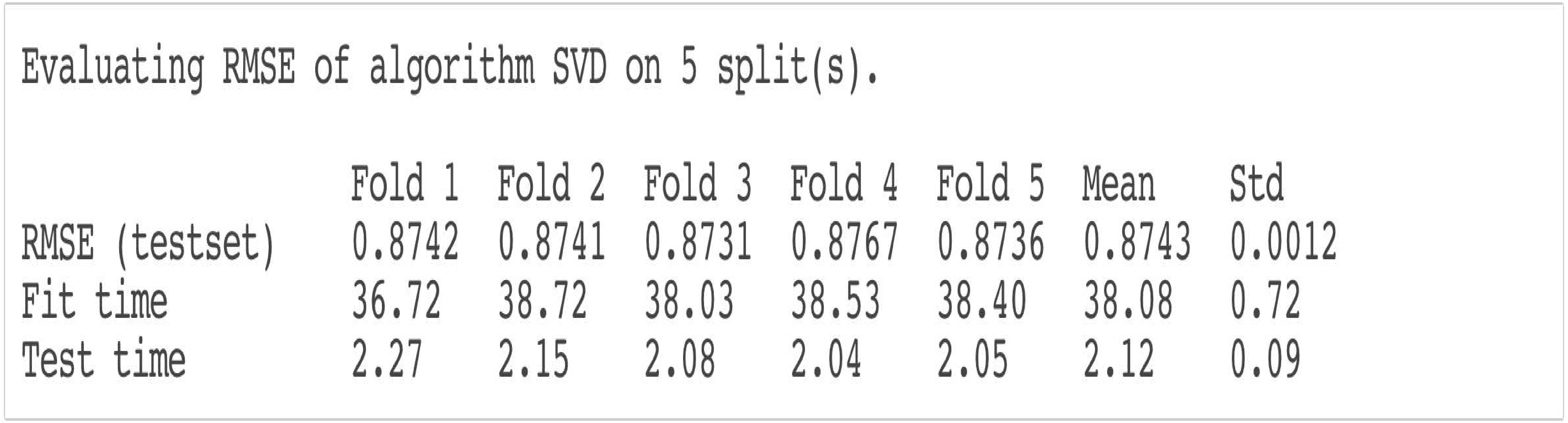
```
# Load Reader library.
reader = Reader()

# Load ratings dataset with the Dataset library.
data = Dataset.load_from_df(ratings[['user_id', 'movie_id', 'rating']], reader)
```

Model evaluation: compute RMSE

```
# Use the SVD algorithm.
svd = SVD()
```

```
# Compute the RMSE of the SVD algorithm.
evaluate_model = cross_validate(svd, data, measures=['RMSE'], cv=5, verbose=True)
```



```
Evaluating RMSE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8742  0.8741  0.8731  0.8767  0.8736  0.8743  0.0012
Fit time          36.72   38.72   38.03   38.53   38.40   38.08   0.72
Test time         2.27    2.15    2.08    2.04    2.05    2.12    0.09
```

Model evaluation: fitting the model

- By looking into the RMSE values, we can evaluate the recommender system
- The lower the RMSE, the better our recommender system is performing
- Now let's create training data and fit our model

```
trainset = data.build_full_trainset()
svd.fit(trainset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD object at 0x7f24b6876990>
```

Model evaluation: prediction

```
# User 1310 and his prior ratings.
ratings[ratings['user_id'] == 1310].head()
```

```
        user_id  movie_id  rating
215928     1310      2988       3
215929     1310      1293       5
215930     1310      1295       2
215931     1310      1299       4
215932     1310      2243       4
```

```
# Average rating user 1310 will give to movie ID 1994.
svd.predict(1310, 1994)
```

```
Prediction(uid=1310, iid=1994, r_ui=None, est=3.655999305076593, details={'was_impossible':
False})
```

- `est` is the average rating the user with ID 1310 will give to movie with ID 1994 based on his previous ratings and the similarity of the movie 1994 with those movies

Knowledge check 2

Exercise 2

Module completion checklist

Objective	Complete
Describe the collaborative filtering recommender system and its types	✔
Build an item-based collaborative filtering algorithm	✔
Summarize a model-based collaborative filtering recommender system and the concept of SVD	✔
Implement a model-based collaborative algorithm and generate predictions	✔
Evaluate the model using performance metrics	✔

Summary and next steps

- In today's class we learned about:
  - recommendation engines and their use cases
  - collaborative filtering recommender system and its types
  - item-based collaborative filtering algorithm
  - model-based collaborative filtering recommender system and the concept of SVD
- There are many data science areas you can explore next: neural networks, deep learning, text mining, and others!

Congratulations on completing this module!