DATA SOCIETY®

Advanced classification - day 6

"One should look for what is and not what he thinks should be."
-Albert Einstein.

Module completion checklist

Objective	Complete
Summarize what recommendation engines are used for and its types	
Load and explore data to get it ready for the recommender system	
Explain the concept of a content-based recommender system	
Build a content-based recommender system	
Generate recommendations from the content based recommender system and discuss the pitfalls	
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	

Recommendation engines - what are they?

- Today, we will be discussing a popular product of data science, **a recommendation engine**
- A recommendation engine can do multiple things. Mainly it:
  - Filters the given data using different algorithms
  - Recommends most relevant items to the users
  - Captures the past behavior of a customer
  - It then uses that behavior to recommend products based on a user or item

Types of recommender system

- Today, we will be exploring the following types of recommender systems:
  - Content based recommender
  - User based collaborative filtering
  - Item based collaborative filtering
  - Model based recommender
- We will understand the concept behind each one and implement and see which is the best model for our dataset

Tasks for today

- Working with the **MovieLens** **dataset in class**
  - To make movie recommendations based on similar movies
  - To make user recommendations based on similar users

Tasks for today

- **Working with music recommendations dataset in exercises**
  - To make recommendations based on similar items
  - To make recommendations based on similar genres and ratings

Movie recommender system

- Today, we'll build a **movie recommender system**
- A good example for it is Netflix!
- Let's understand how it works first
  - User `A` watches `Friends` and `Big Bang Theory`
  - User `B` watches `Friends`, then Netflix suggests `Big Bang Theory` to them from the data collected from user `A`

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your `af-werx` folder

```
# Set `main_dir` to the location of your `af-werx` folder (for Linux).
main_dir = "/home/[username]/Desktop/af-werx"
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Mac).
main_dir = '/Users/[username]/Desktop/af-werx'
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Windows).
main_dir = "C:\\Users\\[username]\\Desktop\af-werx"
```

```
# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory.
data_dir = main_dir + "/data"
```

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
from math import sqrt
from scipy.sparse.linalg import svds
from surprise import Reader, Dataset, SVD, evaluate
```

Working directory

- Set working directory to the `data_dir` variable we set
- We do this using the `os.chdir` function, change directory
- We can then check the working directory using `.getcwd()`
- For complete documentation of the `os` package, ***click here***

```
# Set working directory.
os.chdir(data_dir)
```

```
# Check working directory.
print(os.getcwd())
```

```
/home/[user-name]/Desktop/af-werx/data
```

Load the dataset and check the structure

- We have 3 datasets - **ratings, users, and movies**

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

# Reading users file.
users = pd.read_csv('users.csv', sep='\t', encoding='latin-1',
usecols = ['user_id', 'gender', 'zipcode', 'age_desc', 'occ_desc'])

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

print(ratings.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 3 columns):
user_id     1000209 non-null int64
movie_id    1000209 non-null int64
rating      1000209 non-null int64
dtypes: int64(3)
memory usage: 22.9 MB
None
```

Load the dataset and check the structure

```
print(users.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6040 entries, 0 to 6039
Data columns (total 5 columns):
user_id     6040 non-null int64
gender      6040 non-null object
zipcode     6040 non-null object
age_desc    6040 non-null object
occ_desc    6040 non-null object
dtypes: int64(1), object(4)
memory usage: 236.1+ KB
None
```

```
print(movies.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3883 entries, 0 to 3882
Data columns (total 3 columns):
movie_id    3883 non-null int64
title       3883 non-null object
genres      3883 non-null object
dtypes: int64(1), object(2)
memory usage: 91.1+ KB
None
```

View the head of the dataset

```
print(ratings.head(3))
```

```
   user_id  movie_id  rating
0        1      1193       5
1        1       661       3
2        1       914       3
```

```
print(users.head(3))
```

```
   user_id gender zipcode  age_desc       occ_desc
0        1      F   48067  Under 18   K-12 student
1        2      M   70072       56+  self-employed
2        3      M   55117     25-34      scientist
```

```
print(movies.head(3))
```

```
   movie_id                    title                         genres
0         1         Toy Story (1995)   Animation|Children's|Comedy
1         2           Jumanji (1995)  Adventure|Children's|Fantasy
2         3  Grumpier Old Men (1995)                Comedy|Romance
```

- **Ratings** dataframe contains the rating given by each user to each movie
- **Users** dataframe contains user information
- **Movies** dataframe contains information on the movies

Movies - data exploration

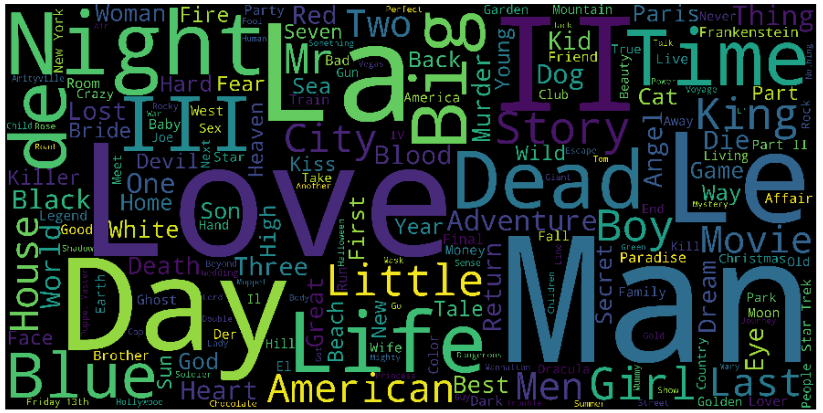
- Let's see which words are repeated the most in the title of the movies

```
# Create a word cloud of the movie titles.
movies['title'] =
movies['title'].fillna("").astype('str')
title_corpus = ' '.join(movies['title'])
title_wordcloud = WordCloud(stopwords =
STOPWORDS, background_color = 'black',
height = 2000, width =
4000).generate(title_corpus)

# Plot the word cloud.
plt.figure(figsize = (16, 8))
plt.imshow(title_wordcloud)
plt.axis('off')
```

```
(-0.5, 3999.5, 1999.5, -0.5)
```

```
plt.show()
```



Ratings - data exploration

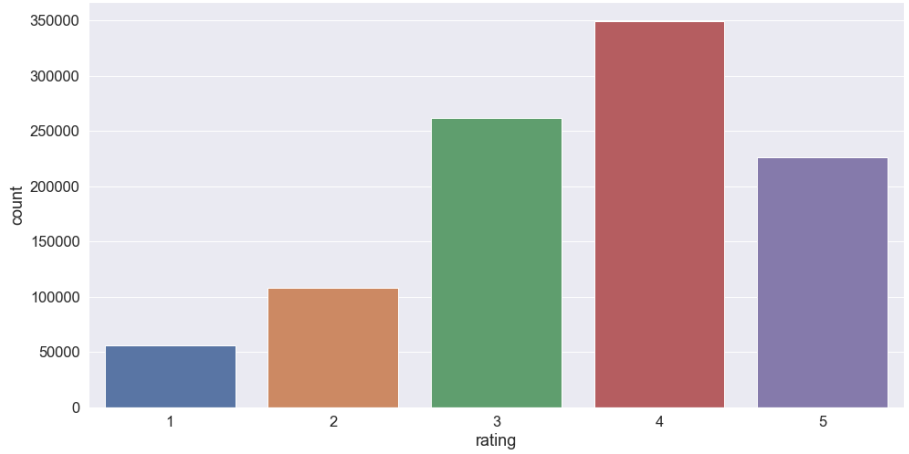
- Let's see the distribution of ratings and most popular ratings from the users to the movies

```
# Get summary statistics of ratings.
print(ratings['rating'].describe())
```

```
count    1.000209e+06
mean     3.581564e+00
std      1.117102e+00
min      1.000000e+00
25%      3.000000e+00
50%      4.000000e+00
75%      4.000000e+00
max      5.000000e+00
Name: rating, dtype: float64
```

```
sns.set_style('whitegrid')
sns.set(font_scale=1.5)

# Display distribution of ratings.
sns.countplot(ratings['rating'])
```



Combining dataframes

- We will combine all three dataframes and take only the movie `title`, `genres` and `rating`
- We will get top 5 movies with highest rating

```
# Join all 3 files into one dataframe.
dataset = pd.merge(pd.merge(movies, ratings), users)

# Display 5 movies with highest ratings.
print(dataset[['title', 'genres', 'rating']].sort_values('rating', ascending = False).head(5))
```

```
                        title                        genres  rating
0             Toy Story (1995)  Animation|Children's|Comedy       5
489283  American Beauty (1999)             Comedy|Drama       5
489259         Election (1999)                    Comedy       5
489257      Matrix, The (1999)   Action|Sci-Fi|Thriller       5
489256     Dead Ringers (1988)            Drama|Thriller       5
```

- Let's fetch all the genres we have in our dataset

```
# Make a census of the genre keywords.
genre_labels = set()
for s in movies['genres'].str.split('|').values:
    genre_labels = genre_labels.union(set(s))
```

Function to count the genres

- We will write a function to count the occurrence of the genres

```
# Create a function that counts the number of times each of
the genre keywords appear.
def count_word(dataset, ref_col, census):
    keyword_count = dict()
    for s in census:
        keyword_count[s] = 0
    for census_keywords in dataset[ref_col].str.split('|'):
        if type(census_keywords) == float and
pd.isnull(census_keywords):
            continue
        for s in [s for s in census_keywords if s in
census]:
            if pd.notnull(s):
                keyword_count[s] += 1

    # Convert the dictionary in a list to sort the keywords
by frequency.
    keyword_occurrences = []
    for k,v in keyword_count.items():
        keyword_occurrences.append([k,v])
    keyword_occurrences.sort(key = lambda x:x[1],
                             reverse = True)
    return keyword_occurrences, keyword_count
```

- Fetch the top 5 genres

```
# Calling this function gives access
to a list of genre keywords, which are
sorted by decreasing frequency.
keyword_occurrences, dum =
count_word(movies, 'genres',
genre_labels)
print(keyword_occurrences[:5])
```

```
[['Drama', 1603], ['Comedy', 1200],
['Action', 503], ['Thriller', 492],
['Romance', 471]]
```

Word cloud for genres

- We will create a word cloud to view the genres

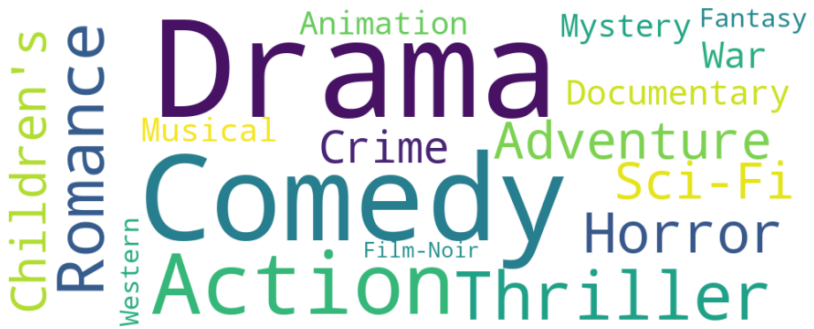
```
# Define the dictionary used to produce the
genre word cloud.
genres = dict()
trunc_occurrences = keyword_occurrences[0:18]
for s in trunc_occurrences:
    genres[s[0]] = s[1]

# Create the word cloud.
genre_wordcloud = WordCloud(width = 1000, height
= 400, background_color = 'white')
genre_wordcloud.generate_from_frequencies(genres)
```

```
# Plot the word cloud.
f, ax = plt.subplots(figsize = (16, 8))
plt.imshow(genre_wordcloud, interpolation =
"bilinear")
plt.axis('off')
```

```
(-0.5, 999.5, 399.5, -0.5)
```

```
plt.show()
```



Knowledge check 1

Exercise 1

Module completion checklist

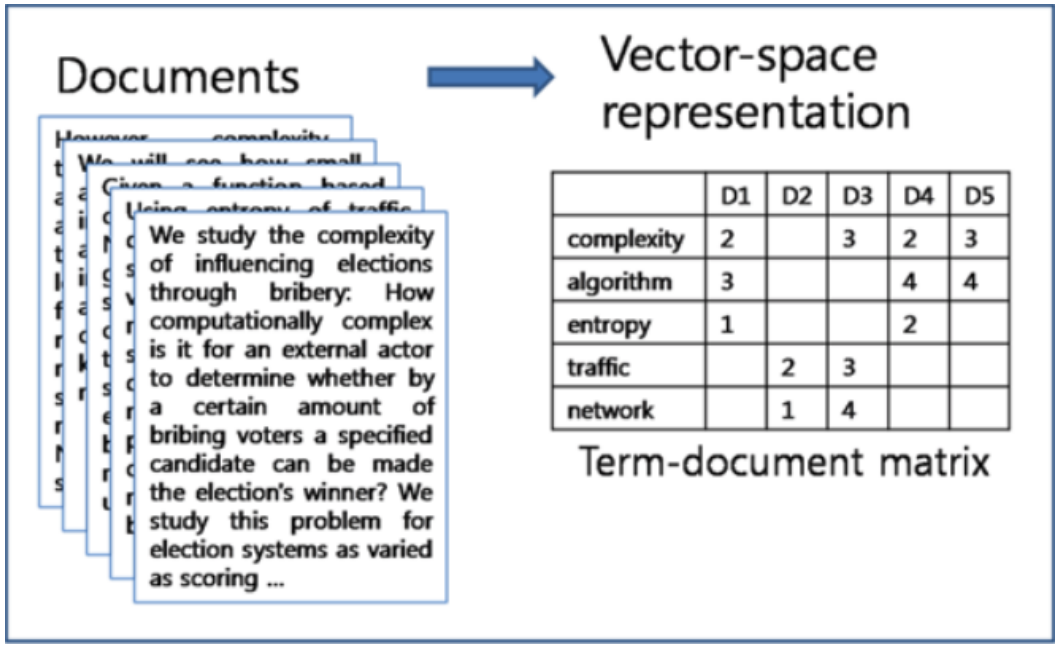
Objective	Complete
Summarize what recommendation engines are used for and and its types	✔
Load and explore data to get it ready for the recommender system	✔
Explain the concept of a content-based recommender system	
Build a content-based recommender system	
Generate recommendations from the content based recommender system and discuss the pitfalls	
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	

Content based recommender system

- A content based recommender system is based on the **category of the items** being recommended
- The idea is that if you like an item, you will also like an item that is similar to the first one
- It can be used mostly when **context or property** of each item can be determined
- For our Movielens dataset, we have the **genre** of each movie which provides the context of each item
- We will build our content based recommender system based on the **TF-IDF algorithm on the genres and find the cosine similarity** for movies based on the genres

Term document frequency

- We will learn about **TF-IDF** (term frequency - inverse document frequency) more in detail when we do text mining, but let's review the basic concept so we can use it in recommender system
- Imagine you have multiple documents D1, D2,...Dn
- Each document has a list of words
- **Term document frequency** is the count of each word in each document



	D1	D2	D3	D4	D5
complexity	2		3	2	3
algorithm	3			4	4
entropy	1			2	
traffic		2	3		
network		1	4		

Term-document matrix

TF-IDF

- Suppose that the word **dog** repeats in the document multiple times - then, we can say that the document is about a **dog**
- But what if we have bunch of documents about different breeds of dog?
- In that case, we want to classify each document based on its breed and not just a general topic **dog**
- IDF is useful in that case where it negates the high frequency words which are important but not useful

TF-IDF

- Term frequency is the **frequency of a word in the document**
- Inverse document frequency (IDF) is the inverse of the document frequency among the whole corpus of documents
- The reason we use TF-IDF weighting is because it **negates the effect of high frequency words** that can overweight the importance of an item
- After getting the TF-IDF scores, we can determine the similarity between the terms by calculating the cosine similarity

Movielens - content based recommendation

Let's see the steps in building the content based recommender for our dataset:

- Split the genres into a **string array**
- Remove the stop words and calculate the **TF-IDF weights** for the genre array
  - For each movie ID and each genre for that movie, a TF-IDF score is calculated
- Find the cosine similarity matrix of the movies
- Find top 20 similar movies to any given movie based on its genres

Content based recommender implementation

- Break the genre string to string array

```
# Break up the big genre string into a string
array.
movies['genres'] =
movies['genres'].str.split('|')

# Convert genres to string values.
movies['genres'] =
movies['genres'].fillna("").astype('str')
print(movies['genres'].head())
```

```
0     ['Animation', "Children's", 'Comedy']
1    ['Adventure', "Children's", 'Fantasy']
2                     ['Comedy', 'Romance']
3                       ['Comedy', 'Drama']
4                                ['Comedy']
Name: genres, dtype: object
```

- Calculate the TF-IDF matrix using sklearn package

```
tf = TfidfVectorizer(analyzer = 'word',
ngram_range = (1, 2),
min_df = 0,
stop_words = 'english')

tfidf_matrix =
tf.fit_transform(movies['genres'])
print(tfidf_matrix.shape)
```

```
(3883, 127)
```

Content based recommender implementation

- Find the cosine similarity of the movies

```
# Cosine similarity for all movies, and look at the first
four rows and columns.
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
print(cosine_sim[:4, :4])
```

```
[[1.         0.14193614 0.09010857 0.1056164 ]
 [0.14193614 1.         0.         0.        ]
 [0.09010857 0.         1.         0.1719888 ]
 [0.1056164  0.         0.1719888  1.        ]]
```

```
print(cosine_sim.shape)
```

```
(3883, 3883)
```

- Build the list of movie titles

```
# Build a 1-dimensional array with
movie titles.
titles = movies['title']
indices = pd.Series(movies.index,
index = movies['title'])
print(titles[0:5])
```

```
0                          Toy Story
(1995)
1                            Jumanji
(1995)
2                 Grumpier Old Men
(1995)
3                Waiting to Exhale
(1995)
4    Father of the Bride Part II
(1995)
Name: title, dtype: object
```

Content based recommender implementation

- Write a function that returns top similar movies based on the cosine similarity value for any given movie

```
# Function that get movie recommendations based on the cosine similarity score of movie genres.
def genre_recommendations(title):
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:21]
    movie_indices = [i[0] for i in sim_scores]
    return titles.iloc[movie_indices]
```

Content based recommender implementation

- Let's generate recommendations to a person who has watched `Toy Story (1995)` movie

```
print(genre_recommendations('Toy Story (1995)').head(20))
```

```
1050                Aladdin and the King of Thieves (1996)
2072                             American Tail, An (1986)
2073          American Tail: Fievel Goes West, An (1991)
2285                           Rugrats Movie, The (1998)
2286                                Bug's Life, A (1998)
3045                                  Toy Story 2 (1999)
3542                               Saludos Amigos (1943)
3682                                  Chicken Run (2000)
3685       Adventures of Rocky and Bullwinkle, The (2000)
236                                Goofy Movie, A (1995)
12                                             Balto (1995)
241                              Gumby: The Movie (1995)
310                            Swan Princess, The (1994)
592                                     Pinocchio (1940)
612                              Aristocats, The (1970)
700                              Oliver & Company (1988)
876     Land Before Time III: The Time of the Great Gi...
1010          Winnie the Pooh and the Blustery Day (1968)
1012                       Sword in the Stone, The (1963)
1020                        Fox and the Hound, The (1981)
Name: title, dtype: object
```

- All these recommendations are somewhat similar to the `Toy Story` movie!

Generate content-based recommendation

```
print(genre_recommendations('Assassins (1995)').head(20))
```

```
22                        Assassins (1995)
101                   Unforgettable (1996)
130                           Jade (1995)
181                    Mute Witness (1994)
188                           Safe (1995)
198             Tie That Binds, The (1995)
223                      Dream Man (1995)
237                       Hideaway (1995)
288                   Poison Ivy II (1995)
316                   Shallow Grave (1994)
369                   Red Rock West (1992)
418                           Blink (1994)
478                     Killing Zoe (1994)
486                          Malice (1993)
536                          Sliver (1993)
550                   Trial by Jury (1994)
557    Killer (Bulletproof Heart) (1994)
575                      Scorta, La (1993)
596                  Love and a .45 (1994)
656                           Fear (1996)
Name: title, dtype: object
```

Generate content-based recommendation

```
print(genre_recommendations('Sense and Sensibility (1995)').head(20))
```

```
24                                Leaving Las Vegas (1995)
34                                     Carrington (1995)
45                   How to Make an American Quilt (1995)
48                           When Night Is Falling (1995)
57                       Postino, Il (The Postman) (1994)
73                                    Bed of Roses (1996)
84                               Angels and Insects (1995)
103                  Bridges of Madison County, The (1995)
129                               Frankie Starlight (1995)
138                          Up Close and Personal (1996)
177                                       Mad Love (1995)
180                        Moonlight and Valentino (1995)
200                                 Total Eclipse (1995)
205                          Walk in the Clouds, A (1995)
213                                 Before Sunrise (1995)
219                               Circle of Friends (1995)
246                                Immortal Beloved (1994)
262    Like Water for Chocolate (Como agua para choco...
267                                   Love Affair (1994)
298                                Picture Bride (1995)
Name: title, dtype: object
```

Pros and cons of a content-based recommender

- Pros
  - No need for data on users which is helpful in **cold start problems**
  - Can recommend **new and unpopular items**
- Cons
  - Finding **appropriate features** can be hard
  - Does not recommend outside user's content profile (recommends movies from a particular genre only to users if they have seen that genre)
  - Cannot make use of **quality judgment of other users**

Knowledge check 2

Exercise 2

Module completion checklist

Objective	Complete
Summarize what recommendation engines are used for and its types	✔
Load and explore data to get it ready for the recommender system	✔
Explain the concept of a content-based recommender system	✔
Build a content-based recommender system	✔
Generate recommendations from the content based recommender system and discuss the pitfalls	✔
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 **particular genre only**
- They don't capture the **taste and preference of user across different genres**
- Collaborative filtering algorithms do a good job of recommending 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 the 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 the 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

Load the subset of data

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

```
os.chdir(data_dir)

# Read in the datasets.
rating_subset = pd.read_csv('ratings-subset.csv')
movies_subset = pd.read_csv('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')

os.chdir(data_dir)
corrMatrix = pd.read_csv('corrMatrix.csv')

print(corrMatrix.head())
```

```
                                    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 = 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 two movies are more similar
- Let's say we want to suggest new movies to user `A`, the item based filtering works as below:

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()

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)
```

```
# 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 3

Exercise 3

Module completion checklist

Objective	Complete
Summarize what recommendation engines are used for and its types	✔
Load and explore data to get it ready for the recommender system	✔
Explain the concept of a content-based recommender system	✔
Build a content-based recommender system	✔
Generate recommendations from the content based recommender system and discuss the pitfalls	✔
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

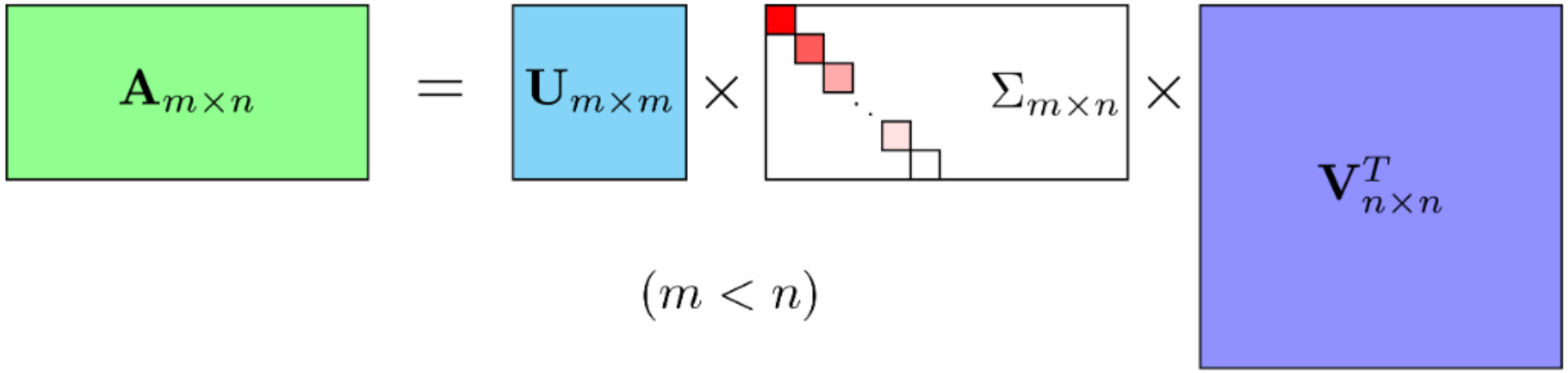
- Memory based algorithms do not scale well to **massive datasets**
- Rating matrices can overfit to noisy representations
- We need to apply a **dimensionality reduction technique** to derive recommendations
- We do that by factorization, which can:
  - **Discover hidden patterns** and correlations
  - 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

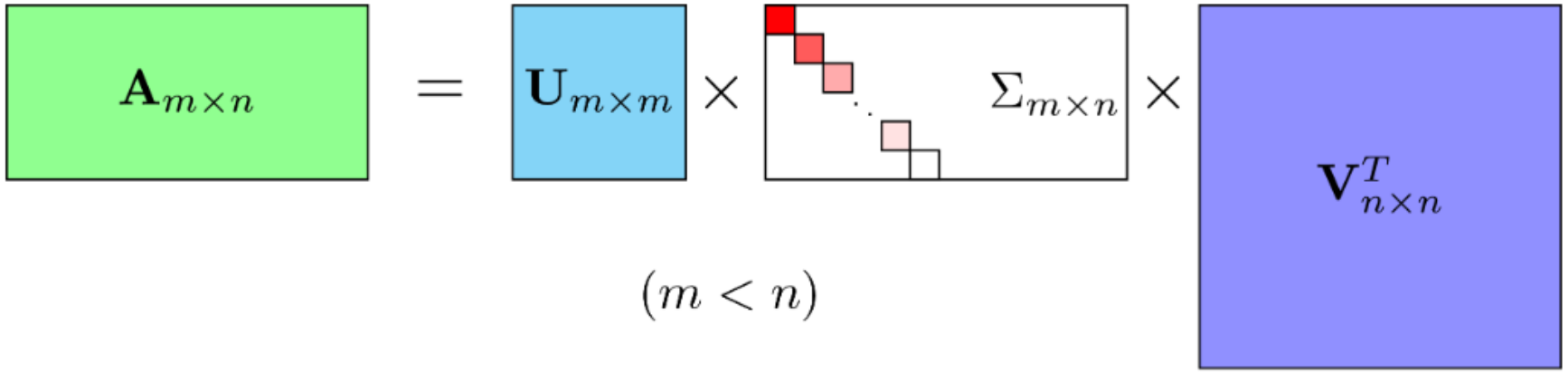
- This type of algorithm deals with **scalability and sparsity**
- It learns the preferences of users from known ratings
- When we have a highly sparse matrix with a lot of dimensions, then we can restructure the user-item matrix by doing matrix factorization
- Use this matrix to approximate the original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



SVD algorithm



- `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

```
# 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

```
warnings.filterwarnings("ignore", category = FutureWarning)
# Normalize the data.
R = Ratings.as_matrix()

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 highest predicted rating that the specified user has not already rated

```
def recommend_movies(predictions, userID, movies, original_ratings, num_recommendations):
    # Get and sort the user's predictions.
    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

```
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

```
# 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)
19     1310                                 Last Emperor, The (1987)
0      1310                                 Melvin and Howard (1980)
14     1310                                 Elephant Man, The (1980)
2      1310              Unbearable Lightness of Being, The (1988)
21     1310                                 Right Stuff, The (1983)
22     1310                                         Platoon (1986)
```

Recommend movies using SVD

```
# 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)
1199      1222                       Full Metal Jacket (1987)
2833      2919             Year of Living Dangerously (1982)
```

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
Summarize what recommendation engines are used for and its types	✔
Load and explore data to get it ready for the recommender system	✔
Explain the concept of a content-based recommender system	✔
Build a content-based recommender system	✔
Generate recommendations from the content based recommender system and discuss the pitfalls	✔
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
- 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)

# Split the dataset for 5-fold evaluation.
data.split(n_folds = 5)
```

Model evaluation

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

```
# Compute the RMSE of the SVD algorithm.
evaluate_model = evaluate(svd, data, measures=
['RMSE'])
```

```
Evaluating RMSE of algorithm SVD.

------------
Fold 1
RMSE: 0.8734
------------
Fold 2
RMSE: 0.8723
------------
Fold 3
RMSE: 0.8746
------------
Fold 4
RMSE: 0.8719
------------
Fold 5
RMSE: 0.8757
------------
------------
Mean RMSE: 0.8736
------------
------------
CaseInsensitiveDefaultDict(list,
                           {'rmse': [0.8734061529097356,
                             0.8723390289175087,
                             0.8746002841514304,
                             0.8719438495726501,
                             0.8756680623524425]})
```

Model evaluation

- By looking into the RMSE values, we can evaluate the recommender system
- The lower the RMSE, the better our recommender system is performing

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

```
<surprise.prediction_algorithms.matrix_factorization.SVD object at 0x1a4368ec88>
```

Model evaluation

```
# 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=3.322283582086774, 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

Other recommender systems

- A lot of companies around us use recommender systems today in practice to give useful recommendations on different products
- Amazon provides recommendations based on your previous purchase history
- UberEats provides food recommendation based on your previous orders from different restaurants
- **How would you want to use a recommender system?**

Knowledge check 4

Exercise 4

Module completion checklist

Objective	Complete
Summarize what recommendation engines are used for and its types	✔
Load and explore data to get it ready for the recommender system	✔
Explain the concept of a content-based recommender system	✔
Build a content-based recommender system	✔
Generate recommendations from the content based recommender system and discuss the pitfalls	✔
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	✔

Workshop!

- Workshops are to be completed in the afternoon either with a dataset for a capstone project or with another dataset of your choosing
- Make sure to annotate and comment your code so that it is easy for others to understand what you are doing
- This is an exploratory exercise to get you comfortable with the content we discussed today
- Today you will:
  - Find a dataset for which you want to generate recommendations
  - Use different recommender system algorithms to generate recommendations and analyze the insights

Congratulations on completing this module!