Sentiment Analysis & Recommender Sytems - Part 3

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

Warm up

Today we will look into recommender systems. Before we start, check out this article on how recommender systems are used in cancer-related issues - ***link***

Welcome back!

- In the last module, we learned about Support Vector Machines
- Today, we will talk about recommender systems. We will:
  - summarize recommendation engines and their use cases
  - prepare the data for the recommender system
  - build a recommender system
  - build an item-based collaborative filtering algorithm
  - implement a model-based collaborative algorithm and generate predictions
  - evaluate the model using performance metrics

Module completion checklist

Objective	Complete
Outline the applications of recommendation engines and their specific outcomes	
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	

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:
  - Selectively filters the given data using different algorithms
  - Recommends most relevant items to the users
  - Captures the past behavior of a user
  - It then uses that behavior to recommend a solution/product based on a user or item

Recommender systems: use cases

- Recommender systems can be used in various fields to give useful recommendations on different products and services
- **Department of Defense and intelligence community**: recommender systems can be used to hasten an analyst's response to cyber attacks. This can be done by selectively filtering information for users. ***Link***
- **E-governance in smart cities**: The overwhelming load of information and services in e-governance applications can make more prominent the development and use of personalized recommendation solutions for the different stakeholders and tasks. ***Link***

- **How would you want to use a recommender system?**

Types of recommender system

- 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 it, and see which one is the best model for our dataset

Movie recommender system

- 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 the user from the data collected from user `A`

Module completion checklist

Objective	Complete
Outline the applications of recommendation engines and their specific outcomes	✔
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	

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 dataset and check the structure

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

```
# Reading the ratings file.
ratings = pd.read_csv(data_dir+ '/ratings.csv', sep='\t', encoding='latin-1',
usecols = ['user_id', 'movie_id', 'rating'])
# Reading users file.
users = pd.read_csv(data_dir+ '/users.csv', sep='\t', encoding='latin-1',
usecols = ['user_id', 'gender', 'zipcode', 'age_desc', 'occ_desc'])
# Reading movies file.
movies = pd.read_csv(data_dir+ '/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):
 #   Column     Non-Null Count    Dtype
---  ------     --------------    -----
 0   user_id    1000209 non-null  int64
 1   movie_id   1000209 non-null  int64
 2   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):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   user_id    6040 non-null   int64 
 1   gender     6040 non-null   object
 2   zipcode    6040 non-null   object
 3   age_desc   6040 non-null   object
 4   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):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   movie_id  3883 non-null   int64 
 1   title     3883 non-null   object
 2   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

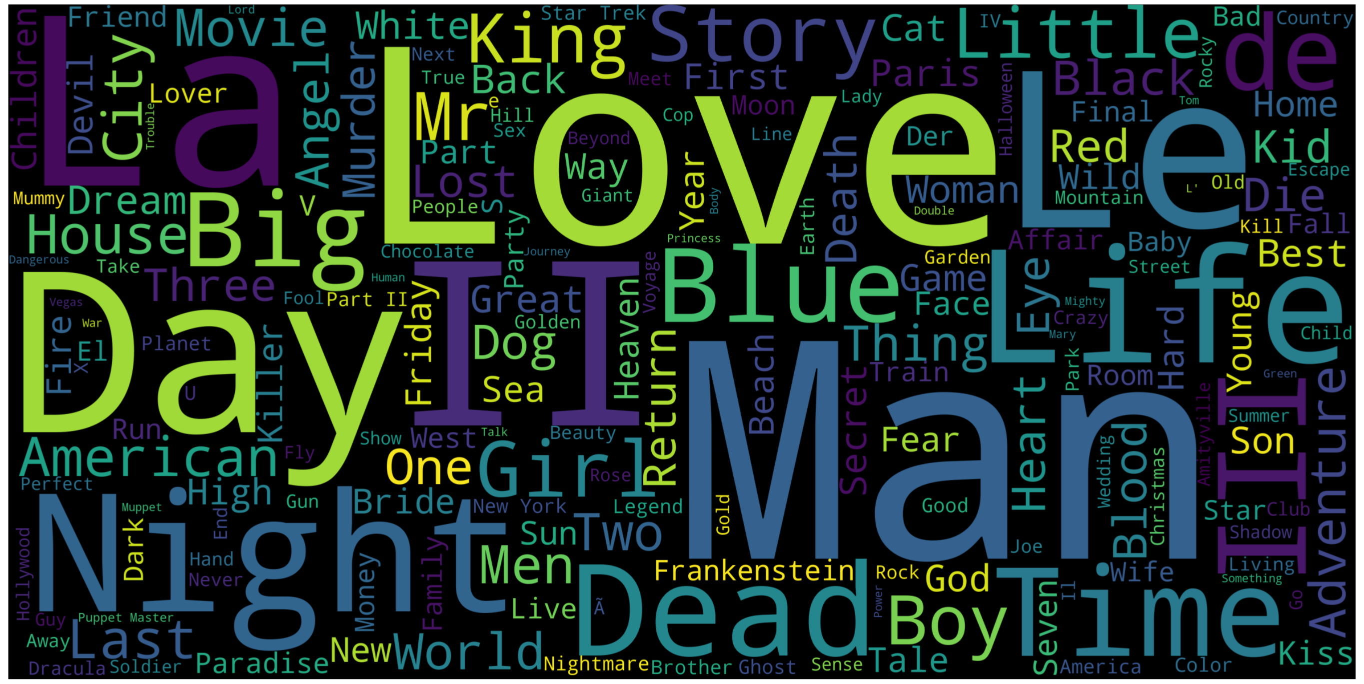
Movies - data exploration

- 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')
```

```
plt.show()
```



Ratings - data exploration

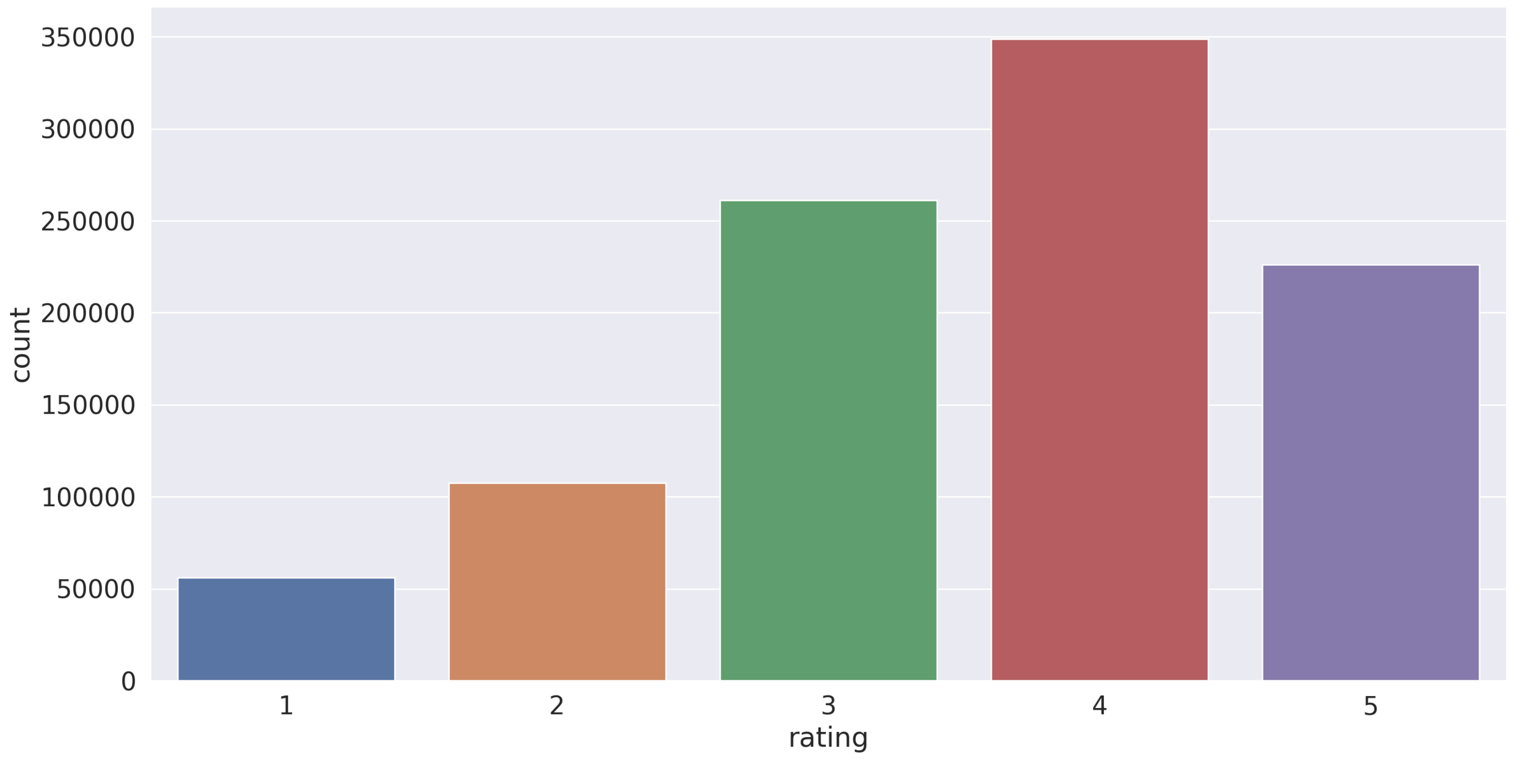
- Let's see the distribution of ratings and the most popular ratings from the users for the movies

```
# 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 the top 5 movies with the 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
```

Function to count the genres

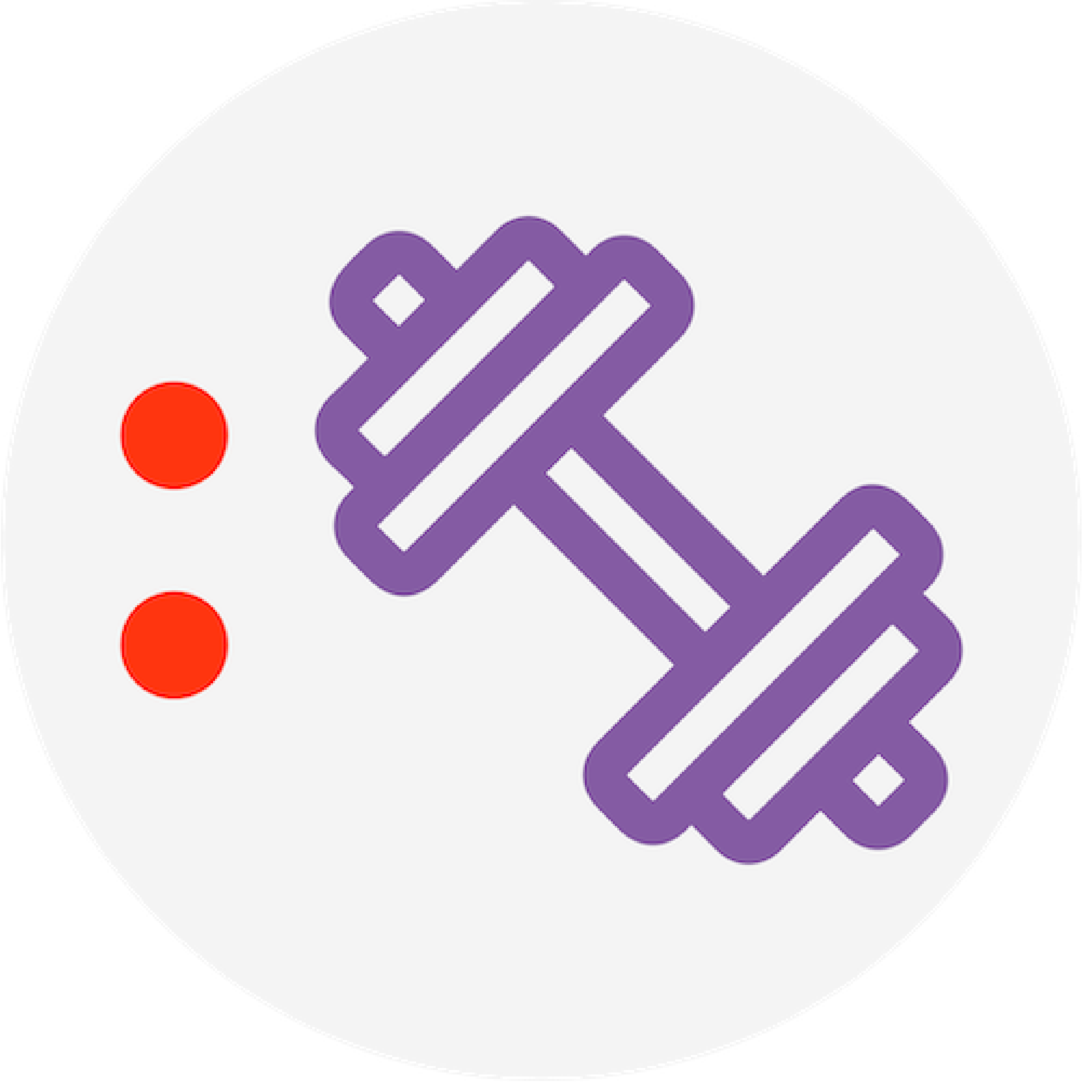
- Fetch the top 5 genres

```
# 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]]
```


Knowledge check 1

Exercise 1



Module completion checklist

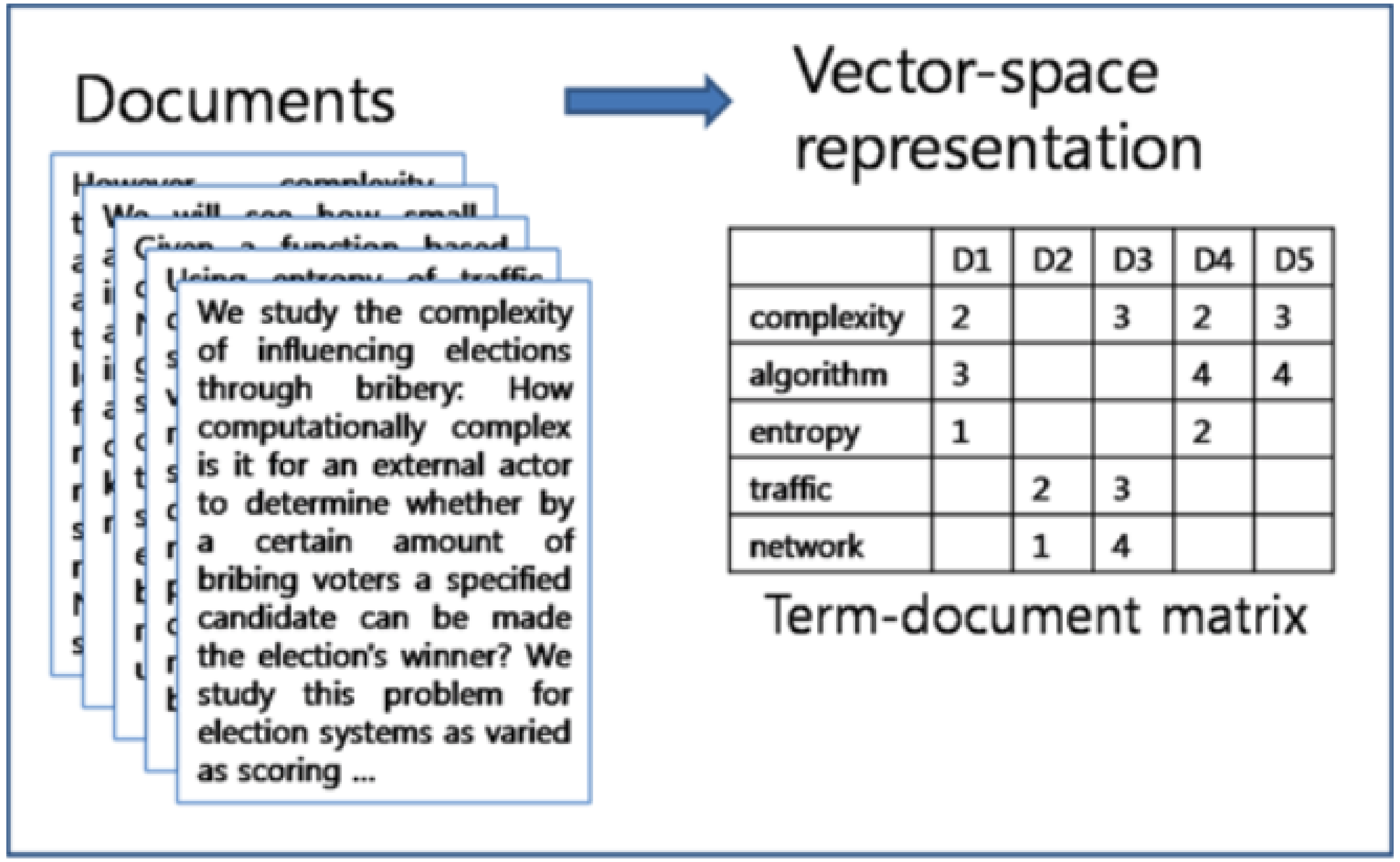
Objective	Complete
Outline the applications of recommendation engines and their specific outcomes	✔
Load and explore data to get it ready for the recommender system	✔
Explain the concept of a content-based recommender system	
Build a content-based recommender system	
Generate recommendations from the content-based recommender system and discuss the pitfalls	

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 another item that is similar to the first one
- It can be used mostly when the **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

- Let's review the basic concept of **TF-IDF** (term frequency - inverse document frequency) that we will use in the 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		

TF-IDF

- If 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 dogs?
- In that case, we want to classify each document based on the breed and not just a general topic **dog**
- IDF is useful in cases where it negates 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 the top 20 similar movies for any given movie based on its genres

Module completion checklist

Objective	Complete
Outline the applications of recommendation engines and their specific outcomes	✔
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	

Content-based recommender implementation

- Break the genre string into a 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
```


Content-based recommender implementation - cont'd

- Calculate the TF-IDF matrix using the 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 - cont'd

- 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 - cont'd

- Write a function that returns the 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 - cont'd

- Let's generate recommendations for a person who has watched the `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)
```

Module completion checklist

Objective	Complete
Outline the applications of recommendation engines and their specific outcomes	✔
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	

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)
```

Generate content-based recommendation

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

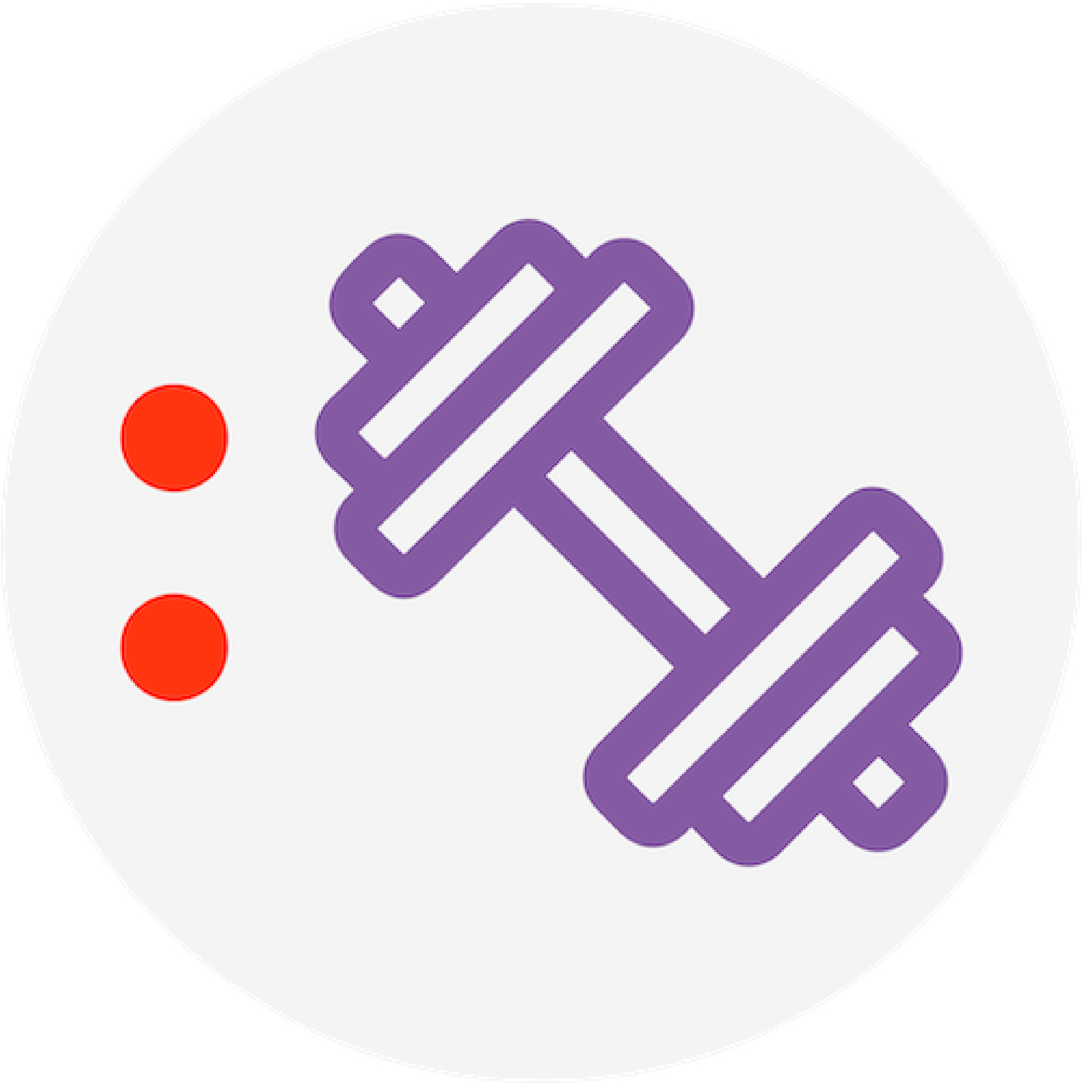
```
24                          Leaving Las Vegas (1995)
34                                 Carrington (1995)
45                How to Make an American Quilt (1995)
48                      When Night Is Falling (1995)
57                  Postino, Il (The Postman) (1994)
73                              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)
```

Pros and cons of a content-based recommender system

- 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 a user's content profile (recommends movies from a particular genre to users only 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
Outline the applications of recommendation engines and their specific outcomes	✔
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	✔

Congratulations on completing this module!