Recommender Systems

Learning Objectives

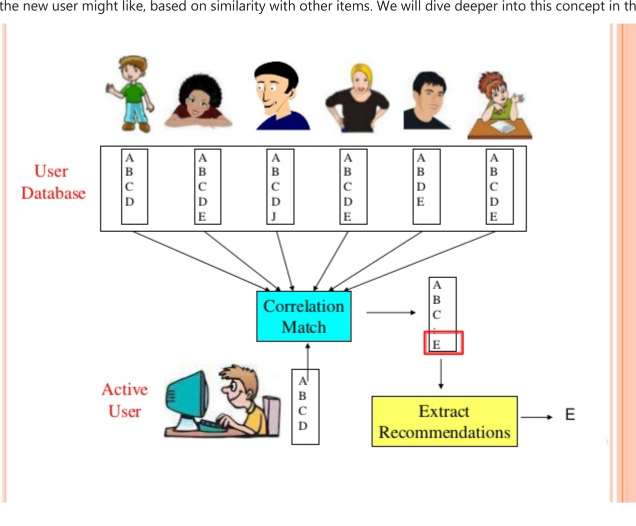
In this lesson, we will cover the following concepts:

- Recommendation system
- The long tail
- A simple popularity-based recommender system
- A collaborative filtering model
- Evaluating a recommendation system

What Is a Recommender System?

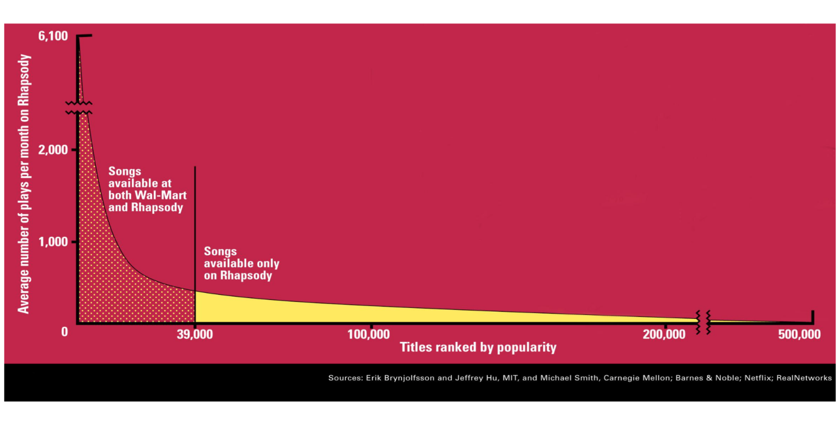
A recommender or recommendation system is a subclass of an information filtering system that seeks to predict the rating or preference that a user would give to an item.

Let's consider the example shown in the figure below. Here, we have a user database, that is, data consisting of items rated by the user. Now, let's suppose that a new user visits and likes five out of ten items on the website. A recommender system recommends the items that the new user might like, based on similarity with other items. We will dive deeper into this concept in the coming sections.



The Theory of Long Tail

- It shows how products in low demand or with low sales volume can collectively make up a market share that exceeds the relatively few current bestsellers and blockbusters but only if the store or distribution channel is large enough.
- The long tail concept looks at less popular goods in lower demand. The use of these goods could increase profitability as consumers navigate away from mainstream markets.
- This can be easily understood by looking at the figure below.



The figure above clearly shows the use of long tail by Rhapsody where they sell music albums both online and off-line. We can clearly observe the following:

- Both Rhapsody and Walmart sell the most popular music albums online, but the former offers 19 times more songs than Walmart. Even though there is a demand for popular music albums, there is also a demand for the less popular online. Recommender systems leverage these less popular items online.

Recommend the Most Popular Items

- Let's consider the movie dataset. We will look carefully at the user ratings and think about what can be done.
- The answer that strikes first is the **most popular item**. This is exactly what we will be doing.
- Technically, this is the fastest method, but it does come with a major drawback, which is a lack of personalization. The dataset has many files; we will be looking at a few of them, mainly the ones that relate to movie ratings.

Popularity-Based Recommender System

- There is a division by section, so the user can look at the section of his or her interest.
- At a time, there are only a few hot topics; there is a high chance that a user wants to read the news which is being read by most others.

Import Libraries

In python, Pandas is used for data manipulation and analysis. NumPy is a package that includes a multidimensional array object and multiple derived objects. Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Seaborn is an open-source Python library built on top of matplotlib. Mean_squared_error is a library that measures the average of the squares of the errors, which is the average squared difference between the estimated values and the actual value.

These libraries are written with the import keyword.

In [1]:
```
import pandas as pd
import os, io
import numpy as np
from pandas import Series, DataFrame, read_table
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import mean_squared_error
%matplotlib inline
```

Exporting Dataset from Zip File

Before reading data you need to download "ml-100k.zip" dataset from the resource section and upload it into the Lab. We will use Up arrow icon which is shown in the left side under View icon. Click on the Up arrow icon and upload the file wherever it is downloaded into your system.

After this you will see the downloaded file will be visible on the left side of your lab with all the .ipynb files.

Then, the below snippet will extract the zip dataset to the corresponding folder.

In [3]:
```
import zipfile
with zipfile.ZipFile('ml-100k.zip', 'r') as zip_ref:
    zip_ref.extractall(".")
```

We start to explore the data set of movie ratings and our interest lies particularly in ratings. Let's see how we recommend the most popular (that is, highly rated) movies.

In [6]:
```
#Load the Ratings data
r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']
ratings = read_table('ml-100k//u.data',header=None,sep='\t')
ratings.columns = r_cols

i_cols = ['movie_id', 'movie title' ,'release date','video release date', 'IMDb URL',
          'unknown', 'Action', 'Adventure',
 'Animation', 'Children\'s', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy',
 'Film-Noir', 'Horror', 'Musical', 'Mystery', 'Romance', 'Sci-Fi',
          'Thriller', 'War', 'Western']
items = read_table('./ml-100k//u.item', sep='|',names=i_cols,
 encoding='latin-1')
```

In [7]:
```
ratings.head()
```

Out[7]:

	user_id	movie_id	rating	unix_timestamp
0	0	50	5	881250949
1	0	172	5	881250949
2	0	133	1	881250949
3	196	242	3	881250949
4	186	302	3	891717742

Ratings is a variable that stores all the columns from the ml-100k dataset in the u.data file. The head() function displays the first five rows from ratings.

Let's Build a Popularity-Based Recommender System

With our initial exploration, we decided that ideal data would be the one where we could also have the movie ratings with us. Let's see how we are able to do this.

We will use the pd.merge function that is used to combine data on common columns or indices.

In [8]:
```
new_data = pd.merge(items,ratings,on='movie_id')
new_data  = new_data[['movie_id','movie title','user_id','rating']]
```

In [9]:
```
new_data.head()
```

Out[9]:

	movie_id	movie title	user_id	rating
0	1	Toy Story (1995)	308	4
1	1	Toy Story (1995)	287	5
2	1	Toy Story (1995)	148	4
3	1	Toy Story (1995)	280	4
4	1	Toy Story (1995)	66	3

New_data is a variable that stores data read by the pd.merge function. It consists of items and ratings. The head() function displays the first five rows from new_data.

Before proceeding to build the recommender system, we will observe the following steps to recommend movies:

- Find unique users
- Count the number of times the movie has been seen
- Rank the scores (counts)

In [12]:
```
def popularity(train,title,ids):
    #user_id  #movie title
    train_data_grouped = train.groupby([title])[ids].count().reset_index()
    train_data_grouped.rename(columns = {ids: 'score'},inplace=True)
    train_data_sort = train_data_grouped.sort_values(['score',title], ascending = [0,1])
    train_data_sort['Rank'] = train_data_sort['score'].rank(ascending=0, method='first')
    popularity_recommendations = train_data_sort.head(10)
    return popularity_recommendations
```

In [13]:
```
popularity(new_data,'movie title','user_id')
```

Out[13]:

	movie title	score	Rank
1398	Star Wars (1977)	584	1.0
333	Contact (1997)	509	2.0
498	Fargo (1996)	508	3.0
1234	Return of the Jedi (1983)	507	4.0
860	Liar Liar (1997)	485	5.0
460	English Patient, The (1996)	481	6.0
1284	Scream (1996)	478	7.0
1523	Toy Story (1995)	452	8.0
32	Air Force One (1997)	431	9.0
744	Independence Day (ID4) (1996)	429	10.0

Drawback

Having recommended the movies, we can immediately conclude that the major drawback of such a system would be the **lack of personalization**.

Collaborative Filtering

In the newer, narrower sense, collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if person A has the same opinion as person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person.

Types of Collaborative Filtering

User-Based Collaborative Filtering

In this type, we find look-alike customers (based on similarity) and offer products that the first customer's look-alike chose in the past. This algorithm is very effective but takes a lot of time and resources. It computes every customer pair information, which takes time. Therefore, for big base platforms, this algorithm is hard to implement without a very strong parallelizing system.

1. Build a matrix of things each user bought or viewed or rated
2. Compute similarity scores between users
3. Find users similar to you
4. Recommend stuff they bought or viewed or rated that you haven't yet

Problems

1. People are fickle, so their tastes tend to change
2. There are usually more people than things

Item-Based Collaborative Filtering

It is quite similar to the previous algorithm, but instead of finding customer look-alikes, it tries to find items that look alike. Once we have an item look-alike matrix, we can easily recommend similar items to customers who have purchased an item from the store. This algorithm is far less resource-consuming than user-based collaborative filtering.

1. Find every pair of movies that were watched by the same person
2. Measure the similarity of rating across all the users who watched both
3. Sort movies by the similarity strength

Interesting fact

Item-based collaboration is extensively used in Amazon, and they came out with it in great detail. You can read more at Amazon

Let's get started with building our item-based collaborative recommender system. For convenience, let's split this into two parts.

- To find similarities between items
- To recommend them to users

Item-based collaborative filtering would be the most feasible solution, as the number of items is always lesser than the number of users and it improves the computational speed.

Leverage the Pandas

- To begin with, we will use the pandas pivot table to look at relationships between movies and we will use the pivot table in pandas. Pivot table in pandas is an excellent tool to summarize one or more numeric variable based on two other categorical variables.
- We start building a utility matrix (matrix consisting of movies and ratings)

In [14]:
```
movie_ratings = new_data.pivot_table(index=['user_id'],columns=['movie title'],
                                     values='rating')
```

In [15]:
```
movie_ratings.head()
```

Out[15]:

movie title	'Til There Was You (1997)	1-900 (1994)	101 Dalmatians (1996)	12 Angry Men (1957)	187 (1997)	2 Days in the Valley (1996)	20,000 Leagues Under the Sea (1954)	2001: A Space Odyssey (1968)	3 Ninjas: High Noon At Mega Mountain (1998)	39 Steps, The (1935)	...	Yankee Zulu (1994)	Year of the Horse (1997)	You So Crazy (1994)	Young Frankenstein (1974)
user_id															
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
1	NaN	NaN	2.0	5.0	NaN	NaN	3.0	4.0	NaN	NaN	...	NaN	NaN	NaN	5.0
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	...	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	2.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN

5 rows × 1664 columns

The above table gives information about the rating given by each user against the movie title. There are many NaN as it is not necessary for each user to review each movie. Let's start by looking at the geeks' most favorite, Star Wars, and see how it correlates pairwise with other movies in the table.

Similarity Function

To decide the similarity between two items in the dataset, let's briefly look at the popular similarity functions.

Terminology

- Let r_x denote the rating of the item x given by the user and r_y be the rating of item y. To find the similarity pairwise between two items the following metrics can be used:

cosine Index

$$sim(r_x, r_y) = cos(r_x, r_y) = \frac{r_x r_y}{||r_x||\ ||r_y||}$$

The major problem is that it treats missing values as negative.

Pearson Index

S_{xy} = Items x and y both have ratings

$$sim(r_x, r_y) = \frac{\sum_{x \epsilon s}(r_{xs} - r_{xm})(r_{ys} - r_y)}{(\sqrt{\sum_{s \epsilon s_{xy}}(r_{xs} - r_{xm})^2}(\sqrt{\sum_{s \epsilon s_{xy}}(r_{ys} - r_{ym})^2}}$$

Jaccard Index

$$Jaccard\ Index = \frac{Number\ in\ both\ sets}{Number\ in\ either\ set}$$

Let's start with the Pearson Index in this case. Now that we have understood how similar products can be found, let's start with the movie, Star Wars.

In [16]:
```
StarWarsRatings = movie_ratings['Star Wars (1977)']
StarWarsRatings.head()
```

Out[16]:
```
user_id
0    5.0
1    5.0
2    5.0
3    NaN
4    5.0
Name: Star Wars (1977), dtype: float64
```

Now, let's use the **corrwith()** function to check the pairwise correlation of Star Wars's user rating with other films in the column.

In [17]:
```
similarmovies = movie_ratings.corrwith(StarWarsRatings)
similarmovies =similarmovies.dropna()
df = pd.DataFrame(similarmovies)
df.head()
```

Out[17]:

	0
movie title	
'Til There Was You (1997)	0.872872
1-900 (1994)	-0.645497
101 Dalmatians (1996)	0.211132
12 Angry Men (1957)	0.184289
187 (1997)	0.027398

If we look at the data closely, we will find something incorrect.

The potential reason here is that a handful of people who have seen obscure films are messing up our movies. We want to get rid of the movies that only a few people have watched that show incorrect results.

We have used groupby function that involves some combination of splitting the object, applying a function, and combining the results and sort_values function that sorts by the values along either axis.

In [19]:
```
movie_stats = new_data.groupby('movie title').agg({'rating':[np.size,np.mean]})
```

In [20]:
```
check = movie_stats.sort_values([('rating','mean')],ascending=False)
```

In [21]:
```
check.head()
```

Out[21]:

	rating	
	size	mean
movie title		
They Made Me a Criminal (1939)	1	5.0
Marlene Dietrich: Shadow and Light (1996)	1	5.0
Saint of Fort Washington, The (1993)	2	5.0
Someone Else's America (1995)	1	5.0
Star Kid (1997)	3	5.0

Now, we can clearly observe that there are movies that have very few rating counts (size). Therefore, we set a threshold of the movie count to have at least 100 ratings.

In [22]:
```
popularmovies = movie_stats['rating']['size']>=100

movie_stats[popularmovies].sort_values([('rating','mean')],ascending=False)[:10]
```

Out[22]:

	rating	
	size	mean
movie title		
Close Shave, A (1995)	112	4.491071
Schindler's List (1993)	298	4.466443
Wrong Trousers, The (1993)	118	4.466102
Casablanca (1942)	243	4.456790
Shawshank Redemption, The (1994)	283	4.445230
Rear Window (1954)	209	4.387560
Usual Suspects, The (1995)	267	4.385768
Star Wars (1977)	584	4.359589
12 Angry Men (1957)	125	4.344000
Citizen Kane (1941)	198	4.292929

In [24]:
```
df = movie_stats[popularmovies].join(DataFrame(similarmovies,columns=['similarity']))
df.sort_values('similarity',ascending=False)[:20]
```

Out[24]:

	(rating, size)	(rating, mean)	similarity
movie title			
Star Wars (1977)	584	4.359589	1.000000
Empire Strikes Back, The (1980)	368	4.206522	0.748353
Return of the Jedi (1983)	507	4.007890	0.672556
Raiders of the Lost Ark (1981)	420	4.252381	0.536117
Austin Powers: International Man of Mystery (1997)	130	3.246154	0.377433
Sting, The (1973)	241	4.058091	0.367538
Indiana Jones and the Last Crusade (1989)	331	3.930514	0.350107
Pinocchio (1940)	101	3.673267	0.347868
Frighteners, The (1996)	115	3.234783	0.332729
L.A. Confidential (1997)	297	4.161616	0.319065
Wag the Dog (1997)	137	3.510949	0.318645
Dumbo (1941)	123	3.495935	0.317656
Bridge on the River Kwai, The (1957)	165	4.175758	0.316580
Philadelphia Story, The (1940)	104	4.115385	0.314272
Miracle on 34th Street (1994)	101	3.722772	0.310921
E.T. the Extra-Terrestrial (1982)	300	3.833333	0.303619
Mystery Science Theater 3000: The Movie (1996)	130	3.430769	0.301809
Cinderella (1950)	129	3.581395	0.299163
Batman (1989)	201	3.427861	0.289344
Swingers (1996)	157	3.828025	0.289310

Building an End-to-End Recommender System

We will list points that need to be followed to recommend a movie based on what we did till now :

- Compute the correlation score for every pair in the matrix
- Choose a user and find his or her movies of interest
- Recommend movies to him or her
- Improve on the recommendation

The pandas method **corr()** will compute the correlation score for every pair in the matrix. This gives a correlation score between every pair of movies in turn creating a sparse matrix. Let's see how this looks.

In [25]:
```
corrMatrix = movie_ratings.corr(method='pearson',min_periods=100)
corrMatrix.head()
```

Out[25]:

movie title	'Til There Was You (1997)	1-900 (1994)	101 Dalmatians (1996)	12 Angry Men (1957)	187 (1997)	2 Days in the Valley (1996)	20,000 Leagues Under the Sea (1954)	2001: A Space Odyssey (1968)	3 Ninjas: High Noon At Mega Mountain (1998)	39 Steps, The (1935)	...	Yankee Zulu (1994)	Year of the Horse (1997)	You So Crazy (1994)	Youn Frankenstei (1974
movie title															
'Til There Was You (1997)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
1-900 (1994)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
101 Dalmatians (1996)	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
12 Angry Men (1957)	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
187 (1997)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na

5 rows × 1664 columns

Now, we want to recommend movies to a friend, so let's have a look at the movies our friend has rated.

In [26]:
```
friend_ratings = movie_ratings.loc[1].dropna()[1:4]
friend_ratings
```

Out[26]:
```
movie title
12 Angry Men (1957)                      5.0
20,000 Leagues Under the Sea (1954)      3.0
2001: A Space Odyssey (1968)             4.0
Name: 1, dtype: float64
```

In [27]:
```
simcandidates= pd.Series()
for i in range(0,len(friend_ratings.index)):
    print('Adding similars to ', friend_ratings.index[i])

    print('--------------------------------')
    sims = corrMatrix[friend_ratings.index[i]].dropna()
    sims = sims.map(lambda x: x*friend_ratings[i]) # Assigning lower weights to movies with lower ratings.
    simcandidates  = simcandidates.append(sims)

    print('sorting')

    simcandidates.sort_values(inplace=True,ascending=False)

    print(simcandidates.head(10))
```

```
Adding similars to  12 Angry Men (1957)
--------------------------------
sorting
12 Angry Men (1957)               5.000000
Star Wars (1977)                  0.921447
Raiders of the Lost Ark (1981)    0.646672
dtype: float64
Adding similars to  20,000 Leagues Under the Sea (1954)
--------------------------------
sorting
12 Angry Men (1957)               5.000000
Star Wars (1977)                  0.921447
Raiders of the Lost Ark (1981)    0.646672
dtype: float64
Adding similars to  2001: A Space Odyssey (1968)
--------------------------------
sorting
12 Angry Men (1957)                                                      5.000000
2001: A Space Odyssey (1968)                                             4.000000
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1963)    1.571663
Clockwork Orange, A (1971)                                               1.552285
Citizen Kane (1941)                                                      1.481653
Lawrence of Arabia (1962)                                                1.324881
Chinatown (1974)                                                         1.311644
Apocalypse Now (1979)                                                    1.251388
Birds, The (1963)                                                        1.226125
Godfather, The (1972)                                                    1.222868
dtype: float64
```

Some movies come up more than once, because they are very similar to the ones that the user has rated. Let's eliminate them.

In [28]:
```
simcandidates = simcandidates.groupby(simcandidates.index).sum()
simcandidates.sort_values(inplace=True,ascending=False)
simcandidates.head(10)
```

Out[28]:
```
12 Angry Men (1957)                                                      5.000000
2001: A Space Odyssey (1968)                                             4.000000
Star Wars (1977)                                                         1.844984
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1963)    1.571663
Clockwork Orange, A (1971)                                               1.552285
Citizen Kane (1941)                                                      1.481653
Raiders of the Lost Ark (1981)                                           1.438781
Lawrence of Arabia (1962)                                                1.324881
Chinatown (1974)                                                         1.311644
Apocalypse Now (1979)                                                    1.251388
dtype: float64
```

Having done all the computations using pandas, we can see that it is computationally intensive. We have a Python module that does that for us.

Using the Surprise Module

Python Surprise is an easy-to-use Python scikit for recommender systems. Let's see how to build a recommender system using the surprise module and focus on the model inspired by K-Nearest Neighbors (KNN).

Common Practice

1. Define Similarity S_{ij} in terms of i and j
2. Select K nearest neighbors N(i;X)
   - Items most similar to i that were rated by X
3. Estimate rating r_{xi} as the weighted average

$$r_{x_i} = b_{x_i} + \frac{\sum_{j \epsilon N(i;x)} S_{ij}(r_{x_j} - b_{x_j})}{\sum_{j \epsilon N(i;x)} S_{ij}}$$

Here, the term b_{x_i} is the baseline estimator for the rating comprising three terms: the overall mean movie rating, rating deviation of user x, and rating deviation of the movie i.

Evaluation Metrics

Comparing Predictions with Known Ratings

RMSE

- Root Mean Square Error (RMSE)
  - $ \sqrt{\frac{1}{N}\sum_{x_i}(\textbf{$r_{xi}−r{xi}^*})^2} *here $r{xi}isthepredictedratingandr{x_i}^*$* is the actual rating
- Precision at top 10
  - % of those in top 10

Note: In this lesson, we saw the use of the recommender systems.