**CONTENT-BASED FILTERING**

Recommendation systems are a collection of algorithms used to recommend items to users based on information taken from the user. These systems have become ubiquitous, and can be commonly seen in online stores, movies databases and job finders. In this notebook, we will explore Content-based recommendation systems and implement a simple version of one using Python and the Pandas library.

# Acquiring the Data

!wget -O moviedataset.zip https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/moviedataset.zip

print('unziping ...')

!unzip -o -j moviedataset.zip

#Dataframe manipulation library

import pandas as pd

#Math functions, we'll only need the sqrt function so let's import only that

from math import sqrt

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

#Storing the movie information into a pandas dataframe

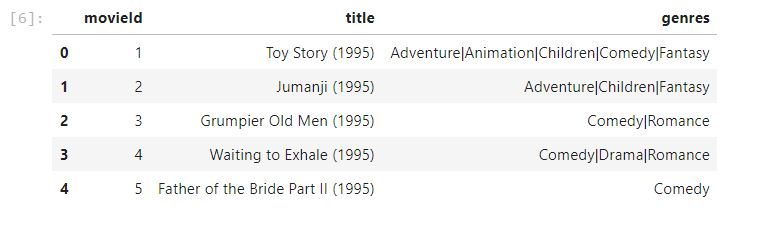
movies\_df = pd.read\_csv('movies.csv')

#Storing the user information into a pandas dataframe

ratings\_df = pd.read\_csv('ratings.csv')

#Head is a function that gets the first N rows of a dataframe. N's default is 5.

movies\_df.head()



Let's also remove the year from the **title** column by using pandas' replace function and store in a new **year** column.

#Using regular expressions to find a year stored between parentheses

#We specify the parantheses so we don't conflict with movies that have years in their titles

movies\_df['year'] = movies\_df.title.str.extract('(\(\d\d\d\d\))',expand=False)

#Removing the parentheses

movies\_df['year'] = movies\_df.year.str.extract('(\d\d\d\d)',expand=False)

#Removing the years from the 'title' column

movies\_df['title'] = movies\_df.title.str.replace('(\(\d\d\d\d\))', '')

#Applying the strip function to get rid of any ending whitespace characters that may have appeared

movies\_df['title'] = movies\_df['title'].apply(lambda x: x.strip())

movies\_df.head()



#Every genre is separated by a | so we simply have to call the split function on |

movies\_df['genres'] = movies\_df.genres.str.split('|')

movies\_df.head()



Since keeping genres in a list format isn't optimal for the content-based recommendation system technique, we will use the One Hot Encoding technique to convert the list of genres to a vector where each column corresponds to one possible value of the feature. This encoding is needed for feeding categorical data. In this case, we store every different genre in columns that contain either 1 or 0. 1 shows that a movie has that genre and 0 shows that it doesn't. Let's also store this dataframe in another variable since genres won't be important for our first recommendation system.

#Copying the movie dataframe into a new one since we won't need to use the genre information in our first case.

moviesWithGenres\_df = movies\_df.copy()

#For every row in the dataframe, iterate through the list of genres and place a 1 into the corresponding column

for index, row in movies\_df.iterrows():

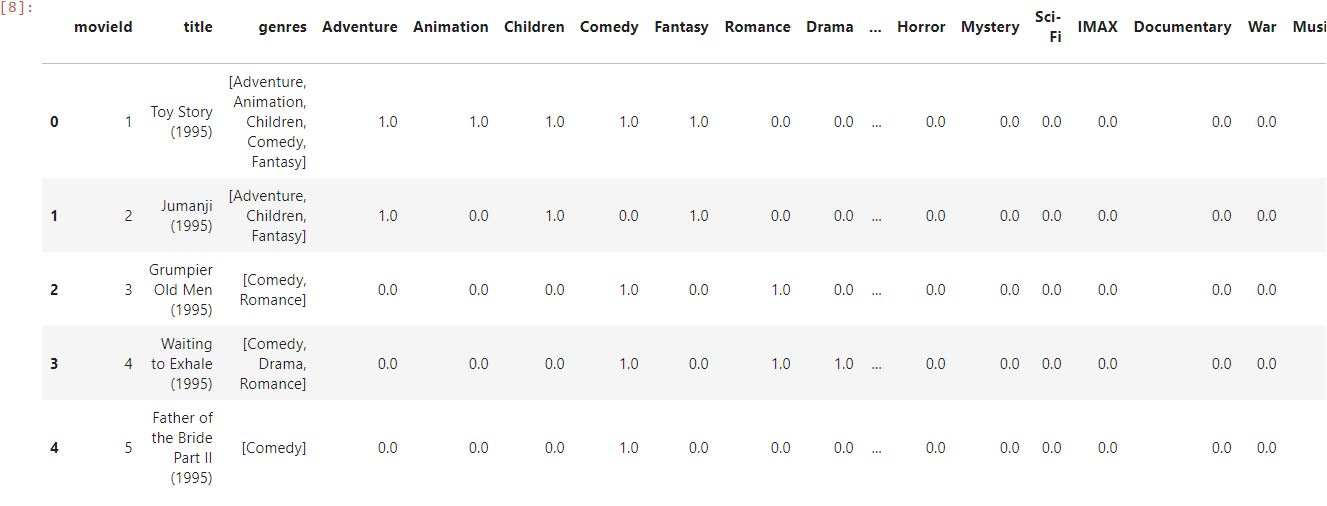
for genre in row['genres']:

moviesWithGenres\_df.at[index, genre] = 1

#Filling in the NaN values with 0 to show that a movie doesn't have that column's genre

moviesWithGenres\_df = moviesWithGenres\_df.fillna(0)

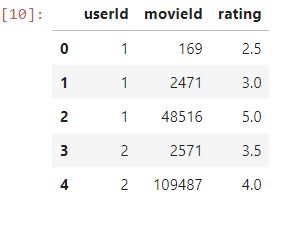
moviesWithGenres\_df.head()



#Drop removes a specified row or column from a dataframe

ratings\_df = ratings\_df.drop('timestamp', 1)

ratings\_df.head()



# Content-Based recommendation system

Now, let's take a look at how to implement **Content-Based** or **Item-Item recommendation systems**. This technique attempts to figure out what a user's favourite aspects of an item is, and then recommends items that present those aspects. In our case, we're going to try to figure out the input's favorite genres from the movies and ratings given.

Let's begin by creating an input user to recommend movies to:

userInput = [

{'title':'Breakfast Club, The', 'rating':5},

{'title':'Toy Story', 'rating':3.5},

{'title':'Jumanji', 'rating':2},

{'title':"Pulp Fiction", 'rating':5},

{'title':'Akira', 'rating':4.5}

]

inputMovies = pd.DataFrame(userInput)

inputMovies

#### Add movieId to input user

With the input complete, let's extract the input movie's ID's from the movies dataframe and add them into it.

We can achieve this by first filtering out the rows that contain the input movie's title and then merging this subset with the input dataframe. We also drop unnecessary columns for the input to save memory space.

#Filtering out the movies by title

inputId = movies\_df[movies\_df['title'].isin(inputMovies['title'].tolist())]

#Then merging it so we can get the movieId. It's implicitly merging it by title.

inputMovies = pd.merge(inputId, inputMovies)

#Dropping information we won't use from the input dataframe

inputMovies = inputMovies.drop('genres', 1)

#Final input dataframe

#If a movie you added in above isn't here, then it might not be in the original

#dataframe or it might spelled differently, please check capitalisation.

inputMovies

#Filtering out the movies from the input

userMovies = moviesWithGenres\_df[moviesWithGenres\_df['movieId'].isin(inputMovies['movieId'].tolist())]

userMovies

We'll only need the actual genre table, so let's clean this up a bit by resetting the index and dropping the movieId, title, genres and year columns.

#Resetting the index to avoid future issues

userMovies = userMovies.reset\_index(drop=True)

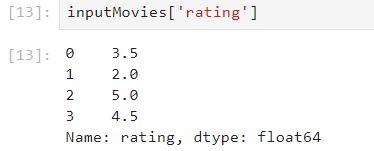
#Dropping unnecessary issues due to save memory and to avoid issues

userGenreTable = userMovies.drop('movieId', 1).drop('title', 1).drop('genres', 1).drop('year', 1)

userGenreTable

Now we're ready to start learning the input's preferences!

To do this, we're going to turn each genre into weights. We can do this by using the input's reviews and multiplying them into the input's genre table and then summing up the resulting table by column. This operation is actually a dot product between a matrix and a vector, so we can simply accomplish by calling Pandas's "dot" function.

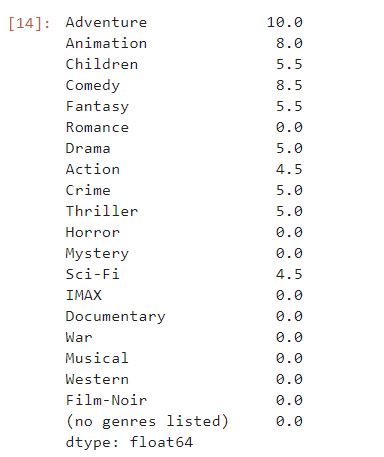


#Dot produt to get weights

userProfile = userGenreTable.transpose().dot(inputMovies['rating'])

#The user profile

userProfile



Let's start by extracting the genre table from the original dataframe:

#Now let's get the genres of every movie in our original dataframe

genreTable = moviesWithGenres\_df.set\_index(moviesWithGenres\_df['movieId'])

#And drop the unnecessary information

genreTable = genreTable.drop('movieId', 1).drop('title', 1).drop('genres', 1).drop('year', 1)

genreTable.head()

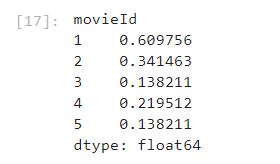


With the input's profile and the complete list of movies and their genres in hand, we're going to take the weighted average of every movie based on the input profile and recommend the top twenty movies that most satisfy it.

#Multiply the genres by the weights and then take the weighted average

recommendationTable\_df = ((genreTable\*userProfile).sum(axis=1))/(userProfile.sum())

recommendationTable\_df.head()

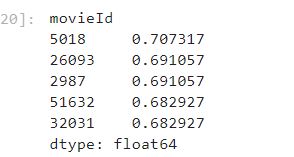


#Sort our recommendations in descending order

recommendationTable\_df = recommendationTable\_df.sort\_values(ascending=False)

#Just a peek at the values

recommendationTable\_df.head()



#The final recommendation table

movies\_df.loc[movies\_df['movieId'].isin(recommendationTable\_df.head(20).keys())]

### Advantages and Disadvantages of Content-Based Filtering

##### Advantages

* Learns user's preferences
* Highly personalized for the user

##### Disadvantages

* Doesn't take into account what others think of the item, so low quality item recommendations might happen
* Extracting data is not always intuitive
* Determining what characteristics of the item the user dislikes or likes is not always obvious