Team IDK Project Proposal

Movie Recommender System

Table of Contents

1 Team Members

Our team comprises:

1. Yingxuan Wu
2. Zijun Li
3. Rena Pei Qi Chong
4. Chua Ming Feng

2 Our Dataset

2.1 Description of dataset

Our dataset (MovieLens 25M, ml-25m) is obtained from MovieLens, a web-based movie recommender service. ml-25m contains 25000095 ratings and 1093360 tag applications across 62423 movies. The data was collected by MovieLens from 162541 unique users between January 09 1995 and November 21 2019, where each user had rated at least 20 movies.

The ml-25m dataset consists of the following files:

1. Tag Genome (genome-scores.csv & genome-tags.csv)
2. Links Data (links.csv)
3. Movies Data (movies.csv)
4. Ratings Data (ratings.csv)
5. Tags Data (tags.csv)

2.2 Breakdown of data files

Data File	Features	Description
Tag Genome (genome-scores.csv & genome-tags.csv)	genome-scores.csv: movieId; tagId; relevance	Contains the tag relevance scores for the movies. Each movie has a value for every tag in the genome.

	genome-tags.csv: tagId; tag	genome-scores.csv provides movie-tag relevance. genome-tags.csv provides the tag descriptions for each tag ID.
Links Data (links.csv)	movieId; imdbId; tmdbId	Can be used to link to other sources of movie data.
Movies Data (movies.csv)	movieId; title; genres	Each movie is labeled with the corresponding genres.
Ratings Data (ratings.csv)	userId; movieId; rating; timestamp	Ratings were made on a scale of 5 stars, with 0.5 star increments. Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1 1970.
Tags Data (tags.csv)	userId; movieId; tag; timestamp	Each movie is given a tag, which could be a single word or short phrase.

This is the source link to our chosen dataset: https://grouplens.org/datasets/movielens/25m/

3 Motivations and Goals of the Project

Recommender systems have become an integral part of our lives, supposedly working seamlessly to suggest new content and material for user consumption. It is a field that is constantly worked and improved on in the industry and in research, and users have thus been able to enjoy the quality of life changes on their favorite platforms. Recommender systems can identify relevant products in the form of suggestions, provide personalized content recommendations etc. For instance, on Amazon, each item listing is accompanied by various other product listings catered for the user. Recommender systems are also integral to companies like Youtube and Netflix, who utilize them to entice and recommend relevant content to users based on the choices they make, such as likes, duration of video elapsed, watch history etc.

Tentatively, our goal is to create a functional movie recommender system built upon item-based collaborative filtering which searches for similar movies and recommends similar movies that the user has not watched. Further improvements will be made after model diagnosis.

4 Exploratory Data Analysis

MovieLens Dataset

Source: https://files.grouplens.org/datasets/movielens/ml-25m-README.html

About the dataset: https://files.grouplens.org/datasets/movielens/ml-25m-README.html

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns
```

movies.csv

Variables: - movieId (integer) - title (string) -> year is in brackets behind - genres (string) -> multiple genres separated by |

```
movies = pd.read_csv("../ml-25m/movies.csv")

movies
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...

62418	209157	We (2018)	Drama
62419	209159	Window of the Soul (2001)	Documentary
62420	209163	Bad Poems (2018)	Comedy|Drama
62421	209169	A Girl Thing (2001)	(no genres listed)
62422	209171	Women of Devil's Island (1962)	Action|Adventure|Drama

```
# strip white spaces from strings

movies["title"] = movies["title"].str.strip()

movies["genres"] = movies["genres"].str.strip()

movies.isnull().values.any()

False
```

Movie release year

We extract the year from the “title” column of the dataframe. Movies without a year specified in the title column will have the “year” column value specified as NaN.

```
# extract movie year from title as new column

movies["year"] = movies["title"].str.extract("(\(\d{4}\))$")

# movies["year"] = movies["title"].str[-6:]

movies["year"] = movies["year"].str[1:5]

movies["year"].unique()

array(['1995', '1994', '1996', '1976', '1992', '1988', '1967', '1993',

        '1964', '1977', '1965', '1982', '1990', '1991', '1989', '1937',
```

```
'1940', '1969', '1981', '1973', '1970', '1960', '1955', '1959',

'1968', '1980', '1975', '1986', '1948', '1943', '1950', '1946',

'1987', '1997', '1974', '1956', '1958', '1949', '1972', '1998',

'1933', '1952', '1951', '1957', '1961', '1954', '1934', '1944',

'1963', '1942', '1941', '1953', '1939', '1947', '1945', '1938',

'1935', '1936', '1926', '1932', '1985', '1979', '1971', '1978',

'1966', '1962', '1983', '1984', '1931', '1922', '1999', '1927',

'1929', '1930', '1928', '1925', '1914', '2000', '1919', '1923',

'1920', '1918', '1921', '2001', '1924', '2002', '2003', '1915',

'2004', '1916', '1917', '2005', '2006', '1902', '1903', '2007',

'2008', '2009', '1912', '2010', nan, '1913', '2011', '1898',

'1899', '1894', '2012', '1910', '2013', '1896', '2014', '2015',

'1895', '1909', '1911', '1900', '2016', '2017', '2018', '2019',

'1905', '1904', '1891', '1892', '1908', '1897', '1887', '1888',

'1890', '1878', '1874', '1901', '1907', '1906', '1883', '1880'],

dtype=object)

movies[movies["year"].isna()] # movies without year labelled
```

	movieId	title	genres	year
15036	79607	Millions Game, The (Das Millionenspiel)	Action|Drama|Sci-Fi|Thriller	NaN
18789	98063	Mona and the Time of Burning Love (Mona ja pal...	Drama	NaN
25387	123619	Terrible Joe Moran	(no genres listed)	NaN

26284	125571	The Court-Martial of Jackie Robinson	(no genres listed)	NaN
26309	125632	In Our Garden	(no genres listed)	NaN
...	...	...	...	...
62071	207714	Tales of Found Footage	(no genres listed)	NaN
62104	207884	Enduring Destiny	(no genres listed)	NaN
62285	208597	Punk the Capital: Building a Sound Movement	Documentary	NaN
62326	208763	Yosemite: The Fate of Heaven	(no genres listed)	NaN
62380	208973	The Falklands War: The Untold Story	(no genres listed)	NaN

From here, we can obtain the number of movies corresponding to each release year.

```
movies["year"].value_counts()

2015    2513

2016    2488

2014    2406

2017    2374

2013    2173

        ...

1883      1

1887    1
```

```
1874    1

1878    1

1880    1

Name: year, Length: 135, dtype: int64

sns.countplot(x="year", data=movies)

<AxesSubplot:xlabel='year', ylabel='count'>
```

Movie genres

Genres for this dataset: Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, IMAX, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western

If there are no genres listed for the movie: (no genres listed)

```
movies[movies["genres"] == "(no genres listed)"]
```

	movieId	title	genres	year
15881	83773	Away with Words (San tiao ren) (1999)	(no genres listed)	1999
16060	84768	Glitterbug (1994)	(no genres listed)	1994
16351	86493	Age of the Earth, The (A Idade da Terra) (1980)	(no genres listed)	1980
16491	87061	Trails (Veredas) (1978)	(no genres listed)	1978
17404	91246	Milky Way (Tejút) (2007)	(no genres listed)	2007
...	...	...	...	...

62400	209101	Hua yang de nian hua (2001)	(no genres listed)	2001
62401	209103	Tsar Ivan the Terrible (1991)	(no genres listed)	1991
62407	209133	The Riot and the Dance (2018)	(no genres listed)	2018
62415	209151	Mao Zedong 1949 (2019)	(no genres listed)	2019
62421	209169	A Girl Thing (2001)	(no genres listed)	2001

There are 5062 movies with no genres listed.

```
movies_with_genre = movies[movies["genres"] != "(no genres listed)"]

movies["genres"] = movies_with_genre["genres"].apply(lambda x: x.split("|"))
```

The genres for each movies is split by the delimeter "|". Applying the above function will split the genres by the delimeter and make all genres of the movie into a list. For movies with "(no genres listed)", the value becomes NaN. We can confirm this in the next two code blocks.

```
movies
```

	movieId	title	genres	year
0	1	Toy Story (1995)	[Adventure, Animation, Children, Comedy, Fantasy]	1995
1	2	Jumanji (1995)	[Adventure, Children, Fantasy]	1995
2	3	Grumpier Old Men (1995)	[Comedy, Romance]	1995

3	4	Waiting to Exhale (1995)	[Comedy, Drama, Romance]	1995
4	5	Father of the Bride Part II (1995)	[Comedy]	1995
...	...	...	...	...
62418	209157	We (2018)	[Drama]	2018
62419	209159	Window of the Soul (2001)	[Documentary]	2001
62420	209163	Bad Poems (2018)	[Comedy, Drama]	2018
62421	209169	A Girl Thing (2001)	NaN	2001
62422	209171	Women of Devil's Island (1962)	[Action, Adventure, Drama]	1962

```
movies[movies["genres"].isnull()] # movies with no genres listed, which are the same as those before the genre column was manipulated
```

	movieId	title	genres	year
15881	83773	Away with Words (San tiao ren) (1999)	NaN	1999
16060	84768	Glitterbug (1994)	NaN	1994
16351	86493	Age of the Earth, The (A Idade da Terra) (1980)	NaN	1980

16491	87061	Trails (Veredas) (1978)	NaN	1978
17404	91246	Milky Way (Tejút) (2007)	NaN	2007
...	...	...	...	...
62400	209101	Hua yang de nian hua (2001)	NaN	2001
62401	209103	Tsar Ivan the Terrible (1991)	NaN	1991
62407	209133	The Riot and the Dance (2018)	NaN	2018
62415	209151	Mao Zedong 1949 (2019)	NaN	2019
62421	209169	A Girl Thing (2001)	NaN	2001

From here, we explore the number of movies categorised into each genre.

```
# genre_count = movies["genres"].apply(lambda x: [i for i in x]).stack().value_counts()

genre_count = movies["genres"].apply(lambda x: pd.Series(x).value_counts()).sum()

genre_count = genre_count.astype("int")

genre_count.sort_index(inplace=True)

genre_count

Action          7348

Adventure       4145

Animation       2929

Children        2935
```

```
Comedy         16870
Crime          5319
Documentary    5605
Drama          25606
Fantasy          2731
Film-Noir      353
Horror         5989
IMAX           195
Musical        1054
Mystery        2925
Romance        7719
Sci-Fi         3595
Thriller       8654
War            1874
Western        1399
dtype: int32
genre_count.plot.bar()
<AxesSubplot:>
```

ratings.csv

Variables: - userId (integer) - movieId (integer) - rating (float) - timestamp (integer) -> seconds since midnight of UTC timezone

```
ratings = pd.read_csv("../ml-25m/ratings.csv")
ratings
```

	userId	movieId	rating	timestamp

0	1	296	5.0	1147880044
1	1	306	3.5	1147868817
2	1	307	5.0	1147868828
3	1	665	5.0	1147878820
4	1	899	3.5	1147868510
...	...	...	...	...
25000090	162541	50872	4.5	1240953372
25000091	162541	55768	2.5	1240951998
25000092	162541	56176	2.0	1240950697
25000093	162541	58559	4.0	1240953434
25000094	162541	63876	5.0	1240952515

```
ratings["datetime"] = pd.to_datetime(ratings["timestamp"], unit="s") # format timestamp to datetime

ratings["year"] = pd.DatetimeIndex(ratings["datetime"]).year # extract year from datetime
```

Spread of year ratings were made

```
ratings["year"].value_counts()
```

2016	1757440
2000	1735398
2017	1689935
2005	1613550
2015	1604971
1996	1430093
2018	1310761
2019	1200634
1999	1059080
2001	1058750
2004	1048116
2006	1038458
2008	1018001
2007	931432
2003	920295
2009	810127
2010	792436
2002	776654
2011	676498
2012	635208
1997	626202
2013	515684
2014	478270
1998	272099

```
1995          3
Name: year, dtype: int64
rating_year_spread = sns.countplot(x="year", data=ratings)
plt.ticklabel_format(style='plain', axis='y')
rating_year_spread.bar_label(rating_year_spread.containers[0],           rotation="vertical",
padding=5)
rating_year_spread.set_xticklabels(rating_year_spread.get_xticklabels(), rotation=45)
None
ratings
```

	userId	movieId	rating	timestamp	datetime	year
0	1	296	5.0	1147880044	2006-05-17 15:34:04	2006
1	1	306	3.5	1147868817	2006-05-17 12:26:57	2006
2	1	307	5.0	1147868828	2006-05-17 12:27:08	2006
3	1	665	5.0	1147878820	2006-05-17 15:13:40	2006
4	1	899	3.5	1147868510	2006-05-17 12:21:50	2006
...	...	...	...	...	...	...
25000090	162541	50872	4.5	1240953372	2009-04-28 21:16:12	2009
25000091	162541	55768	2.5	1240951998	2009-04-28 20:53:18	2009

25000092	162541	56176	2.0	1240950697	2009-04-28 20:31:37	2009
25000093	162541	58559	4.0	1240953434	2009-04-28 21:17:14	2009
25000094	162541	63876	5.0	1240952515	2009-04-28 21:01:55	2009

genome-tags.csv

Variables: - tagId (integer) - tag (text)

```
genome_tags = pd.read_csv("../ml-25m/genome-tags.csv")

genome_tags
```

	tagId	tag
0	1	007
1	2	007 (series)
2	3	18th century
3	4	1920s
4	5	1930s
...	...	...
1123	1124	writing
1124	1125	wuxia
1125	1126	wwii

1126	1127	zombie
1127	1128	zombies

genome-scores.csv

Variables: - movieId (integer) - tagId (integer) - relevance (float) -> score for relevance of tag to the movie

Each movie has a score for every tag

```
genome_scores = pd.read_csv("../ml-25m/genome-scores.csv")
genome_scores
```

	movieId	tagId	relevance
0	1	1	0.02875
1	1	2	0.02375
2	1	3	0.06250
3	1	4	0.07575
4	1	5	0.14075
...	...	...	...
15584443	206499	1124	0.11000
15584444	206499	1125	0.04850
15584445	206499	1126	0.01325

15584446	206499	1127	0.14025
15584447	206499	1128	0.03350

Mean tag relevance score for tags across all movies

```
tag_mean_scores = genome_scores[["tagId", "relevance"]].groupby(["tagId"]).mean()

tag_mean_scores.reset_index(inplace=True)

tag_mean_scores = tag_mean_scores.merge(genome_tags, how="inner", on="tagId")

tag_mean_scores.sort_values("relevance", ascending=False, inplace=True)

tag_mean_scores
```

	tagId	relevance	tag
741	742	0.724424	original
645	646	0.541578	mentor
187	188	0.476752	catastrophe
467	468	0.475400	great ending
971	972	0.450228	storytelling
...	...	...	...
975	976	0.007379	studio ghibli

572	573	0.007345	james bond
116	117	0.007059	batman
118	119	0.005523	beatles
488	489	0.004099	hannibal lecter

tags.csv

This file is to be explored further on how it can be integrated with our project

```
tags = pd.read_csv("../ml-25m/tags.csv")

tags
```

	userId	movieId	tag	timestamp
0	3	260	classic	1439472355
1	3	260	sci-fi	1439472256
2	4	1732	dark comedy	1573943598
3	4	1732	great dialogue	1573943604
4	4	7569	so bad it's good	1573943455
...	...	...	...	...

1093355	162521	66934	Neil Patrick Harris	1427311611
1093356	162521	103341	cornetto trilogy	1427311259
1093357	162534	189169	comedy	1527518175
1093358	162534	189169	disabled	1527518181
1093359	162534	189169	robbery	1527518193

links.csv

Variables: - movieId - imdbId - tmdbId

Potential use of this file: merging with other datasets (TMDB or IMDB) to obtain more information about the movie

```
links = pd.read_csv("../ml-25m/links.csv")

links
```

	movieId	imdbId	tmdbId
0	1	114709	862.0
1	2	113497	8844.0
2	3	113228	15602.0
3	4	114885	31357.0
4	5	113041	11862.0
...	...	...	...

62418	209157	6671244	499546.0
62419	209159	297986	63407.0
62420	209163	6755366	553036.0
62421	209169	249603	162892.0
62422	209171	55323	79513.0

5 Project Techniques

Preliminary ideas on techniques the team will apply to achieve your goals

We have done our preliminary research on commonly used recommender systems and have decided on building our project using the item-based collaborative filtering approach in the initial stage. This approach would search for similar movies and recommend movies that the user has not watched. The similarity measure can be computed using cosine similarity, which measures the degree of similarity between 2 non-zero vectors defined in an inner product space.

$$cosine\ similarity = \cos(\theta) = \frac{A\dot{B}}{||A||\,||B||}$$

This filtering technique functions by searching for every pair of movies that were rated by the same user and measures the cosine similarity of those rated across all users who rated both movies. We can implement this approach using the K-Nearest Neighbors (KNN) algorithm for prediction of potential relevant movies for the user.

The data frame would have to be of $m \times n$ dimension, where m and n represent the number of movies and number of users respectively. We can represent the ratings of a movie as a vector (rating vector) in n-dimensional space. The data in the data frame is expected to be sparse as it is likely for there to be movies that are rated by a small number of users. The converse is also true where there is likely to be users who have only rated a small number of movies. Thus, it is important for us to address this data sparsity when training our model.

When presented with a target movie from the user, the KNN algorithm can calculate the distance between the rating vector of this targeted movie to all other rating vectors. The distances are ranked and the algorithm returns the top K nearest movies as the most similar movie recommendations.