STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

CS 593:
Data Mining II: Advanced Algorithms for Mining Big Data

Final Project
Recommendation System for Movies

Team Members

Noah Suttora

Aditi Duggal

Michelle Morrone

Dataset Explanation

Dataset link: https://grouplens.org/datasets/movielens/25m/

- Source: MovieLens 25M Dataset
- Number of ratings: 25M
- Number of Movies: 62,423
- Number of users: 162,541
- 2 CSV files:

movies.csv

	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

ratings.csv

	userId	movieId	rating
0	1	296	5.0
1	1	306	3.5
2	1	307	5.0
3	1	665	5.0
4	1	899	3.5

Initial Pre-processing

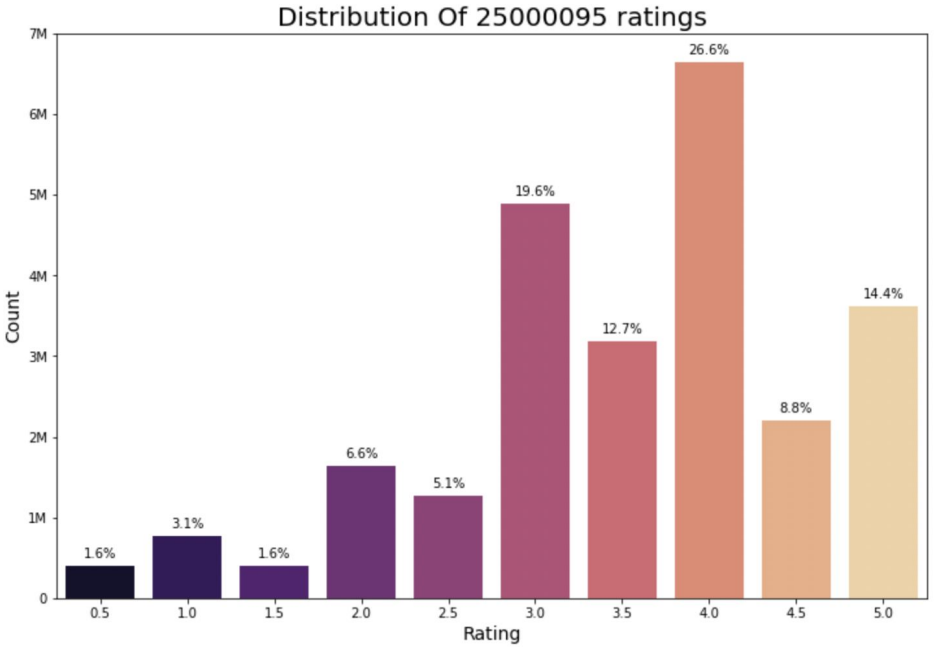
- Extracted the movie year from the movie title and appended to new column which holds the year of the movie
- Cleaned the movie title
- The duplicate movies with same movie title and year were dropped from the movies.csv

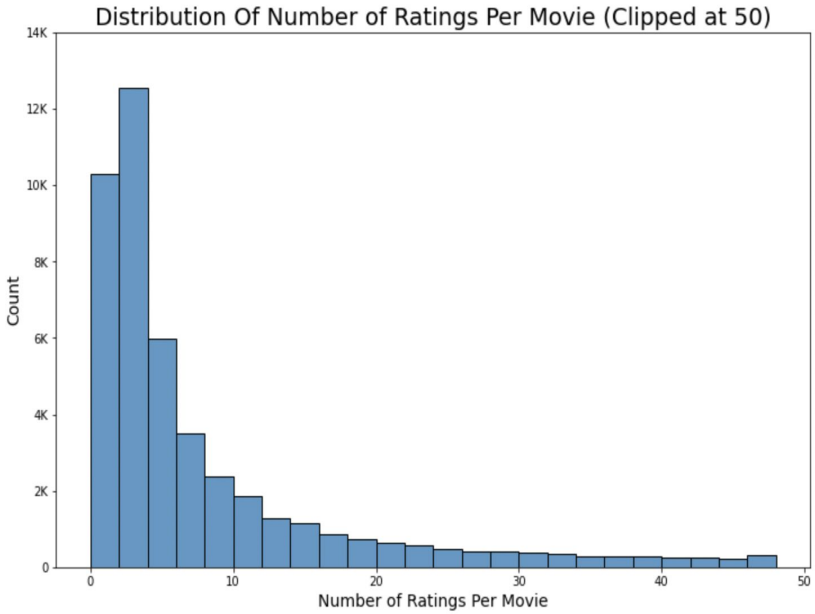
	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	
3	4	Waiting to Exhale (1995)	
4	5	Father of the Bride Part II (1995)	

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

EDA

Distribution Graph of 25M Ratings

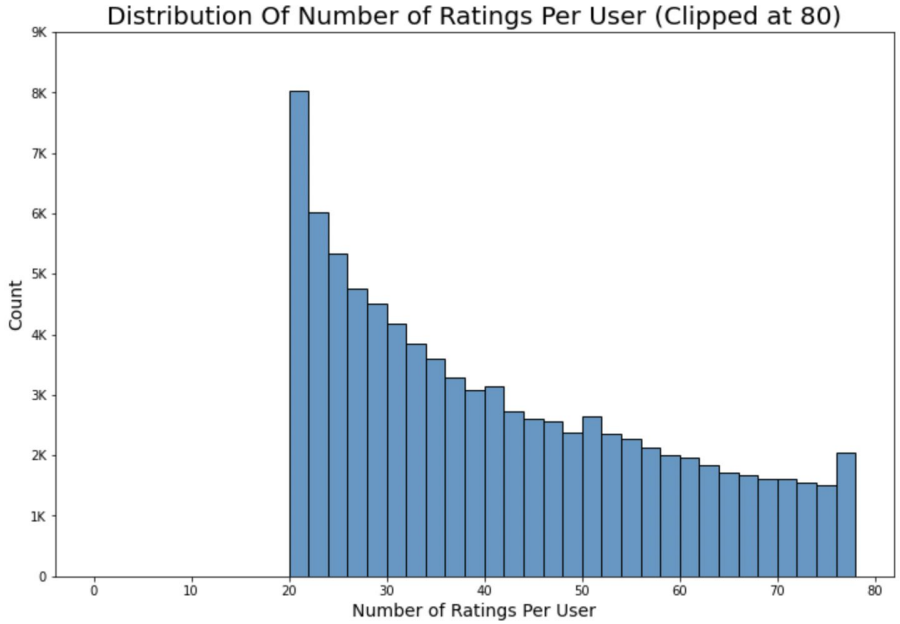


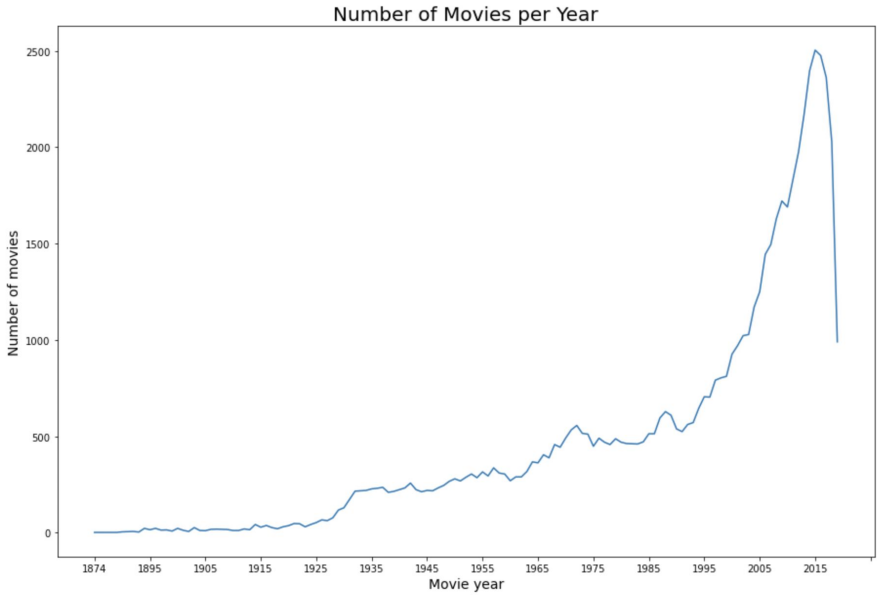


Distribution Graph of number of ratings per Movie

EDA

Distribution Graph for number of Ratings per User





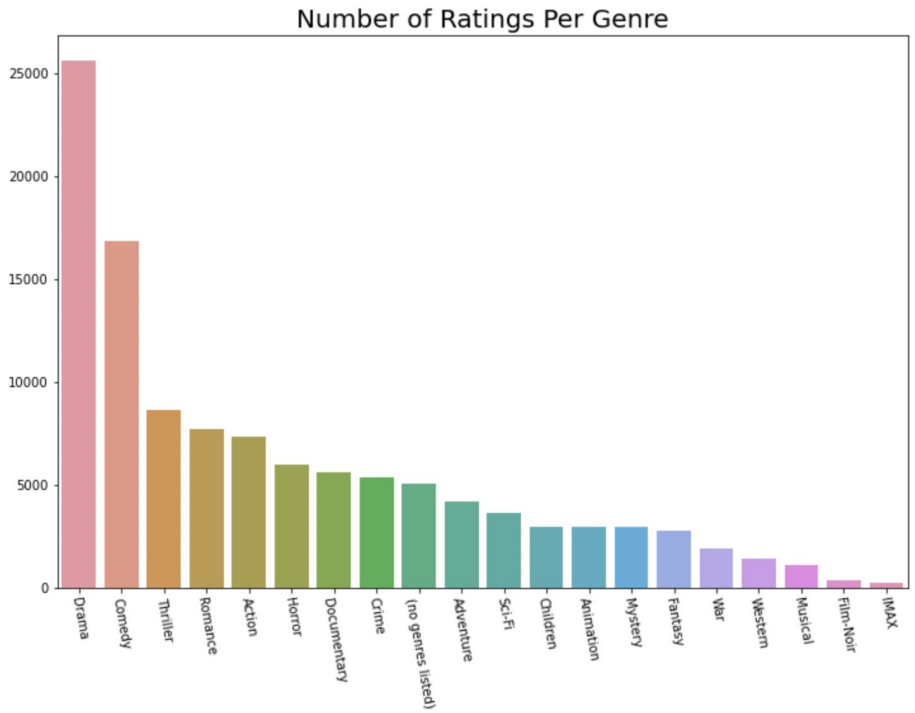
Number of Movies per year

EDA

Count of each Genre

	count
Drama	25569
Comedy	16849
Thriller	8631
Romance	7708
Action	7340
Horror	5973
Documentary	5596
Crime	5311
(no genres listed)	5052

Adventure	4140
Sci-Fi	3587
Children	2932
Animation	2927
Mystery	2924
Fantasy	2727
War	1873
Western	1399
Musical	1054
Film-Noir	353
IMAX	195



Number of Ratings per Genre

Preprocessing Dataframes

In [15]:

- The ratings and movie dataframes were combined together to be used for the recommendation system

	movieId	title	genres	year	userId	rating
0	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	1995	2	3.5
1	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	1995	3	4.0
2	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	1995	4	3.0
3	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	1995	5	4.0
4	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	1995	8	4.0
...	...	...	...	...	...	...
24998451	209157	We	Drama	2018	119571	1.5
24998452	209159	Window of the Soul	Documentary	2001	115835	3.0
24998453	209163	Bad Poems	Comedy|Drama	2018	6964	4.5
24998454	209169	A Girl Thing	(no genres listed)	2001	119571	3.0
24998455	209171	Women of Devil's Island	Action|Adventure|Drama	1962	119571	3.0

24998456 rows × 6 columns

Concepts & Algorithms Used

- Sparseness to efficiently represent the gigantic dataset
- Content-Based Filtering
- Collaborative Filtering
- Matrix Factorization Using SVD

Sparse Representation of Big Data

- For the utility matrix with 162,541 rows for the unique users and 59,047 columns for the unique movies, 71.5GB of memory storage would be needed
- Since the utility matrix is extremely sparse, we employed a sparse representation for this matrix that could easily and efficiently be held on a device, along with efficient computation on this matrix
- Compressed Sparse Row (CSR) format used for utility matrix
  - Represents a matrix by 3 one-dimensional arrays, that respectively contain nonzero values, the extents of rows, and the column indices
  - Allows for fast row access and matrix-vector multiplication

$$\begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 6 & 0 & 0 \end{pmatrix}$$

```
V         = [ 5 8 3 6 ]
COL_INDEX = [ 0 1 2 1 ]
ROW_INDEX = [ 0 1 2 3 4 ]
```

```
# sparsity measure of utility matrix
nonzero_count = util_mat_sparse.count_nonzero()
total_elements = util_mat_sparse.shape[0] * util_mat_sparse.shape[1]
sparsity = 1.0 - (nonzero_count / total_elements)
print('Number of Non-Zero Elements of Utility Matrix: ', nonzero_count)
print('Total Number of Elements of Utility Matrix:    ', total_elements)
print('Percent Sparseness of Utility Matrix:          {:.2%}'.format(sparsity))
```

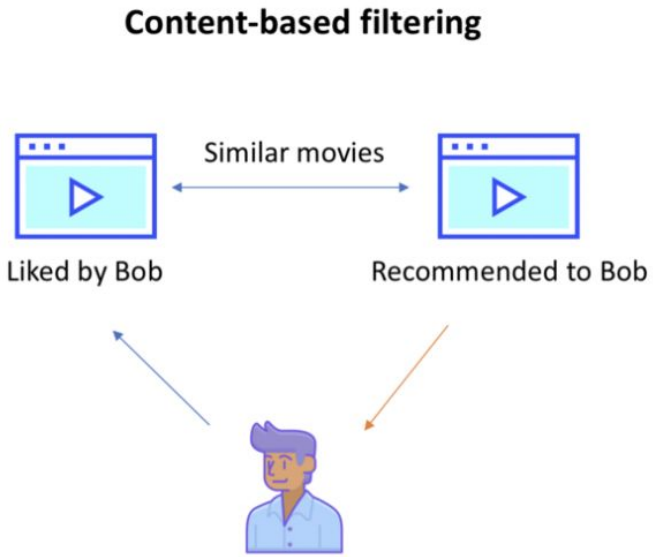
✓ 0.1s

```
Number of Non-Zero Elements of Utility Matrix:  24998456
Total Number of Elements of Utility Matrix:     9582767196
Percent Sparseness of Utility Matrix:           99.74%
```

Content-Based Filtering

- Content based recommenders use data exclusively about the items.
- For this we need to have a minimal understanding of the users' preferences, so that we can then recommend new items with similar keywords (here we use genres) to those specified by the user.
- TF-IDF vectorizer possesses some interesting properties that are useful in order to obtain a vector representation of the data.

- $tfidf_{i,j} = tf_{i,j} \times \log(N/df_i)$
  - $tf_{i,j}$ is total number of occurrences of i in j
  - df_i total number of documents containing i
  - N is the total number of documents
- Here we have the product of the term frequency, i.e. the amount of times a given term (genre) occurs in a document (genres of a movie), times the right side factor, which basically scales the term frequency depending on the amount of times a given term appears in all documents (movies).



Content-Based Filtering

- TF-IDF will help capture the important genres of each movie by giving a higher weight to the less frequent genres
- To obtain the tf-idf vectors we used sklearn's TfidfVectorizer
- Following shows the tf-idf vectors

```
tfidf_df.head()
```

title	(no genres listed)	action	adventure	animation	children	comedy	crime	documentary	drama	fantasy	film-noir	horror
Toy Story	0.0	0.0	0.446566	0.48833	0.488084	0.277717	0.0	0.0	0.00000	0.496748	0.0	0.0
Jumanji	0.0	0.0	0.539795	0.00000	0.589981	0.000000	0.0	0.0	0.00000	0.600454	0.0	0.0
Grumpier Old Men	0.0	0.0	0.000000	0.00000	0.000000	0.598464	0.0	0.0	0.00000	0.000000	0.0	0.0
Waiting to Exhale	0.0	0.0	0.000000	0.00000	0.000000	0.537355	0.0	0.0	0.44022	0.000000	0.0	0.0
Father of the Bride Part II	0.0	0.0	0.000000	0.00000	0.000000	1.000000	0.0	0.0	0.00000	0.000000	0.0	0.0

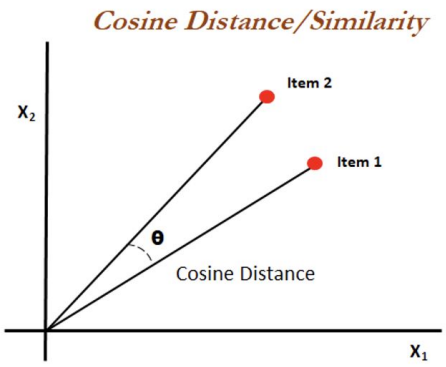
Content-Based Filtering

- The next step will be to find similar vectors (movies)
- similarity = $\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$
- Here we'll be obtaining the cosine by taking the inner product between both vectors, and normalising by their respective magnitudes
- Sklearn.metrics.pairwise has pairwise distance metrics, cosine_similarity to compute cosine similarities between all tf-idf vectors

```
cosine_sim_df.sample(5, axis=1).round(2)
```

```
Shape: (3883, 3883)
```

title	Me, Myself and Irene (2000)	3 Ninjas: High Noon On Mega Mountain (1998)	Towering Inferno, The (1974)	Normal Life (1996)	Saturn 3 (1979)
title					
Toy Story (1995)	0.34	0.46	0.00	0.00	0.00
Jumanji (1995)	0.00	0.40	0.00	0.00	0.30
Grumpier Old Men (1995)	0.57	0.00	0.00	0.00	0.00
Waiting to Exhale (1995)	0.76	0.00	0.34	0.28	0.00
Father of the Bride Part II (1995)	1.00	0.00	0.00	0.00	0.00
...	...	...	...	...	...
Meet the Parents (2000)	1.00	0.00	0.00	0.00	0.00
Requiem for a Dream (2000)	0.00	0.00	0.53	0.43	0.00
Tigerland (2000)	0.00	0.00	0.53	0.43	0.00
Two Family House (2000)	0.00	0.00	0.53	0.43	0.00
Contender, The (2000)	0.00	0.00	0.28	0.23	0.44



Content-Based Filtering

- To find the highest weights or tf-idf scores for a given movie, argpartition() is used to get the k highest values in similarity matrix.
- We then index on the columns in similarity matrix, and further slice to get from highest weights to lowest.

k=15

```
genre_recommendations('Toy Story', cos_sim_df, movies[['title', 'genres','year']], 15)
```

	title	genres	year
0	Missing Link	Adventure|Animation|Children|Comedy|Fantasy	2019
1	UglyDolls	Adventure|Animation|Children|Comedy|Fantasy	2019
2	Scooby-Doo! Mask of the Blue Falcon	Adventure|Animation|Children|Comedy|Fantasy	2012
3	Shrek the Third	Adventure|Animation|Children|Comedy|Fantasy	2007
4	Puss in Book: Trapped in an Epic Tale	Adventure|Animation|Children|Comedy|Fantasy	2017
5	Moana	Documentary	1926
6	Moana	Adventure|Animation|Children|Comedy|Fantasy	2016
7	Wonder Park	Adventure|Animation|Children|Comedy|Fantasy	2019
8	Brother Bear 2	Adventure|Animation|Children|Comedy|Fantasy	2006
9	The Magic Crystal	Adventure|Animation|Children|Comedy|Fantasy	2011
10	DuckTales: The Movie - Treasure of the Lost Lamp	Adventure|Animation|Children|Comedy|Fantasy	1990
11	Penguin Highway	Adventure|Animation|Children|Comedy|Fantasy	2018
12	The Good Dinosaur	Adventure|Animation|Children|Comedy|Fantasy	2015
13	The Dragon Spell	Adventure|Animation|Children|Comedy|Fantasy	2016
14	Boxtrolls, The	Adventure|Animation|Children|Comedy|Fantasy	2014

k=20

```
genre_recommendations('Saving Private Ryan', cos_sim_df,
                      movies[['title', 'genres','year']], 20)
```

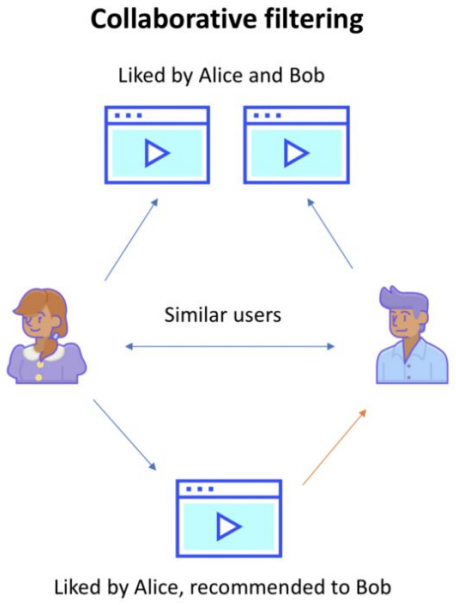
	title	genres	year
0	The Battalion	Action|Drama|War	2015
1	Seas Beneath	Action|Drama|War	1931
2	Victory (a.k.a. Escape to Victory)	Action|Drama|War	1981
3	Tali-Ihantala 1944	Action|Drama|War	2007
4	The Fortress	Action|Drama|War	2017
5	Bridge Too Far, A	Action|Drama|War	1977
6	The Great Raid	Action|Drama|War	2005
7	Chittagong	Action|Drama|War	2012
8	Seal Team Eight: Behind Enemy Lines	Action|Drama|War	2014
9	Hamburger Hill	Action|Drama|War	1987
10	Tunnel Rats (1968 Tunnel Rats)	Action|Drama|War	2008
11	Battleground	Action|Drama|War	1949
12	Thin Red Line, The	Action|Drama|War	1998
13	Beaufort	Action|Drama|War	2007
14	Brest Fortress (Brestskaya krepost)	Action|Drama|War	2010
15	Jarhead	Action|Drama|War	2005
16	Zulu	Action|Drama|War	1964
17	Zulu	Crime|Drama|Thriller	2013
18	New One-Armed Swordsman, The (Xin du bi dao)	Action|Drama|War	1971
19	Flat Top	Action|Drama|War	1952

Collaborative Filtering

- Collaborative Filtering provides recommendations based on the interests of similar users
  - If two users like many of the same movies, then the movies that only one of them has liked could be recommended to the other user
- For this, we focus on the user ratings for the movies
  - We just need to use userId, movieId, ratings, and titles
- A new pivot table was created to show the ratings given by each user for each movie
- This was then converted to a CSR matrix

userId	1	2	3	4	5	6	7	8	9	10	...	162532	162533	162534
title														
'Til There Was You	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
'burbs, The	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
1-900 (06)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
101 Dalmatians	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
12 Angry Men	0.0	0.0	0.0	0.0	0.0	5.0	0.0	4.0	0.0	0.0	...	0.0	0.0	4.5
187 (One Eight Seven)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
2 Days in the Valley	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
2 ou 3 choses que je sais d'elle (2 or 3 Things I Know About Her)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
20,000 Leagues Under the Sea	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
2001: A Space Odyssey	0.0	0.0	5.0	4.0	0.0	4.0	0.0	5.0	3.0	4.5	...	0.0	0.0	5.0

10 rows × 161577 columns



Collaborative Filtering

- To make the recommendations, a K Nearest Neighbors model was used
  - It is ideal for this type of problem since it looks for similar points (or users and movies in this case)
- The below model was created
  - nearest neighbors was set to 16 and the metric used was cosine
- The model was then trained with the sparse matrix

```
model = NearestNeighbors(n_neighbors=16, algorithm='brute', metric='cosine')
```

Collaborative Filtering

- Using the trained model, recommendations can then be made for specific movies or users based on what other users with similar tastes enjoyed

```
collabrec('Toy Story')
```

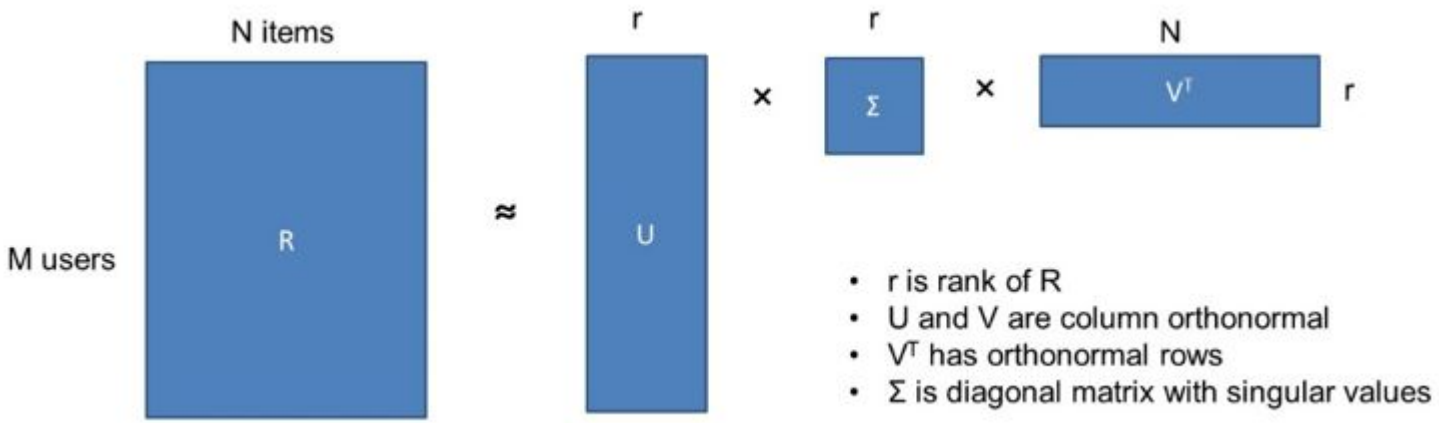
```
Index(['Star Wars: Episode IV - A New Hope', 'Back to the Future',
       'Forrest Gump', 'Jurassic Park',
       'Star Wars: Episode VI - Return of the Jedi',
       'Independence Day (a.k.a. ID4)', 'Lion King, The',
'Aladdin',
       'Star Wars: Episode V - The Empire Strikes Back',
       'Men in Black (a.k.a. MIB)',
       'Raiders of the Lost Ark (Indiana Jones and the Raiders of
the Lost Ark)',
       'Pulp Fiction', 'Groundhog Day', 'Mission: Impossible',
       'Willy Wonka & the Chocolate Factory'],
      dtype='object', name='title')
```

```
collabrec('Forrest Gump')
```

```
Index(['Jurassic Park', 'Shawshank Redemption, The',
       'Silence of the Lambs, The', 'Pulp Fiction',
'Braveheart',
       'Terminator 2: Judgment Day', 'Schindler's List',
'Apollo 13',
       'Fugitive, The', 'Seven (a.k.a. Se7en)', 'Lion King,
The', 'Speed',
       'Mrs. Doubtfire', 'Back to the Future', 'Toy Story'],
      dtype='object', name='title')
```

Matrix Factorization using SVD

- Type of collaborative filtering method
- Singular Value Decomposition (SVD) is an algorithm that decomposes a matrix into the best lower rank (smaller/simpler) approximation of the original matrix.
- $A = U\Sigma V^T$
  - A is the utility matrix
  - U is the user features matrix, representing how much each user likes each feature
  - Σ is the diagonal matrix of weights (singular values)
  - V^T is the movie features matrix, representing how relevant each feature is to each movie
- After applying SVD on the CSR utility matrix, reconstruct the utility matrix using U, Σ, and V^T to create a matrix containing movie rating predictions for every user



Matrix Factorization using SVD

- Python package scipy has a function called ‘svds’ that can directly calculate the SVD of a sparsely-represented matrix with the desired number of singular values to compute
- With the reconstruction using simple math, we can directly apply cosine similarity to get accurate recommendations for a given user

```
svd_recommendation(userId=1)
```

	movieId	title	genres	year
1171	1201	Good, the Bad and the Ugly, The (Buono, il bru...	Action|Adventure|Western	1966
3897	4001	Code of Silence	Action	1985
4399	4504	Feds	Comedy	1988
4584	4689	Cat o' Nine Tails, The (Gatto a nove code, Il)	Mystery|Thriller	1971
5243	5351	Warm Water Under a Red Bridge (Akai hashi no s...	Comedy|Drama	2001
5272	5380	Importance of Being Earnest, The	Comedy|Drama|Romance	2002

```
svd_recommendation(userId=100)
```

	movieId	title	genres	year
1472	1524	Turning, The	Drama	1992
4405	4510	Heartbreak Hotel	Comedy	1988
4674	4780	Liam	Drama	2000
5900	6012	Guy Thing, A	Comedy|Romance	2003
6484	6607	Red Pony, The	Drama	1949
6516	6639	Wait Until Dark	Drama|Thriller	1967
7544	7934	Zelig	Comedy	1983

Conclusion/Future Scope

- All of these methods use the same data to make different recommendations to users
- Each method is good for a different type of recommendation
  - This leads to many possible recommendations for each user and a higher likelihood that each user will find a movie that they like within the recommendations
- These methods can be combined together to make a comprehensive movie recommendation system
  - Similar to the types of systems used by Netflix and other streaming services
- Partial or comprehensive recommendation systems are useful for both consumers and providers
  - It can be convenient for users to be given a subset of options they are more likely to enjoy
  - Giving recommendations to users makes users more likely to watch other movies which is good for the streaming services providing those movies
- Recommendation systems using these methods can also be applied to many other goods and services besides movies
  - Similar data (users, items/services, category, ratings) just needs to be available
  - For example, shopping websites like Amazon can and do use these recommendation systems

STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

stevens.edu

Thank You