

```
In [1]: !pip install scikit-surprise
```

DEPRECATION: Configuring installation scheme with distutils config files is deprecated and will no longer work in the near future. If you are using a Homebrew or Linuxbrew Python, please see discussion at <https://github.com/Homebrew/homebrew-core/issues/76621>

Requirement already satisfied: scikit-surprise in /usr/local/lib/python3.9/site-packages (1.1.1)

Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.9/site-packages (from scikit-surprise) (1.6.0)

Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.9/site-packages (from scikit-surprise) (1.0.0)

Requirement already satisfied: six>=1.10.0 in /usr/local/Cellar/protobuf/3.14.0/libexec/lib/python3.9/site-packages (from scikit-surprise) (1.15.0)

Requirement already satisfied: numpy>=1.11.2 in /usr/local/lib/python3.9/site-packages (from scikit-surprise) (1.22.3)

DEPRECATION: Configuring installation scheme with distutils config files is deprecated and will no longer work in the near future. If you are using a Homebrew or Linuxbrew Python, please see discussion at <https://github.com/Homebrew/homebrew-core/issues/76621>

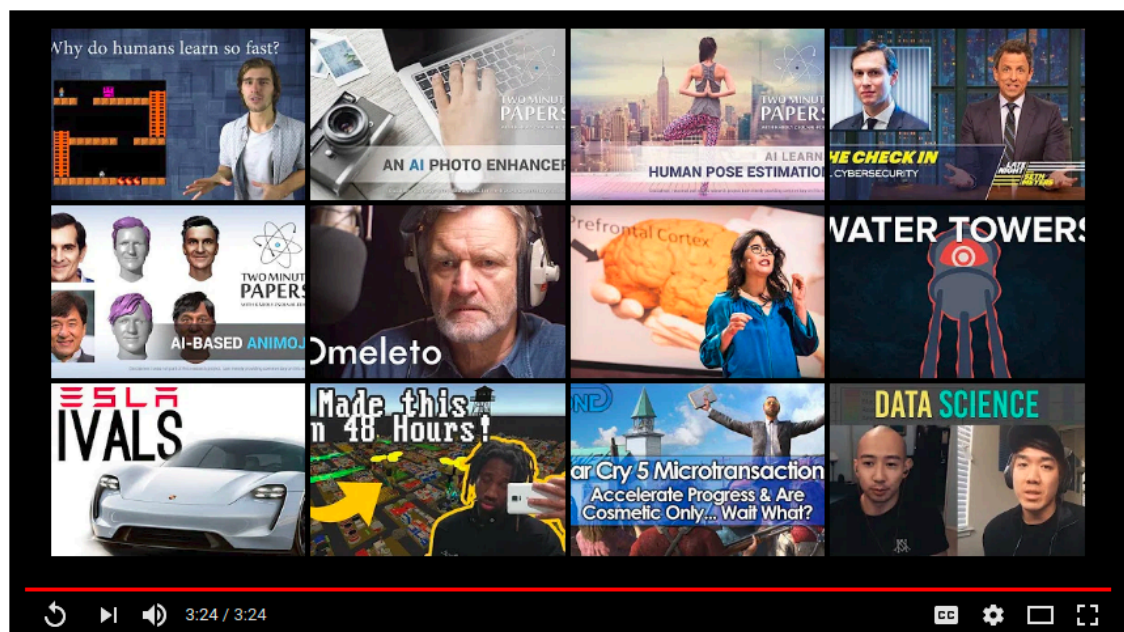
## Index

- Introduction to Recommendation System
- Exploratory Data Analysis(EDA)
- Content based filtering
- Collaborative Filtering
  - Memory based collaborative filtering
    - User-Item Filtering
    - Item-Item Filtering
  - Model based collaborative filtering
    - Single Value Decomposition(SVD)
    - SVD++
- Evaluating Collaborative Filtering using SVD
- Hybrid Model

## Introduction to Recommendation System



A recommendation system is any system that automatically suggests content, products or services which should interest customers based on their preferences.



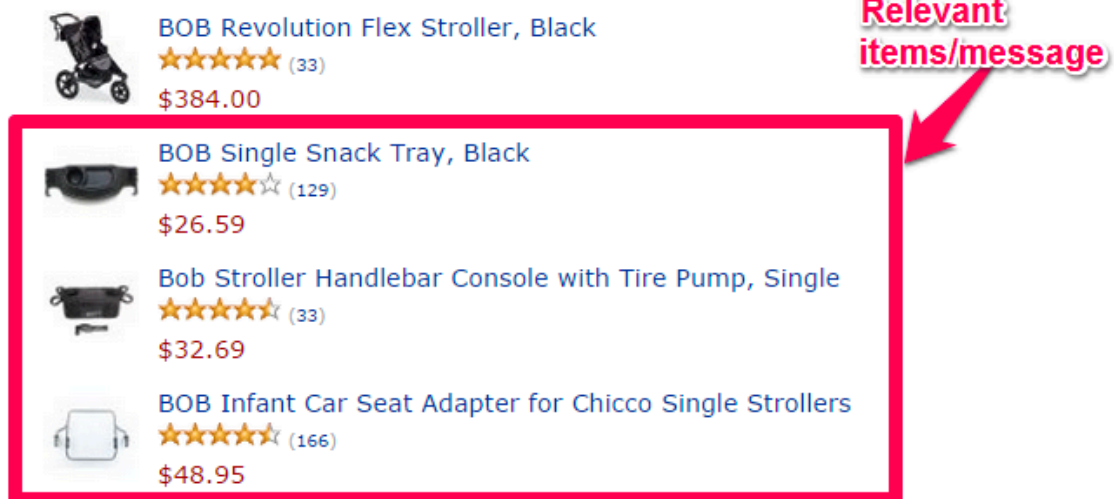
Why recommendation system is useful for organization?

- Help to increase the site's page views, dwell time, click-through rate, and retention
- Help generate more advertising revenue

- Increase upselling and crossselling

### What Other Items Do Customers Buy After Viewing This Item?

**Relevant items/message**



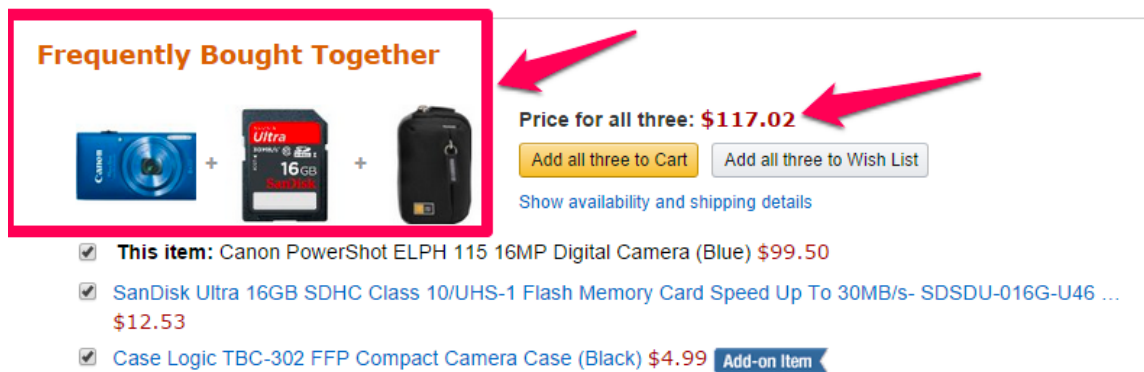
BOB Revolution Flex Stroller, Black  
★★★★★ (33)  
\$384.00

BOB Single Snack Tray, Black  
★★★★☆ (129)  
\$26.59

Bob Stroller Handlebar Console with Tire Pump, Single  
★★★★★ (33)  
\$32.69

BOB Infant Car Seat Adapter for Chicco Single Strollers  
★★★★★ (166)  
\$48.95

**Frequently Bought Together**



Price for all three: **\$117.02**

Add all three to Cart Add all three to Wish List

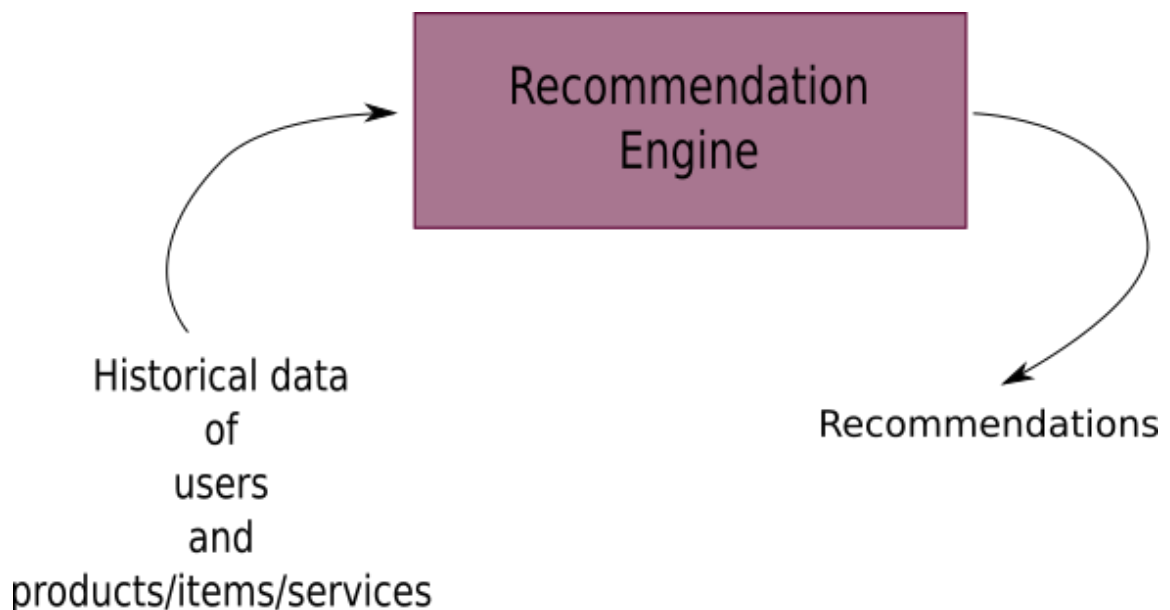
Show availability and shipping details

☒ **This item:** Canon PowerShot ELPH 115 16MP Digital Camera (Blue) \$99.50

☒ SanDisk Ultra 16GB SDHC Class 10/UHS-1 Flash Memory Card Speed Up To 30MB/s- SDSDU-016G-U46 ... \$12.53

☒ Case Logic TBC-302 FFP Compact Camera Case (Black) \$4.99 [Add-on Item](#)

### How we build Recommendation Engine ?



### Movie recommendation system

Recommender systems are one of the most successful and widespread application of machine learning technologies in business. You can find large scale recommender systems in retail, video on demand, or music streaming.

- **Examples of recommendation systems are:**

1. Offering news articles to on-line newspaper readers, based on a prediction of reader interests.
2. Offering customers of an on-line retailer suggestions about what they might like to buy, based on their past history of purchases and/or product searches.
3. Recommending movies to user based on there previous watch

## Load Libraries

```
In [2]: from math import sqrt
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
```

## Dataset : Movielens

<https://grouplens.org/datasets/movielens/100k>

```
In [3]: # Reading ratings file
ratings = pd.read_csv('ratings.csv', sep=',', encoding='latin-1', usecols=[

# Reading movies file
movies = pd.read_csv('movies.csv', sep=',', encoding='latin-1', usecols=['m
```

```
In [4]: df_movies = movies
df_ratings = ratings
```

```
In [5]: df_movies
```

Out [5]:

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
...	...	...	...
9737	193581	Black Butler: Book of the Atlantic (2017)	Action Animation Comedy Fantasy
9738	193583	No Game No Life: Zero (2017)	Animation Comedy Fantasy
9739	193585	Flint (2017)	Drama
9740	193587	Bungo Stray Dogs: Dead Apple (2018)	Action Animation
9741	193609	Andrew Dice Clay: Dice Rules (1991)	Comedy

9742 rows × 3 columns

In [6]: df\_ratings

Out [6]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
...	...	...	...	...
100831	610	166534	4.0	1493848402
100832	610	168248	5.0	1493850091
100833	610	168250	5.0	1494273047
100834	610	168252	5.0	1493846352
100835	610	170875	3.0	1493846415

100836 rows × 4 columns

## Exploratory Data Analysis(EDA)

In [7]: df\_movies.head(5)

Out [7]:

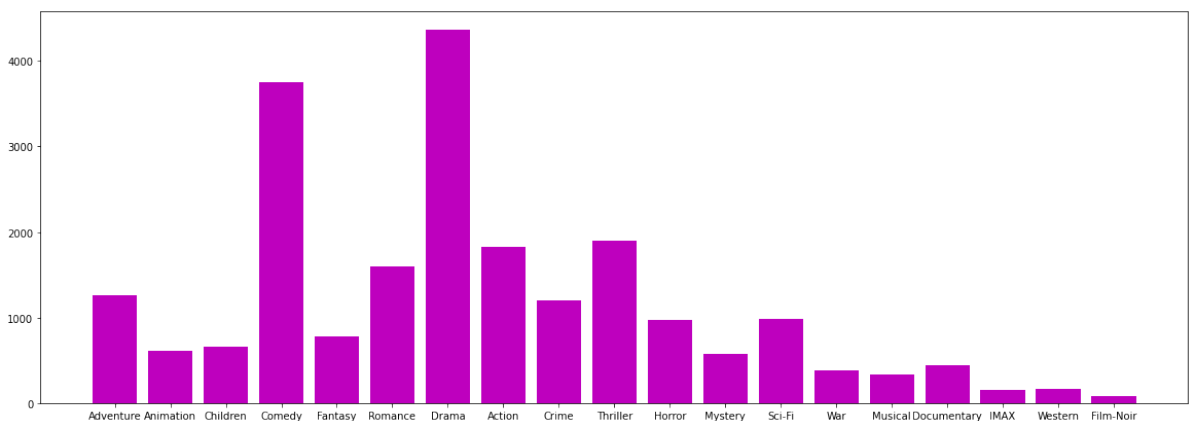
	movieId	title	genres
0	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
1	2	Jumanji (1995)	Adventure Children Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama Romance
4	5	Father of the Bride Part II (1995)	Comedy

## Most popular genres of movie released

```
In [8]: plt.figure(figsize=(20,7))
generlist = df_movies['genres'].apply(lambda generlist_movie : str(generlist_movie))
generlist_movie = {}

for generlist_movie in generlist:
    for gener in generlist_movie:
        if(generlist_movie.get(gener,False)):
            generlist_movie[gener]=generlist_movie[gener]+1
        else:
            generlist_movie[gener] = 1
generlist_movie.pop("(no genres listed)")
plt.bar(generlist_movie.keys(),generlist_movie.values(),color='m')
```

Out [8]: <BarContainer object of 19 artists>



```
In [9]: df_ratings.head(5)
```

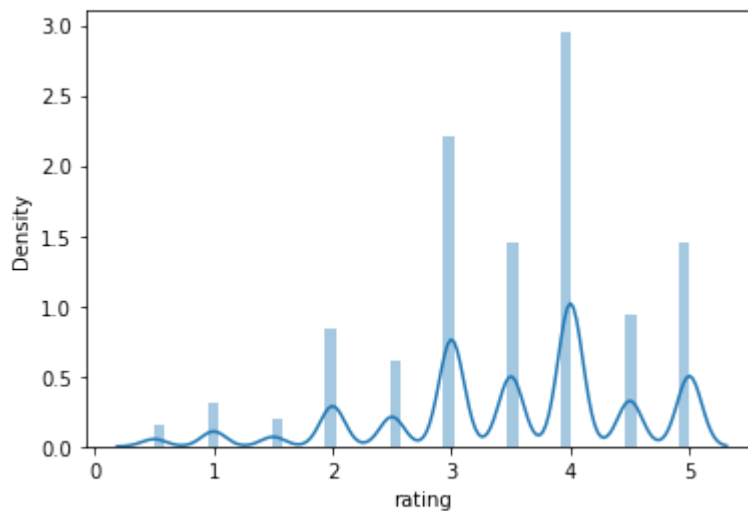
Out [9]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

## Distribution of users rating

```
In [10]: sns.distplot(df_ratings["rating"]);
```

```
/usr/local/lib/python3.9/site-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
warnings.warn(msg, FutureWarning)
```



```
In [11]: print("Shape of frames: \n"+ " Rating DataFrame"+ str(df_ratings.shape)+"\n")
```

```
Shape of frames:
Rating DataFrame(100836, 4)
Movies DataFrame(9742, 3)
```

```
In [12]: merge_ratings_movies = pd.merge(df_movies, df_ratings, on='movieId', how='inner')
```

```
In [13]: merge_ratings_movies.head(2)
```

```
Out[13]:
```

	movieId	title	genres	userId	rating	timestamp
0	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy	1	4.0	964982703
1	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy	5	4.0	847434962

```
In [14]: merge_ratings_movies = merge_ratings_movies.drop('timestamp', axis=1)
```

```
In [15]: merge_ratings_movies.shape
```

```
Out[15]: (100836, 5)
```

Grouping the rating based on user

```
In [16]: ratings_grouped_by_users = merge_ratings_movies.groupby('userId').agg([np.sum, np.mean])
```

```
In [17]: ratings_grouped_by_users.head(2)
```

Out[17]:

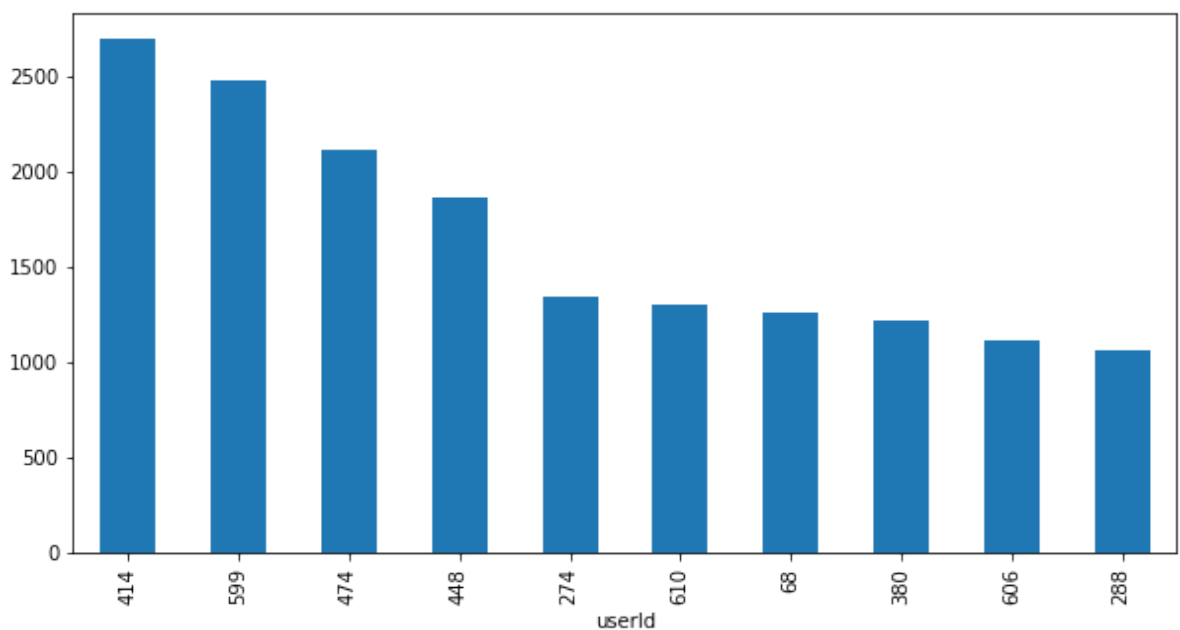
	movieId		rating	
	size	mean	size	mean
userId				
1	232	1854.603448	232.0	4.366379
2	29	70350.275862	29.0	3.948276

In [18]: ratings\_grouped\_by\_users = ratings\_grouped\_by\_users.drop('movieId', axis = 1)

## Top 10 users who have rated most of the movies

In [19]: ratings\_grouped\_by\_users['rating']['size'].sort\_values(ascending=False).head(10)

Out[19]: &lt;AxesSubplot:xlabel='userId'&gt;



In [20]: ratings\_grouped\_by\_movies = merge\_ratings\_movies.groupby('movieId').agg([np

In [21]: ratings\_grouped\_by\_movies.shape

Out[21]: (9724, 2)

In [22]: ratings\_grouped\_by\_movies.head(3)

Out[22]:

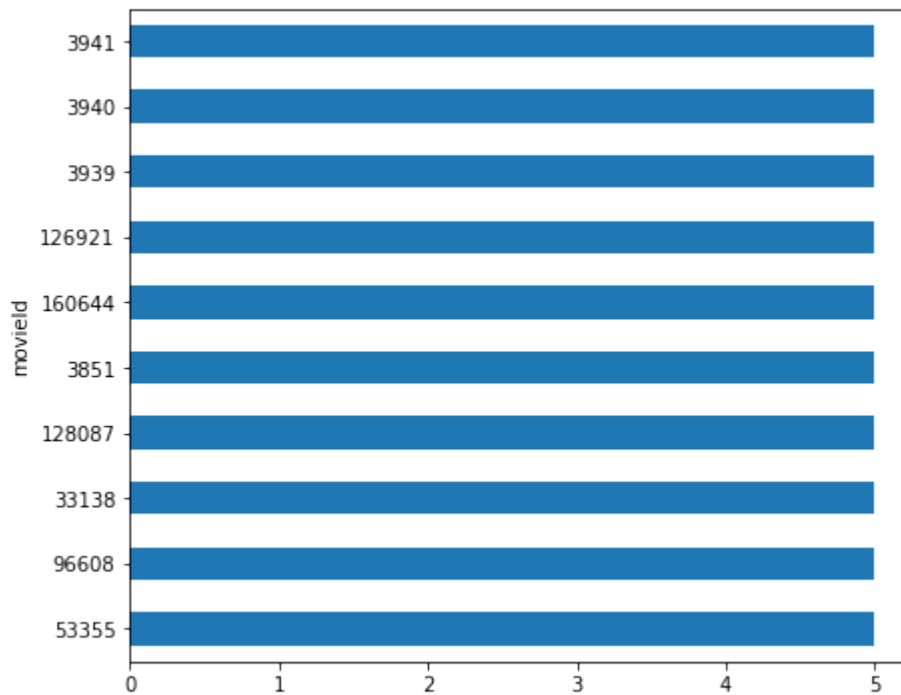
	userId	rating
	mean	mean
movieId		
1	306.530233	3.920930
2	329.554545	3.431818
3	283.596154	3.259615

In [23]: ratings\_grouped\_by\_movies = ratings\_grouped\_by\_movies.drop('userId', axis=1)

## Movies with high average rating



```
In [24]: ratings_grouped_by_movies['rating']['mean'].sort_values(ascending=False).head(10)
```

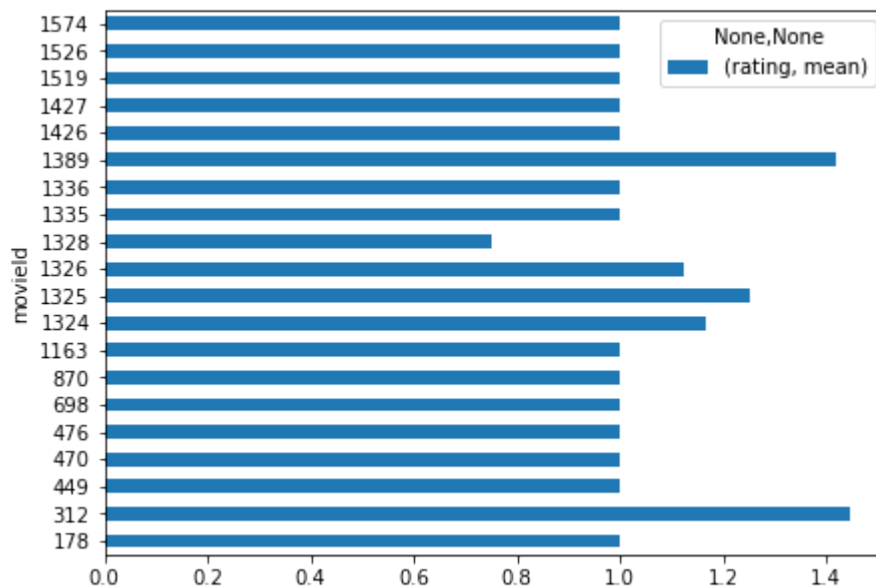


## Movies with low average rating

```
In [25]: lowRatedMoviesFilter = ratings_grouped_by_movies['rating']['mean'] < 1.5
```

```
In [26]: lowRatedMovies = ratings_grouped_by_movies[lowRatedMoviesFilter]
```

```
In [27]: lowRatedMovies.head(20).plot(kind='barh', figsize=(7,5));
```



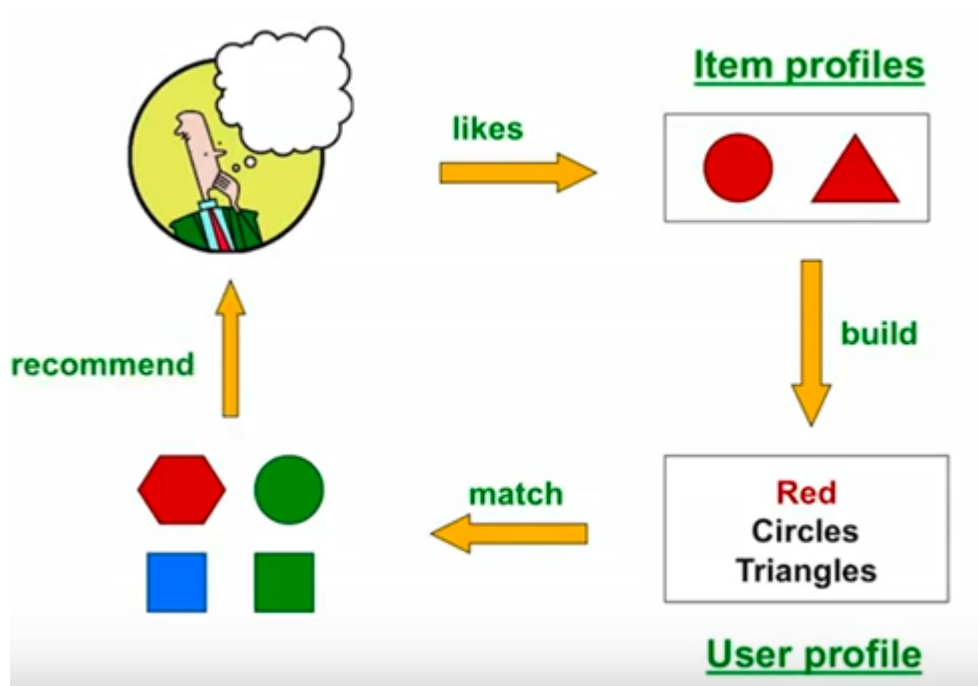
```
In [28]: lowRatedMovies.head(10)
```

Out [28]:

	rating
movielfid	mean
178	1.000000
312	1.444444
449	1.000000
470	1.000000
476	1.000000
698	1.000000
870	1.000000
1163	1.000000
1324	1.166667
1325	1.250000

## Content based filtering

- Content based system suggests you what you like based on what you liked in past.
- Recommendation of items are based on : user's previous purchases, reviews and likes
- Combine: Item description and User profile description
- Generate similarity score
- Recommend the items based on similarity score



The concepts of Term Frequency (TF) and Inverse Document Frequency (IDF) are used in information retrieval systems and also content based filtering mechanisms (such as a content based recommender). They are used to determine the relative importance of a document / article / news item / movie etc.

## Term Frequency (TF) and Inverse Document Frequency (IDF)

TF is simply the frequency of a word in a document. IDF is the inverse of the document frequency among the whole corpus of documents. TF-IDF is used mainly because of two reasons: Suppose we search for "the rise of analytics" on Google. It is certain that "the" will occur more frequently than "analytics" but the relative importance of analytics is higher than the search query point of view. In such cases, TF-IDF weighting negates the effect of high frequency words in determining the importance of an item (document).

We will consider genres as an important parameter to recommend user the movie he watches based on genres of movie user has already watched.

$$TFIDF \text{ score for term } i \text{ in document } j = TF(i, j) * IDF(i)$$

where

*IDF* = Inverse Document Frequency

*TF* = Term Frequency

$$TF(i, j) = \frac{\text{Term } i \text{ frequency in document } j}{\text{Total words in document } j}$$

$$IDF(i) = \log_2 \left( \frac{\text{Total documents}}{\text{documents with term } i} \right)$$

and

*t* = Term

*j* = Document

For calculating distances, many similarity coefficients can be calculated. Most widely used similarity coefficients are Euclidean, Cosine, Pearson Correlation etc.

**Cosine similarity** is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. Given two vectors of attributes, A and B, the cosine similarity,  $\cos(\theta)$ , is represented using a dot product and magnitude as  $\frac{A \cdot B}{\|A\| \|B\|}$ .

We will use cosine distance here. Here we are interested in similarity. That means higher the value more similar they are. But as the function gives us the distance, we will deduct it from 1.

```
In [29]: # Define a TF-IDF Vectorizer Object.
tfidf_movies_genres = TfidfVectorizer(token_pattern = '[a-zA-Z0-9\-\_]+')

#Replace NaN with an empty string
df_movies['genres'] = df_movies['genres'].replace(to_replace="(no genres listed)", value='')

#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_movies_genres_matrix = tfidf_movies_genres.fit_transform(df_movies['genres'])
# print(tfidf_movies_genres.get_feature_names())
# Compute the cosine similarity matrix
```

```
# print(tfidf_movies_genres_matrix.shape)
# print(tfidf_movies_genres_matrix.dtype)
cosine_sim_movies = linear_kernel(tfidf_movies_genres_matrix, tfidf_movies_genres_matrix)
# print(cosine_sim_movies)
```

```
In [30]: def get_recommendations_based_on_genres(movie_title, cosine_sim_movies=cosine_sim_movies):
        """
        Calculates top 2 movies to recommend based on given movie titles genres.
        :param movie_title: title of movie to be taken for base of recommendation
        :param cosine_sim_movies: cosine similarity between movies
        :return: Titles of movies recommended to user
        """
        # Get the index of the movie that matches the title
        idx_movie = df_movies.loc[df_movies['title'].isin([movie_title])]
        idx_movie = idx_movie.index

        # Get the pairwise similarity scores of all movies with that movie
        sim_scores_movies = list(enumerate(cosine_sim_movies[idx_movie][0]))

        # Sort the movies based on the similarity scores
        sim_scores_movies = sorted(sim_scores_movies, key=lambda x: x[1], reverse=True)

        # Get the scores of the 10 most similar movies
        sim_scores_movies = sim_scores_movies[1:11]

        # Get the movie indices
        movie_indices = [i[0] for i in sim_scores_movies]

        # Return the top 10 most similar movies
        return df_movies['title'].iloc[movie_indices]
```

```
In [31]: get_recommendations_based_on_genres("Father of the Bride Part II (1995)")
```

```
Out[31]: 17          Four Rooms (1995)
        18    Ace Ventura: When Nature Calls (1995)
        58          Bio-Dome (1996)
        61          Friday (1995)
        79    Black Sheep (1996)
        90    Mr. Wrong (1996)
        92    Happy Gilmore (1996)
        104    Steal Big, Steal Little (1995)
        108    Flirting With Disaster (1996)
        113    Down Periscope (1996)
        Name: title, dtype: object
```

```
In [32]: def get_recommendation_content_model(userId):
        """
        Calculates top movies to be recommended to user based on movie user has
        :param userId: userid of user
        :return: Titles of movies recommended to user
        """
        recommended_movie_list = []
        movie_list = []
        df_rating_filtered = df_ratings[df_ratings["userId"]== userId]
        for key, row in df_rating_filtered.iterrows():
            movie_list.append((df_movies["title"][row["movieId"]]==df_movies["movieId"]))
        for index, movie in enumerate(movie_list):
            for key, movie_recommended in get_recommendations_based_on_genres(movie_title=movie_list[index]):
                recommended_movie_list.append(movie_recommended)

        # removing already watched movie from recommended list
        for movie_title in recommended_movie_list:
            if movie_title in movie_list:
                recommended_movie_list.remove(movie_title)
```

```
    return set(recommended_movie_list)  
get_recommendation_content_model(1)
```

```

Out[32]: {'*batteries not included (1987)',
'101 Dalmatians (One Hundred and One Dalmatians) (1961)',
'10th Kingdom, The (2000)',
'13 Going on 30 (2004)',
'39 Steps, The (1935)',
'48 Hrs. (1982)',
'5,000 Fingers of Dr. T, The (1953)',
'52 Pick-Up (1986)',
'7th Voyage of Sinbad, The (1958)',
'8MM (1999)',
'A Million Ways to Die in the West (2014)',
'A Wrinkle in Time (2018)',
'Above the Rim (1994)',
'Absolute Power (1997)',
'Ace Ventura: When Nature Calls (1995)',
'Action Jackson (1988)',
'Adventures in Babysitting (1987)',
'Adventures of Baron Munchausen, The (1988)',
'Adventures of Buckaroo Banzai Across the 8th Dimension, The (1984)',
'Adventures of Ichabod and Mr. Toad, The (1949)',
'Adventures of Pinocchio, The (1996)',
'Adventures of Rocky and Bullwinkle, The (2000)',
'Adventures of Sharkboy and Lavagirl 3-D, The (2005)',
'Agent Cody Banks (2003)',
'Air Force One (1997)',
'Aladdin (1992)',
'Alamo, The (1960)',
'Alaska (1996)',
'Alice (1990)',
'Alice in Wonderland (1951)',
'Alien Nation (1988)',
'All Dogs Christmas Carol, An (1998)',
'All Dogs Go to Heaven 2 (1996)',
'All That Jazz (1979)',
'Allan Quatermain and the Lost City of Gold (1987)',
'Allegro non troppo (1977)',
'Almost Heroes (1998)',
'Alvin and the Chipmunks: The Squeakquel (2009)',
'Always (1989)',
'Amateur (1994)',
'Amazing Panda Adventure, The (1995)',
'American Astronaut, The (2001)',
'American Pop (1981)',
'Americanization of Emily, The (1964)',
'Amistad (1997)',
'Amityville 1992: It's About Time (1992)',
'Amityville 3-D (1983)',
'Amityville: A New Generation (1993)',
'Amityville: Dollhouse (1996)',
'Anaconda (1997)',
'Anastasia (1997)',
'Anchors Aweigh (1945)',
'And Now... Ladies and Gentlemen... (2002)',
'And the Ship Sails On (E la nave va) (1983)',
'Angel Eyes (2001)',
'Angels and Insects (1995)',
'Annie (1982)',
'Antonia's Line (Antonia) (1995)',
'Antz (1998)',
'Aristocats, The (1970)',
'Arizona Dream (1993)',
'Armour of God (Long xiong hu di) (1987)',
'Armour of God II: Operation Condor (Operation Condor) (Fei ying gai wak) (1991)',

```

'Army of Darkness (1993)',  
 'Around the World in 80 Days (1956)',  
 'Arrival, The (1996)',  
 'Assassins (1995)',  
 'Asterix & Obelix: Mission Cleopatra (AstÃ©rix & ObÃ©lix: Mission ClÃ©opÃ©tre) (2002)',  
 'Asterix and the Vikings (AstÃ©rix et les Vikings) (2006)',  
 'Atlantis: The Lost Empire (2001)',  
 'Attack the Block (2011)',  
 'Austin Powers: The Spy Who Shagged Me (1999)',  
 'Avengers, The (1998)',  
 'Awfully Big Adventure, An (1995)',  
 'B/W (2015)',  
 'BURN-E (2008)',  
 'Babes in Toyland (1934)',  
 'Babes in Toyland (1961)',  
 'Back to the Future Part II (1989)',  
 'Backbeat (1993)',  
 'Bad Taste (1987)',  
 'Balto (1995)',  
 'Bananas (1971)',  
 'Band Wagon, The (1953)',  
 'Band of Outsiders (Bande Ã  part) (1964)',  
 'Bandits (2001)',  
 'Barb Wire (1996)',  
 'Barry Lyndon (1975)',  
 'Basic Instinct (1992)',  
 'Basketball Diaries, The (1995)',  
 'Batman/Superman Movie, The (1998)',  
 'Batman: Gotham Knight (2008)',  
 'Batman: Mask of the Phantasm (1993)',  
 'Batman: Mystery of the Batwoman (2003)',  
 'Batman: Year One (2011)',  
 'Battle for the Planet of the Apes (1973)',  
 'Beach Blanket Bingo (1965)',  
 'Beat the Devil (1953)',  
 'Beautiful Creatures (2000)',  
 'Beauty and the Beast: The Enchanted Christmas (1997)',  
 'Bed of Roses (1996)',  
 'Before the Rain (Pred dozhdot) (1994)',  
 'Bell, Book and Candle (1958)',  
 'Ben-Hur (1959)',  
 'Beneath the Planet of the Apes (1970)',  
 'Beverly Hills Cop (1984)',  
 'Beverly Hills Cop II (1987)',  
 'Big Bounce, The (2004)',  
 'Big Bully (1996)',  
 'Big Country, The (1958)',  
 'Big Fish (2003)',  
 'Big Sleep, The (1946)',  
 '"Bill & Ted's Excellent Adventure (1989)",  
 'Bio-Dome (1996)',  
 'Black Christmas (1974)',  
 'Black Christmas (2006)',  
 'Black Hole, The (1979)',  
 'Black Knight (2001)',  
 'Black Sea (2015)',  
 'Black Sheep (1996)',  
 'Blackhat (2015)',  
 'Blade (1998)',  
 'Blade II (2002)',  
 'Blade Runner (1982)',  
 'Blade: Trinity (2004)',  
 'Blazing Saddles (1974)',

'Blind Swordsman: Zatoichi, The (ZatÅ'ichi) (2003)',  
'Blink (1994)',  
'Blob, The (1958)',  
'Blood and Wine (Blood & Wine) (1996)',  
'Blue Velvet (1986)',  
'Blue in the Face (1995)',  
'Blueberry (2004)',  
'Blues Brothers 2000 (1998)',  
'Boomerang (1992)',  
'Boot, Das (Boat, The) (1981)',  
'Borrowers, The (1997)',  
'Bourne Identity, The (1988)',  
'Boys from Brazil, The (1978)',  
'Boys of St. Vincent, The (1992)',  
'Boys on the Side (1995)',  
'Brave Little Toaster, The (1987)',  
'Breakdown (1997)',  
'Breaking the Waves (1996)',  
'Bride of Frankenstein, The (Bride of Frankenstein) (1935)',  
'Bridges of Madison County, The (1995)',  
'Bright (2017)',  
'Broken Arrow (1996)',  
'Brother Bear (2003)',  
'Brotherhood of the Wolf (Pacte des loups, Le) (2001)',  
'Brothers Bloom, The (2008)',  
'Bruce Almighty (2003)',  
'Bug's Life, A (1998)",  
'Butch Cassidy and the Sundance Kid (1969)',  
'Can't Stop the Music (1980)",  
'Candyman: Farewell to the Flesh (1995)',  
'Cape Fear (1962)',  
'Captain Blood (1935)',  
'Captain Underpants: The First Epic Movie (2017)',  
'Carabineers, The (Carabiniers, Les) (1963)',  
'Care Bears Movie, The (1985)',  
'Carpool (1996)',  
'Casanova (2005)',  
'Casino Royale (1967)',  
'Casper (1995)',  
'Castle Freak (1995)',  
'Cat Ballou (1965)',  
'Cat Returns, The (Neko no ongaeshi) (2002)',  
'Cat from Outer Space, The (1978)',  
'Catch-22 (1970)',  
'Cats Don't Dance (1997)",  
'Catwoman (2004)',  
'Cellular (2004)',  
'Cemetery Man (Dellamorte Dellamore) (1994)',  
'Chain Reaction (1996)',  
'Changeling, The (1980)',  
'Chicken Little (2005)',  
'Children of the Corn IV: The Gathering (1996)',  
'Chinatown (1974)',  
'Chitty Chitty Bang Bang (1968)',  
'Chronicles of Narnia: The Lion, the Witch and the Wardrobe, The (2005)',  
'Cinderella (1950)',  
'Cinderella (1997)',  
'Cirque du Freak: The Vampire's Assistant (2009)",  
'City Slickers (1991)',  
'City Slickers II: The Legend of Curly's Gold (1994)",  
'City of Angels (1998)',  
'City of Ember (2008)',  
'Claim, The (2000)',  
'Clash of the Titans (1981)',



'Claymation Christmas Celebration, A (1987)',  
'Click (2006)',  
'Cliffhanger (1993)',  
'Cloak & Dagger (1984)',  
'Clueless (1995)',  
'Cocoon (1985)',  
'Cocoon: The Return (1988)',  
'Colonel Chabert, Le (1994)',  
'Company of Wolves, The (1984)',  
'Conan the Barbarian (1982)',  
'Coneheads (1993)',  
'Confessions of a Dangerous Mind (2002)',  
'Conquest of the Planet of the Apes (1972)',  
'Constantine (2005)',  
'Cop Land (1997)',  
'Corruptor, The (1999)',  
'Covenant, The (2006)',  
'Crimson Pirate, The (1952)',  
'Crocodile Dundee II (1988)',  
'Crow, The (1994)',  
'Crow: City of Angels, The (1996)',  
'Cry, the Beloved Country (1995)',  
'Cuckoo, The (Kukushka) (2002)',  
'Cure (1997)',  
'Curious George (2006)',  
'Cutthroat Island (1995)',  
'D.A.R.Y.L. (1985)',  
'Da Sweet Blood of Jesus (2014)',  
'Dangerous Minds (1995)',  
'Dante's Peak (1997)',  
'Darby O'Gill and the Little People (1959)',  
'Dark Crystal, The (1982)',  
'Dark Portals: The Chronicles of Vidocq (Vidocq) (2001)',  
'Date with an Angel (1987)',  
'Davy Crockett, King of the Wild Frontier (1955)',  
'Day Watch (Dnevnoy dozor) (2006)',  
'Day the Earth Stood Still, The (1951)',  
'Dead End (2003)',  
'Dead Man Walking (1995)',  
'Dead Men Don't Wear Plaid (1982)',  
'Dead or Alive: Final (2002)',  
'Death Becomes Her (1992)',  
'Deep Impact (1998)',  
'Defending Your Life (1991)',  
'Demolition Man (1993)',  
'Desperate Measures (1998)',  
'Despicable Me (2010)',  
'Destry Rides Again (1939)',  
'Devil and Max Devlin, The (1981)',  
'Dial M for Murder (1954)',  
'Die Hard (1988)',  
'Die Hard 2 (1990)',  
'Die Hard: With a Vengeance (1995)',  
'Digimon: The Movie (2000)',  
'Dinosaur (2000)',  
'Dinotopia (2002)',  
'Dirty Dozen, The (1967)',  
'Discreet Charm of the Bourgeoisie, The (Charme discret de la bourgeoisie, Le) (1972)',  
'District 13 (Banlieue 13) (2004)',  
'Django Unchained (2012)',  
'Doctor Dolittle (1967)',  
'Doctor Who: Planet of the Dead (2009)',  
'Doctor Who: The Next Doctor (2008)',

'Doctor Who: The Waters of Mars (2009)',  
 'Doctor Zhivago (1965)',  
 'Dogma (1999)',  
 'Dorian Gray (2009)',  
 'Double Jeopardy (1999)',  
 'Double Life of Veronique, The (Double Vie de V ronique, La) (1991)',  
 'Double, The (2011)',  
 'Doug's 1st Movie (1999)',  
 'Down Periscope (1996)',  
 'Down to Earth (2001)',  
 'Dr. Horrible's Sing-Along Blog (2008)',  
 'Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)',  
 'Dragonheart (1996)',  
 'Drop Dead Fred (1991)',  
 'Drop Zone (1994)',  
 'Duck, You Sucker (1971)',  
 'Dungeons & Dragons (2000)',  
 'Earth Girls Are Easy (1988)',  
 'Edge, The (1997)',  
 'Eight Below (2006)',  
 'Eight Crazy Nights (Adam Sandler's Eight Crazy Nights) (2002)',  
 'Ella Enchanted (2004)',  
 'Emerald Forest, The (1985)',  
 'Emerald Green (2016)',  
 'Emperor's New Groove, The (2000)',  
 'English Patient, The (1996)',  
 'Enigma (2001)',  
 'Erik the Viking (1989)',  
 'Escape From Tomorrow (2013)',  
 'Escape from L.A. (1996)',  
 'Escape from New York (1981)',  
 'Escape from the Planet of the Apes (1971)',  
 'Evita (1996)',  
 'Ewok Adventure, The (a.k.a. Caravan of Courage: An Ewok Adventure) (1984)',  
 'Ewoks: The Battle for Endor (1985)',  
 'Executive Decision (1996)',  
 'Explorers (1985)',  
 'Extraordinary Adventures of Ad le Blanc-Sec, The (2010)',  
 'Eyes Wide Shut (1999)',  
 'Eyes Without a Face (Yeux sans visage, Les) (1959)',  
 'Eyes of Laura Mars (1978)',  
 'Faculty, The (1998)',  
 'Fantastic Mr. Fox (2009)',  
 'Far From Home: The Adventures of Yellow Dog (1995)',  
 'Fatal Beauty (1987)',  
 'Faust (1926)',  
 'Fear (1996)',  
 'FearDotCom (a.k.a. Fear.com) (a.k.a. Fear Dot Com) (2002)',  
 'Feast (2005)',  
 'Ferngully: The Last Rainforest (1992)',  
 'Finding Nemo (2003)',  
 'Fire and Ice (2008)',  
 'Fish Called Wanda, A (1988)',  
 'Fisher King, The (1991)',  
 'Fistful of Dollars, A (Per un pugno di dollari) (1964)',  
 'Fitzcarraldo (1982)',  
 'Fled (1996)',  
 'Flight of the Phoenix, The (1965)',  
 'Flipper (1996)',  
 'Flirting With Disaster (1996)',  
 'Fly Away Home (1996)',  
 'Flying Tigers (1942)',

'Fog, The (1980)',  
 'Following (1998)',  
 'Fool's Gold (2008)',  
 'For a Few Dollars More (Per qualche dollaro in pi ) (1965)',  
 'Forbidden Kingdom, The (2008)',  
 'Force 10 from Navarone (1978)',  
 'Forget Paris (1995)',  
 'Formula 51 (2001)',  
 'Four Musketeers, The (1974)',  
 'Four Rooms (1995)',  
 'Frankenstein Must Be Destroyed (1969)',  
 'Frankenstein Unbound (1990)',  
 'Frantic (1988)',  
 'Freaks (1932)',  
 'Freddy vs. Jason (2003)',  
 'Freeway (1996)',  
 'French Twist (Gazon maudit) (1995)',  
 'Fresh (1994)',  
 'Friday (1995)',  
 'Friday the 13th (1980)',  
 'Friend Is a Treasure, A (Chi Trova Un Amico, Trova un Tesoro) (Who Finds  
 a Friend Finds a Treasure) (1981)',  
 'Frisco Kid, The (1979)',  
 'From Dusk Till Dawn 2: Texas Blood Money (1999) ',  
 'From Here to Eternity (1953)',  
 'Frosty the Snowman (1969)',  
 'Funny Face (1957)',  
 'G-Force (2009)',  
 'G.I. Joe: The Movie (1987)',  
 'Galaxy Quest (1999)',  
 'Gamers, The: Dorkness Rising (2008)',  
 'Gattaca (1997)',  
 'General's Daughter, The (1999)',  
 'General, The (1926)',  
 'Georgia (1995)',  
 'Get Shorty (1995)',  
 'Ghost in the Shell (2017)',  
 'Ghostbusters (a.k.a. Ghost Busters) (1984)',  
 'Gigli (2003)',  
 'Ginger Snaps Back: The Beginning (2004)',  
 'Girl Walks Into a Bar (2011)',  
 'Girl Who Played with Fire, The (Flickan som lekte med elden) (2009)',  
 'Glimmer Man, The (1996)',  
 'Godfather: Part III, The (1990)',  
 'Gods Must Be Crazy, The (1980)',  
 'Godzilla (1998)',  
 'Godzilla (Gojira) (1954)',  
 'Gone in 60 Seconds (2000)',  
 'Gone with the Wind (1939)',  
 'Goodbye Lover (1999)',  
 'Grand Day Out with Wallace and Gromit, A (1989)',  
 'Great Dictator, The (1940)',  
 'Great Race, The (1965)',  
 'Great Yokai War, The (Y  kai daisens  ) (2005)',  
 'Great Ziegfeld, The (1936)',  
 'Guilty of Romance (Koi no tsumi) (2011) ',  
 'Hancock (2008)',  
 'Hanna (2011)',  
 'Happiness of the Katakuris, The (Katakuri-ke no k  fuku) (2001)',  
 'Happy Gilmore (1996)',  
 'Hard Rain (1998)',  
 'Harlem Nights (1989)',  
 'Harry Potter and the Chamber of Secrets (2002)',  
 'Harry Potter and the Sorcerer's Stone (a.k.a. Harry Potter and the Philos

opher's Stone) (2001)",  
 'Harvey (1950)',  
 'Hate (Haine, La) (1995)',  
 'He Loves Me... He Loves Me Not (À\x80 la folie... pas du tout) (2002)',  
 'Heart and Souls (1993)',  
 'Heartbreakers (2001)',  
 'Heavenly Creatures (1994)',  
 'Helen of Troy (2003)',  
 'Hell Comes to Frogtown (1988)',  
 'Hellboy (2004)',  
 'Help! (1965)',  
 'Henry: Portrait of a Serial Killer (1986)',  
 'Hercules (1997)',  
 'Hey Arnold! The Movie (2002)',  
 'Hideaway (1995)',  
 'High Heels and Low Lifes (2001)',  
 'High Rise (2015)',  
 'History of Future Folk, The (2012)',  
 'History of the World: Part I (1981)',  
 'Hit by Lightning (2014)',  
 'Hitch Hikers Guide to the Galaxy, The (1981)',  
 'Hitcher, The (2007)',  
 'Holiday Inn (1942)',  
 'Home (2015)',  
 'Home for the Holidays (1995)',  
 'Homeward Bound II: Lost in San Francisco (1996)',  
 'Honey, I Blew Up the Kid (1992)',  
 'Hook (1991)',  
 'Horse Soldiers, The (1959)',  
 'Host, The (2013)',  
 'Hostel: Part II (2007)',  
 'Hot Shots! (1991)',  
 'House of Games (1987)',  
 'How the Grinch Stole Christmas! (1966)',  
 'How to Make an American Quilt (1995)',  
 'How to Steal a Million (1966)',  
 'I Served the King of England (Obsluhoval jsem anglickÃ©ho krÃ¡le) (2006)',  
 'Ice Age (2002)',  
 'Ice Age: A Mammoth Christmas (2011)',  
 'If Lucy Fell (1996)',  
 'Imagine That (2009)',  
 'Impostor (2002)',  
 'In Bruges (2008)',  
 'In Like Flint (1967)',  
 'In Time (2011)',  
 'In the Army Now (1994)',  
 'In the Bleak Midwinter (1995)',  
 'In the Heat of the Night (1967)',  
 'In the Line of Fire (1993)',  
 'In the blue sea, in the white foam. (1984)',  
 'Indian in the Cupboard, The (1995)',  
 'Informant!, The (2009)',  
 'Inkheart (2008)',  
 'Insomnia (1997)',  
 'Insomnia (2002)',  
 'Into the Woods (1991)',  
 'Into the Woods (2014)',  
 'Iron Sky (2012)',  
 'JFK (1991)',  
 'Jabberwocky (1977)',  
 'Jack Reacher: Never Go Back (2016)',  
 'Jane Austen's Mafia! (1998)",  
 "Jason's Lyric (1994)",

'Jean de Florette (1986)',  
'Jeffrey (1995)',  
'Jennifer 8 (1992)',  
'Jesus Christ Vampire Hunter (2001)',  
'Jewel of the Nile, The (1985)',  
'Jimmy Neutron: Boy Genius (2001)',  
'Joe's Apartment (1996)',  
'Johnny Mnemonic (1995)',  
'Jonah: A VeggieTales Movie (2002)',  
'Journey to the Center of the Earth (1959)',  
'Judge Dredd (1995)',  
'Judgment Night (1993)',  
'Juliet of the Spirits (Giulietta degli spiriti) (1965)',  
'Jungle Book, The (1967)',  
'Just Cause (1995)',  
'Karate Kid, Part II, The (1986)',  
'Kaspar Hauser (1993)',  
'Kid, The (2000)',  
'Killer Movie (2008)',  
'Killer, The (Die xue shuang xiong) (1989)',  
'Killing Zoe (1994)',  
'King Kong vs. Godzilla (Kingukongu tai Gojira) (1962)',  
'King Solomon's Mines (1937)',  
'King Solomon's Mines (1950)',  
'King and I, The (1999)',  
'King of Hearts (1966)',  
'Kingdom of Heaven (2005)',  
'Kirikou and the Sorceress (Kirikou et la sorcière) (1998)',  
'Kiss of Death (1995)',  
'Kite (2014)',  
'Klute (1971)',  
'Knockin' on Heaven's Door (1997)',  
'Kull the Conqueror (1997)',  
'Kung Fu Panda: Secrets of the Furious Five (2008)',  
'L.A. Slasher (2015)',  
'La Cérémonie (1995)',  
'Lady Vanishes, The (1938)',  
'Lady in White (a.k.a. The Mystery of the Lady in White) (1988)',  
'Land Before Time III: The Time of the Great Giving (1995)',  
'Land Before Time, The (1988)',  
'Land and Freedom (Tierra y libertad) (1995)',  
'Land of the Dead (2005)',  
'Laputa: Castle in the Sky (Tenkū no shiro Rapyuta) (1986)',  
'Lara Croft Tomb Raider: The Cradle of Life (2003)',  
'Lassie (1994)',  
'Last Action Hero (1993)',  
'Last House on the Left, The (1972)',  
'Last Man Standing (1996)',  
'Last Man on Earth, The (Ultimo uomo della Terra, L') (1964)',  
'Last Mimzy, The (2007)',  
'Last Unicorn, The (1982)',  
'Last of the Dogmen (1995)',  
'Laura (1944)',  
'Leaves of Grass (2009)',  
'Leaving Las Vegas (1995)',  
'Legal Eagles (1986)',  
'Legend of Sleepy Hollow, The (1949)',  
'Legends of the Fall (1994)',  
'Lemony Snicket's A Series of Unfortunate Events (2004)',  
'Let's Get Harry (1986)',  
'Lethal Weapon (1987)',  
'Lethal Weapon 2 (1989)',  
'Lethal Weapon 3 (1992)',  
'Lethal Weapon 4 (1998)',

'Librarian: Quest for the Spear, The (2004)',  
 'Life Aquatic with Steve Zissou, The (2004)',  
 'Life Eternal (2015)',  
 'Life Is Beautiful (La Vita È bella) (1997)',  
 'Life Less Ordinary, A (1997)',  
 'Lifeforce (1985)',  
 'Lilo & Stitch (2002)',  
 'Lion King 1½, The (2004)',  
 'Lion King II: Simba's Pride, The (1998)',  
 'Little Drummer Boy, The (1968)',  
 'Little Shop of Horrors (1986)',  
 'Lonesome Dove (1989)',  
 'Looper (2012)',  
 'Lord of the Rings: The Fellowship of the Ring, The (2001)',  
 'Lord of the Rings: The Two Towers, The (2002)',  
 'Losers, The (2010)',  
 'Lost World: Jurassic Park, The (1997)',  
 'Lost in Space (1998)',  
 'Love & Human Remains (1993)',  
 'Luna Papa (1999)',  
 'Léon: The Professional (a.k.a. The Professional) (Léon) (1994)',  
 'M\*A\*S\*H (a.k.a. MASH) (1970)',  
 'Machine Girl, The (Kataude mashin gāru) (2008)',  
 'Mad Love (1995)',  
 'Madagascar (2005)',  
 'Magnificent Seven, The (1960)',  
 'Major Dundee (1965)',  
 'Malice (1993)',  
 'Mallrats (1995)',  
 'Man Bites Dog (C'est arrivé près de chez vous) (1992)',  
 'Man Who Knew Too Little, The (1997)',  
 'Man Who Would Be King, The (1975)',  
 'Man in the Iron Mask, The (1998)',  
 'Man on Fire (2004)',  
 'Many Adventures of Winnie the Pooh, The (1977)',  
 'Mars Attacks! (1996)',  
 'Maverick (1994)',  
 'Maximum Risk (1996)',  
 'Me, Myself & Irene (2000)',  
 'Medallion, The (2003)',  
 'Meet the Feebles (1989)',  
 'Meet the Robinsons (2007)',  
 'Men in Black II (a.k.a. MIIB) (a.k.a. MIB 2) (2002)',  
 'Merlin (1998)',  
 'Midnight Run (1988)',  
 'Midsummer Night's Dream, A (1935)',  
 'Milagro Beanfield War (1988)',  
 'Miss Congeniality 2: Armed and Fabulous (2005)',  
 'Mister Roberts (1955)',  
 'Misérables, Les (1995)',  
 'Mod Squad, The (1999)',  
 'Monday (2000)',  
 'Monster House (2006)',  
 'Monsters, Inc. (2001)',  
 'Moonraker (1979)',  
 'Mortal Kombat (1995)',  
 'Mortal Kombat: Annihilation (1997)',  
 'Mortal Thoughts (1991)',  
 'Mouse That Roared, The (1959)',  
 'Mr. & Mrs. Smith (2005)',  
 'Mr. Holland's Opus (1995)',  
 'Mr. Wrong (1996)',  
 'Mulholland Drive (2001)',  
 'Mulholland Falls (1996)',

'Muppet Movie, The (1979)',  
'Muppet Treasure Island (1996)',  
'Muppets Most Wanted (2014)',  
'Murder at 1600 (1997)',  
'Murder on the Orient Express (1974)',  
'Musketeer, The (2001)',  
'Mutiny on the Bounty (1935)',  
'My Favorite Martian (1999)',  
'My Neighbor Totoro (Tonari no Totoro) (1988)',  
'Mystery Men (1999)',  
'Mystery Science Theater 3000: The Movie (1996)',  
'Name of the Rose, The (Name der Rose, Der) (1986)',  
'Nashville (1975)',  
'National Treasure (2004)',  
'Natural Born Killers (1994)',  
'Net, The (1995)',  
'NeverEnding Story II: The Next Chapter, The (1990)',  
'NeverEnding Story III, The (1994)',  
'NeverEnding Story, The (1984)',  
'New Adventures of Pippi Longstocking, The (1988)',  
'New Jersey Drive (1995)',  
'Newsies (1992)',  
'Night of the Comet (1984)',  
'Nightbreed (1990)',  
'Nightmare Before Christmas, The (1993)',  
'Nim's Island (2008)',  
'Nine Lives of Tomas Katz, The (2000)',  
'Nine Months (1995)',  
'No Such Thing (2001)',  
'Nobody Loves Me (Keiner liebt mich) (1994)',  
'North Pole: Open For Christmas (2015)',  
'North by Northwest (1959)',  
'Nothing But Trouble (1991)',  
'Nothing Personal (1995)',  
'Nutty Professor, The (1963)',  
'O Brother, Where Art Thou? (2000)',  
'Odd Life of Timothy Green, The (2012)',  
'Old Man and the Sea, The (1958)',  
'Oliver & Company (1988)',  
'Oliver! (1968)',  
'Omega Man, The (1971)',  
'Omen, The (1976)',  
'On Her Majesty's Secret Service (1969)',  
'Once Upon a Time in the West (C'era una volta il West) (1968)',  
'Once Upon a Time... When We Were Colored (1995)',  
'Once Were Warriors (1994)',  
'Once a Thief (Zong heng si hai) (1991)',  
'One Man Band (2005)',  
'One Tough Cop (1998)',  
'Operation 'Y' & Other Shurik's Adventures (1965)',  
'Operation Petticoat (1959)',  
'Orlando (1992)',  
'Othello (1995)',  
'Our Man Flint (1965)',  
'Outbreak (1995)',  
'P.S. (2004)',  
'Pacific Heights (1990)',  
'Parasite (1982)',  
'Partly Cloudy (2009)',  
'Party Monster (2003)',  
'Paths of Glory (1957)',  
'Patriot Games (1992)',  
'Pearl Harbor (2001)',  
'Pee-wee's Big Adventure (1985)',

'Penn & Teller Get Killed (1989)',  
 'Percy Jackson & the Olympians: The Lightning Thief (2010)',  
 'Perfect Crime, The (Crimen Ferpecto) (Ferpect Crime) (2004)',  
 'Perfect World, A (1993)',  
 'Persuasion (1995)',  
 'Peter Pan (1953)',  
 'Peter Pan (1960)',  
 'Peter Pan (2003)',  
 'Phantom Tollbooth, The (1970)',  
 'Phantom, The (1996)',  
 'Phineas and Ferb the Movie: Across the 2nd Dimension (2011)',  
 'Picnic at Hanging Rock (1975)',  
 'Picture of Dorian Gray, The (1945)',  
 'Pie in the Sky (1996)',  
 'Pink Panther, The (2006)',  
 '"Pirates of the Caribbean: At World's End (2007)"',  
 'Pirates of the Caribbean: The Curse of the Black Pearl (2003)',  
 'Pixel Perfect (2004)',  
 'Plan 9 from Outer Space (1959)',  
 'Planet 51 (2009)',  
 'Playing God (1997)',  
 'Pleasantville (1998)',  
 'Pledge, The (2001)',  
 'Plunkett & MacLeane (1999)',  
 'Pocahontas II: Journey to a New World (1998) ',  
 'Pokemon 4 Ever (a.k.a. PokÃ©mon 4: The Movie) (2002)',  
 'PokÃ©mon the Movie 2000 (2000)',  
 'PokÃ©mon: The First Movie (1998)',  
 'Ponyo (Gake no ue no Ponyo) (2008)',  
 'Poseidon Adventure, The (1972)',  
 'Powerpuff Girls, The (2002)',  
 'Presto (2008)',  
 'Price of Milk, The (2000)',  
 'Prince of Egypt, The (1998)',  
 'Princess and the Frog, The (2009)',  
 'Psycho II (1983)',  
 'Puppet Masters, The (1994)',  
 'Pure Formality, A (Pura formalitÃ\x00, Una) (1994)',  
 'Purple Rose of Cairo, The (1985)',  
 '"Pusher III: I'm the Angel of Death (2005)"',  
 'Queen of the Damned (2002)',  
 'Quest for Camelot (1998)',  
 'Quest for Fire (Guerre du feu, La) (1981)',  
 'Quigley Down Under (1990)',  
 'R.I.P.D. (2013)',  
 'Race to Witch Mountain (2009)',  
 'Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)',  
 'Rapture, The (1991)',  
 'Ratatouille (2007)',  
 'Ratchet & Clank (2016)',  
 'Re-Animator (1985)',  
 'Rear Window (1954)',  
 'Red Balloon, The (Ballon rouge, Le) (1956)',  
 'Red Dragon (2002)',  
 'Red Rock West (1992)',  
 'Red Sonja (1985)',  
 'Red Violin, The (Violon rouge, Le) (1998)',  
 'Replacement Killers, The (1998)',  
 'Repo Man (1984)',  
 'Rescuers Down Under, The (1990)',  
 'Restoration (1995)',  
 'Return from Witch Mountain (1978)',  
 'Return of Jafar, The (1994)',



'Return of the Musketeers, The (1989)',  
 'Return of the Pink Panther, The (1975)',  
 'Return to Never Land (2002)',  
 'Return to Snowy River (a.k.a. The Man From Snowy River II) (1988)',  
 'Ride the High Country (1962)',  
 'Ring, The (2002)',  
 'Rio Grande (1950)',  
 'River Wild, The (1994)',  
 'Road to El Dorado, The (2000)',  
 'Robin Hood: Prince of Thieves (1991)',  
 'RoboCop 2 (1990)',  
 'RoboCop 3 (1993)',  
 'RoboGeisha (Robo-geisha) (2009)',  
 'Rock-A-Doodle (1991)',  
 'Rollerball (1975)',  
 'Rollo and the Woods Sprite (R  lli ja mets  nhenki) (2001)',  
 'Ruby Red (2013)',  
 'Rudolph, the Red-Nosed Reindeer (1964)',  
 'Rules of Attraction, The (2002)',  
 'Run Lola Run (Lola rennt) (1998)',  
 'Rundown, The (2003)',  
 'Russians Are Coming, the Russians Are Coming, The (1966)',  
 'Saboteur (1942)',  
 'Sabrina (1995)',  
 'Safe (1995)',  
 'Sands of Iwo Jima (1949)',  
 'Sanjuro (Tsubaki Sanj  r  ) (1962)',  
 'Santa Claus: The Movie (1985)',  
 'Sapphire Blue (2014)',  
 'Saving Private Ryan (1998)',  
 'Saw (2003)',  
 'Saw III (2006)',  
 'Saw IV (2007)',  
 'Scalphunters, The (1968)',  
 'Scooby-Doo (2002)',  
 'Scream 2 (1997)',  
 'Scream 4 (2011)',  
 'Screamers (1995)',  
 'Scrooge (1970)',  
 'Scrooged (1988)',  
 'Secret Life of Walter Mitty, The (1947)',  
 'Secret World of Arrietty, The (Kari-gurashi no Arietti) (2010)',  
 'Selena (1997)',  
 'Serial Mom (1994)',  
 'Set It Off (1996)',  
 'Seven Samurai (Shichinin no samurai) (1954)',  
 'Shadow Conspiracy (1997)',  
 'Shadow, The (1994)',  
 'Shallow Hal (2001)',  
 'Shanghai Knights (2003)',  
 'Shanghai Surprise (1986)',  
 'Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)',  
 'Shawshank Redemption, The (1994)',  
 'Shenandoah (1965)',  
 'Sherlock Holmes and the Secret Weapon (1942)',  
 'Sherlock Holmes: Terror by Night (1946)',  
 'Sherlock Jr. (1924)',  
 'Sherlock: The Abominable Bride (2016)',  
 'Shrek 2 (2004)',  
 'Shrek the Third (2007)',  
 'Silverado (1985)',  
 'Simple Wish, A (1997)',  
 'Sin City (2005)',  
 'Sinbad and the Eye of the Tiger (1977)',

'Sinbad: Legend of the Seven Seas (2003)',  
 'Six-String Samurai (1998)',  
 'Sleeper (1973)',  
 'Sleepers (1996)',  
 'Sliver (1993)',  
 'Smoke (1995)',  
 'Snake Eyes (1998)',  
 'Snakes on a Plane (2006)',  
 'Snatch (2000)',  
 'Snow White and the Seven Dwarfs (1937)',  
 'Snowman, The (1982)',  
 'Snowpiercer (2013)',  
 'Solo (1996)',  
 'Solo: A Star Wars Story (2018)',  
 'Some Like It Hot (1959)',  
 'Something Wicked This Way Comes (1983)',  
 'Son of the Mask (2005)',  
 'Sonatine (Sonachine) (1993)',  
 'Song of the Sea (2014)',  
 'Song of the South (1946)',  
 'Sorcerer's Apprentice, The (2010)",  
 'Sorrow (2015)',  
 'South Pacific (1958)',  
 'Space Buddies (2009)',  
 'Spaced Invaders (1990)',  
 'Spanish Prisoner, The (1997)',  
 'Spartacus (1960)',  
 'Spawn (1997)',  
 'Species (1995)',  
 'Species II (1998)',  
 'Splash (1984)',  
 'SpongeBob SquarePants Movie, The (2004)',  
 'Stagecoach (1939)',  
 'Stalingrad (1993)',  
 'Stand by Me (1986)',  
 'Star Trek II: The Wrath of Khan (1982)',  
 'Star Trek III: The Search for Spock (1984)',  
 'Star Trek IV: The Voyage Home (1986)',  
 'Star Trek V: The Final Frontier (1989)',  
 'Star Trek: First Contact (1996)',  
 'Star Trek: Nemesis (2002)',  
 'Star Wars: Episode IV – A New Hope (1977)',  
 'Star Wars: Episode VI – Return of the Jedi (1983)',  
 'Stardust (2007)',  
 'Stargate (1994)',  
 'Steal Big, Steal Little (1995)',  
 'Steamboat Willie (1928)',  
 'Stendhal Syndrome, The (Sindrome di Stendhal, La) (1996)',  
 'Sting, The (1973)',  
 'Stir of Echoes (1999)',  
 'Straight Story, The (1999)',  
 'Strange Magic (2015)',  
 'Street Fighter (1994)',  
 'Striking Distance (1993)',  
 'Stripes (1981)',  
 'Striptease (1996)',  
 'Stunt Man, The (1980)',  
 'Suicide Squad (2016)',  
 'Sukiyaki Western Django (2008)',  
 'Super Mario Bros. (1993)',  
 'Supercop (Police Story 3: Supercop) (Jing cha gu shi III: Chao ji jing ch  
 a) (1992)',  
 'Supercop 2 (Project S) (Chao ji ji hua) (1993)',  
 'Superhero Movie (2008)',

'Surviving the Game (1994)',  
'Swan Princess, The (1994)',  
'Switchback (1997)',  
'Sword in the Stone, The (1963)',  
'Take the Money and Run (1969)',  
'Taking of Pelham One Two Three, The (1974)',  
'Tale of Despereaux, The (2008)',  
'Talk of the Town, The (1942)',  
'Tangled Ever After (2012)',  
'Tango & Cash (1989)',  
'Tango (1998)',  
'Tarzan (1999)',  
'Tarzan and the Lost City (1998)',  
'Tea with Mussolini (1999)',  
'Terminal Velocity (1994)',  
'Terminator 2: Judgment Day (1991)',  
'Texas Rangers (2001)',  
'The Alamo (2004)',  
'The BFG (2016)',  
'The Cobbler (2015)',  
'The Devil's Advocate (1997)',  
'The Magnificent Seven (2016)',  
'The Mummy (2017)',  
'The Professional: Golgo 13 (1983)',  
'The Spiral Staircase (1945)',  
'The Star Wars Holiday Special (1978)',  
'The Voices (2014)',  
'They Call Me Trinity (1971)',  
'Thief of Bagdad, The (1940)',  
'Thin Man, The (1934)',  
'Third Man, The (1949)',  
'Thirty-Two Short Films About Glenn Gould (1993)',  
'Three Caballeros, The (1945)',  
'Three Musketeers, The (1948)',  
'Thumbelina (1994)',  
'Tiger and the Snow, The (La tigre e la neve) (2005)',  
'Tigger Movie, The (2000)',  
'Time Lapse (2014)',  
'Timecop (1994)',  
'Tin Drum, The (Blechtrommel, Die) (1979)',  
'Titan A.E. (2000)',  
'To Be or Not to Be (1942)',  
'Tokyo Tribe (2014)',  
'Tom and Huck (1995)',  
'Too Late for Tears (1949)',  
'Tora! Tora! Tora! (1970)',  
'Touch (1997)',  
'Toy Story 2 (1999)',  
'Toys (1992)',  
'Train of Life (Train de vie) (1998)',  
'Troll 2 (1990)',  
'Tron (1982)',  
'True Crime (1996)',  
'True Grit (1969)',  
'True Lies (1994)',  
'Turbo (2013)',  
'Turbulence (1997)',  
'Twilight Saga: Breaking Dawn – Part 1, The (2011)',  
'Twin Peaks: Fire Walk with Me (1992)',  
'Two Mules for Sister Sara (1970)',  
'Two if by Sea (1996)',  
'U.S. Marshals (1998)',  
'Underneath (1995)',  
'Underworld (2003)',

```
'Underworld: Evolution (2006)',
'Unstrung Heroes (1995)',
'Up (2009)',
'Up Close and Personal (1996)',
'Valiant (2005)',
'Van Helsing (2004)',
'Victor Frankenstein (2015)',
'Village of the Damned (1995)',
'Virtuosity (1995)',
'Virus (1999)',
'WALL•E (2008)',
'Wagons East (1994)',
'Walkabout (1971)',
'Walker (1987)',
'Walking Dead, The (1995)',
'Wallace & Gromit in The Curse of the Were-Rabbit (2005)',
'Wallace & Gromit: The Wrong Trousers (1993)',
'War of the Worlds, The (1953)',
'Watch Out for the Automobile (Beregis avtomobilya) (1966)',
'Way of the Dragon, The (a.k.a. Return of the Dragon) (Meng long guo jian
g) (1972)',
'We're Back! A Dinosaur's Story (1993)",
'Were the World Mine (2008)',
'What Dreams May Come (1998)',
'What Happened Was... (1994)',
'What Planet Are You From? (2000)',
'What's Love Got to Do with It? (1993)",
'When Night Is Falling (1995)',
'Wild Bunch, The (1969)',
'Wild Things (1998)',
'Wild Wild West (1999)',
'Wild, The (2006)',
'William Shakespeare's A Midsummer Night's Dream (1999)",
'Willow (1988)',
'Wing Commander (1999)',
'Wings of Desire (Himmel Äber Berlin, Der) (1987)',
'Witches, The (1990)',
'Wiz, The (1978)',
'Wizards of Waverly Place: The Movie (2009)',
'Wolf Creek (2005)',
'World's End, The (2013)",
'Yellow Submarine (1968)',
'Yojimbo (1961)',
'Young Guns II (1990)',
'Young Poisoner's Handbook, The (1995)",
'Your Highness (2011)',
'Zathura (2005)',
'ÄiThree Amigos! (1986)'}

```

## Model evaluation with KNN

Here the model is evaluated on based of if there is exact match of genres with the genres of movie which is already watch by user

```
In [33]: from sklearn.neighbors import KNeighborsClassifier
def get_movie_label(movie_id):
    """
    Get the cluster label to which movie belongs by KNN algorithm.
    :param movie_id: movie id
    :return: genres label to movie belong
    """

```

```

classifier = KNeighborsClassifier(n_neighbors=5)
x= tfidf_movies_genres_matrix
y = df_movies.iloc[:, -1]
classifier.fit(x, y)
y_pred = classifier.predict(tfidf_movies_genres_matrix[movie_id])
return y_pred

```

```

In [34]: true_count = 0
false_count = 0
def evaluate_content_based_model():
    """
    Evaluate content based model.
    """
    for key, columns in df_movies.iterrows():
        movies_recommended_by_model = get_recommendations_based_on_genres(columns)
        predicted_genres = get_movie_label(movies_recommended_by_model.index)
        for predicted_genre in predicted_genres:
            global true_count, false_count
            if predicted_genre == columns["genres"]:
                true_count = true_count+1
            else:
                # print(columns["genres"])
                # print(predicted_genre)
                false_count = false_count +1
evaluate_content_based_model()
total = true_count + false_count
print("Hit:" + str(true_count/total))
print("Fault:" + str(false_count/total))

```

```

Hit:0.8776739889139807
Fault:0.1223260110860193

```

```

In [35]: from sklearn.metrics import pairwise_distances
from scipy.spatial.distance import cosine, correlation

```

```

In [36]: df_movies = movies
df_ratings = ratings

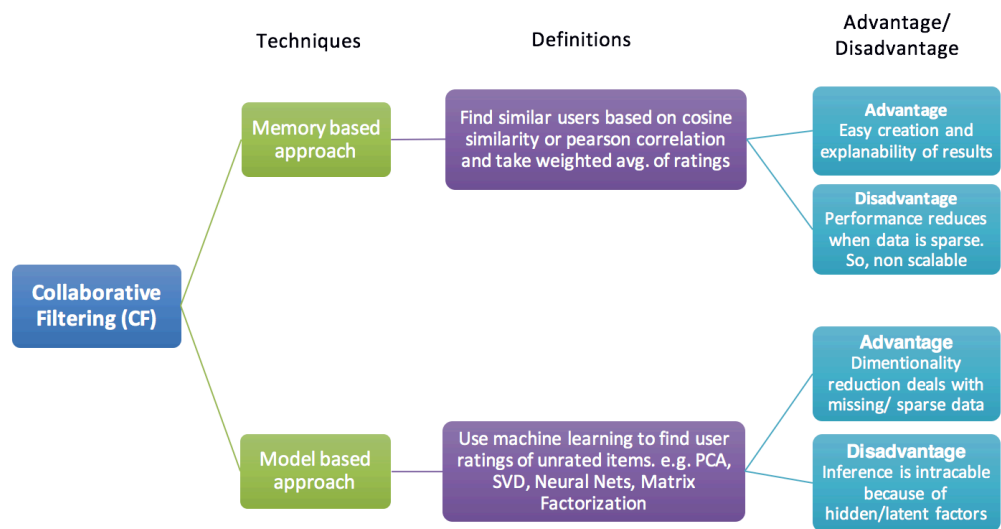
```

## Collaborative Filtering

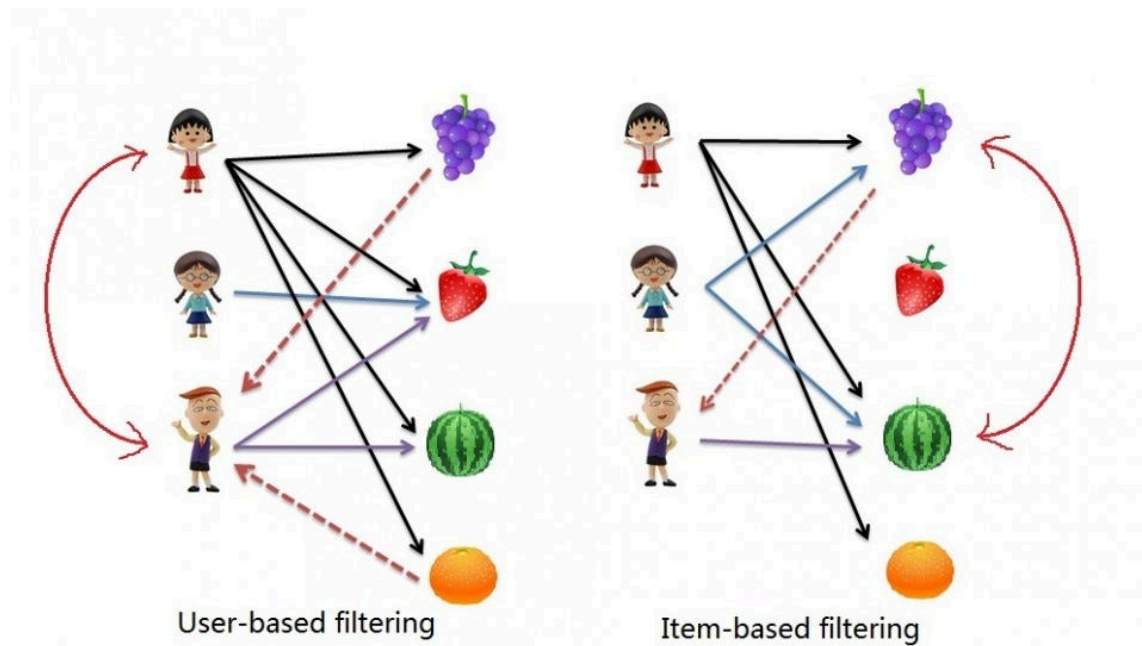
Types of collaborative filtering techniques

- Memory based
  - User-Item Filtering
  - Item-Item Filtering
- Model based
  - Matrix Factorization
  - Clustering

## ■ Deep Learning



## Memory Based Approach



In either scenario, we build a similarity matrix. For user-user collaborative filtering, the user-similarity matrix will consist of some distance metrics that measure the similarity between any two pairs of users. Likewise, the item-similarity matrix will measure the similarity between any two pairs of items.

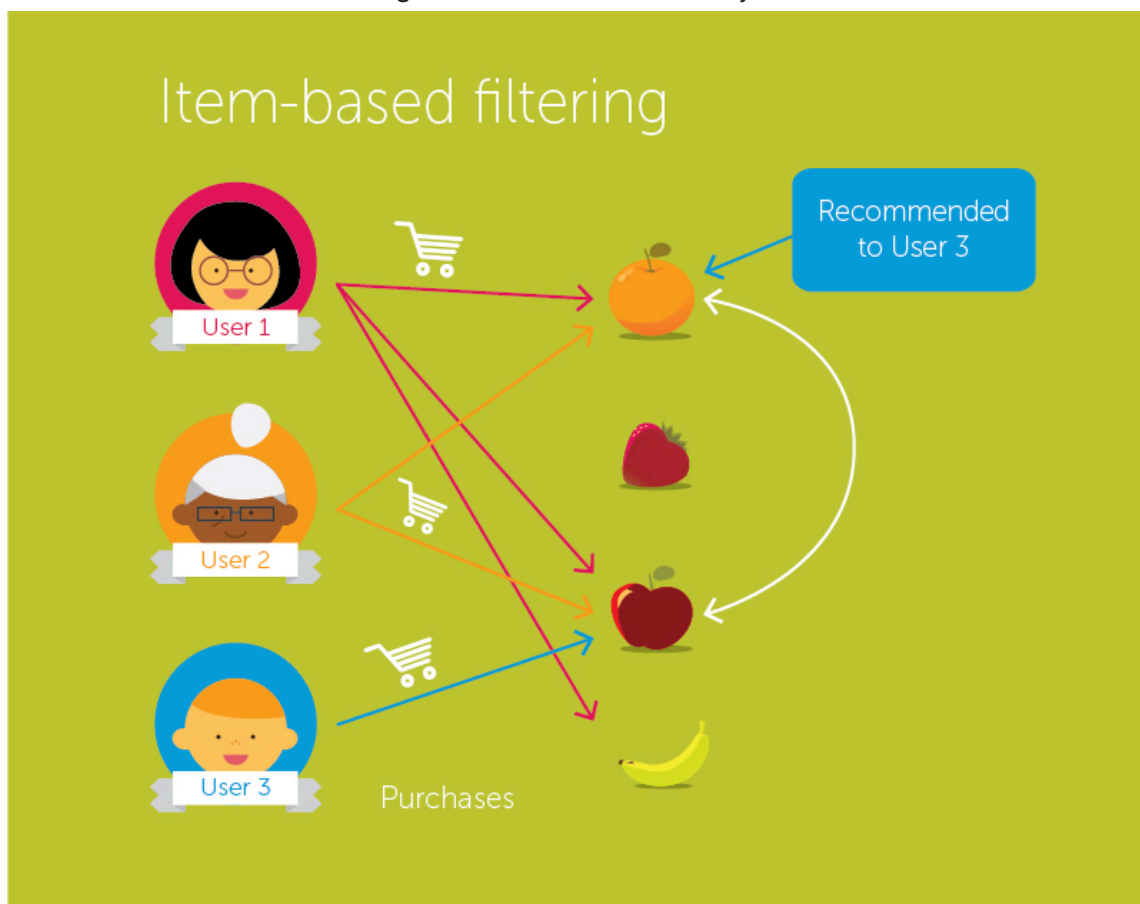
There are 3 distance similarity metrics that are usually used in collaborative filtering:

- **Jaccard Similarity**
- **Cosine Similarity**
- **Pearson Similarity**

## Item-Item Filtering

Item-item collaborative filtering, or item-based, or item-to-item, is a form of collaborative filtering for recommender systems based on the similarity between items calculated using people's ratings of those items.

Item-item collaborative filtering was invented and used by Amazon.com



ITEM-ITEM collaborative filtering look for items that are similar to the articles that user has already rated and recommend most similar articles. But what does that mean when we say item-item similarity? In this case we don't mean whether two items are the same by attribute like Fountain pen and pilot pen are similar because both are pen. Instead, what similarity means is how people treat two items the same in terms of like and dislike.

It is quite similar to previous algorithm, but instead of finding user's look-alike, we try finding movie's look-alike. Once we have movie's look-alike matrix, we can easily recommend alike movies to user who have rated any movie from the dataset. This algorithm is far less resource consuming than user-user collaborative filtering. Hence, for a new user, the algorithm takes far lesser time than user-user collaborate as we don't need all similarity scores between users. And with fixed number of movies, movie-movie look alike matrix is fixed over time.

## Implementation of Item-Item Filtering

```
In [37]: df_movies_ratings=pd.merge(df_movies, df_ratings)
```

```
In [38]: df_movies_ratings
```

Out[38]:

	movieId	title	genres	userId	rating	times
<b>0</b>	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy	1	4.0	96498
<b>1</b>	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy	5	4.0	84743
<b>2</b>	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy	7	4.5	110663
<b>3</b>	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy	15	2.5	151057
<b>4</b>	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy	17	4.5	130569
...	...	...	...	...	...	...
<b>100831</b>	193581	Black Butler: Book of the Atlantic (2017)	Action Animation Comedy Fantasy	184	4.0	153710
<b>100832</b>	193583	No Game No Life: Zero (2017)	Animation Comedy Fantasy	184	3.5	153710
<b>100833</b>	193585	Flint (2017)	Drama	184	3.5	153710
<b>100834</b>	193587	Bungo Stray Dogs: Dead Apple (2018)	Action Animation	184	3.5	153710
<b>100835</b>	193609	Andrew Dice Clay: Dice Rules (1991)	Comedy	331	4.0	153715

100836 rows × 6 columns

Here Pivot table function is used as we want one to one mapping between movies, user and their rating. So by default pivot\_table command takes average if we have multiple values of one combination.

```
In [39]: ratings_matrix_items = df_movies_ratings.pivot_table(index=['movieId'], columns=['userId'], values='rating', fillna=0, inplace=True)
ratings_matrix_items.shape
```

Out[39]: (9724, 610)



In [40]: ratings\_matrix\_items

Out[40]:

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606
0	4.0	0.0	0.0	0.0	4.0	0.0	4.5	0.0	0.0	0.0	...	4.0	0.0	4.0	3.0	4.0	2.5
1	0.0	0.0	0.0	0.0	0.0	4.0	0.0	4.0	0.0	0.0	...	0.0	4.0	0.0	5.0	3.5	0.0
2	4.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	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	0.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	3.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9719	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	0.0	0.0	0.0
9720	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	0.0	0.0	0.0
9721	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	0.0	0.0	0.0
9722	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	0.0	0.0	0.0
9723	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	0.0	0.0	0.0

9724 rows × 610 columns

In [41]:

```

movie_similarity = 1 - pairwise_distances( ratings_matrix_items.to_numpy(),
np.fill_diagonal( movie_similarity, 0 ) #Filling diagonals with 0s for future use
ratings_matrix_items = pd.DataFrame( movie_similarity )
ratings_matrix_items

```

Out[41]:

	0	1	2	3	4	5	6	7
0	0.000000	0.410562	0.296917	0.035573	0.308762	0.376316	0.277491	0.131629
1	0.410562	0.000000	0.282438	0.106415	0.287795	0.297009	0.228576	0.172498
2	0.296917	0.282438	0.000000	0.092406	0.417802	0.284257	0.402831	0.313434
3	0.035573	0.106415	0.092406	0.000000	0.188376	0.089685	0.275035	0.158022
4	0.308762	0.287795	0.417802	0.188376	0.000000	0.298969	0.474002	0.283523
...	...	...	...	...	...	...	...	...
9719	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
9720	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
9721	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
9722	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
9723	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

9724 rows × 9724 columns

Below function will take the movie name as a input and will find the movies which are similar to this movie. This function first find the index of movie in movies frame and then take the similarity of movie and align in movies dataframe so that we can get the similarity of the movie with all other movies.

```
In [42]: def item_similarity(movieName):
        """
        recommends similar movies
        :param data: name of the movie
        """
        try:
            #user_inp=input('Enter the reference movie title based on which reco
            user_inp=movieName
            inp=df_movies[df_movies['title']==user_inp].index.tolist()
            inp=inp[0]

            df_movies['similarity'] = ratings_matrix_items.iloc[inp]
            df_movies.columns = ['movie_id', 'title', 'release_date', 'similarity']
        except:
            print("Sorry, the movie is not in the database!")
```

Here we provide the user id of the user for which we have to recommend movies. Then we find the movies which are rated 5 or 4.5 by the user for whom we want to recommend movies. We are finding this because as we know that in Item-Item similarity approach we recommended movies to the user based on his previous selection. So to foster our algorithm we are finding movies which are liked by the user most and on bases of that we will recommend movies which are similar to movies highly rated by the user. Then our function has appended the similarity of the movie highly rated by the user to our movies data frame. Now we will sort the frame as per the similarity in descending order so that we can get the movies which are highly similar to movie highly rated by our customer. Now we filter the movies which are most similar as per the similarity so if similarity is greater than 0.45 then we are considering the movies. Now the function goes ahead and see which all movies user has seen and then filter out the movies which he has not seen and then recommended that movies to him.

```
In [43]: def recommendedMoviesAsPerItemSimilarity(user_id):
        """
        Recommending movie which user hasn't watched as per Item Similarity
        :param user_id: user_id to whom movie needs to be recommended
        :return: movieIds to user
        """
        user_movie= df_movies_ratings[(df_movies_ratings.userId==user_id) & df_r
        user_movie=user_movie.iloc[0,0]
        item_similarity(user_movie)
        sorted_movies_as_per_userChoice=df_movies.sort_values( ["similarity"], a
        sorted_movies_as_per_userChoice=sorted_movies_as_per_userChoice[sorted_r
        recommended_movies=list()
        df_recommended_item=pd.DataFrame()
        user2Movies= df_ratings[df_ratings['userId']== user_id]['movieId']
        for movieId in sorted_movies_as_per_userChoice:
            if movieId not in user2Movies:
                df_new= df_ratings[(df_ratings.movieId==movieId)]
                df_recommended_item=pd.concat([df_recommended_item,df_new])
                best10=df_recommended_item.sort_values(["rating"], ascending = l
        return best10['movieId']
```

```
In [44]: def movieIdToTitle(listMovieIDs):
        """
        Converting movieId to titles
        :param user_id: List of movies
        :return: movie titles
        """
        movie_titles= list()
```

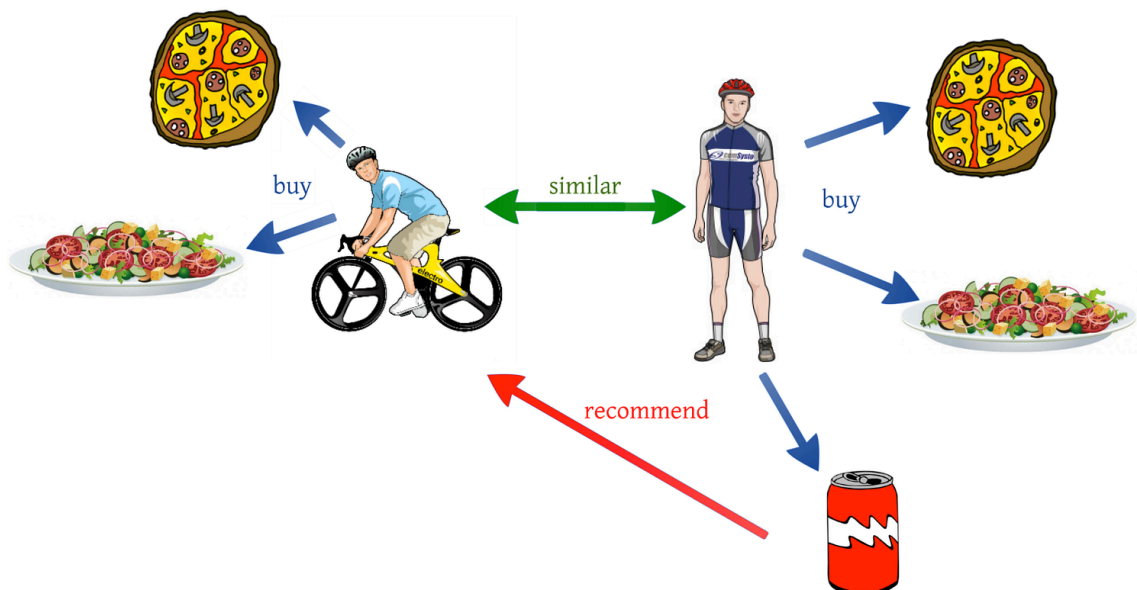
```
for id in listMovieIDs:
    movie_titles.append(df_movies[df_movies['movie_id']==id]['title'])
return movie_titles
```

```
In [45]: user_id=50
print("Recommended movies,:\n",movieIdToTitle(recommendedMoviesAsperItemSim:
```

```
Recommended movies,:
[659    Godfather, The (1972)
Name: title, dtype: object, 922    Godfather: Part II, The (1974)
Name: title, dtype: object, 510    Silence of the Lambs, The (1991)
Name: title, dtype: object, 922    Godfather: Part II, The (1974)
Name: title, dtype: object, 922    Godfather: Part II, The (1974)
Name: title, dtype: object, 659    Godfather, The (1972)
Name: title, dtype: object, 659    Godfather, The (1972)
Name: title, dtype: object, 659    Godfather, The (1972)
Name: title, dtype: object, 1644    Willow (1988)
Name: title, dtype: object]
```

## User-Item Filtering

The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a 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.



Here we find look alike users based on similarity and recommend movies which first user's look-alike has chosen in past. This algorithm is very effective but takes a lot of time and resources. It requires to compute every user pair information which takes time. Therefore, for big base platforms, this algorithm is hard to implement without a very strong parallelizable system.

## Implementation of User-Item Filtering

In similar way as we did for ItemItem similarity we will create a matrix but here we will keep rows as user and columns as movieId as we want a vector of different users. Then in similar ways we will find distance and similarity between users.

```
In [46]: ratings_matrix_users = df_movies_ratings.pivot_table(index=['userId'],columns='movieId',
ratings_matrix_users.fillna( 0, inplace = True )
movie_similarity = 1 - pairwise_distances( ratings_matrix_users.to_numpy(),
np.fill_diagonal( movie_similarity, 0 ) #Filling diagonals with 0s for future use
ratings_matrix_users = pd.DataFrame( movie_similarity )
ratings_matrix_users
```

```
Out[46]:
```

	0	1	2	3	4	5	6	7	
0	0.000000	0.027283	0.059720	0.194395	0.129080	0.128152	0.158744	0.136968	0.0
1	0.027283	0.000000	0.000000	0.003726	0.016614	0.025333	0.027585	0.027257	0.0
2	0.059720	0.000000	0.000000	0.002251	0.005020	0.003936	0.000000	0.004941	0.0
3	0.194395	0.003726	0.002251	0.000000	0.128659	0.088491	0.115120	0.062969	0.0
4	0.129080	0.016614	0.005020	0.128659	0.000000	0.300349	0.108342	0.429075	0.0
...	...	...	...	...	...	...	...	...	...
605	0.164191	0.028429	0.012993	0.200395	0.106435	0.102123	0.200035	0.099388	0.0
606	0.269389	0.012948	0.019247	0.131746	0.152866	0.162182	0.186114	0.185142	0.0
607	0.291097	0.046211	0.021128	0.149858	0.135535	0.178809	0.323541	0.187233	0.0
608	0.093572	0.027565	0.000000	0.032198	0.261232	0.214234	0.090840	0.423993	0.0
609	0.145321	0.102427	0.032119	0.107683	0.060792	0.052668	0.193219	0.078153	0.0

610 rows × 610 columns

Here now we have similarity of users in columns with respective users in row. So if we find maximum value in a column we will get the user with highest similarity. So now we can have a pair of users which are similar.

```
In [47]: ratings_matrix_users.idxmax(axis=1)
```

```
Out[47]:
```

0	265
1	365
2	312
3	390
4	469
...	...
605	473
606	569
607	479
608	339
609	248

Length: 610, dtype: int64

```
In [48]: ratings_matrix_users.idxmax(axis=1).sample( 10, random_state = 10 )
```

```
Out[48]: 547      76
         241     467
         277     337
         348     454
         218     238
         407     278
         352      45
         97      600
         381      20
         607     479
         dtype: int64
```

```
In [49]: similar_user_series= ratings_matrix_users.idxmax(axis=1)
         df_similar_user= similar_user_series.to_frame()
```

```
In [50]: df_similar_user.columns=['similarUser']
```

```
In [51]: df_similar_user
```

```
Out[51]:
```

	similarUser
0	265
1	365
2	312
3	390
4	469
...	...
605	473
606	569
607	479
608	339
609	248

610 rows × 1 columns

Below function takes id of the user to whom we have to recommend movies. On basis of that, we find the user which is similar to that user and then filter the movies which are highly rated by the user to recommend them to given user.

```
In [52]: movieId_recommended=list()
         def getRecommendedMoviesAsperUserSimilarity(userId):
             """
             Recommending movies which user hasn't watched as per User Similarity
             :param user_id: user_id to whom movie needs to be recommended
             :return: movieIds to user
             """
             user2Movies= df_ratings[df_ratings['userId']== userId]['movieId']
             sim_user=df_similar_user.iloc[0,0]
             df_recommended=pd.DataFrame(columns=['movieId','title','genres','userId'])
             for movieId in df_ratings[df_ratings['userId']== sim_user]['movieId']:
                 if movieId not in user2Movies:
                     df_new= df_movies_ratings[(df_movies_ratings.userId==sim_user) &
                     df_recommended=pd.concat([df_recommended,df_new])
```

```
best10=df_recommended.sort_values(['rating'], ascending = False )[1:10]
return best10['movieId']
```

```
In [53]: user_id=50
recommend_movies= movieIdToTitle(getRecommendedMoviesAsperUserSimilarity(user_id))
print("Movies you should watch are:\n")
print(recommend_movies)
```

Movies you should watch are:

```
[1431    Rocky (1976)
Name: title, dtype: object, 742    African Queen, The (1951)
Name: title, dtype: object, 733    It's a Wonderful Life (1946)
Name: title, dtype: object, 939    Terminator, The (1984)
Name: title, dtype: object, 969    Back to the Future (1985)
Name: title, dtype: object, 510    Silence of the Lambs, The (1991)
Name: title, dtype: object, 1057   Star Trek II: The Wrath of Khan (1982)
Name: title, dtype: object, 1059   Star Trek IV: The Voyage Home (1986)
Name: title, dtype: object, 1939   Matrix, The (1999)
Name: title, dtype: object]
```

## Evaluating the model

```
In [54]: def get_user_similar_movies( user1, user2 ):
        """
        Returning common movies and ratings of same for both the users
        :param user1,user2: user ids of 2 users need to compare
        :return: movieIds to user
        """
        common_movies = df_movies_ratings[df_movies_ratings.userId == user1].merge(
            df_movies_ratings[df_movies_ratings.userId == user2],
            on = "movieId",
            how = "inner" )
        common_movies.drop(['movieId','genres_x','genres_y', 'timestamp_x','timestamp_y'])
        return common_movies
```

```
In [55]: get_user_similar_movies(587,511)
```

```
Out[55]:
```

	title_x	userId_x	rating_x	userId_y	rating_y
0	Forrest Gump (1994)	587	4.0	511	4.5
1	Life Is Beautiful (La Vita È bella) (1997)	587	5.0	511	4.5
2	Matrix, The (1999)	587	4.0	511	5.0

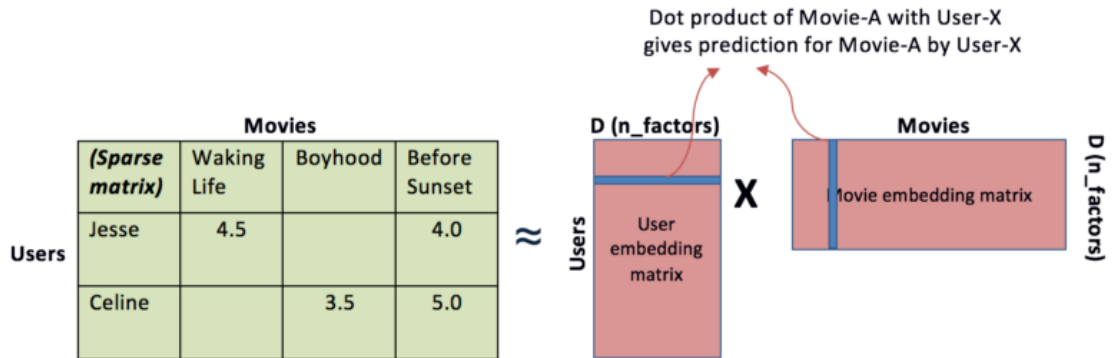
## Pros and Cons of two methods

### Challenges with User similarity

- The challenge with calculating user similarity is the user need to have some prior purchases and should have rated them.
- This recommendation technique does not work for new users.
- The system need to wait until the user make some purchases and rates them. Only then similar users can be found and recommendations can be made. This is called cold start problem.

# Single Value Decomposition

Singular value decomposition is a method of decomposing a matrix into three other matrices:



$$A = USV^T$$

(1)

Where:

$A$  is an  $m \times n$  matrix  
 $U$  is an  $m \times r$  orthogonal matrix  
 $S$  is an  $r \times r$  diagonal matrix  
 $V$  is an  $r \times n$  orthogonal matrix

$$a_{ij} = \sum_{k=1}^n u_{ik} s_k v_{jk}$$

Note how we've collapsed the diagonal matrix,  $S$ , into a vector, thus simplifying the expression into a single summation. The variables,  $\{s_i\}$ , are called singular values and are normally arranged from largest to smallest:

$$s_{i+1} \leq s_i$$

The columns of  $U$  are called left singular vectors, while those of  $V$  are called right singular vectors.

We know that  $U$  and  $V$  are orthogonal, that is:

$$U^T U = V V^T = I$$

Where  $I$  is the identity matrix. Only the diagonals of the identity matrix are 1, with all other values being 0. Note that because  $U$  is not square we cannot say that  $U^T U = I$ , so  $U$  is only orthogonal in one direction.

Using the orthogonality property, we can rearrange (1) into the following pair of eigenvalue equations:

## Data reduction

A typical machine learning problem might have several hundred or more variables, while many machine learning algorithms will break down if presented with more than a few dozen. This makes singular value decomposition indispensable in ML for variable reduction.

We have already seen in Equation (6) how an SVD with a reduced number of singular values can closely approximate a matrix. This can be used for data compression by storing the truncated forms of  $U$ ,  $S$ , and  $V$  in place of  $A$  and for variable reduction by replacing  $A$  with  $U$ .

## Matrix Factorization

Memory-based collaborative filtering approaches that compute distance relationships between items or users have these two major issues:

- It doesn't scale particularly well to massive datasets, especially for real-time recommendations based on user behavior similarities—which takes a lot of computations.
- Ratings matrices may be overfitting to noisy representations of user tastes and preferences. When we use distance based “neighborhood” approaches on raw data, we match to sparse low-level details that we assume represent the user's preference vector instead of the vector itself.

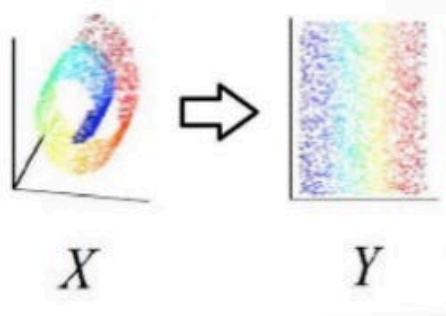


# Reasons to Reduce Dimensionality

Quicker and more accurate results from machine learning methods

1. Computational issues with large numbers of predictors
2. Inability to use certain statistical methods when not enough observations per predictor exist (such as regression)
3. Less accurate results if the data is noisy (model over-fitting)

Easier to visualize data



Easier to data mine with fewer predictors

Thus we need to apply Dimensionality Reduction technique to derive the tastes and preferences from the raw data, otherwise known as doing **low-rank matrix factorization**.

- We can discover hidden correlations / features in the raw data.
- We can remove redundant and noisy features that are not useful.
- We can interpret and visualize the data easier.
- We can also access easier data storage and processing.

## Let's try out the same in Python

```
In [56]: # Import libraries
import numpy as np
import pandas as pd

# Reading ratings file
ratings = pd.read_csv('ratings.csv', sep=',', encoding='latin-1', usecols=[

# Reading users file
#users = pd.read_csv('users.dat', sep='\t', encoding='latin-1', usecols=['us

# Reading movies file
movies = pd.read_csv('movies.csv', sep=',', encoding='latin-1', usecols=['mo
```

```
In [57]: n_users = ratings.userId.unique().shape[0]
n_movies = ratings.movieId.unique().shape[0]
print('Number of users = ' + str(n_users) + ' | Number of movies = ' + str(n_movies))

Number of users = 610 | Number of movies = 9724
```

Now we want the format of our ratings matrix to be one row per user and one column per movie. To do so, we will pivot ratings to get that and call the new variable Ratings (with a capital \*R).

```
In [58]: Ratings = ratings.pivot(index = 'userId', columns = 'movieId', values = 'rating')
Ratings.head()
```

```
Out[58]:
```

	movieId	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193572
	userId															
	1	4.0	0.0	4.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
	2	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	0.0
	3	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	0.0
	4	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	0.0
	5	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0

5 rows x 9724 columns

Last but not least, we need to de-normalize the data (normalize by each users mean) and convert it from a dataframe to a numpy array.

```
In [59]: R = Ratings.to_numpy()
#print(R)
user_ratings_mean = np.mean(R, axis = 1)
#print(user_ratings_mean.shape)
print(user_ratings_mean.size)
Ratings_demeaned = R - user_ratings_mean.reshape(-1, 1) ## Making the user_
```

610

With our ratings matrix properly formatted and normalized, we are ready to do some dimensionality reduction. But first, let's go over the math.

## Model-Based Collaborative Filtering

**Model-based Collaborative Filtering** is based on matrix factorization (MF) which has received greater exposure, mainly as an unsupervised learning method for latent variable decomposition and dimensionality reduction. Matrix factorization is widely used for recommender systems where it can deal better with scalability and sparsity than Memory-based CF:

- The goal of MF is to learn the latent preferences of users and the latent attributes of items from known ratings (learn features that describe the characteristics of ratings) to then predict the unknown ratings through the dot product of the latent features of users and items.
- When you have a very sparse matrix, with a lot of dimensions, by doing matrix factorization, you can restructure the user-item matrix into low-rank structure, and you can represent the matrix by the multiplication of two low-rank matrices, where the rows contain the latent vector.

- You fit this matrix to approximate your original matrix, as closely as possible, by multiplying the low-rank matrices together, which fills in the entries missing in the original matrix.

For example, let's check the sparsity of the ratings dataset:

```
In [60]: sparsity = round(1.0 - len(ratings) / float(n_users * n_movies), 3)
print('The sparsity level of MovieLens100K dataset is ' + str(sparsity * 100))
```

The sparsity level of MovieLens100K dataset is 98.3%

A well-known matrix factorization method is Singular value decomposition (SVD). At a high level, SVD is an algorithm that decomposes a matrix  $A$  into the best lower rank (i.e. smaller/simpler) approximation of the original matrix  $A$ . Mathematically, it decomposes  $A$  into a two unitary matrices and a diagonal matrix:

$$\begin{array}{c}
 \boxed{\mathbf{A}_{m \times n}} = \boxed{\mathbf{U}_{m \times m}} \times \boxed{\Sigma_{m \times n}} \times \boxed{\mathbf{V}_{n \times n}^T} \\
 (m < n)
 \end{array}$$
  

$$\begin{array}{c}
 \boxed{\mathbf{A}_{m \times n}} = \boxed{\mathbf{U}_{m \times m}} \times \boxed{\Sigma_{m \times n}} \times \boxed{\mathbf{V}_{n \times n}^T} \\
 (m > n)
 \end{array}$$

where  $A$  is the input data matrix (users's ratings),  $U$  is the left singular vectors (user "features" matrix),  $\Sigma$  is the diagonal matrix of singular values (essentially weights/strengths of each concept), and  $V^T$  is the right singular vectors (movie "features" matrix).  $U$  and  $V^T$  are column orthonormal, and represent different things.  $U$  represents how much users "like" each feature and  $V^T$  represents how relevant each feature is to each movie.

To get the lower rank approximation, I take these matrices and keep only the top  $k$  features, which can be thought of as the underlying tastes and preferences vectors.

## Setting Up SVD

Scipy and Numpy both have functions to do the singular value decomposition. We are going to use the Scipy function `svds` because it let's us choose how many latent factors we want to use to approximate the original ratings matrix (instead of having to truncate it after).

```
In [61]: from scipy.sparse.linalg import svds
```

```
In [62]: U, sigma, Vt = svds(Ratings_demeaned, k = 50)
```

```
In [63]: print('Size of sigma: ', sigma.size)
```

Size of sigma: 50

As we are going to leverage matrix multiplication to get predictions, we will convert the  $\Sigma$  (now are values) to the diagonal matrix form.

```
In [64]: sigma = np.diag(sigma)
```

```
In [65]: print('Shape of sigma: ', sigma.shape)
print(sigma)
```

```
Shape of sigma: (50, 50)
[[ 67.86628347  0.          0.          ...  0.          0.
   0.          ]
 [  0.          68.1967072  0.          ...  0.          0.
   0.          ]
 [  0.          0.          69.02678246 ...  0.          0.
   0.          ]
 ...
 [  0.          0.          0.          ... 184.86187801  0.
   0.          ]
 [  0.          0.          0.          ...  0.          231.22453421
   0.          ]
 [  0.          0.          0.          ...  0.          0.
  474.20606204]]
```

```
In [66]: print('Shape of U: ', U.shape)
print('Shape of Vt: ', Vt.shape)
```

```
Shape of U: (610, 50)
Shape of Vt: (50, 9724)
```

## Making Predictions from the Decomposed Matrices

We now have everything we need to make movie ratings predictions for every user. We can do it all at once by following the math and matrix multiply  $U$ ,  $\Sigma$ , and  $V^T$  back to get the rank  $k = 50$  approximation of  $A$ .

But first, we need to add the user means back to get the actual star ratings prediction.

```
In [67]: all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt) + user_ratings_mean
```

```
In [68]: print('All user predicted rating : ', all_user_predicted_ratings.shape)
```

```
All user predicted rating : (610, 9724)
```

With the predictions matrix for every user, we can build a function to recommend movies for any user. We return the list of movies the user has already rated, for the sake of comparison.

We will use the column names from the ratings df

```
In [69]: print('Rating Dataframe column names', Ratings.columns)

Rating Dataframe column names Int64Index([ 1, 2, 3, 4,
5, 6, 7, 8,
9, 10,
...
193565, 193567, 193571, 193573, 193579, 193581, 193583, 193585,
193587, 193609],
dtype='int64', name='movieId', length=9724)
```

```
In [70]: preds = pd.DataFrame(all_user_predicted_ratings, columns = Ratings.columns)
preds.head()
```

```
Out[70]:
```

movieId	1	2	3	4	5	6	7	
0	2.167328	0.402751	0.840184	-0.076281	-0.551337	2.504091	-0.890114	-0.026
1	0.211459	0.006658	0.033455	0.017419	0.183430	-0.062473	0.083037	0.024
2	0.003588	0.030518	0.046393	0.008176	-0.006247	0.107328	-0.012416	0.003
3	2.051549	-0.387104	-0.252199	0.087562	0.130465	0.270210	0.477835	0.040
4	1.344738	0.778511	0.065749	0.111744	0.273144	0.584426	0.254930	0.128

5 rows × 9724 columns

Now we write a function to return the movies with the highest predicted rating that the specified user hasn't already rated. Though we didn't use any explicit movie content features (such as genre or title), we will merge in that information to get a more complete picture of the recommendations.

```
In [71]: def recommend_movies(predictions, userID, movies, original_ratings, num_reco
.....:
Implementation of SVD by hand
:param predictions : The SVD reconstructed matrix,
userID : UserID for which you want to predict the top rated movies,
movies : Matrix with movie data, original_ratings : Original Rating matrix
num_recommendations : num of recos to be returned
:return: num_recommendations top movies
.....:

# Get and sort the user's predictions
user_row_number = userID - 1 # User ID starts at 1, not 0
sorted_user_predictions = predictions.iloc[user_row_number].sort_values(

# Get the user's data and merge in the movie information.
user_data = original_ratings[original_ratings.userId == (userID)]
user_full = (user_data.merge(movies, how = 'left', left_on = 'movieId',
sort_values(['rating'], ascending=False)
)

print('User {0} has already rated {1} movies.'.format(userID, user_full
print('Recommending highest {0} predicted ratings movies not already rat

# Recommend the highest predicted rating movies that the user hasn't seen
recommendations = (movies[~movies['movieId'].isin(user_full['movieId'])]
merge(pd.DataFrame(sorted_user_predictions).reset_index(), how = '
left_on = 'movieId',
right_on = 'movieId').
rename(columns = {user_row_number: 'Predictions'})).
sort_values('Predictions', ascending = False).
```

```
        iloc[:num_recommendations, :-1]  
    )  
  
    return user_full, recommendations
```

Let's try to recommend 20 movies for user with ID 150.

```
In [72]: already Rated, predictions = recommend_movies(preds, 150, movies, ratings, 20)
```

User 150 has already rated 26 movies.

Recommending highest 20 predicted ratings movies not already rated.

```
In [73]: # Top 20 movies that User 1310 has rated  
already Rated.head(20)
```

Out [73]:

	userId	movieId	rating	timestamp	title	genres
25	150	1356	5.0	854203229	Star Trek: First Contact (1996)	Action Adventure Sci-Fi Thriller
5	150	32	5.0	854203071	Twelve Monkeys (a.k.a. 12 Monkeys) (1995)	Mystery Sci-Fi Thriller
12	150	141	5.0	854203072	Birdcage, The (1996)	Comedy
17	150	648	4.0	854203072	Mission: Impossible (1996)	Action Adventure Mystery Thriller
2	150	6	4.0	854203123	Heat (1995)	Action Crime Thriller
4	150	25	4.0	854203072	Leaving Las Vegas (1995)	Drama Romance
6	150	36	4.0	854203123	Dead Man Walking (1995)	Crime Drama
7	150	52	4.0	854203163	Mighty Aphrodite (1995)	Comedy Drama Romance
23	150	805	4.0	854203230	Time to Kill, A (1996)	Drama Thriller
20	150	780	4.0	854203071	Independence Day (a.k.a. ID4) (1996)	Action Adventure Sci-Fi Thriller
19	150	733	4.0	854203123	Rock, The (1996)	Action Adventure Thriller
15	150	608	4.0	854203123	Fargo (1996)	Comedy Crime Drama Thriller
24	150	1073	3.0	854203163	Willy Wonka & the Chocolate Factory (1971)	Children Comedy Fantasy Musical
22	150	786	3.0	854203163	Eraser (1996)	Action Drama Thriller
21	150	784	3.0	854203163	Cable Guy, The (1996)	Comedy Thriller
18	150	653	3.0	854203163	Dragonheart (1996)	Action Adventure Fantasy
0	150	3	3.0	854203124	Grumpier Old Men (1995)	Comedy Romance
16	150	628	3.0	854203229	Primal Fear (1996)	Crime Drama Mystery Thriller
14	150	494	3.0	854203124	Executive Decision (1996)	Action Adventure Thriller
1	150	5	3.0	854203124	Father of the Bride Part II (1995)	Comedy

In [74]: *# Top 20 movies that User 1310 hopefully will enjoy predictions*

Out [74]:

	movieid	title	genres
574	736	Twister (1996)	Action Adventure Romance Thriller
0	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
211	260	Star Wars: Episode IV - A New Hope (1977)	Action Adventure Sci-Fi
607	802	Phenomenon (1996)	Drama Romance
12	17	Sense and Sensibility (1995)	Drama Romance
87	112	Rumble in the Bronx (Hont faan kui) (1995)	Action Adventure Comedy Crime
558	708	Truth About Cats & Dogs, The (1996)	Comedy Romance
599	788	Nutty Professor, The (1996)	Comedy Fantasy Romance Sci-Fi
886	1210	Star Wars: Episode VI - Return of the Jedi (1983)	Action Adventure Sci-Fi
634	852	Tin Cup (1996)	Comedy Drama Romance
565	719	Multiplicity (1996)	Comedy
1047	1393	Jerry Maguire (1996)	Drama Romance
80	104	Happy Gilmore (1996)	Comedy
9	14	Nixon (1995)	Drama
532	661	James and the Giant Peach (1996)	Adventure Animation Children Fantasy Musical
587	762	Striptease (1996)	Comedy Crime
4	9	Sudden Death (1995)	Action
621	832	Ransom (1996)	Crime Thriller
523	637	Sgt. Bilko (1996)	Comedy
103	140	Up Close and Personal (1996)	Drama Romance

These look like pretty good recommendations. It's good to see that, although we didn't actually use the genre of the movie as a feature, the truncated matrix factorization features "picked up" on the underlying tastes and preferences of the user. We have recommended some Action, Adventure, Romance, Thriller movies - all of which were genres of some of this user's top rated movies.

## Model Evaluation

Can't forget to evaluate our model, can we?

Instead of doing manually like the last time, we will use the Surprise library that provided various ready-to-use powerful prediction algorithms including (SVD) to evaluate its RMSE (Root Mean Squared Error) on the MovieLens dataset. It is a Python scikit building and analyzing recommender systems.

```
In [75]: # Import libraries from Surprise package
import surprise
```



```

from surprise import SVD, Reader
from surprise import Dataset
from surprise.model_selection import cross_validate
from surprise.model_selection import KFold

# Load Reader library
reader = Reader()

# Load ratings dataset with Dataset library
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)

# Split the dataset for 5-fold evaluation
#data.split(n_folds=5)

kf = KFold(n_splits=5)
kf.split(data)

```

Out[75]: <generator object KFold.split at 0x12ceedf20>

```

In [76]: # Use the SVD algorithm.
svd = SVD()

# Compute the RMSE of the SVD algorithm.
#evaluate(svd, data, measures=['RMSE'])

```

```

In [77]: cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

```

Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.8787	0.8671	0.8752	0.8722	0.8743	0.8735	0.0038
MAE (testset)	0.6772	0.6652	0.6696	0.6710	0.6705	0.6707	0.0039
Fit time	5.52	5.23	5.03	5.09	5.06	5.19	0.18
Test time	0.16	0.14	0.14	0.14	0.24	0.16	0.04

```

Out[77]: {'test_rmse': array([0.8787132 , 0.86712917, 0.87519903, 0.87220126, 0.8743
3722]),
'test_mae': array([0.67720385, 0.66516829, 0.6696205 , 0.67098869, 0.67054
537]),
'fit_time': (5.5246899127960205,
5.228315114974976,
5.030495882034302,
5.091771841049194,
5.063137054443359),
'test_time': (0.15938782691955566,
0.1427910327911377,
0.14094209671020508,
0.14005708694458008,
0.23831486701965332)}

```

We get a mean Root Mean Square Error of 0.87 which is pretty good. Let's now train on the dataset and arrive at predictions.

```

In [78]: trainset = data.build_full_trainset()
svd.fit(trainset)

```

Out[78]: <surprise.prediction\_algorithms.matrix\_factorization.SVD at 0x12c7ae130>

We will pick again user with ID 150 and check the ratings he has given.

```

In [79]: ratings[ratings['userId'] == 150]

```

Out[79]:

	userId	movieId	rating	timestamp
22277	150	3	3.0	854203124
22278	150	5	3.0	854203124
22279	150	6	4.0	854203123
22280	150	7	3.0	854203124
22281	150	25	4.0	854203072
22282	150	32	5.0	854203071
22283	150	36	4.0	854203123
22284	150	52	4.0	854203163
22285	150	58	3.0	854203163
22286	150	62	3.0	854203072
22287	150	79	3.0	854203229
22288	150	95	3.0	854203072
22289	150	141	5.0	854203072
22290	150	376	3.0	854203124
22291	150	494	3.0	854203124
22292	150	608	4.0	854203123
22293	150	628	3.0	854203229
22294	150	648	4.0	854203072
22295	150	653	3.0	854203163
22296	150	733	4.0	854203123
22297	150	780	4.0	854203071
22298	150	784	3.0	854203163
22299	150	786	3.0	854203163
22300	150	805	4.0	854203230
22301	150	1073	3.0	854203163
22302	150	1356	5.0	854203229

Now let's use SVD to predict the rating that User with ID 150 will give to a random movie (let's say with Movie ID 1994).

In [80]: `svd.predict(150, 1994)`Out[80]: `Prediction(uid=150, iid=1994, r_ui=None, est=3.5987436795302594, details={'was_impossible': False})`

For movie with ID 1994, I get an estimated prediction of 3.18210. The recommender system works purely on the basis of an assigned movie ID and tries to predict ratings based on how the other users have predicted the movie.

In [81]: `svd.predict(150, 100)`Out[81]: `Prediction(uid=150, iid=100, r_ui=None, est=3.304750834809633, details={'was_impossible': False})`

```

In [82]: svd.predict(150, 194)

Out[82]: Prediction(uid=150, iid=194, r_ui=None, est=3.7534718598619397, details={'was_impossible': False})

In [83]: svd.predict(150, 190)

Out[83]: Prediction(uid=150, iid=190, r_ui=None, est=3.315415422759387, details={'was_impossible': False})

In [84]: from __future__ import (absolute_import, division, print_function,
                                unicode_literals)

        from surprise import SVDpp
        from surprise import SVD
        from surprise import Dataset
        from surprise import accuracy
        from surprise.model_selection import train_test_split
        from surprise.model_selection import GridSearchCV
        from surprise.model_selection import cross_validate

In [85]: # Use movielens-100K
data = Dataset.load_builtin('ml-100k')
trainset, testset = train_test_split(data, test_size=.15)

type(data)

Out[85]: surprise.dataset.DatasetAutoFolds

```

## SVD++

To build a robust recommender system, we need to develop models which factor in both explicit and implicit user feedback. For our Movielens dataset, a less obvious kind of implicit data does exist. The dataset does not only tell us the rating values, but also which movies users rate, regardless of how they rated these movies. In other words, a user implicitly tells us about her preferences by choosing to voice her opinion and vote a (high or low) rating. This reduces the ratings matrix into a binary matrix, where "1" stands for "rated", and "0" for "not rated". Admittedly, this binary data is not as vast and independent as other sources of implicit feedback could be. Nonetheless, we have found that incorporating this kind of implicit data – which inherently exist in every rating based recommender system – significantly improves prediction accuracy. SVD++ factors in this implicit feedback and gives better accuracy as shown below.

```

In [86]: algo_svdpp = SVDpp(n_factors=160, n_epochs=10, lr_all=0.005, reg_all=0.1)
algo_svdpp.fit(trainset)
test_pred = algo_svdpp.test(testset)
print("SVDpp : Test Set")
accuracy.rmse(test_pred, verbose=True)

SVDpp : Test Set
RMSE: 0.9377
0.9377385757264571

Out[86]:

```

We have explored and used Surprise package: This package has been specially developed to make recommendation based on collaborative filtering easy. It has default implementation for a variety of CF algorithms.

## Evaluating Collaborative Filtering

**Hit Ratio** It is ratio of number of hits/ Total recommendation

In [87]: `user_id=50`

```
In [88]: def evaluation_collaborative_svd_model(userId,userOrItem):
        """
        hybrid the functionality of Collaborative based and svd based model to s
        :param userId: userId of user, userOrItem is a boolean value if True it
        :return: dataframe of movies and ratings
        """
        movieIdsList= list()
        movieRatingList=list()
        movieIdRating= pd.DataFrame(columns=['movieId','rating'])
        if userOrItem== True:
            movieIdsList=getRecommendedMoviesAsperUserSimilarity(userId)
        else:
            movieIdsList=recommendedMoviesAsperItemSimilarity(user_id)
        for movieId in movieIdsList:
            predict = svd.predict(userId, movieId)
            movieRatingList.append([movieId,predict.est])
            movieIdRating = pd.DataFrame(np.array(movieRatingList), columns=['mo
            count=movieIdRating[(movieIdRating['rating'])>=3]['movieId'].count()
            total=movieIdRating.shape[0]
            hit_ratio= count/total
        return hit_ratio
```

```
In [89]: print("Hit ratio of User-user collaborative filtering")
print(evaluation_collaborative_svd_model(user_id,True))
print("Hit ratio of Item-Item collaborative filtering")
print(evaluation_collaborative_svd_model(user_id,False))
```

```
Hit ratio of User-user collaborative filtering
0.4444444444444444
Hit ratio of Item-Item collaborative filtering
0.8888888888888888
```

# Hybrid model:

After developing individual models as discussed earlier, we now stack them in order to get better results.

## Content Based Filtering + SVD

Steps:

1. Run Content based filtering and determine the movies which we want to recommend to the user.
2. Filter and sort the recommendations of CF using SVD predicted ratings.

```
In [90]: df_movies=movies
def hybrid_content_svd_model(userId):
    """
    hybrid the functionality of content based and svd based model to recommend
    :param userId: userId of user
    :return: list of movies recommended with rating given by svd model
    """
    recommended_movies_by_content_model = get_recommendation_content_model(df_movies)
    recommended_movies_by_content_model = df_movies[df_movies.apply(lambda r: r['movieId'] in recommended_movies_by_content_model.iterrows()):
    for key, columns in recommended_movies_by_content_model.iterrows():
        predict = svd.predict(userId, columns["movieId"])
        recommended_movies_by_content_model.loc[key, "svd_rating"] = predict
    #         if(predict.est < 2):
    #             recommended_movies_by_content_model = recommended_movies_by_content_model
    return recommended_movies_by_content_model.sort_values("svd_rating", ascending=False)

hybrid_content_svd_model(user_id)
```

```
/usr/local/lib/python3.9/site-packages/pandas/core/indexing.py:1596: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
self.obj[key] = _infer_fill_value(value)
```

```
/usr/local/lib/python3.9/site-packages/pandas/core/indexing.py:1763: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
isetter(loc, value)
```

Out [90] :

	movieId	title	genres	svd_rating
257	296	Pulp Fiction (1994)	Comedy Crime Drama Thriller	3.636076
46	50	Usual Suspects, The (1995)	Crime Mystery Thriller	3.625740
3638	4993	Lord of the Rings: The Fellowship of the Ring,...	Adventure Fantasy	3.584074
933	1233	Boot, Das (Boat, The) (1981)	Action Drama War	3.498524
224	260	Star Wars: Episode IV - A New Hope (1977)	Action Adventure Sci-Fi	3.489179
4137	5952	Lord of the Rings: The Two Towers, The (2002)	Adventure Fantasy	3.484158
1702	2289	Player, The (1992)	Comedy Crime Drama	3.459736
934	1234	Sting, The (1973)	Comedy Crime	3.442752
210	246	Hoop Dreams (1994)	Documentary	3.430844
686	904	Rear Window (1954)	Mystery Thriller	3.400115
910	1209	Once Upon a Time in the West (C'era una volta ...	Action Drama Western	3.396319

## Conclusion

We implemented and evaluated different widely used Movie Recommendation models. Also, we developed our own models like the hybrid model (Content based + SVD) as discussed earlier. We evaluated each model with the appropriate evaluation metric. We implemented novel technique to evaluate collaborative filtering algorithm by using SVD and hit ratio as a metric

We attempted to build a model-based Collaborative Filtering movie recommendation sytem based on latent features from a low rank matrix factorization method called SVD and SVD++. As it captures the underlying features driving the raw data, it can scale significantly better to massive datasets as well as make better recommendations based on user's tastes. For SVD model we get an RMSE of **0.87** and for SVD++ model we get an RMSE of **0.938**

## Citation

- <https://www.datacamp.com/community/tutorials/recommender-systems-python>
- <https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0>
- [https://medium.com/@james\\_aka\\_yale/the-4-recommendation-engines-that-can-predict-your-movie-tastes-bbec857b8223](https://medium.com/@james_aka_yale/the-4-recommendation-engines-that-can-predict-your-movie-tastes-bbec857b8223)
- <http://www.awesomestats.in/python-recommending-movies/>
- [https://medium.com/@james\\_aka\\_yale/the-4-recommendation-engines-that-can-predict-your-movie-tastes-bbec857b8223](https://medium.com/@james_aka_yale/the-4-recommendation-engines-that-can-predict-your-movie-tastes-bbec857b8223)
- [https://en.wikipedia.org/wiki/Singular\\_value\\_decomposition](https://en.wikipedia.org/wiki/Singular_value_decomposition)
- <https://github.com/gpfvic/IRR/blob/master/Factorization%20meets%20the%20neighbor%20a%20multifaceted%20collaborative%20filtering%20model.pdf>

- <https://surprise.readthedocs.io/en/stable/index.html>
- <https://www.quora.com/Whats-the-difference-between-SVD-and-SVD++>
- <https://blog.statsbot.co/singular-value-decomposition-tutorial-52c695315254>
- <https://medium.com/recombee-blog/machine-learning-for-recommender-systems-part-1-algorithms-evaluation-and-cold-start-6f696683d0ed>

In [ ]: