DSC 478: Final Project Report

Movie Recommender System

By: Paripon Thantong(PT), Yili Lin(YL), Harsha Puvvada(HP)

Executive Summary (HP):

This report showcases the findings from a movie recommender system built as part of the final project for DSC478. Initially, the user is shown the top 50 movies based on the IMDB weighted score and IMDB popularity. The user picks 5 movies that they have seen before and rates them out of 10. The recommender system is built using these algorithms: Term Frequency-Inverse Document Frequency, Singular Value Decomposition, Cosine Similarity and K nearest Neighbors. It then performs bagging based on majority vote to find the best movie recommendations based on the different results from the various algorithms. The algorithm then returns the top 10 movie picks for the user. The various algorithms and their recommendations are then evaluated.

Introduction (YL):

In 2018, there were over 5,000 different Tv shows and movies on Netflix, millions of items on amazon, billions of users on Facebook and millions on Twitter. Netflix always knows the list of movies and tv shows that a user would want to watch next. Amazon knows the potential shopping list of users sometimes even better than the users themselves. Facebook and Twitter always send out the right possible known friend requests and display the right kinds of ads to users. This is all made possible using recommender systems. Most companies with huge databases use recommender systems. A recommender system is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. Recommender systems have been developed to explore research articles and experts, collaborators, financial services, and life insurance. They also help a user navigate through the huge amount of consumable goods and services.

In this paper, we develop a movie recommender system based on user ratings with these algorithms: Term Frequency-Inverse Document Frequency, Singular Value Decomposition, Cosine Similarity, and K nearest Neighbors. The recommendations from these machine learning algorithms were then bagged to vote for the five final recommendations. In this project, a user is asked to give ratings to five different movies. To make it easier for the user to recall various movies they have watched before, the program shows a list of top 25 movies based on IMDB weighted Score and top 25 movies based on IMDB popularity. Each method generates five movie recommendations per user rated movie. So, this means that each machine learning model generates twenty-five movie recommendations. Bagging of the movie recommendations is done by counting frequencies and choosing the top 5 movie recommendations that appear in the greatest number of machine learning algorithms.

Dataset:

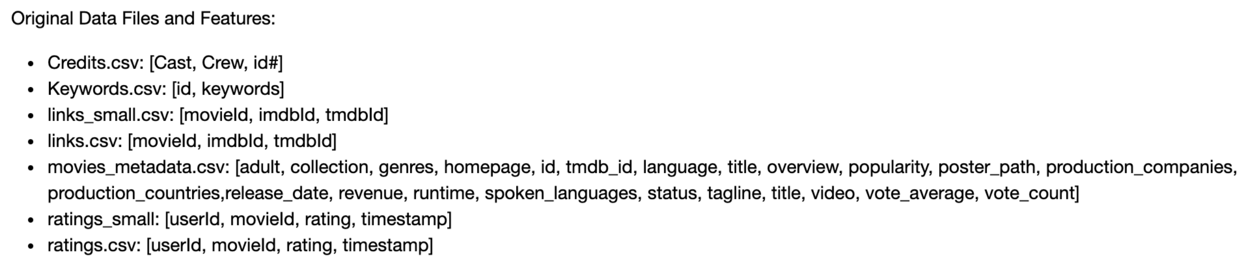
Overview (PT):

As part of the knowledge discovery process, data integration, cleaning and visualization were performed on the dataset to obtain the most useful information. The datasets on Kaggle were originally collected from the IMDB website. Some more data had been added before being posted on Kaggle. The dataset had 45,000 movies with variables such as cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts and vote averages and had 7 total files containing other information like cast, crew and IMDB user ratings.

Methodology:

Data Integration, Manipulation and Visualization are found in the notebook entitled: FinalProject_DataCleaning_Exploration.ipynb

Data Integration ***(HP)****:*



Original Data Files and Features:

- Credits.csv: [Cast, Crew, id#]
- Keywords.csv: [id, keywords]
- links_small.csv: [movieId, imdbId, tmdbId]
- links.csv: [movieId, imdbId, tmdbId]
- movies_metadata.csv: [adult, collection, genres, homepage, id, tmdb_id, language, title, overview, popularity, poster_path, production_companies, production_countries,release_date, revenue, runtime, spoken_languages, status, tagline, title, video, vote_average, vote_count]
- ratings_small: [userId, movieId, rating, timestamp]
- ratings.csv: [userId, movieId, rating, timestamp]

Figure 1: List of files and their features

Figure 1 above shows all the files obtained from the original data set obtained from kaggle. “links_small.csv” and “links.csv” were deleted as they were not relevant to the project. Unneeded columns were removed from “movies_metadata.csv”. Keywords from “keywords.csv” were then merged into the “movies_metadata.csv”. Cast and crew information from the “credits.csv” were then merged into “movies_metadata.csv”. The aim of this is to have all the movie metadata in “movies_metadata.csv” and all the user ratings would be from “ratings_small.csv” and “ratings.csv”. This way the recommender system only has to rely on two separate data files.

*Data Cleaning **(HP)**:*

For “movies_metadata.csv”, empty, invalid and missing rows were dropped. Non-English language movies were dropped. Some of the columns like ‘genres’, ‘cast’ and ‘keywords’ were in JavaScript Object Notation (JSON), so they were converted to a list of strings for easier data manipulation. The “release_date” was modified so that it showed the year the movie was released. All movies older than 1980 were also removed. After this, there were 16,470 movies left in the cleaned dataset.

Dropping some movies in the “movies_metadata.csv” meant that there were some movies still listed in the “ratings.csv” and “ratings_small.csv” that had been dropped. The changes were merged, and the movies were standardized throughout all the files. After all the cleaning, the project was left with “cleanedRatings.csv”, “cleanedMoviesMetadata.csv” and “cleanedSmallRatings”. These three files were used to do the data visualization and to run the Machine Learning Models.

*Data Visualization **(YL)**:*

As shown in Table 1 below, there were 16,470 movies. The original rating file includes around 6 million ratings. Due to the lack of computational power, this project only used the small rating file with 23,712 ratings.

Table 1: Cleaned Data files total number of variables

File name	Count
“cleanedMoviesMetadata.csv”	16,470 movies
“cleanedRatings.csv”	5,979,146 ratings
“cleanedSmallRatings”	23,712 ratings

The next few figures showcase the various insights that were gained from further data exploration.

In figure 2, the number of movies released per year increased with time. This is to be expected as the demand for the movie market and developments in movie technology push the movie industry to produce more movies every year. The number of movies produced in 2016 were almost the same as that of total number of years between 1980 to 1984.

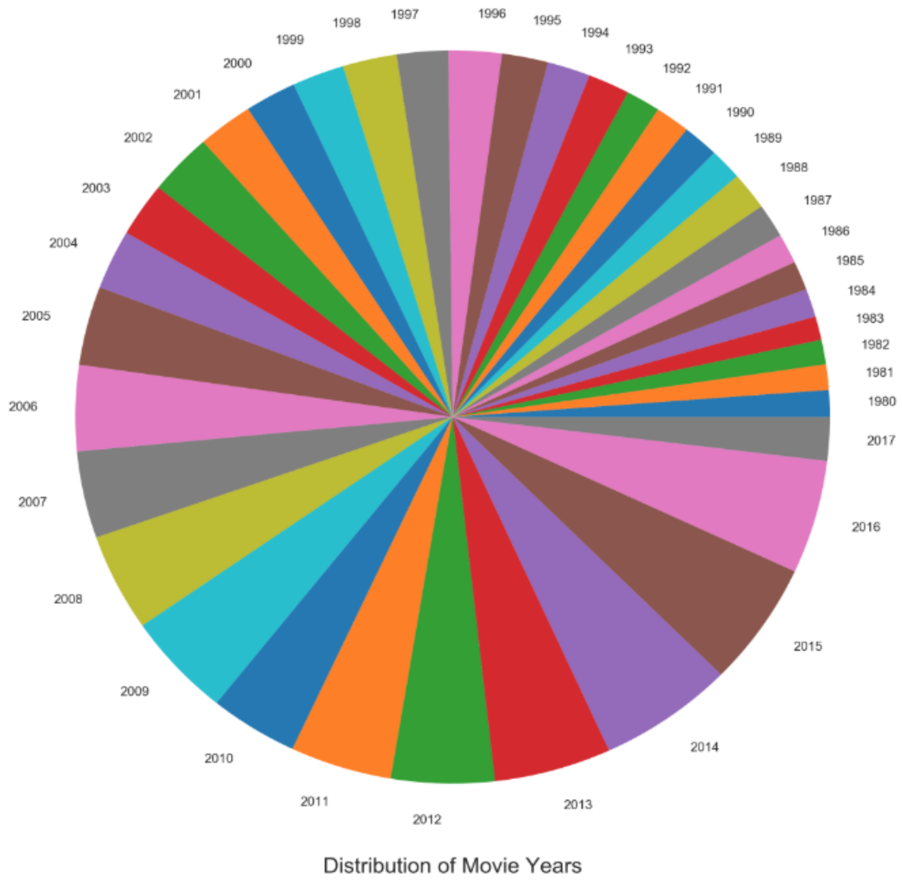


Figure 2: Distribution of movies over the years

In this data set, every movie row includes several genres to identify the movie. As seen in Figure 3, the most popular genre is Drama, followed by comedy and thriller.

```
'Animation'            752
'Comedy'              5282
'Family'              1406
'Adventure'           1606
'Fantasy'             1054
'Romance'             2453
'Drama'               7084
'Action'              2779
'Crime'               1739
'Thriller'            3731
'Horror'              2267
'History'              458
'Science Fiction'     1546
'Mystery'             1033
'War'                  310
'Music'                630
'Documentary'         1949
'Foreign'              262
'Western'              162
'TV Movie'             436
dtype: int64
```

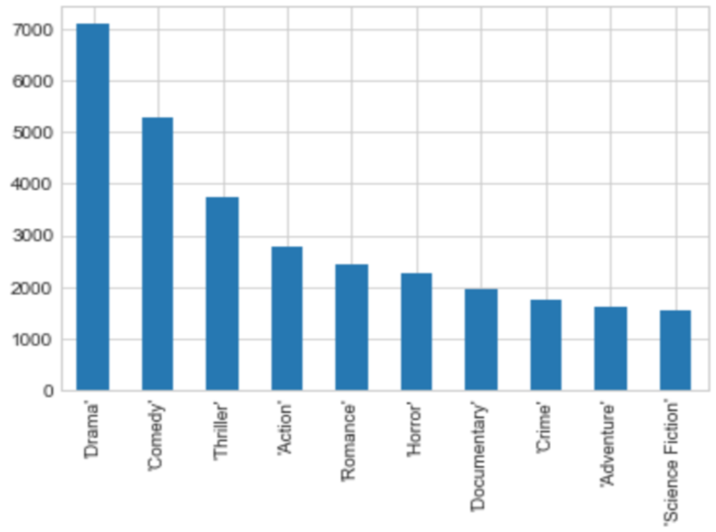


Figure 3: Genre distribution of movies

Figure 4 below shows the visualization of the words in title, the most frequently words are girls, man and love, seems like story in drama movies. Word death, dead life, night seem to the thriller movies.

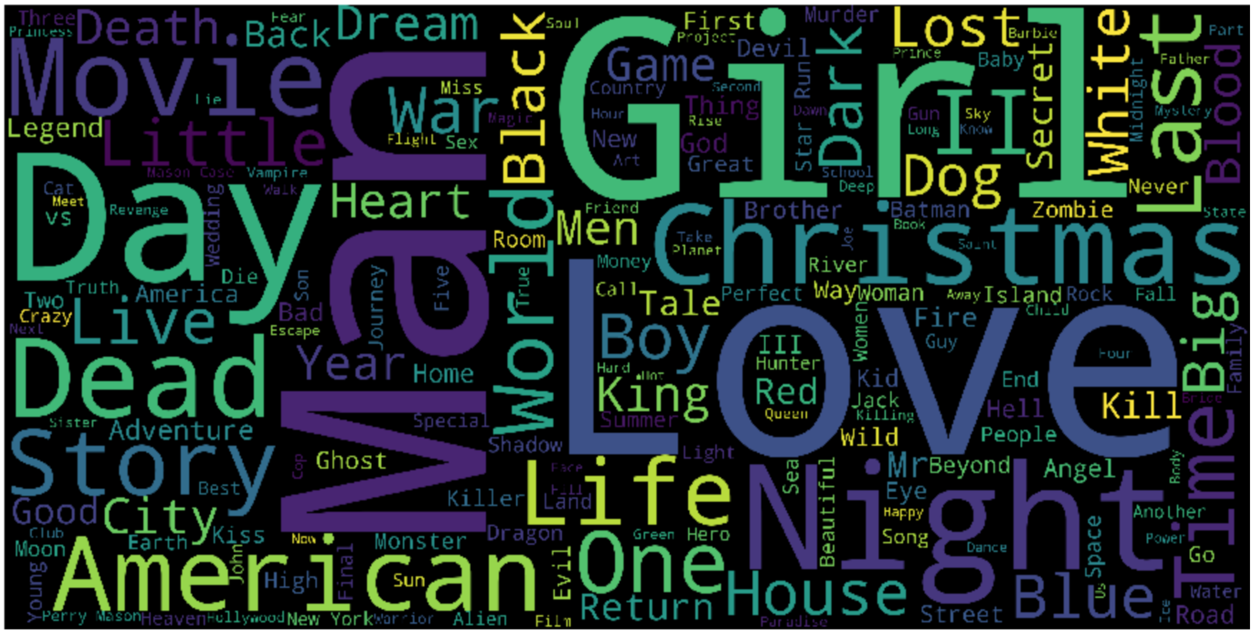


Figure 4: Word Cloud of Movie Titles

In figure 5, the most popular movie in the dataset is the Minions, almost double the popularity of the second-placed movie: wonder woman.

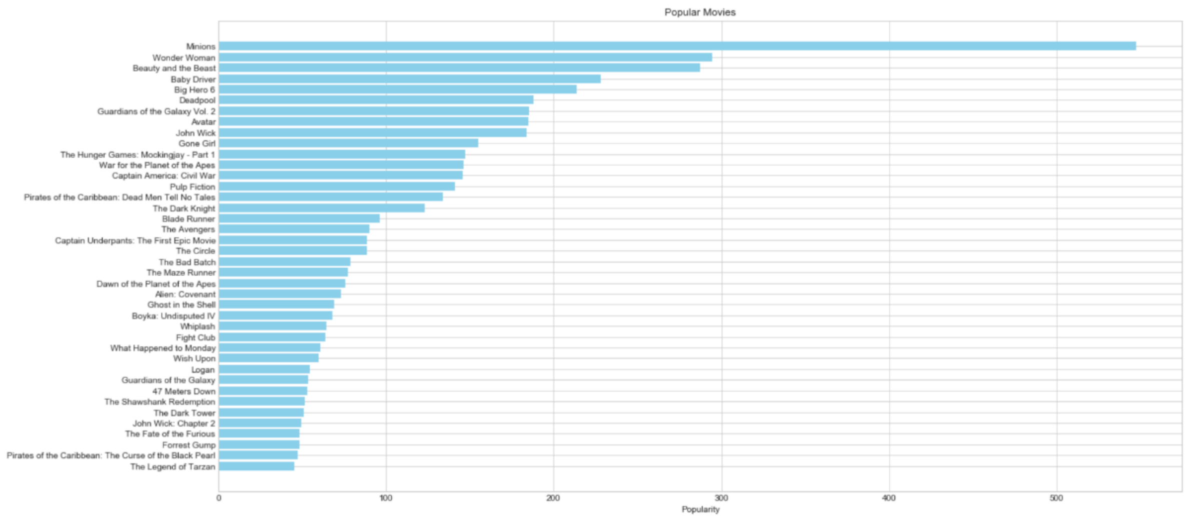


Figure 5: Most popular movies according to User ratings

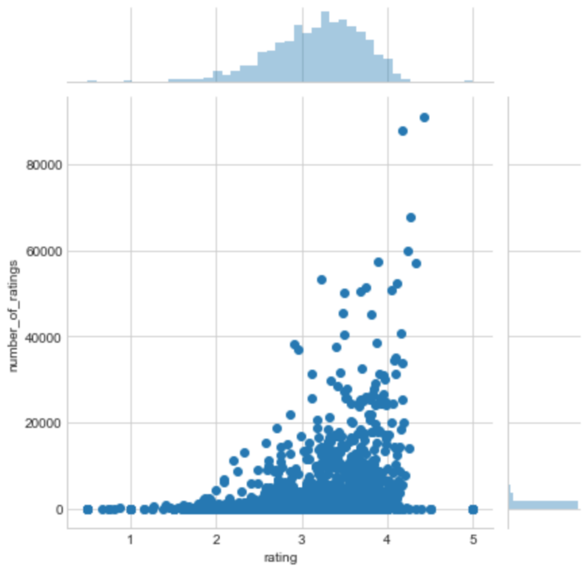


Figure 6: Total number of Ratings vs Average Rating and movie distribution over the years

In figure 6 above, the plots show the relationship between voting number and year, the relationship between average rating and year, the relationship between vote number and average rating score and the rating score count. The most common vote ratings are three and four, which caused the average of most movies to be in the same range. Average rating numbers for the movie bloomed up since the internet came into daily use in the late 20th century. With the help of the internet, everyone can rate every movie inside the IMDB data base, which may have caused the average of moving rating to fall down a little bit.

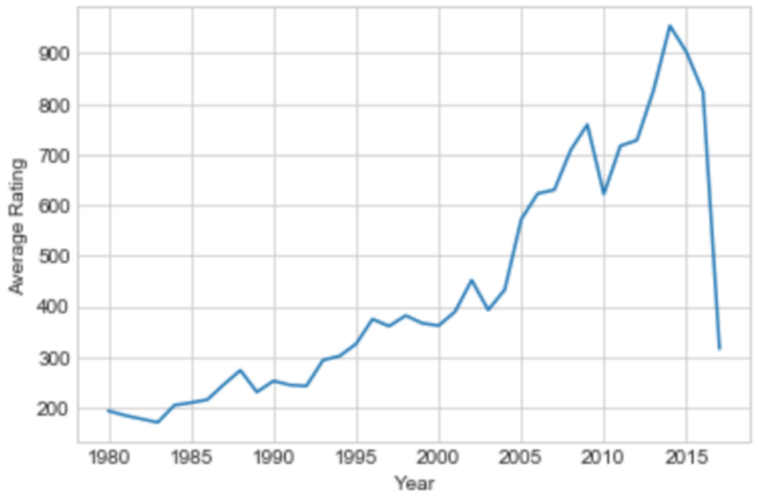


Figure 7: Average number of user ratings per movie per year

Figure 7 shows the relationship between the number of ratings and year. The average seems to be around 600 user ratings per movie.

Machine Learning Models:

Content Based Algorithms:

*Displaying top 50 movies at beginning of program and user Input **(HP):***

Initially, the user is presented with the top 25 movies based on the IMDB weighted score and the top 25 movies based on popularity. This content-based algorithm works by calculating the IMDB Weighted Score. The IMDB Weighted Score takes into account the popularity and the average vote based on IMDB users.

Weighted Rating (WR) = $\left(\frac{v}{v+m} . R\right) + \left(\frac{m}{v+m} . C\right)$

where,

- *v* is the number of votes for the movie
- *m* is the minimum votes required to be listed in the chart
- *R* is the average rating of the movie
- *C* is the mean vote across the whole report

Figure 8: Equation for calculating the IMDB Weighted Rating

Next, the top 25 movies are shown by sorting based on popularity. Figure 8 above shows how the weighted IMDB score is calculated. Presenting the top 50 makes it easier for the user to name and rate 5 movies that they have seen before.

*TF-IDF + cosine similarity based on [title, genres, cast, keywords, overview **(HP):***

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 weighting negates the effect of high frequency words in determining the importance of an item (document).

$$\mathbf{tfidf}_{i,j} = \mathbf{tf}_{i,j} \times \log\left(\frac{\mathbf{N}}{\mathbf{df}_i}\right)$$

$\mathbf{tf}_{i,j}$ = total number of occurences of i in j
$\mathbf{df}_i$ = total number of documents (speeches) containing i
$\mathbf{N}$ = total number of documents (speeches)

Figure 9: Equation for calculating tf-idf values

The equation above shows how the tf-idf values are calculated. For this paper's algorithm, tf-idf values are calculated based on the overview of the movies. The overview for the user rated movies are compared with all the movie overviews. Cosine similarity values are calculated for these tf-idf values and the movies that have the closest cosine similarity are presented as results.

Collaborative Filtering Algorithms:

*Singular value decomposition (SVD)**(YL)**:*

SVD is a factorization of a complex matrix. It is the generalization of the eigen decomposition of a normal matrix to any m*n matrix using an extension of the polar decomposition. The former matrix is a factorization of U, Σ and V, where U is an m*m complex unitary matrix, Σ is an m*n rectangular diagonal matrix with non-negative real numbers on the diagonal, and V is an n*n complex unitary matrix.

The input rating of 5 movies are inserted into the rating dataset as the user id 672. Then two data sets are merged into the data frame combining the rating, the movie titles and the movie id. A movie matrix is created with the index of user id and the columns of movie id, filling the NA values with 0.

U, Σ and V are processed with the svds from the scikit-learn package. The U, Σ and V are then used to create all-users prediction matrix. From the matrix, the closet movie list is created based on the rating information of the input movie ratings.

*Cosine similarity **(YL)**:*

Cosine similarity is a measure of similarity between two vectors that measure the cosine of the angle between them, as shown in Figure 10. By applying the definition of similarity, this will be in fact equal to 1 if the two vectors are identical, and it will be 0 if the two are orthogonal.

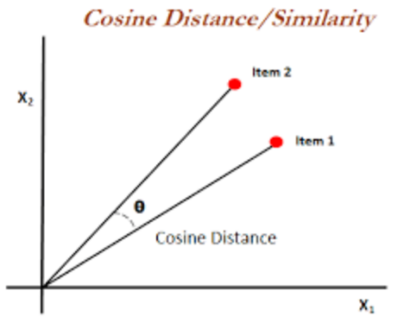


Figure 10: Cosine similarity

In other words, the similarity is a number bounded between 0 and 1 that tells us how much the two vectors are similar.

The process of cosine similarity is very close to the SVD process. After a matrix with the indexes of user id, columns of movie id, the values of ratings for each movie is created, the method uses the cosine_similarity from the scikit-learn package to calculate the cosine similarity between the movies. Finally, the program returns the five most similar movie based on one input movie. In this project five movies are fed to the method which then returns a list of twenty-five movies.

K-nearest neighborhood search - Unsupervised method ***(PT):***

In this method, the recommendation system is applied with a K-Nearest Neighbors (KNN) search algorithm to find similar movies to the list of movies that the user selected. The KNN method used in this method is unsupervised. KNN will cluster the closest movies and recommend it to the user. The distance parameter is set to be cosine similarity and with brute force search. To begin with, two datasets are pre-processed for which the number of user ratings are counted and assigned to a variable as total rating counts for each movie. Then, the new variable is merged with the movie metadata data frame. The duplicate data points are dropped, and the data frame is reshaped with the pivot method in pandas. Once the data frame is in a pivot table, the SciPy sparse function is applied to it.

For the algorithm applied in this method, nearest neighborhood search from sci-kit learn is applied. Before recommending the movie, average movie rating over 4.5 is computed and 10 of the movies are randomly selected as a base line for recommending. Finally, the recommender function is implemented. The function requires the index of the movie list with the average rating over 4.5. As a result, it returns 10 closest movies with their distances.

Bagging ***(HP)****:*

The four different machine learning models presented in this paper will recommend 25 movies each for a total of 100 movie recommendations. Bagging is performed to combine all of the models' results. The frequency distribution of the movie recommendations are analyzed and sorted in descending order. A movie that appears in more models will be more likely to be recommended to the user. The top 5 most common movies amongst the models are presented as recommendations to the user.

Results:

For the demo, the five user movies and users ratings are: 1408 - rating 9/10, 2 Days in Paris - 7/10, 2010 rating 8/10, 24 Hours Party People rating 8/10, xXx rating 10/10. Table 2 below shows the 25 recommendations from each method.

Table 2: The 25 Recommendations from each Machine Learning Model

TF-IDF + Cosine Similarity Recommender	KNN Search	SVD	Cosine Similarity
Old Enough	Jarhead	Fools Rush In	Catch Me If You Can
Vincent N Roxxy	Almost Famous	Scarface	Angel Baby
Hot Girls Wanted	Armageddon	The Good Shepherd	An American Tail
Seventh Son	Men in Black II	The Bachelor	Children of the Corn IV: The Gathering
The Opponent	Dancer in the Dark	Swept from the Sea	The Machinist
The Reckoning	Absolute Power	Gerry	Desperately Seeking Susan
Hotel Transylvania 2	Ernest Saves Christmas	The Chronicles of Riddick: Dark Fury	Long Pigs
VeggieTales: The Pirates Who Don't Do Anything	Finding Forrester	Galaxy Quest	Paranoid Park
The Congress	Snatch	Dawn of the Dead	Dangerous Game
Striking Distance	The Queen	Syriana	The Believer
Just Before Dawn	Trick	Terminator 3: Rise of the Machines	The Squeeze
Midnight Crossing	The Great Global Warming Swindle	Speed 2: Cruise Control	Elizabeth
Messages Deleted	As Good as It Gets	Ghost Dog: The Way of the Samurai	The Times of Harvey Milk
Mission to Mir	Spider-Man 2	Saw II	Catwoman
The Fraternity	Land of Plenty	Short Circuit	Evening
Familiar Strangers	Elvira, Mistress of the Dark	My Best Friend's Wedding	At Risk
Only the Strong Survive	Jarhead	Arlington Road	The Weather Man
Cradle 2 the Grave	Interview with the Vampire	Light of Day	Cars 2
Mickey's Christmas Carol	Flashdance	The Out-of-Towners	A Night in Heaven
Pure Country	Lock, Stock and Two Smoking Barrels	Finding Neverland	My Tutor
Perry Mason: The Case of the Fatal Framing	Chasing Amy	Local Color	Return to Me
Charlie and the Chocolate Factory		Star 80	The Great Outdoors

Point Break The Three Burials of Melquiades Estrada Wavemakers	Wet Hot American Summer The Net The Shawshank Redemption Inside Man	A Brief History of Time Married to the Mob Under Siege 2: Dark Territory	Harry Potter and the Philosopher's Stone Dangerous Game Snow Cake

The five final recommended movies with bagging are Jarhead, Dangerous Game, Children of the Corn IV: The Gathering, Long Pigs and Gerry. Evaluating the results proved to be hard since there is no objective way to predict a user's taste.

Conclusion (HP):

The paper showcases the entire Knowledge Discovery Pipeline. It goes from data integration, data cleaning, data visualization and data mining. The recommender system is built using these algorithms: Term Frequency-Inverse Document Frequency, Singular Value Decomposition, Cosine Similarity and K nearest Neighbors. The user enters 5 movies and the algorithm combines the different results from the various machine learning models to recommend 5 movies to the user. The algorithm also lays down the groundwork for a more complex recommender system.

Future Work and Discussion (YL):

In this project, the recommender system works separately to generate the recommendations. The top 25 recommendations vary from each algorithm. And due to the lack of computational power, the system runs only on part of the rating data set.

In future work, genres can be added to the algorithm as part of the recommender system since people liking one movie are very likely to enjoy similar movie from the same genres. The four methods can be ensembled and weighted to output the top recommendations. Another way to make predictions is to use neural networks. They have very high accuracy and there has been lots of success using them in movie recommender systems.

References:

Brownlee, Jason. "A Gentle Introduction to Sparse Matrices for Machine Learning." Machine Learning Mastery, 9 Aug. 2019, https://machinelearningmastery.com/sparse-matrices-for-machine-learning/.

"Sklearn.neighbors.NearestNeighbors." Scikit, https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html#sklearn.neighbors.NearestNeighbors.

Banik, Rounak. "The Movies Dataset." Kaggle, Kaggle.com, 10 Nov. 2017, https://www.kaggle.com/rounakbanik/the-movies-dataset#movies_metadata.csv.

Sharma, Pulkit. "Comprehensive Guide to Build Recommendation Engine from Scratch." Analytics Vidhya, 4 Sept. 2019, https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/.

Harrington, Peter. Machine Learning in Action. Manning Publications, 2012.