Movie Recommendation System

Team: Deep Blue Deep Learning

Members: Haoyu Zhang, Chuanrui Liu, Yunzhou Liu, Dingyuan Liu, Yiyao Qu, Yumeng Zhang

"It is **NOT OK** to use an anonymized version of this report as an example of a great project for future classes"

Part 1: Problem Explanation and Project Goal

In the information era, we are living under the significant impacts from algorithms inside various recommendation systems. Every day we can experience how current recommendation systems help us identify and extend our knowledge on the basis of our interests. If we shop a mug on Amazon, on the page of one mug we click in, there would display several lists of "products related to this item" and "customers who viewed this item also viewed". When we watch videos on YouTube, we can see the platform has already created a list of movies that we might be interested in watching after the current video ends. Data is being used, being analyzed, and being utilized as strong predictors to generate useful predictions.

Motivated by the various recommendation systems, such as the ones designed by Amazon and YouTube, we decided to focus on building recommendation systems for a user-based online service to make recommendations on movies.

Our goal is to empower the movie recommendation systems to give some feasible movie suggestion lists based on the features of each movie or user-related information. In this case, the assumed circumstance would be that when a user lists a movie he or she likes, our recommendation systems would generate a recommendation list of movies that are closely correlated with the input movie from different perspectives. In this way, the recommendation system would hopefully expand the user's interest field, just like how Amazon, YouTube, or Spotify impact us in our daily lives.

Part 2: Dataset introduction and Exploratory data analysis

- Dataset introduction

The dataset we use for this project is downloaded from Kaggle under the link https://www.kaggle.com/rounakbanik/the-movies-dataset.

This dataset, "The Movies Dataset", includes the credits, keywords, average ratings, and content information of 45,000 movies listed in the full MovieLens Dataset and 26 million ratings (scaled from 1-5) from 270,000 users. To specify, this dataset contains 7 files, including credits.csv, keywords.csv, links.csv, links_small.csv, movies_metadata.csv, ratings.csv, and ratings_small.csv, which provides sufficient movie data from different perspectives. For our project, we mainly use 4 of those files, including movies_metadata.csv, credits.csv, keywords.csv, and ratings.csv. The major file movies_metadata.csv contains 45,000 movies from the MovieLens dataset with abundant features, including genres, budget,

languages, posters, revenues, runtime, release date, and so on, which allows further exploration. The credits.csv file contains the information of cast and crew for all these movies. The keywords.csv file contains the keywords describing key plots for MovieLens movies, and the ratings.csv file contains the ratings given by users.

We played with the raw data and planned to design two recommendation systems, a content-based recommendation system and a collaborative recommendation system. The first one focuses on the similarity between the movies themselves, and the second one is based on different users' ratings for each movie.

1. *Content-based Recommendation System*: We obtained the dataset for analysis by joining the raw data in movies_metadata.csv, credits.csv and keywords.csv on movie ID.

```
2 new.head()
```

Out[44]:

cast	crew	id	adult	belongs_to_collection	budget	genres	homepage	imdb_id	original_language	...	rel
ast_id': 14, racter': 'Woody oice)',...	[{'credit_id': '52fe4284c3a36847f8024f49', 'de...	862	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	http://toystory.disney.com/toy-story	tt0114709	en	...	1
ast_id': 1, racter': 'Alan ish', '...	[{'credit_id': '52fe44bfc3a36847f80a7cd1', 'de...	8844	False	NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	NaN	tt0113497	en	...	1
ast_id': 2, racter': 'Max ldman', 'c...	[{'credit_id': '52fe466a9251416c75077a89', 'de...	15602	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...	NaN	tt0113228	en	...	1
ast_id': 1, racter': vannah nnah...	[{'credit_id': '52fe44779251416c91011acb', 'de...	31357	False	NaN	16000000	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	NaN	tt0114885	en	...	1
ast_id': 1, racter': George nks', '...	[{'credit_id': '52fe44959251416c75039ed7', 'de...	11862	False	{'id': 96871, 'name': 'Father of the Bride Col...	0	[{'id': 35, 'name': 'Comedy'}]	NaN	tt0113041	en	...	1

2. *Collaborative Recommendation System*: We obtained the dataset by merging "movies_metadata.csv" and "ratings.csv" on movie ID, and calculated a user's average rating for movies of the same genre.

	avg_romance_rating	avg_comedy_rating
1	3.33	4.25
2	4.00	2.80
3	3.33	3.20
4	3.17	3.00
5	5.00	5.00

- Exploratory Data Analysis for Content-based model

We have carefully identified some abnormal patterns and tried to clean the data for our future exploration. Specifically, for the dataset we prepared for the content-based recommendation

system, we checked and deleted duplications in the data. After we joined these datasets together on movie ID, we found that there are 76 duplicates. Then we further decided that the initial solution for the duplication problem would be to delete them from the joined dataset that we planned to use.

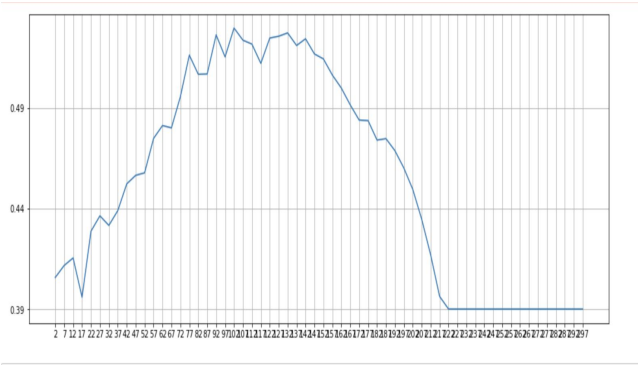
Then, considering that genres of movies were indispensable factors for us to detect similarity and to connect one movie to another set of movies, we further explored how the values of genres in the dataset looked like. However, we found that there were 2442 rows containing no value in the column "genres". As we noticed that even the genre information was deficient in these rows, there still existed useful values in other feature columns. In this way, instead of deleting the incomplete rows where the genre entry was missing, we decided to leave them as empty cells and later combined the column "genres" with the column "keywords" to implement our further exploration on the similarity of movies, which ensured the completeness and accuracy of our study.

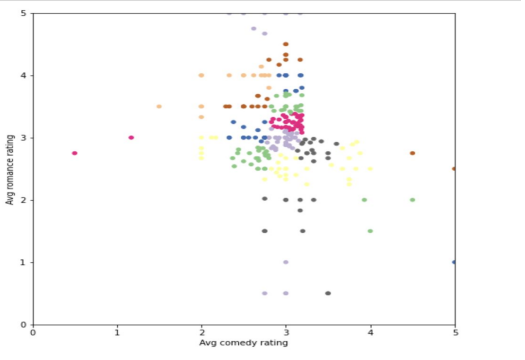
```
[{'id': 18, 'name': 'Drama'}]                                                                    5000
[{'id': 35, 'name': 'Comedy'}]                                                                   3621
[{'id': 99, 'name': 'Documentary'}]                                                              2723
[]                                                                                               2442
[{'id': 18, 'name': 'Drama'}, {'id': 10749, 'name': 'Romance'}]                                  1301
[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'name': 'Drama'}]                                      1135
[{'id': 27, 'name': 'Horror'}]                                                                    974
[{'id': 35, 'name': 'Comedy'}, {'id': 10749, 'name': 'Romance'}]                                  930
[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'name': 'Drama'}, {'id': 10749, 'name': 'Romance'}]     593
[{'id': 18, 'name': 'Drama'}, {'id': 35, 'name': 'Comedy'}]                                       532
Name: genres, dtype: int64
```

(Note: There are 2442 movies with empty values in the column "genres", and they hold the values "[]" in the column "genres". The id of each genre in the column "genres" is uniquely linked to the name of genres, and there are a number of movies having multiple genres at the same time.)

- Exploratory Data Analysis for Collaborative model

Under K-means, we started with the "rating.csv" dataset and calculated each user's average rating for movies of the same genre. We randomly picked two genres (Comedy and Romance) for our initial detailed analysis. Then, we applied K-means technique with small Ks to Comedy and Romance, and clustered the data based on users' average ratings for movies of these two types. We interpreted the plots we got and confirmed that this method is capable of clustering people with similar rating preferences into the same group. Then, we wrote a function to iterate through all possible Ks, plotted the ascending values of k versus the total error calculated using that k, choose the appropriate K, which was 7 in this case, by the elbow method, and then successfully reclustered our data with our new K.





Part 3: Content-based Recommendation system

A content-based recommendation system is recommending the most similar movies that the user has watched to the user. It uses the information about the movies, and the movies' similarity can be measured in various ways.

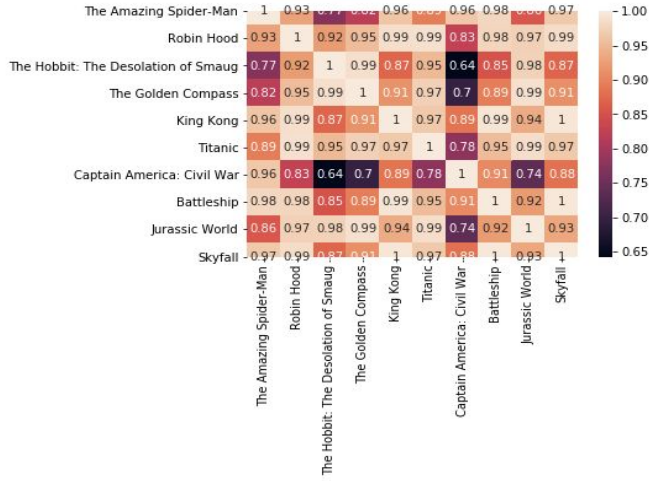
Based on our initial EDA, we observed that we could:

1. Based on the assumption that the description of a movie will correctly summarize the key features in that movie and that users' would prefer watching movies with similar contents or styles, we could apply some pre-trained Natural Language Processing models on movie descriptions to transform them into vectors. We could then apply some similarity measurement algorithms to identify similar movies based on their meanings.
2. Under the hypothesis that genres and user-generated tags could cluster similar movies, we try to use some bag-of-words models on the genres and user-generated tags and see if two movies share the same genre or tag;
3. Use the features popularity and ratings of each movie to recommend relatively popular or higher quality movies;

For similarity measurements, our final decision regarding the model settled down to a Cosine-similarity measurement because it was proved to perform well for text mining.

$$cos(\theta) = \frac{\vec{u}\cdot\vec{v}}{|\vec{u}|\cdot|\vec{v}|}$$

The similarity matrix will eventually look like this attachment, in which each movie has a similarity score respect to every other movie.



Overview-Based Recommender:

1. **Bert:**

After checking out documents for word2vec, glove, and bert algorithm, and trying out demo data, we decided to use a pretrained bert model as it gives slightly better results and it is relatively more suitable for analyzing texts in contexts of movie overview. We first noticed

that, when analyzing the meaning of a paragraph, we should not only focus on each of the words, but also should consider the relationship between words. With multiple levels of Long-Short Term Memory layers, a BERT pre-trained model could easily understand the correlation between words and therefore will produce a better vectorized representation of the input paragraph.

Loaded from the *sentence-transformers* module, a simple pre-trained BERT-uncased model could give its interpretation of the input paragraph based on its weights. After each description is vectorized, we measure the cosine-similarity between movies to obtain the similarity matrix and recommend movies based on the similarity.
CODES:

```
sentence_embeddings = bert.encode(Movies['overview'].tolist())
```

```
similarity = cosine_similarity(sentence_embeddings)
```

OUTPUT: For the movie *The Dark Knight Rises*, the top ten movies it recommends are:

```
Batman Begins
The Dark Knight
Batman Forever
Batman: Mask of the Phantasm
Batman
Batman: Year One
Teenage Mutant Ninja Turtles
Batman Returns
RoboCop
The Punisher
```

2. **tf-idf:**

Another approach is to construct our bag-of-words vectors using term frequency-index document frequency, and store each of the resulting vectors into the dataset. It takes less time than using the pre-trained bert model but still gives a reasonable result. If a user watched a movie, we could measure cosine-similarity between this movie's bag-of-words vector and that of all other movies, and recommend movies with the highest similarity.

Scikit-learn gives a built-in TfIdfVectorizer class that produces the TF-IDF matrix. The dot product of the TF-IDF vectorizer will directly give us the cosine similarity matrix using the linear kernel function.

CODES:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
tfidf = TfidfVectorizer(stop_words='english')
#Replace NaN with an empty string
new['overview'] = new['overview'].fillna('')
tfidf_matrix = tfidf.fit_transform(new['overview'])
# Compute the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

OUTPUT:

```
1 get_recommendations('The Dark Knight Rises')
```

```
12501                                        The Dark Knight
150                                           Batman Forever
1331                                          Batman Returns
15541                           Batman: Under the Red Hood
585                                                  Batman
21235    Batman Unmasked: The Psychology of the Dark Kn...
9244                     Batman Beyond: Return of the Joker
18071                                     Batman: Year One
19827              Batman: The Dark Knight Returns, Part 1
3100                           Batman: Mask of the Phantasm
Name: title, dtype: object
```

Problems Encountered and Solutions

Compared to other methods, we should notice that:

1. Although we do not need to train this model by ourselves, it still takes a long time for this model to calculate the corresponding vector for an input paragraph. Additionally, if we encounter a new movie in our dataset, then we would need to re-construct the entire similarity matrix, which could be extremely time-consuming if the dataset grows larger. To deal with this issue, we decided to create the following algorithm:
    a. Building of the vectorized dataset: We calculated the vectorized representation of each movie by feeding them to the pre-trained model, and stored these information to a new dataset.
    b. Calculation of the Similarity matrix: We selected a small subset of the vectors, calculated pairwise similarity, and stored them into another dataset.
    c. Appendage of new observations: For each of the remaining vectors and any new movies, we calculated their pairwise similarity with all movies in the similarity matrix, and appended the corresponding rows and columns into the dataset.
    d. Recommendation handling: Each time when a user watches a movie, we extract the corresponding row in the similarity matrix, sort this row by similarity, and return the corresponding movies.
2. We should also observe that this model largely relies on our aforementioned two assumptions that movie descriptions could correctly summarize movies and that a user would like to watch movies with similar contents. Thus, this model does not provide us an optimal solution under the following situations:
    a. The user watched a movie, but does not like this movie. In this case, this model cannot detect the user's preference. A better approach would be to cluster movies and randomly recommend movies from other clusters that do not include this movie.
    b. Rather than an inclination to watch movies of the same type, the user would like to watch movies with other similar properties. For instance, it is totally possible that a user only likes movies with a particular actor or actress, which is an inclination that this model cannot capture. Under this situation, a better solution would be a clustering on other properties of movies, and we analyze the user inclination based on the historical data of this user.

Bag-of-words Model on Genre, Keywords, Directors, and Cast:
Besides the overview information of the movies, we also wanted to consider other features like genres, tags, directors, and actors. The assumption here is that a person's preference for certain movies may indicate their preferences to certain directors, actors, and genres. After preprocessing the data, we selected the first three items in features:

```
1 New[['title', 'cast', 'director', 'keywords', 'genres']].head(3)
```

index	title	cast	director	keywords	genres
0	Toy Story	[tomhanks, timallen, donrickles]	johnlasseter	[jealousy, toy, boy]	[animation, comedy, family]
1	Jumanji	[robinwilliams, jonathanhyde, kirstendunst]	joejohnston	[boardgame, disappearance, basedonchildren'sbook]	[adventure, fantasy, family]
2	Grumpier Old Men	[waltermatthau, jacklemmon, ann-margret]	howarddeutch	[fishing, bestfriend, duringcreditsstinger]	[romance, comedy]

After that, we combined all the features into one single string with space between words. When measuring the similarity, we included the director twice since we wanted to assign more weights to this feature. Instead of using TF-IDF, we tried to use CountVectorizer(), which converts text documents to vectors that give information of token counts. Each feature was mapped to an index without facing the risk of hash collision occurring in TF-IDF, which tends to take less weight of the presence of an actor or director if they have appeared in many other movies. The output we received from this model seems to be more varied than the previous output, especially including more movies of the same director.

CODES & OUTPUTS:

```
from sklearn.feature_extraction.text import CountVectorizer
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(New['combine'])


from sklearn.metrics.pairwise import cosine_similarity
cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
```

```
20 get_recommendations('The Dark Knight Rises', cosine_sim2)
```

```
12490    The Dark Knight
10129      Batman Begins
11364       The Prestige
2467           Following
44677            Dunkirk
5259            Insomnia
22880       Interstellar
15489          Inception
25898          Doodlebug
9236              Shiner
Name: title, dtype: object
```

Problems Encountered and Solutions:
The dataset has a lot of information for each movie. It even includes the job title of all the film-making teams, all the actors, and a lot of keywords for a particular movie. Many words do not have any meaning and are not likely to appear in any other movies. Thus, we decided

to set a frequency threshold to filter out some meaningless low-frequency words. We also took account of some specific names like “World War II” by deleting the gap within words, so that it will not be recognized as one word instead of three. In addition, we selected and restricted our interest to the first three items in features. Even though it might be more meaningful to be able to use all the available information, using only the first three items made the data more readable and easier for modeling.

Ratings-Based Recommender:
So far, we had tried some text mining techniques but used very little information about the film’s ratings, which is a very important evaluation of the quality of the movies. The dataset gave us the average ratings for each movie and the total number of people who rated those movies (total votes). To use this information, we first scaled the ratings because sometimes a movie with a higher score may be due to its smaller number of voters.
The weighted rating formula we used is the IMDB ratings formula:

$$WeightedRating(WR) = (\frac{v}{v+m} \cdot R) + (\frac{m}{v+m} \cdot C)$$

v-the number of votes
m-the least amount of votes required to be listed
R-average rating
C-mean vote

After experimenting with the different threshold, we decided to set m to be 15, which means a movie has to have at least 15 votes to be considered (>60th quantile of the entire dataset). The top 15 movies are given in the output. The “score” feature is the scaled average ratings of the movies.

CODES & OUTPUT:

```
# Define a new feature 'score' and calculate its value with `weighted_rating()`
q_movies['score'] = q_movies.apply(weighted_rating, axis=1)
q_movies = q_movies.sort_values('score', ascending=False) #sort based on score
q_movies
q_movies[['title', 'vote_count', 'vote_average', 'score','id']].head(15) #print top 15 movies
```

]:

	title	vote_count	vote_average	score	id
10325	Dilwale Dulhania Le Jayenge	661.0	9.1	9.022733	19404
43389	Planet Earth II	50.0	9.5	8.604120	420714
39153	Planet Earth	176.0	8.8	8.550093	192040
314	The Shawshank Redemption	8358.0	8.5	8.494837	278
835	The Godfather	6024.0	8.5	8.492841	238
40320	Your Name.	1030.0	8.5	8.458629	372058
12501	The Dark Knight	12269.0	8.3	8.296725	155
2848	Fight Club	9678.0	8.3	8.295849	550
292	Pulp Fiction	8670.0	8.3	8.295368	680
522	Schindler's List	4436.0	8.3	8.290961	424
23718	Whiplash	4376.0	8.3	8.290838	244786
5489	Spirited Away	3968.0	8.3	8.289899	129
2215	Life Is Beautiful	3643.0	8.3	8.289002	637
1181	The Godfather: Part II	3418.0	8.3	8.288281	240
1155	One Flew Over the Cuckoo's Nest	3001.0	8.3	8.286660	510

Problems & Notes:
This approach did not take into account the tastes of individual users but gave more of a big picture of the recommendation system. If we do not have any information of users' actions and tastes, this might be a good approach, without driving them away with recommendation of poor-quality movies.

Part 4: Collaborative Recommendation

Our ultimate goal in this part was to cluster people with similar rating preferences into the same group, and then predict a target individual's rating for the movies he has not seen based on the rating history of the users within the same group. While we successfully finished clustering in EDA when we were only dealing with two genres, we found that the same logic can not be applied to the whole data due to the difficulty to visualize. Under careful consideration, we decided to adjust our raw data and merge "rating.csv" with "movies_metadata.csv" to produce a new dataframe which has userID as row names and movie names as column names, and proceed with the new data frame. In terms of major clustering techniques, we stick to K-means because we consider this as a good opportunity to practice and apply what we were introduced to in class.

Due to the massive amount of data, we first sorted out most rated movies and users who rated the most movies to narrow our data of interest, and then visualized it using heatmaps. On the graph, we have user id on the y-axis, and movie names on the x-axis. We denoted numerical rating by different colors of the cells (the blank means the user has not rated the movie yet), with the values for each color available for check on the legend to the right.

To deal with the NaN values in the data frame, we decided to cast it into the sparse csr matrix type before applying K-means. Then, we used K-means to cluster users based on their rating preference, which in other words, leads to a similar taste for movies within each cluster.

After clustering, we moved on to prediction, which is the recommendation part. We decide to proceed our prediction within each group. Our goal is to predict the specific score that a target audience will give to a random movie, and then use this value to replace the NaN values in each clustered data frame. As our logic in K-means implies that users presumably have similar taste within each group, it is reasonable to represent the target audience's rating with the average score calculated from the rest of the users of his cluster who have already watched and rated the movie. In this way, we could successfully replace all NaN values with numerical data. As we obtain predicted ratings for all movies that the target audience has not rated, we sort those movies by their respective predicted rating with descending value. Finally, we will choose the first 20 movies and recommend it to the user.

OUTPUTS:
Attached here is one example where we randomly picked a user, predicted the ratings that he will give for the movies he had not seen, and filed the recommendation list.

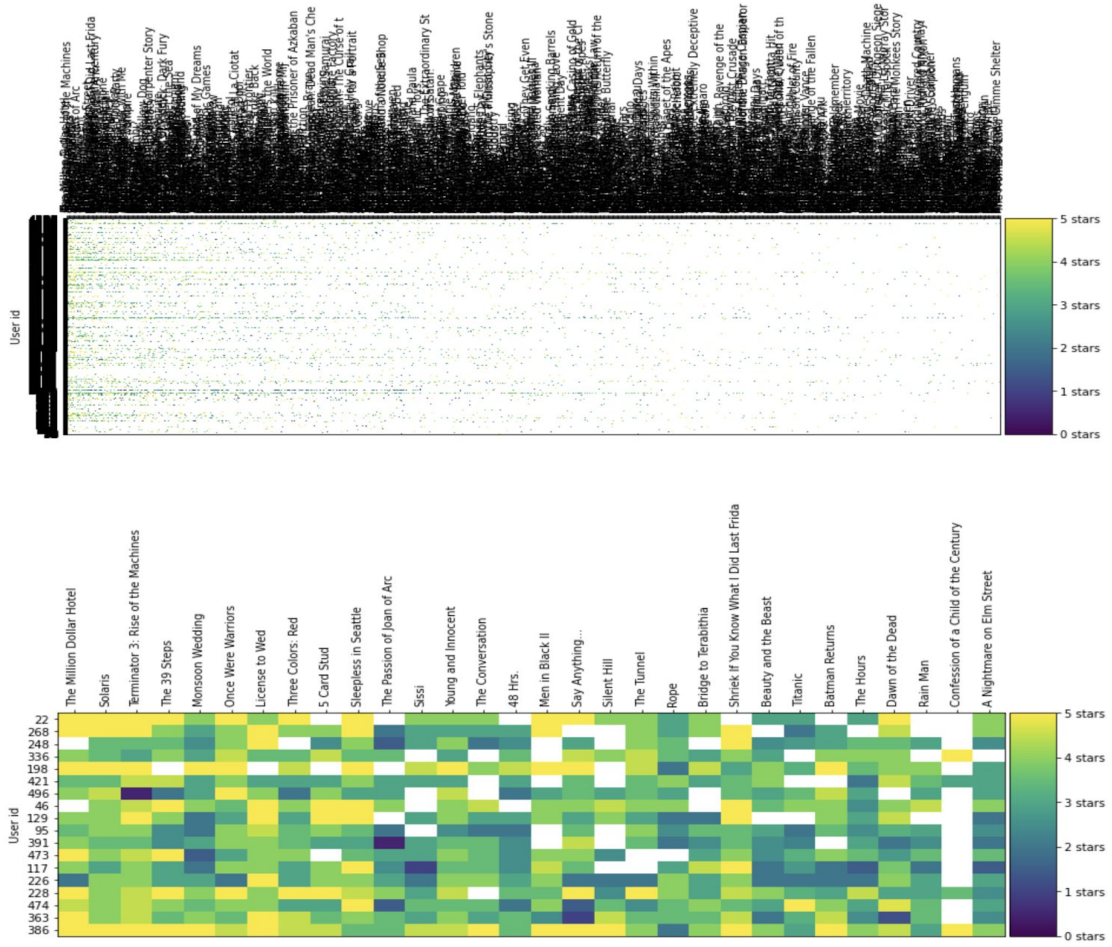
```
# Pick a user ID from the dataset
user_id = 27
# Get all this user's ratings
user_2_ratings  = cluster.loc[user_id, :]
# Which movies did they not rate?
user_2_unrated_movies =  user_2_ratings[user_2_ratings.isnull()]
# What are the ratings of these movies the user did not rate?
avg_ratings = pd.concat([user_2_unrated_movies, cluster.mean()], axis=1, join='inner').loc[:,0]
# Let's sort by rating so the highest rated movies are presented first
avg_ratings.sort_values(ascending=False)[:20]
```

```
Human Nature              4.428571
Deep Blue Sea             4.333333
Ghost Rider               4.307692
Monsieur Batignole        4.285714
What's New Pussycat?      4.166667
Stranger Than Fiction     4.125000
Judgment Night            4.109375
Stitches                  4.100000
The Sixth Sense           4.045455
Secret Agent              4.035714
Hour of the Gun           3.964286
The Dreamers              3.961538
Princesses                3.944444
Star 80                   3.928571
Don Q Son of Zorro        3.916667
Leon: The Professional    3.900000
The Killing               3.875000
Felidae                   3.863636
Ken Park                  3.861111
Stand by Me               3.857143
Name: 0, dtype: float64
```

In summary, we used the ratings information of each individual user to build a collaborative recommendation system. That is, to find the similar users of the target user and recommend movies prefered by the similar users to the target user.

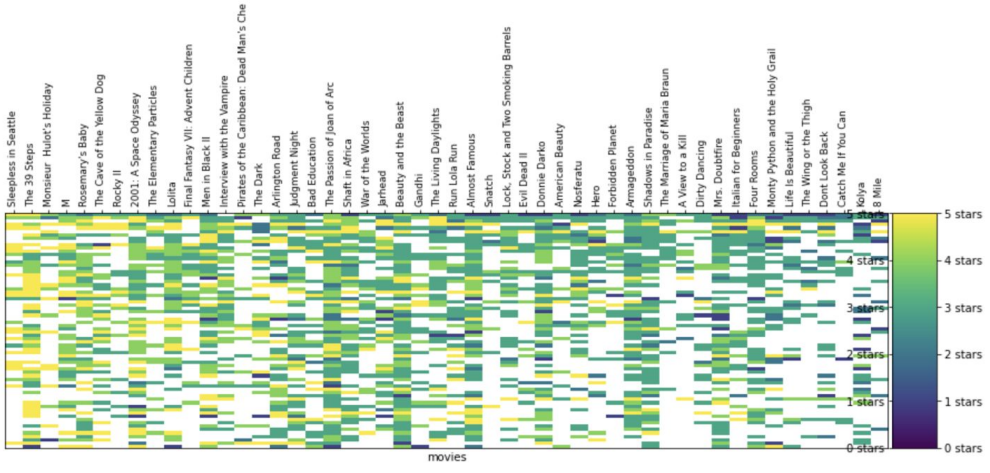
Problems encountered and solutions:

1. When we were applying K-means technique and logic to more genres, while the logic still worked fine, we met the issue of visualizing the high-dimensional data. Under thorough research, we found heatmap as an alternative visualization tool.
2. After creating the data frame with user ID in the rows and movie names on the column, we discovered that it was sparse with a lot of Nah values since not every user rated every movie. We also discovered that K-means did not deal with sparse datasets well. To deal with this, we decided to cast the data frame into the sparse csr matrix type defined in the SciPi library. We first convert it to SparseDataFrame, then use pandas' to_coo() method to finish the conversion to the csr matrix.
3. We tried to visualize the whole dataset (the data frame with user ID in the rows and movie names on the column) using a heatmap, but our computer can not deal with data with such large size. Firstly, we tried to narrow down our data of interest to the first 5000 rows of this dataset. Then we sorted both the movies and users by rating times, and tried to visualize the first 1000 rows. As we could see, the visualization was successful, but not satisfying. So we narrowed down again, to only visualize 30 most rated movies and 18 users who rated the most movies.



4. When applying K-means, we still met the issue that our dataset was too large. So we restricted our data to the first 5000 users, filtered out 1000 most rated movies among them, and proceeded with this sub-dataframe. When visualizing the clusters, we restricted the number of users to 70, and the number of movies to 50; or else the heatmap would be too dense. Attached here is one the graph of one of our clusters.

```
cluster # 0
# of users in cluster: 220. # of users in plot: 70
```



5. When we choose reasonable values of K, it would be sound to iterate through range from 1 to the length of userId plus one, so that we could cluster all observations from one group to as many groups as the number of observations. However, it will be too computationally-heavy for our computer to finish the task. The solution we had is to try arbitrary Ks, compare resulting errors and set our final K (K=15).
6. In our model, we decide if the target audience has watched a certain movie or not by whether he has rated that movie or not. But there are chances when that audience actually watched the movie and left without rating. Our model did not cover that probability. Therefore, there are chances that our recommendation contains the movies he likes, and also the movies he has already watched. To solve this, we think it is necessary for us to find new raw datasets that keep track of not only the ratings but also the watching history.

CODES: All codes in separate files.

Part 5: Summary (take away)

Our team has created recommendation systems using both a content-based model and collaborative filtering model. The content-based system took account of movie information and ratings. Given a movie, our system would recommend the most similar and highly rated movies to it. In cases when there is no movie introduction information available, we will instead recommend based on ratings.

For collaborative filtering approach, we mainly focused on k-means clustering. We used k-means to cluster users based on their rating preferences, and recommend to a target audience based on the preferences of the people in the same cluster.

In a short semester, we approached our topic in two ways with multiple models. On the one hand, all of our team members were introduced to, and then deeply researched on these two prevalent systems. We had the chance to understand in detail about the rationale behind, and built ourselves a solid foundation for further study in the future, no matter in which direction. On the other hand, during the process of trying to construct these systems, we polished our coding skills, we experienced the difficulty to transform theory to applicable model, learnt additional topics that were not covered in class (such as cosine similarity), and deepened our understanding of topics that were covered in class (such as K-means).

Part 6: Drawbacks and suggestions for future improvement

One of the most serious drawbacks of our recommendation system is that it is hard to explain or evaluate our results. Without practical trials, we are unable to validate the effectiveness of our recommendations nor can we have any idea of users' real preferences. When evaluating the similarity between movies, there could be many other measurements and modeling techniques. Each could give distinct recommendations. At this point, it might make more sense to consider multiple dimensions of the data and build a hybrid recommendation system which takes input of userID, and the movie he/she most

recently watched. The output would find the similar movies with highest popularity and the larger possibility for this user to rate the movie at a high score. If we have more time, we will work on combining the different techniques we tried to build such an overall hybrid recommendation system.

Responsibilities:

Haoyu Zhang: Explored content-based recommendation system, specifically the rating-based and bag of words model using genres, directors, cast, and keywords. Wrote Part 3, Part5, and Part6 of the report.

Chuanrui Liu: Integrated the logics of the problem explanation, project goals, and project set up; implemented exploratory data analysis on the datasets; identified initial problems and solutions; researched the prior work related to recommendation system; wrote Part 1 and Part 2 of the report

Yunzhou Liu: Construction and Analysis of Overview-based Recommendation System using pre-trained BERT and Similarity Matrix

Dingyuan Liu: Explored K-means based recommendation system, specifically how to choose K, run predictions for all users, and construct a way of casting sparse matrices.

Yiyao Qu:

Researched and found tutorials on how to use K-means clustering; helped Yumeng Zhang with some coding issue; wrote part 4 and 5 of report

Yumeng Zhang:

Focused on the coding part of collaborative recommendation systems, including K-means clustering, sorting, heatmap visualization, and prediction.

Bibliography

Rounak Banik. “The Movies Dataset.”
https://www.kaggle.com/rounakbanik/the-movies-dataset

Vanshika Dhamija. 2019. *PySpark: CountVectorizer|HashingTF.*
https://towardsdatascience.com/countvectorizer-hashingtf-e66f169e2d4e
Ansh Bordia. 2020. *Building a Recommendation Engine in Python.*
https://medium.com/analytics-vidhya/building-a-movie-recommendation-engine-in-python-53fb47547ace

Code Heroku. “Building a Movie Recommendation Engine | Machine Learning Projects.” *YouTube* video, 1:37:07. February 2, 2019.
https://www.youtube.com/watch?v=XoTwndOgXBM&t=1195s

Syed Muhammad Asad. 2020. *AI Movies Recommendation System Based on K-Means Clustering Algorithm.*
https://asdkazmi.medium.com/ai-movies-recommendation-system-with-clustering-based-k-means-algorithm-f04467e02fcd

Victor Roman. 2019. *Unsupervised Classification Project: Building a Movie Recommender with Clustering Analysis and K-means.*
https://towardsdatascience.com/unsupervised-classification-project-building-a-movie-recommender-with-clustering-analysis-and-4bab0738efe

BERT Model:
Devlin, Jacob; Chang, Ming-Wei; Lee, Kenton; Toutanova, Kristina. 2018. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* arXiv:1810.04805v2 [cs.CL].

Pre-trained BERT Basic Model:
Zhu, Yukun; Kiros, Ryan; Zemel, Rich; Salakhutdinov, Ruslan; Urtasun, Raquel; Torralba, Antonio; Fidler, Sanja. 2015. *Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books*, 19–27. arXiv:1506.06724 [cs.CV].

TF-IDF in Recommendation System:
Breitinger, Corinna; Gipp, Bela; Langer, Stefan. 2015. “Research-paper recommender systems: a literature survey.” *International Journal on Digital Libraries*. 17 (4): 305–338. doi:10.1007/s00799-015-0156-0. ISSN 1432-5012. S2CID 207035184.

Theory of TF-IDF:
Rajaraman, A.; Ullman, J.D. 2011. "Data Mining". *Mining of Massive Datasets*, 1–17. doi:10.1017/CBO9781139058452.002. ISBN 978-1-139-05845-2.