STOR 565 Final Project Report

Movie Rating Classification and Recommendation

Group 13: Manhui Zhu, Chengze Xie

I. Project Overview

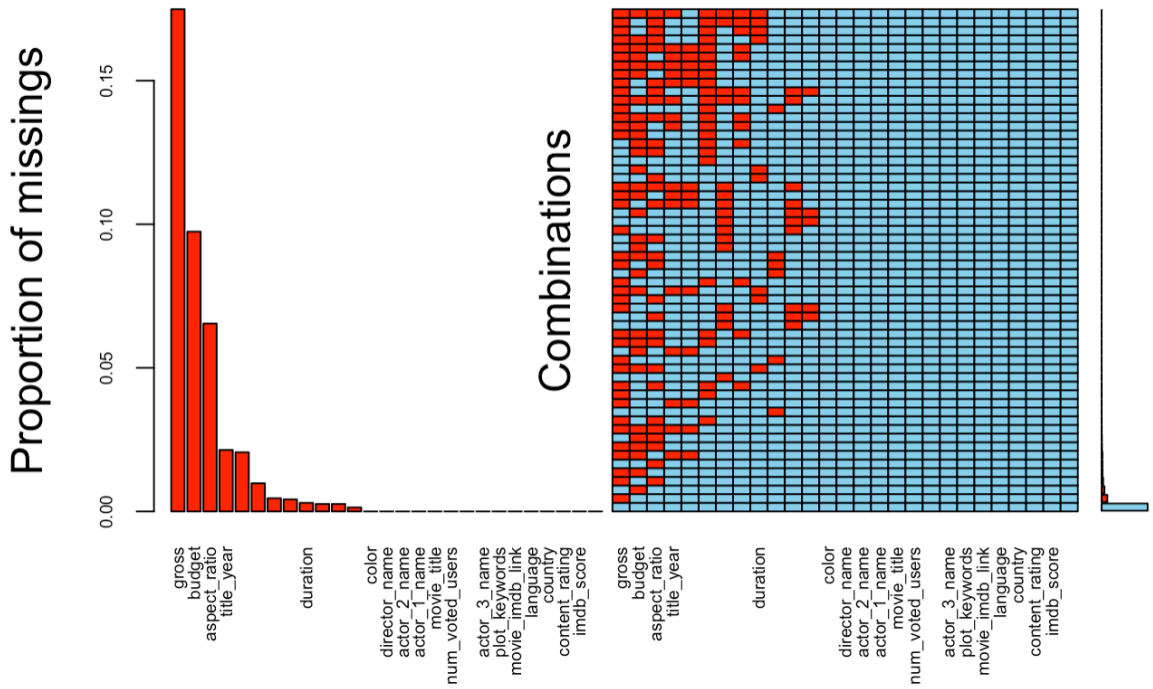
In this project, we do the movie rating classification and recommendation. For classification, we use *The IMDB Movie Dataset* from Kaggle, which has 28 variables and 5043 observations. Before we apply the model to do the classification, we first do data cleaning work to evaluate whether some categorical variables are significant to the variable *`imdb_score`*, and we also dispose missing values so that the model can give better results. The goal for the classification part is not to find the exact score for each movie, the dataset has the variables *`imdb_score`* that is the exact rating on IMDB website. Instead, we just want to know how good or how bad a movie is. So we create 4 classes: "bad", "fair", "good", and "excellent" according to the *`imdb_score`* of each movie. Then we used classification trees (unpruned and pruned), bagging, random forest, and KNN to classify all movies.

For recommendation, we implement two recommendation algorithms, content based and collaborative filtering. Then we assemble these two models to come up with our final recommendation system. For the first algorithm, we use the same dataset as we used before, while for the second algorithm, we add another dataset, *The Movie dataset*, which consists of 26,000,000 ratings and 750,000 tag applications applied to 45,000 movies by 270,000 users. It also includes tag genome data with 12 million relevance scores across 1,100 tags. Readers can check our code on GitHub: https://github.com/Chengze1234/565-final-project.

II. Movie Rating Classification

1) Data cleaning

Before constructing models to do the classification, we first need to carefully evaluate some categorical variables and dispose the missing values in the dataset. We first delete 45 duplicated rows and remove the white space and special character (Â) in the variable *`movie_titile`*. To check which variable has missing value and how many missing values they have, we plot a heat map to visualize the missing values. The plot shows that *`gross`* and *`budget`* are variables that have first and second most missing values in the dataset. But we want to use these two predictors in the following analysis, so we have to delete rows with NULL values for *`gross`* and *`budget`*.



After disposing the *`gross`* and *`budget`*, the variable that has the most missing value is *`aspect_ratio`*, which is also a categorical variable in the dataset. For each unique value of *`aspect_ratio`*, the values that appear the most are 1.85 and 2.35. Therefore, for analysis convenience, we group other values together. But first we need to replace "NA" in this variable with 0, then we calculate the mean *`imdb_sore`* for the current 3 different groups of *`aspect_ratio`*. We find that all means fall in the range of 6.3 ~ 6.7, which means that *`aspect_ratio`* doesn't relate much to the *`imdb_score`*, so we remove the *`aspect_ratio`* variable.

We notice that there are some "0" values which should also be regarded as missing values except for the predictor *`facenumber_in_poster`*. First we replace "NA" with column average for variable *`facenumber_in_poster`*, then replace "0" in other predictors with "NA`", and lastly replace all "NA" with their respective column mean for other predictors.

Now, let us check other **categorical variables**:

- *`content_rating`*: There are many different values in this variable. But according to the new ratings system, there are only four categories: *G* (general audiences), *M* (mature audiences, changed in 1969 to *PG*, parental guidance suggested), *R* (restricted, no children under 17 allowed without parents or adult guardians), and *X* (no one under 17 admitted). Therefore, We replace the values "M" and "GP" with "PG", and replace "X" with "NC-17". We also replace "Approved", "Not Rated", "Passed", "Unrated" with the most common rating "R". Now, we only have 5 different values in this variable: "G", "NC-17", "PG", "PG-13", and "R".

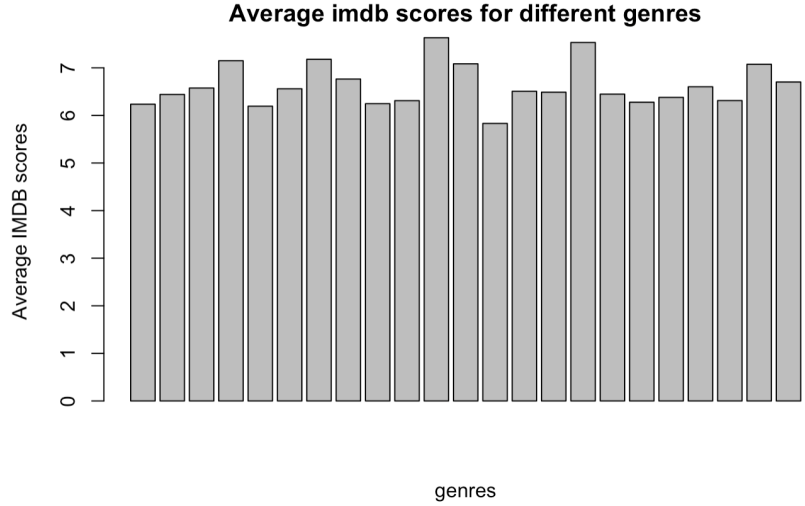
- *`color`*: We find more than 96% of movies are colored, which means this predictor is like a constant and not influential. So we remove *`color`*.
- *`language`*: We find that over 95% of movies are in English, which means this predictor is like a constant and not influential. So we remove *`language`*.
- *`country`*: We find that about 79% of movies are from the USA, 8% from the UK, and 13% from other countries, so we group movies from other countries together. Now, we only have 3 different values in this variable: "USA", "UK", and "Others".
- *`title_year`*: We find that only 2.5% of movies in the current dataset are released before 1980, since there aren't many records of movies released before 1980. It's better to remove those records because they might not be representative.
- *`director_names`*, *`actor_1_names`*, and *`actor_2_names`*: Since all names are so different, there is no meaning to use names to predict score, we remove these variables. SO we remove these three variables.
- *`plot_keyword`* and *`movie_title`*: They are too diverse to be meaningful to the movie rating. So we remove them.
- *`link`*: This variable is meaningless and redundant to the movie rating. So we remove it.

2) Exploratory Data Analysis

- **Whether genres are important to movie rating?**

Intuitively, we think that genres are important factor to the rating, to find the relationship between *`genres`* and *`imdb_score`*, we first divide the string for the *`genres`* variables of each observation (because each movie have various genres), and save each substring along with its corresponding *`imdb_score`*. Then we create a new data frame of genres, separate different types into new columns, and set factors for each type. We calculate the mean *`imdb_score`* for each type and plot a graph of mean *`imdb_score`* versus *`genres`*. Here is a view of the new genres data frame and the plot.

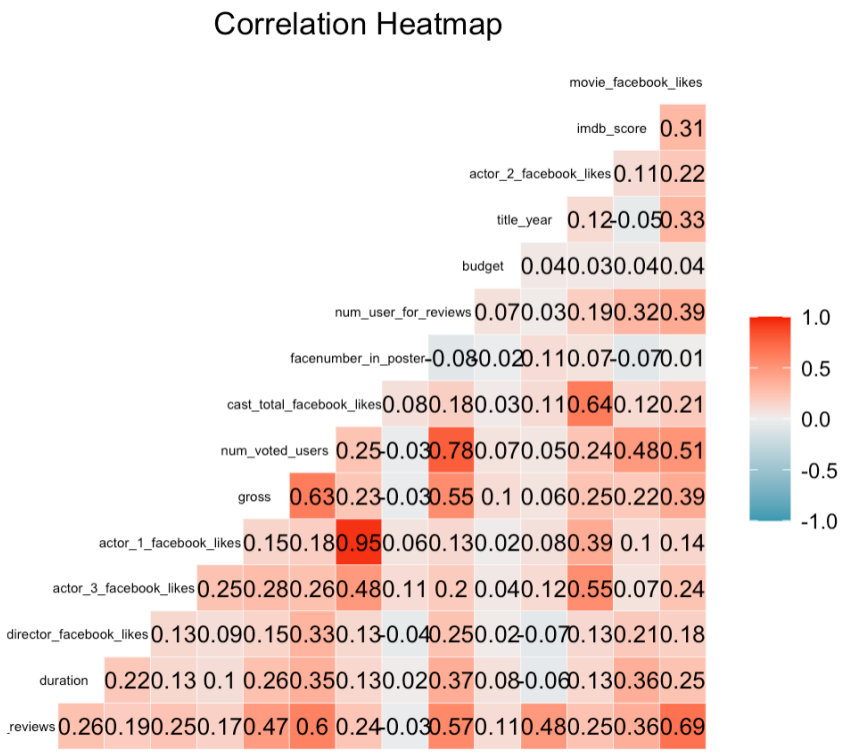
genres	imdb_score	Action	Adventure	Animation	Biography	Comedy	Crime
Action|Adventure|Fantasy|Sci-Fi	7.9	1	1	0	0	0	0
Action|Adventure|Fantasy	7.1	1	1	0	0	0	0
Action|Adventure|Thriller	6.8	1	1	0	0	0	0
Action|Thriller	8.5	1	0	0	0	0	0



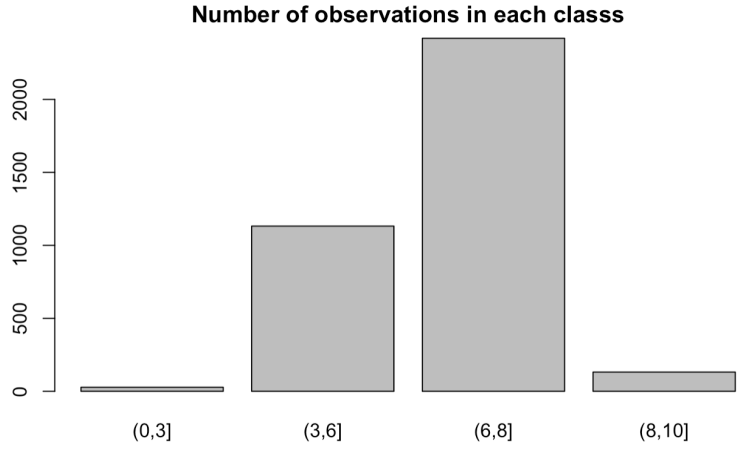
From this plot, we can see that almost all averages are in the range from 6-8. It seems that *`genres`* doesn't have much effect on *`imdb_score`*. Therefore, we remove the predictor *`genres`*.

- **Are our variables correlated?**

we look through the relationship between the current existing variables. Here is a heatmap of correlation. We can see some high correlations (greater than 0.7) between some predictors. According to the highest correlation value 0.95, we find *`actor_1_facebook_likes`* is highly correlated with the *`cast_total_facebook_likes`*, and both *`actor_2_facebook_likes`* and *`actor_3_facebook_likes`* are also somehow correlated to the *`coast_total_facebook_likes`*. So we modify them into two variables: *`actor_1_facebook_likes`* (unchanged) and *`other_actors_facebook_likes`* (combined by actor_1 and actor_2 face likes). There are also high correlations among *`num_voted_users`*, *`num_user_for_reviews`* and *`num_critic_for_reviews`*. We keep *`num_voted_users`* (unchanged) and take the ratio of *`num_critic_for_reviews`* to *`num_user_for_reviews`*.



- **Is our dataset balanced or unbalanced?**



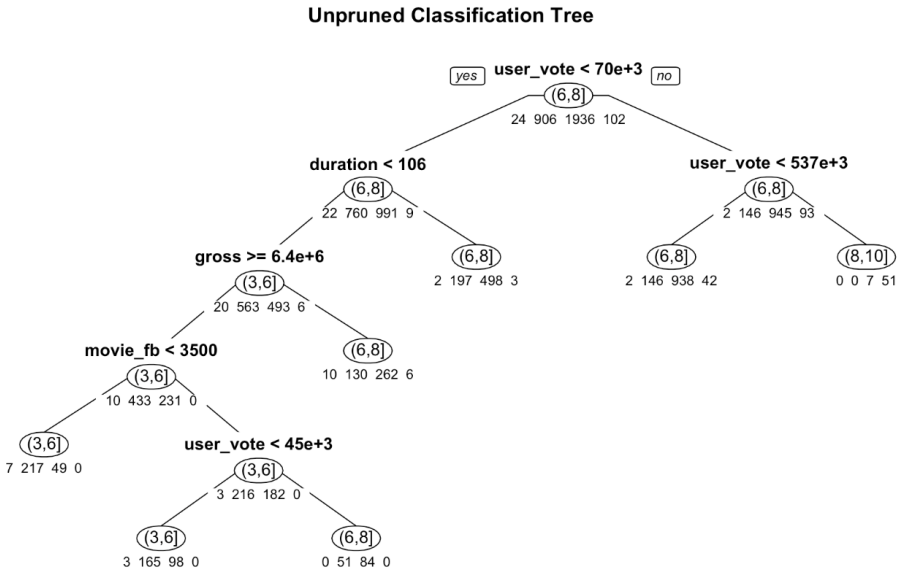
Let's see the distribution of all observations in 4 classes. From the below plot, it is obvious that the dataset is unbiased. Almost 95% of the movies are fair and good movies, less than 5% of movies are bad and excellent movies. This definitely impacts the test accuracy in the following classification models.

3) Classification Models

In our project, we split our dataset into a training set and a testing set with the ratio of 8:2. We use four different methods: classification tree (unpruned and pruned), bagging, random forest, and K-nearest neighbors.

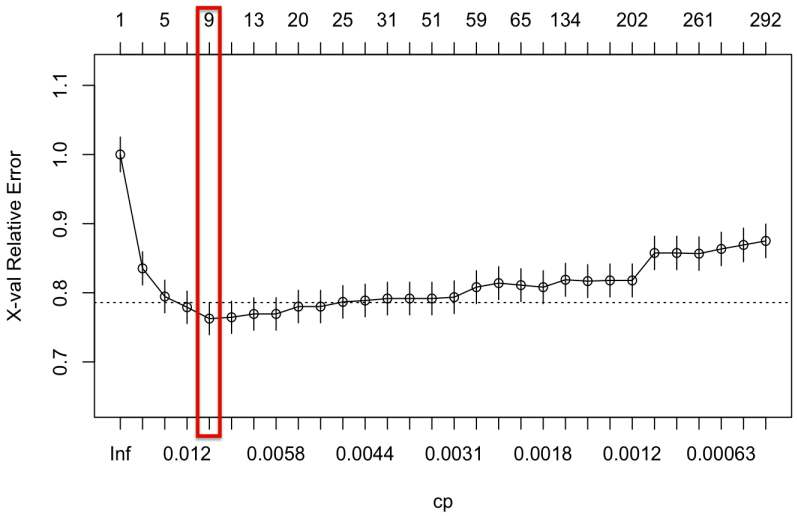
- **Classification Tree (unpruned):**

From the tree plot, we can conclude that movies with more votes on the IMDB website tend to have a higher score, which really makes sense because popular movies will have more attention which attracts more people to vote for them. If a movie has few votes, it can still be a good movie if the duration is longer. But it is kind of surprising that movies making less profit are good, which reflects that movies that have commercial success don't mean they also have critical acclaim. The test accuracy for unpruned classification tree is 71.2%.

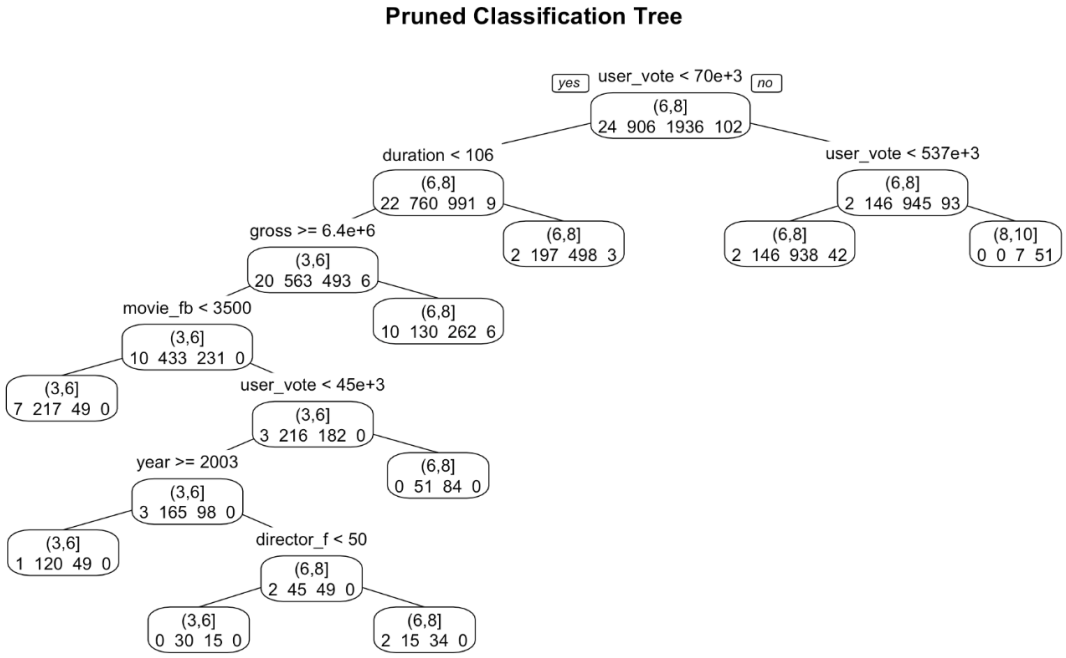


- **Classification Tree (pruned)**:

By using the cross validation, we find that when the number of splits is 9, the cross-validation error is the smallest, which is close to 0.76260. Then we apply this parameter to build a new classification tree, the test accuracy is also 71.2%, which is the same as the unpruned classification tree. It is reasonable if we see the two tree plots, the split points of them are similar,

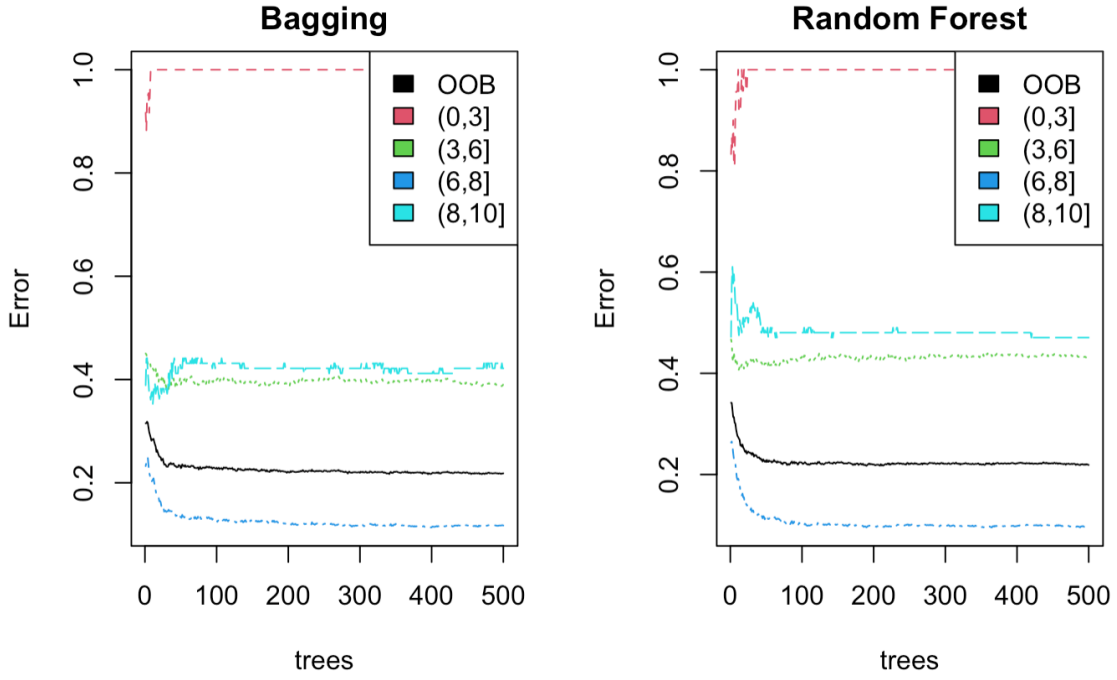


which make a small difference to the results.



- **Bagging**

From the bagging model, the black line in the figure below means that the overall error rate falls below 30%. The red, green, blue and aqua lines show the error rate for bad, fair, good and excellent movies respectively. By using bagging, we classified good movies more accurately, but we cannot classify bad movies very well. The test accuracy is 78.3%, which is better than the simple classification tree model. The bagging model shows that the three most important variables for the movie rating are *`num_user_vote`*, *`duration`*, and *`gross`*.

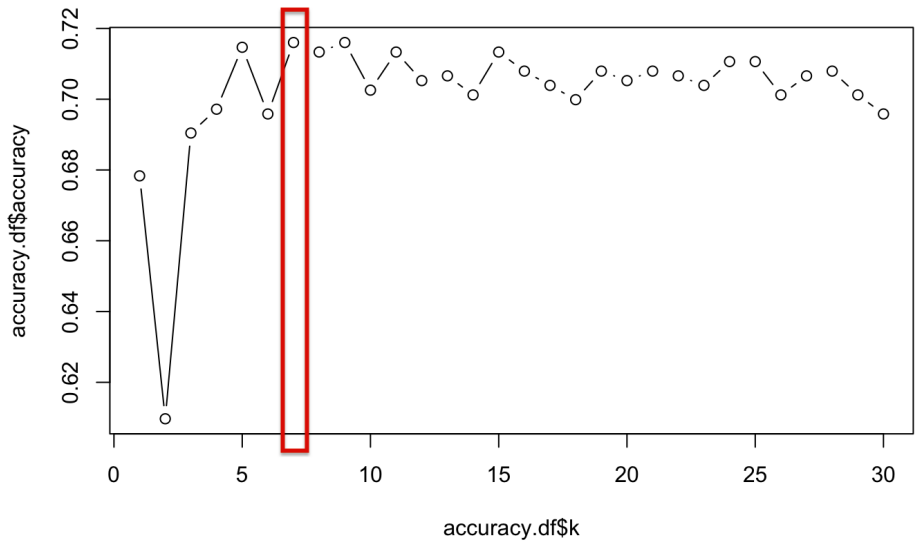


- **Random Forest**

The plot above shows that results of random forest are similar to the results of bagging, but the error rate of classifying excellent movies and fair movies seems a little bit higher than that in the bagging model. The test accuracy is 77.7%, which is a slightly lower than bagging model. The first three important variables given by random forest model are *`num_user_vote`*, *`duration`*, and *`gross`*, which are the same as the bagging model.

- **K-nearest Neighbors (KNN)**

To use the KNN model, we first create dummy variables for all categorical variables, then make all variables in the dataset into the numerical type, and we also normalize our data. To find the best parameter "k" in the model, we set k from 1 to 30 and build 30 KNN models with different values of "k". From the plot below, we can find that when "k" is 7, the test accuracy reaches the maximum, which is 71.6%.



III. Movie Recommendation

Many movie rating websites will give users a general list of movies that have the highest ratings. Here, because our dataset has the IMDB ratings, we can simply sort pandas dataframe to pick all movies by a descending order. Also, we can select different types of genres and sort them in the same way as we did before such that we can give users a recommendation of the best movies of each type of movie. Here is the formula of IMDB Weighted Rating.

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

- 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

However, if we want to give users recommendations based on movies they are watching, such a simple method may fail to give a good recommendation, because it only depends on one variable to give recommendation.

Our model, Content Based Recommender, is to give users some recommended movies based on what they are watching. This recommender totally depends on movies' information so that we do not need users' information(*What content-based filtering is & why you should use it | upwork*). To do that, we first select some categorical variables, which are usually treated as "contents" to have a dataframe. Here, we choose *`movie_title`*, *`genres`*, *`director_name`*, *`plot_keywords`*, *`content_rating`*, *`language`*, *`country`*. We vectorize them for further analysis by using a built-in TF-IDF Vectorizer of Sklearn library. TF stands for "Term Frequency" and IDF stands for "Inverse Document Frequency". TF-IDF is widely used in NPL problems(Author: Fatih Karabiber Ph.D. in Computer Engineering et al.). The calculation and idea is also straightforward. TF measures the frequency of the appearance of words in the document, and IDF is the inverse of the document frequency which measures the informativeness of the term. We generally compute a score for each word to signify its importance in the document and corpus. Finally, by taking a multiplicative value of TF and IDF, we get the TF-IDF score. Here is the mathematical formula to calculate TF-IDF. In the IDF formula, T is the total number of terms, and N is the number of documents with term T in it.

$$TF\text{–}IDF = TF \times IDF$$
$$IDF = log(\frac{T}{N})$$

Because we want to compare our model in different versions,we add some Soups, Bag of Words in different combinations of several different variables. For example, for Soup 1, we combine *`movie_title`*, *`genres`*, and *`director_name`*. The actual idea behind the algorithm we used is to calculate the numerical similarity between two movies by Cosine Similarity. Therefore, we use the vectorized matrix we got before to calculate the cosine similarity matrices. Here is an example.

```
[[1.         0.09168364 0.07171201 ... 0.         0.         0.        ]
 [0.09168364 1.         0.05388071 ... 0.         0.         0.        ]
 [0.07171201 0.05388071 1.         ... 0.02169959 0.         0.        ]
 ...
 [0.         0.         0.02169959 ... 1.         0.0097975  0.        ]
 [0.         0.         0.         ... 0.0097975  1.         0.        ]
 [0.         0.         0.         ... 0.         0.         1.        ]]
```

Finally, we get all of our similarity matrices and we can randomly choose one of them and see their performance. Here, we sort this matrix and get the scores of the 10 most similar movies. For example, when we want to give users a recommendation list, our model will give output that contains 10 movies. However, some recommendations only consider the similarity to the movies the user watched, which means they don't consider the quality of movies themselves. To solve this problem, we set up a benchmark to filter those movies that have very bad quality (based on their *`imdb_score`*). Here is an example of the recommendation list, the figure on the left is the recommendation before we sort out the bad quality movies, and the figure on the right is the version after the modification.

```
get_recommendations('Spectre', cosine_sim3)
```

```
3493                        Skyfall
291                      True Lies
457              Road to Perdition
365     Die Hard with a Vengeance
1114           Revolutionary Road
286                 Casino Royale
2944                Casino Royale
2606              American Beauty
32                     Iron Man 3
Name: movie_title, dtype: object
```

```
get_recommendations('The Dark Knight Rises', cosine_si
```

```
1066                                  Insomnia
97                                   Inception
3716                                  Memento
1233                              The Prestige
120                              Batman Begins
96                                Interstellar
4017    Batman: The Dark Knight Returns, Part 2
3509                        Dark Angel
1816                                Dark City
Name: movie_title, dtype: object
```

From the previous method, we know that anyone using our engine for recommendations based on a specific movie will receive the same recommendations for that movie, regardless of who he or she is. To give customized recommendations for each user, we use Collaborative Filtering to capture the taste of each individual. Collaborative Filtering is to measure similarity of a user's behavior to predict how much one would like a particular product or service (Luo, 2019). To achieve this goal, we use the SVD method which can be found in the Surprise library. Then we combine this with the previous content-based recommender, and our final model outputs a recommending list based on both content and users' information. Here is an example ("1" in our commend is the user's id, and "Avatar" is the name of the movie).

```
hybrid(1, 'Avatar')
```

	title	vote_count	vote_average	year	id	est
1011	The Terminator	4208.0	7.4	1984	218	3.083605
522	Terminator 2: Judgment Day	4274.0	7.7	1991	280	2.947712
8658	X-Men: Days of Future Past	6155.0	7.5	2014	127585	2.935140
1621	Darby O'Gill and the Little People	35.0	6.7	1959	18887	2.899612
974	Aliens	3282.0	7.7	1986	679	2.869033
8401	Star Trek Into Darkness	4479.0	7.4	2013	54138	2.806536
2014	Fantastic Planet	140.0	7.6	1973	16306	2.789457
922	The Abyss	822.0	7.1	1989	2756	2.774770
4966	Hercules in New York	63.0	3.7	1969	5227	2.703766
4017	Hawk the Slayer	13.0	4.5	1980	25628	2.680591

IV. Conclusion and Future Work

For the movie rating classification part, the bagging model performs the best with the test accuracy being 78.3%. For the specificity and sensitivity of four classes for these models, one anomaly is that the specificity of bad movies for all models is 1, which means that all models can identify which movies are not bad movies with 100% accuracy. But intuitively it is hard to reach 1 for specificity in classification. We think that our result is because no bad movies are split into the testing set, and it is also confirmed by looking at the confusion matrix for all models. It also indicates the problem for the unbalanced data. In our future work, we hope to solve this problem by oversampling the bad movies and excellent movies or undersampling the good movies and fair movies.

For the movie recommendation part, we use the cosine similarity matrix to measure and rank the similarity between movies. And we set a benchmark to filter movies that are too bad to be acceptable for most viewers. Furthermore, we tried to implement a collaborative filtering method to utilize users' information to give recommendations. In the future, we can use our model on other datasets to check the performance or we can try some state of the art methods on the dataset we find to make some experiments. Plus, we may consider having a user interface to boost viewers' experience of using our engine, and we can design a questionnaire and collect feedbacks from users to test whether we give them a precise recommendation.

V. Reference

Author: Fatih Karabiber Ph.D. in Computer Engineering, Fatih Karabiber Ph.D. in Computer Engineering, Psychometrician, E. R. L., & Editor: Brendan Martin Founder of LearnDataSci. (n.d.). *TF-idf - term frequency-inverse document frequency*. Learn Data Science - Tutorials, Books, Courses, and More. Retrieved November 30, 2022, from https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/

Cosine similarity. Cosine Similarity - an overview | ScienceDirect Topics. (n.d.). Retrieved November 30, 2022, from https://www.sciencedirect.com/topics/computer-science/cosine-similarity

What content-based filtering is & why you should use it | upwork. (n.d.). Retrieved November 30, 2022, from https://www.upwork.com/resources/what-is-content-based-filtering

Luo, S. (2019, February 6). *Intro to Recommender System: Collaborative filtering*. Medium. Retrieved November 30, 2022, from https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26

Banik, R. (2017, November 10). *The Movies Dataset*. Kaggle. Retrieved November 30, 2022, from https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset

M, K. (2018, February 23). *Movie IMBD Dataset*. Kaggle. Retrieved November 30, 2022, from https://www.kaggle.com/datasets/kevalm/movie-imbd-dataset