Gideon Blinick

Capstone Project 2 Milestone Report 1: Movie Recommender/Clusterer

Problem statement

This project seeks to solve a subproblem of the recommendation problem. The domain is movies and the recommender should work based not on collaborative filtering but on content-based filtering. More specifically, this project seeks to solve the specific problem of recommending the most similar movie to a given, provided movie.

Different measures of similarity will be tried to determine what 'most similar' means, and ultimately one will be chosen that works best.

The recommender will find the most similar movie based on a variety of features, most notably plot text information (plot synopses, keywords, outlines) and additional non-text information about the movies including numerical data (year of release, runtime, ratings, and number of votes) and categorical data (genres).

Client and Use

Our client is anyone who enjoys watching movies and wants to watch movies they enjoy. The ideal use for this engine is for a client to think of a movie they have enjoyed in the past and to enter that movie in our engine. The engine will return the movie most similar to their given movie. If the client hasn't seen the movie, they now have a movie to watch that they will hopefully enjoy. If they have seen the movie, the engine will also be able to output the next most similar movies after the top movie, and the user can browse that list until they find a movie that they haven't seen which they can then enjoy.

The goal is to be able to provide movie-watchers with a convenient and fast way of finding content they love.

Dataset Description

Dataset creation notebook available here:
https://github.com/gblinick/Movie-Recommender-Clusterer/blob/master/Dataset%20Creation.ipynb

Our dataset was custom created from the IMDb website using 2 different sources. The first source was IMDb's public dataset's repository: https://www.imdb.com/interfaces/

This repository contains seven different tsv files of thousands of movie and TV episode information. All 7 files were downloaded and unzipped into the project's data folder and each was individually examined for useful information. Since the goal of this project is to use plot text data as the primary means by which to recommend movies, it was disappointing to find that none of those files contained text data. Still, 3 of the public tables were used, including the title_basics, title_crew, and title_ratings tables.

The title_basics table contained the most useful information we wanted including movie ID, movie name, release year, runtime, and genre. The title_ratings table was useful for its information about movie ratings and the number of votes each movie had received, both features we wanted to include in our dataset. The title_crew table was initially included for the information it had about directors and writers. In theory, adding these features to our dataset would improve the accuracy of our model. However, after some consideration, it was decided remove these features out because they would dramatically increase the dimensionality of our feature space and would likely lead to poorer recommendation performance. That is, since there are many, many directors and writers out there, we would have needed many, many columns in our dataset because one-hot-encoding is the appropriate way for handling such categorical data.

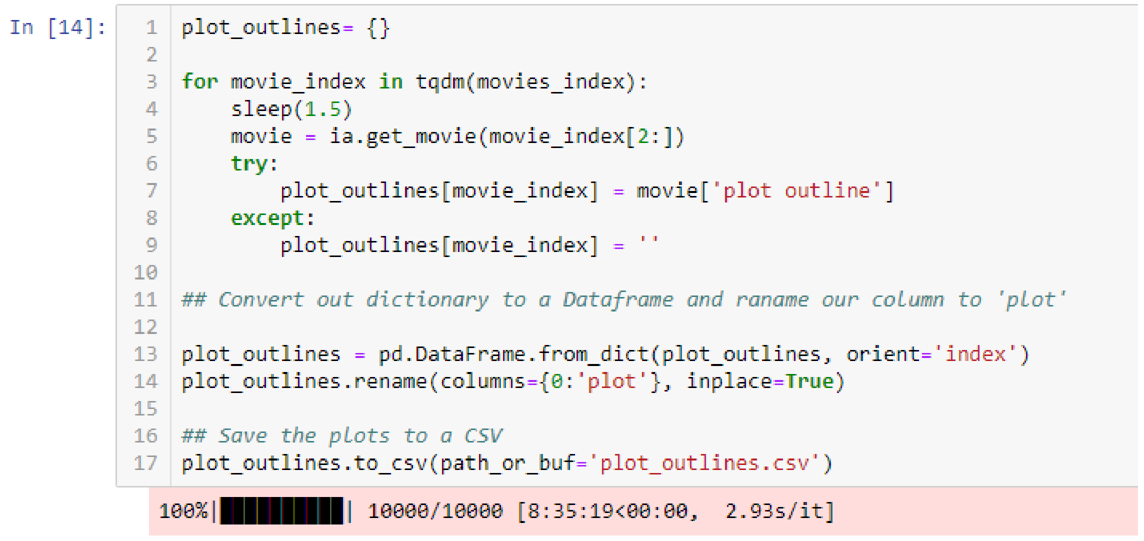
The other tables were of no use to us because they contained information about people (name.basics, title.principals) , TV episodes (title.episode), and titles (title_akas), all features we don't need or want in our dataset.

To get the plot information we needed, we used the wonderful IMDbPY API: https://imdbpy.sourceforge.io/

IMDbPY made pulling plot data from IMDb very easy. All that we needed to do was call the API 10,000 times to get whatever plot information we wanted. We called it 10,000 times because we wanted our dataset to contain a sufficient number of movies with which to recommend other movies, and 10,000 was a nice, round number for this. Furthermore, to make our engine more relevant we only called the API for the 10,000 most popular movies, where popularity was determined by the number of votes a movie received. This information was available from the title_ratings table.

We called the API a few times because there were multiple types of plot information that we wanted to retrieve: basic plot descriptions (usually a sentence or two), plot outlines (usually a paragraph), plot keywords (a list of words describing the movie), and plot synopses (the longest plot descriptions that describe the movie scene by scene).

After making the API calls, the data was stored in CSV filed so the API didn't need to be used again. Below is the code used for retrieving the plot outlines information and saving to a CSV. The same code works, with minor adjustments, for obtaining the other plot information.



```
In [14]:
plot_outlines= {}

for movie_index in tqdm(movies_index):
    sleep(1.5)
    movie = ia.get_movie(movie_index[2:])
    try:
        plot_outlines[movie_index] = movie['plot outline']
    except:
        plot_outlines[movie_index] = ''

## Convert out dictionary to a Dataframe and raname our column to 'plot'

plot_outlines = pd.DataFrame.from_dict(plot_outlines, orient='index')
plot_outlines.rename(columns={0:'plot'}, inplace=True)

## Save the plots to a CSV
plot_outlines.to_csv(path_or_buf='plot_outlines.csv')
```

```
100%|██████████| 10000/10000 [8:35:19<00:00,  2.93s/it]
```

Once we had finished all the API calls, we were able to join all our information together into one Dataframe using the pandas' join function. This was possible since each of the constituent dataframes had the unique movie ID ('tconst') in them and we set that column as the index in order to join them.

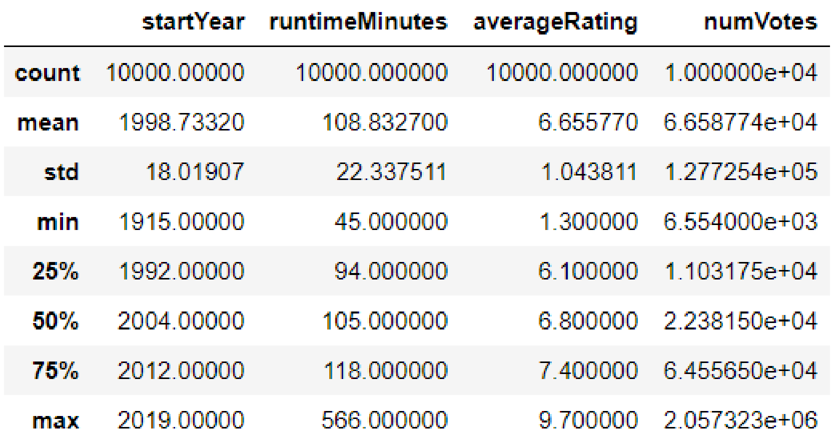
Finally, we removed unnecessary columns including titleType (we were only interested in movies so we only retrieved movies and thus this column provided no information), originalTitle (we didn't care if it was the same or different from its current title), isAdult (none of the movies in our set of 10,000 most popular movies was ‘Adult’), and endYear (the values for this column ended in ‘\N’ for all movies, so it was unhelpful).

Exploratory Data Analysis (EDA)

EDA notebook available here:
https://github.com/gblinick/Movie-Recommender-Clusterer/blob/master/EDA.ipynb

In our EDA, we examined the numeric features (startYear, runtimeMinutes, averageRating, and numVotes) and categorical feature (genres). We did not examine the text features (plot, plot outline, keywords, and synopsis) because this requires some NLP techniques which are part of the modelling process.

To begin, we examined the numeric features, first individually, then in pairs. We use the pandas’ describe to get some summary statistics of the features. The results are below:



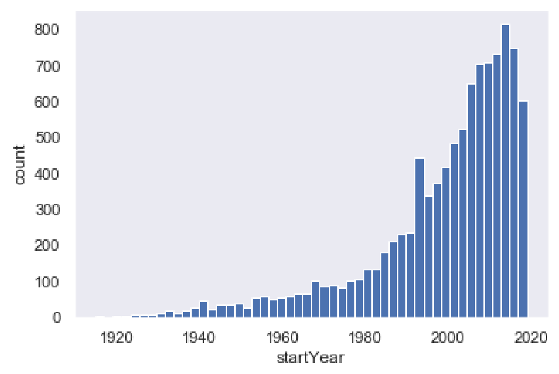
	startYear	runtimeMinutes	averageRating	numVotes
count	10000.00000	10000.000000	10000.000000	1.000000e+04
mean	1998.73320	108.832700	6.655770	6.658774e+04
std	18.01907	22.337511	1.043811	1.277254e+05
min	1915.00000	45.000000	1.300000	6.554000e+03
25%	1992.00000	94.000000	6.100000	1.103175e+04
50%	2004.00000	105.000000	6.800000	2.238150e+04
75%	2012.00000	118.000000	7.400000	6.455650e+04
max	2019.00000	566.000000	9.700000	2.057323e+06

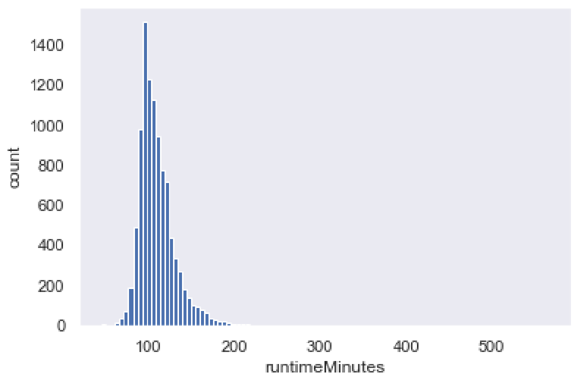
Looking at the table, we made a couple of quick points. First, most movies in our dataset are from 2000 and on. This is seen from the fact that the median year is 2004 in the table. This result makes sense when considering that more movies are released each year and more recent movies are more likely to be voted on given that more people today watch movies than ever before (and in general this has increased with time).

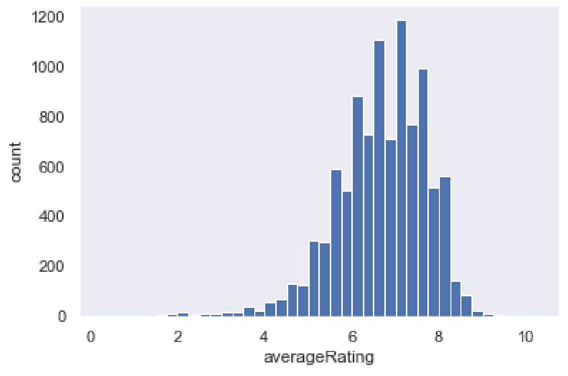
Second, the average rating is about 6.7 stars and is close to the median of 6.8 stars.

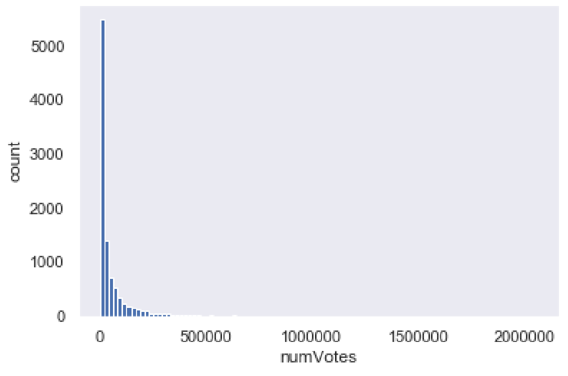
Third, we see that the mean number of votes for a movie is about 66.5 thousand. The standard deviation is almost double that though indicating most movies are below the mean, and a few are **way** above it. This can also be seen from the fact that the 75th percentile is smaller than the mean.

Next, we plot the distribution of each feature to get a more intuitive feel for them.



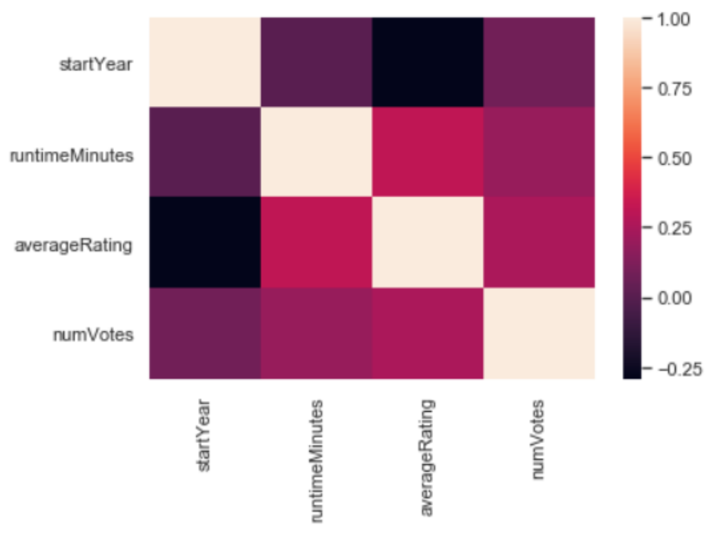






This histograms of the distributions of our numerical features confirm our earlier observations about those features. We move on to examining correlations between the features.

The following plot displays a heatmap of our variables:

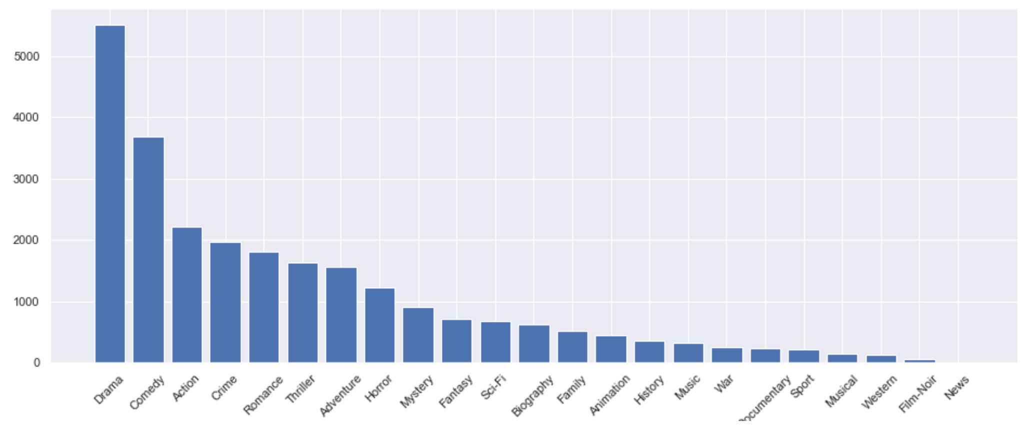


The corresponding correlations are:

```
Feature 1      Feature 2
runtimeMinutes averageRating      0.314285
averageRating  numVotes           0.254650
runtimeMinutes numVotes           0.196031
startYear      numVotes           0.083236
               runtimeMinutes     0.007503
               averageRating     -0.292108
```

We see that longer movies tend to get better ratings, better rated movies are voted on more, and that more recent movies are likely to get lower ratings, among other things. These trends are explored in greater depth in the above linked notebook where we create scatter plots of the features and plot the line of best fit through each scatter plot.

We conclude by analyzing the genres column. This column contains genre tags or labels for a movie. Each movie can be assigned 1 to 3 genre tags. After preprocessing the column by one-hot-encoding the genres, we get the following breakdown of genres applied to movies:



So we see that Drama is the most common genre, then Comedy, then Action, and so on. It's important to note that the above chart counts the number of times each label was applied, and since most movies had more than one label applied, the number of genre labels will exceed the number of movies (10,000). Indeed, 6189 movies in our dataset had 3 labels applied, 2765 had 2 applied, and only 1046 were 1-label movies.

Finally, we looked at which genres are most correlated and anti-correlated with each other. That is, which genres are most likely to appear together, and which are most likely to not appear together.

Gideon Blinick

For positive correlation, we obtained the following table:

		Pearson Correlation Coefficient
Genre 1	**Genre 2**	
Adventure	**Animation**	0.318
Action	**Adventure**	0.274
Biography	**History**	0.217

We see that Adventure and Animation, Action and Adventure, and Biography and History are genres that are fairly correlated with each other. These facts are unsurprising. We are using a benchmark of 0.2 to define what is significantly correlated.

On the negative correlation side, we obtained the following:

Genre 1	**Genre 2**	
Drama	**Comedy**	-0.245
	Action	-0.246
	Adventure	-0.259
	Horror	-0.229
Comedy	**Thriller**	-0.297
Action	**Romance**	-0.208

We see that Drama is negatively correlated with a bunch of other genres (Comedy, Action, Adventure and Horror). It seems like the Drama genre likes to 'hog' the movies it is tagged to, that is, it doesn't like to share tags with other movies. We also see that Comedy and Thriller and Action and Romance are negatively correlated, which is unsurprising since those genres can be described as opposites.

This completes our exploration of the data. These trends and a few other are explored in greater detail in the Jupyter notebook linked to above.