Luis Ruiz

Recommendation System

Final Report:

Recommendation System

Problem Statement

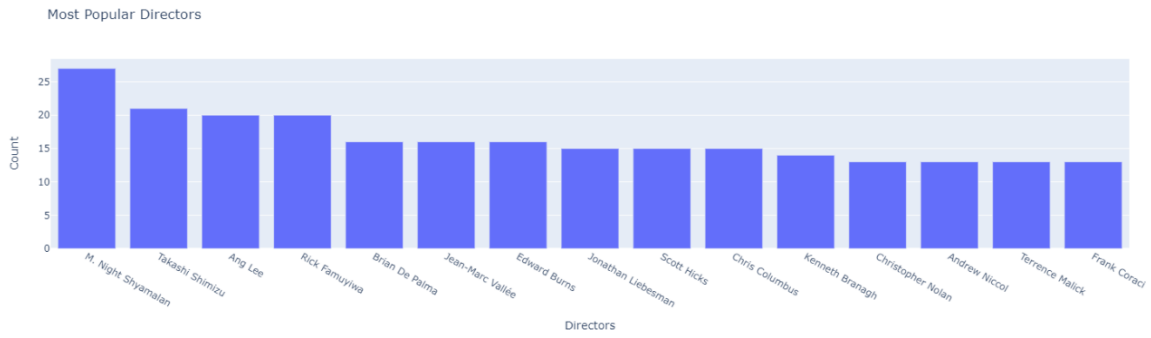
How can machine learning be used to create a movie recommendation system that will return a film based on previous viewed films? The film will be selected using a scale that measures compatibility using a 'match' scale. Using this scale will decrease the average time it takes to select a film.

There are hundreds of movies released on an annual basis, some great and some not so great. Being able to narrow down on what movies to watch based on personal preference is a strategy many companies are using in today's age. For example, on Amazon they'll recommend similar items to buy based on what you recently bought or what was the last item you viewed. Imagine a personalized movie recommendation system that returns what movie to watch based on previous movies. My goal is to create a recommendation system that makes selecting a film much easier by returning a match factor that ranges from 0-100. 0 being you will not like it and 100 meaning you will love it. Our system aims to return at most 10 movies, so the user does not suffer from choice overload. Now I will only focus on films and maybe once I create an accurate enough system I will try to venture into the anime or tv shows. Some constraints we might face are dealing with movies that fall under several genres. Trying to combine collaborative filtering and popularity-based selection. To verify if our model is accurate, I will feed it movies I like and don't like based on the recommendation I will identify if it is accurate. I will also ask other people to verify my findings. The dataset I will use contains 3 different csv lists that contain the movies, ratings, and keywords.

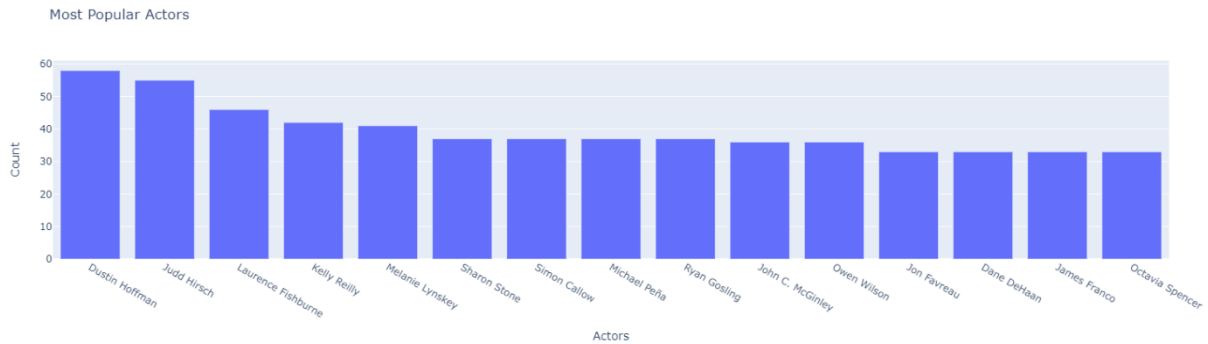
Data Wrangling & Exploratory Data Analysis

My initial dataset contained about one million films with 15 columns, but dataset was eliminated due to the lack of computing power I had available. I ended up selecting two csv files, one

named movies_collab.csv which contained the movies titles along with other relevant features like genres, keywords, overview. This csv filled contained 4809 rows and 23 columns. The other csv file, collab_rating.csv, contained a movieId which was used as a unique identifier for each movie that was later used to merge both of our datasets. Other important columns it contained were ratings, userId, number of ratings just to name a few. The movie dataset contained a lot of unnecessary features so using both lasso regression and my domain knowledge to eliminate some of the features. This left us with 9 columns. There were four columns that were converted from a string literal to a string. The columns were genres, crew, keywords, and cast. The crew column contained everyone involved in the production of the movie, so the backend of filming production like directors, writers etc. The director was used for our model since my I only cared for the director.



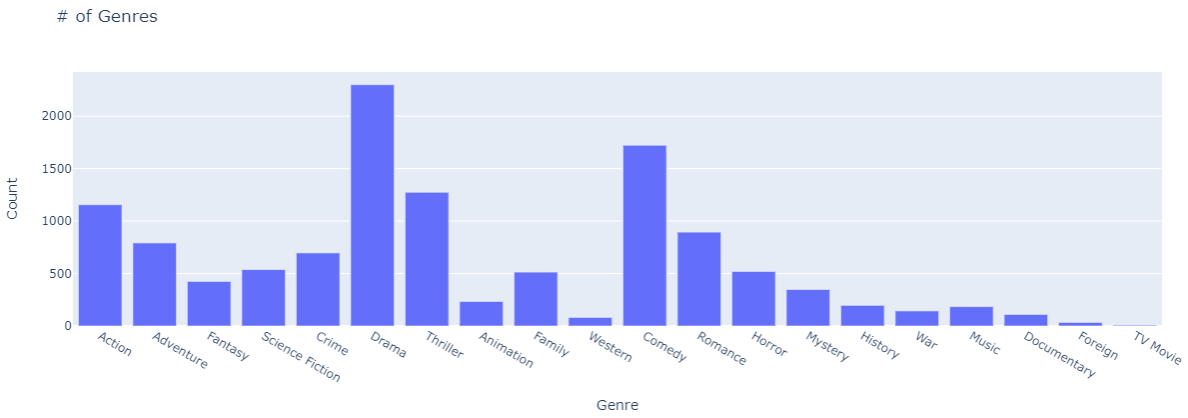
The cast column contained the front-end of the film crew so actors and extras. I extracted the first 10 actors to cut down the complexity. I graphed the most popular actors in our dataset to see how many of



them I knew.

The genre column was a dictionary of string literals which made it hard to work with but after converting the column to a string I was able to extract the columns.

Visualizing the genre column using a bar-plot, it was clear that there wasn't an even number of movies per genre. This could be because movies can be classified with multiple genres as well as the popularity of certain genres vs others. I did not perform oversampling or under sampling because our dataset is a good representation of films.



Feature Engineering & Modeling

Demographic Filtering

I created three different types of recommendation system for our models. The first time of system is a **Demographic Filtering** system if you are familiar with Netflix, we can think of this system to identify films that are "Now Trending". My approach was to us IMDB's rating formula that

$WR = (v/v + m) * R = (m/v + m) * C$ where, v is the number of votes for each film, m is the minimum number of votes required, R is the average rating of the movie and C is the mean vote across

our report. Using IMDB's formula I was able to create a score column that returned the most popular movies based on rating. The most popular movies in our dataset are:

	title	vote_count	vote_average	score
1887	The Shawshank Redemption	8205	8.5	8.058779
662	Fight Club	9413	8.3	7.939080
65	The Dark Knight	12002	8.2	7.919910
3237	Pulp Fiction	8428	8.3	7.904531
96	Inception	13752	8.1	7.863227
3342	The Godfather	5893	8.4	7.851248
95	Interstellar	10867	8.1	7.809563
809	Forrest Gump	7927	8.2	7.803313
329	The Lord of the Rings: The Return of the King	8064	8.1	7.727547
1996	The Empire Strikes Back	5879	8.2	7.698363

Out of the top movies our model returned I have seen 9 of them. I can say that our model is a good representation of the best rated movies. This type of system however suffers from the lack of user input but is good if we want to watch a popular movie.

Content Filtering System

My second approach to solving my movie selection issue was to use previous watched movies to predict what other movies I would like. Comparing this system using Netflix as an example think about it as 'Because you watch this film you will also like...'. This system requires us to use CountVectorizer and Cosine Similarities algorithms that are included in the Sklearn library. Our system first tokenizes each occurrence of a words using CountVectorizer using the following columns, 'cast', 'keywords', 'crew', and 'genres'. After tokenizing our metadata, we convert it to a multi-dimensional space where each dimension corresponds to a word a film. In simple terms our objective is to quantitatively estimate the

similarity between films. Using this I input one of my favorite films, Interstellar based on that it returned the following movies:

```
get_recommendations('Interstellar', cosine_sim2)

✓ 0.7s

65                              The Dark Knight
95                                 Interstellar
96                                    Inception
119                             Batman Begins
1036                                   Insomnia
1199                            The Prestige
3578                                    Memento
0                                        Avatar
1       Pirates of the Caribbean: At World's End
2                                       Spectre
Name: title, dtype: object
```

Out of the 10 movies our model returned I have seen 7 out of the 10 movies. Giving it a 70-accuracy using myself as a testing example. This recommendation system is better than our demographic system at using user input as well as the film's features to then return similar movies. Drawbacks that we encounter with this type of system is that it can only make recommendations based on the existing interest of the user and the model hence only has limited ability to expand on the users' existing interest. Overall, a content-based system requires more data to create a better system.

Collaborative Filtering

The last system that I created was a collaborative filtering system, we want to think about this type of system as one that returns movies based on what users with similar taste watch. Our approach to building our engine starts like our content-based system but we utilized the columns ratings, usedID and ItemID which is the movie. Our data then gets converted into a matrix. We give our data a list of

movies with the rating. Then using different algorithms our system recommends movies based on a calculated similarity score. For our model we used three different algorithms. The first is Non-negative matrix factorization (NMF). To understand how our algorithms works we can briefly touch on the math behind it. We have one matrix that contains the userId as rows and ItemID as columns and within our matrix each value ranges between 0-1 hence non-negative matrix factorization. We then decompose our matrix into two smaller matrices. The first matrix has the same number of rows as the main matrix and each row belongs to each person, while the second matrix has a column for each item. Each of the two matrices expresses the power of association between a user or an item with latent factors or hidden features. Since not every user is going to rate each item, the user-item interaction matrix will be sparse. Therefore, the goal of matrix factorization method is to predict missing entries. This algorithm has its drawbacks, since the number of latent factors can vary as the latent factors increase, more hidden factors have been extracted from the data and therefore the quality of recommendation system improves. But, at some points too much increase of the number of latent factors causes over-fitting over the data and thereby reducing recommendation quality.

The movies that I gave the system were there following:

	userId	title	rating
0	696969	Remember the Titans	3.0
1	696969	Transformers	3.0
2	696969	Happy Gilmore	4.0
3	696969	Interstellar	5.0
4	696969	The Lion King	3.5
5	696969	Gravity	3.0
6	696969	Titanic	5.0
7	696969	The Godfather	5.0

Using the input from above our NMF engine recommended the following films:

NMF

```
algo = NMF()
algo.fit(data.build_full_trainset())
my_recs = []
for iid in movies_to_predict:
    my_recs.append((iid, algo.predict(uid=696969, iid=iid).est))

pd.DataFrame(my_recs, columns=['Movies', 'predictions']).sort_values(
    'predictions', ascending=False).head(10)
```

✓ 10m 55.3s MagicPython

	iid	predictions
0	"Great Performances" Cats (1998)	3.531935
5196	Pat and Mike (1952)	3.531935
5164	Paprika (Papurika) (2006)	3.531935
5163	Papillon (1973)	3.531935
5162	Paperman (2012)	3.531935
5161	Paper, The (1994)	3.531935
5160	Paper Towns (2015)	3.531935
5159	Paper Moon (1973)	3.531935
5158	Paper Clips (2004)	3.531935
5157	Paper Chase, The (1973)	3.531935

The other algorithm us was Single Variable Decomposition, it was believed that Netflix uses a version of SVD in their system. SVD works like our matrix factorization algorithm, but the biggest difference is that it decomposes the main matrix into 3 instead of two. The math behind it is beyond my understanding

but the idea is similar. This is what our SVD engine recommended:

SVD

```
algo = SVD()
algo.fit(data.build_full_trainset())

my_recs = []
for iid in movies_to_predict:
    my_recs.append((iid, algo.predict(uid=696969, iid=iid).est))

pd.DataFrame(my_recs, columns=['Movies', 'predictions']).sort_values(
    'predictions', ascending=False).head(10)
```

9m 12.9s MagicPython

	Movies	predictions
5340	Planet Earth II (2016)	4.529168
5339	Planet Earth (2006)	4.469794
6784	The Godfather Trilogy: 1972-1990 (1992)	4.399866
6100	Shawshank Redemption, The (1994)	4.391491
878	Black Mirror: White Christmas (2014)	4.383577
623	Band of Brothers (2001)	4.377269
4884	Night, The (Notte, La) (1960)	4.361050
7571	Winter Light (Nattvardsgästerna) (1963)	4.356637
1795	Decalogue, The (Dekalog) (1989)	4.326188
6744	The Blue Planet (2001)	4.317353

The final algorithm used was SVD++. The difference is that SVD++, uses users rating an item is in itself an indication of preference. In order words, the '++' means incorporating implicit feedback. Our SVD++

engine returned the following movies:

	Movies	predictions
5340	Planet Earth II (2016)	4.490610
5339	Planet Earth (2006)	4.405024
878	Black Mirror: White Christmas (2014)	4.340749
6784	The Godfather Trilogy: 1972-1990 (1992)	4.328347
623	Band of Brothers (2001)	4.322573
6100	Shawshank Redemption, The (1994)	4.317663
5060	Open Hearts (Elsker dig for evigt) (2002)	4.270364
3307	I Am a Fugitive from a Chain Gang (1932)	4.256419
6883	Thin Man Goes Home, The (1945)	4.244912
107	7 Plus Seven (1970)	4.242237

Evaluation

Using my mentor's idea to evaluate our model due to our computing limitations he suggested we perform cross validation as a method of evaluating. Each model was cross validated and split into 3 folds due to the limitations. I am glad I decided to model smaller dataset because when performing the evaluation, it took approximately four hours and 30 minutes to complete.

Model Evaluation

```
cv = []
# Iterate over all recommender system algorithms
for recsys in [NMF(), SVD(), SVDpp()]:
    # Perform cross validation
    tmp = cross_validate(recsys, data, measures=['RMSE'], cv=3, verbose=False)
    cv.append((str(recsys).split(' ')[0].split('.')[-1], tmp['test_rmse'].mean()))

pd.DataFrame(cv, columns=['Recommendation Sys', 'RMSE'])
```

✓ 252m 41.9s

	Recommendation Sys	RMSE
0	NMF	0.885323
1	SVD	0.832026
2	SVDpp	0.824335

The idea behind our method of evaluation was limited so I had to choose between Mean Absolute Error and Root Mean Squared Error. I picked RMSE because I wanted to penalize the system when outliers or bad predictions were present. As you can see from the table above the best performing model was our SVD++ model with a RMSE score of about .82, the lower the value the better. The worst performing model was our NMF that had a RMSE of about .89. Respectfully the best and worst performing model were also the longest and shortest to complete.

Conclusion & Recommendations

Each model has its tradeoffs, knowing and understanding them will lead to creating a more accurate model for our desire choice. Let's first discuss Collaborative filtering (CF), and some of its strengths: It was able to identify cross-genres niches and we saw this by the type of movies it recommended. Although not shown in our report, it was better at adapting which improves quality over time. Its weakness was that it requires a large dataset. Suffers from what is called new user cold start and new item cold start. What this means is that if our user has a unique preference there might not be similar matches of other users. Content based (CN) is great when our data is limited, we saw this when our input was a single movie. Like CF it also adapts to the feedback that leads to an improvement of quality, but CN requires a large dataset. Demographic filtering can identify cross genres niches, but it doesn't take user preference into consideration. It is sole based on other users' ratings. My recommendation would be to create a hybrid system to uses CF and CN. We see this type of system being executed in a more complex way by Netflix. If you've ever used Netflix before it has a row where it recommends movies because 'you watched' as well as a row for movies because 'other users watched'. This is a good example of how a hybrid system can use user input, similar users and movie features.