Problem Statement

Movies taste is largely subjective. Thus, making a recommendation is an interesting challenge. One person's taste is not necessarily shared by everyone. For instance, one person may like superhero movies, but dislike movies focus on drama and dialog while another might like horror but not family films.

From a commercial perspective, entertainment is a hit driven industry. This means that a successful movie drives majority of the revenue. The return is high stake. To improve the effectiveness of the return on investment, such as paying actors, hiring the production crew, marketing and etc, I am attempting a movie recommendation system.

The movie recommendation system at the fundamental level is to recommend similar movies that a person rated highly. If successful, the algorithm could:

1. increase audience engagement for lesser known films
2. improve effectiveness on marketing

I will take the data hosted on Kaggle. The data set is the Movie Lens dataset (https://www.kaggle.com/rounakbanik/the-movies-dataset).

Data Wrangling

I downloaded the 5 datasets from the page. They are Movie Metadata, Credits, keywords, links, and ratings. Aside from some missing values, the data is clean in terms of errors or typos. The first step is to work with some of the data structure in some columns.

Some of the columns are originally in a Json format but converted to csv. Hence reading some of the columns will take the values as strings. For example, in the credits dataset, the cast column has all the cast in one row for each movie. I may want to only use the top N actors from the cast to be part of my movie recommendation engine.

To extract all the elements, say, cast for each movie and make it into a Python List, I applied the following custom function.

```
# Returns the list of names
def get_list(x):
    if isinstance(x, list):
        names = [i['name'] for i in x]
        return names
    #Return empty list in case of missing/malformed data
```

```
return []
```

The Movie Metadata dataset contains information on 45,000 movies featured in the Full MovieLens dataset. Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies.

The Credits dataset contains the cast and crew information.

The keywords dataset contains the movie id and the keywords relevant to the plot.

The ratings dataset contains each user's ratings that on the movies he or she has rated.

The link dataset is mapping the IMBD movie id and the id we are using.

I start with the Movie Metadata and transform the JSON stringfield column to a list for columns genre, production country. I also dropped 3 rows where genre is n/a. Due to memory limit, I took only a portion of the entire movie metadata.

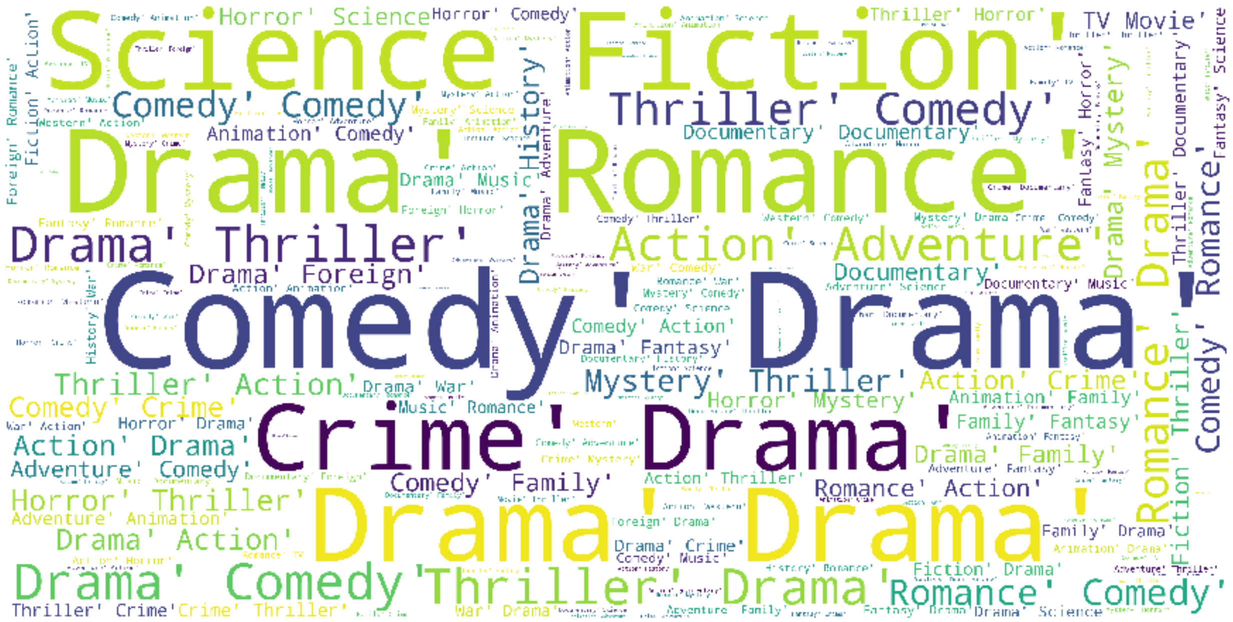
Similarly, for the credit dataset, I extracted the top 5 actors and directors for each movie. After that, I joined the two processed datasets together on id.

Data Analysis

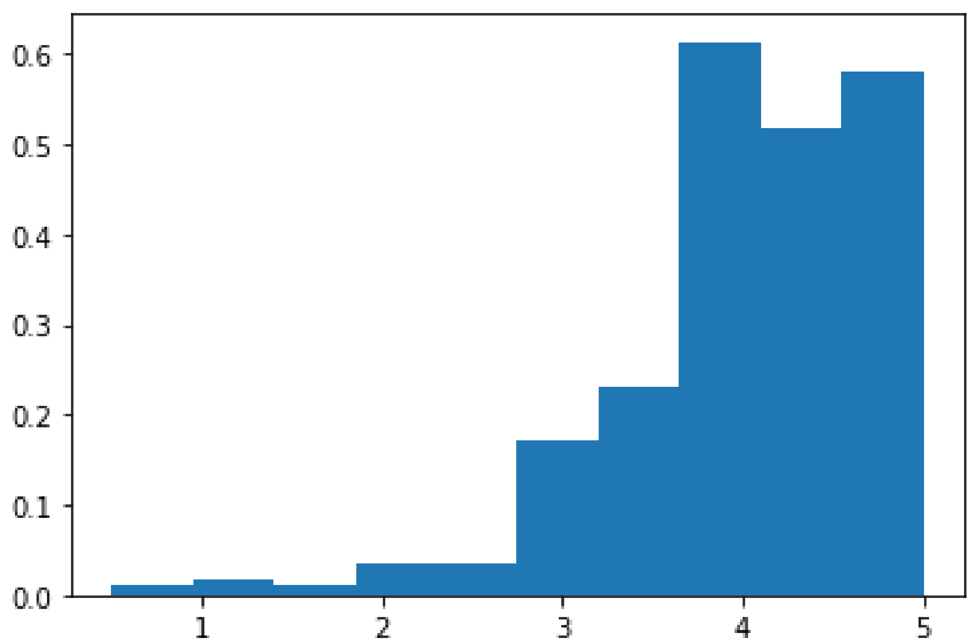
Initial data exploration, I look at word cloud on title, genre, and the overview of the movie.

The title word cloud is below. The larger the text the more frequent it shows up in the title. The word cloud below shows that Love Man and Girl are the top 3 words to show up in the title.

In the genre world cloud, we can see that drama does come up a lot. The combination of Comedy Drama seems like to be the most produced genre. I am speculating because it is a popular genre.



Another exploratory analysis is with the rating data set. From looking at the overall rating histogram we see a right skewed distribution.



This is common on rating items. My guess is that the users tends to watch things they might intuitively think they would likely enjoy.

Data Modeling

For constructing my models, I have taken 3 approach. With movie recommendation systems, we can apply content base filtering, collaborative filtering or a hybrid model that combine the two previous methods.

Content base filtering, as its name suggest, is to make recommendation to the user based on the content that user have watched. Thus, maybe we may find a particular user would watch movies from a particular actor and that may be a significant signal that could help the engine find movies featuring this actor to recommend this specific user. Or we may find specific keywords in description of the movie that are similar to form a recommendation.

My first attempt at content-based filtering is to recommend based on keyword and overview. From those columns I applied TF-IDF. TF-IDF basically is a method that counts frequency of terms but also identify commonly used words like 'the' that has no significance to modeling and lessen their impact. From that we apply the cosine similarity scoring to identify how other movies is close to a given movie. Then the recommendation will be a list of top 30 movies in terms of similarity score.

Working on the training data, the output looks like this. The following example shows the recommendation for movies like 'Toy Story' based on overview and tagline.

```
In [40]:  j = get_recommendations('Toy Story').head(10).index

          metadata_df_sub.loc[j,['title','tagline','overview']]
```

Out[40]:

	title	tagline	overview
3024	Toy Story 2	The toys are back!	Andy heads off to Cowboy Camp, leaving his toy...
10389	The 40 Year Old Virgin	The longer you wait, the harder it gets	Andy Stitzer has a pleasant life with a nice a...
8410	The Champ	NaN	Dink Purcell loves his alcoholic father, ex-he...
3084	Man on the Moon	Hello, my name is Andy and this is my movie.	A film about the life and career of the eccent...
11715	Factory Girl	When Andy met Edie, life imitated art.	In the mid-1960s, wealthy debutant Edie Sedgwi...
6504	What's Up, Tiger Lily?	WOODY ALLEN STRIKES BACK!	In comic Woody Allen's film debut, he took the...
1092	Rebel Without a Cause	The bad boy from a good family.	After moving to a new town, troublemaking teen...
11508	For Your Consideration	NaN	Three actors learn that their respective perfo...
5859	Class of 1984	Class of 1984. Is this the future?	Andy is a new teacher at a inner city high sch...
1952	Condorman	An action adventure romantic comedy spy story.	Comic artist and writer Woody performs a simpl...

It seems like the recommender is pickup more the comedy element of the movies rather than other animations.

The second content base filtering was based on movie meta data instead. I looked at characteristics such as top 5 actors, director, genres, and keywords. Using ‘The Godfather’ as our target movie, we get the following recommendation.

```
get_recommendations('The Godfather', cosine_sim=cosine_sim2)
```

```
3]: 1199              The Godfather: Part II
    1934             The Godfather: Part III
    10261                        The Outfit
    11733          The Consequences of Love
    5309                         The Gambler
    3327               ...And Justice for All
    4602                      The Cotton Club
    9517                       The Black Lapp
    1614                       The Rainmaker
    12221                        10th & Wolf
    1648                      Ill Gotten Gains
    3487          Jails, Hospitals & Hip-Hop
    6744                                Ruby
    7772                            Mitchell
    8001       The Night of the Following Day
    5793                     True Confessions
    1430                       Donnie Brasco
    549                         Trial by Jury
    2112                     The Paradine Case
    10729                   House of Strangers
    13361                   Manhattan Melodrama
    276                     Murder in the First
    9874                       Amongst Friends
    1186                       Apocalypse Now
    3640                              Serpico
    4080                               Pixote
    7052                 Shoot the Piano Player
    8698                      Branded to Kill
    11287                               G-Men
    13597                             Il Divo
```

This seems a fairly good recommendation for a first attempt.

Next, I also attempted collaborative filtering. Collaborative filtering on the other hand is based other people ratings to make a recommendation. I believe more sophisticated method would group similar users first and find recommendations for other users in that segment. For the collaborative filtering, I am using the Surprise package.

The last method is the hybrid approach. Hybrid methods are mixture of the methods mention above. The method I have adopted here is to take the recommended movie for a user via Content Base filtering first then run a collaborative filtering to predict the user’s rating to make recommendation. So first user 1, our recommender made the following for ‘The Godfather movie’.

```
hybrid(1,"The Godfather",cosine_sim2)
```

2]:

	title	vote_count	vote_average	id	est
1186	Apocalypse Now	2112.0	8.0	28	4.613409
7052	Shoot the Piano Player	69.0	7.2	1818	4.352920
3640	Serpico	429.0	7.5	9040	4.330476
1199	The Godfather: Part II	3418.0	8.3	240	4.316102
4012	Gardens of Stone	25.0	5.5	28368	4.285536
1430	Donnie Brasco	1175.0	7.4	9366	4.125541
11733	The Consequences of Love	125.0	7.6	24653	4.119607
1614	The Rainmaker	239.0	6.7	11975	4.048667
3327	...And Justice for All	118.0	7.1	17443	4.004266
4080	Pixote	24.0	8.4	42148	3.994565

Model Validation

To see how my model performs, I apply precision at K metric. The idea here is to see how many items that with the actual rating higher than the predicted rating. The value range from 0 to 1 where 1 is perfect. I compare my model which is using the SVD algorithm to the base line. We can see there is significant difference between the models.

SVD

[86]:

```
for trainset, testset in kf.split(data):
    algo.fit(trainset)
    predictions = algo.test(testset)
    precisions, recalls = precision_recall_at_k(predictions, k=15, threshold=3.5)

    # Precision and recall can then be averaged over all users
    print(sum(prec for prec in precisions.values()) / len(precisions))
    print(sum(rec for rec in recalls.values()) / len(recalls))
```

```
0.6773641924688965
0.5724082177629027
0.6781626724448315
0.5809743942524471
0.6856334618028078
0.5842087998220512
0.6876841223307756
0.5858252813718233
0.6831897090921978
0.5819061289397528
```

Base line with Normal distribution

```
In [88]:  for trainset, testset in kf.split(data):
               algo_norm.fit(trainset)
               predictions = algo_norm.test(testset)
               precisions, recalls = precision_recall_at_k(predictions, k=15, threshold=3.5)

               # Precision and recall can then be averaged over all users
               print(sum(prec for prec in precisions.values()) / len(precisions))
               print(sum(rec for rec in recalls.values()) / len(recalls))

           0.5821217419887966
           0.4348215174573881
           0.5853237597005768
           0.4336387928090972
           0.5765403600913539
           0.4320687982419843
           0.583347459044794
           0.42839280690906817
           0.5825129631943087
           0.43548330660210877
```

Final Word

Further refinement may improve the recommender. I think I could use apply clustering to group users that may have preference with certain genres or other characteristics. In this project, I encounter memory issues and cannot fully utilize the entire dataset. If I can get more memory perhaps I can improve the recommendation.