A RECOMMENDER SYSTEM

My exploration in developing a movie recommendation algorithm

Spencer Goble

11.24.2021

Capstone 2 White Paper

Data Science Career Track

Springboard

[Project Repository]

INTRODUCTION

My interest in the art of recommendation grew from my passion for DJing. As a DJ the act of delivering the appropriate content to the lister is integral to the artform. The controller of the music has to read the crowd, feel the mood, understand tempo, time, and harmony and use all that data to keep the listener's attention - to maintain a rhythm of consumption. The DJ is essentially a living algorithm; training and testing models on the fly, receiving feedback and iterating.

As I parsed through datasets on Kaggle I came across The Movies Dataset. It dawned on me that a set of thousands of films is analogous to a vast record collection. I have been fascinated by Spotify's ability to recommend songs as a DJ does and decided it would be fun to develop a film recommendation system.

DATA

The dataset consisted of 45,000 movies. Along with expected features like title and genre there was also extensive information on the cast beyond just the main actors, as well as the entire crew from director all the way to lighting assistant. There were also numeric metrics like budget and revenue along with scores unique to TMDB, like popularity and vote average. These scores are derived from users who log on to the TMDB website to search and rate movies. I dropped several features that I deemed unimportant, cleaned and wrangled the remaining information and was left with a dataset consisting of the following features:

1. Movie ID (a unique number to track the film)
2. Title
3. Genre
4. Keywords (topical terms related to the film's content)
5. Director
6. Top 3 Actors
7. Overview (a brief movie summary)
8. Language
9. Production Country
10. Production Company
11. Release Year
12. Runtime

13. Budget
14. Revenue
15. Average Vote Score
16. Vote Count
17. Popularity Score

There was also a dataset of user ratings. This was a list of over 600 users that had made a cumulative total of 100,000 movie ratings. The ratings dataset consisted of the following features:

1. User ID
2. Movie ID
3. Rating

APPROACH

In the development of recommender systems there currently exists two main approaches; **Content-based** and **Collaborative Filtering**. The former uses information about the film to find similar films. The latter compares the ratings that different viewers give to similar items in order to find viewers with similar taste. For example, if User A and User B both like Pulp Fiction and Requiem For A Dream and User A also likes Snatch, then the assumption is that User B will also like Snatch. One issue with Collaborative Filtering is called the **'cold start problem'**. That is, when starting out with very few ratings, there hasn't been enough consumption by users to draw similarities. E-commerce businesses often run into this issue when starting out. Since I had a dataset of user ratings, this wasn't an issue for me.

Libraries:

Tons of libraries are available for Python machine learning projects. The most important ones I used are:

1. **Numpy** - Supports large multidimensional arrays and matrices. Allows for quick computation of statistical metrics. Formats datasets for modeling.
2. **Pandas** - Creates well formatted data-frames that allow for data organization and manipulation.
3. **SciKit Learn** - An extremely robust collection of machine learning algorithms and tools.

4. **Surprise** - A tool-kit specifically designed for building recommender systems, with an emphasis on Collaborative Filtering.
5. **Matplotlib** - A very robust toolkit for creating readable plots and graphs.
6. **Seaborn** - Data visualization tool based on Matplotlib. Allows for simple and quick data plotting.

PROCEDURE

There are many ways to approach **Content-based** recommendation. **Text vectorization** is one of the most popular ways. This method takes all of the text information available for each film and mashes it together into a string of words. This creates a text 'profile' for each movie. After the profile is created, **cosine similarity** is used to draw correlations between profiles. **Cosine similarity** is a function that plots each film's profile into a high dimensional space and draws a line from the origin (0,0) to the point of the profile. Then the angle between lines is measured and lines with the smallest difference in angle are the most similar. This method is great for items with high dimensional variation because two items might be far away from each other in terms of Euclidean distance (point A to point B) but the degree of difference relative to the origin is similar. The image below is how the dataset looked after text-vectorization. The 'text_info' column is a mash-up of words that is hard to interpret for a human but designed for a computer to 'read':

	id	title	text_info
0	862	Toy Story	jealousi toy boy friendship friend rivalri boy...
1	8844	Jumanji	boardgam disappear basedonchildren'sbook newho...
2	15602	Grumpier Old Men	fish bestfriend duringcreditssting oldmen walt...
3	31357	Waiting to Exhale	basedonnovel interracialrelationship singlemot...
4	11862	Father of the Bride Part II	babi midlifecrisi confid age daughter motherda...
...	...	...	...
46619	439050	Subdue	tragiclov leilahatami kouroshtahami elhamkorda...
46620	111109	Century of Birthing	artist play pinoy angelaquino perrydizon hazel...
46621	67758	Betrayal	erikaeleniak adambaldwin juliedupage markl.les...
46622	227506	Satan Triumphant	iwanmosschuchin nathalielissenko pavelpavlov y...
46623	461257	Queerama	daisyasquith daisyasquith daisyasquith

44984 rows × 3 columns

The other approach I employed is called **Clustering**. This method groups n-points into n-groups based on a given measure of proximity or similarity. For this technique to work the data has to be all numerical. In order to achieve this I needed to **one-hot encode** my data, which is a process of taking the values of a given column in a dataset and creating a new column for that value. For example, if one of the columns was *Genre*, and each film had the name of a gene as a value (Action, Comedy, etc.), then each genre name becomes a new column and if a given film is of that genre it's assigned a '1' as the column value, if not it's assigned a '0'.

Before One-Hot Encoding:

Title	Genre
Die Hard	Action
Meet The Parents	Comedy
The Notebook	Romance

After One-Hot Encoding:

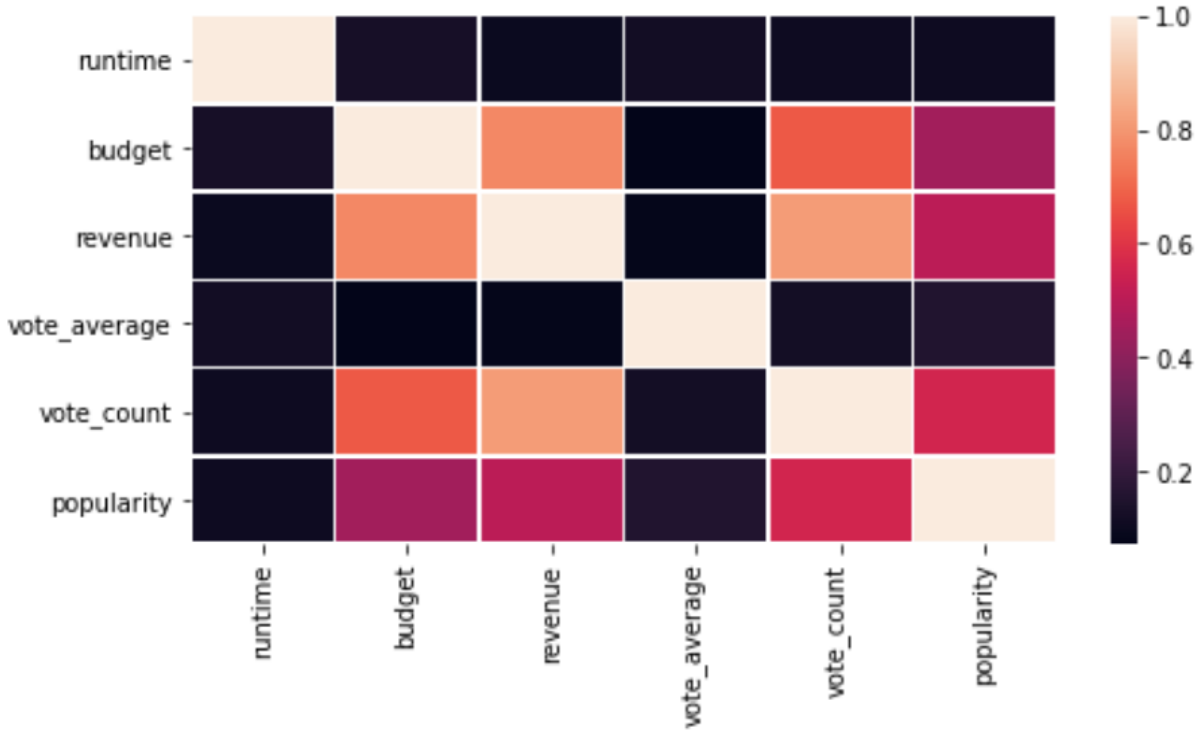
Title	Action	Comedy	Romance
Die Hard	1	0	0
Meet The Parents	0	1	0
The Notebook	0	0	1

The process for **Collaborative Filtering** was a bit different. Since I have user ratings for dozens of films I was able to treat this process as a **supervised learning** one. That means I have a target variable (the rating a particular user has given a particular film) that I can use to evaluate the efficacy of my model. The Python library ***Surprise*** has a great set of

tools for Collaborative Filtering. I evaluated three different algorithms; **Singular Value Decomposition (SVD), Non-negative Matrix Factorization (NMF), and K-Nearest Neighbors with Means (KNN)**. **SVD** and **NMF** are both matrix factorization techniques that use vectors to transform matrices. The **KNN** technique is a classic **classification** approach that evaluates an instance, in this case a user and a rating, and creates groups based on similarity with other instances of user ratings.

EXPLORATORY DATA ANALYSIS

Being that my overall project required an 'unsupervised learning' approach with text-centric categorical data my EDA step was a bit minimal. I did a bit of exploratory analysis of some of the numerical data. Below is a heat map of the numerical features of my dataset:



There are some clear correlations here. Budget and revenue are **positively correlated**, which makes perfect sense. With a higher budget, a film is more likely to be of high quality and receive high amounts of publicity and proper promotion. There is also a positive correlation between budget/revenue and popularity. The popularity score is determined by the TMDB website and calculated by several factors including; average

rating, vote count, amount of searches, etc. So the more 'buzz' a film has, even if it's negative, the higher the popularity score. There is also a positive correlation between budget/revenue/popularity and vote count. The more 'buzz' a movie has the more likely people are to score it, thus increasing the vote count.

PRE-PROCESSING

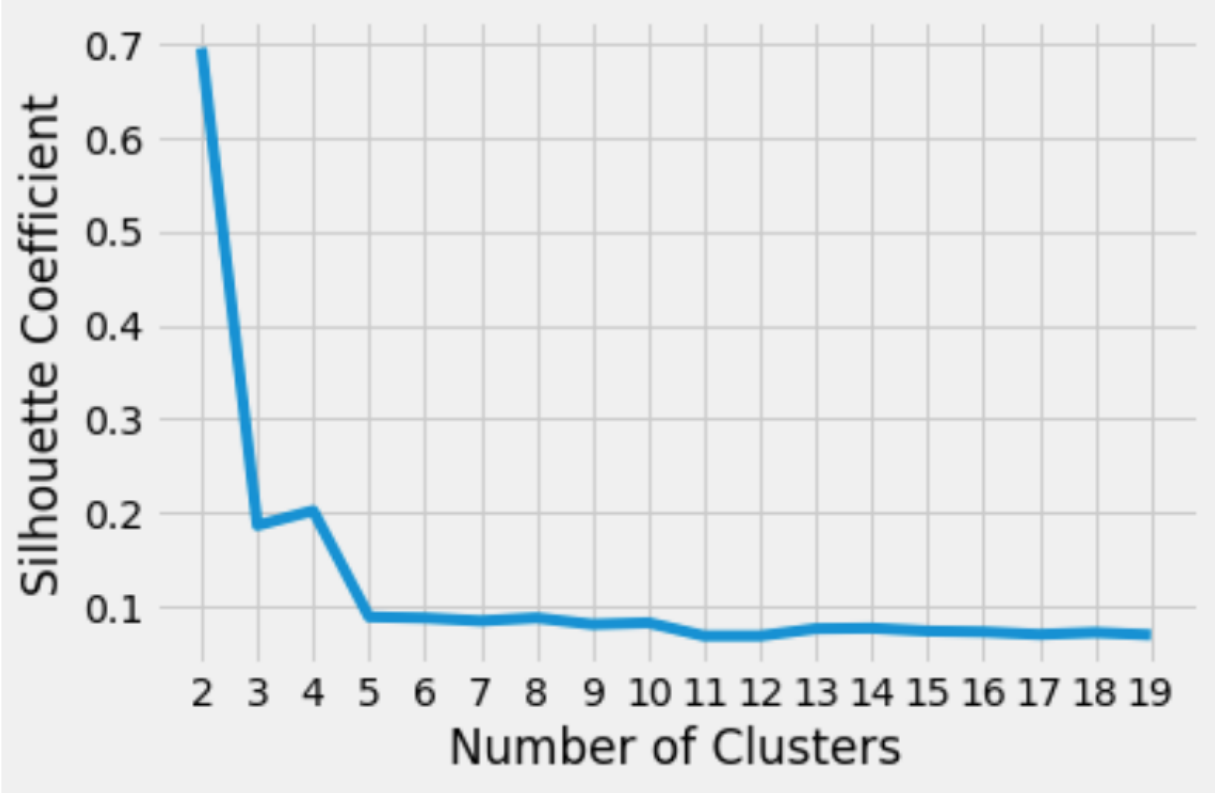
Clustering:

I began by exploring the parameters of the **K Means** clustering algorithm. In this approach I start by predetermining a set amount of clusters. Data points which are closest to a given cluster center get grouped together. The **sum of squared errors (SSE)** is a metric used to measure the efficacy of clustering. The lower the SSE the more effective the parameters are. As the number of clusters increased the SSE decreased, revealing that a high number of clusters was necessary to effectively group films. Below is chart showing the SEE decreasing as number of clusters increases:

Note: My use of clustering was difficult because this method can be very computationally expensive. I was required to reduce the dimensionality of my dataset because my computer

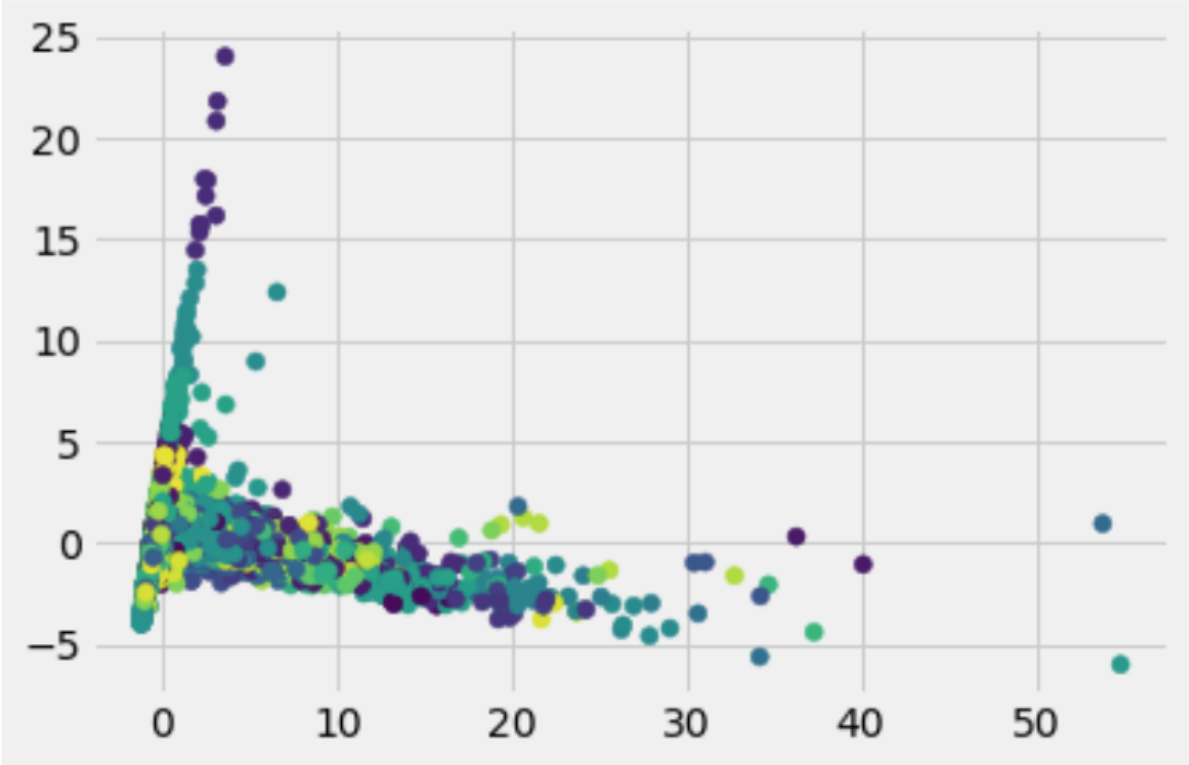
was having trouble while scanning for optimal hyper-parameters. My dataset was initially 3000+ columns, but I scaled it back to 350.

Next I employed a method that produces the **Silhouette of Coefficients**. This method gives a score that represents how similar a datapoint is to its cluster compared to other clusters. It evaluates the efficacy of clustering performance. The score ranges from -1 to 1, where a higher score is better. A higher score shows that data points are close/similar to their own cluster and far/different from neighboring clusters. Below is a chart showing Silhouette scores for various cluster amounts You can see the score decrease dramatically after the first 5 clusters are formed. This shows that after establishing 5 clusters the difference between grouped data points shrinks and they share similarities despite different cluster membership.



Next I instantiated a clustering algorithm called **DBScan** or Density Based Spatial Clustering of Applications with Noise. As opposed to a predetermined amount of clusters, this algorithm uses a parameter that dictates the maximum distance that two data points can share in order to be clustered together. After experimenting and researching this algorithm I determined it wasn't optimal for my project, because in general DBSCAN does not do well with very wide and sparse datasets, which mine was.

Next up I employed **Principal Component Analysis (PCA)**. PCA is a method of dimensionality reduction that allows one to observe the amount of components, which are derived from features, that are responsible for most of the variability in the data. I plotted 2 components for the dataset. The data points made an interesting L-shape on the 2-dimensional plot. For component 1, the variability was spread out pretty wide from 0 to 20. For component 2 the data was more clustered between -5 and 0. I believe that because my dataset is so wide and sparse, the data when the feature space was reduced to 2 principal components. The chart is below:



Collaborative Filtering:

Next up was the preprocessing of the user ratings dataset. I performed 5-fold cross validation for the SVD, NMF, and KNN algorithms and evaluated the **Root Mean Squared Error (RMSE)** of each. The RMSE measures the degree of error of a model's prediction. A lower RMSE score suggests a more effective model. Listed below are the mean RMSE scores for the 5-folds of cross validation of each algorithm:

1. SVD: **0.8961**

2. NMF: **0.9467**
3. KNN: **0.9189**

SVD has the lowest score so I planned to use it in my final model. KNN wasn't far behind SVD so I would consider testing it as well during modeling.

MODELING

Clustering:

I began my modeling step with the Clustering approach. I employed both **K Means** and **DBScan**(out of sheer curiosity). I used the **'fit' and 'predict' methods** to run the algorithms on my dataset and generated predictions. I then created two new columns in my dataset (one for each algo), that contained the cluster number that a given movie was assigned to. As expected K Means appeared to be more effective. DBScan created one very large cluster, containing over ~50% of the films. I returned to the parameters of this model to adjust the distance metric, but it was never able to match the efficacy of K Means. K Means on the other hand dispersed the films throughout the clusters nicely. I experimented with adjusting the cluster number and 300 seemed to be an appropriate amount of groups. I then created a function that takes in a movie title and produces a list of films that have the same cluster number, i.e. the recommendations. I entered *Inception* and the following list was pretty decent:

1. *The Matrix*
2. *Iron Man*
3. *The Hunger Games*
4. *Django Unchained*
5. *Interstellar*
6. *Guardians of the Galaxy*
7. *Mad Max: Fury Road*

The recommendations had some very solid ones: *Interstellar* is another Christopher Nolan film. *The Matrix* is a philosophical action film about perception and society. The remainder of the films are all action/adventure films that are in the ballpark!

Collaborative Filtering:

Next up, was the Collaborative Filtering approach. I used a method called Grid Search Cross-Validation to tune my hyperparameters. This method tests out a bunch of parameter settings. It fits the data and generates a RMSE score based on the given parameter set and then determines which settings are the most effective. I applied this method to both the SVD and KNN with Means algorithms. As expected based on my Preprocessing step, the optimal hyper parameters were more effective for SVD than for KNN. After generating the optimal parameters I applied each model to generate a predicted rating that User X would give movie X and compared both against the actual rating that User X gave movie X. SVD predicted a rating of 3.35, KNN predicted 3.09 and the actual rating was 3.5. SVD takes the cake as the most effective collaborative filtering model!

Cosine Similarity:

Next up, was the Text-Vectorized dataset. It was time to employ the Cosine Similarity algorithm. First I created a vectorization matrix using 'count_vectorizer' from SKLearn, which is similar to One-hot encoding. It parses through all the strings of text I created for each film and creates columns for each unique value. Essentially it creates an insanely wide dataset (439,716 columns to be exact). Then using the cosine_similarity function from SKLearn, I generated similarity scores for every movie based on the matrix. So now, every film has a score for how similar it is to every other film. Next, I wrote a function that takes in the title of a film and produces a list of similar films based on the similarity scores. The list for *Interstellar* is as follows:

1. *Inception*
2. *Doodlebug*
3. *The Prestige*
4. *The Dark Knight*
5. *The Dark Knight Rises*
6. *Following*
7. *Dunkirk*
8. *Batman Begins*
9. *Insomnia*
10. *Memento*

The first most notable thing is the correlation by director. I added more weight to the director when producing the text string during the wrangling step. Since the director

really determines the feeling of a movie I figured this would make sense when making recommendations.

Below is the recommendation list for *Mallrats*:

1. *Clerks - The Flying Car*
2. *Jay and Silent Bob Strike Back*
3. *Clerks*
4. *Clerks II*
5. *Jersey Girl*
6. *Dogma*
7. *Zack and Miri Make a Porno*
8. *Chasing Amy*
9. *Cop Out*
10. *Tusk*

Another solid list. *Mallrats* is a favorite film of a good friend of mine, when I relayed the recommendation list he smiled and claimed to love all the films in the list.

Hybrid System:

With some solid recommendations it was time to design a Hybrid system. That is, a combination of Content Based and Collaborative Filtering. There were two ratings sets that came with the original batch of data. Due to limited computing power I needed to use the smaller set. My plan was to combine my Cosine Similarity algorithm with the SVD Collaborative Filtering model. The small ratings dataset contained less films than were in the text-vectorized dataset, so I reduced the size of the text dataset to contain only films that were in the ratings dataset. Then I ran the cosine_similarity model on the reduced dataset to create the similarity matrix. I then created a function that takes in a movie title and a User ID. It then generates the list of similar movies and the predicted rating the given user will assign that film.

I then made a function that produced a list of movies a User has already rated well (3.5 or higher). This list served as a starting place for the recommender, because up until this point I had been *assuming* that a user liked the movie that I generated a list for.

To bring it all together I created a more elaborate function that takes a movie title and User ID as an input and produces a list of similar films and the predicted rating. This list however gets ordered by the predicted rating. So the film that User X is predicted to like

the most is at the top. The final recommendation list for *Inception* looked like this:

Based on your enjoyment of *Inception*, we think you may like the following:

1. *Memento 4.34*
2. *The Dark Knight 4.29*
3. *The Prestige 4.27*
4. *Interstellar 4.21*
5. *Batman Begins 4.06*
6. *The Dark Knight Rises 3.97*
7. *Don Jon 3.86*
8. *Following 3.82*
9. *Insomnia 3.75*
10. *Pacific Rim 3.72*

The title and the predicted rating are present and ordered appropriately!

CONCLUSIONS

Overall the cosine similarity felt the most thorough. Joining it with collaborative filtering created an even stronger set of recommendations.The clustering technique produced the weakest outcome. I strongly believe this is mostly due to my lack of computing power. If I was able to generate an immensely wide dataset with every actor/director/tag then clustering would have performed better. My initial data frame that contained the top 1000 actors/directors/tags was too vast and subsequently crashed when I ran the algorithm, which is why I had to scale it back to the top 100 actors/directors/tags, resulting in a decline of feature space quality.

After learning about Netflix's 2006 challenge to make their recommendation system 10% more accurate I was tickled to feel as though I was able to partake (though over a decade later). Had I been a Data Scientist at the time I feel I would have had some valuable skills to contribute!