Blake Burns
Christian Coulter
Alexander Pegg
CS 4347
May 7, 2020

Movie Recommendation System

Abstract

Watching movies is and always has been an immensely popular pastime. With the rise of streaming services in the 2010's, movie consumption is at a high and continually increasing. Typically, users are aware of only a small set of all available movies but are often in search of movies outside of their realm of familiarity. Because of this, movie recommendation systems are becoming increasingly popular.

In this project, we create a movie recommendation system based on ratings and genre and explore several algorithms and their performance in our system. These algorithms include k-nearest neighbors and matrix factorization using singular value decomposition. We found that the k-nearest neighbors algorithm consistently outperformed the singular value decomposition algorithm.

Introduction

For our project, we decided to develop a recommendation system for movies. The goal of this experiment was to find similarities between users and their ratings and create a model that would predict a number of movies that a user would enjoy when the system is given a movie title. We used a movie lens dataset with two csv files to find these similarities based on different recommendation techniques. The movie csv file contained 3 features, a movieID, which is a unique id number for every movie in the dataset, the title of each movie, and the genre of movie. The ratings csv file contained 4 features, a userID, which is a unique ID number for each user giving movie ratings, a movieID (same ID as in the movie csv file), the rating feature with a rating from 0 to 5 for each review given by a user, and a timestamp of the rating.

Methodology

One of the ways of implementing is using an item-based collaborative model. This could be done by something like k-nearest neighbor (or KNN), which would calculate the similarity between rating vectors, and then calculate the distance between a target movie and all the other movies in the database. Then it would rank these distances and give the top K nearest neighbors and these would be the recommended movies in our case.

Using a matrix factorization model, such as singular value decomposition (SVD), is another way of developing a recommender model. This method would allow for discovering latent features from the interactions between users and movies. Similar to kNN, you would create a matrix of the data and decompose it using SVD, then find the correlation for every movie pair in the dataset. Then using the movies with highest correlation, output them as the recommended movies.

Data Analysis

For this experiment, we used the kNN model as our baseline and compared it to another algorithm, SVD, using a cosine similarity score. Our baseline using kNN used a matrix created from the two csv files that had the columns as users and the rows as movie titles, and the values of this matrix were the ratings of movies given by each user (filled with zeros for non-rated movies). Once we had the matrix, we used the unsupervised algorithm NearestNeighbors. This algorithm used to compute nearest neighbors is "brute" and a cosine metric was specified to calculate the cosine similarity between rating vectors. After fitting the model, we specified 11 neighbors using kneighbors in order to calculate the distances of the 10 nearest neighbors and output them as the movie recommendations.

The other algorithm we used for recommending movies is with SVD. Similar to KNN, we create a 2-dimensional matrix, but with the columns as movies and the rows as movie titles. After this, we use TruncatedSVD to decompose it and fit it into the model for dimensionality reduction. We then calculated the Pearson's R correlation coefficient for every movie pair in our matrix. Then, we find the movies with correlation coefficients between 0.9 and 1.0 and output the top 10 results.

The metric we used to evaluate the models is average cosine similarity. We used sklearn to implement the function which calculates the pairwise angle between two vectors and returns the cosine of this angle, meaning that the closer the cosine similarity between two vectors is to 1, the higher the similarity between the two vectors. The function calculates this for each of the recommendations, which are paired with the queried title, and returns the average of this result .

KNN recommendations:

```
Enter a movie title for recommendations: Thor
Recommendations for Thor :

1: Iron Man 3 (distance: 0.3431007237569832)
2: Men in Black III (distance: 0.3589941135357907)
3: Iron Man 2 (distance: 0.43299411950968825)
4: Captain America: The First Avenger (distance: 0.43693210485283873)
5: X-Men: First Class (distance: 0.43879627195940263)
6: Captain America: The Winter Soldier (distance: 0.44430721940701623)
7: Avengers, The (distance: 0.4642197006447156)
8: Amazing Spider-Man, The (distance: 0.4683878924549958)
9: Daybreakers (distance: 0.4703141126264213)
10: Green Lantern (distance: 0.4728633286083196)

Average cosine similarity: 0.5602630422266682
```

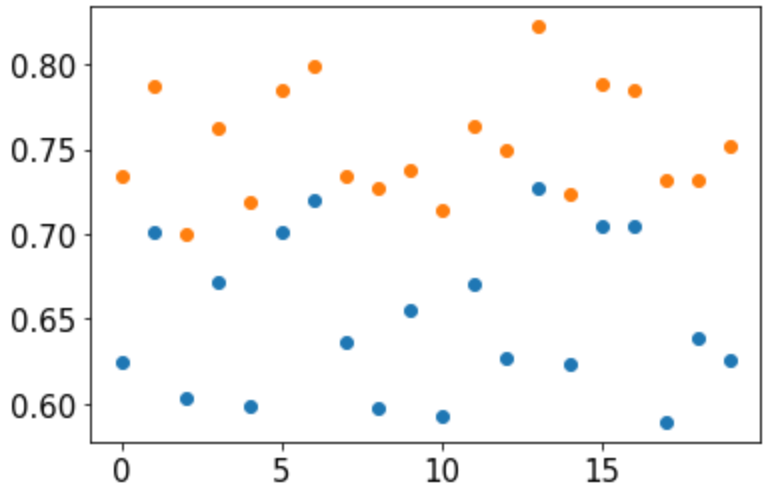
SVD recommendations:

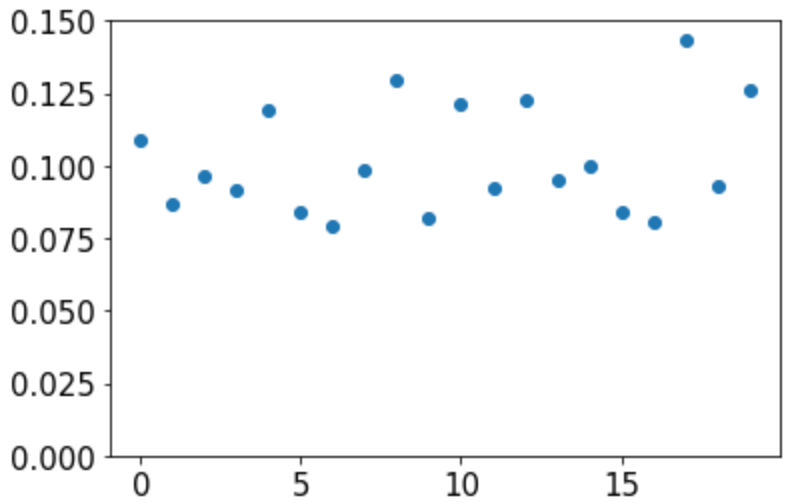
```
Enter a movie title for recommendations: Thor
Iron Man 3
Megamind
Scott Pilgrim vs. the World
Iron Man 2
Who Am I?
Despicable Me
Chappie
Wreck-It Ralph
Avengers, The
Dredd

Average cosine similarity: 0.447648765468142
```

In our tests, we took a sample of 100 random movies in the dataset and ran each movie through both of the algorithms. We compared the two algorithms and noticed that KNN typically outperformed SVD with a mean average cosine similarity of between 0.1 and 0.15 greater than that of SVD's.

```
Average average cosine similarity for KNN: 0.7517973139913608
Average average cosine similarity for SVD: 0.6387066864903324
```

We did the above 20 times with 50 random samples per run on both algorithms. The results are visualized below. The blue points were SVD runs and the orange points were KNN runs. The second plot is the difference between the mean average cosine similarity of the two algorithms. In all runs, KNN had a higher mean average cosine similarity than SVD.





Although there seem to be better results from KNN, there are still some ways for the model to be improved. One way to possibly get better results is by cleaning the initial data by combining the rating data with total counts of ratings, and filter out the less popular movies that would not be relative to the movie recommendations. Another way to improve this model would be to create a threshold that would limit the movies to having a certain amount of ratings to be considered in the recommendations.

Conclusion

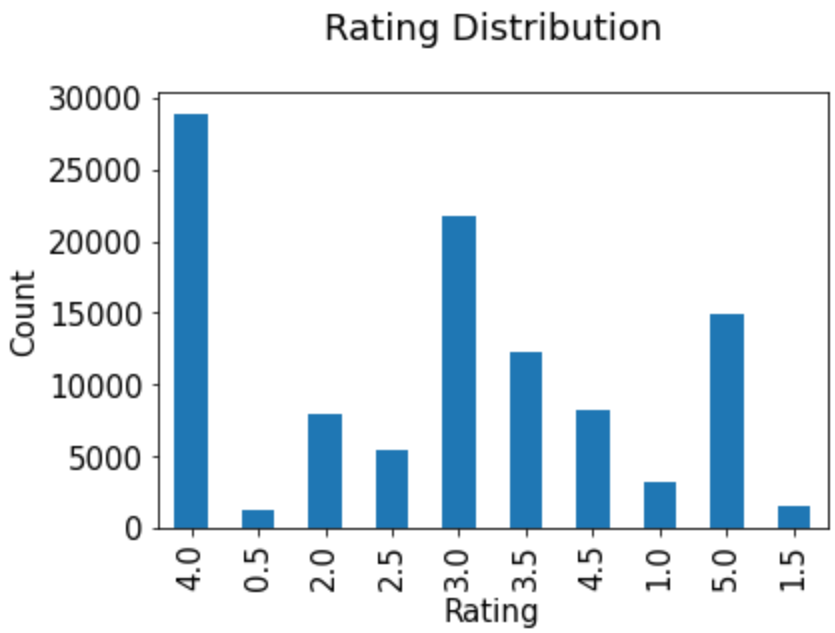
In conclusion, we were able to implement two models that seem to compute good results for movie recommendations and used cosine similarity scores for each to find which gave better results. For future use, a deep learning model could be implemented to possibly give better results than these models since embedding layers would allow for better fitting and learning of the model.

Appendix

Contributions of each member:

- Blake Burns: Imported and manipulated the dataset so that it was easy to work with. Helped with the KNN algorithm. Contributed to the report.
- Christian Coulter: Helped with the KNN algorithm. Created the SVD algorithm. Contributed to the report.
- Alexander Pegg: Helped with the SVD algorithm. Implemented average cosine similarity metric. Contributed to the report.

Distribution of ratings in the movie dataset:


Rating Distribution
Count
30000
25000
20000
15000
10000
5000
0
4.0
0.5
2.0
2.5
3.0
3.5
4.5
1.0
5.0
1.5
Rating