6/27/2018

What should we watch tonight? A Movie Recommender System

By: Ben Sturm

Project Summary

Project Design

For this project, I built a movie recommender based on the idea that people with similar tastes as ours may also like similar movies. This is known as a collaborative filtering based approach to recommendation. The data source I used to build my recommender is the MovieLens 20 Million Dataset, which consist of 20 million ratings of movies. Because of the large size of this dataset, there were some challenges to build my recommender system in a computationally efficient approach. I will discuss more about these challenges as well as some of the outcomes of this project.

Tools

To build my recommender system, I made significant use of the pandas, NumPy, and SciPy libraries. I used pandas to read in the data into a DataFrame and do all of my data cleaning and preprocessing. This dataset was very sparse, meaning that if I constructed a complete movie ratings matrix, most of the values would be NaN. Because of this sparseness, I used SciPy's sparse matrix package for handling my data. I then used SciPy's svds method, which is specifically designed to run SVD on sparse matrices. Using this approach, I was able to avoid having to construct the full Movie Ratings matrix, which would have likely made the problem intractable.

Data

As mentioned previously, I used the MovieLens 20 Million Dataset. The two files I used for this project were the Ratings data file and the Movies data file. The Ratings data file consisted of 20 million ratings from 138,000 different users across 27,000 different movies. The ratings scores were on a 0.5 – 5.0 scale. The Movies data file consisted of the different movie titles along with their genres. During my preprocessing and postprocessing steps, I had to merge these two datafiles in order to provide a predicted movie rating along with the movie title.

Algorithms

The algorithms I used for my recommender were part of the general class of algorithms called low rank matrix factorization. Specifically, I used singular value decomposition to reduce the Ratings Matrix with a large number of features to one with a smaller subset of features that allows us to approximate the original matrix. In order to optimize my model, I first did a train/test split of my data and then performed SVD on the training data while adjust the free parameter associated with the number of components k. Then, I ran an error analysis of the predicted scores versus the actual scores of the test dataset as a function of k.

I also came up with my own unique algorithm for this project that addresses the issue of providing ratings for two people watching a movie together. I call this my couples movie recommender. The way I addressed this problem was by creating separate recommendations for each member of the couple and then merging the two lists using a group rating function. My group rating function is given as follows:

$$R_G = \frac{R_1 + R_2}{2} - \frac{|R_1 - R_2|}{5}$$

Where R_1 and R_2 are the predicted scores for a particular movie for User1 and User2, respective, and R_G is the group rating function. This group rating function calculates the average predicted score minus a penalty term that is proportional to the difference of the two ratings. The purpose for the penalty term is to give more weighting to movies that have more similar predicted scores.

Things I'd do differently next time

Although I'm quite satisfied with the results of my project, there are a few things I would have done differently if I had more time. For starters, I would have liked to implement a more advanced algorithm than plain vanilla SVD. For example, the group that won the Netflix prize competition developed a collaborative filtering algorithm that does a better job at capturing both user and movie biases in the data. Thus if certain movies have higher than average ratings, these movies don't completely dominate the recommender algorithm. I did in part account for this fact by implementing a mean normalization of the movie ratings data, but the Netflix prize winners adopted a more advanced approach to this problem that likely does better.

Another thing I'd do differently if I had more time is to incorporate movie genre into my model. One of the issues I encountered is that if a user gives high ratings to a genre such as romantic comedies and gives low ratings to all other genres, the user does in fact get recommendations in the romantic comedy genre, but they also get other highly rated movies in other genres. I think this is partially where the plain vanilla SVD fails. So, I could try to implement some automatic filtering approach that would automatically detect that the user only likes a certain kind of movie (e.g., Romantic Comedy) and only return movies within this genre. It's possible, though, that once I adopt the move advanced algorithm used by the Netflix prize winners, I wouldn't need to do this.

Overall, I enjoyed working on this project and I plan to continue working on my model in order to improve its performance. I also plan to build a Flask application in order to allow anybody to use my recommender system.