Movie Recommendation System Report

Introduction

A recommender system is an algorithm used to recommend options to users based on their past preferences/actions. These systems are especially useful for applications such as streaming services, e-commerce and social networks where users have a large amount of options, and need direction on what choices to explore and engage with. A few types of recommendation systems are user-based, item-based and random walk. The suer based approach observes what items users similar to the original user have interacted with/rated highly and recommends those options. Meanwhile, the item based approach focuses on the items the original user interacted with and recommends new items with highly similarity to those items. Meanwhile, a graph based random walk uses nodes in a bipartite graph to find and identify relationships based on connections. It traverses the graph from user to item and helps identify indirect connection based on the walk.

Dataset description

In the jupyter notebook we are using the MovieLens 100k dataset. The dataset consists of 3 files: the u.data file containing the user-movie, rating that has the user id, movie id, rating and timestamp. It also contains the u.item file which contains the movie metadata and has fields movie id, title, release date, and IMDB website. And the final file in the dataset is the u.user file and contains demographic information about the user such as the user_id, age, gender, occupation and zip code. From the u.data file we extract the following features, movie id, title, and release date. Meanwhile the u.item file we extract the features movie id, title, and release date. Finally, from the u.user file we extract the features user id, age, gender and occupation. Some of the preprocessing that we performed on the data is that from rating we converted the timestamps to dates. For all 3 sets of data we also checked for any missing values and dropped the missing data. We also ended up dropping any duplicate data from all of the data sets ensuring that everything is unique.

Methodology

In the Jupter notebook we use three different recommendation techniques implemented. The first recommendation we implemented was the user-based collaborative filtering. In this method we created the user_movie_matrix. We had the rows of the matrix represent users and the columns of the matrix represent movies. To check the similarity of the users were used cosine similarity, and those are the values that are present in the cells of the matrix. The cosine

similarity is based on the rating between users for the similarity. If a movie has not been rated we just filled in the missing values as 0. Finally the system finds the top n movies that are similar and returns them.

The second recommendation system that was implemented is the item-user based collaborative filtering. This is similar to the user-based collaborative filtering. However, in this method when computing the cosine the user_movie_matri has to be transposed before calculating the cosine similarity since it is targeting similarity between the movies rather than users. Through this we can identify how similar different movies are. After creating the similarity matrix we identify the most similar movies based on their cosine similarity values and recommend the top n most similar movies.

The third recommendation system that was implemented is the graph based recommender that is inspired by pinterest's pixie inspired algorithm. The algorithm works through the use of a bipartite graph with nodes representing either users or movies and the edges connecting opposite type nodes, and representing the interactions between users and a movie. This method works through the use of random walks which go through the graph starting at either a movie or user, and randomly traveling through connected nodes. After travelling through a fixed number of nodes, the most visited nodes are returned as the recommendations. This approach is effective because it shows the indirect relationships between users and items as well as the direct relationships. Through utilizing the structure it allows the recommendations to be more personalized and to have them updated easier since only the connection needs to be updated and similarity does not need to keep being recalculated.

Implementation Details

For all methods the user_movie_matrix was used. It was created through the use of the pivot function with the index being user_id, column being movie_id and values being the ratings. This method was used on the ratings matrix to reshape the data. The item-based collaborative filtering was implemented through the use of python and the pandas, numpy and the sklearn libraries. To calculate the cosine similarity we use the cosine_similarity function from sklearn on the transposed user_movie_matirx module and then fill the values that are na with 0s. The python data frame is created from the matrix. After that a recommended movies matrix function is written where it takes a movie name, and the amount of recommendations the user wants with a default of 5. First it checks if the movie id exists in the matrix if it doesn't exist then the movie not found is returned. Then the similarity values from the dataframe are obtained, and sorted by ascending order through the sort_values function. Finally, a new data frame is created by merging the similarity scores with the movies matrix and the titles of the top n movies are returned in a dataframe with the index ranking.

The user-based collaborative filtering was also implemented through the use of python pandas, numpy and sklearn libraries. We calculated the cosine similarity the same as we did for the item based collaborative filtering, by using the cosine_similarity function through sklearn.

Then we created a pandas dataframe for the user_similairty matrix. After that the recommend_movies_for_user function was created, that took the user_id, and an input of how many recommendations the user wanted. For the number of recommendations input it is set at a default of 5. First thing the function does is remove the score for the user itself so the recommendation doesn't just return its own recommendation. Then it sorts the similarities in descending order through the use of the sort_values method for data frames. AFter this the average rating per movie across users is calculated and sorted and the top n movies are retrieved. Finally the top movies are merged with the movies dataframes on the 'movie_id' index and a dataframe is returned displayingthe rankings and movie name.

The graph based recommender was also implemented in python through the use of the pandas and random libraries. First before the algorithm was created the movies dataset had to be preprocessed and a graph had to be constructed. It was created by merging the ratings and movies matrix on the movie_id column, to create the new ratings matrix. This is then grouped by user_id, movie_id, and title, with the mean rating being computed for each user using the mean() function. After this the ratings are normalized using the group by function to group ratings by user, and the ratings are adjusted using the transform functions with a lambda function x: x - x.mean(). Finally the graph is created through the use of a dictionary where we add in the connections for the user_id, andmovie_id pairs from the ratings dataset using a for loop.

After the graph is created we need to implement the random walk using pandas and random. It si simplimed through the random walk function which takes in a graph, the starting node, a walk length and the number of recommendations desired. In the function a movie_visits dictionary is created and the original node is kept track. Meanwhile, the movie_ids are converted to a set. Then a for loop is created for the length of the walk. A list of neighbors is created that contains the connections of the current node, if there are no connections then an empty list is returned and we leave the loop. Otherwise, if the node is a movie, and the movie is not the original movie given then the movie visit for that movie is incremented by one. Then through the choice function from the random library a random index of the neighbors array is selected and serves as the next node. When the loop is complete, then the dictionary is sorted by highest to lowest using the sort method in the dictionary items, and the top n movies are returned. Using the random_walk function we can create the weighted_pixie_recommend_user and weighted_pixie_recommend_movie methods. These both take in the graph, walk length, and number of movies with the only parameter that is different is that one takes in the movie id and the other takes in the user id for the starting node. They both call the random walk function, and get the results which they convert to data frames using the pandas method DataFrame. Finally they merge the data frames with the movie matrix using the merge function on movie id, and return the title column of the dataframe.

Results and Evaluation

For the user based recommendation system we generated 5 movie recommendations for the user with id 10. The system returned the recommendations of Star Wars, Fargo, Return of the Jedi, Contact and Raiders of the Lost Ark. The main thing that I uncovered from the user based recommendation system is that the system mainly returns very mainstream movies and seems to stay away from niche movies. This is likely because people will have more interactions with popular movies and thus have overlap with popular movies. A limitation that is present in our implementation is that the matrix was sparse so the system ends up relying on users with more common items. Meanwhile with item based similarity I used the movie GoldenEye to generate a recommendation of 5 movies. The recommendations that I received were Under Siege, Top Gun, True Lies, Batman, and Stargate. This approach gave more consistent results that are more genre specific and relevant to the movie at hand, and is less susceptible to user sparsity. It however, is prone to missing the user context and may miss nuances of what the user prefers. Meanwhile with the graph based approach I used user id one to generate 5 recommendations which were Cinema Paradiso, Rosewood, Manchurian Candidate, Life, Less Ordinary, and Kids in The Hall. I also generated 5 recommendations based off the movie Jurassic Park, which returned Robin Hood, Crow, Three Colors, Henry V, and Nixon. This approach I noticed returns more diverse recommendations compared to user and item based systems. It finds more indirect connections between users and items. The main limitations are that compared to item based and user based recommendation systems the reason why a certain item was recommended is less explainable and the system may hit a wall if a movie or user has no connections and hits a dead end.

Conclusion

In this project we implemented three recommendation systems: user-based, item-based, graph based. The main key takeaways that were found from each method is that all three methods are great tools for finding recommendations for users but have different circumstances where they are optional due to their trade offs. User based collaborative filtering is best when you prioritize capturing personalized preferences and don't mind recommending mainly mainstream data. This approach should be generally avoided when you have sparse data or lots of new users. Meanwhile, the item-based approach is more optimal when you need more consistent recommendations that are more scalable, but aren’t as personalized. Finally, the graph based approach is best when you want diversity in recommendation and want to uncover any hidden connections. Some potential improvements that we could make to the user based and item based approach is to combine them into one hybrid model. We could start off by using an item based approach at first to give users recommendations when there is no data, and then transition to a more user-based approach to increase personalization. Additionally, in the graph based approach we could weight which user nodes that the walk would travel to based on user preferences. For example, if they have a list of friends the algorithm would be weighted to have the walk biased towards the friends nodes. A few real world applications of these models is with user-based recommendations you can use them for things such as book recommendation

systems where people get recommendations for books based on the books others read. Additionally, item based collaborative filtering can be used in e-commerce to recommend users new products based on what they have already bought. Finally a graph based method can be used in social media networks to find connections to recommend to users that aren't immediately obvious such as in LinkedIn.