Collaborative Filtering Methods

Gweneth Stewart
Foundations of Data Science
Boston University
gstew@bu.edu

1 Introduction

In today's world of high consumerism and fast technology it is incredibly important to companies and consumers to have reliable and accurate recommendations. The ability to predict and know what a customer wants to before they know allows for a more personalized customer experience. With a more personalized experience follows increased customer interaction and in turn brings in more money to the company. However, creating this personalized experience comes with challenges. Large amounts of sparse data, new customers with no previous data, and scalability issues cause problems for data scientists. This is why research on recommendation systems is necessary and beneficial. Today there are many different methods that combat these issues. I plan to investigate a few of the more basic methods to evaluate their performance and see if they would still be useful to implement even though there are more advanced methods that exist. Sometimes it is better to have a wide range of recommendations rather than a few very specific and accurate recommendations. A basic method can perform efficiently and as well an advanced method without overcomplicating the recommendation system.

2 Related Works

There are two main groups of recommendation systems, collaborative filtering methods and content-based methods. I plan to focus my research on a few basic collaborative filtering methods. Collaborative filtering (CF) uses user and item similarities to make recommendations while content-based methods (CB) use item and user features to make recommendations. For example, a content-based model would look at user age, gender and location to make recommendations while a collaborative filtering model would look at data like ratings, number of clicks or purchases from a website. CB methods perform well when faced with the cold start problem because these methods can look at a new users features to make recommendations and does not need to analyze a user's previous interactions[4]. There are a few different CB methods; Term frequency-inverse document frequency(TF-ID) which recommend based the frequency of important features, cosine similarity, and K-nearest neighbors which can also be used in CF methods.

CF methods can be further differentiated into two different types, memory-based and model-based. Memory-based CF techniques, sometimes called neighborhood-based techniques, make recommendations based on similar users or similar items using a similarity measure like cosine similarity or pearson correlation similarity. In my research I used cosine similarity which measures the angle between two vectors. In the context of recommendation systems, the vectors either represent users and their dimension represents the number of items or conversely the vectors represent items and the dimension is the number of users. Pearson correlation similarity, alternatively requires that to compare the items they must both be rated by the users[7]. I chose to implement cosine similarity instead because the methods I focused on involved a linear algebra approach to recommendations and I found the visualization of cosine similarity to be more interesting. I also used K-nearest neighbors model coupled with the similarity measure to apply the prediction which will be discussed more in the methods section of this text.

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

Model-based CF involves more machine learning techniques compared to memory-based models. Some methods include graph based models using neural networks, Bayesian personalized ranking, and matrix factorization. I chose to research matrix factorization because I found it interesting that this simple method was able to extract and apply hidden user biases and preferences in an efficient way.

3 Resources

For my recommender system I used the ml-latest-small dataset from MovieLens[3] which is a very popular dataset for recommendation system research. Google Colab allows virtual access to a default CPU (Intel Xeon CPU), 13GB of RAM and a default GPU (NVIDIA Tesla K80) with 12GB of VRAM.

All code can be found in this Google Colab Notebook here
The corresponding data can be found here

4 Methods

4.1 Naive Model: Average-Based Predictions

For my exploration into CF, I wanted to see how a simple, naive recommendation system would perform against more advanced methods. For my baseline method I used an item-based approach and a user-based approach. The item-based method assumes that the average rating of a movie is a good prediction of how other users will rate it. In the item-based approach the average rating is found for each movie. Then all users who have not yet rated a movie will be predicted to rate that movie with its average rating. For the user-based approach the average rating for each user is determined. Then, for each user the predicted rating for all unrated movies is that users average rating. These techniques are very fast and simple to implement because they simply fill in all missing data values with a calculated average. These methods are purposefully naive in that it assumes a user will rate future items in a similar way or it assumes an item's future ratings will be similar to its previous ratings.

4.2 User-Based and Item-Based Collaborative Filtering

The next CF method that I researched was User-based and Item-based methods. User-based CF is a technique that predicts a user's future rating of an item based on a similar user's ratings of that item. Similarly, Item-based methods use pairs of similar items to determine a user's missing ratings[5]. Both methods use a similarity function to determine similar users and similar items. I chose to use cosine similarity because it seemed to be a commonly used similarity function for this problem.

$$cos(\theta) = \frac{A \cdot B}{||A||||B||}$$

Cosine similarity determines similarity between users by creating a vector of each user with each dimension corresponding to a movie. The cosine similarity has a range of [-1,1] where -1 denotes directly opposite users, 1 denotes users that are exactly the same, and 0 denotes orthogonal users meaning they are not correlated[8]. Using a K-nearest-neighbor model then determines the top k similar users and from there, the cosine similarity matrix of all ratings can be made. The item-based method is very similar to the user-based but instead the cosine similarity is found from item vectors where each dimension represents a user. Both methods use a K-nearest neighbor approach to predicting user ratings. Each similar rating is weighted by the similarity score and then divided by the sum of all similarity scores[7]. To implement these methods I used the Surprise method KNNBasic with the similarity measure configured to cosine similarity. This function fits the training data by computing the similarity scores between users or items depending on if KNNBasic is set to user-based or not. To make the predictions based on the model, the function finds the k-nearest neighbors for each user-item pair and takes a weighted average to predict the unknown rating.

4.3 Matrix Factorization

Matrix factorization is a model based method of collaborative filtering. Matrix factorization works very well with recommendation tasks because it is able to perform well with sparse data. Also, this is the method that won the Netflix prize for producing optimal recommendations and outperforming neighbor-based methods[4].This method reduces the representation of users and items to the dimensions of assumed latent features[1]. This allows for capturing certain trends and hidden features of the data that cannot be seen through solely visualizing the data. Matrix factorization is one solution to quantifying latent features of the user-item interactions which in this case are the ratings from each user. In matrix factorization the user-item interaction matrix R is decomposed into two matrices P and Q, the user and item matrices respectively.

$$R_{mxn} \approx P_{mxK} \cdot Q_{Kxn}$$

K is representative of the number of latent features within matrix R, m is the number of users and n is the number of movies. Each row of matrix P represents the features of a user and each column of Q represents the feature vector of a movie. I took two different approaches to factor the rating matrix into P and Q.

4.3.1 Standard Value Decomposition

Singular Value Decomposition is a technique used to identify latent factors in data so I knew that it had potential to work well in a collaborative filtering model. SVD decomposes a mxn matrix into three matrices U, Σ, and V. The first (k?) columns of U represent the column space of the matrix and the first (k?) columns of V represent the row space of the matrix. Because the rating matrix is highly sparse ($\%98.3$) SVD cannot accurately make predictions and the solution UV^T will be very close to 0[2]. To fix this issue I implemented stochastic gradient descent to minimize the error function.

$$min||R - PQ^T||^2$$

I formulated this by constructing the matrices P and Q from finding the SVD of the training set rating matrix R with an optimal k value of 30.

$$P = U\Sigma$$

$$Q = \Sigma V$$

Then to minimize the error and in-turn, fit the training set to the model, I used a stochastic gradient descent algorithm that cyclically updates P and Q until convergence or until a tolerance was met.

4.3.2 Stochastic gradient descent

The training matrix R is not equivalent to the dot product of user and item matrices P and Q. To minimize the error between them implemented stochastic gradient descent. The algorithm iterates until the error between R and PQ^T is below a specified tolerance or convergence. The error is calculated for each rating within R.

$$e_{ui} = r_{ui} - p_u q_i^T$$

Each value in P and Q are updated with the gradient descent of the error function with step size α and regularization parameter β. The step size and regularization parameter are incredibly important to the performance of the model and must be properly chosen for optimal results which I discuss in a later section. The regularization parameter β is included to prevent overfitting the training data.

$$e_{ui} = r_{ui} - p_u q_i^T + \beta(||p_u||^2 + ||q_i||^2)$$

$$q_i \leftarrow q_i + \alpha(2e_{ui}p_u - 2\beta q_i)$$

$$p_u \leftarrow p_u + \alpha(2e_{ui}q_i - 2\beta p_u)[6]$$

I used this algorithm on the SVD initialized user and item matrices and created the final model by multiplying the user and item matrices after the error for each rating has been minimized. To compare

the performance I also ran the algorithm on a user and item matrices that were initialized uniformly at random.

Matrix factorization is an optimal solution for sparse matrices because it is able to reduce the dimensions of a large utility matrix, or in this case, rating matrix. As shown, the matrix R is not equivalent to the dot product of the user and item matrices so this matrix factorization problem becomes an optimization problem. One way to optimize this function is to minimize the squared error using a stochastic gradient descent algorithm. The algorithm updates the matrices P and Q using the computed prediction error, learning rate α and regularization parameter β until convergence or the number of maximum steps is reached.

4.4 Evaluation

For my evaluation of the different collaborative filtering models I chose to use root mean square error as it was a popular evaluation metric for collaborative filtering methods.

$$RMSE = \sqrt{\frac{1}{n}\sum(p_{ui} - r_{ui})^2}$$

Here P represents the predicted user-interaction matrix of ratings, n is the number of all ratings and R is the training set of the user-interaction matrix. A lower RMSE is indicative of a higher performing model because it means the difference between the predictions and the actual ratings is small.

5 Experimental results

To begin working with the MovieLens dataset I needed to see the distribution of the ratings and how sparse the data is since sparsity can play an important role in the model's performance.

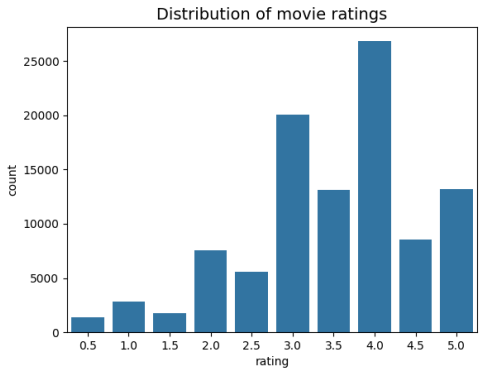

Figure 1: Distribution of Movie Ratings

As seen in figure 1, there are more ratings in the range [3,4]. Before applying any CF methods the data must be centered and normalized in order to get the best an most accurate results. Another step that needed to be completed before training the data was hyper-parameter tuning for the stochastic gradient descent algorithm (SGD). To do so, I created a function that kept track of the best alpha and k value. These values were found by calling the SGD algorithm with the specified learning rate and k latent features and computing the RMSE of the predicted user-interaction matrix from its result. I used a list of decreasing learning rates and a list of increasing k values to determine the optimal values.

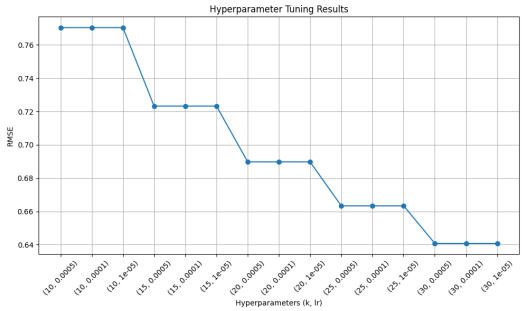

Figure 2: Graph of RMSE, Learning Rate, and Latent Features

I did expect the RMSE values to increase as the learning rate decreased and the k values increased to show what parameters would cause overfitting but evidently that did not happen. I knew that choosing a high k value increased the risk of overfitting the data because k is representative of the number of latent features for the user and item matrices. Using a large k value can introduce noise and patterns in the training data which can interfere with relevant patterns that I am trying to capture. I did notice however that at the k values 25-30 the RMSE started to decrease at a lower rate so I decided that a k value of 30 and a learning rate value of $1e-05$ were the best values for my gradient descent algorithm.

5.1 Training the Data

After these preliminary steps were completed, I could split the data into training and test sets to be fit to the various methods. I used an 80-20 split for the training and test sets. I then normalized the training and test sets by subtracting the mean of each column and dividing by the standard deviation. To prevent dividing by 0, I set the standard deviation to 0.1 in all cases where it was 0.

Then I was able to fit the training set to the different methods starting with the Naive models. Because these models performed very simple calculations and applied them uniformly to the training data, they ran very fast. The user-based and item-based CF models ran a bit slower as expected. For these methods I did have to use a different training set because to use the Surprise functions it was necessary to use the train_test_split method from their library rather than the method from sklearn. Because of this I understood that the evaluation of these two functions may not match the other functions as well. In advance, I predicted that these two functions would perform similarly but would not perform as well as the matrix factorization model.

For matrix factorization model I fit the training data to two different user and item matrices. For the first set I initialized the user and item matrices with SVD with a k value of 30, computing them with the dot product of $U\Sigma$ and ΣV respectively. The final prediction matrix model was then created with dot product of the returned user and item matrices from the SGD algorithm. Similarly I computed the prediction matrix model with user and item matrices that were initialized uniformly at random with a k value of 30. Fitting the training data with the matrix factorization model took much longer than the previous models because the function was much more complex and had to minimize the error between the training set and the predicted user-interaction matrix of ratings.

5.2 Evaluation

To evaluate the models I used RMSE as before mentioned.

Method	RMSE
Naive	
Item-avg Prediction	0.881255
User-avg Prediction	0.949746
Memory Based: Cosine Similarity	
User-user	0.971401
Item-item	0.971255
Model Based: Matrix Factorization	
SVD Initialized	0.784836
Random Initialized	1.021532

Table 1: RMSE of Collaborative Filtering Techniques

I found it interesting that the naive methods outperformed the memory based methods. Although they use a different training and test set, the item-average prediction method outperforms both cosine similarity methods by a noticeable amount (approximately 0.09). This leads me to believe that sometimes a simple and naive method can still perform well depending on the dataset and the distribution of data. Overall, the model-based matrix factorization method coupled with SVD outperformed all methods. This result makes sense because SVD is able to capture latent features and these features were preserved and optimized in the SGD algorithm. The result of the randomly initialized user and item matrices was puzzling because I predicted this to have the lowest RMSE score of all the methods. I believe that the data was overfit in the SGD algorithm with this method because when performing hyper-parameter tuning I used the SVD initialized user and item matrices.

6 Conclusion

This research confirms previously reported results on matrix factorization and its optimal performance on sparse data. It also further proves that machine learning, model based, collaborative filtering techniques outperform memory based techniques. In addition, including SVD in matrix factorization model performs better and fits the training data faster because the user and item matrices constructed from SVD contain some of the latent patterns of the training data. I do believe that the random initialized user and item matrices would perform as well as or potentially outperform the SVD matrix factorization method if I had correctly executed hyper-parameter tuning for that method. In consequence, the results for that method can not be applied to any future research. Aside from that, my research further proves that simple collaborative filtering techniques are still useful for recommendation tasks and can perform well against previously stated issues like the data sparsity and the cold start problem.

References

[1] Amir Alipour Yengejeh. A recommender system for movie ratings with matrix factorization algorithm, 05 2023.

[2] Google Developers. Collaborative Filtering with Matrix Factorization. `https://developers.google.com/machine-learning/recommendation/collaborative/matrix`, Accessed 2024.

[3] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), dec 2015.

[4] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[5] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[6] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender Systems Handbook*, pages 73–105. Springer, 2011.

[7] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295, 2001.

[8] Wikipedia contributors. Cosine similarity — Wikipedia, the free encyclopedia, 2024. [Online; accessed 30-April-2024].