Recommendation System for Amazon Products Using Collaborative Filtering

CS 480 Project Report

Shagufta Anjum - 19XJ1A0568
Rayid Ali - 18XJ1A0530

Under the supervision of
Dr. Yayati Gupta

(June 2021)

Contents

Abstract

With the growth of e-commerce websites, the need for efficient recommendation systems to help users discover items they might like is ever increasing. In this paper, we attempt to understand the various techniques and algorithms used to model real-life recommendation systems. We then present a simple recommendation engine to make recommendations for Amazon products using collaborative filtering (CF) techniques. Given a set of users and their reviews of Amazon products, our system generates a ranked list of the top k products to recommend to individual users based on the preferences of similar users. We have created two such systems: one that uses memory-based user-item CF, and one that relies on model-based CF using singular value decomposition techniques.

1. Introduction

Recommendation systems have revolutionized the way users interact with e-commerce websites. Almost every company is attempting to harness the power of recommendation systems to improve their user engagement by providing personalized choices and consequently boost sales. These systems have attained great results in solving the problem of ‘too many choices’ caused by the rapid increase of digital information. There is an incredibly large number of products that are listed today on e-commerce websites like eBay, Flipkart and Amazon. Recommender systems are able to filter through this large amount of data based on a user’s historical interests and preferences, and then make predictions about what items a user would prefer to buy.

There is a great variety of techniques that are used to build recommendation engines and abundant research has been done on this topic. In this project, we attempt to understand these techniques in detail and build our own recommendation engine using collaborative filtering. The rest of the paper is structured as follows. We first present the past work we reviewed, followed by a description of the dataset and preprocessing methodology. We then present the techniques used to build the recommendation system, followed by our results. Lastly, the challenges we encountered, and possible future work.

2. Literature Review

The idea of using a bipartite graph in recommendations has been around for a while. We used *Applying Link Prediction to Recommendation Systems for Amazon Products, Evan Darke, Zhouheng Zhuang, and Ziyue Wang, CS224W Project Final Report* [1] as initial inspiration. They used Link Prediction to get a ranked list of the top products likely to be co-purchased with an input node q. We wanted to build a similar recommendation system using various approaches.

Evaluating Prediction Accuracy for Collaborative Filtering Algorithms in Recommender Systems, Safir Najafi, Ziad Salam [15] attempts to understand which collaborative filtering algorithms made optimal predictions on the MovieLens dataset. The report looks in detail at various recommendation algorithm approaches and the general problems associated with these systems. In addition to proposing some interesting content, collaborative and hybrid approaches, it also gave us an insight into appropriate evaluation metrics.

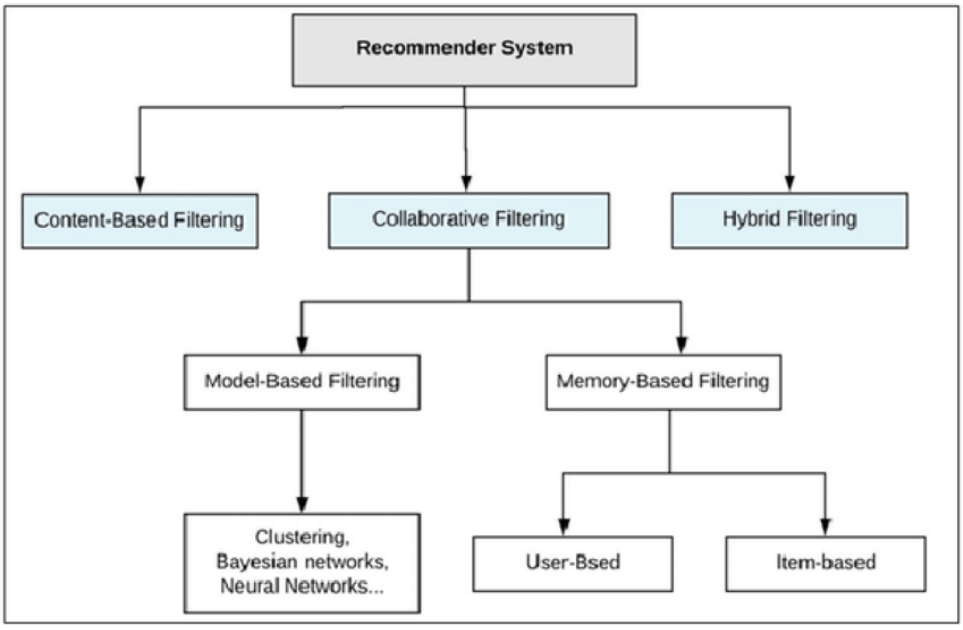
Collaborative Filtering Recommender Systems, Rahul Makhijani, Saleh Samaneh, Megh Mehta [16], deals with a problem similar to ours to build a user-restaurant prediction model. They implement some modified versions of the algorithms we use to enhance the results.

The paper *Recommendation System Models in Product Rating Predictions, Xiaoye Liu, CS224W Project report [17]* evaluates the rate-prediction performance of Collaborative Filtering for user-based, item-based, and matrix Factorization strategies, and graph-based Network Inference model. They use an additional Graph-based Network Inference model that is quite different from CF.

Additionally, we looked into various other papers (mentioned in the references section) which used community detection along with link prediction to fully understand how others have used graphs and algorithms to create different approaches to a recommendation engine. After studying various approaches, techniques and models used, we decided to use a collaborative filtering approach. Since we used a real-world dataset, it is quite sparse. We concluded that collaborative filtering tools would work best with our data.

Finally, we found many great articles [11], [12], [18] on sites such as towardsdatascience.com and medium.com that we used to learn the basics about the types of recommendation systems and their applications in the real world.

3. Background: Recommendation Systems



3.1 Content-Based Methods

Content-based techniques use extra information about users and items.
The concept of content-based strategies is to attempt to construct a model, primarily based totally on the available "features", that specify the discovered user-item interactions. If we manage to get any such model, then making new predictions for a user is quite easy: we simply need to study the profile (age, sex, etc.) of this user and decide applicable items primarily based totally on this information to suggest.

Content-based strategies suffer far much less from the cold start trouble than collaborative techniques: new users or items may be defined via way of means of their characteristics (content) and so applicable suggestions may be performed for those new entities.

3.1.1 User-based

In order to make a brand new recommendation to a user, the user-based approach more or less attempts to discover users with the maximum similar "interactions profile" (nearest neighbours) so as to indicate items which might be the most famous amongst those neighbours (and which

might be "new" to our user). This approach is stated to be "user-centred" because it represents users primarily based totally on their interactions with items and evaluates distances among users.
Assume that we need to make a recommendation for a given user. First, each user may be represented with the aid of using its vector of interactions with the specific items ("its line" withinside the interaction matrix). Then, we are able to compute a few types of "similarities" among our user of interest and each different user. That similarity degree is such that users with comparable interactions at the identical items ought to be taken into consideration as being close. Once similarities to each consumer had been computed, we are able to hold the k-nearest neighbours to our user after which recommend the most famous items amongst them (searching on the items that our reference user has not interacted with yet).

3.1.2 Item-based

To make a new recommendation to a user, the concept of the item-based technique is to locate items much like those the user already "positively" interacted with. Two items are taken into consideration to be comparable if the maximum of the users who have interacted with each of them did it in a similar way. This technique is stated to be "item-centred" because it constitutes items based on interactions users had with them and examines distances among the one's items.
Assume that we need to make a recommendation for a given person. First, we bear in mind the item this user appreciated the most and represent it (as all of the different items) through its vector of interaction with each user ("its column" withinside the interplay matrix). Then, we will compute similarities among the "best item" and all of the different items. Once the similarities were computed, we will then maintain the k-nearest-neighbours to the selected "best item" which can be new to our person of interest and suggest those items

3.2 Collaborative Filtering Methods

Collaborative techniques for recommender systems are techniques that are based entirely on the past interactions recorded among users and items so one can produce new suggestions. These interactions are saved withinside the so-referred to as "user-item interactions matrix".
Then, the primary concept that guidelines collaborative methods are that those past user-item interactions are enough to detect comparable users and items and make predictions based on those predicted proximities.
The class of collaborative filtering algorithms is split into sub-classes usually referred to as memory-based and model-based strategies. Memory-based strategies directly work with values of recorded interactions, assuming no model, and are basically based on nearest neighbours search (for example, discover the nearest users from a user of interest and propose the maximum famous items amongst those neighbours). Model-based strategies anticipate an underlying

“generative” model that explains the user-item interactions and attempts to find out it so one can make new predictions.
The fundamental benefit of collaborative strategies is they require no information about users or items and, so, they may be used in lots of situations. Moreover, the more users have interacted with items the more new suggestions turn out to be accurate: for a set of users and items, new interactions recorded over the years deliver new information and make the machine increasingly effective.
However, because it only considers past interactions to make suggestions, collaborative filtering suffers from the “cold start problem”: it's far not possible to advise something to new users or to advise a brand new item to any users and plenty of users or items have too few interactions to be successful.

3.2.1 Memory-Based CF

In memory-based collaborative techniques, no latent model is assumed. The algorithms work with the user-item interactions: for example, users are represented through their interactions with items and the nearest neighbours search on those representations is used to provide suggestions.

3.2.2 Model-Based CF

In model-based collaborative techniques, a few latent interaction models are assumed. The model is trained to reconstruct user-item interaction values from its very own illustration of users and items. New recommendations can then be accomplished based on this model. The users and items latent representations extracted with the aid of using the model have a mathematical meaning that may be difficult to interpret for a human being.

3.3 Hybrid Methods

A hybrid model attempts to combine functionalities from both content-based and collaborative filtering models in an attempt to beat the “cold start” and data sparsity problems and to utilise the best of both worlds. A hybrid approach can be done in many ways - by implementing content-based and collaborative-based techniques separately and combining them; by adding certain content-based techniques to a collaborative-based approach or vice versa; or by combining the two in a single model.

4. Dataset Analysis and Preprocessing

4.1 Raw Dataset

We used the Amazon Review Data (2018) aggregated by Jianmo Ni from UCSD (https://nijianmo.github.io/amazon/index.html). This dataset contains product reviews and metadata from Amazon, including 233.1 million reviews. It includes reviews (ratings, text, helpfulness votes), product metadata (descriptions, category information, price, brand, and image features), and links (also viewed/also bought graphs).

We used the Electronics reviews dataset, which has 20,994,353 reviews.

Each field (user-product edge) has the following information:

- reviewerID - ID of the reviewer, e.g. A2SUAM1J3GNN3B
- asin - Amazon Standard Identification Number, a unique identifier for each product, e.g. 0000013714
- reviewerName - name of the reviewer
- vote - helpful votes of the review
- style - a description of the product metadata, e.g., "Format" is "Hardcover"
- reviewText - text of the review
- overall - rating of the product
- summary - summary of the review
- unixReviewTime - time of the review (unix time)
- reviewTime - time of the review (raw)
- image - images that users post after they have received the product.

The relevant fields to our work are the User ID (reviewerID), Product ID (asin) and the rating given by the user to that product (overall). These were renamed to userId, productId and Rating respectively.

4.2 Network Structure

For the initial understanding of the structure of our data, we represent it as a graph in Figure 1 with a node for every user and a node for every product. There is an edge between users and the products purchased by them. This is structured as a bipartite graph with edges only between users and products.

Figure 1

4.3 Analysis of Data

In our initial dataset, we have 78,24,481 rows, each corresponding to one user review. It has 4201696 unique users and 476001 unique products. Due to a lack of computing power for the nature of our calculations, we choose a random subset of 10,00,000 reviews to work with. There were no missing rating fields in these reviews.

To check for a disparity in the popularity of products, we plot productID vs No.of Ratings (Figure 2). We notice that a couple of products have an insanely high number of ratings, while most products have very few.

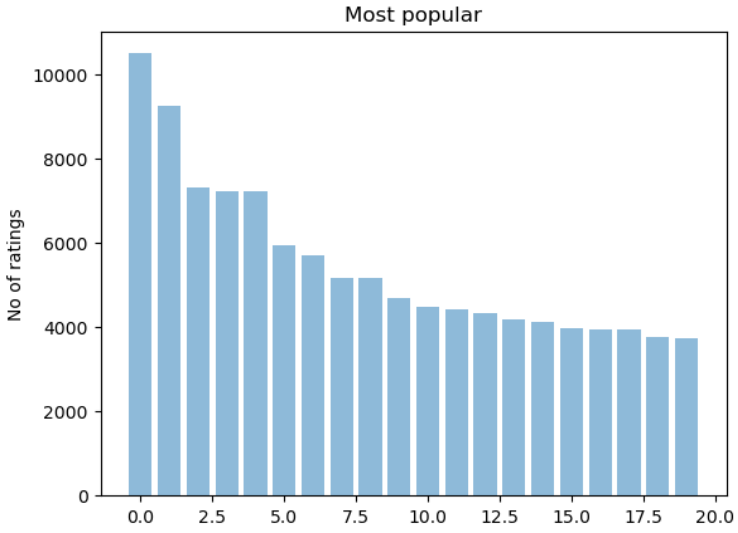


Figure 2

To further explore this, we look at a sorted list of products and the number of ratings they received. Most products were rated only once, and the average rating per product is 15.23. Similarly, we also look at the no. of ratings given by each user. The variation is even higher in this case, with very few users rating more than one product. The average ratings per user are only 1.15. The data in this state is very sparse, and users and products with fewer links will not contribute much to the recommendation system. To make it denser, we take a subset of the data by removing the less significant users and products. To choose a minimum threshold, we plot the distribution of user ratings (Figure 3) and distribution of product ratings (Figure 4).

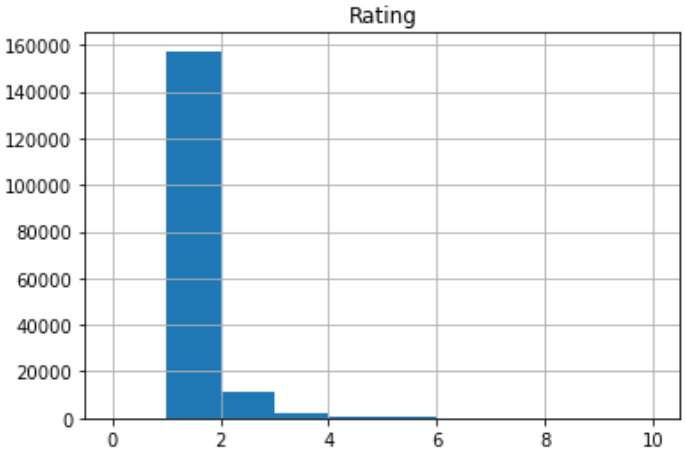


Figure 3

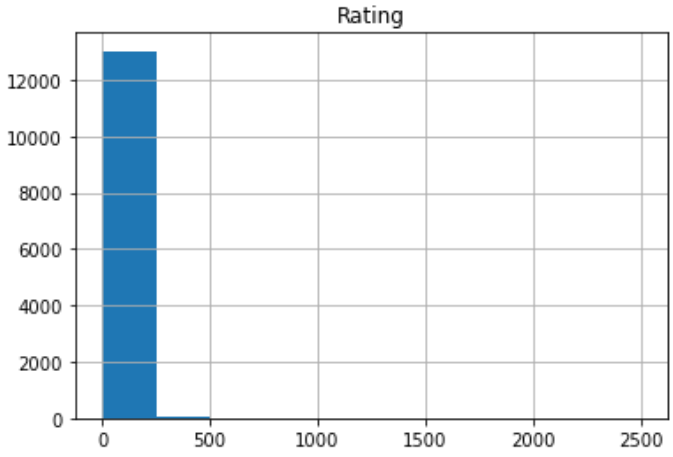


Figure 4

Finally, we analyze the ratings. The ratings all lie in the range from 1 to 5. The mean of the ratings is about 4.01389, showing that most ratings are on the higher side. In fact, more than half of the ratings are 5. The distribution of the ratings can be seen in Figure 5.

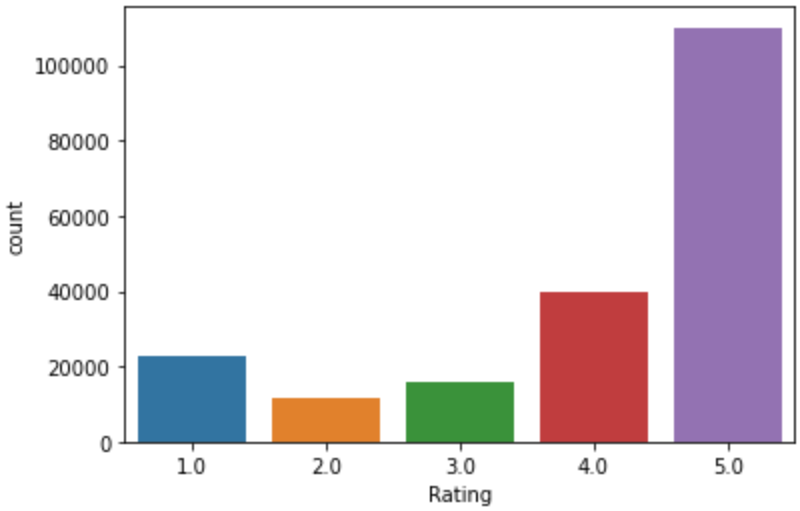


Figure 5

4.4 Data Preprocessing

Based on the distribution of user and product ratings discussed earlier, we decide to remove all products that have less than 10 ratings and all users who have rated less than 5 products. This filtered data has much fewer rows. The no. of unique users is 12676 and the no. of unique products is 26082. This subset is slightly denser than the initial data, which might make it more suitable for making recommendations.

The data is split randomly into two parts, the training set (70%) and the test set (30%). The training set has 69228 reviews and the test set has 29670 reviews. This is the final data that we work with. We implement the prediction algorithms on our training set to predict ratings given by a user to a product-based both memory and model-based collaborative filtering. The results are analyzed using the test dataset.

5. Approach

We chose the collaborative filtering (CF) approach to build our recommendation system. This is the most widely used approach to build recommendation systems and has been successfully employed in multiple applications. A lot of research has been done on different collaborative filtering models. We implement and compare both variations of this approach, the memory-based and model based-techniques. The key difference in the memory-based approach from model-based techniques is that the system does not learn any new parameters using training or optimization algorithms, making it easier to use. The closest user or items are calculated by using metrics such as Cosine similarity or Pearson correlation coefficients, which are only based on arithmetic operations. However, its performance decreases when we have sparse data, which is common in most real-world problems, and in our own Amazon dataset as well. This hinders the scalability of this approach for real-world applications.

In the model-based approach, using the given users and their ratings of certain products, machine learning algorithms are used to predict the ratings the users would give to products they have not rated by assuming a latent model. Model-based collaborative filtering can be further broken down into types based on the algorithms used. One such CF technique is Matrix Factorization based algorithms. The issue of the sparsity of the rating matrix is well taken care of by the Matrix Factorization method.

5.1 The Memory-Based Approach

There are many similarity measures that can be used to compute the similarity between two vectors. Some of them are cosine similarity, dot product, euclidean distance, manhattan distance and Pearson similarity. We choose cosine similarity as it is the most commonly used in recommender systems.

5.1.1 Cosine Similarity

Cosine Similarity is a metric that quantifies the similarity among two vectors. The cosine similarity is the cosine of the angle among vectors. The vectors are commonly non-zero and are

inside an inner product space. Cosine similarity is defined mathematically as the division among the dot product of vectors and the product of the euclidean norms or magnitude of every vector.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

For our dataset, if ratings are represented by r, the function will look like

$$similarity(i,j) = \frac{\sum_u^U r_{(u,i)} r_{(u,j)}}{\sqrt{\sum_u^U r_{(u,i)}^2}\sqrt{\sum_u^U r_{(u,j)}^2}}$$

To generate the actual predictions, the similarity matrix needs to be combined with the past history of products rated by the users. This is done by applying the following formula:

$$score(u,i) = \frac{\sum_j^I similarity(i,j)(r_{(u,j)} - \bar{r_j})}{\sum_j^I similarity(i,j)} + \bar{r_i}$$

Here, score(u, i) represents the predicted rating for the user-item pair.

For user-based collaborative filtering, we predict that a user's rating for a particular product is the weighted sum of all other users' ratings for that product. The weights will be the cosine similarity between the current user and all other users. The score needs to be normalized to make sure it lies between 1 and 5. Lastly, we attempt to remove biases associated with users. This bias may result when some users tend to always give a high or low rating to products. This is done by subtracting every user's average rating while summing over similar user's ratings and then add that average later. This will generate a matrix of predicted ratings for users and products, and the highest-rated products can be recommended to users.

5.2 The Model-Based Approach

5.2.1 Matrix Factorization

Matrix factorization algorithms decompose the massive and sparse user-item interaction matrix right into a product of smaller and denser matrices: a user-factor matrix (representing users) improved with the aid of using a factor-item matrix (representing items). The foremost assumption behind matrix factorization is that there exists a quite low dimensional latent space of features wherein we are able to constitute each user and product such that the interaction among them may be received through computing the dot product of corresponding dense vectors in that space. However, we don't need to explicitly present those features to our model. Instead, we favour allowing the system to find out those beneficial features by itself and make its very own representations of each user and item. These extracted features taken individually have a mathematical meaning however no intuitive interpretation. But often, those features rising from matrix factorization are extraordinarily near the intuitive decomposition that we as human beings can imagine.

We initially had a set U of users and a set D of items. Matrix R of length |U| |D| is the matrix that we pointed out in advance that includes all of the ratings that the users have assigned to the items. Our intention is to extract K latent features from this matrix. Using matrix factorization, the matrix R is broken up into matrices:
1. Matrix P (of size |U| K)
2. Matrix Q (of size K |D|)
such that the product of P and Q equals R cap, which is an approximation to R.

$$\mathbf{R} \approx \mathbf{P} \times \mathbf{Q}^T = \hat{\mathbf{R}}$$

In this manner, every row might constitute the power of the institutions among a user and the features. Similarly, every row of $\mathbf{Q}$ might constitute the power of the institutions among an item and the features. To get the prediction of a rating of an item d_j by u_i, we are able to calculate the dot product of the 2 vectors corresponding to and :

$$\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{k} p_{ik} q_{kj}$$

Now, we must discover a manner to obtain $\mathbf{P}$ and $\mathbf{Q}$. One way to approach this trouble is the first initialize the 2 matrices with a few values, calculate how `different' their product is to $\mathbf{M}$, after which attempt to minimize this distinction iteratively. Such a technique is known as gradient descent, aiming at locating a nearby minimum of the distinction.

The distinction here, commonly known as the mistake among the expected score and the actual score, may be calculated with the aid of using the subsequent equation for every user-item pair:

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2$$

Here we keep in mind the squared error due to the fact the expected rating may be both better or lower than the actual rating.
To limit the error, we must understand in which course we must alter the values of p_{ik} and q_{kj}. In other words, we want to understand the gradient on the current values, and consequently, we differentiate the above equation with recognize those variables separately:

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(q_{kj}) = -2e_{ij} q_{kj}$$

$$\frac{\partial}{\partial q_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(p_{ik}) = -2e_{ij} p_{ik}$$

Having received the gradient, we are able to now formulate the update rules for each p_{ik} and q_{kj}:

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij} q_{kj}$$
$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij} p_{ik}$$

Here, α is a constant whose value determines the rate of approaching the minimum. Usually, a small value is selected for α, say 0.0002. This is due to the fact that if we make too massive a step toward the minimum we can also additionally run into the chance of missing the minimum and become oscillating across the minimum.

A query may have come to your thoughts with the aid of using now: if we discover matrices $\mathbf{P}$ and $\mathbf{Q}$ such that $\mathbf{P} \times \mathbf{Q}$ approximates $\mathbf{R}$, isn't that our predictions of all of the unseen ratings will all be zeros? In fact, we aren't simply seeking to come up with $\mathbf{P}$ and $\mathbf{Q}$ such that we are able to reproduce $\mathbf{R}$ exactly. Instead, we can best attempt to limit the errors of the discovered user-item pairs. In other words, if we allow T being a set of tuples, each of which is within the form of (u_i, d_j, r_{ij}), such that T includes all of the observed user-item pairs

collectively with the related scores, we're only seeking to limit each e_{ij} for $(u_i, d_j, r_{ij}) \in T$ (In different words, T is our set of training data.)

As for the rest of the unknowns, we can be capable of deciding their values as soon as the associations among the users, items, and features are learned.
Using the above update regulations, we are able to then iteratively carry out the operation till the error converges to its minimum. We can test the general error as calculated by the usage of the subsequent equation and decide when we have to forestall the process.

$$E = \sum_{(u_i,d_j,r_{ij})\in T} e_{ij} = \sum_{(u_i,d_j,r_{ij})\in T} (r_{ij} - \sum_{k=1}^{K} p_{ik}q_{kj})^2$$

Regularization:
The above algorithm is a completely fundamental algorithm for factorizing a matrix. There are a number of techniques to make matters look complicated. A common extension to this fundamental algorithm is to introduce regularization to keep away from overfitting. This is achieved with the aid of using including a β parameter and alter the squared error as follows:

$$e_{ij}^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik}q_{kj})^2 + \frac{\beta}{2}\sum_{k=1}^{K}(||P||^2 + ||Q||^2)$$

In different words, the new parameter β is used to govern the magnitudes of the user-feature and item-feature vectors such that P and Q would deliver a very good approximation of R while not having to include massive numbers. In practice, β is set to a few values withinside the range of 0.02. The new update regulations for this squared error may be received with the aid of using a system much like the only one defined above. The new update rules are as follows.

$$p'_{ik} = p_{ik} + \alpha\frac{\partial}{\partial p_{ik}}e_{ij}^2 = p_{ik} + \alpha(2e_{ij}q_{kj} - \beta p_{ik})$$
$$q'_{kj} = q_{kj} + \alpha\frac{\partial}{\partial q_{kj}}e_{ij}^2 = q_{kj} + \alpha(2e_{ij}p_{ik} - \beta q_{kj})$$

5.2.2 Singular Value Decomposition (SVD)

The Singular Value Decomposition (SVD), a technique from linear algebra that has been normally used as a dimensionality reduction method in machine learning. SVD is a matrix

factorization method, which reduces the number of features of a dataset with the aid of decreasing the space dimension from N-size to K-size. In the context of the recommender system, the SVD is used as a collaborative filtering method. It makes use of a matrix shape wherein every row represents a user, and every column represents an item. The factors of this matrix are the rankings that are given to items with the aid of using users.

$$A = USV^T$$

The factorization of this matrix is achieved with the aid of using the singular value decomposition. It reveals factors of matrices from the factorization of a high-level (user-item-rating) matrix.

The singular value decomposition is a way of decomposing a matrix into 3 different matrices as given below: Where A is a m x n software matrix, U is a m x r orthogonal left singular matrix, which represents the connection among users and latent factors, S is a r x r diagonal matrix, which describes the strength of every latent factor and V is a r x n diagonal right singular matrix, which suggests the similarity among items and latent factors. The latent factors right here are the traits of the objects, for example, the style of the music.

The SVD decreases the dimension of the utility matrix A by extracting its latent factors. It maps every user and every item right into an r-dimensional latent space. This mapping allows a clean illustration of relationships among users and items.

6. Implementation

We present the relevant data fields in the form of a pivot matrix, with the rows representing users and the columns representing products. The cells are filled with the ratings given by users to the products purchased by them. Since not all users buy all products, there are bound to be many missing values in the cells of this table.

For the memory-based approach, our metric of choice is cosine similarity. We use the from pairwise_distances function from sklearn and set the input metric as cosine similarity. This computes the similarity between users. We plug this into the formula discussed above to generate the user-product prediction matrix.

For the model-based approach, the null values in the pivot table are filled with 0 such that the filled values are provided for multiplication by the SVD algorithm. Based on the ratings that are already known, our SVD algorithm fills in the empty cells with the predicted ratings. Our final predictions matrix is populated with the predicted ratings for each user-product pair.

6.1 Making predictions using our models

After creating models for both approaches and obtaining the final predictions matrix, we can use it to predict the top k products to recommend to each user. Each row in the prediction corresponds to the predicted ratings for all products for a particular user. We create a function to recommend the top k products. The function takes the userId, the prediction matrix and the value of k as parameters. For a given user ID, the k highest ratings are chosen from that row and the products corresponding to those ratings are recommended in ranked order.

```
Below are the recommended items for user(user_id = 20):

                  user_ratings  user_predictions
Recommended Items
B00004ZCJE                 0.0          0.136136
B00005ARK3                 0.0          0.106266
B00006HYKM                 0.0          0.098032
B00006B7HB                 0.0          0.094819
B00004VX3T                 0.0          0.089562
```

7. Evaluation

For the evaluation of our models, we choose one of the most popular error metrics used in the evaluation of recommender systems, the root mean squared error or RMSE. The root mean squared error is a prediction accuracy metric that represents how close the prediction ratings are to the true ratings. It calculates the square root of the mean value of the squares of the differences between the true and the predicted ratings. It uses squared deviations due to which larger errors are more amplified, and is good for use when very large errors are unwanted.

$$\text{RMSE} = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(d_i - \hat{d}_i)^2}$$

d_i is the actual rating
$\hat{d}_i$ is the predicted rating
n is the amount of ratings

7.1 Results

Model	RMSE
Content-Based using Cosine Similarity	4.33481
Collaborative Filtering using SVD	0.00121

We also compare the average predicted ratings with the average actual ratings for products.

productId	Avg_actual_ratings	Avg_predicted_ratings	item_index
0594481813	0.002812	0.000301	0
0972683275	0.014058	0.001461	1
1400501466	0.009372	0.000698	2
1400501520	0.004686	0.000044	3
1400501776	0.004686	0.000044	4

The average predicted ratings turned out to be much lower than the average actual ratings.

8. Conclusion

8.1 Problems faced

A major issue that we faced in the implementation was the sparsity of the user-item matrix. It is very likely in a real-world scenario for even the most active users to rate a tiny fraction of products in comparison to the total products in the dataset. Similarly, even the most popular products are rated by a very small number of users. This makes the computation of similarity between users challenging. Since we used a user-based approach, perhaps a higher product to user ratio would lead to better recommendations.

The other major issue faced by recommendation systems is the “cold start” problem. In our approach, we used collaborative filtering which makes predictions based on a user’s previous purchase history. In the case of a new user, there is no product purchase history available, making it a futile approach. Both of these problems could be partly solved by using a combination of content and collaborative approaches.

Another problem encountered was the large size of the dataset. We had access to 31 GB of Amazon review data, which we could not take advantage of because of a lack of access to machines capable of processing that amount of raw data. We used a dataset of about 8 million rows and had to use a small subset of 1 million rows for our actual implementation due to a lack of computational power.

Yet another problem arises due to user behaviour. A large fraction of users purchase products but do not rate them, and are hence not a part of our dataset. Some users may tend to only rate a particular type of items rather than all items they purchase. This bias could possibly skew the results.

8.2 Future Scope

- Using other metrics such as Pearson similarity and Euclidean distance in our content-based approach to compare their performances and choose the one with the best performance.
- Implementing other collaborative filtering algorithms: Clustering algorithms such as K-Nearest Neighbours and alternate Matrix Factorization Based algorithms such as Probabilistic MF and Non-negative MF.

- Designing a hybrid model to include a content-based approach along with collaborative filtering so that we can address some of the above-mentioned issues.
- Using different metrics to evaluate and compare performances such as Mean squared error (MAE), Mean Average Precision @K (MAP@K) etc.
- Using more computing power to work with a larger dataset for better and more accurate results.
- Including product purchase metadata like “also bought together” and “also viewed” to get a better understanding of the user preferences and improve our model accordingly.
- Extracting sentiment from user reviews to understand the user-product interaction better. It would be interesting to see how taking additional data into consideration will impact the results.

9. References

[1] Applying Link Prediction to Recommendation Systems for Amazon Products, Evan Darke, Zhouheng Zhuang, and Ziyue Wang, CS224W Project Final Report.

[2] Using community detection and link prediction to improve Amazon recommendations, Amit Garg, Senthilnathan Viswanathan, Shloka Desai, CS224W Project Final Report.

[3] Graph-Based Recommendations of Amazon Products, Aaron Effron, Kelly Shen, Ryan Mui.

[4] Link Prediction in evolving networks based on the popularity of nodes, Tong Wang, Ming-yang Zhoul, and Zhong-qian Fu.

[5] Prediction of Co-purchasing Products, Yilun Liu, Chunhao Wu, Xiaohui Tong.

[6] Product Recommendation System, Jianfeng Hu, Bo Zhang, CS224W Project Report.

[7] Recommendation System Models in Product Rating Predictions, Xiaoye Liu, CS224W Project.

[8] Studying Recommendation Algorithms by Graph Analysis, Batul J. Mirza, Benjamin J. Keller, Naren Ramakrishnan.

[9] Network-based recommendation: Using graph structure in user-product rating networks to generate product recommendations, David Cummings, Ningxuan (Jason) Wang.

[10] https://github.com/ajohannsdottir/Tutorials/blob/master/Implementing%20your%20own%20recommender%20systems%20in%20Python%20.ipynb

[11] https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada

[12] https://towardsdatascience.com/build-a-user-based-collaborative-filtering-recommendation-engine-for-anime-92d35921f304

[13] https://blog.cambridgespark.com/nowadays-recommender-systems-are-used-to-personalize-your-experience-on-the-web-telling-you-what-120f39b89c3c

[14] Dataset: https://nijianmo.github.io/amazon/index.html

[15] Evaluating Prediction Accuracy for Collaborative Filtering Algorithms in Recommender Systems, Safir Najafi, Ziad Salam

[16] Collaborative Filtering Recommender Systems -Rahul Makhijani, Saleh Samaneh, Megh Mehta

[17] Recommendation System Models in Product Rating Predictions, Xiaoye Liu, CS224W Project report

[18] https://medium.com/the-owl/recommender-systems-f62ad843f70c